Modernizing Markov Chains Monte Carlo for Scientific and Bayesian Modeling

Charles C. Margossian

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2022

Abstract

Modernizing Markov chains Monte Carlo for Scientific and Bayesian Modeling

Charles C. Margossian

The advent of probabilistic programming languages has galvanized scientists to write increasingly diverse models to analyze data. Probabilistic models use a joint distribution over observed and latent variables to describe at once elaborate scientific theories, non-trivial measurement procedures, information from previous studies, and more. To effectively deploy these models in a data analysis, we need inference procedures which are reliable, flexible, and fast. In a Bayesian analysis, inference boils down to estimating the expectation values and quantiles of the unnormalized posterior distribution. This estimation problem also arises in the study of non-Bayesian probabilistic models, a prominent example being the Ising model of Statistical Physics.

Markov chains Monte Carlo (MCMC) algorithms provide a general-purpose sampling method which can be used to construct sample estimators of moments and quantiles. Despite MCMC's compelling theory and empirical success, many models continue to frustrate MCMC, as well as other inference strategies, effectively limiting our ability to use these models in a data analysis. These challenges motivate new developments in MCMC. The term "modernize" in the title refers to the deployment of methods which have revolutionized Computational Statistics and Machine Learning in the past decade, including: (i) hardware accelerators to support massive parallelization, (ii) approximate inference based on tractable densities, (iii) high-performance automatic differentiation and (iv) continuous relaxations of discrete systems.

The growing availability of hardware accelerators such as GPUs has in the past years motivated

a general MCMC strategy, whereby we run many chains in parallel with a short sampling phase, rather than a few chains with a long sampling phase. Unfortunately existing convergence diagnostics are not designed for the "many short chains" regime. This is notably the case of the popular $\hat{R}$ statistic which claims convergence only if the effective sample size *per chain* is large. We present the nested $\hat{R}$, denoted $\text{n}\hat{R}$, a generalization of $\hat{R}$ which does not conflate short chains and poor mixing, and offers a useful diagnostic provided we run enough chains and meet certain initialization conditions. Combined with $\text{n}\hat{R}$, the short chain regime presents us with the opportunity to identify optimal lengths for the warmup and sampling phases, as well as the optimal number of chains; tuning parameters of MCMC which are otherwise chosen using heuristics or trial-and-error.

We next focus on semi-specialized algorithms for latent Gaussian models, arguably the most widely used of class of hierarchical models. It is well understood that MCMC often struggles with the geometry of the posterior distribution generated by these models. Using a Laplace approximation, we marginalize out the latent Gaussian variables and then integrate the remaining parameters with Hamiltonian Monte Carlo (HMC), a gradient-based MCMC. This approach combines MCMC and a distributional approximation, and offers a useful alternative to pure MCMC or pure approximation methods such as Variational Inference. We compare the three paradigms across a range of general linear models, which admit a sophisticated prior, i.e. a Gaussian process and a Horseshoe prior. To implement our scheme efficiently, we derive a novel automatic differentiation method called the *adjoint-differentiated Laplace approximation*. This differentiation algorithm propagates the minimal information needed to construct the gradient of the approximate marginal likelihood, and yields a scalable differentiation method that is orders of magnitude faster than state of the art differentiation for high-dimensional hyperparameters. We next discuss the application of our algorithm to models with an unconventional likelihood, going beyond the classical setting of general linear models. This necessitates a non-trivial generalization of the adjoint-differentiated Laplace approximation, which we implement using higher-order adjoint methods. The generalization works out to be both more general and more efficient. We apply the resulting method to an unconventional latent Gaussian model, identifying promising features and highlighting persistent

challenges.

The final chapter of this dissertation focuses on a specific but rich problem: the Ising model of Statistical Physics, and its generalization as the Potts and Spin Glass models. These models are challenging because they are discrete, precluding the immediate use of gradient-based algorithms, and exhibit multiple modes, notably at cold temperatures. We propose a new class of MCMC algorithms to draw samples from Potts models by augmenting the target space with a carefully constructed auxiliary Gaussian variable. In contrast to existing methods of a similar flavor, our algorithm can take advantage of the low-rank structure of the coupling matrix and scales linearly with the number of states in a Potts model. The method is applied to a broad range of coupling and temperature regimes and compared to several sampling methods, allowing us to paint a nuanced algorithmic landscape.

Table of Contents

[1]This section is based on the case study *Bayesian model of planetary motion: exploring ideas for a modeling workflow when dealing with ordinary differential equations and multimodality* (Margossian and Gelman 2020).

[2]This section is based on the conference poster, *Solving ODEs in a Bayesian context: challenges and opporunities*

presented at the 2021 Population Approach Group in Europe conference; see Margossian et al. 2021.

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I first thank my advisor Andrew Gelman who throughout the years has been a source of encouragement, thought-provoking questions, and irreverence. Andrew encouraged me to pursue a PhD degree in Statistics, instilling in me the conviction that as a statistician I would be able to uphold my commitment to the natural sciences. With his guidance I was able to work at the interface of rigorous statistics and scientific applications. Thank you for the many prolific brainstorming sessions and for always finding the time to meet.

Next, I thank the members of my dissertation committee. Aki Vehtari has been a steady source of enlightenment, giving me clarity in the face of many puzzling problems. His vision helped me think beyond the next paper and reason about how the technology we are developing will be used in the years to come. Sumit Mukherjee's class in Theoretical Statistics remains the most engaging, if overwhelming, course I have taken. I will fondly remember the time we spent at the blackboard thinking about Ising models and grappling with a literature scattered across Probability, Physics, and Machine Learning. During my final summer as a PhD student, I was fortunate to work with Matt Hoffman who instantly became a key collaborator; our work on convergence diagnostics is one of the major pillars of this thesis. Lastly I thank Dave Blei, whose class on Graphical Models deepened my understanding of Bayesian modeling. Being his student and later his Teaching Assistant has been a privilege.

Much of my experience as a researcher has been shaped by the Stan community, which brings together an international community of scholars across multiple fields. Within that community,

I am particularly grateful to Michael Betancourt. Over the years, Michael and I had numerous conversations during which he shared his unique insights on Bayesian modeling. Working with him allowed me to develop expertise orthogonal to traditional Statistics curriculums. I also learned a lot about scientific writing both by getting his detailed feedback on several manuscripts and reading his work. I also thank Bob Carpenter, who reviewed my first pull request for Stan back in 2016. Had it not been for his feedback, I wouldn't be the programmer I am today. Thank you, and all those who reviewed my code in the past years, for empowering me to write algorithms in a low level language. Dan Simpson, along with Aki, pushed me to work on the integrated Laplace approximation, despite my struggling with the subject and asking him to reexplain the concept time and time again. Your enthusiasm and trust helped me persist. Thank you Mitzi Morris, Steve Bronder, Lu Zhang, Philip Greengard, Julien Riou, Liza Semenova, Léo Grinsztajn, Shoshana Vasserman, Vianey Leos-Barajas, Ben Bales, Rok Češnovar, Raj Agrawal, Hyunji Moon, Sean Talts, Yuling Yao, Lauren Kennedy, Jonah Gabry, and Paul-Christian Bürkner. Amongst the Stan users, I am particularly grateful to my colleagues in Pharmacometrics. A heartfelt thank you to my fellow Torsten developers, Bill Gillespie and Yi Zhang and my colleagues at Metrum Research Group, as well as Sebastian Weber. Thank you to France Mentré, who introduced me to the field of Pharmacometrics, as well as her team at l'INSERM: Julie Bertrand, Jéremie Guedj, Aurélien Marc, and Marion Kerioui.

Next I am grateful to the Department of Statistics at Columbia University. Thank you to Bodhi Sen, director of graduate studies during my first year, for his high academic expectations and for giving me the support to meet those expectations. Thank you to Tian Zheng, the chair of the department, for all her work towards making the department a more welcoming place and for championing the role of Data Science within the department. To my cohort – Elliott Gordon-Rodriguez, Nick Galbraith, Yuanzhe "Manuel" Xu, Chi Wing "George" Chu, Reed Palmer, Naburun Deb, and Shun Xu – you are the best. As painful as the coursework was during the first semester, this was also the time in my life during which I laughed the most. Thank you for the banter, the dinners, the drinks, and Dan Brownian motion. And thank you to all the students who showed up

to happy hours, the student retreat, the soccer pitch, and the student lounge on the 10^{th} floor. A special mention to Andrew Davison who was always down for a quick chat and more than once helped me jumpstart a proof. A huge shoutout to Dood Kalicharan and Anthony Cruz who keep the department running. Thank you to my office mates at Aalto University and fellow roadtrippers, Alejandro Catalina, Kunal Gosh, and Federico Pavone. Thank you to the Bayesflow team at Google Research, especially my co-host Pavel Sountsov, and also Sharad Vikram, Christopher Sutter, Alexey Radul, and Rif Santos.

Outside the academic circle, I am grateful to my parents, my brother, my family, my significant other, and my friends – notably the infamous Order of the Seaweed – for believing in me, listening to my thoughts and sharing theirs, and for the admirable melding of spiritual comments and outrageous ideas during our freeflowing conversations. You make my life infinitely richer. Lastly, thank you to the Columbia Ballroom team and the New York Ballroom community for allowing me to dance everyday, connecting me with people outside the department with wild backgrounds and even wilder dreams, and for reminding me of the importance of practicing the basic steps, over and over and over again.

Dedication

À papi Albert.

Preface

Chapters 1 and 2 serve as an introduction, describing the big problems I have tried to tackle and setting the stage for the book. These chapters review well-studied topics, as well as more novel results which have appeared in recent papers and at conferences proceedings, on some of which I have had the privilege to be a co-author. The main body of this dissertation is based on three papers which form the basis for Chapters 3, 4, and 6. Chapter 3 includes several extensions of the result initially presented by Margossian, Hoffman, and Sountsov (2021). Chapter 5 presents a generalization of the results introduced in Chapter 4 (or Margossian et al. 2020b), and serves as the early scaffolding of an upcoming paper. All chapters are meant to mostly stand on their own, meaning each can be read independently.

While this dissertation focuses on inference algorithms, much of my approach and interest in Statistics is driven by the applied work I do with outstanding colleagues, mostly in Pharmacometrics and Epidemiology, but also in Physics, Astronomy, Ecology, Genetics, and Econometrics. That is why the presented algorithms are often stress-tested by models which arise in scientific contexts and break away from traditional statistics. Oftentimes I use the term "simple" to describe a model by which I do not mean that the model is trivial, rather that it is a simplification of a more elaborate scientific model. These simplifications help me distill the challenges posed by a class of models and run experiments in a more timely manner.

To my own surprise, a substantial fraction of my work concerns *automatic differentiation*, that is the algorithmic computation of derivatives. I debated including a chapter introducing automatic differentiation with the concern that the topic is quite a rabbit hole and maybe somewhat orthogo-

nal to my committee's interests. A candidate for such a chapter was the review I wrote a few years ago, titled "A Review of automatic differentiation and its efficient implementation" (Margossian 2019), which I believe is a fairly gentle introduction to the subject and seems to have gotten a fair amount of attention. Another option was the paper I wrote with Michael Betancourt on the scalable differentiation of implicit functions (Margossian and Betancourt 2022). This paper contains a detailed introduction to automatic differentiation, which may seem extensive but safeguards us against many pitfalls (a common feature in Michael's writing, who deserves most of the credit for said introduction). The rest of the paper presents a unified framework for the application of automatic differentiation to implicit functions, establishing connections between seemingly disparate methods. A consequence is that steps, which in past derivations seem pulled out of thin air, are now well-motivated and even natural. This work is useful to many statistical applications but it remains tangential (pun intended). I have therefore decided to include this chapter as an appendix to an already long dissertation.

Chapter 1: Letting Uncertainty intervene

CHARLES C. MARGOSSIAN

> Ainsi ce que [l'homme absurde] exige de lui-même, c'est de vivre *seulement* avec ce qu'il sait, de s'arranger de ce qui est, et ne rien faire intervenir qui ne soit certain[1].
>
> Albert Camus,
>
> Le Mythe de Sisyphe, 1942

In his essay *Le Mythe de Sysiphe*, Albert Camus asks the "question of suicide" – that is whether or not to live – and begins with a review of what past philosophers have written on the subject. What interests us here is not the object of Camus' essay but his method, which is in part captured by the above citation and the mandate to "not let anything intervene which is not certain." His predecessors, Camus argues, all resort to a *leap*, at one critical point or the other, to reach their conclusions. Camus wants to safeguard himself from these leaps. He readily admits that nothing is certain and wants to use this certitude (that nothing is certain) as the one and only starting point for his reasoning. His goal is to construct a philosophy which does not cheat its way out of difficult questions and reaches its conclusions only with the upmost rigor.

Camus may remind us of a mathematician, who doesn't go beyond the logical implications of their axioms. Once the parameters of the problem are set, everything else must follow. My undergraduate education in Physics exposed me to a similar notion: we were given a problem with exact inputs to which we applied the *laws* of Physics to obtain an exact solution. The problems I encountered in experimental Physics, and later observational Astronomy, were quite different

[1] *The [practitioner of my philosophy] demands of himself to live only we what he knows, to only deal with what is and not let anything intervene which is not certain.* Note that the term *l'homme absurde* in the original French refers to the person who accepts the absurdity of the world, and the adjective *absurde* is not meant pejoratively.

because we were dealing with imperfect data. Writing down the equation and plugging in the numbers was not enough. The task of running calculations was deeply complicated. In some sense to analyze data is to violate Camus' mandate: we are letting something uncertain intervene. Suddenly the perfect, reductionist machinery of Physics must be complemented by the machinery of Statistics.

Statistics asks a simple question: We observe something, what do we learn? The answer is often neither nothing, nor everything. We cannot jump to conclusions, nor ignore the data. Things would be simpler if we could stick to indisputable facts. One solution is to construct very precise experiments so as to reduce, even eliminate, uncertainty. Then something can be said, then the theory can be confirmed or falsified. This is the scientific paradigm historically epitomized by Antoine Lavoisier, who constructed several careful experiments to enable precise measurements (Lavoisier 1789), thereby replacing alchemy with modern chemistry.

Unfortunately most of the data scientists have does not meet any standard of perfection. It is contaminated by noise. Does this make the data worthless? No. It makes it non-ideal and the uncertainty must be dealt with. Our conclusions and our propositions should be expressed in a language suitable to describe uncertainty and that language is the language of probability. This dissertation deals with probabilistic models and Bayesian inference. In the Bayesian framework we endow unknown quantities with a probability measure, whereby we quantity the distribution of plausible values a latent variable may take, according to our model and our data.

This is a departure from the point estimation framework, in which we estimate unknown quantities using a single value which optimizes an objective function. While for instance the maximum likelihood estimator produces, by some measure, the best fit to the data, it usually does not yield the only plausible fit. Bayesian inference allows us, at least in theory, to consider an ensemble of fits to our data and quantify how good each fit is relative to one another through the posterior density. Consider a standard pharmacokinetic model, which describes how the drug diffuses in a

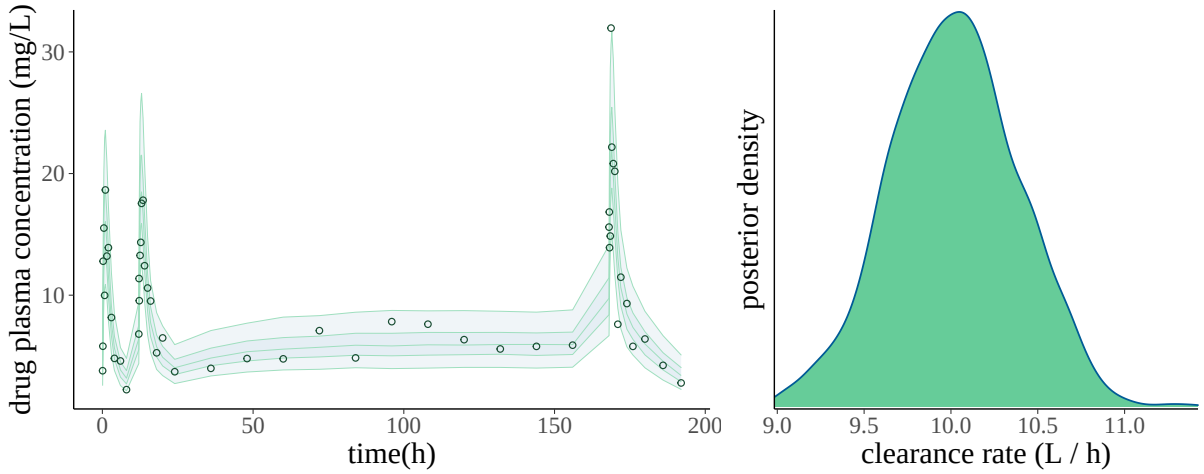

Figure 1.1: Bayesian inference on a pharmacokinetic model. *(Left) The points represent the measured drug concentration. The ribbons are obtained by simulating from a fitted model and represent the* 50% *and* 95% *posterior interval. (Right) While we cannot exactly calculate the clearance rate, the posterior gives us a distribution of plausible values.*

patient's blood during a clinical trial. The model adopts a log-normal likelihood,

$$y \sim \text{logNormal}(\log f(t, \theta), \sigma^2),$$

where f describes the evolution of the drug over time. The function f can be derived by solving a differential equation and properly describing the clinical event schedule of the trial. For more details see, for example, Margossian, Zhang, and Gillespie (2022, section 2). f is parameterized by various pharmacological parameters, including CL, the clearance rate of the drug. Based on our noisy data, can we calculate CL? No, not exactly at least. We can however construct a distribution of plausible values for CL (Figure 1.1). Bayesian inference provides us with a measure of uncertainty, since a broad posterior indicates that wildly different fits explain the data. Similarly a large posterior variance for an unknown quantity of interest suggests that our model is not confident in what this quantity might be. This can inform our decision to trust the model, revise it, or collect more data. There are also practical points in dealing with a distribution, rather than an optimum, since for some models the objective function may be ill-posed. Two important examples are mixtures of distributions and hierarchical models, which do not admit a finite likelihood but

still produce well-posed posterior distributions with finite moments.

Bayesian models also allow us to encode assumptions about certain variables through prior distributions. The non-probabilistic alternative would be to either fix the variables or treat them as completely unknown. This can limit the types of model we use and it is often not an accurate description of our state of knowledge. Let us consider a concrete example: in an early analysis of the COVID-19 outbreak (Hauser et al. 2020), my colleagues and I estimated the mortality rate, based on data collected on a subsample of the population, biased towards individuals with severe symptoms. To make the model identifiable, we needed to provide the rate of symptomatic individuals. This quantity is unknown and fixing it would have meant making a strong assumption. There were however isolated circumstances where an entire population had been tested, whether individuals were symptomatic or not. A meta-analysis of these cases allowed us to construct a prior distribution on the symptomatic rate, which reflected our knowledge (and lack thereof). The Bayesian machinery furthermore ensured that the uncertainty encoded in this prior propagated to the posterior distribution.

One point of contention with Bayesian Statistics is that we are treating unknown variables as random variables when in fact they may be constant. Here the term *random variable* needs to be understood in its abstract measure theoretical sense (e.g. Jacod and Protter 2004, Chapters 1, 5); we want to use this mathematical formalism however it may be useful to do so, and not conflate this formalism with colloquial connotations of the word "random". It is possible to estimate the speed of light in vacuum, c, using a Bayesian model, without violating the postulates of Special Relativity (e.g Gelman et al. 2013, Chapter 3). So long as the constant is unknown, it is useful to endow it with a distribution to describe how consistent various values of the unknown are with the data and the model.

A more substantial objection is that any conclusion based on the posterior distributions takes the model we are using as given. This is a leap. The phrase "according to our model" should be more ubiquitous. Crucially inference must be distinguished from Statistics as a broader field, which puts our modeling choices under scrutiny. Bayesian modeling does not stop at Bayesian

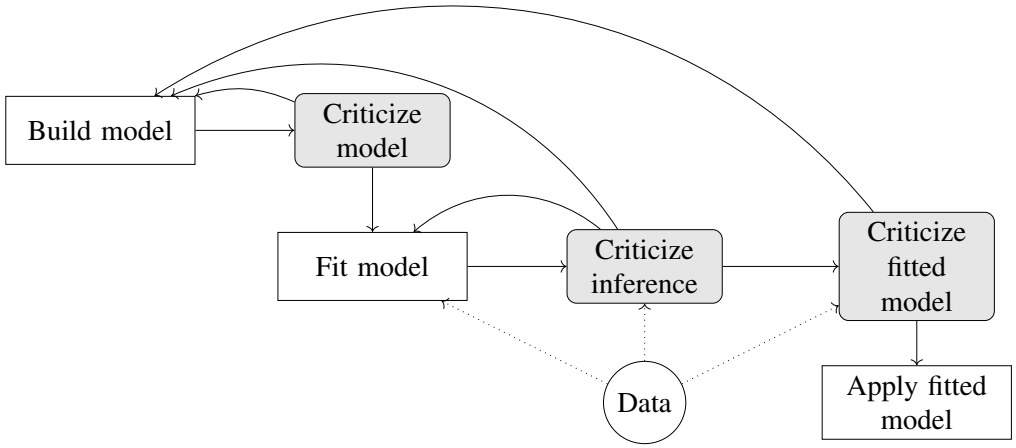

Figure 1.2: Model development as an iterative process. *Fitting the model is only one step in the broader workflow we use to analyze data. Figure borrowed from Grinsztajn et al. (2021).*

inference. Models must be criticized and, if needed, revised. This idea is reflected in many articles, notably discussions on *Box's loop* (Box and Draper 1987; Blei 2014) and the *Bayesian Workflow* (Gabry et al. 2019; Betancourt 2020; Gelman et al. 2020), which describes model development as an iterative process (Figure 1.2). Going back to the COVID-19 example, the model presented by Hauser et al. (2020) is the $\sim 15^{\text{th}}$ iteration, obtained after multiple iterations of Box's loop.

Colleagues and I explore in details the development process of the COVID-19 model in a subsequent paper (Grinsztajn et al. 2021), an article I am particularly fond of because it devotes a great deal of time to failing models. In doing so, our paper elucidates the oftentimes hidden missteps required to develop a useful model. While we can be methodical in our approach, there is no systematic way to address failing models, something papers on the Bayesian Workflow readily recognize. It is difficult, if not impossible, to automate the workflow, certainly once we are willing to consider a rich enough set of models to analyze our data. At some point, the modeler makes a choice. As my colleague Dan Simpson pointed out to me, the ability to revise the model presents us with a degree of freedom that our modeling theories have a difficult time accounting for. This can be a problem, particularly when we are trying to use data to verify a model. The pragmatic realization here is that, once we have exposed the limitations of a model, we are morally obligated

to revise the model. A confirmatory analysis which falsifies a model becomes exploratory.

If I am willing to use expressions such as "criticize the model" or "failing models", surely I must have in mind some absolute measure of how good a model is. I do not. What bearing on reality a model has is a deep and difficult question. Researchers in Machine Learning adopt a pragmatic perspective. They cut to the chase: how well does a model perform a task? It remains to define the task: prediction, classification, sorting, or measurement (e.g. inverse problems) to name a few examples. Often however we expect models to give us insights in more than one way, which complicates model criticism. For example, we may ask of our model to be both predictive and scientifically interpretable.

This dissertation focuses primarily on probabilistic computation and Bayesian inference. While we study these subjects we should keep in mind the bigger picture of the workflow. How might a modeler use the algorithms in this book, not only to do one calculation but more broadly to do scientific research? How well does an algorithm support the workflow? Is it amiable to criticism? Can an algorithm be applied to a range of models without requiring too much tuning effort from the user? What trade-offs does it present between speed and flexibility, and how does this impact model development? Once we consider enough dimensions, it becomes difficult to elect one algorithm as uniformly the best. Different methods present different trade-offs, and we must do our best to paint a nuanced algorithmic landscape.

Acknowledgment

I thank Andrew Gelman, Brynn Deprey, and Lowri Thomas for proofreading an early draft of this chapter.

Chapter 2: What makes Bayesian computation hard?

CHARLES C. MARGOSSIAN

Many scientific problems require us to understand the properties of a probability distribution with an unknown normalizing constant. This is a fundamental problem in the study of probabilistic models, such as Ising models, and in Bayesian statistics, where all our inferential conclusions follow from the posterior distribution. This chapter discusses challenges that arise when trying to compute the posterior. Our goal is to gain additional perspectives with which we can understand the role of our algorithms across a rich modeling space. Surveying many examples allows us to distinguish specialized algorithms which are efficient on a narrow class of models and general purpose tools. Among the latter, Hamiltonian Monte Carlo (HMC) has emerged as a powerful method for Bayesian inference. This chapter provides a review of HMC and its implementations in probabilistic programming languages. Despite its broad success, HMC may still struggle with certain classes of models. To understand what makes a model difficult, I propose to reason about the computational challenges respectively posed by the prior and the likelihood. I review hierarchical priors, a well-studied and notoriously difficult class of priors. As a case study for challenging likelihoods, I analyze ordinary differential equation (ODE)-based likelihoods, exploring new ideas on how to study ODEs in a Bayesian context. Throughout this chapter, the probabilistic programming language Stan is used to demonstrate the implementation of inference algorithms and perform Bayesian analysis on a few examples.

2.1 General purpose and specialized algorithms

Our ability to efficiently compute estimators of the posterior mean, variance, and quantiles varies wildly across models. To reason about what makes a Bayesian model computationally chal-

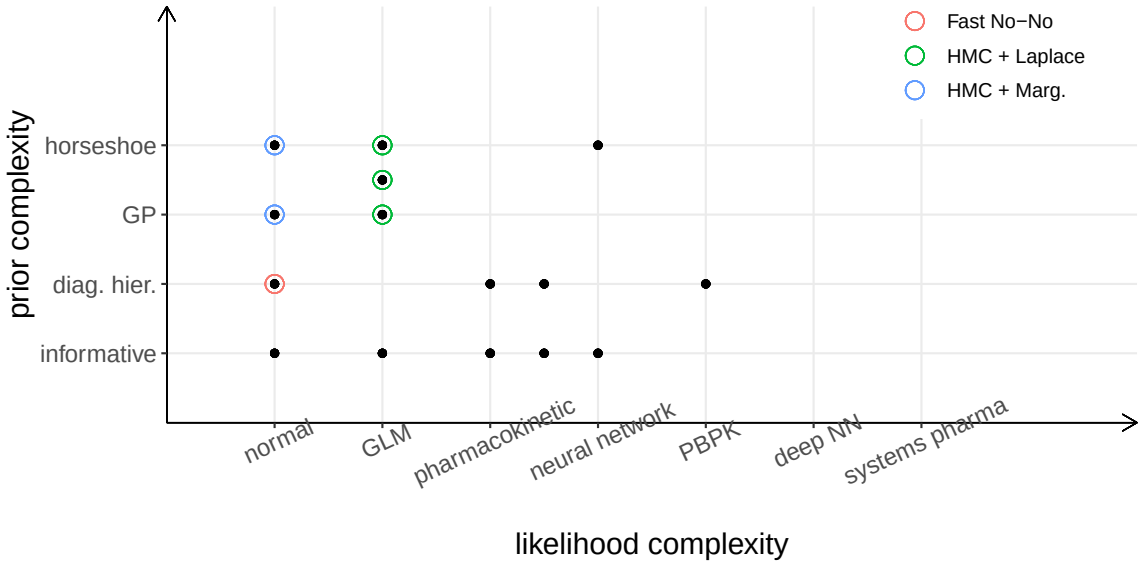

Figure 2.1: Qualitative topology of Bayesian models. *A set of models (Table 2.1) organized according to how computationally challenging their prior and their likelihood are. Specialized algorithms, such as the fast normal-normal ("Fast No-No") or the embedded Laplace approximation ("HMC + Laplace") scale well with large data and can handle challenging priors for certain models. A general purpose tool of choice is MCMC. GP stands for "Gaussian process" and PBPK for "physiologically based pharmacokinetic".*

	Likelihood	Prior	Reference
Regularized linear regression	Normal	Normal	
COVID-19 survey	Normal	Diagonal hierarchical normal	(Greengard et al. 2021)
Spatiotemporal COVID-19 survey	Normal	Dense Gaussian process	Section 2.1
Prostate cancer microarray (simplified)	Normal	Horseshoe prior	
Regularized Poisson log	Poisson log	Normal	
Disease map of Finland	Poisson log	Gaussian process	Chapter 4
Sparse kernel interaction model	Bernoulli logit	Horse shoe Gaussian process	Chapter 4
Prostate cancer microarray	Bernoulli logit	Horse shoe	Chapter 4
PK 2 cpt	linear ODE with clinical event schedule	Informative	e.g (Margossian, Zhang, and Gillespie 2022)
Population PK 2 cpt	linear ODE with clinical event schedule	Diagonal hierarchical normal	e.g. (Margossian, Zhang, and Gillespie 2022)
Friberg-Karlsson semi-mechanistic PK/PD	nonlinear ODE with clinical event schedule	Informative	(e.g. Friberg et al. 2002, Margossian, Zhang, and Gillespie 2022)
Population Friberg-Karlsson semi mechanistic - PK/PD	nonlinear ODE with clinical event schedule	Diagonal hierarchical normal	(e.g. Friberg et al. 2002, Margossian, Zhang, and Gillespie 2022)
Prostate cancer NN	Neural network	Normal	(Broussard et al. 2021)
Sparse prostate cancer NN	Neural network	Horseshoe on first layers, normal otherwise	(Broussard et al. 2021)
"Monster" PBPK	nonlinear ODE	Diagonal hierarchical, strongly informative	(Gelman, Bois, and Jiang 1996) (in progress)
Bone mineral density systems pharmacology (no full Bayesian inference)	mechanistic model, large non-linear ODE	Informative	(Peterson and Riggs 2010)

Table 2.1: Models displayed in Figure 2.1. *PK stands for "pharmacokinetics", PD for "pharmacodynamics", PB for "physiologically based" and NN for "neural network". For additional references on pharmacokinetics and pharmacodynamics models, see Gibaldi and Perrier (1982) and Gastonguay and Metrum Institute Facility (2013).*

lenging, we may consider three qualitative axis: (i) the "complexity" of the likelihood (ii) the "complexity" of the prior and (iii) the size of the data. Each of these axis alone can lead to computational challenges, resulting in prohibitively slow and inaccurate inference. Useful algorithms present trade-offs across this modeling space. Figure 2.1 and Table 2.1 present a diverse set of models with a combination of challenging priors and likelihoods, some of which we discuss in upcoming chapters.

A general purpose inference tool of choice is Markov chains Monte Carlo (MCMC)(Neal 1993; Robert and Casella 2004). Over the past decade the *Hamiltonian Monte Carlo* (HMC) sampler has emerged as one of the more useful MCMC algorithms (Duane et al. 1987; Neal 2012; Betancourt 2018a). Provided we can evaluate the gradient of the log posterior density it is possible to apply HMC and, under certain regularity conditions we will review in Section 2.2.1, obtain accurate estimations of the posterior moments. As such HMC can be *attempted* on a large class of models, for example every model in Figure 2.1. This does not mean that HMC immediately gives us satisfactory results. Certain models may require careful tuning and reparameterizations and can take a prohibitively long time to achieve a desired accuracy. Long run times are problematic when fitting a single model and the problem is exacerbated when we consider the Bayesian Workflow, in which we build and fit many models.

A specialized algorithm works extremely well on a narrow class of models. The *Fast No-No* algorithm my colleagues and I developed efficiently computes posterior moments for hierarchical models with a normal likelihood, a normal prior with diagonal covariance, and an arbitrary hyper-prior (Greengard et al. 2021). Relative to HMC, the method proves useful in scenarios where we have large data. One example in our paper concerns the analysis of survey data during the first wave of COVID-19, wherein individuals self-report symptoms. We fit a *multilevel regression and poststratification* model (Gelman and Little 1997) to several weeks of data, covering 100,000's of observations, across 1,000's of cities. Running HMC takes ~10 hours; by contrast the Fast No-No algorithm achieves higher accuracy in ~7 seconds. This in itself is an important speed up and furthermore it means the model can be scaled to larger data. On the other hand, any complication

of the model itself is impossible. For example a more elaborate version of the model describes spatiotemporal variations using a Gaussian Process prior, and is no longer compatible with the Fast No-No method.

Approximate Bayesian algorithms try to find within a family of tractable distributions, $\mathcal{Q}$, the distribution which best approximates the posterior by some measure. Examples include the Laplace approximation (Laplace 1774), the integrated Laplace approximation (Tierney and Kadane 1986; Rue, Martino, and Chopin 2009; Rue et al. 2017), and variational inference (Blei, Kucukelbir, and McAuliffe 2017). These approaches are relatively specialized because they rely on the existence of a distribution in $\mathcal{Q}$ which approximates the posterior reasonably well.

In Chapter 4, I present a hybrid approach which combines HMC with an integrated Laplace approximation designed to do Bayesian inference on latent Gaussian models with a non-normal likelihood[1]. For these models, the posterior distribution factors into multiple terms, some of which can be well handled by a Laplace approximation, while others require a more general algorithm. Empirically this hybrid method works well for Bayesian models with a log-concave likelihood, without placing severe restrictions on the type of priors we can use (Figure 2.1). In this sense this algorithm constitutes a semi-specialized algorithm. Chapter 5 explores the potential of this method as we move along the likelihood axis.

Once we consider less conventional likelihoods, it becomes more challenging to find useful approximations. We might choose to simplify the model (hopefully without oversimplifying) which is not unlike using approximate inference. The field of Climate Science presents interesting examples in which one attempts to replace a detailed mechanistic model with a Physics informed Machine Learning model (e.g Willard et al. 2021). In many settings however MCMC remains a tool of choice, especially if we care about faithfully doing full Bayesian inference on our model of choice.

[1] If the likelihood is normal, the Laplace approximation can be replaced by analytical calculations. In this case HMC is combined with an exact marginalization.

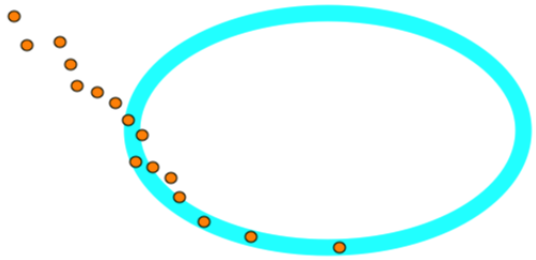

Figure 2.2: Drifting Markov chain. *After being initialized in the target space, a Markov chain produced by MCMC (orange dots) finds and explores the region where the probability mass concentrates (blue band).*

2.2 A Review of Hamiltonian Monte Carlo

This section presents a high-level review of MCMC and HMC, thereby setting the stage for upcoming chapters. I review certain elementary facts, necessary to understand existing challenges, and discuss research directions which have emerged in recent years.

2.2.1 A primer on Markov chains Monte Carlo

MCMC samplers generate approximate samples from a target distribution, say $\pi(\theta \mid y)$, with which we can construct sample estimators, also termed *Monte Carlo estimators*, for various quantities of interest. Classically an MCMC sampler over $\theta \in \Theta$ is fully characterized by two properties:

1. an initial distribution, π_0, from which we draw the first point of our Markov chain,
2. a transition kernel, defined by the conditional distribution $\Gamma(\theta^{(i+1)} \mid \theta^{(i)})$, and which informs how subsequent points of the Markov chain are generated.

Conceptually the transition kernel is designed to guide the Markov chain towards regions where the target probability mass concentrates (Figure 2.2). In contrast to exact independent draws from the posterior, MCMC samples suffer from two defects: (i) the samples are correlated and (ii) the samples are biased by our initialization.

A particularly useful property is for the stationary distribution of the Markov chain to be $\pi(\theta \mid y)$. Under certain conditions, the Markov chain converges to its stationary distribution, meaning that asymptotically a point drawn by our MCMC sampler will have the wanted distribution, and moreover that as we run the algorithm the initial bias decays. This informs the practice of "burn-in" whereby we discard the first, presumably heavily biased, samples produced by our algorithm. While very attractive, this theoretical property does not always translate into a practical algorithm, especially in pathological cases where the initial bias cannot be overcome in any reasonable amount of time. This is a well-known problem in multimodal cases, where transitions between modes can be exceedingly rare. Fortunately in these cases we can often rely on diagnostics, such as the $\hat{R}$ statistics and its variants, to warn us that we have not achieved convergence (Gelman and Rubin 1992; Vehtari et al. 2020).

Another desirable property is for our Monte Carlo estimators to observe a central limit theorem (CLT). Equipped with a CLT, we can characterize the precision of our estimators and furthermore the efficiency of MCMC algorithms. Consider the mean estimator

$$\widehat{\mathbb{E}} f(\theta) = \frac{1}{N} \sum_{i=1}^{N} f\left(\theta^{(i)}\right), \tag{2.1}$$

meant to estimate the expectation value $\mathbb{E} f(\theta)$, using MCMC samples $\{\theta^{(1)}, \theta^{(2)}, \cdots, \theta^{(N)}\}$. *If* a central limit theorem holds, then we have the following convergence in distribution as $N \to \infty$:

$$\frac{\sqrt{\mathrm{ESS}}\left(\widehat{\mathbb{E}} f(\theta) - \mathbb{E} f(\theta)\right)}{\sigma_f} \xrightarrow{d} \mathrm{Normal}(0, 1), \tag{2.2}$$

where

$$\sigma_f^2 = \mathrm{Var}_{\theta|y} f(\theta), \quad \mathrm{ESS} = \frac{\mathrm{Var}_\Gamma \widehat{\mathbb{E}} f(\theta)}{\mathrm{Var}_{\theta|y} f(\theta)}, \tag{2.3}$$

and we assume $\sigma_f < \infty$. Here ESS stands for *effective sample size*. Writing it as a ratio of variance provides a useful perspective; another option is to write it using the Markov chain's autocorrelation,

ρ_t, at lag t (Geyer 2012),

$$\text{ESS} = \frac{N}{1 + 2\sum_{t=1}^{\infty} \rho_t}. \tag{2.4}$$

This formula highlights how a positive autocorrelation causes the effective sample size to be smaller than the actual sample size, N. This in turn manifests as a slower convergence rate in Equation 2.2 and a larger variance for our estimator. Consequently MCMC algorithms which produce Markov chains with a low autocorrelation tend to perform better, although, as we will see in Chapter 3, this condition can be somewhat relaxed.

While the conditions under which we obtain convergence to the stationary distribution and a central limit theorem are verified in many applications, these conditions remain non-trivial. Roberts and Rosenthal (2004) and Betancout (2021) provide reviews on this subject.

MCMC techniques span a history of several decades and continue to be developed (Robert and Casella 2011), with some novel advances presented in this dissertation. One indication of how popular MCMC is amongst the scientific community is the fundamental role it plays in statistical software and probabilistic programming languages. The Metropolis-Hastings algorithm (Metropolis et al. 1953; Hastings 1970) and the Gibbs sampler (Geman and Geman 1984) are the algorithmic foundation of the software BUGS (Lunn et al. 2000) and JAGS (Plummer 2003). The *Hamiltonian Monte Carlo* (HMC) sampler (Duane et al. 1987; Neal 2012; Betancourt 2018a) is favored amongst more recent software, notably Stan (Carpenter et al. 2017), PyMC3 (Salvatier, Wiecki, and Fonnesbeck 2016), TensorFlow Probability (Lao et al. 2020) and Turing (Ge, Xu, and Ghahramani 2018).

HMC offers an attractive alternative to more traditional MCMC algorithms because it is less vulnerable to geometric structures, such as strong correlations across high dimensions (e.g Hoffman and Gelman 2014), resulting in faster convergence to stationarity and lower autocorrelation. Furthermore bias explorations of the posterior distribution can often be diagnosed with HMC, notably through the detection of *divergent transitions* (Betancourt and Girolami 2015; Betancourt 2018a).

Other MCMC algorithms continue to be developed and receive extensive attention in recent lit-

erature, notably sequential Monte Carlo (e.g. Dai et al. 2020), ensemble particles for online learning (e.g Lu and Van Roy 2017), strategies to sample over discrete spaces (e.g. Zhang et al. 2012; Grathwohl et al. 2021; Margossian and Mukherjee 2021), and even Metropolis-Hastings and Gibbs samplers. A discussion of these algorithms is beyond the scope of this chapter and our focus is on HMC.

2.2.2 Transition kernel

I here review elementary details of HMC and direct the reader to Neal (2012) and Betancourt (2018a) for a more thorough introduction. For simplicity, assume $\theta \in \Theta = \mathbb{R}^D$. HMC introduces an auxiliary momentum variable, $r \in \mathbb{R}^D$, and proposes a sampling scheme over the augmented space $(\theta, r) \in \mathbb{R}^{2D}$. At each MCMC iteration, we simulate the trajectory of a fictitious particle over a time τ, subject to a potential energy determined by our posterior distribution. Given a starting point $\theta^{(i)}$, HMC proceeds in two steps:

1. draw $r \sim p(r)$, where $p(r)$ is a distribution we will define,

2. simulate a trajectory, $\theta^{(i)} \to \theta^{(i+1)}$, according to Hamilton's equations of motion.

In this context, Hamilton's equations reduce to the following system of two differential equations

$$\frac{\mathrm{d}r}{\mathrm{d}t} = -\frac{\partial U(\theta)}{\partial \theta}, \qquad \frac{\mathrm{d}\theta}{\mathrm{d}t} = M^{-1} r, \tag{2.5}$$

where U is a *potential energy* function, M is the *mass matrix*, and t is the fictitious time. To obtain a useful transition kernel, we set

$$U(\theta) = -\log \pi(\theta \mid y). \tag{2.6}$$

The change in momentum or equivalently (up to a constant) the *acceleration* of the fictitious particle is then determined by $\nabla_\theta \log \pi(\theta \mid y)$, whilst the movement of the particle is in turn informed by the momentum, $r(t)$.

We can define an energy or *Hamiltonian* for our fictitious particle, according to

$$\mathcal{H}(\theta, r) = U(\theta) + K(r), \tag{2.7}$$

where U is the above defined potential energy and

$$K(r) = \frac{1}{2} r^T M^{-1} r, \tag{2.8}$$

is the *kinetic energy*. Along a simulated trajectory, $\mathcal{H}$ is constant. Furthermore the joint distribution of (θ, r) is given by the *canonical distribution* of Statistical Mechanics,

$$p(\theta, r) \propto \exp\left(-H(\theta, r)\right) = \exp\left(-U(\theta) - K(r)\right), \tag{2.9}$$

which indicates our physical system prefers states with a low energy. The factorization of the Hamiltonian implies that $p(\theta, r) = p(\theta)p(r)$, from which we recover the marginal distributions,

$$p(\theta) \propto \exp\left(-U(\theta)\right) = \pi(\theta \mid y), \tag{2.10}$$

as desired, and

$$p(r) \propto \exp\left(-\frac{1}{2} r^t M^{-1} r\right), \tag{2.11}$$

which is the kernel of a normal distribution with covariance matrix M.

2.2.3 Tuning parameters

The performance of HMC is highly sensitive to certain tuning parameters. Here performance must be understood as the ability to (i) converge quickly to the stationary distribution and (ii) efficiently generate samples with a low autocorrelation. Unfortunately there exists no optimal tuning that works across the range of models we are interested in. Tuning HMC can be a significant burden for users, which might explain the somewhat slow adoption of the method by the scientific

community.

A major milestone for the popularization of the algorithm is reached with the development of self-tuning or *adaptive* HMC, such as the No-U-Turn-Sampler (NUTS) (Hoffman and Gelman 2014) and its later variation as dynamic HMC (Betancourt 2018a). The burn-in phase now serves not only to overcome the initial bias but also to tune the sampler, using information from the early samples. Hence the term *burn-in* is often replaced with the more appropriate name *warmup*. The wide implementation of these methods in statistical software and their application to many scientific problems speaks to their success. Nevertheless the problem of tuning MCMC is not completely solved. Even dynamic HMC can require fine-tuning, once confronted to difficult enough problems, as we will see in Chapter 4.

Another critique of NUTS and dynamic HMC is that their tuning strategy is somewhat incompatible with MCMC regimes which exploit modern hardware such as Graphics and Tensor Processing Units (GPUs, TPUs). This is notably a problem in contexts where we run many chains in parallel to achieve good performance (Hoffman and Ma 2020; Hoffman, Radul, and Sountsov 2021). As a result, new adaptive HMC samplers are being developed, with recent additions to the literature including the ChEES-HMC (Hoffman, Radul, and Sountsov 2021) and the SNAPER-HMC (Sountsov and Hoffman 2021) algorithms, both implemented in TensorFlow Probability. In Chapter 3 we further study how modern hardware can help us address general MCMC tuning problems by studying the question of diagnosing convergence.

Integrator step size

I now briefly review the tuning parameters of HMC. The first challenge we encounter is that we cannot solve Hamilton's equations of motion analytically. Instead we resort to a *leapfrog integrator* (Algorithm 2.1), a numerical integrator parametrized by a step size ϵ. This tuning parameter controls the trade-off between the precision and speed of the integrator. The accuracy of the integrator can be vetted by checking that the Hamiltonian, $\mathcal{H}(\theta, r)$, is conserved along the trajectory. To correct for errors in our computation, we can add a Metropolis step after simulating

the trajectory, $\theta^{(i)} \rightarrow \theta^{(i+1)}$, from time $t = 0$ to time $t = \tau$, with acceptance probability,

$$\Pr(\text{accept}) = \min\left(\frac{p(\theta^{(i+1)}, r(\tau))}{p(\theta^{(i)}, r(0))}\right). \tag{2.12}$$

Another option, notably implemented in Stan, is to sample along the trajectories, using the Hamiltonian as weights (Betancourt 2018a). A target acceptance probability can be used to tune ϵ during the first MCMC iterations. The default target acceptance in Stan is 0.8, although it is often useful to increase this to higher values.

Algorithm 2.1: Leapfrog integrator for simulating Hamiltonian trajectory. *The second gradient evaluated in a leapfrog step can be used as the first gradient for the next step. Hence we only need one gradient evaluation per step.*

input: trajectory length L, step size ϵ, starting position $\theta(0)$, and momentum $r(0)$.
for $t \in \{0, 1, \cdots, L-1\}$ **do**
  $r(t + \epsilon/2) \leftarrow r(t) + \frac{\epsilon}{2}\nabla_\theta \log \pi(\theta(t) \mid y)$
  $\theta(t + \epsilon) \leftarrow \theta(t) + \epsilon M^{-1} r(t + \epsilon/2)$
  $r(t + \epsilon) \leftarrow r(t + \epsilon/2) - \frac{\epsilon}{2}\nabla_\theta \pi(\theta(t + \epsilon) \mid y)$
end
return: $\theta(L\epsilon)$

Trajectory length

The next crucial tuning parameter is the length of each simulation, or τ, before we resample the momentum. A short trajectory results in inefficient movement across Θ, while a long trajectory is computationally wasteful. How to best set τ remains an active research area with different criterions presenting various trade-offs (e.g Neal 2012; Hoffman and Gelman 2014; Hoffman, Radul, and Sountsov 2021; Sountsov and Hoffman 2021).

Mass matrix

Finally we need to pick the mass matrix, M. What mass could we possibly ascribe to our fictitious particle? Once again the analogy with Physics serves us well: mass is the resistance to

acceleration, and moreover from Equation 2.5, reproduced below,

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = M^{-1}r,$$

it is clear that M determines the directions in Θ along which the fictitious particle moves the least. This naturally suggests setting M^{-1} to the posterior covariance, or rather some early estimate of the covariance. The particle then moves more along directions with a high variance. This heuristic breaks down when the directions along which we wish to move varies with θ. This is a notorious problem for hierarchical priors, which are prone to generate Neal's funnel (Neal 2003) (Figure 2.3, right). A more refined approach is to use a dynamic mass matrix and set $M(\theta)$ to the local curvature, as we move across the parameter space. This is the underlying idea behind the Riemannian Hamiltonian Monte Carlo (RHMC) sampler (Girolami and Calderhead 2011). Unfortunately RHMC's expansive requirement to compute and store higher-order derivatives of $\pi(\theta \mid y)$ has limited its widespread use.

The mass matrix receives various treatments across implementations. Hoffman, Radul, and Sountsov (2021), in their discussion of ChEES-HMC, simply take it to be the identity matrix. Stan's default is to use a diagonal inverse mass matrix, M^{-1}, with each entry corresponding to the sample posterior variance, estimated using early samples during the warmup phase. In the common case where the posterior covariance is non-zero, a dense inverse matrix can improve the per iteration performance of HMC. This however comes at a dire computational cost: each leap frog step involves a matrix multiplication (Algorithm 2.1, line 4),

$$\theta(t + \epsilon) \leftarrow \theta(t) + \epsilon M^{-1} r(t + \epsilon/2), \tag{2.13}$$

with costs $O(D^2)$ if M^{-1} is dense, versus $O(D)$ for diagonal M^{-1}. A similar complication occurs when we consider the cost of sampling the auxiliary momentum. The problem is exacerbated by memory considerations in high-dimensions and the fact that the sample covariance may be inaccurate. Stan's default is based on the expectation that these added computational costs do not

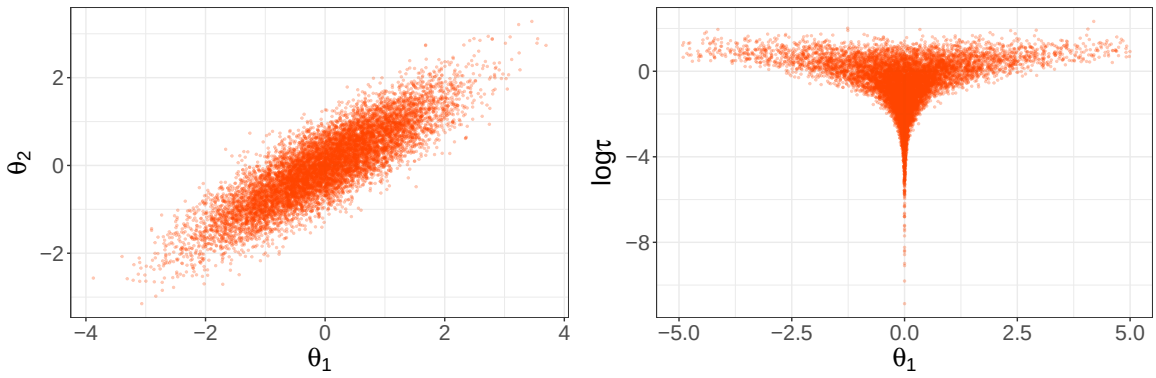

Figure 2.3: Posterior geometry in hierarchical models. *(Left) The directions along which to move can be well characterized by a non-diagonal mass matrix which approximates the posterior covariance. (Right) In the funnel case, we cannot build a suitable mass matrix because the curvature of the posterior varies across the parameter space and diverges as* $\tau \to 0$.

justify the improved adaptation.

Bales et al. (2019) study a strategy using a low-rank approximation of the posterior covariance, which is for instance suitable when considering elliptical posterior distributions (Figure 2.3, left). The costs of each HMC iteration remains linear in D with additional costs coming from the rank of M^{-1}. Zhang et al. (2021) also propose to estimate a low-rank mass matrix, this time using a variational approximation of $\pi(\theta \mid y)$, rather than a sample covariance.

Adaptive warmup

HMC's warmup serves two purposes: (i) converging (approximatively) to the stationary distribution and (ii) adapting the tuning parameters in order to reduce the autocorrelation of the chain once we start sampling. Regarding (i), it is worth noting that the Markov chain typically quickly finds *the sampling region*, i.e. the region where the probability mass concentrates. Even then the chain remains biased by its initialization and it must sojourn across the sampling region in order to make the initial bias negligible. In Stan, adaptation of the tuning parameters occurs over different phases (Table 2.2). Phases I and III are short, and serve to adapt the step size, ϵ. The more substantial Phase II is used to adapt the mass matrix, M. The length of each phase, and moreover the length of the warmup is determined ahead of time, with Stan's default being 1,000 iterations.

	Stan's adaptive HMC	Prototype Path finder
I (75 iter)	• starting at initialization, θ_0, early exploration with highly varying θ. • convergence to sampling region. • initial tuning of the step size ϵ.	• Construct optimization path between θ_0 and the mode of $\pi(\theta \mid y)$. • Find sampling region along path and initialize HMC Phase I.
II (875 iter)	• semi-stable exploration of sampling region. • extensive tuning of the inverse mass matrix, as we learn more about $\pi(\theta \mid y)$.	• Use variational approx. of $\pi(\theta \mid y)$ to tune sampling parameters. • OR use Phase II of HMC.
III (50 iter)	• continued exploration of the sampling region. • final tuning of the step size ϵ.	• Use HMC's Phase III.

Table 2.2: Adaptive Warmup of Stan's HMC and the prototype pathfinder over 1000 iterations

Strategies exist to adaptively set the warmup length (e.g. Zhang et al. 2020; Margossian, Hoffman, and Sountsov 2021) and will be examined in Chapter 3.

Zhang et al. (2021) explore several directions to bypass certain steps of Stan's warmup (Table 2.2). Initial convergence to the sampling region is replaced with a *pathfinding* procedure, wherein one finds the mode (or pole in cases where the mode is ill-defined) of $\pi(\theta \mid y)$, and identifies a point on the sampling region along the optimization path (Figure 2.4). The procedure is further enhanced by using a variational approximation of the posterior distribution. Using this approximation, we can generate initial samples for the Markov chain and estimate the posterior covariance, thereby setting M^{-1} and bypassing Phase II of Stan's warmup. We experiment with this approach on several pharmacokinetic examples and find the benefits to be uneven across models (Section 2.4).

Another warmup strategy is to share information between multiple chains running in parallel. In an HMC context this idea is used in the cross-chain warmup by Zhang et al. (2020) and the ChESS-HMC by Hoffman, Radul, and Sountsov (2021). These strategies naturally take advantage of scenarios where we run many chains in parallel. But like all parallel schemes, they assume a more or less homogeneous behavior amongst the chains. ChEES-HMC's strategy to use the same step size and trajectory length for all chains can be problematic, notably when the behavior of the likelihood varies dramatically across the parameter space. I examine such examples in Section 2.4.

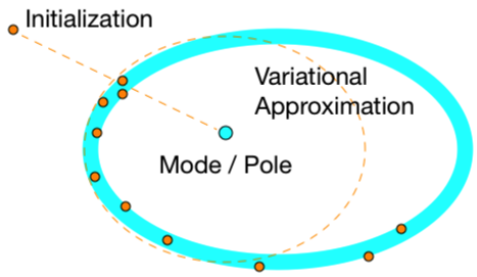

Figure 2.4: Pathinder. *Starting far off in the parameter space, the pathfinder uses an optimizer to find the posterior mode. Due to concentration of measure, the probability mass concentrates away from the mode, hence we want to initialize MCMC not at the mode, but along the optimization path.*

In Chapter 3 I study MCMC strategies which run many chains in parallel with a short sampling phase, potentially with a single iteration. If each chain generates exactly one draw, we no longer care about the autocorrelation of the chain. It remains to insure that the chains converge. In other words, the warmup now only serves one goal. On the other hand a well-tuned sampler is expected to converge faster. Understanding how much we can compromise on the warmup in the many-short-chains regime therefore remains an important open question.

2.2.4 Gradient computation

HMC requires computing the gradient of the log posterior density. Fortunately this can be obtained by differentiating the log prior and log likelihood, given

$$\nabla_\theta \log \pi(\theta \mid y) = \nabla_\theta \log \pi(\theta) + \nabla_\theta \log \pi(y \mid \theta). \tag{2.14}$$

The burden of implementing gradients by hand is in large parts alleviated by *automatic differentiation*, which numerically evaluates derivatives based on code to compute $\log \pi(y, \theta)$ (Griewank and Walther 2008; Baydin et al. 2018; Margossian 2019). Despite its wild success, automatic differentiation does not, at least not immediately, solve the problem of efficiently computing the derivatives of non-trivial functions. We will extensively concern ourselves with this problem in Chapters 4 and 5, and Appendix A, notably when discussing likelihoods based on implicit functions.

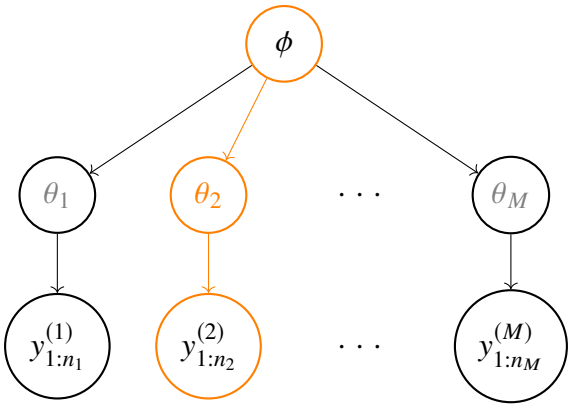

Figure 2.5: Graphical representation of a simple hierarchical model. *The super index on the $y^{(i)}_j$ indicates the observation belongs to the i^{th} group, parameterized by θ_i. For fixed ϕ, the marginal posterior distribution $\pi(\theta_2 \mid y, \phi)$ only depends on observations in the second group.*

2.3 Example of challenging priors: hierarchical models

Priors can frustrate our inference algorithms by inducing non-trivial correlations between parameters. These correlations manifest as geometric structures in the posterior density, such as the ellipse and the funnel displayed in Figure 2.3. Anisotropic densities can be difficult to estimate with approximate inference: for example, mean field variational inference assumes the parameters have no posterior correlation, an easily violated assumption. MCMC too can be frustrated by anisotropies: random-walk algorithms, such as the Gibbs and Metropolis-Hastings samplers, may fail to find good directions along which to "walk" and become inefficient. In the case of HMC, it becomes difficult to adapt the mass matrix and other tuning parameters, resulting in slow convergence and large autocorrelation.

An important example of challenging prior arises in hierarchical models, which are defined by a series of conditional distributions

$$\begin{aligned} \phi &\sim \pi(\phi), \\ \theta_1, \theta_2, \cdots, \theta_M \mid \phi &\sim \pi(\boldsymbol{\theta} \mid \phi), \\ y_1, y_2, \cdots, y_N \mid \boldsymbol{\theta}, \phi &\sim \pi(y \mid \boldsymbol{\theta}, \phi). \end{aligned} \tag{2.15}$$

Hierarchical models constitute a broad class of models and are used to describe data generated by heterogeneous sources. Consider the following example: a population pharmacokinetic model describes the drug diffusion process using data from multiple patients and ascribes to each patient their own physiological parameters. The data for the k^{th} patient is conditionally dependent only on the parameters corresponding to that patient,

$$\pi(y_{i\in g(k)} \mid \boldsymbol{\theta}) = \pi(y_{i\in g(k)} \mid \boldsymbol{\theta}_{j\in g(k)}), \tag{2.16}$$

where g maps to the relevant indices for the k^{th} patient.

While each patient is assigned their own distinct parameters, we model these parameters as belonging to the same population. For simplicity, let us assume that there is only one parameter, θ_k, per patient, and furthermore that the hierarchical prior is normal with a diagonal covariance,

$$\boldsymbol{\theta} \sim \text{Normal}(\boldsymbol{\mu}, \tau^2 I_M). \tag{2.17}$$

We also assume that the elements of $\boldsymbol{\mu}$ are all identical, meaning the θ's are regressed to a common mean. Our hyperparameters are then simply $\phi = (\mu, \tau)$. The above formula states that the θ_k's are heterogeneous but still bear certain similarities, since they all come from the same population. If μ and τ are fixed, we have constructed a regularizing prior. Conditional on y and ϕ, the θ_k's only depend on the observations in the k^{th} group (Figure 2.5). A hierarchical model treats the hyperparameters as additional unknown variables. This allows a "back-and-forth" flow of information and the phenomenon of *partial pooling*, whereby the marginal posterior distribution of θ_k depends on the data for patient k and the data for all the other patients. This dependence structure can be seen by writing out the corresponding graphical model (Figure 2.6). Conceptually this means that what we learn from one patient can teach us something (though not everything) about another patient.

A side-effect of partial pooling is that the θ_k's may be correlated. The interaction between τ and θ_k also generates geometric structure. As τ goes to 0, the hierarchical prior strongly regresses θ_k to μ. For lager τ, little regularization over θ_k occurs, meaning the variance of θ_k increases. The

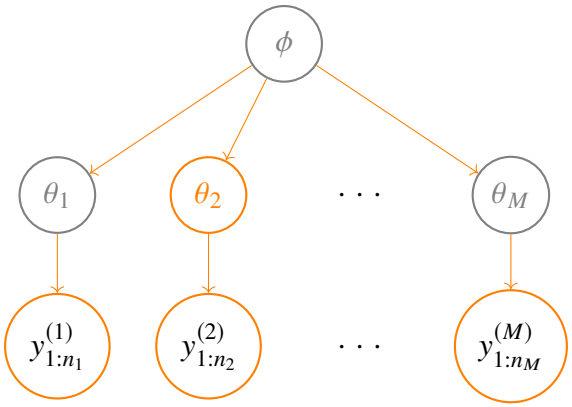

Figure 2.6: Graphical representation of a simple hierarchical model. *If we marginalize out ϕ and $\boldsymbol{\theta}_{-2}$, the marginal distribution $\pi(\theta_2 \mid y)$ depends on the observations in all groups.*

result is a funnel shape (Figure 2.3). In the limit where τ goes to 0, the joint density blows up leading to numerical instability. The funnel is manifested by the prior; whether it actually appears in the posterior depends on the likelihood. Sparse data typically allow the prior to dominate and result in a funnel.

There exists various approaches to address this well-known problem. A popular strategy is reparameterization (Papaspiliopoulos, Roberts, and Sköld 2007). Our above formulation of the model is a *centered parameterization*. The *non-centered parameterizartion* produces an equivalent data generating process using a standard normal vector, $\boldsymbol{\eta}$, and reconstructs $\boldsymbol{\theta}$ as follows:

$$\begin{aligned} \eta_k &\sim N(0,1), \\ \theta_k &= \tau\eta_k + \mu. \end{aligned}$$

Our sampler is then applied to $\pi(\boldsymbol{\eta}, \tau, \mu \mid y)$, with posterior samples of $\boldsymbol{\theta}$ obtained using the above transformation. The absence of an explicit prior on $\boldsymbol{\eta}$ which depends on τ and μ can result in a better behaved posterior density. The geometry can still be problematic, producing inverted funnels, depending on how the likelihood and the prior interact. In general it is difficult to choose which parameterization to use without trial and error, and the problem is exacerbated by the possibility that a particular parameterization may be appropriate only for a subset of the θ_k's, especially when

the amount of data in each group is uneven.

A different angle is to marginalize the θ_k's, thereby removing the offending geometry. This strategy proceeds in two steps:

1. draw samples from the marginal posterior $\pi(\mu, \tau \mid y) \propto \pi(y \mid \mu, \tau)\pi(\mu, \tau)$.

2. draw samples from the conditional distribution $\pi(\boldsymbol{\theta} \mid y, \mu, \tau)$.

Provided we can implement this approach, we obtain samples from the joint posterior distribution, $\pi(\boldsymbol{\theta}, \mu, \tau \mid y)$. In all but the simplest cases, we lack an analytical expression for $\pi(y \mid \mu, \tau)$ and $\pi(\boldsymbol{\theta} \mid y, \mu, \tau)$. We can in certain instances resort to approximations of these distributions, for instance using a Laplace approximation (Tierney and Kadane 1986; Rue, Martino, and Chopin 2009; Margossian et al. 2020b), a topic we will discuss in Chapters 4 and 5.

2.4 Example of challenging likelihoods: ODE-based models

Bayesian inference algorithms require us to compute the log joint distribution, $\log \pi(y, \theta)$ or its gradient $\nabla_\theta \log \pi(y, \theta)$, many times for many different values of θ. Calculation of the gradient with automatic differentiation requires us to evaluate the joint distribution, meaning the computation of derivatives is at least as expansive as the evaluation of the log joint density. Certain algorithms further require the evaluation of higher-order derivatives, at least for certain terms in the log joint density.

Consider the example of HMC: the gradient must be evaluated once per step of the leapfrog integrator (Algorithm 2.1). If we have an average discrete trajectory length L and a total of N warmup and sampling iterations, the number of gradient evaluations is LN. Plugging in some realistic numbers, for $L = 8$ and $N = 2 \times 10^3$, our algorithm requires 1.6×10^4 gradient evaluations! A computational bottleneck arises if each gradient evaluation is expansive. This is a challenge when large data imply expensive calculations of the log density. Another complication is that the gradient must be evaluated for different values of θ, which may drastically impact the computational

cost of differentiation. This is notably the case for likelihoods based on implicit functions, such as algebraic equations, ordinary differential equations (ODEs), and marginal densities.

Likelihoods based on implicit functions arise in models that integrate a scientific model inside a probabilistic model. While at times these implicit functions admit an analytical or semi-analytical expression (e.g. matrix solve for linear algebraic equations, matrix exponential solution for linear ODEs), many times we must resort to numerical integrators to obtain a solution. Efficient differentiation of implicit functions can be done by solving an augmented system, a topic which is extensively explored in Appendix A. This augmented system is usually much larger and introduces additional overhead. Careful implementation is required to avoid computational pitfalls. Anecdotally a first implementation of the disease transmission model by Hauser et al. 2020 took 3 days to fit. Rewriting the code to evaluate the log joint density in such a way to make automatic differentiation more efficient reduced the runtime to 2 hours; see Grinsztajn et al. (2021) for a detailed discussion. With all this said I here focus on the original system and the case of ODEs, noting that the augmented system inherits many properties of the original system.

A first-order ODE, with linear dependency on the derivative, is defined by the equality constrain

$$\begin{aligned} y' &= f(y, t, \theta), \\ y(t = 0) &= y_0(\theta), \end{aligned}$$

where

- $y \in \mathbb{R}^N$ is the (vector) solution we wish to evaluate at a desired time τ,
- $y' \in \mathbb{R}^N$ is the total derivative of y with respect to the time, t,
- $\theta \in \mathbb{R}^K$ are the model parameters,
- $y_0 \in \mathbb{R}^N$ is the initial condition.

We generally assume that f and y_0 can be evaluated.

Popular numerical integrators for ODEs include the Runge-Kutta $4^{\text{th}}/5^{\text{th}}$ order (RK45) solver (Dormand and Prince 1980; Karsten and Mulansky 2011), the backward differentiation (BDF) solver (Cohen and Hindmarsh 1996; Serban and Hindmarsh 2005) and the leapfrog integrator, which we have already encountered. Hindmarsh and Serban (2020) provide a comprehensive discussion on numerical integrators and their software implementation in the SUNDIALS C++ suite. Solvers present various trade-offs but they all rely on the fundamental concepts introduced by Euler's now outdated method (Algorithm 2.2), namely the use of a Taylor approximation (first-order in the case of Euler's method) of y in a neighborhood t,

$$y_{\text{e}}(t + \epsilon) := y(t) + \epsilon f(y(t), t, \theta), \tag{2.18}$$

The *local error* introduced by the approximation can be obtained by taking the difference between $y_{\text{e}}(t + \epsilon)$ and $y(t + \epsilon)$, and doing a Taylor expansion of the latter:

$$y_{\text{e}}(t + \epsilon) - y(t + \epsilon) = \frac{1}{2}\epsilon^2 y''(t) + O(\epsilon^3). \tag{2.19}$$

The error here is $O(\epsilon^2)$ and cumulates at each step. More recent integrators may have local error $O(\epsilon^t)$, with $t > 2$, and a global error which does not grow linearly with the number of steps. All the same, a small step size ϵ improves the approximation but at the cost of requiring more steps to go from the initial condition $t = 0$ to the desired time $t = \tau$. Usually the user does not specify ϵ, rather a tolerance for error. The algorithm then adaptively finds a step size which achieves the desired precision.

2.4.1 Example: planetary motion[2]

To illustrate how the behavior of an ODE varies with the model parameters, we consider the simple example of a planet in orbital motion around a star. The orbital dynamics of this system

[2]This section is based on the case study *Bayesian model of planetary motion: exploring ideas for a modeling workflow when dealing with ordinary differential equations and multimodality* (Margossian and Gelman 2020).

Algorithm 2.2: Euler's method

input: initial condition y_0, step size ϵ, time τ, derivative function f, model parameters θ.
$t \leftarrow 0$
$y \leftarrow y_0$
while $t < \tau$ **do**
    $t \leftarrow t + \epsilon$
    $y \leftarrow y + \epsilon f(y, t, \theta)$
end
return: y

are described by Hamilton's equations of motion (Equation 2.5) subject to a gravitational potential (Figure 2.7). The resulting ODE is

$$\begin{aligned} q' &= \frac{p}{m}, \\ p' &= -\frac{k}{r^3}(q - q_\odot), \end{aligned} \tag{2.20}$$

with

- $q(t)$: the planet's position over time,
- $p(t)$: the planet's momentum over time,
- $k = GMm$, with G the gravitational constant, M the star's mass, and m the planet's mass,
- $q_\odot$: the star's position, assumed to not vary over time.
- $r = \sqrt{(q - q_\odot)^T (q - q_\odot)}$ is the distance between the planet and the star.

Consider the model parameter k, which controls the strength of the gravitational interaction. As k increases, so do the magnitude of p' and higher-order derivatives of p. From Equation 2.19 we see that the error is $O(\epsilon^2 y''(t))$ when using Euler's method. In general, for a fixed ϵ, the error grows with the magnitude of higher-order derivatives of y. Given a target tolerance, larger higher-order derivatives must be compensated by smaller step sizes, requiring the numerical integrator to take more steps in order to go from $t = 0$ to $t = \tau$. Consequently the time required by a numerical integrator to solve the ODE depends on k!

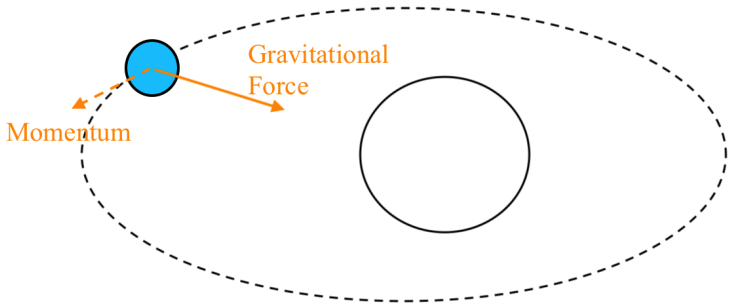

Figure 2.7: Orbital motion. *The gravitational potential, induced by an attractive force between the star and the planet, combined with a momentum orthogonal to the force, induces an orbital motion.*

Chain ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
time (s)	10.56	3.40	4433.93	181.98	3.50	10.74	3.59	10.42

Table 2.3: Run time for 8 Markov chains (500 warmup + 500 sampling iterations) on a planetary motion model. *The marginal posterior distribution of k is highly multimodal. At certain modes, the large value of k requires a smaller step size ϵ to accurately solve Equation 2.20.*

When estimating k using the observed positions of the planet with MCMC, we find that the runtime of different Markov chains varies wildly, with some chains taking less than 4 seconds to undergo 1,000 iterations, while others take minutes or hours (Table 2.3). This is because the posterior distribution is multimodal in k. To be precise, the probability mass concentrates around a single mode, however minor modes, while carrying a marginal probability mass, can still trap Markov chains. A Markov chain caught in a mode where k is large must solve a more difficult ODE and takes a longer time to run. For a more extensive discussion on this model, see Margossian and Gelman (2020).

2.4.2 Example: Michaelis-Menten pharmacokinetic model[3]

The required step size to achieve a wanted precision is not the only property of an ODE which can change across the parameter space. Another consideration is whether the ODE system is *stiff*, in which case numerical stability rather than precision requirements constrain the step length (Lambert 1992). Non-stiff integrators, such as RK45, need a very small step size to achieve numerical stability when used on a stiff system, resulting in slow computation. A stiff solver such as BDF is then more appropriate. On the other hand if the system is non-stiff, the RK45 integrator can be much faster. It can be difficult to know ahead of time whether or not a system of interest is stiff, and which integrator to use.

In a Bayesian context we furthermore need to account for the ways in which an ODE system changes across the parameter space. As a motivating example, consider a simplified Michaelis-Menten pharmacokinetic model,

$$\begin{aligned} y_1' &= -k_a y_1, \\ y_2' &= k_a y_1 - \frac{V_m C}{K_m + C}, \end{aligned} \tag{2.21}$$

with

- y_1 (mg): the drug mass in the Gut (the drug is administered orally),
- y_2 (mg): the drug mass in the central compartment (blood and tissues into which the drug diffuses quickly),
- k_a (h^{-1}): rate constant at which the drug flows from the gut to the central compartment
- $C = y_2/V$ (mg / L): the drug concentration in the central compartment, with V the volume of the central compartment,
- V_m (mg / h): maximum rate achieved by the system,

[3]This section is based on the conference poster, *Solving ODEs in a Bayesian context: challenges and opporunities* presented at the 2021 Population Approach Group in Europe conference; see Margossian et al. 2021.

- K_m (mg / L): Michaelis constant.

The patient receives a single dose, after which we measure the drug plasma concentration, c_{obs}, over time and estimate the model parameters $\theta = (k_a, V, V_m, K_m, \sigma)$. The full data generating process is

$$
\begin{aligned}
k_a &\sim \text{logNormal}(\log 2.5, 3), \\
V &\sim \text{logNormal}(\log 35, 0.5), \\
V_m &\sim \text{logNormal}(\log 10, 0.5), \\
K_m &\sim \text{logNormal}(\log 2.5, 3), \\
\sigma &\sim \text{Normal}^{+}(0, 1), \\
C &= y/V, \\
c_{\text{obs}} &\sim \text{Normal}(C, \sigma^2).
\end{aligned}
$$

In the above model the priors, notably for k_a and K_m, are weakly informative. We run 8 chains initialized with draws from the prior distribution.

Faced with this model, which numerical integrator should we use? Stan supports a non-stiff RK45 solver and a stiff BDF solver. It is also possible to analytically solve the first ODE in Equation 2.21 and plug in the result in the second ODE[4].

We run the model with both the RK45 and BDF integrators and compare the runtimes across 8 chains, taking care to distinguish the various warmup phases of HMC (Figure 2.8). From this experiment, we can make a few observations:

1. The runtime between chains does not vary much when using a BDF solver, in sharp contrast to what we see with the RK45 solver.

2. On average, the RK45 solver is faster. However when running chains in parallel, the slowest

[4]I do not explore this option in this experiment and leave it to upcoming work. In this example, I do not expect any significant speed up, although this approach has been useful when handling more sophisticated ODEs, such as the Friberg-Karlsson semi-mechanistic model (Table 2.1); see Margossian and Gillespie (2017).

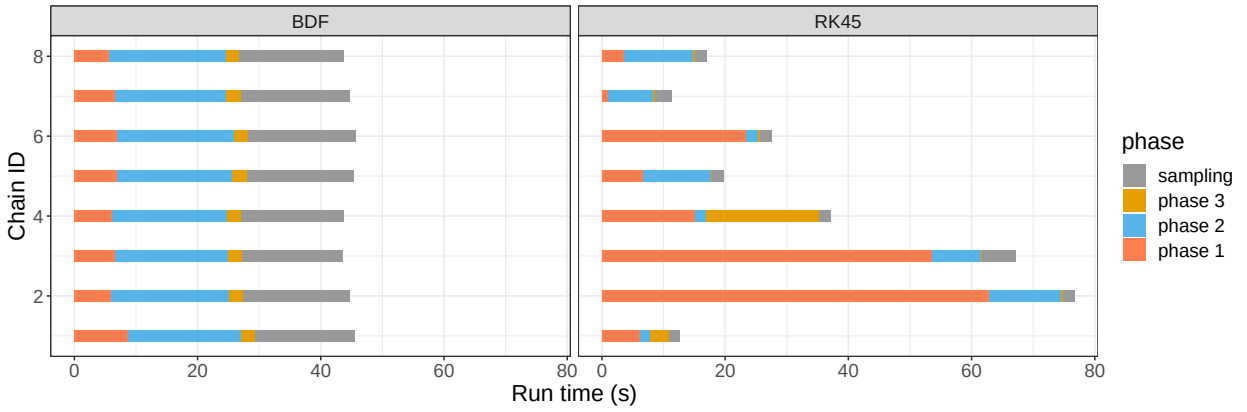

Figure 2.8: Runtime over 8 chains when fitting a pharmacokinetic model with HMC using (left) a BDF solver or (right) an RK45 solver.

chain determines the computation time. With this consideration in mind, the BDF solver is faster.

3. The computation in RK45 is largely dominated by the warmup phases. The sampling phase is much shorter than when using a BDF solver.

To more formally measure the performance of MCMC using either an RK45 or a BDF integrator, we check that each method returns consistent estimates. We then calculate the average time required to increase the ESS by 1 for the log joint distribution, $\log p(\theta, c_{\text{obs}})$ (Figure 2.10). While the joint distribution is not necessarily a quantity of interest, it is a function which depends on all the model parameters. Focusing on one scalar allows for straightforward comparisons, even as the number of parameters increases. Our measure may be termed the average *relaxation time*. Note that our calculations account for the warmup time, meaning

$$\text{Relaxation time} := \frac{\text{Warmup time} + \text{Sampling time}}{\text{ESS}}. \tag{2.22}$$

This definition should be contrasted with calculations of the time required to increase the ESS from 0 to 1, or the time required to increase the ESS by 1 after convergence is achieved.

Results on using the RK45 or the BDF solvers motivate a strategy whereby we use a different

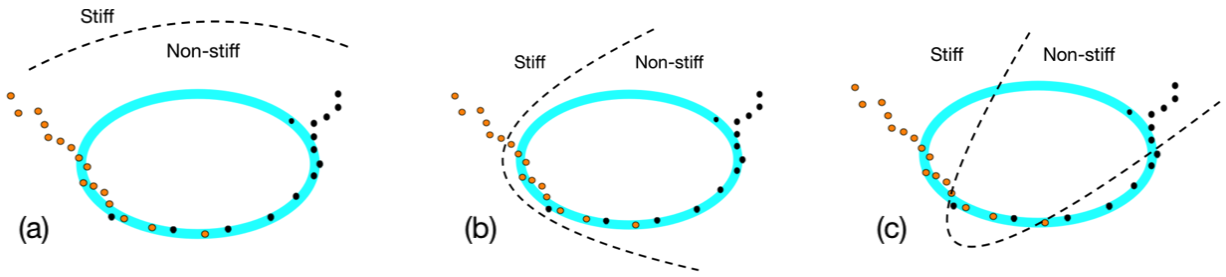

Figure 2.9: Behavior of ODE across the parameter space. *As the Markov chains (orange and black dots) move across the parameter space, the behavior of the ODE may change. The elliptical blue band represents the sampling region, i.e. the region where the posterior probability mass concentrates. Three hypothetical cases for how the ODE may behave: (a) a non-stiff integrator can be used for all phases of MCMC, (b) a stiff integrator may be suited to initialize the chain before switching to a non-stiff solver, (c) no optimal solver can be used across the sampling region. Another consideration is that different chains, depending on their respective positions, may deal with different ODE behaviors, resulting in varying runtimes.*

integrator for different phases. Our reasoning is that the parameter values, θ, vary more wildly at the start of the MCMC process, when the chains have not converged to the sampling region and/or the tuning parameters are not well adapted. In these early regions, the parameters may take more extreme values, requiring a more stable integrator. If the ODE in our model is stiff only in the "outer" regions of the parameter space, then using a stiff integrator at first and switching to a non-stiff integrator can lead to more efficient computation. Naturally there are many possible scenarios for the behavior of an ODE across the parameter space (Figure 2.9).

Table 2.4 presents alternatives to uniformly using one integrator when running dynamic HMC. We also consider using a prototype pathfinder to find good initializations and adapt the mass matrix before running HMC (Zhang et al. 2021). Here too we may consider using a BDF or an RK45 integrator at various phases of the algorithm. Figure 2.10 plots the average relaxation time for all methods, when applied to our pharmacokinetic model and a population extension with 3 patients, using a hierarchical prior. The results of our numerical experiments may be summarized as follows:

1. For the single patient model, using BDF during the warmup phase improves the stability of the sampler. Improved initializations do not warrant the additional cost of running the

	Pathfinding	Phase I	Phase II	Phase III	Sampling
HMC warmup, early switch	NA	BDF	RK45	RK45	RK45
HMC warmup, late switch	NA	BDF	BDF	RK45	RK45
Pathfinder, RK45	BDF	RK45	RK45	RK45	RK45
Pathfinder, late switch	BDF	BDF	BDF	RK45	RK45
Pathfinder, approx. tuning	BDF	NA	NA	RK45	RK45

Table 2.4: Combining different integrators. *A stiff integrator may be necessary during the warmup, but overkill when sampling. Applying this heuristic, we propose several schemes.*

pathfinder[5]; however estimating tuning parameters based on the variational approximation produces the most efficient sampler.

2. For the population model, we suspect the additional data stabilizes the posterior distribution, making uniform RK45 the best option. In this scenario, running the pathfinder is expansive and the estimated tuning parameter is suboptimal (we can diagnose this with the Pareto smooth importance sampling (Zhang et al. 2021)).

2.4.3 Concluding remarks

Picking which integrator to use when fitting a Bayesian model is a delicate problem. This tuning problem becomes more difficult when we contemplate switching integrators between various phases of an algorithm. Unfortunately there is in general no way to anticipate how an ODE might behave along the path taken by a Markov chain. What is more, if we run many chains in parallel, we are more likely to encounter a parameter configurations for which an ODE is difficult to integrate. This leads to the common case where at least one chain lags behind. Strategies which run many chains in parallel are particularly vulnerable to this phenomenon.

How to best tune ODE integrators in a Bayesian context remains an open question. In practice preventing Markov chains from exploring regions where parameters take on absurd values, through

[5]The here presented prototype is written in R; a new prototype written in C++ may yield better performance.

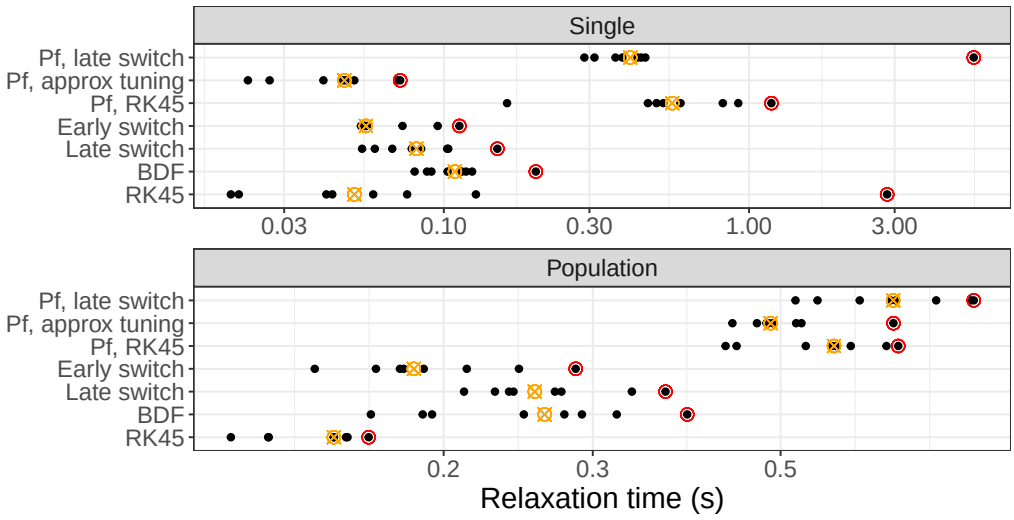

Figure 2.10: Performance of algorithms when switching integrators during various phases across 8 chains. *Relaxation time, i.e. time to increase the effective sample size by 1, measured for* $\log p(\theta, c_{obs})$*. The orange crossed dot is the median time, and the red circled dot the worst time. For each method which run 8 or more chains in parallel, we may prefer consistency to good median performance.*

careful initialization and the use of informative priors when these are available, can be a viable option.

2.5 Acknowledgment

I thank Andrew Gelman for comments on a draft of this chapter.

Chapter 3: Nested $\hat{R}$: assessing the convergence of Markov chains Monte Carlo when running many short chains

CHARLES C. MARGOSSIAN[1], MATTHEW D. HOFFMAN[2], AND PAVEL SOUNTSOV[2]

When using Markov chain Monte Carlo (MCMC) algorithms, we can increase the number of samples either by running longer chains or by running more chains. Practitioners often prefer the first approach because chains need an initial "warmup" phase to forget their initial states; the number of operations needed for warmup is constant with respect to chain length but increases linearly with the number of chains. However, highly parallel hardware accelerators such as GPUs may allow us to run many chains in parallel almost as quickly as a single chain. This makes it more attractive to run many chains with a short sampling phase. Unfortunately, existing diagnostics are not designed for the "many short chains" regime. This is notably the case for the popular $\hat{R}$ statistic which claims convergence only if the effective sample size *per chain* is sufficiently large. We present $\mathfrak{n}\hat{R}$, a generalization of $\hat{R}$, which does not conflate short chains and poor mixing, and offers a useful diagnostic provided we run enough chains and meet certain initialization conditions. We define what constitutes a proper warmup in the many-chains regime and recommend a threshold for $\mathfrak{n}\hat{R}$. Furthermore we use $\mathfrak{n}\hat{R}$ to construct a warmup scheme with an adaptive length, allowing users to avoid running their MCMC algorithms longer than necessary.

3.1 Introduction

The growing availability of parallel computation on CPUs, GPUs, and even specialized chips such as TPUs, has generated interest in MCMC strategies where we run many chains in parallel, using a short sampling phase (Lao et al. 2020; Hoffman and Ma 2020). New MCMC algorithms,

[1]Columbia University, Department of Statistics; work mostly done while an intern at Google Research.
[2]Google Research

such as ChEES-HMC and SNAPER-HMC are particularly well adapted to run many chains, leveraging the parallelization capacities of accelerators (Hoffman, Radul, and Sountsov 2021; Sountsov and Hoffman 2021). For many Bayesian inference applications, an effective sample size (ESS) of 100 may be adequate, as suggested by the default diagnostics in the statistical software Stan (Carpenter et al. 2017). In that case, we may want to run as many as ∼100 chains. Indeed, if the chains are independent and stationary – in other words, they have been properly warmed up –, then the number of chains lower-bounds the ESS, provided the target distribution has a finite variance. This lower bound is attained in the limit where each chain has one iteration. We note that, in our experience, the target ESS may vary between applications, at times being as low as 10 or as high as 500. Per the above reasoning, the target ESS informs how much parallelization is useful.

Existing diagnostics are not designed for the regime where we have many chains each made up of only a few iterations. This is notably the case for the widely used $\hat{R}$ statistic (e.g. Gelman and Rubin 1992; Vehtari et al. 2020). $\hat{R}$ checks for consistency between chains and converges to 1 if the chains produce Monte Carlo estimators in sufficiently good agreement. As we shall see, convergence to 1 however requires the measured ESS *per chain* to be large, a condition which is incompatible with running many short chains; if diagnosing convergence requires us to run long chains, we cannot reap the benefits of running many short chains in parallel.

We present a generalization of $\hat{R}$, called nested $\hat{R}$, and denoted $\mathfrak{n}\hat{R}$, which checks for consistency between groups of chains. $\mathfrak{n}\hat{R}$ is a useful diagnostic for the many-short-chains regime and works under conditions similar to those that the classic $\hat{R}$ assumes. The main additional requirement is to initialize each chain within a group at the same point. Because properly warmed-up MCMC forgets its starting point, this requirement does not affect the quality of our estimates; it does, however, help us diagnose convergence failures. While $\hat{R}$ converges to 1 only when the number of post-warmup samples is large, $\mathfrak{n}\hat{R}$ can go to 1 (once the chains are properly warmed up) when either the number of iterations or the number of chains is large. Hence for a sufficiently large number of chains we can diagnose convergence with only a few sampling iterations.

If we use a short sampling scheme, the computation of MCMC is dominated by the warmup

phase. To reduce this computational burden, we propose an adaptive warmup length scheme, wherein we compute $\mathfrak{n}\hat{R}$ using a few proposal samples after a short warmup window, and repeat until convergence is diagnosed.

3.2 Three perspectives on $\hat{R}$

Given a random variable, $\theta \in \Theta$, with a target distribution π, and a scalar function of interest, $f : \Theta \to \mathbb{R}$, our goal is to estimate the expectation value $\mathbb{E}_\pi f$, where we assume $\mathrm{Var}_\pi f$ is finite. We call a *Monte Carlo estimator* any statistic, T, intended to estimate $\mathbb{E}_\pi f$. This definition encompasses the traditional Monte Carlo estimator used when doing MCMC sampling but it is deliberately broader. Let Γ be the probability measure which generates T. Any Monte Carlo estimator admits an *effective sample size* (ESS):

$$\mathrm{ESS} \triangleq \frac{\mathrm{Var}_\pi f}{\mathrm{Var}_\Gamma T}. \tag{3.1}$$

Consider an MCMC sampler with stationary distribution π, which generates M chains, each with N iterations. Each chain is initialized at a point $\theta^{(0m)}$. We denote the nth sample from chain m as $\theta^{(nm)}$. It is common practice to first "warm up" the sampler for some number of iterations. During the warmup phase, the Markov chains travel to the region where the probability mass concentrates and begin exploring this region (Betancourt 2018a); early exploration also allows us to tune the parameters of our sampler (e.g Hoffman and Gelman 2014; Hoffman, Radul, and Sountsov 2021). Once the chain is warmed up, we generate the samples we use for our inference. For ease of notation, we consider estimation of $\mathbb{E}_\pi \theta$ and note all our calculations generalize to functions of θ with a finite variance.

The classic MCMC estimator is:

$$\bar{\theta}^{(\cdot\cdot)} = \frac{1}{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} \theta^{(nm)}. \tag{3.2}$$

A useful perspective is to view this as an average of M Monte Carlo estimators,

$$\bar{\theta}^{(\cdot m)} = \frac{1}{N} \sum_{n=1}^{N} \theta^{(nm)}, \tag{3.3}$$

each based on a single chain.

The $\hat{R}$ statistic, in its first and simplest iteration as the *potential scale reduction factor* (Gelman and Rubin 1992), compares a measure of the total variance across all chains, $A = \mathrm{Var}_\Gamma \theta^{(nm)}$, to a measure of within-chain variance, $W = \mathbb{E}_\Gamma \left[\mathrm{Var}_\Gamma (\theta^{(nm)} \mid m = m^*) \right]$. In words, W is the expected variance given that the samples are all in the same chain. We estimate these quantities using the sample variance estimators

$$\hat{W} = \frac{1}{M} \sum_{m=1}^{M} \frac{1}{N-1} \sum_{n=1}^{N} \left(\theta^{(nm)} - \bar{\theta}^{(\cdot m)} \right)^2 \tag{3.4}$$

and

$$\begin{aligned} \hat{A} &= \hat{W} + \frac{1}{M-1} \sum_{m=1}^{M} \left(\bar{\theta}^{(\cdot m)} - \bar{\theta}^{(\cdot\cdot)} \right)^2 \\ &\triangleq \hat{W} + \hat{B}. \end{aligned} \tag{3.5}$$

The above expression is motivated by the law of total variance[3]. $\hat{B}$ measures the between-chain variance, $B = \mathrm{Var}_\Gamma \bar{\theta}^{(\cdot m)}$. We then define

$$\hat{R} \triangleq \sqrt{\hat{A}/\hat{W}}. \tag{3.6}$$

If the chains are mixing, we expect the total variance and the within-chain variance to converge in probability to $\mathrm{Var}_\pi \theta$ as N grows, meaning $\hat{R}$ converges to 1. A common practice to assess convergence of the chains is to check that $\hat{R} \leq 1 + \epsilon$, for some $\epsilon > 0$. The choice of ϵ varies

[3]The original $\hat{R}$ uses a slightly different estimator for the within-chain variance when computing $\hat{A}$. There $\hat{W}$ is scaled by $1/N$, rather than $1/(N-1)$. This explains why occasionally $\hat{R} < 1$. This is of little concern when N is large, but we care about the case where N is small, and therefore adjust the $\hat{R}$ statistic slightly.

across in the literature, and has evolved over time, starting at $\epsilon = 0.1$ and recently using the more conservative value $\epsilon = 0.01$ (Vats and Knudson 2021; Vehtari et al. 2020).

A second perspective on $\hat{R}$ is due to Vats and Knudson (2021) who observe that when the chains are stationary $\hat{W}/\hat{B}$ is a measure of the effective sample size *per chain*. Observing that

$$\hat{R} = \sqrt{\hat{A}/\hat{W}} = \sqrt{1 + \hat{B}/\hat{W}}, \tag{3.7}$$

we see that $\hat{R}$ decays to 1 only once we produce chains with a large effective sample size.

We now present a third and novel perspective, wherein we view $\hat{R}$ as a measure of agreement between the Monte Carlo estimators produced by each chain without assuming stationarity. Each chain is distinguished by its initialization point and its seed but $\mathbb{E}_\pi\theta$ is independent of both of these factors; if either influences our estimator too much then that estimator may not be accurate. Equation 3.7 shows that $\hat{R}$ converges to 1 if $\hat{B}$ is small relative to $\hat{W}$. For a small ϵ, such that $(1+\epsilon)^2 \approx 1 + 2\epsilon$, $\hat{R} \leq 1 + \epsilon$ is approximately equivalent to

$$\hat{B} \lesssim 2\epsilon\hat{W}. \tag{3.8}$$

This inequality establishes a tolerance value for $\hat{B}$, determined by $\hat{W}$ and ϵ. From the above formula, we can make a few observations.

First, for small N, the variance of the per-chain Monte Carlo estimator, $\theta^{(\cdot m)}$, stays relatively large, meaning $\hat{B}$ will not be small even if $\mathrm{Var}_\Gamma \bar{\theta}^{(\cdot\cdot)}$ is. We demonstrate this on the two-dimensional "Banana" distribution,

$$\begin{aligned} \theta_1 &\sim \text{Normal}(0, 10); \\ \theta_2 \mid \theta_1 &\sim \text{Normal}(0.03(\theta_1^2 - 100), 1). \end{aligned} \tag{3.9}$$

This is a simple transformation of a two-dimensional normal with highly non-convex level sets. After warming up 512 chains for 200 iterations, we only require 1 sampling iteration to achieve

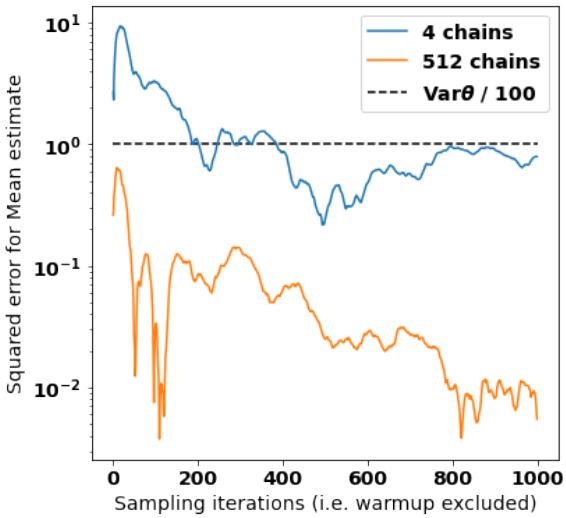

Figure 3.1: Accuracy of Monte Carlo for $\mathbb{E}\theta_1$ in Banana problem. *When using many chains, we only require a single sampling iteration per chain to reach the wanted precision, once the chains are warmed up.*

a squared error below $\text{Var}_\pi\theta/100$. Using 4 chains requires a sampling phase with ~200 - 400 iterations (Figure 3.1). Still $\hat{R}$ is indifferent to the number of chains we run (Figure 3.2).

Second, $\hat{R}$ focuses on the variance of the per-chain Monte Carlo estimator, not its squared error. $\hat{R}$ therefore cannot detect when all our Monte Carlo estimators suffer from the same bias. A canonical example is a multimodal target π, with all chains getting stuck with high probability in the same mode after N iterations. Changing the MCMC process, for instance by using overdispersed initializations can help ensure $\hat{B}$ stays large unless we achieve stationarity.

Third, in the case where $\text{Var}_\pi\theta = \infty$, $\hat{W}$ diverges and the tolerance bound in Equation 3.8 becomes meaningless. The inadequacy of $\hat{R}$ when the target distribution has non-finite variance is discussed by Vehtari et al. (2020), who address this problem using a rank-normalization scheme. We will explore this idea in Section 3.3.4.

3.3 Nested $\hat{R}$

The key idea behind $\text{n}\hat{R}$ is to compare Monte Carlo estimators whose variance decreases with both the number of iterations and the number of chains. A natural way to do this is to group chains

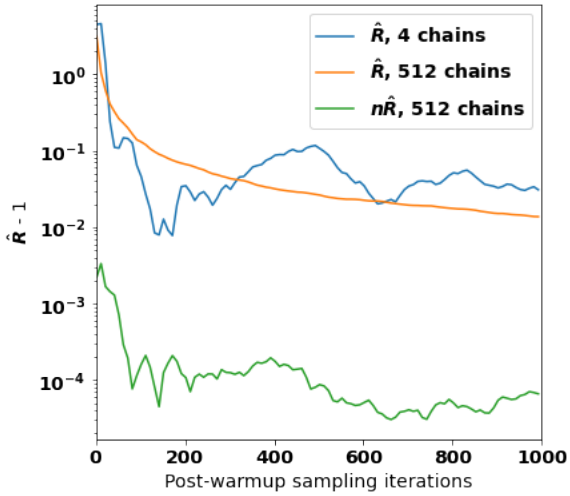

Figure 3.2: Applying $\hat{R}$ and $\mathfrak{n}\hat{R}$ to the first dimension of the Banana problem. *$\hat{R}$ is indifferent to the number of chains we run, despite the increased precision. $\mathfrak{n}\hat{R}$ on the other hand is sensitive to the number of chains.*

into *super chains*. Our MCMC process now has K super chains, each comprising M chains, and we denote $\theta^{(nmk)}$ the nth sample from chain m of super chain k. The Monte Carlo estimators compared by $\mathfrak{n}\hat{R}$ are the sample means of each super chain,

$$\bar{\theta}^{(..k)} = \frac{1}{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} \theta^{(nmk)}. \tag{3.10}$$

Provided we have enough chains per group, $\mathfrak{n}\hat{R}$ quickly goes to 1, reflecting the high precision of $\bar{\theta}^{(\cdot\cdot)}$ after a few iterations (Figure 3.2).

The new diagnostic follows the same formula as the original $\hat{R}$ except that $\hat{B}$ and $\hat{W}$ are now measures of the between and within-super-chain variances. Overloading our notation, for $\mathfrak{n}\hat{R}$ we define

$$\hat{B} = \frac{1}{K-1} \sum_{k=1}^{K} \left(\bar{\theta}^{(\cdot\cdot k)} - \bar{\theta}^{(\cdots)} \right)^2 \tag{3.11}$$

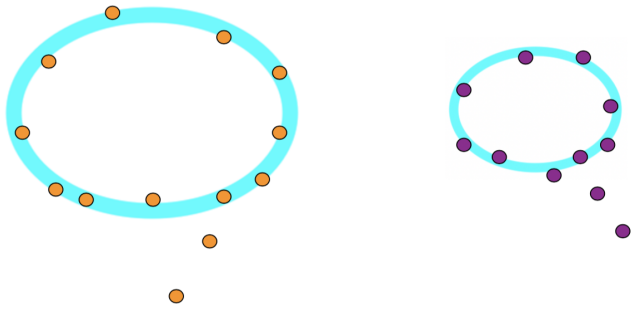

Figure 3.3: Bimodal target. *When faced with a multimodal target, most MCMC schemes commit to one mode and only transition to another mode with a low probability, resulting in poor mixing. Which mode a chain commits to is often determined by where the chain is initialized.*

and

$$\begin{aligned}\hat{W} &= \frac{1}{K}\sum_{k=1}^{K}\left[\frac{1}{M-1}\sum_{m=1}^{M}\left(\bar{\theta}^{(\cdot mk)} - \bar{\theta}^{(\cdot\cdot k)}\right)^2\right. \\ &+ \left.\frac{1}{M}\sum_{m=1}^{M}\frac{1}{N-1}\sum_{n=1}^{N}\left(\theta^{(nmk)} - \bar{\theta}^{(\cdot mk)}\right)^2\right]. \end{aligned} \tag{3.12}$$

In the special case where $N = 1$ we take the within-chain variance to be 0, meaning the second term in the above equation vanishes. Note this makes $\hat{W}$ smaller and results in a stricter tolerance on $\hat{B}$. Similarly when $M = 1$ we take the between chain variance to be 0, thereby removing the first term when calculating $\hat{W}$. $\text{n}\hat{R}$ then reduces to $\hat{R}$. We interpret the classical $\hat{R}$ setting as the special case where we run one chain per super chain.

3.3.1 Limitations when using independent chains

Unfortunately $\text{n}\hat{R}$ as defined above is too optimistic, as illustrated by the following thought experiment. Consider a bimodal target distribution

$$\pi = 0.3\ \text{Normal}(-10, 1) + 0.7\ \text{Normal}(10, 1),$$

and suppose each chain is either initialized at -10 or 10 with equal probability. Typical MCMC chains will not be able to mix across the high energy barrier between the modes. As a result, which mode a chain explores depends strongly on initialization, and MCMC incorrectly ascribes (on average) probability mass 0.5 to each mode (Figure 3.3). The per-chain Monte Carlo estimators are inconsistent, since they estimate the means at different modes. This allows $\hat{R}$ to diagnose the issue. On the other hand, the per-super-chain Monte Carlo estimators may look consistent and $\mathfrak{n}\hat{R}$ incorrectly claims convergence.

To see why, let $\bar{\theta}_N^{(.mk)}$ be the Monte Carlo estimator we obtain from chain m of super chain k after N iterations. Let $\sigma_N^2 = \mathrm{Var}\bar{\theta}_N^{(.mk)}$. We use a subscript on the variance because we do not assume stationarity. Because the chains are independent, we have

$$\mathrm{Var}\bar{\theta}^{(..k)} = \frac{1}{M}\sigma_N^2.$$

Regardless of whether the chain is stationary or not, this value becomes arbitrarily small for large M. This is not unexpected: we obtain good agreement between MCMC estimators so long as they are based on draws from the same distribution, whether or not it is the stationary distribution. One extreme case would be for $\mathfrak{n}\hat{R}$ to claim convergence after a single iteration because all the estimators are accurately computing expectation values with respect to the initialization distribution π_0.

3.3.2 Initialization requirement for the chains

To remedy the above issue, we impose the constraint that all chains within a super chain are initialized at the same point, θ_0^k. This allows us to track the influence of the initial point. In our thought experiment, the initial point determines which mode each chain explores. Our proposed initialization scheme therefore produces super chains in disagreement with one another, meaning we can now identify with $\mathfrak{n}\hat{R}$ that our MCMC estimates are unreliable.

We can understand the behavior of $\mathfrak{n}\hat{R}$ under this initialization condition via a variance decom-

position:

$$\text{Var}\bar{\theta}^{(..k)} = \mathbb{E}\left[\text{Var}\left(\bar{\theta}^{(..k)} \mid \theta_0^k\right)\right] + \text{Var}\left[\mathbb{E}\left(\bar{\theta}^{(..k)} \mid \theta_0^k\right)\right]. \tag{3.13}$$

We denote $B := \text{Var}\bar{\theta}^{(..k)}$, $B_e := \mathbb{E}[\text{Var}(\bar{\theta}^{(\cdot\cdot k)} \mid \theta_0^k)]$, and $B_v := \text{Var}[\mathbb{E}(\bar{\theta}^{(\cdot\cdot k)} \mid \theta_0^k)]$. The above equation is in this notation

$$B = B_e + B_v. \tag{3.14}$$

Conditional on θ_0^k, the chains are independent. Thus B_e goes to 0 either as M increases or, once we have reached stationarity, as N increases. B_v, on the other hand, is indifferent to M. For it to decay, the expected value of $\bar{\theta}^{(..k)}$ must become independent of θ_0^k. To use a common phrase, the chains must "forget" where they started. We will further refine this decomposition in Section 3.3.3.

What we have with our proposed initialization scheme is a useful compromise between $\hat{R}$ and $\mathfrak{n}\hat{R}$ applied to independent chains. We term the latter the *naive* $\mathfrak{n}\hat{R}$, and save the original term, $\mathfrak{n}\hat{R}$, devoid of any prefix, for the case where we use one intial point per super chain. $\mathfrak{n}\hat{R}$ first behaves like $\hat{R}$ and, as the correlation to the starting point decays, switches to behaving like the naive $\mathfrak{n}\hat{R}$ if the chains converge. We demonstrate this transient behavior (or lack thereof) on two examples: (i) a Gaussian distribution where the chains mix and the transient behavior occurs; (ii) a mixture of two Gaussian distributions where the chains fail to mix and $\mathfrak{n}\hat{R}$ does not transition to the naive behavior (Figure 3.4).

Returning to our thought experiment in Section 3.3.1, we may wander about the case where the target π is multimodal and which mode each chain commits to is *not* determined by the initial point (Figure 3.5). That is the seed of the chain determines the mode where the chain drifts. It then becomes impossible to distinguish the samples generated by different super chains, especially as M grows, even though the chains are not mixing. This calls for yet another revision to our MCMC scheme, whereby we insure that all chains within a super chain commit to the same mode. To be clear, from an inferential perspective, there is no benefit to restricting multiple chains to explore the same mode. Rather our assumption here is that reliable inference is not possible, and we want to make sure that $\mathfrak{n}\hat{R}$ diagnoses MCMC's failure.

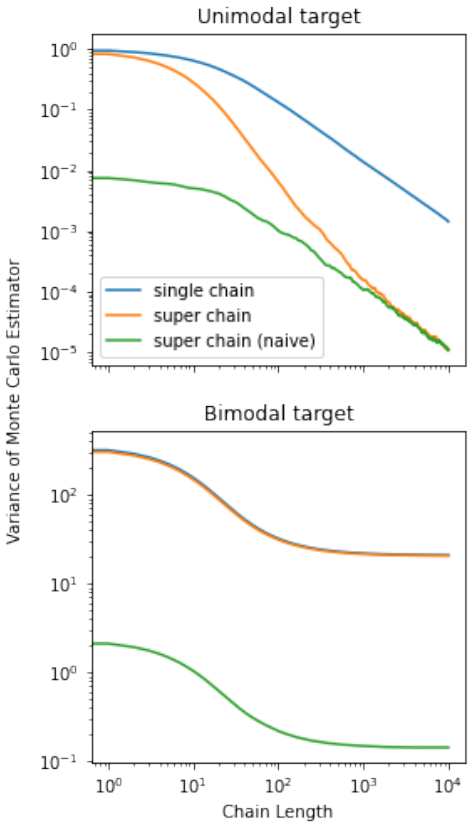

Figure 3.4: Sample variance of Monte Carlo estimators. *We compare estimators using a single chain, a naive super chain made of independent chains, and a super chain made of chains initialized at the same point.*

A natural way to achieve our goal is to run K *initializing chains* for I iterations and then split each chain into M chains with a different seed. For a sufficiently large I, all M chains start within the orbit of the same mode. We fall back into the safe case where the initialization determines which mode a super chain samples from. Which value of I to use is unclear. An alternative would be to use the pathfinder by Zhang et al. (2021) to generate initial points for each super chain in the orbit of a mode.

With all this said, we have yet to encounter an application where the seed alone determines which mode a chain converges to. This seems to require a particularly unfortunate initial distribution, π_0, with respect to the target π. For now we resort to the simpler initialization scheme where all chains in a super chain are initialized at the same point. We leave to future work the investigation of other initialization strategies.

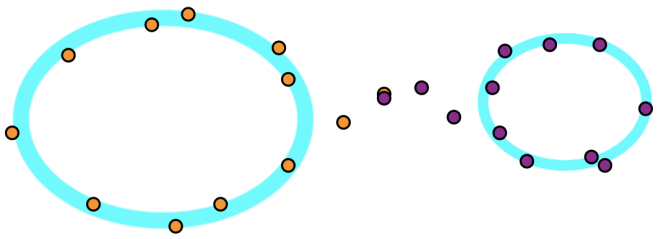

Figure 3.5: Bimodal target. *Two chains initialized at the same point drift to a different mode.*

3.3.3 Violation of stationarity and threshold for $\mathfrak{n}\hat{R}$

An important tuning parameter for $\mathfrak{n}\hat{R}$ is the threshold ϵ, which we use to determine if the chain is approximatively stationary. The condition $\mathfrak{n}\hat{R} \leq 1 + \epsilon$ is equivalent to

$$\frac{\hat{B}}{\hat{W}} \leq \delta, \tag{3.15}$$

where $\delta = 2\epsilon + \epsilon^2 \approx 2\epsilon$. To simplify our calculations, assume each chain only contains one iteration, that is $N = 1$, which is the most interesting case in the many-short-chains regime. Denote $\mathcal{W}$ the number of warmup iterations. Until stationarity is achieved, the distribution of θ depends on $\mathcal{W}$.

Definition 3.1. *We say an estimator is L-consistent if it converges in probability to its estimand as $L \to \infty$. Furthermore an (L, J)-consistent estimator converges in probability to its estimand if $L \to \infty$ and $J \to \infty$.*

Assuming θ admits a second moment, $\hat{B}$ is a K-consistent estimator of

$$B = \mathrm{Var}\bar{\theta}^{(\cdot\cdot k)}_{\mathcal{W}}, \tag{3.16}$$

and $\hat{W}$ is a (K, M)-consistent estimator of

$$W = \mathbb{E}\left(\mathrm{Var}\left(\bar{\theta}^{(\cdot mk)}_{\mathcal{W}} \mid \theta^k_0\right)\right). \tag{3.17}$$

The behavior of B/W, which corresponds to what $\hat{B}/\hat{W}$ measures, may be understood via the following Lemma.

Lemma 3.1. *Suppose $N = 1$. Then*

$$\frac{B}{W} = \frac{1}{M} + \frac{Var\left(\mathbb{E}\left(\bar{\theta}_{\mathcal{W}}^{(\cdot\cdot k)} \mid \theta_0^k\right)\right)}{\mathbb{E}Var\left(\bar{\theta}_{\mathcal{W}}^{(\cdot mk)} \mid \theta_0^k\right)}, \tag{3.18}$$

and the condition $B/W \leq \delta$ is equivalent to

$$Var\left(\mathbb{E}\left(\bar{\theta}_{\mathcal{W}}^{(\cdot\cdot k)} \mid \theta_0^k\right)\right) \leq \left(\delta - \frac{1}{M}\right)\mathbb{E}Var\left(\bar{\theta}_{\mathcal{W}}^{(\cdot mk)} \mid \theta_0^k\right). \tag{3.19}$$

Proof. Given that the chains are independent, conditional on θ_0^k, we have

$$\mathrm{Var}\left(\bar{\theta}_{\mathcal{W}}^{(\cdot\cdot k)} \mid \theta_0^k\right) = \frac{1}{M}\mathrm{Var}\left(\bar{\theta}_{\mathcal{W}}^{(\cdot mk)} \mid \theta_0^k\right).$$

Thus

$$\begin{aligned}
\frac{B}{W} &= \frac{\mathrm{Var}\bar{\theta}_{\mathcal{W}}^{(\cdot\cdot k)}}{M\mathbb{E}\mathrm{Var}\left(\bar{\theta}_{\mathcal{W}}^{(\cdot\cdot k)} \mid \theta_0^k\right)} \\
&= \frac{\mathbb{E}\mathrm{Var}\left(\bar{\theta}_{\mathcal{W}}^{(\cdot\cdot k)} \mid \theta_0^k\right) + \mathrm{Var}\left(\mathbb{E}(\bar{\theta}_{\mathcal{W}}^{(\cdot\cdot k)} \mid \theta_0^k)\right)}{M\mathbb{E}\mathrm{Var}\left(\bar{\theta}_{\mathcal{W}}^{(\cdot\cdot k)} \mid \theta_0^k\right)} \\
&= \frac{1}{M} + \frac{\mathrm{Var}(\mathbb{E}\left(\bar{\theta}_{\mathcal{W}}^{(\cdot\cdot k)|\theta_0^k)}\right)}{\mathbb{E}\mathrm{Var}\left(\bar{\theta}_{\mathcal{W}}^{(\cdot mk)} \mid \theta_0^k\right)}.
\end{aligned}$$

Equation 3.19 follows from plugging the above in $B/W \leq \delta$ and rearranging the terms. □

Equation 3.18 decomposes B/W into a stationary and a non-stationary term. The first term, $1/M$, is indifferent to $\mathcal{W}$, and at stationarity corresponds to the inverse effective sample size per super chain. This is roughly equivalent to the term recovered by Vats and Knudson (2021), who analyze $\hat{R}$ under the assumption of stationarity. The second term emerges once we drop the

assumption of stationarity and can thus be interpreted as our chains' *violation of stationarity*. This is the quantity we want to control. If we are mindful of the correction incurred by the stationary term, we can pick ϵ to set a tolerance on the violation. Equation 3.18 warns us that B/W is lower-bounded by $1/M$, which may make it difficult to achieve a certain tolerance criterion. More generally when $N > 1$, δ may not be a reasonably sharp upper bound on the violation of stationarity if the effective sample size per super chain is not sufficiently large. This motivates running a few samples when M is relatively small.

This analysis does not immediately give us a principled threshold value to use but it makes the tuning parameter ϵ more interpretable. In practice, we find using $\epsilon = 0.01$ works reasonably well, although our numerical experiments suggest sometimes using lower values. The above analysis comes with the caveat that $\hat{B}/\hat{W}$ is an estimate of B/W, meaning our results hold exactly only in an asymptotic sense along K and M (however N need not be large!). We discuss some steps to reduce the variance of $\hat{B}/\hat{W}$ in the next section.

3.3.4 Rank-normalization and analysis under normal approximation

Using a rank-normalization of the $\theta^{(i)}$'s allows us to make our diagnostic robust to the case where $\text{Var}\theta = \infty$ (Vehtari et al. 2020). Under the transformation and for a large sample size, our MCMC samples become approximatively normal, which enables further analysis of $\mathfrak{n}\hat{R}$.

Rank-normalization replaces each sample, $\theta^{(nmk)}$, by its rank across all samples, $r^{(nmk)}$, and then applies an inverse-normal transformation. This procedure is intended to avoid assumptions of normality and finds extensive use in a hypothesis testing framework (Friedman 1937; Fisher and Yates 1938; Blom 1958). The rank-normalized samples are obtained via

$$z^{(nmk)} = \Phi^{-1}\left(\frac{r^{(nmk)} - 3/8}{NMK + 1/4}\right), \tag{3.20}$$

where Φ^{-1} is the inverse-normal CDF. Blom (1958) proposes a fractional offset to address the edge cases posed by the first and last fractional rank, respectively 0 and 1, which go to $\pm\infty$ under the Φ^{-1}

transformation. As the total number of samples, $T = NMK$, grows, we have $P\left(r^{(t)} \leq r\right) - r \leq 1/T$ and thus

$$\lim_{T\to\infty} \frac{r^{(t)} - 3/8}{T + 1/4} \xrightarrow{d} \text{Uniform}(0, 1). \tag{3.21}$$

From this we deduce convergence in distribution of $z^{(t)}$ to a standard normal and an error which decreases with T (due to the non-linearity of Φ^{-1} the error is larger in the tails than near the mode).

Once again, we consider the useful edge case where $N = 1$ and moving forward, we assume that r has a normal stationary distribution. We can then work out an approximate distribution for $\mathfrak{n}\hat{R}$ under the assumption of stationarity. This allows us to lay the foundation for a hypothesis test and study how to pick K and M in a limiting case.

Lemma 3.2. *Suppose (i) the Markov chains are stationary, (ii) $N = 1$, meaning each chain has a single sample, and (iii) the stationary distribution is standard normal. Then*

$$\frac{\hat{B}}{\hat{W}} \sim \frac{1}{M} F_{K-1,K(M-1)}. \tag{3.22}$$

Proof. At stationarity, the chains have forgotten their initialization and are mutually independent. Since $N = 1$ and by assumption (iii), our samples are i.i.d standard normal. Denote our samples $z^{(mk)}$, where we drop the (n) index since $N = 1$. Let

$$\begin{aligned} S_p^2 &= \frac{1}{MK - K} \sum_{k=1}^{K} \sum_{m=1}^{M} \left(z^{(mk)} - \bar{z}^{(\cdot k)}\right)^2 \\ &= \frac{1}{K} \sum_{k=1}^{K} \frac{1}{M-1} \sum_{m=1}^{M} \left(z^{(mk)} - \bar{z}^{(\cdot k)}\right)^2, \end{aligned} \tag{3.23}$$

and note that $S_p^2 = \hat{W}$, in the special case where $N = 1$. Then following the argument by Casella and Berger (2002, chapter 11) on the ANOVA null, we have that

$$\frac{\sum_{k=1}^{K} M\left(\bar{z}^{(k\cdot)} - \bar{z}^{(\cdot\cdot)}\right)^2}{\hat{W}} \sim (K-1) F_{K-1,MK-K}. \tag{3.24}$$

Rearranging the terms,

$$\frac{\frac{1}{K-1}\sum_{k=1}^{K}\left(\bar{z}^{(k\cdot)}-\bar{z}^{(\cdot\cdot)}\right)^2}{\hat{W}}=\frac{\hat{B}}{\hat{W}}\sim\frac{1}{M}F_{K-1,MK-K}, \tag{3.25}$$

as desired. □

Corollary 3.3. *Assume conditions (i), (ii), and (iii) from Lemma 3.2. The variance of $\hat{B}/\hat{W}$ is then well defined if*

$$KM-4>M.$$

Proof. The variance of an F distribution with degrees of freedom, d_1 and d_2, is well-defined when $d_2 > 4$. In our context, this means $K(M-1) < 4$. Rearranging the terms we obtain the desired result. □

Towards a hypothesis test for convergence

Under the normal approximation, Lemma 3.2 gives us an F test statistic and a straightforward way to control the Type I error.

Remark 3.1. *Unlike in many applied contexts where one would use a hypothesis test, we are "rooting" for the null hypothesis, not the alternative. This means corrections, such as the Bonferroni correction, make our test less conservative.*

Remark 3.2. *To study the alternative hypothesis, we need to understand the behavior of $\hat{B}/\hat{W}$ before stationarity. While the samples may still be approximatively normal, they are no longer independent due to our initialization requirements (Section 3.3.2). This means we are violating certain assumptions of the ANOVA setup (albeit having the same null hypothesis).*

Unfortunately we cannot control the Type II error (incorrect claims we have converged) because the chains can be almost stationary. In fact finite chains are never stationary and we leave to a future analysis how close to stationarity the chains need to be in order to produce useful Monte Carlo estimators. This can then inform how to calibrate our hypothesis test.

Tuning K and M

An important tuning parameter for $\text{n}\hat{R}$ is, given a total number of chains, T, how many super chains should we use? Here we need to balance two considerations:

(i) the more initializations we have, the less likely we are to have all our chains suffer from the same bias and moreover the more likely we are to have overdispersed initializations.

(ii) we want to minimize the variance of $\hat{B}/\hat{W}$.

Finding formal conditions under which achieve overdispersed initializations is an important open problem. The variance of $\hat{B}/\hat{W}$ can be studied in the limiting case described in Lemma 3.2, i.e. the chains are stationary and the normal approximation of the rank normalized samples is good.

Lemma 3.4. *Assume conditions (i), (ii), and (iii) from Lemma 3.2 and furthermore that* $1 < M < T - 4$. *Then the solution to the constrained optimization problem*

$$K^*, M^* = \underset{K,M}{\text{argmin}} \, \text{Var}\frac{\hat{B}}{\hat{W}}(K, M), \text{ subject to } KM = T,$$

is the root of the third degree polynomial in M

$$\varphi(M) = \frac{2}{M-1} + \frac{1}{T-M} - \frac{2(T-3)}{(T-3)M - T} - \frac{(T-4)}{(T-4)M - T},$$

where K and M are allowed to be in $\mathbb{R}$.

The above root-finding problem admits an analytical solution, albeit a not particularly insightful one. The proof is straightforward, if algebraically tedious.

In practice, K and M are constrained to be integers and it is straightforward to solve this optimization problem numerically. Consider the case where $T = 128$, meaning we want to achieve an effective sample size of ~128. Then the optimal number of super-chains is $K = 2$. Using only two initialization points however seems overly optimistic, in light of our desire to use overdispersed initializations. Setting $K = 4$ only marginally increases the variance of $\hat{B}/\hat{W}$(Table 3.1) and is in

K	2	4	8	16
Scaled variance	1.00	1.33	2.28	4.27

Table 3.1: Change in the variance of $\hat{B}/\hat{W}$ as we deviate from the optimum at K = 2 when $KM = 128$.

line with current MCMC practices. Experimentally we find setting $K = 4$ to work well. A more refine analysis of how tune K and M requires a deeper discussion on how to construct overdispersed initializations.

3.3.5 Limitations of $\mathfrak{n}\hat{R}$

A drawback of $\mathfrak{n}\hat{R}$, relative to $\hat{R}$, is that our proposed diagnostic is more sensitive to poor initialization. If the initial distribution π_0 generates samples which share the same bias relative to the target distribution, then the per-super-chain Monte Carlo estimators may appear to be in good agreement even though the chains are still transient. In this scenario $\text{Var}\bar{\theta}^{(\cdots)}$ does not characterize well the squared error of our Monte Carlo estimator. This can occur if we run a short warmup phase or, somewhat equivalently, if the chains mix slowly. $\hat{R}$'s implicit requirement for long chains enforces long MCMC runs which increases our chances of overcoming the transient behavior. Furthermore, comparing different sections of a chain, as is done with *split* $\hat{R}$ (Gelman et al. 2013), can help identify transient behaviors. With short chains, this approach is not an option and we need additional analysis, e.g. examining traceplots during the warmup phase. Using overdispersed initialization is generally recommended for any MCMC scheme, a point that deserves further emphasis when using $\mathfrak{n}\hat{R}$.

A related limitation of $\mathfrak{n}\hat{R}$ is that it cannot detect when the chains have converged to a *perturbed* stationary distribution. This may happen if we are adapting the parameters of the MCMC kernel during warmup, and do not freeze the parameters for enough iterations before sampling. $\mathfrak{n}\hat{R}$ will detect that all of the chains are sampling from the same distribution, but not that it differs subtly from the stationary distribution. This can be avoided by freezing adaptation earlier, but in any case if we are averaging adaptation signals across many chains, then this undetected bias is likely to be small, since the influence of any one chain on the adaptation is diminished.

3.4 Adaptive warmup length

When using a short sampling phase, the computation is dominated by the warmup. Which warmup length to use depends on the specific distribution we sample from and the MCMC algorithm we use. Current practice amongst modelers is to prespecify the warmup length and then run a long sampling phase. Only then do we run various diagnostics and if needed adjust the warmup length. This practice means the warmup length is rarely optimal. In this section, we discuss how $\mathfrak{n}\hat{R}$ can be used to remedy this problem.

3.4.1 Existing method

Zhang et al. (2020) propose a cross-chain warmup scheme for Stan's dynamic Hamiltonian Monte Carlo (Betancourt 2018a; Hoffman and Gelman 2014). This warmup strategy shares tuning parameters by pooling information between chains. Note that ChEES-HMC similarly shares information between chains to tune the sampler. Additionally, rather than run a warmup phase with a fixed length, the cross-chain warmup uses multiple *warmup windows* of length w. Stan's default warmup length is 1,000 iterations; the proposed window size $w = 100$ or 200. At the end of each window, we compute $\hat{R}$, as well as the bulk and tail ESS (Vehtari et al. 2020) for the unnormalized log target density using samples from the most recent warmup window. If $\hat{R} < \hat{R}^*$ and $\text{ESS} > \text{ESS}^*$, for a prespecified $\hat{R}^*$ and ESS^*, we end the warmup phase. We would like to build on this promising approach.

Our main concern is that the values we should use for $\hat{R}^*$ and ESS^* depend on both w and the specifics of our target distribution. How to pick the "right" threshold remains an open question, even if we can make a sensible guess. $\mathfrak{n}\hat{R}$ lets us address this challenge.

3.4.2 Revision using $\mathfrak{n}\hat{R}$

We propose the following modification to the adpative warmup scheme. At the end of each warmup window, we generate 5 sampling iterations and compute $\mathfrak{n}\hat{R}$ based on those samples. If

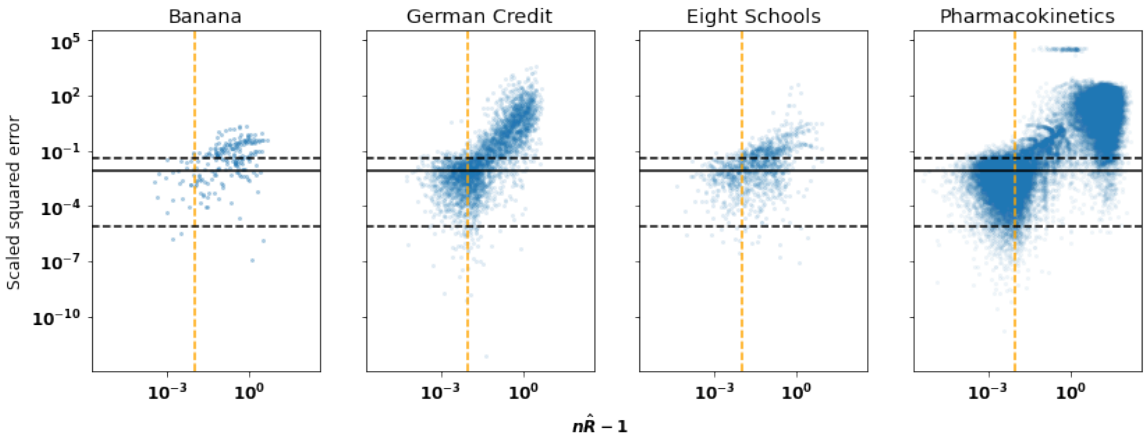

Figure 3.6: Scaled squared error against $\mathfrak{n}\hat{R}$. *The yellow line occurs at* $\epsilon = 0.01$. *The dotted lines form a* 0.95 *coverage for the distribution of the squared error at stationarity and the solid line is the expectation value (Equation 3.26).*

$w \approx 100$, computing these additional iterations is relatively cheap. This use of $\mathfrak{n}\hat{R}$ aligns with its natural intent and the window length, w, does not impact our diagnostic.

Additionally, rather than examine the unnormalized log target density, we compute $\mathfrak{n}\hat{R}$ for all quantities of interest. While it is convenient to describe how well the chains are mixing using a single scalar, convergence in one quantity may not indicate convergence in all quantities we may care about. We therefore adopt a more conservative approach but one that is more aligned with the modeler's goals.

Algorithm 3.1 summarizes the procedure. We denote the MCMC sampler Γ, which admits two arguments: ϕ, the tuning parameters of the sampler, and θ, the current state for all chains. If we draw warmup samples, both ϕ and θ are updated at each iteration. After each warmup window of length w, we use s samples to construct Monte Carlo estimators for all Q scalar functions of interest, denoted $f_1, f_2, \cdots, f_Q$. For each function, we compute $\mathfrak{n}\hat{R}$ and check that it is smaller than $1 + \epsilon$. If the number of chains is smaller than the target ESS, we can run the chains for more iterations after warmup until we achieve the desired ESS.

Algorithm 3.1: Adaptive warmup length

Input: Γ, π_0, ϕ_0; K, M ▷ Tuning parameters for MCMC
  $f_1, \cdots, f_Q$, w, s, ϵ, $T_{\max}$ ▷ Tuning parameters for adaptive warmup
for $k \in \{1, 2, \cdots, K\}$ **do**
  $\theta_0^k \sim \pi_0$
  Initialize all chains in k^{th} group at θ_0^k.
  Set tuning parameters, $\phi \leftarrow \phi_0$.
  Set chain states, $\theta \leftarrow \theta_0$.
end
for $t \in \{1, 2, \cdots, T_{max}\}$ **do**
  Run w warmup iterations using $\Gamma(\phi, \theta)$.
  Update ϕ and θ.
  Draw θ_{prop} for s sampling iterations using $\Gamma(\phi, \theta)$.
  $\Lambda \leftarrow$ TRUE ▷ Track convergence condition for all q's.
  for $q \in \{1, 2, \cdots, Q\}$ **do**
    Compute $\mathfrak{n}\hat{R}$ for $f_q(\theta_{\text{prop}})$.
    if $(\mathfrak{n}\hat{R} > 1 + \epsilon)$ **then**
      $\Lambda \leftarrow$ FALSE
      Break
    end
  end
  if $(\Lambda =$ TRUE$)$ **then**
    Terminate warmup.
  end
end
Return: θ_{prop}; $f_1(\theta_{\text{prop}})$, ..., $f_Q(\theta_{\text{prop}})$; ϕ.

3.5 Experiments

We study the behavior of $\mathfrak{n}\hat{R}$ on four target distributions, which we describe below.

Banana (D = 2). See Section 3.2, Equation 3.9.

Logistic regression (D = 25). A logistic regression model on the numerical version of the German credit dataset (Dua and Graff 2017). There are 24 features and an intercept term. The full model is

$$\theta \sim \text{Normal}(0, 1); \quad y_n \sim \text{Bernoulli}\left(\frac{1}{1 + e^{-\theta^T x_n}}\right).$$

Hierarchical model (D = 10). A model describing the effect of an SAT training program, as measured by the performance of students across 8 schools (Rubin 1981). We estimate the group mean and the population mean and variance. To avoid a funnel shaped posterior density, we use a non-centered parameterization:

$$\mu \sim \text{Normal}(5, 3); \sigma \sim \text{Normal}^+(0, 10); \eta_n \sim \text{Normal}(0, 1);$$
$$\theta_n, \sigma = \mu + \eta_n \sigma; \quad y_n = \text{Normal}(\theta_n, \sigma_n).$$

Pharmacokinetics (D = 205). A hierarchical model describing the diffusion of a drug in the body, using data simulated over 100 patients. We use a one-compartment model with first-order absorption from the gut, described by the differential equation:

$$m'_{\text{gut}} = -k_1 m_{\text{gut}}; \quad m'_{\text{cent}} = k_1 m_{\text{gut}} - k_2 m_{\text{cent}},$$

where m is the drug mass in the gut and the *central compartment* (blood and organs into which

the drug diffuses profusely). This differential equation admits an analytical solution. Each patient receives a drug dose at a regular interval. The drug plasma concentration is measured over time. Our measurement model is, for each patient indexed by n,

$$y_n(t) \sim \text{logNormal}(\log m_{\text{cent}}(t), \sigma).$$

For each patient, we estimate the transmission rates k_1^n and k_2^n. We use a hierarchical prior on (k_1^n, k_2^n) and estimate the population means and variances for both k_1 and k_2. The model is fitted to data sampled from the prior. Further details on the model can be found in the Supplementary Material.

We run Hamiltonian Monte Carlo, using the ChEES adaptive scheme (Hoffman, Radul, and Sountsov 2021) with 4 super chains, each composed of 32 chains, for a total of 128 chains. We compute the squared error of our Monte Carlo estimators, using long MCMC runs to calculate with high precision the correct posterior mean and variance. Invoking the Central Limit Theorem, we assume $\bar{\theta}^{(\cdots)}$ is approximately normally distributed for stationary Markov chains. Then for $N = 1$,

$$\frac{KM}{\text{Var}\theta}(\bar{\theta}^{(\cdots)} - \mathbb{E}\theta)^2 \overset{\text{approx.}}{\sim} \chi_1^2. \tag{3.26}$$

For each model, we compute $\mathfrak{n}\hat{R}$ using 5 samples after warmups of varying lengths

$$\ell = (10, 20, 30, ..., 100, 200, 300, ..., 1000).$$

The warmup lengths are not set ahead of time. Rather we use the adaptive warmup scheme in Section 3.4, starting with a small warmup window, $w = 10$, and later expanding this window to $w = 100$. The initial short windows have no practical use outside this experimental setting, where they help us monitor the behavior of $\mathfrak{n}\hat{R}$ when the squared error is large. Each adaptive warmup is repeated 10 times. At the end of each window, we record $\mathfrak{n}\hat{R}$ and the squared error of Monte Carlo estimators using a single sample per chain for each dimension. Figure 3.6 plots the squared

	$\mathfrak{n}\hat{R} \leq 1.01$	$\mathfrak{n}\hat{R} > 1.01$
Banana	0.080	0.593
German Credit	0.042	0.636
Eight Schools	0.053	0.466
Pharmacokinetics	0.036	0.812

Table 3.2: Fraction of estimators with a scaled error above the 97.5$^{\text{th}}$ quantile of a χ^2_1 distribution with $\mathfrak{n}\hat{R}$ either below or above 1.01. *If the chain is stationary, this number should be close to* 0.025.

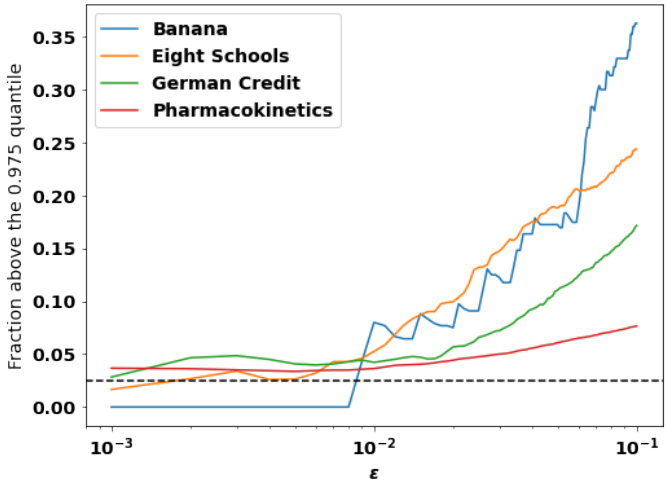

Figure 3.7: Fraction of estimators with a scaled error above the 97.5$^{\text{th}}$ quantile of a χ^2_1 distribution for varying values of ϵ. *The high fluctuation for the Banana problem is due to the relatively low number of points, since the problem is only two-dimensional.*

error, scaled by $KM/\text{Var}\theta$, against $\mathfrak{n}\hat{R}$. As desired, there is good correlation between $\mathfrak{n}\hat{R}$ and the squared error, and our diagnostic is consistently large over regions that admit an important squared error. When $\mathfrak{n}\hat{R} < 1.01$, the squared error mostly falls within the 0.95 coverage area of the χ^2_1 distribution. However the fraction of points which fall above the 97.5$^{\text{th}}$ quantile exceeds 0.025 (Table 3.2), which suggests our diagnostics could be more conservative. Figure 3.7 examines the benefits of using a more conservative threshold.

We now use the Eight School model as an illustrative example to highlight further analysis. Using a warmup with length 1,000 is amply sufficient, as can be checked by examing the squared error. In this scenario, $\mathfrak{n}\hat{R}$ falls consistently below the proposed threshold, bearing a few excep-

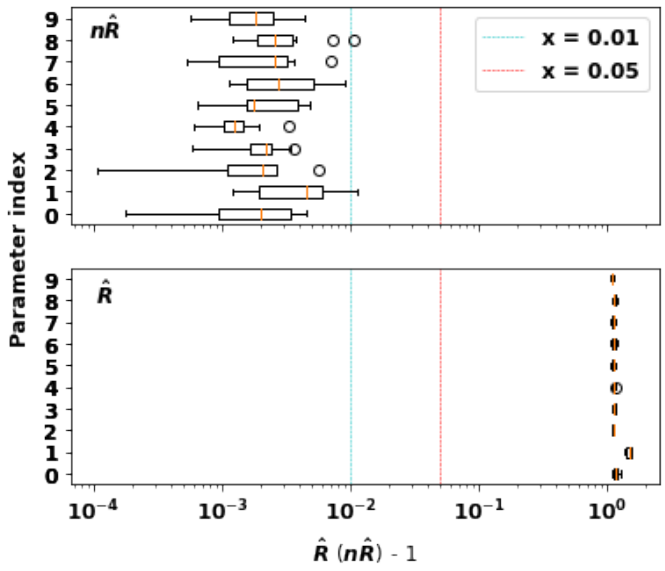

Figure 3.8: $\mathfrak{n}\hat{R}$ and $\hat{R}$ across all 10 parameters of the Eight Schools after a long warmup. *$\hat{R}$ is consistently above 2, while $\mathfrak{n}\hat{R}$ is below 1.01.*

tions, while $\hat{R}$ produces values larger than what would be considered acceptable (Figure 3.8).

We run the adapative warmup algorithm 50 times, using different seeds both for the initialization and the MCMC. Again, we examine the squared error when using one sample per chain and find it to agree reasonably well with what we would expect from independent draws from the stationary distribution. For this relatively simple problem, the warmup length varies between 100 and 1,000 iterations (Figure 3.9), likely because our initialization strongly influences how quickly the Markov chains reach the stationary distribution. There is no obvious correlation between the observed squared error and the length of the warmup, suggesting the algorithm does a good job of gauging when to stop.

3.6 Discussion

$\mathfrak{n}\hat{R}$ reproduces many of the desirable properties of $\hat{R}$ and can be applied to the regime where we run many chains with a short sampling phase. This makes it a useful tool to monitor MCMC warmup, without the requirement to run long chains. The utility of $\mathfrak{n}\hat{R}$ lies in its ability to check whether non-independent chains behave as though they were independent, per the notion that a

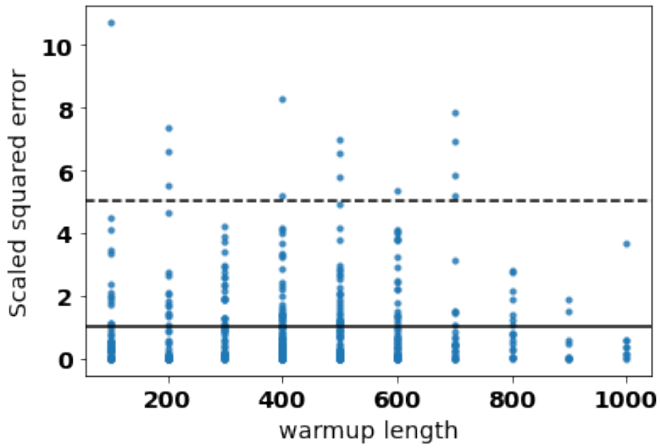

Figure 3.9: Squared Error for the Eight Schools over expected squared error across all dimensions. *The warmup length is not prespecified, but obtained using Algorithm 3.1. The solid line is the expected scaled squared error and the dotted line the* 97.5^{th} *quantile of the* χ^2_1 *distribution.*

proper warmup should "decouple" chains initialized at the same point. We may be able to relax this condition by starting chains in a super chain within the same region, rather than at the same point. The potential benefits of such an approach would be to cover more ground earlier during the warmup phase, which can improve tuning the sampler. We note that there exist many variations on $\hat{R}$ which can also motivate further improvements to $\mathfrak{n}\hat{R}$.

In the classic MCMC setting, the warmup phase needs to adapt the tuning parameters of the sampler to insure the Markov chains move efficiently across the parameter space during the sampling phase. That is we want to reduce the autocorrelation of our Markov chains, in order to generate chains with a large effective sample size. When one sample per chain suffices, autocorrelation is no longer a concern, which raises the possibility that our warmup phase can be less sophisticated and ultimately shorter. On the other hand, tuning the sampler well may be a prerequisite to achieve stationarity and obtain the unavoidable first sample.

3.7 Acknowledgment

We thank the entire TensorFlow Probability team at Google, especially Alexey Radul, as well as Aki Vehtari and Andrew Gelman for many helpful discussions. We thank Aki Vehtari, Rif A.

Saurous, Andrew Davison, and Owen Ward for their helpful comments on this manuscript.

3.8 Appendix (Supplementary Material): Pharmacokinetic model

The one-compartment pharmacokinetic model with first-order absorption from the gut describes the diffusion of a drug compound inside a patient's body. Oral administration of a bolus drug dose induces a discrete change in the drug mass inside the patient's gut. The drug is then absorbed into the *central compartment*, which represents the blood and organs into which the drug diffuses profusely. This diffusion process is described by the ordinary differential equation:

$$\begin{aligned} \frac{\mathrm{d}m_{\text{gut}}}{\mathrm{d}t} &= -k_1 m_{\text{gut}} \\ \frac{\mathrm{d}m_{\text{cent}}}{\mathrm{d}t} &= k_1 m_{\text{gut}} - k_2 m_{\text{cent}}, \end{aligned} \tag{3.27}$$

which admits the analytical solution, when $k_1 \neq k_2$,

$$\begin{aligned} m_{\text{gut}}(t) &= m^0_{\text{gut}} \exp(-k_1 t) \\ m_{\text{cent}}(t) &= \frac{\exp(-k_2 t)}{k_1 - k_2} \left(m^0_{\text{gut}} k_1 (1 - \exp[(k_2 - k_1)t] + (k_1 - k_2) m^0_{\text{cent}}) \right). \end{aligned} \tag{3.28}$$

Here m^0_{gut} and m^0_{cent} are the initial conditions at time $t = 0$.

A patient typically receives multiple doses. To model this, we solve the differential equations between dosing events, and then update the drug mass in each compartment, essentially resetting the boundary conditions before we resume solving the differential equations. In our example, this means adding m_{dose}, the drug mass administered by each dose, to $m_{\text{gut}}(t)$ at the time of the dosing event. We denote x the dosing schedule.

Our measurement model is given by

$$y(t) \sim \text{logNormal}(m_{\text{cent}}(t, x), \sigma). \tag{3.29}$$

This is somewhat of a simplification because in practice we measure the drug plasma concentration,

e.g. using blood sample, not the drug mass and the volume of the central compartment is unknown.

Each patient receives a total of 14 doses, taken every 12 hours. Measurements are taken at times $t = (0.083, 0.167, 0.25, 0.5, 0.75, 1, 1.5, 2, 3, 4, 6, 8)$ hours after the 1^{st}, 2^{nd}, and 13^{th}, and before all other dosing events.

We simulate data for 100 patients and for each patient, indexed by n, we estimate the coefficients (k_1^n, k_2^n). We use a hierarchical prior to pool information between patients and estimate the population parameters $(k_1^{\text{pop}}, k_2^{\text{pop}})$, with a non-centered parameterization. The full Bayesian model is given by

$$
\begin{aligned}
&\textit{hyperpriors} \\
k_1^{\text{pop}} &\sim \text{logNormal}(\log 1, 0.1) \\
k_2^{\text{pop}} &\sim \text{logNormal}(\log 0.3, 0.1) \\
\sigma_1 &\sim \text{logNormal}(\log 0.15, 0.1) \\
\sigma_2 &\sim \text{logNormal}(\log 0.35, 0.1) \\
\sigma &\sim \text{logNormal}(-1, 1) \\
&\textit{hierarchical priors} \\
\eta_1^n &\sim \text{Normal}(0, 1) \\
\eta_2^n &\sim \text{Normal}(0, 1) \\
k_1^n &= k_1^{\text{pop}} \exp(\eta_1^n \sigma_1) \\
k_2^n &= k_2^{\text{pop}} \exp(\eta_2^n \sigma_2) \\
&\textit{likelihood} \\
y_n &\sim \text{logNormal}(\log m_{\text{cent}}(t, k_1^n, k_2^n, x), \sigma).
\end{aligned}
$$

Note we fit the model on the unconstrained scale, meaning the Markov chains explore the param-

eter space of, for example, $\log k_1^{\text{pop}} \in \mathbb{R}$, rather than $k_1^{\text{pop}} \in \mathbb{R}^+$.

Chapter 4: Hamiltonian Monte Carlo using an adjoint-differentiated Laplace approximation

CHARLES C. MARGOSSIAN[1], AKI VEHTARI[2], DANIEL SIMPSON[3], RAJ AGRAWAL[4]

Gaussian latent variable models are a key class of Bayesian hierarchical models with applications in many fields. Performing Bayesian inference on such models can be challenging as Markov chain Monte Carlo algorithms struggle with the geometry of the resulting posterior distribution and can be prohibitively slow. An alternative is to use a Laplace approximation to marginalize out the latent Gaussian variables and then integrate out the remaining hyperparameters using dynamic Hamiltonian Monte Carlo, a gradient-based Markov chain Monte Carlo sampler. To implement this scheme efficiently, we derive a novel adjoint method that propagates the minimal information needed to construct the gradient of the approximate marginal likelihood. This strategy yields a scalable differentiation method that is orders of magnitude faster than state of the art differentiation techniques when the hyperparameters are high dimensional. We prototype the method in the probabilistic programming framework Stan and test the utility of the embedded Laplace approximation on several models, including one where the dimension of the hyperparameter is ~6,000. Depending on the cases, the benefits can include an alleviation of the geometric pathologies that frustrate Hamiltonian Monte Carlo and a dramatic speed-up.

[1]Columbia University, Department of Statistics; work partially done while a visiting doctoral student at the Department of Computer Science, Aalto University, Finland

[2]Aalto University, Department of Computer Science

[3]University of Toronto, Department of Statistical Sciences

[4]Massachusetts Institute of Technology, CSAIL

4.1 Introduction

Latent Gaussian models observe the following hierarchical structure:

$$\phi \sim \pi(\phi), \qquad \theta \sim \text{Normal}(0, K(\phi)), \qquad y \sim \pi(y \mid \theta, \phi).$$

Typically, single observations y_i are independently distributed and only depend on a linear combination of the latent variables, that is $\pi(y_i \mid \theta, \phi) = \pi(y_i \mid a_i^T \theta, \phi)$, for some appropriately defined vectors a_i. This general framework finds a broad array of applications: Gaussian processes, spatial models, and multilevel regression models to name a few examples. We denote θ the *latent Gaussian variable* and ϕ the *hyperparameter*, although we note that in general ϕ may refer to any latent variable other than θ. Note that there is no clear consensus in the literature on what constitutes a "latent Gaussian model"; we use the definition from the seminal work by Rue, Martino, and Chopin (2009).

We derive a method to perform Bayesian inference on latent Gaussian models, which scales when ϕ is high dimensional and can handle the case where $\pi(\phi \mid y)$ is multimodal, provided the energy barrier between the modes is not too strong. This scenario arises in, for example, general linear models with a regularized horseshoe prior (Carvalho, Polson, and Scott 2010) and in sparse kernel interaction models (Agrawal et al. 2019). The main application for these models is studies with a low number of observations but a high-dimensional covariate, as seen in genomics.

The inference method we develop uses a gradient-based Markov chain Monte Carlo (MCMC) sampler, coupled with a Laplace approximation to marginalize out θ. The key to successfully implementing this scheme is a novel adjoint method that efficiently differentiates the approximate marginal likelihood. In the case of a classic Gaussian process (Section 4.4), where $\dim(\phi) = 2$, the computation required to evaluate and differentiate the marginal is on par with the GPstuff package (Vanhatalo et al. 2013), which uses the popular algorithm by Rasmussen and Williams (2006). The adjoint method is however orders of magnitude faster when ϕ is high dimensional. Figure 4.1 shows the superior scalability of the adjoint method on simulated data from a sparse

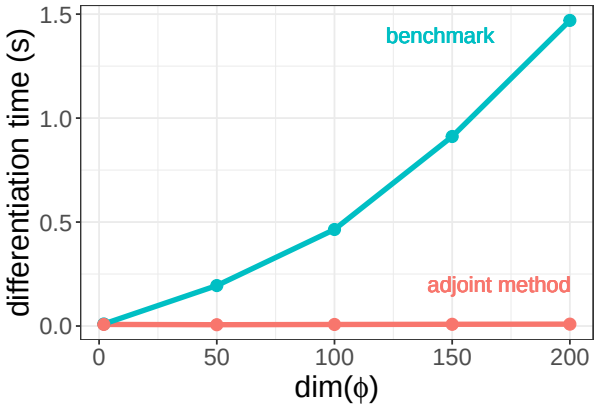

Figure 4.1: Wall time to differentiate the marginal density using the adjoint method (Algorithm 4.2) and, as a benchmark, the method by Rasmussen and Williams (2006) (Algorithm 4.1).

kernel interaction model. We lay out the details of the algorithms and the experiment in Section 4.3.

4.1.1 Existing methods

Bayesian computation is, broadly speaking, split between two approaches: (i) MCMC methods that approximately sample from the posterior, and (ii) approximation methods in which one finds a tractable distribution that approximates the posterior (e.g. variational inference, expectation propagation, and asymptotic approximations). The same holds for latent Gaussian models, where we can consider (i) Hamiltonian Monte Carlo (HMC) sampling (Neal 2012; Betancourt 2018a) and (ii) approximation schemes such as variational inference (VI) (Blei, Kucukelbir, and McAuliffe 2017) or marginalizing out the latent Gaussian variables with a Laplace approximation before deterministically integrating the hyperparameters (Tierney and Kadane 1986; Rue, Martino, and Chopin 2009).

Hamiltonian Monte Carlo sampling. When using MCMC sampling, the target distribution is

$$\pi(\theta, \phi \mid y) \propto \pi(y \mid \theta, \phi)\pi(\theta \mid \phi)\pi(\phi),$$

and the Markov chain explores the joint parameter space of θ and ϕ.

HMC is a class of MCMC algorithms that powers many modern probabilistic programming

languages, including Stan (Carpenter et al. 2017), PyMC3 (Salvatier, Wiecki, and Fonnesbeck 2016), and TensorFlow Probability (Dillon et al. 2017). Its success is both empirically and theoretically motivated (e.g. Betancourt et al. 2017) and, amongst other things, lies in its ability to probe the geometry of the target distribution via the gradient. The algorithm is widely accessible through a combination of its dynamic variants (Hoffman and Gelman 2014; Betancourt 2018a), which spare the users the cumbersome task of manually setting the algorithm's tuning parameters, and automatic differentiation, which alleviates the burden of calculating gradients by hand (e.g Margossian 2019; Baydin et al. 2018; Griewank and Walther 2008). There are known challenges when applying HMC to hierarchical models, because of the posterior distribution's problematic geometry (Betancourt and Girolami 2015). In the case of latent Gaussian models, this geometric grief is often caused by the latent Gaussian variable, θ, and its interaction with ϕ. Certain samplers, such as Riemannian HMC (Girolami, Calderhead, and Chin 2019; Betancourt 2013) and semi-separable HMC (Zhang and Sutton 2014), are designed to better handle difficult geometries. While promising, these methods are difficult to implement, computationally expensive, and to our knowledge not widely used.

Variational inference. VI proposes to approximate the target distribution, $\pi(\theta, \phi \mid y)$, with a tractable distribution, $q(\theta, \phi)$, which minimizes the Kullback-Leibler divergence between the approximation and the target. The optimization is performed over a pre-defined family of distributions, $\mathcal{Q}$. Adaptive versions, such as black-box VI (Ranganath, Gerrish, and Blei 2014) and automatic differentiation VI (ADVI) (Kucukelbir et al. 2017), make it easy to run the algorithm. VI is further made accessible by popular software libraries, including the above-mentioned probabilistic programming languages, and others packages such as GPyTorch for Gaussian processes (Gardner et al. 2018). For certain problems, VI is more scalable than MCMC, because it can be computationally much faster to solve an optimization problem than to generate a large number of samples. There are however known limitations with VI (e.g Blei, Kucukelbir, and McAuliffe 2017; Yao et al. 2018; Huggins et al. 2020; Dhaka et al. 2020). Of interest here is that $\mathcal{Q}$ may not include

appropriate approximations of the target: mean field or full rank Gaussian families, for instance, will underestimate variance and settle on a single mode, even if the posterior is multimodal (e.g. Yao et al. 2018).

Marginalization using a Laplace approximation. The embedded Laplace approximation is a popular algorithm, and a key component of the R packages INLA (*integrated nested Laplace integration*, Rue, Martino, and Chopin 2009; Rue et al. 2017) and TMB (*template model builder*, Kristensen et al. 2016), and the GPstuff package (Vanhatalo et al. 2013). The idea is to marginalize out θ and then use standard inference techniques on ϕ.

We perform the Laplace approximation

$$\pi(\theta \mid \phi, y) \approx \pi_{\mathcal{G}}(\theta \mid y, \phi) := \text{Normal}(\theta^*, \Sigma^*),$$

where θ^* matches the mode and $[\Sigma^*]^{-1}$ the curvature of $\pi(\theta \mid \phi, y)$. Then, the marginal posterior distribution is approximated as follows:

$$\pi(\phi \mid y) \approx \pi_{\mathcal{G}}(\phi \mid y) := \pi(\phi) \frac{\pi(\theta^* \mid \phi)\pi(y \mid \theta^*, \phi)}{\pi_{\mathcal{G}}(\theta^* \mid \phi, y)\pi(y)}.$$

Once we perform inference on ϕ, we can recover θ using the conditional distribution $\pi_{\mathcal{G}}(\theta \mid \phi, y)$ and effectively marginalizing ϕ out. For certain models, this yields much faster inference than MCMC, while retaining comparable accuracy (Rue, Martino, and Chopin 2009). Furthermore the Laplace approximation as a marginalization scheme enjoys very good theoretical properties (Tierney and Kadane 1986).

In the R package INLA, approximate inference is performed on ϕ, by characterizing $\pi(\phi \mid y)$ around its presumed mode. This works well for many cases but presents two limitations: the posterior must be well characterized in the neighborhood of the estimated mode and it must be low dimensional, *"2–5, not more than 20"* (Rue et al. 2017). In one of the examples we study, the posterior of ϕ is both high dimensional (~6000) and multimodal.

Hybrid methods. Naturally we can use a more flexible inference method on ϕ such as a standard MCMC, as discussed by Gómez-Rubio and Rue (2018), and HMC as proposed in GPstuff and TMB, the latter through its extension TMBStan and AdNuts (*automatic differentiation with a No-U-Turn Sampler* (Monnahan and Kristensen 2018)). The target distribution of the HMC sampler is now $\pi_{\mathcal{G}}(\phi \mid y)$.

To use HMC, we require the gradient of $\log \pi_{\mathcal{G}}(y \mid \phi)$ with respect to ϕ. Much care must be taken to ensure an efficient computation of this gradient. TMB and GPstuff exemplify two approaches to differentiate the approximate marginal density. The first uses automatic differentiation and the second adapts the algorithms in Rasmussen and Williams (2006). One of the main bottlenecks is differentiating the estimated mode, θ^*. In theory, it is straightforward to apply automatic differentiation, by brute-force propagating derivatives through θ^*, that is, sequentially differentiating the iterations of a numerical optimizer. But this approach, termed the *direct method*, is prohibitively expensive. A much faster alternative is to use the implicit function theorem (e.g Bell and Burke 2008; Margossian 2019). Given any accurate numerical solver, we can always use the implicit function theorem to get derivatives, as notably done in the Stan Math Library (Carpenter et al. 2015) and in TMB's *inverse subset algorithm* (Kristensen et al. 2016). One side effect is that the numerical optimizer is treated as a black box. By contrast, Rasmussen and Williams (2006) define a bespoke Newton method to compute θ^*, meaning we can store relevant variables from the final Newton step when computing derivatives. In our experience, this leads to important computational savings. But overall this method is much less flexible, working well only when ϕ is low dimensional and requiring the user to pass the tensor of derivatives, $\partial K / \partial \phi$.

4.2 Aim and results of the paper

We improve the computation of HMC with an embedded Laplace approximation. Our implementation accommodates any covariance matrix K, without requiring the user to specify $\partial K / \partial \phi$, efficiently differentiates $\log \pi_{\mathcal{G}}(y \mid \phi)$, even when ϕ is high dimensional, and deploys dynamic HMC to perform inference on ϕ. We introduce a novel adjoint method to differentiate $\log \pi_{\mathcal{G}}(y \mid$

ϕ), build the algorithm in C++, and add it to the Stan language. Our approach combines the Newton solver of Rasmussen and Williams (2006) with a non-trivial application of automatic differentiation.

Equipped with this implementation, we test dynamic HMC with an embedded Laplace approximation on a range of models, including ones with a high dimensional and multimodal hyperparameter. We do so by benchmarking our implementation against Stan's dynamic HMC, which runs MCMC on both the hyperparameter and the latent Gaussian variable. For the rest of the paper, we call this standard use of dynamic HMC, *full HMC*. We refer to marginalizing out θ and using dynamic HMC on ϕ, as the *embedded Laplace approximation*. Another candidate benchmark is Stan's ADVI. Yao et al. (2018) however report that ADVI underestimates the posterior variance and returns a unimodal approximation, even when the posterior is multimodal. We observe a similar behavior in the models we examine. For clarity, we relegate most of our analysis on ADVI to the Supplementary Material.

Our computer experiments identify cases where the benefits of the embedded Laplace approximation, as tested with our implementation, are substantial. In the case of a classic Gaussian process, with $\dim(\phi) = 2$ and $\dim(\theta) = 100$, we observe an important computational speed up, when compared to full HMC. We next study a general linear regression with a sparsity inducing prior; this time $\dim(\phi) \approx 6,000$ and $\dim(\theta) \approx 100$. Full HMC struggles with the posterior's geometry, as indicated by divergent transitions, and requires a model reparameterization and extensive tuning of the sampler. On the other hand, the embedded Laplace approximation evades many of the geometric problems and solves the approximate problem efficiently. We observe similar results for a sparse kernel interaction model, which looks at second-order interactions between covariates (Agrawal et al. 2019). Our results stand in contrast to the experiments presented in Monnahan and Kristensen (2018), who used a different method to automatically differentiate the Laplace approximation and reported at best a minor speed up. We do however note that the authors investigated different models than the ones we study here.

In all the studied cases, the likelihood is log-concave. Combined with a Gaussian prior, log-

concavity guarantees that $\pi(\theta \mid \phi, y)$ is unimodal. Detailed analysis on the error introduced by the Laplace approximation for log-concave likelihoods can be found in references (e.g Kuss and Rasmussen 2005; Vanhatalo, Pietiläinen, and Vehtari 2010; Cseke and Heskes 2011; Vehtari et al. 2016) and are consistent with the results from our computer experiments.

4.3 Implementation for probabilistic programming

In order to run HMC, we need a function that returns the approximate log density of the marginal likelihood, $\log \pi_{\mathcal{G}}(y \mid \phi)$, and its gradient with respect to ϕ, $\nabla_\phi \log \pi_{\mathcal{G}}(y \mid \phi)$. The user specifies the observations, y, and a function to generate the covariance K, based on input covariates x and the hyperparameters ϕ. In the current prototype, the user picks the likelihood, $\pi(y \mid \theta, \phi)$, from a set of options[5]: for example, a likelihood arising from a Bernoulli distribution with a logit link.

Standard implementations of the Laplace approximation use the algorithms in Rasmussen and Williams (2006, chapter 3 and 5) to compute (i) the mode θ^* and $\log \pi_{\mathcal{G}}(y \mid \phi)$, using a Newton solver; (ii) the gradient $\nabla_\phi \log \pi_{\mathcal{G}}(y \mid \phi)$ (Algorithm 4.1), and (iii) simulations from $\pi_{\mathcal{G}}(\theta \mid y, \phi)$. The major contribution of this paper is to construct a new differentiation algorithm, i.e. item (ii).

4.3.1 Using automatic differentiation in the algorithm of Rasmussen and Williams (2006)

The main difficulty with Algorithm 4.1 from Rasmussen and Williams (2006) is the requirement for $\partial K / \partial \phi_j$ at line 8. For classic problems, where K is, for instance, an exponentiated quadratic kernel, the derivatives are available analytically. This is not the case in general and, in line with the paradigm of probabilistic programming, we want a method that does not require the user to specify the tensor of derivatives, $\partial K / \partial \phi$.

Automatic differentiation allows us to numerically evaluate $\partial K / \partial \phi$ based on computer code to

[5]More likelihoods can be implemented through a C++ class that specifies the first three derivatives of the log-likelihood.

Algorithm 4.1: Gradient of the approximate marginal density, $\pi_{\mathcal{G}}(y \mid \phi)$, with respect to the hyperparameters ϕ, *adapted from algorithm 5.1 by Rasmussen and Williams (2006, chapter 5). We store and reuse terms computed during the final Newton step, algorithm 3.1 in Rasmussen and Williams (2006, chapter 3).*

input: y, ϕ, $\pi(y \mid \theta, \phi)$

saved input from the Newton solver: θ^*, K, $W^{\frac{1}{2}}$, L, a

$Z \leftarrow \frac{1}{2} a^T \theta^* + \log \pi(y \mid \theta^*, \phi) - \sum \log(\text{diag}(L))$

$R \leftarrow W^{\frac{1}{2}} L^T \setminus (L \setminus W^{\frac{1}{2}})$

$C \leftarrow L \setminus (W^{\frac{1}{2}} K)$

$s_2 \leftarrow -\frac{1}{2} \text{diag}(\text{diag}(K) - \text{diag}(C^T C)) \nabla_\theta^3 \log \pi(y \mid \theta^*, \phi)$

for $j \in \{1, \cdots, \dim(\phi)\}$ **do**

  $K' \leftarrow \partial K / \partial \phi_j$

  $s_1 \leftarrow \frac{1}{2} a^T K' a - \frac{1}{2} \text{tr}(RK')$

  $b \leftarrow K' \nabla_\theta \log \pi(y \mid \theta, \phi)$

  $s_3 \leftarrow b - KRb$

  $\frac{\partial}{\partial \phi_j} \pi(y \mid \phi) \leftarrow s_1 + s_2^T s_3$

end

return $\nabla_\phi \log \pi_{\mathcal{G}}(y \mid \phi)$

evaluate K. To do this, we introduce the map $\mathcal{K}$

$$\begin{aligned} \mathcal{K} \quad &: \quad \mathbb{R}^p \to \mathbb{R}^{n(n+1)/2} \\ &\phi \to K, \end{aligned}$$

where p is the dimension of ϕ and n that of θ. To obtain the full tensor of derivatives, we require either p forward mode sweeps or $n(n+1)/2$ reverse mode sweeps. Given the scaling, we favor forward mode and this works well when p is small. However, once p becomes large, this approach is spectacularly inefficient.

4.3.2 Adjoint method to differentiate the approximate log marginal density

To evaluate the gradient of a composite map, it is actually not necessary to compute the full Jacobian matrix of intermediate operations. This is an important, if often overlooked, property of automatic differentiation and the driving principle behind *adjoint methods* (e.g Errico 1997; Margossian and Betancourt 2022). This idea motivates an algorithm that does not explicitly construct $\partial K / \partial \phi$, a calculation that is both expensive and superfluous. Indeed, it suffices to evaluate $\Omega^T \partial K / \partial \phi$ for the correct *cotangent matrix*, Ω^T, an operation we can do in a single reverse mode sweep of automatic differentiation.

Theorem 4.1. *Let* $\log \pi_{\mathcal{G}}(y \mid \phi)$ *be the approximate log marginal density in the context of a latent Gaussian model. Let a be defined as in the Newton solver by Rasmussen and Williams (2006, chapter 3), and let R and* s_2 *be defined as in Algorithm 4.1. Then*

$$\nabla_\phi \log \pi_{\mathcal{G}}(y \mid \phi) = \Omega^T \frac{\partial K}{\partial \phi},$$

where the gradient is with respect to ϕ *and*

$$\Omega^T = \frac{1}{2} a a^T - \frac{1}{2} R + (s_2 + RKs_2)[\nabla_\theta \log \pi(y \mid \theta, \phi)]^T.$$

The proof follows from Algorithm 4.1 and noting that all the operations in $\partial K / \partial \phi_j$ are linear. We provide the details in the Supplementary Material. Armed with this result, we build Algorithm 4.2, a method that combines the insights of Rasmussen and Williams (2006) with the principles of adjoint methods.

Algorithm 4.2: Gradient of the approximate marginal log density, $\log \pi_{\mathcal{G}}(y \mid \phi)$, with respect to the hyperparameters, ϕ, using reverse mode automatic differentiation and theorem 4.1.

1 **input:** y, ϕ, $\pi(y \mid \theta, \phi)$

2 Do lines 2 - 6 of Algorithm 4.1.

3 Initiate an expression graph for automatic differentiation with $\phi_v = \phi$.

4 $K_v \leftarrow \mathcal{K}(\phi_v)$

5 $\Omega^T \leftarrow \frac{1}{2} a a^T - \frac{1}{2} R + (s_2 + RKs_2)[\nabla_\theta \log \pi(y \mid \theta, \phi)]^T$

6 Do a reverse sweep over K, with Ω^T as the initial cotangent to obtain $\nabla_\phi \log \pi_{\mathcal{G}}(y \mid \phi)$.

7 **return:** $\nabla_\phi \log \pi_{\mathcal{G}}(y \mid \phi)$.

Figure 4.1 shows the time required for one evaluation and differentiation of $\log \pi_{\mathcal{G}}(y \mid \phi)$ for the sparse kernel interaction model developed by Agrawal et al. (2019) on simulated data. The covariance structure of this model is nontrivial and analytical derivatives are not easily available. We simulate a range of data sets for varying dimensions, p, of ϕ. For low dimensions, the difference is small; however, for $p = 200$, Algorithm 4.2 is more than 100 times faster than Algorithm 4.1, requiring 0.009 s, instead of 1.47 s.

4.4 Gaussian process with a Poisson likelihood

We fit the disease map of Finland by Vanhatalo, Pietiläinen, and Vehtari (2010) which models the mortality count across the country. The data is aggregated in $n = 911$ grid cells. We use 100 cells, which allows us to fit the model quickly both with full HMC and HMC using an embedded Laplace approximation. For the i^{th} region, we have a 2-dimensional coordinate x_i, the counts of

deaths y_i, and the standardized expected number of deaths, y_e^i. The full latent Gaussian model is

$$(\rho, \alpha) \sim \pi(\rho, \alpha), \qquad \theta \sim \text{Normal}(0, K(\alpha, \rho, x)), \qquad y_i \sim \text{Poisson}(y_e^i e^{\theta_i}),$$

where K is an exponentiated quadratic kernel, α is the marginal standard deviation and ρ the characteristic length scale. Hence $\phi = (\alpha, \rho)$.

Fitting this model with MCMC requires running the Markov chains over α, ρ, and θ. Because the data is sparse — one observation per group — the posterior has a funnel shape which can lead to biased MCMC estimates (Neal 2003; Betancourt and Girolami 2015). A useful diagnostic for identifying posterior shapes that challenge the HMC sampler is *divergent transitions*, which occur when there is significant numerical error in the computation of the Markov chain trajectory (Betancourt 2018a).

To remedy these issues, we reparameterize the model and adjust the *target acceptance rate*, δ_a. δ_a controls the precision of HMC, with the usual trade-off between accuracy and speed. For well behaved problems, the optimal value is 0.8 (Betancourt and Girolami 2015) but posteriors with highly varying curvature require a higher value. Moreover, multiple attempts at fitting the model must be done before we correctly tune the sampler and remove all the divergent transitions. See the Supplementary Material for more details.

An immediate benefit of the embedded Laplace approximation is that we marginalize out θ and only run HMC on α and ρ, a two-dimensional and typically well behaved parameter space. In the case of the disease map, we do not need to reparameterize the model, nor adjust δ_a.

We fit the models with both methods, using 4 chains, each with 500 warmup and 500 sampling iterations. A look at the marginal distributions of α, ρ, and the first two elements of θ suggests the posterior samples generated by full HMC and the embedded Laplace approximation are in close agreement (Figure 4.2). With a Poisson likelihood, the bias introduced by the Laplace approximation is small, as shown by Vanhatalo, Pietiläinen, and Vehtari (2010). We benchmark the Monte Carlo estimates of both methods against results from running 18,000 MCMC iterations. The em-

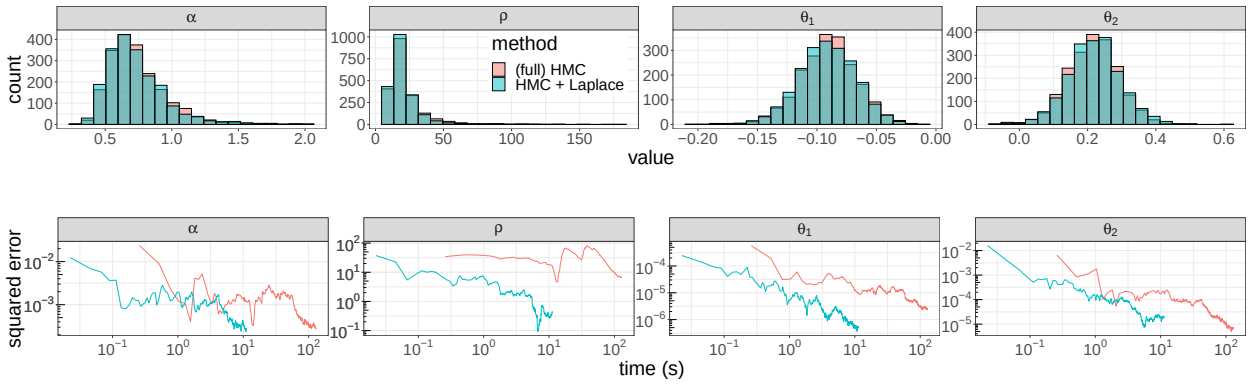

Figure 4.2: (Up) Posterior samples obtained with full HMC and the embedded Laplace approximation when fitting the disease map. (Down) Error when estimating the expectation value against wall time. Unreported in the figure is that we had to fit full HMC twice before obtaining good tuning parameters.

bedded Laplace approximations yields comparable precision, when estimating expectation values, and is an order of magnitude faster (Figure 4.2). In addition, we do not need to tune the algorithm and the MCMC warmup time is much shorter (~10 seconds against ~200 seconds for full HMC).

4.5 General linear regression model with a regularized horseshoe prior

Consider a regression model with n observations and p covariates. In the "$p \gg n$" regime, we typically need additional structure, such as sparsity, for accurate inference. The horseshoe prior (Carvalho, Polson, and Scott 2010) is a useful prior when it is assumed that only a small portion of the regression coefficients are non-zero. Here we use the regularized horseshoe prior by Piironen and Vehtari (2017). The horseshoe prior is parameterized by a global scale term, the scalar τ, and local scale terms for each covariate, λ_j, $j = 1, \ldots, p$. Consequently the number of hyperparameters is $O(p)$.

To use the embedded Laplace approximation, we recast the regularized linear regression as a latent Gaussian model. The benefit of the approximation is not a significant speedup, rather an improved posterior geometry, due to marginalizing θ out. This means we do not need to reparameterize the model, nor fine tune the sampler. To see this, we examine the genetic microarray classification data set on prostate cancer used by Piironen and Vehtari (2017) and fit a regression

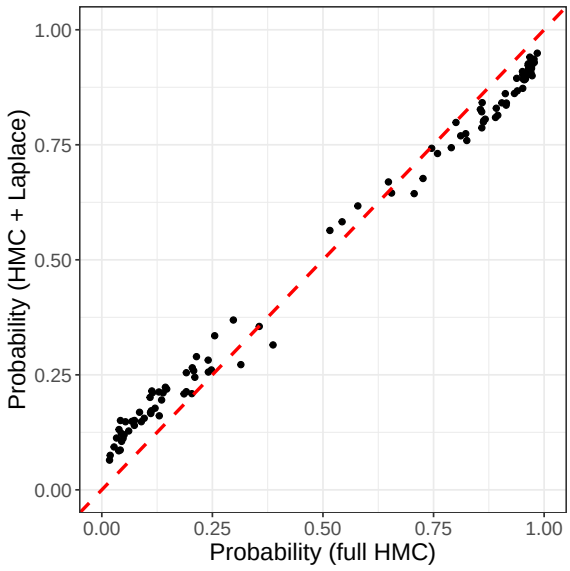

Figure 4.3: Expectation value for the probability of developing prostate cancer, as estimated by full HMC and HMC using an embedded Laplace approximation.

model with a Bernoulli distribution and a logit link. Here, $\dim(\theta) = 102$ and $\dim(\phi) = 5,966$.

We use 1,000 iterations to warm up the sampler and 12,000 sampling iterations. Tail quantiles, such as the 90^{th} quantile, allow us to identify parameters which have a small local shrinkage and thence indicate relevant covariates. The large sample size is used to reduce the Monte Carlo error in our estimates of these extreme quantiles.

Fitting this model with full HMC requires a fair amount of work: the model must be reparameterized and the sampler carefully tuned, after multiple attempts at a fit. We use a non-centered parameterization, set $\delta_a = 0.999$ (after attempting $\delta_a = 0.8$ and $\delta_a = 0.99$) and do some additional adjustments. Even then we obtain 13 divergent transitions over 12,000 sampling iterations. The Supplementary Material describes the tuning process in all its thorny details. By contrast, running the embedded Laplace approximation with Stan's default tuning parameters produces 0 divergent transitions. Hence the approximate problem is efficiently solved by dynamic HMC. Running ADVI on this model is also straightforward.

Table 4.1 shows the covariates with the highest 90^{th} quantiles, which are softly selected by full HMC, the embedded Laplace approximation and ADVI. For clarity, we exclude ADVI from the remaining figures but note that it generates, for this particular problem, strongly biased infer-

(full) HMC	2586	1816	4960	4238	4843	3381
HMC + Laplace	2586	1816	4960	4647	4238	3381
ADVI	1816	2416	4284	2586	5279	4940

Table 4.1: Top six covariate indices, i, with the highest 90$^{\text{th}}$ quantiles of $\log \lambda_i$ for the general linear model with a regularized horseshoe prior. The first two methods are in good agreement; ADVI selects different covariates, in part because it approximates the multimodal posterior with a unimodal distribution (see the Supplementary Material).

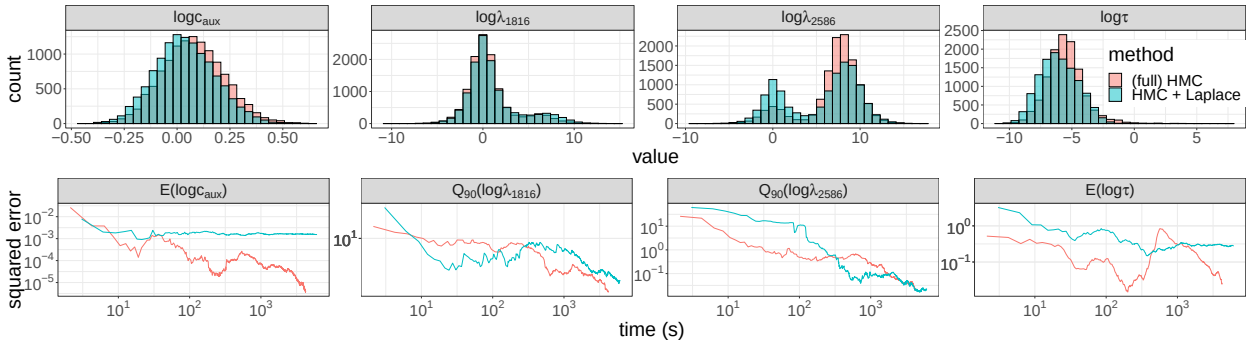

Figure 4.4: (Up) Posterior samples obtained with full HMC and HMC using an embedded Laplace approximation when fitting a general linear regression with a regularized horseshoe prior. (Down) Error when estimating various quantities of interest against wall time. E stands for "expectation" and Q_{90}, "90$^{\text{th}}$ quantile". Unreported in the figure is that we had to run full HMC four times before obtaining reasonable tuning parameters.

ence; more details can be found in the Supplementary Material. Figure 4.3 compares the expected probability of developing cancer. Figure 4.4 compares the posterior samples and the error when estimating various quantities of interest, namely (i) the expectation value of the global shrinkage, τ, and the slab parameter, c_{aux}; and (ii) the 90$^{\text{th}}$ quantile of two local shrinkage parameters. As a benchmark we use estimates obtained from 98,000 MCMC iterations.

The Laplace approximation yields slightly less extreme probabilities of developing cancer than the corresponding full model. This behavior is expected for latent Gaussian models with a Bernoulli observation model, and has been studied in the cases of Gaussian processes and Gaussian random Markov fields (e.g Kuss and Rasmussen 2005; Cseke and Heskes 2011; Vehtari et al. 2016). While introducing a bias, the embedded Laplace approximation yields accuracy compa-

rable to full HMC when evaluating quantities of interest.

4.6 Sparse kernel interaction model

A natural extension of the general linear model is to include interaction terms. To achieve better computational scalability, we can use the kernel interaction trick by Agrawal et al. (2019) and build a sparse kernel interaction model (SKIM), which also uses the regularized horseshoe prior by Piironen and Vehtari (2017). The model is an explicit latent Gaussian model and uses a non-trivial covariance matrix. The full details of the model are given in the Supplementary Material.

When fitting the SKIM to the prostate cancer data, we encounter similar challenges as in the previous section: ∼150 divergent transitions with full HMC when using Stan's default tuning parameters. The behavior when adding the embedded Laplace approximation is much better, although there are still ∼3 divergent transitions,[6] which indicates that this problem remains quite difficult even after the approximate marginalization. We also find large differences in running time. The embedded Laplace approximation runs for ∼10 hours, while full HMC takes ∼20 hours with $\delta_a = 0.8$ and ∼50 hours with $\delta_a = 0.99$, making it difficult to tune the sampler and run our computer experiment.

For computational convenience, we fit the SKIM using only 200 covariates, indexed 2500 - 2700 to encompass the 2586^{th} covariate which we found to be strongly explanatory. This allows us to easily tune full HMC without altering the takeaways of the experiment. Note that the data here used is different from the data we used in the previous section (since we only examine a subset of the covariates) and the marginal posteriors should therefore not be compared directly.

As in the previous section, we generate 12,000 posterior draws for each method. For full HMC we obtain 36 divergent transitions with $\delta_a = 0.8$ and 0 with $\delta_a = 0.99$. The embedded Laplace approximation produces 0 divergences with $\delta_a = 0.8$. Table 4.2 shows the covariates which are softly selected. As before, we see a good overlap between full HMC and the embedded

[6] We do our preliminary runs using only 4000 sampling iterations. The above number are estimated for 12000 sampling iterations. The same holds for the estimated run times.

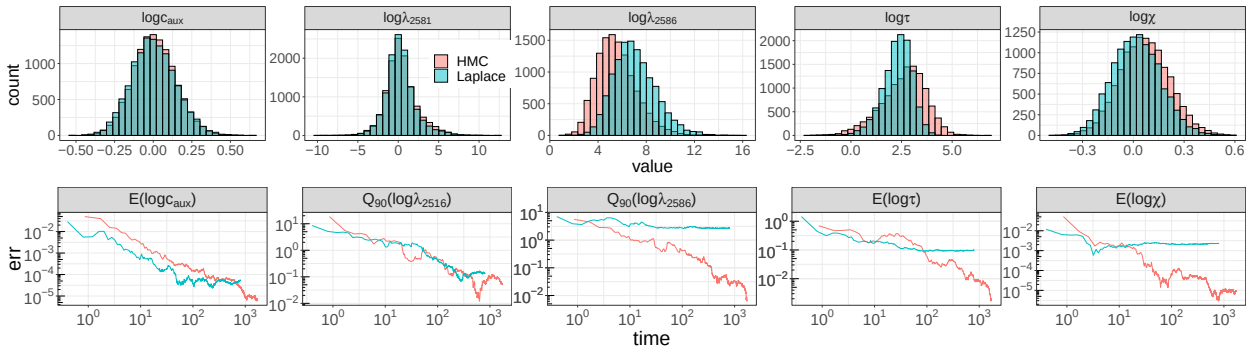

Figure 4.5: (Up) Samples obtained with full HMC and HMC using an embedded Laplace approximation when fitting the SKIM. (Down) Error when estimating various quantities of interest against wall time. E stands for "expectation" and Q_{90}, "90$^{\text{th}}$ quantile". Unreported in the figure is that we had to run full HMC twice before obtaining reasonable tuning parameters.

(full) HMC	2586	2660	2679	2581	2620	2651
HMC + Laplace	2586	2679	2660	2581	2620	2548
ADVI	2586	2526	2106	2550	2694	2166

Table 4.2: Top six covariate indices, i, with the highest 90$^{\text{th}}$ quantiles of $\log \lambda_i$ for the SKIM.

Laplace approximation, and mostly disagreeing results from ADVI. Figure 4.5 compares (i) the posterior draws of full HMC and the embedded Laplace approximation, and (ii) the error over time, benchmarked against estimates from 98,000 MCMC iterations, for certain quantities of interest. We obtain comparable estimates but note that the Laplace approximation introduces a bias, which becomes more evident over longer runtimes.

4.7 Discussion

Equipped with a scalable and flexible differentiation algorithm, we expand the regime of models to which we can apply the embedded Laplace approximation. HMC allows us to perform inference even when ϕ is high dimensional and multimodal, provided the energy barrier is not too strong. In the case where $\dim(\theta) \gg \dim(\phi)$, the approximation also yields a dramatic speedup. When $\dim(\theta) \ll \dim(\phi)$, marginalizing θ out can still improve the geometry of the posterior, sav-

ing the user time otherwise spent tuning the sampling algorithm. However, when the posterior is well-behaved, the approximation may not provide any benefit.

Our next step is to further develop the prototype for Stan. We are also aiming to incorporate features that allow for a high performance implementation, as seen in the packages INLA, TMB, and GPstuff. Examples include support for sparse matrices required to fit latent Markov random fields, parallelization and GPU support.

We also want to improve the flexibility of the method by allowing users to specify their own likelihood. TMB provides this flexibility but in our view two important challenges persist. Recall that unlike full HMC, which only requires first-order derivatives, the embedded Laplace approximation requires the third-order derivative of the likelihood (but not of the other components in the model). It is in principle possible to apply automatic differentiation to evaluate higher-order derivatives and most libraries, including Stan, support this; but, along with feasibility, there is a question of efficiency and practicality (e.g. Betancourt 2018b): the automated evaluation of higher-order derivatives is often prohibitively expensive. The added flexibility also burdens us with more robustly diagnosing errors induced by the approximation. There is extensive literature on log-concave likelihoods but less so for general likelihoods. Future work will investigate diagnostics such as importance sampling (Vehtari et al. 2019), leave-one-out cross-validation (Vehtari et al. 2016), and simulation based calibration (Talts et al. 2020).

Acknowledgment

We thank Michael Betancourt, Steve Bronder, Alejandro Catalina, Rok Češnovar, Hyunji Moon, Sam Power, Sean Talts and Yuling Yao for helpful discussions.

CM thanks the Office of Naval Research, the National Science Foundation, the Institute for Education Sciences, and the Sloan Foundation. CM and AV thank the Academy of Finland (grants 298742 and 313122). DS thanks the Canada Research Chairs program and the Natural Sciences and Engineering Research Council of Canada. RA's research was supported in part by a grant from DARPA.

We acknowledge computing resources from Columbia University's Shared Research Computing Facility project, which is supported by NIH Research Facility Improvement Grant 1G20RR030893-01, and associated funds from the New York State Empire State Development, Division of Science Technology and Innovation (NYSTAR) Contract C090171, both awarded April 15, 2010.

We also acknowledge the computational resources provided by the Aalto Science-IT project.

The authors declare to have no conflict of interest.

4.8 Appendix (Supplementary Material)

We review the Newton solver proposed by Rasmussen and Williams (2006) and prove theorem 4.1, the main result required to do build an adjoint method for the embedded Laplace approximation. We next present our prototype code and provide details for the models used in our computer experiments.

4.8.1 Newton solver for the embedded Laplace approximation

Algorithm 4.3: Newton solver for the embedded Laplace approximation Rasmussen and Williams 2006, chapter 3

input: K, y, $\pi(y \mid \theta, \phi)$

$\theta^* \leftarrow \theta_0$ ▷ initialization

while *convergence is not achieved* **do**

  $W \leftarrow -\nabla_\theta \nabla_\theta \log \pi(y \mid \theta^*, \phi)$

  $L \leftarrow \text{Cholesky}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$

  $b \leftarrow W\theta^* + \nabla_\theta \log \pi(y \mid \theta^*, \phi)$

  $a \leftarrow b - W^{\frac{1}{2}} L^T \setminus (L \setminus (W^{\frac{1}{2}} K b))$

  $\theta^* \leftarrow Ka$

end

$\log \pi(y \mid \phi) \leftarrow -\frac{1}{2} a^T \theta^* + \log \pi(y \mid \theta^*, \phi) - \sum_i \log L_{ii}$

return: θ^*, $\log \pi_{\mathcal{G}}(y \mid \phi)$

Algorithm 4.3 is a transcription of the Newton method by Rasmussen and Williams (2006,

chapter 3) using our notation. As a convergence criterion, we use the change in the objective function between two iterations

$$\Delta \log \pi(\theta \mid y, \phi) \leq \epsilon$$

for a specified ϵ. This is consistent with the approach used in GPStuff (Vanhatalo et al. 2013). We store the following variables generated during the final Newton step to use them again when computing the gradient: θ^*, K, $W^{\frac{1}{2}}$, L, and a. This avoids redundant computation and spares us an expensive Cholesky decomposition.

4.8.2 Building the adjoint method

To compute the gradient of the approximate log marginal with respect to ϕ, $\nabla \log \pi_{\mathcal{G}}(y \mid \phi)$, we exploit several important principles of automatic differentiation. While widely used in statistics and machine learning, these principles remain arcane to many practitioners and deserve a brief review. We will then construct the adjoint method (theorem 4.1 and algorithm 4.2) as a correction to algorithm 4.1.

Automatic differentiation

Given a composite map

$$f = f^L \circ f^{L-1} \circ ... f^1,$$

the chain rule teaches us that the corresponding Jacobian matrix observes a similar decomposition:

$$J = J_L \cdot J_{L-1} \cdot ... \cdot J_1.$$

Based on computer code to calculate f, a *forward mode sweep* automatic differentiation numerically evaluates the action of the Jacobian matrix on the initial tangent u, or *directional derivative*

$J \cdot u$. Extrapolating from the chain rule

$$\begin{aligned} J \cdot u &= J_L \cdot J_{L-1} \cdot \ldots \cdot J_3 \cdot J_2 \cdot J_1 \cdot u \\ &= J_L \cdot J_{L-1} \cdot \ldots \cdot J_3 \cdot J_2 \cdot u_1 \\ &= J_L \cdot J_{L-1} \cdot \ldots \cdot J_3 \cdot u_2 \\ &\ldots \\ &= J_L \cdot u_{L-1}, \end{aligned}$$

where the u_l's verify the recursion relationship

$$\begin{aligned} u_1 &= J_1 \cdot u, \\ u_l &= J_l \cdot u_{l-1}. \end{aligned}$$

If our computation follows the steps outlined above we never need to explicitly compute the full Jacobian matrix, J_l, of an intermediate function, f^l; rather we only calculate a sequence of Jacobian-tangent products. Similarly a *reverse mode sweep* evaluates the contraction of the Jacobian matrix with a cotangent, w^T, yielding $w^T J$, by computing a sequence cotangent-Jacobian products.

Hence, in the case of the embedded Laplace approximation, where

$$\begin{aligned} \mathcal{K}: \quad \phi &\rightarrow K \\ \mathbb{R}^p &\rightarrow \mathbb{R}^{(n+1)n/2} \end{aligned}$$

is an intermediate function, we do not need to explicitly compute $\partial K / \partial \phi$ but only $w^T \partial K / \partial \phi$ for the appropriate cotangent vector. This type of reasoning plays a key role when differentiating functionals of implicit functions – for example, probability densities that depend on solutions to ordinary differential equations – and leads to so-called *adjoint methods* (e.g. Errico 1997).

Derivation of the adjoint method

In this section we provide a proof of theorem 4.1. As a starting point, assume algorithm 4.1 is valid. The proof can be found in Rasmussen and Williams (2006, chapter 5). The key observation is that all operations performed on

$$\frac{\partial K}{\partial \phi_j}$$

are linear. Algorithm 1 produces a map

$$\begin{aligned} \mathcal{Z} \quad &: \partial K / \partial \phi_j \quad \to \frac{\partial}{\partial \phi_j} \pi(y \mid \phi) \\ &: \mathbb{R}^{n \times n} \quad \to \mathbb{R}, \end{aligned}$$

and constructs the gradient one element at a time. By linearity,

$$\frac{\partial}{\partial \phi_j} \mathcal{Z}(K) = \mathcal{Z}\left(\frac{\partial K}{\partial \phi_j}\right).$$

Thus an alternative approach to compute the gradient is to calculate the scalar $\mathcal{Z}(K)$ and then use a single reverse mode sweep of automatic differentiation, noting that $\mathcal{Z}$ is an analytical function. This produces Algorithm 4.4. At this point, the most important is done in order to achieve

Algorithm 4.4: Gradient of the approximate marginal log density, $\log \pi_{\mathcal{G}}(y \mid \phi)$, with respect to the hyperparameters, ϕ, using reverse mode automatic differentiation

1 **input:** y, ϕ, $\pi(y \mid \theta, \phi)$
2 Do lines 2 - 6 of Algorithm 2.
3 Initiate an expression tree for automatic differentiation with $\phi_v = \phi$.
4 $K_v \leftarrow \mathcal{K}(\phi_v)$
5 $z \leftarrow \mathcal{Z}(K_v)$
6 Do a reverse-sweep over z to obtain $\nabla_\phi \log \pi(y \mid \phi)$.
7 **return:** $\nabla_\phi \log \pi(y \mid \phi)$.

scalability: we no longer explicitly compute $\partial K / \partial \phi$ and are using a single reverse mode sweep.

Automatic differentiation, for all its relatively cheap cost, still incurs some overhead cost. Hence, where possible, we still want to use analytical results to compute derivatives. In particular, we can analytically work out the cotangent

$$\Omega^T := \frac{\partial z}{\partial K}.$$

For the following calculations, we use a lower case, k_{ij} and r_{ij}, to denote the $(ij)^{\text{th}}$ element respectively of the matrices K and R.

Consider

$$\mathcal{Z}(K) = s_1 + s_2^T s_3,$$

where, unlike in Algorithm 1, s_1 and s_3 are now computed using K, not $\partial K/\partial \phi_j$. We have

$$s_1 = \frac{1}{2} a^T K a - \frac{1}{2}\text{tr}(RK).$$

Then

$$\frac{\partial}{\partial k_{i'j'}} a^T K a = \frac{\partial}{\partial k_{i'j'}} \sum_i \sum_j a_i k_{ij} a_j = a_{i'} a_{j'},$$

and

$$\frac{\partial}{\partial k_{i'j'}} \text{tr}(RK) = \frac{\partial}{\partial k_{i'j'}} \sum_l r_{il} k_{li} = r_{j'i'}.$$

Thus

$$\frac{\partial s_1}{\partial K} = \frac{1}{2} a a^T - \frac{1}{2} R^T.$$

For convenience, denote $l = \nabla_\theta \log \pi(y \mid \theta, \phi)$. We then have

$$b = Kl,$$
$$s_3 = b - \tilde{K}Rb = (I - \tilde{K}R)b,$$

where $\tilde{K} = K$, but is maintained fixed, meaning we do not propagate derivatives through it. Let $\tilde{A} = I - \tilde{K}R$ and let $\tilde{a}_{ij}$ denote the $(i,j)^{\text{th}}$ element of $\tilde{A}$. Then

$$s_2^T s_3 = \sum_i (s_2)_i \left(\sum_j \tilde{a}_{ij} \sum_m k_{jm} l_m \right).$$

Thus

$$\frac{\partial}{\partial k_{i'j'}} s_2^T s_3 = \sum_i (s_2)_i \tilde{a}_{ii'} l_{j'} = l_{j'} \sum_i (s_2)_i \tilde{a}_{ii'},$$

where the sum term is the $(i')^{\text{th}}$ element of $\tilde{A}s_2$. The above expression then becomes

$$\frac{\partial}{\partial K} s_2^T s_3 = \tilde{A}s_2 l^T = s_2 l^T - KRs_2 l^T.$$

Combining the derivative for s_1 and $s_2^T s_3$ we obtain

$$\Omega^T = \frac{1}{2} a a^T - \frac{1}{2} R + (s_2 + RKs_2)[\nabla_\theta \log \pi(y \mid \theta, \phi)]^T,$$

as prescribed by Theorem 4.1. This result is general, in the sense that it applies to any covariance matrix, K, and likelihood, $\pi(y \mid \theta, \phi)$. Our preliminary experiments, on the SKIM, found that incorporating the analytical cotangent, Ω^T, approximately doubles the differentiation speed.

4.8.3 Computer code

The code used in this work is open source and detailed in this section.

Prototype Stan code

The Stan language allows users to specify the joint log density of their model. This is done by incrementing the variable `target`. We add a suite of functions, which return the approximate log marginal density, $\log \pi_{\mathcal{G}}(y \mid \phi)$. Hence, the user can specify the log joint distribution by incrementing `target` with $\log \pi_{\mathcal{G}}(y \mid \phi)$ and the prior $\log \pi(\phi)$. A call to the approximate marginal density looks as follows:

```
target +=
  laplace_marginal_*_lpmf (y | n, K, phi, x, delta,
                           delta_int, theta0);
```

The `*` specifies the obervation model, typically a distribution and a link function, for example `bernoulli_logit` or `poisson_log`. The suffix `lpmf` is used in Stan to denote a log posterior mass function. `y` and `n` are sufficient statistics for the latent Gaussian variable, θ; `K` is a function that takes in arguments `phi`, `x`, `delta`, and `delta_int` and returns the covariance matrix; and `theta0` is the initial guess for the Newton solver, which seeks the mode of $\pi(\theta \mid \phi, y)$. Moreover, we have

- `y`: a vector containing the sum of counts/successes for each element of θ,
- `n`: a vector with the number of observation for each element of θ,
- `K`: a function defined in the functions block, with the signature `(vector, data matrix, data real[], data int[]) ==> matrix`. Note that only the first argument may be used to pass variables which depend on model parameters, and through which we propagate derivatives. The term `data` means an argument may not depend on model parameters.
- `phi`: the vector of hyperparameters,
- `x`: a matrix of data. For Gaussian processes, this is the coordinates, and for the general linear regression, the design matrix,

- `delta`: additional real data,
- `delta_int`: additional integer data,
- `theta0`: a vector of initial guess for the Newton solver.

It is also possible to specify the tolerance of the Newton solver. This structure is consistent with other higher-order functions in Stan, such as the algebraic solver and the ordinary differential equation solvers. It gives users flexibility when specifying K, but we recognize it is cumbersome. One item on our to-do list is to use variadic arguments, which remove the constraints on the signature of `K`, and allows users to pass any combination of arguments to `K` through `laplace_marignal_*_lpmf`.

For each observation model, we implement a corresponding random number generating function, with a call

```
theta =  laplace_marginal_*_rng (y, n, K, phi, x, delta,
                                 delta_int, theta0);
```

This generates a random sample from $\pi_{\mathcal{G}}(\theta \mid y, \phi)$. This function can be used in the generated quantities blocks and is called only once per iteration – in contrast with the target function which is called and differentiated once per integration step of HMC. Moreover the cost of generating θ is negligible next to the cost evaluating and differentiating $\log \pi(y \mid \phi)$ multiple times per iteration.

The interested reader may find a notebook with demo code, including R scripts and Stan files, at https://github.com/charlesm93/StanCon2020, as part of the 2020 Stan Conference (Margossian et al. 2020a).

C++ code

We incorporate the Laplace suite of functions inside the Stan-math library, a C++ library for automatic differentiation (Carpenter et al. 2015). The library is open source and available on GitHub, https://github.com/stan-dev/math. The prototype used in this paper exists on the

branch `try-laplace_approximation2`[7]. The code is structured around a main function

```
laplace_approximation (likelihood, K_functor, phi, x, delta,
                       delta_int, theta0);
```

with

- `likelihood`: a class constructed using `y` and `n`, which returns the log density, as well as its first, second, and third order derivatives.
- `K_functor`: a functor that computes the covariance matrix, K
- ...: the remaining arguments are as previously described.

A user can specify a new likelihood by creating the corresponding class, meaning the C++ code is expandable.

To expose the code to the Stan language, we use Stan's new OCaml transpiler, stanc3, https://github.com/stan-dev/stanc3 and again the branch `try-laplace_approximation2`.

Important note: the code is prototypical and currently not merged into Stan's release or development branch.

Code for the computer experiment

The code is available on the GitHub public repository, https://github.com/charlems93/laplace_manuscript.

We make use of two new prototype packages: CmdStanR (https://mc-stan.org/cmdstanr/) and posterior (https://github.com/jgabry/posterior).

4.8.4 Tuning dynamic Hamiltonian Monte Carlo

In this article, we use the dynamic Hamiltonian Monte Carlo sampler described by Betancourt (2018a) and implemented in Stan. This algorithm builds on the No-U Turn Sampler by Hoffman

[7]Our first prototype is was on the branch `try-laplace_approximation`, and was used to conduct the here presented computer experiment. The new branch modifies the functions' signatures to be more consistent with the Stan language. In this Supplement, we present the new signatures.

and Gelman (2014), which adaptively tunes the sampler during a warmup phase. Hence for most problems, the user does not need to worry about tuning parameters. However, the models presented in this article are challenging and the sampler requires careful tuning, if we do not use the embedded Laplace approximation.

The main parameter we tweak is the *target acceptance rate*, δ_a. To run HMC, we need to numerically compute physical trajectories across the parameter space by solving the system of differential equations prescribed by Hamilton's equations of motion. We do this using a numerical integrator. A small step size, δ, makes the integrator more precise but generates smaller trajectories, which leads to a less efficient exploration of the parameter space. When we introduce too much numerical error, the proposed trajectory is rejected. Adapt delta, $\delta_a \in (0, 1)$, sets the target acceptance rate of proposed trajectories. During the warmup, the sampler adjusts δ to meet this target. For well-behaved problems, the optimal value of δ_a is 0.8 (Betancourt and Girolami 2015).

It should be noted that the algorithm does not necessarily achieve the target set by δ_a during the warmup. One approach to remedy this issue is to extend the warmup phase; specifically the final fast adaptation interval or *term buffer* (see Hoffman and Gelman 2014; Stan development team 2020). By default, the term buffer runs for 50 iterations (when running a warmup for 1,000 iterations). Still, making the term buffer longer does not guarantee the sampler attains the target δ_a. There exist other ways of tuning the algorithm, but at this points, the technical burden on the user is already significant. What is more, probing how well the tuning parameters work usually requires running the model for many iterations.

4.8.5 Automatic differentiation variational inference

ADVI automatically derives a variational inference algorithm, based on a user specified log joint density. Hence we can use the same Stan file we used for full HMC and, with the appropriate call, run ADVI instead of MCMC. The idea behind ADVI is to approximate the posterior over the unconstrained space using a Gaussian distribution, either with a diagonal covariance matrix – leading to a mean-field approximation – or with a full rank covariance matrix. The details of

this procedure are described in Kucukelbir et al. 2017. Compared to full HMC, ADVI can be much faster, but in general it is difficult to assess how well the variational approximation describes the target posterior distribution without using an expensive benchmark (Yao et al. 2018; Huggins et al. 2020). Furthermore, it can be challenging to assess the convergence of ADVI (Dhaka et al. 2020).

To run ADVI, we use the Stan file with which we ran full HMC. We depart from the default tuning parameters by decreasing the learning rate η to 0.1, adjusting the tolerance, `rel_tol_obj`, and increasing the maximum number of iterations to 100,000. Our goal is to improve the accuracy of the optimizer as much as possible, while insuring that convergence is reached.

We compare the samples drawn from the variational approximation to samples drawn from full HMC in Figures 4.6, 4.7 and 4.8. For the studied examples, we find the approximation to be not very satisfactory, either because it underestimates the posterior variance, does not capture the skewness of the posterior distribution, or returns a unimodal approximation when in fact the posterior density is multimodal. These are all features which cannot be captured by a Gaussian over the unconstrained scale. Naturally, a different choice for Q could lead to better inference. Using a custom VI algorithm is however challenging, as we need to derive a useful variational family and hand-code the inference algorithm, rather than rely on the implementation in a probabilistic programming language.

4.8.6 Model details

We review the models used in our computer experiments and point the readers to the relevant references.

Disease map

The disease map uses a Gaussian process with an exponentiated squared kernel,

$$k(x_i, x_j) = \alpha^2 \exp\left(-\frac{(x_i - x_j)^T (x_i - x_j)}{\rho^2}\right).$$

The full latent Gaussian model is

$$\begin{aligned}
\rho &\sim \text{invGamma}(a_\rho, b_\rho), \\
\alpha &\sim \text{invGamma}(a_\alpha, b_\alpha), \\
\theta &\sim \text{Normal}(0, K(\alpha, \rho, x)), \\
y_i &\sim \text{Poisson}(y_e^i e^{\theta_i}),
\end{aligned}$$

where we put an inverse-Gamma prior on ρ and α.

When using full HMC, we construct a Markov chain over the joint parameter space (α, ρ, θ). To avoid Neal's infamous funnel (Neal 2003) and improve the geometry of the posterior distribution, it is possible to use a *non-centered parameterization*:

$$\begin{aligned}
(\rho, \alpha) &\sim \pi(\rho, \alpha), \\
z &\sim \text{Normal}(0, I_{n\times n}), \\
L &= \text{Cholesky decompose}(K), \\
\theta &= Lz, \\
y_i &\sim \text{Poisson}(y_e^i e^{\theta_i}).
\end{aligned}$$

The Markov chain now explores the joint space of (α, ρ, z) and the θ's are generated by transforming the z's. With the embedded Laplace approximation, the Markov chain only explores the joint space (α, ρ).

To run ADVI, we use the same Stan file as for full HMC and set `tol_rel_obj` to 0.005.

Regularized horseshoe prior

The horseshoe prior (Carvalho, Polson, and Scott 2010) is a sparsity inducing prior that introduces a global shrinkage parameter, τ, and a local shrinkage parameter, λ_i for each covariate slope, β_i. This prior operates a soft variable selection, effectively favoring $\beta_i \approx 0$ or $\beta_i \approx \hat{\beta}_i$, where $\hat{\beta}_i$

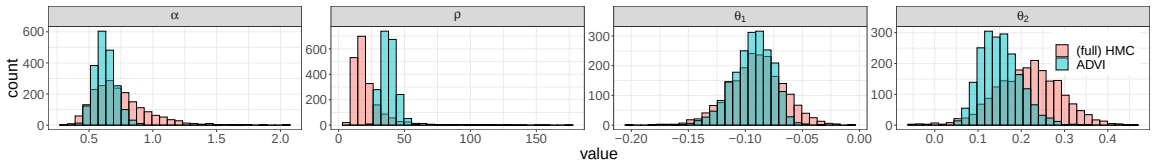

Figure 4.6: Samples obtained with full HMC and sampling from the variational approximation produced by ADVI when fitting the disease map. Unlike the embedded Laplace approximation, ADVI strongly disagrees with full HMC.

is the maximum likelihood estimator. Piironen and Vehtari (2017) add another prior to regularize unshrunk βs, Normal$(0, c^2)$, effectively operating a "soft-truncation" of the extreme tails.

Details on the prior. For computational stability, the model is parameterized using c_{aux}, rather than c, where

$$c = s_{\text{slab}}\sqrt{c_{\text{aux}}}$$

with s_{slab} the slab scale. The hyperparameter is $\phi = (\tau, c_{\text{aux}}, \lambda)$ and the prior

$$\begin{aligned}
\lambda_i &\sim \text{Student}_t(\nu_{\text{local}}, 0, 1), \\
\tau &\sim \text{Student}_t(\nu_{\text{global}}, 0, s_{\text{global}}), \\
c_{\text{aux}} &\sim \text{inv}\Gamma(s_{\text{df}}/2, s_{\text{df}}/2), \\
\beta_0 &\sim \text{Normal}(0, c_0^2).
\end{aligned}$$

The prior on λ independently applies to each element, λ_i.

Following the recommendation by Piironen and Vehtari (2017), we set the variables of the priors as follows. Let p be the number of covariates and n the number of observations. Additionally, let p_0 be the expected number of relevant covariates – note this number does not strictly enforce the number of unregularized βs, because the priors have heavy enough tails that we can depart

from p_0. For the prostate data, we set $p_0 = 5$. Then

$$\begin{aligned} s_{\text{global}} &= \frac{p_0}{\sqrt{n}(p - p_0)}, \\ \nu_{\text{local}} &= 1, \\ \nu_{\text{global}} &= 1, \\ s_{\text{slab}} &= 2, \\ s_{\text{df}} &= 100, \\ c_0 &= 5. \end{aligned}$$

Next we construct the prior on β,

$$\beta_i \sim \text{Normal}(0, \tau^2 \tilde{\lambda}_i^2),$$

where

$$\tilde{\lambda}_i^2 = \frac{c^2 \lambda_i^2}{c^2 + \tau^2 \lambda_i^2}.$$

Formulations of the data generating process

The data generating process is

$$\begin{aligned} \phi &\sim \pi(\phi), \\ \beta_0 &\sim \text{Normal}(0, c_0^2), \\ \beta &\sim \text{Normal}(0, \Sigma(\phi)), \\ y &\sim \text{Bernoulli\_logit}(\beta_0 + X\beta), \end{aligned}$$

or, equivalently,

$$\begin{aligned}
\phi &\sim \pi(\phi), \\
\theta &\sim \text{Normal}(0,\, c_0^2 I_{n\times n} + X\Sigma(\phi)X^T), \\
y &\sim \text{Bernoulli_logit}(\theta).
\end{aligned}$$

For full HMC, we use a non-centered parameterization of the first formulation, much like we did for the disease map. The embedded Laplace approximation, as currently implemented, requires the second formulation, which is mathematically more convenient but comes at the cost of evaluating and differentiating $K = c^2 I_{n\times n} + X\Sigma(\phi)X^T$. In this scenario, the main benefit of the Laplace approximation is not an immediate speed-up but an improved posterior geometry, due to marginalizing θ (and thus implicitly β and β_0) out. This means we do not need to fine tune the sampler.

Fitting the model with full HMC. This section describes how to tune full HMC to fit the model at hand. Some of the details may be cumbersome to the reader. But the takeaway is simple: tuning the algorithm is hard and can be a real burden for the modeler.

Using a non-centered parameterization and with Stan's default parameters, we obtain ~150 divergent transitions[8]. We increase the target acceptance rate to $\delta_a = 0.99$ but find the sampler now produces 186 divergent transitions. A closer inspection reveals the divergences all come from a single chain, which also has a larger adapted step size, δ. The problematic chain also fails to achieve the target acceptance rate. These results are shown in Table 4.3. From this, it seems increasing δ_a yet again may not provide any benefits. Instead we increase the term buffer from 50 iterations to 350 iterations. With this setup, we however obtain divergent transitions across all chains.

This outcome indicates the chains are relatively unstable and emphasizes how difficult it is,

[8]To be precise, we here did a preliminary run using 4000 sampling iterations and obtained 50 divergent transitions (so an expected 150 over 12000 sampling iterations).

Chain	Step size	Acceptance rate	Divergences
1	0.0065	0.99	0
2	0.0084	0.90	186
3	0.0052	0.99	0
4	0.0061	0.99	0

Table 4.3: Adapted tuning parameters across 4 Markov chains with $\delta_a = 0.99$.

for this type of model and data, to come up with the right tuning parameters. With $\delta_a = 0.999$ and the extended term buffer we observe 13 divergent transitions. It is possible this result is the product of luck, rather than better tuning parameters. To be clear, we do not claim we found the optimal model parameterization and tuning parameters. There is however, to our knowledge, no straightforward way to do so.

Fitting the model with the embedded Laplace approximation. Running the algorithm with Stan's default tuning parameters produces 0 divergent transitions over 12,000 sampling iterations.

Fitting the model with ADVI. To run ADVI, we use the same Stan file as for full HMC and set `tol_rel_obj` to 0.005.

The family of distribution, $\mathcal{Q}$, over which ADVI optimizes requires the exact posterior distribution to be unimodal over the unconstrained scale. This is a crucial limitation in the studied example, as shown in Figure 4.7. This notably affects our ability to select relevant covariates using the 90^{th} posterior quantile. When examining the top six selected covariates (Table 1 in the main text), we find the result from ADVI to be in disagreement with full HMC and the embedded Laplace approximation. In particular, λ_{2586} which corresponds, according to our other inference methods, to the most relevant covariate, has a relatively low 90^{th} quantile. This is because ADVI only approximates the smaller mode of $\pi(\lambda_{2586} \mid y)$. Our results are consistent with the work by Yao et al. (2018), who examine ADVI on a similar problem.

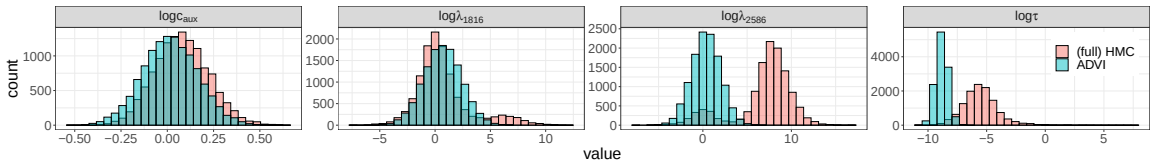

Figure 4.7: Samples obtained with full HMC and sampling from the variational approximation produced by ADVI when fitting a general linear model with a regularized horseshoe prior.

Sparse kernel interaction model

SKIM, developed by Agrawal et al. (2019), extends the model of Piironen and Vehtari (2017) by accounting for pairwise interaction effects between covariates. The generative model shown below uses the notation in 4.8.6 instead of that in Appendix D of Agrawal et al. (2019):

$$
\begin{aligned}
\chi &\sim \text{inv}\Gamma(s_{\text{df}}/2, s_{\text{df}}/2), \\
\eta_2 &= \frac{\tau^2}{c^2}\chi, \\
\beta_i \mid \tau, \tilde{\lambda} &\sim \text{Normal}(0, \tau^2\tilde{\lambda}_i^2), \\
\beta_j \mid \tau, \tilde{\lambda} &\sim \text{Normal}(0, \tau^2\tilde{\lambda}_i^2), \\
\beta_{ij} \mid \eta_2, \tilde{\lambda} &\sim \text{Normal}(0, \eta_2^2\tilde{\lambda}_i^2\tilde{\lambda}_j^2), \\
\beta_0 \mid c_0^2 &\sim \text{Normal}(0, c_0^2),
\end{aligned}
$$

where β_i and β_{ij} are the main and pairwise effects for covariates x_i and $x_i x_j$, respectively, and τ, $\tilde{\lambda}$, c_0 are defined in 4.8.6.

Instead of sampling $\{\beta_i\}_{i=1}^p$ and $\{\beta_{ij}\}_{i,j=1}^p$, which takes at least $O(p^2)$ time per iteration to store and compute, Agrawal et al. (2019) marginalize out all the regression coefficients, only sampling $(\tau, \xi, \tilde{\lambda})$ via MCMC. Through a kernel trick and a Gaussian process re-parameterization of the model, this marginalization takes $O(p)$ time instead of $O(p^2)$. The Gaussian process covariance matrix K induced by SKIM is provided below:

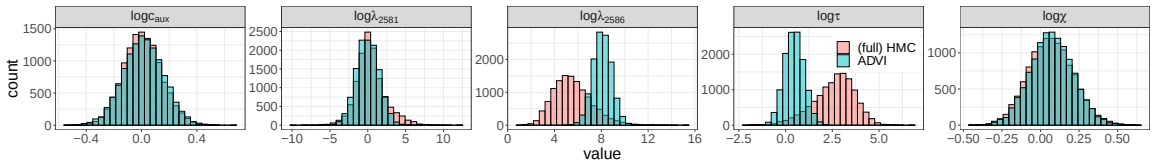

Figure 4.8: Samples obtained with full HMC and sampling from the variational approximation produced by ADVI when fitting the SKIM.

$$
\begin{aligned}
K_1 &= x\ \mathrm{diag}(\tilde{\lambda}^2)\ x^T, \\
K_2 &= [x \circ x]\ \mathrm{diag}(\tilde{\lambda}^2)\ [x \circ x]^T,
\end{aligned}
$$

where "$\circ$" denotes the element-wise Hadamard product. Finally,

$$
\begin{aligned}
K &= \frac{1}{2}\eta_2^2(K_1 + 1) \circ (K_1 + 1) - \frac{1}{2}\eta_2^2 K_2 - (\tau^2 - \eta_2^2)K_1 \\
&\quad + c_0^2 - \frac{1}{2}\eta_2^2.
\end{aligned}
$$

Chapter 5: General Adjoint-differentiated Laplace approximation

CHARLES C. MARGOSSIAN

The hierarchical prior used in Latent Gaussian models (LGMs) induces a posterior geometry prone to frustrate inference algorithms. Marginalizing out the latent Gaussian variable using an integrated Laplace approximation removes the offending geometry, allowing us to do efficient inference on the remaining parameters. To use gradient-based inference we need to compute the approximate marginal likelihood and its gradient. The adjoint-differentiated Laplace approximation by Margossian et al. (2020b) offers a scalable method to differentiate LGMs with an arbitrary prior covariance but places restrictions on the likelihood. Indeed the existing implementation only works for likelihoods with a diagonal Hessian and requires users to pass analytical expressions for the first three derivatives of the likelihood. I propose a generalization which is applicable to a broad class of likelihoods and does not require any analytical derivatives. The added flexibility comes at no computational cost: when compared to existing methods on a standard LGM, the proposed generalization is in fact slightly faster. I also apply the general method to an LGM with an unconventional likelihood. This example highlights the algorithm's potential, as well as persistent challenges.

5.1 Introduction

Latent Gaussian models (LGMs) are a popular class of Bayesian models, which include Gaussian processes, multilevel models with a sparsity inducing prior, and population models, for example in Pharmacometrics. Their general formulation is

$$\phi \sim \pi(\phi), \qquad \theta \sim \text{Normal}(0, K(\phi)), \qquad y \sim \pi(y \mid \theta, \eta),$$

where $\phi \in \mathbb{R}^p$ and $\eta \in \mathbb{R}^T$ denote the (potentially overlapping) hyperparameters and $\theta \in \mathbb{R}^n$ is the latent Gaussian variable. The hierarchical prior on θ induces a challenging posterior geometry, specifically an uneven curvature produced by the interaction between ϕ and θ, and a strong posterior correlation between the different components of θ; see Section 2.3 for further details. The *integrated Laplace approximation*, also termed the embedded Laplace approximation, proposes to marginalize out θ, thereby allowing us to do inference on the more manageable distribution $\pi(\phi, \eta \mid y)$. The posterior distribution of θ can then be studied by approximating the conditional distribution $\pi(\theta \mid \phi, \eta, y)$ in a post-sampling step.

Most existing implementations of the integrated Laplace approximation algorithmically restrict the class of LGMs we can fit. This is because developers focus on specific motivating problems and write algorithms whose speed and stability depend on certain regularity conditions. To expand the scope of the integrated Laplace approximation, we must develop methods which do not rely on such conditions. Furthermore the advent of automatic differentiation in Machine Learning and Computational Statistics presents new opportunities to write code which is both more general and more efficient (Baydin et al. 2018; Margossian 2019; Margossian and Betancourt 2022). Beyond algorithmic limitations, there is also an inferential limitation, which is that the Laplace approximation may not be adequate. It is worth noting the quality of the approximation depends not only on the likelihood distribution but also on the interaction of the likelihood and the prior. In the limiting case where there is no observation tied to a particular parameter θ_i (e.g. empty cell in spatial model), the posterior distribution of θ_i is exactly normal. If the data is sparse, it will be approximatively normal. For examples on how the data regime influences the quality of the approximation, see the discussion by Vanhatalo, Jylänki, and Vehtari (2009) and Talts et al. (2020).

Here I merely address the problem of constructing and differentiating the Laplace approximation, while recognizing that the approximation is not always useful. This project has two immediate benefits:

(i) when writing software to support a menu of likelihoods, a single function can be used for all likelihoods, thereby making the code shorter, more readable and straightforward to

expand;

(ii) it is possible to experiment with new likelihoods and conduct research on the utility of the integrated Laplace approximation.

I achieve both of these goals by building a prototype of the method in the probabilistic programming language Stan (Carpenter et al. 2015; Carpenter et al. 2017).

5.1.1 Classical implementation and limitation

Fast algorithms take advantage of the convenient structure found in classical models at the expanse of applications to less conventional cases. For example, the seminal algorithms by Rasmussen and Williams (2006) for inference on Gaussian processes assume the following:

(i) $\eta = \emptyset$

(ii) The log likelihood $\log \pi(y \mid \theta, \eta) = \log \pi(y \mid \theta)$ has a diagonal Hessian. This often means each observation y_i can only depend on a single component of θ.

(iii) $\pi(y \mid \theta, \eta)$ is a log-concave likelihood and the Hessian is negative definite.

We can take advantage of these conditions to build a numerically stable Newton solver to find the mode of $\pi(\theta \mid y, \phi, \eta)$ when constructing the Laplace approximation.

Calculations of the approximate log marginal likelihood, $\log \pi_{\mathcal{G}}(y \mid \phi, \eta)$, and its gradient with respect to ϕ require methods to explicitly compute the derivative of the prior covariance, $\partial K / \partial \phi$, and the first three derivatives of the likelihood with respect to θ. This means users must either pick from a menu of prior covariances and likelihoods, with pre-coded derivatives, or engage in the time-consuming and error prone task of hand-coding the requisite derivatives. Many software implementations inherit at least some of these limitations (e.g Vanhatalo et al. 2013; Rue et al. 2017; Margossian et al. 2020b).

The following examples violate the above conditions and required adjustments to make the computation of the Laplace approximation feasible:

Gaussian process regression with a Student-t likelihood. Vanhatalo, Jylänki, and Vehtari (2009) and later Jylänki, Vanhatalo, and Vehtari (2011) propose to use a Student-t likelihood in order to make Gaussian process regression robust to outliers. The Student-t likelihood is parameterized by the latent Gaussian variable and an additional scale parameter, meaning $\eta \neq \emptyset$. Furthermore the Student-t distribution is not log-concave and thus its Hessian not negative definite.

Motorcyle Gaussian process example. This example, described by Tolvanen, Jylänki, and Vehtari (2014) and Vehtari (2021), combines two Gaussian processes,

$$\begin{aligned} y &\sim \text{normal}(\mu(x), \Sigma) \\ \mu &\sim \text{GP}(0, K_1(x)) \\ \tau &\sim \text{GP}(0, K_2(x)), \end{aligned}$$

where

$$\Sigma = \begin{pmatrix} \exp(\tau_1(x)) & 0 & \cdots & \cdots & 0 \\ 0 & \exp(\tau_2(x)) & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & \exp(\tau_n(x)) \end{pmatrix}$$

Both the mean and variance to vary as a function of the covariate x. Conditional on μ_i and τ_i, the y_i's are independent of the other elements of y. Combining the latent Gaussian variables into one,

$$\theta = (\mu_1, \tau_1, \mu_2, \tau_2, \cdots, \mu_n, \tau_n),$$

the above reduces to a single Gaussian process. The resulting model admits a 2×2 block-diagonal Hessian and is typically not negative-definite. The GPStuff package implements an integrated Laplace approximation for this example, using an optimizer with carefully tuned initializations (Vanhatalo et al. 2013).

Additional examples of LGMs with a less conventional structure include population pharma-

cometrics models with a differential equation based likelihood (e.g Gibaldi and Perrier 1982; Gastonguay and Metrum Institute Facility 2013; Margossian, Zhang, and Gillespie 2022) and neural networks with a horseshoe prior (Broussard et al. 2021). To my knowledge, the integrated Laplace approximation has not been used for these or related examples. Section 5.6 presents an attempt to use Hamiltonian Monte Carlo with an integrated Laplace approximation on a standard pharmacometrics model, using the general implementation developed in this article.

5.1.2 Existing methods

Expanding the scope of the integrated Laplace approximation requires finding more general methods to (i) construct a Laplace approximation of $\pi(\theta \mid \phi, \eta, y)$ which is fundamentally an optimization problem, and (ii) differentiate the approximate marginal distribution $\pi_{\mathcal{G}}(y \mid \phi, \eta)$.

Alternative Newton solvers

The Newton solver employed by Rasmussen and Williams (2006) assumes the negative Hessian

$$W \triangleq -\frac{\partial^2 \pi(y \mid \theta, \eta)}{\partial \theta^2}$$

is diagonal and positive definite, meaning $W_{ii} \geq 0$. If the likelihood is not log concave this condition is violated. To address this Vanhatalo, Jylänki, and Vehtari (2009) and Rasmussen and Nickish (2010) propose modified Newton solvers. Another direction may be to use the Fisher information matrix rather than the Hessian[1]. The information matrix is always semi-positive definite while the definiteness of the Hessian depends on the value of θ, and crucially values of θ we encounter along the optimization path.

Gradient computation

Early implementations of the integrated Laplace approximation, for example in the GPStuff (Vanhatalo et al. 2013) and in the INLA (Rue et al. 2017) packages require methods which explic-

[1] Personal communication with Jarno Vanhatalo.

itly compute derivatives, $\partial K / \partial \phi$, and higher-order derivatives of $\log \pi(y \mid \theta, \eta)$ with respect to θ and η. These derivatives are then plugged into a differentiation algorithm. The optimization step, which produces the Laplace approximation, and the differentiation of the approximate marginal distribution are done conjointly, meaning a host of terms can be carried over from one calculation to the other. In the algorithm by Rasmussen and Williams (2006) this notably spares us an expansive Cholesky decomposition during the differentiation step.

Kristensen et al. (2016) apply automatic differentiation to the Laplace approximation and remove the requirement for any analytical derivatives, implementing their flexible strategy in the TMB package. Their *inverse subset algorithm* bypasses the explicit computation of intermediate Jacobians and can in this sense be seen as an adjoint method. Margossian et al. (2020b) use the same principles of automatic differentiation, showing that the calculation of $\partial K / \partial \phi$ is both superfluous and prohibitively expansive for high-dimensional ϕ. We shall see that a similar reasoning applies to derivatives of the likelihood. The adjoint-differentiated Laplace approximation by Margossian et al. (2020b) differs from the inverse subset algorithm in two ways. First it requires analytical higher-order derivatives of the likelihood; the main object of this article is to eliminate this requirement. Secondly, the inverse subset algorithm treats the optimizer as a black box, while the adjoint-differentiated Laplace approximation jointly optimizes and differentiates, in line with the procedure by Rasmussen and Williams (2006). This comes at the cost of only working for a specific type of Newton solver with the limitations described in Section 5.1.1.

5.1.3 Aim and results

I present a generalization of the adjoint-differentiated Laplace approximation for a class of Newton solvers, which have appeared in the literature (e.g. Rasmussen and Williams 2006; Vanhatalo, Jylänki, and Vehtari 2009; Rasmussen and Nickish 2010; Vanhatalo, Foster, and Hosack 2021). This unifying framework follows from the realization that most of these solvers are distinguished only by their application of the Woodburry-Sherman-Morrison matrix inversion lemma. The key step to insure a numerically stable algorithm is to carefully choose a B-matrix which can

be safely inverted. Judicious choices of B depend on the properties of the Hessian and the prior covariance. The cost of the Newton step is then dominated by an $O(n^3)$ decomposition of B.

Automatic differentiation removes the requirement for analytical derivatives of the likelihood, provided all operations in the evaluation of the likelihood support both forward and reverse mode differentiation. Differentiating the Laplace approximation requires propagating derivatives through the mode of $\log \pi(\theta \mid y, \theta, \eta)$. This can be done using the implicit function theorem. Gaebler (2021) show that the cost of differentiation is then dominated by an LU decomposition. This costly operation is eliminated by reusing the B-matrix, decomposed in the final Newton step. As in Margossian et al. (2020b), I use the adjoint method of automatic differentiation, taking care to adapt the method to higher-order derivatives. Another important step for an efficient implementation is to exploit the sparsity of the Hessian. Without loss of generality, suppose the Hessian, $\nabla^2_\theta \log \pi(y \mid \theta, \eta)$, is block diagonal, with block size $m \times m$. The number of automatic differentiation sweeps required to differentiate the approximate marginal density, $\pi_{\mathcal{G}}(y \mid \theta, \eta)$, is $O(m)$. Crucially this number does not depend on either the data size, the dimension of the latent Gaussian θ, nor the dimension of the hyperparameters ϕ and η.

The resulting algorithm only requires users to specify code to evaluate the prior covariance and the log likelihood but not their derivatives. In general switching from analytical derivatives to automatic differentiation incurs some computational cost. However adjoint methods, in addition to automating the computation of derivatives, bypass the evaluation of certain terms, which may be expansive to calculate even when analytical expressions exist. When applied to the sparse kernel interaction model (SKIM) studied by Margossian et al. (2020b) (Figure 4.1), the general adjoint-differentiation outperforms the adjoint-differentiation, despite not having access to analytical derivatives (Figure 5.1). Hence the generalization presents code which is more readable, more expandable, and slightly more efficient.

Understanding the benefits of the general adjoint-differentiated Laplace approximation on unconventional likelihoods remains ongoing work. Section 5.6 examines a population pharmacokinetic model, which a likelihood parameterized by a linear ordinary differential equation. This ex-

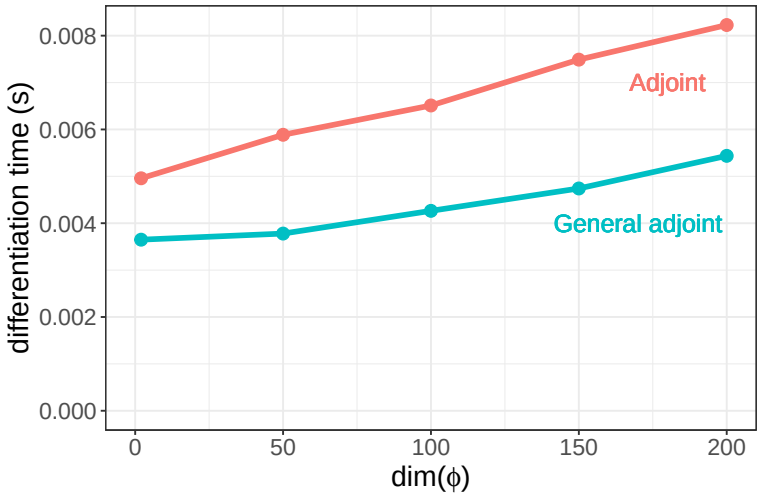

Figure 5.1: Wall time to differentiate the marginal density of a SKIM using the general adjoint-differentiated Laplace approximation (Algorithm 5.4), benchmarked against the method by Margossian et al. (2020b).

amples demonstrates the integrated Laplace approximation can be very accurate in an unorthodox setting, but highlights how challenging it can be to compute the approximation when the likelihood is not log-concave.

5.2 Newton solvers and B-matrices

Algorithm 5.1: Abstract Newton solver

input: y, ϕ, η, K, $\pi(y \mid \theta, \eta)$, θ_0, Δ ;

$\theta \leftarrow \theta_0$ ▷ initial guess

$\Psi_{\text{new}} \leftarrow -\infty$ ▷ initial objective

while $(|\Psi_{\text{new}} - \Psi_{\text{old}}| \geq \Delta)$ **do**

  $W \leftarrow -\nabla^2_\theta \log \pi(y \mid \theta, \eta)$

  $\theta \leftarrow (K^{-1} + W)^{-1}(W\theta + \nabla_\theta \log \pi(y \mid \theta, \eta))$

  $\Psi_{\text{old}} \leftarrow \Psi_{\text{new}}$

  $\Psi_{\text{new}} \leftarrow -\frac{1}{2}\theta^T K^{-1}\theta + \log \pi(y \mid \theta, \eta)$

end

return: $\log \pi_{\mathcal{G}}(y \mid \phi, \eta) = \Psi_{\text{new}} - \frac{1}{2}\log|K||K^{-1} + W|$

Algorithm 5.1 describes an *abstract Newton solver*, without the details required to insure a numerically stable implementation. The expression for the approximate marginal likelihood, $\log \pi_{\mathcal{G}}(y \mid \phi, \eta)$, is obtained via a Gaussian integral; see Rasmussen and Williams (2006, Section 3.4.4). We use this abstraction to define a class of Newton solvers which can be used to compute the approximate marginal likelihood and its gradient, while requiring minimal adjustments to our algorithm when we switch optimizer.

The main difficulty with a "brute force" implementation of the above solver is that we cannot safely invert the prior covariance matrix, K, or the sum $(K^{-1} + W)$, whose eigenvalues may be arbitrarily close to 0. We will encounter similar issues when trying to invert W in our calculation of the gradient. Our main asset to avoid these difficult inversions is the Woodburry-Sherman-Morrison formula, stated below for convenience.

Lemma 5.1. *(Woodburry-Sherman-Morrison formula.) Given $Z \in \mathbb{R}^{n\times n}$, $W \in \mathbb{R}^{m\times m}$, $U \in \mathbb{R}^{n\times m}$, and $V \in \mathbb{R}^{n\times m}$, we have*

$$(Z + UWV^T)^{-1} = Z^{-1} - Z^{-1}U(W^{-1} + V^T Z^{-1} U)^{-1} V^T Z^{-1},$$

where we assume the relevant inverses all exist.

Lemma 5.1 offers several decompositions we can take advantage of. We consider three:

$$\begin{aligned}
(K^{-1} + W)^{-1} &= K - KW^{\frac{1}{2}}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})^{-1} W^{\frac{1}{2}} K, \\
&= K^{\frac{1}{2}}(I + K^{\frac{1}{2}T} W K^{\frac{1}{2}})^{-1} K^{\frac{1}{2}T}, \\
&= K - KW(I + KW)^{-1} K, \qquad (5.1)
\end{aligned}$$

where $A^{\frac{1}{2}}$ is an equivalence class of matrices, such that $A^{\frac{1}{2}} A^{\frac{1}{2}T} = A$.

All three decompositions present us with a matrix which is safe to invert. Overloading notation, I denote B the matrix to invert on the RHS. Remarkably all B-matrices have the same log determinant, which is equal to the log determinant needed to compute the approximate marginal

distribution. This can be shown using the determinant version of the Woodbury-Sherman-Morrison formula, stated below for convenience.

Lemma 5.2. *(Woodbury-Sherman-Morrison for determinants) Given* $Z \in \mathbb{R}^{n\times n}$, $W \in \mathbb{R}^{m\times m}$, $U \in \mathbb{R}^{n\times m}$, *and* $V \in \mathbb{R}^{n\times m}$, *we have*

$$|Z + UWV^T| = |Z||W||W^{-1} + V^T Z^{-1} U|,$$

where we assume the relevant inverses all exist.

Then some careful manipulations give us

$$\begin{aligned} \log|K||K^{-1} + W| &= \log|I + W^{\frac{1}{2}} K W^{\frac{1}{2}}| \\ &= \log|I + K^{\frac{1}{2}T} W K^{\frac{1}{2}}| \\ &= \log|I + KW|. \end{aligned} \tag{5.2}$$

5.2.1 $B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}}$

To use the first decomposition we need to compute $W^{\frac{1}{2}}$. One option is to take the matrix square-root which requires W to be quasi-triangular[2]. To achieve better computation we may exploit the sparsity of the Hessian, which is often block-diagonal. Denoting $m \times m$ the size of each block and $n \times n$ the size of the Hessian, computing the square root or the Cholesky decomposition block-wise costs $O(m^2 n)$ operations, rather than $O(n^3)$. In the case where W is positive-definite and diagonal, it suffices to take the element-wise square-root for a total complexity $O(n)$.

The advantage of this decomposition is that B is a symmetric matrix. We can therefore compute a Cholesky decomposition, L, and in turn use it for *solve* methods, and to compute the log

[2]See the implementation in Eigen, https://eigen.tuxfamily.org/dox/unsupported/group__MatrixFunctions__Module.html.

determinant of B,

$$\log |B| = \log |L||L^T| = 2 \sum_i \log L_{ii}.$$

We can now write a numerically stable version of the Newton solver under the assumption that $W^{\frac{1}{2}}$ exists (Algorithm 5.2). This is the Newton solver used by Rasmussen and Williams (2006) and Margossian et al. (2020b), except that we do not assume W is diagonal. One particularly useful detail is that at each Newton step we compute

$$\mathbf{a} = K^{-1}\theta, \tag{5.3}$$

without actually inverting K. The vector $\mathbf{a}$ is then used to compute the objective function and during the adjoint-differentiation step (Section 5.4.5).

The Newton iteration can be augmented with a linesearch step. Let $\mathbf{a}_{\text{old}}$ and θ_{old} be respectively the a-vector and the guess for θ computed at the previous iteration. Then reducing the step length by a factor of 2 is done by updating $\mathbf{a}$, given that

$$\mathbf{a} \leftarrow \frac{\mathbf{a} + \mathbf{a}_{\text{old}}}{2} \iff \theta \leftarrow \frac{\theta_{\text{new}} + \theta_{\text{old}}}{2}.$$

This procedure is repeated until a chosen condition is met. We may for example require the objective function, Ψ, to decrease at each Newton iteration. Checking this condition is cheap, given that equipped with $\mathbf{a}$, the computation of the objective function is inexpensive.

If all our optimizers subscribe to a similar structure, we can write algorithms for the approximate marginal likelihood and its gradient which are (mostly) agnostic to which optimizer we use.

5.2.2 $B = I + K^{\frac{1}{2}T} W K^{\frac{1}{2}}$

This decomposition is proposed by Vanhatalo, Foster, and Hosack (2021) for the case where the likelihood is not log-concave meaning that W is not amiable to any straightforward decomposition.

Algorithm 5.2: Newton solver using $B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}}$

input: K, y, $\pi(y \mid \theta, \phi)$, Δ
$\theta \leftarrow \theta_0$ (initial guess)
$\Psi_{\text{new}} \leftarrow -\infty$ (initial objective)
while $(|\Psi_{\text{new}} - \Psi_{\text{old}}| \geq \Delta)$ **do**
  $W \leftarrow -\nabla_\theta \nabla_\theta \log \pi(y \mid \theta, \phi)$
  $L \leftarrow \text{Cholesky}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$
  $\mathbf{b} \leftarrow W\theta + \nabla_\theta \log \pi(y \mid \theta, \phi)$
  $\mathbf{a} \leftarrow \mathbf{b} - W^{\frac{1}{2}} L^T \setminus (L \setminus (W^{\frac{1}{2}} K \mathbf{b}))$
  $\theta \leftarrow K\mathbf{a}$
  $\Psi_{\text{old}} = \Psi_{\text{new}}$
  $\Psi_{\text{new}} \leftarrow -\frac{1}{2}\mathbf{a}\theta + \log \pi(y \mid \theta, \eta)$
end
$\log \pi(y \mid \phi) \leftarrow \Psi_{\text{new}} - \sum_i \log L_{ii}$
return: θ, $\log \pi_{\mathcal{G}}(y \mid \phi)$

On the other hand, we assume K can be safely inverted. Since K is a covariance matrix, $K^{\frac{1}{2}}$ can be computed using a Cholesky decomposition. B is still symmetric and admits a Cholesky decomposition, $LL^T = B$.

With this B-matrix, the Newton step is

$$\begin{aligned} \theta^{\text{new}} &= (K^{-1} + W)^{-1}(\nabla \log \pi(y \mid \theta, \eta) + W\theta) \\ &= K^{\frac{1}{2}}(I + K^{\frac{1}{2}T} W K^{\frac{1}{2}})^{-1} K^{\frac{1}{2}}(\nabla \log \pi(y \mid \theta, \eta) + W\theta) \\ &\triangleq K^{\frac{1}{2}}\mathbf{c}, \end{aligned}$$

which does not quite have the desired form $\theta^{\text{new}} = K\mathbf{a}$. We can however compute $\mathbf{a} = K^{-\frac{1}{2}}\mathbf{c}$, which is cheap given $K^{\frac{1}{2}}$ is triangular. Reconstructing $\mathbf{a}$ is not strictly necessary but it makes this approach more consistent with the Newton steps used for other B-matrices.

5.2.3 $B = I + KW$

The third and final decomposition makes no strong assumptions on K and W. The major drawback of this approach is that the B-matrix is not symmetric and therefore does not admit a Cholesky decomposition. Instead, we resort to the more expansive LU-decomposition (with partial-pivoting given B is invertible),

$$B = LU,$$

where L is lower-triangle, U upper-triangle, and which can then be used for solve methods with B and to compute $\log |B|$.

Algorithm 5.3 provides a general computation of the Laplace approximation, which admits a choice of B-matrix as an option.

5.3 Gradients with respect to ϕ and η

Lemma 5.3. *Without loss of generality, assume K only depends on ϕ, while the likelihood $\log \pi(y \mid \theta, \eta)$ only depends on η. Denote $\hat{\theta}$ the argument which maximizes $\pi(\theta \mid y, \phi, \eta)$. The derivative of the approximate marginal likelihood with respect to an element ϕ_j of ϕ is*

$$\begin{aligned} \frac{\partial}{\partial \phi_j} \log \pi_{\mathcal{G}}(y \mid \phi, \eta) &= -\frac{1}{2} \hat{\theta}^T K^{-1} \frac{\partial K}{\partial \phi_j} K^{-1} \hat{\theta} \\ &\quad -\frac{1}{2} \operatorname{trace}\left((W^{-1} + K)^{-1} \frac{\partial K}{\partial \phi_j} \right) \\ &\quad + \sum_{i=1}^{n} \frac{\partial}{\partial \hat{\theta}_i} \log \pi_{\mathcal{G}}(y \mid \phi, \eta) \cdot (I + KW)^{-1} \frac{\partial K}{\partial \phi_j}, \end{aligned} \tag{5.4}$$

where we assume the requisite derivatives exist and

$$\frac{\partial}{\partial \hat{\theta}_i} \log \pi_{\mathcal{G}}(y \mid \phi, \eta) = -\frac{1}{2} \operatorname{trace}\left((K^{-1} + W)^{-1} \frac{\partial W}{\partial \hat{\theta}_i} \right). \tag{5.5}$$

This result is worked out by Rasmussen and Williams (2006, Section 5.5.1). A similar result is obtained when differentiating with respect to η.

Algorithm 5.3: General Newton solver. *Writing all three Newton solvers in one algorithm highlights common features between the methods, which leads to lighter code and more general methods for calculating gradients.*

input: y, ϕ, η, $K(\phi)$, $\pi(y \mid \theta, \eta)$, B-matrix
$\theta = \theta_0$ (initial guess)
$\Psi_{\text{old}} = -\infty$ (initial objective)
while $(|\Psi_{\text{new}} - \Psi_{\text{old}}| > \Delta)$ **do**
    $W = -\nabla^2_\theta \log \pi(y \mid \theta, \eta)$
    $\mathbf{b} = W\theta + \nabla_\theta \log \pi(y \mid \theta, \eta)$
    if $(B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$ **then**
        $L = \text{Cholesky}(B)$
        $\mathbf{a} = \mathbf{b} - W^{\frac{1}{2}} L^T \backslash (L \backslash W^{\frac{1}{2}} K \mathbf{b}))$
    else if $(B = I + K^{\frac{1}{2}T} W K^{\frac{1}{2}})$ **then**
        $L = \text{Cholesky}(B)$
        $\mathbf{c} = L^T \backslash (L \backslash (K^{\frac{1}{2}} \mathbf{b}))$
        $\mathbf{a} = K^{\frac{1}{2}} \backslash \mathbf{c}$
    else if $(B = I + KW)$ **then**
        $(L, U) = \text{LUdecomposition}(I + KW)$
        $\mathbf{a} = \mathbf{b} - WU \backslash (L \backslash (K\mathbf{b}))$
    end
    $\Psi_{\text{old}} = \Psi_{\text{new}}$
    $\Psi_{\text{new}} = -\frac{1}{2}\mathbf{a}\theta + \log \pi(y \mid \theta, \eta)$
    if $(B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}}$ *or* $B = I + K^{\frac{1}{2}T} W K^{\frac{1}{2}})$ **then**
        $\log |B| = 2 \sum_i \log L_{ii}$
    else if $(B = I + KW)$ **then**
        $\log |B| = \sum_i \log L_{ii} + \sum_i \log U_{ii}$
    end
end
return: $\log \pi_{\mathcal{G}}(y \mid \phi, \eta) = \Psi_{\text{new}} - \frac{1}{2} \log |B|$

Lemma 5.4. *The derivative of the approximate marginal likelihood with respect to an element η_l of η is*

$$\begin{aligned}\frac{\partial}{\partial \eta_l}\log \pi_{\mathcal{G}}(y \mid \phi, \eta) &= \frac{\partial}{\partial \eta_l}\log \pi(y \mid \hat{\theta}, \eta) \\ &\quad + \frac{1}{2}\text{trace}\left((K^{-1}+W)^{-1}\frac{\partial \nabla^2_\theta \log \pi(y \mid \hat{\theta}, \eta)}{\partial \eta_j}\right) \\ &\quad + \sum_{i=1}^{n}\frac{\partial \log \pi_{\mathcal{G}}(y \mid \phi, \eta)}{\partial \hat{\theta}_i}\cdot (I+KW)^{-1}K\frac{\partial}{\partial \eta_l}\nabla_\theta \log \pi(y \mid \hat{\theta}, \eta), \quad (5.6)\end{aligned}$$

where we assume the requisite derivatives exist.

The proof is in the appendix at the end of this chapter. Several of the above expressions can be simplified in the special cases where W or K are diagonal.

Remark 5.1. *The derivative with respect to either η or θ decomposes into three terms:*

(i) an explicit term (partial derivative of the objective function),

(ii) the derivative of a log determinant, which becomes a trace,

(iii) a dot product with a gradient with respect to $\hat{\theta}$.

The expressions in Lemmas 5.3 and 5.4 are organized accordingly. I will use this organization in Section 5.4.4.

Once again we must contend with inverted matrices, namely $(K^{-1} + W)^{-1}$, which we have already dealt with when building the Newtons solver, and $(I + KW)^{-1}$. It turns out the latter can also be handled with decompositions of the B-matrix performed during the final Newton step, meaning no further Cholesky or LU decomposition is required. Indeed

$$(I + KW)^{-1} = I - K(K + W^{-1})^{-1}, \quad (5.7)$$

and $(K + W^{-1})^{-1}$ can be expressed in terms of any of three B-matrices we are working with:

$$\begin{aligned} R \triangleq (K + W^{-1})^{-1} &= W^{\frac{1}{2}}(I + W^{\frac{1}{2}}KW^{\frac{1}{2}})^{-1}W^{\frac{1}{2}} \\ &= W - WK^{\frac{1}{2}}(I + K^{\frac{1}{2}T}WK^{\frac{1}{2}})^{-1}K^{\frac{1}{2}T}W \\ &= W - W(I + KW)^{-1}KW. \end{aligned} \tag{5.8}$$

The last equality is superfluous, since we can directly handle the original matrix $(I + KW)^{-1}$ if we use $B = I + KW$. To run the adjoint-differentiated Laplace approximation by Margossian et al. (2020b), extended to handle derivatives with respect to η, methods to compute the following derivatives (analytically or otherwise) need to be provided:

- $\nabla_\theta \log \pi(y \mid \theta, \eta)$,
- $\nabla^2_\theta \log \pi(y \mid \theta, \eta)$,
- $\nabla^3_\theta \log \pi(y \mid \theta, \eta)$,
- $\nabla_\eta \log \pi(y \mid \theta, \eta)$,
- $\nabla_\eta \nabla_\theta \log \pi(y \mid \theta, \eta)$,
- $\nabla_\eta \nabla^2_\theta \log \pi(y \mid \theta, \eta)$.

5.4 Automatic differentiation of the likelihood

Fortunately we can eliminate the burden of analytically computing derivatives by applying automatic differentiation, as was already done to remove calculations of $\partial K / \partial \phi$.

5.4.1 Allowed operations with automatic differentiation

Consider a function

$$\begin{aligned} f: \quad & \mathbb{R}^m \to \mathbb{R}^n \\ & x \to f(x). \end{aligned}$$

A forward mode sweep of automatic differentiation allows us to compute the directional derivative

$$\frac{\partial f}{\partial x} \cdot v$$

for an initial tangent $v \in \mathbb{R}^m$. A reverse mode sweep, on the other hand, computes the co-directional derivative

$$w^T \cdot \frac{\partial f}{\partial x}$$

for an initial cotangent $w \in \mathbb{R}^n$. We can compute higher-order derivatives by iteratively applying sweeps of automatic differentiation. This procedure is straightforward for forward mode, less so for reverse mode. When doing multiple sweeps in Stan, we may only use a single reverse mode sweep and this sweep must be the final sweep (Carpenter et al. 2015).

5.4.2 Principles for an efficient implementation

I observe the following principles to write an efficient implementation:

1. *Contraction.* Avoid computing full Jacobian matrices. Instead, only compute directional derivatives by contracting Jacobian matrices with the right tangent or cotangent vectors.

2. *Linearization.* In the original algorithm, identify linear operators, Φ, which take in and return a derivative, e.g.

$$\frac{\partial c}{\partial a_i} = \Phi\left(\frac{\partial b}{\partial a_i}\right) \in \mathbb{R}$$

   Then rather than compute the derivatives one element at a time, compute $\Phi(b)$ and apply a

reverse mode sweep of automatic differentiation to obtain the desired gradient. This can be seen as a strategy to identify the directions along which to compute derivatives.

3. *Sparse computation.* For sparse objects of derivatives, only compute the non-zero elements, again by carefully picking the directions along which we compute the derivatives.

At each step of the Newton solver, I compute the full negative Hessian, W, even though this is not strictly necessary. This is because (i) the Hessian is typically sparse and therefore relatively cheap to compute, and (ii) W is used many times both in the computation of the Laplace approximation and its differentiation.

5.4.3 Differentiating the negative Hessian, W

Most LGMs admit a likelihood with a block-diagonal or even diagonal Hessian. In typical automatic differentiation frameworks, the cost of computing a block-diagonal Hessian with block size $m \times m$ is $2m$ sweeps. This is notably the case with Stan as I will demonstrate. Somewhat contrary to general wisdom, it is actually possible to compute a Hessian matrix with a single reverse mode sweep using a graphical model and an algebraic model (Gower and Mello 2011). I have not investigated this approach but believe it is promising[3].

Diagonal Hessian

Consider a function

$$\begin{aligned} f: \quad \mathbb{R}^n &\to \mathbb{R} \\ \theta &\to f(\theta). \end{aligned}$$

[3] ... following a conversation with Robert Gower.

Suppose for starters that f admits a diagonal Hessian. To get the non-zero elements of $\nabla^2_\theta \log \pi(y \mid \theta, \eta)$, we only need to compute the Hessian-vector product,

$$[\nabla^2_\theta \log \pi(y \mid \theta, \eta)] \cdot \mathbf{v},$$

where $\mathbf{v} = (1, 1, ..., 1) \in \mathbb{R}^n$. This operation can be done using one forward mode and one reverse mode sweep of automatic differentiation. In details: we introduce the auxiliary function

$$\begin{aligned} g: \quad \mathbb{R}^n &\to \mathbb{R}, \\ \theta &\to \nabla_\theta \log \pi(y \mid \theta, \eta) \cdot \mathbf{v}, \end{aligned}$$

which returns the sum of the partial derivatives and can be computed using one forward sweep. Then $\mathbf{a}^T[\nabla_\theta g(\theta)]$ can be computed using one reverse mode sweep, where $\mathbf{a} = (1)$ is a vector of length 1 which contracts the $1 \times n$ gradient into a scalar. Concisely

$$[\nabla^2_\theta \log \pi(y \mid \theta, \eta)] \cdot \mathbf{v} = \mathbf{a}^T \cdot \nabla_\theta (\nabla_\theta \log \pi(y \mid \theta, \eta) \cdot \mathbf{v}). \tag{5.9}$$

A similar procedure can be used to compute the diagonal tensor of third-order derivatives, noting that it too only contains n non-zero elements:

$$\text{Diag}\left(\nabla^3_\theta \log \pi(y \mid \theta, \eta)\right) = \mathbf{a}^T \cdot \nabla_\theta \left(\nabla_\theta \left(\nabla_\theta f \cdot \mathbf{v}\right) \cdot \mathbf{v}\right). \tag{5.10}$$

This operation is performed in three sweeps, starting from the inner-most parenthesis and expanding out.

Block-diagonal Hessian

Now suppose f admits a 2×2 block diagonal Hessian. I use the 2×2 case to develop some intuition before generalizing. Consider the vectors $\mathbf{v_1} = (1, 0, 1, 0, ..., 1, 0) \in \mathbb{R}^n$ and $\mathbf{v_2} = (0, 1, ..., 0, 1) \in$

$\mathbb{R}^n$. Then

$$\nabla^2 f \cdot \mathbf{v_1} = \begin{pmatrix} \partial^2_{\theta_1^2} f & \partial^2_{\theta_1\theta_2} f & 0 & 0 & \dots \\ \partial^2_{\theta_1\theta_2} f & \partial^2_{\theta_2^2} f & 0 & 0 & \dots \\ 0 & 0 & \partial^2_{\theta_3^2} f & \partial^2_{\theta_3\theta_4} f & \dots \\ 0 & 0 & \partial^2_{\theta_3\theta_4} f & \partial^2_{\theta_4^2} f & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ \dots \end{pmatrix} = \begin{pmatrix} \partial^2_{\theta_1^2} f \\ \partial^2_{\theta_1\theta_2} f \\ \partial^2_{\theta_3^2} f \\ \partial^2_{\theta_3\theta_4} f \\ \dots \end{pmatrix}$$

and

$$\nabla^2 f \cdot \mathbf{v_2} = \begin{pmatrix} \partial^2_{\theta_1^2} f & \partial^2_{\theta_1\theta_2} f & 0 & 0 & \dots \\ \partial^2_{\theta_1\theta_2} f & \partial^2_{\theta_2^2} f & 0 & 0 & \dots \\ 0 & 0 & \partial^2_{\theta_3^2} f & \partial^2_{\theta_3\theta_4} f & \dots \\ 0 & 0 & \partial^2_{\theta_3\theta_4} f & \partial^2_{\theta_4^2} f & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ \dots \end{pmatrix} = \begin{pmatrix} \partial^2_{\theta_1\theta_2} f \\ \partial^2_{\theta_2^2} f \\ \partial^2_{\theta_3\theta_4} f \\ \partial^2_{\theta_4^2} f \\ \dots \end{pmatrix}.$$

These two Hessian-vector products return all the non-zero elements of the Hessian. Thus computing the full Hessian requires 2 times the effort to evaluate a diagonal Hessian. Specifically, we first compute $\nabla f \cdot \mathbf{v_1}$ using a forward sweep, followed by one reverse mode sweep, and repeat the process with $\mathbf{v_2}$. The total cost for computing a 2×2 block-diagonal Hessian is thus 4 sweeps[4]. The average cost of the sweeps can be reduced by exploiting the symmetry of the Hessian (e.g Griewank and Walther 2008; Gower and Mello 2011).

In the $m \times m$ block-diagonal case, we need to construct m initial tangents,

$$\begin{aligned} \mathbf{v_1} &= (\underbrace{1,0,\cdots,0}_{m},\underbrace{1,0,\cdots,0}_{m},\cdots,\underbrace{1,0,\cdots,0}_{m}) \\ \mathbf{v_2} &= (\underbrace{0,1,\cdots,0}_{m},\underbrace{0,1,\cdots,0}_{m},\cdots,\underbrace{0,1,\cdots,0}_{m}) \\ &\vdots \\ \mathbf{v_m} &= (\underbrace{0,0,\cdots,1}_{m},\underbrace{0,0,\cdots,1}_{m},\cdots,\underbrace{0,0,\cdots,1}_{m}). \end{aligned} \tag{5.11}$$

[4]If we could start with reverse mode and then run forward mode, we could imagine computing the Hessian in 3 sweeps.

The elements of the Hessian are then computed using one forward mode along m directions, followed by a reverse mode.

5.4.4 Differentiating the approximate marginal density

Differentiating the approximate marginal density with respect to the hyperparameters ϕ and η requires computing three terms: (i) an explicit term, (ii) a differentiated log determinant, and (iii) a dot product with a derivative with respect to θ (Lemmas 5.3 and 5.4, and Remark 5.1). The terms for ϕ are handled by Rasmussen and Williams (2006) and Margossian et al. (2020b), so I will focus on η. The explicit term for η is simply the partial derivative of $\log \pi(y \mid \theta, \eta)$ with respect to η which can be obtained using one reverse mode sweep of automatic differentiation.

Differentiating the log determinant

Differentiating a log determinant produces a trace (see appendix). From Lemmas 5.3 and 5.4 we see that we must evaluate a higher-order derivative inside a trace with respect to both η and $\hat{\theta}$,

$$\text{trace}\left((K^{-1} + W)^{-1}\frac{\partial W}{\partial \eta_l}\right) \quad \text{and} \quad \text{trace}\left((K^{-1} + W)^{-1}\frac{\partial W}{\partial \hat{\theta}_j}\right).$$

Here I apply the linearization principle of automatic differentiation. Let $A = (K^{-1} + W)^{-1}$ and note that $\text{trace}(AW)$ is a linear function of W. Once we *evaluate* and *tape* $\text{trace}(AW)$, it remains to do one reverse mode sweep to obtain a gradient with respect to η and $\hat{\theta}$. I use the term *tape* to indicate that we must store the expression graph for $\text{trace}(AW)$ in order to perform the reverse mode sweep, given W depends on both η and $\hat{\theta}$.

Now

$$\text{trace}(AW) = \sum_i \sum_k A_{ik} W_{ki} = \sum_i \sum_k A_{ik} W_{ik}, \tag{5.12}$$

where the second equality follows from the fact W is symmetric. Without loss of generality, assume W is block diagonal, with block size $m \times m$. As in our calculations of the Hessian, we start with a forward mode sweep along an initial tangent v_1. Next we do a second forward mode sweep with

an initial tangent which contains the first column of each $m \times m$ block in A,

$$\mathbf{w}_1 = (A_{1,1}, A_{2,1}, \cdots, A_{m,1}, A_{m+1,m+1}, A_{m+2,m+1}, \cdots). \tag{5.13}$$

That is $\mathbf{w}_1$ contains the colored elements in the below representation of A,

$$\begin{bmatrix}
A_{1,1} & A_{1,2} & \cdots & \cdots & \cdots & \cdots \\
A_{2,1} & A_{2,2} & \cdots & \cdots & \cdots & \cdots \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots \\
A_{m,1} & A_{m,2} & \cdots & \cdots & \cdots & \cdots \\
A_{m+1,1} & \cdots & \cdots & A_{m+1,m+1} & A_{m+1,m+2} & \cdots \\
\cdots & \cdots & \cdots & A_{m+2,m+1} & A_{m+2,m+2} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\
A_{2m,1} & \cdots & \cdots & A_{2m,m+1} & A_{2m,m+2} & \cdots \\
A_{2m+1,1} & \cdots & \cdots & A_{2m+1,m+1} & A_{2m+1,m+2} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}.$$

To obtain the full trace, we repeat this process m times using the appropriate initial tangents, $\mathbf{v}_j$ and $\mathbf{w}_j$, $j \in [m]$. It may be surprising that computing a scalar would require so many sweeps but this seems to be because there is an implicit matrix-matrix multiplication, which prevents us from describing the sum given by the trace using only two tangent vectors. In other words, we cannot describe all the relevant nm elements of A using only $2n$ elements, unless A is low-rank.

To avoid repeating these calculations for $\hat{\theta}$ and η, I compute the forward mode sweeps with respect to both $\hat{\theta}$ and η at once. To do so, I append T 0's to v_j and w_j, where T is the dimension of

η, e.g.

$$\begin{aligned}
\mathbf{v}_1^* &= (\underbrace{1,0,\cdots,0}_{m},\underbrace{1,0,\cdots,0}_{m},\cdots,\underbrace{0,0,\cdots,0}_{T}), \\
\mathbf{w}_1^* &= (\underbrace{A_{1,1},A_{2,1}}_{m},\cdots,\underbrace{A_{m,1},A_{m+1,m+1},A_{m+2,m+1}}_{m},\cdots,\underbrace{0,0,\cdots,0}_{T}).
\end{aligned} \tag{5.14}$$

Unfortunately this procedure requires computing A, a potentially expansive operation. The cost can be reduced by only computing the block-diagonal elements of A.

Differentiating the dot product

The dot product results from the chain rule, which requires us to multiply $\partial \log \pi_{\mathcal{G}}(y \mid \phi, \eta)/\partial \hat{\theta}_j$ and $\partial \hat{\theta}_j / \partial \eta_l$. Let

$$s_2 \triangleq \nabla_{\hat{\theta}} \log \pi_{\mathcal{G}}(y \mid \phi, \eta) = -\frac{1}{2}\text{trace}\left((K^{-1} + W)^{-1}\right). \tag{5.15}$$

s_2 can be computed using the method outlined in the previous section. Once again these calculations are subject to useful simplifications when W is diagonal.

Per Lemma 5.4, the dot product is taken with

$$\begin{aligned}
s_3 &\triangleq (I + KW)^{-1} K \frac{\partial}{\partial \eta_l} \nabla_\theta \log \pi(y \mid \hat{\theta}, \eta) \\
&= (I - KR) K \frac{\partial}{\partial \eta_l} \nabla_\theta \log \pi(y \mid \hat{\theta}, \eta),
\end{aligned} \tag{5.16}$$

where the second equality follows from Equation 5.7 and R can safely be computed using any of the three B-matrices introduced in Section 5.2. Here too I apply the linearization principle and consider the function

$$\tilde{s}_3 \triangleq (I - KW)^{-1} K \nabla_{\hat{\theta}} \log \pi(y \mid \hat{\theta}, \eta) = (I - KR) K \nabla_{\hat{\theta}} \log \pi(y \mid \hat{\theta}, \eta). \tag{5.17}$$

The scalar $s_2^T \tilde{s}_3$ can be computed using one forward sweep with respect to $\hat{\theta}$ and initial tangent

$$\mathbf{u} = K(I - KW)^{-1} s_2 = K(I - KR)s_2, \tag{5.18}$$

where I drop the transpose on the term before s_2 due to symmetry. It then remains to apply a reverse mode sweep with respect to η.

5.4.5 General adjoint-differentiation

We are finally ready to write down the general adjoint-differentiation (Algorithm 5.4).

Let

$$\mathrm{AD}(f, [\mathrm{fwd}, \mathbf{v}, x])$$

be a single forward mode sweep which returns $\partial f / \partial x \cdot \mathbf{v}$. Similarly, $\mathrm{AD}(f, [\mathrm{rev}, \mathbf{w}, x])$ returns $\mathbf{w}^T \cdot \partial f / \partial x$. Furthermore, we can encode multiple sweeps, for example $\mathrm{AD}(f, [\mathrm{fwd}, \mathbf{v}, x], [\mathrm{rev}, \mathbf{w}, x])$, where sweeps are executed from the left to the right.

5.5 Posterior draws for the marginalized out parameters

I review a procedure to generate posterior draws for the latent Gaussian variable, θ, following Rasmussen and Williams (2006), and show how this approach fits in the B-matrix framework.

After generating posterior draws for the hyperparameters, (ϕ, η), we recover posterior draws for θ using the Laplace approximation:

$$\begin{aligned} (\phi, \eta) &\sim \pi(\phi, \eta \mid y), \\ \theta &\sim \pi_{\mathcal{G}}(\theta \mid y, \phi, \eta) \approx \pi(\theta \mid y, \phi, \eta). \end{aligned}$$

We may also want to draw new latent Gaussian variables, θ^*, given a prior distribution $\pi(\theta, \theta^*)$. In a Gaussian process, the covariance matrix, K, is typically parameterized by ϕ and a covariate X. Hence, for a new set of observations, the new prior covariance would be $K(\phi, X^*)$. Let $K^* =$

Algorithm 5.4: General adjoint-differentiation. *Note: additional specialized steps can be taken in the case where W or K is diagonal.*

input: y, ϕ, η, $\pi(y \mid \theta, \eta)$
saved input from the Newton solver: $\hat{\theta}$, $\mathbf{a}$, $\nabla_{\hat{\theta}} \log \pi(y \mid \hat{\theta}, \eta)$, K, B-matrix
  $W^{\frac{1}{2}}$, L ▷ $B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}}$, $LL^T = B$
  W, $K^{\frac{1}{2}}$, L ▷ $B = I + K^{\frac{1}{2}T} W K^{\frac{1}{2}}$, $LL^T = B$
  W, L, U ▷ $B = I + KW$, $LU = B$
if $(B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$ **then**
  $R \leftarrow W^{\frac{1}{2}} L^T \backslash (L \backslash W^{\frac{1}{2}})$ ▷ $R = (K + W^{-1})^{-1}$
  $C \leftarrow L \backslash (W^{\frac{1}{2}} K)$
  $A \leftarrow K - C^T C$ ▷ A contains initial tangents for log det. derivative.
else if $(B = I + K^{\frac{1}{2}T} W K^{\frac{1}{2}})$ **then**
  $D \leftarrow L \backslash K^{\frac{1}{2}} W$
  $R \leftarrow W - D^T D$
  $C \leftarrow L \backslash K^{\frac{1}{2}T}$
  $A \leftarrow C^T C$
else if $(B = I + KW)$ **then**
  $R \leftarrow W - WCW$
  $C \leftarrow U \backslash L \backslash K$
  $A \leftarrow K - KWC$
end
$V \leftarrow [\mathbf{v}_1^*, \mathbf{v}_2^*, \cdots, \mathbf{v}_m^*]$ ▷ Initial tangents, Equations 5.11 and 5.14
$W \leftarrow [\mathbf{w}_1^*, \mathbf{w}_2^*, \cdots, \mathbf{w}_m^*]$ ▷ Initial tangents using A, Equations 5.13 and 5.14
$s \leftarrow 0$
for $i \in \{1, 2, \cdots, m\}$ **do**
  $s \leftarrow s + \mathrm{AD}(\log \pi(y \mid \hat{\theta}, \eta), [\mathrm{fwd}, \mathbf{v}_i^*, \vartheta], [\mathrm{fwd}, \mathbf{w}_i^*, \vartheta])$ ▷ $\vartheta \triangleq (\hat{\theta}, \eta)$
end
$s \leftarrow \mathrm{AD}(s, [\mathrm{rev}, 1, \vartheta])$
$s_2 \leftarrow s[1 : n]$ ▷ Gradient of log determinant w.r.t $\hat{\theta}$
$s_2' \leftarrow s[(n+1) : (n+T)]$ ▷ Gradient of log determinant w.r.t η
$\Omega^T = \frac{1}{2}\mathbf{a}\mathbf{a}^T - \frac{1}{2}R + (s_2 + RKs_2)[\nabla\hat{\theta} \log \pi(y \mid \hat{\theta}, \eta)]^T$ ▷ (Margossian et al. 2020b, 3.2)
$\nabla_\phi \log \pi_{\mathcal{G}}(y \mid \phi, \eta) = \mathrm{AD}(K, [\mathrm{rev}, \Omega, \phi])$
if $(B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$ *or* $(B = I + K^{\frac{1}{2}T} W K^{\frac{1}{2}})$ **then**
  $\mathbf{u} = K(I - KR)s_2$ ▷ Initial tangent, Equation 5.18
else if $(B = I + KW)$ **then**
  $\mathbf{u} = U \backslash L \backslash K$ ▷ Initial tangent, Equation 5.18
end
$\nabla_\eta \log \pi_{\mathcal{G}}(y \mid \phi, \eta) = \mathrm{AD}(\log \pi(y \mid \hat{\theta}, \eta), [\mathrm{rev}, 1, \eta]) + s_2'$ ▷ Lemma 5.4
  $+\mathrm{AD}(\log \pi(y \mid \hat{\theta}, \eta), [\mathrm{fwd}, \mathbf{u}, \theta], [\mathrm{rev}, 1, \eta])$
return: $\nabla_\phi \log \pi_{\mathcal{G}}(y \mid \phi, \eta)$, $\nabla_\eta \log \pi_{\mathcal{G}}(y \mid \phi, \eta)$

$K(X, X^*)$ be the prior covariance between θ and θ^*, i.e. the upper-right block in the complete covariance matrix for (θ, θ^*). Similarly note that $K = K(X, X)$. The mean of the approximating normal is then

$$\mathbb{E}_{\mathcal{G}}(\theta^* \mid X, y, \phi, X^*) = K^* K^{-1} \hat{\theta} = K^* \nabla_{\hat{\theta}} \log \pi(y \mid \hat{\theta}, \eta).$$

The second equality follows from the fact $\hat{\theta}$ is the mode of the conditional distribution, $\pi(\theta \mid y, \phi, \eta)$, meaning

$$\nabla \Psi = 0 \implies \hat{\theta} = K \nabla_{\hat{\theta}} \log \pi(y \mid \hat{\theta}, \eta).$$

Furthermore,

$$\begin{aligned} \Sigma_{\mathcal{G}}(\theta^* \mid X, y, \phi, X^*) &= K^* - K^*(K + W^{-1})^{-1} K^* \\ &= K^*(I - RK^*), \end{aligned}$$

where R can be computed using any of three B-matrices. The procedure is summarized in Algorithm 5.5. Here the computation is dominated by the evaluation of the covariance matrix, $\Sigma_{\mathcal{G}}$.

5.6 Numerical experiment

The integrated Laplace approximation is a well established method with success in many applications; see Rue et al. 2017 and references therein. A handful of papers study the potential of the integrated Laplace approximation when combined with MCMC, which is useful to expand the range of priors we can use (e.g. Gómez-Rubio and Rue 2018; Monnahan and Kristensen 2018; Margossian et al. 2020b). Applications to less conventional likelihoods include the work by Vanhatalo, Jylänki, and Vehtari (2009), Jylänki, Vanhatalo, and Vehtari (2011), Joensuu et al. (2012), Riihimäki and Vehtari (2014), and Vanhatalo, Foster, and Hosack (2021). I examine the application of Hamiltonian Monte Carlo (HMC) using a general adjoint-differentiated Laplace approximation to a population pharmacokinetic model. This model falls outside the traditional framework of

Algorithm 5.5: Posterior draws for latent Gaussian θ^*

intput: $y, \phi, \eta, X, X^*, K(\phi, X, X^*), \pi(y \mid \theta, \eta)$

saved input from the Newton solver: $\hat{\theta}$, W, K, $\nabla_{\hat{\theta}} \log \pi(y \mid \hat{\theta}, \eta)$

$W^{\frac{1}{2}}, L$ ▷ $B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}}$, $LL^T = B$

$W, K^{\frac{1}{2}}, L$ ▷ $B = I + K^{\frac{1}{2}T} W K^{\frac{1}{2}}$, $LL^T = B$

W, L, U ▷ $B = I + KW$, $LU = B$

$K^* \leftarrow K(X, X^*)$

$\mu^* \leftarrow K^* \nabla_{\hat{\theta}} \log \pi(y \mid \theta, \eta)$

if $(B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$ **then**

  $V \leftarrow L \backslash W^{\frac{1}{2}} K^*$

  $\Sigma^* \leftarrow K^* - V^T V$

else if $(B = I + K^{\frac{1}{2}T} W K^{\frac{1}{2}})$ **then**

  $D \leftarrow L \backslash K^{\frac{1}{2}} W$

  $R \leftarrow W - D^T D$

  $\Sigma^* \leftarrow K^*(I - RK^*)$

else if $(B = I + KW)$ **then**

  $\Sigma^* = K^* - K^*(W - WU \backslash L \backslash KW) K^*$

end

$\theta^* \sim \text{Normal}(\mu^*, \Sigma^*)$

return: θ^*.

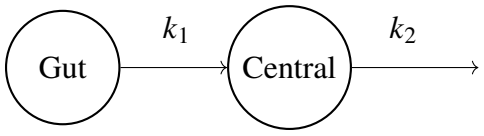

Figure 5.2: One compartment model with first-order absorption from the gut. *The drug enters the body through the gut (bolus dose) and is then absorb in to the central compartment (blood and tissues) at a rate k_1. Over time, the drug is cleared at a rate k_2.*

LGMs and is meant to stress-test our algorithm, while taking advantage of its flexibility.

The one-compartment pharmacokinetic model with a first-order absorption from the gut describes the diffusion of an orally administered drug compound in the patient's body (Figure 5.2). Two parameters of interest are the absorption rate, k_1, and the clearance rate, k_2. The population model endows each patient with their own rate parameters, subject to a hierarchical normal prior. Of interest are the population parameters, $k_{1,\text{pop}}$ and $k_{2,\text{pop}}$, and their corresponding population standard deviation, $\tau_{1,\text{pop}}$ and $\tau_{2,\text{pop}}$.

The full model is given below

$$
\begin{aligned}
&\textit{hyperpriors} \\
k_{1,\text{pop}} &\sim \text{Normal}(2, 0.5) \\
k_{2,\text{pop}} &\sim \text{Normal}(1, 0.5) \\
\tau_1 &\sim \text{Normal}^+(0, 1) \\
\tau_2 &\sim \text{Normal}^+(0, 1) \\
\sigma &\sim \text{Normal}^+(0, 1)
\end{aligned}
$$

$$
\begin{aligned}
&\textit{hierarchical priors} \\
k_1^n &\sim \text{Normal}(k_{1,\text{pop}}, \tau_1) \\
k_2^n &\sim \text{Normal}(k_{2,\text{pop}}, \tau_2)
\end{aligned}
$$

$$
\begin{aligned}
&\textit{likelihood} \\
y_n &\sim \text{Normal}\left(m_{\text{cent}}(t, k_1^n, k_2^n), \sigma\right),
\end{aligned}
$$

where the second argument for the Normal distribution is the standard deviation and Normal$^+$ is a normal distribution truncated at 0, with non-zero density only over positive values. m_{cent} is computed by solving the ordinary differential equation (ODE),

$$
\begin{aligned}
\frac{\mathrm{d}m_{\text{gut}}}{\mathrm{d}t} &= -k_1 m_{\text{gut}} \\
\frac{\mathrm{d}m_{\text{cent}}}{\mathrm{d}t} &= k_1 m_{\text{gut}} - k_2 m_{\text{cent}},
\end{aligned}
$$

which admits an analytical solution, when $k_1 \neq k_2$,

$$\begin{aligned} m_{\text{gut}}(t) &= m^0_{\text{gut}} \exp(-k_1 t) \\ m_{\text{cent}}(t) &= \frac{\exp(-k_2 t)}{k_1 - k_2} \left(m^0_{\text{gut}} k_1 (1 - \exp[(k_2 - k_1)t] + (k_1 - k_2) m^0_{\text{cent}}) \right). \end{aligned}$$

Here m^0_{gut} and m^0_{cent} are the initial conditions at time $t = 0$. Each patient receives one dose at time $t = 0$, and measurements are taken at times $t = (0.083, 0.167, 0.25, 1, 2, 4)$ for a total of 6 observations per patient. Data is simulated over 10 patients.

In our notation for LGMs,

$$\begin{aligned} \phi &= (\tau_1, \tau_2), && \textit{Hyperparameters for prior covariance} \\ \eta &= \sigma, && \textit{Hyperparameters for likelihood} \\ \theta &= (\mathbf{k}_1, \mathbf{k}_2), && \textit{Latent Gaussian variable} \end{aligned}$$

where $\mathbf{k}_i$ is a vector with the patient level absorption and clearance parameters. This model violates all three assumptions for the classical integrated Laplace approximation (Section 5.1.1). Indeed, $\eta \neq \emptyset$, the Hessian is block-diagonal, with block size 2×2 (once we organize the parameters to minimize the block size), and the likelihood is not log-concave. W does not admit a matrix square-root and the B-matrix, $B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}}$, cannot be used. The alternative B-matrices both work, with $B = I + KW$ being slightly more stable. In both cases, I use a linesearch step.

The optimization problem underlying the Laplace approximation is difficult, resulting in slow computation and numerical instability as the Markov chain explores the parameter space. This problem is particularly acute during the warmup phase. By contrast, the optimizer behaves reasonably well in the examples studied by Margossian et al. (2020b), all of which employ a general linear model likelihood.

I use HMC applied to the full parameter space, or *full HMC*, as a benchmark. Full HMC does quite well on this example, meaning that, unlike in other cases, the posterior geometry is well-

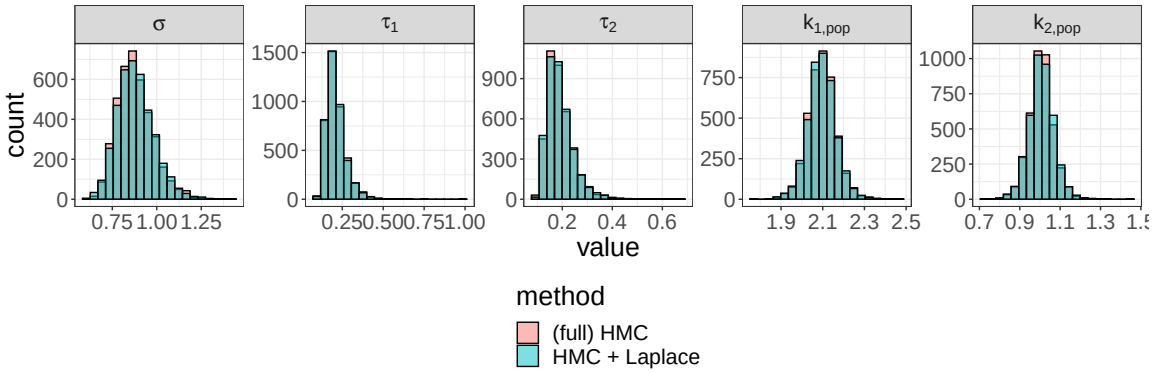

Figure 5.3: Posterior samples obtained with full HMC and the integrated Laplace approximation on a population pharmacokinetic model

behaved. Despite the unorthodox nature of the likelihood, the integrated Laplace approximation produces posterior estimates of the hyperparameters which are in close agreement with full HMC (Figure 5.3). The bias introduced by the approximation is negligible, meaning we can compare the effective sample size (ESS) estimated for both samplers. With both methods, the chain's autocorrelation is relatively small. The integrated Laplace approximation generates a larger ESS, exceeding the actual sample size for σ, meaning the Monte Carlo estimators are super-efficient (Figure 5.4). This is because marginalization allows us to run HMC on a posterior distribution with a well-behaved geometry. That said, the excessively slow optimization for this non-convex problem means the performance of full HMC is vastly superior as measured by the ESS / second (Figure 5.5). More generally, in cases where the posterior geometry does not frustrate MCMC, we may expect full HMC to outperform the integrated Laplace approximation, especially if the underlying optimization problem is difficult.

5.7 Discussion

I propose a generalization of the adjoint-differentiated Laplace approximation by (i) expanding the algorithm to work on three Newton solvers, using B-matrices as unifying framework, and (ii) fully automating the differentiation of the likelihood. The resulting implementation is more flexible and slightly faster than the original method which uses analytical derivatives. This greatly

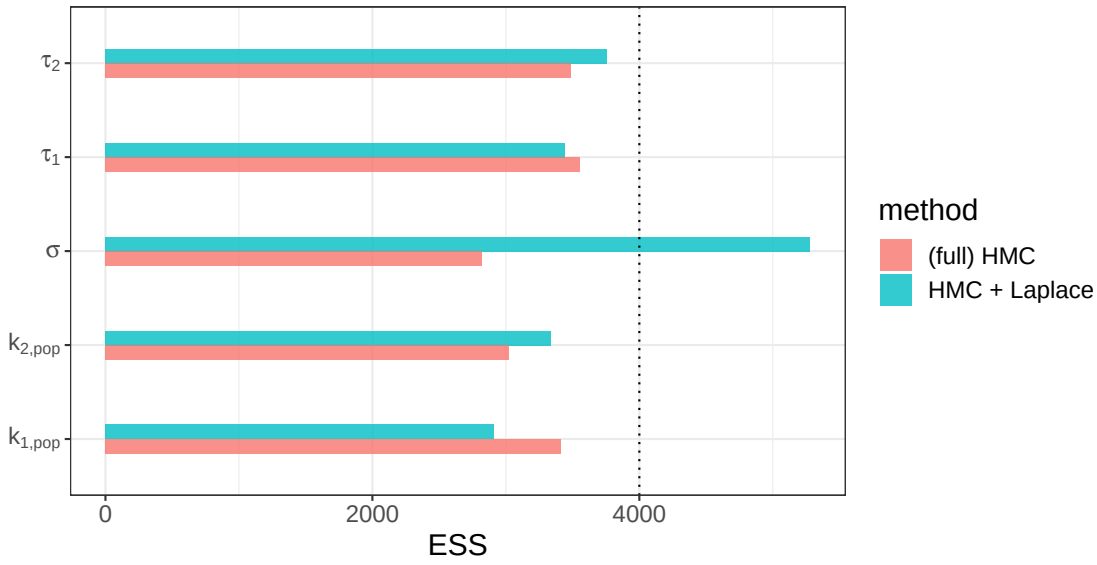

Figure 5.4: Effective sample size obtained with full HMC and the integrated Laplace approximation on a population pharmacokinetic model. *The dotted line represents the actual sample size.*

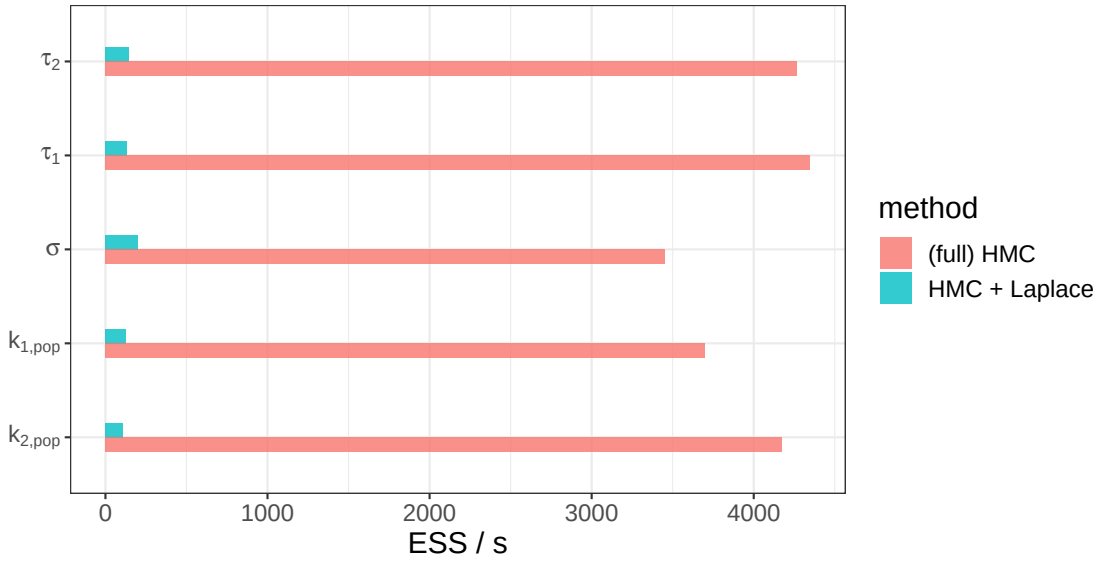

Figure 5.5: Effective sample size per second obtained with full HMC and the integrated Laplace approximation on a population pharmacokinetic model

facilitates implementing the method in software for a broad range of likelihoods. The proposed implementation also makes it straightforward to explore less conventional models such as the population pharmacokinetic model in Section 5.6.

Once we consider a rich enough space of models, it becomes clear that the integrated Laplace approximation confronts us to three challenges:

(i) *Quality of the approximation*. As we move away from log-concave likelihoods and the theory that supports them, how can we asses whether the approximation is reasonable? The population pharmacokinetic example showcases the approximation can be good in an unorthodox setting. In other cases with a non-log-concave likelihood, the conditional posterior of the latent variable can be multimodal and therefore not well approximated by a Laplace approximation; this problem typically also incurs challenges when computing the approximation (e.g. Vanhatalo, Jylänki, and Vehtari 2009). Developing inexpensive diagnostic tools to confirm this without running a golden benchmark remains an open problem. Candidate diagnostics include importance sampling (Vehtari et al. 2019), leave-one-out cross-validation (Vehtari et al. 2016), and simulation based calibration (Talts et al. 2020).

(ii) *Optimization*. Can we efficiently compute the Laplace approximation? The answer to this question varies between likelihoods, and furthermore the hyperparameter values we encounter as we run MCMC.

(iii) *Differentiation*: One persistent limitation is that any operation used to evaluate the log likelihood must support both forward and reverse mode automatic differentiation in order to compute higher-order derivatives. For implicit functions, we will likely need higher-order adjoint methods to insure an efficient implementation.

The general adjoint-differentiated Laplace approximation is prototyped in an experimental branch of Stan. It is straightforward to embed the Laplace approximation in HMC, as I did in Section 5.6, and furthermore in any gradient-based inference algorithm supported by Stan including penalized optimization, automatic differentiation variational inference (Kucukelbir et al. 2017),

and the prototype pathfinder (Zhang et al. 2021). Studying the use of different inference algorithms for the hyperparameter presents an exciting avenue of research.

5.8 Code

The prototype general adjoint-differentiated Laplace approximation for the Stan-math C++ library can be found at https://github.com/stan-dev/math, under the branch `experimental/lapl` Code to expose the suite of Laplace functions to the Stan language can be found at https://github.com/stan-dev/stanc3/tree/update/laplace-rng. Instructions on installing Stan with the relevant branches, along with several examples, can be found at https://github.com/SteveBronder/laplace_testing.

Acknowledgment

I am grateful to Aki Vehtari, Jarno Vanhatalo, and Dan Simpson for many helpful discussions. I am indebted to Steve Bronder who reviewed my C++ code and refactored it in order to improve the API and its overall quality. I thank Ben Bales for helping me understand the use of higher-order automatic differentiation in Stan, and Rok Češnovar for his help with Stan's transpiler.

5.9 Appendix: proof of Lemma 5.4

Following the steps from Rasmussen and Williams (2006) but differentiating with respect to η,

$$\frac{\mathrm{d}}{\mathrm{d}\eta_j} \log \pi_{\mathcal{G}}(y \mid \phi) = \frac{\partial \log \pi_{\mathcal{G}}(y \mid \phi)}{\partial \eta_j} + \sum_i \frac{\partial \log \pi_{\mathcal{G}}(y \mid \phi)}{\partial \hat{\theta}_i} \frac{\mathrm{d}\hat{\theta}_i}{\mathrm{d}\eta_j}. \tag{5.19}$$

The first "explicit" term is

$$\frac{\partial \log \pi_{\mathcal{G}}(y \mid \phi)}{\partial \eta_j} = \frac{\partial \log \pi(y \mid \hat{\theta}, \eta)}{\partial \eta_j} - \frac{1}{2} \frac{\partial \log |K||K + W^{-1}|}{\partial \eta_j}. \tag{5.20}$$

We can work out the first term analytically (or with automatic differentiation). Next we consider the following handy lemma.

Lemma 5.5. *Rasmussen and Williams 2006, equation A.15 For an invertible and differentiable natrix A,*

$$\frac{\partial}{\partial \theta} \log |A| = \operatorname{tr}\left(A^{-1} \frac{\partial A}{\partial \theta}\right).$$

Hence

$$\begin{aligned}
\frac{\partial \log |K||K^{-1} + W|}{\partial \eta_j} &= \frac{\partial \log |K^{-1} + W|}{\partial \eta_j} \\
&= \operatorname{tr}\left((K^{-1} + W)^{-1} \frac{\partial W}{\partial \eta_j}\right) \\
&= -\operatorname{tr}\left((K^{-1} + W)^{-1} \frac{\partial \nabla_\theta^2 \log \pi(y \mid \theta, \eta)}{\partial \eta_j}\right)
\end{aligned}$$

where we recall that η parameterizes W but not K.

Now for the implicit term, we differentiate the self-consistent equation

$$\hat{\theta} = K \nabla_\theta \log \pi(y \mid \theta, \eta).$$

Thus

$$\begin{aligned}
\frac{\partial \hat{\theta}}{\partial \eta_l} &= K\left[\frac{\partial}{\partial \eta} \nabla_\theta \log \pi(y \mid \theta, \eta) + \frac{\partial \nabla_\theta \log \pi(y \mid \theta, \eta)}{\partial \hat{\theta}} \frac{\partial \hat{\theta}}{\partial \eta_l}\right] \\
&= K\left[\frac{\partial}{\partial \eta} \nabla_\theta \log \pi(y \mid \theta, \eta) - W \frac{\partial \hat{\theta}}{\partial \eta_l}\right],
\end{aligned}$$

equivalently,

$$\frac{\partial \hat{\theta}}{\partial \eta_l} = (I + KW)^{-1} K \frac{\partial}{\partial \eta} \nabla_\theta \log \pi(y \mid \theta, \eta).$$

Putting it all together

$$\begin{aligned}\frac{\partial \log \pi_{\mathcal{G}}(y \mid \phi)}{\partial \eta_l} =& \frac{\partial \log \pi(y \mid \hat{\theta}, \eta)}{\partial \eta_l} \\ &+ \frac{1}{2}\mathrm{tr}\left((K^{-1} + W)^{-1} \frac{\partial \nabla^2_\theta \log \pi(y \mid \hat{\theta}, \eta)}{\partial \eta_j}\right) \\ &+ \sum_{i=1}^{n} \frac{\partial \log \pi_{\mathcal{G}}(y \mid \phi)}{\partial \hat{\theta}_i}(I + KW)^{-1} K \frac{\partial}{\partial \eta_l} \nabla_\theta \log \pi(y \mid \theta, \eta),\end{aligned}$$

where

$$\begin{aligned}\frac{\partial \log \pi_{\mathcal{G}}(y \mid \phi)}{\partial \hat{\theta}_i} &= -\frac{1}{2}\frac{\partial}{\partial \hat{\theta}_i} \log |K^{-1} + W| \\ &= -\frac{1}{2}\mathrm{tr}\left((K^{-1} + W)^{-1} \frac{\partial W}{\partial \theta_i}\right) \\ &= \frac{1}{2}\left[(K^{-1} + W)^{-1}\right]_{ii} \frac{\partial^3}{\partial \theta_i^3} \log \pi(y \mid \hat{\theta}, \eta),\end{aligned}$$

with the last line holding only in the special case where the Hessian is diagonal (Rasmussen and Williams 2006, equation 5.23). □

Chapter 6: Simulating Ising and Potts models at critical and cold temperatures using auxiliary Gaussian variables

CHARLES C. MARGOSSIAN[1] AND SUMIT MUKHERJEE[1]

Ising and Potts models are an important class of discrete probability distributions which originated from Statistical Physics and since then have found applications in several disciplines. Simulation from these models is a well known challenging problem. In this paper, we propose a class of MCMC algorithms to simulate from both Ising and Potts models, by using auxiliary Gaussian random variables. Our algorithms apply to coupling matrices with both positive and negative entries, thus including Spin Glass models such as the SK and Hopfield model. In contrast to existing methods of a similar flavor, our algorithm can take advantage of the low-rank structure of the coupling matrix, and scales linearly with the number of states in a Potts model. We compare our proposed algorithm to existing state of the art algorithms, such as the Swendsen-Wang and Wolff algorithms for Ising and Potts models on graphs, and the Heat Bath for Spin Glass models. Our comparison takes into account a wide range of coupling matrices and temperature regimes, focusing in particular on behavior at or below the critical temperature. For cold systems, augmenting our algorithm with a tempering scheme yields significant improvements.

6.1 Introduction

The Ising model is a probability distribution on the space of binary vectors of size n, whose components are allowed to take two values, often termed *spins*, which are traditionally taken to $\{0,1\}$ or $\{-1,1\}$. The Ising model was first introduced in Statistical Physics (Ising 1925) to study ferromagnetism. The Potts model (Potts 1952) generalizes the Ising model to $q \geq 2$ states, where

[1]Columbia University, Department of Statistics

the states can be taken to be $[q] := \{1, 2, \ldots, q\}$ without loss of generality. With this choice, the Potts model is given by the following probability mass function on $[q]^n$:

$$\mathbb{P}(\mathbf{X} = \mathbf{x}) = \frac{\exp\left(\frac{\beta}{2}\sum_{i,j=1}^{n} A_n(i,j)1\{x_i = x_j\}\right)}{Z(\beta, A_n)}. \tag{6.1}$$

Here $\beta > 0$ is the inverse temperature parameter; the coupling matrix A_n is a symmetric matrix which controls the dependence between the components of $\mathbf{X}$, and $Z(\beta, A_n)$ is the normalizing constant, also termed the partition function. We assume that the diagonal entries of A_n equal 0, noting that this does not have any impact on the model (6.1). Since their inception, both the Ising and Potts models have found applications in a wide range of disciplines, ranging fromf Image Processing, Protein Folding, Neuroscience, and Social Sciences. One of the main reasons for the popularity of these models is that they are perhaps the simplest models which exhibit non-trivial dependence across their components.

6.1.1 Examples of Ising and Potts models

In the particular case where the coupling matrix is a scaled adjacency matrix of a graph G_n, the Ising/Potts model is a discrete Markov random field. Of particular interest are the Ising/Potts models on the following graphs:

- **Curie-Weiss model.** Here G_n is the complete graph, and $A_n(i,j) = \frac{1}{n}\delta_{i\neq j}$;
- **Ising model on the integer lattice.** Here A_n is the adjacency matrix of a sub-cube of the integer lattice $[1, n^{1/d}]^d$ (assume $n^{1/d}$ is an integer for simplicity);
- **Ising model on random graphs.** Here $A_n = \frac{1}{\bar{d}(G_n)}G_n$, where $\bar{d}(G_n)$ is the average degree of the graph G_n, and G_n is random (e.g. Erdős-Rényi random graphs, Random regular graphs, graphons).

On the other hand, if the coupling matrix A_n is allowed to take both positive and negative real entries, then the corresponding Ising model is typically referred to as Spin Glass in the literature.

The two most commonly studied examples of Spin Glass models are the following:

- **Sherrington Kirkpatrick model.** Here $\{A_n(i,j)\}_{1\le i<j\le n} \overset{i.i.d.}{\sim} N(0,1/n)$);
- **Hopfield model.** Here $A_n = \frac{1}{\max(n,d)}\eta'\eta$, where η is a $d \times n$ matrix of i.i.d. Rademacher random variables.

In all these examples the scaling of A_n is done to ensure that the log partition function scales in a non-trivial manner for large n. To discuss the scaling choices, we introduce the following standard asymptotic notations:

Definition 6.1. *Suppose $\{a_n\}_{n\ge1}$ and $\{b_n\}_{n\ge1}$ are two positive real sequences. We will say*

$$a_n = o(b_n) \text{ or } a_n \ll b_n \text{ if } \lim_{n\to\infty} \frac{a_n}{b_n} = 0,$$
$$a_n = O(b_n) \text{ if } \limsup_{n\to\infty} \frac{a_n}{b_n} < \infty,$$
$$a_n = \Theta(b_n) \text{ if } a_n = O(b_n) \text{ and } b_n = O(a_n).$$

Throughout the paper, all scalings of A_n are chosen such that $\log Z(\beta, A_n) = \Theta(n)$, i.e. the log partition function scales linearly in n as $n \to \infty$. This choice of scaling ensures that the model has a non-trivial phase transition in the parameter β, which is of interest in Statistical Physics and Probability (cf. Basak and Mukherjee 2017, and references therein). One sufficient condition for $\log Z(\beta, A_n) = \Theta(n)$ is $\lambda_{\max} = \Theta(1)$, where $\lambda_{\max}$ denotes the largest eigenvalue of A_n. Except for very special cases and small systems, we cannot compute the normalizing constant $Z(\beta, A_n)$ in (6.1). This means various quantities of interest are intractable, including the partition function and other derived quantities, and need to be estimated, for instance, using Markov chains Monte Carlo (MCMC) methods.

6.1.2 Existing methods

Drawing samples from Ising/Potts models can be a challenging task, with the performance and even applicability of existing methods depending on the type of coupling matrix A_n and the

system's "temperature" β^{-1}. Perhaps the most popular methods in Physics are auxiliary cluster algorithms, notably the Swendsen-Wang and Wolff algorithms (Swendsen and Wang 1987; Wolff 1989), which continue to be used for high-precision Monte Carlo estimation (e.g. Ferrenberg, Xu, and Landau 2018). These methods are particularly well suited for studying systems at critical and cold temperatures, however they only apply to Ising/Potts models defined on graphs. For Spin Glass models, a well-studied method is Heat Bath, also termed the sequential Gibbs, (e.g Neal 1993), often augmented with a tempering scheme when studying cold systems (e.g. Swendsen and Wang 1986; Hukushima, Takajama, and Yoshina 1998; Katzgraber, Palassini, and Young 2001; Yucesoy 2013).

A relatively recent approach is to introduce an auxiliary multivariate Gaussian variable, $\mathbf{Z}$, and construct MCMC over the augmented space, $(\mathbf{X}, \mathbf{Z})$, to sample from binary random Markov fields, a special class of Ising models where A_n is a (scaled) adjacency matrix (Martens and Sutskever 2010; Zhang et al. 2012). Our work generalizes this approach to Potts models. Martens and Sutskever (2010) note that their method extends to the non-binary case, by introducing an $nq-$multivariate normal, but this results in a Gibbs sampler with computational complexity $O(n^3q^3 + mn^2q^2)$, where m is the number of sampling iterations. The algorithm we propose achieves $O(n^3 + mn^2q)$ complexity, and a suitable low-rank approximation can further reduce this cost. We also note that in their study Martens and Sutskever (2010) found the Auxiliary Gaussian Gibbs sampler did not outperform the Heat Bath, which stands in contrast to our results. By examining a broad range of coupling matrices and temperatures, we identify the model regimes where using an auxiliary Gaussian works best and where it fails.

Rather than use a block Gibbs sampler, it is possible to sample over the marginal space of $\mathbf{Z}$ using Hamiltonian Monte Carlo (HMC), a gradient-based algorithm. $\mathbf{X}$ is then recovered using its distribution conditional on $\mathbf{Z}$ (Zhang et al. 2012). We will see that our method is amiable to this scheme, however this paper focuses on a block Gibbs sampler which is more straightforward to analyze and implement. Preliminary experiments we conducted found that HMC, as implemented in Stan (Carpenter et al. 2017), struggles to explore the marginal space of $\mathbf{Z}$, yielding poor mixing

in the $\mathbf{X}$ space. In line with the theory developed by Zhang et al. (2012), we conjecture that HMC works well only on a specific class of Ising/Potts models and leave this line of research to future investigation. There also exist general purpose gradient-based algorithms which attempt a continuous relaxation of discrete systems (e.g. Grathwohl et al. 2021). How these methods compare to specialized methods and our algorithms for Ising/Potts models is the object of upcoming work.

6.2 Choices of Auxiliary Gaussian

6.2.1 Auxiliary Gaussian for Potts

Given a vector $\mathbf{x} = (x_1, \ldots, x_n) \in [q]^n$, for every $\ell \in [q]$ define a vector $\mathbf{y}_\ell := (1\{x_1 = \ell\}, \ldots, 1\{x_n = \ell\}))' \in \mathbb{R}^n$. In other words, the i^{th} element of $\mathbf{y}_\ell$ is 1 if $x_i = \ell$, and 0 otherwise. This sets up a one-to-one map between $\mathbf{x}$ and $(\mathbf{y}_1, \ldots, \mathbf{y}_q)$. Then we have

$$\frac{1}{2}\sum_{i,j=1}^{n} A_n(i,j)1\{x_i = x_j\} = \frac{1}{2}\sum_{\ell=1}^{q} \mathbf{y}_\ell' A_n \mathbf{y}_\ell.$$

The term $\frac{1}{2}\mathbf{y}_\ell' A_n \mathbf{y}_\ell$ looks like the log moment generating function of a multivariate Gaussian, except that the matrix A_n is not non-negative definite, as the diagonal entries of A_n equal 0. Let $\lambda_{\min}$ be the smallest eigenvalue of A_n. Since changing the diagonal entries of A_n does not impact the Potts model, we can replace A_n by $A_n + \lambda I_n$ where $\lambda > |\lambda_{\min}|$, and $A_n + \lambda I_n$ is positive definite. To check that the model (6.1) does not change under this transformation, note that

$$\begin{aligned}&\sum_{\ell=1}^{q} \mathbf{y}_\ell'(A_n + \lambda I_n)\mathbf{y}_\ell - \sum_{\ell=1}^{q} \mathbf{y}_\ell' A_n \mathbf{y}_\ell \\ =&\lambda \sum_{\ell=1}^{q} \mathbf{y}_\ell' \mathbf{y}_\ell = \lambda \sum_{\ell=1}^{q}\sum_{i=1}^{n} 1\{x_i = \ell\} = n\lambda,\end{aligned}$$

where the last equality follows from the fact $\sum_{\ell=1}^{q} 1\{x_i = \ell\} = 1$ for each $i \in [n]$. Setting $B_n := \beta(A_n + \lambda I_n)$, the marginal probability mass function of $\mathbf{X}$ can be written as

$$\mathbb{P}(\mathbf{X} = \mathbf{x}) = \frac{\exp\left(\frac{1}{2}\sum_{\ell=1}^{q} \mathbf{y}_\ell' B_n \mathbf{y}_\ell\right)}{Z(\beta, B_n)}. \tag{6.2}$$

Given $\mathbf{X} = \mathbf{x}$, let $\mathbf{Z}_1, \ldots, \mathbf{Z}_q$ be mutually independent Gaussian random vectors with

$$\mathbf{Z}_\ell \sim N\left(\mathbf{y}_\ell, B_n^{-1}\right). \tag{6.3}$$

Then one can check that given $\mathbf{Z} := (\mathbf{Z}_1, \ldots, \mathbf{Z}_q)$ the random variables $(X_1, \ldots, X_n)$ are mutually independent, with

$$\mathbb{P}(X_i = \ell \mid \mathbf{Z}) = \frac{\exp\left(\sum_{j=1}^{n} B_n(i,j) z_\ell(j)\right)}{\sum_{a=1}^{q} \exp\left(\sum_{j=1}^{n} B_n(i,j) z_a(j)\right)}. \tag{6.4}$$

Also, the marginal density of $\mathbf{Z}$ is proportional to

$$\exp\Big(-\frac{1}{2}\sum_{\ell=1}^{q} \mathbf{z}_\ell' B_n \mathbf{z}_\ell\Big) \prod_{i=1}^{n} \Big(\sum_{\ell=1}^{q} \exp\Big(\sum_{j=1}^{n} B_n(i,j) z_\ell(j)\Big)\Big). \tag{6.5}$$

The proofs of both (6.4) and (6.5) are in the Supplement.

6.2.2 Sampling based on an Auxiliary Gaussian

By iterating between (6.3) and (6.4), we obtain a block Gibbs sampling algorithm, where one alternates between sampling $(\mathbf{Z} \mid \mathbf{X})$ and $(\mathbf{X} \mid \mathbf{Z})$ (Algorithm 6.1). The proposed scheme is analogous to a Gibbs sampler, with stationary distribution $\pi(\mathbf{q}) = \pi(\mathbf{x}, \mathbf{z})$, where $\mathbf{q} = (\mathbf{x}, \mathbf{z})$. The multidimensional "coordinates" which get updated are $\mathbf{x}$ and $\mathbf{z}$. The theoretical results for the Gibbs sampler thus carry over and can be explicitly derived by, for example, following the proof by Neal (1993). Furthermore having a chain $\mathbf{q}^{(1)}, \mathbf{q}^{(2)}, \cdots, \mathbf{q}^{(k)}$ with invariant distribution $\pi(\mathbf{q})$ immediately gives us a chain with invariant marginal distribution $\pi(\mathbf{x})$.

A key observation is that the covariance for the $\mathbf{Z}_\ell$'s remains unchanged between sampling

Algorithm 6.1: Block Gibbs sampling for Potts model to compute m approximate samples

input: initial state, $\mathbf{x}^{(0)}$; coupling matrix, A_n; inverse temperature, β; number of iterations, m; perturbation, ϵ.

$\lambda \leftarrow |\lambda_{\min}(A_n)| + \epsilon$ ▷ $\lambda_{\min}(A_n)$ = smallest eigenvalue of A_n.

$B_n \leftarrow \beta(A_n + \lambda \mathbf{I}_n)$

$L_n \leftarrow$ Cholesky-decompose$([B_n]^{-1})$

for $i \in \{1, \cdots, m\}$ **do**

  for $\ell \in \{1, \cdots, q\}$ **do**

    $\mathbf{y}_\ell \leftarrow \left(1\left\{x_1^{(i-1)} = \ell\right\}, ..., 1\left\{x_n^{(i-1)} = \ell\right\}\right)$

    $\mathbf{z}_n^* \sim \text{Normal}(0, I_n)$

    $\mathbf{z}_\ell \leftarrow L_n \mathbf{z}_n^* + \mathbf{y}_\ell$

  end

  $P_n = B_n[\mathbf{z}_1, ..., \mathbf{z}_q]$

  for $j \in \{1, \cdots, n\}$ **do**

    $\mathbf{p}_q = (\exp(P_n(j, \ell)), \ell \in [q])$ ▷ vector of unnormalized probability for each state.

    $\mathbf{x}_j^{(i)} \sim \text{Categorical}(\mathbf{p}_q / \sum_{a=1}^{q} p_q(a))$

  end

end

return: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(m)}$.

iterations. Hence we only need to perform once the Cholesky decomposition and inversion of $B_n = \beta A_n + \lambda I_n$ required to sample $\mathbf{Z}_\ell$ (6.3). This is an expensive operation with complexity $O(n^3)$. Within each iteration, the cost is dominated by $2q$ matrix-vector multiplications, each with cost $O(n^2)$. The complexity of the algorithm is therefore $O(n^3 + mn^2q)$. Provided $n \ll m$, the cost of the initial inversion and Cholesky decomposition is marginal.

6.2.3 Low-rank Auxiliary Gaussian

For certain models, the coupling matrix B_n may be low-rank, either exactly or approximately. In this case, it is reasonable to believe that we don't have to work with n-dimensional Gaussian vectors, but instead with a k-dimensional Gaussian vector, where $k = \text{rank}(B_n)$. In fact, such algorithms have already been explored in the setting of Ising models for the complete graph (Mukherjee, Mukherjee, and Yuan 2018), where $A_n = \frac{1}{n}\mathbf{1}\mathbf{1}' - \frac{1}{n}I_n$ has eigenvalues $(1 - \frac{1}{n}, -\frac{1}{n}, \ldots, -\frac{1}{n})$, where the multiplicity of the eigenvalue $-\frac{1}{n}$ is $n - 1$. Even though $\text{rank}(A_n) = n$ in a strict sense, we can add $-\frac{1}{n}I_n$ to A_n to get the matrix $\frac{1}{n}\mathbf{1}\mathbf{1}'$, which has eigenvalues $(1, 0, \ldots, 0)$, and rank 1. A similar approach was successful in analyzing the Ising model of a regular graph with degree $\propto \sqrt{n}$ (Mukherjee and Xu 2021), where $\text{rank}(A_n) \propto \sqrt{n}$.

To propose a general version of the low-rank algorithm, let $B_n = \beta(A_n + |\lambda_{\min}|I_n)$. Note it is sufficient to augment the diagonal elements with $|\lambda_{\min}|$, rather than $\lambda > |\lambda_{\min}|$. Let $\sum_{i=1}^n \mu_i \mathbf{p}_i \mathbf{p}_i'$ denote the spectral expansion of B_n, where $(\mu_1, \ldots, \mu_n)$ are non-negative, and arranged in decreasing order, and $\mathbf{p}_i \in \mathbb{R}^n$ is the eigenvector corresponding to eigenvalue μ_i. Fixing $\varepsilon > 0$, we introduce the low-rank approximation

$$\tilde{B}_n = \sum_{i:\mu_i > \varepsilon} \mu_i \mathbf{p}_i \mathbf{p}_i'.$$

In other words, we treat the eigenvalues below ε as 0. Let $k \leq n$ denote the rank of $\tilde{B}_n$, i.e. $k := \sum_{i=1}^n 1\{\mu_i > \varepsilon\}$. Let $\mathbb{Q}$ be the Potts model with B_n replaced by $\tilde{B}_n$, i.e.

$$\mathbb{Q}(\mathbf{X} = \mathbf{x}) = \frac{\exp(\frac{1}{2}\sum_{i,j=1}^n \tilde{B}_n(i,j)1\{x_i = x_j\})}{Z(\beta, \tilde{B}_n)}. \tag{6.6}$$

The Hamiltonian of the probability measure $\mathbb{Q}$ can be written as

$$\frac{1}{2}\sum_{\ell=1}^{q}\mathbf{y}_\ell'\tilde{B}_n\mathbf{y}_\ell = \frac{1}{2}\sum_{\ell=1}^{q}\sum_{j=1}^{k}\mu_j(\mathbf{p}_j'\mathbf{y}_\ell)^2. \tag{6.7}$$

This representation motivates the following new auxiliary variable. Given $\mathbf{X} = \mathbf{x}$, let $\mathbf{Z} := \{Z_\ell(j)\}_{\ell\in[q],j\in[k]}$ be kq mutually independent Gaussians with

$$Z_\ell(j) \sim \text{Normal}(\mathbf{p}_j'\mathbf{y}_\ell, 1/\mu_j). \tag{6.8}$$

Then the conditional distribution of $\mathbf{X}$ given $\mathbf{Z}$ is given by

$$\mathbb{P}(X_i = \ell \mid \mathbf{Z}) = \frac{\exp\left(\sum_{j=1}^{k}\mu_j z_\ell(j)p_j(i)\right)}{\sum_{a=1}^{q}\exp\left(\sum_{j=1}^{k}\mu_j z_\ell(j)p_j(i)\right)}, \tag{6.9}$$

which is straightforward to sample from. Also, the marginal density of $\mathbf{Z}$ has a similar low-rank structure, and is proportional to

$$\exp\Big(-\frac{1}{2}\sum_{\ell=1}^{q}\sum_{j=1}^{k}\mu_j z_\ell(j)^2\Big)\prod_{i=1}^{n}\Big(\sum_{\ell=1}^{q}\exp(\sum_{j=1}^{k}\mu_j z_\ell(j)p_j(i))\Big) \tag{6.10}$$

The proof of both (6.9) and (6.10) are given in the Supplementary Material.

By iterating between the steps (6.8) and (6.9), we again have a block Gibbs sampler. We no longer need to do an inversion and Cholesky decomposition, but we do one Eigen decomposition instead, which incurs the same complexity $O(n^3)$. The cost per iteration is dominated by (i) calculating the mean of each univariate normal, that is doing qk inner-products of n-vectors, and (ii) calculating the probability of sampling each state for each particle, which is qn inner-products of k-vectors. The resulting Gibbs sampler has complexity $O(n^3 + mqnk)$. This is an improvement over the previously obtained complexity $O(n^3 + mqn^2)$. In particular the improvement is significant when $m \gg n \gg k$.

Remark 6.1. *Another MCMC algorithm arises out of our analysis, using* (6.5) *(or* (6.10)*), which gives the unnormalized marginal density of the matrix* $\mathbf{Z}$*. Since* $\mathbf{Z}$ *is continuous, we can use gradient-based algorithms to simulate* $\mathbf{Z}$ *,and subsequently sample* $\mathbf{X}$ *from* (6.4) *(or* (6.9)*). A low-rank approximation of* A_n *will give a low-rank version of the density of* $\mathbf{Z}$*, which also improves the computation of gradient-based algorithms.*

Remark 6.2. *The proposed auxiliary Gaussian variables decouple the elements of* $\mathbf{X}$ *in the more general case where the Potts model admits a non-zero bias or exterior magnetic field* $\mathbf{b} \in \mathbb{R}^q$*. In this scenario, the probability mass function is*

$$\mathbb{P}(\mathbf{X} = \mathbf{x}) \propto \exp\left(\frac{\beta}{2}\sum_{i,j}^{n} A_n(i,j)1\{x_i = x_j\} + \sum_{i=1}^{n}\sum_{\ell=1}^{q} b(\ell)1\{x_i = \ell\}\right). \tag{6.11}$$

The conditional distribution is updated as follows:

- *Auxiliary Gaussian for Potts* (6.3)

$$\mathbb{P}(X_i = \ell \mid \mathbf{Z}) = \frac{\exp\left(\sum_{j=1}^{n} B_n(i,j)z_\ell(j) + b(\ell)\right)}{\sum_{a=1}^{q}\exp\left(\sum_{j=1}^{n} B_n(i,j)z_a(j) + b(a)\right)}. \tag{6.12}$$

- *Low-rank Auxiliary Gaussian* (6.8)

$$\mathbb{P}(X_i = \ell \mid \mathbf{Z}) = \frac{\exp\left(\sum_{j=1}^{k} \mu_j z_\ell(j)p_j(i) + b(\ell)\right)}{\sum_{a=1}^{q}\exp\left(\sum_{j=1}^{k} \mu_j z_\ell(j)p_j(i) + b(a)\right)}, \tag{6.13}$$

6.2.4 Low-rank approximation error

It is natural to guess that the low-rank Auxiliary Gaussian method outperforms the usual Auxiliary Gaussian Method when the matrix A_n is approximately of low-rank. A more refined question is whether we can provide rigorous bounds on the error in this approximate algorithm, which allows the user to choose the tuning parameter ε. To this effect, we state the following lemma, whose proof is included in the Supplementary Material.

Lemma 6.1. *Let $\mathbb{P}$ be the original Potts measure given in* (6.2)*, and let $\mathbb{Q}$ be the low-rank Potts measure given in* (6.6)*. Then we have the following results:*

(i) The log normalizing constant between the two models satisfies

$$|\log Z(\beta, B_n) - \log Z(\beta, \tilde{B}_n)| \leq \frac{1}{2} n\beta\varepsilon.$$

(ii) With $KL(.|.)$ denoting the Kullback-Leibler divergence between two probability measures, we have

$$\max\left(KL(\mathbb{P}|\mathbb{Q}), KL(\mathbb{Q}|\mathbb{P})\right) \leq n\beta\varepsilon.$$

The low-rank Auxiliary Gaussian gives a parsimonious representation only if $k = \text{rank}(\tilde{B}_n) \ll n$. The following lemma provides a simple sufficient condition as to when this happens.

Lemma 6.2. *Suppose $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues of A_n. Let $B_n := \beta(A_n + |\lambda_{\min}|I_n)$, and $\tilde{B}_n$ be its low-rank approximation for some $\varepsilon > 0$. If*

$$\lambda_{\max} = O(1), \quad \lambda_{\min} = o(1), \tag{6.14}$$

then $k_n := \text{rank}(\tilde{B}_n) = o(n)$.

The proof of this lemma is included in the Supplementary Material.

The above lemma raises the question of whether there are natural examples of matrices A_n which satisfy (6.14). Below we give some examples of matrices A_n which satisfy (6.14), and some examples which don't.

- **Satisfies** (6.14): Ising model on the Complete Graph (Curie-Weiss model), Random regular graphs and Erdős-Rényi graphs with large average degree, Hopfield model with $m \ll n$.
- **Does not satisfy** (6.14): Ising model on graphs with bounded average (including Ising model on the integer lattice), Hopfield model with $m = n$, SK model.

If (6.14) is not satisfied, then it seems reasonable to prefer the regular version of the Auxiliary

Gaussian algorithm over the low-rank version. Numerical experiments suggest the regular version is slightly faster when A_n is full-rank.

6.3 Numerical experiments

We now study the performance of various MCMC samplers on a range of Potts models.

When A_n is the scaled adjacency matrix of a graph G_n, the state of the art algorithms are based on constructing auxiliary clusters, which form a random partition of the vertices of the nodes of G_n. Once an auxiliary cluster has been constructed, all the nodes in one cluster are now assigned a common new spin value according to some probability. Thus potentially many states can be updated at once, which allows for a fast exploration of the sample space. The most well known examples in this class are the Swendsen-Wang and Wolff algorithms (Swendsen and Wang 1987; Wolff 1989). We find in our experiments that the Wolff algorithm is more efficient than Swendsen-Wang, a result consistent with the Physics literature (e.g Wolff 1989; Landau and Binder 2009). For this reason, we only report comparison of our Auxiliary Gaussian algorithms (both the original and the low-rank version) with the Wolff algorithm. In the Supplementary Material, we offer a brief review of auxiliary cluster algorithms. Another algorithm we use to compare our results is the classic Heat Bath algorithm.

For the spin glass models we no longer have access to auxiliary cluster algorithms. In this case we only use the Heat Bath algorithm as the benchmark, and compare it with our proposed Auxiliary Gaussian (AG) samplers. In the special case of the Ising model where $q = 2$, we use a specialized AG algorithm which reduces the dimension of the auxiliary Gaussian variables from $2n$ to n ($2k$ to k when doing a low-rank approximation) which improves the performance of the AG sampler (see Supplementary Material for details on this specialized version of the algorithm for $q = 2$). The non low-rank specialization reduces to the auxiliary Gaussian variable based algorithm introduced by Martens and Sutskever (2010). All samplers are implemented in R using the same packages to attempt a fair comparison. As we explain below, the utility of each method depends heavily on the coupling matrix A_n, and the inverse temperature β.

6.3.1 Performance metrics

Our goal is to construct Monte Carlo estimators for $\mathbb{E}_{\mathbb{P}}\phi(\mathbf{X})$, where $\phi : [q]^n \to \mathbb{R}$, via the sample average

$$\delta_m = \frac{1}{m}\sum_{i=1}^{m} \phi\left(\mathbf{x}^{(i)}\right).$$

Here $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ represent the output of the first m iterations of the MCMC algorithm. If $\mathrm{Var}_{\mathbb{P}}\phi(X) < \infty$ and the samples are i.i.d., δ_m is unbiased, and has variance $\mathrm{Var}(\delta_m) = \mathrm{Var}_{\mathbb{P}}\phi(X)/m$. Unfortunately the samples we generate with MCMC are approximate: they are not independent and, for a finite number of iterations of our MCMC algorithm, they are biased. Practitioners typically expect that, after a certain number of iterations, the Markov chain converges to a stationary distribution and is unbiased. The first samples are discarded, thereby removing the initial bias of the so-called "burn in" phase. However, depending on the inverse-temperature parameter β (and in particular for large β), MCMC algorithms for Ising and Potts models can be notoriously slow to mix: even after many iterations, they may fail to overcome the initial transient bias. One way to diagnose MCMC convergence is to run multiple chains with different initializations, and check that they produce Monte Carlo estimators in good agreement. We perform such a check with the rank-normalized $\hat{R}$ statistics developed by Vehtari et al. (2020), which estimates ratio of the variance across all chains over the variance within each chain. If the chains are mixing, $\hat{R}$ should be close to 1, and eventually it should converge to 1 as the number of iterations converges to $+\infty$. Another sanity check is to confirm that Monte Carlo estimators produced by different algorithms are in agreement with one another.

Once we trust our estimator to be unbiased, we examine the *effective sample size*, defined by

$$m_{\mathrm{eff}} = \frac{\mathrm{Var}\phi(X)}{\mathrm{Var}(\delta_m)}.$$

If the samples are independent, then $m_{\mathrm{eff}} = m$. MCMC typically generates correlated samples, which incurs a loss of information, in which case $m_{\mathrm{eff}} < m$. A popular method for computing m_{eff}

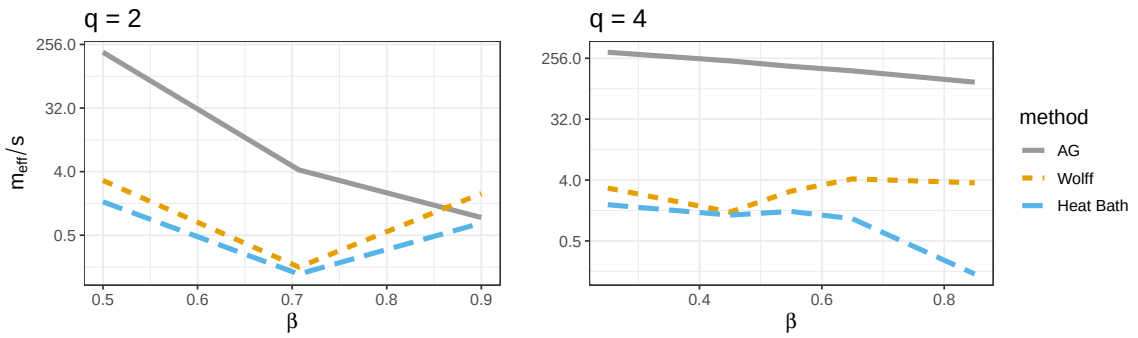

Figure 6.1: Effective sample size per second for grid model.

is via an autocorrelation function (c.f. Geyer 2012). We compute $\hat{R}$ and m_{eff} using the R package Posterior (Gabry, Buekner, and Kay 2021).

Our performance metric is the effective sample size per second, m_{eff}/s. The effective sample size alone considers how well the MCMC algorithm mixes over iterations but it does not consider the computation cost of each iteration. A useful sampler finds a good balance between computing iterations quickly, and generating samples with a low autocorrelation. We focus on the summary function $\phi(\mathbf{x}) = \beta \sum_{i,j=1}^{n} A_n(i,j) 1\{x_i = x_j\}$, which is the system's Hamiltonian.

6.3.2 Potts models on a graph

We begin with the case where A_n is the adjacency matrix of a graph G_n scaled by its average degree. We consider two choices of G_n which are of wide interest in Statistical Physics: the Ising model on the integer lattice and the Curie-Weiss model. For the subsequent experiments in this section, we run 4 chains each for $m = 10,000$ iterations, and discard the first 1,000 iterations of each chain as burn-in.

Lattice graph. The two-dimensional lattice graph $[1, \sqrt{n}]^2$ with two states (i.e. $q = 2$) is possibly the most well-studied example. We examine a 16×16 graph (i.e. $n = 256$), and consider the cases $q = 2, 4$. We focus on a parameter range in β around the critical temperature $\beta_c(q)$, which is approximately 0.44 and 0.55 for $q = 2$ and $q = 4$ respectively, as reported in Monroe 2002. Overall the AG sampler offers the best performance, as measured by m_{eff}/s (Figure 6.1). We note, however, a very sharp drop in performance of the AG algorithm as β increases beyond criticality,

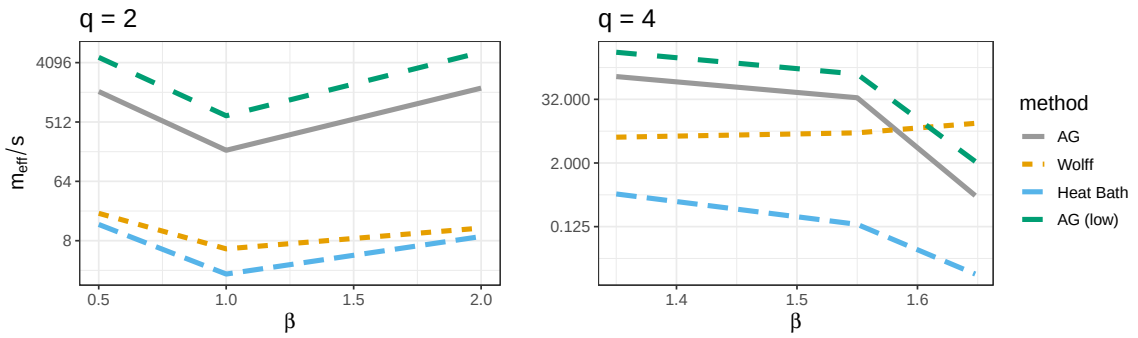

Figure 6.2: Effective sample size per second for Curie-Weiss model.

for $q = 2$. In comparison, for $q = 4$ our algorithm performs much better at or beyond criticality. This may be because the larger sample space allows for more exploration, and as such decreases autocorrelation between updates. In the case $q = 2$, the Wolff sampler, by contrast, is more robust to large β; we believe this is because at cold temperatures the coupling between nodes is stronger, allowing the Wolff algorithm to construct large auxiliary clusters and update more nodes at once. These global updates lead to a decrease in autocorrelation. The expected slow down at critical temperature for $q = 2$ is apparent for all the three samplers.

Curie-Weiss model. In this model $A_n = \frac{1}{n}\mathbf{1}\mathbf{1}' - \frac{1}{n}I_n$ is the scaled adjacency matrix of a complete graph, where each node is connected to every other node. We once again consider the cases $q = 2, 4$, with $n = 256$. The critical value $\beta_c(q)$ is 1 and ~1.64 for $q = 2$ and $q = 4$ respectively (e.g Bollobás, Grimmett, and Janson 1996), around which we focus our simulations. Adjusting the diagonal elements, we obtain a rank-1 coupling matrix on which we can apply the low-rank AG sampler. The same update rule is obtained by truncating the spectral expansion of B_n, using the threshold $\varepsilon = 10^{-12}$ which is slightly above machine precision, as our tolerance for eigenvalues. Figure 6.2 shows the performance measurements. In the Ising case (i.e. $q = 2$), the AG samplers offer the best performance by several orders of magnitude. The low-rank sampler offers a ~3 - 4 fold speed up over the regular AG sampler. The story is quite different at $q = 4$. As we near the value $\beta = 1.65$, we observe a sharp drop in efficiency for the Heat Bath and the AG samplers. For $\beta > 1.65$, the chains fail to mix, as notably indicated by the $\hat{R}$ statistic, and produce wildly inaccurate estimates of $\mathbb{E}\phi(\mathbf{X})$. A previous study found that the Heat Bath undergoes a dramatic

slow down near $\beta \approx 1.65$ (Cuff et al. 2012). Clearly this behavior also affects the AG samplers. The Wolff and Swendsen-Wang algorithms are by contrast not affected by this phenomenon, and produce accurate Monte Carlo estimates for $\beta > 1.65$.

6.3.3 Spin Glass Potts models

In a Spin Glass model, the entries in A_n can take both positive and negative values. Two important examples are the Hopfield model and the Sherrington-Kirkpatrick (SK) models. For these problems, the Wolff and Swendsen-Wang algorithms are no longer an option.

Hopfield model. As before, we run 4 chains each with $m = 10,000$ sampling iterations, discarding the first 1,000 samples as a burn-in phase. Let $\eta \in \mathbb{R}^{d\times n}$ be a matrix of i.i.d random variables with $P(\eta_{ik} = \pm 1) = 1/2$, and let

$$A_n = \max(n, d)^{-1}\eta'\eta. \tag{6.15}$$

By convention we should zero out the diagonals of A_n. But since changing diagonal entries does not impact the distribution, and (6.15) is the commonly specified form of the Hopfield matrix, we will use (6.15). Now A_n can take values which are positive, negative, or zero, meaning the spins X_i can be correlated, anti-correlated, or uncorrelated. It is usually not possible to set each node to be equal, so as to maximize $A_n(i, j)1\{x_i = x_j\}$ for all (i, j), as a result of which the emergent system is said to be frustrated.

In this case, if $d \ll n$, then standard concentration results show that $A_n = n^{-1}\eta'\eta \approx I_n$, which has all positive eigenvalues. Thus for $d \ll n$, all eigenvalues of $A_n - \text{diag}(A_n)$ should be close to 0 with high probability. Consequently, (6.14) will hold, and so the low-rank algorithm should be used. As d increases, the low-rank algorithm should become worse, until it becomes comparable to (or slightly worse than) the regular AG algorithm. We use this example to examine the validity of the above prediction, and to compare the regular and the low-rank AG samplers across varying ranks. We set $\beta = 1$, $q = 4$, $n = 256$, and vary d from 1 to 100. Setting $\varepsilon = 10^{-12}$, as we did

with the Curie-Weiss model, we find the low-rank AG incorrectly treats A_n as full-rank. This is due to the numerical errors that arise when computing eigenvalues. Setting $\varepsilon = 10^{-10}$ fixes the issue for this particular problem. Overall the AG samplers offer performance which is two orders of magnitude faster than the Heat Bath. The low-rank AG offers the best performance for small d, but this advantage vanishes as the rank of A_n increases (Figure 6.3).

Remark 6.3. *In all our experiments, the regular and low-rank AG generate roughly the same effective sample size. The difference in performance is therefore driven by the difference in the computational cost of each iteration.*

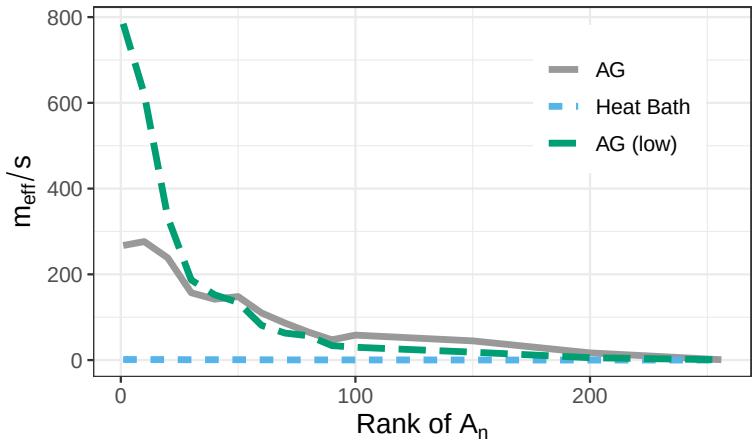

Figure 6.3: Effective sample size per second for Hopfield Potts model with $q = 4$ and $\beta = 1$.

Cold SK model. In the SK model, the elements of a coupling matrix $\{A_n(i,j)\}_{1\leq i<j\leq n}$ are mutually independent, drawn from a standard normal scaled by $1/\sqrt{n}$, equivalently a normal with variance $1/n$:

$$A_n(i,j) = A_n(j,i) \sim \text{Normal}\left(0, n^{-1}\right).$$

This coupling matrix unfortunately does not admit a good low-rank approximation (c.f. Basak and Mukherjee 2017). An important application of this model in Physics is the study of spin glass systems at cold temperatures, i.e. below the critical temperature (e.g Katzgraber, Palassini, and Young 2001; Yucesoy 2013). We have seen in our previous experiments that the performance of

the Heat Bath and the AG sampler suffers as β increases. This is also true for the SK model. For high β, the Potts model becomes highly multimodal, resulting in a slow or incomplete exploration by the Markov chains.

Tempering algorithms can mitigate this problem and be coupled with any sampling scheme. The tempered Heat Bath is a popular method to draw samples from cold Spin Glass models (e.g Swendsen and Wang 1986; Hukushima, Takajama, and Yoshina 1998). As we shall see, this strategy is further enhanced when we replace the Heat Bath with an AG sampler (Figure 6.4). A tempering algorithm runs multiple chains or *replicas* over a sequence of temperatures for a set number of iterations and then exchanges, with a certain probability, the states of two replicas with a neighboring temperature. The probability of exchanging two neighboring replicas is chosen so as to maintain detailed balance. At high temperatures, the Markov chain can move more easily across the target space and overcome the energy barriers between modes, before being cooled down again to sample at the temperature of interest. Naturally we can run replicas in parallel.

When implementing a tempering algorithm, we need to choose the number of replicas and their respective temperatures. The first replica is at the cold temperature of interest. We then progressively increase the temperature of each replica to insure that the exchange probability is high, and that we eventually reach a warm temperature at which the sampler is not frustrated by high energy barriers. Additional details about the tempering algorithm can be found in the Supplementary Material.

In our experiment, the target inverse-temperature is $\beta = 3$, which is way above the critical parameter. We create 11 replicas with $\beta = (0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3)$. We find this yields reasonable, if somewhat uneven, acceptance rates for the exchange proposals. A more principled approach to setting temperatures is described by Hukushima, Takajama, and Yoshina 1998. In total, we attempt 40 exchanges and run 1,000 iterations between exchanges, for a total of 40,000 iterations. As before, we run 4 chains – or here 4 sets of replicas – with different initializations to compute $\hat{R}$ and monitor whether the chains are mixing. The size of the system is $n = 128$.

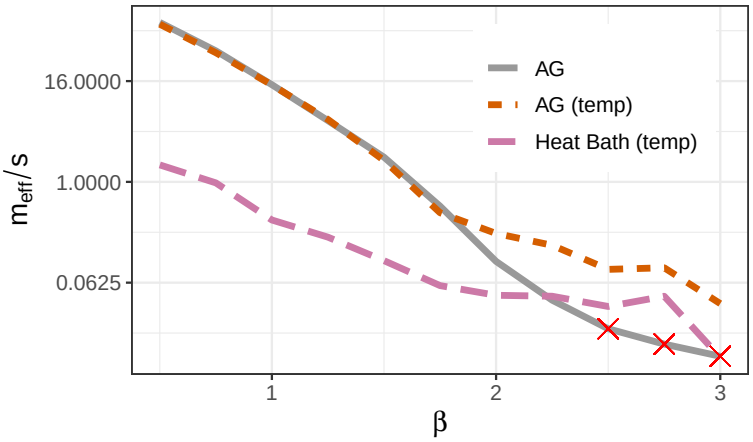

Figure 6.4: Effective sample size per second for SK Potts model. *For $\beta \geq 2.5$, the AG sampler, without tempering, returns estimates in disagreement with the other samplers and has poor $\hat{R}$ diagnostics. The measured effective sample size at these temperatures for the AG sampler should not be trusted, as indicated by an "×".*

Figure 6.4 presents the results of our experiment. We measure the efficiency of a tempered AG sampler, a tempered Heat Bath, and a regular AG sampler. The non-tempered AG suffers at colder temperatures and for $\beta > 2.5$, AG produces Monte Carlo estimates in disagreement with competing algorithms; furthermore the large $\hat{R}$ values ($\hat{R} > 1.2$) suggest the Markov chains are not mixing. The tempered AG offers the best performance across all temperatures. We note that at $\beta = 3$, for both tempered algorithms, $\hat{R} \approx 1.13$, which suggests we at least ought to run more iterations to produce more accurate estimates. Finally while tempering offers significant improvement, the sharp drop in performance with increasing β is still present.

6.4 Discussion

AG samplers offer a competitive and at times largely superior performance across a range of models, particularly in high temperature regimes and at criticality. There are some exceptions to this in the low temperature regime, such as the Ising model on the 2D lattice ($q = 2$), and the Curie-Weiss Potts models with $q = 4$. Similar to the Heat Bath algorithm, the AG algorithms are sensitive to the system's temperature, and tempering can alleviate this issue. Examining the alternative

algorithms (cluster algorithms and the Heat Bath), we find there is no universal algorithm that efficiently handles all the pathologies we may encounter when dealing with Ising and Potts models for various coupling matrices across different choices of q. Care must be taken when picking a sampler.

Various operations in Algorithm 6.1 can be parallelized to further improve performance. This observation also holds for the low-rank AG sampler, and gradient-based samplers that target the marginal distribution of $\mathbf{Z}$. As such, Auxiliary Gaussian methods are amiable to use on accelerators such as GPUs. We remind readers that high-performance computing should be used responsibly, given its energy consumption.

6.5 Aknowledgment

The second author gratefully thanks NSF (DMS-1712037, DMS-2113414) for support during this research.

6.6 Appendix (Supplementary Material)

The Appendix contains proofs missing in the main body. We derive the conditional and marginal distributions, that arise when we introduce either a regular or low-rank auxiliary Gaussian variable for Potts models. We prove Lemma 6.1, which characterizes the error committed in using the low-rank algorithm, and Lemma 6.2, which characterizes the type of coupling matrices for which we may want to use a low-rank algorithm. Specialized algorithms for the Ising model ($q = 2$) are derived. We also provide an overview on auxiliary cluster and tempering algorithms.

The code used for our numerical experiments can be found at https://github.com/charlesm93/potts_simulation. The ReadMe provides instructions on how to run the code.

6.6.1 Missing proofs

Conditional and marginal distributions when using the regular Auxiliary Gaussian

We begin by proving Equations (6.4) and (6.5). Recall from (6.2) that the marginal p.m.f. of $\mathbf{X}$ under $\mathbb{P}$ is proportional to

$$\exp\Big(\frac{1}{2}\sum_{\ell=1}^{q}\mathbf{y}_\ell' B_n \mathbf{y}_\ell\Big).$$

Also from (6.3), the conditional density of $\mathbf{Z}$ given $\mathbf{X} = \mathbf{x}$ is proportional to

$$\exp\Big(-\frac{1}{2}\sum_{\ell=1}^{q}(\mathbf{z}_\ell - \mathbf{y}_\ell)' B_n (\mathbf{z}_\ell - \mathbf{y}_\ell)\Big).$$

Thus, the joint distribution of $\mathbf{X}$ and $\mathbf{Z}$ is proportional to

$$\begin{aligned}
&\exp\left(\frac{1}{2}\sum_{\ell=1}^{q}\Big[\mathbf{y}_\ell' B_n \mathbf{y}_\ell - (\mathbf{z}_\ell - \mathbf{y}_\ell)' B_n (\mathbf{z}_\ell - \mathbf{y}_\ell)\Big]\right)\\
&= \exp\left(-\frac{1}{2}\sum_{\ell=1}^{q}\mathbf{z}_\ell' B_n \mathbf{z}_\ell + \sum_{\ell=1}^{q}\mathbf{z}_\ell' B_n \mathbf{y}_\ell\right)\\
&= \exp\left(-\frac{1}{2}\sum_{\ell=1}^{q}\mathbf{z}_\ell' B_n \mathbf{z}_\ell + \sum_{\ell=1}^{q}\sum_{i=1}^{n}\sum_{j=1}^{n} z_\ell(j) B_n(i,j) 1\{x_i = \ell\}\right).
\end{aligned} \tag{6.16}$$

From (6.16), we see that given $\mathbf{Z}$, the random variables $(X_1, \ldots, X_n)$ are mutually, independent, with

$$\mathbb{P}(X_i = \ell | \mathbf{Z}) \propto \exp\Big(\sum_{j=1}^{n} B_n(i,j) z_\ell(j)\Big),$$

which verifies (6.4). In the case where we have a non-zero bias or magnetic field, one simply adds

$$\sum_{i=1}^{n}\sum_{\ell=1}^{q} b(\ell) 1\{x_i = \ell\}$$

to the exponent when defining the marginal p.m.f of $\mathbf{X}$ and carries this term across all subsequent calculations to obtain (6.12).

The marginal distribution of $\mathbf{Z}$ is obtained from (6.16) by summing over $\mathbf{x} \in [q]^n$, from which

(6.5) follows.

Conditional and marginal distributions when using the low-rank Auxiliary Gaussian

Next we prove Equations (6.8) and (6.10). Recall from (6.7) that the marginal p.m.f. of $\mathbf{X}$ under $\mathbb{P}$ is proportional to

$$\exp\Big(\frac{1}{2}\sum_{\ell=1}^{q}\sum_{j=1}^{k}\mu_j(\mathbf{p}_j' y_\ell)^2\Big).$$

Also from (6.9), the conditional density of $\mathbf{Z}$ given $\mathbf{X} = \mathbf{x}$ is proportional to

$$\exp\Big(-\frac{1}{2}\sum_{\ell=1}^{q}\sum_{j=1}^{k}\mu_j(z_\ell(j) - \mathbf{p}_j'\mathbf{y}_\ell)^2\Big).$$

Thus, the joint distribution of $\mathbf{X}$ and $\mathbf{Z}$ is proportional to

$$\begin{aligned}
&\exp\left(\frac{1}{2}\sum_{\ell=1}^{q}\sum_{j=1}^{k}\mu_j\Big[(\mathbf{p}_j' y_\ell)^2 - (z_\ell(j) - \mathbf{p}_j'\mathbf{y}_\ell)^2\Big]\right)\\
&= \exp\Big(-\frac{1}{2}\sum_{\ell=1}^{q}\sum_{j=1}^{k}\mu_j z_\ell(j)^2 + \sum_{\ell=1}^{q}\sum_{j=1}^{k}\mu_j z_\ell(j)\mathbf{p}_j'\mathbf{y}_\ell\Big)\\
&= \exp\Big(-\frac{1}{2}\sum_{\ell=1}^{q}\sum_{j=1}^{k}\mu_j z_\ell(j)^2 + \sum_{\ell=1}^{q}\sum_{j=1}^{k}\sum_{i=1}^{n}\mu_j z_\ell(j)p_j(i)1\{x_i = \ell\}\Big). \qquad (6.17)
\end{aligned}$$

Using (6.17), given $\mathbf{Z} = \mathbf{z}$ the random variables $(X_1, \ldots, X_n)$ are mutually, independent, with

$$\mathbb{P}(X_i = \ell|\mathbf{Z}) \propto \exp\Big(\sum_{j=1}^{k}\mu_j z_\ell(j)p_j(i)\Big),$$

which verifies (6.9). As in the previous section, the conditional distribution when the bias is non-zero (6.13) is obtained by adding the bias term to the exponent when defining the marginal p.m.f of $\mathbf{X}$ and carrying this term over the subsequent calculations.

The marginal distribution of $\mathbf{Z}$ is obtained from (6.17) by summing over $\mathbf{x} \in [q]^n$, from which (6.10) follows.

Proof of Lemma 6.1

With B_n and $\tilde{B}_n$ as in the Hamiltonians of the distributions $\mathbb{P}$ and $\mathbb{Q}$ in (6.2) and (6.6) respectively, for any $\mathbf{x} \in [q]^n$, we have

$$\begin{aligned}\Big|\sum_{i,j=1}^{n} B_n(i,j)1\{x_i = x_j\} - \sum_{i,j=1}^{n} \tilde{B}_n(i,j)1\{x_i = x_j\}\Big| &= \Big|\sum_{\ell=1}^{q} \mathbf{y}_\ell'(B_n - \tilde{B}_n)\mathbf{y}_\ell\Big| \\ &\leq \sum_{\ell=1}^{q} \|\mathbf{y}_\ell\|_2^2 \|B_n - \tilde{B}_n\|_2 \\ &= \|B_n - \tilde{B}_n\|_2 \sum_{\ell=1}^{q}\sum_{i=1}^{n} 1\{x_i = \ell\} \qquad (6.18) \\ &= n\|B_n - \tilde{B}_n\|_2 \leq n\varepsilon. \qquad (6.19)\end{aligned}$$

where the last bound uses the definition of $\tilde{B}_n$ to note that all eigenvalues of $B_n - \tilde{B}_n$ are smaller than ε in absolute value. Thus, for any $\mathbf{x} \in [q]^n$, we have

$$e^{\frac{-n\beta\varepsilon}{2}} \leq \frac{\exp\left(\frac{\beta}{2}\sum_{i,j=1}^{n} B_n(i,j)1\{x_i = x_j\}\right)}{\exp\left(\frac{\beta}{2}\sum_{i,j=1}^{n} \tilde{B}_n(i,j)1\{x_i = x_j\}\right)} \leq e^{\frac{n\beta\varepsilon}{2}}. \qquad (6.20)$$

On summing over $\mathbf{x} \in [q]^n$, this gives

$$e^{-\frac{n\beta\varepsilon}{2}} \leq \frac{Z(\beta, A_n)}{Z(\beta, \tilde{A}_n)} \leq e^{\frac{n\beta\varepsilon}{2}}, \qquad (6.21)$$

which verifies part (i). For verifying part (ii), taking a ratio of (6.20) and (6.21) we get

$$e^{-n\beta\varepsilon} \leq \frac{\mathbb{P}(\mathbf{X} = \mathbf{x})}{\mathbb{Q}(\mathbf{X} = \mathbf{x})} \leq e^{n\beta\varepsilon},$$

which gives $\max(\mathrm{KL}(\mathbb{P}|\mathbb{Q}), \mathrm{KL}(\mathbb{Q}|\mathbb{P})) \leq n\beta\varepsilon$. This verifies part (ii), and hence completes the proof of the lemma.

Proof of Lemma 6.2

To begin, we bound k by examining the trace of B_n^2, and noting that, per Markov's inequality,

$$k \leq \sum_{i=1}^{n} \mu_i^2/\varepsilon^2 = \mathrm{tr}(B_n^2)/\varepsilon^2.$$

Also, we have

$$\mathrm{tr}(B_n^2) = \mathrm{tr}(A_n^2 + 2\lambda_{\min} A_n + \lambda_{\min}^2 I_n) = \mathrm{tr}(A_n^2) + n\lambda_{\min}^2. \tag{6.22}$$

Combining the above two displays, it suffices to show that the RHS of (6.22) is $o(n)$. Since $\lambda_{\min} = o(1)$, it suffices to show $\mathrm{tr}(A_n^2) = o(n)$, which is the focus of the rest of the proof.

To this effect, recall that $\mathrm{tr}(A_n) = 0$. Thus, with $\lambda_1(A_n), \ldots, \lambda_n(A_n)$ denoting the eigenvalues of A_n we have

$$\sum_{i=1}^{n} \lambda_i(A_n) 1\{\lambda_i(A_n) > 0\} = \sum_{i=1}^{n} |\lambda_i(A_n)| 1\{\lambda_i(A_n) < 0\} \leq n|\lambda_{\min}|.$$

This in turn gives

$$\sum_{i=1}^{n} |\lambda_i(A_n)| = \sum_{i=1}^{n} \lambda_i(A_n) 1\{\lambda_i(A_n) > 0\} + \sum_{i=1}^{n} |\lambda_i(A_n)| 1\{\lambda_i(A_n) < 0\} \leq 2n|\lambda_{\min}| = o(n),$$

and so

$$\mathrm{tr}(A_n^2) = \sum_{i=1}^{n} \lambda_i(A_n)^2 \leq \lambda_{\max} \sum_{i=1}^{n} |\lambda_i(A_n)| = o(n),$$

where the last equality uses the assumption $\lambda_{\max} = O(1)$. This completes the proof of the lemma.

6.6.2 Specialized Auxiliary Gaussian algorithm for $q = 2$

Standard Auxiliary Gaussian algorithm

Setting $\mathbf{W} := \mathbf{Z}_1 - \mathbf{Z}_2$ and using (6.3), under $\mathbb{P}$ we have

$$(\mathbf{W}|\mathbf{X} = \mathbf{x}) \sim N(\mathbf{y}_1 - \mathbf{y}_2, 2B_n^{-1}). \tag{6.23}$$

Also, using (6.4), we have

$$\begin{aligned}\mathbb{P}(X_i = 1|\mathbf{Z} = \mathbf{z}) =& \frac{\exp\left(\sum_{j=1}^n B_n(i,j)z_1(j)\right)}{\exp\left(\sum_{j=1}^n B_n(i,j)z_1(j)\right) + \exp\left(\sum_{j=1}^n B_n(i,j)z_2(j)\right)} \\ =& \frac{\exp\left(\frac{1}{2}\sum_{j=1}^n B_n(i,j)w(j)\right)}{\exp\left(\frac{1}{2}\sum_{j=1}^n B_n(i,j)w(j)\right) + \exp\left(-\frac{1}{2}\sum_{j=1}^n B_n(i,j)w(j)\right)},\end{aligned} \tag{6.24}$$

where $w_j = z_1(j) - z_2(j)$. A similar calculation gives

$$\mathbb{P}(X_i = 2|\mathbf{Z} = \mathbf{z}) = \frac{\exp\left(-\frac{1}{2}\sum_{j=1}^n B_n(i,j)w(j)\right)}{\exp\left(\frac{1}{2}\sum_{j=1}^n B_n(i,j)w(j)\right) + \exp\left(-\frac{1}{2}\sum_{j=1}^n B_n(i,j)w(j)\right)}.$$

Thus, instead of using the $2n$ dimensional Gaussian $\mathbf{Z}$, one can use the n dimensional Gaussian W, and iterate between (6.23) and (6.24), This is the same exact algorithm, but we work with a lower dimensional representation of the auxiliary variable, which helps in faster computations. We note that this is the exact same algorithm as Martens and Sutskever (2010) in the Ising case.

Low rank Auxiliary Gaussian algorithm

Setting $\mathbf{W} := \mathbf{Z}_1 - \mathbf{Z}_2$ as before, using (6.8) under $\mathbb{Q}$ we have

$$(\mathbf{W}_j|\mathbf{X} = \mathbf{x}) \sim N(\mathbf{p}_j'(\mathbf{y}_1 - \mathbf{y}_2), 2/\mu_j), \tag{6.25}$$

with $(W_1, \ldots, W_k)$ mutually independent. Also, using (6.9), we have

$$\begin{aligned}\mathbb{Q}(X_i = 1|\mathbf{Z} = \mathbf{z}) =& \frac{\exp\left(\sum_{j=1}^k \mu_j z_1(j) p_j(i)\right)}{\exp\left(\sum_{j=1}^k \mu_j z_1(j) p_j(i)\right) + \exp\left(\sum_{j=1}^k \mu_j z_1(j) p_j(i)\right)} \\ =& \frac{\exp\left(\frac{1}{2}\sum_{j=1}^k \mu_j w_j p_j(i)\right)}{\exp\left(\frac{1}{2}\sum_{j=1}^k \mu_j w_j p_j(i)\right) + \exp\left(-\sum_{j=1}^k \mu_j w_j p_j(i)\right)}, \end{aligned} \tag{6.26}$$

where $w_j = z_1(j) - z_2(j)$. A similar calculation gives

$$\mathbb{Q}(X_i = 2|\mathbf{Z} = \mathbf{z}) = \frac{\exp\left(-\frac{1}{2}\sum_{j=1}^k \mu_j w_j p_j(i)\right)}{\exp\left(\frac{1}{2}\sum_{j=1}^k \mu_j w_j p_j(i)\right) + \exp\left(-\sum_{j=1}^k \mu_j w_j p_j(i)\right)}.$$

Thus, instead of using the $2k$ dimensional Gaussian $\mathbf{Z}$, one can use the k dimensional Gaussian W, and iterate between (6.25) and (6.26).

6.6.3 Overview of the Swendsen-Wang and Wolff algorithms

The Heat Bath updates one node, x_j, at a time, conditional on the distribution of the other nodes, $\mathbf{x}_{-j}$. As a result, it can take many iterations to meaningfully change $\mathbf{x}$ and the produced samples are strongly correlated. The Swendsen-Wang algorithm attempts to address this issue by updating large clusters of particles at each iteration. This is done by introducing an auxiliary random graph, as illustrated in Figure 6.5. Below we provide a sketch of the algorithm:

1. Construct auxiliary clusters until every node belongs to a cluster, per the following steps:

   (a) Start a cluster at a random node, x_i, which does not already belong to an auxiliary cluster (call this a *free* node).

   (b) For each free neighbor x_j of x_i, i.e. such that $A_n(i,j) > 0$, if $x_i = x_j$, add x_j to the cluster with probability

$$p(\beta) = 1 - \exp(-2\beta A_n(i,j))$$

(c) Continue growing the cluster by repeating the above step for each added node. Once the cluster stops growing, start a new cluster at one of the free nodes.

2. Once all the nodes have been assigned to a cluster, randomly assign a new state in $[q]$ to each cluster, where each state has equal probability.

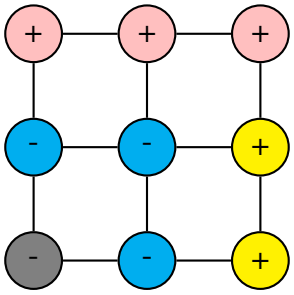

Figure 6.5: Color-coded random auxiliary clusters on a grid graph. *In the Ising model ($q = 2$), each particle takes values in $\{-, +\}$, referring to the two states of the Ising model. The Swendsen-Wang algorithm randomly connects nodes in the same state to construct auxiliary clusters: here 4 auxiliary clusters are constructed (pink, blue, yellow, and gray).*

Remark a. *The algorithm can be generalized to the case where the Potts model admits an exterior magnetic field, which favors certain states over the others, by adjusting the probability in step 1(b).*

The Wolff algorithm only constructs one auxiliary cluster per iteration, even if this means not every node is assigned to an auxiliary cluster.The motivation for this is to not waste computational time building small clusters, as we do once there are only a few free nodes left. Another change is that in the Ising case ($q = 2$), the Wolff algorithm always flips the state of the cluster, rather than assigning $\{-1, +1\}$ with equal probability.

6.6.4 Overview of the tempering algorithm

A tempering algorithm runs multiple chains or *replicas* over a sequence of temperatures for a set number of iterations and then exchanges, with a certain probability, the states of two replicas with a neighboring temperature. The probability of exchanging two neighboring states, with

inverse temperatures β_t and β_{t+1}, can be worked out from the detailed balance condition, following the argument by Hukushima and Nemeto (1996). For convenience, we introduce the *partial Hamiltonian*

$$\mathcal{H}(A_n, \mathbf{x}_t) = -\frac{1}{2}\sum_{i,j=1}^{n} A_n(i,j)1\{x_{t,i} = x_{t,j}\}$$

Assume $\beta_1 < ... < \beta_t < \beta_{t+1} < ... < \beta_T$. Consider the joint distribution over T replicas, $\{\mathbf{X}\} = \{\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_T\}$. The replicas are mutually independent, with joint p.m.f.

$$\mathbb{P}(\mathbf{X}_1 = \mathbf{x}_1, ..., \mathbf{X}_T = \mathbf{x}_T) = \prod_{t=1}^{T} \frac{\exp\left(\frac{\beta_t}{2}\sum_{i,j}^{n} A_n(i,j)1\{x_{t,i} = x_{t,j}\}\right)}{Z(\beta_t, A_n)} = \prod_{t=1}^{T} \frac{\exp\left(-\beta_t \mathcal{H}(A_n, \mathbf{x}_t)\right)}{Z(\beta_t, A_n)}.$$

We now introduce the transition kernel, $\Gamma(\mathbf{X}_t, \beta_t \mid \mathbf{X}_{t+1}, \beta_{t+1})$, which is the probability of exchanging the t^{th} and $(t+1)^{\text{th}}$ replicas. The detailed balance equation is satisfied, if

$$\begin{aligned}
&\mathbb{P}(\mathbf{X}_1 = \mathbf{x}_1, ..., \mathbf{X}_t = \mathbf{x}_t, \mathbf{X}_{t+1} = \mathbf{x}_{t+1}, ..., \mathbf{X}_T = \mathbf{x}_T)\Gamma(\mathbf{X}_t, \beta_t \mid \mathbf{X}_{t+1}, \beta_{t+1}) \\
&= \mathbb{P}(\mathbf{X}_1 = \mathbf{x}_1, ..., \mathbf{X}_t = \mathbf{x}_{t+1}, \mathbf{X}_{t+1} = \mathbf{x}_t, ..., \mathbf{X}_T = \mathbf{x}_T)\Gamma(\mathbf{X}_{t+1}, \beta_t \mid \mathbf{X}_t, \beta_{t+1}).
\end{aligned}$$

Thus

$$\begin{aligned}
\frac{\Gamma(\mathbf{X}_t, \beta_t \mid \mathbf{X}_{t+1}, \beta_{t+1})}{\Gamma(\mathbf{X}_{t+1}, \beta_t \mid \mathbf{X}_t, \beta_{t+1})} &= \frac{\exp\left(-\beta_t \mathcal{H}(A_n, \mathbf{x}_{t+1}) - \beta_{t+1}\mathcal{H}(A_n, \mathbf{x}_t)\right)}{\exp\left(-\beta_t \mathcal{H}(A_n, \mathbf{x}_t) - \beta_{t+1}\mathcal{H}(A_n, \mathbf{x}_{t+1})\right)} \\
&= \exp\left((\beta_{t+1} - \beta_t)(\mathcal{H}(A_n, \mathbf{x}_{t+1}) - \mathcal{H}(A_n, \mathbf{x}_t)\right).
\end{aligned}$$

From this we deduce a Metropolis update with acceptance probability

$$\begin{aligned}
&\Gamma(\mathbf{X}_t, \beta_t \mid \mathbf{X}_{t+1}, \beta_{t+1}) \\
=&\min\left(1, \exp\left[(\beta_{t+1} - \beta_t)(\mathcal{H}(A_n, \mathbf{x}_{t+1}) - \mathcal{H}(A_n, \mathbf{x}_t)\right]\right) \\
=&\min\left(1, \exp\left[\frac{1}{2}(\beta_{t+1} - \beta_t)\left(-\sum_{i,j=1}^{n} A_n(i,j)1\{x_{t+1,i} = x_{t+1,j}\} + \sum_{i,j=1}^{n} A_n(i,j)1\{x_{t,i} = x_{t,j}\}\right)\right]\right). \quad (6.27)
\end{aligned}$$

Algorithm 6.2 provides the pseudo-code for a practical implementation.

Notation for the algorithm

- $\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_T)$: an array containing the inverse temperature for each replica, where T is the number of replicas.
- $\mathbf{x}_t^{(i)}$: the i^{th} sample for t^{th} replica.
- n_{ex}: the number of exchange steps.
- n_{mc}: the number of sampling iterations between exchange steps. Hence the total number of generated samples is $n_{\text{ex}} \times n_{\text{mc}}$.
- γ: the sampler used between exchanges.

Algorithm 6.2: Tempering

input: $\left(\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, ..., \mathbf{x}_T^{(0)}\right)$, $\boldsymbol{\beta}$, n_{ex}, n_{mc}, γ
for $i \in \{1, \cdots, n_{\text{ex}}\}$ **do**
  for $t \in \{1, \cdots, T\}$ **do**
    Draw $\mathbf{x}_t^{[(i-1)n_{\text{mc}}+1]:[in_{\text{mc}}]}$ using γ.
  end
  for $t \in \{1, ..., T-1\}$ **do**
    $\mathcal{H}(\mathbf{x}_t) \leftarrow -0.5 \sum_{\ell=1}^{q} \left(\mathbf{y}_{\ell,t}^{(in_{\text{mc}})}\right)' A_n \left(\mathbf{y}_{\ell,t}^{(in_{\text{mc}})}\right)$
    $\mathcal{H}(\mathbf{x}_{t+1}) \leftarrow -0.5 \sum_{\ell=1}^{q} \left(\mathbf{y}_{\ell,t+1}^{(in_{\text{mc}})}\right)' A_n \left(\mathbf{y}_{\ell,t+1}^{(in_{\text{mc}})}\right)$
    $\Delta \leftarrow (\beta_{t+1} - \beta_t)\left(\mathcal{H}(\mathbf{x}_{t+1}) - \mathcal{H}(\mathbf{x}_t)\right)$
    Draw $u \sim \text{uniform}(0, 1)$.
    if $(u \leq \exp(\Delta))$ **then**
      $\mathbf{x}_{\text{saved}} \leftarrow \mathbf{x}_{t+1}^{(in_{\text{mc}})}$
      $\mathbf{x}_t^{in_{\text{mc}}} \leftarrow \mathbf{x}_{t+1}^{(in_{\text{mc}})}$
      $\mathbf{x}_{t+1}^{in_{\text{mc}}} \leftarrow \mathbf{x}_{\text{saved}}$
    end
  end
end
return: $\mathbf{x}_1^{(1)}, ..., \mathbf{x}_1^{(n_{\text{ex}}n_{\text{mc}})}, ..., \mathbf{x}_T^{(1)}, ..., \mathbf{x}_T^{(n_{\text{ex}}n_{\text{mc}})}$

Remark b. *The for loop at line 3 of Algorithm 6.2 can be parallelized, since between exchanges, samples for the replicas are generated independently.*

Remark c. *We can rewrite line 5 to avoid redundant computations of $\mathcal{H}(\mathbf{x}_t)$, and insure that only one Hamiltonian is evaluated per iteration of the* ***for*** *loop at line 4. This requires some careful book-keeping, which we omitted to make the algorithm more readable. The computational gain from this step is marginal.*

Remark d. *One can in theory use a different sampler for different replicas, though preliminary tests suggest there are no benefits to doing this.*

In our numerical experiment on Spin Glass models (Section 6.3.3), we use $n_{\text{ex}} = 40$ and $n_{\text{mc}} = 1000$. Our experiment does not parallelize replicas.

References

Agrawal, Raj, et al. 2019. "The Kernel Interaction Trick: Fast Bayesian Discovery of Pairwise Interactions in High Dimensions". *Proceedings of the 36th International Conference on Machine Learning* 97.

Bales, Ben, et al. 2019. "Selecting the Metric in Hamiltonian Monte Carlo". *arXiv:1905.11916*.

Basak, Anirban, and Sumit Mukherjee. 2017. "Universality of the mean-field for the Potts model". *Probability Theory and Related Fields* 168:557 –600.

Baydin, Atilim Gunes, et al. 2018. "Automatic differentiation in machine learning: a survey". *Journal of Machine Learning Research* 18:1 –43.

Bell, Bradley M, and James V Burke. 2008. "Algorithmic Differentiation of Implicit Functions and Optimal Values". In *Advances in Automatic Differentiation. Lecture Notes in Computational Science and Engineering*, ed. by C.H Bischof et al., vol. 64. Springer, Berlin, Heidelberg.

Betancourt, M, CC Margossian, and V Leos-Barajas. 2020. "The Discrete Adjoint Method: Efficient Derivatives for Functions of Discrete Sequences". *arXiv:2002.00326*.

Betancourt, Michael. 2018a. "A Conceptual Introduction to Hamiltonian Monte Carlo". *arXiv:1701.02434v1*.

— . 2013. "A general metric for Riemannian manifold Hamiltonian Monte Carlo". *arXiv:1212.4693*.

— . 2018b. "A Geometric Theory of Higher-Order Automatic Differentiation". *arXiv:1812.11592*.

— . 2020. "Towards a principled Bayesian workflow". *betanalpha.github.io/assets/case_studies/principled_bayes*

Betancourt, Michael, and Mark Girolami. 2015. "Current Trends in Bayesian Methodology with Applications". Chap. Hamiltonian Monte Carlo for Hierarchical Models. Chapman / Hall/CRC.

Betancourt, Michael, et al. 2017. "The Geometric Foundations of Hamiltonian Monte Carlo". *Bernoulli* 23 (4A): 2257 –2298.

Betancout, Michael. 2021. "A Short Review of Ergodicity and Convergence of Markov chain Monte Carlo Estimators". *arXiv:2110.07032*.

Blei, David M. 2014. "Build, Compute, Critique, Repeat: Data Analysis with Latent Variable Models". *Annual Review of Statistics and Its Application* 1.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe. 2017. "Variational Inference: A Review for Statisticians". *Journal of the American Statistical Association* 112 (518). arXiv: 1601.00670.

Blom, Gunnar. 1958. *Statistical Estimates and Transformed Beta-Variables*. Ed. by New York Wiley.

Bollobás, B, G Grimmett, and S Janson. 1996. "The random-cluster model on the complete graph". *Probability Theory and Related Fields* 104:283 –317.

Box, G. E. P, and N.R Draper. 1987. *Empirical model-building and response surfaces*. Ed. by England: John Wiley & Sons Oxford.

Bradbury, James, et al. 2018. *JAX: composable transformations of Python+NumPy programs*. Version 0.2.5.

Broussard, Julien, et al. 2021. "Understanding Priors for Bayesian neural networks". *Final project for STAT 6103 GR: Theory of Machine Learning*.

Cao, Y, et al. 2002. "Adjoint Sensitivity Analysis for Differential-Algebraic Equations: The Adjoint DAE System and Its Numerical Solution". *SIAM Journal on Scientific Computing* 24 (3): 1076 –1089.

Carpenter, Bob, et al. 2017. "Stan: A Probabilistic Programming Language". *Journal of Statistical Software* 76 (1): 1 –32.

Carpenter, Bob, et al. 2015. "The Stan Math Library: Reverse-Mode Automatic Differentiation in C++". *arXiv 1509.07164*.

Carvalho, Carlos M., Nicholas G. Polson, and James G. Scott. 2010. "The Horseshoe estimator for sparse signals". *Biometrika* 97 (2): 465–480.

Casella, George, and Roger L Berger. 2002. *Statistical Inference*. Wadsworth.

Cohen, Scott D, and Alan C Hindmarsh. 1996. "CVODE, a Stiff/Nonstiff ODE Solver in C". *Computers in Physics* 10 (2): 138 –143.

Cseke, Botond, and Tom Heskes. 2011. "Approximate marginals in latent Gaussian models". *Journal of Machine Learning Research* 12 (2).

Cuff, P, et al. 2012. "Glauber Dynamics for the Mean-Field Potts Model". *Journal of Statistical Physics* 149:432 –477.

Dai, Chenguang, et al. 2020. "An invitation to sequential Monte Carlo samplers". *arXiv:2007.11936*.

Dhaka, Akash Kumar, et al. 2020. “Robust, Accurate Stochastic Optimization for Variational Inference”. In *Advances in Neural Information Processing Systems 34*, to appear.

Dillon, Joshua V, et al. 2017. “Tensorflow distributions”. *arXiv preprint arXiv:1711.10604*.

Dormand, John R, and Peter J Prince. 1980. “A Family of Embedded Runge-Kutta Formulae”. *Journal of Computational and Applied Mathematics* 6 (1): 19 –26.

Dua, D., and C. Graff. 2017. *UCL machine learning repository*.

Duane, S., et al. 1987. “Hybrid Monte Carlo”. *Physics Letters B* 195:216 –222.

Errico, M. 1997. “What is an adjoint model?” *Bulletin of the American Meteorological Society* 78:2577 –2591.

Ferrenberg, Alan M, Jiahao Xu, and David P Landau. 2018. “Pushing the Limits of Monte Carlo Simulations for the 3d Ising Model”. *Physical Review E* 97.

Fisher, Ronald R, and Frank Yates. 1938. *Statistical Tables for Biological, Agricultural, and Medical Research*. Ed. by Edinburgh Oliver & Boyd.

Friberg, LE, et al. 2002. “Model of chemotherapy-induced myelosuppression with parameter consistency across drugs”. *Journal of Clinical Oncology* 20 (24): 4713 –4721.

Friedman, Milton. 1937. “The use of ranks to avoid the assumption of normality implicit in the analysis of variance”. *Journal of the American Statistical Association* 32 (200): 675 –701.

Gabry, Jonah, Paul Buekner, and Matthew Kay. 2021. *posterior*.

Gabry, Jonah, et al. 2019. “Visualization in Bayesian workflow”. *Journal of the Royal Statistical Society, Series A* 182:389 –402.

Gaebler, Johann D. 2021. *Autodiff for Implicit Functions in Stan*.

Gardner, Jacob R, et al. 2018. “GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration”. In *Advances in Neural Information Processing Systems*.

Gastonguay, Marc, and Metrum Institute Facility. 2013. *MI-210: Essentials of Population PKPD Modeling and Simulation*.

Ge, Hong, Kai Xu, and Zoubin Ghahramani. 2018. “Turing: a language for flexible probabilistic inference”. *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*.

Gelman, Andrew, Frederic Bois, and Jiming Jiang. 1996. “Physiological Pharmacokinetic Analysis Using Population Modeling and Informative Prior Distributions”. *Journal of the American Statistical Association* 91 (436): 1400 –1412.

Gelman, Andrew, and Thomas Little. 1997. “Poststratification into many categories using hierarchical logistic regression”. *Survey Methodology* 23:127 –135.

Gelman, Andrew, and D. Rubin. 1992. “Inference from Iterative Simulation using Multiple Sequences”. *Statistical Science* 7:457 –511.

Gelman, Andrew, et al. 2013. *Bayesian Data Analysis*. Third. London: Chapman &Hall/CRC Press.

Gelman, Andrew, et al. 2020. “Bayesian Workflow”. *arXiv:2011.01808*.

Geman, Stuart, and Donald Geman. 1984. “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (6): 721 –741.

Geyer, Charles. 2012. “Introduction to Markov chain Monte Carlo”. In *Handbook of Markov chain Monte Carlo*. Brooks, Steve.

Gibaldi, Milo, and Donald Perrier. 1982. *Pharmacokinetics*. 2nd ed. Ed. by CRC Press. ISBN: 9780824710422.

Girolami, M, B Calderhead, and S A Chin. 2019. “Riemannian Manifold Hamiltonian Monte Carlo”. *arXiv:0907.1100*.

Girolami, Mark, and Ben Calderhead. 2011. “Riemann manifold Langevin and Hamiltonian Monte Carlo methods”. *Journal of the Royal Statistical Society B* 73 (2): 123–214.

Gómez-Rubio, Virgilio, and Havard Rue. 2018. “Markov chain Monte Carlo with the Integrated Nested Laplace Approximation”. *Statistics and Computing* 28:1033 –1051.

Gower, R. M., and M. P. Mello. 2011. “A new framework for the computation of Hessians”. *Optimization methods and software* 27 (2): 251–273.

Grathwohl, Will, et al. 2021. “Oops I Took A Gradient: Scalable Sampling for Discrete Distributions”. *Proceedings of the 2021 International Conference on Machine Learning*.

Greengard, Philip, et al. 2021. “Fast methods for posterior inference of two-group normal-normal models”. *Preprint. arXiv:2110.03055.*

Griewank, Andreas, and Andrea Walther. 2008. *Evaluating derivatives*. Second. Society for Industrial / Applied Mathematics (SIAM), Philadelphia, PA.

Grinsztajn, Léo, et al. 2021. “Bayesian workflow for disease transmission modeling in Stan”. *Statistics in Medicine*.

Hastings, W. 1970. “Monte Carlo sampling methods using Markov chains and their application”. *Biometrika* 57.

Hauser, Anthony, et al. 2020. “Estimation of SARS-CoV-2 mortality during the early stages of an epidemic: a modeling study in Hubei, China and six regions in Europe”. *PLOS Medicine* 17 (7).

Hindmarsh, A, and R Serban. 2020. “User Documentation for CVODES v5.1.0”. *Technical Report*.

Hoffman, M. D., and Y. Ma. 2020. “Black-Box Variational Inference as a Parametric Approximation to Langevin Dynamics”. *International Conference on Machine Learning*.

Hoffman, M. D., A. Radul, and P. Sountsov. 2021. “An Adaptive MCMC Scheme for Setting Trajectory Lengths in Hamiltonian Monte Carlo”. *International Conference on Artificial Intelligence and Statistics*.

Hoffman, Matthew D., and Andrew Gelman. 2014. “The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo”. *Journal of Machine Learning Research* 15 (1): 1593–1623.

Huggins, Jonathan, et al. 2020. “Validated Variational Inference via Practical Posterior Error Bounds”, ed. by Silvia Chiappa and Roberto Calandra, 108:1792–1802. Proceedings of Machine Learning Research. Online: PMLR.

Hukushima, Koji, and Koji Nemeto. 1996. “Exchange Monte Carlo Method and Application to Spin Glass Simulations”. *Journal of the Physical Society of Japan* 65 (6): 1604 –1608.

Hukushima, Koji, Hajime Takajama, and Hajime Yoshina. 1998. “Exchange Monte Carlo Dynamics in the SK Model”. *Journal of the Physical Society of Japan* 67 (1): 12 –15.

Ising, Ernst. 1925. “Beitrag zur Theorie des Ferromagnetismus”. *Zeitschrift für Physik* 31 (1).

Jacod, Jean, and Philip Protter. 2004. *Probability Essentials*. 2nd. Ed. by Springer.

Joensuu, Heikki, et al. 2012. “Risk of recurrence of gastrointestinal stromal tumour after surgery: an analysis of pooled population-based cohorts”. *The Lancet Oncology* 13 (3): 265–274.

Jylänki, Pasi, Jarno Vanhatalo, and Aki Vehtari. 2011. “Robust Gaussian Process Regression with a Student-*t* Likelihood”. *Journal of Machine Learning Research* 12:3227 –3257.

Karsten, Ahnert, and Mario Mulansky. 2011. “Odeint–Solving Ordinary Differential Equations in C++”. *arXiv 1110.3397*.

Karush, W. 1939. “Minima of Functions of Several Variables with Inequalities as Side Constraints”. *(M.Sc. thesis). Dept. of Mathematics, Univ. of Chicago, Chicago, Illinois*.

Katzgraber, Helmut G., Matteo Palassini, and A. P. Young. 2001. “Monte Carlo Simulations of Spin Glasses at Low Temperatures”. *Physical Review B* 63 (18).

Kesavan, S. 2020. *Nonlinear functional analysis—a first course*. Second. 28:xii+164. Texts and Readings in Mathematics. Hindustan Book Agency, New Delhi. ISBN: 978-93-86279-85-9.

Kristensen, Kasper, et al. 2016. “TMB: Automatic Differentiation and Laplace Approximation”. *Journal of statistical software* 70 (5): 1 –21.

Kucukelbir, Alp, et al. 2017. “Automatic differentiation variational inference”. *Journal of machine learning research* 18 (14): 1 –45.

Kuhn, H. W., and A. W. Tucker. 1951. “Nonlinear programming”. Ed. by Berkeley: University of California Press. *Proceedings of 2nd Berkeley Symposium*: 481–492.

Kuss, Malte, and Carl E Rasmussen. 2005. “Assessing Approximate Inference for Binary Gaussian Process Classification”. *Journal of Machine Learning Research* 6:1679 –1704.

Lambert, J. D. 1992. *Numerical Methods for Ordinary Differential Systems*. Ed. by New York: Wiley. ISBN: 978-0-471-92990-1.

Landau, D, and K Binder. 2009. *A Guide to Monte Carlo Simulations in Statistical Physics*. CAMBRIDGE UNIVERSITY PRESS.

Lao, J., et al. 2020. “tfp.mcmc: Modern Markov chain Monte Carlo tools built for modern hardware”. *arXiv:2002.01184*.

Laplace, Pierre Simon. 1774. *Mémoire de Mathématique et de Physique, Tome Sixième*.

Lavoisier, Antoine Laurent. 1789. *Traité élémentaire de chimie*. Ed. by Cuchet.

Li, Xuechen, et al. 2020. “Scalable Gradients for Stochastic Differential Equations”. *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics, PMLR*.

Lorraine, J, P Vicol, and D Duvenaud. 2019. “Optimizing Millions of Hyperparameters by Implicit Differentiation”. *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics, PMLR* 108:1540 –1552.

Lu, Xiuyuan, and Benjamin Van Roy. 2017. “Ensemble Sampling”. *Advances in Neural Information Processing Systems (NeurIPS)* 30.

Lunn, DJ, et al. 2000. “WinBUGS — a Bayesian modelling framework: concepts, structure, and extensibility”. *Statistics and Computing* 10:325 –337.

Margossian, Charles C. 2019. “A Review of automatic differentiation and its efficient implementation”. *Wiley interdisciplinary reviews: data mining and knowledge discovery* 9 (4).

Margossian, Charles C, and Michael Betancourt. 2022. “Efficient Automatic Differentiation of Implicit Functions”. *arXiv:2112.14217*.

Margossian, Charles C, and Andrew Gelman. 2020. “Bayesian model of planetary motion: exploring ideas for a modeling workflow when dealing with ordinary differential equations and multimodality”. In *Stan Case Studies*, vol. 7.

Margossian, Charles C, and William R Gillespie. 2017. “Gaining Efficiency by Combining Analytical and Numerical Methods to Solve ODEs: Implementation in Stan and Application to Bayesian PK/PD”. *Journal of Pharmacokinetics and Pharmacodynamics* 44.

Margossian, Charles C, Matthew D Hoffman, and Pavel Sountsov. 2021. “Nested $\hat{R}$: assessing convergence for Markov chains Monte Carlo when using many short chains”. *arXiv:2110.13017.*

Margossian, Charles C, and Sumit Mukherjee. 2021. “Simulating Ising and Potts models at critical and cold temperatures using auxiliary Gaussian variables”. *arXiv:2110.10801*.

Margossian, Charles C, Yi Zhang, and William R Gillespie. 2022. “Flexible and efficient Bayesian pharmacometrics modeling using Stan and Torsten, Part I”. *arXiv:2109.10184.*

Margossian, Charles C, et al. 2020a. “Approximate Bayesian inference for latent Gaussian models in Stan”. In *StanCon 2020*.

— . 2020b. “Hamiltonian Monte Carlo using an adjoint-differentiated Laplace approximation: Bayesian inference for latent Gaussian models and beyond”. *Advances in Neural Information Processing Systems (NeurIPS)* 33.

Margossian, Charles C, et al. 2021. “Solving ODEs in a Bayesian context: challenges and opportunities”. In *Population Approach Group in Europe (PAGE) 29*.

Martens, James, and Ilya Sutskever. 2010. “Parallelizable Sampling of Markov Random Fields”. *Proceedings of the* 13^{th} *International Conference on Artificial Intelligence and Statistics (AISTATS)* 9.

Metropolis, N, et al. 1953. “Equations of state calculations by fast computing machines”. *Journal of Chemical Physics* 26 (6).

Monnahan, Cole C, and Kasper Kristensen. 2018. “No-U-turn sampling for fast Bayesian inference in ADMB and TMB: Introducing the adnuts and tmbstan R packages”. *Plos One* 13 (5).

Monroe, James L. 2002. “Critical temperature estimates for higher-spin Ising and Potts models”. *Physical Review* 66.

Mukherjee, Rajarshi, Sumit Mukherjee, and Ming Yuan. 2018. “Global testing against sparse alternatives under Ising models”. *Annals of Statistics* 46 (5).

Mukherjee, Sumit, and Yuanzhe Xu. 2021. “Statistics of the two-star ERGM”. *arXiv:1310.4526*.

Neal, Radford M. 2012. “MCMC using Hamiltonian Dynamics”. In *Handbook of Markov Chain Monte Carlo*. Chapman & Hall / CRC Press.

Neal, Radford M. 1993. “Probabilistic Inference Using Markov Chain Monte Carlo Methods”. *Technical Report CRG-TR-93-1*.

Neal, Radford M. 2003. “Slice sampling”. *Annals of statistics* 31 (3): 705 –767.

Papaspiliopoulos, Omiros, Gareth O Roberts, and Matin Sköld. 2007. “A General Framework for the Parametrization of Hierarchical Models”. *Statistical Sciences* 22 (1): 59 –73.

Peterson, Mark C, and Matthew M Riggs. 2010. “A physiologically based mathematical model of integrated calcium homeostasis and bone remodeling”. *Bone* 46 (1): 49 –63.

Piironen, Juho, and Aki Vehtari. 2017. “Sparsity information and regularization in the horseshoe and other shrinkage priors”. *Electronic Journal of Statistics* 11 (2): 5018–5051.

Plummer, Martyn. 2003. *JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling*.

Pontryagin, LS, et al. 1963. *The Mathematical Theory of Optimal Processes*. Ed. by John Wiley and New York Sons Inc.

Potts, Renfrey B. 1952. “Some Generalized Order-Disorder Transformations”. *Mathematical Proceedings* 48 (1): 106 –109.

Rackauckas, Christopher, et al. 2018. “A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions”. *arXiv:1812.01892*.

Ranganath, Rajesh, Sean Gerrish, and David M Blei. 2014. “Black Box Variational Inference”. *Artificial Intelligence and statistics*: 814 –822.

Rasmussen, C. E., and Hannes Nickish. 2010. “Gaussian processes for machine learning (GPML) toolbox”. *Journal of Machine Learning Research* 11:3011 –3015.

Rasmussen, C. E., and C. K. I. Williams. 2006. *Gaussian Processes for Machine Learning*. The MIT Press.

Riihimäki, Jaakko, and Aki Vehtari. 2014. “Laplace Approximation for Logistic Gaussian Process Density Estimation and Regression”. *Bayesian analysis* 9 (2): 425 –448.

Robert, Christian P, and George Casella. 2011. *Statistical Science* 26 (1): 102 –115.

— . 2004. *Monte Carlo Statistical Methods*. Second. Ed. by Springer Science+Business Media.

Roberts, Gareth O, and Jeffrey S Rosenthal. 2004. “General state space Markov chains and MCMC algorithms”. *Probability Surveys* 1:20 –71.

Rubin, Donald B. 1981. “Estimation in parallelized randomized experiments”. *Journal of Educational Statistics* 6 (4): 377 –400.

Rue, Havard, Sara Martino, and Nicolas Chopin. 2009. “Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations”. *Journal of Royal Statistics B* 71 (2): 319 –392.

Rue, Havard, et al. 2017. “Bayesian Computing with INLA: A Review”. *Annual Review of Statistics and its Application* 4:395 –421.

Salvatier, John, Thomas V. Wiecki, and Christopher Fonnesbeck. 2016. “Probabilistic programming in Python using PyMC3”. *PeerJ Computer Science* 2 (e55).

Serban, Radu, and Alan C Hindmarsh. 2005. “CVODES: The Sensitivity-Enabled ODE Solver in SUNDIALS.” *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*: 257 –269.

Sountsov, Pavel, and Matthew D Hoffman. 2021. “Focusing on Difficult Directions for Learning HMC Trajectory Lengths”. *arXiv:2110.11576*.

Stan development team. 2020. *Stan reference manual*.

Swendsen, Robert H, and J.-S Wang. 1987. “Nonuniversal critical dynamics in Monte Carlo simulations”. *Physics review letter* 58 (2).

Swendsen, Robert H, and Jian-Sheng Wang. 1986. “Replica Monte Carlo Simulation of Spin-Glasses”. *Physical Review letters* 57 (21).

Talts, Sean, et al. 2020. “Validating Bayesian inference algorithms with simulation-based calibration”. *arXiv:1804.06788v1*.

Tierney, Luke, and Joseph B. Kadane. 1986. “Accurate Approximations for Posterior Moments and Marginal Densities”. *Journal of the American Statistical Association* 81 (393): 82–86. eprint: https://amstat.tandfonline.com/doi/pdf/10.1080/01621459.1986.10478240.

Tolvanen, Ville, Pasi Jylänki, and Aki Vehtari. 2014. "Expectation propagation for nonstationary heteroscedastic Gaussian process regression". *Machine Learning for Signal Processing (MLSP), 2014 IEEE International Workshop on*.

Vanhatalo, Jarno, Scott D. Foster, and Geoffrey Hosack. 2021. "Spatiotemporal clustering using Gaussian processes embedded in a mixture model". *Environmetrics* 32 (7).

Vanhatalo, Jarno, Pasi Jylänki, and Aki Vehtari. 2009. "Gaussian process regression with Student-*t* likelihood". In *Advances in Neural Information Processing Systems 22*, ed. by Y. Bengio et al., 1910–1918. Curran Associates, Inc.

Vanhatalo, Jarno, Ville Pietiläinen, and Aki Vehtari. 2010. "Approximate inference for disease mapping with sparse Gaussian processes". *Statistics in Medicine* 29 (15): 1580–1607.

Vanhatalo, Jarno, et al. 2013. "GPstuff: Bayesian Modeling with Gaussian Processes". *Journal of Machine Learning Research* 14:1175–1179.

Vats, Dootika, and D. Knudson. 2021. "RRevisiting the Gelman-Rubin Diagnostic". *Statistical Science*.

Vehtari, Aki. 2021. *Gaussian process demonstration with Stan*.

Vehtari, Aki, et al. 2016. "Bayesian Leave-One-Out Cross-Validation Approximations for Gaussian Latent Variable Models". *Journal of Machine Learning Research* 17 (103): 1–38.

Vehtari, Aki, et al. 2019. "Pareto smoothed importance sampling". *arXiv:1507.02646*.

Vehtari, Aki, et al. 2020. "Rank-normalization, folding, and localization: An improved $\hat{R}$ for assessing convergence of MCMC". *arXiv:1903.08008*.

Willard, Jared, et al. 2021. "Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems". *arXiv:2003.04919*.

Wolff, Ulli. 1989. "Collective Monte Carlo Updating for Spin Systems". *Physical review journal* 62 (4).

Yao, Yuling, et al. 2018. "Yes, but Did It Work?: Evaluating Variational Inference", 80:5581–5590. Proceedings of Machine Learning Research. PMLR.

Yucesoy, Burcu. 2013. *Replica Exchange Monte Carlo Simulations of the Ising Spin Glass: Static and Dynamic Properties*. Vol. 832.

Zhang, Lu, et al. 2021. "Pathfinder: Parallel quasi-Newton variational inference". *arXiv:2108.03782*.

Zhang, Y., et al. 2020. “Speed up population Bayesian inference by combining cross-chain warmup and within-chain parallelization”. *Journal of Pharmacokinetics and Pharmacodynamics* 47.

Zhang, Yichuan, and Charles Sutton. 2014. “Semi-Separable Hamiltonian Monte Carlo for Inference in Bayesian Hierarchical Models”. In *Advances in Neural Information Processing Systems 27*, 10–18. Curran Associates, Inc.

Zhang, Yichuan, et al. 2012. “Continuous Relaxations for Discrete Hamiltonian Monte Carlo”. *Advances in Neural Information Processing Systems (NeurIPS)* 25.

Appendix A: Efficient Automatic Differentiation of implicit functions

CHARLES C. MARGOSSIAN[1] AND MICHAEL BETANCOURT[2]

Derivative-based algorithms are ubiquitous in statistics, machine learning, and applied mathematics. Automatic differentiation offers an algorithmic way to efficiently evaluate these derivatives from computer programs that execute relevant functions. Implementing automatic differentiation for programs that incorporate implicit functions, such as the solution to an algebraic or differential equation, however, requires particular care. Contemporary applications typically appeal to either the application of the implicit function theorem or, in certain circumstances, specialized adjoint methods. In this paper we show that both of these approaches can be generalized to *any* implicit function, although the generalized adjoint method is typically more effective for automatic differentiation. To showcase the relative advantages and limitations of the two methods we demonstrate their application on a suite of common implicit functions.

A.1 Introduction

Automatic differentiation is a powerful tool for algorithmically evaluating derivatives of functions implemented as computer programs (Griewank and Walther 2008; Baydin et al. 2018; Margossian 2019). The method is implemented in an increasing diversity of software packages such as Stan (Carpenter et al. 2015; Carpenter et al. 2017) and Jax (Bradbury et al. 2018), driving state of the art computational tools such as the aforementioned Stan and TensorFlow (Dillon et al. 2017). Each automatic differentiation package provides a library of differentiable expressions and routines that propagate derivatives through programs comprised of those expressions.

Implicit functions are defined not as explicit expressions but rather by a potentially infinite set

[1]Columbia University, Department of Statistics
[2]Symplectomorphic, LLC

of equality constraints that an output must satisfy for a given input. Common examples include algebraic equations, optima, and differential equations. Although defined only implicitly by the constraints they must satisfy, these functions and their derivatives can be evaluated at a given input which puts them within the scope of automatic differentiation.

Many approaches to evaluating the derivatives of implicit functions have been developed, most designed for specific classes of implicit functions. Here we focus on two approaches: direct application of the implicit function theorem and adjoint methods. The former is commonly applied to finite-dimensional systems such as algebraic equations and optimization problems for example Bell and Burke 2008; Lorraine, Vicol, and Duvenaud 2019; Gaebler 2021 while the latter is particularly well-suited to infinite-dimensional systems such as ordinary differential equations for example Pontryagin et al. 1963; Errico 1997, algebraic differential equations (Cao et al. 2002), and stochastic differential equations (Li et al. 2020). Adjoint methods have also been derived for some finite-dimensional systems such as difference equations (Betancourt, Margossian, and Leos-Barajas 2020). When they can be derived the performance of adjoint methods often scales better than the performance of implicit function theorem methods; the details of those derivations, however, can change drastically from one system to another.

In this paper we derive implicit function theorem and adjoint methods that implement automatic differentiation for *any* implicit function regardless of its dimensionality. We begin by reviewing derivatives of real-valued functions – drawing a careful distinction between total, partial, and directional derivatives – and then introduce the basics of automatic differentiation. Next we derive implicit function theorem and adjoint methods for any finite-dimensional implicit function and demonstrate their application to the reverse mode automatic differentiation of general algebraic equations, the special case of difference equations, and optimization problems. Finally we generalize these methods to infinite-dimensional implicit functions with demonstrations on ordinary and algebraic differential equations. In each example we examine the particular challenges that arise with each method.

A.2 Automatic Differentiation

Before discussing the automatic differentiation of implicit functions, in this section we will review the basics of differentiating real-valued functions and then how derivatives of computer programs can be implemented algorithmically as automatic differentiation.

A.2.1 A Little Derivative

Differentiation is a pervasive topic, but terminology and notation can vary strongly from field to field. In this section we review the mathematics of derivatives on real spaces and introduce all of the terminology and notation that we will use throughout the paper. We first discuss the parameterization of real spaces and the vector space interpretation that emerges before introducing formal definitions for total and directional derivatives and their properties.

Locations and Directions.

An I-dimensional real space $\mathbb{R}^I$ models a rigid and smooth continuum of points. Here we will consider a subset of the real numbers $X \subseteq \mathbb{R}^I$ which may or may not be compact.

A *parameterization* of X decomposes the I-dimensional space into I copies of the one-dimensional real line,

$$X \approx \mathbb{R}_1 \times \ldots \times \mathbb{R}_i \times \ldots \times \mathbb{R}_I,$$

which we refer to as a *coordinate system*. Within a parameterization each point $x \in X$ can be identified by I real numbers denoted *parameters* or *coordinates*,

$$x = (x_1, \ldots, x_i, \ldots, x_I).$$

Every real space X admits an infinite number of parameterizations (Figure A.1). A one-to-one map from X into itself can be interpreted as a map from one parameterization to another and consequently is often denoted a *reparameterization* or *change of coordinate system*.

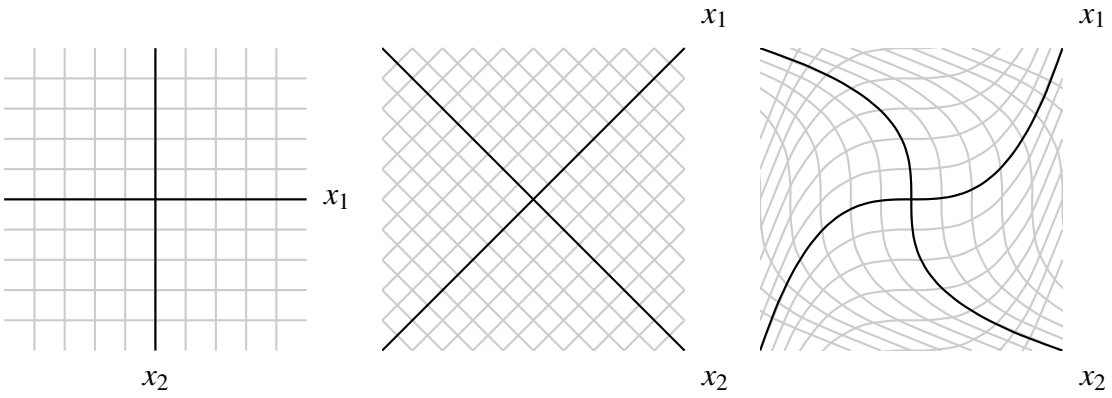

Figure A.1: Every real space X admits an infinite number of parameterizations, or coordinate systems, each of which are capable of uniquely identifying every point with an ordered tuple of real numbers.

The choice of any given parameterization endows X with a rich *geometry*. In particular we can use coordinates to define how to scale any point $x \in X$ by a real number $\alpha \in \mathbb{R}$,

$$\begin{aligned} \alpha \cdot x &= \alpha \cdot (x_1, \ldots, x_i, \ldots, x_I) \\ &= (\alpha \cdot x_1, \ldots, \alpha \cdot x_i, \ldots, \alpha \cdot x_I) \\ &= x' \in X, \end{aligned}$$

as well as add two points $x, x' \in X$ together,

$$\begin{aligned} x + x' &= (x_1, \ldots, x_i, \ldots, x_I) + (x'_1, \ldots, x'_i, \ldots, x'_I) \\ &= (x_1 + x'_1, \ldots, x_i + x'_i, \ldots, x_I + x'_I) \\ &= x'' \in X. \end{aligned}$$

These properties make the parameterization of X a *vector space over the real numbers*; each point $x \in X$ identifies a unique vector $\mathbf{x}$ and the coordinates define a distinguished vector space basis. If

we further use the coordinates to define an inner product,

$$\langle \mathbf{x}, \mathbf{x}' \rangle \equiv \sum_{i=1}^{I} (x_i - x'_i)^2$$

then this vector space becomes a *Euclidean vector space*, $E(X)$.

This Euclidean vector space structure defines a notion of *direction* and *orientation* in X. For example any point $x \in X$ can be interpreted as a vector $\mathbf{x}$ stretching from the origin at $(0, \ldots, 0, \ldots, 0) \equiv O$ to x. Similarly any two points $x, x' \in X$ are connected by the vector

$$\begin{aligned} \boldsymbol{\Delta}\mathbf{x} &= \mathbf{x}' - \mathbf{x} \\ &= (x'_1 - x_1, \ldots, x'_i - x_i, \ldots, x'_I - x_I). \end{aligned}$$

Equivalently we can think of vectors as a way to translate from one point to another (Figure A.2). For example $\mathbf{x}$ shifts the origin to x,

$$\begin{aligned} x &= (x_1, \ldots, x_i, \ldots, x_I) \\ &= (0 + x_1, \ldots, 0 + x_i, \ldots, 0 + x_I) \\ &= (0, \ldots, 0, \ldots, 0) + (x_1, \ldots, x_i, \ldots, x_I) \\ &= O + \mathbf{x}, \end{aligned}$$

while $\boldsymbol{\Delta}\mathbf{x}$ shifts from x to x',

$$x' = x + (x' - x) = x + (\mathbf{x}' - \mathbf{x}) = x + \boldsymbol{\Delta}\mathbf{x}.$$

Consequently once we fix a parameterization each set of coordinates $(x_1, \ldots, x_i, \ldots, x_I)$ can be interpreted as either a *location* $x \in X$ or a *direction* $\mathbf{x} \in E(X)$.

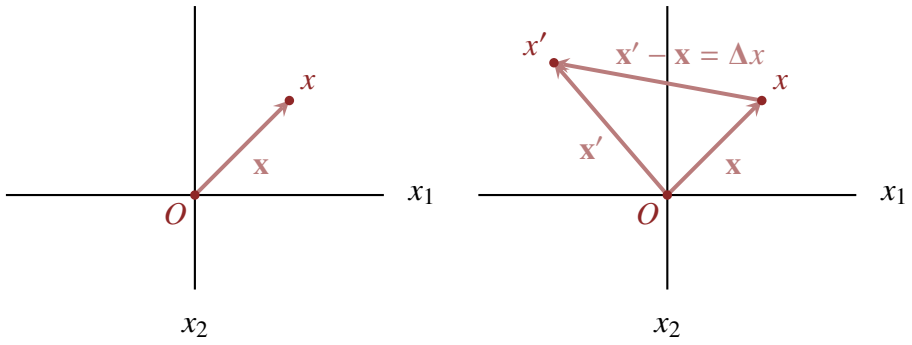

Figure A.2: Given a parameterization of the real space X vectors quantify the direction and distance between points, such as $\mathbf{x}$ between point x and the origin O or $\mathbf{x}' - \mathbf{x}$ between x and x'. By following a vector we can also translate from the initial point to the final point.

Transforming Locations and Directions.

Given two real spaces $X \subseteq \mathbb{R}^I$ and $Y \subseteq \mathbb{R}^J$ a *real-valued function* $f : X \rightarrow Y$ maps points in X to points in Y. Once a parameterization has been fixed for both spaces such a mapping also induces a map from vectors in $E(X)$ to vectors in $E(Y)$, $F : E(X) \rightarrow E(Y)$.

Induced maps that preserve the additive and multiplicative structure of the Euclidean vector space,

$$F(\alpha \cdot \mathbf{x} + \beta \cdot \mathbf{x}') = \alpha \cdot F(\mathbf{x}) + \beta \cdot F(\mathbf{x}'),$$

are said to be *linear*. Linear maps can be represented by a matrix of real numbers that maps the components of the input vector to the components of the output vector,

$$y_j = \sum_{i=1}^{I} F_{ji}\, x_i,$$

or in standard linear algebra notation,

$$\mathbf{y} = F(\mathbf{x}) = \mathbf{F} \cdot \mathbf{x}.$$

We will denote the space of linear maps between $E(X)$ and $E(Y)$ as $L(X,Y)$.

The Total Derivative.

The *total derivative* of a function $f : X \to Y$ quantifies the behavior of f in the local neighborhood of an input point $x \in X$. This local behavior provides a way of mapping vectors that represent infinitesimal translations from x to vectors that represent infinitesimal translations from $f(x)$.

More formally the total derivative assigns to each point $x \in X$ a linear transformation between $E(X)$ and $E(Y)$,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x} : X &\to L(X, Y) \\ x &\mapsto \frac{\mathrm{d}f}{\mathrm{d}x}(x) \equiv J(x), \end{aligned}$$

that quantifies the first-order variation of f in the neighborhood of each input x. We can then say that *the total derivative of f at x* is the linear transformation $J(x) : E(X) \to E(Y)$.

In components the total derivative at a point $x \in X$ is specified by a matrix of partial derivative functions evaluated at x,

$$J_{ji}(x) = \frac{\partial f_j}{\partial x_i}(x),$$

denoted the *Jacobian*. The action of the total derivative at x on a vector $\mathbf{v} \in E(X)$ is then given by matrix multiplication,

$$J(x)(\mathbf{v}) = \mathbf{J}(x) \cdot \mathbf{v} = \sum_{i=1}^{I} J_{ji}(x)\, v_i.$$

Directional Derivatives.

The total derivative of a function f at a point x applied to a vector $\mathbf{v} \in E(X)$, or more compactly $J(x)(\mathbf{v})$, quantifies how much the output of f varies as x is translated infinitesimally in the direction of v. Consequently $J(x)(\mathbf{v}) = \mathbf{J}(x) \cdot \mathbf{v}$ is called the *forward directional derivative*.

The forward directional derivative is often used to construct the linear function between X and Y that best approximates f at x. The best approximation evaluated at x' is given by translating

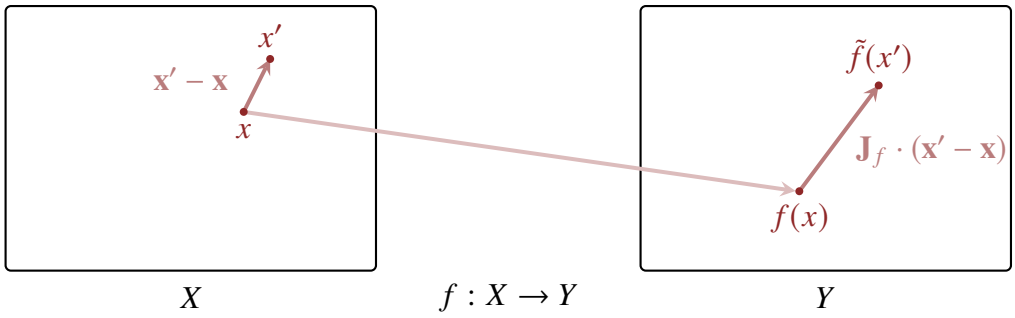

Figure A.3: The forward directional derivative of the function f at $x \in X$, $\mathbf{J}_f$, propagates infinitesimal perturbations to the input, $\mathbf{x}' - \mathbf{x}$, to infinitesimal perturbations of the output, $\mathbf{J}_f \cdot (\mathbf{x}' - \mathbf{x})$. Translating the function output $f(x)$ by $\mathbf{J}_f \cdot (\mathbf{x}' - \mathbf{x})$ generates the best linear approximation to f at x, $\tilde{f}$.

$f(x)$ along $J(x)(\mathbf{x}' - \mathbf{x})$ (Figure A.3)

$$\begin{aligned}\tilde{f}(x') &= f(x) + J(x)(\mathbf{x}' - \mathbf{x}) \\ &= f(x) + \mathbf{J}(x) \cdot (\mathbf{x}' - \mathbf{x}),\end{aligned}$$

or in components,

$$\tilde{f}_j(x'_1, \ldots, x'_I) = f_j(x_1, \ldots, x_I) + \sum_{i=1}^{I} \frac{\partial f_j}{\partial x_i}(x_1, \ldots, x_I) \cdot (x'_i - x_i).$$

The total derivative also defines an *adjoint transformation* that maps vectors in $E(Y)$ to vectors in $E(X)$, $J^\dagger(x) : E(Y) \to E(X)$. The matrix components of this adjoint transformation are given by the transpose of the Jacobian matrix,

$$J^\dagger_{ji}(x) = J_{ij}(x).$$

The application of this adjoint transformation to a vector $\boldsymbol{\alpha} \in E(Y)$,

$$\boldsymbol{\beta} = J^\dagger(x)(\boldsymbol{\alpha}) = \mathbf{J}^T \cdot \boldsymbol{\alpha},$$

quantifies how the inputs need to vary around x in order to achieve the given output variation $\boldsymbol{\alpha}$.

We will refer to this action as a *reverse directional derivative*.

The Chain Rule.

The chain rule provides an explicit construction for the total derivative of a composite function constructed from many component functions. Consider for example a sequence of functions $f_n : X_n \to X_{n+1}$ and the composite function

$$f = f_N \circ \ldots \circ f_n \circ \ldots \circ f_1 : X_1 \to X_{N+1}.$$

The total derivative of f at any point $x_1 \in X_1$ is given by composing the total derivatives of each component function together in the same order,

$$J_f = J_{f_N}(x_N) \circ \ldots \circ J_{f_n}(x_n) \circ \ldots \circ J_{f_1}(x_1),$$

where $x_n = f_{n-1}(x_{n-1})$. Likewise the components of the composite Jacobian matrix are given by a sequence of matrix products,

$$\mathbf{J}_f = \mathbf{J}_{f_N}(x_N) \cdot \ldots \cdot \mathbf{J}_{f_n}(x_n) \cdot \ldots \cdot \mathbf{J}_{f_1}(x_1).$$

Unfortunately these intermediate matrix products are expensive to evaluate, especially when the intermediate spaces X_n are high-dimensional. Constructing the full composite Jacobian matrix is usually computationally burdensome.

On the other hand this composite structure is well-suited to the evaluation of directional derivatives. For example the forward directional derivative of a composite function is given by

$$J_f(\mathbf{v}) = \big(J_{f_N}(x_N) \circ \ldots \circ J_{f_n}(x_n) \circ \ldots \circ J_{f_1}(x_1)\big)(\mathbf{v}),$$

or

$$\mathbf{J}_f \cdot \mathbf{v} = \big(\mathbf{J}_{f_N}(x_N) \cdot \ldots \cdot \mathbf{J}_{f_n}(x_n) \cdot \ldots \cdot \mathbf{J}_{f_1}(x_1)\big) \cdot \mathbf{v}.$$

Because of the associativity of matrix multiplication we can apply each component derivative to $\mathbf{v}$ in sequence and avoid the intermediate compositions entirely,

$$J_f(\mathbf{v}) = J_{f_N}(x_N)(\cdots J_{f_n}(x_n)(\cdots (J_{f_1}(x_1)(\mathbf{v}))\cdots)\cdots),$$

or

$$\mathbf{J}_f \cdot \mathbf{v} = \mathbf{J}_{f_N}(x_N) \cdot (\cdots \mathbf{J}_{f_n}(x_n) \cdot (\cdots \mathbf{J}_{f_1}(x_1) \cdot \mathbf{v})\cdots)\cdots).$$

In other words we can evaluate the total forward directional derivative iteratively,

$$\begin{aligned}
\mathbf{v}_1 &= J_{f_1}(x_1)(\mathbf{v}) \\
\mathbf{v}_2 &= J_{f_2}(x_2)(\mathbf{v}_1) \\
&\cdots \\
\mathbf{v}_n &= J_{f_n}(x_n)(\mathbf{v}_{n-1}) \\
&\cdots \\
\mathbf{v}_{N-1} &= J_{f_{N-1}}(x_{N-1})(\mathbf{v}_{N-2}) \\
J_f(x)(\mathbf{v}) = \mathbf{v}_N &= J_{f_N}(x_N)(\mathbf{v}_{N-1}),
\end{aligned}$$

or in components,

$$\begin{aligned}
\mathbf{v}_1 &= \mathbf{J}_{f_1}(x_1) \cdot \mathbf{v} \\
\mathbf{v}_2 &= \mathbf{J}_{f_2}(x_2) \cdot \mathbf{v}_1 \\
&\cdots \\
\mathbf{v}_n &= \mathbf{J}_{f_n}(x_n) \cdot \mathbf{v}_{n-1} \\
&\cdots \\
\mathbf{v}_{N-1} &= \mathbf{J}_{f_{N-1}}(x_{N-1}) \cdot \mathbf{v}_{N-2} \\
\mathbf{J}_f(x) \cdot \mathbf{v} = \mathbf{v}_N &= \mathbf{J}_{f_N}(x_N) \cdot \mathbf{v}_{N-1}.
\end{aligned}$$

The evaluation of each intermediate directional derivative requires only a matrix-vector product which is substantially less expensive to implement than the matrix-matrix products needed to evaluate the composite Jacobian.

The reverse directional derivative can be evaluated sequentially as well,

$$\begin{aligned}
\boldsymbol{\alpha}_N &= J^\dagger_{f_N}(x_N)(\boldsymbol{\alpha}) \\
\boldsymbol{\alpha}_{N-1} &= J^\dagger_{f_{N-1}}(x_{N-1})(\boldsymbol{\alpha}_N) \\
&\cdots \\
\boldsymbol{\alpha}_{N-n} &= J^\dagger_{f_{N-n}}(x_{N-n})(\boldsymbol{\alpha}_{N-n+1}) \\
&\cdots \\
\boldsymbol{\alpha}_2 &= J^\dagger_{f_2}(x_2)(\boldsymbol{\alpha}_3) \\
J^\dagger_f(x)(\boldsymbol{\alpha}) = \boldsymbol{\alpha}_1 &= J^\dagger_{f_1}(x_1)(\boldsymbol{\alpha}_2),
\end{aligned}$$

or in components,

$$\begin{aligned}
\boldsymbol{\alpha}_N &= \mathbf{J}^T_{f_N}(x_N) \cdot \boldsymbol{\alpha} \\
\boldsymbol{\alpha}_{N-1} &= \mathbf{J}^T_{f_{N-1}}(x_{N-1}) \cdot \boldsymbol{\alpha}_N \\
&\cdots \\
\boldsymbol{\alpha}_{N-n} &= \mathbf{J}^T_{f_{N-n}}(x_{N-n}) \cdot \boldsymbol{\alpha}_{N-n+1} \\
&\cdots \\
\boldsymbol{\alpha}_2 &= \mathbf{J}^T_{f_2}(x_2) \cdot \boldsymbol{\alpha}_3 \\
\mathbf{J}^T_f(x) \cdot \boldsymbol{\alpha} = \boldsymbol{\alpha}_1 &= \mathbf{J}^T_{f_1}(x_1) \cdot \boldsymbol{\alpha}_2.
\end{aligned}$$

A.2.2 Express Yourself

Before we can consider how to implement derivatives of real-valued functions in practice we first have to consider how to implement the real-valued functions themselves. Functions $f : X \to Y$

Pseudo-Code

```
real f(vector[3] x1) {
  real z1 = x1[1] + x1[2];
  real z2 = x1[2] * x1[3];
  return z1 / z2;
}
```

Expression Graph

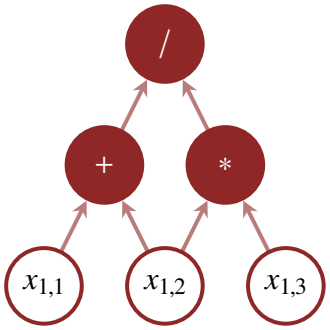

Figure A.4: This pseudo-code implements a function $f : \mathbb{R}^3 \to \mathbb{R}$ that maps every input $x_1 = (x_{1,1}, x_{1,2}, x_{1,3})$ to a real-valued output through three intermediate expressions. The dependencies between these expressions and the input variables forms an expression graph.

are often implemented as computer programs which take in any input value $x \in X$ and return the corresponding output value $f(x) \in Y$. These computer programs are themselves implemented as a sequence of *expressions*, each of which transforms some descendent of the initial value towards the final output value.

If these expressions defined full component functions then we could use the chain rule to automatically propagate directional derivatives through the program as we discussed in the previous section. Well-defined component functions, however, would depend on only the output of the previous component function, while expressions can depend on the output of multiple previous expressions. Because of this expressions do not immediately define valid component functions on their own, and the chain rule does not immediately apply.

That said we can manipulate each expression into a well-defined component function with a little bit of work. First we'll need to take advantage of the fact that the expressions that comprise a complete program can be represented as a directed acyclic graph, also known as an *expression graph* (Figure A.4). In each expression graph the root nodes designate the input variables, internal and leaf nodes designate the expressions, and edges designate the dependencies between the expressions.

A *topological sort* of an expression graph is any ordering of the nodes such that each expression

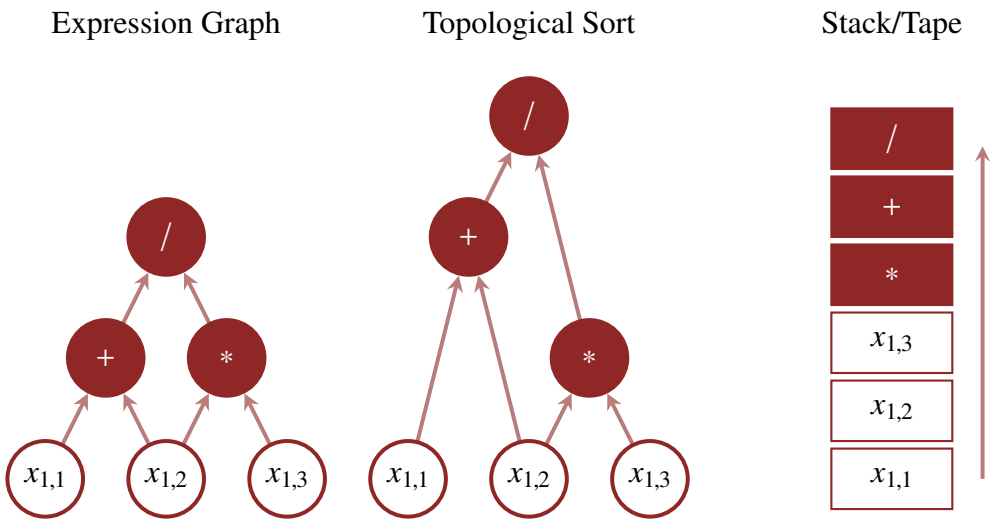

Figure A.5: A topological sort of an expression graph is an ordering of the expressions, often called a *stack* or a *tape*, that guarantees that when progressing across the stack in order each expression will not be evaluated until all of the expressions on which it depends have already been evaluated.

follows all expressions on which it depends (Figure A.5). Such a sorting ensures that if we process the sorted expressions in order then we will evaluate an expression only once all of the expressions on which it depends have already been evaluated.

Any topological sort provides an explicit sequence of expressions, but each expression can still depend on the output of any expression that precedes it in the stack. We can use the ordering to limit this dependence to only the previous output, however, if we can buffer each expression with any of the previous outputs that are used by future expressions. For example this buffering can be implemented by introducing identify expressions that propagate any necessary values forward (Figure A.6). Together each initial expression and the added identify expressions define a *layer* of expressions that depend on only the expressions in the previous layer, so that these layers define a sequence of valid component functions (Figure A.7).

Once we've derived these component functions we can finally apply the chain rule. In particular we can evaluate forward directional derivatives by propagating an initial vector through the constructed sequence of component functions. At the same time we can evaluate a reverse directional derivative by first evaluating all of the component functions in a forward sweep through the

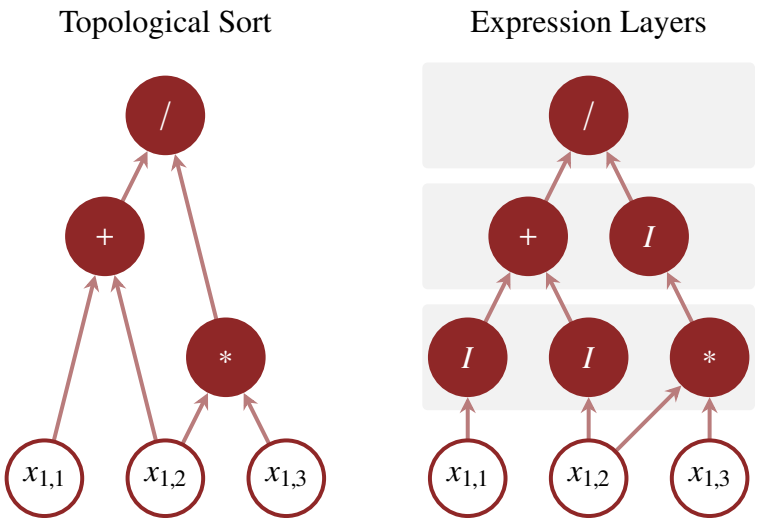

Figure A.6: In order to turn each topologically-sorted expression into a valid component function they must be complemented with identity maps that carry forward intermediate values needed by future expressions. The resulting layers of expressions depend on only expressions in the previous layer.

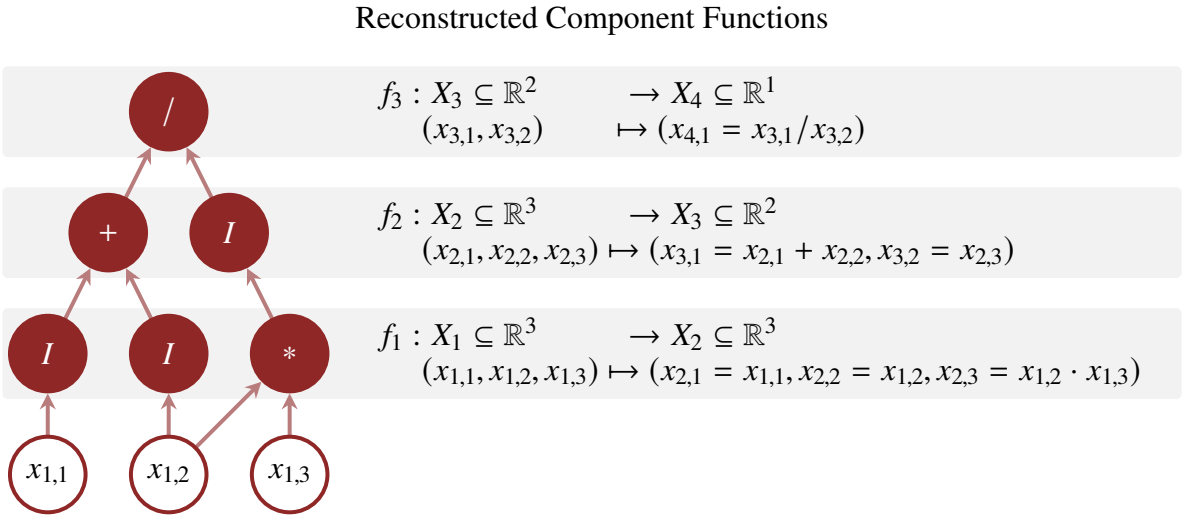

Figure A.7: Complementing each topologically-sorted expression with the appropriate identity maps defines valid component functions. When composed together these component functions yield the function implemented by the computer program, here $f = f_3 \circ f_2 \circ f_1$, and allow for the application of the chain rule to differentiate through the program.

sequence before executing a reverse sweep that propagates a vector in the output space to the input space.

Something interesting happens, however, when we evaluate the total derivative of one of these reconstructed component functions. Let's denote the action of the component function as $f_n : (x, x') \mapsto (g(x), I(x'))$, where g is the action implemented by the initial expression and I is the identify map that propagates any auxiliary outputs. We can also decompose the input vector $\mathbf{v}$ into components that map onto g and I, respectively,

$$\mathbf{v} = (\mathbf{v}_g, \mathbf{v}_I)^T.$$

In this notation the forward directional derivative becomes

$$\begin{aligned}
J_{f_n}(\mathbf{v}) &= J_{f_n}((\mathbf{v}_g, \mathbf{v}_I)^T) \\
&= \mathbf{J}_{f_n} \cdot (\mathbf{v}_g, \mathbf{v}_I)^T \\
&= \begin{pmatrix} \frac{\partial g}{\partial x} & \frac{\partial g}{\partial x'} \\ \\ \frac{\partial I}{\partial x} & \frac{\partial I}{\partial x'} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{v}_g \\ \\ \mathbf{v}_I \end{pmatrix} \\
&= \begin{pmatrix} \frac{\partial g}{\partial x} & 0 \\ \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{v}_g \\ \\ \mathbf{v}_I \end{pmatrix} \\
&= \begin{pmatrix} \frac{\partial g}{\partial x} \cdot \mathbf{v}_g \\ \\ 0 \end{pmatrix}
\end{aligned}$$

The only non-vanishing contribution to the directional derivative comes from the initial expression; the vector corresponding to the buffered outputs, $\mathbf{v}_I$, completely decouples from any derivatives that follow. Consequently the only aspect of the reconstructed component function that

influences propagation of the forward directional derivative is the forward directional derivative of the expression itself.

In other words we can evaluate the forward directional derivative of a function implemented by a computer program by propagating only the forward directional derivatives of the individual expressions; at no point do we actually have to construct explicit component functions! The action of the adjoint derivative behaves similarly, allowing us to evaluate the reverse directional derivative using only the reverse directional derivatives local to each expression. When considering higher-order derivatives, however, the equivalence between propagating vectors across expressions and full component functions is not always preserved and explicit component functions may be needed. For more see Betancourt 2018b.

A.2.3 Automatic Differentiation

Automatic differentiation exploits the equivalence between propagating vectors across expressions and full component functions to algorithmically evaluate forward and reverse directional derivatives.

Forward mode automatic differentiation implements forward directional derivatives, passing intermediate values and intermediate forward directional derivatives between expressions as the program is evaluated. These intermediate forward directional derivatives are denoted *tangents* or *sensitivities*.

Any implementation of a function g that supports forward mode automatic differentiation must provide not only a map from input values to output values but also a map from input tangents to output tangents,

$$\mathbf{v}' = J_g \cdot \mathbf{v}.$$

Similarly *reverse mode automatic differentiation* implements reverse directional derivatives. Here the program must be evaluated first before intermediate reverse directional derivatives are propagated between expressions in the *reverse* order that the expressions are evaluated. These intermediate reverse directional derivatives are denoted *cotangents* or *adjoints*.

Any implementation of a function g that supports reverse mode automatic differentiation must provide not only a map from input values to output values but also a map from output cotangents to input cotangents,

$$\boldsymbol{\alpha}' = J_g^T \cdot \boldsymbol{\alpha}.$$

A.3 Automatic Differentiation of Finite-Dimensional Implicit Functions

A finite collection of constraint functions that an output has to satisfy for a given input implicitly defines a map from inputs to outputs, or an implicit function. In this section we discuss how implicit functions are formally defined, how to differentiate these implicitly defined functions, and then finally demonstrate their application on several instructive examples.

A.3.1 The Finite-Dimensional Implicit Function Theorem

Consider a finite-dimensional real-valued space of known inputs, X, a finite-dimensional real-valued space of unknown outputs, Y, a finite-dimensional real-valued space of constraint values, Z, and the *constraint function*

$$\begin{aligned} c : X \times Y &\to Z \\ (x, y) &\mapsto c(x, y). \end{aligned}$$

The implicit function theorem defines the conditions under which the constraint function implicitly defines a map from $f : X \to Y$ which satisfies $c(x, f(x)) = 0$ for all inputs in a local neighborhood $x \in U \subset X$.

More formally consider the neighborhoods $0 \in W \subset Z$ and $V \subset Y$ such that the kernel of the constraint function falls into product of U and V,

$$c^{-1}(0) = U \times V,$$

and assume that a function $f : U \to V$ that satisfies $c(x, f(x)) = 0$ exists. If the constraint function c is differentiable across $U \times V$ then the total derivative of c evaluated at $(x, f(x))$ is given by

$$\begin{aligned} 0 &= \frac{\mathrm{d}}{\mathrm{d}x} c(x, f(x)) \\ &= \frac{\partial c}{\partial x}(x, f(x)) + \frac{\partial c}{\partial y}(x, f(x)) \circ \frac{\mathrm{d}f}{\mathrm{d}x}(x). \end{aligned}$$

When the dimension of Z equals the dimension of Y then the partial derivative $\partial c / \partial y \ (x, y)$ might define a bijection from V to W. If it does then we can solve for the total derivative of the assumed function,

$$\underbrace{\frac{\mathrm{d}f}{\mathrm{d}x}(x)}_{U \to V} = - \underbrace{\left(\frac{\partial c}{\partial y}(x, f(x)) \right)^{-1}}_{W \to V} \circ \underbrace{\frac{\partial c}{\partial x}(x, f(x))}_{U \to W}.$$

When $\partial c / \partial y \ (x, y)$ is bijective this derivative is well-defined, and the existence and uniqueness of ordinary differential equation solutions guarantees that an implicit function that satisfies $c(x, f(x)) = 0$ is well-defined in the neighborhood around x. In other words the system of constraints defines an implicit function if and only if the partial derivative $\partial c / \partial y \ (x, y)$ is invertible

The implicit function theorem determines when an implicit function is well-defined, but not how to evaluate it. In practice we typically have to rely on numerical methods that heuristically search the output space for a value y that satisfies $c(x, y) = 0$ for the given input x.

A.3.2 Evaluating Directional Derivatives of Finite-Dimensional Implicit Functions

To incorporate implicit functions into an automatic differentiation library we need to be able to evaluate not only the output consistent with a given input but also the directional derivatives. Here we consider three general approaches: a *trace method* that works with a given numerical solver, a method that utilizes intermediate results of the implicit function theorem, and an adjoint method that evaluates the directional derivative directly.

In all three approaches we will consider not the implicit function alone but rather its composition with a summary function that maps the outputs into some real space, $g : Y \to \mathbb{R}^K$. Often g

will be the identify map, but the flexibility offered by this summary function will facilitate some of the examples that we consider below.

When $X = \mathbb{R}^I$ and $Y = Z = \mathbb{R}^J$ the composition of the summary function with the implicit function defines the real-valued function

$$h = g \circ f : \mathbb{R}^I \to \mathbb{R}^J \to \mathbb{R}^K$$

and our goal will be to evaluate either the forward directional derivative $J_{g \circ f}(x)(\mathbf{v})$ or the reverse directional derivative $J^{\dagger}_{g \circ f}(x)(\boldsymbol{\alpha})$.

Trace Method.

Each step that an iterative numerical solver takes while searching for a consistent output can be interpreted as a map from the output space to itself given the fixed input x,

$$\begin{aligned} \tilde{f}_n : X \times Y &\to Y \\ (x, y_{n-1}) &\mapsto y_n. \end{aligned}$$

The *trace* of the solver's evaluation then defines a composite function,

$$\tilde{f}(x, y_0) = \left(\tilde{f}_N(x) \circ \tilde{f}_{N-1}(x) \circ \ldots \circ f_1(x)\right)(y_0)$$

that maps the input and an initial guess to an approximate solution,

$$\begin{aligned} \tilde{f} : X \times Y &\to Y \\ (x, y_0) &\mapsto \tilde{y} \end{aligned}$$

satisfying $c(x, \tilde{y}) \approx 0$.

If each of these intermediate steps are differentiable and supported by an automatic differenti-

ation library then we can evaluate the directional derivatives of $\tilde{f}$ using automatic differentiation and use them to approximate the directional derivatives of the exact implicit function f. While straightforward to implement this approach can suffer from poor performance in practice, especially as the number of solver iterations grows and propagating derivatives through the composite function becomes slow and memory intensive. See for example (Bell and Burke 2008) for further discussion of this trace method applied to optimization problems and (Margossian 2019) for one on algebraic equations.

Finite-Dimensional Implicit Function Theorem.

Conveniently the derivative of the implicit function f is explicitly constructed in the derivation of the implicit function theorem,

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x) = -\left(\frac{\partial c}{\partial y}(x, f(x))\right)^{-1} \circ \frac{\partial c}{\partial x}(x, f(x)).$$

If we can evaluate these derivatives of the constraint function then we can immediately evaluate the derivative for f once we have numerically solved for the output of the implicit function $y = f(x)$,

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x) = -\left(\frac{\partial c}{\partial y}(x, y)\right)^{-1} \circ \frac{\partial c}{\partial x}(x, y).$$

The Jacobian of the composite function $g \circ f$ is then given by

$$\begin{aligned} J_{g\circ f}(x) &= \frac{\partial (g \circ f)}{\partial x}(x) \\ &= \frac{\mathrm{d}g}{\mathrm{d}y}(f(x)) \circ \frac{\mathrm{d}f}{\mathrm{d}x}(x) \\ &= -\frac{\mathrm{d}g}{\mathrm{d}y}(f(x)) \circ \left(\frac{\partial c}{\partial y}(x, f(x))\right)^{-1} \circ \frac{\partial c}{\partial x_i}(x, f(x)). \end{aligned}$$

In components this becomes

$$\begin{aligned}(J_{g\circ f})_{ik}(x) &= \frac{\partial h_k}{\partial x_i}(x) \\ &= \sum_{j=1}^{J} \frac{\mathrm{d}g_k}{\mathrm{d}y_j}(f(x)) \cdot \frac{\mathrm{d}f_j}{\mathrm{d}x_i}(x) \\ &= -\sum_{j=1}^{J}\sum_{j'=1}^{J} \frac{\mathrm{d}g_k}{\mathrm{d}y_j}(f(x)) \circ \left(\frac{\partial c_{j'}}{\partial y_j}(x, f(x))\right)^{-1} \circ \frac{\partial c_{j'}}{\partial x_i}(x, f(x)),\end{aligned}$$

or in more compact matrix notation,

$$\mathbf{J}_{g\circ f} = -\mathbf{J}_g \cdot \mathbf{C}_y^{-1} \cdot \mathbf{C}_x,$$

where

$$\begin{aligned}(\mathbf{C}_y)_{ij} &= \frac{\partial c_i}{\partial y_j}(x, f(x)) \\ (\mathbf{C}_x)_{ij} &= \frac{\partial c_i}{\partial x_j}(x, f(x)) \\ (\mathbf{J}_g)_{ij} &= \frac{\partial g_j}{\partial y_i}(f(x)).\end{aligned}$$

In order to incorporate this composite function into forward mode automatic differentiation we need to evaluate the action of the Jacobian contracted against a tangent vector,

$$\begin{aligned}\left(\mathbf{J}_{g\circ f}(x) \cdot \mathbf{v}\right)_j &= \sum_{i=1}^{I} (J_{g\circ f})_{ij}(x) \cdot v_i \\ &= -\sum_{i=1}^{I} \frac{\mathrm{d}g_j}{\mathrm{d}y}(f(x)) \circ \left(\frac{\partial c}{\partial y}(f(x))\right)^{-1} \circ \frac{\partial c}{\partial x_i}(x) \cdot v_i,\end{aligned}$$

or equivalently

$$\mathbf{J}_{g\circ f}(x) \cdot \mathbf{v} = -\mathbf{J}_g \cdot \mathbf{C}_y^{-1} \cdot \mathbf{C}_x \cdot \mathbf{v}.$$

Similarly to incorporate this composite function into reverse mode automatic differentiation we

need to evaluate the action of the Jacobian contracted against a cotangent vector,

$$\begin{aligned}(\mathbf{J}_{g\circ f}^T(x)\cdot\boldsymbol{\alpha})_i &= \sum_{j=1}^{J}(J_{g\circ f})_{ij}(x)\cdot\alpha_j \\ &= -\sum_{j=1}^{J}\frac{\mathrm{d}g_j}{\mathrm{d}y}(f(x))\circ\left(\frac{\partial c}{\partial y}(f(x))\right)^{-1}\circ\frac{\partial c}{\partial x}(f(x))\cdot\alpha_j,\end{aligned}$$

or

$$\begin{aligned}\mathbf{J}_{g\circ f}^T(x)\cdot\boldsymbol{\alpha} &= -\mathbf{C}_x^T\cdot\left(\mathbf{C}_y^{-1}\right)^T\cdot\mathbf{J}_g^T\cdot\boldsymbol{\alpha} \\ &= -\mathbf{C}_x^T\cdot\left(\mathbf{C}_y^T\right)^{-1}\cdot\mathbf{J}_g^T\cdot\boldsymbol{\alpha}.\end{aligned}$$

With so many terms there are multiple ways to evaluate these directional derivatives. The *forward* method, for example, explicitly constructs $\mathbf{J}_{g\circ f}(x) = -\mathbf{J}_g\cdot\mathbf{C}_y^{-1}\cdot\mathbf{C}_x$ before evaluating the contractions $\mathbf{J}_{g\circ f}(x)\cdot\mathbf{v}$ or $\mathbf{J}_{g\circ f}^T(x)\cdot\boldsymbol{\alpha}$. With careful use of automatic differentiation, however, we can evaluate the final directional derivatives more efficiently. In particular we don't need to explicitly construct $\mathbf{J}_g$ and $\mathbf{C}_x$ at all.

For example when evaluating $\mathbf{J}_{g\circ f}(x)\cdot\mathbf{v}$ we can avoid $\mathbf{C}_x$ by using one sweep of forward mode automatic differentiation to evaluate $\mathbf{u} = \mathbf{C}_x\cdot\mathbf{v}$ directly. Because of the inversion we have to construct the entirety of $\mathbf{C}_y$, for example with J sweeps of forward mode or reverse mode automatic differentiation, but we can avoid constructing $(\mathbf{C}_y)^{-1}$ by solving for only the linear system

$$\mathbf{C}_y\cdot\mathbf{t} = \mathbf{u}.$$

Finally we can avoid constructing $\mathbf{J}_g$ with one sweep of forward mode automatic differentiation to evaluate $\mathbf{J}_g\cdot\mathbf{t}$. These steps are outlined in Algorithm A.1.

Similar optimizations are also possible when evaluating the reverse directional derivative $\mathbf{J}_{g\circ f}^T(x)\cdot\boldsymbol{\alpha}$. We first evaluate $\boldsymbol{\beta} = \mathbf{J}_g^T\cdot\boldsymbol{\alpha}$ using one sweep of reverse mode automatic differentiation and then

Algorithm A.1: Forward mode automatic differentiation of finite-dimensional implicit function.

1 **input:** constraint function, c; summary function, g; tangent, $\mathbf{v}$. ▷ *Note: we assume we have a differentiable program for all the input functions.*
2 $\mathbf{u} \leftarrow \mathbf{C}_x \cdot \mathbf{v}$ ▷ one forward mode sweep
3 $\mathbf{C}_y \leftarrow \partial c / \partial y$ ▷ J forward or reverse mode sweep
4 $\mathbf{t} \leftarrow \mathbf{C}_y^{-1} \cdot \mathbf{u}$ ▷ linear solve
5 $\mathbf{J}_{g \circ f} \cdot \mathbf{v} \leftarrow -\mathbf{J}_g \cdot \mathbf{t}$ ▷ one forward sweep
6 **return:** $\mathbf{J}_{g \circ f} \cdot \mathbf{v}$

construct $\mathbf{C}_y$ as above. This allows us to solve the linear system

$$\mathbf{C}_y^T \cdot \boldsymbol{\gamma} = \boldsymbol{\beta},$$

and then evaluate $\mathbf{C}_x^T \cdot \boldsymbol{\beta}$ with one more sweep of reverse mode automatic differentiation through the constraint function. These steps are outlined in Algorithm A.2.

Algorithm A.2: Reverse mode automatic differentiation of finite-dimensional implicit function.

1 **input:** constraint function, c; summary function, g; cotangent, $\boldsymbol{\alpha}$. ▷ *Note: we assume we have a differentiable program for all the input functions.*
2 $\boldsymbol{\beta} \leftarrow \mathbf{J}_g^T \cdot \boldsymbol{\alpha}$ ▷ one reverse mode sweep
3 $\mathbf{C}_y \leftarrow \partial c / \partial y$ ▷ J forward or reverse mode sweep
4 $\boldsymbol{\gamma} \leftarrow \left(\mathbf{C}_y^T\right)^{-1} \boldsymbol{\beta}$ ▷ linear solve
5 $\mathbf{J}_{g \circ f}^T \cdot \boldsymbol{\alpha} \leftarrow -\mathbf{C}_x^T \cdot \boldsymbol{\gamma}$ ▷ one reverse mode sweep
6 **return:** $\mathbf{J}_{g \circ f}^T \cdot \boldsymbol{\alpha}$

Often the form of a particular constraint function results in sparsity structure in $\mathbf{C}_x$ and $\mathbf{C}_y$ that can be exploited to reduce the cost of the irreducible linear algebraic operations. Identifying this structure and implementing faster operations, however, requires substantial experience with numerical linear algebra.

The Finite-Dimensional Adjoint Method.

The adjoint method provides a way of directly implementing the contractions that give forward and reverse directional derivatives without explicitly constructing the Jacobian $\mathrm{d}f/\mathrm{d}x(x)$. Adjoint methods have historically been constructed for specific implicit functions but here we present a general construction for any finite-dimensional system of constraints.

We begin by constructing a binary *Lagrangian* function,

$$\mathcal{L} : X \times Y \to \mathbb{R},$$

whose derivative is equal to the desired contraction when $c(x, y) = 0$. More formally we compose $\mathcal{L}$ with the implicit function defined by the constraints to give the unary function

$$\mathcal{L}(x) = \mathcal{L}(x, f(x)) : x \mapsto \mathbb{R},$$

and then require that

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x_i}(x) = (\mathbf{J}_{g \circ f} \cdot \mathbf{v})_i(x).$$

for forward mode automatic differentiation or

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x_i}(x) = (\mathbf{J}^T_{g \circ f} \cdot \boldsymbol{\alpha})_i(x)$$

for reverse mode automatic differentiation. To simplify the presentation from here on we will focus on only this latter application to reverse mode automatic differentiation.

Next we introduce *any* mapping $\Lambda : Z \to \mathbb{R}$ that preserves the kernel of the constraint function, $\Lambda \circ c(x, y) = 0$ whenever $c(x, y) = 0$. This allows us to define a second function

$$Q = \Lambda \circ c : X \times Y \to \mathbb{R}$$

with $Q(x, f(x)) = 0$ for all $x \in X$.

These two functions together then define an *augmented Lagrangian* function,

$$\mathcal{J} = \mathcal{L} + Q : X \times Y \to \mathbb{R}.$$

Substituting $y = f(x)$ gives a unary function,

$$\begin{aligned}\mathcal{J}(x) &= \mathcal{J}(x, f(x)) \\ &= \mathcal{L}(x, f(x)) + Q(x, f(x)).\end{aligned}$$

The second term, however, vanishes by construction so that $\mathcal{J}(x)$ reduces to $\mathcal{L}(x)$ and

$$\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}x_i}(x) = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x_i}(x) = (\mathbf{J}_{g \circ f}^T \cdot \boldsymbol{\alpha})_i(x).$$

While the contribution from $C(x, f(x))$ vanishes, its inclusion into the augmented Lagrangian introduces another way to evaluate the desired directional derivative. The total derivative of the unary augmented Lagrangian is

$$\begin{aligned}\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}x_i}(x) &= \frac{\mathrm{d}\mathcal{J}}{\mathrm{d}x_i}(x, f(x)) \\ &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x_i}(x, f(x)) + \frac{\mathrm{d}Q}{\mathrm{d}x_i}(x, f(x)) \\ &= \quad \frac{\partial \mathcal{L}}{\partial x_i}(x, f(x)) + \left(\frac{\partial \mathcal{L}}{\partial y}(x, f(x)) \circ \frac{\partial f}{\partial x_i}\right)(x) \\ &\quad + \frac{\partial Q}{\partial x_i}(x, f(x)) + \left(\frac{\partial Q}{\partial y}(x, f(x)) \circ \frac{\partial f}{\partial x_i}\right)(x) \\ &= \quad \frac{\partial \mathcal{L}}{\partial x_i}(x, f(x)) + \frac{\partial Q}{\partial x_i}(x, f(x)) \\ &\quad + \left(\left(\frac{\partial \mathcal{L}}{\partial y}(x, f(x)) + \frac{\partial Q}{\partial y}(x, f(x))\right) \circ \frac{\partial f}{\partial x_i}\right)(x).\end{aligned}$$

Once we've identified the solution $y = f(x)$ the first two terms are ordinary partial derivatives that are straightforward to evaluate. The second two terms, however, are tainted by the total derivative of the implicit function, $\mathrm{d}f/\mathrm{d}x$, which is expensive to evaluate as we saw in the previous section.

At this point, however, we can exploit the freedom in the choice of kernel-preserving function Λ and hence $\mathcal{J}$ itself. If we can engineer a Λ such that

$$\frac{\partial \mathcal{L}}{\partial y}(x, f(x)) + \frac{\partial Q}{\partial y}(x, f(x)) = 0$$

then the contribution from the last two terms, and the explicit dependence on the derivative of the implicit function vanishes entirely!

The adjoint method attempts to solve this *adjoint system*

$$\frac{\partial \mathcal{L}}{\partial y}(x, f(x)) + \frac{\partial \Lambda}{\partial c}(c(x, f(x))) \circ \frac{\partial c}{\partial y}(x, f(x)) = 0$$

for a suitable Λ and then evaluate the desired contraction from the remaining two terms,

$$\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}x_i}(x) = \frac{\partial \mathcal{L}}{\partial x_i}(x, f(x)) + \frac{\partial \Lambda}{\partial c}(c(x, f(x))) \circ \frac{\partial c}{\partial x_i}(x, f(x)).$$

If a suitable Λ exists, and we can find it, then this two-stage method allows us to evaluate the directional derivative directly without constructing the derivatives of the implicit function.

A.3.3 Demonstrations

To compare and contrast the two presented methods for evaluating directional derivatives of an implicit function we examine their application on three common implicit systems: algebraic equations, difference equations, and optimization problems.

Algebraic systems.

When $X = \mathbb{R}^I$ and $Y = \mathbb{R}^J$ a system of J transverse constraint functions $c_j(x, y)$ defines an algebraic system and a well-defined implicit function. In Section A.3.2 we saw that the finite-

dimensional implicit function theorem gives the reverse directional derivative

$$\mathbf{J}^T(x) \cdot \boldsymbol{\alpha} = -\mathbf{C}_x^T \cdot \left(\mathbf{C}_y^{-1}\right)^T \cdot \mathbf{J}_g^T \cdot \boldsymbol{\alpha}.$$

where

$$\begin{aligned}(\mathbf{J}_g)_{ij} &= \frac{\partial g_j}{\partial y_i}(f(x)) \\ (\mathbf{C}_y)_{ij} &= \frac{\partial c_i}{\partial y_j}(x, f(x)) \\ (\mathbf{C}_x)_{ij} &= \frac{\partial c_i}{\partial x_j}(x, f(x)).\end{aligned}$$

Here we will take g to be the identify function so that $\mathbf{J}_g$ reduces to the identify matrix and the reverse directional derivative simplifies to

$$\mathbf{J}^T(x) \cdot \boldsymbol{\alpha} = -\mathbf{C}_x^T \cdot \left(\mathbf{C}_y^{-1}\right)^T \cdot \boldsymbol{\alpha}.$$

To apply the adjoint method we need to construct a Lagrangian function that satisfies

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x_i}(x, f(x)) = (\mathbf{J}^T \cdot \boldsymbol{\alpha})_i(x).$$

For example we can take

$$\mathcal{L} = \mathbf{f}^T(x) \cdot \boldsymbol{\alpha} = \sum_{j=1}^{J} f_j(x) \cdot \alpha_j.$$

Next we need to augment $\mathcal{L}$ with the contribution from the constraints $\mathcal{Q}$. Because inner products vanish whenever either input is zero we can take

$$\mathcal{Q} = \mathbf{c}^T(x, y) \cdot \boldsymbol{\lambda} = \sum_{j=1}^{J} c_j(x, y) \cdot \lambda_j,$$

for any real-valued, non-zero constants λ_j.

With these choices of $\mathcal{L}$ and Q the adjoint system becomes

$$\begin{aligned} 0 &= \frac{\partial \mathcal{L}}{\partial y_j} + \frac{\partial Q}{\partial y_j} \\ &= \alpha_j + \sum_{j'=1}^{J} \frac{\partial c_{j'}}{\partial y_j}(x, f(x)) \cdot \lambda_{j'}, \end{aligned}$$

or, in matrix notation,

$$0 = \boldsymbol{\alpha} + \mathbf{C}_y^T \cdot \boldsymbol{\lambda}.$$

Because $\mathbf{C}_y$ is non-singular we can directly solve this for $\boldsymbol{\lambda}$,

$$\boldsymbol{\lambda} = -(\mathbf{C}_y^{-1})^T \cdot \boldsymbol{\alpha}.$$

Substituting this into the remaining terms then gives

$$\begin{aligned} (\mathbf{J}^T \cdot \boldsymbol{\alpha})_i(x) &= \frac{\mathrm{d}\mathcal{J}}{\mathrm{d}x_i}(x, f(x)) \\ &= \frac{\partial \mathcal{L}}{\partial x_i}(x, f(x)) + \frac{\partial Q}{\partial x_i}(x, f(x)) \\ &= 0 + \sum_{j'=1}^{J} \lambda_{j'} \cdot \frac{\partial c_{j'}}{\partial x_i}(x, f(x)) \\ &= (\mathbf{C}_x^T \cdot \boldsymbol{\lambda})_i \\ &= (-\mathbf{C}_x^T \cdot (\mathbf{C}_y^{-1})^T \cdot \boldsymbol{\alpha})_i, \end{aligned}$$

or

$$\mathbf{J}^T \cdot \boldsymbol{\alpha} = -\mathbf{C}_x^T \cdot (\mathbf{C}_y^{-1})^T \cdot \mathbf{J}_g^T \cdot \boldsymbol{\alpha}.$$

which is exactly the same as the result from the implicit function theorem.

In this general case there isn't any structure in the constraint functions to exploit and consequently both methods yield equivalent calculations.

Difference Equations.

To better contrast the two methods let's consider a more structured system. A discrete dynamical system over the state space $\mathbb{R}^N$ defines trajectories

$$y = (\mathbf{y}_1, \ldots, \mathbf{y}_i, \ldots, \mathbf{y}_I),$$

where the individual states $\mathbf{y}_i \in \mathbb{R}^N$ are implicitly defined by the difference equations

$$\mathbf{y}_{i+1} - \mathbf{y}_i = \boldsymbol{\Delta}(\mathbf{y}_i, x, i),$$

for some initial condition $\mathbf{y}_0 = \mathbf{u}(x)$. To simplify the notation we will write

$$\boldsymbol{\Delta}_i = \boldsymbol{\Delta}(\mathbf{y}_i, x, i)$$

from here on.

Organizing these difference equations into constraint equations defines a highly structured algebraic system,

$$\mathbf{y}_1 - \mathbf{y}_0 - \boldsymbol{\Delta}_0 = c_1(x, y) = 0$$

$$\ldots$$

$$\mathbf{y}_i - \mathbf{y}_{i-1} - \boldsymbol{\Delta}_{i-1} = c_i(x, y) = 0$$

$$\ldots$$

$$\mathbf{y}_I - \mathbf{y}_{I-1} - \boldsymbol{\Delta}_{I-1} = c_I(x, y) = 0,$$

which then sets the stage for the implicit function machinery.

Formally this system of constraints defines an entire trajectory; the output space is given by $Y \subset \mathbb{R}^{N \times I}$. Often, however, we are interested not in the entire trajectory but only the final state,

$\mathbf{y}_I \in \mathbb{R}^N$. Conveniently we can readily accommodate this by using the summary function to project out the final state,

$$\begin{aligned} g : Y = \mathbb{R}^{N \times I} &\to \mathbb{R} \\ (\mathbf{y}_1, \ldots, \mathbf{y}_i, \ldots, \mathbf{y}_I) &\mapsto \mathbf{y}_I. \end{aligned}$$

Having defined an implicit system and summary function we can now apply the implicit function theorem and adjoint methods to derive the gradients of the implicitly-defined final state. Although these two methods yield equivalent results, the adjoint method more directly incorporates the natural structure of the problem without any explicit linear algebra.

Differentiation with the Implicit Function Theorem. To apply the implicit function theorem directly we proceed as in A.3.3 and and compute

$$\mathbf{J}^T \cdot \boldsymbol{\alpha} = -\mathbf{C}_x^T \cdot (\mathbf{C}_y^{-1})^T \cdot \mathbf{J}_g^T \cdot \boldsymbol{\alpha}.$$

term by term from the right.

Our chosen summary function yields the Jacobian matrix

$$\mathbf{J}_g = \frac{\partial g}{\partial \mathbf{y}} = [\mathbf{0}_N, \cdots, \mathbf{0}_N, \mathbf{I}_N],$$

where $\mathbf{0}_N$ is an $N \times N$ matrix of zeros and $\mathbf{I}_N$ is the $N \times N$ identity matrix. The first contraction on the right then gives

$$\boldsymbol{\beta}^T = (\mathbf{J}_g^T \cdot \boldsymbol{\alpha})^T = [\underbrace{0, 0, \cdots, 0}_{(I-1)N}, \underbrace{\alpha_1, \alpha_2, \cdots, \alpha_N}_{N}].$$

At this point we construct $\mathbf{C}_y$ from the derivatives of the constraint functions,

$$\mathbf{C}_y = \begin{bmatrix} \mathbf{I}_N & \mathbf{0}_N & \cdots & & \\ \mathbf{C}_y^1 & \mathbf{I}_N & \mathbf{0}_N & \cdots & \\ \mathbf{0}_N & \mathbf{C}_y^2 & \mathbf{I}_N & \mathbf{0}_N & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ \mathbf{0}_N & \cdots & \cdots & \mathbf{C}_y^{I-1} & \mathbf{I}_N \end{bmatrix},$$

where

$$\mathbf{C}_y^i = \frac{\partial}{\partial \mathbf{y}_i}(\mathbf{y}_{i+1} - \mathbf{y}_i - \boldsymbol{\Delta}_i) = -1 - \frac{\partial \boldsymbol{\Delta}_i}{\partial \mathbf{y}_i},$$

and then solve the linear system

$$\mathbf{C}_y^T \cdot \boldsymbol{\gamma} = \boldsymbol{\beta}.$$

Because of the structure of the constraints the matrix $\mathbf{C}_y$ is triangular and with enough linear algebra proficiency we would know that we can efficiently solve for $\boldsymbol{\gamma}$ with backwards elimination. To clarify the derivation we first split the elements of $\boldsymbol{\gamma}$ into subvectors of length N,

$$\boldsymbol{\gamma}^T = [\boldsymbol{\gamma}_1, ..., \boldsymbol{\gamma}_I].$$

The linear system can then be written as

$$\begin{bmatrix} \mathbf{I}_N & \mathbf{C}_y^1 & \mathbf{0}_N & \cdots & \\ \mathbf{0}_N & \mathbf{I}_N & \mathbf{C}_y^2 & \mathbf{0}_N & \cdots \\ \mathbf{0}_N & \mathbf{0}_N & \mathbf{I}_N & \mathbf{C}_y^3 & \mathbf{0}_N \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} & \mathbf{I}_N \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \boldsymbol{\gamma}_3 \\ \vdots \\ \boldsymbol{\gamma}_I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \boldsymbol{\alpha} \end{bmatrix}.$$

Starting from the bottom and going up we can solve this system recursively to obtain the *backward*

difference equations

$$\boldsymbol{\gamma}_i - \left(1 + \frac{\partial \boldsymbol{\Delta}_i}{\partial y_i}\right) \boldsymbol{\gamma}_{i+1} = 0$$

with terminal condition

$$\boldsymbol{\gamma}_I = \boldsymbol{\alpha}.$$

We now evaluate $\mathbf{C}_x$; for $i > 1$ the derivatives are straightforward,

$$(\mathbf{C}_x)_i = \frac{\partial c_i}{\partial x} = -\frac{\partial \boldsymbol{\Delta}_{i-1}}{\partial x},$$

but for $i = 1$ we have to be careful to incorporate the implicit dependence of the initial state, $\mathbf{y}_0 = \mathbf{u}(x)$, on x,

$$\begin{aligned}
(\mathbf{C}_x)_1 &= \frac{\partial c_1}{\partial x} \\
&= \frac{\partial}{\partial x} \left(\mathbf{y}_1 - \mathbf{y}_0 - \boldsymbol{\Delta}(\mathbf{y}_0, x, 0)\right) \\
&= \frac{\partial}{\partial x} \left(\mathbf{y}_1 - \mathbf{u}(x) - \boldsymbol{\Delta}(\mathbf{u}(x), x, 0)\right) \\
&= -\frac{\partial \mathbf{u}}{\partial x} - \left(\frac{\partial \boldsymbol{\Delta}}{\partial x} + \frac{\partial \boldsymbol{\Delta}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial x}\right) \\
&= -\left(1 + \frac{\partial \boldsymbol{\Delta}}{\partial \mathbf{u}}\right) \frac{\partial \mathbf{u}}{\partial x} - \frac{\partial \boldsymbol{\Delta}}{\partial x}.
\end{aligned}$$

Finally we multiply $\mathbf{C}_x$ and $\boldsymbol{\gamma}$ to give

$$\begin{aligned}
\left(\frac{\mathrm{d}\mathbf{y}_I}{\mathrm{d}x}\right)^T \cdot \boldsymbol{\alpha} &= -\mathbf{C}_x^T \cdot \boldsymbol{\gamma} \\
&= \left(\frac{\partial \mathbf{u}}{\partial x}\right)^T \cdot \left(1 + \frac{\partial \boldsymbol{\Delta}_0}{\partial \mathbf{u}}\right)^T \cdot \boldsymbol{\gamma}_1 + \sum_{i=1}^{I} \left(\frac{\partial \boldsymbol{\Delta}_{i-1}}{\partial x}\right)^T \cdot \boldsymbol{\gamma}_i.
\end{aligned}$$

Although the steps are straightforward, implementing them correctly, let alone efficiently, has required careful organization.

Differentiation with the Adjoint Method. Because this discrete dynamical system is a special case of an algebraic system we could appeal to the augmented Lagrangian that we constructed in Sec-

tion A.3.3 and then repeat the same linear algebra needed for the implicit function theorem method. A more astute choice of Lagrangian, however, allows us to directly exploit the structure of the constraints and the summary function.

Given our choice of summary function we need to construct a Lagrangian function which satisfies

$$\frac{\partial \mathcal{L}}{\partial x_i}(x, \mathbf{y}_I) = \left(\frac{\mathrm{d}\mathbf{y}_I}{\mathrm{d}x_i}\right)^T \cdot \boldsymbol{\alpha}.$$

Because $\boldsymbol{\Delta}_i$ is a telescoping series,

$$\mathbf{y}_I = \mathbf{u}(x) + \sum_{i=1}^{I} \boldsymbol{\Delta}_i,$$

a natural choice is

$$\begin{aligned}\mathcal{L}(x, \mathbf{y}) &= \mathbf{y}_I^T \cdot \boldsymbol{\alpha} \\ &= \mathbf{u}^T(x) \cdot \boldsymbol{\alpha} + \sum_{i=1}^{I} \boldsymbol{\Delta}_i^T \cdot \boldsymbol{\alpha}.\end{aligned}$$

For the constraint term C we utilize a similar form as in the general algebraic case,

$$\begin{aligned}Q(x, y) &= \sum_{i=1}^{I} \mathbf{c}_i^T \cdot \boldsymbol{\lambda}_i \\ &= \sum_{i=1}^{I} [\mathbf{y}_i - \mathbf{y}_{i-1} - \boldsymbol{\Delta}_{i-1}]^T \cdot \boldsymbol{\lambda}_i,\end{aligned}$$

where $\boldsymbol{\lambda}_i \in \mathbb{R}^N$.

The adjoint system defined by $\mathcal{L}$ and Q decouples into the equations

$$\begin{aligned}0 &= \frac{\partial \mathcal{L}}{\partial \mathbf{y}_i} + \frac{\partial Q}{\partial \mathbf{y}_i} \\ &= \left(\frac{\partial \boldsymbol{\Delta}_i}{\partial y_i}\right)^T \cdot \boldsymbol{\alpha} + \lambda_{i+1} - \lambda_i - \left(\frac{\partial \boldsymbol{\Delta}_i}{\partial y_i}\right)^T \cdot \lambda_i.\end{aligned}$$

for $i \in \{1, \ldots, I-1\}$ along with the terminal condition for $i = I$,

$$0 = \boldsymbol{\lambda}_I.$$

In other words the adjoint system defines a *backward difference equation* that we can solve recursively from λ_I to λ_{I-1} all the way to λ_1.

Once we have solved for these adjoint states we can substitute them into the remaining terms to give the desired directional derivative,

$$\begin{aligned}\left(\frac{\mathrm{d}\mathbf{y}_I}{\mathrm{d}x}\right)^T \cdot \boldsymbol{\alpha} &= \frac{\partial \mathcal{L}}{\partial x} + \frac{\partial \mathcal{Q}}{\partial x} \\ &= \left(\frac{\partial \mathbf{u}}{\partial x}\right)^T \cdot \left(1 + \frac{\partial \boldsymbol{\Delta}_0}{\partial \mathbf{u}}\right)^T \cdot (\boldsymbol{\alpha} - \boldsymbol{\lambda}_0) + \sum_{i=1}^{I} \left(\frac{\partial \boldsymbol{\Delta}_i}{\partial x}\right)^T \cdot (\boldsymbol{\alpha} - \boldsymbol{\lambda}_i).\end{aligned}$$

As expected the adjoint method has provided an alternative path to the same result we obtained with the implicit function theorem. Indeed matching the two expressions for $(\mathrm{d}\mathbf{y}/\mathrm{d}x)^T \cdot \boldsymbol{\alpha}$, suggests taking

$$\boldsymbol{\gamma}_i = \boldsymbol{\alpha} - \boldsymbol{\lambda}_i,$$

in which case the intermediate difference equations that arise in both methods are exactly the same as well! The advantage of the adjoint method is that we did not have to construct $\mathbf{J}_g$, $\mathbf{C}_y$, or $\mathbf{C}_x$, let alone manage their sparsity to ensure the most efficient computation.

Note also that this procedure yields the same result for difference equations derived in (Betancourt 2020) only with fewer steps, and hence fewer opportunities for mistakes.

Optimization.

Another common class of finite-dimensional, implicit functions are defined as solutions to optimization problems. Given an *objective function* $F : X \times Y \to \mathbb{R}$ we can define the output of

the implicit function as the value which maximizes that objective function for a given input,

$$y = \underset{v}{\text{argmax}}\, F(x, v).$$

In a neighborhood $U \times V \subset X \times Y$ where $F(x, -)$ is convex for all $x \in U$, this implicit function is also defined by the differential constraint

$$c(x, y) = \frac{\partial F}{\partial y}(x, y) = 0,$$

which allows us to apply our machinery to evaluate the derivatives of this implicit function. The convexity constraint is key here; without it the constraint function will identify only general extrema which can include not only maxima but also minima and saddle points.

When $X \subset \mathbb{R}^I$ and $Y \subset \mathbb{R}^J$ the differential constraint function reduces to an algebraic system and we can directly apply the results of Section A.3.3. In particular without any assumptions on the objective function there is no difference in the implementation of the implicit function theorem or adjoint methods.

We start by setting the summary function to the identity so that $\mathbf{J}_g = \mathbb{I}$ and

$$\boldsymbol{\beta} = \mathbf{J}_g^T \cdot \boldsymbol{\alpha} = \boldsymbol{\alpha}.$$

Next we compute

$$(\mathbf{C}_y)_{ij} = \frac{\partial c_i}{\partial y_j}(x, y) = \frac{\partial^2 F}{\partial y_i \partial y_j}(x, y),$$

for example analytically or with higher-order automatic differentiation (Griewank and Walther 2008; Betancourt 2018b). Conveniently when the optimization is implemented with a higher-order numerical method this Hessian matrix will already be available at the final solution. Once $\mathbf{C}_y$ has been constructed we then solve the linear system

$$\frac{\partial^2 F}{\partial y_i \partial y_j}(x, y) \cdot \boldsymbol{\gamma} = \boldsymbol{\alpha}.$$

Finally we construct

$$(\mathbf{C}_x)_{ij} = \frac{\partial c_i}{\partial x_j} = \frac{\partial^2 F}{\partial y_i \partial x_j},$$

and then evaluate

$$\mathbf{J}^T \cdot \boldsymbol{\alpha} = -\mathbf{C}_x^T \cdot \boldsymbol{\gamma}.$$

This final contraction defines a second-order directional derivative of the objective function. Conveniently it can be evaluated with higher-order automatic differentiation without having to explicitly construct $\mathbf{C}_x$.

These optimization problems become more sophisticated with the introduction of an additional equality constraint so that the output of the implicit function is now defined by the condition

$$y = \underset{s}{\text{argmax}}\ F(x, s) \text{ such that } k(x, y) = 0,$$

for the auxiliary constraint function $k : \mathbb{R}^I \times \mathbb{R}^J \to \mathbb{R}^K$.

In order to define an implicit system that consistently incorporates both of these constraints we have to augment the output space. We first introduce the Lagrange multipliers $\mu \in M \subset \mathbb{R}^K$ and the augmented objective function

$$\begin{aligned} \Phi : X \times Y \times M &\to \mathbb{R} \\ (x, y, \mu) &\mapsto F(x, y) + \mu \cdot k(x, y). \end{aligned}$$

A consistent solution to the constrained optimization problem is then given by

$$(y, \mu) = \underset{s,m}{\text{argmax}}\ \Phi(x, s, m).$$

In other words we can incorporate all of the constraints directly on the augmented output space $\zeta = (y, \mu)$ and then project back down to the original output space using the summary function $g : (y, \mu) \mapsto y$.

Within a sufficiently convex neighborhood we can also define the constrained optimization problem with a constraint function over this augmented output space,

$$c(x, \zeta) = \frac{\partial \Phi}{\partial \zeta}(x, \zeta) = 0,$$

which allows us to apply our methods for evaluating directional derivatives.

From the projective summary function we first construct

$$\boldsymbol{\beta} = \mathbf{J}_g^T \cdot \boldsymbol{\alpha}.$$

Next we differentiate the constraint function on the augmented output space,

$$(\mathbf{C}_\zeta)_{ij} = \frac{\partial^2 \Phi}{\partial \zeta_i \partial \zeta_j}(x, \zeta),$$

and then solve the linear system

$$\frac{\partial^2 \Phi}{\partial \zeta_i \partial \zeta_j}(x, \zeta) \cdot \boldsymbol{\gamma} = \boldsymbol{\beta}.$$

Finally we construct

$$(\mathbf{C}_x)_{ij} = \frac{\partial^2 \Phi}{\partial \zeta_i \partial x_j},$$

and then evaluate

$$\mathbf{J}^T \cdot \boldsymbol{\alpha} = -\mathbf{C}_x^T \cdot \boldsymbol{\gamma}.$$

As before this final contraction defines a second-order directional derivative that can be directly evaluated with higher-order automatic differentiation, although this time on the augmented output space.

Inequality constraints introduce an additional challenge. While the Karush-Kuhn-Tucker conditions define an appropriate system of constraints (Karush 1939; Kuhn and Tucker 1951), the dimension of the constraint space varies with the input x as different inequality constraints become active. Moreover even when the objective function and the additional constraint functions are all

smooth the implicit function they define might not be differentiable at every x. We leave a detailed treatment of this problem to future work.

A.4 Automatic Differentiation of Infinite-Dimensional Implicit Functions

With care in how derivatives are defined we can immediately generalize the finite dimensional methods for evaluating directional derivatives of implicit functions to infinite dimensional systems, for example systems that implicitly define entire trajectories, fields, or even probability distributions. In this section we review the basics of differentiable, infinite dimensional spaces and the generalization of the implicit function theorem before generalizing the directional derivative evaluation methods and demonstrating them on two instructive examples.

A.4.1 The Infinite-Dimensional Implicit Function Theorem

The machinery of differential calculus over the real numbers generalizes quite naturally to a Fréchet calculus over Banach vector spaces. In this section we review the key concepts and then use them to construct an infinite-dimensional implicit function theorem. For a more in depth presentation of these topics see for example Kesavan 2020.

Fréchet derivatives.

The *Fréchet derivative* generalizes the concept of a derivative that we introduced for real spaces to the more general Banach vector spaces, or more compactly *Banach spaces*. Banach spaces include not only finite-dimensional Euclidean vector spaces but also infinite-dimensional function spaces.

Consider two Banach spaces, X and Y, and a function $f : X \to Y$ mapping between them. If f is Fréchet differentiable then the Fréchet derivative assigns to each input point $x \in X$ a bounded, linear map

$$\frac{\delta f}{\delta x} : X \to \mathrm{BL}(X, Y),$$

where $\text{BL}(X, Y)$ is the space of bounded, linear functions from X to Y. In other words the Fréchet derivative of f evaluated at x defines a bounded, linear map from X to Y,

$$\frac{\delta f}{\delta x}(x) : X \to Y.$$

Note that unlike the total derivative we introduced on real spaces the Fréchet derivative is defined directly on a vector space and so there is no distinction between locations and directions.

If Z is a third Banach space then the composition of f with $g : Y \to Z$ defines a map from X to Z,

$$g \circ f : X \to Z.$$

The Fréchet derivative of this composition,

$$\frac{\delta (g \circ f)}{\delta x}(x) : X \to Z,$$

follows a chain rule,

$$\frac{\delta (g \circ f)}{\delta x}(x) = \underbrace{\frac{\delta g}{\delta x}(f(x))}_{Y \to Z} \circ \underbrace{\frac{\delta f}{\delta x}(x)}_{X \to Y}.$$

A binary map $h : X \times Y \to Z$ also admits a partial Fréchet derivative. For example the partial Fréchet derivative of h with respect to the first input defines a map

$$\frac{\delta h}{\delta x} : X \times Y \to \text{BL}(X, Z).$$

Equivalently the partial Fréchet derivative of h *evaluated at* (x, y) is a bounded, linear map from X to Z,

$$\frac{\delta h}{\delta x}(x, y) : X \to Z.$$

Similarly we can define a partial derivative with respect to the second output as

$$\frac{\delta h}{\delta y} : X \times Y \to \mathrm{BL}(Y, Z)$$

with

$$\frac{\delta h}{\delta y}(x, y) : Y \to Z.$$

When a binary map like h is composed with unary maps in each argument then we can use the chain rule to define a notion of a total Fréchet derivative. Let $h : X \times Y \to Z$, $f : W \to X$ and $g : W \to Y$. The component-wise composition $h(f(\,), g(\,))$ then defines a unary map

$$q = h(f(\,), g(\,)) : W \to Z,$$

with the corresponding Fréchet derivative

$$\frac{\delta q}{\delta w}(w) : W \to Z,$$

which decomposes into contributions from each argument,

$$\frac{\delta q}{\delta x}(w) = \frac{\delta h}{\delta x}(f(w), g(w)) \circ \frac{\delta f}{\delta w}(w) + \frac{\delta h}{\delta y}(f(w), g(w)) \circ \frac{\delta g}{\delta w}(w).$$

The Implicit Function Theorem.

The finite dimensional implicit function theorem immediately generalizes to Banach spaces with the use of Fréchet derivatives. Consider a Banach space of known inputs, X, a Banach space of unknown outputs, Y, a Banach space of constraint values, Z, and the constraint function

$$\begin{aligned} c : X \times Y &\to Z \\ (x, y) &\mapsto c(x, y). \end{aligned}$$

Once again, we examine a particular input, $x \in X$, and the neighborhoods $x \in U \subset X$, $V \subset Y$ and $W \subset Z$.

If c is Fréchet differentiable across $U \times V$ then it defines two partial Fréchet derivatives

$$\frac{\delta c}{\delta x}(x, y) : U \to W$$

and

$$\frac{\delta c}{\delta y}(x, y) : V \to W.$$

When $\delta c / \delta y(y)$ defines a bijection from V to W we can also define a corresponding inverse operator,

$$\left(\frac{\delta c}{\delta y}(x, y) \right)^{-1} : W \to V.$$

In this case the implicit function theorem guarantees that the kernel of the constraint function, $c^{-1}(0)$, implicitly defines a Fréchet differentiable function from the input space to the output space, $f : U \to V$ that satisfies $c(x, f(x)) = 0$.

As in the finite-dimensional case we can calculate the Fréchet derivative of this implicit function by differentiating the constraint function,

$$\begin{aligned} 0 &= \frac{\delta c}{\delta x}(x) \\ &= \frac{\delta c}{\delta x}(x, y) + \frac{\delta c}{\delta y}(x, y) \circ \frac{\delta y}{\delta x}(x). \end{aligned}$$

Because $\delta c / \delta y$ is invertible at $(x, f(x))$ we can immediately solve for $\delta y / \delta x$ to give

$$\underbrace{\frac{\delta f}{\delta x}(x)}_{U \to V} = - \underbrace{\left(\frac{\delta c}{\delta y}(x, f(x)) \right)^{-1}}_{W \to V} \circ \underbrace{\frac{\delta c}{\delta x}(x, f(x))}_{U \to W}.$$

Moving forward we will denote $J_f = \delta f / \delta x(x)$ the Fréchet Jacobian.

A.4.2 Evaluating Directional Derivatives of Infinite-Dimensional Implicit Functions

While we can handle infinite dimensional spaces mathematically any practical automatic differentiation implementation will be restricted to finite dimensional inputs and final outputs. To that end we will assume that the input space is real, $X = \mathbb{R}^I$, while allowing the output space Y to be infinite dimensional, for example corresponding to a smooth trajectory or a latent field. We will use the summary function, however, to project that potentially-infinite dimensional output to a finite-dimensional real space, $g : Y \to \mathbb{R}^J$. For example we might consider a dynamical system that defines an entire trajectory but project out only the finite-dimensional final state.

In order to implement such a system in a reverse mode automatic differentiation library we then need to be able to evaluate the reverse directional derivative

$$J^\dagger_{g \circ f}(x)(\boldsymbol{\alpha}) = \left(J^\dagger_f(x) \circ J^\dagger_g(f(x))\right)(\boldsymbol{\alpha}).$$

Because $g \circ f$ is a real-to-real map the total action is given by a matrix-vector product,

$$J^\dagger_{g \circ f}(x)(\boldsymbol{\alpha}) = \mathbf{J}^T_{g \circ f} \cdot \boldsymbol{\alpha},$$

but the component operators $J^\dagger_f(x)$ and $J^\dagger_g(f(x))$ will not be unless Y is a finite-dimensional real space.

Here we will consider the generalizations of the three methods for evaluating this directional derivative that we constructed in Section A.3.

Trace Method.

Although we can't practically construct a numerical method for an implicit function $f : \mathbb{R}^I \to Y$ with infinite-dimensional output space, we often can construct numerical methods for the finite-dimensional composition $g \circ f : \mathbb{R}^I \to \mathbb{R}^J$. Numerical integrators for ordinary and partial differential equations, for example, discretize Y in order to approximate the finite dimensional outputs

of $g \circ f$ without having to confront an infinite dimensional space directly.

Because the intermediate calculations of the numerical solver will be finite-dimensional we immediately apply the trace method discussed in Section A.3.2, automatically differentiating through each iteration of the solve, to approximate $J_{g \circ f}(x)(\boldsymbol{\alpha})$. As in the finite-dimensional case, however, this direct approach is often too computationally expensive and memory intensive to be practical.

Infinite-Dimensional Implicit Function Theorem.

In theory the reverse directional derivative is given immediately by the implicit function theorem,

$$\begin{aligned} J_{g \circ f}^{\dagger}(x)(\boldsymbol{\alpha}) &= \left(J_f^{\dagger}(x) \circ J_g^{\dagger}(f(x))\right)(\boldsymbol{\alpha}) \\ &= \left(-\left(\frac{\delta c}{\delta x}(x, f(x))\right)^{\dagger} \circ \left(\left(\frac{\delta c}{\delta y}(x, f(x))\right)^{\dagger}\right)^{-1} \circ J_g^{\dagger}(f(x)) \right)(\boldsymbol{\alpha}). \end{aligned}$$

Unfortunately whenever Y is infinite-dimensional each of these Fréchet derivatives will be infinite dimensional operators that are difficult, if not impossible, to implement in practice. For example $J_g^{\dagger}(f(x))$ maps the finite covector $\boldsymbol{\alpha}$ to an infinite dimensional space that can't be represented in finite memory.

The Infinite-Dimensional Adjoint Method.

With a careful use of Fréchet derivatives the adjoint method defined in Section 6 generalizes to the case where Y and Z are general Banach spaces. As usual we consider a particular input, $x \in X$, and the neighborhoods $x \in U \subset X$, $V \subset Y$, and $W \subset Z$.

We first define a binary Lagrangian functional

$$\mathcal{L} : U \times V \to \mathbb{R}$$

that gives a unary functional when we substitute the solution of the implicit function,

$$\mathcal{L}(x) = \mathcal{L}(x, f(x))$$

with the Fréchet derivative

$$\frac{\delta \mathcal{L}}{\delta x}(x) = J^{\dagger}_{g \circ f}(x)(\boldsymbol{\alpha}).$$

Next we introduce a mapping $\Lambda : W \to \mathbb{R}$ that preserves the kernel of the constraint function, $\Lambda \circ c(x, y) = 0$ whenever $c(x, y) = 0$. Composing this mapping with the constraint function gives the constraint Lagrangian functional,

$$\mathcal{Q} = \Lambda \circ c : U \times V \to \mathbb{R},$$

with $\mathcal{Q}(x, f(x)) = 0$ for all $x \in U$.

Together these two functionals define an augmented Lagrangian functional,

$$\mathcal{J} = \mathcal{L} + \mathcal{Q},$$

along with the corresponding unary functional

$$\begin{aligned} \mathcal{J}(x) &= \mathcal{J}(x, f(x)) \\ &= \mathcal{L}(x, f(x)) + \mathcal{Q}(x, f(x)) \\ &= \mathcal{L}(x, f(x)), \end{aligned}$$

because the constraint functional vanishes by construction when evaluated at the solution to the constraint problem.

The total derivative of this augmented unary functional is given by

$$\begin{aligned}\frac{\delta \mathcal{J}}{\delta x}(x) &= \frac{\delta \mathcal{J}}{\delta x}(x, f(x)) \\ &= \frac{\delta \mathcal{J}}{\partial x}(x, f(x)) + \frac{\delta \mathcal{J}}{\delta y}(x, f(x)) \circ \frac{\delta f}{\delta x}(x).\end{aligned}$$

If we could engineer a map Λ such that the *adjoint system* $\delta \mathcal{J} / \delta y$ vanishes,

$$\begin{aligned}0 &= \frac{\delta \mathcal{J}}{\delta y}(x, f(x)) \\ &= \frac{\delta \mathcal{L}}{\delta y}(x, f(x)) + \frac{\delta \mathcal{Q}}{\delta y}(x, f(x)) \\ &= \frac{\delta \mathcal{L}}{\delta y}(x, f(x)) + \frac{\delta \Lambda}{\delta c}(x, f(x)) \circ \frac{\delta c}{\delta y}(x, f(x))\end{aligned}$$

then the reverse directional derivative would reduce to

$$\begin{aligned}J^{\dagger}_{g \circ f}(x)(\boldsymbol{\alpha}) &= \frac{\delta \mathcal{J}}{\delta x}(x, f(x)) \\ &= \frac{\delta \mathcal{L}}{\delta x}(x, f(x)) + \frac{\delta \mathcal{Q}}{\delta x}(x, f(x)) \\ &= \frac{\delta \mathcal{L}}{\delta x}(x, f(x)) + \frac{\delta \Lambda}{\delta c}(x, f(x)) \circ \frac{\delta c}{\delta x}(x, f(x)).\end{aligned}$$

Unfortunately if Y is infinite-dimensional then the adjoint system also becomes infinite-dimensional, and the general Fréchet derivatives will typically be too ungainly to implement in practice.

One important exception is when Y is a *Sobolev space*. Informally a Sobolev space of order k is an infinite-dimensional Banach space comprised of integrable, real-valued functions whose first k derivatives are sufficiently well-defined. What makes Sobolev spaces so useful is that a large class of functionals over these spaces can be written as integrals.

Consider for example the input space $T \subseteq \mathbb{R}$, the output space $S \subseteq \mathbb{R}^N$, and the Sobolev space Y of k-times differentiable functions $\mathbf{y} : T \to S$, such as those arising from the solutions to k-th

order ordinary differential equations. Any integral of the form

$$\begin{aligned}\mathcal{G}(\mathbf{y}) &= \int_T \mathrm{d}t\, g\left(t, \mathbf{y}(t), \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t}(t), \ldots, \frac{\mathrm{d}^K\mathbf{y}}{\mathrm{d}t^K}(t)\right) \\ &\equiv \int_T \mathrm{d}t\, g\Big(t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big)\end{aligned}$$

defines a unary, real-valued functional $\mathcal{G} : Y \to \mathbb{R}$ whose Fréchet derivative is given by

$$\begin{aligned}\frac{\delta\mathcal{G}}{\delta\mathbf{y}}(\mathbf{y}) &= \frac{\delta}{\delta\mathbf{y}} \int_T \mathrm{d}t\, g\Big(t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \\ &= \int_T \mathrm{d}t \Bigg[\frac{\partial g}{\partial \mathbf{y}}\Big(t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \\ &\qquad + \sum_{k=1}^{K} \frac{\partial g}{\partial \mathbf{y}^{(k)}}\Big(t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \cdot \frac{\delta \mathbf{y}^{(k)}}{\delta \mathbf{y}} \Bigg].\end{aligned}$$

Similarly if $X \subseteq \mathbb{R}^I$ then

$$\mathcal{G}(x, \mathbf{y}) = \int_T \mathrm{d}t\, g\left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\right)$$

defines a binary, real-valued functional $\mathcal{G} : X \times Y \to \mathbb{R}$. Consequently we can construct a large class of Lagrangian functionals and constraint functionals as integrals,

$$\begin{aligned}\mathcal{L}(x, \mathbf{y}) &= \int_T \mathrm{d}t\, l\Big(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \\ \mathcal{Q}(x, \mathbf{y}) &= \int_T \mathrm{d}t\, q\Big(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big)\end{aligned}$$

with the corresponding augmented Lagrangian,

$$\begin{aligned}\mathcal{J}(x, \mathbf{y}) &= \int_T \mathrm{d}t\, j\Big(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \\ &= \int_T \mathrm{d}t \Big[l\Big(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \\ &\qquad + q\Big(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \Big].\end{aligned}$$

The choice of l needs to verify the condition that

$$\begin{aligned} J^{\dagger}_{g\circ f}(x)(\boldsymbol{\alpha}) &= \frac{\delta}{\delta x}\mathcal{L}(x, f(x)) \\ &= \frac{\delta}{\delta x}\int_T \mathrm{d}t\; l\Big(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \\ &= \int_T \mathrm{d}t \Bigg[\quad \frac{\partial l}{\partial x}\Big(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \\ &\qquad + \sum_{k=1}^{K} \frac{\partial l}{\partial \mathbf{y}^{(k)}}\Big(t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big) \cdot \frac{\delta \mathbf{y}^{(k)}}{\delta x} \Bigg]. \end{aligned}$$

Depending on the problems, choosing l can be straightforward or require a bit more work, as we will see in the examples. The choice of q needs to satisfy the condition that, for all $x \in U$,

$$\begin{aligned} 0 &= Q(x, f(x)) \\ &= \int_T \mathrm{d}t\; q\Big(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\Big). \end{aligned}$$

One straightforward strategy is to define the integrand q such that it vanishes whenever evaluated at the solution, $(x, f(x))$. We can accomplish this for example by taking q to be the Sobolev inner product of the constraint function, f, with an auxiliary function $\lambda \in Z$,

$$q(x, f(x)) = \langle \lambda,\; c(x, f(x)) \rangle,$$

where $\langle\, , \rangle$ denotes the Sobolev inner product. When evaluated at the solution the constraint function c vanishes so that the above inner product, and hence the integrand q, also vanish.

When using functionals of this form the reverse directional derivative becomes

$$\begin{aligned}
J^{\dagger}_{g\circ f}(x)(\boldsymbol{\alpha}) &= \frac{\delta \mathcal{J}}{\delta x}(x, f(x)) \\
&= \int_T \mathrm{d}t \Bigg[\quad \frac{\partial j}{\partial x}\left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\right) \\
&\qquad + \frac{\partial j}{\partial \mathbf{y}}\left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\right) \cdot \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x}(t) \\
&\qquad + \sum_{k=1}^{K} \frac{\partial j}{\partial \mathbf{y}^{(k)}}\left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\right) \cdot \frac{\delta \mathbf{y}^{(k)}}{\delta \mathbf{y}} \cdot \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x}(t) \Bigg] \\
&= \int_T \mathrm{d}t \Bigg[\quad \frac{\partial j}{\partial x}\left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\right) \\
&\qquad + \frac{\partial j}{\partial \mathbf{y}}\left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\right) \cdot \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x}(t) \\
&\qquad + \sum_{k=1}^{K} \frac{\partial j}{\partial \mathbf{y}^{(k)}}\left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t)\right) \cdot \frac{\mathrm{d}\mathbf{y}^{(k)}}{\mathrm{d}x}(t) \Bigg].
\end{aligned}$$

Because the elements of the Sobalev space are continuous we can simplify the terms in the last line by repeated application of Stokes' Theorem,

$$\begin{aligned}
\int_T \mathrm{d}t \, \frac{\partial j}{\partial \mathbf{y}^{(k)}} \cdot \frac{\mathrm{d}\mathbf{y}^{(k)}}{\mathrm{d}x} = \quad & \sum_{k'=0}^{k} (-1)^{k'} \left[\frac{\mathrm{d}^{k'}}{\mathrm{d}t^{k'}} \left(\frac{\partial j}{\partial \mathbf{y}^{(k)}} \right) \frac{\mathrm{d}^{k-k'-1}}{\mathrm{d}t^{k-k'-1}} \left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x} \right) \right]_{\partial T} \\
& + (-1)^k \int_T \mathrm{d}t \, \frac{\mathrm{d}^k}{\mathrm{d}t^k} \left(\frac{\partial j}{\partial \mathbf{y}^{(k)}} \right) \cdot \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x},
\end{aligned}$$

where we have dropped the arguments to ease the notational burden. Substituting this into the reverse directional derivative gives

$$\begin{aligned}
J^{\dagger}_{g\circ f}(x)(\boldsymbol{\alpha}) = \quad & \int_T \mathrm{d}t \, \frac{\partial j}{\partial x} \\
& + \sum_{k=1}^{K} \sum_{k'=0}^{k} (-1)^{k'} \left[\frac{\mathrm{d}^{k'}}{\mathrm{d}t^{k'}} \left(\frac{\partial j}{\partial \mathbf{y}^{(k)}} \right) \frac{\mathrm{d}^{k-k'-1}}{\mathrm{d}t^{k-k'-1}} \left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x} \right) \right]_{\partial T} \\
& + \int_T \mathrm{d}t \, \left[\frac{\partial j}{\partial \mathbf{y}} + \sum_{k=1}^{K} (-1)^k \frac{\mathrm{d}^k}{\mathrm{d}t^k} \left(\frac{\partial j}{\partial \mathbf{y}^{(k)}} \right) \right] \cdot \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x}.
\end{aligned}$$

If we can engineer a q such that j solves the *differential adjoint system*

$$0 = \sum_{k=1}^{K} \sum_{k'=0}^{k} (-1)^{k'} \left[\frac{\mathrm{d}^{k'}}{\mathrm{d}t^{k'}} \left(\frac{\partial j}{\partial \mathbf{y}^{(k)}} \right) \frac{\mathrm{d}^{k-k'-1}}{\mathrm{d}t^{k-k'-1}} \left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x} \right) \right]_{\partial T}$$
$$0 = \frac{\partial j}{\partial \mathbf{y}} + \sum_{k=1}^{K} (-1)^{k} \frac{\mathrm{d}^{k}}{\mathrm{d}t^{k}} \left(\frac{\partial j}{\partial \mathbf{y}^{(k)}} \right)$$

then the reverse directional derivative reduces to the manageable form

$$\begin{aligned} J^{\dagger}_{g \circ f}(x)(\boldsymbol{\alpha}) &= \int_T \mathrm{d}t \, \frac{\partial j}{\partial x} \left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t) \right) \\ &= \int_T \mathrm{d}t \left[\frac{\partial l}{\partial x} \left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t) \right) \right. \\ &\qquad \left. + \frac{\partial q}{\partial x} \left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \ldots, \mathbf{y}^{(K)}(t) \right) \right]. \end{aligned}$$

For different choices of Y the form of the functionals, their Fréchet derivatives, and the resulting differential adjoint system will vary, but the basic procedure of the adjoint method remains the same. By considering more sophisticated Sobelev spaces this general adjoint method can be applied to partial differential equations and other more sophisticated infinite-dimensional implicit systems.

A.4.3 Demonstrations

The particular differential adjoint system we have derived in Section A.4.2 is immediately applicable to implicit systems defined by ordinary differential and algebraic differential equations. In this section we demonstrate that application on two such systems.

Ordinary Differential Equations.

Consider a time interval $T = [0, \tau] \subset \mathbb{R}$ and the N-dimensional trajectories that map each time point to an N-dimensional state, $\mathbf{y} : T \to \mathbb{R}^N$. The space of trajectories that are at least once-differentiable forms a first-order Sobalev space, Y.

A linear system of first-order, ordinary differential equations

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{r}(x, \mathbf{y}, t),$$

along with an initial condition $\mathbf{y}(0) = \mathbf{u}(x)$, defines a separate constraint for the trajectory behavior at each $t \in T$,

$$\begin{aligned} c(x, \mathbf{y})(t) &= \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t}(t) - \mathbf{r}(x, \mathbf{y})(t) \\ c(x, \mathbf{y})(0) &= \mathbf{y}(0) - \mathbf{u}(x). \end{aligned}$$

Collecting all of these constraints together defines an infinite-dimensional constraint function $c : X \times Y \to Z$ where Z is also the space of differentiable functions that map from T to $\mathbb{R}^N$.

The summary function $g : \mathbf{y} \mapsto \mathbf{y}(\tau)$ projects infinite-dimensional trajectories down to their N-dimensional final states so that the composition $g \circ f$ maps inputs $x \in X$ to a final state at time $t = \tau$. In order to implement this map into a reverse mode automatic differentiation library we need to be able to implement the reverse directional derivative $J_{g \circ f}(x)(\boldsymbol{\alpha})$.

To implement the adjoint method we need a Lagrangian functional that satisfies

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x_i}(x, f(x)) = \left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x_i}(T) \right)^T \cdot \boldsymbol{\alpha}.$$

Integrating the defining differential equation gives an integral functional that provides a particularly useful option,

$$\mathcal{L}(x, \mathbf{y}) = \mathbf{u}^T(x) \cdot \boldsymbol{\alpha} + \int_0^\tau \mathrm{d}t \; \mathbf{r}^T(x, \mathbf{y}, t) \cdot \boldsymbol{\alpha}.$$

Next we need to construct a functional Q that vanishes when when evaluated at the implicit solution. Integrating over the constraint function gives

$$Q(x, \mathbf{y}) = \left[\mathbf{y}(0) - \mathbf{u}(x) \right]^T \cdot \boldsymbol{\mu} + \int_0^T \mathrm{d}t \; \left[\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t}(t) - \mathbf{r}(x, \mathbf{y}, t) \right]^T \cdot \boldsymbol{\lambda}(t),$$

for any function $\boldsymbol{\lambda} : \mathbb{R} \to \mathbb{R}^N$ and constant $\boldsymbol{\mu} \in \mathbb{R}^N$.

Substituting these integral functionals into the differential adjoint system that we derived in Section A.4.2 yields the system

$$\begin{aligned} 0 &= \left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x}(0)\right)^T \cdot (\boldsymbol{\mu} - \boldsymbol{\lambda}(0)) + \left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x}(\tau)\right)^T \cdot \boldsymbol{\lambda}(\tau) \\ 0 &= \left(\frac{\partial \mathbf{r}}{\partial \mathbf{y}}(x, \mathbf{y}, t)\right)^T \cdot (\boldsymbol{\alpha} - \boldsymbol{\lambda}(t)) - \frac{\mathrm{d}\boldsymbol{\lambda}}{\mathrm{d}t}(t). \end{aligned}$$

The boundary terms vanish if we take $\boldsymbol{\mu} = \boldsymbol{\lambda}(0)$ and $\boldsymbol{\lambda}(\tau) = 0$ while the differential term vanishes for the $\boldsymbol{\lambda}(t)$ given by integrating $\boldsymbol{\lambda}(\tau) = 0$ backwards from $t = \tau$ to $t = 0$. In other words the differential adjoint system defines a linear, first-order ordinary differential equation that reverses time relative to our initial ordinary differential equation.

Once we've solved for $\boldsymbol{\lambda}(t)$ the reverse directional derivative is given by the partial derivatives of the augmented Lagrangian with respect to x,

$$J^{\dagger}_{g \circ f}(x)(\boldsymbol{\alpha}) = \left(\frac{\partial \mathbf{u}}{\partial x}(x)\right)^T \cdot (\boldsymbol{\alpha} - \boldsymbol{\lambda}(0)) + \int_0^T \mathrm{d}t \, \left(\frac{\partial \mathbf{r}}{\partial x}(x, \mathbf{y}, t)\right)^T \cdot (\boldsymbol{\alpha} - \boldsymbol{\lambda}(t)) \, .$$

While we do have to solve both the nominal and adjoint differential equations, these solves require evaluating only finite-dimensional derivatives. We have completely avoided any infinite-dimensional Fréchet derivatives.

Differential Algebraic Equations.

Introducing an algebraic constraint to the previous systems defines a differentiable algebraic system, or DAE. A DAE might, for example, impose the constraint that the component states at each time sum to one so that the states can model how the allocation of a conserved quantity evolves over time.

To simplify the derivation we start by decomposing the trajectories $\mathbf{y}(t) \in \mathbb{R}^N$ into a differential component, $\mathbf{y}^d \in \mathbb{R}^D$, and an algebraic component, $\mathbf{y}^a \in \mathbb{R}^A$, with $N = D + A$. A differential

algebraic constraint function can similarly be decomposed into a differential constraint function,

$$c^d(x, \mathbf{y}, \dot{\mathbf{y}}, t) \in \mathbb{R}^D,$$

where $\dot{\mathbf{y}}$ is shorthand for $\mathrm{d}\mathbf{y}/\mathrm{d}t$, and an algebraic constraint function,

$$c^a(x, \mathbf{y}, t) \in \mathbb{R}^A.$$

with $\mathbf{c} = (c^d, c^a)^T$.

If the differential constraint is given by a linear, first-order differential equation then the differential constraint function becomes

$$\begin{aligned} c^d(x, \mathbf{y}, \dot{\mathbf{y}}, t) &= \dot{\mathbf{y}}^d - \mathbf{r}^d(x, \mathbf{y}, t) \\ c^d(x, \mathbf{y}, \dot{\mathbf{y}}, 0) &= \mathbf{y}(0) - \mathbf{u}(x). \end{aligned}$$

When the constraints are consistent this differential algebraic system implicitly defines a map from inputs $x \in X$ to N-state trajectories, $\mathbf{y} \in T \times \mathbb{R}^N$. We can also write this as a map from inputs and times to states,

$$f : X \times T \to \mathbb{R}^N,$$

such that $c^d(x, \mathbf{f}(x), \dot{\mathbf{f}}(x), t) = 0$ and $c^a(x, \mathbf{f}(x), t) = 0$.

As in the previous example we will consider a summary function that projects the infinite-dimensional trajectories down to their N-dimensional final states, $g : \mathbf{y} \mapsto \mathbf{y}(\tau)$, so that the composition $g \circ f$ maps inputs $x \in X$ to a final state at time $t = \tau$. Our goal is then to evaluate the finite-dimensional reverse directional derivative $J_{g \circ f}(x)(\boldsymbol{\alpha})$.

As with ordinary differential systems integrating the trajectory provides an appropriate integral functional,

$$\mathcal{L}(x, \mathbf{y}) = \mathbf{u}^T(x) \cdot \boldsymbol{\alpha} + \int_0^\tau \mathrm{d}t \; \dot{\mathbf{y}}^T(x, t) \cdot \boldsymbol{\alpha}$$

that satisfies

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}x}(x, \mathbf{f}(x)) = \left(\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}x}(x, \tau)\right)^T \cdot \boldsymbol{\alpha}.$$

Unlike in the ordinary differential case, however, the differential algebraic system does not immediately provide an analytical expression for $\dot{\mathbf{y}}(x)$.

The first-order linear differential equation does provide the derivative of the differential component,

$$\dot{\mathbf{y}}^d = \mathbf{r}^d.$$

In order to obtain the derivative of the algebraic component we have to differentiate the algebraic constraint,

$$\begin{aligned} 0 &= \frac{\mathrm{d}}{\mathrm{d}t} c^a(x, \mathbf{y}, t) \\ &= \frac{\partial c^a}{\partial \mathbf{y}^d} \dot{\mathbf{y}}^d(x, t) + \frac{\partial c^a}{\partial \mathbf{y}^a} \dot{\mathbf{y}}^a(x, t) + \frac{\partial c^a}{\partial t}(x, t), \end{aligned}$$

or

$$\dot{\mathbf{y}}^a = -\left[\frac{\partial c^a}{\partial \mathbf{y}^a}\right]^{-1} \left(\frac{\partial c^a}{\partial \mathbf{y}^d} \dot{\mathbf{y}}^d + \frac{\partial c^a}{\partial t}\right),$$

where we assume $\partial c^a / \partial \mathbf{y}^a$ is square invertible.

The constraint functional $\mathcal{Q}$ is identical to that from the ordinary differential equation system,

$$C(x, \mathbf{y}) = \left[\mathbf{y}(0) - \mathbf{u}(x)\right]^T \cdot \boldsymbol{\mu} + \int_0^\tau \mathrm{d}t\, \mathbf{c}(x, \mathbf{y}, t)^T \cdot \boldsymbol{\lambda}(t).$$

Plugging $\mathcal{J} = \mathcal{L} + \mathcal{Q}$ into the results of Section A.4.2 gives the system

$$\begin{aligned} 0 &= \left[\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x_i}(0) - \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}x_i}(x)\right]^T \cdot \boldsymbol{\mu} + \left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x_i}\right)^T \cdot \boldsymbol{\lambda}_{D0}(t)\Big|_0^\tau \\ 0 &= \left(\frac{\partial \mathbf{r}}{\partial \mathbf{y}}(x, \boldsymbol{y}, t)\right)^T \cdot \boldsymbol{\alpha} + \left(\frac{\partial \mathbf{c}}{\partial \mathbf{y}}(x, \boldsymbol{y}, t)\right)^T \cdot \boldsymbol{\lambda}(t) - \dot{\boldsymbol{\lambda}}_{D0}(t) \end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{\lambda}_{D0}(t) &= \left(\frac{\partial \mathbf{c}}{\partial \mathbf{y}}(x, \mathbf{y}, t)\right)^T \cdot \boldsymbol{\lambda}(t) \\
&= \begin{bmatrix} \frac{\partial \mathbf{c}^d}{\partial \mathbf{y}_d} & \frac{\partial \mathbf{c}^d}{\partial \mathbf{y}_a} \\ \frac{\partial \mathbf{c}^a}{\partial \mathbf{y}_d} & \frac{\partial \mathbf{c}^a}{\partial \mathbf{y}_a} \end{bmatrix} \cdot \boldsymbol{\lambda}(t) \\
&= \begin{bmatrix} \mathbf{I}_{D\times D} & \mathbf{0}_{D\times A} \\ \mathbf{0}_{A\times D} & \mathbf{0}_{A\times A} \end{bmatrix} \cdot \boldsymbol{\lambda}(t) \\
&= \begin{bmatrix} \boldsymbol{\lambda}_{1:D}(t) \\ \mathbf{0}_{1\times A} \end{bmatrix}.
\end{aligned}$$

Decomposing each term in the differential adjoint system into algebraic and a differential components gives

$$\begin{bmatrix} \left(\frac{\partial \mathbf{r}}{\partial \mathbf{y}_d}(x, \mathbf{y}, t)\right)^T \cdot \boldsymbol{\alpha} + \left(\frac{\partial \mathbf{c}_d}{\partial \mathbf{y}_d}(x, \mathbf{y}, t)\right)^T \cdot \boldsymbol{\lambda}_d(t) + \left(\frac{\partial \mathbf{c}_a}{\partial \mathbf{y}_d}(x, \mathbf{y}, t)\right)^T \cdot \boldsymbol{\lambda}_a(t) - \dot{\boldsymbol{\lambda}}_{D0}(t) \\ \left(\frac{\partial \mathbf{r}}{\partial \mathbf{y}_a}(x, \mathbf{y}, t)\right)^T \cdot \boldsymbol{\alpha} + \left(\frac{\partial \mathbf{c}_d}{\partial \mathbf{y}_a}(x, \mathbf{y}, t)\right)^T \cdot \boldsymbol{\lambda}_d(t) + \left(\frac{\partial \mathbf{c}_a}{\partial \mathbf{y}_a}(x, \mathbf{y}, t)\right)^T \cdot \boldsymbol{\lambda}_a(t) \end{bmatrix} = \begin{bmatrix} \mathbf{0}_D \\ \mathbf{0}_A \end{bmatrix}.$$

which defines an adjoint DAE

To ensure that the boundary term vanish we need to set $\boldsymbol{\mu} = \boldsymbol{\lambda}_{D0}(0)$ and $\boldsymbol{\lambda}_{D0}(\tau) = \mathbf{0}_N$. The algebraic component of $\boldsymbol{\lambda}$ at $t = \tau$ is also given by

$$\begin{aligned}
\mathbf{0}_A &= \left(\frac{\partial \mathbf{r}}{\partial \mathbf{y}_a}(x, \mathbf{y}, \tau)\right)^T \cdot \boldsymbol{\alpha} + 0 + \left(\frac{\partial \mathbf{c}_a}{\partial \mathbf{y}_a}(x, \mathbf{y}, \tau)\right)^T \cdot \boldsymbol{\lambda}_a(\tau) \\
\boldsymbol{\lambda}_a(\tau) &= -\left[\left(\frac{\partial \mathbf{c}_a}{\partial \mathbf{y}_a}(x, \mathbf{y}, \tau)\right)\right]^{-1} \cdot \left(\frac{\partial \mathbf{r}}{\partial \mathbf{y}_a}(x, \mathbf{y}, \tau)\right)^T \cdot \boldsymbol{\alpha},
\end{aligned}$$

where we recall our assumption that $\partial \mathbf{c}_a / \partial \mathbf{y}_a$ is square invertible.

Now we can ensure that the differential term vanishes by integrating this adjoint DAE backwards from the terminal condition $\boldsymbol{\lambda}(\tau) = (\boldsymbol{\lambda}^d(\tau), \boldsymbol{\lambda}^a(\tau))^T$ at $t = \tau$ to an initial condition at $t = 0$.

Once we have solved for $\boldsymbol{\lambda}(t)$ the reverse directional derivative reduces to

$$J^{\dagger}_{g\circ f}(x)(\boldsymbol{\alpha}) = -\left(\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}x_i}\right)^T \cdot \boldsymbol{\lambda}_{D0}(0) + \int_0^{\tau} \mathrm{d}t \left(\frac{\partial \mathbf{r}}{\partial x_i}(x, \boldsymbol{y}, t) + \boldsymbol{\lambda}^T \frac{\partial \mathbf{c}}{\partial x_i}(x, \boldsymbol{y}, t)\right)^T \cdot \boldsymbol{\lambda}(t).$$

A.5 Discussion

The implicit function theorem allows us to construct an expression for the directional derivatives of an implicit function as a composition of Fréchet derivatives. The adjoint method computes these directional derivatives directly without evaluating any of the intermediate terms.

When the output of the implicit function is finite-dimensional these differential operators can be implemented with linear algebra, although care and experience is required to ensure that the linear algebra operations are as efficient as possible. While the adjoint method yields the same result, it naturally incorporates any available structure in the implicit system so that optimal performance can be achieved automatically as we saw in our discussion on difference equations (Section A.3.3).

If the output of the implicit function is infinite-dimensional then the component Fréchet derivatives that make up the directional derivatives of the implicit function can no longer be evaluated directly, making the composition intractable. Implementing the adjoint method also requires Fréchet derivatives which in general we cannot evaluate. In the important special case where the output of the implicit function falls into a Sobolev space, however, we can engineer the augmented Lagrangian so that the Fréchet derivatives reduce to tractable functional derivatives. This is notably the strategy we deploy when in the case of ODEs and DAEs (Section A.4.3).

While the adjoint method is more generally applicable it is not as systematic as the implicit function theorem method. The practicality and performance of the method depends on the choice of Lagrangian and constraint functionals. Engineering performant functionals, let alone valid functionals at all, is by no means trivial. Fortunately in many problems the structure of the implicit system guides the design.

Beyond the implicit function theorem and adjoint methods, we may use the trace method which automatically differentiates through the trace of a numerical solver. In most cases this approach

leads to computationally expensive and memory intensive algorithms.

For finite-dimensional systems we could also construct a "forward method" that computes the Jacobian $J = J_{g \circ f}$ before evaluating its action on an input sensitivity or adjoint to form the wanted directional derivatives. As we discussed in Section A.3.2, however, fully computing J first is always less efficient that the iterative evaluation of the directional derivative; see also Gaebler (2021).

Although not as general, we can also construct a forward method that fully computes the Jacobian $J = J_{g \circ J_f}$ for certain infinite-dimensional problems. This approach notably applies to certain classes of ODEs and DAEs (Appendix A.7). In these cases the computational trade-offs between the forward method and the adjoint method is more nuanced; which method is more efficient depends on the specific of the problem. For example Rackauckas et al. (2018) compares the forward and adjoint approaches to implementing automatic differentiation for ODEs. For small ODE systems the overhead cost associated with solving the adjoint system can make the method relatively slow, but as the size of the system and the dimension of x increases the adjoint method benefits from superior scalability. See also Hindmarsh and Serban (2020) and Betancourt, Margossian, and Leos-Barajas (2020) for additional scaling discussions.

A.6 Acknowledgment

We thank David Childers, David Kent, and Dalton AR Sakthivadivel for helpful discussion.

A.7 Appendix: Infinite-Dimensional Forward Method

As in Section A.4.2 consider finite-dimensional real input, $X = \mathbb{R}^I$, an infinite-dimensional output space Y, and the summary function $g : Y \to \mathbb{R}^J$. The forward method explicitly computes the matrix representation of the full Jacobian, $\mathbf{J}_{g \circ f}$, before contracting this matrix with a sensitivity or adjoint vector to implement directional derivatives for automatic differentiation.

The implicit function theorem prescribes a composite expression for $\mathbf{J}_{g \circ f}$, but in general we cannot evaluate the intermediate Fréchet derivatives. In certain special cases, however, we can

bypass the implicit function theorem and evaluate $\mathbf{J}_{g \circ f}$ directly.

As with the adjoint method, we focus on the special case where Y is a Sobolev space. Theoretically we can reduce a key Fréchet derivative to a functional derivative, which we can evaluate by solving a *forward differential system*. In practice our ability to construct such a system *and* solve it depends on the specifics of the problem.

Let Y be the order K Sobolev space of functions $T \subset \mathbb{R} \to \mathbb{R}^N$, and suppose that we can construct a functional

$$\begin{aligned} \mathcal{P} : \quad X \times Y \quad &\to \mathbb{R}^J \\ x, \mathbf{y} \quad &\mapsto \mathcal{P}(x, \mathbf{y}), \end{aligned}$$

which satisfies

$$\frac{\delta \mathcal{P}}{\delta x}(x) = J_{g \circ f}(x).$$

In addition, assume there exists such a functional which takes the form of an integral,

$$\mathcal{P}(x, \mathbf{y}) = \int_T dt \; p(x, t, \mathbf{y}(t), \ldots).$$

For example if our summary function g is a Sobolev inner product with the implicit function,

$$\begin{aligned} (g \circ f)(x) &= \langle \gamma, \; f(x) \rangle \\ &= \int dt \; \gamma(t) \, f(x, t) \\ &= \int dt \; \boldsymbol{\gamma}^T(t) \cdot \mathbf{y}(x, t) \end{aligned}$$

then we can take

$$p = \boldsymbol{\gamma}^T(t) \cdot \mathbf{y}(x, t),$$

and obtain a satisfactory functional $\mathcal{P}$.

Taking the derivative with respect to x gives,

$$\begin{aligned}\frac{\delta \mathcal{P}}{\delta x} &= \int_T \mathrm{d}t \, \frac{\mathrm{d}p}{\mathrm{d}x}\left(x, t, \mathbf{y}(t), \mathbf{y}^{(1)}(t), \cdots, \mathbf{y}^{(K)}(t)\right) \\ &= \int_T \mathrm{d}t \, \frac{\partial p}{\partial x} + \sum_{k=0}^{K} \frac{\partial p}{\partial \mathbf{y}^{(k)}} \frac{\mathrm{d}\mathbf{y}^{(k)}}{\mathrm{d}x},\end{aligned}$$

where crucially the Fréchet derivative reduces to a functional derivative.

In order to evaluate this integral we need to evaluate the derivatives of the implicit function, $\mathrm{d}\mathbf{y}^{(k)}/\mathrm{d}x$, at all times $t \in T$. In theory we could achieve this by fully constructing the first-order Fréchet derivative from the implicit function theorem, repeatedly differentiating it, and then evaluating all of those Fréchet derivatives at each time t.

The need to evaluate Fréchet derivatives, however, makes this approach infeasible in practice. A more viable alternative is to evaluate the derivatives only at specific times, where they reduce to manageable finite-dimensional objects.

By definition our constraint function defines a map $c : X \times Y \to Z$ where Y and Z are both the space of functions which map from T to $\mathbb{R}^N$. In this case the constraint function can equivalently be defined as a collection of maps $X \times Y \to \mathbb{R}^N$ for each $t \in T$. Denoting these maps as $c(x, y, t)$ the implicit function $f : X \to Y$ is defined by the system of constraints,

$$c(x, f(x), t) = 0, \, \forall \, t \in T.$$

Fixing t and then differentiating with respect to the input x gives

$$\begin{aligned}0 &= \frac{\partial c}{\partial x}(t) + \sum_{k=0}^{K} \frac{\partial c}{\partial \mathbf{y}^{(k)}} \frac{\mathrm{d}\mathbf{y}^{(k)}}{\mathrm{d}x}(t) \\ &= \frac{\partial c}{\partial x}(t) + \sum_{k=0}^{K} \frac{\partial c}{\partial \mathbf{y}^{(k)}} \left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x}\right)^{(k)}(t).\end{aligned}$$

This *forward differential system* implicitly defines the derivative evaluations at each t as a differential equation. Once we've solved for $\mathbf{y}(t)$ we can, at least in theory, solve this forward differential

system for each $\mathrm{d}\mathbf{y}^{(k)}/\mathrm{d}x\,(x, f(x), t)$ and then evaluate

$$\int_T \mathrm{d}t \sum_{k=0}^{K} \frac{\partial p}{\partial \mathbf{y}^{(k)}} \frac{\mathrm{d}\mathbf{y}^{(k)}}{\mathrm{d}x} = \sum_{k=0}^{K} \int_T \mathrm{d}t \frac{\partial p}{\partial \mathbf{y}^{(k)}} \frac{\mathrm{d}\mathbf{y}^{(k)}}{\mathrm{d}x},$$

as a sum of one-dimensional numerical quadratures.

If $c(x, y, t)$ is a linear a function of $\mathrm{d}y/\mathrm{d}t$ and does not depend on higher-order derivatives,

$$c(\mathbf{y}, x, t) = \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t}(t) - f(x, \mathbf{y}, t) = 0,$$

then the forward differential system becomes particularly manageable. In particular the forward differential system is also linear in the first-order derivative with respect to t,

$$\frac{\mathrm{d}c}{\mathrm{d}x}(\mathbf{y}, x, t) = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x}(t) - \frac{\partial f}{\partial x}(x, \mathbf{y}, t) - \frac{\partial f}{\partial \mathbf{y}}\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t}(t) = 0.$$

In the absence of such a linearity we need to solve for the evaluations of the trajectory $\mathbf{y}$, the first-order derivative $\mathrm{d}\mathbf{y}/\mathrm{d}x$, and the higher-order derivatives of $\mathbf{y}$ at the given x and *every* t needed for the numerical quadratures.

For a demonstration of this forward approach on certain ODEs and DAEs see (Hindmarsh and Serban 2020).

Finally similar to the Sobolev adjoint method (Section A.4.2) the above derivation generalizes immediately to constrained systems over Sobolev spaces of functions $T \subset \mathbb{R}^M \to \mathbb{R}^N$, with $M > 1$. Here instead of an ordinary differential forward system we recover a partial differential forward system, and the Jacobian is recovered as a sum of multidimensional integrals instead of one-dimensional integrals.