6.885 Final Project: Inferring Invisible Force Fields using Gaussian Processes

James Gilles

12 May 2020

1 Abstract

We specify a challenging inference problem involving non-observable force fields affecting particles moving in a 2d plane. We implement and evaluate a baseline solution using Gaussian Processes in Gen (Cusumano-Towner et al., 2019), and outline directions for future work.

Our code is available at: `https://github.com/kazimuth/6.885`

2 Introduction

There's recently been a surge of interest in using sparsity to accelerate differentiable algorithms, in particular physical simulations. Projects like the Tiramisu compiler (Baghdadi et al., 2018) have successfully leveraged sparsity for large speedups running deep neural networks on GPU. Similarly, the Taichi programming language (Hu et al., 2019) allows spatially sparse, differentiable physical simulations to be quickly prototyped to run on GPUs.

We believe that tools like these could also be used to accelerate algorithms based on a Bayesian framework. In fact, our initial plan for the project was to attempt to implement Taichi with Gen (Cusumano-Towner et al., 2019), an innovative probabilistic programming environment.

However, we realized that, before beginning porting efforts, it would be useful to have benchmark problems and sample implementations, for the sake of ensuring correctness and comparing performance. So, for this project, we set out to create a small, self-contained benchmark problem using Bayesian inference, which nevertheless exhibits enough complexity to be interesting.

Our benchmark problem involves a set of particles moving on a 2d plane, affected by an externally-imposed static force field. The force field is modeled as a pair of independent Gaussian Processes, one for the x and one for the y component of the field. Given noisy observations of particle positions over a period of time, we use Metropolis-Hastings Monte-Carlo with a custom proposal distribution to infer the values taken by the force field.

3 Related work

Gen (Cusumano-Towner et al., 2019) is a probabilistic programming environment integrated with the Julia programming language (Bezanson et al., 2012). We implement our baseline solution using a combination of Gen and other Julia libraries.

Our project was inspired by work like Taichi (Hu et al., 2019) and Tiramisu (Baghdadi et al., 2018), both of which connect to larger body of work starting with Halide (Ragan-Kelley et al., 2017). All of these projects involve a separation between the specification of an algorithm and its implementation, allowing the data structures and control flow of the program to be modified while still conforming to a high-level specification.

Our work models a force field using a pair of 2d Gaussian Processes (Rasmussen, 2004). There is a large body of work on methods for sparsely sampling gaussian processes (Bauer et al. (2016) Yang et al. (2019) Quiñonero-candela et al. (2005)). We reviewed some of this literature, but have not yet implemented it in our model.

We had originally intended to use Hamiltonian Monte Carlo (Duane et al., 1987) as part of our inference algorithm, but unfortunately we weren't able to get it working in time.

Our custom proposal distribution requires numerical differentiation of noisy data. We reviewed Chartrand (2011) for suggestions as to how to reduce noise amplified by differentiation, although we did not use their full algorithm, leaving that for future work.

4 Problem specification

We select a rectangular region of interest in the real plane, $R = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \subset \mathbb{R}^2$, and a span of time $[t_0, t_1] \subset \mathbb{R}$. There are n particles in the region, moving freely. Each particle has a mass m_i, a position $\mathbf{x}_i \in R$, and a velocity $\dot{\mathbf{x}}_i$. A particle's velocity is affected by the force within its grid cell $\mathbf{F}$, and by a drag force which opposes velocity, parameterized by a

variable k:

$$\ddot{\mathbf{x}}_i(t) = \mathbf{F}(\mathbf{x}(t))/m_i - (1-k)\dot{\mathbf{x}}$$

Particles collide elastically with the boundary of the region, retaining their velocities in the opposite direction. We select initial positions uniformly from the region and set initial velocities to 0:

$$\dot{\mathbf{x}}_i(t_0) = 0$$

$$\mathbf{x}_i(t_0) \sim U(R)$$

We model the force as a pair of independent gaussian processes, modeling the x and y components of the force separately. These forces are sampled at an even grid $\mathbf{g}$ of points of resolution x_{res} by y_{res}.

The gaussian processes are parameterized by an exponentially decaying covariance kernel with a length scale ℓ and noise parameter σ:

$$\begin{aligned}
f(\mathbf{a}, \mathbf{b}) &= \mathrm{e}^{-\frac{1}{2}\ell\,||\mathbf{a}-\mathbf{b}||_2} \\
\Sigma_{ij} &= f(\mathbf{g}_i, \mathbf{g}_j) + \begin{cases} \sigma & i = j \\ 0 & \text{otherwise} \end{cases} \\
F_x(\mathbf{g}_i) &\sim \mathcal{N}(0, \Sigma) \\
F_y(\mathbf{g}_i) &\sim \mathcal{N}(0, \Sigma) \\
F(\mathbf{g}_i) &= \begin{bmatrix} F_x(\mathbf{g}_i) \\ F_y(\mathbf{g}_i) \end{bmatrix}
\end{aligned}$$

(where $||\mathbf{a}||_2$ denotes the ℓ_2 norm.)

We place priors on the length scale and noise, using gamma distributions with set parameters:

$$\begin{aligned}
\ell &\sim \Gamma(1, \ell_{\text{scale}}) + \ell_{\min} \\
\sigma &\sim \Gamma(1, \sigma_{\text{scale}}) + \sigma_{\min}
\end{aligned}$$

We discretize time into T timesteps of length Δt, and approximate positions and velocities using simple Euler integration:

$$\begin{aligned}
\dot{\mathbf{x}}_i[t+1] &= k(\dot{\mathbf{x}}_i[t] + \mathbf{F}(\mathbf{x}_i[t])\Delta t) \\
\mathbf{x}_i[t+1] &= \mathbf{x}_i[t] + \dot{\mathbf{x}}_i[t]\Delta t
\end{aligned}$$

The force affecting each particle is chosen by nearest-neighbor sampling from the grid.

$$\mathbf{F}(\mathbf{x}) = \mathbf{F}(\mathbf{x}_g)$$
$$\mathbf{x}_g = \arg\min_{\mathbf{x}_g \in \mathbf{g}} ||\mathbf{x} - \mathbf{x}_g||_2$$

At each time step, we observe particle positions with a noise factor o_{scale}:

$$\mathbf{o}_x \sim \mathcal{N}(\mathbf{x}_x, o_{\text{scale}})$$

$$\mathbf{o}_y \sim \mathcal{N}(\mathbf{x}_y, o_{\text{scale}})$$

Note that, once forces and initial positions are chosen, the simulation is deterministic! It is only the observations of particle positions that are noisy.

In this setting, we pose the problem: given a vector of observations $\mathbf{O}$ (and potentially observations of other parameters, such as ℓ, σ, or) sample from the posterior distribution $\text{p}(\mathbf{F}(\mathbf{g}) \mid \mathbf{O})$.

In our implementation, we choose the following values for free parameters:

parameter	value
(x_{min}, x_{max})	(0.0, 1.0)
(y_{min}, y_{max})	(0.0, 1.0)
x_{res}	10
y_{res}	10
ℓ_{scale}	0.1
ℓ_{min}	0.01
σ_{scale}	0.1
σ_{min}	0.01
o_{scale}	0.002
k	0.9
n	10

However, these are easy to change.

5 Implementation & Results

To start, we implemented and tested a deterministic simulator for the problem, together with static and animated visualizations. Given a force grid and starting positions, our simulator could show us the particle's paths. We show static visualizations here; see the source code to view animated visualizations.

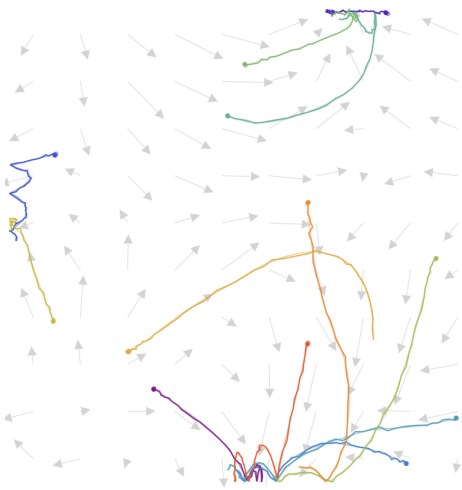

Figure 1: A static visualization of particle paths. Circles represent particle starting points. Gray arrows represent the force grid and are scaled by Δt; that is, they correspond to the amount a particle's velocity will change in one time step. Note the small amount of noise in the observations of the paths. Also note particles bouncing off the boundary.

Then, we implemented metropolis-hastings MCMC, with all values observed except for the forces on the grid. This turned out to be barely functional! For more than a few particles, it took an incredibly long time to converge. We realized that this was because it had to guess the entire field essentially a priori. The algorithm was not informed by the observed accelerations applied to the particles. So the vast majority of time, transitions would be rejected, because the particle position traces they generated were too implausible.

We tried accelerating the algorithm by switching to Gen's static modeling language. This gave very good speedups, but didn't address the inefficiency in the algorithm; it was still slow to converge.

Using hamiltonian monte-carlo similarly failed. It's possible this is because gradients were not being propagated well through the body of the our sampling algorithm – it was somewhat difficult to debug the behavior of gradients.

We realized that we had a lot of information we weren't leveraging in the algorithm. At every time step, we can measure (noisy) changes in a particle's velocity. This should tell us the approximate force value of the grid cell containing the particle. We could use this information by building

a custom proposal distribution for the metropolis-hastings transition step, which Gen makes extremely easy.

Of course, measured forces would have some variance; so we would treat them as Gaussians when sampling. Particles might not cross all cells, however. Luckily, we could sample very nice values for unobserved cells using Gaussian Process regression.

We implemented this algorithm... and it still didn't work. We tried visualizing the resulting proposals, and noticed that they were **much noiser** than the original prior, rather than being less noisy! This turned out to be a result of the differentiation algorithm we were using. Naive second differences turns out to massively magnify noise (Chartrand, 2011). We surveyed denoising algorithms, and decided to smooth the observed accelerations using linear least squares. We fit a cubic function to the x and y components of the observed acceleration traces, and this gave satisfactory results. (See Figure 2 for a visualization of this smoothing.)

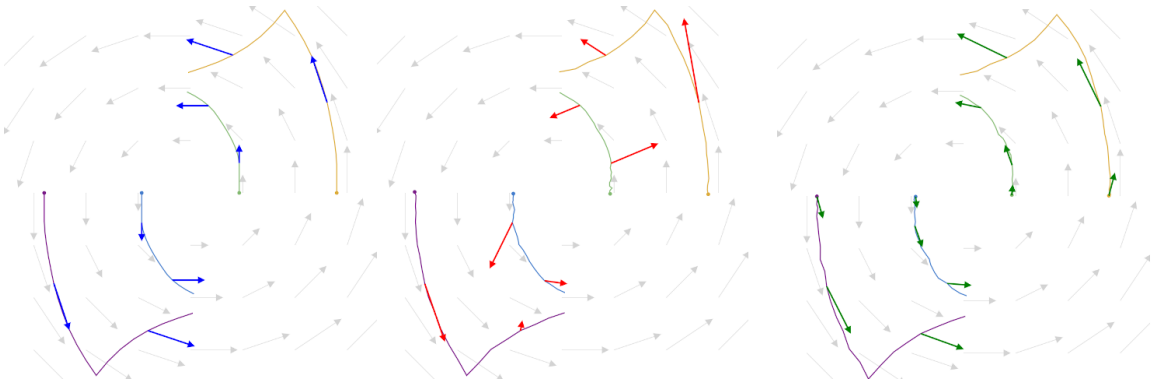

Figure 2: Second differences of observed paths. Blue arrows show true accelerations at selected timesteps. Red arrows show accelerations computed from noisy observations using second differences. Note the high variance! Green arrows show smoothed second differences. Observe how they are much closer to the true values of the grid cells close to them.

This greatly improved convergence times, and allowed us to infer force fields with many particles (20-30).

(One caveat of this algorithm was that it was significantly complicated by boundary behavior. We ended up having to observe whether particles bounced off the boundary without any noise, in order to properly understand the boundaries... we leave lifting this restriction to future work.)

We then implemented heirarchical monte carlo, attempting to infer the length scale of the GP kernel and particle initial positions. We were able to get this working by implementing a custom proposal distribution for initial particles, and by moving the length scale proposal into our custom proposal. (Otherwise alternative length scales would never be selected, because they would resample the field without our proposal's knowledge.)

Our algorithm often converges to reasonable approximations of the force field. See Figure 3 for examples of its outputs.

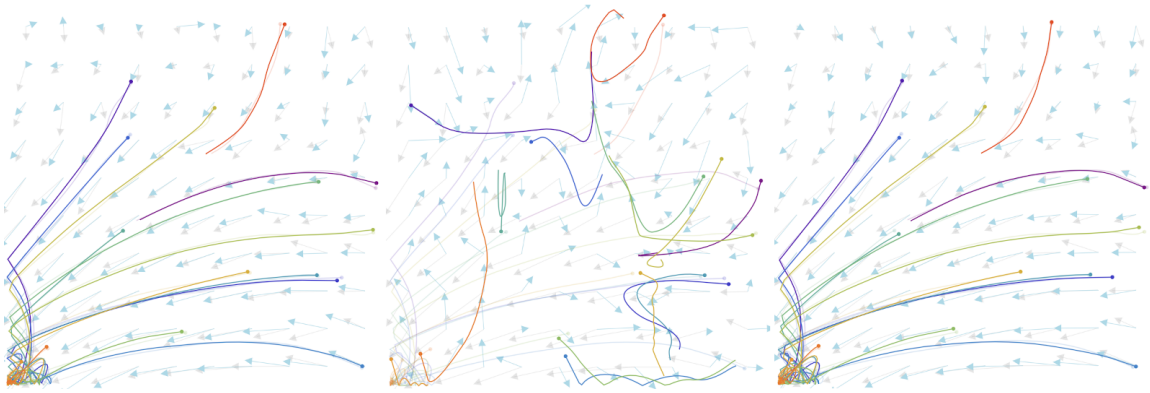

Figure 3: Samples from the posterior distribution after 500 steps of metropolis-hastings. True starting positions and paths are drawn lightly; the inferred force field is drawn lightly, and paths the particles would have taken if placed in it are drawn darkly. Note that the middle case failed to converge in the time limit.

6 Future Work

In the future, we'd like to try using smarter denoising algorithms, like that proposed in Chartrand (2011); or, alternatively, neural denoising.

We'd also like to try implementing a sparse GP approximation, possibly using Taichi, and compare its performance with our implementation.

References

Riyadh Baghdadi, Jessica Ray, Malek Ben Romdhane, Emanuele Del Sozzo, Abdurrahman Akkas, Yunming Zhang, Patricia Suriana, Shoaib Kamil,

and Saman Amarasinghe. Tiramisu: a polyhedral compiler for expressing fast and portable code. *CoRR*, 2018. URL http://arxiv.org/abs/1804.10694v5.

Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen. Understanding probabilistic sparse gaussian process approximations. *CoRR*, 2016. URL http://arxiv.org/abs/1606.04820v2.

Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Julia: a fast dynamic language for technical computing. *CoRR*, 2012. URL http://arxiv.org/abs/1209.5145v1.

Rick Chartrand. Numerical differentiation of noisy, nonsmooth data. *ISRN Applied Mathematics*, 2011(nil):1–11, 2011. doi: 10.5402/2011/164564. URL https://doi.org/10.5402/2011/164564.

Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pp. 221–236, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6712-7. doi: 10.1145/3314221.3314642. URL http://doi.acm.org/10.1145/3314221.3314642.

Simon Duane, A.D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216–222, 1987. doi: 10.1016/0370-2693(87)91197-x. URL https://doi.org/10.1016/0370-2693(87)91197-x.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi. *ACM Transactions on Graphics*, 38(6):1–16, 2019. doi: 10.1145/3355089.3356506. URL https://doi.org/10.1145/3355089.3356506.

Joaquin Quiñonero-candela, Carl Edward Rasmussen, and Ralf Herbrich. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:2005, 2005.

Jonathan Ragan-Kelley, Andrew Adams, Dillon Sharlet, Connelly Barnes, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. Halide. *Communications of the ACM*, 61(1):106–115, 2017. doi: 10.1145/3150211. URL https://doi.org/10.1145/3150211.

Carl Edward Rasmussen. *Gaussian Processes in Machine Learning*, pp. 63–71. Advanced Lectures on Machine Learning. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-28650-9_4. URL https://doi.org/10.1007/978-3-540-28650-9_4.

Ang Yang, Cheng Li, Santu Rana, Sunil Gupta, and Svetha Venkatesh. Sparse spectrum gaussian process for bayesian optimisation. *CoRR*, 2019. URL http://arxiv.org/abs/1906.08898v1.