http://www.math.nyu.edu/faculty/goodman/teaching/MonteCarlo20/

Class notes: Monte Carlo methods
Week 10, Affine invariant samplers, simulated tempering
tentative Jonathan Goodman
November 18, 2020

1 Affine invariant samplers

An *affine transformation* is a function change of variables

$$y = Ax + b \;, \quad x(\in \mathbb{R}^d) \longrightarrow y(\in \mathbb{R}^d) \;. \tag{1}$$

[Terminology: a function with $f(x_1 + ax_2) = f(x_1) + af(x_2)$ is *linear.* In finite dimensions, a linear map has the form $f(x) = Ax$, for some matrix A. The map (1) is linear only when $b = 0$. Otherwise it is *affine.* Linear transformations suffice for everything in today's class. We will always have $b = 0$. In this context, even strictly linear transformations are called affine.] Affine transformations are used in computing as *pre-conditioners* to make numerical algorithms work better. There are situations where a good pre-conditioner makes a big difference, but such a pre-conditioner is hard to find and depend sensitively on problem specific details.

An *affine invariant* algorithm, informally, is one that does the same thing with or without an affine (linear usually) pre-conditioning. You can think of affine invariant methods as automatically working in the best set of variables that can be achieved by a linear transformation. Affine invariant algorithms are more likely to work well "out of the box" without problem-specific tuning. Affine invariant MCMC algorithms are the basis of some popular software packages for sampling, particularly in Bayesian statistics.

There is an analogy between sampling and optimization. Both problems involve exploring an energy surface given by a function $\phi(x)$, either to sample the distribution $\frac{1}{Z}e^{-\phi(x)}$ or to find x values with small ϕ. Optimization is a simpler context to explain ill-conditioning and the power of an affine invariant algorithm.

A basic optimization algorithm is *gradient descent*, which is the iteration

$$x_{k+1} = x_k - s_k \nabla\phi(x_k) \;. \tag{2}$$

The number s_k is the *step size* or *learning rate.* Suppose you have a *current iterate*, x_k and want to look in some direction in $\mathbb{R}^d$ for a better iterate. We call this the *search direction*, p. For gradient descent, the search direction is

$$p_k = -\nabla\phi(x_k) \;. \tag{3}$$

and the step is

$$x_{k+1} = x_k + s_k p_k \;.$$

The direction (3) at least is a direction determined by the problem, if $\nabla\phi(x) \neq 0$. It is a *descent direction* in the sense that ϕ is decreasing in that direction. This means that if the step size is small enough, then ϕ decreases.

$$\frac{d}{ds}\phi(x+xp)\Big|_{s=0} < 0 \implies \phi(x+sp) < \phi(x) \;, \;\; \text{if } s \text{ is small enough .}$$

The derivative on the left is calculated using the chain rule, in vector notation, as (assuming $\nabla\phi(x) \neq 0$)

$$[\nabla\phi(x)]^t\, p = [\nabla\phi(x)]^t\, \nabla\phi(x) = -\left\|\nabla\phi(x)\right\|^2 < 0 \;.$$

This allows you to prove the following

Theorem. (Proof not given, but a proof oriented person could do it) Suppose ϕ is is twice differentiable and the learning rate parameters $s_k > 0$ satisfy the two conditions

$$\max_k s_k \text{ is small enough ,} \tag{4}$$

$$\sum_{k=1}^{\infty} s_k = \infty \;. \tag{5}$$

Then at least one of the following things happens

- $\|x_k\| \to \infty$ as $k \to \infty$
- $\lim\limits_{k\to\infty} x_k = x_*$ exists, and $\nabla\phi(x_*) = 0$.

The first condition (4) is natural if you want x_k to have a limit, but how small is "small enough" depends on ϕ. If the second condition is violated than x_k can converge to x_* with $\nabla(x_*) \neq 0$. An example is $d = 1$ and $\phi(x) = x$.

Gradient descent can work poorly for optimization problems where ϕ is *ill conditioned*. In this context, "ill conditioned" just means that gradient descent works better after pre-conditioning. In other scientific computing contexts, "ill conditioned" means that the answer depends so sensitively on the data that it is hard to compute in floating point arithmetic. These senses are related in the sense that if $\phi(x) = x^t Ax$, and the linear algebra problem: "find x so that $Ax = b$" is ill conditioned in the numerical computing sense, then ϕ is ill conditioned for gradient descent. For example, consider

$$\phi(x_1, x_2) = \frac{1}{2}\left(x_1^2 + \epsilon x_2^2\right) \;. \tag{6}$$

Then

$$p = -\nabla\phi(x) = (-x_1, -\epsilon x_2) \;.$$

If you move in the direction of p, then x_2 changes little. If you adjust s so that x_2 changes a lot, then x_1 changes too much. Another example is

$$\phi(y_1, y_2) = (y_1 + y_1)^2 + \epsilon(y_1 - y_2)^2 \;. \tag{7}$$

These are equivalent, via the orthogonal transformation

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \ . \tag{8}$$

Dimensional analysis is a quick way to get some disrespect for gradient descent. We write in components as

$$x_{j,k+1} = x_{j,k} - p_k \frac{\partial \phi(x_k)}{\partial x_j} \ .$$

In a Bayesian statistics problem such as the amplitude and frequency problem from Week 9, different components of x have different units. In that case, some are amplitudes and some are frequencies. Some of the components have the foem

$$\text{new amplitude } = \text{ old amplitude } + s \frac{\partial \phi}{\partial \text{ amplitude}}$$

This makes dimensional sense if s has units

$$[s] = \frac{[\text{amplitude}]^2}{[\phi]} \ .$$

The frequency components make dimensional sense if s has units

$$[s] = \frac{[\text{frequency}]^2}{[\phi]} \ .$$

These cannot both be true.

An optimization code using gradient descent might ask the user to use units for the different components x_j so that the same step size makes sense for all components. Mathematically, changing units on component x_j mean multiplying x_j by a unit conversion factor (for example, 100 centimeters in a meter). That is the same as multiplying x by a diagonal matrix with the scaling factors. For the problem (6), the scaling matrix could be

$$A = \begin{pmatrix} 1 & 0 \\ 0 & \epsilon^{-\frac{1}{2}} \end{pmatrix} \ .$$

It might take a user some time to do a good job with this, as the natural sizes of component x_j might depend not only on the problem class (amplitudes and frequencies), but on the details of a specific problem (how well a frequency is determined by the data).

The example (7) illustrates another form of ill conditioning that arises in statistics. Here, ϕ is strongly effected by the sum $y_1 + y_2$ but depends weakly on the difference $y_1 - y_2$. This kind of thing happens when $y_1 + y_2$ is strongly constrained by the data but $y_1 - y_2$ is not. For example, suppose we have frequencies ω_1 and ω_2 that are nearly the same. Then $A_1 \cos(\omega_1 t) + A_2 \cos(\omega_2 t)$ depends more strongly on $A_1 + A_2$ than on the difference. This depends on the relation between ω_1 and ω_2, which can change from problem to problem.

To summarize, gradient descent can work poorly for ill conditioned problems. An affine change of variables can turn an ill conditioned problem into one that is well conditioned (not ill conditioned). Affine pre-conditioning is necessary and difficult for gradient descent to work well in practice.

Newton's method is an optimization algorithm that uses the Hessian matrix

$$H_{ij}(x) = \partial_{x_i}\partial_{x_j}\phi(x) \ .$$

The search direction is

$$p = H(x)^{-1}\nabla\phi(x) \ .$$

This is affine invariant in the following sense. Suppose $y = Ax$ is a linear transformation and $\psi(x) = \phi(Ax)$. Suppose $y_k = Ax_k$, Suppose you move y by applying Newton's method to ϕ and you move x by applying Newton's method to ψ. Then $y_{k+1} = Ax_{k+1}$. The proof is Exercise 1.

In the "optimization community" (people who spend their days developing optimization algorithms), it is widely understood that Newton's method is better than gradient descent for general problems that optimization software would be used for. This is explained by the true fact that Newton's method had faster local convergence. Once x_k is close enough to x_*, the convergence is very fast. This is a nice property (local quadratic convergence), but I think the power of Newton's method for general problems is that it is affine invariant.

An affine invariant sampler would have the property that if $\rho(y) = \frac{1}{Z}e^{-\phi(y)}$ and if $\sigma(x) = \frac{1}{Z}e^{-\phi(Ax)}$, and if the sampler has the form $Y_{k+1} = F(Y_k, \xi_k, \phi)$, where ξ_k is independent of the problem and i.i.d., and if $X_{k+1} = F(X_k, \xi_k, \psi)$ (still taking $\psi(x) = \phi(Ax)$), then $X_{k+1} = AY_{k+1}$. Affine invariant samplers can lead to MCMC software that is quite robust, even for problems that have not been "tuned" (pre-conditioned).

There seem to be two ways to create affine invariant samplers. One is *ensemble samplers*. The other is samplers that use derivative information.

An *ensemble sampler* moves an ensemble ("ensemble" is "set" in French) of samples, called *walkers*. An ensemble of size L has walkers $x_1, \cdots, x_L$. The ensemble is $E = (X_1, \cdots, X_L)$. The target density for E is the distribution in which the walkers are independent samples of the target density for the walkers, $\rho(x)$. That is, we seek

$$E \sim f(e) = \prod_{i=1}^{L} \rho(x_i) \ . \tag{9}$$

Clearly, if E is in its target distribution, then the individual walkers are in the target distribution. The dimension of "ensemble space" is Ld, where L is the ensemble size and d is the number of components of x.

A sampler could be affine invariant if it used the covariance matrix of the target distribution to determine proposals. An affine change of variables then would change the covariance matrix, which would change the proposal distribution. This would allow the proposal distribution to be aware of the conditioning of the problem, and take big steps in directions where the covariance is large (loosely constrained) but small steps in directions that are more tightly constrained. It is impossible to know the covariance in advance, but the empirical covariance matrix of the walkers in the ensemble are an estimator of the true covariance.

Ensemble samplers typically work in Gibbs sampler mode – moving one walker at a time leaving the other walkers fixed. We will call X_i the walker being moved and $E_i^c = \{X_j \mid j \neq i\}$ the *complementary ensemble*. Ensemble samplers rely on the observation that it is possible to maintain detailed balance in ensemble space for the target distribution (9) even when the move of X_i depends on the complementary ensemble. The argument is the same as for the Gibbs sampler. Suppose we fix the walkers in the complementary ensemble and move X_i to X_i' that has density

$$X_i' \sim R(x_i' \mid X_i, E_i^c) \ .$$

This satisfies detailed balance for component i if

$$\rho(x_i)\, R(x_i' \mid x_i, E_i^c) = \rho(x_i')\, R(x_i \mid x_i', E_i^c) \ .$$

This is the same as before, but conditioned on the complementary ensemble:

$$\Pr(\text{ observe } x_i \to x_i' \mid E_i^c) = \Pr(\text{ observe } x_i' \to x_i \mid E_i^c) \ .$$

You can see that this condition implies detailed balance in ensemble space. If E' is the same as E except with X_i' instead of X_i, then

$$\Pr(\text{ observe } E \to E') = \Pr(\text{ observe } E' \to E) \ . \tag{10}$$

I will come back to this after talking about some specific ensemble moves.

These moves really are proposal distributions for a proposal $X_i \to Y_i$. If the proposal is accepted, then $X_i' = Y_i$. Otherwise $X_i' = X_i$. The *walk move* is defined by the empirical covariance matrix of the complementary ensemble. The complementary mean is

$$\overline{X}_i^c = \frac{1}{L-1} \sum_{j \neq i} X_j \ .$$

The complementary covariance matrix is

$$C(E_i') = \frac{1}{L-1} \sum_{j \neq i} \left(X_j - \overline{X}_i^c \right) \left(X_j - \overline{X}_i^c \right)^t \ .$$

The move proposes

$$Y \sim \mathcal{N}(X_i, r^2 C(E_i')) \ .$$

Here r is a proposal size parameter. You tune the sampler by choosing a good r. This is not easy, but it is easier than trying to find a covariance matrix for the proposal distribution. It is one number, not a $d \times d$ matrix with about $\frac{1}{2}d^2$ parameters. Moreover, being affine invariant, it will have a reasonable order of magnitude for r even without tuning.

2 Exercises

1. Formulate an equation that says Newton's method for optimization is affine invariant and verify that it is true.

2. Show that if you use an affine invariant sampler, then the auto-correlation function for any observable is invariant under affine transformations of the variables. Note, this is not necessarily true of the auto-covariance function because these might involve the scale factors.