Scientific Computing, Courant Institute, Fall 2018

http://www.math.nyu.edu/faculty/goodman/teaching/ScientificComputing2018/ScientificComputing.html

Always check the `classes` message board before doing any work on the assignment.

Assignment 5

Corrections: [none yet]

1. (*Gradient descent and condition number*) The gradient descent algorithm for minimizing $V(x)$ is

$$x_{n+1} = x_n - t_n \nabla V(x_n) \ .$$

The positive number t_n is the step size, or *learning rate* for iteration n. For this exercise, assume a constant learning rate, which is $t_n = t$ for all n. The local convergence of gradient descent to a local minimum is determined by the local behavior of V near the local minimum, x_*. Suppose V is smooth near x_* and the Hessian $H(x_*)$ is positive definite. Then the local convergence behavior is determined by H (you don't have to show this). Consider a function in n dimensions $V(x) = \frac{1}{2} x^t H x - x^t b$, for some positive definite H and vector $b \in \mathbb{R}^n$.

   (a) Show that V is convex and the unique global minimum satisfies $Hx_* = b$.

   (b) Show that the Hessian of V is constant.

   (c) Suppose that the eigenvalues of H are $\lambda_n \geq \lambda_{n-1} \geq \cdots \geq \lambda_1 > 0$.

   (d) Show that when analyzing convergence for this V, we may assume $b = 0$ and $x_* = 0$. Hint: let $y_n = x_n - x_*$ and find y_{n+1} in terms of y_n.

   (e) Show that gradient descent converges for any initial guess if and only if $|1 - t\lambda_n| < 1$ and $|1 - t\lambda_1| < 1$.

   (f) Show that

$$\|x_n - x_*\|_2 \sim C r^n \ \text{ as } \ n \to \infty$$

   for generic initial guess if

$$r = \max_k |1 - t\lambda_k| \ .$$

   and $r < 1$. (Note: part (c) is a consequence of this).

   (g) Find a formula for the learning rate (t) that gives the optimal generic convergence rate (r).

   (h) Show that condition number of H (in the norm $\|\cdot\|_2$) satisfies

$$\kappa_2(H) = \|H\|_2 \left\|H^{-1}\right\|_2 = \frac{\lambda_n}{\lambda_1} \ .$$

(i) Suppose that $\kappa_2(H) \gg 1$ (that is, H is ill conditioned). Show in this case that the optimal generic convergence rate (with the optimal learning rate) satisfies

$$r_{\text{opt}} \approx 1 - \frac{2}{\kappa_2(H)} \ .$$

(j) Estimate the number of gradient descent iterations required to reduce $\|x - x_*\|_2$ by a factor of e as a function of $\kappa_2(H)$ if H is ill conditioned. This shows that gradient descent is slow for ill conditioned problems even with the optimal parameters.

(k) Suppose A is an $n \times n$ invertible matrix, and that $W(y) = V(Ay)$. Consider gradient descent applied to W with the optimal learning rate for that problem. Show that if A is well chosen, then the W problem may have much faster convergence than the V problem. The matrix A is called a *preconditioner*.

2. It is commonly said that gradient descent is globally convergent if the learning rate is small enough, and if V is smooth and strictly convex, and if there is a minimum. Show that this is not true. Hint: try $V(x) = e^x + e^{-x}$.

3. (*Affine invariance*) If $x \in \mathbb{R}^n$, a transformation of the form $x = Ay + b$ is called *affine*. For this exercise, b is irrelevant and we take $b = 0$, but we retain the terminology *affine*. Consider Newton's method for optimizing $V(x)$ without safeguards. Call the iterates x_n. They satisfy $x_{n+1} = x_n - H_V(x_n)^{-1}\nabla V(x_n)$. Suppose $W(y) = V(Ay)$ and we apply Newton's method to W, resulting in iterates y_n. Show that if $x_n = Ay_n$ then $x_{n+1} = Ay_{n+1}$. That is, Newton's method is *affine invariant*. Affine invariance means that an affine preconditioner does not change the convergence behavior or convergence rate.

4. (*Gauss-Newton*) The nonlinear least squares problem is to find $x \in \mathbb{R}^N$ to minimize

$$V(x) = \sum_{j=1}^{M} \left(D_j - f_j(x)\right)^2 \ .$$

Problems like this come up in data fitting. You have M measured data values D_j and a model that predicts $d_j = f_j(x)$. The model cannot fit all the data because the model is not exactly correct and because the data are not measured exactly. If the model, the functions $f_j(x)$, is nonlinear, then the least squares criterion becomes non-linear. The functions f_j form the components of $f(x) \in \mathbb{R}^M$. The data values D_j form the components of $D \in \mathbb{R}^M$. The objective function may be written in vector notation as

$$V(x) = \|D - f(x)\|_2^2 \ .$$

If $N = M$, we can hope to set all the residuals $D_j - f(x_j)$ to zero and solve the nonlinear equations $D = f(x)$. If $M > N$ (more data than

parameters), in general it is impossible to set all the residuals to zero. Instead, the objective function is to minimize the residuals in the least squares sense.

The derivatives of f are

$$A_{jk} = \frac{\partial f_j}{\partial x_k} \ , \quad B_{jkl} = \frac{\partial^2 f_j}{\partial x_k \partial x_l}$$

Here, A is the Jacobian matrix $A(x) = f'(x)$, and $B(x)$ is a three index array of second partials written informally as $B(x) = f''(x)$. The *linearization* of the model f about a point x is the linear approximation

$$f(x+y) \approx f(x) + A(x)(y-x) \ .$$

The *Gauss Newton* method is an iteration $x_n \to x_{n+1}$ where we find x_{n+1} by solving the linearized least squares problem, $x_{n+1} = x_n + y$, where y satisfies

$$\min_y \|D - (f(x_n) + A(x_n)(y - x_n)\|_2^2 \ .$$

Recall that Newton's method for solving the nonlinear equations uses the linearization to get x_{n+1} from x_n and has local quadratic convergence. Assume that $M > N$ and that $A(x)$ has full rank N at every x.

(a) Find a formula for $H(x) = V''(x)$ and show that the Gauss-Newton method for nonlinear least squares is not the same as Newton's method for minimizing V.

(b) Show that the Gauss-Newton method produces a descent direction – the M it uses in place of the Hessian is positive definite.

(c) Show that the Gauss-Newton iteration converges (if it converges) linearly rather than quadratically.

(d) Show that the linear convergence rate improves as the residual at the solution decreases.

5. Write a Python procedure to compute the modified Cholesky factorization of a symmetric H. Your code should use Python vector instructions (rather than scalar loops) for the inner loops as much as possible. You should call it using `L,p = mCh(H)`, where H is a Numpy array of the $n \times n$ symmetric matrix H. It should return a tuple `L, P`, where L is the $n \times n$ lower triangular factor and `P` is a logical variable which is `true` if H was positive definite and `false` otherwise. Test your procedure on several matrices of your choice including some that are positive definite and some that are not. Calculate an example that you can use to see that the code produces the correct answer in both cases.