Scientific Computing, Fall 2024
http://www.math.nyu.edu/faculty/goodman/teaching/SciComp2024/index.html

Assignment 7

1. A *nonlinear iteration* is function F with d arguments and d components.

$$F(x) = \begin{pmatrix} F_1(x_1, \cdots, x_d) \\ \vdots \\ F_d(x_1, \cdots, x_d) \end{pmatrix} .$$

We write $F : \mathbb{R}^d \to \mathbb{R}^d$ and say: "F takes $\mathbb{R}^d$ to $\mathbb{R}^d$." Suppose x_* is a *fixed point*, which means $F(x_*) = x_*$ and suppose F is differentiable near to x_*. The fixed point is *stable* (more precisely, locally asymptotically stable) if any starting point x_0 produces a sequence of iterates x_k by $x_{k+1} = F(x_k)$ with $x_k \to x_*$ as $k \to \infty$ if x_0 is close enough to x_*. The *linearization* of a nonlinear mapping F about a fixed point x_* is the linear iteration

$$x_{k+1} - x_* = A\,(\,x_k - x_*\,)\ . \tag{1}$$

We showed in class that a linear iteration is stable if all the eigenvalues of A have $|\lambda| < 1$. We also showed that if $|\lambda| > 1$ for any eigenvalue of A, then the linear iteration (1) is *unstable* in the sense that no matter how small you require $\|x_0\|$ to be, it is possible to choose x_0 so that $\|x_k\| \to \infty$ as $k \to \infty$. We say that x_* is *linearly stable* if the linearization (1) is stable, where A is the Jacobian matrix $A = Df(x_*)$. *Lyapunov's theorem* states that if x_* is a linearly stable fixed point than x_* is locally asymptotically stable for the nonlinear iteration. The proof of Lyapunov's theorem also shows that if the linearization is unstable then the nonlinear iteration is locally unstable in the sense that there is an $r > 0$ so that for any $\epsilon > 0$ there is an x_0 with $\|x_0\| < \epsilon$ but $\|x_k\| > r$. This implies that an unstable fixed point (a fixed point that is linearly unstable) has no basin of attraction. Thus, linear stability or instability implies nonlinear local stability or instability.[1]

Suppose u is a loss/objective function (one component of output, d components of input) that has all derivatives up to third order well defined in a neighborhood of x_*, and x_* is a stationary point, meaning that $\nabla u(x_*) = 0$. Gradient descent with learning rate/step size s is the iteration

$$F(x) = x - s\nabla u(x)\ . \tag{2}$$

[1] Warning: An eigenvalue with $|\lambda| = 1$ is called *neutrally stable*. A one dimensional iteration $\xi_{k+1} = \lambda \xi_k$ with such a λ gives trajectories that are not asymptotically stable nor unstable in the sense that $|\xi_k|$ does not go to zero or to infinity as $k \to \infty$. If A has an unstable eigenvalue $|\lambda| > 1$, then the fixed point is unstable. But if all the eigenvalues are (strongly) stable ($|\lambda| < 1$ or neutrally stable, then Lyapunov's theorem does not apply, and x_* might or might not be stable for the nonlinear iteration.

Newton's method with step size $s = 1$ is

$$F(x) = x - \left(D^2u(x)\right)^{-1} \nabla u(x) \ . \tag{3}$$

Here, D^2u is the Hessian matrix of second partial derivatives.

(a) Show that if $H = D^2u(x_*)$ is positive definite then x_* is at least a local minimizer of u (meaning that x_* might or might not be a global minimizer).

(b) Show that if H is positive definite and s is small enough, then the gradient descent iteration (2) has x_* as a stable fixed point.

(c) Show that if H is non-singular but is a saddle point (at least one $\lambda > 0$ and one $\lambda < 0$), then x_* is an unstable fixed point of gradient descent (2), for any $s > 0$.

(d) Show that the Newton iteration (3) has x_* as a stable fixed point as long as H is non-singular whether or not H is positive definite. *Hint.* You know the eigenvalues of the zero matrix. *Emphasis.* Gradient descent converges to local minima but saddle points are unstable for gradient descent. In practice, this means that gradient descent is unlikely to converge to a saddle point even if the initial guess is close to it. Newton iteration can and does converge to saddles.

2. Consider the convex function of one variable

$$f(x) = \sqrt{1 + x^2} \ . \tag{4}$$

(a) Show that there is a learning rate parameter s so that gradient descent converges locally linearly from any starting point $x_0 \neq 0$.

(b) Show that for any fixed step size parameter s there is an r so that if $x_0 > r$, then the Newton iterates with step size s diverge to infinity: $|x_n| \to \infty$ as $n \to \infty$.

(c) Show that Newton's method with step size $s = 1$ has local quadratic convergence for this example.

3. Consider the convex function of one variable

$$f(x) = x^4 \ . \tag{5}$$

(a) Show that for any fixed learning rate parameter s there is an r so that if $x_0 > r$, then the gradient descent iterates with learning rate s diverge to infinity: $|x_n| \to \infty$ as $n \to \infty$.

(b) Show that for step size parameter $s = 1$, Newton's method iterates x_n converge linearly to zero as $n \to \infty$ for any initial guess x_0. Explain why the local convergence is linear rather than quadratic.

4. Write a module that implements Newton's method for a loss function $u(x) = u(x_1, \cdots, x_d)$. Your algorithm should have two safeguards. One is the linear search safeguard used for gradient descent in Assignment 6, but always starting with step size $s_n = 1$ before expanding or contracting s_n to insure sufficient decrease. The other safeguard is a guarantee that the search direction is a descent direction. Take the standard Newton search direction
$$p = -H^{-1}\nabla u$$
if H is positive definite (H is the Hessian matrix of u). Take $p = -\nabla u$ if H is not positive definite.[2] The Python/Numpy.linalg routine that computes the Cholesky decomposition can determine whether H is positive definite (with issues about roundoff that we will ignore for this assignment, alas). Find a simple $d = 2$ example function u that is unimodal but with regions where H is not positive definite. Your u should not be quadratic (why not), but should have a non-degenerate minimizer and minimum. Present thoughtful output (well formatted tables or scatterplots, not too much, not too little) to demonstrate global convergence even from distant initial guesses where H is not positive definite. Also demonstrate local quadratic convergence. If the optimizer is x_* (you may choose $x_* = 0$ for convenience), make sure your test problem has lots of non-vanishing third partial derivatives at $x = x_*$. Most real loss functions are like this. A test problem with vanishing third derivatives at $x = x_*$ will have "super-quadratic" convergence, which is rare in real problems.
You will be graded on the quality and thoughtfulness of the code, the output, and the choice of test problem. Think about how you will test for local quadratic convergence. This should be something more quantitative than simply: wow, fast local convergence. Be aware that roundoff error can interfere with local quadratic convergence once errors get too small, and handle this issue thoughtfully.

5. This exercise uses Newton and gradient minimization to find arrangements of n ions[3] in an ion trap. The ions are held in what is called a *Paul trap* by what amounts to a quadratic potential. The trick for doing this won Wolfgang Paul a Nobel Prize in physics. The ions also repel each other because they are electrically charged. The Wikipedia page (clickable link) has more background and on quantum computing and an image of actual ions in a trap (clickable link). The ions position themselves to minimize the total potential energy of the system. Suppose the ions are arranged on a line and ion j is at a distance r_j on that line (the line is vertical in the linked image), the potential has *one body* and *two body* terms (the

[2] We explained in class that there are better versions of this safeguard, but all of them seem too complicated for a quick weekly assignment.

[3] A normal atom has the same number of electrons as protons and is electrically neutral. The atoms in ion traps typically are missing one electron so they have a net positive electrical charge.

"bodies" are ions). The one body potential for an ion at r_j is

$$U_1(r_j) = \frac{1}{2} r_j^2 \ .$$

This represents the force that holds ions in the trap. The two body potential for two ions at r_j and r_k is

$$U_2(r_j, r_k) = \frac{-1}{|r_j - r_k|} \ . \tag{6}$$

This represents electro-static repulsion between ions. If there are n ions, the total potential is

$$U_{\text{tot}}(r_1, \cdots, r_n) = \sum_{j=1}^{n} U_1(r_j) + \sum_{j<k} U_2(r_j, r_k) \ . \tag{7}$$

There is a term in the last sum for every pair of distinct ions $j \neq k$. You insure that every pair contributes once by requiring $j < k$. The coulomb potential (6) would be infinite if $j = k$, but those "diagonal" terms are omitted from the sum.

The *force* on ion j is the negative derivative of the potential U_{tot} with respect to r_j with all the other positions held constant. If you differentiate the sum (7) with respect to r_j, you find a one body force

$$F_1(r_j) = -r_j \ .$$

and two body forces

$$F_2(r_j, r_k) = \frac{\text{sign}(r_j - r_k)}{(r_j - r_k)^2} \ .$$

If $r_j > r_k$, then the sign in the numerator is $+$ and the force pushed ion j to the right, which is away from ion k. Ion j is also pushed away from ion k if $r_j < r_k$. This is the Coulomb electro-static repulsion. You can check that the total force on ion j is

$$F_{\text{tot}}(r_j) = -r_j + \sum_{k \neq j} \frac{\text{sign}(r_j - r_k)}{(r_j - r_k)^2} \ . \tag{8}$$

If you differentiate the double sum on the right of (7) with respect to one of the ion positions, say r_i, there are terms where $j = i$ and $k \neq i$ and there are other terms where $i = k$. The Coulomb potential (6) is symmetric with respect to j and k, so all the terms in the force sum (8) have the same form. The forces on ion j are in balance if the total force on ion j is zero. The gradient of U_{tot} with respect to the variables r_j is equal to zero (as an n component gradient vector) if the forces on all the ions are in balance. This implies that stationary points of U_{tot} are configurations that don't move.

Write code to implement this model with n particles (n being a parameter in the code). This should take the form of a Python object that has attributes that evaluate the potential function, one for the gradient and one for the Hessian (three in all). Try the minimization using gradient descent (code from Assignment 6) and Newton's method (code from Exercise 4). Try to make a thoughtful initial guess. In particular, the starting configuration (initial guess) cannot have $r_j = r_k$ for $j = k$ because that makes the potential infinite and derivatives undefined. Things to look for and *comment on*. Do as many of these or others that you find interesting as time permits. You should do some but need not do all.

- You can find the analytical solution for $n = 2$ – an easy calculus problem. Check that both optimizers give the right answer.
- Check that the optimizers give answers that agree to the accuracy to which they are computed. The hard part of this is figuring out what is "the accuracy to which they are computed".
- Compare the convergence rates – local and global – for small and large n and for good and bad starting configurations.
- Make some visualizations of the equilibrium ion positions and the total potential as a function of iteration (particularly for poor initial guess and for gradient descent). Think about making a movie of the convergence for gradient descent. The ions will slowly settle into their equilibrium positions.
- See how large n you codes can handle.
- (for people with more applied math or physics training or interest). Try to understand the density of particles as a function of r when n is large. You can do this computationally by making histograms as we did to estimate probability density. You can even think about doing this analytically, but that is beyond the training of most people taking Scientific Computing.