CS 421: Numerical Analysis
Fall 2000
Problem Set 5


Due: Fri., Nov. 17 in lecture.

1. It has been proposed in the literature to use Newton’s method to compute the inverse of a matrix. Let $A$ be an $n \times n$ nonsingular matrix. Consider the nonlinear equations $f(X) = A - X^{-1}$. Then if $f(X^*) = 0$, clearly $X = A^{-1}$. It can be shown that the Newton iteration for solving $f(X) = 0$ is $X^{(k+1)} = 2X^{(k)} - X^{(k)}AX^{(k)}$.

(a) Show by a direct argument that this iteration converges quadratically provided that all the eigenvalues of $AX^{(0)} - I$ are less than 1 in absolute value. [Hint: Let $Y^{(k)} = AX^{(k)} - I$. Find a formula for $Y^{(k+1)}$ in terms of $Y^{(k)}$.]

(b) Show that there exists an $\alpha > 0$ such using $\alpha A^T$ for $X^{(0)}$ satisfies the condition in (a) (i.e., for this particular $X^{(0)}$, all eigenvalues of $AX^{(0)} - I$ are less than 1 in absolute value). Note that solving part (b) of this question does not require knowing how to solve part (a).

2. Let $f: \mathbb{R}^n \to \mathbb{R}$ be twice-continuously differentiable. Let $H$ denote $\nabla^2 f(x)$ for some $x \in \mathbb{R}^n$. Consider the step defined by $h = -(H + tI)^{-1}\nabla f(x)$, proposed in lecture. Note that $h$ depends on $t$. Considering $H$ and $\nabla f(x)$ as fixed, show that $\|h\|_2$ is a nonincreasing function of $t$ for $t \in (-\lambda_{\text{min}}(A), \infty)$. [Hint: Diagonalize.]

3. Consider the problem of solving a symmetric positive definite linear system $A\mathbf{x} = \mathbf{b}$. Derive an iterative algorithm for solving this system based on the following idea. Apply the steepest descent optimization algorithm to the quadratic function

$$f(x) = \mathbf{x}^T A\mathbf{x}/2 - \mathbf{b}^T \mathbf{x}.$$ 

Note that the minimizer $\mathbf{x}^*$ of $f$ is the same as the solution to $A\mathbf{x}^* = \mathbf{b}$, as proved in lecture. An exact (optimal) line-search should be used.

Write out this algorithm. Every step of the algorithm should be in closed form. In particular, come up with a closed-form expression to compute the optimal line-search parameter $\alpha_k$. Each iteration should require a linear number of flops, plus a constant number of matrix-vector multiplies.

[Note: for two points extra credit, explain how to implement this algorithm with $O(n)$ flops plus only one matrix-vector multiplication per iteration. Extra credit cannot raise your score on this problem set above 40.]

4. The distance geometry problem is a well-known NP-hard problem used by chemists to interpret the data from NMR experiments on molecules. The problem is as follows: a
molecule is composed of \( N \) atoms whose positions are \( \mathbf{x}_1, \ldots, \mathbf{x}_N \), where each \( \mathbf{x}_i \) is an unknown 3-vector. The NMR experiment gives back a set \( L \) that is a subset of pairs of indices (i.e., \( L \subset \{1, \ldots, N\} \times \{1, \ldots, N\} \)). Furthermore, for each \((i, j) \in L\), the NMR experiment gives the value \( d_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||_2 \). (To prevent ambiguities, assume that \( i < j \) for each \((i, j) \in L\).) The problem is to reconstruct the atomic positions from this partial distance data.

In Matlab, implement the Gauss-Newton method for the two-dimensional version (i.e., each \( \mathbf{x}_i \in \mathbb{R}^2 \)) of the distance geometry problem. In two dimensions, distance geometry is still NP-hard. The nonlinear least-squares formulation of distance geometry is

\[
\min_{(i,j) \in L} \sum (||\mathbf{x}_i - \mathbf{x}_j||_2^2 - d_{ij}^2)^2
\]

where \( \mathbf{x}_1, \ldots, \mathbf{x}_N \) are the unknowns (i.e., \( 2N \) unknowns total).

Unfortunately, the Jacobian is rank-deficient for this problem because a uniform translation or rotation of all coordinates does not affect the distances. To force the Jacobian to be full rank, assume \((1, 2) \in L\) (which is possible WLOG by renumbering the atoms), and fix \( \mathbf{x}_1 = (0, 0) \) and \( \mathbf{x}_2 = (0, d_{12}) \). This leaves only \( 2N - 4 \) unknowns, namely, \( \mathbf{x}_3, \ldots, \mathbf{x}_N \).

Test your Gauss-Newton program on some test data. To assist in the preparation of test cases, I have created some m-files which you may download from the course website. Invoking \texttt{pairlist = threepath(N)}; where \( N \) is the number of atoms desired, creates a list \( L \) using randomization. Each row of the return variable \texttt{pairlist} is a pair in \( L \), i.e., the return variable has two columns and all whole number entries. The first row of \texttt{pairlist} is always \((1, 2)\).

Then, to make distances, use the call \texttt{[distlist,sol] = makedistlist(pairlist)}; where \texttt{pairlist} is the return variable from the previous routine. Say that \( L \) is \( p \times 2 \); then this routine returns a \( p \)-vector holding distances for testing your software. It computes the distances by actually making a randomly positioned molecule and then measuring the distances. In this way, you are assured that the global minimum for the least-squares problem is zero. (Recall from lecture that Gauss-Newton works best if the solution to which it is converging has zero residual.)

These two arrays, \texttt{pairlist} and \texttt{distlist}, define the problem data. You should write a routine to solve the problem using Gauss-Newton. Your routine should have two loops: the outer loop is on randomized initial guesses for the positions and the inner loop is the actual Gauss-Newton loop. The outer loop is necessary because solving the distance geometry problem is an example of \textit{global optimization}, but Gauss-Newton carries out only local optimization. A naive but popular strategy for global optimization is to simply re-run a local optimization algorithm for many starting points.

To assist in the grading, please organize your routines as follows. There should be a routine for evaluating the function yielding the vector \( \mathbf{g}(\mathbf{x}_3, \ldots, \mathbf{x}_N) \in \mathbb{R}^{p-1} \) encoding the objective (i.e., the components of \( \mathbf{g} \) are the various values of \( ||\mathbf{x}_i - \mathbf{x}_j||_2^2 - d_{ij}^2 \) for each of the \( p - 1 \) entries in \( L \), omitting the first one, which is fixed). This function should have the form:
function fval = evalfun(positions, pairlist, distlist)

There should be a routine for evaluating the Jacobian of g:

function jac = evaljac(positions, pairlist, distlist)

This function should return a \((p - 1) \times (2N - 4)\) matrix.

In both of these calls, \texttt{positions} is an \(N \times 2\) array of the current iterate’s atom positions, and \texttt{pairlist} and \texttt{distlist} are as above.

Remark: it is easier to write evaljac to first compute a \(p \times (2N)\) matrix, and then to delete the unneeded rows and columns at the end using subscripting operations. This remark also applies to \texttt{evalfun}. Second remark: the Jacobian is sparse. So you are welcome to experiment with Matlab sparse matrices, though this is not required for the question.

In your Gauss-Newton iteration, terminate when \(J^T g\) is sufficiently small (how small?), where \(J\) denotes the Jacobian. Gauss-Newton does not always converge, so you need a second termination test that activates when too many iterations go by (how many?) Note also that even though \(\|J^T g\|\) is involved in the Gauss-Newton test, to see whether the problem is actually solved (i.e., you have found a globally optimal fit for the data), you must also check \(\|g\|\). The case when \(\|J^T g\| = 0\) but \(\|g\| \neq 0\) occurs when Gauss-Newton converges to a local optimum that is not globally optimal.

Turn in listings of all m-files and a few paragraphs describing your design choices, and describing how well everything worked. Plots are optional, but a picture is worth a thousand words! With my implementation, I was able to globally solve problems with \(N = 12\) but not with \(N = 20\).