CS 421: Numerical Analysis
Fall 2005
Problem Set 5

Handed out: Fri., Nov. 11.

Due: Mon., Nov. 21 in lecture.

1. The *Krylov space* $K_k(A, \mathbf{b})$ for an $n \times n$ matrix A and n-vector $\mathbf{b}$ is defined to be

$$K_k(A, \mathbf{b}) = \text{Span}(\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \cdots, A^{k-1}\mathbf{b}).$$

   (a) Argue by induction that $\mathbf{x}^{(k)}$ computed by conjugate gradient lies in $K_k(A, \mathbf{b})$. (Assume the starting guess is $\mathbf{x}^{(0)} = \mathbf{0}$.)

   (b) Show that the solution $\mathbf{x}$ to the linear system $A\mathbf{x} = \mathbf{b}$ lies in $K_n(A, \mathbf{b})$. Assume A is symmetric and positive definite. [Hint: Clearly the $n+1$ vectors $\mathbf{b}, A\mathbf{b}, \ldots, A^n\mathbf{b}$ must be linearly dependent since they lie in $\mathbf{R}^n$. Write out an equation of linear dependence, and pay attention to the index i such that the coefficient of $A^i\mathbf{b}$ is nonzero, and such i is minimal with this property.]

2. Consider finding an real eigenpair of a matrix $A \in \mathbf{R}^{n\times n}$. This can be accomplished by solving the system of $n+1$ nonlinear equations $A\mathbf{x} = \lambda\mathbf{x}$, $\mathbf{x}^T\mathbf{x} = 1$ for the $n+1$ variables $(\mathbf{x}, \lambda)$.

   (a) Write out Newton's method for these nonlinear equations.

   (b) Show how a preliminary Hessenberg factorization of A can reduce the number of flops need per Newton iteration.

3. (a) Consider minimizing the quadratic objective function $f(\mathbf{x}) = \mathbf{x}^T H\mathbf{x}/2 + \mathbf{b}^T\mathbf{x}$, where H is a given symmetric positive definite matrix and $\mathbf{b}$ is an n-vector. Show that Newton's method for minimization from any starting point solves this problem exactly in a single step. Note: One of the examples in Chapter 6 of the book illustrates this fact.

   (b) In the Armijo line search presented in lecture, the somewhat arbitrary constant 0.1 appeared in the algorithm. It is possible to use a different value. But explain why that constant should never exceed 0.5. [Hint: consider using the Armijo line search in conjunction with NM for the problem described in part (a). Something goes wrong if the constant exceeds 0.5.]

4. Implement Newton's method for finding minimum-area surfaces. In more detail, the problem is as follows. We want to determine an unknown surface S embedded in $\mathbf{R}^3$. We are given the boundary Γ of S, which is assumed to be a simple closed curve in $\mathbf{R}^3$, We want to find S, a surface of minimum area whose boundary is Γ. In fact, we will settle for an approximation to S by triangles.

To simplify the problem, we assume that projection of S into the (x, y) plane is a bijection, i.e., no two points of S have the same (x, y) coordinates. This means that we can use the following solution procedure to find an approximation to S. First, project the boundary curve Γ into the (x, y) plane and compute a triangulation of its interior. Then use Newton's method for unconstrained multivariate optimization to minimize the sum of the areas of all the triangles once they are embedded in $\mathbf{R}^3$, where the boundary curve Γ is embedded into its given 3D position, the (x, y) coordinates of interior nodes are also fixed in their given position, and the z-coordinates of the interior nodes are the variables (unknowns) of the Newton problem. Therefore, part of your task is to write a routine that, given the z-coordinates of the interiors, computes the surface area of the resulting 3D mesh. (This is the routine that takes $\mathbf{x}$ and returns $f(\mathbf{x})$ for use in Newton's method.)

Test this routine on three cases, which are as follows. In all cases the boundary Γ, when projected onto the (x, y) coordinate plane, is the boundary of a unit square. In the first case, the z coordinates of all boundary nodes are 1's. Then the minimum-area surface will have z-coordinate 1. In the second case, the z coordinates of all boundary nodes are the sum of the x and y coordinates. Again in this case, the minimum-area surface is flat, and the z-coordinates of interior nodes will also come out to be sums of x and y coordinates. The last case tests curved boundaries; sin functions are used to make curves that are concave up along two of the square's edges and concave down on the other two edges.

In more detail, here are the five routines that make up this question.

- `[xyz,trilist,isbdry] = unitsquare_meshgen(n, type)`. This routine is provided for you on the course webpage. Its job is to take as input n, the number of nodes per side in the unit square, and `type`, which is either 0, 1, or 2. This `type` variable determines which of the three boundary conditions (enumerated above) will be applied. The return variables for this function are documented on the web page. Briefly, `xyz` holds the (x, y, z) coordinates of the mesh nodes. The (x, y) coordinates for all the nodes are valid. The z-coordinates for the nodes are valid only on the boundary; z-coordinates of interior nodes are initialized as 0's.
- `[f,g,h] = minsurf(xyz, trilist)`. You must write this routine. It takes as input `xyz` and `trilist` and returns the function value f, i.e., the area of the surface given by the 3D triangles specified by input arguments `xyz` and `trilist`. It should also return the first and second derivatives in variables g and h respectively of surface area with respect to the z-coordinates (all z-coordinates, not just those of interior nodes).
- `[f,g,h] = minsurf_wrapper(xyz, trilist,isbdry,x)`. This routine takes as input same as above, plus a vector `x` that should contain the z-coordinates of the interior nodes. It returns f, g, h as above, except that it masks out the entries of g and h associated with boundary nodes (which are not used in Newton's method). This routine is provided for you on the course webpage.
- `[x,nrmgseq,alphaseq] = newton(fghfun, x0, tol, maxit)`. This routine implements Newton's method. It is provided for you. Its input and output argu-

ments are described in its comments.

- `driverq4(n,type)`. This routine is the driver for the whole process. You must write this routine. Its arguments `n` and `type` should be passed as input to `unitsquare_meshgen`. Then it makes a function handle that wraps `minsurf_wrapper`. It then calls `newton`. The convergence tolerance can be 10^{-10}, and the initial guess can be all 0's. The routine should display or plot the sequence of $\|\mathbf{g}\|$ values and α values. Then it should show the final mesh on the screen using the `trisurf` function. See the example below.

Some hints for writing `minsurf` are as follows. The area of a 3D triangle is the half of the 2-norm of the cross-product of two of its sides (represented as vectors). Review the cross-product from your calculus book if you don't recall it, or else lookup some other formula for 3D triangle area. In my implementation of this function, I wrote down the formula for the square of the 2-norm by hand, and then I used Matlab's symbolic toolbox to differentiate it once and twice. I cut and pasted the output from the toolbox into my program. The derivatives of the actual function (i.e., the square root of the squared 2-norm) can then be obtained by applying the chain rule and product rule.

To help you debug your code, I have posted a sample run of my code with some random data in the file randomfgh.txt.

As for writing `driverq4`, one technique you need is to generate a function handle. The issue here is that `newton` takes as input a handle to a function `fghfun` that takes a single argument `x` and returns f, g, h. On the other hand, the actual function that evaluates f, g, h, namely, `minsurf_wrapper`, needs three additional arguments, namely, `xyz`, `trilist`, and `isbdry`. The standard Matlab technique of hiding additional parameters in a function handle takes care this issue. To read about this technique, look in the Matlab on-line help by clicking `help` > `Full product family` in the command window menu bar, clicking the tab labeled "contents" in the help window, and then going to the topic Matlab > Programming > Types of Functions > Anonymous Functions > Examples of Anonymous Functions > Example 1.

Here is an example of the output from driverq4: