Introduction to Numerical Simulation (Fall 2003)
Problem Set #6 - due October 27

Note: Please get started early! This problem set is not due until the 27th, but it has several problems!

1) When modeling distributions of charged particles governed by drift-diffusion equations, equilibrium analysis typically leads to the following nonlinear Poisson equation

$$-\frac{\partial^2 \psi(x)}{\partial x^2} = -(e^{\psi(x)} - e^{-\psi(x)}), \tag{1}$$

where we will consider the interval $x \in [0, 1]$ with the boundary conditions $\psi(0) = -V$ and $\psi(1) = V$.

If a simple finite-difference scheme is used to solve the nonlinear Poisson equation on an N-node grid, the discrete equations are

$$2\psi_i - \psi_{i+1} - \psi_{i-1} + \Delta x^2(e^{\psi_i} - e^{-\psi_i}) = 0$$

for $i \in [2, ..., N-1]$,

$$2\psi_1 - \psi_2 - (-V) + \Delta x^2(e^{\psi_1} - e^{-\psi_1}) = 0,$$

and

$$2\psi_N - \psi_{N-1} - V + \Delta x^2(e^{\psi_N} - e^{-\psi_N}) = 0.$$

Note, $\Delta x = 1/(N+1)$ (not $1/(N-1)$–the nodes at $x = 0$ and $x = 1$ are not included in the discretization, but rather enter through the boundary conditions).

a) Prove that the Jacobian associated with the above discretized equations is nonsingular regardless of the values for the ψ_i's. What does this imply about damped Newton methods applied to solving this problem.

b) Solve the above equations with a multidimensional Newton's method using a zero initial guess for the ψ_i's, and $N = 100$. Demonstrate that your program achieves quadratic convergence, and determine the number of Newton iterations required to insure one part in 10^6 accuracy in the solution for two cases: when $V = 1$ and when $V = 20$. What happens when $V = 100$?

c) Try using damped Newton to solve for the ψ_i's when $V = 100$. For roughly how large a V can you compute ψ to one part in 10^6 accuracy in fewer than 200 residual evaluations (be sure to count the function evaluations you perform to determine how much to damp the newton delta). How many linear system solutions were required?

2) Consider the Jacobian-free method described in class. In that approach, Newton's method is used to solve $F(x) = 0$, and the GCR algorithm is used to solve the Newton update equation,

$J_F(x^k)(x^{k+1} - x^k) = -F(x^k)$. Using GCR implies that $J_F(x^k)$ will not factored, but is instead only used to form matrix-vector products, $J_F(x^k)r$. The method is then made Jacobian (or matrix)-free by using the approximation

$$J_F(x^k)r \approx \frac{1}{\alpha}[F(x^k + \alpha r) - F(x^k)].$$

So, the name Jacobian-free follows from the fact that only a routine to compute $F(x)$ is needed, but not a routine to compute $J_F(x)$.

a) For the discretized nonlinear poisson equation from problem one, compare the convergence rate of standard Newton to the convergence rate of the Jacobian-free method. To compare converge rates, make a semilog plot of $||F(x^k)||/||F(x^0)||$ versus k for the two methods and explain your results. For purposes of this experiment, use $N = 100$ and $V = 10$. For the GCR algorithm, assume convergence when $\frac{||r^k||}{||r^0||} < \epsilon$ where r^k is the k^{th} residual and $\epsilon = 0.1$. Finally, when approximating the matrix-vector product, use $\alpha = 10^{-3}$ (defined above).

b) How accurate can you make your Jacobian-free method by modifying ϵ and α? That is, how close to zero can you make $F(x^k)$? How close to zero can you make $F(x^k)$ using standard Newton's method?

c) Suppose you are interested in low accuracy calculations, and only need to converge Newton's method so that $||F(x^k)|| < 10^{-3}$. For what value of ϵ will you achieve the desired accuracy in the fewest function evaluations?

d) Another approach to trying to apply Newton's method using only function evaluations is to compute the Jacobian approximately by finite-differences. That is,

$$(J_F(x^k))_{i,j} \approx \frac{1}{\alpha}[F_i(x^k + \alpha e_j) - F_i(x^k)]$$

where e_j is the vector with 1 in the j^{th} entry and zeros elsewhere. How many function evaluations per Newton iteration would such a finite-difference method require?

3) In this problem, you will debug our matlab based Newton solver for calculating the force equilibrium position of joints in a loaded structure (or you are certainly welcome to write your own Newton solver from scratch, whichever you find easier).

The program should read the following format as input.

joints:

jlabel *nodenumber* *xposition* *yposition*

where *label* is an arbitrary label, *nodenumber* is an integer joint node number, and *xposition* and *yposition* are the joint's x and y coordinates when the struts are not stretched (the structure is not loaded).

struts:

slabel *node1* *node2* *elasticity*

where *label* is an arbitrary label, *node1* and *node2* are integer joint node numbers, and *elasticity* is the elasticity of the strut.

loads:

llabel *nodenumber* *xforce* *yforce*

where *label* is an arbitrary label, *nodenumber* is the joint node number where the load force is applied, and *xforce* and *yforce* are the x and y components of the force.

If you decide to use our matlab scripts with carefully inserted bugs, then on the course web page, you will find a set of Matlab files which use Newton's method to find the equilibrium position of a set of struts and joints subjected to external forces. The code uses the nodal analysis formulation, so you may wish to review that material before starting this problem.

The driver files for the struts/joints code are *readsystem.m* and *newton.m*. *readsystem.m* reads in the unstretched (initial) joint positions, the strut connectivity, and the applied forces. *newton.m* contains the code for Newton's method. The file *loadNewton.m* constructs the right-hand side and Jacobian for Newton's method. The file *force.m* contains code to calculate the strut forces and derivatives of forces. It may be necessary to modify any or all of *newton.m*, *loadNewton.m* and *force.m*. There are three test files (which will give clues as to what is wrong) – test1.sys, test2.sys and test3.sys.

a) Provide a Newton solver that works on test1.sys, test2.sys and test3.sys. To do this, determine if the provided code works (if it does, you have to justify this conclusion, if it doesn't, you'll have to explain why and fix it).

b) Demonstrate that your resulting Newton's method, applied to each of the example problems, converges quadratically.

4) There are a few more test examples that will cause problems even for a correctly programmed Newton method.

a) Try your debugged Newton-based struts and joints solver on example test4.sys. What happens and why?

b) Try your debugged Newton-based struts and joints solver on examples test5.sys and test6.sys. Why are the answers different?

5) (Final Entries by midnight October 31st) The traditional 6.336 Newton method contest!!! A Prize for the person who can get Newton's method to converge on the most examples, and in the case of a tie, in the fewest iterations.

a) Change your struts and joints program so that the force is now exponentially related to how much the strut has stretched. Specifically, suppose the magnitude of the strut restoring force is given by

$$|f| = e^{elasticity * (length - unstretchedlength)} - 1.0.$$

Note, if the change in length is small, the above form reduces to the model you already implemented. Check your new program on some SIMPLE examples. Verify quadratic convergence!

b) Try whatever schemes you can think of to get Newton's method to converge on as many of test7.sys through test11.sys as you can. Be forewarned, however, even we haven't necessarily succeeded with all of them.