CS 421: Numerical Analysis
Fall 2004
Problem Set 6

Handed out: Mon., Nov. 22.

Due: Fri., Dec. 3 in lecture.

1. (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.]

2. (a) (From Trefethen's book on the web.) Consider the IVP $dy/dt = y^{1/2}$, $y(0) = 0$. Show that this IVP has two different solutions of the form $y = at^b$ for constants a and b. Then show that in fact the IVP has an infinite number of solutions that can be obtained by gluing these two solutions together (i.e., follow one solution for a period of time then the other).

   (b) Look up the theorem that guarantees uniqueness for the solution of IVP's on p. 387 of the text. Explain why the hypotheses of that uniqueness theorem do not apply to this IVP.

3. For both parts of this question, assume constant stepsize.

   (a) Determine the local truncation error (including the coefficient) of 2-step Adams Moulton rule, which is

   $$y_{k+1} = y_k + (5h/12)f(y_{k+1}, t_{k+1}) + (8h/12)f(y_k, t_k) - (h/12)f(y_{k-1}, t_{k-1})$$

   using the R-method from lecture.

   (b) Show that the order of accuracy of the method $y_{k+1} = 2y_k - y_{k-1}$ is 1 (i.e., same order as EM). There is something obviously wrong with this method! Explain why this method is completely useless in practice.

4. Implement Newton's method and the Gauss-Newton method for the following data-fitting problem. You are given time samples of a signal in the form of ordered pairs (t_k, y_k). The signal is suspected to be a damped oscillation, i.e., the relation $y_k = z\exp(ut_k)\sin(vt_k + w)$ should hold for some unknown model parameters u, v, w, z. The

problem is to find the best values for the model parameters to fit the data. This is a nonlinear least squares problem: find u, v, w, z to minimize

$$f(u, v, w, z) = \sum_{k=1}^{n} g_k(u, v, w, z)^2$$

where

$$g_k(u, v, w, z) = z \exp(ut_k) \sin(vt_k + w) - y_k.$$

The actual dataset of (t, y) pairs is downloadable from the course web page.

In your implementation of Newton's method, you do not have to implement a line search nor a safeguard against non-positive definite Hessian. Obviously, this will decrease the robustness of Newton's method. Use as a termination criterion that the gradient of f is sufficiently small, say 10^{-10}. Also, include a max-iteration bound to prevent infinite loops. In your implementation of the Gauss Newton method, use the same termination tests.

Finally, test also the built-in curve-fitting function `lsqcurvefit` on the same data. To have consistency for the sake of grading,implement a single function that evaluates f, g, ∇f, ∇g and $\nabla^2 f$. The form of this function should be

```
[f, f_grad, f_hess, g, g_jac] = fit_damped_osc(param,t,y)
```

where `param` is a vector of the four parameters listed in the order above. The entries of the derivatives should follow the same order. It is suggested that this routine `fit_damped_osc` should first evaluate `g` and `g_jac` since the derivatives of f can be expressed in terms of those of g_k's (How? Show your equations.) On the class webpage you will find a sample run of my version of `fit_damped_osc` using random data. You can try to match my version with the same random data to assist you in debugging your code.

This function is then invoked by both the Newton solver and the Gauss-Newton solver. It should also be invoked by `lsqcurvefit`, but through a wrapper function. A wrapper is needed because `lsqcurvefit` expects its function to have input and output arguments in a certain order and form. Type `help lsqcurvefit` for more information. The wrapper function sets input argument `y` to 0's and extracts output argument `g`.

Once all of this is implemented, test it as follows. First, find a parameter set close to optimum for which all three methods converge quickly. This can be done by hand: plot the data and inspect the plot to guess u, v, w, z.

Then try starting parameter sets that are farther away from the true solution. For example, you could multiply all the good parameters by .8, then .7, etc. See which of the three methods is able to converge from farther away from the starting set. Hand in a plot that shows on the x-axis the distance of the starting point to the solution and on the y-axis how many iterations each of the three methods requires. Hand in a second plot with the same x-axis but with a y-axis that shows the the residual norm (that is $\|\mathbf{g}\|_2$) at the converged solution of the methods. This second plot should probably use a logarithmic scale for the y-axis.