Scientific Computing, Fall 2015
Assignment IV: Nonlinear Equations and Optimization

Aleksandar Donev
Courant Institute, NYU, donev@courant.nyu.edu

Posted Oct 16th, 2015
Due by Sunday **Nov 1st**, 2015

For grading purposes the maximum is considered to be 75 points, but you can get up to 100 points with extra credit.

1 [25 points] Newton-Raphson Method in One Dimension

Consider finding the three roots of the polynomial

$$f(x) = 816x^3 - 3835x^2 + 6000x - 3125,$$

which happen to all be real and all contained in the interval $[1.4, 1.7]$ [due to Cleve Moler].

1.1 [5pts] The roots

Plot this function on the interval $[1.4, 1.7]$ and find all of the zeros of this polynomial using the MATLAB function $fzero$ [*Hint: The roots values can be obtained in MATLAB using the built-in function roots but Maple tells us the roots are* $25/16$, $25/17$ *and* $5/3$].

1.2 [10 pts] Newton's Method

[5pts] Implement Newton's method (no safeguards necessary) for finding the roots of $f(x)$ and test it with some initial guess in the interval $[1.4, 1.7]$.

[5pts] Verify that the order of convergence is quadratic, as predicted by the theory from class:

$$\lim_{k\to\infty} \frac{\left|e^{k+1}\right|}{\left|e^{k}\right|^2} \to C = \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|.$$

[*Hint: Due to roundoff errors and the very fast convergence, the error quickly becomes comparable to roundoff, so one must be careful not to use very large k. The errors must be dominated by truncation errors and not roundoff errors for the above theory to apply!*]

1.3 [10pts] Robustness

Starting from many (say 100) guesses in the interval $[1.4, 1.7]$ [*Hint: In MATLAB, you can create a grid of 100 points with* $x_0 = linspace(1.4, 1.7, 100)$], run 100 iterations of Newton's method and plot the value to which it converges, if it does, as a function of the initial guess. If the initial guess is sufficiently close to one of the roots α, i.e., if it is within the *basin of attraction* for root α, it should converge to α. What is the basin of attraction for the middle root ($\alpha = 25/16$) based on the plot?

2 [25 pts + 25 extra credit] Nonlinear Least-Squares Fitting

In homework 2 you considered fitting a data series (x_i, y_i), $i = 1, \ldots, m$, with a function that that depends linearly on a set of unknown fitting parameters $\boldsymbol{c} \in \mathbb{R}^n$. Consider now fitting data to a nonlinear function of the fitting parameters, $y = f(x; \boldsymbol{c})$. The least-squares fit is the one that minimizes the squared sums of errors,

$$\boldsymbol{c}^\star = \arg\min_{\boldsymbol{c}} \sum_{i=1}^{m} \left[f(x_i; \boldsymbol{c}) - y_i\right]^2 = \arg\min_{\boldsymbol{c}} \left\|\boldsymbol{f} - \boldsymbol{y}\right\|_2^2, \tag{1}$$

where $\boldsymbol{f}(\boldsymbol{c}) \equiv f(\boldsymbol{x};\boldsymbol{c})$ is a vector of m function values, evaluated at the data points, for a given set of the parameters $\boldsymbol{c}$. We can think of this as an overdetermined system of nonlinear equations for $\boldsymbol{c}$,

$$f(\boldsymbol{c}) = \boldsymbol{y}, \tag{2}$$

although of course in the end this is an optimization problem rather than solving a square system of equations (remember that the two are closely related).

We will consider here fitting an exponentially-damped sinusoidal curve with four unknown parameters (amplitude c_1, decay c_2, period c_3, and phase c_4, respectively),

$$f(x;\boldsymbol{c}) = c_1 e^{-c_2 x} \sin(c_3 x + c_4), \tag{3}$$

to a synthetic data set.

2.1 [5pts] Synthetic Data

Generate synthetic data by generating $m = 100$ points randomly and uniformly distributed in the interval $0 \le x \le 10$ by using the *rand* function. Compute the actual function

$$f(x;\boldsymbol{c}) = e^{-x/2} \sin(2x), \tag{4}$$

and then add perturbations with absolute value on the order of $\epsilon = 10^{-2}$ to the y values (use the *rand* or the *randn* function),

$$y_i = e^{-x_i/2} \sin(2x_i) + \epsilon \cdot rand(). \tag{5}$$

Compare the synthetic data to the actual function on the same plot to make sure your synthetic data closely (but not exactly!) follows the relation (4).

2.2 [25+5 pts] Gauss-Newton Method

The basic idea behind the Gauss-Newton method is to make a linearization of the function $f(x_i;\boldsymbol{c})$ around the current estimate $\boldsymbol{c}_k$,

$$\boldsymbol{f}(\boldsymbol{c}) \approx \boldsymbol{f}(\boldsymbol{c}_k) + [\boldsymbol{J}(\boldsymbol{c}_k)](\boldsymbol{c} - \boldsymbol{c}_k) = \boldsymbol{f}(\boldsymbol{c}_k) + [\boldsymbol{J}(\boldsymbol{c}_k)]\,\Delta\boldsymbol{c}_k,$$

where the Jacobian $m \times n$ matrix is the matrix of partial derivatives $\partial f/\partial c$ evaluated at the data points:

$$\boldsymbol{J}(\boldsymbol{c}) = \boldsymbol{\nabla}_{\boldsymbol{c}} \boldsymbol{f}(\boldsymbol{c}).$$

This approximation (linearization) transforms the non-linear problem (2) into a linear least-squares problem, i.e., an overdetermined linear system

$$[\boldsymbol{J}(\boldsymbol{c}_k)]\,\Delta\boldsymbol{c}_k = \boldsymbol{J}_k\,\Delta\boldsymbol{c}_k = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{c}_k) = \boldsymbol{y} - \boldsymbol{f}_k, \tag{6}$$

which you know how to solve from previous homeworks and lectures. The standard approach is to use the normal equations

$$\left(\boldsymbol{J}_k^T \boldsymbol{J}_k\right)\,\Delta\boldsymbol{c}_k = \boldsymbol{J}_k^T(\boldsymbol{y} - \boldsymbol{f}_k), \tag{7}$$

which does not lead to substantial loss of accuracy if one assumes that the original problem is well-conditioned (you are welcome to use the QR factorization or backslash if you prefer, just report what you did). Gauss-Newton's algorithm is a simple iterative algorithm of the form

$$\boldsymbol{c}_{k+1} = \boldsymbol{c}_k + \Delta\boldsymbol{c}_k,$$

starting from some initial guess $\boldsymbol{c}_0$. The iteration is terminated, for example, when the increment $\|\Delta\boldsymbol{c}_k\|$ becomes too small.

[10 pts] Implement Gauss-Newton's algorithm and see whether it works for the problem at hand, using an initial guess $\boldsymbol{c}_0$ that is close to the correct values.

[5 pts] Is the convergence to the answer quadratic or linear? Answer this based on empirical (numerical) results for the purpose of this homework. If you can give a theoretical argument you can get extra extra credit points.

[5pts] If you start with $\boldsymbol{c}_0 = (1, 1, 1, 1)$, does the method converge to the correct answer? Play around a bit with initial guesses and see if the method converges most of the time, and whether it converges to the "correct" solution or other solutions.

[5 pts] If the synthetic data points have no error, i.e., if $\epsilon = 0$ in (5), how many digits of accuracy in $\boldsymbol{c}$ can you obtain? How many steps do you need to achieve this accuracy?

[5pts extra credit] Is this method the same or even similar to using Newton's method (for optimization) to solve the non-linear problem (1).

2.3 [20pts Extra Credit] Levenberg-Marquardt Algorithm

The Gauss-Newton algorithm is not very robust. It is not guaranteed to have even local convergence. A method with much improved robustness can be obtained by using a modified (regularized) version of the normal equations (7),

$$\left[\left(\boldsymbol{J}_k^T \boldsymbol{J}_k\right) + \lambda_k \operatorname{Diag}\left(\boldsymbol{J}_k^T \boldsymbol{J}_k\right)\right] \Delta \boldsymbol{c}_k = \boldsymbol{J}_k^T \left(\boldsymbol{y} - \boldsymbol{f}_k\right), \tag{8}$$

where $\lambda_k > 0$ is a damping parameter that is used to ensure that $\Delta \boldsymbol{c}_k$ is a descent direction, in the spirit of quasi-Newton algorithms for optimization. Here $\operatorname{Diag}(\boldsymbol{A})$ denotes a diagonal matrix whose diagonal is the same as the diagonal of $\boldsymbol{A}$ [*Hint: The MATLAB call diag(diag(A)) can be used to obtain* $\operatorname{Diag}\boldsymbol{A}$*, as used in (8)*].

If λ_k is large, the method will converge slowly but surely, while a small λ_k makes the method close to the Gauss-Newton Method, which converges rapidly if it converges at all. So the idea is to use a larger λ_k when far from the solution, and then decrease λ_k as approaching the solution. The actual procedure used to adjust the damping parameter can be found in the literature, and consists of doubling λ if things are not going well, or halving λ if things are going well. Here we study one simple strategy: Start with some sufficiently large initial value λ_1, and then reduce it by a factor of 2 each iteration, $\lambda_k = \lambda_{k-1}/2$.

[15pts] Implement a code that tries this method and try it for a value of the initial guess for which the Gauss-Newton method in part 2.2 above failed, i.e., for which $\lambda_1 = 0$ does not work. Try different initial values of λ_1 and see how large it has to be before the method converges.

[5pts] Do you notice any difference in the speed of convergence for different values of λ_1 (i.e., is the speed of convergence faster for smaller values or larger values)?

3 [25 pts] Quadratically-Constrained Quadratic Optimization

Consider the quadratically-constrained quadratic convex optimization problem of finding the point on an ellipse/ellipsoid that is closest to the origin:

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathbb{R}^n} \left\{ f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2 = \boldsymbol{x}\cdot\boldsymbol{x} = \textstyle\sum_{i=1}^n x_i^2 \right\} \\ \text{s.t.}\quad & h(\boldsymbol{x}) = (\boldsymbol{x}-\boldsymbol{x}_0)^T \boldsymbol{A} (\boldsymbol{x}-\boldsymbol{x}_0) - 1 = \textstyle\sum_{i,j=1}^n a_{ij} (x_i - x_{0,i})(x_j - x_{0,i}) - 1 \quad . \end{aligned} \tag{9}$$

where $\boldsymbol{A}$ is a symmetric positive-definite matrix and $\boldsymbol{x}_0$ is the location of the centroid of the ellipsoid. Note that if the sign (direction) of $\boldsymbol{x}$ is reversed it is still a solution (this non-uniqueness may cause some numerical problems!).

As an example, consider minimizing $x_1^2 + x_2^2$ along the ellipse ($n = 2$ variables)

$$x_1^2 + 2x_1x_2 + 3x_2^2 = 1, \tag{10}$$

which is centered around the origin, i.e., $\boldsymbol{x}_0 = \boldsymbol{0}$. Write down the matrix $\boldsymbol{A}$ (remember that it must be symmetric positive-definite). One of the two exact solutions for the optimal point is $x_1^\star = \frac{1}{\sqrt{2}} - \frac{1}{2}$ *and* $x_2^\star = \frac{1}{2}$. In this problem you will try to solve this problem numerically using the penalty method.

In the penalty method, we minimize the *penalty function* (9)

$$\min_{\boldsymbol{x}} \left\{ \mathcal{L}_\alpha = f(\boldsymbol{x}) + \alpha \left[h(\boldsymbol{x})\right]^2 \right\} \tag{11}$$

for a given penalty parameter α. Write a MATLAB (or other) program that implements Newton's method for minimizing $\mathcal{L}_\alpha$ for some given value of α. If you can, try to write the MATLAB code so that it works for any dimension of the problem n and any ellipsoid, i.e., for any $\boldsymbol{A}$ and $\boldsymbol{x}_0$ (not necessary but a good challenge for you!).

1. [10pts] Try your Newton's method code for some specific values, e.g., $\alpha = 1$ and some reasonable initial guess, e.g., $\boldsymbol{x}^0 = \mathbf{1}$ (all ones) or $\boldsymbol{x}^0 = randn(n, 1)$ and tell us how you verified that your code works correctly. [*Hint: Newton's method should converge quadratically to a critical point of* $\mathcal{L}_\alpha$]
2. [15pts] For the two-dimensional example (10), solve the penalized problem numerically for increasing penalty parameter $\alpha = \alpha_k = 10^k$ for $k = 0, 1, \ldots$, stopping when the increment becomes too small, for example, $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_\infty \leq \varepsilon = 10^{-12}$. Use the solution for the previous k as an initial guess to speed up Newton's method and help it converge. Plot the error in the solution to (11) as compared to the exact answer as a function of α. How large does α need to be before you can get a solution accurate to 6 significant digits?