Numerical Methods I, Fall 2014
Assignment IV: Non-linear Equations and Optimization

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

October 16th, 2014
Due: November 2nd, 2014

For the purposes of grading the maximum number of points is considered to be 100 points, so you do *not* need to solve all of the problems. Choose the ones that challenge and interest you most.

1 [35 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 [15 pts] Newton's Method

[7.5pts] Implement Newton's method (no safeguards necessary) and test it with some initial guess in the interval $[1.4, 1.7]$.

[7.5pts] Verify that the order of convergence is quadratic, as predicted by the theory from class:

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

[*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]

1.3 [15pts] Robustness

[7.5pts] Starting from many (say 100) guesses in the interval $[1.4, 1.7]$, run 100 iterations of Newton's method and see 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?

[5pts] The theory from the lecture suggested an estimate for the width of each basin of attraction around a given root α of the form:

$$\left|x^0 - \alpha\right| \leq \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|^{-1}.$$

Compare these estimates to what is actually observed numerically.

[2.5pts] Is the estimate particularly bad for one of the roots, and if so, can you think of reasons why?

2 [30 pts] Generalizations of Newton's Method

2.1 [10 pts] Multiple roots

Assume that we are using Newton's method in the vicinity of a root of multiplicity $m > 1$, meaning that the first nonzero derivative of the function $f(x)$ is $f^{(m)}(x)$.

1. [5pts] Show that Newton's method gives linear convergence. Then show that modifying Newton's method to account for the multiplicity,
$$x^{n+1} = x^n - m\frac{f(x^n)}{f'(x^n)},$$
gives second-order of convergence.
2. [5pts] Also show that applying Newton's method to the equation
$$\tilde{f}(x) = \frac{f(x)}{f'(x)} = 0$$
restores the quadratic convergence (but requires evaluating second-derivatives of f as well).
Note: This is not something to do in practice, it is merely a toy example to practice analysis on!

2.2 [20 pts] Steffensen's Method

Newton's method requires evaluation of the first derivative of the function, which can be a problem. An obvious alternative is to use a finite-difference approximation for the derivative:
$$f'(x) \approx \frac{f(x+h) - f(x)}{h},$$
which however has some truncation and roundoff error in it (recall problem 3.1 in homework 1). *Note: This is not something to do in practice, it is merely a toy example to practice analysis on!*

Our task is to figure out how to choose h to get the best convergence without too much extra work. Note that the centered difference requires evaluating the function also at $f(x-h)$ so it costs a lot more and is likely not worth it. The secant method sets $h = x_{n-1} - x_n$ so that the previous function evaluation $f(x_{n-1})$ can be reused. However, the secant method is not second-order convergent.

1. [10pts] Intuition says that in order to get true (formal) second-order convergence the derivative should become more accurate near the root, i.e., h should become smaller as $f(x)$ becomes smaller. Indeed, Steffensen's method sets $h = f(x)$. Prove that this choice gives second-order convergence. [*Hint: Express Steffensen's method as a fixed-point iteration and then apply the convergence theory from the lectures. Be careful in calculating the derivatives using the chain rule, and be very careful about the limit $x \to \alpha$.*]
2. [10pts extra credit] Implement Steffensen's method in MATLAB and verify second-order convergence (for example, using the test function from problem #1). Is roundoff a problem and if so can you do anything about it?

3 [30 pts + 30 extra credit] Nonlinear Least-Squares Fitting

In previous homeworks 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,
$$\boldsymbol{c}^\star = \arg\min_{\boldsymbol{c}} \sum_{i=1}^{m} \left[f(x_i; \boldsymbol{c}) - y_i\right]^2 = \arg\min_{\boldsymbol{c}} \left(\boldsymbol{f} - \boldsymbol{y}\right)^T \left(\boldsymbol{f} - \boldsymbol{y}\right), \tag{1}$$
where $\boldsymbol{f}\left(\boldsymbol{c}\right) = f(\boldsymbol{x}; \boldsymbol{c})$ is a vector of m function values, evaluated at the data points, for a given set of the parameters $\boldsymbol{c}$.

We will consider here fitting an exponentially-damped sinusoidal curve with four unknown parameters (amplitude, decay, period, and phase, respectively),
$$f(x; \boldsymbol{c}) = c_1 e^{-c_2 x} \sin\left(c_3 x + c_4\right), \tag{2}$$
to a synthetic data set.

3.1 [5pts] Synthetic Data

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

$$f(x; \boldsymbol{c}) = e^{-x/2} \sin(2x), \tag{3}$$

and then add perturbations with absolute value on the order of 10^{-2} to the y values (use the *rand* or the *randn* function). Compare the synthetic data to the actual function on the same plot to see how closely the data set follows the relation (3).

3.2 [25 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 (1) into a linear least-squares problem, i.e., an overdetermined linear system

$$[\boldsymbol{J}(\boldsymbol{c}_k)]\ \Delta\boldsymbol{c}_k = \boldsymbol{J}_k\,\Delta\boldsymbol{c}_k = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{c}_k), \tag{4}$$

which you know how to solve from previous homeworks and lectures. The standard approach is to use the normal equations, which does not lead to substantial loss of accuracy if one assumes that the original problem is well-conditioned. 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.

[15 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.

[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.

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

3.3 [30pts 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 for the overdetermined system (4),

$$\left[\left(\boldsymbol{J}_k^T \boldsymbol{J}_k\right) + \lambda_k \,\mathrm{Diag}\,\left(\boldsymbol{J}_k^T \boldsymbol{J}_k\right)\right]\ \Delta\boldsymbol{c}_k = \boldsymbol{J}_k^T\left(\boldsymbol{y} - \boldsymbol{f}\right), \tag{5}$$

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 $\mathrm{Diag}(\boldsymbol{A})$ denotes a diagonal matrix whose diagonal is the same as the diagonal of $\boldsymbol{A}$.

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 is somewhat of an art, and here we study one simple but effective strategy.

[20 pts] Implement a code that repeats the following cycle for $k = 1, 2, \ldots$ until the increment $\|\Delta\boldsymbol{c}_k\|$ becomes too small, starting with an initial value $\lambda = 1$ and some guess $\boldsymbol{c}_0$:

1. Evaluate the function for the present estimate of the parameters, $\boldsymbol{f}_k = f(\boldsymbol{x}; \boldsymbol{c}_k)$, and then compute the residual
$$r_k = (\boldsymbol{f}_k - \boldsymbol{y})^T (\boldsymbol{f}_k - \boldsymbol{y}) .$$
Recall that the goal is to minimize the residual.
2. Using $\lambda_k = \lambda$, compute a new trial point $\boldsymbol{c}_{k+1}$ by solving the system (5), evaluate the new $\boldsymbol{f}_{k+1} = f(\boldsymbol{x}; \boldsymbol{c}_{k+1})$, and then compute the new residual r_{k+1}.
3. Repeat step 2 for $\lambda_k = \lambda/2$ and compute the resulting residual $\tilde{r}_{k+1}$. We now have three residuals, the previous value r_k, and the two new trial values r_{k+1} and $\tilde{r}_{k+1}$.
4. If $\tilde{r}_{k+1}$ is the smallest of the three residuals, then accept the trial value $\boldsymbol{c}_{k+1}$ obtained for $\lambda_k = \lambda/2$, decrease $\lambda \leftarrow \lambda/2$, and repeat the cycle again.
5. If r_{k+1} is the smallest of the three residuals, then accept the trial value $\boldsymbol{c}_{k+1}$ obtained for $\lambda_k = \lambda$, and repeat the cycle again.
6. If r_k is the smallest of the three residuals, then increase $\lambda \leftarrow 2\lambda$ and compute the resulting residual. Keep doubling λ until the residual is smaller than r_k. Accept the new value of $\boldsymbol{c}_{k+1}$ and repeat the cycle.

[*Hint: The MATLAB call diag(diag(A)) can be used to obtain* Diag$\boldsymbol{A}$*, as used in (5).*]

Test that the algorithm converges using an initial guess $\boldsymbol{c}_0$ that is close to the correct values.

[2.5 pts] If you start with $\boldsymbol{c}_0 = (1, 1, 1, 1)$, does the improved method converge to the correct answer?

[2.5 pts] Can you find some initial guess $\boldsymbol{c}_0$ for which even the improved method does not converge to the correct values? Does it converge to another local minimum or not converge at all?

[5 pts] If the synthetic data points have no error (i.e., $y_i = f(x_i; \boldsymbol{c})$ to within roundoff error), how many digits of accuracy in $\boldsymbol{c}$ can you obtain, starting with $\boldsymbol{c}_0 = (1, 1, 1, 1)$? How many steps do you need to achieve this accuracy.

4 [20 pts + 40 extra credit] 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:

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathbb{R}^n} \left\{ f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2 = \boldsymbol{x}\cdot\boldsymbol{x} = \textstyle\sum_{i=1}^n x_i^2 \right\} \\ \text{s.t.} \quad &(\boldsymbol{x} - \boldsymbol{x}_0)^T \boldsymbol{A} (\boldsymbol{x} - \boldsymbol{x}_0) = \textstyle\sum_{i,j=1}^n a_{ij} (x_i - x_{0,i}) (x_j - x_{0,i}) = 1 \quad . \end{aligned} \tag{6}$$

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!).

4.1 [20pts total] Analytical Solution

1. [5pts] Focus on the case $\boldsymbol{x}_0 = 0$. Consider a change of coordinates in which the matrix $\boldsymbol{A}$ is diagonal, i.e., compute the eigenvalue decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\star$ [*Hint: It is unitarily diagonalizable and all eigenvalues are positive*]. The geometric interpretation is that the unitary matrix $\boldsymbol{U}$ is a rotation matrix and the eigenvalues are the inverse squares of the semiaxes of the ellipsoid. Express $\boldsymbol{x}$ in the orthonormal basis formed by the eigenvectors of $\boldsymbol{A}$ and transform the optimization problem to the new basis, and use this to solve it analytically.
2. [5pts] As an example, consider minimizing $x_1^2 + x_2^2$ along the ellipse
$$x_1^2 + 2x_1x_2 + 3x_2^2 = 1. \tag{7}$$
Write down the matrix $\boldsymbol{A}$. Find the solution analytically (meaning with pen-and-paper), for example, using the solution in part 1 above, or maybe in part 3 below, and evaluate it numerically. [*Hint: The exact solution is* $x_1^\star = \frac{1}{\sqrt{2}} - \frac{1}{2}$ and $x_2^\star = \frac{1}{2}$].
3. [10pts total] Write the Lagrangian function $\mathcal{L}$ and the first-order optimality conditions.
[5pts] For the case $\boldsymbol{x}_0 = \boldsymbol{0}$, can you interpret the optimality conditions as an eigenvalue problem and solve it [*Hint: This is closely related to part 1 above*].

4.2 [Up to 40pts extra credit] Numerical Solution via the Penalty Method

Write a MATLAB (or other) program that implements Newton's method for minimizing the *penalty function* (6)

$$\min_{\boldsymbol{x}} \left\{ \mathcal{L}_\alpha = f(\boldsymbol{x}) + \alpha \left[h(\boldsymbol{x}) \right]^2 \right\} \tag{8}$$

for a given penalty parameter α and some reasonable initial guess, e.g., $\boldsymbol{x}^0 = \mathbf{1}$ (all ones) or $\boldsymbol{x}^0 = randn(n, 1)$. 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 $\boldsymbol{x}_0$. [*Hint: If you implemented Newton's method correctly the convergence will be quadratic, which should be easy to see if you print some convergence statistics.*]

1. [15pts] For the two-dimensional example (7) from part 2.1 above, 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, $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}\|_\infty \leq \varepsilon = 10^{-12}$ [*Hint: You may get faster convergence if you use the last known solution as an initial guess for the next* α].
   Plot the error in the solution to (8) 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?
2. [5pts] Apply the same approach as in part 1 above to the same matrix $\boldsymbol{A}$ but with $\boldsymbol{x}_0 = [0.1, -0.2]$ [*Hint: The solution is* $\boldsymbol{x}^\star = [0.126553521589213, 0.368363384837439]$].
3. [5pts] Apply the penalty method to a randomly-generated three-dimensional problem with $\boldsymbol{x}_0 = \mathbf{0}$ [*Hints: A random matrix* $\boldsymbol{A}$ *can be generated in MATLAB using the function gallery*$('randcorr', n)$*. Verify your answer against the analytical solution from part 1.*].
4. [15pts] You will observe that the convergence of the solution to problem (8), $\boldsymbol{x}(\alpha)$, to the solution of (6), $\boldsymbol{x}^\star = \boldsymbol{x}(\alpha \to \infty)$ is slow and a rather large α is required to get an accurate answer. One can accelerate the convergence by using a technique similar to Richardson's extrapolation. The idea is to postulate how $\boldsymbol{x}(\alpha)$ depends on α and then extrapolate to $\alpha \to \infty$. Assume that

   $$\boldsymbol{x}(\alpha) \approx \boldsymbol{x}^\star - \alpha^{-1} \boldsymbol{c}$$

   for some vector $\boldsymbol{c}$. As you change the penalty α_k, for example, $\alpha = \alpha_k = 10^k$, you can estimate $\boldsymbol{c} \approx \boldsymbol{c}(\alpha_k)$ from the last two solutions, $\boldsymbol{x}(\alpha_{k-1})$ and $\boldsymbol{x}(\alpha_k)$, and use that estimate to compute an improved estimate of $\boldsymbol{x}^\star$,

   $$\boldsymbol{x}^\star(\alpha_k) = \boldsymbol{x}(\alpha_k) - \alpha_k^{-1} \boldsymbol{c}(\alpha_k).$$

   For the example (7), compute the accelerated sequence $\boldsymbol{x}^\star(\alpha)$ for $k = 1, 2, \ldots$ and plot its error versus α. How fast does the accelerated sequence converge compared to the original sequence $\boldsymbol{x}(\alpha)$ computed in part 1 above?
   Note that this process can in principle be repeated recursively to improve the estimate even further!