Fall 2017: Numerical Methods I
Assignment 3 (due Oct. 19, 2017)

The same rules as for the previous assignment apply. Again, I recommend that you only work on the extra credit problem once you are done with the regular problems.

1. **[Normal equations, Givens rotations, 2+2pt]**

   (a) Let $A \in \mathbb{R}^{m\times n}$ with $m \geq n$. Show that A^TA is positive definite if and only if A has full column rank.[1]

   (b) Use Givens rotations to compute (with paper and pencil) the QR-factorization of the matrix

   $$A = \begin{bmatrix} 4 & 1 \\ -3 & 3 \\ 0 & 4 \end{bmatrix}.$$

   Specify how many and which Givens rotations you used. Specify R and write Q as product of these Givens rotations—no need to multiply them out to obtain Q.

2. **[Pseudo-inverse and QR, 2+1pt]** The pseudo inverse Z of a matrix $A \in \mathbb{R}^{m\times n}$ is (uniquely) defined through the following conditions (called Penrose axioms):

   $$\begin{aligned} ZA &= (ZA)^T, \\ AZ &= (AZ)^T, \\ ZAZ &= Z, \\ AZA &= A. \end{aligned}$$

   (a) Show that for a full-rank matrix $A \in \mathbb{R}^{m\times n}$, the solution operator $A^+ := (A^TA)^{-1}A^T$ occurring in the normal equations satisfies the Penrose axioms.

   (b) Compute the pseudo inverse of A from its QR-factorization.

3. **[Linear least squares and a modification for outliers 2+1+3pt]** We fit the nonlinear model function

   $$\Phi(t, \boldsymbol{x}) = x_1 + x_2 t + x_3 t^2 + x_4 \exp(t), \text{ with } \boldsymbol{x} = (x_1, x_2, x_3, x_4)^T$$

   to the data points (t_i, b_i), $i = 1, \ldots, m$, where $m \geq n := 4$. For given model parameters $\boldsymbol{x}$, the deviations between model and data are given by

   $$\Delta_i := \Phi(t_i, \boldsymbol{x}) - b_i.$$

   Using the least squares approach, we find the optimal model parameters as minimization over all model parameters $\boldsymbol{x} \in \mathbb{R}^n$ of the function

   $$\sum_{i=1}^{m} |\Delta_i|^2 = \|A\boldsymbol{x} - \boldsymbol{b}\|^2, \tag{1}$$

   with appropriately chosen $A \in \mathbb{R}^{m\times n}$, and $\boldsymbol{b} = (b_1, \ldots, b_m)^T$. As discussed in class, the optimal model parameters $\boldsymbol{x}$ depend on how the deviations are measured, and we will explore the difference

[1]This implies that the normal equations $A^TA\boldsymbol{x} = A^T\boldsymbol{b}$ has a unique solution if A has full column rank.

between the least squares approach and minimizing the sum of absolute values, i.e., the 1-norm of the vector of deviations:[2]

$$\sum_{i=1}^{m} |\Delta_i|. \tag{2}$$

We use noisy observation data generated as follows:[3]

```
t = 2*rand(20,1);
b = 1 + 0.8*t - 0.5*t.^2 + 0.3*exp(t) + 0.2*randn(20,1);
plot(t,b,'ro');
```

(a) Use the linear least squares approach to fit the model parameters $\boldsymbol{x}$ to the data points. Present a plot that shows the data points, your numerically found model and the true model (which you can see from how the data is generated when you ignore the noise term).

(b) Add the observation point $(t_{21}, b_{21}) = (1, 8)$ to the data set. This point, which could come from a corrupted measurement, is obviously an outlier. Again, fit the data with the least squares model and plot the resulting model curve. How does the added point influence your fitted model?

(c) To improve the stability of the model in the presence of corrupted measurements, let us try to minimize (2) instead of (1). To find conditions that must be satisfied by the minimizer $\bar{x}$ of (2), we would like to compute its derivative, which must be zero at a minimizer. However, the absolute value function is not differentiable. As a remedy, we replace (2) by the regularized problem

$$\sum_{i=1}^{m} \sqrt{|\Delta_i|^2 + \varepsilon} \tag{3}$$

with a small $\varepsilon > 0$. For $\varepsilon = 0$, (3) obviously coincides with (2). For every $\varepsilon > 0$, (3) can be differentiated, and the derivative is

$$G(\boldsymbol{x}) = A^T \begin{bmatrix} \frac{(A\boldsymbol{x}-\boldsymbol{b})_1}{\sqrt{(A\boldsymbol{x}-\boldsymbol{b})_1^2+\varepsilon}} \\ \vdots \\ \frac{(A\boldsymbol{x}-\boldsymbol{b})_m}{\sqrt{(A\boldsymbol{x}-\boldsymbol{b})_m^2+\varepsilon}} \end{bmatrix} = A^T D(\boldsymbol{x})(A\boldsymbol{x} - \boldsymbol{b}),$$

with $(\cdot)_i$ denoting the i-th component of a vector, and $D(\boldsymbol{x})$ is a diagonal matrix with diagonal entries $1/\sqrt{(A\boldsymbol{x} - \boldsymbol{b})_i^2 + \varepsilon}$, $1 \le i \le m$. To find the minimizer $\bar{x}$, we thus have to solve the nonlinear equation $G(\boldsymbol{x}) = 0$. Let us use the following fixed point method for doing that: Initialize $\boldsymbol{x}^0$, and compute $\boldsymbol{x}^k$, $k = 1, 2, \dots$ from

$$A^T D(\boldsymbol{x}^{k-1}) A\boldsymbol{x}^k = A^T D(\boldsymbol{x}^{k-1})\boldsymbol{b}.$$

Use this fixed point iteration with $\varepsilon = 0.05$ to fit the data that included the outlier. As above, plot the resulting model function and the measurements. What do you observe? How many iterations does the fixed point method need to converge until $\|G(\boldsymbol{x}^k)\| \le 10^{-6}$ for different values of ε? Try smaller values of ε (we want to solve the original problem (2) after all) and report the iteration numbers. What do you observe?

[2]Studying theoretical properties and algorithms related to minimization problems involving the 1-norm is a very active field of research, and is part of a field called *compressed sensing*.

[3]Please make sure that you generate this data only once for all the tasks below, as each new generation will give you different data due to the (pseudo-)random functions used.

4. **[Fixed point and Newton's method, 2+2+1+3pt]**

   (a) The function $f(x) := x\exp(x) - 1$ has only one root $\bar{x} \approx 0.5$. Consider the fixed point form

   $$x = \Phi(x) := x + kf(x) \quad \text{with } k \in \mathbb{R}.$$

   Determine k such that $\Phi'(0.5) = 0$[4] and solve the problem with the resulting fixed point method using $x_0 = 0.5$ as initialization. Terminate the fixed point iteration when the difference between consecutive iterates is less than 10^{-8} and report the number of iterations. Compare with the number of iterations required for convergence with another value of k of your choosing.

   (b) Let $x = F(x)$, with F continuously differentiable, be a fixed point form for computing the fixed point $\bar{x}$. Assume that $|F'(\bar{x})| > 1$, and thus the corresponding fixed point iteration diverges. Prove that the fixed point form

   $$x = F^{-1}(x)$$

   generates a convergent fixed point iteration if initialized close to $\bar{x}$.

   (c) Use Newton's method to find the root of $f(x) = 1/x + 2\ln(x) - 2$, for $x > 0$. Print a few iterates and observe the convergence of correct digits. Use results from problem 1, Assignment #2 to estimate the convergence rate.

   (d) We would like to compute the intersection points of the parabola $y - x^2 + 2 = 0$ and the ellipse $x^2/16 + y^2/4 - 1 = 0$. Implement Newton's method for the appropriate function $F : \mathbb{R}^2 \to \mathbb{R}^2$ to solve this problem. Plot the iterates as dots on top of the curves for three different initial points $\boldsymbol{x}^0 = (x^0, y^0) = (0.01, 3)$, $\boldsymbol{x}^0 = (-0.01, 3)$ and for $\boldsymbol{x}^0 = (2, -2)$. Do you find the same or different intersection points for these initializations? What convergence order do you observe for each iteration? Why is the result for the last initialization not contradict the theoretic convergence theory for Newton's method?

5. **[Nonlinear least squares, 3pt]** You are given the data points[5] (t_i, f_i) for $i = 1, \dots, 10$ in the interval [-5,5]. You are told that these points should be fitted with a sum of two Gaussians of the following form:

   $$\varphi(t) = \sum_{j=1}^{2} \omega_j \exp(-\frac{1}{2\sigma_j^2}(t - \tau_j)^2).$$

   Use a nonlinear least squares formulation to find the best-fitting parameters $\boldsymbol{x} = (\omega_1, \sigma_1, \tau_1, \omega_2, \sigma_2, \tau_2)^T$ using a Gauss-Newton method. To be precise, find $\boldsymbol{x}$ such that

   $$\min_{\boldsymbol{x} \in \mathbb{R}^6} \sum_{i=1}^{10} (\varphi_1(t_i; \boldsymbol{x}) - f_i)^2.$$

   Here, we have used the notation $\varphi_1(t) = \varphi_1(t; \boldsymbol{x})$ to emphasize the dependence of the model function on the parameters $\boldsymbol{x}$. Use $\boldsymbol{x}^0 = (1, 1, -3, 1, 1, 1.5)$ as the initial guess. What kind of convergence rate do you observe? How does this compare with the theoretical convergence speed for the Gauss-Newton method? Hand in the code of your implementation.

[4]Based on the theory for fixed point iterations, these methods converge faster the stronger contractions they are, which means the Lipschitz constant (or, somewhat equivalently for smooth functions, the norm of the derivative) is small. Thus we are trying to pick k such that the absolute value of the derivative is as small as possible.

[5]You can download the data points from `http://www.cims.nyu.edu/~stadler/num1/material/data.m`.

6. **[Matrix factorization reuse and reordering, 2pt extra credit]** This problem describes a simple numerical scheme for approximating the solution to a partial differential equation—a topic you will learn more about when you attend the second part of this class in the spring. Here, his only serves as an example for reusing matrix factorizations and the role of the order of the unknowns for the sparsity of the Cholesky factors.[6]

   Consider the following partial differential equation (PDE), which describes the diffusion of heat over time. The equation is know as the the heat equation:

$$\begin{aligned} u_t - \kappa\,\Delta u &= 0, \quad \text{in } \Omega \times [0, T_f], \\ u(\boldsymbol{x}, 0) &= u_0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \\ u &= 0, \quad \text{on } \partial\Omega \times [0, T_f]. \end{aligned}$$

   Here $\Omega = (0,1) \times (0,1)$ is the physical domain, $\kappa > 0$ is the diffusion constant, and T_f is the final time. For each $(\boldsymbol{x}, t) \in \Omega \times [0, T_f]$, $u = u(\boldsymbol{x}, t)$ is the temperature at the point $\boldsymbol{x}$ at time t.

   To solve the problem numerically, we need to discretize the PDE over space and time:

   Spatial discretization. The discretization of the operator $\mathcal{L}u = -\Delta u = -u_{xx} - u_{yy}$ can be done using finite differences, i.e., derivatives are approximated using difference quotients (this is somewhat similar to problem 5 on the assignment 1. Let A be the spatial discretization of the operator $-\Delta$, and let the vector $\boldsymbol{u}(t)$ be the discretized solution at time t. That is, if we let n_x be the number of spatial grid points in each spatial coordinate direction, we have $\boldsymbol{u}(t) \in \mathbb{R}^N$, with $N = (n_x - 1)^2$.

   Temporal discretization. Next, we consider discretizing over time. The semi-discrete (discretized in space only) problem is given by

$$\boldsymbol{u}'(t) + A\boldsymbol{u}(t) = 0, \quad \boldsymbol{u}(0) = \boldsymbol{u}_0, \quad t \in [0, T_f]. \tag{4}$$

   This is a large system of ordinary differential equations for the spatial unknowns over time. This initial value problem can be solved in many ways. We consider the so-called implicit Euler method. Let $\Delta_t = T_f/n_t$, and $t_k = k\Delta t$, $k = 0, \ldots, n_t$. Letting $\boldsymbol{u}^k \in \mathbb{R}^N$ denote (approximation to) $\boldsymbol{u}(t_k)$, the implicit Euler discretization gives

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k - \Delta t A \boldsymbol{u}^{k+1}, \quad k = 0, \ldots, n_t - 1,$$

   which we can rearrange to obtain

$$(I + \Delta t A)\boldsymbol{u}^{k+1} = \boldsymbol{u}^k, \quad k = 0, \ldots, n_t - 1.$$

   Note that you have to solve a linear system at every time step. The coefficient matrix, however, does not change.

   A basic solver for this problem is available for download.[7] In the solver, you will find the specific choices for the diffusion coefficient, the initial state, and the final time.

[6] This problem was put together by Alen Alexanderian from NC State—I appreciate that he lets me use it.

[7] These file is written in MATLAB, and besides the solver, there is also a file that shows the time evolution as a movie. The two files can be downloaded from http://www.cims.nyu.edu/~stadler/num1/material/heat.zip You can run execute them as follows:

```
> [U_array ti Xi Yi] = solve_heat_2D_implicit;
> animate_sol(ti, Xi, Yi, U_array);
```

Your task. Your task is to make the provided solver run faster. In particular, instead of doing a linear solve using the backslash operator at each step, implement the following: (i) factorize the matrix $(I + \Delta t A)$ first (before time-stepping begins) by computing its Cholesky factor, and then perform the linear solves during time-stepping using the computed Cholesky decomposition[8]; (ii) use MATLAB's `symamd` command in conjunction with `chol` to reorder the unknowns and thus enhance the sparsity of the Cholesky factor, which you will then use to solve the system.[9]

Your solution report for this problem should include the following:

- a brief description of what you did to change the computer code.
- computational times for solving the PDE, when (i) you only use backslash for the linear solves (i.e., the original code), (ii) you use `chol` to precompute the Cholesky factor, and (iii) you use `chol` and `symamd` approach.
- plots of the initial state (u at $t = 0$), u at a couple of intermediate times, and u at the final simulation time, computed using your fastest solver. Note that for each fixed time t_0, $u = u(\boldsymbol{x}, t_0)$ is a function of $\boldsymbol{x} = (x, y) \in \Omega$.
- a brief discussion of the results.

[8] That is, in each iteration you use one forward and backward substitution—the backslash command recognizes that the matrices are tridiagonal and will automatically call a triangular solver for you. Transpose and save the Choleski factor before the loop—that should be faster than always transposing it inside the loop

[9] My slides from September 21 contain at the end some experiments I showed in class, that illustrate reordering using these MATLAB commands and the resulting reduced fill in. Also, you should make use of MATLAB's help functions. You can basically work in reordered unknowns, which is fast. You must order them back before outputting them, of course.