Homework 2

Due: Wednesday 11:59pm, Oct 10, 2024, on Gradescope

Notes: Please read the syllabus for information on programming expectations, the late homework policy, and the collaboration policy. Please submit both the written and coding portions of your homework on Gradescope. The written portion should be submitted as a PDF (prepared however you like, but LaTeX is excellent), with plots and tables in the appropriate places. Please mark the location of each problem in your Gradescope submission—this makes it much easier for us to grade.

Plotting and formatting: Put thought into presenting your data and results in the clearest way possible. Think about what information you want to convey, and how to convey it most effectively. Sometimes, using a table can be useful, but at other times data is best presented using a plot. All tables and plots should be accompanied by some discussion of what we are intended to learn from them.

Choose proper ranges and scales (e.g., semilogx, semilogy, loglog), always label axes, and give meaningful titles. Make sure your axis numbering and labels are large enough to be readable—getting this right will require some exploration in MATLAB, or whatever language you are using. If you do print numbers, in MATLAB for example, use `fprintf` to format the output nicely. Use `format compact` and other format commands to control how MATLAB prints things. When you create figures using MATLAB (or Python or Julia), please try to export them in a vector graphics format (.eps, .pdf, .dxf) rather than raster graphics or bitmaps (.jpg, .png, .gif, .tif). Vector graphics-based plots avoid pixelation and thus look much cleaner.

Code: Try to write clean, concise, and easy-to-read code, with meaningful variable names. Your code must be well-commented. Every line of code should be explained by comments inside the code, unless it is absolutely self-explanatory. Commenting your code is like "showing your work", and grading will take this into account in a similar manner.

You should submit five code files for this assignment: `prob1.m`, `prob2.m`, `lu_nopivot.m`, `forwardsub.m`, and `backsub.m`.

---

1. Let $f(x) = e^x - x^2 - 6x - 9$ and $g(x) = 2\log(x+3)$, and assume $x > -3$.
   (a) **[2 pts]** Show that the roots of $f(x)$ are identical to the fixed points of $g(x)$.
   (b) **[2 pts]** Plot $y = g(x)$ and $y = x$, and indicate all fixed points. You don't need to calculate them.
   (c) **[8 pts]** Argue that $g(x)$ has a fixed point in $[3, 4]$ (to do this, you are free to evaluate the exponential or log functions numerically, but you cannot simply refer to the plot). Then use Thm. 4.5.1 in Greenbaum & Chartier to argue that fixed point iteration with an initial guess in that interval must converge to a fixed point.

(d) **[8 pts]** Write a program `prob1.m` which implements fixed point iteration for $g(x)$ with initial guess $x_0 = 3.5$. Use this to obtain the root of $f(x)$ in $[3, 4]$ to within a residual $|f(x)| < 10^{-10}$, and write down the result with at least ten significant digits. Report the number of iterations required to obtain this result.

2. Newton's method can be generalized to vector-valued functions of multiple variables: for example, $f : \mathbb{R}^n \to \mathbb{R}^n$. To do this, we use the first two terms of the generalization of Taylor's theorem, given by

$$f(x) \approx f(x_0) + J(x_0)(x - x_0),$$

where

$$J(x_0) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

is the Jacobian matrix. This is the linearization of f about the point x_0. The approximate equality can be made precise by including a generalization of the remainder term in Taylor's theorem, but we will not need that here.

(a) **[4 pts]** A point x^* is a zero of $f : \mathbb{R}^n \to \mathbb{R}^n$ if $f_1(x^*) = f_2(x^*) = \cdots = f_n(x^*) = 0$. Using the linearization above, write down a generalization of Newton's method to the vector-valued, multivariate case. Remember that the idea of Newton's method is to replace the function f at the current iterate x_k by its linearization, and guess the next iterate x_{k+1} as the zero of the linearization.

(b) **[4 pts]** Consider $f \in \mathbb{R}^2 \to \mathbb{R}^2$ given by

$$\begin{aligned} f_1(x_1, x_2) &= 5x_1 + x_1 x_2^2 + \cos^2(3x_2) - 1 \\ f_2(x_1, x_2) &= e^{2x_1 - x_2} + 4\sin(x_2) - 2. \end{aligned}$$

Create a 3D plots of $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$ on the domain $x_1, x_2 \in [-1, 1]$ using the following MATLAB code:

```
f1 = @(x1,x2) 5*x1 + x1.*x2.^2 + cos(3*x2).^2 - 1;
f2 = @(x1,x2) exp(2*x1-x2)+4*sin(x2) - 2;
x = linspace(-1,1,30);
[X,Y] = meshgrid(x);
figure(1);
mesh(X,Y,f1(X,Y),'FaceAlpha','0.5','FaceColor','red'); hold on;
mesh(X,Y,f2(X,Y),'FaceAlpha','0.5','FaceColor','green');
mesh(X,Y,zeros(size(X)),'FaceAlpha','0.5','FaceColor','blue');
xlabel('x'); ylabel('y'); zlabel('z');
hold off
```

Please understand what each of these lines does by looking at the MATLAB documentation online; generating clear 3D plots can be challenging, and some tricks have been used here to make the plot clearer. Include a copy of this plot and explain what it shows. Based on the plot, where in the domain do you think f has a zero? Give a rough (eyeball) approximation of the location of the zero. To do so, it might help to use the rotate feature in MATLAB's plotting window.

(c) **[10 pts]** Write a program `prob2.m` which implements your vector-valued, multivariate Newton's method for the function f, and finds a zero of f in the domain given above. Use your eyeball approximation of the zero as the initial guess. Run Newton's method until you find an approximate zero x such that $||f(x)|| < 10^{-10}$, where $||\cdot||$ denotes the Euclidean norm in $\mathbb{R}^2$. Report your result, the residual error, and the number of iterations required by Newton's method to achieve the result. Note that a linear system $Ax = b$ can be solved in MATLAB using the code `x = A\b`; this is referred to as the "backslash" operation in MATLAB.

3. Let us investigate the numerical stability of two methods of computing derivatives.

(a) **[3 pts]** Consider a function $f \in C^2$ with domain $[a, b]$, and assume that f'' is bounded on $[a, b]$. Using Taylor's theorem with remainder, argue that the error of the *finite difference approximation*

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

is $\mathcal{O}(h)$; that is, the error is bounded by Ch, where C is a constant. We say that the approximation is first-order accurate, and refer to the error as the *truncation error* or *discretization error.*

(b) **[4 pts]** Suppose this finite difference approximation is computed in finite precision arithmetic with machine precision ε. For simplicity, assume that rounding errors are only made in the calculation of $f(x+h)$ and $f(x)$. Show that the absolute error due to rounding is bounded by $2\varepsilon M/h$, where we assume $|f(x)| \leq M$ for all $x \in [a, b]$. Hint: If rounding errors are only made in the calculation of $f(x+h)$ and $f(x)$, then we can assume the value obtained by the computer is $(f(x+h)(1+\delta_1) - f(x)(1+\delta_2))/h$, where $|\delta_1|, |\delta_2| \leq \varepsilon$.

(c) **[3 pts]** If we assume that the discretization error is given precisely by $e_{\text{disc}}(h) = h$, and that the roundoff error is given precisely by $e_{\text{round}}(h) = \varepsilon/h$, then determine approximately how accurately $f'(x)$ can be computed using this finite difference approximation by minimizing the sum $e_{\text{disc}}(h) + e_{\text{round}}(h)$ of the errors. What is the best value of h to use?

(d) **[10 pts]** Let $f(x) = -4+3x-2x^2+x^3$. Write a program to approximate $f'(1)$ using the finite difference method above, for a given value of h. Using that $f'(1) = 2$, plot the absolute error of the finite difference approximation as a function of h for values of h ranging from 10^{-16} to 1 (Hint: Use a loglog plot). As dashed lines, also plot the lines $y = \frac{|f''(1)|}{2}h$ and $y = \frac{2\varepsilon|f(1)|}{h}$ (Bonus **[3 points]**: Explain where the slopes of these lines come from.). Note that $\varepsilon \approx 2.22 \times 10^{-16}$ in double precision. Explain what the plot shows. Is it roughly consistent with your result above? Is it possible to obtain an approximation of $f'(1)$ to within an error 10^{-12} using this method?

(e) [**3 pts**] There is nothing inherently ill-conditioned about the computational task of differentiating a cubic polynomial; the finite difference method is simply numerically unstable due to catastrophic cancellation in the subtraction step. Perhaps we can find a better method. Let us represent a cubic polynomial by a vector in $\mathbb{R}^4$ containing its four coefficients: for example, $f(x) = -4 + 3x - 2x^2 + x^3$ is represented by the vector $v = (-4, 3, -2, 1)^T$. Show that the matrix

$$D = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

computes the coefficients of the derivative of a cubic polynomial, giving a vector in $\mathbb{R}^3$ representing the coefficients of the derivative, which is a quadratic polynomial. We call D a *differentiation matrix*.

(f) [**2 pts**] Consider the vector $e^T = (1, 1, 1)^T$. Show that if $v \in \mathbb{R}^3$ contains the coefficients of a quadratic polynomial $f(x) = a_0 + a_1 x + a_2 x^2$, $v = (a_0, a_1, a_2)^T$, then $e^T v = f(1)$.

(g) [**2 pts**] Compute the row vector $d^T = e^T D$, and argue that $d^T v = f'(1)$, where $v \in \mathbb{R}^4$ contains the coefficients of a cubic polynomial $f(x)$.

(h) [**3 pts**] In MATLAB, form d and v for the cubic polynomial $f(x) = -4 + 3x - 2x^2 + x^3$ given above, and compute $f'(1)$ by this method. Report the error. What happens if we add a bit of noise at the level of machine ε to the coefficients, for example via `v = v + eps*rand(4,1);`? Is this procedure to differentiate a cubic polynomial numerically stable?

4. (a) [**5 pts**] Modify the simple Gaussian elimination code `gausselim_nopivot.m` to compute the LU factorization of a given matrix, without pivoting. Put your function in a file `lu_nopivot.m`. The function should return a lower-triangular matrix L and an upper-triangular matrix U.

(b) [**6 pts**] Write a function to solve $Lx = b$ by forward substitution for lower-triangular L, and a function to solve $Ux = b$ by back substitution for upper-triangular U. Put these in files `forwardsub.m` and `backsub.m`.

(c) [**4 pts**] Use your code to solve the linear system

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

by first LU factorizing the matrix, and then solving two triangular linear systems. The exact solution is given by $x = (1, 1, 1)^T$. Report your solution and the error. You can report the error in the Euclidean norm $||x - x_{\text{exact}}||$, where the Euclidean norm $||\cdot||$ can be computed using MATLAB's `norm` function.

(d) [**4 pts**] Use your code to solve the linear system

$$\begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

The exact solution is given by $x = (1,1)^T$ to within the double machine precision. Report your solution and the error (in the Euclidean norm). For this system, we need a code which performs pivoting!

(e) **[3 pts]** Use your code to solve a random 1000×1000 linear system $Ax = b$ generated using

```
A = 2*rand(1000)-1;
A = A/norm(A);
xtrue = 2*rand(1000,1)-1;
xtrue = xtrue/norm(xtrue);
b = A*xtrue;
```

Here we construct a random matrix A and vector x_{true} with entries in $[-1,1]$, and then divide by their norms to obtain a matrix and vector of reasonable magnitude. We then generate the vector $b = Ax_{\text{true}}$. In this way, we have generated a vector b for which the solution to $Ax = b$ is known. Report the error of your solution in the Euclidean norm. Does your solver produce a solution accurate to near machine ε?

5. **[10 pts]** Show that the forward substitution algorithm requires approximately n^2 simple operations (additions, subtractions, multiplications, divisions). (Hint: Ignore terms of order $\mathcal{O}(n)$.)