Homework 2

January 27, 2024

Please submit your HW on Canvas; include a PDF printout of any code and results, clearly labeled, e.g. from a Jupyter notebook. For coding problems, we recommend using Julia, but you can use other languages if you wish. It is due Friday February 2nd by 11:59pm EST.

Problem 1

Continue reading the draft course notes (linked from `https://github.com/mitmath/matrixcalc/`). Find another place that you found confusing, in a different chapter from your answer in homework 1, and write a paragraph explaining the source of your confusion and (ideally) suggesting a possible improvement.

(Any other corrections/comments are welcome, too.)

Problem 2

The course notebook on finite differences includes, without derivation, a mysterious four-line Julia function called `stencil` that can compute finite-difference rules for an arbitrary number of points. In particular, if you want to compute the m-th derivative of a smooth (analytic) scalar function $f(x)$ at x_0, it returns the weights w_k of an n-point ($n > m$) finite-difference rule from evaluating f at points x_k for $k = 1 \ldots n$:

$$f^{(m)}(x_0) \approx \sum_{k=1}^{n} w_k f(x_k)$$

by solving the system of equations $Aw = e_{m+1}$, where $e_j \in \mathbb{R}^n$ is the Cartesian unit vector in the j-th direction and A is an $n \times n$ matrix with entries $A_{ij} = \frac{(x_j - x_0)^{i-1}}{(i-1)!}$. Here, you will analyze and derive this technique.

1. Let $x_0 = 0$. According to the notes, you can then compute $f^{(m)}(y) \approx \frac{1}{h^m} \sum_{k=1}^{n} w_k f(y + hx_k)$ for an arbitrary point y and an arbitrary step-size scaling factor h (which can be made smaller and smaller to reduce truncation errors). Derive this formula (via the chain rule).
2. Evaluate the `stencil` function (or its equivalent in another language if you want to re-implement it) for $x_0 = 0$ and $x = [0, 1]$ with $m = 1$. Check that the resulting w corresponds to the familiar forward-difference approximation from class. (You could alternatively solve $Aw = e_{m+1}$ analytically here, since it is 2×2.) In Julia, you can pass `0//1` for x_0 and it will return exact rational weights.
3. Now evaluate it for $x_0 = 0$ (`0//1` in Julia for exact results) and $x = [0, 1, 2, 3]$ with $m = 1$, i.e. using $n = 4$ equally spaced points $\geq x_0$. Use the resulting weights, in the formula scaled by h as above, to approximate the derivative $f'(1)$ for $f(x) = \sin(x)$, and plot the relative error (compared to the exact derivative) as a function of h on a log–log scale, similar to the course notebook. What power law in h does the truncation error (approximately) seem to follow? That is, what is the "order of accuracy"?
4. Derive the stencil equation $Aw = e_{m+1}$ above: write out the first $n - 1$ terms of the Taylor series for $f(x_0 + \delta x)$, and try to find a linear combination of this series evaluated at $\delta x = x_k - x_0$ for $k = 1 \ldots n$ in such a way that you obtain $f^{(m)}(x_0)$.

Problem 3

Consider the following system $g(x, p) = 0$ of two nonlinear equations in two variables $x \in \mathbb{R}^2$, parameterized by three parameters $p \in \mathbb{R}^3$:

$$g(x, p) = \begin{pmatrix} p_1 x_1^2 - x_2 \\ x_1 x_2 - p_2 x_2 + p_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

For $p = [1, 2, 1]$ this has an exact solution $x = [1, 1]$.

1. What are the Jacobian matrices $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial p}$? (That is, as defined in class, the linear operators such that $dg = \frac{\partial g}{\partial x} dx + \frac{\partial g}{\partial p} dp$ for any change dx and dp, to first order.)

2. In Julia (or Python etc.), implement Newton's method to solve $g(x, p) = 0$ from a given starting guess x. Using $p = [1, 2, 1]$, start your Newton iteration at $x = [1, 2]$ and show that it converges rapidly to $x = [1, 1]$ (it should converge to machine precision in < 10 steps).

3. Now, consider the function $f(p) = \|x(p)\|$, where the "implicit function"[1] $x(p)$ is a solution of $g(x, p) = 0$. Given a solution $x(p)$ for some p, explain how to compute ∇f (see the adjoint-method notes from lecture 5). Implement this algorithm in Julia (etc.), and validate it against a finite-difference approximation for $p = [1, 2, 1]$, $x(p) = [1, 1]$, and a random small δp (solving for $x(p + \delta p)$ by Newton's method starting from $x(p)$).

Problem 4

1. Suppose that $f(A)$ is a function that maps (real) $m \times m$ matrices to $m \times m$ matrices, and its derivative is the linear operator $f'(A)[dA]$. For the Frobenius inner product $\langle X, Y \rangle = \operatorname{trace}(X^T Y)$, it turns out that we typically have

$$\langle X, f'(A)[Y] \rangle = \left\langle f'(A^T)[X], Y \right\rangle,$$

which conceptually corresponds to "transposing" the linear operator $f'(A)^T = f'(A^T)$. Your job is to show this.

   (a) Show this for $f(A) = A^n$ for any $n \geq 0$. (Hint: From the product rule, it is easy to see that $f'(A)[dA] = \sum_{k=0}^{n-1} A^k dA\, A^{n-1-k}$; we've already seen this explicitly for several n. Combine this with the cyclic rule for the trace.)

   It immediately follows that this identity also works for any $f(A)$ described by a Taylor series in A (any "analytic" f), such as e^A.

   (b) Show this for $f(A) = A^{-1}$.

   (You can then compose the above cases to show that it works for any $f(A) = p(A)q(A)^{-1}$ for any polynomials p and q, i.e. for any rational function of A. You need not do this, however.)

2. Consider the function $f(A) = \det(\exp(A))$.

   (a) Write $f'(A)[dA]$ in terms of $\exp'(A)[dA]$. (You learned how to compute $\exp'$ in pset 1.)

   (b) Using the identity from the previous part, write ∇f in a way that can be evaluated efficiently ("reverse mode") using only one or two evaluations of exp (and/or $\exp'$) and det, independent of the size of A.

3. Check your answer from the previous part in Julia (or Python etc.): choose a random 5×5 `A=randn(5,5)` and a random small `dA=randn(5,5)*1e-8`, compute $df = f(A + dA) - f(A)$ and ∇f (at A), and verify that $df \approx \langle \nabla f, dA \rangle$. Compute $\exp'(A)[dA]$ using the same technique as in pset 1.

[1] This $x(p)$ can be defined uniquely in some neighborhood of a root like the one above, thanks to the implicit-function theorem.