Homework 2

January 25, 2025

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 January 31 by 11:59pm EST.

Problem 1 (10 points)

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 (5+5+5 points)

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 $x = [1.2, 1.3]$ 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 notes from lecture 4). Implement this algorithm 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 $x(p)$ and checking that $(\nabla f)^T \delta p \approx \delta f$).

Problem 3 (10 points)

In pset 1, you checked numerically that if $Aq_k = \lambda_k q_k$, where $A = A^T$ is a real-symmetric matrix and $q_k^T q_k = 1$, then $\nabla \lambda_k(A) = q_k q_k^T$. We will show a nice derivation of this in class.

Alternatively, you should be able to derive this by implicit differentiation: since λ_k is a root $g = 0$ of $g(\lambda, A) = \det(A - \lambda I)$, apply the determinant-gradient formula to show that $\nabla \lambda_k(A) = q_k q_k^T$.

Hint: the determinant gradient involves the adjugate matrix $\operatorname{adj}(M)$, which is normally obtained as $M^{-1} \det M$, but the latter formula doesn't work if M is singular. Instead, write the adjugate in terms of the diagonalization of A and take the limit $\lambda \to \lambda_k$ in order to obtain a formula that works when the determinant is zero.

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

Problem 4 (5+5+5 points)

In class, we computed the Jacobian of the double-pendulum solution with respect to the initial conditions, using forward-mode differentiation.

1. Describe the differential equations and initial conditions you would solve to instead compute the Jacobian of the solution u with respect to the lengths L_1 and L_2 of the pendulums. (As in class, you can rely on AD to differentiate the right-hand side $f(u,p,t)$ as needed; you need not write out these derivatives explicitly.)
2. Implement your algorithm (either modifying the Julia code from class or re-implementing in the language of your choice). Plot $\partial\theta_2/\partial L_1$ and $\partial\theta_2/\partial L_2$ for the same initial conditions $[\theta_1(0),\theta_2(0),\dot\theta_1(0),\dot\theta_2(0)] = [2,2,0,0]$ and timespan $t \in [0,10]$ as in class.
3. Check your answer for $\partial\theta_2/\partial L_1$ at $t=1$ against a finite-difference approximation, and verify that a few digits match.

Problem 5 (10 points)

Consider the following Julia function, acting on vectors $x, y, z \in \mathbb{R}^n$:

```
function f(x,y,z)
    a = (x'y) * z
    b = sin.(a) + x .* exp.(z)
    c = log.(b)
    d = (b'b) * b
    e = b
    w = e * sin(c'd)
    return w
end
```

Draw the computational graph of this function (as in the course notes, section 8.3), labelling the nodes with the variable names and the edges with the partial derivatives. Using this, write down the Jacobian $f' = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \end{bmatrix}$ in two ways: as it would be computed in forward mode and in reverse mode.

Problem 6 (6+6+4 points)

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 $f(A) = \det(A + \exp(A))$.

   (a) Write $f'(A)[dA]$ in terms of $\exp'(A)[dA]$ and other standard matrix operations. (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.