Homework 2

January 24, 2026

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 30th 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 points)

Do Problem 7.1 from the course notes, which is based on a common task in multivariate statistics.

Problem 3 (4+4+4+4 points)

Let $f(A)$ be a function that maps $m \times m$ matrices to $m \times m$ matrices. Recall that its derivative $f'(A)$ is a linear operator that maps any change δA in A to the corresponding change $\delta f = f(A + \delta A) - f(A) \approx f'(A)[\delta A]$, to first order in δA.

In this problem, you will study and prove a remarkable identity (Mathias, 1996): if $f(A)$ is sufficiently smooth,[1] then for *any* δA (not necessarily small!) the following formula holds *exactly*:

$$f\left(\underbrace{\begin{bmatrix} A & \delta A \\ & A \end{bmatrix}}_{M}\right) = \begin{bmatrix} f(A) & f'(A)[\delta A] \\ & f(A) \end{bmatrix}.$$

That is, one applies f to a $2m \times 2m$ "block upper-trianguar" matrix M (blank lower-left = zeros), and the desired derivative is in the upper-right $m \times m$ corner of the result $f(M)$. (Note: please do your *own* derivation here, don't just look it up.)

1. Check this identity numerically against a finite-difference approximation $f(A + \delta A) - f(A)$, which should match the exact $f'(A)[\delta A]$ to a few digits, for $f(A) = \exp(A)$ (the **matrix exponential** e^A, computed by `exp(A)` in Julia, or `expm` in Scipy or Matlab—do *not* use the elementwise exponential!), for a random 3×3 matrix `A = randn(3,3)` and a random small perturbation `dA = randn(3,3) * 1e-8`. Note that you can make the block matrix above in Julia by `using LinearAlgebra` followed by `M = [A dA; 0I A]`, and you can extract an upper-right corner by (*e.g.*) `M[1:3,4:6]`.

   It is also worth verifying (by a finite-difference check) that the derivative of the matrix exponential is *not* simply multiplication by e^A (on either the left or right or both): $(e^A)'[dA] \neq e^A\, dA$ or $dA\, e^A$ or $e^{A/2}\, dA\, e^{A/2}$, unlike the scalar case.

2. Prove the identity by explicit computation for the cases: $f(A) = I$, $f(A) = A$, $f(A) = A^2$, and $f(A) = A^3$ (the latter two derivatives were computed in class). (Two of these are trivial! This is "bargain-basement induction": do a few small examples and see the pattern.)

3. Prove the identity for $f(A) = A^n$ for any $n \geq 0$ by induction: assume it works for A^{n-1} and show using the product rule that it therefore must work for A^n. (You already proved the trivial $n = 0$ base case in the previous part.)

[1]The result is easiest to show when $f(A)$ has a Taylor series (is "analytic"), and in fact you will do this below, but Higham (2008) shows that it remains true whenever f is $2m - 1$ times differentiable, or even just differentiable if A is "normal" ($AA^T = A^T A$).

Remark: Once it works for any A^n, it immediately follows that it works for any $f(A)$ described by a Taylor series, such as $\exp(A) = I + A + A^2/2 + A^3/6 + \cdots + A^n/n! + \cdots$, since such a function is just a linear combination of A^n terms.

4. Prove the identity for $f(A) = A^{-1}$: since we know (from class) that $f'(A)[\delta A] = -A^{-1}\,\delta A\,A^{-1}$, plug this into the right-hand side of the formula above and show that it is the inverse of M (multiply by M and show you get I).[2]

Problem 4 (5+5 points)

Implement Newton's method on the characteristic-polynomial function $f(z) = \det(zI - A)$ to find an eigenvalue of the $m \times m$ matrix A (a root λ where $f(\lambda) = 0$), using the formula for the derivative of a determinant from class.

1. Show that each step of Newton's method can be implemented via a single matrix inversion (without knowing the eigenvalues of A).[3]

2. Implement Newton's method using your algorithm (using some programming language with a linear-algebra library for matrix inversion etcetera) for the 3×3 matrix
$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix},$$
starting from an initial guess $z = \operatorname{trace} A = 15$. Compare to your library's built-in eigenvalues function—how many iterations are needed to get one of the eigenvalues to at least 10 significant digits?

Problem 5 (10 points)

In class (and in the course notes, chapter 9), we went over the example of backpropagation in a 3-layer neural network (NN). Evaluation of the neural network for inputs v_1 produced "hidden layer" data v_2 and v_3, and finally an output v_4 that was fed into a loss function $L(v_4)$. To compute the gradient with respect to the NN parameters x (e.g. weights and biases), evaluated in reverse mode, we first derived a "backwards recurrence" ("adjoint" recurrence) $w_3 = \nabla_{v_4} L \to w_2 = \frac{\partial F_3}{\partial v_3}^T w_3 \to w_1 = \frac{\partial F_2}{\partial v_2}^T w_2$, from which we obtained the gradient
$$(\nabla_x \text{loss})^T = \sum_{k=1}^{3} w_k^T \frac{\partial F_k}{\partial x}.$$
(And there were further simplifications if the layers had disjoint parameters x_k, were of specific forms, etcetera.)

Suppose that instead our loss function is
$$\text{loss} = L(v_4) + \alpha \|v_3\|^2$$
so that it depends on *both* the final output v_4 and on the "hidden" layer v_3 with some scalar strength $\alpha > 0$. (Loss functions that depend on the hidden layers are common in machine learning, e.g. as regularizations to prevent the hidden values from becoming too big.) Give the modified backpropagation algorithm for ∇_xloss in this case—does the backwards recurrence for w_k change, or the final formula for the gradient in terms of w_k, or both?

[2] This is a special case of the famous "Schur complement" formula for the inverse of a 2×2 block matrix.

[3] In practice, there are ways to accelerate this further by preprocessing A, for example by putting A in "upper-Hessenberg" form. See `https://github.com/JuliaLinearAlgebra/GenericLinearAlgebra.jl/issues/167`. But you need not do this here—this is not a numerical linear-algebra class!