Homework 2 Solutions

February 2, 2025

See also the accompanying Julia notebook for computational solutions.

Problem 1 (5 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.)

Solution:

Student-dependent, but full marks if clearly written and explained.

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 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 $x(p)$).

Solution:

1. $\frac{\partial g}{\partial x} = \begin{pmatrix} 2p_1 x_1 & -1 \\ x_2 & x_1 - p_2 \end{pmatrix}$, $\frac{\partial g}{\partial p} = \begin{pmatrix} x_1^2 & 0 & 0 \\ 0 & -x_2 & 1 \end{pmatrix}$.
2. See accompanying Julia notebook.
3. As explained in the lecture-5 slides and the course notes, $df = -f'(x) \left(\frac{\partial g}{\partial x}\right)^{-1} \frac{\partial g}{\partial p} dp = (\nabla f)^T dp$. Here, $f = \|x\| \implies f' = \frac{x^T}{\|x\|}$ (from pset 1), so we can transpose to obtain:

$$\boxed{\nabla f = -\left(\frac{\partial g}{\partial p}\right)^T \left[\underbrace{\left(\frac{\partial g}{\partial x}\right)^{-T} \frac{x}{\|x\|}}_{v} \right]},$$

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

where the brackets indicate that we want to compute it in the order shown: first compute the "adjoint" solution v (similar in cost to a single step of Newton's method), and then multiply it by $\left(\frac{\partial g}{\partial p}\right)^T$, all evaluated at the current p and $x(p)$.

See the accompanying Jupyter notebook to solve this problem numerically and validate it against a finite-difference solution.

Although you were *not required* to do so, in this particular case, we can solve everything analytically. For $p = [1, 2, 1]$, we have $x = [1, 1]$ and hence $\frac{\partial g}{\partial x} = \begin{pmatrix} 2 & -1 \\ 1 & -1 \end{pmatrix}$, giving (using the standard formula for the inverse of a 2×2 matrix, noting that this matrix has determinant -1):

$$\nabla f = -\frac{1}{\sqrt{2}} \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix}}_{\left(\frac{\partial g}{\partial p}\right)^T} \left[\underbrace{\begin{pmatrix} 1 & 1 \\ -1 & -2 \end{pmatrix}}_{\left(\frac{\partial g}{\partial x}\right)^{-T}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right] = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 2 \\ -3 \end{pmatrix} = \boxed{\frac{1}{\sqrt{2}} \begin{pmatrix} -2 \\ -3 \\ 3 \end{pmatrix} \approx \begin{pmatrix} -1.414214 \\ -2.121320 \\ 2.121320 \end{pmatrix}}.$$

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.

Solution:

For our implicit function $\lambda(A)$ defined by $g(\lambda, A) = 0$, we use $dg = \frac{\partial g}{\partial A}[dA] + \frac{\partial g}{\partial \lambda} d\lambda$, which as in class we can solve for

$$d\lambda = -\left(\frac{\partial g}{\partial \lambda}\right)^{-1} \frac{\partial g}{\partial A}[dA] = \langle \nabla_A \lambda, dA \rangle = \operatorname{tr}((\nabla_A \lambda)^T dA),$$

where $\partial g / \partial \lambda$ is a scalar and the rhs is just the definition of $\nabla_A \lambda$ in terms of the Frobenius inner product. Now, let's see if we can simplify these terms to solve for $\nabla_A \lambda$.

Recall that $d(\det M) = \operatorname{tr}(\operatorname{adj}(M) dM)$ where $\operatorname{adj} M$ is the adjugate matrix: it equals $M^{-1} \det M$ for non-singular M, and more generally is the transpose of the cofactor matrix of M. Let $M(\lambda) = A - \lambda I$. Hence, by the chain rule, we have

$$\frac{\partial g}{\partial \lambda} = \operatorname{tr}(\operatorname{adj}(M) M'(\lambda)) = -\operatorname{tr}(\operatorname{adj}(M))$$

since $M'(\lambda) = -I$ (similar to problem 5.2 of pset 1), and

$$\frac{\partial g}{\partial A}[dA] = \operatorname{tr}(\operatorname{adj}(M) dA).$$

In both cases, we need to compute $\operatorname{adj} M$, and we need to do so at our root $\lambda = \lambda_k$ (one of the eigenvalues of A). The tricky part is that $M(\lambda_k)$ is singular (by definition of an eigenvalue), so we can't use the usual adjugate formula involving M^{-1}.

Instead, let's take the hint and use the diagonalizaiton $A = Q \Lambda Q^T$ of A, where the columns of Q are the orthonormal eigenvectors q_k and the diagonal elements of Λ are the real eigenvalues λ_k (noting that A is real-symmetric so that it has such a diagonalization). Then $M(\lambda) = A - \lambda I = Q(\Lambda - \lambda I) Q^T$ has the same eigenvectors, but eigenvalues shifted by λ. First, let's consider the case where λ is *not* an eigenvalue, so that we can use the explicit adjugate formula:

$$\operatorname{adj} M = M^{-1} \det M = Q(\Lambda - \lambda I)^{-1} Q^T \det(\Lambda - \lambda I) = Q \operatorname{adj}(\Lambda - \lambda I) Q^T,$$

where the adjugate of the diagonal matrix $\Lambda - \lambda I$ is easy: you just invert the diagonal entries and multiply by the determinant (the product of the diagonal entries). When you do this, however, the diagonal entry that you are dividing by *cancels* the corresponding entry from the determinant, and you are left with diagonal entries:

$$(\Lambda - \lambda I)_{ii} = \prod_{j \neq i} (\lambda_j - \lambda).$$

Now we can take the limit as $\lambda \to \lambda_k$ (*one* of the eigenvalues), and we see that all of the diagonal terms vanish (because they contain $\lambda_k - \lambda_k$) *except* for the k-th term, leaving us with a rank-1 matrix:

$$\operatorname{adj} M(\lambda_k) = \left[\prod_{j \neq k} (\lambda_j - \lambda_k)\right] q_k q_k^T .$$

Moreover, $-\operatorname{tr}(\operatorname{adj}(M(\lambda_k))) = -\operatorname{tr}(\operatorname{adj}(\Lambda - \lambda_k I))$ simply becomes $-\prod_{j\neq k}(\lambda_j - \lambda_k)$. So, when we perform the division in $-\left(\frac{\partial g}{\partial \lambda}\right)^{-1} \frac{\partial g}{\partial A}[dA]$, these $\prod_{j\neq k}$ terms (along with the minus signs) *cancel*, leaving us with:

$$d\lambda = \operatorname{tr}(q_k q_k^T dA) ,$$

so the gradient is $\nabla_A \lambda_k = (q_k q_k^T)^T = q_k q_k^T$ as desired.

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.

Solution:

1. Let $L = [L_1, L_2]$. The forward-mode equations in class for the Jacobian $\partial u/\partial L$ can then be directly quoted from the course notes (just replacing p with L):

$$\boxed{\frac{\partial}{\partial t}\left(\frac{\partial u}{\partial L}\right) = \underbrace{\frac{\partial f}{\partial u}}_{4\times 4} \underbrace{\frac{\partial u}{\partial L}}_{4\times 2} + \underbrace{\frac{\partial f}{\partial L}}_{4\times 2}},$$

   where $f(u, L)$ is the right-hand side of our ODE (which depends on L), and the initial conditions are $\boxed{\left.\frac{\partial u}{\partial L}\right|_{t=0} = 0_{4\times 2}}$ (since the initial conditions don't depend on L).

2. See accompanying Julia notebook.
3. See accompanying Julia notebook.

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.

Solution:

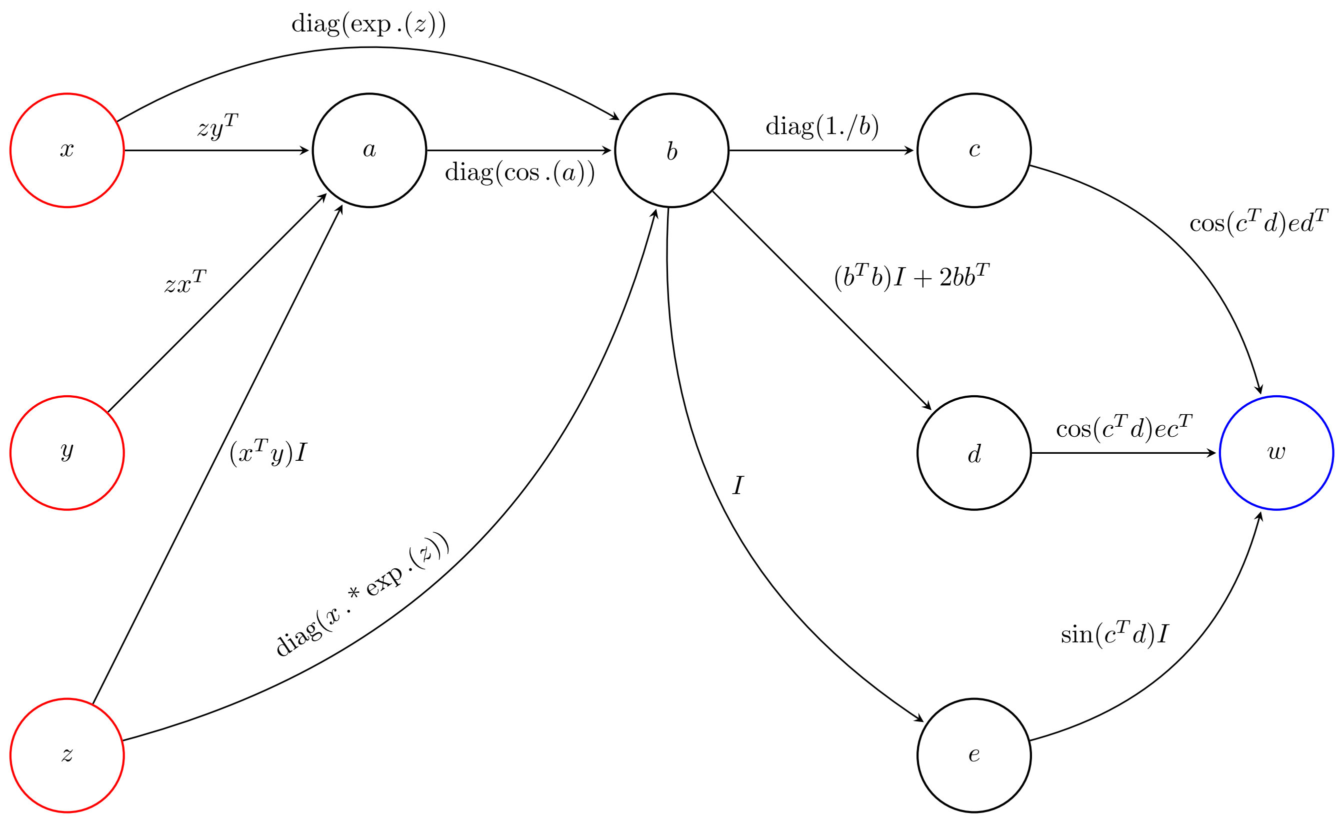
x
y
z
a
b
c
d
e
w
$\operatorname{diag}(\exp.(z))$
zy^T
$\operatorname{diag}(\cos.(a))$
$\operatorname{diag}(1./b)$
zx^T
$(x^Ty)I$
$\operatorname{diag}(x.*\exp.(z))$
$(b^Tb)I+2bb^T$
I
$\cos(c^Td)ed^T$
$\cos(c^Td)ec^T$
$\sin(c^Td)I$

Figure 1: The computational graph for problem 5, with nodes labeled by variables (inputs x, y, z at left, output w at right) and edges labeled by partial derivatives.

The computational graph is shown in Fig. 1 (where diag v denotes the diagonal matrix with the entries of v on the diagonal).

In the following, we refer to the edges above by partial derivatives ($\partial b/\partial a$ for the edge from a to b, etcetera) as in class.

1. Forward mode, which proceeds by going from each input to the output, and finally adding the results:
   (a) Paths from x. First we compute $\frac{db}{dx} = \frac{\partial b}{\partial x} + \frac{\partial b}{\partial a}\frac{\partial a}{\partial x} = \operatorname{diag}(\exp.(z)) + \operatorname{diag}(\cos.(a))(zy^T)$ (summed over two paths). Then we sum over the paths from b to w:

$$\frac{dw}{dx} = \frac{\partial w}{\partial c}\left[\frac{\partial c}{\partial b}\frac{db}{dx}\right] + \frac{\partial w}{\partial d}\left[\frac{\partial d}{\partial b}\frac{db}{dx}\right] + \frac{\partial w}{\partial e}\left[\frac{\partial e}{\partial b}\frac{db}{dx}\right]$$
$$= (\cos(c^Td)ed^T)\left[\operatorname{diag}(1./b)\frac{db}{dx}\right] + (\cos(c^Td)ec^T)\left[((b^Tb)I + 2bb^T)\frac{db}{dx}\right] + (\sin(c^Td)I)\left[I\frac{db}{dx}\right].$$

   Note that all products are performed *right to left* (inputs to outputs).

   (b) Paths from y: Similarly, we go from y to b: $\frac{db}{dy} = \frac{\partial b}{\partial a}\frac{\partial a}{\partial y} = \operatorname{diag}(\cos.(a))(zx^T)$ and then from b to w:

$$\frac{dw}{dy} = \frac{\partial w}{\partial c}\left[\frac{\partial c}{\partial b}\frac{db}{dy}\right] + \frac{\partial w}{\partial d}\left[\frac{\partial d}{\partial b}\frac{db}{dy}\right] + \frac{\partial w}{\partial e}\left[\frac{\partial e}{\partial b}\frac{db}{dy}\right]$$

   which is exactly the same as the expression for dw/dx except that we have db/dy instead of db/dx. If we parenthesized it from left-to-right, we could factor out the common expressions, but forward mode does not do this!

   (c) Paths from z: Similarly, we go from z to b: $\frac{db}{dz} = \frac{\partial b}{\partial a}\frac{\partial a}{\partial z} + \frac{\partial b}{\partial z} = \operatorname{diag}(\cos.(a))((x^Ty)I) + \operatorname{diag}(x.*\exp.(z))$ (two paths) and then again from b to w:

$$\frac{dw}{dy} = \frac{\partial w}{\partial c}\left[\frac{\partial c}{\partial b}\frac{db}{dz}\right] + \frac{\partial w}{\partial d}\left[\frac{\partial d}{\partial b}\frac{db}{dz}\right] + \frac{\partial w}{\partial e}\left[\frac{\partial e}{\partial b}\frac{db}{dz}\right]$$

   where again there are a lot of repeated calculations that could be factored out *if* we changed the order.

2. Reverse mode: we follow a similar procedure, except this time work from outputs to inputs:

(a) From w to b (summed over three paths):

$$\frac{dw}{db} = \frac{\partial w}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial w}{\partial d}\frac{\partial d}{\partial b} + \frac{\partial e}{\partial b}\,.$$

Note that this corresponds to factoring out db/dx, db/dy, and db/dz from the forward-mode expressions above and performing the common multiplications/additions *once*. Reverse mode does this for you!

(b) From b to a:

$$\frac{dw}{da} = \frac{dw}{db}\frac{\partial b}{\partial a}\,.$$

Note that we multiply *on the right* (or equivalently we could transpose everything and multiply on the left).

(c) To x (two paths from a and b):

$$\frac{dw}{dx} = \frac{dw}{da}\frac{\partial a}{\partial x} + \frac{dw}{db}\frac{\partial b}{\partial x} = \left(\frac{dw}{db}\frac{\partial b}{\partial a}\right)\frac{\partial a}{\partial x} + \frac{dw}{db}\frac{\partial b}{\partial x}$$

(d) To y (one path from a):

$$\frac{dw}{dx} = \frac{dw}{da}\frac{\partial a}{\partial y} = \left(\frac{dw}{db}\frac{\partial b}{\partial a}\right)\frac{\partial a}{\partial y}$$

(e) To z (two paths from a and b):

$$\frac{dw}{dz} = \frac{dw}{da}\frac{\partial a}{\partial z} + \frac{dw}{db}\frac{\partial b}{\partial z} = \left(\frac{dw}{db}\frac{\partial b}{\partial a}\right)\frac{\partial a}{\partial z} + \frac{dw}{db}\frac{\partial b}{\partial z}$$

Notice that going from outputs-to-inputs (multiplying left-to-right), as is the style in reverse mode, automatically factored out the common sub-expression dw/da and computes it only once.

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 $\langle X, Y\rangle = \operatorname{tr}(X^T Y)$, 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 $\exp'(A)[dA]$. (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.

Solution:

1. (a) For $f(A) = A^n$, we will use the suggested form of $f'(A)[dA]$ (which is simply the product rule, summing over which of the n A terms becomes dA). Plugging this into the inner product (and using linearity), we find:

$$\begin{aligned}
\langle X, f'(A)[Y]\rangle &= \sum_{k=0}^{n-1} \operatorname{tr}\left(X^T A^k Y A^{n-1-k}\right) && \text{linearity}\\
&= \sum_{k=0}^{n-1} \operatorname{tr}\left(A^{n-1-k} X^T A^k Y\right) && \text{cyclic trace}\\
&= \operatorname{tr}\left(\left[\sum_{k=0}^{n-1} (A^T)^k X (A^T)^{n-1-k}\right]^T Y\right) && \text{transpose + linearity}\\
&= \langle f'(A^T)[X], Y\rangle && \text{Q.E.D.}
\end{aligned}$$

(b) For $f(A) = A^{-1}$, the proof is similar, relying on the fact that $(A^{-1})^T = (A^T)^{-1}$:

$$\begin{aligned}
\langle X, f'(A)[Y]\rangle &= -\operatorname{tr}\left(X^T A^{-1} Y A^{-1}\right) && \text{derivative of } A^{-1}\\
&= -\operatorname{tr}\left(A^{-1} X^T A^{-1} Y\right) && \text{cyclic trace}\\
&= \operatorname{tr}\left([-(A^T)^{-1} X (A^T)^{-1}]^T Y\right) && \text{transpose}\\
&= \langle f'(A^T)[X], Y\rangle && \text{Q.E.D.}
\end{aligned}$$

2. Here, $f(A) = \det(A + \exp(A))$.

(a) From class, we saw that $\det'(X)[dX] = \det(X)\operatorname{tr}(X^{-1} dX)$. Here, $X = A + \exp(A)$, so by the chain rule we can plug in $dX = dA + \exp'(A)[dA]$:

$$df = f'(A)[dA] = \boxed{\det(\underbrace{A + \exp(A)}_{X}) \operatorname{tr}\left(X^{-1}(\underbrace{dA + \exp'(A)[dA]}_{dX})\right)}.$$

(b) We need to write $df = \langle \nabla f, dA\rangle = \operatorname{tr}[(\nabla f)^T dA]$. From above, already have one term in this form, giving us a term $\det(X) X^{-T}$ in the gradient. The other term is our $\exp'(A)$ term, and we can use the theorem in the previous part to shift this to act on X by transposing A. (Making $\exp'$ act "to the left" or "transposing the operator" is, in fact, an instance of reverse-mode or "adjoint" differentiation.) Hence, we have:

$$df = \det(X)\langle X^{-T} + \exp'(A^T)[X^{-T}], dA\rangle$$

which gives

$$\boxed{\nabla f = \underbrace{\det(X)}_{f(A)}\left(X^{-T} + \exp'(A^T)[X^{-T}]\right)}.$$

where $X = A + e^A$ and $X^{-T} = (X^{-1})^T = (X^T)^{-1}$. This only requires us to compute one determinant (which can be re-used from the computation of $f(A)$), one exp, one $\exp'$ application, and one inverse X^{-T}.[2]

3. See accompanying Julia notebook for finite-difference check.

[2]Note that matrix inversion is *much* faster than matrix exponentiation, although both operations have $O(m^3)$ cost for $m \times m$ matrices—about $10\times$ faster on my 2021 laptop for $m = 1000$. Also, both matrix inversion and determinants start with LU factorization, so they can share some computation.