# Homework 2 Solutions

February 3, 2024

**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+3+5+5 points)

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 $2 \times 2$.) 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 $n$ terms of the Taylor series (up to the $f^{(n-1)}$ derivative) 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 $f^{(m)}(x_0)$.

**Solution:**

1. Consider the function $g(x) = f(y + hx)$. By the chain rule, $g^{(m)}(x) = h^m f^{(m)}(y + hx)$, so it follows that $f^{(m)}(y) = \frac{1}{h^m} g^{(m)}(0)$. Since $x_0 = 0$, plug in the finite-difference formula for $g^{(m)}(0) \approx \sum_{k=1}^{n} w_k g(x_k) = \sum_{k=1}^{n} w_k f(y + hx_k)$, and the result follows.

(The key thing to remember is that the finite-difference stencil is for *any* function, not just for functions called "$f$".)

2. For $x = [0, 1]$, the `stencil` function returns $\boxed{w = [-1, 1]}$ (see attached Julia notebook). From the formula in the previous part, this gives $\boxed{f'(y) \approx \frac{-f(y) + f(y+h)}{h}}$, which is exactly the forward-difference approximation from class.

3. For $x = [0, 1, 2, 3]$, the `stencil` function returns $w = [-11/6, 3, -3/2, 1/3]$ (see attached Julia notebook). Trying it out numerically for $\sin'(1)$, we find that the error scales as $\boxed{\sim h^3}$, i.e. it is *third-order accurate.*

4. The familiar Taylor series formula using $\delta x = (x_j - x_0)$ takes the form $f(x_0 + (x_j - x_0)) = f(x_j) = \sum_{i=1}^{\infty} \frac{f^{(i-1)}(x_0)}{(i-1)!}(x_j - x_0)^{i-1}$. Letting $A_{ij} = (x_j - x_0)^{i-1}/(i-1)!$, as suggested, we have that $f(x_j) = \sum_{i=1}^{\infty} A_{ij} f^{(i-1)}(x_0)$. To form the approximation we truncate to $n$ terms, and write the equations in matrix form:
$$\begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} = A^T \begin{pmatrix} f(x_0) \\ \vdots \\ f^{(n-1)}(x_0) \end{pmatrix}.$$
Taking an inner product with $e_m$ and rearranging we see that
$$\begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}^T A^{-1} e_m = f^{(m)}(x_0)$$
as desired, since the left hand side is exactly $\sum_{k=1}^{n} w_k f(x_k)$

## Problem 3 (4+5+8 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 $\nabla f$ (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 $\delta p$ (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 $2 \times 2$ 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 4 ((5+3)+(5+3)+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 $\exp'(A)[dA]$. (You learned how to compute $\exp'$ in pset 1.)

   (b) Using the identity from the previous part, write $\nabla f$ 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 $5 \times 5$ `A=randn(5,5)` and a random small `dA=randn(5,5)*1e-8`, compute $df = f(A + dA) - f(A)$ and $\nabla f$ (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:

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

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

$$\begin{aligned}
\langle X, f'(A)[Y]\rangle &= -\operatorname{trace}\left(X^T A^{-1} Y A^{-1}\right) && \text{derivative of } A^{-1} \\
&= -\operatorname{trace}\left(A^{-1} X^T A^{-1} Y\right) && \text{cyclic trace} \\
&= \operatorname{trace}\left([-(A^T)^{-1} X (A^T)^{-1}]^T Y\right) && \text{transpose} \\
&= \langle f'(A^T)[X], Y\rangle && \text{Q.E.D.}
\end{aligned} \tag{2}$$

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

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

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

(b) We need to write $df = \langle \nabla f, dA\rangle = \operatorname{trace}[(\nabla f)^T dA]$. 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.