MTH 373/573:
Scientific Computing

Solutions to HW 1

IIIT-Delhi, Monsoon 2021
Kaushik Kalyanaraman

Problem 1: Condition Number — 5 + 5 = 10 points

Recast the following problems as evaluation problems $y = f(x)$ for the input x, a fixed parameter a and output y, and compute their condition numbers $\kappa(x)$:

(a) $y - a^x = 0, \quad a > 0$

(b) $x - y + 1 = 0$

For function evaluation, $f : \mathbb{R} \longrightarrow \mathbb{R}$, the condition number is given by:

$$\kappa(x) = \left| \frac{x f'(x)}{f(x)} \right|.$$

(a) $f(x) = y = a^x$. Thus, $\kappa(x) = |x| \, |\log a|$.

(b) $f(x) = y = x + 1 \implies \kappa(x) = \left| \dfrac{x}{x+1} \right|$.

Problem 2: Vector Norms are Lipschitz Continuous — 30 points

Show that any vector norm is Lipschitz continuous, that is, it satisfies the inequality:

$$\big| \|x\| - \|y\| \big| \leq \|x - y\|, \qquad \text{for all } x, y \in \mathbb{R}^n.$$

Remark. The statement is also referred to as the *reverse triangle inequality*.

Proof. For any $x, y \in \mathbb{R}^n$, we have that $\|x\| = \|x - y + y\| \leq \|x - y\| + \|y\|$ by triangle inequality for norms. Thus, $\|x\| - \|y\| \leq \|x - y\|$. Again, by the same argument, $\|y\| = \|y - x + x\| \implies \|y\| - \|x\| \leq \|y - x\| = \|x - y\|$. Thus, $\big| \|x\| - \|y\| \big| \leq \|x - y\|$. □

Problem 3: p-norms and Tensor Product — 15 + 5 = 20 points

(a) Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. Define the *tensor product* of x and y to be the following $\mathbb{R}^{mn}$ vector:

$$x \otimes y := \begin{bmatrix} x y_1 \\ \vdots \\ x y_m \end{bmatrix}_{mn}.$$

What is $\|x \otimes y\|_p$ in terms of $\|x\|_p$ and $\|y\|_p$ for each of the cases $p = 1, 2, \infty$? Demonstrate for each case via step-by-step calculations.

We can show this generally for any p-norm:

$$\|x \otimes y\|_p^p = \sum_{j=1}^{m}\sum_{i=1}^{n} |x_i y_j|^p = \sum_{j=1}^{m}\sum_{i=1}^{n} |x_i|^p \, |y_j|^p = \sum_{j=1}^{m} |x_i|^p \sum_{i=1}^{n} |y_j|^p = \|x\|_p^p \|y\|_p^p.$$

Thus, $\|x \otimes y\|_p = \|x\|_p \|y\|_p$, and in particular, this holds for $p = 1, 2, \infty$.

(b) Based on your solution, conjecture for the generalization to tensor product of two matrices: if A is a $\mathbb{R}^{m\times n}$ matrix and B is a $\mathbb{R}^{k\times \ell}$ matrix, their tensor product is the following $\mathbb{R}^{mk\times n\ell}$ matrix:

$$A \otimes B := \begin{bmatrix} AB_{11} & \dots & AB_{1\ell} \\ \vdots & \ddots & \vdots \\ AB_{k1} & \dots & AB_{k\ell} \end{bmatrix}.$$

For matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{k\times \ell}$, we have $\|A \otimes B\|_p = \|A\|_p \|B\|_p$.

Remark. We now provide a proof for part (b) that is rather straightforward in a "following one's nose" sort of way. The sought claim is also a simple simple example of how one can form conjectures for a related (or "higher") structure based on theorems or properties of a known or simpler structure. (The structures here are the algebra of vectors and matrices on normed spaces.)

We shall show the claim for the subordinate matrix p-norm and start from its definition as applied to a tensor product of matrices as in the problem statement:

$$\|A \otimes B\|_p = \max_{x\in\mathbb{R}^{n\ell}} \frac{\|(A \otimes B)\, x\|_p}{\|x\|_p}.$$

Since $x \in \mathbb{R}^{n\ell}$ can be written as a tensor product of two vectors $u \in \mathbb{R}^n$ and $w \in \mathbb{R}^\ell$, we can next write:

$$\|A \otimes B\|_p = \max_{u\in\mathbb{R}^n, w\in\mathbb{R}^\ell} \frac{\|(A \otimes B)\,(u \otimes w)\|_p}{\|u \otimes w\|_p} = \max_{u\in\mathbb{R}^n, w\in\mathbb{R}^\ell} \frac{\|(A \otimes B)\,(u \otimes w)\|_p}{\|u\|_p \|w\|_p}.$$

Now, we have not talked about, and it is beyond the scope of discussion in this course, how a tensor product of matrices *acts on*, that is multiplies with, a vector (generally) or a tensor product of vectors as above. This "tensored" matrix acting on a "tensored" vector is something that arises in quantum mechanics (but in a physicists' *bra-ket* formalism) and also in the study of deeper *differential* geometry. We digress however and returning back to the calculation on hand, the following is not surprising but requires more definitional work that $(A \otimes B)\,(u \otimes v) = (Au) \otimes (Bw)$. *This* statement is beyond the scope of this class and why a proof was not required. However, with it available we can complete the argument:

$$\|A \otimes B\|_p = \max_{u\in\mathbb{R}^n, w\in\mathbb{R}^\ell} \frac{\|(Au)\otimes(Bw)\|_p}{\|u\|_p\|w\|_p} = \max_{u\in\mathbb{R}^n, w\in\mathbb{R}^\ell} \frac{\|Au\|_p\|Bw\|_p}{\|u\|_p\|w\|_p} = \left(\max_{u\in\mathbb{R}^n} \frac{\|Au\|_p}{\|u\|_p}\right)\left(\max_{w\in\mathbb{R}^\ell} \frac{\|Bw\|_p}{\|w\|_p}\right),$$

where the last equality is due to the maximum of a product of positive functions being the product of the maximums of the functions. Thus $\|A \otimes B\|_p = \|A\|_p\|B\|_p$. □

We have thus demonstrated that the matrix subordinate p-norm of the tensor product of two matrices is the product of the subordinate p-norms of the matrices. However, there are other matrix norms for which this is true as well. The following nearly fifty year old paper provides a characterization of all such normed vector spaces.

Lancaster, P. and H. K. Farahat (1972). "Norms on direct sums and tensor products". In: *Math. Comp.* 26, pages 401–414. ISSN: 0025-5718. DOI: 10.2307/2005167.

Problem 4: Nilpotent Matrix is Singular 10 points

Let $A \in \mathbb{R}^{n\times n}$ be a nilpotent matrix with index 2, that is $A^2 = \mathbb{O}$ where $\mathbb{O} \in \mathbb{R}^{n\times n}$ is the zero matrix. Using the definition of singularity in terms of linearly independence of columns (rows) of a matrix, show that A is singular.

A matrix $A \in \mathbb{R}^{n\times n}$ is singular if its columns (rows) are linearly dependent. That is, there exists a $0 \neq x \in \mathbb{R}^n$ such that $Ax = 0$ (respectively, $x^T A = 0$). For a nonzero x, we thus have that $A^2 = \mathbb{O} \iff A^2x = \mathbb{O}x = 0$. Denote $y = Ax$; so $Ay = 0$. Either $y = 0$ which means for $x \neq 0$, $Ax = 0$ or $y \neq 0$ but $Ay = 0$. Thus A is singular.

Problem 5: Condition Number of Matrices 5 points

Show that for any nonsingular matrices $A, B \in \mathbb{R}^{n\times n}$, their condition number (in any subordinate matrix norm) satisfies: $\kappa(AB) \leq \kappa(A)\,\kappa(B)$.

Using the definition of the matrix condition number and submultiplicative property of subordinate matrix norm, we have:

$$\kappa(AB) = \|AB\|\|(AB)^{-1}\| = \|AB\|\|B^{-1}A^{-1}\| \leq \|A\|\|B\|\|B^{-1}\|\|A^{-1}\| = \kappa(A)\,\kappa(B).$$

Problem 6: Exploring IEEE Double Precision using Python 15 points

(a) What does this code snippet do? Type it in an IPython terminal or Jupyter Notebook. Explain what you observe when you run it and why in no more than 2-3 sentences.

```
a = 1.
while a != 0:
    a /= 2
    print(a)
```

A partial output from running the code snippet is below:

```
0.5
0.25
0.125
0.0625
0.03125
0.015625
0.0078125
.
.
.
1.6e-322
8e-323
4e-323
2e-323
1e-323
5e-324
0.0
```

The code snippet thus keeps halving numbers starting from 1 and the last number printed before exiting at 0 is approximately the *underflow* or *UFL* of the IEEE double precision floating point number system.

> **Remark.** The UFL here is with gradual underflow to 0 with denormal numbers.

(b) What does this code snippet do? Type it in an IPython terminal or Jupyter Notebook. Explain what you observe when you run it and why in no more than 2-3 sentences.

```
a = 1.; eps = 1.; b = a + eps
while a != b:
    eps /= 2
    b = a + eps
print(eps)
```

The output from running the code snippet is below:

```
1.1102230246251565e-16
```

The printed number is the value of eps which when added to 1 does not change it in floating point arithmetic and is related to the machine epsilon ε_M. In the IEEE double precision floating point number system, the rounding rule used is rounding to even. That is, rounding to nearest with ties broken by choosing that number whose last bit in the floating point representation is even. In this system, the value of ε_M is $2.220446049250313 \times 10^{-16}$, and the printed `eps` is half of this value which leaves 1 unchanged on addition.

(c) What does this code snippet do? Type it in an IPython terminal or Jupyter Notebook. Explain what you observe when you run it and why in no more than 2-3 sentences.

```
    a = 1.
    while a != inf:
        a *= 2
        print(a)
```

Warning: The decimal point "." after the 1 in the variable a is extremely important.

A partial output from running the code snippet is below:

```
    2.0
    4.0
    8.0
    16.0
    32.0
    64.0
    128.0
    .
    .
    .
    2.8088955232223686e+306
    5.617791046444737e+306
    1.1235582092889474e+307
    2.247116418577895e+307
    4.49423283715579e+307
    8.98846567431158e+307
    inf
```

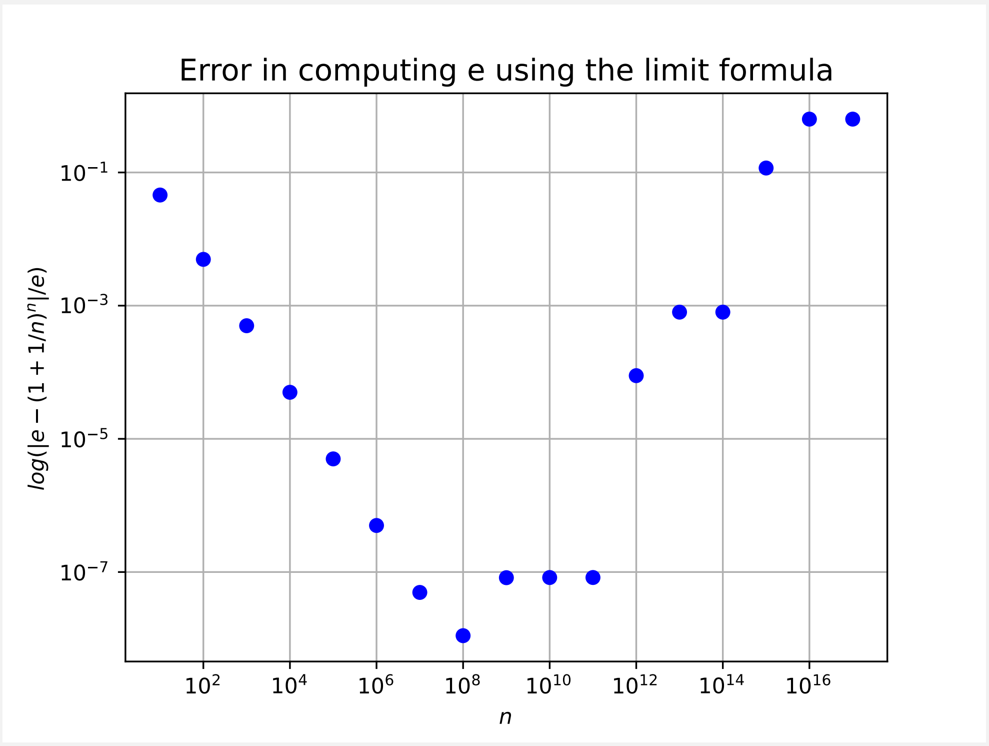
The code snippet keeps doubling numbers starting from 1 and the last number printed before exiting at the special number `inf` is approximately the *overflow* or *OFL* of the IEEE double precision floating point number system.

Problem 7: Understanding Round off Error 20 + 15 = 35 points

(a) Recall that the number e is defined by $\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$. Estimate the value of e by writing a Python code to compute the expression $\left(1+\frac{1}{n}\right)^n$ for $n = 10^k$, $k = 1, 2, \ldots, 15$. Using `plt.plot` from `matplotlib` (see Python tip in Guidelines on first page), plot the relative error in this computation for each value of k. For what value of k do you get the most accurate estimate for e using this limit formula? Why does this not match with the mathematical expectation that this formula should become increasingly more accurate as n increases?

Reason in terms of floating point arithmetic and its relation with machine epsilon ε_M.

The Python code for this problem is in `problem_7a.py` and the plot of relative error in the computation of e using the limit definition is shown below.



Mathematically, e should be increasingly better approximated for large n, yet this computation using IEEE double precision floating point fails beyond around 10^{16}. Since $\varepsilon_M = 2.220446049250313 \times 10^{-16}$, $\log_{10} \dfrac{1}{\varepsilon_M} \approx 15.65$. Hence, for $n \geq 10^{16}$, $\mathrm{fl}\left(1 + \dfrac{1}{n}\right) = \mathrm{fl}(1)$ leading to the erroneous computation of e as 1! (Here $\mathrm{fl}(x)$ is the IEEE double precision floating point representation of $x \in \mathbb{R}$.)

What happens to this computation at around 10^8 and until around 10^{16} is interesting however. And, hopefully you could learn a few things here! First, we obtain the relative condition number of the evaluation problem:

$$f(x) = \left(1 + \frac{1}{x}\right)^x \quad \text{as} \quad \kappa(x) = \left| x \log\left(\frac{1}{x} + 1\right) - \frac{x}{x+1} \right|.$$

Check that $\kappa(1) \approx 0.1931$, $\kappa(10) \approx 0.0440$, $\kappa(10^8) \approx 1.07 \times 10^{-9}$ and $\lim_{x\to\infty} \kappa(x) = 0$. Thus, as expected, the mathematical problem is increasingly very well conditioned for large x, and in the limit, any *backward* error will be amplified by a very small factor. However, our Python computations happen in the discrete floating point regime, not in the continuum $\mathbb{R}$.

Therefore, all our computational error – since the algorithm is identical to the mathematical problem – is due to representation in the floating point number systems and the resulting round off error. Recall that according to the definition of ε_M, for any $x \in \mathbb{R}$,

$\mathrm{fl}(x) \in \mathbb{F}$ (the floating point number system) such that:

$$\left|\frac{\mathrm{fl}(x)-x}{x}\right| = \mathcal{O}(\varepsilon_M) \iff \mathrm{fl}(x) = x(1+\varepsilon) \text{ for some } |\varepsilon| \le \varepsilon_M.$$

Here, ε is the round off error in the floating point representation of x. Note that the round off error can be both positive or negative. Thus, we should expect that $\left(1+\frac{1}{n}\right)$ to be prone to being erroneous due to round off error. Let us suppose that:

$$\mathrm{fl}\left(1+\frac{1}{n}\right) = 1+\frac{1}{n}+\varepsilon, \quad |\varepsilon| \le \frac{\varepsilon_M}{2}.$$

We are not using $\left(1+\frac{1}{n}+\varepsilon+\frac{\varepsilon}{n}\right)$ as would be expected if applying the floating point round off error "as is". This is because $\left(1+\frac{1}{n}\right) \in [1,2]$, and in fact very close to 1 for large n. Thus, the gap between two floating point numbers in this interval is ε_M, and the largest round off error possible is $\frac{\varepsilon_M}{2}$ (due to rounding to nearest in IEEE double precision).

Thus our limit computation for e is actually going to be the floating point computation $\left(1+\frac{1}{n}+\varepsilon\right)^n$. We need to use this to perform an error analysis in this computation as compared to the originally expected value of e for the computation. We can do so as follows:

$$\left(1+\frac{1}{n}+\varepsilon\right)^n = e^{\log\left(1+\frac{1}{n}+\varepsilon\right)^n} = e^{n\log\left(1+\frac{1}{n}+\varepsilon\right)},$$

where we use that for any $a \in \mathbb{R}$, $a = e^{\log a}$. (In other words, that e^x and $\log x$ are inverse functions of each other.) We need to do this seemingly "pulling a rabbit out of the hat" step because of our having to compare this computation against the true value of e.

Next, let us use the following Taylor series for $\log(1+x)$:

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots, \quad x \in (-1,1],$$

in our previous analysis to obtain:

$$\left(1+\frac{1}{n}+\varepsilon\right)^n = e^{n\left(\left(\frac{1}{n}+\varepsilon\right)-\frac{1}{2}\left(\frac{1}{n}+\varepsilon\right)^2+\cdots\right)} = e^{\left(1+n\varepsilon-\frac{1}{2n}-\varepsilon-\frac{n\varepsilon^2}{2}+\cdots\right)} \approx e^{\left(1+n\varepsilon-\frac{1}{2n}-\varepsilon\right)},$$

ignoring the $\mathcal{O}(\varepsilon^2)$ terms. Finally, using the Taylor series for e^x around $x=0$ and again ignoring the $\mathcal{O}(\varepsilon^2)$ terms, we get:

$$\begin{aligned}\left(1+\frac{1}{n}+\varepsilon\right)^n &\approx e^{\left(1+(n-1)\varepsilon-\frac{1}{2n}\right)} = e\cdot e^{\left((n-1)\varepsilon-\frac{1}{2n}\right)} \\ &= e\cdot\left(1+\left((n-1)\varepsilon-\frac{1}{2n}\right)+\frac{1}{2}\left((n-1)\varepsilon-\frac{1}{2n}\right)^2+\cdots\right) \\ &= e\cdot\left(1+(n-1)\varepsilon-\frac{1}{2n}+\frac{(n-1)^2\varepsilon^2}{2}-\frac{(n-1)\varepsilon}{2n}+\frac{1}{4n^2}+\cdots\right)\end{aligned}$$

$$\approx e\left(1+(n-1)\left(1-\frac{1}{2n}\right)\varepsilon+\frac{1}{4n^2}\right).$$

Thus, we can finally write down the relative computational error as:

$$\text{Error} = \left|\frac{e-\left(1+\frac{1}{n}+\varepsilon\right)^n}{e}\right| \approx \left|(n-1)\left(1-\frac{1}{2n}\right)\varepsilon+\frac{1}{4n^2}\right|.$$

For $\varepsilon \leq \frac{\varepsilon_M}{2} \lesssim 10^{-16}$ and $n = 10^8$, plugging into the expression above, we get a relative error of around 10^{-8} which nearly exactly agrees with our computation. Notice that this expression also explains the errors we computationally obtain for $n = 10^k$ for $k = 9, 10, \ldots, 15$ but beyond that it is not valid because, as explained earlier, $\mathrm{fl}\left(1+\frac{1}{n}+\varepsilon\right) = 1$.

(b) Now implement the Taylor series computation for e:

$$e = 1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots.$$

First, using your understanding of ε_M, what stopping criterion should be used for determining where to truncate this Taylor series? Once you have determined this, implement this using an appropriate looping construct in Python. What is the relative error in the computation of e in this case?

We know that floating point addition of two unequal magnitude numbers can potentially saturate if a "small enough" number is added to the relatively larger number. For example, at fl(1), $\mathrm{fl}(1+\varepsilon) = \mathrm{fl}(1)$ if $|\varepsilon| \leq \frac{\varepsilon_M}{2}$ where ε_M is the machine epsilon.

We can rewrite the infinite series sum for e as:

```
sum[1] = 1
for k = 1, 2, ...
    sum[n + 1] = sum[n] + 1/k!
```

The sum will saturate if `1/k!` is "small enough" in comparison with `sum[n]`. Since $e \in [2, 4]$ and the gap between floating point numbers in this interval is $2\varepsilon_M$, thus if $\frac{1}{k!} \leq \frac{2\varepsilon_M}{2} = \varepsilon_M$, the sum will saturate. Thus, the stopping criterion for this computation should be to check if `sum[n + 1] - sum[n]` $\geq \varepsilon_M$.

The Python code for this problem is in `problem_7b.py` and its output is as below.

```
The computation converged after summing 18 terms.
The relative error in computing e via its truncated Taylor
series is 1.6337129034990842e-16.
```

Note that $\frac{1}{18!} \approx 1.5619\times10^{-16}$ as expected. The error in estimating e in this computation is less than ε_M and this is a much better algorithm unlike in part (a).