Homework #4
CS 575: Numerical Linear Algebra
Spring 2023

John Tran

March 2, 2023

Important Notes

- `Python` 3.11 was used to run the notebook (i.e., to use the `match` statement)
- the partially completed notebook was used to complete the coding assignment, so many of the things asked (e.g., verification and testing) was done for us and they were modified for the mat-mat portion
- the `README` was done in `Markdown` but the raw text can still be viewed

Note: Problem 5 was canceled for HW 4

Problem 1

Note: assume A is an $m \times m$ matrix and any vectors mentioned (e.g., x, y) are $m \times 1$ column vectors

(a) If we allow k to be some large positive integer, then it would be more efficient to compute the LU decomposition first in $O(m^3)$ time and then solve the system $Ay = b$ for y where $y = A^{k-1}x$ and keep repeating the process until $y = Ax$ (iteratively solve the system with a different solution vector y with the power decreased by 1 per iteration but with the same matrix A each time, utilizing the LU decomposition since it stays the same). Since each solve would only require forward-sub and a backward-sub taking $O(m^2)$ time, it would reduce the overall time since we only need an expensive LU decomposition once.

(b) If we look at the format of $\alpha = c^T A^{-1} b$, we can see that $A^{-1}b$ is just a column vector and c^T times a column vector is just a dot product (relatively cheap operation) so we just need to find an efficient way to deal with the inverse of A, or $A^{-1}b$. Looking at it more closely, it resembles the linear system we have been used to solving $Ax = b$ where we can multiply by the inverse of A on both sides to get $x = A^{-1}b$. So, we just need to solve for x to get our desired quantity and an efficient method we have discussed so far is to use Gaussian Elimination with Partial Pivoting taking $O(m^3)$ time. Then for the dot product $c^T x$, that would take $O(n^2)$ time so it would be bottle-necked by how efficiently we can solve for x.

(c) If we want to solve the matrix equation $AX = B$ where B is an $m \times n$ matrix, we can make use of the fact that we are just solving a series of linear equations that is represented in a matrix form for X and B. If we decompose the columns of X and B as $[x_1 x_2 \dots x_n]$ and $[b_1 b_2 \dots b_n]$, respectively, where x_i and b_i are column vectors for $i \in \{1, 2, \dots, n\}$, we have the following

$$\begin{aligned} Ax_i &= b_i \quad \text{for } x \in 1, 2, \dots, n \\ [Ax_1 Ax_2 \dots Ax_n] &= [b_1 b_2 \dots b_n] \end{aligned}$$

where the last statement is the matrix form given to us.

So, we just need to solve n systems of linear equations with the same matrix A. As with part (a), we can use an LU decomposition and do a forward-sub and a backward-sub for each of the x_i and b_i vectors and combine the column vectors together at the very end.

Problem 2

We are asked to the inequality showing how the relative error in the solution vector x relates to the residual. To start off, we can use the ratio of the relative error and residual

$$
\begin{aligned}
\frac{\frac{\|e\|}{\|x\|}}{\frac{\|r\|}{\|b\|}} &= \frac{\|e\|\|b\|}{\|r\|\|x\|} \\
&= \frac{\|e\|\|b\|}{\|r\|\|x\|} \\
&= \frac{\|A^{-1}r\|\|Ax\|}{\|r\|\|x\|} \quad \text{since } A \text{ is invertible and does not change} \\
&\leq \frac{\|A^{-1}\|\|r\|\|A\|\|x\|}{\|r\|\|x\|} \quad \text{sub multiplicative inequality for induced matrix norm} \\
&= \left\|A^{-1}\right\|\|A\| \\
&= \text{cond}(A) \quad \text{def. of condition number} \\
&\iff \\
\frac{\|e\|}{\|x\|} &\leq \text{cond}(A)\frac{\|r\|}{\|b\|}
\end{aligned}
$$

Problem 3

Problem 4

(a) We are asked to tabulate the condition numbers for the Hilbert matrices for $8 \leq n \leq 12$ (the `scipy.linalg.hilbert` and `numpy.linalg.cond` were used to find the Hilbert matrices and conditions, respectively, in `Python`)

Hilbert matrix size (N)	condition number
8	15257575566.627958
9	493153404551.01215
10	16025028168113.176
11	522020733204514.75
12	1.6211639047474996e+16

Figure 1: The condition numbers of Hilbert matrices for various n values

Hilbert matrix size (N)	condition number
8	1.525758e+10
9	4.931534e+11
10	1.602503e+13
11	5.220207e+14
12	1.621164e+16

Figure 2: (Modified to show scientific notation) The condition numbers of Hilbert matrices for various n values

(b) We are asked to find the true solution x_t to the linear system $Ax = b$ with the specifications given in the problem. If we look at the definition of the linear system for a given b_i,

$$b_i = \sum_{j=1}^{n} a_{ij} x_j \quad \text{def. of mat-vec multiplication}$$
$$= \sum_{j=1}^{n} a_{ij} \quad \text{given condition}$$

We can see a clear solution for x that satisfies our constraint on b (for all components b_i) if we let all components $x_i = 1$ where the dot product of the ith row of A (a_i) and x reduces to just the sum of the components in a_i.

(c) Since we are not allowed to solve the system, we can use the equation provided for d and estimate how many correct decimal digits we should expect given the condition numbers we just found where

$$d = |\log_{10}(\epsilon)| - \log_{10}(\kappa(A))$$

Since we know that machine epsilon is about 1.1e^{-16} (or 2.2e^{-16} depending on the definition), then $|\log_{10}(\epsilon)|$ is about 16 and $\log_{10}(\kappa(A))$ is about 10-16 from the condition numbers above. That means we can expect about 6 correct decimal digits for $n = 8$ all the way down to 0 correct decimal digits for $n = 12$...so only a few correct decimal digits for our range of n values.

(d) Now we are asked to write down our solution vectors for $8 \leq n \leq 12$

Hilbert matrix size (N)	solution vector x
8	[0.9999999999681907 1.0000000016822066 0.9999999782000777
	1.0000001174172146 0.9999996848830713 1.0000004448897650
	0.9999996839051423 1.0000000890763439]
9	[0.9999999998088036 1.0000000130887288 0.9999997796130058
	1.0000015674037004 0.9999942673865873 1.0000116780493731
	0.9999866137742437 1.0000080725488052 0.9999980081985083]
10	[0.9999999988121090 1.0000001004503016 0.9999978957924024
	1.0000188750065389 0.9999109586890151 1.0002425099467109
	0.9996052623095678 1.0003788560225120 0.9998022952831284
	1.0000432483796637]
11	[0.9999999958426009 1.0000004222705359 0.9999893051088595
	1.0001172564692462 0.9993127363607148 1.0023832424312256
	0.9948719161340677 1.0069195945055029 0.9943037786753380
	1.0026145788086986 0.9994871714243700]
12	[0.9999999528179779 1.0000059096165135 0.9998158495003417
	1.0024906029623015 0.9818522445253225 1.0793319375135959
	0.7799196038043006 1.3968785552237171 0.5362388104715811
	1.3386630883813226 0.8595581443147454 1.0252453298241522]

Figure 3: Tabulated the solution vectors for various n values

(e) If we look at the digits of accuracy in the solution components for different values of n, we can see look at the digits $0.99\ldots9x$ and $1.00\ldots0x$ for the digit x does not equal to 9 or 0, respectively. So for $n = 8$, there are components with more than 6 digits of accuracy (like the first few components) but as we check more, we do see that there are components with just 6 digits of accuracy. We can see this is about the case for other n values as well, and if we check $n = 12$ where we can expect no digits of accuracy...that does seem to be the case (e.g., components that doesn't start with a 0 or 9 in the first decimal place). Overall, the trend does seem to be consistent with what we found in part (c).

(f) **Note:** the norms of vectors were calculated using the 2-norm (default in `numpy` for vectors)

We are asked to tabulate the relative error in the solution vectors x and the residual

Hilbert matrix size (N)	relative error in solution	residual
8	2.289490e-07	6.558244241556096e-17
9	6.831866e-06	1.324188329850844e-16
10	2.018434e-04	1.6160436568167624e-16
11	3.301377e-03	2.119561965201541e-16
12	2.165459e-01	1.3793022792304736e-16

Figure 4: Tabulated the relative errors

As n increases, we can see that the residual does remain consistently and relatively small at around 10^{-16}. However, if we look at the relative error in the solution, it actually increases by a little more than an order of magnitude per 1 increase in n. If we use the equation for the correct digits d given to us, we can see that not only does the number of digits decrease with n, we have more incorrect components as well since n increases. So, there is nothing to balance off the gain in errors since the solution vector x stays as a vector of components of all 1s (norm of x only increases marginally as n increases). This is not the case for the residual because even though errors do accumulate, this is balanced by the norm of b also increasing with n to match the growth of errors given by $b_i = \sum_{j=1}^{n} a_{ij}$, increasing with the size of the matrix and the row components of A.

Problem 6

(a) An anonymous function was created in `Python` to calculate $f(x)$ for $x = 1.2 \times 10^{-8}$

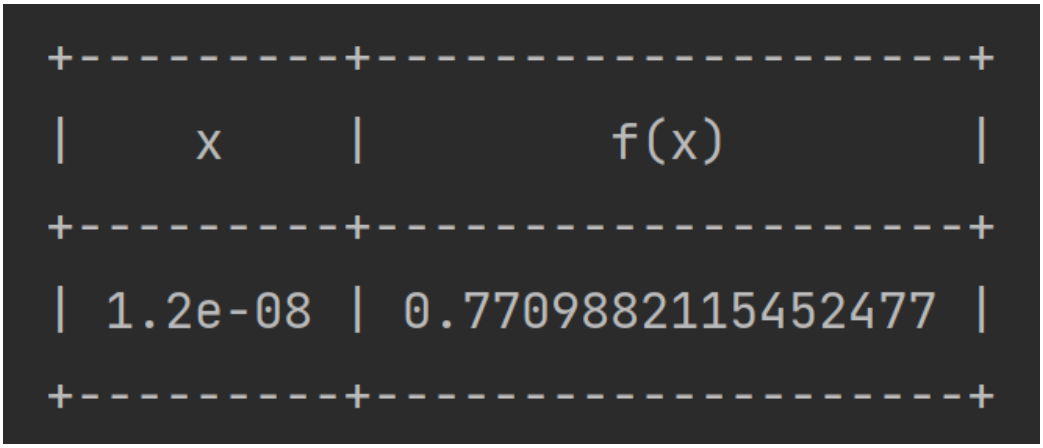

Figure 5: Result in `Python`

(b) We are asked to show that

$$\lim_{x \to 0} f(x) = \frac{1}{2}$$

So, we substitute the half angle formula $\cos x = 1 - 2\sin\left(x/2\right)^2$ for $f(x)$ and find the limit as $x \to 0$

$$
\begin{aligned}
f(x) &= \frac{1-\cos x}{x^2} \\
&= \frac{1-(1-2\sin^2(x/2))}{x^2} \\
&= \frac{1-1+2\sin^2(x/2)}{x^2} \\
&= \frac{2\sin^2(x/2)}{x^2} \\
\lim_{x\to 0} f(x) &= \lim_{x\to 0} \frac{2\sin^2(x/2)}{x^2} \\
&= \frac{0}{0} \quad \text{indeterminate} \\
&= \lim_{x\to 0} \frac{4\sin(x/2)(\cos(x/2)\cdot 1/2)}{2x} \quad \text{L'Hopital's Rule} \\
&= \lim_{x\to 0} \frac{2\sin(x/2)\cos(x/2)}{2x} \\
&= \lim_{x\to 0} \frac{\sin(x/2)\cos(x/2)}{x} \\
&= \frac{0}{0} \quad \text{indeterminate} \\
&= \lim_{x\to 0} \frac{1/2\sin^2(x/2)+1/2\cos^2(x/2)}{1} \quad \text{L'Hopital's Rule} \\
&= \lim_{x\to 0} \frac{1/2(\sin^2(x/2)+\cos^2(x/2))}{1} \\
&= \lim_{x\to 0} \frac{1/2}{1} \quad \sin^2(x/2)+\cos^2(x/2)=1 \text{ for any angle} \\
&= 1/2
\end{aligned}
$$

(c) Then we use the half angle formula to recalculate the same expression in `Python`

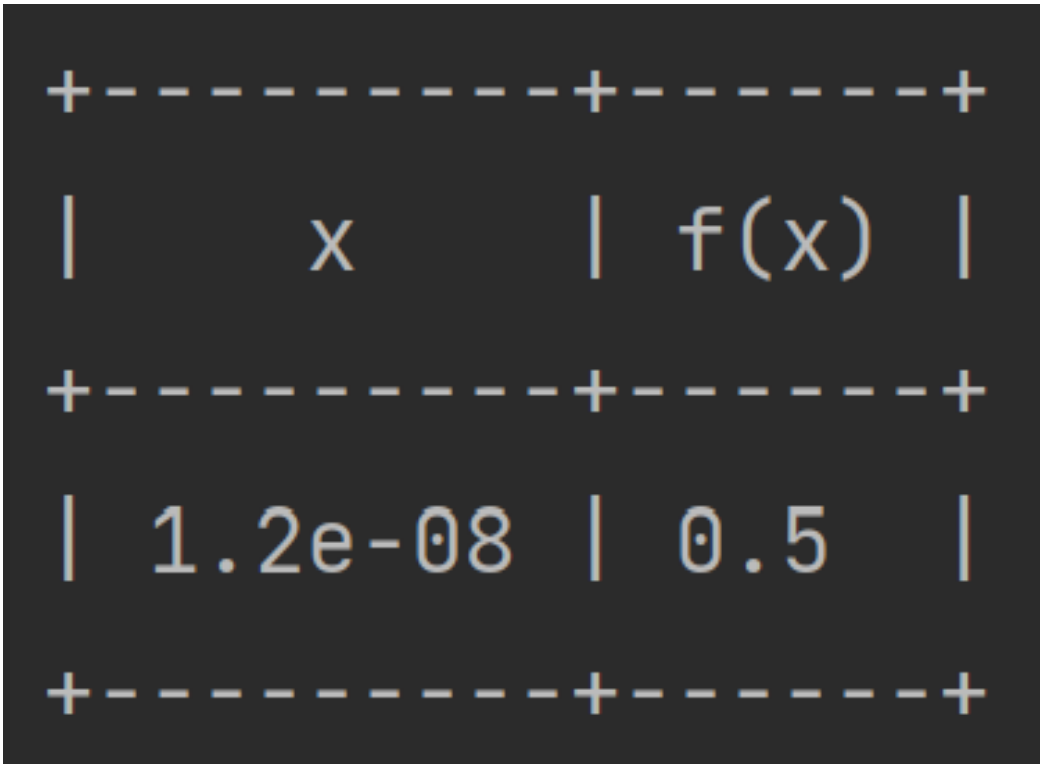

Figure 6: Result in `Python` with the half angle formula

This result is different from part (a) because of how we are subtracting a small quantity from 1 in the numerator: $1 - \cos x$. At a small value of x, $\cos x$ is really close to 1 and precision is lost to compute this intermediate step in the calculation. This is because of the subtraction itself–not sure if we covered this in class but the condition number for this subtraction is inversely proportion to this difference (from another class we have taken in the past) and since this difference is small, it explodes the condition number for this particular calculation. However, when we substitute the half angle formula in, we observe that the 1 in the numerator cancels out and all we have left in the numerator is $2\sin^2 x/2$, which can be represented more precisely since we just have a small quantity to begin with.