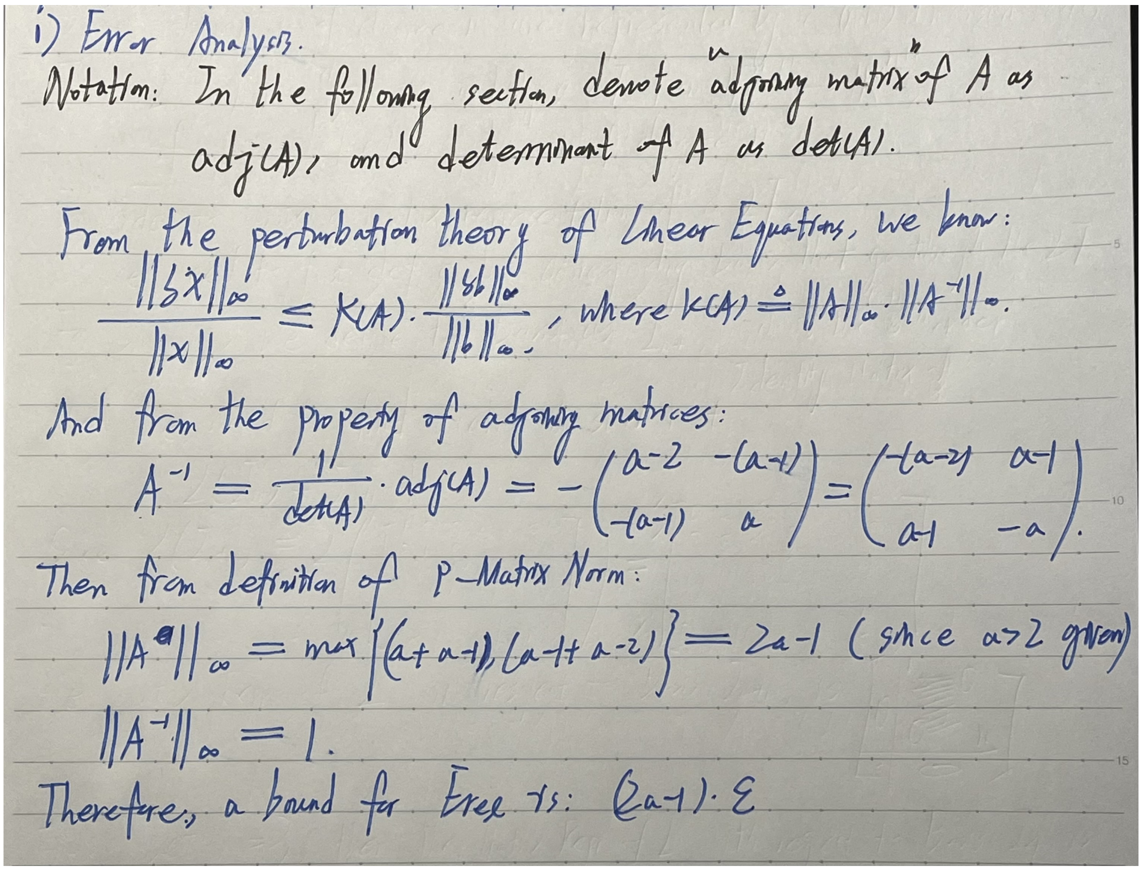
ECS 230 - Homework 02

Qirun Dai
`qrdai@ucdavis.edu`

November 1st, 2023

1 Problem 1: Error Analysis



i) Error Analysis.

Notation: In the following section, denote "adjoining matrix" of A as adj(A), and determinant of A as det(A).

From the perturbation theory of Linear Equations, we know:

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \le K(A) \cdot \frac{\|\delta b\|_\infty}{\|b\|_\infty}, \text{ where } K(A) \triangleq \|A\|_\infty \cdot \|A^{-1}\|_\infty .$$

And from the property of adjoining matrices:

$$A^{-1} = \frac{1}{\det(A)} \cdot \operatorname{adj}(A) = -\begin{pmatrix} a-2 & -(a-1) \\ -(a-1) & a \end{pmatrix} = \begin{pmatrix} -(a-2) & a-1 \\ a-1 & -a \end{pmatrix}.$$

Then from definition of p-Matrix Norm:

$$\|A\|_\infty = \max\{(a+a-1), (a-1+a-2)\} = 2a-1 \text{ (since } a>2 \text{ given)}$$

$$\|A^{-1}\|_\infty = 1.$$

Therefore, a bound for Error is: $(2a-1) \cdot \varepsilon$

Figure 1: Handwritten Solution to Problem 1

2 Problem 2: Using timing functions

2.1 Timing1.c (Table 1)

The timing procedure. There are two noteworthy points involved in the timing procedure of *timing*1.*c*. First, it calculates both CPU time and total execution time (i.e. wall-clock time) of the program. Specifically, using the system call *clock*(), *timing*1.*c* first calculates the amount of CPU time that is only consumed by this program itself. On the contrary, *timing*1.*c* also uses the system call *gettimeofday*() to measure the the wall-clock time, which includes not only the time the program is actually running on the CPU but also time taken by other processes or consumed while waiting for I/O. Second, since the execution time of computing a square root only once is so small that it may fall within the error margin of the timing function due to extra time cost brought by memory access or system calls, *timing*1.*c* amortizes the total time of a large number of square root operations so as to get a relatively accurate estimation of per computation cost. Such an amortization approach effectively reduces the relative error induced by extra time cost that is irrelevant to the time estimation of square root computation.

The reproducibility between runs. As can be found in Table 1, the observed CPU time does not always center around certain value, and there may be outliers for the timing results. Actually, poor reproducibility between different runs is not unusual, especially in a multitasking operating system where the program competes for CPU time with other processes. The system's load at each run can vary greatly, affecting how much CPU time *timing*1*.c* receives. Moreover, although we use *clock*() to only include the CPU time, it is noteworthy that *clock*() actually involves a system call, which can add significant overhead to the measurement and lead to outliers in timing results.

The estimation based on observed results. Based on the analysis above, we only use the observed CPU time for estimation, and exclude the outliers from calculating the average. Since the performance of amortization is inclined to increase with the number of total runs, we use data obtained under the CLI argument 2×10^7. Therefore, the estimated time (in seconds) for one square root computation based on *timing*1*.c* is:

$$\frac{0.042363}{2 \times 10^7} = 2.11815 \times 10^{-9} \tag{1}$$

run\argv	5×10^6	1×10^7	2×10^7
1	0.009750	0.022732	0.042367
2	0.009745	*0.055102*	0.042552
3	0.009738	0.023049	*0.083201*
4	0.009859	0.022649	0.042508
5	0.009741	0.023058	0.042293
6	0.009761	0.023418	0.042268
7	0.009743	0.022985	0.042239
8	*0.013546*	*0.055676*	0.042464
9	0.009734	0.022843	0.042530
10	0.009735	0.020387	0.042047
Average	**0.009756**	**0.022640**	**0.042363**

Table 1: Observed data for *timing*1*.c*. Note that the time listed here is CPU time instead of wall-clock time. All the units are converted to **second**, and all the experiments are conducted on the CSIF machine `pc25`. *Italicized* entries are outliers and are excluded from averaging calculation.

2.2 Timing2.c (Table 2)

The timing procedure. *timing*2*.c* also measures only the CPU time associated with the process's execution by using the *rdtsc* instruction which reads the time stamp counter (TSC) of Intel x86 processors. The TSC is a 64-bit register that increments with every clock cycle and, as such, is a very high-resolution measure of processor time. On modern CPUs, *rdtsc* can increment at a rate of several GHz (billions of cycles per second), which allows for much finer granularity in timing measurements compared to *clock*(). In addition, *timing*2*.c* employs the same amortization strategy as *timing*1*.c*.

The reproducibility between runs. As can be seen from Table 2, there are far fewer outliers in the results estimated by *rdtsc* than *clock*(), indicating good reproducibility using *timing*2*.c*. This can be explained in the following aspects: first, as is mentioned above, the exceptionally high resolution of *rdtsc* allows for much finer granularity in timing measurements than *clock*(), while the latter typically has a resolution limited by the constant $CLOCKS\_PER\_SEC$, which is fixed to 1 million, thus only providing a granularity of 1 microsecond at best. Second, the *rdtsc* instruction is executed directly by the CPU hardware and has very low overhead, while the *clock*() invovles a system call to the OS software, thus introducing more uncertain factors which not only increase overhead but also degrade the reproducibility of time measurement.

The estimation based on observed results. Since the *rdtsc* instruction only returns the total number of CPU cycles, we need to find the clock rate of the CPU by looking at the file `/proc/cpuinfo`, which is $2.9GHz$. Finally, with the total number of CPU cycles averaged over 10 runs (excluding 1 outlier) under the CLI argument 2×10^7, which is equal to 122660099, we can compute the number of clock cycles needed for one square root operation as follow:

$$\frac{122660099}{2 \times 10^7} = 6.133$$

Divide this value by the previously retrieved clock rate, we can get the final estimation of square root computing time (in seconds):

$$\frac{6.133}{2.9 \times 10^9} = 2.11483 \times 10^{-9} \tag{2}$$

Compared with the result obtained in 1, we can find that the time estimation results using two different timing procedures are quite close in value, demonstrating that both procedures are generally effective for time estimation of the square root operation.

run\argv	5×10^6	1×10^7	2×10^7
1	39033727	65760629	122369133
2	38155652	66824378	123455826
3	40727753	66064724	123132463
4	38538145	66478316	122538921
5	38885230	65776479	122170824
6	39170898	66555277	*230291094*
7	38112777	65837822	122189710
8	38803179	66683840	122608197
9	39356625	65896513	122918798
10	38245192	65904656	122557019
Average	**38902918**	**66178263**	**122660099**

Table 2: Observed data for *timing2.c*. All the units are converted to **the number of clock cycles**, and all the experiments are conducted on the CSIF machine `pc25`. *Italicized* entries are outliers and are excluded from averaging calculation.