CSCI / MATH 2072U - Assignment 3

Syed Arham Naqvi
100590852

February 27, 2023

QUESTION 1

(b) Given the matrix A:

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 3 & a & 1 \\ 1 & -2 & 3 \end{bmatrix}$$

The secant iteration method, with initial guesses $a^{(0)} = 0$ and $a^{(1)} = 5$, can be used to calculate the "a" value required so that matrix A has a condition number of 4 within residual and error tolerances of 10^{-8}.

After performing all necessary computations we can see that matrix A has a condition number of 4 when $a \approx 3.85100409$.

QUESTION 2

(b) We begin with a linear system $Vx = c$ where V is a 26x26 Vandermonde matrix with $n = 25$. The resultant vector c is taken to be the second last column of V, meaning that the true solution vector x must be the nth column of a 26x26 identity matrix. Because we know the true solution, we can directly calculate and compare the maximal relative error and the true relative error on both sides of the following inequality:

$$\begin{aligned} (\text{relative error of } x_*) &\leq (\text{maximal relative error of } x_*) \\ (\text{relative error of } x_*) &\leq (\text{condition number of } V) * (\text{relative residual of } x_*) \\ \frac{||e||}{||x||} &\leq K * \left(\frac{||r||}{||c||}\right) \\ \frac{||x - x_*||}{||x||} &\leq K * \left(\frac{||c - Vx_*||}{||c||}\right) \end{aligned}$$

The error vector (e), is defined as the difference between the true solution (x) and the solution approximation (x_*). The residual vector (r), is defined as the difference between the true resultant vector (c) and the approximated resultant vector (Vx_*). The relative error of x_* can now be calculated as the 2-norm of e divided by the 2-norm of x, while the maximal relative error of x_* is found by multiplying K, which is the condition number of V, with the 2-norm of r divided by the 2-norm of c.
After performing all calculations above, we should see that the relative error of x_* is less than or equal to the maximal relative error of x_*.

Each of the above variables was calculated using python functions in the following order:

$$V = numpy.vander(numpy.linspace(-1, 1, 26)) \tag{1}$$
$$c = V[:, 24] \tag{2}$$
$$x = numpy.identity(26)[:, 24] \tag{3}$$
$$x_* \text{ is among the returned values of the function } LUPsolve(V, c) \tag{4}$$
$$K = numpy.linalg.cond(V, 2) \tag{5}$$
$$e = x - x_* \tag{6}$$
$$r = c - Vx_* \tag{7}$$
$$\frac{||e||}{||x||} = scipy.linalg.norm(e, 2) \ / \ scipy.linalg.norm(x, 2) \tag{8}$$
$$K * \left(\frac{||r||}{||c||}\right) = K \ * \ (scipy.linalg.norm(r, 2) \ / \ scipy.linalg.norm(c, 2)) \tag{9}$$

After computing all values and substituting into the inequality above we find that:

$$\begin{aligned}(\text{relative error of } x_*) &= 2.731258289630759 * 10^{-07} \\ &\leq 1.168238332485655 * 10^{-05} \\ &= (\text{maximal relative error of } x_*)\end{aligned}$$

(d) After including the relative residuals and condition numbers on the same plot, it can be observed that with increasing size of the Vandermonde matrices due to the increasing n values, the relative residuals are roughly consistent around error values of approximateley 10^{-16}. After $n = 41$, condition number $= 1.4648836183042544 \times 10^{17}$ and relative residual $= 1.8673122326933411 \times 10^{-16}$ however, the relative residual values begin to increase rapidly in error. Beyond $n = 41$, the perturbations in the approximated solutions are large enough to where the yielded resultant vectors are that of radiacally different systems altogether, thus our solutions begin to lose meaning. Using the max-norm in our computations has no noticeable effect on this phenomenon.

QUESTION 3

(a) What follows is a pseudocode algorithm describing the multiplication of a vector x with a unit lower-triangular matrix L such that unnecessary multiplications with 0's and 1's are excluded.

Input: Matrix $L \in \mathbb{R}^{n \times n}$ *s.t.* $(L_{i,j} = 0 \iff j > i \ \& \ L_{j,j} = 1)$, Vector $x \in \mathbb{R}^n$
for $(i = 1 : n)$ **do**
  $y_i \leftarrow y_i + x_i$
  for $(j = i + 1 : n)$ **do**
    $y_j \leftarrow y_j + L_{ij} \times x_i$
  end
end
Output: Vector $y \in \mathbb{R}^n$ *s.t.* $(y = Lx)$

(d) The algorithm for part (a) requries the computation of 1 flop on line 2 and 2 flops on line 4. This translates to the following computational complexity calculations using flop summations:

$$\begin{aligned}
\text{Computational Complexity} &= \sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}2\right)\\
&= \sum_{i=1}^{n}\left(1+2\sum_{k=1}^{n-i}1\right)\\
&= \sum_{i=1}^{n}(1+2(n-i))\\
&= \sum_{i=1}^{n}(1+2n-2i)\\
&= n+2n(n)-2\sum_{i=1}^{n}i\\
&= n+2n^2-2\left(\frac{n(n+1)}{2}\right)\\
&= n+2n^2-n^2-n\\
&= n^2\\
&= \mathcal{O}(n^2)
\end{aligned}$$

(g) The behaviour of this plot falls well within expectations. Here, the $GenMatVec()$ algorithm has a complexitiy of $2n^2-n$ flops while the $LowTriangMatVec()$ algorithm from part (a) was found to have a complexitiy of n^2 flops. Both algorithms have a Big-Oh complexity of $\mathcal{O}(n^2)$ meaning that for very large n values, the number of flops are roughly $f \approx \alpha n^2$ (for some positive alpha). This means both algorithms can then be represented using a logarithmic relationship as follows:

$$ln(f) \approx ln(\alpha) + 2(ln(n))$$

As we can see from the relationship above, the slope for both graphs should be 2 which is clearly evident from the fact that both lines are almost perfectly parrallel in the log-log plot.

(h) The outputted computation times for the built-in python matrix multiplication algorithm are far less then those of the $GenMatVec()$ algorithm or even the $LowTriangMatVec()$ algorithm. The reason for the superior time complexitiy of the built-in algorithms is due to the fact that they are written in highly optimized BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra PACKage) machine code implementations that are designed to effeciently handle things like matrices and vectors.