MACM 316 – Computing Assignment #4

Due Date: Wednesday November 8 at 11:00pm.

Instructions: Upload to Crowdmark a single PDF file that consists of only two pages: page #1 is your report (containing all discussions, data and figures), and page #2 is a listing of your Matlab code. The assignment is due at **11:00pm sharp!** If Crowdmark indicates that your submission was late, then you will be assigned a grade of zero, with no exceptions – so don't leave submitting until the last minute. When this CA is released, you should receive an e-mail containing a link for uploading your assignment, so make sure you save this message.

- Carefully review the **"Guidelines for Computing Assignments"** posted on Canvas.
- If you have questions about this assignment then you can obtain help in the computational workshops, tutorials, or the "Computing Assignment" discussion group on Canvas.

CA4 – Approximating matrix exponentials

Recall from calculus that the exponential of a real number x has the Taylor series expansion

$$e^x = \exp(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots$$

If you have any experience with complex variables like $z = x + iy$, then you may have encountered the complex exponential $e^z = e^x e^{iy}$ which can also be represented with this Taylor series. But it gets even better! If A is an $n \times n$ matrix, then the exponential of A can be defined analogously as

$$e^A = \exp(A) = I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \cdots \qquad (*)$$

where I is the $n \times n$ identity matrix and A^k is represents the matrix product $\underbrace{A \cdot A \cdots A}_{k \text{ times}}$. Matrix exponentials are useful in a number of problems in computational mathematics, especially for the solution of systems of differential equations.

WARNING: The exponential of a matrix is not equal to the exponential of its entries!! For example:

$$\exp\left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right) = \begin{bmatrix} 1.5430806348152 & 1.1752011936438 \\ 1.1752011936438 & 1.5430806348152 \end{bmatrix} \neq \begin{bmatrix} e^0 & e^1 \\ e^1 & e^0 \end{bmatrix} = \begin{bmatrix} 1 & e \\ e & 1 \end{bmatrix}.$$

In this CA, you will implement a simple algorithm for approximating the matrix exponential by summing the first k terms in the infinite series $(*)$. The aim is to investigate the accuracy, efficiency and robustness of this algorithm. You will test your code on a special matrix A that is stored in one of four files named `CA4matrix#.mat`[†], where #=1, 2, 3 or 4. Select one of the matrix files from Canvas and load it into Matlab using

```
load('CA4matrix#.mat');
```

which assigns your chosen matrix to the variable `A`. Look at the entries of this matrix and note that it is a full, 500×500 matrix with complex entries.

[†]These files are in "mat-file" format, which provides a simple and efficient method for storing and accessing variables saved in a previous Matlab session.

(a) Write a Matlab code that implements the truncated series approximation for e^A. Test your code on your chosen matrix A with $k = 50$ and $k = 150$ terms in the series, and assign the result to the variable `expAk`. You can visualize the size of the entries in your approximation of the matrix exponential using the following two commands:

```
imagesc(real(expAk))
colormap gray
```

This generates a gray-scale plot with each pixel depicting the size of a matrix entry in e^A by its colour: large entries are white, small entries are black. Describe the differences between your two images in terms of "image quality".

HINT: When writing your code, you should make use of the following formula for the k^{th} term in the series:

$$\frac{1}{k!}A^k = \frac{1}{k}A \cdot \left(\frac{1}{(k-1)!}A^{k-1}\right)$$

This simple trick can be used to increase the efficiency of your code by reducing the number of matrix multiplications required in your truncated series approximation!!

(b) Run your algorithm again using a wider range of k values with $k = 5, 10, 15, \ldots, 150$. Determine the execution time required in each case using Matlab's `tic` and `toc` commands, which are invoked as follows:

```
tic;
   {{ my code }}
cpu_time = toc;
```

The variable `cpu_time` is the elapsed CPU time required to execute the code between the `tic/toc` commands.Plot your CPU times versus k and describe how they depend on k. Estimate the number of floating point operations or "flops" required by your algorithm as a function of k, and then explain whether or not your flop estimate is consistent with your plot.

(c) Matlab has a built-in command called `expm` for computing the matrix exponential $\exp(A)$:

```
expAexact = expm(A);
```

First, determine the CPU time required for this `expm` calculation and compare it with the timings for your algorithm from part (b).

If you then treat this as your "exact result," you can estimate the error in your truncated series approximation using the matrix 2-norm:

```
err = norm(expAexact-expAk, 2);
```

Plot the error against k for the same range of k values as in part (b), this time using a logarithmic scale for the error. How does the error depend on k?

(d) Complete your report with a summary discussion of how your algorithm compares to Matlab's `expm` in terms of the following three criteria:

- *accuracy:* measured by the values of `err`,
- *efficiency:* comparing CPU time for both methods,
- *robustness:* in terms of which method is most "reliable".[‡]

[‡]Go back to the lectures notes for Section 1a, where I define robustness, which addresses questions such as "Is the algorithm guaranteed to return a 'good' result?" and "How strongly do accuracy and efficiency depend on the inputs?"