CS/SE 4X03 — Assignment 3

3 March 2020

Due date: 17 March, in class.

- **A hardcopy must be submitted before 1:20pm.**
- **Between 1:20pm and 1:20pm on the 20th, an assignment will be accepted only with an MSAF.**
- **If you write your solutions by hand, please ensure your handwriting is legible and easy to read. Otherwise, please type your solutions. We will deduct marks for messy hand-written solutions.**
- **Please submit to Avenue only the files that are required. Also, please submit in the hard copy only what is required.**

Problem 1 (7 points)

Standard matrix multiplication of $N \times N$ matrices has complexity $O(N^3)$. Strassen's algorithm[1] for matrix multiplication has complexity $O(N^{2.8074})$.

Download the files strassen.m and strassenw.m from https://es.mathworks.com/matlabcentral/fileexchange/2360-the-matrix-computation-toolbox?focused=5041535&tab=function.

For this problem, you need to determine empirically the constants a and b in $a \cdot N^b$, in the big-O notation, for Matlab's matrix multiplication, strassen.m and strassenw.m.

(5 points) Implement the function

```
function [a, b, n, time] = bigOconstants(alg)
%Estimates the constants in big-O notation for matrix
%multiplication, that is a and b in a*N^b.
%If alg == 1, a and b are computed for Matlab's *
%If alg == 2, a and b are computed for strassen
%If alg == 3, a and b are computed for strassenw
%n is a vector of matrix sizes and time is the corresponding
%computing time.
%E.g. if n = [100 200 300], time(1) is the time
%for multiplying 100x100 matrices.
```

This function times a matrix multiplication for various sizes and finds a and b. Then run the script main_bigO.m. When plotted, your data points should be nearly on straight lines.

(2 points) The a constants from strassen.m and strassenw.m are much larger than the a from Matlab's $*$. Explain why this is the case.

Submit

- Hardcopy: the produced two plots, bigOconstants.m, your explanation
- Avenue: bigOconstants.m

Problem 2 (12 points)

(7 points) Implement in Matlab the composite midpoint rule, the composite trapezoidal rule, and the composite Simpson's rule. Each rule should be implemented as a function with four input parameters (1) the function to be integrated, (2-3) the end points of the integration interval, and (4) the number of subintervals. The return value should be the approximated integral. Your implementation must not use any loops.

Using the above functions, implement

[1]https://en.wikipedia.org/wiki/Strassen_algorithm

```
function [eT, eM, eS] = error_int(f, a, b, n, ref)
%Evaluates the integral of f over [a,b] using
%n(i) subintervals.
%n is a row integer vector. ref is an accurate value for this
%integral
%eT(i),  eM(i), eS(i) are absolute errors in
%the composite trapezoid, midpoint, and Simpson,
%respectively, when evaluated over n(i) subintervals.
```

Execute the script `main_int.m`. If your implementation is correct, you should see two parallel nearly straight lines and another nearly straight line with some "wiggles" at the bottom.
(5 points) Explain

- why these two lines are parallel
- the reason for those "wiggles"
- the slopes of the straight lines

Submit

- Hardcopy: the produced plot, the Matlab code without `main_int.m`, your explanations
- Avenue: all Matlab code related to this problem.

Problem 3 (4 points)

(3 points) Consider the integral $\int_0^1 \sin(\pi x^2/3)dx$. Suppose that we wish to integrate it numerically with an absolute error of at most 10^{-10} using equally spaced points and the trapezoidal rule. What is the minimum number n of subintervals that are needed to achieve this accuracy?
(1 point) Using this n and your implementation of the trapezoidal rule, verify that you indeed obtain an accuracy within 10^{-10}

Problem 4 (5 points)

The period of a simple pendulum is determined by the complete elliptic integral of the first kind

$$K(x) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1-x^2\sin^2\theta}}.$$

Implement the adaptive Simpson's algorithm and use it to to evaluate this integral for enough values of x to draw a smooth plot of $K(x)$ over the range $-1 \le x \le 1$.

Submit

- Hardcopy: plot of $K(x)$
- Avenue: all your Matlab code related to this problem. Name your main program `main_pend.m`

Problem 5 (5 points)

One can evaluate the double integral

$$\int_a^b \int_c^d f(x,y)dxdy$$

using a subroutine for evaluating a single integral. Denoting

$$F(x) = \int_c^d f(x,y)dy$$

we have

$$\int_a^b \int_c^d f(x,y)dxdy = \int_a^b \left(\int_c^d f(x,y)dy \right) dx = \int_a^b F(x)dx.$$

Use your implementation of adaptive Simpson to implement

```
function Q = myqyad2d(fun, a, b, c, d, tol)
%Approximate the integral of fun(x,y) over [a,b]x[c,d]
%by calling adaptive Simpson with tolerance tol
```

To check the accuracy of your results run e.g.

```
f = @(x,y) exp(x-y).*sin(x+y);
a = 0; b = 1;
c = 0; d = 1;
tol = 1e-8;
Q = myquad2d(f,a,b,c,d,tol);
err = abs(Q-quad2d(f,a,b,c,d,'AbsTol', 1e-12))
```

Submit

- Hardcopy: `myquad2d.m`, the error from the above script
- Avenue: `myquad2d.m`

Problem 6 (5 points)

Construct an example of a function $f(x)$, interval $[a,b]$ and tolerance `tol` such that, to achieve about the same accuracy, the composite Simpson's rule on a uniform mesh requires at least 100 times more function evaluations than your adaptive Simpson.

For this purpose consider the following steps. Assume `tol = 1e-8`, but you can also use other values.

(a) Use Matlab's `quad` function to compute an accurate approximation. For example, you can use tolerance `1e-10` in `quad`.

To measure the error produced by your implementations, you can subtract from the `quad`'s result and take absolute values.

(b) Select an n in your composite Simpson such that the error is about `1e-8`.

Count the number of function evaluations; denote them by C_1.

(c) In the adaptive Simpson, use the same tolerance of `1e-8` and compare with the result from `quad`, to ensure that your computed result is within the tolerance.

Count the the number of function evaluations; denote them by C_2.

Submit

- Hardcopy: plot of $f(x)$, the values for C_1 and C_2; $C_1 \geq 100C_2$ should hold, the measured errors
- Avenue: the Matlab code producing the above. Name the main program `main_simpson.m`

Problem 7 (4 points)

Read about the 200-day moving average, e.g. `https://www.dailyfx.com/forex/education/trading_tips/daily_trading_lesson/2019/07/29/200-day-moving-average.html`.

(a) Obtain the closing prices of Microsoft through

```
stock = get_yahoo_stockdata3('msft','1-Jan-2018');
```

Plot on the same plot these prices and the 200-day moving average. The first one is computed on the 200th trading day (counting from 1-Jan-2018).

(b) Instead of the moving average, now use a polynomial of degree one. That is, given a trading day $d \geq 200$ fit $y = ax + b$ over the last 200 days, including d. Then evaluate y at d. Perform this over all $d \geq 200$ and plot on the same plot the closing prices and your computed values.

(c) Repeat (b) with a quadratic.

(d) Repeat (b) with a cubic.

Submit

- Hardcopy: 4 plots
- Avenue: nothing