University of Nottingham
UK | CHINA | MALAYSIA

MATH2019 Introduction to Scientific Computation

— Coursework 2 (10%) —

Submission deadline: ***2pm, Monday, 8th March 2021*** + ***grace period***

Note: This is currently **version 3** of the PDF document.

This coursework contributes **10%** towards the overall grade for the module.

Rules:

- Each student is to submit their own coursework.
- You are allowed to work together and discuss in small groups (2 to 3 people), but *you must write your own coursework and program all code by yourself*.
- Please be informed of the UoN Academic Misconduct Policy (incl. plagiarism, false authorship, and collusion).

Piazza and Live Computing Classes:

- You are allowed to ask questions on Piazza to obtain clarification of the coursework questions, as well as general Matlab queries. This is certainly encouraged!
- However, **when using Piazza, please ensure that you are not revealing any answers to others**. Also, **please ensure that you are not directly revealing any code that you wrote**. Doing so is considered Academic Misconduct.
- **When in doubt, please simply arrange a 1-on-1 meeting with the demonstrators or lecturers during the Live Computing Classes.**

Coursework Aim & Assessment:

- In this coursework you will develop Matlab code related to algorithms to perform polynomial interpolation & numerical differentiation and you will study some of the algorithms' behaviour.
- A total of 40 points can be obtained in this coursework.

Some Advice:

- You are expected to have basic familiarity with Matlab (in particular: vectors and matrices, plotting, logic and loops, function handles, writing functions and m-files).
- Helpful resources: Matlab Guide (by Tom Wicks), Matlab's Online Documentation
- Write your code as neatly and concisely as possible so that it is easy to follow.
- Add some comments to your scripts that indicate what a piece of code does.

Getting Started:

- Download the contents of the "Coursework 3 Pack" folder from Moodle.

Submission Procedure:

- Submission will open after 23rd February 2021.
- To submit, simply upload the required m-files on Moodle. (Your submission will be checked for plagiarism using *turnitin*.)
- Your m-files will be marked (mostly) automatically: This is done by running your functions and comparing their output against the true results.
- You can check the outputs that your m-files generate by running the file `GenerateYourOutputsCW3.m`, which can be found in folder "Coursework 3 Pack" from Moodle.

▶ Lagrange Polynomial Interpolation

Recall, given $p+1$ distinct points $\{x_i\}_{i=0}^{p}$, the *Lagrange interpolating polynomials* are:

$$L_i(x) = \frac{\prod_{\substack{j=0 \\ j\neq i}}^{p}(x - x_j)}{\prod_{\substack{j=0 \\ j\neq i}}^{p}(x_i - x_j)}$$

A function $f : \mathbb{R} \to \mathbb{R}$ can then be *interpolated* by the function $p_p(x)$ given by

$$p_p(x) = \sum_{i=0}^{p} f(x_i)L_i(x). \tag{1}$$

$\boxed{0}$ Introductory question, not for credit

- Let $f(x) = \cos(\pi x) + x$ and let $x_0 = -\frac{1}{2}$, $x_1 = 0$ and $x_2 = \frac{1}{2}$.
- Find the (quadratic) Lagrange polynomials $L_0(x)$, $L_1(x)$ and $L_2(x)$ based on the points x_0, x_1 and x_2. (You can do this either by hand, or by using Matlab). Use Matlab to plot these 3 Lagrange polynomials on one figure and make sure they look correct.
- Use (1) to construct the polynomial interpolant $p_2(x)$ of $f(x)$. (Again, you can do this either by hand, or using Matlab). Once you have this interpolant, use Matlab to plot it and compare against $f(x)$ over the interval $[-3, 3]$.
- Go to the Geogebra demonstration https://www.geogebra.org/m/bwmpekfa and compare your answers with the one obtained there when $a = -\frac{1}{2}$ and $b = \frac{1}{2}$. Investigate using higher polynomials degrees and try different functions $f(x)$ over different intervals.

$\boxed{1}$

- Write a *function* m-file called `lagrangePoly.m`, which, given a set of $p+1$ distinct nodal points $\{x_i\}_{i=0}^{p}$ and a set of n evaluation points $\{\hat{x}_j\}_{j=1}^{n}$, will return a $(p+1) \times n$ matrix, called `L`, where the ijth entry of the matrix is $L_{i-1}(\hat{x}_j)$.
- Your function should also perform a check to make sure that the nodal points $\{x_i\}_{i=0}^{p}$ are distinct. If the points are distinct, then the output variable `errorFlag` should be set to 0, otherwise `errorFlag` should be set to 1. No other checks on the inputs are required. *Hint:* you should not use == to check for equality of floating point numbers.
- Also add a brief description at the top of your `lagrangePoly` m-file. This description should become visible whenever one types: `help lagrangePoly`.
- The function *must* have the following prototype:

```
function [L,errorFlag] = lagrangePoly(p,x,n,xhat)
```

• Test your function (for example) by typing the following into the command window.

```
p = 3;
x = linspace(-0.5,0.5,4);
n = 6;
xhat = linspace(-1,1,6);
[L,errorFlag] = lagrangePoly(p,x,n,xhat)
```

In this case you should obtain the matrix

$$\mathrm{L} = \begin{array}{cccccc} 6.5625 & 1.6445 & 0.0385 & 0.0165 & -0.1495 & -2.1875 \\ -11.8125 & -1.1385 & 1.0395 & -0.0945 & 0.6435 & 8.4375 \\ 8.4375 & 0.6435 & -0.0945 & 1.0395 & -1.1385 & -11.8125 \\ -2.1875 & -0.1495 & 0.0165 & 0.0385 & 1.6445 & 6.5625 \end{array}$$

and `errorFlag = 0`.

[8 / 40]

Marks can be obtained for your `lagrangePoly` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked:

- The size and values of `L`
- The correctness of `errorFlag` for various inputs `x`.
- The output of "help lagrangePoly".

(Run the file `GenerateYourOutputsCW3.m` to see these outputs generated for a particular set of inputs. Note that in marking your work, different input(s) may be used.)

2 • Write a *function* m-file called `polyInterpolation.m` that will evaluate the pth order polynomial interpolant of a function f at a set of points $\{\hat{x}_j\}_{j=1}^n$. The nodal interpolation points should be *uniformly spaced* over the interval $[\mathrm{a}, \mathrm{b}]$.

• Also add a brief description at the top of your `polyInterpolation` m-file. This description should become visible whenever one types: `help polyInterpolation`.

• The function *must* call `lagrangePoly` from Q1 and should have the following prototype:

```
function interp = polyInterpolation(a,b,p,xhat,f)
```

Here, `interp` is a vector containing the values $\{p_p(\hat{x}_j)\}_{j=1}^n$.

- Test your function (for example) by typing the following into the command window.

```
a = -0.5;
b = 0.5;
p = 3;
xhat = linspace(-1,1,6);
interp = polyInterpolation(a,b,p,xhat,@(x) cos(pi*x)+x)
```

In this case, your output should be

$$\texttt{interp} = \quad -3.9228 \quad -1.0287 \quad 0.6184 \quad 1.0184 \quad 0.1713 \quad -1.9228$$

[4 / 40]

Marks can be obtained for your `polyInterpolation` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked:

- The size and correctness of `interp` for various inputs `x`.
- The output of "help polyInterpolation".

(Run the file `GenerateYourOutputsCW3.m` to see these outputs generated for a particular set of inputs. Note that in marking your work, different input(s) may be used.)

▶ Lagrange Interpolation Errors

Recall, if polynomial interpolation of order p is used to approximate a function $f : [a,b] \to \mathbb{R}$, with nodal points $[x_0, \dots, x_p]$, then for sufficiently smooth f , we have

$$\max_{x\in[a,b]} |p_p(x) - f(x)| \leq \max_{\xi\in[a,b]} \left| \frac{f^{(p+1)}(\xi)}{(p+1)!} \right| \max_{x\in[a,b]} \left| \Pi_{j=0}^{p} (x - x_j) \right| .$$

3

- Write a *function* m-file called `lagrangeErrors` that when run will compute (an approximation to) the error $\max_{x\in[a,b]} |p_{p_j}(x) - f(x)|$ for a range of polynomial degrees $\mathbf{P} = \{p_j\}_{j=1}^{l}$. *Hint:* In order to find an approximation to the error, evaluate the error $|p_p(x) - f(x)|$ for 2000 equally spaced points over $[a,b]$ and take the maximum of those.

- The function should return a vector E of length l, filled with the errors so that $E_j = \max_{x\in[a,b]} |p_{p_j}(x) - f(x)|$.

- The function should make use of `polyInterpolation.m` from Q2 to compute the interpolant. *i.e.* equally spaced nodal points should be used to construct the interpolant.

- The function should also plot the errors $\{E_j\}_{j=1}^{l}$ against $\{p_j\}_{j=1}^{l}$, using `semilogy`. This plot should have all appropriate labels and legends.

- Add a description at the top of your `lagrangeErrors.m` that when 'help lagrangeErrors' is typed will comment on the results when $\mathbf{P} = \{1, 2, 3, \dots, 10\}$ and

(a) $f(x) = \mathrm{e}^{2x}$, $[a, b] = [0, 1]$;

(b) $f(x) = \frac{1}{1+25x^2}$, $[a, b] = [-5, 5]$.

- The function should have the following prototype

```
function E = lagrangeErrors(a,b,P,f)
```

- Test your function (for example) by typing the following into the command window.

```
P = 1:5;
a = -1; b = 1;
E = lagrangeErrors(a,b,P,@(x)1./(x+2))
```

In this case you should obtain

$$\mathrm{E} = \quad 0.1786 \quad 0.0463 \quad 0.0137 \quad 0.0043 \quad 0.0014.$$

[6 / 40]

Marks can be obtained for your `lagrangeErrors` m-file for generating the required output, for certain set(s) of inputs, as well as the plots and comments. The correctness of the following will be checked:
- The size and values of `E`;
- The error plots produced;
- The output of "help lagrangeErrors" and in particular the quality of explanation for the outputs seen.

(Run the file `GenerateYourOutputsCW3.m` to see these outputs generated for a particular set of inputs. Note that in marking your work, different input(s) may be used.)

► Piecewise Polynomial Interpolation

Recall, given a function $f : [a, b] \to \mathbb{R}$, we can construct its piecewise polynomial interpolant of order p by splitting $[a, b]$ up into uniform subintervals of width h and applying the Lagrange interpolant of order p on each subinterval. Hence, using n subintervals $\{[x_{i-1}, x_i]\}_{i=1}^n$, the piecewise interpolant $S_p^n(x)$ satisfies:

$$S_p^n(x)|_{[x_{i-1},x_i]} = p_p^i(x), \quad i = 1, \dots, n$$

where $p_p^i(x)$ is the polynomial interpolant of $f(x)$ on $[x_{i-1}, x_i]$.

$\boxed{4}$ • Write a *function* m-file called `piecewiseInterpolation` that given an interval $[a, b]$ will compute the piecewise polynomial approximation of order p of a function $f(x)$ with n uniformly spaced subintervals of width h. Your function should make use of `polyInterpolation` from Q2 on each subinterval.

- Also add a brief description at the top of your `piecewiseInterpolation` m-file. This description should become visible whenever one types: `help piecewiseInterpolation`.

- The function must have the following prototype:

```
function p_interp = piecewiseInterpolation(a,b,p,n,xhat,f)
```

Here `xhat` is a vector of points (assumed in the interval $[a, b]$) at which the piecewise interpolant should be evaluated. On return, `p_interp` should hold these interpolant values.

- Test your function (for example) by typing the following into the command window.

```
p = 2;
a = -1; b = 1;
n = 5;
xhat = linspace(-0.9,0.9,7);
p_interp = piecewiseInterpolation(a,b,p,n,xhat,@(x) 1./(x+2))
```

In this case you should obtain

$$\texttt{p\_interp} = \quad 0.9107 \quad 0.7143 \quad 0.5878 \quad 0.5000 \quad 0.4349 \quad 0.3846 \quad 0.3448$$

[4 / 40]

Marks can be obtained for your `piecewiseInterpolation` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked:
- The size and values of `p_interp`.
- The output of "help piecewiseInterpolation".

(Run the file `GenerateYourOutputsCW3.m` to see these outputs generated for a particular set of inputs. Note that in marking your work, different input(s) may be used.)

5

- Write a *function* m-file called `piecewiseErrors` that when run will compute (an approximation to) the error $\max_{x\in[a,b]} |S^{n_i}_{p_j}(x) - f(x)|$ for a range of polynomial degrees $\mathbf{P} = \{p_i\}_{j=1}^{l}$ and number of subintervals $\mathbf{N} = \{n_i\}_{i=1}^{m}$. The function should return a matrix E of size $m \times l$, filled with the errors so that $E_{ij} = \max_{x\in[a,b]} |S^{n_i}_{p_j}(x) - f(x)|$. The function should make use of the `piecewiseInterpolation` from Q4.

- The function should also plot for each polynomial degree p_j the errors $\{E_{ij}\}_{i=1}^{m}$ against $\{h_i = \frac{b-a}{n_i}\}_{i=1}^{m}$. You should therefore have l error plots all on a single set of axes with a log-log scale. This plot should have all appropriate labels and legends.

- Add a description at the top of your `piecewiseErrors.m` that when 'help piecewiseErrors' is typed will comment on the results when $\mathbf{P} = \{1,2,3,4,5,6\}$, $\mathbf{N} = \{4,8,16,32,64,128,256\}$ and

(a) $f(x) = \mathrm{e}^{2x}$, $[a,b] = [0,1]$;

(b) $f(x) = \frac{1}{1+25x^2}$, $[a,b] = [-5,5]$.

- The function should have the following prototype

```
function E = piecewiseErrors(a,b,P,N,f)
```

- Test your function (for example) by typing the following into the command window.

```
P = 1:3;
N = [5;10;15]
a = -1; b = 1;
E = piecewiseErrors(a,b,P,N,@(x) 1./(x+2))
```

In this case you should obtain the matrix

$$\mathtt{E} = \begin{matrix} 0.0240 & 0.0017 & 0.0001 \\ 0.0076 & 0.0003 & 0.0000 \\ 0.0037 & 0.0001 & 0.0000 \end{matrix}$$

[6 / 40]

Marks can be obtained for your `piecewiseErrors` m-file for generating the required output, for certain set(s) of inputs, as well as the plots and comments . The correctness of the following will be checked:
- The size and values of `E`;
- The error plots produced;
- The output of "help piecewiseErrors" and in particular the quality of explanation for the outputs seen.

(Run the file `GenerateYourOutputsCW3.m` to see these outputs generated for a particular set of inputs. Note that in marking your work, different input(s) may be used.)

▶ Numerical Differentiation

Recall that, given a point x and a set of distinct points $\{x_j\}_{i=0}^{p}$, such that $x_i = x$ for some $0 \le i \le p$ we can find an approximation to $f'(x)$ as follows:

$$f'(x) \approx p_p'(x) = \sum_{i=0}^{p} f(x_i) L_i'(x),$$

where $\{L_i\}_{i=0}^{p}$ are the Lagrange polynomials through the points $\{x_i\}_{i=0}^{p}$

6 • Write a *function* m-file called `derivLagrangePoly.m`, which, given a set of $p+1$ distinct nodal points $\{x_i\}_{i=0}^{p}$ and a set of n evaluation points $\{\hat{x}_j\}_{j=1}^{n}$, will return a $(p+1)\times n$ matrix, called `LDiff`, where the ijth entry of the matrix is $L'_{i-1}(\hat{x}_j)$.

- **No checks** on the inputs are required.
- Also add a brief description at the top of your `derivLagrangePoly` m-file. This description should become visible whenever one types: `help derivLagrangePoly`.
- *Hint:* Derive a formula for $L'_i(x)$ by hand involving sums and products and then implement this. **Do not** use the symbolic toolbox in Matlab.
- The function *must* have the following prototype:

```
function LDiff = derivLagrangePoly(p,x,n,xhat)
```

- Test your function (for example) by typing the following into the command window.

```
p = 3;
x = linspace(-0.5,0.5,4);
n = 6;
xhat = linspace(-1,1,6);
LDiff = derivLagrangePoly(p,x,n,xhat)
```

In this case you should obtain the matrix

$$\texttt{LDiff} = \begin{matrix} -17.8750 & -7.4350 & -1.3150 & 0.4850 & -2.0350 & -8.8750 \\ 41.6250 & 13.9050 & -0.8550 & -2.6550 & 8.5050 & 32.6250 \\ -32.6250 & -8.5050 & 2.6550 & 0.8550 & -13.9050 & -41.6250 \\ 8.8750 & 2.0350 & -0.4850 & 1.3150 & 7.4350 & 17.8750 \end{matrix}$$

Marks can be obtained for your `derivLagrangePoly` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked: **[6 / 40]**

- The size and values of `L`
- The output of "help derivLagrangePoly".

(Run the file `GenerateYourOutputsCW3.m` to see these outputs generated for a particular set of inputs. Note that in marking your work, different input(s) may be used.)

7 • Write a *function* m-file called `polyDerivative.m` that will evaluate the derivative of the pth order polynomial interpolant of a function f at a point x. The interpolating points should be equally separated so that $\{x_j\}_{j=0}^{p}$ are such that

$$x_j = x_0 + jh, \quad j = 0, \ldots, p,$$

for $h > 0$ and should be positioned so that $x_k = x$, where k is an additional input to `polyDerivative.m`.

- The function should make use of `derivLagrangePoly.m` from Q6
- Also add a brief description at the top of your `polyDerivative` m-file. This description should become visible whenever one types: `help polyDerivative`.
- The function should have the following prototype:

```
function dInterp = polyDerivative(x,p,h,k,f)
```

Here, `dInterp` is a single value containing the approximation to the derivative at x.

- Test your function (for example) by typing the following into the command window.

```
p = 3;
h = 0.1;
x = 0.5
for k=0:3
        dInterp = polyDerivative(x,p,h,k,@(x) cos(pi*x)+x)
end
```

In this case, your output should be (shown horizontally for conciseness):

```
dInterp =        dInterp =        dInterp =        dInterp =

        -2.1505       -2.1406          -2.1406          -2.1505
```

[2 / 40]

Marks can be obtained for your `polyDerivative` m-file for generating the required output, for certain set(s) of inputs. The correctness of the following will be checked:
- The correctness of `dInterp` for various inputs `x`, `p` and `k`.
- The output of ”help polyDerivative”.

(Run the file `GenerateYourOutputsCW3.m` to see these outputs generated for a particular set of inputs. Note that in marking your work, different input(s) may be used.)

$\boxed{8}$ • Write a *function* m-file called `derivativeErrors.m` that given a set of **even** polynomial degrees $\mathbf{P} = \{p_i\}_{i=1}^{l}$ and a set of widths $\mathbf{H} = \{h_j\}_{j=1}^{m}$, will return an $l \times m$ matrix E, where

$$E_{ij} = |f'(x) - p'_{p_i,h_j}(x)|, \quad 1 \le i \le l,\ 1 \le j \le m$$

and $p_{p_i,h_j}(x)$ is the polynomial interpolant of order p_i with interval width h_j such $x = x_{p_i/2}$. *i.e.* $p'_{p_i,h_j}(x)$ is the **centred** $p_i + 1$ point approximation to $f'(x)$.

• The function should also plot $\{E_{ij}\}_{j=1}^{m}$ against $\{h_j\}_{j=1}^{m}$ for each p_i. A single set of axes with a logarithmic scale on both axes should be used. The plot should have all relevant labels and legends.

• Add a description at the top of your `derivativeErrors.m` that when 'help derivativeErrors' is typed will comment on the results when $\mathbf{P} = \{2, 4, 6, 8\}$, $\mathbf{H} = \{1/4, 1/8, 1/16, 1/32, 1/64, 1/128, 1/256\}$, $f(x) = \mathrm{e}^{2x}$ and $x = 1$.

• The function must call `polyDerivative` from Q7 and should have the following prototype

```
function E = derivativeErrors(x,P,H,f,fdiff)
```

Here `fdiff` is a function that is the exact derivative of `f`.

• Test your function (for example) by typing the following into the command window.

```
P = [2,4,6];
H = [1/4,1/8,1/16];
x = 0;
format long
E = derivativeErrors(x,P,H,@(x)1./(x+2),@(x) -1./((x+2).^2)
```

In this case you should obtain

$$\mathrm{E} = \begin{matrix} 0.003968253968254 & 0.000980392156863 & 0.000244379276637 \\ 0.000264550264550 & 0.000015561780268 & 0.000000958350105 \\ 0.000043290043290 & 0.000000567028432 & 0.000000008497685 \end{matrix}$$

[4 / 40]

Marks can be obtained for your `derivativeErrors` m-file for generating the required output, for certain set(s) of inputs, as well as the plots and comments. The correctness of the following will be checked:

• The size and values of `E`;

• The error plots produced;

• The output of "help derivativeErrors" and in particular the quality of explanation for the outputs seen.

(Run the file `GenerateYourOutputsCW3.m` to see these outputs generated for a particular set of inputs. Note that in marking your work, different input(s) may be used.)