University of Nottingham
School of Mathematical Sciences

MATH4063 SCIENTIFIC COMPUTATION AND C++

Submission Date: Friday 8th January 2021, 3:00pm (GMT) **Assessed Coursework 2**

The following questions are to be used for the coursework assessment in the module MATH4063.

A single zip or tar file containing your answers to the questions below and the code you used to obtain these answers should be submitted electronically via the MATH4063 Moodle page before the deadline at the top of this page. You should follow the instructions on the accompanying Coursework Submission template which is also provided on Moodle. Since this work is assessed, your submission must be entirely your own work (see the University's policy on Academic Misconduct).

Please note that, since the original deadline for this coursework was after 18th December 2020 **the week of grace does not apply to this coursework in the usual way**. *Instead, the coursework has been distributed earlier than originally planned and the deadline has been extended to the one stated at the top of this page. Submissions up to a week after this deadline will be subject to a penalty of 5% per day. However, extenuating circumstances forms may be submitted which request deadline extensions and will be considered in the usual way.*

The style and efficiency of your programs is important. A barely-passing solution will include attempts to write programs which include some of the correct elements of a solution. A borderline distinction solution will include working code that neatly and efficiently implements the relevant algorithms, and that shows evidence of testing.

An indication is given of the weighting of each question by means of a figure enclosed by square brackets, e.g. [12]. All non-integer calculations should be done in double precision.

Background Material

Polynomial Interpolation

Let $\mathcal{P}_n$ be the set of polynomials of degree at most n. Given a set of points $\{(x_j, f_j)\}_{j=0}^n$, where $x_j \in \mathbb{R}$ are distinct, there is a unique polynomial $p_n \in \mathcal{P}_n$ satisfying $p(x_j) = f_j$ for $j = 0, \ldots, n$. One method for computing this polynomial is by constructing it from the n^{th}-order Lagrange polynomials $L_j \in \mathcal{P}_n$, which gives

$$p_n(x) \;=\; \sum_{j=0}^{n} f_j\, L_j(x) \qquad \text{where} \qquad L_j(x) \;=\; \frac{\prod_{k\neq j} x - x_k}{\prod_{k\neq j} x_j - x_k}\,. \tag{1}$$

This can be used to approximate a function $f : \mathbb{R} \to \mathbb{R}$ by taking $f_j = f(x_j)$.

Best Approximation in the 2-Norm

A polynomial $p_n \in \mathcal{P}_n$ is the polynomial of best approximation of degree n to a function $f \in L^2(a,b)$ in the 2-norm if and only if

$$\int_a^b (f - p_n)\, \phi \;\mathrm{d}x \;=\; 0 \qquad \forall \phi \in \mathcal{P}_n\,. \tag{2}$$

Note that $L^2(a,b)$ is the linear space of all real-valued functions defined on (a,b) such that $|f(x)|^2$ is integrable on (a,b).

Since the set of n^{th}-order Lagrange polynomials forms a basis of $\mathcal{P}_n$ the condition (2) only needs to be tested for those polynomials, *i.e.*

$$\int_a^b p_n\, L_i \;\mathrm{d}x \;=\; \int_a^b f\, L_i \;\mathrm{d}x \qquad \text{for } i = 0, \ldots, n. \tag{3}$$

Furthermore, any $p_n \in \mathcal{P}_n$ can be written as a linear combination of the Lagrange basis functions, so

$$p_n \;=\; \sum_{j=0}^{n} (p_n)_j\, L_j \qquad \text{and} \qquad \sum_{j=0}^{n} (p_n)_j \int_a^b L_j\, L_i \;\mathrm{d}x \;=\; \int_a^b f\, L_i \;\mathrm{d}x \tag{4}$$

for $i = 0, \ldots, n$. The result is a linear system of equations $\mathbf{A}\mathbf{p} = \mathbf{f}$, where

$$\mathbf{A}_{ij} \;=\; \int_a^b L_j\, L_i \;\mathrm{d}x\;, \qquad \mathbf{p}_j \;=\; (p_n)_j\;, \qquad \mathbf{f}_i \;=\; \int_a^b f\, L_i \;\mathrm{d}x\;, \tag{5}$$

which can be solved to find the coefficients $(p_n)_j$ of the Lagrange basis functions. In general, quadrature has to be used to compute the elements of $\mathbf{A}$ and $\mathbf{f}$.

Discontinuous Piecewise Polynomial Approximation

The best approximation in the 2-norm can be computed over the whole interval (a, b) in which f is to be approximated. However, as the degree of the polynomial is increased to provide more accurate approximations, the accuracy of the quadrature rules used to construct the system of equations must also be increased. An alternative is to divide the interval in to subintervals, and to construct the best approximation in the 2-norm on each of these subintervals. This will result in a piecewise polynomial approximation over the whole interval.

The result of this procedure will, in general, be a discontinuous function. It is possible to construct a best continuous piecewise polynomial approximation in the 2-norm. This will not be requested in this coursework but the finite element method implemented in Coursework 1 gives precisely this method when it is used to approximate the equation $u = f(x)$.

Numerical Integration (Quadrature)

A general N_q-point quadrature rule for approximating a one-dimensional definite integral of a function f over a closed interval $[a, b]$ is given by

$$\int_a^b f(x) \;\mathrm{d}x \;\approx\; \sum_{q=1}^{N_q} w_q\, f(x_q)\,, \tag{6}$$

in which x_q are the integration points and w_q are the corresponding weights.

Many integration rules are defined over a specific interval and must be rescaled before applying the quadrature rule over a general interval $[a, b]$. For example, the points and weights for Gaussian quadrature rules are usually defined for the interval $[-1, 1]$, but can be applied to a general interval by evaluating

$$\int_a^b f(x) \;\mathrm{d}x \;\approx\; \frac{b-a}{2} \sum_{q=1}^{N_q} w_q\, f(x_q) \qquad \text{where} \qquad x_q \;=\; \frac{b-a}{2}\,\xi_q + \frac{b+a}{2} \tag{7}$$

and ξ_q are the quadrature points defined on the interval $[-1, 1]$.

Coursework Questions

In `Templates/` you will find a set of folders, one for each question. The folders contain a small amount of code (`.hpp` and `.cpp` files) as well as empty files, which you must edit for the coursework. You can use any software you want to produce the plots requested below.

You must keep the folder structure and all file names as they are in the templates: the folder `Q1` in your submission, for instance, should be self-contained, and should include all the code necessary to compile and reproduce your results for Question 1. The template folders may also serve as a checklist for your submission. As part of your submission, you may also add files to the folders (for instance new classes, output files, plotting routines, etc.): if you do so, then write a brief `README.txt` file, containing a short description of each new file. When you attempt Question 2, make a new folder and put all the files necessary to produce your results in it; if needed, copy some files from `Q1` to `Q2`, etc.

This coursework requires you to implement function approximation algorithms in an object-oriented manner, then use them to approximate the following functions over the interval $x \in [0, 1]$:

$$f_1(x) = x^3\,, \quad f_2(x) = \sin(2x)\,, \quad f_3(x) = \frac{1}{1+25(2x-1)^2}\,, \quad f_4(x) = \sqrt{x}\,. \tag{8}$$

1. In this question you will use Lagrange polynomials, as defined in Equation (1), to compute interpolants of functions.

   (a) Write an abstract class `AbstractApproximator` which contains the following members:

   - Protected variables for the bounds of the interval over which the function will be approximated

     ```
     double mXmin;
     double mXmax;
     ```

   - A protected variable for the number of interpolation points

     ```
     int mNpoints;
     ```

   - A protected pointer for the function to be approximated

     ```
     double (*mpFunction) (double);
     ```

   - A pure virtual public method

     ```
     virtual void Approximate(const int nxvalues) = 0;
     ```

     in which `nxvalues` is the number of x values at which to print out the value of the interpolant to a file (for plotting).
   - Any other member that you choose to implement. Your choices and your ability to design this class according to object-orientation design principles will be assessed.

[15]

(b) Write a class Lagrange, derived from AbstractApproximator, with the following members:

- A public constructor

```
    Lagrange(double (*pFunction) (double),
        const double xmin, const double xmax,
        const int npoints,
        const std::string outputFileName="output.dat");
```

  which defines the function to be approximated, the interval over which it will be approximated, the number of interpolation points ($n+1$, where n is the polynomial degree) and the output file name.

- A public approximation method

```
    void Approximate(const int nxvalues);
```

  which computes the values of the Lagrange polynomial interpolant at nxvalues uniformly spaced points in the interval [xmin,xmax] and saves them to a file. This method should also compute the maximum error in the approximation at these nxvalues points (an approximation to the ∞-norm error) and print it to the screen.

- Any other member that you choose to implement. Your choices and your ability to design this class according to object-orientation design principles will be assessed.

[15]

(c) Write and execute a main Driver.cpp file which:

  i. Approximates each of the four functions (8) on the interval $x \in [0,1]$ using Lagrange interpolation through 10 uniformly spaced interpolation points, including the end-points of the interval, and prints the maximum error to the screen. You should include these error values in your report, along with four plots which compare the interpolant with the original function, produced from the data saved to file with nxvalues=101.

  ii. Approximates each of the four functions (8) on the interval $x \in [0,1]$ using Lagrange interpolation of different polynomial degrees. You should use uniformly spaced interpolation points, including the end-points of the interval, and print the maximum error to the screen, using nxvalues=101. You should include these error values in your report, making it clear which functions and polynomial degrees they relate to.

  Comment on how the maximum error depends on the degree of the polynomial interpolant for each of the functions in (8). You should use the error data you show to demonstrate any patterns you see in your results and any differences you see in the patterns for different functions. You should also describe briefly the properties of the functions which cause these patterns.

[10]

Note that you will need to use arrays for this coursework. This coursework has been written assuming use of the classes Vector and Matrix, which are provided and have been seen in the lecture material. However, you will not be penalised if you correctly use standard arrays and/or vectors instead.

2. In this question you will use quadrature to approximate integrals of functions and products of functions (of the form seen in the linear system derived from least-squares approximation (5)).

   (a) Write an abstract class `AbstractQuadratureRule` which contains the following members:

   - Protected variables for the bounds of integration

   ```
       double mXmin;
       double mXmax;
   ```

   - A protected pointer for the function to be integrated

   ```
       double (*mpFunction) (double);
   ```

   - Three pure virtual public methods

   ```
       virtual double IntegrateFunction() = 0;
       virtual double IntegrateRHSProduct(const int i,
           int npoints, Vector* pPoints) = 0;
       virtual double IntegrateMatrixProduct(const int i,
           const int j, int npoints, Vector* pPoints) = 0;
   ```

   which integrate the functions f, fL_i and L_iL_j, respectively. The Lagrange basis functions, L_i and L_j, are defined using `npoints` points at positions `pPoints`.

   - A protected method for evaluating the Lagrange basis function L_j

   ```
       double EvaluateLagrangeBasis(const double xval,
           const int j, const int npoints,
           const Vector* pPoints) const;
   ```

   - Any other member that you choose to implement. Your choices and your ability to design this class according to object-orientation design principles will be assessed.

   [10]

   (b) Write two classes, `Simpson` and `Gauss4point`, both of which should be derived from `AbstractQuadratureRule`, which implement Simpson's rule and the 4-point Gauss rule for approximating integrals.

   Each class should have an appropriate specialised constructor and functions which implement the three pure virtual methods for the specified quadrature rule.

   [15]

(c) Write and execute a main `Driver.cpp` file which:

i. Approximates the integrals of each of the four functions (8) over the interval $[0, 1]$ and outputs the result to the screen for each of the two quadrature rules.

ii. Approximates the vector $\mathbf{f}$ in (5) for the function $f_1(x) = x^3$ over the interval $[0, 1]$ and outputs the result to the screen for each of the two quadrature rules. You should compute answers for linear and quadratic Lagrange basis functions, for which the exact answers are, respectively,

$$\mathbf{f} = \frac{1}{20}\begin{pmatrix} 1 \\ 4 \end{pmatrix} \qquad \text{and} \qquad \mathbf{f} = \frac{1}{60}\begin{pmatrix} -1 \\ 8 \\ 8 \end{pmatrix}. \tag{9}$$

You should include these results in your report.

iii. Approximates the matrix $\mathbf{A}$ in (5) for each of the two quadrature rules. You should compute answers for linear and quadratic Lagrange basis functions, for which the exact answers are, respectively,

$$\mathbf{A} = \frac{1}{6}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \qquad \text{and} \qquad \mathbf{A} = \frac{1}{30}\begin{pmatrix} 4 & 2 & -1 \\ 2 & 16 & 2 \\ -1 & 2 & 4 \end{pmatrix}. \tag{10}$$

You should include these results in your report.

iv. Solves the system $\mathbf{Ap} = \mathbf{f}$ in each of the above cases. You should include these results in your report.

A method `GaussianElimination` is provided which solves a linear system of equations using Gaussian elimination. Pivoting is not required for this coursework. The method uses the `Vector` and `Matrix` classes, so you may need to alter it if you are using standard arrays or vectors elsewhere in your code, or find another method which implements the same algorithm.

Comment on how your results compare with the exact values for $\mathbf{A}$ and $\mathbf{f}$. You should provide a reason for any differences you see.

[5]

Note that the solutions given in Equations (9) and (10) may also be used in later questions if you are unable to get your methods for computing them working for this question. You can show results for quadratic polynomials instead of cubic polynomials. However, if you do that you will not be able to achieve full marks on the later questions.

3. In this question you will re-use and modify the classes developed in Questions 1 and 2 to find the best polynomial approximation to a function in the 2-norm.

   (a) Write a class `BestL2Fit`, derived from `AbstractApproximator`, with the following members:

   - A public constructor

```
    BestL2Fit(double (*pFunction) (double),
        AbstractQuadratureRule* pIntegrator,
        const double xmin, const double xmax,
        const int npoints,
        const std::string outputFileName="output.dat");
```

     which defines the function to be approximated, the quadrature rule to be used, the interval over which it will be approximated, the number of interpolation points ($n+1$, where n is the degree of the polynomials used to find the best approximation in Equations (4) and (5)) and the output file name.

   - A public approximation method

```
    void Approximate(const int nxvalues);
```

     which computes the vector $\mathbf{p}$ for the best polynomial approximation in the 2-norm to the given function on the interval `[xmin,xmax]` and writes the values of this approximation at `nxvalues` uniformly spaced points in that interval to a file.

   - Any other member that you choose to implement. Your choices and your ability to design this class according to object-orientation design principles will be assessed.

   [10]

   (b) Write and execute a main `Driver.cpp` file which:

   i. Approximates each of the four functions (8) on the interval $x \in [0,1]$ using the best cubic polynomial approximation in the 2-norm and prints the maximum error and an approximation to the 2-norm of the error to the screen. You should use the 4-point Gauss rule to carry out the quadrature and `nxvalues=101`.

   You should use the data at the `nxvalues` points used when saving the output to a file to approximate the 2-norm of the error,

$$||f-p||_2 = \left(\int_a^b (f(x)-p(x))^2 \, \mathrm{d}x\right)^{\frac{1}{2}} \approx \left(\frac{b-a}{N}\sum_{i=1}^{N}(f(x_i)-p(x_i))^2\right)^{\frac{1}{2}}. \quad (11)$$

   You should include these error values in your report, along with four plots which compare the best approximation polynomial with the original function, produced from the data saved to file.

   Use your code to find the best quartic polynomial approximation in the 2-norm and comment on how your result compares with the exact solution for the function f_1.

   [5]

4. In this question you will modify the class developed in Question 3 to find the best discontinuous piecewise polynomial approximation to a function in the 2-norm.

   (a) Write a class `LocalBestL2Fit`, derived from `AbstractApproximator`, with the following members:

   - A public constructor

   ```
       LocalBestL2Fit(double (*pFunction) (double),
           AbstractQuadratureRule* pIntegrator,
           const double xmin, const double xmax,
           const int npoints, const int nintervals,
           const std::string outputFileName="output.dat");
   ```

     which defines the function to be approximated, the quadrature rule to be used, the interval over which it will be approximated, the number of interpolation points to be used in each subinterval, the number of subintervals and the output file name.

   - A public approximation method

   ```
       void Approximate(const int nxvalues);
   ```

     which computes the vector $\mathbf{p}$ for the best polynomial approximation in the 2-norm to the given function on each of the subintervals and writes the values of this approximation to a file, using `nxvalues` uniformly spaced points in each subinterval.

   - Any other member that you choose to implement. Your choices and your ability to design this class according to object-orientation design principles will be assessed.

   [10]

   (b) Write and execute a main `Driver.cpp` file which:

   i. Approximates each of the four functions (8) on the interval $x \in [0, 1]$ using the best discontinuous linear polynomial approximation in the 2-norm with 5 subintervals of equal length and prints the maximum error and an approximation to the 2-norm error to the screen. You should use the 4-point Gauss rule to carry out the quadrature and `nxvalues=101`. You should include the error values in your report, making it clear which functions they relate to, along with four plots which compare the discontinuous piecewise polynomial approximation with the original function.

   ii. Approximates each of the four functions (8) on the interval $x \in [0, 1]$ using the best discontinuous linear and cubic polynomial approximations in the 2-norm with 2^n subintervals of equal length, for $n = 1, 2, 3, \ldots$ Your code should print the maximum error and an approximation to the 2-norm error to the screen and you should include these error values in your report, making it clear which functions, polynomial degrees and subinterval lengths they relate to. You should again use `nxvalues=101`

   Comment on how the errors for linear and cubic approximations depend on the number of subintervals for each of the functions in (8). You should use the error data you show to demonstrate any patterns you see in your results and any differences you see in the patterns for different functions. You should also provide brief explanations for the patterns you see.

   [5]

The output requested in Questions 1c, 2c, 3b, 4b should be included in your submission, in the format provided by the solution template file. Please be selective in the output you present: none of the questions should require more than 100 lines of code output to be presented (most need significantly fewer than that).