The University of Nottingham

SCHOOL OF MATHEMATICAL SCIENCES

AUTUMN SEMESTER 2024-2025

MATH4063 - SCIENTIFIC COMPUTING AND C++

Coursework 1

Deadline: 15:00, Monday 18/11/2024

As this work is assessed, your submission must be entirely your own work (see the University's policy on Academic Misconduct).

The marks for each question are given by means of a figure enclosed by square brackets, eg [20]. There are a total of 100 marks available for the coursework and it contributes 45% to the module. The marking rubric available on Moodle will be applied to each full question to further break down this mark.

You are free to name the functions you write as you wish, but bear in mind these names should be meaningful. Functions should be grouped together in `.cpp` files and accessed in other files using correspondingly named `.hpp` files.

All calculations should be done in double precision.

A single zip file containing your full solution should be submitted on Moodle through the Coursework 1 submission box. This zip file should contain three folders called `main`, `source` and `include`, with the following files in them:

main:

- q1d.cpp
- q2c.cpp
- q3c.cpp
- q4d.cpp

source:

- linear_algebra.cpp
- legendre.cpp
- polygon.cpp
- gauss_quadrature.cpp
- optimisation.cpp

include:

- linear_algebra.hpp
- legendre.hpp
- polygon.hpp
- gauss_quadrature.hpp
- optimisation.hpp

Prior to starting the coursework, please download the `CW1_code.zip` from Moodle and extract the files. More information about the contents of the files included in this zip file is given in the questions below.

Hint: When using a C++ `struct` with header files, the whole struct needs to be defined fully in the header file, and the header file included in the corresponding `.cpp` file. Include guards should also be used.

In this coursework you will build an 'optimised' quadrature (numerical integration) rule for a general polygonal domain Ω. That is, find $Q^\Omega = \{(\boldsymbol{x}_i, w_i)\}_{i=1}^n$, with quadrature weights $w_i > 0$ and quadrature points $\boldsymbol{x}_i \in \Omega$, such that

$$\int_\Omega f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \sum_{i=1}^n w_i f(\boldsymbol{x}_i).$$

The quadrature is optimised in the sense that n is close to having the fewest possible points that will integrate exactly a polynomial function of a given order.

In order to solve this problem, you will first construct a suitable quadrature over a general triangular domain, then apply this quadrature to each of the sub-triangles of the polygonal domain Ω to obtain a non-optimised quadrature. An optimisation algorithm will then be applied to improve the quadrature by removing some of the quadrature points $\boldsymbol{x}_i$ and recalculating the weights.

1. Suppose we wish to find an approximation to the integral

$$I = \int_{-1}^1 f(x)\,\mathrm{d}x.$$

Gauss quadrature provides a set of n points and weights $\{x_i, w_i\}_{i=1}^n$ that will integrate polynomials of degree $2n-1$ exactly over $(-1,1)$ (in real arithmetic).

The Gauss quadrature points are the roots of the nth order Legendre polynomial $L_n(x)$. These Legendre polynomials $\{L_i(x)\}_{i\geq 0}$ are orthogonal over the interval $(-1,1)$ so that:

$$\int_{-1}^1 L_i(x)L_j(x)\,\mathrm{d}x = 0, \quad \text{for } i \neq j.$$

Furthermore, the Legendre polynomials can be constructed using the recurrence relationship:

$$\begin{aligned} L_0(x) &= 1, \\ L_1(x) &= x, \\ nL_n(x) &= (2n-1)xL_{n-1}(x) - (n-1)L_{n-2}(x), \quad \text{for } n \geq 2. \end{aligned}$$

Once an x_i has been found, the corresponding weight w_i can be computed via the formula:

$$w_i = \frac{2}{(1-x_i^2)[L_n'(x_i)]^2}.$$

(a) In `legendre.cpp` is a function that will compute the value of $L_n(x)$, given n and x. Write a similar function (based on a recurrence relation) that will compute $L_n'(x)$ given n and x.

(b) In `legendre.cpp`, write a function that, given an initial guess x_0, will use the Newton-Raphson method to find an approximation to a root of $L_n(x)$. You can assume convergence when the iterates $\{x_m\}_{m>0}$ first satisfy $|x_m - x_{m-1}| < \text{Tol}$, where Tol is an input to the function.

(c) In `gauss_quadrature.cpp`, write a function that will construct the one-dimensional Gauss quadrature, $\{x_i, w_i\}_{i=1}^n$ for a given number of points n. As initial guesses to find $\{x_i\}_{i=1}^n$, you should use the roots of the nth order Chebyshev polynomials:

$$\hat{x}_i = \cos\left(\frac{\pi\left(i-\frac{1}{2}\right)}{n}\right), \quad i = 1, \ldots, n.$$

(d) In `Q1d.cpp`, write a main program that will ask the user to supply a number of quadrature points n, will construct and print to the screen the Gauss quadrature rule with n points and also will return the approximation to the following integrals using the quadrature:

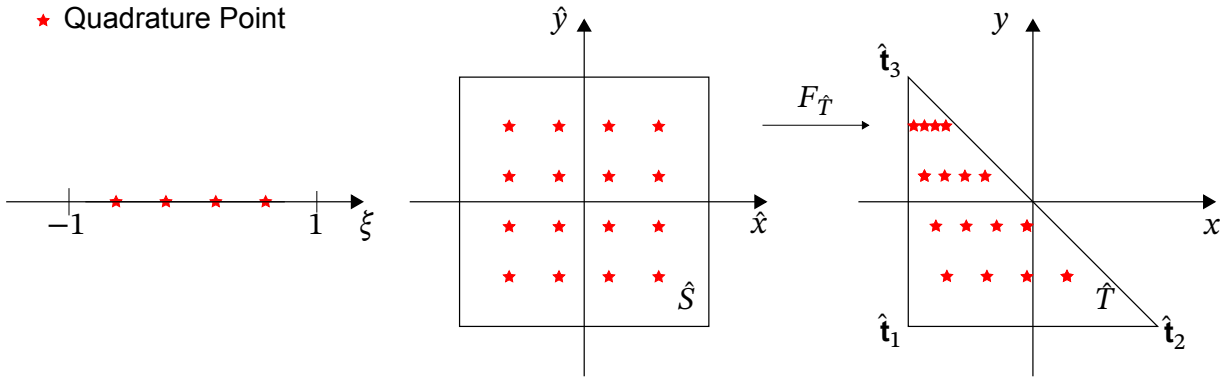
$$I_1 = \int_{-1}^{1} \mathrm{e}^x \, \mathrm{d}x,$$
$$I_2 = \int_{-1}^{1} \cos\left(\frac{\pi x}{2}\right) \mathrm{d}x.$$

[20 marks]

2. A 1D quadrature can be used to construct a 2D quadrature on a square. Suppose we have the 1D quadrature $\hat{Q} = \{\xi_i, \tilde{w}_i\}_{i=1}^n$, on the interval $(-1, 1)$, then a 2D quadrature $Q^{\hat{S}} = \{(\hat{x}_i, \hat{y}_j), \hat{w}_{i,j}\}_{i,j=1}^n$ on the square domain $\hat{S} := (-1, 1)^2$ can be constructed as follows:

$$(\hat{x}_i, \hat{y}_j) = (\xi_i, \xi_j),$$
$$\hat{w}_{i,j} = \tilde{w}_i \tilde{w}_j.$$

Once this quadrature has been constructed, a mapping $F_{\hat{T}} : \hat{S} \to \hat{T}$ where $\hat{T}$ is the triangle with vertices $\hat{\mathbf{t}}_1 = (-1, -1)$, $\hat{\mathbf{t}}_2 = (1, -1)$, $\hat{\mathbf{t}}_3 = (-1, 1)$ can be used to construct a quadrature $Q^{\hat{T}}$ on $\hat{T}$. The idea is shown below.



(a) In `gauss_quadrature.cpp`, write a function that will construct the Gauss quadrature on the square $\hat{S}$ with n quadrature points in each direction.

(b) By defining a suitable mapping from $\hat{S}$ to $\hat{T}$, in `gauss_quadrature.cpp`, write a function that will construct a quadrature on $\hat{T}$ with n quadrature points in each coordinate direction.

(c) In `Q2c.cpp` write a main function that will use the quadrature on both $\hat{S}$ and $\hat{T}$ to integrate a function $f(x, y)$ defined as a C++ function with prototype:

```
double func(double x, double y);
```

Your code should ask the user to enter the number of quadrature points n to be used in each coordinate direction and print to the screen the value of the integrals over both $\hat{S}$ and $\hat{T}$ for the following functions:

$$f(x, y) = 3x^3y^2 + 2x^2y + 2x;$$
$$f(x, y) = \sin(\pi(x + y));$$
$$f(x, y) = \mathrm{e}^{x^2+y^2}.$$

Hint: You may wish to define a struct to store your quadratures. [30 marks]

3. Once a quadrature $Q^{\hat{T}}$ has been constructed on $\hat{T}$, a further mapping F_T that maps $\hat{T}$ to any general triangle T, with vertices $(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$ (ordered anticlockwise) can be used to construct a quadrature Q^T.

   For a polygon $\Omega = \cup_{i=1}^m T_i$ composed of triangles T_i, a quadrature Q^Ω on Ω can be constructed by combining the quadratures Q^{T_i} on each of T_i. Hence,

$$Q^T = \bigcup_{i=1}^{m} Q^{T_i}.$$

   In `polygon.hpp` you will find definitions for two `structs`. `Triangle` defines a triangle data structure, while `GeneralPolygon` defines a polygonal structure composed of triangles.

   (a) By defining a suitable affine (linear) map $F_T : \hat{T} \to T$ such that $F_T(\hat{\mathbf{t}}_i) = \mathbf{t}_i$, $i = 1, 2, 3$, write a C++ function that will construct a quadrature Q^T based on $Q^{\hat{T}}$. As input, the function should take in a variable of type `Triangle` and the number of points n in each coordinate direction.

   (b) Write a function that, given a general polygonal domain Ω stored as variable of type `GeneralPolygon` and a number of points n, will construct Q^Ω by applying the function from part (b) to each of the T_i.

   (c) In `polygon.cpp` are three functions that will generate a triangular subdivision of a square, an L-shaped domain and a regular hexagon.

   In `Q3d.cpp`, write a main function that will ask the user which polygon to construct a quadrature on, the number of 'refinements' of the basic triangulation to perform and the number of quadrature points to use in each coordinate direction per sub-triangle. The code should print the quadrature to the screen and give approximations using the quadrature to the same integrals as in Q2.

[30 marks]

4. The method for constructing Q^Ω from Q3 will produce a quadrature that integrates polynomials exactly over Ω and with positive weights. However, this quadrature is likely to have many more quadrature points than is necessary, resulting in slow application of the quadrature. We would like to be able to optimise the quadrature by removing some of the points and redefining the corresponding weights of the remaining points, but with the quadrature still being able to integrate polynomials to a given order. We would also like the weights to remain positive.

   Suppose that $A \in \mathbb{R}^{m\times n}$ with $m \leq n$ and $\mathbf{b} \in \mathbb{R}^m$ and suppose we would like to find $\mathbf{x} \in \mathbb{R}^n$ such that

$$A\mathbf{x} = \mathbf{b},$$

   with the constraint $x_i \geq 0$, $i = 1, \ldots, n$.

   One algorithm to find $\mathbf{x}$ that ensures all entries are positive is shown in Algorithm 1. The algorithm works on an active/inactive set principle, where to be active means $x_i = 0$ and to be inactive means $x_i > 0$. Throughout the algorithm, it is assumed that:

   - P is a set that contains the indices of the inactive variables. If $p = \text{card}\, P$ (*i.e* the number of indices in P), then,

$$A^P := [A_{ij}]_{j\in P}.$$

     That is, $A^P \in \mathbb{R}^{m\times p}$ is the matrix comprising the columns of A whose indices correspond to those in P.
   - For a vector $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{w}^P \in \mathbb{R}^p$ denotes $\mathbf{w}$ but restricted to being those entries with indices in P.
   - For $\mathbf{w}^P \in \mathbb{R}^p$, $\mathbf{w} \in \mathbb{R}^n$ is the extension of $\mathbf{w}^P$ so that $w_i = 0$ for $i \notin P$.
   - For a matrix B and a right hand side $\mathbf{c}$, to solve $B\mathbf{y} = \mathbf{c}$ means to solve the equation in the least squares sense (see Unit 6 notes).

Algorithm 1 Optimisation Algorithm

Require: $A \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\epsilon > 0$.
  \\ Initialise
  $P = \varnothing$
  $\mathbf{x} = \mathbf{0}_n$
  $\mathbf{v} = A^T(\mathbf{b} - A\mathbf{x})$
  while card $P < n$ and $\max(\mathbf{v}) > \epsilon$ **do**
    $j = \arg\max_{k\notin P}(\mathbf{v} = \{v_k\})$
    $P = P \cup \{j\}$
    \\ Solve the restricted problem
    Solve $A^P\mathbf{s}^P = \mathbf{b}$

    \\ Correct any negative values
    while $\min(\mathbf{s}^P) \leq 0$ **do**
      $\alpha = \min\left(\frac{x_i}{x_i - s_i}\right)$, for $s_i \in \mathbf{s}^P$ and $s_i \leq 0$.
      $\mathbf{x} = \mathbf{x} + \alpha(\mathbf{s} - \mathbf{x})$
      $P = P\backslash\{j\}$ for j such that $x_j = 0$

      \\ Solve the restricted problem
      Solve $A^P\mathbf{s}^P = \mathbf{b}$
    end while

    \\ Update
    $\mathbf{x} = \mathbf{s}$
    $\mathbf{v} = A^T(\mathbf{b} - A\mathbf{x})$
  end while

(a) In `optimisation.cpp`, write a function that will implement Algorithm 1. Some helpful functions and `structs` can be found in `linear_algebra.cpp/linear_algebra.hpp`, although you will need to add some extra functions to these files to make the optimiser work. After each iteration, the function should print to the screen the iteration number and the norm of the residual $\|\mathbf{b} - A\mathbf{x}\|_2$.

(b) Let $\{\phi_k\}_{k=1}^{(p+1)^2}$ be a basis for polynomials of degree p in 2D, where

$$\phi_k(x, y) = L_i(x)L_j(y), \quad k = i(p+1) + j + 1, 0 \leq i, j \leq p.$$

Then, to find a set of optimised quadratures weights (keeping the points fixed), the optimisation algorithm from part (a) can be applied in the case where A and $\mathbf{b}$ are defined as

$$A_{kj} = \phi_k(x_j, y_j) \quad ((x_j, y_j), w_j) \in Q^\Omega \text{ and } 1 \leq k \leq (p+1)^2,$$
$$b_k = \int_\Omega \phi_k(x, y)\,\mathrm{d}x\,\mathrm{d}y, \quad 1 \leq k \leq (p+1)^2.$$

Write a function that, given a quadrature and a polynomial degree p, will construct the matrix A and vector $\mathbf{b}$.

(c) Using parts (a) and (b), write a function that will produce an optimised quadrature Q^Ω_{opt} for a given polygonal domain Ω (stored as a `GeneralPolygon`). The function should also take as input the degree of polynomial that should be integrated exactly. A tolerance of $\epsilon = 1.0\times10^{-12}$ can be set in Algorithm 1.

Note, the algorithm should produce some 0 weights; the corresponding quadrature points should be removed from the final quadrature.

(d) In `Q4d.cpp`, write a main program that will ask the user to select which of the polygonal domains from Q3 to construct an optimised quadrature for and the polynomial degree that should be integrated

exactly. The quadrature should be printed to the screen as well as the approximate values of the integrals from Q2.

[20 marks]