Queen Mary
University of London

Main Examination period 2021/22

MTH6150: Numerical Computing in C and C++

Assignment date: Monday 14/3/2022
Submission deadline: Friday 6/5/2022 at 23:59 BST

The coursework is due by **Friday, 6th May 2022 at 23:59 BST**. Please submit a **report (in `pdf` format)** containing answers to all questions, complete with written explanations and printed tables or figures. Tables should contain a label for each column. Figures must contain a title, axis labels and legend. Please provide a brief explanation of any original algorithm used in the solution of the questions (if not discussed in lectures). **Code comments** should be used to briefly explain your code. You must show that your program works using suitable examples. Build and run your code with any of the free IDEs discussed in class (such as Visual Studio, Xcode, CLion, or an online compiler), and **include the code and its output in your report**. **The C++ code for each question must be pasted into the `pdf` file of your report**, so that it can be commented on when marking. The code used to answer each question should also be submitted separately, together with the report, in a single `cpp` file or multiple `cpp` files. You may organise the code in different directories, one for each question. Late submissions will be treated in accordance with current College regulations. **QMPlus automatically screens for plagiarism. Plagiarism is an assessment offence and carries severe penalties.** In the questions below, use `long double` if your compiler supports it. If this is not supported, e.g. if you are using Visual Studio, you may use `double`.
Please read the instructions above carefully to avoid common mistakes. If in doubt, please ask. *Reports that do not contain C++ code in it (but have code included separately in .cpp files) are subject to a 25% penalty. Reports that consist only of code and no explanation are subject to a 40% penalty. C++ code submitted without a report is subject to a 50% penalty. Reports not accompanied by any C++ code cannot be evaluated and will receive 0 marks. Only material submitted through QMPlus will be accepted.*

Coursework [100 marks]

Question 1. [20 marks] *Self-consistent iteration.*
Solve the transcendental equation

$$x = e^{-x}$$

using fixed-point iteration. That is, using the initial guess $x_0 = 1$, obtain a sequence of real numbers

$$\begin{aligned} x_1 &= e^{-x_0} \\ x_2 &= e^{-x_1} \\ &\vdots \\ x_{50} &= e^{-x_{49}} \\ &\vdots \end{aligned}$$

which tends to the value $x_\infty = 0.567143\ldots$, that is, a root of the function $f(x) = x - e^{-x}$. Formally, the iteration can be written as

$$x_{i+1} = e^{-x_i} \text{ for } i = 0, 1, 2, \ldots \text{ with } x_0 = 1.$$

The limit x_∞ satisfies $f(x_\infty) = 0$.

(a) Write a `for` loop that performs the above iteration, starting from the initial condition $x_0 = 1$. Use an `if` and a `break` statement to stop the loop when the absolute value of the difference $|x_{i+1} - x_i|$ between two consecutive iterations is less than a prescribed tolerance $\epsilon = 10^{-15}$. Use a `long double` to store the values of x, and the change in x, between iterations. Use `setprecision(18)` to print out the final value of x to 18 digits of accuracy. [**10**]

(b) In how many iterations did your loop converge? [**5**]

(c) What is the final error in the above transcendental equation? [Hint: use the final value of x to compute and display the difference $x - e^{-x}$.] Is the error what you expected (i.e. is it smaller than ϵ)? [**5**]

Question 2. [20 marks] *Inner products, sums and norms.*

(a) Write a function named `dot` that takes as input two vectors $\vec{A} = \{A_1, ..., A_n\} \in \mathbb{R}^n$ and $\vec{B} = \{B_1, ..., B_n\} \in \mathbb{R}^n$ (of type `valarray<long double>`) and returns their inner product

$$\vec{A} \cdot \vec{B} = \sum_{i=1}^{n} A_i B_i \tag{1}$$

as a `long double` number. **[5]**

(b) Use this function to compute the sum $\sum_{i=1}^{n} \frac{1}{i^2}$ for $n = 10^6$. To do so, you may define a Euclidean n-vector `A=`$\{$`1,1/2,1/3,...,1/1000000`$\}$ as a `valarray<long double>` and compute the inner product $\vec{A} \cdot \vec{A}$. Print the difference $\vec{A} \cdot \vec{A} - \pi^2/6$ between your result and the exact value of the infinite sum on the screen.

Remark: Define $\pi = 3.1415926535897932385$ to `long double` accuracy. **[5]**

(c) Repeat Question 2(a) using compensated (Kahan) summation and name your function `cdot`. Use the old `dot` function and the new `cdot` function to compute the sum $\sum_{i=1}^{n} c^2$ for $n = 10^6$ and $c = 0.1$.

Hint: Define a constant Euclidean n-vector `C=`$\{$`c,c,...,c`$\}$ as a `valarray<long double>` of size n and compute the inner product $\vec{C} \cdot \vec{C}$.

Print the difference between your result and the exact value nc^2. **[5]**

(d) Write code for a function object that has a member variable m of type `int`, a suitable constructor, and a member function of the form
`double operator()(const valarray<long double> A) const {`
which returns the weighted norm

$$l_m(\vec{A}) = \sqrt[m]{\sum_{i=1}^{n} |A_i|^m} = \left(\sum_{i=1}^{n} |A_i|^m \right)^{1/m} \tag{2}$$

Use this function object to calculate the norm $l_2(\vec{A})$ for the n-vector $\vec{A}$ in Question 2b. Does the quantity $l_2(\vec{A})^2$ equal the inner product $\vec{A} \cdot \vec{A}$ that you obtained above? [Note: half marks awarded if you use a regular function instead of a function object.] **[5]**

Question 3. [20 marks] *Finite differences.*

(a) Write a C++ program that uses finite difference methods to numerically evaluate the second derivative $f''(x)$ of a function $f(x)$ whose values on a fixed grid of points are specified $f(x_i) = f_i,\ i = 0, 1, ..., N$. Your code should use `valarray<long double>` containers to store the values of x_i, f_i and f''_i. Assume the grid-points are located at $x_i = (2i - N)/N$ on the interval $[-1, 1]$ and use 2nd order finite differencing to compute an approximation f''_i for $f''(x_i)$:

$$\begin{aligned} f''_0 &= \frac{f_{i+2} - 2f_{i+1} + f_i}{\Delta x^2} \quad \text{if} \quad i = 0 \\ f''_i &= \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} \quad \text{if} \quad i = 1, 2, ..., N-1 \\ f''_N &= \frac{f_i - 2f_{i-1} + f_{i-2}}{\Delta x^2} \quad \text{if} \quad i = N \end{aligned}$$

with $\Delta x = 2/N$. Demonstrate that your program works by evaluating the second derivatives of a known function, $f(x) = e^{-16x^2}$, with $N + 1 = 64$ points. Compute the difference between your numerical derivatives and the known analytical ones:

$$e_i = f''_{\text{numerical}}(x_i) - f''_{\text{analytical}}(x_i)$$

at each grid-point. Output the values x_i, f_i, f''_i, e_i of each `valarray<long double>` on the screen and tabulate (or plot) them in your report to 10 significant digits of accuracy. Comment on whether the error is what you expected. **[10]**

(b) For the same choice of $f(x)$, demonstrate 2nd-order convergence, by showing that, as N increases, the mean error $\langle e \rangle$ decreases proportionally to $\Delta x^2 = (2/N)^2$. You may do so by tabulating the quantity $N^2 \langle e \rangle$ for different values of N (e.g. $N + 1 = 16,\ 32,\ 64,\ 128$) and checking if this quantity is approximately constant. (Alternatively, you may plot $\log\langle e \rangle$ vs. $\log N$ and check if the dependence is linear and if the slope is -2.) Here, the mean error $\langle e \rangle$ is defined by

$$\langle e \rangle = \frac{1}{N+1} \sum_{i=0}^{N} |e_i| = \frac{1}{N+1} l_1(\vec{e}).$$

Hint: you may use your code from Question 2d to compute the mean error. **[10]**

Question 4. [20 marks] *Numerical integration.*
We wish to compute the definite integral

$$I = \int_a^b \sqrt{(b-x)x}\,dx$$

numerically, with endpoints $a = 0$ and $b = 4$, and compare to the exact result, $I_{\text{exact}} = 2\pi$. In Questions 4a, 4b and 4c below, use instances of a `valarray<long double>` to store the values of the gridpoints x_i, function values $f_i = f(x_i)$ and numerical integration weights w_i.

(a) Use the composite trapezium rule

$$\int_a^b f(x)dx \simeq \sum_{i=0}^{N} w_i f_i = \vec{w} \cdot \vec{f}, \quad \vec{w} = \frac{\Delta x}{2} * \{1, 2, 2, ..., 2, 1\}$$

to compute the integral I, using $N + 1 = 64$ equidistant points in $x \in [a, b]$, that is, $x_i = a + i\Delta x$, for $i = 0, 1, ..., N$, where $\Delta x = \frac{b-a}{N}$ is the grid spacing. Output your numerical result $I_{\text{trapezium}}$ and the difference $I_{\text{trapezium}} - I_{\text{exact}}$. [Hint: Use the function `dot` or `cdot`, created in Question 2a or 2b, to easily evaluate the inner product $\vec{w} \cdot \vec{f}$]. **[5]**

(b) Use the extended Simpson rule

$$\int_a^b f(x)dx \simeq \sum_{i=0}^{N} w_i f_i, \quad \vec{w} = \frac{\Delta x}{48} * \{17, 59, 43, 49, 48, 48, ..., 48, 49, 43, 59, 17\}$$

to compute the integral I, using $N + 1 = 64$ equidistant points in $x \in [a, b]$. Output your numerical result I_{Simpson} and the difference $I_{\text{Simpson}} - I_{\text{exact}}$. **[5]**

(c) Use the Clenshaw-Curtis quadrature rule

$$\int_a^b f(x)dx \simeq \sum_{i=0}^{N} w_i f_i, \quad w_i = \frac{b-a}{2} * \begin{cases} \frac{1}{N^2}, & i = 0 \text{ or } i = N \\ \frac{2}{N}\left(1 - \sum_{k=1}^{(N-1)/2} \frac{2\cos(2k\theta_i)}{4k^2-1}\right) & 1 \le i \le N-1 \end{cases},$$

on a grid of $N + 1 = 64$ points $x_i = [(a + b) + (a - b)\cos\theta_i]/2$, where $\theta_i = i\pi/N$, $i = 0, 1, ..., N$ to compute the integral I.

Hint: First compute the values of θ_i, x_i, $f_i = f(x_i)$ and w_i and store them in `valarray<long double>` containers, as shown in the lab. Then, you may use the function from Question 2a or 2c to compute the inner product $\vec{w} \cdot \vec{f}$.

Output to the screen (and list in your report) your numerical result $I_{\text{ClenshawCurtis}}$ and the difference $I_{\text{ClenshawCurtis}} - I_{\text{exact}}$. **[5]**

(d) Compute the integral I using a Mean Value Monte Carlo method with $N = 10000$ samples. Output to the screen (and list in your report) your numerical result $I_{\text{MonteCarlo}}$ and the difference $I_{\text{MonteCarlo}} - I_{\text{exact}}$. **[5]**

Question 5. [20 marks] *Solitons.*

The Korteweg–De Vries (KDV) equation is a non-linear partial differential equation describing solitons. It can be reduced to a first-order system of ordinary differential equations of the form:

$$\begin{cases} \dfrac{dq}{dt} = p \\ \dfrac{dp}{dt} = -V'(q) \end{cases}$$

with a cubic potential $V(q) = -\big(q^3 + \frac{1}{2}cq^2 + Aq\big)$, initial conditions $q(0) = -\frac{1}{2}c,\ p(0) = 0$, and parameters $A = 0,\ c = 1$.

(a) Use a 2nd-order Runge-Kutta (RK2) method to evolve the system from the initial time $t_{\text{initial}} = 0$ until the final time $t_{\text{final}} = 10$ with $N = 100000$ constant time steps $t_i = t_{\text{initial}} + i\Delta t$ of step size $\Delta t = (t_{\text{final}} - t_{\text{initial}})/N$, where $i = 0, 1, ..., N$. Use `valarray<long double>` containers to store the values t_i, $q(t_i) = q_i$, $p(t_i) = p_i$, $E(t_i) = E_i$ of the time t, position $q(t)$, momentum $p(t)$ and energy $E(t) = \frac{1}{2}p^2 + V(q)$ in each time step, for $i = 0, 1, ..., N$. Describe how your code works. Output to the screen (and tabulate in your report) the values of $q(t)$, $p(t)$, and $E(t)$ for $t = 0,\ t = 1,\ t = 2,\ ..., t = 10$. How well is $E(t)$ conserved numerically? Is this what you expected? Why? Can you explain why the energy is or is not numerically conserved? **[5]**

(b) Compute the difference $e(t) = q_{\text{numerical}}(t) - q_{\text{exact}}(t)$ between your numerical solution $q_{\text{numerical}}(t)$ and the exact solution $q_{\text{exact}}(t) = -\frac{1}{2}c\ \text{sech}^2(\frac{\sqrt{c}}{2}t)$ for all time steps t_i. Output the error values for $t = 0,\ t = 1,\ t = 2,\ ..., t = 10$ to the screen and tabulate them in your report. Comment on whether the error is what you expected. **[5]**

(c) Use the trapezium rule to repeat parts (a) and (b). Comment again on whether energy is conserved numerically and why.
Hint: the trapezium rule results in an implicit system:

$$\begin{cases} q_{i+1} - q_i = \displaystyle\int_{t_i}^{t_{i+1}} p(t)dt \simeq \frac{\Delta t}{2}(p_{i+1} + p_i) \\ p_{i+1} - p_i = -\displaystyle\int_{t_i}^{t_{i+1}} V'(q(t))dt \simeq -\frac{\Delta t}{2}[V'(q_{i+1}) + V'(q_i)] \end{cases}$$

for $i = 0, 1, ..., N-1$. To solve this implicit system for each i, one can perform a self-consistent iteration (similar to Question 1) at each time step, using a nested `for` loop (5-10 self-consistent iterations at each time step would suffice in this case). (That is, for each i, one can use the initial guess $q_{i+1} = q_i$ and $p_{i+1} = p_i$ on the right hand side of the above system, evaluate the left hand side, use the new values q_{i+1} and p_{i+1} and substitute them into the right hand side, compute the left hand side again, and so forth, for 5-10 iterations, then move on to the next time step, as demonstrated in the lab.) **[10]**

End of Paper.