Queens College, CUNY, Department of Computer Science

Numerical Methods

CSCI 361 / 761

Spring 2018

Instructor: Dr. Sateesh Mane

Final Spring 2018

due Thursday May 24, 2018, 11:59 pm

- **NOTE: It is the policy of the Computer Science Department to issue a failing grade to any student who either gives or receives help on any test.**
- This is an **open-book** test.
- **Any problem to which you give two or more (different) answers receives the grade of zero automatically.**
- This is a **take home exam.**
  Please submit your solution via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`. The file name should have either of the formats:

  `StudentId_first_last_CS361_final_May2018`

  `StudentId_first_last_CS761_final_May2018`

  Acceptable file types are txt, doc/docx, pdf (also cpp, with text in comment blocks).
- **In all questions where you are asked to submit programming code, programs which display any of the following behaviors will receive an automatic F:**
  1. Programs which do not compile successfully (compiler warnings which are not fatal are excluded, e.g. use of deprecated features).
  2. Array out of bounds.
  3. Dereferencing of uninitialized variables (including null pointers).
  4. Operations which yield NAN or infinity, e.g. divide by zero, square root of negative number, etc. *Infinite loops.*
  5. Programs which do NOT implement the public interface stated in the question.
- **In addition, note the following:**
  1. Programs which compile and run successfully but have memory leaks will receive a poor grade (but not F).
  2. All debugging and/or output statements (e.g. `cout` or `printf`) will be commented out.
  3. Program performance will be tested solely on function return values and the values of output variable(s) in the function arguments.
  4. In other words, program performance will be tested solely via the public interface presented to the calling application. (I will write the calling application.)

General information

- **You are permitted to copy and use the code in the online lecture notes.**
- **Value of π to machine precision on any computer.**
  1. Some compilers support the constant `M_PI` for π, in which case you can write

     ```
     const double pi = M_PI;
     ```

  2. If your compiler does not support `M_PI`, the value of π can be computed via

     ```
     const double pi = 4.0*atan2(1.0,1.0);
     ```

- **64–bit computers**
  1. The questions in this exam do not involve problems of overflow.
  2. Solutions involving the writing of algorithms will not be judged if they work on a 64–bit instead of a 32–bit computer.

- **If you submit code, put all your code in ONE cpp file.**
  Else include all the code in your main docx or pdf or txt file.
  DO NOT SUBMIT MULTIPLE CPP FILES.

1 Question 1 (submit code)

- You are given input arrays a and b, both of length n and both of type `double`.
- Hence the array elements are `a[0],...,a[n-1]` and `b[0],...,b[n-1]`.
- **Write functions to efficiently calculate the following sums of products:**

$$S_1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a[i]\, b[j]\,. \tag{1.1}$$

$$S_2 = \sum_{i=0}^{n-1} \sum_{j=0}^{i} a[i]\, b[j]\,. \tag{1.2}$$

- **Return** 0.0 **if** $n \leq 0$**.**
- The function signatures are

```
double sum_prod1(int n, const double a[], const double b[]);
double sum_prod2(int n, const double a[], const double b[]);
```

- *Hint: If you do not immediately see the answer, it may help to work out a few simple cases "by hand" for example* $n = 1$*,* $n = 2$*, etc., to see the general pattern.*

2 Question 2 (submit code)

- It was explained in class what is a Grey code and how to create a Grey code.
- You are given an input `nbits`, and define an integer $n = 2^{\text{nbits}}$.
- **Write a function to calculate a Grey code of length n.**
  1. The function signature is

     ```
     void Grey_code(int nbits, int n, int gc[], std::string gcstr[]);
     ```

  2. Here `gc[]` and `gcstr[]` are output arrays both of length n.
  3. The array `gc[]` contains the Grey code integers.
  4. **The value of `gc[i]` should lie in the interval** $0 \leq \texttt{gc[i]} \leq n-1$**.**
  5. The value of `gcstr[i]` contains the binary representation of `gc[i]`.
  6. (Optional, but better if you do it.) The string `gcstr[i]` should be padded so that its length is `nbits`.
- Your function will be tested using the following calling program

```
void Grey_code_test()
{
  for (int nbits = 2; nbits < something; ++nbits) {  // *** I will set the limits ***
    int n = 1 << nbits; // 2^n
    int gc[n];
    std::string gcstr[n];
    Grey_code(nbits, n, gc, gcstr);
    for (int i = 0; i < n; ++i) {
      std::cout << std::setw(10) << gc[i] << std::setw(10) << gcstr[i] << std::endl;
    }
    std::cout << endl;
  }
}
```

- Note the following:
  1. A Grey code is not unique.
  2. **All correct answers will be accepted *(but use nonnegative integers only).***
  3. You are permitted to call `Grey_code(...)` recursively.
  4. You are permitted to write "helper functions" to perform additional tasks and `Grey_code(...)` can call them.
  5. **Solutions which implement a lookup table which contains a Grey code to a large value of n will receive a failing grade.**

3 Question 3 (submit code)

- It was derived in class that the Fourier series of a rectangular function of unit area, centered on $\theta = 0$ and half width θ_0 (so $f(\theta) = 0$ if $|\theta| > \theta_0$) is

$$a_j = \frac{1}{\pi} \frac{\sin(j\theta_0)}{j\theta_0} \qquad (j \geq 1)\,. \tag{3.1}$$

- In addition $a_0 = 1/\pi$ and $b_j = 0$ for all j.
- Let us construct a Fourier series with $a_0 = 1/\pi$ and $b_j = 0$ for all j and

$$a_j = \frac{1}{\pi} \frac{|\sin(j\theta_0)|}{j\theta_0} \qquad (j \geq 1)\,. \tag{3.2}$$

- **We shall use eq. (3.2) for a_j and set $\theta_0 = \pi/8$ in this question.**
- The function $f(\theta)$ is given by the sum of the Fourier series,

$$f_\infty(\theta) = \frac{1}{2\pi} + \frac{1}{\pi} \sum_{j=1}^{\infty} \frac{|\sin(j\theta_0)|}{j\theta_0} \cos(j\theta)\,. \tag{3.3}$$

- Exactly at $\theta = 0$ the function diverges logarithmically

$$f_\infty(0) = \frac{1}{2\pi} + \frac{1}{\pi} \sum_{j=1}^{\infty} \frac{|\sin(j\theta_0)|}{j\theta_0} \to \infty\,. \tag{3.4}$$

- For practical computation, we restrict the sum to a finite number of n terms:

$$f_n(\theta) = \frac{a_0}{2} + \sum_{j=1}^{n-1} a_j \cos(j\theta)\,. \tag{3.5}$$

- We employ a finite sum in eq. (3.5) hence we obtain a finite value for $f_n(0)$.
- **Compute the sum in eq. (3.5) for $f_n(\theta = 0)$ and fill the table below.**

n	$f_n(0)$
10^{16}	2 d.p.
10^{20}	2 d.p.
10^{24}	2 d.p.

- The function $f_\infty(\theta)$ also has weaker logarithmic singularities at $\theta = \pm\pi/4$, $\theta = \pm\pi/2$, $\theta = \pm 3\pi/4$ and $\theta = \pm\pi$.

- **Set $n = 2^{11} = 2048$ in the rest of this question.**
- **Plot a graph of $f_n(\theta)$ for $-\pi \le \theta \le \pi$.**
    1. Define $\theta_i = 2\pi i/n$, where $-n \le i \le n$, so $-\pi \le \theta_i \le \pi$.
    2. Compute the sum in eq. (3.5) at $\theta = \theta_i$ only.
    3. Plot the value of θ_i/π on the horizontal axis, so the numbers go from -1 to 1.
- **Calculate the area "under" the curve. State your answer to 3 decimal places.**
    1. The integral is given as follows:
$$A_n = \int_{-\pi}^{\pi} f_n(\theta)\, d\theta\,. \tag{3.6}$$
    2. State your choice of integration method and any other relevant details.
    3. The integral in eq. (3.6) is an improper integral (but we know where the function $f_\infty(\theta)$ diverges).
    4. Explain how you will deal with the improper integral.
- The function $f_n(\theta)$ crosses zero once at $\theta = \pm\theta_r$, where $0 < \theta_r < \pi$.
- **Employ a root finding algorithm to determine the value of θ_r to an accuracy of 4 decimal places.**
    1. State your choice of algorithm.
    2. Also state the value of your starting iterate(s).
    3. **Justify your choice** for your starting iterate(s).
    4. Display a table of your iterates to show convergence to 4 decimal places.
- **Calculate the area under the curve for $0 < \theta < \theta_r$, say A_r.**
    1. The integral is given by
$$A_r = \int_{0}^{\theta_r} f_n(\theta)\, d\theta\,. \tag{3.7}$$
    2. *This calculation requires some precautions.*
    3. It is an improper integral because $f_\infty(0) = \infty$ hence the function diverges at one endpoint.
    4. Although the value of $f_n(0)$ is finite, that is cheating.
    5. We must employ the midpoint rule, which does not evaluate the function at the endpoints of the interval.
    6. Now we encounter some confusion about the meaning of n.
    7. We have used n to denote the number of Fourier harmonics in the sum in eq. (3.5).
    8. Let the number of subintervals for the midpoint rule be n_{mid}.
    9. **Calculate the value of the integral in eq. (3.7) using $n_{\text{mid}} = 1000$ subintervals.**
    10. **State your answer to 3 decimal places.**

4 Question 4 (submit code)

- A **parametric curve** is a curve (x, y) where $x(t)$ and $y(t)$ are functions of a parameter t.
- We shall employ the following parametric curve:

$$\begin{aligned} x(t) &= (1 - s\rho)\cos(t) + s\rho\,\cos\Big(\frac{1-\rho}{\rho}\,t\Big)\,, \\ y(t) &= (1 - s\rho)\sin(t) - s\rho\,\sin\Big(\frac{1-\rho}{\rho}\,t\Big)\,. \end{aligned} \tag{4.1}$$

- We shall also require the following derivatives later in this question.

$$\begin{aligned} \frac{dx}{dt} &= -(1 - s\rho)\sin(t) - s(1-\rho)\,\sin\Big(\frac{1-\rho}{\rho}\,t\Big)\,, \\ \frac{dy}{dt} &= \quad (1 - s\rho)\cos(t) - s(1-\rho)\,\cos\Big(\frac{1-\rho}{\rho}\,t\Big)\,. \end{aligned} \tag{4.2}$$

- **Set the parameter values $\rho = 1/6$ and $s = 5/4 = 1.25$ in this question.**
- **Warning: the parameter t is not the same as the polar angle θ.**
  1. In polar coordinates (r, θ), $x = r\,\cos\theta$ and $y = r\,\sin\theta$ and $\theta = \arctan(y/x)$.
  2. It is obvious from eq. (4.1) that $t \neq \arctan(y/x)$, hence $\theta \neq t$ in general.
  3. However, it *is* true that $\theta = 0$ when $t = 0$ and $\theta = 2\pi$ when $t = 2\pi$.
  4. Hence the interval $0 \leq t \leq 2\pi$ yields a closed curve in the (x, y) plane.
- A graph of the curve is plotted in Fig. 1, for $0 \leq t \leq 2\pi$.
- **Calculate the area enclosed in the central portion of the curve.**
- That is to say, the area excluding the lobes.
- *Let's not panic.*
- First find the value of t such that the curve crosses the x-axis at the point marked α in Fig. 1.
- Call this value t_α. It is the smallest value $t > 0$ such that $y(t_\alpha) = 0$.
- **Employ a root finding algorithm to find t_α, which is the smallest value $t > 0$ such that $y(t_\alpha) = 0$.**
  1. State your choice of algorithm.
  2. Also state the value of your starting iterate(s).
  3. **Justify your choice** for your starting iterate(s).
  4. Iterate until the value of t_α converges to 4 decimal places.
  5. **State the value of t_α.**
  6. Display a table of your iterates to show convergence to 4 decimal places.

- **Find the second value of t where the curve self-intersects. Call the value t_β.**
- This is the point marked β in Fig. 1.
- The area enclosed by the curve, from t_α to t_β is given by

$$A(t_\alpha, t_\beta) = \int_{t_\alpha}^{t_\beta} \left(x \frac{dy}{dt} - y \frac{dx}{dt} \right) dt \,. \tag{4.3}$$

- By the six-fold symmetry, the total area of the central enclosed region is $A_{\text{tot}} = 6A(t_\alpha, t_\beta)$.
  1. **Calculate the value of A_{tot} using $n = 1000$ subintervals.**
  2. Use (i) trapezoid rule, (ii) midpoint rule, (iii) Simpson's rule.
  3. State your answers to 4 decimal places.
- Let us perform some algebra and trigonometry to process the integrand in eq. (4.3).

$$x \frac{dy}{dt} - y \frac{dx}{dt} = (1 - s\rho)^2 - s^2\rho(1-\rho) + s(2\rho - 1)(1 - s\rho) \cos\left(\frac{t}{\rho}\right). \tag{4.4}$$

- Then the area is

$$A_{\text{tot}} = 6\big[\,(1 - s\rho)^2 - s^2\rho(1-\rho)\,\big](t_\beta - t_\alpha) + 6s\rho(2\rho - 1)(1 - s\rho) \left[\sin\left(\frac{t_\beta}{\rho}\right) - \sin\left(\frac{t_\alpha}{\rho}\right) \right]. \tag{4.5}$$

- **Calculate the value of A_{tot} using eq. (4.5).**
- State your answer to 4 decimal places.
- This will give an estimate of the accuracy of the numerical integration.
- Of course, the value from eq. (4.5) is not exact because the values of t_α and t_β are not known exactly.

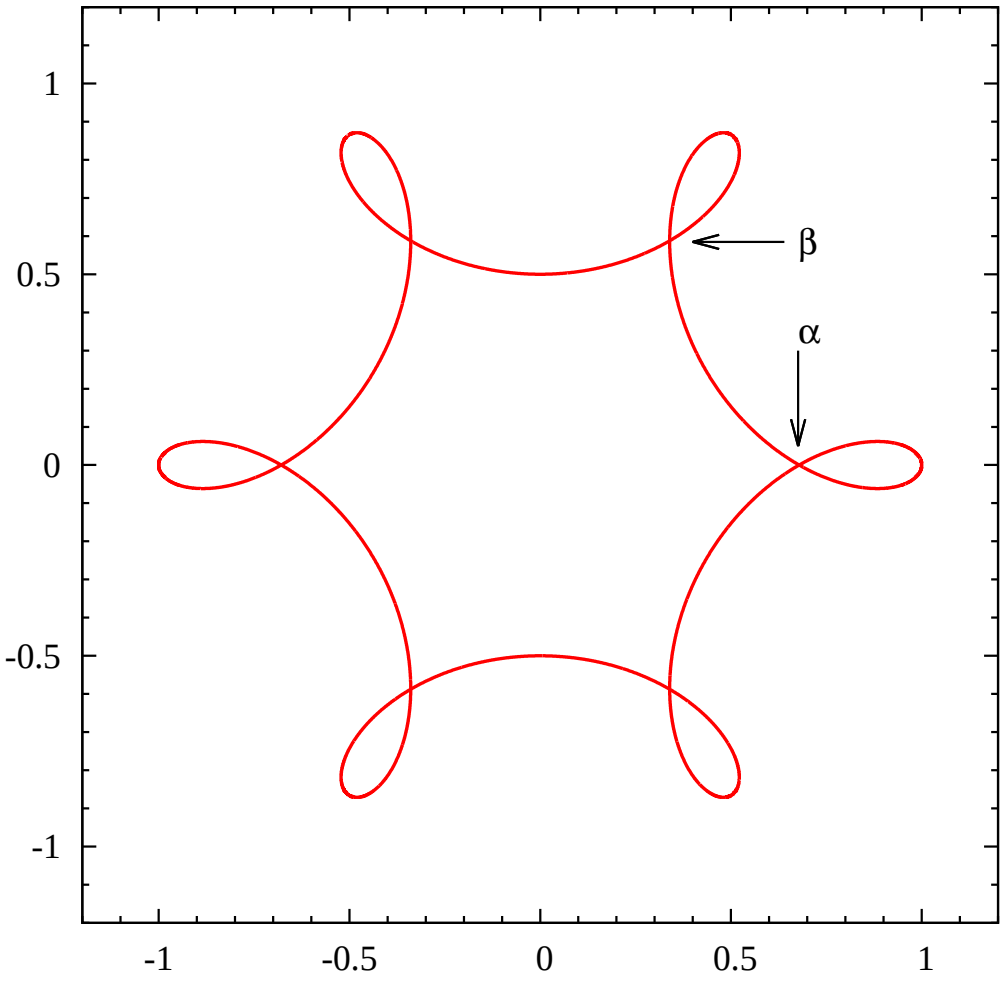

Figure 1: Graph of the curve in Question 4.

Lissajous figure (non-question)

- **There is no Lissajous figure.**
- I substituted the curve in Question 4 instead of a Lissajous figure (prettier).
- The curve in Question 4 is an example of a **hypotrochoid.**
- The special case $\rho = \frac{1}{2}$ is an ellipse.
- The special case $s = 1$ is a **hypocycloid.**
- For $s = 1$ and $\rho = 1/3$ the curve is called a **deltoid.**
- For $s = 1$ and $\rho = 1/4$ the curve is called an **astroid.**
- For $s > 1$ the curves develop lobes and self-intersect, as in Fig. 1.
- *No, there's nothing for you to calculate here.*
- Just look at the pretty pictures and learn their names.

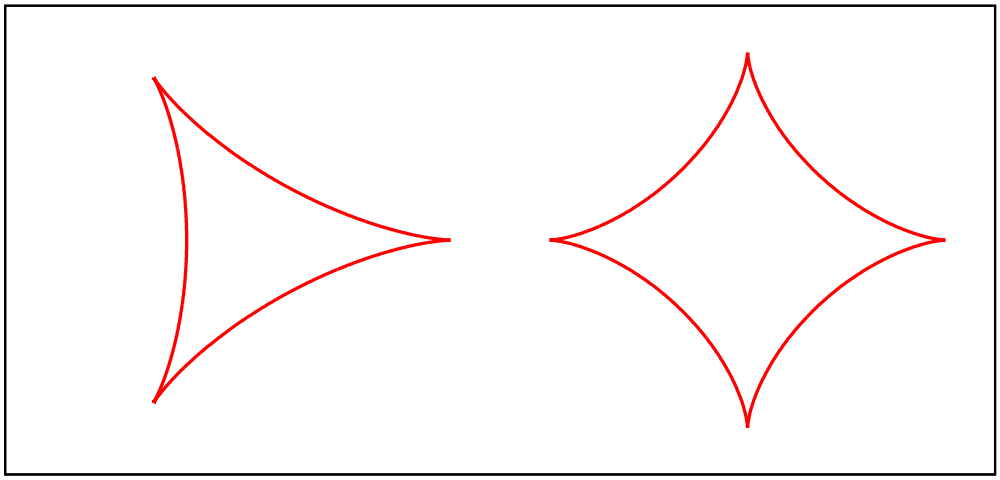

Figure 2: Plots of deltoid and astroid.

5 Question 5 (submit code)

- You are given the following first order ordinary differential equation:

$$\frac{dA}{dx} = 4\left(1 - |x|^{1.7}\right)^{1/3.2}. \tag{5.1}$$

- The initial condition is $A(1) = 0$ at $x = 1$.
- **Use Runge–Kutta fourth order `RK4` to integrate eq. (5.1) from $x = 1$ down to $x = 0$.**
- Submit your code as part of your solution.
- Use $n = 1000$ and a stepsize $h = -1/n = -1/1000$.
- Fill in the table below as you proceed with the integration.

x	$A(x)$
1	0
0.75	4 d.p.
0.5	4 d.p.
0.25	4 d.p.
0	4 d.p.

- **This question was directly inspired by a student solution in Fall 2017.**
- Who says we need Simpson's ruke, etc., to calculate the area of a superellipse?
- *Runge-Kutta will do the job very nicely and will moreover yield intermediate results for the areas of slices of the superellipse.*
- I have respect for students who think "outside the box" and have the imagination to use their knowledge in novel ways.

6 Question 6 (submit code)

- You are given the following inhomogeneous ordinary differential equation:

$$\frac{d^2y}{dx^2} + y = \frac{\cos(x)}{\pi}\,. \tag{6.1}$$

- **Write down the homogeneous ordinary differential equation associated with eq. (6.1).**
- **Show that $y_1(x)$ and $y_2(x)$ are solutions of the associated homogeneous ordinary differential equation, where**

$$y_1(x) = \sin(x)\,, \qquad y_2(x) = \cos(x)\,. \tag{6.2}$$

- **Show that $P(x)$ is a particular solution of eq. (6.1), where**

$$P(x) = \frac{x\sin(x)}{2\pi}\,. \tag{6.3}$$

- **Write down the general solution of eq. (6.1).**
- **Find the solution of eq. (6.1) which satisfies the following initial conditions at $x = 0$:**

$$y(0) = 0\,, \qquad \left[\frac{dy}{dx}\right]_{x=0} = 0\,. \tag{6.4}$$

- **Using the initial conditions in eq. (6.4), calculate the value of $y(x)$ at $x = 5\pi/2$.**
- Denote this value by y_* below, i.e. $y_* = y(5\pi/2)$.
- **Employ an auxiliary variable $v = dy/dx$ to express eq. (6.1) as a pair of coupled first order ordinary differential equations.**
- You are required to integrate the coupled equations numerically from $x_0 = 0$ forwards to $x_n = 5\pi/2$ in n steps and then backwards to $x_{2n} = 0$ also in n steps.
- Hence the stepsize is $h = 2\pi/n$.
- Let the value of y at $x = x_n = 5\pi/2$ be y_n and let the value of y at $x = x_{2n} = 0$ be y_{2n}.
- **Use Runge–Kutta fourth order `RK4` to integrate eq. (6.1) with the initial conditions in eq. (6.4). Find a value of n such that the following inequalities are satisfied:**

$$10^{-5} \le |y_n - y_*| \le 10^{-4}\,. \tag{6.5}$$

- ***Note the lower bound.*** The value of n should not be too small but it should also not be too large.
- **Next use `RK4` to integrate eq. (6.1) backwards to $x_{2n} = 0$.**
- **State the value of y_{2n} that you obtain.**

- **Submit your code for the integration as part of your solution (including your main program).**
- **You will be graded on your code as well as on your numerical results.**
- **You are permitted to copy code or code fragments from the online lecture notes.**
- **(Optional) Plot a graph of your numerical solution of eq.** (6.1)**.**

7 Question 7 (submit code)

- The Bessel function $J_\nu(x)$ satisfies the following linear ordinary differential equation:

$$x^2 \frac{d^2y}{dx^2} + x \frac{dy}{dx} + (x^2 - \nu^2)y = 0\,. \tag{7.1}$$

- Here ν can be any number, even complex. **We shall use $\nu = 1.5$ in this question.**
- Define a set of $n+1$ equally spaced points with x_i via $x_0 = 0$ and $x_n = 20$.
- Hence the interval size we shall employ in this question is $h = 20/n$.
- **Express $y'(x_i)$ and $y''(x_i)$ using centered finite differences, for $i = 1, \ldots, n-1$.**
- We shall deal with the special cases $i = 0$ and $i = n$ later.
- **Substitute your expressions for the centered finite differences into eq. (7.1).**
- **Multiply through by h^2 to obtain an equation of the following form:**

$$a_i y_i + b_i y_{i-1} + c_i y_{i+1} = 0 \qquad (i = 1, \ldots, n-1)\,. \tag{7.2}$$

- **State your expressions for a_i, b_i and c_i, for $i = 1, \ldots, n-1$.**
- For the special cases, set $a_0 = 1$ **and** $c_0 = 0$ for $i = 0$ and $a_n = 1$ **and** $b_n = 0$ for $i = n$.
- You are given that $J_{1.5}(0) = 0$ and $J_{1.5}(20) \simeq -0.064663$. The above equations will satisfy the boundary conditions $y_0 = 0$ and $y_n = -0.064663$.
- If you have done everything correctly, you should obtain a tridiagonal set of equations of the following form. (Blanks denote zeroes in the tridiagonal matrix below.)

$$\begin{pmatrix} 1 & 0 & & & & & \\ b_1 & a_1 & c_1 & & & & \\ & b_2 & a_2 & c_2 & & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & b_{n-1} & a_{n-1} & c_{n-1} \\ & & & & & 0 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ \mathbf{-0.064663} \end{pmatrix}. \tag{7.3}$$

- *The matrix in eq.* (7.3) *is NOT diagonally dominant. Do not worry.*
- **Solve eq. (7.3) using $n = 20000$.**
- Note that $J_{1.5}(0) = 0$. In addition, $J_{1.5}(x)$ oscillates and equals zero multiple times in the interval $0 < x < 20$.
- **Use your solution of the tridiagonal equations to determine where $J_{1.5}(x)$ equals zero in the interval $0 < x < 20$. State your answers to a tolerance of ± 0.001.**
- **(Optional) Plot a graph of $J_{1.5}(x)$ for $0 \le x \le 20$.**

8 Question 8 (submit code (your function only))

- This is the question I promised near the beginning of the semester.
- This is adapted from a real research problem in my career.
- The device I had to design was called an *electrostatic quadrupole*.
- *Read the question carefully and don't panic.*
- A cross-section of the electrostatic quadrupole is shown in Fig. 3.
    1. There is an electric potential inside the quadrupole, call it V (for 'voltage').
    2. The electric potential is zero on the outer boundary (see Fig. 3).
    3. There are four electrodes, where the potential is set to fixed values.
    4. The electric potential is $V = 100$ on the left and right electrodes and $V = -100$ on the top and bottom electrodes (see Fig. 3).
    5. I solve the partial differential equation to calculate the value of the potential at the other points in the volume.
    6. *I perform the mathematical computation for you.*
- We measure the electric potential $V(\theta)$ on the dashed circle (see Fig. 3).
- We expand $V(\theta)$ in a Fourier series.
- Because of the four-fold symmetry, and also because the voltage on the electrodes form a $(+-+-)$ pattern, the only nonzero Fourier harmonics in $V(\theta)$ are

$$V(\theta) = a_2 \cos(2\theta) + a_6 \cos(6\theta) + a_{10} \cos(10\theta) + \cdots \tag{8.1}$$

- **The values of the Fourier coefficients a_2, a_6, a_{10}, etc., depend on the width w of the electrodes.**
- **Your task is to find the value of the width w such that $a_6 = 0$.**
- The procedure to do so is given below.
- Compile the supplied C++ code for the `class Node` and `class ElectrostaticQuadrupole`.
- Define an integer `digit = 1` if the first digit of your student id is odd and `digit = 2` if the first digit of your student id is even.
    1. For example if your student id is 12345678 or 34562345, then `digit = 1`.
    2. For example if your student id is 24681357 or 45623451, then `digit = 2`.
    3. Also set $n = 2^{12}$ to compute the values of $V(\theta)$ around the circle.

- Implement the code below. Use `digit` as an argument to the class constructor.

```
const int ngrid = 240;
const int n = 1 << 12;            // power of 2
std::vector<double> V(n, 0.0);

const int digit = 1;          // depends on student id
//const int digit = 2;        // depends on student id
ElectrostaticQuadrupole equad(ngrid, digit);
```

- The parameter n_{grid} sets the size of the grid inside the electrostatic quadrupole object, to solve a partial differential equation. In this question we hard code $n_{\text{grid}} = 240$. Do not change this value.

- **Now we iterate the value of w.**

  1. Choose a value for w.
  2. *The value of w must lie in the range $0 < w < 10$. The code in the class* `ElectrostaticQuadrupole` *contain validation checks to reject values of w outside this interval.*
  3. Run the code below.

```
double w = 4.0;         // choose some value for w
equad.SOR(w);
equad.circle(n, V);
```

  4. The class method "`SOR`" solves the partial differential equation.
  5. On my computer, one functon call requires about 8 sec.
  6. The running time will vary depending on your computer.
  7. The class method "`circle`" cmputes the value of V at n ppints around the circle.
  8. The output value of V is $_j = V(\theta_j)$, where $\theta = j = 2\pi j/n$, for $j = 0, \ldots, n-1$.

- So now you have an array (a `std::vector`) V with entries $V(\theta_j)$.

- **Compute the coefficients a_j and b_j of the Fourier series for V.**

- *If you have done your cork correctly, the only nonzero values will be $a_2, a_6, a_{10}, \ldots$. All the other values will be zero or very small.*

- Choose another value for w and calculate the values of the coefficients a_j and b_j again.

- **Iterate the value of w until you obtain $|a_6/a_2| < 10^{-3}$.**
- Your overall code structure should look like this:

```
const int ngrid = 240;
const int n = 1 << 12;            // power of 2
std::vector<double> V(n, 0.0);

const int digit = 1;          // depends on student id
//const int digit = 2;        // depends on student id
ElectrostaticQuadrupole eq(ngrid, digit);

// *** begin iteration for value of w ***
w = something
equad.SOR(w);
equad.circle(n, V);

// *** compute Fourier coefficients of V
// *** test if abs(a[6]/a[2]) < 0.001
// *** repeat interation until convergence
```

- Also tabulate the value of a_{10}/a_2.
- Fill the table below with your iterated values.

	w	a_6/a_2	a_{10}/a_2
0	4 d.p.	4 dp.p	4 d.p.
1	4 d.p.	4 dp.p	4 d.p.
2	4 d.p.	4 dp.p	4 d.p.
$\vdots$	4 d.p.	4 dp.p	4 d.p.

- **Some personal comments, for your information.**
- In the ideal design, we want only $a_2 \neq 0$ and all others zero.
- To achieve this, the shape of the requires electrodes must be hyperbolas.
- A hyperbolic shape is difficult to manufacture and moreover extends to $\pm\infty$,
- A real quadrupole electrode cannot extend to $\pm\infty$.
- Flat quadrupole electrodes are simple to manufacture.
- The design is robust and easy to install and maintain.
- By setting the width of the electrodes suitably, the leading error term (the unwanted a_6 Fourier harmonic) was eliminated.
- The Fourier harmonic a_{10} was also unwanted but not eliminated. However the magnitude of a_{10}/a_2 was small and negligible.
- The full expression for the potential in polar coordinates is

$$V(r,\theta) = a_2 r^2 \cos(2\theta) + a_6 r^6 \cos(6\theta) + a_{10} r^{10} \cos(10\theta) + \dots \tag{8.2}$$

- With elimination of the a_6 term, the value of the potential is

$$V(r,\theta) = a_2 r^2 \cos(2\theta) + a_{10} r^{10} \cos(10\theta) + \dots \tag{8.3}$$

- In the aboves question, the potential was calculated on the circle $r = 1$.
- For $r < 1$, the value of r^{10} is very small, hence the contribution from the a_{10} term is negligible.
- The above quadrupole design greatly simplified construction and maintenance and saved a lot of cost.

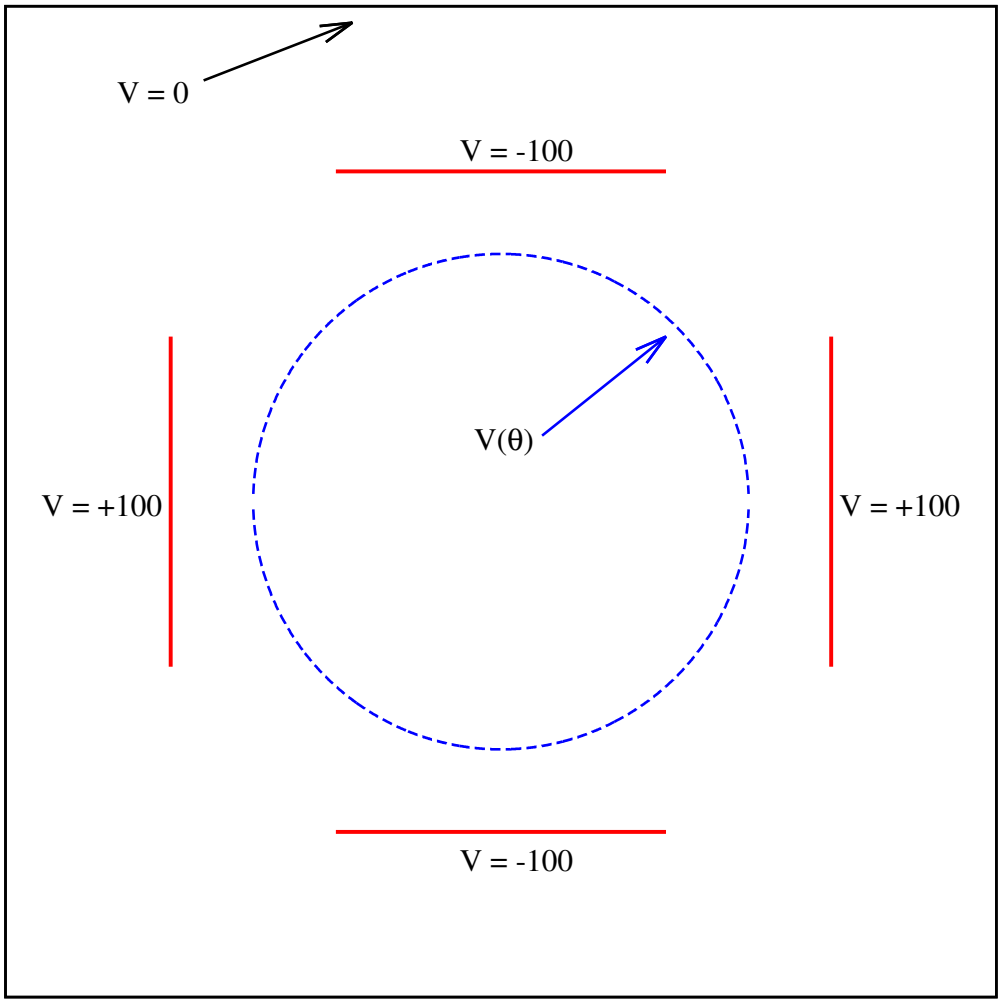

Figure 3: Cross-section of electrostatic quadrupole in Question 8.

Background information (not for examination)

- The electric potential satisfies Laplace's equation, which in two dimensions is

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0\,. \tag{8.4}$$

- We express the partial derivatives as numerical differenes on a rectangular grid with stepsizes (h, k) along x and y.

- Then we obtain, with indices (i, j) for x and y,

$$\frac{V_{i+1,j} + V_{i-1,j} - 2V_{ij}}{h^2} + \frac{V_{i,j+1} + V_{i,j-1} - 2V_{ij}}{k^2} = 0\,. \tag{8.5}$$

- It is simplest to employ a square grid. Hence we set $h = k$, which yields

$$\frac{V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} - 4V_{ij}}{h^2} = 0\,. \tag{8.6}$$

- This leads to the simple equation

$$V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} - 4V_{ij} = 0\,. \tag{8.7}$$

- We solve eq. (8.7) using the boundary conditions $V_{ij} = 0$ on the outer edges of the box and $V_{ij} = +100$ on the left and right electrodes and $V_{ij} = -100$ on the top and bottom electrodes. See Fig. 3.

- We solve eq. (8.7) as follows. Define a 'time' parameter t and define

$$\Delta V_{ij}^{(t)} = \left(V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} - 4V_{ij}\right)^{(t)}\,. \tag{8.8}$$

- Then iterate for $t = 0, 1, \ldots$ via

$$V_{ij}^{(t+1)} = V_{ij}^{(t)} + \Delta V_{ij}^{(t)}\,. \tag{8.9}$$

- This technique is called **Gauss–Seidel iteration.**

- We iterate until $|\Delta V_{ij}^{(t)}| < \texttt{tol}$ at all points (i, j).

- This is a much more restrictive tolerance criterion than integration in one dimension.

- *Let's be greedy.* Let us try to 'jump ahead' to the final answer by introducing a parameter ω

$$V_{ij}^{(t+1)} = V_{ij}^{(t)} + \omega\,\Delta V_{ij}^{(t)}\,. \tag{8.10}$$

- We set $\omega > 1$ to try to 'take a bigger increment' to reach the solution more quickly.

- Actually, if $\Delta V_{ij}^{(t)}$ alternates in sign, it might be better to set $\omega < 1$.

- The technique in eq. (8.10) is called **successive overrelaxation (SOR).**
- If $0 < \omega < 1$ some authors called it *underrelaxation.*
- SOR is stable only if $0 < \omega < 2$.
- If there are totally n^2 points in the grid, each update of the grid requires n^2 computations.
- In addition, the Gauss-Seidel method requires $O(n^2)$ iterations in the time step t.
- SOR can reduce the number of iterations in t to $O(n)$.
- What is the optimal value of ω to employ?
    1. Nobody knows.
    2. There are results for special cases, but no general formula to determine the optimum.
    3. I set $\omega = 1.95$ in the code.
    4. I found that using $\omega = 1$ (Gauss-Seidel) required about 8 seconds for convergence.
    5. I found that using $\omega = 1.95$ (SOR) required about 0.35 seconds for convergence.
    6. I obtained the following approximate table of running times to obtain convergence.

ω	time
1.0	8.0
1.5	2.0
1.9	0.49
1.94	0.47
1.95	0.35
1.96	0.41
1.98	0.65

- You understand why the calculation function is called `SOR`.

```
#include <vector>
#include <cmath>
#include <complex>

class Node
{
public:
  Node() { x = 0; y = 0; V = 0; bdy = false; }
  double x;
  double y;
  double V;
  bool bdy;
};

class ElectrostaticQuadrupole
{
public:
  ElectrostaticQuadrupole(int n, int k) : _halfedge(5.0) {
    _n = n;
    _off = -k;
    allocate();
    init();
  }

  ~ElectrostaticQuadrupole() { clear(); }

private:
  void allocate() {
    _nodes = new Node*[_n+1];
    for (int i = 0; i <= _n; ++i) {
      _nodes[i] = new Node[_n+1];
    }
  }

  void clear() {
    for (int i = 0; i <= _n; ++i) {
      delete [] _nodes[i];
    }
    delete [] _nodes;
    _n = 0;
  }

  void init() {
    for (int i = 0; i <= _n; ++i) {
      bool bx = ((i==0) || (i==_n)) ? true : false;
```

```
    double x = -_halfedge + 2.0*_halfedge*double(i)/double(_n);
    Node * nx = _nodes[i];
    for (int j = 0; j <= _n; ++j) {
      bool by = ((j==0) || (j==_n)) ? true : false;
      double y = -_halfedge + 2.0*_halfedge*double(j)/double(_n);
      Node & nxy = nx[j];
      nxy.x = x;
      nxy.y = y;
      nxy.V = 0;
      nxy.bdy = (bx && by);
    }
  }
}

void set(double a) {
  for (int i = 0; i <= _n; ++i) {
    //double x = -_halfedge + 2.0*_halfedge*double(i)/double(_n);
    Node * nx = _nodes[i];
    Node & bottom = nx[0];
    Node & top = nx[_n];
    double &x = top.x;
    top.V = 0.0;
    bottom.V = 0.0;
    top.bdy = true;
    bottom.bdy = true;
  }

  for (int i = 0; i <= _n; ++i) {
    Node * nx = _nodes[i];
    int j = _n/20 + _off;
    Node & bottom = nx[j];
    Node & top = nx[_n-j];
    double &x = top.x;
    if (std::abs(x) <= 0.5*a) {
      top.V = -100.0;
      bottom.V = -100.0;
      top.bdy = true;
      bottom.bdy = true;
    }
  }

  for (int j = 0; j <= _n; ++j) {
    //double y = -_halfedge + 2.0*_halfedge*double(j)/double(_n);
    Node * left = _nodes[0];
    Node * right = _nodes[_n];
```

```
      Node & lj = left[j];
      Node & rj = right[j];
      double &y = lj.y;
      lj.V = 0.0;
      rj.V = 0.0;
      lj.bdy = true;
      rj.bdy = true;
    }

    for (int j = 0; j <= _n; ++j) {
      //double y = -_halfedge + 2.0*_halfedge*double(j)/double(_n);
      int i = _n/20 + _off;
      Node * left = _nodes[i];
      Node * right = _nodes[_n-i];
      Node & lj = left[j];
      Node & rj = right[j];
      double &y = lj.y;
      if (std::abs(y) <= 0.5*a) {
        lj.V = 100.0;
        rj.V = 100.0;
        lj.bdy = true;
        rj.bdy = true;
      }
    }

    for (int i = 1; i < _n; ++i) {
      Node * nx = _nodes[i];
      for (int j = 1; j < _n; ++j) {
        Node & nxy = nx[j];
        if (nxy.bdy == false) {
  nxy.V = 0;
        }
      }
    }
  }

public:
  int SOR(double a) {
    if ((a <= 0.0) || (a >= 2.0*_halfedge)) {
      return 1; // fail
    }
    set(a);
    const double tol = 1.0e-8;
    const double omega = 1.95;
    const int max_iter = 10000;
```

```
for (int iter = 0; iter < max_iter; ++iter) {
  double norm = 0;
  // chessboard
  // sweep i+j = even
  for (int i = 1; i < _n; i++) {
    Node * nmid = _nodes[i];
    Node * nleft = _nodes[i-1];
    Node * nright = _nodes[i+1];
    for (int j = 1; j < _n; j++) {
      if ((i+j)%2 == 1) continue;
      Node & n0 = nmid[j];
      if (n0.bdy == true) continue;

      Node & n1 = nright[j];
      Node & n2 = nmid[j+1];
      Node & n3 = nleft[j];
      Node & n4 = nmid[j-1];
      double delta_V = 0.25*(n1.V +n2.V +n3.V +n4.V) - n0.V;
      if (norm < delta_V*delta_V) {
        norm = delta_V*delta_V;
      }
      n0.V += omega*delta_V;
    }
  }
  // sweep i+j = odd
  for (int i = 1; i < _n; i++) {
    Node * nmid = _nodes[i];
    Node * nleft = _nodes[i-1];
    Node * nright = _nodes[i+1];
    for (int j = 1; j < _n; j++) {
      if ((i+j)%2 == 0) continue;
      Node & n0 = nmid[j];
      if (n0.bdy == true) continue;

      Node & n1 = nright[j];
      Node & n2 = nmid[j+1];
      Node & n3 = nleft[j];
      Node & n4 = nmid[j-1];
      double delta_V = 0.25*(n1.V +n2.V +n3.V +n4.V) - n0.V;
      if (norm < delta_V*delta_V) {
        norm = delta_V*delta_V;
      }
      n0.V += omega*delta_V;
    }
  }
```

```
    if (norm <= tol) break;
  }
  return 0; // success
}

void circle(int nc, std::vector<double> &V) const {
  const double pi = 4.0*atan2(1.0,1.0);
  const double r = _halfedge * 0.8;
  V.resize(nc, 0.0);

  for (int ic = 0; ic < nc; ++ic) {
    double theta = 2.0*pi*double(ic)/double(nc);
    double x0 = r*cos(theta);
    double y0 = r*sin(theta);
    int i0 = _n * (x0 / _halfedge + 1.0)/2.0;
    int j0 = _n * (y0 / _halfedge + 1.0)/2.0;
    bool interp = false;
    for (int i = i0-2; i <= i0+2; ++i) {
      double x1 = -_halfedge + 2.0*_halfedge*double(i)/double(_n);
      double x2 = x1 + 2.0*_halfedge/double(_n);
      if ((x1 > x0) || (x2 < x0)) continue;
      double lambda = (x0 - x1)/(x2 - x1);
      Node * nx1 = _nodes[i];
      Node * nx2 = _nodes[i+1];
      for (int j = j0-2; j <=j0+2; ++j) {
        double y1 = -_halfedge + 2.0*_halfedge*double(j)/double(_n);
        double y2 = y1 + 2.0*_halfedge/double(_n);
        if ((y1 > y0) || (y2 < y0)) continue;
        double mu = (y0 - y1)/(y2 - y1);
        Node & nx1y1 = nx1[j];
        Node & nx1y2 = nx1[j+1];
        Node & nx2y1 = nx2[j];
        Node & nx2y2 = nx2[j+1];
        double V_interp = (1.0-lambda)*(1.0-mu)*nx1y1.V
                          +lambda*(1.0-mu)*nx2y1.V
                          +lambda*mu*nx2y2.V
                          +(1.0-lambda)*mu*nx1y2.V;
        V[ic] = V_interp;
        interp = true;
        break;
      }
      if (interp == true) break;
    }
  }
}
```

```
private:
  // data
  int _n;
  int _off;
  Node **_nodes;
  const double _halfedge;
};
```