Numerical Methods in Physics and Astrophysics (2024-25)

Kostas Kokkotas

November 12, 2024

Contents

1 Non-Linear Equations & Systems **2**
- 1.1 Root Finding: Bisection, Linear Interpolation, $x = g(x)$ and Newton 3
- 1.2 Nonlinear Systems of Equations 4
- 1.3 Fractals through Newton-Raphson 4

2 Linear Systems & Matrices **6**

3 Interpolation & Extrapolation **10**
- 3.1 Interpolation Example – Equation of State (*) 11

4 Numerical Differentiation **14**
- 4.1 Numerical Differentiation of Functions 15

5 Numerical Integration **16**

1 | Non-Linear Equations & Systems

1

[1] This set of problems for the course developed over the last 12 years with the help of my collaborators:

- Dr. Wolfgang Kastaun (2007-9)
- Dr. Buchart Zink (2010-13)
- Dr. Tanja Bode (2013-15)
- Dr. Daniela Doneva (2015-17)
- Dr. Praveen Manoharan (2021-22)

together with many valuable comments and suggestions from my ex-students.

1.1 | Root Finding: Bisection, Linear Interpolation, $x = g(x)$ and Newton

Deadline: 29.10.2024

Root finding is ubiquitous in physics codes. Here we pursue and experiment with methods for root finding from the most basic to the most commonly used. Pay attention to convergence and corner cases throughout.

PROBLEM I

We will consider specifically the following function:

$$f(x) = e^{\sqrt{5}x} - 13.5\cos(0.1x) + 25x^4 \tag{1.1}$$

1. Find this root again, using **linear interpolation**.
2. Implement **Newton's method**.
3. Examine the convergence of the three methods.
4. Explore the robustness of the methods with varying initial guesses and tolerances. What happens when you specify a tolerance below machine precision?

Tips, Hints, and Suggestions:

- Start by defining a function that calculates $f(x)$
- Code design is up to you, but I suggest *one* of the following:
  - Write a function which calls the desired solving method with the initial guess or range and the required accuracy.
  - Use a shared header file to coordinate information among several independent programs
- Avoid getting stuck in infinite loops: cap iteration numbers and check if one stops moving towards a solution.
- Use print statements to debug your code!

PROBLEM II

Find the eigenvalues λ of the differential equation $y'' + \lambda^2 y = 0$ with the following boundary conditions $y(0) = 0$ and $y(1) = y'(1)$.

Tips, Hints, and Suggestions:

As first step you should try to solve the problem analyticaly by considering the general solution of this wave equation.
Then use the boundary conditions to simplify the solution and the problem will be reduced in finding the roots of a non-linear equation.

1.2 | Nonlinear Systems of Equations

We first consider the following nonlinear system of equations

$$f_1(x, y) = xy - 0.1 \tag{1.2}$$
$$f_2(x, y) = x^2 + 3y^2 - 2 \tag{1.3}$$

A1. Solve the nonlinear system of equations using the generalized Newton method. Make sure you find all solutions (there are 4!). Under what conditions would the solver create problems? What checks can you put on/in your solver to avoid diverging from the solution or continuing through with NaNs?

A2. Reformulate the equations and add a function for the "improved" $x = g(x)$ method to the above program. What are the requirements for convergence? How does this affect solving the above system of equations? How is the relative performance for the various roots? How does the program behave if you guess $(x, y) = (2, 1)$?

1.3 | Fractals through Newton-Raphson

Deadline: 5.11.2024

Fractals and chaos theory are intricately entwined. In a nutshell, chaos theory amounts to the sensitive, diverging nature of a solution with small changes in the initial conditions. We will go through another chaos problem later in these exercises, but here we delve deeper into the fractal nature of root-finding via the Newton-Raphson method. For these solutions, as 1-dimensional fractals are not as pretty, we will look at solutions to simple functions in the complex plane. The iteration scheme in Newton-Raphson here is the same as for functions in real space:

$$z_{n+1} = z_n - \frac{f(z)}{f'(z)}$$

For each function $f(z)$ we consider in this lab, we create a map of convergence in the (x_0, y_0)-plane where $z_0 = x_0 + iy_0$ is the initial guess to our solver. For which z_0 does the method converge? How fast does it converge? To which root does it converge?

1. Consider first the solution of the function

$$f(z) = z^3 - 1 = 0 \tag{1.4}$$

- Learn how to handle complex numbers in your language of choice.
- Write a function `solve_cnewton` which takes an initial guess and returns the root (if any), the number of iterations need to converge, and the quality of the solution $f(z)$. This should be as simple as grabbing your old newton solver from Lab 1 and upgrading to use complex numbers and interface with the function `get_fdf`.
- Write a program that takes a square $N \times N$ grid in the complex plane and, for each point, uses the above functions to find a root to which the method converges. For a convergence criteria, look for when the change in z is less than ϵ with a certain number of maximum steps M. That is when within M steps $|z_{n+1} - z_n| < \epsilon$. Start with $M = 200$, $\epsilon = 10^{-9}$, and $N = 400$. Define the range of the grid by two gridpoint coordinates (e.g. upper left and lower right). If you know how, take these points as arguments to the main function to ease exploring a given function; otherwise hardcode them initially at $|x_0| < 2$ and $|y_0| < 2$.
- Have the main program print out or save, for every point z_0 of your $N \times N$ grid, a line of format

$$x_0 \quad y_0 \quad k(z) \quad f(z) \quad \log_{10}(n_{\texttt{itns}})$$

with one blank line after each row of the matrix (if you are working in C in linux). Here $k(z)$ is either the complex root to which the method converges, or zero. n_{itns} is the number of iterations taken to converge, if it does.

- ▼ Plot $\Im(k(z))$ on the (x, y) plane with a colorscaling in map mode (see hint). Find an interesting region and look at it closer. Repeat. Choose an interesting region and save the plot (with full labels of axes of course!).
- ▼ Plot $\log(n_{\text{itns}})$ in the (x, y) plane, focusing around a region where the solver does not converge. Comment on what you see.

2. Look at the website http://www.chiark.greenend.org.uk/~sgtatham/newton/ and use your basic code with different functions to explore and understand the convergence and sensitivities of Newton-Raphson. Consider particularly solutions to the function

$$f(z) = 35z^9 - 180z^7 + 378z^5 - 420z^3 + 315z \tag{1.5}$$

Extra: Choose another function of your choice to present to and discuss with the class.

Tips, Hints, and Suggestions:

- If you are on a linux machine and you choose to print out the results to stdout, the program `gnuplot` can be used to study the results as follows:

  > gnuplot
  gnuplot> set pm3d map
  gnuplot> splot "<./ComplexRoots" u 1:2:3

  where `ComplexRoots` is the program name. (If your on another architecture ... perhaps write to a file and read into your favorite visualizer?).

- There was no standard for complex numbers in C until the C99 standard. The compiler `gcc` (and others) know a data type `double complex`, which requires including the extra header "complex.h". A complex number $z = 3 + 5i$ is then set by
  `double complex z = 3.0 + 5*I`.
  The absolute value, real part, imaginary part, and complex phase of a complex number can be calculated by the similarly provided functions `cabs(z)`, `creal(z)`, `cimag(z)`, and `carg(z)` respectively. C++ has <complex> containers, akin to <vector>. Matlab assumes any number could be complex.

- The roots of $f(z) = z^3 - 1$ are $\{1, e^{\pm\frac{2}{3}\pi i}\}$. Or 1 and $(-0.5, \pm 0.8660254040\, i)$.

2 | Linear Systems & Matrices

To be returned by 12.11.2024

PROBLEM I

Should the system of equations be linear but large is size, matrix methods are more useful. Consider the system of equations

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 10 \\ 5x_2 + 6x_3 &= 11 \\ 9x_3 &= 12 \end{aligned} \tag{2.1}$$

B1. Write a function which multiplies an N-element vector with a $N \times N$ matrix and prints the resulting vector to screen (even Matlab users). This should be $\lesssim 10$ lines of code. Use this function to calculate the following test case:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \cdot \begin{pmatrix} 10 \\ 11 \\ 12 \end{pmatrix} \tag{2.2}$$

B2. Write a function that solves the system of equations

$$\mathbf{M} \cdot \vec{x} = \vec{b}$$

assuming that $\mathbf{M}$ is an upper-diagonal matrix (back-substitution). Choose a suitable test case. Build in a test to make sure the solution is valid (and/or) test the quality of the solution, then solve the above system of equations. This program will be built upon in the next lab.

Tips, Hints, and Suggestions:

- Represent both matrices and vectors as arrays. For matrices, one may define a two-dimensional array $m_{ij} = m[i][j]$. Whether defined as 1- or 2-dimensional, one can access an N^2 array's elements in a 1-dimensional sense as $m_{ij} = m[i * N + j]$. A helper function `get_index(i,j,N)` can be useful.
- Remember that indices in C start at zero.
- The name of an array alone is a pointer to the first element of the array, $m = \&m[0]$.
- Good coding practice does not use single-letter variables except for canonical loop variables i, j, k.

PROBLEM II

1. Write a function with a prototype (in C) [you may use your favourite computer language as well.]
   `int gauss_solve( double * Mij, double * bi, double * x, int n )`
   that solves the linear system of equations $\mathbf{M} \cdot \vec{x} = \vec{b}$ using Gaussian elimination. Initially, assume no pivoting is necessary.

2. Use the above program to solve the system of equations

$$\begin{pmatrix} 2.0 & 0.1 & -0.2 \\ 0.05 & 4.2 & 0.032 \\ 0.12 & -0.07 & 5.0 \end{pmatrix} \cdot \vec{x} = \begin{pmatrix} 10 \\ 11 \\ 12 \end{pmatrix}$$

   and verify the solution.

3. (This question is optional) First use your program as to solve the following system of equations as well.

$$\begin{pmatrix} 1 & 1 & 0 \\ 2 & 2 & -2 \\ 0 & 3 & 15 \end{pmatrix} \cdot \vec{x} = \begin{pmatrix} 1 \\ -2 \\ 33 \end{pmatrix}$$

   Record your result, then add pivoting to the implementation. How does the answer compare between pivoting and no pivoting?

Tips, Hints, and Suggestions:

- For all this exercise, expand on your program for part B of Exercise Set 2.
- *Without* Pivoting, the Gauss method can be described by the following pseudo-code:

---

for $i = 0$ to size-1 **do**
  Divide row i by the diagonal element m_{ii}
  Divide $b[i]$ by the diagonal element m_{ii}
  for $j = i + 1$ to size-1 **do**
    Subtract (row i) * m_{ji} from row j
    Subtract $b[i] * m(j, i)$ from $b[j]$
  end for
end for

  This method creates an upper-diagonal matrix.

- After the matrix and vector has been modified into a system of equations with an upper-diagonal operator and a modified $b[i]$, use (or copy) the back-substitution code from the previous exercise to solve the equation.
- When implementing, attempt to be efficient by setting up the loops so as to avoid multiplying elements that are already guaranteed to vanish. Start with the above pseudo-code and implement this optimization after the above has already been tested (Best Practices in Scientific Coding #6)
- Remember, when passing C-arrays as function arguments, you need the argument to be a pointer to the array. Inside the function, there is no way to test how large the array is, so you will also need to pass integers describing the sizes of the array.

PROBLEM III

1. Find the largest eigenvalue and the corresponding eigenvector of the matrix

$$\begin{pmatrix} 6 & 5 & -5 \\ 2 & 6 & -2 \\ 2 & 5 & -1 \end{pmatrix} \tag{2.3}$$

2. Use Aitken's acceleration to improve the final solution
3. Optional: calculate the other 2 eigenvalues.

Tips, Hints, and Suggestions:

The eigenvalues are (1, 4, 6). The power method gives $r_8 = 6.00000198458$.

PROBLEM IV (optional)

Can you generate using random numbers a real $n \times n$ matrix $\mathbf{A}$ and a random real n-dimension vector $\vec{b}$ and then solve the system $\mathbf{A} \cdot \vec{x} = \vec{b}$ using Gauss-Seidel method or overelaxation? [This exercise can replace any of the first two or come in addition].

3 Interpolation & Extrapolation

3.1 | Interpolation Example – Equation of State (*)

To be returned by 19.11.2024

NOTE: You should try problems I and II. Problem III will be solved later once Praveen discusses the associated physics.

PROBLEM I

Solve one of the following 2.
You may use MATHEMATICA or MAPLE if you have no time for the detailed calculations

Problem I-A Prove that the Padé approximations $P[3,4]$ and $P[2,5]$ of the function $f(x) = e^x$ are the following:

$$\begin{aligned} P[7,0] &\approx 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7 \quad \text{(Taylor)} \\ P[3,4] &\approx \frac{1 + \frac{3}{7}x + \frac{1}{14}x^2 + \frac{1}{210}x^3}{1 - \frac{4}{7}x + \frac{1}{7}x^2 - \frac{2}{105}x^3 + \frac{1}{840}x^4} \\ P[2,5] &\approx \frac{1 + \frac{2}{7}x + \frac{1}{42}x^2}{1 - \frac{5}{7}x + \frac{5}{21}x^2 - \frac{1}{21}x^3 + \frac{1}{168}x^4 - \frac{1}{2520}x^4} \end{aligned} \tag{3.1}$$

Then test which one of the three is more accurate, compare at $x = 0.5, 1, 2, 5$. What do you observe for the accuracy of the 3 cases. Also, try to plot the 3 approximations as well as the original function.

Problem I-B

Prove that the Padé approximations $P[2,3]$ and $P[4,1]$ of the function $f(x) = e^x$ are the following:

$$\begin{aligned} P[5,0] &\approx 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 \quad \text{(Taylor)} \\ P[2,3] &\approx \frac{1 + \frac{2}{5}x + \frac{1}{20}x^2}{1 - \frac{3}{5}x + \frac{3}{20}x^2 - \frac{1}{60}x^3} \\ P[1,4] &\approx \frac{1 + \frac{1}{5}x}{1 - \frac{4}{5}x + \frac{3}{10}x^2 - \frac{1}{15}x^3 + \frac{1}{120}x^4} \end{aligned} \tag{3.2}$$

Then test which one of the three is more accurate, compare at $x = 0.5, 1, 2, 5$. What do you observe for the accuracy of the 3 cases Also, try to plot the 3 approximations as well as the original function.

PROBLEM II

Find the polynomial of 3rd degree that takes the 4 values listed in the $f(x_k)$ column below at the corresponding arguments x_k.

k	x_k	$f(x_k)$	$\Delta f(x_k)$	$\Delta^2 f(x_k)$	$\Delta^3 f(x_k)$
0	4	1			
			2		
1	6	3		3	
			5		4
2	8	8		7	
			12		
3	10	20			

Table 3.1: Difference matrix

Hint: After you construct the Newton polynomial if you simply it you will get

$$P(x_k) = \frac{1}{24}\left(2x_k^3 - 27x_k^2 + 142x_k - 240\right)$$

PROBLEM III

NOTE: This will be solved at a later time. José will help you understand what is needed.

Equations of states (EoSs) in hydrodynamics are the relation of the pressure locally to the local variables of state of a system (e.g. matter density ρ, specific internal energy ϵ or temperature T). Simulations of hydrodynamic systems generally use an EOS to parameterize the microphysics which cannot be captured in first principle by continuum hydrodynamic equations. Only the simplest regimes have analytic equation EOSs, like an ideal gas law $P = \rho\epsilon(\Gamma - 1)$. In the regime of neutron star interiors, the applicable EoSs are very much a topic of research. Realistic EoSs for neutron star interiors are often generated by physicists using varying degrees of relativity, quantum field theory, and particle physics models.

1. Write a function with a prototype similiar to
   `double interp( double * xs, double * ys, int n, double x )`
   that uses Lagrange-Polynomial interpolation in its generalized form for an unspecified number of known positions (n). Here `xs` are the points where the solution is known, `ys` the evaluated function at these points, `n` the number of known points, and `x` the location at which the value of the function is desired.

2. While the simplest EoS, the barotropic EoS, is of the form $P(\rho) = \kappa\rho^{\Gamma}$, a simple step to add more physics (e.g. by imitating a change in chemistry above a certain density ρ_0) is to create a hybrid-barotropic EoS which we will approximate here as:

   $$P(\rho) = H(\rho_T - \rho)\kappa_1\rho^{\Gamma_1} + H(\rho - \rho_T)\kappa_2\rho^{\Gamma_2} \tag{3.3}$$

   where $\Gamma_1 = 4/3$, $\Gamma_2 = 5/3$, κ_1=20, κ_2=1. For the transition function, $H(\rho)$, use one of the following analytical approximations of the Heaviside function:

   $$H(x) = \frac{1}{1 + e^{-2\alpha x}} \tag{3.4}$$

   $$H(x) = \frac{1}{2}(1 + \tanh(\alpha x)) \tag{3.5}$$

   with the transition density $\rho_T = 5$, and transition speed parameter $\alpha = 5$. Analytically fill the known data arrays, $xs[i] = \rho_i$ and $ys[i] = P(\rho_i)$ over a uniform distribution of n points $x_i = \rho_i$ on the interval $I = [\rho_0, \rho_1]$.

3. *Testing and Analysis:* Create a second set of arrays x_k and y_k for a different set of points x_k uniformly distributed across I with $m = 200$ elements. Fill the y_k array by calling the `interp` function you have just written over the data xs and ys.

   A. Look at the interval $I = [4.5, 5.5]$. Plot the interpolated values for $n = 5, 10, 20$ together with the analytical function. Discuss the errors.

   B. Repeat the above for the interval $I = [0, 10]$. What happens now?

   C. Calculate the chi-square error for the entire function given a number of interpolated points m as

   $$E = \sqrt{\frac{1}{m}\sum_{k=0}^{m-1}(f(x_k) - y_k)^2}. \tag{3.6}$$

Plot $\log(E)$ vs n for a range $n = 3$ to $n = 40$ for the intervals $I = [4.5, 5.5]$, $I = [0, 10]$, and $I = [0 : 30]$.

D. Finally, given a set interval around the transition density ρ_0 of your choosing, explore on your own how different parameters of the above hybrid EoS affect the ability for the interpolator to get the pressure right. Be prepared to discuss this with the class.

Tips, Hints, and Suggestions:

- In order not to need to recompile for each choice of n, make this number a command line argument. To catch commandline arguments,

```
1 int main( int argc, const char **argv) {
2   if ( argc != 2 ) return -1; \\ argc = number of parameters + 1
3   int n = atoi(argv[1]); \\ argv[N] = Nth parameter
4   \\ [...]
5   return 0;
6 }
```

  then the program can be executed with the value of $n = 5$, for example, by executing `./myinterp 5`.

- For those on linux, to plot data graphically, one way to plot this is using the program `gnuplot`:

  >gnuplot
  gnuplot> plot [0:10] 1/(1+exp(-2*a*x)) ... with l
  gnuplot> replot "<./myinterp 5" with p
  gnuplot> replot "<./myinterp 10" with p
  gnuplot> replot "<./myinterp 20" with p

  where `[0:10]` is the desired interval. For others, I would suggest writing a file I/O routine and plot with your favorite visualizer.

- If you're feeling adventurous, make the function `interp` take an array of points to interpolate and an array in which to provide the results. Overhead from function calls can be expensive. This will get you ahead for the next week's exercises.

NOTE:
The suggestion to use specific routines or programs e.g. here "gnuplot" is optional. You may use any other program that serves the purpose.

4 Numerical Differentiation

4.1 | Numerical Differentiation of Functions

To be returned by 26.11.2024

PROBLEM I

Prove the central difference relation

$$y(x_0)'' = y_0'' = \frac{y_1 - 2y_0 + y_{-1}}{h^2} - \frac{h^2}{12} y^{(4)}(\xi) \tag{4.1}$$

Try to calculate the explicit form of the error term $O(h^2)$.

PROBLEM II

Prove the central difference relation

$$y_0'' = \frac{-y_2 + 16y_1 - 30y_0 + 16y_{-1} - y_{-2}}{12h^2} + O(h^4) \tag{4.2}$$

Try to calculate the explicit form of the error term $O(h^4)$.

PROBLEM III

This problem will be addressed at a later time

Consider the function $f(x) = \tan^{-1}(x)$ and calculate the derivative at $x = 0$:

a) using 2 iterations of Richardson extrapolation and compare with the results of the central difference relations.

b) using splines.

5 Numerical Integration

Problems I, II, III by 3.12.2024 & IV and V by 10.12.2024

PROBLEM I

Find analytically which one of the formulae in Table I we practically implement if we combine Simpson (1/3) with Romberg.

PROBLEM II

Calculate the integral

$$f(x) = \frac{2^x \sin(x)}{x}$$

by using:

1. Simpson (1/3)
2. Romberg with Simpson (1/3)
3. Gauss-Legendre with 4 points (use either data from table 2, or from page 28).

PROBLEM III

Prove **numerically** the following:

$$\int_0^1 \left(\int_0^2 xy^2 dx \right) dy = \frac{2}{3}$$

$$\int_0^1 \left(\int_{2y}^2 xy^2 dx \right) dy = \frac{4}{15}$$

$$\int_0^2 \left(\int_0^{x/2} xy^2 dy \right) dx = ?$$

PROBLEM IV

Quantum Mechanics Application (*)

Besides root-finding, integration is the most ubiquitous fundamental numerical method used in computational physics and astrophysics.

- ► Write a function to fill arrays with the analytical function, to be provided to an integration routine as data. The function should take as arguments the array to be filled, the interval, the number of data points, the *function* to be evaluated, and the stepsize of the data points h.
- ► Write a function for numerical integration using the Newton-Cotes with 2nd order polynomial. Assume half an odd number of data points. The function should return not only the integrated value, but also a measure of its error, E (see tips below).
- ▼ *Application and Testing:*
  (a) Use both above functions to integrate the polynomial $f(x) = x + x^3$ in the interval $[0, 1]$ with $n = 5$ data points. The numerical result should be, up to rounding errors, exact. Why?
  (b) Suppose a particle in a potential well has a wavefunction of the form

$$\Psi(x,t) = \begin{cases} \frac{1}{\sqrt{L}}\left[\sin(\frac{\pi x}{L})e^{-\imath\omega_1 t} + \sin(\frac{2\pi x}{L})e^{-\imath\omega_2 t}\right] & \text{if } 0 < x < L \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

  The probability distribution of the particle $P(x,t) \equiv |\Psi(x,t)|^2$. Find the probability that the particle is in the far right quarter of the potential well

$$P(t \,|\, x \in [3L/4, L]) = \int_{3L/4}^{L} P(x,t)dx \tag{5.2}$$

  for times $t \in \{0, \pi/\Delta\omega\}$ where $\Delta\omega = \omega_2 - \omega_1$. Assume $L = 2, \omega_1 = 3$, and $\omega_2 = 4.5$. Compare the results for all odd data numbers $n \in [5, 501]$. For each n, print n, $\log(E)$, and $\log(h)$ to stdout (or a file). Plot $\log(E)$ vs. $\log(h)$ for both times. Are the plots compatible with the expected convergence rate?
- ▼ *Extra Application*: Consider instead the wavefunction of an electron in the $2s$ state of a hydrogen atom:

$$\Psi_{2s}(r) = \frac{1}{4\sqrt{2\pi a_0^3}}\left[2 - \frac{r}{a_0}\right]e^{-r/(2a_0)} \tag{5.3}$$

  where $a_0 = \frac{\hbar^2}{me^2} = 0.0529$ nm is the first Bohr radius.
  - Calculate the mean radius $\langle r\rangle$ as well as its standard deviation $\sigma^2 = \sqrt{\langle(r - \langle r\rangle)^2\rangle} = \sqrt{\langle r^2\rangle}$ for the $2s$ state knowing

$$\langle r^n\rangle = 4\pi\int_0^\infty \Psi^*(r,t)r^n\Psi(r,t)dr. \tag{5.4}$$

    Find a way to handle the integration endpoints.

Tips, Hints, and Suggestions:

- In practice, it is often necessary to pass a function as a parameter. If you haven't done this yet, in C, an example of this is:

```
1 double wah(double x) { return x*x; }
2
3 double at_zero(double (*fun) (double)) {
4   return fun(0.0);
5 }
6
7 int main() {
8   double a = at_zero(wah);
9 }
```

- To release the assumption of an odd number of data points, one could use the 4-point form for the last section. The reason to choose the 4-point form is that this form has the same convergence order as that used for the rest of the method.

- Remember, the error estimate of the method is proportional the the 4th derivative at the center of the range. Look closely at the central differencing scheme:

$$f_0^{(4)} = \frac{f_2 - 4f_1 + 6f_0 - 4f_{-1} + f_{-2}}{h^4} + O(h^2) \tag{5.5}$$

  and the error for the 2nd order Newton-Cotes integration scheme:

$$E = \frac{1}{90} h^4 f^{(4)}(\chi_1) \tag{5.6}$$

- If you begin with the solution for last time, you'll end up looping over all the n. One way around this is to have a wrapping function which calls what used to be the main function, looping over the number of data points, n. A quick workaround is also to use loops and output redirection in bash (that environment you're typing into when you compile if you aren't using an IDE). At the command line ...

  > for i in (5..501); do
  > rem=$(($i % 2));
  > if [$rem -ne 0]; then
  > ./exe/myIntegration $i >> IntegrationData.txt
  > fi
  > done

  The operation % is the modulo operator. The >> operator takes stdout from your program and appends it to the file following the operator.

PROBLEM V

Mode Decomposition Application (*)

Integration is so integral to computational physics that we will take another week on its basic application. The application this week will be focused on mode decomposition. In many nonlinear physical processes the behavior of a system is considered and understood through an interaction / energy transferral between different modes. Examples of this are neutron star oscillations, gravitational wave content/black hole ringdown, and turbulence.

For an *orthonormal* basis $b_i(\vec{r})$, data which is of the form $\Psi(\vec{r}) = \Sigma_i^\infty A_i b_i(\vec{r})$ can be decomposed through a set of integrals

$$A_i = \int b_i^*(\vec{r})\Psi(\vec{r})dV$$

- Establish functions with a prototype akin to `intg_mymethod(struct integrand)` for numerically integrating a data-based integrand via 2 of the following methods:
  (a) Trapezoidal Rule
  (b) 2nd order Newton-Cotes (from last lab)
  (c) Simpson's 3/8 Rule
  (d) Euler-Maclaurin method (a generalization of integration using splines)
  (e) Gauss-Legendre methods of order 2, 4, and 8
  (f) Splines*

  Make sure to note, and handle accordingly, when the integration method requires a specific integral form and/or boundaries!

- *Testing and Comparison* of your method functions

  - *Basic Comparison:* For each function below, compare the behavior of the *applicable* integration methods over their corresponding intervals, I, rewriting the function to fit a specific form/bounds if straightforwardly possible.

$$f_1(x) = e^x \cos(x), \quad \text{in I=[0,}\pi/2] \tag{5.7}$$

$$f_2(x) = e^x, \quad \text{in I=[-1,3]} \tag{5.8}$$

$$f_3(x) = \begin{cases} e^{2x} & x < 0 \\ x - 2\cos(x) + 4 & x \geq 0 \end{cases}, \quad \text{in I=[-1,1]} \tag{5.9}$$

    For each function and method, calculate the actual error E using the analytical answer for all $n \in [5, 501]$, then plot $\log(E)$ vs $\log(h)$ for all methods (properly labeled – a gnuplot script is useful here to streamline changing functions.). That is, for each function you should have one graph comparing the performance of the methods. Study the convergence rate for the methods and discuss:
    (a) Convergence rates: What was expected? What was observed? Can you give a reason if it these do not match?
    (b) What other integration method could be applied to which integrands to possibly achieve better performance?

- *Application:* [1]Provided on the exercise's website is a sample data set, two resolutions & ascii text format, of $\Psi(\theta)$ from an imaginary π-symmetric experiment/simulation. Find the coefficients A_k

[1]This part of the exrcise leave it aside till we discuss the Partial Differential Equations

for the decomposition of this data of the form

$$\Psi(\theta) = \Sigma_{i=0}^{k_{\max}} A_k P_k^0(\cos(\theta))$$

where $P_l^m(\cos(\theta))$ are the Legendre polynomials ($m = 0$ imposing axisymmetry) whose normalization condition is

$$\int_{-1}^{1} P_n(x) P_k(x) dx = \left(\frac{2}{2n+1}\right) \delta_{nk}$$

You can create a function to provide analytic hard-coded Legendre polynomials up to a reasonable order for applying to an integrand. Make sure you address the normalization properly! The coefficients used in the generation of the data is in the file header.

(a) Which integration method did you choose, and why?

(b) Which modes were you able to recover?

(c) What are the various sources of error in the mode decomposition?

(d) Given the provided data set, which $k_{\max}$ makes sense? Which coefficients are trustworthy?

Tips, Hints, and Suggestions:

- Consider creating your integration method routines to take only a structure, e.g. `struct integral` which contains:
  - `N`, the number of data points to assume
  - `I[2]`: a 2-element array containing the interval of integration
  - `*xdata, *ydata`: pointers to data arrays (declared & allocated elsewhere)
  - `(*get_f)` and/or `(*get_fdf)`: Function handles for the integrand

  This way a simple call to a helper function `fill_integrand(struct integral *)` can fill in the integrand with the current function handle and size n.
- Find a work-around for Simpson's 3/8 Rule where n is not a factor of 3.
- For the Gauss-Legendre method, the roots are given for the specified orders in the lecture notes.
- For spline integration, exploit your matrix libraries from Lab 3. I would suggest developing the lab without this integration method, and then spending time on this.