PHYS2160 Introductory Computational Physics 2021/22 Semester 2 Summary

Chapter 1 Introduction

- What is Computational physics?
- Basic Concepts of Computer Programming
- Overview of Low-level and High-level Programming Languages
- Programming with Python

Chapter 2 Python Programming for Physicists

- Basic Elements of a Program
- Expressions and Assignments
  - ⋆ Note that everything in Python is an object!
  - ⋆ Python uses a "variable as a sticky note" model for assignment.
- Numeric and Boolean Data Types: `int`, `float`, `complex`, and `bool`
- Operators for Numeric Data Types

  How to perform operations for different numeric data types?
- Functions, Modules, and Packages

  How to use built-in functions as well as functions defined in modules and packages?
- Input and Output Statements
- Control Structures
  - ⋆ One-way decisions using `if` vs two-way decisions using `if-else` vs multi-way decisions using `if-elif-else`
  - ⋆ Indefinite loops using `while` vs definite loops using `for`
  - ⋆ Use of `break`, `continue`, and `pass` in loops
  - ⋆ Note that all compound statements can be nested!

- Strings
  - ⋆ Commonly used escape sequences
  - ⋆ Indexing and slicing of strings
  - ⋆ Common methods for manipulating and transforming strings
- List and Tuples
  - ⋆ How to construct lists and tuples?
  - ⋆ Similarities and differences between lists, strings, and tuples
  - ⋆ Common methods for handling lists
  - ⋆ List comprehension for creation of lists
  - ⋆ Beware of how to create an independent copy of a list!
  - ⋆ Tuple packing and unpacking
  - ⋆ Working on iterable objects with `range` and `enumerate`
- Formatted Input and Output

  How to create formatted strings by using `format` and f-strings?
- User-defined Functions
  - ⋆ Components in a function definition
  - ⋆ Positional arguments vs keyword arguments
  - ⋆ How to set default arguments in a function definition?
  - ⋆ Scope of variables in function definitions (LEGB rule)
  - ⋆ Note that in Python arguments are passed by value, but that value is a reference to an object.
  - ⋆ Nested functions and recursive functions
- File Processing

  How to read data from a file? How to write data from a file?
- Arrays
  - ⋆ Creation of arrays
  - ⋆ Indexing and slicing arrays
  - ⋆ Copying and sorting arrays

  * ⋆ Shape manipulation of arrays
  * ⋆ Operations and comparisons of arrays
  * ⋆ Universal functions
  * ⋆ Statistical analysis of arrays

- Dictionaries
  * ⋆ Creation of dictionaries
  * ⋆ Working on keys and values in dictionaries
  * ⋆ Dictionary comprehensions

Chapter 3 Object-oriented Programming in Python

- Basic Concepts of Object-Oriented Programming
  * ⋆ A class is a blueprint from which individual objects are made.
  * ⋆ Attributes and methods of a class
  * ⋆ Superclass and subclass

- Creating Classes in Python
  * ⋆ Components in a class definition
  * ⋆ Special methods of classes (`__init__`, `__call__`, `__str__`, `__repr__`, ...)
  * ⋆ Class variables, static methods, and class methods

- Class Inheritance in Python
  * ⋆ How to create a subclass from a superclass?
  * ⋆ Relation between superclass and subclass instances
  * ⋆ Creating subclass using "is-a relationship" and "has-a relationship"

Chapter 4 Scientific Programming with NumPy, Matplotlib, and SciPy

- NumPy
  * ⋆ Reading and writing an array to a file

    How to read/write arrays to files using `save`, `load`, `loadtxt`, `genfromtxt`? How to store the data read from a file as a structured array?

- ⋆ Polynomials

  How to work on polynomials using the `Polynomial` class?
  How to work on classical orthogonal polynomials using classes such as `Legendre` and `Hermite`?

- ⋆ Linear Algebra

  How to work on matrices through numpy arrays using `dot`, `transpose`, `det`, ...?
  How to solve systems of linear scalar equations using `solve`?
  How to find the "line of best-fit" using `lstsq`?

- ⋆ Random Sampling

  How to create uniformly distributed random numbers using `rand`, `randint`, `random_sample`, ...?
  How to create random samples from the normal, binomial, and poisson distributions using `normal`, `binomial`, and `poisson`?

- Matplotlib
  - ⋆ Basics for using matplotlib
  - ⋆ Scatter plots and contour plots
  - ⋆ Bar charts and pie charts
  - ⋆ Multiple subplots
  - ⋆ Heatmaps
  - ⋆ Polar plots and histograms
  - ⋆ 3D plots

- SciPy
  - ⋆ Physical Constants and Special Functions

    How to load the values of the physical constants provided by the `scipy.constants` package? How to evaluate special functions using the `scipy.special` package?

  - ⋆ Integration and Ordinary Differential Equations

    How to use `quad`, `dplquad`, `tplquad`, `nquad` from the `scipy.integrate` to evaluate ordinary, double, triple, multiple integrals? How to use the function `scipy.integrate.odeint` to solve single 1st-order ODEs, coupled 1st-order ODEs, and 2nd-order ODEs?

- ⋆ Interpolation

  How to use `interp1d`, `interp2d`, `RectBivariateSpline`, `griddata` from the `scipy.interpolate` package to estimate intermediate values from a set of known data points by employing different interpolation methods?

- ⋆ Data-fitting and Root-finding

  How to use `leastsq` and `curve_fit` from the `scipy.optimize` package to find the best fit curve to a set of data points by performing weighted and unweighted least square fitting? How to use `brentq`, `brenth`, `ridder`, `bisect`, and `newton` from the `scipy.optimize` package to obtain the roots of both univariate and multivariate functions?

Chapter 5 Errors and Uncertainties in Computation

- Types of Errors

  Blunders or bad theory, random errors, truncation errors, and round-off errors

- Representation of Numbers in Computer
  - ⋆ All numbers are stored in memory in binary form.
  - ⋆ Single-precision and double-precision floating point numbers
  - ⋆ Representation of floating-point numbers in Python

- Round-off Error in Computation
  - ⋆ If we subtract two large numbers and end up with a small one, then there will be less significance, and possibly a lot less significance, in the small one.
  - ⋆ We can approximate the accumulation of round off error in a calculation involving a large number of steps by viewing the error in each step as a literal step in a random walk.

- Propagation of Errors in Numerical Algorithms
  - ⋆ The truncation error decreases rapidly with the number of steps. In contrast, the round-off error tends to grow slowly and somewhat randomly with the number of steps.
  - ⋆ The best is to quit the calculation when the round-off error becomes approximately equal to the truncation error.

Chapter 6 Numerical Calculus

- Simple Methods for Numerical Integration
  - ⋆ Trapezoidal rule

$$\boxed{\int_a^b f(x)dx \approx h\left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{k=1}^{N-1} f(a+kh)\right]}$$

  - ⋆ Simpson's rule

$$\boxed{\begin{aligned}\int_a^b f(x)dx \approx\ & \frac{1}{3}h\Bigg[f(a) + f(b) + 4\sum_{k=1}^{N/2} f(a+(2k-1)h) \\ & +2\sum_{k=1}^{N/2-1} f(a+2kh)\Bigg]\end{aligned}}$$

- Errors on Numerical Integrals
  - ⋆ Truncation error for trapezoidal rule

$$\boxed{\epsilon = \frac{1}{12}h^2[f'(a) - f'(b)]}$$

  - ⋆ Truncation error for Simpson's rule

$$\boxed{\epsilon = \frac{1}{180}h^4[f'''(a) - f'''(b)]}$$

- Number of Steps for Numerical Integration
  - ⋆ Adaptive trapezoidal rule

$$\boxed{I_i = \frac{1}{2}I_{i-1} + h_i\sum_{j=1}^{N_i/2} f(a+(2j-1)h_i)} \text{ with error } \boxed{\epsilon_i = \frac{1}{3}(I_i - I_{i-1})}$$

  - ⋆ Adaptive Simpson's rule

$$\boxed{S_i = \frac{1}{3}\left[f(a) + f(b) + 2\sum_{j=1}^{N_i/2-1} f(a+2jh_i)\right]}$$

$$\boxed{T_i = \frac{2}{3}\sum_{j=1}^{N_i/2} f(a+(2j-1)h_i)}, \qquad \boxed{S_i = S_{i-1} + T_{i-1}},$$

$$\boxed{I_i = h_i(S_i + 2T_i)} \text{ with error } \boxed{\epsilon_i = \frac{1}{15}(I_i - I_{i-1})}$$

- Romberg Integration

$$\boxed{R_{i,m+1} = R_{i,m} + \frac{1}{(4^m-1)}(R_{i,m} - R_{i-1,m})} \text{ where } \boxed{R_{i,1} = I_i}$$

$$\text{Error of } R_{i,m}\text{: } \boxed{c_m h_i^{2m} = \frac{1}{(4^m-1)}(R_{i,m} - R_{i-1,m}) + O(h_i^{2m+2})}$$

- Higher-order Integration methods

  We can make a higher-order approximation to an integral by fitting a higher-order polynomial instead of a straight line or a quadratic curve!

- Numerical Derivatives

  ⋆ Forward difference

$$\boxed{f'(x) \approx \frac{f(x+h) - f(x)}{h}} \text{ with the error of } O(h)$$

  ⋆ Backward difference

$$\boxed{f'(x) \approx \frac{f(x) - f(x-h)}{h}} \text{ with the error of } O(h)$$

  ⋆ Central difference

$$\boxed{f'(x) \approx \frac{f(x+h/2) - f(x-h/2)}{h}} \text{ with the error of } O(h^2)$$

$$\boxed{f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h}} \text{ with the error of } O(h^2)$$

  ⋆ Unlike numerical integrals, both round-off and truncation errors are important for numerical derivatives.

  ⋆ We can derive the higher-order approximations to $f'(x)$ by fitting a higher-order polynomial to a set of sample points and then calculate the derivative of the polynomial at x.

- Interpolation

  Linear interpolation

$$\boxed{f(x) \approx \frac{(b-x)f(a) + (x-a)f(b)}{(b-a)}}$$

  with the error of $O(h^2)$

Chapter 7 Solutions of Nonlinear Equations

- The Relaxation Method

$$\boxed{x_{i+1} = f(x_i)},$$

$$\text{Error of } x_{i+1}\text{: } \boxed{\epsilon_{i+1} \approx \frac{x_i - x_{i+1}}{1 - 1/f'(x_i)}} \text{ or } \boxed{\epsilon_{i+1} \approx \frac{(x_i - x_{i+1})^2}{(2x_i - x_{i-1} - x_{i+1})}}$$

  It will converge to a solution at x^* if and only if $|f'(x^*)| < 1$.

- The Bisection Method

  (1) Choose x_1 and x_2 so that $f(x_1)f(x_2) < 0$, and a target accuracy δ for your answer.

  (2) Calculate the midpoint $x_m = (x_1 + x_2)/2$ and evaluate $f(x_m)$.

  (3) If $f(x_m)f(x_1) > 0$, then set $x_1 = x_m$; otherwise set $x_2 = x_m$.

  (4) If $|x_1 - x_2| > \delta$, repeat from step 2. Otherwise, calculate the midpoint x_m once more to get the final estimate of the position of the root.

- The Newton's Method

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}}, \qquad \text{Error of } x_{i+1}\text{: } \boxed{\epsilon_{i+1} \approx x_{i+2} - x_{i+1}}$$

  It requires us to know $f'(x)$ and it can fail to converge if the shape of $f(x)$ is unfavorable.

- The Secant Method

$$\boxed{x_{i+2} = x_{i+1} - f(x_{i+1}) \left[\frac{x_{i+1} - x_i}{f(x_{i+1}) - f(x_i)}\right]}$$

  It is similar to the Newton's method.

- Methods for Two or More Variables

  Both relaxation and Newton's methods can be generalized easily to the solution of simultaneous nonlinear equations in two or three variables.

- Beware of the pros and cons for each of the methods discussed here!

Chapter 8 Ordinary Differential Equations

- First-order Differential Equations

  ⋆ Euler's method

  $$\boxed{x(t+h) \approx x(t) + hf(x,t)} \text{ with the error of } O(h^2)$$

  ⋆ Second-order Runge-Kutta method

  $$\boxed{\begin{gathered} k_1 = hf(x,t),\ k_2 = hf\left(x + \frac{1}{2}k_1, t + \frac{1}{2}h\right) \\ x(t+h) \approx x(t) + k_2 \end{gathered}} \text{ with the error of } O(h^3)$$

  ⋆ Fourth-order Runge-Kutta method

  $$\boxed{\begin{gathered} k_1 = hf(x,t),\ k_2 = hf\left(x + \frac{1}{2}k_1, t + \frac{1}{2}h\right) \\ k_3 = hf\left(x + \frac{1}{2}k_2, t + \frac{1}{2}h\right),\ k_4 = hf(x + k_3, t + h) \\ x(t+h) \approx x(t) + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \end{gathered}}$$

  with the error of $O(h^5)$

  ⋆ Beware of the pros and cons for each of these methods!

- Simultaneous Differential Equations

  $$\boxed{\begin{gathered} \mathbf{k}_1 = h\mathbf{f}(\mathbf{r},t),\ \mathbf{k}_2 = h\mathbf{f}\left(\mathbf{r} + \frac{1}{2}\mathbf{k}_1, t + \frac{1}{2}h\right) \\ \mathbf{k}_3 = h\mathbf{f}\left(\mathbf{r} + \frac{1}{2}\mathbf{k}_2, t + \frac{1}{2}h\right),\ \mathbf{k}_4 = h\mathbf{f}(\mathbf{r} + \mathbf{k}_3, t + h) \\ \mathbf{r}(t+h) \approx \mathbf{r}(t) + \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \end{gathered}}$$

  where $\mathbf{r} = (x, y, \ldots,)$ and $\mathbf{f}(\mathbf{r},t) = (f_x(\mathbf{r},t), f_y(\mathbf{r},t), \ldots)$.

- Second-order Differential Equations

  We seek the solution by using the same set of equations for solving simultaneous 1st order ODEs except that $\mathbf{r} = (x, x', \ldots, x^{(n)})$ and $\mathbf{f}(\mathbf{r}, t) = (x', x'', \ldots, f(x, x', \ldots, x^{(n-1)}, t))$.

- Adaptive Step Size Method

  (1) Starting from the same point $x(t)$, perform two steps of size h and one step of size $2h$ to get two estimates x_1 and x_2 of $x(t + 2h)$.

  (2) Calculate $\rho = 30h\delta/|x_1 - x_2|$ where δ is the target accuracy per unit time.

  (3) If $\rho \geq 1$, then keep the results for step size h and move on to time $t + 2h$ to continue our solution but with step size $h' = \min(\rho^{1/4}, 2)h$.

  (4) If $\rho < 1$, repeat the current step again, but with $h' = \rho^{1/4}h$.

  (5) Continue the calculation up to the desired ending time.

  It can be also used for solving simultaneous and higher-order ODEs.

Final Examination

- Date: 16 May 2022 (Monday)
- Time: 9:30 am – 11:30 am
- Platform: OLEX-Moodle
- Topics: Chapters 2 to 8
- Contents: 5 questions for programming plus 2 questions for computational methods (in a similar format as the sample exam paper)

  Answer ALL the questions
- Remarks:

  Remember to bring the calculator!! The list of approved calculators is available here.

  The instructions to candidates sitting online exam can be found here. In particular, students must answer the exam paper by handwriting and use a THIRD device (e. g. another mobile phone or a separate scanner) to scan their handwritten script. And students will be given extra 15 minutes for scanning and uploading their answer script to Moodle.

  Candidates are permitted to bring to the examination FIVE sheets of A4-sized paper with printed/written notes on both sides. Internet searching and crowdsourcing from group messages, online forums or social media, etc. are strictly forbidden.

  Students must attend the mock exam held on 4 May 2022 (Wed), 10:30 am to familiarize themselves with the exam arrangements and test the equipment before the actual exam.