PHAS0030
Practical Mathematics
Exam 2019

[David Bowler]

This page intentionally blank.

Examination conditions.

You will have two and a half hours to complete this examination. During this examination, the only programs that you may run are Firefox/Chrome, and the Jupyter Notebook server from Anaconda. You may use the built-in help in IPython (either add a question mark before the function, or use Shift-Tab with the cursor by the function name). The use of any other program, or any attempt to access websites external to your computer (other than Moodle at the end of the examination for upload), will be taken as a breach of examination regulations.

If you need scrap paper, it will be provided. Take care to save your notebooks at regular intervals: work that is lost because you did not save cannot be marked.

Type your STUDENT NUMBER clearly at the start of your notebook—do NOT include your name. You should name your notebook as follows: PHAS0030_Exam_XXXXX where you should replace XXXXX with your student number. You are advised to restart the kernel and clear output, and then test your complete notebook, before submission.

At the end of the examination, you will upload your answer notebook to Moodle (you will be given instructions on how to do this).

You may only import the modules numpy, matplotlib, scipy and math; you may also use the command `from mpl_toolkits.mplot3d import Axes3D` for 3D plots if required. If you write code that depends on other modules, this will be treated as incorrect.

Answer **all six** questions in Section A and **all three** questions from Section B.

The numbers in square brackets show the provisional allocation of maximum marks per question or part of question.

Section A

[Part marks]

1. Consider the differential equation for the height z of an object falling in one dimension under the influence of gravity and air resistance:

$$m\frac{\mathrm{d}^2 z}{\mathrm{d}t^2} = -mg - \mu\frac{\mathrm{d}z}{\mathrm{d}t} \tag{1}$$

where $g = 9.8\,\mathrm{m/s^2}$ is the acceleration due to gravity and μ is a constant due to air resistance.

   (a) Separate the equation into two first-order equations and write a function that takes a two entry array as input (position and velocity), and returns two first-order derivatives as an array. **[3]**

   (b) For a projectile with $m = 1\,\mathrm{kg}$ and initial height and velocity $z_0 = 10\,\mathrm{m}$, $v_0 = 50\,\mathrm{m/s}$, use `odeint` from `scipy.integrate` to integrate the equations of motion over a total time of 12s with a timestep $\mathrm{d}t = 0.1\mathrm{s}$ with $\mu = 0.0\,\mathrm{kg/s}$. Plot the height as a function of time, and estimate the time when $z = 0$. [Hint: remember to define g]. **[2]**

   (c) Now set $\mu = 0.1\,\mathrm{kg/s}$ and repeat the calculation, plotting the two curves on the same graph. Briefly explain the change in time to hit the ground in a text cell. **[2]**

2. (a) The hyperbolic sine function (sinh) has the following Taylor expansion:

$$\sinh(x) \simeq \sum_{n=0}^{N} \frac{x^{2n+1}}{(2n+1)!} \tag{2}$$

   Write a python function to evaluate this expansion to N terms; the function should take two parameters, x and N. You may use the `math.factorial` function. **[3]**

   (b) Plot curves from $-5 < x < 5$ for the first few values of N, stopping when you think that you have a good approximation. Explain your choice of value for N (you may use `np.sinh` for comparison if you wish). **[3]**

[Part marks]

3. (a) Create a `numpy` array of 630 values of x in the range $0 \leq x < 2\pi$ and an array of the corresponding values of $\sin(x)$. **[1]**

   (b) Use `np.random.random` to add random noise to your sine array with amplitude 0.1 (Hint: use the `size` parameter for `np.random.random`, and think about minimum and maximum values produced by the function). Plot a curve from the array. **[2]**

   (c) Use `np.fft.fft` to transform the array. By zeroing a suitable part of the resulting array and transforming back (using `np.fft.ifft`), remove as much of the noise as you can (explain your choice briefly). Plot the noisy and the smooth curves on the same figure. **[3]**

4. (a) Write a simple function to calculate the forward finite difference of an array, `f`. The function should take two inputs, the array `f` and a spacing `dx`, and return an array of the finite difference `df/dx`. [Hint: be careful about how you shift arrays.] **[4]**

   (b) Create an array containing sin(x) with appropriate range and spacing, and apply your finite difference scheme. Plot this against the analytic result for the derivative of sin(x) and comment on the difference. **[3]**

5. This question uses the rejection method, described at the end of the question, to generate a particular distribution.

   (a) Write a very simple python function to calculate Ae^{-x^2}, where A is an externally defined variable. Now for `x` running from -5 to 5, using `simps` from `scipy.integrate`, find the correct value of A to normalise this function for the range $-5 \leq x \leq 5$. [Hint: note that `simps` takes two arrays as input.] **[2]**

   (b) Write a python function that returns values drawn from a rectangle completely enclosing your Gaussian for the same x range, and plot both on the same graph to be sure. **[2]**

   (c) Write a python function to implement the rejection method to generate a normal/Gaussian distribution (the target distribution) of 1000 random numbers, using the rectangular distribution as the test distribution. You should use the python functions you have just created. Output the number of trials required. Plot a histogram of the resulting distribution (use `plt.hist`) and **briefly** comment on the efficiency of the method. **[3]**

   [The rejection method selects a value x_i at random from the test distribution, $p(x)$. It evaluates $z_i = p(x_i)$. A random number is then chosen, lying between zero and z_i. If this is less than the value of the target function at x_i, $q(x_i)$, then the value of x_i is retained, otherwise it is discarded.]

[Part marks]

6. The populations of predators and prey in an ecosystem can be modelled with the following linked differential equations:

$$\begin{aligned} \frac{\mathrm{d}F}{\mathrm{d}t} &= Fa(1 - Sb) \\ \frac{\mathrm{d}S}{\mathrm{d}t} &= -Sc(1 - Fd)\,, \end{aligned}$$

(where F might represent the population of fish, the prey, and S the population of sharks, the predators, and a, b, c, d are parameters).

(a) Write a function that takes a two component array containing both F and S, and returns the appropriate derivatives as a two component array. **[2]**

(b) Create a simple Euler numerical integration scheme (you need not use a function) to integrate up to a limit of $t = 100$ for a given timestep dt. You should set $F_0 = 1, S_0 = 1$ and the parameters $a = c = 0.3, b = d = 0.9$. Set $dt = 0.001$ and solve. Plot the solutions against time on the same plot (you should see roughly sinusoidal behaviour). **[3]**

(c) Now solve for timestep: $dt = 0.1$ and plot S against F in a separate plot for both timesteps (you may wish to plot the solutions against time as well). What do you observe as the timestep gets larger ? Explain briefly what is happening in a text cell. **[2]**

[Part marks]

Section B

7. In this question, you will solve for the orbit of a planet around a star using a velocity Verlet approach. We will not specify units; use the magnitudes of the quantities given. You should work in two dimensional Cartesian coordinates ($\mathbf{r} = (x, y)$) throughout.

   The following parameters are defined: the gravitational constant $G = 1$; the star has mass M_1, starts at the origin and has initial velocity $\mathbf{v}_1 = (0, 0)$; the planet has mass M_2, starts at $\mathbf{r} = (1, 0)$ and has initial velocity $\mathbf{v}_2 = (0, \omega)$ where $\omega = \sqrt{M_1}$; the period of the star's orbit, $T = 2\pi/\omega$.

   The gravitational force on object a from object b is given by:

$$\mathbf{F}_a = \frac{M_a M_b}{r_{ab}^2}\hat{\mathbf{r}}_{ab} \tag{3}$$

   with $r_{ab} = \mid \mathbf{r}_b - \mathbf{r}_a \mid$, and the unit vector $\hat{\mathbf{r}}_{ab} = (\mathbf{r}_b - \mathbf{r}_a)/r_{ab}$. The velocity Verlet algorithm proceeds as follows:

   - Given positions, $\mathbf{r}(t)$, calculate the forces and accelerations, $\mathbf{a}(t)$;
   - Update the positions: $\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \Delta t\mathbf{v}(t) + \frac{1}{2}\Delta t^2\mathbf{a}(t)$;
   - Calculate a new set of accelerations, $\mathbf{a}(t + \Delta t)$;
   - Update the velocities: $\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \frac{1}{2}\Delta t\,(\mathbf{a}(t + \Delta t) + \mathbf{a}(t))$.

   (a) Write a function to implement Equation 3. Your function should take two two-component position arrays and two masses and return a two-component force array. **[5]**

   (b) Write a function to update the positions, following the velocity Verlet scheme described above. It should take position, velocity, acceleration and timestep (Δt) as arguments, and return updated positions. **[3]**

   (c) Write a second function to update the velocities, taking velocity, accelerations at t and $t + dt$ and timestep as arguments and returning updated velocities. **[3]**

   (d) Write an algorithm to implement the velocity Verlet scheme for the planetary system, setting $M_1 = 200,000$ and $M_2 = 1$. You should cover one period with an appropriate number of points, storing the positions of the planet and the star. Make clear in the code (with comments) the initialisation phase and the iteration phase. **[6]**

   (e) Run the algorithm, and plot the orbit of the planet. **[3]**

[Part marks]

8. We will model a polymer using a square lattice model: each monomer will reside at a point on the lattice. We will build up the polymer by adding one monomer at a time.

   (a) We will work on a 2D square lattice of side length 101, creating polymers of length 100 segments. Use appropriate `numpy` functions to create two **integer** arrays, initialised to zero: the first, which should be two dimensional (101×101), to monitor the occupancy of the lattice sites; the second, which should be a two column vector (2×101), to record the positions of the polymer monomers. **[4]**

   (b) Create an array containing the relative positions of the four neighbours of a point on the lattice ($dx = -1$ or $+1$ with $dy = 0$; or $dy = -1$ or $+1$ with $dx = 0$) **[2]**

   (c) Write a function to increment the polymer (to increase the length of the polymer by adding a monomer to the end). It should take the current monomer index, the polymer array and the lattice array as arguments. To extend the polymer, pick a neighbour (by choosing an integer randomly from $0 \rightarrow 3$ inclusive), and use this to find the displacement using the array of neighbours created above. You should then store this new site as the next monomer in the polymer array, and mark the site just stored in the lattice array as occupied (value 1). **[4]**

   (d) Initialise the arrays with the first monomer at $(50, 50)$, and iterate appropriately to create a polymer of length 100. Plot the resulting structure. **[2]**

   (e) Add a counter to your increment routine to count the number of crossings (when the lattice site chosen is already occupied). Create ten polymers each of length 100, recording and reporting the number of crossings. **[3]**

   (f) We will now use a Monte Carlo process to reduce the number of crossings. Create a new increment function for the polymer, with a while loop for picking the next site. For a given site, accept it with probability 1 if it is empty, or probability set by a variable `thresh` if the site is full (choose another random number from 0 to 1 to do this). If the site is rejected, try again. **[3]**

   (g) For values of `thresh` of 0.5 and 0.1, create ten polymers each of length 100 for each value, and report the number of crossings. Comment on your result. **[2]**

[Part marks]

9. The heat equation in one dimension describes the evolution of thermal energy density, $u(x, t)$, and is given by:

$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2} \tag{4}$$

   (a) Write a python function to calculate the second order finite difference approximation for an array; it should take two parameters, the array and the spacing `dx` and return the second derivative. You should set the second derivatives at the ends of the array to zero. [Hint: you may find `np.roll` to be useful.] **[5]**

   (b) Using a timestep of `dt=0.003` and spatial grid with `dx=0.1`, create appropriate numpy arrays for `x` and `t` covering $0 \leq x \leq 5$, $0 \leq t < 10$. **[2]**

   (c) At `t=0`, $u(x, 0) = 0$. Maintaining $u(5, t) = 0$ and with a driving term $u(0, t) = sin(t)$, write a simple integration scheme for the time evolution of the energy density, using the arrays you have just created (a simple, first-order forward-difference scheme for the time derivative is all that is required), and solve for $u(x, t)$. **[5]**

   (d) Make a surface plot using `ax.plot_surface` for the resulting distribution against x and t. [Hint: remember to import `mplot3d` and to use the extra argument `projection='3d'` in calls to `matplotlib` such as `add_subplot`. The function `np.meshgrid` will be useful; remember the `indexing='ij'` argument.] **[4]**

   (e) The heat equation is easily extended to two dimensions by changing the dependence of the energy density ($u(x, y, t)$) and adding a second term on the right-hand side ($\partial^2 u(x, y, t)/\partial y^2$. Write a new python function to calculate the second order finite difference approximation in either x or y (depending on a flag) for a two dimensional function $u(x, y)$, applying the same zero boundaries as before. [Hint: Remember that the function `np.roll` takes an `axis` argument] **[4]**

PHAS0030/2019 **END OF EXAMINATION PAPER**