CANDIDATE NUMBER:

THE UNIVERSITY OF SUSSEX

F3212

MPhys/Bsc Problem Sheet. Academic Year: 2022-2023

SCIENTIFIC COMPUTING F3212

Complete all of the questions.

To be submitted on CANVAS on Wednesday 19th October 2022, before 12:00 GMT.

The numbers beside the questions indicate the approximate marks that can be gained from the corresponding parts of the questions.

1.**Number representations and truncation errors**

(i) Assuming that signed intergers are represented on your computer with the formula:

$$I = (-1)^s \left(\alpha_n 2^n + \alpha_{n-1} 2^{n-1} + \cdots + \alpha_1 2^1 + \alpha_0 2^0\right) \tag{1}$$
$$n = N - 2 \;\; ; \;\; s = \left\{ \begin{array}{l} 0 \\ 1 \end{array} \right. ; \;\; \alpha_n = \left\{ \begin{array}{l} 0 \\ 1 \end{array} \right. ,$$

where the binary numbers map to $\{s, \alpha_n, \ldots, \alpha_0\}$.
What are the binary values that correspond to the minimum and maximum values of a signed 4-bit integer? What decimal values do these take? [2]

(ii) Define what machine precision means and state the typical values for single and double precision numbers. [1]

(iii) Write down the Maclaurin series expansion for the function $\sinh(x)$. If you only summed this sequnce to N terms what would your estimate of the truncation error be for the $N + 1$th term for this sequence? [2]

(iv) Write a function to evaluate $\sinh(x)$ in a jupyter notebook. This function should rely only on the existence of `math.factorial` routine from the math library. The function should take three arguments `MySinh(x,N=1000,epsi=1.0e-5)`, where the first is the value of x at which we want to evaluate the function and the other two are optional arguments, the first of which sets the maximum number of terms in the series and the second sets the precision with which we want to know the answer. Validate your code using the `numpy.sinh(x)` function evaluated for the values 0.1, 1.0, 10.0, 100.0. [7]

2.**Lagrange interpolation**

The quadratic Lagrange interpolating polynomial for the function $f(x)$ between three data points (x_1, f_1), (x_2, f_2) and (x_3, f_3) over the interval $x \in [x_1, x_3]$ can be written

$$f(x) = f_1 P_1^{(2)}(x) + f_2 P_2^{(2)}(x) + f_3 P_3^{(2)}(x) \tag{2}$$

where the $P_k^{(2)}$ are the 2nd order Lagrange interpolating polynomials, e.g.

$$P_1^{(2)}(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} \, .$$

(i) Following the lecture notes to prove the above formula we needed to invert the matrix

$$M = \begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}$$

Compute the the inverse of this matrix. [4]

(ii) Following the lecture notes, the coefficients of the polynomial a_2, a_1 and a_0 could be found through the matrix product:

$$\vec{a} = \mathbf{M}^{-1} \vec{f} \, .$$

where $\vec{a} = (a_2, a_1, a_0)^T$ and $\vec{f} = (f_1, f_2, f_3)^T$. Give explicit expressions for the a_2, a_1 and a_0. These coefficients are the parameters of the quaratic equation $f(x) = a_2 x^2 + a_1 x + a_0$. [2]

(iii) Write down explict expressions for the polynomials $P_2^{(2)}(x)$ and $P_3^{(2)}(x)$. [2]

(iv) Write down the explicit expressions for the $P_1^{(2)}$, $P_2^{(2)}$ and $P_3^{(2)}$ when $(x_1, f_1) = (0, 1)$, $(x_2, f_2) = (1, 3)$, $(x_3, f_3) = (2, 2)$. [3]

(v) Using a jupyter notebook, write a code that implements the quadratic Lagrange interpolation of these three points. Plot your results to prove that your interpolating function model line goes through the points. [5]

(vi) By analogy with the quadratic case, write down the form of the cubic interpolating polynomial that interpolates between the four points (x_1, f_1), (x_2, f_2), (x_3, f_3) and (x_4, f_4) over the interval $x \in [x_1, x_4]$. [2]

z	$\chi(z\|\Omega_\mathrm{m} = 0.25)$ [h^{-1}Mpc]	$\chi(z\|\Omega_\mathrm{m} = 0.3)$ [h^{-1}Mpc]
1.0000000e-01	4.3327843e+02	4.3490824e+02
2.0000000e-01	8.4565604e+02	8.5199526e+02
3.0000000e-01	1.2367089e+03	1.2504843e+03
4.0000000e-01	1.6065104e+03	1.6300282e+03
5.0000000e-01	1.9555308e+03	1.9906572e+03
6.0000000e-01	2.2845323e+03	2.3327094e+03
7.0000000e-01	2.5944725e+03	2.6567584e+03
8.0000000e-01	2.8864243e+03	2.9635438e+03
9.0000000e-01	3.1615125e+03	3.2539121e+03
1.0000000e+00	3.4208678e+03	3.5287680e+03

Table 1: Redshift–distance data for two cosmological models.

3.**scipy interpolation**

In modern cosmological surveys of galaxies the distances to galaxies can be determined from measuring the redshift z – this is the stretching of the wavelength of light due to the expansion of spacetime. The redshifts are then turned into distances using a distance–redshift formula $\chi(z)$. This depends on the cosmological model and its parameters (in particular Ω_m the matter density) and to compute it normally requires us to solve a complicated numerical integral. Table 1 shows in the first column the redshift z, and in the next two columns, the corresponding distances in a low-density and slightly higher density universe, respectively. The distance units are in h^{-1}Mpc where h is the dimensionless Hubble parameter, M stands for mega and where pc is 1 parsec $\sim 3 \times 10^{16} m$.

(i) Using a jupyter notebook, write a code to interpolate the distance–redshift data for the two cosmological models. Use the `scipy.interpolate` function to perform linear interpolation with the routine `interp1d`. [5]

(ii) Make a plot that compares the results for the two models, using your linear interpolation routine from above. [2]

(iii) Repeat your analysis from above but this time with scipy's cubic spline code. [1]

(iv) Evaluate the interpolation functions at $z = \{0.15, 0.25, 0.35, 0.45\}$ and plot these points on your graphs for the two models. [2]

— End of paper —