COMPUTATIONAL PHYSICS EXAMINATION

05/06/2020

BIHAN BANERJEE

DAA, TIFR

1.

R ≡ Row

$$\begin{pmatrix} 4 & 1 & 2 \\ 2 & 4 & -1 \\ 1 & 1 & -3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 9 \\ -5 \\ -9 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 1 & 0.25 & 0.5 \\ 2 & 4 & -1 \\ 1 & 1 & -3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2.25 \\ -5 \\ -9 \end{pmatrix}$$

[Dividing first row by 4
$R_1 \rightarrow R_1/4$]

$$\Rightarrow \begin{pmatrix} 1 & 0.25 & 0.5 \\ 0 & 3.5 & -2 \\ 0 & 0.75 & -3.5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2.25 \\ -9.5 \\ -11.25 \end{pmatrix}$$

$\begin{bmatrix} R_2 \rightarrow R_2 - 2R_1 \\ R_3 \rightarrow R_2 - R_1 \end{bmatrix}$

$$\Rightarrow \begin{pmatrix} 1 & 0.25 & 0.5 \\ 0 & 1 & -0.57 \\ 0 & 0.75 & -3.5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2.25 \\ -2.71 \\ -11.25 \end{pmatrix}$$

$[R_2 \rightarrow R_2/3.5]$

$$\Rightarrow \begin{pmatrix} 1 & 0.25 & 0.5 \\ 0 & 1 & -0.57 \\ 0 & 0 & -3.93 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2.25 \\ -2.71 \\ -13.28 \end{pmatrix}$$

$R_3 \rightarrow R_3 - (R_2 \times 0.75)$

In each step, we have rounded off upto two decimal places,

$$\therefore\ x_3 = +\frac{13.28}{3.93} = 3.38$$

$$x_2 - 0.57x_3 = -2.71$$

$$\Rightarrow x_2 = -2.71 + 0.57x_3 = -0.78$$

$$x_1 + 0.25x_2 + 0.5x_3 = 2.25$$

$$\Rightarrow x_1 = 2.25 - 0.25x_2 - 0.5x_3$$

$$= 0.75$$

Therefore, answer we will get from this machine:

$$(x_1, x_2, x_3) = (0.75, -0.78, 3.38)$$

Exact result (from inspection) : $(1, -1, 3)$

2.

(a) In Python: numpy. fft. fft (performs discrete fourier transformation of a sample)

* C: Library fftw3.h

(b) Python: numpy. linalg. qr

(c) numpy. random. lognormal ( ) [In the argument, number of points to be drawn, can be specified.]

(d) scipy. integrate. solve_ivp (.., ..., method = 'RK78', ...)

(e) numpy. linalg. svd

(f) If the PDF is normal distribution then,

scipy. stats. multivariate_normal.

(g) scipy. integrate. odeint (, ..., ..., h_0 = 'initial step size',

h_{max} = 'Maximum step-size', h_{min} = 'Minimum step size', ...)

(h) np. random. rand (9)

(i) scipy. integrate. solve_bvp

(j) numpy. linalg. eig

3.

Suppose A is an $n \times n$ tridiagonal matrix.

And we want to solve the system of linear equations:

$$AX = \cancel{b}g \quad \text{[Instead of } b \text{ we are writing } g \text{ just for notational convenience]}$$

If we write it in matrix form; we get:

$$\begin{pmatrix} b_1 & c_1 & 0 & \cdots & & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 \\ 0 & & \ddots & & & \vdots \\ \vdots & & & \cdots & b_{n-1} & c_{n-1} \\ 0 & & & \cdots & b_n & c_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix} \qquad (1)$$

Suppose, we want to solve it by Gauss-elimination method.

* First step is to make first entry of A, (which is b_1) 1.

  i.e. we need to divide first row of A and g by b_1.

  But first row of A only has two non-zero entries.

  And g_1 is non-zero in general.

  Therefore, we only need to perform 3 divisions.

  And the matrix we get is:

$$\begin{pmatrix} 1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \cdots & 0 \\ 0 & a_3 & b_3 & c_3 \cdots & 0 \\ \vdots & & \ddots & & \\ 0 & & & \cdots & c_n \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix} \qquad (2)$$

Where we have re-labelled $c_1 = c_1' = \frac{c_1}{b_1}$ $\quad g_1 = g_1' = \frac{g_1}{b_1}$

etc.

Now the next step is to make all elements below (A_{11}) in first column zero.

But our task is reduced because $(n-2)$ elements are already 0.

we have to make only one element, $(A_{21} = a_2)$ zero.

Therefore, we multiply the first row by a_2 (3 operations) and then subtract from second row (3 operations again) and we obtain:

$$\begin{pmatrix} 1 & c_1 & 0 & \cdots & \cdots & 0 \\ 0 & b_2 & c_2 & \cdots & \cdots & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 \\ 0 & & \ddots & \ddots & \ddots & \\ \vdots & & & \ddots & \ddots & \\ 0 & & & & & c_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix} \quad \cdots\cdots (3)$$

Therefore, to reach ~~to~~ (3) from (1), we have made:

$p =$ 3 divisions + 3 multiplication + 3 subtraction

$=$ 9 operations.

And to get our desired result, we have to repeat it n times.

$\therefore$ ~~Indeed,~~ Total number of operations $\sim np \sim 9n$.

$\therefore$ Indeed, it is $O(n)$ operation.

5.

The essential three criteria would be:

(1) ~~The result produce~~

(1) Accuracy of the result produced using the library.

(2) Time taken to compute the result using the library.

(3) And of course, how much user friendly the library is. In addition how much freedom is given to the user is also important.

7. A linear congruential generator is defined as :

$$X_n = (a X_n + c) \% m$$

where a, c, m are fixed parameters.

* Suppose, we choose, $a = 3$, $c = 3$, $m = 4$

And our choice of seed $X_0 = 1$

Then, $X_1 = (3+3) \% 4 = 2$

$$X_2 = (3X_1 + 3) \% 4 = (6+3) \% 4 = 1$$

$\therefore$ We get back the seed just after two steps.

However, if a, c, m are large integers, say,

$a = 10^{10}$, $c = 10^4$, $m = 2^{32}$, a seed does not ~~repeat back~~ appear again, unless, $n \sim m$.