End-Sem Exam Computational Physics.

Siddhartha Karmakar

1.)

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

deviding row 1 by 4,

$$\begin{bmatrix} 1 & 0.25 & 0.50 \\ 2 & 4 & -1 \\ 1 & 1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2.25 \\ -5 \\ -9 \end{bmatrix}$$

doing (row(2)) − (row 1) * 2, & (row 3) − (row 1)

$$\begin{bmatrix} 1 & 0.25 & 0.50 \\ 0 & 3.50 & -2.00 \\ 0 & 0.75 & -3.50 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2.25 \\ -9.50 \\ -11.25 \end{bmatrix}$$

doing (row 2) / 3.50,

$$\begin{bmatrix} 1 & 0.25 & 0.50 \\ 0 & 1 & -0.57 \\ 0 & 0.75 & -3.50 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2.25 \\ -2.71 \\ -11.25 \end{bmatrix}$$

doing (row 3) − (row 2) * 0.75

$$\begin{bmatrix} 1 & 0.25 & 0.50 \\ 0 & 1 & -0.57 \\ 0 & 0 & -3.07 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2.25 \\ -2.71 \\ -9.22 \end{bmatrix}$$

So,

$$x_3 = \frac{9.22}{3.07} = 3.00$$

$$x_2 = -2.71 - (-0.57 \times 3.00) = -1.00$$

$$x_1 = 2.25 - 0.5 \times 3.00 - 0.25 \times (-1) = 1.00$$

So, the solution is

$x_1 = 1.00$
$x_2 = -1.00$
$x_3 = 3.00$

2) (a) numpy.fft.fft

(b) numpy.linalg.qr

(c) np.random.lognormal (mean, sigma, size)
can take size = 1000000

(d) gsl_odeiv2 , which contains steptype gsl_odeiv2_step_rk8pd for solving initial value ODE using 8^{th} order Runge-Kutta.

(e) numpy.linalg.svd

(f) numpy.random contains many pdf that can generate random sampling of any size required. For example

numpy.random.rand (n, 548)

will give n , 548-dimensional random number from the uniform pdf.

(g) gsl_odeiv2_control

(h) gsl_monte_plain_integrate

(i) scipy.integrate.odeint

(j) numpy.linalg.eig

(3) A tridiagonal matrix A is of the form

$$A = \begin{bmatrix} a_{11} & a_{12} & & & & 0 \\ a_{21} & a_{22} & a_{23} & & & \\ & a_{32} & a_{33} & a_{34} & & \\ & & a_{43} & \ddots & \ddots & \\ & 0 & & \ddots & & a_{n-1,n} \\ & & & & a_{n,n-1} & a_{nn} \end{bmatrix} \; \& \; b = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

We consider Gauss ellimination method to count the order of number of steps required.

For the first row,

3 div. to get the first element as 1.

3 mult + 3 sub to make a_{21} zero.

These 9 no. of operations will be repeated for $(n-1)$ rows (except the last row).

So there will be $9.(n-1)$ operations to get the form,

$$\begin{bmatrix} 1 & a_{12} & & & 0 \\ & 1 & a_{23} & & \\ & & 1 & a_{34} & \\ & & & \ddots & a_{n-1,n} \\ & 0 & & & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

Now in the back substitution method, (5)
to get, x_n

1 div.

to get, x_{n-1}

1 mult + 1 sub.

And
(1 mult + 1 sub) will be required for
rest of x's.

So in back substitution, in total,

$(n-1) \times (2) + 1$ operations.

So overall in finding the solution
total number of operations,

$$9(n-1) + 2(n-1) + 1$$

$$= 11n - 10$$

$\Rightarrow O(n)$ operations required.

4)

(e) The pdf of the uniformly generated numbers is like a constant function.
Thus, when we take the fourier transform of these numbers, we expect to get a ~~dirac-~~ Dirac-delta like functionz
That is what we are getting for our code.
As, we are getting large $\tilde{f}(k)$ only arround $k=0$, in the power spectrum also we expect to get a peak arround $k=0$.

5) Three criterias:

(i) Language and compatibility:

I will consider whether I have the compiler for the Language in which the library software is written and also whether my operating system and hardware configuaration will be able to run the library software.

(ii) Ease of us:

Will check how simple or complicated it is to use the library commands and whether there is a good enough documention available for the users.

(iii) Time efficiency:

Will compare the time taken to perform the same task by the different libraries.

(6). I have compared the solution with analytical one, that is

$$y_1 = \frac{1}{3}\left(2x - e^{-100x} + 2e^{-x}\right)$$

$$y_2 = \frac{1}{3}\left(-x + 2e^{-100x} - e^{-x}\right)$$

to check whether my solution is correct or not. But, to argue whether the code solution is correct or not, just by the code solution itself, we can atleast compare the boundary conditions which we are getting correct.

7.) With modulus $(m) = 10$
multiplier $(a) = 7$
increment $(c) = 7$ and
seed $(X_0) = 7$

the random number generated by Linear cong. $X_{i+1} = (X_i * a + c) \pmod{m}$

repeats after some numbers. i-e,

7, 6, 9, 0, 7; 6, 9, 0 ...

This is an example where the seed appears again.

Now the number X_{i+1} is always less than ~~or equal to~~ m. So if we take the seed $X_0 > m$, the seed won't reappear after the first appearance

Example. with, $m = 10$, $a = 7$, $c = 7$ but $X_0 = 15$, we get,

15, 2, 1, 4, 5, 2, 1, 4, 5, ...