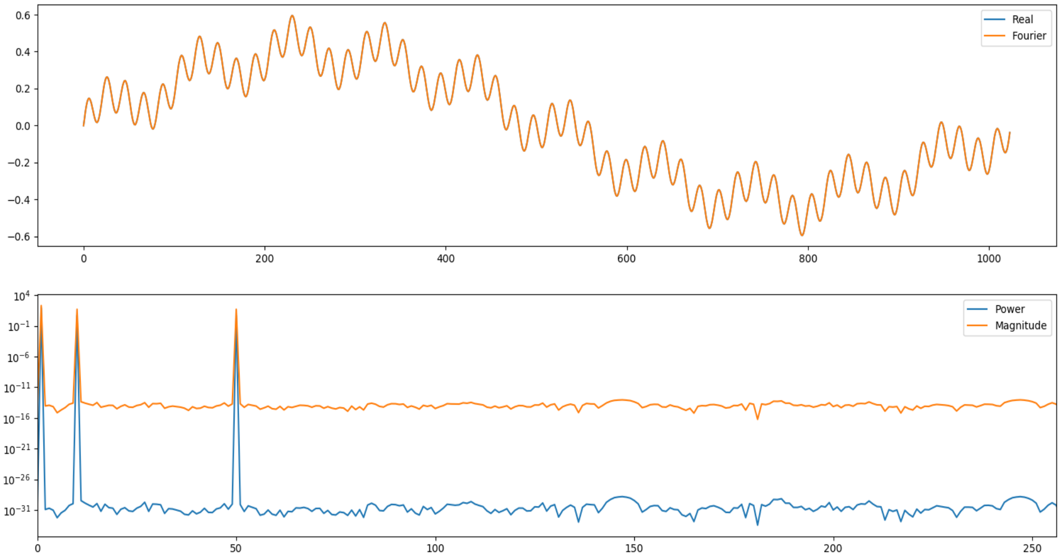
Final exam Solutions: *Hirak Bandyopadhyay*

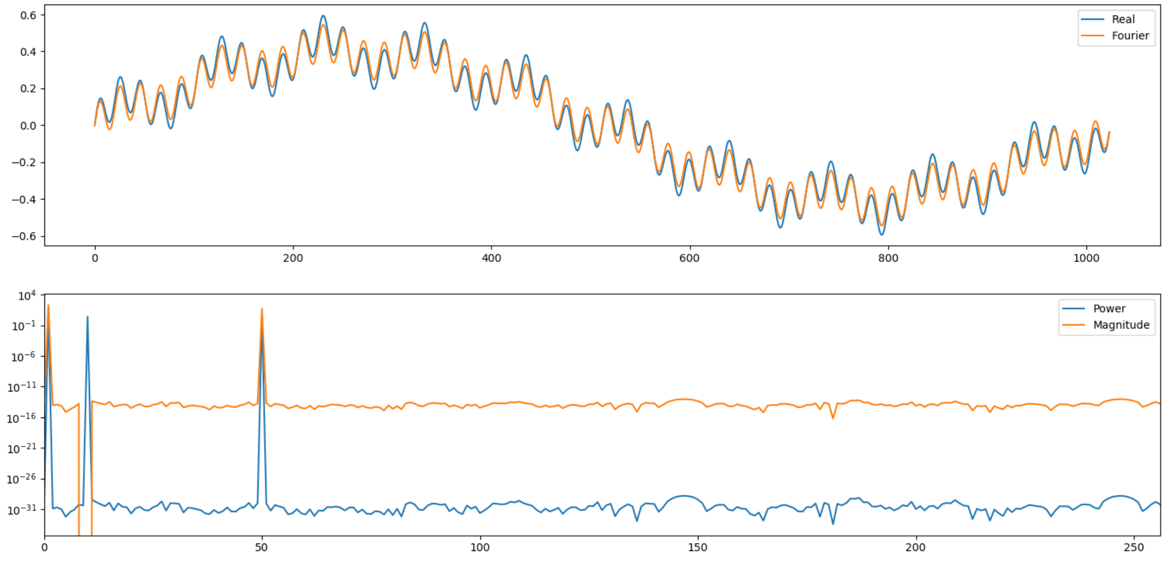
Problem 1:

a) In the first part of this problem we are required to plot the given function in time domain. So I create a program which takes the function and evaluates it in matplotlib to give us the plotted function. I have also introduced an option for padding if we do not have n = 1024. The plot is attached herewith.



b) Now, this plot is also valid for the second problem as we are required to find the fourier transform of the function. As we see that there are three distinct frequencies in our fourier plot. As the given function has f0 = 25Hz, f1 = `0Hz and f2 = 1Hz, we can see that in the fourier domain there are peaks at 1, 10 and 50. The peak at 50 is simply because we have a form of sin(wt)cos(wt) which is clearly sin(2wt). So we do see the frequency as 2*25 = 50Hz.

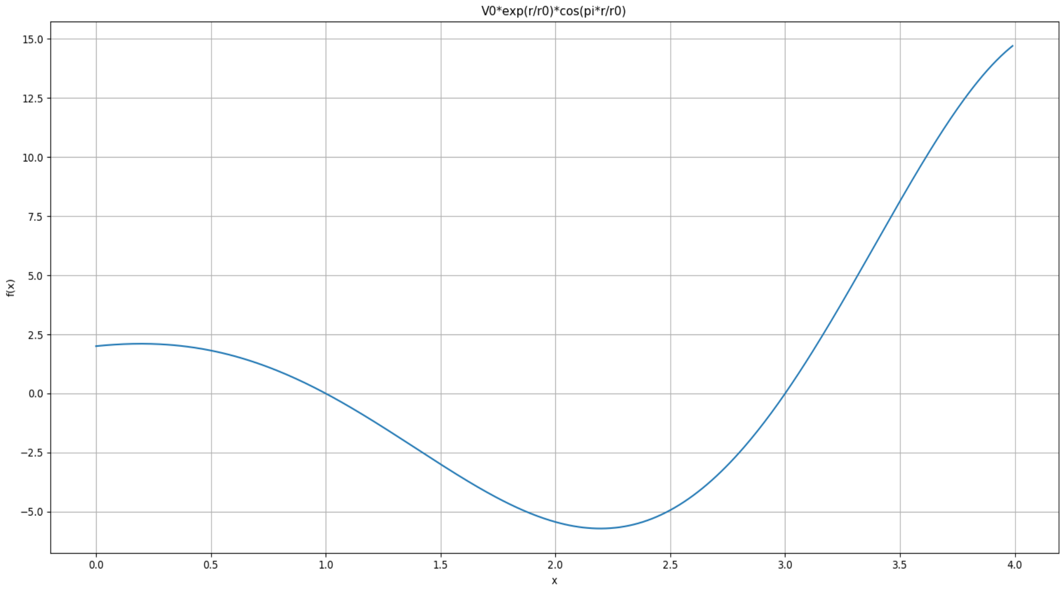
c) Now in part c we are required to filter wavenumbers corresponding to f1 = 10Hz. So I setup the range given to smooth the peak at 10 Hz using the smoothening, and then replot the ifft over the actual function to see that f1 has been removed from the function.



Here we can easily see that the noise around f1 = 10 is removed and the new spectrum is plotted over the old one.

Problem 2:

a) In this part we are required to plot the given function using matplotlib. The plot is attached herewith.



Now, we can clearly see that the function has one local minima and one local maxima in between and one local minima and one local maxima at two ends. Also it has two roots, presumably around 1 and 3.

b) Now, for root finding I use simple search as well as bisection search. For that we use the root_finding.hpp hearer file and then we define the function and ask the user for their guesses. The terminal output for the first root in this range is given below.

```
Algorithms for root of y = V0*exp(r/r0)*cos(M_PI*r/r0)

 -----------------------------------------------

 1. Simple search

 Enter initial guess x_0, step dx, and desired accuracy: 0

0.1

0.001

 answer = number of steps for simple root finding algorithm = 17

1


 2. Bisection search

 Enter bracketing guesses x_1, x_2, (one root must lie between x1 and x2) and desired accuracy:
0

1.5

0.001

 Answer = number of steps for bisection algorithm = 11

1.00012
```

Now, for the second root, we give guesses accordingly.

```
Algorithms for root of y = V0*exp(r/r0)*cos(M_PI*r/r0)

 -----------------------------------------------

 1. Simple search

 Enter initial guess x_0, step dx, and desired accuracy: 2.1

0.1
```

```
0.001

 answer = number of steps for simple root finding algorithm = 21

2.99844


 2. Bisection search

 Enter bracketing guesses x_1, x_2, (one root must lie between x1 and x2) and desired accuracy:
2.5

4.2

0.001

 Answer = number of steps for bisection algorithm = 11

3.00012
```

We can clearly see that both the searches give us a good agreement to our assumption that the roots are 1 and 3.

c) In the final part of this problem we are going to find minima and maximas of the function. In order to do so, we create another macro named final_2c.py, and here we define the function and the negative of this function. Now, we use the scipy.optimize.minimize function and we put in the values to find the maxima and minima. The program prompts user for the guess and the tolerance and then gives us the closest minima and maxima.

```
Enter starting point: 1

1

1.0

 Enter desired accuracy: 0.001

minimize

[ 2.19614828] -5.71437638873

maximize

[ 0.19623161] 2.1022015909
```

Now, we can see that the minima is at 2.19614828 with a function value -5.71437638873. Also we see that the maxima is at 0.19623161 and the function value is 2.1022015909. Now, to get the maxima of the other side, we need to give closer values.

```
Enter starting point: 3.5
3.5
3.5
 Enter desired accuracy: 0.001
minimize
[ 2.1961725] -5.71437639856
maximize
[ 3.99956713] 14.7749106215
```

Clearly, this time the function gives us the maxima at 3.99956713 which is basically at 4.

```
Enter starting point:0. 1
0.1
0.1
 Enter desired accuracy: 0.001
minimize
[ 0.003275989] 2.0013274459
maximize
[ 0.19623161] 2.1022015909
```

And this way we get all the maxima and minima in the range specified.

For global scenario, the function has many maxima and minima, but the functional value diverges very fast.

Problem 3:

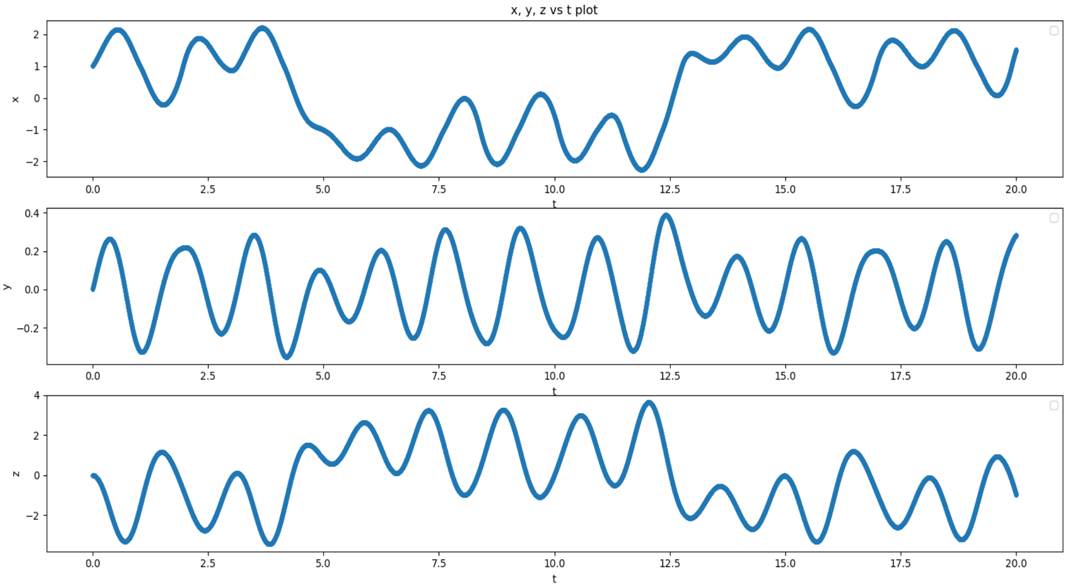
a) In this problem we are required to solve for the Chua's circuit. This is a complex system and can be represented by the equations given.

$$\frac{dx}{dt} = \alpha[y - x - f(x)]$$

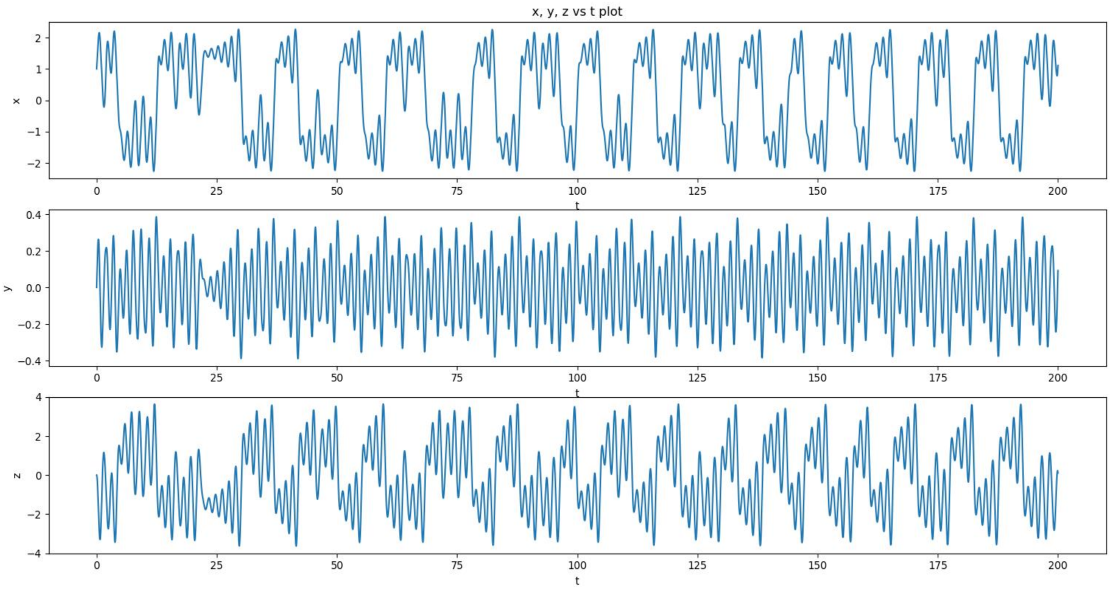
$$\frac{dy}{dt} = [x - y + z]$$

$$\frac{dz}{dt} = -\beta y$$

I create a program which takes the definitions of each functions corresponding to $\frac{dx}{dt}, \frac{dy}{dt}$ and $\frac{dz}{dt}$ and name them p,q and r. Now, I use the rk4 scheme to solve simultaneously for these three first order differential equations. When solved in loop for a given time step, we recoed the evaluations in arrays and then use matplotlib to plot the parameters as a function of time. The initial conditions given are, x(0)=1, y(0)=z(0)=0.

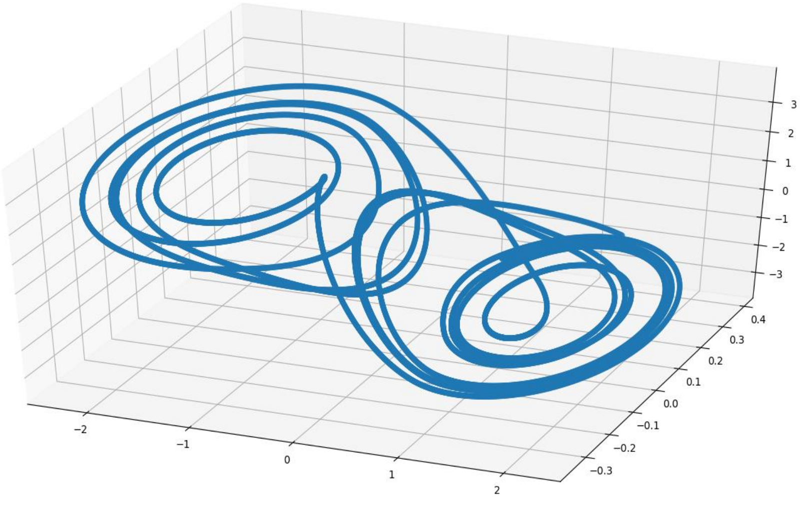


This is for small time. In a long run we see interesting features.



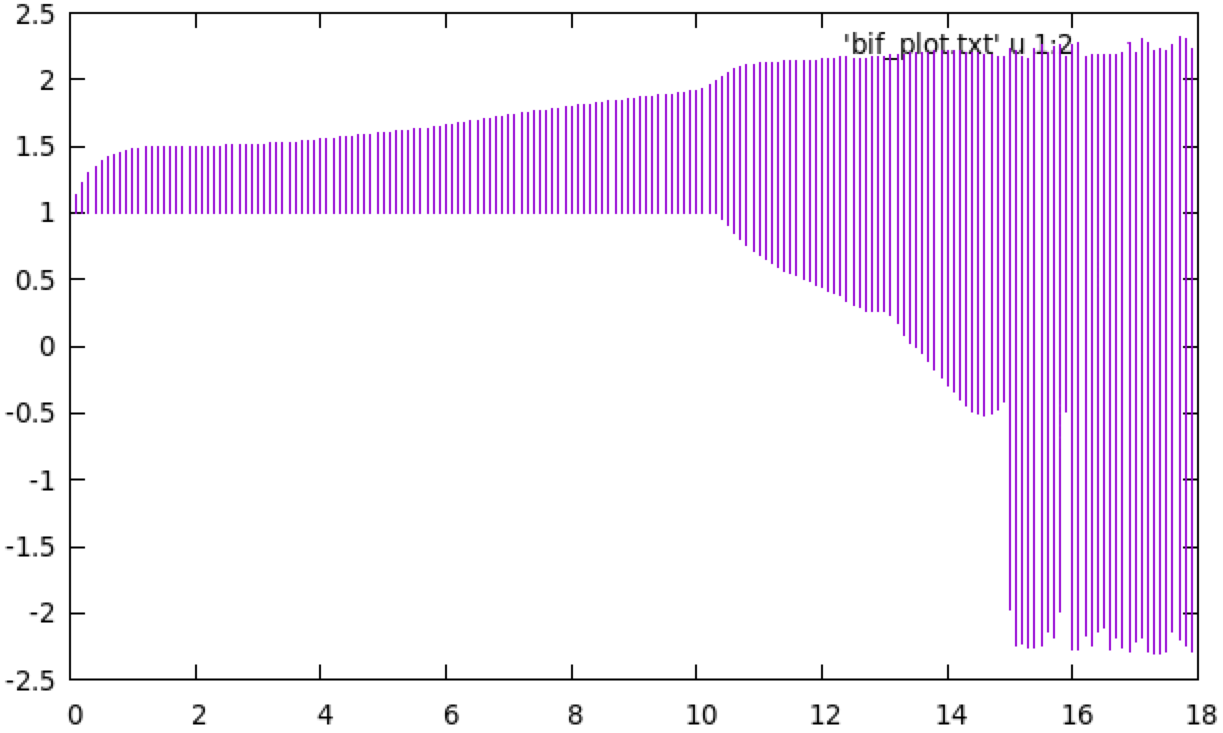
We see period doubling is occurring.

b) We are now required to plot x,y,z simultaneously, so we need 3D plot. The plot is given below.



We see that there are two attractors in the phase space and longer we observe the evolution, more pronounced the orbits become.

c) In the final part we are required to plot the bifurcation diagrams. We can clearly see that for a set of given initial conditions, the values of alpha and beta can create completely chaotic behavior. For example, keeping beta fixed, if we change the value of alpha, we see that the period goes from one to two to four to completely chaotic behavior. We can easily do the same with beta keeping alpha fixed. So we can say certain combinations of values of alpha and beta for given initial conditions will create a bifurcation diagram in periods of the functions. The diagram is given below.



I am really very sorry that this looks awful. I tried but this is the best I could get without my computer giving up!

Problem 4:

a) We are given the elements of vector u at some state k to be evolved to u_{k+1}. Now, the matrix equation will look like

$$u_{k+1} = Ku_k$$

Where the matrix K is a 9*9 matrix and it is a symmetric matrix.

$$K = \begin{pmatrix} 3 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 3 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 3 \end{pmatrix}$$

Clearly we can see that the matrix is symmetric. So this is indeed the symmetric matrix equation as $A\vec{x} = eS\vec{x}$.

b) For the second part of this problem, we have to solve for the eigenvalues and eigenvectors. Now, we create program which uses the cpt directory and uses the solve_eigen_symmetric function or solve_eigen_generalized function to solve for the eigenvalues and eigenvectors. So, I create a 9*9 matrix for K and a 9*9 identity matrix for solve_eigen_generalized and then solve for those. The final result is given below.

```
K =

|  3.00        -1.00  -1.00   0.00   0.00   0.00   0.00   0.00   0.00 |

| -1.00         3.00  -1.00  -1.00   0.00   0.00   0.00   0.00   0.00 |

| -1.00        -1.00   3.00  -1.00  -1.00   0.00   0.00   0.00   0.00 |

|  0.00        -1.00  -1.00   3.00  -1.00  -1.00   0.00   0.00   0.00 |

|  0.00         0.00  -1.00  -1.00   3.00  -1.00  -1.00   0.00   0.00 |

|  0.00         0.00   0.00  -1.00  -1.00   3.00  -1.00  -1.00   0.00 |

|  0.00         0.00   0.00   0.00  -1.00  -1.00   3.00  -1.00  -1.00 |

|  0.00         0.00   0.00   0.00   0.00  -1.00  -1.00   3.00  -1.00 |
| 0.00        0.00   0.00   0.00   0.00   0.00  -1.00  -1.00   3.00 |
Eigenvalues =
|  4.00|
|  3.24|
|  3.41|
|  4.34|
|  4.89|
|  5.00|
|  2.09|
|  0.59|
| -0.57|


Eigenvectors =
| -0.00      -0.14  -0.56  -0.47   0.09   0.27   0.47   0.34  -0.18 |
| -0.50       0.35   0.12   0.37   0.27  -0.27   0.32   0.41  -0.27 |
|  0.50      -0.31   0.12   0.26  -0.44  -0.27   0.11   0.41  -0.36 |
|  0.00       0.37   0.40  -0.28  -0.16   0.53  -0.29   0.24  -0.41 |
| -0.00      -0.50   0.00   0.04   0.63   0.00  -0.40   0.00  -0.44 |
|  0.00       0.37  -0.40  -0.28  -0.16  -0.53  -0.29  -0.24  -0.41 |
| -0.50      -0.31  -0.12   0.26  -0.44   0.27   0.11  -0.41  -0.36 |
|  0.50       0.35  -0.12   0.37   0.27   0.27   0.32  -0.41  -0.27 |
| -0.00      -0.14   0.56  -0.47   0.09  -0.27   0.47  -0.34  -0.18 |
```

These are the eigenvalues. The eigenvectors are the vertical columns of the last Eigenvector matrix. We can see that there are 9 different eigenvalues and eigenvectors.

c) Now, we need to observe the state as k gets large. For that, first we need to see what is going on. Suppose we have an equation,

$u_{k+1} = Ku_k = \lambda u_k$
$u_{k+2} = Ku_{k+1} = K.Ku_k = \lambda^2 u_k$
$u_{k+3} = Ku_{k+2} = K.K.Ku_k = \lambda^3 u_k$ ….

Now, in order to solve for large k, we have to multiply the matrix k times and then see how the eigenvalues are.

In order to do so, create an inverse of K, and then use the Gauss-Jordan elimination method to solve for the final state given the initial state. Now, each time we have a final state, that becomes the initial state of the next iteration and then it computes the final state again and this can be carried for large values of k in a loop. We do exactly the same, and we observe that for large values of k, the set of eigenvalues are very large for negative and positive solutions, and the eigenvectors show two different behavior for odd and even k. For odd k, it shows that the final state is almost similar to the initial state superposition of eigenvalues and eigenvectors because of the symmetric eigenvalues, and for even k, we see that the final state is a superposition of large eigenvalues and the eigenvectors.

Note: For Some problems, we need the cpt directory, which I did not include here.