GUI based
Transfer Function Analysis Package

Mini Project Report

Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

In

Electrical and Electronics Engineering

By

NAME	ROLL NO
ANNEPU SRAVYA	B130910EE
GORTHI HARSHITHA	B130810EE
NIVEDITA SURESH	B130211EE
SHAHEERA BANO	B130186EE

Under the guidance of

Mr. K S Sureshkumar

Department of Electrical and Electronics Engineering

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT

MAY 2016

CERTIFICATE

*This is to certify that the report entitled "***GUI based TRANSFER FUNCTION ANALYSIS PACKAGE***" is a bona fide record of the Mini project done by* ANNEPU SRAVYA(B130910EE), GORTHI HARSHITHA(B130810EE), NIVEDITA SURESH(B130211EE) and SHAHEERA BANO(B130186EE) *towards the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology In Electrical and Electronics Engineering from National Institute Of Technology Calicut for the year 2016 under my guidance and supervision.*

Mr. K S SURESHKUMAR
Project Guide
EED

Dr. JEEVAMMA JACOB
Professor and Head
EED

Place: Nit Calicut

Date: 9/5/2016

ACKNOWLEDGEMENT

We express our sincere gratitude to our guide Mr. K S Sureshkumar, Associate Professor, Department Of Electrical And Electronics Engineering, for his guidance and support throughout this endeavor. We thank Dr. Jeevamma Jacob, Head of the Department, for providing all the facilities required for the project in the department. We would like to extend our sincere thanks to Dr. Elizabeth P Cheriyan, Associate Professor, EED and the mini project coordinator for giving us an opportunity to work in this project area.

ABSTRACT

This project, is a GUI based transfer function analysis package (TFAP) done in Python. It takes in the values of poles, zeroes and the gain factor of a transfer function in a GUI format. From the given inputs it estimates the frequency plot, both absolute and bode plot, and the time response which includes the step and impulse response. It can also be used for estimating the stability of the system.

This software gives the user, better understanding on how the frequency and time response depends on the poles, zeroes and the gain factor.

Frequency Response curves are often used to indicate the accuracy of electronic components or systems. When a system or component reproduces all desired input signals with no emphasis or attenuation of a particular frequency band, the system or component is said to be “flat” or to have a flat Frequency Response curve. So it is very important to study the frequency and time response of a system to ensure that the system is stable and produces the required output.

CONTENTS

LIST OF FIGURES

Chapter 1 INTRODUCTION TO TFAP

1.1 INTRODUCTION

A transfer function is the ratio of the output of the system to the input of the system in Laplace domain considering its initial conditions and equilibrium point to be zero. The frequency domain transfer function is a complex function of w, hence it can be separated into magnitude function and phase function. Now the magnitude and phase functions will be real functions of w and they are called frequency response. Similarly the time response of the system is the output of the system as a function of time. A GUI based transfer function analysis package is created for understanding the frequency and time responses.

Graphical User Interface (GUI) is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators. Programming in GUI needs a language as a platform. The software adapts Python to develop a code which gives the desired output. Using GUI programming in Python, a package is designed to obtain the frequency and time response of a system by computing its transfer function. Python provides various options for developing graphical user interfaces (GUIs) out of which one is Tkinter. Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications.

The program(TFAP) makes use of this interactive feature of a Graphic User Interface to help users study how addition of poles and zeroes affect the response of the system with respect to time and frequency.

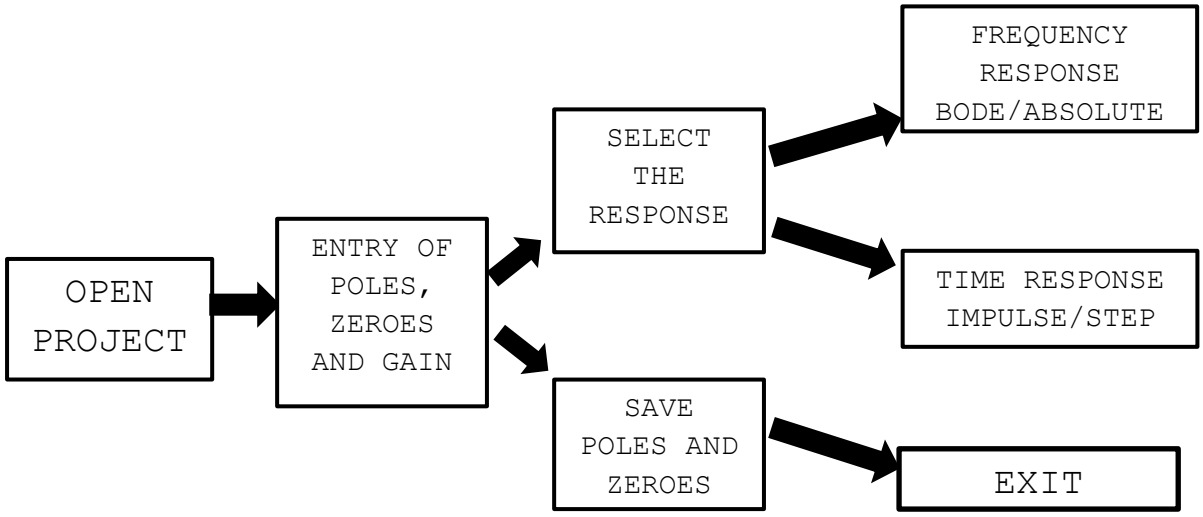
1.2 METHODOLOGY AND SALIENT FEATURES OF THE TFAP

The software provides a platform to the users to study the frequency and time response of a transfer function. The software takes in the values of poles and zeroes of a transfer function and computes the frequency and time responses. The frequency response is computed by substituting different values of w, upon a suitable range selection, in the transfer function and plotting them by joining adjacent points with a straight line. The time response is computed by calculating the Laplace inverse of the transfer function and substituting different values of t, upon a selection of suitable range, in the transfer function and plotting them by joining adjacent points with straight lines.

The entry of poles and zeroes are done graphically. The entered poles and zeroes are displayed on the canvas. The user can specify the gain factor or the software takes the default value as 1. It gives the user a provision to add more poles and zeroes once the entry is complete on a click of the DONE button. The required plots are plotted. The desired portion of the plots can be scrolled into as per the user's requirement. The system stability can also be predicted. Ample help and guidance is provided to the user through help buttons so that the software may be very user friendly. Both frequency response and time response of a system can be analysed simultaneously. The entered poles and zeroes can be saved to a file for future references.

Chapter 2 CODING THE TFAP

2.1 Block diagram representation of TFAP



2.2 Main Window

2.2.1 Menubar

The Menubar in the main window contains options which can be torn-off from the main Menu bar. It contains options such as File, Frequency Response, Time Response, Help and Exit. The command for Menu bar is given in python using the Menu command.

```
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New",command=newcd)
filemenu.add_command(label="Save",command=savefn)
filemenu.add_separator()
filemenu.add_command(label="Exit",command=close_window)
menubar.add_cascade(label="File", menu=filemenu)
freqmenu = Menu(menubar, tearoff=0)
freqmenu.add_command(label="Absolute Plot",command=freqop)
freqmenu.add_command(label="Bode Plot",command = freqbodeop)
menubar.add_cascade(label="Frequency Response", menu=freqmenu,state='disabled')
timemenu = Menu(menubar, tearoff=0)
timemenu.add_command(label="Step Response",command=time_step)
timemenu.add_command(label="Impulse Response",command=time_imp)
menubar.add_cascade(label="Time Response", menu=timemenu,state='disabled')
menubar.add_cascade(label="Help",command=helpmenu)
menubar.add_cascade(label="Exit",command=close_window)
root.config(menu=menubar)
```

FILE: File menu contains provision to restart, save and exit the file.

The New command causes the program to terminate its execution and call itself. The root.destroy command is used for this.

```
def newcd() :
    root.destroy()
    import tfap
```

The Save menu invokes the save function savefn() that creates a sub window which accepts a file name through an entry box and then invokes another function saving() which stores the poles and zeroes entered into a file in textfile format.

```
def savefn () :
    global sub4
    global name
    global e3
    sub4=Toplevel(root)
    sub4.title("wm min/max")
    sub4.resizable(0,0)
    sub4.title('SAVE')
    sub4.minsize(300,50)
    l3 = Label(sub4,text='FILE NAME : ')
    l3.grid(column=0,row=0)
    l3.pack()
    e3 = Entry(sub4)
    e3.grid(column=1,row=0)
    e3.pack()
    b3= Button(sub4,text='SAVE',command=saving)
    b3.grid(column=1,row=1)
    b3.pack()
    sub4.mainloop()



def saving() :
    global sub4
    global name
    global e3
    fo = open('%s.txt'%(e3.get()) , 'w')
    for g in range (k) :
        fo.write( str(rnum[g]) + ' ' + str(inum[g]) + '\n' )
    fo.write( '\n' )
    for m in range (l) :
        fo.write( str(rden[m]) + ' ' + str(iden[m]) + '\n'
    fo.close()
    sub4.destroy()
    flag=1
```

FREQUENCY RESPONSE: The frequency response command contains two options, to display the absolute plot or the bode plot. This button is initially deactivated and become active only upon the selection of the DONE button. The options in the FREQUENCY RESPONSE menu will directly invoke the function freqop() and freqbodeop(). The detailed explanation for this will be given in the future sections.

TIME RESPONSE: The time response command contains two options, to display the Impulse Response plot or the Step Response plot. This button is initially deactivated and become active only upon the selection of the DONE button. The options in the TIME RESPONSE menu will directly invoke the function time_imp() and time_step(). The detailed explanation for this will be given in the future sections.

HELP: The Help button invokes a function which creates a canvas which contains the help text for the easy use and understanding of the software to the user.

```
def helpmenu() :
    sub3 = Toplevel(root)
    sub3.title("wm min/max")
    sub3.title('HELP')
    sub3.resizable(0,0)
    scrollbar = Scrollbar(sub3)
    scrollbar.pack( side = RIGHT, fill=Y )
    canvash=Canvas(sub3,width=500,height=300,bg='#eeeee0',
    yscrollcommand=scrollbar.set,scrollregion=(0,0,600,1200))
    canvash.create_text(250,620,text= var)
    canvash.pack()
    scrollbar.config( command = canvash.yview,orient= VERTICAL )
    sub3.mainloop()
```

EXIT: The exit button destroys the existing root window thereby terminating the program.

```
def close_window ():
    root.destroy()
```

2.2.2 The Canvas

The main canvas window is completely GUI based and uses several features like button click and motion control for the entry of the poles and the zeroes. The Bind feature of Tkinter enables us to link the keyboard keys with a user defined function. A coordinate plane representing the s-plane allows the user to choose the required pole and zero by either moving the mouse pointer to the desired location or using UP and DOWN arrows for magnitude adjustments and LEFT and RIGHT arrow keys for angle adjustments. For this a movable circle and line is created in the canvas representing the magnitude and angle. The coordinates of the points are immediately displayed on the canvas. Selection of the poles and zeroes are achieved by ENTER key or RIGHT click of the mouse for the pole entry and SHIFT or LEFT click of the mouse for the zero entry. Once a selection of a pole or zero is made the DONE button at the right bottom of the main window gets activated. Upon the selection of this button, the circle and line disappears and the Frequency Response, Time Response and ADD POLES AND ZEROES buttons are activated. Using the ADD POLES AND ZEROES button, the user can add more poles and zeroes.

GAIN ENTRY BOX: A gain entry box is provided at the bottom of the main window to enter the gain while calculating the frequency response. If nothing is entered, a default value of 1 is taken.

```
canvas = Canvas(root,width=900,height=500,bg='#eee8aa')

tag= canvas.create_text(10,10,text="",anchor="nw")

canvas.pack()

canvas.create_line(470,0,470,500,width=2,fill='white')

canvas.create_line(50,250,450,250,width=2)

canvas.create_line(250,450,250,50,width=2)

canvas.create_text(550,60,text="ZEROES")

canvas.create_text(780,60,text="POLES")

canvas.create_text(680,20,text="Use SHIFT/RIGHT CLICK for entry of zeroes \n   and
ENTER/LEFT CLICK for entry of poles")

canvas.create_text(250,480,text="Use UP and DOWN arrow keys for magnitude
adjustments \n and LEFT and RIGHT arrow keys for angle adjustments")

oval=canvas.create_oval(250,250,250,250,width=1)

lin=canvas.create_line(250,250,250,250)


canvas.bind('<Motion>',motion)

canvas.bind('<Button-1>',poles)

canvas.bind('<Button-3>',zeroes)
```

```
root.bind('<Return>',poles)


root.bind('<Shift_L>',zeroes)
root.bind('<Up>',up)
root.bind('<Down>',down)
root.bind('<Right>',right)
root.bind('<Left>',left)


for i in range(21):
    x= 50 + (i*20)
    canvas.create_line(x,250,x,245,width=2)
    canvas.create_text(x,254,text='%d'%(i-10),anchor=N)
for j in range(21):
    y= 450 - (j*20)
    canvas.create_line(250,y,255,y,width=2)
    canvas.create_text(245,y,text='%d'%(j-10),anchor=E)
x=0;
y=0;
l1= Label(root,text='GAIN')
l1.pack(side='left')
z1=Entry(root)
z1.pack(side='left')
z1.insert(3,'1')
gain=float(z1.get())
but=Button(root,text='DONE',command=done,state='disabled')
but.pack(side=RIGHT)
but1=Button(root,text='ADD POLES AND ZEROES ',state='disabled',command=addpz)
but1.pack(side=BOTTOM)



def motion(event) :
    global t
    global h
    global p
```

```
    global q
    global but
    t = ((450-event.y)/20)-10
    h=((event.x-50)/20)-10
    p =math.sqrt(math.pow(h,2) + math.pow(t,2))
    q= math.degrees(math.atan2(t,h))
    if(p>10) :
        p=10
        canvas.itemconfigure(tag,text="(-,-)")
        canvas.coords(oval,(250-20*10,250-20*10,250+20*10,250+20*10))
        canvas.coords(lin,(250,250,250,250))
    else :
        canvas.coords(oval,(250-20*p,250-20*p,250+20*p,250+20*p))
        canvas.coords(lin,(250,250,event.x,event.y))
        canvas.itemconfigure(tag,text="%.2f"%p  + "  < " + "%.2f"%q )



def up(event) :
    global h
    global t
    global p
    global q
    p= p+0.1
    h= p*math.cos(math.radians(q))
    t = p*math.sin(math.radians(q))
    if(q==180) :
        t=0
    if(p>10) :
        p=10
        canvas.itemconfigure(tag,text="(-,-)")
        canvas.coords(oval,(250-20*10,250-20*10,250+20*10,250+20*10))
        canvas.coords(lin,(250,250,250,250))
    else :
        canvas.coords(oval,(250-20*p,250-20*p,250+20*p,250+20*p))
```

```
        canvas.coords(lin,(250,250,50+20*(10+h),450-20*(10+t)))
        canvas.itemconfigure(tag,text="%.2f"%p  + "  < " + "%.2f"%q )

def down(event) :
    global h
    global t
    global p
    global q
    p= p-0.1
    if(p<0) :
        p=0
    h= p*math.cos(math.radians(q))
    t = p*math.sin(math.radians(q))
    if(q==180) :
        t=0
    canvas.coords(oval,(250-20*p,250-20*p,250+20*p,250+20*p))
    canvas.coords(lin,(250,250,50+20*(10+h),450-20*(10+t)))
    canvas.itemconfigure(tag,text="%.2f"%p  + "  < " + "%.2f"%q )

def left(event) :
    global h
    global t
    global p
    global q
    q= q + 1
    h= p*math.cos(math.radians(q))
    t = p*math.sin(math.radians(q))
    if(q==180) :
        t=0
    if(p>10) :
        p=10
        canvas.itemconfigure(tag,text="(-,-)")
        canvas.coords(oval,(250-20*10,250-20*10,250+20*10,250+20*10))
        canvas.coords(lin,(250,250,250,250))
    else :
```

```
        canvas.coords(oval,(250-20*p,250-20*p,250+20*p,250+20*p))
        canvas.coords(lin,(250,250,50+20*(10+h),450-20*(10+t)))
        canvas.itemconfigure(tag,text="%.2f"%p  + "  < " + "%.2f"%q )

def right(event) :
    global h
    global t
    global p
    global q
    q= q - 1
    h= p*math.cos(math.radians(q))
    t = p*math.sin(math.radians(q))
    if(q==180) :
        t=0
    if(p>10) :
        p=10
        canvas.itemconfigure(tag,text="(-,-)")
        canvas.coords(oval,(250-20*10,250-20*10,250+20*10,250+20*10))
        canvas.coords(lin,(250,250,250,250))
    else :
        canvas.coords(oval,(250-20*p,250-20*p,250+20*p,250+20*p))
        canvas.coords(lin,(250,250,50+20*(10+h),450-20*(10+t)))
        canvas.itemconfigure(tag,text="%.2f"%p  + "  < " + "%.2f"%q )

def done() :
    canvas.unbind('<Motion>')
    canvas.unbind('<Button-1>')
    canvas.unbind('<Button-3>')
    root.unbind('<Return>')
    root.unbind('<Shift_L>')
    root.unbind('<Up>')
    root.unbind('<Down>')
    root.unbind('<Right>')
    root.unbind('<Left>')
    canvas.itemconfigure(tag,text="(-,-)")
```

```
    canvas.coords(oval,(250,250,250,250))

    canvas.coords(lin,(250,250,250,250))

    menubar.entryconfig("Frequency Response",state='normal')

    menubar.entryconfig("Time Response",state='normal')

    but1.config(state='normal')

    but.config(state='disabled')

    if(k>=l):

        messagebox.showerror("ERROR", "THE NUMBER OF ZEROES MUST NOT EXCEED THE
NUMBER OF POLES. PLEASE EDIT THE ENTRY" )

        addpz()



def addpz():

    canvas.bind('<Motion>',motion)

    canvas.bind('<Button-1>',poles)

    canvas.bind('<Button-3>',zeroes)

    root.bind('<Return>',poles)

    root.bind('<Shift_L>',zeroes)

    root.bind('<Up>',up)

    root.bind('<Down>',down)

    root.bind('<Right>',right)

    root.bind('<Left>',left)

    menubar.entryconfig("Frequency Response",state='disabled')

    menubar.entryconfig("Time Response",state='disabled')

    but1.config(state='disabled')

    but.config(state='normal')

    root.title('GUI SYSTEM RESPONSE PACKAGE ')
```

2.3 Inputting Poles And Zeroes

Entry of the poles and zeroes will invoke the functions poles() and zeroes() which will take in the coordinates of the point and append it to an array. Four lists are defined for this purpose. rnum[] and inum[] are for real and imaginary parts of zeroes and rden[] and iden[] are for real and imaginary parts of poles. Complex poles and zeroes are taken in as conjugate pairs. There is a restriction on the number of poles and zeroes. The number of poles must always be greater than the number of zeroes. Multiplicity of poles and zeroes are also not permitted.

```
def zeroes(event) :
    global but
    but.config(state='active')
    global k
    global h
    global t
    global p
    global zcount
    if(h in rnum and t in inum) :
        messagebox.showerror("ERROR", "MULTIPLE ORDERS ARE NOT PERMITTED" )
    else :
        if(p<10) :
            if(t==0) :
                zcount=zcount+1;
                rnum.append(h)
                inum.append(t)
                if(pow(t,2) > 2*pow(h,2)) :
                    Wr= math.sqrt(pow(t,2) - pow(h,2))
                    wr.append(Wr)
                gat2= canvas.create_text(478,70+25*(zcount),text="",anchor="nw")
                canvas.itemconfigure(gat2,text="%.2f"%h + " + " + "%.2f"%t + " i")
                gat3= canvas.create_text(((10+h)*20+50),(450-(10+t)*20),text="")
                canvas.itemconfigure(gat3,text="o")
                k=k+1
            else :
```

```
                zcount=zcount+1;
                rnum.append(h)
                inum.append(t)
                rnum.append(h)
                inum.append(-1*t)
                if(pow(t,2) > 2*pow(h,2)) :
                    Wr= math.sqrt(pow(t,2) - pow(h,2))
                    wr.append(Wr)
                    Wn=math.sqrt(pow(h,2) + pow(t,2))
                    delta = h/ Wn
                    sc.append(Wn*(1-2*pow(delta,2)+math.sqrt(2-
4*pow(delta,2)+4*pow(delta,4))))
                gat= canvas.create_text(478,70+25*zcount,text="",anchor="nw")
                gat1= canvas.create_text(578,70+25*zcount,text="",anchor="nw")
                gat5= canvas.create_text(((10+h)*20+50),(450-(10+t)*20),text="")
                canvas.itemconfigure(gat5,text="o")
                gat4= canvas.create_text(((10+h)*20+50),(450-(10-t)*20),text="")
                canvas.itemconfigure(gat4,text="o")
                canvas.itemconfigure(gat,text="%.2f"%h + " + " + "%.2f"%t + " i")
                canvas.itemconfigure(gat1,text="%.2f"%h +"+"+"%.2f"%(-1*t) + " i")
                k=k+2



def poles(event) :
    global but
    but.config(state='active')
    global l
    global h
    global t
    global p
    global q
    global pcount
    if(h in rden and t in iden) :
        messagebox.showerror("ERROR", "MULTIPLE ORDERS ARE NOT PERMITTED" )
    else:
```

```
        if(p<10) :
            if(t==0) :
                pcount=pcount+1;
                rden.append(h)
                iden.append(t)
                if(pow(t,2) > 2*pow(h,2)) :
                    Wr= math.sqrt(pow(t,2) - pow(h,2))
                    wr.append(Wr)
                cat2= canvas.create_text(700,70+25*pcount,text="",anchor="nw")
                canvas.itemconfigure(cat2,text="%.2f"%h + " + " + "%.2f"%t + " i")
                cat3= canvas.create_text(((10+h)*20+50),(450-(10+t)*20),text="")
                canvas.itemconfigure(cat3,text="x")
                l=l+1
            else :
                pcount=pcount+1;
                rden.append(h)
                iden.append(t)
                rden.append(h)
                iden.append(-1*t)
                if(pow(t,2) > 2*pow(h,2)) :
                    Wr= math.sqrt(pow(t,2) - pow(h,2))
                    wr.append(Wr)
                    Wn=math.sqrt(pow(h,2) + pow(t,2))
                    delta = h/ Wn
                    sc.append(Wn*(1 - 2*pow(delta,2)+ math.sqrt( 2- 4*pow(delta,2)
+ 4*pow(delta,4))))
                cat= canvas.create_text(700,70+25*pcount,text="",anchor="nw")
                cat1= canvas.create_text(800,70+25*pcount,text="",anchor="nw")
                cat5= canvas.create_text(((10+h)*20+50),(450-(10+t)*20),text="")
                canvas.itemconfigure(cat5,text="x")
                cat4= canvas.create_text(((10+h)*20+50),(450-(10-t)*20),text="")
                canvas.itemconfigure(cat4,text="x")
                canvas.itemconfigure(cat,text="%.2f"%h + " + " + "%.2f"%t + " i")
               canvas.itemconfigure(cat1,text="%.2f"%h +" +"+"%.2f"%(-1*t) + " i")
                l= l+2
```

2.4 Output Features

2.4.1 Canvas Creation

The main canvas window for the output is created using the Toplevel() function. It creates a sub-window which is connected to the main window. A scrollbar is given for each plot permitting the user to scan the whole plot. Lines representing the coordinate axes are also drawn on the canvas.

```
    sub2 = Toplevel(root)
    sub2.title("wm min/max")
    sub2.resizable(0,0)
    sub2.title("BODE PLOT")
    frame=Frame(sub2)
    frame.pack()
    bottomframe=Frame(sub2)
    bottomframe.pack(side= BOTTOM)
    w=0
    mag= 1;
    ang= 0;
    xvar = [ ];
    wm = [ ];
    Ang = [ ];
    numa=0;
    dena=0;
    global k
    global l


    scrollbar = Scrollbar(frame)
    scrollbar.pack( side = BOTTOM, fill=Y )
    canvas3 =
Canvas(frame,width=600,height=300,bg='#eeeee0',xscrollcommand=scrollbar.set,scrollr
egion=(0,0,1110,1110))
    canvas3.pack()
```

```
    scrollbar1 = Scrollbar(bottomframe)

    scrollbar1.pack( side = BOTTOM, fill=Y )

    canvas4 =
Canvas(bottomframe,width=600,height=300,bg='#cdcdc1',xscrollcommand=scrollbar.set,s
crollregion=(0,0,1110,1110))

    canvas4.pack()

    scrollbar.config( command = canvas3.xview,orient= HORIZONTAL )

    scrollbar1.config( command = canvas4.xview,orient= HORIZONTAL )

    canvas3.create_line(100,250,100,50,width=2)

    canvas3.create_line(100,150,1100,150,width=2)

    canvas4.create_line(100,250,100,50,width=2)

    canvas4.create_line(100,150,1100,150,width=2)

    canvas3.create_text(20,80,text="\n".join("MAGNITUDE"), anchor="nw")

    canvas3.create_text(300,20,text="MAGNITUDE PLOT")

    canvas3.create_text(300,280,text="FREQUENCY (in rad/s)")

    canvas4.create_text(20,120,text="\n".join("ANGLE"), anchor="nw")

    canvas4.create_text(300,20,text="ANGLE PLOT")

    canvas4.create_text(300,280,text="FREQUENCY (in rad/s)")
```

2.4.2 Selection of X-Scale

Bode Plot

For bode plot, the frequency axis is a logarithmic scale. Points marked on the scale are in powers of 10. The upper limit of the scale is 10^4 and the scale begins from 0.1. It is equidistant for every increase in powers of 10.

```
w=pow(10,((d-200)/200))
```

where d is the distance from the origin and the 200 represents the distance between each powers of 10. The magnitude values corresponding to each w is calculated for every 1 cm increase in d, i.e., d belongs to (0,1000).

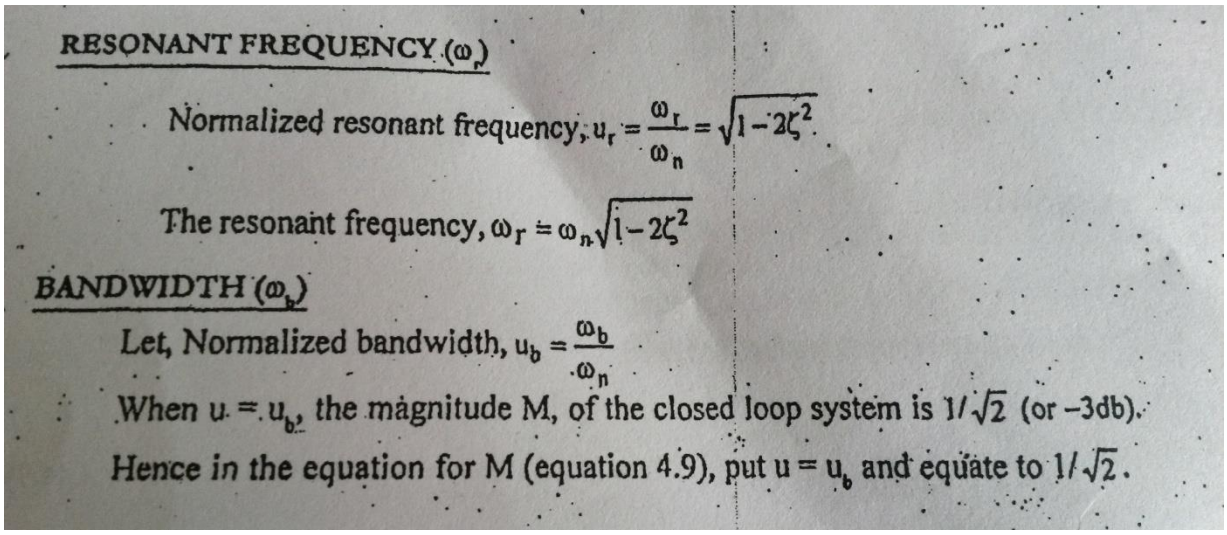
Absolute Plot

To find the scale and the upper limit of the frequency axis, we need to first calculate the bandwidth. Bandwidth is the range of frequencies for which the system normalized gain in more than -3db. Complex poles and zeroes contribute to resonant peaks. Hence, bandwidth is calculated for complex poles and zeroes.

Let a and b be the real and imaginary part of a pole/zero.

Wn=(a^2+b^2)^0.5

δ = (a/Wn)



RESONANT FREQUENCY (ω_r)

Normalized resonant frequency, $u_r = \frac{\omega_r}{\omega_n} = \sqrt{1-2\zeta^2}$

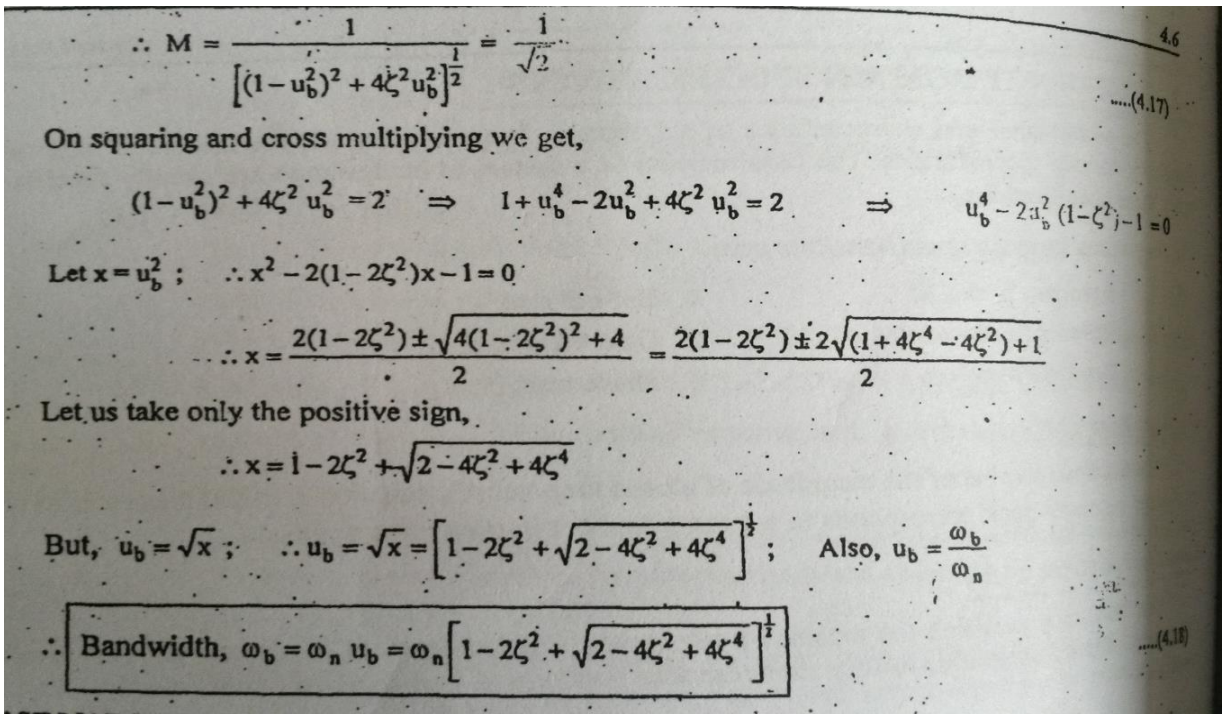
The resonant frequency, $\omega_r = \omega_n\sqrt{1-2\zeta^2}$

BANDWIDTH (ω_b)

Let, Normalized bandwidth, $u_b = \frac{\omega_b}{\omega_n}$

When $u = u_b$, the magnitude M, of the closed loop system is $1/\sqrt{2}$ (or –3db).

Hence in the equation for M (equation 4.9), put $u = u_b$ and equate to $1/\sqrt{2}$.



$$\therefore M = \frac{1}{\left[(1-u_b^2)^2 + 4\zeta^2 u_b^2\right]^{\frac{1}{2}}} = \frac{1}{\sqrt{2}} \qquad \text{4.6}$$

.....(4.17)

On squaring and cross multiplying we get,

$$(1-u_b^2)^2 + 4\zeta^2 u_b^2 = 2 \Rightarrow 1 + u_b^4 - 2u_b^2 + 4\zeta^2 u_b^2 = 2 \Rightarrow u_b^4 - 2u_b^2(1-\zeta^2) - 1 = 0$$

Let $x = u_b^2$; $\therefore x^2 - 2(1-2\zeta^2)x - 1 = 0$

$$\therefore x = \frac{2(1-2\zeta^2) \pm \sqrt{4(1-2\zeta^2)^2 + 4}}{2} = \frac{2(1-2\zeta^2) \pm 2\sqrt{(1+4\zeta^4-4\zeta^2)+1}}{2}$$

Let us take only the positive sign,

$$\therefore x = 1 - 2\zeta^2 + \sqrt{2 - 4\zeta^2 + 4\zeta^4}$$

But, $u_b = \sqrt{x}$; $\therefore u_b = \sqrt{x} = \left[1 - 2\zeta^2 + \sqrt{2 - 4\zeta^2 + 4\zeta^4}\right]^{\frac{1}{2}}$; Also, $u_b = \frac{\omega_b}{\omega_n}$

$$\therefore \boxed{\text{Bandwidth, } \omega_b = \omega_n u_b = \omega_n\left[1 - 2\zeta^2 + \sqrt{2 - 4\zeta^2 + 4\zeta^4}\right]^{\frac{1}{2}}} \qquad \text{.....(4.18)}$$

```
Wr= math.sqrt(pow(t,2) - pow(h,2)
wr.append(Wr)
Wn=math.sqrt(pow(h,2) + pow(t,2))
delta = h/ Wn
sc.append(Wn*(1 - 2*pow(delta,2)+ math.sqrt( 2- 4*pow(delta,2) + 4*pow(delta,4))))

if(len(sc)==1) :
     scale = 0.005
else :
     sc.remove(0)
     scale= 0.005*min(sc)

UL= scale*500
ful=-1
    remul=UL
    while(remul>=1) :
    remul= remul/ 10
    ful=ful+1;
if(UL <1):
        rem1ul = UL
        ful=0
        while(rem1ul<=1):
            rem1ul=rem1ul*10;
            ful=ful-1;
UL= pow(10,(ful+1))
```

Time Response

The time response time scale is calculated using the following code. The same code is followed for both step and impulse response.

```
scale= 1/(150*abs(max(rden)))
UL= 3/abs(min(rden))
```

```
ful=-1
remul=UL
while(remul>=1) :
      remul= remul/ 10
      ful=ful+1;
      if(UL <1):
          rem1ul = UL
          ful=0
while(rem1ul<=1):
      rem1ul=rem1ul*10;
      ful=ful-1;
UL= pow(10,(ful+1))
```

2.4.3 Selection of Y-scale

The y-axis scale is dynamically computed by determining the maximum value to which magnitude of the response can peak up. For this, a for loop is given which estimates the magnitude for each value of w. The maximum magnitude is divided by 10 repetitively until only a single digit is left behind. The succeeding number appended with trailing zeroes will be the new maximum magnitude. Each division is one-tenth of the maximum magnitude.

```
while(w<UL):
        num=1;
        den=1;
        numa=0;
        dena=0;
        for g in range (k) :
            num= num*cmath.sqrt(pow((w-inum[g]),2)+ pow(rnum[g],2));
            m=math.degrees(math.atan2((w-inum[g]),rnum[g]))
            if(m<0):
                m=m+360

            numa = numa + m;
        for m in range (l) :
            den= den*cmath.sqrt(pow((w-iden[m]),2)+pow(rden[m],2));
```

```
    md=math.degrees(math.atan2((w-iden[m]),rden[m]))
    if(md<0):
        md=md+360
    dena = dena + md;
mag=mag1;
ang=ang1;
mag1=abs(num/den);
ang1=(numa - dena);
if(mag1>magmax) :
    magmax=mag1
    rem=magmax
    f=-1
    while(rem>=10) :
        rem= rem/ 10
        f=f+1;
    div=int(rem+1)*pow(10,f)
    if(magmax <1):
        rem1 = magmax
        f=0
        while(rem1<=1):
            rem1=rem1*10;
            f=f-1;
            div= int(rem1+1)*pow(10,f-1)
if(abs(ang1)>angmax) :
    angmax=abs(ang1)
    rema=angmax
    f1=1
    if(rema>=45) :
        rema= int(rema/ 180)
        f1=f1+rema;
    if(angmax <45):
        rema1 = angmax
        f1=(1/4.5)
w=w+scale;
n=n+1;
```

2.4.4 Calculation of Residues and Laplace Inverse for Time Response Plots

The inverse Laplace transform

The formula for the inverse Laplace transform was obtained in the previous section as:

$$f(t) = \frac{1}{2\pi j}\int_{s=\sigma-j\infty}^{\sigma+j\infty} F(s)e^{st}ds$$

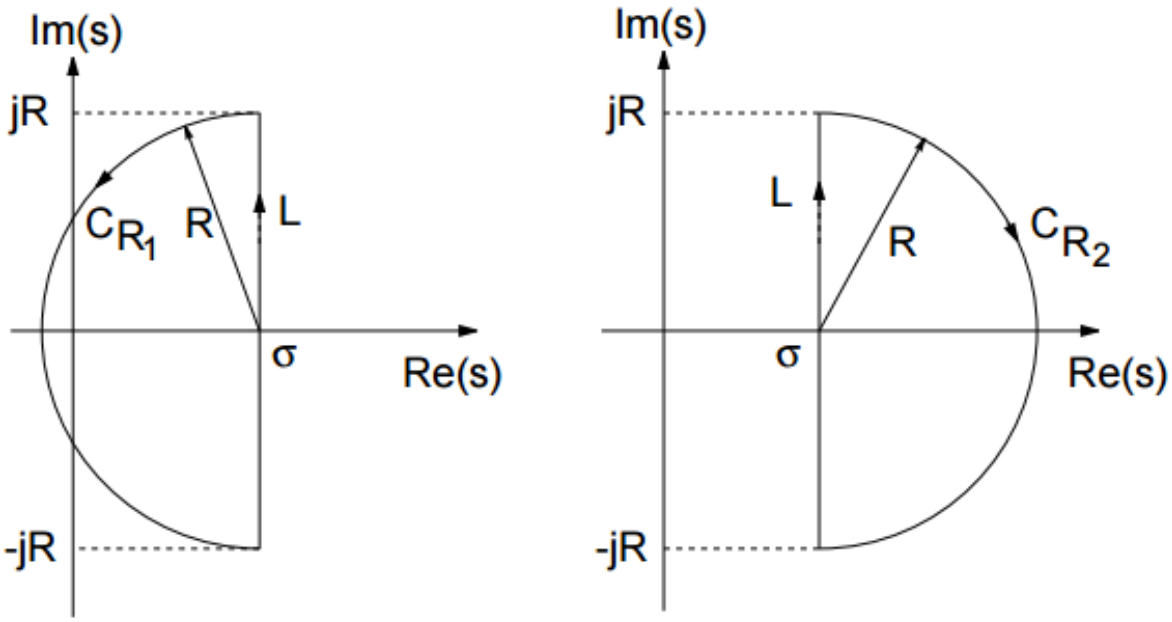


Fig1. Calculation of Residues and Laplace inverse

First, consider the closed contour $C_1 = C_{R_1} + L$ shown in Figure 1*a*. Then,

$$\begin{aligned}\oint_{C_1} F(s)e^{st}ds &= \int_L F(s)e^{st}ds + \int_{C_{R_1}} F(s)e^{st}ds \\ &= \int_{\sigma-jR}^{\sigma+jR} F(s)e^{st}ds + \int_{C_{R_1}} F(s)e^{st}ds \\ &= 2\pi j \sum_{\text{poles in } C_1} \text{Res}[F(s)e^{st}]\end{aligned}$$

Example 1

Find the inverse transform of $F(s) = \frac{1}{s+a}$.
The function $\frac{e^{st}}{s+a}$ has a simple pole at $s = -a$. Hence

$$\text{Res}_{s=-a}[F(s)e^{st}] = \lim_{s\to -a}\left[(s+a)\frac{e^{st}}{s+a}\right] = e^{-at}$$

and so,

$$L^{-1}\left[\frac{1}{s+a}\right] = e^{-at}, \quad t \geq 0$$
$$= 0, \quad t < 0$$

As explained above the residue at a pole is calculated by eliminating that pole and calculating the value of the obtained transfer function at this pole. The function used for this in TFAP is given below.

```
while(nv<l):
    a=[];
    s=(rden[nv]+iden[nv]*(1j))
    for hop in range(l) :
        a.append(rden[hop]+iden[hop]*(1j))
    a.remove(s)
    sb=l-1
    while(sa<k):
        lnum=lnum*(s-(rnum[sa]+inum[sa]*(1j)))
        sa=sa+1
    while(gh<sb):
        lden=lden*(s-a[gh])
        gh=gh+1
    lm=(lnum/lden)
    if(iden[nv]==0):
        ab.append(lm)
        abc.append(math.exp(-1*s.real))
    else:
        xy.append(lm)
        xy1.append(rden[nv])
        xy2.append(iden[nv])
```

```
        nv=nv+1
    uq=0
    xyz=[]
    xyz1=[]
    xyz2=[]
    while(uq<len(xy)):
        xyz.append(xy[uq]+xy[uq+1])
        xyz1.append(xy1[uq])
        xyz2.append(xy2[uq])
        uq=uq+2
```

2.4.5 Plotting the graphs

The final feature of the output is the plotting of the graphs. The graphs are plotted using the canvas.createline feature of the Tkinter. Points are joined by straight lines between them to get the graph. For this a for loop is used to get the magnitude or angle at different values of w. After this the axes are scaled and marked. The code is given below.

```
    while(t<UL) :
        mag=mag1
        for go in range (len(ab)) :
            mag1=mag1+ ((ab[go]).real*math.pow(abc[go],t))
        for go1 in range (int(uq/2)) :
            mag1 = mag1 + ((xyz[go1]).real * math.exp(-
1*xyz1[go1]*t)*math.cos(xyz2[go1] *t))


        canvas8.create_line((100+n),(150-(10*mag/div)),(101+n),(150-
(10*mag1/div)),width=2)
        t=t+scale
        n=n+1


    for i in range(11):
        x= 100 + (i*(0.1*UL/scale))
        canvas8.create_line(x,150,x,145,width=2)
        canvas8.create_text(x,155,text='%.1f'%(i*0.1*UL),anchor="ne")
```

```
for j in range(11):
    y= 250 - (j*20)
    canvas8.create_line(100,y,105,y,width=2)
    canvas8.create_text(95,y,text='%.2f'%((-5 +j)* 2*div),anchor=E)
```

This concludes a brief explanation about the various modules and functions used in TFAP.

CHAPTER 3

A VISUAL OVERVIEW OF TFAP

The Transfer Function Analysis package has the main window with the menu bar, canvas and buttons as explained before. Fig2 shows the main window of TFAP.

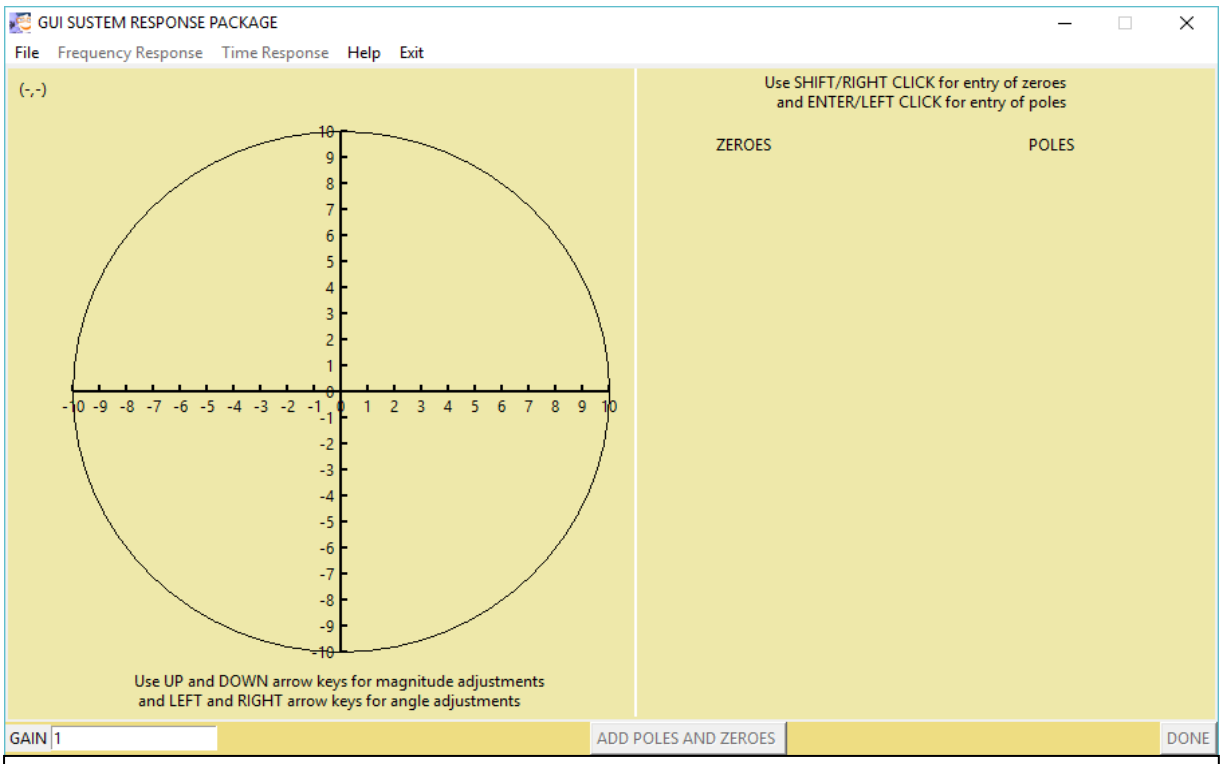


Fig2. The TFAP main window

The entry of poles and zeroes can be done by positioning the mouse pointer to appropriate position or by using the arrow keys as shown in fig3.

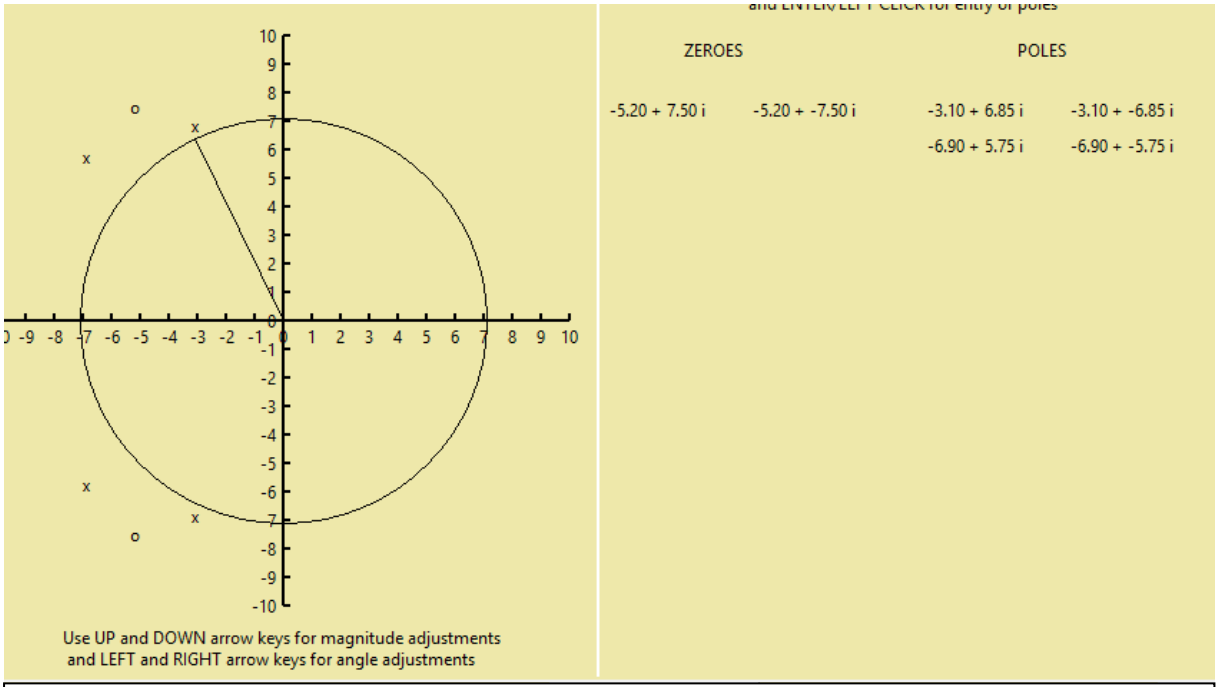


Fig3. Entry of poles and zeroes

When the number of poles exceeds the number of zeroes an error message is shown to the user as shown below in fig4. Error message is also produced when repeated poles and zeroes are entered.

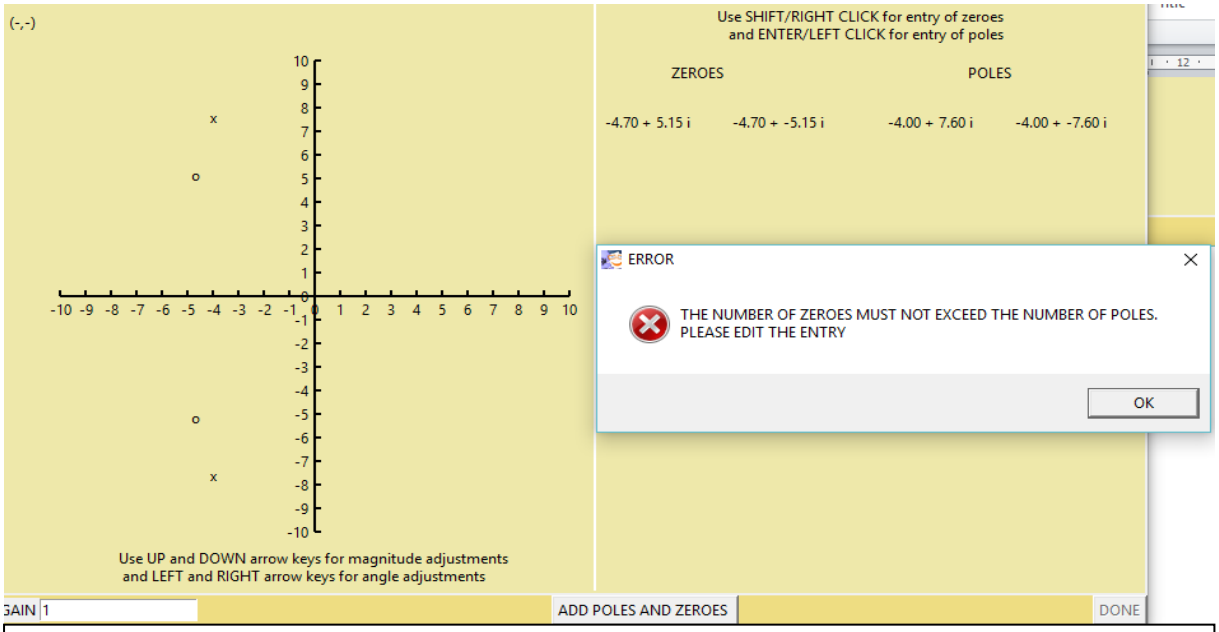


Fig4. Error message when number of poles is not greater than zeroes

Once the entry of poles and zeroes is done, the done button can be selected which activates ADD POLES AND ZEROES button and the menubar buttons such as FREQUENCY RESPONSE AND TIME RESPONSE buttons which were initially inactive get activated. These menubar options are shown below in Fig5 and Fig6.

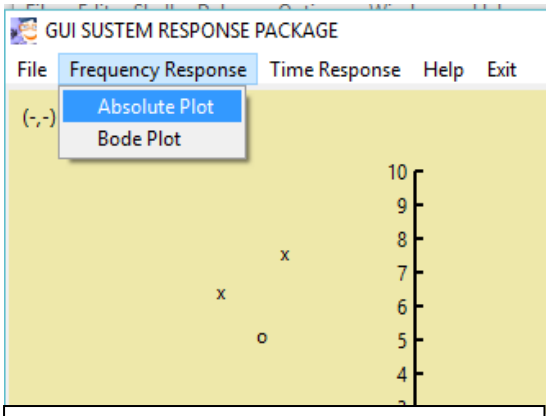


Fig5. Frequency response menu

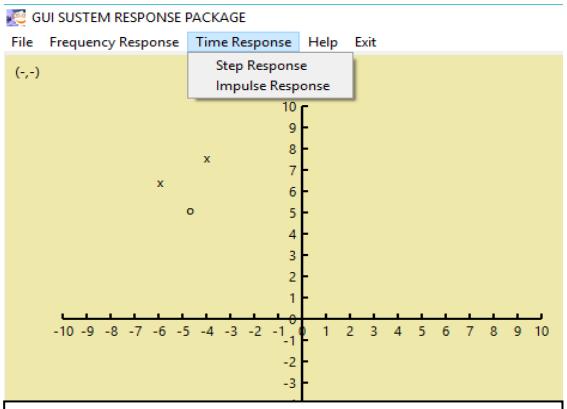


Fig6. Time response menu

The Help window of TFAP provides the user with the guidelines to use the software. The window is shown in the following figure.

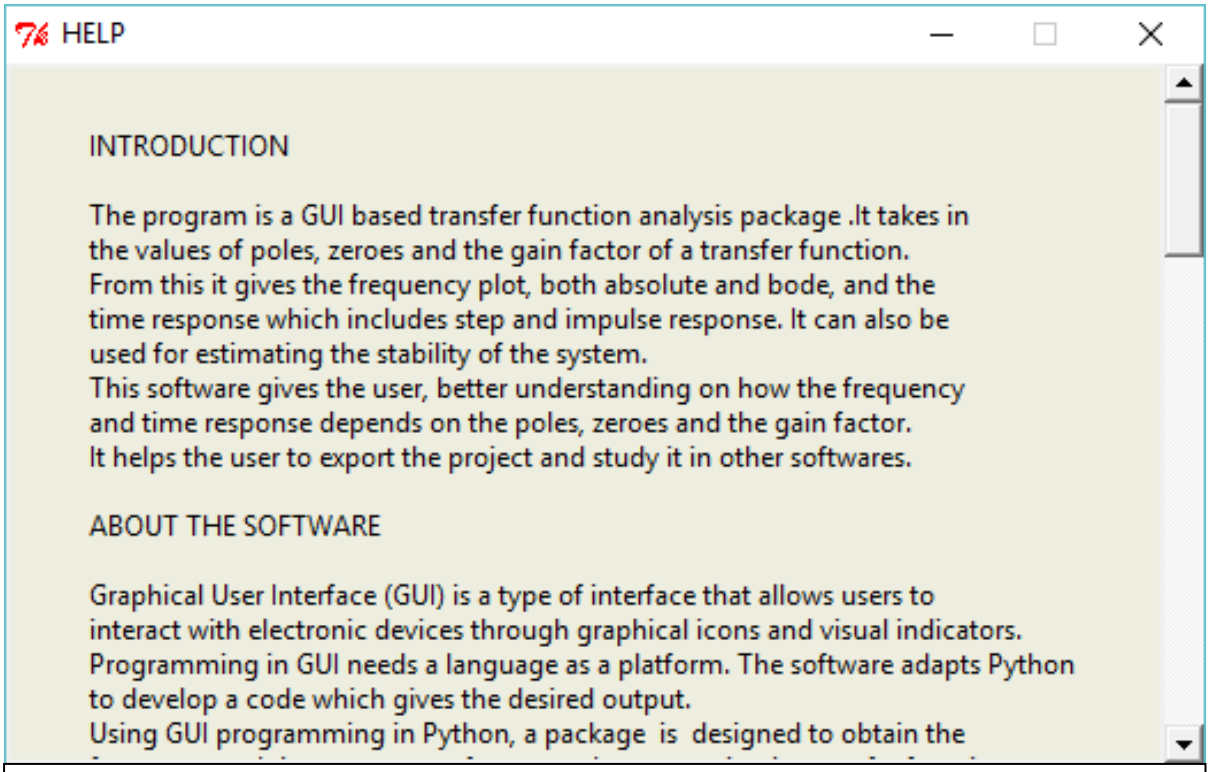


Fig7. Help menu

The file menu has options like New, Save and Exit. These are shown in the following figures.

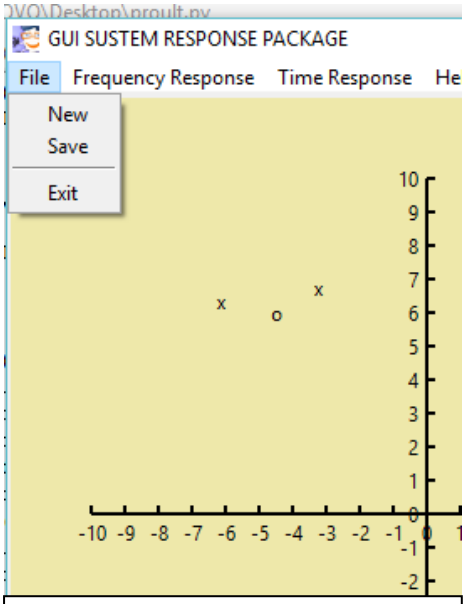


Fig8. File menu

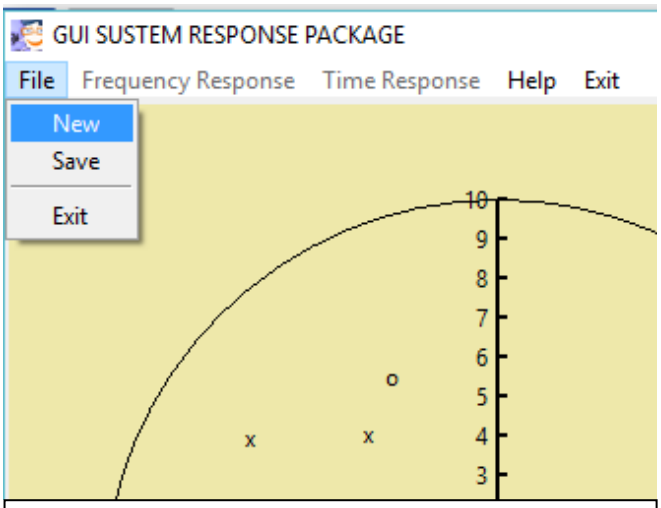


Fig9. New button

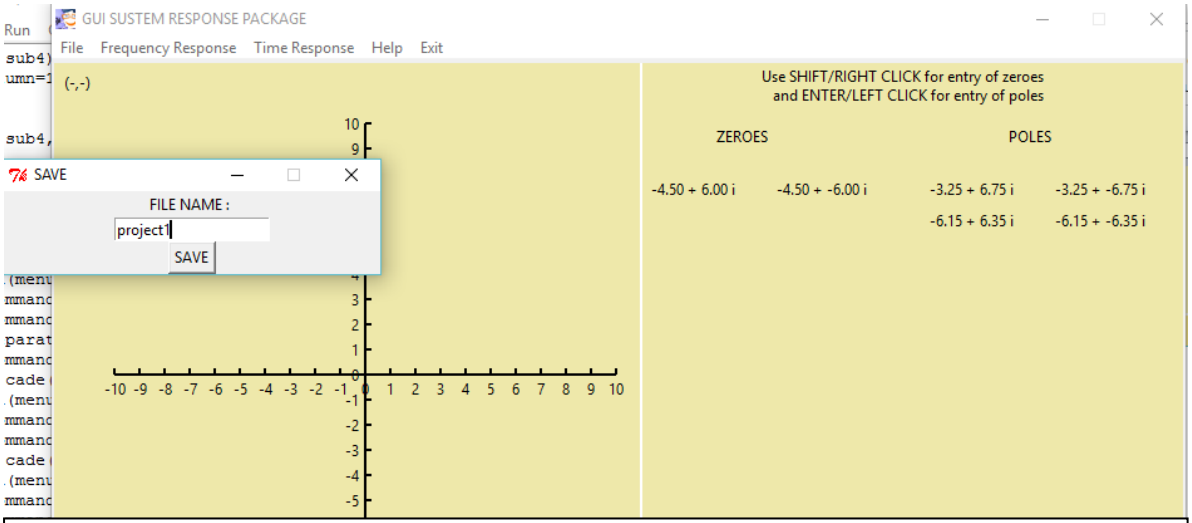


Fig10. Save command

The output plots are frequency response both absolute and bode and time response consisting of impulse and step response. These are shown in the following figures.

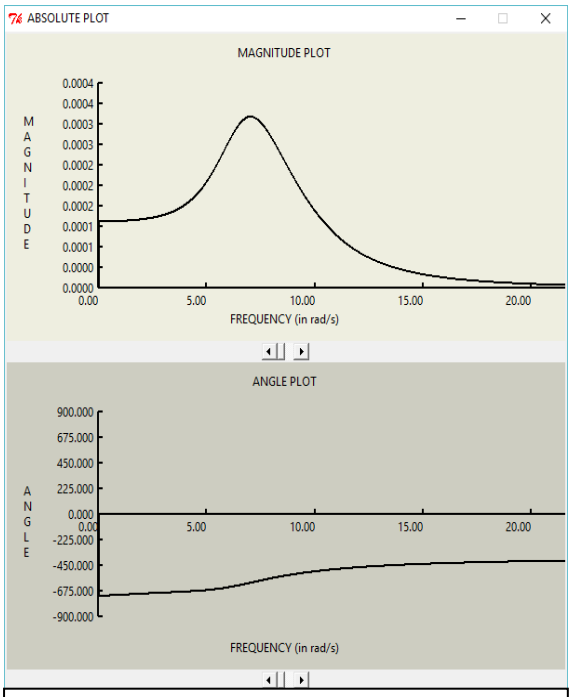


Fig11. Absolute plots

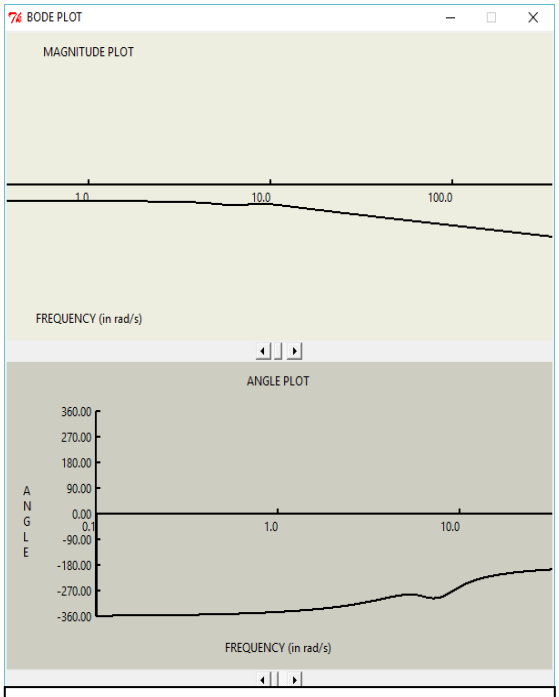


Fig12. Bode plots

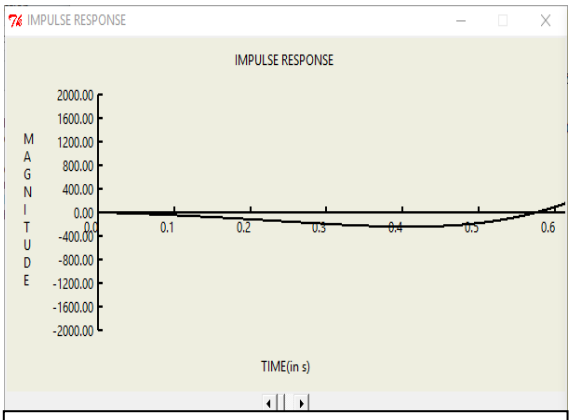


Fig13. Impulse Plot

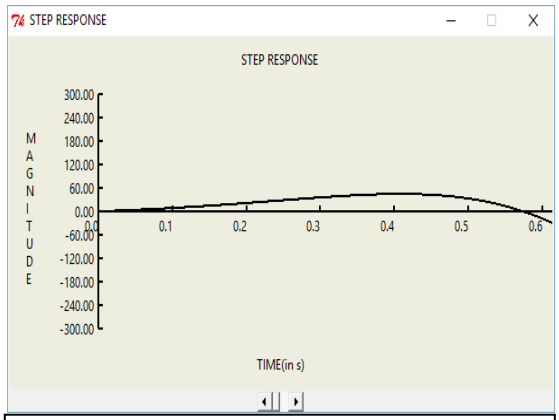


Fig14. Step Plot

CHAPTER 4

SHORTCOMINGS OF TFAP AND FUTURE EXTENSIONS POSSIBLE

The TFAP falls short of the expected software in some aspects.

The load function of the File Menu could not be properly executed. A load function is a function which should be able to load an already saved file or load a file written in the appropriate text format.

The initial program was expected to save the obtained output into a neat image format file such as a .png . However the module for this function could not be properly installed and linked with the python code to achieve this. This resulted in a code which could only convert the canvas into a postscript and not an image file, thus rendering it futile.

The New and Exit command of the Menubar could not be linked with the Save function to allow the AutoSaving of the file while closing it or opening a new one ,as an existing or running root could not be deleted while being saved.

Editing of the poles and zeroes are limited to the addition of them and no provision is given in TFAP for the removal of poles and zeroes. This drawback occurred due to the inability of the code to distinguish between real and complex roots to exactly obtain which pole or zero to be eliminated.

The save option only allows the file to be saved in the default save drive of the user. Thus no provision is given to the user for selecting any random location in her/his drive to save the file. This can be achieved using a module WinAPI which gives a platform to link many commands in python with the standard windows command.

Another major drawback is that the TFAP runs only on the Windows and is not compatible with any other OS.

These shortcomings need to be overcome and extensions must be incorporated to make the software more user-friendly and efficient.

REFERENCES

Control Systems by A. Nagoor Kani

Modern Control Engineering by Katsuhiko Ogata

Python Programming language by Tutorials point

www.wikipedia.org

www.effbot.org

www.wiki.python.org

www.python-course.eu

https://docs.python.org

http://stackoverflow.com/