Assignemnt
Error Analysis and Clenshaw algorithm

Arunima Sarkar: EE21S062

December 2021

1 Quadratic equation

The Quadratic equation can be written as :

$$z^2 + \frac{a}{\sqrt{b}}z + 1 = 0 \tag{1}$$

And if $2\alpha = \frac{a}{\sqrt{b}}$, then the equation has a single parameter with solution as,

$$z = -\alpha \pm \sqrt{\alpha^2 - 1} \tag{2}$$

1.1 Codes

The corresponding C code to check the problems that arise in quadratic equation and how the accurate formula performs is.

```
#include<stdio.h>
#include<math.h>
#include<complex.h>

int sgn(float x)
{
    if(x>0)
     return 1;
    if(x<0)
     return -1;
    else
     return 0;

}
void main()
{
    int N=10000,j=0;
```

```
float complex z1[N],z2[N],z3[N],z4[N];
double complex e1[N],e2[N];
double alpha[N],i;
FILE *fp1,*fp2;
fp1=fopen("error_from_exact_formula.txt","w");
fp2=fopen("error_from_accurate_formula.txt","w");
for (i=0;i<=1000;i=i+0.1)
{
    alpha[j]=i;
    j++;
}

for (j=0;j<N;j++)
{
   double complex y=0;
   if(fabs(alpha[j])<1)
      y=I*sqrt(1-pow(alpha[j],2));
   else
     y= sqrt(pow(alpha[j],2)-1);

    e1[j]=-(alpha[j])+ y;
    e2[j]=-(alpha[j])- y;

}


for (j=0;j<N;j++)
{
   float complex y=0;
   if(fabs(alpha[j])<1)
      y=I*sqrt(1-pow(alpha[j],2));
   else
     y= sqrt(pow(alpha[j],2)-1);

    z1[j]=-(alpha[j])+ y;
    z2[j]=-(alpha[j])- y;
    fprintf(fp1,"%.15f\t",alpha[j]);
    fprintf(fp1,"%.15f\t",cabs(z1[j]-e1[j]));
    fprintf(fp1,"%.15f\n",cabs(z2[j]-e2[j]));

}

for (j=0;j<N;j++)
{
    float complex p = 0;
    if(fabs(alpha[j])<1)
```

```
        p=I*sqrt(1-pow(alpha[j],2));
      else
       p= sqrt(pow(alpha[j],2)-1);
       z3[j]=1/(-(alpha[j])-p);
       z4[j]=1/(-(alpha[j])+p);

      fprintf(fp2,"%.15f\t",alpha[j]);
      fprintf(fp2,"%.15lf\t",cabs(z3[j]-e1[j]));
      fprintf(fp2,"%.15lf\n",cabs(z4[j]-e2[j]));


    }

    fclose(fp1);
    fclose(fp2);
}
```

The corresponding python code for plotting,

```
from matplotlib import colors
import numpy as np
import matplotlib.pyplot as plt

x=np.loadtxt("error_from_exact_formula.txt",usecols=0)
ya=np.loadtxt("error_from_exact_formula.txt",usecols=1)
ys=np.loadtxt("error_from_exact_formula.txt",usecols=2)
x=np.delete(x,0,0)
ya=np.delete(ya,0,0)
ys=np.delete(ys,0,0)

plt.clf()
x1,=plt.loglog(x,ya,'g',label="Non cancelling root")
y,=plt.loglog(x,ys,'r',label="Roots with near cancellation")
plt.legend(handles=[x1,y])
plt.grid()
plt.savefig("Error from exact formula")

x1=np.loadtxt("error_from_accurate_formula.txt",usecols=0)
ya1=np.loadtxt("error_from_accurate_formula.txt",usecols=1)
ys1=np.loadtxt("error_from_accurate_formula.txt",usecols=2)
x1=np.delete(x1,0,0)
ya1=np.delete(ya1,0,0)
ys1=np.delete(ys1,0,0)
plt.clf()
m,=plt.loglog(x1,ya1,'g',label="Roots with  near cancellation")
n,=plt.loglog(x1,ys1,'r',label="Non cancelling root")
```

```
plt.legend(handles=[m,n])
plt.grid()
plt.savefig("Error from accurate formula")
```

1.2 Corresponding diagrams

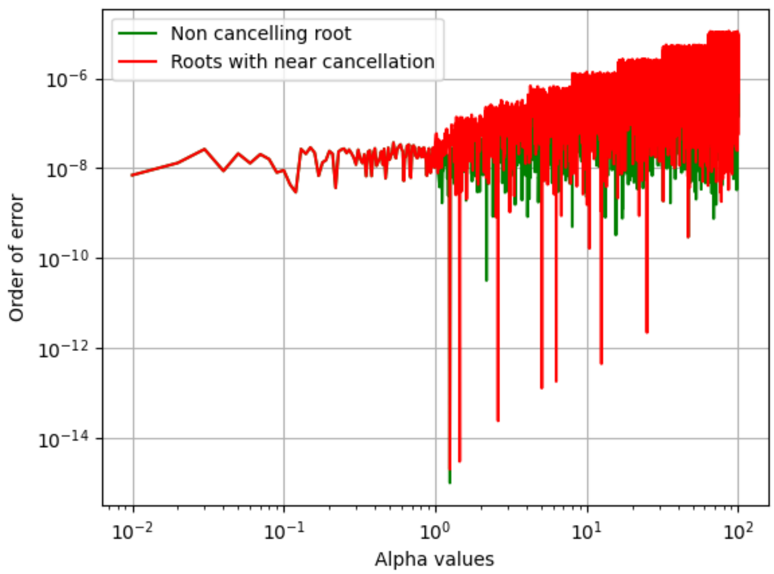

Figure 1: Error from exact formula

The roots with near cancellation reaches around order 4 accuracy but the one without near cancellation is reaching order 6 accuracy if eq(2) is followed. The algorithm below is used for calculating numerically accurate near cancelling roots.

$$p = -\big(\alpha + (sgn)\alpha\sqrt{\alpha^2 - 1}\big) \tag{3}$$

z1 = p
z2 = 1/p

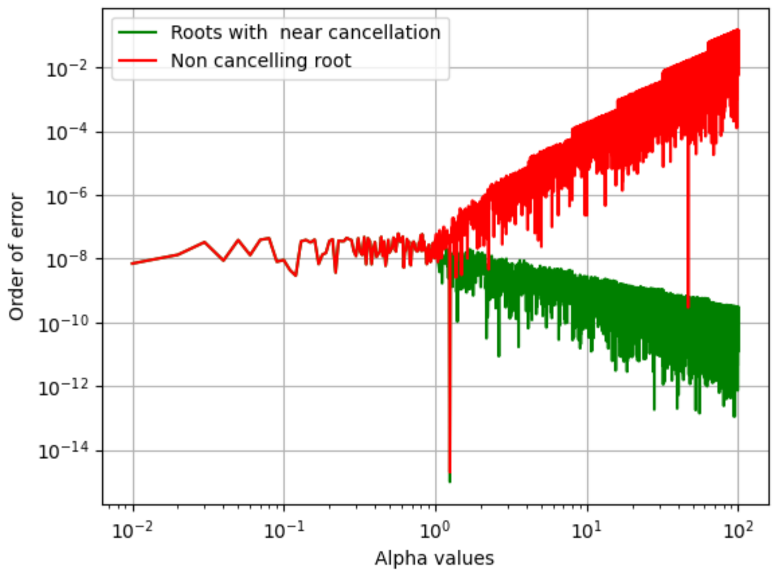

Figure 2: Error from accurate formula for near cancelling roots

For the root $z = -\alpha + \sqrt{\alpha^2 - 1}$ the accurate formula shows $z = \frac{1}{-\alpha - \sqrt{\alpha^2 - 1}}$. Hence near cancellations are avoided and accuracy reaches upto order 11. However for the root which shows better accuracy with exact formula, $z = -\alpha - \sqrt{\alpha^2 - 1}$ has value $z = \frac{1}{-\alpha + \sqrt{\alpha^2 - 1}}$, which shows near cancelling in denominator.Thus error reaches to order 1.

Hence it can be concluded that such numerical errors can be avoided with the modified formula.

2 Stable and Unstable Series

Below is the code for Forward and backward recursion.

2.1 Forward Recursion

```
import matplotlib
matplotlib.use('nbagg')
import matplotlib.pyplot as plt
import numpy as np
import scipy.special as sp


def s(x,N):
    y=0.0
    for i in range(N+1):
        y=y+(1/(1+i))*sp.jv(i,x)
    return y

# Forward recursion
x=15
j=[0,sp.jv(0,x) ]
for i in range(41):
    j.append(((2*(i))/x)*j[-1]-j[-2])

sum=0
error=[]
for n in range(41):
    sum=sum+j[n]/(1+n)
    error.append(abs(s(x,n)-sum))

xval=[i for i in range(41)]
plt.clf()
plt.semilogy(xval,error,'c')
plt.xlabel("n Values")
plt.ylabel(" Order of error")
plt.title("Forward recursion Error for x=15")
plt.grid()
plt.show()
```

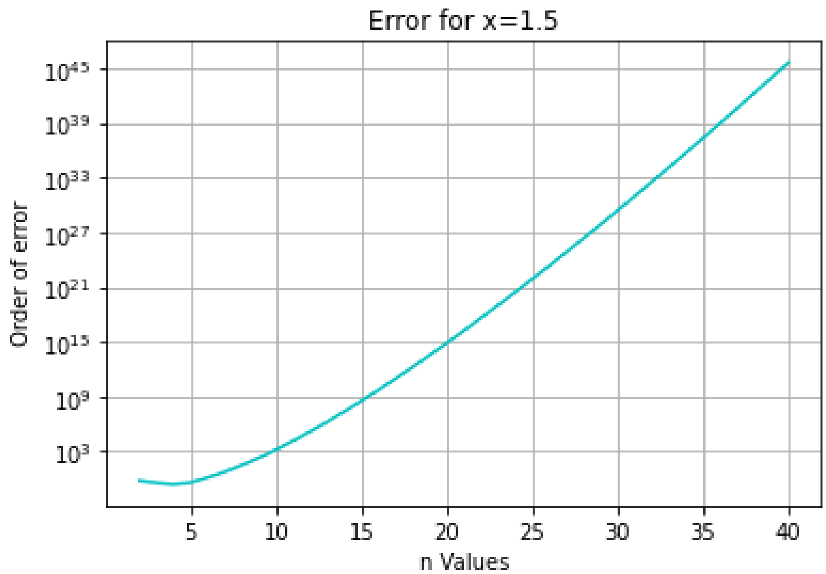

Figure 3: Forward recursion error,x=1.5

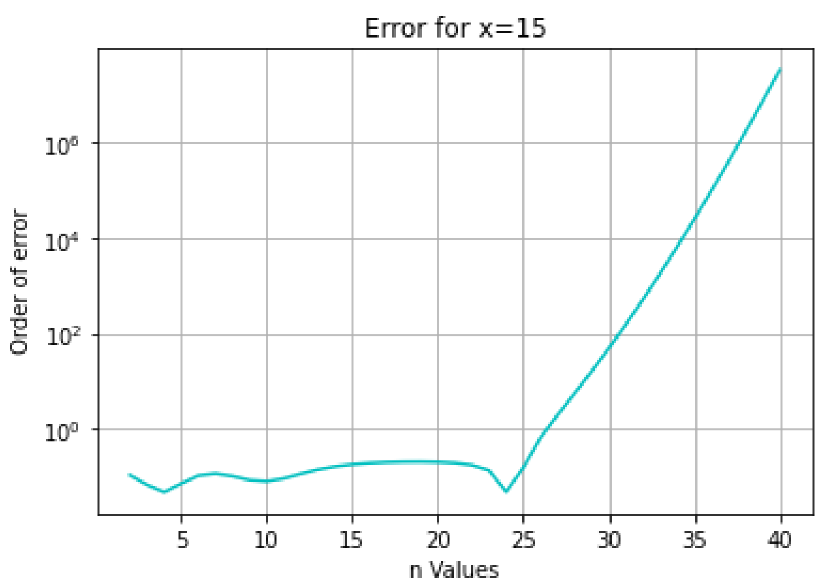

Figure 4: Forward recursion error,x=15

2.2 Backward recursion

```
import matplotlib
matplotlib.use('nbagg')
import matplotlib.pyplot as plt
import numpy as np
import scipy.special as sp

x=1.5
vals1p5=[0,1] #J61, J60
for i in range(60):
        vals1p5.append(2*(60-i)*vals1p5[-1]/x-vals1p5[-2])


alpha=vals1p5[-1]/sp.jv(0,x)
for i in range(len(vals1p5)):
    vals1p5[i]=vals1p5[i]/alpha

vals1p5=vals1p5[::-1][:41]
j2=vals1p5
error1=[]
sum1=0
for n in range(41):
    sum1=sum1+ j2[n]/(1+n)
    error1.append(abs(s(x,n)-sum1))

xval=[i for i in range(41)]
plt.clf()
plt.semilogy(xval,error1,'g')
plt.grid()
plt.xlabel("n Values")
plt.ylabel("Order of Error")
plt.title(" Backward recursion")
plt.show()
```

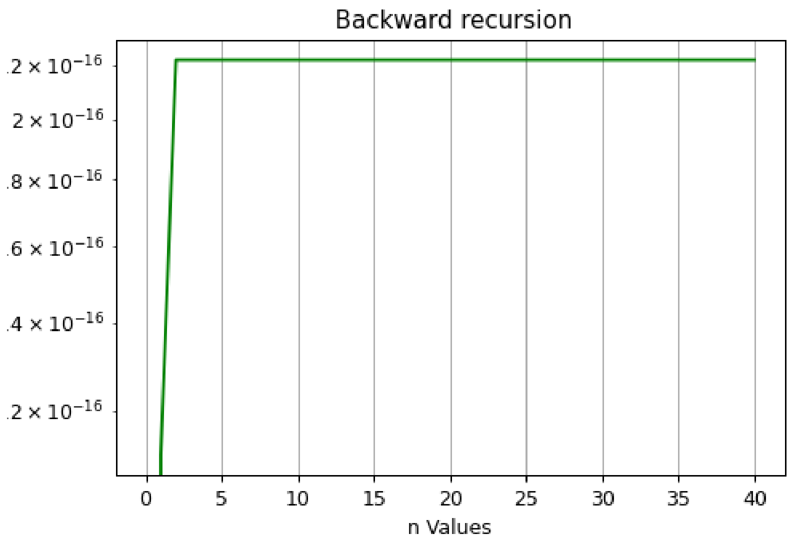

Figure 5: Backward recursion error,x=1.5

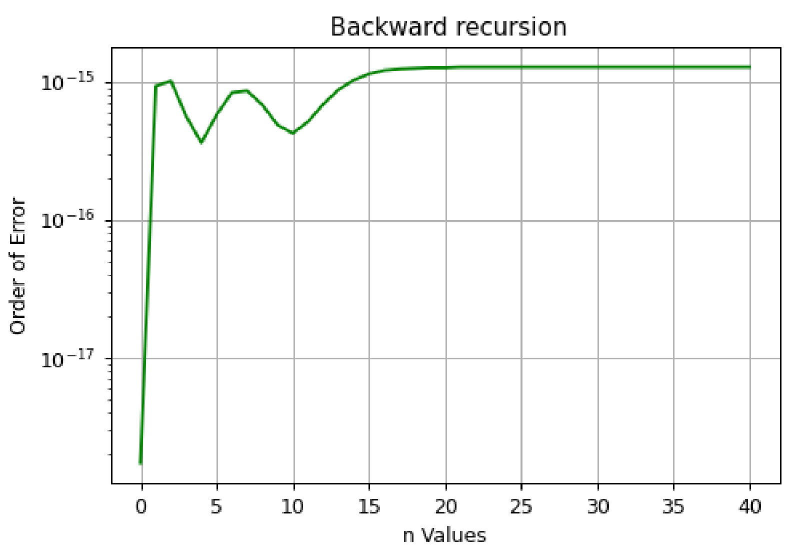

Figure 6: Backward recursion error,x=15

3 Clenshaw Algorithm

The corresponding code for Clenshaw algorithm is below. As predicted it doesn't work well with the J_n.

```
import matplotlib
matplotlib.use('nbagg')
import matplotlib.pyplot as plt
import numpy as np
import scipy.special as sp

def s(x,N):
    y=0.0
    for i in range(N+1):
        y=y+(1/(1+i))*sp.jv(i,x)
    return y

def alpha(x,i):
    return(2*(i+1)/x)

def beta():
    return -1

def c(i):
    return (1/(1+i))

def F(i,x):
    if (i==0):
        return sp.jv(0,x)
    if (i==1):
        return sp.jv(1,x)

func=[]
x=1.5
for n in range(2,41):
    y=[0]*(n+2)
    for k in range(n-1,-1,-1):
        y[k]=alpha(x,k)*y[k+1] + beta()*y[k+2] + c(k)
    func.append(beta()*F(0,x)*y[1] + F(1,x)*y[0] + F(0,x)*c(0))

err=[]
for i in range(2,41):
    err.append(abs(s(x,i)-func[i-2]))

xval=[i for i in range (2,41)]
plt.clf()
```

```
plt.semilogy(xval,err,'m')
plt.xlabel("n Values")
plt.ylabel("Error order")
plt.title("Error for x=1.5")
plt.grid()
plt.show()
```

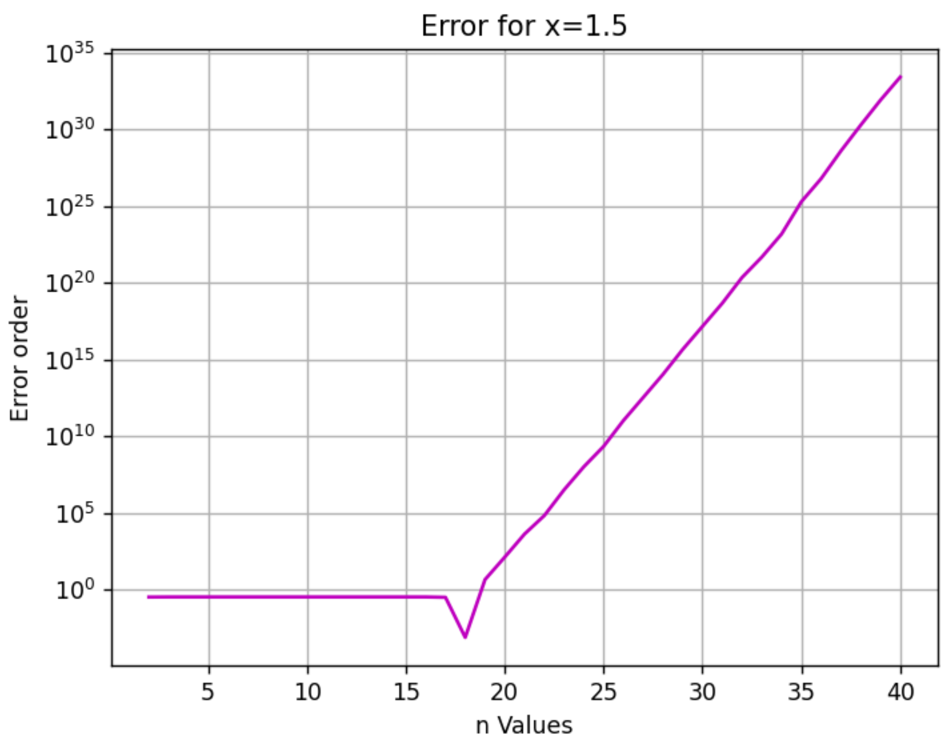

Figure 7: Error using Clenshaw Algorithm, x=1.5

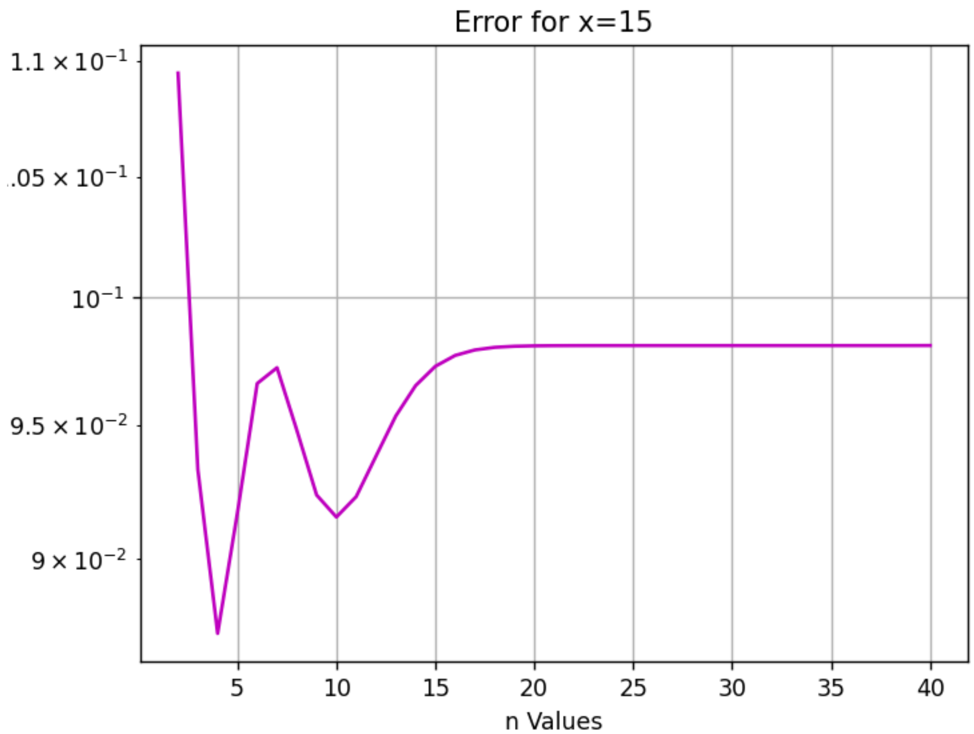

Figure 8: Error using Clenshaw Algorithm, x=15