Student Name: Sang Yang
Student ID: 224642

Numerical Analysis 2022
Assignment 1

1 Question

let $S_N = \sum_{j=2}^{N} \frac{1}{j^2-1}$ which has exact value $\frac{1}{2}\left(\frac{3}{2} - \frac{1}{N} - \frac{1}{N+1}\right)$).There are two methods to compute S_N
(1)$S_N = \frac{1}{2^2-1} + \frac{1}{3^2-1} + \cdots + \frac{1}{N^2-1}$;
(2)$S_N = \frac{1}{N^2-1} + \frac{1}{(N-1)^2-1} + \cdots + \frac{1}{2^2-1}$.

Try to compute $S_{10^2}, S_{10^4}, S_{10^6}$, and compare the computing results and significant figures. (Operation by single floating-point number on computer.)

2 Code

Code compiled by Dev C++

```
#include <stdio.h>
float Sn_1(int n)
  { int i;
    float sum = 0;
    for (i = 2; i < n; i ++)
    {
    sum += 1.0f / ((float)i * (float)i - 1);
    }
    return sum; }
float Sn_2(int n) {
    int i;
    float sum = 0;
    for (i = n; i >= 2; i --)
   {
      sum += 1.0f / ((float)i * (float)i - 1);
    }
    return sum; }
int main()
  { int n;
    while(1) {
        printf("Please enter the value of n: ");
        scanf("%d", &n);
        printf("Sn by Sn1: %f\n", Sn_1(n));
        printf("Sn by Sn2: %f\n", Sn_2(n));
      }
   }
```

Figure 1: The code

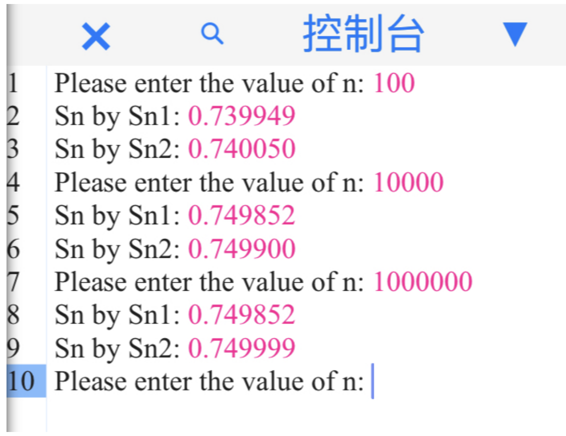

Figure 2: Results

3 Result

Results running by the Dev C++ are above in figure 2.

4 Analyse

It can be seen from the output results of the program that there are indeed errors between the results of different algorithms, and if the direct calculation of the formula is taken as the benchmark, the results calculated from large to small are obviously more accurate. This is because of the problem of "large numbers eat decimals" in the numerical calculation: due to the truncation characteristics of computer calculation, the calculation from large to small will cause the significant places of smaller decimals to be ignored.

Student Name: Sang Yang
Student ID: 224642

Numerical Analysis 2022
Assignment 2

SOUTHEAST UNIVERSITY

---

1 Question

Given equation $f(x) = x^3/3 - x = 0$, it is easy to know it has three roots $x_1^* = -\sqrt{3}$,$x_2^* = 0$, $x_3^* = \sqrt{3}$.

(1) Try to find the $\delta > 0$ as large as possible such that the sequence converges to x_2^* for any $x_0 \in (-\delta, \delta)$;

(2) If the initial data $x_0 \in (-\infty, -1), (-1, -\delta), (-\delta, \delta), (\delta, 1) or (1, +\infty)$, which root does the sequence converge?

(3) What do you understand?

2 Code

Code compiled by Dev C++

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

double f(double x) {
   return x * x * x / 3 - x;
}

double df(double x) {
   return x * x - 1;
}

double newton_iteration(double x0, double max_error,
double (*f) (double), double (*df) (double),
int enalbe_print) {
   double x1 = x0;
double x2 = x1 - f(x1) / df(x1);
int k = 0;

   if (enalbe_print) {
      printf(" k       x(k)\r\n", k, x1);
   }
   while (fabs(x1 - x2) >= max_error) {
      if (enalbe_print) {
         printf("%3d  %13.10f\r\n", k, x1);
      }
      x1 = x2;
      x2 = x1 - f(x1) / df(x1);
      k++;
   }
   return x1;
}
```

Figure 1: The code

```
}

double get_maxium_delta(double step) {
    double x = step;
double y = 0;

    do {
        x += step;
        y = newton_iteration(x, 1e-5, f, df, 0);
    } while (fabs(y) <= step);
    return x;
}

double input_values[] = {-20,  -10,    -5,      -1.1,
                        -0.99, -0.9, -0.85, -0.774599,
                       -0.774, -0.1,    0.1,      0.774,
                      0.774559, 0.85,    0.9,      0.99,
                           1.1,     5,     10,        20};

int main() {
    double res;
    int i;
    printf("****************(1)****************\r\n\r\n");
    res = get_maxium_delta(1e-5);
    printf("Max delta: %.10f\r\n\r\n", res);
    printf("****************(2)****************\r\n\r\n");
    for (i = 0; i < sizeof(input_values) / sizeof(double); i++) {
        newton_iteration(input_values[i], 1e-5, f, df, 1);
        printf("\r\n\r\n");
    }
}
```

Figure 2: The code

3 Result

1.max delta:0.7746000000

2.

$x_0 \in (-\infty, -1)$

k	x(k)	k	x(k)	k	x(k)	k	x(k)
0	-20.0000000000	0	-10.0000000000	0	-5.0000000000	0	-1.1000000000
1	-13.3667502089	1	-6.7340067340	1	-3.4722222222	1	-4.2253968254
2	-8.9613225229	2	-4.5905702249	2	-2.5241804370	2	-2.9840686821
3	-6.0495468647	3	-3.2128401249	3	-1.9960683633	3	-2.2410506211
4	-4.1463281301	4	-2.3716525794	4	-1.7766182124	4	-1.8654706910
5	-2.9349334547	5	-1.9229810688	5	-1.7336735825	5	-1.7451216056
6	-2.2136048174	6	-1.7571748201			6	-1.7321962046
7	-1.8541260399	7	-1.7325795665				
8	-1.7431364747						
9	-1.7321556694						

$x_0 \in (-1, -\delta)$:

k	x(k)	k	x(k)	k	x(k)	k	x(k)
0	−0.9900000000	0	−0.9000000000	0	−0.8500000000	0	−0.7745990000
1	32.5058291457	1	2.5578947368	1	1.4753753754	1	0.7746106540
2	21.6910813342	2	2.0129154848	2	1.8194435475	2	−0.7746805832
3	14.4915209495	3	1.7816615329	3	1.7379691099	3	0.7751003575
4	9.7072379892	4	1.7340488445	4	1.7320809013	4	−0.7776261845
5	6.5409059090					5	0.7930439968
6	4.4649659279					6	−0.8960486959
7	3.1338395799					7	2.4334596439
8	2.3260746019					8	1.9519278837
9	1.9023031006					10	1.7643723922
10	1.7524783955						
11	1.7324025123						

$x_0 \in (-\delta, \delta)$:

k	x(k)	k	x(k)	k	x(k)	k	x(k)
0	−0.7740000000	0	−0.1000000000	0	+0.1000000000	0	+0.7740000000
1	0.7710269677	1	0.0006734007	1	−0.0006734007	1	−0.7710269677
2	−0.7535428201					2	0.7535428201
3	0.6600467532					3	−0.6600467532
4	−0.3396982623					4	0.3396982623
5	0.0295419584					5	−0.0295419584
6	−0.0000172031					6	0.0000172031

$x_0 \in (\delta, 1)$:

k	x(k)	k	x(k)	k	x(k)	k	x(k)
0	0.7745590000	0	0.8500000000	0	0.9000000000	0	0.9900000000
0	0.7745990000	1	−1.4753753754	1	−2.5578947368	1	−32.5058291457
1	−0.7746106540	2	−1.8194435475	2	−2.0129154848	2	−21.6910813342
2	0.7746805832	3	−1.7379691099	3	−1.7816615329	3	−14.4915209495
3	−0.7751003575	4	−1.7320809013	4	−1.7340488445	4	−9.7072379892
4	0.7776261845					5	−6.5409059090
5	−0.7930439968					6	−4.4649659279
6	0.8960486959					7	−3.1338395799
7	−2.4334596439					8	−2.3260746019
8	−1.9519278837					9	−1.9023031006
9	−1.7643723922					10	−1.7524783955
10	−1.7329177972					11	−1.7324025123

$x_0 \in (1, \infty)$:

k	x(k)	k	x(k)	k	x(k)	k	x(k)
0	+1.1000000000	0	+5.0000000000	0	+10.0000000000	0	+20.0000000000
1	4.2253968254	1	3.4722222222	1	6.7340067340	1	13.3667502089
2	2.9840686821	2	2.5241804370	2	4.5905702249	2	8.9613225229
3	2.2410506211	3	1.9960683633	3	3.2128401249	3	6.0495468647
4	1.8654706910	4	1.7766182124	4	2.3716525794	4	4.1463281301
5	1.7451216056	5	1.7336735825	5	1.9229810688	5	2.9349334547
6	1.7321962046			6	1.7571748201	6	2.2136048174
				7	1.7325795665	7	1.8541260399
						8	1.7431364747
						9	1.7321556694

4 Analyse

The above results show that within each interval, the iterative sequence converges, and in $(-\infty, -1)$ the interval converges within $-\sqrt{3}$, that is x1* ; in $(-1, -\delta)$ the interval converges within $\sqrt{3}$, that is x3*; in $(-\delta, \delta)$ the interval converges within 0 , that is x2*; in $(\delta, 1)$the interval converges within $-\sqrt{3}$, that is x*; in $(1, +\infty)$ the interval converges within $\sqrt{3}$, that is x3*

Through this question, I understand that when using the Newton method to solve the roots of multiple equations, the iterative sequence will converge to a certain root in a certain region, so there is a certain interval limit using the Newton method. On a question in one area, the iteration result may Locally converges to different roots.

The Newton iteration method is more demanding for the selection of initial values, so the simple iteration method can be used first when Newton iterates or Try the method, find the range of rough roots, and then carry out Newton iteration.

Student Name: Sang Yang
Student ID: 224642

Numerical Analysis 2022
Assignment 3

1 Question

Program the Gauss elimination with partial pivoting to solve the following linear equations

$$Rx = v \tag{1}$$

where

$$\boldsymbol{R} = \begin{bmatrix} 31 & -13 & 0 & 0 & 0 & -10 & 0 & 0 & 0 \\ -13 & 35 & -9 & 0 & -11 & 0 & 0 & 0 & 0 \\ 0 & -9 & 31 & -10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -10 & 79 & -30 & 0 & 0 & 0 & -9 \\ 0 & 0 & 0 & -30 & 57 & -7 & 0 & -5 & 0 \\ 0 & 0 & 0 & 0 & -7 & 47 & -30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -30 & 41 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5 & 0 & 0 & 27 & -2 \\ 0 & 0 & 0 & -9 & 0 & 0 & 0 & -2 & 29 \end{bmatrix} \tag{2}$$

2 Code

Code compiled by Python

```
import numpy as np

row = 9
col = 9
R = np.array([[ 31, -13,   0,   0,   0, -10,   0,  0,  0],
              [-13,  35,  -9,   0, -11,   0,   0,  0,  0],
              [  0,  -9,  31, -10,   0,   0,   0,  0,  0],
              [  0,   0, -10,  79, -30,   0,   0,  0, -9],
              [  0,   0,   0, -30,  57,  -7,   0, -5,  0],
              [  0,   0,   0,   0,  -7,  47, -30,  0,  0],
              [  0,   0,   0,   0,   0, -30,  41,  0,  0],
              [  0,   0,   0,   0,  -5,   0,   0, 27, -2],
              [  0,   0,   0,  -9,   0,   0,   0, -2, 29]], dtype='float')
V = np.array([[-15], [27], [-23], [0], [-20], [12], [7], [7], [10]], dtype='float')
m = np.concatenate((R, V), axis=1)

for r1 in range(row):
    max_val = max(abs(m[r1:row, r1]))
    for r2 in range(r1 + 1, row):
        if abs(m[r2, r1]) == max_val:
            m1 = np.copy(m[r1, :])
            m[r1, :] = m[r2, :]
            m[r2, :] = m1
    for r2 in range(r1 + 1, row):
        m[r2, :] = m[r2, :] + m[r1, :]*(-m[r2, r1]/m[r1, r1])
print("(1) Gaussian elimination method's result:\r\n", np.around(m, 5))

x = np.zeros((row, 1), dtype='float')
for i in range(row-1, -1, -1):
    sum = 0.0
    for j in range(i+1, col):
        sum = sum + m[i, j] * x[j]
    x[i] = (m[i, col]-sum)/m[i, i]
print('(2) Answer vector of I is:\r\n', np.around(x, 5))
```

Figure 1: The code

3 Result

(1) Gaussian elimination method's result:

$$
\boldsymbol{R} = \begin{pmatrix}
31 & -13 & 0 & 0 & 0 & -10 & 0 & 0 & 0 & -15 \\
0 & 29.54839 & -9 & 0 & -11 & -4.19355 & 0 & 0 & 0 & 20.70968 \\
0 & 0 & 28.25873 & -10 & -3.35044 & -1.27729 & 0 & 0 & 0 & -16.6921 \\
0 & 0 & 0 & 75.46127 & -31.1856 & -0.452 & 0 & 0 & -9 & -5.9069 \\
0 & 0 & 0 & 0 & 44.602 & -7.17969 & 0 & -5 & -3.57799 & -22.3483 \\
0 & 0 & 0 & 0 & 0 & 45.87319 & -30 & -0.78472 & -0.56154 & 8.49257 \\
0 & 0 & 0 & 0 & 0 & 0 & 21.3807 & -0.51319 & -0.36724 & -1.44605 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 26.41308 & -2.42 & 4.6081 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 27.3895 & 7.94922
\end{pmatrix}
$$

(2) Answer vector of $\boldsymbol{x} = \begin{pmatrix} 0.34544 \\ -0.71281 \\ -0.22061 \\ -0.4304 \\ 0.15431 \\ -0.05782 \\ 0.20105 \\ 0.29023 \end{pmatrix}$

Student Name: Sang Yang
Student ID: 224642

Numerical Analysis 2022
Assignment 4

東南大學 SOUTHEAST UNIVERSITY

---

1 Question

Use the RK_4, AB_4, and $AB_4 - AM_4$ predictor-corrector method with $h = 0.1$ to solve the initial-value problem:

$$\begin{cases} y' = -x^2y^2 \quad (0 \leqslant x \leqslant 1.5) \\ y(0) = 3 \end{cases} \tag{1}$$

and compare the numerical solution with exact solutions $y(x) = 3/\left(1 + x^3\right)$

2 Code

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
#include<math.h>

ofstream outfile("data.txt");


double f1(double x,double y)
{
    double f1;
    f1=(-1)*x*x*y*y;
    return f1;
}

double f2(double x)
{
    double f2;
    f2=3/(1+x*x*x);
    return f2;
}

void accurate(double a,double b,double h)

{
    double x[100],accurate[100];
    x[0]=a;
    int i=0;
    outfile<<"输出函数准确值的程序结果:\n";
    do{
        x[i]=x[0]+i*h;
        accurate[i]=f2(x[i]);
        outfile<<"accurate["<<i<<"]="<<accurate[i]<<'\n';
        i++;
    }while(i<(b-a)/h+1);
}

```

Figure 1: Accurate Code

```
void RK4(double a,double b,double h,double c)
{
    int i=0;
    double k1,k2,k3,k4;
    double x[100],y[100];
    y[0]=c;
    x[0]=a;
    outfile<<"输出经典Runge-Kutta公式的程序结果:\n";
    do
    {
        x[i]=x[0]+i*h;
        k1=f1(x[i],y[i]);
        k2=f1((x[i]+h/2),(y[i]+h*k1/2));
        k3=f1((x[i]+h/2),(y[i]+h*k2/2));
        k4=f1((x[i]+h),(y[i]+h*k3));
        y[i+1]=y[i]+h*(k1+2*k2+2*k3+k4)/6;
        outfile<<"y"<<"["<<i<<"]="<<y[i]<<'\n';
        i++;
    }while(i<(b-a)/h+1);
}
```

Figure 2: RK4 Code

```
void AB4(double a,double b,double h,double c)
{
    double x[100],y[100],y1[100];
    double k1,k2,k3,k4;
    y[0]=c;
    x[0]=a;
    outfile<<"输出4阶Adams显式方法的程序结果:\n";
    for(int i=0;i<=2;i++)
    {
        x[i]=x[0]+i*h;
        k1=f1(x[i],y[i]);
        k2=f1((x[i]+h/2),(y[i]+h*k1/2));
        k3=f1((x[i]+h/2),(y[i]+h*k2/2));
        k4=f1((x[i]+h),(y[i]+h*k3));
        y[i+1]=y[i]+h*(k1+2*k2+2*k3+k4)/6;
    }
    int j=3;
    y1[0]=y[0];
    y1[1]=y[1];
    y1[2]=y[2];
    y1[3]=y[3];
    do
    {
        x[j]=x[0]+j*h;
        y1[j+1]=y1[j]+(55*f1(x[j],y1[j])-59*f1(x[j-1],y1[j-1])
        +37*f1(x[j-2],y1[j-2])-9*f1(x[j-3],y1[j-3]))*h/24;
        outfile<<"y1"<<"["<<j<<"]="<<y1[j]<<'\n';
        j++;
    }while(j<(b-a)/h+1);
}

void main(void)
{
    double a,b,h,c;
    cout<<"输入上下区间、步长和初始值:\n";
    cin>>a>>b>>h>>c;
    accurate(a,b,h);
    RK4(a,b,h,c);
    AB4(a,b,h,c);
}
```

Figure 3: AB4 Code

3 Result

The results obtained by RK_4 are as follows.

i	x_i	y	$y(x_i)$	$\lvert y(x_i) - y_i \rvert$
0	0	3	3	0
1	0.1	2.997	2.997	1.87138 e-007
2	0.2	2.97619	2.97619	3.91665 e-007
3	0.3	2.92113	2.92113	7.58342 e-007
4	0.4	2.81955	2.81955	1.61101 e-006
5	0.5	2.66666	2.66667	3.17735 e-006
6	0.6	2.4671	2.46711	5.00551 e-006
7	0.7	2.2338	2.2338	5.77233 e-006
8	0.8	1.98412	1.98413	4.12954 e-006
9	0.9	1.73511	1.73511	1.15554 e-007
10	1.0	1.50001	1.5	5.80668 e-006
11	1.1	1.28701	1.287	1.13075 e-005
12	1.2	1.09972	1.09971	1.54242 e-005
13	1.3	0.938397	0.93838	1.77272 e-005
14	1.4	0.8013	0.801282	1.83754 e-005
15	1.5	0.685732	0.685714	1.78 e-005

The results obtained by AB_4 are as follows.

y1[0] = 3	y1[1] = 2.997	y1[2] = 2.97619	y1[3] = 2.92113	y1[4] = 2.81839
y1[5] = 2.66467	y1[6] = 2.4652	y1[7] = 2.23308	y1[8] = 1.98495	y1[9] = 1.73704
y1[10] = 1.50219	y1[11] = 1.28876	y1[12] = 1.10072	y1[13] = 0.93871	y1[14] = 0.801135
y1[15] = 0.685335				

RK_4, AB_4 both have 4 degrees of accuracy.

Student Name: Sang Yang
Student ID: 224642

Numerical Analysis 2022
Assignment 5

1 Question

(1) Program Crank-Nicolson method to approximate the solution for the parabolic equation with $a=1$, $f(x,t)=0$, $\varphi(x)=\exp x$, $\alpha(t)=\exp t$, $\beta(t)=\exp(1+t)$, $M=40$, $N=40$. Output the numerical solutions on the mesh points (0.2,1.0), (0.4,1.0), (0.6,1.0), (0.8,1.0) .

(2) Compare the numerical results at (0.2,1.0),(0.4,1.0),(0.6,1.0),(0.8,1.0) to the actual solution $u(x,t)=\exp(x+t)$ with $M=N=40,80,160$.

2 Code

```
#include <iostream>
#include <math.h>

float h=0.025,k=0.025;
int m=40;int n=40;
float y[40][40],r=a*k/(h*h);

void Input()
{
  int i,j;
  cout<<"Loading Input Data..."<<endl;
  for(i=0;i<m;i++)
    {
      for(j=0;j<n;j++)
        if (i==j) a[i][j]=1+r;
      for(j=0;j<n;j++)
        if ((j=i+1)||(i=j+1)) a[i][j]=-r/2;
    }
}

int main()
{
  Input(); //read data
  int r,i;
  for(k=0;k<(m-1);k++)
    {
      int select();  //select main element
      r=select();
      void exchange(int g);
      exchange(r);  //exchange
      void analyze();
      analyze();  //analyze
    }
  void ret();
  ret();  // replace back

  cout<<"The solution vector is below:"<<endl;
  for(i=0;i<m;i++)
    cout<<"x["<<i<<"]="<<x[i]<<endl;
  return 0;
}

int select()
{
  int f,t; float max;
  f=k;
  float si(int u,int v);
  max=float(fabs(si(k,k)));
  for(t=(k+1);t<(m-1);t++)
    {
      if(max<fabs(si(t,k)))
    {
      max=float(fabs(si(t,k)));
      f=t;
    }
    }
  return f;
```

```
float si(int u,int v)
{
  float sum=0; int q;
  for(q=0;q<k;q++)
    sum+=a[u][q]*a[q][v];
  sum=a[u][v]-sum;
  return sum;
}

void exchange(int g)
{
  int t; float temp;
  for(t=0;t<n;t++)
    {
      temp=a[k][t];
      a[k][t]=a[g][t];
      a[g][t]=temp;
    }
}

void analyze()
{
   int t;
   float si(int u,int v);
   for(t=k;t<n;t++)
     a[k][t]=si(k,t);
   for(t=(k+1);t<m;t++)
     a[t][k]=(float)(si(t,k)/a[k][k]);
}

void ret()
{
   int t,z;float sum;
   x[m-1]=(float)a[m-1][m]/a[m-1][m-1];
   for(t=(m-2);t>-1;t--)
     {
       sum=0;
       for(z=(t+1);z<m;z++)
         sum+=a[t][z]*x[z];
       x[t]=(float)(a[t][m]-sum)/a[t][t];
     }
}
```

Figure 1: Code

3 Result

Results obtained by this program are listed in table.

Coordinate	NumericalResult	AccurateValue	Errors
(0.2,1.0)	3.31947	3.32012	0.00045
(0.4,1.0)	4.05432	4.05520	0.00088
(0.6,1.0)	4.95238	4.95303	0.00065
(0.8,1.0)	4.04897	6.04965	0.00067