Homework assignments sheet 2
Paul Monderkamp 2321677

Exercise 2.1

code:

```
#include <iostream>
#include <fstream>
#include <cmath>

using namespace std;
const double pi = 3.14159265358979323846264338327;

int main()
{
double t0			= 0.0;
double tmax	= 50.0;
double dt			= 0.1;
double x0			= 1.0;
double v0			= 0.0;

int N = (int)(tmax - t0)/dt + 1;

double y[N][3];

y[0][0] = x0;
y[0][1] = v0;
y[0][2] = 0.5*(y[0][0]*y[0][0]+y[0][1]*y[0][1]);
ofstream out("problem_2_1.txt");

out << 0 << "	" << y[0][0] << " " << y[0][1] << " " << y[0][2] << endl;

for (int i = 0; i<N; i++)
	{
		y[i+1][0] = y[i][0] + dt * y[i][1];
		y[i+1][1] = y[i][1] - dt * y[i][0];
		y[i+1][2] = 0.5*(y[i+1][0]*y[i+1][0]+y[i+1][1]*y[i+1][1]);
		out << i+1 << "   " << y[i+1][0] << "		" << y[i+1][1] << "		" << y[i+1][2] << endl;
	}

//cout << N << endl;


out.close()

return 0;
}
```

Results:

a)

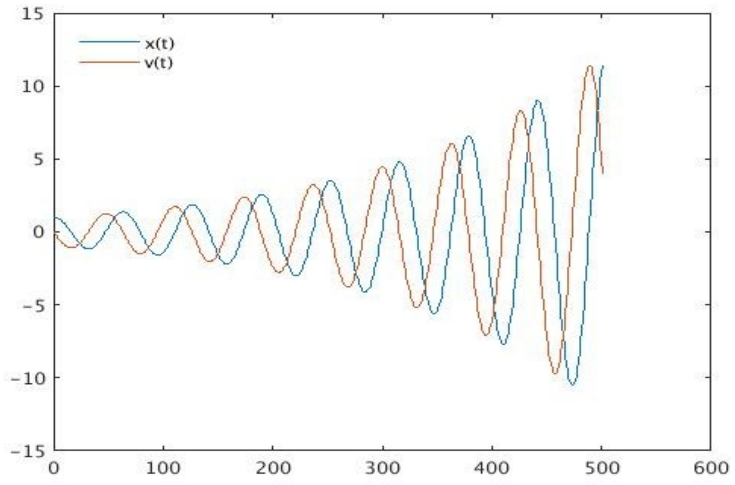


Figure 1

Figure 1 shows x(t) and v(t) as solution of the harmonic oscillator equation with the given initial conditions from the exercise, solved via euler algorithm. The y axis shows x(t) and v(t) in dimensionless units. The x axis of the diagram shows the number of euler steps where 500 euler steps correspond to t = 50 in dimensionless units, hence a stepsize of 0.1.

It is clearly visible that the algorithm is not stable since the amplitude should not chance.

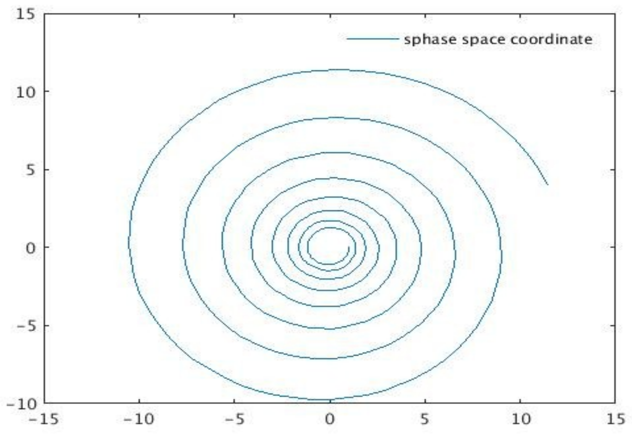


Figure 2

Figure 3 shows the trajectory of the system in phase space where the x axis corresponds to the position and the y axis corresponds to the momentum/ velocity which is equivalent due to the mass m being set to 1. Its visible how the energy diverges as for the harmonic oscillator the total energy is equal to 0.5* (x(t)^2 + v(t)^2) which is propotional to the norm of the phase space coordinate squared.

b)

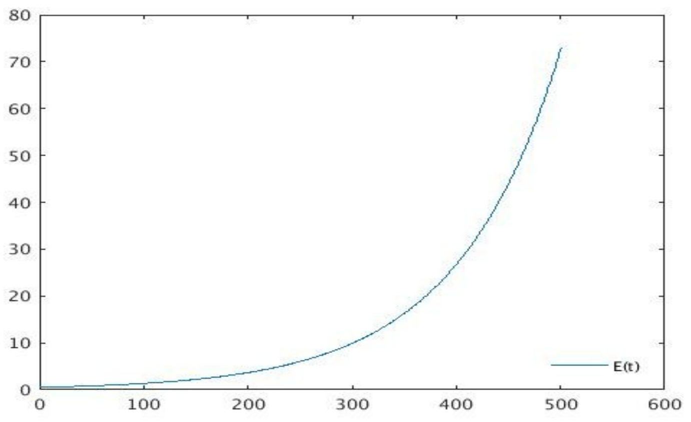


Figure 3

Figure 3 shows the Energy of the system as a function of the stepnumber. Correspondingly the energy should be conserved, which is visibly once again not the case for the Euler algorithm.

Exercise 2.2

code:

```
#include <iostream>
#include <fstream>
#include <cmath>

using namespace std;

const double pi = 3.14159265358979323846264338327;

int main()
{
double t0                = 0.0;
double tmax      = 50.0;
double dt                = 0.2;
double x0                = 1.0;
double v0                = 0.0;

int N = (int)(tmax - t0)/dt + 1;

double y[N][3];

y[0][0] = x0;
y[0][1] = v0;
y[0][2] = 0.5*(y[0][0]*y[0][0]+y[0][1]*y[0][1]);
ofstream out("problem_2_2.txt");

out << 0 << "      " << y[0][0] << " " << y[0][1] << " " << y[0][2] << endl;

for (int i = 0; i<N; i++)
        {
                y[i+1][0] = y[i][0] + dt * y[i][1];
                y[i+1][1] = y[i][1] - dt * y[i+1][0];
                y[i+1][2] = 0.5*(y[i+1][0]*y[i+1][0]+y[i+1][1]*y[i+1][1]);
                out << i+1 << "   " << y[i+1][0] << "          " << y[i+1][1] << "          " << y[i+1][2] << endl;
        }

//cout << N << endl;


out.close();

return 0;
}
```

Results:

a)

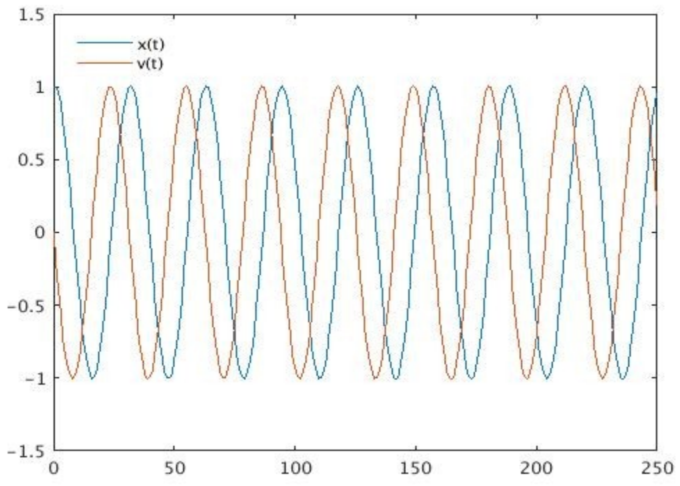


Figure 4

Figure 4 shows the solutions of the harmonic oscillator for x(t) and v(t) in dimensionless units plottet against stepn number. These solutions are acquired via Euler-Cromer algorithm. The global time frame is the same, but the step size is twice as big such that the number of steps reduces by a factor 2. It is clearly visible how the solution stays stable.

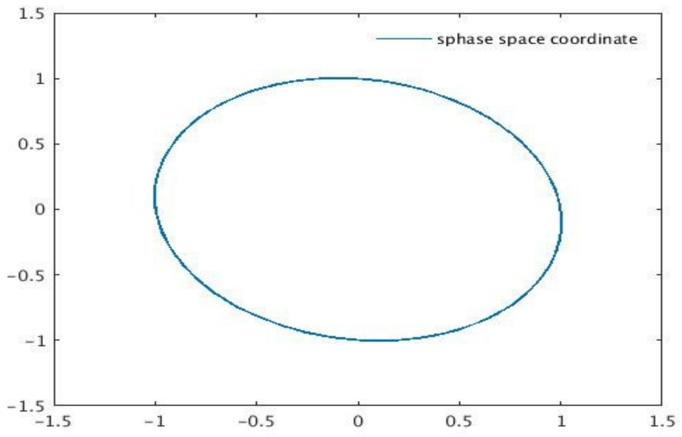


Figure 5

Figure 5 shows the phase space trajectory of the system. The exact solution of the harmonic oscillator shows an ellipse with the priciple axes parallel to the cordinate axes. This is visibly not the case in this phase space portrait. This is due to the fact that as discussed in the lecture, the energy of the numerical solution oscillates around the energy of the exact solution while the system propagates. This corresponds to a time dependent norm of the coordinate in phase space.

b)

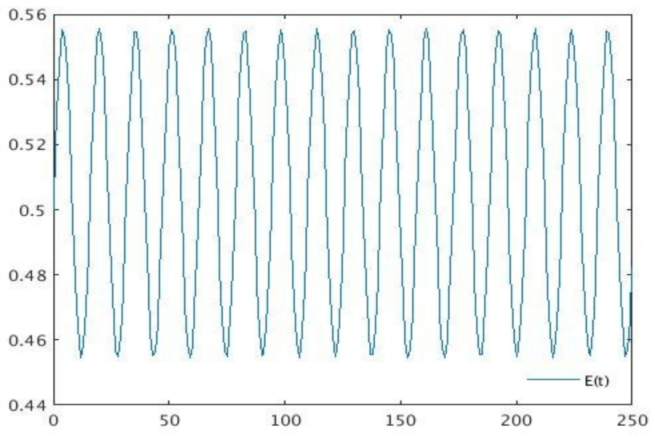
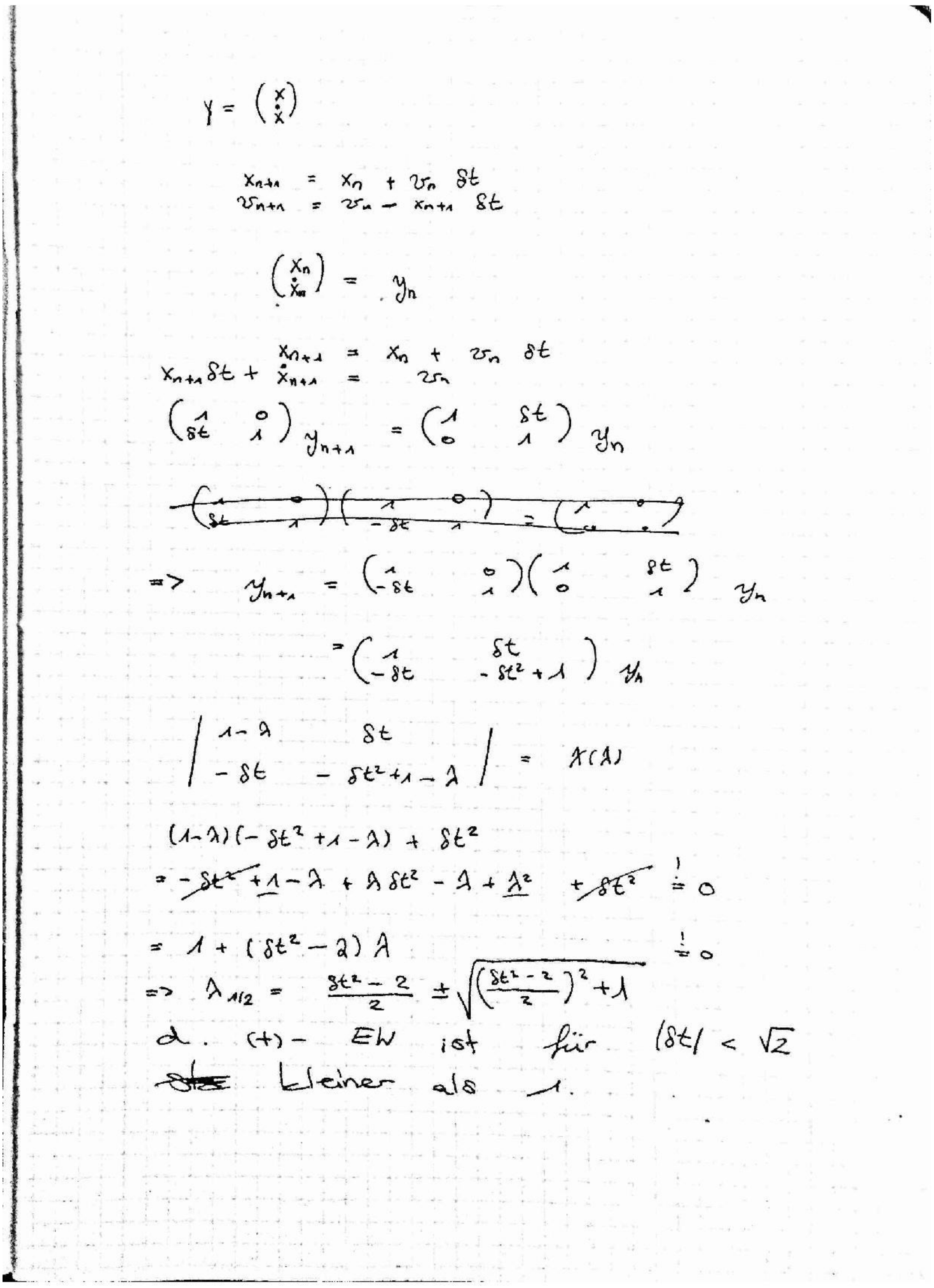


Figure 6

Figure 6 shows the total energy as a function of stepnumber. It is visible that the energy does not diverge as with the Euler Algorithm but stays bound to the exact value of 0.5. As mentioned above the numerical solution for the energy oscillates around the exact solution.

c) The calculation of the eigenvalues shows that the algorithm is stable for dt < sqrt(2)



$$y = \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

$$x_{n+1} = x_n + v_n \, \delta t$$
$$v_{n+1} = v_n - x_{n+1} \, \delta t$$

$$\begin{pmatrix} x_n \\ \dot{x}_n \end{pmatrix} = y_n$$

$$x_{n+1} = x_n + v_n \, \delta t$$
$$x_{n+1} \delta t + \dot{x}_{n+1} = v_n$$

$$\begin{pmatrix} 1 & 0 \\ \delta t & 1 \end{pmatrix} y_{n+1} = \begin{pmatrix} 1 & \delta t \\ 0 & 1 \end{pmatrix} y_n$$

~~$\begin{pmatrix} 1 & 0 \\ \delta t & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -\delta t & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$~~

$$\Rightarrow \quad y_{n+1} = \begin{pmatrix} 1 & 0 \\ -\delta t & 1 \end{pmatrix} \begin{pmatrix} 1 & \delta t \\ 0 & 1 \end{pmatrix} y_n$$

$$= \begin{pmatrix} 1 & \delta t \\ -\delta t & -\delta t^2 + 1 \end{pmatrix} y_n$$

$$\begin{vmatrix} 1-\lambda & \delta t \\ -\delta t & -\delta t^2 + 1 - \lambda \end{vmatrix} = \chi(\lambda)$$

$$(1-\lambda)(-\delta t^2 + 1 - \lambda) + \delta t^2$$
$$= \cancel{-\delta t^2} + \underline{1} - \lambda + \lambda \delta t^2 - \lambda + \underline{\lambda^2} \quad \cancel{+\delta t^2} \overset{!}{=} 0$$

$$= 1 + (\delta t^2 - 2)\lambda \quad \overset{!}{=} 0$$

$$\Rightarrow \lambda_{1/2} = \frac{\delta t^2 - 2}{2} \pm \sqrt{\left(\frac{\delta t^2 - 2}{2}\right)^2 + 1}$$

d. (+) - EW ist für $|\delta t| < \sqrt{2}$

~~δt~~ kleiner als 1.

Exercise 2.3 a)

$$\int_{-1}^{1} f(x)\,dx \approx A_1 f(x_1) + A_2 f(x_2) + A_3 f(x_3) + A_4 f(x_4)$$

Due to symmetry in coefficients, roots and the integral it is sufficient to show

$$\int_0^1 f(x)\,dx \approx A_3 f(x_3) + A_4 f(x_4)$$

from definition: $A_3 + A_4 = 1$

$$\int_0^1 f(x)\,dx \approx \int_0^1 \left[f(x_3) + (x - x_3)\lambda\right] \cdot C_3 + \left[f(x_4) + (x - x_4)\lambda\right] \cdot C_4 \; dx$$

two Taylor exp. at diff nodes;

$C_3 + C_4 \overset{!}{=} 1$ such that in the limit of infinite taylor terms it equates to $f(x)$

$$= C_3 \left(f(x_3) \cdot (1-0) + \frac{(x-x_3)^2}{2}\Big|_0^1 \right) + C_4 \left(f(x_4) \cdot (1-0) + \frac{(x-x_4)^2}{2}\Big|_0^1 \right)$$

$$= C_3 f(x_3) + C_4 f(x_4) + \underbrace{C_3 \frac{(1-x_3)^2}{2} - C_3 \frac{x_3^2}{2} + C_4 \frac{(1-x_4)^2}{2} - C_4 \frac{x_4^2}{2}}_{(1)}$$

$$c_3 \left(\tfrac{1}{2} - x_3\right) + c_4 \left(\tfrac{1}{2} - x_4\right) \overset{!}{=} 0$$

$$c_4 = 1 - c_3$$

$$c_3\left(\tfrac{1}{2} - x_3\right) + (1 - c_3)\left(\tfrac{1}{2} - x_4\right) = 0$$

$$\cancel{\frac{c_3}{2}} - x_3 c_3 + \frac{1}{2} - x_4 - \cancel{\frac{c_3}{2}} + c_3 x_4 = 0$$

$$c_3 (x_4 - x_3) + \frac{1}{2} - x_4 = 0$$

$$\Rightarrow \quad c_3 = \frac{x_4 - 1/2}{x_4 - x_3}$$

c_3 is chosen such that the approximation to the integral is equal to $c_3 f(x_3) + c_4 f(x_4)$ where the leading correction term vanishes

b)

Code:

```
#include <iostream>
#include <cmath>
#include <fstream>

using namespace std;

double f(double x)
{
return((35.0*pow(x,4.0)+0.0*pow(x,3.0)-30.0*pow(x,2.0)-0.0*pow(x,1.0)+3.0)/8.0);
}


double fstrich(double x)
{
return((140.0*pow(x,3.0)+0.0*pow(x,2.0)-60.0*pow(x,1.0)-0.0)/8.0);
}


int main()
{
ofstream out("root4.txt");
double actualroot = 0.861136311594053;
//ddouble actualroot = 0.339981043584856;

double epsilon = 1.0e-15;
cout << "tolerance in f(x) if what is equal to zero = " << epsilon << endl;

            int counter = 0;
            double x0 = 0.7;


            while (fabs(f(x0)) > epsilon)
                  {
                  out << counter << "          " << fabs(x0 - actualroot) << endl;
                  cout << counter << "         " << fabs(x0 - actualroot) << endl;
                  cout << "x0 = " << x0 << endl;
                  //cout << fabs(f(x0)) << endl;
                  x0 = x0 - f(x0)/fstrich(x0);
                  counter++;
                  }

out.close();
getchar();
return 0;
}
```

Results:

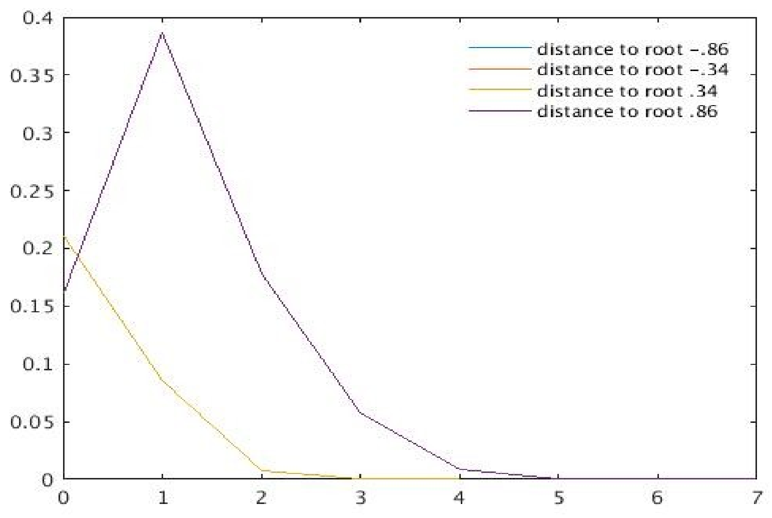


Figure 7

Figure 7 shows the difference of the iterative numerical result for the roots of the 4th Legendre polynomial to the exact result as a function of iteration steps. This data is generated via the Newton-Raphson method. Due to the symmetry of the function and the choice of starting points for the algorithm 2 graphs each lie exactly on top of each other. The graphs for roots appr. +/- .86 and the graphs for roots appr. +/- .34 are identical.

c)

code:

Gauss-Legendre Quadrature

```
#include <iostream>
#include <cmath>
#include <fstream>

using namespace std;

double f(double u)
{
        double x = 2.0;
        return ((1.0/sqrt(M_PI))*exp(-(0.5*x*(1.0+u))*(0.5*x*(1.0+u)))*x);
}



double quadrature()
{
//returns the result of the integral from -1 to 1

double a = (0.347854845137454*f(-0.861136311594053)+
                0.652145154862546*f(-0.339981043584856)+
                0.652145154862546*f(0.339981043584856) +
                0.347854845137454*f(0.861136311594053));

return(a);
}
```

Bool's rule

```
#include <iostream>
#include <cmath>
#include <fstream>

using namespace std;

double f1(double x)
{

        return ((2.0/sqrt(M_PI))*exp(-x*x));
}



double boolsrule(double a, double b)
{
double h = (b - a)/4.0;
double x[5];
for (int i = 0; i < 5; i++)
        {
                x[i] = a + i*h;
        }

double c = (2.0*h/45.0)*(7.0*f1(x[0])+32.0*f1(x[1])+12.0*f1(x[2])+32.0*f1(x[3])+7.0*f1(x[4]));

return(c);
}
```

The main function which gives out the corresponding values only consists of several lines writing the result into a .txt file.

The substitution necessary to compute the value for the error function is described below:

$$f(x) = \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$$

$$u \in [-1, 1]$$

$$\frac{x}{2} + \frac{x}{2} \cdot u = t$$

$$\frac{x}{2}(1 + u) = t \quad \rightarrow \quad u = \frac{2}{x} t - 1$$

$$t = 0 \Rightarrow u = -1$$
$$t = x \Rightarrow u = 1$$

$$\cancel{du} \quad \frac{dt}{du} = \frac{x}{2}$$

$$\Rightarrow \cancel{dt} = \frac{x}{2}\, du$$

$$\Rightarrow \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{-1}^{1} \underbrace{e^{-[\frac{x}{2}(1+u)]^2} \frac{x}{2}}\, du$$

d) **Results:**

the output from the main function into the .txt file is as follows:

```
quadrature: 0.995489
boolsrule: 0.998921

deviation from exact value:
quadrature: -0.000166667
boolsrule: -0.00359829

relative error:
quadrature: -1.67451e-19
boolsrule: -3.6152e-18
```

which gives an approximative numerical result for erf(2).
It can be seen that the relative error for the quadrature method is about 20 times smaller as for Bool's method.