```
%% Initialisierung

% g = sinint(t)' = sin(t)/t =: f

% Näherung für sin(t)/t bei t=0
% Geschlossene Newton-Cotes Formeln (Trapez, Simpson) sonst nicht anwendbar
%f = @(t) sin(t)/((t>0).*t + (t==0).*10^-25);

% Originalfunktion - Trapez und Simpson ohne Ergebnis
%f = @(t) sin(t)/t;

% Sinc-Funktion mit sinc(x)=sinx/x und sinc(0)=1
f = @(t) (t>0)*sin(t)/(t+(t==0))+(t==0);
a = 0;
b = pi;

m_vec = 2.^(1:5);


I_matlab = sinint(b);


%% Mittelpunkt (a) - falsch
F_vec = [];
for m = m_vec

    h = (b-a)/m;

    I = 0;

    for i = 1:m

        xi = a + (i-1)*h;

        xm = xi + h/2;

        gm = f(xm);

        I = I + gm*h;

    end

    F_vec = [F_vec I_matlab-I];
end
disp("Mittelpunkt: " + approx_ordnung(F_vec));

%% Trapez (b) - richtig
% ca. um Faktor 2 verschieden, außerdem geschlossene Newton Cotes Formel
% (nur bei sin(x)/x relevant)
F_vec = [];
for m = m_vec

    I = 0;

    h = (b-a)/m;

    for i = 1:m

        xi = a + (i-1)*h;
        x0 = xi;
        x1 = xi + h;

        g0 = f(x0);
        g1 = f(x1);

        I = I + g0+g1;
    end

    I = I * h/2;

    F_vec = [F_vec I_matlab-I];
end
```

```
disp("Trapez: " + approx_ordnung(F_vec));

%% Simpson (c) - falsch
% ca. um Faktor 2^2 verschieden, außerdem geschlossene Newton Cotes Formel
% (nur bei sin(x)/x relevant)
F_vec = [];
for m = m_vec

    h = (b-a)/m;

    I = 0;

    for i = 1:m

        xi = a + (i-1)*h;
        x0 = xi;
        x1 = xi + h/2;
        x2 = xi + h;

        g0 = f(x0);
        g1 = f(x1);
        g2 = f(x2);

        I = I + g0+4*g1+g2;

    end

    I = I * h/6;

    F_vec = [F_vec I_matlab-I];
end
disp("Simpson: " + approx_ordnung(F_vec));

%% Milne (d) - richtig
F_vec = [];
for m = m_vec

    h = (b-a)/m;

    I = 0;

    for i = 1:m

        xi = a + (i-1)*h;
        x0 = xi + h/4;
        x1 = xi + h/2;
        x2 = xi + 3*h/4;

        g0 = f(x0);
        g1 = f(x1);
        g2 = f(x2);

        I = I + 2*g0 - g1 + 2*g2;

    end

    I = I * h/3;

    F_vec = [F_vec I_matlab-I];
end
disp("Milne: " + approx_ordnung(F_vec));


%% Hilfsfunktion
% Zur Bestimmung der Ordnung Omega
function w = approx_ordnung(v)
    count = size(v,2)-1;

    F = zeros(count,1);
    for i=1:count
        F(i) = abs(v(i)/v(i+1));
    end
```

```
    w = log2(mean(F));
end
```

```
clear, clc

% Übungsblatt 10 - Beispiel 4
% A und C richtig.

% Werte für die Integration von f,g
Value_f_pi = 1.85193705198246;
Value_g_1 = 2/3;

% Matrixaufbau (5x5)
N_g = zeros(5,5);
N_f = zeros(5,5);

a = 0;
b_1 = pi;
b_2 = 1;
m = 1; % Startwert für m
h_1 = (a+b_1)/m;
h_2 = (a+b_2)/m;

for Index1 = 1:(m+1)
    % Erstelle Schrittweite t
    t_f(Index1) = a+(Index1-1)*h_1;
    t_g(Index1) = a+(Index1-1)*h_2;
end

for Index1 = 1:(m)
    % Mittelpunkte
    t_bar_f(Index1) = (t_f(Index1)+t_f(Index1+1))/2;
    t_bar_g(Index1) = (t_g(Index1)+t_g(Index1+1))/2;

    % Berechnung von N_0(h)
    N_f(1,1) = N_f(1,1) + h_1*f(t_bar_f(Index1));
    N_g(1,1) = N_g(1,1) + h_2*g(t_bar_g(Index1));
end

for Index1 = 2:5
    % Berechnung des linken bzw. rechten Drittels
    t_cup_f = t_bar_f - h_1/3;
    t_hat_f = t_bar_f + h_1/3;
    t_cup_g = t_bar_g - h_2/3;
    t_hat_g = t_bar_g + h_2/3;

    % Berechnung von N_0(h/3), N_0(h/9), ...
    N_f(Index1,1) = 1/3*(N_f(Index1-1,1) + h_1*sum(f(t_cup_f)) +
 h_1*sum(f(t_hat_f)));
    N_g(Index1,1) = 1/3*(N_g(Index1-1,1) + h_2*sum(g(t_cup_g)) +
 h_2*sum(g(t_hat_g)));

    for Index2=2:Index1
        % Richardson Extrapolation (Genauigkeit)
```

```
        N_g(Index1,Index2) = (9^(Index2-1)*N_g(Index1,Index2-1) -
 N_g(Index1-1,Index2-1))/(9^(Index2-1)-1);
        N_f(Index1,Index2) = (9^(Index2-1)*N_f(Index1,Index2-1) -
 N_f(Index1-1,Index2-1))/(9^(Index2-1)-1);
    end
    % Nächster Schritt in Romberg
    m = 3*m;

    %Speichern der Werte als neue Mittelpunkte
    t_bar_f = [t_cup_f, t_bar_f, t_hat_f];
    t_bar_g = [t_cup_g, t_bar_g, t_hat_g];

    % h wird auf neue Stufe reduziert
    h_1 = h_1/3;
    h_2 = h_2/3;
end

% Berechnung des Errors
Error_g = Value_g_1 - N_g;
Error_f = Value_f_pi - N_f;

% b) und d)
g_5_5 = double(abs(Error_g(5,5)));
f_4_4 = double(abs(Error_f(4,4)));

for Index = 3:4
   Error_f_temp(Index-2) = Error_f(Index,3)/Error_f(Index+1,3);
end
for Index = 2:4
   Error_g_temp(Index-1) = Error_g(Index,2)/Error_g(Index+1,2);
end

Error_f_log = mean(Error_f_temp);
Error_g_log = mean(Error_g_temp);

% a) und c)
Error_f_log = round(log(Error_f_log)/log(3),2);
Error_g_log = round(log(Error_g_log)/log(3),2);

% print
fprintf("a) Error von f (zu w3) = %0.2f \nb) f_4,4 = %d \nc) Error von
 g (zu w2) = %0.2f  \nd) g_5,5 = %d \n",...
    Error_f_log, f_4_4, Error_g_log, g_5_5);


% Funktionen g und f
function Return = g(t)
    Return = sqrt(t);
end

function Return = f(x)
    Return = sin(x)./x;
end
```

```
a) Error von f (zu w3) = 6.09
b) f_4,4 = 7.207792e-10
c) Error von g (zu w2) = 1.50
d) g_5,5 = 3.738983e-05
```

*Published with MATLAB® R2017a*

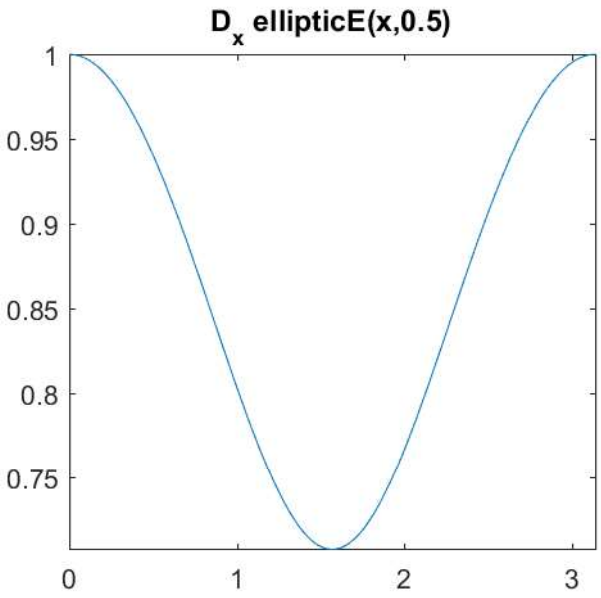
```
% Numerik 1 Blatt 10 Beispiel 6
clear
% Initialisierung der x_n,i und a_n,i (Wikipedia)
x_00 = 0; a_00 = 2;
x_10 = - sqrt(1/3);                     a_10 = 1;
x_11 = + sqrt(1/3);                     a_11 = 1;
x_20 = - sqrt(3/5);                     a_20 = 5/9;
x_21 = 0;                               a_21 = 8/9;
x_22 = + sqrt(3/5);                     a_22 = 5/9;
x_30 = - sqrt(3/7 + 2/7*sqrt(6/5));     a_30 = (18 - sqrt(30))/36;
x_31 = - sqrt(3/7 - 2/7*sqrt(6/5));     a_31 = (18 + sqrt(30))/36;
x_32 = + sqrt(3/7 - 2/7*sqrt(6/5));     a_32 = (18 + sqrt(30))/36;
x_33 = + sqrt(3/7 + 2/7*sqrt(6/5));     a_33 = (18 - sqrt(30))/36;
x_40 = - 1/3*sqrt(5 + 2*sqrt(10/7));    a_40 = (322 - 13*sqrt(70))/900;
x_41 = - 1/3*sqrt(5 - 2*sqrt(10/7));    a_41 = (322 + 13*sqrt(70))/900;
x_42 = 0;                               a_42 = 128/225;
x_43 = + 1/3*sqrt(5 - 2*sqrt(10/7));    a_43 = (322 + 13*sqrt(70))/900;
x_44 = + 1/3*sqrt(5 + 2*sqrt(10/7));    a_44 = (322 - 13*sqrt(70))/900;

X = [x_00,0,0,0,0;x_10,x_11,0,0,0;x_20,x_21,x_22,0,0;..
    x_30,x_31,x_32,x_33,0;x_40,x_41,x_42,x_43,x_44];
A = [a_00,0,0,0,0;a_10,a_11,0,0,0;a_20,a_21,a_22,0,0;..
    a_30,a_31,a_32,a_33,0;a_40,a_41,a_42,a_43,a_44];

% erstellen der Funktionen
f = @(x) sqrt(1-0.5*sin(x).^2);
% f(x) = D_x ellipticE(x,0.5)
g = @(x) 1./sqrt(1-0.99*sin(x).^2);
% g(x) = D_x ellipticF(x,0.99)

df = @(x) -sin(x)*cos(x)./(2*f(x));
dg = @(x) 0.99*sin(x)*cos(x)*(g(x)).^3;
% Ableitungen von f und g

m = [1,2,4];
a = 0; b = pi;      % Integralsgrenzen

subplot(2,2,1)
fplot(f,[a,b]);
title('D_x ellipticE(x,0.5)')
subplot(2,2,3)
fplot(df,[a,b]);
title('D_{xx} ellipticE(x,0.5)')
subplot(2,2,2)
fplot(g,[a,b]);
title('D_x ellipticF(x,0.99)')
subplot(2,2,4)
fplot(dg,[a,b]);
title('D_{xx} ellipticF(x,0.99)')

% Unterpunkt a
I_1a = [];          % approximiertes Integral mit Gauss-Legendre-Quadratur
n=1;
for mj=1:length(m)
    sumj = 0;
    for j=1:m(mj);
        sumi = 0;
        for i=0:n;
            Gj = transf(f,m(mj),j,a,b);
            sumi = sumi + A(n+1,i+1)*Gj(X(n+1,i+1));
```

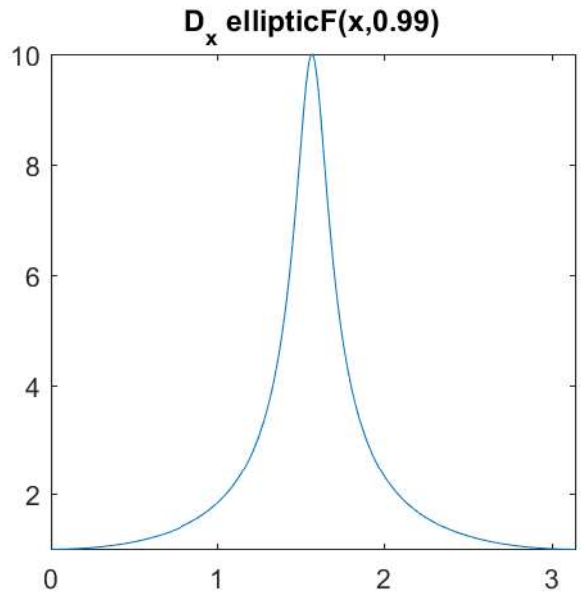
```
        end
        sumj = sumj + sumi;
    end
    I_1a(mj) = sumj;
end
Fa = [];
for k=1:length(m);
    Fa(k) = ellipticE(pi,0.5)-ellipticE(0,0.5) - I_1a(k);
end
qFa = [];
for l=1:(length(m)-1);
    qFa(l) = abs(Fa(l))/abs(Fa(l+1));
end
wa = mean(log2(qFa))
Fa


% Unterpunkt b
I_3b = [];
n=3;
for mj=1:length(m)
    sumj = 0;
    for j=1:m(mj);
        sumi = 0;
        for i=0:n;
            Gj = transf(f,m(mj),j,a,b);
            sumi = sumi + A(n+1,i+1)*Gj(X(n+1,i+1));
        end
        sumj = sumj + sumi;
    end
    I_3b(mj) = sumj;
end
Fb = [];
for k=1:length(m);
    Fb(k) = ellipticE(pi,0.5)-ellipticE(0,0.5) - I_3b(k);
end
qFb = [];
for l=1:(length(m)-1);
    qFb(l) = abs(Fb(l))/abs(Fb(l+1));
end
wb = mean(log2(qFb))
Fb


% Unterpunkt c
I_2c = [];
n=2;
for mj=1:length(m)
    sumj = 0;
    for j=1:m(mj);
        sumi = 0;
        for i=0:n;
            Gj = transf(g,m(mj),j,a,b);
            sumi = sumi + A(n+1,i+1)*Gj(X(n+1,i+1));
        end
        sumj = sumj + sumi;
    end
    I_2c(mj) = sumj;
end
Fc = [];
```

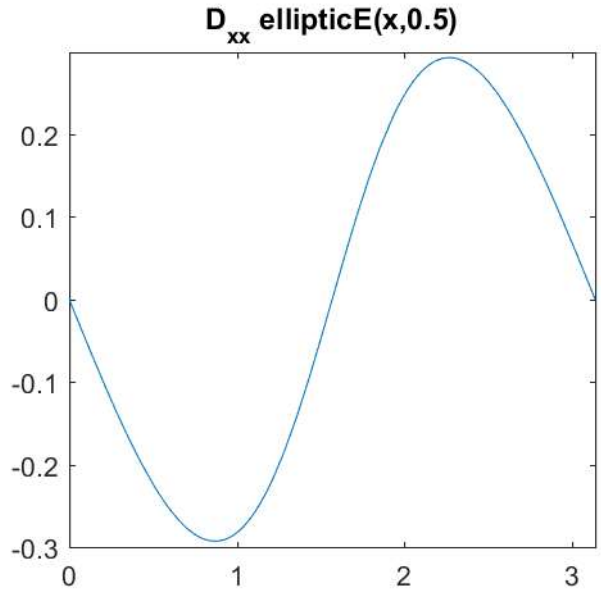
```
for k=1:length(m);
    Fc(k) = ellipticF(pi,0.99)-ellipticF(0,0.99) - I_2c(k);
end
qFc = [];
for l=1:(length(m)-1);
    qFc(l) = abs(Fc(l))/abs(Fc(l+1));
end
wc = mean(log2(qFc))
Fc


% Unterpunkt d
I_4d = [];
n=4;
for mj=1:length(m)
    sumj = 0;
    for j=1:m(mj);
        sumi = 0;
        for i=0:n;
            Gj = transf(g,m(mj),j,a,b);
            sumi = sumi + A(n+1,i+1)*Gj(X(n+1,i+1));
        end
        sumj = sumj + sumi;
    end
    I_4d(mj) = sumj;
end
Fd = [];
for k=1:length(m);
    Fd(k) = ellipticF(pi,0.99)-ellipticF(0,0.99) - I_4d(k);
end
qFd = [];
for l=1:(length(m)-1);
    qFd(l) = abs(Fd(l))/abs(Fd(l+1));
end
wd = mean(log2(qFd))
Fd


% die Ordnung w des Approximationsfehlers F ist in den Unterpunkten (c) und
% (d) für die Funktion g = D_x ellipticF(x,0.99) geringer als für die
% Funktion f = D_x ellipticE(x,0.5) in den Unterpunkten (a) und (b), da g
% im Vergleich zu f (siehe Grafik - insbesondere deren Ableitung) spitzer
% ist

% A und B sind richtig
```

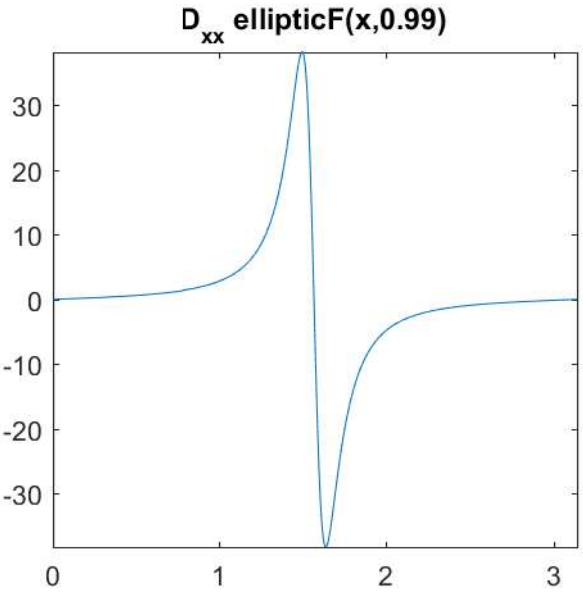
```
% Funktion zur Erstellung von G_j(x) bei zusammengesetzter Gauss-Quadratur
% Koordinatentransformation
function Gj = transf(f,m,j,a,b)
h = (b-a)/m;
tj = a+(j-1)*h;
Tj = @(x) tj + (x+1)*h/2;
Gj = @(x) f(Tj(x))*h/2;
end
```

```
wa =

    5.7577


Fa =

   -0.1264    0.0046    0.0000


wb =

    7.2413


Fb =

   -0.0042    0.0000    0.0000


wc =

    3.2476


Fc =

   -8.4308    0.2147   -0.0935


wd =

    4.5036


Fd =

   -4.2982   -0.0954   -0.0084

>>
```


$D_x$ ellipticE(x,0.5)
1
0.95
0.9
0.85
0.8
0.75
0
1
2
3


$D_x$ ellipticF(x,0.99)
10
8
6
4
2
0
1
2
3


$D_{xx}$ ellipticE(x,0.5)
0.2
0.1
0
-0.1
-0.2
-0.3
0
1
2
3


$D_{xx}$ ellipticF(x,0.99)
30
20
10
0
-10
-20
-30
0
1
2
3

## Beispiel 8:

Gram Schmidt Verfahren:

```
function U = GS(V)
  [n,m]=size(V);
  U=[];
  for i = 1:m
    u = V(:,i);
    U=[U,u];
    for j = 1:(i-1)
      U(:,i)=U(:,i) - dot(U(:,j),V(:,i))*U(:,j);
    endfor
    U(:,i) = U(:,i)/norm(U(:,i),2);
  endfor

endfunction
```

Stabilisiertes Gram Schmidt Verfahren:

```
function W = SGS(V)
  [n,m]=size(V);
  W=V;
  for i = 1:m
    W(:,i)=W(:,i)/norm(W(:,i),2);
    for j = (i+1):m
      W(:,j) = W(:,j) - dot(W(:,i),W(:,j))*W(:,i);
    endfor
  endfor
endfunction
```

Legendre Polynome:

```
function p = Legendre(x,k)
  switch k
    case 0
      p = ones(size(x));
    case 1
      p = x;
    otherwise
      %index shift because recursion is defined for k+1
      k = k - 1;
      p = (2*k+1).* x .* Legendre(x,k)-k.* Legendre(x,k-1);
      p = p ./ (k+1);
  endswitch
endfunction
```

Main:

```
clc; clear;
%Definiere n und m, a) 100,10; b) 100,20; c) 1000,10; d) 1000,20
n=100;  m=10;

%Definiere x und V
x=[];
for i=1:n
  x=[x; -1+ 2*(i-1)/(n-1)];
endfor
V=[ones(n,1)];
for k=2:m
  v=x.^(k-1);
  V=[V,v];
endfor

%Definiere P über eine rekursive Funktion
L = [];
for k = 0:1:m
  L = [L, Legendre(x,k)];
endfor
P = [];
for k = 1:m
  P = [P, L(:,k)/norm(L(:,k), 2)];
endfor

U=GS(V);
W=SGS(V);

Wert_1_ab=norm(U'*U-eye(m), Inf)
Wert_2_ab=norm(W'*W-eye(m), Inf)
Wert_3_ab=norm(P'*P-eye(m), Inf)
Wert_1_cd=norm(U-P, Inf)
Wert_2_cd=norm(W-P, Inf)
```

Ergebnis a): (korrekt)

```
Wert_1_ab =    3.9646e-12
Wert_2_ab =    5.5686e-14
Wert_3_ab =  0.51801
```

Ergebnis b): (korrekt)

```
Wert_1_ab =  0.12081
Wert_2_ab =  0.00000000019254
Wert_3_ab =  2.6949
```

Ergebnis c): (erster Wert korrekt, zweiter Wert falsch)

```
Wert_1_cd =  0.014474
Wert_2_cd =  0.014474
```

Ergebnis d): (erster Wert korrekt, zweiter Wert falsch)

```
Wert_1_cd =  0.38907
Wert_2_cd =  0.18196
```