Dr. R. Käppeli
M. Sprecher

Musterlösung 13

1. **a)** Es gilt

$$
\begin{aligned}
c_1 &= -2x_k y_k^2 && (1)\\
c_2 &= -(y_k + 2c_1 x_k) y_k && (2)\\
c_3 &= -(4c_1 y_k + 2x_k(c_1^2 + 2c_2 y_k))/3 && (3)\\
c_4 &= -\left(c_1^2/2 + c_2 y_k + x_k(c_1 c_2 + c_3 y_k)\right) && (4)
\end{aligned}
$$

b) Durch Separation der Variabeln

$$y(x) = \frac{1}{x^2}.$$

c) Für den lokalen Fehler gilt

$$|y(x_1) - y_1| = \mathcal{O}(h^5).$$

Die Konsistenzordnung betraegt also 4. Für den globalen Fehler gilt

$$|y(x_n) - y_n| = \mathcal{O}(h^4).$$

```
%aufgabe1c.m

%initial values
x0=1;
y0=1;
T=2;

m=10;
h=2.^(-1:-1:-m);
%lokaler Fehler
lok_err=zeros(1,m);
%y(h)
y1ex=(1+h).^(-2);

%globaler Fehler
```

```
glob_err=zeros(1,m);
yex=1/4;


for i=1:m
    y=taylormethod(x0,y0,h(i),T);
    lok_err(i)=abs(y1ex(i)-y(2));
    glob_err(i)=abs(yex-y(end));
end

loglog(h,lok_err,h,glob_err);
legend('Lokaler Fehler','Globaler Fehler');
xlabel('h')
ylabel('err');
print('-dpdf','taylor.pdf')

polyfit(log(h),log(lok_err),1)
polyfit(log(h(3:end)),log(glob_err(3:end)),1)

function [y] = taylormethod(x0,y0,h,T)
%Taylor method
%(x0,y0): Initial values
%h: stepwidth
%T: Intervalend

n=floor((T-x0)/h);
x=x0:h:x0+n*h;
y=zeros(1,n+1);
y(1)=y0;

for i=1:n
    y(i+1)=taylorstep(x(i),y(i),h);
end

end


function [yn] = taylorstep(xk,yk,h)
%One step of Taylor expansion method

c=taylorkoff(xk,yk);
yn=yk+h*(c(1)+h*(c(2)+h*(c(3)+h*c(4))));
```

Siehe nächstes Blatt!

```
end


function [c] = taylorkoff(xk,yk)
%TAYLORKOFF computes the coefficients of the
%taylorexpansion of y(x) at x=xk for the
%solution of the differential equation
%y'=-2xy^2

c=zeros(1,4);
c(1)=-2*xk*yk^2;
c(2)=-(yk+2*c(1)*xk)*yk;
c(3)=-(4*c(1)*yk+2*xk*(c(1)^2+2*c(2)*yk))/3;
c(4)=-(c(1)^2/2+c(2)*yk+xk*(c(1)*c(2)+c(3)*yk));


end
```

2. **a)** Wir haben eine lineare Differentialgleichung zweiter Ordnung.

b) Um die DGL in ein System erster Ordnung umzuschreiben, definieren wir $z_1(t) := y(t)$, $z_2(t) := y'(t)$ und erhalten daraus folgendes System erster Ordnung:

$$\begin{pmatrix} z_1'(t) \\ z_2'(t) \end{pmatrix} = \begin{pmatrix} z_2(t) \\ -\frac{D}{m} z_1(t) \end{pmatrix}$$

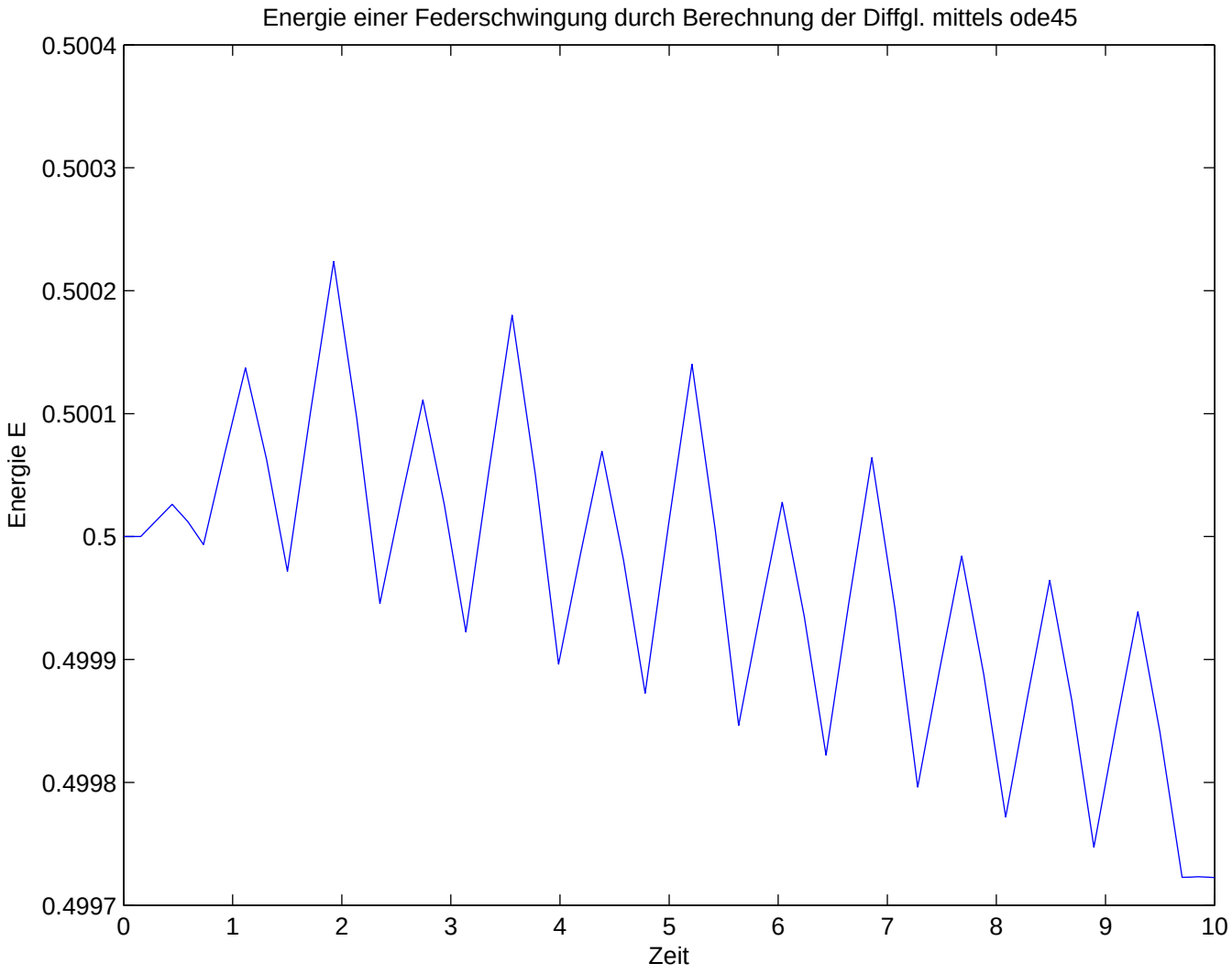
mit Anfangsbedingung

$$\begin{pmatrix} z_1(t_0) \\ z_2(t_0) \end{pmatrix} = \begin{pmatrix} x_0 \\ v_0 \end{pmatrix}.$$

c)

```
f=@(t,y) [y(2);-y(1)];
[T,Y]=ode45(f,[0,10],[1;0]);
E=(Y(:,1).^2+Y(:,2).^2)/2;
plot(T,E)
title('Energie einer Federschwingung durch Berechnung der Diffgl. mit
xlabel('Zeit');
ylabel('Energie E')
print('-dpdf','energie.pdf');
```

Bitte wenden!



Siehe nächstes Blatt!

Die exakte Lösung ist $x_0(t) = \cos(t)$ und die Energie bleibt konstant. Bei der Approximation die mittels *ode45* berechnet wurde bleibt die Energie nicht konstant (Abweichung von ca. 10^{-4}), was physikalisch nicht korrekt ist.

d)
```
%aufgabe2d.m
m=1;
D=1;
T=1;
f=@(x,y) [y(2);-D/m*y(1)];
y0=[1;0];
t0=0;
%Colors
C=['r','g','b'];
h=[1 0.1 0.001];
for i=1:3
    [t,y]=implicit_trapez(f,t0,y0,h(i),T);
    Eerr=sum(y.^2)/2-0.5;
    plot(t,Eerr,'Color',C(i));
    hold on;
end
legend(['h=' num2str(h(1))],['h=' num2str(h(2))],['h=' num2str(h(3))]
title(['Energie mit impliziter Trapezregel']);
xlabel('Zeit');
ylabel('Fehler Energie')
print('-dpdf','energie_trap.pdf');

function [t,y] = implicit_trapez(f,t0,y0,h,T)
%implicit trapezoidal rule
%(x0,y0): Initial values
%h: stepwidth
%T: Intervalend

n=floor((T-t0)/h);
t=t0:h:t0+n*h;
%Dimension
d=max(size(y0));
y=zeros(d,n+1);
y(:,1)=y0;

for i=1:n
    y(:,i+1)=implicit_trapez_step(f,t(i),y(:,i),h);
end
```

Bitte wenden!

```
end

function [yk1] = implizit_trapez_step(f,xk,yk,h)
%computes one step of the implicit midpoint rule


phi=@(yk1) yk+(f(xk,yk)+f(xk+h,yk1))/2*h-yk1;
yk1=fsolve(phi,yk);

end
```

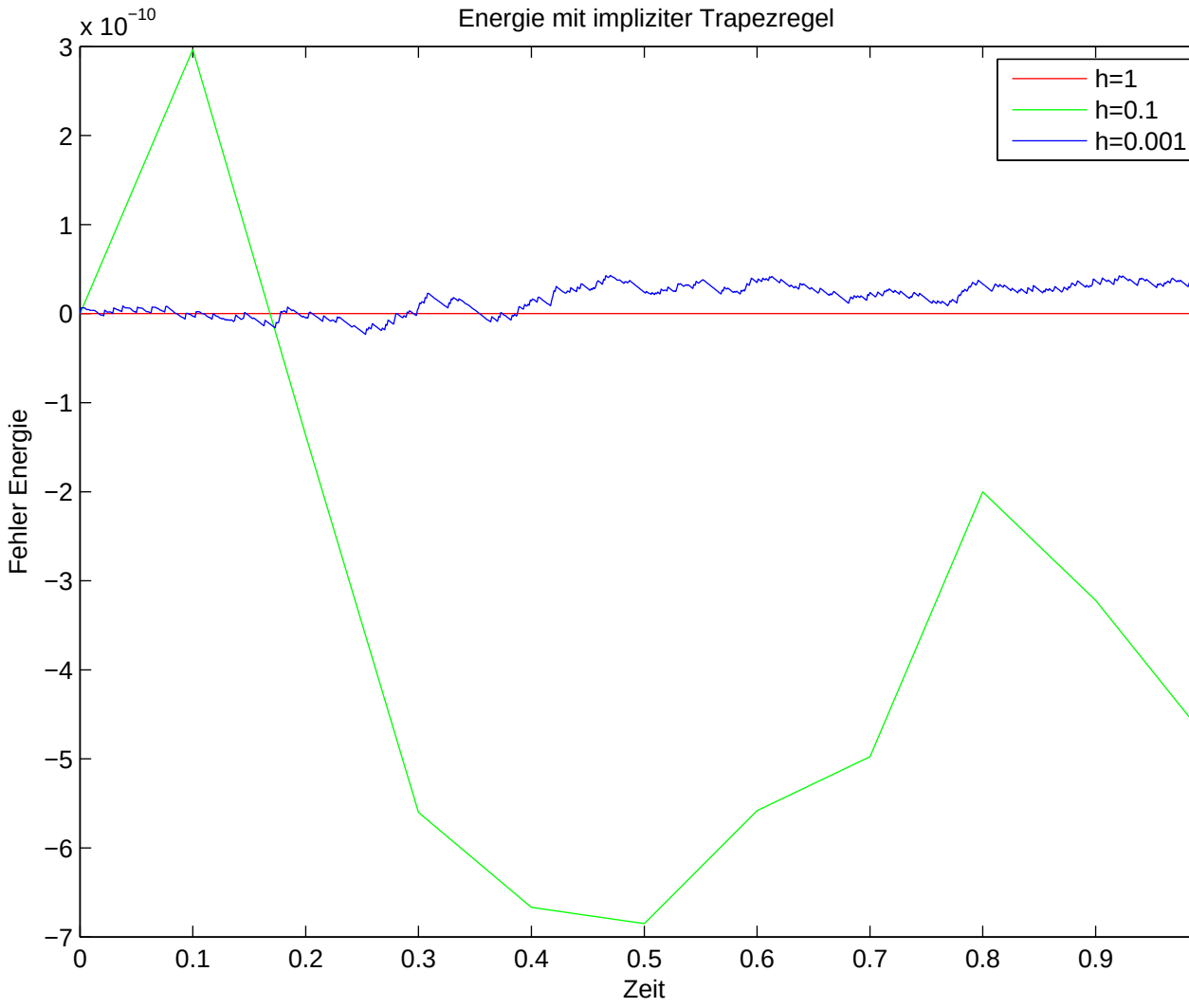
Die Energie bleibt bei der impliziten Trapezregel bis auf einen Fehler von ca. 10^{-10} erhalten. Der Fehler von ca. 10^{-10} ist auf Rundungsfehler zuruckzuführen. Man kann nämlich zeigen, dass die implizite Trapezregel quadratische Invarianten erhält.

3. Die exakte Lösung ist

$$\begin{array}{rcrcrcr} y_1(t) & = & 15e^{-0.5t} & - & 12e^{-45t} & + & e^{-75t} \\ y_2(t) & = & & + & 12e^{-45t} & + & e^{-75t} \\ y_3(t) & = & & + & 4e^{-45t} & - & 3e^{-75t} \end{array}$$

```
%Matrix der Differentialgleichung
A=[-.5 32.6 35.7;0 -48 9;0 9 -72];
y0=[4;13;1];



%AWP exakt loesen
%eigenvalues and eigenvectors of A
[V,D]=eig(A);
%startvektor als Linearkombination der Eigenvektoren
c=V\y0;
for i=1:3
    V(:,i)=c(i)*V(:,i);
end
V
D=diag(D)
%Loesung plotten
m=1000;
```

Siehe nächstes Blatt!



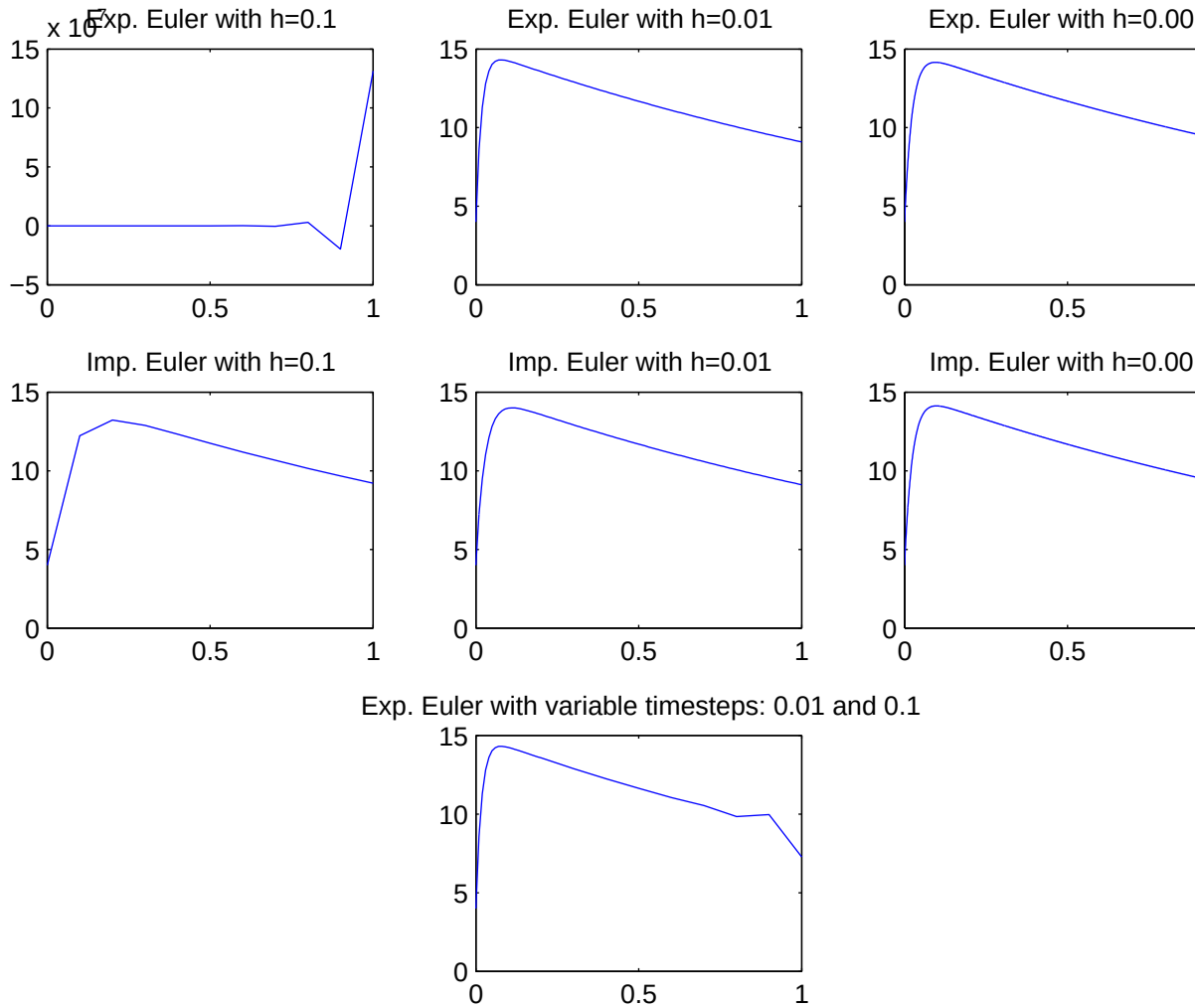
Bitte wenden!

```
t=linspace(0,1,m);
yex=zeros(3,m);
for i=1:m
    yex(:,i)=V*exp(t(i)*D);
end
plot(t,yex(1,:),t,yex(2,:),t,yex(3,:))
legend('y1','y2','y3')
print('-dpdf','stiff_ex.pdf')


%Explizit Euler
h=10.^(-(1:3));
for i=1:3
    hi=h(i);
    n=floor(1/hi);
    t=0:h(i):n*hi;
    y=zeros(3,n+1);
    y(:,1)=y0;
    IhA=eye(3)+hi*A;
    for j=1:n
        y(:,j+1)=IhA*y(:,j);
    end
    subplot(3,3,i);
    plot(t,y(1,:));
    title(['Exp. Euler with h=' num2str(hi)]);
end


%Implizit Euler
for i=1:3
    hi=h(i);
    n=floor(1/hi);
    t=0:h(i):n*hi;
    y=zeros(3,n+1);
    y(:,1)=y0;
    IhA=inv(eye(3)-hi*A);
    for j=1:n
        y(:,j+1)=IhA*y(:,j);
    end
    subplot(3,3,3+i);
    plot(t,y(1,:));
    title(['Imp. Euler with h=' num2str(hi)]);
```

Siehe nächstes Blatt!

```
end

%Explizit Euler mit variabler Schrittweite
    n=28;
    %kleine Schrittweite
    hs=0.01;
    %grosse Schrittweite
    hl=0.1;
    t=[0:hs:0.2 0.3:hl:1];
    y=zeros(3,n+1);
    y(:,1)=y0;
    IhA=eye(3)+hs*A;
    for j=1:20
        y(:,j+1)=IhA*y(:,j);
    end
    IhA=eye(3)+hl*A;
    for j=21:n
        y(:,j+1)=IhA*y(:,j);
    end
    subplot(3,3,8);
    plot(t,y(1,:));
    title(['Exp. Euler with variable timesteps: ' num2str(hs) ' and ' nu
    print('-dpdf','stiff.pdf')
```

Die Lösung des expliziten Eulerverfahrens mit $h = 10^{-1}$ liefert keine qualitativ richtige Lösung.


Exp. Euler with h=0.1
x 10^5
Exp. Euler with h=0.01
Exp. Euler with h=0.00
Imp. Euler with h=0.1
Imp. Euler with h=0.01
Imp. Euler with h=0.00
Exp. Euler with variable timesteps: 0.01 and 0.1