Institute of Scientific Computing
Technical University Braunschweig
Dr. Bojana Rosić
Dr. Thilo Moshagen

Summer Term 2015

Advanced Methods for ODEs and DAEs: 2.: Impl. Runge-Kutta Methods

The main script

```
% Ass. 3 solutions:
%Exercise 1: Draw A stability region (in Matlab) of the Runge Kutta method below
%Exercise 2:
%Consider two masses m1 = 2kg and m2 = 2kg on the distance r
%attracted by gravitational force gm1 m2 /r2 , where g = 2m3 kg \u22121 s\u22122 is the gravitat
%The mass movement can be described by a second Newton law.
%T.Moshagen 2014


clear all;
ex1=false %true
if ex1
gridReal=linspace(-3, 0.5, 100);
gridIm=linspace(-3, 3, 100);
gridReal=linspace(-10, 0.5, 100);
gridIm=linspace(-13, 13, 100);
[X,Y]=meshgrid(gridReal, 1i*gridIm);
Z=X+Y;

%%Define 4-stage Runge-Kutta: bounded stability region (not A-stable)
A_RK4=[0    0   0   0
    1/2 0   0   0
    0 1/2   0   0
   0    0   1   0];
b_RK4=[1/6  1/3 1/3 1/6];
Vals=stabPol(reshape(Z,1,[]),A_RK4,b_RK4);
absvals = abs(reshape(Vals, size(Z)));
contour(gridReal, gridIm, absvals, [1, 1.1])
title( 'Stab. region of 4stage-Runge Kutta');
grid on;


else
%solve movement of 2 attracting masses (Kepler)
%Method: Gauss-Legendre 2
A_GaussLegendre=[1/4      1/4-sqrt(3)/6
        1/4+sqrt(3)/6  1/4];
b_GaussLegendre=[1/2  1/2];
c_GaussLegendre=sum(A_GaussLegendre,2)

%problem
funcPtr=@twoBodyRhs;
jacFuncPtr=@jacTwoBodyRhs;
```

```
%Initial condition as demanded:
u0=[ -1   0   0 -1 1 0 0 1 ];
%try those:
%u0=[ -1   0   0 -0.3 1 0 0 1 ];
%fails, because masses fall into singularity of forces (like black hole):
%u0=[ -1   0   0   0 1 0 0 0 ];
t(1)=0;
solGaussLegendre(1,:)=u0;
t_end = 5.5
h=0.01
maxiter =6;tol= 1e-8;

for step = 1:1:t_end/h
        solGaussLegendre(step+1,:)=generalImplicitMethod(funcPtr,jacFuncPtr, t(step),solGaussLeg
    t(step+1)=t(step)+h;
end
    plot(t,solGaussLegendre(:,1),'b',t, solGaussLegendre(:,2),'b.', t,solGaussLegendre(:,4),'r',
    hold on;
    legend( 'x_1', 'y_1' , 'x_1', 'y_1');
figure;
plot(solGaussLegendre(:,1),solGaussLegendre(:,2),'r',solGaussLegendre(:,5),solGaussLegendre(:,6)
title('Phase Diagram')
end
```

implements exercise 1 and two. Set switch *ex1=false/true* accordingly.

1 Exercise 1

The function

```
function val = stabPol(z,A,b)
if size(b,1)>size(b,2)
    b=b';
end
s=size(b,2)

k=zeros(s,1 );

if 0 %Test for expl methods
x=1;
h=1;
for j=1:max(size(z))
    lamb=z(j);
    for i=1:s
         k(i)=lamb*(x+h*(A(i,:)*k));
    end
    val(j) =x+ h*(b)*k;
end
else
   for j=1:max(size(z))
%    for k=1:size(z,2)
        val(j)  =1 + z(j)*b* inv(eye(s)-z(j)*A)*ones(s,1);
%    end
   end
```

```
end

end
```

implements an "idle run" of an explicit Runge Kutta method as well as calculation of

$$\mathbf{R}(z) = (1 + \mathbf{b}^T z(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{e})$$

as given in Lecture 5, slide 5. First part of the main script calls it and plots the stability region of the method in question.

2 Exercise 2

The demanded Two-Masses System has to be transformed into a explicit first-Order ODE by setting

$$\dot{\mathbf{x_1}} = \mathbf{v_1} \tag{1}$$

$$\dot{\mathbf{x_2}} := \dot{\mathbf{v_1}} = \frac{m_2}{r^3}(\mathbf{x_2} - \mathbf{x_1}) \tag{2}$$

$$\dot{\mathbf{x_1}} = \mathbf{v_1} \tag{3}$$

$$\dot{\mathbf{x_2}} := \dot{\mathbf{v_1}} = -\frac{m_1}{r^3}(\mathbf{x_2} - \mathbf{x_1}) \tag{4}$$

As RHS function it is implemented in

```
function udot =twoBodyRhs(t,u)

%Sol vector is
%x1
%v1
%x2
%v2
%vi,xi being vector valued and denoting mi's velocity and pos, resp..
g=2;
m1=2;
m2=2; %kg

deltaX=[u(5) - u(1)
    u(6) - u(2)];
dist = norm(deltaX)
fac=g/dist^3;

%u=[v,a]
udot = fac*[0
 0
 m2*deltaX(1)
 m2*deltaX(2)
 0
  0
 -m1*deltaX(1)
 -m1*deltaX(2) ]+ [u(3); u(4); 0;  0; u(7); u(8); 0; 0];
%m*f                       x'=v
end
```

The Jacobian can be determined analytically:

```
function Du =jacTwoBodyRhs(t,u)
%Sol vector is
%x1
%v1
%x2
%v2
%vi,xi being vector valued and denoting mi's velocity and pos, resp..
%T.Moshagen 2014
g=2;
m1=2;
m2=2; %kg
deltaX=[u(5) - u(1)
    u(6) - u(2)];
dist = norm(deltaX);
deltaXN= deltaX/dist;
fac=-g/3*dist^3;
%For gravitational part, lower half = -upper half could be used.
Du=[0    0   0   0   0   0   0   0
0 0  0   0   0   0   0   0
 fac*m2*(deltaXN(1)^2+1)     fac*m2*deltaXN(1)*deltaXN(2)  0 0   -fac*m2*(deltaXN(1)^2 +1)
-fac*m2*deltaXN(1)*deltaXN(2) 0 0
 fac*m2*deltaXN(1)*deltaXN(2)    fac*m2*(deltaXN(2)^2+1)     0 0     -fac*m2*deltaXN(1)*deltaXN(
-fac*m2*(deltaXN(2)^2 +1) 0 0
0 0  0   0   0   0   0   0
0 0  0   0   0   0   0   0
 -fac*m1*(deltaXN(1)^2+1)    -fac*m1*deltaXN(1)*deltaXN(2)  0 0 fac*m1*(deltaXN(1)^2 +1)
fac*m1*deltaXN(1)*deltaXN(2) 0 0
 -fac*m1*deltaXN(1)*deltaXN(2)   -fac*m1*(deltaXN(2)^2+1)       0 0 fac*m1*deltaXN(1)*deltaXN(2)
fac*m1*(deltaXN(2)^2 +1) 0 0]+...
[0   0   1   0   0   0   0   0
0    0   0   1   0   0   0   0
0    0   0   0   0   0   0   0
0    0   0   0   0   0   0   0
0    0   0   0   0   0   1   0
0    0   0   0   0   0   0   1
0    0   0   0   0   0   0   0
0    0   0   0   0   0   0   0];
```

One has to implement general implicit Runge-Kutta method for real-valued problems in Matlab, and pcg should be used as linear solver:

```
function u=generalImplicitMethod(my_fun,my_der,t,u,h,A,b,c,maxiter,tol)

% FUNCTION
%
%     general_implicit_method(my_fun,my_der,t,u,h,c,A,b,maxiter,tol)
%
%  solves the ODE equation given by my_fun (the function for ODE)
%  and my_der (the function derivative over u)
%
%  Input:
%
%       my_fun - function (function handle)
%       my_der - derivative of a function (function handle)
%       t - current time step
```

```
%       u - the solution from previous step m (vector of dimension n)
%       h - time step
%       A,b,c - Bucher table of the method
%       maxiter - maximal number of iterations
%       tol - the tolerance for Newton method
%
%  Output:
%
%       u - the solution in the next step m+1
%
% IMPORTANT:
%      input functions have two arguments: time, parameter.
%
%
%  B. Rosic
%  wire@tu-bs.de
%  2011

    if nargin<1
        disp('No arguments specified')
    elseif nargin<8
        disp('Not enough input arguments')
    elseif nargin<9
        maxiter=100;
        tol=1e-10;
    elseif nargin<10
        tol=1e-10;
    end

    % check dimensions

    if size(u,2)>size(u,1)
        u=u';
    end

    if size(b,2)<size(b,1)
        b=b';
    end

     if size(c,2)<size(c,1)
        c=c';
    end

    % Find stage

    s = length(b);

    % Size of the system of equations

    n=length(u);

    % final size of parameters

    w=n*s;

    % Term h(A\kron Id) in Eq. (39)
```

```
    Id=eye(n);

    AId=h*kron(A,Id);

    % Term e\kron u_m in Eq. (39)

    e=ones(s,1);

    eu_m=kron(e,u);

    u0=reshape(eu_m,[],1);

    % initial value for v

    v=zeros(n*s,1);
    v=repmat(u,s,1);
    u_old=u;
    % vector t+c*h

    ch=ones(size(c))*t+c*h;

    convg=0;

%     Newton iteration

    niter=0;

    while (niter<maxiter & ~convg)

         v_old=v;
         niter=niter+1;

        % Term G(t_m,v) in Eq. (39)
        for i=1:s
          G(((i-1)*n+1):i*n,1)=feval(my_fun,ch(i),v(((i-1)*n+1):i*n));

         % Jacobian
          dGdV(((i-1)*n+1):i*n,((i-1)*n+1):i*n)=feval(my_der,ch(i),v(((i-1)*n+1):i*n));%%Verify:
        end

        J=eye(w)-AId*dGdV;

        % compute residual
        R=v-eu_m-AId*G;

        % solve system of equations
        %ok:v=v-J\R;
%demanded in assignment 3:
        v=v-pcg(J,R,tol/10);


     %   v= gmres(J,-R,10,tol);

        if norm(v-v_old)<tol%Criteria checks past progress, not state
```

```
            fprintf('Newton method converged in iteration %d with the norm %1.5e \n',niter,norm(
          %           disp(norm(v-v_old))
           convg=1;

            for i = 1:s
            % iteration converged: compute k and return
            %idx = (l-1)*n+1:l*n;
%ok ld, not my style:                k(((i-1)*n+1):i*n,l)=feval(my_fun,ch(i),v(((i-1)*n+1):i*n))
%u_old+h*k_old(((i-1)*n+1):i*n));
            k(1:n,i)=feval(my_fun,ch(i),v(((i-1)*n+1):i*n));  %each col 1 stage
            %??k(:,l) = feval(my_fun, t + c(l)*h, u0(idx));
           end
        else if (niter==maxiter-1)
            fprintf('Newton method did not converge in iteration %d, the norm %1.5e \n',niter,no
             end
         k=zeros(n,s);  %to enable continuing

        end

    end
% compute u_m+1
    u = u + h*k*b';
    %other opportunity: u=u+(b'*AId)*(v-eu_m)
```