Linear Multistep solver for differential equations + Newton's method & order of convergence analysis

Yan D. R. Machado

Optoelectronics Laboratory - Physics Department, Pontifical Catholic University of Rio de Janeiro

A q-step method (*q* $\geq$ *1*) is one which, $\forall$ *n* $\geq$ *q-1*, u_{n+1} depends on u_{n+1-q}, but not on the values u_k with *k* < *n+1-q*.

$$u_{n+1} = \sum_{j=0}^{p} a_j u_{n-j} + h \sum_{j=0}^{p} b_j f_{n-j} + h b_{-1} f_{n+1}, \; n = p, p+1, \ldots$$

[1]

Those are p+1-step methods, p ≥ 1. For p = 0, we recover one-step methods.

The coefficients a_j, b_j are real and fully identify the method; they are such that for a_p $\neq$ 0 or b_j $\neq$ 0. If b_{-1} $\neq$ 0 the method is implicit (and we need to use Newton's Method, already defined as a function to approximate solutions), otherwise its explicit.

```
% Parameters

f = @(t,y) ; %function you want to approximate
t0 = ; %initial time value
T = ; %final time value
dt = ; %time interval (I usually go for 0.01)
y_real = @(t); %analytic solution (use to plot error values)
dfy  = @(t,y) '
y0 = ;
a = ;
b = ;
b_1 = ;
u_1 = multistep_general(a,b,b_1,t0,T,dt,f,y0,10000,1e-4,dfy)
conv_order = give_convergence_ms(f,dfy,y_real,y0,t0,T,a,b,b_1,1e-6,5000)


% Plot

figure
plot(t0:dt:T,y_real(t0:dt:T),'Linewidth',2, 'color', 'r')
hold on
```

```
plot(t0:dt:T,u_1,'--', 'Linewidth', 2, 'color','b')
title('Adam Moulton and Exact Solution')
legend('Solução Exata','Aproximação Numérica')
grid on;
hold off


% Error Plot

figure
plot(t0:dt:T,(y_real(t0:dt:T)-u_1),'Linewidth',2, 'color', 'g')
title('Erro')
grid on;


% Phase Portrait

for j=1:length(t0:dt:T)
    y1_dot_aprox(j)=u_1(1,j);
    y2_dot_aprox(j)=u_1(2,j);
end
plot(y1_dot_aprox,y2_dot_aprox,'r')
```

```
% General function for multistep methods

function [u] = multistep_general(a,b,b_1,t0,T,dt,f,y0,max_it,tol,dfy)
    q = length(a);
    time = t0:dt:T;
    u(:,1) = y0;
    p=q-1;
    for i = 1:q-1
        u(:,i+1) = heun(time,i,dt,f,u(:,i));
    end

    for i = (q+1):length(time)
        sizeu = length(u);
        soma = 0;
        for j=1:p+1
            soma = soma + a(j)*u(:,i-j) + dt * b(j)*f(time(i-j),u(:,i-1));
        end
        if b_1==0
            u(:,i) = soma;
        else
            Func = @(soma,dt,b_1,f,time,i,x) x-soma-dt*b_1*(f(time(i)+dt,x));
            u(:,i)=Newton_(Func,u(:,i-1),f,dfy,time,i,b_1,dt,soma,max_it,tol);
        end
    end

    function [uj_next] = heun(time,j,dt,f,uj)
        uj_next=uj+dt/2*(f(time(j),uj)+f(time(j+1),uj+dt*f(time(j),uj)));
    end
```

```
    function [x] = Newton_(Func,x,f,dfy,time,i,b_1,dt,soma,max_it,tol)
        count = 1; erro=10;
        while erro>tol && count < max_it
            val=(Func(soma,dt,b_1,f,time,i,x))';
            J = eye(size(x))-dt*b_1*dfy(time(i)+dt,x);
            df = (-J/val)';
            x=x+df';
            erro = norm(df,Inf);
            count = count+1;
            if count == max_it
                disp('Max iterations achieved: check parameters')
            end
        end
    end
end
```

```
% Convergence Order

function [ord] = give_convergence_ms(f,dfy,y_real,y0,t0,T,a,b,b_1,tol,itmax)
    for j = 1:5
        dt = (0.1)/(2^(j-1));
        time = t0:dt:T;
        u_real = y_real(time);
        u_method = multistep_general(a,b,b_1,t0,T,dt,f,y0,5000,1e-4,dfy);
        for i=1:length(time)-1
            error_method(i) = norm(u_real(i)-u_method(i),'inf');
        end
        error(j) = max(error_method);
        if j>1
            ord(j-1)=log2(error(j-1)/error(j));
        end
    end
end
```

Below some examples of linear multistep methods and their parameter values. [1]

I - Adam-Bashforth (AB, Explicit Adam):

If p = 0 we recover Forward Euler.

2-step: a_j **= [1 0],** b_j **= [3/2 -1/2],** b_{-1} **= 0**

$$u_{n+1} = u_n + \frac{h}{2}\left[3f_n - f_{n-1}\right]$$

3-step: a_j **= [1 0 0],** b_j **= [23/12 -16/12 5/12],** b_{-1}**= 0**

$$u_{n+1} = u_n + \frac{h}{12}\left[23f_n - 16f_{n-1} + 5f_{n-2}\right]$$

4-step: a_j **= [1 0 0 0],** b_j **= [55/24 -59/24 37/24 -9/24],** b_{-1}**= 0**

$$u_{n+1} = u_n + \frac{h}{24}\left(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}\right)$$

II - Adam-Moulton (AM, Implicit Adam):

If p = -1 we recover Backward Euler.

2-step: a_j **= [1 0],** b_j **= [8/12 -1/12],** b_{-1}**= 5/12**

$$u_{n+1} = u_n + \frac{h}{12}\left[5f_{n+1} + 8f_n - f_{n-1}\right]$$

3-step: a_j **= [1 0 0],** b_j **= [19/24 -5/24 1/24],** b_{-1}**= 9/24**

$$u_{n+1} = u_n + \frac{h}{24}\left(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}\right)$$

4-step: a_j **= [1 0 0 0],** b_j **= [646/720 -264/720 106/720 -19/720],** b_{-1}**= 251/720**

$$u_{n+1} = u_n + \frac{h}{720}\left(251f_{n+1} + 646f_n - 264f_{n-1} + 106f_{n-2} - 19f_{n-3}\right)$$

III - Backward Differentiation Methods (BDF, Implicit)

$$u_{n+1} = \sum_{j=0}^{p} a_j u_{n-j} + h b_{-1} f_{n+1}$$

Coefficients of zero-stable BDF methods for p = 0, 1, ..., 5.

p	a_0	a_1	a_2	a_3	a_4	a_5	b_{-1}
0	1	0	0	0	0	0	1
1	$\frac{4}{3}$	$-\frac{1}{3}$	0	0	0	0	$\frac{2}{3}$
2	$\frac{18}{11}$	$-\frac{9}{11}$	$\frac{2}{11}$	0	0	0	$\frac{6}{11}$
3	$\frac{48}{25}$	$-\frac{36}{25}$	$\frac{16}{25}$	$-\frac{3}{25}$	0	0	$\frac{12}{25}$
4	$\frac{300}{137}$	$-\frac{300}{137}$	$\frac{200}{137}$	$-\frac{75}{137}$	$\frac{12}{137}$	0	$\frac{60}{137}$
5	$\frac{360}{147}$	$-\frac{450}{147}$	$\frac{400}{147}$	$-\frac{225}{147}$	$\frac{72}{147}$	$-\frac{10}{147}$	$\frac{60}{137}$

References:

[1] Quarteroni, A., Sacco, R., & Saleri, F. (2010). *Numerical mathematics* (Vol. 37). Springer Science & Business Media.