Numerical Analysis – Lab 12

1 Taylor's Methods (optional)

Implementation of Taylor's Time Stepping. Suppose for the IVP

$$y'(t) = f(t, y(t)) \quad a \le t \le b, \qquad y(a) = y_a$$

the total derivatives of $f(t, y(t)$ are available[1]. Then Taylor's method is implemented as follows.

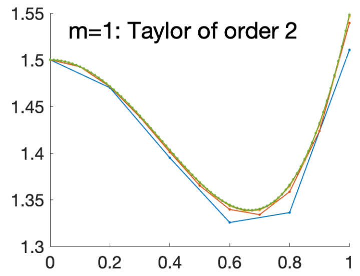
```
function [t, y] = taylorstep(f,df,m,a,b,n,ya)
% Taylor explicit time-stepping for
% y'=f(t,y), y(a) = ya
% df(f,t,y(t),j) is the total derivative about t for j times
% m = max j, (y(i+1)-y(i))/dt is approximated within error of O(h^(m+1))
t = zeros(n+1,1); t(1) = a;
h = (b-a)/n; d = numel(ya);
y = zeros(n+1,d); y(1,:) = ya;
for i = 1:n
    s = zeros(1,d);
    for j=m:-1:1 % add smaller term first
        s = s + h^(j+1)*df(f,t(i),y(i,:),j)/factorial(j+1);
    end
    y(i+1,:) = y(i,:) + s + h*f(t(i),y(i,:));
    t(i+1) = t(i) + h;
end
```

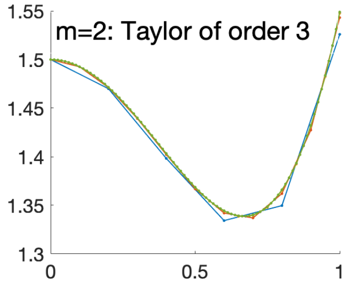
To use it, we need to define `f`, `df` as required. In the following example, we use the 2nd order Taylor's method ($m = 1$) with number of time steps $n = 5, 10, 20, 40, 80$.

```
f = @(t,y) -t*y+3*t.^3;
ya = 1.5;

% Taylor
m = 1; nrange = 5*2.^(0:4); hold on;
for n = nrange
    [t, y] = taylorstep(f,@df,m,0,1,n,ya);
    plot(t,y,'.-','LineWidth',1);
end
set(gca,'FontSize',18); hold off;

function re = df(f,t,y,j)
if j==1
    re = -y+3*3*t^2-t*f(t,y);
elseif j==2
    re = -2*f(t,y)+3*3*t-t*(-y+3*3*t^2-t*f(t,y));
end
end
```





[1]The calculation of the total derivatives may be done by hand for low order but quickly becomes intractable by hand as the order increases. In general, one would need the *automatic differentiation* technique. Some packages using the technique can be found on the web e.g. `https://cody.unizar.es/code/tides/` and `https://bluescarni.github.io/heyoka/`

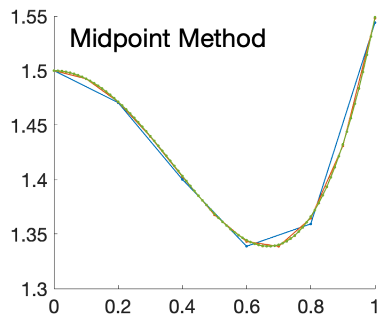
2 Runge-Kutta Methods

Midpoint Method. We have the following implementation.

```
function [t, y] = midstep(f,a,b,n,ya)
% Mid-point i.e. Runge's explicit time-stepping for
% y'=f(t,y), y(a) = ya
t = zeros(n+1,1); t(1) = a;
h = (b-a)/n; d = numel(ya);
y = zeros(n+1,d); y(1,:) = ya;
for i = 1:n
    y1 = y(i,:) + h/2*f(t(i),y(i,:));
    y(i+1,:) = y(i,:) + h*f(t(i)+h/2,y1);
    t(i+1) = t(i) + h;
end
```

We solve again the previous problem with a range of time steps. The convergence is observed.

```
% Midpoint
nrange = 5*2.^(0:4); hold on;
for n = nrange
    [t, y] = midstep(f,0,1,n,ya);
    plot(t,y,'.-','LineWidth',1);
end
set(gca,'FontSize',18); hold off;
```

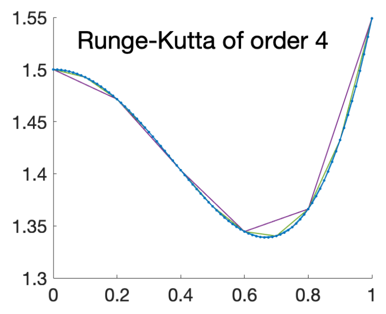


Classical Runge-Kutta Method. The 4th order Runge-Kutta method is implemented as follows.

```
function [t, y] = rkstep(f,a,b,n,ya)
% Classical Runge-Kutta explicit time-stepping for
% y'=f(t,y), y(a) = ya
t = zeros(n+1,1); t(1) = a;
h = (b-a)/n; d = numel(ya);
y = zeros(n+1,d); y(1,:) = ya;
for i = 1:n
    k1 = h*f(t(i),y(i,:));
    k2 = h*f(t(i)+h/2,y(i,:)+k1/2);
    k3 = h*f(t(i)+h/2,y(i,:)+k2/2);
    k4 = h*f(t(i)+h,y(i,:)+k3);
    y(i+1,:) = y(i,:) + (k1+2*k2+2*k3+k4)/6;
    t(i+1) = t(i) + h;
end
```

To test it on the previous problem, we need only to change `midstep` to `rkstep`.

```
% Runge-Kutta
nrange = 5*2.^(0:4); hold on;
for n = nrange
    [t, y] = rkstep(f,0,1,n,ya);
    plot(t,y,'.-','LineWidth',1);
end
set(gca,'FontSize',18); hold off;
```


Runge-Kutta of order 4