Numerical Methods for ODEs: Some One Step Methods

Introduction:

In this document we will show how to use ***Mathematica*** to study the different one step numerical methods for the solution of Ordinary Differential Equations (ODEs). Specifically we will show how to obtain an approximate solution to an Initial Value Problem (IVP) of the form $y' = f(t, y)$, $y(t_0) = y_0$, obtaining values of the solution by using equally spaced points over the interval $[t_0, t_N]$, i.e. $t_n = t_0 + n\,h$, to $n = 0, \ldots, N$. The distance $h = \frac{t_N - t_0}{N}$ is called the **step lenght**. y_n will be the approximate value of $y(t_n)$ and we will write $f_n = f(t_n, y_n)$.

1. Using NDSolve.

It is well known that ODEs cannot always be integrated exactly. If ***Mathematica*** cannot solve an ODE exactly, the **DSolve** command Output is the same as that obtained using the Input instruction:

```
ecu = DSolve[y'[t] == 1 + y[t]^2 - t^3, y[t], t]
```

```
DSolve[y′[t] == 1 - t^3 + y[t]^2, y[t], t]
```

When this happens, the ODE can be solved approximately, whenever the problem to solve has just one solution, i.e. it is an IVP. The appropriate instruction to give an approximate solution to an ODE is **NDSolve** using the syntax:

```
    NDSolve[{equation, initials conditions}, function, {variable , interval}]
```

In general, **NDSolve[{equation,initials conditions},x[t],{t,a,b}]** gives an approximation of the solution *x(t)* for the IVP in the interval $[a, b]$. For example, if a solution for the IVP $y' = 1 + y^2 - t^3$, $y(0) = 0$ in the interval [0, 4], is required, the appropriate instruction is:

```
solution = NDSolve[{y'[t] == 1 + y[t]^2 - t^3, y[0] == 0}, y[t], {t, 0, 4}]
```

```
{{y[t] → InterpolatingFunction[{{0., 4.}}, <>][t]}}
```

As it can be seen, the Output of the **NDSolve** command is an assignment to a function defined in terms of an interpolating function. Therefore, the expression must be coverted to a value table in order to plot the approximate solution obtained. The following method is used:

```
yapprox[t_] = y[t] /. solution[[1]]
```

```
InterpolatingFunction[{{0., 4.}}, <>][t]
```

Now the function value at any point of the interval considered can be calculated:

```
yapprox[1]
```

```
1.16197
```

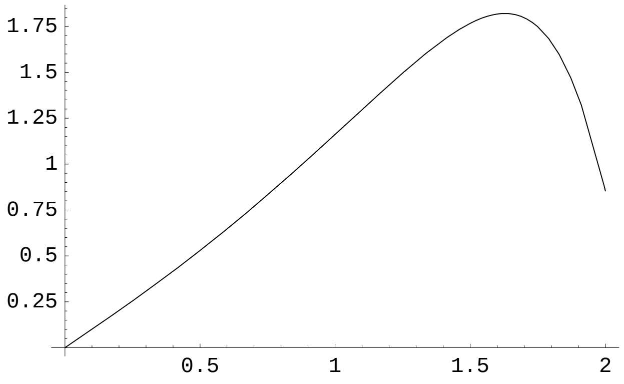
A table of equispaced value in the interval [0, 1] can also be calculated using the step $h = 0.1$:

```
Table[{t, yapprox[t]}, {t, 0, 1, 0.1}] // TableForm // Chop
```

```
0       0
0.1     0.10031
0.2     0.202305
0.3     0.307248
0.4     0.416024
0.5     0.529206
0.6     0.647092
0.7     0.769722
0.8     0.896859
0.9     1.02794
1.      1.16197
```

or the solution can be represented graphically using the **Plot** command. For example, its graphical representation can be calculated in the interval [0, 2]:

```
Plot[yapprox[t], {t, 0, 2}]
```



REMARKS:

1.- The solution obtained is only defined in the interval considered. If the solution has to be evaluated at a point outside the defined interval, *Mathematica* informs the user that extrapolation techniques are being implemented:

```
yapprox[6]
```

```
InterpolatingFunction::dmval :
 Input value {6} lies outside the range of data in the interpolating
   function. Extrapolation will be used. More…
```

```
-14.5568
```

2.- If the program Help is consulted, the different options i.e. the numerical method used for the approximation, etc. that the **NDSolve** function has at its command are specified. These include Adams, Runge-Kutta, Euler, etc.

Example 1.

Obtain the approximate values of the solution of the IVP , y'=y(1-sin t), y(0)=1, in the interval [0,1], with equis-paced point at distance 0.05 apart.

Solution

The command **NDSolve** is used directly to solve it.

```
sol11 = NDSolve[{y'[t] == y[t] (1 - Sin[t]), y[0] == 1}, y[t], {t, 0, 1}]
```

```
{{y[t] → InterpolatingFunction[{{0., 1.}}, <>][t]}}
```

The solution function is defined and the required value table is calculated:

```
yap11[t_] = y[t] /. sol11[[1]]
```

```
InterpolatingFunction[{{0., 1.}}, <>][t]
```

```
Table[{t, yap11[t]}, {t, 0, 1, 0.05}] // TableForm
```

```
0       1.
0.05    1.04996
0.1     1.09966
0.15    1.14886
0.2     1.1973
0.25    1.24472
0.3     1.2909
0.35    1.33559
0.4     1.37859
0.45    1.4197
0.5     1.45875
0.55    1.4956
0.6     1.5301
0.65    1.56218
0.7     1.59176
0.75    1.61881
0.8     1.6433
0.85    1.66526
0.9     1.68474
0.95    1.70179
1.      1.71653
```

2. Euler's Method.

Euler's method is the easiest simple step method. With the usual notation, the Euler's algorithm to approximate the solution of the IVP $y' = f(t, y)$ with $y(t_0) = y_0$ at the interval $[t_0 = a, b]$, is the expression:

$$y_{n+1} = y_n + h\, f(t_n, y_n) = y_n + h\, f_n \qquad n = 0, \ldots, m-1$$

where $h = \dfrac{b-a}{m}$ is the step length and $t_n = t_{n-1} + h = a + n\,h$.

In accordance with the general theory of Numerical Methods, the characteristic equation is *p(x) = x - 1* so it is easy to show that it this is a convergent numerical method, because it is stable and consistent. The last expression can be coded with *Mathematica* as follows:

```
euler [f_, h_, ini_, a_, b_] := Module [ {y, t, ytable, c}, c = (b - a) / h;
 y[0] = ini; t[n_] := a + n h; y[n_] := y[n] = y[n - 1] + h f[t[n - 1], y[n - 1]];
 ytable = Table[y[i], {i, 0, c}];
Table[{t[i], ytable[[i + 1]]}, {i, 0, c}] // TableForm]
```

where **f** is the function associated with the ODE, **h** is the step length, **ini** is the value of the initial condition,and **a** and **b** are the boundary points. The Output provides the table of values of the approximate solution to the ODE.

Next, the previous procedure trial is done to solve the IVP $y' = -t\,y + \frac{4\,t}{y}$, *y(0)=1* approximately at the interval [0,1], with step length 0.1. The corresponding function is defined:

```
f[t_, y_] = -t y + 4 t/y;
```

and the previous algorithm is applied knowing that, in this case and according with the conditions of the wording, $h = 0.1$, $ini = 1$, $a = 0$, $b = 1$.

```
euler[f, 0.1, 1, 0, 1]
```

```
0       1
0.1     1
0.2     1.03
0.3     1.08707
0.4     1.16485
0.5     1.25561
0.6     1.35211
0.7     1.44849
0.8     1.5404
0.9     1.6249
1.      1.70021
```

In order to evaluate the accuracy of the approximation, the exact solution of the IVP is calculated and its values compared with the solution obtained using Euler's method. The command **DSolve** is applied

```
exactsol = DSolve[{y'[t] == -t y[t] + 4 t/y[t], y[0] == 1}, y[t], t]
```

$$\left\{\left\{y[t] \to e^{-\frac{t^2}{2}} \sqrt{-3 + 4 e^{t^2}}\right\}\right\}$$

The solution function obtained is defined and its corresponding value table is calculated

```
yex[t_] = exactsol[[1, 1, 2]]
```

$$e^{-\frac{t^2}{2}} \sqrt{-3 + 4 e^{t^2}}$$

```
Table[{t, yex[t]}, {t, 0, 1, 0.1}] // TableForm
```

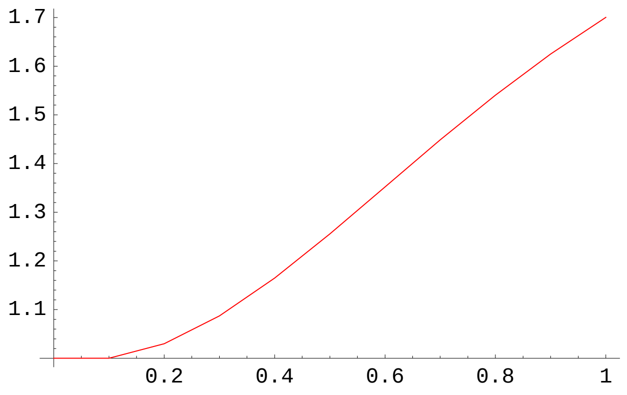
```
0       1
0.1     1.01482
0.2     1.05718
0.3     1.1217
0.4     1.20149
0.5     1.28981
0.6     1.38093
0.7     1.47042
0.8     1.55503
0.9     1.63261
1.      1.70187
```

The comparison can be shown graphically. A procedure of ***Mathematica*** is programmed to present the graphical representation, in red color, of the solution obtained with Euler's method. This new procedure **grafeuler** has the same input as the procedure programmed before with the same meaning for the parameters used

```
grafeuler[f_, h_, ini_, a_, b_] :=
 Module[{y, t, ytable, c}, c = (b - a)/h; y[0] = ini; t[n_] := a + n h;
  y[n_] := y[n] = y[n - 1] + h f[t[n - 1], y[n - 1]]; ytable = Table[y[i], {i, 0, c}];
  ListPlot[Table[{t[i], ytable[[i + 1]]}, {i, 0, c}], Joined → True,
   PlotStyle → {RGBColor[1, 0, 0]}, PlotRange → All]]
```

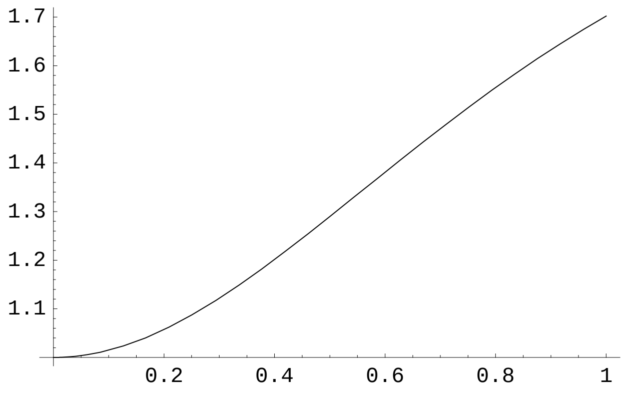
The approximate solution and the exact solution are represented together to allow the graphical comparison

```
grafeuler[f, 0.1, 1, 0, 1]
```

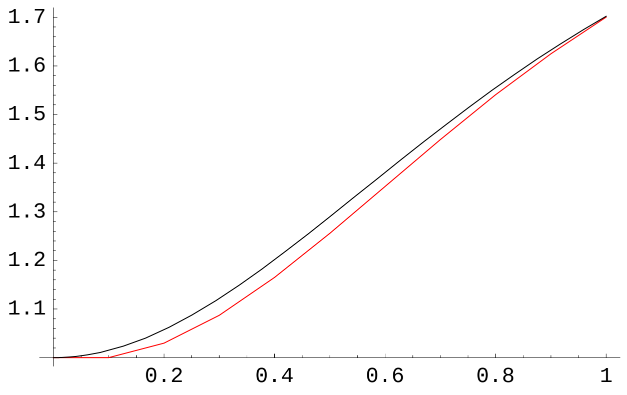


```
aproxi = %;
```

```
exact = Plot[yex[t], {t, 0, 1}]
```



```
Show[aproxi, exact]
```



Example 2.

Using Euler's method, produce an approximate solution to the IVP $y' = 3\,(y + t)$, $y(0) = 1$, on the interval $[0,1]$, with equispaced points at the distance 0.1 apart. Represent graphically the obtained solution with the exact solution calculated using the command **DSolve**. *Comment on the bad behaviour of the approximation and obtain an improved approximation by reducing the step length.*

Solution

The rough solution and the exact solution are calculated using **DSolve**:

```
f[t_, y_] := 3 (y + t)
```

```
euler[f, 0.1, 1, 0, 1]
```

```
0       1
0.1     1.3
0.2     1.72
0.3     2.296
0.4     3.0748
0.5     4.11724
0.6     5.50241
0.7     7.33314
0.8     9.74308
0.9     12.906
1.      17.0478
```

```
solexacta22 = DSolve[{y'[t] == 3 (y[t] + t), y[0] == 1}, y[t], t]
```

$$\left\{\left\{y[t] \to \frac{1}{3}\left(-1 + 4\,e^{3\,t} - 3\,t\right)\right\}\right\}$$

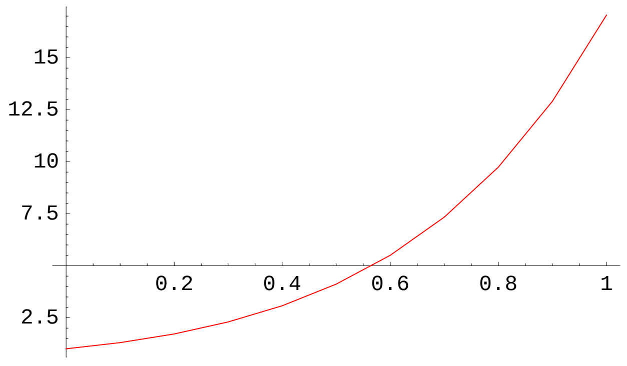
The obtained solution function is defined

```
yex22[t_] = solexacta22[[1, 1, 2]]
```

$$\frac{1}{3}\left(-1 + 4\,e^{3\,t} - 3\,t\right)$$

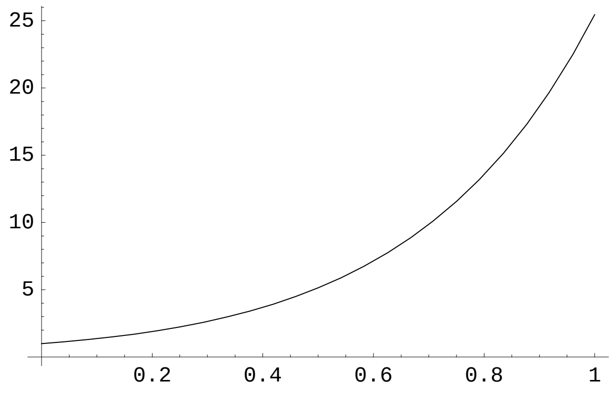
The graphical representation of both solutions is calculated as before

```
grafeuler[f, 0.1, 1, 0, 1]
```

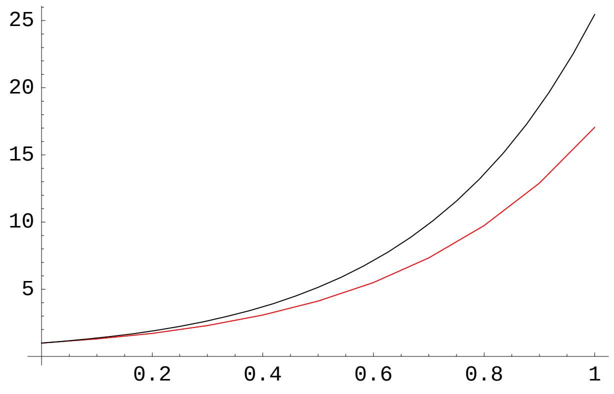


```
aproximada22 = %;
```

```
exacta22 = Plot[yex22[t], {t, 0, 1}]
```

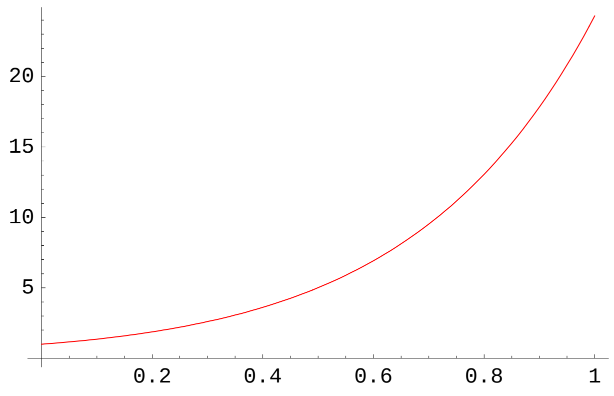


```
Show[aproximada22, exacta22]
```



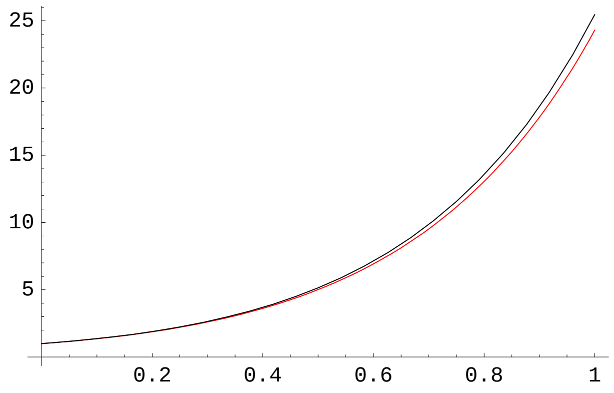
As you can see from the graphs, the approximation obtained by the Euler method is not good. This is because the exact solution rises very rapidly due to the exponential term. A better approximation can be obtained using a smaller step length

```
grafeuler[f, 0.01, 1, 0, 1]
```



```
aproximada22bis = %;
```

```
Show[aproximada22bis, exacta22]
```


25
20
15
10
5
0.2
0.4
0.6
0.8
1