11.2.4 Checking the Accuracy of Numeric Solutions of Differential Equations.

A method to check the accuracy of numeric solutions of differential equations is to run **NDSolve[]** at least twice with different values assigned to the options **AccuracyGoal, PrecisionGoal** and **Working-Precision** (s. 11.2.3). Another method for such a check applicable to differential equations describing a conservative system is to compare the initial and final values of total energy. The latter check, however, is a rather weak one. A third check is to run the
system in a second run in the revers direction and to check whether the final data of the latter run agree with the initial date of the first run. All this is shown for the nonlinear system discussed in more detail in 11.2.6 .

Equations of Motion

```
Clear[t, r, x, y, vx, vy, sys, en, tmax, force]
r[t_] = {x[t], y[t]}; v[t_] = {vx[t], vy[t]};
force = {-x[t] - 2 x[t] y[t], -y[t] + y[t]^2 - x[t]^2} ;
sys = D[Join[r[t], v[t]], t] == Join[v[t], force ] // Thread
```

$$\{x'[t] == vx[t],\ y'[t] == vy[t],\ vx'[t] == -x[t] - 2\,x[t]\,y[t],\ vy'[t] == -x[t]^2 - y[t] + y[t]^2\}$$

```
en = (vx[t]^2 + vy[t]^2) / 2 + (x[t]^2 + y[t]^2) / 2 +
     x[t]^2 y[t] - y[t]^3 / 3;

ad = {x[0], y[0], vx[0], vy[0]};
```

Run with default precision for a short time

```
anf = {0, 0.1, 0.15, 0.55}; tmax = 15.;

sol = NDSolve[ Join[sys, Thread[ad == anf]], {x, y, vx, vy}, {t, 0, tmax}] // Flatten ;

eni = ( en /. sol ) /. t -> 0
```

```
0.167167
```

```
enf = ( en /. sol ) /. t -> tmax
```

```
0.167167
```

```
(enf - eni) / eni
```

$$-8.96089 \times 10^{-7}$$

This check is useful inasmuch as a large difference in the values of the initial and the final energies indicates a serious inaccuracy. But energy is a stationary quantity; so it is insensitive to errors in the data used to evaluate it. Therefore this check represents more a necessary than a sufficient condition.

A more sensitive check is to evaluate the coordinates and velocities at the final time point, to run the system backwards in time till the initial time and to compare the final coordinates and velocities of this run to the initial coordinates and velocities used in the first run. The time reversal just described requires to use the final coordinates and the negative values of the final velocities of the first run as initial data for the second run.

Return with default precision for the same short time

```
{x[tmax], y[tmax], vx[tmax], vy[tmax]} /. sol // Flatten
```

```
{0.11961, -0.309081, 0.0691334, -0.457029}
```

```
anfr = %{1, 1, -1, -1}
```

```
{0.11961, -0.309081, -0.0691334, 0.457029}
```

```
sor =
  NDSolve[ Join[sys, Thread[ad == anfr]], {x, y, vx, vy}, {t, 0, tmax}] // Flatten ;
```

```
{x[tmax], y[tmax], vx[tmax], vy[tmax]} /. sor // Flatten
```

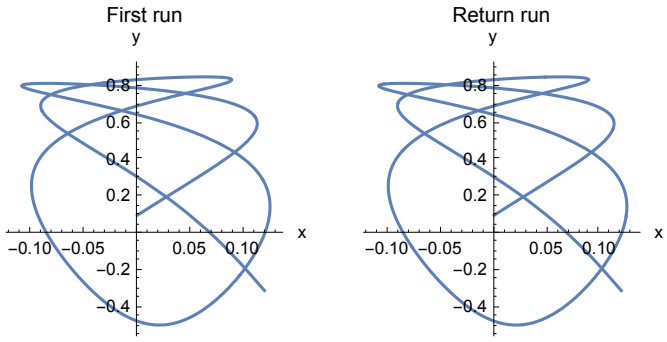
$$\{-1.42224 \times 10^{-6},\ 0.100004,\ -0.149998,\ -0.549999\}$$

```
anf
```

$$\{0,\ 0.1,\ 0.15,\ 0.55\}$$

Now the final data of the reversed run can be compared to the inital data (anf). Agreement is quite good. The different sign of the velocity is correct as in the second run the trajectory returns to the starting ing point of the first run. The trajectories in coordinate space are indistinguishable.

```
p1 = ParametricPlot[Evaluate[r[t] /.sol], {t, 0, tmax},
    AxesLabel → {"x", "y"}, PlotLabel → "First run", AspectRatio → 1];
p2 = ParametricPlot[Evaluate[r[t] /.sor], {t, 0, tmax},
    AxesLabel → {"x", "y"}, PlotLabel → "Return run", AspectRatio → 1];
Show[GraphicsRow[{p1, p2}]]
```



The solutions of this highly nonlinear system near the stability limit require appropriate options for runs over long time intervalls; without these the solutions become inaccurate soon. But these options increase the CP time, the running time and data storage space very much. The accuracy is checked by running backwards from the end point to compare these end data with the initial data.

Run with default precision for a long time

```
tmax = 150.;
```

```
{time, sol0} = NDSolve[ Join[sys, Thread[ad == anf]], {x, y, vx, vy}, {t, 0, tmax},
    MaxSteps → 5000] // Flatten // Timing
```

$$\{0.005099,\ \{x \to \text{InterpolatingFunction}[\ \langle\text{Domain: } \{\{0.,\ 150.\}\}\ \text{Output: scalar}\rangle\ ],$$

$$y \to \text{InterpolatingFunction}[\ \langle\text{Domain: } \{\{0.,\ 150.\}\}\ \text{Output: scalar}\rangle\ ],$$

$$vx \to \text{InterpolatingFunction}[\ \langle\text{Domain: } \{\{0.,\ 150.\}\}\ \text{Output: scalar}\rangle\ ],$$

$$vy \to \text{InterpolatingFunction}[\ \langle\text{Domain: } \{\{0.,\ 150.\}\}\ \text{Output: scalar}\rangle\ ]\}\}$$

```
g0x = Plot[ Evaluate[ x[t] /. sol0], {t, 0, tmax},
          AxesLabel -> {"t", "x[t]"},
     PlotStyle -> {{AbsoluteThickness[1], Dashing[{.01}]}},
     PlotPoints -> 500];
g0y = Plot[ Evaluate[ y[t] /. sol0 ], {t, 0, tmax},
          AxesLabel -> {"t", "y[t]"},
     PlotStyle -> {{AbsoluteThickness[1], Dashing[{.01}]}},
     PlotPoints -> 500];
```

Return with default precision for a long time

```
{ x[tmax], y[tmax], vx[tmax], vy[tmax] } /. sol0
```

```
{-0.47746, -0.140588, -0.143229, -0.358235}
```

```
anfr = %{1, 1, -1, -1}
```

```
{-0.47746, -0.140588, 0.143229, 0.358235}
```

```
{time,sor0} = NDSolve[Join[sys,Thread[ad==anfr]], {x,y,vx,vy}, {t,0,tmax},
                      MaxSteps->5000]//Flatten//Timing
```

```
{0.005205, {x → InterpolatingFunction[ 

],
  y → InterpolatingFunction[ 

],
  vx → InterpolatingFunction[ 

],
  vy → InterpolatingFunction[ 

]}}
```

```
{ x[tmax], y[tmax], vx[tmax], vy[tmax] } /. sor0
```

```
{0.00441213, 0.229267, -0.131878, -0.521904}
```

```
anf
```

```
{0, 0.1, 0.15, 0.55}
```

```
eni = (en /. sol0)  /.  t -> 0
```

```
0.167167
```

```
enr = (en /. sor0)  /.  t -> tmax
```

```
0.167167
```

It is obvious that the end values of the return run differ strongly from the intial data (anf) of the first run, although the various values of total energy do not differ very much. To get reliable values for the solutions over the full time intervall appropriate options must be prescribed in **`NDSolve[]`**.

Run with high precision for a long time

```
{time,sol1} = NDSolve[Join[sys,Thread[ad==anf]], {x,y,vx,vy}, {t,0,tmax},
              AccuracyGoal  -> 20,  WorkingPrecision -> 25,
              PrecisionGoal -> 20, MaxSteps -> 400000]//Flatten//Timing
```

NDSolve::precw : The precision of the differential equation

$$\left(\left\{x'[t] == vx[t],\ y'[t] == vy[t],\ vx'[t] == -x[t] - 2\,x[t]\,y[t],\ vy'[t] == -x[t]^2 - y[t] + y[t]^2,\ x[0] == 0,\ y[0] == 0.1,\ vx[0] == 0.15,\ vy[0] == 0.55\right\},\ \{\},\ \{\},\ \{\},\ \{\}\right)$$ is less than WorkingPrecision (25.`). ≫

```
{1.416479, {x → InterpolatingFunction[ <Domain: {{0, 150.00000000000000000000}}
                                       Output: scalar> ],
  y → InterpolatingFunction[ <Domain: {{0, 150.00000000000000000000}}
                              Output: scalar> ],
  vx → InterpolatingFunction[ <Domain: {{0, 150.00000000000000000000}}
                               Output: scalar> ],
  vy → InterpolatingFunction[ <Domain: {{0, 150.00000000000000000000}}
                               Output: scalar> ]}}
```

The above warning tells us that the statements describing the differential equations and the initial conditions are not given with the working precision (25). The final tests show that the precision obtained suffices for our purposes.

```
g1x = Plot[ Evaluate[ x[t]/.sol1], {t,0,tmax},
           AxesLabel -> {"t","x[t]"},
           PlotStyle -> AbsoluteThickness[1],
           PlotPoints -> 500];
g1y = Plot[ Evaluate[ y[t]/.sol1 ], {t,0,tmax},
           AxesLabel -> {"t","y[t]"},
           PlotStyle -> AbsoluteThickness[1],
           PlotPoints -> 500];
```

Return with high precision for a long time

```
SetPrecision[{ x[tmax], y[tmax], vx[tmax], vy[tmax] } /. sol1,30]
```

```
{-0.49916375710205523885676370810 0, -0.44350829738654745648673838331 9,
 -0.04184630361147952115619119695 11, 0.22264895142268473215985125079 9}
```

```
anfr = %{1, 1, -1, -1}
```

```
{-0.49916375710205523885676370810 0, -0.44350829738654745648673838331 9,
 0.04184630361147952115619119695 11, -0.22264895142268473215985125079 9}
```

```
{time,sor1} = NDSolve[Join[sys,Thread[ad==anfr]], {x,y,vx,vy}, {t,0,tmax},
              AccuracyGoal  -> 20, WorkingPrecision -> 25,
              PrecisionGoal -> 20, MaxSteps->400000]//Flatten//Timing
```

```
{1.283930, {x → InterpolatingFunction[ Domain: {{0, 150.00000000000000000000}}
                                       Output: scalar ],
  y → InterpolatingFunction[ Domain: {{0, 150.00000000000000000000}}
                             Output: scalar ],
  vx → InterpolatingFunction[ Domain: {{0, 150.00000000000000000000}}
                              Output: scalar ],
  vy → InterpolatingFunction[ Domain: {{0, 150.00000000000000000000}}
                              Output: scalar ]}}
```

```
{ x[tmax], y[tmax], vx[tmax], vy[tmax] } /. sor1
```

$$\{2.26943 \times 10^{-6}, 0.0999848, -0.150001, -0.550002\}$$

```
anf
```

```
{0, 0.1, 0.15, 0.55}
```

```
eni = ( en /. sol1 )  /. t -> 0
```

```
0.16716666666666690758506
```

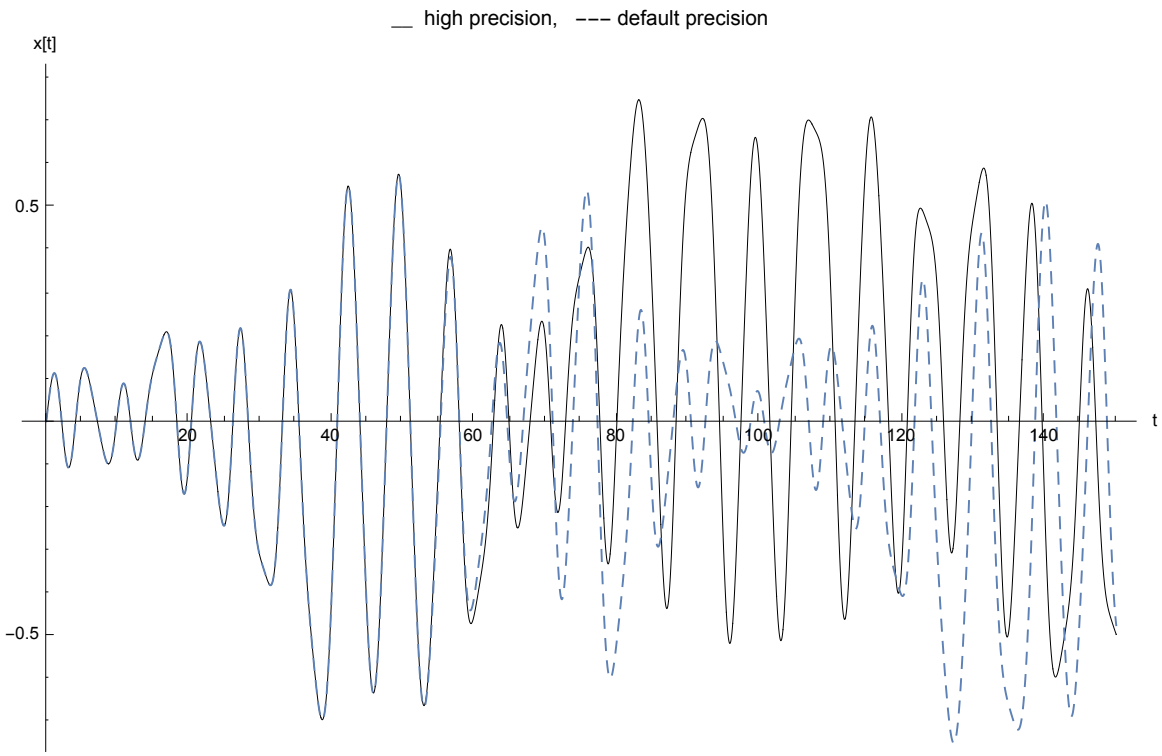
```
enr = ( en /. sor1 )  /. t -> tmax
```

```
0.167167
```

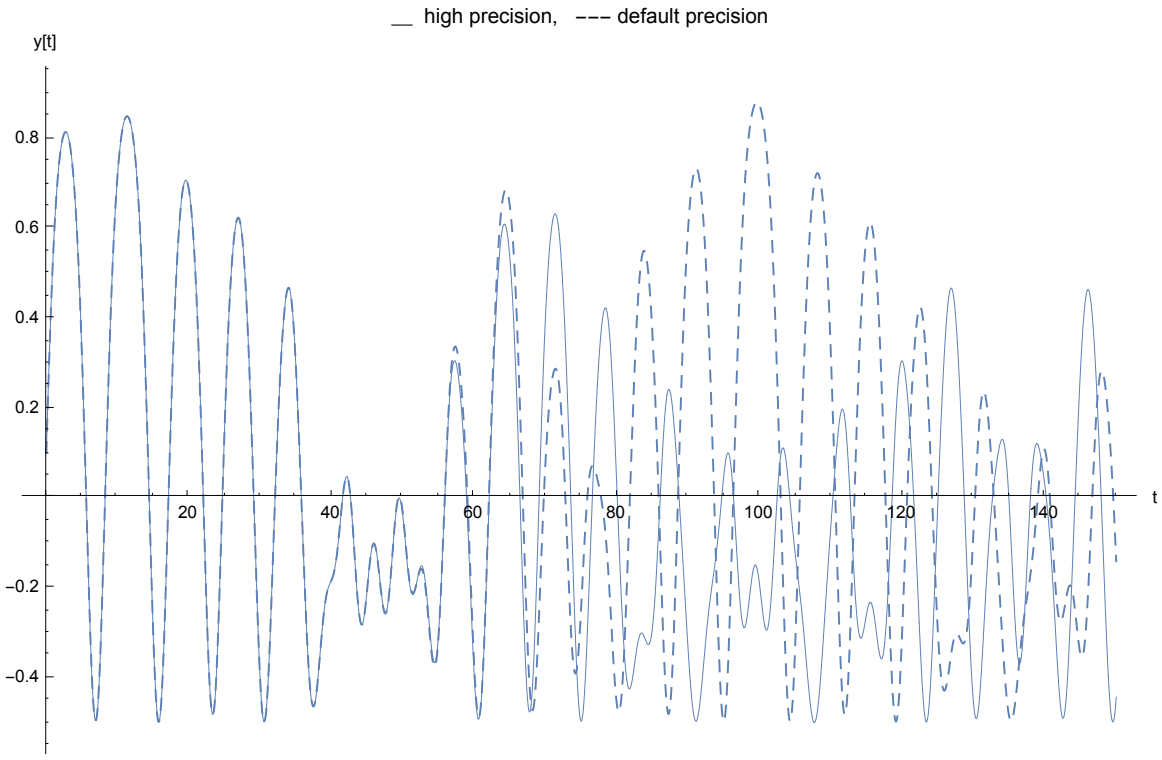
Now the end values of the return run agree well with the initial data of the first run. So one can be sure that the solutions are accurate over the whole intervall of the integration. Below the coordinates of the points are compared as obtained with the stronger options (continuous line) and with unsufficient options (dashed line).

```
g1x = Plot[ Evaluate[ x[t]/.sol1] , {t,0,tmax},
           AxesLabel -> {"t","x[t]"},
           PlotStyle -> {AbsoluteThickness[0.5],Black},
           PlotPoints -> 500];
g1y = Plot[ Evaluate[ y[t]/.sol1 ], {t,0,tmax},
           AxesLabel -> {"t","y[t]"},
           PlotStyle -> AbsoluteThickness[0.5],
           PlotPoints -> 500];
```

```
Show[g1x,g0x, ImageSize -> 600,
PlotLabel -> "__  high precision,   --- default precision"]
```




```
Show[g0y, g1y, ImageSize → 600,
 PlotLabel → "__  high precision,   --- default precision"]
```



Diagrams in x,y-space are less suitable to show the difference between accurate and inaccurate solutions. From the outputs above it is seen that the options for higher precision entail about a factor 100 in CP time. The CP times, however, do not reflect the real running times, wich last still longer. So other programming languages compare favourably for such tasks.

In fact, a few years ago, using integration routines *) written in C++ and MATLAB graphics software, N. Ivanovic needed about 2 to 3 sec computer time to prepare figures corresponding to those given above. At that time *Mathematica* needed about 20 minutes to get the results.

*) W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery: Numerical Recipies in C. 2nd ed., 1992,

Cambridge University Press.

11.2.4.1 Stiff Differential Equations

Now the difficulties occuring in the numeric solution of a stiff differential equation are discussed at an example.
These are notoriously difficult in numeric solutions.

```
sys = y'[t] == y[t] - t^2;  DSolve[sys, y[t], t]
```

$$\{\{y[t] \to 2 + 2\,t + t^2 + e^t\,C[1]\}\}$$

This is the general solution. Note that it consists of a polynomial and of an exponential function. The latter grows much faster with increasing time as compared to the former; in general, it will win over the polynomial in the long run. But there are special initial data, which render the constant C[1] = 0. But in a numeric solution round-off errors mix it into the solution so that its initial suppression gets lost gradually. This explains why both numeric solutions deviate from the corresponding analytic solution (displayed by dashed curves).

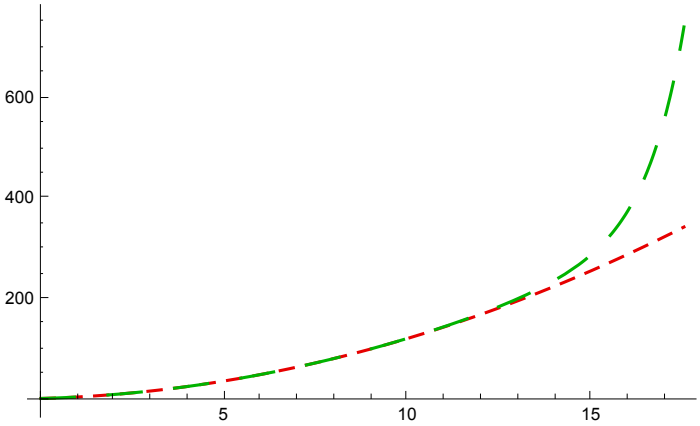
```
anf1 = y[0] == 2; so1 = DSolve[{sys, anf1}, y[t], t]
```

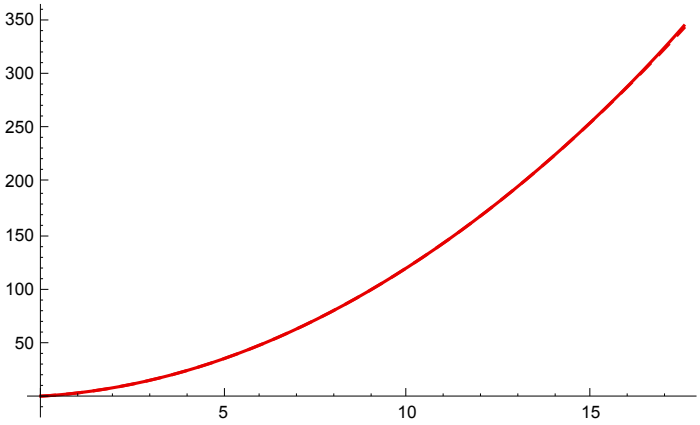
$$\{\{y[t] \to 2 + 2\,t + t^2\}\}$$

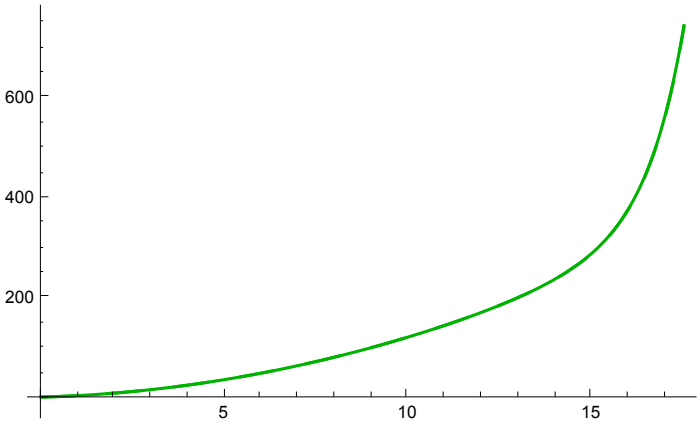
```
anf2 = y[0] == 2.00001 ; so2 = DSolve[{sys, anf2}, y[t], t]
```

$$\{\{y[t] \to 0.00001\,\left(200\,000. + 1.\,e^t + 200\,000.\,t + 100\,000.\,t^2\right)\}\}$$

```
tmax = 17.5`; dino = DisplayFunction → Identity;
p1 = Plot[Evaluate[y[t] /.so1], {t, 0, tmax}, Evaluate[dino],
    PlotStyle → {{RGBColor[0.9`, 0, 0], Dashing[{0.02`}]}}] ;
p2 = Plot[Evaluate[y[t] /.so2], {t, 0, tmax}, Evaluate[dino],
    PlotStyle → {{RGBColor[0, 0.7`, 0], Dashing[{0.05`}]}}] ;
Show[p1, p2, PlotRange → All]
no1 = Flatten[NDSolve[{sys, anf1}, y, {t, 0, tmax}]];
no2 = Flatten[NDSolve[{sys, anf2}, y, {t, 0, tmax}]];
pn1 = Plot[Evaluate[y[t] /.no1], {t, 0, tmax}, PlotStyle → RGBColor[0.9`, 0, 0]];
pn2 = Plot[Evaluate[y[t] /.no2], {t, 0, tmax}, PlotStyle → RGBColor[0, 0.7`, 0]];
Show[pn1, p1, PlotRange → All]
Show[pn2, p2, PlotRange → All]
```

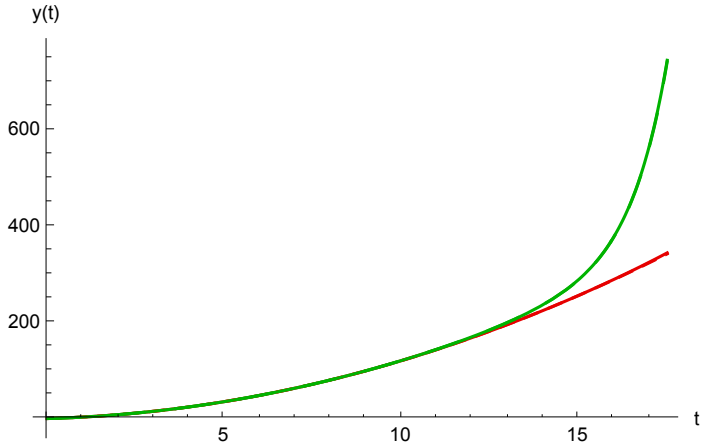






```
Show[p1, pn1, pn2, p2, PlotRange → All,
 AxesLabel → {"t", "y(t)"}, BaseStyle → {FontSize → 10},
 PlotLabel → Row[{"Red = sol1,  Green = sol2,  Dashed = numeric\n\n"}]]
```

Red = sol1, Green = sol2, Dashed = numeric



The routines *Mathematica* uses now are so good that the differences visible in earlier versions are no longer present.