Numerical Solution of ODE

Euler and Implicit Euler

```
> restart;
> with(DEtools):
> with(plots):
```

The package plottools contains more functions for plotting, especially a function to draw a single line:

```
> with(plottools):
> with(LinearAlgebra):
```

The population model again

In the previous worksheet, we solved an ordinary differential equation for a population model:

```
> popeq := diff(p(t),t) = (0.03 - 0.0005*p(t))*p(t);
```

$$popeq := \frac{d}{dt} p(t) = (0.03 - 0.0005\, p(t))\, p(t) \qquad \textbf{(1.1.1)}$$

```
> p0 := 2;
```

$$p0 := 2 \qquad \textbf{(1.1.2)}$$

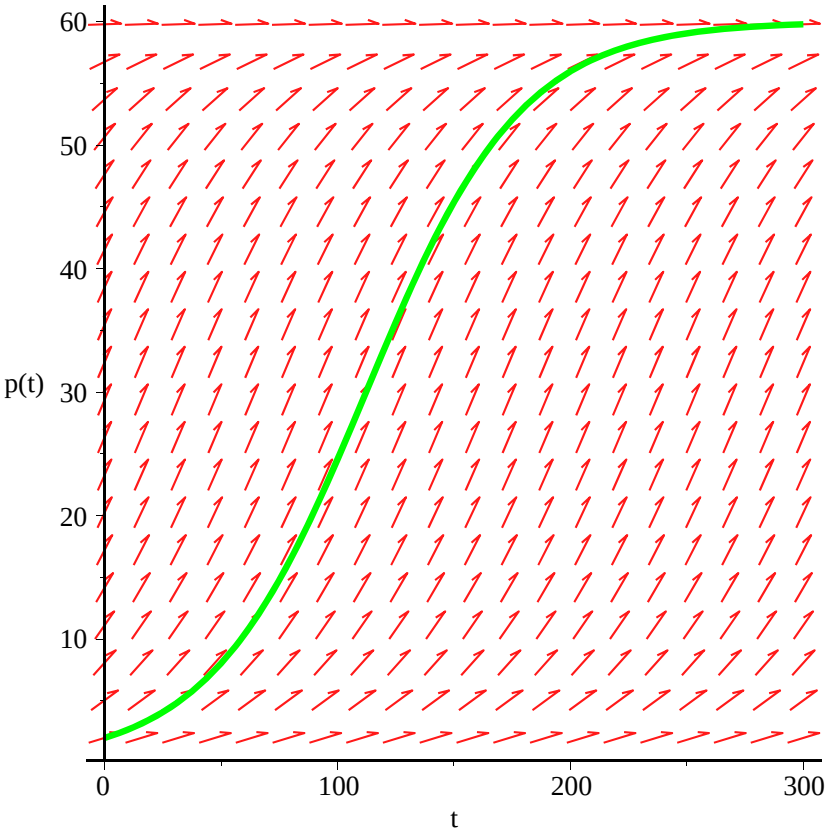
```
> dsolve({popeq, p(0)=p0},p(t));
```

$$p(t) = \frac{60}{1 + 29\, e^{-\frac{3t}{100}}} \qquad \textbf{(1.1.3)}$$

```
> sol := rhs(%);
```

$$sol := \frac{60}{1 + 29\, e^{-\frac{3t}{100}}} \qquad \textbf{(1.1.4)}$$

```
> DEplot({popeq},[p(t)],t=0..300,[[p(0)=p0]],linecolor=
  GREEN);
```



As we have a formula for the solution, we can easily compute the solution for some value of t - say, TEnd=100. We store this value and the corresponding DEplot-graph:

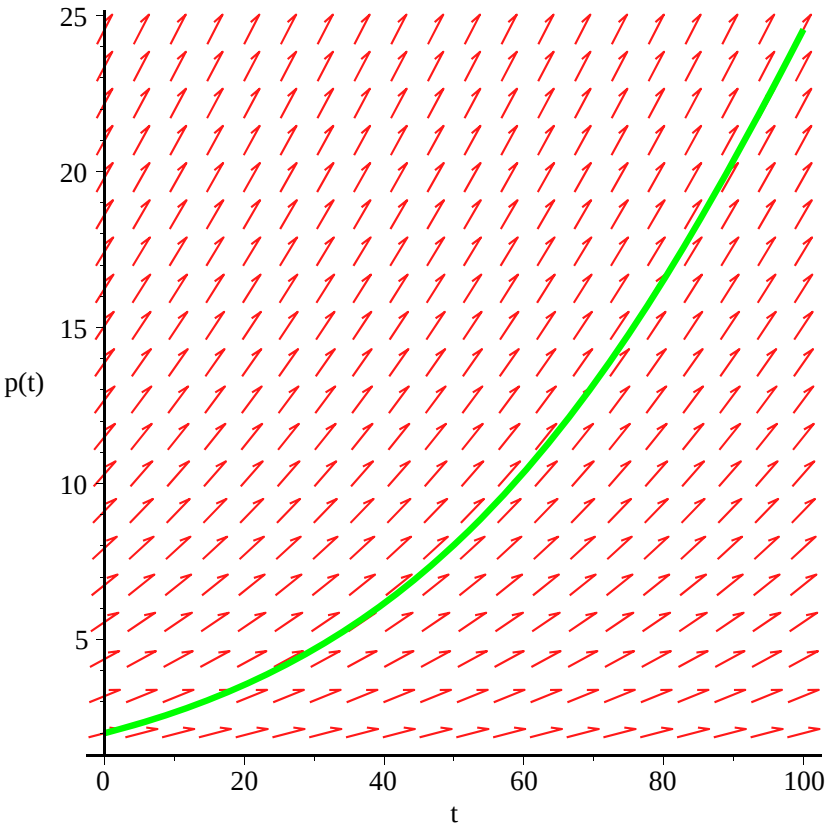
```
> TEnd := 100;
```

$$TEnd := 100 \qquad \textbf{(1.1.5)}$$

```
> pend := evalf(subs(t=TEnd, sol));
```

$$pend := 24.55167634 \qquad \textbf{(1.1.6)}$$

```
> graph_exact := DEplot({popeq},[p(t)],t=0..TEnd,[[p(0)=p0]
  ],linecolor=GREEN):
> display(graph_exact);
```



Preparation for the numerical solution

Although we get an exact solution for this problem, we will solve it again numerically (so we can compare the resulting approximation with the exact solution).
We will compute an approximate solution for the value of $pend = p(TEnd)$ from above.

First, for some integer number n, we divide the Interval [0,Tend] into n subintervals - here, we choose all subintervals of the same length delta_t = Tend/n:

```
> n := 5;
```

$$n := 5 \qquad \textbf{(1.2.1)}$$

```
> tau := TEnd/n;
```

$$\tau := 20 \qquad \textbf{(1.2.2)}$$

The numerical integration will compute values $pnum_k$, k=1..n+1 for $p(tp_k)$ with $tp_k = (k-1)\ delta\_t$. We store the grid points *tp* into a vector:

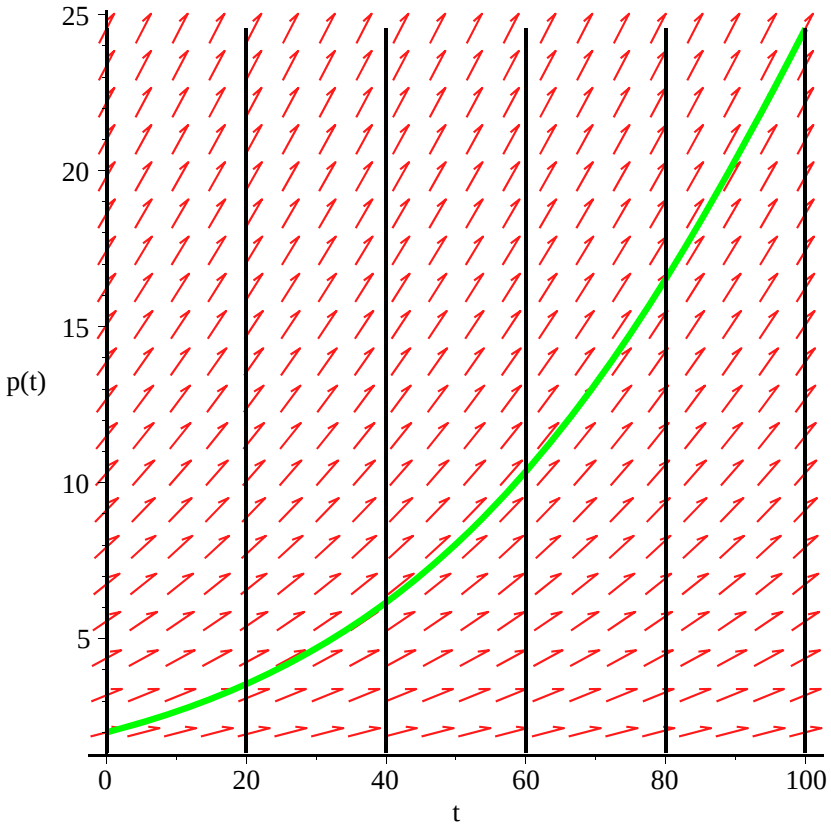
```
> tp := <seq((k-1)*tau,k=1..n+1)>;
```

(1.2.3)

$$tp := \begin{bmatrix} 0 \\ 20 \\ 40 \\ 60 \\ 80 \\ 100 \end{bmatrix} \quad \textbf{(1.2.3)}$$

We can draw the intervals with respect to t in the direction field:

```
> grid := seq(line([tp[k],0],[tp[k],pend]),k=1..n+1):
> display([graph_exact,grid]);
```



Create an vector with n+1 elements pp[1]...pp[n+1] (the method itself will not require this storage, but we save all intermediate results, so that we can plot them afterwards):

```
> pp := Vector(1..n+1);
```

$$pp := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \textbf{(1.2.4)}$$

We know the first entry of p from the initial condition:

```
> pp[1] := p0;
```

$$pp_1 := 2 \tag{1.2.5}$$

Euler's method

To compute the other values p_k, k=2..n+1, we use Euler's method, that approximates p_{k+1} by $p_k + delta\_t\, f(t_k, p_k)$, where $f(t, p)$ is the right hand side of the differential equation:

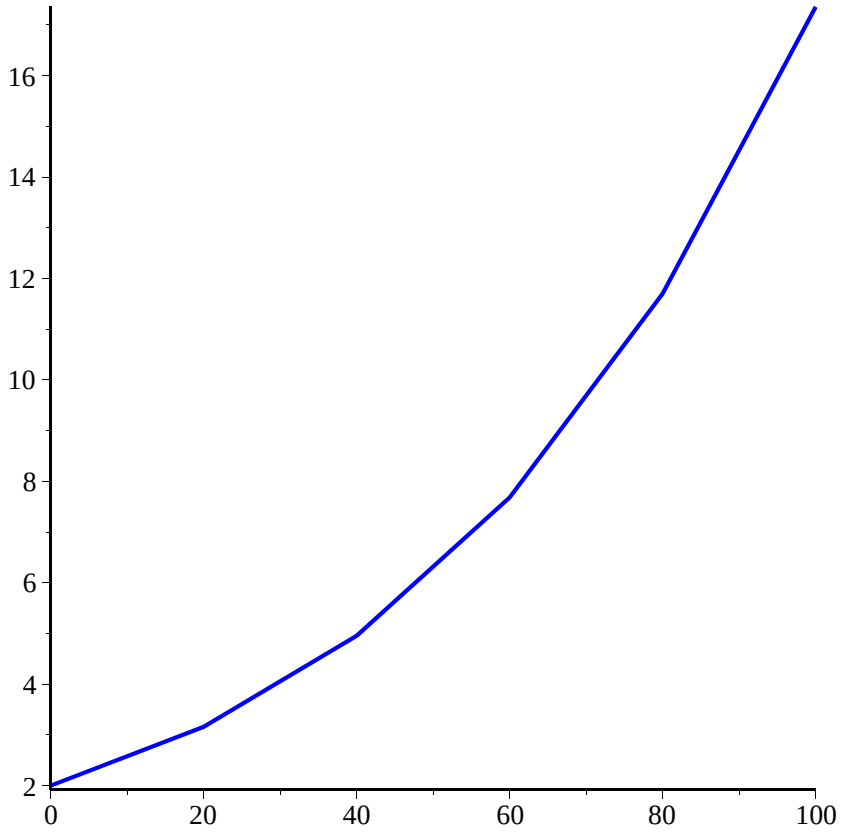
```
> for k from 1 to n do
    ft := subs({p(t)=pp[k]},rhs(popeq));
    pp[k+1] := pp[k] + tau*ft
  od;
```

$$ft := 0.0580$$
$$pp_2 := 3.1600 \tag{1.3.1}$$
$$ft := 0.08980720000$$
$$pp_3 := 4.956144000$$
$$ft := 0.1364026383$$
$$pp_4 := 7.684196766$$
$$ft := 0.2010024630$$
$$pp_5 := 11.70424603$$
$$ft := 0.2826326933$$
$$pp_6 := 17.35689990$$

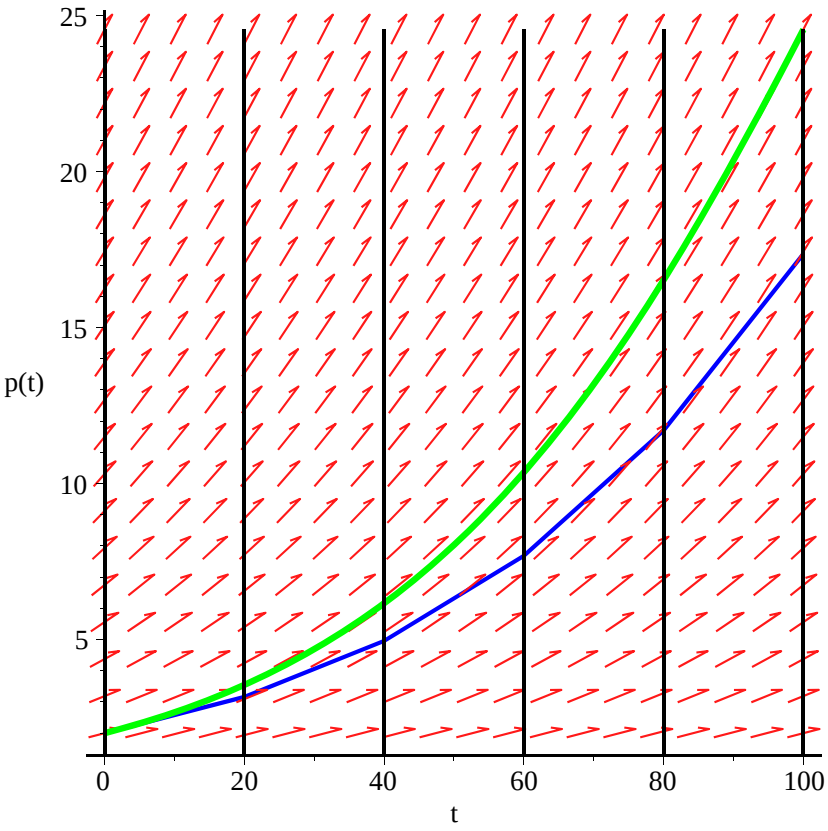
```
> tp := <seq((k-1)*tau,k=1..n+1)>:
  points := [seq([tp[k],pp[k]],k=1..n+1)];
```

$$points := [[0, 2], [20, 3.1600], [40, 4.956144000], [60, 7.684196766], [80, 11.70424603], [100, 17.35689990]] \tag{1.3.2}$$

```
> pointplot(points,style=LINE,thickness=2,color=BLUE);
```




```
> display([%,graph_exact,grid]);
```



Implicit Euler

Using the implicit Euler method, we no longer have an explicit formula for the solution in the next time step.
In principle, we would like to do something like this:

```
> n := 5;
  tau := TEnd/n;

  pp := Vector(1..n+1);
  pp[1] := p0;

  for k from 1 to n do
    # note the change: pp[k+1] instead of pp[k]
    ft := subs(p(t)=pp[k+1],rhs(popeq));
    pp[k+1] := pp[k] + tau*ft
  od;
```

$$n := 5 \qquad \textbf{(1.4.1)}$$
$$\tau := 20$$

$$pp := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$pp_1 := 2$$
$$ft := 0.$$
$$pp_2 := 2.$$
$$ft := 0.$$
$$pp_3 := 2.$$
$$ft := 0.$$
$$pp_4 := 2.$$
$$ft := 0.$$
$$pp_5 := 2.$$
$$ft := 0.$$
$$pp_6 := 2.$$

However, this doesn't work - we have to solve an equation in every step, instead:

```
> pp := Vector(1..n+1);
  pp[1] := p0;
  for k from 1 to n do
    # use pnew as the value in the next step
    ft := subs(p(t)=pnew,rhs(popeq));
    stepeq := pnew = pp[k] + tau*ft;
    # the quadratic equation stepeq has two solutions
    # in this example, we can alsways take the bigger
  (positive) one
    pp[k+1] := max( solve( stepeq, pnew ) );
  od;
```

$$pp := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \textbf{(1.4.2)}$$

$$pp_1 := 2$$
$$ft := (0.03 - 0.0005\, pnew)\, pnew$$
$$stepeq := pnew = 2 + 20\,(0.03 - 0.0005\, pnew)\, pnew$$
$$pp_2 := 4.494897428$$
$$ft := (0.03 - 0.0005\, pnew)\, pnew$$
$$stepeq := pnew = 4.494897428 + 20\,(0.03 - 0.0005\, pnew)\, pnew$$
$$pp_3 := 9.146007322$$
$$ft := (0.03 - 0.0005\, pnew)\, pnew$$
$$stepeq := pnew = 9.146007322 + 20\,(0.03 - 0.0005\, pnew)\, pnew$$

$$pp_4 := 16.25742313$$

$$ft := (0.03 - 0.0005\,pnew)\,pnew$$

$$stepeq := pnew = 16.25742313 + 20\,(0.03 - 0.0005\,pnew)\,pnew$$

$$pp_5 := 25.00824717$$

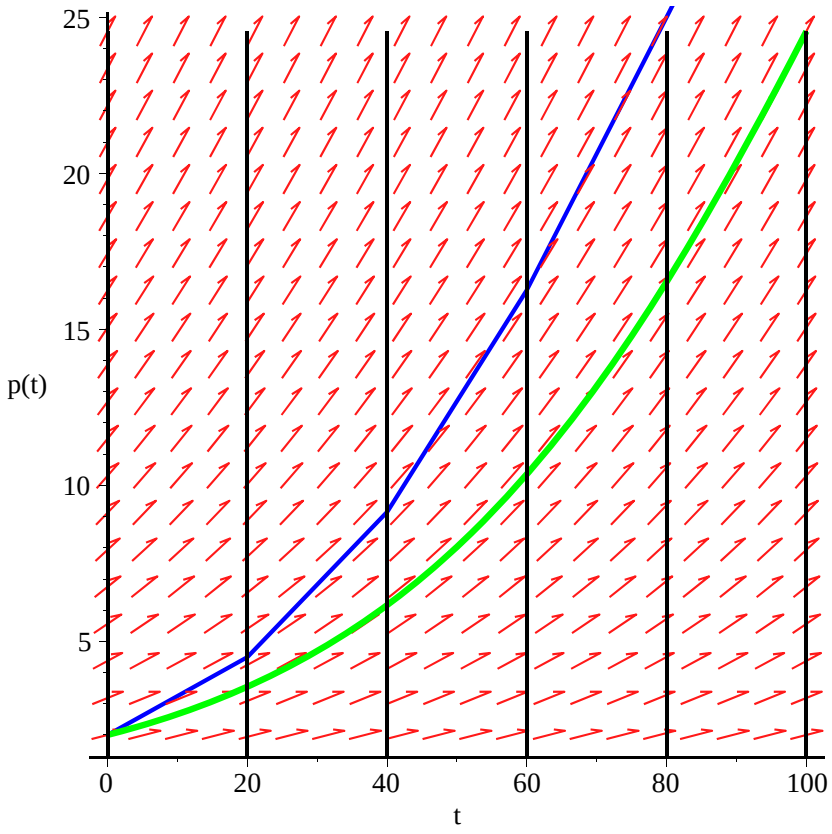
$$ft := (0.03 - 0.0005\,pnew)\,pnew$$

$$stepeq := pnew = 25.00824717 + 20\,(0.03 - 0.0005\,pnew)\,pnew$$

$$pp_6 := 33.85930483$$

```
> tp := <seq((k-1)*tau,k=1..n+1)>:
  points := [seq([tp[k],pp[k]],k=1..n+1)];
  impplot := pointplot(points,style=LINE,thickness=2,color=
  BLUE):
  display([impplot,graph_exact,grid]);
```

points := [[0, 2], [20, 4.494897428], [40, 9.146007322], [60, 16.25742313], [80, 25.00824717], [100, 33.85930483]]



Possible Problems

Stability

```
> stabeq := diff(y(t),t) = -2*y(t) + 1;
```

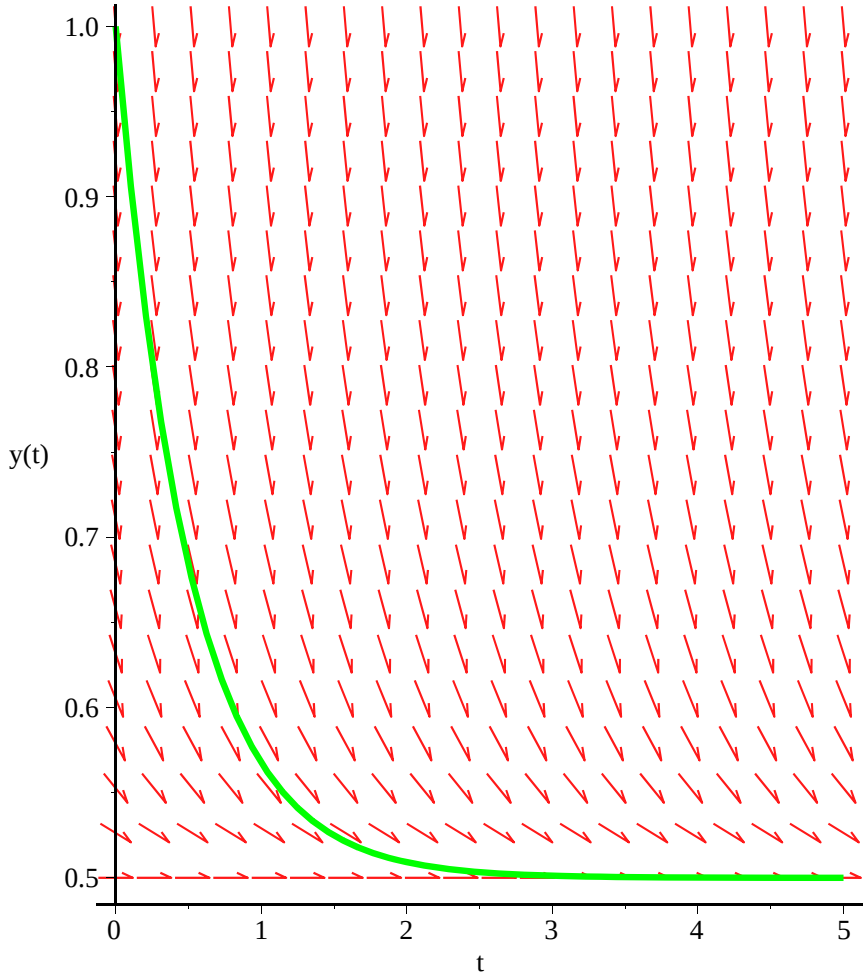
(2.1.1)

$$stabeq := \frac{d}{dt} y(t) = -2\,y(t) + 1 \qquad \textbf{(2.1.1)}$$

```
> dsolve({stabeq, y(0)=1}, y(t)):
  stabsol := rhs(%);
```

$$stabsol := \frac{1}{2} + \frac{1}{2}\,e^{-2t} \qquad \textbf{(2.1.2)}$$

```
> graph_exact := DEplot({stabeq},[y(t)],t=0..5,[[y(0)=1]],
  linecolor=GREEN):
  display(graph_exact);
```




```
> TEnd := 5; n := 5; tau := TEnd/n;
```

$$TEnd := 5 \qquad \textbf{(2.1.3)}$$
$$n := 5$$
$$\tau := 1$$

We apply the midpoint rule:

```
> pp := Vector(1..n+1);
  pp[1] := 1;
  pp[2] := evalf( subs(t=tau,stabsol) );
```

```
  for k from 2 to n do
    # use pnew as the value in the next step
    ft := subs( y(t)=pp[k], rhs(stabeq) );
    pp[k+1] := pp[k-1] + 2*tau*ft;
  od;
```

$$pp := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \textbf{(2.1.4)}$$

$$pp_1 := 1$$

$$pp_2 := 0.5676676416$$

$$ft := -0.135335283$$

$$pp_3 := 0.729329434$$

$$ft := -0.458658868$$

$$pp_4 := -0.3496500944$$

$$ft := 1.699300189$$

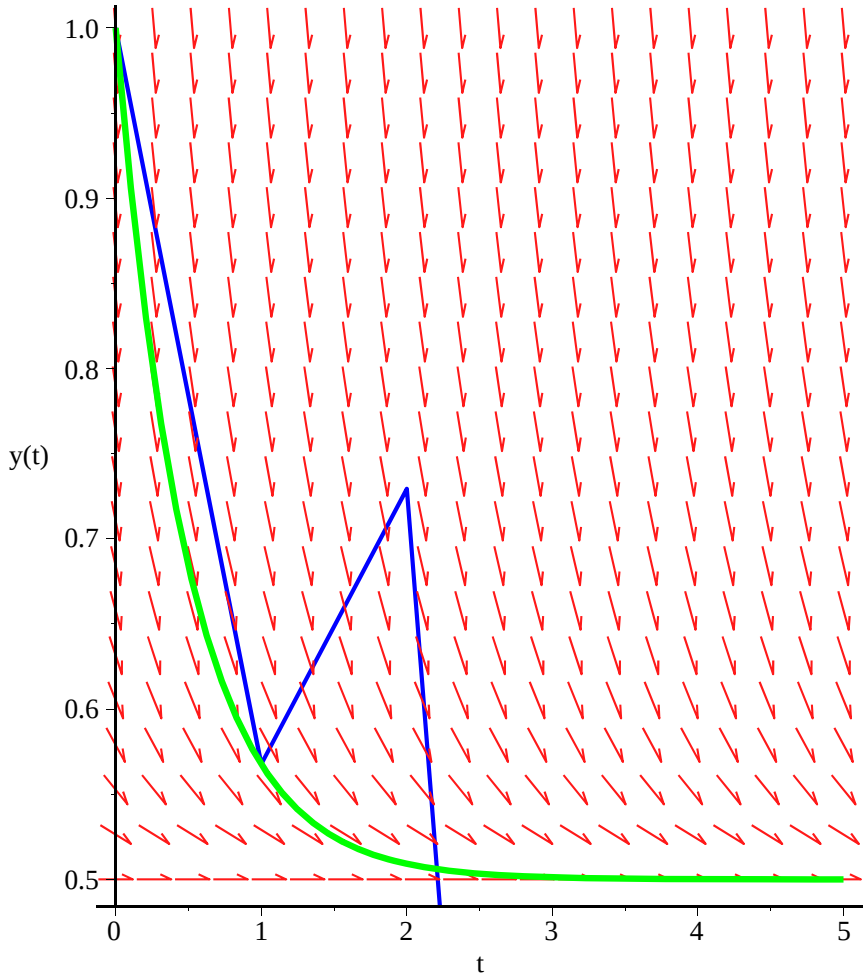
$$pp_5 := 4.127929812$$

$$ft := -7.255859624$$

$$pp_6 := -14.86136934$$

```
> tp := <seq((k-1)*tau,k=1..n+1)>:
  points := [seq([tp[k],pp[k]],k=1..n+1)];
  impplot := pointplot(points,style=LINE,thickness=2,color=
  BLUE):
  display([impplot,graph_exact]);
```

$points := [[0, 1], [1, 0.5676676416], [2, 0.729329434], [3, -0.3496500944], [4, 4.127929812], [5, -14.86136934]]$



Stiff Equation

```
> stiffeq := diff(y(t),t) = 100*( 1-y(t) );
```

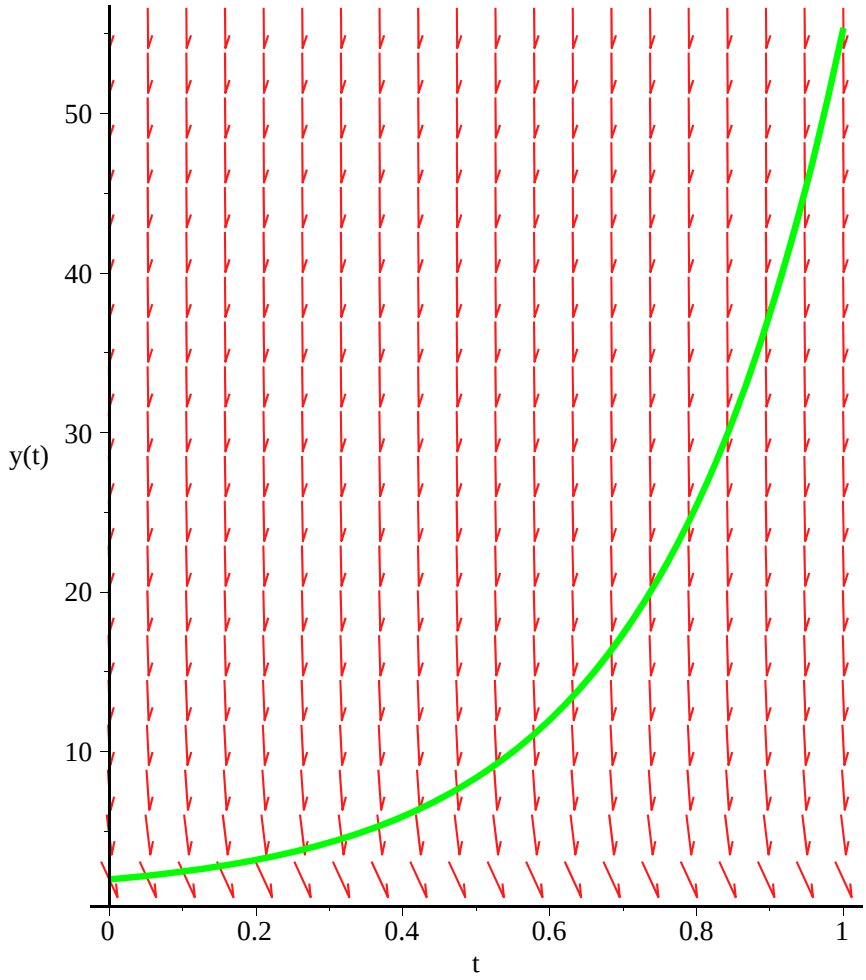
$$stiffeq := \frac{d}{dt} y(t) = 100 - 100\, y(t) \qquad \textbf{(2.2.1)}$$

```
> dsolve({stiffeq, y(0)=2}, y(t)):
  stiffsol := rhs(%);
```

$$stiffsol := 1 + e^{-100\,t} \qquad \textbf{(2.2.2)}$$

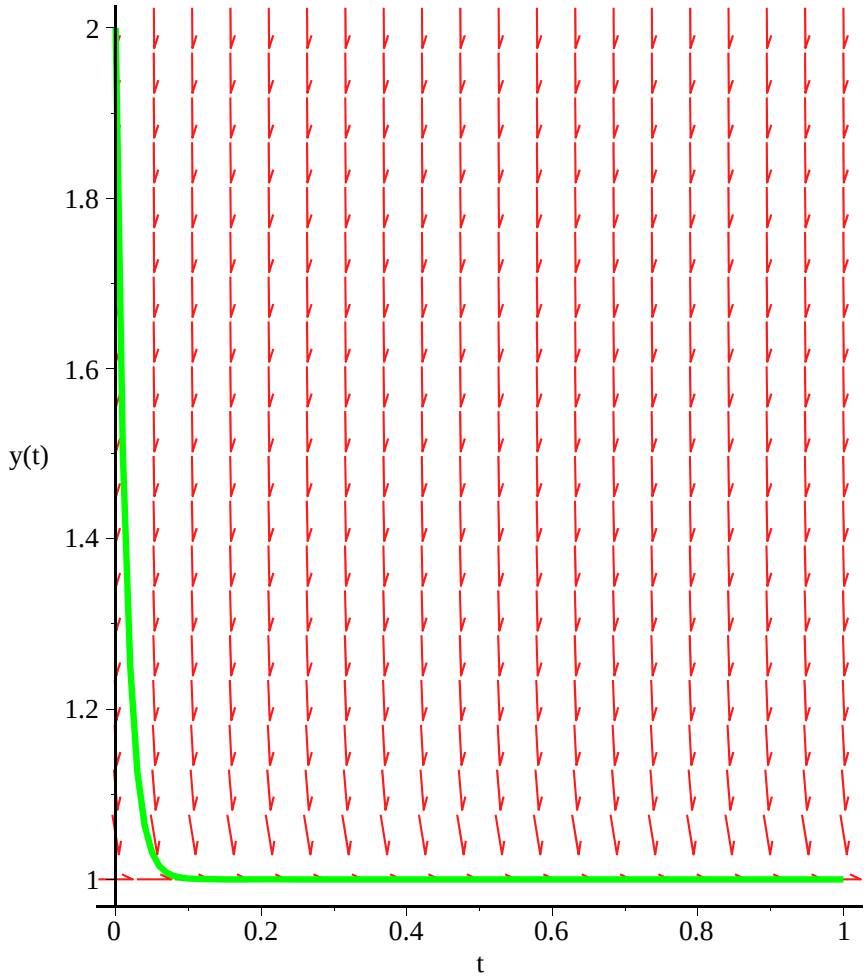
Maple's built-in Runge-Kutta-2nd-order scheme fails

```
> graph_exact := DEplot({stiffeq},[y(t)],t=0..1,[[y(0)=2]],
  linecolor=GREEN,method=classical[heunform]):
  display(graph_exact);
```



Unless we specify a very small stepsize:

```
> graph_exact := DEplot({stiffeq},[y(t)],t=0..1,[[y(0)=2]],
  stepsize=0.01,linecolor=GREEN,method=classical[heunform]):
  display(graph_exact);
```



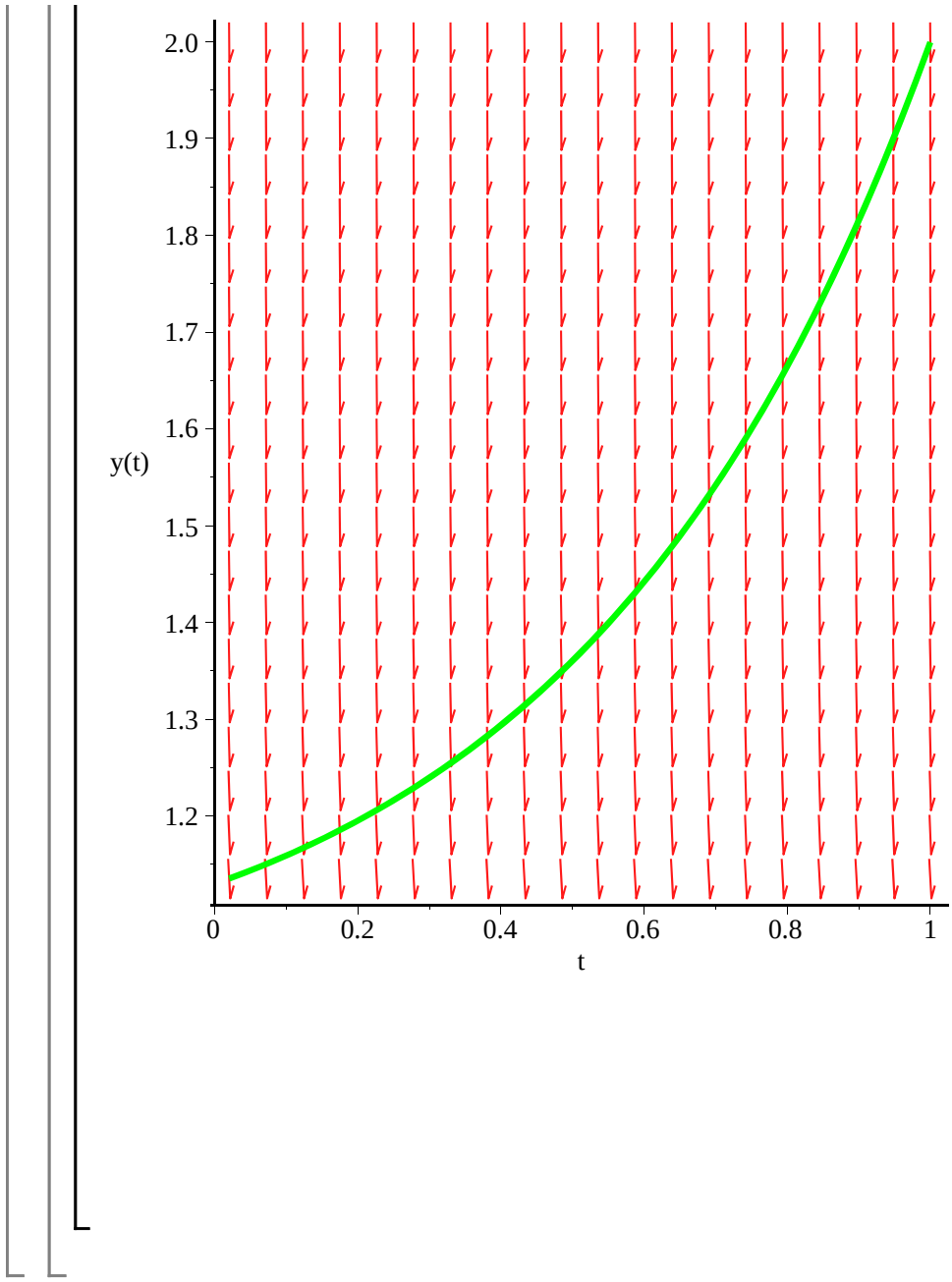
Maybe it's easier, if we start a little bit later (where the solution is less steep)?

```
> newstart := evalf( subs(t=0.02, stiffsol) );
```

newstart := 1.135335283 **(2.2.3)**

Unfortunately not . . . (but try t=0.1 as start ...)

```
> graph_exact := DEplot({stiffeq},[y(t)],t=0.02..1,[[y(0)=
  newstart]],linecolor=GREEN,method=classical[heunform]):
  display(graph_exact);
```


2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
y(t)
0
0.2
0.4
0.6
0.8
1
t