Exploding Equation and High Precision Numbers

Ulrich Mutze ,www.ulrichmutze.de, 2010-01-14

We consider the differential equation $\frac{d}{dt}h(t) = h(t)^2$ with the initial condition $h(0)=1$. The solution obviously is $h(t) = \frac{1}{1-t}$ and thus tends to infinity as t tends to 1. We now consider the discrete approximation to this solution as provided by the asynchronous leap-frog method. Since the evolution step does not involve potentially undefined operations, any such discrete approximation is well-defined for all its t-value (which we assume here to form an equidistant chain $t_n = n\,dt$). Of course, these values will grow dramatically and will soon transcend what can be represented even with *Mathematica*'s arbitrary precision numbers. Since the asynchronous leap-frog method is a reversible integration method, we should be able to go on each finite discrete trajectory back to its initial point. If the final point was 'close to infinity' the computation needs to be done with a large number of digits in order to come back to the initial point. This is what the following interactive graphics allows to study.

As this graphics is set up, it shows a reversible trajectory. Switching to using machine precision shows that that the trajectory in forward direction is created completely but the reversed part consists of a single point (although no overflow is reported).

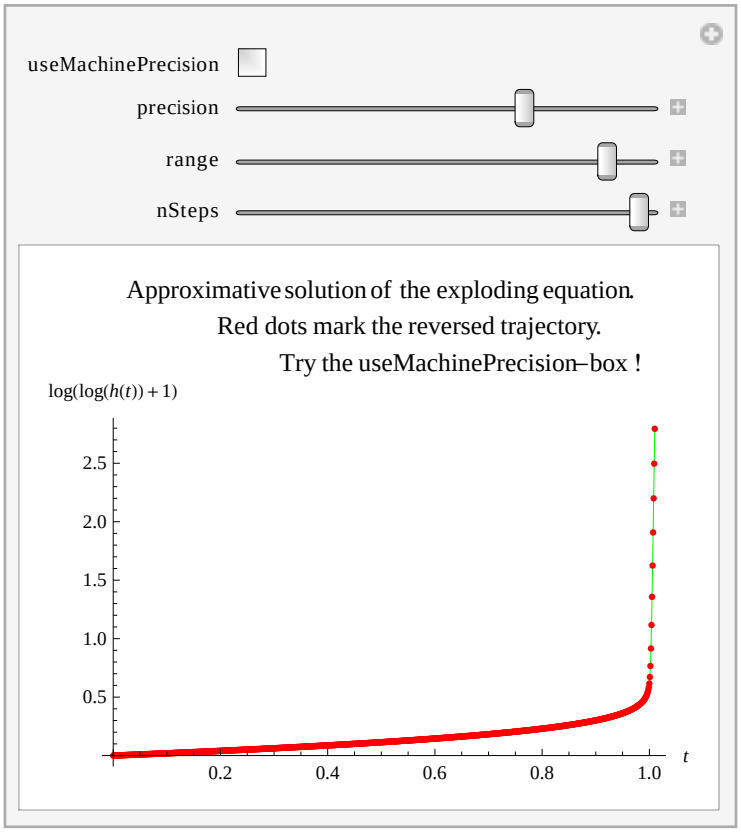
In[1]:=
```
f[y_] := y * y (* defines the right-hand side of the differential equation *)
dis[y_] := Log10[Log10[y] + 1] (* quantity for graphical representation, dis[1]==0 *)
```

In[3]:=
```
step[state[x_, v_, t_], dt_] :=
 (* evolution step for the asynchronous leap-frog integrator *)
 Module[{τ = dt / 2, ta = t, xa = x, va = v},
  (* Replacing dt/2 by dt*0.5 lets the following
   computation no longer depend on 'precision' (what is  n o t  wanted).*)
  ta += τ;
  xa += va τ;
  va = 2 f[xa] - va;
  xa += va τ;
  (*Print[xa]*)
  (* The last value before overflow turned out to be 1.821048...*10^174965752 both for
    useMachinePrecision==True and useMachinePrecision==False. In the latter case the
    number of the further digits depends on precision *)
  ta += τ;
  state[xa, va, ta] (* return value *)
 ]

time[state[x_, v_, t_]] := t (* access to state data *)
h[state[x_, v_, t_]] := x  (* access to state data *)

Manipulate[Module[{i, c0, c1, rangeStep, tMax, dt, x0, t0, v0, h0, s0, val, valr, lp, lpr},
  (* The basic interactive functionality of the notebook *)
  c0 = If[useMachinePrecision, N[0], N[0, precision]];
  c1 = If[useMachinePrecision, N[1], N[1, precision]];
  (* should make all the following computation to be done with that precision *)
  t0 = c0;
  x0 = c1;
  rangeStep = c1 / 100;
  (*rangeStep=c1*0.01;*)
  (* Replacing the previous definition by the present one lets the following
   computation no longer depend on 'precision'
   (what is  n o t  wanted). The general rule seems to
   be that floating point numbers, written with a decimal
   point enforce using machine precision
   computation in all expressions which depend on that floating point number *)
  tMax = c1 + rangeStep * range;
  dt = tMax / nSteps;
  v0 = f[x0];
  s0 = state[x0, v0, t0];
  h0 = x0; val = {{t0, dis[h0]}}; i = 0;
  While[i < nSteps, s0 = step[s0, dt];
   t0 = time[s0]; h0 = h[s0]; val = Append[val, {t0, dis[h0]}]; i++];
  lp = ListLinePlot[val, PlotStyle → Green, PlotRange → All,
    AxesLabel → {t, log[1 + log[h[t]]]},
    PlotLabel → "Approximative solution of the exploding equation.
        Red dots mark the reversed trajectory.
        Try the useMachinePrecision-box ! "];
  valr = {{t0, dis[h0]}}; i = 0;
  While[i < nSteps, s0 = step[s0, -dt];
   t0 = time[s0]; h0 = h[s0]; valr = Append[valr, {t0, dis[h0]}]; i++];
  lpr = ListPlot[valr, PlotStyle → Red, PlotRange → All];
  (*doing a list plot of the reverse motion *)
  Show[{lp, lpr}]],
 {useMachinePrecision, {False, True}}, {{precision, 1400}, 10, 2000},
 {{range, 1}, -10, 2}, {{nSteps, 1000}, 1, 1000, 1}
]
```

Out[6]=



In[7]:=
```
{"Case where machine precision is superior" :>
  {nSteps = 49, precision = 20.`, range = 1, useMachinePrecision = False}}
```

Out[7]=
```
{Case where machine precision is superior :>
  {nSteps = 49, precision = 20., range = 1, useMachinePrecision = False}}
```