Leaking Bucket Equation and Reversibility

Ulrich Mutze ,www.ulrichmutze.de, 2010-01-11

There is a simple and powerful integrator (the 'asynchronous leap-frog integrator') for the general ordinary differential equation which is strictly reversible in the sense that each discrete trajectory can be reconstructed exactly if only the final state is known. It is an interesting question how such a behavior manifests itself in situations where the physical process represented by the differential equation is not reversible. We are going to study this now by considering a simple example.

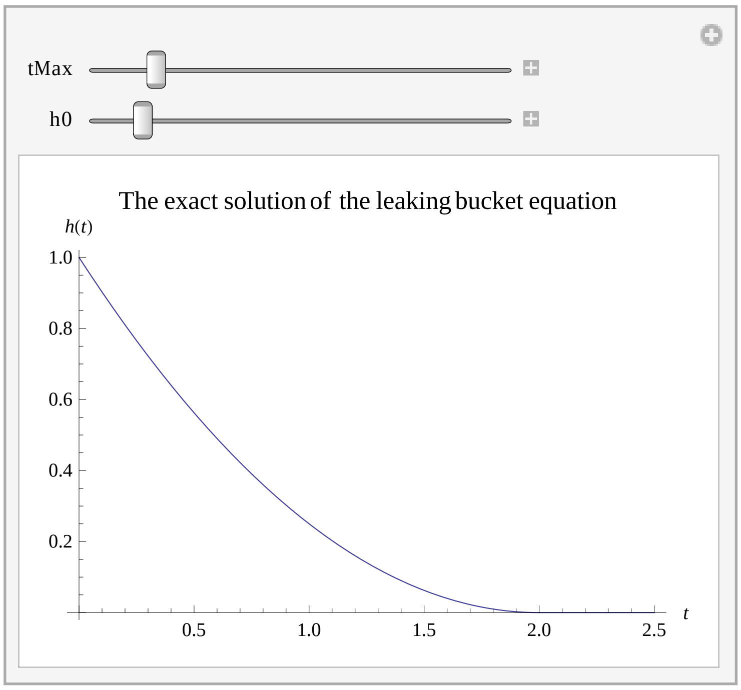
A well-known irreversible physical process is the emptying of a water-filled bucket due to a hole in the bottom. The water level *h* above the bottom is a decreasing function of time, for which self-suggesting idealizations and considerations (including the choice of adjusted units of length and time) yield the differential equation defined within the following DSolve function call. Unfortunately *Mathematica* 7 does not find the solution, although it is straigtforward to find for a human calculator. The solution which satisfies the initial condition $h(0) = h_0$ is given as a function definition in (1.3).

```
In[1]:= f[h_] := If[h < 0, 0, -Sqrt[h]] (* 1.1 *)
        DSolve[{h'[t] == f[h[t]], h[0] == h0}, h[t], t] (* 1.2 *)
        hExact[t_, h0_] := If[t > 2 Sqrt[h0] , 0, h0 - t Sqrt[h0] + (t / 2)^2] (* 1.3 *)
```

The exact solution for various initial conditions h_0 and time ranges can be studied in the next interactive display:

```
In[4]:= Manipulate[
         Plot[hExact[t, h0], {t, 0, tMax}, AxesLabel → {t, h[t]},
          PlotLabel → "The exact solution of the leaking bucket equation"],
         {{tMax, 2.5}, 0.1, 20}, {{h0, 1}, 0.1, 10}
        ]
```

Out[4]=



Integrator according to the asynchronous leap - frog method. Notice that the function stepALF is defined to take two arguments, the first being of the 'type' *state*. In *Mathematica* objects have no internal state; the data associated with them have to come with the arguments *x*, *v*, and *t*.

```
In[5]:= stepALF[state[x_, v_, t_], dt_] :=
         Module[{τ = dt / 2, ta = t, xa = x, va = v},
          ta += τ;
          xa += va τ;
          va = 2 f[xa] - va;
          xa += va τ;
          ta += τ;
          state[xa, va, ta] (* return value *)
         ] (* 1.4 *)
```

Integrator according to the Runge-Kutta second order method. To have the same logic as for the leap-frog method, the velocity is made part of the state data. Unlike the leap-frog method, the present method needs memory for intermediary state data and more than one evaluations of *f*. We use this method here for comparison. The method is not reversible and the difference in the behavior of the leap-frog method and the Runge-Kutta method can be seen in the next display by activating the 'useRK checkbox'.

```
In[6]:= stepRK[state[x_, v_, t_], dt_] :=
         Module[{τ = dt / 2, ta = t, xa = x, va = v, k1, k2},
          k1 = dt va;
          ta += τ;
          k2 = dt f[xa + k1 / 2];
          xa += k2;
          va = f[xa];
          ta += τ;
          state[xa, va, ta]
         ] (* 1.5 *)
```

What follows is a common interface of the two integrators :

```
In[7]:= step[state[x_, v_, t_], dt_, rk_] :=
         If[rk == True, stepRK[state[x, v, t], dt], stepALF[state[x, v, t], dt]]
```

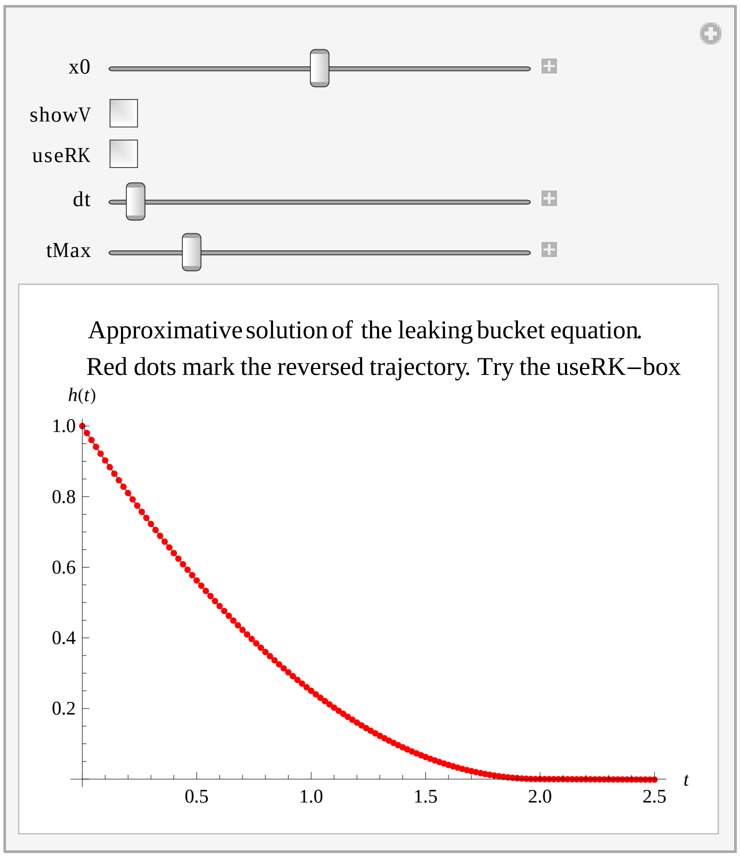
These access functions let type state behave as if it were a class with internal data (attributes):

```
In[8]:= time[state[x_, v_, t_]] := t (* 1.6 *)
        h[state[x_, v_, t_]] := x (* 1.7 *)
        hDot[state[x_, v_, t_]] := v (* 1.8 *)
```

The following interactive display lets the bucket run empty and then change the time direction and evolves the state back to the the initial point 0 in time. We may switsch between leap-frog integration and second order Runge-Kutta. Also, we may choose to inspect the velocity (rate of change of the water level) instead of the water level itself. Notice the main point: Although the leap-frog integration represents emptying with sufficient accuracy from a practical point of view, the slightly non-constant and non-zero 'empty' state still carries the information concerning the instant of actual emptying and thus allows to follow the trajectory backwards in time. Although the asynchronous leap-frog method is exactly reversible, numerical noise may prevent actual reversion of computed trajectories over very long time spans.

```
In[11]:= Manipulate[
          Module[{v0, h0, t0, s0, val, valr, lp1, lp2, tMin = 0},
           v0 = f[x0];
           h0 = If[showV == True, v0, x0];
           t0 = 0;
           val = {{t0, h0}};
           s0 = state[x0, v0, t0];
         While[t0 < tMax, s0 = step[s0, dt, useRK]; t0 = time[s0];
            h0 = If[showV == True, hDot[s0], h[s0]]; val = Append[val, {t0, h0}]];
           lp1 = ListLinePlot[val, AxesLabel → {t, h[t]},
             PlotLabel → "Approximative solution of the leaking bucket equation.
                 Red dots mark the reversed trajectory. Try the useRK-box ! "
            ]; (* doing a line plot of the natural motion *)
         valr = {{t0, h0}};
           While[t0 > tMin, s0 = step[s0, -dt, useRK]; t0 = time[s0];
            h0 = If[showV == True, hDot[s0], h[s0]]; valr = Append[valr, {t0, h0}]];
           lp2 = ListPlot[valr, PlotStyle → Red]; (*doing a list plot of the reverse motion *)
           Show[{lp1, lp2}]],
          {{x0, 1}, 0, 2}, {showV, {False, True}},
          {useRK, {False, True}}, {{dt, 0.02}, 0.001, 1}, {{tMax, 2.5}, 1, 10}]
```

Out[11]=


x0
showV
useRK
dt
tMax
Approximative solution of the leaking bucket equation.
Red dots mark the reversed trajectory. Try the useRK–box
h(t)
1.0
0.8
0.6
0.4
0.2
0.5
1.0
1.5
2.0
2.5
t