Event Handling Solver Compared to Modelica Dassl

Boucing Ball Benchmark

T.H. Gallois (IFPEN), J. Brac (IFPEN), T. Soriano (SUPMECA)

Outline

- Introduction and problem definition
- Benchmark
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
- Conclusion

Outline

- Introduction and problem definition
- Benchmark
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
- Conclusion

Introduction and problem definition

- What is a hybrid system ?

- What are the type of events ?
  - predictable events
  - unpredictable events

- What are the types of model modifications after an event ?
  - change of initial conditions
  - change of equations
  - change of the state vector

Outline

- Introduction and problem definition
- Benchmark
  - Dynamic equations
  - About the numerical scheme
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
- Conclusion

Outline

- Introduction and problem definition
- Benchmark
  - Dynamic equations
  - About the numerical scheme
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
  - Constraint vector
  - System of equations
  - Results
- Conclusion

Benchmark : The Bouncing Ball

Dynamic Equations of the boucing ball with air resistance

Air resistance : $\mathbf{F}_{drag} = -\frac{1}{2} C_x \rho_{air} S ||\mathbf{v}||\mathbf{v}$

$$\begin{cases} m\dfrac{d}{dt}v_x = -\dfrac{1}{2} C_x \rho_{air} S ||\mathbf{v}|| v_x \\ m\dfrac{d}{dt}v_y = -m\,g - \dfrac{1}{2} C_x \rho_{air} S ||\mathbf{v}|| v_y \\ \dfrac{dx}{dt} = v_x \\ \dfrac{dy}{dt} = v_y \\ \textit{Initial conditions} \\ v_x(t=0) = v_{x0} \qquad v_y(t=0) = v_{y0} \\ x(t=0) = x_0 \qquad y(t=0) = y_0 \end{cases}$$

Outline

- Introduction and problem definition
- Benchmark
  - Dynamic equations
  - About the numerical scheme
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
  - Constraint vector
  - System of equations
  - Results
- Conclusion

About the numerical scheme

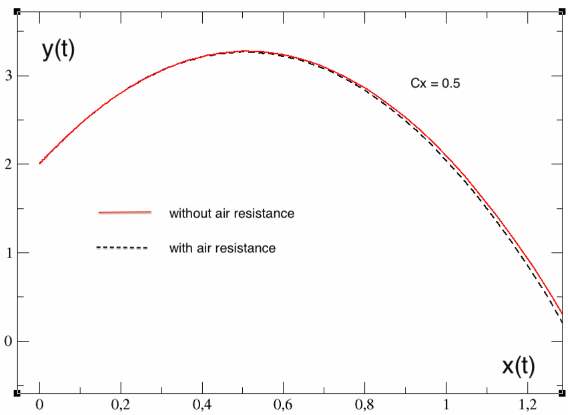
Numerical scheme : Backward Differentiation Formula of high order

$\rightarrow$ good stability

After each event : Implicit Runge-Kutta scheme

$\rightarrow$ A-stable, allows to adapt the time step at each restart of the solver

Comparison with and without air resistance



This small difference before the first bounce will be bigger and bigger after many bounces.

Outline

- Introduction and problem definition
- Benchmark
- **The constraint vector**
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
- Conclusion

The constraint vector

- Hybrid dynamic systems have several constraints. All these constraints are the component of a vector **k**.
- The sign change of a constraint triggers the occuring of an event.
- In our example **k** is a scalar and $\mathbf{k} = y - y_{ground} \geq 0$

$$
\left\{
\begin{array}{ll}
m\frac{d}{dt}v_x = -\frac{1}{2}C_x\rho_{air}S||\mathbf{v}||v_x & \\
m\frac{d}{dt}v_y = -m\,g\ -\frac{1}{2}C_x\rho S||\mathbf{v}||v_y & \\
\frac{dx}{dt} = v_x & \frac{dy}{dt} = v_y \\
\boxed{k = y - y_{ground} \geq 0} & \\
\textit{Initial conditions} & \\
v_x(t=0) = v_{x0} & v_y(t=0) = v_{y0} \\
x(t=0) = x_0 & y(t=0) = y_0
\end{array}
\right.
$$

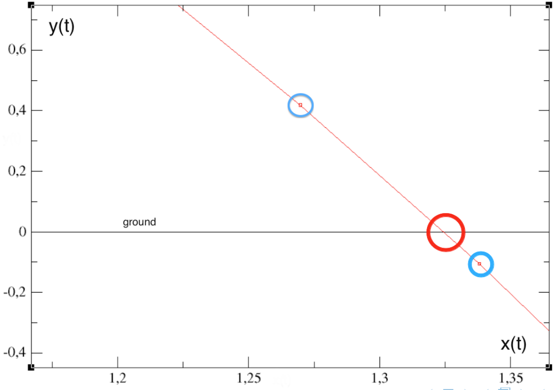
Outline

- Introduction and problem definition
- Benchmark
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
- Conclusion

Computation of the event time t*

Difficulties

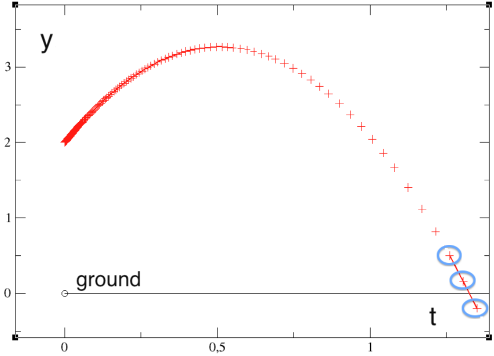
- The numerical integrator is discretized in time
  → we have the solution of the problem only at the points of the integration.
- The challenge is to compute the instant when the ball hits the ground



Computation of the event time t*

Interpolation

- The idea is to interpolate a second degree time polynomial at y_{n+1}, y_n, y_{n-1}



- Compute analytically the roots and choose the appropriate one.
- We get the time t* when the sign of the constraint changes.

Outline

- Introduction and problem definition
- Benchmark
- The constraint vector
- Computation of the event time t*
- **Computation of the solution at t***
- Results for hard spheres
- Results for soft spheres
- Conclusion

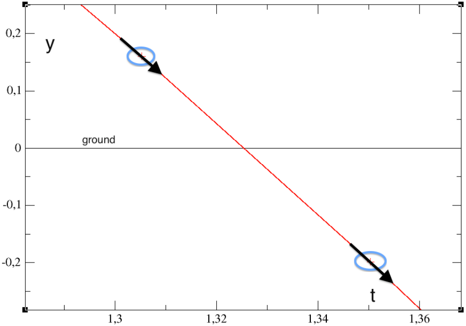
Computation of the solution at time t*

Interpolation

- Now we have t* we need to compute the solution at this time.
- If the ODE system is rewritten as $\dot{X} = f(X)$ and by considering X the state vector $X = \begin{pmatrix} v_x \\ v_y \\ x \\ y \end{pmatrix}$. The solver gives the solution X_n at a given time t_n and $X_n = \begin{pmatrix} v_{xn} \\ v_{yn} \\ x_n \\ y_n \end{pmatrix}$
  The idea is to interpolate a third degree time polynomial for each component of the state vector by considering $X_n, X_{n+1}, f(X_n), f(X_{n+1})$

Computation of the solution at time t*

Interpolation



- Evaluate this polynomial at t* to get $X^* = X(t^*)$.
- X* is used afterwards as an initial condition .

Outline

- Introduction and problem definition
- Benchmark
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- **Results for hard spheres**
- Results for soft spheres
- Conclusion

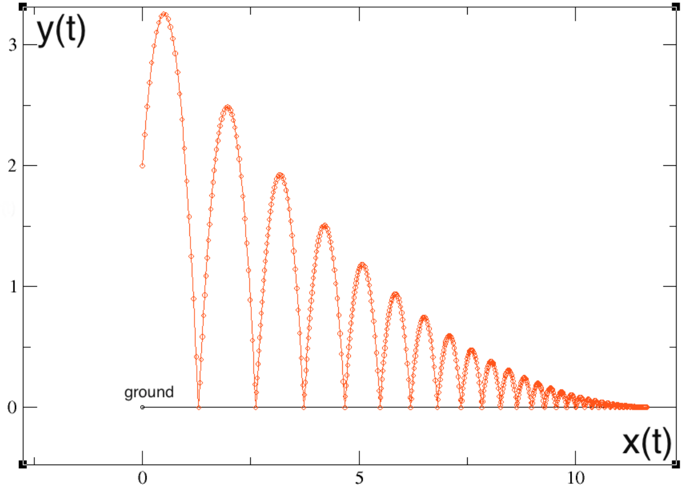
Solving the system after an event

- In hard spheres context after each event (contact with the ground) the equations do not change .
- Only the initial conditions of the vertical speed component are modified and have the opposite value at t* multiplied by a damping factor $1-\epsilon$
- Then we get the following system after the ball hits the ground :

$$
\begin{cases}
m\dfrac{d}{dt}v_x = -\dfrac{1}{2}C_x\rho_{air}S||\mathbf{v}||v_x \\
m\dfrac{d}{dt}v_y = -m\,g - \dfrac{1}{2}C_x\rho_{air}S||\mathbf{v}||v_y \\
\dfrac{dx}{dt} = v_x \qquad \dfrac{dy}{dt} = v_y \\
c = y > 0 \\
\textit{New initial conditions} \\
v_x(t=t*) = v_{x*} \qquad v_y(t=t*) = -v_{y*}(1-\epsilon) \\
x(t=t*) = x^* \qquad y(t=t*) = y^*
\end{cases}
$$

Results on a flat and sinusoidal ground

On a flat ground :



Comparison with DASSL

The bouncing ball model is implemented in Modelica code :

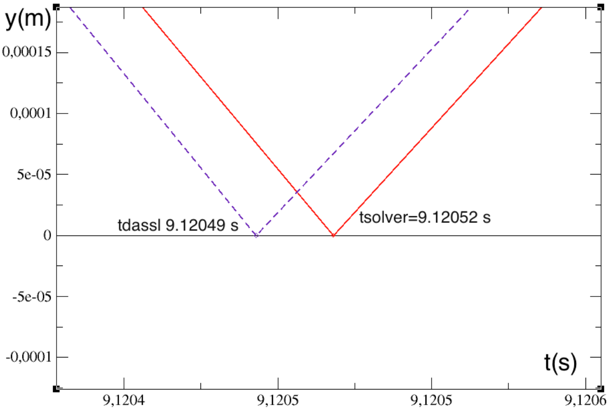
```
model BouncingBall
  Real vx(start = 1);
  Real vy(start = 5);
  Real x(start = 0);
  Real y(start = 2);
  parameter Real m = 1.1;
  parameter Real Cx = 0.5;
  parameter Real rho = 1.293;
  parameter Real S = 3.14 * 0.1 * 0.1;
  constant Real g = 9.81;
equation
  m * der(vx) = -0.5 *Cx * rho * S *
                      sqrt(vx ^ 2 + vy ^ 2) * vx;
  m * der(vy) = -m * g - 0.5 * Cx * rho * S *
                      sqrt(vx ^ 2 + vy ^ 2) * vy;
  der(x) = vx;
  der(y) = vy;
  when y <= 0 then
      reinit(vy, -0.9 * pre(vy));
  end when;
end BouncingBall;
```

Speed comparison

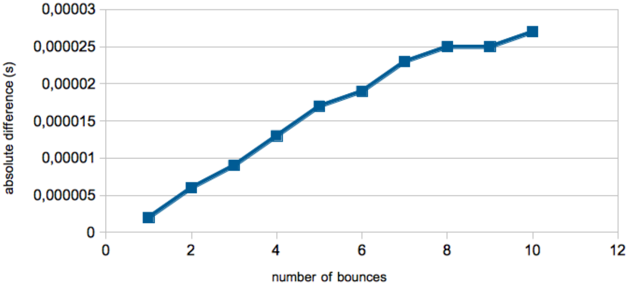
Comparison of the CPU time for the simulation of the bouncing ball during 20 seconds with DASSL and our own solver.
relative tolerance : 10^{-6}
coefficient of air friction : $C_x \rho S = 0.0203\ kg/m$.

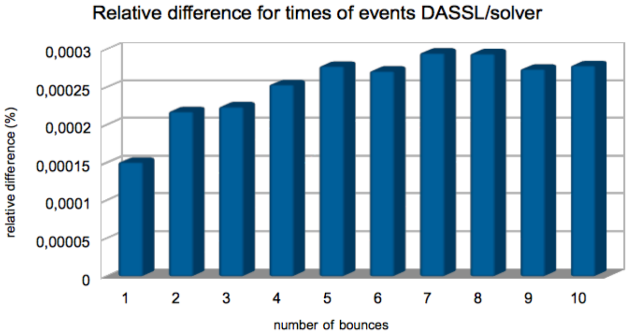
CPU time DASSL	CPU time our solver
0.068s	0.032s

Precision comparison



Absolute difference for times of events DASSL/solver





Relative difference for times of events DASSL/solver
relative difference (%)
0,0003
0,00025
0,0002
0,00015
0,0001
0,00005
0
1
2
3
4
5
6
7
8
9
10
number of bounces

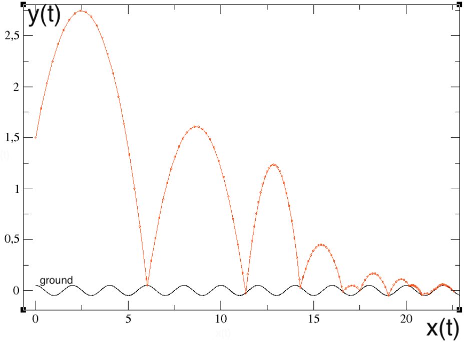
Comparison of DASSL and our solver with the analytical solution :

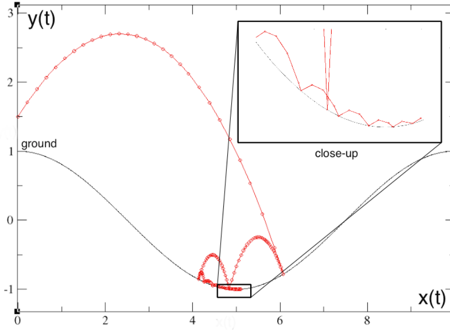
Table: Comparison of the time of the first event.

number of bounce	relative tolerance	analytical solution	DASSL	error DASSL	our solver	error our solver
1st	10^{-6}	0.640714	0.640715	$1.56\ 10^{-6}$	0.640714	0
	10^{-3}	0.640714	0.641162	$7\ 10^{-4}$	0.640730	$2.50\ 10^{-5}$
2nd	10^{-6}	1.769519	1.769523	$2.26\ 10^{-6}$	1.769515	$2.26\ 10^{-6}$
	10^{-3}	1.769519	1.769915	$2.23\ 10^{-4}$	1.769562	$2.43\ 10^{-5}$

Results on a flat and sinusoidal ground

On a sinusoidal ground :





With a small amplitude

With a big amplitude

Outline

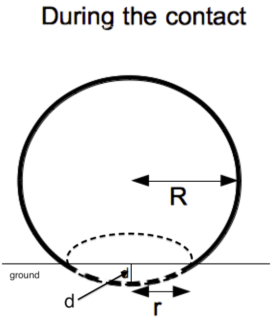
- Introduction and problem definition
- Benchmark
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
  - Constraint vector
  - System of equations
  - Results

Outline

Results in soft spheres context

Constraint vector

Now the ball is supposed to be deformed during the contact as :



As the ball is now soft we have to consider now its radius, and so the constraint vector becomes :

$$\mathbf{c} = [y(t) - R]$$

Outline

- Introduction and problem definition
- Benchmark
  - Dynamic equations
  - About the numerical scheme
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
  - Constraint vector
  - System of equations
  - Results
- Conclusion

System of equations during the contact

During the contact the equations of the model change.

We consider now the elastic force : $\mathbf{F}_{elas} = 2\pi\ E\sqrt{2R}|R-y|^{\frac{3}{2}}\mathbf{u_y}$

and the dissipative force : $\mathbf{F}_{diss} = -\mu\mathbf{v}$.

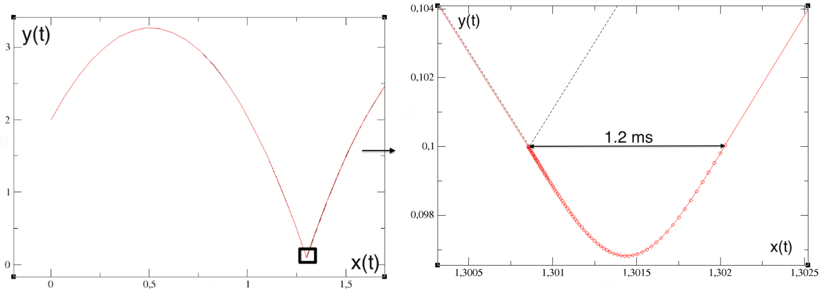
So after determining t* et X* we have the following system :

$$\begin{cases} m\dfrac{d}{dt}v_x = -\mu v_x \\ m\dfrac{d}{dt}v_y = -m\,g + 2\pi\ E\sqrt{2R}\,|R-y|^{\frac{3}{2}} - \mu v_y \\ \dfrac{dx}{dt} = v_x \qquad \dfrac{dy}{dt} = v_y \\ c = y - R \\ \textit{Initial conditions} \\ v_x(t=t*) = v_{x*} \qquad v_y(t=t*) = v_{y*} \\ x(t=t*) = x^* \qquad y(t=t*) = y^* \end{cases}$$

Outline

Results for soft spheres

After the event "the ball hits the ground" we switch to another model described by the previous system. When the ball does not touch the ground anymore there is another event and we go back to the original model.



Modelica code for the soft spheres

```
model SoftBouncingBall
  Real v_x(start = 1);
  Real v_y(start = 5);
  Real x(start = 0);
  Real y(start = 2);
  parameter Real Cx = 0.5;
  parameter Real rho = 1.293;
  parameter Real pi = 3.141592653;
  parameter Real S = pi * 0.1 * 0.1;
  parameter Real R = 0.1.
  parameter Real E=0.1*10^9;
  parameter Real density=500;
  parameter Real m = (4/3)*pi*R^3*density;
  constant Real g = 9.81;
equation
  if y > R then
    m * der(v_x) = -0.5 * Cx * rho * S * sqrt(v_x ^ 2 + v_y ^ 2) * v_x;
    m * der(v_y) = -m * g - 0.5 * Cx * rho * S * sqrt(v_x ^ 2 + v_y ^ 2) *
    der(x) = v_x;
    der(y) = v_y;
else
     m*der(v_x) = 0;
    m*der(v_y) = -m*g+2*3.14*E*sqrt(2*R)*abs(R-y)^(3/2);
     der(x) = v_x;
     der(y) = v_y;
end if;
end SoftBouncingBall;
```

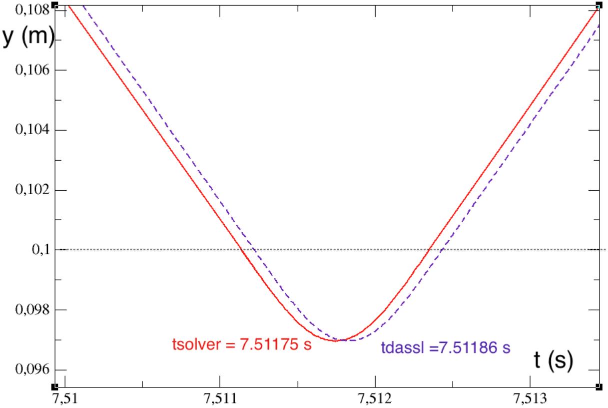
Speed comparison

Let us compare the CPU time for the simulation of the soft bouncing ball during 10 seconds with DASSL and our own solver.
relative tolerance : 10^{-6}

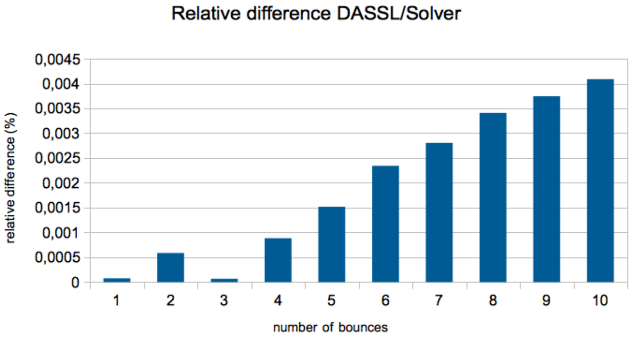
CPU time DASSL	CPU time our solver
0.061s	0.045s

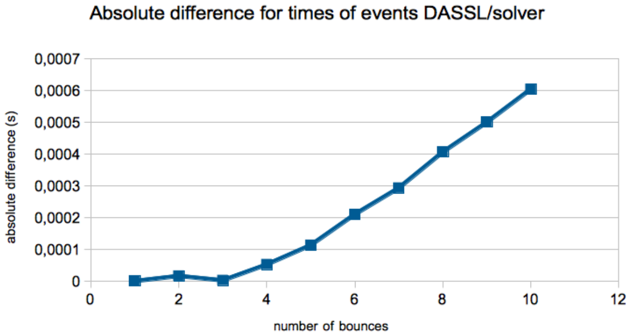
Precision comparison

Let us look both results for the tenth bounce around 7.51 s



Now we plot the absolute difference of the event times between DASSL and our own solver for the first 10 bounces





Absolute difference for times of events DASSL/solver
0,0007
0,0006
0,0005
0,0004
0,0003
0,0002
0,0001
0
absolute difference (s)
0
2
4
6
8
10
12
number of bounces

Outline

- Introduction and problem definition
- Benchmark
- The constraint vector
- Computation of the event time t*
- Computation of the solution at t*
- Results for hard spheres
- Results for soft spheres
- Conclusion

Conclusion

- Objectives realized :
  - Build an example of model with different types of events
    $\rightarrow$ change of only initial conditions : hard spheres
    $\rightarrow$ change of model : soft spheres
  - Propose a new method of event handling
    $\rightarrow$ computation of event time t*
    $\rightarrow$ computation of the solution X* at t*

- Perspectives
  - Consider many balls to get a multi-events model which needs to add a new model of collision between balls
  - Consider the rotation of the ball on itself and the Magnus effect
  - Add this benchmark to the standard library of OpenModelica in order to share it with the community