ODE solvers recursive equations

Aurelie Degletagne

June 9, 2015

Contents

1 Notations

In each following equation h is the time step, x_k,v_k,a_k the position, velocity and acceleration at the time step k, K the stiffness, M the mass, r_m the rayleigh mass, r_k the rayleigh stiffness and F_{ext} the external forces.

2 Euler solver

2.1 EulerExplicit solver

It is the simplest time integration solver.

The integration scheme is based on the following equations:

$v_{k+1} = v_k + ha_k$

$x_{k+1} = x_k + hv_k$

2.2 EulerImplicit solver

It is a semi-implicit time integrator using backward Euler scheme for first and second degree Ordinary Differential Equations (ODE). This is based on [Baraff and Witkin, Large Steps in Cloth Simulation, SIGGRAPH 1998]

The integration scheme is based on the following equations:

$v_{k+1} = v_k + h\frac{Fext_k + v_k(-(r_k+h)K - r_m M)}{h(h+r_k)K + (1+hr_m)M}$

$x_{k+1} = x_k + hv_k$

3 Newmark solver

It is an implicit time integrator using Newmark scheme. The implementation computes a_k directly then solves the first equation to compute a_{k+1}, and finally computes the new velocity v_{k+1} and the new position x_{k+1}. In Sofa one uses for Newmark coefficients $\gamma = 0.5$ and $\beta = 0.25$.

$$a_{k+1} = \frac{Fext_k + h*cstAcc*a_k + cstVel*v_k}{(h^2\beta + h\gamma r_k)K + (1 + h\gamma r_m)M} \begin{cases} cstAcc &= (-r_m(1-\gamma)M - h(0.5-\beta)K - rk(1-\gamma)K) \\ cstVel &= (-r_m M - (h + r_k)K) \end{cases}$$

$$v_{k+1} = v_k + h(1-\gamma)a_k + h\gamma a_{k+1}$$

$$x_{k+1} = x_k + hv_k + 0.5h^2(1-2\beta)a_k + h^2\beta a_{k+1}$$

4 VariationalSymplectic solver

It is explicit and implicit time integrator using the Variational Symplectic Integrator as defined in: Kharevych, L et al. Geometric, Variational Integrators for Computer Animation. ACM SIGGRAPH Symposium on Computer Animation 4 (2006): 4351. p is the momentum.

4.1 VariationalSymplecticExplicit solver

$$v_{k+1} = \frac{2(Fext_k + p_k - hr_m v_k M)}{M}$$

$$p_{k+1} = Mv_{k+1}$$

$$x_{k+1} = x_k + hv_{k+1}$$

4.2 VariationalSymplecticImplicit solver

The current implementation for implicit integration assume $alpha = 0.5$ (quadratic accuracy) and uses several Newton steps to estimate velocity.

$i = 0$, $err = 0$

while $i < newtonSteps$ && $err > newtonError$) || $(i == 0)$ **do**

The position increment res is searched such that q(k,i+1)=q(k,i)+res

*res must minimize the Lilyan, and is solution of a linearized system around previous estimate Equation is : matrix * res = b*

$$with \begin{cases} b &= f(q(k,i-1)) - K(q(k,i-1))res(i-1) + (2/h)p^(k) \\ matrix &= -K + 4/h^(2)M \end{cases}$$

$res(i) = matrix^{-1} * b$

$q(k,i) = res(i) + q(k,0)$

$resi = q(k,i) - q(k,i-1)$ save the residual as a stopping criterion

$q(k,i) = q(k,i-1)$

$err = resi.norm()/(q(k,0)).norm()$ this should decrease

$i{+}{+}$

end while

$v_{k+1} = \frac{2res(i)}{h}$

$p_{k+1} = (M + \frac{h^2}{4})v_{k+1} + \frac{h}{2}Fext_k$

$x_{k+1} = x_k + hv_{k+1}$

Notations

err current newton error

i current newton iteration

$newtonError$ error tolerance for newton iterations

$newtonSteps$ maximum number of newton steps

$q(k, i)$ estimate of the mid point position at iteration k and newton step i

res position increment

5 RungeKutta

It is a popular explicit time integration method, much more precise than euler explicit solver.

5.1 Runge Kutta second order

It is the Runge-Kutta method with order 2 or the middle point rule. Functions are evaluated two times at each step.

At time t + h/2

$x_{k+\frac{1}{2}} = x_k + \frac{1}{2}hv_k$

$v_{k+\frac{1}{2}} = v_k + \frac{1}{2}ha_k$

$a_{k+\frac{1}{2}} = \frac{Fext_{k+\frac{1}{2}}}{m}$

At time t+h

$x_{k+1} = x_k + h * v_{k+\frac{1}{2}}$

$v_{k+1} = v_k + h * a_{k+\frac{1}{2}}$

$a_{k+1} = \frac{Fext_{k+1}}{m}$

5.2 Runge Kutta fourth order

It is the Runge-Kutta method with order four. Functions are evaluated four times at each step.

Variables

$stepBy2 = \frac{h}{2}$ $stepBy3 = \frac{h}{3}$ $stepBy6 = \frac{h}{6}$

At time t :

- **Step 1**

  $k1v = v_k$

  $k1a = a_k$

- **Step 2**

  $k2x = x_k + k1v * stepBy2$

  $k2v = v_k + k1a * stepBy2$

  $k2a = (Fext_{stepBy2}(k2x, k2v))/m$

- **Step 3**

  $k3x = x_k + k2v * stepBy2$

  $k3v = v_k + k2a * stepBy2$

  $k3a = (Fext_{stepBy2}(k3x, k3v))/m$

- **Step 4**

  $k4x = x_k + k3v * h$

  $k4v = v_k + k3a * h$

  $k4a = (Fext_{k+1}(k4x, k4v))/m$

At time t + h :

$x_{k+1} = x_k + k1v * stepBy6 + k2v * stepBy3 + k3v * stepBy3 + k4v * stepBy6$

$v_{k+1} = v_k + k1a * stepBy6 + k2a * stepBy3 + k3a * stepBy3 + k4a * stepBy6$

$a_{k+1} = (Fextk + 1)/m$

6 CentralDifference solver

It is an explicit time integrator using central difference (also known as Verlet or Leap-frog). The equations are separated in two cases: either the rayleigh mass is null or not.

- if $rm = 0$

  $a_{k+1} = \frac{Fext_{k+1}}{m}$

  $v_{k+1} = v_k + ha_{k+1}$

  $x_{k+1} = x_k + hv_{k+1}$

- if $rm! = 0$

  $a_{k+1} = \frac{Fext_{k+1}}{m}$

$$v_{k+1} = cstVel * v_k + cstAcc * a_k \begin{cases} cstVel &= \dfrac{\dfrac{1}{h} - \dfrac{r_m}{2}}{\dfrac{1}{h} + \dfrac{r_m}{2}} \\ cstAcc &= \dfrac{1}{\dfrac{1}{h} + \dfrac{r_m}{2}} \end{cases}$$

$$x_{k+1} = x_k + hv_{k+1}$$