The Art of Computational Science

The Kali Code

vol. 16

Individual Time Steps:
Individual Integration Schemes

Piet Hut and Jun Makino

September 13, 2007

Contents

Preface

0.1 xxx

We thank xxx, xxx, and xxx for their comments on the manuscript.
Piet Hut and Jun Makino

Chapter 1

XXX

1.1 xxx

In file `world2.rb` many other methods, beside Hermite: forward Euler and leapfrog and most interestingly, a fourth-order multi-step scheme.

Chapter 2

XXX

2.1 xxx

```
  def ms4_setup(time)
    @time = @next_time = time
    @acc = @pos*0
    @jerk = @pos*0
    @snap = @pos*0
    @crackle = @pos*0
    @pop = @pos*0
    @nsteps = 0
    @minstep = VERY_LARGE_NUMBER
    @maxstep = 0
  end
```

```
  def ms4_startup_first_round(acc, jerk, timescale, dt_param, dt_max)
    @acc = acc
    @jerk = jerk
    dt = timescale * dt_param
    dt = dt_max if dt > dt_max
    @next_time = @time + dt
  end
```

```
  def ms4_startup_second_round(snap, crackle)
    @snap = snap
    @crackle = crackle
  end
```

```
def ms4_correct(wa, acc, timescale, dt_param, dt_max)
  d = [[acc]]
  t = [@time]
  dt = [0]
  wa.each do |wp|
    d[0].push(wp.acc)
    d.push([])
    t.push(wp.time)
    dt.push(@time - wp.time)
  end
  for k in (1..wa.size)
    for i in (0..wa.size-k)
      d[k][i] = (d[k-1][i] - d[k-1][i+1])/(t[i]-t[i+k])
    end
  end
  @pop = 24*d[4][0]
  @crackle = 6*(d[3][0] + d[4][0]*(dt[1] + dt[2] + dt[3]))
  @snap = 2*(d[2][0] + d[3][0]*(dt[1]+dt[2]) +
                       d[4][0]*(dt[1]*dt[2] + dt[2]*dt[3] + dt[3]*dt[1]))
  @jerk = d[1][0] + dt[1]*( d[2][0] + dt[2]*( d[3][0] + dt[3]*d[4][0] ) )
  @acc = acc
  dt = wa[0].time - @time
  @vel = wa[0].vel - ( @acc*dt + (1/2.0)*@jerk*dt**2 + (1/6.0)*@snap*dt**3 +
                       (1/24.0)*@crackle*dt**4 + (1/120.0)*@pop*dt**5 )
  @pos = wa[0].pos - ( @vel*dt + (1/2.0)*@acc*dt**2 + (1/6.0)*@jerk*dt**3 +
                       (1/24.0)*@snap*dt**4 + (1/120.0)*@crackle*dt**5 +
                       (1/720.0)*@pop*dt**6 )
  dt = timescale * dt_param
  dt = dt_max if dt > dt_max
  @next_time = @time + dt
  admin(wa[0].time)
  self
end
```

```
def ms4_full_extrapolate(t)
  if t > @next_time
    raise "t = " + t.to_s + " > @next_time = " + @next_time.to_s + "\n"
  end
  wp = WorldPoint.new
  wp.minstep = @minstep
```

```
    wp.maxstep = @maxstep
    wp.nsteps = @nsteps
    wp.mass = @mass
    wp.time = t
    dt = t - @time
    wp.pos = @pos + @vel*dt + (1/2.0)*@acc*dt**2 + (1/6.0)*@jerk*dt**3 +
             (1/24.0)*@snap*dt**4 + (1/120.0)*@crackle*dt**5
    wp.vel = @vel + @acc*dt + (1/2.0)*@jerk*dt**2 + (1/6.0)*@snap*dt**3 +
             (1/24.0)*@crackle*dt**4
    wp.acc = @acc + @jerk*dt + (1/2.0)*@snap*dt**2 + (1/6.0)*@crackle*dt**3
    wp.jerk = @jerk + @snap*dt + (1/2.0)*@crackle*dt**2
    wp.snap = @snap + @crackle*dt
    wp.crackle = @crackle
    wp
  end
```

```
  def ms4_extrapolate(t)
    if t > @next_time
      raise "t = " + t.to_s + " > @next_time = " + @next_time.to_s + "\n"
    end
    wp = WorldPoint.new
    wp.minstep = @minstep
    wp.maxstep = @maxstep
    wp.nsteps = @nsteps
    wp.mass = @mass
    wp.time = t
    dt = t - @time
    wp.pos = @pos + @vel*dt + (1/2.0)*@acc*dt**2 + (1/6.0)*@jerk*dt**3
    wp.vel = @vel + @acc*dt + (1/2.0)*@jerk*dt**2
    wp.acc = @acc + @jerk*dt
    wp.jerk = @jerk
    wp
  end
```

```
  def ms4_interpolate(other, t)
    wp = WorldPoint.new
    wp.minstep = @minstep
    wp.maxstep = @maxstep
    wp.nsteps = @nsteps
    wp.mass = @mass
    wp.time = t
```

```
  dt = other.time - @time
  dt = t - @time
  wp.pos = @pos + @vel*dt + (1/2.0)*@acc*dt**2 + (1/6.0)*@jerk*dt**3 +
           (1/24.0)*@snap*dt**4 + (1/120.0)*@crackle*dt**5 +
           (1/720.0)*@pop*dt**6
  wp.vel = @vel + @acc*dt + (1/2.0)*@jerk*dt**2 + (1/6.0)*@snap*dt**3 +
           (1/24.0)*@crackle*dt**4 + (1/120.0)*@pop*dt**5
  wp
end
```

```
def ms4_setup(time)
  @worldpoint[0].ms4_setup(time)
end
```

```
def ms4_number_of_steps
  4
end
```

```
def ms4_startup_done?(era, dt_max)
  wp = @worldpoint[0]
  if not defined? @startup_cycle
    @startup_cycle = 1
    acc, jerk = era.acc_and_jerk(self, wp)
    timescale = era.timescale(self, wp)
    wp.ms4_startup_first_round(acc, jerk, timescale, @dt_param, dt_max)
    return false
  elsif @startup_cycle == 1
    @startup_cycle += 1
    snap, crackle = era.snap_and_crackle(self, wp)
    wp.ms4_startup_second_round(snap, crackle)
    time = wp.time
    dt = wp.next_time - time
    for i in (1..3)
      @worldpoint.unshift(wp.ms4_full_extrapolate(time - i*dt))
    end
    return true
  else
    raise "@startup_cycle = #{@startup_cycle} != 1"
  end
end
```

```
  def take_ms4_step(era, dt_max)
    new_point = ms4_predict
    acc = era.acc(self, new_point)
    timescale = era.timescale(self, new_point)
    k = @worldpoint.size
    wp_array = @worldpoint[k-4...k].reverse
    new_point.ms4_correct(wp_array, acc, timescale, @dt_param, dt_max)
  end
```

```
  def ms4_predict
    wp = @worldpoint.last
    wp.ms4_extrapolate(wp.next_time)
  end
```

nil nil nil nil nil nil nil nil

Chapter 3

Literature References

[to be provided]