Verlet Algorithm Applied to Particle in Central Gravitational Potential And Newton-Rapheson Method Used to Solve Kepler's Equation

Adam Reyes

October 15, 2013

1 THE VERLET ALGORITHM

Typically the Verlet Algorithm computes the value of some dependent variable x whose second derivative is given as some function of itself and some independent variable t:

$$\ddot{x} = F(x(t), t). \tag{1.1}$$

The algorithm computes the next value x^{n+1} as follows, using the current value of x, x^n, and the previous, x^{n-1}:

$$x^{n+1} = 2x^n - x^{n-1} + h^2 F. \tag{1.2}$$

One can note that the first derivative, $\dot{x} \equiv v$, in the limit of small steps, h, can be written, at the half step between n and $n-1$ as

$$v^{n-1/2} = \frac{x^n - x^{n-1}}{h} \tag{1.3}$$

Substituting this into equation (1.2) and using a first order Taylor series expansion for $v^{n+1/2}$, we get

$$\begin{aligned} v^{n+1/2} &= v^{n-1/2} + hF \\ x^{n+1} &= x^n + h v^{n+1/2} \end{aligned} \tag{1.4}$$

It can be readily seen from equation (1.2) that this algorithm is symmetric in t.

1.1 Central Gravitational Potential

I will now apply the Verlet algorithm to the problem of a central gravitational potential where F is now given by

$$\vec{F}(x(t), y(t), t) = -\frac{1}{r^2}\hat{r} = -\frac{1}{(x^2+y^2)^{\frac{3}{2}}}(x, y) \tag{1.5}$$

Figure 1.1 shows the computed trajectories for a particle in F given the same initial conditions, over a long time interval. Both Start off doing roughly the same orbit, but after some time the RK4 trajectory begins to lose its elliptic symmetry and shoots out and becomes unbounded (over the time computed the RK4 trajectory shoots much farther out of the limits of the plot). Meanwhile the Verlet trajectory maintains a roughly elliptical orbit throughout the entire computation. This is reflected again in the bottom plot of the Energy of the two systems over time. While neither remains constant, the RK4 energy, over time, shoots out, while that of the Verlet trajectory oscillates roughly between the same values.

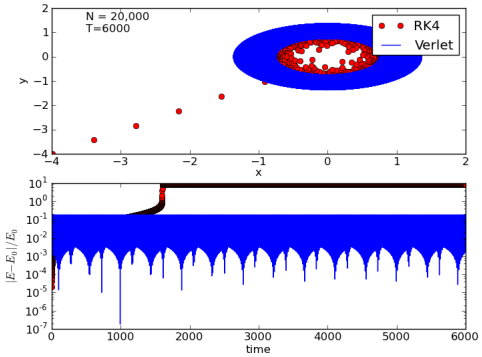


Figure 1.1: (top) Plot of the computed trajectory using RK4 and Verlet for 6,000 time units and 20,000 steps. (bottom) plot of the change in energy vs time.

2 Newton-Rapheson

The Newton-Rapheson method is an algorithm to find the roots, x, of some function $f(x)$. This is accomplished by beginning with some guess value of x and using the first derivative, $f'(x)$ at that guess to extrapolate the root of the function. This is then iteratively repeated using the extrapolated root until some tolerance condition is met.
To demonstrate this method I have computed the root of the function

$$f(x) = x^3 - cos(x) \tag{2.1}$$

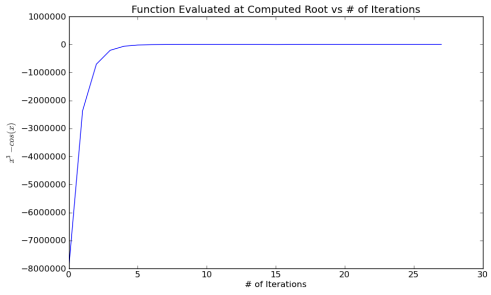


Figure 2.1: Plot of $f(x_0)$ against the number of iterations used to get to the root x_0, beginning with a guess of $x_0 = -200$.

Figure 2.1 shows how in only a few iterations Newton-Rapheson can quickly approach the true root. The routine gives the root, $x_0 = 0.865$ with an error of 2.109×10^{-8}.

3 Kepler's Equation

Kepler's Equation is given as

$$M = E - \epsilon sin(E) \tag{3.1}$$

where M is time dependent, and so the roots of this equation (E_0) can exactly parameterize an orbit given M at a particular time and some eccentricity, ϵ, of the orbit.

ϵ can be calculated directly from the energy and angular momentum(both constants of the system), which themselves are easily obtained from initial conditions.

To maintain consistency with my script for the Verlet Algorithm, as well as previous Runge-Kutta methods, I have opted to use initial conditions in cartesian coordinates. All of the conversion to polar coordinates, which the paramaterization of Kepler's Equation uses, are fairly trivial. My implementation of $\dot{\theta}$, the angular velocity, may not be particularly obvious so I dedicate a little section to it.

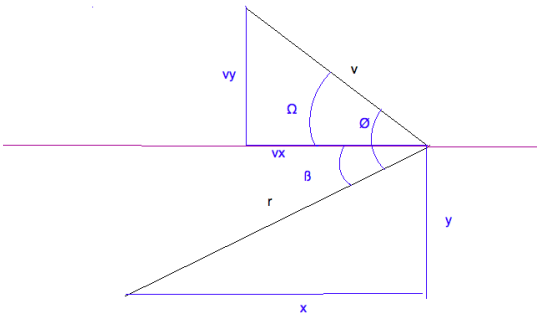


Figure 3.1: Diagram of $\vec{r}$ and $\vec{v}$ at some arbitrary point in time.

We begin with the definition of $\dot{\theta}$

$$\dot{\theta} = \frac{v}{r} sin\phi. \tag{3.2}$$

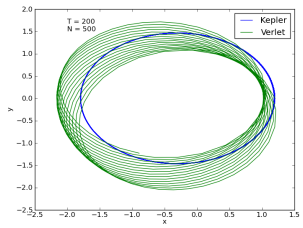
Where ϕ is the angle between $\vec{v}$ and $\vec{r}$. From figure 3.1 it is clear that $\phi = \Omega + \beta$, then it follows

$$\begin{aligned} sin\phi &= sin\Omega cos\beta = cos\Omega sin\beta \\ &= \frac{1}{vr}(v_y x + v_x y) \end{aligned} \tag{3.3}$$

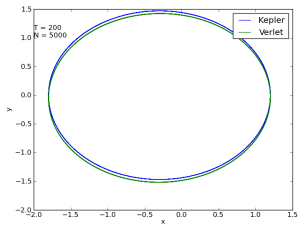
And it follows easily that the angular momentum, l is

$$l = r^2\dot{\theta} = v_y x + v_x y \tag{3.4}$$

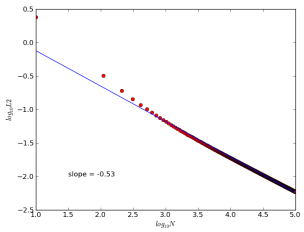
Figure 3.2 (a) shows an orbit computed from Kepler's equation compared to the trajectory calculated using the Verlet Algorithm. In this plot the orbit from the Verlet Method appears to be the right shape but offset from the trajectory from the Kepler equation. In addition after each period the orbit is being offset by some angle about a point. Figure 3.2 (b) shows that at higher resolution the Verlet trajectory converges to that calculated from Kepler's Equation. Since the Verlet Algorithm is a second order method, I would expect that a log-log plot of L-2 norm as a function of steps would be a line of slope 2. But as calculated and shown in figure 3.2 (c) I found a line with slope roughly 1/2, suggesting the method converging roughly as $\sqrt{N}$.



(a) Computed Trajectories using Verlet Method and from solving Kepler's equation for 200 time units in 500 steps.



(b) Computed Trajectories using Verlet Method and from solving Kepler's equation for 200 time units in 5,000 steps



(c) log-log plot of the L2 error against the number of steps along with a computed linear fit of slope -0.53.

Figure 3.2: Plots comparing Verlet computations to Newton-Rapheson solutions of Kepler's Equation

4 Source Code

4.1 Verlet Algorithm

```
#include <stdio.h>
#include <math.h>
double a2(double x){
  double temp = pow(x,2);
  return( temp );
}
//integration functions
double dv_idt( double t, double i, double j, double v){
  double denom = pow( a2(i) + a2(j), 1.5);
  return( -i/denom );
}
double didt( double t, double i, double v_i){
  return(v_i);
}
//end integration functions
//solutions
double rk4final( double x, double h1, double h2, double h3, double h4){
  return(x + (h1 + 2*h2 + 2*h3 + h4)/6);
}
void rk4( double t, double dt, double x, double y, double v_x, double v_y, double* x_n,
  double hvx1 = dv_idt(t, x, y, v_x)*dt;
  double hvy1 = dv_idt(t, y, x, v_y)*dt;
  double hx1 = didt(t, x, v_x)*dt;
  double hy1 = didt(t, y, v_y)*dt;
```

```
  double hvx2 = dv_idt(t + 0.5*dt, x + 0.5*hx1, y + 0.5*hy1, v_x + 0.5*hvx1)*dt;
  double hvy2 = dv_idt(t + 0.5*dt, y + 0.5*hy1, x + 0.5*hx1, v_y+0.5*hvy1)*dt;
  double hx2 = didt(t+0.5*dt, x+0.5*hx1, v_x+0.5*hvx1)*dt;
  double hy2 = didt(t+0.5*dt, y+0.5*hy1, v_y+0.5*hvy1)*dt;

  double hvx3 = dv_idt(t + 0.5*dt, x + 0.5*hx2, y + 0.5*hy2, v_x + 0.5*hvx2)*dt;
  double hvy3 = dv_idt(t + 0.5*dt, y + 0.5*hy2, x + 0.5*hx2, v_y + 0.5*hvy2)*dt;
  double hx3 = didt(t+0.5*dt, x + 0.5*hx2, v_x + 0.5*hvx2)*dt;
  double hy3 = didt(t + 0.5*dt, y + 0.5*hy2, v_y + 0.5*hvy2)*dt;

  double hvx4 = dv_idt(t + dt, x + hx3, y + hy3, v_x + hvx3)*dt;
  double hvy4 = dv_idt(t + dt, y + hy3, x + hx3, v_y + hvy3)*dt;
  double hx4 = didt(t + dt, x + hx3, v_x + hvx3)*dt;
  double hy4 = didt(t+dt, y + hy3, v_y + hvy3)*dt;

  *v_nx = rk4final(v_x, hvx1, hvx2, hvx3, hvx4);
  *v_ny = rk4final(v_y, hvy1, hvy2, hvy3, hvy4);
  *x_n = rk4final(x, hx1, hx2, hx3, hx4);
  *y_n = rk4final(y, hy1, hy2, hy3, hy4);
}

void verlet( double t, double dt, double x, double y, double v_x, double v_y, double* x_
  /////solves for x(t) and v(t+dt/2) using x(t-dt) and v(t-dt/2)//////
  *v_nx = v_x + dt*dv_idt( t, x, y, v_x );
  *v_ny = v_y + dt*dv_idt( t, y, x, v_y );

  *x_n = x + dt*(*v_nx);
  *y_n = y + dt*(*v_ny);
}

double Energy( double x, double y, double v_x, double v_y){
  double E = 0.5*(a2(v_x) + a2(v_y)) - 1/(sqrt( a2(x) + a2(y) ));
  return(E);
}

////////////////Begin Code//////////////////

int main(void){
  ////////////Set Initial Conditions//////
  double T = 200.0;
  double t, xv, yv, v_xv, v_yv, x4, y4, v_x4, v_y4, v_nx, v_ny, x_n, y_n, x_old, y_old,
  t = 0;
  int N = 500; ////////////////////
```

```
int i;
double dt = T/(double)N;
FILE *fid;
fid = fopen("assign2.dat", "w");

x4 = 1.2;
y4 = 0.0;
xv = x4;
yv = y4;
v_x4 = -0.0;
v_y4 = 1.0;
//////////RK4 to get next value/////////////
rk4(t, dt, x4, y4, v_x4, v_y4, &x_n, &y_n, &v_nx, &v_ny);
//////////Average v's to get v at the half step//////
v_xv = 0.5*(v_x4 + v_nx);
v_yv = 0.5*(v_y4 + v_ny);
E0 = Energy(x4, y4, v_x4, v_y4);
E4 = fabs( (Energy(x4, y4, v_x4, v_y4) - E0)/E0);
Ev = E4;
printf("%f %f\n", E4, Ev);

///////loop over everything//////////
for(i=0; i<N; ++i){
  fprintf(fid, "%e %e %e %e %e %e %e\n", t, x4, y4, xv, yv, E4, Ev );
  /////////Run RK4/////////////
  /*
  rk4(t, dt, x4, y4, v_x4, v_y4, &x_n, &y_n, &v_nx, &v_ny);
  x4 = x_n;
  y4 = y_n;
  v_x4 = v_nx;
  v_y4 = v_ny;
  E4 = fabs( (Energy(x4, y4, v_x4, v_y4) - E0)/E0);
  */
  ////////Do Verlet//////////
  verlet( t, dt, xv, yv, v_xv, v_yv, &x_n, &y_n, &v_nx, &v_ny);
  xv = x_n;
  yv = y_n;
  v_x = 0.5*(v_xv + v_nx);
  v_y = 0.5*(v_yv + v_ny);
  v_xv = v_nx;
  v_yv = v_ny;
  Ev = fabs( (Energy(xv, yv, v_xv, v_yv) - E0)/E0 );
  t += dt;
}
```

```
}
```

4.2 NEWTON-RAPHESON

```
#include <stdio.h>
#include <math.h>

double f(double x){
  return(pow(x,3) - cos(x));
}
double dfdx( double x){
  return(3.0*pow(x,2) + sin(x));
}
double del( double x){
  return(-f(x)/dfdx(x));
}
int main(void){
  double tol, x, x_n;
  tol = 10e-10;
  x = -200.0;
  FILE *fid;
  fid = fopen("newraph.dat", "w");
  int count;
  count = 0;

  while(count < 50){
    fprintf(fid, "%d %e %e\n", count, x, f(x));
    printf( "%d %e %e\n", count, x, f(x));
    x_n = x + del(x);
    x = x_n;
    if (fabs(f(x)) < tol){
      {break;}
    }
    count += 1;
  }
}
```

4.3 KEPLER'S EQUATION

```
#include <stdio.h>
#include <math.h>
```

```
double a2(double x){
  return(pow(x, 2));
}
double h(double M, double E, double e){
  return(M - E + e*sin(E));
}
double dfdE( double E, double e){
  return(-1.0 + e*cos(E));
}
double del( double M, double E, double e){
  return(-h(M, E, e)/dfdE(E, e));
}
double Energy( double x, double y, double v_x, double v_y){
  double E = 0.5*(a2(v_x) + a2(v_y)) - 1/(sqrt( a2(x) + a2(y) ));
  return(E);
}
double v(double v_x, double v_y){
  return(sqrt(pow(v_x, 2) + pow(v_y, 2)));
}
int main(void){
  FILE *fid;
  fid = fopen("kepler.dat", "w");

  double x, y, a, b, E, E_n, e, f, M, l, phi, v_r, v_p, v_x, v_y, U, T, dt, t, tol;
  int N = 500;
  tol = 10e-11;
  T = 200.0;
  dt = T/(double)N;
  t = 0.0;
  x = 1.2;
  y = 0;
  v_x = -0.0;
  v_y = 1.0;
  double r = sqrt(pow(x,2) + pow(y,2));

  U = Energy(x, y, v_x, v_y);
  l = (v_y*x + v_x*y);

  a = 0.5*1/(fabs(U));
  b = l/(sqrt(2.0*fabs(U)));
  f = sqrt(1.0 - 2.0*pow(l,2)*fabs(U))/(2.0*fabs(U));
  e = f/a;
  printf("%f %f %f %f %f %f\n", a, b, f, e, l, U);
  int i;
```

```
  int count;
  for(i = 0; i<N; i++){
    fprintf(fid, "%e %e %e\n", t, x, y);
    //do newton rapheson//
    count = 0;
    while(count < 30){
      M = t*l/(a*b);
      E_n = E + del(M, E, e);
      E = E_n;
      if ( h( M, E, e) < tol){
        {break;}
      }
      count += 1;
    }
    //////calculate x and y////////
    x = a*cos(E) - f;
    y = b*sin(E);
    t += dt;
  }
}
```

4.4 L-2 Norm

```
#include <stdio.h>
#include <math.h>
double a2(double x){
  return(pow(x, 2));
}
double h(double M, double E, double e){
  return(M - E + e*sin(E));
}
double dfdE( double E, double e){
  return(-1.0 + e*cos(E));
}
double del( double M, double E, double e){
  return(-h(M, E, e)/dfdE(E, e));
}
double Energy( double x, double y, double v_x, double v_y){
  double E = 0.5*(a2(v_x) + a2(v_y)) - 1/(sqrt( a2(x) + a2(y) ));
  return(E);
}
double v(double v_x, double v_y){
  return(sqrt(pow(v_x, 2) + pow(v_y, 2)));
```

```
}
double dv_idt( double t, double i, double j, double v){
  double denom = pow( a2(i) + a2(j), 1.5);
  return( -i/denom );
}
void verlet( double t, double dt, double x, double y, double v_x, double v_y, double* x_
  /////solves for x(t) and v(t+dt/2) using x(t-dt) and v(t-dt/2)//////
  *v_nx = v_x + dt*dv_idt( t, x, y, v_x );
  *v_ny = v_y + dt*dv_idt( t, y, x, v_y );

  *x_n = x + dt*(*v_nx);
  *y_n = y + dt*(*v_ny);
}

double dr2( double dt, double x, double y, double x_t, double y_t){
  double dx = x - x_t;
  double dy = y - y_t;
  return( dt*(pow(dx, 2) + pow(dy, 2)));
}
int main(void){
  printf("%f\n", dr2(1.0, 0.0, 1, 0.5, 2.0));
  FILE *fid, *fl2;
  fl2 = fopen("l2.dat", "w");
  fid = fopen("kepler.dat", "w");

  double x, y, a, b, E, E_n, e, f, M, l, phi, v_r, v_p, v_x, v_y, U, T, dt, t, tol, sum;
  double xv, yv, v_xv, v_yv, x_n, y_n, v_nx, v_ny;
  int N = 10;
  tol = 10e-11;
  T = 6.0;
  dt = T/(double)N;
  t = 0.0;
  x = 1.2;
  y = 0.0;
  v_x = -0.0;
  v_y = 1.0;
  xv = x;
  yv = y;
  v_xv = v_x;
  v_yv = v_y;
  double r = sqrt(pow(x,2) + pow(y,2));

  U = Energy(x, y, v_x, v_y);
  l = (v_y*x + v_x*y);
```

```
a = 0.5*l/(fabs(U));
b = l/(sqrt(2.0*fabs(U)));
f = sqrt(1.0 - 2.0*pow(l,2)*fabs(U))/(2.0*fabs(U));
e = f/a;

int i, j;
for (j = 0; j< 1000; j++){
  int count;

  sum = 0;
  for(i = 0; i<N; i++){
    if (j == 0){
      fprintf(fid, "%e %e %e\n", t, x, y);
    }
    //do newton rapheson//
    count = 0;
    while(count < 30){
      M = t*l/(a*b);
      E_n = E + del(M, E, e);
      E = E_n;
      if ( h( M, E, e) < tol){
        {break;}
      }
      count += 1;
    }
    //////calculate x and y////////
    x = a*cos(E) - f;
    y = b*sin(E);
    ///////////do verlet///////////
    verlet( t, dt, xv, yv, v_xv, v_yv, &x_n, &y_n, &v_nx, &v_ny);
    xv = x_n;
    yv = y_n;
    v_xv = v_nx;
    v_yv = v_ny;

    /////Integrate L2 Error////////
    sum += dr2(dt, xv, yv, x, y);
    t += dt;
  }
  fprintf(fl2, "%d %e\n", N, sqrt(fabs(sum)));
  printf("%d %e\n", N, fabs(sum) ) ;
  N += 100;
  dt = T/(double)N;
```

```
    t = 0;
    x = 1.2;
    y = 0.0;
    v_x = -0.0;
    v_y = 1.0;
    xv = x;
    yv = y;
    v_xv = v_x;
    v_yv = v_y;
  }
}
```