Project 2
Getting Started…

PHYS2G03

McMaster University

Steps

1. Submit a paragraph or two on avenue under "project" (Due Oct 22nd but do it sooner!)
2. Get written feedback from us on how to make a successful program and project

Each student will have an assigned TA and individual feedback and help

A scientific goal

- Most proposals were ok to start

- They contain a basic idea that is good for testing and making sure things work

  e.g. 1 predator + 1 prey population with some equations for how they change

- Then they propose some more complex extensions and/or cases to try,

  e.g. adding a 3rd variable: like a competing predator or a food source for the prey

A scientific goal

- Some proposals are not models – they just suggest plugging in numbers to an expression or just plotting a formula.

  That is too simple

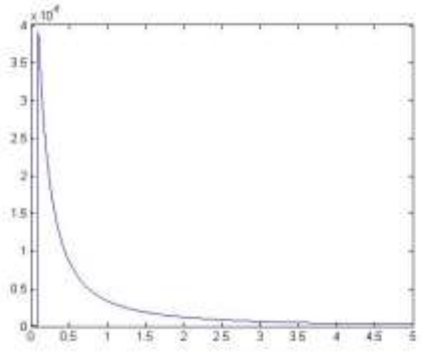
  Plotting a fixed answer is not a model. You need a **numerical experiment** – a system that evolves and responds differently in different cases. It must be possible to explore your model by trying things with it.

Failed Project Idea

A formula

Neuron response: E = V/L exp(-t/t0)

- V is the voltage
- I made a plot – E vs. time t
- If I double V – my E is twice as big
- This is not a model – every response to change is obvious and predictable
- The program is trivial
- A model has to be sufficiently complex so that behaviour is not a trivial response to changes
- Neuron response dE/dt = … that is getting interesting



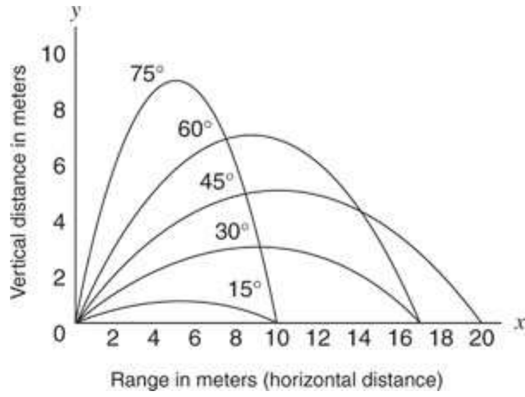
A scientific goal

- Some models were also simple – only suitable as a basic starting point for a project, e.g. 2-bodies in orbit
- Typically we suggest moving on to a more complicated model
- *Alternately* – you can try 1000’s of cases and explore probabilities and outcomes.

  That makes it ok – suitable for experimenting

  This approach to exploring probabilities is often called **Monte Carlo**

Monte Carlo
Example: Projectiles



- Projectile motion is simple
- Calculating many different trajectories is not
- Find out which trajectories are *interesting*
- Interesting means what scores (trajectory of a ball in game), what most often hits a target (given a limit on the accuracy)
- What makes the trajectory interesting is related to the goal of the project

Monte Carlo
Example: Simple Orbits

- Calculating an orbit around one object is too simple
- Calculating many orbits is not
- Find out which orbits are *interesting*
- Interesting could mean dangerous (Earth crossing asteroids), productive (building planets), strange (horseshoe orbits), …
- What makes the orbits interesting is related to the scientific goal of the project

Classes of Problems

- Solving systems of ODE’s
- Monte Carlo
- Optimization
- AI: Neural nets
- Solving PDE’s
- Particles (Coupled ODE’s)

Solvers

- We’ll discuss ways to solve these in class
- We can go over them several times – just ask when you are stuck

- Note: Some projects could choose between very different approaches but most are pretty clear

Getting a good grade

- A working program

Don’t write the big program to solve it all at once – get simple stages to work (and save working versions) and build up to a solution for the full problem

You will be expected to demo at least part of your program by the end of term – in class: Save a working version!

Getting a good grade

- Test your model

  If you don’t have ideas on how to do that we’ll help you

  It isn’t enough to just write and run a program – you need to show it works correctly

Getting a good grade

- Describe a science goal

  Ideally your program will demonstrate something scientifically interesting: how real populations grow and fail, how to identify interesting orbits, the outcome of a process assuming an initial state …

  Even if the complex version of the program doesn't work, you can get a lot of credit by discussing what you were aiming to do

Project – Goals and Write-up

Your proposal is a sketch of the introduction and method sections of the final write-up

The basic program suggestion is what we recommend to ensure a reasonable grade. If you want to ensure an outstanding grade we have suggested a more complex program and a question your program can be used to address

Integrating ODE's

Basic idea:

$dy/dx = f(x)$

Taylor series:

$y(x+\Delta x) = y(x) + dy/dx\ \Delta x + \frac{1}{2}\ d^2y/dx^2\ \Delta x^2 + \ldots$

Simplest scheme to find $y(x+\Delta x)$ given $y(x)$:

Forward Euler

$y(x+\Delta x) = y(x) + f(x) * \Delta x + O(\Delta x^2)$

Forward Euler:
Really inaccurate Errors O(h)

```
// Forward Euler
// step size delta x = h
x = x0;
y = y0;
// Go from y(x0) to y(x0+h*100)
for (i=0;i<100;i++) {
    y = y + dydxfunction(x,y)*h;
    x = x + h;
}
```

Forward Euler

$dy/dx = y^2$

Starting at $x=x1$ with $y=y1$

Go to $x=x2$ in N steps

$\Delta x = (x2-x1)/N$

$y(0) = y1$

$y(1) = y(0) + y(0)*y(0)* \Delta x$

$y(2) = y(1) + y(1)*y(1)* \Delta x$

$y(3) = y(2) + y(2)*y(2)* \Delta x$

…

$y(N) = y(N-1) + y(N-1)*y(N-1)* \Delta x$

$y(x+\Delta x) = y(x) + dy/dx\, \Delta x + ½\, d^2y/dx^2\, \Delta x^2$

Total Error: add up N individual errors: approximately

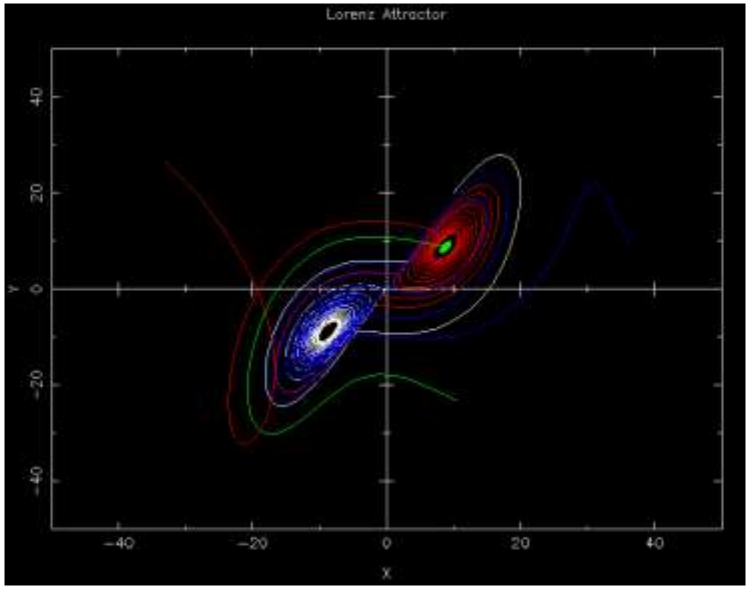
$<½\, d^2y/dx^2> \Delta x^2 N \sim <y^3> \Delta x\ (x2-x1)$

Forward Euler method has errors proportional to Δx

Predictor Corrector: Better, Errors O(h^2)

```
// Predictor Corrector
// step size delta x = h
x = x0;
y = y0;
for (i=0;i<100;i++) {
    ypred = y + dydxfunction(x,y)*h/2;
    y = y + dydxfunction(x+h/2,ypred)*h;
    x = x + h;
}
```

Detailed example:
Climate and chaos



Simplified model:

Lorenz Attractor

Equations:

$$\frac{dX}{dt} = a\,(Y-X)$$
$$\frac{dY}{dt} = X\,(b-Z) - Y$$
$$\frac{dZ}{dt} = X\,Y - c\,Z$$

Parameters: e.g. $a = 10$, $b = 28$, $c = 8/3$

Solved with predictor corrector…

```
// Integrate for nStep steps
for (iStep = 0; iStep < nStep;iStep++) {
  // Save point for plotting
  xp[iStep] = x;
  yp[iStep] = y;
  zp[iStep] = z;

  xpred = x + 0.5*dt*a*(y-x);
  ypred = y + 0.5*dt*(x*(b-z)-y);
  zpred = z + 0.5*dt*(x*y - c*z);

  x = x + dt*a*(ypred-xpred);
  y = y + dt*(xpred*(b-zpred)-ypred);
  z = z + dt*(xpred*ypred - c*zpred);
}
xp[nStep] = x;
yp[nStep] = y;
zp[nStep] = z;
```

see /home/2G03/lorenz/lorenz.cpp

Lorenz Attractor Equations:

$$dX/dt = a\,(Y-X)$$

$$dY/dt = X\,(b-Z) - Y$$

$$dZ/dt = X\,Y - c\,Z$$

```
// Integrate fo
for (iStep = 0;
   // Save point
   xp[iStep] = x
   yp[iStep] = y
   zp[iStep] = z

   xpred = x + 0.5*dt*a*(y-x);
   ypred = y + 0.5*dt*(x*(b-z)-y);
   zpred = z + 0.5*dt*(x*y - c*z);

   x = x + dt*a*(ypred-xpred);
   y = y + dt*(xpred*(b-zpred)-ypred);
   z = z + dt*(xpred*ypred - c*zpred);
}
xp[nStep] = x;
yp[nStep] = y;
zp[nStep] = z;
```

see /home/2G03/lorenz/lorenz.cpp

```
// Integrate for nStep steps
for (iStep = 0; iStep < nStep;iStep++) {
  // Save point for plotting
  xp[iStep] = x;
  yp[iStep] = y;
  zp[iStep] = z;
  // Do some substeps without saving for plots
  for (iSubstep = 0; iSubstep < nSubstep; iSubstep++) {
        xpred = x + 0.5*dt*a*(y-x);
        ypred = y + 0.5*dt*(x*(b-z)-y);
        zpred = z + 0.5*dt*(x*y - c*z);

        x = x + dt*a*(ypred-xpred);
        y = y + dt*(xpred*(b-zpred)-ypred);
        z = z + dt*(xpred*ypred - c*zpred);
  }
}
xp[nStep] = x;
yp[nStep] = y;
zp[nStep] = z;
```

see /home/2G03/lorenz/lorenz.cpp

Trick to reduce number of plot points

Take steps without saving the value

```
// Integrate for nStep
for (iStep = 0; iStep
   // Save point for pl
   xp[iStep] = x;
   yp[iStep] = y;
   zp[iStep] = z;
   // Do some substeps without saving for plots
   for (iSubstep = 0; iSubstep < nSubstep; iSubstep++) {
         xpred = x + 0.5*dt*a*(y-x);
         ypred = y + 0.5*dt*(x*(b-z)-y);
         zpred = z + 0.5*dt*(x*y - c*z);

         x = x + dt*a*(ypred-xpred);
         y = y + dt*(xpred*(b-zpred)-ypred);
         z = z + dt*(xpred*ypred - c*zpred);
   }
}
xp[nStep] = x;
yp[nStep] = y;
zp[nStep] = z;
```

see /home/2G03/lorenz/lorenz.cpp

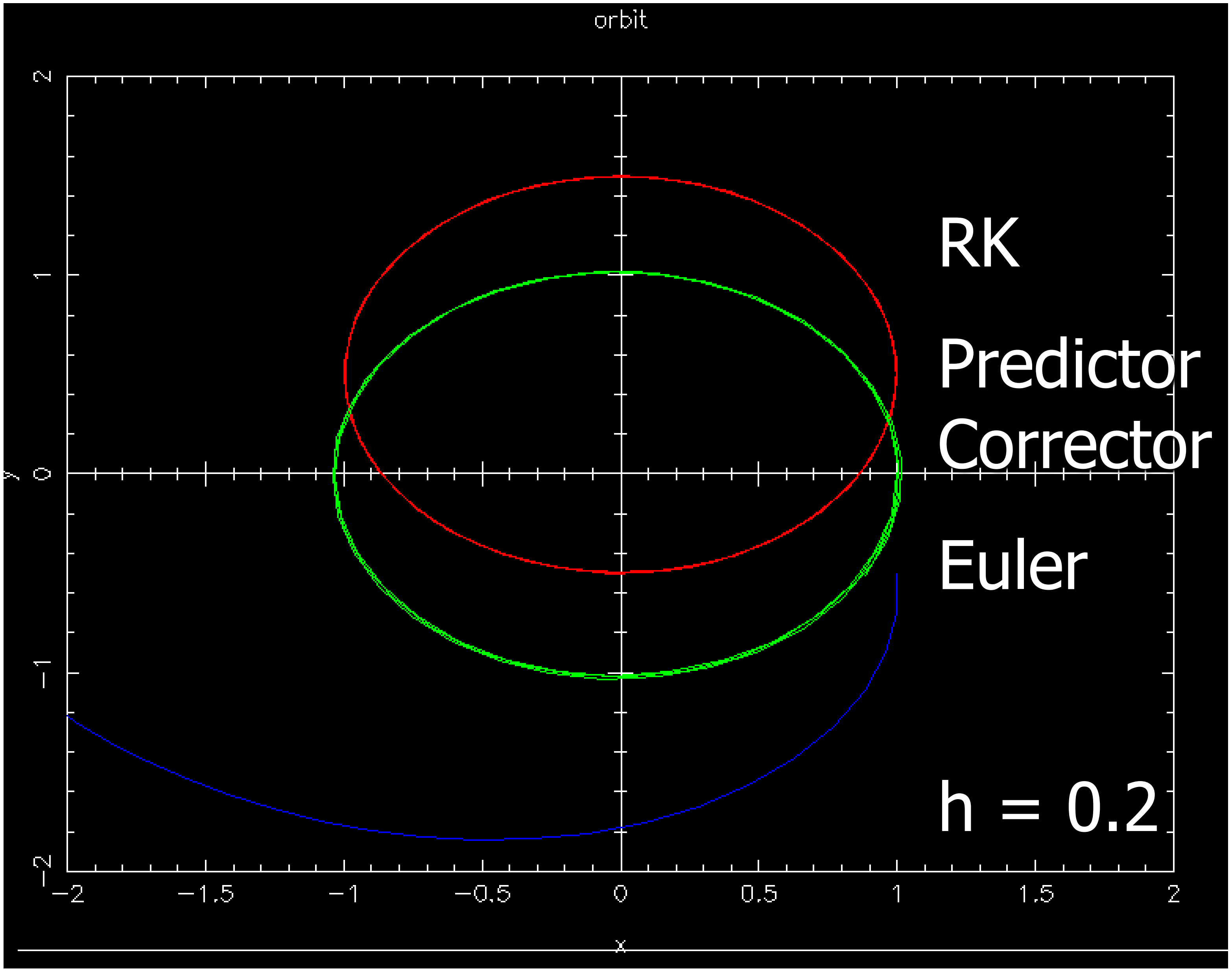
Most accurate: Runge-Kutta: Small Errors O(h^4)

Replace:

```
y = y + dydxfunction(x) * h;
```

With:

```
k1=h*dydx(x,y1);
k2=h*dydx(x+h/2,y+k1/2);
k3=h*dydx(x+h/2,y+k2/2);
k4=h*dydx(x+h,y+k3);
y = y  + (1.0/6.0)*(k1 + 2*k2 + 2*k3 + k4);
```


orbit
RK
Predictor
Corrector
Euler
h = 0.2
x
y
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2

Take care: step size, h

What is the best value for h?

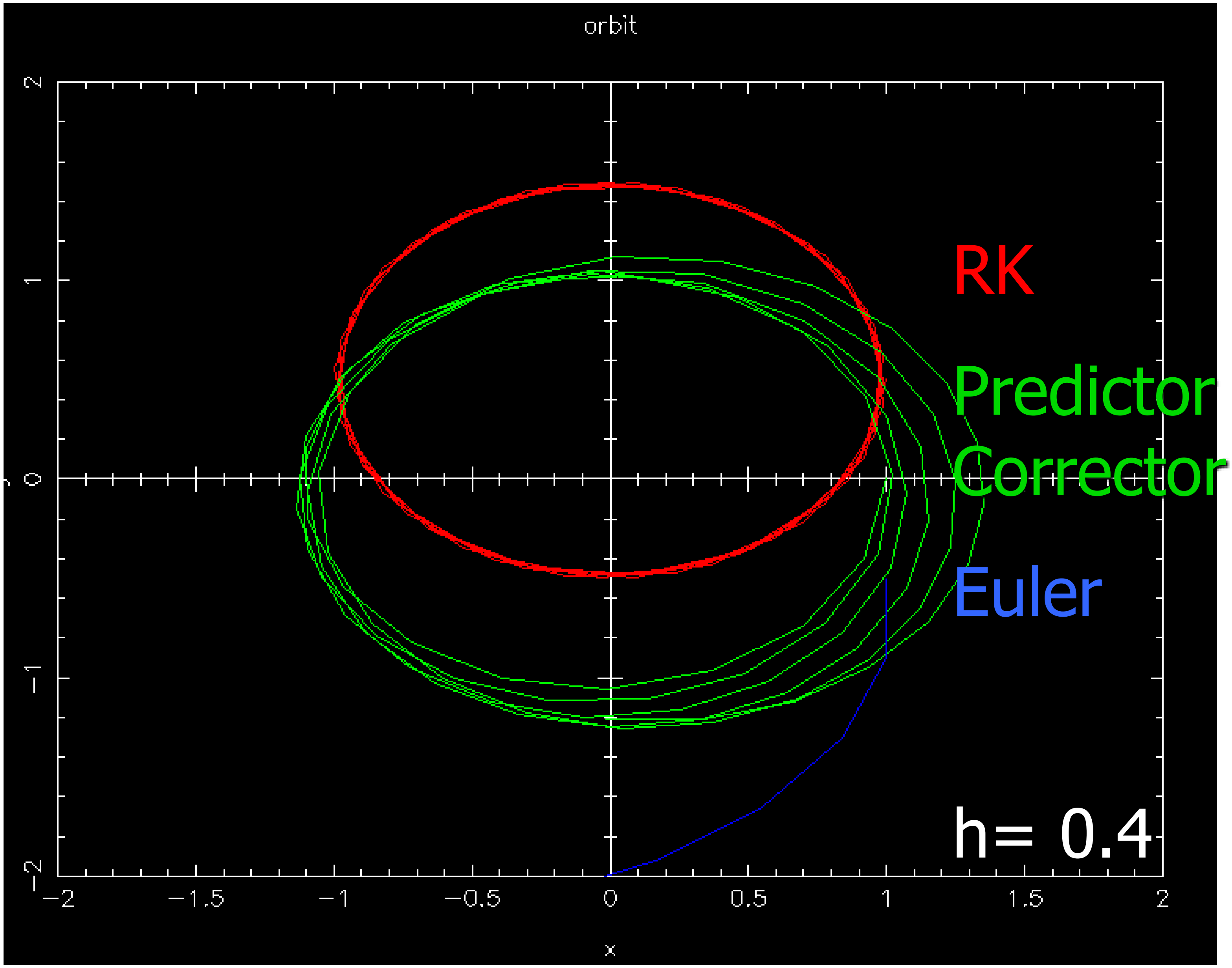
To do this properly you should do a course in numerical analysis or modeling

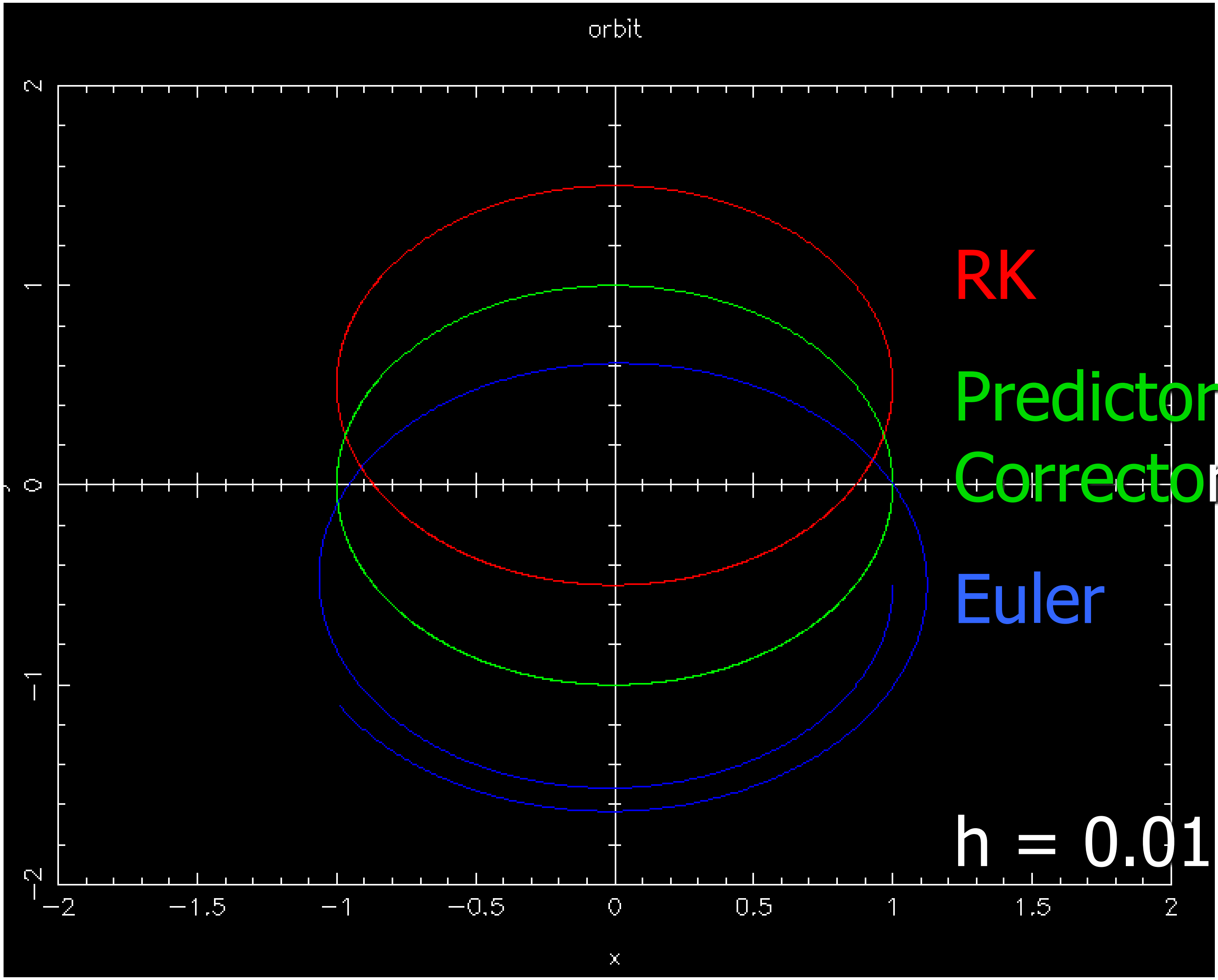
For now – if the results are strange your step is probably too big: reduce h !

In general once your result doesn't change much if you halve the step then you have a good step size

Note: Reducing h means it takes more steps to go the same distance. In general use h=L/nStep

Where L is the fixed distance you want. Increase nStep for better accuracy


orbit
RK
Predictor
Corrector
Euler
h= 0.4
x
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
−2
−1
0
1
2


orbit
2
1
0
−1
−2
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
x
RK
Predictor Corrector
Euler
h = 0.01

Predictor Corrector: two equations

$$\text{Find } x(t), y(t) \text{ given } x(t_0), y(t_0) \text{ and } \frac{dx}{dt} = f(x,y,t), \quad \frac{dy}{dt} = g(x,y,t)$$

```
// All the predictors
xpred = x + dt/2*dxdt(t,x,y);
ypred = y + dt/2*dydt(t,x,y);
// All the correctors
x = x  + dt*dxdt(t+dt/2,xpred,ypred);
y = y  + dt*dydt(t+dt/2,xpred,ypred);
```

Interleave x and y

Predictor Corrector: two equations

Find $x(t), y(t)$ given $x(t_0), y(t_0)$ and $\dfrac{dx}{dt} = f(x,y,t)\,,\quad \dfrac{dy}{dt} = g(x,y,t)$

```
// All the predictors
// xpred =  x(t+dt/2)  ypred = y(t+dt/2)
 xpred = x + dt/2*dxdt(t,x,y);
 ypred = y + dt/2*dydt(t,x,y);
 // All the correctors
 // x(t) to x(t+dt)  y(t) to y(t+dt)
 x = x  + dt*dxdt(t+dt/2,xpred,ypred);
 y = y  + dt*dydt(t+dt/2,xpred,ypred);
```

Interleave x and y for $O(h^2)$

Incorrect Predictor Corrector: two equations

Find $x(t), y(t)$ given $x(t_0), y(t_0)$ and $\frac{dx}{dt} = f(x, y, t)$, $\frac{dy}{dt} = g(x, y, t)$

```
// x(t) to x(t+dt)
 xpred = x + dt/2*dxdt(t,x,y);
 x = x  + dt*dxdt(t+dt/2,xpred,ypred);
//  y(t) to y(t+dt)
ypred = y + dt/2*dydt(t,x,y);
 y = y  + dt*dydt(t+dt/2,xpred,ypred);
```

WRONG!

This is not $O(h^2)$

This is no better than Euler!

Runge-Kutta: two equations

Find $x(t), y(t)$ *given starting values* $x0, y0$ *and* $\frac{dx}{dt} = f(x,y,t)$, $\frac{dy}{dt} = g(x,y,t)$

```
kx1=dt*dxdt(t,x,y);
ky1=dt*dydt(t,x,y);
kx2=dt*dxdt(t+dt/2,x+kx1/2,y+ky1/2);
ky2=dt*dydt(t+dt/2,x+kx1/2,y+ky1/2);
kx3=dt*dxdt(t+dt/2,x+kx2/2,y+ky2/2);
ky3=dt*dydt(t+dt/2,x+kx2/2,y+ky2/2);
kx4=dt*dxdt(t+dt,x+kx3,y+ky3);
ky4=dt*dydt(t+dt,x+kx3,y+ky3);
x = x  + (1.0/6.0)*(kx1 + 2*kx2 + 2*kx3 + kx4);
y = y  + (1.0/6.0)*(ky1 + 2*ky2 + 2*ky3 + ky4);
```

Interleave x and y

For RK too

Runge-Kutta: two equations

Find a$(r), b(r)$ given a$(r_0), b(r_0)$ and $\frac{da}{dr} = f(a,b)$, $\frac{db}{dr} = g(a,b)$

```
ka1=h*dadr(a,b)
kb1=h*dbdr(a,b);
ka2=h*dadr(a+ka1/2,b+kb1/2);
kb2=h*dbdr(a+ka1/2,b+kb1/2);
ka3=h*dadr(a+ka2/2,b+kb2/2);
kb3=h*dbdr(a+ka2/2,b+kb2/2);
ka4=h*dadr(a+ka3,b+kb3);
kb4=h*dbdr(a+ka3,b+kb3);

a = a  + (1.0/6.0)*(ka1 + 2*ka2 + 2*ka3 + ka4)
b = b  + (1.0/6.0)*(kb1 + 2*kb2 + 2*kb3 + kb4)
```

Sometimes there's no explicit x or t or r to worry about

Runge-Kutta: two equations

x and y at time t

```
kx1=dt*dxdt(t,x,y);
ky1=dt*dydt(t,x,y);
kx2=dt*dxdt(t+dt/2,x+kx1/2,y+ky1/2);
ky2=dt*dydt(t+dt/2,x+kx1/2,y+ky1/2);
kx3=dt*dxdt(t+dt/2,x+kx2/2,y+ky2/2);
ky3=dt*dydt(t+dt/2,x+kx2/2,y+ky2/2);
kx4=dt*dxdt(t+dt,x+kx3,y+ky3);
ky4=dt*dydt(t+dt,x+kx3,y+ky3);
x = x  + (1.0/6.0)*(kx1 + 2*kx2 + 2*kx3 + kx4);
y = y  + (1.0/6.0)*(ky1 + 2*ky2 + 2*ky3 + ky4);
```

Must calculate all k1's together

all k2s together

all k3s together

all k4s together

Push x and y forward by dt

x and y at time t+dt

Orbits – Conserved Quantities

- For a system like Newtonian Gravity, there are conserved quantities such as angular momentum and energy
- Some methods do better at conserving these quantities than others even if the formal accuracy (Order) appears to give large errors
- The reason is that errors do not systematically accumulate for special integrators.
    - Modest errors that don’t increase over time are better than small errors than add up indefinitely!

Leapfrog
Good for orbits O(dt^2)

Find $x(t), y(t), v_x(t), v_y(t)$ given $x(t_0), y(t_0),\ v_x(t_0), v_y(t_0)$

and $\dfrac{dx}{dt} = v_x, \dfrac{dy}{dt} = v_y, \dfrac{dv_x}{dt} = f(x,y)\ ,\quad \dfrac{dv_y}{dt} = g(x,y)$

```
vx = vx + dt/2*dvxdt(x,y);
vy = vy + dt/2*dvydt(x,y);

x = x  + dt*vx;
y = y  + dt*vy;

vx = vx + dt/2*dvxdt(x,y);
vy = vy + dt/2*dvydt(x,y);
```

Leapfrog
Good for orbits $O(dt^2)$

Velocity and position at time t

```
vx = vx + dt/2*dvxdt(x,y);
vy = vy + dt/2*dvydt(x,y);
```

Kick velocity forward by dt/2
Must use position + acceleration at time t

Now Velocity at time t+dt/2

```
x = x  + dt*vx;
y = y  + dt*vy;
```

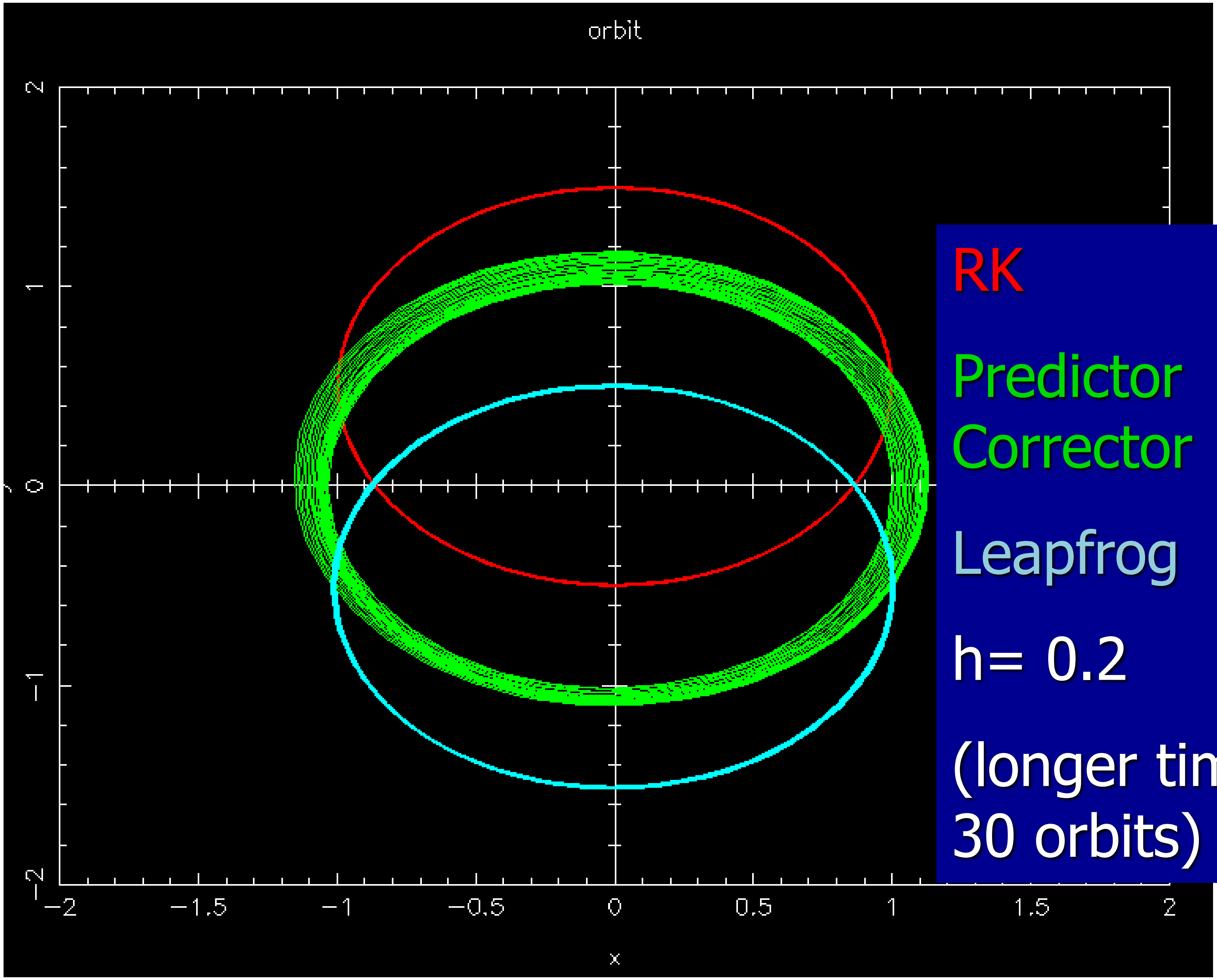
Drift position forward by dt
Must use velocity at time t+dt/2

Now position at time t+dt

```
vx = vx + dt/2*dxvdt(x,y);
vy = vy + dt/2*dvydt(x,y);
```

Kick velocity forward by another dt/2
Use position + acceleration at time t+dt

Now both position and Velocity at time t+dt



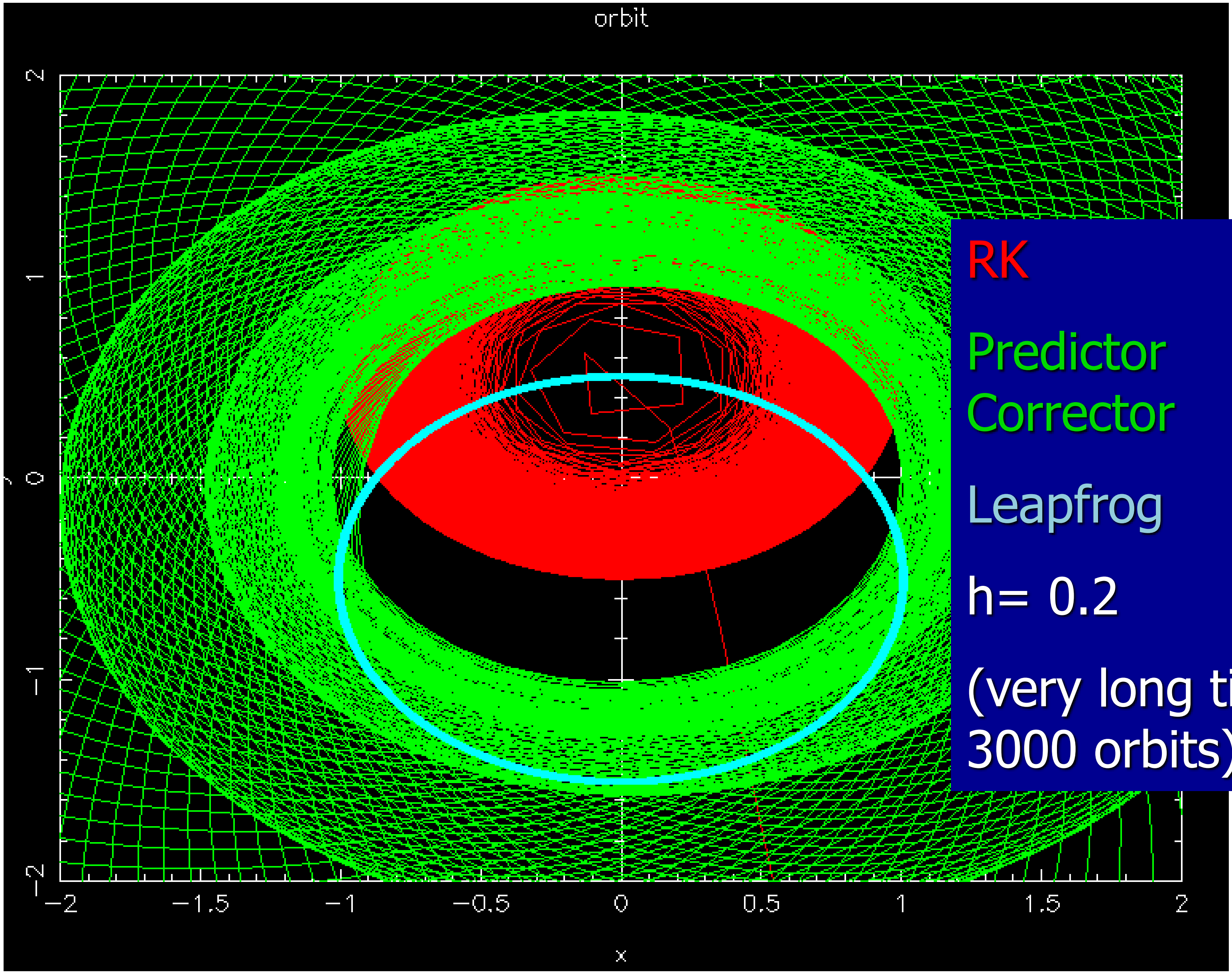
RK

Predictor Corrector

Leapfrog

h= 0.2

(longer time = 30 orbits)



RK

Predictor Corrector

Leapfrog

h= 0.2

(very long time = 3000 orbits)

Leapfrog
Good for dynamical systems

Find $x(t), y(t), v_x(t), v_y(t)$ given $x(t_0), y(t_0),\ v_x(t_0), v_y(t_0)$

and $\frac{dx}{dt} = v_x, \frac{dy}{dt} = v_y, \frac{dv_x}{dt} = f(x,y)\ , \quad \frac{dv_y}{dt} = g(x,y)$

Dynamical systems derived from a Hamiltonian do very well with Leapfrog (2nd order symplectic)

Cases that do well include orbits, particles and colliding objects

Leapfrog, also called Verlet, used in game engines

Solving Second order ODE

$$\frac{d^2y}{dx^2} + g(x,y)\ \frac{dy}{dx} + f(x,y) = 0 \qquad (1)$$

Make a new variable: $z = \dfrac{dy}{dx}$

(1) becomes:

$$\frac{dz}{dx} + g(x,y)\, z + f(x,y) = 0$$

Solve two 1st order ODEs together:

$$\frac{dz}{dx} = -\, g(x,y)\, z - \ f(x,y)$$

$$\frac{dy}{dx} = z$$

Example: Simple Harmonic Oscillator

$$\frac{d^2x}{dt^2} = -\, k\ x \qquad (1)$$

Define velocity, v: $v = \dfrac{dx}{dt}$

(1) becomes: $\dfrac{dv}{dt} = \ -\ k\ x$

Solve two 1st order ODEs together:

$$\frac{dv}{dt} = \ -\ k\ x \qquad (1a)$$

$$\frac{dx}{dt} = v \qquad (1b)$$

Partial Differential Equations: Finite Difference Methods

In general you need to replace all derivatives with numerical approximations

$$\frac{\partial u}{\partial t} + g(x,y)\frac{\partial u}{\partial x} + f(x,y) = 0$$

du/dx ~ (u(x+dx)-u(x-dx))/(2*dx)

du/dt ~ (u(t+dt)-u(t))/dt

Rewrite PDE as algebra, solve for u(t+dt)

Similar to Numerical Calculus homework

Note: We'll explicitly help you set this up for PDE projects. Some PDE's are difficult to solve.

Partial Differential Equations: Finite Difference Methods

e.g. Diffusion Equation $\frac{\P u}{\P t} = D \frac{\P^2 u}{\P x^2}$

u = Temperature or concentration, etc..

Represent current state (time t) with array u[n]

Each element u[i] or u_i represents u at position i*Δx

$du/dx|_{i+1/2} = (u_{i+1} - u_i)/\Delta x$

$d^2u/dx^2|_I \;=\; (du/dx|_{i+1/2} - du/dx|_{i-1/2})/\Delta x$

$\qquad\qquad = \; (u_{i+1} - 2u_i + u_{i-1})/\Delta x^2$

$du/dt|_i \;=\; (unew_i\, |_{t+\Delta t} - u_i|_t)/\Delta t$

For each i: $unew_i = u_i + D\, \Delta t/\Delta x^2 (u_{i+1} - 2u_i + u_{i-1})$

Partial Differential Equations: Stability

Solutions are unstable if you remove more u per step than was originally there.

Clues: negative values, oscillations in plots

e.g. Diffusion Equation $$\frac{\P u}{\P t} = D \frac{\P^2 u}{\P x^2}$$

$unew_i = u_i + D\,\Delta t/\Delta x^2(u_{i+1} – 2u_i + u_{i-1})$

Consider $u_{i+1} = u_{i-1} = 0$

$unew_i$ only stays positive if $\Delta t < \Delta x^2/(2D)$

In general solutions are not that accurate if you are nearly unstable anyway

Parameter Values

- In SI units, parameter values can be really large or small
- e.g. orbit radius $r = 10^{11}$ metres
- This can cause a lot of trouble with precision and things taking too long to change (or changing too fast …)
- For this project, choose values close to one, especially for testing
- e.g. $r = 1$, $G = 1$, $M = 1$
- This automatically means $h = 0.01$ is ok and helps you identify early bugs without worrying if precision is a problem

Monte Carlo: MC

- General idea:

I have a distribution of events, how often does a certain outcome occur?

OR

I have a distribution of input events – what does the distribution of outcomes look like?

Exploring cases

- x=(1.0*rand())/RAND_MAX returns values evenly distributed between 0 and 1
- x=(2*3.14159*rand())/RAND_MAX returns values evenly distributed between 0 and 2π for random angles
- You can also explore values in a regular way. This is often more efficient

e.g. for (i=0;i<100;i++) x=(i+0.5)/100. explores values from 0 to 1 in an evenly distributed way

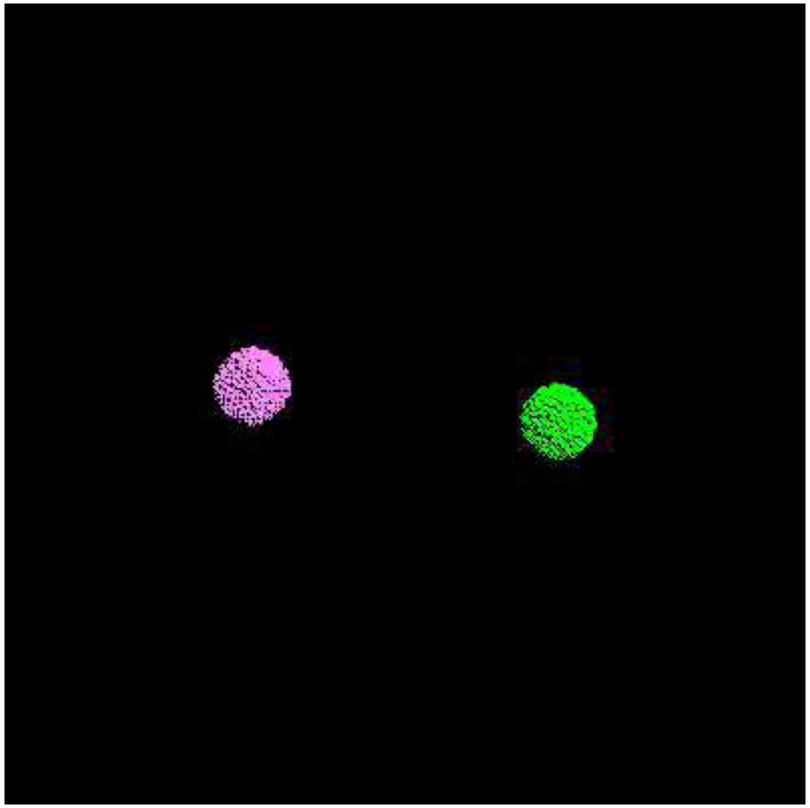
Monte Carlo to extend a simple model

Write a basic program – e.g. integrate a single orbit

Work out how to classify the result

Using for loops run the basic code for many cases – determine which cases given which results (or the probability of certain results)

Monte Carlo – Fancy Example



Collide Two Asteroids

Cases: Initial speed,
initial asteroid sizes,
initial impact parameter,
initial spin (5 loops!)

Results to record:

Primary Outcomes

1 Miss entirely
2 Merge (largest piece bigger than before)
3 Disrupt (largest piece smaller than before)

Secondary Outcome: Binary system formed?

Written Feedback to project proposal on avenue

1. Methods to use to build your program, e.g. ODE (RK/Leapfrog), Monte Carlo
2. Program milestones (initial goal for your program, e.g. for your demo)
3. Ways you might test the basic program components for correctness
4. Key parameters for your program, e.g. how many steps or cases to consider
5. An overall goal for your program – how the basic idea can be expanded to tackle specific questions
6. Scientific goals and tests for the expanded program – questions to explore either in code or in the write-up