ODEs

- ODEs arise in many physics problems
- Classifications:
  - Initial value problems
  - Boundary value problems
- As with the other topics, there are a large number of different methods
  - We just want to see the basic ideas and popular methods
- We'll primarily follow the discussion from Garcia with some additions along the way

Example: Projectile Motion

- Evolution:

$$\ddot{\mathbf{r}} = \frac{1}{m}\mathbf{F_a}(\dot{\mathbf{r}}) - g\hat{\mathbf{y}}$$

- System of equations:

$$\dot{\mathbf{v}} = \frac{1}{m}\mathbf{F_a}(\mathbf{v}) - g\hat{\mathbf{y}} \qquad \dot{\mathbf{r}} = \mathbf{v}$$

- Note: most 2nd and higher-order ODEs can be written as a system of first-order ODEs by introducing new variables.

Example: Projectile Motion

- Air resistance
  - $\mathbf{F_a} = -\frac{1}{2} C \rho A |v| \mathbf{v}$
  - Coefficient, C , depends on geometry and speed of projectile
    - Smooth sphere, C relative high at low speeds but drops when vortices and turbulence appears.
    - C likely varies over the flight

- We know the answer for $C = 0$
  - Develop and test for this case to determine robustness of method
  - Only then go on to look at the cases with air resistance
  - This becomes a test case

Euler's Method

- Start with our first order derivative:

$$\frac{\mathbf{v}^{n+1} - \mathbf{v}^n}{\tau} = \mathbf{a}(\mathbf{r}^n, \mathbf{v}^n) + \mathcal{O}(\tau)$$

$$\frac{\mathbf{r}^{n+1} - \mathbf{r}^n}{\tau} = \mathbf{v}^n + \mathcal{O}(\tau)$$

  - Here we use the convention that the time-level is denoted by superscripts

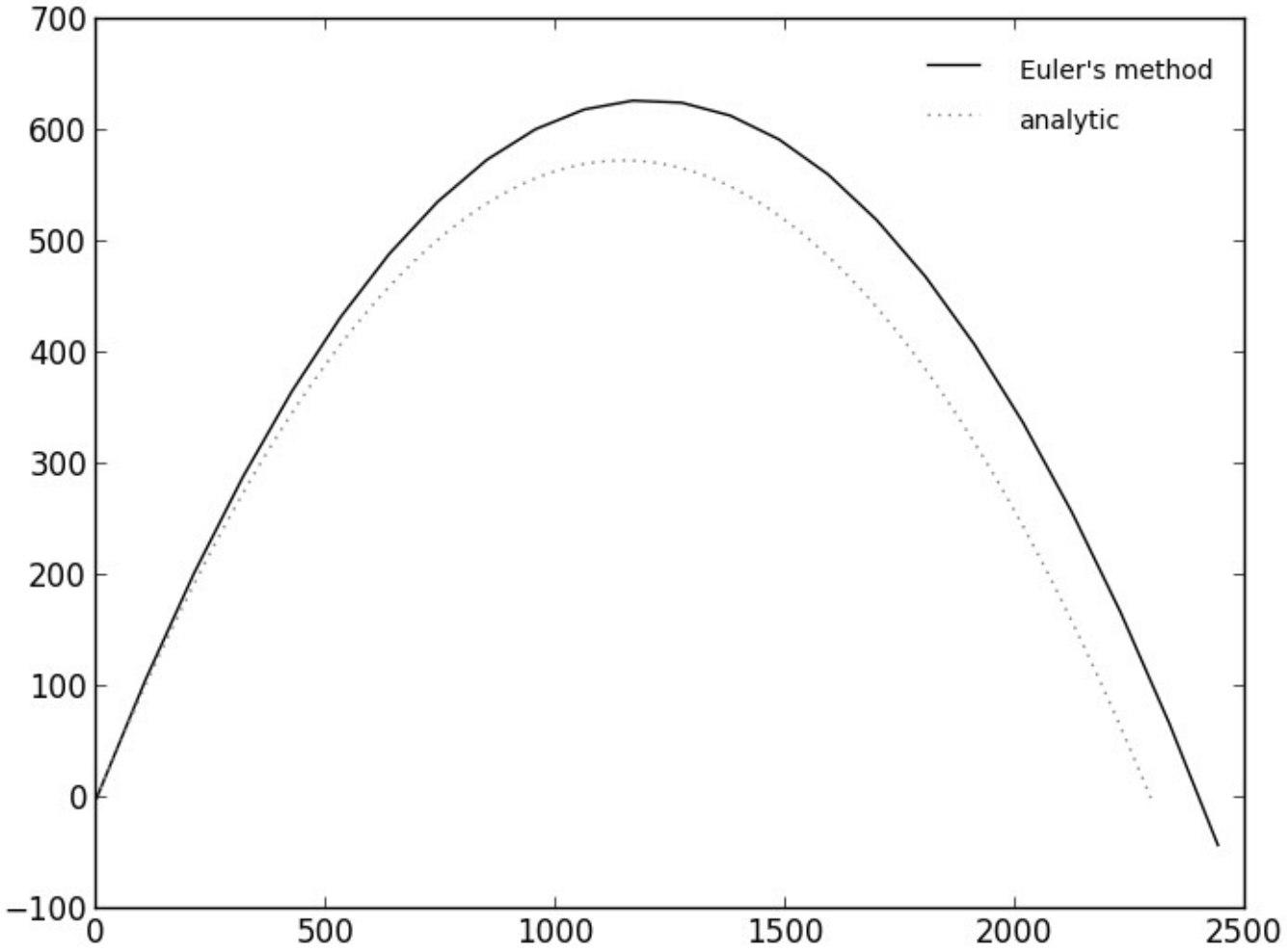
- Expressing the new state in terms of the old:

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \tau\mathbf{a}^n + \mathcal{O}(\tau^2)$$

$$\mathbf{r}^{n+1} = \mathbf{r}^n + \tau\mathbf{v}^n + \mathcal{O}(\tau^2)$$

  - Here we see that the local truncation error is τ^2

Euler's Method



Projectile w/o air-resistance, initial velocity is 15 m/s at 45°

Euler-Cromer Method

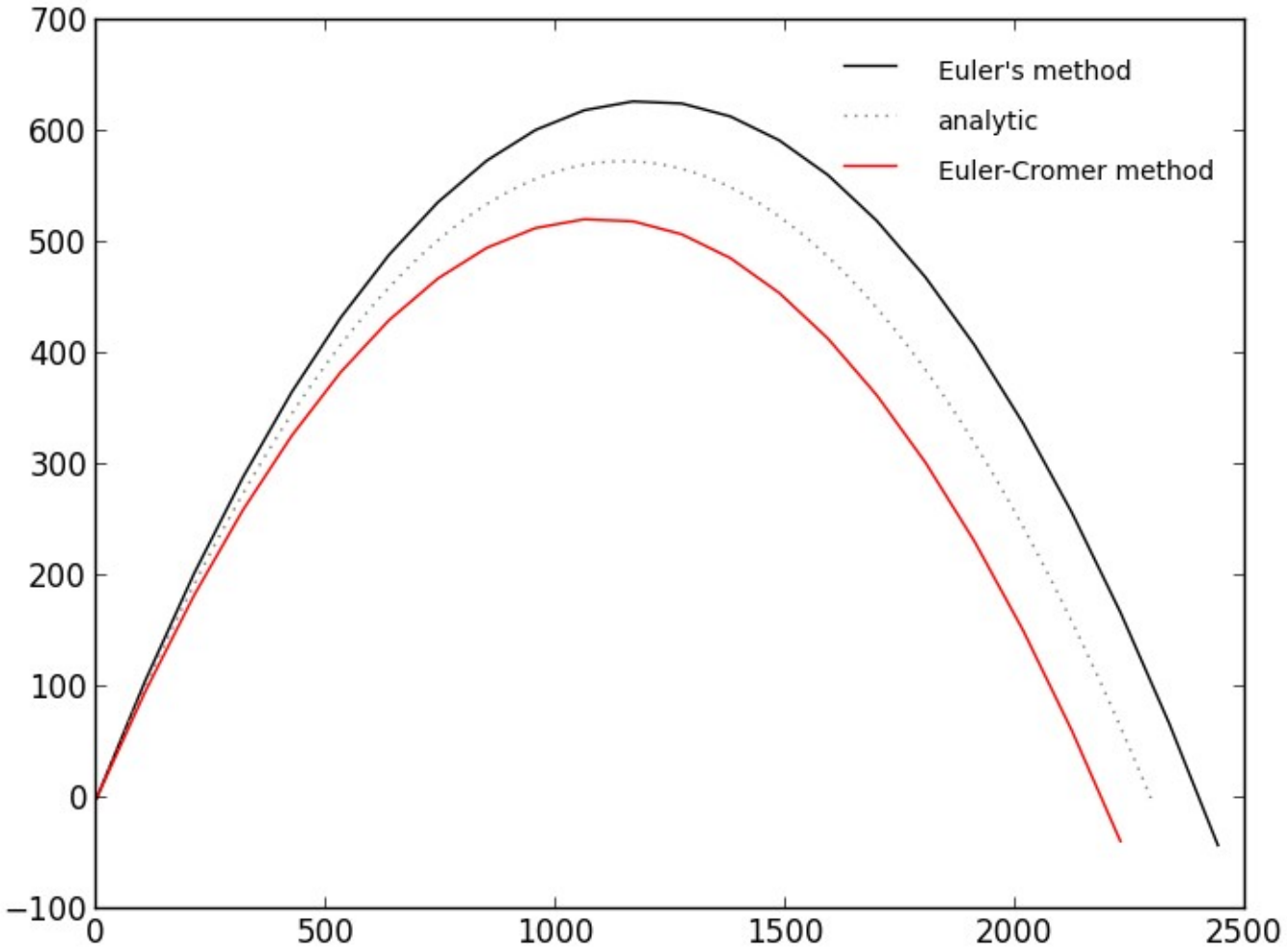
- Use updated velocity to update position:

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \tau \mathbf{a}^n + \mathcal{O}(\tau^2)$$

$$\mathbf{r}^{n+1} = \mathbf{r}^n + \tau \mathbf{v}^{n+1} + \mathcal{O}(\tau^2)$$

  - Simple change, but we'll see later that it has better conservation properties in some cases (we'll explore this in the homework...)
  - Still same formal local truncation error

Euler-Cromer Method

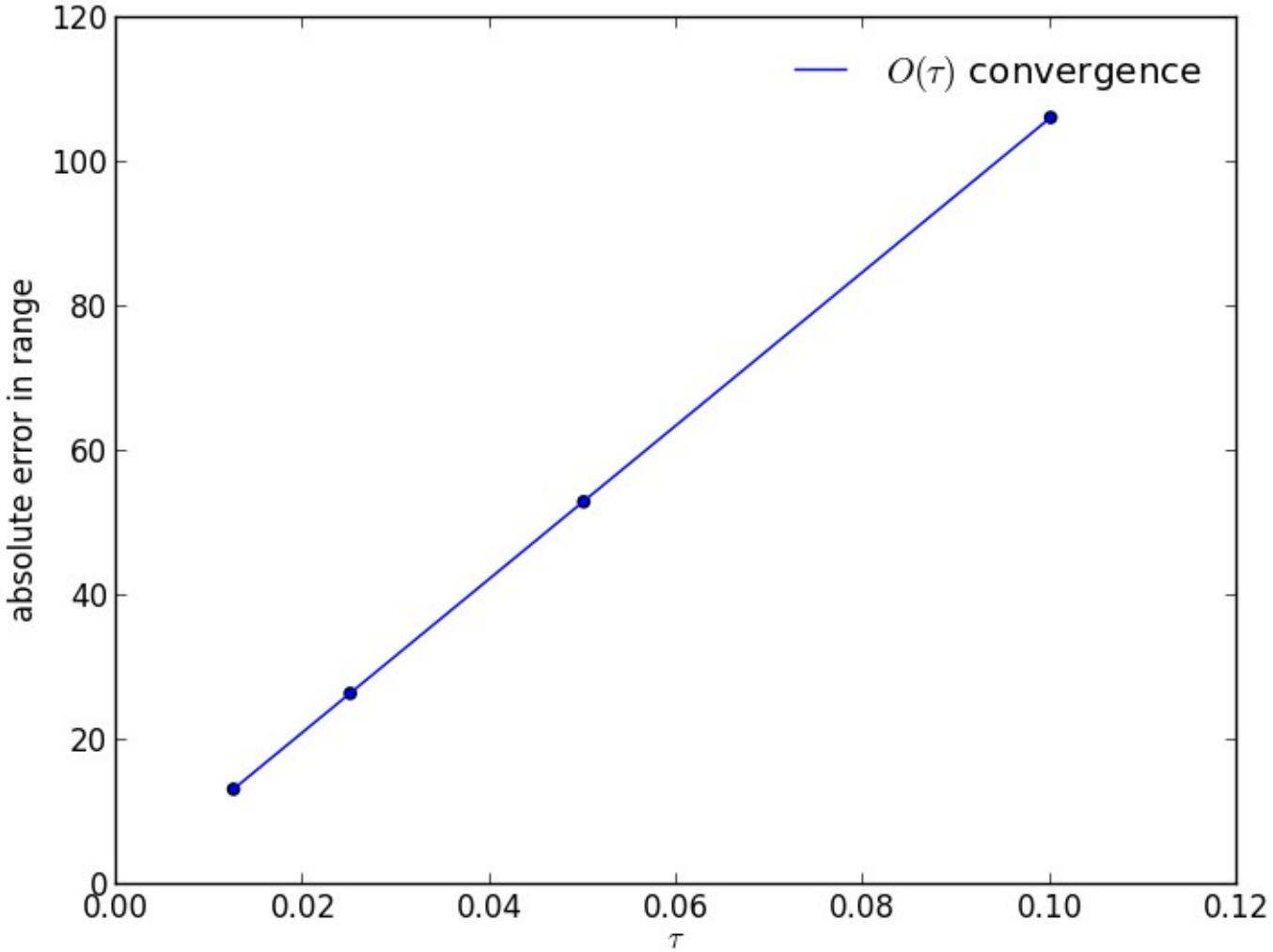


Let's look at the code...

Local vs. Global Error

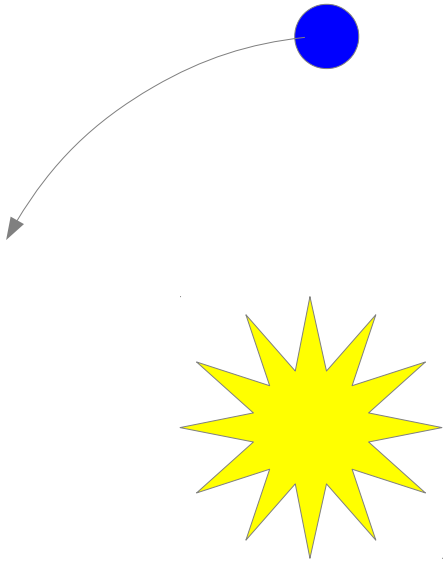
- What do we use as our metric for convergence?
  - Total range as compared to analytic result?
  - Note that the integration stops when $y < 0$—some interpolation needed.
  - Time of flight? Peak height? ...
- Integration consists of many steps—local error accumulates.
  - Number of steps to integrate to time T is $N = T/\tau$
  - Global truncation is ~ $N \times$ local truncation
    - Euler: global truncation is ~ τ
- Choice of stepsize is complicated
  - Ideally it should be chosen such that the solution doesn't change much over a step

Euler Method Convergence



Here the error was computed by first doing a quadratic interpolation of x(y) at the end of the integration and evaluating at x(0) to get the computed range.

Example: Orbits



- Consider orbits around the Sun
  - Another simple system that allows us to explore the properties of ODE integrators

$$\dot{\mathbf{x}} = \mathbf{v} \qquad \dot{\mathbf{v}} = -\frac{GM\mathbf{r}}{r^3}$$

  - Kepler's law (neglecting orbiting object mass):

$$4\pi^2 a^3 = GM_\star P^2$$

  - Work in units of AU, solar masses, and years
    - $GM = 4\ \pi^2$

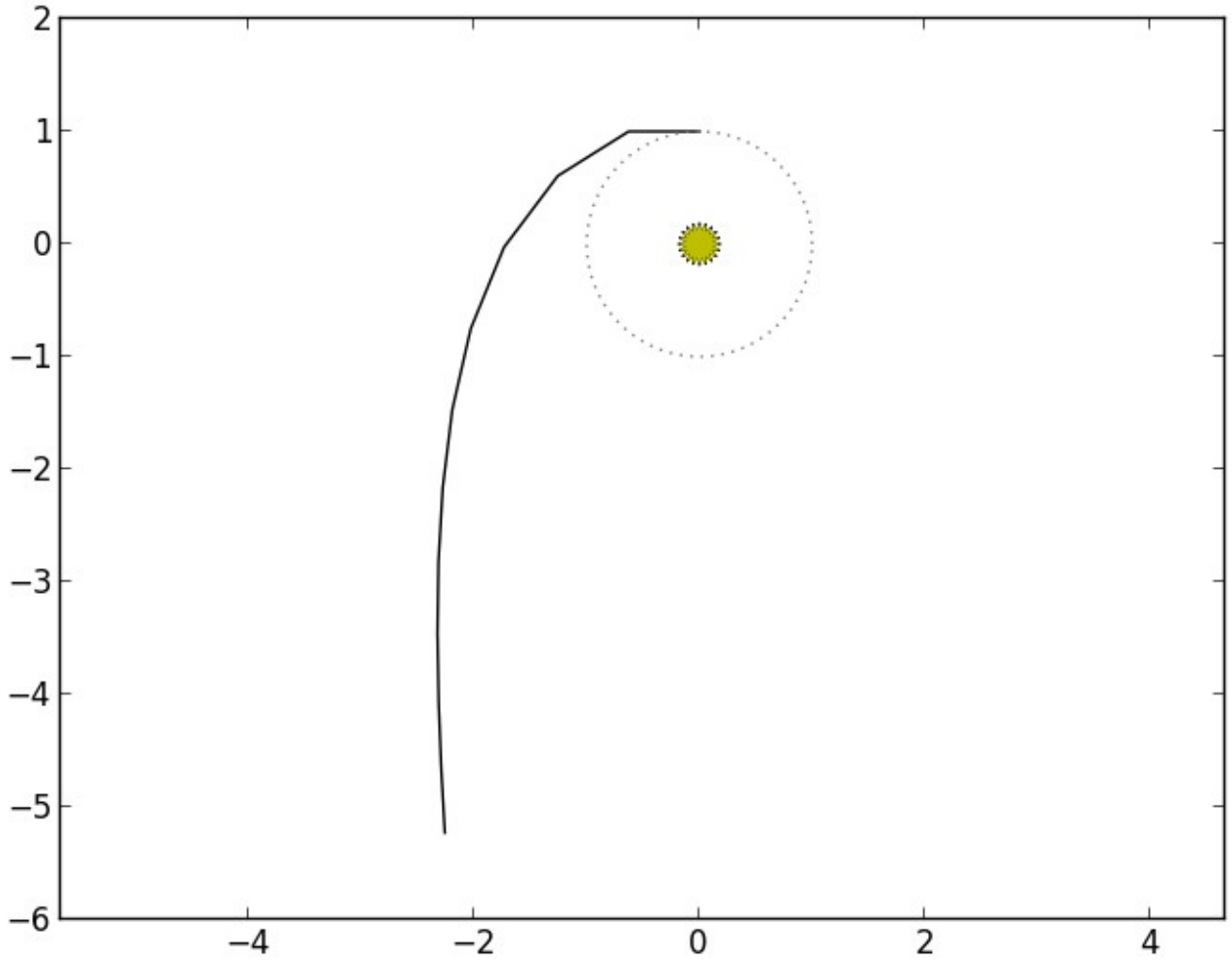
Orbits: Euler's Method

- Simplest case again: Euler's method

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \tau \mathbf{v}^n \qquad \mathbf{v}^{n+1} = \mathbf{v}^n + \tau \mathbf{a}^n$$

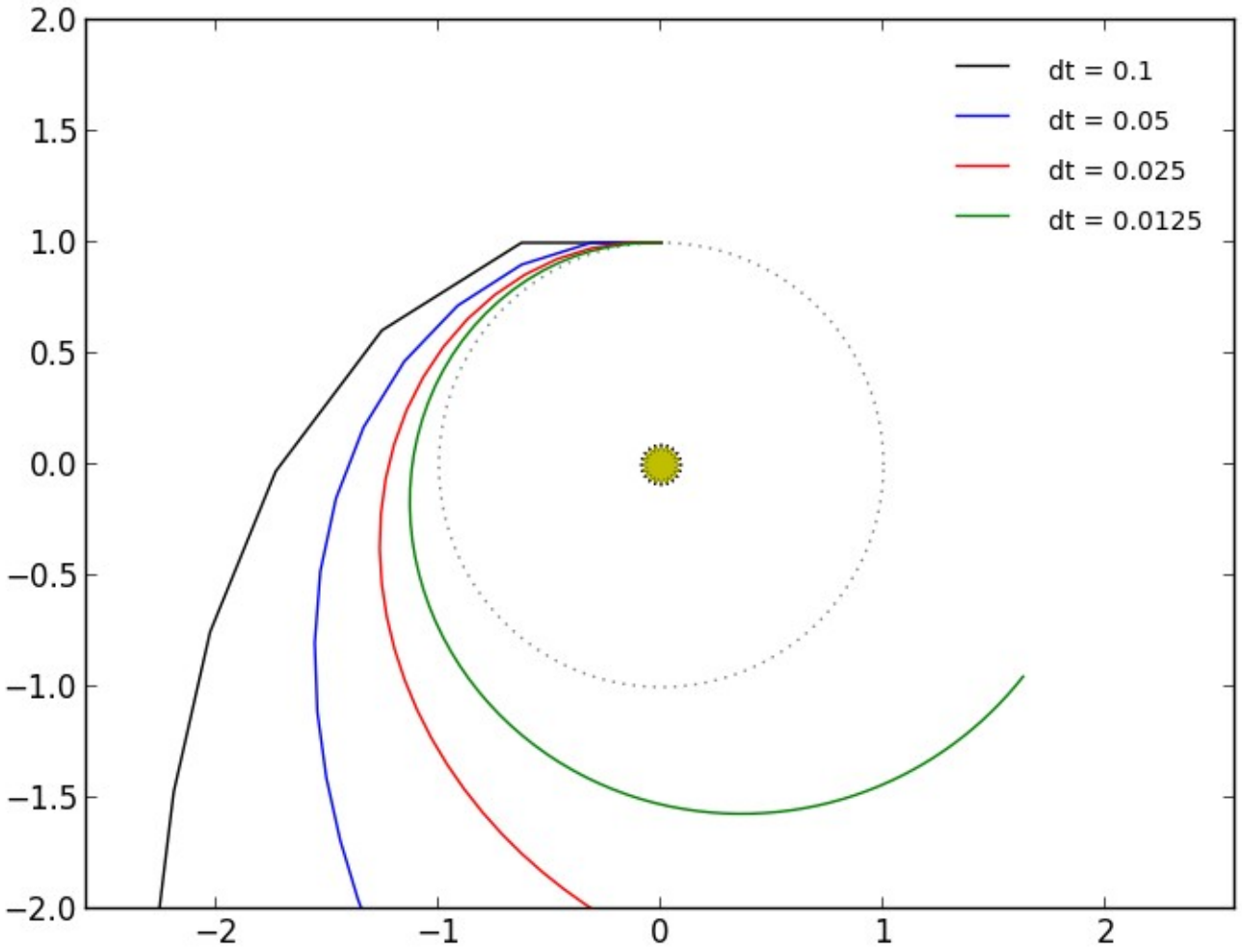
- Need to specify a semi-major axis and eccentricity
- Initial conditions:
  - $x = 0,\ y = a(1-e)$
  - $u = -\sqrt{\frac{GM}{a}\frac{1+e}{1-e}},\ v = 0$
    - This is counter-clockwise orbiting

Orbits: Euler's Method



Our planet escapes—clearly energy is not conserved here!

Orbits: Euler's Method



Things get better with a smaller timestep, but this is still first-order

Let's look at the code and see how small we need to get a closed circle

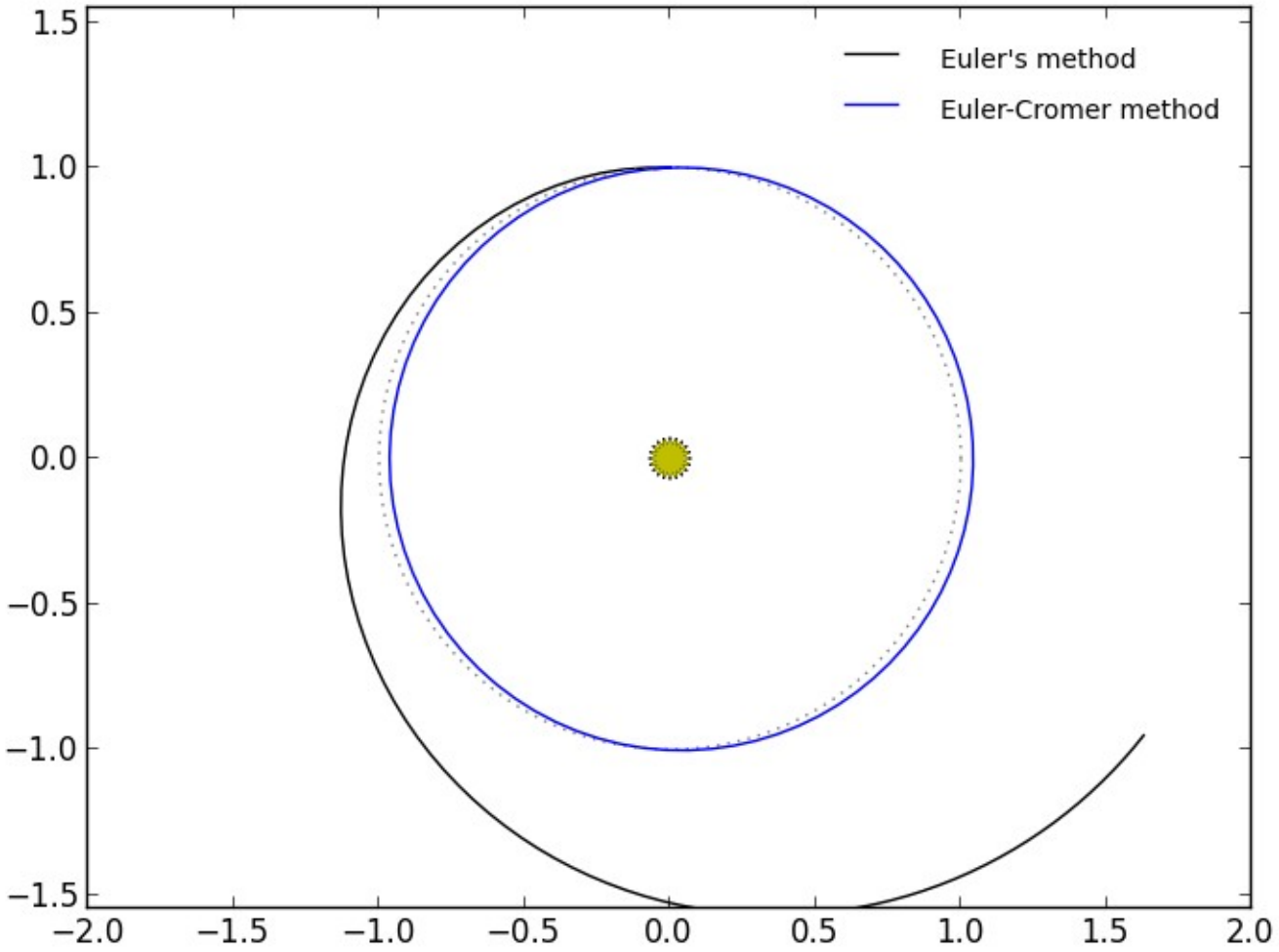
Orbits: Euler-Cromer Method

- Euler-Cromer is a very simple change (again):

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \tau \mathbf{v}^{n+1} \qquad \mathbf{v}^{n+1} = \mathbf{v}^n + \tau \mathbf{a}^n$$

  - This treats the position implicitly.
  - We didn't see much improvement with projectiles...

Orbits: Euler-Cromer Method



Simple change, still first-order, but much better. Why?

Euler-Cromer and Angular Momentum

- Consider the angular momentum of an orbit:

$$l = |\mathbf{v} \times \mathbf{r}| = uy - vx$$

- Using the E-C system:

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \tau \mathbf{a}(\mathbf{r}^n)$$
$$\mathbf{r}^{n+1} = \mathbf{r}^n + \tau \mathbf{v}^{n+1}$$

- New angular momentum can be expressed as:

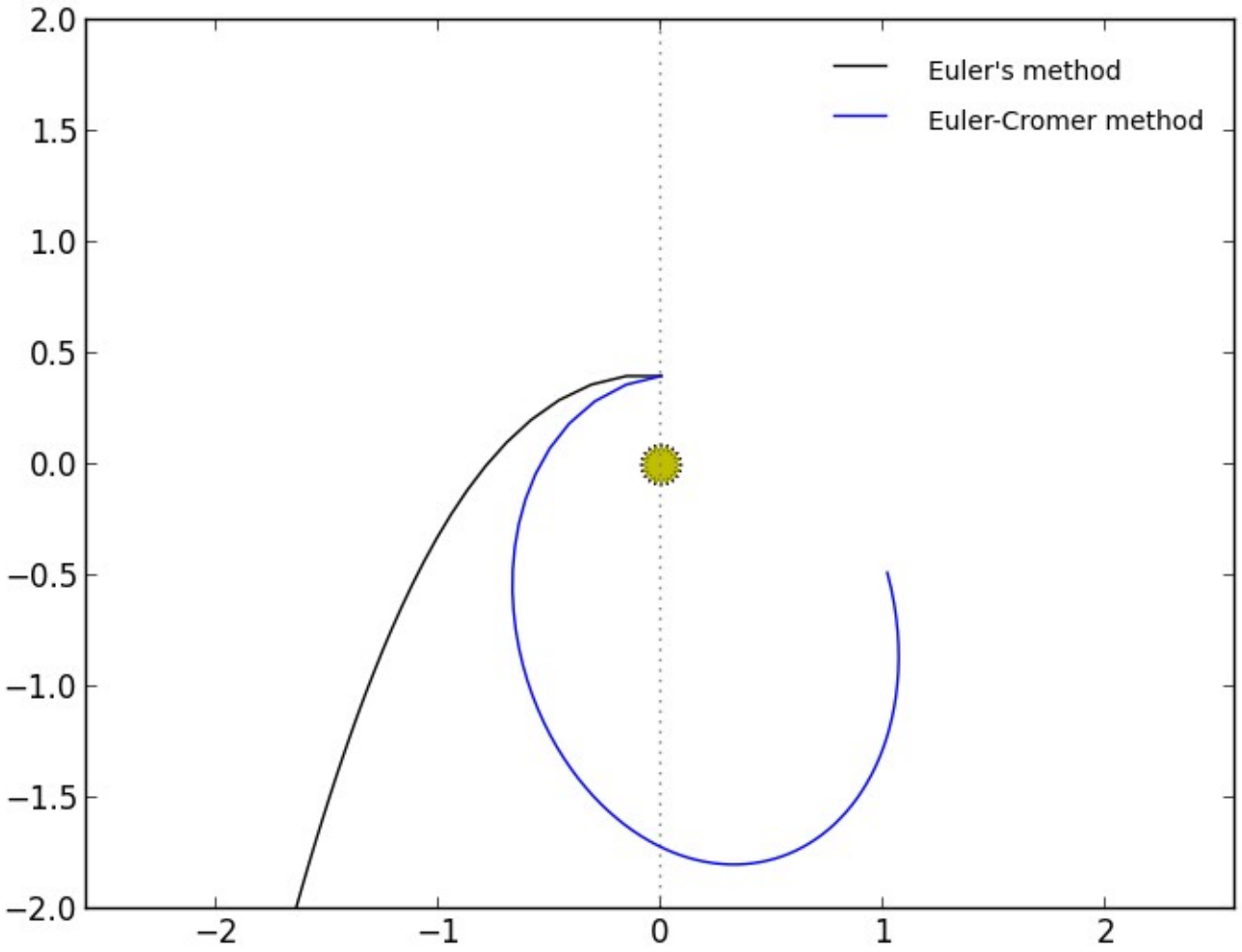
$$l^{n+1} = u^{n+1}y^{n+1} - v^{n+1}x^{n+1} = \underbrace{u^n y^n - v^n x^n}_{\text{This is } l^n} + \tau(\underbrace{a^n_{(x)} y^n - a^n_{(y)} x^n}_{\text{This is } a \times r \text{ = 0 for central potental}})$$

- Angular momentum is conserved!

Elliptical Orbits

- We can just as easily model elliptical orbits.
- Our initial conditions start at perihelion
- Kepler's 3rd law still applies here, so the period is unchanged (for a given semi-major axis)

Elliptical Orbits



Euler-Cromer has problems here—notice that the ellipse is precessed.

Higher-order Methods

- Recall that the center-difference was second-order accurate
- Consider:

$$\frac{\mathbf{r}^{n+1} - \mathbf{r}^n}{\tau} = \mathbf{v}^{n+1/2} + \mathcal{O}(\tau^2) \qquad \frac{\mathbf{v}^{n+1} - \mathbf{v}^n}{\tau} = \mathbf{a}^{n+1/2} + \mathcal{O}(\tau^2)$$

- The updates are then:

$$\mathbf{r}^{n+1} = \mathbf{r}^n + \tau \mathbf{v}^{n+1/2} + \mathcal{O}(\tau^3)$$

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \tau \mathbf{a}^{n+1/2} + \mathcal{O}(\tau^3)$$

- This is third-order accurate (locally), but we don't know how to compute the state at the half-time

Higher-order Methods

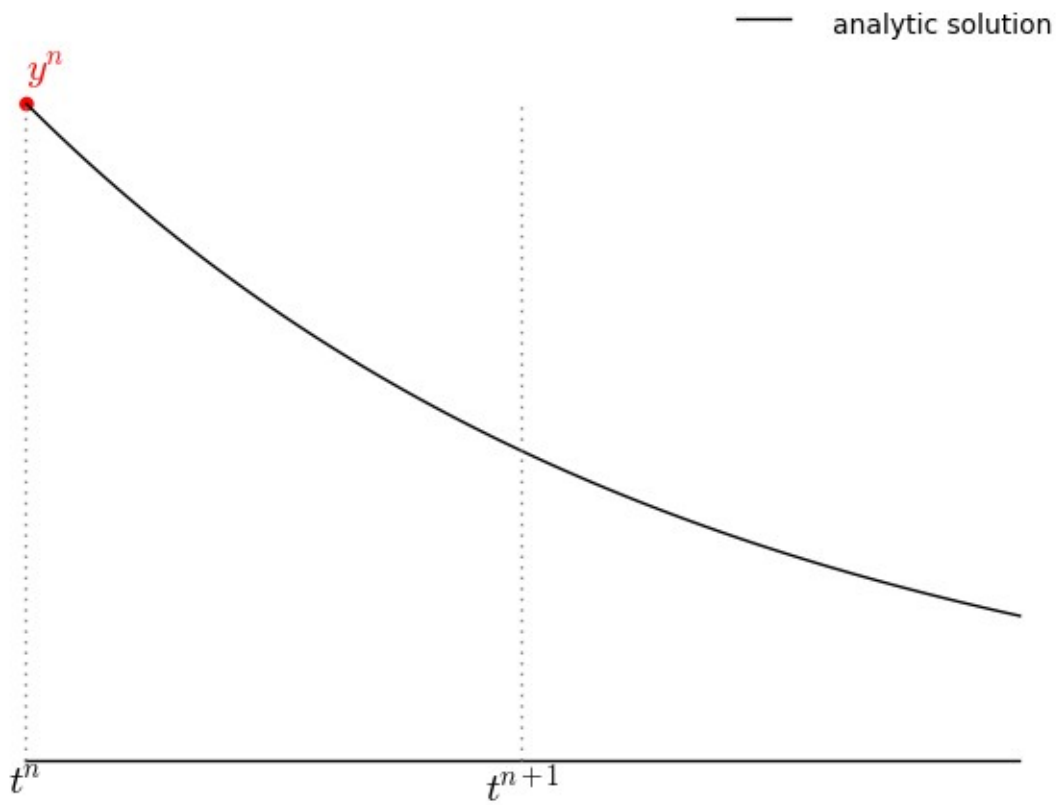
- We can first compute the state at the half-time using an Euler step through $\tau/2$
  - Two-step process

$$\mathbf{r}^\star = \mathbf{r}^n + (\tau/2)\mathbf{v}^n$$
$$\mathbf{v}^\star = \mathbf{v}^n + (\tau/2)\mathbf{a}^n$$

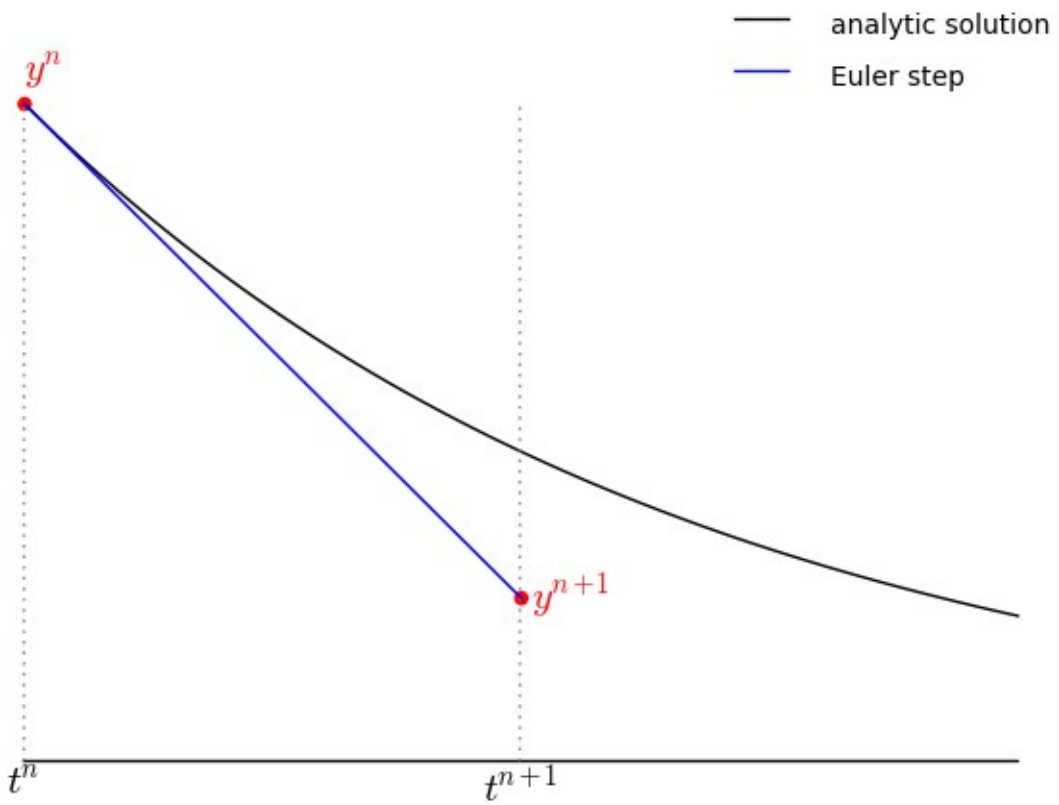
$$\mathbf{r}^{n+1} = \mathbf{r}^n + \tau\mathbf{v}^\star$$
$$\mathbf{v}^{n+1} = \mathbf{v}^n + \tau\mathbf{a}(\mathbf{r}^\star)$$

- This is taking a half step to allow us to evaluate the righthand side of the system at a point centered in the timestep.
- Locally third-order accurate, globally second-order
- Midpoint or 2nd order Runge-Kutta method

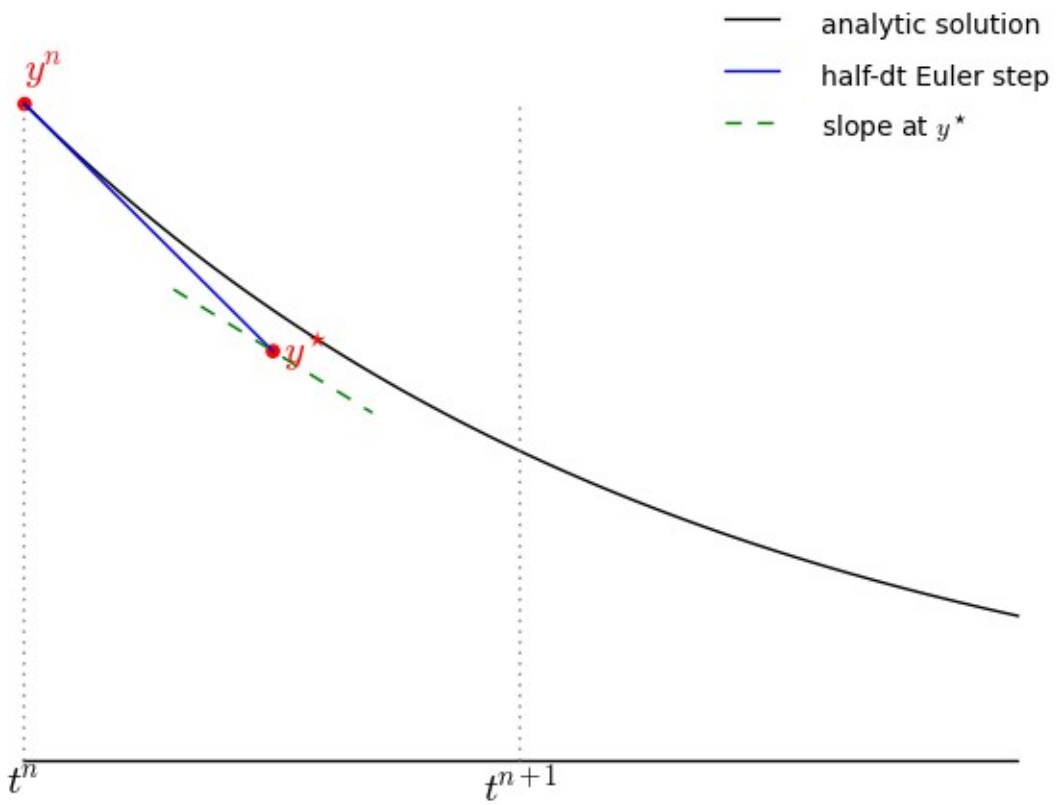
2nd-order Runge-Kutta



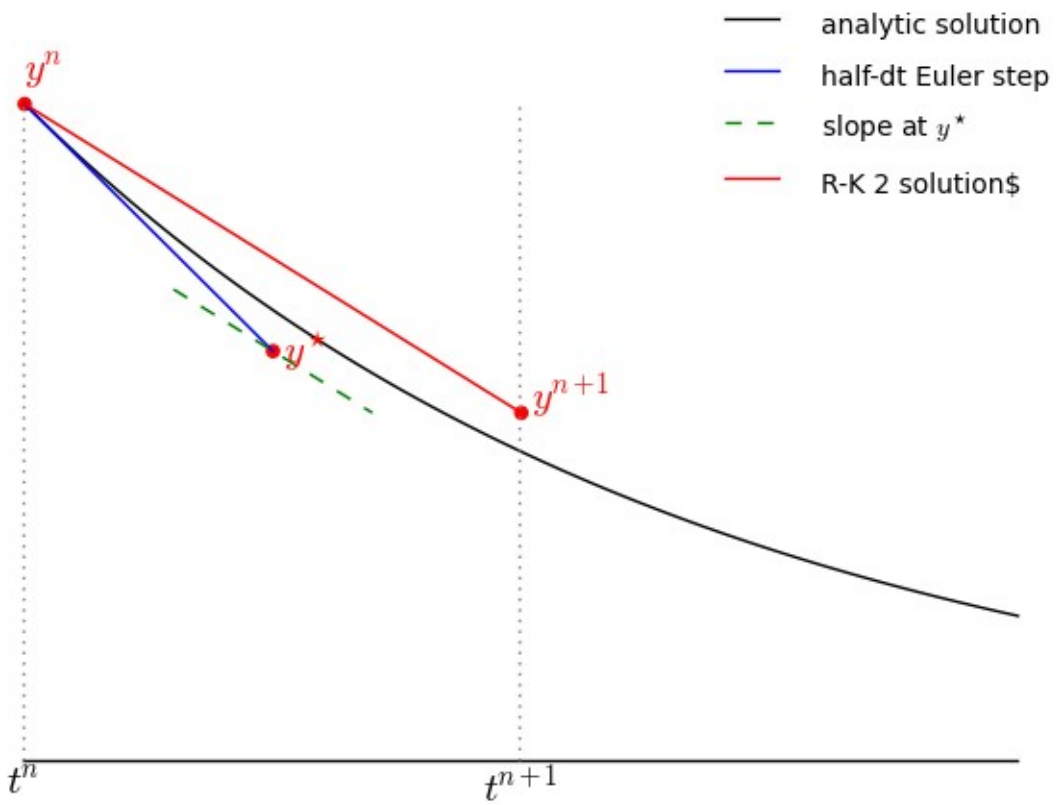
2nd-order Runge-Kutta



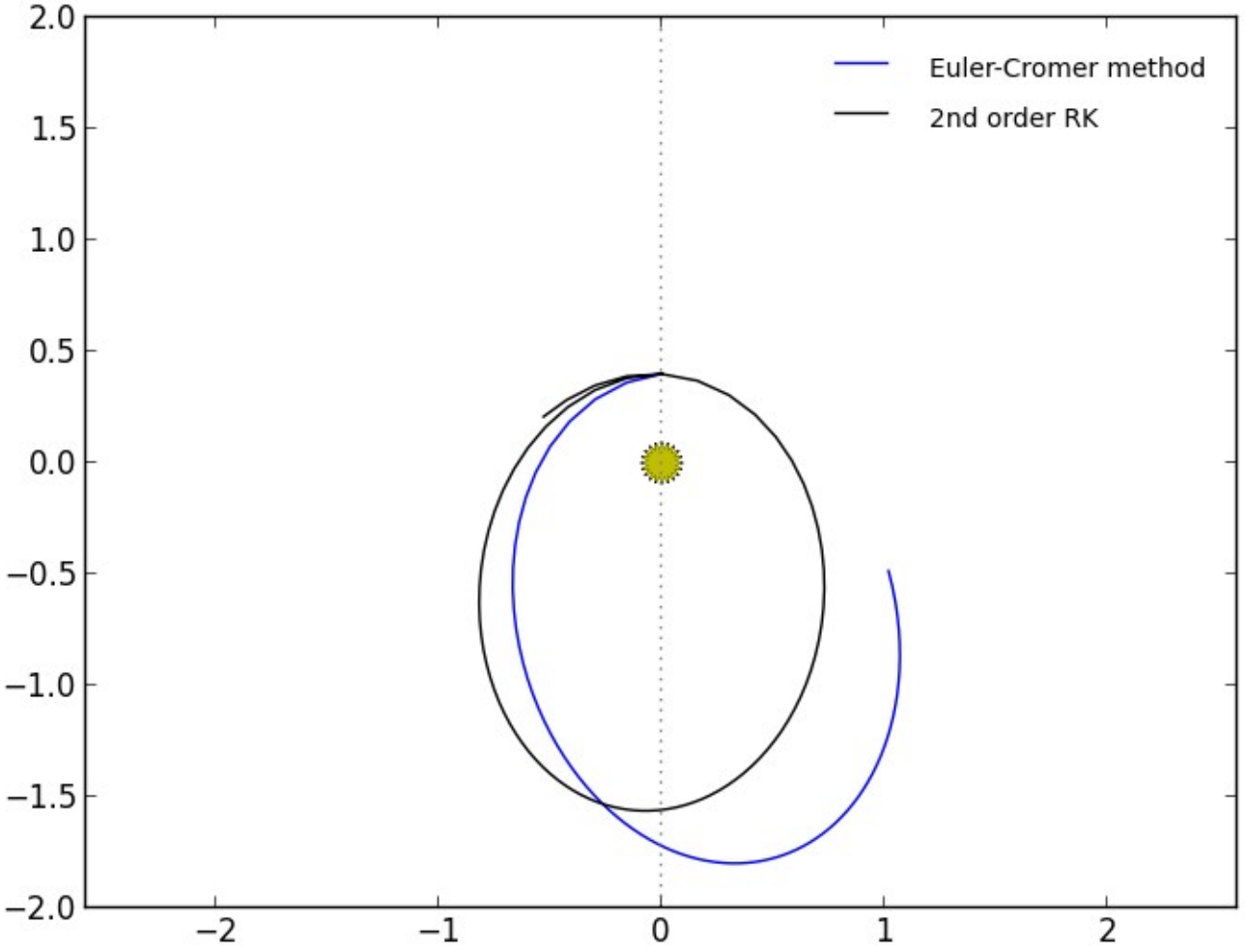
2nd-order Runge-Kutta



2nd-order Runge-Kutta



2nd-order Runge-Kutta



4th-order Runge-Kutta

- Other methods exist—most popular is 4th-order Runge-Kutta
  - Consider system: $\dot{\mathbf{y}} = \mathbf{g}(t, \mathbf{y})$
  - Update through τ :

$$\mathbf{y}^{n+1} = \mathbf{y}^n + \frac{\tau}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) + \mathcal{O}(\tau^5)$$

$$\mathbf{k}_1 = \mathbf{g}(t, \mathbf{y})$$

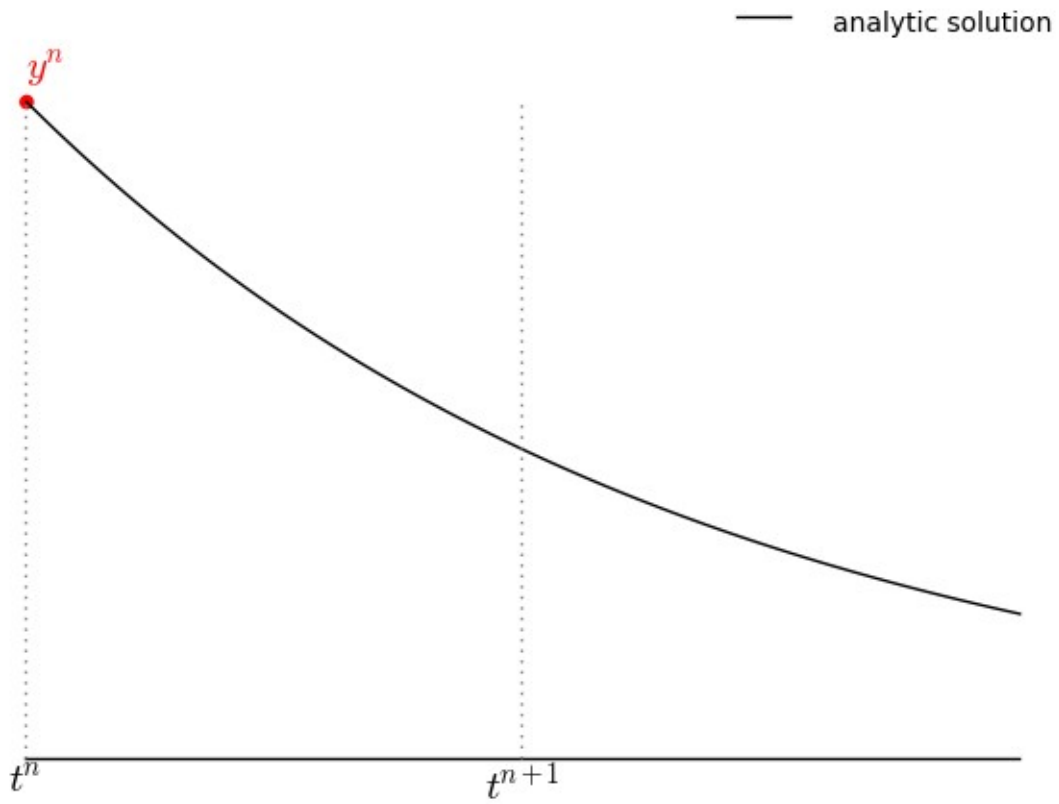
$$\mathbf{k}_2 = \mathbf{g}(t + \tfrac{\tau}{2}, \mathbf{y} + \tfrac{\tau}{2}\mathbf{k}_1)$$

$$\mathbf{k}_3 = \mathbf{g}(t + \tfrac{\tau}{2}, \mathbf{y} + \tfrac{\tau}{2}\mathbf{k}_2)$$

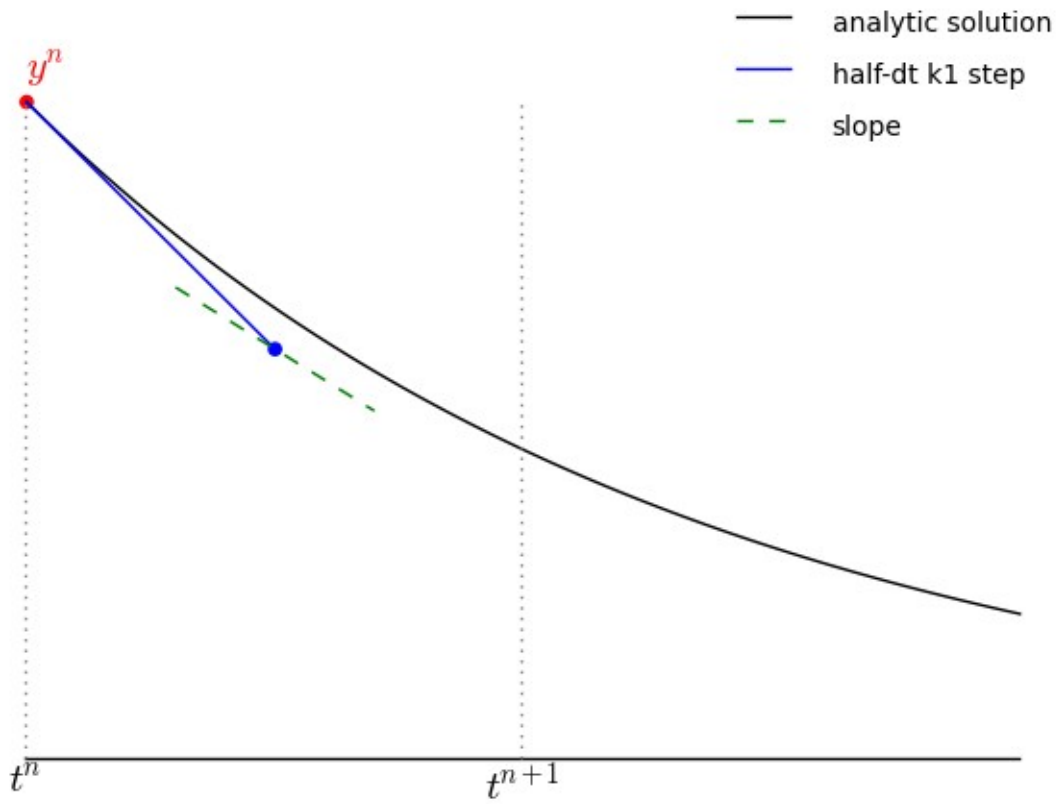
$$\mathbf{k}_4 = \mathbf{g}(t + \tau, \mathbf{y} + \tau\mathbf{k}_3)$$

- Notice the similarity to Simpson's integration
- Derivation found in many analysis texts

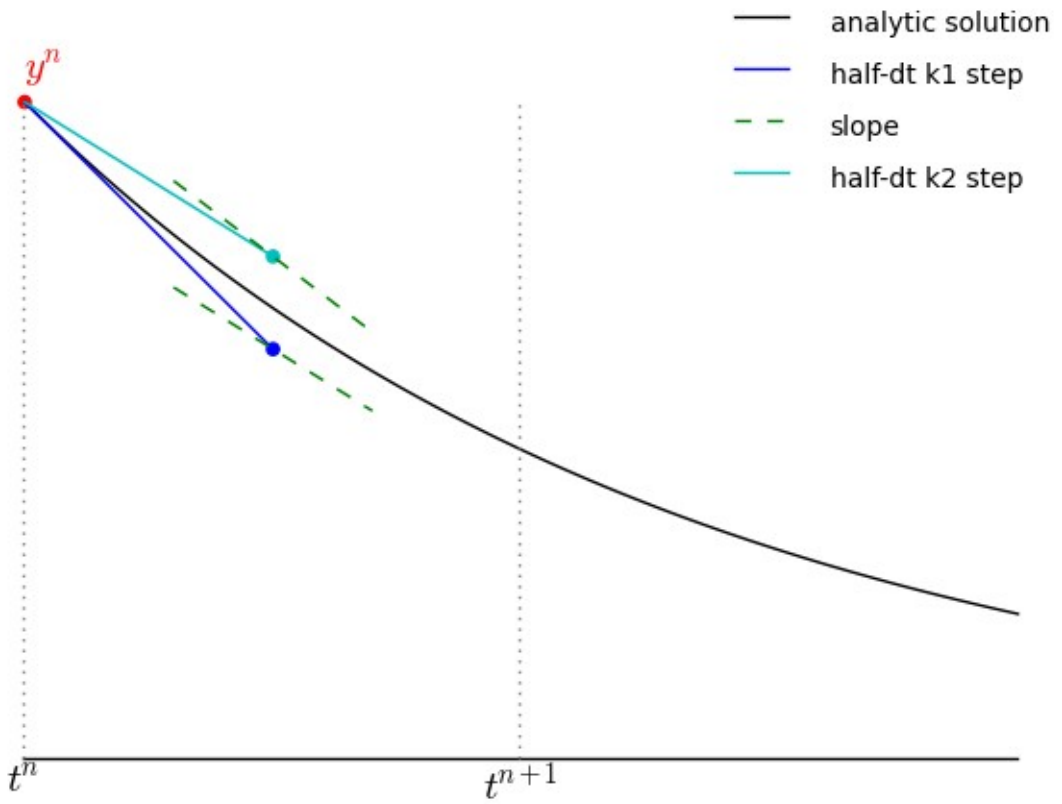
4^{th}-order Runge-Kutta



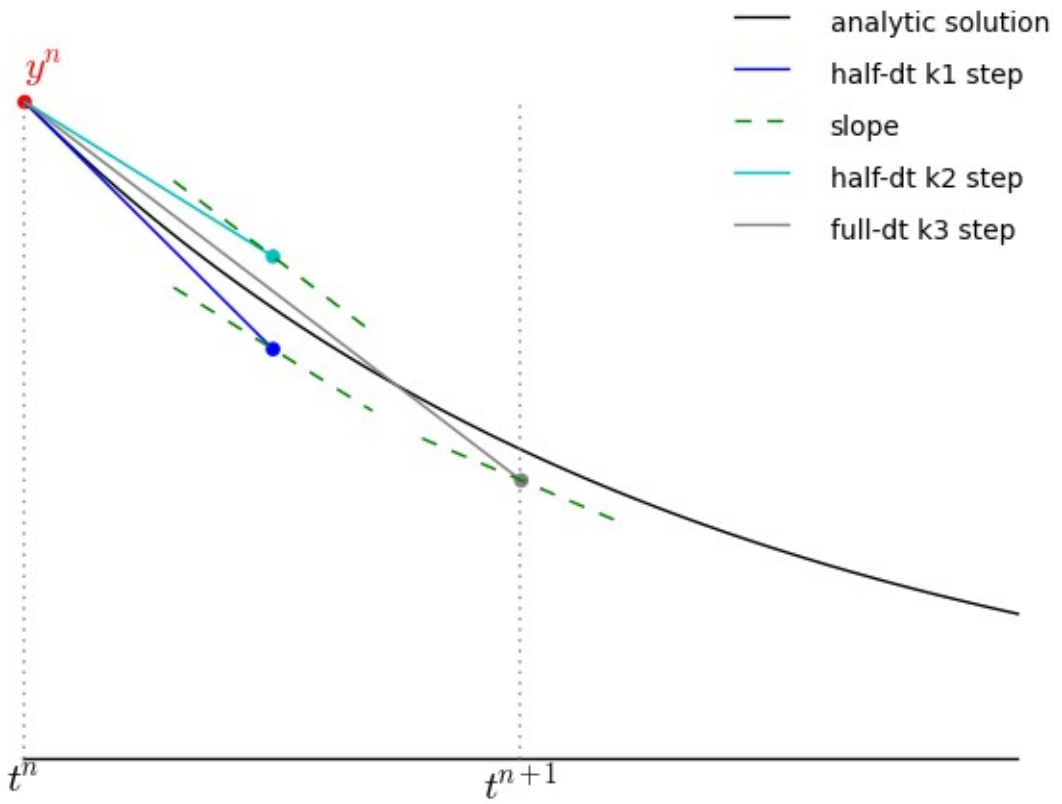
4^{th}-order Runge-Kutta



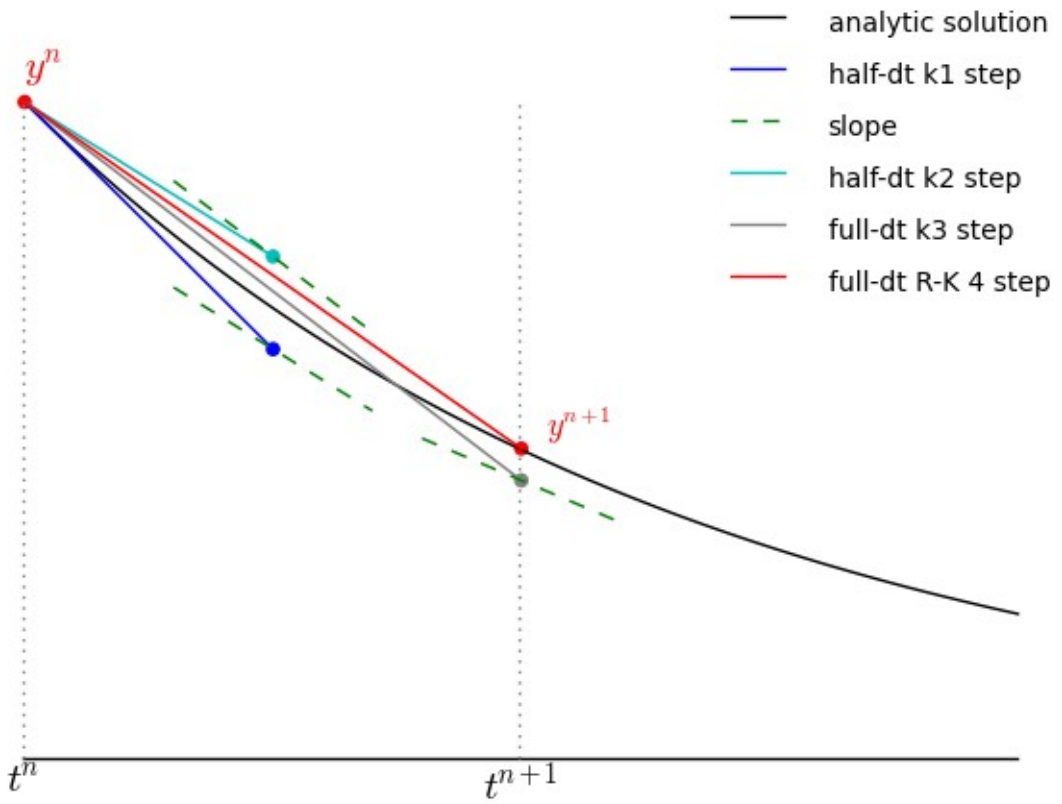
4^{th}-order Runge-Kutta



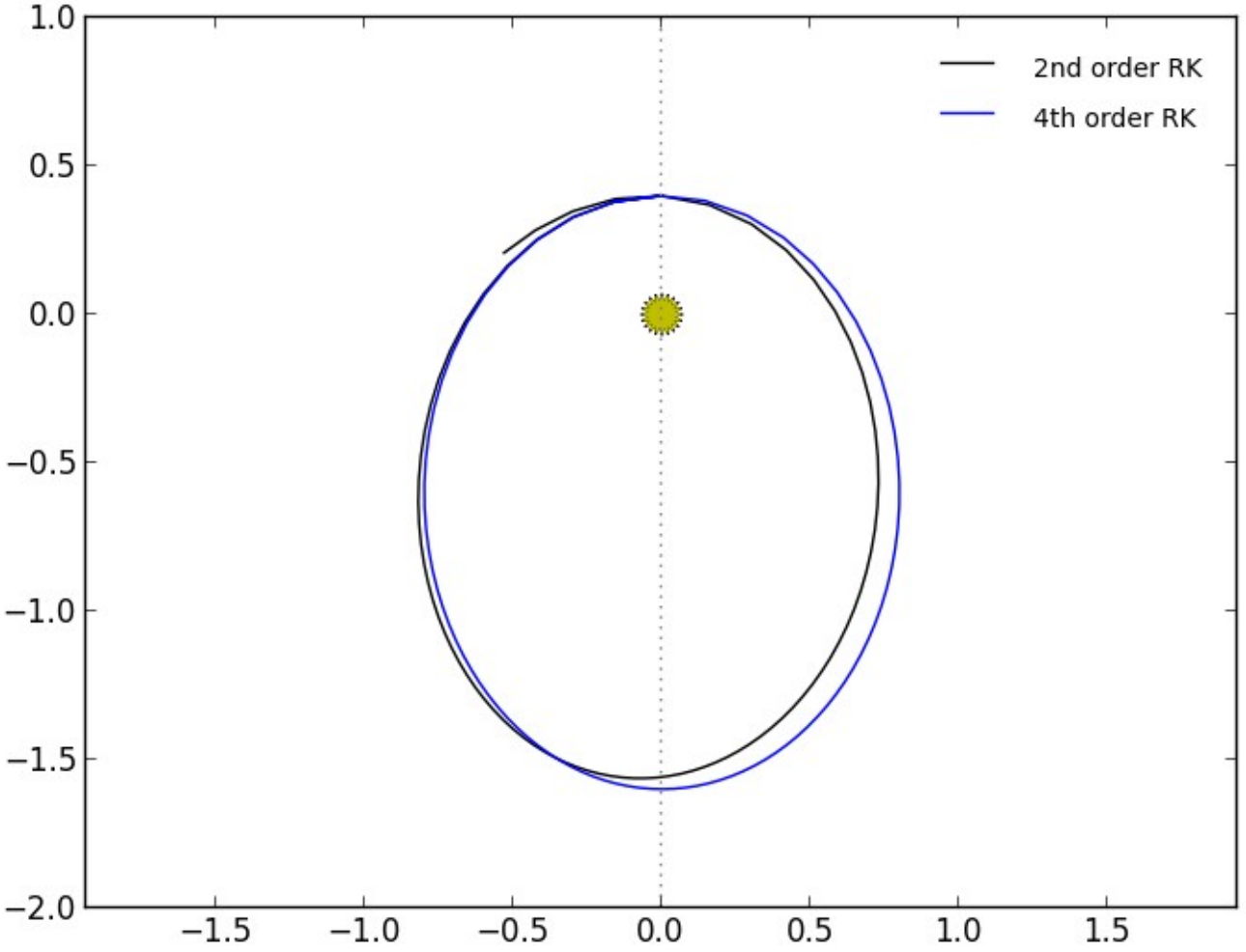
4^{th}-order Runge-Kutta



4^{th}-order Runge-Kutta

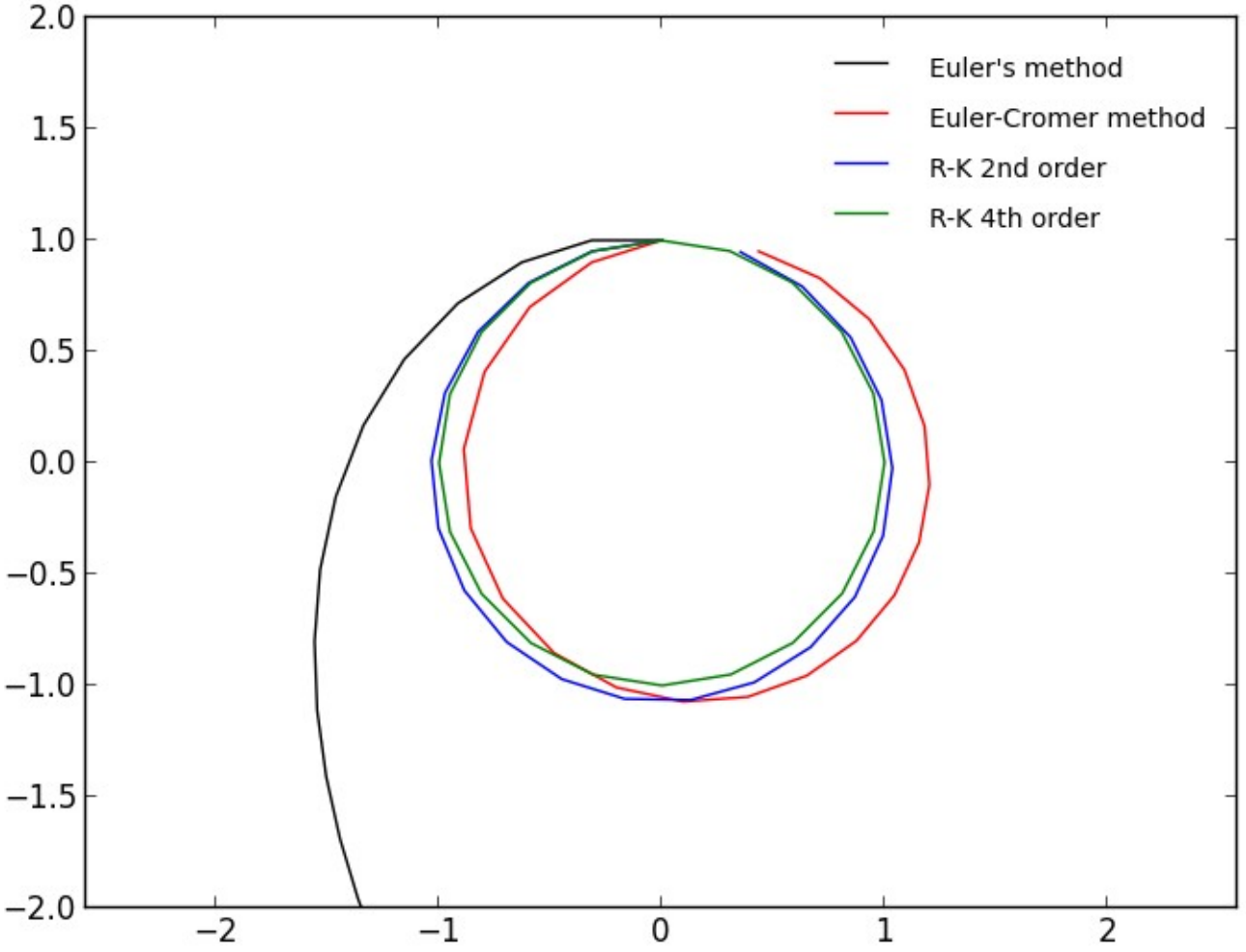


4th-order Runge-Kutta



This looks great! Let's look at the code...

4th-order Runge-Kutta

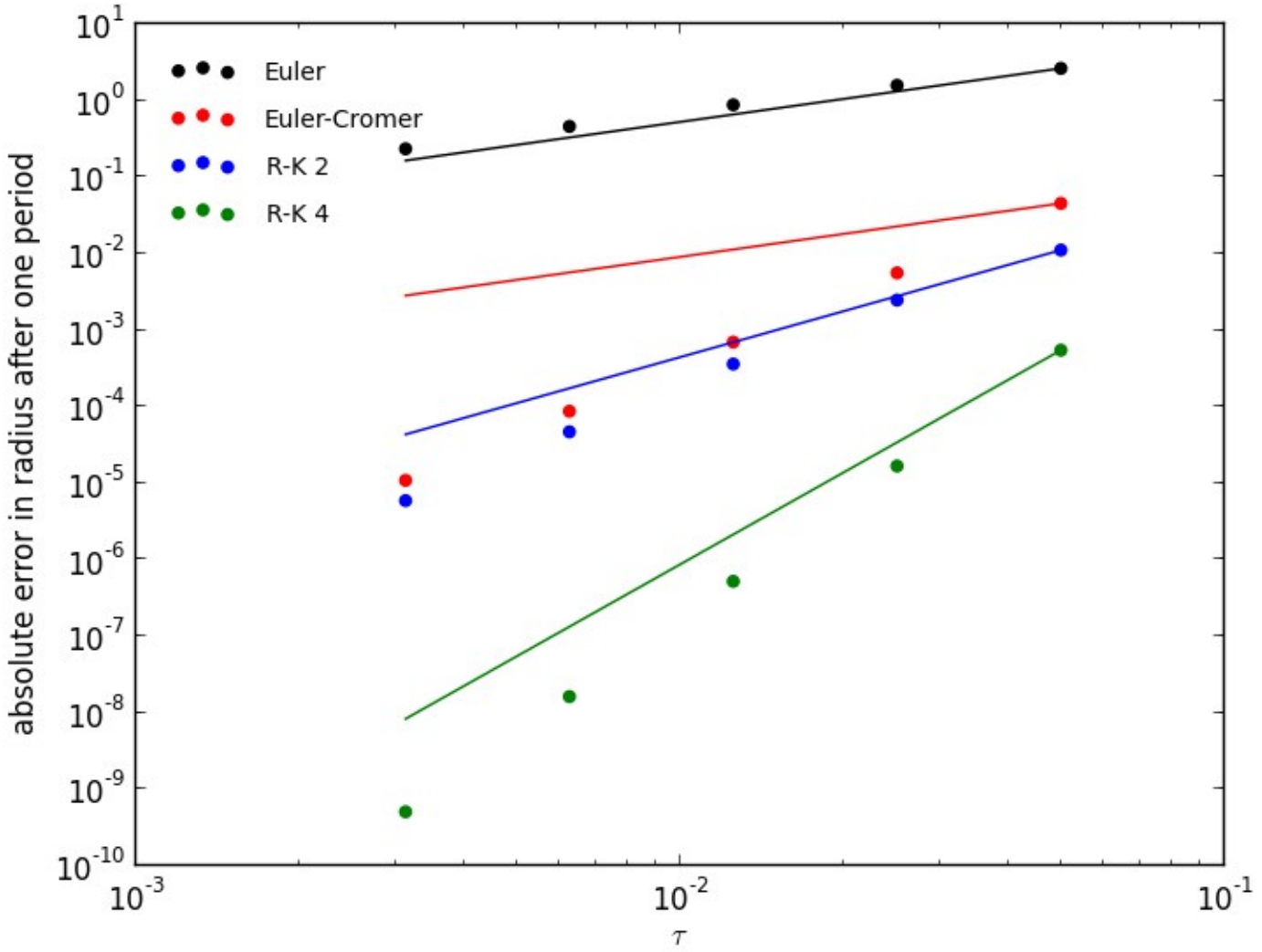


Even with a coarse timestep, RK4 does very well.

Convergence

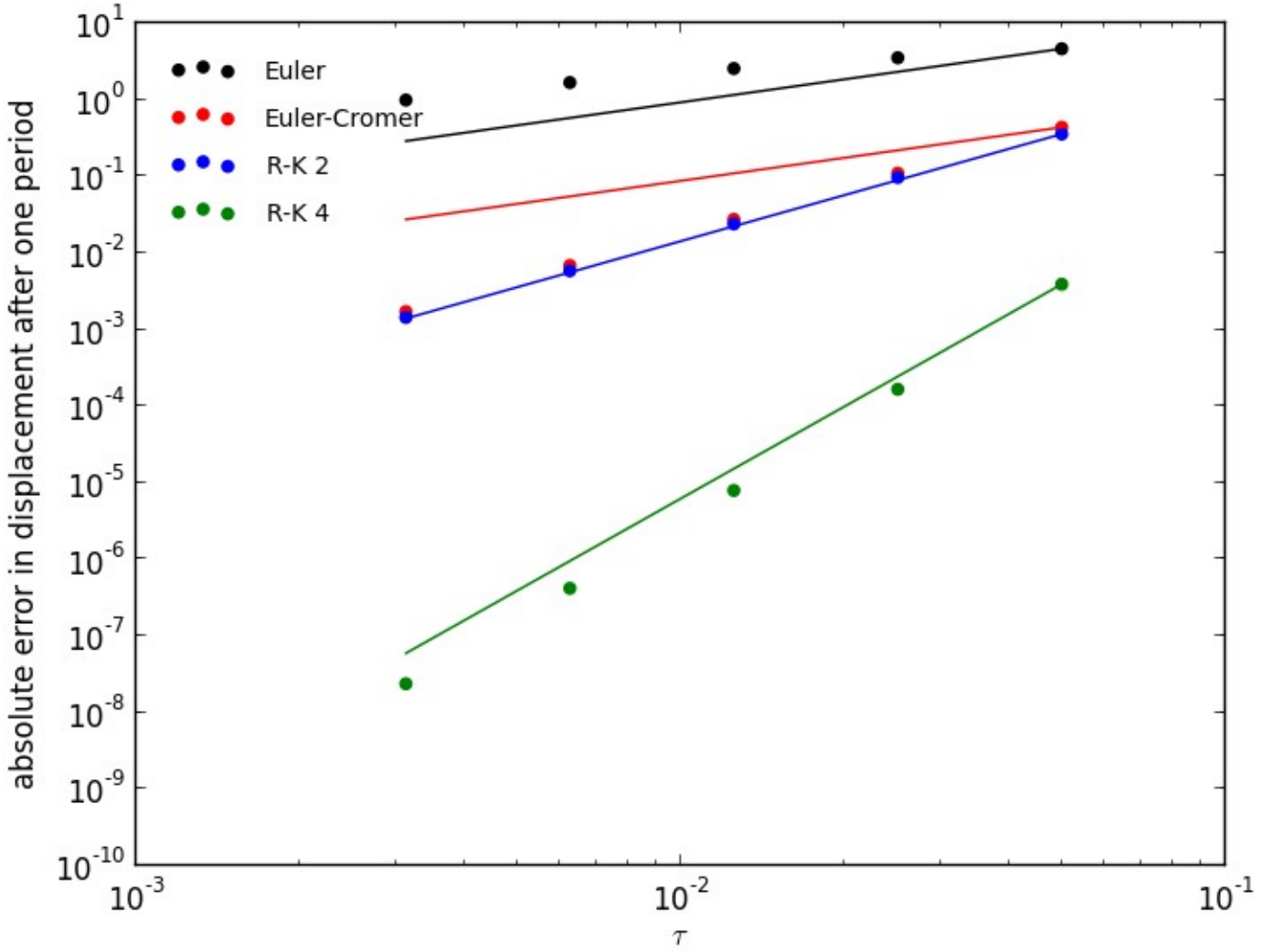
- What do we choose as our convergence criteria?
  - Problem dependent
  - For the orbit, we can consider:
    - Change in radius
    - Distance from start position after one orbit
    - Total energy at the end of integration?
    - ...
- Be careful, because of roundoff, your last step may take you over P—usually you check on the step to ensure it doesn't go past where you want

Convergence



Convergence using the final radius after one period as the error measure. Solid lines indicate 1st, 2nd, and 4th order scaling.

Convergence



Convergence using the net displacement after one period as the error measure. Solid lines indicate 1st, 2nd, and 4th order scaling.

Adaptive Stepping

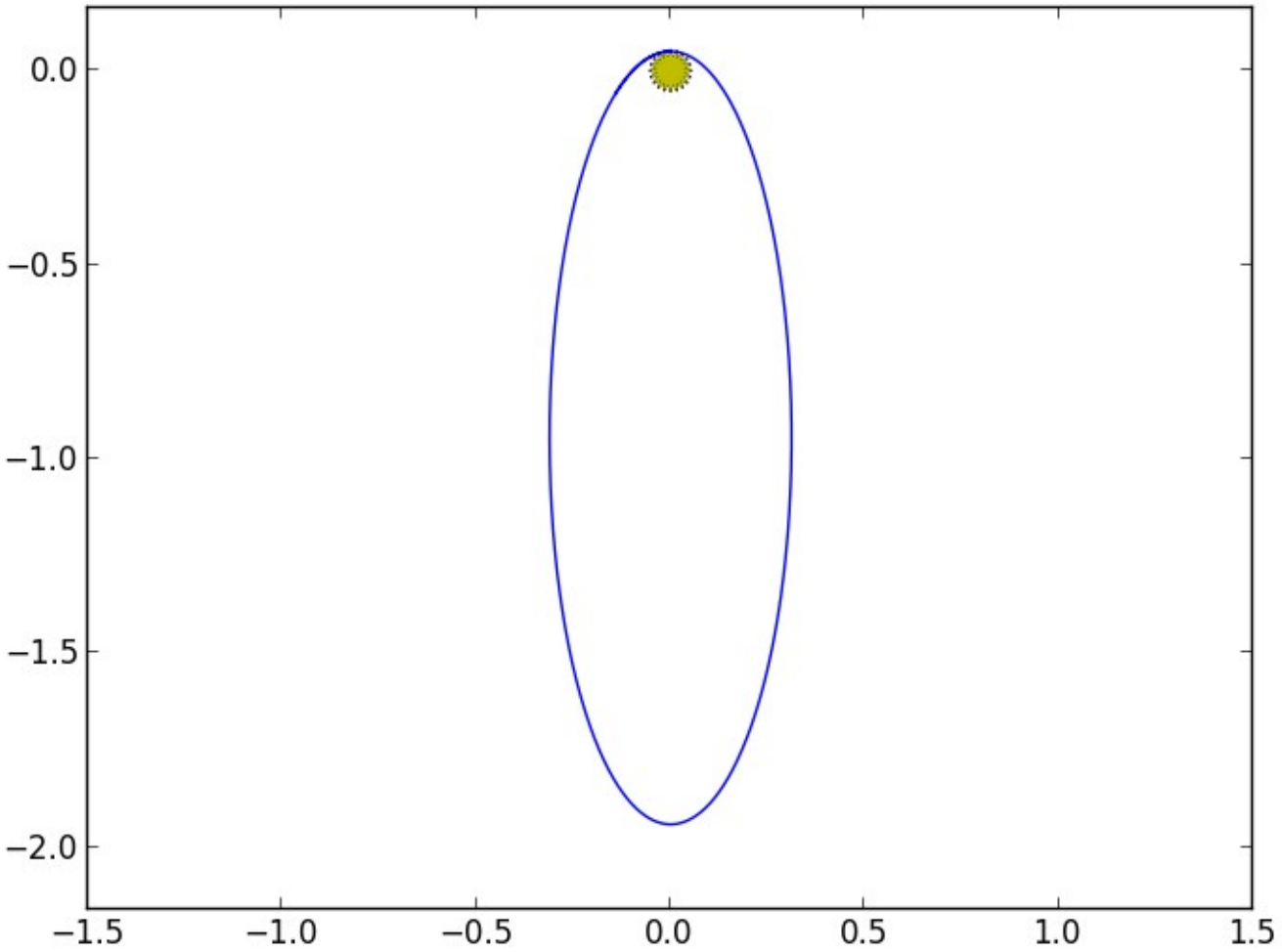
- We need a smaller timestep where the solution changes most rapidly
  - We can get away with large timesteps in regions of slow evolution
- Monitoring the error can allow us to estimate the optimal timestep to reach some desired accuracy
- Lot's of different techniques for this in the literature
  - Take two half steps and compare to one full state
  - Compare higher and lower order methods
- We'll follow the discussion in Garcia—this encompasses the major ideas.

Ex: Highly Elliptical Orbit

- Consider a highly elliptical orbit: a = 1.0, e = 0.95
  - Sun-grazing comet

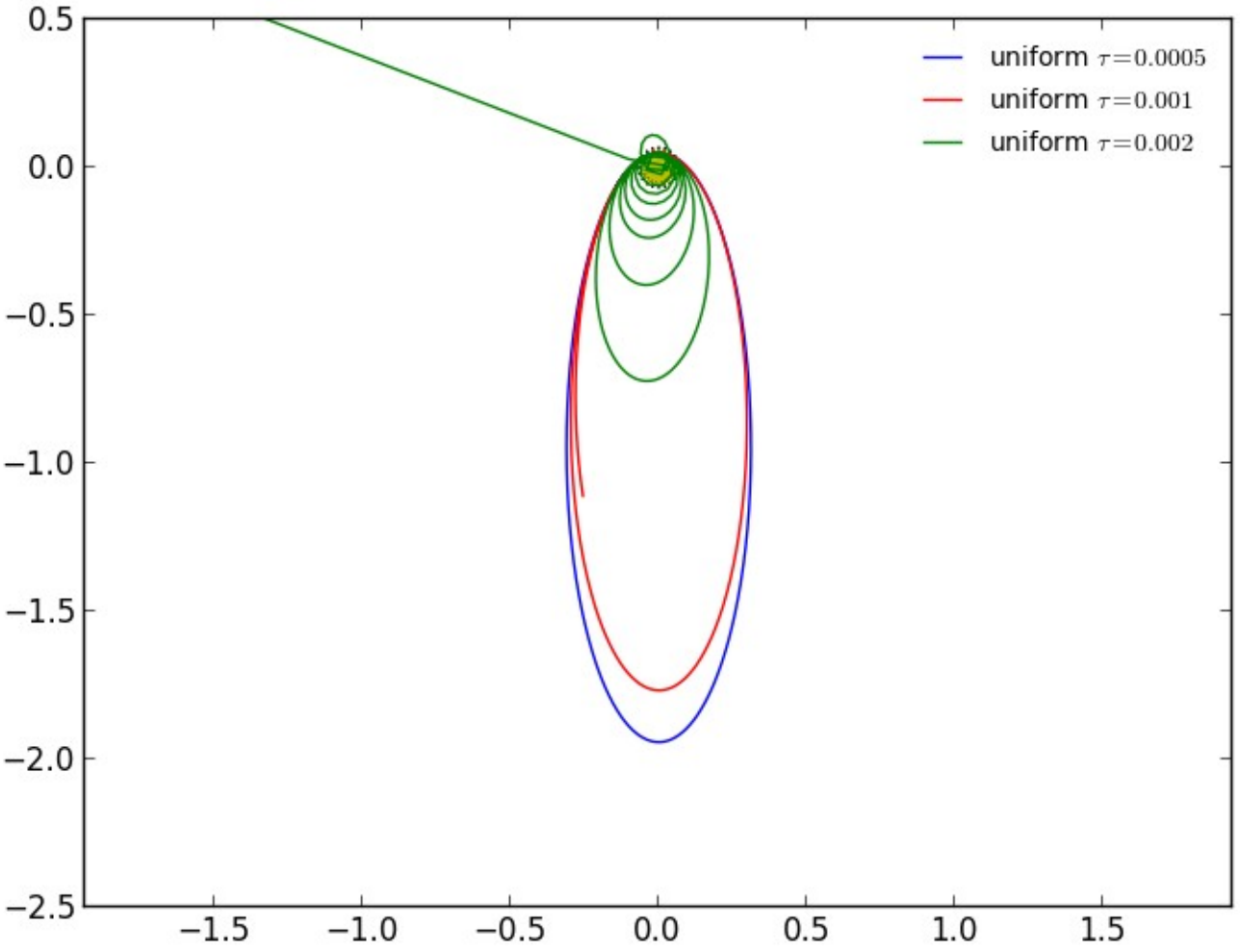
Just to get a reasonable-looking solution, we needed to use $\tau = 0.0005$

This takes 2001 steps



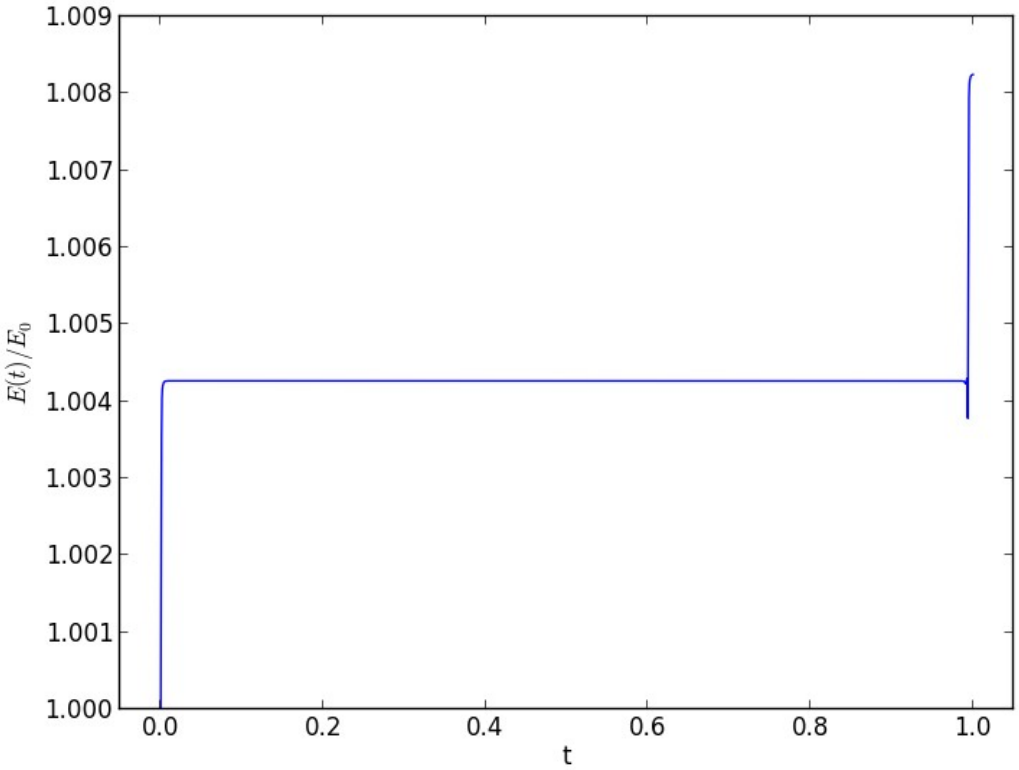
Ex: Highly-Elliptical Orbit

- Solutions with various uniform timesteps



Ex: Highly-Elliptical Orbit

- Look at the total energy
  - At perihelion, conservation is the worse
  - Perihelion is where the velocity is greatest, and therefore the solution changes the fastest
- We can take a larger timestep at aphelion than perihelion



Ex: Highly Elliptical Orbit

Adaptive Stepping

- To do adaptive stepping we need:
  - An error estimate
  - A method to predict a better timestep
- Error estimate: compare solution with single τ step (y_s) to one with two $\tau/2$ steps (y_d)
  - Solution with two smaller steps should be more accurate
  - Relative error is

$$\epsilon_{\rm rel} = \left| \frac{y_d - y_s}{y_d} \right|$$

  - User supplies a desired error, ϵ

Adaptive Stepping

- Timestep correction:
  - Local truncation error of 4th-order Runge-Kutta is ~ τ^5
  - Estimated timestep to meet our desired accuracy is:

$$\tau_{\rm est} = \tau \left(\frac{\epsilon}{\epsilon_{\rm rel}} \right)^{1/5}$$

  - We'll be a little conservative, and use:

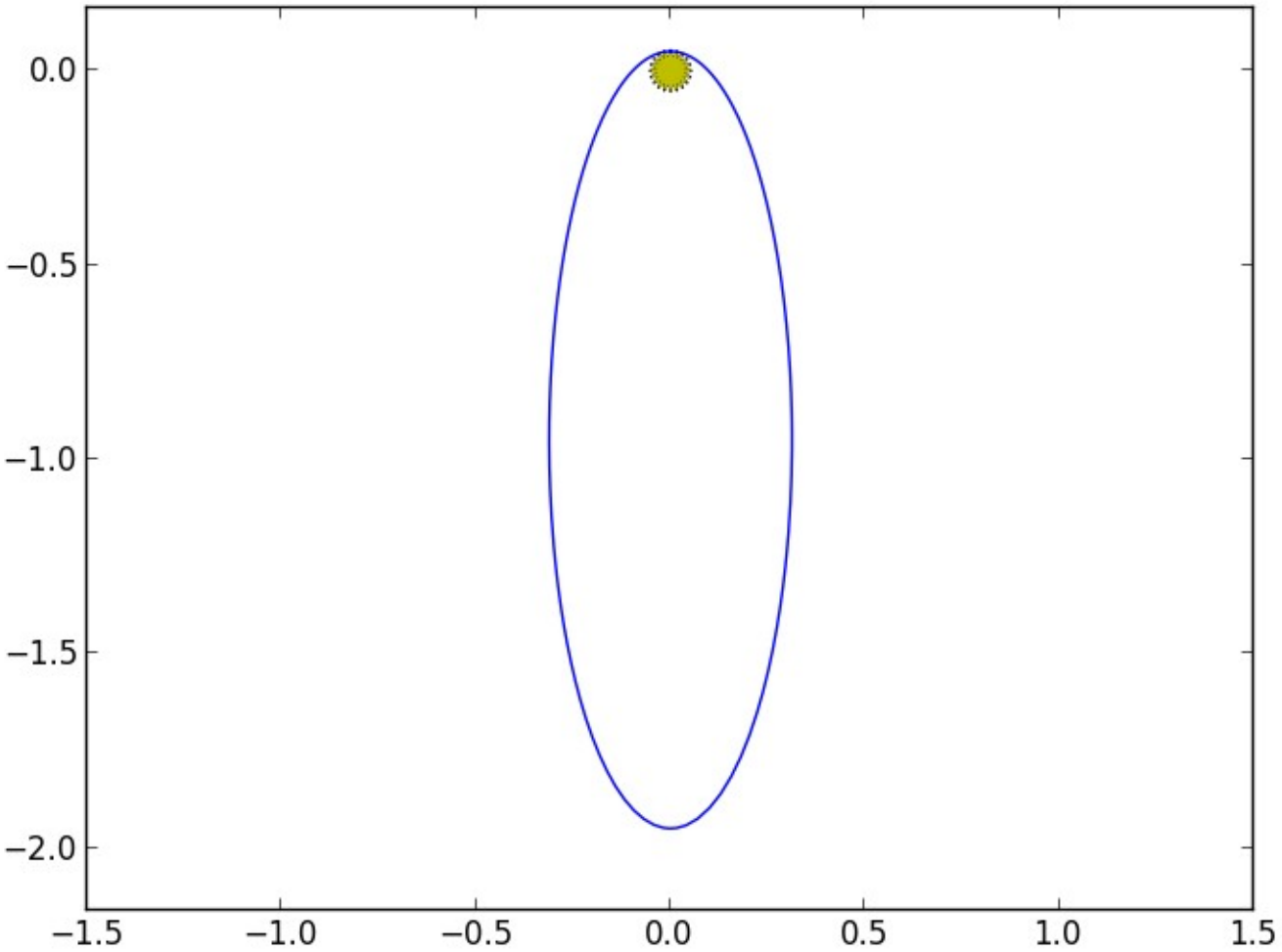
$$\tau_{\rm new} = \min(\max(S_1 \tau_{\rm est}, \tau/S_2), S_2 \tau)$$

- If our step does not meet the desired accuracy, we throw out the new solution, get the new timestep, and redo the step
- Look at the code... and vary tolerances

Adaptive Stepping

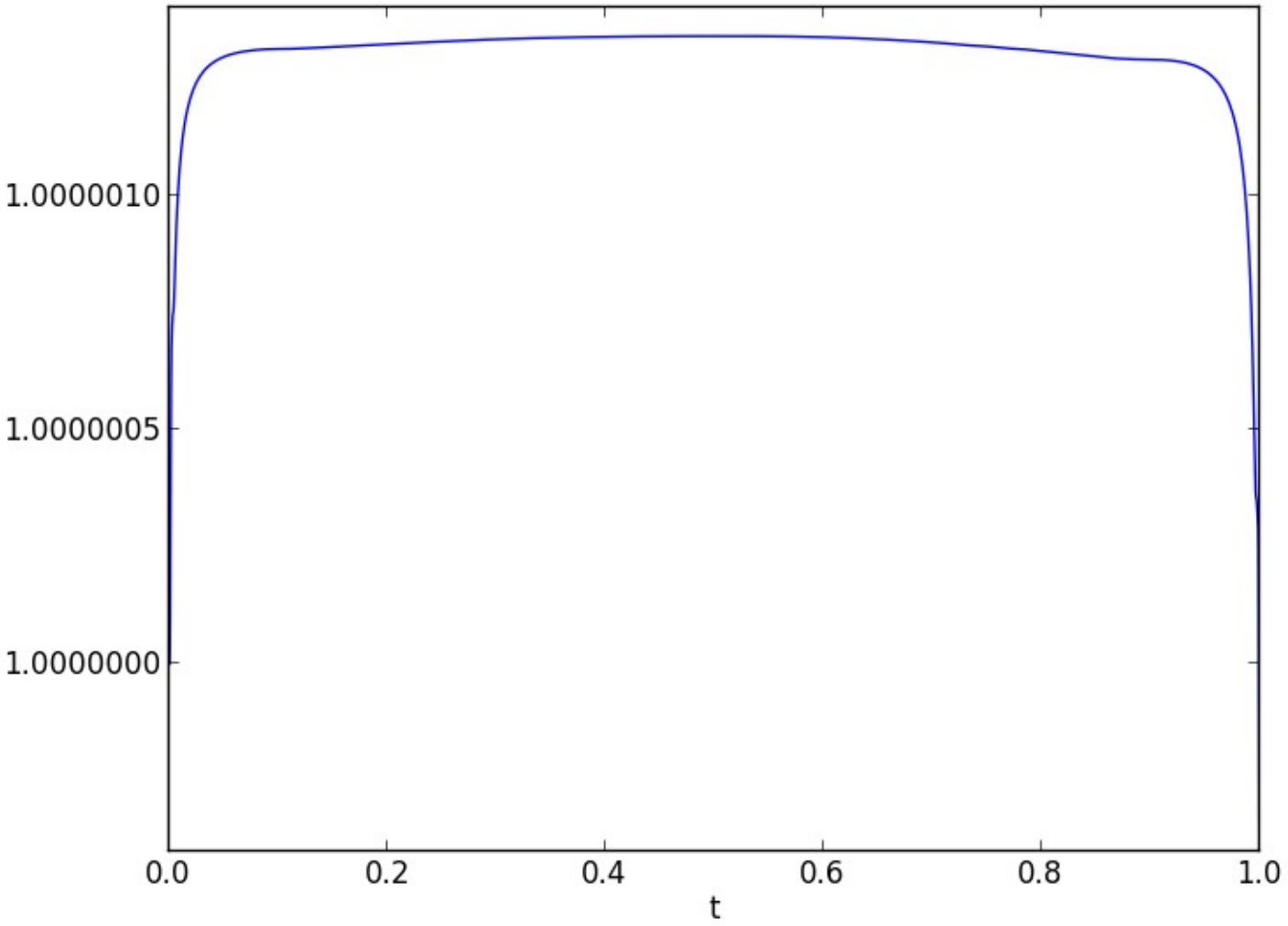
- Adaptive stepping, asking for $\epsilon = 10^{-7}$, with initial timestep the same as the non-adaptive case ($\tau = 0.005$)

This takes only 215 steps



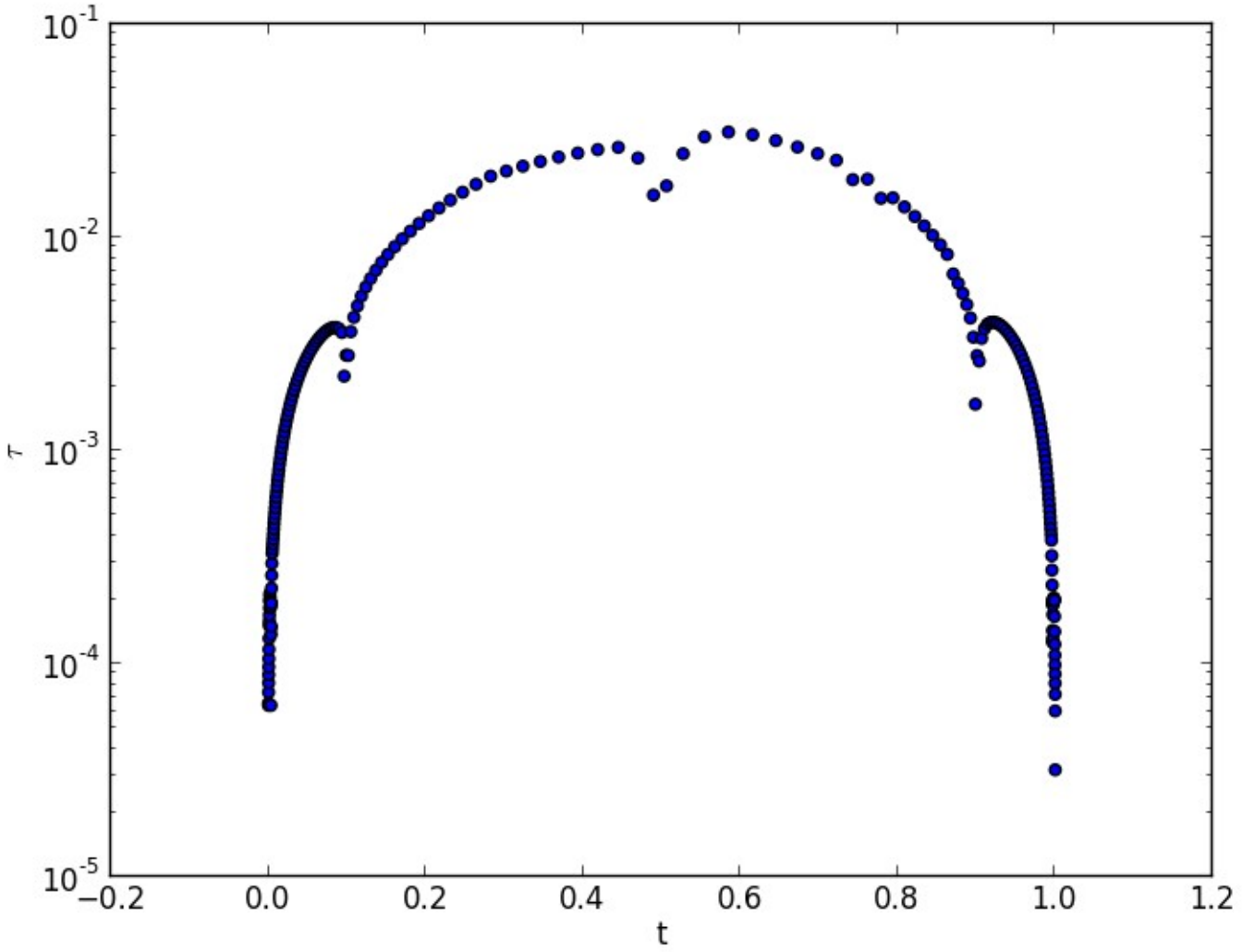
Adaptive Stepping

- Energy conservation is now far superior



Adaptive Stepping

- Timestep varies significantly over the evolution



Adaptive Stepping/Error Estimation

- You should always perform some sort of error estimation
- Specifying absolute and relative errors in the state variables ensures you know about the quality of the solution
- Many ODE packages exist that control all of this for you

Two-Point Boundary Problems

- Consider a Poisson equation:

$$u'' = f$$

  - Second-order: two boundary conditions
  - Take: $u(0) = a$, $u(1) = b$
- Our methods so far don't know how to deal with conditions specified on each boundary
- Later we'll see relaxation methods for this problem
- Now, we'll consider shooting
  - We'll follow the discussion from Pang

Two-Point Boundary Problems

- Rewrite as a system: $y_1 = u,\ y_2 = u'$:

$$y_1' = y_2; \qquad y_2' = f$$

- Left boundary conditions: $y_1(0) = a;\ y_2(0) = \eta$
  - This allows us to integrate the system from 0 to 1
  - What is η ?
    - Parameter that we adjust to make the solution yield $y1(1) = b$ at the end of integration
- Shooting algorithm:
  - Guess η
  - Integrate system to right boundary
  - Use secant method to zero $f(\eta) = b - y_1{}^{(\eta)}(1)$
  - Repeat

Two-Point Boundary Problems

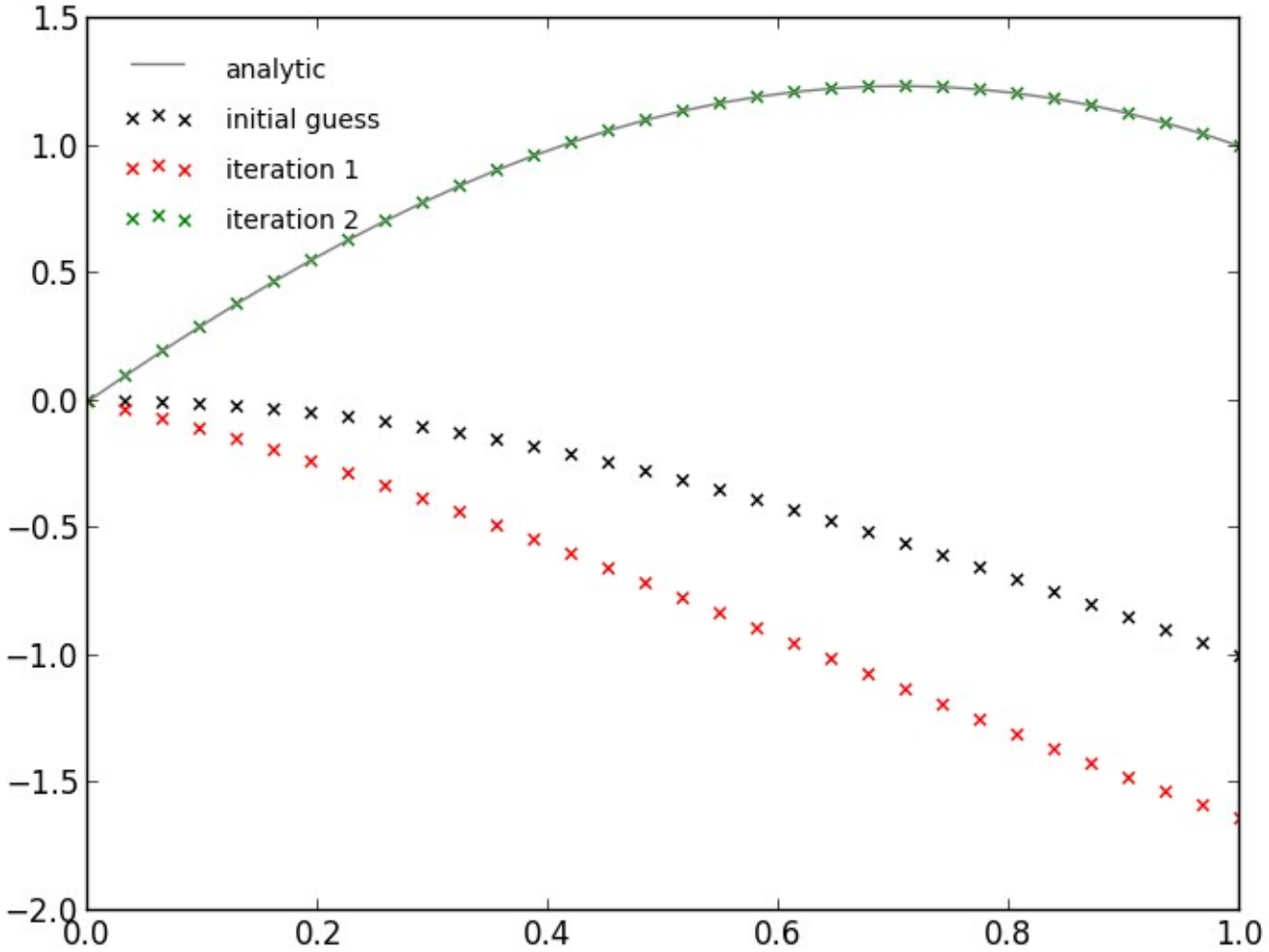
- Example (from Pang):

$$u'' = -\tfrac{1}{4}\pi^2(u+1); \qquad u(0) = 1; \quad u(1) = 1$$

- Analytic solution:

$$u(x) = \cos(\pi x/2) + 2\sin(\pi x/2) - 1$$

Two-Point Boundary Problems



Look at the code...

Two-Point Boundary Problems

- Shooting can work with systems of ODEs
- Commonly used with the equations of stellar structure
  - Central p and T unknown, surface L and R unknown.
  - There we integrate out from the center and in from the surface simultaneously
  - Meet in the middle
  - Adjust parameters to get a match at the middle
  - Iterate

Adams-Bashforth

- Multistep method
  - Uses information from the previous steps
    - (R-K doesn't do this—it takes some intermediate exploratory steps starting always from the same point)
- Adams-Bashforth can require fewer function evaluations per timestep, but need more storage
  - Explicit methods
  - Ex: two-step Adam's Bashforth:

$$y^{n+1} = y^n + \frac{3}{2}\tau f(t^n, y^n) - \frac{1}{2}\tau f(t^{n-1}, y^{n-1})$$

  - Requires some other method to get it started
  - Popular in astrophysics

Stiff Equations / Implicit Methods

- Consider the ODE (example from Byrne & Hindmarsh 1986):

$$\dot{y} = -10^3(y - e^{-t}) - e^{-t}$$

$$y(0) = 0$$

- This has the exact solution:

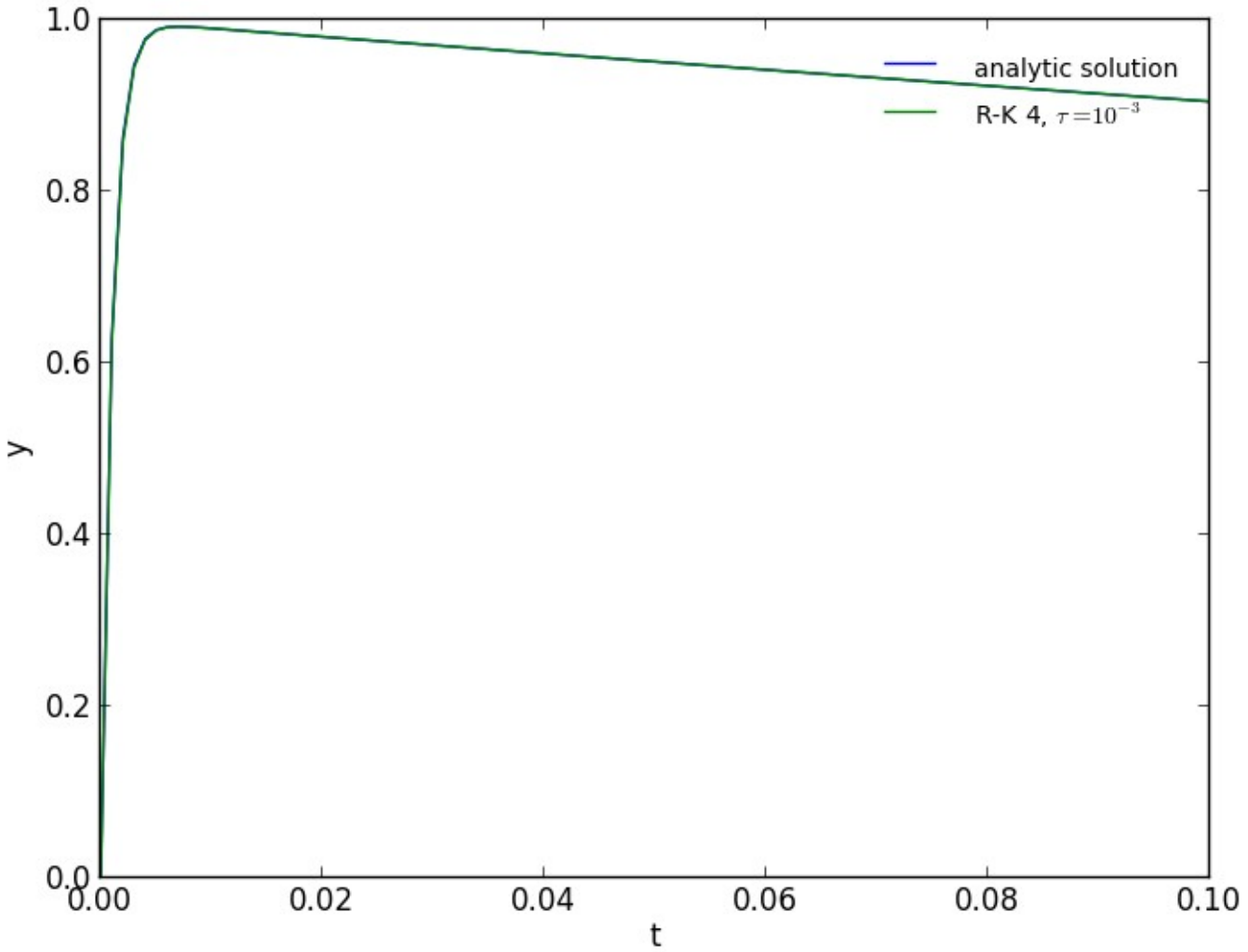
$$y(t) = e^{-t} - e^{-10^3 t}$$

  - Looking at this, we see that there are two characteristic timescales for change, $\tau_1 = 1$ and $\tau_2 = 10^{-3}$
  - A problem with dramatically different timescales for change is called stiff

- Stiff ODEs can be hard for the methods we discussed so far
  - Stability requires that we evolve on the shortest timescale

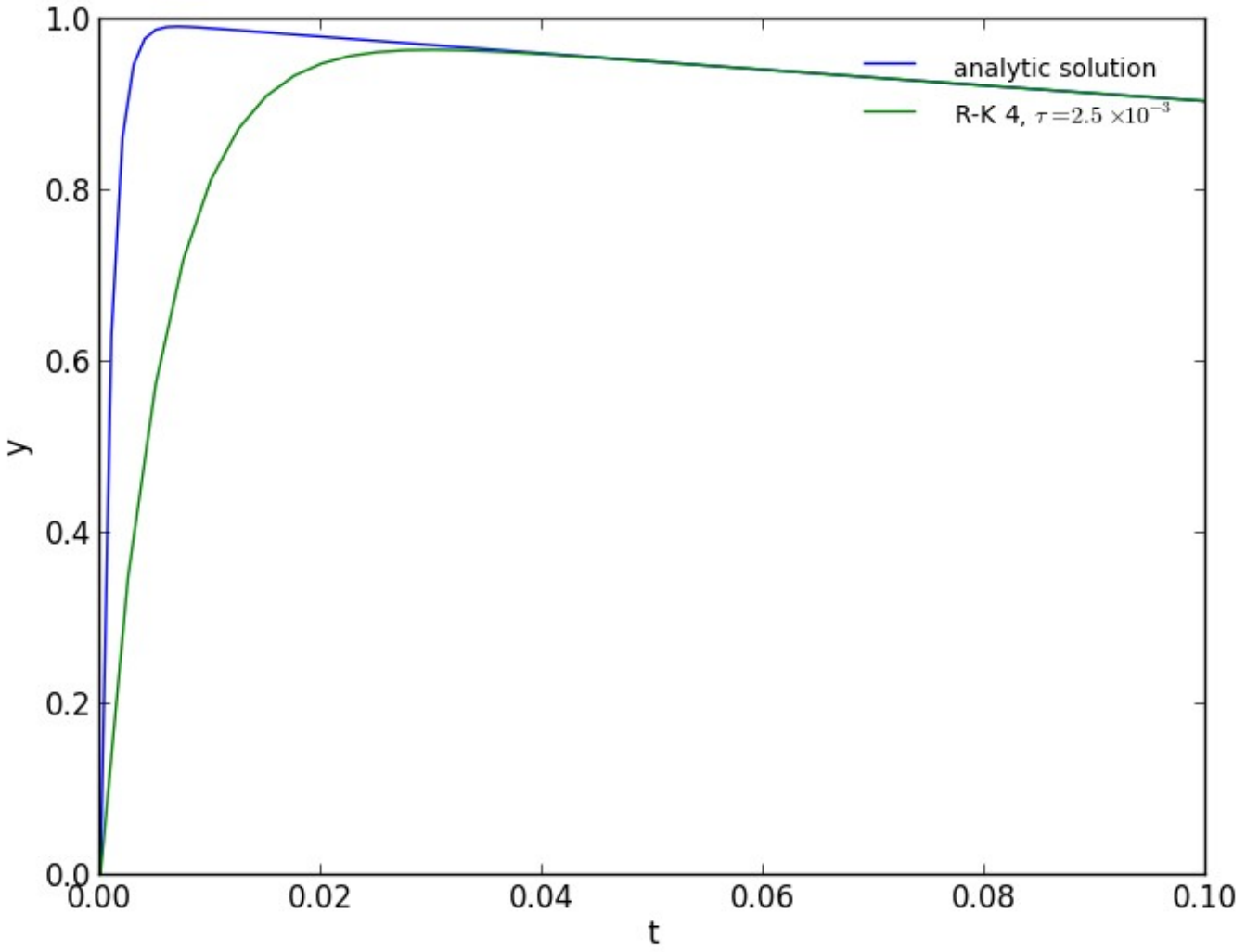
Stiff Equations / Implicit Methods

- 4th order Runge-Kutta solution with timestep $\tau = 10^{-3}$



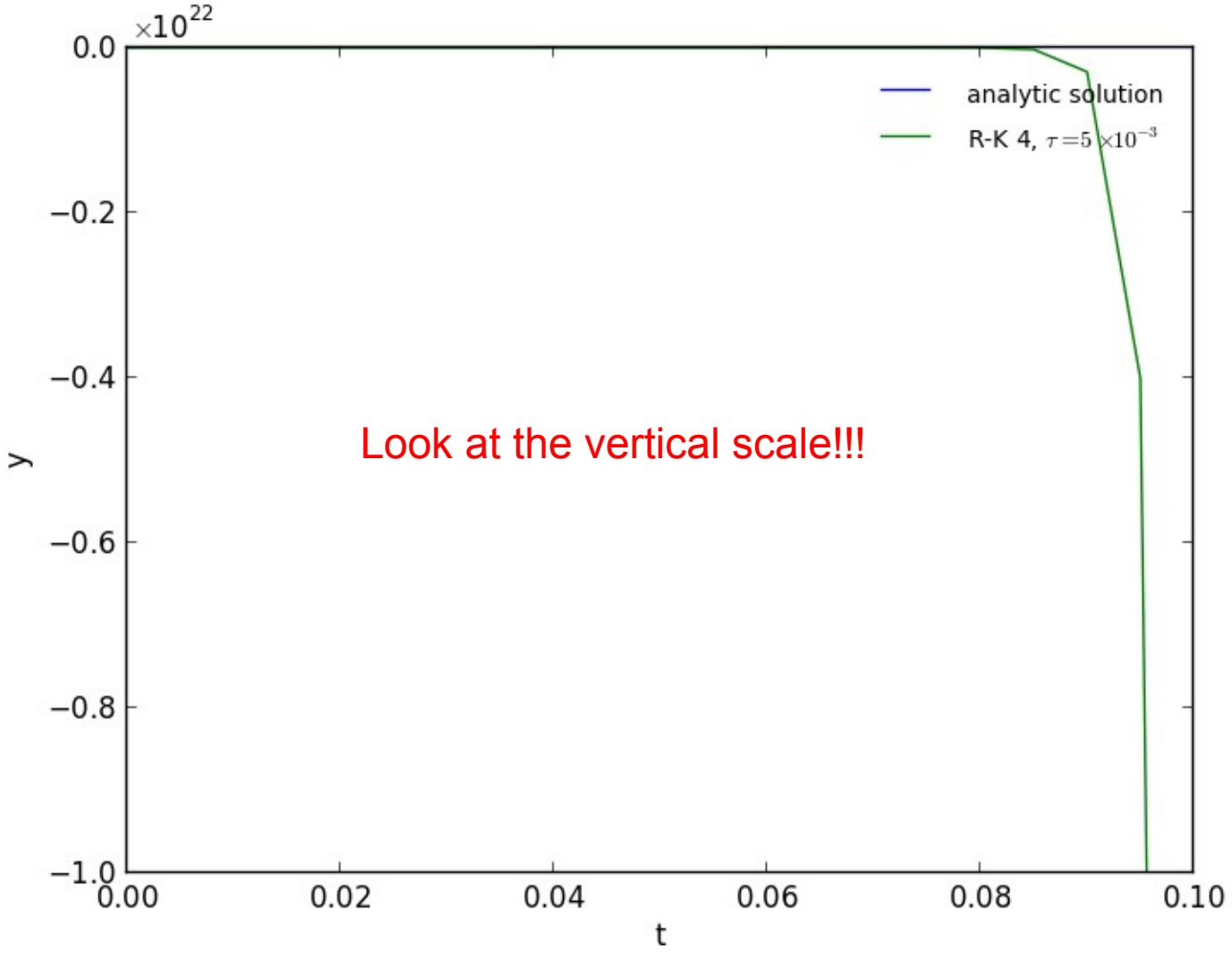
Stiff Equations / Implicit Methods

- 4th order Runge-Kutta solution with timestep $\tau = 2.5 \cdot 10^{-3}$



Stiff Equations / Implicit Methods

- 4th order Runge-Kutta solution with timestep $\tau = 5 \cdot 10^{-3}$



Stiff Equations / Implicit Methods

- 4th-order Runge-Kutta has been our star so far, but it fails miserably
- Consider instead a simple implicit method

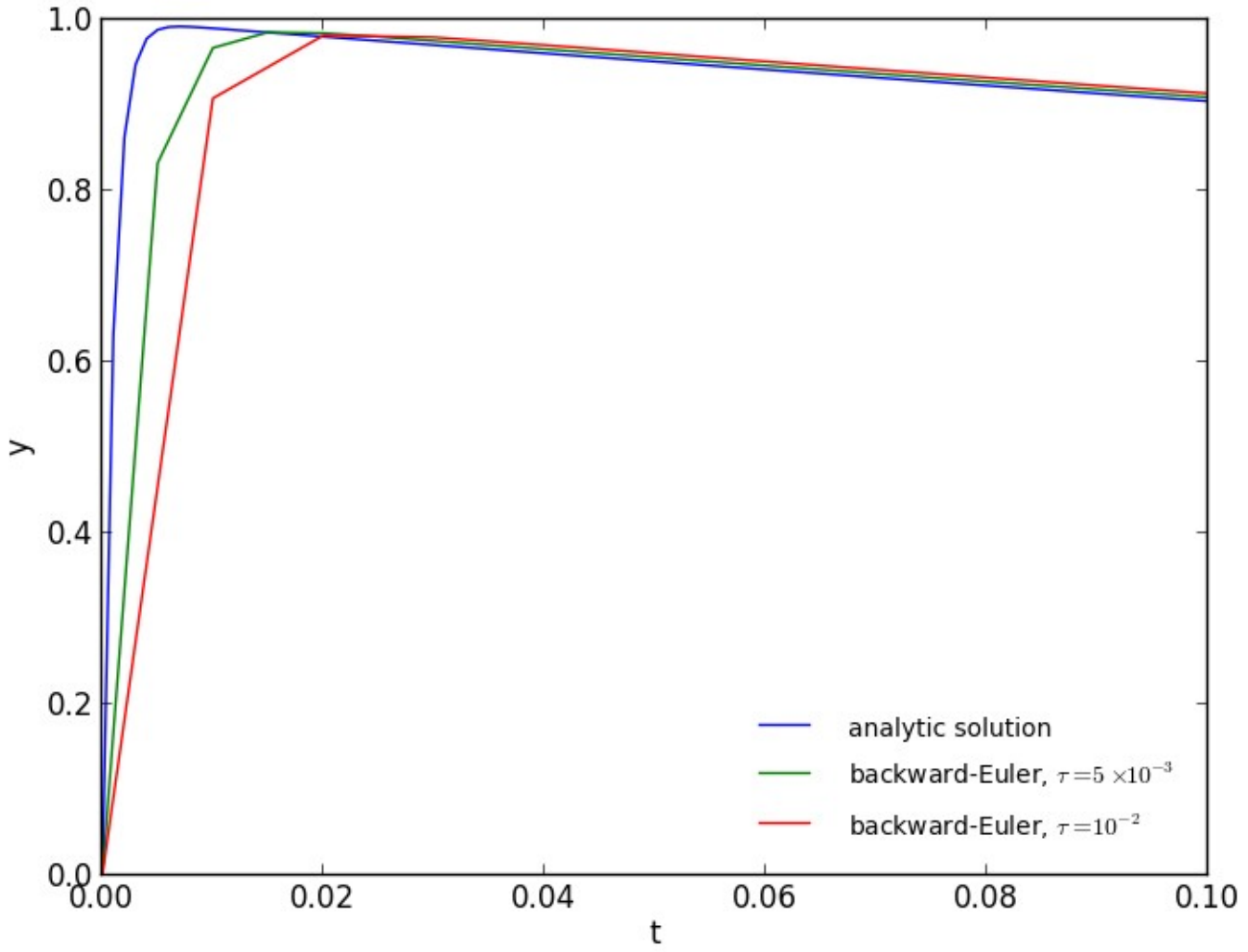
$$y^{n+1} = y^n + \tau \dot{y}^{n+1}$$

  - Here the new state depends on the new state
  - For our model problem, we can solve for the new state algebraically (blackboard):

$$y^{n+1} = \frac{y^n + 10^3 \tau e^{-t} - \tau e^{-t}}{1 + 10^3 \tau}$$

- This method is the backward-Euler or implicit Euler method.
  - It is first-order accurate

Stiff Equations / Implicit Methods



Notice that even with these large timesteps, none of the solutions blow-up!

Stiff Equations / Implicit Methods

- So far we've only been worrying about accuracy
- There is also the concept of stability
  - We'll motivate in a moment that explicitly methods are unstable when applied to stiff problems

Stiff Systems of Equations

- What about a system of equations? Consider the following:

$$\frac{dY_A}{dt} = -\alpha Y_A + \beta Y_B$$

$$\frac{dY_B}{dt} = \alpha Y_A - \beta Y_B$$

  - This is a linear system of ODEs
  - This models reactive flow with forward and inverse reactions (system from F. Timmes summer school notes)
  - In matrix form:

$$\frac{d}{dt}\begin{pmatrix} Y_A \\ Y_B \end{pmatrix} = \begin{pmatrix} -\alpha & \beta \\ \alpha & -\beta \end{pmatrix}\begin{pmatrix} Y_A \\ Y_B \end{pmatrix}$$

  - Or:

$$\dot{Y} = AY$$

Stiff Systems of Equations

- This problem has the solution:

$$\frac{Y_B}{Y_A} = \frac{e^{(\alpha+\beta)t} - 1}{\frac{\beta}{\alpha}e^{(\alpha+\beta)t} + 1}$$

  for $Y_B(0) = 0$

  - Characteristic timescale for change here is $t = 1/(\alpha + \beta)$
  - Long term behavior: $Y_B/Y_A \rightarrow \alpha/\beta$
  - Also, $Y_A + Y_B = 1$ and $d(Y_A + Y_B)/dt = 0$

Stiff Systems of Equations

- Backward-Euler discretization of this system:

$$Y^{n+1} = Y^n + \Delta t A Y^{n+1}$$

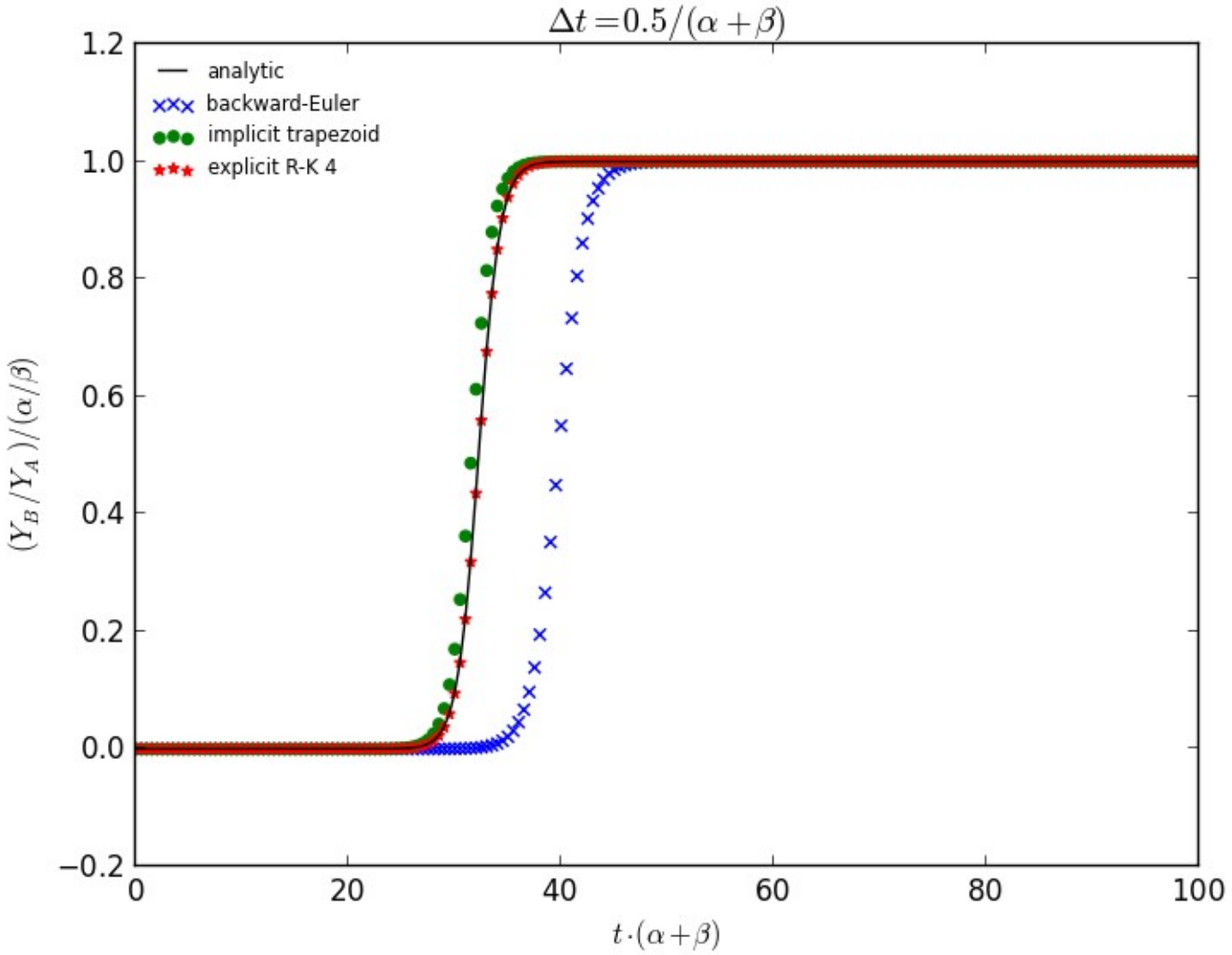
- Solving for the new state:

$$Y^{n+1} = \underbrace{(I - \Delta t A)^{-1}}_{\text{This is a matrix inverse}} Y^n$$

  - Alternately, we can write this as a linear system:

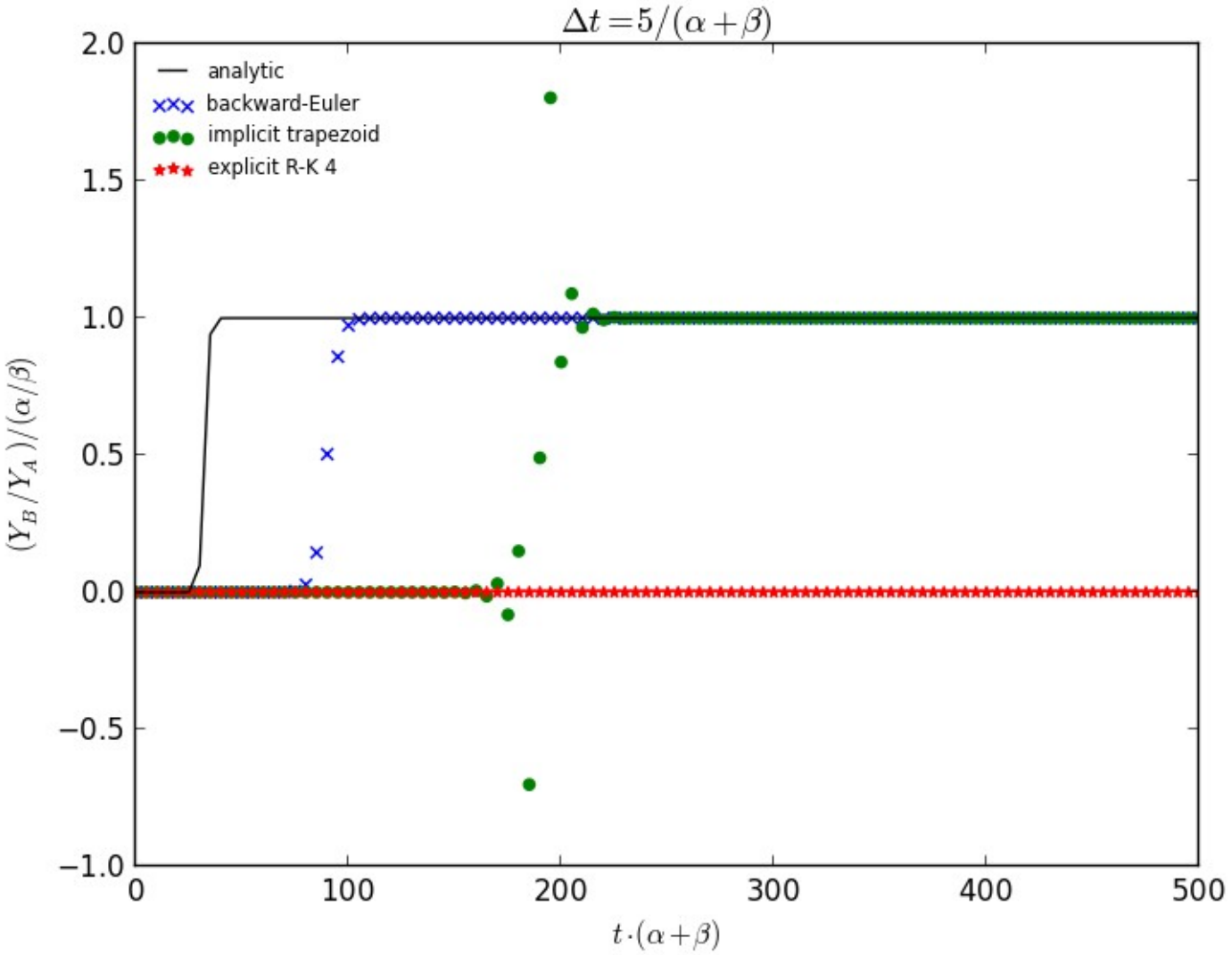
$$\underbrace{(I - \Delta t A) Y^{n+1} = Y^n}_{\text{Analogous to “Ax = b”}}$$

Stiff Systems of Equations



Notice the R-K 4 method does well when we step at < characteristic timescale of change

Stiff Systems of Equations

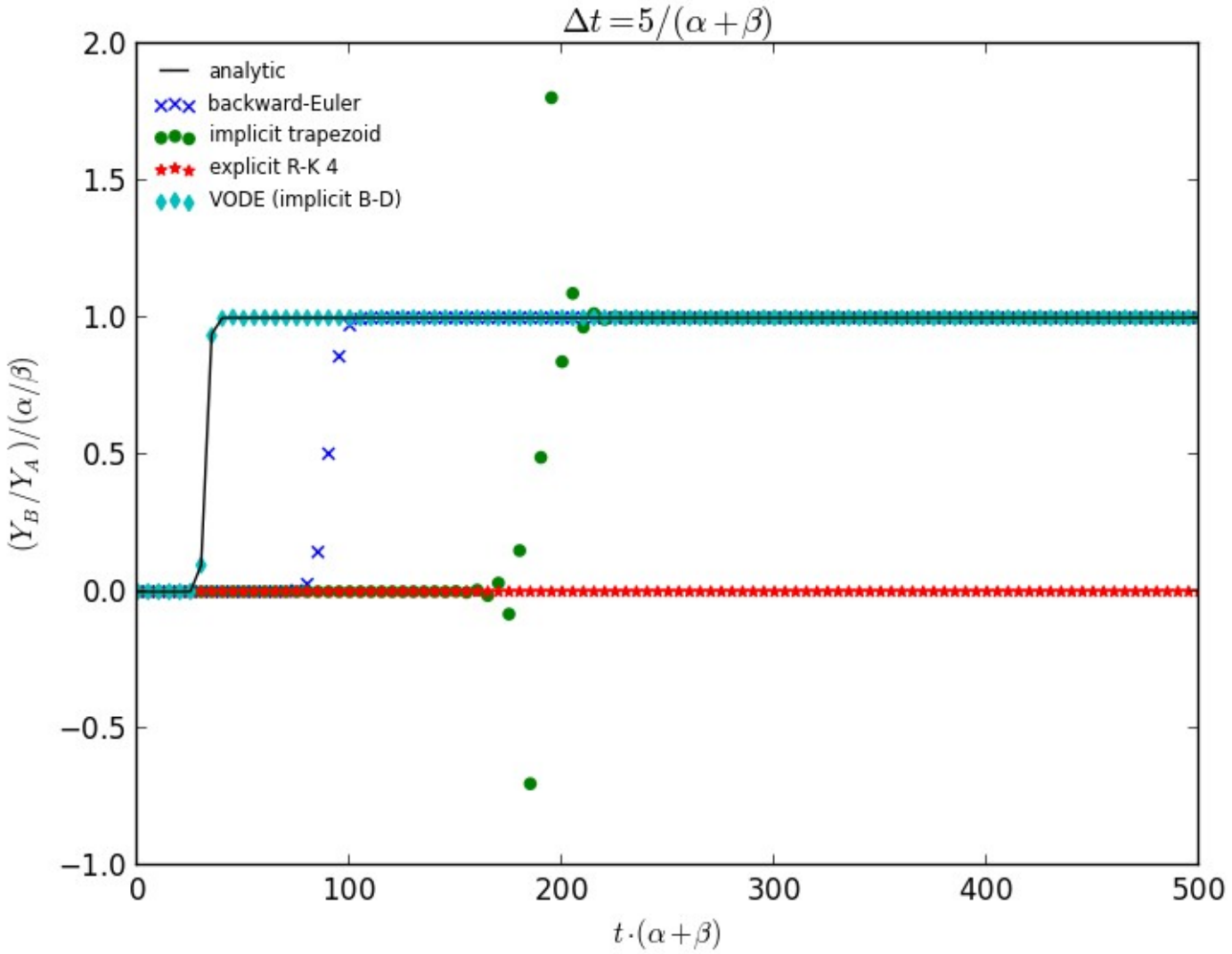


At 5× the characteristic timescale, R-K 4 fails completely. Backward-Euler still has a large error (it's only first-order), but remains stable.

VODE

- For stiff systems, robust integration packages exist.
- VODE is a popular one (F77, but also in python/scipy)
  - Uses a 5^{th}-order implicit method, with error estimation (alternately, a 15^{th} order explicit method)
  - Can either use a user-supplied Jacobian or compute one numerically
  - Two types of tolerances: absolute and relative—these have a big influence on your solution
  - Will do adaptive timesteps to reach your specified stop time

VODE w/ Linear Stiff System



Variable timestep methods like VODE should do well here, since the solution is flat for most of the time, allowing larger timesteps.

Non-linear Stiff Systems

- What about a non-linear system of equations?

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y})$$

  - Expand about current solution (linearize)

$$\mathbf{f}(\mathbf{y}^{n+1}) = \mathbf{f}(\mathbf{y}^n) + \underbrace{\left.\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right|_n}_{\text{This is the Jacobian}} (\mathbf{y}^{n+1} - \mathbf{y}^n) + \ldots$$

  - This gives (for Backward-Euler):

$$\mathbf{y}^{n+1} = \mathbf{y}^n + \tau \mathbf{f}(\mathbf{y}^{n+1}) = \mathbf{y}^n + \tau \left[\mathbf{f}(\mathbf{y}^n) + \mathbf{J}(\mathbf{y}^{n+1} - \mathbf{y}^n)\right]$$

  - Which is the linear system:

$$(\mathbf{I} - \tau \mathbf{J})\mathbf{y}^{n+1} = (\mathbf{I} - \tau \mathbf{J})\mathbf{y}^n + \tau \mathbf{f}(\mathbf{y}^n)$$

VODE w/ Non-linear Stiff System

- Example from chemical kinetics (see, ex. Byrne & Hindmarsh 1986, or the VODE source code):

$$\frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} -0.04y_1 & +10^4 y_2 y_3 & \\ 0.04y_1 & -10^4 y_2 y_3 & -3\times 10^7 y_2^2 \\ & & 3\times 10^7 y_2^2 \end{pmatrix}$$

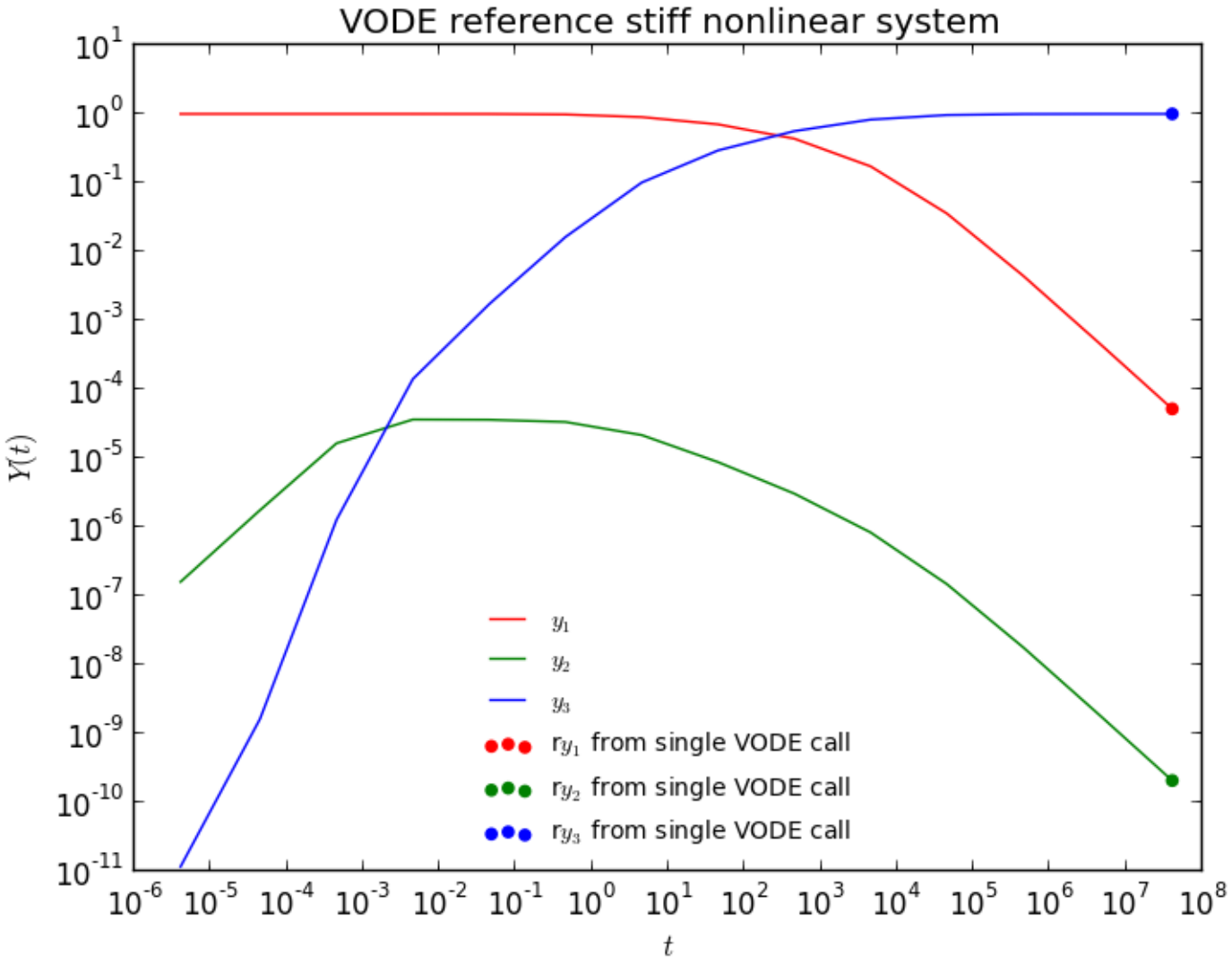
$$\mathbf{J} = \begin{pmatrix} -0.04 & 10^4 y_3 & 10^4 y_2 \\ 0.04 & -10^4 y_3 - 6\times 10^7 y_2 & -10^4 y_2 \\ 0 & 6\times 10^7 y_2 & 0 \end{pmatrix}$$

- Start with $y_1(0) = 0, y_2(0) = y_3(0) = 0$
- Long term behavior: $y_1, y_2 \to 0, y_3 \to 1$
- Although y_2 is initially 0, it will build up quickly, feeding the creation of y_3

VODE w/ Non-linear Stiff System

VODE called in two different passes here. The first, to generate the continuous lines, started with a small timestep, called VODE, and increased the timestep by 10× each subsequent call.

The second called VODE only once for the entire evolution, and had VODE internally chose timesteps. This results in the end points on each line.



Look at the code...

Practical Definition of Stiff

- Common definition: look at the eigenvalues of the Jacobian. If

$$\frac{\max |\lambda_k|}{\min |\lambda_k|} \gg 1$$

  and the eigenvalue are negative, then the problem is called stiff (see, e.g. LeVeque Ch. 8)

  - This definition doesn't apply to scalar problems—but we already saw the notion of two distinct timescales

- More rigorous (Shampine & Gear 1979, as discussed in Byrne & Hindmarsh 1986)

  - "By a stiff problem we mean one for which no solution component is unstable (no eigenvalue [of the Jacobian matrix] has a real part which is at all large and positive) and at least some component is very stable (at least one eigenvalue has a real part which is large and negative). Further, we will not call a problem stiff unless its solution is slowly varying with respect to the most negative part of the eigenvalues...Consequently a problem may be stiff for some intervals and not for others."

Quick Discussion of Stability

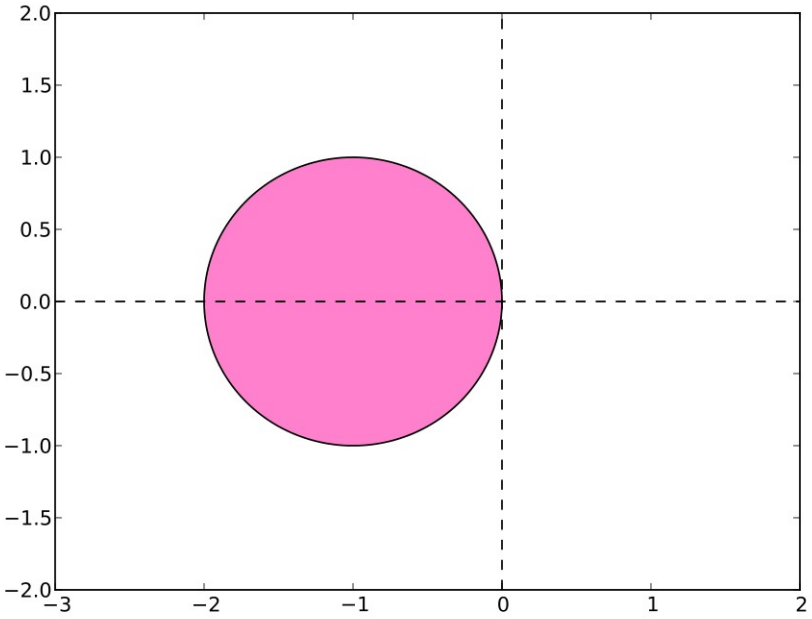
- A very nice discussion of stability for ODEs is in LeVeque, “Finite Different Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems”
  - Following Leveque...
- We are concerned with absolute stability
  - Consider linear problem: $u'(t) = \lambda\, u(t)$
  - Euler's method: $u^{n+1} = (1 + \tau\lambda)\, u^n$
  - Absolute stability requires:

$$|1 + \tau\lambda| \leq 1$$

  - Typically take $z = \tau\lambda$ and look in the complex plane
- Method is call A-stable if stability region includes the entire left half of the complex plane

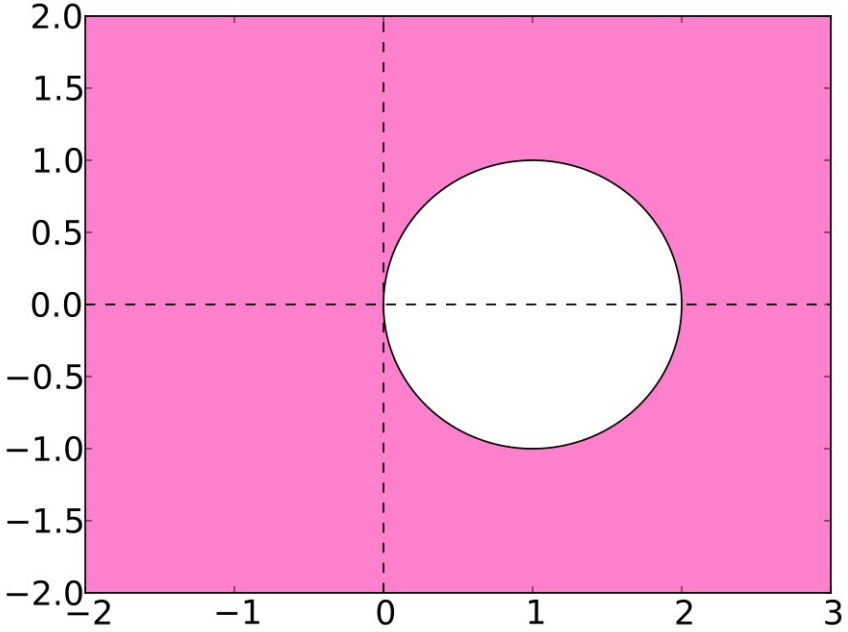
Quick Discussion of Stability

- Regions of absolute stability for foward and backward Euler



(Helmut Podhaisky/Jitse Neisen//Wikipedia)

Stablity region for explicit Euler. Note that if λ is large and negative, then τ needs to be correspondingly small to be stable



Stablity region for backward (implicit) Euler. Note that the entire left half of the complex plane is included.