Overview

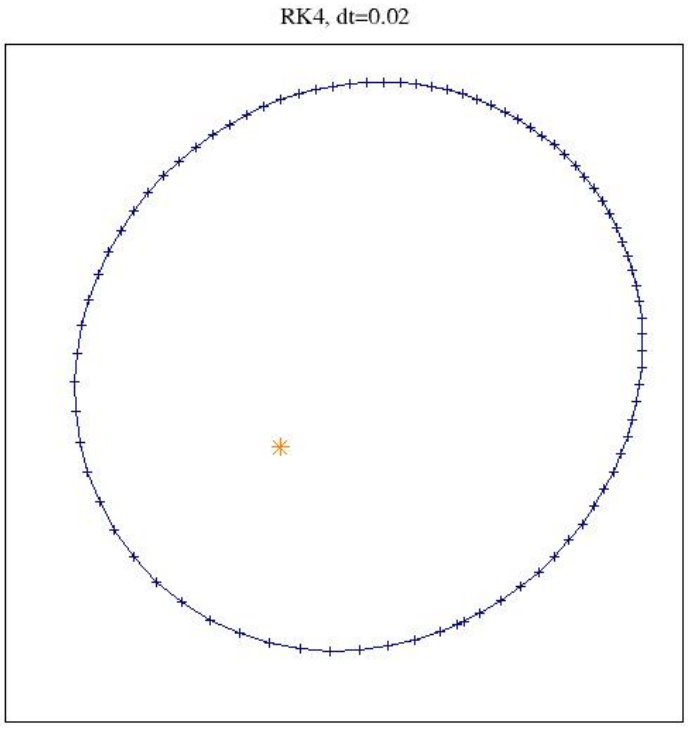
Recap

Runge–Kutta methods: systematic improvement in accuracy

- Midpoint method = second order Runge–Kutta
- Fourth-order Runge–Kutta very widely used
- Higher order used in adaptive stepsize algorithms

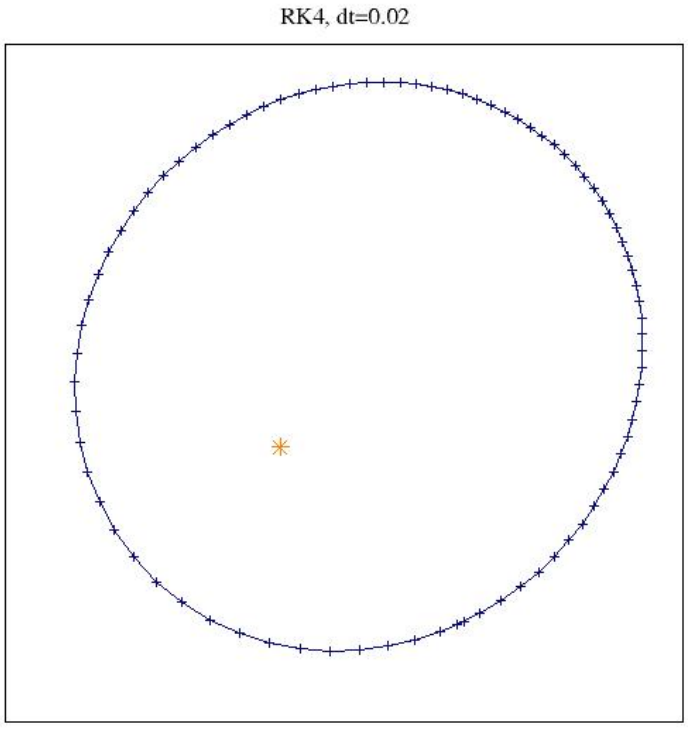
Slow vs fast motion

Example: Kepler problem



Slow vs fast motion

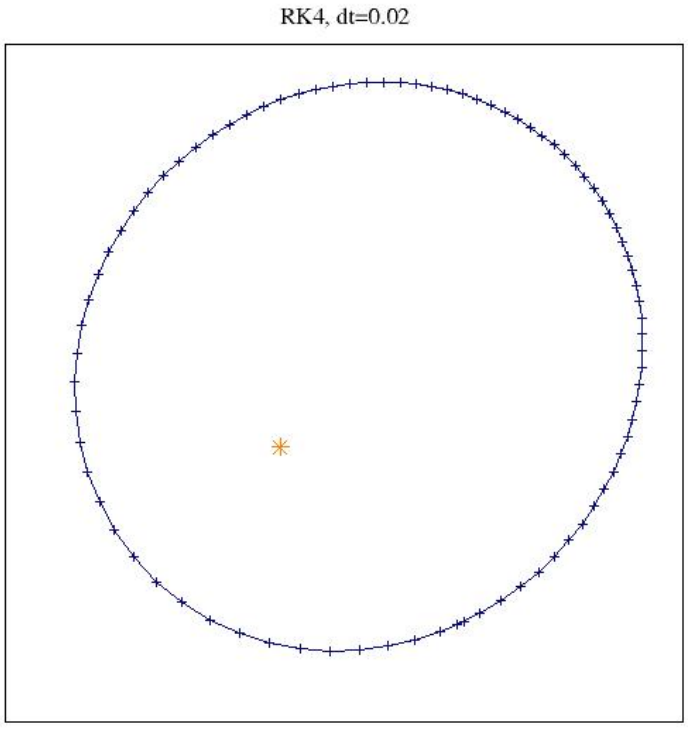
Example: Kepler problem



- Near perihelion: fast motion
- $r(t), \theta(t)$ vary rapidly
- need small time steps

Slow vs fast motion

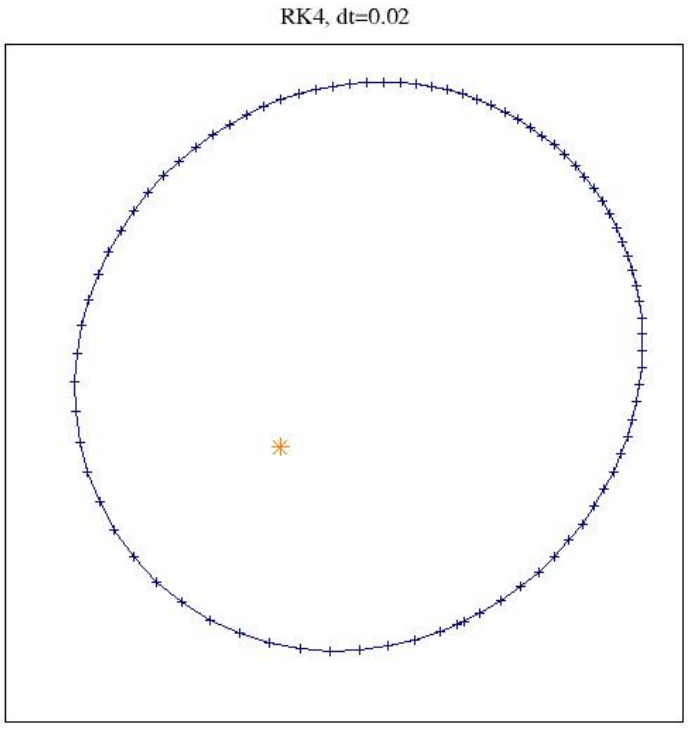
Example: Kepler problem



- Near perihelion: fast motion
- $r(t), \theta(t)$ vary rapidly
- need small time steps
- Near aphelion: slow motion
- can afford bigger time steps

Slow vs fast motion

Example: Kepler problem



- Near perihelion: fast motion
- $r(t), \theta(t)$ vary rapidly
- need small time steps
- Near aphelion: slow motion
- can afford bigger time steps

Solution: Let computer decide what stepsize is needed!

Adaptive stepsize

Let computer decide stepsize depending on what precision we want

- Estimate error at current Δt
- Increase Δt when error estimate 'too small'
- Decrease Δt when error estimate 'too big'

Adaptive stepsize

Let computer decide stepsize depending on what precision we want

- Estimate error at current Δt
- Increase Δt when error estimate 'too small'
- Decrease Δt when error estimate 'too big'

How to estimate the error?

Two approaches:

- Take two steps with $\Delta t/2$, compare with one step with Δt
- Find difference between higher-order step and lower-order (eg RK4 and midpoint)

Adaptive stepsize

What is too small, too big?

- A fixed number is not useful:
  in Kepler problem θ is in radians but r could be in metres!
- Try to keep

$$\left|\frac{\Delta r}{r}\right| \sim \left|\frac{\Delta \theta}{\theta}\right| \sim \Delta$$

— relative tolerance

Adaptive stepsize

What is too small, too big?

- A fixed number is not useful:
  in Kepler problem θ is in radians but r could be in metres!
- Try to keep

$$\left|\frac{\Delta r}{r}\right| \sim \left|\frac{\Delta \theta}{\theta}\right| \sim \Delta$$

  — relative tolerance
- θ can become zero, so we also need absolute tolerance $\Delta\theta > \Delta^{\theta}_{\text{min}}$
- Use physical insight!

Adaptive stepsize

What is too small, too big?

- A fixed number is not useful:
  in Kepler problem θ is in radians but r could be in metres!
- Try to keep

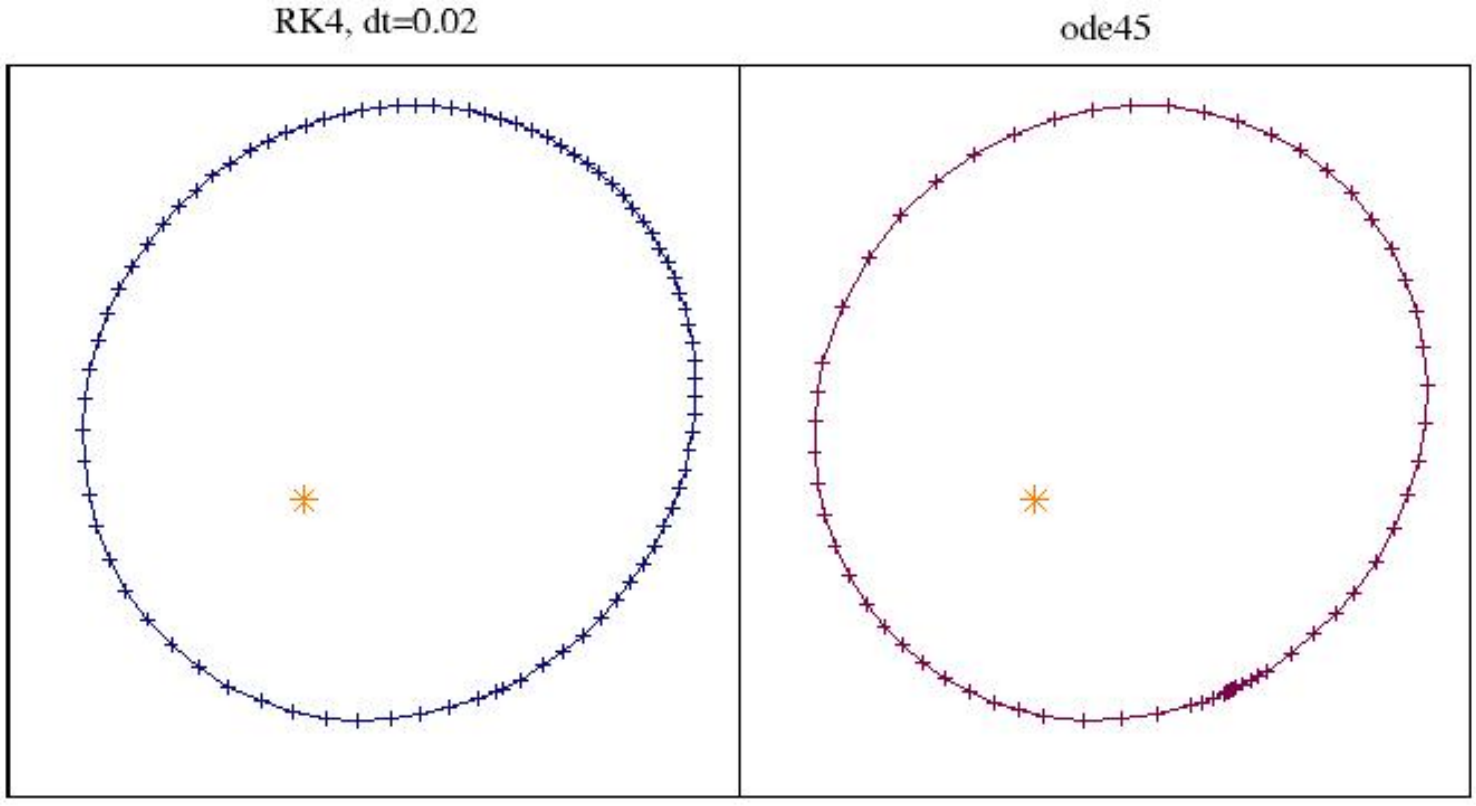
$$\left|\frac{\Delta r}{r}\right| \sim \left|\frac{\Delta \theta}{\theta}\right| \sim \Delta$$

  — relative tolerance
- θ can become zero, so we also need absolute tolerance $\Delta\theta > \Delta^{\theta}_{\text{min}}$
- Use physical insight!

The 'optimal' step size can be estimated using Taylor expansion:

$$\Delta t_{\text{opt}} = \Delta t \left|\frac{\Delta_{\text{rel}}}{\Delta}\right|^{1/N}, \qquad N = \text{order of algorithm}$$

Fixed vs adaptive stepsize



Subjects not covered

- Stiff sets of equations
- Implicit methods
- Richardson extrapolation

Boundary value problems

We want to solve some ordinary differential equation (ODE):

$$y^{(n)}(x) = F\Big(y^{(n-1)}(x), \ldots, y'(x), y(x), x\Big)$$

Boundary value problems

We want to solve some ordinary differential equation (ODE):

$$y^{(n)}(x) = F\Big(y^{(n-1)}(x), \ldots, y'(x), y(x), x\Big)$$

We can always reduce this to a set of n coupled first-order ODEs:

$$\begin{aligned} z_1'(x) &= z_2(x) \\ &\vdots \\ z_{n-1}'(x) &= z_n(x) \\ z_n'(x) &= F(z_n(x), \ldots, z_2(x), z_1(x), x) \end{aligned}$$

For example, $y''(x) = \sin(x)y'(x) + (x+1)\ln y(x) + x^2$ becomes

$$\begin{aligned} z_1'(x) &= z_2(x) \\ z_2'(x) &= \sin(x)z_2(x) + (x+1)\ln z_1(x) + x^2 \end{aligned}$$

Initial vs boundary values

n equations: need n conditions to specify problem completely

Initial vs boundary values

n equations: need n conditions to specify problem completely

Initial value problem

All the conditions are specified at the same point:

$$y(x_1) = a_0\,;\; y'(x_1) = a_1\,;\; \ldots\,;\; y^{(n-1)}(x_1) = a_{n-1}$$

Initial vs boundary values

n equations: need n conditions to specify problem completely

Initial value problem

All the conditions are specified at the same point:

$$y(x_1) = a_0\,;\ y'(x_1) = a_1\,;\ \ldots\,;\ y^{(n-1)}(x_1) = a_{n-1}$$

We have a unique solution, which we can find explicitly:

- Start at $x = x_1$
- Integrate using Euler–Cromer, Runge–Kutta, Stoer–Bullirsch,...

Initial vs boundary values

n equations: need n conditions to specify problem completely

Initial value problem

All the conditions are specified at the same point:

$$y(x_1) = a_0\,;\; y'(x_1) = a_1\,;\; \ldots\,;\; y^{(n-1)}(x_1) = a_{n-1}$$

We have a unique solution, which we can find explicitly:

- Start at $x = x_1$
- Integrate using Euler–Cromer, Runge–Kutta, Stoer–Bullirsch,...

Boundary value problem

Everything else!

Boundary value problems

Conditions are specified at different x:

$$y(0) = a_0\,;\ y(1) = b_0 \quad \text{or}$$
$$y'(0) = a_1\,;\ b_0 y(1) + b_1 y'(1) = c \quad \text{etc}$$

Boundary value problems

Conditions are specified at different x:

$$y(0) = a_0\,;\ y(1) = b_0 \quad \text{or}$$
$$y'(0) = a_1\,;\ b_0 y(1) + b_1 y'(1) = c \quad \text{etc}$$

Much harder

- Can have 0, 1 or many solutions
- Explicit methods cannot be used
- Must guess, then improve

Boundary value problems

Conditions are specified at different x:

$$y(0) = a_0\,;\ y(1) = b_0 \quad \text{or}$$
$$y'(0) = a_1\,;\ b_0 y(1) + b_1 y'(1) = c \quad \text{etc}$$

Much harder

- Can have 0, 1 or many solutions
- Explicit methods cannot be used
- Must guess, then improve

Two general methods: shoot, or relax
Linear ODEs can also be transformed to matrix equations.

Shoot or relax

Shooting method

- Guess initial conditions, integrate to final point
- Adjust your initial guess and repeat

Shoot or relax

Shooting method

- Guess initial conditions, integrate to final point
- Adjust your initial guess and repeat

Relaxation method

- Make a guess for solution in range $[x_1, x_2]$
- Adjust values to bring them closer to satisfying equations

Shoot or relax

Shooting method

- Guess initial conditions, integrate to final point
- Adjust your initial guess and repeat

Relaxation method

- Make a guess for solution in range $[x_1, x_2]$
- Adjust values to bring them closer to satisfying equations
- We will use this to solve PDEs — next semester!

Shoot or relax

Shooting method

- Guess initial conditions, integrate to final point
- Adjust your initial guess and repeat

Relaxation method

- Make a guess for solution in range $[x_1, x_2]$
- Adjust values to bring them closer to satisfying equations
- We will use this to solve PDEs — next semester!

Matrix method

- Divide interval $[x_1, x_2]$ into equally spaced, discrete points
- Write down equation for each point using discrete derivatives
- The set of equations can be written as a matrix equation $Ay = b$
- Solve to find y — next semester!

Shooting method

Shoot first, ask questions later!

Shooting method

Shoot first, ask questions later!

1. Guess the unknown initial conditions [load & aim]

Shooting method

Shoot first, ask questions later!

1. Guess the unknown initial conditions [load & aim]
2. Integrate system of odes to final point [shoot]

Shooting method

Shoot first, ask questions later!

1. Guess the unknown initial conditions [load & aim]
2. Integrate system of odes to final point [shoot]
3. Check how close you got [ask questions]

Shooting method

Shoot first, ask questions later!

1. Guess the unknown initial conditions [load & aim]
2. Integrate system of odes to final point [shoot]
3. Check how close you got [ask questions]
4. Adjust your aim, goto 1

Shooting method

Shoot first, ask questions later!

1. Guess the unknown initial conditions [load & aim]
2. Integrate system of odes to final point [shoot]
3. Check how close you got [ask questions]
4. Adjust your aim, goto 1

"Hitting the target" is a question of reducing the discrepancy $\overrightarrow{y}(x_2) - \overrightarrow{b}$ to zero = root finding

For a second-order ode we can use bisection, secant, Ridders',...

Shooting method

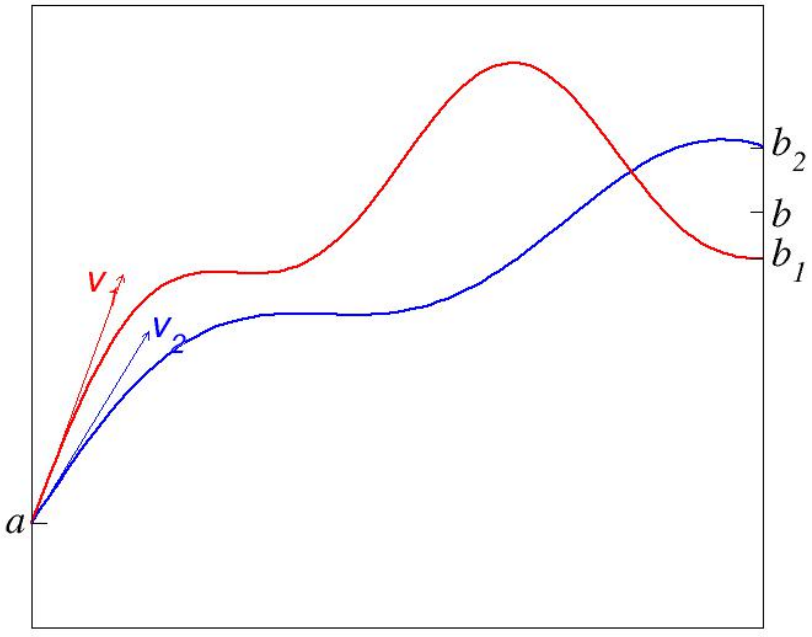
Shoot first, ask questions later!

1. Guess the unknown initial conditions [load & aim]
2. Integrate system of odes to final point [shoot]
3. Check how close you got [ask questions]
4. Adjust your aim, goto 1

"Hitting the target" is a question of reducing the discrepancy $\overrightarrow{y}(x_2) - \overrightarrow{b}$ to zero = root finding

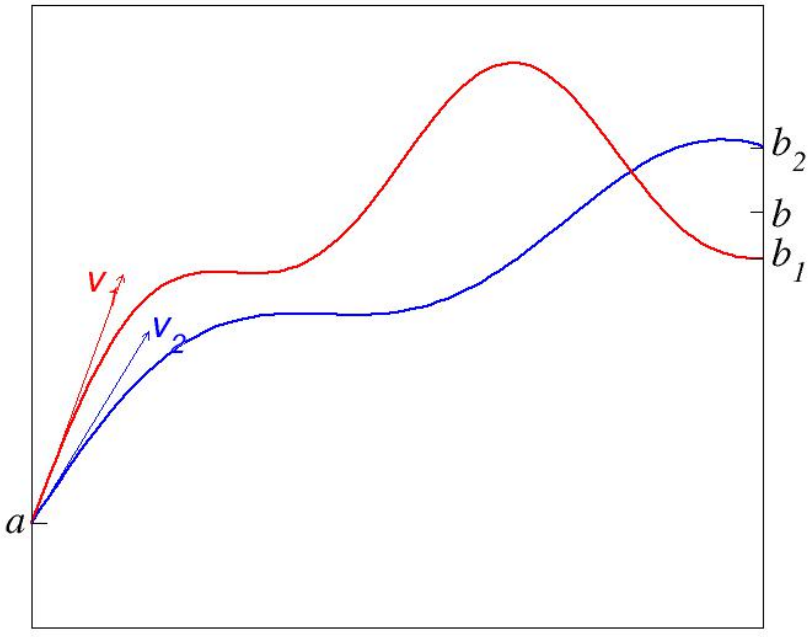
For a second-order ode we can use bisection, secant, Ridders',...
Higher order: Newton–Raphson

Shooting method



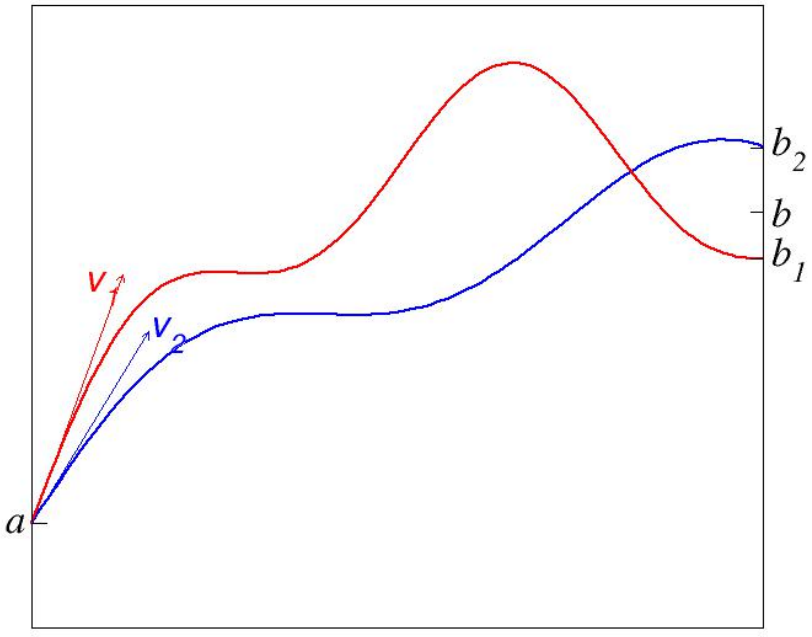
1. We want
   $y(x_1) = a, y(x_2) = b.$

Shooting method



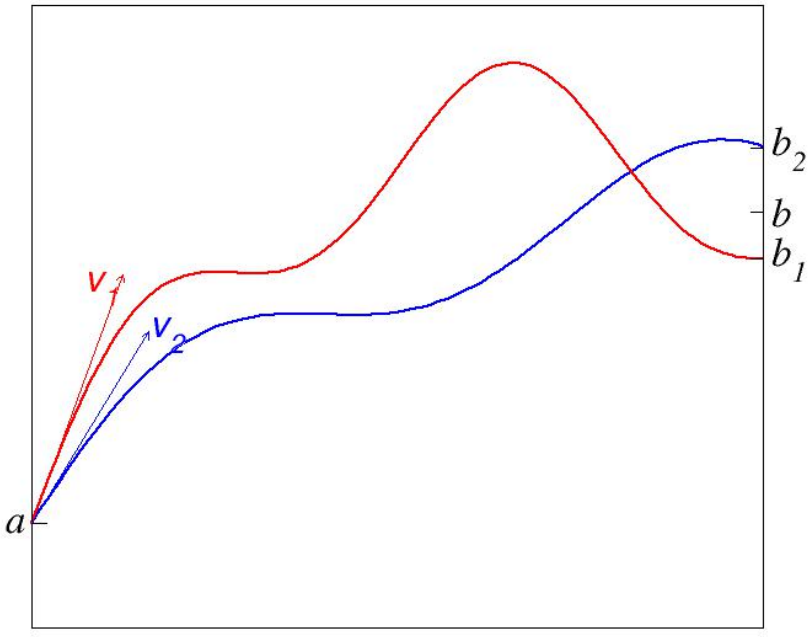
1. We want $y(x_1) = a, y(x_2) = b$.
2. We guess $y'(x_1) = v_1$ and get $y(x_2) = b_1 < b$.

Shooting method



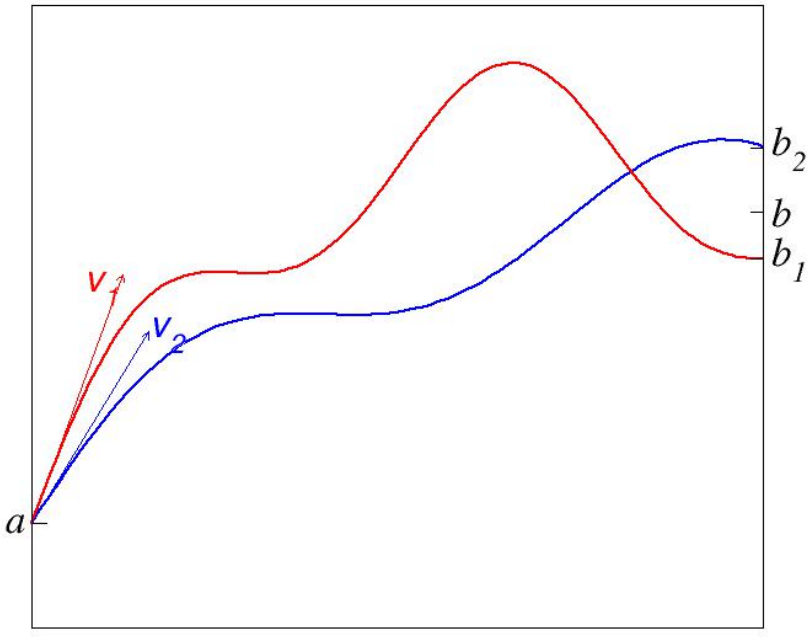
1. We want $y(x_1) = a, y(x_2) = b$.
2. We guess $y'(x_1) = v_1$ and get $y(x_2) = b_1 < b$.
3. We guess $y'(x_1) = v_2$ and get $y(x_2) = b_2 > b$.

Shooting method



1. We want $y(x_1) = a, y(x_2) = b$.
2. We guess $y'(x_1) = v_1$ and get $y(x_2) = b_1 < b$.
3. We guess $y'(x_1) = v_2$ and get $y(x_2) = b_2 > b$.
4. The correct value for $y'(x_1)$ is between v_1 and v_2.

Shooting method



1. We want $y(x_1) = a, y(x_2) = b$.
2. We guess $y'(x_1) = v_1$ and get $y(x_2) = b_1 < b$.
3. We guess $y'(x_1) = v_2$ and get $y(x_2) = b_2 > b$.
4. The correct value for $y'(x_1)$ is between v_1 and v_2.

We can find the value for $y'(x_1)$ using bisection!
This will give us the solution of the boundary value problem.

Summary

Adaptive stepsize

- Allows us to vary the stepsize locally during integration
- Based on estimating the error in the algorithm

Summary

Adaptive stepsize

- Allows us to vary the stepsize locally during integration
- Based on estimating the error in the algorithm

Boundary value problems

- Boundary value problems much harder than inital value
- Must (usually) start by guessing a solution!

Summary

Adaptive stepsize

- Allows us to vary the stepsize locally during integration
- Based on estimating the error in the algorithm

Boundary value problems

- Boundary value problems much harder than inital value
- Must (usually) start by guessing a solution!
- Shooting method:
  1. Guess unknown initial values v_i
  2. Solve ODE with these values: $f(x|v_i)$
  3. Find solution at final point x_f
  4. Solve $f(x_f|v_i) - v_f = 0$ — root finding!