# Numerical ODE Solutions (Runge-Kutta and Extensions)

Kiel Williams
09/23/2016 Algorithms Group

# Form of the Problem

- Need to solve:

$$\frac{dy}{dx} = f(x, y),\ y(0) = a$$

- Other initial condition types exist for higher-order equations (boundary-values)
- Accurate ODE solutions essential to countless theoretical problems
- Many, many, many different approaches for doing this. We'll review some of the most common and straightforward

# Simplest Guess: Euler Approach

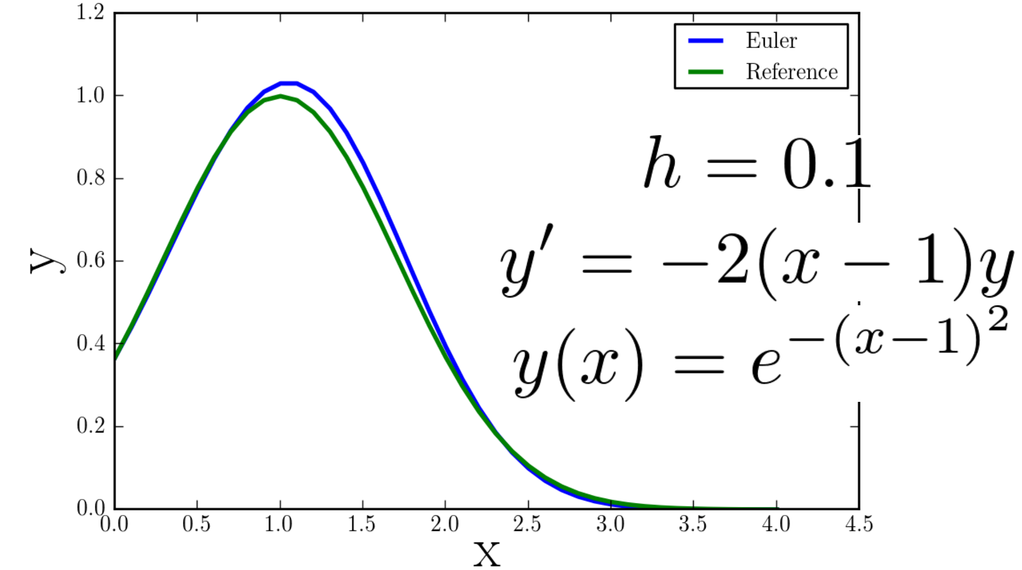
- Simplest guess for discretizing solution:

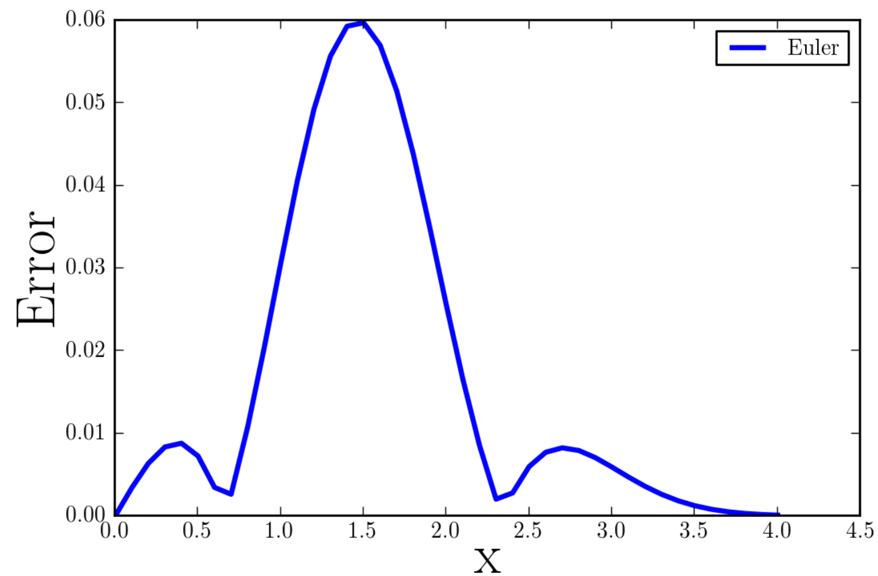
$$y_{n+1} = y_n + hf(x_n, y_n)$$

- But method works poorly:

$$\Delta y \propto \mathcal{O}(h^2)$$

- How can we do better in controlled way?
  - Runge-Kutta family of techniques



# Going Beyond: Runge-Kutta

- Runge-Kutta methods all take form:

$$k_1 = f(x_n, y_n)$$
$$k_2 = f(x_n + c_2 h, y_n + h(a_{21} k_1))$$

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i,$$

$$k_s = f(x_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1})$$

- Described pictorially by Butcher tables:

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & & \ddots & & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \end{array}$$

- For the Euler method:

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

# 4th-Order Runge-Kutta

- ***The*** Runge-Kutta method typically refers to 4th-order Runge-Kutta:

$$y_{n+1} = y_n + h\left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right)$$

$$k_1 = f(x_n, y_n)$$

$$k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$$

$$k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$$

$$k_4 = f(x_n + h, y_n + hk_3)$$

- In Butcher form:

| | | | | |
|---|---|---|---|---|
| 0 | | | | |
| 1/2 | 1/2 | | | |
| 1/2 | 0 | 1/2 | | |
| 1 | 0 | 0 | 1 | |
| | 1/6 | 1/3 | 1/3 | 1/6 |

# 4th-Order Runge-Kutta

- ***The*** Runge-Kutta method typically refers to 4th-order Runge-Kutta:

$$y_{n+1} = y_n + h\left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right)$$

$$k_1 = f(x_n, y_n)$$

$$k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$$

$$k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$$

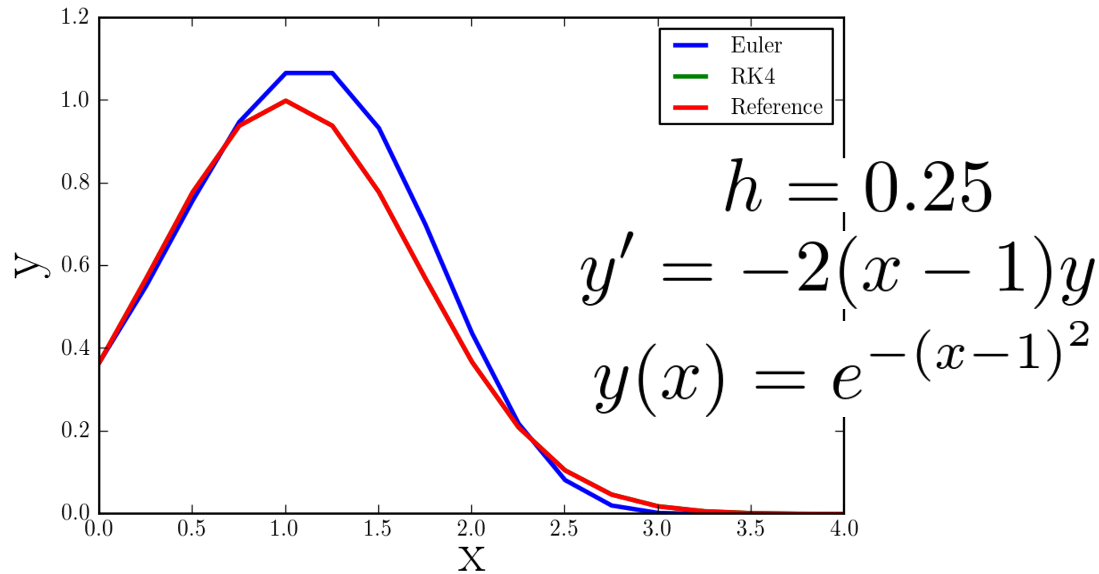
$$k_4 = f(x_n + h, y_n + hk_3)$$

- In Butcher form:

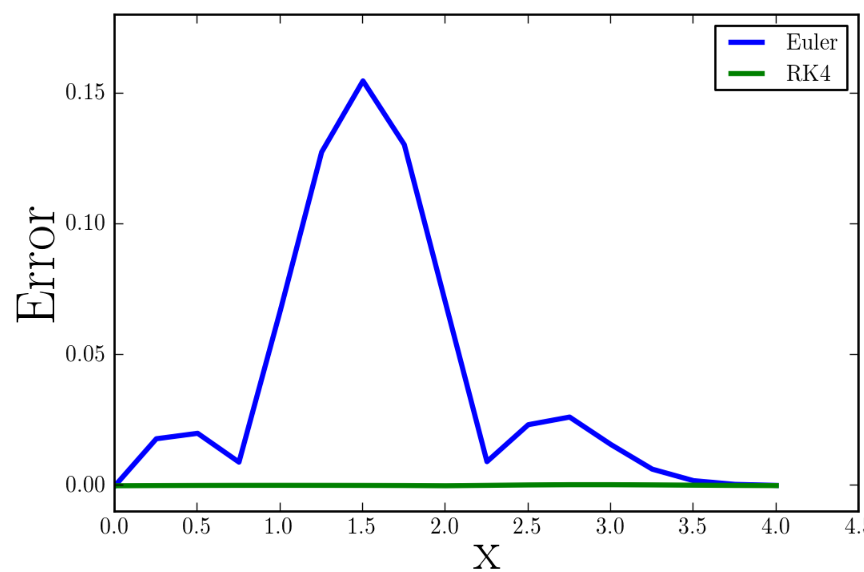
| | | | | |
|---|---|---|---|---|
| 0 | | | | |
| 1/2 | 1/2 | | | |
| 1/2 | 0 | 1/2 | | |
| 1 | 0 | 0 | 1 | |
| | 1/6 | 1/3 | 1/3 | 1/6 |

# Runge-Kutta Examples and Contrast

- RK4 does well, even for large step-sizes
  - RK4 error of ~0.0001, compared to ~0.1 for Euler




- RK4 error scales as O($h^5$)
- If y' depends strictly on *x*, RK4 is equivalent to Simpson's Rule integration



# Runge-Kutta Alternatives: Multi-Step Methods

- Runge-Kutta isn't the only feasible option
  - Instead of expanding the Butcher table, evaluate the derivative at more places

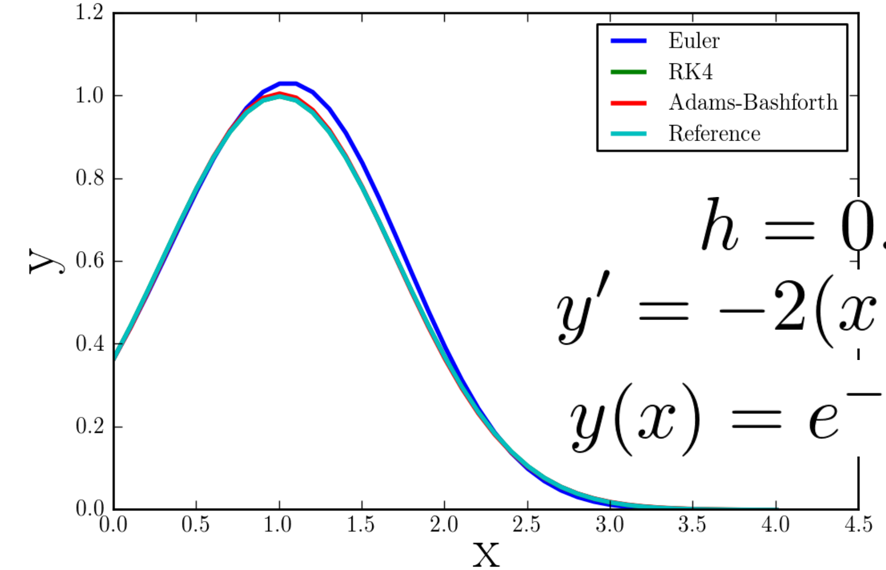
- 2-Step Adams-Bashforth is one of the simplest useful methods:

$$y_{n+2} = y_{n+1} + \frac{3}{2} h f(x_{n+1}, y_{n+1}) - \frac{1}{2} h f(x_n, y_n)$$

- Like Euler's method, but weights first-derivative value at different places
- Coefficient determined by Lagrange polynomial interpolation formula

# Runge-Kutta Alternatives: Multi-Step Methods

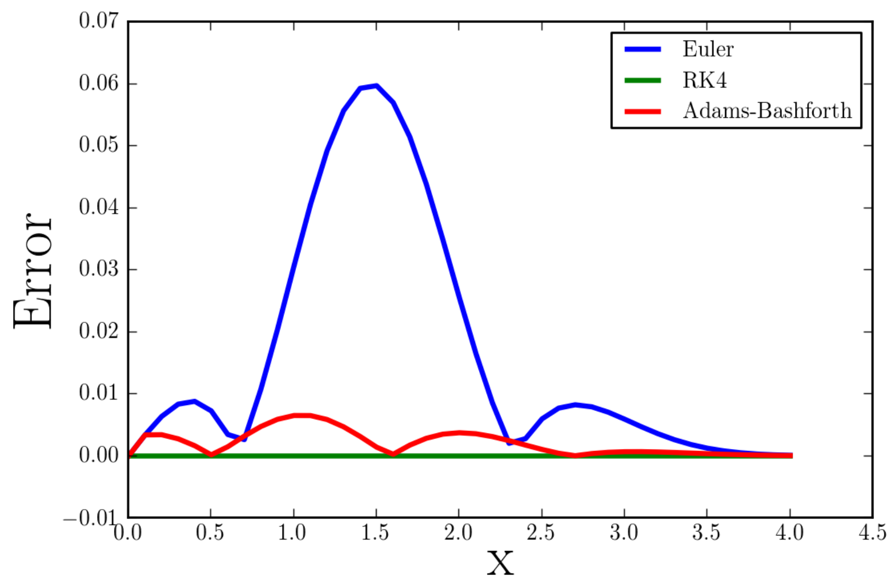
- Adams-Bashforth substantially beats Euler
  - A-B error of ~0.01, compared to ~0.1 for Euler
- Adams-Bashforth error scales as O($h^3$)
- One drawback: need 2 points to start the chain
  - Need one Euler or RK4 step to initiate




$$h = 0.1$$

$$y' = -2(x-1)y$$

$$y(x) = e^{-(x-1)^2}$$



# Implicit Methods for ODE's

- All methods shown so far are ***explicit*** methods, with recursion relations of form:

$$y_{n+1} = F(y_n, y_{n-1}, ..., y_0)$$

- ***Implicit*** methods involve recursions relations of the form:

$$y_{n+1} = F(y_{n+1}, y_n, y_{n-1}, ..., y_0)$$

- Offer improved accuracy, but need to solve an equation to get $y_{n+1}$, evaluate right-hand side of equation
- Typical ways to do this: fixed-point iteration, Newton's method

# Implicit Methods for ODE's, Backward-Euler

- Backward's Euler is simplest implicit method:

$$y_{n+1} = y_n + hf(x_n, y_n)$$ (Forward Euler, Explicit)

↓

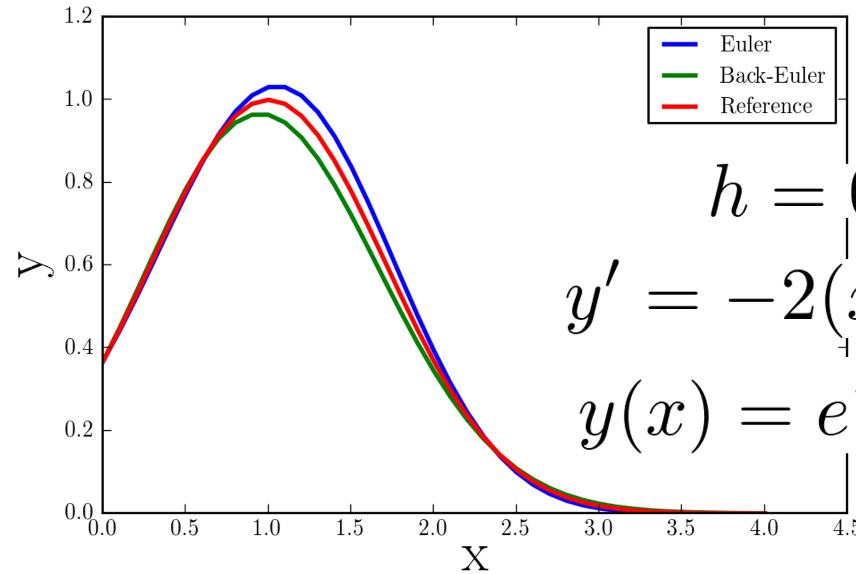
$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1})$$ (Backward Euler, Implicit)

- To extract value of $y_{n+1}$ needed to evaluate right-hand side, use fixed-point iteration to achieve self-consistency:

$$y_{n+1}^{[0]} = y_n, \quad y_{n+1}^{[k+1]} = y_n + hf(x_{n+1}, y_{n+1}^{[k]})$$

# Implicit Methods for ODE's, Backward-Euler

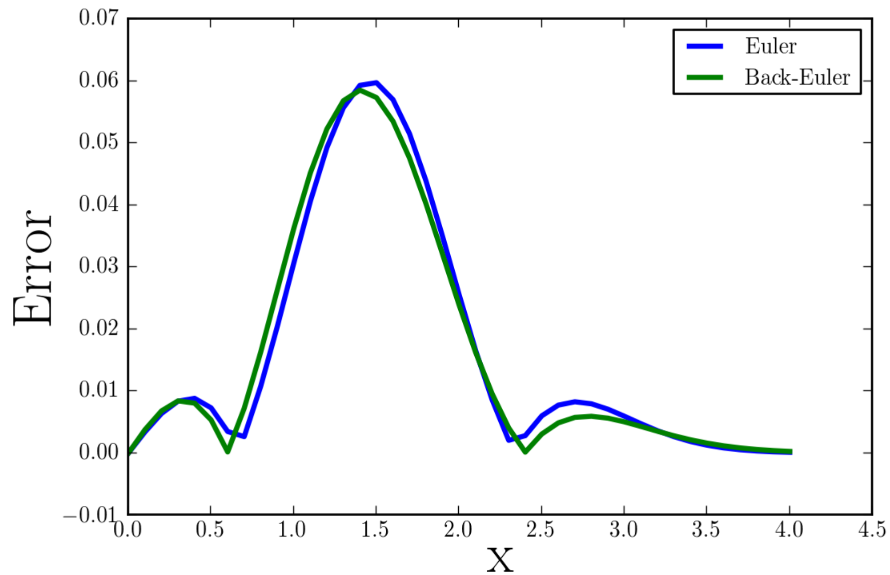
- In this example, backward-Euler doesn't do much better than basic Euler
  - Not always true!
- Error-scaling is the sar $\Delta y \propto \mathcal{O}(h^2)$
- Added complication: need input tolerance for self-consistency loop
  - Best to have tolerance as function of *h*




$$h = 0.1$$

$$y' = -2(x-1)y$$

$$y(x) = e^{-(x-1)^2}$$



# Implicit Multi-Step: Adams-Moulton

- Adams-Moulton methods family combine Adams-Bashforth multi-step approach with implicit techniques

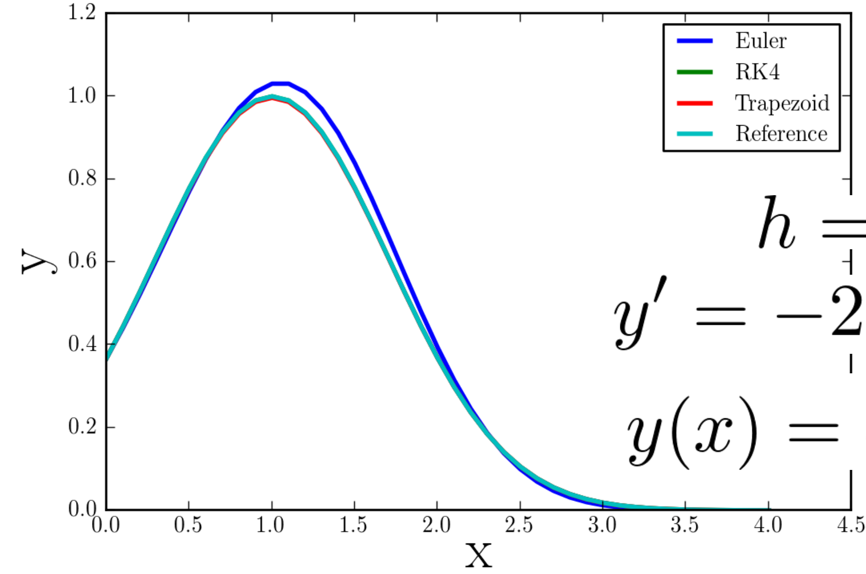
- Most-obvious non-trivial example is ODE analog to the trapezoid rule:

$$y_{n+1} = y_n + \frac{1}{2}h(f(x_{n+1}, y_{n+1}) + f(x_n, y_n))$$

- Arbitrarily high-order algorithms generated very similarly to higher-order Adams-Bashforth approach

# Implicit Multi-Step: Adams-Moulton

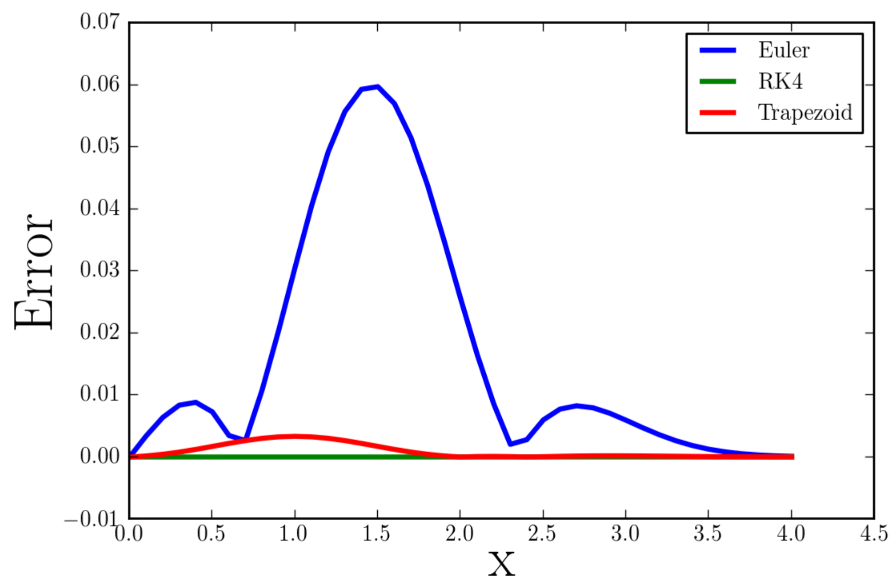
- Trapezoid much better Euler, competitive with RK4
  - Much simpler algorithm than RK4!
- Error scaling goes as O($h^4$) – compare to O($h^3$) for 2-step Adams-Bashforth




$$h = 0.1$$

$$y' = -2(x-1)y$$

$$y(x) = e^{-(x-1)^2}$$



# Exponential Integrators

- Equations whose solutions contain $e^{ax}$ terms notoriously hard to handle – exp. integrators consider ODE's of form:

$$y' = -A\,y + N(y)$$

- We can discretize the exact formal solution to this equation:

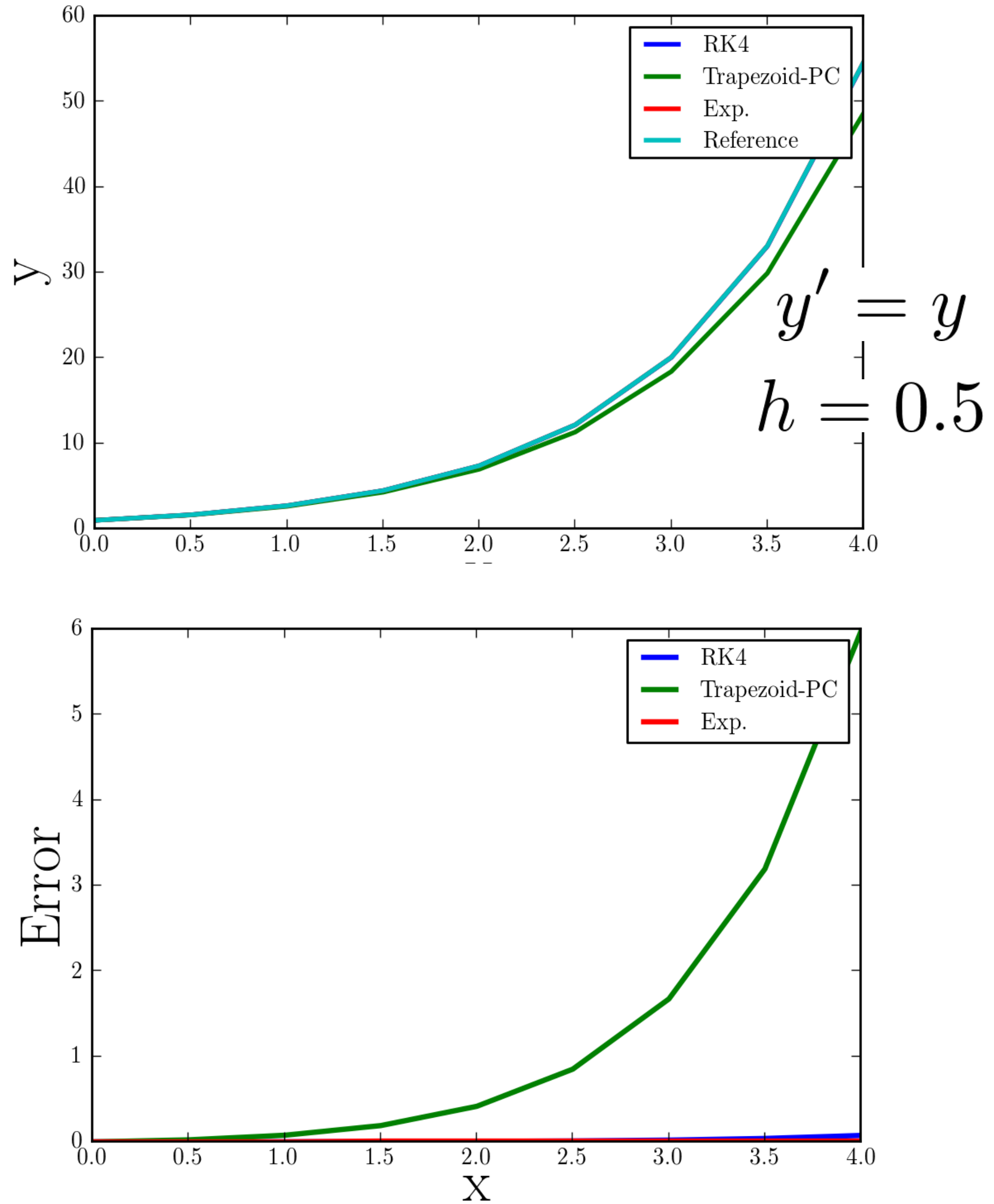
$$y_{n+1} = e^{-Ah}y_n + \int_0^h e^{-(h-\tau)A}N(y(t_n+\tau))d\tau$$

$$\longrightarrow \quad y_{n+1} \approx e^{-Ah}y_n + A^{-1}N(y(t_n))(1-e^{-Ah})$$

- Allows exponential part of y' to be handled exactly – can treat the “rest” of y' as a perturbative expansion

# Exponential Integrators

- Exponential methods exactly solve y = y'
  - Even “good” explicit methods accumulate large errors
- Big drawback: one must often approximate to get ODE in proper form to implement



# Summary

- Euler method is poor, motivates superior techniques:
  - Explicit methods solve ODE by extrapolating from values of y, y' at previous points
  - Examples include all Runge-Kutta type methods, including RK4, multi-step methods like Adams-Bashforth

- Implicit methods require knowledge of function value at next point:
  - Require solving an equation, but give better scaling for same # of function evaluations
  - Often preferred in solution of “stiff” ODE's.