The Runge-Kutta Method and Its Applications

Olga Golovatskaia

Mount Holyoke College

November 25, 2024

Outline

Outline

Euler's Method

- Simple first-order approximation: $y_{n+1} = y_n + hf(t_n, y_n)$
  - y - function
  - n - number of steps
  - h - step size
- Gets more accurate with smaller step sizes and/or more steps
- Works on first-order equations
- Easy to extend to multiple dimensions
- Limited accuracy - numerical error directly proportional to step size

Improved Euler's Method

- Uses the average of two slopes to compute each new value
- More accurate than basic Euler's method
- Steps:
  1. Calculate $k_1 = f(t_n, y_n)$
  2. Calculate $k_2 = f(t_n + h, y_n + hk_1)$
  3. Update using $y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2)$
- Still has limitations but offers improved accuracy while remaining relatively simple

Why We Need Better Methods

- Both methods have issues crossing equilibrium solutions
- Can blow up at points where the solution should be stable
- Smaller step sizes needed for accuracy = more computation
- Many practical applications require higher precision
- Need methods that provide better accuracy without excessive computational cost

Outline

Runge-Kutta Method - Overview

- Uses **four slopes** to compute each step
- Takes a weighted average of these slopes
- The solution to the differential equation $\frac{dy}{dt} = f(t, y)$ is approximated by:
  - First slope m_k: Computed at the beginning of the interval
    $m_k = f(t_k, y_k)$
  - Second slope n_k: Calculated halfway through the interval
    $n_k = f(\tilde{t}, \tilde{y}_k)$ where $\tilde{y}_k = y_k + m_k \frac{\Delta t}{2}$
  - Third slope q_k: Also calculated halfway through the interval
    $q_k = f(\tilde{t}, \hat{y}_k)$ where $\hat{y}_k = y_k + n_k \frac{\Delta t}{2}$
  - Fourth slope p_k: Calculated at the endpoint of the interval
    $p_k = f(t_{k+1}, \bar{y}_k)$ where $\bar{y}_k = y_k + q_k \Delta t$
- The weighted average gives us:

$$y_{k+1} = y_k + \left(\frac{m_k + 2n_k + 2q_k + p_k}{6} \right) \Delta t$$

Runge-Kutta Method - Advantages

- **More accurate** - fourth-order method
- **Not much harder to compute** than simpler methods
- **Can be applied to higher-order DEs** - convert to system of first-order equations
- Only practical method for studying second and higher-order DEs numerically
- Well-suited for implementations in scientific computing software

Outline

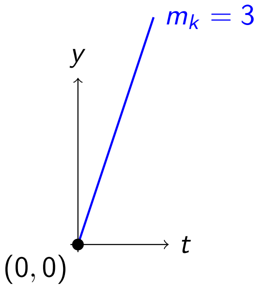
Example Problem

- Differential equation: $\frac{dy}{dt} = (3 - y)(y + 1)$
- Initial condition: $y(0) = 0$
- Time interval: $0 \leq t \leq 5$
- Using $n = 10$ steps (step size $\Delta t = 0.5$)

First Slope Calculation

- First slope m_k at the beginning of the interval:

$$\begin{aligned} m_k &= f(t_k, y_k) \\ &= f(0,0) \\ &= (3-0)(0+1) \\ &= 3 \end{aligned}$$



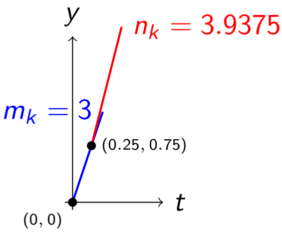
Second Slope Calculation

- Second slope n_k, halfway through the interval:

$$\tilde{t} = 0 + \frac{\Delta t}{2} = 0 + \frac{0.5}{2} = 0.25$$

$$\tilde{y}_k = y_k + m_k \cdot \frac{\Delta t}{2} = 0 + 3 \cdot \frac{0.5}{2} = 0.75$$

$$\begin{aligned} n_k &= f(\tilde{t}, \tilde{y}_k) \\ &= (3 - 0.75)(0.75 + 1) \\ &= 2.25 \cdot 1.75 \\ &= 3.9375 \end{aligned}$$

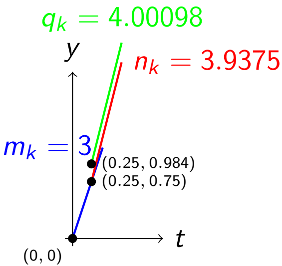


Third Slope Calculation

- Third slope q_k, also at the midpoint but using the second slope:

$$\hat{y}_k = y_k + n_k \cdot \frac{\Delta t}{2} = 0 + 3.9375 \cdot \frac{0.5}{2} = 0.984375$$

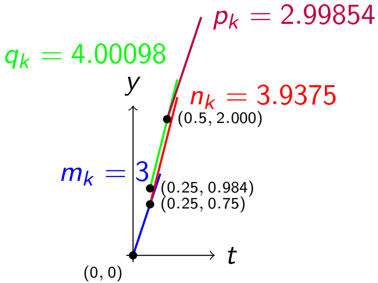
$$\begin{aligned} q_k &= f(\tilde{t}, \hat{y}_k) \\ &= (3 - 0.984375)(0.984375 + 1) \\ &= 2.015625 \cdot 1.984375 \\ &= 4.00098 \end{aligned}$$



Fourth Slope Calculation

- Fourth slope p_k at the end of the interval:

$$\begin{aligned} t_{k+1} &= 0 + \Delta t = 0 + 0.5 = 0.5 \\ \bar{y}_k &= y_k + q_k \cdot \Delta t = 0 + 4.00098 \cdot 0.5 = 2.00049 \\ p_k &= f(t_{k+1}, \bar{y}_k) \\ &= (3 - 2.00049)(2.00049 + 1) \\ &= 0.99951 \cdot 3.00049 \\ &= 2.99854 \end{aligned}$$



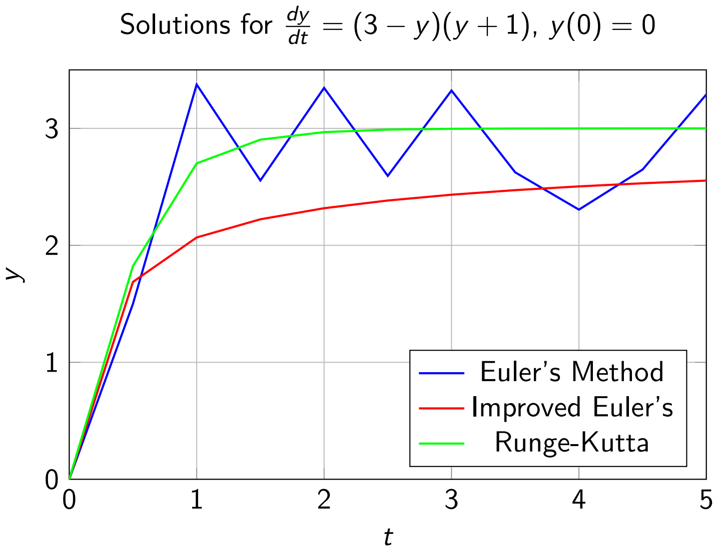
Computing the Next Step

- Weighted average of all four slopes:

$$
\begin{aligned}
y_{k+1} &= y_k + \frac{m_k + 2n_k + 2q_k + p_k}{6} \cdot \Delta t \\
&= 0 + \frac{3 + 2 \cdot 3.9375 + 2 \cdot 4.00098 + 2.99854}{6} \cdot 0.5 \\
&= 0 + \frac{3 + 7.875 + 8.00196 + 2.99854}{6} \cdot 0.5 \\
&= 0 + \frac{21.8755}{6} \cdot 0.5 \\
&= 0 + 3.64592 \cdot 0.5 \\
&= 1.823
\end{aligned}
$$

- So $y_1 = 1.823$ is our next approximation

Comparison of Methods



Outline

What is Order?

- Order measures accuracy of a numerical approximation
- Formal definition:
  - Let u be the exact solution
  - Let u_h be the approximation depending on parameter h (step size)
  - Let C be a constant
- A numerical approximation has order of accuracy p when:

$$|u_h - u| \leq Ch^p$$

- Error is roughly proportional to step size to the power of order
- For higher order, making step size smaller makes approximation significantly more accurate

Order Analysis Example

- Example problem:

$$\frac{dx}{dt} = 1 + \frac{x}{t}, \quad 1 \leq t \leq 6$$
$$x(1) = 1$$

- Actual solution: $x(t) = t(1 + \ln(t))$
- At $t = 6$, actual solution $x(6) \approx 16.75055682$
- We'll compare approximations using different methods and step sizes

Euler's Method Error Analysis

Step size h	Euler approx.	Error	Error ratio
1	14.7	2.05055682	-
$1/2$	15.61926407	1.13129275	1.81257841
$1/4$	16.15574907	0.59480775	1.90194688
$1/8$	16.44564019	0.30491663	1.95072260
$1/16$	16.59620493	0.15435189	1.97546411
$1/32$	16.67290649	0.07765033	1.98778145

- **Key takeaway:** When step size changes by a factor of 2, the error also changes by about a factor of 2 (especially for small h)
- This confirms Euler's is a first-order method ($p = 1$)
- Error relationship is approximately linear: $|u_h - u| \approx 2h$

Comparison of Errors for Different Methods

Improved Euler's Method

h	Approx.	Error	Ratio
1	16.17417	0.57639	-
1/2	16.56419	0.18636	3.09284
1/4	16.69729	0.05327	3.49840
1/8	16.73632	0.01423	3.74249
1/16	16.74688	0.00368	3.87100
1/32	16.74962	0.00093	3.93566

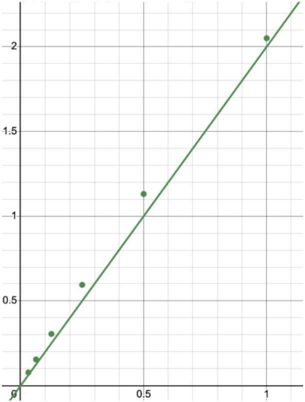
Runge-Kutta Method

h	Approx.	Error	Ratio
1	16.72842	0.02213	-
1/2	16.74839	0.00217	10.19443
1/4	16.75039	0.00017	13.03626
1/8	16.75055	0.00001	14.65958
1/16	16.75056	7.4e-7	15.31442
1/32	16.75056	5.2e-8	14.40075

Visualization of Order and Error

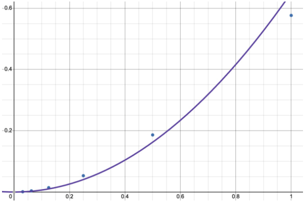
Euler's Method

- 1st order ($p = 1$)
- Error $\approx 2h$
- Linear relationship



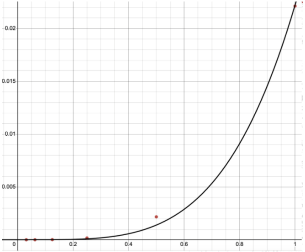
Improved Euler's

- 2nd order ($p = 2$)
- Error $\approx 0.65h^2$
- Quadratic relationship



Runge-Kutta

- 4th order ($p = 4$)
- Error $\approx \frac{1}{45}h^4$
- Quartic relationship



Specialized Runge-Kutta Methods

Based on Van der Houwen & Sommeijer (1987):

- Special Runge-Kutta methods for systems of ODEs:

$$\frac{d^k y}{dt^k} = f(x, y), \quad k = 1, 2$$

- For oscillatory ODEs:

$$\frac{d^k y}{dt^k} = (i\omega)^k y, \quad \omega \text{ is real}$$

- m-stage Runge-Kutta method in the form:

$$y_n^{(0)} = y_{n-1}$$

$$y_n^{(j)} = y_{n-1} + h \sum_{l=0}^{j-1} \lambda_{j,l} f(t_{n-1} + \mu_l h, y_n^{(l)}), \quad j = 1, \ldots, m$$

- With application to oscillation equations, leads to:

$$y_n = a^n y_0, \quad a^n := A_m(v^2) + ivB_m(v^2), \quad v := \omega h$$

where A_m and B_m are polynomials in v^2

Outline

5. Appendix: MATLAB Code

```
f = @(t,y) (3-y)*(y+1); % DE
y0 = 0; % y IC
t0 = 0; % start time
tfinal = 5; % stop time
h = 0.5; % stepsize

t = t0:h:tfinal;
ystar = zeros(size(t));
ystar(1) = y0;

for i = 2:numel(t) % replace 2nd entry and on with y-values
    ystar(i) = ystar(i-1)+h*f(t(i-1),ystar(i-1)); % euler's method step
end
plot(t',ystar','go-','LineWidth',2)
title('Euler''s Method')
variable_names = {'t','y'};

xlim([0 5])
ylim([0 4])
```

```
varnames = {'t','y'};
table(t',ystar','VariableNames',varnames)

hold on
```

Listing 1: Euler's Method Implementation

```
f = @(t,y) (3-y)*(y+1); % DE
y0 = 0; % y IC
t0 = 0; % start time
tfinal = 5; % stop time
h = 0.5; % stepsize

t = t0:h:tfinal;
ystar = zeros(size(t));

ystar(1) = y0;

for i = 2:numel(t) % replace 2nd entry and on with y-
    values
    nslope = f(t(i-1),ystar(i-1)); % slope at left
```

```
    side of step
    ytilde = ystar(i-1)+h*nslope; % euler's method
    step
    mslope = f(t(i),ytilde); % slope at end of euler's
     method step
    ystar(i) = ystar(i-1)+((nslope+mslope)/2)*h; %
    improved euler's method
end
plot(t,ystar,'mo-','LineWidth',2)
title('Improved Euler''s Method')

varnames = {'t','y'};
table(t',ystar','VariableNames',varnames)%plot points

xlim([0 5])
ylim([0 4])


hold on
```

Listing 2: Improved Euler's Method Implementation

```
clc;
clear all;
h=0.5;
x = 0:h:5;
y = zeros(1,length(x));
y(1) = 0;                                                  % initial condition
F_xy = @(t,r) (3-r)*(r+1);
for i=1:(length(x)-1)                                      % calculation loop
    k_1 = F_xy(x(i),y(i));
    k_2 = F_xy(x(i)+0.5*h,y(i)+0.5*h*k_1);
    k_3 = F_xy((x(i)+0.5*h),(y(i)+0.5*h*k_2));
    k_4 = F_xy((x(i)+h),(y(i)+k_3*h));
    y(i+1) = y(i) + (1/6)*(k_1+2*k_2+2*k_3+k_4)*h;   % main equation
end

tspan = [0,100]; y0 = y(1);
[tx, yx] = ode45(F_xy, tspan, y0);
plot(x,y,'bo-', 'LineWidth',1.5, 'MarkerSize',8)  %
```

```
    RK4 solution in blue
title('Runge-Kutta Method')

xlim([0 5])
ylim([0 4])

varnames = {'t','y'};
RungeKuttaMethod = table(x',y','VariableNames',
    varnames)
```

Listing 3: Runge-Kutta 4th Order Method Implementation

References

Blanchard, P., Devaney, R. L., & Hall, G. R. (2012). *Differential Equations*. Cengage Learning.

Bradie, S. I. (2006). *A friendly introduction to numerical analysis* (2nd ed.). Pearson.

Lebl, J. (2024). *Introduction to Differential Equations*. `https://www.jirka.org/diffyqs/html/diffyqs.html`

LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations*. SIAM.

Kumar, A., & Unny, T. E. (1977). Application of Runge-Kutta method for the solution of nonlinear partial differential equations. *Applied Mathematical Modelling, 1*, 199-204.

Graney, L., & Richardson, A. A. (1981). The numerical solution of non-linear partial differential equations by the method of lines. *Journal of Computational and Applied Mathematics, 7*(4), 229–236.

Van der Houwen, P. J., & Sommeijer, B. P. (1987). Explicit Runge-Kutta (-Nyström) Methods with Reduced Phase Errors for Computing Oscillating Solutions. *SIAM Journal on Numerical Analysis, 24*(3), 595–617.