Outline

1. Examples, and Recap
   - Euler’s, Heun’s, and Runge’s Methods
   - Recap: Deriving Runge-Kutta Methods
   - Recap: Pending Issues

2. Runge-Kutta: Outstanding Issues
   - Error Estimation
   - Stability Analysis
   - Consistency

3. A Brief History, and RK-Construction Methods
   - Runge-Kutta Methods, Historical Overview
   - s-stage Runge-Kutta Methods, a recap
   - Order Conditions

4. Rooted Trees
   - Definitions
   - The Quantities $\Phi(t)$, and $\gamma(t)$
   - Designing a Runge-Kutta Scheme Based on $\Phi(t)$ and $\gamma(t)$

5. Stability of Explicit Runge-Kutta Methods
   - Some Notes...
Recapping the mission...

- We are trying to solve the ODE

\[ y'(t) = f(t, y), \quad y(t_0) = y_0, \quad t < T \]

using a numerical scheme applied to the discretization
\[ t_n = t_0 + n \cdot h, \text{ where } h \text{ is the step-size (in time)}. \]
Recapping the mission...

- We are trying to solve the ODE

\[ y'(t) = f(t, y), \quad y(t_0) = y_0, \quad t < T \]

using a numerical scheme applied to the discretization

\[ t_n = t_0 + n \cdot h, \text{ where } h \text{ is the step-size (in time)}. \]

- In Euler’s method we use the slope \( f(t, y) \) evaluated at the current (known) time level \((t_n, y_n)\) and use that value as an approximation of the slope throughout the interval \([t_n, t_{n+1}]\).
Recapping the mission...

- We are trying to solve the ODE
  \[ y'(t) = f(t, y), \quad y(t_0) = y_0, \quad t < T \]
  using a numerical scheme applied to the discretization
  \[ t_n = t_0 + n \cdot h, \text{ where } h \text{ is the step-size (in time)}. \]

- In Euler’s method we use the slope \( f(t, y) \) evaluated at the current (known) time level \((t_n, y_n)\) and use that value as an approximation of the slope throughout the interval \([t_n, t_{n+1}]\).

- RK-methods improve on Euler’s method by looking at the slope at **multiple points**.
Euler’s Method — $y'(t) = y(t) + 2t - 1$, $y(0) = 1$ ($h = 1/2$)

Euler’s Method samples the slope at the beginning of the step only.

![Diagram showing comparison between exact solution and Euler solution for the given differential equation. The diagram illustrates the step-by-step application of Euler's method, highlighting the discrepancy between the exact solution and the Euler solution, especially noticeable in the second step.]
Heun’s method samples the slope at the beginning and the end, and uses the average as the final approximation of the slope.

Step#1: $k_1 = f(t_0, y_0)$. 
Heun’s method samples the slope at the beginning and the end, and uses the average as the final approximation of the slope.

Heun’s Method: $y'(t) = y(t) + 2t - 1, \ y(0) = 1 \ (h = 1/2)$

Step#1: $k_1 = f(t_0, y_0), \ k_2 = f(t_0 + h, y_0 + hk_1), \ y_1 = y_0 + \frac{h}{2}(k_1 + k_2)$.
Heun’s Method — $y'(t) = y(t) + 2t - 1$, $y(0) = 1$ ($h = 1/2$)

Heun’s method samples the slope at the beginning and the end, and uses the average as the final approximation of the slope.

Step#1: $k_1 = f(t_0, y_0)$, $k_2 = f(t_0 + h, y_0 + hk_1)$, $y_1 = y_0 + \frac{h}{2}(k_1 + k_2)$. Step#2: $k_1 = f(t_1, y_1)$.
Heun’s Method — $y'(t) = y(t) + 2t - 1, \ y(0) = 1 \ (h = 1/2)$

Heun’s method samples the slope at the beginning and the end, and uses the average as the final approximation of the slope.

![Graph showing Heun's method comparison]

Step#1: $k_1 = f(t_0, y_0)$, $k_2 = f(t_0 + h, y_0 + hk_1)$, $y_1 = y_0 + \frac{h}{2}(k_1 + k_2)$. Step#2: $k_1 = f(t_1, y_1)$, $k_2 = f(t_1 + h, y_1 + hk_1)$, $y_2 = y_1 + \frac{h}{2}(k_1 + k_2)$. 

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩  Runge-Kutta Methods, Continued — (5/7)
Heun’s Method — \( y'(t) = y(t) + 2t - 1, \ y(0) = 1 \) \((h = 1/2)\)

Heun’s method samples the slope at the beginning and the end, and uses the average as the final approximation of the slope.

\[
\begin{align*}
\text{Step}\#1: \quad k_1 &= f(t_0, y_0), \quad k_2 = f(t_0 + h, y_0 + hk_1), \quad y_1 = y_0 + \frac{h}{2}(k_1 + k_2). \\
\text{Step}\#2: \quad k_1 &= f(t_1, y_1), \quad k_2 = f(t_1 + h, y_1 + hk_1), \quad y_2 = y_1 + \frac{h}{2}(k_1 + k_2).
\end{align*}
\]
Runge’s Method — \( y'(t) = y(t) + 2t - 1, \ y(0) = 1 \ (h = 1/2) \)

Stage#1: \( k_1 = f(t_0, y_0) \)
Examples, and Recap
Runge-Kutta: Outstanding Issues
A Brief History, and RK-Construction Methods
Rooted Trees
Stability of Explicit Runge-Kutta Methods

Runge’s Method — \( y'(t) = y(t) + 2t - 1, \ y(0) = 1 \ (h = 1/2) \)

Stage\#1: \( k_1 = f(t_0, y_0), \ k_2 = f(t_0 + h/2, y_0 + hk_1/2). \)
Runge’s Method — $y'(t) = y(t) + 2t - 1, \ y(0) = 1 \ (h = 1/2)$

Stage#1: $k_1 = f(t_0, y_0), \ k_2 = f(t_0 + h/2, y_0 + hk_1/2), \ k_3 = f(t_0 + h/2, y_0 + hk_2/2)$. 

Peter Blomgren, [blomgren.peter@gmail.com]  Runge-Kutta Methods, Continued  — (6/47)
Runge’s Method — $y'(t) = y(t) + 2t - 1$, $y(0) = 1$ ($h = 1/2$)

Stage#1: $k_1 = f(t_0, y_0)$, $k_2 = f(t_0 + h/2, y_0 + hk_1/2)$, $k_3 = f(t_0 + h/2, y_0 + hk_2/2)$, $k_4 = f(t_0 + h, y_0 + hk_3)$, $y_1 = y_0 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$. 
Runge’s Method — $y'(t) = y(t) + 2t - 1,\ y(0) = 1\ (h = 1/2)$

Stage#1: $k_1 = f(t_0, y_0)$, $k_2 = f(t_0 + h/2, y_0 + hk_1/2)$, $k_3 = f(t_0 + h/2, y_0 + hk_2/2)$, $k_4 = f(t_0 + h, y_0 + hk_3)$, $y_1 = y_0 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$.

Stage#2: $k_1 = f(t_1, y_1)$. 

---

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩  Runge-Kutta Methods, Continued — (6/47)
Runge’s Method — $y'(t) = y(t) + 2t - 1$, $y(0) = 1$ ($h = 1/2$)

Stage#1: $k_1 = f(t_0, y_0)$, $k_2 = f(t_0 + h/2, y_0 + hk_1/2)$, $k_3 = f(t_0 + h/2, y_0 + hk_2/2)$, $k_4 = f(t_0 + h, y_0 + hk_3)$, $y_1 = y_0 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$.

Stage#2: $k_1 = f(t_1, y_1)$, $k_2 = f(t_1 + h/2, y_1 + hk_1/2)$.
Runge’s Method — $y'(t) = y(t) + 2t - 1, \ y(0) = 1 \ (h = 1/2)$

Stage#1: $k_1 = f(t_0, y_0), k_2 = f(t_0 + h/2, y_0 + hk_1/2), k_3 = f(t_0 + h/2, y_0 + hk_2/2), k_4 = f(t_0 + h, y_0 + hk_3),$

$y_1 = y_0 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).$

Stage#2: $k_1 = f(t_1, y_1), k_2 = f(t_1 + h/2, y_1 + hk_1/2), k_3 = f(t_1 + h/2, y_1 + hk_2/2).$
Runge’s Method — $y'(t) = y(t) + 2t - 1$, $y(0) = 1$ ($h = 1/2$)

Stage#1: $k_1 = f(t_0, y_0)$, $k_2 = f(t_0 + h/2, y_0 + hk_1/2)$, $k_3 = f(t_0 + h/2, y_0 + hk_2/2)$, $k_4 = f(t_0 + h, y_0 + hk_3)$, $y_1 = y_0 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$.

Stage#2: $k_1 = f(t_1, y_1)$, $k_2 = f(t_1 + h/2, y_1 + hk_1/2)$, $k_3 = f(t_1 + h/2, y_1 + hk_2/2)$, $k_4 = f(t_1 + h, y_1 + hk_3)$, $y_2 = y_1 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$. 

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩
Runge’s Method — $y'(t) = y(t) + 2t - 1, \ y(0) = 1 \ (h = 1/2)$

Stage#1: $k_1 = f(t_0, y_0)$, $k_2 = f(t_0 + h/2, y_0 + hk_1/2)$, $k_3 = f(t_0 + h/2, y_0 + hk_2/2)$, $k_4 = f(t_0 + h, y_0 + hk_3)$, $y_1 = y_0 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$.

Stage#2: $k_1 = f(t_1, y_1)$, $k_2 = f(t_1 + h/2, y_1 + hk_1/2)$, $k_3 = f(t_1 + h/2, y_1 + hk_2/2)$, $k_4 = f(t_1 + h, y_1 + hk_3)$, $y_2 = y_1 + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$. 

Exact Solution
- Runge 4-stage Solution
- k1 stage 1
- k2 stage 1
- k3 stage 1
- k4 stage 1
- k1 stage 2
- k2 stage 2
- k3 stage 2
- k4 stage 2

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩ Runge-Kutta Methods, Continued
You may say… “No Big Surprise There!”

“Of course we do better with 8 measurements of the derivative (Runge with \( h = \frac{1}{2} \)), I bet if we used Euler’s method with 8 measurements (\( h = \frac{1}{8} \)) we’d do just as good a job — and we wouldn’t have to figure out the coefficients!”
You may say… “No Big Surprise There!”

“Of course we do better with 8 measurements of the derivative (Runge with \( h = \frac{1}{2} \)), I bet if we used Euler’s method with 8 measurements (\( h = \frac{1}{8} \)) we’d do just as good a job — and we wouldn’t have to figure out the coefficients!”
You may say… “No Big Surprise There!”

“Of course we do better with 8 measurements of the derivative (Runge with $h = \frac{1}{2}$), I bet if we used Euler’s method with 8 measurements ($h = \frac{1}{8}$) we’d do just as good a job — and we wouldn’t have to figure out the coefficients!”

Runge, $h = \frac{1}{2}$; $\text{LTE}(h) \sim O(h^4)$

Euler, $h = \frac{1}{8}$; $\text{LTE}(h) \sim O(h)$
Summary: Runge-Kutta vs. Euler

- By combining multiple “measurements” of the slope \( y'(t) = f(t, y) \) in the step-interval, the RK-method builds up a more accurate final step.

  - In the previous example, where \( \text{LTE}_{\text{RK}}(h) \sim O(h^4) \), cutting the step-size \( h \) in half (\( \Leftrightarrow \) doubling the number of measurements), reduces the error by a factor of \( \frac{1}{2^4} = \frac{1}{16} \).

  - Roughly \( \text{Work} \times \text{Error} \sim O(h^3) \)

- Euler’s method with the same number of “measurements” (smaller step-size \( h \)) is still a first order method.

  - Doubling the number of measurements reduces the error by \( \frac{1}{2} \)

  - Roughly \( \text{Work} \times \text{Error} \sim O(1) \)
The Butcher array for a 2-stage explicit RK method has the form:

\[
\begin{array}{c|cc}
0 & 0 & 0 \\
c_2 & a_{2,1} & 0 \\
\hline
b_1 & b_2 & \sim
\end{array}
\begin{array}{c|cc}
0 & 0 & 0 \\
c_2 & c_2 & 0 \\
\hline
b_1 & 1 - b_1 & \end{array}
\]

Hence,

\[
\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + c_2 h, y_n + c_2 hk_1) \\
y_{n+1} &= y_n + h [b_1 k_1 + (1 - b_1) k_2]
\end{aligned}
\]

Describes all possible explicit 2-stage RK-methods.

We Taylor expand to determine the parameters \( c_2 \) and \( b_1 \)...
With the following Taylor expansions:

\[
\begin{align*}
y_{n+1} &= y_n + hf_n + \frac{h^2}{2} f_n' + O(h^3) \\
k_1 &= f_n \\
k_2 &= f(t_n + c_2 h, y_n + c_2 h k_1) \\
&= f_n + (c_2 h) \frac{\partial}{\partial t} f(t_n, y_n) + (c_2 h) \frac{\partial}{\partial y} f(t_n, y_n) y'(t) + O(h^2)
\end{align*}
\]

We can define the Local Truncation Error

\[
\text{LTE}(h) = \frac{y_{n+1} - y_n}{h} - b_1 k_1 - (1 - b_1) k_2
\]

\[
= \left[ f_n + \frac{h}{2} f_n' + O(h^2) \right] - \\
- \left[ b_1 f_n + (1 - b_1) \left( f_n + (c_2 h) \left[ \frac{\partial}{\partial t} f_n + \frac{\partial}{\partial y} f_n \cdot f_n \right] \right) \right] \\
= \frac{h}{2} \left[ \frac{\partial}{\partial t} f_n + \frac{\partial}{\partial y} f_n \cdot f_n \right] - b_2 c_2 h \left[ \frac{\partial}{\partial t} f_n + \frac{\partial}{\partial y} f_n \cdot f_n \right] + O(h^2)
\]
We have

\[
\text{LTE}(h) = \frac{h}{2} \left[ \frac{\partial}{\partial t} f_n + \frac{\partial}{\partial y} f_n \cdot f_n \right] - b_2 c_2 h \left[ \frac{\partial}{\partial t} f_n + \frac{\partial}{\partial y} f_n \cdot f_n \right] + O(h^2)
\]

Now, if

\[
\frac{h}{2} - b_2 c_2 h = 0 \quad \Leftrightarrow \quad 2 b_2 c_2 = 1
\]

we get \( \text{LTE}(h) \sim O(h^2) \), i.e. our 2-stage RK-method is second order. The corresponding family of Butcher arrays is

\[
\begin{array}{c|ccc}
0 & 0 & 0 \\
c_2 & c_2 & 0 \\
\hline
& 1 - 1/(2 c_2) & 1/(2 c_2)
\end{array}
\]

Sanity check: \( c_2 = 1/2 \) gives Euler’s Midpoint Method, and \( c_2 = 1 \) gives Heun’s Method.
Runge-Kutta Methods: Issues to clear up...

- Error Estimation using Richardson’s Extrapolation
- Error Analysis
  - LTE($h$)
  - consistency
- Stability Analysis
In addition to computing the numerical solution, we also need an estimate on the quality of the solution — an error estimate.
In addition to computing the numerical solution, we also need an estimate on the quality of the solution — an error estimate.

Suppose we have used a Runge-Kutta method (with step-size $h_1 = h$) of order $p$ to get the numerical solution $y^*_n$ at $t_{n+1}$, then the local error in the solution is:

$$e^* = y(t_{n+1}) - y^*_n = Ch^{p+1} + O(h^{p+2})$$
In addition to computing the numerical solution, we also need an estimate on the quality of the solution — an error estimate.

Suppose we have used a Runge-Kutta method (with step-size $h_1 = h$) of order $p$ to get the numerical solution $y_{n+1}^*$ at $t_{n+1}$, then the local error in the solution is:

$$e^* = y(t_{n+1}) - y_{n+1}^* = C h^{p+1} + O(h^{p+2})$$

If we have another solution $y_{n+1}^{**}$, computed with $h_2 = h/2$,

$$e^{**} = y(t_{n+1}) - y_{n+1}^{**} = C \left[ \frac{h}{2} \right]^{p+1} + O(h^{p+2})$$
Estimating the Error “on the fly”

Keeping only the leading order (principal part, $h^{p+1}$-term) of the error expansion we can write:

$$y_{n+1}^{**} - y_{n+1}^* = -Ch^{p+1} \left[ \frac{1}{2p+1} - 1 \right]$$

We have

$$y_{n+1}^{**} - y_{n+1}^* = -C h^{p+1} \left[ \frac{1}{2p+1} - 1 \right] = -C \left[ \frac{h}{2} \right]^{p+1} \left[ 1 - 2^{p+1} \right]$$
Thus,

\[ C \left( \frac{h}{2} \right)^{p+1} e^{**} = \frac{y^{**}_{n+1} - y^{*}_{n+1}}{2^{p+1} - 1} \]

is an estimate for principal local truncation error (PLTE).

This works well in practice. The only problem is that it is expensive to implement — 3 times the evaluations of the slope \( f(t, y) \) (a total of 12 evaluations for Runge’s 4th order scheme) — 200% overhead.
Finding a More Efficient Error Estimate

It’d be great if we could find an error estimate directly from the computed slopes (the $k_i$’s)...
Finding a More Efficient Error Estimate

It’d be great if we could find an error estimate directly from the computed slopes (the $k_i$’s)...

This idea was introduced by Merson in 1957. The idea is to derive two Runge-Kutta methods of orders $p$ and $p + 1$ using the same set of $k_i$’s... In terms of the Butcher array:

\[
\begin{array}{c|c}
\tilde{c} & A \\
\tilde{b}^T & \\
\tilde{b}_2^T & \tilde{E}^T \\
\end{array}
\]

Where $(A, \tilde{c}, \tilde{b})$ defines a method of order $p$, and $(A, \tilde{c}, \tilde{b}_2)$ a method of order $p + 1$. The vector $\tilde{E}^T = \tilde{b}_2 - \tilde{b}$, and the error estimate is given by $h \sum_{i=1}^{s} E_i k_i$. 

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩
The most commonly seen 4th-5th order method is RKF45:

\[
\begin{array}{c|cccccccc}
\tilde{c} & A & & & & & & & \\
\hline
0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
\frac{1}{4} & 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \frac{1}{4} \\
\frac{3}{8} & 3 & \frac{9}{32} & \cdots & \cdots & \cdots & \cdots & \frac{3}{8} \\
\frac{12}{13} & 1932 & -7200 & 7296 & \cdots & \cdots & \cdots & \frac{12}{13} \\
\frac{25}{216} & 439 & -8 & 3680 & -845 & \cdots & \cdots & \frac{25}{216} \\
\frac{1}{2} & -8 & 2 & -3544 & 1859 & -11 & 0 & \frac{1}{2} \\
\frac{16}{135} & 1408 & -2197 & -1 & 0 & \frac{16}{135} \\
\frac{1}{360} & 128 & -2197 & 1 & 2 & \frac{1}{360} \\
\end{array}
\]

RKF45 uses 6 evaluations of \( f(t, y) \) to obtain a 4th order method with an error estimate — **50% overhead**.
Stability Analysis of RK-methods

By applying the RK-methods to the scalar test-problem
\[ y'(t) = \lambda y(t), \quad y(t_0) = y_0 \]
we will find the regions of stability for the methods.

Consider Heun’s Method

\[
\begin{array}{c|cc}
  c_1 & a_{1,1} & a_{1,2} \\
  c_2 & a_{2,1} & a_{2,2} \\
  b_1 & b_2 \\
\end{array} = \begin{array}{ccc}
  0 & 0 & 0 \\
  1 & 1 & 0 \\
  1/2 & 1/2 \\
\end{array}
\]

Hence

\[
\begin{align*}
  k_1 &= f(t_n, y_n) = \lambda y_n \\
  k_2 &= f(t_n + h, y_n + hk_1) = \lambda (y_n + hk_1) = \lambda y_n + h\lambda^2 y_n \\
  y_{n+1} &= y_n \left[ 1 + \frac{h}{2} \left[ 2\lambda + h\lambda^2 \right] \right] = y_n \left[ 1 + h\lambda + \frac{(h\lambda)^2}{2} \right]
\end{align*}
\]
The stability region is given by

$$|R(h\lambda)| = \left| 1 + h\lambda + \frac{(h\lambda)^2}{2} \right| \leq 1$$

We find the boundary of the region by finding the complex roots of

$$1 - e^{i\theta} + h\lambda + \frac{(h\lambda)^2}{2} = 0, \quad \forall \theta \in [0, 2\pi)$$
For notational convenience we absorb \( h\lambda \rightarrow \hat{h} \).

Using the \( A \) from the Butcher array, we can write the \( k_i \)'s

\[
\tilde{k} = \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_s \end{bmatrix} = y_n \tilde{1} + \hat{h}A\tilde{k}, \quad \text{where} \quad \tilde{1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \text{ s ones}
\]

thus, we can solve for \( \tilde{k} \):

\[
\tilde{k} = (I - \hat{h}A)^{-1} \tilde{1} y_n
\]

Further,

\[
y_{n+1} = y_n + \hat{h}\tilde{b}^T\tilde{k} = y_n + \hat{h}\tilde{b}^T(I - \hat{h}A)^{-1} \tilde{1} y_n
\]
We have
\[ y_{n+1} = y_n + \hat{h} \tilde{b}^T \tilde{k} = y_n + \hat{h} \tilde{b}^T (I - \hat{h}A)^{-1} \tilde{1} y_n \]

Thus, the stability function is
\[ R(\hat{h}) = 1 + \hat{h} \tilde{b}^T (I - \hat{h}A)^{-1} \tilde{1} \]

As usual, the method is stable for \( \hat{h} \) such that \( |R(\hat{h})| \leq 1 \).

For explicit methods, \( A \) strictly lower triangular, the quantity
\[ \tilde{d} = (I - \hat{h}A)^{-1} \tilde{1} \]

is easily computable using forward substitution.
Stability Region for RKF45

\[ R(\hat{h}) = 1 + \hat{h} + \frac{\hat{h}^2}{2} + \frac{\hat{h}^3}{6} + \frac{\hat{h}^4}{24} + \frac{\hat{h}^5}{104} \]
An RK-method

\[
\frac{y_{n+1} - y_n}{h} = \sum_{i=1}^{s} b_i k_i
\]

where

\[
k_i = f \left( t_i + c_i h, y_n + h \sum_{j=1}^{s} a_{ij} k_j \right)
\]

is consistent with the ODE, \( y'(t) = f(t, y) \), if and only if \( \sum b_i = 1 \).
“Proof” by vigorous hand-waving

We note that each \( k_i = f(t_n, y_n) + O(h) \). Hence we have

\[
LTE(h) = (1 - \sum b_i)f(t, y) + O(h).
\]

Since we need

\[
\lim_{h \to 0} LTE(h) = 0,
\]

we must have \( 1 - \sum b_i = 0. \) \( \Box \)
Homework #2, Due 11:00am, 2/20/2015

1. Find the stability function for Runge’s 4th-order 4-stage method.

2. Implement RKF45 (don’t use matlab’s ode45!). Solve

\[
\begin{align*}
  y'(t) &= y(t) + 2t - 1 \\
  y(0) &= 1 \\
  t &\in [0, 1]
\end{align*}
\]

with step-length \( h \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}\} \).

Plot the exact, and estimated errors at the terminating point \((t = 1)\) vs. the step-length \( h \) on a log-log scale (in matlab: `loglog(the_h_values, the_exact_errors, '-o', the_h_values, the_estimated_errors, '-*')`).
1895  The idea of multiple evaluations of the derivative for each
time-step is attributed to Runge.

1900  Heun makes several contributions.

1901  Kutta characterizes the set of Runge-Kutta methods of order
4; proposed the first order 5 method.

1925  Nyström proposes special methods for second order ODEs.

1956  Huta introduces 6th order methods.

Modern analysis of Runge-Kutta methods developed by

1951  Gill

1957  Merson

1963  Butcher
s-stage Runge-Kutta for \( \{ y'(t) = f(t, y), \ y(t_0) = y_0 \} \)

The Butcher array for a general s-stage RK method is

\[
\begin{array}{c|cccc}
  c_1 & a_{1,1} & a_{1,2} & \cdots & a_{1,s} \\
  c_2 & a_{2,1} & a_{2,2} & \cdots & a_{2,s} \\
  \vdots & \vdots & \vdots & & \vdots \\
  c_s & a_{s,1} & a_{s,2} & \cdots & a_{s,s} \\
\hline
  b_1 & b_2 & \cdots & b_s
\end{array}
\]

is a compact shorthand for the scheme

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

where the \( k_i \)s are multiple estimates of the right-hand-side \( f(t, y) \)

\[
k_i = f \left( t_n + c_i h, \ y_n + h \sum_{j=1}^{s} a_{i,j} k_j \right), \quad i = 1, 2, \ldots, s
\]
Conditions on the Butcher Array

The Butcher array must satisfy the following row-sum condition

\[ c_i = \sum_{j=1}^{s} a_{i,j} \quad i = 1, 2, \ldots, s \]

and consistency requires

\[ \sum_{j=1}^{s} b_j = 1. \]

Beyond that, we are left with the formidable task of selecting \( \tilde{b}, \tilde{c}, \) and the matrix \( A \). Up to this point our only tool is (tedious) Taylor expansions.
Explicit 3-stage RK Methods

If we want to build an explicit 3-stage method,

\[
\begin{array}{c|ccc}
0 & \ & \ & \\
c_2 & a_{21} & \ & \\
c_3 & a_{31} & a_{32} & \\
\hline & b_1 & b_2 & b_3
\end{array}
\]

it can be shown (Taylor expansion) that in order to achieve a 3rd order scheme, we must satisfy the **Order Conditions**:

\[
\begin{align*}
b_1 + b_2 + b_3 &= 1 \\
b_2c_2 + b_3c_3 &= \frac{1}{2} \\
b_2c_2^2 + b_3c_3^2 &= \frac{1}{3} \\
b_3a_{32}c_2 &= \frac{1}{6}
\end{align*}
\]
Finding the Order Conditions

Clearly, deriving a Runge-Kutta scheme boils down to a two-stage process:

1. Find the order conditions: — a set of non-linear equations in the parameters sought.
2. Find a solution, or family of solutions, to the order conditions.

As the desired order of the method increases, both deriving and solving these algebraic conditions become increasingly complicated.

We now consider a structured way of deriving the order conditions without explicit Taylor expansions.
Rooted Trees

Definition (Rooted Tree)

A rooted tree is a graph, which is connected, has no cycles, and has one vertex designated as the root.

Definition (Order of a Rooted Tree)

The order of a rooted tree is the number of vertices in the tree.

Definition (Leaves)

A leaf is vertex in a tree (with order greater than one) which has exactly one vertex joined to it.
Examples: Trees

**Figure:** Trees of order 2, 3, 4, 5, and 8. By convention, we place the root at the bottom of the graph, and let the tree grow “upward.”
For each tree $t$, we define two quantities

1. $\Phi(t)$: a polynomial in the coefficients which will define a Runge-Kutta method.
2. $\gamma(t)$: an integer
We label each vertex of the tree, except the leaves, e.g.
We label each vertex of the tree, except the leaves, e.g.

Next, we write down a sequence of factors, starting with $b_i$ (the root factor). For each arc of the tree, write down a factor $a_{jk}$ where $j$ and $k$ are the beginning and end of the arc (in the sense up upward growth). Finally, for the leaves write down a factor $c_j$, where $j$ is the label attached to the beginning of the arc: e.g.
Now, sum the product of these factors, for all possible choices of the labels \{1, 2, \ldots, s\}:

\[
\Phi(t) = \sum_{ij} b_i c_i^2 a_{ij} c_j^2
\]
Building $\gamma(t)$

In order to build $\gamma(t)$, we associate a factor with each vertex in the tree:

- The factor for the leaves is 1.
- For all other vertices, the factor is 1 added to the sum of the factors of the upward growing neighbors.

Figure: $\gamma(t)$ is the product of all the factors, here $\gamma(t) = 6 \cdot 3 \cdot 1^4 = 18$. 

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩ Runge-Kutta Methods, Continued
### Rooted Trees Up to Order 4

<table>
<thead>
<tr>
<th>Tree Order</th>
<th>Tree</th>
<th>( \Phi(t) )</th>
<th>( \gamma(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image" alt="Tree 1" /></td>
<td>( \sum_i b_i )</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td><img src="image" alt="Tree 2" /></td>
<td>( \sum_i b_i c_i )</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td><img src="image" alt="Tree 3" /></td>
<td>( \sum_i b_i c_i^2 )</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td><img src="image" alt="Tree 4" /></td>
<td>( \sum_i b_i c_i^3 )</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tree Order</th>
<th>Tree</th>
<th>( \Phi(t) )</th>
<th>( \gamma(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td><img src="image" alt="Tree 3" /></td>
<td>( \sum_{ij} b_i a_{ij} c_j )</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td><img src="image" alt="Tree 4" /></td>
<td>( \sum_{ij} b_i a_{ij} c_j^2 )</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td><img src="image" alt="Tree 4" /></td>
<td>( \sum_{ijk} b_i a_{ij} a_{jk} c_k )</td>
<td>24</td>
</tr>
</tbody>
</table>

**Definitions**

The Quantities \( \Phi(t) \), and \( \gamma(t) \)

Designing a Runge-Kutta Scheme Based on \( \Phi(t) \) and \( \gamma(t) \)

---

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩

Runge-Kutta Methods, Continued
In designing an $s$-stage RK-method, the coefficients must satisfy

$$
\Phi(t) = \frac{1}{\gamma(t)}, \quad \forall t : \text{order}(t) \leq s
$$
Runge-Kutta Scheme Based on $\Phi(t)$ and $\gamma(t)$: 4-stage Example

A 4-stage explicit scheme, where $a_{ij} = 0$ whenever $i \geq j$, thus yields 8 conditions for $\{b_1, b_2, b_3, b_4, c_2, c_3, c_4, a_{32}, a_{42}, a_{43}\}$:

\[
\begin{align*}
    b_1 + b_2 + b_3 + b_4 &= 1 \quad (1) \\
    b_2 c_2 + b_3 c_3 + b_4 c_4 &= \frac{1}{2} \quad (2) \\
    b_2 c_2^2 + b_3 c_3^2 + b_4 c_4^2 &= \frac{1}{3} \quad (3) \\
    b_3 a_{32} c_2 + b_4 a_{42} c_2 + b_4 a_{43} c_3 &= \frac{1}{6} \quad (4) \\
    b_2 c_2^3 + b_3 c_3^3 + b_4 c_4^3 &= \frac{1}{4} \quad (5) \\
    b_3 c_3 a_{32} c_2 + b_4 c_4 a_{42} c_2 + b_4 c_4 a_{43} c_3 &= \frac{1}{8} \quad (6) \\
    b_3 a_{32} c_2^2 + b_4 a_{42} c_2^2 + b_4 a_{43} c_3^2 &= \frac{1}{12} \quad (7) \\
    b_4 a_{43} a_{32} c_2 &= \frac{1}{24} \quad (8)
\end{align*}
\]
Kutta identified five cases where a solution to this non-linear system is guaranteed to exist:

**Case 1** \( c_2 \not\in \{0, \frac{1}{2}, \frac{1}{2} \pm \frac{\sqrt{3}}{6}\} \), \( c_3 = 1 - c_2 \)

**Case 2** \( b_2 = 0, \ c_2 \neq 0, \ c_3 = \frac{1}{2} \)

**Case 3** \( b_3 \neq 0, \ c_2 = \frac{1}{2}, \ c_3 = 0 \)

**Case 4** \( b_4 \neq 0, \ c_2 = 1, \ c_3 = \frac{1}{2} \)

**Case 5** \( b_3 \neq 0, \ c_2 = c_3 = \frac{1}{2} \)
The number of rooted trees of order $s$ increases rapidly as $s$ goes beyond 4. For $s = 5$ we have the following 9 rooted trees:

Each which leads to a nonlinear condition. (Fun!)
Beyond 4 Stages...

The number of rooted trees of order $s$ increases rapidly as $s$ goes beyond 4. For $s = 5$ we have the following 9 rooted trees:

![Rooted Trees](image)

Each which leads to a nonlinear condition. (Fun!)

For $s \in \{6, 7, 8, 9, 10\}$ we get \{20, 48, 115, 286, 719\} corresponding rooted trees.
Beyond 4 Stages...

**Theorem (Butcher, 2008: p.187)**

*If an explicit s-stage Runge-Kutta method has order p, then s ≥ p.*

**Theorem (Butcher, 2008: p.187)**

*If an explicit s-stage Runge-Kutta method has order p ≥ 5, then s > p.*

**Theorem (Butcher, 2008: p.188)**

*For any positive integer p, an explicit Runge-Kutta method exists with order p and s stages, where*

\[
s = \begin{cases} 
\frac{3p^2 - 10p + 24}{8}, & p = 2k, \ k \in \mathbb{Z} \\
\frac{3p^2 - 4p + 9}{8}, & p = 2k + 1, \ k \in \mathbb{Z}
\end{cases}
\]
Note that the theorem gives an upper bound for the number of required stages (the theorem gives guarantees). The bound grows very quickly.

For certain values of $p$, $s$-stage methods with $s$ lower than this bound are known:

<table>
<thead>
<tr>
<th>Order, $p$ =</th>
<th>5 6 7 8 9 10 11 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stages, $s$ =</td>
<td>8 9 16 17 27 28 41 42</td>
</tr>
<tr>
<td>Scheme, $s$ =</td>
<td>6 7 9 11 17</td>
</tr>
</tbody>
</table>

Project, anyone?
Stability Polynomials, Comments

With every explicit Runge-Kutta method, we can find a stability polynomial $R(h\lambda)$ for which the condition $|R(h\lambda)| \leq 1$ defines the region of stability.

We know that for orders $p = 1, 2, 3, 4$ there are explicit $s$-stage RK-methods with $s = p$, and for higher order methods $s > p$.  

<table>
<thead>
<tr>
<th>Order</th>
<th>Stages</th>
<th>Stability Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$R(z) = 1 + z$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$R(z) = 1 + z + \frac{1}{2}z^2$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>$R(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3$</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$R(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4$</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>$R(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4 + \frac{1}{120}z^5 + Cz^6$</td>
</tr>
</tbody>
</table>

Where, in the case $p = 5$, $s = 6$, the constant $C$ depends on the particular method.
Fact

Since the stability function $R(z)$ is a polynomial for all explicit Runge-Kutta methods, it is never possible to build such a method with unbounded region of stability.
Butcher (2008) develops the theory of rooted trees and their usefulness far beyond what is indicated in the current lecture.

I have deliberately taken a very narrow path through the material and only presented some key ideas that fit into the context of what we have explored so far (Low-order explicit methods).

Some completely ignored topics include

- Two alternative, non-graphical, notations for trees.
- Expression of higher order derivatives in terms of rooted trees.
- Expression of ODEs (linear and non-linear) using rooted trees.
For the mathematically inclined, the study of Runge-Kutta methods have several interesting connections to areas of mathematics which we sometimes consider “less applied,” e.g.

- Graph theory
- Group theory

Also, in the context of step-size ($h$) management, there are some overlap with ideas in

- Control theory

We will revisit some of these topics, as needed, in future lectures.