AM213B Numerical Methods for the Solution of Differential Equations

Lecture 03

List of topics in this lecture

- Consistency, convergence, order of accuracy
- Theorem: consistency + stability = convergence
- Runge-Kutta methods, general form, Butcher tableau
- Embedded Runge-Kutta methods, extended Butcher tableau
- Condition for first order and additional condition for second order accuracy

---

Consistency, convergence, order of accuracy

Previously, we established the theorem below.

Theorem:

For a stable numerical method, if the local truncation error is $e_n(h) = O(h^{p+1})$ then the global error is $E_N(h) = O(h^p)$.

That is, the global error is one order lower than the local truncation error.

This theorem motivates us to define consistency and convergence as follows.

Definition of consistency:

We say a numerical method is consistent with the differential equation if the local truncation error satisfies

$$\lim_{h\to 0}\frac{e_n(h)}{h}=0$$

That is, $e_n(h) = O(h^{p+1})$ with $p > 0$.

Definition of convergence, order of accuracy:

We say a numerical method is convergent if the global error satisfies

$$\lim_{h\to 0} E_N(h)=0$$

That is, $E_N(h) = O(h^p)$ with $p > 0$. Here p is called the order of accuracy.

Remark:

The order of a numerical method is the order of global error $E_N(h) \sim e_n(h)/h$.

It is NOT the order of local truncation error $e_n(h)$.

Example:

Euler and backward Euler methods are both first order.

$e_n(h) = O(h^2)$ ==> $E_N(h) = O(h)$

Trapezoidal method is second order.

$e_n(h) = O(h^3)$ ==> $E_N(h) = O(h^2)$

Now we recast the theorem we previously established into a simple/concise form

Theorem:

Consistency + Stability ==> Convergence

Runge-Kutta methods

Recall that the trapezoidal method is implicit.

Trapezoidal method:

$$u_{n+1} = u_n + \frac{h}{2}\left(f(u_n, t_n) + f(u_{n+1}, t_{n+1})\right) \qquad \text{(E01)}$$

We like to develop an explicit version of the trapezoidal method.

Goal: to maintain the second order and to make it an explicit method.

To maintain $E_N(h) = O(h^2)$, we need to maintain $e_n(h) = O(h^3)$

----> we are allowed an error of $O(h^3)$ in approximating $h\,f(u_{n+1}, t_{n+1})$.

----> we are allowed an error of $O(h^2)$ in approximating u_{n+1} in $f(u_{n+1}, t_{n+1})$.

Strategy:

We use Euler method to predict a value of u_{n+1} with an error of $O(h^2)$.

We use the predicted value of u_{n+1} to approximate $f(u_{n+1}, t_{n+1})$ and then use it in (E01) to calculate a more accurate approximation of u_{n+1}.

Euler ----> $u_{n+1}^{(Predicted)}$

----> use $f(u_{n+1}^{(Predicted)}, t_{n+1})$ in (E01) to calculate $u_{n+1}^{(Corrected)}$.

This gives us an explicit version of the trapezoidal method.

Predictor-Corrector method (Heun's method):

$$\tilde{u}_{n+1} = u_n + hf(u_n, t_n)$$
$$u_{n+1} = u_n + \frac{h}{2}\Big(f(u_n, t_n) + f(\tilde{u}_{n+1}, t_{n+1})\Big)$$

- It is explicit.
- It is a single-step method.
- It is a second order method: $e_n(h) = O(h^3)$.

We write it in a different form by introducing $k_1 = hf(u_n, t_n)$, $k_2 = hf(\tilde{u}_{n+1}, t_{n+1})$.

Predictor-Corrector method in Runge-Kutta form:

$$k_1 = hf(u_n, t_n)$$
$$k_2 = hf(u_n + k_1, t_n + h)$$
$$u_{n+1} = u_n + \frac{1}{2}k_1 + \frac{1}{2}k_2$$

With this example in mind, we now introduce the general form of Runge-Kutta methods.

The general form of explicit Runge-Kutta methods

$$k_i = hf\left(u_n + a_{i1}k_1 + a_{i2}k_2 + \cdots + a_{i(i-1)}k_{i-1}, t_n + c_i h\right), \qquad i = 1, \ldots, p$$
$$u_{n+1} = u_n + b_1 k_1 + b_2 k_2 + \cdots + b_p k_p$$

In the summation notation, it is

$$k_i = hf\left(u_n + \sum_{j=1}^{i-1} a_{ij}k_j, t_n + c_i h\right), \qquad i = 1, \ldots, p$$
$$u_{n+1} = u_n + \sum_{i=1}^{p} b_i k_i$$

The general form of implicit Runge-Kutta methods

$$k_i = hf\left(u_n + \sum_{j=1}^{p} a_{ij}k_j, t_n + c_i h\right), \qquad i = 1, \ldots, p$$
$$u_{n+1} = u_n + \sum_{i=1}^{p} b_i k_i$$

Remark:

When $a_{i\,j} = 0$ for all $j \geq i$, each k_i has no dependence on k_j for $j \geq i$. We can calculate $\{k_i\}$ sequentially starting at k_1. The Runge-Kutta method is explicit.

A Runge-Kutta method is completely specified by 4 items:

p: number of stages (an integer)

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1p} \\ \vdots & & \vdots \\ a_{p1} & \cdots & a_{pp} \end{bmatrix}$$

a $p \times p$ matrix of coefficients in calculation of k_i

$$c = \begin{bmatrix} c_1 & \cdots & c_p \end{bmatrix}$$

a p-component vector of coefficients in $t_n + c_i\, h$

$$b = \begin{bmatrix} b_1 & \cdots & b_p \end{bmatrix}$$

a p-component vector of coefficients in calculation of u_{n+1}

Remarks:

- A Runge-Kutta method is a single-step method.
- A Runge-Kutta method is explicit if and only if matrix A is strictly lower triangular:

  $a_{i\,j} = 0$ for $j \geq i$,

$$A = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{p,1} & \cdots & a_{p,(p-1)} & 0 \end{bmatrix}$$

Butcher tableau

A Runge-Kutta method is visually represented by the "Butcher tableau"

Butcher tableau: $\begin{array}{c|c} c^T & A \\ \hline & b \end{array}$

Example:

Euler, backward Euler, trapezoidal, and predictor-corrector methods are all Runge-Kutta methods.

Euler method:

$$\begin{cases} k_1 = hf(u_n, t_n) \\ u_{n+1} = u_n + k_1 \end{cases}$$

$p=1,\quad A=\begin{pmatrix}0\end{pmatrix},\quad c=\begin{pmatrix}0\end{pmatrix},\quad b=\begin{pmatrix}1\end{pmatrix}$

Butcher tableau: $\begin{array}{c|c} c^T & A \\ \hline & b \end{array} = \begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$

It is one-stage, first order, explicit.

Backward Euler method:

$$\begin{cases} k_1 = hf(u_n + k_1, t_n + h) \\ u_{n+1} = u_n + k_1 \end{cases}$$

$p=1,\quad A=\begin{pmatrix}1\end{pmatrix},\quad c=\begin{pmatrix}1\end{pmatrix},\quad b=\begin{pmatrix}1\end{pmatrix}$

Butcher tableau: $\begin{array}{c|c} c^T & A \\ \hline & b \end{array} = \begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$

It is one-stage, first order, implicit.

Trapezoidal method:

$$\begin{cases} k_1 = hf(u_n, t_n) \\ k_2 = hf\left(u_n + \frac{1}{2}k_1 + \frac{1}{2}k_2, t_n + h\right) \\ u_{n+1} = u_n + \frac{1}{2}k_1 + \frac{1}{2}k_2 \end{cases}$$

$p=2,\quad A=\begin{pmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix},\quad c=\begin{pmatrix} 0 & 1 \end{pmatrix},\quad b=\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix}$

Butcher tableau: $\begin{array}{c|c} c^T & A \\ \hline & b \end{array} = \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$

It is two-stage, second order, implicit.

Predictor-corrector method:

$$\begin{cases} k_1 = hf(u_n, t_n) \\ k_2 = hf(u_n + k_1, t_n + h) \\ u_{n+1} = u_n + \frac{1}{2}k_1 + \frac{1}{2}k_2 \end{cases}$$

$$p = 2, \quad A = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad c = \begin{pmatrix} 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Butcher tableau:

$$\begin{array}{c|c} c^T & A \\ \hline & b \end{array} = \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

It is two-stage, second order, explicit.

More explicit Runge-Kutta methods

Midpoint Runge-Kutta method (RK2): (Skip in lecture)

$$\begin{cases} k_1 = hf(u_n, t_n) \\ k_2 = hf\left(u_n + \frac{1}{2}k_1, t_n + \frac{1}{2}h\right) \\ u_{n+1} = u_n + k_2 \end{cases}$$

Butcher tableau:

$$\begin{array}{c|c} c^T & A \\ \hline & b \end{array} = \begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1 \end{array}$$

It is two-stage, second order, explicit.

Note: This is NOT the same as the two-step midpoint method.

Kutta's third order method (RK3): (Skip in lecture)

$$\begin{cases} k_1 = hf(u_n, t_n) \\ k_2 = hf\left(u_n + \frac{1}{2}k_1, t_n + \frac{1}{2}h\right) \\ k_3 = hf\left(u_n - k_1 + 2k_2, t_n + \frac{1}{2}h\right) \\ u_{n+1} = u_n + \frac{1}{6}k_1 + \frac{2}{3}k_2 + \frac{1}{6}k_3 \end{cases}$$

Butcher tableau: $\begin{array}{c|c} c^T & A \\ \hline & b \end{array} = \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & -1 & 2 & 0 \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$

It is three-stage, third order, explicit.

Classic 4-th order method (RK4, this is the original Runge-Kutta method):

$$\begin{cases} k_1 = hf(u_n, t_n) \\ k_2 = hf\left(u_n + \frac{1}{2}k_1, t_n + \frac{1}{2}h\right) \\ k_3 = hf\left(u_n + \frac{1}{2}k_2, t_n + \frac{1}{2}h\right) \\ k_4 = hf\left(u_n + k_3, t_n + h\right) \\ u_{n+1} = u_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \end{cases}$$

Butcher tableau: $\begin{array}{c|c} c^T & A \\ \hline & b \end{array} = \begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$

It is 4-stage, 4th-order, explicit.

Embedded Runge-Kutta methods

An embedded Runge-Kutta method is a combination of two Runge-Kutta methods that share the same function evaluations of $f(u, t)$.

- They share the same matrix A and the same k_i's to minimize the number of function evaluations (i.e., minimizing the computational cost).
- Two different versions of vector b produce two methods: a higher order method and a lower order method.

- The numerical results of the two methods allow us to estimate errors dynamically and control errors with adaptive time steps.

Higher order method: $u_{n+1} = u_n + \sum_{i=1}^{p} b_i k_i$

Lower order method: $\tilde{u}_{n+1} = u_n + \sum_{i=1}^{p} \tilde{b}_i k_i$

At each time step, the error of one step is estimated as

$$e_n(h) = u_{n+1} - \tilde{u}_{n+1}$$

Based on the estimated error, we decide if we reduce or increase the time step. This approach is called adaptive time step.

Extended Butcher tableau

Embedded Runge-Kutta methods are represented by "Extended Butcher tableau"

Extended Butcher tableau:

c^T	A
	b
	$\tilde{b}$

Heun-Euler method (with orders 1 and 2):

Extended Butcher tableau:

c^T	A
	b
	$\tilde{b}$

$=$

0	0	0
1	1	0
0	$\frac{1}{2}$	$\frac{1}{2}$
	1	0

Fehlberg 45 method (with orders 4 and 5): Extended Butcher tableau:

c^T	A
	b
	$\tilde{b}$

$=$

0	0	0	0	0	0	0
$\frac{1}{4}$	$\frac{1}{4}$	0	0	0	0	
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$	0	0	0	0
$\frac{12}{13}$	$\frac{1932}{2197}$	$\frac{-7200}{2197}$	$\frac{7296}{2197}$	0	0	0
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$\frac{-845}{4104}$	0	0
$\frac{1}{2}$	$\frac{-8}{27}$	2	$\frac{-3544}{2565}$	$\frac{1859}{4104}$	$\frac{-11}{40}$	0
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$\frac{-9}{50}$	$\frac{2}{55}$
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$\frac{-1}{5}$	0

Highest order of explicit Runge-Kutta methods at a fixed number of stages

Based on the methods we looked at so far, it seems that for explicit Runge-Kutta methods, the highest order of accuracy that can be achieved at a fixed number of stages is the same as the number of stages.

One-stage ----> first order

Two-stage ----> second order

Three-stage ----> third order

Four-stage ----> fourth order

However, this conjecture is NOT true. With 5 stages, it is not possible to design a 5-th order explicit Runge-Kutta method. To obtain a 5-th order Runge-Kutta method, we need at least 6 stages. It took a while for the numerical analysis community to prove this and the proof is outside the scope of this course.

The table below shows the highest order of explicit Runge-Kutta methods that can be achieved at each given number of stages.

Table: Highest order explicit Runge-Kutta at each fixed number of stages

Number of stages	Highest order explicit Runge-Kutta method
1	1
2	2
3	3
4	4
5	4
6	5
7	6
8	6
9	7
10	7
11	8

Note: The table gives the highest order explicit Runge-Kutta. For implicit Runge-Kutta, the order of accuracy can go higher, much higher.

Condition for the first order (consistency condition) and additional condition for the second order

Below we derive the conditions on coefficients $\{a_{i\,j}, c_i, b_i\}$, respectively,

- for achieving the first order (consistency), and
- for achieving the second order.

The local truncation error (LTE) of a Runge-Kutta method is defined as

$$e_n(h) = u(t_n + h) - u(t_n) - \sum_{i=1}^{p} b_i k_i$$

Taylor expansion of $u(t_{n+1})$ around t_n gives us

$$u(t_n + h) = u(t_n) + u'(t_n)h + u''(t_n)\frac{h^2}{2!} + O(h^3)$$

Using the differential equation, we write $u'(t_n)$ and $u''(t_n)$ as

$$u'(t_n) = f\left(u(t), t\right)\Big|_{t=t_n} \equiv f\Big|_{t=t_n}$$

$$u''(t_n) = \frac{df\left(u(t), t\right)}{dt}\Bigg|_{t=t_n} = \left(\frac{\partial f\left(u(t), t\right)}{\partial u} u'(t) + \frac{\partial f\left(u(t), t\right)}{\partial t} \right)\Bigg|_{t=t_n}$$

$$= \left(\frac{\partial f\left(u(t), t\right)}{\partial u} f\left(u(t), t\right) + \frac{\partial f\left(u(t), t\right)}{\partial t} \right)\Bigg|_{t=t_n} \equiv \left(\frac{\partial f}{\partial u} f + \frac{\partial f}{\partial t} \right)\Bigg|_{t=t_n}$$

With these expressions of $u'(t_n)$ and $u''(t_n)$, we write the Taylor expansion as

$$u(t_n + h) = u(t_n) + hf\Big|_{t=t_n} + \frac{h^2}{2!}\left(\frac{\partial f}{\partial u} f + \frac{\partial f}{\partial t} \right)\Bigg|_{t=t_n} + O(h^3)$$

Substituting it into the expression of $e_n(h)$, we get

$$e_n(h) = hf\Big|_{t=t_n} + \frac{h^2}{2!}\left(\frac{\partial f}{\partial u} f + \frac{\partial f}{\partial t} \right)\Bigg|_{t=t_n} - \sum_{i=1}^{p} b_i k_i + O\left(h^3\right) \qquad \text{(E02)}$$

where k_i is given in the Runge-Kutta method

$$k_i = hf\left(u(t_n) + \sum_{j=1}^{p} a_{ij} k_j, t_n + c_i h \right) \qquad \text{(K03)}$$

Here in the calculation of LTE, u_n in k_i's is replaced with exact solution $u(t_n)$.

We expand k_i given by (K03) iteratively. First, we notice that $k_i = O(h)$.

Substituting $k_i = O(h)$ back into the RHS of (K03) leads to

$$k_i = hf\left(u(t_n) + O(h), t_n + c_i h\right) = hf\Big|_{t=t_n} + O(h^2) \qquad \text{(K03A)}$$

Using expansion (K03A) to calculate $e_n(h)$ in (E02), we get

$$e_n(h) = hf\Big|_{t=t_n}\left[1-\sum_{i=1}^{p} b_i\right] + O(h^2) \qquad \text{(E02A)}$$

To achieve the consistency (i.e., the first order), we need $e_n(h) = O(h^2)$.

Setting the coefficient of O(h) term to zero in (E02A), we obtain …

Condition for the first order (consistency condition):

$$\boxed{\sum_{i=1}^{p} b_i = 1}$$

To find coefficients of $O(h^2)$ terms in $e_n(h)$ in (E02), we need to calculate coefficients of $O(h^2)$ terms in k_i in (K03). We use an iterative approach to expand k_i.

We substitute $k_i = hf\Big|_{t=t_n} + O(h^2)$ back into the RHS of (K03).

$$k_i = hf\left(u(t_n) + hf\Big|_{t=t_n}\sum_{j=1}^{p} a_{ij},\ t_n + c_i h\right)$$

$$= h\left[f\Big|_{t=t_n} + \frac{\partial f}{\partial u}\Big|_{t=t_n}\left(hf\Big|_{t=t_n}\sum_{j=1}^{p} a_{ij}\right) + \frac{\partial f}{\partial t}\Big|_{t=t_n}(c_i h)\right] + O(h^3)$$

$$= hf\Big|_{t=t_n} + h^2\left[\left(\frac{\partial f}{\partial u} f\right)\Big|_{t=t_n}\cdot\sum_{j=1}^{p} a_{ij} + \frac{\partial f}{\partial t}\Big|_{t=t_n}\cdot c_i\right] + O(h^3) \qquad \text{(K03B)}$$

Using the new expansion (K03B) to calculate $e_n(h)$ in (E02), we obtain

$$\begin{aligned} e_n(h) = {} & hf\Big|_{t=t_n}\left[1-\sum_{i=1}^{p} b_i\right] + h^2\left(\frac{\partial f}{\partial u} f\right)\Big|_{t=t_n}\left[\frac{1}{2} - \sum_{i=1}^{p}\left(b_i\sum_{j=1}^{p} a_{ij}\right)\right] \\ & + h^2\frac{\partial f}{\partial t}\Big|_{t=t_n}\left[\frac{1}{2} - \sum_{i=1}^{p}\left(b_i c_i\right)\right] + O(h^3) \end{aligned} \qquad \text{(E02B)}$$

Setting the coefficient of $O(h^2)$ term to zero in (E02B), we obtain

Additional condition for the second order:

$$\boxed{\begin{aligned} &\sum_{i=1}^{p}\left(b_i\sum_{j=1}^{p} a_{ij}\right) = \frac{1}{2} \\ &\sum_{i=1}^{p}\left(b_i c_i\right) = \frac{1}{2} \end{aligned}}$$

Internal consistency condition:

$$c_i = \sum_{j=1}^{p} a_{ij}$$

Meaning of internal consistency condition:

Consider $k_i = hf\left(u(t_n) + \sum_{j=1}^{p} a_{ij} k_j, t_n + c_i h \right) \equiv hf(u^*, t^*)$.

Using $k_i = hf\big|_{t=t_n} + O(h^2) = u'(t_n)h + O(h^2)$, we write u^* and t^* as

$$u^* = u(t_n) + \sum_{j=1}^{p} a_{ij} k_j \approx u(t_n) + u'(t_n) h \sum_{j=1}^{p} a_{ij} \approx u\left(t_n + h \sum_{j=1}^{p} a_{ij} \right)$$

$$t^* = t_n + c_i h$$

The internal consistency condition means $u^* \approx u(t^*)$, which is something natural to impose although not necessary for consistency.

With the internal consistency condition satisfied, the additional condition for the second order becomes

Additional condition for the second order (with $c_i = \sum_{j=1}^{p} a_{ij}$)

$$\sum_{i=1}^{p} b_i c_i = \frac{1}{2}$$

Additional condition for the third order (with $c_i = \sum_{j=1}^{p} a_{ij}$)

$$\sum_{i=1}^{p} b_i c_i^{\,2} = \frac{1}{3}, \qquad \sum_{i=1}^{p} \sum_{j=1}^{p} b_i a_{ij} c_j = \frac{1}{6}$$

(Derivation not included)

Summary:

Condition for order p (with $c_i = \sum_{j=1}^{p} a_{ij}$):

p = 1: $\sum_{i=1}^{p} b_i = 1$

p = 2: $\sum_{i=1}^{p} b_i c_i = \frac{1}{2}$

$p = 3$: $\sum_{i=1}^{p} b_i c_i^{\,2} = \frac{1}{3}$, $\sum_{i=1}^{p}\sum_{j=1}^{p} b_i a_{ij} c_j = \frac{1}{6}$

Appendix: Behavior of the 2-step midpoint method:

$$u_{n+1} = u_{n-1} + 2h\, f(u_n, t_n)$$

We study its behavior in a model problem: $\begin{cases} u' = -u \\ u(0) = 1 \end{cases}$.

Exact solution of the model problem:

$$u(t) = \exp(-t)$$

which decays exponentially.

Midpoint method applied to the model problem is

$$u_{n+1} = u_{n-1} - 2h\,u_n$$

which is a difference equation.

We try solutions of the form: $u_n = r^n$.

This approach is motivated by that the exact solution of ODE is exponential.

We will see this approach again in subsequent lectures.

Substituting it into the difference equation, we get

$$r^{n+1} = r^{n-1} - 2hr^n$$

==> $r^2 + 2hr - 1 = 0$

The quadratic equation has two roots:

$$r_1 = \sqrt{1+h^2} - h, \qquad r_2 = -(\sqrt{1+h^2} + h)$$

A general solution of the difference equation is

$$u_n = c_1 r_1^{\,n} + c_2 r_2^{\,n}$$

For small h, we expand r, $\log(r)$ and r^n.

$$r_1 = \sqrt{1+h^2} - h = 1 - h + O(h^2)$$

$$\log r_1 = \log\left(1 - h + O(h^2)\right) = -h + O(h^2)$$

$$r_1^{\,n} = \exp\left(n \log r_1\right) = \exp\left(n[-h + O(h^2)]\right)$$

$$= \exp\left(-(nh) + (nh)O(h)\right), \qquad nh = t_n$$

$$= \exp(-t_n)\exp\left(O(h)\right) = \exp(-t_n)\left(1 + O(h)\right)$$

==> $r_1^{\,n} \approx \exp(-t_n)$

which corresponds to the exact solution.

For r_2 we have

$$-r_2 = \sqrt{1 + h^2} + h = 1 + h + O(h^2)$$

$$\log(-r_2) = \log\left(1 + h + O(h^2)\right) = h + O(h^2)$$

$$r_2^{\,n} = (-1)^n (-r_2)^n = (-1)^n \exp\left(n \log(-r_2)\right) = (-1)^n \exp\left(n\left[h + O(h^2)\right]\right)$$

$$= (-1)^n \exp\left((nh) + (nh)O(h)\right), \qquad nh = t_n$$

$$= (-1)^n \exp(t_n)\exp\left(O(h)\right) = (-1)^n \exp(t_n)\left(1 + O(h)\right)$$

==> $r_2^{\,n} \approx (-1)^n \exp(t_n)$

which grows exponentially and oscillates in sign.

Thus, a general solution of the difference equation has the behavior

$$u_n \approx c_1 \exp(-t_n) + c_2 (-1)^n \exp(t_n)$$

It has a decaying mode corresponding to the exact solution. In addition, it has a growing mode. The exponentially growing mode may start with negligible magnitude, excited by round-off error and local truncation error. But eventually, it will take over and ruin the whole numerical solution.