MA3457/CS4033:
Numerical Methods for Calculus and Differential Equations

Course Materials

P A R T IV

B'14
2014-2015

Introductory Notes

Motivation for Numerical Methods for ODEs

- *Engineering practice*, *science*, *real life* frequently bring to consideration *ODEs which are* ***complex enough*** *and* ***cannot be solved with the use of analytical techniques***.
- Or, they could be solved, but analytical solutions are *too complicated and require special treatment*.

Key Concept

Basic idea behind the techniques for 1st order ODEs:

- ***divide the interval of interest into discrete steps*** (of fixed length h) and ***find approximations to the function y at all the points*** $x_1, \dots x_n$:

x_1 x_2 ...

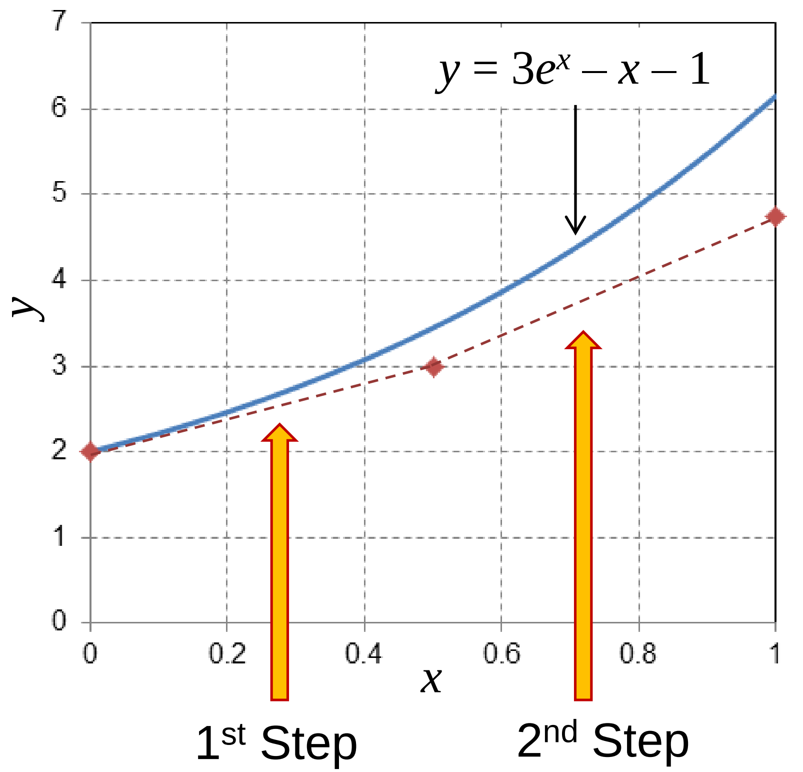
--|--o--o--o--o--o--|->

x_0 x_n

Euler's Method (1)

Exact Solution & First Two Steps of Euler's Method

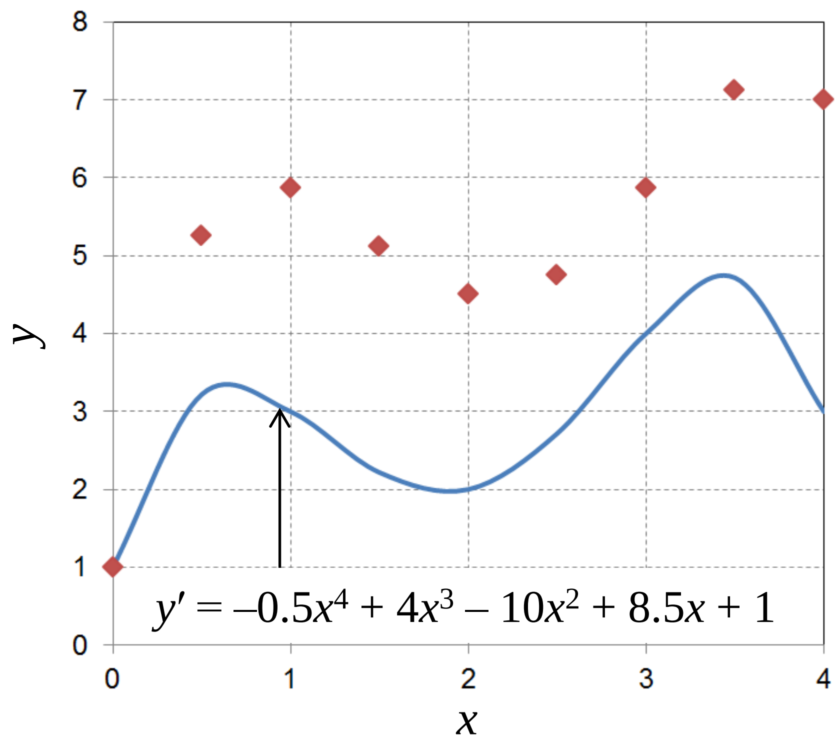
IVP: $y' = x + y,\ x_0 = 0,\ y_0 = 2;\ 0 \le x \le 1$



Euler's Method (2)

Exact Solution & Euler' Solution

<u>IVP</u>: $y' = -2x^3 + 12x^2 - 20x + 8.5;\ x_0 = 0,\ y_0 = 1;\ 0 \le x \le 4$



Euler's Method – MATLAB Script

LIBRARY OF MATLAB PROCEDURES

Euler

Solves differential equation $y' = f(x, y)$ with initial condition $y(a) = y_0$ on the interval $[a, b]$

```
function [x, y] = Euler(f, tspan, y0, n)
%
% The procedure solves d.e. y' = f(x,y) with initial
% condition y(a) = y0 using n steps of Euler's method.
%
% Step size: h = (b-a)/n
%
a = tspan(1); b = tspan(2); h = (b-a)/n;
x = (a+h : h : b);
y(1) = y0 + h*feval(f, a, y0);
%
for i = 2 : n
    y(i) = y(i-1) + h*feval(f, x(i-1), y(i-1));
end
%
x = [a x];
y = [y0 y];
```

```
[x, y] = Euler('fivp', int, ya, n)
```

```
function f_i = fivp(x, y)
f_i = x + y;
```

Applications of Euler (1)

IVP: $y' = x + y,\ x_0 = 0,\ y_0 = 2;\ 0 \le x \le 1$

n = 5

x	Appr. Sol.
0.2	2.4000
0.4	2.9200
0.6	3.5840
0.8	4.4208
1.0	5.4650

n = 10

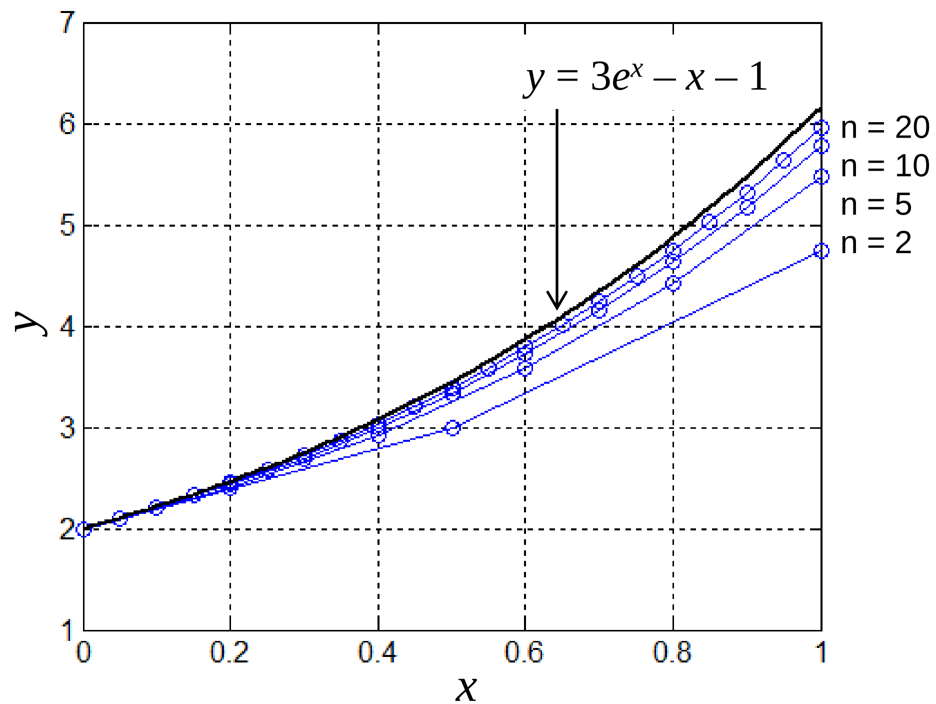
x	Appr. Sol.
0.1	2.2000
0.2	2.4300
0.3	2.6930
0.4	2.9923
…	
1.0	5.7812

n = 20

x	Appr. Sol.
0.05	2.1000
0.1	2.2075
0.15	2.3229
0.2	2.4465
…	
0.4	3.0324
…	
1.0	5.9599

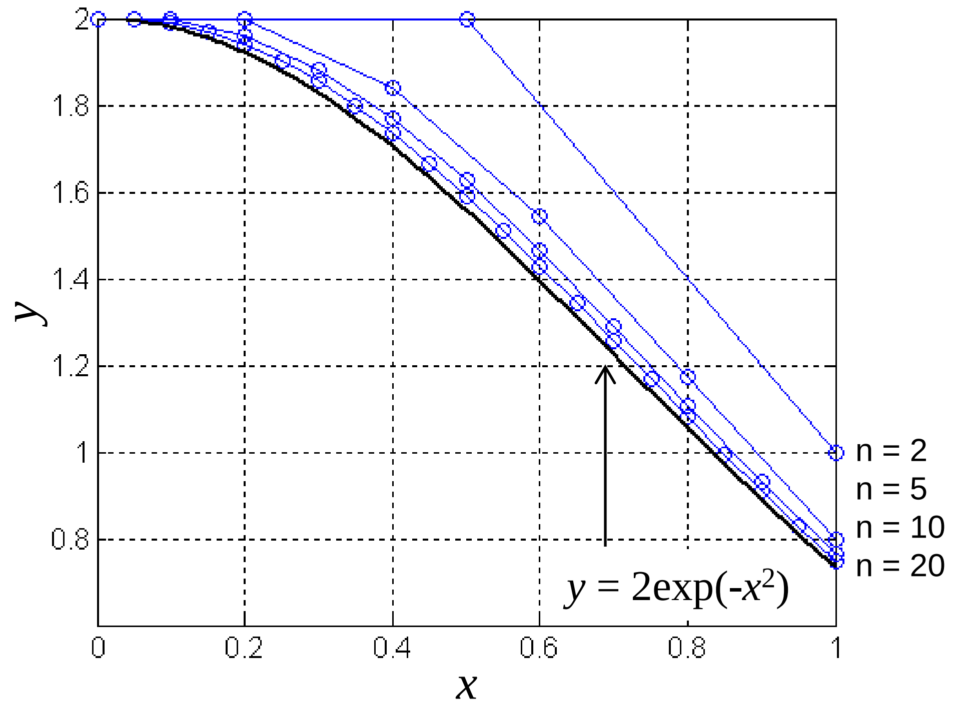
Exact Sol.

x	Exact Sol.
0.1	2.2155
0.2	2.4642
0.3	2.7496
0.4	3.0755
…	
1.0	6.1548



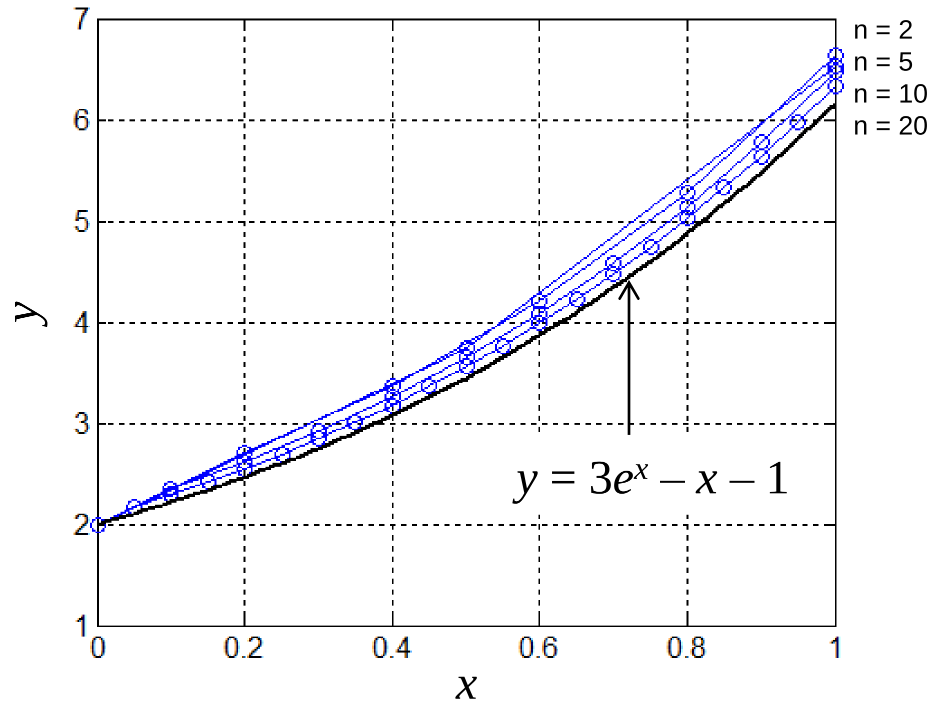
Applications of Euler (2)

IVP: $y' = -2xy,\ x_0 = 0,\ y_0 = 2;\ 0 \leq x \leq 1$



2nd Order Taylor Method

IVP: $y' = x + y,\ x_0 = 0,\ y_0 = 2;\ 0 \le x \le 1$



2nd Order Taylor Method – MATLAB Script

LIBRARY OF MATLAB PROCEDURES

Taylor_2

Solves differential equation $y' = f(x, y)$ with initial condition $y(a) = y_0$ on the interval $[a, b]$ by the 2nd order Taylor method

[redacted]

```
[x, y] = Taylor_2('fivp', 'fdir', int, ya, n)
```

```
function f_i = fivp(x, y)
f_i = x + y;
```

```
function f_d = fdir(x, y)
f_d = 1 + x + y;
```

Application of `Taylor_2`

IVP: $y' = -2xy,\ x_0 = 0,\ y_0 = 2;\ 0 \le x \le 1$

n = 5

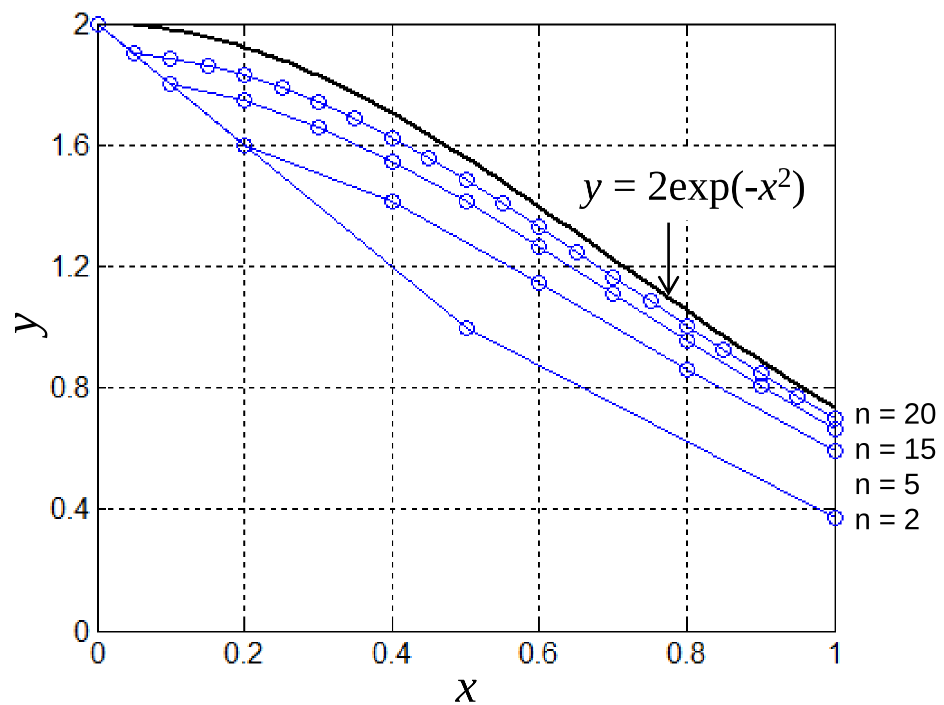
x	Appr. Sol.
0.2	1.6000
0.4	1.4080
0.6	1.1264
…	
1.0	0.5190

n = 10

x	Appr. Sol.
0.1	1.8000
0.2	1.7460
0.3	1.6587
…	
1.0	0.6211

n = 20

x	Appr. Sol.
0.05	1.9000
0.1	1.8857
0.15	1.8622
…	
1.0	0.6766



Errors of the Taylor Methods

Truncation Errors

- ✓ The 1st Order Taylor Method: $O(h^2)$
- ✓ The 2nd Order Taylor Method: $O(h^3)$

Two Types of Errors

At each step, y_{i+1} is computed from the first terms of the Taylor series, and ***once we have them truncated***, we get *the truncation error* – this is ***the Local T.E***.

Accumulation effects of all local truncation errors: the calculated value of y(x+h) *is used at the next step of approximation with the Taylor series as* ***known***, but (if it is not the first one) ***it is not exact – it is an approximated value*** – because of the previous truncation error; this is ***the Global T.E.***

- Therefore, *Taylor Methods of Higher Orders* (3, 4, 5, …) with the explicitly *known truncation errors* ***cannot guarantee much higher accuracy*** – because of the Global T.E.

1st Order Runge-Kutta Method – MATLAB Script

LIBRARY OF MATLAB PROCEDURES

RK:

Solves differential equation $y' = f(x, y)$ with initial condition $y(a) = y_0$ on the interval $[a, b]$ by the 1st order Runge-Kutta method

```
function [x, y] = RK (f, tspan, y0, n)
%
% The procedure solves y' = f(x,y) with initial condition y(a) = y0
% using n steps of the 1th order Runge-Kutta (or the Midpoint) method
%
a = tspan(1); b = tspan(2); h = (b-a)/n;
x = (a+h : h : b);
k1 = h*feval(f, a, y0);
k2 = h*feval(f, a + h/2, y0 + k1/2);
y(1) = y0 + k2;
%
for i = 1 : n-1
    k1 = h*feval(f, x(i), y(i));
    k2 = h*feval(f, x(i) + h/2, y(i) + k1/2);
    y(i+1) = y(i) + k2;
end
%
x = [a x];
y = [y0 y ];
```

4th Order Runge-Kutta Method – MATLAB Script

LIBRARY OF MATLAB PROCEDURES

RK4

Solves differential equation $y' = f(x, y)$ with initial condition $y(a) = y_0$ on the interval $[a, b]$ by the 4st order Runge-Kutta method

```
function [x, y] = RK4(f, tspan, y0, n)
%
% The procedure solves y' = f(x,y) with initial condition y(a) = y0
% using n steps of the classic 4th order Runge-Kutta method
%
a = tspan(1); b = tspan(2); h = (b-a)/n;
x = (a+h : h : b);
k1 = h*feval(f, a, y0);
k2 = h*feval(f, a + h/2, y0 + k1/2);
k3 = h*feval(f, a + h/2, y0 + k2/2);
k4 = h*feval(f, a + h, y0 + k3);
y(1) = y0 + k1/6 + k2/3 + k3/3 + k4/6;
%
for i = 1 : n-1
    k1 = h*feval(f, x(i), y(i));
    k2 = h*feval(f, x(i) + h/2, y(i) + k1/2);
    k3 = h*feval(f, x(i) + h/2, y(i) + k2/2);
    k4 = h*feval(f, x(i) + h, y(i) + k3);
    y(i+1) = y(i) + k1/6 + k2/3 + k3/3 + k4/6;
end
%
x = [a x];
y = [y0 y ];
```

Application of RK2 and RK4 (1)

IVP: $y' = x + y,\ x_0 = 0,\ y_0 = 2;\ 0 \leq x \leq 1$

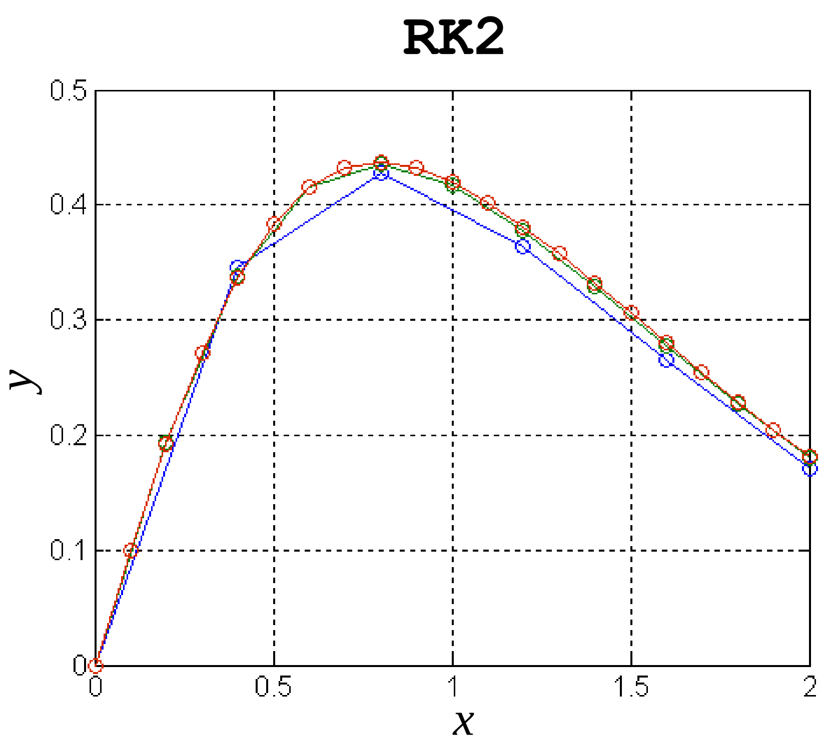
x	Exact solution	1^{st} Order R.-K.	4^{th} Order R.-K.
0.2	2.4642	2.4600 (0.17%)	2.4642 (0.0003%)
0.4	3.0755	3.0652 (0.33%)	3.0755 (0.0007%)
0.6	3.8664	3.8475 (0.49%)	3.8663 (0.0010%)
0.8	4.8766	4.8460 (0.63%)	4.8766 (0.0012%)
1.0	6.1548	6.1081 (0.76%)	6.1548 (0.0015%)

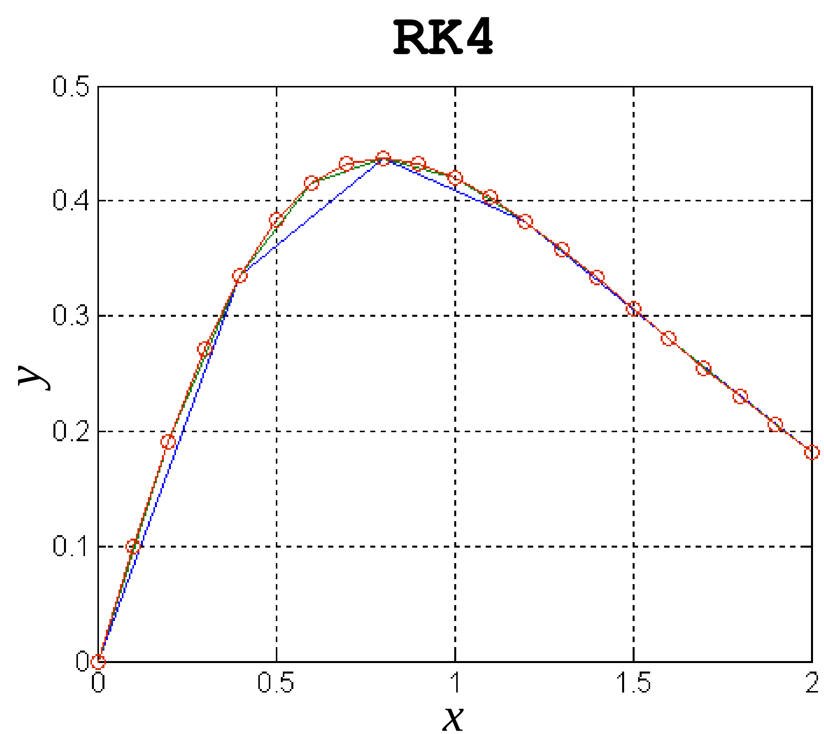
Application of RK2 and RK4 (2)

IVP: $(1+x^2)y' + 2xy = \cos x,\ 0 \le x \le 2,\ y(0) = 0$

RK4

x:	0.25	0.5	0.75	1.0	1.25	1.5	1.75	2.0	
y:		0.3833		0.4201		0.3065		0.1816	(n = 4)
	0.2328	0.3835	0.4362	0.4207	0.3703	0.3069	0.2422	0.1818	(n = 8)
	0.2329	0.3835	0.4362	0.4207	0.3703	0.3069	0.2422	0.1819	(n = 16)





Numerical Methods for IVPs – Some Observations

Runge-Kutta Methods

- Different versions of Runge-Kutta Methods are derived (conditioned by ***different circumstances or dictated by different attractive criteria***) differently and may work *particularly efficiently with particular IVPs*
- Many Runge-Kutta Methods are implemented in computer codes (can be found in many computer algebra systems – MATLAB, Mathematica, etc.)
  - The R.K. methods of the 5^{th} and 6^{th} order are called **Lawson's and Butcher's Methods**.
- Computationally, these methods are very fast – ***no big matrices, no multiple iterations,*** just a few algebraic formulas. (Very small steps and thousands of repetitions – not demanding for modern computer resources.)

Other Methods

Taylor and Runge-Kutta methods ***use only one previous approximate solution value***; in contract to that, ***the Multistep methods*** use ***more than one previous approximate solution*** taken from several previous points.

Stability of Numerical Solution (1)

Phenomenon of Numerical Instability

For some differential equations, ***any errors that occur in computation may be magnified*** – and this happens regardless "qualities" of the numerical method. Such problems are called ***ill-conditioned***.

- A numerical method is called ***stable*** *if errors uncured at one stage of the process do not tend to be magnified at later stages*.

Analysis of Instability

…involves the investigation of the error for a simple problem, such as

$y' = \lambda y$

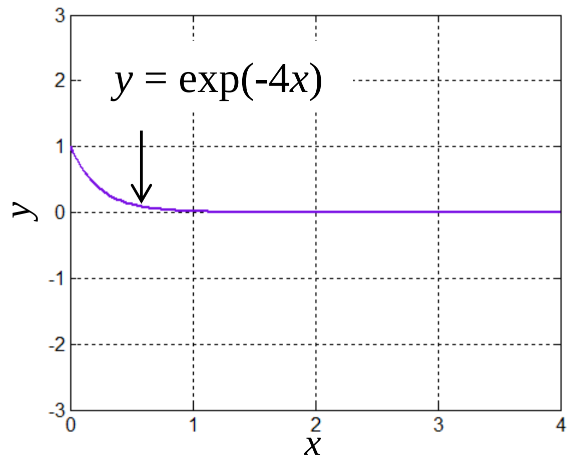
If the method is ***unstable*** for the model equation, it is likely to behave badly for other problems as well.

- If $\lambda > 0$, ***the true solution grows exponentially***, and it is not reasonable to expect the error *to remain small as x increases* – but one can hope that *the error remains small relative to the solution*.
- If $\lambda < 0$, ***the exact solution is a decaying exponential***, and one could expect the error to also go to 0 as $x \to \infty$.

Stability of Numerical Solution (2)

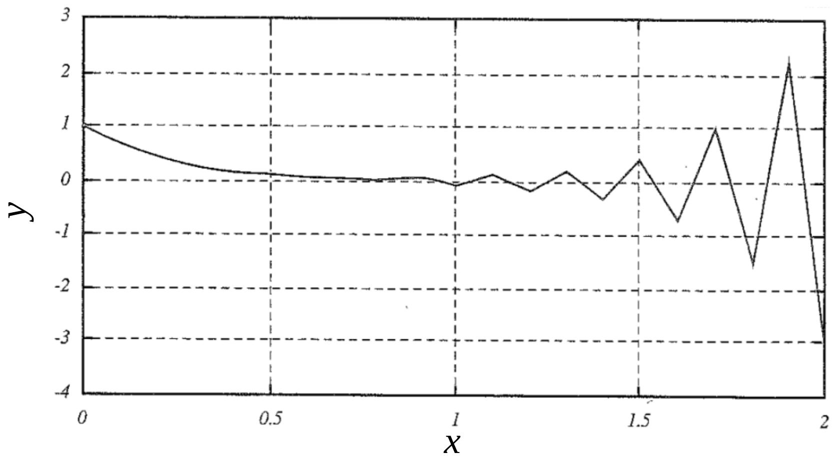
IVP: $y' = -4y,\ x_0 = 0,\ y_0 = 1;\ 0 \leq x \leq 4$

Exact (and Accurate Numerical) Solution



Weakly Stable Numerical Solutions

h = 0.1



h = 0.02