

# Fall 2003 Math 308/501–502

## Numerical Methods

### 1.4 Euler's Method

Fri, 24/Oct

©2003, Art Belmonte

#### Summary

#### Geometrical idea

Euler's method numerically approximates the solution of

$$y' = f(t, y), \quad y(a) = y_0$$

by following tangent lines at discrete points. Think of tangent line segments that form the background of a **dfield7** plot. Figures in Section 1.4 are illustrative in this regard.

#### The numerical algorithm

Let  $[a, b]$  be the interval over which an approximation to the solution is desired. (Thus  $t = a$  and  $t = b$  are the initial and final values of the independent variable, respectively.) Partition this interval into  $N$  subintervals each of length  $h = (b - a)/N$ , called the **step size**. Let  $t_0 = a$  and define

$$t_{k+1} = t_k + h, \text{ for } k = 0, 1, \dots, N - 1.$$

Notice that  $t_N = b$  and the other  $t_k$  so-defined are the interior endpoints of the subintervals. These collectively are the discrete values of the independent variable.

The initial value of the dependent variable is given by the initial condition,  $y(a) = y(t_0) = y_0$ . The other discrete dependent variable values are computed iteratively as follows.

$$\begin{aligned} \text{for } k &= 0 \text{ to } N - 1 \\ t_{k+1} &= t_k + h \\ y_{k+1} &= y_k + hf(t_k, y_k) \end{aligned}$$

Euler's method is a **single-step** numerical solver since it depends only on data obtained from the preceding step. It is a **fixed-step** solver since the lengths of the subintervals of  $[a, b]$  are all equal.

#### The error in the approximation

In general, as the step size decreases, so does the error in the approximation. This error consists of two parts, **round-off error** and **truncation error**. MATLAB does floating point computations in double precision, so round-off error is usually not an issue. The truncation error results from the fact that we are following the tangent line at a given point in the plane rather than a solution curve itself. Moreover, there is a truncation error at each step of the iteration as well as a propagated truncation error due to errors from previous steps.

The truncation error at each step is proportional to  $h^2$ . However, due to propagated truncation error (which can grow exponentially

over time), the maximum total error of the approximation at  $t = b$  satisfies the following error bound.

$$\text{maximum error} \leq \frac{M}{L} (e^{L(b-a)} - 1) h$$

Here  $L = \max_{(t,y) \in R} \left| \frac{\partial f}{\partial y} \right|$  and  $M = \frac{1}{2} \max_{(t,y) \in R} \left| \frac{\partial f}{\partial t} + f \frac{\partial f}{\partial y} \right|$  where  $R$  is a rectangle containing the solution curve. The fact that  $h$  occurs to the first power in the error bound is the reason that we say that Euler's method is a **first-order** method.

#### Systems

Euler's method carries over directly to systems of first-order equations. We simply write the algorithm in terms of vectors. Let

$$\mathbf{u}' = \mathbf{f}(t, \mathbf{u}), \quad \mathbf{u}(t_0) = \mathbf{u}_0$$

where  $\mathbf{u} = [u_1, u_2, \dots, u_n]^T$  is an  $n$ -dimensional *column* vector. With the  $t_k$  defined as above, we have

$$\begin{aligned} \text{for } k &= 0 \text{ to } N - 1 \\ t_{k+1} &= t_k + h \\ \mathbf{u}_{k+1} &= \mathbf{u}_k + h\mathbf{f}(t_k, \mathbf{u}_k) \end{aligned}$$

#### Hand Examples

We'll do one problem by hand. Numerical methods are best done with a computer, so make sure you study the MATLAB examples below carefully!

#### Example A

Calculate the first five iterations of Euler's method for the IVP

$$z' = 5 - z, \quad z(0) = 0$$

Use a step size of  $h = 0.1$ . Here  $f(t, z) = 5 - z$ .

#### Solution

We arrange our hand work in a table.

$k$	$t_k$	$z_k$	$f(t_k, z_k) = 5 - z_k$	$h$	$hf(t_k, z_k)$
0	0.0	0.0000	5.0000	0.1	0.5000
1	0.1	0.5000	4.5000	0.1	0.4500
2	0.2	0.9500	4.0500	0.1	0.4050
3	0.3	1.3550	3.6450	0.1	0.3645
4	0.4	1.7195	3.2805	0.1	0.3281
5	0.5	2.0476	2.9524	0.1	0.2952

#### MATLAB Examples

Polking's routine **eul** implements Euler's method in MATLAB. It works for both a single first-order equation as well as for a system of such equations. Except for trying a couple exercises by hand in your homework, you will use his routine exclusively.

## Example A [revisited]

Here we replicate the preceding hand work. First make a function M-file *f.m* which defines the derivative  $z' = f(t, z)$ , then write a script M-file *s14eA.m* to invoke **eul**. Notice how little we need to type: the **eul** routine rapidly does the work for us. Here are listings of the function M-file along with the diary file *s14eA.txt*.

### Solution

```
% Function M-file f.m
function zp = f(t,z)
zp = 5 - z;
%=====
% Diary file s14eA.txt
%
% NSS4-1.4/Example A
%
% Euler's method; check hand work
tspan = [0.0, 0.5]; z0 = 0; h = 0.1;
[teul,zeul] = eul(@f, tspan, z0, h);
%      ^^^ function handle!
% Table of t and z values
tk_zk = [teul, zeul]
tk_zk =
    0         0
    0.1000    0.5000
    0.2000    0.9500
    0.3000    1.3550
    0.4000    1.7195
    0.5000    2.0476
%
echo off; diary off
```

### M-62/3-i, iii

Find the exact solution of the initial value problem

$$z' = z^2 \cos 2t, \quad z(0) = 1.$$

Plot this solution over the interval  $[0, 6]$  along with the approximate solution provided by Euler's method with a step size of  $h = \frac{1}{8} = 0.125$ .

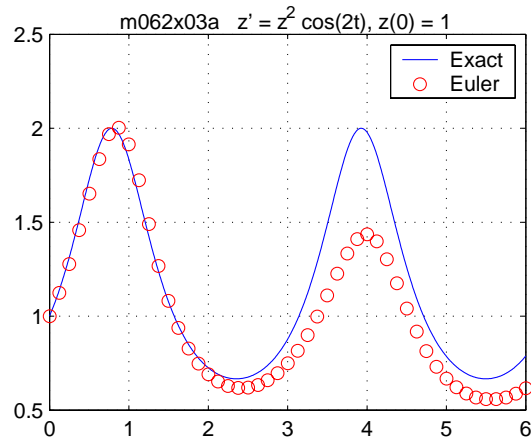
### Solution

Make a function M-file *f.m* which defines the derivative  $z' = f(t, z)$ , then write a script M-file *m062x03a.m* to invoke **eul**. In the latter, we'll use **dsolve** to obtain the exact solution. Here are listings of the function M-file and diary file *m062x03a.txt* as well as the plot.

```
% Function M-file f.m
function zp = f(t,z)
zp = z.^2 .* cos(2.*t); % array smart!
%=====
% Diary file m062x03a.txt
%
```

```
% M-62/3a: Euler's method only
%
% (i) Exact solution
sol = dsolve('Dz = z^2 * cos(2*t)', ...
    'z(0)=1', 't');
pretty(sol)

1
-----
cos(t) sin(t) - 1
%
t = linspace(0, 6, 601);
z = eval(vectorize(sol));
%
% (iii) Euler's method
tspan = [0,6]; z0 = 1; h = 0.125;
[teul,zeul] = eul(@f, tspan, z0, h);
%
% Plot of exact solution together
% with Euler approximate solution
plot(t,z, teul,zeul,'ro')
legend('Exact', 'Euler')
%
echo off; diary off
```



## Example B

Find the exact solution of the initial value problem

$$z' - 2z = xe^{2x}, \quad z(0) = 1.$$

Plot this solution on the interval  $[0, 1]$  along with the approximate solutions provided by Euler's method with step sizes  $h = 0.2, 0.1, 0.05$ .

### Solution

Remember to write the differential equation in normal form!

$$z' = f(x, z) = 2z + xe^{2x}.$$

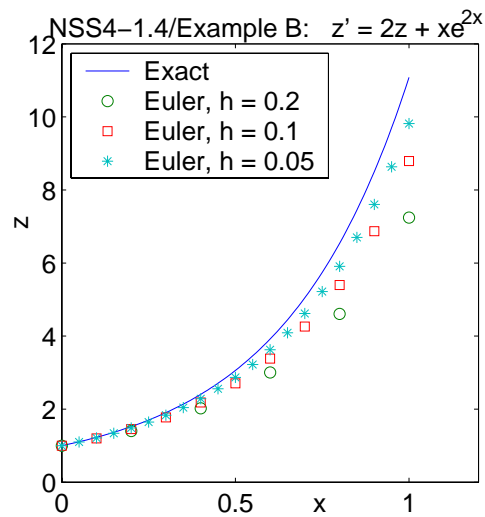
Make a function M-file to define the derivative  $z' = f(t, z)$ , then write a script M-file *s14eB.m* to invoke **eul**. In the latter, we'll use

**dsolve** to obtain the exact solution. Here are listings of the function M-file and diary file *s14eB.txt* as well as the plot.

As discussed in your lab manual (Exercise M-119/1), if you make your ODE function M-file *array smart*, then MATLAB's and Polking's solvers can accommodate multiple initial conditions! Accordingly, we will always make our function M-files *array smart*. In this regard, you will find the **vectorize** command helpful. Typically, use it in the command window, then copy its output into your function M-file.

```
% Function M-file f.m
function zp = f(x,z)
zp = 2.*z + x.*exp(2.*x); % array smart!
%=====
% Diary file s14eB.txt
%
% NSS4-1.4/Example B
%
% (i) Euler approximations
xspan = [0,1]; z0 = 1; h = 0.2;
[xeul1,zeul1] = eul(@f, xspan, z0, h);
h = 0.1;
[xeul2,zeul2] = eul(@f, xspan, z0, h);
h = 0.05;
[xeul3,zeul3] = eul(@f, xspan, z0, h);
%
% (ii) Exact solution
sol = dsolve('Dz - 2*z = x*exp(2*x)', ...
    'z(0)=1', 'x');
pretty(sol)

      2
(1/2 x  + 1) exp(2 x)
%
x = linspace(0, 1, 101);
z = eval(vectorize(sol));
%
%
% (iii) Plot of exact solution together
% with Euler approximate solutions
plot(x,z, xeul1,zeul1,'o', ...
    xeul2,zeul2,'s', xeul3,zeul3,'*')
legend('Exact', 'Euler, h = 0.2', ...
    'Euler, h = 0.1', ...
    'Euler, h = 0.05', 2)
axis([0, 1.2, 0, 12])
%
echo off; diary off
```



### Example C

- Use Euler's method with seven different steps sizes to approximate  $y(2)$ , the value of the solution of the initial value problem

$$y' = ty, \quad y(0) = 1$$

at  $t = 2$ . Use step sizes  $h = 2^{-m}$ ,  $m = 4, 5, \dots, 10$ .

- Find the exact solution, compute  $y(2)$  exactly, then construct a table similar to Table 3 on page 311 of your textbook.
- Make a plot of the error  $E_h$  versus step size  $h$ . Note the approximate linear relationship; i.e.,  $E_h \approx \lambda h$ .
- Estimate  $\lambda$ , then determine the step size required to ensure an error of at most 0.01 when using Euler's method.
- Use Euler's method with this step size and check the error at  $t = 2$ .

### Solution

Here are the function M-file, diary file, and plot.

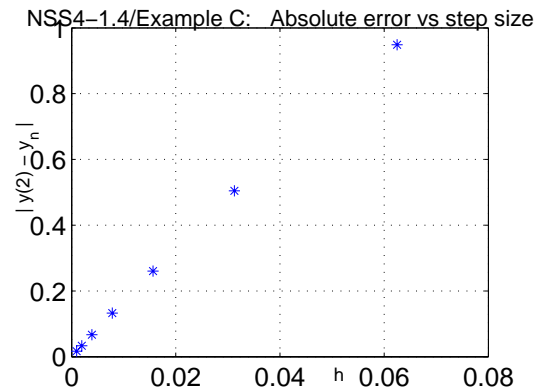
```
% Function M-file f.m
function yp = f(t,y)
yp = t .* y; % array smart!
%=====
% Diary file s14eC.txt
%
% NSS4-1.4/Example C
%
% (i) Euler approximations
tspan = [0,2]; y0 = 1; h = 0.125;
hcol = []; ecol = [];
for k = 1:7
    h = h/2; hcol = [hcol; h];
    [teul,yeul] = eul(@f, tspan, y0, h);
    dim = size(yeul); n = dim(1);
    ecol = [ecol; yeul(n)];
    echo off
end
%
%
%
% (ii) Exact solution and errors
```

```

sol = dsolve('Dy = t*y', 'y(0)=1', 't');
pretty(sol)

      2
    exp(1/2 t )
%
syms t
yxct = double(subs(sol, t, 2))
yxct =
    7.3891
%
% (iii) Plot of error verus step size; table too
aerr = abs(yxct - ecol);
plot(hcol, aerr, '*')
format long
T = [hcol, ecol, ...
    yxct*ones(size(hcol)), aerr]
%
T =
Columns 1 through 3
    0.062500000000000    6.44037380236993    7.38905609893065
    0.031250000000000    6.88449263634241    7.38905609893065
    0.015625000000000    7.12849236470764    7.38905609893065
    0.007812500000000    7.25660281093182    7.38905609893065
    0.003906250000000    7.32227327728956    7.38905609893065
    0.001953125000000    7.3552393155165    7.38905609893065
    0.000976562500000    7.37225460906787    7.38905609893065
Column 4
    0.94868229656072
    0.50456346258824
    0.26056373422301
    0.13245328799883
    0.06678282164109
    0.03353216737900
    0.01680148986278
format short
%
% Table
% (iv) Estimate of the proportionality constant
% and the step size needed to ensure that the
% absolute error is is less that 0.01
L = (aerr(7)-aerr(1)) / (hcol(7)-hcol(1))
L =
    15.1468
hest = 0.01 / L
hest =
    6.6021e-04
a = 0; b = 2; N = (b-a) / hest
N =
    3.0294e+03
% (v) Check whether prediction in (iv) pans out.
h = hest;
[teul,yeul] = eul(@f, tspan, y0, h);
dim = size(yeul); n = dim(1);
eul2 = yeul(n)
eul2 =
    7.3777
aerr = abs(yxct - eul2)
aerr =
    0.0114
%
echo off; diary off

```



### Example D

Calculate the first five iterations of Euler's method for the system

$$x' = y, \quad y' = -x \quad x(0) = 1, \quad y(0) = -1.$$

Use a step size of  $h = 0.1$ .

### Solution

Let  $u_1 = x$  and  $u_2 = y$ . Then

$$u_1' = x' = y = u_2 \quad \text{and} \quad u_2' = y' = -x = -u_1.$$

Hence

$$u_1' = u_2, \quad u_2' = -u_1 \quad u_1(0) = 1, \quad u_2(0) = -1$$

or  $\mathbf{u}' = \mathbf{f}(t, \mathbf{u}) = [u_2; -u_1]$ ,  $\mathbf{u}(0) = [1; -1]$ . Note the use of *column* vectors!

```

% Function M-file f.m
function up = f(t,u)
up = zeros(2,1); % column vector!
up(1) = u(2);
up(2) = -u(1);
%=====
% Diary file s14eD.txt
%
% NSS4-1.4/Example D
%
% Euler's method
tspan = [0.0, 0.5];
u0 = [1;-1]; h = 0.1;
[teul,ueul] = eul(@f, tspan, u0, h);
%
% Table of t, x, and y values
tk_uk = [teul, ueul]
tk_uk =
      0      1.0000     -1.0000
    0.1000    0.9000     -1.1000
    0.2000    0.7900     -1.1900
    0.3000    0.6710     -1.2690
    0.4000    0.5441     -1.3361
    0.5000    0.4105     -1.3905
%
echo off; diary off

```

## Example E

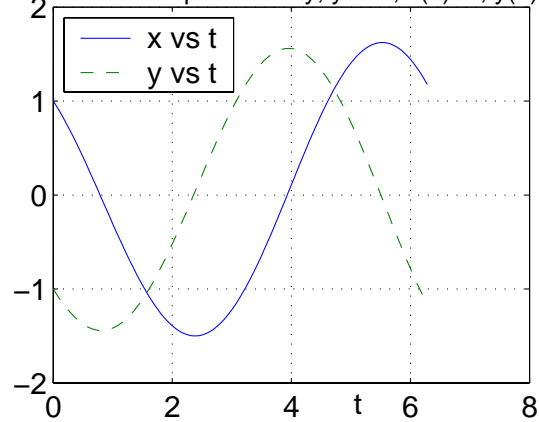
Following up on Example D, use Euler's method with a step size of  $h = 0.05$  to compute an approximate solution on the interval  $[0, 2\pi]$ . Provide plots of  $x$  versus  $t$ ,  $y$  versus  $t$ , and  $y$  versus  $x$ .

## Solution

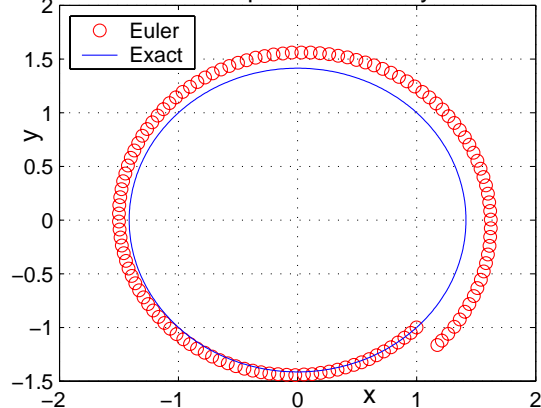
We also use **dsolve** to compute the exact solution, which turns out to have period  $2\pi$ . Accordingly, an  $xy$ -plot of the exact solution forms a closed loop, whereas the plot of the Euler solution does not (due to truncation error).

```
% Function M-file f.m
function up = f(t,u)
up = zeros(2,1); % column vector!
up(1) = u(2);
up(2) = -u(1);
%=====
% Diary file s14eE.txt
%
% NSS4-1.4/Example E
%
% Euler's method
tspan = [0, 2*pi];
u0 = [1;-1]; h = 0.05;
[teul,ueul] = eul(@f, tspan, u0, h);
%
% Plot of x and y versus t
t = teul; % t values
x = ueul(:,1); % x values in 1st col
y = ueul(:,2); % y values in 2nd col
plot(t,x, t,y,'--'); grid on
legend('x vs t', 'y vs t', 2)
%
% Analytical solution
[xx yy] = dsolve('Dx = y, Dy = -x', ...
    'x(0)=1, y(0)=-1', 't')
xx =
cos(t)-sin(t)
yy =
-sin(t)-cos(t)
t = linspace(0, 2*pi, 600);
xx = eval(vectorize(xx));
yy = eval(vectorize(yy));
% Plot of y versus x
figure; plot(x,y, xx,yy,'r--')
grid on; legend('Euler', 'Exact', 2)
%
echo off; diary off
```

NSS4-1.4/Example E:  $x'=y, y'=-x, x(0)=1, y(0)=-1$



NSS4-1.4/Example E Plots of y versus x



## Example F

Change  $y'' + y' + 25y = 0, y(0) = 4, y'(0) = 0$  into a first-order planar system. Use Euler's method with step sizes of  $h = 0.1, 0.01, 0.001$  to compute an approximate solutions on the interval  $[0, 2\pi]$ . Provide three plots of  $v = y'$  versus  $y$  (velocity vs position), one for each step size.

## Solution

Let  $x_1 = y$  and  $x_2 = y'$ . Then  $x_1' = y' = x_2$  and  $x_2' = y'' = -y' - 25y = -x_2 - 25x_1$ . Thus

$$x_1' = x_2, \quad x_2' = -x_2 - 25x_1, \quad x_1(0) = 4, \quad x_2(0) = 0.$$

Here is the function M-file and diary file.

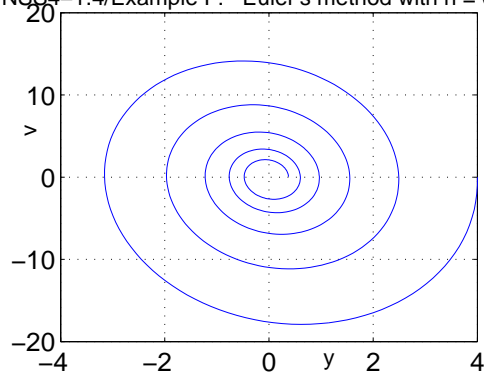
```
% Function M-file f.m
function xp = f(t,x)
xp = zeros(2,1); % column vector!
xp(1) = x(2);
xp(2) = -x(2) - 25*x(1);
%=====
% Diary file s14eF.txt
%
% NSS4-1.4/Example E
%
% First step size and plot of x2 vs x1;
```

```

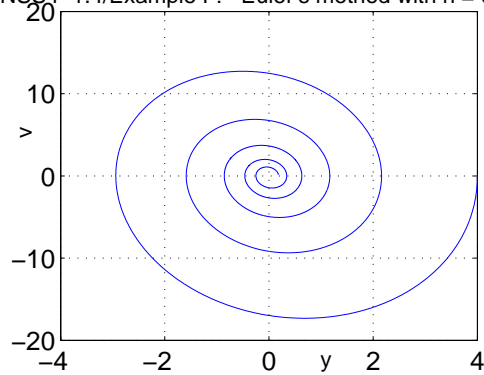
% i.e., of v=y' vs y
tspan = [0, 2*pi]; x0 = [4;0]; h = 0.1;
[teul,xeul] = eul(@f, tspan, x0, h);
y = xeul(:,1); % positions in col 1
v = xeul(:,2); % velocities in col 2
plot(y,v); grid on
%
% 2nd step size plus plot of x2 vs x1;
% i.e., of v=y' vs y
tspan = [0, 2*pi]; x0 = [4;0]; h = 0.01;
[teul,xeul] = eul(@f, tspan, x0, h);
y = xeul(:,1); % positions in col 1
v = xeul(:,2); % velocities in col 2
figure; plot(y,v); grid on
%
% Third step size and plot of x2 vs x1;
% i.e., of v=y' vs y
tspan = [0, 2*pi]; x0 = [4;0]; h = 0.001;
[teul,xeul] = eul(@f, tspan, x0, h);
y = xeul(:,1); % positions in col 1
v = xeul(:,2); % velocities in col 2
figure; plot(y,v); grid on
%
% Analytical solution
sol = dsolve('D2y + Dy + 25*y = 0', ...
'y(0)=4, Dy(0)=0', 't');
pretty(sol)
%
      1/2          1/2
4/33 11 exp(- 1/2 t) sin(3/2 11 t)
      1/2
+ 4 exp(- 1/2 t) cos(3/2 11 t)
%
echo off; diary off

```

NSS4-1.4/Example F: Euler's method with h = 0.01



NSS4-1.4/Example F: Euler's method with h = 0.001



Here are the plots. Notice that the largest step size produces enormous propagated truncation errors, whereas the smaller ones produce much smaller errors and smoother solution approximate solution curves.

NSS4-1.4/Example F: Euler's method with h = 0.1

