

# Mathematical software packages and MATLAB

We provide a review of mathematical software packages and a very brief introduction to the MATLAB environment. The students are encouraged to use the Matlab demos to gain some insights regarding Matlab.

## 1. Environments for numerical simulation.

Several environments/ software packages for mathematical software exist. Among them there are specialized packages such as [Ellpack](#) for solving elliptic boundary value problems.

General-purpose systems are: (i) Mathematica of Wolfram Research; (ii) Matlab of Mathworks); (iii) Maple of Maplesoft; and (iv) IDL. A [comparison](#) of the syntax of basic/useful tasks for the four systems is available.

[Mathematica](#) is the premier all-purpose mathematical software package. It integrates swift and accurate symbolic and numerical calculation, all-purpose graphics, and a powerful programming language. It has a sophisticated ``notebook interface" which is great for documenting and displaying work. It can save individual graphics in any graphics format. Its functional programming language (as opposed to procedural) makes it possible to do complex programming using very short concise commands; it does, however, allow the use of basic procedural programming constructs like Do and For. Drawbacks: steeper learning curve for beginners used to procedural languages; more expensive.

[Matlab](#) combines efficient computation, visualization and programming for linear-algebraic technical work and other mathematical areas. It is widely used in the Engineering schools. Drawbacks: Does not support analytical/symbolic math.

[Maple](#) is a powerful analytical and mathematical software which does the same sorts of things that Mathematica does, with similar high quality. Maple's programming language procedural -- like C or Fortran or Basic -- although it has a few functional programming constructs. Drawbacks: Worksheet interface/typesetting not as developed as Mathematica's, but it is less expensive.

[Interactive Data Language](#) - IDL excels at processing real-world data, especially graphics, and has a reasonably simple syntax, especially for those familiar with Fortran or C. IDL makes it as easy as possible to read in data from files of numerous scientific data formats. IDL is very popular at NASA, universities and research facilities, and especially at C.U. where it was originally developed and is free.

2. Using MATLAB (color/font code: red → new concepts, blue → actual output from the command window, bold → important fact, italics → names of variables or functions).

The workspace → Is the environment (address space) where all variables reside.

After carrying out a calculation, MATLAB assigns the result to the built-in variable called *ans*; a % character marks the beginning of a comment line.

Three primary windows:

1. Command window - used to enter commands and data. The prompt is ">>";

The command window allows the use of Matlab as a calculator when commands are typed in line by line, e.g.,

```
>> a = 77 - 16
ans = 61
>> b = a * 10
ans = 610
```

Examples of **system commands**:

<i>who/whos</i>	→ list all variables in the workspace
<i>clear</i>	→ removes all variables from the workspace
<i>computer</i>	→ lists the system MATLAB is running on
<i>version</i>	→ lists the toolboxes (utilities) available

**A script file** is a set of MATLAB commands. For example the script *factor.m*:

```
function fact = factor(n)
x=1;
for i=1:n
    x=x*i;
end
fact=x;
fprintf('Factor %6.3f %6.3f \n' n, fact);
end
```

Scripts can be executed by: (i) typing their name (without the .m) in the command window; (ii) selecting the Debug, Run (or Save and Run) command in the editing window; or (iii) hitting the F5 key while in the editing window. Option (i) will run the file as it exists on the drive, options (ii) and (iii) save any edits to the file.

For example:

```
>> factor(12)
```

```
ans =
```

479001600

2. Edit window - used to create and edit M-files (programs) such as the factor. For example, we can use the editor to create *factor.m*
3. Graphics window(s) - used to display plots and graphics

**Variable names** → up to 31 alphanumeric characters (letters, numbers) and the underscore ( `_` ) symbol; must start with a letter.

Special variables and constants.

<code>ans</code>	- Most recent answer.
<code>eps</code>	- Floating point relative accuracy.
<code>realmax</code>	- Largest positive floating point number.
<code>realmin</code>	- Smallest positive floating point number.
<code>pi</code>	- 3.1415926535897....
<code>i</code>	- Imaginary unit.
<code>j</code>	- Imaginary unit.
<code>inf</code>	- Infinity.
<code>nan</code>	- Not-a-Number.
<code>isnan</code>	- True for Not-a-Number.
<code>isinf</code>	- True for infinite elements.
<code>isfinite</code>	- True for finite elements.
<code>why</code>	- Succinct answer.

To report the value of variable *kiki* type its name:

```
>> kiki
kiki =
13
```

To prevent the system from reporting the value of variable *kiki* append the semi-solon (;) at the end of a line:

```
>> kiki = 13;
```

As noted earlier, the variable name *pi* is reserved. There are several formats to report the value of a real number, *short* (decimal point and four significant digits), *long* (decimal point and 14 digits for double precision, and 6 for single precision), *eng*. For example:

```
>> format short; pi
ans =
    3.1416
>> format long; pi
ans =
    3.14159265358979
>> format short eng; pi
ans =
    3.1416e+000
>> pi*10000
ans =
    31.4159e+003
```

## Arrays

In addition to scalars, we can use **vectors** (one-dimensional arrays), or **matrices** (two-dimensional arrays). Examples:

```
a = [1 2 3 4 5] % specify the elements of the array individually separated by
                % spaces, or commas
```

```
a =
    1    2    3    4    5
```

```
>> a = 1:2:10 % specify, the first element, the increment, and the # of elements
```

```
a =
    1    3    5    7    9
```

```
>> a = [1 3 5 7 9] % specify the elements separated by commas or spaces
```

```
a =
    1    3    5    7    9
```

**Arrays are stored column wise** (column 1, followed by column 2, etc.). A semicolon marks the end of a row.

```
>> a=[1;2;3]
a =
     1
     2
     5
```

The *transpose* (') operator transform rows in columns and columns in rows;

```
>> b = [2 9 3 ;4 16 5 ;6 13 7 ;8 19 9;10 11 11]
b =
     2     9     3
     4    16     5
     6    13     7
     8    19     9
    10    11    11
```

```
>>b' % The transpose of matrix b
ans =
     2     4     6     8    10
     9    16    13    19    11
     3     5     7     9    11
```

```
>> c = [2 9 3 4 16; 5 6 13 7 8; 19 9 10 11 11]
```

```
c =
     2     9     3     4    16
     5     6    13     7     8
    19     9    10    11     1
```

```
>> b*c % Matrix multiplication
```

```
ans =
    106    99    153    104    137
    183   177    270    183    247
    210   195    257    192    277
    282   267    361    264    379
    284   255    283    238    369
```

**Element by element array operations** (both arrays must have the same number of elements!) (1) multiplication (“.\*”) (2) division (“./”) and (3) exponentiation (notice the “.” before “\*”, “/”, or “^”).

```
>> b.* b
```

```
ans =  
    4    81    9  
   16   256   25  
   36   169   49  
   64   361   81  
  100   121  121
```

```
>> b ./ b
```

```
ans =  
  
    1    1    1  
    1    1    1  
    1    1    1  
    1    1    1  
    1    1    1
```

```
>> b.^b
```

```
ans =  
1.0e+024 *  
  
    0.0000    0.0000    0.0000  
    0.0000    0.0000    0.0000  
    0.0000    0.0000    0.0000  
    0.0000    1.9784    0.0000  
    0.0000    0.0000    0.0000
```

How to identify an element of an array:

```
>> b(4,2) % element in row 4 column 2 of matrix b
```

```
ans =  
    19
```

```
>> b(9) % the 9-th element of b (recall that elements are stored column wise)
```

```
ans =  
    19
```

```
>> b(5) % the 5-th element of b (recall that elements are stored column wise)
```

```
ans =  
    10
```

```
>> b(17)
```

```
??? Attempted to access b(17); index out of bounds because numel(b)=15.
```

```
>> whos
```

Name	Size	Bytes	Class	Attributes
b	5x3	120	double	

### Built-in functions for matrix manipulation

There are several functions for matrix manipulation. To list them type “help elmat”:

```
>> help elmat % produce a list of elementary matrices and functions for matrix  
Elementary matrices and matrix manipulation.
```

Elementary matrices.

- zeros - Zeros array.
- ones - Ones array.
- eye - Identity matrix.
- repmat - Replicate and tile array.
- linspace - Linearly spaced vector.
- logspace - Logarithmically spaced vector.
- freqspace - Frequency spacing for frequency response.
- meshgrid - X and Y arrays for 3-D plots.
- accumarray - Construct an array with accumulation.
- :
- Regularly spaced vector and index into matrix

### Basic array information.

- size - Size of array.
- length - Length of vector.
- ndims - Number of dimensions.
- numel - Number of elements.
- disp - Display matrix or text.
- isempty - True for empty array.
- isequal - True if arrays are numerically equal.
- isequalwithequalnans - True if arrays are numerically equal.

### Matrix manipulation.

- cat - Concatenate arrays.
- reshape - Change size.
- diag - Diagonal matrices and diagonals of matrix.
- blkdiag - Block diagonal concatenation.
- tril - Extract lower triangular part.
- triu - Extract upper triangular part.
- fliplr - Flip matrix in left/right direction.
- flipud - Flip matrix in up/down direction.
- flipdim - Flip matrix along specified dimension.
- rot90 - Rotate matrix 90 degrees.
- : - Regularly spaced vector and index into matrix.
- find - Find indices of nonzero elements.
- end - Last index.
- sub2ind - Linear index from multiple subscripts.
- ind2sub - Multiple subscripts from linear index.
- bsxfun - Binary singleton expansion function.

### Multi-dimensional array functions.

- ndgrid - Generate arrays for N-D functions and interpolation.
- permute - Permute array dimensions.
- ipermute - Inverse permute array dimensions.
- shiftdim - Shift dimensions.
- circshift - Shift array circularly.
- squeeze - Remove singleton dimensions.

### Array utility functions.

- isscalar - True for scalar.
- isvector - True for vector.

### Special variables and constants.

ans - Most recent answer.  
eps - Floating point relative accuracy.  
realmax - Largest positive floating point number.  
realmin - Smallest positive floating point number.  
pi - 3.1415926535897....  
i - Imaginary unit.  
inf - Infinity.  
nan - Not-a-Number.  
isnan - True for Not-a-Number.  
isinf - True for infinite elements.  
isfinite - True for finite elements.  
j - Imaginary unit.  
why - Succinct answer.

### Specialized matrices.

compan - Companion matrix.  
gallery - Higham test matrices.  
hadamard - Hadamard matrix.  
hankel - Hankel matrix.  
hilb - Hilbert matrix.  
invhilb - Inverse Hilbert matrix.  
magic - Magic square.  
pascal - Pascal matrix.  
rosser - Classic symmetric eigenvalue test problem.  
toeplitz - Toeplitz matrix.  
vander - Vandermonde matrix.  
wilkinson - Wilkinson's eigenvalue test matrix.

### Examples:

```
>> eye(7)
```

```
ans =  
 1  0  0  0  0  0  0  
 0  1  0  0  0  0  0  
 0  0  1  0  0  0  0  
 0  0  0  1  0  0  0  
 0  0  0  0  1  0  0  
 0  0  0  0  0  1  0  
 0  0  0  0  0  0  1
```

```
>> size(b)
ans =
    5    3
```

```
>> hadamard(8) %Hadamard 8x8 matrix
ans =
```

```
    1    1    1    1    1    1    1    1
    1   -1    1   -1    1   -1    1   -1
    1    1   -1   -1    1    1   -1   -1
    1   -1   -1    1    1   -1   -1    1
    1    1    1    1   -1   -1   -1   -1
    1   -1    1   -1   -1    1   -1    1
    1    1   -1   -1   -1   -1    1    1
    1   -1   -1    1   -1    1    1   -1
```

```
>> hilb(8) % Hilbert 8x8 matrix
ans =
```

```
    1.0000    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250
    0.5000    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250    0.1111
    0.3333    0.2500    0.2000    0.1667    0.1429    0.1250    0.1111    0.1000
    0.2500    0.2000    0.1667    0.1429    0.1250    0.1111    0.1000    0.0909
    0.2000    0.1667    0.1429    0.1250    0.1111    0.1000    0.0909    0.0833
    0.1667    0.1429    0.1250    0.1111    0.1000    0.0909    0.0833    0.0769
    0.1429    0.1250    0.1111    0.1000    0.0909    0.0833    0.0769    0.0714
    0.1250    0.1111    0.1000    0.0909    0.0833    0.0769    0.0714    0.0667
```

```
>> magic(8) %magic 8x8 matrix.
```

```
ans =
```

```
    64    2    3    61    60    6    7    57
     9    55    54    12    13    51    50    16
    17    47    46    20    21    43    42    24
    40    26    27    37    36    30    31    33
    32    34    35    29    28    38    39    25
    41    23    22    44    45    19    18    48
    49    15    14    52    53    11    10    56
     8    58    59    5    4    62    63    1
```

```
>> logspace(-1,2,10) % to create n numbers logarithmically space between x and y
      % use logspace(x,y,n)
```

```
ans =
    0.1000    0.2154    0.4642    1.0000    2.1544    4.6416   10.0000   21.5443   46.4159
 100.0000
```

## Operators

### 1. Arithmetic operators (priority decreases from top to bottom)

Arithmetic operation	Example
Exponentiation (^)	$5^2 = 25$
Negation (-) (unary operation)	$-7 = -7$
Multiplication (*) and Division (/)	$2 * \pi = 6.2832$  $\pi / 4 = 0.7854$
Left Division (\)	$6 \setminus 2 = 0.3333$
Addition (+) and Subtraction (-)	$3 + 5 = 8$  $3 - 5 = -2$

### 2. Relational operators

Relational operation	Example
Equal (==)	$x == 0$
Not equal (~=)	$v \sim = 'm'$
Less than (<)	$x < 0$
Greater than (>)	$u > v$
Less than or equal to (<=)	$4.7 <= a/9$
Greater than or equal to (>=)	$r >= 0$

### 3. Logical operators

Logical operation	Example
NOT (~)	~a
AND (&)	a & b
OR ( )	a b

### Complex Numbers

```
>> x = 4 + 6i; y = 3 - i;
```

```
>> x+y
```

```
ans =
```

```
7.0000 + 5.0000i
```

```
>> x-y
```

```
ans =
```

```
1.0000 + 7.0000i
```

```
>> x*y
```

```
ans =
```

```
18.0000 +14.0000i
```

```
>> x' % complex conjugate
```

```
ans =
```

```
4.0000 - 6.0000i
```

```
>> x*x'
```

```
ans =
```

```
52
```

To convert a complex number  $x=a+ib$  to polar form:  $q=\text{modulus} \cdot e^{(i \cdot \text{theta})}$  with  $\text{modulus} = \sqrt{a^2 + b^2}$  and  $\text{theta} = \tan^{-1}(b/a)$ .

```
>> x = 4 + 6i;
```

```
>> modulus=abs(x)
```

```
modulus =
```

```
7.2111
```

```
>> theta=angle(x)
```

```
theta =
```

```
0.9828
```

## Functions

**Function files** (do not confuse them with script files) can accept input arguments from and return outputs to the command window; variables created and manipulated within the function do not impact the command window.

The general syntax for a function is:

```
function outvar = funcname(arglist)  
statements  
outvar = value;
```

where

- *outvar*: output variable name
- *funcname*: function's name
- *arglist*: input argument list - comma-delimited list of what the function calls values passed to it
- *statements*: MATLAB commands for the function

Example:

```
function cosapprox(x,niter);  
    ser(1)=1 - (x^2)/2;  
    %fprintf (' i ser(i) error(i) \n' );  
    for i=2:niter  
        ser(i)= ser(i-1) + (-1)^i*((x^(2*i))/(factor(2*i)))  
        error(i)= ((cos(x)-ser(i))/cos(x))*100;  
    end  
    %fprintf (' %12.10f \n', ser);  
    fprintf (' %12.10f \n', error);  
    plot(error(1:niter));  
    xlabel('Number of terms n');  
    ylabel('Error cos(x) approximated by Taylor series with n terms');  
    grid
```

```
>> cosapprox(pi/4,10)
0.0000000000
-0.0455978548
0.0005043600
-0.0000034644
0.0000000162
-0.0000000001
0.0000000000
0.0000000000
0.0000000000
0.0000000000
0.0000000000
```

**Subfunctions:** a function file can contain a *primary function* and one or more *subfunctions*

- The primary function is whatever function is listed first in the M-file - its function name should be the same as the file name.
- Subfunctions are listed below the primary function. They are *only* accessible by the main function and subfunctions within the same M-file and *not* by the command window or any other functions or scripts.

A number of **built-in elementary math functions** are available. To list them type “help elfun”:

```
>> help elfun
Elementary math functions.

Trigonometric.
sin      - Sine.
sind     - Sine of argument in degrees.
sinh     - Hyperbolic sine.
asin     - Inverse sine.
asind    - Inverse sine, result in degrees.
asinh    - Inverse hyperbolic sine.
cos      - Cosine.
```

cosd - Cosine of argument in degrees.  
cosh - Hyperbolic cosine.  
acos - Inverse cosine.  
acosc - Inverse cosine, result in degrees.  
acosh - Inverse hyperbolic cosine.  
tan - Tangent.  
tand - Tangent of argument in degrees.  
tanh - Hyperbolic tangent.  
atan - Inverse tangent.  
atand - Inverse tangent, result in degrees.  
atan2 - Four quadrant inverse tangent.  
atanh - Inverse hyperbolic tangent.  
sec - Secant.  
secd - Secant of argument in degrees.  
sech - Hyperbolic secant.  
asec - Inverse secant.  
asecd - Inverse secant, result in degrees.  
asech - Inverse hyperbolic secant.  
csc - Cosecant.  
cscd - Cosecant of argument in degrees.  
csch - Hyperbolic cosecant.  
acsc - Inverse cosecant.  
acscd - Inverse cosecant, result in degrees.  
acsch - Inverse hyperbolic cosecant.  
cot - Cotangent.  
cotd - Cotangent of argument in degrees.  
coth - Hyperbolic cotangent.  
acot - Inverse cotangent.  
acotd - Inverse cotangent, result in degrees.  
acoth - Inverse hyperbolic cotangent.  
hypot - Square root of sum of squares.

#### Exponential and logarithms.

exp - Exponential.  
expm1 - Compute  $\exp(x)-1$  accurately.  
log - Natural logarithm.  
log1p - Compute  $\log(1+x)$  accurately.  
log10 - Common (base 10) logarithm.

log2 - Base 2 logarithm and dissect floating point number.  
pow2 - Base 2 power and scale floating point number.  
realpow - Power that will error out on complex result.  
reallog - Natural logarithm of real number.  
realsqrt - Square root of number greater than or equal to zero.  
sqrt - Square root.  
nthroot - Real n-th root of real numbers.  
nextpow2 - Next higher power of 2.

#### Complex.

abs - Absolute value.  
angle - Phase angle.  
complex - Construct complex data from real and imaginary parts.  
conj - Complex conjugate.  
imag - Complex imaginary part.  
real - Complex real part.  
unwrap - Unwrap phase angle.  
isreal - True for real array.  
cplxpair - Sort numbers into complex conjugate pairs.

#### Rounding and remainder.

fix - Round towards zero.  
floor - Round towards minus infinity.  
ceil - Round towards plus infinity.  
round - Round towards nearest integer.  
mod - Modulus (signed remainder after division).  
rem - Remainder after division.  
sign - Signum.

## Plotting

Two function to plot:

- *plot* (for 2-D data) and
- *plot3* (for 3-D data).

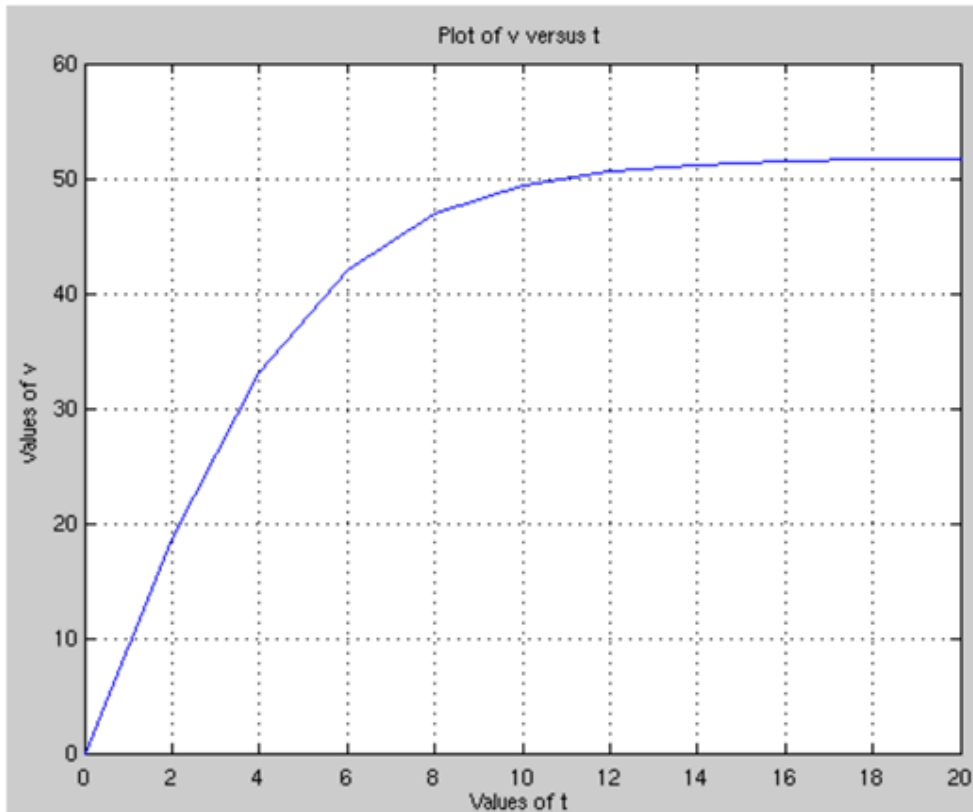
Other plotting commands:

- *hold on* → keep the current data plotted and add the results of any further plot commands to the graph. Should be used *after* the first plot in a series is made. The hold on continues until the *hold off* → clear the graph and start over if another plotting command is given.
- *subplot* → splits the figure window into an  $m \times n$  array of small axes and makes the  $p$ -th one active. The first subplot is at the top left, then the numbering continues across the row. This is different from how elements are numbered within a matrix!

Example

```
>> t = [0:2:20]';  
g = 9.81; m = 68.1; cd = 0.25;  
v = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t);  
plot(t, v)  
title('Plot of v versus t')  
xlabel('Values of t')  
ylabel('Values of v')  
grid
```

The result appears in a graphics window:



## Programming constructs

**Conditional statements** → General format

```

If condition
  then
    statements
  else
    statements

```

Deciding which branch runs is based on the result of *conditions* which are either true or false. If the *condition* is a matrix, it is considered true if and only if all entries are true (or non-zero).

- If an *if* tree hits a *true* condition, that branch (and that branch only) runs, then the tree terminates.
- If an *if* tree gets to an else statement without running any prior branch, that branch will run.

A **for loop** → The general syntax:

```
for index = start:step:finish
  statements
end
```

where the *index* variable takes on successive values in the vector created using the : operator

The *for loop* ends after a specified number of repetitions established by the number of columns given to an index variable.

A **while loop** → The general syntax:

```
while condition
  statements
end
```

where the *condition* is a logical expression. If the *condition* is true, the *statements* will run and when that is finished, the loop will again check on the *condition*. Though the *condition* may become false as the *statements* are running, the only time it matters is after all the statements have run.

A *while loop* is different from a *for loop* since while loops can run an indeterminate number of times.

Sometimes it is useful to break out of a for or while loop early using a break statement, generally in conjunction with an if structure.

Example:

```
>> while (1)
    x = x - 5
    if x < 0, break, end
end

x =
    24

x =
    19

x =
    14

x =
     9

x =
     4

x =
    -1
```

**Anonymous functions** → are simple one-line functions created without the need for an M-file. Syntax:

$$fhandle = @(arg1, arg2, \dots) \textit{expression}$$

**Inline functions** are essentially the same as anonymous functions, but with a different syntax:

$$fhandle = inline(\textit{'expression'}, \textit{'arg1'}, \textit{'arg2'}, \dots)$$

Anonymous functions can use workspace the values of variables upon creation, while inline functions cannot.

Example:

We have the following function file called pde.m

```
function pdeq(deriv,dt,t_init,t_final,y_init);
% solve iteratively  y_{i+1}=y_{t}}+ dy_{i}/dt * Delta_t
% deriv=@(y) 9.81-(0.25/68.1)*y^2; pde(deriv,0.5,0,12,0);
t = t_init;
y = y_init;
h=dt;
while (1)
    if t+dt > t_final
        h=t_final - t;
    end
    y = y + deriv(y)*h;
    %fprintf(' %8.5f  \n', t);
    %fprintf(' %8.5f  \n', y);
    t = t + h;
    if t >= t_final
        break
    end
end
y_final=y;
fprintf(' %8.5f  \n', y_final);
```

```
>>deriv=@(y) 9.81-(0.25/68.1)*y^2; pde(deriv,0.5,0,12,0);
50.92591
```