北京航空航天大學
BEIHANG UNIVERSITY

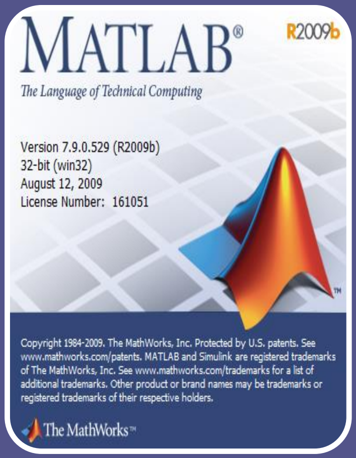
MATLAB Programming
(Lecture 3)

Dr. Sun Bing

School of EIE

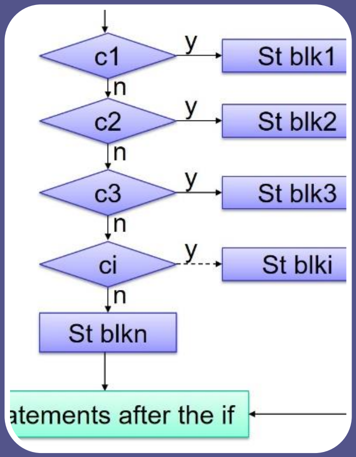
Beihang University

Contents



Matrix and Array

- Matrix and vector
- Array and matrix operations
- Solving linear systems of eq.
- Polynomial and it's operation
- Relational & logical operators
- Some conversion functions
- The precedence of operators



Program design and control flow

- Introduction to M-file
- Top-Down Design Tech.
- Flow Control---branches
- The loop statements
- Programming Examples

3.1 Matrix and Vector

- MATLAB is a matrix-based computing environment. All of the data that you enter into MATLAB is stored in the form of a matrix or a two-dimensional array.

- Row Vector is a matrix of 1-by-n .

- Column Vector is a matrix of n-by-1.

3.1.1 Creating Matrix

1. The simplest way to create a matrix in MATLAB is to use the matrix constructor operator, **[ ]**.

Examples:

```
M = [row1;row2;row3;…;rown];
RV = [e1,e2,…,en];   % Row vector
CV = [e1;e2;…;en];   % Column vector
EM= [];    % Empty matrix
```

3.1.1 Creating Matrix

2. Using Colon operator(**:**) and shortcut expression:

first : incr : last

Examples:

```
1.  V = 1:10;

2.  V2 = 1:0.5:10;

3.   m = [1:4;2:5;3:6;4:7];
```

3.1.1 Creating Matrix

3. Using Specialized Matrix Functions . MATLAB has a number of functions that create different kinds of matrices. For example,

```
>> rand('seed',100),A=rand(3)

A =

    0.2909    0.5046    0.5968

    0.0484    0.3671    0.8085

    0.0395    0.9235    0.9253
```

Table3.1 Matrix Functions

Function	***Description***
`ones`	Create a matrix or array of all ones.
`zeros`	Create a matrix or array of all zeros.
`eye`	Create a matrix with ones on the diagonal and zeros elsewhere.
`diag`	Create a diagonal matrix from a vector.
`rand`	Create a matrix or array of uniformly distributed random numbers.
`randn`	Create a matrix or array of normally distributed random numbers and arrays.

Table3.1 Matrix Functions (continued)

Function	***Description***
`magic`	Create a square matrix with rows, columns, and diagonals that add up to the same number.
`randperm`	Create a vector (1 by n matrix) containing a random permutation of the specified integers.

Note: you can use the *`help elmat`* to list all the functions.

See : example chpt3_1.m

diag()

- `X = diag(v,k)` when `v` is a vector of `n` components, returns a square matrix `X` of order `n+ abs(k)`, with the elements of `v` on the kth diagonal. `k = 0` represents the main diagonal, `k > 0` above the main diagonal, and `k < 0` below the main diagonal.
- `X = diag(v)` puts v on the main diagonal, same as above with `k = 0`.
- `v = diag(X,k)` for matrix `X`, returns a column vector `v` formed from the elements of the kth diagonal of `X`.
- `v = diag(X)` returns the main diagonal of `X`, same as above with `k = 0`. See Example chpt3-1a.m

3.1.2 Matrix concatenate

1. The brackets [] operator discussed earlier serves not only as a matrix constructor, but also as the MATLAB concatenation operator. The expression C = [A B] horizontally concatenates matrices A and B.

```
>> a = [1:3;2:4;3:5];
>> b = ones(3,2);
>> c = [a,b]
c =

     1     2     3     1     1
     2     3     4     1     1
     3     4     5     1     1
```

3.1.2 Matrix concatenate

- **Using Matrix Concatenation Functions**

Function	***Description***
`cat`	Concatenate matrices along the specified dimension
`vertcat`	Vertically concatenate matrices
`horzcat`	Horizontally concatenate matrices

Note: the number of rows or columns of two matrics must be the same.

3.1.2 Matrix concatenate

Concatenate arrays along specified dimension

`C = cat(dim, A, B)`

concatenates the arrays `A` and `B` along dim.

`C = cat(dim, A1, A2, A3, A4, ...)`

concatenates all the input arrays (`A1, A2, A3, A4`, and so on) along dim.

For example:

`cat(2, A, B)` is the same as `[A, B]`, and `cat(1, A, B)` is the same as `[A; B]`.

See Example chpt3_1b.m

Example:

```
>> a = [1 2 3; 2 3 4;3 4 5];
>> b = [ 1 1;1 1; 1 1];
>> d = cat(2,a,b)

d =

     1     2     3     1     1
     2     3     4     1     1
     3     4     5     1     1

>> d2=horzcat(a,b)

d2 =

     1     2     3     1     1
     2     3     4     1     1
     3     4     5     1     1
```

3.1.3 Matrix Transpose

For real matrices, the transpose operation interchanges a_{ij} and a_{ji}. MATLAB uses the apostrophe (or single quote) to denote transpose.

For example,

```
>> A = magic(3)
A =
     8     1     6
     3     5     7
     4     9     2
>> A'
ans =
     8     3     4
     1     5     9
     6     7     2
```

Transposition turns a row vector into a column vector.

3.1.3 Matrix Transpose

For a complex vector or matrix, `z`, the quantity `z'` denotes the complex conjugate transpose, where the sign of the complex part of each element is reversed.

The unconjugated complex transpose, where the complex part of each element retains its sign, is denoted by `z.'`

Example :Transpose of Complex vector

```
>> Z = [ 1+2*i 2+3*i 3+4*i]
Z =
   1.0000 + 2.0000i   2.0000 + 3.0000i   3.0000 +
  4.0000i
>> Z'
ans =
   1.0000 - 2.0000i
   2.0000 - 3.0000i
   3.0000 - 4.0000i
>> Z.'
ans =
   1.0000 + 2.0000i
   2.0000 + 3.0000i
   3.0000 + 4.0000i
```

Table3.2 Functions of getting matrix information

Function	***Description***
`length`	Return the length of the longest dimension. (The length of a matrix or array with any zero dimension is zero.)
`ndims`	Return the number of dimensions.
`numel`	Return the number of elements.
`size`	Return the length of each dimension.

Example:

```
>> length(d)
ans =
     5
>> ndims(d)
ans =
     2
>> numel(d)
ans =
    15
>> size(d)
ans =
     3     5
```

```
Here d is,
d =
     1     2     3     1     1
     2     3     4     1     1
     3     4     5     1     1
```

3.2 Array and Matrix Operations(1)

- Array operations are operations performed between arrays on an elements by elements basis. Matrix Operations follow the normal rules of linear algebra. such as matrix multiplication. In linear algebra, the product `c=a×b` is defined by the equation

$$c(i,j)=\sum_{k=1}^{n} a(i,k)b(k,j)$$

3.2 Array and Matrix Operations(2)

- MATLAB uses a special symbol to distinguish array operations from Matrix operations.
- MATLAB uses a period before symbol to indicate an array operator. For example, Matrix Multiplication operator is “ * ”, and Array Multiplication operator is “ .* ” .
- The following slides show the list of operators.

(1) Array and Matrix + , -

syntax	*operator*	*Description*
`A+B`	**+**	Array addition and Matrix addition are identical. Both arrays must be the same shape, or one of them must be a scalar.
`A-B`	**-**	Array subtraction and Matrix subtraction are identical. Both arrays must be the same shape, or one of them must be a scalar.

(2) Array and Matrix Multiplication

Syntax	*operator*	*Description*
`A.*B`	`.*`	Element-by-element multiplication of `A` and `B`. Both arrays must be the same shape, or one of them must be a scalar.
`A*B`	`*`	Matrix multiplication of `A` and `B`. The number of column in `A` must be equal to the number of row in `B`, or one of them must be a scalar.

(3) Array Division

Note : Both arrays must be the same shape, or one of them is a scalar.

syntax	***operator***	***Description***
`A./B`	`./`	Array Right Division. Element-by-element division of `A` and `B`: `A(i,j)/B(i,j)`
`A.\B`	`.\`	Array Left Division. Element-by-element division of `A` and `B`: but with b in the numerator: `B(i,j)/A(i,j)`.

(4) Matrix Division

syntax	***Operator***	***Description***
`A / B`	`/`	Matrix Right Division. Matrix division defined by `A*inv(B)`, where `inv(B)` is the inverse of matrix `B`.
`A \ B`	`\`	Matrix Left Division. Matrix division defined by `inv(A)*B`, where `inv(A)` is the inverse of matrix a.

(5) Matrix and Array Power

syntax	***operator***	***Description***
`A.^B`	`.^`	Array Power Element-by-element exponentiation of `A` and `B`: `A(i,j)^B(i,j)`. Both arrays must be the same shape,or one of them is scalar.
`M^p`	`^`	Matrix power Calculate the A^p

See chpt3_1.m

3.3 Solving Linear Systems of Equations

- 3.3.1 For Square Systems

```
AX=B
```

- Where `A` is a `n×n` nonsingular square coefficient matrix, `X` and `B` both are `n×1` column vector.

The solution is:

```
X=A\B
```

```
OR              X = inv(A)*B

                X = A^-1 *B;
```

Example: Solving the linear system

```
% sample of Matrix left division operator.
% M-file name is leftdvs.m
A = [1 2 1 4;2 0 4 3;4 2 2 1;-3 1 3 2];
B = [ 13 28 20 6]';
X = A\B;
Y = X';
disp('X=');   disp(X);
str = num2str(Y);   disp(['The X is ',str]);
```

The result after running M-file

- ***The solution is***

```
>> leftdvs

X=

    3.0000

   -1.0000

    4.0000

    2.0000
```

3.3 Solving Linear Systems of Equations

3.3.2. Overdetermined Systems: Overdetermined systems of simultaneous linear equations are often encountered in various kinds of curve fitting to experimental data.

For example,

(1) The experimental data

t	y
0.0	0.82
0.3	0.72
0.8	0.63
1.1	0.60
1.6	0.55
2.3	0.50

(2) The curve model is $y(t) = c1 + c2e^{-t}$

3.3 Solving Linear Systems of Equations

(3) Create the Coefficient Matrix E

```
E =[ones(size(t)) exp(-t)]
```

(4) Solving the Over-determined Systems

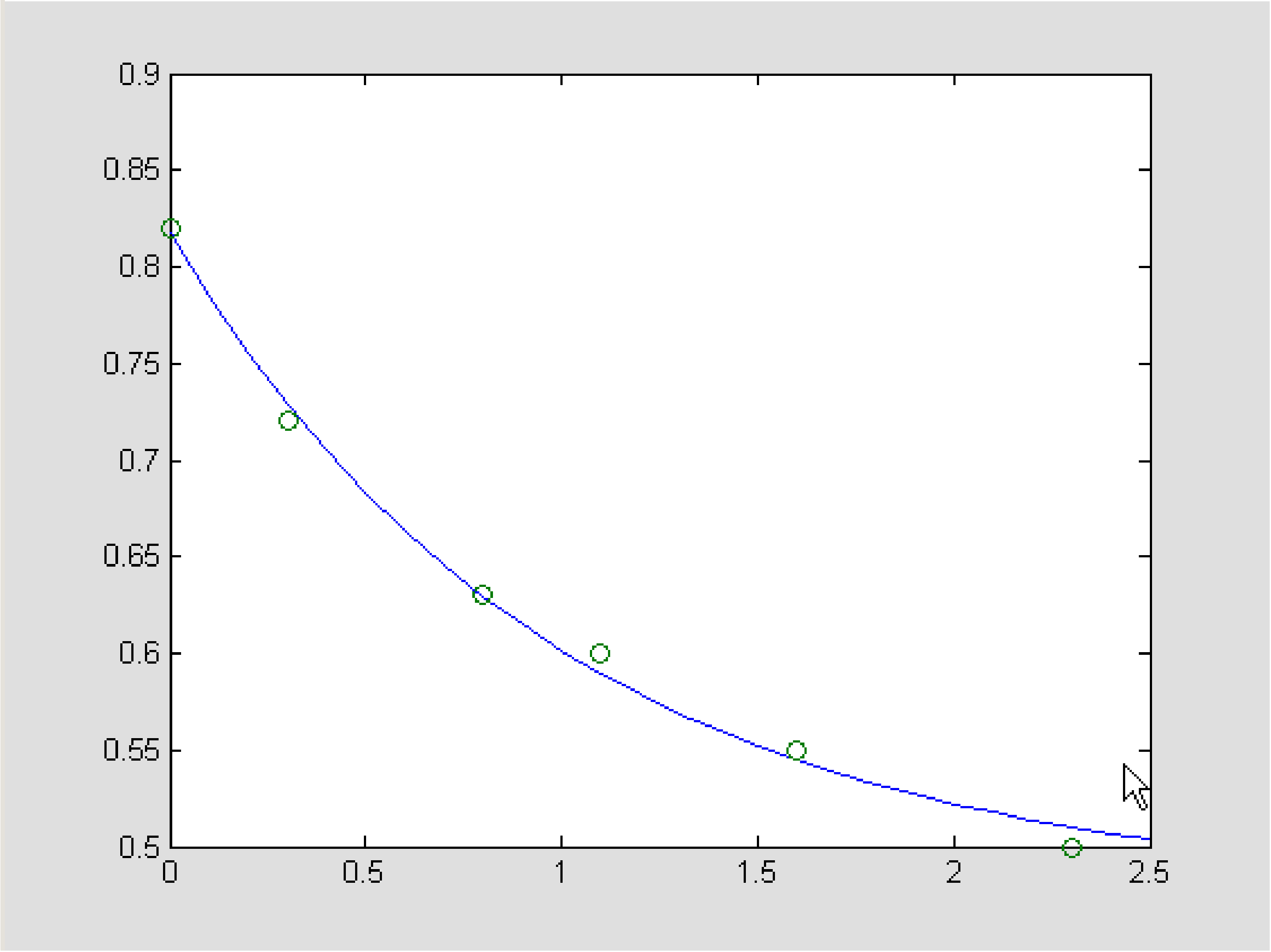
```
E*c = y

c =E\y
```

(5) Plotting the fitting curve.

See M-file chpt3_2.m

Example: Solving Linear Systems of Equations

```
% Example of Overdetermined systems of
% simultaneous linear equations.
% M-file name :chapt3-1.m
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';
% create the Coefficient Matrix E.
E = [ones(size(t)) exp(-t)];   % solving E*c=y.
% Use the backslash operator
% to get the least squares solution, column vector c:
c =E\y;
% c =[0.4760 0.3413]'
% visulized the least squares solution
T = (0:0.1:2.5)';
Y = [ones(size(T)) exp(-T)]*c;
plot(T,Y,'-',t,y,'o');
```



0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0
0.5
1
1.5
2
2.5

3.4 Polynomial and it's operation

- For polynomial of degree n

$$P(x)=a_0x^n+a_1x^{n-1}+\ldots+a_{n-1}x+a_n$$

- MATLAB denotes the coefficient vector as

$$P = [a_0\ a_1\ \ldots\ a_{n-1}\ a_n\ ]$$

- MATLAB provides a set of following functions to manipulate the polynomials.

3.4.1 Polynomial functions

1) `roots` - Find polynomial roots.
2) `poly` - Convert roots to polynomial.
3) `polyval` - Evaluate polynomial.
4) `polyvalm` - Evaluate polynomial with matrix argument.
5) `polyfit` - Fit polynomial to data.
6) `polyder` - Differentiate polynomial.
7) `polyint` - Integrate polynomial analytically.
8) `conv` - Multiply polynomials.
9) `deconv` - Divide polynomials.

3.4.2 Using Polynomials functions

Consider $P(x) = -0.02x^3+0.1x^2-0.2x+1.66$

a) Find the value of `P(4)` and the value of `P(x)` at

`x = [1 2 3 4 5]`

b) Find the roots of `P(x) = 0`

c) Plot the curve of `P(x)` at interval `[1,6]`

d) Find the definite integral of `P(x)` taken over `[1,4]`

e) Find `P'(4)`

See chpt3_3.m

Curve fitting with `polyfit()`

- *MATLAB provides two functions for modeling your data by using a polynomial function.*
- Polynomial Fit Functions

`polyfit(x,y,n)` finds the coefficients of a polynomial `p(x)` of degree n that fits the data y by minimizing the sum of the squares of the deviations of the data from the model (least-squares fit).

Example of function `polyfit()`

- Suppose we measure a quantity y at several values of time t:

```
t  : 0    0.3   0.8   1.1   1.6  2.3
y : 0.5  0.82  1.14  1.25  1.35  1.40
```

The data can be modeled by a polynomial function

$$y=a_0t^2+a_1t+a_2$$

Using least square fit method to find the polynomial coefficients `a0,a1,a2` with function

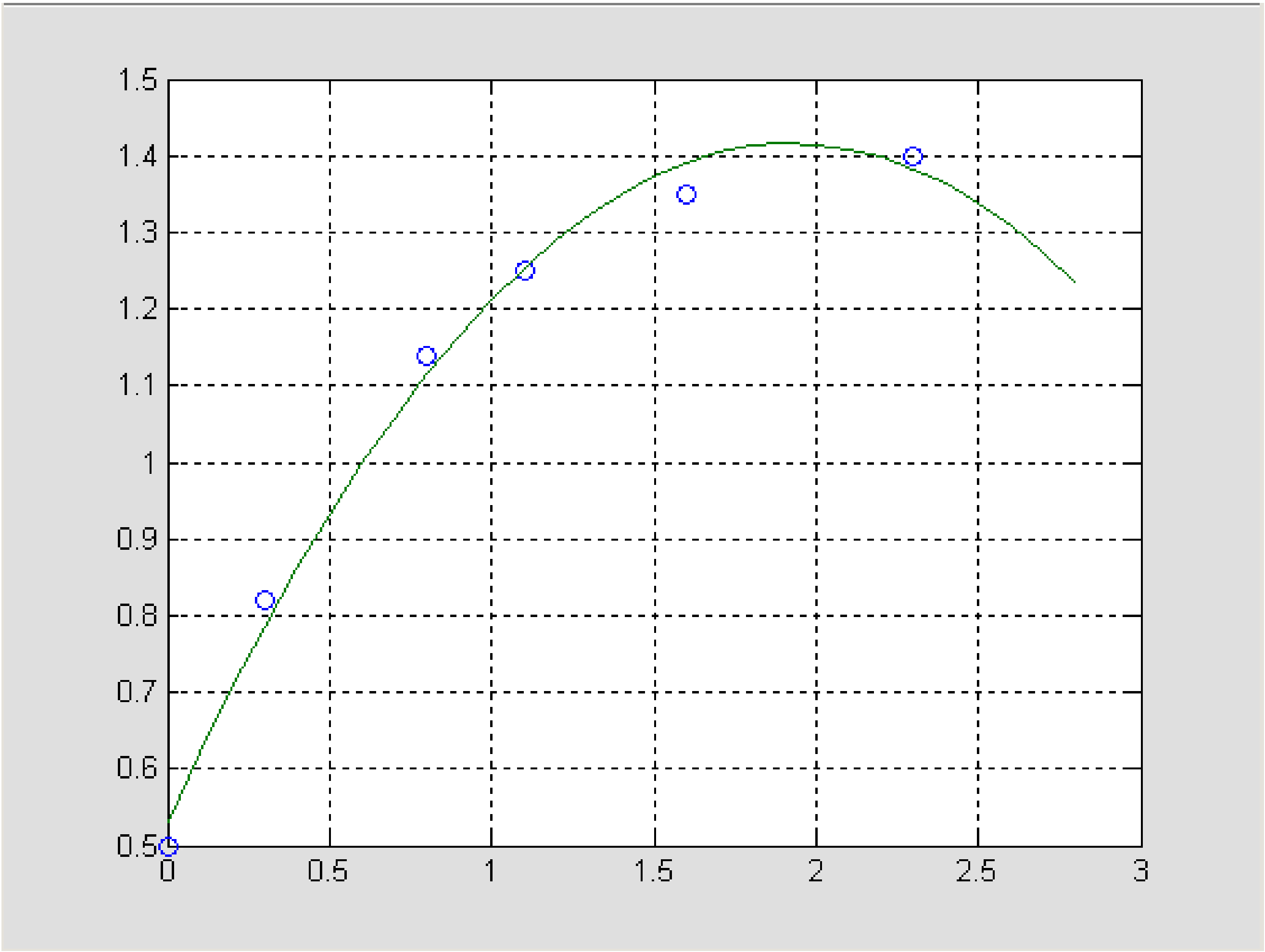
```
p=polyfit(t,y,2)
```

See chpt3_4.m ,that shows the function `polyfit`.

Example of function `polyfit()`

```
% The example of polynomial fit function
% M-file name chapt3_2.m

% create the data column vector
t = [0 .3 .8 1.1 1.6 2.3]';
y = [0.5 0.82 1.14 1.25 1.35 1.40]';
p=polyfit(t,y,2);
% Define a uniformly spaced time vector
t2 = 0:0.1:2.8;
% Evaluate the polynomial on a specific
% independent variable t2
y2=polyval(p,t2);
% plot the origenal data and fit curve.
plot(t,y,'o',t2,y2);
grid on;
```

The result of `polyfit()`



3.5 Relational and logical operators

3.5.1 Relational operators

- The relational expression has the form ***a1 op a2***

Where ***a1*** ***,a2*** are arithmetic expression, variables or string.

The ***op*** is one of the following relational operators.

== **equal to**	~= **Not equal to**
> **greater than**	>= **greater than or equal to**
< **less than**	<= **less than or equal to**
1 **for true**	0 **for false**

3.5 Relational & Logical Operators

3.5.2 logical operators

The logical expression has the forms.

l1 op l2 **or** ***op l1***

Where the *l1* and *l2* are logical expression or variables, and ***op*** is the one of the logical operators shown in the follows.

~	**logical NOT**	&	**logical AND**
|	**logical OR**	xor	**logical Exclusive OR**

- MATLAB interprets a zero value as false, and any nonzero values as true.

3.5 Relational & Logical Operators

3.5.3 Examples

```
>> y2 = [ 1 0 2 4]&[1 0 i 0];
??? Error using ==> and
Operands must be real.

>> y2 = [1 0 2 4]&[ 1 0 1 0]

y2 =

     1     0     1     0
```

```
>> x = 2.5
x =
   2.5000
>> y = (x>1.0)&(x<5.0)
y =
     1
>> y1 = [1 2 3 4 5] >=[ 5 4 3 2 1]
y1 =
     0     0     1     1     1
>> y2 = [1 0 2 4] & [ 1 0 i 0]
y2 =
     1     0     1     0
```

3.5.4 Using Logicals in Array Indexing

- The logical vectors created from logical and relational operations can be used to reference subarrays. Suppose `X` is an ordinary matrix and `L` is a matrix of the same size that is the result of some logical operation. Then `X(L)` specifies the elements of `X` where the elements of `L` are nonzero.

3.5.4 Examples of Using Logical indexing

- This example creates logical array `B` that satisfies the condition `A > 0.5`, and uses the positions of ones in `B` to index into `A`. This is called logical indexing:

```
>> A = rand(5);   %create a 5×5 matrix(array).
>> B = A > 0.5;   %generate a same size logical matrix(array).
>> A(B) = 0         %Logical indexing.
A =
    0.2920    0.3567    0.1133         0    0.0595
         0    0.4983         0    0.2009    0.0890
    0.3358    0.4344         0    0.2731    0.2713
         0         0         0         0    0.4091
    0.0534         0         0         0    0.4740
```

See Example chpt3_6.m

3.5.4 Examples of Using Logical indexing

- This example highlights the location of the prime numbers in a magic square using logical indexing to set the nonprimes to 0:

```
>> A = magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
 >> B = isprime(A)
B =
     0     1     1     1
     1     1     0     0
     0     1     0     0
     0     0     0     0
>> A(~B) = 0;                          % Logical indexing
 >> A
A =
     0     2     3    13
     5    11    0     0
     0     7     0     0
     0     0     0     0
```

3.6 Some Conversion Functions

- **1. int8, int16, int32, int64: Convert to signed integer**

`I = int?(X)` converts the elements of array `X` into signed integers. double and single values are rounded to the nearest `int?` value on conversion.

- **2. ceil Round toward infinity**

`B=ceil(A)` rounds the elements of `A` to the nearest integers greater than or equal to `A`.

- **3. fix Round towards zero.**

`B = fix(A)` rounds the elements of `A` toward zero, resulting in an array of integers.

3.6 Some Conversion Functions

4. floor Round towards minus infinity.

`B = floor(A)` rounds the elements of `A` to the nearest integers less than or equal to `A`.

5. round to nearest integer.

`Y = round(X)` rounds the elements of `X` to the nearest integers.

6. `mod` - Modulus (signed remainder after division).

7. `rem` - Remainder after division.

See Example chpt3-7.m

3.7 The precedence of Operators

1. Parentheses **()**
2. Transpose (**.'**), complex conjugate transpose (**'**), matrix power (**^**) , power (**.^**)
3. Unary plus (**+**), unary minus (**-**), logical negation (**~**)
4. Multiplication (**.***), right division (**./**), left division(**.**), matrix multiplication (*****), matrix right division (**/**), matrix left division (****)
5. Addition (**+**), subtraction (**-**)
6. Colon operator (**:**)
7. Less than (**<**), less than or equal to (**<=**), greater than (**>**), greater than or equal to (**>=**), equal to (**==**), not equal to (**~=**)
8. Element-wise AND (**&**)
9. Element-wise OR (**|**)

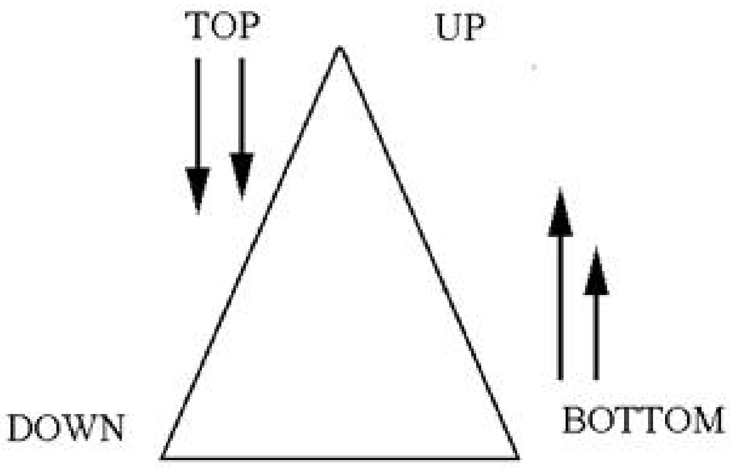
3.8 Introduction to Top-Down Design Techniques

- Top-down

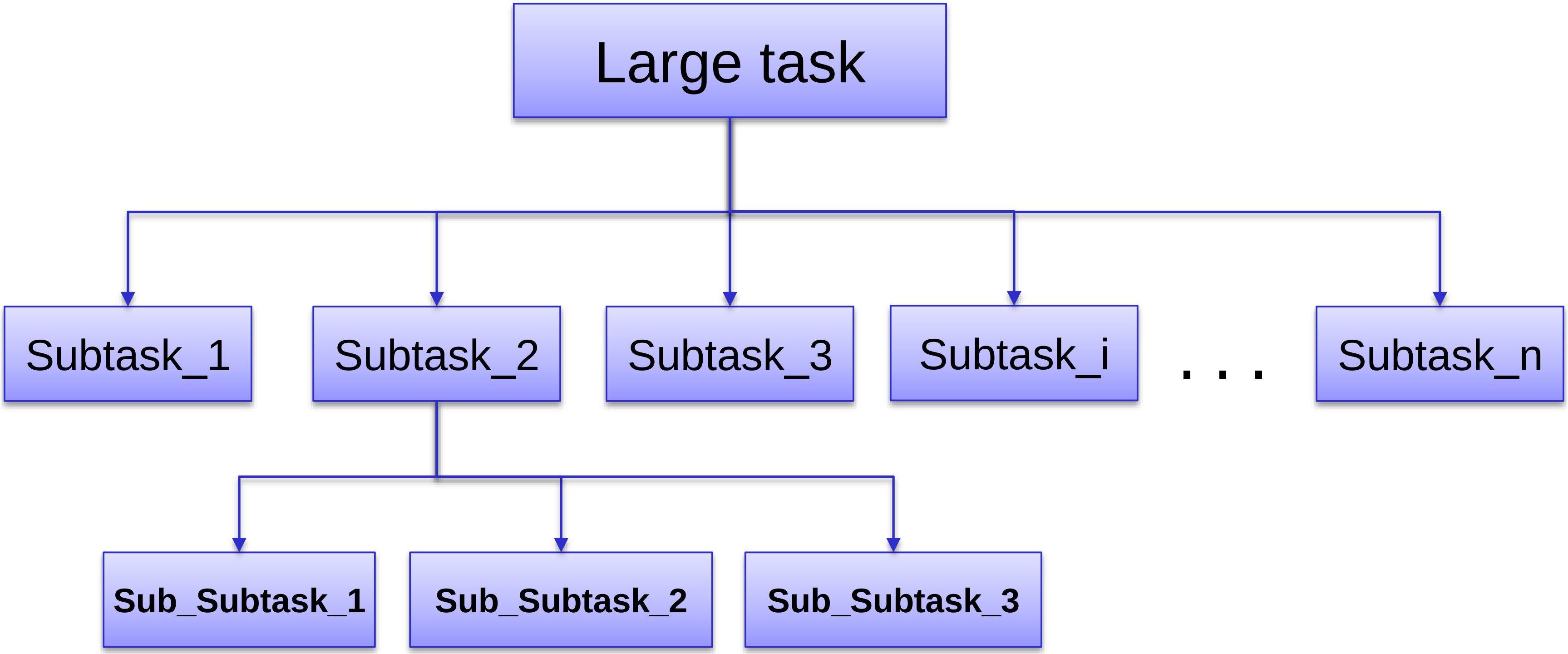
  The process of starting with a large task and breaking it down into smaller, more easily understandable subtasks, which perform a portion of the desired task.

- Bottom-up

  The piecing together of systems to give rise to grander systems, thus making the original systems sub-systems of the emergent system.



3.8 Introduction to Top-Down Design Techniques



- Task Division
- Interface Design

3.8 Introduction to Top-Down Design Techniques

The 5 steps to write a program:

1. Clearly state the problem that you are trying to solve.
2. Define the Input and the Output data.
3. Design or choose the algorithm (pseudocode)
4. Coding convert the algorithm into MATLAB statements
5. Test and debug the MATLAB program.

Design or choose the algorithm.

The techniques used in this step are :

(1) The pseudocode is the hybrid mixture of MATLAB and English.

(2) Stepwise refinement

For example, to solve for roots of the quadratic equation

$$ax^2 + bx + c=0$$

The pseudocode as following:

The first level pseudocode

1) Read in the data a,b and c.

```
a=input('a=?');
b=input('b=?');
c=input('c=?');
```

2) Calculate discriminant : $disc = b^2 - 4ac$

```
disc=(b^2-4*a*c);
```

3) Calculate the roots

If disc>0 there are two distinct real roots

If disc=0 there are two identical real roots.

If disc<0 there are two complex roots.

4) Write out the roots.

Coding

- When the refinement process was carried out very properly, then this step will be very simple. That the programmer will have to do is to replace the pseudocode with the corresponding MATLAB statement line by line.

- See calc_root.m

Test and debug the MATLAB program.

Three types of errors can be found in MATLAB program

1. ***Syntax error*** detected by MATLAB compiler
2. ***Run-time error*** such as divided by zero
3. ***Logical error*** It occurs when the program compiles and runs successfully but produces the wrong answer. It is the most difficult to be found.

Time spent by Programming

For large software project

- Step 1 to step 3 may spend 30~35% time
- Step 4 may spend 15~20% time
- Step 5 will spend 45~55% time

Tip: pay much attention to TEST.

Typical testing process for large program

- **Unit testing:**
  - verifies the functionality of a specific section of code, usually at the function level. It tests individual subtasks.
- **integration testing:**
  - verifies the interfaces between components against a software design. big-bang, mixed (sandwich), top-down, and bottom-up.
- **System testing:**
  - tests a completely integrated system to verify that the system meets its requirements.
- **Operational acceptance testing:**
  - is used to conduct operational readiness (pre-release) of a product, service or system as part of a quality management system.

Test cases

- A specification of the inputs, execution conditions, testing procedure, and expected results that define a single test to be executed to achieve a particular software testing objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

	A	B	C	D	E	F	G	H	I	J	K
1	Test Case ID		BU_001	Test Case Description		Test the Login Functionality in Banking					
2	Created By		Mark	Reviewed By		Bill		Version		2.1	
3											
4	QA Tester's Log		Review comments from Bill incorporated in version 2.1								
5											
6	Tester's Name		Mark	Date Tested		1-Jan-2025		Test Case (Pass/Fail/Not		Pass	
7											
8	S #	Prerequisites:				S #	Test Data				
9	1	Access to Chrome Browser				1	Userid = mg12345				
10	2					2	Pass = df12@434c				
11	3					3					
12	4					4					
13											
14	Test Scenario	Verify on entering valid userid and password, the customer can login									
15											
16	Step #	Step Details		Expected Results		Actual Results			Pass / Fail / Not executed / Suspended		
17											
18	1	Navigate to http://demo.guru99.com		Site should open		As Expected			Pass		
19	2	Enter Userid & Password		Credential can be entered		As Expected			Pass		
20	3	Click Submit		Cutomer is logged in		As Expected			Pass		
21	4										
22											

Test cases

Write test cases for calc_root.m as much as possible:

Test Case ID	Description	Input	Expected results	Actual Result	Pass/Fail

Test cases

Write test cases for calc_root.m as much as possible:

Test Case ID	Description	Input	Expected results	Actual Result	Pass/Fail
1	Test two distinct real roots	a=1 b=3 c=2	X1=-1 X2=-2	X1=-1 X2=-2	Pass
2	Test identical real roots	a=1 b=2 c=1	X1=-1 X2=-1	X1=-1 X2=-1	Pass
3	Test two complex roots	a=1 b=1 c=1	X1=-0.5 - 0.866i X2=-0.5 + 0.866i	X1=-0.5 - 0.866i X2=-0.5 + 0.866i	Pass
4	With special input	a=0 b=1 c=1	X1=-1	?	?
5	with wrong input	?	……	……	……
6	……	……	……	……	……

3.9 Flow Control---branches

- Like other programming language, MATLAB also has branches and loop flow control statements.

- The branch statements include *if* and *switch* statements.

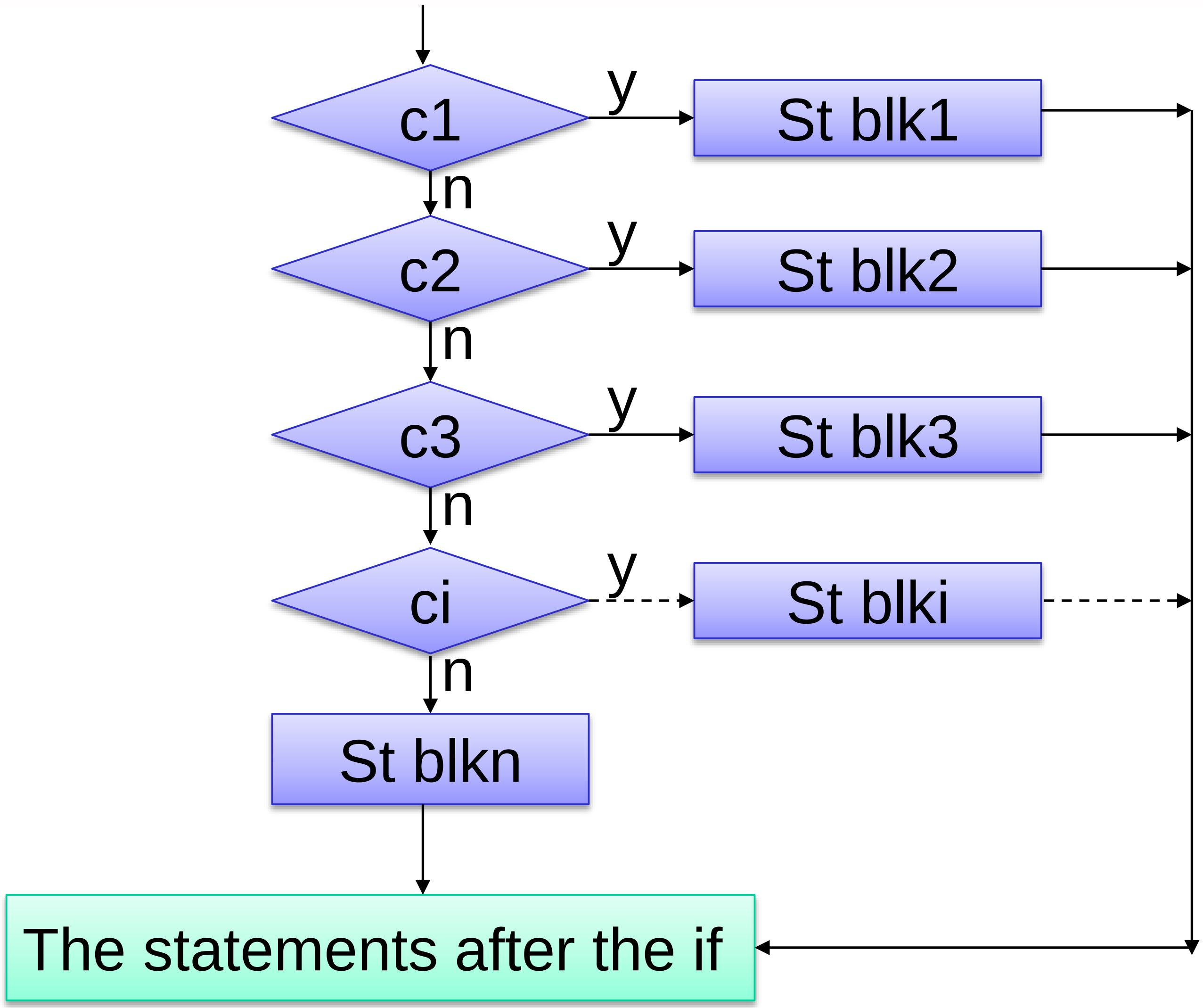
3.9.1 `if` statement(1)

The if construct has the form

```
if  control_expression_1
        statements-block1
elseif   control_expression _2
         statements-block2
 ......
else
           statements-blockn
end
```

In the `if` statement there can be any number of `elseif` clause(0 or more), and can be zero or at most one `else` clause .

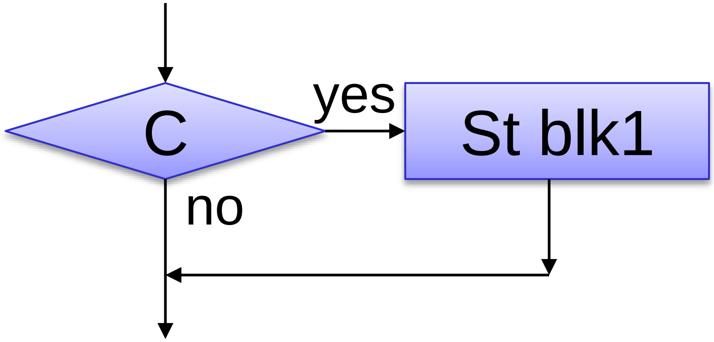
3.9.1 `if` statement construct (2)



3.9.1 `if` statement(3)

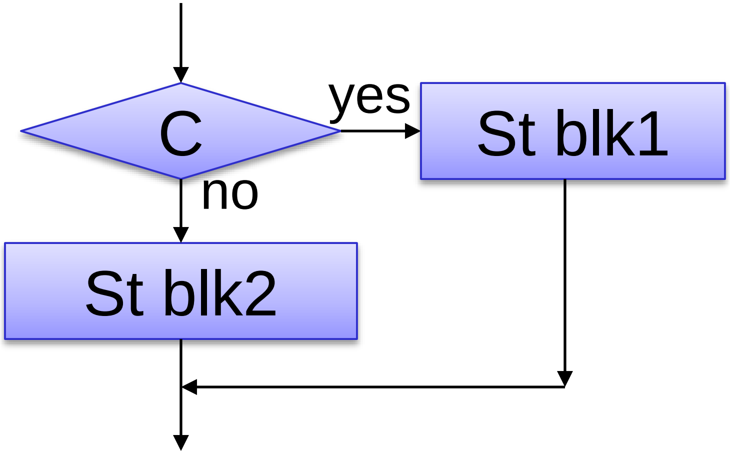
The simplest if construct form:

```
if  control_expression
        statements_block
end
```



The two way branches

```
if control_expression
      statements_block1
else
      statements_block2
end
```



If statement can be nested

3.9.2 **`switch`** statement(1)

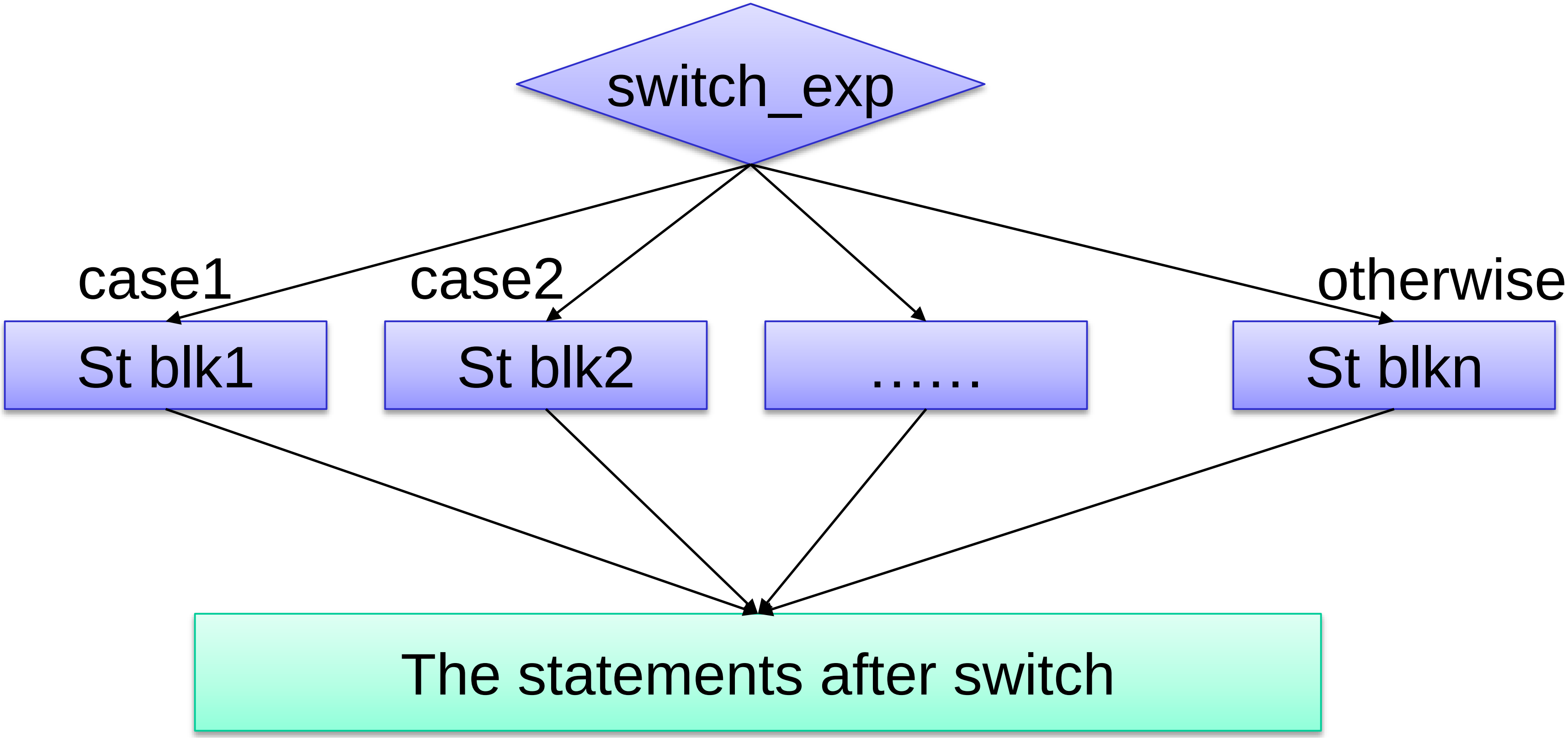
- The switch statement has the form

```
switch ( switch_expression)
      case  case_expr_1,
            statements_block1
      case  case_expr_2,
            statements_block2
       ......
      otherwise
           statements_blockn
   end
```

Note : unlike C language, after each statements block the *break* statement is not required.

The *`switch_expression`* and each *`case_expr`* may be either numerical or string value

3.9.2 **switch** statement(2)



See example : run the M_file d2h.m

3.9.3 Error control: The try/catch Statement

- The `try/catch` construct is a special form branching construct designed to trap errors.
- The general form of a `try/catch` construct is:

```
try
   statement
   ...                 Try block
   statement
catch
   statement
   ...                 Catch block
   statement
end
```

3.9.3 The try/catch statement

- When a `try/catch` statement is reached, the statements in the try block will be executed if no errors occurs, the statements in the catch block will be skipped and execution will continue at the first statement following the end of the `try/catch` statement,
- if an error does occur in the try block, and immediately execute the statements in the catch block.

3.10 The loop statements(1)

1. The while statement has the form

```
while expr
    statements block
end
```

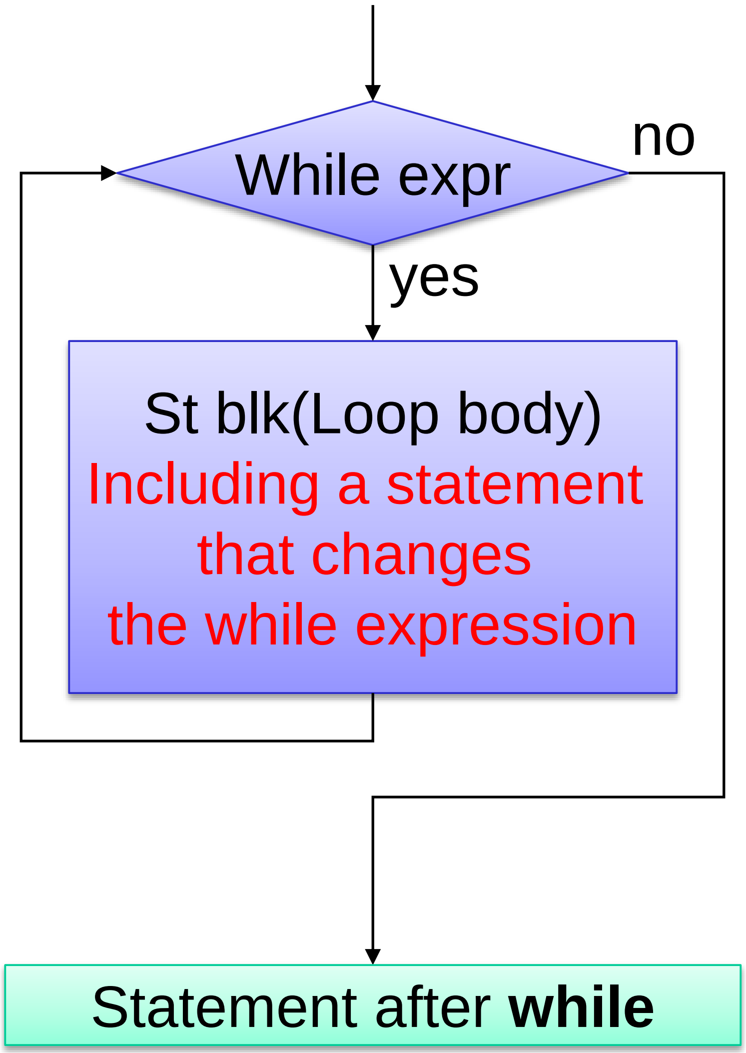
2. The for statement has the form

```
for  index = first: incr: last
    statements block
end
```

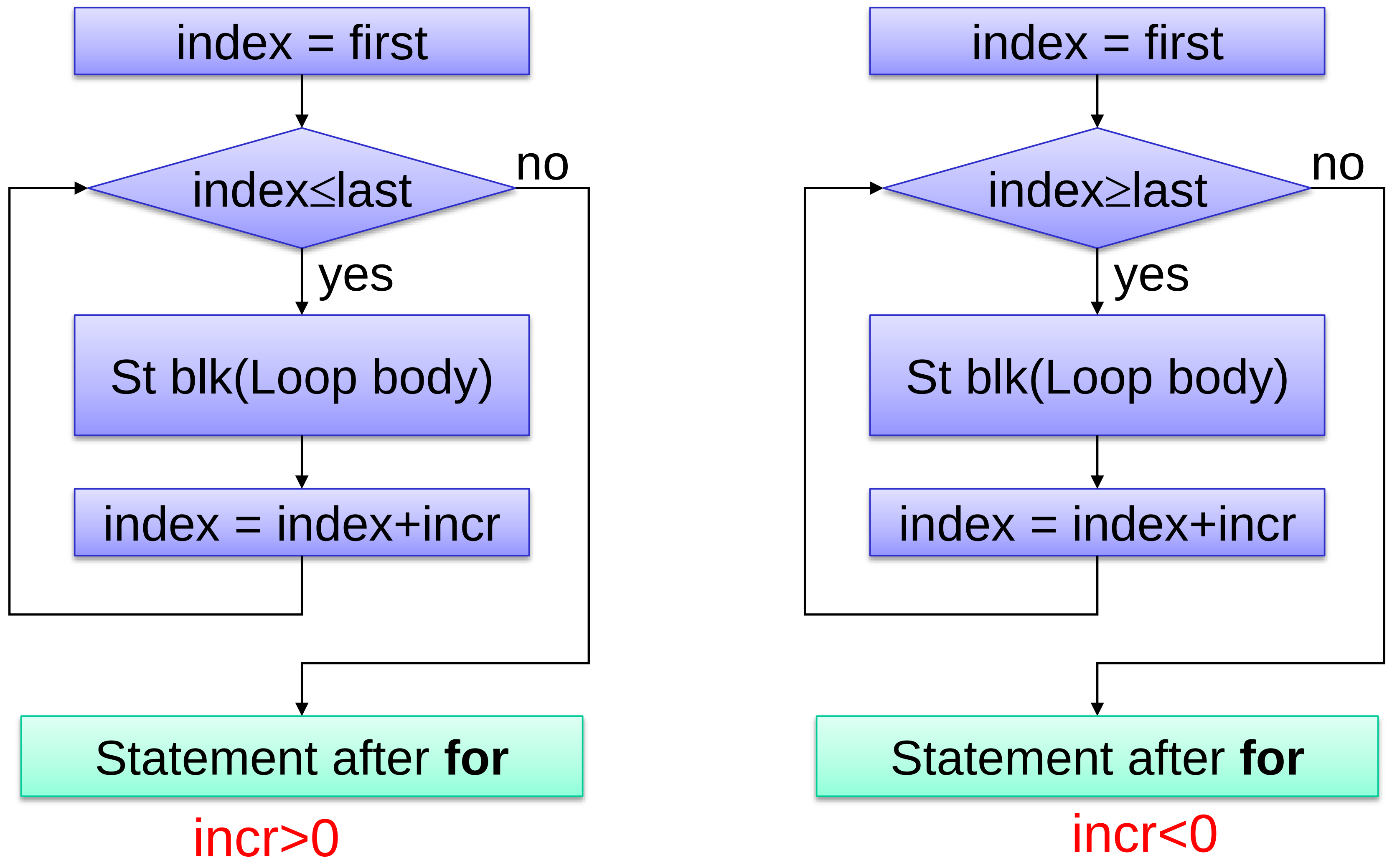
3.10 The loop statements(2)

- In the `while` loop, if the value of `expr` is non zero `true(1)`, then the statements block executes, and the control returns to `while`. The process will be repeated until the expression becomes `false(0)`.
- In the `for` loop, the `index` is loop variable, for each value of `index` from first to last, the loop body executes repeatedly.
- The loop construct can be nested.

3.10.1 `while` and `for` statement diagram



3.10.1 `while` and `for` statement diagram



`break` statment

- `break` Terminate execution of `while` or `for` loop.
- `break` terminates the execution of a `for` or `while` loop. Statements in the loop that appear after the `break` statement are not executed.
- In nested loops, `break` exits only from the loop in which it occurs. Control passes to the statement that follows the end of that loop.
- See also `Return.`

`continue` statment

- `continue` passes control to the next iteration of `for` or `while` loop in which it appears, skipping any remaining statements in the body of the `for` or `while` loop.

- In nested loops, `continue` passes control to the next iteration of `for` or `while` loop enclosing it.

The `for` loop examples

```
1.  for  ii = 1:10
   % ii =1,2,…10, execute 10 times
         statements
    end

2. for ii = 1:2:10
      % ii = 1,3,5,7,9, execute 5 times
         statements
    end

3. for ii = [3 5 7]
      % ii = 3,5,7, execute 3 times.
         statements
    end
```

The `for` loop example

```
% calculate the N!
n = input('Enter n :');
n_factorial = 1;
for ii = 1:n
    n_factorial = n_factorial*ii;
end
fprintf(' %d ! = % f \n',n,n_factorial);
```

3.11 Programming Examples

1. Write a program that converts a decimal number to Hexadecimal number.(d2h.m)

2. Write a program that finds the root of equation $f(x)=\cos(x)-x+1$ in [0.8,1.6] with Bisection method.

3. Write a program that sorts the given data set in ascending order with Bubble Sorting algorithm.

4. Different methods to solve pi in Matlab.

Example 1: Convert a decimal number to Hexadecimal number

- The converting algorithm diagram is shown as follows.

For example

- Given a decimal number 1007, what is it's equivalent hexadecimal number?

$$(3EF)_{16} = 3\times 16^2 + 14\times 16 + 15 = 3\times 256 + 224 + 15 = 1007$$

$$(1007)_{10} = (3EF)_{16} = 3\times 16^2 + 14\times 16 + 15 = (3\times 16 + 14)\times 16 + 15$$

The converting algorithm

$$(3EF)_{16} = 3\times 16^2 + 14\times 16 + 15 = 3\times 256 + 224 + 15 = 1007$$

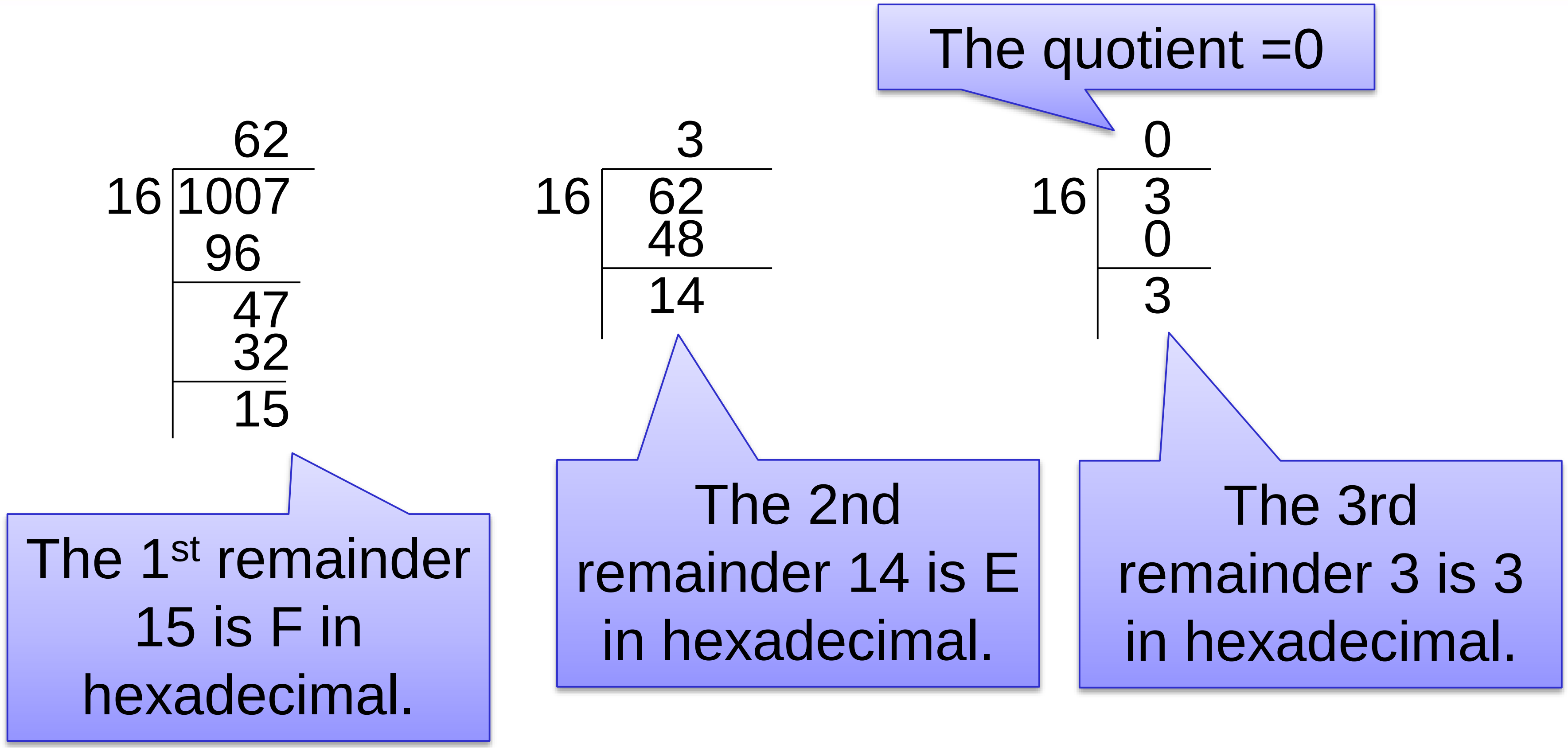
$$(1007)_{10} = 768 + 224 + 15$$

$$= 3\times 256 + 14\times 16 + 15$$

$$= 3\times 16^2 + 14\times 16 + 15 = (3EF)_{16}$$

$$= ((3\times 16 + 14)\times 16 + 15$$

D to H converting algorithm diagram



$$\mathbf{(1007)_{10} = (3EF)_{16}}$$

See example : run the M_file d2h.m & dtoh.m

Example 2. Find the root of $f(x)=\cos(x)-x+1=0$ with Bisection Method.

- Bisection method can find the root of $f(x)=0$, where the $f(x)$ is a continuous function within interval $[a,b]$.

- $f(a)$ and $f(b)$ have opposite signs, or $f(a)*f(b) < 0$.

- The Bisection algorithm

The Bisection algorithm

(1) calculate the midpoint **xm=(a+b)/2**

(2) **if f(a)*f(xm) <0**, the root lies in[a,xm]

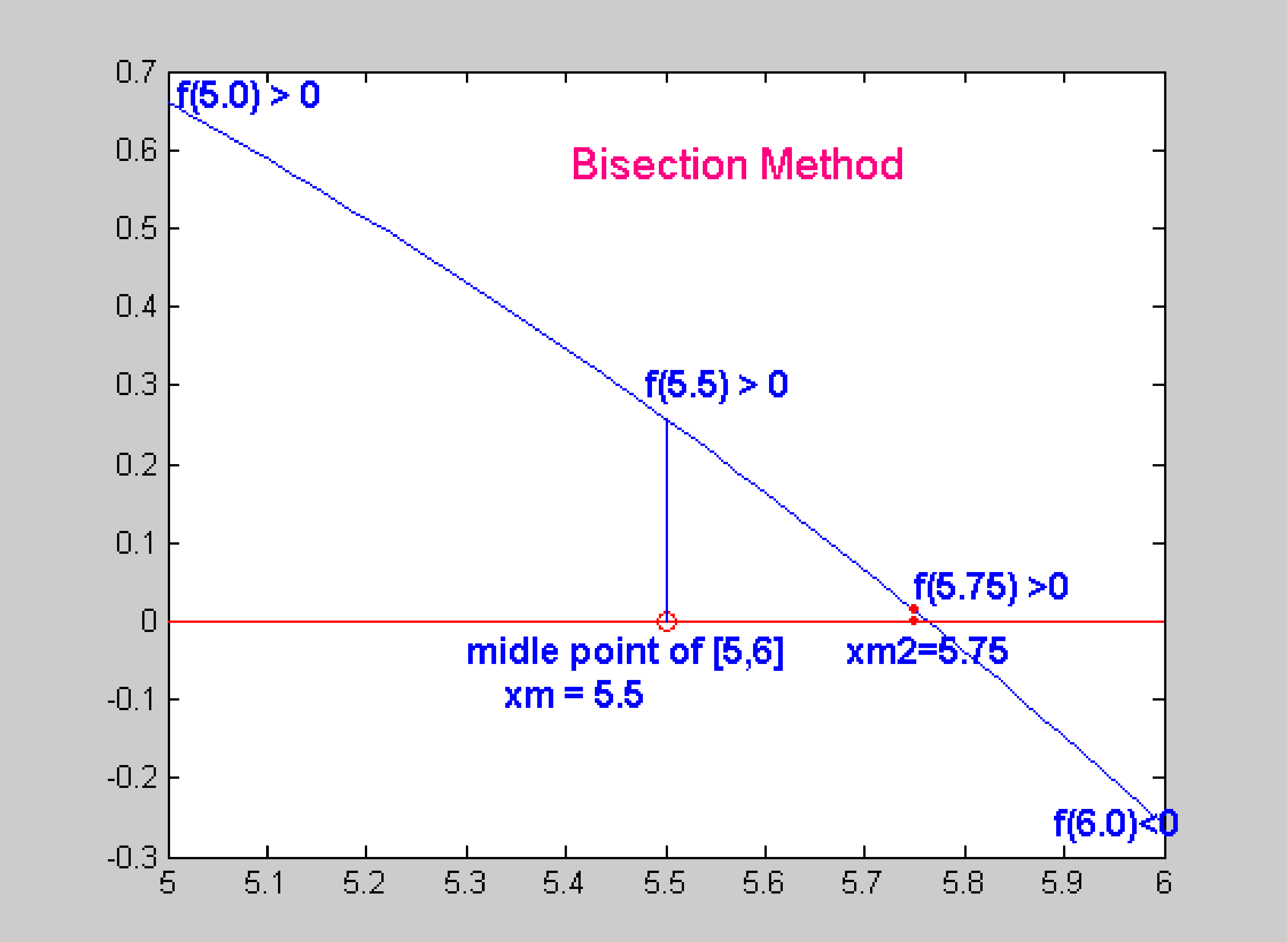
(3) **if f(b)*f(xm)<0**, the root lies in [xm,b]

(4) **if f(xm)=0**, then the root is xm.

The interval [a1,b1] is the half of [a,b]

See bsct.m

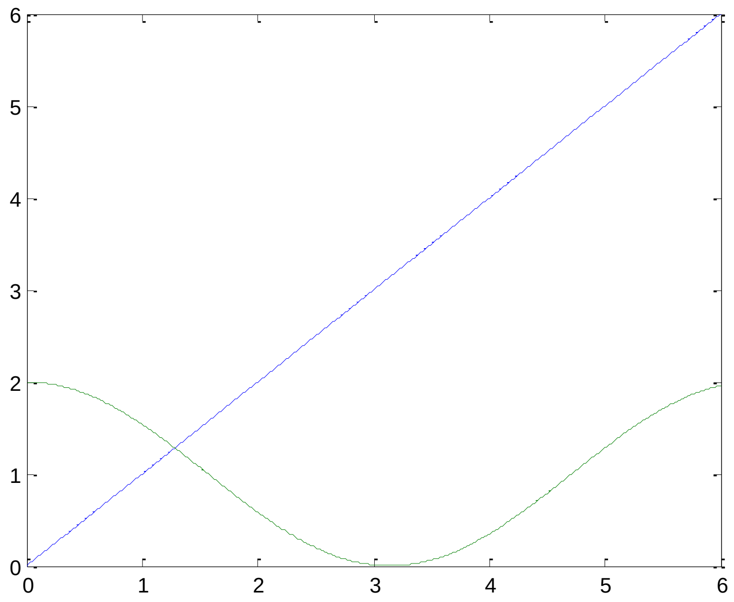
The Fig shows the procedure of Bisection method.



Example 2. Find the root of f(x)=cos(x)-x+1=0 with while.

- f(x)=0 -> x = 1+cos(x)

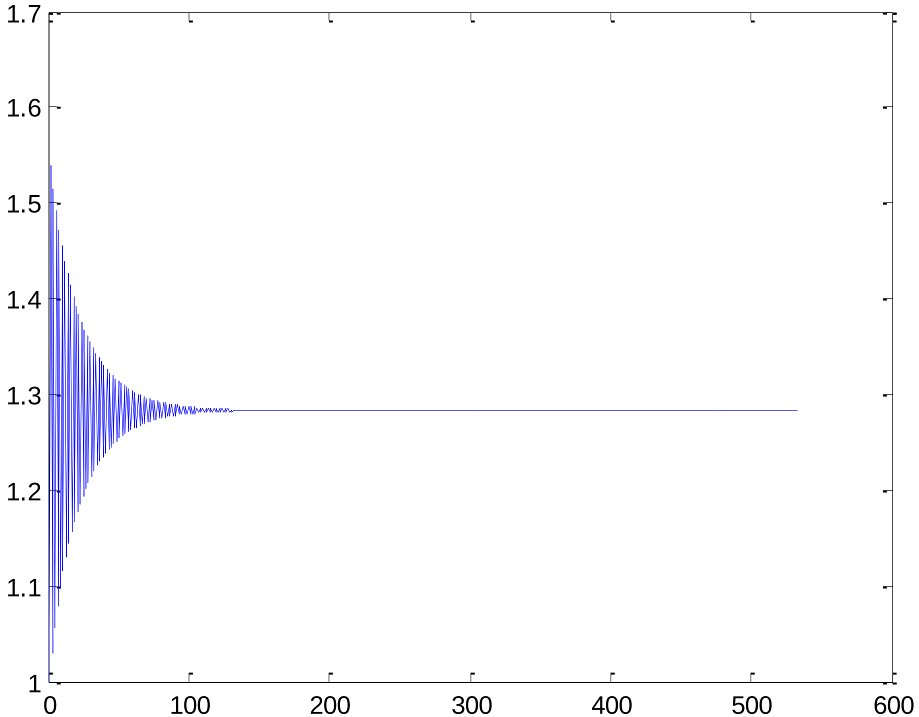
```
x=0:0.01:6;

y1 = x;

y2 = 1+cos(x);

figure,plot(x,y1,'-',x,y2,'-')
```



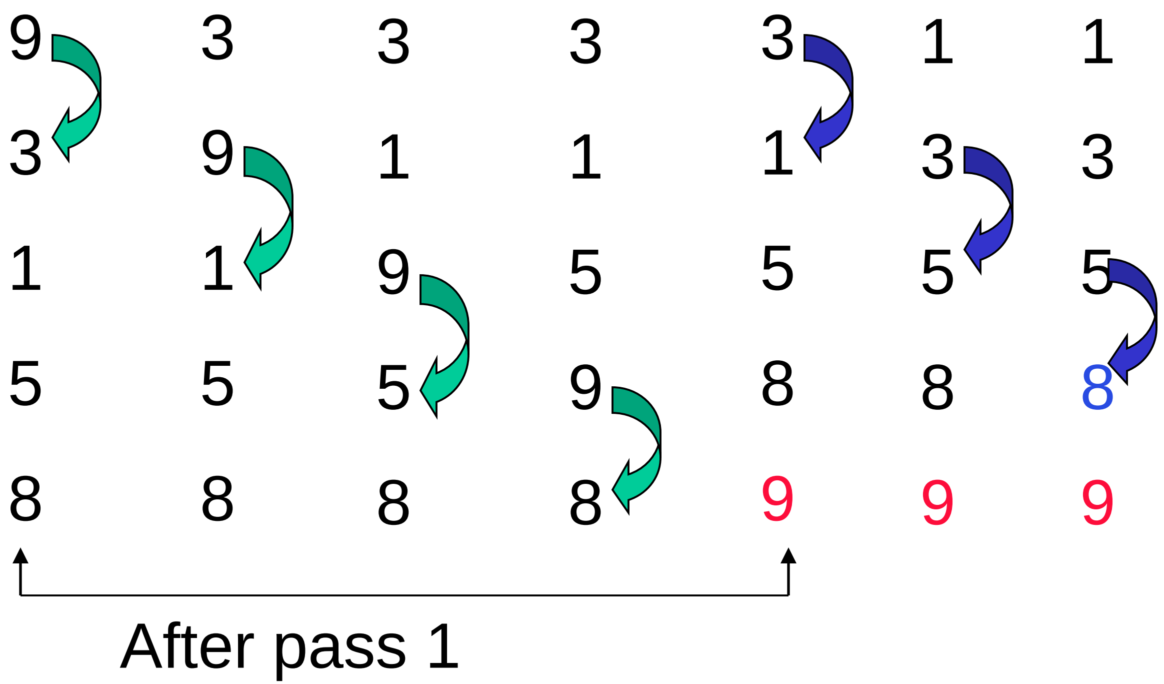
Example 3. Find the root of f(x)=cos(x)-x+1=0 with while.

- f(x)=0 -> x = 1+cos(x)

```
k=1;
delta = 1e-10;
while x~=1+cos(x)
   x0(k)=x;
   x=1+cos(x);
   if(abs(x-x0(k))<delta)
      break;
   end
   k=k+1
end
```



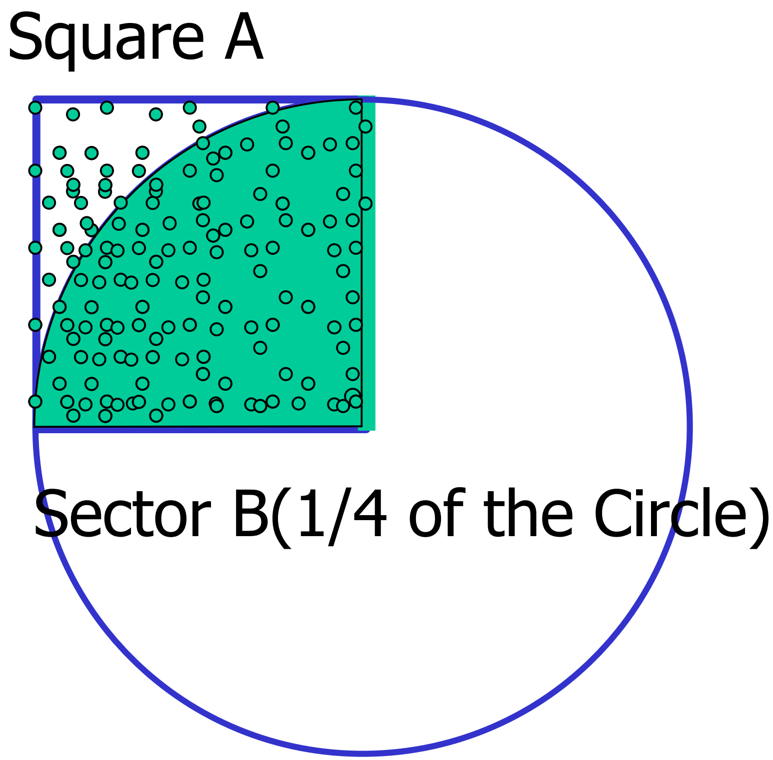
Example 3. Bubble Sort algorithm diagram



See and run the bbsort.m

Example 4. Different methods to solve pi

1.Monte Carlo Method



$$\mathrm{k} = \frac{\mathrm{S_B}}{S_A} = \frac{\mathrm{m}}{\mathrm{n}}$$

m equals to the number of dots in B

n equals to the number of dots in A

$$S_B = \frac{1}{4} S_{Circle} = \frac{1}{4} \pi R^2$$

$$\pi = \frac{4 S_B}{R^2} = \frac{4\mathrm{m}}{n}$$

Example 4. Different methods to solve pi

1.Monte Carlo Method

```
tic
i=1;m=0;n=1000;

for i=1:n
    a=rand(1,2);
    if a(1)^2+a(2)^2<=1
        m=m+1;
    end
end

p=vpa(4*m/n,30);%set 30 to Significant digit
toc
```

Example 4. Different methods to solve pi

1. Monte Carlo Method

```
>> MonteCarlo
Elapsed time is 0.131347 seconds.
>> p


p =


3.164
```

```
p=vpa(4*m/n,30);%set 30 to Significant digit
toc
```

Example 4. Different methods to solve pi

2. Taylor Series Method

Taylor series expansion formula:

$$f(x) = \frac{f(x_0)}{0!} + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_n(x)$$

Thus, arctan x can be expanded as follows:

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \cdots + (-1)^{k-1}\frac{x^{2k-1}}{2k-1} + \cdots$$

When x=1:

$$\frac{\pi}{4} = \arctan 1 = 1 - \frac{1}{3} + \frac{1}{5} - \cdots + (-1)^{n-1}\frac{1}{2n-1}$$

Then π can be calculated by this formula.

Example 4. Different methods to solve pi

2. Taylor Series Method

```
tic
i=1;n=1000;s=0;

for i=1:n
    s=s+(-1)^(i-1)/(2*i-1);
end

p=vpa(4*s,30); %set 30 to Significant digit
toc
```

Example 4. Different methods to solve pi

2. Taylor Series Method

```
>> TylorSeries
Elapsed time is 0.130800 seconds.
>> p

p =

3.14059265383979413499559996126
```

Example 4. Different methods to solve pi

2. Taylor Series Method

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \cdots + (-1)^{k-1}\frac{x^{2k-1}}{2k-1} + \cdots$$

When x=1/2 and x=1/3:

$$\alpha = arctan\frac{1}{2}$$

$$\beta = arctan\frac{1}{3}$$

$$\tan(\alpha+\beta) = \frac{tan\alpha + tan\beta}{1 - tan\alpha tan\beta} = 1$$

$$\alpha + \beta = arctan1 = \frac{\pi}{4}$$

Then we can use this formula to calculate π :

$$\frac{\pi}{4} = \arctan\frac{1}{2} + \arctan\frac{1}{3}$$

Example 4. Different methods to solve pi

2. Taylor Series Method

```
tic
i=1;n=1000;s=0;s1=0;s2=0;

for i=1:n
    s1=s1+(-1)^(i-1)*(1/2)^(2*i-1)/(2*i-1);
    s2=s2+(-1)^(i-1)*(1/3)^(2*i-1)/(2*i-1);
end

s=s1+s2;
p=vpa(4*s,30); %set 30 to Significant digit
toc
```

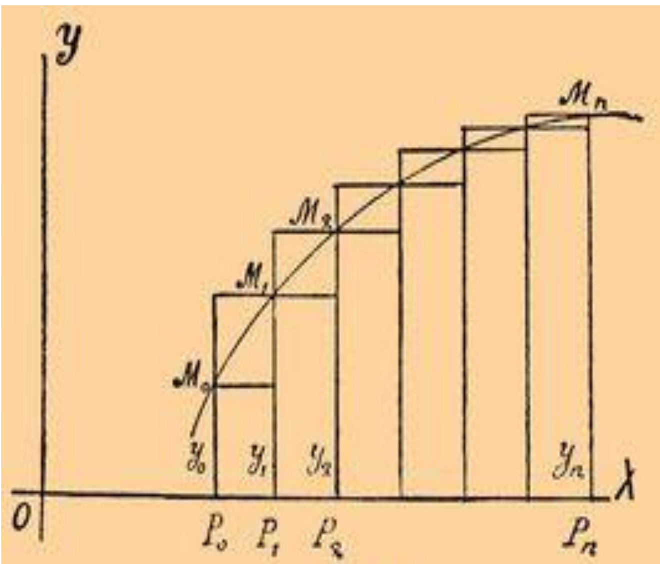
Example 4. Different methods to solve pi

2. Taylor Series Method

```
>> TylorSeries
Elapsed time is 4.728523 seconds.
>> p

p =

3.14159265358979323846264338328
```

Example 4. Different methods to solve pi

3. Numerical Analysis Method



Черт. 2.

$$\int_a^b f(x)dx = \sum_{i=1}^{n} f(\varepsilon i)\Delta xi$$

Divide $[a, b]$ into n equal parts $a,\ x_1,\quad x_2 \ldots x_{n-1}, x = a, x = b$

$$\int_0^1 \frac{1}{1+x^2}dx = \pi/4$$

$$\Downarrow$$

$$\pi$$

Example 4. Different methods to solve pi

3. Numerical Analysis Method

```
tic
s=0;n=1000;

for x=0:(1/n):(1-(1/n))
    s=s+(1/(1+x^2)+1/(1+(x+(1/n))^2))*(1/n)/2;
end

p=vpa(4*s,30);%set 30 to Significant digit
toc
```

Example 4. Different methods to solve pi

3. Numerical Analysis Method

```
>> NumericalAnalysis
Elapsed time is 0.101928 seconds.
>> p


p =


3.14159248692312775830259852228
```

A Frame of Interactive Program

The general form is:

```
yn = 1;
while yn == 1
 ……;
   the processing statements
 ……;
 yn = input(‘try it again? yes =1 no =0’);
end
```

`tic` and `toc` function

- `tic` and `toc`

  Measure performance using stopwatch timer.

`tic` : starts a stopwatch timer.

`toc` : prints the elapsed time since tic was used.

`t = toc` returns the elapsed time to `t`.

program frame

```
……;
tic
    statements  segment  which would be
  measured
toc
t = toc;
```

Homework 2

HW2-1. Write an M-file to make the following four variables.

(a) $A = \begin{bmatrix} 2 & \cdots & 2 \\ \vdots & \ddots & \vdots \\ 2 & \cdots & 2 \end{bmatrix}$ is a 6×6 matrix full of 2's (use ones or zeros).

(b) $B = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 2 & 0 & \cdots & 0 \\ 0 & 0 & 3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$ is a 7×7 matrix of all zeros, but with the values $\begin{bmatrix} 1 & 2 & 3 & 4 & 3 & 2 & 1 \end{bmatrix}$ on the main diagonal (use diag).

(c) $C = \begin{bmatrix} 1 & 11 & \cdots & 91 \\ 2 & 12 & \cdots & 92 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 20 & \cdots & 100 \end{bmatrix}$ is a 10×10 matrix where the vector 1:100 runs down the columns (use reshape).

(d) Make D be a 5×3 matrix of random integers with values on the range 0 to 10 (use rand and floor or ceil).

HW2-2. Assume that *a,b,c* and *d* are as defined, and evaluate the following expressions.

$$a = 2, b = \begin{bmatrix} -2 & 3 \\ 6 & 0 \end{bmatrix}, c = \begin{bmatrix} 0 & 3 \\ 2 & 0 \end{bmatrix}, d = \begin{bmatrix} -2.2 & -0.1 \\ 1.9 & 1.2 \\ 2.1 & 0.1 \end{bmatrix}$$

(a) e is the ceil round of d, output e.

(b) b*c

(c) b.*c

(d) ~(a>e)

(e) a>c & b>c

Submit homework online before Oct 15,2019

Homework 2

HW2-3. Using left division operator ('\') to solve curve fitting problem.

(a) Find the least squares parabola $f(x) = ax^2 + bx + c$ **for the set of data**

x_k	−2	−1	0	1	2
y_k	−9.8	-8.8	-6.3	-5.8	3.2

(b) Compare your results (a,b,c) with the results returned by builtin function p=polyfit(x,y,2).

(c) Plot the sample data points with blue circle marker and the fitting curve.

HW2-4. There is a kind of special three-digit numbers: narcissiatic number. Third power of the its single digit, tens digit and hundreds digits equal itself. For example:

$$1^3 + 5^3 + 3^3 = 153$$

write a script file to find all narcissistic numbers(no more than 1000).

HW2-5. Find the numerical solution of the equation $\frac{x}{2} = \sin x$ **with iteration in the interval** $\left[\frac{\pi}{8}, \pi\right]$.

Submit homework online before Oct 15,2019

Thanks