Introduction

(Basic definitions, various types of process models and the corresponding type of resulting equations, Computational errors, conditioning and stability of algorithms, General process modeling, Modeling examples of lumped parameter and distributed parameter systems. Non-dimensionalization of model equations, Introduction to Fortran, IMSL and MATLAB programming.)

Introduction

- Introduction to Fortran Programming
- Introduction to Matlab Programming

Introduction to Fortran Programming

Program Structure

PROGRAM *program-name*

IMPLICIT NONE

[specification part]

[execution part]

[subprogram part]

END PROGRAM *program-name*

Note: Fortran 90 was the last major standard in Fortran language development – published in 1991.

Also: Fortran 95 – adopted in 1997 and Fortran 2003 – published in 2004.

Introduction to Fortran Programming

IMPLICIT NONES

- IMPLICIT NONE disables the default typing provisions of (earlier versions of) Fortran.
- All variable must appear in an explicit type declaration.
- Enables lots of errors (typically typographical errors) to be detected at compile time.

Introduction to Fortran Programming

Comments

- Comments are used to improve code-readability.
  - All characters following an exclamation mark !
  - An entire line may be a comment
  - A blank line is also interpreted as a comment line.

Introduction to Fortran Programming

Comments

```
PROGRAM TestComment
  IMPLICIT NONE
  .....
  READ(*,*) Year  ! read in the value of Year
  .....
  ! This is a comment line in the middle of a program .....
  Year = Year + 1  ! add 1 to Year
  .....
END PROGRAM TestComment
```


Introduction to Fortran Programming

Continuation Lines

- In Fortran, each statement must start on a new line.
- If a statement is too long to fit on a line, it must be continued:
- If a line is ended with an **ampersand**, **&**, it will be continued on the next line.

```
A = 174.5 * Year &
+ Count / 100
```

is equivalent to

```
A = 174.5 * Year + Count / 100
```


Introduction to Fortran Programming

Continuation Lines

```
A = 174.5 * Year &
! this is a comment line
+ Count / 100
```

is equivalent to

```
A = 174.5 * Year + Count / 100
```

Introduction to Fortran Programming

Conversion functions:

Function	*Meaning*	*Arg. Type*	*Return Type*
INT(x)	integer part x	REAL	INTEGER
NINT(x)	nearest integer to x	REAL	INTEGER
FLOOR(x)	greatest integer less than or equal to x	REAL	INTEGER
FRACTION(x)	the fractional part of x	REAL	REAL
REAL(x)	convert x to REAL	INTEGER	REAL

Other functions:

Function	*Meaning*	*Arg. Type*	*Return Type*
MAX(x1, x2, ..., xn)	maximum of x1, x2, ... xn	INTEGER	INTEGER
		REAL	REAL
MIN(x1, x2, ..., xn)	minimum of x1, x2, ... xn	INTEGER	INTEGER
		REAL	REAL
MOD(x,y)	remainder x - INT(x/y)*y	INTEGER	INTEGER
		REAL	REAL

Introduction to Fortran Programming

Mathematical functions:

Function	*Meaning*	*Arg. Type*	*Return Type*
ABS(x)	absolute value of x	INTEGER	INTEGER
		REAL	REAL
SQRT(x)	square root of x	REAL	REAL
SIN(x)	sine of x radian	REAL	REAL
COS(x)	cosine of x radian	REAL	REAL
TAN(x)	tangent of x radian	REAL	REAL
ASIN(x)	arc sine of x	REAL	REAL
ACOS(x)	arc cosine of x	REAL	REAL
ATAN(x)	arc tangent of x	REAL	REAL
EXP(x)	exp(x)	REAL	REAL
LOG(x)	natural logarithm of x	REAL	REAL

Introduction to Fortran Programming

Variables

- Integer Variables.
- An integer is always held *exactly* in the computer's memory, and has a relatively limited range:
  - Between $-2*10^9$ and $2*10^9$ for typical 4-byte (32-bit) integers.
  - Attempts to use an integer value larger than the largest possible, or smaller than the smallest possible value (i.e. out of range) results in an *overflow* error condition.

Introduction to Fortran Programming

Variables

- Real Variables.
- A real is stored as a *floating-point number* and is held as an approximation to a fixed number of significant digits and has a large range:
  - A typical 32-bit real has a 24-bit mantissa and an 8-bit exponent. Real numbers are characterised by two quantities: **precision** and **range**.

    Precision is the number of significant digits that can be represented – it is determined by the length of the mantissa – a 24-bit mantissa gives approximately 7 significant decimal digits.

    Range is the difference between the largest and smallest numbers that can be represented – it is determined by the length of the exponent – an 8-bit exponent gives range of real numbers from about 10^{-38} and 10^{38}.

Introduction to Fortran Programming

Variables

- Integer variables are declared as:

  INTEGER :: *first_integer, second_integer*
- Real variables are declared as:

  REAL :: *first_real, second_real*
- To repeat, earlier versions of Fortran allowed implicit type declarations – this is not good practice and should be avoided. The problem can be avoided by requiring the complier to check that all variables are declared before use – include

  IMPLICIT NONE

  as the first statement of every program unit.

Introduction to Fortran Programming

Variables

- Named Constants
  - Used to declare a constant that is used repeatedly within a code, but which should never be changed.
  - Named constants can be used in expressions, but cannot appear on the left hand side of an assignment statement.

```
REAL, PARAMETER :: c = 2.99792458 ! speed of light
REAL, PARAMETER :: pi = 4.0*atan(1.0)
```


Introduction to Fortran Programming

Variables

- Integer and real variables also have a KIND type parameter that means that numeric variables can have different ranges of possible values and different levels of numerical accuracy.
- The KIND parameter is optional and, if absent, the variable is defined to be of *default* kind.

  INTEGER :: *i ! integer variable i of default kind*

  REAL :: *a ! real variable a of default kind*
- The KIND parameter can be used to specify variables of non-default kind.

Introduction to Fortran Programming

Variables

- Integers of non-default kind.

  We can use the intrinsic function `selected_int_kind(r)` to determine the kind parameter for an integer data type that is able to represent all integer values *n* in the range $-10^{r} < n < 10^{r}$.

  For example,

```
INTEGER, PARAMETER :: int_range = SELECTED_INT_KIND (10)
INTEGER (KIND = int_range) :: k, result
```

  k and result are integer variables that can takes values in the range $-10^{10} < n < 10^{10}$, at least.

Introduction to Fortran Programming

Variables

- Reals of default kind.

  All implementations of Fortran 90 offer, at least, two default real kinds – corresponding to single and double precision.

  For example, double precision variables can be declared as follows:

```
INTEGER, PARAMETER :: idp = KIND(1.0d0) ! idp is
the kind value of double precision real numbers
REAL (KIND=idp) :: x
```

  Single precision variables can be declared as follows:

```
INTEGER, PARAMETER :: isp = KIND(1.0) ! isp is
the kind value of single precision real numbers
REAL (KIND=isp) :: x
```


Introduction to Fortran Programming

Variables

- **Logical** variables and constants.
- **Character** variables and constants.
- **Complex** variables and constants.

Introduction to Fortran Programming

Variables

- **REAL: REAL X, Y, I, J**
- **INTEGER: X, Y, I, J**
- **CHARACTER: *80, NAME1, NAME2**
- **DIMENSION: X (10), Y(2, 30)**

Introduction to Fortran Programming

Arithmetic and Assignment

- Assignment Statements

  The most common way that a variable is given a value is by an assignment statement:

  name = expression

  where name is the name of a variable and expression is an arithmetic (or other) expression that will be evaluated and assigned to the variable.

  $a = b + c$

  takes the values currently stored in *b* adds to it the value currently stored in *c* and stores the resulting value in *a*.

Introduction to Fortran Programming

Arithmetic Operators

- Precedence of arithmetic operators

Operator	*Priority*
** (exponentiation)	high
* (multiplication)	medium
/ (division)	
+ (addition)	low
- (subtraction)	

- In the absence of parentheses, evaluation proceeds left to right, except in the case of exponentiation when it proceeds right to left.

Introduction to Fortran Programming

List-directed Input and Output

READ *, *variable_1, variable_2, …..*

PRINT *, *item_1, item_2, …..*

- Note that the list of items in the READ statement may contain only variable names, but the list in a PRINT statement may also contain constants and expressions.
- The READ statement takes its input from the default input stream – usually the keyboard, and the PRINT statement writes to the default output stream – usually the screen.

READ *, *real_var_1, real_var_2, integer_var_1*

- The input stream must contain 2 real values and 1 integer value – an error will result if the third value contains a decimal point.

Introduction to Fortran Programming

List-directed Input and Output

- **PROGRAM: KTEST**
- **OPEN: (1, FILE='INPUT.DAT',**
  STATUS='OLD')
- **READ (1, *) X, Y**
- **WRITE (2, 10) (X(i), i=1,10)**
- **FORMAT (1x, 'I=', 10F5.2)**

Introduction to Fortran Programming

Example Program

- Program to read in a Centigrade temperatue, convert it to Fahrenheit, and output the result.
- Use the formula:

$$F = \frac{9 * C}{5} + 32$$

Introduction to Fortran Programming

Example Program

```
PROGRAM fahrenheit_conversion
      IMPLICIT NONE
      REAL :: temp_f, temp_c
!
      PRINT *, 'Input Centigrade temperature'
      READ *, temp_c
!
!     Calculate Fahrenheit temperature
!
      temp_f = 9.0*temp_c/5.0 + 32.0
!
      PRINT *, temp_c, 'degrees Centigrade =', temp_f, 'degrees Fahrenheit'
!
      STOP
    END PROGRAM fahrenheit_conversion
```


Introduction to Fortran Programming

Logical Variables

- Declaration

  LOGICAL :: *logical_var*

Introduction to Fortran Programming

Logical Variables

Relational Operators

`.lt. or <`	less than
`.le. or <=`	less than or equal to
`.eq. or ==`	equal to
`.ne. or /=`	not equal to
`.gt. or >`	greater than
`.ge. or >=`	greater than or equal to

Introduction to Fortran Programming

Logical Variables

Logical Operators

`.not.`	Logical negation *(unary operator)*
`.and.`	Logical intersection
`.or.`	Logical union
`.eqv.`	Logical equivalence
`.neqv.`	Logical non-equivalence

Introduction to Fortran Programming

Control Constructs

- GOTO statement

```
x = y +3.0
GOTO 4
3 x = x+2.0
4 z = x+y
```

- Generally agreed that GOTO statements make code difficult to understand and their use should be restricted.

Introduction to Fortran Programming

If Statement and Construct

LOGICAL IF statement

if (*scalar_logical _expression*) *action_statement*

```
IF (flag) GOTO 10

IF (x .gt. 0.0) y = 1.0
```


Introduction to Fortran Programming

If Statement

- **IF**

IF (CODE.EQ.1) ND=ND+1

- Only one statement after IF (in the same line)

- No THEN needed

- No END IF needed

Introduction to Fortran Programming

BLOCK IF Construct

IF (*scalar_logical _expression*) THEN
    block_of_statements
END IF
IF (*scalar_logical _expression_1*) THEN
    block_of_statements_1
ELSE IF (*scalar_logical _expression_2*) THEN
    block_of_statements_2
ELSE
    block_of_statements_3
END IF

Introduction to Fortran Programming

If Statement

- **BLOCK IF**

```
IF (CODE.EQ.1) THEN
  ND=ND+1
ELSE  IF (CODE.EQ.2) THEN
  NND=NND+1
ELSE
    NE=NE+1
END IF
```


Introduction to Fortran Programming

Program Repetition (Counting Loop) – the BLOCK DO Construct

DO *control_var = initial, final, increment*

block_of_statements

END DO

- *control_var* is an integer variable;
- *initial, final, increment* are integer expressions.

Introduction to Fortran Programming

DO WHILE Construct

DO

block_of_statements_1

If *logical_expression* EXIT

block_of_statements_2

END DO

- EXIT causes causes control to leave the inner-most DO-loop that contains the EXIT statement.
- Must be a single entry point and a single exit point.

Introduction to Fortran Programming

Nested DO Loops

```
DO j = 1, n
  DO i = j+1, n
    a(i,j) = a(j,i)
  END DO
END DO
```

- Copies the contents of the strictly upper triangular part of *A* into the strictly lower triangular part of *A.*

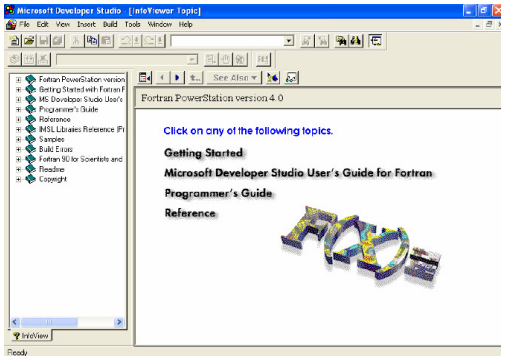
Introduction to Fortran Programming

Execution and Testing

- **Compile to produce object file.**
- **Build to create .exe file.**
- **Execute the .exe file to examine the correctness of the code.**

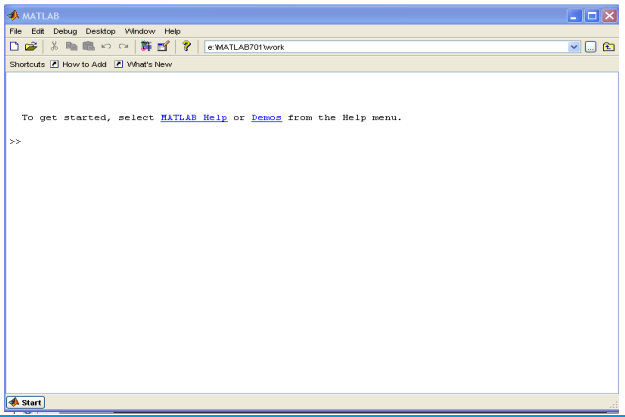
Introduction to Fortran Programming

Fortran PowerStation



Introduction to Matlab Programming

Introduction to Matlab Programming



Introduction to Matlab Programming

- Matlab: numerical development environment.
  - Easy and fast programming
  - data types (vectors, matrices, complex numbers)
  - Complete functionality
  - Powerful toolkits

Introduction to Matlab Programming

MATLAB

- Matlab is a software package that allows you to program the mathematics of an algorithm without getting too bogged down in the details of data structures, pointers, and reinventing the wheel.
- It also includes graphing capabilities, and there are numerous packages available for all kinds of functions, enabling relatively high-level programming.

Introduction to Matlab Programming

MATLAB

- The basic Matlab data structure is a matrix; a scalar is a 1x1 matrix, a vector is an nx1 matrix.
- Vectors and matrices in Matlab are indexed starting from 1; and not from 0; as is more common in modern programming languages. Moreover, the last index of a vector is denoted by the special symbol end.

Introduction to Matlab Programming

Matlab startup

- Start Matlab:
  - open console window
  - type `matlab`
- Online help:
  - `help` keyword (e.g. `help cos`)
  - `doc` keyword
  - `lookfor` keyword
- make & change directory
  - `mkdir 'NMDA'` make new directory
  - `cd 'NMDA'` change directory
  - `pwd` print working directory
  - `ls` list directory

Introduction to Matlab Programming

Matlab basics

- Variables are just assigned (no typedef needed)

```
a=42
s= 'test'
```

- basic operators (+ - * / \ ^)

```
5/2    ans = 2.5000
```

ans is a system variable

- functions (`help elfun`)

```
sqrt(3), sin(pi), cos(0)
```

pi is a system variable

- display & clear variables

```
disp(a), disp('hello')
```

display value of a , "hello"

```
who, whos
```

show all defined variables

```
clear a
```

clear variable a

```
clear
```

clear all variables

- arrow up/down keys recall your last commands

Introduction to Matlab Programming

Matlab examples

- Variables & Operators

```
a=5*(2/3)+3^2                    result is shown
a=2/4 + 4\2        ;             result is not shown
a                                value of a is shown
```

- Elementary functions overview: doc elfun

```
abs(-1), sqrt(2)
tan(0), cos(0), acos(1) …
exp(2), log(1), log10(1) …
```

- Rounding

```
round(2.3), round(2.5)        2      3
```

- `floor(5.7), floor(-1.2)` 5 -2 towards smaller

```
ceil (1.1), ceil(-2.7)        2      -2     towards larger
fix(1.7), fix (-2,7)          1      -2     towards 0
```

- Complex numbers

```
(2+3i) * (1i)                    -3+2i
norm(1+1i)                       1.4142
```


Introduction to Matlab Programming

Matlab functions

Function	Description
sin(x)	sinus of x ; x in radians
cos(x)	cosinus of x ; x in radians
tan(x)	tangent of x ; x in radians
sec(x)	secant of x ; x in radians
csc(x)	cosecant of x; x in radians
cot(x)	cotangent of x ; x in radians
asin(x)	arc sinus of x
sinh(x)	hyperbolic sinus of x
asinh(x)	inverse hyperbolic sinus of x
exp(x)	exponential of x
log(x)	natural logarithm of x
log10(x)	base 10 logarithm of x
sqrt(x)	square root of x
power(x,a)	x to the power of a (equiv to x^4)
abs(x)	absolute value of x
round(x)	round x towards nearest integer
floor(x)	round x towards minus infinity
ceil(x)	Round x towards infinity
fix(x)	Round x towards zero

Introduction to Matlab Programming

Vectors: Matlab's greatest power

- Row and column vectors:

```
[1, 2, 3]  →  (1 2 3)          alternative: [1 2 3]
[1; 2; 3]  →  ( 1                           [1
               2                             2
               3 )                           3]
```

- Initialize a vector: [start : end] or [start : step : end]

```
[1:5]                  (1 2 3 4 5)
[1:3:10]               (1 4  7 10)
[0.5: -0.5 : -0.6]     (0.5  0  -0.5)
```

- linearly and log-spaced vectors with N elements

```
v=linspace(0,1,4)      ( 0  1/3  2/3  1 )
w=logspace(1,4,4)      (10 100 1000 10000)
```

- Set or display an element

```
v(4)                   1                          element  #4
w(3)=99                10 100 99 10000  change element  #3
```

Introduction to Matlab Programming

Working with vectors

- Address elements of vector `v=[1,4,9,16,25]`
  `v([1,3,5])` (1 9 25) elements 1,3,5
  `v([2:4])` (4 9 16) elements 2,3,4
- Set elements of a vector
  `v([1,3,5])=0` (0 4 0 16 0) set elements 1,3,5 to 0
  `v([1,3,5])=[1 2 3]` (1 4 2 16 3) set el. 1,3,5 to [1 2 3]
  `v([1,3,5])=[1 2]` ERROR Vector dim. must agree!
- Find and replace elements `v=[1,4,9,16,25]`
  `p=find(v>8)` (3 4 5) indices of elements >8
  `v(p)` (9 16 25) elements >8
  `v(p)=0` (1 4 0 0 0) set elements >8 to 0
- Simple arithmetics with vectors:
  `[1 2 3] + [4 5 6]` (5 7 9)
  `[1 2 3] + 4` (5 6 7)
  `[1 2 3] * 2` (2 4 6)
- Element-wise operations : `v.*v, 1./w, .^`
  `[1 2].*[3 4]` (3 8)
  `[1 2 3] .^2` (1 4 9)

Introduction to Matlab Programming

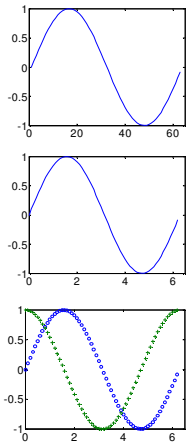
More vector operations

- Transpose vector v: `v'` (complex conjugate transpose)
  `v.'` (normal transpose)
- Inner product: v*v' (check what `v'*v` gives!)
  `[1 2]*[3 4]'` 11
- cross product for 3-dim vectors:
  `cross(v,w)`
- Functions of vectors `v=[1,4,9,16,25]`
  `sqrt(v)` (1 2 3 4 5)
  `sin([0:0.1:2*pi])`
- Sum, product of elements
  `sum(v), prod(v)`
- Euclidian norm of a vector
  `norm(v)` sqrt(sum(abs(v).^2))
- Number of elements
  `length(v)`

Introduction to Matlab Programming

Plotting vectors

- y-plots
  ```
  y=sin([0:0.1:2*pi])
  plot(y)
  ```
- xy-plots
  ```
  x=[0:0.1:2*pi]
  y=sin(x)
  plot (x,y)
  ```
  y could be matrix as well!
- plot formats
  ```
  y1=sin(x)
  y2=cos(x)
  plot (x,y1,'o',x,y2,'+')
  ```
  see help plot
- lin-log and log-log plots
  `semilogx,semilogy,loglog` see help
- Axes, labels
  `axis, xlabel, ylabel` see help



Introduction to Matlab Programming

comparisons and conditional execution

- relational operators
  `<, > , <= , >=` less, greater, less-equal, greater-equal
  `== , ~=` equal, not equal (caution: equal is not =)!
- Logical operators
  `&, | , ~` and, or, not
  `(2>5) & (1>0)` `~(1>1)`
  `(0>0) | (1>=1)` `~(10<=9) & ~(1>0)`
- simple if: **`if`** (condition); … ; **`end`**
  ```
  if (rem(x,3)==0)
    disp('number is divisible by 3')
  end
  ```
  rem gives remainder of a division
- if-else: **`if`** (condition); … ; **`else`** … ; **`end`**
  ```
  if (rem(x,2)==0)
      disp('number is even')
  else
      disp('number is odd')
  end
  ```

Introduction to Matlab Programming

comparisons operators

Operator	Description
$\sim a$	NOT: returns 1 if a = 0 and 1 if a = 0
a==b	returns 1 if a equals b (0 otherwise)
a < b	returns 1 if a is strictly smaller than b
a > b	returns 1 if a is strictly greater than b
a <= b	returns 1 if a is smaller or equal to b
a >= b	returns 1 if a is greater or equal to b
a ~= b	returns 1 if a is different from b
&	AND operator (equiv. to and(bool1,bool2)
|	OR operator (equiv. to or(bool1,bool2)
xor	Exclusive OR operaor

Introduction to Matlab Programming

flow control: loops

- for loops: perform a loop for every element of a vector

  for variable=vector; … ; **end**

```
fib([1 2])=1                    calculate the first 10 Fibonacci numbers
for i=3:10
   fib(i)=fib(i-1)+fib(i-2)
end
```

- while loops: perform a loop while a condition is "true"

  while (condition); … ; **end**

```
fib([1 2])=1                    calculate the first 10 Fibonacci numbers
i=3
while (i<=10)
   fib(i)=fib(i-1)+fib(i-2)
   i=i+1
end
```


Introduction to Matlab Programming

Control Structures in Matlab

- %compute the sign of x
- if x > 0
- s = 1;
- elseif x == 0
- s = 0;
- else
- s = -1;
- end

- while (sin(x) > 0)
- x = x * pi;
- end

- %a `regular' for loop
- for i=1:10
- sm = sm + i;
- end

- %an `irregular' for loop
- for i=[1 2 3 5 8 13 21 34]
- fsm = fsm + i;
- end

Introduction to Matlab Programming

Matrices: the general case of vectors

- Initialize a matrix:

```
[1,2,3; 4,5,6; 7,8,9]
```
gives
```
1 2 3
4 5 6
7 8 9
```

- we can play all the vector tricks

```
M=[1:3:9; 11:3:19; 21:3:29]
```
gives
```
 1  4  7
11 14 17
21 24 27
```

- Access matrix elements: `M(row,col)`

```
M(2,3)      17                  access single element
M(1,:)       1  4  7            access row vector
M(:,2)       4                  access column vector
            14
            24

M(2:3,1:2)
```

- special matrices

```
ones(2)     ones(2,3)     eye(2)
1 1         1 1 1          1 0
1 1         1 1 1          0 1
```

- find,replace matrix elements matching an expression

```
find (M>15)          M(find(M>15))=-1
```

Introduction to Matlab Programming

Matrix operations

- Matrix product: `A*B` — number of columns in A must be number of rows in B
- element wise product: `A.*B`
- left/right division:
  `A/B` = $((B')^{-1}*A')'$
  `A\B` = $A^{-1}*B$ — calculation is done without calculating A^{-1} => more efficient
- inverse matrix: `inv(A)`
- determinant: `det(A)`
- Eigenvalues: `eig(A)` — A should be square
- create a diagonal matrix: `A=diag(v)`
- size of a matrix: `size(A)`

Introduction to Matlab Programming

Matrix operations

Function	Description
det(A)	determinant of matrix A
trace(A)	trace of matrix A
rank(A)	rank of the matrix
norm(A)	norm of the matrix (can be applied to vector)
inv(A)	inverse of matrix A
poly(A)	Characteristic polynomial of the matrix A
eig(A)	eigenvalues/eigenvectors of A
svd(A)	singular value decomposition of A
min(V)	minimum value in vector V
max(V)	maximum value in vector V
sum(V)	sum of the elements of vector V

Introduction to Matlab Programming

Input/Output
printing text and variables

- to output data (to screen or to a file) we need a way to control the format of numbers
- fprintf (format-string,var1,var2,var3…) does just that
- the format string defines how variables are printed.

Example 1:

```
a=1; b=2.65;c=12345;d='test'
fprintf ('%d %f %e %s \n',a,b,c,d)
```

1 2.650000 1.234500e+04 test

Example 2:

```
fprintf ('a:%2d b:%1.2f c:%1.4e \n',a,b,c)
```

a: 1 b:2.65 c:1.2345e+04

Introduction to Matlab Programming

saving & loading data

- variables can be saved to binary and text files
- save and load the complete workspace (binary)
  `save filename, load filename`
- save & load variables to/from file ' test.mat'
  `save 'test' a b c; load 'test'`
- reading a formatted data file
  `[A,B,C,...] = textread('filename','format')`

  example file: `[A,B,C]=textread('test.dat', '%f %f %f')`

```
A =
    1.2000
    5.5000
    1.0000
```

- general data input/output
  `f=fopen('filename')` open file
  `fprintf (f,format, v1,…)` formatted output to file
  `fscanf (f,format, v1,…)` formatted reading from file
  `fclose(f)` close file

Introduction to Matlab Programming

Programming in Matlab

- If you are going to do any serious programming in Matlab, you should keep your commands in a file. Matlab loads commands from `.m' files. If you have the following in a file called myfunc.m:

```
function [y1,y2] = myfunc(x1,x2)
% comments start with a '%'
% this function is useless, except as an example of functions.
% input:
% x1 a number
% x2 another number
% output:
% y1 some output
% y2 some outpu
y1 = cos(x1) .* sin(x2);
y2 = norm(y1);
```


Introduction to Matlab Programming

Programming in Matlab

- then you can call this function from Matlab, as follows:

```
> myfunc(2,3)
ans = -0.058727
> [a,b] = myfunc(2,3)
a = -0.058727
b = 0.058727
> [a,b] = myfunc([1 2 3 4],[1 2 3 4])
a =
0.45465 -0.37840 -0.13971 0.49468
b = 0.78366
```


Introduction to Matlab Programming

m-files

- Matlab commands can be put in a textfile with ending .m
- .m files can be edited with the Matlab Java editor or any other text editor (e.g. `xedit`)
- if the file example.m is in the current directory (`pwd, cd`) or in the load path, it can be called by typing `example`
- Modifying the load path: `path, addpath`
- Modular programming: function .m-files

Introduction to Matlab Programming

function definitions

- A function is a name for a particular calculation or subroutine. It can be called be that name, can take arguments and return a value
- It must be put in a .m-file with same name as function
- Define function:
  `function` returnvariable=name(arguments) … **`end`**
  in separate file square.m:

```
function ret=square(n)
% calculate n^2
   ret=n*n;
end
```

- `square(3)` ans=9
  `help (square)`

Introduction to Matlab Programming

Example: print number in binary system

file binary.m:

```
function b=binary(num)
% return num in binary format as string
   b='';
   x=num;
   while (x>0) | (strcmp(b,''))
      if rem(x,2)==0 b=['0' b];
        else b=['1' b];
       end
       x=fix(x/2);
   end
end
```

file example.m

```
  n=input('Please enter a number:');
  disp(binary(n));
```


Introduction to Matlab Programming

Exercise now: convert binary to decimal

- Now, write a function which takes a binary number in string bin and converts it to a number
- hints:
  - you can use a for or a while loop
  - access i^{th} character of a string s with s(i):
    `s='test';`
    `s(3)` 's'
- **For cracks**: Do the binary-to-number conversion in one line, without loop!
  hint: Use all the vector tricks,
  Use `s-48`

Introduction to Matlab Programming

Solution

```
function r=bintonum(s)
% takes binary number s as string,returns integer
n=0;
i=length(s)-1;
for (c=s)
    n=n+str2num(c)*2^i;
    i=i-1;
end
r=n
end
```

One-liner: (much faster!!!)

```
r=sum((s-48).*2.^[length(s)-1:-1:0])
```