Computational Physics

David.Lary@utdallas.edu
David Lary

Some Details

- Homework each week due on Wednesday
- Books:
  - Numerical Recipes, third edition

Languages

- We will use three high level languages:
  - Matlab "**mat**rix **lab**oratory"
  - **Python** after Monty Python
  - **R** statistical language

Why people use Matlab (or Python or R)

- Programmer productivity. Why write several thousand lines of C++ code when you can solve a problem with a short Matlab or Python script.

- Toolboxes. Matlab has a large range of toolboxes and Python a large range of plugins. Advanced programs can quickly be assembled from existing elements.

Why people use Matlab or Python

- Performance. Matlab and Python can easily call out to C or Fortran code for performance of critical elements.
- Fun. Programmers like using Matlab or Python.

Matlab Matrices

- Separate the elements of a row with blanks or commas.
- Use a semicolon, ; , to indicate the end of each row.
- Surround the entire list of elements with square brackets, [ ].
- `A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]`

sum, transpose, and diag

- **Sum:** `sum(A)`
- **Transpose:** `A'`
- **Diagonal:** `diag(A)`
- **Sum of the diagonal:** `sum(diag(A))`

sum transpose and diag

Subscripts

- The element in row i and column j of A is denoted by `A(i,j)`.
- It is also possible to refer to the elements of a matrix with a single subscript, `A(k)`. So, for A in the earlier example, `A(8)` is another way of referring to the value 15 stored in `A(4,2)`.

The Colon Operator

- The colon, :, is one of the most important MATLAB operators. It occurs in several different forms. Try the following:
  - `1:10`
  - `100:-7:50`
  - `0:pi/4:pi`
  - `sum(A(:,end))`

Variables

- MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

```
num_students = 25
```

- creates a 1-by-1 matrix named num_students and stores the value 25 in its single element. To view the matrix assigned to any variable, simply enter the variable name.

Variables

- Although variable names can be of any length, MATLAB uses only the first N characters of the name, (where N is the number returned by the function namelengthmax), and ignores the rest. Hence, it is important to make each variable name unique in the first N characters to enable MATLAB to distinguish variables.

```
N = namelengthmax

N =

    63
```

- The `genvarname` function can be useful in creating variable names that are both valid and unique.

Variables

A MATLAB variable is essentially a tag that you assign to a value while that value remains in memory. The tag gives you a way to reference the value in memory so that your programs can read it, operate on it with other data, and save it back to memory.

MATLAB provides three basic types of variables:

- Local Variables
- Global Variables
- Persistent Variables

Numbers

- MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter e to specify a power-of-ten scale factor. Imaginary numbers use either i or j as a suffix. Some examples of legal numbers are:

```
3              -99            0.0001
9.6397238      1.60210e-20    6.02252e23
1i             -3.14159j      3e5i
```

Operators

Expressions use familiar arithmetic operators and precedence rules.

+ Addition

- Subtraction

* Multiplication

/ Division

\ Left division (see in Linear Algebra in MATLAB documentation)

^ Power

' Complex conjugate transpose

( ) Specify evaluation order

Functions

MATLAB provides a large number of standard elementary mathematical functions, including `abs`, `sqrt`, `exp`, and `sin`. Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically. MATLAB also provides many more advanced mathematical functions, including Bessel and gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type

```
help elfun
```

Functions

For a list of more advanced mathematical and matrix functions, type

```
help specfun

help elmat
```

Functions

Several special functions provide values of useful constants.

```
pi      3.14159265...
i       Imaginary unit
j       Same as i
eps     Floating-point relative precision,
realmin Smallest floating-point number,
realmax Largest floating-point number,
Inf     Infinity
NaN     Not-a-number
```

Trig Functions

When angle is in radians use:

```
sin, cos, tan, sinh, cosh,
tanh, asin, acos, atan,
asinh, acosh, atanh
```

When angle is in degrees use:

```
sind, cosd, tand, asind,
acosd, atand
```

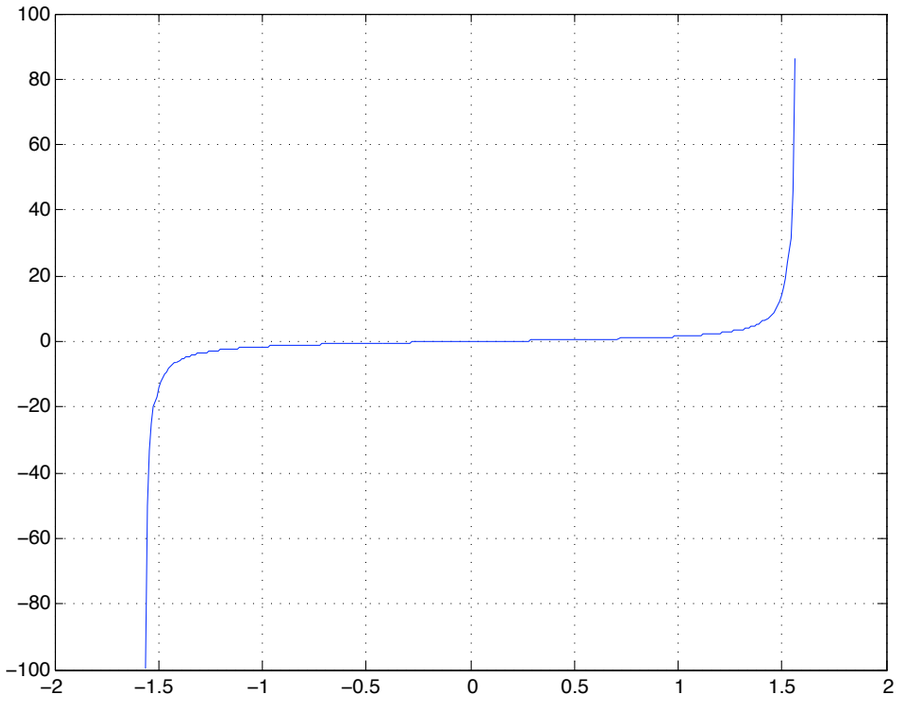
Trig function example

Graph the tangent function over the domain

$$-\frac{\pi}{2} < x < \frac{\pi}{2}$$

```
x = (-pi/2)+0.01:0.01:(pi/2)-0.01;

plot(x,tan(x)), grid on
```



Comments

- You can make any line in an M-file a comment by typing % at the beginning of the line. To put a comment within a line, type % followed by the comment text; MATLAB software treats all the information after the % on a line as a comment.
- To comment a contiguous group of lines, type %{ before the first line and %} after the last line you want to comment. This is referred to as a block comment. The lines that contain %{ and %} can contain spaces, but not contain any other text. After typing the opening block comment symbol, %{, all subsequent lines assume the syntax highlighting color for comments until you type the closing block comment symbol, %}. Remove the block comment symbols, %{ and %}, to uncomment the lines.

Creating arrays

`zeros` create array of all zeros

Syntax

```
Y = zeros(n)
Y = zeros(m,n)
Y = zeros([m n])
Y = zeros(m,n,p,...)
Y = zeros([m n p ...])
Y = zeros(size(A))
zeros(m, n,...,classname)
zeros([m,n,...],classname)
```

classname can have the following values: `'double', 'single', 'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'int64', or 'uint64'`

Create an array and set all elements to Nan

```
x=zeros(5,10)

iwant=find(x==0)

x(iwant)=NaN
```

Creating arrays

`ones` **create array of all ones**

Syntax

```
Y = ones(n)
Y = ones(m,n)
Y = ones([m n])
Y = ones(m,n,p,...)
Y = ones([m n p ...])
Y = ones(size(A))
ones(m, n,...,classname)
ones([m,n,...],classname)
```

classname can have the following values: `'double', 'single', 'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'int64', or 'uint64'`

Creating arrays

Other ways of creating ways include:

```
doc eye

doc rand

  Uniformly distributed random
  elements

doc randn

  Normally distributed random
  elements

doc complex
```

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.
accumarray	Distribute elements of an input matrix to specified locations in an output matrix, also allowing for accumulation.
diag	Create a diagonal matrix from a vector.
magic	Create a square matrix with rows, columns, and diagonals that add up to the same number.
rand	Create a matrix or array of uniformly distributed random numbers.
randn	Create a matrix or array of normally distributed random numbers and arrays.
randperm	Create a vector (1-by-n matrix) containing a random permutation of the specified integers.

Function	Description
cat	Concatenate matrices along the specified dimension
horzcat	Horizontally concatenate matrices
vertcat	Vertically concatenate matrices
repmat	Create a new matrix by replicating and tiling existing matrices
blkdiag	Create a block diagonal matrix from existing matrices

Concatenation

- Concatenation is the process of joining small matrices to make bigger ones. In fact, you made your first matrix by concatenating its individual elements. The pair of square brackets, `[]`, is the concatenation operator. For an example, start with the 4-by-4 magic square, `A`, and form

```
B = [A  A+32; A+48  A+16]
```

Getting Information on a Matrix

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.

Function	Description
isa	Detect if input is of a given data type.
iscell	Determine if input is a cell array.
iscellstr	Determine if input is a cell array of strings.
ischar	Determine if input is a character array.
isfloat	Determine if input is a floating-point array.
isinteger	Determine if input is an integer array.
islogical	Determine if input is a logical array.
isnumeric	Determine if input is a numeric array.
isreal	Determine if input is an array of real numbers.
isstruct	Determine if input is a MATLAB structure array.

Function	Description
isempty	Determine if input has any dimension with size zero.
isscalar	Determine if input is a 1-by-1 matrix.
issparse	Determine if input is a sparse matrix.
isvector	Determine if input is a 1-by-n or n-by-1 matrix.

Deleting Matrix Elements

You can delete rows and columns from a matrix by assigning the empty array `[]` to those rows or columns. Start with

```
A = magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Then, delete the second column of `A` using

```
A(:, 2) = []
```

This changes matrix `A` to

```
A =

    16     3    13
     5    10     8
     9     6    12
     4    15     1
```

Reshaping a Matrix

Function	Description
reshape	Modify the shape of a matrix.
rot90	Rotate the matrix by 90 degrees.
fliplr	Flip the matrix about a vertical axis.
flipud	Flip the matrix about a horizontal axis.
flipdim	Flip the matrix along the specified direction.
transpose	Flip a matrix about its main diagonal, turning row vectors into column vectors and vice versa.
ctranspose	Transpose a matrix and replace each element with its complex conjugate.

Pre-allocating Memory

Repeatedly expanding the size of an array over time, (for example, adding more elements to it each time through a programming loop), can adversely affect the performance of your program. This is because:

- MATLAB has to spend time allocating more memory each time you increase the size of the array.
- This newly allocated memory is likely to be noncontiguous, thus slowing down any operations that MATLAB needs to perform on the array.

The preferred method for sizing an array that is expected to grow over time is to estimate the maximum possible size for the array, and preallocate this amount of memory for it at the time the array is created. In this way, your program performs one memory allocation that reserves one contiguous block

Shifting and Sorting Matrices

Function	Description
circshift	Circularly shift matrix contents.
sort	Sort array elements in ascending or descending order.
sortrows	Sort rows in ascending order.
issorted	Determine if matrix elements are in sorted order.

Syntax

```
B = sort(A)
B = sort(A,dim)
B = sort(...,mode)
[B,IX] = sort(A,...)
```

Description

`B = sort(A)` sorts the elements along different dimensions of an array, and arranges those elements in ascending order.

If A is a ...	sort(A) ...
Vector	Sorts the elements of `A`.
Matrix	Sorts each column of `A`.
Multidimensional array	Sorts `A` along the first non-singleton dimension, and returns an array of sorted vectors.
Cell array of strings	Sorts the strings in ASCII dictionary order.

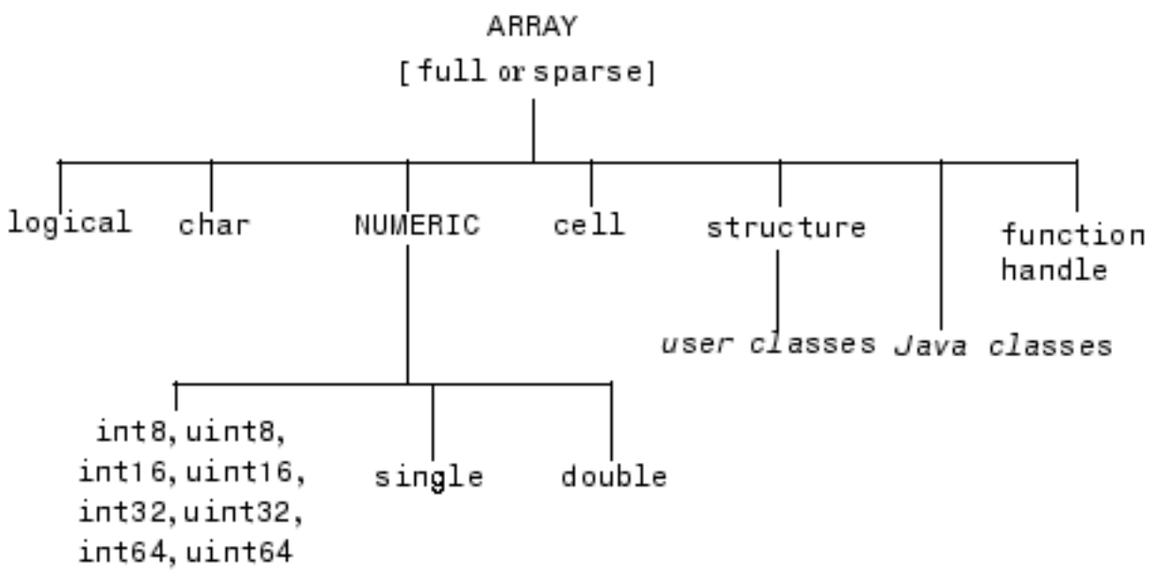
Classes

Fundamental Classes

There are many different classes of data that you can work with in the MATLAB software. You can build matrices and arrays of floating-point and integer data, characters and strings, logical `true` and `false` states, etc. Two of the MATLAB classes, structures and cell arrays, provide a way to store dissimilar types of data in the same array. You can also develop your own classes.

There are 15 fundamental classes in MATLAB. Each of these classes is in the form of a matrix or array. This matrix or array is a minimum of 0-by-0 in size and can grow to an n-dimensional array of any size.

All of the fundamental classes are shown in lowercase, plain nonitalic text in the diagram below.



If

```
if expression1
    statements1
elseif expression2
    statements2
else
    statements3
end
```

Switch and Case

```
switch switch_expr
 case case_expr
    statement, ..., statement
    case {case_expr1, case_expr2, ...}
      statement, ..., statement
    otherwise
      statement, ..., statement
end
```

for

The for loop executes a statement or group of statements a predetermined number of times. Its syntax is

```
for index = start:increment:end

    statements

end
```

The default increment is 1. You can specify any increment, including a negative one. For positive indices, execution terminates when the value of the index exceeds the end value; for negative increments, it terminates when the index is less than the end value.

Vectorizing loops

The MATLAB software uses a matrix language, which means it is designed for vector and matrix operations. You can often speed up your M-file code by using vectorizing algorithms that take advantage of this design. Vectorization means converting for and while loops to equivalent vector or matrix operations.

Simple Example of Vectorizing

Here is one way to compute the sine of 1001 values ranging from 0 to 10:

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

A vectorized version of the same code is

```
t = 0:.01:10;
y = sin(t);
```

The second example executes much faster than the first and is the way MATLAB is meant to be used.

while

The while loop executes a statement or group of statements repeatedly as long as the controlling expression is true (1). Its syntax is

```
while expression
    statements
end
```

If the expression evaluates to a matrix, all its elements must be 1 for execution to continue. To reduce a matrix to a scalar value, use the all and any functions.

For example, this while loop finds the first integer n for which n! (n factorial) is a 100-digit number.

```
n = 1;
while prod(1:n) < 1e100
    n = n + 1;
end
```

Exit a while loop at any time using the `break` statement.

continue and break

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

In for loops, the loop counter is incremented by the appropriate value (either 1 or the specified step value) at the start of the next iteration. continue works the same way in nested loops. That is, execution continues at the beginning of the loop in which the continue statement was encountered.

The `break` statement terminates the execution of a `for` loop or `while` loop. When a `break` statement is encountered, execution continues with the next statement outside of the loop. In nested loops, `break` exits from the innermost loop only.

try, catch

Error control statements provide a way for you to take certain actions in the event of an error. Use the `try` statement to test whether a certain command in your code generates an error. If an error does occur within the `try` block, MATLAB immediately jumps to the corresponding `catch` block. The `catch` part of the statement needs to respond in some way to the error.

The general form of a `try-catch` statement sequence is

```
try
    statement
    ...
catch
    statement
    ...
end
```

return

After a MATLAB function runs to completion, it terminates and returns control either to the function that called it, or to the keyboard.

If you need to exit a function prior to the point of normal completion, you can force an early termination using the `return` function. `return` immediately terminates the current sequence of commands and exits the currently running function.

Roundoff Error

A round-off error (rounding error), is the difference between the calculated approximation of a number and its exact mathematical value.

Numerical analysis specifically tries to estimate this error when using approximation equations and/or algorithms, especially when using finite digits to represent infinite digits of real numbers. This is a form of quantization error.

Roundoff Error

Example

Notation	Represent	Approximate	Error
1/7	$0.\overline{142\ 857}$	0.142 857	$0.000\ 000\ \overline{142\ 857}$
ln 2	0.693 147 180 559 945 309 41...	0.693 147	0.000 000 180 559 945 309 41...
$\log_{10} 2$	0.301 029 995 663 981 195 21...	0.3010	0.000 029 995 663 981 195 21...
$\sqrt[3]{2}$	1.259 921 049 894 873 164 76...	1.25992	0.000 001 049 894 873 164 76...
$\sqrt{2}$	1.414 213 562 373 095 048 80...	1.41421	0.000 003 562 373 095 048 80...
e	2.718 281 828 459 045 235 36...	2.718 281 828 459 045	0.000 000 000 000 000 235 36...
π	3.141 592 653 589 793 238 46...	3.141 592 653 589 793	0.000 000 000 000 000 238 46...

There are at least two ways of performing the termination at the limited digit place:

- **truncation: simply chop off the remaining digits.**
  $0.\overline{142857} \approx 0.142$ (dropping all significant digits after 3rd)
- **rounding**: add 5 to the next digit and then chop it. The result may **round up** or **round down**.
  $0.\overline{142857} \approx 0.143$ (rounding the 4th significant digit. This is rounded up because $8 \geq 5$)
  $0.\overline{142857} \approx 0.14$ (rounding the 3rd significant digit. This is rounded down because $2 < 5$)

Reading Assignment Numerical Recipes Chapter 1