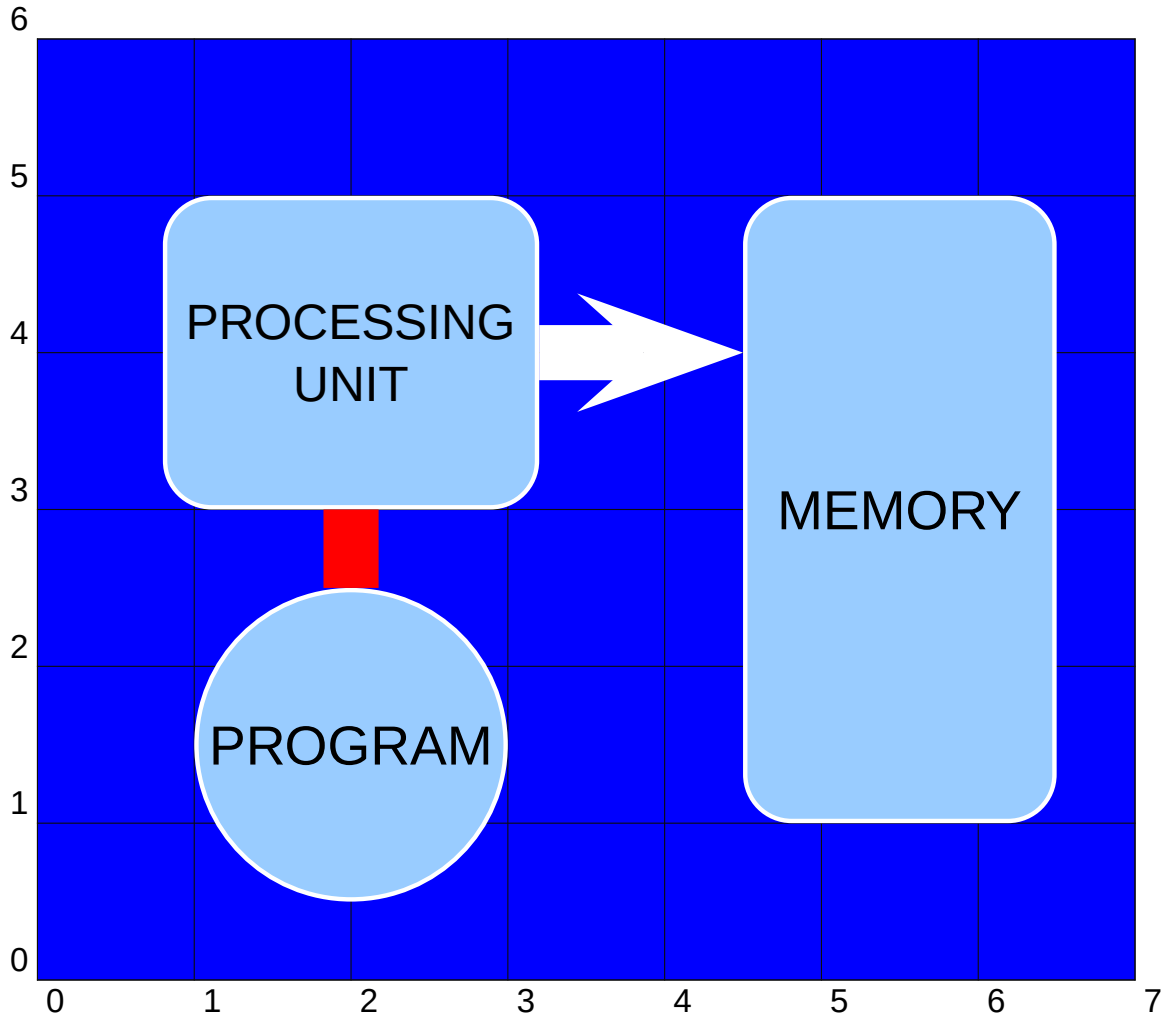
MATLAB 101

Matrix fun

Christoph Best

March 7, 2005

Concepts: Computers

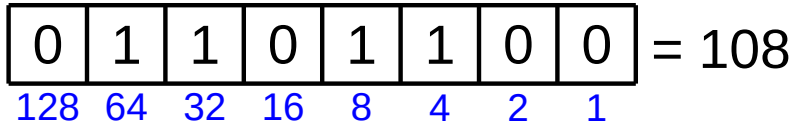


- Memory stores data (=numbers)
- Processing unit modifies memory according to a program

Concepts: Data

All data are numbers.

One byte = Eight binary digits = a number between 0 and 255



Numbers can represent different things:

- Negative numbers: -128 . . . +127
- Larger numbers:
  - 2 bytes = 16 bits = 0 . . . 65536 or -32768 . . . 32767
  - 4 bytes = 32 bits = 0 . . . 4.294.967.296
- Floating-point numbers: 1023×10^{16}
  single-precision: 4 bytes, double-precision: 8 bytes
- Characters: 'A' = 65, 'B' = 66, . . . (ASCII)
- Addresses in memory

Concepts: variables

- Variables name memory locations
- Workspace = set of all variables visible
- Declaration: associate a memory location with a name
- Definition: associate a value with a variable
- Assignment: replace the value in a variable by a new value
- Reference: use the value of a variable in an expression

Concepts: Programs

- Programs consist of statements (≈ lines)
- Statements are executed in sequences
- Assignment statement:
  - left-hand side: a variable to be assigned
  - right-hand side: an expression to be evaluated
- Expression:
  - Mathematical prescription to calculate a value
  - Can contain function calls

Concepts: Control structures

- Not all statements have to be executed in sequence
- Control structures:
  - Conditional: execute only if some expression has a certain value (equals/greater/less)
  - Loop: execute repeatedly until a condition is met

Concepts: Functions

- Functions (subroutines)
  - Sequence of statements
  - Parameters
  - Return value
- How a function is called
  - Create a workspace for local variables
  - Evaluate the parameters of the function call
  - Assign the parameters to local variables
  - Evaluate the statements of the function
  - Find the return value and use it as the value of the function call

Matrices

Everything is a matrix (even vectors)

- Indexing: `a(1)`, `a(1,2)` one-based
  allocation/resizing is automatic
- Construction: `[1 3 5]`, `[1 2 3;4 5 6;7 8 9]`
  `zeros(`*n*`,`*m*`)` `diag(`*n*`,`*m*`)`
  Empty matrix: `[]` Vector: $1 \times n$ or $n \times 1$
- Vectors: `1:10` `0:1:0.1`
- Information: `size(a)` `ndims(a)` `nelem(a)`
- Slices: `a(1,:)` `a(5:10)` `a(5:10,5:10)`
- Concatenation: `[a b]` `[a; b]` `cat(3,a,b)`

Indexing matrices

- Matrices are rectangular, `size(a)` gives their shape
- Storage: column-wise (FORTRAN ordering)

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \longrightarrow [147258369]$$

  Linear indexing: $l = i_1 + i_2 n_1 + i_3 n_1 n_2$
- Indices can be vectors: `a(1,1:3)` `a(2,ix)`
- Linear indexing: flatten dimensions
- Logical indexing/masking
- Assignments

Operators

- Standard math operations: `+`, `-`, `*`, `/`, `^`
- Vector operations: `+`, `-`, `.*`, `./`
  Most operations operate element-wise: `sin(x)`
- Matrix multiplication: a `*` b
  Transpose: a`'`
- Relational: `<`, `<=` `>`, `>=`, `==`, `=`
  also operator elementwise: `[1 2]==[1 3]` → `[1 0]`
- Logical: `~`, `&`, `|`, `&&`, `||`

Functions

- Notation: `z = f(x,y)`
  Multiple returns: `[z1,z2] = f(x,y)`
- Math: `sin(x)`, `exp(x)`, `abs(x)`, `sqrt(x)` ...
- Conversions: `double(i)`, `int16(i)`, `char(i)`
  `floor(x)`, `ceil(x)`, `fix(x)`, `round(x)`
- Tests: `isfloat(x)`, `isinteger(x)`, `isempty(x)`, ...
- Constants: `pi`, `NaN`, `inf`, `eps`, `i`, `j`
- Matrices: `norm(M)`, `trace(M)`, `inv(M)`
- Data analysis: `min`, `max`, `mean`, `median`, `sum`, `cumsum`, `sort`
- Eigensystems:
  `D=eig(M)` or `[V,D] = eig(M)`

Data import and export

- Import wizard: interactive
- `save` and `load` for MAT-files
- `load` also understands simple ASCII files
- `imread` and `imwrite` for images
- `textscan` for tables
- `xmlread`, `xmlwrite` for XML
- `xlsread`, `xlswrite` for spreadsheets
- `fread`, `fwrite` for low-level IO
- and of course: `tom_emread`, `tom_embrowse`

Data types

- Basic types
  - integers: `int8`, `uint8`, `int16`, `...`, `uint32`
  - floating-point: `single`, `double`
  - `logical` (boolean)
  - `char` (strings are vectors of `char`)
- Complex types:
  - structures
  - cell arrays
  - function handles

and, of course, everything is a matrix

Characters and strings

- Strings: `'The big brown fox'` are vectors of characters
- Concatenation: `['The ' 'big ' 'brown ' 'fox ']`
  just as for vectors
- Matrices: `['The ';'big ';'brown ';'fox ']`
  must have same length → cell arrays
  or create them with `char('The','big','brown','fox')`
- Comparison: `strcmp(a,b)` 0 signalizes inequality!
- Conversion: `str2num`, `num2str`
- Conversion: `char()`

Structures

- hashes in Perl, dictionaries in Python, records in Pascal
- Used to store heterogeneous data by text labels
- Implicit creation: `a.x = 1`
- Explicit creation: `struct('x',1)`
- Can be nested
- Very important for GUI programming
- Structures have dimensions too: `a(1,1).x`

Cell arrays

- Lists: Ordered sequences of heterogeneous objects
- Represented as matrices of *cells* (object handles, pointers, . . .)
- Explicit creation: `a = { 'a', 1, 'c' }`
  `a(1) = {'a'}; a(2) = {1}; a(3) = {'c'}`
- `a(1)` is a cell
- `a{1}` is the content of a cell
- Cell arrays can be used instead of comma separated lists:
  `A{1:3}` is equivalent to typing `A{1}, A{2}, A{3}`
  Can be used in vectors, function calls

Plotting data

- Data are plotted into figures and axes
- 2D plots:

```
plot(X,Y)
plot(X,sin(X),X,cos(X))
plot([X' X'],[sin(X)' cos(X)'])
```

- More commands: `axis`, `xlabel`, `ylabel`, `title`, `grid`, `...`
- Current figure (`gcf`, `figure`) and current axes (`axes`, `gca`)
- Figure and axes handles: `h = figure()`
- Properties:

```
get(gca(),'LineWidth')
get(gca(),'LineWidth',2)
```

can also be passed at the end:

```
plot(X,Y,'LineWidth',2)
```

Images

- Images are matrices
  - 2D: values are indices into the current color map
    CDataMapping property: direct, scaled
  - 3D: RGB values between 0 and 1
- Colormaps
- `image(mat)`
- `imagesc(mat)` automatically scales
- `tom_imagesc(em)`

Fourier transforms

- one-dimensional discrette fast Fourier transform (FFT):
  - `fft(a)`, inverse `ifft(a)`
  - operates column-wise on matrices
- two-dimensional: `fft2(M)`, inverse `ifft2(M)`
- n-dimensional: `fftn(M)`, inverse `ifftn(M)`
- `fftshift` to shift frequence 0 to center of plot

Functions and M-Files

- Used-defined functions are stored in M-files
- One function per file, file named like the function (no symbol table, no modules imports)
- Files are looked for in the search path
- Syntax:

```
function res = name(arg1,arg2,...)
% FUNCTION_NAME description
% more documentation
% ...
statements
res = ...
```

Argument passing

- Arguments are passed by value
  even structures and cell arrays
  luckily its copy-on-demand
- Several return values:
  `function [res1,res2] = f(x,y,z)`
  `[a,b] = f(1,2,3)`
- Variable number of arguments:
  `function res = f(x,y,varargin)`
  `varargin{1}`, cell array `nargin`, number of arguments
- Variable number of outputs:
  `function varargout = f(x,y)`
  `varargout{1}`, cell array `nargout`, number of desired output arguments

Types of functions

- built-in functions
- primary (M-file) functions
- subfunctions: only visible in local M-file
- nested functions
  - declared inside another function (required `end`)
  - statically scoped variables
- anonymous functions:

  `sqr = @(x) x.^2`

  `sqr(5)` $\rightarrow$ 25

  can be passed like any other value

  also:`@(x,y) x^y`, `@() datestr(now)`

Scoping

- Variables in scripts have global scope
- All variables in functions are local
- If you want to modify a value, you must explicitly return it
  `a = modify(a,'how')`
- Variables are created when first used, destroyed when scope (function) is exited
- Exception: variables declared global
  `global a`
  or persistent (keep value, but local scope)
  `persistent a`
  Declare before first use!

Control structures

No concept of blocks
always use `end`

- Conditional:

  `if` *expr*

  statements

  `else`

  statements

  `end`

  in one line: `if` *expr*`;` *statements*`;` `end`

- Loop:

  `for i=`*vector*

  statements

  `end`

  Usually something like `for i=1:10`

- `break` and `continue`

- While loop: `while` *expr*`;` *statements*`;` `end`