Rose's Programming in MATLAB

Rose Enos

2025

Adapted from the following sources:

- Notes by Professor Jiawen Hou at the University of California, Irvine for MATH 9

Contents

1 Program Structures

Matrix Laboratory, or **MATLAB**, is a Python-like functional scripting language for engineering, mathematics, and data science. Interactive mode and script mode are like the respective Python modes. Live script mode is like a Jupyter notebook, and interactive elements can be placed into block scripts.

Arguments can be placed in parentheses (Python-like) or after a space (no quotes for strings). The return value is saved to `ans` by default.

`;` suppresses shell output. `disp(val)` returns the value of `val`. `%` makes an inline comment. `error(val)` displays an error message. `warning(val)` displays a warning message.

Array unpacking is done

```
[var1, ..., varn] = func
```

where `~` in place of `vari` discards the `i`th element.

`help(func)` returns documentation for `func`. `lookfor(keyword)` returns commands containing `keyword`. `clc` clears the shell.

`clear([var])` deletes all variables or `var`. `save([file])` saves all variables to `matlab.mat` or `file.mat`. `load(file)` loads variables from `file.mat`. `whos` returns all variables and their details.

`open(file)` opens a file. `edit(file)` opens a file for editing.

A conditional statement is done

```
if condition
    body
elseif condition
    body
else
    body
end
```

A switch statement is done

```
switch val
    case case1
        body
    case case2
        body
    otherwise
        body
end
```

A logical switch can be done by setting `val=true` and each `casei` a logical statement. Switch comparison works for strings.

An anonymous function is created

```
@(params) retval
```

A function is created at the top of the `func.m` file:

```
function [ret1,...,retn] = func(params)
%FUNC shortdescription
% More detailed description with params, return value, behavior, etc.
body
ret1 = ...
...
retn = ...
end
```

The comments are returned by `help func`. Then `func` calls the function, and `@func` returns the `function_handle`. A local (non-exported) function is created similarly at the bottom of a script. By default, arguments are deep copies.

A while loop is done

```
while condition
    body
end
```

A for loop is done

```
for elmt = rowvector
    body
end
```

`val` is in the outer scope and an inner-scoped copy is made for each iteration. This enhanced for loop is slower than a Python-style classical for loop. Control is managed with `break`, `continue`, and `return`.

2 Data Structures

The primitive variables are the matrix types `logical`, `string`, `char`, `{,u}int{8,16,32,64}`, `single`, `double`, `table`, `cell`, and `struct`, and the scalar type `function_handle`. `class(val)` returns the type of `val` as a `char`. Casting is done `newtype(val)`.

All numeric values are `double` by default. The `logical` values are `true` and `false`. Numeric values support `e` for scientific notation and `i` for the imaginary number. `format rat` makes all later results display as rational numbers, if applicable.

A variable is created

```
[ a11, ..., a1n;
  ..., ..., ...;
  an1, ..., ann ]
```

Consistently-dimensioned variables can be similarly combined into new variables. The conjugate transpose is `A'`. The pure transpose is `A.'`. `size(val)` returns the dimensions of `val` as a row vector. `length(val)` returns the maximum dimension of `val`.

`a[:dx]:b` returns a row vector from `a` to `b` (inclusive) by steps of size `1` or `dx` (may be negative). `linspace(a,b,n)` returns a row vector from `a` to `b` (inclusive) with `n` linear steps. `logspace(a,b,n)` returns a row vector from `10^a` to `10^b` with `n` logarithmic steps.

Indexing is done

- `val(n)`: `n`th element, flattened column wise.
- `val(end)`: last element.
- `val(a[:dx]:b)`: elements indexed by the range.
- `val([...])`: elements indexed by the matrix.
- `val(a,b)`: elements with row `a` and column `b`, where `a` and `b` may be any of the previous kinds of indexes or `:` for all elements.
- `val(idx)`: elements corresponding to the `true` elements of `idx` which is a `logical`.

Copies are deep. `reshape(val,m,n)` reshapes `val` to be `m`×`n`, pivoting column wise.

Special matrices are `zeros([m,]n)`, `eye(n)` (identity), `ones([m,]n)`, `diag(a11,...,ann)`, `rand([m,]n)` (uniform), `randn([m,]n)` (standard normal), and `randi(a,b)` (integer).

A `cell` is created `{val1,...,valn}`, where each `vali` can be of any type. `cell(dim1,...,dimn)` creates an empty `cell`. `val{idx}` returns an element as its original type. `celldisp(val)` displays a `cell` in detail.

3 Algorithms

Operations are `+`, `-`, `*`, `/`, and `^`. Scalar–matrix operations (except `^`), matrix–scalar assignment, consistently-dimensioned operations (except `*`), and comparison are done element wise. Matrix–scalar arithmetic, `^`, and `*` are not done element wise unless the operator is preceded by `.`. One operand may have one consistent dimension and one size-`1` dimension; then it will be expanded perpendicular to the size-`1` dimension for the operation. Element-wise functions are `exp(val)`, `sin(val)`, `abs(val)`, `log(val)` (natural), `mod(val)`, and `fix(val)` (round toward zero).

`sum(val)` returns the sum of each column by default. `mean(val,"all")` returns the mean of all elements. `var(val,0,2)` returns the unbiased variation of each row. `max(val)` returns the maximum element of each column.

Element-wise logical operators are `==`, `~=`, `&`, `~` (unary), `|`, `xor(a,b)`, `any(val)`, `all(val)`. Scalar logical operators `&&` and `||` have lazy evaluation. Logical operators are done element wise. Non-element-wise logical functions are `isequal(a,b)`. `0` converts to `false`, and any other number converts to `true`.

`strcmp(a,b)` compares `string`s and `char`s. `strlength(val)` returns the length of a `string`. `str2num(val)` converts a `string` or `char` to a `double`. `num2str(val,prec)` converts a number to a `char` of length `prec`.

`+` concatenates two `string`s, automatically converting the second operand. `strjoin(val,del)` joins the `string`s in `val` with delimiter `del`.

`split(val,del)` splits `val` into an array by delimiter `del`. `strfind(val,pat)` returns the indices of pattern `pat` in `val`. `contains(val,pat)` returns the index array for elements of `val` containing pattern `pat`.

`numel(val)` returns the number of elements in `val`. `find(val)` returns the indices of the nonzero elements of `val`, as a row vector if `val` is a row vector and as a column vector otherwise. `ismember(val,mem)` returns the index array for elements of `val` containing an element of `mem`.

`num2cell(val)` converts a number to a `cell`. `cell2mat(val)` converts a consistently-typed `cell` to the original type. Comparison to a cell array succeeds if comparison to any element of the cell array succeeds. `cellfun(fun,val)` applies a function element wise.

`rng(seed)` sets the global random seed. `rng('shuffle')` resets the seed.

4 Plots

`plot(x1,y1,...,xn,yn)` generates `n` images of graphs from consistently-dimensioned `x` and `y` with connecting lines. Use a large number of points to approximate a smooth curve. `legend(label1,...,labeln)` prints a legend onto following graphs.

`fplot(func,[a, b])` plots a smooth curve of `func` from `a` to `b`. `integral(func,a,b)` returns the definite integral of `func` from `a` to `b`.

`hold on` keeps succeeding plots in the same figure until `hold off`. `plt.Marker` is the value of the character making the plot points (`o`,`+`,`s`).

5 Add-Ons

The "Parallel Computing Toolbox" add-on allows `parpool(num)` to create a parallel computing pool with `num` workers. Then a parallel for loop is done with `parfor`.

The "Statistics and Machine Learning Toolbox" add-on allows `ksdensity(x)` to return y and x values, respectively, of a smooth approximation of the probability density from the observed values x.