Numerical Analysis
Notes on Matlab

Aleksandar Donev
Courant Institute, NYU[1]
donev@courant.nyu.edu

[1]Course MATH-UA.0252/MA-UY_4424, Spring 2021

Spring 2021

Peculiarities of MATLAB

- MATLAB is an **interpreted language**, meaning that commands are interpreted and executed as encountered. MATLAB caches some stuff though...
- Many of MATLAB's **intrinsic routines** are however compiled and optimized and often based on well-known libraries (BLAS, LAPACK, FFTW, etc.).
- Variables in scripts/worspace are global and persist throughout an interactive session (use *whos* for info and *clear* to clear workspace).
- Every variable in MATLAB is, unless specifically arranged otherwise, a matrix, **double precision float** if numerical.
- Vectors (column or row) are also matrices for which one of the dimensions is 1.
- **Complex arithmetic** and complex matrices are used where necessary.

Matrices

```
>> format compact; format long
>> x=-1; % A scalar that is really a 1x1 matrix
>> whos('x')
  Name        Size                Bytes  Class     Attributes
  x           1x1                     8  double

>> y=sqrt(x) % Requires complex arithmetic
y =                   0 + 1.000000000000000i
>> whos('y')
  Name        Size                Bytes  Class     Attributes
  y           1x1                    16  double    complex

>> size(x)
ans =     1     1
>> x(1)
ans =    -1
>> x(1,1)
ans =    -1
>> x(3)=1;
>> x
x =    -1     0     1
```

Vectorization / Optimization

- MATLAB uses **dynamic memory management** (including garbage collection), and matrices are re-allocated as needed when new elements are added.
- It is however much better to **pre-allocate space** ahead of time using, for example, *zeros*.
- The **colon notation** is very important in accessing array sections, and x is different from $x(:)$.
- **Avoid for loops** unless necessary: Use array notation and intrinsic functions instead.
- To see how much CPU (computing) time a section of code took, use *tic* and *toc* (but beware of timing small sections of code).
- MATLAB has built-in **profiling tools** (*help profile*).

Pre-allocation (fibb.m)

```
format compact; format long
clear; % Clear all variables from memory

N=100000; % The number of iterations

% Try commenting this line out:
f=zeros(1,N); % Pre-allocate f

tic;
f(1)=1;
for i=2:N
    f(i)=f(i-1)+i;
end
elapsed=toc;

fprintf('The result is f(%d)=%g, computed in %g s\n', ...
        N, f(N), elapsed);
```

Vectorization (vect.m)

```
function vect(vectorize)
    N=1000000; % The number of elements
    x=linspace(0,1,N); % Grid of N equi-spaced points

    tic;
    if(vectorize) % Vectorized
        x=sqrt(x);
    else % Non-vectorized
        for i=1:N
            x(i)=sqrt(x(i));
        end
    end
    elapsed=toc;

    fprintf('CPU time for N=%d is %g s\n', N, elapsed);
end
```

MATLAB examples

```
>> fibb % Without pre-allocating
The result is f(100000)=5.00005e+09, computed in 6.53603 s

>> fibb % Pre-allocating
The result is f(100000)=5.00005e+09, computed in 0.000998 s

>> vect(0) % Non-vectorized
CPU time for N=1000000 is 0.074986 s

>> vect(1) % Vectorized -- don't trust the actual number
CPU time for N=1000000 is 0.002058 s
```

Vectorization / Optimization

- (Almost) everything in MATLAB is a double-precision matrix, called **array**.
- Row vectors are just matrices with first dimension 1. Column vectors have row dimension 1. Scalars are 1×1 matrices.
- The syntax x' can be used to construct the **conjugate transpose** of a matrix.
- The **colon notation** can be used to select a subset of the elements of an array, called an **array section**.
- The default arithmetic operators, +, -, *, / and ^ are **matrix addition/subtraction/multiplication**, linear solver and matrix power.
- If you prepend a **dot before an operator** you get an **element-wise operator** which works for arrays of the same shape.

Matrix/vector stuff #1

```
>> x=[1 2 3; 4 5 6] % Construct a matrix
x =     1     2     3
        4     5     6

>> size(x) % Shape of the matrix x
ans =     2     3

>> y=x(:) % All elements of y
y =     1     4     2     5     3     6

>> size(y)
ans =     6     1

>> x(1,1:3)
ans =     1     2     3

>> x(1:2:6)
ans =     1     2     3
```

Matrix/vector stuff #2

```
>> sum(x)
ans =
     5     7     9

>> sum(x(:))
ans =
    21

>> z=1i; % Imaginary unit

>> y=x+z
y =
   1.0000 + 1.0000i   2.0000 + 1.0000i   3.0000 + 1.0000i
   4.0000 + 1.0000i   5.0000 + 1.0000i   6.0000 + 1.0000i

>> y'
ans =
   1.0000 - 1.0000i   4.0000 - 1.0000i
   2.0000 - 1.0000i   5.0000 - 1.0000i
   3.0000 - 1.0000i   6.0000 - 1.0000i
```

Matrix/vector stuff #3

```
>> x*y
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> x.*y
ans =
   1.0000 + 1.0000i   4.0000 + 2.0000i   9.0000 + 3.0000i
  16.0000 + 4.0000i  25.0000 + 5.0000i  36.0000 + 6.0000i

>> x*y'
ans =
  14.0000 - 6.0000i  32.0000 - 6.0000i
  32.0000 -15.0000i  77.0000 -15.0000i

>> x'*y
ans =
  17.0000 + 5.0000i  22.0000 + 5.0000i  27.0000 + 5.0000i
  22.0000 + 7.0000i  29.0000 + 7.0000i  36.0000 + 7.0000i
  27.0000 + 9.0000i  36.0000 + 9.0000i  45.0000 + 9.0000i
```

Coding Guidelines

- Learn to reference the **MATLAB help**: Including reading the examples and "fine print" near the end, not just the simple usage. **Know what is under the hood!**
- **Indendation, comments, and variable naming** make a big difference! Code should be readable by others.
- Spending a few extra moments on the code will pay off when using it.
- Spend some time learning how to **plot in MATLAB**, and in particular, how to plot with different symbols, lines and colors using *plot, loglog, semilogx, semilogy.*
- Learn how to **annotate plots**: *xlim, ylim, axis, xlabel, title, legend.* The intrinsics *num2str* or *sprintf* can be used to create strings with embedded parameters.
- Finer controls over fonts, line widths, etc., are provided by the intrinsic function *set*...including using the LaTex interpreter to typeset mathematical notation in figures.