Computational Finance

Introduction to Matlab

Marek Kolman
Matlab

- program/programming language for technical computing
- particularly for numerical issues
- works on matrix/vector basis
- usually used for functional programming (X object programming)
Matlab GUI

- command window
- workspace
- history
Variables in Matlab

- boolean
- scalar
- vector
- matrix
- others

Matlab works with both $\mathbb{R}$ and $\mathbb{C}$
Vector

- vec = [1, 2, 3, 4, 5]
- index starts with 1, not 0
- subvector is vec(a:b)
- vector can be defined as a list of values, or e.g. like this: 1:5 or linspace(1,5,5)
- sum the above vector and a number 1
- sum the above vector and 11:15
- sum the above vector and 11:14
- remove the number 1 from the beginning of the vector
- add number $\pi$ to the end of the vector
Matrix

- matrix is a "vector of vectors"
- matrix 3x3 can be represented as mat=[1, 2, 3; 4, 5, 6; 7, 8, 9] (the semicolon is a row separator)
- matrix can be generated by a list of values, assembled from vectors or generated by some function (eye, magic)
- what is the trace of the matrix mat?
- what is the rank of the matrix mat?
- what is the diagonal of the matrix mat?
- pick a submatrix from the matrix mat. It includes rows 1 to 2 and columns 1 up to the last column
- append a row vector [20 30 40] to the matrix (from below)
- append a column matrix [20 30 40] to the matrix (from the right)
Solution to a linear system

Solve

\[
\begin{align*}
6x_1 + 31x_2 + 73x_3 &= 5 \\
-\pi x_1 + \sin(\sqrt{2})x_2 + e^2 x_3 &= 19 \\
2x_1 + 11x_2 - 91x_3 &= 41
\end{align*}
\]

- an alternative is to use a ”matrix division” \( A \backslash b \)
Element-wise product vs product

- Multiplication in Matlab is not the same kind of multiplication we assume.

- Matlab, by default, assumes matrix (vector) multiplication.

- Try to square the vector \([1, 2, 3, 4]\).

- Try to multiply \([1, 2, 3, 4]\) by the vector \([10, 20, 30, 40]\).

- Try to multiply \([1, 2, 3, 4]\) by the vector \([10; 20; 30; 40]\).

- Element-wise operations are performed with . (dot operator), matrix operations without the dot.
Conventions of naming variables

- good code is easy to understand (not only for you but also for others)
- use intuitive names of variables
- forget about variables such as $x, y$ but use rather $prices, volatility$ etc.
- variable name composed of two words is usually written as $vecPrices, sumOfSquares$
- or alternatively $vec\_prices, sum\_of\_squares$
- note that if you refer to no-existing variable Matlab might use $[]$ instead
In-built functions

- there is a myriad of pre-defined functions in Matlab (beware using a variable with the same name)
- useful common functions `log`, `exp`, `sum`, `zeros`, `size`, `numel`, `find`, `min`, `max`, `quad`...
- then also functions form a particular package. E.g. `randn`, `normcdf`, `fmincon`, `fzero`, `fft`
- for a standard user the list of functions is endless
- rule - use the inbuilt functions as frequently as possible. They are well optimized
Function vs. procedure

- procedure performs some action but does not return an output
- function *does* return an output
- functions are the cornerstone of Matlab
- functions can be stored as a file (’.m file’) or dynamically in the memory (so called anonymous function)
- syntax (anonymous vs function as a file)
- write a function (a file) that returns square root and square of a given argument (i.e. it has two outputs)
- make an anonymous function that computes
  \[ f(x|b) = bx_1^2 - e^{-bx_2} \]
- anonymous functions are, in general, used for simple computations
Matlab vs other programming languages

Why Matlab

- difficult applications can be coded with a little effort
- vector/matrix basis for computations
- no need for variable datatype definition
- intimidating amount of inbuilt functions

Why, say, C++

- object oriented
- strict! Code does not run unless everything is completely defined
- indices start from 0. In finance we often use variables such as $S(0) = S_0$. Here 0-indexing is more intuitive
- nested applications can be controlled easier
- richer set of debug options
Controlling program-flow

- typical application is nested. An example is a choice to price american or european option
- typical application has a loop. We might for example calculate value of a bond maturing in \([1, 2, 3, \ldots]\) years
- typical application includes binary if statement. For example, if a year is a leap year it has 366 days otherwise 365
- typical application is doing something until some condition is valid. For example, find parameters that provide to best match to a yield curve and iterate unless the match is good enough
- in the above examples you should see command *switch, for, if* and *while*, respectively. The most common commands are *for, if*
test, if both $A$ and $B$ are true: $(A \&\& B)$

- test, if either $A$ or $B$ is true: $(A || B)$

- test, if $A$ is different from $B$: $(A \sim= B)$

- test, if $A$ is equal to $B$: $(A == B)$

Attention! A frequent mistake is to use $(A = B)$ for equality check!

Let $A = 3$, and $B = -5$. Test if:

- $A \leq 4$ and simultaneously $B > -10$
- $A < 12$ or $B = 7$
- $A \neq B$
- $A$ coincides with $B$
Back to program-flow: if and for

if:

- syntax 1: `if (cond) do something; else do something else; end`
- syntax 2: `if (cond) do something I; elseif (cond) some something II else do something else end`
- code a function that returns number -1 if an argument $x$ is less than 10, number 0 when $x$ equals 10 and number 1 if $x$ is greater than 10

for

- syntax: `for $i=1$:$\text{count}$ do something($i$) end`
- $i$ is called iteration variable
- write a program (function) that calculates sum of all elements in a vector, in terms of `for` and without using `sum`
- assume a vector [1, 2, 3, 4, 5]. Write a program that for every number from this vector calculates a) square if the number is even and b) square root if the number is odd
- use the function ($mod(x, y)$)
Back to program-flow: while and switch

while
  ■ this loop statement runs until a condition is true
  ■ syntax `while (cond) do something end`
  ■ `while` is actually a generalized `for` cycle

switch
  ■ command serves to choose among various options
  ■ syntax: `switch (cond) case option1 do choice 1 case option2 do choice 2 otherwise do something else end`
  ■ it actually replaces a sequence of `elseif` (looks cool)
Speeding up in Matlab

- Speed of code depends on how good is your implementation
- A stupid mistake can bottleneck your algorithm
- Matlab loves if you preallocate arrays
- Write a function with three options each of which will create a \( n \)-dimensional vector of random numbers
  - Option 1 uses a `for` loop. In every iteration extend the vector by one random number
  - Option 2 - As 1 but preallocate the array and insert a random number in every iteration
  - Option 3 - Create one-shot \( n \)-dimensional random vector

Rule: Readable code is always superimposed to speed!
Visual outputs in Matlab

- basic command for a plot is... suprisingly \textit{plot}. This is a highly customizable object
- make a plot of functions $\sin(x)$ and $\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$ in one figure $x \in [-4, 4]$, spacing 0.1
- the first plot will be visualized by a red dashed line
- the second plot will be represented by green circles
- label the plot 'two functions'
- show a grid in the plot
- enable legend
- name the axes $x$ and $f(x)$, respectively
If we still have time...

- code a matrix multiplication function
- code a factorial function (using recursion and a loop)