Introduction to Matlab

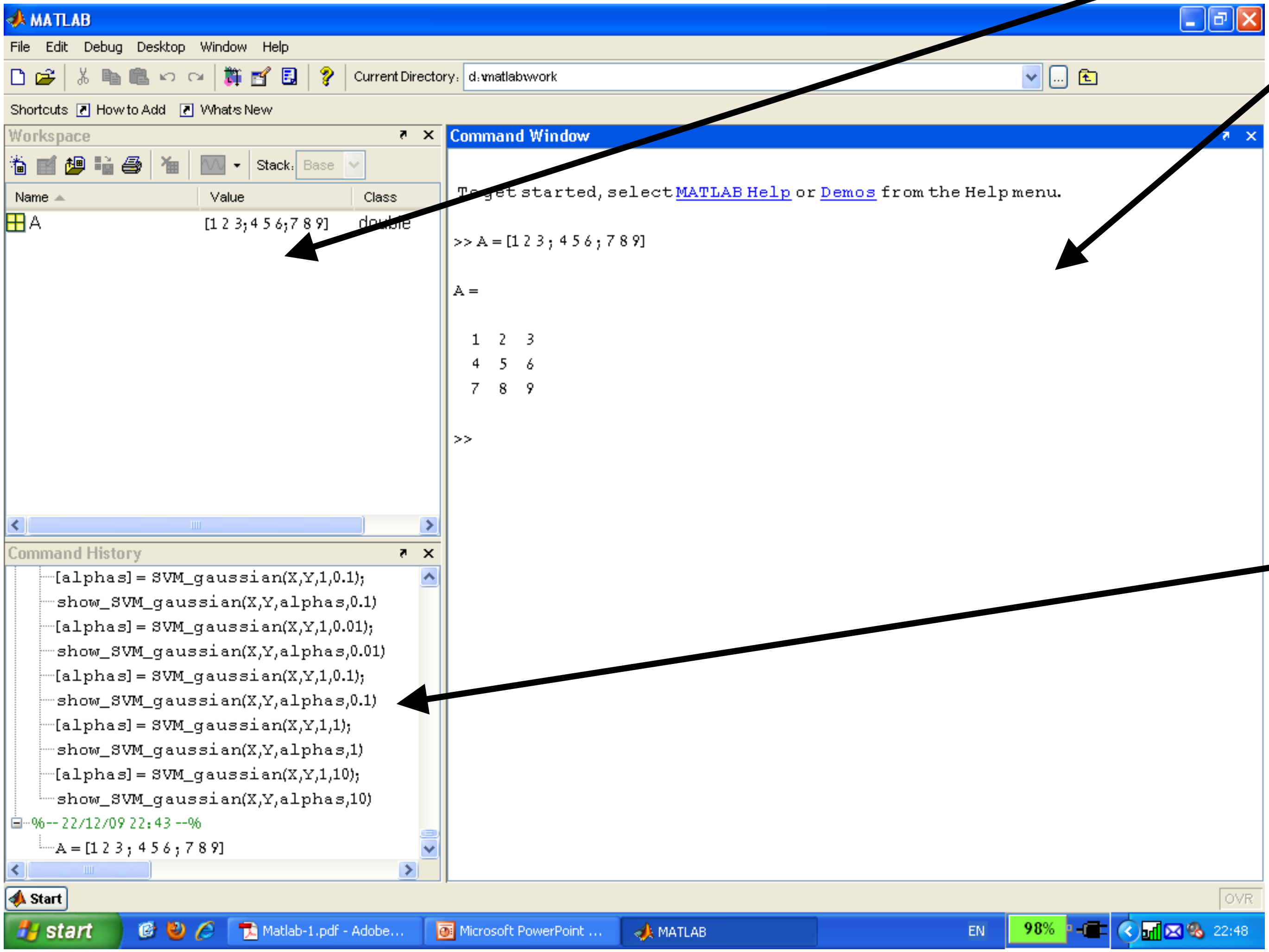
Prepared by Ohad Shamir, based on slides by Yonatan Amit

- This presentation is intended to provide a quick and dirty introduction to Matlab, focusing on stuff you’ll need for the programming mini-project.
- There are tons of other things you can do with Matlab.
- If you have a question which is not answered here, see Matlab’s help, google it, or email me.

Why Matlab?

- Matlab is a programming tool for the lazy.
- Used mainly for numerical computation.
  - Basic data structure is a matrix.
  - Has tons of functions for mathematical calculations.
- Generally easy and forgiving (no memory management; programs are interpreted rather than compiled…)
- Excellent for trying out algorithms based on numerical computations (such as those taught in the course)
  - For full-scale, speed-optimized implementations, other programming languages are often used (such as C++).
- Matlab is installed on all public computers on campus.

Welcome to Matlab



Current Directory: d:\matlab\work

Name	Value	Class
A	[1 2 3;4 5 6;7 8 9]	double

```
To get started, select MATLAB Help or Demos from the Help menu.

>> A = [1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>>
```

```
[alphas] = SVM_gaussian(X,Y,1,0.1);
show_SVM_gaussian(X,Y,alphas,0.1)
[alphas] = SVM_gaussian(X,Y,1,0.01);
show_SVM_gaussian(X,Y,alphas,0.01)
[alphas] = SVM_gaussian(X,Y,1,0.1);
show_SVM_gaussian(X,Y,alphas,0.1)
[alphas] = SVM_gaussian(X,Y,1,1);
show_SVM_gaussian(X,Y,alphas,1)
[alphas] = SVM_gaussian(X,Y,1,10);
show_SVM_gaussian(X,Y,alphas,10)
%-- 22/12/09 22:43 --%
A = [1 2 3; 4 5 6; 7 8 9]
```

Variables

Work area (command line interface)

History

Basic Workflow

- Commands are executed from the command line interface.
- Basically, an elaborate calculator.
  - Try typing `1+1` <enter>.
  - Try `log(sqrt(3)+4^2)` <enter>
  - Can also store variables! Try `x=3` <enter>, `sin(x)` <enter>

Basic Workflow

- (Almost) anything in Matlab is a matrix.
- Scalars (like x in the previous slide) are just a 1*1 matrix.
- Try the following and see what happens:
  - `A =[1 2 3 ; 4 5 6 ; 7 8 9]`
  - `B = [4;5;6]`
  - `w = A*B` (multiplies the matrices)
  - `w*A` (should return an error..)
  - `w’` (this transposes w)
  - `w’*A` (now it works…)

Easy Ways to Create Matrices and Vectors

- Try the following:
  - `A=1:3`
  - `A=(1:3)’`
  - `A = 1:2:6` (go from 1 till 6 in steps of 2)
  - `A = 5:-2:0` (go from 5 down to 0 in steps of -2)
  - `A = zeros(5,2)` (create a zero matrix with 5 rows and 2 columns)
  - `A = ones(3,4)` (create a matrix of ones with 3 rows and 4 columns)
  - `A = eye(5)` (create an identity matrix of size 5*5).

Hiding the Output

- Often, we want to do a calculation without seeing the output explicitly.
  - Try `A=ones(500,500)`. This should look ugly..
    - Tip: press `CTRL+<c>` to halt a computation which takes too long.
  - Now try `A=ones(500,500);`
  - The matrix was created and stored as A, without us seeing the matrix explicitly.

Accessing a matrix

- Given a matrix A:
  - We can access its i,j element: `A(i,j)`
  - We can access its i’th column: `A(:,i)`
  - We can access its last column: `A(:,end)`
  - We can access the last 10 rows:

```
A(end-10:end,:)
```

  - It is also useful for assignments:

```
A(1,[3 4]) = [5 6]
```

  - And so forth…

What else we can do with matrices

- Erase them: `clear A`
- Erase all of them: `clear`
- Save them: `save filename A`
  - Data is saved into a .mat file
- Load them: `load filename`
- List them: `who` or `whos`

Operations on Matrices

- Scalar operations
  - `A*5` (multiplies all entries by 5)
  - `A+5` (add 5 to all entries)
- Matrix operations
  - `A*B, A+B`
  - `A’`
- Element-wise operations
  - `A.*B`
  - `A.^B`

Matrix dimensions must agree

Scripts and Functions

- What if we want to do a complicated calculation? (e.g. run a perceptron algorithm on some data…)
- Matlab provides two mechanisms to encapsulate code: scripts and functions.
- In both cases, you write code as a separate file (called an m-file), and then invoke the script/function from the command line.

Scripts

- Just a sequence of commands.
- Go to File->New->M-file. This opens the m-file editor.
- Type (for example):

```
A = [1 2 3];
B = [4 5 6];
A.*B
```

- Save this file as `xxx.m` in Matlab’s current directory.
- Now, go to the command prompt, and type `xxx.`

Functions

- Also written as an m-file.
- Must have input and an output.
- M-file must begin with a declaration such as:

```
[a,b] = my_function(c,d,e)
```

- File name should be the same as function name.
- The file may contain other private functions (not needed for the miniproject…)
- The function can be invoked from the command line, just like any other function.

If…

- Create some scalar `a`, and try to run the following script:

```
if (a==5)
  display(‘a is just right’);
else
  display(‘a is not right’);
end
```

If…

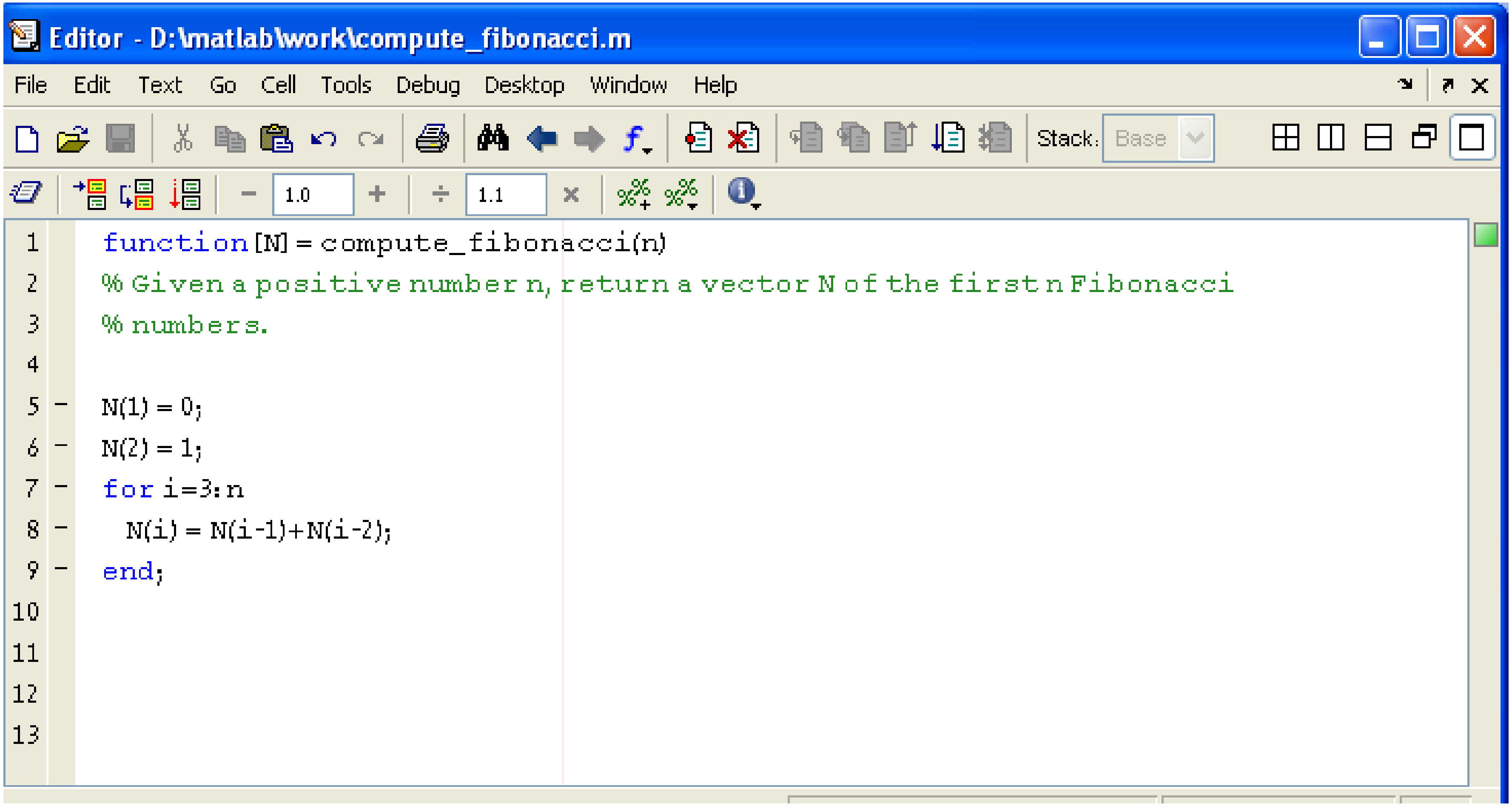
- Create some scalar `a`, and try to run the following script:

```
if (a>5)
  display(‘a is too big’);
elsif (a<5)
  display(‘a is too small’);
else
  display(‘a is just right’);
end
```

Looping

```
For i=[1,6:8]
  i+3
end

i=0;
while i<3
  i = i+1;
  disp('Hello');
end
```

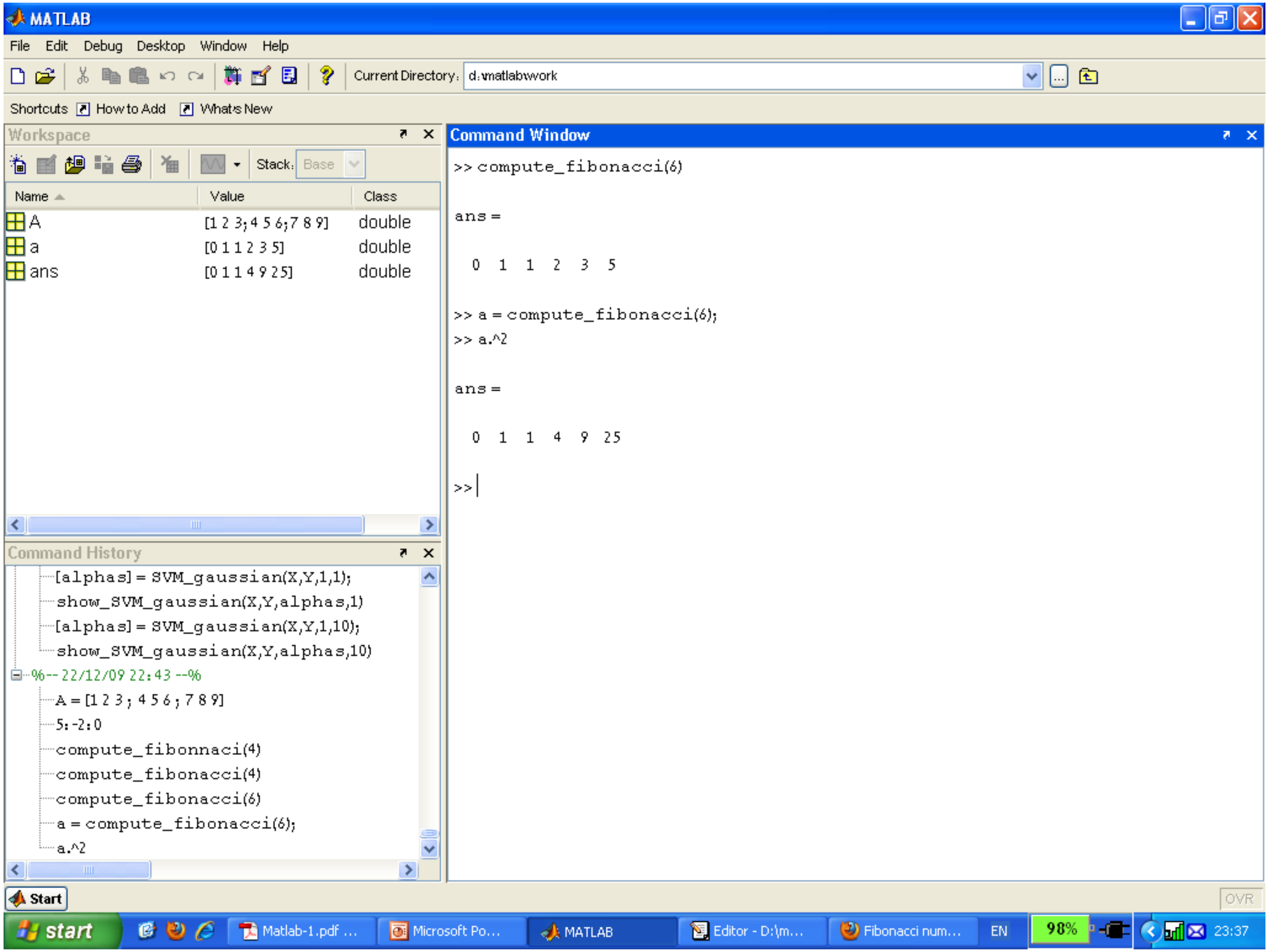
Example



Editor - D:\matlab\work\compute_fibonacci.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1

```
function [N] = compute_fibonacci(n)
% Given a positive number n, return a vector N of the first n Fibonacci
% numbers.

N(1) = 0;
N(2) = 1;
for i=3:n
  N(i) = N(i-1)+N(i-2);
end;
```

Example



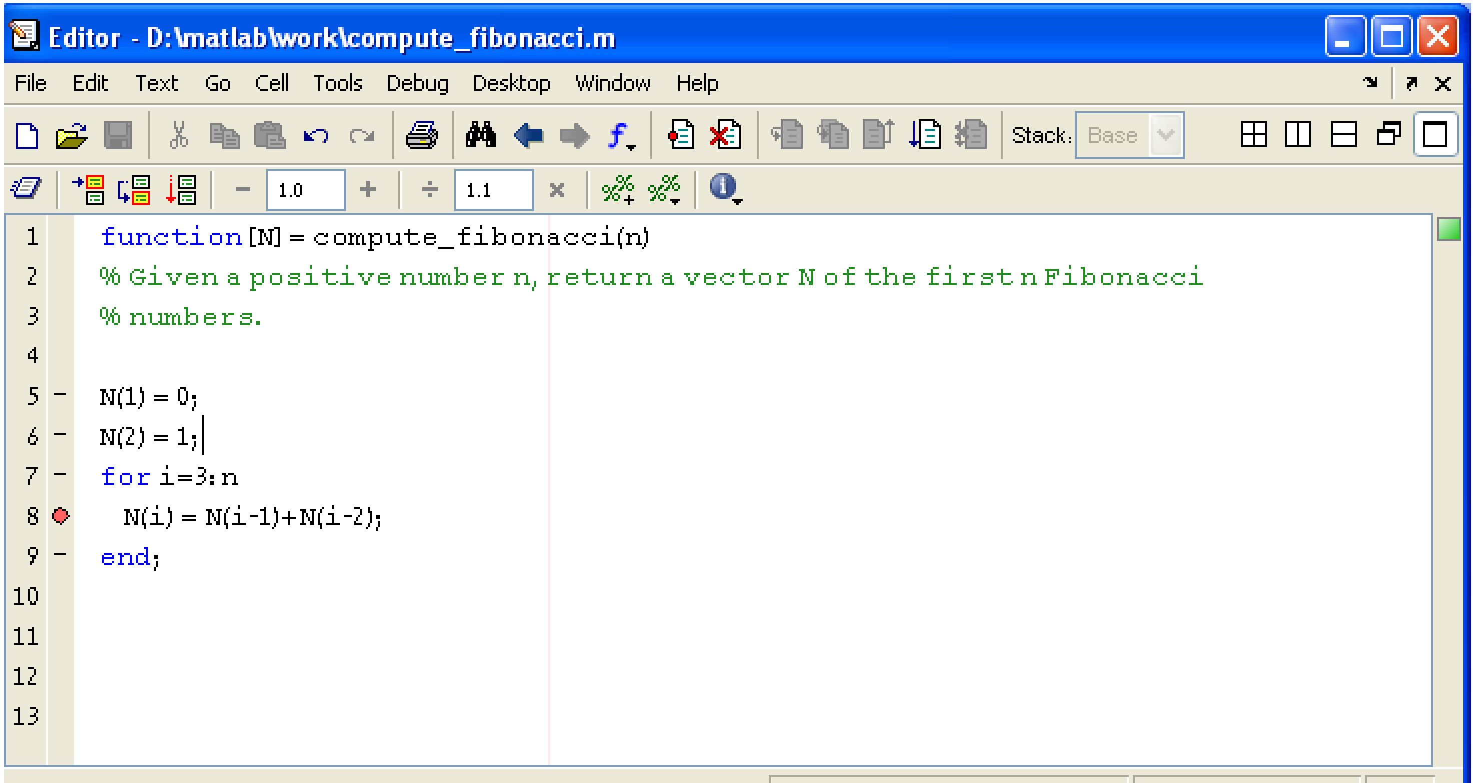
Name	Value	Class
A	[1 2 3;4 5 6;7 8 9]	double
a	[0 1 1 2 3 5]	double
ans	[0 1 1 4 9 25]	double

```
>> compute_fibonacci(6)

ans =

   0   1   1   2   3   5

>> a = compute_fibonacci(6);
>> a.^2

ans =

   0   1   1   4   9  25

>>
```

```
[alphas] = SVM_gaussian(X,Y,1,1);
show_SVM_gaussian(X,Y,alphas,1)
[alphas] = SVM_gaussian(X,Y,1,10);
show_SVM_gaussian(X,Y,alphas,10)
%-- 22/12/09 22:43 --%
A = [1 2 3 ; 4 5 6 ; 7 8 9]
5:-2:0
compute_fibonnaci(4)
compute_fibonacci(4)
compute_fibonacci(6)
a = compute_fibonacci(6);
a.^2
```

The Matlab Debugger

- Edit a file and mark a line by clicking on the gray area to the left of the line.


Editor - D:\matlab\work\compute_fibonacci.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: Base
1.0
1.1

```
function [N] = compute_fibonacci(n)
% Given a positive number n, return a vector N of the first n Fibonacci
% numbers.

N(1) = 0;
N(2) = 1;
for i=3:n
  N(i) = N(i-1)+N(i-2);
end;
```

The Matlab Debugger

- When you run the function, the execution will stop when the breakpoint is reached, and control is returned to the command line.
- You can then check the variables, add/remove breakpoints etc.
- Use the debug menu for continuing (continue as normal, step one command ahead etc…)
- Can also add conditions to the breakpoints.

Programming Matlab the right way

- Matlab is very fast in doing calculations over matrices.
- It is *very* slow in doing loops.
- Always try to avoid loops if possible, by doing vector/matrix calculations.

Programming Matlab the right way

- Suppose you want to calculate the norm of a kernel classifier, given weights $\alpha_1,\ldots,\alpha_m$ and the training instances $x_1,\ldots,x_m$. Recall that:

$$\|w\|^2 = \left\langle \sum_{i=1}^{m} \alpha_i \Psi(x_i), \sum_{i=1}^{m} \alpha_i \Psi(x_i) \right\rangle$$

$$= \sum_{i,j=1}^{m} \alpha_i \alpha_j \langle \Psi(x_i), \Psi(x_j) \rangle$$

$$= \sum_{i,j=1}^{m} \alpha_i \alpha_j K(x_i, x_j)$$

Programming Matlab the right way

- So, if we have a matrix `G` where row i and column j contain $K(x_i, x_j)$, and a vector `alphas` where `alphas(i)` is α_i, the naïve way to calculate the norm is:

```
norm2 = 0;
for i=1:m
  for j=1:m
     norm2 = norm2+alphas(i)*alphas(j)*G(i,j);
  end
end
classifier_norm = sqrt(norm2);
```

Programming Matlab the right way

- The smart way is to do the following (try to understand why it is equivalent)

```
classifier_norm = sqrt(alphas’*G*alphas);
```

- On my laptop, on a 1000*1000 matrix `G`, this implementation runs 435 times faster than the naïve implementation!

A list of useful functions

- repmat : replicate a vector/matrix, e.g.

```
>> repmat([1 2 3],3,1)

ans =

     1     2     3
     1     2     3
     1     2     3
```

A list of useful functions

- `size(X,1)`: number of rows of matrix X.
- `size(X,2)`: number of columns of matrix X.
- `sum(a)`: sum the elements of a vector a.
- `sum(X,1)`: sum the rows of matrix X.
- `sum(X,2)`: sum the columns of matrix X.

A list of useful functions

- `rand`: Return a random number uniformly distributed on [0,1].
- `randi(n)`: Return a random number uniformly distributed on {0,1,..,n}.
- `randperm(m)`: return a random permutation of the numbers 1,2,…,m
- `floor(x)`: return the largest integer smaller than a number x.
- `[val,ind] = min(a)`: return the smallest element `val` in vector `a`, so that `a(ind)` is equal to `val`.
- `find(a>5)`: return the indices of the entries in vector `a` which are larger than 5 (for example…)