Quick introduction to Matlab

PASCAL Bootcamp in Machine Learning – 2007
Edited by Luca Cosmo
Outline

• Matlab introduction
• Matlab elements
  • Types
  • Variables
  • Matrices
• Scripts and functions
• Matlab Programming language
• Ploting
Matlab introduction

• Matlab is a program for doing numerical computation. It was originally designed for solving linear algebra type problems using matrices. It’s name is derived from MATrix LABoratory.

• Matlab is also a programming language that currently is widely used as a platform for developing tools for Machine Learning
Matlab main features

- Large toolbox of numeric/image library functions
- Very useful for displaying, visualizing data
- **High-level:** focus on algorithm structure, not on low-level details
- Allows quick prototype development of algorithms
- Powerful debugging features
Matlab introduction

- Some other aspects of Matlab
  - Matlab is an interpreter -> not as fast as compiled code
    - Typically quite fast for an interpreted language
    - Often used early in development -> can then convert to C (e.g.,) for speed
  - Can be linked to C/C++, JAVA, SQL, etc
  - Commercial product, but widely used in industry and academia
    - Many algorithms and toolboxes freely available
Opening Matlab

- Current Directory
- Script Editor
- Command Window
- Working Path
- Working Memory
- Command History
Help

- Within Matlab
  - Type `help` at the Matlab prompt or `help` followed by a function name for help on a specific function
  - Type `doc` to get the graphical version of help
- Online
  - Online documentation for Matlab at the MathWorks website
  - There are also numerous tutorials online that are easily found with a web search.
Data Types

```
ARRAY
  [full or sparse]
   logical  char  NUMERIC  cell  structure  function
             handle
  int8, uint8,
  int16, uint16,
  int32, uint32,
  int64, uint64
    single  double
user classes  java classes
```
Variables

- Have not to be previously declared
- Variable names can contain up to 63 characters
- Variable names must start with a letter followed by letters, digits, and underscores.
- Variable names are case sensitive

```matlab
>> x = 10     --> x = 10
>> y = 3e-3   --> y = 0.0030
>> a = 'hello' --> a = hello
>> A       --> ??? Undefined function or variable 'A'.
```
Workspace

All the assigned variables are added to the workspace.

You can remove a specific variable form the workspace using:

```matlab
>> clear 'var_name'
```

or remove all the variables using:

```matlab
>> clear
```
We can see the value of a variable by typing `is name` on the command window:

```plaintext
>> z --> z = 3
```

Terminating a line with a `;` suppresses the output of the assigned variable value:

```plaintext
>> x=10 --> x = 10
>> x=10; -->
```

In the expression is not an assignment its value is automatically assigned to the special variable `ans`:

```plaintext
>> 10 --> ans = 10
```
The default printing format shows only the first 4 decimal of a number:

```matlab
>> x=1/3
x =
 0.3333
```

With the *format* command, you can set different output formats for numbers:

```matlab
>> format long
>> x=1/3
x =
 0.333333333333333
```

**EXERCISE**: Use the help of Matlab to find out how to print a number in scientific notation \(3.3333\times10^{-1}\)
Matlab Assignment & Operators

Assignment  =  \texttt{a = b}  \textit{(assign b to a)}
Addition    +  \texttt{a + b}
Subtraction -  \texttt{a - b}
Multiplication  \*  or  .\*  \texttt{a*b}  or  \texttt{a.*b}
Division  /  or  ./  \texttt{a/b}  or  \texttt{a./b}
Power  ^  or  .^  \texttt{a^b}  or  \texttt{a.^b}

Function call:

\texttt{func\_name(p1, p2, ...)}
\texttt{[o1, o2, ...] = func\_name(p1, p2, ...)}
Operators exercise

Given two points in $\mathbb{R}^2$ with coordinates $x_1, y_1$ and $x_2, y_2$, compute their Euclidean distance:

- Assign a value to $x_1, y_1, x_2, y_2$
- Use the operators provided by Matlab to compute the distance between the two points and assign it to a variable $d$

$$A = Y_2 - Y_1$$
$$B = X_2 - X_1$$
$$C^2 = A^2 + B^2$$
Solution

```matlab
>> x1 = 1;
>> y1 = 1;

>> x2 = 2;
>> y2 = 2;

>> d = ((x1-x2).^2 + (y1-y2).^2).^0.5
   d =
       1.4142e+000
```
# Matlab Usefull Constants

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pi</code></td>
<td>Value of ( \pi )</td>
</tr>
<tr>
<td><code>eps</code></td>
<td>Smallest incremental number</td>
</tr>
<tr>
<td><code>inf</code></td>
<td>Infinity</td>
</tr>
<tr>
<td><code>NaN</code></td>
<td>Not a number e.g. 0/0</td>
</tr>
<tr>
<td><code>realmin</code></td>
<td>The smallest usable positive real number</td>
</tr>
<tr>
<td><code>realmax</code></td>
<td>The largest usable positive real number</td>
</tr>
</tbody>
</table>

The inbuilt Matlab constants can be overwritten.
Matlab Matrices

• Matlab treats all variables as matrices. For our purposes a matrix can be thought of as a bidimensional array, in fact, that is how it is stored.

• Vectors are special forms of matrices and contain only one row OR one column.

• Scalars are matrices with only one row AND one column
Matlab Matrices

• A matrix with only one row is called a row vector. A row vector can be created in Matlab as follows (note the commas):

```matlab
>> rowvec = [12, 14, 63]
rowvec =
  12  14  63
```
Matlab Matrices

- A matrix with only one column is called a column vector. A column vector can be created in MATLAB as follows (note the semicolons):

```matlab
» colvec = [13 ; 45 ; -2]
```

```
colvec =

13
45
-2
```
Regularly spaced vectors

- A regularly spaced vector can be created using the colon (:) operator.

<table>
<thead>
<tr>
<th>j:k</th>
<th>is the same as [j,j+1,...,k], or empty when j &gt; k.</th>
</tr>
</thead>
<tbody>
<tr>
<td>j:i:k</td>
<td>is the same as [j,j+i,j+2i, ...,j+m*i], where m = fix((k-j)/i), for integer values. This syntax returns an empty matrix when:</td>
</tr>
<tr>
<td></td>
<td>• i == 0,</td>
</tr>
<tr>
<td></td>
<td>• i &gt; 0 and j &gt; k,</td>
</tr>
<tr>
<td></td>
<td>• i &lt; 0 and j &lt; k.</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\gg 1:4 & \rightarrow \text{ans} = 1 \quad 2 \quad 3 \quad 4 \\
\gg 1:2:10 & \rightarrow \text{ans} = 1 \quad 3 \quad 5 \quad 7 \quad 9
\end{align*}
\]
Matlab Matrices

- A matrix can be created in Matlab as follows (note the commas AND semicolons):

```matlab
>> matrix = [1, 2, 3; 4, 5, 6; 7, 8, 9]
```

```
matrix =

1 2 3
4 5 6
7 8 9
```
Selecting an element of a vector/matrix

We can access the n-th element of a vector by using the \((n)\) operator.

The indexes in Matlab start from 1

To access the first element of a row vector:

\[
\text{rowvec}(1) \quad \rightarrow \quad \text{ans} = 12
\]

To access the an element of a matrix we have to indicate its row,\(i\) and column indexes:

\[
\text{matrix}(1,1) \quad \rightarrow \quad \text{ans} = 1
\]
Extracting a Sub-Matrix

• A portion of a matrix can be extracted and stored in a smaller matrix by specifying the names of both matrices and the rows and columns to extract. The syntax is:

\[
\text{sub_matrix} = \text{matrix} \left( r1 : r2 , c1 : c2 \right) ;
\]

where \( r1 \) and \( r2 \) specify the beginning and ending rows and \( c1 \) and \( c2 \) specify the beginning and ending columns to be extracted to make the new matrix.
Matlab Matrices

<table>
<thead>
<tr>
<th>A column vector can be extracted from a matrix. As an example we create a matrix below:</th>
</tr>
</thead>
</table>
| ```matlab
  » matrix = [1, 2, 3; 4, 5, 6; 7, 8, 9]
```
| matrix =
| 1  2  3 |
| 4  5  6 |
| 7  8  9 |

<table>
<thead>
<tr>
<th>Here we extract column 2 of the matrix and make a column vector:</th>
</tr>
</thead>
</table>
| ```matlab
  » col_two = matrix(:, 2)
```
| col_two =
| 2 |
| 5 |
| 8 |
Matlab Matrices

- A row vector can be extracted from a matrix.
- Here we extract row 2 of the matrix and make a row vector.  
  Note that the 2 specifies the second row and the 1:3 specifies which columns of the row.

```matlab
» rowvec = matrix(2, 1 : 3)
rowvec =
    4     5     6
```
Colon Operator - Examples

>> 1:10
ans =
    1   2   3   4   5   6   7   8   9  10

>> X = 1:3:20
X =
    1   4   7  10  13  16  19

>> A = diag(X);
>> A(end-1:end,end-1:end)
ans =
    16     0
    0    19
## Colon Operator

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>j:k</td>
<td>is the same as [j,j+1,...,k] is empty if j &gt; k</td>
</tr>
<tr>
<td>j:i:k</td>
<td>is the same as [j,j+i,j+2i, ..,k] is empty if i &gt; 0 and j &gt; k or if i &lt; 0 and j &lt; k</td>
</tr>
<tr>
<td>A(:,j)</td>
<td>is the j-th column of A</td>
</tr>
<tr>
<td>A(i,:)</td>
<td>is the i-th row of A</td>
</tr>
<tr>
<td>A(:,:,k)</td>
<td>is the k-th page of three-dimensional array A.</td>
</tr>
<tr>
<td>A(i,j,k,:)</td>
<td>is a vector in four-dimensional array A. The vector includes A(i,j,k,1), A(i,j,k,2), A(i,j,k,3), and so on.</td>
</tr>
<tr>
<td>A(;)</td>
<td>is all the elements of A, regarded as a <strong>single column</strong>. On the left side of an assignment statement, A(;) fills A, preserving its shape from before. In this case, the right side must contain the same number of elements as A.</td>
</tr>
</tbody>
</table>

For matrices this is the same as A.
Matlab Matrices

- **Accessing Single Elements of a Matrix**
  \[ A(i,j) \]

- **Selecting the diagonal elements**
  \[ d = \text{diag}(A) \]
  
  d is a vector containing the diagonal elements of A

- **Accessing Multiple Elements of a Matrix**
  \[ A(1,4) + A(2,4) + A(3,4) + A(4,4) \]
  \[ \text{sum}(A(1:4,4)) \text{ or } \text{sum}(A(:,end)) \]
  
  The keyword `end` refers to the *last* row or column.

- **Deleting Rows and Columns**
  to delete the second column of X, use \[ X(:,2) = [] \]

- **Concatenating Matrices A and B**
  \[ C=[A;B] \]
Some matrix functions in Matlab

- \( X = \text{ones}(r,c) \)  \% Creates matrix full with ones
- \( X = \text{zeros}(r,c) \)  \% Creates matrix full with zeros
- \( A = \text{rand}(r,c) \)  \% Creates a matrix with random numbers uniformly distributed in \([0,1]\)
- \( B = \text{diag}(x) \)  \% Creates squared matrix with vector \( x \) in diagonal

- \([r,c] = \text{size}(A)\)  \% Return dimensions of matrix \( A \)
- \(+, -, *, .+,.-, .* ,./\)  \% Standard operations
- \( v = \text{sum}(A) \)  \% Vector with sum of columns
You can transpose a matrix using ‘:

\[ A = B' \]

\[
A = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
\end{bmatrix}
\]

\[ B = A' \]

\[
B = \begin{bmatrix}
1 & 4 \\
2 & 5 \\
3 & 6 \\
\end{bmatrix}
\]
Exercise

Play with matrix indices and operators:
- Create a random 4x4 matrix
- Print the second column
- Subtract the first column from the diagonal
- Create a matrix of N vectors in R^2 (Nx2)
- Compute and print the Euclidean norm of the vectors
- Consider the previous matrix as a matrix of points, select the points which distance from the origin is lower than the average distance (of all points from the origin).

HINT: - use the Matlab help to learn about the `find` command;
- the function `sqrt(A)` compute the square root of each element of the matrix
**M-files**

- M-Files are text files containing Matlab programs.
- Can be called from the command line or from other M-files.
- Present “.m” extension.
- Two kind of M-files:
  - Scripts
  - Functions
Matlab Editor

- Matlab comes with its own text editor.
- To edit the file `myscript.m` enter the command `edit myscript`
- If the file `myscript.m` does not exist a new empty file will be created in the current directory.
M-files: Scripts

- Without input arguments, they do not return any value.
- They are simply a list of commands that are executed in sequence.
- The commands of a script use the current workspace.
M-files: Script Example

1) >> **edit** statistics.m
2) Write into the editor:

3) Save the file
4) >> **run** statistics
5) >> mean_x, stddev_x

```
x = [4 3 2 10 -1];
n = length(x);
sum1 = 0; sum2 = 0;
for i=1:n
    sum1 = sum1 + x(i);
    sum2 = sum2 + x(i)*x(i);
end
mean_x = sum1/n;
stddev_x = sqrt(sum2/n – mean_x*mean_x);
```

mean_x = 3.6000
stddev_x = 3.6111
M-files: Functions

- With parameters and returning values.
- Only visible variables defined inside the function or parameters (they have their own workspace).
- Usually one file for each function defined.
- Structure:

```matlab
function [out1, ..., outN] = name-function (par1, ..., parM)
    sentence;
    ...
    sentence;
end
```
M-files: Functions Example

1) >>> edit fstatistics.m
2) Write into the editor:

```
function [mean_x, stddev_x] = fstatistics(x)
    n = length(x);
    [sum1,sum2] = sums(x,n);
    mean_x = sum1/n;
    stddev_x = sqrt(sum2/n - mean_x.^2);
end
```

3) Save the file
4) >>> edit sums.m
5) Write into the editor:

```
function [outsum1,outsum2] = sums(y,m)
    outsum1 = 0;
    outsum2 = 0;
    for i=1:m
        outsum1 = outsum1 + y(i);
        outsum2 = outsum2 + y(i)*y(i);
    end
end
```

6) Save the file
7) >>> [p,d] = fstatistics([4 3 2 10 -1])
   p = 3.6000
   d = 3.6111
M-file execution

- We can execute a m-file writing its name on the console:

```plaintext
>> sayHello
    hw = Hello World
```

- We can run the current file in the editor pressing **F5**
- We can run the selected commands in the editor pressing **F9**

- In the m-file we can delimit some portion of the commands using two comment characters `%%`
  A delimited section using **CTRL+Enter**
We can run only m-files located in the Matlab Search Path or in current directory.

- We can add a folder temporarily to the Search Path using:

  ```matlab
  addPath(‘directory_path’)
  ```

- Or permanently:
  1. Go to "File->Set Path" from within MATLAB or type "pathtool" at the MATLAB prompt.
  2. Use the "Add" button to add your desired folder(s) to the MATLAB path.
  3. Click "Save" so that this path is used in future MATLAB sessions.
Debugging a .m file

Matlab have a powerful debugger.

We can set/unset a breakpoints clicking on the right side of the line number. The execution flow stops when a breakpoint is reached and we can:

- watch the workspace state of the function scope.
- change the value of the variables
- run commands that uses the current workspace.

```matlab
function [p,Xin,Xout] = aproxPi(n)

%Generate n points within a 2x2
X = rand(n,2)*2-1;

%Compute the squared distance from the origin
D = dot(X',X');
Xin = X(D<=1,:);
```
Exercise

Write a function that normalizes a given vector.

```matlab
>> n_vec = normalized(vec);
```

In a Matlab script generate 2 random vectors and measure the angle between them.

**Hint:** the dot product between two normalized vectors is equal to the cosine of the angle between them.
Solution

>> edit normalized

function [ nvec ] = normalized( vec )
%NORMALIZED Returns the normalized vector
% NV = normalized(V) returns the vector V./norm(vec)
    nvec = vec./norm(vec);
end

>> edit measureangle

% generate two random vectors
x1 = rand(2,1);
x2 = rand(2,1);
% The smallest angle between two normalized vectors is
% equal to the arcocosine of the dot product between them
angle = acos(normalized(x1)'*normalized(x2))*180/pi

>> measureangle
angle =
    17.9611
Matlab programming language

- Elements of Matlab as a programming language:
  - Expressions
  - Flow Control blocks
    - Conditional
    - Iterations
  - Scripts
  - Functions
Expressions: Matlab Relational Operators

- MATLAB supports six relational operators.
  - Less Than <
  - Less Than or Equal ≤
  - Greater Than >
  - Greater Than or Equal ≥
  - Equal To ==
  - Not Equal To ~=
Expressions: Matlab Logical Operators

- MATLAB supports three logical operators.
  - not ~ % highest precedence
  - and & % equal precedence with or
  - or | % equal precedence with and
Expressions: Matlab Logical Functions

- MATLAB also supports some logical functions.

  any (x)    returns 1 if any element of x is nonzero
  all (x)    returns 1 if all elements of x are nonzero
  isnan (x)  returns 1 at each NaN in x
  isinf (x)  returns 1 at each infinity in x
  finite (x) returns 1 at each finite value in x
Matlab Conditional Structures

```matlab
if expression cond.
sentences
elseif expr. cond.
sentences
else
sentences
end
```

```matlab
a = input('valor1? '); 
b = input('valor2? '); 
if a == b,
    fprintf('a is equal to b\n');
elseif a > 0 && b > 0
    fprintf('both positive\n');
else
    fprintf('other case\n');
end
```
Matlab Iteration Structures (I)

```matlab
M = rand(10,10); suma = 0;
for i = {2,5:8}
    for j = {1:5,8:9}
        suma = suma + M(i,j);
    end
end
fprintf('sum = %d
',suma);
```

```matlab
M = rand(4,4); suma = 0;
for i = 1:4
    for j = 1:4
        suma = suma + M(i,j);
    end
end
fprintf('sum = %d
',suma);
```

```matlab
M = rand(4,4); suma = 0;
for i = 1:4
    for j = 1:4
        suma = suma + M(i,j);
    end
end
fprintf('sum = %d
',suma);
```

```matlab
M = rand(10,10); suma = 0;
for i = {2,5:8}
    for j = {1:5,8:9}
        suma = suma + M(i,j);
    end
end
fprintf('sum = %d
',suma);
```
Matlab Iteration Structures (II)

```
while expr
  sentence;
  ...
  sentence;
end

M = rand(4,4);
i = 1; j = 1; suma = 0;

while i <= 4
  while j <= 4
    suma = suma + M(i,j);
    j = j+1;
  end
  i = i+1;
end

fprintf('suma = %f\n',suma);
```
(Optimizing code: vectorization)

- Loops should be avoided when possible:

```matlab
for ind = 1:10000
    b(ind)=sin(ind/10)
end
```

Alternatives:

```matlab
x=0.1:0.1:1000;  
x=1:10000;  
b=sin(x);  
b=sin(x/10);
```

Most of the loops can be avoided!!!
Exercise

Given two matrices of n points in $\mathbb{R}^2$ compute the average distance between each pair.

Compare the execution time of two different implementation:
- using loops
- avoiding loops

Use the instructions `tic` and `toc` to measure the elapsed time.

```matlab
>> tic;pause(0.1);toc;
Elapsed time is 0.105363 seconds.
```
>> edit avoidloops

% generata 2 matrices of 100000 points
X1 = rand(2,1000000);
X2 = rand(2,1000000);

%loops version
tic
dists = zeros(1,1000000);
for i=1:1000000
dists(i)=sqrt( (X1(1,i)-X2(1,i))^2 ... 
+ (X1(2,i)-X2(2,i))^2);
end
fprintf('loop elapsed time: %fs\n',toc);

%matrices version
tic
dists = sqrt(sum((X1-X2).^2));
fprintf('matrices elapsed time: %fs\n',toc);

>> avoidloops
loop elapsed time: 1.300240s
matrices elapsed time: 0.026092s
Exercise: Fibonacci

• Write a function which compute and return a vector containing the first n numbers of the Fibonacci series.

```matlab
>> fib(10)
ans =
     1     1     2     3     5     8    13    21    34    55
```

• Write also a recursive implementation of the same function.
Plotting with Matlab

• Matlab has a lot of function for plotting data.
• The basic version requires two input vectors, one for the abscissae (x values) and one for the ordinates (y values).
• The vectors have to be the same length.

>> plot (time, dist) % plotting versus time

• We can give the plot function only the ordinates (y values). The vector indices are then considered as abscissae.

>> plot (dist) % plotting versus index

• To display multiple graphs at the same time we need to open a new window using the “figure“ command.
• The plot will be drawn in the last opened window.
time = 0:0.1:10;
dist = 0.5.*9.8.*time.^2;

% plot distance over time
plot(time,dist);

% open a new window
figure;
% plot distance over indices
plot(dist);
Plotting in Matlab

We can specify the line and the marker style with an additional parameter of `plot` function.

```matlab
>> dist = dist + (rand(size(dist)) - 0.5) * 20;
>> plot(time, dist, ':*r');
```

```matlab
x = rand(1,100);
y = rand(1,100);
plot(x, y, '*')
```
Plotting in Matlab

>> doc LineSpec

<table>
<thead>
<tr>
<th>Line Style Specifiers</th>
<th>Color Specifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifier</td>
<td>Color</td>
</tr>
<tr>
<td>-</td>
<td>Red</td>
</tr>
<tr>
<td>--</td>
<td>Green</td>
</tr>
<tr>
<td>:</td>
<td>Blue</td>
</tr>
<tr>
<td>- .</td>
<td>Cyan</td>
</tr>
<tr>
<td>m</td>
<td>Magenta</td>
</tr>
<tr>
<td>y</td>
<td>Yellow</td>
</tr>
<tr>
<td>k</td>
<td>Black</td>
</tr>
<tr>
<td>w</td>
<td>White</td>
</tr>
</tbody>
</table>

Marker Specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Marker Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Plus sign</td>
</tr>
<tr>
<td>o</td>
<td>Circle</td>
</tr>
<tr>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td>.</td>
<td>Point (see note below)</td>
</tr>
<tr>
<td>x</td>
<td>Cross</td>
</tr>
<tr>
<td>'square' or s</td>
<td>Square</td>
</tr>
<tr>
<td>'diamond' or d</td>
<td>Diamond</td>
</tr>
<tr>
<td>^</td>
<td>Upward-pointing triangle</td>
</tr>
<tr>
<td>v</td>
<td>Downward-pointing triangle</td>
</tr>
<tr>
<td>&gt;</td>
<td>Right-pointing triangle</td>
</tr>
<tr>
<td>&lt;</td>
<td>Left-pointing triangle</td>
</tr>
<tr>
<td>'pentagram' or p</td>
<td>Five-pointed star (pentagram)</td>
</tr>
<tr>
<td>'hexagram' or h</td>
<td>Six-pointed star (hexagram)</td>
</tr>
</tbody>
</table>
We can plot multiple functions on the same graph.

```matlab
X=0:0.01:2*pi;
cosx=cos(X);
sinx=sin(X);
plot(X,sinx,'--r', X,cosx,':b');
```
Plotting with Matlab

- There are commands in Matlab to "annotate" a plot to put on axis labels, titles, and legends. For example:

```matlab
% To put a title on the plot, we would use:
title('Title of my plot')
% To put a label on the axes we would use:
xlabel('X-axis label')
ylabel('Y-axis label')
% To add a legend we should use:
legend('fname1','fname2');
```
We can save the current plot to a file using:

```python
>> print -dpng 'filename'
```

We use different output formats, ie:
- `-dpng`: save to a png image file (Rasterized)
- `-dpdf`: save to a pdf file (Vectorized)
- `-dsvg`: save to svg (vectorized)
Exercise

Plot multiple sine functions over the time \((t)\) with different frequencies \((f)\): 

\[ y = \sin(f \cdot t) \]
Exercise: pi approximation

If we consider a circle with radius \( r \), we can approximate its area randomly sampling the rectangle in which the circle is inscribed.

The approximate area of the circle is equal to the product of the rectangle area and the probability of hitting the circle.

- Write a function which takes the number of samples as input and returns the approximate value of \( \pi \)
- How many samples do I need to obtain an accuracy to the third digit of \( \pi \)?
Solution

function p = approxpi(n)
%APROXPI Returns an approximation of Pi based on the statistical sampling of
%a circle inscribed on a rectangle area.
% P = approxpi(N) returns the value of Pi computed using N samples.

    %Generate n points within a 2x2 square centered in the origin
    X = rand(n,2)*2-1;

    %Compute the squared distance from the origin
    D = dot(X',X');
    Xin = X(D<=1,:);
    Xout = X(D >1,:);
    p = 4*size(Xin,1)/size(X,1);

plot(Xin(:,1),Xin(:,2),'.g',Xout(:,1),Xout(:,2),'.r')
axis equal
end