Evolutionary Algorithms

The workgroup sessions

General

The workgroups are provided by:

- Zhiwei Yang
  - z.yang@liacs.leideuniv.nl
  - Room 160
- Kaifeng Yang
  - k.yang@liacs.leidenuniv.nl
  - Room 160

Follow the webpage:

- http://natcomp.liacs.nl/EA

General (cont'd)

- The werkcolleges are there for you!
- The werkcolleges are devoted to:
  - providing you with theoretical exercise for exam
  - helping you with the practical assignment

- **Use these meetings to ask questions!!!**

Overview of the Workgroups

- Sep. 3: *No workgroup*
- Sep. 10: MATLAB, LAB session – MATLAB
- Sep. 17: Practical Assignment 1, LAB session – MATLAB (cont'd)
- Sep. 24: Workgroup session – Binary Mutations
- Oct. 1: *No workgroup*
- Oct. 8: LAB session – Genetic Algorithms in MATLAB
- Oct. 15: Workgroup session – Genetic Algorithms
- Oct. 22: LAB session – Genetic Algorithms in MATLAB (cont'd)
  - **Oct. 24: Deadline Practical Assignment 1**
- Oct. 29: *No workgroup*
- Nov. 5: Workgroup session – Evolution Strategies
- Nov. 12: LAB session – Evolution Strategies in MATLAB
- Nov. 19: LAB session – Evolution Strategies in MATLAB (cont'd)
- Nov. 26: LAB session – Evolution Strategies in MATLAB (cont'd)
  - **Nov. 28: Deadline Practical Assignment 2**
- Dec. 3: *No workgroup*

An introduction to MATLAB

What is MATLAB?

- Stands for **MATrix LABoratory**
- Developed by The Mathworks, Inc. http://www.mathworks.com
- An interactive, integrated environment
  - for numerical computations
  - for symbolic computations
  - for scientific visualizations

- It is a programming language!

Characteristics of MATLAB

- Programming language based on matrix notation
  - Slow (compared to Fortran or C) (it is an *interpreted language*)
  - Efficient when matrices are involved!
  - Automatic memory management
  - Intuitive, easy to use, and compact code
  - Shorter program development time
  - Can be converted into C code via MATLAB compiler
- Many application-specific toolboxes available

MATLAB Toolboxes

- Statistics
- Signal Processing
- Curve-Fitting
- Bioinformatics
- Financial
- Image Processing
- Neural Networks
- Databases
- …and many others!

MATLAB at LIACS

- Installed version: 7.14.0.739 (R2012a)

- Invoke at system prompt (under LINUX)
- 1. run terminal
  2. type: pushd /vol/share/software/matlab/liacs/r2012b/bin; ./matlab;

Getting started (1)

Row vector:

```
>> x = [3,6,9]
   x =
                3       6       9
```

or…

```
>> x = [3 6 9]
   x =
                3       6       9
```

Column vector:

```
>> x = [3; 6; 9]
   x =
          3
          6
          9
```

Getting started (2)

A row vector of ones:

```
>> x = ones(1,3)
   x =
                1       1       1
```

A column vector of ones:

```
>> x = ones(3,1)
   x =
                1
                1
                1
```

A row vector of zeros:

```
>> x = zeros(1,3)
   x =
                0       0       0
```

A row vector of random numbers between 0 and 1:

```
>> x = rand(1,3)
   x =
                0.9501          0.2311          0.6068
```

Inner product / outer product

```
>> x = [3,6,9]                          % a row vector
   x =
                3       6       9

>> y = [1;2;3]                          % a column vector
   y =
                1
                2
                3

>> x*y
  ans =
        42

>> y*x
   ans =
        3        6        9
        6       12       18
        9       18       27
```

Element-wise operations

Use the '.' character for element-wise operations:

Element-wise division:

```
>> x = [3,6,9]
>> y = [3,3,3]
>> x ./ y
   ans =
                1       2       3
```

Element-wise multiplication:

```
>> x = [3,6,9]
>> y = [3,3,3]
>> x .* y
   ans =
                9      18      27
```

And then matrices

Use semicolons to separate the rows:

```
>> A = [1 2 0; 2 5 -1; 4 10 -1]
   A =
                1      2      0
                2      5     -1
                4     10     -1
```

And the transpose of ***A***:

```
>> B = A'
   B =
                1      2      4
                2      5     10
                0     -1     -1
```

MATLAB and matrices

- MATLAB does not require you to deal with matrices as an array of numbers
- MATLAB knows when you are dealing with matrices and adjusts your calculations accordingly

- Matrix multiplication:

```
>> C = A*B;
```

- Element-wise multiplication:

```
>> D = A.*B;
```

- Inverse matrix:

```
>> E = inv(A);
```

- Eigenvalues:

```
>> e = eig(A);
```

Random numbers

Efficient pseudorandom number generator:

```
A = rand(4,4); % Uniform[0,1] 4x4 matrix
B = 2*rand(2,2) - 1; % Uniform[-1,1] 2x2 matrix
C = randn(1,10); % Normal(0,1) dist 1x10 matrix
```

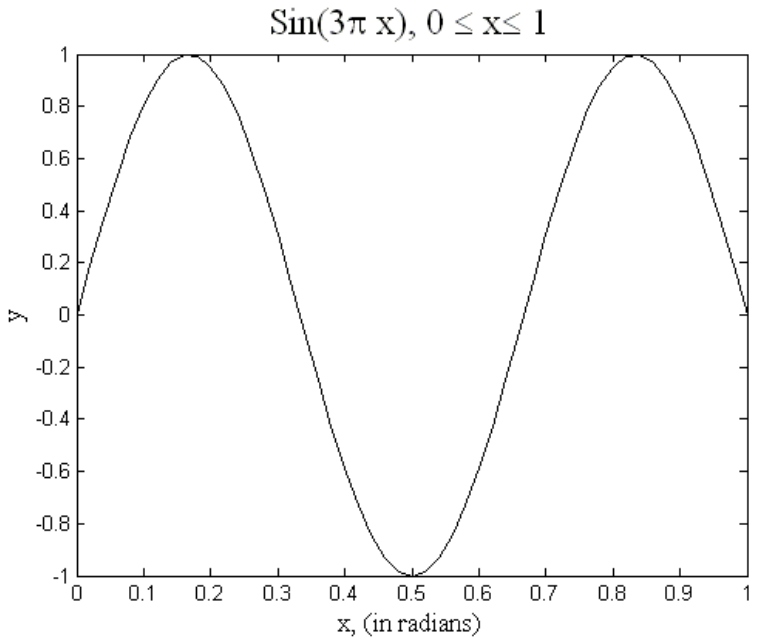
Example: generating a bitstring of length 15:

```
D = rand(15,1) > 0.5;
```

Basic plots

Plot of $f(x) = \sin(3\pi x)$ using 51 points on the interval [0,1]:

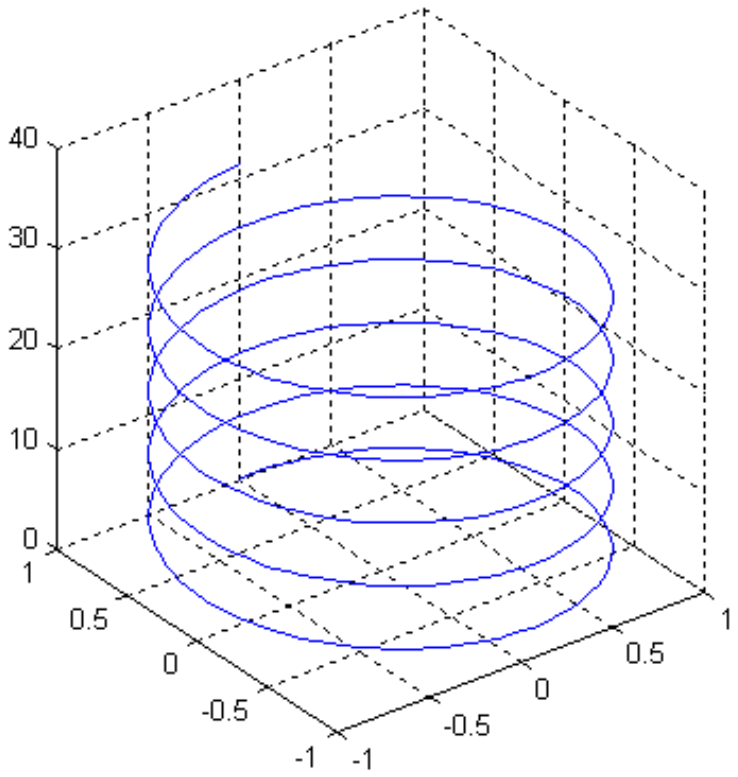
```
>> x=[0:1/50:1];
>> y=sin(3*pi*x);
>> plot(x,y,'k-');
```



You can fancy up the plot with a title, axis labels, a legend, etc.

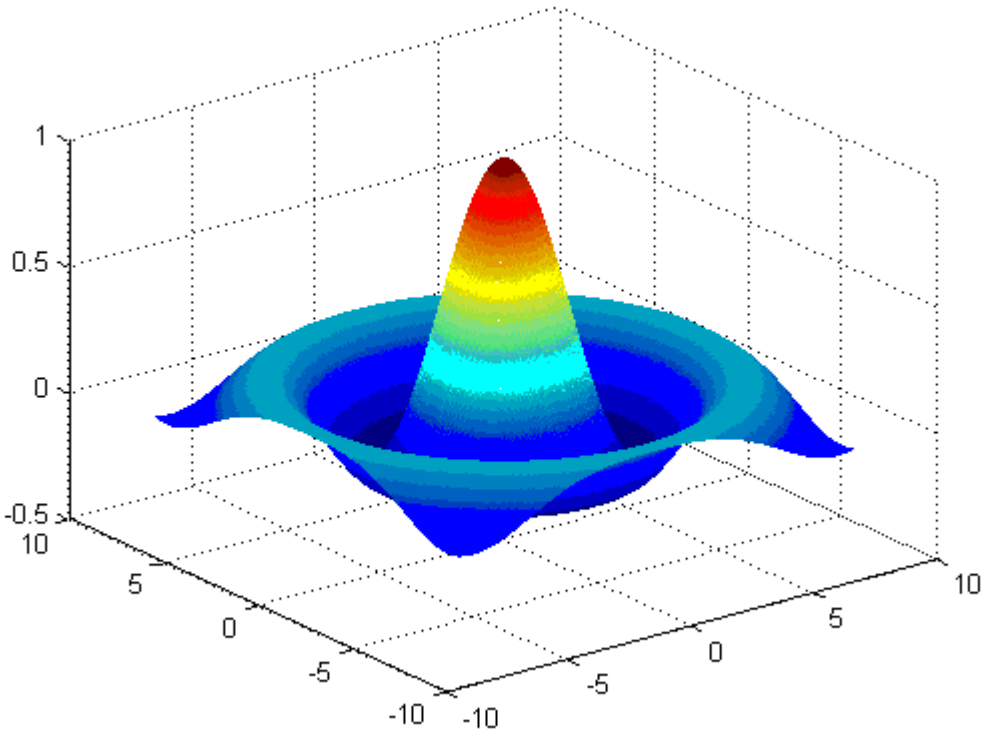
3D Line Plots

```
>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t);
>> grid on
>> axis square
```



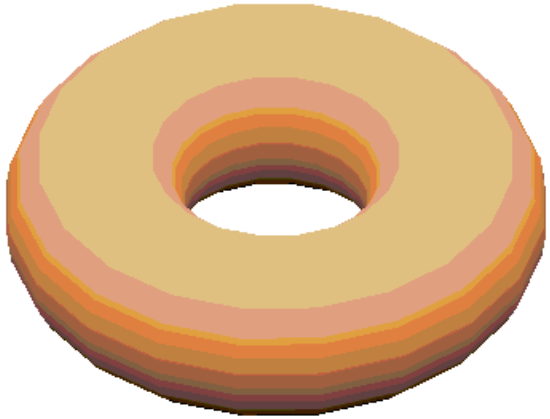
3D Graphics: **mesh**

```
>> [X,Y] = meshgrid(-8:0.05:8);
>> R = sqrt(X.^2 + Y.^2) + eps;
>> Z = sin(R)./R;
>> mesh(X,Y,Z);
```



3D Graphics: **surf**

```
>> [s,t] = meshgrid(0:2*pi/20:2*pi,0:2*pi/20:2*pi);
>> x = (2 + cos(s)) .* cos(t);
>> y = (2 + cos(s)) .* sin(t);
>> z = sin(s);
>> surf(x,y,z),shading('interp'),colormap(copper);
>> axis('equal'),axis('off'),view([-10,40])
```



Creating your own functions

You can create your own MATLAB functions by creating a file ‘function_name.m’. A simple example, traparea.m:

```
function area = traparea(a,b,h)
% traparea(a,b,h)    Computes the area of a trapezoid
%                    given the dimensions a, b and h,
%                    where a and b are the lengths of
%                    the parallel sides and h is the
%                    distance between these sides
      area = 0.5 * (a + b) * h;
end
```

Typing
>> help traparea
will display this section

Conditional statements

It is just as you would expect:

```
if x > 2 && x < 5
   x = x + 5;
elseif x >= 5
   x = x - 1;
else
   x = x - 10;
end
```

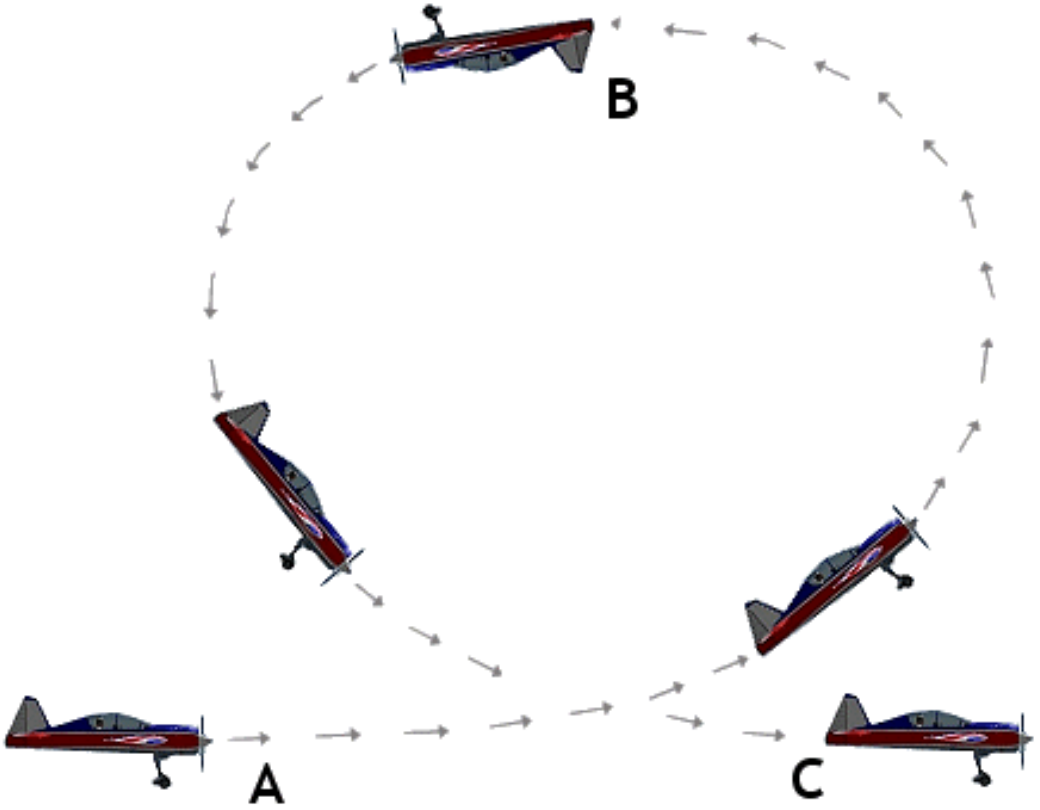
Loops

For loops:

```
for i = 1:10
   x(i) = i^2;
end
```

While loops:

```
i = 1;
while i < 10
   x(i) = i^2;
   i = i + 1;
end
```



Function handles

- Pass function as argument to other functions
- Useful in programming optimizer algorithms: one optimizer for different fitness functions
- Create a function handle by adding ‘@’ symbol in front of the function name

Function handle example

```
function value = example(func_handle, x)
    value = func_handle(x);
end
```

A function to be calculated, for instance **`sin`**, can be passed as follows:

```
>> a = [0:0.1:10]
>> example(@sin, a)
```

Or like this:

```
>> func_handle = @(x) sin(x)
>> example(func_handle, a)
```

MATLAB efficiency

- User-defined MATLAB functions are *interpreted*
- For this reason MATLAB programs can be much slower than programs written in a compiled language such as Fortran or C
- Use **built-in functions and operators** whenever possible, executing compiled rather than interpreted code!
- Furthermore, use matrix operations instead of loops
  - Built-in functions operating on matrices
  - Element-wise operations

Matrix operations versus loops

Compare:

```
>> dx = pi/30;
>> nx = 1 + 2*pi/dx;
>> for i = 1:nx
     x(i) = (i-1)*dx;
     y(i) = sin(3*x(i));
   end
```

To:

```
>> x = 0:pi/30:2*pi;
>> y = sin(3*x);
```

Efficiency continued

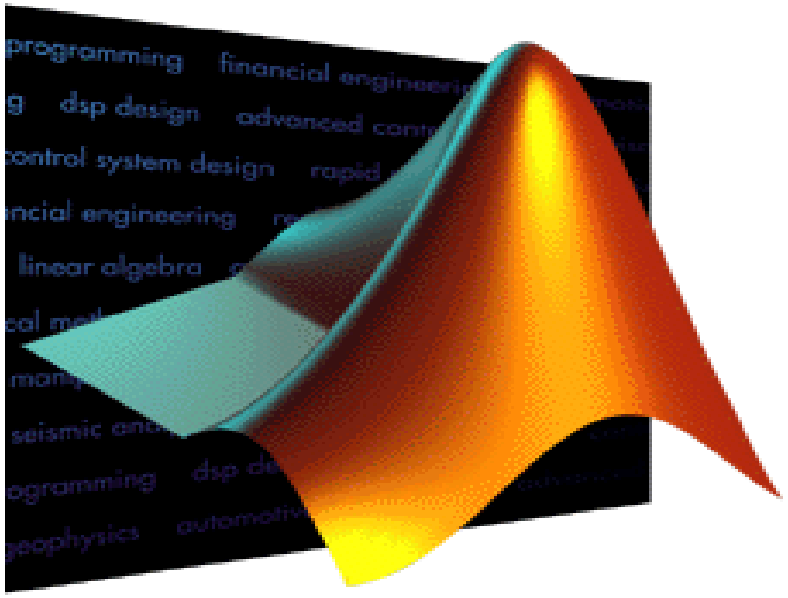
- Bottom line – try to work as much as possible with matrix notation!

- Examine the time execution of your code:

```
>> Tic;
   -- Operation --
>> Toc;
```

MATLAB HELP!

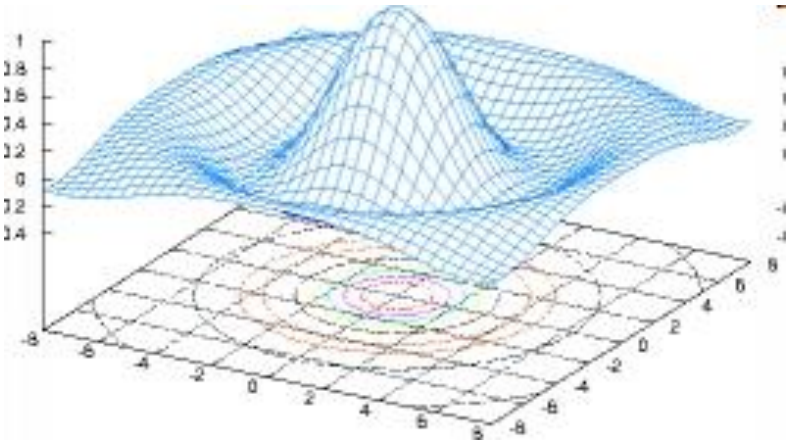
The most important feature in MATLAB…
the **`help`** command!



Great documentation, demos etc.! Use it!

Octave → the free MATLAB alternative

- GNU Octave is an open source tool that is largely compatible with MATLAB code.
- It is not as fast as MATLAB, nor does it provide the nice MATLAB look-and-feel, but it is free!
- Website: http://www.gnu.org/software/octave/



Optimization with MATLAB

Binary Monte Carlo Search

Binary Monte Carlo Search

Our problem:

- Search space: $\{0,1\}^n$
- An objective function, to be maximized: $f(a) \rightarrow \max$

The algorithm:

Input: objective function ***f***, bitstring length ***n***, the number of iterations ***iters***
Output: (sub)optimal bitstring ***aopt***

```
1. aopt := generate random bitstring of length n
2. for i = 2 to iters do
3.    a := generate random bitstring of length n
4.    if f(a) >= f(aopt)
5.      aopt := a
6.    end
7. end
8. return aopt
```

From pseudo-code to MATLAB

Input: objective function *f*, bitstring length *n*, the number of iterations *iters*
Output: (sub)optimal bitstring *aopt*

```
1. aopt := generate random bitstring of length n
2. for i = 2 to iters do
3.   a := generate random bitstring of length n
4.   if f(a) >= f(aopt)
5.     aopt := a
6.   end
7. end
8. return aopt
```

```
function aopt = binary_monte_carlo_optimization(f, n, iters)
  aopt = rand(n,1) > 0.5;
  for i = 2:iters
    a = rand(n,1) > 0.5;
    if (f(a) >= f(aopt))
      aopt = a;
    end
  end
end
```

Readability: Add comments for Help

```
function aopt = binary_monte_carlo_optimization(f, n, iters)
% function aopt = binary_monte_carlo_optimization(f, n, iters)
%
% Performs a binary Monte Carlo search. Given objective f, bitstring
% length n, and number of iterations, this algorithm will try
% to find the bitstring that maximizes f.
%
% Author: Johannes Kruisselbrink
% Last modified: September 6, 2010
  aopt = rand(n,1) > 0.5;
  for i = 2:iters
    a = rand(n,1) > 0.5;
    if (f(a) >= f(aopt))
      aopt = a;
    end
  end
end
```

Displayed upon calling

```
help binary_monte_carlo_optimization
```

Store fitness function value

- Function evaluations are costly; do not waste them!

```
function aopt = binary_monte_carlo_optimization(f, n, iters)
  aopt = rand(n,1) > 0.5;
  fopt = f(aopt);
  for i = 2:iters
    a = rand(n,1) > 0.5;
    fa = f(a);
    if (fa >= fopt)
      aopt = a;
      fopt = fa;
    end
  end
end
```

And so we save half of our objective function evaluations!!!

Statistics maintenance

You may want to see more than just the optimal bitstring

```
function [aopt, histf ] = binary_monte_carlo_optimization(f, n, iters)
  aopt = rand(n,1) > 0.5;
  fopt = f(aopt);
  histf(1) = fopt;
  for i = 2:iters
    a = rand(n,1) > 0.5;
    fa = f(a);
    if (fa >= fopt)
      aopt = a;
      fopt = fa;
    end
    histf(i) = fopt;
  end
end
```

Plotting statistics on-the-fly

```
function [aopt, histf] = binary_monte_carlo_optimization(f, n, iters)
  aopt = rand(n,1) > 0.5;
  fopt = f(aopt);
  histf(1) = fopt;
  for i = 2:iters
    a = rand(n,1) > 0.5;
    fa = f(a);
    if (fa >= fopt)
      aopt = a;
      fopt = fa;
    end
    histf(i) = fopt;
    plot(histf)
    drawnow()
  end
end
```

Use *plot* to plot, and *drawnow* to force MATLAB to do it right away (and not after loop termination)

Initialize vectors beforehand

```
function [aopt, histf] = binary_monte_carlo_optimization(f, n, iters)
  histf = zeros(1,iters);
  aopt = rand(n,1) > 0.5;
  fopt = f(aopt);
  histf(1) = fopt;
  for i = 2:iters
    a = rand(n,1) > 0.5;
    fa = f(a);
    if (fa >= fopt)
      aopt = a;
      fopt = fa;
    end
    histf(i) = fopt;
    plot(histf)
    drawnow()
  end
end
```

Initialize vectors beforehand if possible. It can save tremendous amounts of time!
(Otherwise, MATLAB will re-allocate space every time you increase their size)