UNIVERSITÀ DEGLI STUDI
DI MILANO

Genetic Algorithms Using Matlab

Ruggero Donida Labati

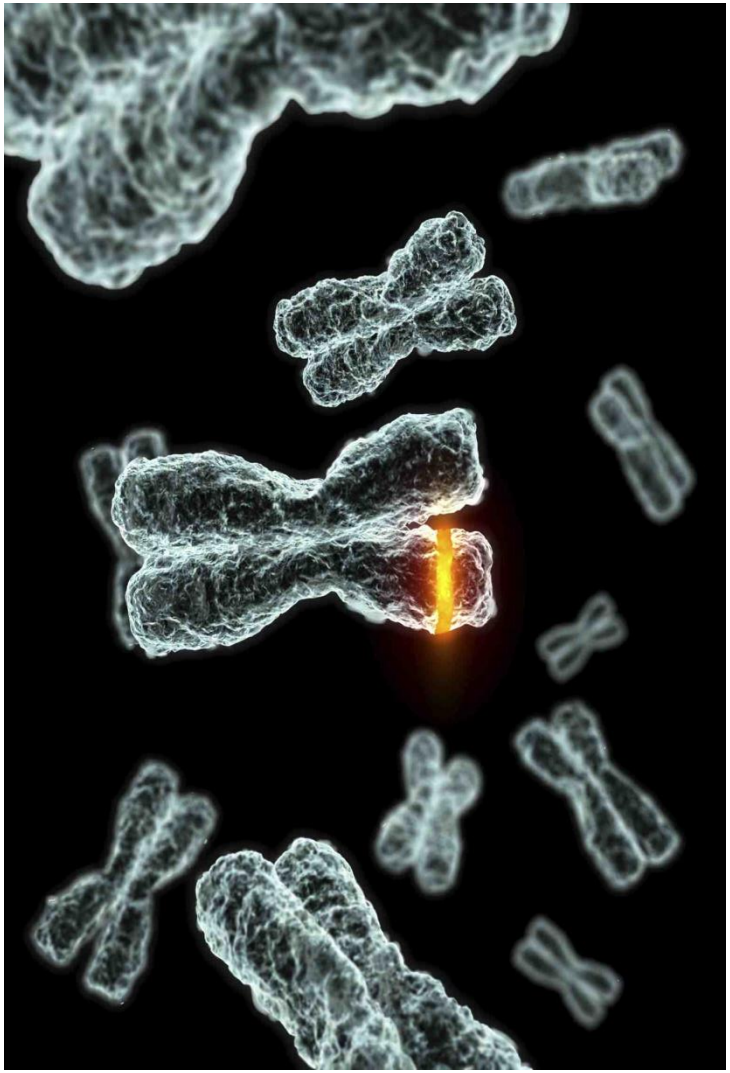
Dipartimento di Tecnologie dell'Informazione
via Bramante 65, 26013 Crema (CR), Italy
ruggero.donida@unimi.it

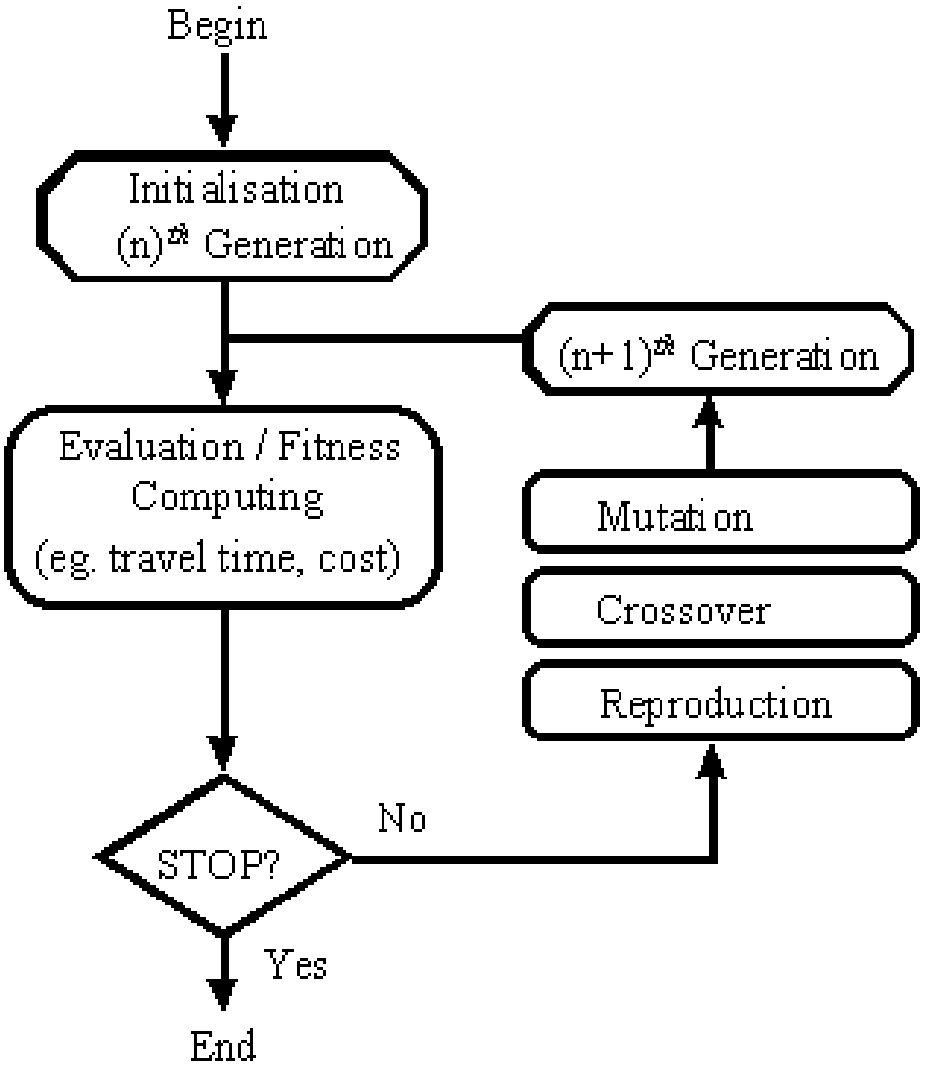
Why?

Genetic algorithms can be used to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear.

Classical Algorithm	Genetic Algorithm
Generates a single point at each iteration. The sequence of points approaches an optimal solution.	Generates a population of points at each iteration. The best point in the population approaches an optimal solution.
Selects the next point in the sequence by a deterministic computation.	Selects the next population by computation which uses random number generators.

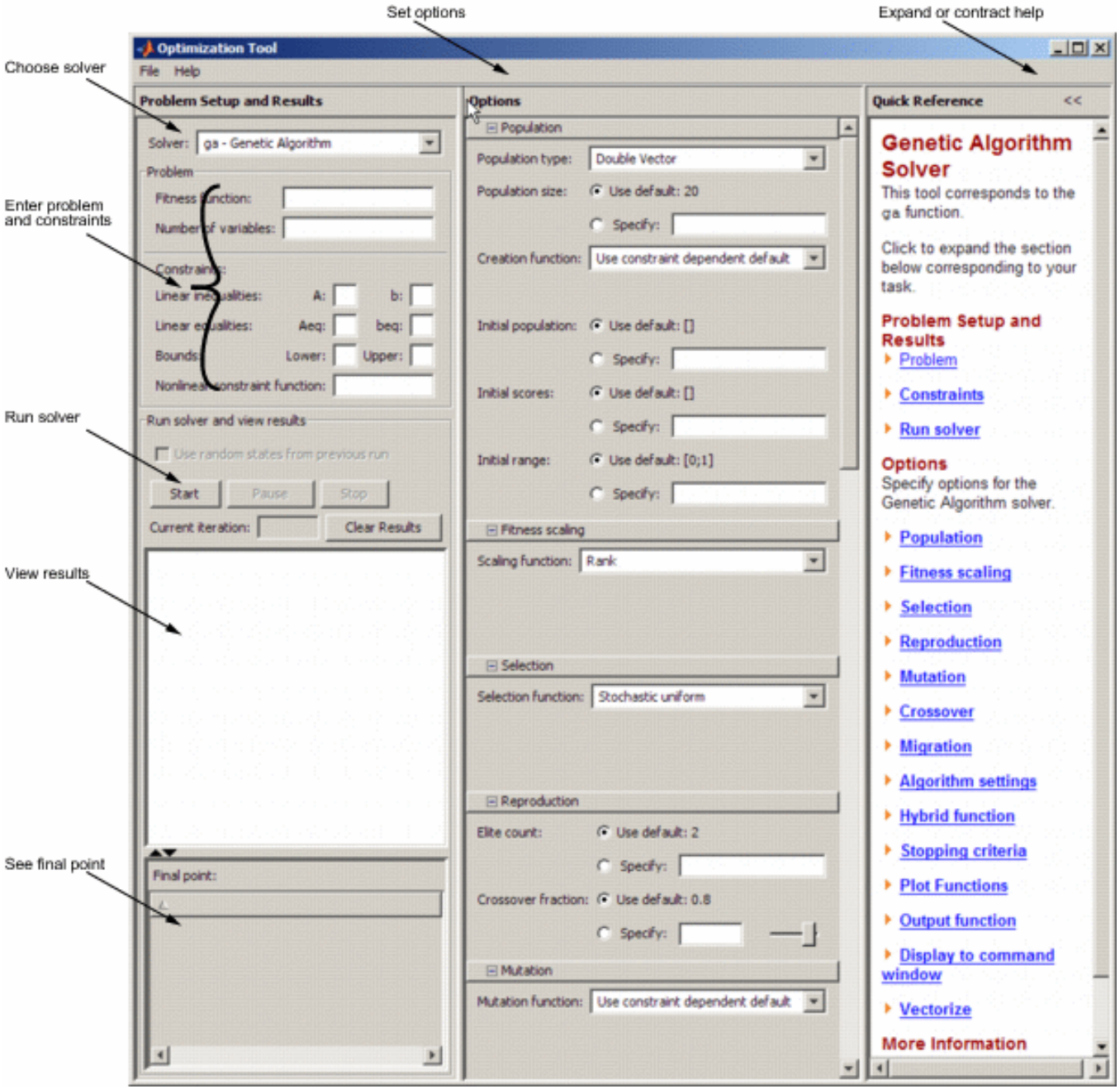
The schema





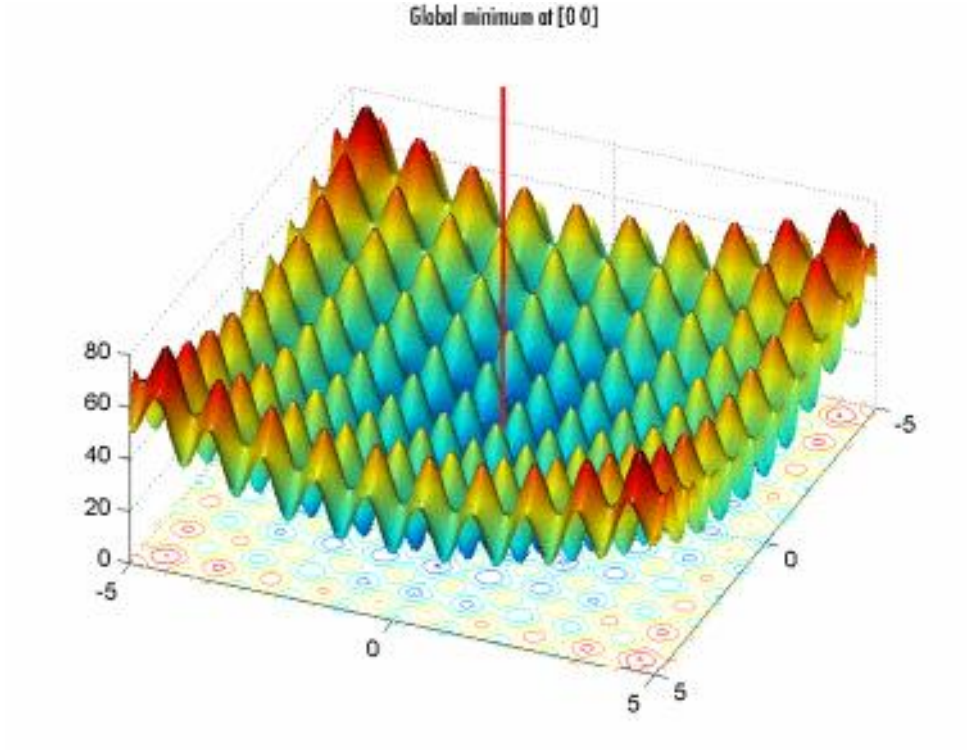
Matlab

- Global Optimization Toolbox™
- Calling the Function ga at the Command Line
  - [x fval] = ga(@fitnessfun, nvars, options)
- Using the Optimization Tool
  - optimtool('ga')



Example — Rastrigin's Function

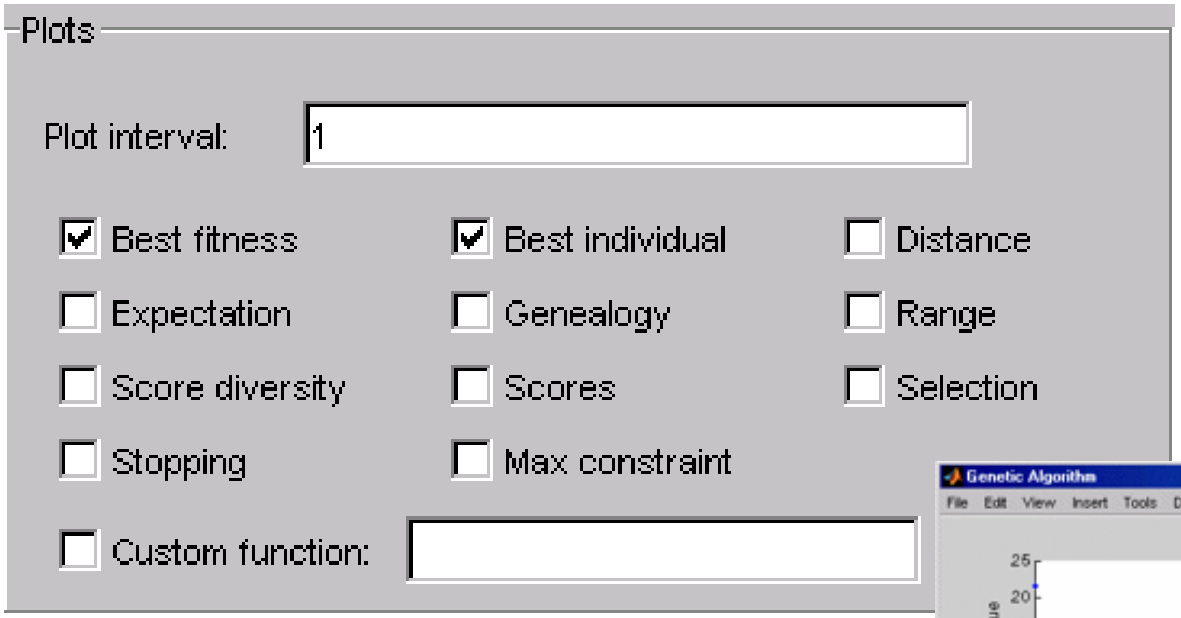
- For two independent variables, Rastrigin's function is defined as $Ras(x) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$.

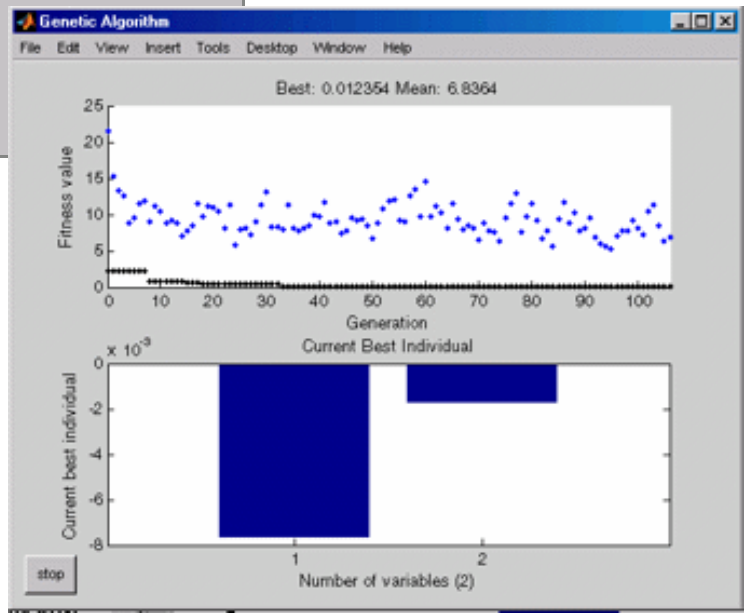


- Global minimum?
- Tehoretical:
  - x = 0;
  - y = 0;
  - z = 0;

- Fitness function = @rastriginsfcn
- Number of variables = 2

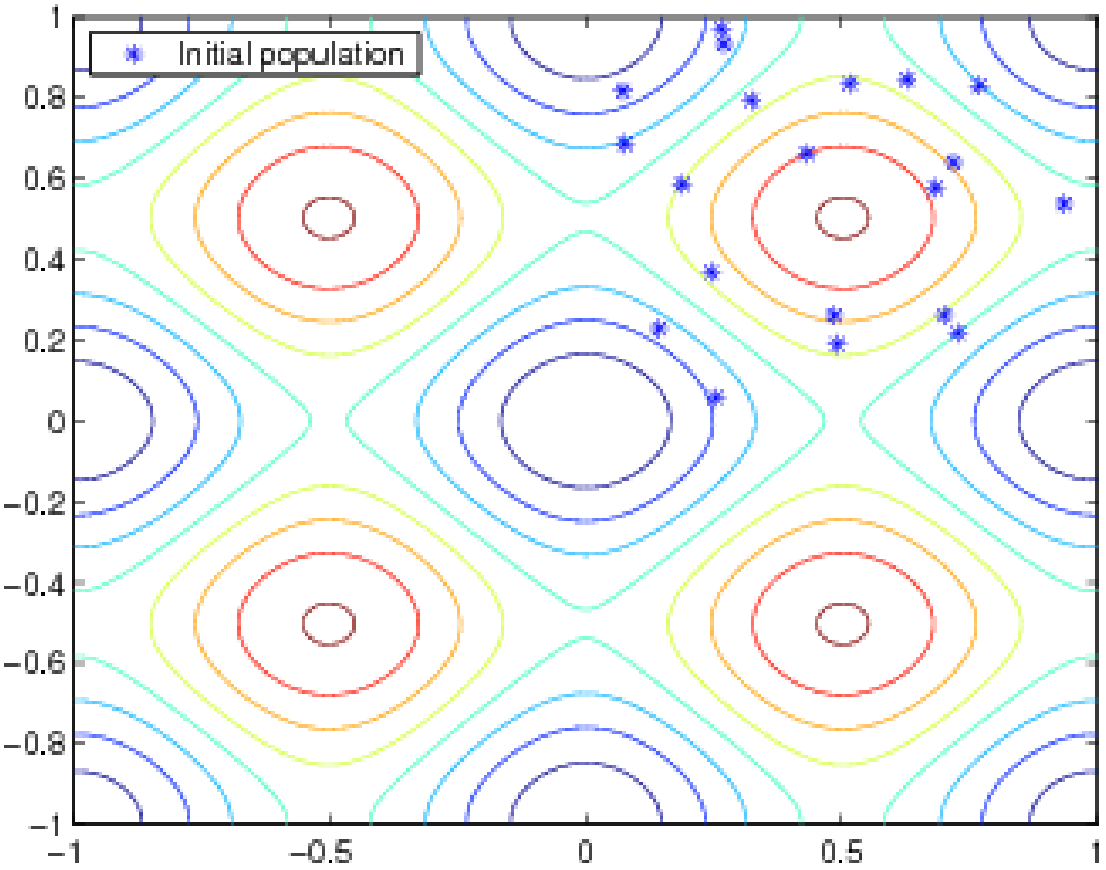
Displaying Plots





Initial Population

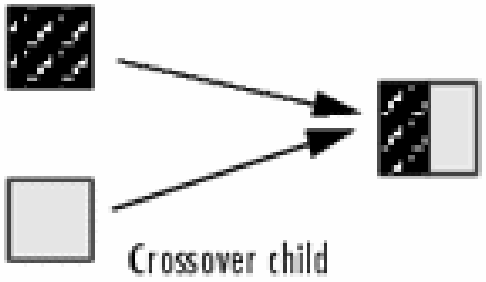
- Population size
  - Intial = 20
- Range
  - Initial = [0;1]



Creating the Next Generation

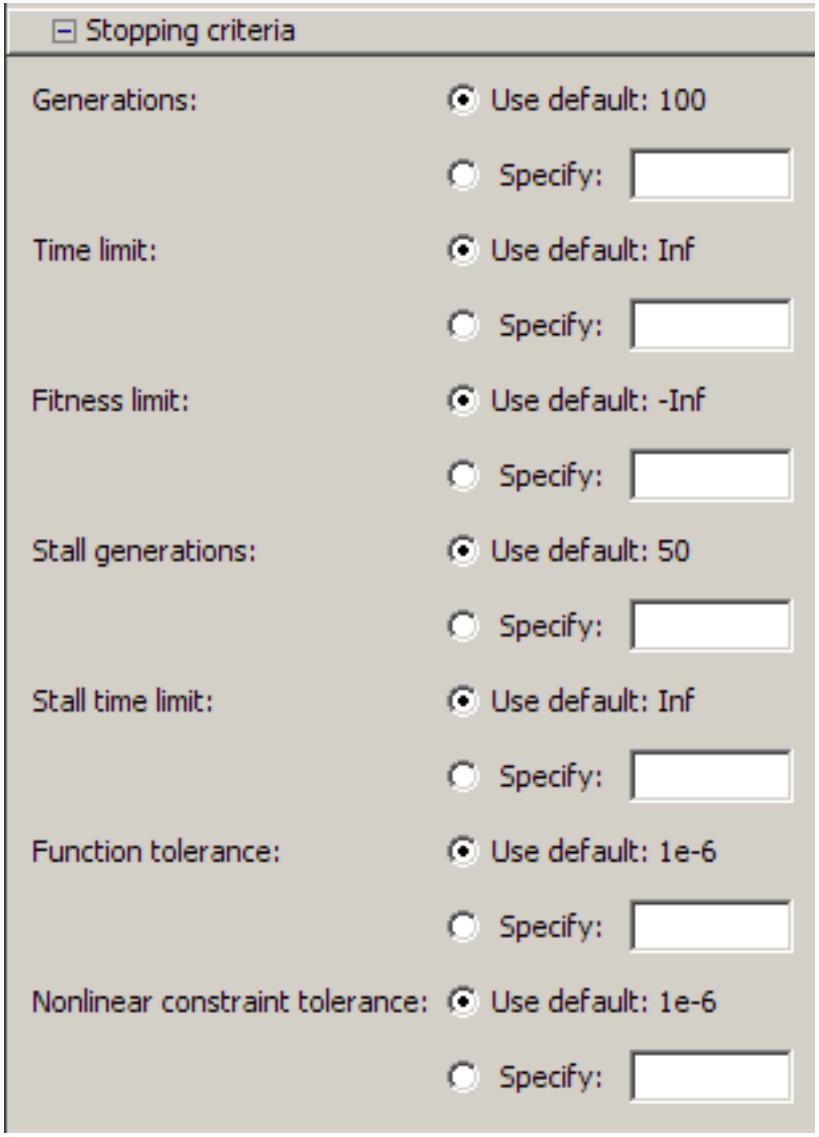
- The genetic algorithm creates three types of children for the next generation:
  - *Elite children*

  - *Crossover children*



  - *Mutation children*

Stopping Conditions for the Algorithm 1/2



Stopping Conditions for the Algorithm 2/2

- Generations — The algorithm stops when the number of generations reaches the value of Generations.
- Time limit — The algorithm stops after running for an amount of time in seconds equal to Time limit.
- Fitness limit — The algorithm stops when the value of the fitness function for the best point in the current population is less than or equal to Fitness limit.
- Stall generations — The algorithm stops when the weighted average change in the fitness function value over Stall generations is less than Function tolerance.
- Stall time limit — The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to Stall time limit.
- Function Tolerance — The algorithm runs until the weighted average change in the fitness function value over Stall generations is less than Function tolerance.
- Nonlinear constraint tolerance — The Nonlinear constraint tolerance is not used as stopping criterion. It is used to determine the feasibility with respect to nonlinear cnstraints.

Creating the Custom Plot Function

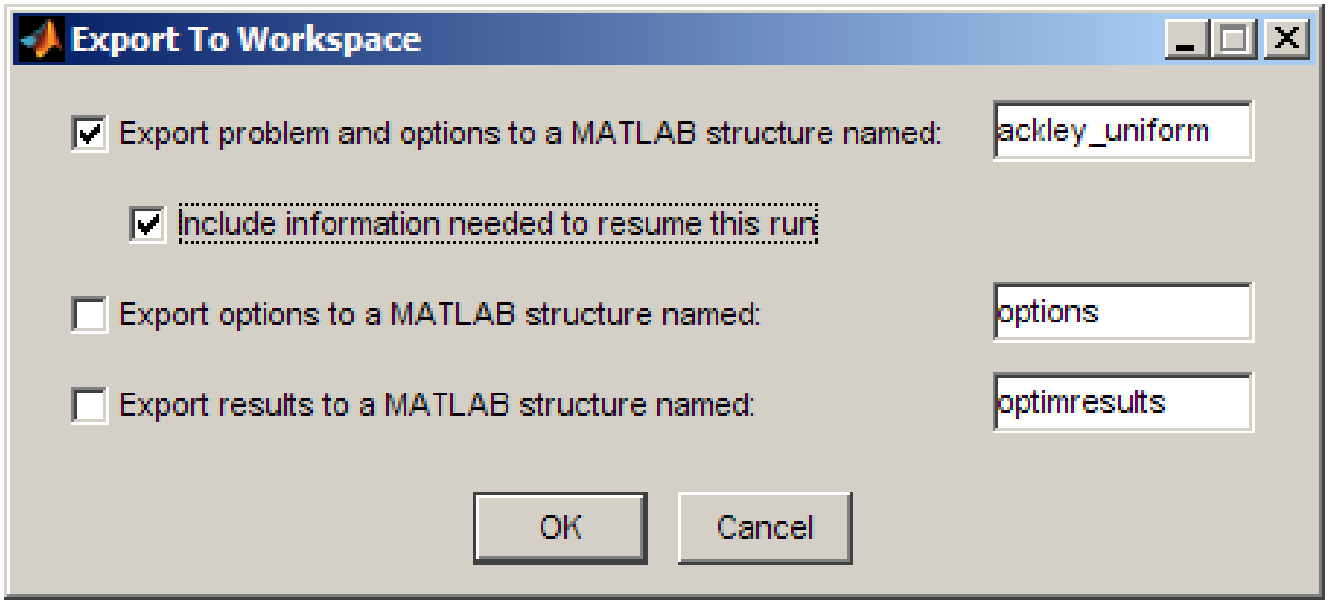
```
function state = gaplotchange(options, state, flag)
%  GAPLOTCHANGE Plots the logarithmic change in the best score from the previous
% generation.

persistent last_best % Best score in the previous generation
if(strcmp(flag,'init')) % Set up the plot
   set(gca,'xlim',[1,options.Generations],'Yscale','log');
   hold on;
   xlabel Generation
   title('Change in Best Fitness Value')
end
best = min(state.Score); % Best score in the current generation
if state.Generation == 0 % Set last_best to best.
   last_best = best;
else
     change = last_best - best; % Change in best score
     last_best=best;
     plot(state.Generation, change, '.r');
     title(['Change in Best Fitness Value'])
end
```

@gaplotchange

Resuming the Genetic Algorithm from the Final Population 1/2

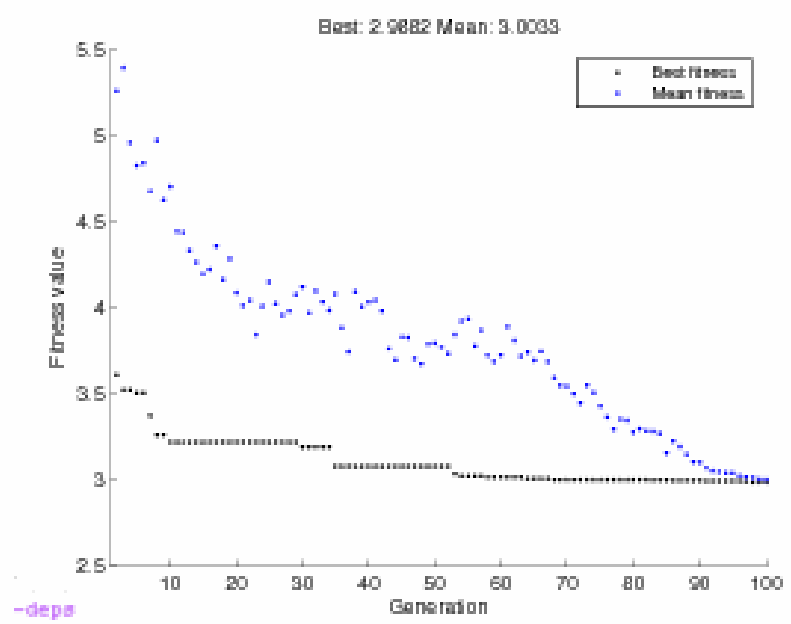
- **Export to Workspace** from the **File** menu



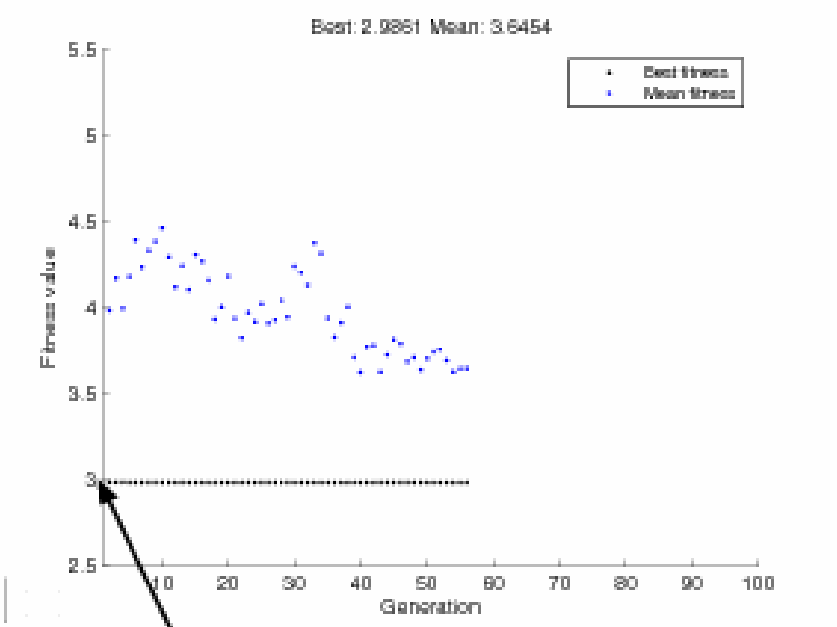
- **Import Problem** from the **File** menu

Resuming the Genetic Algorithm from the Final Population 2/2

- New test…



First run



Run resumes here

Using the Genetic Algorithm from the Command Line

- [x fval] = ga(@fitnessfun, nvars)
  - *@fitnessfun* — A function handle to the M-file that computes the fitness function.
  - *nvars* — The number of independent variables for the fitness function.
  - *x* — The final point
  - *fval* — The value of the fitness function at x

- [x fval exitflag output population scores] = ga(@fitnessfcn, nvars)
  - *exitflag* — Integer value corresponding to the reason the algorithm terminated
  - *output* — Structure containing information about the performance of the algorithm at each generation
  - *population* — Final population
  - *scores* — Final scores

Running ga from an M-File

```
options = gaoptimset('Generations',300);
rand('twister', 71); % These two commands are only included to
randn('state', 59); % make the results reproducible.
record=[];
for n=0:.05:1
    options = gaoptimset(options,'CrossoverFraction', n);
    [x fval]=ga(@rastriginsfcn, 10,[],[],[],[],[],[],[],options);
    record = [record; fval];
end

plot(0:.05:1, record);
xlabel('Crossover Fraction');
ylabel('fval')
```

Example: Coding and Minimizing a Fitness Function 1/2

Here we want to minimize a simple function of two variables

$$\min_{x} f(x) = 100 * (x(1)^2 - x(2))^2 + (1 - x(1))^2;$$

Example: Coding and Minimizing a Fitness Function 2/2

- Fitness function
  - function y = simple_fitness(x)

    y = 100 * (x(1)^2 - x(2)) ^2 + (1 - x(1))^2;

- Parametrized Fitness function
  - function y = parameterized_fitness(x,a,b)

    y = a * (x(1)^2 - x(2)) ^2 + (b - x(1))^2;

- FitnessFunction = @(x)parameterized_fitness(x,a,b)

Constrained Minimization Problem 1/2

We want to minimize a simple fitness function of two variables x1 and x2

$$\min_x f(x) = 100 * (x1^2 - x2)^2 + (1 - x1)^2;$$

such that the following two nonlinear constraints and bounds are satisfied

$x1*x2 + x1 - x2 + 1.5 <= 0$, (nonlinear constraint)

$10 - x1*x2 <= 0$, (nonlinear constraint)

$0 <= x1 <= 1$, and (bound)

$0 <= x2 <= 13$ (bound)

Constrained Minimization Problem 2/2

- Fitness function
  - function y = simple_fitness(x)

    y = 100 * (x(1)^2 - x(2)) ^2 + (1 - x(1))^2;

- Constraint function
  - function [c, ceq] = simple_constraint(x)

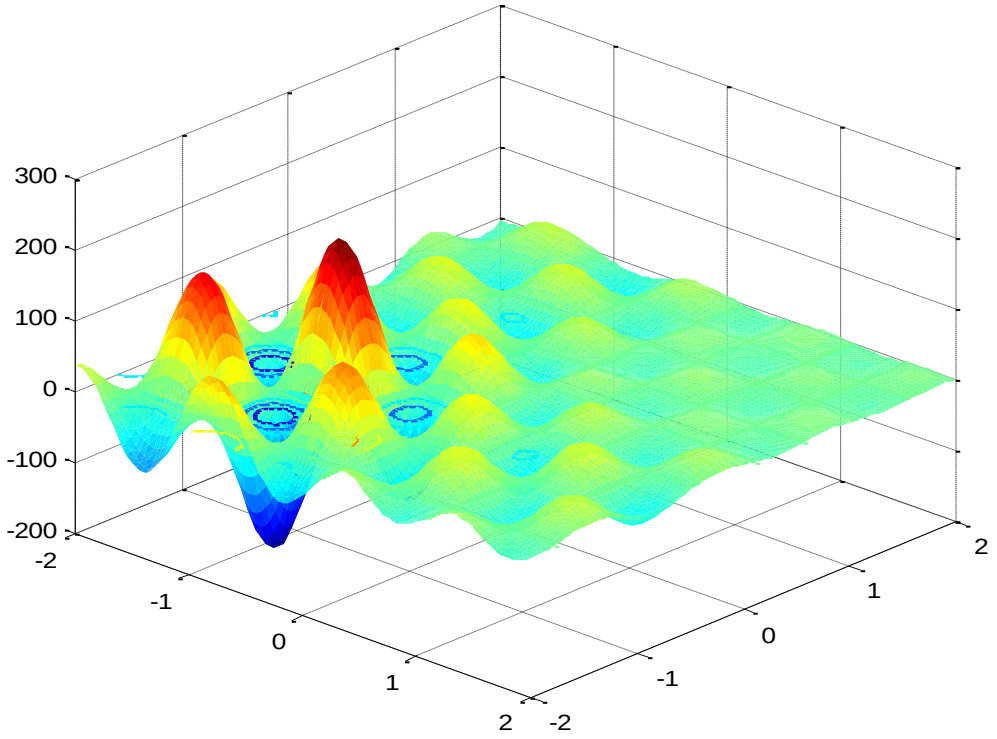
    c = [1.5 + x(1)*x(2) + x(1) - x(2); -x(1)*x(2) + 10];

    ceq = [];

- Bounds
  - Lower = [0 0]
  - Upper = [1 13]

Exercise 1

- What is the minimum of the function *shufcn* in [-2,2; -2,2]?



figure, plotobjective(@shufcn,[-2 2; -2 2]);

Exercise 2

- We want to minimize a fitness function of two variables x1 and x2

  min f(x) = a * (x1^3 - x2)^2 + b - x1;

  x

  such that the following two nonlinear constraints and bounds are satisfied

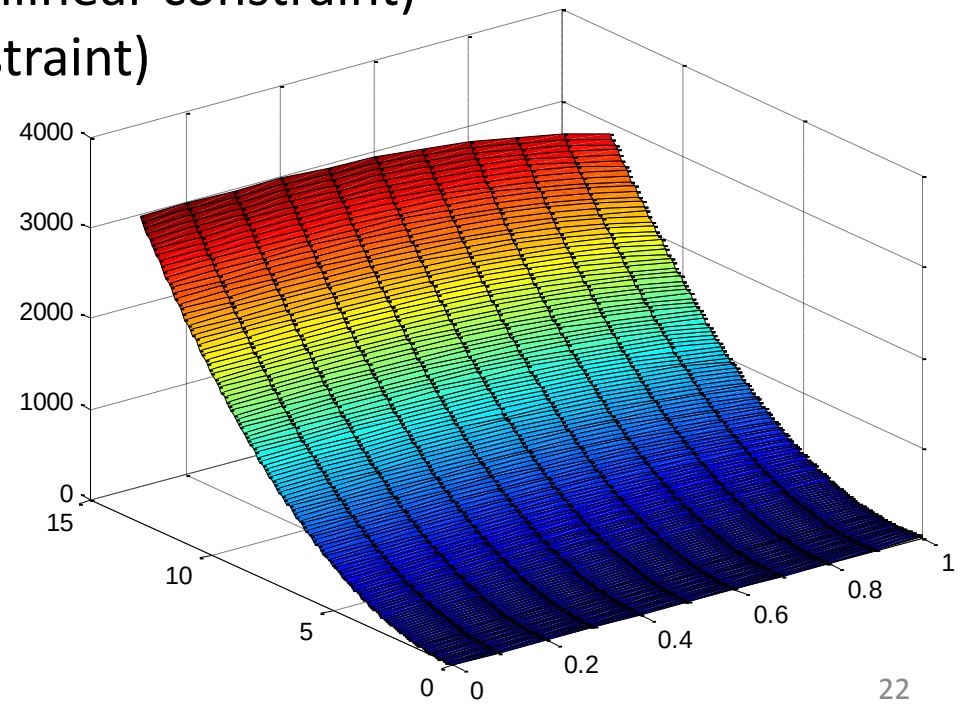
  x1*x2 + x1 - x2 + 0.7 <=0, (nonlinear constraint)

  8 - x1*x2 <=0, (nonlinear constraint)

  0 <= x1 <= 1, and (bound)

  0 <= x2 <= 13 (bound)

- a = 20 and b = 3
- a = 5 and b = 2



- *plot_ex2.m*

Multiobjective Optimization Using the GA (1/2)

- GAMULTIOBJ can be used to solve multiobjective optimization problem in several variables. Here we want to minimize two objectives, each having one decision variable.

min F(x) = [objective1(x); objective2(x)]
x

where,
objective1(x) = (x+2)^2 - 10, and
objective2(x) = (x-2)^2 + 20

Multiobjective Optimization Using the GA (2/2)

- Fitness function
  - function y = simple_multiobjective(x)
    y(1) = (x+2)^2 - 10;
    y(2) = (x-2)^2 + 20;

- Multiobjective Optimization

```
% Plot two objective functions on the same axis
x = -10:0.5:10;
f1 = (x+2).^2 - 10;
f2 = (x-2).^2 + 20;
plot(x,f1);
hold on;
plot(x,f2,'r');
grid on;
title('Plot of objectives ''(x+2)^2 - 10'' and ''(x-2)^2 + 20''');

FitnessFunction = @simple_multiobjective;
numberOfVariables = 1;
[x,fval] = gamultiobj(FitnessFunction,numberOfVariables);

fprintf('\nRESULTS\n');
fprintf('x\t\t\tfval\n');
for i = 1 : size(x,1)
   fprintf([num2str(x(i)), '\t\t\t', num2str(fval(i)), '\n']);
end
```

Custom Data Type Optimization Using the Genetic Algorithm

- Traveling Salesman Problem