

Optimization Methods for Mechanical Design

Lecturer:

Giovanni Berselli

giovanni.berselli@unige.it

Assistant Instructor:

Pietro Bilancia

pietro.bilancia@edu.unige.it

DIME – Via All’Opera Pia 15/A - Genova



A different approach to design

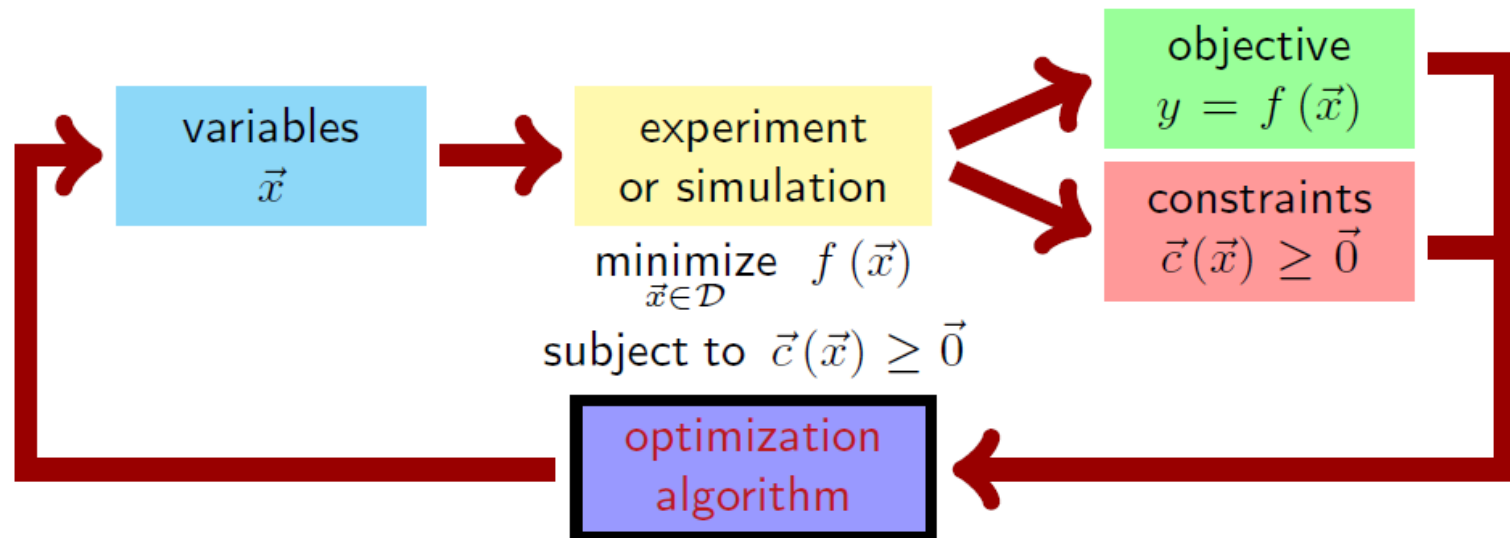
company know-how and
designer experience



systematic design by means
of **optimization** techniques

A definition

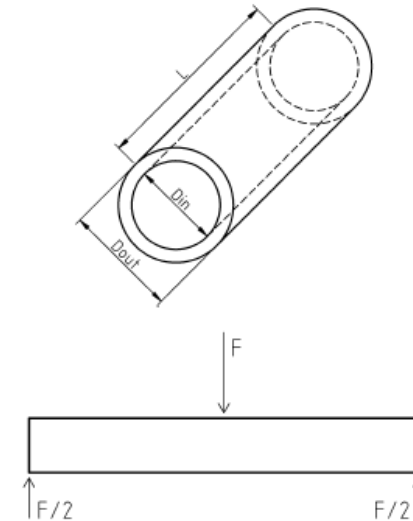
In an optimization **problem** we seek values of the **variables** that lead to an optimal value of the **function** that is to be optimized



- **Optimization Problem:** the object of the optimization
- **Input Parameters:** set of parameters characterizing the problem
- **Input Variables:** subset of the input parameters addressed in the optimization problem. A range is commonly associated to each variable
- **Sample:** a set of input variables
- **Design space** or **Domain:** space of all the possible samples
- **Experiment** or **Simulation:** process through which information about a sample can be extracted
- **Output Parameters:** the set of information extracted from a simulation
- **Objective Function:** the performance measure to be minimized or maximized, function of the input and/or output parameters
- **Constraints:** set of functions of the input and/or output parameters that the optimum solution must satisfy
- **Solutions space:** Co-domain of the design space

An Example

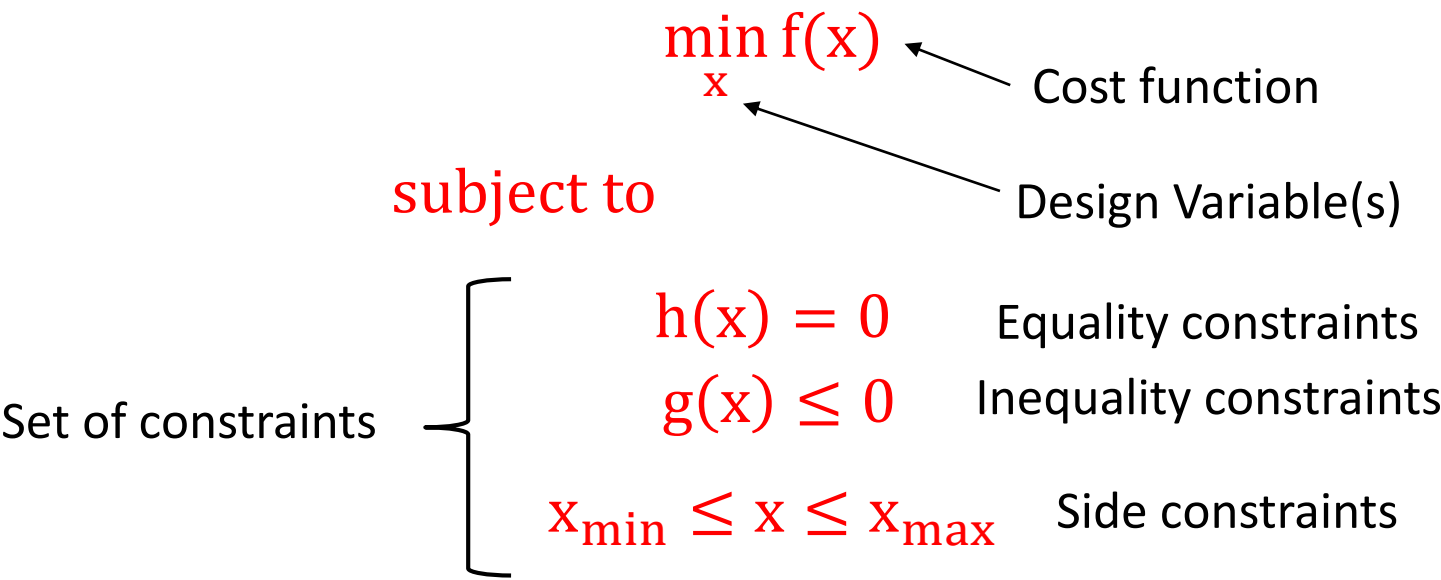
- **Optimization Problem:** piston pin
- **Input Parameters:** diameters (D_{in} , D_{out}), length (L), load (F), density (ρ)
- **Input Variables:** D_{in} , D_{out} , L
- **Constants:** $F = 3000$ N, $\rho = 7850$ kg/m³
- **Output Parameters:**
mass $M = (D_{out}^2 - D_{in}^2)L\rho\pi/4$
max momentum $C_{max} = FL/4$
moment of inertia $I = (D_{out}^4 - D_{in}^4)\pi/64$
max stress $\sigma_{max} = C_{max}D_{out}/2I$
- **Objective Function:** minimize M
- **Constraints:**
 $\sigma_{max} \leq 200$ MPa
 $80 \leq L \leq 100$ mm
 $13 \leq D_{in} \leq 16$ mm
 $17 \leq D_{out} \leq 19$ mm



- **Optimum Solution:**
 $L = 80$ mm
 $D_{in} = 16$ mm
 $D_{out} = 18.72$ mm
 $M = 46.53$ g
 $\sigma_{max} = 200$ MPa

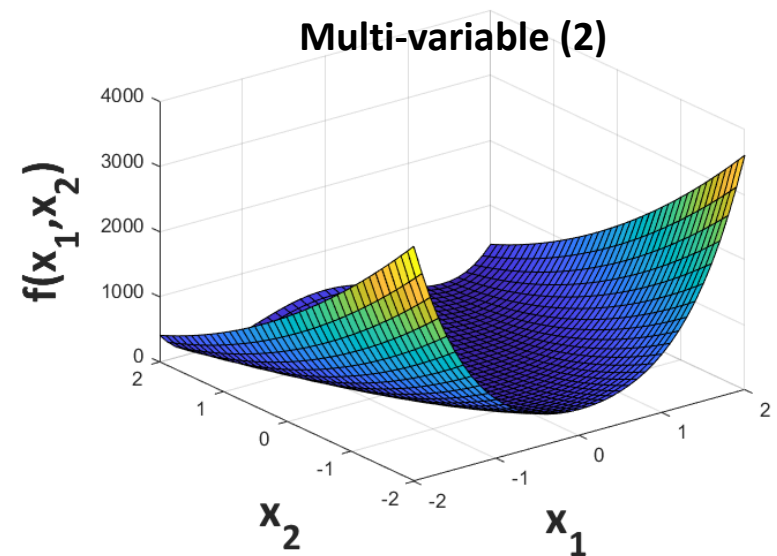
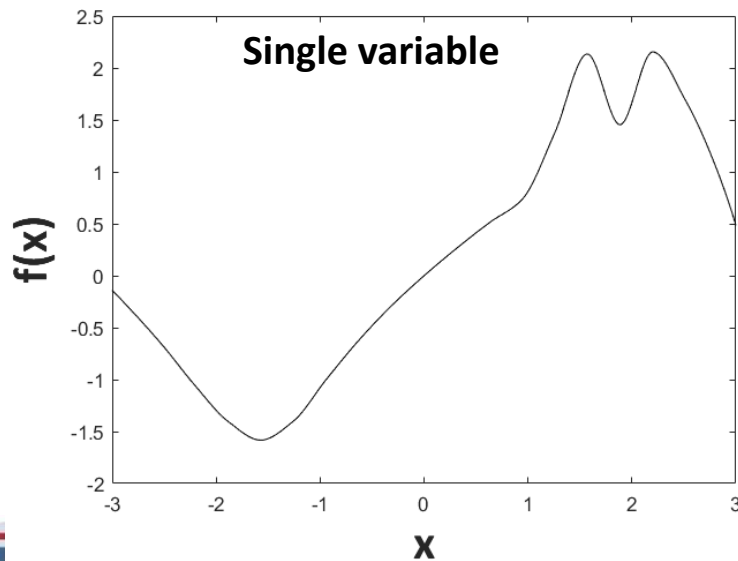
Optimization:

- Concerns the minimization or maximization of functions
- Standard Optimization Problem

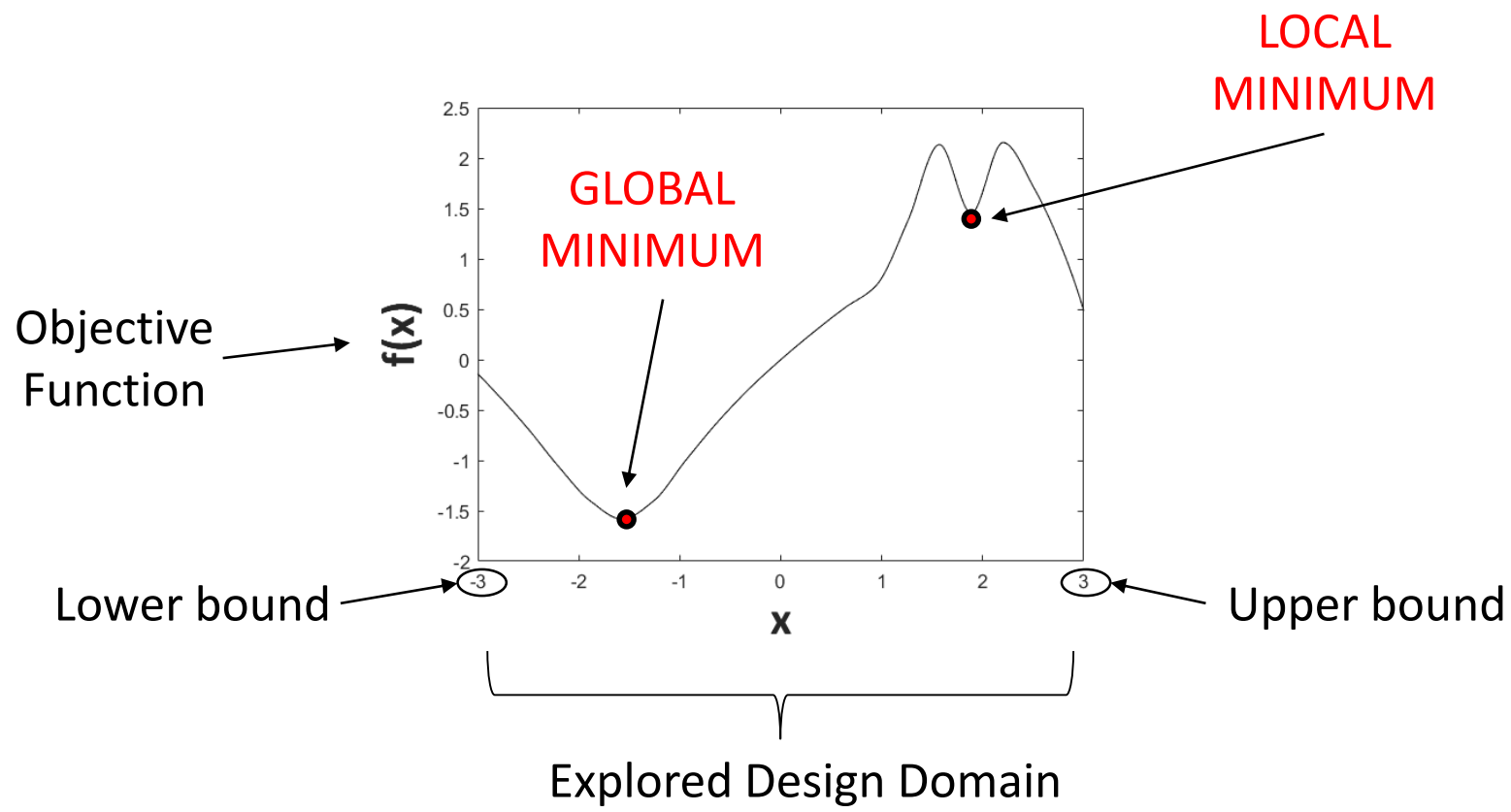


$f(\mathbf{x})$ is the objective function, which measures and evaluates the performance of a system. In a standard problem, we are minimizing the function. For maximization, it is equivalent to minimize $-f(\mathbf{x})$.

\mathbf{x} is a column vector of design variables, which can affect the performance of the system.



Function Minimization:



Constraints:

- Limitation to the design space
- Can be linear or nonlinear, explicit or implicit functions

$$h(x) = 0$$

Equality constraints

Many cases
require
less than!!!

$$g(x) \leq 0$$

Inequality constraints

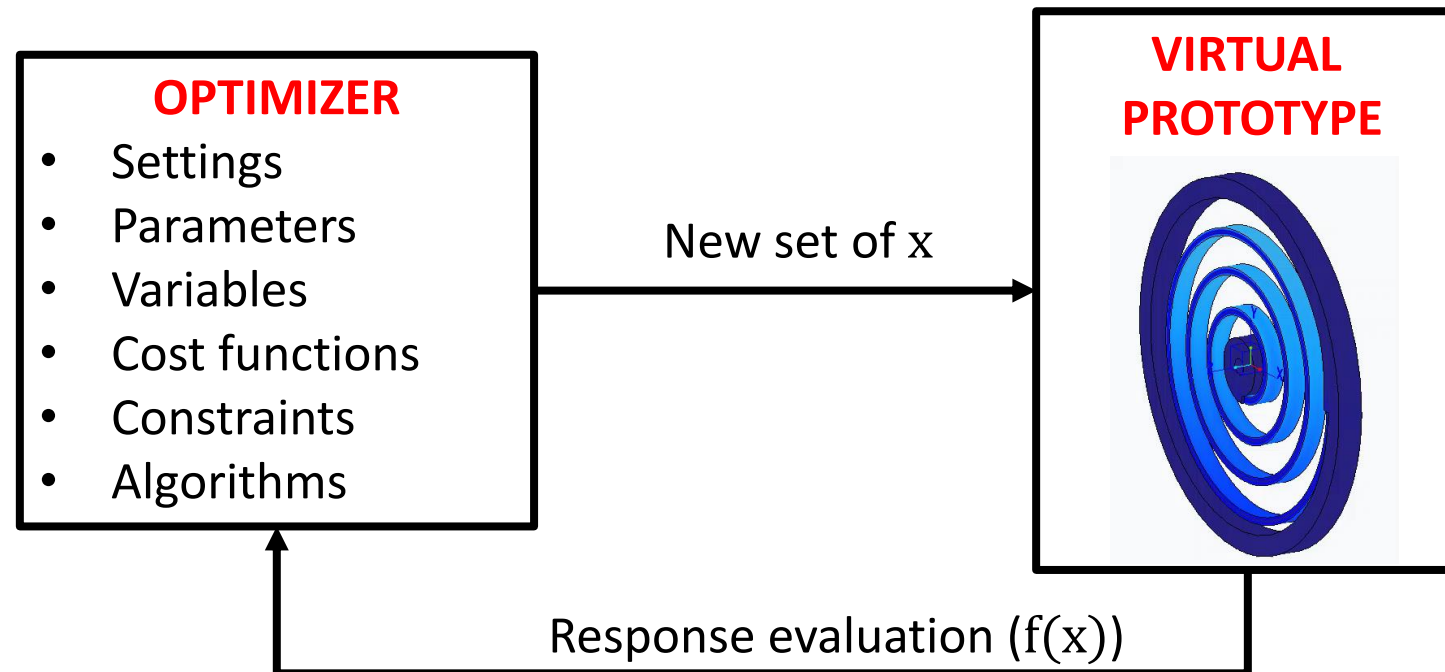
$$x_{\min} \leq x \leq x_{\max}$$

Side constraints

Usually
referred to as
lower/upper
Bounds



Optimization Loop:



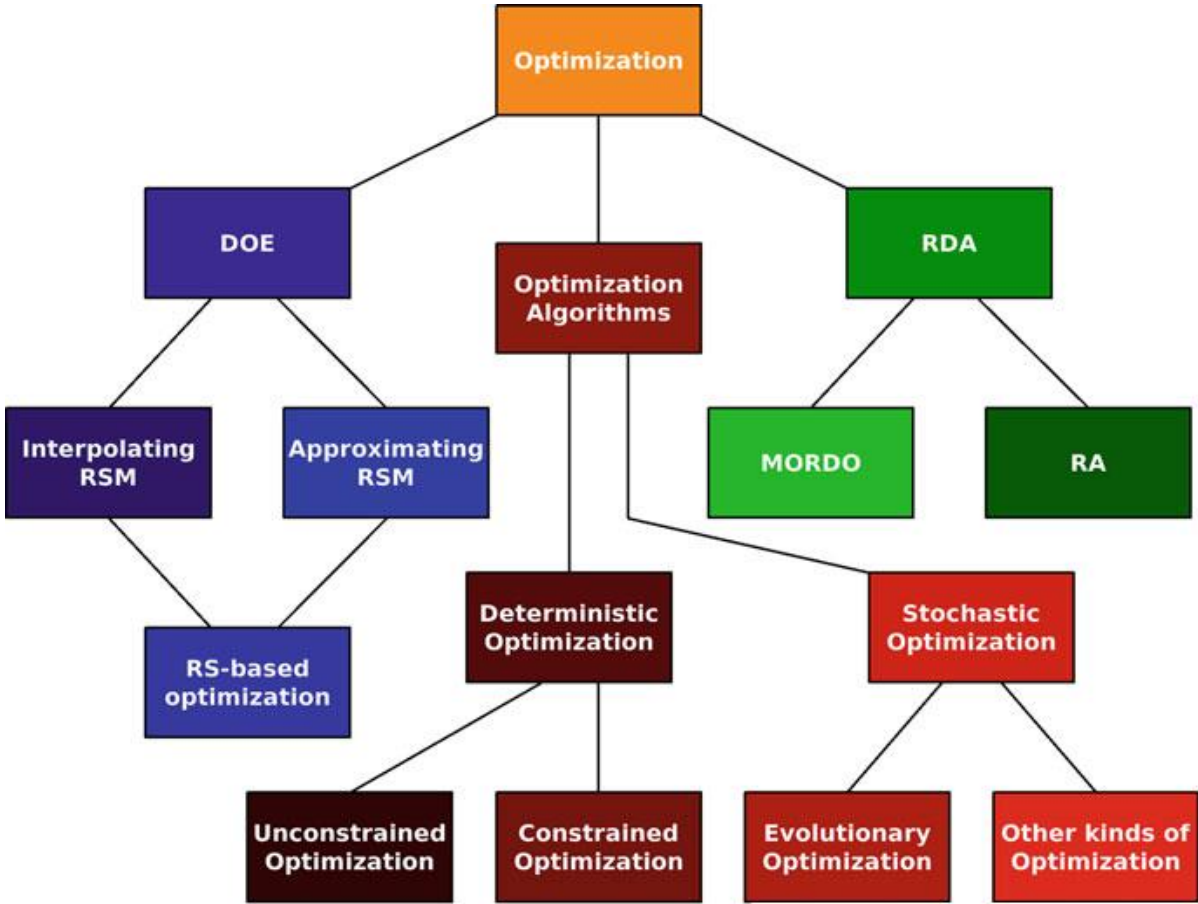
Optimization Approaches:

Let us focus on the most used approaches:

Design Of Experiment (DOE) + Response Surface Modeling (RSM)

Deterministic Algorithms

Stochastic Algorithms



Deterministic Optimization (DO):

Deterministic Optimization (DO):

- A rigid mathematical scheduling is followed and no random elements appear.
- It is also called mathematical programming.
- DOs rely on the computation of the gradient and the Hessian of the response function
- They show a very fast convergence
- DO methods generally start from a feasible point and generate a sequence of points converging to the solution

Implementing DO in Matlab:

Matlab Optimization Toolbox provides a wide library of algorithms. The toolbox includes routines for:

- **Unconstrained OR Constrained** optimization
- **Gradient-based OR derivative free** methods
- **Single OR multi-function** (≠ multi-objective) problems

Implementing DO in Matlab:

Gradient-based

Derivative free

Type	Formulation	Solver
Scalar minimization	$\min_x f(x)$ <p>such that $lb < x < ub$ (x is scalar)</p>	fminbnd
Unconstrained minimization	$\min_x f(x)$	fminunc, fminsearch
Linear programming	$\min_x f^T x$ <p>such that $A \cdot x \leq b, Aeq \cdot x = beq, lb \leq x \leq ub$</p>	linprog
Mixed-integer linear programming	$\min_x f^T x$ <p>such that $A \cdot x \leq b, Aeq \cdot x = beq, lb \leq x \leq ub, x(intcon)$ is integer-valued.</p>	intlinprog
Quadratic programming	$\min_x \frac{1}{2} x^T H x + c^T x$ <p>such that $A \cdot x \leq b, Aeq \cdot x = beq, lb \leq x \leq ub$</p>	quadprog
Constrained minimization	$\min_x f(x)$ <p>such that $c(x) \leq 0, ceq(x) = 0, A \cdot x \leq b, Aeq \cdot x = beq, lb \leq x \leq ub$</p>	fmincon
Semi-infinite minimization	$\min_x f(x)$ <p>such that $K(x,w) \leq 0$ for all $w, c(x) \leq 0, ceq(x) = 0, A \cdot x \leq b, Aeq \cdot x = beq, lb \leq x \leq ub$</p>	fseminf

<https://it.mathworks.com/help/optim/ug/problems-handled-by-optimization-toolbox-functions.html>

Implementing DO in Matlab:

Type	Formulation	Solver
Goal attainment	$\min_{x,\gamma}$ such that $F(x) - w \cdot \gamma \leq \text{goal}$, $c(x) \leq 0$, $\text{ceq}(x) = 0$, $A \cdot x \leq b$, $Aeq \cdot x = beq$, $lb \leq x \leq ub$	<code>fgoalattain</code>
Minimax	$\min_x \max_i F_i(x)$ such that $c(x) \leq 0$, $\text{ceq}(x) = 0$, $A \cdot x \leq b$, $Aeq \cdot x = beq$, $lb \leq x \leq ub$	<code>fminimax</code>

Multi-function Optimization

Overview of the available optimizers

<https://it.mathworks.com/help/optim/ug/problems-handled-by-optimization-toolbox-functions.html>

Deterministic Optimization (DO):

Unconstrained Optimization

(fminunc routine)

Unconstrained Optimization – **fminunc** :

Nonlinear programming solver. It Finds the minimum of a problem specified by:

$$\min_x f(x)$$

where:

$f(x)$ is a function that returns a scalar

x is a vector or a matrix

Unconstrained Optimization – **fminunc** :

Syntax

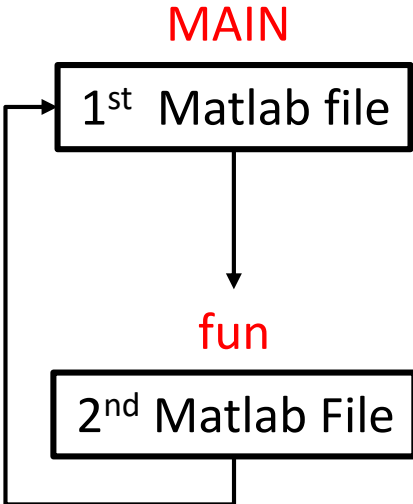
Command

INPUT

OUTPUT

```
x = fminunc(fun,x0)
x = fminunc(fun,x0,options)
x = fminunc(problem)
[x,fval] = fminunc(___)
[x,fval,exitflag,output] = fminunc(___)
[x,fval,exitflag,output,grad,hessian] = fminunc(___)
```

- where:
- fun → function to be minimized
 - x0 → starting point
 - x → optimal value
 - fval → objective function value at solution



Unconstrained Optimization – **fminunc** :

Example: Minimize the Rosenbrock's function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

2-variables function $\longrightarrow x = [x_1, x_2]$

External
costfun.m
file

```
function f = costfun(x)

% Rosenbrock's function
x1=x(1);
x2=x(2);

f=100*(x2-x1^2)^2 + (1-x1)^2;
```

Unconstrained Optimization – **fminunc** :

```
% design variables x1=x(1), x2=x(2)

% cost function
fun=@costfun;

% starting point
x0 = [-1.2,1];

%options
alg='quasi-newton'; %(default)
%alg='trust-region';

options = optimoptions('fminunc','Algorithm',alg,'Display','iter','PlotFcns',@optimplotfval);

% optimization (x-->optimal point fval-->related function value)
[xyopt,fval] = fminunc(fun,x0,options);
```

MAIN

Used to set the routine
(e.g. the algorithm)

External
costfun.m
file

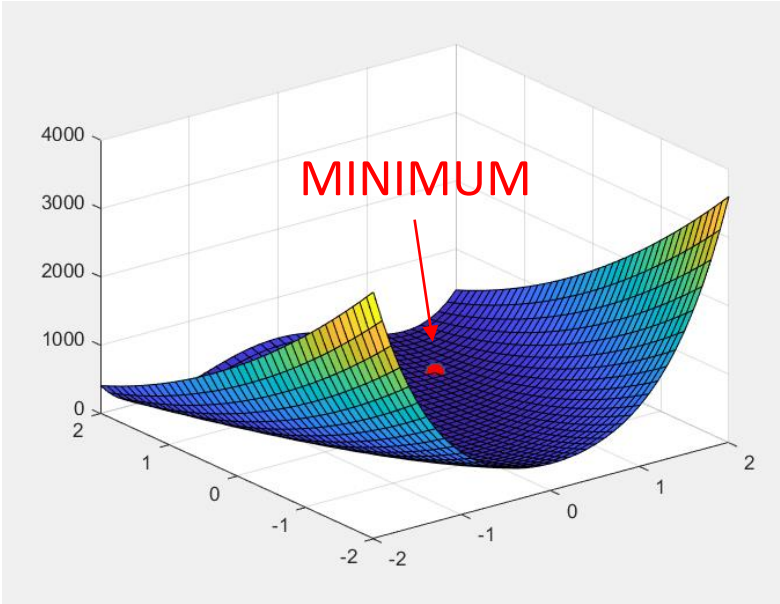
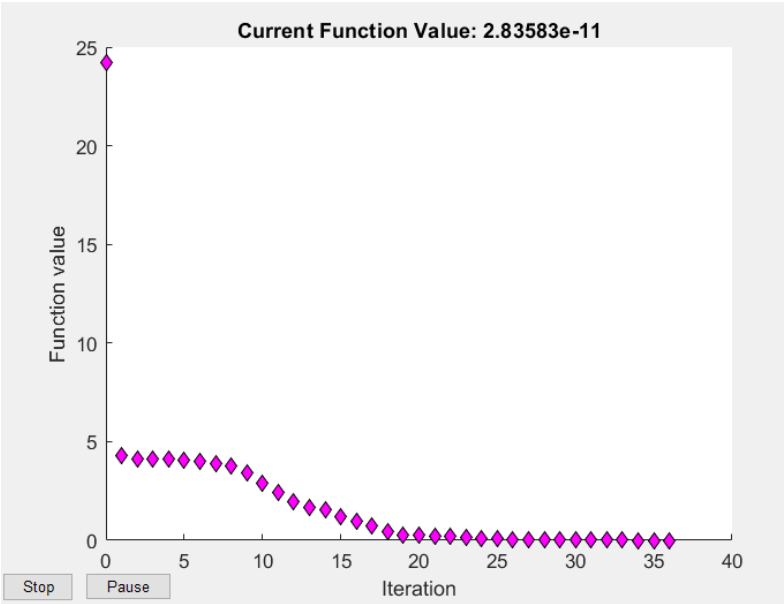
```
function f = costfun(x)

% Rosenbrock's function
x1=x(1);
x2=x(2);

f=100*(x2-x1^2)^2 + (1-x1)^2;
```



Unconstrained Optimization – **fminunc** :



alg	'quasi-newton'
fun	@costfun
fval	2.8358e-11
options	1x1 Fminunc
x	41x41 double
x0	[-1 2000 1]
xyopt	[1.0000,1.0000]
y	41x41 double
z	41x41 double

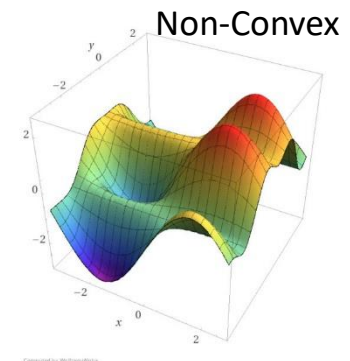
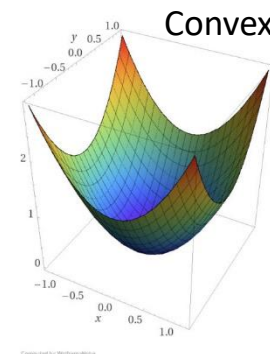
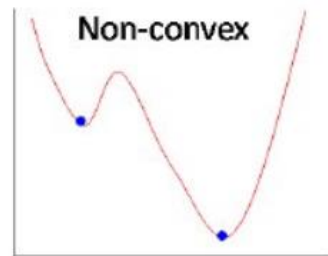
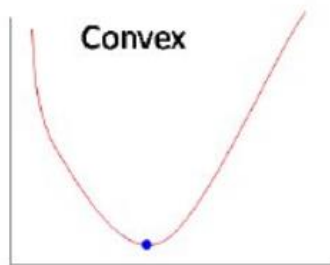


Unconstrained Optimization – **fminunc** :

Some practical considerations:

- If your objective function includes a gradient, use 'Algorithm' = 'trust-region', and set the “SpecifyObjectiveGradient” option to true.
- Otherwise, use 'Algorithm' = 'quasi-newton'.
- For non-convex functions, the solver may find local minima instead of global minima. Thus, the starting point (x_0) assumes importance as it strongly influences the solution.

Solution: Several RUN
with different x_0



Unconstrained Optimization:

fminunc VS **fminsearch**

- Once again, **fminsearch** exploits a derivative free method (**Nelder-Mead** simplex direct search)
- **fminunc** and **fminsearch** provide quite similar results when considering **SMOOTH** objective functions. A smooth function is a function that has continuous derivatives up to some desired order over some domain.
- Nonsmooth functions need the use of **fminsearch**
- In the last case, also **patternsearch** and **fmincon** may become valid alternatives (see the comparative study reported in <https://it.mathworks.com/help/gads/examples/optimize-a-nonsmooth-function.html>)



Deterministic Optimization (DO):

Constrained Optimization

(fmincon routine)

Constrained Optimization – **fmincon** :

Nonlinear programming solver. It Finds the minimum of a problem specified by:

$$\text{Problem} \left\{ \begin{array}{l} \min_x f(x) \\ Ax \leq b \\ A_{eq} x = b_{eq} \\ c(x) \leq 0 \\ c_{eq}(x) = 0 \\ lb \leq x \leq ub \end{array} \right. \begin{array}{l} \longrightarrow \text{Linear Constraints} \\ \longrightarrow \text{Nonlinear Constraints} \\ \longrightarrow \text{Side Constraints} \end{array}$$

where:

$f(x)$ is a function that returns a scalar

x is a vector or a matrix



Constrained Optimization – **fmincon** :

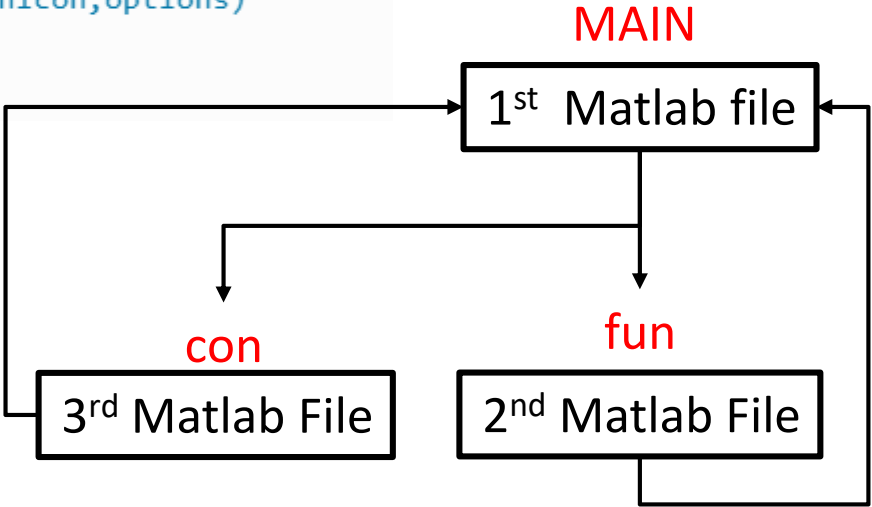
Syntax

Command

```
x = fmincon(fun,x0,A,b) INPUT
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x = fmincon(problem)
[x,fval] = fmincon(__)
```

OUTPUT

- where:
- fun → function to be minimized
 - x0 → starting point
 - x → optimal value
 - fval → objective function value at solution



Constrained Optimization – **fmincon** :

Example: Minimize the Rosenbrock's function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

2-variables function $\longrightarrow x = [x_1, x_2]$

External costfun.m file

```
function f = costfun(x)

% Rosenbrock's function
x1=x(1);
x2=x(2);

f=100*(x2-x1^2)^2 + (1-x1)^2;
```

Linear constraint \longrightarrow Previous (unconstrained, **fminunc**) solution (1,1) is now avoided

Added via simple vectors
In the main Matlab file

$$x_1 + 2x_2 \leq 1 \longrightarrow \begin{matrix} A = [1,2] \\ b = 1 \end{matrix}$$

Constrained Optimization – **fmincon** :

Example: Minimize the Rosenbrock's function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

2-variables function $\longrightarrow x = [x_1, x_2]$

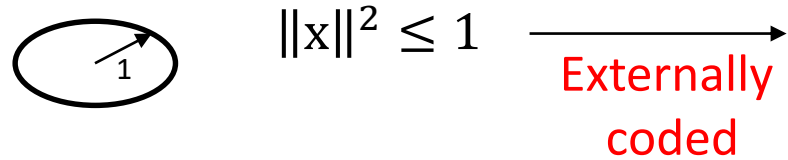
External costfun.m file

```
function f = costfun(x)

% Rosenbrock's function
x1=x(1);
x2=x(2);

f=100*(x2-x1^2)^2 + (1-x1)^2;
```

Nonlinear constraint \rightarrow e.g. unit disk



External constraint.m file

```
function [c, ceq] = constraint(x)

c = x(1)^2+x(2)^2-1;
ceq = [];
```

Constrained Optimization – **fmincon** :

```
% cost function
fun=@costfun;

% starting point
x0 = [-1.2,1];

% lower/upper bounds for variables
lb=[];
ub=[];

%linear constraints (defined in the main script)
A = [1,2];
b = 1;
Aeq=[];
beq=[];

%nonlinear constraints (defined via an external function)
nonlcon = [];

%options
alg='interior-point'; %(default)
%alg='trust-region-reflective';
%alg='sqp';
%alg='sqp-legacy';
%alg='active-set';

options = optimoptions('fmincon','Algorithm',alg,'Display','iter');

% optimization (x-->optimal point fval-->related function value)
[xyopt,fval] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options);
```

MAIN

External
costfun.m
file

```
function f = costfun(x)

% Rosenbrock's function
x1=x(1);
x2=x(2);

f=100*(x2-x1^2)^2 + (1-x1)^2;
```

Linear
Constraints

Used to set
the routine
(e.g. the
algorithm)



Constrained Optimization – **fmincon** :

Nonlinear
Constraints

Used to set
the routine
(e.g. the
algorithm)

```
% cost function
fun=@costfun;

% starting point
x0 = [-1.2,1];

% lower/upper bounds for variables
lb=[];
ub=[];

%linear constraints (defined in the main script)
A=[];
b=[];
Aeq=[];
beq=[];

%nonlinear constraints (defined via an external function)
nonlcon = @constraint;

%options
alg='interior-point'; %(default)
%alg='trust-region-reflective';
%alg='sqp';
%alg='sqp-legacy';
%alg='active-set';

options = optimoptions('fmincon','Algorithm',alg,'Display','iter');

% optimization (x-->optimal point fval-->related function value)
[xyopt,fval] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options);
```

MAIN

External
costfun.m
file

```
function f = costfun(x)

% Rosenbrock's function
x1=x(1);
x2=x(2);

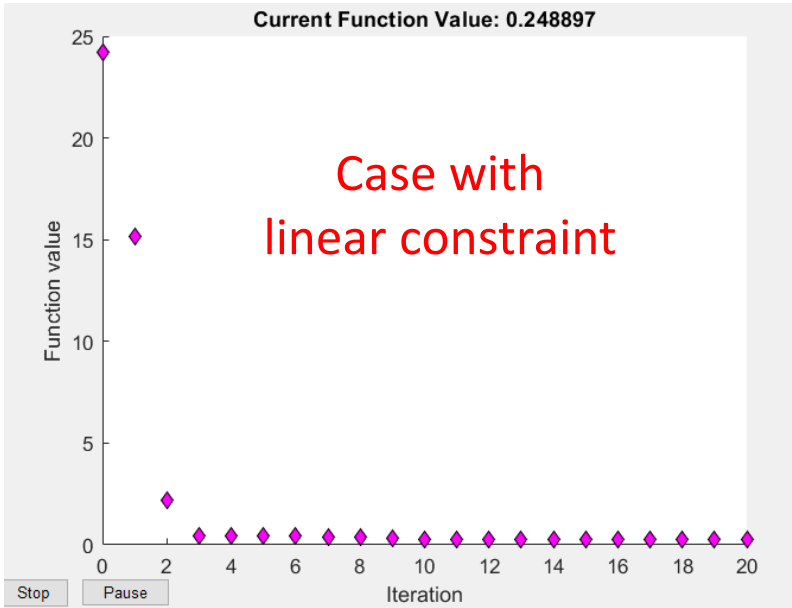
f=100*(x2-x1^2)^2 + (1-x1)^2;
```

External
constraint.m
file

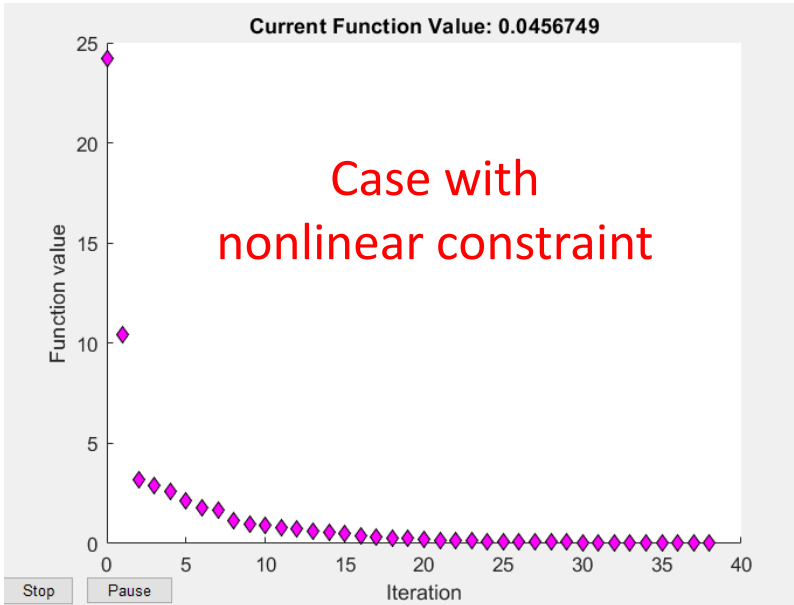
```
function [c, ceq] = constraint(x)

c = x(1)^2+x(2)^2-1;
ceq = [];
```

Constrained Optimization – **fmincon** :



fval	0.2489
lb	[]
nonlcon	[]
options	1x1 Fmincon
ub	[]
x	41x41 double
x0	[-1.2000,1]
xyopt	[0.5022,0.2489]



fval	0.0457
lb	[]
nonlcon	@constraint
options	1x1 Fmincon
ub	[]
x	41x41 double
x0	[-1.2000,1]
xvopt	[0.7864,0.6177]



Constrained Optimization – **fmincon** :

Some practical considerations:

- Use the 'interior-point' algorithm (default) first. To speed up the routine in small/medium-sized problems, try 'sqp' next, and 'active-set' last. Use 'trust-region-reflective' when applicable. Your problem must have: objective function that includes gradient, only bounds, or only linear equality constraints (but not both).
- For non-convex functions, TO AVOID LOCAL MINIMA, try the **GlobalSearch** or the **MultiStart** options.

```
rng default
problem = createOptimProblem('fmincon','x0',x0,'objective',fun,'lb',lb,'ub',ub,'nonlcon',nonlcon);
gs = GlobalSearch;
ms = MultiStart;
[xyopt,fval] =run(gs,problem);
```

<https://it.mathworks.com/help/gads/how-globalsearch-and-multistart-work.html>



GlobalSearch Vs MultiStart:

- **GlobalSearch** analyzes start points and rejects those points that are unlikely to improve the best local minimum found so far. **MultiStart** runs all start points (or, optionally, all start points that are feasible with respect to bounds or inequality constraints).
- **MultiStart** gives a choice of local solver: fmincon, fminunc, etc. **GlobalSearch** uses fmincon.
- Use **GlobalSearch** to find a single global minimum most efficiently in a single run.
- Use **MultiStart** to find multiple local minima, to use a solver other than fmincon or to explore your own start points.



Deterministic Optimization (DO):

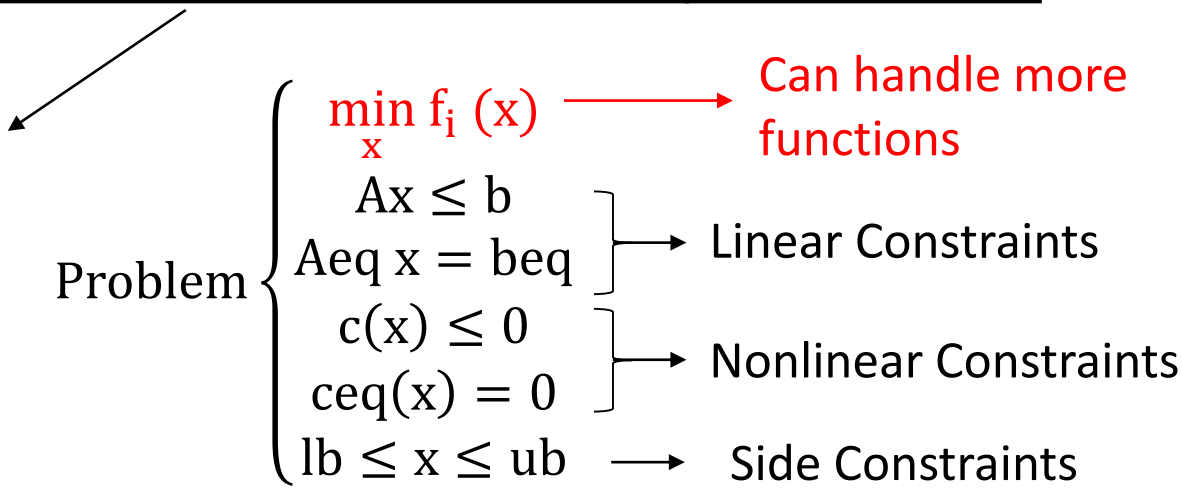
Constrained Optimization

**“Multi-function”
(fminimax routine)**

“Multi-function” Optimization – **fminimax**

It seeks a point that minimizes the maximum of a set of objective functions:

Deterministic opt. is by definition **single objective**. Fully multi-objective opt. is part of stochastic opt.



where:

$f(x)$ is a function that returns a scalar

x is a vector or a matrix

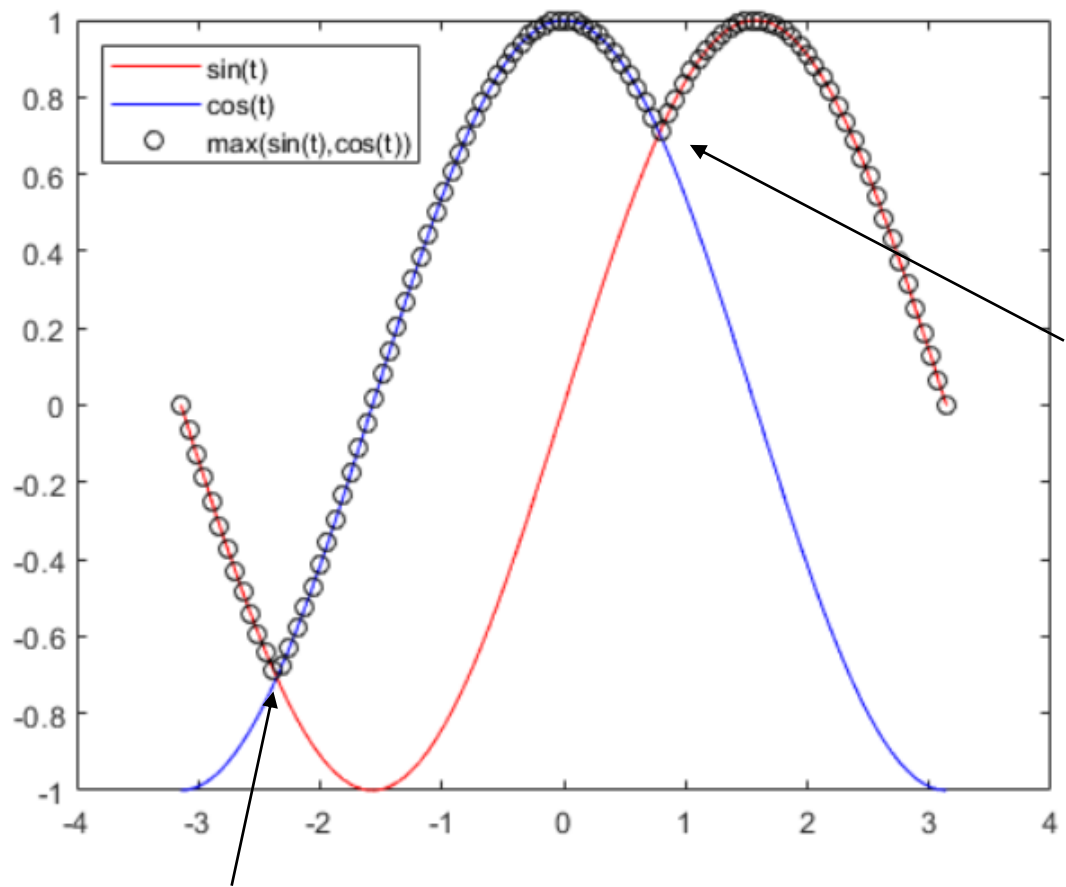
“Multi-function” Optimization – **fminimax**

Function 1:

$$f_1 = \sin(x)$$

Function 2:

$$f_2 = \cos(x)$$



Minimum found
by fminimax if
 $x_0=1.5$

Minimum found by fminimax if $x_0=-1$

Stochastic Optimization (SO):

Stochastic Optimization (SO):

Compared to DO, SO methods:

- Mathematically speaking are much easier but have a much slower convergence
- Contain randomness in the search procedure, which allows the sample to move freely in the design space overcoming local minima
- Can handle multi-objective optimizations



Stochastic **Multi-Objective** Optimization:

- It relies on the concept of **Pareto optimality**. The result of a single-objective optimization is a single optimum solution. The result of a multi-objective optimization is a set of non-dominated solutions.
- Consider an optimization problem having the two objectives of minimizing the function $f_1(x)$ and minimizing the function $f_2(x)$
- Consider two samples x_1 and x_2 :
 - if $f_1(x_1) < f_1(x_2)$ and $f_2(x_1) < f_2(x_2)$ then x_1 dominates x_2 (better performance for both the functions).
 - if $f_1(x_1) < f_1(x_2)$ and $f_2(x_1) > f_2(x_2)$ it is not possible to establish which sample is better, thus the two samples are non-dominated



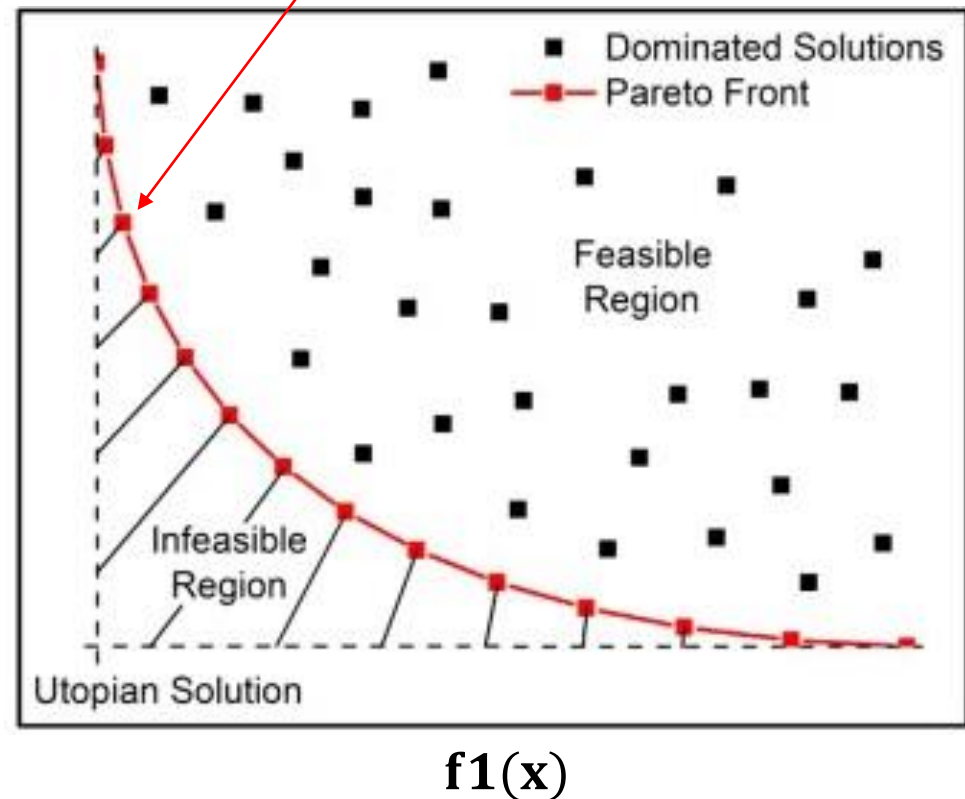
Pareto front:

The Pareto front is the set of

non-dominated solutions in the

- After the optimization, an optimum solution can be picked from the Pareto front following certain criteria. This process is called multi-criteria decision making.

- If in the future a different trade-off between the objectives will be preferred, there will be no need to run the optimization again

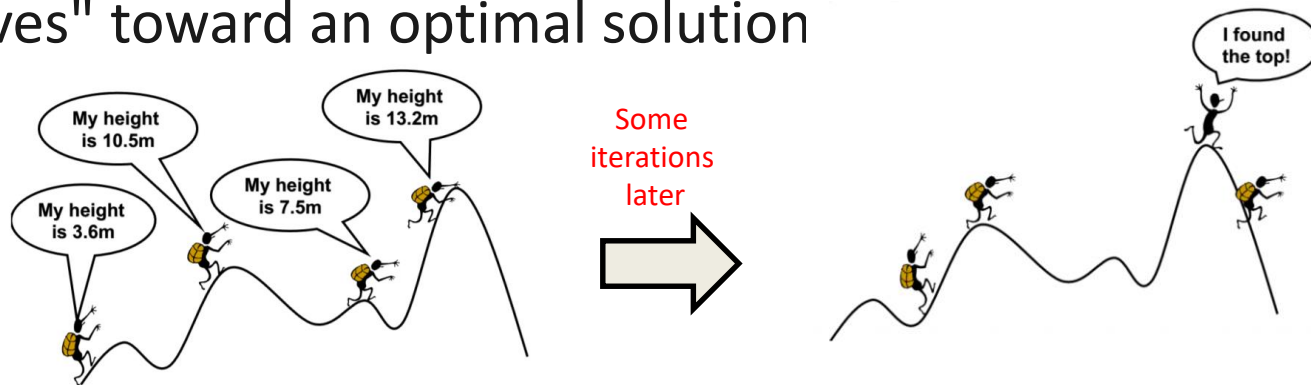


Stochastic Optimization (SO):

Genetic Algorithm
(ga routine)

Genetic Algorithm – ga

- It solves both constrained and unconstrained optimization problems based on a natural selection process that mimics biological evolution.
- It usually converges to the **GLOBAL MINIMUM**.
- At each step, the genetic algorithm selects individuals from the current population and uses them as parents to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution

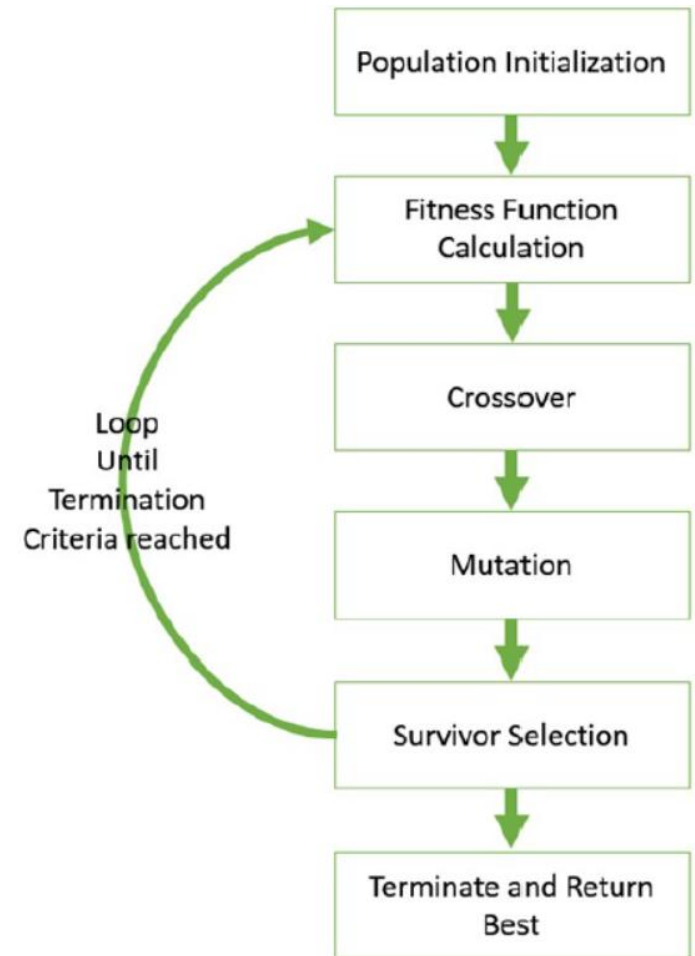


Genetic Algorithm – ga

ga uses the current population to create the children for the next generation.

Three types of children for the next generation:

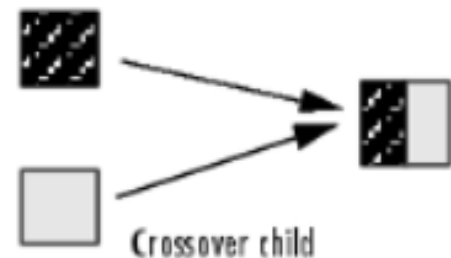
- **Elite** are the individuals in the current generation with the best fitness values. These individuals automatically survive to the next generation.
- **Crossover** are created by combining the vectors of a pair of parents.
- **Mutation** children are created by introducing random changes, or mutations, to a single parent.



Genetic Algorithm – ga



Elite children go to the next generations without changes



Crossover children by selecting vector entries, or genes, from a pair of individuals in the current generation and combines them to form a child



Mutation children by applying random changes to a single individual in the current generation to create a child

The user can specify how many of each type of children the algorithm creates

Stop Conditions?

- When there has been no changes in the results (function value) for X iterations. In other words, when the difference between the i -th iteration and the $(i+1)$ -th iteration is less than a certain, user defined, tolerance.

OR

- When the optimization routine reaches the maximum number of generations (also specified by the user).

Genetic Algorithm – ga

Syntax

Command

```

x = ga(fun,nvars)
x = ga(fun,nvars,A,b)      INPUT
OUTPUT x = ga(fun,nvars,A,b,Aeq,beq)
x = ga(fun,nvars,A,b,Aeq,beq,lb,ub)
x = ga(fun,nvars,A,b,Aeq,beq,lb,ub,nonlcon)
x = ga(fun,nvars,A,b,Aeq,beq,lb,ub,nonlcon,options)
x = ga(fun,nvars,A,b,[],[],lb,ub,nonlcon,IntCon)
x = ga(fun,nvars,A,b,[],[],lb,ub,nonlcon,IntCon,options)
x = ga(problem)
[x,fval] = ga( __ )

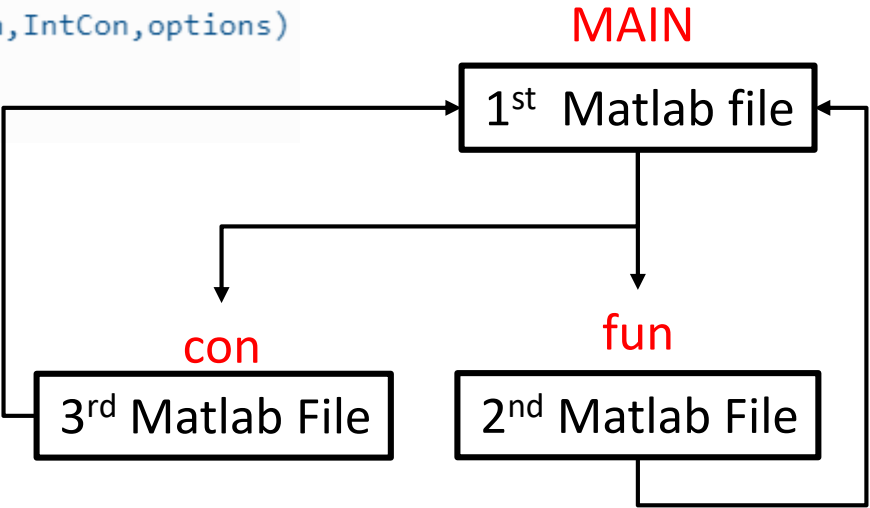
```

where:

fun → function to be minimized

x → optimal value

fval → objective function value at solution



Let us test the Rosenbrock's function again:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

2-variables function $\longrightarrow x = [x_1, x_2]$

External costfun.m file

```
function f = costfun(x)

% Rosenbrock's function
x1=x(1);
x2=x(2);

f=100*(x2-x1^2)^2 + (1-x1)^2;
```

Linear constraint

Added via simple vectors
In the main Matlab file

$$x_1 + 2x_2 \leq 1 \longrightarrow \begin{matrix} A = [1,2] \\ b = 1 \end{matrix}$$

MAIN

```
% design variables x1=x(1), x2=x(2)
nvar=2;

% cost function
fun=@costfun;

% lower/upper bounds for variables
lb=[];
ub=[];

%linear constraints (defined in the main script)
A = [1,2];
b = 1;
Aeq=[];
beq=[];

%nonlinear constraints (defined via an external function)
nonlcon =[]; %@constraint

%options
Gen=200; % if not specified --> 100*nvar
PoP_size=50; % if not specified --> 50 when nvar<= 5, 200 otherwise
MaxStallGen=50; % if not specified --> 50
Tol=1e-6; % if not specified --> 1e-6
CrossoverFrac=0.8; % if not specified --> 0.8
EliteFrac=0.05; % if not specified --> 0.05

options = optimoptions('ga','Generations',Gen,'PopulationSize',PoP_size,'MaxStallGenerations',MaxStallGen,'EliteCount',ceil(EliteFrac*PoP_size),...
'CrossoverFraction',CrossoverFrac,'FunctionTolerance',Tol,'PlotFcn',{@gaplotbestf});

tic
% optimization (x-->optimal point fval-->related function value)
[xyopt,fval] = ga(fun,nvar,A,b,Aeq,beq,lb,ub,nonlcon,options);
toc
```

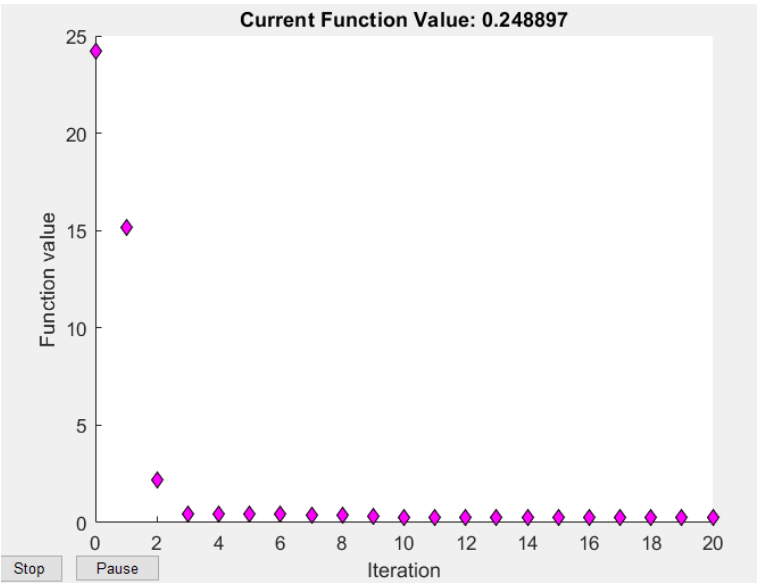
Used to set
the routine



Constrained Optimization – **fmincon** Vs **ga** :

Case with linear constraint

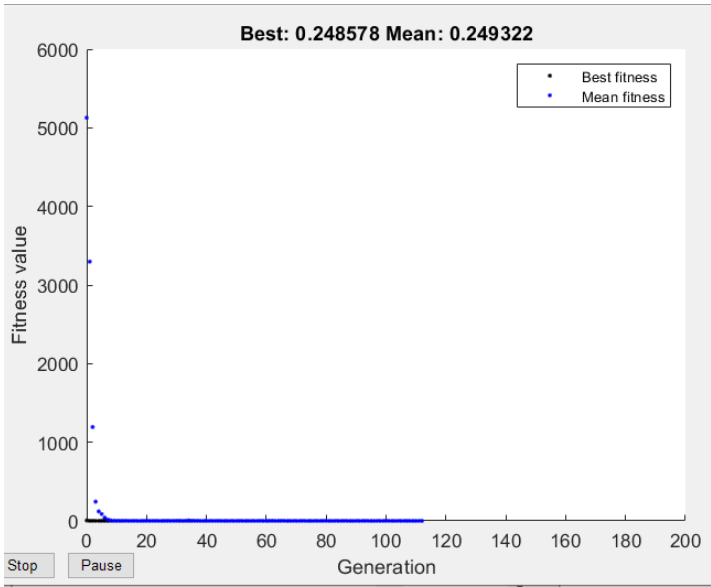
fmincon



$x=[0.50,0.25]$

$tot_time=1.1s$

ga



$x=[0.50,0.25]$

$tot_time=3.1s$

Nonconvex function– **fmincon** Vs **ga** :

Example: Minimize the Rastrigin's function (it has many local minima, with a global minimum at (0,0)):

$$f(x) = 20 + x_1^2 + x_2^2 - 10(\cos(2\pi x_1) + \cos(2\pi x_2))$$

We use $x_0 = [20, 30]$
to test the routine

2-variables function $\longrightarrow x = [x_1, x_2]$

External
costfun.m
file

```
function f = costfun(x)

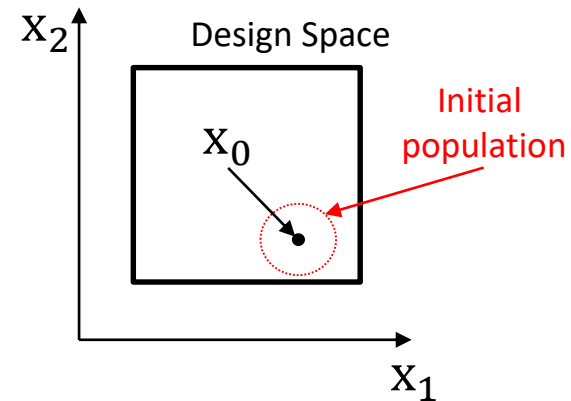
% Rastrigin's function
x1=x(1);
x2=x(2);

f=20+x1^2+x2^2-10*(cos(2*pi*x1)+cos(2*pi*x2));
```



How to set x_0 with ga?

ga works with a population of candidates, thus we need to create an “initial population” in the neighborhood of x_0



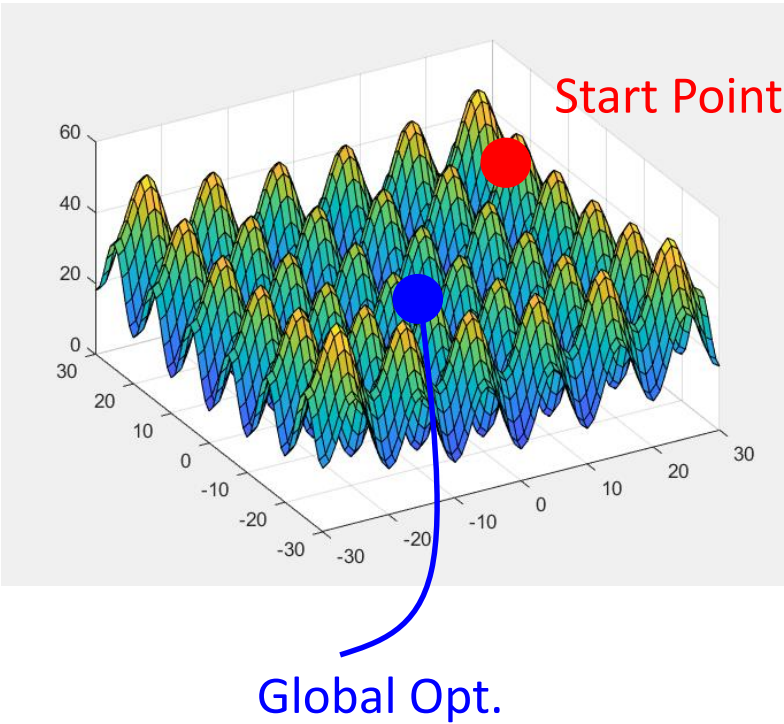
```
%options
MaxGen=200; % if not specified --> 100*nvar
PoP_size=50; % if not specified --> 50 when nvar<= 5, 200 otherwise
MaxStallGen=50; % if not specified --> 50
Tol=1e-6; % if not specified --> 1e-6
CrossoverFrac=0.8; % if not specified --> 0.8
EliteFrac=0.05; % if not specified --> 0.05
initpop = 10*randn(20,nvar) + repmat(x0,20,1); %instead of setting x0, an initial pop is created.
%If Pop_size is 50, 20 of each points are around x0 in the init generation.

options = optimoptions('ga', 'InitialPopulationMatrix',initpop, 'Generations',MaxGen, 'PopulationSize',PoP_size, 'MaxStallGenerations',MaxStallGen, ...
'EliteCount',ceil(EliteFrac*PoP_size), 'CrossoverFraction',CrossoverFrac, 'FunctionTolerance',Tol, 'PlotFcn',{@gaplotbestf});
```

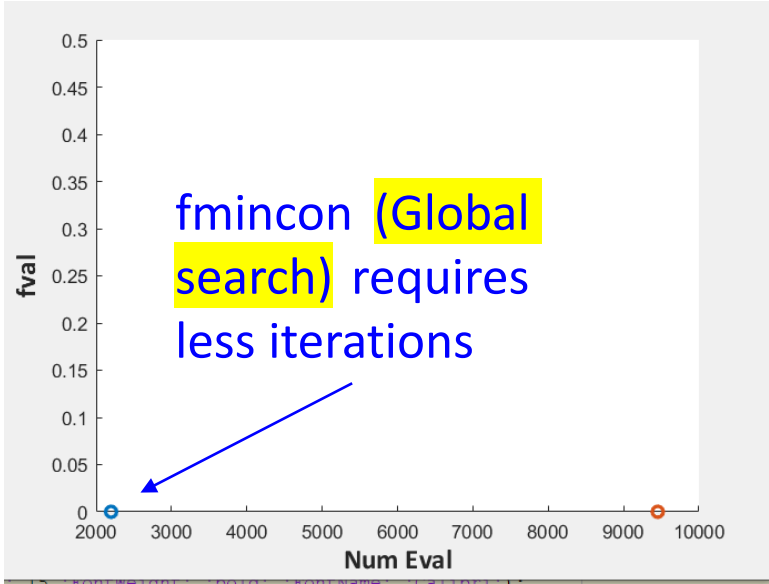


Nonconvex function– **fmincon** Vs **ga** :

Rastrigin's function



Both routines converged at the global minimum ($fval=0$, $x1=0, x2=0$)



fmincon Vs ga :

It seems like fmincon works better. So, why/when to use ga?

- In case of Multi-objective optimizations (please refer to `gamultiobj`)
- To find a GLOBAL minimum (it should be remarked that fmincon finds global minima only when the “GlobalSearch” option is activated).
- For its large and wide solution space search ability.
- Complex (non smooth) multi-variable (e.g. >10) variables are most likely to be approached via SO rather than DO.

Hybrid algorithms:

- It is good practice to take advantage of the capabilities of both SO and DO in cascade
- Objective functions with non-smooth regions can be handled with SO to avoid mathematical issues. Once the design space is explored and the local minima are avoided, a DO is to be applied to refine the results of a SO and to quickly conclude the optimization study.

```
hybridopts = optimoptions('fmincon','OptimalityTolerance',1e-10);  
options = optimoptions('ga','HybridFcn',{ 'fmincon', hybridopts});  
[x,fval] = ga(fun,nvars,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

→ These are available in the **ga** help page

Other interesting SO routine:

Particle Swarm Optimization → **particleswarm**

Syntax

```
x = particleswarm(fun,nvars)
x = particleswarm(fun,nvars,lb,ub)
x = particleswarm(fun,nvars,lb,ub,options)
x = particleswarm(problem)
[x,fval,exitflag,output] = particleswarm(__)
```

Comparisons between solvers:

<https://it.mathworks.com/help/gads/examples/optimize-a-nonsmooth-function.html>

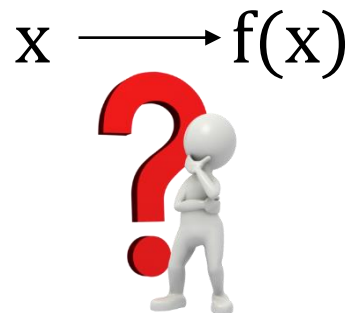
<https://it.mathworks.com/help/gads/example-comparing-several-solvers.html#bueuzxq>



Design of Experiments (DOE) + Response Surface Modeling (RSM)

DOE+RSM:

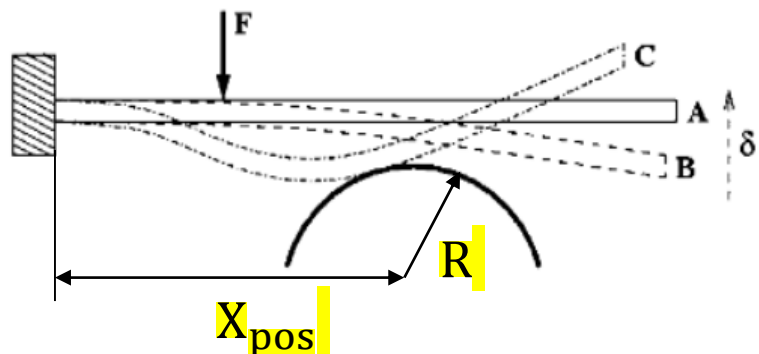
- It is not an optimization technique itself
- To be applied when analytical correlations between the design variables and the objective functions are not-available/too complex to be defined via theories.
- The aim is to collect the maximum amount of information on the design space using the minimum amount of resources



DOE+RSM:

Sometimes (complex scenarios) we do not know the correlation between design variables and objective functions.

Example

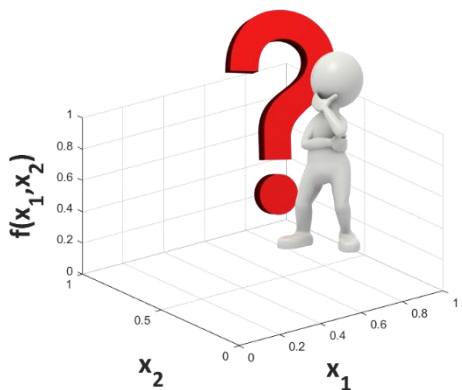


Design Variables

$x = [X_{pos}, R]$ \longrightarrow

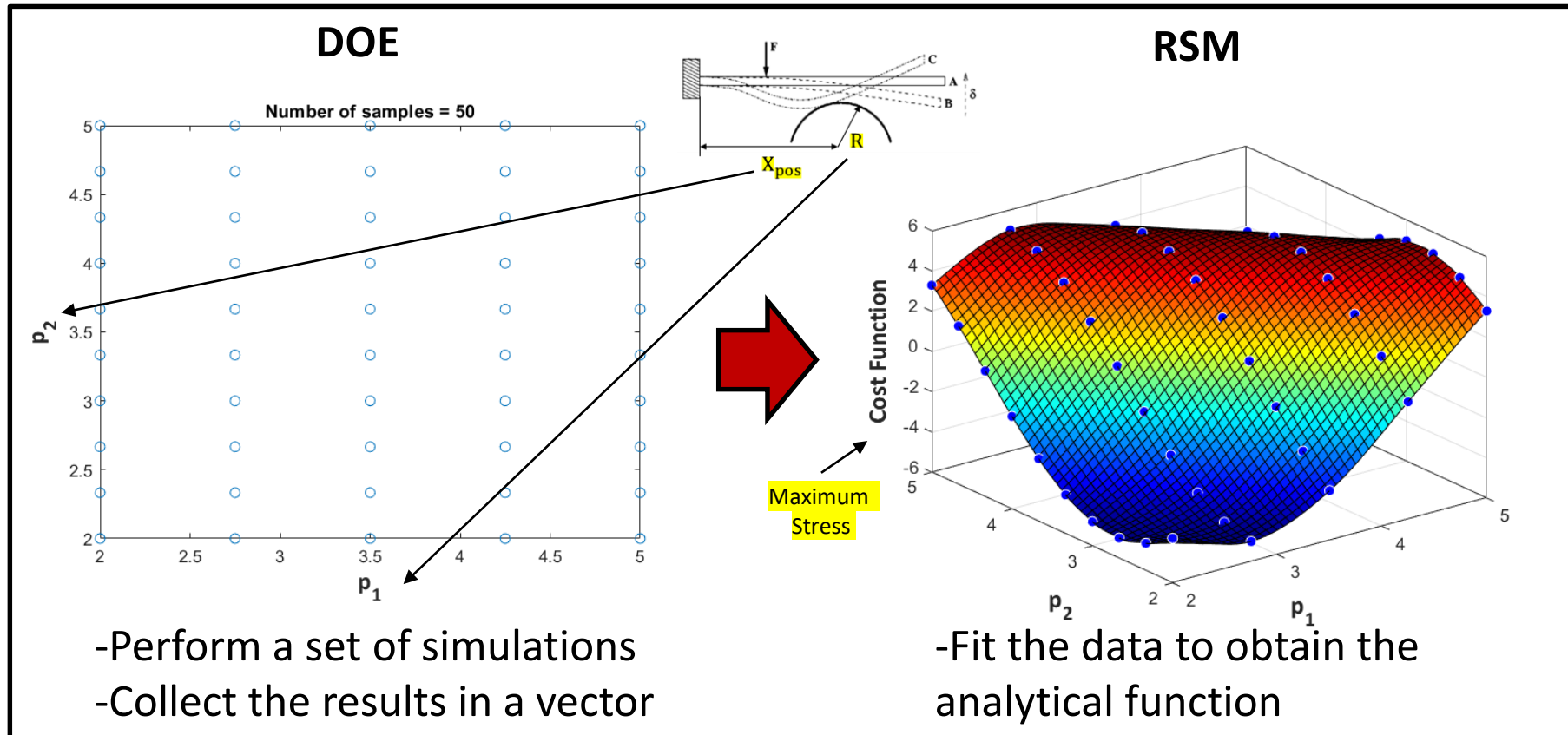
Objective Function

Maximum Stress
 $f(x) = \sigma$



DOE+ RSM :

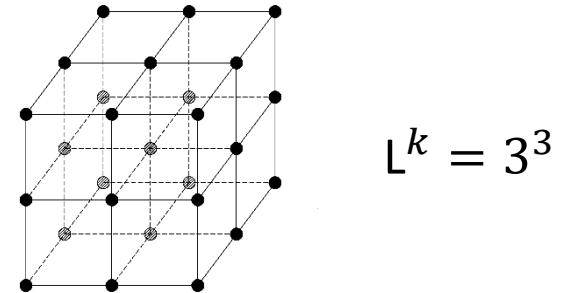
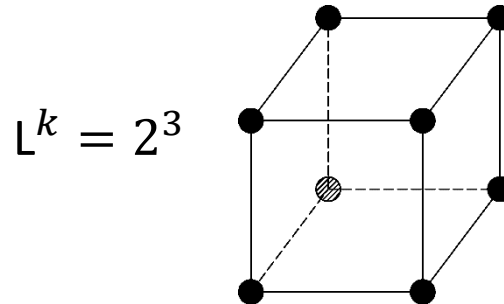
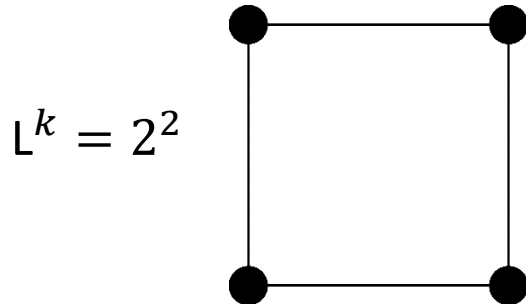
Let us build the relationship through well-defined experiments:



DOE – How to explore the design space?

Full-Factorial (FF)

- It is the most common and intuitive DOE strategy
- Suitable for optimization purpose as it allows precise reconstructions of the objective functions
- Assuming k factors and L levels, the samples are given by every possible combination of the factors value (the sample size is $N = L^k$)



DOE – How to explore the design space?

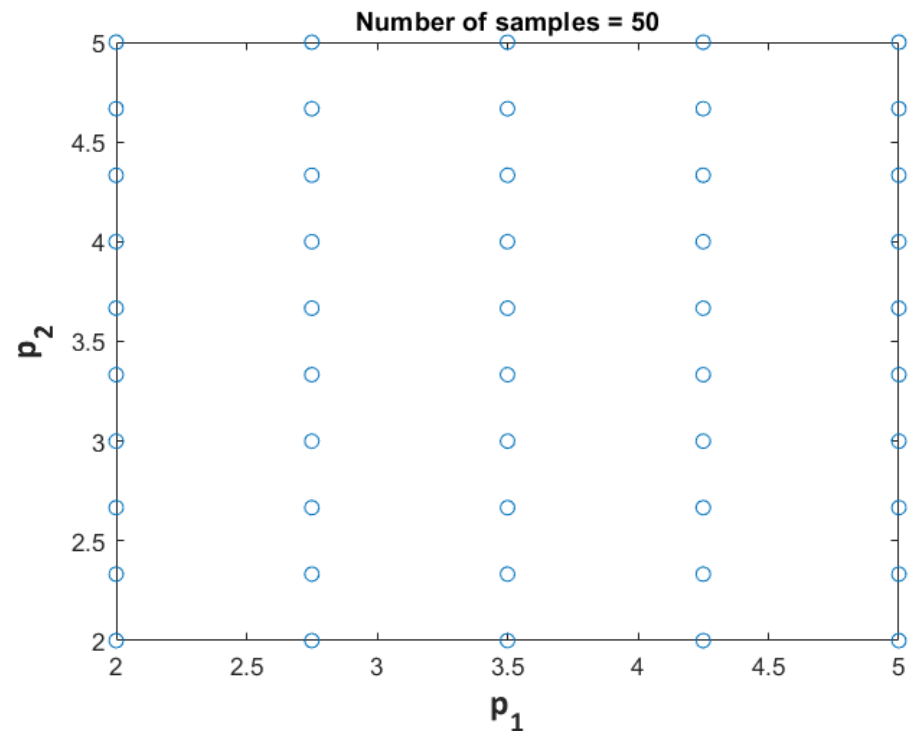
Full-Factorial (FF)

```
% Design of Experiment (DOE)

% Lower/Upper Bounds for DOE
p1_min = 2;
p1_max = 5;
p2_min = 2;
p2_max = 5;

% Parameters for Full-Factorial
levels_p1=5;
levels_p2=10;
```

```
FF = fullfact([levels_p1 levels_p2]);
vector_points_p1=rescale(FF(:,1));
vector_points_p2=rescale(FF(:,2));
matrix_points=[vector_points_p1 vector_points_p2];
```

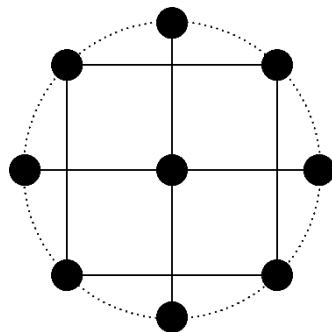


DOE – How to explore the design space?

Central Composite (CC)

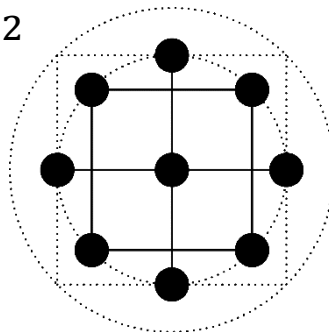
- A central composite design is a 2^k FF to which the central and the star points are added. A CC can be circumscribed, inscribed, faced, or scaled.
- A 2^k design has $N = 2k + 2k + 1$. Being $N > 2^k$ it allows the curvature of the design space to be estimated with a polynomial fitting.

Circumscribed

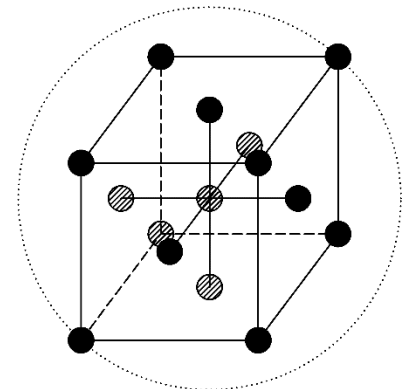


$$L^k = 2^2$$

Inscribed



$$L^k = 2^3$$



DOE – How to explore the design space?

Central Composite (CC)

```
% Design of Experiment (DOE)
```

```
% Lower/Upper Bounds for DOE
```

```
p1_min = 2;
```

```
p1_max = 5;
```

```
p2_min = 2;
```

```
p2_max = 5;
```

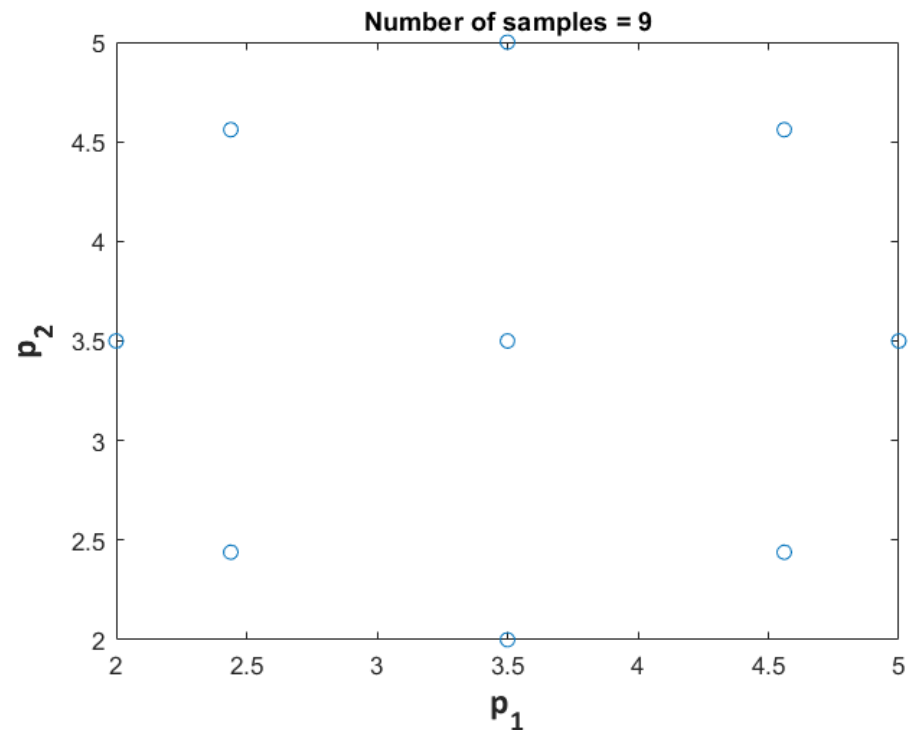
```
% Parameters for Central Composite
```

```
%type='circumscribed';
```

```
type='inscribed';
```

```
%type='faced';
```

```
matrix_points = ccdesign(2, 'type', type)*0.5+0.5;
```

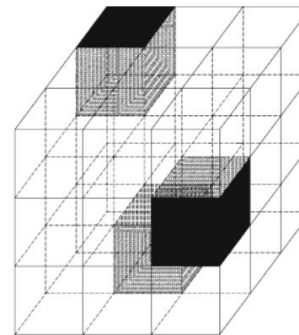
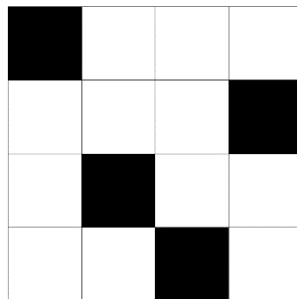


DOE – How to explore the design space?

Latin Hypercube (LH)

- In LH the design space is subdivided into an orthogonal grid with N elements per parameter.
- Within the grid N sub-volumes are located so that along each row and column of the grid only one sub-volume is chosen.
- Inside each sub-volume a sample is chosen randomly. Sub-volumes must be chosen in order to minimize spurious correlations

$k = 2, N = 4$



$k = 3, N = 3$

DOE – How to explore the design space?

Latin Hypercube (LH)

```
% Design of Experiment (DOE)
```

```
% Lower/Upper Bounds for DOE
```

```
p1_min = 2;
```

```
p1_max = 5;
```

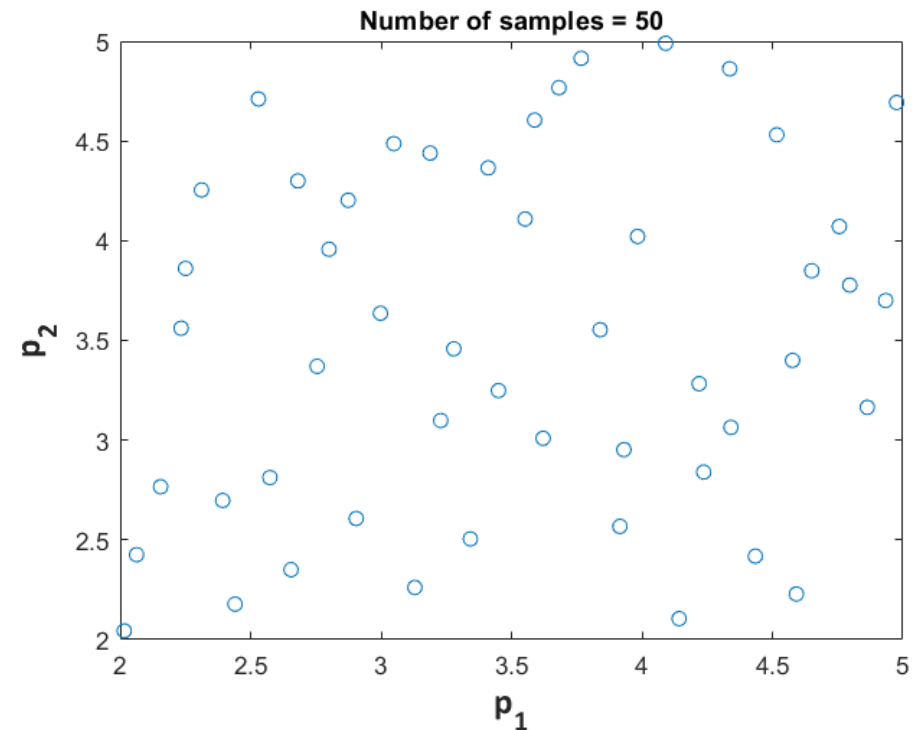
```
p2_min = 2;
```

```
p2_max = 5;
```

```
% Parameters for Latin Hypercube
```

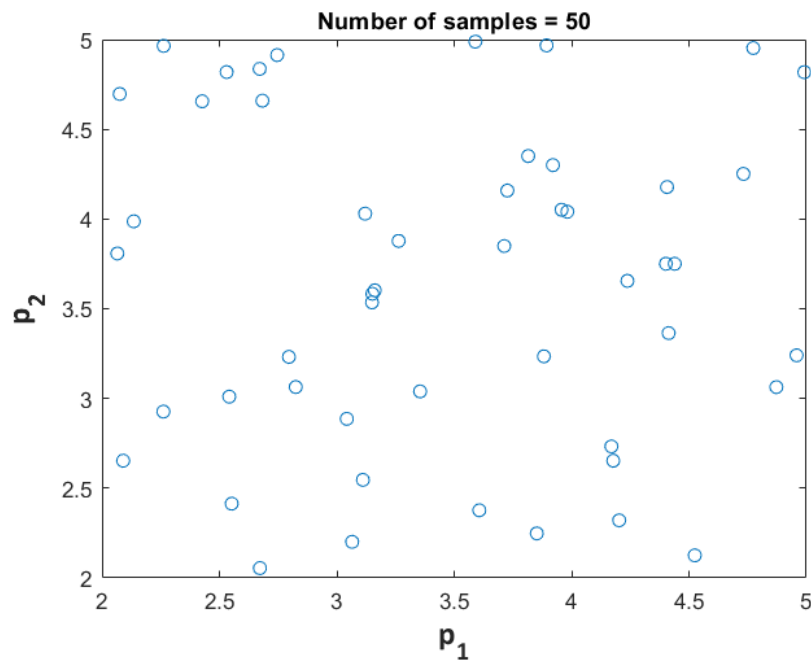
```
N_Sampl_LH=50;
```

```
matrix points = lhsdesign(N_Sampl_LH,2);
```



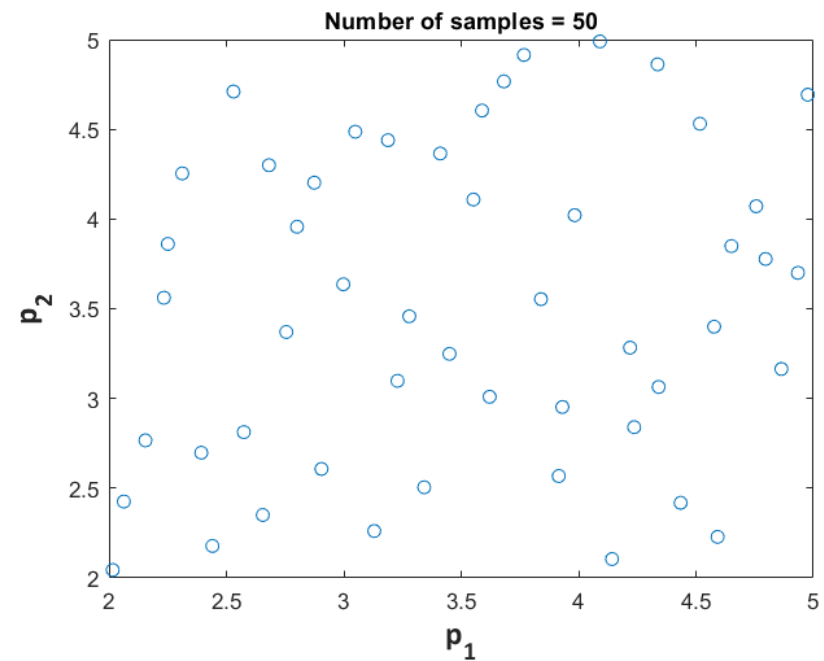
DOE – How to explore the design space?

RANDOM DOE



Many “areas” of the design space are not studied

Latin Hypercube DOE



It ensures a better distribution of the samples

Many DOE techniques are available. The best choice depends on the nature of the problem to be investigated. The cost of the DOE grows exponentially with N , k , and L . It is important to consider:

- the number of experiments N
- the number of parameters k : unless very high N can be afforded it is good practice to start from a preliminary study of the main effects in order to reduce k in the following analysis
- the number of levels L for each parameter: much depends on the expected regularity of the response variable. A L levels method roughly allows a $L-1$ order response surface to be built
- the aim of the DOE: For building a metamodel FF and CC could be fine, but in general space-filling techniques (LH) behave better for irregular or unknown responses

Other interesting Space Filling DOE:

Sobol → **Sobolset**

Halton → **Haltonset**

<https://it.mathworks.com/help/stats/design-of-experiments-1.html>

What about the RSM?

Many RSM techniques are available:

- Kriging
- Radial Basis Functions
- Neural Networks

Detailed mathematical formalism as well as comparative examples and guidelines for proper selection in design problems can be found in

Cavazzuti, M. "Optimization methods: from theory to design scientific and technological aspects in mechanics". Springer Science & Business Media, 2012.

To conclude our exercises, let us consider the Matlab Fit tool:

```
% Fitting  
%ft='linearinterp';  
ft='cubicinterp';  
%ft='poly23';
```

This algorithm works with curve (1 variable) or surface (2 variables) data

```
[myRSM,gof]=fit([x,y],z,ft,'Normalize','on');
```



To sum up:

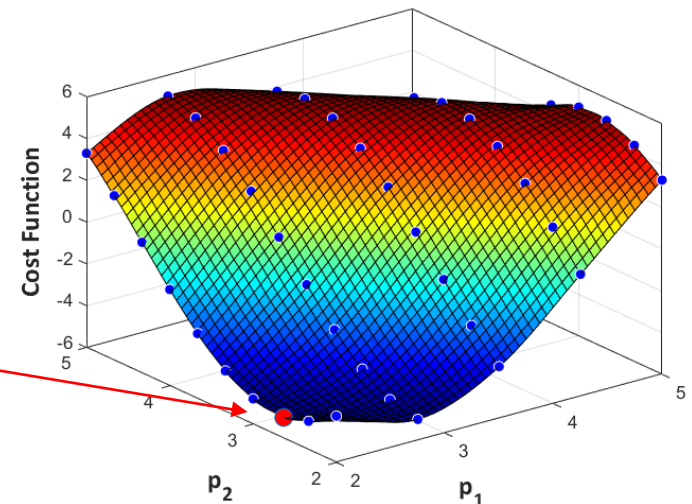
DOE+RSM are most likely to be used to:

- Build performance maps of our mechanical system.
- Find out the analytical correlation between a set of design variables and the objective function(s).

$$f(x) = \alpha(x_1 - \beta)x_2 + \dots$$

- Perform Optimization Studies

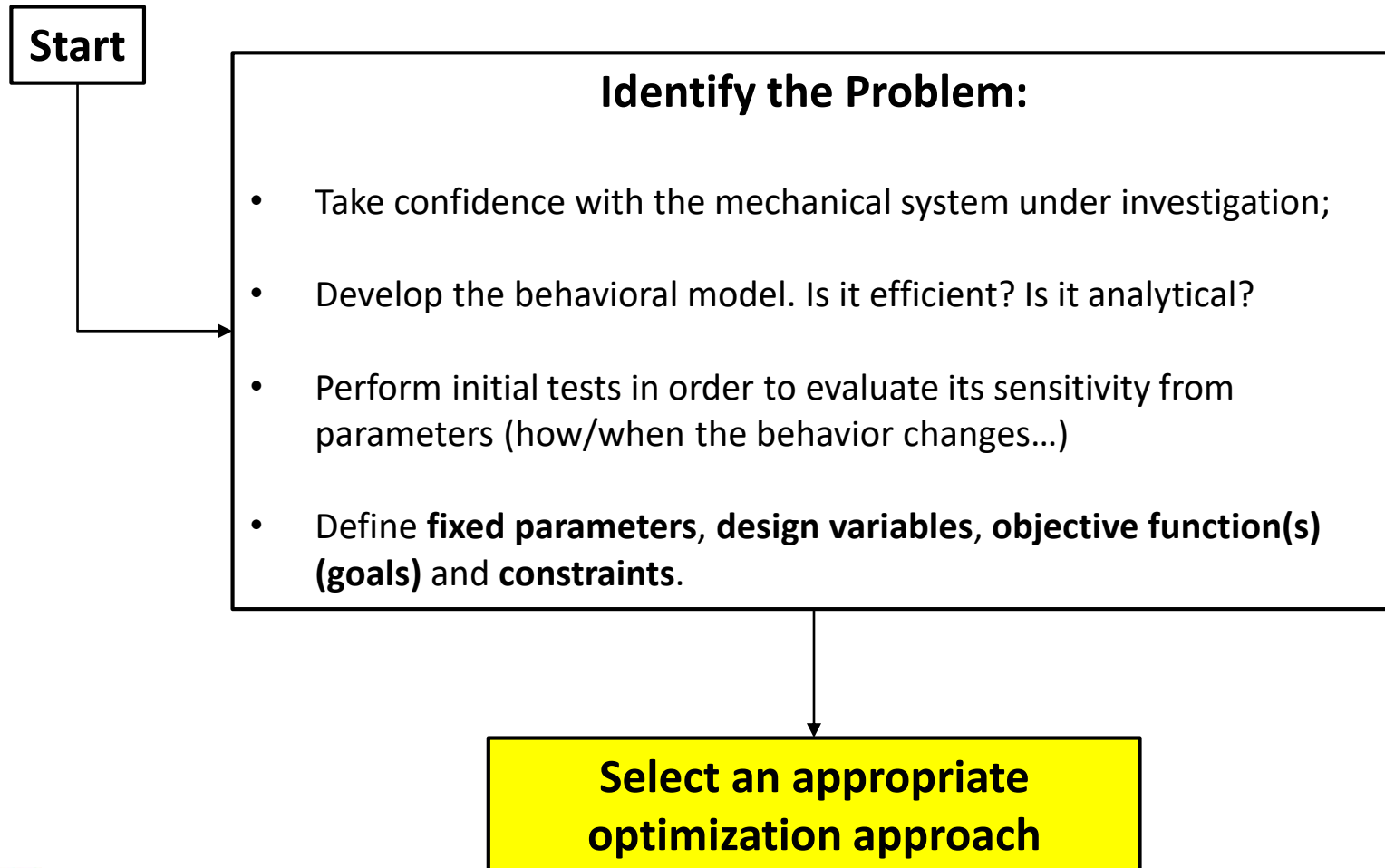
DO or SO can be applied to find the minimum



We should now explore our knowledge...



Optimization Problem Definition:



How to select the optimization approach?

Let us take a look at:

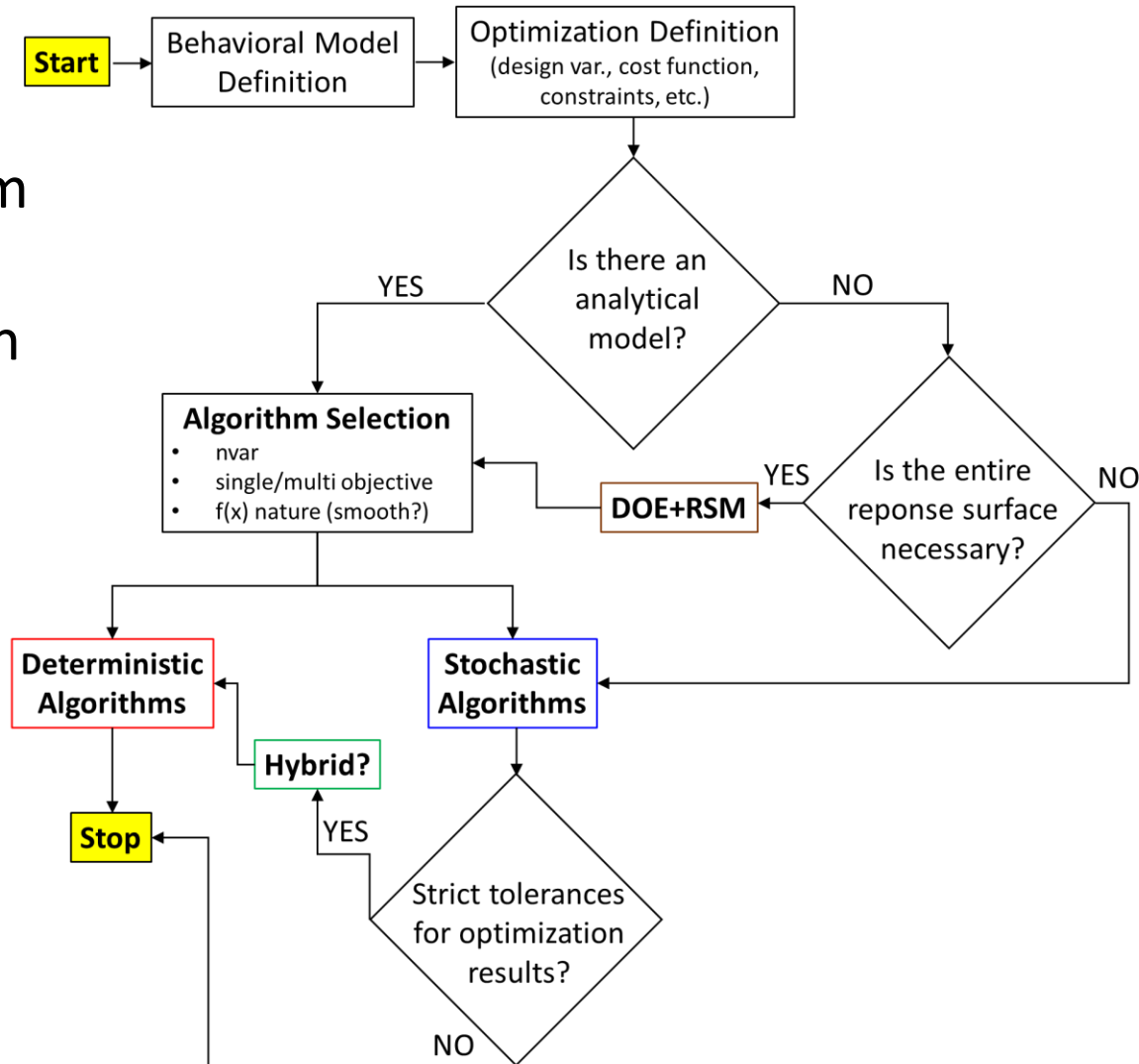
1. Number of declared **design variables**
2. Number of declared **objective functions** (single objective Vs multi-objective optimization problem)
3. Computational efficiency (time required to solve a single simulation)
4. Personal skills with theories and/or software tools →

Where/how
did I build my
model?



Logical Schematic:

- The optimization problem and the design intents should have already been defined
- The user should know where to develop the virtual model



Modeling Technique:

**MECHANICAL
SYSTEM
(e.g. Archimedean
Spiral)**

**Modeling
Technique**

**Are Theoretical approaches
(equations) available?**

To calculate the rotational stiffness, the length of the flexible element must be determined. As highlighted in Fig. 2, the central portion of the spiral is considered part of the rigid inner ring. Therefore, with respect to the central coordinate system, the characteristic spring length is given by

$$L_{sp} = L_{ext} - L_{in} \tag{7}$$

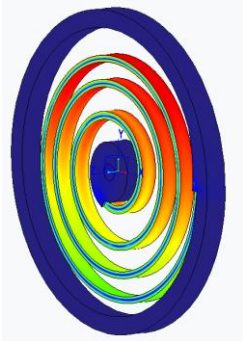
where

$$L_{ext} = \frac{P}{4\pi} \left[\alpha_{max} \sqrt{1 + \alpha_{max}^2} + \ln \left(\alpha_{max} \sqrt{1 + \alpha_{max}^2} \right) \right] \tag{8}$$
$$L_{in} = \frac{P}{4\pi} \left[\alpha_{min} \sqrt{1 + \alpha_{min}^2} + \ln \left(\alpha_{min} \sqrt{1 + \alpha_{min}^2} \right) \right] \tag{9}$$

**ALWAYS DESIRABLE
to speed up the process**

Do I need Numerical approaches?

- FEA
- Multibody
- etc.



Modeling/Optimization Tool(s):

Which environments are **SURELY** involved?

- **OPTIMIZER** → Main Env. (from which the entire simulation process is overseen)

In addition, we **MAY** need:

- Computer Aided Design (CAD) tools → Generate/Update the geometry
- Computer Aided Engineering (CAE) tools → Simulation



Let us consider commercial environments:



Optimizer
Programming
~~CAD~~
~~CAE~~



Optimizer
Programming (macro)
CAD
CAE



Optimizer
Programming
CAD
CAE

Software	Modeling/Simulation	Optimizer
MATLAB	Theoretical Models (equations coded)	Powerful
PTC CREO	Parametric CAD with CAE packages	Very limited options
ANSYS	Dedicated CAE software with CAD module	Moderate

When to choose a single environment for the modeling & optimization?



1) My system can be modelled with equations



1) No equations available

2) I need a parametric CAD

3) I need to solve relatively simple simulations

4) I do not need complex and programmable optimization algorithms



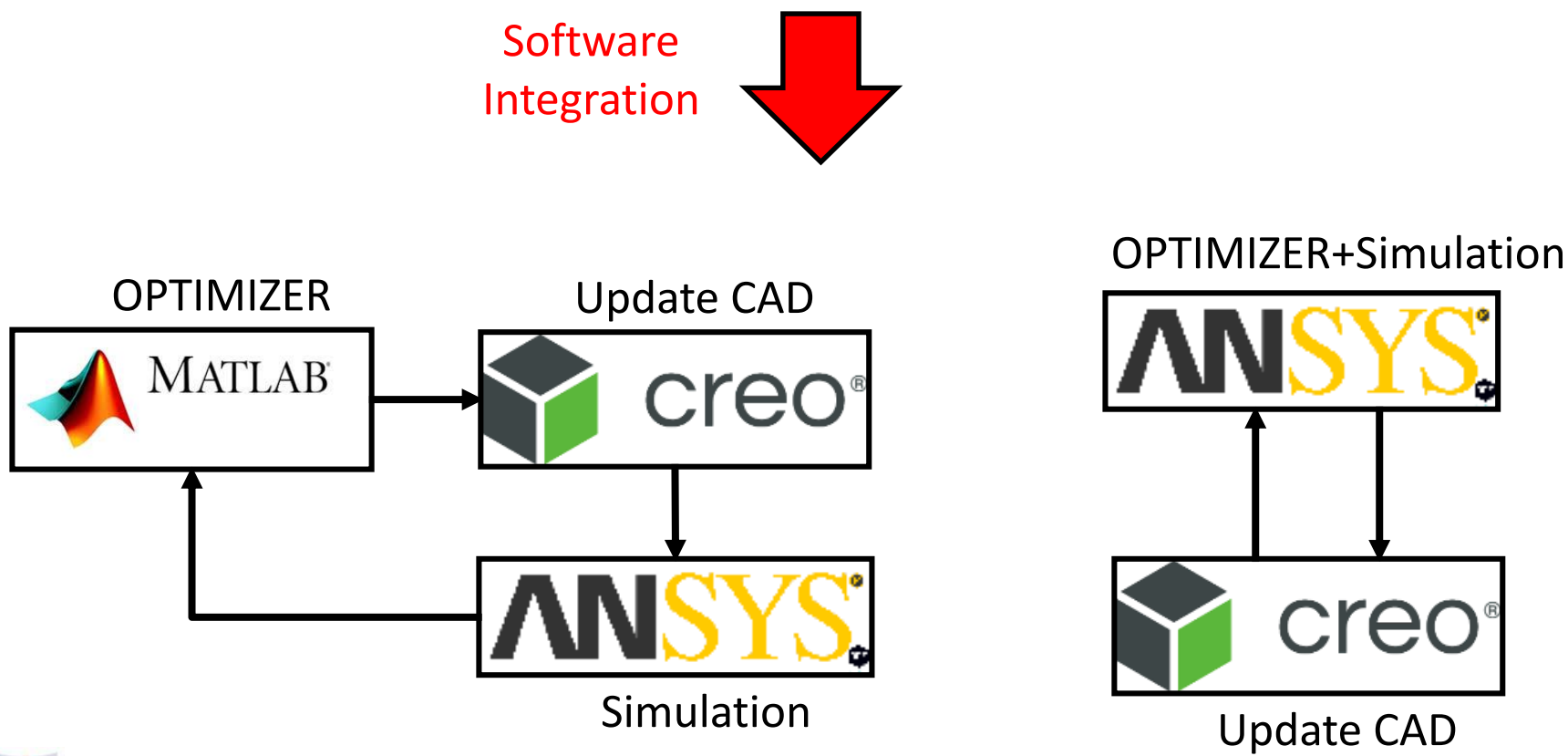
1) No equations available

2) I need to solve complex simulations

3) I need a specific algorithm (DOE+RSM)



What if my problem needs BOTH dedicated modeling features and robust optimizer?



Example – Design of a Archimedean Spiral

Input Parameters:

Principal Dimensions $\left\{ \begin{array}{l} R_{in} = 10\text{mm} \\ R_{ext} = 50\text{mm} \end{array} \right.$

Coils $\rightarrow n_t = 4$

ABS Plastic $\rightarrow E = 1800\text{MPa}$

Deflection $\rightarrow \theta = 0.7\text{rad}$

Design variables

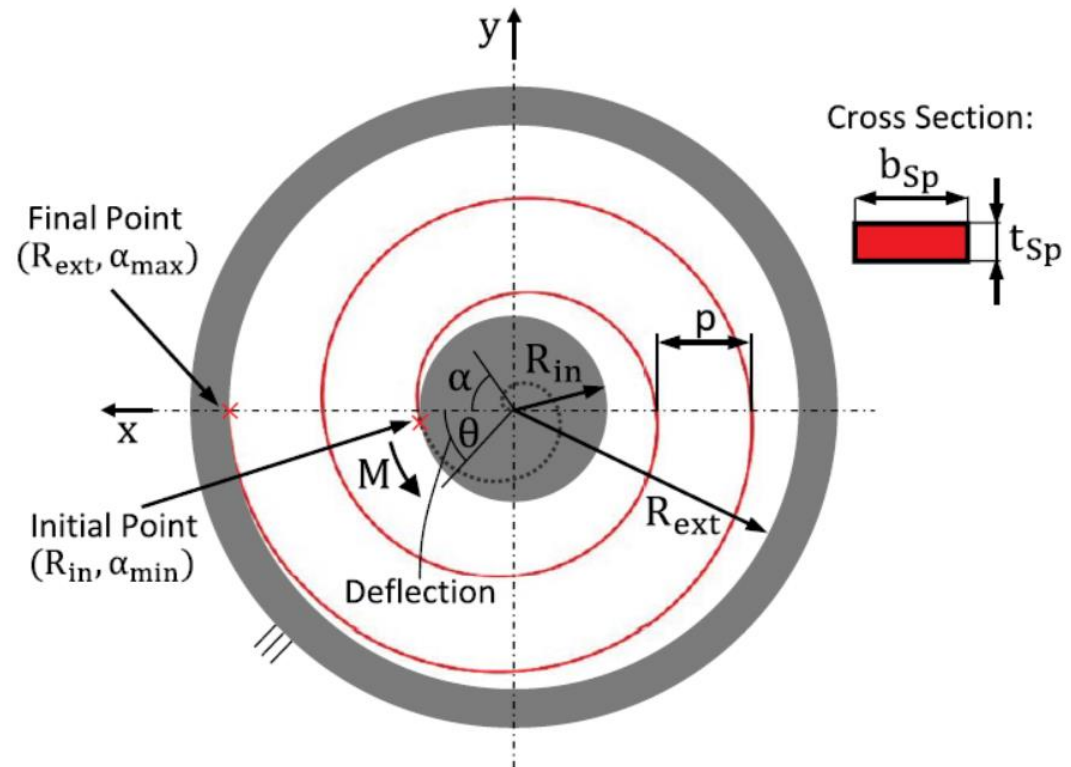
b_{sp}, t_{sp}

Cost Function

$$e_k = |k_{sp} - k_{target}|$$

Constraint

$$\sigma < \sigma_{max}$$



Analytical Model:

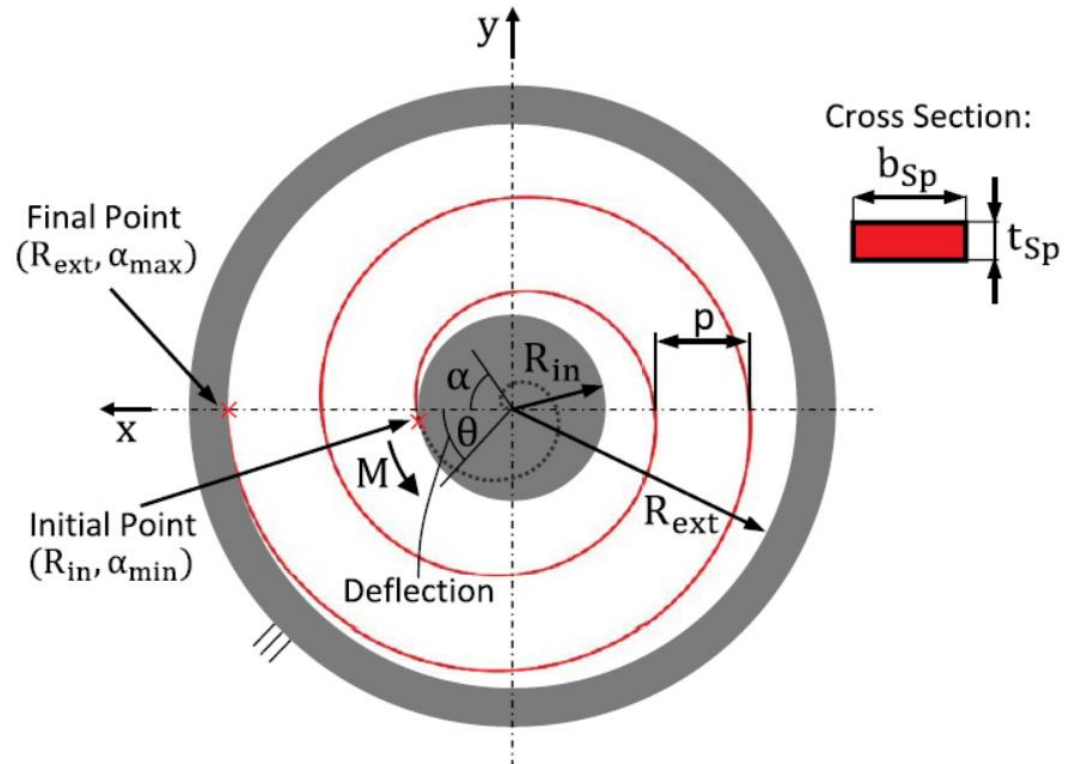
Geometrical
Parameters

$$\alpha_{\max} = 2\pi n_t$$
$$p = \frac{2\pi R_{\text{ext}}}{\alpha_{\max}}$$
$$\alpha_{\min} = \frac{2\pi R_{\text{in}}}{\alpha_{\max}}$$

Spiral
Geometry

$$R(\alpha) = \frac{p\alpha}{2\pi}$$
$$x = R \cos(\alpha)$$
$$y = R \sin(\alpha)$$

$$L_{\text{Spiral}} \rightarrow k_{\text{Sp}} = \frac{E_{\text{Sp}} b_{\text{Sp}} t_{\text{Sp}}^3}{12 L_{\text{Sp}}} \longrightarrow M_{\text{Sp}} = k_{\text{Sp}} \theta \longrightarrow \sigma_{\max} = \frac{6 M_{\text{Sp}}}{b_{\text{Sp}} t_{\text{Sp}}^2}$$

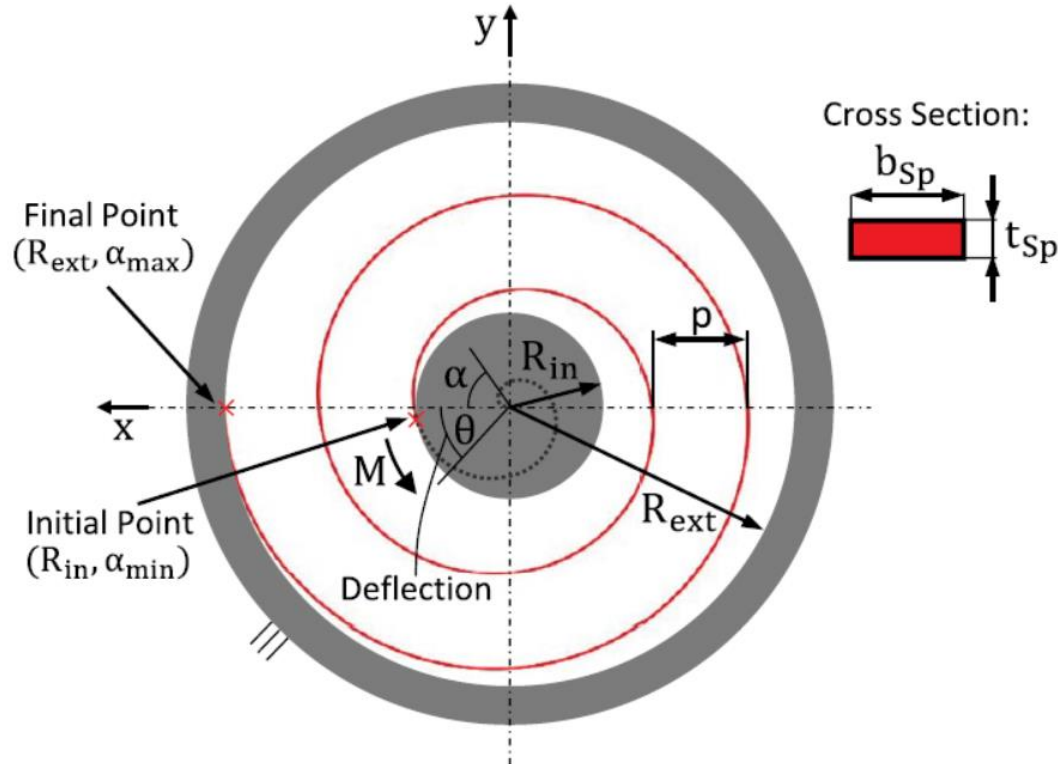


Proposed Exercises:

1. Matlab modeling + optimization
2. Creo CAD modeling + Internal CAE simulation + Internal optimization
3. Creo CAD modeling + ANSYS WB CAE simulation + ANSYS WB optimization
4. ANSYS APDL CAE Modeling + Matlab optimization



Molla torsionale – Spirale di Archimede:



Parametri Costruttivi Fissati:

$$\begin{aligned} R_{in} &= 10\text{mm} \\ R_{ext} &= 50\text{mm} \\ n_t &= 4 \end{aligned}$$

Materiale:

$$\begin{aligned} E &= 200000\text{MPa} \\ \nu &= 0.33 \end{aligned}$$

Deflessione Richiesta in esercizio:

$$\theta = 0.7\text{rad}$$

Si richiede di valutare la rigidezza torsionale e lo stress al variare della sezione resistente:

Variabili di progetto

$$b_{sp}, t_{sp}$$

Funzioni di interesse

$$k_{sp}(b_{sp}, t_{sp})$$

$$\sigma_{max}(b_{sp}, t_{sp})$$

Molla torsionale – Spirale di Archimede:

Come costruire questo modello con gli strumenti che conosco? Ho una vasta scelta...

1) Modello teorico



Esiste un riferimento teorico (equazioni)?



Sì, se il numero di spire è sufficientemente elevato (≥ 4) e non sono richieste deflessioni molto elevate (grandi sì, esagerate no...)

2) Modello FEM

2a)

CAD + FEM interno



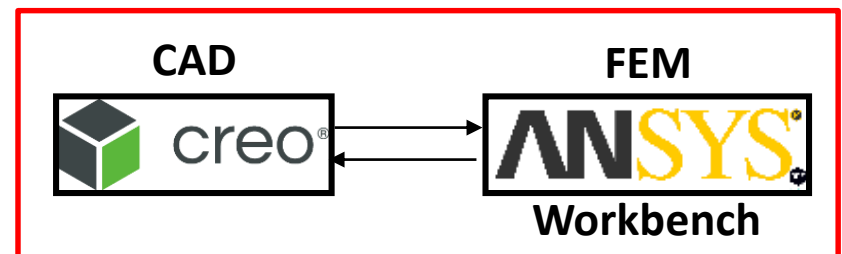
2b)

Geometria + FEM



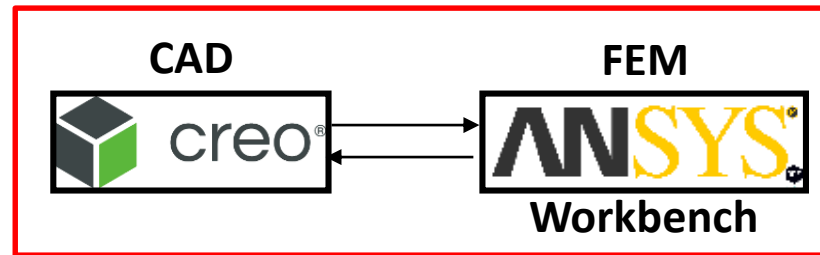
APDL oppure Workbench

2c)

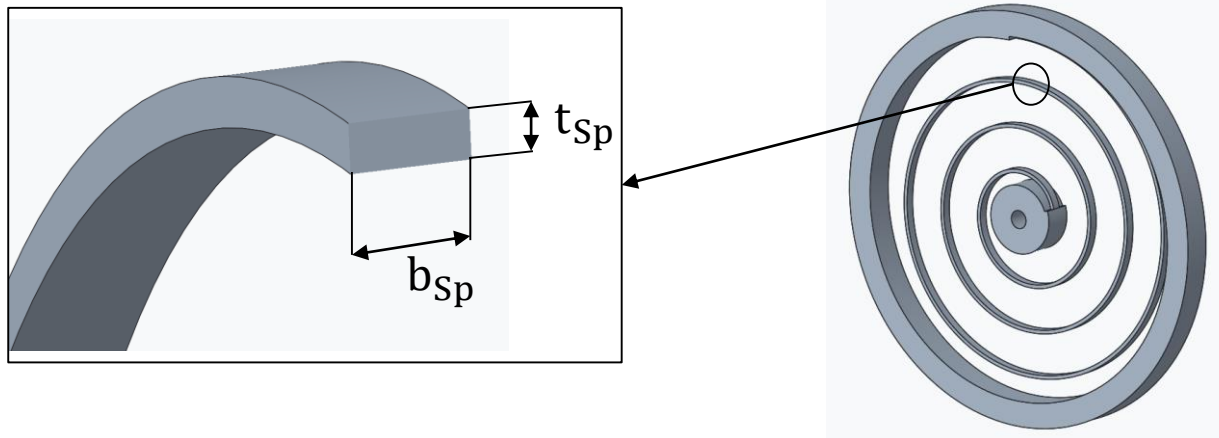


Molla torsionale – Spirale di Archimede:

Supponiamo di avere un modello CAD già sviluppato e di voler impostare uno studio parametrico veloce con l'ausilio di Workbench...



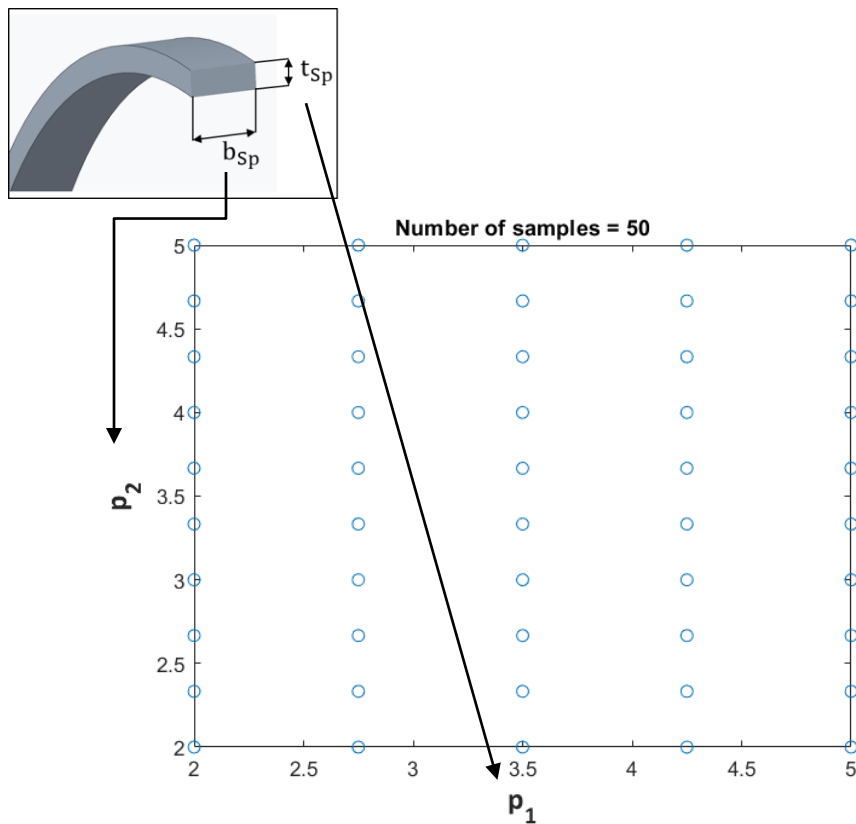
Si vuole osservare come varia il comportamento della spirale al variare delle dimensioni della sezione resistente:



Molla torsionale – Spirale di Archimede:

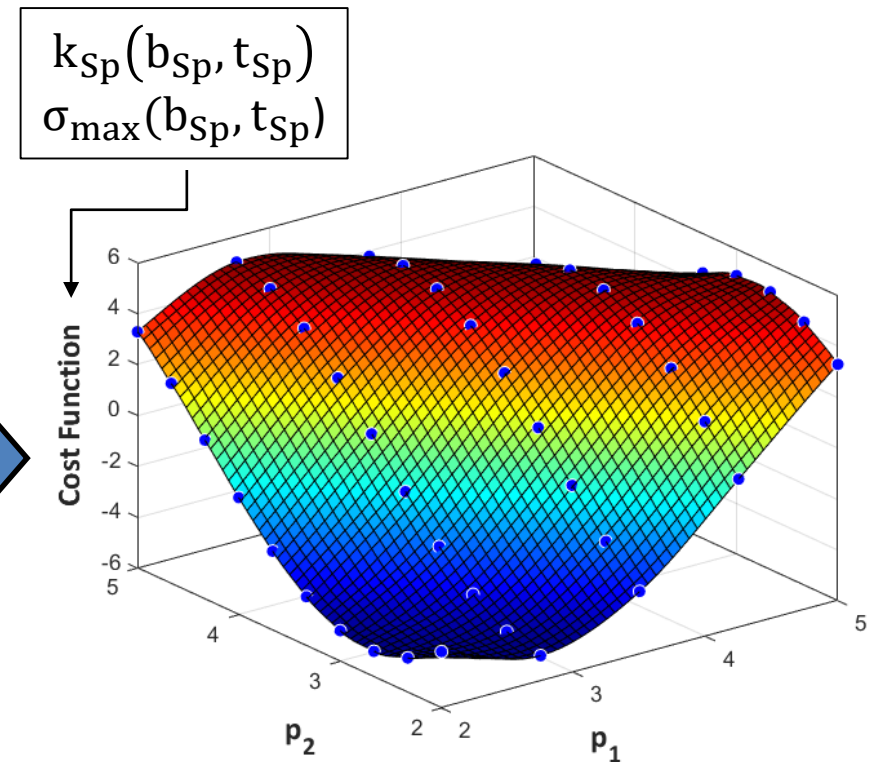
Fase 1 – Design of Experiments

Si esplora il dominio effettuando una serie di simulazioni “mirate”



Fase 2 – Fitting dei risultati

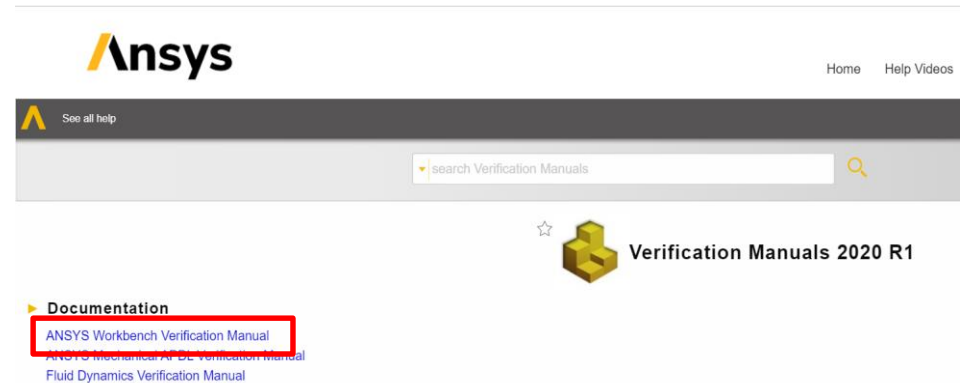
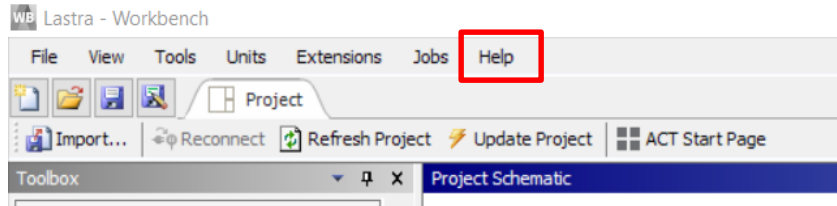
Si ricavano le superfici di risposta (correlazioni analitiche tra le variabili e funzioni di interesse)



Alcuni tutorials utili per ANSYS WB:

- **ANSYS Verification Manuals**

https://ansyshelp.ansys.com/account/secured?returnurl=/Views/Secured/prod_page.html?pn=Verification%20Manuals&lang=en



- **Cornell University**

<https://confluence.cornell.edu/display/SIMULATION/ANSYS+Learning+Modules>

- **Università di Pisa (Prof. Ciro Santus)**

<http://people.unipi.it/static/ciro.santus/Didattica.html>

- **Link a tutorial selezionati per funzionalità specifiche**



SI VEDA CARTELLA FORNITA



References:

- Cavazzuti, M. "Optimization methods: from theory to design scientific and technological aspects in mechanics". Springer Science & Business Media, 2012.
- Matlab Optimization Toolbox: <https://it.mathworks.com/products/optimization.html>, assessed 05/28/2020.
- Park, H. S., Dang, X. P. "Structural optimization based on CAD–CAE integration and metamodeling techniques". Computer-Aided Design, 42(10), pp. 889-902, 2010.
- Jutte, C. V., Kota, S. "Design of nonlinear springs for prescribed load-displacement functions". Journal of Mechanical Design, 130(8), 2008.
- Berselli, G., Balugani, F., Pellicciari, M., Gadaleta, M. "Energy-optimal motions for Servo-Systems: A comparison of spline interpolants and performance indexes using a CAD-based approach". Robotics and Computer-Integrated Manufacturing, 40, 55-65, 2016.
- Bilancia, P., Berselli, G., Bruzzone, L., Fanghella, P. "A CAD/CAE Integration Framework for Analyzing and Designing Spatial Compliant Mechanisms via Pseudo-Rigid-Body Methods". Robotics and Computer-Integrated Manufacturing, 56, pp. 287-302, 2019.
- Gadaleta, M., Pellicciari, M., Berselli, G. "Optimization of the energy consumption of industrial robots for automatic code generation". Robotics and Computer-Integrated Manufacturing, 57, 452-464, 2019.
- Bilancia, P., Smith, S. P., Berselli, G., Magleby, S., Howell, L. "Zero Torque Compliant Mechanisms Employing Pre-buckled Beams". Journal of Mechanical Design, 142(11), 2020.
- Bilancia, P., Berselli, G. "Design and Testing of a Monolithic Compliant Constant Force Mechanism". Smart Material and Structures, 29(4), 44001, 2020.
- Bilancia, P., Berselli, G., Magleby, S., Howell, L. "On the Modeling of a Contact-Aided Cross-Axis Flexural Pivot". Mechanism and Machine Theory, 143, 103618, 2020.
- Bilancia, P., Berselli, G., Palli, G. "Virtual and Physical Prototyping of a Beam-Based Variable Stiffness Actuator for Safe Human-Machine Interaction". Robotics and Computer-Integrated Manufacturing, 65, 101886, 2020.



THANK YOU

Questions? Suggestions?