Optimum Design: Numerical Solution Process

- Introduction to numerical search methods
- Optimum design: aspects of problem formulation
- Numerical solution process for optimum design
- EXCEL: Solver
- MATLAB: Optimization Toolbox
- Mathematica: Optimization Toolbox

Numerical Search Methods (1)

- Graphical method
  - Two-variable problems only
- Approach to solve optimality conditions
  - Difficult to use when the number of variables and/or the number of constraints is greater than three
  - Leads to a set of nonlinear equations that needs to be solved using a numerical method anyway
- Numerical methods
  - Can handle many variables and constraints, as well as directly search for optimum points
  - Start with an initial design estimate
  - Search the feasible set for optimum designs
  - Derivative-Based Methods / Direct Search Methods / Derivative-Free Methods / Nature-Inspired Search Methods

Numerical Search Methods (2)

- Derivative-Based Methods: gradient-based search methods (Ch.10~13)
  - All functions: continuous and at least twice continuously differentiable
  - Accurate first-order derivatives of all the functions are available
  - Design variables: assumed to be continuous within their allowable range
  - Extensively developed since the 1950s, and many good ones are available to solve smooth nonlinear optimization problems
  - Always converge to a local minimum point only, global solutions?

$$\begin{cases} \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)};\ k = 0,1,2,\ldots \\ \text{where} \begin{cases} \mathbf{x}^{(0)}:\ \text{starting design point} \\ \Delta\mathbf{x}^{(k)} = \alpha_k \mathbf{d}^{(k)} \xrightarrow{\text{two separate subproblems}} \begin{cases} \alpha_k > 0:\ \text{step size} \\ \mathbf{d}^{(k)}:\ \text{search direction} \end{cases} \end{cases} \\ x_i^{(k+1)} = x_i^{(k)} + \Delta x_i^{(k)};\ k = 0,1,2,\ldots;\ i = 1,\ldots,n \end{cases}$$

Numerical Search Methods (3)

- Direct Search Methods (Ch.11.9)
  - Do not calculate/use/approximate derivatives of the problem functions
  - Functions are assumed to be continuous and differentiable; however, their derivatives are either unavailable or not trustworthy
  - Only functions' values are calculated and used in the search process
  - Hooke–Jeeves and Nelder–Mead in 1960s and 1970s
  - Simplicity and ease of use
- Derivative-Free Methods
  - Do not require explicit calculation of analytical derivatives of the functions
  - Approximation of derivatives is used to construct a local model: finite difference approach
  - *Response surface methods* that generate approximation for complex optimization functions

Numerical Search Methods (4)

- Nature-Inspired Search Methods
  - Use only the values of the problem functions
  - Classified as direct search methods
  - Use statistical concepts and random numbers to advance the search toward a solution point
    - Simulated annealing (Ch.15.5)
    - Genetic algorithms (Ch.17.1)
  - Quite general: solve all kinds of optimization problems
  - Quite time-consuming: require a large number of function evaluations to reach an acceptable solution
  - Do not have a good stopping criterion (no optimality conditions)

Selection of a Method

- Are the design variables continuous (can have any value in their range), discrete (must be selected from a specified list) or integer?
- Are the problem functions continuous and differentiable?
- Are derivatives of all the problem functions available (can be calculated efficiently)?

General Guidelines

- Formulation of a design task as an optimization problem
  - Define a realistic model for the engineering system
  - Use designer's engineering knowledge, intuition, and experience
- Generate a mathematical optimization model
  - In an initial formulation of the problem, all of the possible parameters should be viewed as potential design variables
  - The *existence of an optimum solution* to a design optimization model depends on its formulation
  - The problem of optimizing more than one objective functions simultaneously (*multi-objective problems*) can be transformed into the standard problem
  - In general, *it is desirable to normalize all of the constraints with respect to their limit values*

Scaling of Constraints

- In numerical calculations, it is impossible to require
  - An equality constraint to be precisely equal to zero
  - An active inequality constraint to be precisely equal to zero

$$h(\mathbf{x}) = 0 \rightarrow |h(\mathbf{x})| \leq \varepsilon$$

$$g(\mathbf{x}) \leq 0 \rightarrow |g(\mathbf{x})| \leq \varepsilon$$

$$\varepsilon : \text{ feasibility tolerance}$$

- Different constraints can involve different orders of magnitude → *constraint normalization*

$$\sigma \leq \sigma_a \rightarrow g_1 = \sigma - \sigma_a \leq 0 \rightarrow g_1 = \frac{\sigma}{\sigma_a} - 1 \leq 0 \rightarrow g_1 = R - 1 \leq 0$$

$$\delta \leq \delta_a \rightarrow g_2 = \delta - \delta_a \leq 0 \rightarrow g_2 = \frac{\delta}{\delta_a} - 1 \leq 0 \rightarrow g_2 = R - 1 \leq 0$$

- Some constraints that cannot be normalized: 0 ≤ x

Constraint Normalization

$$\left.\begin{aligned} &h(x_1,x_2) = x_1^{\,2} + \frac{1}{2}x_2 - 18 = 0 \\ &\left|h(4.0,4.2)\right| = \left|+0.1\right| > \varepsilon(0.01) \\ &\left|h(-4.5,-4.8)\right| = \left|-0.15\right| > \varepsilon(0.01) \end{aligned}\right\} \rightarrow \left\{\begin{aligned} &\bar{h}(x_1,x_2) = \frac{1}{18}x_1^{\,2} + \frac{1}{36}x_2 - 1 = 0 \\ &\left|\bar{h}(4.0,4.2)\right| = \left|+0.0056\right| < \varepsilon(0.01) \\ &\left|\bar{h}(-4.5,-4.8)\right| = \left|-0.0083\right| < \varepsilon(0.01) \end{aligned}\right.$$

$$\frac{\partial h}{\partial x_1} = 2x_1 \rightarrow \frac{\partial \bar{h}}{\partial x_1} = \frac{1}{18}(2x_1)$$

$$g(x_1,x_2) = 500x_1 - 30{,}000x_2 \le 0 \rightarrow \left\{\begin{aligned} &\xrightarrow[30{,}000|x_2|]{\text{divided by}} \bar{g}(x_1,x_2) = \frac{x_1}{60|x_2|} - \frac{x_2}{|x_2|} \le 0 \\ &\bar{g}(x_1,x_2) = \frac{1}{60}x_1 - x_2 \le 0 \end{aligned}\right.$$

$$\left.\begin{aligned} &g(80,1) = 10{,}000 > \varepsilon(0.01) \\ &g(60,0.995) = 150 > \varepsilon(0.01) \end{aligned}\right\} \rightarrow \left\{\begin{aligned} &\bar{g}(80,1) = 0.33 > \varepsilon(0.01) \\ &\bar{g}(60,0.995) = 0.005 < \varepsilon(0.01) \end{aligned}\right.$$

Scaling of Design Variables

$$(1)\ a \le x \le b \leftrightarrow -1 \le y \le 1$$

$$(b-a):2=(x-a):(y+1) \rightarrow y=-1+\frac{2(x-a)}{b-a}=\frac{2}{b-a}x-\frac{b+a}{b-a}$$

$$(2)\ a \le x \le b \leftrightarrow \frac{a}{b} \le \frac{x}{b} \le 1 \xrightarrow{y=x/b} \frac{a}{b} \le y \le 1$$

$$(3)\ a \le x \le b \xleftrightarrow[y=1]{x=\frac{a+b}{2}} \frac{a}{\frac{a+b}{2}} \le \frac{x}{\frac{a+b}{2}} \le \frac{b}{\frac{a+b}{2}} \xleftrightarrow{y=\frac{2x}{a+b}} \frac{2a}{a+b} \le y \le \frac{2b}{a+b}$$

$$(4)\ \begin{cases} x_1 \sim O(10^5) \rightarrow \dfrac{x_1}{10^5}=y_1 \\ x_2 \sim O(10^{-5}) \rightarrow \dfrac{x_2}{10^{-5}}=y_2 \end{cases}$$

$$\frac{\partial f}{\partial y}=\frac{\partial f}{\partial x}\frac{\partial x}{\partial y}$$

Iterative Process for Development of Problem Formulation

- Many practical applications are complex requiring repeated updating of the initial formulation of the problem
  - Some of the practical constraints may have been missed
  - Limits for some of the constraints may not be realistic
  - There may be conflicting constraints in the formulation
  - Constraint limits may be too severe such that there is no feasible solution for the problem

Numerical Solution Process

- Optimization algorithm for smooth problems
  - Calculation of cost and constraint functions and their gradients at the current point
  - Definition of a subproblem
    - Determine the search direction
    - Step size determination in the search direction
  - Update the current design point

- General purpose software: integration of
  - Problem functions
  - Gradient evaluation software
  - Optimization software

A Feasible Point Cannot Be Obtained

- Check the formulation to ensure that the constraints are formulated properly and that there are no inconsistencies in them
- Scale the constraints if they have different orders of magnitude
- Check the feasibility of individual constraints or a subset of constraints while ignoring the remaining ones
- Ensure that the formulation and data are properly transferred to the optimization software
- Constraint limits may be too severe
- Check the constraint feasibility tolerance
- Check derivation and implementation of the gradients of the constraint functions
- Increase precision of all calculations, if possible

Algorithm Does Not Converge (1)

- Check the formulation to ensure that the constraints and the cost function are formulated properly
- Ensure that all of the functions are continuous and differentiable for a smooth optimization algorithm
- Scale the constraints and the cost function if they have different orders of magnitude
- Check implementation of the cost function and the constraint functions evaluations
- Check the derivation and implementation of the gradients of all of the functions, If the gradients are evaluated using finite differences, then their accuracy needs to be verified
- Examine the final point reported by the program
- If an overflow of calculations is reported, the problem may be unbounded

Algorithm Does Not Converge (2)

- Try different starting points
- Ignore some of the constraints and solve the resulting problem
- Use a smaller limit on the number of iterations and restart the algorithm with the final point of the previous run of the program as the starting point
- If two design variables are of differing orders of magnitude, scale them so that the scaled variables have the same order of magnitude
- Ensure that the optimization algorithm has been proven to converge to a local minimum point starting from any initial point
- Increase the precision of the calculations, if possible

EXCEL Solver(해찾기)

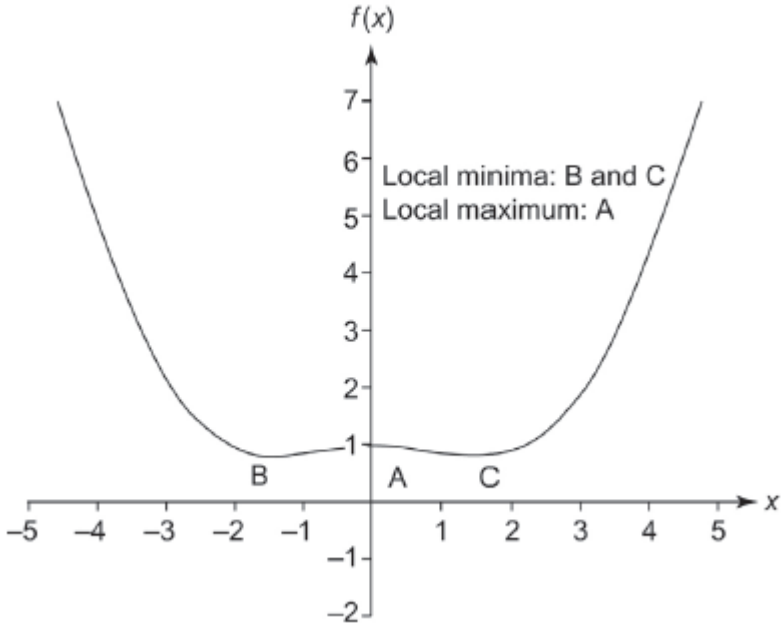
- Introduction
- Roots of a nonlinear equation
- Roots of a set of nonlinear equation
- Unconstrained optimization problems
- Linear programming problems
- Nonlinear programming
  - Optimum design of spring
  - Optimum design of plate girders

Example 4.22

- Numerical solution for the first-order necessary conditions

$$f(x)=\frac{1}{3}x^2+\cos x \rightarrow f'(x)=\frac{2}{3}x-\sin x=0$$

$$\begin{cases} x^*=0,\ f(0)=1 \\ x^*=1.496,\ f(1.496)=0.821 \\ x^*=-1.496,\ f(1.496)=0.821 \end{cases}$$



Example 4.31

- Solution of the KKT necessary conditions

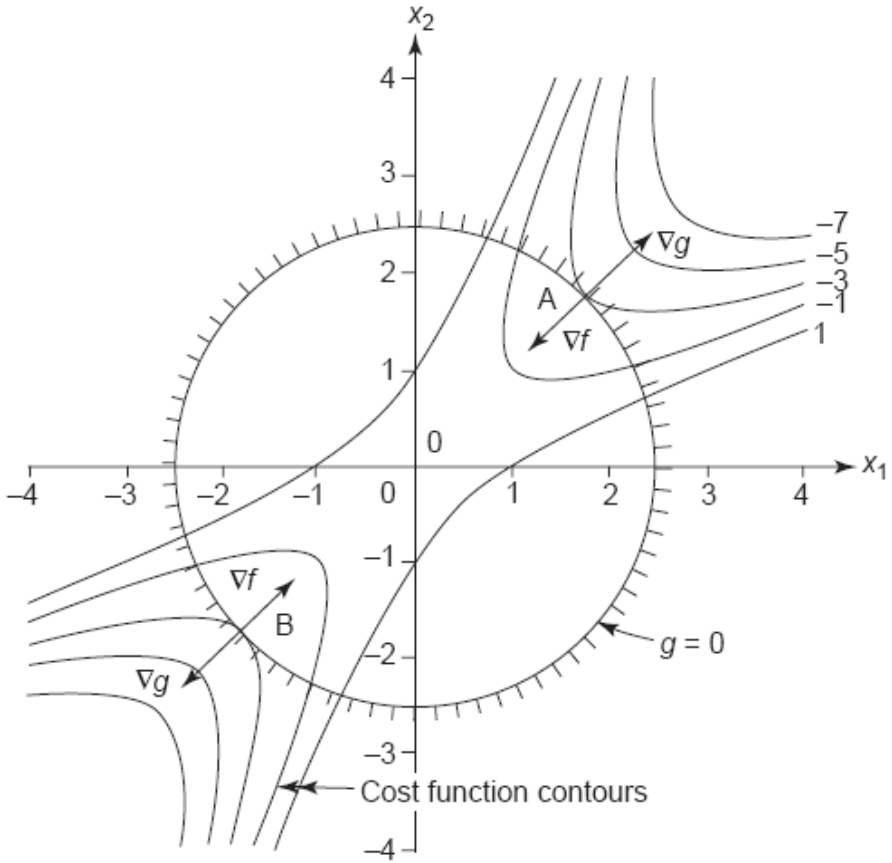
$$\underset{\mathbf{x}}{\text{Minimize}}\ f(\mathbf{x}) = x_1^2 + x_2^2 - 3x_1x_2$$

$$\text{subject to}\ g = x_1^2 + x_2^2 - 6 \le 0$$

$$L = x_1^2 + x_2^2 - 3x_1x_2 + u\left(x_1^2 + x_2^2 - 6 + s^2\right)$$

$$\begin{cases} \dfrac{\partial L}{\partial x_1} = 2x_1 - 3x_2 + 2ux_1 = 0 \\ \dfrac{\partial L}{\partial x_2} = 2x_2 - 3x_1 + 2ux_2 = 0 \\ \dfrac{\partial L}{\partial u} = x_1^2 + x_2^2 - 6 + s^2 = 0 \\ \dfrac{\partial L}{\partial s} = 2us = 0; \quad u \ge 0 \end{cases}$$

$$\rightarrow \begin{cases} x_1^* = x_2^* = 0,\ u^* = 0,\ s = 6,\ f = 0 \\ x_1^* = x_2^* = \sqrt{3},\ u^* = 1/2,\ s = 0, f = -3 \\ x_1^* = x_2^* = -\sqrt{3},\ u^* = 1/2,\ s = 0, f = -3 \end{cases}$$



EXCEL vs. MATLAB

	A	B	C	D	E
1	solution of KKT conditions				
2	variables				
3	x1	1			1.732051
4	x2	-2			1.732051
5	u	2			0.5
6	s	0			0
7	equations				
8	2*x1-3*x2+2*u*x1	12	=	0	-2.5E-07
9	-3*x1+2*x2+2*u*x2	-15	=	0	5.48E-08
10	x1^2+x2^2-6+s^2	-1	=	0	-1.2E-07
11	u*s	0	=	0	0
12	s^2	0	>=	0	0
13	u	2	>=	0	0.5

```
Function F=kktsystem(x)
F=[2*x(1)-3*x(2)+2*x(3)*x(1);
2*x(2)-3*x(1)+2*x(3)*x(2);
x(1)^2+x(2)^2-6+x(4)^2;
x(3)*x(4)];

x0=[1;1;1;1];
options=optimset('Display','iter')
x=fsolve(@kktsystem,x0,options)
```

Unconstrained Optimization Problem

$$\underset{x,y,z}{\text{Min}}\ f(x,y,z) = x^2 + 2y^2 + 2z^2 + 2xy + 2yz$$

	A	B	C	D	E
1	Unconstrained Optimization Problem				
2					
3	Variables	Value			
4	x	2			
5	y	4			
6	z	10			
7					
8	Objective function				
9		=x*x+2*y*y+2*z*z	=2*x*y+2*y*z	=B9+C9	

Solver Parameters

Set Objective: $D$9

To: Max Min Value Of: 0

By Changing Variable Cells:

$B$4:$B$6

Subject to the Constraints:

Add

Change

Delete

Reset All

Load/Save

Make Unconstrained Variables Non-Negative

Select a Solving Method: GRG Nonlinear

Options

Solving Method

Select the GRG Nonlinear engine for Solver Problems that are smooth nonlinear. Select the LP Simplex engine for linear Solver Problems, and select the Evolutionary engine for Solver problems that are non-smooth.

Linear Programming Problem

$$\underset{\mathbf{x}}{\text{Maximize}} \quad z = x_1 + 4x_2$$

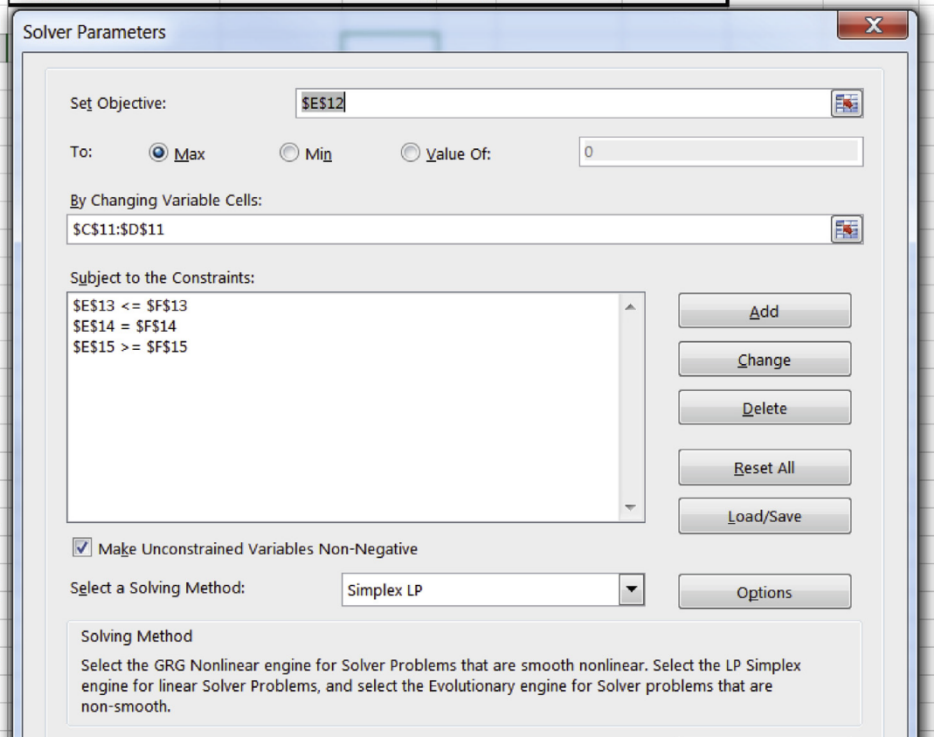
$$\text{subject to} \quad x_1 + 2x_2 \le 5$$

$$2x_1 + x_2 = 4$$

$$x_1 - x_2 \ge 1$$

$$x_1, x_2 \ge 0$$

	A	B	C	D	E	F	G	H
1	Linear programming problem							
3	*Problem* is to maximize	x1+4x2						
4	subject to	x1+2x2<=5						
5		2x1+x2=4						
6		x1-x2>=1						
7		x1, x2>=0						
9	*Problem set up for Solver*							
10	Variables		x1	x2	Sum of LHS	RHS Limit		
11	Variable value		0	0				
12	Objective function: max		1	4	0			
13	Constraint 1		1	2	0	5		
14	Constraint 2		2	1	0	4		
15	Constraint 3		1	-1	0	1		



Answer and Sensitivity Report

Microsoft Excel 15.0 Answer Report
Worksheet: [Figure 6.8(LP)-2015.xlsx]Figure 6.8
Report Created: 7/11/2015 2:51:41 PM
Result: Solver found a solution. All Constraints and optimality conditions are satisfied.
Solver Engine
Engine: Simplex LP
Solution Time: 0 Seconds.
Iterations: 2 Subproblems: 0
Solver Options
Max Time 100 sec, Iterations 100, Precision 0.000001
Solve Without Integer Constraints, Assume NonNegative

Objective Cell (Max)

Cell	Name	Original Value	Final Value
$E$12	Objective function: max Sum of LHS	0	4.333333333

Variable Cells

Cell	Name	Original Value	Final Value	Integer
$C$11	Variable value x1	0	1.666666667	Contin
$D$11	Variable value x2	0	0.666666667	Contin

Constraints

Cell	Name	Cell Value	Formula	Status	Slack
$E$13	Constraint 1 Sum of LHS	3	$E$13<=$F$13	Not Binding	2
$E$14	Constraint 2 Sum of LHS	4	$E$14=$F$14	Binding	0
$E$15	Constraint 3 Sum of LHS	1	$E$15>=$F$15	Binding	0

Microsoft Excel 15.0 Sensitivity Report
Worksheet: [Figure 6.8(LP)-2015.xlsx]Figure 6.8
Report Created: 7/11/2015 2:51:41 PM

Variable Cells

Cell	Name	Final Value	Reduced Cost	Objective Coefficient	Allowable Increase	Allowable Decrease
$C$11	Variable value x1	1.666666667	0	1	7	1E+30
$D$11	Variable value x2	0.666666667	0	4	1E+30	3.5

Constraints

Cell	Name	Final Value	Shadow Price	Constraint R.H. Side	Allowable Increase	Allowable Decrease
$E$13	Constraint 1 Sum of LHS	3	0	5	1E+30	2
$E$14	Constraint 2 Sum of LHS	4	1.666666667	4	2	2
$E$15	Constraint 3 Sum of LHS	1	-2.333333333	1	1	2

Optimum Design with MATLAB

- Introduction to Optimization Toolbox
- Unconstrained optimum design problem
- Constrained optimum design problem
- Optimum design examples
  - Location of maximum shear stress for two spherical bodies in contact
  - Column design for minimum mass
  - Flywheel design for minimum mass

Optimization Toolbox Functions

Problem type	Formulation	MATLAB function
One-variable minimization in fixed interval	Find $x \in [x_L\ x_U]$ to minimize $f(x)$	`fminbnd`
Unconstrained minimization	Find $\mathbf{x}$ to minimize $f(\mathbf{x})$	`fminunc` `fminsearch`
Constrained minimization: Minimize a function subject to linear inequalities and equalities, nonlinear inequalities and equalities, and bounds on the variables	Find $\mathbf{x}$ to minimize $f(\mathbf{x})$ subject to $\mathbf{Ax} \le \mathbf{b}$, $\mathbf{Nx} = \mathbf{e}$ $g_i(\mathbf{x}) \le 0, i = 1$ to m $h_j = 0, j = 1$ to p $x_{iL} \le x_i \le x_{iU}$	`fmincon`
Linear programming: minimize a linear function subject to linear inequalities and equalities	Find $\mathbf{x}$ to minimize $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$ subject to $\mathbf{Ax} \le \mathbf{b}$, $\mathbf{Nx} = \mathbf{e}$	`linprog`
Quadratic programming: Minimize a quadratic function subject to linear inequalities and equalities	Find $\mathbf{x}$ to minimize $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{Hx}$ subject to $\mathbf{Ax} \le \mathbf{b}$, $\mathbf{Nx} = \mathbf{e}$	`quadprog`

Syntax

```
[x,FunValue,ExitFlag,Output]=fminX('ObjFun',...,options)
```

Argument	Description
x	The solution vector or matrix found by the optimization function. If `ExitFlag` > 0 then x is a solution, otherwise x is the latest value from the optimization routine.
`FunValue`	Value of the objective function, `ObjFun`, at the solution x.
`ExitFlag`	The exit condition for the optimization function. If `ExitFlag` is positive then the optimization routine converged to a solution x. If `ExitFlag` is zero then the maximum number of function evaluations was reached. If `ExitFlag` is negative then the optimization routine did not converge to a solution.
`Output`	The `Output` structure contains several pieces of information about the optimization process. It provides the number of function evaluations (`Output.iterations`), the name of the algorithm used to solve the problem (`Output.algorithm`), and Lagrange multipliers for constraints, etc.

Unconstrained Optimization: Single-Variable

$$\min_x f(x) = 2 - 4x + e^x, \quad -10 \le x \le 10$$

```
% All comments start with %
% File name: Example7_1.m
% Problem: minimize f(x) = 2 – 4x + exp(x)
  clear all
% Set lower and upper bound for the design variable
  Lb = -10; Ub = 10;
% Invoke single variable unconstrained optimizer fminbnd;
% The argument ObjFunction7_1 refers to the m-file that
% contains expression for the objective function
  [x,FunVal,ExitFlag,Output] = fminbnd('ObjFunction7_1',Lb,Ub)
```

```
% File name: ObjFunction7_1.m
% Example 7.1 Single variable unconstrained minimization
  function f = ObjFunction7_1(x)
  f = 2 - 4*x + exp(x);
```

x = 1.3863, FunVal = 0.4548, ExitFlag = 1 > 0 (ie, minimum was found),
output = (iterations: 14, funcCount: 14, algorithm: golden section search, parabolic interpolation).

Unconstrained Optimization: Multivariable

$$\underset{\mathbf{x}}{\text{Min}}\ f(\mathbf{x}) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2,\ \mathbf{x}^{(0)} = (-1.2, 1.0) \rightarrow \mathbf{x}^* = (1.0, 1.0),\ f(\mathbf{x}^*) = 0$$

```
[x,FunValue,ExitFlag,Output]=fminsearch('ObjFun',x0,options)

[x,FunValue,ExitFlag,Output]=fminunc('ObjFun',x0,options)
```

`ObjFun` = the name of the m-file that returns the function value and its gradient if programmed
`x0` = the starting values of the design variables
`options` = a data structure of parameters that can be used to invoke various conditions for the optimization process

`fminsearch` uses the Simplex search method of Nelder–Mead, which does not require numerical or analytical gradients of the objective function (see Subsection 11.9.3 for details of this algorithm). Thus it is a *nongradient-based method* (*direct search method*) that can be used for problems where the cost function is not differentiable.

Since `fminunc` does require the gradient value, with the option `LargeScale` set to `off`, it uses the BFGS quasi-Newton method (refer to chapter: More on Numerical Methods for Unconstrained Optimum Design for details) with a mixed quadratic and cubic line search procedure. The DFP formula (refer to chapter: More on Numerical Methods for Unconstrained Optimum Design, for details),

```
% File name: Example7_2
% Rosenbruck valley function with analytical gradient of
% the objective function
  clear all
  x0 = [-1.2 1.0]';  Set starting values
% Invoke unconstrained optimization routines
% 1. Nelder-Mead simplex method, fminsearch
   % Set options: medium scale problem, maximum number of function evaluations
   % Note that "..." indicates that the text is continued on the next line
    options = optimset('LargeScale', 'off', 'MaxFunEvals', 300);
            [x1, FunValue1, ExitFlag1, Output1] =  ...
             fminsearch ('ObjAndGrad7_2', x0, options)
% 2. BFGS method, fminunc, dafault option
   % Set options: medium scale problem, maximum number of function evaluations,
   % gradient of objective function
    options = optimset('LargeScale', 'off', 'MaxFunEvals', 300,...
            'GradObj', 'on');
    [x2, FunValue2, ExitFlag2, Output2] = ...
            fminunc ('ObjAndGrad7_2', x0, options)
% 3. DFP method, fminunc, HessUpdate = dfp
   % Set options: medium scale optimization, maximum number of function evaluation,
   % gradient of objective function, DFP method
    options = optimset('LargeScale', 'off', 'MaxFunEvals', 300, ...
            'GradObj', 'on', 'HessUpdate', 'dfp');
    [x3, FunValue3, ExitFlag3, Output3] = ...
            fminunc ('ObjAndGrad7_2', x0, options)
```

```
% File name: ObjAndGrad7_2.m
% Rosenbrock valley function
  function [f, df] = ObjAndGrad7_2(x)
% Re-name design variable x
  x1 = x(1); x2 = x(2);  %
% Evaluate objective function
  f = 100*(x2 - x1^2)^2 + (1 - x1)^2;
% Evaluate gradient of the objective function
  df(1) = -400*(x2-x1^2)*x1 - 2*(1-x1);
  df(2) = 200*(x2-x1^2);
```

Constrained Optimization

$$\left.\begin{aligned} &\underset{\mathbf{x}}{\text{Minimize}}\ f(\mathbf{x}) = (x_1 - 10)^2 + (x_2 - 20)^2 \\ &\text{subject to}\ g_1(\mathbf{x}) = 100 - (x_1 - 5)^2 + (x_2 - 5)^2 \le 0 \\ &\qquad g_2(\mathbf{x}) = -82.81 - (x_1 - 6)^2 + (x_2 - 5)^2 \le 0 \\ &\qquad 13 \le x_1 \le 100,\ 0 \le x_2 \le 100 \end{aligned}\right\} \rightarrow \mathbf{x}^* = (14.095, 0.84296),\ f(\mathbf{x}^*) = -6961.8$$

Active constraints: 5, 6 [ie, g(1) and g(2)]
x = (14.095,0.843), FunVal = −6.9618e + 003, ExitFlag = 1 > 0 (ie, minimum was found)
output = (iterations: 6, funcCount: 13, stepsize: 1, algorithm: medium scale: SQP, quasi-Newton, line-search).

```
% File name: Example7_3
% Constrained minimization with gradient expressions available
% Calls ObjAndGrad7_3 and ConstAndGrad7_3

  clear all

% Set options; medium scale, maximum number of function evaluation,
% gradient of objective function, gradient of constraints, tolerances
% Note that three periods "..." indicate continuation on next line

  options = optimset ('LargeScale', 'off', 'GradObj', 'on',...
  'GradConstr', 'on', 'TolCon', 1e-8, 'TolX', 1e-8);

% Set bounds for variables

  Lb = [13; 0]; Ub = [100; 100];

% Set initial design

  x0 = [20.1; 5.84];

% Invoke fmincon; four [ ] indicate no linear constraints in the problem

  [x,FunVal, ExitFlag, Output] = …
  fmincon('ObjAndGrad7_3',x0,[ ],[ ],[ ],[ ],Lb, …
  Ub,'ConstAndGrad7_3',options)
```

```
% File name: ObjAndGrad7_3.m

function [f, gf] = ObjAndGrad7_3(x)

% f returns value of objective function; gf returns objective fun
% Re-name design variables x

x1 = x(1); x2 = x(2);

% Evaluate objective function

f = (x1-10)^3 + (x2-20)^3;

% Compute gradient of objective function

if nargout > 1
        gf(1,1) = 3*(x1-10)^2;
        gf(2,1) = 3*(x2-20)^2;
end
```

```
% File name: ConstAndGrad7_3.m

function [g, h, gg, gh] = ConstAndGrad7_3(x)

% g returns inequality constraints; h returns equality constraints
% gg returns gradients of inequalities; each column contains a gradient
% gh returns gradients of equalities; each column contains a gradient
% Re-name design variables

x1 = x(1); x2 = x(2);

% Inequality constraints

g(1) = 100-(x1-5)^2-(x2-5)^2 ;
g(2) = -82.81+ (x1-6)^2 + (x2-5)^2;

% Equality constraints (none)

h = [ ];

% Gradients of constraints

if nargout > 2
        gg(1,1) = -2*(x1-5);
        gg(2,1) = -2*(x2-5);
        gg(1,2) = 2*(x1-6);
        gg(2,2) = 2*(x2-5);
        gh = [];
end
```