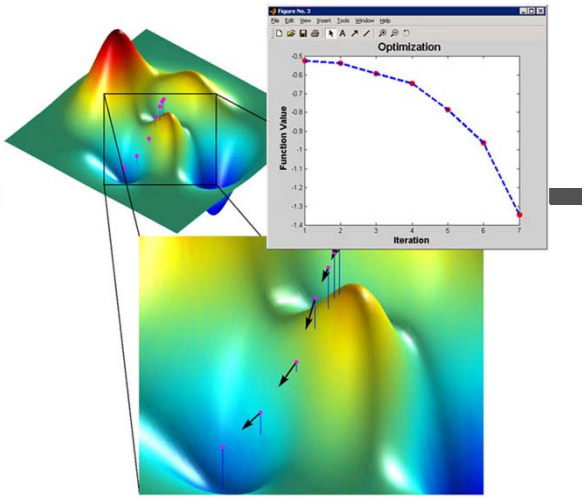
The MathWorks
Accelerating the pace of engineering and science



Optimization Toolbox

V5.0 (R2010a)

Seungjae Min

Department of Automotive Engineering
Hanyang University

Contents

- Setting up an optimization
  - Choosing a solver
  - Writing objective functions
  - Writing constraints
  - Setting options
- Examining results
  - Exit flags and exit messages
  - First-order optimality measure
  - Displaying iterative output
- Steps to take after running a solver
  - When the solver fails
  - When the solver might have succeeded
  - When the solver succeeds

Categories of Solvers

- Minimizers
  - Find a local minimum of the objective function near a starting point x_0
  - Unconstrained optimization, linear programming, quadratic programming, general nonlinear programming
- Multiobjective minimizers
  - Minimize the maximum value of a set of functions (`fminimax`)
  - Find a location where a collection of functions is below some prespecified values (`fgoalattain`)
- Equation solvers
  - Find a solution to a scalar- or vector-valued nonlinear equation f(x) = 0 near a starting point x_0
- Least-Squares (curve-fitting) solvers
  - Minimize a sum of squares

Optimization Decision Table

Constraint Type	Objective Type				
	Linear	Quadratic	Least Squares	Smooth nonlinear	Nonsmooth
None	n/a (f = const, or min = $-\infty$)	`quadprog`	`\`, `lsqcurvefit`, `lsqnonlin`	`fminsearch`, `fminunc`	`fminsearch`, *
Bound	`linprog`	`quadprog`	`lsqcurvefit`, `lsqlin`, `lsqnonlin`, `lsqnonneg`	`fminbnd`, `fmincon`, `fseminf`	*
Linear	`linprog`	`quadprog`	`lsqlin`	`fmincon`, `fseminf`	*
General smooth	`fmincon`	`fmincon`	`fmincon`	`fmincon`, `fseminf`	*
Discrete	`bintprog`				

Choosing a Solver: Minimization Problems (1)

Type	Formulation	Solver
Scalar Minimization	$\min_a f(a)$ such that $a_1 < a < a_2$	`fminbnd`
Unconstrained Minimization	$\min_x f(\boldsymbol{x})$	`fminunc`, `fminsearch`
Linear Programming	$\min_x \boldsymbol{f}^T \boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b},\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq},\ \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}$	`linprog`
Quadratic Programming	$\min_x \frac{1}{2}\boldsymbol{x}^T \boldsymbol{H}\boldsymbol{x} + \boldsymbol{f}^T \boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b},\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq},\ \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}$	`quadprog`
Constrained Minimization	$\min_x f(\boldsymbol{x})$ such that $\boldsymbol{c}(\boldsymbol{x}) \le \boldsymbol{0},\ \boldsymbol{c}_{eq}(\boldsymbol{x}) = \boldsymbol{0}$ $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b},\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq},\ \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}$	`fmincon`

Choosing a Solver: Minimization Problems (2)

Type	Formulation	Solver
Semi-infinite minimization	$\min_{\boldsymbol{x}} f(\boldsymbol{x})$ such that $K(\boldsymbol{x}, w) \le 0$ for all w, $\boldsymbol{c}(\boldsymbol{x}) \le \boldsymbol{0}$, $\boldsymbol{c}_{eq}(\boldsymbol{x}) = \boldsymbol{0}$, $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}$, $\boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq}$, $\boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}$	`fseminf`
Binary integer programming	$\min_{\boldsymbol{x}} \boldsymbol{f}^T\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}$, $\boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq}$, $\boldsymbol{x}$ binary	`bintprog`

Choosing a Solver: Multiobjective Problem

Type	Formulation	Solver
Goal attainment	$\min_{\boldsymbol{x},\gamma} \gamma$ such that $F(\boldsymbol{x}) - w \cdot \gamma \le \text{goal},$ $\boldsymbol{c}(\boldsymbol{x}) \le \boldsymbol{0},\ \boldsymbol{c}_{eq}(\boldsymbol{x}) = \boldsymbol{0},$ $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b},\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq},\ \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}$	`fgoalattain`
Minmax	$\min_{\boldsymbol{x}} \max_{\{F_i\}} \{F_i(\boldsymbol{x})\}$ such that $\boldsymbol{c}(\boldsymbol{x}) \le \boldsymbol{0},\ \boldsymbol{c}_{eq}(\boldsymbol{x}) = \boldsymbol{0}$ $\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b},\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq},\ \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}$	`fminimax`

Objective Functions

- Scalar-valued objective functions
  - $f(x), \nabla f(x), H(x) = \partial^2 f / \partial x_i \partial x_j$
  - options = optimset('GradObj','on','Hessian','on')
- Vector-valued or matrix-valued objective functions
  - Jacobian $J_{ij}(x) = \dfrac{\partial F_i(x)}{\partial x_j}$
- Anonymous function objectives
  - Without gradient or Hessian information
- Maximizing an objective
  - Define $g(x) = -f(x)$ and minimize g
- Passing extra parameters
  - Anonymous functions, nested functions, global variables

Objective Functions: Rosenbrock's function

```
function [f g H] = rosenboth(x)
% Calculate objective f
f = 100*(x(2) - x(1)^2)^2 + (1-x(1))^2;

if nargout > 1 % gradient required
    g = [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1));
        200*(x(2)-x(1)^2)];

    if nargout > 2 % Hessian required
        H = [1200*x(1)^2-400*x(2)+2, -400*x(1);
            -400*x(1), 200];
    end

end
options = optimset('GradObj','on','Hessian','on');
[x fval] = fminunc(@rosenboth,[-1;2],options)


anonrosen = @(x)(100*(x(2) - x(1)^2)^2 + (1-x(1))^2);
options = optimset('LargeScale','off');
[x fval] = fminunc(anonrosen,[-1;2],options)
```

Constraints

- Bounds
  - Violation: `fmincon active-set algorithm, fgoalattain, fminimax, fseminf`
- Linear inequality: $Ax \leq b$
- Linear equality: $A_{eq}x = b_{eq}$
- Nonlinear: $c(x) \leq 0$, $c_{eq}(x) = 0$

Constraints: Example

```
function fullexample
x0 = [1; 4; 5; 2; 5];
lb = [-Inf; -Inf;  0; -Inf;   1];
ub = [ Inf;  Inf; 20];
Aeq = [1 -0.3 0 0 0];
beq = 0;
A = [0 0  0 -1  0.1
     0 0  0  1 -0.5
     0 0 -1  0  0.9];
b = [0; 0; 0];

[x,fval,exitflag]=fmincon(@myobj,x0,A,b,Aeq,beq,lb,ub,...
                                        @myconstr)

%------------------------------------------------------------
function f = myobj(x)

f = 6*x(2)*x(5) + 7*x(1)*x(3) + 3*x(2)^2;

%------------------------------------------------------------
function [c, ceq] = myconstr(x)

c = [x(1) - 0.2*x(2)*x(5) - 71
     0.9*x(3) - x(4)^2 - 67];
ceq = 3*x(2)^2*x(5) + 3*x(1)^2*x(3) - 20.875;
```

$$\frac{x_1^2}{9}+\frac{x_2^2}{4}\le 1,$$
$$x_2 \ge x_1^2 - 1$$

```
function [c,ceq,gradc,gradceq]=ellipseparabola(x)
% Inside the ellipse bounded by (-3<x<3),(-2<y<2)
% Above the line y=x^2-1
c(1) = x(1)^2/9 + x(2)^2/4 - 1;
c(2) = x(1)^2 - x(2) - 1;
ceq = [];
if nargout > 2
    gradc = [2*x(1)/9, 2*x(1);...
             x(2)/2, -1];
    gradceq = [];
end
options=optimset('GradConstr','on');
[x,fval] = fmincon(@myobj,x0,A,b,Aeq,beq,lb,ub,...
           @ellipseparabola,options)
```

Setting Options

- Default options
- Changing the default settings
- Tolerances and stopping criteria
- Checking validity of gradients or Jacobians
- Choosing the algorithm

Changing Default Settings: Example

- Available options for derivatives change as the type of supplied derivatives change
  - Approximated by solver
    - User-supplied derivatives: Hessian sparsity pattern
    - Approximated derivatives: (Finite differences:) minimum / maximum perturbation
  - Gradient supplied
    - User-supplied derivatives: validate user-supplied derivatives, Hessian sparsity pattern
  - Gradient and Hessian supplied
    - User-supplied derivatives: validate user-supplied derivatives, Hessian sparsity pattern, Hessian multiply function

Changing Default Settings: Choosing an Algorithm

  - Some solvers explicitly use an option called `LargeScale`
    - `fmincon`, `linprog`
  - Other solvers do not use the `LargeScale` attribute explicitly, but have an option called `Algorithm`
    - `fmincon`, `fsolve`

- Large-Scale vs. Medium-Scale Algorithms
  - Large-scale: sparse linear algebra
  - Medium-scale: dense linear algebra, full matrices

Large-Scale Methods

Function	Problem Formulations	Additional Information Needed	For Large Problems
`fminunc`	$\min_x f(x)$	Must provide gradient for `f(x)` in `fun`.	• Provide sparsity structure of the Hessian, or compute the Hessian in `fun`. • The Hessian should be sparse.
`fmincon`	• $\min_x f(x)$ such that $l \le x \le u$, where $l < u$. • $\min_x f(x)$ such that $Aeq \cdot x = beq$, and Aeq is an m-by-n matrix where $m \le n$.	Must provide gradient for `f(x)` in `fun`.	• Provide sparsity structure of the Hessian or compute the Hessian in `fun`. • The Hessian should be sparse. • Aeq should be sparse.

Medium-Scale Algorithm

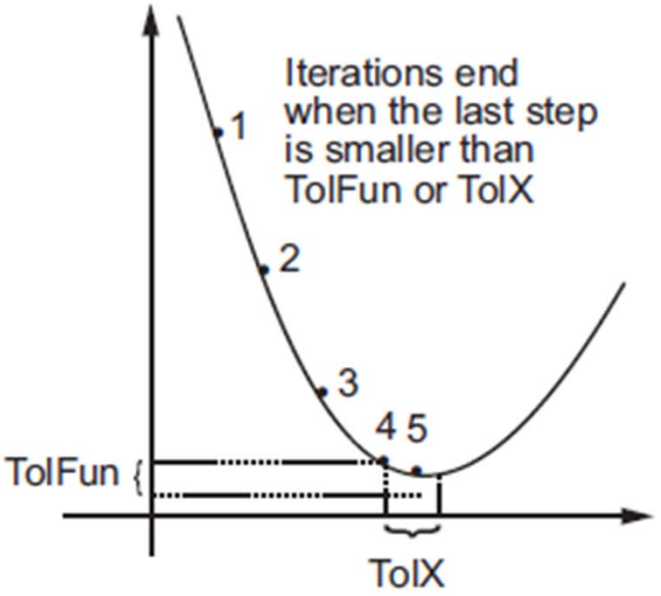
- Unconstrained minimization
  - Nelder-Mead simplex search method
  - BFGS (Broyden, Fletcher,Goldfarb, and Shanno) quasi-Newton method
- Constrained minimization
  - Minimax
  - Goal attainment
  - Semi-infinite optimization
  - Variations of *sequential quadratic programming* (SQP)
- Nonlinear least-squares problems
  - Gauss-Newton method
  - Levenberg-Marquardt method
- Nonlinear equation solving
  - Trust-region dogleg algorithm
- Line search strategy
  - For unconstrained minimization and nonlinear least-squares problems
  - Safeguarded cubic method
  - Quadratic interpolation method
  - Extrapolation method

Large-Scale Algorithm

- Trust-region method (except LP)
- Bound constrained problems
  - reflective Newton method
- Equality constrained problems
  - Projective preconditioned conjugate gradient iteration
- Linear Programming
  - Primal-dual interior-point method

Tolerances and Stopping Criteria

- TolX $|(x_i - x_{i+1})| < \texttt{TolX}*(1 + |x_i|)$
  - Lower bound on the size of a step
- TolFun $|f(x_i) - f(x_{i+1})| < \texttt{TolFun}(1 + |f(x_i)|)$
  - Lower bound on the change in the value of the objective function during a step
  - Bound on the first-order optimality measure
- TolCon
  - Upper bound on the magnitude of any constraint functions
- MaxIter
  - Bound on the number of solver iterations
- MaxFunEvals
  - Bound on the number of function evaluations



Choosing the Algorithm (1)

- `fmincon` Algorithms
  - interior-point (large-scale algorithm)
    - handle large, sparse problems, as well as small dense problems, satisfy bounds at all iterations
  - sqp (not large-scale algorithm)
    - satisfy bounds at all iterations
  - active-set (not large-scale algorithm)
    - can take large steps, which adds speed
    - effective on some problems with nonsmooth constraints
  - trust-region-reflective (default) (large-scale algorithm)
    - require you to provide a gradient, and allows only bounds or linear equality constraints, but not both

Trust Region Methods

- Difficulties in Newton's method
  - Not p.d. Hessian or highly nonlinear function→outside the validity region of the quadratic approximation
- Size of the trust region (restricted step): dynamically adjusted

$$\Omega_k = \left\{ x : \left\| x - x_k \right\| \le h_k \right\}$$

- Quadratic Programming problem

$$\min \quad q(x) = f(x_k) + \nabla f(x_k)^T \delta + \frac{1}{2} \delta^T \nabla^2 f(x_k) \delta$$

$$\text{subject to} \quad -h_k \le \delta_i \le h_k \qquad i = 1, \ldots, n$$

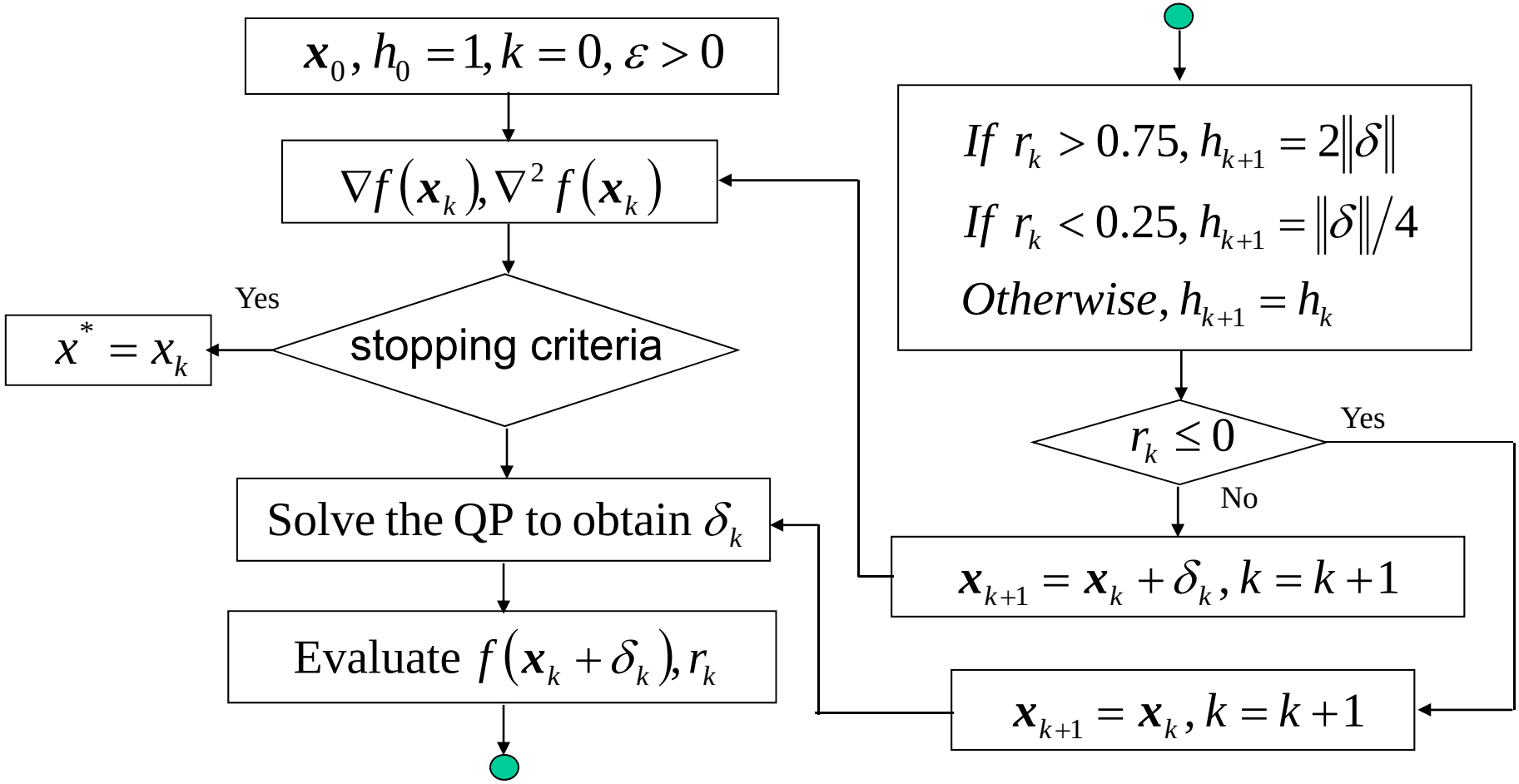
- Reliability index

$$\left.\begin{array}{l} \text{actual change}: \ \Delta f = f(x_k) - f(x_k + \delta_k) \\ \text{predicted change}: \ \Delta q = f(x_k) - q(\delta_k) \end{array}\right\} \rightarrow \boxed{r_k = \frac{\Delta f}{\Delta q}}$$

$r_k \to 1$: better agreement

- Not very sensitive to scaling, very robust

Trust Region Algorithm



Choosing the Algorithm (2)

- fmincon Algorithms
  - Recommendations
    - Use the 'interior-point' algorithm first.
    - To run an optimization again to obtain more speed on small- to medium-sized problems, try 'sqp' next, and 'active-set' last.
    - Use 'trust-region-reflective' when applicable. Your problem must have: objective function includes gradient, only bounds, or only linear equality constraints (but not both).
  - sequential quadratic programming (SQP) method
    - Update an estimate of the Hessian of the Lagrangian at each iteration using the BFGS formula
    - Quadratic programming solution
    - Perform a line search using a merit function

Choosing the Algorithm (2)

- `linprog` Algorithms
  - Large-scale interior-point
  - Medium-scale active set
  - Medium-scale Simplex
  - Recommendations
    - Use 'LargeScale' = 'on'
- `quadprog` Algorithms
  - Large-scale
  - Medium-scale
  - Recommendations
    - If you have only bounds, or only linear equalities, use 'LargeScale' = 'on‘
    - If you have linear inequalities, or both bounds and linear equalities, use 'LargeScale' = 'off'

Input Argument (1)

Argument	Description	Used by Functions
A, b	The coefficients of the linear inequality constraints and the corresponding right-hand side vector: $\boldsymbol{Ax} \le \boldsymbol{b}$	fmincon, fminimax, linprog, quadprog
Aeq, beq	The coefficients of the linear equality constraints and the corresponding right-hand side vector: $\boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq}$	fmincon, fminimax, linprog, quadprog
c, ceq	Nonlinear inequality constraints and nonlinear equality constraints: $\boldsymbol{c} \le \boldsymbol{0}$, $\boldsymbol{c}_{eq} = \boldsymbol{0}$	fmincon
f	The vector of coefficients for the linear term in the linear equation f'*x or the quadratic equation x'*H*x+f'*x.	linprog, quadprog
fun	The function to be optimized.	fminbnd, fmincon, fminimax, fminsearch, fminunc
H	The matrix of coefficients for the quadratic terms.	quadprog

Input Argument (2)

Argument	Description	Used by Functions
nonlcon	The function that computes the nonlinear inequality and equality constraints.	fmincon, fminimax
lb,ub	Lower and upper bound vectors(or matrices).	fmincon, fminimax, linprog, quadprog
options	An optimization options parameter structure. Use optimset to create or edit optimization options parameter structure.	all functions
P1,P2,...	Additional arguments to be passed to fun, nonlcon (if it exists)when the optimization function calls the functions fun, nonlcon using these calls: f = feval(fun,x,P1,P2,...) [c, ceq] = feval(nonlcon,x,P1,P2,...)	fminbnd,fmincon, fminimax, fminsearch, fminunc
x0	Starting point (a scalar, vector or matrix).	All functions except fminbnd
x1, x2	The interval over which the function is minimized.	fminbnd

Output Argument (1)

Argument	Description	Used by Functions
exitflag	The exit condition. > 0: converge to a solution x = 0: maximum number of function evaluations is reached < 0: did not converge to a solution	`All functions`
fval	The value of the objective function fun at the solution x.	`All functions`
grad	The value of the gradient of fun at the solution x.	`fmincon, fminunc`
hessian	The value of the Hessian of fun at the solution x.	`fmincon, fminunc`
lambda	The Lagrange multipliers at the solution x. lambda is a structure where each field is for a different constraint type lambda.lower/upper: value of the multipliers at the lower/upper bound lambda.eqlin/ineqlin: value of the multipliers tied to the linear equality/ inequality constraints lambda.eqnonlin/ineqnonlin: value of the multipliers tied to the nonlinear equality/inequality constraints	`fmincon, fminimax, linprog, quadprog`

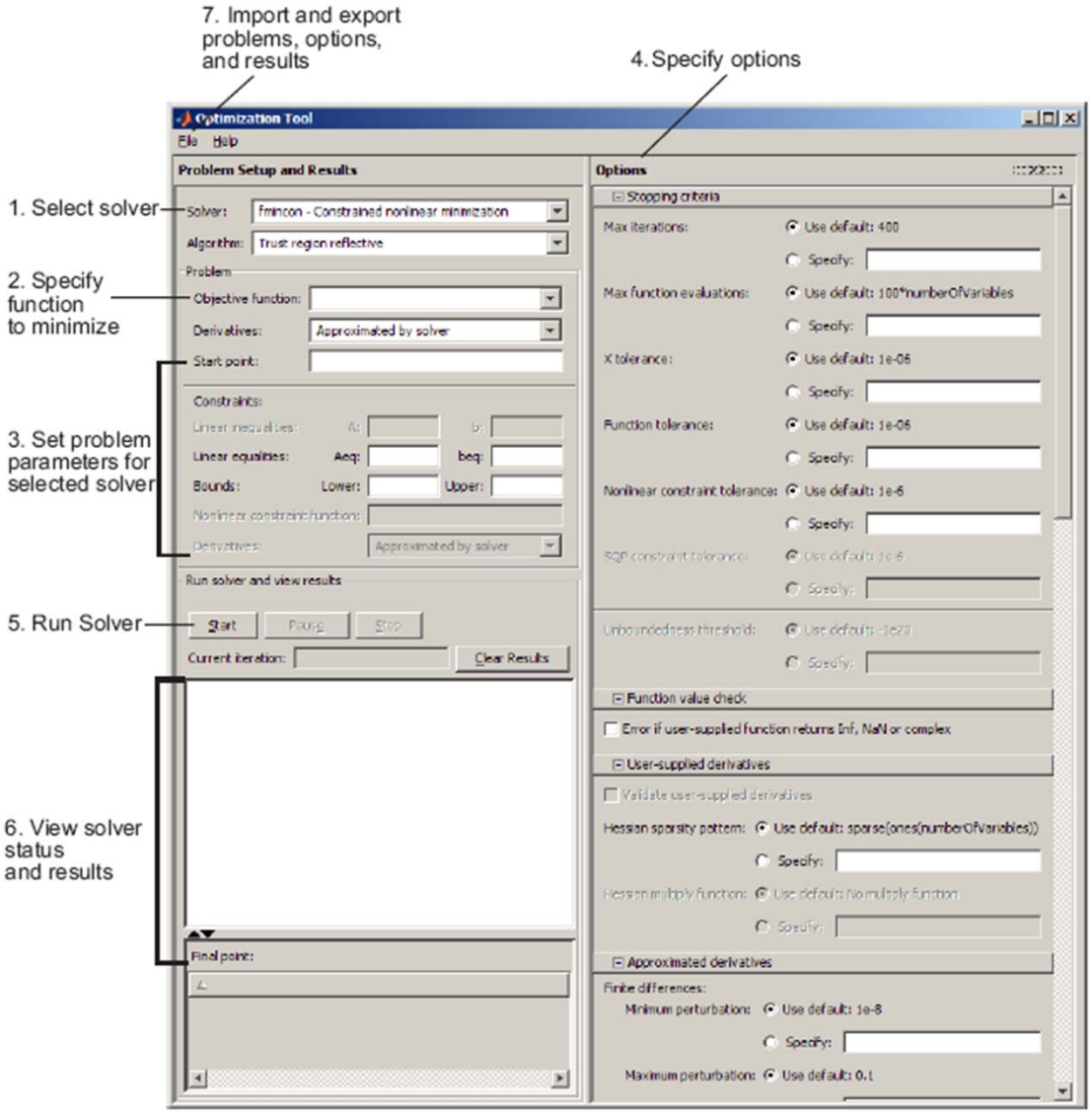
Output Argument (2)

Argument	Description	Used by Functions
maxfval	max{fun(x)} at the solution x.	`fminimax`
output	An output structure that contains information about the results of the optimization. output.iterations: number of function evaluations output.algorithm: name of the algorithm used to solve the problem output.funcCount: function count	`All functions`
x	The solution found by the optimization function.	`All functions`

- Stopping criteria
  - First-order optimality measure, `TolX`, `TolFun`, `TolCon`, `MaxIter`, `MaxFunEvals`
- First-order optimality measure
  - Unconstrained optimality: infinite-norm of gradient
  - Constrained optimality: KKT conditions

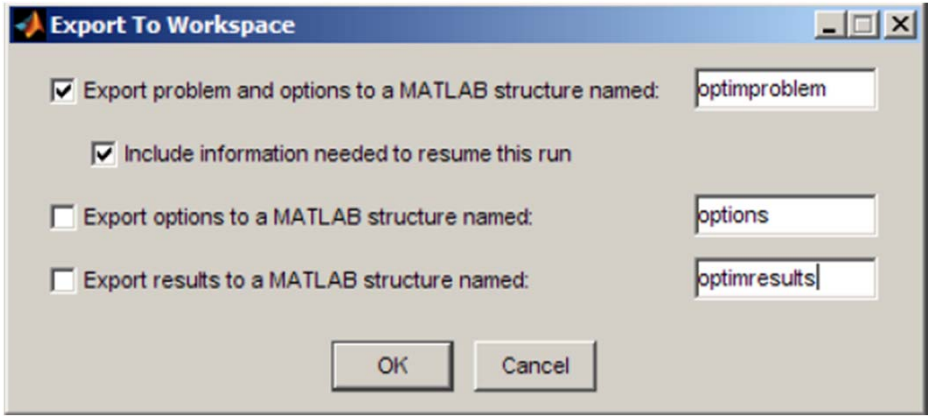
Optimization Tool

- optimtool



Import / Export

- Exporting to the MATLAB workspace
  - File > Export to Workspace
  - fmincon(optimproblem)



- Importing your work
  - Optimtool(options)
  - File > Import Options
  - File > Import Problem

- Generating an M-file
  - File > Generate M-file

OPTIMSET parameters

`Diagnostics`	Print diagnostic information about the function to be minimized [on | {off}]
`DiffMaxChange`	Maximum change in variables for finite difference gradients [positive scalar | {1e-1}]
`DiffMinChange`	Minimum change in variables for finite difference gradients [positive scalar | {1e-8}]
`Display`	Level of display [off | iter | {final}]
`GradConstr`	Gradients for the nonlinear constraints defined by user [on | {off}]
`GradObj`	Gradient(s) for the objective function(s) defined by user [on | {off}]
`Hessian`	Hessian for the objective function defined by user [on | {off}]
`HessUpdate`	Quasi-Newton updating scheme [{bfgs} | dfp | gillmurray | steepdesc]
`LineSearchType`	Line search algorithm choice [cubic poly | {quadcubic}]
`MaxFunEvals`	Maximum number of function evaluations allowed [positive integer]
`MaxIter`	Maximum number of iterations allowed [positive integer]
`TolCon`	Termination tolerance on the constraint violation [positive scalar]
`TolFun`	Termination tolerance on the function value [positive scalar]
`TolX`	Termination tolerance on X [positive scalar]

>> options = optimset('Diagnostics','on','Display','iter','MaxIter','500','HessUpdate,'dfp','TolCon',1.0e-08)

>> options = optimset('fmincon') : default values

Parameter Setting

- fminbnd, fminunc, fminsearch, fmincon, fminimax
- Syntax
  - options = optimset('param1',value1,'param2',value2,…)
  - options = optimset(options, 'Display','iter')
  - options = optimset(options,'GradObj','on','GradConstr','on')

Parameter	Description	Value
MaxIter	Maximum number of iterations allowed	Positive integer
MaxFunEvals	Maximum number of function evaluations allowed	Positive integer
Display	Level of display	off | iter | notify | final
GradObj	Gradient(s) for the objective function(s) defined by user	on | {off}
GradConstr	Gradients for the nonlinear constraints defined by user	on | {off}
LargeScale	Use large-scale algorithm if possible In fmincon LargeScale off ➔ SQP	{on} | off

Output Functions

- output from an optimization algorithm at each iteration
  - options = optimset('OutputFcn', @outfun)

```
function stop = outfun(x,optimValues,state)
stop = false;

   switch state
       case 'init'
           hold on
       case 'iter'
           % Concatenate current point and objective function
           % value with history. x must be a row vector.
           history.fval = [history.fval; optimValues.fval];
           history.x = [history.x; x];
           % Concatenate current search direction with
           % searchdir.
           searchdir = [searchdir;...
                        optimValues.searchdirection'];
           plot(x(1),x(2),'o');
           % Label points with iteration number.
           text(x(1)+.15,x(2),num2str(optimValues.iteration));
       case 'done'
           hold off
       otherwise
   end
end
```

Output Functions: Example

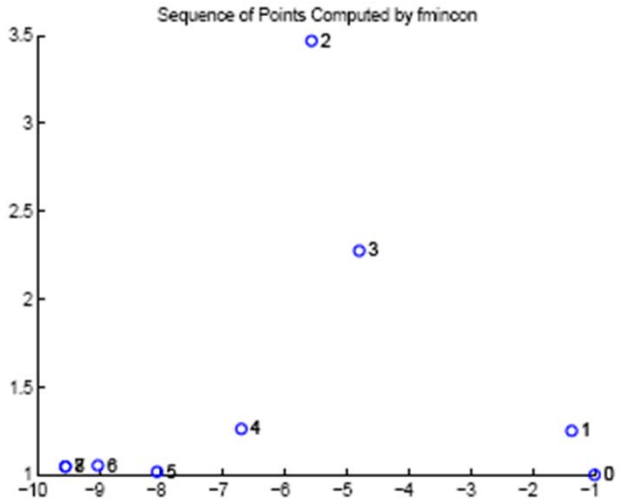
- Plots the current point computed by the algorithm.
- Stores the point and its corresponding objective function value in a variable called `history`, and stores the current search direction in a variable called `searchdir`.

```
function [history,searchdir] = runfmincon

% Set up shared variables with OUTFUN
history.x = [];
history.fval = [];
searchdir = [];

% call optimization
x0 = [-1 1];
options = optimset('outputfcn',@outfun,'display','iter',...
'Algorithm','active-set');
xsol = fmincon(@objfun,x0,[],[],[],[],[],[],@confun,options);
 function f = objfun(x)
     f = exp(x(1))*(4*x(1)^2 + 2*x(2)^2 + 4*x(1)*x(2) +...
                    2*x(2) + 1);
 end

 function [c, ceq] = confun(x)
     % Nonlinear inequality constraints
       c = [1.5 + x(1)*x(2) - x(1) - x(2);
           -x(1)*x(2) - 10];
       % Nonlinear equality constraints
       ceq = [];
   end
 end
```



When the Solver Fails (1)

- Too Many Iterations or Function Evaluations
  - Enable Iterative Display
  - Relax Tolerances
  - Start the Solver From Different Points
  - Check Objective and Constraint Function Definitions
  - Center and Scale Your Problem
  - Provide Gradient or Hessian
- No Feasible Point
  - Check Linear Constraints
  - Check Nonlinear Constraints

When the Solver Fails (2)

- Problem Unbounded
  - Your problem might be truly unbounded.
  - Check that your problem is formulated correctly.
  - Try scaling or centering your problem.
  - Relax the objective limit tolerance by using optimset to reduce the value of the ObjectiveLimit tolerance.
- Solver Takes Too Long
  - Enable Iterative Display
  - Enable FunValCheck
  - Use Appropriate Tolerances
  - Use a Plot Function
  - Enable DerivativeCheck
  - Use an Output Function
  - Use a Sparse Solver or a Multiply Function
  - Use Parallel Computing

Local Minimum Possible?

  - The solver might have reached a local minimum, but cannot be certain because the first-order optimality measure is not less than the TolFun tolerance.

- To see if the reported solution is reliable,
  - Rerun Starting At Final Point
  - Try a Different Algorithm
  - Change Tolerances
  - Rescale the Problem
  - Check Nearby Points
  - Change Finite Differencing Options
  - Provide Analytic Gradients or Jacobian
  - Provide a Hessian

Local Minimum Found

- Lesson: check your results, even if the solver reports that it “found” a local minimum. To see if the reported solution is reliable,
  - Change the Initial Point
  - Check Nearby Points
  - Check your Objective and Constraint Functions

Examples

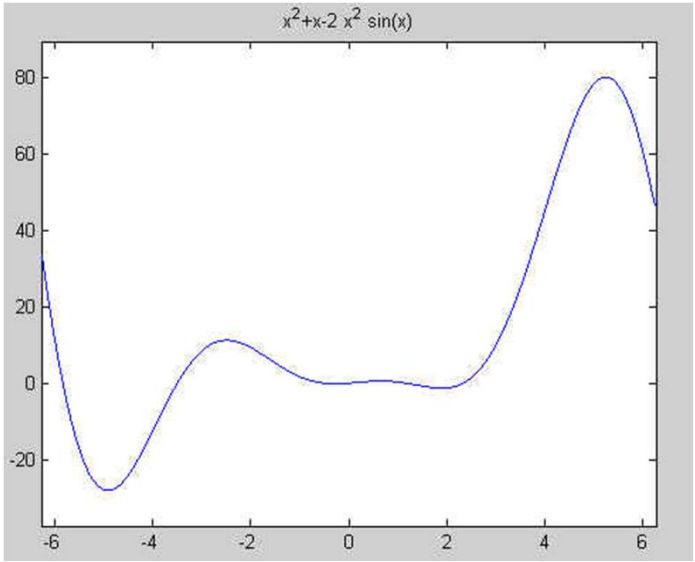
- `Fminbnd`
- `Fminunc`
- `Fminsearch`
- `Fmincon`
- `Linprog`
- `Quadprog`
- `Fminimax`

fminbnd

- finds the minimum of a function of one variable within a fixed interval(local optimum)
- Syntax
  - x = fminbnd(fun,x1,x2)
  - x = fminbnd(fun,x1,x2,options)
  - x = fminbnd(fun,x1,x2,options,P1,P2,...)
  - [x,fval] = fminbnd(...)
  - [x,fval,exitflag] = fminbnd(...)
  - [x,fval,exitflag,output] = fminbnd(...)

- example

$$f(x) = x^2 + x - 2x^2 \sin(x), -6 \le x \le 6$$




```
>> f='x^2+x-2*x^2*sin(x)'
>> x=fminbnd(f,-6,6)
```

Optimum Point : x =1.9057

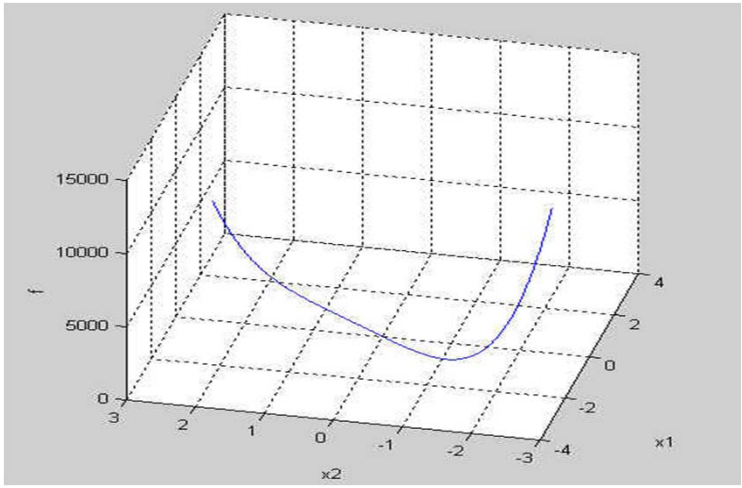
fminunc

- Find the minimum of an unconstrained multivariable function.(BFGS Quasi-Newton method)

- Syntax
  - x = fminunc(fun,x0)
  - x = fminunc(fun,x0,options)
  - x = fminunc(fun,x0,options,P1,P2,...)
  - [x,fval] = fminunc(...)
  - [x,fval,exitflag] = fminunc(...)
  - [x,fval,exitflag,output] = fminunc(...)
  - [x,fval,exitflag,output,grad] = fminunc(...)
  - [x,fval,exitflag,output,grad,hessian] = fminunc(...)

- example

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



>> x0=[-2,1] % starting point

>> x= fminunc('fun',x0)

Optimum Point : x =[1.0000 1.0000]

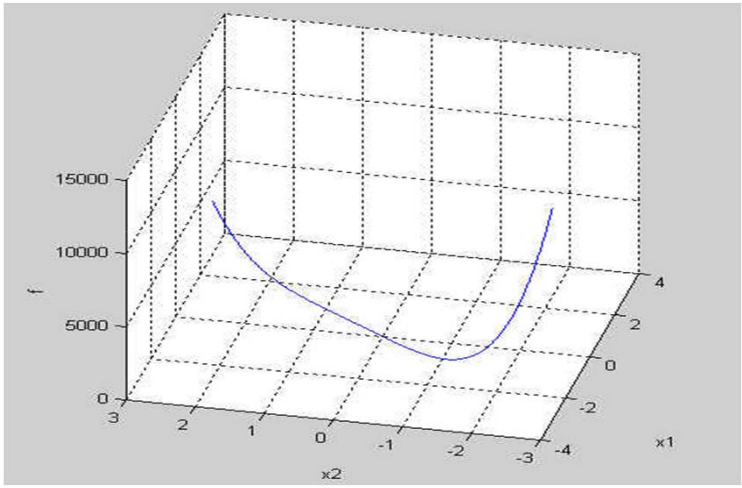
```
%fun must be a function
function f=fun(x)
f=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

fminsearch

- Find the minimum of an unconstrained multivariable function.(Nelder-Mead simplex method)
- Syntax
  - x = fminsearch(fun,x0)
  - x = fminsearch(fun,x0,options)
  - x = fminsearch(fun,x0,options,P1,P2,...)
  - [x,fval] = fminsearch(...)
  - [x,fval,exitflag] = fminsearch(...)
  - [x,fval,exitflag,output] = fminsearch(...)

- example

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




```
>> x0=[-2,1] % starting point
>> x= fminsearch('fun',x0)
```

Optimum Point : x =[1.0000 1.0000]

```
%fun must be a function : m-file
function f=fun(x)
f=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

fmincon (1)

- Find the minimum of a constrained nonlinear multivariable function.
- Syntax
  - x = fmincon(fun,x0,A,b)
  - 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(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options, P1,P2, ...)
  - [x,fval] = fmincon(...)
  - [x,fval,exitflag] = fmincon(...)
  - [x,fval,exitflag,output] = fmincon(...)
  - [x,fval,exitflag,output,lambda] = fmincon(...)
  - [x,fval,exitflag,output,lambda,grad] = fmincon(...)
  - [x,fval,exitflag,output,lambda,grad,hessian] = fmincon(...)

- example

(1) Linear inequality constraint

$$\min\ f(x) = -x_1 x_2 x_3$$
$$\text{s.t.}\ \ g_1(x) = x_1 + 2x_2 + 2x_3 - 72 \le 0$$
$$g_2(x) = -x_1 - 2x_2 - 2x_3 \le 0$$

```
>> x0=[10,10,10] % starting point
>>A=[1 2 2;-1 -2 -2]
>> b=[72;0]
>> x= fmincon('fun',x0,A,b)
```

Optimum Point :

x =[24.0000 12.0000 12.0000]

```
%fun must be a function : m-file
function f=fun(x)
f=-x(1)*x(2)*x(3);
```

fmincon (2)

(2) Linear equality constraint

$$\begin{aligned}\min\ & f(x) = (x_1-3)^2 + (x_2-3)^2 \\ \text{s.t.}\ & g_1(x) = x_1 - 3x_2 - 1 = 0 \\ & g_2(x) = x_1 + x_2^2 - 4 \le 0\end{aligned}$$

```
>> x0=[1,1] % starting point
>> Aeq=[1 -3]
>> beq=[1]
>> x= fmincon('fun',x0,[],[],Aeq,beq,[],[],'confun')
```

Optimum Point : x =[3.3739 0.7913]

```
%fun must be a function : m-file for objective
function f=fun(x)
f=(x(1)-3)^2+(x(2)-3)^2;
%fun must be a function : m-file for constraints
function [c,ceq]=confun(x)
c(1)=x(1)+x(2)^2-4
ceq=[] % nonlinear equality constraint
```

(3) Nonlinear constraint

$$\begin{aligned}\min\ & f(x) = x_1^2 + x_2^2 - 3x_1x_2 \\ \text{s.t.}\ & g_1(x) = \frac{1}{6}x_1^2 + \frac{1}{6}x_2^2 - 1 \le 0 \\ & g_2(x) = -x_1 \le 0 \\ & g_3(x) = -x_2 \le 0\end{aligned}$$

```
>> x0=[1,1] % starting point
>> x= fmincon('fun',x0,[],[],[],[],[],[],[],'confun')
```

Optimum Point : x =[1.7320 1.7320]

```
%fun must be a function : m-file for objective
function f=fun(x)
f=x(1)^2+x(2)^2-3*x(1)*x(2);
%fun must be a function : m-file for constraints
function [c,ceq]=confun(x)
c(1)=(1/6)*x(1)^2+(1/6)*x(2)^2-1
c(2)=-x(1)
c(3)=-x(2)
ceq=[] % nonlinear equality constraint
```

fmincon (3)

(4) Side constraint

$$\min\ f(x) = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)$$
$$\text{s.t. } g_1(x) = x_1x_2 - x_1 - x_2 + 1.5 \le 0$$
$$g_2(x) = -x_1x_2 - 10 \le 0$$
$$x_1 \ge 0,\ x_2 \ge 0$$

```
>> x0=[-1,1] % starting point
>> lb=[0,0]  % Set lower bounds
>> ub=[inf,inf]  % Set upper bounds
>> x= fmincon('fun',x0,[],[],[],[],lb,ub,'confun')
```

Optimum Point : x =[0 1.5000]

```
%fun must be a function : m-file for objective
function f=fun(x)
f=exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1)
%fun must be a function : m-file for constraints
function [c,ceq]=confun(x)
c(1)=1.5+x(1)*x(2)-x(1)-x(2);
c(2)=-x(1)*x(2)-10;
ceq=[]; % nonlinear equality constraint
```

(5) With gradient info

$$\nabla f(x) = \begin{bmatrix} e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1) \\ \qquad\qquad + e^{x_1}(8x_1 + 4x_2) \\ e^{x_1}(4x_1 + 4x_2 + 2) \end{bmatrix}$$

$$\nabla g(x) = \begin{bmatrix} x_2 - 1 & -x_2 \\ x_1 - 1 & -x_1 \end{bmatrix}$$

```
>> x0=[-1,1] % starting point
>> lb=[0,0] % Set lower bounds
>> ub=[] % Set upper bounds
>>options=optimset('LargeScale','off')
>>options=optimset(options,'GradObj','on',
                'GradConstr','on');
>> x= fmincon('fun',x0,[],[],[],[],lb,ub,'confun',
                options);
```

Optimum Point : x =[0 1.5000]

fmincon (4)

```
%fun must be a function : m-file for objective
function [f,df]=fun(x)
f=exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)
+1);
df(1)=exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*
x(2)+1)+exp(x(1))*(8*x(1)+4*x(2));
df(2)=exp(x(1))*(4*x(1)+4*x(2)+2);

%fun must be a function : m-file for constraints
function [c,ceq,dc,dceq]=confun(x)
c(1)=1.5+x(1)*x(2)-x(1)-x(2);
c(2)=-x(1)*x(2)-10;
dc(1,1)=x(2)-1;
dc(1,2)=-x(2)
dc(2,1)=x(1)-1
dc(2,2)=-x(1)
ceq=[] % nonlinear equality constraint
dceq=[];
```

linprog

- Solve a linear programming problem
- Syntax
  - x = linprog(f,A,b,Aeq,beq)
  - x = linprog(f,A,b,Aeq,beq,lb,ub)
  - x = linprog(f,A,b,Aeq,beq,lb,ub,x0)
  - x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
  - [x,fval] = linprog(...)
  - [x,fval,exitflag] = linprog(...)
  - [x,fval,exitflag,output] = linprog(...)
  - [x,fval,exitflag,output,lambda] = linprog(...)

- example

$$\begin{aligned} \min.\ & f(x_1, x_2) = -x_1 - x_2 \\ \text{s.t.}\ & g_1(x) = 2x_1 + 3x_2 - 12 \le 0 \\ & g_2(x) = 2x_1 + x_2 - 8 \le 0 \\ & g_3(x) = -x_1 \le 0 \\ & g_4(x) = -x_2 \le 0 \end{aligned}$$

```
>> f=[-1;-1]
>> A=[2,3;2,1;-1,0;0,-1];
>> b=[12;8;0;0]
>>x= linprog(f,A,b)
```

Optimum Point : x =[3.0000 1.9999]

quadprog

- Solve the quadratic programming problem
- Syntax
  - x = quadprog(H,f,A,b)
  - x = quadprog(H,f,A,b,Aeq,beq)
  - x = quadprog(H,f,A,b,Aeq,beq,lb,ub)
  - x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
  - x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
  - x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options, p1,p2,...)
  - [x,fval] = quadprog(...)
  - [x,fval,exitflag] = quadprog(...)
  - [x,fval,exitflag,output] = quadprog(...)
  - [x,fval,exitflag,output,lambda] = quadprog(...)

- example

$$\min\ f(x_1,x_2)=\frac{1}{2}x_1^2+\frac{1}{2}x_2^2-x_1x_2-2x_1-6x_2$$

$$\text{s.t. } g_1(x)=x_1+x_2-2\le 0$$

$$g_2(x)=-x_1+2x_2-2\le 0$$

$$g_3(x)=2x_1+x_2-3\le 0$$

$$x_1\ge 0,\ x_2\ge 0$$

```
>> H=[1 -1;-1 2]
>> f=[-2;-6]
>> A=[1,1;-1,2;2,1];
>> b=[2;2;3]
>>lb=[0,0]
>>ub=[]
>>x= quadprog(H,f,A,b,[],[],lb,ub)
```

Optimum Point : x =[0.6667 1.3333]

fminimax

- Solve the minimax problem
- Syntax
  - x = fminimax(fun,x0)
  - x = fminimax(fun,x0,A,b)
  - x = fminimax(fun,x0,A,b,Aeq,beq)
  - x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub)
  - x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
  - x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
  - x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,...)
  - [x,fval] = fminimax(...)
  - [x,fval,maxfval] = fminimax(...)
  - [x,fval,maxfval,exitflag] = fminimax(...)
  - [x,fval,maxfval,exitflag,output] = fminimax(...)
  - [x,fval,maxfval,exitflag,output,lambda] = fminimax(...)

- example

Find values of $\boldsymbol{x}$ that minimize the maximum value of $[f_1(x),\ f_2(x),\ f_3(x),\ f_4(x),\ f_5(x)]$

where $f_1(x) = 2x_1^2 + x_2^2 - 48x_1 - 40x_2 + 304$

$f_2(x) = -x_1^2 - 3x_2^2$

$f_3(x) = x_1 + 3x_2 - 18$

$f_4(x) = -x_1 - x_2$

$f_5(x) = x_1 + x_2 - 8$

```
>>x0 = [0.1; 0.1];
>> x = fminimax('fun',x0)
```

Optimum Point : x =[4.0000 4.0000]

```
function f = fun(x)
f(1)= 2*x(1)^2+x(2)^2-48*x(1)-40*x(2)+304;
f(2)= -x(1)^2 - 3*x(2)^2;
f(3)= x(1) + 3*x(2) -18;
f(4)= -x(1)- x(2);
f(5)= x(1) + x(2) - 8;
```