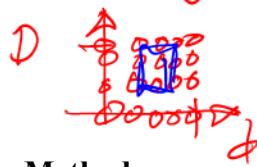


torsion spring N_a d D minimum weight



n09_optimization
$$W = \rho (\pi d^2/4) (\pi D) N_a$$

1 of 14

Numerical Optimization

Methods

- analytical
- exhaustive search
- gradient search
- non-gradient methods

1 design parameter \times

1 output

$$f(x) = x^2 - 4x + 5$$

$$\partial f / \partial x = 2x - 4$$

$$\min(f) \text{ for } \partial f / \partial x = 0$$

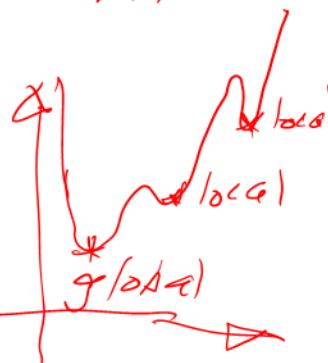
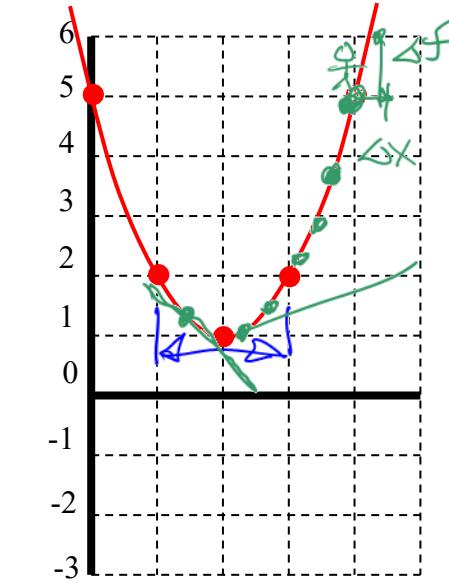
$$x = 2$$

$$x^2 - 4x + 5, 9$$

x	f(x)
0	5
1	2
2	1
3	2
4	5

exhaustive

given x design variable
find $f(x)$ objective



$$\frac{\Delta f}{\Delta x} \approx \frac{\partial f}{\partial x}$$

negative gradient

step size
inversely proportional
to magnitude
of gradient

convergence criteria

Considerations

- multivariable
- multiple objective functions
- computational cost to evaluate objective function
- constraints

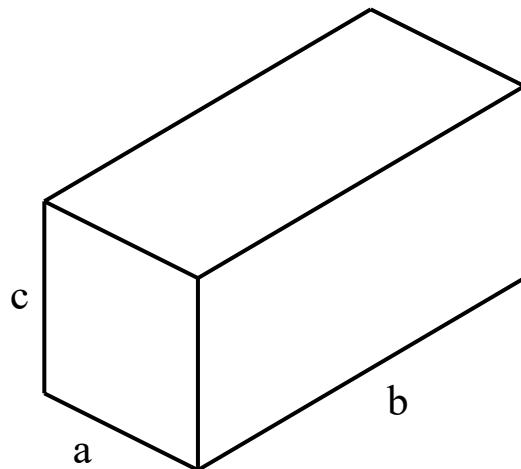
weight
constraint
objectives

fewest number
of evaluations
of objective

- inequality
- exact
- penalty function
- equality

- global versus local minima
- continuous versus discrete variables
- convergence

Lagrange Multiplier for Closed Box



Find minimum area of material A required to construct a closed box with given volume V

$$\text{Minimize } \underline{A = 2ab + 2bc + 2ac} \quad \text{Subject to } abc - V = 0$$

$$L = (2ab + 2bc + 2ac) - \lambda(abc - V)$$

$$\frac{\partial L}{\partial a} = 2b + 2c - \lambda bc = 0$$

$$b = \frac{2c}{\lambda c - 2}$$

$$\frac{\partial L}{\partial b} = 2a + 2c - \lambda ac = 0$$

$$a = \frac{2c}{\lambda c - 2}$$

$$\frac{\partial L}{\partial c} = 2b + 2a - \lambda ab = 0$$

$$\frac{\partial L}{\partial \lambda} = abc - V = 0$$

$$\frac{4c}{\lambda c - 2} + \frac{4c}{\lambda c - 2} - \lambda \frac{4c^2}{(\lambda c - 2)^2} = 0$$

solve for c

$$4c(\lambda c - 2) + 4c(\lambda c - 2) - \lambda(4c^2) = 0 \quad 4c^2\lambda + 4c^2\lambda - 4c^2\lambda = 16c$$

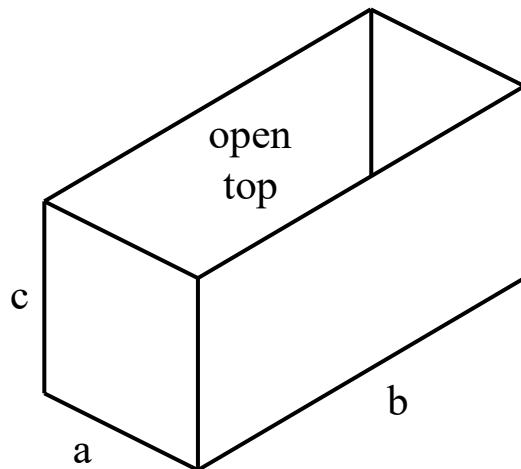
$$c = \frac{4}{\lambda}$$

$$a = \frac{4}{\lambda} \quad b = \frac{4}{\lambda}$$

$$V = abc = \frac{64}{\lambda^3}$$

$$\lambda = \sqrt[3]{\frac{64}{V}}$$

Lagrange Multiplier for Open Box



Find minimum area of material A required to construct an open box with given volume V

$$\text{Minimize } A = ab + 2bc + 2ac \quad \text{Subject to } abc - V = 0$$

$$L = (ab + 2bc + 2ac) - \lambda(abc - V)$$

$$\frac{\partial L}{\partial a} = b + 2c - \lambda bc = 0 \quad b = \frac{2c}{\lambda c - 1}$$

$$\frac{\partial L}{\partial b} = a + 2c - \lambda ac = 0 \quad a = \frac{2c}{\lambda c - 1}$$

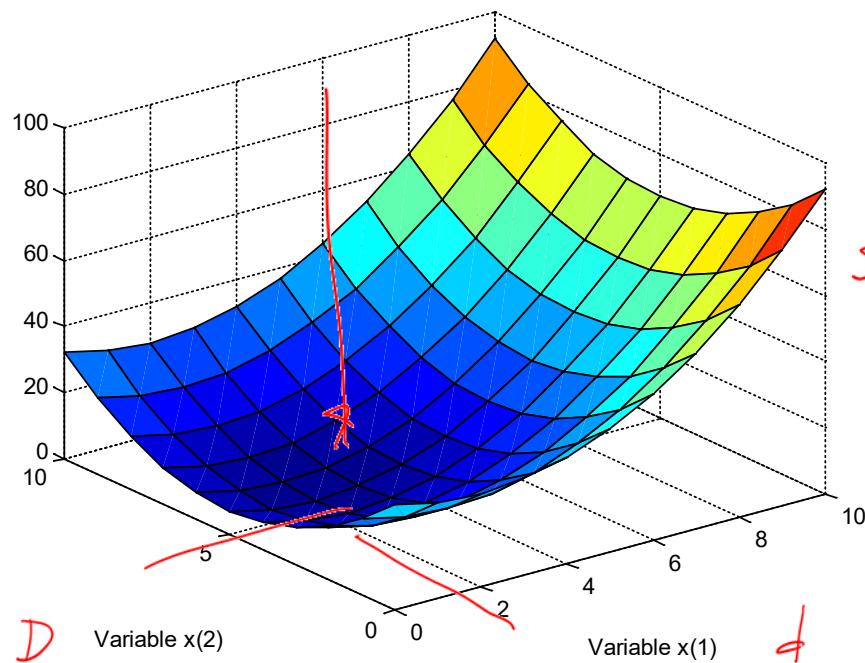
$$\frac{\partial L}{\partial c} = 2b + 2a - \lambda ab = 0$$

$$\frac{\partial L}{\partial \lambda} = abc - V = 0$$

$$\frac{4c}{\lambda c - 1} + \frac{4c}{\lambda c - 1} - \lambda \frac{4c^2}{(\lambda c - 1)^2} = 0$$

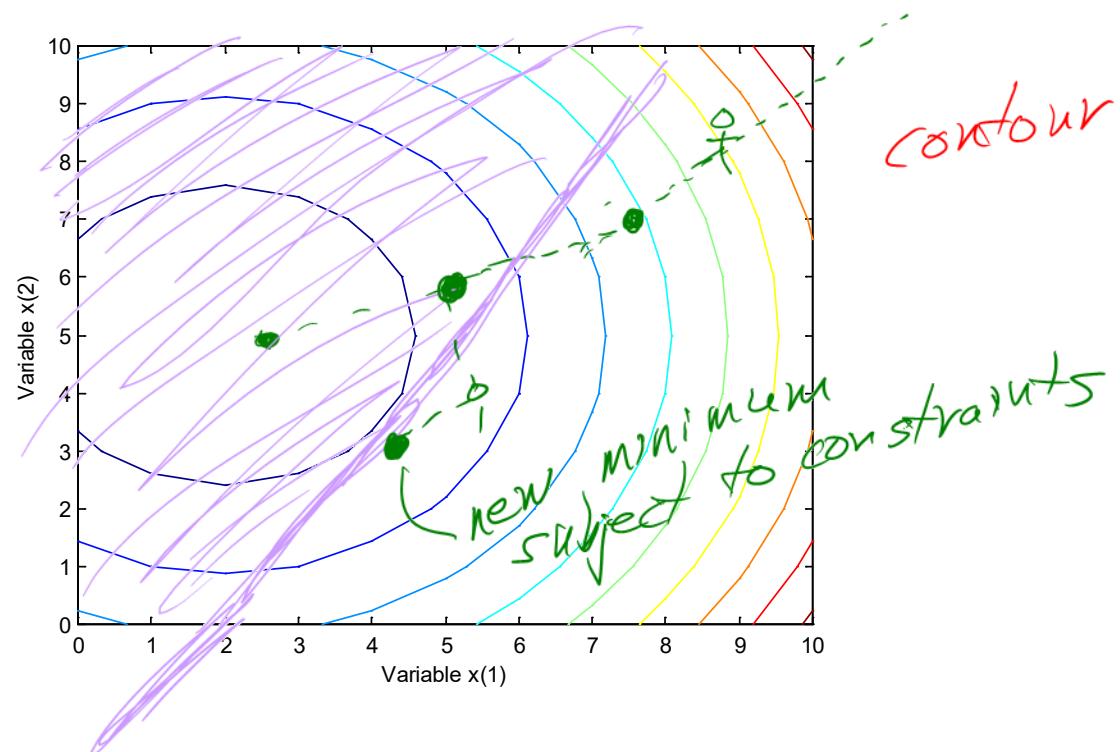
$$4c(\lambda c - 1) + 4c(\lambda c - 1) - \lambda(4c^2) = 0 \quad c = \frac{2}{\lambda} \quad a = \frac{4}{\lambda} \quad b = \frac{4}{\lambda}$$

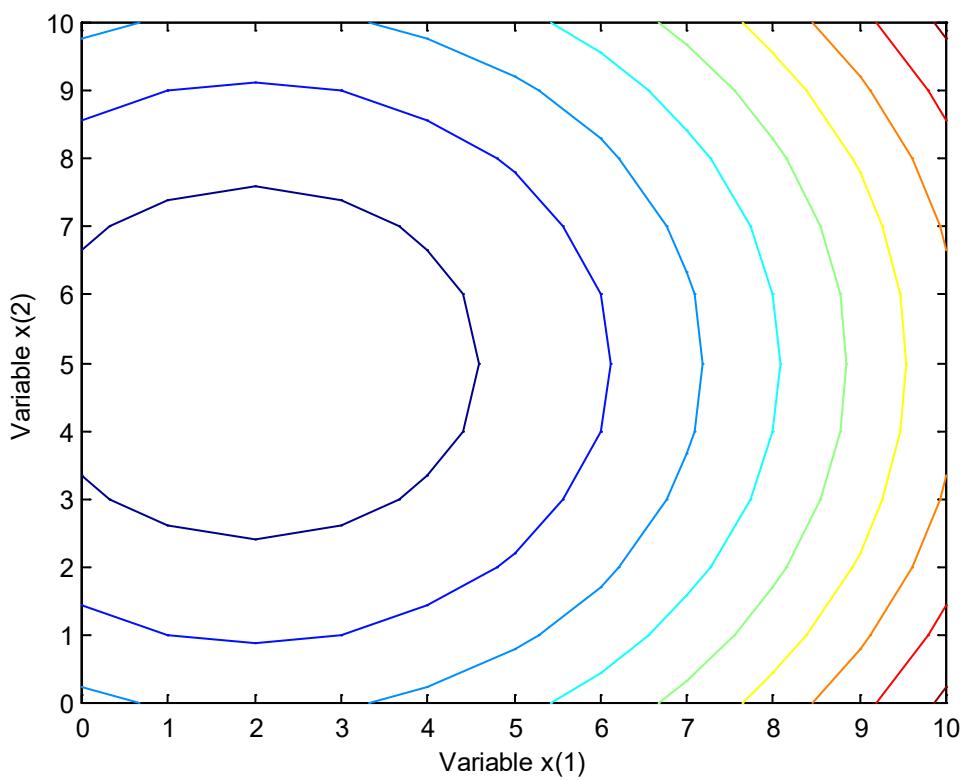
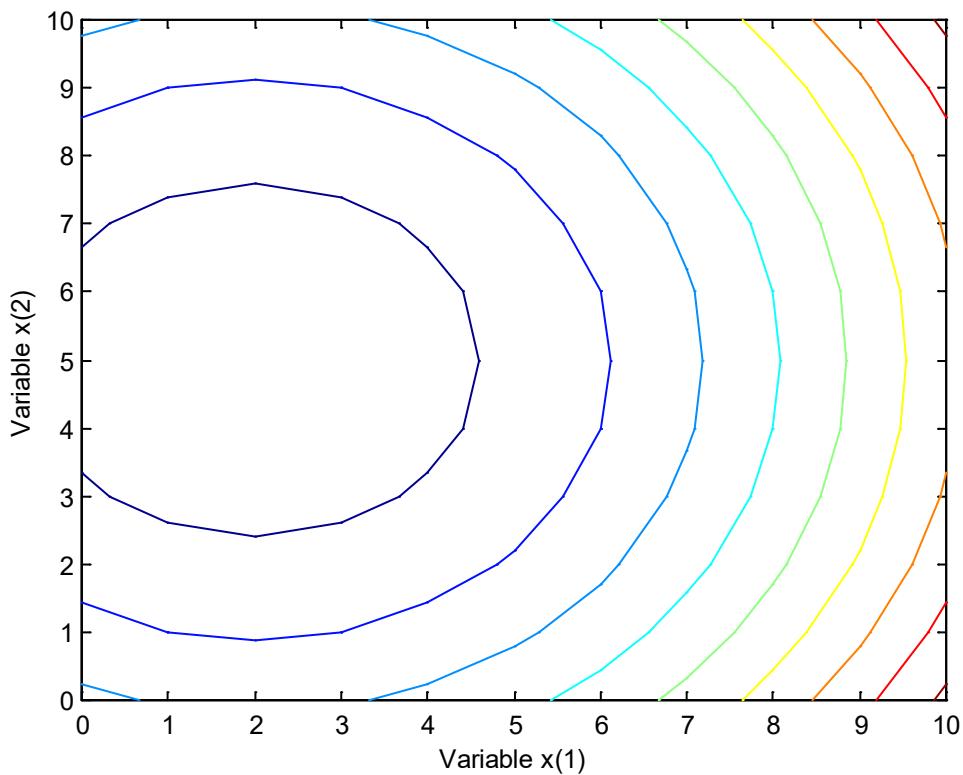
$$V = abc = \frac{32}{\lambda^3} \quad \lambda = \sqrt[3]{\frac{32}{V}}$$



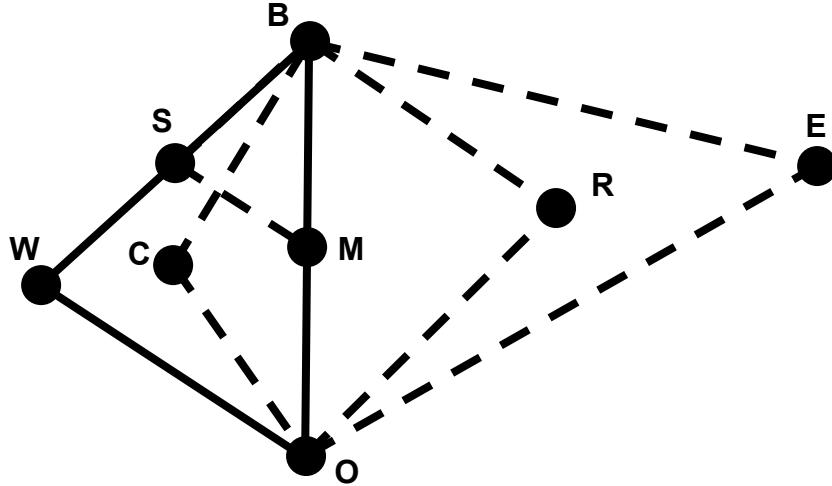
$$z = (x_1^2 - 4x_1 + 4) + (x_2^2 - 10x_2 + 25) + 3$$

$$\partial z / \partial x_1 = 2x_1 - 4 \quad \partial z / \partial x_2 = 2x_2 - 10$$





Simplex Minimization (Nelder-Mead)



current simplex
B = best
W = worst
O = others
new vertices
R = reflect
E = expand
C = contract
S = shrink

k = number of design parameters

$\{x\} = \{x_1 \quad x_2 \quad \dots \quad x_i \quad \dots \quad x_k\}^T$ = column vector of design parameters

$n = k+1$ = number of vertices for simplex

$\{\{x\}_1 \quad \{x\}_2 \quad \dots \quad \{x\}_j \quad \dots \quad \{x\}_n\}$ = list of vertices (for $j=1$ to n)

z_j = objective function evaluated at all $\{x\}_j$

$\{x\}_B$ = best vertex B based on objective function ($z_B < z_j < z_W$)

$\{x\}_W$ = worst vertex W based on objective function ($z_B < z_j < z_W$)

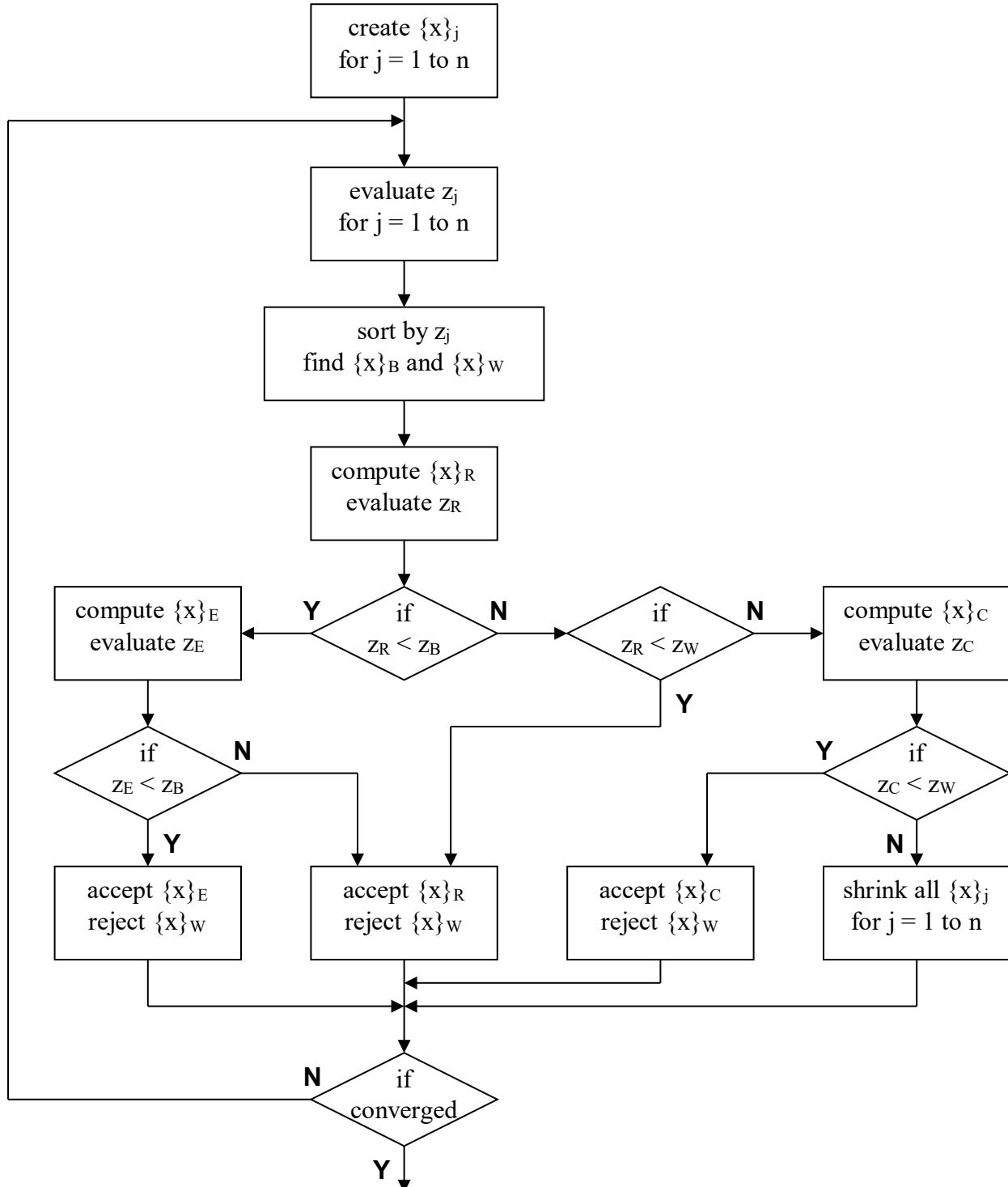
$\{x\}_M = \left[\left(\sum_{j=1}^n \{x\}_j \right) - \{x\}_W \right] / (n - 1)$ = mean of vertices M excluding the worst

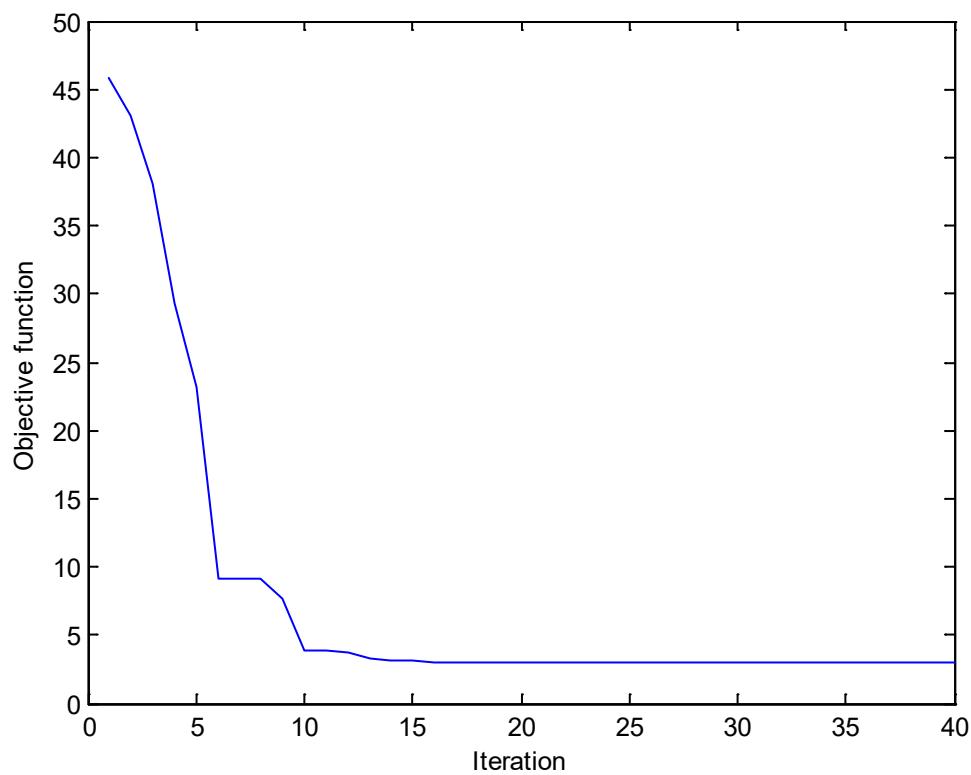
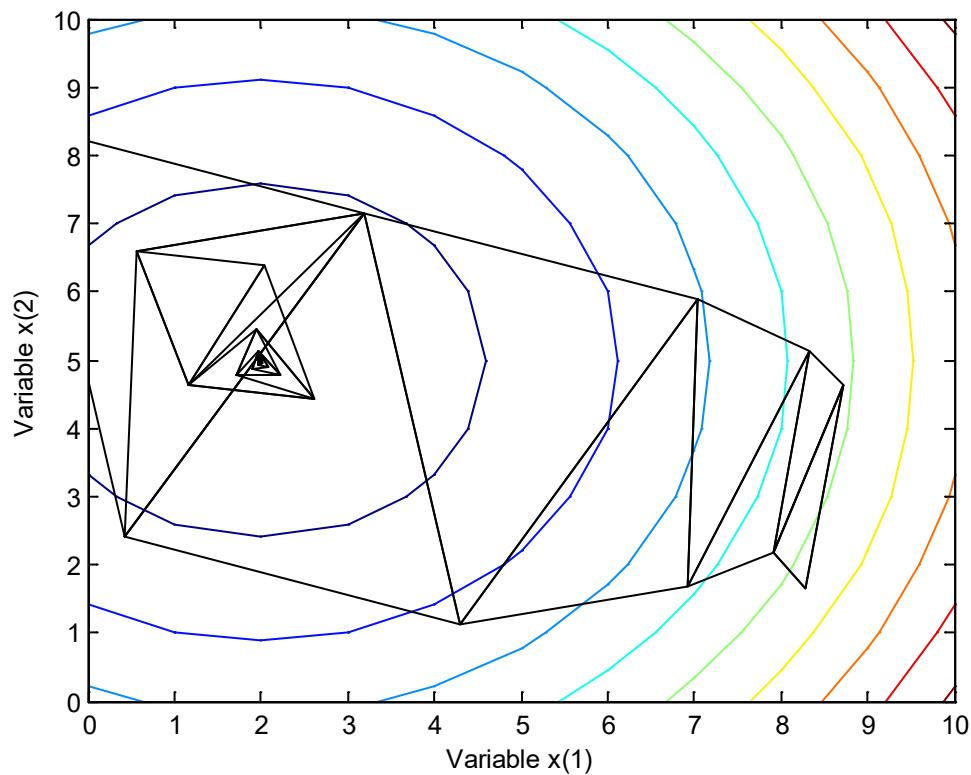
$\{x\}_R = \{x\}_M + (\{x\}_M - \{x\}_W) = 2\{x\}_M - \{x\}_W$ = new reflected vertex R

$\{x\}_E = \{x\}_M + 2(\{x\}_M - \{x\}_W) = 3\{x\}_M - 2\{x\}_W$ = new expanded vertex E

$\{x\}_C = \{x\}_M - 0.5(\{x\}_M - \{x\}_W) = (\{x\}_M + \{x\}_W) / 2$ = new contracted vertex C

$\{x\}_{j_NEW} = (\{x\}_{j_OLD} + \{x\}_B) / 2$ = new shrunken vertices S (for $j = 1$ to n)





```
% simplex.m - Simplex minimization per Kuester/Mize p. 298
% HJSIII, 12.10.15

clear

% 2D and 3D plots of biquadratic surface
% z = ( x(1)^2 - 4*x(1) + 4 ) + ( x(2)^2 - 10*x(2) + 25 ) + 3
biquad_surf

% maximum number of iterations
maxiter = 50;

% initial guess in column vector
guess = [ 8 3 ]'; % 2D biquadratic example

% random radius around start
rad = 1;

% convergence distance
eps = 0.001;

% k = number of variables
% n = number of vertices in simplex
k = length( guess );
n = k + 1;

% random initial simplex
simp = guess*ones(1,n) + rad * randn(k,n);

% evaluate each vertex using external function
neval = 0;
for j = 1 : n,
    neval = neval + 1;
    z(1,j) = simplex_eval( simp(:,j) ); % external function
end

% plot initial simplex - use countour plot from biquad_surf
hold on
h = patch( simp(1,:), simp(2,:), 'b' );
set( h, 'FaceColor', 'none' )

% iteration loop
niter = 0;
while niter <= maxiter
    niter = niter + 1;

    % check convergence
    del = max( max(simp') - min(simp') );
    if del < eps,
        break
    end

    % plot each new iteration
    h = patch( simp(1,:), simp(2,:), 'b' );
    set( h, 'FaceColor', 'none' )

    % wait for keystroke
    pause

    % sort in ascending order to find min/max
    [ y, ind ] = sort( z );
    z_b = y(1);
    i_b = ind(1);
    x_b = simp(:,i_b);

    z_w = y(n);
    i_w = ind(n);
    x_w = simp(:,i_w);

    % hold convergence history
    holdbest(niter) = z_b;
```

```

% reflect about centroid excluding the worst
cen = ( sum(simp,2) - x_w ) / (n-1);
d = cen - x_w;
x_r = cen + d;

% evaluate reflection using external function
neval = neval + 1;
z_r = simplex_eval( x_r );    % external function

% try expansion if reflection is better than best
if z_r < z_b,
  x_e = cen + 2 * d;

% evaluate expansion using external function
neval = neval + 1;
z_e = simplex_eval( x_e );    % external function

% keep the better of expansion or reflection
if z_e < z_b,
  simp(:,i_w) = x_e;
  z(i_w) = z_e;
else
  simp(:,i_w) = x_r;
  z(i_w) = z_r;
end
else

% keep reflection if better than worst
if z_r < z_w,
  simp(:,i_w) = x_r;
  z(i_w) = z_r;
else

% try contraction if reflection is worse than worst
x_c = cen - 0.5 * d;

% evaluate contraction using external function
neval = neval + 1;
z_c = simplex_eval( x_c );    % external function

% keep contraction if better than worst
if z_c < z_w,
  simp(:,i_w) = x_c;
  z(i_w) = z_c;
else

% otherwise shrink toward best
for j = 1:n,
  simp(:,j) = ( simp(:,j) + x_b ) / 2;

% evaluate each shrink using external function
neval = neval + 1;
z(1,j) = simplex_eval( simp(:,j) );    % external function
end % shrink
end % contraction
end % expansion
end % reflection

% next iteration
end

x_b
z_b

% plot convergence history
figure( 3 )
clf
plot( holdbest )
xlabel( 'Iteration' )

```

```
ylabel( 'Objective function' )  
% bottom of simplex
```

```
function z = simplex_eval( x )
% biquadratic test function for fminsearch
% HJSIII, 12.10.15

% unconstrained minimum = 3 at x(1)=2 and x(2)=5
z = ( x(1)^2 - 4*x(1) + 4 ) + ( x(2)^2 - 10*x(2) + 25 ) + 3;

% penalty function to provide inequality constraint
% constrained minimum = 3.8 at x(1)=2.4 and x(2)=4.2
%t = 0.5 * x(1) + 3;
%if x(2) > t,
%  z = z + 100;
%end

% bottom of simplex_eval

+++++++
% biquad_surf.m - plot contours of biquadratic
% HJSIII, 12.10.15

clear

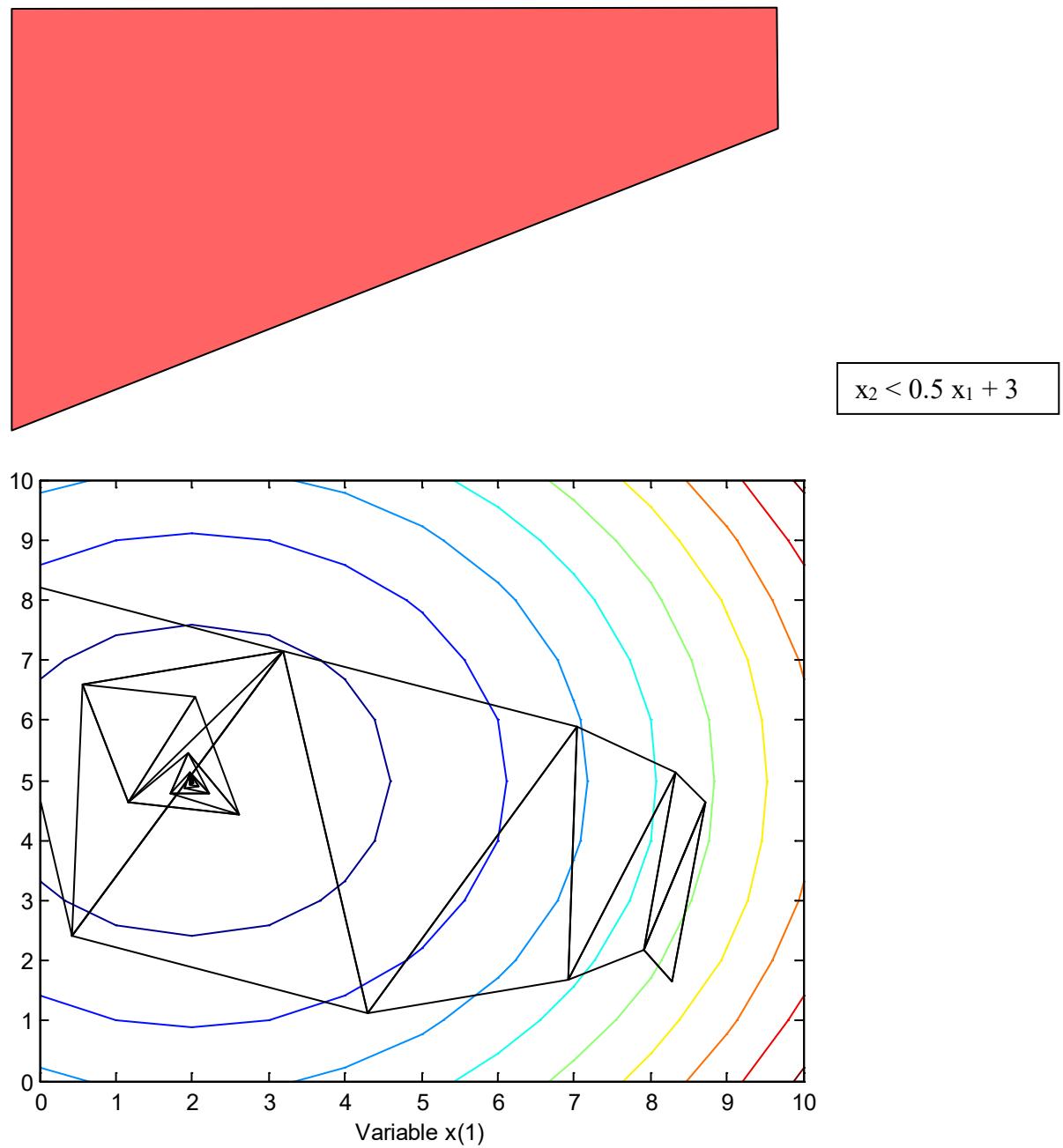
% design space
x1_val = ( 0 : 1 : 10 )';
x2_val = ( 0 : 1 : 10 )';
n1 = length( x1_val );
n2 = length( x2_val );

% exhaustive search
for i = 1 : n1,
    x(1) = x1_val(i);
    for j = 1 : n2;
        x(2) = x2_val(j);
        z_ij(i,j) = ( x(1)^2 - 4*x(1) + 4 ) + ( x(2)^2 - 10*x(2) + 25 ) + 3;
    end
end

% plot surface
z_ij = z_ij';
figure( 1 )
clf
h_surf = surf( x1_val, x2_val, z_ij );
xlabel( 'Variable x(1)' )
ylabel( 'Variable x(2)' )

% plot contours
figure( 2 )
clf
h_cont = contour( x1_val, x2_val, z_ij );
xlabel( 'Variable x(1)' )
ylabel( 'Variable x(2)' )

% bottom of biquad_surf
```



add penalty function to objective if constraint is violated

```
>> help fminsearch
FMINSEARCH Multidimensional unconstrained nonlinear minimization (Nelder-Mead).
X = FMINSEARCH(FUN,X0) starts at X0 and attempts to find a local minimizer
X of the function FUN. FUN is a function handle. FUN accepts input X and
returns a scalar function value F evaluated at X. X0 can be a scalar, vector
or matrix.

X = FMINSEARCH(FUN,X0,OPTIONS) minimizes with the default optimization
parameters replaced by values in the structure OPTIONS, created
with the OPTIMSET function. See OPTIMSET for details. FMINSEARCH uses
these options: Display, TolX, TolFun, MaxFunEvals, MaxIter, FunValCheck,
PlotFcns, and OutputFcn.

X = FMINSEARCH(PROBLEM) finds the minimum for PROBLEM. PROBLEM is a
structure with the function FUN in PROBLEM.Objective, the start point
in PROBLEM.x0, the options structure in PROBLEM.options, and solver
name 'fminsearch' in PROBLEM.solver. The PROBLEM structure must have
all the fields.

[X,FVAL] = FMINSEARCH(...) returns the value of the objective function,
described in FUN, at X.

[X,FVAL,EXITFLAG] = FMINSEARCH(...) returns an EXITFLAG that describes
the exit condition of FMINSEARCH. Possible values of EXITFLAG and the
corresponding exit conditions are

1 Maximum coordinate difference between current best point and other
points in simplex is less than or equal to TolX, and corresponding
difference in function values is less than or equal to TolFun.
0 Maximum number of function evaluations or iterations reached.
-1 Algorithm terminated by the output function.

[X,FVAL,EXITFLAG,OUTPUT] = FMINSEARCH(...) returns a structure
OUTPUT with the number of iterations taken in OUTPUT.iterations, the
number of function evaluations in OUTPUT.funcCount, the algorithm name
in OUTPUT.algorithm, and the exit message in OUTPUT.message.
```

Examples

```
FUN can be specified using @:
X = fminsearch(@sin,3)
finds a minimum of the SIN function near 3.
In this case, SIN is a function that returns a scalar function value
SIN evaluated at X.

FUN can also be an anonymous function:
X = fminsearch(@(x) norm(x,[1;2;3])
returns a point near the minimizer [0;0;0].
```

If FUN is parameterized, you can use anonymous functions to capture the problem-dependent parameters. Suppose you want to optimize the objective given in the function myfun, which is parameterized by its second argument c. Here myfun is an M-file function such as

```
function f = myfun(x,c)
f = x(1)^2 + c*x(2)^2;
```

To optimize for a specific value of c, first assign the value to c. Then create a one-argument anonymous function that captures that value of c and calls myfun with two arguments. Finally, pass this anonymous function to FMINSEARCH:

```
c = 1.5; % define parameter first
x = fminsearch(@(x) myfun(x,c),[0.3;1])
```

FMINSEARCH uses the Nelder-Mead simplex (direct search) method.

See also optimset, fminbnd, function_handle.