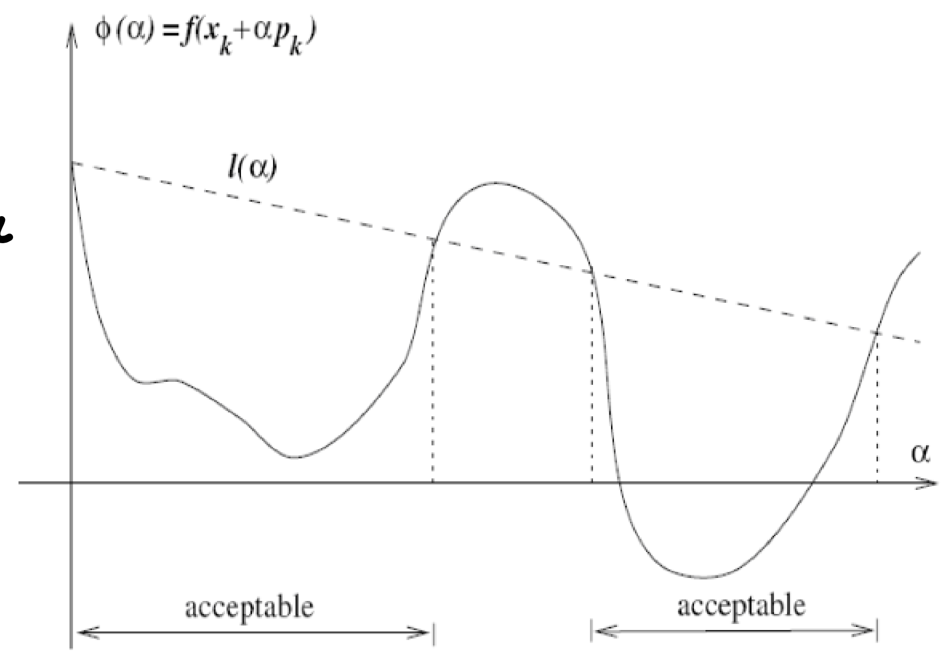
Lecture 5

Inexact line search, Order Zero Unconstrained Optimization

Sufficient Reduction Condition

* $f(\underline{x}_k + \alpha \underset{\sim}{p}_k) \leq f(\underline{x}_k) + c_1 \alpha \nabla f(\underline{x}_k)^T \underset{\sim}{p}_k$

* This condition may return small steps for small values of α.



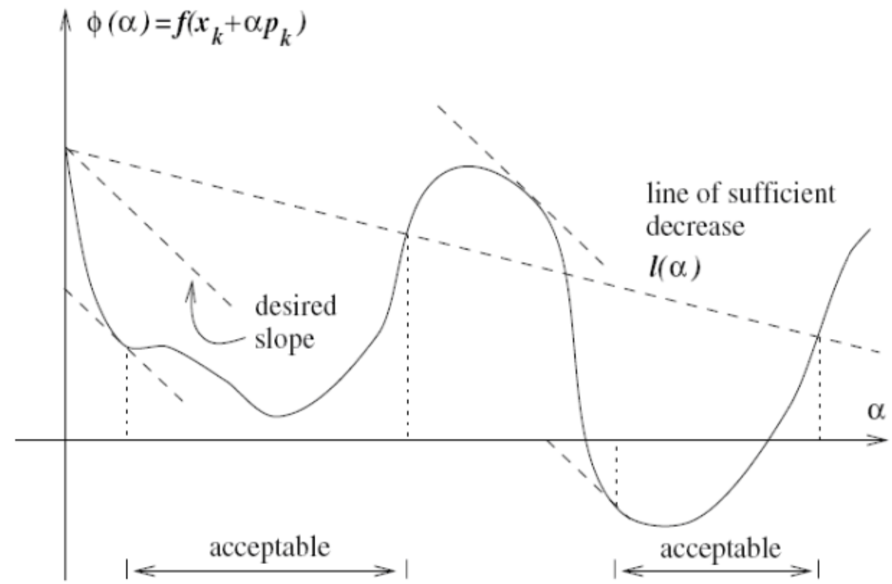
Nocedal et al.

Wolf's Conditions

$$f(\underline{x}_k + \alpha \underset{\sim}{p}_k) \leq f(\underline{x}_k) + c_1 \alpha \nabla f(\underline{x}_k)^T \underset{\sim}{p}_k$$

$$\nabla f(\underline{x}_k + \alpha \underset{\sim}{p}_k)^T \underset{\sim}{p}_k \geq c_2 \nabla f(\underline{x}_k)^T \underset{\sim}{p}_k$$

$$0 < c_1 < c_2 < 1$$



Nocedal et al.

Why Unconstrained optimization

* Most practical optimization problems are constrained.
* Sometimes the constraints are redundant!
* Understanding unconstrained optimization is useful for constrained optimization.
* They are classified into order zero, order 1, and order 2 techniques.

Random Jumping Method

* This techique assumes that the solution is bracketed in an n-dimensional interval $\quad l_i \le x_i \le u_i \;, \quad i = 1, 2, \dots, n$

* A large number of random points are evaluated in this interval

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} l_1 + r_1(u_1 - l_1) \\ l_2 + r_2(u_2 - l_2) \\ \vdots \\ l_n + r_n(u_n - l_n) \end{bmatrix}, \quad r_i \in [0, 1] \text{ are random numbers}$$

MATLAB Code

```matlab
%This program carries out the random jump algorithm
%A sequence of random points is generated within the region of interest
%first introduce the region of solution in the n_dimensional space
NumberOfParameters=2; %This is n for this problem
UpperValues=[10     10]'; %upper values
LowerValues=[-10   -10]'; %lower values
OldPoint=0.5*(UpperValues+LowerValues); %select center of interval as old point
OldValue=getObjective(OldPoint); %Get the objective function at the old point
MaximumNumberOfIterations=1000; %maximum number of allowed iterations
IterationCounter=0; %iteration counter
while(IterationCounter<MaximumNumberOfIterations) %repeat until maximum number of iteration is achieved
    RandomVector=rand(NumberOfParameters,1); %get a vector of random variables
    NewPoint=LowerValues+RandomVector.*(UpperValues-LowerValues); %Get new random point
    NewValue=getObjective(NewPoint); %get new value
    if(NewValue<OldValue) %is there an improvement?.  Then store the new point and value
        OldPoint=NewPoint;
        OldValue=NewValue;
    end
    IterationCounter=IterationCounter+1; %increment the iteration counter
end
OldPoint
OldValue
```

Example

Minimize the function

$$f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$$

in the interval $-10 \leq x_i \leq 10$ using the random jump method.

Code output:

```
IterationCounter = 100     OldPoint =[-1.4815   1.9512]     OldValue = -1.0172
IterationCounter = 1000    OldPoint =[-0.9644   1.4277]     OldValue = -1.2474
```

Analytic Solution

$$\underset{\sim}{\nabla} f = \begin{bmatrix} 1 + 4x_1 + 2x_2 \\ -1 + 2x_1 + 2x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Solving, we get $\underset{\sim}{x}^* = \begin{bmatrix} -1.0 & 1.5 \end{bmatrix}^T$

$$\underset{\sim}{H} = \begin{bmatrix} 4 & 2 \\ 2 & 2 \end{bmatrix}$$ (+ve definite, why?)

<u>Random Walk</u>

* At the ith iteration, the current solution is $\underline{x}_i$.
* A new suggested point is created by taking a step λ in a random direction

$$\underline{x}_s = \underline{x}_i + \lambda \underset{\sim}{u}$$

* A maximum of N trials is allowed. The step λ is reduced afterwards.
* A point is accepted if it gives a reduction

$$f(\underline{x}_s) < f(\underline{x}_i) \Rightarrow \underline{x}_{i+1} = \underline{x}_s$$

MATLAB CODE

```
NumberOfParameters=2; %This is n for this problem
OldPoint=[0  0]'; %This is the starting point
OldValue=getObjective(OldPoint); %Get the objective function at the old point
OnesVector=ones(NumberOfParameters,1); %get a vector of ones
negativeOnesVector=-1.0*OnesVector; %get a vector of -ve ones
MaximumNumberOfIterations=100; %maximum number of allowed iterations
IterationCounter=0;  %iteration counter
LambdaMax=3; %maximum value of Lambda
Tolerance=0.001;
while(IterationCounter<MaximumNumberOfIterations) %repeat until maximum number of iteration is achieved
    RandomVector=rand(NumberOfParameters,1); %get a vector of random variables
    u=negativeOnesVector+RandomVector.*(OnesVector-negativeOnesVector); %make the random vector between -1 and 1
for each component
    LambdaOptimal = GoldenSection('getObjective',Tolerance,OldPoint,u,LambdaMax); %get the optimal lambda
    NewPoint=OldPoint+LambdaOptimal*u; %Get new random point
    NewValue=getObjective(NewPoint); %get new value
    if(NewValue<OldValue) %is there an improvement?.  Then store the new point and value
        OldPoint=NewPoint;
        OldValue=NewValue;
    end
    IterationCounter=IterationCounter+1; %increment the iteration counter
    pause
end
```

Code Output

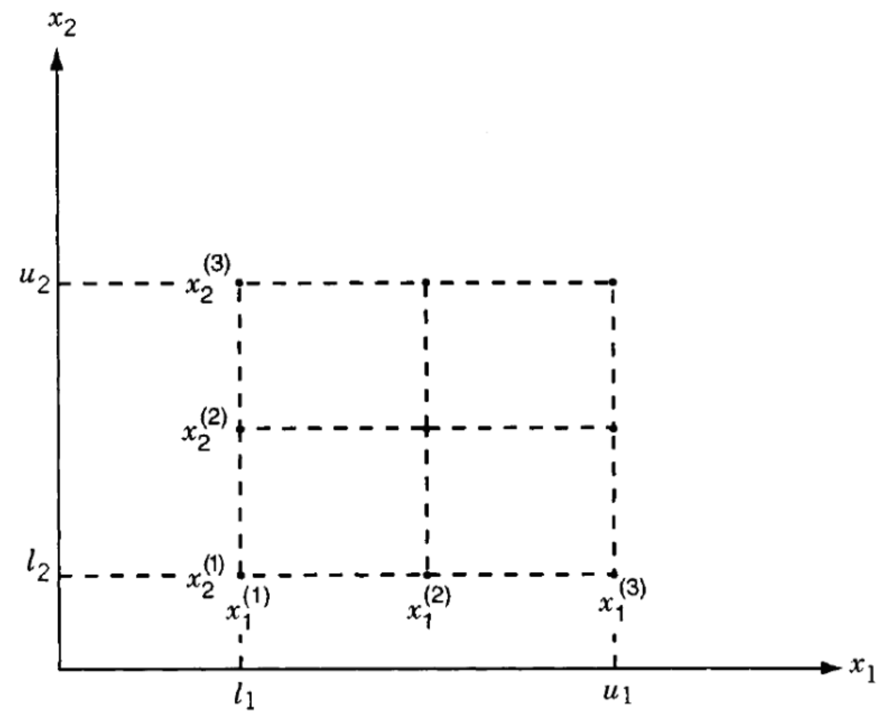
Applying the code to the function $f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$, we get the output

```
OldPoint = [0   0]'   OldValue = 0   LambdaOptimal = 0.9969
u = [0.0442   0.3353]'  NewPoint = [0.0441   0.3343]'  NewValue =-0.1451

OldPoint = [0.0441  0.3343]'  OldValue = -0.1451  LambdaOptimal =1.8230
u =[-0.1383  0.4191]'  NewPoint =[-0.2081  1.0983]  NewValue = -0.4707

OldPoint =[-0.2081  1.0983]  OldValue = -0.4707  LambdaOptimal = 0.3375
u =[-0.9456  -0.9202]   NewPoint =[-0.5273  0.7877]  NewValue =-0.9692

OldPoint =[-1.0000  1.4999]'  OldValue =-1.2500  LambdaOptimal = 4.2009e-004
u =[-0.2312  0.3371]'  NewPoint =[-1.0001  1.5000]' NewValue = -1.2500
```

Samples of successful iterations are shown!

Grid Search Method

* A grid of points is generated.
* The objective function is then evaluated at these points
* The number of points is p^n for p points in every direction.



Univariate Method

* At every iteration, we search in every coordinate direction separately.

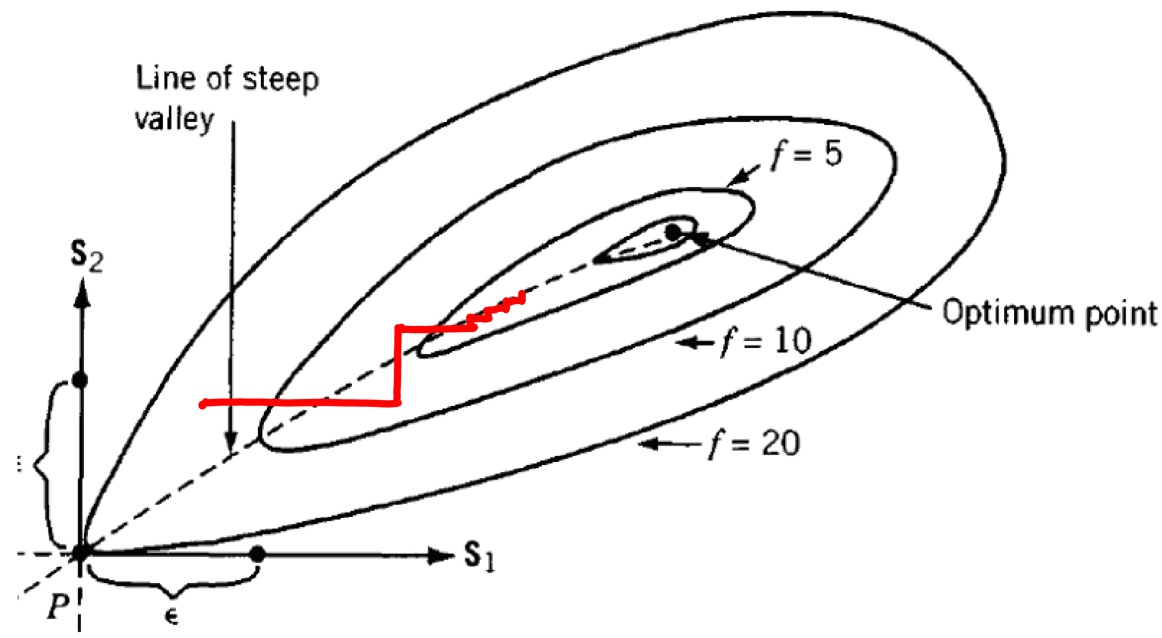
* $\underline{x}_{i+1} = \min\limits_{\lambda_i} f(\underline{x}_i + \lambda_i \underline{e}_i)$ (λ is +ve or -ve)

$\underline{e}_i = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $\quad i = 1, n+1, 2n+1, \ldots$

$\underline{e}_i = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$ $\quad i = 2, n+2, 2n+2, \ldots$

$\underline{e}_i = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$ $\quad i = n, 2n, 3n, \ldots$

Univariate Method (Cont'd)



* This approach may not Converge or it may exhibit slow convergence.

MATLAB CODE

```
NumberOfParameters=3; %This is n for this problem
OldPoint=[3  -3  5]'; %This is the starting point
OldValue=getObjective(OldPoint); %Get the objective function at the old point
Idenitity =eye(NumberOfParameters); %Get identity matrix of size n
negativeIdenitity=-1.0*Idenitity; % Get matrix of negative identity
MaximumNumberOfIterations=100; %maximum number of allowed iterations
IterationCounter=0;  %iteration counter
LambdaMax=3; %maximum value of Lambda
Tolerance=0.001; %terminating tolerance for line search
Epsilon=0.001; %exploration step
while(IterationCounter<MaximumNumberOfIterations) %repeat
    for i=1:NumberOfParameters
      up=Idenitity(:,i); %get the vector ei
      un=-1.0*up; %get the vector -ei
      %we do first exploration in the -ve and +ve directions
      fp=feval('getObjective',OldPoint+Epsilon*up); %get +ve pertubed function value
      if(fp<OldValue) %positive direction is promising
          u=up; %choose the positive coordinate direction
      else
          u=un; %choose the negative coordinate direction
      end
      LambdaOptimal = GoldenSection('getObjective',Tolerance,OldPoint,u,LambdaMax); %get the optimal value
      NewPoint=OldPoint+LambdaOptimal*u; %Get new rpoint
      NewValue=getObjective(NewPoint); %get new value
      if(NewValue<OldValue) %is there an improvement?.  Then store the new point and value
        OldPoint=NewPoint;
        OldValue=NewValue;
      end
    end
    IterationCounter=IterationCounter+1; %increment the iteration counter
end
```

Example

Utilize the univariate approach to find the minimum of the function

$$f(x_1, x_2, x_3) = x_1^2 + 3x_2^2 + 6x_3^2$$

Solution: exact solution is $\underline{x}^* = [0 \quad 0 \quad 0]^T$.

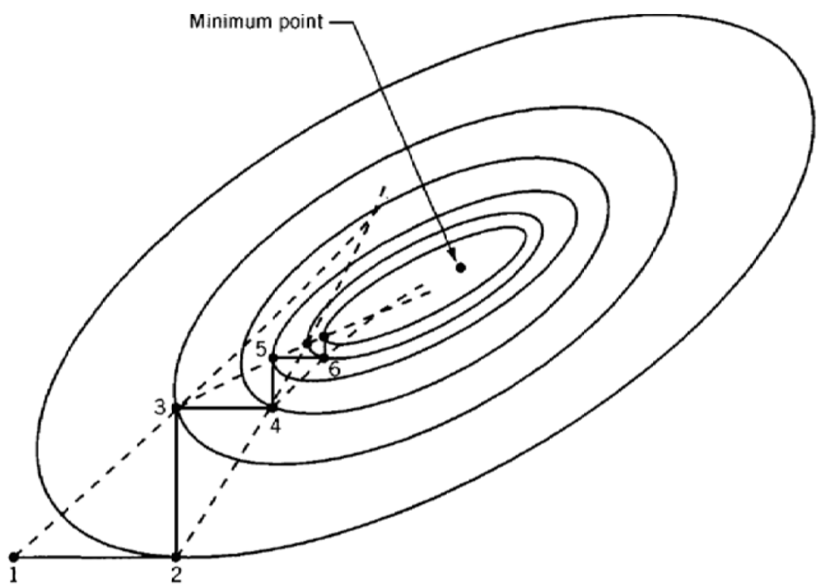
The final code output is

IterationCounter = 100   OldPoint =1.0e-003 *[-0.0000   -0.4201   0.0711]'   OldValue = 5.5979e-007

Pattern Search Methods

* These methods attempt at improving the univariate approach.
* In addition to searching along the coordinate directions, we also search along the "pattern directions".
* The convergence of these methods is better than the univariate method.

Patter Search Methods (Cont'd)



* For a quadratic function, pattern directions go through the minimum!.
* The ith pattern search direction is given by

$$\underset{\sim}{s}_i = \underset{\sim}{x}_i - \underset{\sim}{x}_{i-n} \quad \text{(n number of parameters)}$$

Hooke and Jeever Method

* At the ith step, we first do an exploratory move: $\underset{\sim}{Y}_{k0} = \underline{X}_k$

$$\underset{\sim}{Y}_{ki} = \begin{cases} \underset{\sim}{Y}_{k,i-1} + \Delta x_i \underset{\sim}{u}_i & \text{if } f^+ = f(\underset{\sim}{Y}_{k,i-1} + \Delta x_i \underset{\sim}{u}_i) < f = f(\underset{\sim}{Y}_{k,i-1}) \\ \underset{\sim}{Y}_{k,i-1} - \Delta x_i \underset{\sim}{u}_i & \text{if } f^- = f(\underset{\sim}{Y}_{k,i-1} - \Delta x_i \underset{\sim}{u}_i) < f = f(\underset{\sim}{Y}_{k,i-1}) \\ \underset{\sim}{Y}_{k,i-1} & \text{if } f = f(\underset{\sim}{Y}_{k,i-1}) < \min(f^+, f^-) \end{cases}$$

* If $\underset{\sim}{Y}_{k,n} = \underline{X}_k \Rightarrow$ exploration failed $\Rightarrow$ reduce the lengths Δx_i

Hooke and Jeeves Method (Cont'd)

* If $\underset{\sim}{y}_{k,n}$ is different from $\underset{\sim}{x}_k \Longrightarrow$ exploration successful $\Longrightarrow \underline{x}_{k+1} = \underset{\sim}{y}_{k,n}$
* pattern move: Determine the pattern direction

$$\underset{\sim}{s} = \underline{x}_{k+1} - \underline{x}_k$$

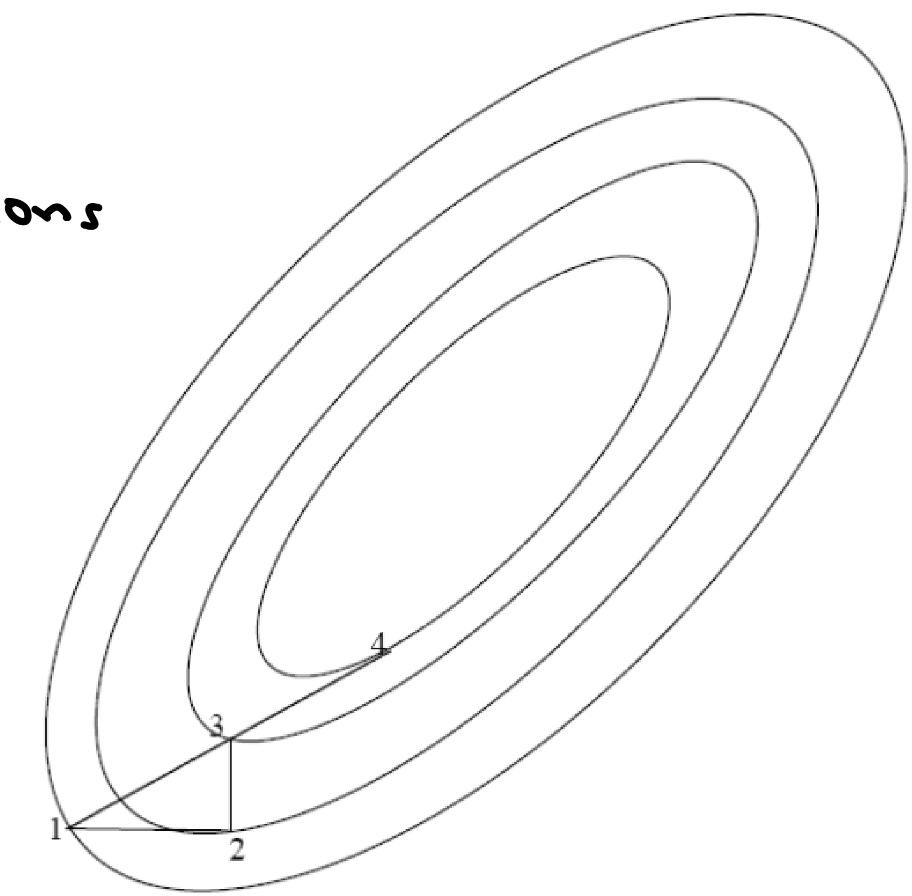
* Determine the minimum along this direction

$$\lambda^* = \min_{\lambda} f(\underline{x}_{k+1} + \lambda \underset{\sim}{s}) \Longrightarrow \underset{\sim}{y}_{k+1,0} = \underset{\sim}{x}_{k+1} + \lambda^* \underset{\sim}{s}$$

* Do exploration again and repeat.

Hooke and Jeever Method (Cont'd)

fixed steps along coordinate directions and line search along the pattern search direction



MATLAB CODE

```
NumberOfParameters=3; %This is n for this problem
OldPoint=[3  -3  5]' %This is the starting point
OldValue=getObjective(OldPoint) %Get the objective function at
the old point
Idenitity =eye(NumberOfParameters); %Get identity matrix of
size n
negativeIdenitity=-1.0*Idenitity; % Get matrix of negative
identity
LambdaMax=3; %maximum value of Lambda
Tolerance=0.001; %terminating tolerance for line search
Epsilon=0.001; %exploration step
StepNorm=1000; %initialize stepNorm
MinimumDistance=1.0e-4; %termination condition
while(StepNorm>MinimumDistance) %repeat
   %first we do search in the directions of the coordinates
    YOld=OldPoint; %start exploring from the current point
    YOldValue=OldValue; %store also the old value
    for i=1:NumberOfParameters %repeat for all coordinates
      up=Idenitity(:,i); %get the vector ei
      un=-1.0*up; %get the vector -ei
      fp=feval('getObjective',YOld+Epsilon*up);
      if(fp<OldValue) %positive direction is promising
          u=up; %choose the positive coordinate direction
      else
          u=un; %choose the negative coordinate direction
      end
LambdaOptimal = GoldenSection
('getObjective',Tolerance,YOld,u,LambdaMax);
      YNew=YOld+LambdaOptimal*u; %Get new  exploration
      YNewValue=feval('getObjective',YNew);
      if(YNewValue<YOldValue) %is there an improvement?
        YOld=YNew;
        YOldValue=YNewValue;
      end
    end
    PatternDirection=YOld-OldPoint; % pattern direction
    %now we do a line search in this direction starting from
YOld
    LambdaOptimal = GoldenSection
('getObjective',Tolerance,YOld,PatternDirection,LambdaMax);
    NewPoint=YOld+LambdaOptimal*PatternDirection;
    NewValue=feval('getObjective',NewPoint);
    StepNorm=norm(NewPoint-OldPoint);
    OldPoint=NewPoint %update the current point
    OldValue=NewValue %update the current value
end
```

Example

Utilize the univariate approach to find the minimum of the function $f(x_1, x_2, x_3) = x_1^2 + 3x_2^2 + 6x_3^2$ starting from $\underline{x}^{(0)} = [3 \quad -3 \quad 5]^T$.

Code output

```
OldPoint = [3      -3      5]          OldValue = 186
OldPoint =[-0.0008  0.0008  5.0000]          OldValue =150.0
OldPoint =1.0e-003*[0.7258   -0.7258   -0.1984]  OldValue =2.3436e-006
OldPoint =1.0e-003 *[-0.2139  0.2139   -0.1984]  OldValue =4.1926e-007
```

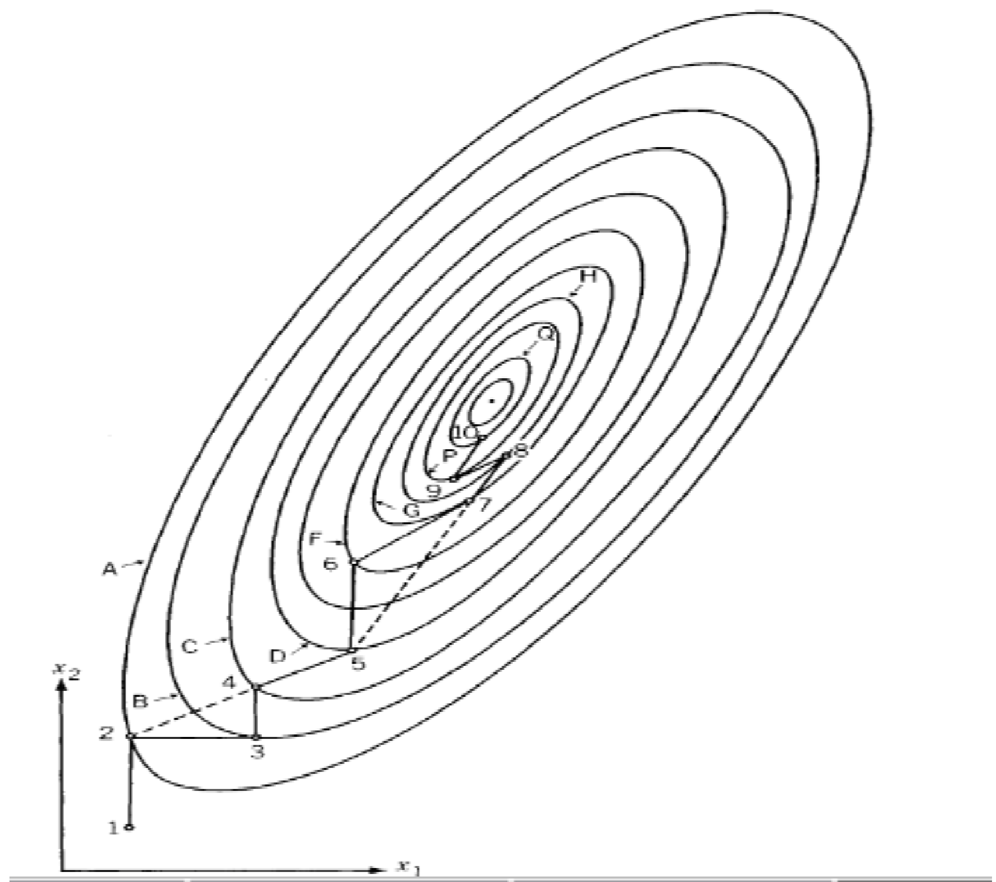
Solution is obtained in 3 iterations only!

Why?

Powell's Method

* This method is the most widely used order zero method!
* It can be shown that for a quadratic function, the minimum is reached in at most n iterations!
* This technique can be shown to be a conjugate direction method.

Powell's Method (Cont'd)



Powell's Method (Cont'd)

* At every step, we minimize sequentially along n directions with new pattern search directions at every step.

Iteration 1: $\underset{\sim}{S}_1, \underset{\sim}{S}_2, \cdots, \underset{\sim}{S}_{n-1}, \underset{\sim}{S}_n$ (pure coordinates)

Iteration 2: $\underset{\sim}{S}_2, \underset{\sim}{S}_3, \ldots, \underset{\sim}{S}_n, \underset{\sim}{S}_p^{(1)}$

Iteration 3: $\underset{\sim}{S}_3, \ldots, \underset{\sim}{S}_n, \underset{\sim}{S}_p^{(1)}, \underset{\sim}{S}_p^{(2)}$

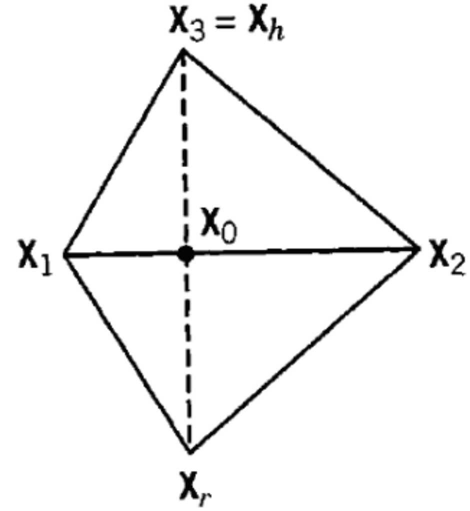
$\vdots$

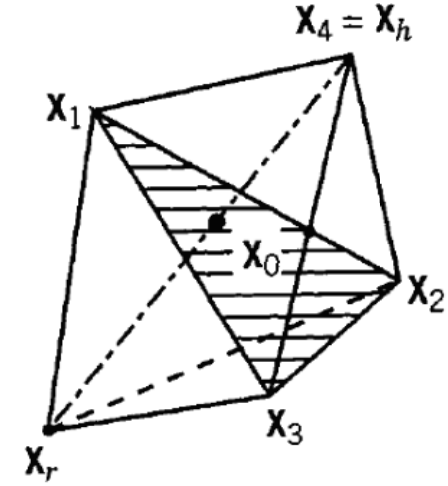
Iteration n: $\underset{\sim}{S}_p^{(1)}, \underset{\sim}{S}_p^{(2)}, \ldots, \underset{\sim}{S}_p^{(n-1)}, \underset{\sim}{S}_p^{(n)}$

The Simplex Method

* The simplex is a geometric figure formed of $(n+1)$ points in n dimensional space.
* In 2D, it is a triangle and in 3D it is a tetrahedron.
* The objective function values at these points indicate how the function is behaving.
* The simplex can be reflected, expanded, or contracted.

Reflection

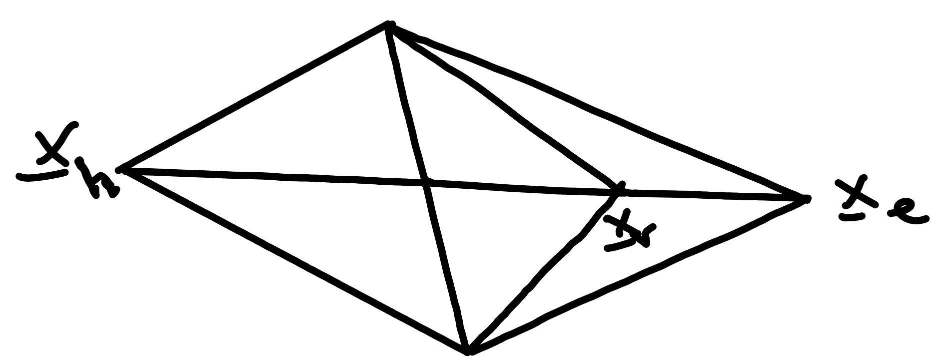




If $f(\underline{x}_h) = \max\limits_{1 \le i \le n+1} f(\underline{x}_i)$, we reflect the simplex to get the new point

$$\underline{x}_r = (1+\alpha)\,\underline{x}_0 - \alpha\,\underline{x}_h\,, \quad \underline{x}_0 = \frac{1}{n}\sum_{\substack{i=1\\ i\neq h}}^{n} \underline{x}_i$$

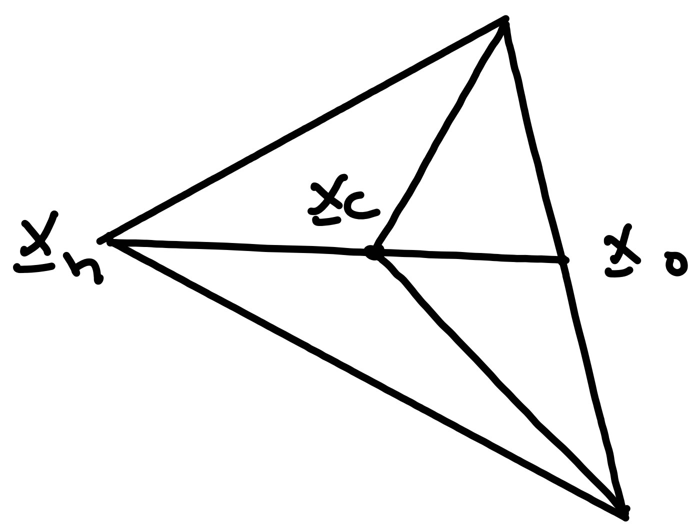
Expansion



If the reflected point $\underline{x}_r$ is better than all other simplex points $\Rightarrow f(\underline{x}_r) < f(\underline{x}_l)$, we can move further in the reflection direction

$$\Rightarrow \underline{x}_e = (1+\beta)\,\underline{x}_r - \beta\,\underline{x}_o \quad , \quad \beta > 0$$

Contraction



If the reflected point is bad, i.e. $f(\underline{x}_r) > f(\underline{x}_i)$ $\forall i$, we replace $\underline{x}_h$ by a contracted point

$$\underline{x}_c = \beta\, \underline{x}_h + (1-\beta)\, \underline{x}_o.$$

$\underline{x}_h$ is replaced by $\underline{x}_c$ if improvment is achieved.

Algorithm Flow

* At the ith iteration determine $f(\underline{x}_i), \forall i$. Determine $\underline{x}_h$, $\underline{x}_\ell$, and $\underline{x}_o$
* Reflection: determine $\underline{x}_r$. If $f(\underline{x}_r) < f(\underline{x}_h)$ set $\underline{x}_h = \underline{x}_r$. If $f(\underline{x}_r) < f(\underline{x}_\ell)$, expand the simplex. If $f(\underline{x}_e) < f(\underline{x}_r)$ set $\underline{x}_h = \underline{x}_e$
If $f(\underline{x}_r) > f(\underline{x}_h)$ contract the simplex. Repeat contraction until $f(\underline{x}_c) < f(\underline{x}_h)$ or until simplex is small enough.

MATLAB CODE

```
while(IterationCounter<10) %repeat
    [MaxValue  MaxIndex]=max(SimplexValues); %Get the maximum value and its index for the current simplex
    [MinValue MinIndex]=min(SimplexValues); %Get the minimum value and its index for the current simplex
    CenterLower=GetCenter(Simplex,MaxIndex); %Get the center of all points with lower function values
    %now we start by doing reflectionn
    ReflectedPoint=(1.0+ReflectionCoefficient)*CenterLower-ReflectionCoefficient*Simplex(:,MaxIndex);
    ReflectedValue=feval('getObjective',ReflectedPoint); %Get the reflected value
    if(ReflectedValue<MaxValue) %is there an improvment
        %now let see if there is a room for expansion
        if(ReflectedValue<MinValue) %is this point better than anything we had so far do expansion
           ExpandedPoint=(1+ExpansionCoefficient)*ReflectedPoint-ExpansionCoefficient*CenterLower; %expansion
           ExpandedValue=feval('getObjective',ExpandedPoint); %get the expanded value
           if(ExpandedValue<ReflectedValue)
              ReflectedPoint=ExpandedPoint;  %store the better values in the reflected point
              ReflectedValue=ExpandedValue;
           end
        end
        Simplex(:,MaxIndex)=ReflectedPoint;  %now we store the reflected point or the better expanded point
        SimplexValues(1,MaxIndex)=ReflectedValue;
    else  %no improvment.  we must contract
        ContractionPoint=ContractionCoefficient*Simplex(:,MaxIndex)+(1-ContractionCoefficient)*CenterLower;
        ContractionValue=feval('getObjective',ContractionPoint);
        %now we update the highest value
        Simplex(:,MaxIndex)=ContractionPoint;
        SimplexValues(1,MaxIndex)=ContractionValue;
    end
    IterationCounter=IterationCounter+1; %increase the iteration counter
end
```

Example

Find the minimum of the function $f(\underset{\sim}{x}) = (x_1-1)^2 + (x_2-5)^2 + (x_3-4)^2$ using the Simplex Method. Start with the initial simplex

$$\underset{\sim}{S} = \begin{bmatrix} 0 & 0 & 0.6 & 0 \\ 0 & 0.4 & 0 & 0 \\ 0.3 & 0 & 0 & 0 \end{bmatrix}$$

Code Output

```
MinValue1 = 38.1600
Min1 =[  0
       0.4000
          0 ]

MinValue2 = 36.0903
Min2 =[ 0.5000
         0.3333
         0.2500]

MinValue3 = 34.0060
Min3 =   [-0.4833
            0.6111
            0.4583]

after 20 iterations

MinValue4 =0.0975
Min15 =  [0.9648
           5.2921
           4.1044]

exact solution [1.0  5.0  4.0]'
```