PROJECT 1.3

Due date November 5, 2012

Description of the project

1. **You must implement the following MatLab notebook in sage and ipython notebook interfaces (You should download and install ipython notebook in your computer)**
2. **Before each cell in sage notebook you should insert the comments that have been provided in MatLab notebook including pictures that are not part of the MatLab output.**
3. **Try to find a way to insert text above each cell in ipython notebook and let me know.**
4. **Notice that the Matlab codes in this notebook are executable directly from the word doc!**

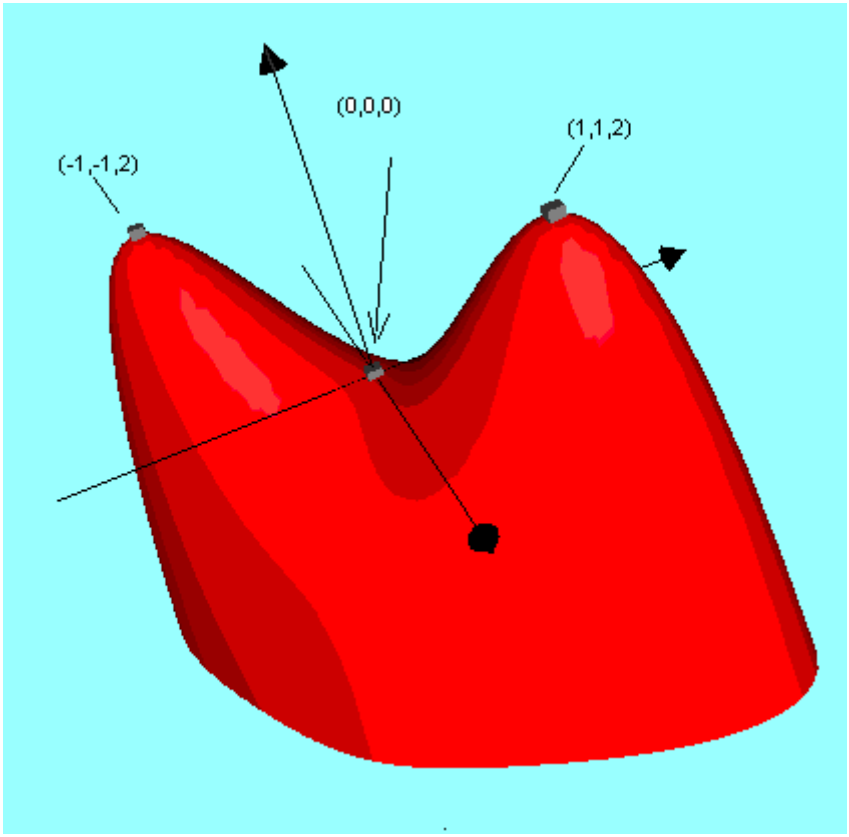
Non-linear optimization

Q: What is non-linear optimization?
A: That means to find a minimum/maximum of a non-linear function.

This optimization can be constrained (if there is some limit to the values of variables) and unconstrained.

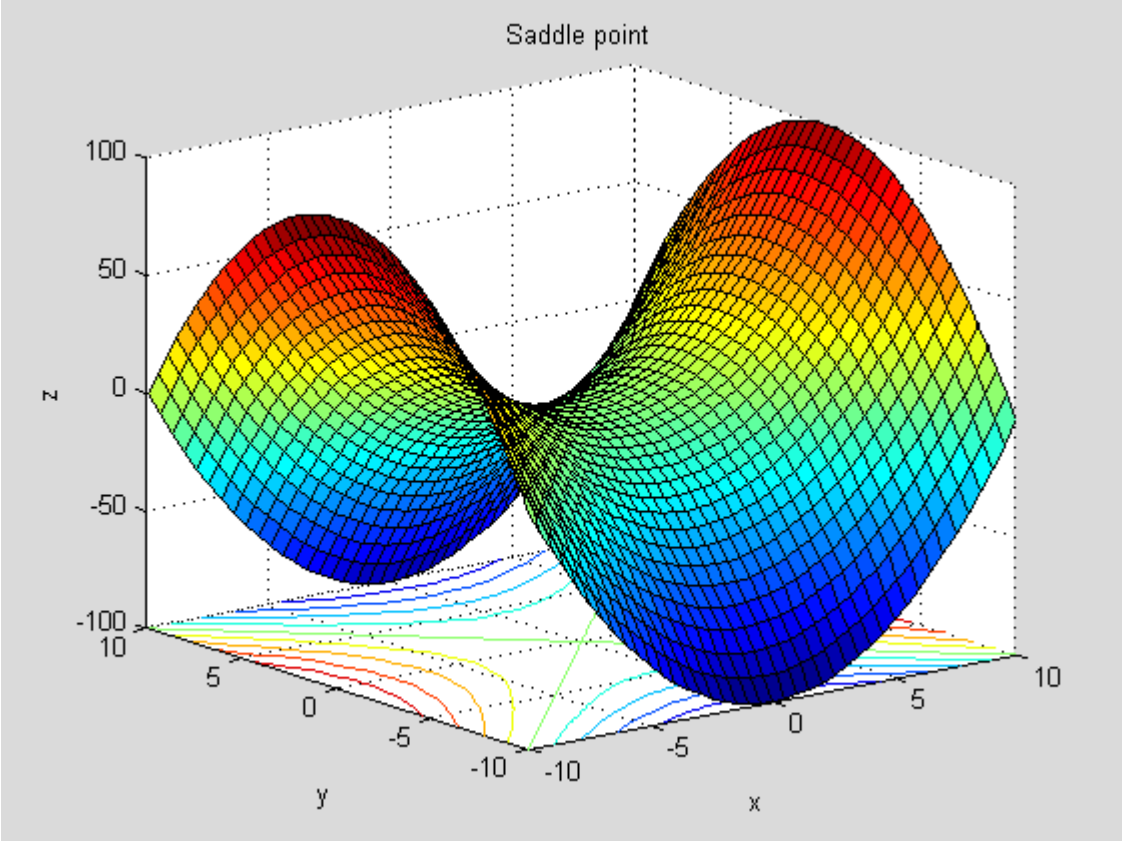
Here, we will consider un-constrained optimization.

A 3-Dimensional graph of function f shows that f has two local maxima at (-1, -1, 2) and (1, 1, 2) and a saddle point at (0, 0, 0).



A differentiable function $\phi(\xi; \psi)$ has a saddle point at a critical point $(\alpha; \beta)$ if in every open disk centered at $(\alpha; \beta)$ there are domain points $(\xi; \psi)$ where $\phi(\xi; \psi) > \phi(\alpha; \beta)$ and domain points $(\xi; \psi)$ where $\phi(\xi; \psi) < \phi(\alpha; \beta)$. The corresponding point $(\alpha; \beta; \phi(\alpha; \beta))$ on the surface $\zeta = \phi(\xi; \psi)$ is called a saddle point of the surface.

```
% Create a grid of x and y data
y = -10:0.5:10;
x = -10:0.5:10;
[X, Y] = meshgrid(x, y);

% Create the function values for Z = f(X,Y)
Z=(X.^2-Y.^2);
% Create a surface contour plor using the surfc function
figure;
surfc(X, Y, Z);

% Adjust the view angle
view(-38, 18);

% Add title and axis labels
title('Saddle point');
xlabel('x');
ylabel('y');
zlabel('z');
```



Non-linear unconstrained optimization problem is:

Maximize (or minimize) $f(x_1,x_2\ldots,x_n)$ where f is non-linear function.

Consider the problem:

$$\max f(x,y)=\frac{x-y}{x^2+y^2+1}$$

Q: How would we solve this problem analytically?
A: We would need calculus of multiple variables.

Remember from simple calculus of the necessary condition for extremal values

In case of single value function the **extremal points** of a function f(x) can be found among these points where the first derivative:

$f'(x)=\frac{df}{dx}$ is equal to zero.

In case of multivalued functions the partial derivatives of f are zero at the extremal points!

What will we do here? We will compute so-called "partial" derivatives.

Q: What is partial derivative?
A: If a function depends on multiple variables, we will observe only one variable, and differetiate by it. The other variables will be considered as constants.

Example 1

$$f(x,y)=x^2+y$$

partial derivative with respect to x is: $\frac{df(x,y)}{dx}=\frac{d}{dx}x^2+\frac{d}{dx}y=2x+0=2x$ (y is considered constant)

partial derivative with respect to y is: $\frac{df(x,y)}{dy}=\frac{d}{dy}x^2+\frac{d}{dy}y=0+1=1$ (x is considered constant)

MatLab Code

```
syms x y dx dy z gradient   %define variables as symbolic
z=x^2+y      %define function to be maximized
dx=diff(z,x)                %partial derivatives
dy=diff(z,y)
```

```
z =x^2 + y
dx =2*x
dy =1
```

Example 2:

$$f(x,y)=\sin(x)+e^y$$

partial derivative with respect to x is: $\frac{df(x,y)}{dx}=\frac{d}{dx}\sin(x)+\frac{d}{dx}e^y=\cos x+0=\cos x$ (y is considered constant)

partial derivative with respect to y is: $\frac{df(x,y)}{dy}=\frac{d}{dy}\sin(x)+\frac{d}{dy}e^y=0+e^y=e^y$ (x is considered constant)

MatLab Code

```
syms x y dx dy z gradient   %define variables as symbolic
z=sin(x)+exp(y)       %define function to be maximized
dx=diff(z,x)                %partial derivatives
dy=diff(z,y)
```

```
z =exp(y) + sin(x)
dx =cos(x)
```

```
dy =exp(y)
```

Example 3

$f(x,y)=e^x \log(y)$

partial derivative with respect to x is: $\frac{df(x,y)}{dx}=\frac{d}{dx}e^x \log(y)=\log(y)\frac{d}{dx}e^x=\log(y)e^x$ (y and hence log(y) is considered constant)

partial derivative with respect to y is: $\frac{df(x,y)}{dy}=\frac{d}{dy}e^x \log(y)=e^x \frac{d}{dy}\log(y)=e^x \frac{1}{y}=\frac{e^x}{y}$ (x and hence e^x is considered constant)

```
syms x y dx dy z gradient   %define variables as symbolic
z=exp(x)*log(y)       %define function to be maximized
dx=diff(z,x)                %partial derivatives
dy=diff(z,y)

z =exp(x)*log(y)
dx =exp(x)*log(y)
dy =exp(x)/y
```

Example 4:

$$f(x,y)=\frac{x-y}{x^2+y^2+1}$$

```
syms x y dx dy z gradient   %define variables as symbolic
z=(x-y)/(x^2+y^2+1)       %define function to be maximized
dx=diff(z,x)                %partial derivatives
dy=diff(z,y)

z =(x - y)/(x^2 + y^2 + 1)
dx =1/(x^2 + y^2 + 1) - (2*x*(x - y))/(x^2 + y^2 + 1)^2
dy =- 1/(x^2 + y^2 + 1) - (2*y*(x - y))/(x^2 + y^2 + 1)^2
```

Determine points where the partial derivatives of function $f(x,y)=\frac{x-y}{x^2+y^2+1}$ are zero.

```
syms x y dx dy z u v gradient   %define variables as symbolic
z=(x-y)/(x^2+y^2+1)       %define function to be maximized
```

```
dx=diff(z,x)                    %partial derivatives
dy=diff(z,y)
S=solve('1/(x^2 + y^2 + 1) - (2*x*(x - y))/(x^2 + y^2 + 1)^2=0',...
'- 1/(x^2 + y^2 + 1) - (2*y*(x - y))/(x^2 + y^2 + 1)^2=0')
S.x
S.y
f=@(x,y)(x-y)/(x^2+y^2+1)
f(S.x(1,1),S.y(1,1))
f(S.x(2,1),S.y(2,1))

z =
(x - y)/(x^2 + y^2 + 1)
dx =
1/(x^2 + y^2 + 1) - (2*x*(x - y))/(x^2 + y^2 + 1)^2
dy =
- 1/(x^2 + y^2 + 1) - (2*y*(x - y))/(x^2 + y^2 + 1)^2
S =
    x: [4x1 sym]
    y: [4x1 sym]
ans =
    0.70710678118654752440084436210485
   -0.70710678118654752440084436210485
 -0.70710678118654752440084436210485*i
  0.70710678118654752440084436210485*i
ans =
   -0.70710678118654752440084436210485
    0.70710678118654752440084436210485
 -0.70710678118654752440084436210485*i
  0.70710678118654752440084436210485*i
f =
    @(x,y)(x-y)/(x^2+y^2+1)
ans =
0.70710678118654752440084436210485
ans =
-0.70710678118654752440084436210485
```

Sage cells

```
var('x,y')
f(x,y)=(x-y)/(x**2+y**2+1)
f
dx=derivative(f,x)
dx
```

```
(x, y) |--> -2*(x - y)*x/(x^2 + y^2 + 1)^2 + 1/(x^2 + y^2
+ 1)
```

```
dy=derivative(f,y)
dy
```

```
(x, y) |--> -2*(x - y)*y/(x^2 + y^2 + 1)^2 - 1/(x^2 + y^2
+ 1)
```

```
solve([dx==0,dy==0],x,y)
```

```
[[x == -1/2*sqrt(2), y == 1/2*sqrt(2)], [x ==
1/2*sqrt(2), y ==
-1/2*sqrt(2)], [x == -1/2*I*sqrt(2), y == -
1/2*I*sqrt(2)], [x ==
1/2*I*sqrt(2), y == 1/2*I*sqrt(2)]]
```

```
x == -1/2*sqrt(2)
y == 1/2*sqrt(2)
```

```
x == 1/2*sqrt(2)
y == -1/2*sqrt(2)
```

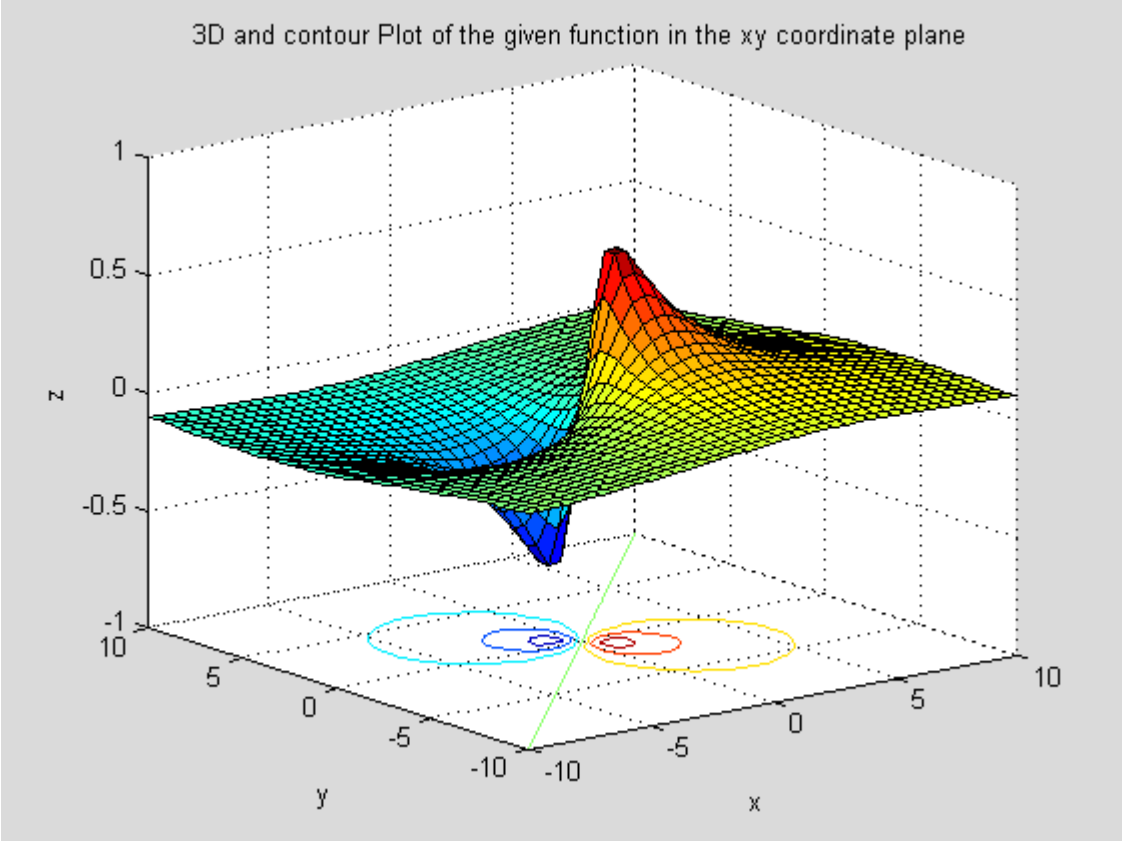
```
(x - y)/(x^2 + y^2 + 1) == -1/2*sqrt(2)
```

```
(x - y)/(x^2 + y^2 + 1) == 1/2*sqrt(2)
```

Q: Where the max and min of the function occurs?
A: Check the analytic and numerical roots of the above equations and the graph below

```
% Create a grid of x and y data
y = -10:0.5:10;
x = -10:0.5:10;
[X, Y] = meshgrid(x, y);

% Create the function values for Z = f(X,Y)
Z=(X-Y);
Z=Z./(X.^2+Y.^2+1);
% Create a surface contour plor using the surfc function
figure;
surfc(X, Y, Z);

% Adjust the view angle
view(-38, 18);

% Add title and axis labels
title('3D and contour Plot of the given function in the xy coordinate
plane');
```

```
xlabel('x');
ylabel('y');
zlabel('z');
```



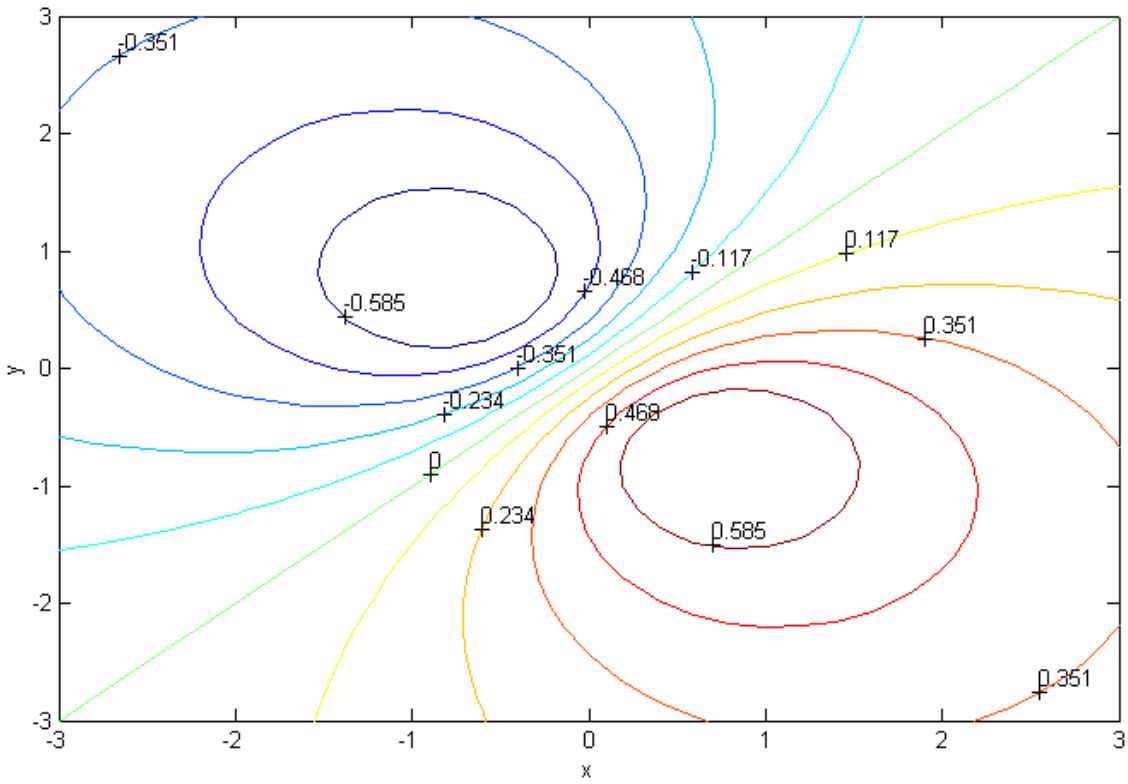
How we can find extremum (minimum, maximum…) *numerically*?

Idea: Apply "hill climbing" strategy.

Hill climbing

Assume that we want to maximize objective function. The idea is to set initial point and to find the direction in which the objective function grows the fastest. Then, to move in that direction for some (small) step and to repeat this procedure until the value of the function does not change significantly.

Contour plot of our function
(each contour corresponds to a constant value of the function)



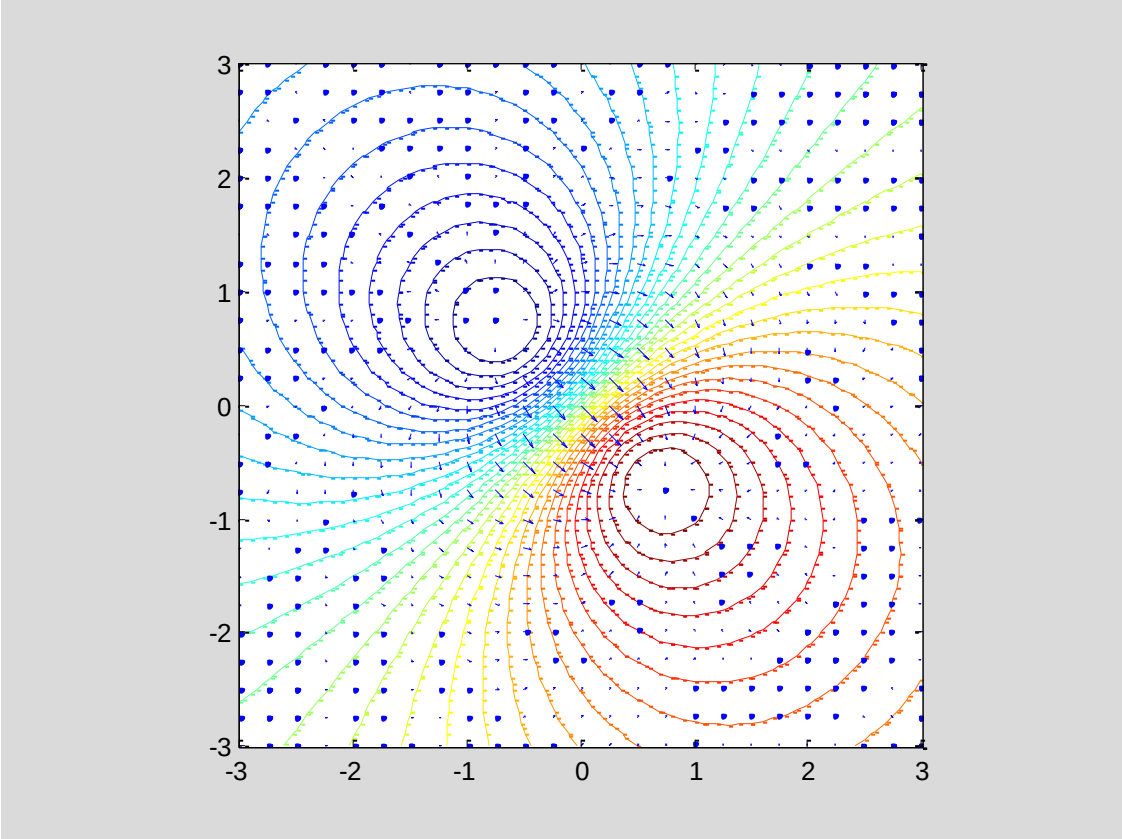
We can observe: No two contour lines are intersecting and the denser contour lines, the faster function grows.

Next, we observe the behavior of the gradient i.e of the vector of the partial derivatives of f.

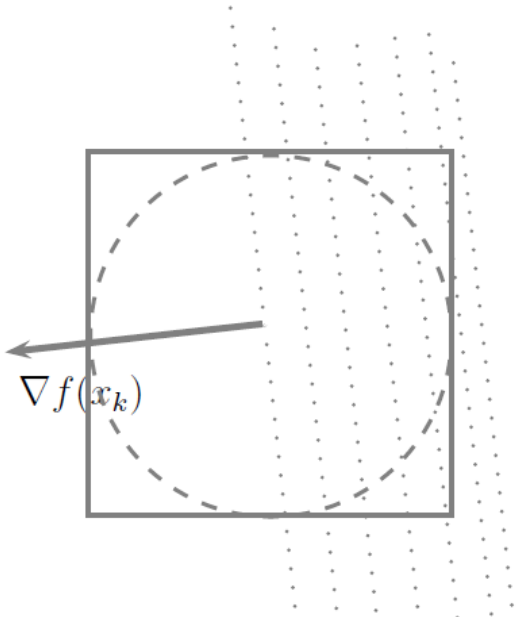
```
%%Visualizing Gradients of Functions of Two Variables
%The gradient of a function of several variables is the vector-valued
%function whose components are the partial derivatives of the function.
syms x y z
f=(x-y)/(x^2+y^2+1)
gradf=jacobian(f,[x,y])
%Plot gradients
[xx, yy] = meshgrid(-3:.1:3,-3:.1:3);
ffun = @(x,y) eval(vectorize(f));
fxfun = @(x,y) eval(vectorize(gradf(1)));
fyfun = @(x,y) eval(vectorize(gradf(2)));
figure(1);
contour(xx, yy, ffun(xx,yy), 30)
hold on
[xx, yy] = meshgrid(-3:.25:3,-3:.25:3);
quiver(xx, yy, fxfun(xx,yy), fyfun(xx,yy), 0.6)
axis equal tight, hold off
```

```
f =(x - y)/(x^2 + y^2 + 1)
```

```
gradf =[ 1/(x^2 + y^2 + 1) - (2*x*(x - y))/(x^2 + y^2 + 1)^2, - 1/(x^2 +
y^2 + 1) - (2*y*(x - y))/(x^2 + y^2 + 1)^2]
```



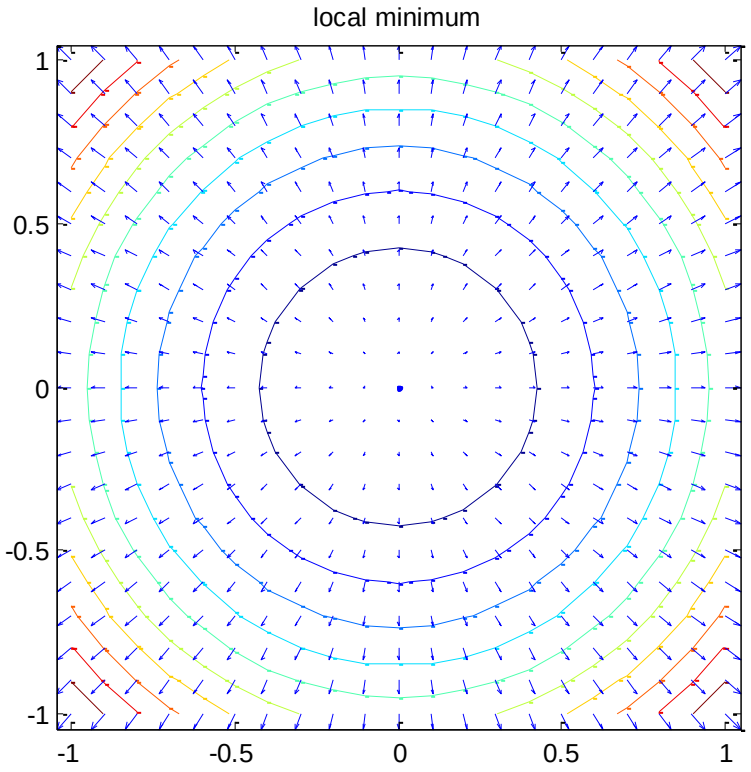
In each point, we can observe the direction in which the function grows fastest. The directions of vectors correspond to the fastest ascent and the intensities of the vector correspond to the slope. In the above contour plot, we can see that the vectors of maximal ascent are orthogonal to contour lines and that the function grows fastest around x=0, y=0.

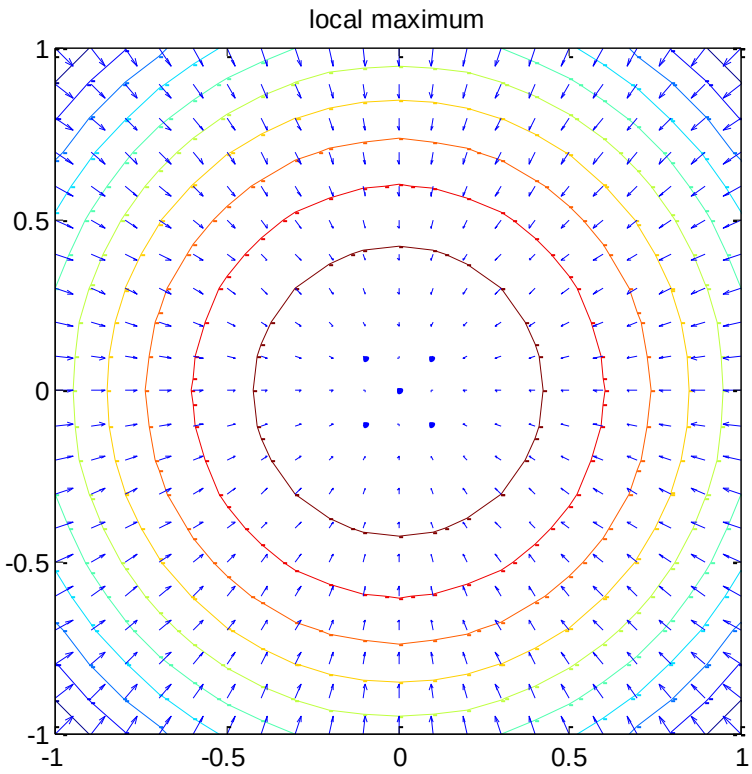


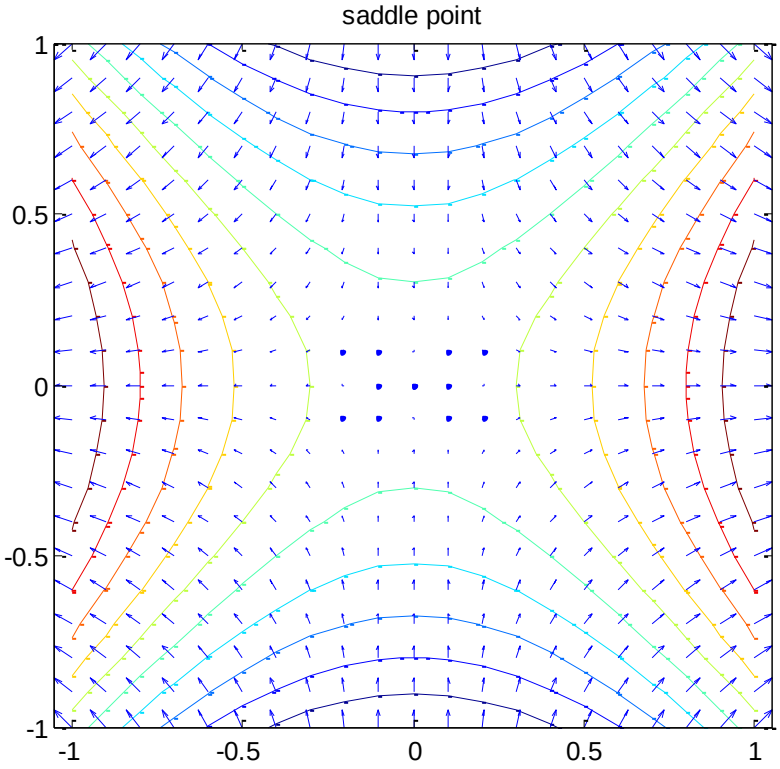
Direction of the gradient vertical to countour lines

```
%%Behavior Near Critical Points
%A plot such as this one can be interpreted to give information
regarding
%critical points of the function. Critical points are points where the
%gradient vector vanishes. A critical point is called on-degenerate if
%behavior of the function near the critical point is controlled by the
%second derivatives (so that the 'second derivative test' applies). For
%functions of two variables, there are three kinds of non-degenerate
critical points.
%You can recognize them from the following three kinds of pictures:
%local minimum
figure(1);
f1 = x^2 + y^2; gradf1 = jacobian(f1,[x,y]);
f1fun = @(x,y) eval(vectorize(f1));
f1xfun = @(x,y) eval(vectorize(gradf1(1)));
f1yfun = @(x,y) eval(vectorize(gradf1(2)));
[xx, yy] = meshgrid(-1:.1:1,-1:.1:1);
contour(xx, yy, f1fun(xx, yy), 10)
hold on
quiver(xx, yy, f1xfun(xx, yy), f1yfun(xx, yy), 0.5)
title('local minimum'), axis equal tight, hold off
set(gca, 'YTick', -1:.5:1)
%local maximum
figure(2)
contour(xx, yy, -f1fun(xx, yy), 10)
hold on
quiver(xx, yy, -f1xfun(xx, yy), -f1yfun(xx, yy), 0.5)
title('local maximum'), axis equal tight, hold off
set(gca, 'YTick', -1:.5:1)
%saddle points
figure(3)
f2 = x^2 - y^2; gradf2 = jacobian(f2,[x,y]);
f2fun = @(x,y) eval(vectorize(f2));
```

```
f2xfun = @(x,y) eval(vectorize(gradf2(1)));
f2yfun = @(x,y) eval(vectorize(gradf2(2)));
contour(xx, yy, f2fun(xx, yy), 10)
hold on
quiver(xx, yy, f2xfun(xx, yy), f2yfun(xx, yy), 0.5)
title('saddle point'), axis equal tight, hold off
set(gca, 'YTick', -1:.5:1)
```




local maximum
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1


saddle point
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

If we follow the rule that in each point we travel in the direction of maximal ascent (or steepest descent for finding the minimum), we will have a "trajectory" towards the solution as the following method, called gradient, and its MatLab implementation indicates. This rule for steepest descent is described by the following pseudocall

Gradient Algorithm or First Derivative Methods

1. choose x_0 in the domain of f and tolerance $\varepsilon>0$
2. set $k = 0$
3. loop compute $g_k = \nabla f(x_k)$

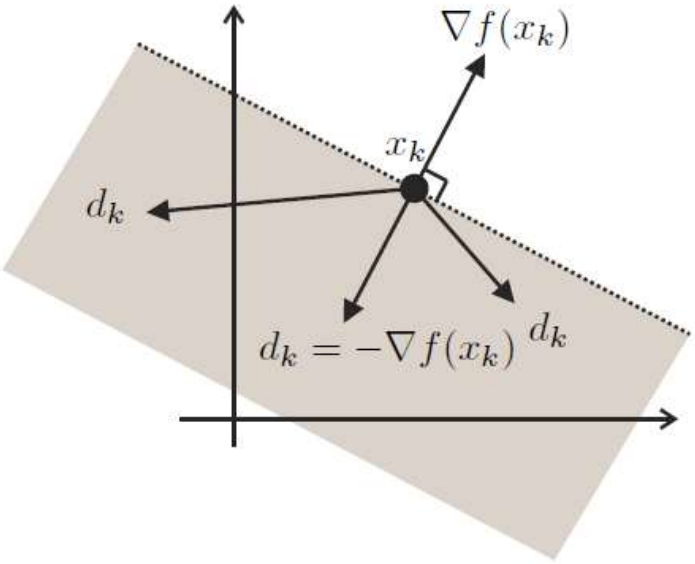
   if $\|g_k\| < \varepsilon$ stop

   compute ascent (descent) direction $d_k = \nabla f(x_k)$ $(-\nabla f(x_k))$

   compute step size μ_κ

   update $x_{k+1} = x_k + \mu_\kappa d_k$

   $k \leftarrow k+1$ and return to loop



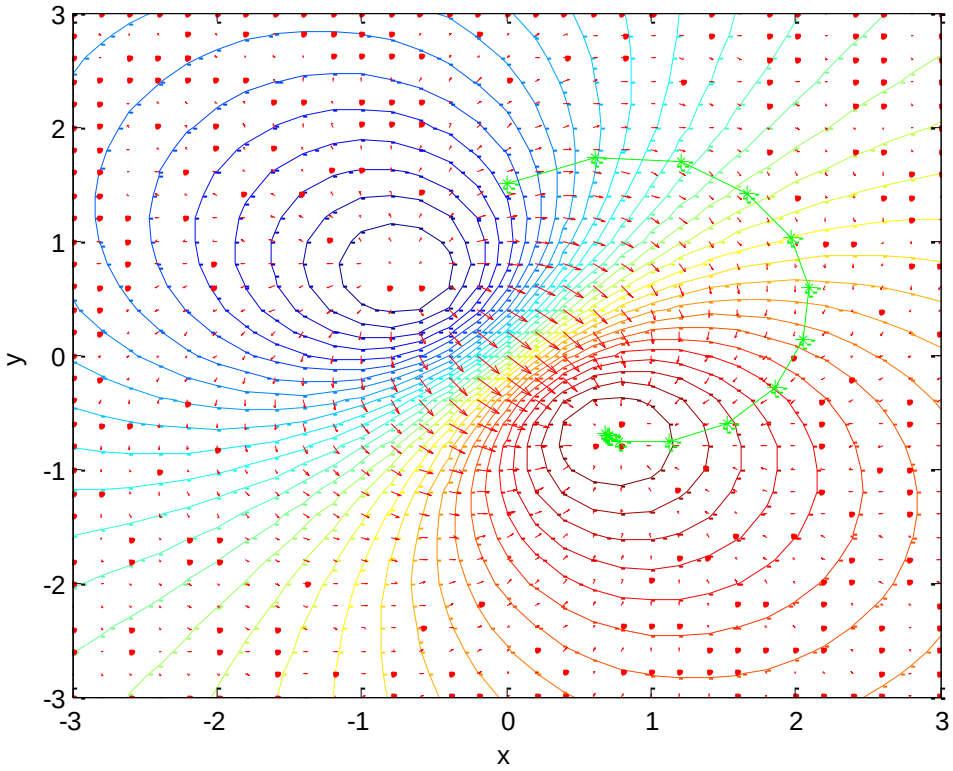
In the following implementation of gradient method we keep μ_k fix.

```
%   Gradient method
syms x y dx dy z gradient   %define variables as symbolic
f=(x-y)/(x^2+y^2+1) % function to find maximun
gradf=jacobian(f,[x,y])% gradient vector of f

% search space
a=-3;
b=3;
c=-3;
d=3;
%Plot gradients
[xx, yy] = meshgrid(a:.1:b,c:.1:d);
```

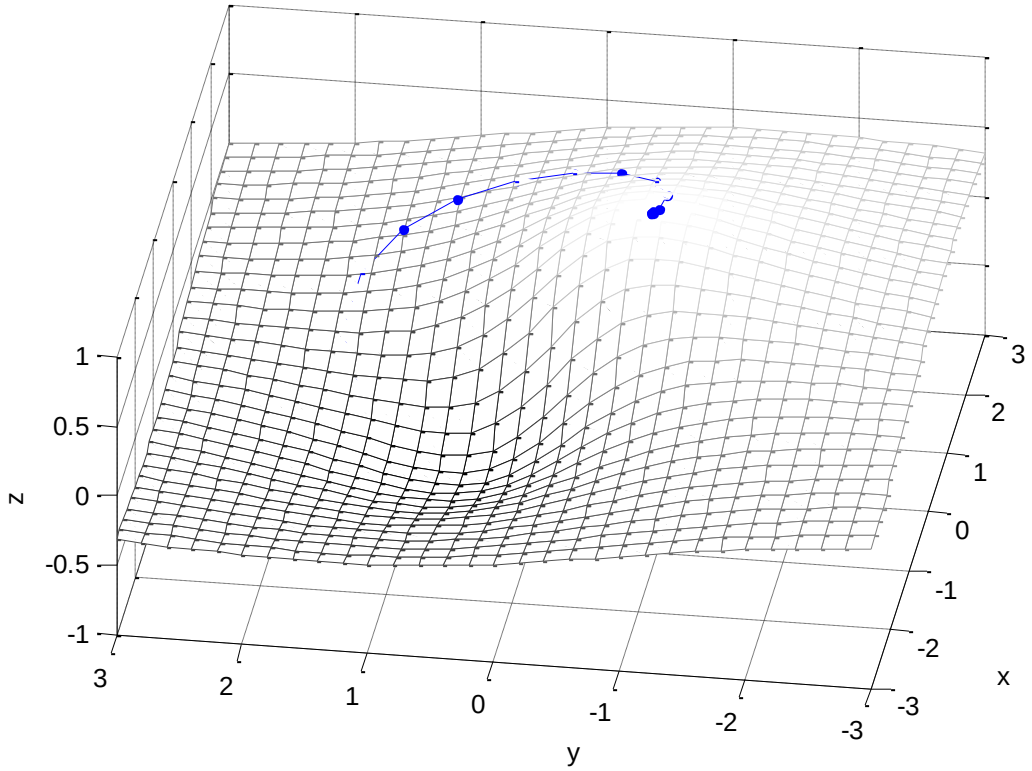
```
ffun = @(x,y) eval(vectorize(f));
fxfun = @(x,y) eval(vectorize(gradf(1)));
fyfun = @(x,y) eval(vectorize(gradf(2)));

% application of gradient method
N_iter=15;                  %number of iterations
mu=2;                       %algorithms coefficient

x0=0;                       %starting point
y0=1.5;
%Algorithm
solution=[x0 y0];
clear sol_arh
sol_arh(1,:)=solution;      %we archive solutions...
for i=1:N_iter
    x=solution(1);
    y=solution(2);
    solution=solution+mu*eval(gradf);        %New value of decision
variables in the direction of gradient
    grad_arh(i+1,:)=eval(gradf);             %we archive current values
of gradient
    norm_arh(i+1)=norm( grad_arh(i+1,:));       %and its norm
    sol_arh(i+1,:)=solution;
end

figure(1)
%Plot contours of the function and gradient vectors on 2D pl
[x,y] = meshgrid(a:.2:b, c:.2:d); %Generate function values on
rectangular grid for
z=(x-y)./(x.^2+y.^2+1); %NOTE: THIS is numer, not a symbolic value

contour(x, y, ffun(x, y), 30)  %Plot contours of the function on 2D pl
hold on
quiver(x, y, fxfun(x, y), fyfun(x, y), 'r')%plot gradient vectors on
contour plot
% Add title and axis labels
title('Contour plot the obective function f, its gradient, and the
sequence of appr solution');
xlabel('x');
ylabel('y');
zlabel('z');
xx=sol_arh(:,1);yy=sol_arh(:,2);zz=(xx-
yy)./(xx.^2+yy.^2+1);plot(xx,yy,'*-g') %compute function values for
solutions found in the

%iterations and plot them on 2d plot
figure(2)
mesh(x,y,z)          %plots function in 3D
colormap gray
hold on;plot3(xx,yy,zz,'.-') %plot values for solutions from the
iterations on 3D plot
% Add title and axis labels
title('Plot the obective function f in 3D and the sequence of appr
solution');
xlabel('x');
ylabel('y');
zlabel('z');
```

```
figure(3)
error=log(abs(zz-sqrt(0.5))); %this is log of absolute error (absolute
difference between achieved value of objective
                                %function in the iterations and the
optimal value which is (we should know that!) sqrt(0.5)


plot(error(1:length(error)-1),error(2:length(error)))   %The slope of
this plot is the order of convergence!!!
% Add title and axis labels
title('Plots the log of error for the appr solution knowing that true
solution is sqrt(0.5)');
xlabel('x');
ylabel('y');
zlabel('z');


f =
(x - y)/(x^2 + y^2 + 1)
gradf =
[ 1/(x^2 + y^2 + 1) - (2*x*(x - y))/(x^2 + y^2 + 1)^2, - 1/(x^2 + y^2 +
1) - (2*y*(x - y))/(x^2 + y^2 + 1)^2]
```

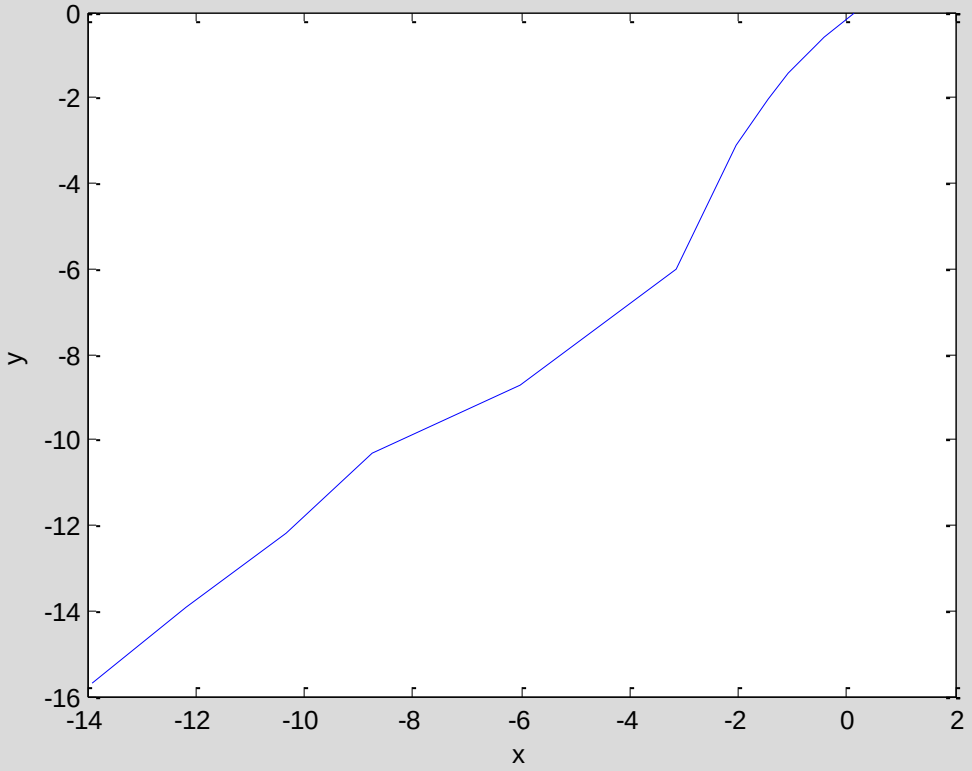
Contour plot the obective function f, its gradient, and the sequence of appr solution



Plot the obective function f in 3D and the sequence of appr solution

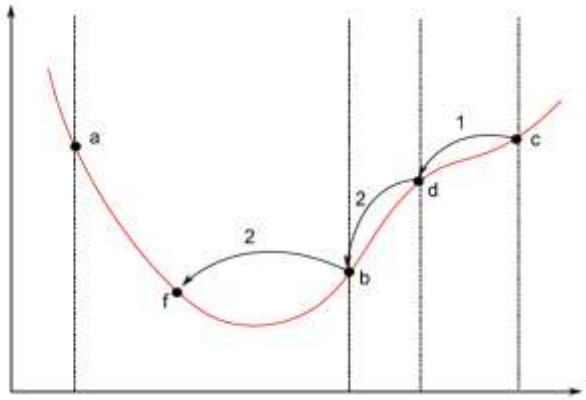


Plots the log of error for the appr solution knowing that true solution is sqrt(0.5)

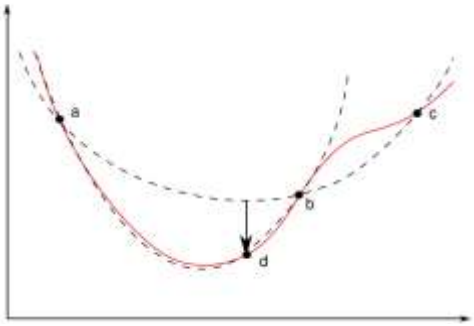


In order to find the optimal value of μ_k we apply one dimensional optimization methods. The most common ones are the golden search and quadratic inteprolation. Their basic ideas are described in the following figures. The algorithm for golden search is given in Houstis ebook.

Section Search



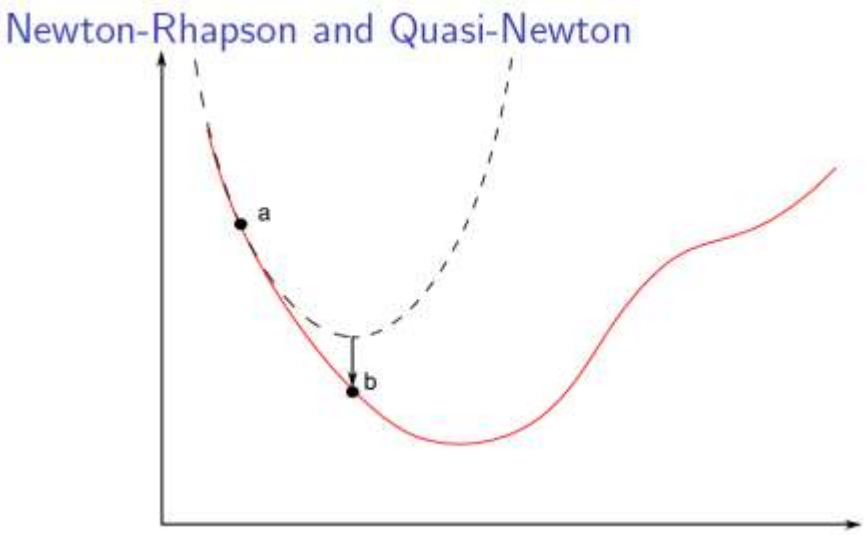
Quadratic Interpolation



1. **Second derivative:** Newton's method.

Newton: use $f'(x)$ and $f''(x)$ to construct parabola

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$



Example implementation of Newton's method for single function with two variables. **Notice that it requires as input the second derivative (Hessian).**

```
syms x  y
f=(x-2)^4+(x-2*y)^2
df=jacobian(f,[x,y])
ddf=jacobian(df,[x,y])
evalf=@(x,y) eval(f);
evaldf=@(x,y) eval(df);
evalddf=@(x,y)eval(ddf);
% this is the pure form of newton's method
t=[0;0];
max_iter=20;
tol=0.0001
k = 0;
delf = evaldf(t(1),t(2));
alpha = 1;
while norm(delf)>tol & k <= max_iter
    H = evalddf(t(1),t(2));
    d = -H\delf';
    k = k+1;
    t = t + alpha*d;
    delf = evaldf(t(1),t(2));
end
minimumf=t
gradientf=delf
```

```
f =

(x - 2*y)^2 + (x - 2)^4

df =

[ 2*x - 4*y + 4*(x - 2)^3, 8*y - 4*x]

ddf =
```

```
        [ 12*(x - 2)^2 + 2, -4]
        [               -4,  8]
        tol =
          1.0000e-004
        minimumf =
            1.9769
            0.9884
        gradientf =
          1.0e-004 *
   -0.4945         0
```

2. **First derivative:** quasi-Newton's methods.

The ***derivative-based methods*** are not really new - they solve a nonlinear equation *f(x)=0*. The most important one is the Newton's method. The Newton's method has very attractive convergence properties but can be very expensive due to computations of the second derivative. Therefore, there is a whole family of "first derivative methods", some trying to approximate the second derivative (quasi-Newton) and others, less ambitious, looking for the zero without any information about the second derivative at all. It is a natural idea to try to combine those two strategies in a kind of hybrid methods.

Quasi-Newton: approximate $f''(x_n)$ with $\frac{f'(x_n)-f'(x_{n-1})}{x_n-x_{n-1}}$

- Given approximate solution values a, b, c, with function values f_a, f_b, f_c, next approximate solution found by fitting quadratic polynomial to a, b, c as function of f_a, f_b, f_c, then evaluating polynomial at 0
- Based on nontrivial derivation using Lagrange interpolation, we compute

$$u = f_b/f_c, \quad v = f_b/f_a, \quad w = f_a/f_c$$
$$p = v(w(u-w)(c-b) - (1-u)(b-a))$$
$$q = (w-1)(u-1)(v-1)$$

  then new approximate solution is $b + p/q$
- Convergence rate is normally $r \approx 1.839$

```
myfun=inline('c*x(1)^2 + 2*x(1)*x(2) + x(2)^2','x','c')
c = 3;  % define parameter first
x = fminunc(@(x)myfun(x,c),[1;1])
```

```
myfun =
     Inline function:
     myfun(x,c) = c*x(1)^2 + 2*x(1)*x(2) + x(2)^2
Warning: Gradient must be provided for trust-region algorithm;
```

```
  using line-search algorithm instead.
> In fminunc at 347

Local minimum found.

Optimization completed because the size of the gradient is less than
the default value of the function tolerance.



x =
  1.0e-006 *
    0.2541
   -0.2029
```

In case the gradient of the function is known then the method is much faster. The following code can NOT be executed from the note book because the functions have to be defined inline. Inline functions jn MatLab do not produce multivalue output.

```
function [f,gradf]=objfun(x)
f=(x(1)^2+x(2)^2)^2-x(1)^2-x(2)+x(3)^2;
gradf=[4*x(1)*(x(1)^2+x(2)^2)-2*x(1);4*x(2)*(x(1)^2+x(2)^2)-1;2*x(3)];
options=optimset('GradObj','on');
x0=[1;1;1];[x,fval]=fminunc('objfun',x0,options)
```

Output

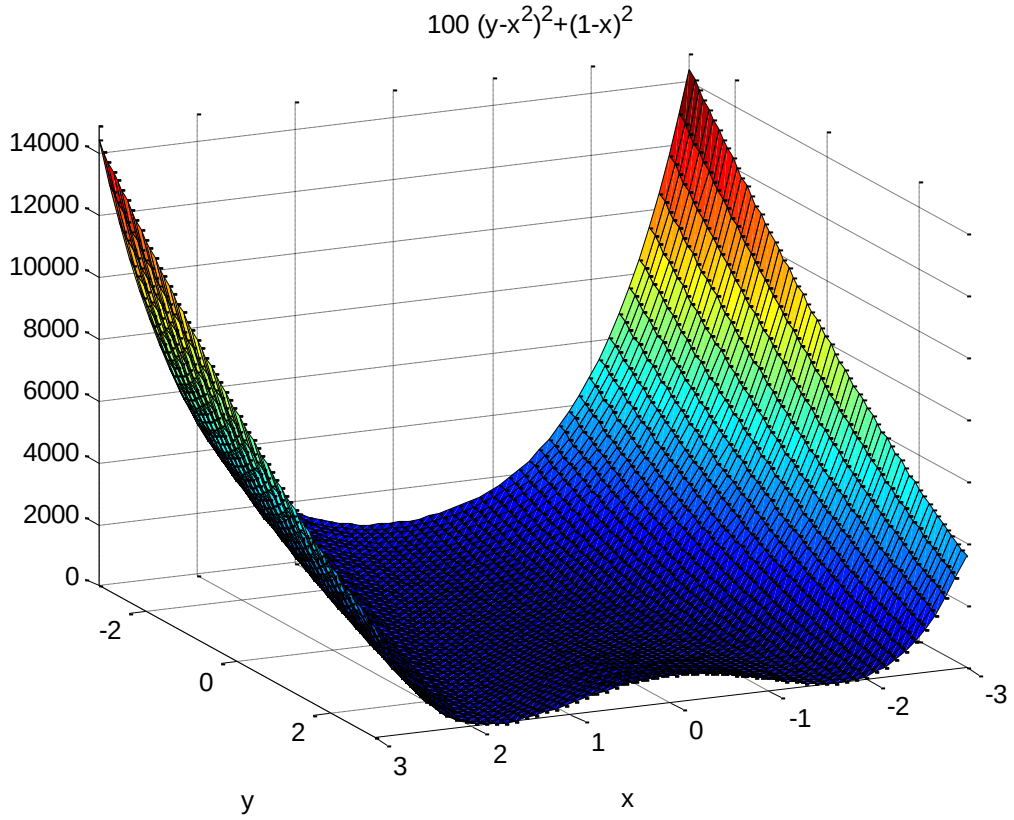
```
Local minimum possible.

fminunc stopped because the final change in function value relative to
its initial value is less than the default value of the function
tolerance.

<stopping criteria details>


x =

    0.5005
    0.4998
    0.0000


fval =

   -0.5000
```

3. **Non-derivative methods:** golden section search, parabolic interpolation.

   MatLab function fminsearch implements an non derivative method. It's usage is demostrated in the following code

```
fbanana = @(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2;
banana = @(x,y)100*(y-x.^2).^2+(1-x).^2;
ezsurf(banana, [-3 3]); view(155, 20); hold on;
xmin = fminsearch(fbanana, [-1.2 1])
hold off
```

```
xmin =
    1.0000    1.0000
```



Multiobjective unconstrained optimization

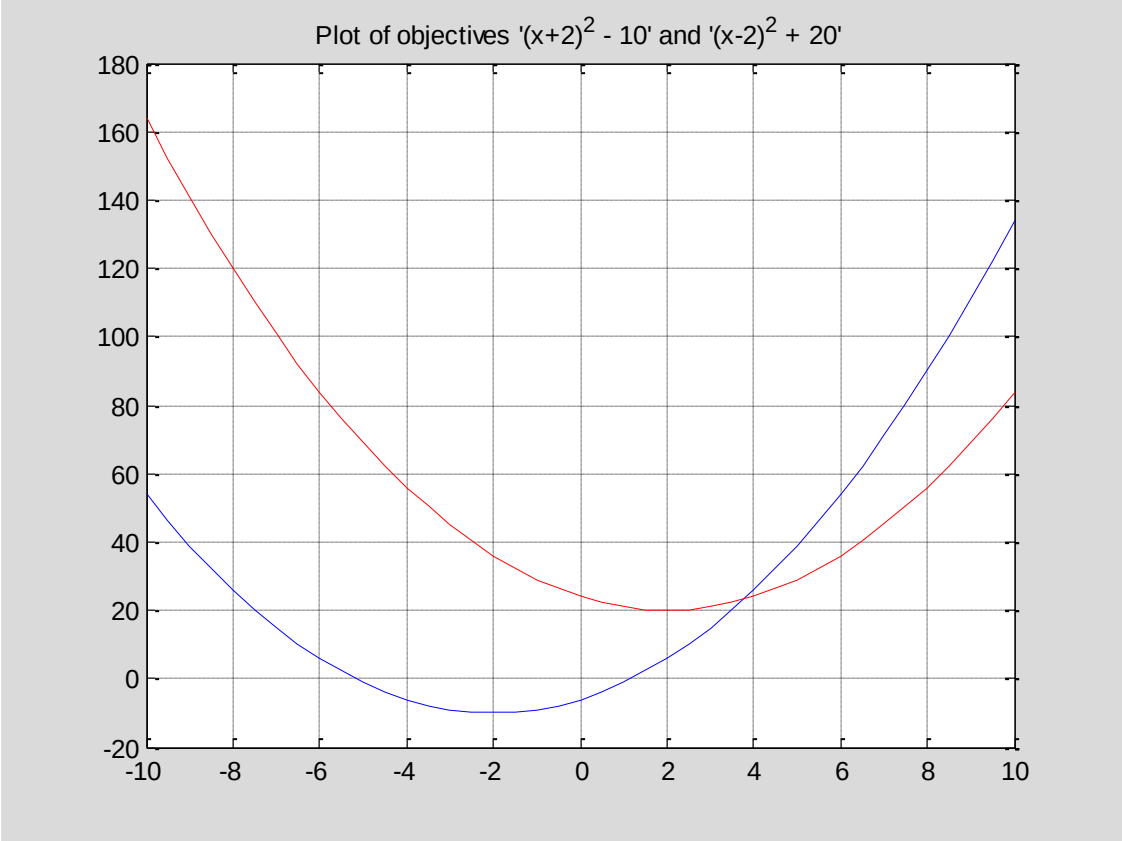
Genetic Algorithm

- Can find global optimum (but I do not know whether this has been formally proven)
- Begin with random "population" of points, $\{x_n^0\}$, then
  1. Compute "fitness" of each point $\propto -f(x_n^i)$
  2. Select more fit points as "parents"
  3. Produce children by mutation $x_n^{i+1} = x_n^i + \epsilon$, crossover $x_n^{i+1} = \lambda x_n^i + (1-\lambda)x_m^i$, and elites, $x_n^{i+1} = x_n^i$
  4. Repeat until have not improved function for many iterations
- In Matlab, `[x fval] = ga(@f,nvars,options)`
  - Requires Genetic Algorithm and Direct Search Toolbox
  - Many variations and options
    - ★ Options can affect whether converge to local or global optimum
    - ★ Read the documentation and/or use `optimtool`

`gamultiobj` can be used to solve multiobjective optimization problem in several variables. Here we want to minimize two objectives, each having one decision variable.

```
min F(x) = [objective1(x); objective2(x)]

 x

where, objective1(x) = (x+2)^2 - 10, and

       objective2(x) = (x-2)^2 + 20
```

```
% Plot two objective functions on the same axis
x = -10:0.5:10;
f1 = (x+2).^2 - 10;
f2 = (x-2).^2 + 20;
plot(x,f1);
hold on;
plot(x,f2,'r');
grid on;
title('Plot of objectives ''(x+2)^2 - 10'' and ''(x-2)^2 + 20''');
```




```
FitnessFunction = @(x) [(x+2)^2 - 10; (x-2)^2 + 20];
numberOfVariables = 1; [x,fval] =
gamultiobj(FitnessFunction,numberOfVariables)
```

```
Optimization terminated: maximum number of generations exceeded.
x =
   -2.0000
    2.0000
    2.0000
   -2.0000
   -0.4915
    1.3024
fval =
  -10.0000   36.0000
    6.0000   20.0000
    6.0000   20.0000
  -10.0000   36.0000
   -7.7244   26.2076
    0.9058   20.4866
```

Simulated Annealing and Threshold Acceptance

- Can find global optimum, and under certain conditions, which are difficult to check, finds the global optimum with probability 1
- Algorithm:
  1. Random candidate point $x = x_i + \tau\epsilon$
  2. Accept $x^{i+1} = x$ if $f(x) < f(x_i)$ or
     - ★ simulated annealing: with probability $\frac{1}{1+e^{(f(x)-f(x_i))/\tau}}$
     - ★ threshold: if $f(x) < f(x_i) + T$
  3. Lower the temperature, τ, and threshold, T
- In Matlab, `[x,fval] = simulannealbnd(@objfun,x0)` and `[x,fval] = threshacceptbnd(@objfun,x0)`
  - Requires Genetic Algorithm and Direct Search Toolbox
  - Many variations and options
    - ★ Options can affect whether converge to local or global optimum
    - ★ Read the documentation and/or use `optimtool`