Gradient Descent Method:

Consider the following optimization problem:

Minimize f(x)= ½\* xTQx + qTx

Where Q is a n x n positive definite matrix, and it is not necessarily a diagonal matrix. Note that the solution to this problem is x\*= -Q-1q. As an alternative, design a gradient descent algorithm that will solve this unconstrained problem..

**Solution:**

The objective function f(x) is convex and twice differentiable (which implies that the **dom** f is open).

Gradient of Objective Function:

= Qx+q

Hessian of Objective Function:

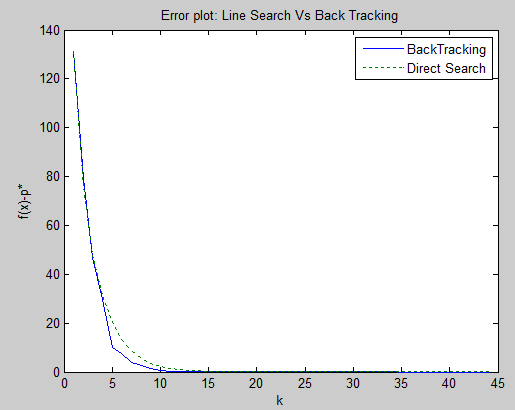
= Q

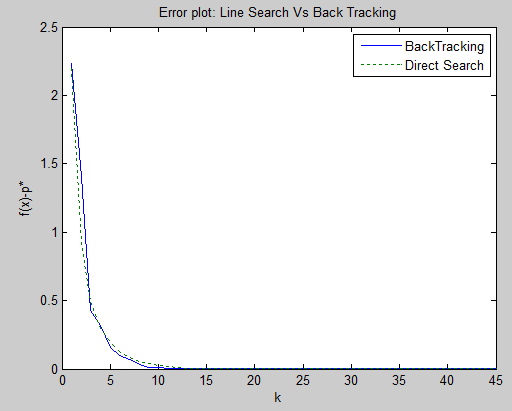
Since, f is differentiable and convex, a necessary and sufficient condition for a point x\* to be optimal is

By this, the optimal value is given by x\* = -Q-1q

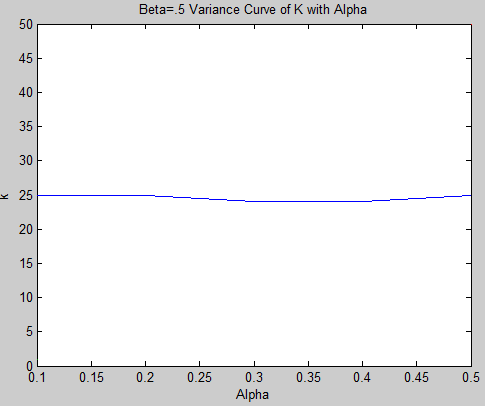
Algorithm:

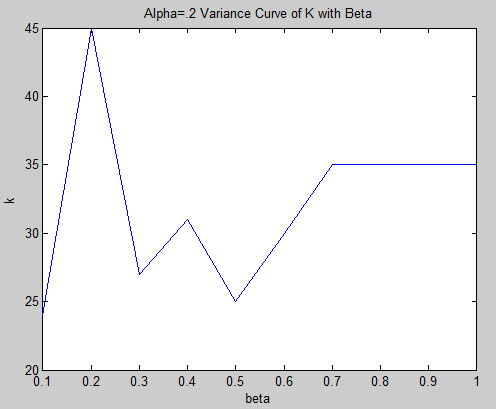
1. Generate Q, q and initial x arrays
2. Use CVX to find p\_opt
3. Gradient Descent Algorithm
   1. Define check parameters
   2. While (parameters ==OK)
      1. Calculate new gradient
      2. Predict t
      3. Calculate new x
      4. Update values
4. Optimum values are the last values



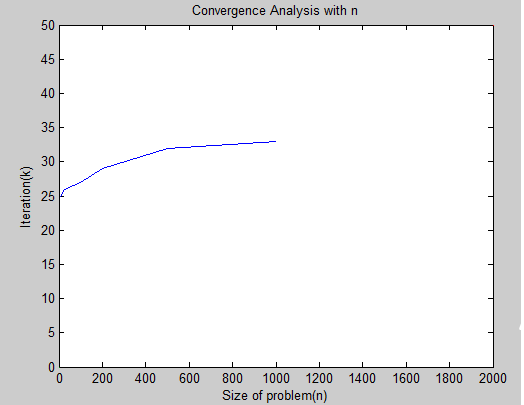


Figures above show the plot of no of iteration versus the difference of new f(x)- p\* for various iteration number. Figures have different starting point and both seem to do similarly for same starting point with only some variation in the difference value.



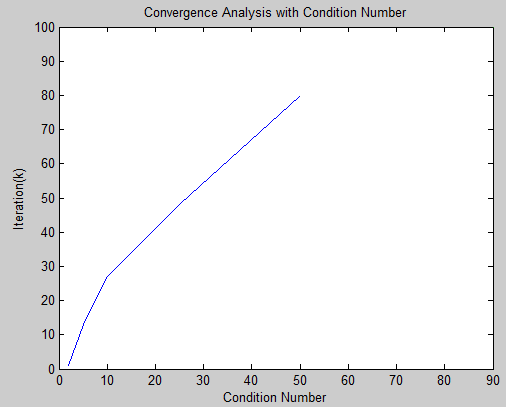


The variance in convergence was seen huge while alpha was kept constant with beta varying, as seen in figure 1. While on the other hand, varying alpha didn't have significant change in convergence.

d) 

We can see that even when the size of the problem increases so high, the no of iterations remains almost same.

e)



We can see that with the increase in condition number, the convergence decrease ie. number of iterations increase significantly.

(f) Design a steepest descent algorithm with the choice of norm is given as given at the bottom of pp. 476

of our textbook, where 𝑃 is selected as a diagonal matrix, whose diagonal elements are the same as those

of 𝑄. Is this new algorithm as sensitive to the condition number of 𝑄?

2 10000

cond =5 k=8

10 323

25 223

50 182

(g) Comment on how the gradient descent algorithm can be used to solve a least-squares problem and give an example of a least-square problem that you solve with this algorithm. Also, comment on the

computational advantages of using the gradient descent algorithm as compared with just solving the linear

system of equations 𝑥∗ = −𝑄−1𝑞 . Do these advantages remain when the steepest descent approach

described above is used? What are some potential disadvantages to using gradient descent, or steepest

descent as compared with just solving the linear system of equations?

The lease squares problem is a special case of quadratic minimization problem.

minimize || Ax-b||22 =xT(ATA)x-2(ATb)Tx+ bTb

And, the optimality conditions for the least square problem is ATAx\* = ATb

Gradient descent method is comparatively faster method of solving linear equations. Calculating the inverse of a matrix [Q-1] requires O(n3) steps so for large n values, it is a slow process.

The steepest descent method also requires calculation of inverse of a matrix [P-1] in order to determine the steepest descent direction. So, it is computationally slower than gradient descent.

The potential disadvantage of using descent methods over just solving the system of linear equations is that descent methods use zigzag pattern when gadient is nearly orthogonal to the direction of minimum point. This shows the convergence of descent methods.

2. *Newton's Method for Unconstrained Problems*: Replicate Fig. 9.20 for the objective function in equation

(9.20) in the textbook. Please include the expressions for the gradient and the Hessian in your report. Also

add a few other initial starting points.

Objective Function:

minimize f(x1,x2) =

Gradient of Objective Function

=

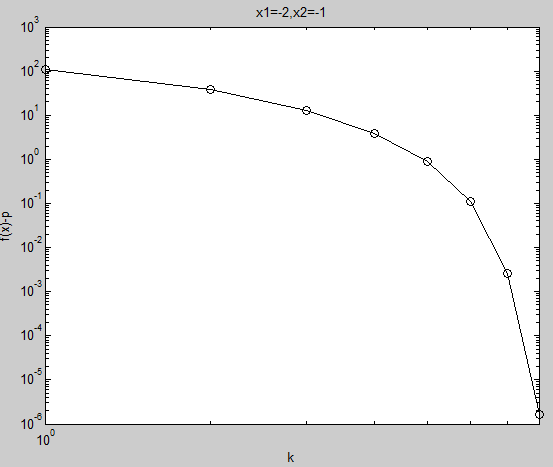
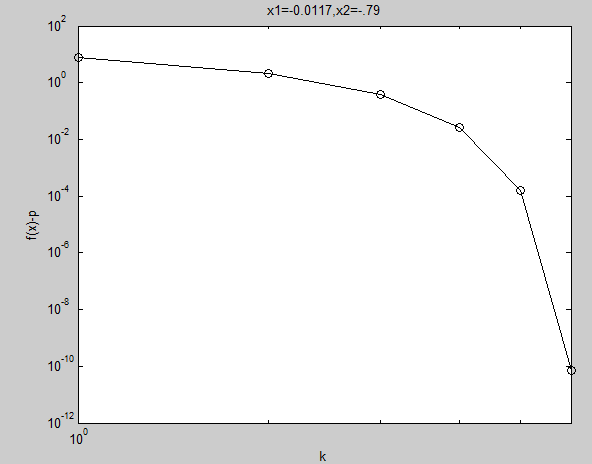
Hessian of Objective Function:

=

=

=

=



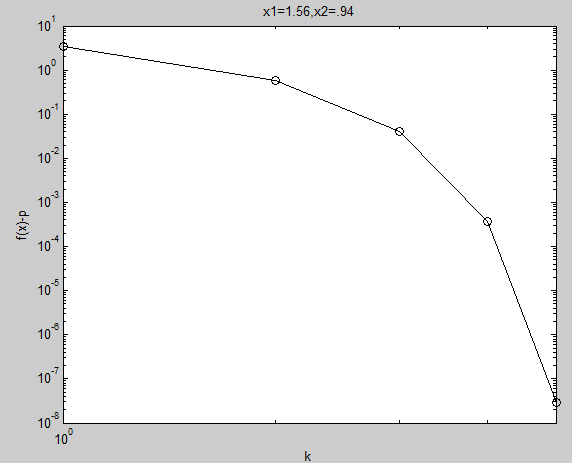
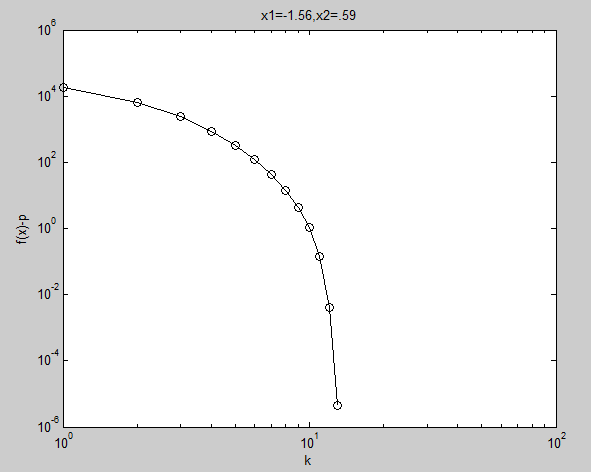


Fig. f(xk)-p plot for various starting points

3. *Newton's Method with Equality Constraints*: Consider the equality constrained entropy maximization

problem

minimize 𝑓(𝑥) = Σ 𝑥𝑖 log 𝑥𝑖

subject to 𝐴𝑥 = 𝑏

with 𝐝𝐨𝐦𝑓 = 𝐑++ and 𝐴 ∈ 𝐑𝑝×𝑛, with 𝑝 < 𝑛. Generate a problem instance with 𝑛 = 100 and 𝑝 = 30 by

choosing A randomly (checking that it has full rank), choosing 𝑥̂ as a random positive vector (e.g., with

entries uniformly distributed on [0, 1]) and then setting 𝑏 = 𝐴𝑥̂. (Thus, 𝑥̂ is feasible.)

Compute the solution of the problem using *Infeasible start Newton method*. You can use initial point 𝑥(0) = 𝑥̂ (to compare with the *standard Newton method*), and also the initial point 𝑥(0) = 𝟏**.**

**Note**: You need to implement the elimination method to solve KKT systems.

Solution:

Gradient of Objective Function: = for all i =1,2,..n

Hessian of objective function:

= for all i,j=1,2,..n

= diag

4. *Feasibility of an LP*: Develop an algorithm to test the feasibility of an LP using the approach in (11.19)

where 𝑓𝑖 (𝑥) are affine functions of 𝑥. Express the gradients and the Hessians needed for the algorithm

explicitly in your report and describe the algorithm.

For basic phase I optimization problem, it must be restated in log-barrier form as follows:

min ts-

where t **R** is a scalar value used in barrier method.

Gradient of Objective Function:

=

=

Hessian of objective function:

=

=

=

=

Effect of :

For small , barrier method parameter t increases by small amount in each iteration.

For large barrier method parameter t increases by a large amount in each iteration, so each iternation is not a good approximation of the next iteration which implies that the more number of Newton Method iterations are required to minimize the log-barrier optimization function.

Appendix

Question 1 c

%function[xopt,error]= gradient\_Descent(Q,q,n)

% input dimension of Q matrix

% n=input('Enter the dimension of the Q:');

n=5;

%n=25,50,100,200,500,1000 for size analysis

%% randomly generated matrix Q, vector q and initial point x0

% example 1 with n=2;

% load ex1\_n2 % generate Q,q,x0 with saved data

% load ex2\_n2\_cond

P=randn(n);

%% generate PSD matrix with definite condition numebr

con\_num=2; % con\_num=2,5,10,25,50 ==> condition number is con\_num

P=diag(diag(randi(con\_num,[n,n])));

ma=randi(n);

mi=randi(n);

while mi==ma

mi=randi(n);

end

P(ma,ma)=con\_num;

P(mi,mi)=1;

%%

% Q=[3,0,0,0,0;0,10,0,0,0;0,0,1,0,0;0,0,0,7,0;0,0,0,0,1]

% q=[0.877628740871287;1.03360564858667;0.419791320369650;0.601069227881255;-0.674018313717088]

% % make sure Q is PSD

Q=P;

q=randn(n,1);

% x0=[1;1;1;1;1]

%x0=[5;5;5;5;-15]

x0=round(randn(n,1));

eigen\_Q=eig(Q);

if (any(eigen\_Q)<0)

disp('Matrix Q is not a PSD matrix');

end

%% run cvx to find p\_opt

cvx\_begin

variable x(n);

dual variable y;

minimize ((1/2)\*quad\_form(x,Q)+q'\*x);

cvx\_end

p\_opt=cvx\_optval;

%% Parameters initialization

% termination tolerance

tol = 1e-6;

% maximum number of allowed iterations

num\_iter = 10000;

% minimum allowed perturbation

del\_x\_min = 1e-6;

% step size r)

alpha = 0.2;

beta=.5;

% initialize gradient norm, optimization vector, iteration counter, perturbation

norm\_g = inf;

x = x0;

num\_iter = 0;

Dx = inf;

% define the objective function:

f=@(x) (1/2)\*x'\*Q\*x+q'\*x;

fp=feval(f,x);

%% Gradient decent Algorithm

while( norm\_g>=tol && fp-p\_opt>=tol && num\_iter <= num\_iter && Dx >= del\_x\_min)

% calculate gradient:

gradient = (1/2)\*(Q'+Q)\*x+q;

norm\_g = norm(gradient);

dx=-gradient;

%t=exact\_line(x,dx,Q,q);

t=backtrack\_linesearch(f,dx,x,beta,alpha);

% update x

xnew = x - t\*gradient;

% check step

% update termination metrics

num\_iter = num\_iter + 1;

Dx = norm(xnew-x);

x = xnew;

fp=feval(f,x);

error(num\_iter)=abs(fp-p\_opt);

end

xopt = x;

fopt = f(xopt);

num\_iter = num\_iter - 1;

%% calculating condition number for quadratic optimization problem

e=eig(Q);

ymax=max(e);

ymin=min(e);

cond=ymax/ymin;

%%

plot(error)

% end

Plots for different sections:

%for b

%for x0=[1;1;1;1;1]

ber=[2.23618337414283,1.41039054480372,0.427817358932676,0.318977659909932,0.158780600271105,0.101717967063847,0.0659599016141989,0.0331532344528582,0.0134792641018469,0.00936671150600887,0.00549761558247519,0.00371763792049074,0.00117257466621734,0.000910632319293336,0.000461833292199509,0.000295086562971947,0.000192826791713840,9.45449868244852e-05,3.92871492005398e-05,2.70638016510727e-05,1.60661759911473e-05,1.05328002825145e-05,3.40962807243717e-06,2.61604976170560e-06,1.34834690967978e-06,8.56581964914227e-07,2.36175769375846e-05,1.56872806157393e-05,1.04202705952794e-05,6.92129669344066e-06,4.59699397059588e-06,3.05314350368402e-06,2.02775853475057e-06,1.34678020169421e-06,8.94517718119658e-07,3.45134497237165e-05,2.24851349378552e-05,1.46513167931239e-05,9.54669860830926e-06,6.21960212054962e-06,4.05195633845779e-06,2.64023746776587e-06,1.72032440481917e-06,1.12075409453016e-06,7.30118471459207e-07]

der=[2.19334256677091,0.906508352379073,0.489935941088911,0.299307690580731,0.193867767671015,0.127619594412269,0.0845762366813239,0.0561342192558386,0.0372792558443673,0.0247610825012668,0.0164464730963635,0.0109245693068344,0.00725630186578086,0.00481999898302321,0.00320155571000869,0.00212662169752886,0.00141256046305582,0.000938282418079139,0.000623236716384756,0.000413977643924257,0.000274978644840718,0.000182650188235822,0.000121323460934852,8.05866660937848e-05,5.35291511748470e-05,3.55554243920420e-05,2.36175769375846e-05,1.56872806157393e-05,1.04202705952794e-05,6.92129669344066e-06,4.59699397059588e-06,3.05314350368402e-06,2.02775853475057e-06,1.34678020169421e-06,8.94517718119658e-07,3.45134497237165e-05,2.24851349378552e-05,1.46513167931239e-05,9.54669860830926e-06,6.21960212054962e-06,4.05195633845779e-06,2.64023746776587e-06,1.72032440481917e-06,1.12075409453016e-06,7.30118471459207e-07]

x=[1:45]

%for x0=[5;5;5;5;-15]

% ber=[131.258281782310,79.3607258285895,46.9032203158418,29.8266600860421,9.86429356223374,7.46048462495035,3.89983080304858,2.46477816433668,1.63244294232760,0.764587624049605,0.330758826385471,0.224244592989289,0.135880435436900,0.0840995925840448,0.0285819711590035,0.0214483720869654,0.0113860821591980,0.00715997023855197,0.00477375924801360,0.00218476985012861,0.000964621453457326,0.000648813463722586,0.000397172214772645,0.000238698774712720,8.31781560061140e-05,6.17240036603350e-05,3.32546019856661e-05,2.08060912642027e-05,1.39604939886073e-05,6.24685127070812e-06,2.81357586873199e-06,1.87796377126848e-06,1.16088103496725e-06,6.77819510963573e-07,0,0,0,0,0,0,0,0,0,0]

% der=[131.134208503708,76.1003842310023,48.2238569671502,31.1976512524421,20.2907009233020,13.2055651899246,8.59732714467624,5.59850043727281,3.64592750142172,2.37464761389501,1.54671577941589,1.00764908577409,0.656490227937958,0.427759197190729,0.278733167056970,0.181588927441919,0.118305915779816,0.0770866917722485,0.0502304243378962,0.0327242287325606,0.0213198753022865,0.0138918299527562,0.00905201034557945,0.00589724907328038,0.00384204857280457,0.00250345331699520,0.00163126011674231,0.00106274603710821,0.000692374395715345,0.000451148450791861,0.000293968836541314,0.000191517896278803,0.000124772509258797,8.13015973193387e-05,5.29759566936239e-05,3.45134497237165e-05,2.24851349378552e-05,1.46513167931239e-05,9.54669860830926e-06,6.21960212054962e-06,4.05195633845779e-06,2.64023746776587e-06,1.72032440481917e-06,1.12075409453016e-06,7.30118471459207e-07]

% x=[1:44];

p1=plot(x,ber(x),x,der(x),':','LineWidth',1)

title('Error plot: Line Search Vs Back Tracking')

legend('BackTracking','Direct Search');

ylabel('f(x)-p\*')

xlabel('k')

% % plot(ler)

% x=[0:0.01:2\*pi];

% p1h=plot(x,sin(x),'k--',x,sin(x+pi/6),'k:',x,sin(x+pi/3),'k-.','LineWidth',1)

%%

%qs c plot

k=[24,45,27,31,25,30,35,35,35,35]

beta=[.1:.1:1];

p2=plot(beta,k)

title('Alpha=.2 Variance Curve of K with Beta')

ylabel('k')

xlabel('beta')

%%

k=[25,25,24,24,25]

alpha=[.1:.1:.5];

l=[1]

beta=[.1]

m=[50]

zeta=[.5]

p2=plot(alpha,k,beta,l,zeta,m)

title('Beta=.5 Variance Curve of K with Alpha')

ylabel('k')

xlabel('Alpha')

%%

%qs d

n=[5,25,100,200,500,1000]

ite=[25,26,27,29,32,33]

l=[0]

beta=[1]

m=[50]

zeta=[2000]

p2=plot(n,ite,beta,l,zeta,m)

title('Convergence Analysis with n')

ylabel('Iteration(k)')

xlabel('Size of problem(n)')

%%

%qs e

p=[2,5,10,25,50]

ite=[1,13,27,48,80]

m=[90]

zeta=[100]

p2=plot(p,ite,m,zeta)

title('Convergence Analysis with Condition Number')

ylabel('Iteration(k)')

xlabel('Condition Number')

Question 2

% define the Hesssian of the objective

function h = Hessian(x1,x2)

h = [exp(x1+3.\*x2-0.1)+exp(x1-3.\*x2-0.1)+exp(-x1-0.1) 3.\*exp(x1+3.\*x2-0.1)-3.\*exp(x1-3.\*x2-0.1)

3.\*exp(x1+3.\*x2-0.1)-3.\*exp(x1-3.\*x2-0.1) 9.\*exp(x1+3.\*x2-0.1)+9.\*exp(x1-3.\*x2-0.1)];

end

% define the gradient of the objective

function g = grad(x1,x2)

g = [exp(x1+3.\*x2-0.1)+exp(x1-3.\*x2-0.1)-exp(-x1-0.1);3.\*exp(x1+3.\*x2-0.1)-3.\*exp(x1-3.\*x2-0.1)];

end

function t = backtrack\_linesearch(f,dx,x,beta,alpha)

t = 1;

fk = feval(f,x);

xx = x;

x = x + t\*dx;

fk1 = feval(f,x);

while fk1 > fk + alpha\*t\*(dx'\*dx),

ff=fk + alpha\*t\*(dx'\*dx);

t = t\*beta;

x = xx + t\*dx;

fk1 = feval(f,x);

end

clear all

clc

% define starting point

% x0 = round(3\*rand(2,1));

x0=[-1;2];

% termination tolerance

tol = 1e-6;

% maximum number of allowed iterations

maxiter = 1000;

% minimum allowed perturbation

dxmin = 1e-6;

% step size ( beta=0.8 causes instability, beta=0.6 accurate with lowest iteration number)

alpha = 0.01;

beta=0.5;

% initialize gradient norm, optimization vector, iteration counter, perturbation

gnorm = inf;

x = x0;

niter = 0;

dx = inf;

% define the objective function:

f=@(x1,x2) exp(x1+3\*x2-0.1)+exp(x1-3\*x2-0.1)+exp(-x1-0.1);

% plot objective function contours for visualization for 2D:

figure(1); clf; fcontour(f); %axis equal;

hold on

% redefine objective function syntax for use with optimization:

f2 = @(x) f(x(1),x(2));

% gradient descent algorithm:

while and(gnorm>=tol, and(niter <= maxiter, dx >= dxmin))

% calculate gradient:

g = grad(x(1),x(2));

gnorm = norm(g);

d=-g;

% take step:

t=backtrack\_linesearch(f2,d,x,beta,alpha);

xnew = x + t\*d;

% check step

if ~isfinite(xnew)

display(['Number of iterations: ' num2str(niter)])

error('x is inf or NaN')

end

% plot current point for 2D

plot([x(1) xnew(1)],[x(2) xnew(2)],'ko-')

refresh

% update termination metrics

niter = niter + 1;

dx = norm(xnew-x);

x = xnew;

end

xopt = x;

fopt = f2(xopt);

niter = niter - 1;

...............................................

clear all

clc

% define starting point

x0 = randn(2,1);

%x0=[-2;-1];

% termination tolerance

tol = 1e-6;

% maximum number of allowed iterations

maxiter = 1000;

% minimum allowed perturbation

dxmin = 1e-6;

% step size

alpha = 0.1;

beta=0.7;

% initialize gradient norm, optimization vector, iteration counter, perturbation

landa = inf;

x = x0;

niter = 0;

dx = inf;

% define the objective function:

f=@(x1,x2) exp(x1+3\*x2-0.1)+exp(x1-3\*x2-0.1)+exp(-x1-0.1);

% redefine objective function syntax for use with optimization:

f2 = @(x) f(x(1),x(2));

% plot objective function contours for visualization for 2D:

% figure(1); clf; fcontour(f,[-5 5;-5 5]); axis equal; hold on

[X,Y]=meshgrid(-5:0.1:5,-5:0.1:5);

Z=exp(X+3.\*Y-0.1)+exp(X-3.\*Y-0.1)+exp(-X-0.1);

figure(1); clf; contour(X,Y,Z); axis equal; hold on

% gradient descent algorithm:

while and((landa/2)>=tol,(niter <= maxiter))%, dx >= dxmin

% calculate gradient:

g = grad(x(1),x(2));

% calculate hessian

h=Hessian(x(1),x(2));

landa = g'\*inv(h)\*g;

% direction value

d=-inv(h)\*g;

% take step:

t=backtrack\_linesearch(f2,d,x,beta,alpha);

xnew = x + t\*d;

% check step

if ~isfinite(xnew)

display(['Number of iterations: ' num2str(niter)])

error('x is inf or NaN')

end

% plot current point for 2D

plot([x(1) xnew(1)],[x(2) xnew(2)],'ko-')

refresh

% update termination metrics

niter = niter + 1;

% calculate error

xx=[-0.3465;1.612e-07];

p\_star=feval(f2,xx);

fiter=f2(x);

error(niter)=fiter-p\_star;

% dx = norm(xnew-x);

x = xnew;

end

xopt = x;

fopt = f2(xopt);

niter = niter - 1;

figure(2)

%%

loglog(error,'ko-')

ylabel('f(x)-p')

xlabel('k')

title('x1=-0.0117,x2=-.79')

...................................................................................

Question 3

clear all

clc

n=100;

p=30;

%load A

A=rand(100,30)

x0=ones(n,1);

%%

MAXITERS = 100;

ALPHA = 0.01;

BETA = 0.5;

RESTOL = 1e-7;

x=x0;

nu=zeros(p,1);

p\_star=-34.347345215642460;

for i=1:MAXITERS

% build the residual vector

r = [1+log(x)+A'\*nu; A\*x-b];

% solve the equation Ax=b based on KKT matrix

sol = -[diag(1./x) A'; A zeros(p,p)] \ r;

Dx = sol(1:n); Dnu = sol(n+[1:p]);

% check stopping criterion

if (norm(r) < RESTOL), break; end;

% implement backtracking line search

t=1;

while (min(x+t\*Dx) <= 0), t = BETA\*t; end;

while norm([1+log(x+t\*Dx)+A'\*(nu+Dnu); A\*(x+Dx)-b]) > ...

(1-ALPHA\*t)\*norm(r), t=BETA\*t; end;

% update x and v

x = x + t\*Dx; nu = nu + t\*Dnu;

% calculate error

f=x'\*log(x);

error(i)=f-p\_star;

res\_dual(i)=norm(r(1:n));

res\_pri(i)=norm(r(n+[1:p]));

end;

figure(1)

loglog(error,'ko-')

i=[1:6]

loglog(i,res\_dual,i,res\_pri);

Question 4