Optimization Project

Jose Canela, Paul Chang, Sara Kwan
Math 458
Professor Mancera

Table of Contents

Introduction

Problem 1

Consider the linear program:

minimize $x_1 + 2x_2 + 3x_3\ x_1$
subject to: $x_1 + x_2 + x_3 = 1,\ x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0$

Problem 1 is a linear programming optimization problem in which we are asked to minimize a three-variable function subject to inequality constraints. The function we are asked to minimize is a simple first-degree three-variable polynomial. Our main constraint function is also a first-degree three variable polynomial set equal to 1, with three inequality constraints (one on each variable). Essentially, the set of constraints limit x_1, x_2, and x_3 to values between 0 and 1 (inclusive).

Part A

Draw a diagram that illustrates the geometry of the problem.

Part A asks us to create a visual representation of the optimization problem in MATLAB. This can be accomplished by graphing the function to be minimized on the same plot as the constraint function, and then limiting the possible variable values based on the constraint inequalities.

Part B

Write a computer program to implement the simplex method to solve this problem.

Part C

Write a computer program to implement an interior point method to solve this problem.

Part C asks us to solve the linear programming problem specifically using an interior point method; an interior point method uses an algorithm to approach the problem from an iterative fashion, using points from a feasible region defined by a set of constraints. Any point in this region is called a *feasible solution*. We can visualize our feasible region clearly in the plot created when solving part A.

Problem 2

Consider an investor who wants to allocate one unit of wealth among n assets offering random rates of return $e_1, \dots, e_n$ *respectively. The means* $\bar{e}_i = E\{e_i\}$, i = 1, … , n *and the covariance matrix*

$$Q = \begin{bmatrix} E\{(e_1 - \bar{e}_1)^2\} & \dots & E\{(e_1 - \bar{e}_1)(e_n - \bar{e}_n)\} \\ \dots & \dots & \dots \\ E\{(e_n - \bar{e}_n)(e_1 - \bar{e}_1)\} & \dots & E\{(e_1 - \bar{e}_1)^2\} \end{bmatrix}$$

are known, and we assume that Q is invertible. If xi is the amount invested in the ith asset, the mean and the variance of the return investment $y = \sum_{i=1}^{n} e_i x_i$ *are* $\bar{y} = E\{y\} = \sum_{i=1}^{n} \bar{e}_i x_i$ *and* $\sigma^2 = x'Qx$

The investor's problem is to find the portfolio x = (x1,...,xn) that minimizes the variance $E\{(y - \bar{y})^2\}$ *to achieve a given level of mean return E{y}, say E{y} = m. Thus the problem is*

Minimize x'Qx

Subject to: $\sum_{i=1}^{n} x_i = 1, \quad \sum_{i=1}^{n} \bar{e}_i x_i = m$

Use Lagrange Multipliers theory to find the solution to this problem and also see how the solution varies with m.

Problem 2 asks us to find the portfolio that minimizes the variance in order to achieve a given level of mean return. In order to do this, we implement the Lagrange Multipliers theory to minimize the variance subject to the constraints, and find the unknown portfolio. We also observe how the solution varies with changes in m, the given level of mean return.

Problem 3

Choose a 500x500 positive definite matrix Q *and a 500 vector* b*. Using the conjugate directions method get a good approximation to the solution of the system* Qx = b*. Use*

also Gauss elimination method to solve this system and compare the results and comment on the advantages and disadvantages of each method.

Methods

Problem 1

Part A

Part A is simply asking for the graph of both functions on one plot. We can do so by creating a function and within it, defining our function to be minimized as well as our constraint function. Since we want to show $x_1+2x_2+3x_3$ subject to our constraint set, we set it equal to 1 on our graph; this is because $x_1+x_2+x_3=1$. In order to scale the graph down, we can finally limit our x, y, and z axes to values between 0 and 1.

Part B

Given the properties of this linear programming problem, I converted the problem into a standard vector form:

$$minimize\ c^T x$$
$$subject\ to\ Ax = b,\ x \geq 0$$

Hence, the following components:

$$c = [1,\ 2,\ 3] \qquad x^T = [x_1,\ x_2,\ x_3] \qquad b^T = [1, 0, 0] \qquad A = \begin{matrix}[1,\ 1,\ 1]\\ [0,\ 0,\ 0]\\ [0,\ 0,\ 0]\end{matrix}$$

One way to solve this problem is to use the simplex method algorithm:

1. **Create a canonical augmented matrix $[I_m, Y_{m,\ n-m}, y_0]$ corresponding to an initial basic feasible solution $[x_B^T, 0^T] = [y_{10},...,y_{m0},0,...,0]^T$**
2. **Compute the relative cost coefficients r_j corresponding to basic variables that aren't basic.**
3. **IF $r_j \geq 0$ for all j, END.**
    a. **The current basic feasible solution is optimal.**
4. Select an index q such that $r_q < 0$.
5. IF no $y_{iq} > 0$, END because the problem is unbounded;

   a. ELSE,
      i. Compute the index p = argmin$_i$ {$y_{i0}/y_{iq} : y_{iq} > 0$}. (If more than one index i minimizes y_{i0}/y_{iq}, we let p be the smallest such index.)
6. Update the canonical augmented matrix by pivoting about the (p, q)th element.
7. Repeat Steps 2 to 6 until Step 3 is satisfied.

- The following list is the set of formulas for calculating y_{ij} and r_j:
  - z = value of obj function cx^T. Hence, for the basic solution, $z = z_0 = c_B x_B^T$.
  - we update the canonical augmented matrix using the pivot equations
    - $y'_{ij} = y_{ij} - (y_{pj}/y_{pq})\, y_{iq},\ i \neq p$
    - $y'_{pj} = (y_{pj}/y_{pq})$
  - $r_j = c_i - z_i$ for i = m+1,...,n.

Part C

For the purposes of the question, we will transform our problem data into the following vectors:

$c = [1,\ 2,\ 3]$ $x = [x_1,\ x_2,\ x_3]$ $b = [1]$ $A^T = [1,\ 1,\ 1]$

With this notation, our problem looks like this:

$$minimize\ c^T x$$
$$subject\ to\ Ax = b,\ x \geq 0$$

To solve this problem using an interior point method, we used an affine scaling method. This is an algorithm that starts at a point within the feasible region (established by the constraints) and moves within it to identify an optimal solution. It is important to note that our constraints involve both an equation and inequalities; due to the inequality restraints, we know that any point p_0 within the feasible region Ω has all nonnegative components. The goal of the algorithm is to find a different point p_1 by moving in a direction that minimizes the objective function. Observe that this means that we should choose a p_0 that is a solution of $Ax = b$ from near the center of Ω so that we can maximize our movement in any direction. At the very least, p_0 must be strictly interior.

The equation $p_1 = p_0 + \alpha_0 d_0$ can be used to represent the new point p_1, where d_0 is the direction of movement and α_0 is the step size. One method of determining the search direction is to take the gradient of the objective function; however, that does not work

for this method because taking the gradient does not guarantee that the new point lies in the feasible set--which is a requirement for an interior point method.

In order to ensure that p_1 lies in the feasible set, we need it to lie in the nullspace of our vector A. This is because in order for our solutions to come from the feasible set, we must satisfy both $Ap_0 = b$ and $Ap_1 = b$. We can linearly combine these two equations to get $A(p_0 - p_1) = \alpha_0 A d_0 = 0$. So, to keep d_0 close to the gradient of the objective function and also in the nullspace of A, we orthogonally project $-c$ onto $null(A)$; the resulting projection becomes d_0. This is done using an orthogonal projector $P = I_n - A^T(AA^T)^{-1}A$ such that $d_0 = -Pc$. Now, we find our point $p_1 = p_0 - \alpha_0 Pc$; this is actually going to be the first iteration of an algorithm used to project the gradient.

In the event that we are given a p_0 that is not centralized, the affine scaling algorithm can transform the point to a center point $e = [e_1 ... e_n]$. The formula for this transformation is $e = D_0^{-1} p_0$ where D_0 is a diagonal matrix with the entries of the main diagonal being the components to the vector p_0. Since p_0 must be strictly interior, we know this matrix is invertible.

Now, since we've changed our p_0 value, our entire coordinate system and therefore linear programming problem has changed. Our new linear program can be viewed as follows:

$$minimize\ c^{*T}x^*$$
$$subject\ to\ A^*x^* = b,\ x^* \geq 0$$

We now have $c_0^* = D_0 c$ and $A_0^* = AD_0$. With these new values, we can compute a new P_0^*, d_0^* and p_1^*. This process can be repeated iteratively to generate a sequence of points $\{p_k\}$ such that $p_{k+1} = p_k + \alpha_k d_k$. At each stage, to ensure that p_k remains a strictly interior point, we limit our step size to $\alpha_k \in (0, 1)$, typically choosing values such as 0.9 or 0.99.

Since this algorithm is capable of an iterating an infinite number of times, we just need to add a stopping criteria to end our algorithm. We can set our algorithm to end when a chosen value is below an $\varepsilon > 0$ of our choosing; for example, we can stop when $\frac{|cp_{k+1} - cp_k|}{max\{1, |cp_k|\}} < \varepsilon$.

Problem 2

First, we set up the problem's initial conditions. The means and covariance matrix are both known in the problem, so we initialize the vector of means, e, and covariance matrix, Q, by assigning random values to them. In addition, we create a positive definite matrix for the covariance matrix, as it makes it easier to work with. Our unknown portfolio, x, is a vector x = (x1,...,xn).

```
%initial parameters
n = 5;
m = 5;
M = randn(n,n);
Q = transpose(M) * M; %positive definite matrix
e = randn(n,1); %expected values of rates of returns
x = sym('x',[1 n]); %unknown portfolio
```

Q =

	1	2	3	4	5
1	4.3378	-0.7009	1.8764	-4.4862	-0.8504
2	-0.7009	1.0546	-0.5433	1.7346	-1.3049
3	1.8764	-0.5433	4.4706	0.7386	0.0352
4	-4.4862	1.7346	0.7386	11.0023	-5.1949
5	-0.8504	-1.3049	0.0352	-5.1949	10.2375

e =

	1
1	-1.0511
2	-0.4174
3	1.4022
4	-1.3677
5	-0.2925

With these values initialized, we use the Lagrange Multipliers Theory to minimize the variance: x'Qx. This is done by creating a Lagrangian expression, defined by: $\mathcal{L}(x,y,\lambda) = f(x,y) - \lambda * g(x,y)$. In our case, we have multiple constraints, as well as n assets. So, our Lagrangian expression results in: $\mathcal{L}(x,y,\lambda) = x'Qx + \lambda_1(m-x*e) + \lambda_2(1-x*1)$, with unknown λ_1 and λ_2.

Now that we have created our Lagrangian expression, we take the gradient of the expression, taking partial derivatives with respect to each variable, and set them equal to zero. We do this in order to solve for the unknown λs. Taking the partial derivative of the Lagrangian with respect to x, we can solve the equation with respect to x. Using that equation, we can substitute the x's in the partial derivatives with respect to λ_1 and λ_2 in order to solve for the lambdas.

$\frac{\partial L}{\partial x} = 0 = 2Qx - \lambda_1 e - \lambda_2 1 \Rightarrow x = \frac{1}{2}\lambda_1 Q^{-1} e + \frac{1}{2}\lambda_2 Q^{-1} 1$
$\frac{\partial L}{\partial \lambda 1} = 0 = m - x'e \Rightarrow \frac{1}{2}\lambda_1 (e'Q^{-1}e) + \frac{1}{2}\lambda_2 (e'Q^{-1}1)$
$\frac{\partial L}{\partial \lambda 2} = 0 = 1 - x'1 \Rightarrow \frac{1}{2}\lambda_1 (x'Q^{-1}1) + \frac{1}{2}\lambda_2 (1'Q^{-1}1)$

After λ_1 and λ_2 are found, we can plug them back into the first equation in order to find the values of x, the portfolio that minimizes the variance. In order to calculate variance, compute σ^2 = x'Qx.

In order to see the effect that differing values of m have on optimizing this problem, we set m to different values, and run the Lagrange multiplier, recording variance to see how it changes in accordance to differing values of m.

Problem 3

To solve for Qx = b using Gaussian Elimination, one performs elementary row operations in order to get the augmented matrix [Q|b] into row echelon form. In this case it was important to note that we created a random 500 x 500 matrix A = A^T and performed the operation A*A^T to ensure that it was positive definite. There after one performs a back substitution by solving x given that one element of x is known. The Conjugate Gradient method (CGM) is quite different. The following is a summary of the CGM:

1. Set k = 0 and choose an initial point x_0.
2. Let g_0 = b - Q*x_0.
    a. IF g_0 = 0, END; The x_0 is the optimal solution.
    b. ELSE, set $d_0 = -g_0$.

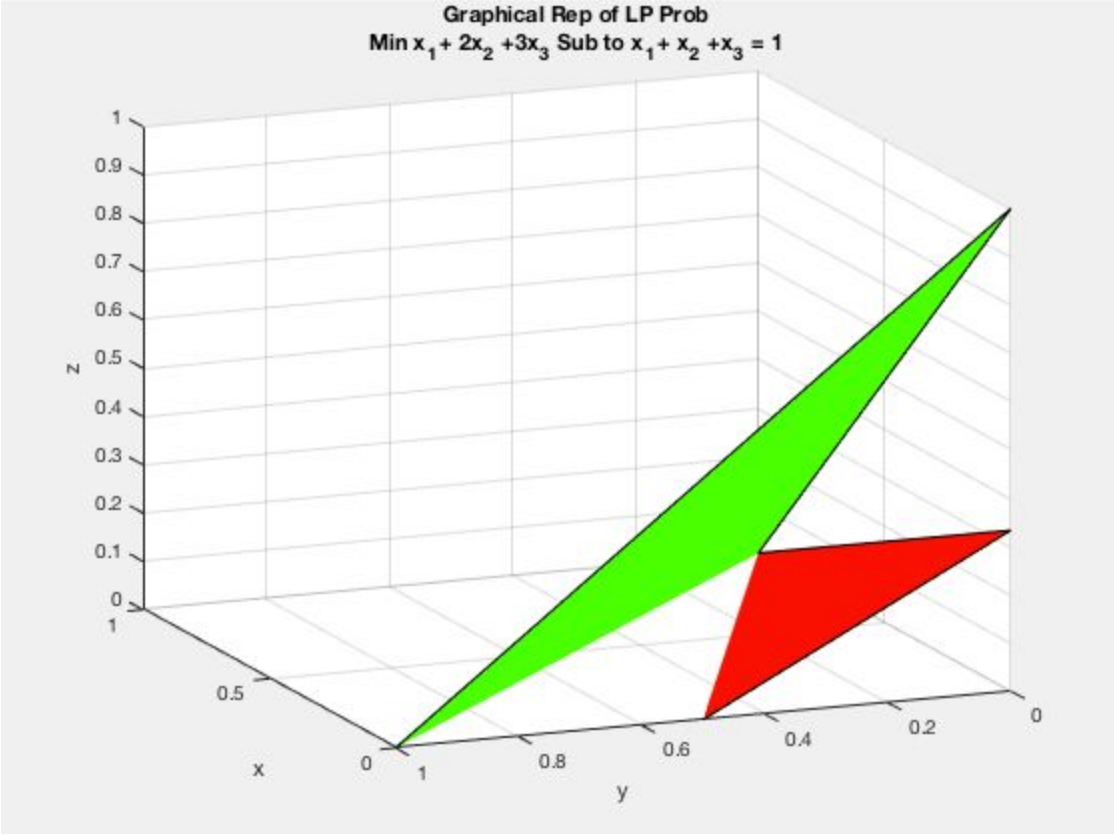
   Now these next steps are done for k = 1 to k=n:
3. Next compute the step size alpha or α:
    a. α_k = -dot(d_k,g_k)/dot(d_k,Q*d_k)
        i. Note: dot(x,y) is the matlab function for the dot product of two vectors.
4. Now, we update x:
    a. $x_{k+1} = x_k + \alpha_k * d_k$.
5. Calculate g_{k+1} = b - Q*x_{k+1}. IF g_{k+1} = 0, END.
6. Now, if Step 5 isn't satisfied we must update the direction we are moving in to a direction that is orthogonal to our new point in the direction the optimal solution.This is known as the Q-conjugate direction. But do this we need a step size called Beta or β_k:
    a. β_k= dot(g_{k+1},Q*d_k)/dot(d,Q*d_k).
7. Update direction: $d_{k+1} = -g_{k+1} + \beta_k * d_k$
8. Set k= k + 1 and repeat steps 3 to 7 until step 5 is met.

- It is important to note that although the algorithm says to stop if the residual g = b - Q*x = 0, in practice one uses the condition that, if the norm of the residual is less that a given allowed error tolerance, atol, one stops the algorithm.

Results

Problem 1

Part A



As shown in the graph above, the planes created by our two functions intersect at the point (1, 0, 0). We should expect to get this as our solution for parts b and c.

Part B

Command window results:

```
>> test_1b


Initial tableau:
    1    1    1    1
    0    0    0    0
    0    0    0    0
    0    1    2   -1
x =
    1
    0
    0
v =
    1
    2
    3
```

As seen above, applying the simplex method to the linear programming problem gives a solution of (1,0,0). Hence, the objective function $x_1 + 2x_2 + 3x_3$ reaches a minimum of 1. The reason the tableau provides a cost of -1 is due to fact that this tableau is providing the maximum cost for the linear problem: *max* $-x_1 - 2x_2 - 3x_3$ *subject to:* $x_1 + x_2 + x_3 = 1,\ x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0$. This is equivalent to the problem given in the prompt. Also note that the elements of v correspond to the indices corresponding to the basic solution of Ax = b.

Part C

Command window results:

```
>> test_1c
Terminating: Relative change in objective function <
  1.0000e-07
Final point =
  1.0000e+00   9.8982e-10   2.8466e-09
```

```
Number of iterations =
    8
```

As seen above, our interior point method algorithm gives the solution (1, 0, 0), which is the same result as returned by the simplex method. This method took 8 iterations. Since our constraint equation was simple, it was easy to pick a starting p_0 solution (or u in our code) that was centralized; however, if we had picked a different point, the affine scaling algorithm would still have been able to locate a more central point and find the same solution.

For example, here are the results when our initial solution is coded in as $u = [0.1; 0.8; 0.1]$.

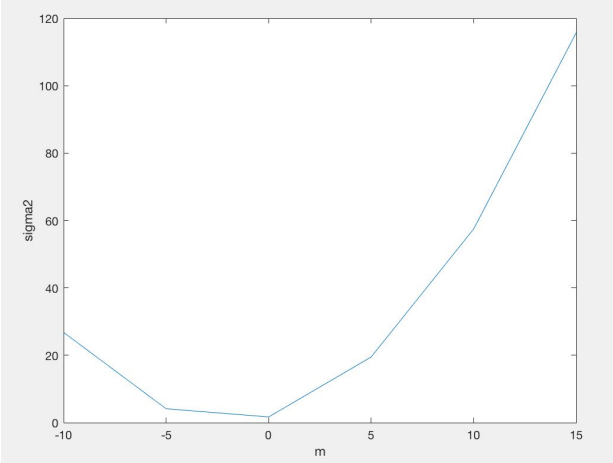
```
>> test_1c
Terminating: Relative difference between iterates <
  1.0000e-07
Final point =
  1.0000e+00   6.5965e-09   4.9786e-10
Number of iterations =
    8
```

As you can see, we still reach the same optimal solution when our initial solution is not centered.

Problem 2

Given the initial parameters set in the problem, we found the portfolio that minimizes the variance:
x = [-3.2557 5.1440 3.4911 -2.6425 -0.7370]
sigma2 = 19.4883

We observed how the solution changed with differing values of m:
m = [5,10,15,20,25,30]
We plotted the resulting variance in accordance to the different values of m.



From the data observed, we see that the minimum variance of the portfolio rises as the value of m rises.

By calculating sigma^2, we find that it has a parabolic form along mean m:

$\sigma^2 = 1/(ac-b^2)\ (cm^2 - 2bm + a)$

- a = ½e'Q^{-1}e
- b = ½e'Q^{-1}1
- c = ½1Q^{-1}1

Our results agrees with the mathematical form shown above.

Problem 3

Initial Parameters for Gaussian Elimination:

A = randn(500,500);
Q = transpose(A)*A; %positive definite
b = randn(500,1);
Aug =[A b] %Augmented matrix

Initial Parameters for Conjugate Gradient Method:

x = randn(500,1) %x_0
Q = transpose(A)*A; %positive definite
atol = 1e-8

There are various differences between the Conjugate Gradient Method (CGM) and Gaussian Elimination (GE) starting with errors. Given that n = 500, there is a vast amount of operations (about 83,333,333 operations) . This leads to significant round off error accumulation. In particular, the norm of the residual b - Ax corresponding to Gaussian Elimination was 486.42, whereas the norm of the residual b - Ax corresponding to Conjugate Gradient Method was 6.518. In addition, the Conjugate

Gradient Method is unique in that given that $Q = Q^T > 0$, this method converges to a solution in n steps (or 500 steps in this case). Depending on what you consider a step in Gaussian Elimination, GE takes 499 steps. However, again, the amount of operations provides a massive handicap given susceptibility to round off error and especially time. While GE took 2.4 seconds to solve Ax = b, CGM took 0.78 seconds to solve Ax = b! In other words, CGM was 300% faster than GE!

Appendix Code

Problem 1A

```
%% Final Project: Optimization Question 1a
% minimize x1+2x2+3x3
% subject to x1+x2+x3 = 1; x1,x2,x3 >= 0
% Part A: Draw a diagram that illustrates the geometry of the problem.
% For readability, I am using x = x1, y = x2, and z = x3
 function LP_GraphRep_partA
close all;
figure;
plot1 = gca;
% Converting the constraint into a function
syms f(x, y, z)
f(x, y, z) = x + y + z == 1;
syms g(x, y);
g(x, y) = solve(f(x, y, z), z);
% Initializing the range that the x, y, z values can take
xrange = 0:1;
yrange = 0:1;
zrange = zeros(numel(xrange), numel(yrange));
% Computing the values of x, y, z corresponding to the initial constraint
for i = 1:numel(yrange)
  for j = 1:numel(xrange)
      a = g(xrange(j), yrange(i));
      zrange1(i, j) = double(a(1));
  end
end
%Plotting the initial constraint
surf(plot1, xrange,yrange,zrange1, 'FaceColor', 'green')

hold on %Used to have two plots on the same graph
% Creating the Obj Function subject to the given constraints
syms f(x, y, z)
f(x, y, z) = x + 2*y + 3*z ==1; % Constraint of Intersection
syms g(x, y);
g(x, y) = solve(f(x, y, z), z);
% Initializing the range that the x,y,z values can take
xrange = 0:1;
```

```
yrange = 0:1;
zrange = zeros(numel(xrange), numel(yrange));
% Computing the values of x, y, z of the Obj Function
% given the initial constraints and the intersection constraint
for i = 1:numel(yrange)
 for j = 1:numel(xrange)
      a = g(xrange(j), yrange(i));
      zrange1(i, j) = double(a(1));
 end
end
% Plotting Obj Function given the Constraints
surf(plot1, xrange, yrange, zrange1, 'FaceColor', 'red')
% Customizing the overall graph
xlim(0:1);
ylim(0:1);
zlim(0:1);
xlabel('x')
ylabel('y')
zlabel('z')
title({'Graphical Rep of LP Prob'; 'Min x_1+ 2x_2 +3x_3 Sub to x_1+ x_2 +x_3 = 1'})
end
```

Problem 1B

Simplex Method

```
%% Simplex Method for Solving LinProg Prob: min c'x sub to Ax=b, x>=0
function [x,v]=SIMPLEX(c,A,b,v,options)
% Inputs/Arguments:
%               c = Coefficient vector corresponding to c'x
%               A = Matrix corresponding to the constraint Ax=b
%               b = A Constant Vector
%           [A b] = The canonical form of Ax=b
%               v = The vector of indices of the basic columns of A
%                   * v_i-th column of A is the i-th standard basis
%                     vector.
%         options : Parameters that indicate whether or not the results
%                   will be displayed tabularly and how pivot element
%                   of the alg. is selected.
%                     * If options(1) = 1, a tabular output of
%                       results.
```

```
%                   * If options(1) = 0, no tabular output is shown.
%                   * If options(5) = 1, use Bland's Rule for
%                   pivoting.
%                   * If options(5) = 0,
%                     ** q is the index that corresponds to the
%                        most negative relative cost coefficient
%                        r_q.
%                     ** p is the index that mins  y_i0/y_iq.
%                        *** If more than one index i mins
%                            y_i0/y_iq,let p be the smallest such
%                            index.
% Output:
%              x = Solution
%                 OR
%              x = Solution
%              v = The vector of indices of the basic columns of A
format compact;
%format short e; %converts values w/ long decimals into scientific notation
options = foptions(options);
display = options(1);
n=length(c); % number of coefficients corresponding to c'x
m=length(b); % number of basic columns in A (and number rows in b)
c_B=c(v(:)); % cost vector corresponding to the basis B
r = c'-c_B'*A; % row vector of relative cost coefficients
cost = -c_B'*b;
tableau=[A b;r cost]; %Matrix that is a tabular rep. of the problem
% If display = option(1) = 1, display a tabular version of the LP prob.
if display
  disp(' ');
  disp('Initial tableau:');
  disp(tableau);
end

% While r_j <0 for all j (if r_j >=0 for all j, the current basic solution
% is optimal)
while ones(1,n)*(r' >= zeros(n,1)) ~= n
    if options(5) == 0 % Selecting an index q s.t. r_q<0
      [r_q,q] = min(r);
    else % Use Bland's Rule
```

```
            q=1;
        while r(q) >= 0
            q=q+1;
        end
    end
        min_ratio = inf;  % If the above if-else statements aren't met,
        p=0;              % y_iq=<0. Hence, p=0. This is very bad.

% However, if the above if-else statements are met, we can calculate p.
    for i=1:m
        if tableau(i,q)>0
            if tableau(i,n+1)/tableau(i,q) < min_ratio
                min_ratio = tableau(i,n+1)/tableau(i,q);
                p = i;
            end
        end
    end

%  But, again if p = 0 (NO y_iq>0), we must stop b/c prob is unbounded.
    if p == 0
            disp('Problem unbounded');
        break;
    end
     tableau=pivot(tableau,p,q); % Update Tableau given new pair of (p,q)
%   Go back to the initial step displaying the new Tableau and repeat the
%   process given better/updated row vector of relative cost coefficients,
%   r, and  p index of v (q-th column of the basis enters the basis and
%   the p-th column leaves).
    if display
        disp('Pivot point:');
        disp([p,q]);
        disp('New tableau:');
        disp(tableau);
    end
      v(p) = q;
      r = tableau(m+1,1:n);
end
% Getting Solution x
x=zeros(n,1);
```

```
x(v(:))=tableau(1:m,n+1);
end
```

Pivoting Method

```
%% Pivoting about the (p,q)th Element of the Given Matrix G.
function G_new=PIVOT(G,p,q)
% For speed purposes, let's reallocate memory for G_new by having
G_new = zeros(size(G));
for i=1:size(G,1)
   if i==p
        G_new(p,:)=G(p,:)/G(p,q);
   else
        G_new(i,:)=G(i,:)-G(p,:)*(G(i,q)/G(p,q));
   end
end
end
```

Test_1b

```
A=[1 1 1; 0 0 0; 0 0 0];
b=[1;0; 0];
c=[1;2;3];
v=[1;2;3];
options(1)=1;
[x,v]=SIMPLEX(c,A,b,v,options)
```

Problem 1C

affine_scaling_1c.m

```
function [x,N] = affscale(c,A,b,y,choices)
x_k=y;
%choices(1) controls display output;
%1 returns tabular display of results, and 0 (the default) shows no results
%choices(2) is the precision required for the final point
%choices(3) is the precision required cost value
%choices(14) = max number of iterations
%choices(18) = alpha
if length(choices) >= 14
 if choices(14)==0
   choices(14)=1000*length(x_k);
 end
```

```
else
  choices(14)=1000*length(x_k);
end
%if length(choices) < 18
  choices(18)=0.99; %optional step size
%end
%clc;
format compact;
format short e;
choices = foptions(choices);
print = choices(1);
epsilon_x = choices(2);
epsilon_f = choices(3);
max_iter=choices(14);
alpha=choices(18);
n=length(c);
m=length(b);
for k = 1:max_iter,
  xcurr=x_k;
  D = diag(xcurr);
transA = A*D;
  transP = eye(n) - transA'*inv(transA*transA')*transA;
  d = -D*transP*D*c;
  if d ~= zeros(n,1),
    nonzd = find(d<0);
    r = min(-xcurr(nonzd)./d(nonzd));
  else
    disp('Terminating: d = 0');
break; end
  x_k = xcurr+alpha*r*d;
  if print,
    disp('Iteration number k =')
    disp(k);  %print iteration index k
    disp('alpha_k =');
    disp(alpha*r);  %print alpha_k
    disp('New point =');
    disp(x_k'); %print new point
end %if
  if norm(x_k-xcurr) <= epsilon_x*norm(xcurr)
```

```
    disp('Terminating: Relative difference between iterates <');
    disp(epsilon_x);
    break;
end %if
  if abs(c'*(x_k-xcurr)) < epsilon_f*abs(c'*xcurr),
    disp('Terminating: Relative change in objective function < ');
    disp(epsilon_f);
    break;
end %if
  if k == max_iter
    disp('Terminating with maximum number of iterations');
  end %if
end %for
if nargout >= 1
  x=x_k;
  if nargout == 2
    N=k;
  end
else
  disp('Final point =');
  disp(x_k');
  disp('Number of iterations =');
  disp(k);
end %if
```

test_1c.m

```
A = [1 1 1];
b = [1];
c = [1; 2; 3];
y = [0.33; 0.33; 0.34];
choices(1)=0;
choices(2)=10^(-7);
choices(3)=10^(-7);
% Solve for x
affine_scaling_1c(c,A,b,y,choices)
```

Problem 2

```
%% Problem #2
%initial conditions
```

```
n = 5;
m = 5;
rng('default'); %seeded
rng(1);
M = randn(n,n);
Q = transpose(M) * M; %positive definite matrix
e = randn(n,1); %expected values of rates of returns
x = sym('x',[1 n]); %unknown portfolio
%find portfolio and sigma^2 using Lagrange Multipliers
[xx,sigma2] = Lagrange(Q,e,m,x)
%Solve for different m
m = [5,10,15,20,25,30];
sigma2 = zeros(size(m));
for i = 1:length(m)
   [xx,sigma2(i)] = Lagrange(Q,e,m(i),x);
end
%graph m and sigma2
plot(m,sigma2);
xlabel('m')
ylabel('sigma2')
```

Lagrange Function

```
function [x,sigma2] = Lagrange(Q,e,m,x)
%Input arguments:
%        Q = covariance matrix
%        e = vector of expected value of returns
%        m = constraint of problem
%        x = vector of unknown portfolio
%  Use Lagrange Multiplier theory to find a portfolio that minimizes
%  variance to achieve a given level of mean return, m.
lambda = sym('lambda', [1 2]); %unknown lambdas
 %Lagrange
L = x*Q*transpose(x) + lambda(1)*(m-x*e) + lambda(2)*(1-x*ones(5,1));
%partial diff
diffLx = Q*transpose(x) - lambda(1)*e - lambda(2)*ones(5,1);
diffL1 = diff(L,lambda(1));
diffL2 = diff(L,lambda(2));
%solve for lambdas
```

```
A = [.5*transpose(e)*inv(Q)*e, .5*transpose(e)*inv(Q)*ones(5,1);
.5*transpose(e)*inv(Q)*ones(5,1), .5*ones(1,5)*inv(Q)*ones(5,1)];
b = [m;1];
lambda = inv(A)*b;
lambda1 = lambda(1);
lambda2 = lambda(2);
%substitute in x
diffLx = subs(diffLx);
%solve for x
a = solve(diffLx,x);
x(1) = a.x1;
x(2) = a.x2;
x(3) = a.x3;
x(4) = a.x4;
x(5) = a.x5;
x = double(x);
%find sigma^2
sigma2 = x * Q * transpose(x);
end
```

Problem 3

Conjugate Gradient Method

```
%% Conjugate Gradient Method for Solving Ax=b
function [x,numIt] = CONJGRAD(A,x,b,atol)
% Inputs/Arguments:
%               A = Matrix A=A'>0
%               x = Initial Condition of x=x0
%               b = A Constant Vector
%            atol = Allowed Error Tolerance
% Output:
%               x = Solution
%           numIt = Number of Iterations

n = length(b); % Number of rows/elements in b (also a Step Counter for
          % the max number of iterations needed for the problem to be
          % solved.
g = -(b - A*x); % Residual of Ax=b
d = -g; % Q-conjugate direction
tic
```

```
for numIt = 1:n % Given that CGM solves this problem in max n steps
   Ad = A*d;
   alpha = dot(d,-g)/dot(d,Ad); % Step Size for updating x
   x = x + alpha*d;
   g = -(b - A*x);
   if sqrt(dot(-g,-g)) < atol % If norm of residual is less that atol,
      return              % we stop b/c the min has been achieved.
   else % If norm of residual is not less than atol, we update: d.
      beta = -dot(-g,Ad)/dot(d,Ad); % Step Size for updating d
      d = -g + beta*d;
   end
toc
end
```

Conjugate Gradient Method (Upgraded)

```
%% Conjugate Gradient Method for Solving Ax=b
function [x_new,numIt] = CONJGRAD1(A,x_new,b,atol)
% Inputs/Arguments:
%               A = Matrix A=A'>0
%              x = Initial Condition of x = x0
%               b = A Constant Vector
%            atol = Allowed Error Tolerance
% Output:
%               x = Final Solution
%           numIt = Number of Iterations
n = length(b); % Number of rows/elements in b (also a Step Counter for
           % the max number of iterations needed for the problem to be
           % solved.
g_curr = -(b - A*x_new);% Current Residual of Ax=b
if norm(g_curr)<= atol % Checking if norm of residual is already <= atol
  disp('current residual is less than');
  disp(atol);
   return;
end
d = -g_curr; % Q-conjugate direction
tic
for numIt = 1:n % Given that CGM solves this problem in max n steps
   x_curr=x_new;
   Ad = A*d;
```

```
    alpha = dot(d,-g_curr)/dot(d,Ad); % Step Size for updating x
    x_new = x_curr + alpha*d;
    if norm(x_new - x_curr)<= atol*norm(x_curr) % Check if x_new is min pt.
        disp('norm of diff between iterates is less than');
        disp(atol);
         return;
    end
    g_old = g_curr;
    g_curr = -(b - A*x_new);
    if norm(g_curr) <= atol % If norm of residual is less than atol,
        disp('norm of gradient is less than')
        disp(atol);
        return;              % we stop b/c the min has been achieved.
    else % If norm of residual is not less than atol, we update: d.
        beta = -dot(-g_curr,Ad)/dot(d,Ad); % Step Size for updating d
        d = -g_curr + beta*d;
    end
end
toc
disp('Conj grad residual is');
disp(norm(g_curr))
end
```

Comparing Conjugate Gradient Method & Gaussian Elimination

Finaltest3.m

```
rng('default'); %seeded
rng(1);
A = randn(500,500);
Q = transpose(A)*A; %positive definite
b = randn(500,1);
x_new = randn(500,1);
atol = 1e-8;
%Conj Grad
[x_new,numIt] = CONJGRAD1(Q,x_new,b,atol)
pause %PAUSE
%Gaussian Elimination Test
 [m,n] = size(A);
Aug = [A b];
```

```
%forward elimination
tic
for k = 1:n-1
   for i = k+1:n
      factor = Aug(i,k)/Aug(k,k);
      Aug(i,k:n+1) = Aug(i,k:n+1) - factor*Aug(k,k:n+1);
   end
end
%back substitution
x = zeros(n,1);
x(n) = Aug(n,n+1)/Aug(n,n);
for i = n-1:-1:1
   x(i) = (Aug(i,n+1) - Aug(i,i+1:n)*x(i+1:n))/Aug(i,i);
end
toc
x;
resid_GE = norm(b - Q*x)
```