EE364b Homework 6

1. *Conjugate gradient residuals.* Let $r^{(k)} = b - Ax^{(k)}$ be the residual associated with the kth element of the Krylov sequence. Show that $r^{(j)T} r^{(k)} = 0$ for $j \neq k$. In other words, the Krylov sequence residuals are mutually orthogonal. Do not use the explicit algorithm to show this property; use the basic definition of the Krylov sequence, *i.e.*, $x^{(k)}$ minimizes $(1/2)x^T Ax - b^T x$ over $\mathcal{K}_k$.

   Solution. Since $x^{(k)}$ minimizes $f(x) = (1/2)x^T Ax - b^T x$ over $\mathcal{K}_k = \mathrm{span}\{b, Ab, \ldots, A^{k-1}b\}$, $\nabla f(x^{(k)})^T y = 0$ for all $y \in \mathcal{K}_k$. The residual $r^{(k)} = b - Ax^{(k)} = -\nabla f(x^{(k)})$, and for any $j < k$ the residual $r^{(j)} \in \mathcal{K}_k$. Therefore $r^{(k)T} r^{(j)} = 0$ for all $j \neq k$.

2. *CG and PCG example.* In this problem you explore a variety of methods to solve $Ax = b$, where $A \in \mathbf{S}^n_{++}$ has block diagonal plus sparse structure: $A = A_{\mathrm{blk}} + A_{\mathrm{sp}}$, where $A_{\mathrm{blk}} \in \mathbf{S}^n_{++}$ is block diagonal and $A_{\mathrm{sp}} \in \mathbf{S}^n$ is sparse. For simplicity we assume A_{blk} consists of k blocks of size m, so $n = mk$. The matrix A_{sp} has N nonzero elements.

   (a) What is the approximate flop count for solving $Ax = b$ if we treat A as dense?

   (b) What is the approximate flop count for an iteration of CG? (Assume multiplication by A_{blk} and A_{sp} are done exploiting their respective structures.) You can ignore the handful of inner products that need to be computed.

   (c) Now suppose that we use PCG, with preconditioner $M = A_{\mathrm{blk}}^{-1}$. What is the approximate flop count for computing the Cholesky factorization of A_{blk}? What is the approximate flop count per iteration of PCG, once a Cholesky factorization of A_{blk} if found?

   (d) Now consider the specific problem with A_{blk}, A, and b generated by `ex_blockprecond.m`. Solve the problem using direct methods, treating A as dense, and also treating A as sparse. Run CG on the problem for a hundred iterations or so, and plot the relative residual versus iteration number. Run PCG on the same problem, using the block-diagonal preconditioner $M = A_{\mathrm{blk}}^{-1}$. Give the solution times for dense direct, sparse direct, CG (to relative residual 10^{-4}, say), and PCG (to relative residual 10^{-8}, say). For PCG break out the time as time for initial Cholesky factorization, and time for PCG iterations.

   Hints.

   - To force Matlab to treat A as dense, use `full(A)`.
   - You do not need to implement the conjugate gradient algorithm; instead use the `pcg` function in Matlab.
   - To block precondition with $M = A_{\mathrm{blk}}^{-1}$, first find the Cholesky factorization of A_{blk}, *i.e.*, lower triangular L with $LL^T = A_{\mathrm{blk}}$. The Matlab code to implement block preconditioning is

```
L = chol(A_blk)';
[...] = pcg(A,b,tolerance,MAXITER,L,L');
```

Matlab uses a sparse Cholesky to factor A_{blk}, which is less efficient than using a dense Cholesky factorization on each block separately, but it is efficient enough to make the point.

Solution.

(a) The order is $n^3 = k^3 m^3$.

(b) Each iteration is dominated by multiplication by A. Multiplication by A_{blk} costs order km^2; multiplication by A_{sp} costs order N. The total order is $km^2 + N$.

(c) To efficiently calculate $M^{-1}z$, we first find a Cholesky factorization of M (which costs order km^3); once this is done we can calculate $M^{-1}z$ at a cost of order km^2 by two substitution steps. This is added to the cost of multiplying by A, which is order $km^2 + N$. The overall order is $km^2 + N$, the same as for CG. (But we have to count the original cost of the Cholesky factorization as well.)

(d) The following Matlab code carries out CG and PCG.

```
%block preconditioning solution
clear all;
ex_blockprecond;

A_full = full(A);
fprintf('\nStarting dense direct solve ...\n');
time_start = cputime;
x_star_dense = A_full\b;
time_end = cputime;
relres_dense = norm(A*x_star_dense - b)/norm(b);
fprintf('Relative residual: %e\n', relres_dense);
fprintf('Dense direct solve done.\nTime taken: %e\n',...
                                        time_end - time_start);

fprintf('\nStarting sparse direct solve ...\n');
time_start = cputime;
x_star_sparse = A\b;
time_end = cputime;
relres_sparse = norm(A*x_star_dense - b)/norm(b);
fprintf('Relative residual: %e\n', relres_sparse)
fprintf('Sparse direct solve done.\nTime taken: %e\n',...
                                        time_end - time_start);

fprintf('\nStarting CG ...\n');
```

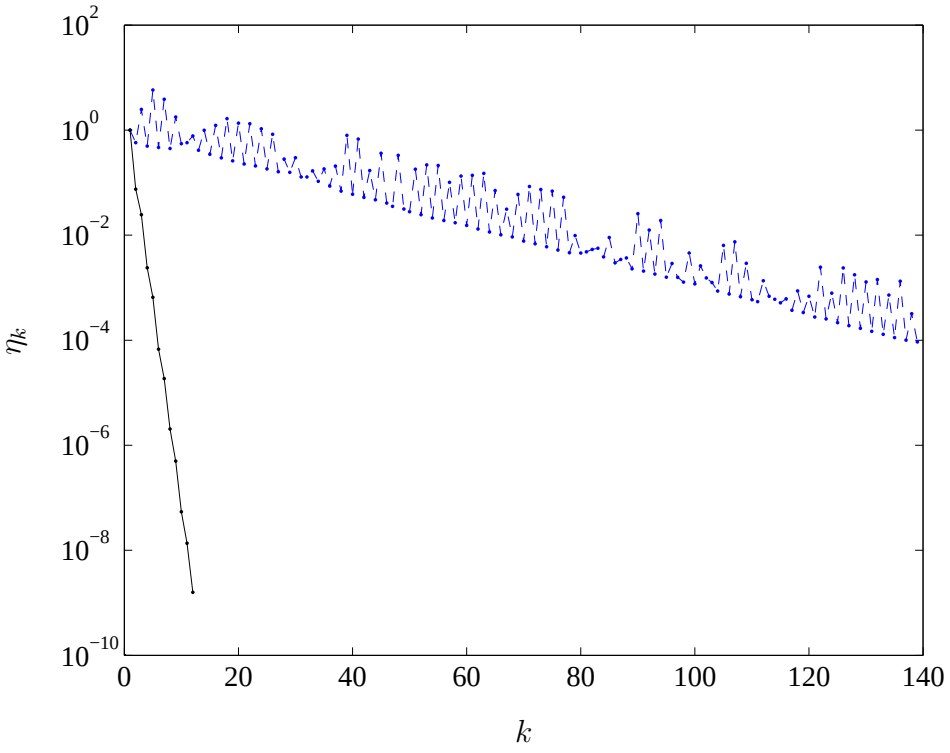
```
time_start = cputime;
[x,flag,relres,iter,resvec] = pcg(A,b,1e-4,200);
time_end = cputime;
fprintf('CG done. Status: %d\nTime taken: %e\n', flag,...
                                          time_end - time_start);
figure; semilogy(resvec/norm(b), '.--'); hold on;
set(gca,'FontSize', 16, 'FontName', 'Times');
xlabel('cgiter'); ylabel('relres');

time_start = cputime;
L = chol(A_blk)';
time_end = cputime;
tchol = time_end - time_start;
fprintf('\nCholesky factorization. Time taken: %e\n', tchol);

fprintf('\nStarting CG with block preconditioning ...\n');
time_start = cputime;
[x,flag,relres,iter,resvec] = pcg(A,b,1e-8,200,L,L');
time_end = cputime;
fprintf('PCG done. Status: %d\nTime taken: %e\n', flag,...
                                          time_end - time_start);
semilogy(resvec/norm(b), 'k.-'); hold off;
print('-depsc', 'ex_blockprecond_relres.eps');
fprintf('Total time block preconditioned PCG: %e\n',...
                                      tchol+time_end - time_start);
```

This generates the following output.

```
Starting dense direct solve ...
Relative residual: 1.084699e-13
Dense direct solve done.
Time taken: 1.392000e+01

Starting sparse direct solve ...
Relative residual: 1.084699e-13
Sparse direct solve done.
Time taken: 6.470000e+00

Starting CG ...
CG done. Status: 0
Time taken: 5.760000e+00

Cholesky factorization. Time taken: 4.200000e-01
```

```
Starting CG with block preconditioning ...
PCG done. Status: 0
Time taken: 9.600000e-01
Total time block preconditioned PCG: 1.380000e+00
```

The following plot shows residual convergence with and without block preconditioning.



3. *Sum-of-norms heuristic for block sparsity.* The basic ℓ_1 heuristic for finding a sparse vector x in a convex set $\mathcal{C}$ is to minimize $\|x\|_1$ over $\mathcal{C}$. We are given a partition of x into subvectors: $x = (x^{(1)}, \ldots, x^{(k)})$, with $x^{(i)} \in \mathbf{R}^{n_i}$. Our goal is to find an $x \in \mathcal{C}$ with the fewest number of subvectors $x^{(i)}$ nonzero. Like the basic sparsity problem, this problem is in general difficult to solve. But a variation on the ℓ_1 heuristic can work well to give a block sparse, if not the block sparsest, vector. The heuristic is to minimize

$$\sum_{i=1}^{k} \|x^{(i)}\|$$

over $x \in \mathcal{C}$.

   (a) What happens if the norms above are ℓ_1 norms? Would you expect x to be sparse? Block sparse? (Your answer can be very brief.)

(b) Generate a nonempty polyhedron $\{y \mid Ay \preceq b\}$, with $A \in \mathbf{R}^{50\times 100}$, and $b \in \mathbf{R}^{50}$. We will divide x into 20 subvectors, each of length 5. Use the heuristic above, with ℓ_1, ℓ_2, and ℓ_∞ norms on the subvectors, to find block sparse x in your polyhedron.
Hints. You may find the functions `norms` and `reshape` useful.

Solution.

(a) If the norms are ℓ_1 norms, the objective is simply $\|x\|_1$. We would expect x to be sparse, but not necessarily block sparse.

(b) The following code generates the problem instance, and applies the sum-of-norms heuristic.

```
randn('state',0);
m = 50; n = 100;
A = randn(m,n); b = randn(m,1);

% \ell_1 heuristic
cvx_begin
    variable x(n)
    A*x <= b;
    minimize(norm(x,1))
cvx_end
xell1 = x;

% \ell_2 norm
cvx_begin
    variable x(n)
    A*x <= b;
    minimize(sum(norms(reshape(x,5,20))))
cvx_end
xell2 = x;

% \ell_\infty norm
cvx_begin
    variable x(n)
    A*x <= b;
    minimize(sum(norms(reshape(x,5,20),inf)))
cvx_end
xellinf = x;
```

For the particular problem instance we generate, the ℓ_1 norm gives 15 nonzero blocks, the ℓ_2 norm gives 13 nonzero blocks, and the ℓ_∞ norm gives 12 nonzero blocks.

4. *Sparse Bayes network identification.* Suppose you are given samples $y_1, \ldots, y_N \in \mathbf{R}^n$ from an $\mathcal{N}(0, \Sigma)$ distribution, where $\Sigma \succ 0$. (See page 355 of the textbook.) From these samples you need to estimate the parameter Σ. You'd like Σ^{-1} to be sparse, if possible. A sparse Σ^{-1} corresponds to a compact Bayes network representation, since $(\Sigma^{-1})_{ij} = 0$ means that x_i and x_j are conditionally independent, given the other x_k's.

   (a) Suggest a simple method for finding Σ with Σ^{-1} sparse, with the likelihood of Σ nearly maximum. Your method should involve a parameter $\lambda \geq 0$ that adjusts the trade-off between suboptimality in likelihood, and sparsity of Σ^{-1}.

   (b) Carry out your method on the data found in `sp_bayesnet_data.m`. The true inverse covariance, Σ^{-1}, is stored in the matrix `Siginv`, and the samples $y_1, \ldots, y_N$ are stored in the matrix `Ys`. (You may need to experiment with different values of λ.) Plot the sparsity patterns of Σ^{-1} and your reconstruction $\hat{\Sigma}^{-1}$.

   Solution. To find the maximum likelihood estimate (see page 355 of the textbook), we solve the problem,

$$\begin{array}{ll} \text{maximize} & \log\det S - \mathbf{Tr}(SY) \\ \text{subject to} & S \succ 0 \end{array}$$

   where S is our variable, and

$$Y = \frac{1}{N}\sum_{k=1}^{N} y_k y_k^T$$

   is the sample covariance of $y_1, \ldots, y_N$. We set $\hat{\Sigma} = S^{-1}$. To find a sparse $\hat{\Sigma}^{-1}$, we solve the ℓ_1 regularized problem,

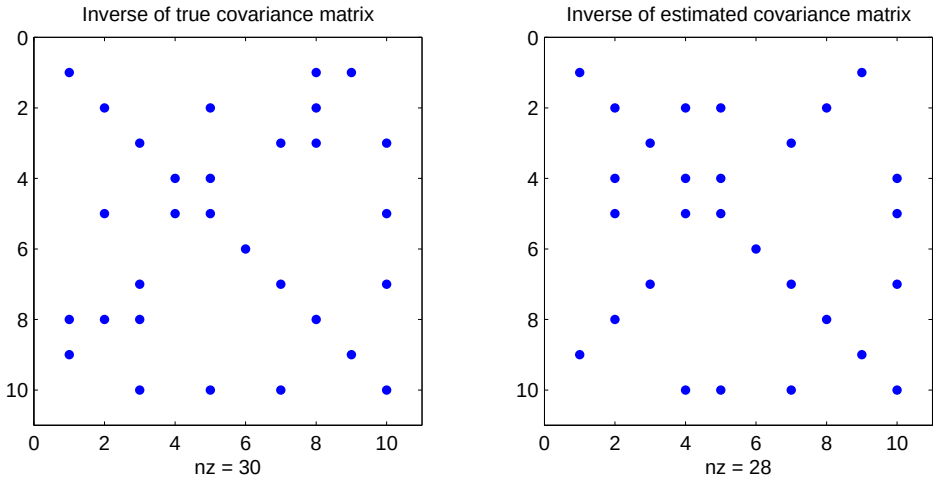
$$\begin{array}{ll} \text{maximize} & \log\det S - \mathbf{Tr}(SY) - \lambda \sum_{i,j} |S_{ij}| \\ \text{subject to} & S \succ 0 \end{array}$$

   with parameter $\lambda \geq 0$. The following code implements this method. We use $\lambda = 0.025$.

```
sp_bayesnet_data;
Y = (Ys*Ys')./N; lambda = 0.025;

cvx_begin sdp
    variable S(n,n) symmetric
    maximize(log_det(S)-trace(S*Y)-lambda*sum(norms(S,1)))
    S >= 0;
cvx_end
S(find(abs(S) < 1e-4)) = 0;

figure;
subplot(121);
spy(Siginv);
```

```
title('Inverse of true covariance matrix')
subplot(122);
spy(S);
title('Inverse of estimated covariance matrix');
print('-depsc','sp_bayesnet.eps');
```

The following figure shows the sparsity patterns of Σ^{-1}, and the reconstruction $\hat{\Sigma}^{-1}$.



5. *Rank minimization via the nuclear norm.* A simple heuristic for finding a matrix of low (if not minimum) rank, in a convex set, is to minimize its nuclear norm, *i.e.*, the sum of its singular values, over the convex set. Test this method on a numerical example, obtained by generating data matrices $A_0, \ldots, A_n \in \mathbf{R}^{p\times q}$ and attempting to minimize the rank of $A_0 + x_1A_1 + \cdots + x_nA_n$. You can use $n = 50$, $p = 10$, $q = 10$.

Solution. The following matlab code implements the method for $p = 10$, $q = 10$ and $n = 50$.

```
randn('state',0);
n = 50; p = 10; q = 10;
A = zeros(p,q,n); A0 = randn(p,q);
for i = 1:n A(:,:,i) = randn(p,q); end;

cvx_begin
    variable x(n)
    obj = A0;
    for i = 1:n obj = obj+x(i)*A(:,:,i); end;
    minimize(norm_nuc(obj))
cvx_end
```

```
B = obj;
sig = svd(B);
```

At the optimum, the singular values of $A_0 + x_1A_1 + \cdots + x_nA_n$ are

$$(5.60, 3.58, 2.65, 1.70, 1.48, 0.17, 0, 0, 0, 0),$$

i.e., the matrix has rank 6.

6. *Portfolio investment with linear and fixed costs.* The goal is to optimally invest an initial amount B in a portfolio of n assests. Let x_i be the amount (in dollars) of asset i that we purchase. We assume that no short selling is allowed, *i.e.*, $x \succeq 0$. We pay a fixed cost β for each asset we invest in, *i.e.*, for each asset with $x_i > 0$. We also pay a fee that is proportional to the amount purchased, given by $\alpha^T x$, where α_i is the fee (rate) associated with asset i. The budget constraint can be expressed as

$$\mathbf{1}^T x + \beta\, \mathbf{card}(x) + \alpha^T x \leq B.$$

The mean return on the portfolio is given by $\mu^T x$, where μ_i is the mean return of asset i, and the variance of the portfolio is $x^T \Sigma x$, where Σ is the covariance of the price changes. Our goal is to minimize the portfolio standard deviation, subject to meeting a minimum mean return requirement. This can be expressed as the problem

$$\begin{array}{ll} \text{minimize} & (x^T \Sigma x)^{1/2} \\ \text{subject to} & \mu^T x \geq R_{\min}, \qquad x \succeq 0 \\ & \mathbf{1}^T x + \beta\, \mathbf{card}(x) + \alpha^T x \leq B. \end{array} \tag{1}$$

This is a convex-cardinality problem. Once the choice is made of which assets to invest in, the problem becomes convex. But there are 2^n possible subsets of assets to invest in, so exhaustive search over all of these is not practical for $n \geq 20$.

You will focus on a particular instance of this problem, with data given in the file `l1_heuristic_portfolio_data.m` on the class web site.

(a) *Heuristic portfolio investment.* You will use the following ℓ_1-norm based heuristic to approximately solve the portfolio investment problem. First we replace $\mathbf{card}(x)$ with $\gamma \mathbf{1}^T x$ (which is the same as $\gamma \|x\|_1$, since $x \succeq 0$), where $\gamma \geq 0$ is a parameter, to get the problem

$$\begin{array}{ll} \text{minimize} & (x^T \Sigma x)^{1/2} \\ \text{subject to} & \mu^T x \geq R_{\min}, \qquad x \succeq 0 \\ & \mathbf{1}^T x + \beta\gamma \mathbf{1}^T x + \alpha^T x \leq B. \end{array} \tag{2}$$

Solve (2) for, say, 50 values of γ in $[0, 25]$. Note that when you solve the problem for $\gamma = 0$, you are just ignoring the fixed investment costs. The solutions of

(2) need not be feasible for the portfolio investment problem (1), since the true budget constraint can be violated.

For each solution of (2), note the sparsity pattern of x. Fix this sparsity pattern (which makes $\mathbf{card}(x)$ constant), and solve the problem (1). This procedure is called *polishing*.

Plot the portfolio standard deviation obtained (after polishing) versus γ. What is the minimum standard deviation, $\sigma_{\min}$, that you obtain? Give the best portfolio found (*i.e.*, the assets purchased, and the amounts for each one). Also plot $\mathbf{card}(x)$, *i.e.*, the number of assets invested in, versus γ.

Hint. To determine the sparsity pattern of x after solving (2), you'll need to use a reasonable (positive) threshold to determine if $x_i = 0$, as in `find(x<1e-3)`.

Solution.

```
cvx_quiet(1);
l1_heuristic_portfolio_data;

% l1 heuristic
gammas = linspace(0,25,50);
stds = [];
cards = [];
xs = [];
for i=1:length(gammas)
    % replace card(x) with gamma*||x||_1 = gamma*1'*x
    cvx_begin
        variable x(n)
        minimize quad_form(x,Sigma)
        x>=0
        mu'*x>=Rmin
        sum(x) + Beta*gammas(i)*sum(x) + Alpha'*x <= B
    cvx_end
    % polishing
    idx = find(x<1e-3);
    card = n-length(idx);
    cards = [cards card];
    cvx_begin
        variable x(n)
        minimize quad_form(x,Sigma)
        x>=0
        mu'*x>=Rmin
        sum(x) + Beta*card + Alpha'*x <= B
        x(idx) == 0
    cvx_end
    stds = [stds sqrt(cvx_optval)];
```

```
    xs = [xs x];
end
[std_l1, id] = min(stds);
x_l1 = xs(:,id);
mean_l1 = mu'*x_l1;
disp('The minimum standard deviation we obtain is: ');
disp(std_l1);
disp('The associated portfolio is: ');
disp(x_l1);

figure;
plot(gammas, stds)
xlabel('gamma');
ylabel('stddev');
figure
plot(gammas, cards)
xlabel('gamma');
ylabel('card(x)');
```

The minimum standard deviation found after polishing is 0.1384 and the associated portfolio is

```
x_l1 =
    0.0000
    0.0000
    0.0000
    1.1015
    0.6008
    0.4820
    0.0000
    0.3117
    0.0000
    0.2521
    0.0000
    0.0000
    0.0000
    0.0606
    0.0000
    0.0000
    0.1787
    0.0000
    0.0111
    0.0533
```

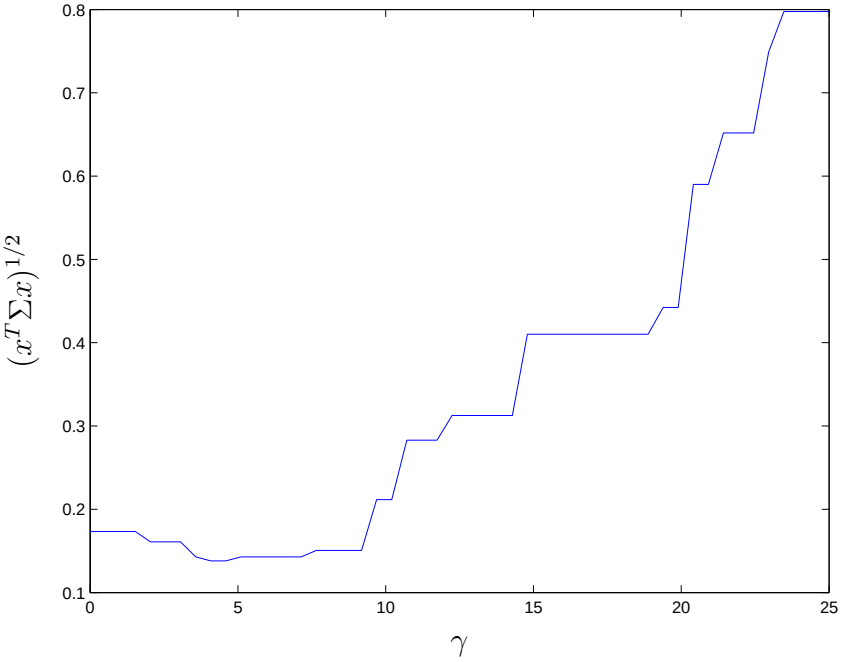


Figure 1: Standard deviation of return versus γ

```
0.0000
0.0000
0.1605
0.0000
0.0000
0.0000
0.0710
0.2192
0.0000
0.0916
```

The MATLAB script returns the plots shown in figures 1 and 2.

(b) *A lower bound.* For some values of γ the optimal value of (2) is a lower bound on the optimal value of the original portfolio investment problem (1). Find a simple value $\tilde{\gamma}$ of γ for which this is the case. Compute the corresponding lower bound and compare it to the standard deviation found in (a).

Solution.

Take $\tilde{\gamma} = 1/B$. It is obvious that $0 \leq x_i \leq B$ for $i = 1, \ldots, n$. Since

$$\frac{1}{B}\|x\|_1 \leq \mathbf{card}(x)$$

for $\|x\|_\infty \leq B$, the constraint

$$\mathbf{1}^T x + \beta G \mathbf{1}^T x + \alpha^T x \leq B$$

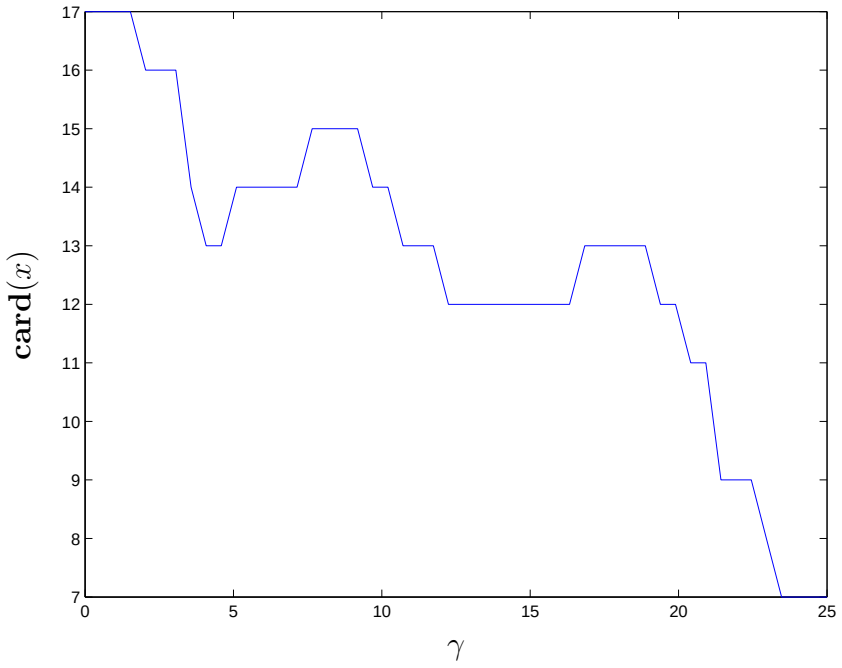


Figure 2: Number of assets invested in versus γ

is a relaxation of

$$\mathbf{1}^T x + \beta\, \mathbf{card}(x) + \alpha^T x \leq B.$$

This implies that the optimal value of (2) for $\gamma = 1/B$ is a lower bound to (1). The following MATLAB script computes this lower bound.

```
Gamma = 1/B;
cvx_begin
    variable x(n)
    minimize quad_form(x,Sigma)
    x>=0
    mu'*x>=Rmin
    sum(x) + Beta*Gamma*sum(x) + Alpha'*x <= B
cvx_end
std_lb1 = sqrt(cvx_optval)
```

The squareroot of the lower bound is found to be 0.1159. So now we know the optimal value of the investment problem (1) is between 0.1159 and 0.1381.

(c) *A more sophisticated lower bound.* Let $x^\star$ denote an optimal point for the original portfolio investment problem (1). Suppose $x_i^\star \leq u_i$ for $i = 1, \ldots, n$. Explain why the optimal value of the problem

$$\begin{array}{ll} \text{minimize} & (x^T \Sigma x)^{1/2} \\ \text{subject to} & \mu^T x \geq R_{\min}, \qquad x \succeq 0 \\ & \mathbf{1}^T x + \beta v^T x + \alpha^T x \leq B, \end{array}$$

where $v_i = 1/u_i$, gives a lower bound on the optimal value of the original portfolio investment problem (1).

A simple choice for u_i is $u_i = B$. Better (*i.e.*, smaller) values can be found as the optimal values of the problems

$$\begin{array}{ll} \text{maximize} & x_i \\ \text{subject to} & \mu^T x \geq R_{\min}, \qquad x \succeq 0 \\ & (x^T \Sigma x)^{1/2} \leq \sigma_{\min} \\ & \mathbf{1}^T x + \beta\tilde{\gamma}\mathbf{1}^T x + \alpha^T x \leq B, \end{array}$$

where $\sigma_{\min}$ is the standard deviation of the portfolio found in part (a). Justify this.

Carry out this procedure for the given problem instance, and compare the resulting lower bound to the results from parts (a) and (b).

Solution.

If $0 \leq x_i \leq u_i$ for $i = 1, \ldots, n$, $\sum_{i=1}^n x_i/u_i$ is a lower bound on $\mathbf{card}(x)$. This implies that the feasible set of (1) is included in the feasible set of

$$\begin{array}{ll} \text{minimize} & (x^T \Sigma x)^{1/2} \\ \text{subject to} & \mu^T x \geq R_{\min}, \qquad x \succeq 0 \\ & \mathbf{1}^T x + \beta v^T x + \alpha^T x \leq B. \end{array} \tag{3}$$

This shows that the optimal value of (3) is a lower bound on the optimal value of (1).

The optimal portfolio investment $x^\star$ is a feasible point of

$$\begin{array}{ll} \text{maximize} & x_i \\ \text{subject to} & \mu^T x \geq R_{\min}, \qquad x \succeq 0 \\ & (x^T \Sigma x)^{1/2} \leq \sigma_{\min} \\ & \mathbf{1}^T x + \beta\tilde{\gamma}\mathbf{1}^T x + \alpha^T x \leq B. \end{array} \tag{4}$$

The optimal value of (4) is therefore a valid upper bound on $x_i^\star$.

The following MATLAB script computes the lower bound presented above.

```
% better lower bound
Gamma = 1/B;
U = [];
for i=1:n
    cvx_begin
        variable x(n)
        maximize x(i)
        quad_form(x,Sigma) <= std_l1^2
        x>=0
        mu'*x>=Rmin
```

```
        sum(x) + Beta*Gamma*sum(x) + Alpha'*x <= B
    cvx_end
    U = [U; cvx_optval];
end
cvx_begin
    variable x(n)
    minimize quad_form(x,Sigma)
    x>=0
    mu'*x>=Rmin
    sum(x) + Beta*(1./U)'*x+ Alpha'*x <= B
cvx_end
std_lb2 = sqrt(cvx_optval)
```

The lower bound returned by this MATLAB script is 0.1254, which is smaller than 0.1381 as expected. Note that this is indeed a better lower bound since the lower bound found in part (b) was equal to 0.1159.