Numerical optimization for large scale problems and stochastic optimization
Constrained optimization

Elisa Salvadori (302630), Sonia Vittone(302673)

Contents

1 Quadratic Programming with Equality Constraints

The general problem of constrained optimization is written as:

$$\min_{x \in X} f(x),$$

where $f : \mathbb{R}^n \to \mathbb{R}$ continously differentiable, and X is a nonempty, convex and closed set.
We focus on the case of $f(x)$ quadratic with equality constraints only, we have:

$$\begin{aligned} &\min\ f(x) \\ &s.t.\ h(x) = 0, \end{aligned}$$

with $h(x)$ linear.
In particular, let us consider a Quadratic Programming problem with equality constraints:

$$\begin{aligned} \min \quad & \frac{1}{2} x^T Q x + c^T x \\ s.t. \quad & Ax = b, \end{aligned} \tag{1}$$

where $Q \in \mathbb{R}^{n \times n}$ is symmetric and positive (at least) semidefinite, $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.
Now we want to apply the KKT conditions to (1), the Lagrangian function is:

$$L(x, \lambda) = \frac{1}{2} x^T Q x + c^T x + \lambda^T (Ax - b).$$

Recalling that if x^* is a local minimum for (1) and $\{\nabla h_i(x^*)\}_i$ are linearly independent vectors, then $\exists \lambda \in \mathbb{R}^m$ such that

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x^*),$$

where λ_i are called Lagrange multipliers, so one necessary condition is that $\{\nabla h_i(x^*)\}_{i=1,..,m}$ are required to be linearly independent, this happens if rows of A are linearly independent that means A is full rank.
The KKT conditions (to be solved w.r.t. (x, λ)) for this kind of problem are the following:

$$\begin{cases} \nabla_x \mathcal{L}(x, \lambda) = Qx + c + A^T \lambda = 0 \\ \nabla_\lambda \mathcal{L}(x, \lambda) = Ax - b = 0 \end{cases} \tag{2}$$

Then we can use a matrix rapresentation of the type:

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix} \qquad \Longrightarrow \qquad K\omega = d, \tag{3}$$

where K is again a symmetric matrix.
Now we can solve this new system considering the full problem or considering a reduce form of it.

2 Methods for the full problem

To solve the KKT conditions we could consider the full problem $K\omega = d$, with K symmetric and attack it:

- LDL^T factorization or LU factorization, but it could be problematic if $n + m$ is very large and K is sparse (fill-in problem);
- GMRES or other iterative methods.

3 Methods for reduced form of the problem

These methods consider reduced forms of $K\omega = d$:

- Schur Complement Resolution (Q non singular);
- Null Space Method.

3.1 Schur Complement Resolution

Starting from (2), from the first equation we obtain:

$$x = -Q^{-1}A^T\lambda - Q^{-1}c,$$

replacing it in the second equation, λ^* is the solution of:

$$\hat{Q}\lambda = -AQ^{-1}c - b,$$

where $\hat{Q} = (AQ^{-1}A^T)$ is the Schur complement (symmetric positive definite, assuming Q non singular and A full rank). Once we get λ^* we find x^*.
This approach is convienent if Q is easy to invert and if m is small w.r.t. n.

3.2 Null Space Method

Assumimg that we know a particular solution $\hat{x}$ to $Ax = b$ and we have a full rank matrix $Z \in \mathbb{R}^{n\times(n-m)}$ such that $AZ = 0_{m\times(n-m)}$ $(dim(Ker(A)) = n - m)$, a general solution to $Ax = b$ can be written as:

$$x = Z\nu + \hat{x}.$$

Replacing this solution into the first equation of (2):

$$Z^TQZ\nu = -Z^Tc - Z^TQ\hat{x},$$

solving the system for ν then we obtain x.

4 Analysis of the problem

Consider the problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \; & \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n-1} x_i x_{i+1} + \sum_{i=1}^{n} x_i \\ s.t. \; & \text{the sum } x_1 + x_{1+k} + x_{1+2k} + ... = 1 \\ & \text{the sum } x_2 + x_{2+k} + x_{2+2k} + ... = 1 \\ & \vdots \\ & \text{the sum } x_k + x_{2k} + x_{3k} + ... = 1. \end{aligned}$$

This is a QP problem with:

- the matrix $Q \in \mathbb{R}^{n\times n}$ is tridiagonal with 2 in the main diagonal and -1 in the upper and in the lower diagonal,

$$Q = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 2 & -1 & 0 \\ 0 & \cdots & 0 & 0 & -1 & 2 & -1 \\ 0 & \cdots & 0 & 0 & 0 & -1 & 2 \end{bmatrix};$$

- the vector $c \in \mathbb{R}^n$ equal to $c = (1, 1, ..., 1)^T$;
- the matrix $A \in \mathbb{R}^{k \times n}$ is formed by all identity matrix of dimension $k \times k$ repeated n/k times,

$$A = \begin{bmatrix} I_{k\times k} & I_{k\times k} & \cdots & I_{k\times k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & & \cdots & & & \ddots & \\ 0 & 0 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix};$$

- and the vector $b \in \mathbb{R}^k$ as follows: $b = (1, 1, ..., 1)^T$.

5 Testing and results

In this section we show our computational results for the problem using full form methods and reduced form methods. In particular we use LU and LDL' decomposition as direct methods and GMRES with tolerance equal to 10^{-12} and 100 iterations as the iterative one.
The problem is also tested with the Schur resolution, thanks to the non singularity of Q, with the inverse matrix (calculated directly using *inv* in MATLAB) and without it, and also with the Null Space method. To test the problem we use combinations of n and k, with $n = 10^4, 10^5$ and $k = 100, 500$.

5.1 Case 1

This case is tested with $n = 10^4$ and $k = 100$. The table shows the value of the function and the *Time* in seconds.

$n = 10^4$, $k = 100$											
LU		LDL'		GMRES		*Schur*		*Schur autoinv*		*Null space*	
$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*
100.00	0.5165	100.00	0.2023	100.00	0.3460	100.00	0.6746	99.99	0.0200	100	0.1737

Table 1: Comparison between the methods.

First we must consider that the iterative method (GMRES) doesn't converge to the desired tolerance $1e-12$, the iterate returned has relative residual $1.2e-06$.
The LDL' decomposition is faster than LU because it exploits the simmetry of K.
In general the reduced forms take less time than the full forms except for the Schur with inverse matrix, the latter takes longer time because it needs to invert the matrix Q that is not easily invertible. Schur autoinv which implements the *backslash* command is more efficient. Furthermore we can easily see that the solution value function found is almost the same for each method.

5.2 Case 2

This case is tested with $n = 10^4$ and $k = 500$. The table shows the value of the function and the *Time* in seconds.

$n = 10^4$, $k = 500$											
LU		LDL'		GMRES		*Schur*		*Schur autoinv*		*Null space*	
$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*
500.00	4.9963	500.00	0.4844	500.00	0.4937	500.00	1.1301	499.99	0.1073	500	0.0640

Table 2: Comparison between the methods.

The table underlines the same behaviour of the previous case, overall the full form methods take longer time than the reduced form methods. We must consider that the iterative method (GMRES) doesn't converge to the desired tolerance $1e-12$, the iterate returned has relative residual $6e-06$.
The LDL' decomposition is faster than LU because it exploits the simmetry of K.
Furthermore we can easily see that the solution value function found is almost the same for each method.

5.3 Case 3

This case is tested with $n = 10^5$ and $k = 100$. The table shows the value of the function and the *Time* in seconds.

$n = 10^5$, $k = 100$											
LU		LDL'		GMRES		*Schur*		*Schur autoinv*		*Null space*	
$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*
100.00	20.4642	100.00	7.6577	100.00	1.8866	Out of memory	/	102.70	0.2223	100	100.8225

Table 3: Comparison between the methods.

For this case the matrix Q has dimension $10^5 \times 10^5$ and MATLAB can't compute the inverse matrix because it's too large, the Schur autoinv works very well in terms of time but has the value function slightly different from others.
In this case the Null space method is the slowest one because of the linear system it has to solve. We must consider that the iterative method (GMRES) doesn't converge to the desired tolerance $1e-12$, the iterate returned has relative residual $3.8e-08$.
The LDL' decomposition is faster than LU because it exploits the simmetry of K.

5.4 Case 4

This case is tested with $n = 10^5$ and $k = 500$. The table shows the value of the function and the *Time* in seconds.

$n = 10^5$, $k = 100$											
LU		LDL'		GMRES		*Schur*		*Schur autoinv*		*Null space*	
$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*	$f(x)$	*Time*
500.00	73.5240	500.00	16.0290	500.00	2.1206	Out of memory	/	502.76	0.6971	500	3.7602

Table 4: Comparison between the methods.

For this case the matrix Q has dimension $10^5 \times 10^5$ and MATLAB can't compute the inverse matrix because it's too large, the Schur autoinv works very well in terms of time but has the value function slightly different from others.
We must consider that the iterative method (GMRES) doesn't converge to the desired tolerance $1e-12$, the iterate returned has relative residual $1.9e-07$.
The LDL' decomposition is faster than LU because it exploits the simmetry of K.

6 Conclusion

The direct methods LU and LDL' are not the best choice because they are affected from the fill-in problem. The figures below show the output of *spy* command for K, L, U matrices for case 1 with the LU decomposition.

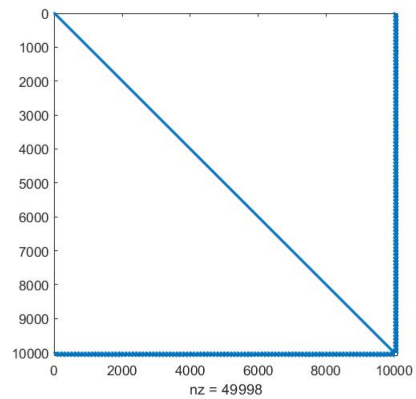

Figure 1: Matrix K for the case with $n = 10^4$ and $k = 100$.

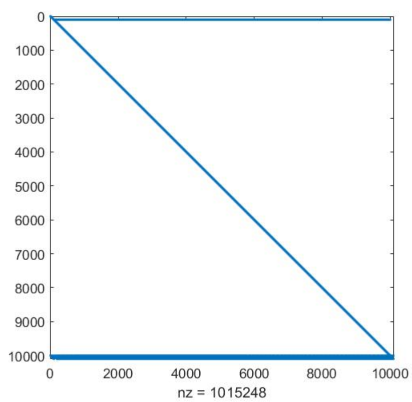

Figure 2: Matrix L for the case with $n = 10^4$ and $k = 100$.

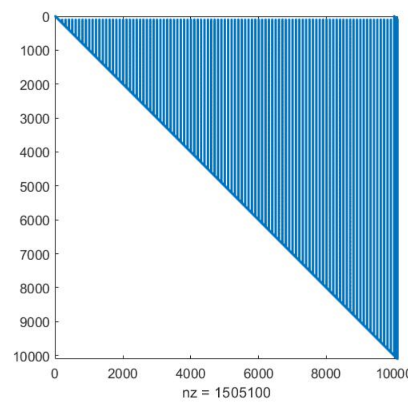

Figure 3: Matrix U for the case with $n = 10^4$ and $k = 100$.

We can observe that the number of nonzero elements of K is 49998 out of $(n+m) \times (n+m) = 102010000$, the percentage is less than 0.05%, the number of nonzero elements of L is 1015248 and of U is 1505100, together almost 2.47%.
The GMRES method is not affected from the fill-in problem but doesn't converge within the maximum number of iteration.
The Schur method needs to compute the inverse matrix Q^{-1} and this computation may lead to problem of storage, we see that with $n = 10^5$ this method doesn't work. On the other hand the Schur method autoinv, which uses the *backslash* operator insted of computing the inverse matrix, works well with all the cases.
The Null Space method has precise solution but when k is too small w.r.t. n the running time is bigger than others.
In conclusion, to solve a quadratic programming problem with equality constraints only, we prefer the reduced form methods for better results.

Appendix

Schur method

```
function [xstar, fxstar, lambda_star] = ...
    QPeq_Schur(Q, Qinv, c, A, b)
% function [xstar, fxstar, lambda_star] = ...
%     QPeq_Schur(Q, Qinv, c, A, b)
% INPUTS:
% Q = Matrix n-by-n of the quadratic loss function
% Qinv = inverse of Q
% c = n-dimensional vector of the quadratic loss function
% A = matrix m-by-n of the equality constraints
% b = m-dimensional vector of the equality constraints
%
% OUTPUTS:
% xstar = solution returned by the function
% fxstar = value of the loss in xstar
% lambda_star = lagrangian multiplier computed by the function
%

% Schur complement computation
S = A * Qinv * A';

% find lambda_star solving the linear system
beta = -b - A * Qinv * c;
lambda_star = S\beta;

% Compute xstar given lambda_star
xstar = Qinv * (-c - A' * lambda_star);

% compute fxstar
fxstar = 0.5 * xstar' * Q * xstar + c' * xstar;

end
```

Schur autoinv method

```
function [xstar, fxstar, lambda_star] = ...
    QPeq_Schur_autoinv(Q, c, A, b)
% function [xstar, fxstar, lambda_star] = ...
%     QPeq_Schur_autoinv(Q, c, A, b)
% INPUTS:
% Q = Matrix n-by-n of the quadratic loss function
% c = n-dimensional vector of the quadratic loss function
% A = matrix m-by-n of the equality constraints
% b = m-dimensional vector of the equality constraints
%
% OUTPUTS:
% xstar = solution returned by the function
% fxstar = value of the loss in xstar
% lambda_star = lagrangian multiplier computed by the function
%

% DON'T USE inv(Q)

% Schur complement computation
S = A * (Q\A');
```

```

% find lambda_star solving the linear system
beta = -b - A * (Q\c);
lambda_star = S\beta;

% Compute xstar given lambda_star
xstar = Q\(-c - A' * lambda_star);

% compute fxstar
fxstar = 0.5 * xstar' * Q * xstar + c' * xstar;


end
```

Null Space method

```
function [xstar, fxstar, lambda_star, v_star] = ...
    QPeq_Null(Q, c, A, b, x2)
% function [xstar, fxstar, lambda_star, v_star] = ...
%     QPeq_Null(Q, c, A, b, x2)
% INPUTS:
% Q = Matrix n-by-n of the quadratic loss function
% c = n-dimensional vector of the quadratic loss function
% A = matrix m-by-n of the equality constraints
% b = m-dimensional vector of the equality constraints
% x2 = (n-m) column vector
%
% OUTPUTS:
% xstar = solution returned by the function
% fxstar = value of the loss in xstar
% lambda_star = lagrangian multiplier computed by the function
% v_star = solution of linear system


% Z initialization
[m, n] = size(A);
A1 = A(:, 1:m);
A2 = A(:, m+1:end);
Z = [-A1\A2; speye(n-m)];

% xhat initialization
xhat = [A1\(b - A2 * x2); x2];

% compute v_star as solution of:
% (Z' Q Z)v = -Z'(Q xhat + c )
V = Z' * Q * Z;
beta = -Z' * (Q * xhat + c);
v_star = V\beta;

% compute xstar, given v_star and xhat
xstar = Z * v_star + xhat;

% compute lambda_star given xstar
lambda_star = (A * A')\(-A *(c + Q * xstar));

% compute fxstar
fxstar = 0.5 * xstar' * Q * xstar + c' * xstar;
```

```

end
```

LU,LDL',GMRES methods

```
%% LU
tic
%LU decomposition
[L1, U] = lu(K);
%solve Kw = d
wLU = U\(L1\d);
%recall that w = [x;lambda]
xLU = wLU(1:n);
lambdaLU = wLU(n+1:end);
fxLU = 0.5*xLU'*Q*xLU + c'*xLU
toc


%% LDL'
tic
%LDL decomposition
[L2,D,P,S] = ldl(K);
%solve Kw = d
wLDL = S*P*((L2*D*L2')\(P'*S*d));

xLDL = wLDL(1:n);
lambdaLDL = wLDL(n+1:end);
fxLU = 0.5*xLDL'*Q*xLDL + c'*xLDL
toc


%% GMRES
tic
tol = 1e-12;
imax = 100; %number of max iterations

%solve Kw = d
wGM = gmres(K, d, [], tol, imax);

xGM = wGM(1:n);
lambdaGM = wGM(n+1:end);
fxGM = 0.5*xGM'*Q*xGM + c'*xGM
toc
```

Initialization

```
n = 10000;
k = 100;
c = ones (n,1) ;
b = ones (k,1);
Q = spdiags ([ -c 2*c -c ] , -1:1 ,n, n );
A = repmat (speye(k),1,n/k) ;

[m,n] = size(A);


O = zeros(m, m);
K = [Q A';
```

```
    A O];
d = [-c; b];

Qinv = inv(Q);
```

Testing example

```
%Schur with inverse

disp('**** SCHUR (with Qinv) *****')
tic
[xstar, fxstar, lambda_star] = ...
    QPeq_Schur(Q, Qinv, c, A, b);
toc
disp('***********************************')
disp(['xstar: ', mat2str(xstar), ';'])
disp(['f(xstar): ', num2str(fxstar), ';'])
disp(['lambda_star: ', mat2str(lambda_star),';'])
disp('***********************************')

%Schur without inverse

disp('**** SCHUR (without Qinv) *****')
tic
[xstar, fxstar, lambda_star] = ...
    QPeq_Schur_autoinv(Q, c, A, b);
toc
disp('***********************************')
disp(['xstar: ', mat2str(xstar), ';'])
disp(['f(xstar): ', num2str(fxstar), ';'])
disp(['lambda_star: ', mat2str(lambda_star),';'])
disp('***********************************')

%Null space

disp('**** NULL SPACE *****')
tic
[xstar, fxstar, lambda_star, v_star] = ...
    QPeq_Null(Q, c, A, b, zeros(n-m, 1));
toc
disp('***********************************')
disp(['xstar: ', mat2str(xstar), ';'])
disp(['f(xstar): ', num2str(fxstar), ';'])
disp(['lambda_star: ', mat2str(lambda_star),';'])
disp(['v_star: ', mat2str(v_star),';'])
disp('***********************************')
```