Project Non-ML 33

Gioele Bertoncini

23/04/2023

1 Problem

This report analizes the convex quadratic nonseparable knapsack problem, defined as follows:

$$(P) \quad f(x) \triangleq \min x^T Q x + qx \tag{1.1}$$

$$\textit{subject to} \ \sum_i a_i x_i \leq b, \tag{1.2}$$

$$l \leq x \leq u \tag{1.3}$$

where $x \in \mathbb{R}^n$, q, $0 \leq l < u$ and $a > 0$ are fixed vectors of $\mathbb{R}^n$, b is a fixed positive real number, and Q is a $n \times n$ positive semidefinite matrix. If all the bounds are finite w.l.o.g we can assume that $l = 0$, so the box constraint 1.3 can be rewritten as:

$$0 \leq x \leq u \tag{1.4}$$

In particular we aim to solve (P) with an active-set strategy.

2 Active-set method

The active set $A(x)$ is the set of constraints that are satisfied with equality at a given point x. The problem is subject to box constraints 1.4, so in this case the active-set partition the variables in three sets $x = [x_L, x_F, x_U]$ where only the x_F variables are free to change: $x = [0, x_F, u_U]$. Therefore the problem can be rewritten as an unconstrained reduced problem in a smaller space defined by the active box constraints:

$$f_F(x_F) \triangleq \min x_F^T Q_{FF} x_F + (q_F + u_U^T Q_{UF}) x_F + u_U^T Q_{UU} u_U + q_U u_U \tag{2.1}$$

The knapsack constraint ($\sum_i a_i x_i \leq b$) can also be part of the active-set, in that case the reduced problem 2.1 is subject to the following equality constraint:

$$(h_{KN}) \qquad a_F x_F = b - a_U u_U \tag{2.2}$$

2.1 KKT conditions

The active-set algorithm builds at each iteration a set of constraints $A(x)$ then a sub-problem subject only to the active-set constraints $h_i(x)\ \forall i \in A(x)$ as equalities, is solved. The optimum point found at each iteration must satisfy the following KKT conditions for the equality-constrained sub-problem:

$$h_i(x) = 0 \quad \forall i \in A(x) \qquad \textit{(KKT-F)} \tag{2.3}$$

$$\nabla f(x) + \sum_{i \in A(x)} \mu_i \nabla h_i(x) = 0 \qquad \textit{(KKT-G)} \tag{2.4}$$

Since the problem is subject to box constraints, at each iteration the algorithm will solve the reduced problem on the free variables 2.1. There are two possible cases: if the knapsack constraint h_{KN} does not belong to the active-set the reduced problem has no constraints, therefore the optimality condition for the reduced problem is just the stationarity of x_F: $\nabla f(x_F) = 0$. Thus an optimal solution is obtained solving the system:

$$Q_{FF} x_F = -(q_F + u_U^T Q_{UF})$$

Otherwise, if the active-set contains the knapsack constraint the KKT conditions became the following:

$$a_F x_F = b - a_U x_U \qquad \textit{(KKT-F)} \tag{2.5}$$

$$\nabla f_F(x) + \mu_{KN} a_F = 0 \qquad \textit{(KKT-G)} \tag{2.6}$$

where μ_{KN} is the Lagrangian multiplier associated to the reduced knapsack constraint h_{KN} (2.2). The optimum of the reduced problem x_{F*} can be found solving the KKT system, formed by the conditions 2.4 and 2.3:

$$\begin{aligned} &Q_{FF} x_F + q_F + u_U^T Q_{UF} + a_F \mu_{KN} = 0 && \textit{(KKT-G)} \\ &a_F x_F = b - a_U u_U && \textit{(KKT-F)} \end{aligned}$$

$$\begin{bmatrix} Q_{FF} & a_F^T \\ a_F & 0 \end{bmatrix} \begin{bmatrix} x_{F*} \\ \mu_{KN} \end{bmatrix} = \begin{bmatrix} -(q_F + u_U^T Q_{UF}) \\ b - a_U u_U \end{bmatrix} \tag{2.7}$$

The only multiplier explicitly computed in this phase is the one associated to h_{KN}, the multipliers of the active box constraints μ_i (one for each variable x_i since the sets L and U are mutually exclusive) can be derived from the stationarity condition 2.4. Each variable x_i in the active-set is associated with a box constraint h_i:

$$h_i(x_i) = \begin{cases} -x_i \leq 0 & \text{if } i \in L \\ x_i \leq u_i & \text{if } i \in U \end{cases}$$

so the condition 2.4 became:

$$\begin{cases} g_i - \mu_i = 0 & \text{if } i \in L \\ g_i + \mu_i = 0 & \text{if } i \in U \end{cases} \qquad (KKT - G) \tag{2.8}$$

where g_i is the i-th entry of the gradient vector $g = \nabla f(x)$. There μ_i can be computed as:

$$\mu_i = \begin{cases} g_i & \text{if } i \in L \\ -g_i & \text{if } i \in U \end{cases} \tag{2.9}$$

If the h_{KN} is active, the stationarity condition become:

$$\begin{cases} g_i + \mu_{KN} a_i - \mu_i = 0 & \text{if } i \in L \\ g_i + \mu_{KN} a_i + \mu_i = 0 & \text{if } i \in U \end{cases} \qquad (KKT - G) \tag{2.10}$$

therefore the μ_i can be computed as:

$$\mu_i = \begin{cases} g_i + \mu_{KN} a_i & \text{if } i \in L \\ -g_i - \mu_{KN} a_i & \text{if } i \in U \end{cases} \tag{2.11}$$

2.2 Solving the KKT system

If the matrix

$$K = \begin{bmatrix} Q_{FF} & a_F^T \\ a_F & 0 \end{bmatrix} \tag{2.12}$$

is non-singular the KKT system have a unique solution x_F. Since K is symmetric it can be factorized in a lower triangular matrix L and a diagonal matrix D using the LDL factorization. Then a solution $[x_F \quad \mu_{KN}]$ can be computed by forward and backward substitution with the following algorithm:

1: **procedure** SOLVE_LDL(K, b)
2: $[L, D]$ = LDL_factorization(K)
3: y = forward_substitution(L, b)
4: z= substitution(D, y)
5: x_F = backward_substitution(L^T, z)
6: **return** x_F

However the LDL factorization is not numerically stable, since it is possible to find small elements on the main diagonal of K that are close to machine precision. A more stable algorithm can be obtained by using symmetric pivoting via exchanging rows and corresponding columns, but there are still cases in which a small element on the diagonal can't be avoided, such as the following matrix:

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \end{bmatrix}$$

To overcome this issue the Bunch-Parlett algorithm [1] uses both 1x1 and 2x2 blocks pivots. This factorization produces a lower triangular matrix L with ones on the diagonal, a block diagonal matrix D and a permutation matrix P such that:

$$PKP^T = LDL^T$$

Given this factorization a solution for a symmetric non-singular system can be computed with the following algorithm:

1: **procedure** SOLVE_BP(K, b)
2: $[L, D, P]$= Bunch-Parlett_factorization(K)
3: y = forward_substitution($L, P \cdot b$)
4: z = block_diagonal_substitution(D, y)
5: x_F = backward_substitution(L^T, z)
6: **return** $P^T \cdot x_F$

2.3 Singular KKT matrix

Since Q is positive semidefinite, the matrix K may possibly be singular and therefore may have no solution. Depending on the number of zero eigeinvalues of $Q_{FF} \in \mathbb{R}^{m \times m}$ we distinguish two cases:

- Case 0: Q_{FF} is singular and the knapsack constraint is not active. In this case $K = Q_{FF}$ and the system is singular.
- Case 1: Q_{FF} has only one zero eigenvalue and the knapsack constraint is active. In this case if $a^T v = 0$, where $v \in \mathbb{R}^m$ is the kernel vector of Q_{FF} such that $Q_{FF}v = 0$, the system will be singular.

  Proof. The KKT matrix $K \in \mathbb{R}^{(m+1)\times(m+1)}$ 2.12 is singular iff it has at least one zero eigenvalue λ, that satisfies the equations: $Ax = \lambda x = 0 \cdot x = 0$, this is equivalent to say that $\exists r \in Ker(K)$ with $r \neq 0$. A vector $r \in \mathbb{R}^{m+1}$ can be constructed as $r = [v \quad 0]$ where v is the kernel vector of Q_{FF}. We can verify that $Kr = 0$ separating the first m equations from the last one:

  - $Q_{FF}v + a \cdot 0 = 0$ is true because $v \in Ker(Q_{FF})$.
  - $a^T v + 0 \cdot 0 = 0$ is true for hypothesis.

  □
- Case 2: If Q_{FF} has two or more zero eigenvalues. In this case, the KKT system is always singular.

  Proof. In this case, we can always construct a vector $r = [t \quad 0] \in Ker(K)$ such as: $t = c_1 v + c_2 w$ and where $v, w \in Ker(Q_{FF})$ also when $a^T v \neq 0$ and $a^T w \neq 0$.

  - $Q_{FF}t + a \cdot 0 = 0$ is true because t belongs to the null space of Q_{FF}.
  - $a^T t + 0 \cdot 0 = 0 \implies a^t(c_1 v + c_2 w) = 0 \implies c_1 a^T v + c_2 a^T w = 0$ is true if we chose $c_1 = -c_2 \frac{a^T w}{a^T v}$.

  □

2.3.1 Infinite descent direction

Since when the KKT system is singular a unique solution can't be found, then an infinite descent direction d such that $Qd = 0$ and $ad = 0$ is computed. Starting from a symmetric decomposition of a singular matrix K, the size of $Ker(K)$ can be determined by counting the zero elements on the diagonal matrix D. The decomposition of a singular matrix $K \in \mathbb{R}^{n\times n}$ will be

$$K = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I^k \end{bmatrix} \begin{bmatrix} D_{11} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & I^k \end{bmatrix}$$

where I^k is an identity block of size k, $D_{11} \in \mathbb{R}^{n-k\times n-k}$ is a diagonal block (or possibly block diagonal if 2x2 pivots are used), $L_{11} \in \mathbb{R}^{n-k\times n-k}$ is a lower triangular block with ones on the diagonal and $L_{21} \in \mathbb{R}^{k\times n-k}$ is a dense block. As discussed in Section 2.3 it is sufficient to find a single kernel vector $r \in Ker(K)$ to determine a vector $v \in$

$Ker(Q_{FF})$, therefore there is no need to compute the whole factorization, and we can stop the process when the first zero on the diagonal is encountered, obtaining a partial factorization:

$$K_s = \underbrace{\begin{bmatrix} L_{11} & 0 \\ l_{21} & 1 \end{bmatrix}}_{L_s} \underbrace{\begin{bmatrix} D_{11} & 0 \\ 0 & 0 \end{bmatrix}}_{D_s} \underbrace{\begin{bmatrix} L_{11}^T & l_{21} \\ 0 & 1 \end{bmatrix}}_{L_s^T}$$

From this factorization the kernel vector $r = [r_s \quad 0] \in \mathbb{R}^n$ can be computed solving the following system by backward substitution:

$$L_s^T r_s = i^k$$

where $r_s \in \mathbb{R}^{n-k+1}$ and i^k is the last column of the identity matrix $I \in \mathbb{R}^{n-k+1}$.

Note that if a factorization with pivoting is used (e.g. Bunch-Parlett) the vector r needs to be changed using the full permutation matrix $P \in \mathbb{R}^n$:

$$r \leftarrow P^T r$$

At this point if we are in Case 0, then $v = r \in Ker(Q_{FF})$, otherwise a kernel vector for Q_{FF} is obtained removing the last element from r: $v = r_{1:n-1} \in Ker(Q_{FF})$.

From the kernel vector v an infinite descent direction d is determined depending on the the sign of it's product with the gradient in the free variables:

$$d = \begin{cases} v & \text{if } g_F v < 0 \\ -v & \text{if } g_F v > 0 \end{cases}$$

where $g_F = \{g_i : i \in \neg L \cap \neg U\}$.

3 Structure of the algorithm

1: **procedure** ASKQP(Q, q, a, b, l, u)
2: $x \leftarrow$ initial feasible solution;
3: $L, U, F, k \leftarrow$ compute active-set;
4: **while** (true) **do**
5: $K, qb \leftarrow$ construct reduced KKT system
6: $Lt, D, P, s \leftarrow$ stop_Bunch_Parlett_algorithm(K)
7: **if** s **then**
8: $d_F \leftarrow$ compute infinite descent direction
9: $d \leftarrow [0, d_F, 0]$
10: **else**
11: $[x_F, \mu_{KN}] \leftarrow$ solve_BP(K, qb)
12: $x_* = [0, x_F, u_U]$
13: $\mu_i =$2.9 or 2.11
14: **if** $l \leq x_* \leq u \wedge ax_{F*} \leq b$ **then**
15: **if** $\mu_i \geq 0 \wedge \mu_{KN} \geq 0$ **then**
16: **return** x_* ▷ Optimal
17: **else**
18: $h \leftarrow \min\{i \notin F : \mu_i < 0\}$
19: $L, U, F, k \leftarrow$ remove h from active-set
20: **continue**
21: $d \leftarrow x_* - x$
22: $\alpha_U = \min\{\alpha_i = (u_i - x_i)/d_i : i \in F, d_i > 0\}$;
23: $\alpha_L = \min\{\alpha_i = (-x_i)/d_i : i \in F, d_i < 0\}$;
24: **if** $\neg k \wedge a_F d_F > 0$ **then**
25: $\alpha_k = (b - a_F x_F)/a_F d_F$;
26: $\bar{\alpha} \leftarrow \min\{\alpha_L, \alpha_U, \alpha_k\}$;
27: **else**
28: $\bar{\alpha} \leftarrow \min\{\alpha_L, \alpha_U\}$;
29: $x \leftarrow x + \bar{\alpha} d$;
30: $L, U, F, k \leftarrow$ compute active-set;

In the algorithm described above the active-set (line 3, 19, 30) is represented by a flag k that is true when the knapsack constraint is active and three masks for the variables L, U, F that select respectively the variables stuck to the lower bound, the variables stuck to the upper bound and the free variables.

The knapsack constraint is mostly handled inside the sub-procedures, in particular at line 5, the resulting K and qb will be respectively equal to Q_{FF} and $-(q_F + u_U^T Q_{UF})$ when the knapsack constraint is not active (as discussed in 2.1).

At line 6, if a null element on the diagonal is encountered, a partial factorization $[L_s, D_s, P]$ (as described in Section 2.3.1) is returned and the flag s is set to true. The infinite descent direction at line 8 is computed in two different ways whether the

knapsack constraint belongs to the active set or not, as discussed in 2.3.1.

At line 18, h represent the index of a generic active constraint to be removed, that can be the knapsack or a box one.

4 Experiments

In this section are showed the results of some experiments performed with our algorithm on different variation of our quadratic problem. The instances of the problem used to conduct the experiments illustrated in this section are generated using the *genBCQP.m* script with parameters $actv = 0.6$, $ecc = 0.6$, and for the knapsack constraint $a = [1 \dots 1]$ and $b = 1000$. Any parameter different from those default values is specified. The singular matrices $Q \in \mathbb{R}^{n \times n}$ with z zero eigenvalues, is generated as $Q = AA^T$ where $A \in \mathbb{R}^{n \times n-z}$ is a random matrix.

The major bottleneck of the algorithm is the factorization of the matrix K, that can become very slow when n is big. Two different starting point of the algorithm has been analysed, ASKP_F uses a point $x = u/2^i$ where i is the smallest such that $a^T x < b$ with no active constraints. ASKP_L starts from $x = 0$, where all the lower bound constraints are active. In Table 1 is showed the convergence time of our algorithm, with every function computed from scratch with two different starting point (ASKP_F and ASKP_L), and using matlab functions 'ldl' and 'null', with no active constraint at the starting point (ASKP_m), compared with quadprog. As can be seen in Table 1 the

	ASKP_F	**ASKP_m**	**ASKP_L**	**quadprog**
n=100	0.52s	0.07s	0.25s	0.02s
n=500	78.64s	7.55s	3.94s	0.12s
n=1000	1255.89s	23.78s	8.83s	0.37s

Table 1: Total computation time of different strategies on a non-singular problem.

ASKP_L algorithm is faster than the other implementations. As shown in Table 2 it is faster in particular also when the knapsack constraint is very tight and a large number of variables are stuck at the lower bound at the optimum.

	b=10	**b=1000**
n=100	0.03s, 28 iter.	0.23s, 161 iter.
n=100, K singular	0.71s, 224 iter.	1.13s, 307 iter.
n=500	0.03s, 29 iter.	6.40s, 546 iter.
n=500, K singular	8.99s, 722 iter.	233.95s, 3048 iter.

Table 2: Total computation time and number of iterations of ASKP_L with tight or large knapsack constraint.

In Figure 1 is showed the number of iterations of ASKP_m and ASKP_L, changing the parameter *actv* that controls the percentage of active box constraints at the

optimum. Although the number of iterations for ASKP_L remains high, the results in Table 3 shows that the more the number of active constraints at the optimum is high, the more ASKP_L is fast.

	ASKP_F	**ASKP_m**	**ASKP_L**
actv=0.1	26.80s, 55 iter.	0.73s, 55 iter	65.72s, 565 iter.
actv=0.5	73.37s, 260 iter.	2.08s, 260 iter.	12.64s, 534 iter.
actv=0.9	82.80s, 461 iter.	2.69s, 461 iter.	1.53s, 503 iter.

Table 3: Total computation time and number of iteration of the different versions of the algorithm, with different % of active box constraints at the optimum and $n = 500, b = 10000, ecc = 0.1$.

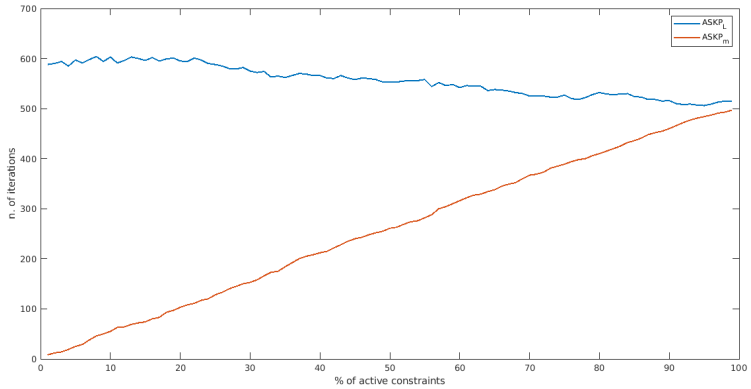

Figure 1: Number of iterations to reach the optimum changing the % of active box constraint at the optimum on a problem with n=500, b=10000, ecc=0.1, using ASKP_L and ASKP_m.

In Figure 2 and Figure 3 are showed respectively the convergence plot of ASKP_m with a non-singular starting Q and a singular starting Q with 95% zero eigenvalues. Although with different computing times, both ASKP_m and ASKP_L have the same convergence behaviours.

In Figure 4 and Figure 6 it can be seen that the more the eccentricity of the matrix Q is high and the more zero eigenvalues has the matrix Q if it's singular, the greater is the number of iterations that the algorithm needs to find the optimal active-set. In Table 4 are showed how the total computation time and the number of iterations of ASKP_L varies when the eccentricity of the matrix Q changes, it can be seen that the results follows the same trend illustrated in Figure 5 also for different sizes of the problem.

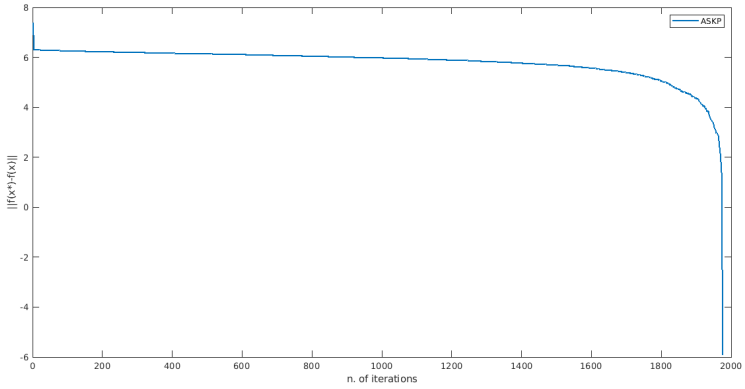

Figure 2: Convergence plot of ASKP_m with n=2000 and a non-singular matrix Q.

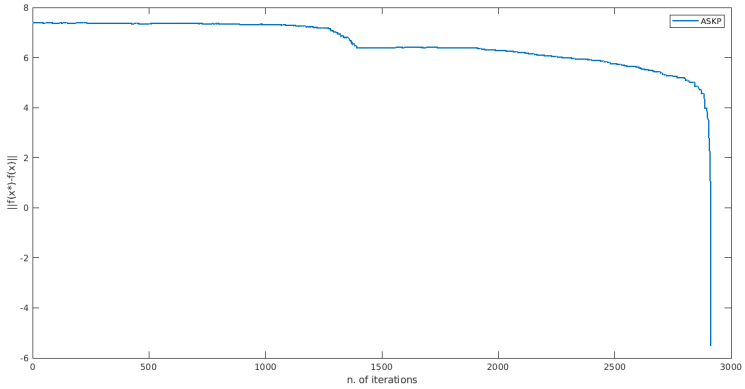

Figure 3: Convergence plot of ASKP_m with n=2000 and a singular matrix Q, with 90% zero eigenvalues.

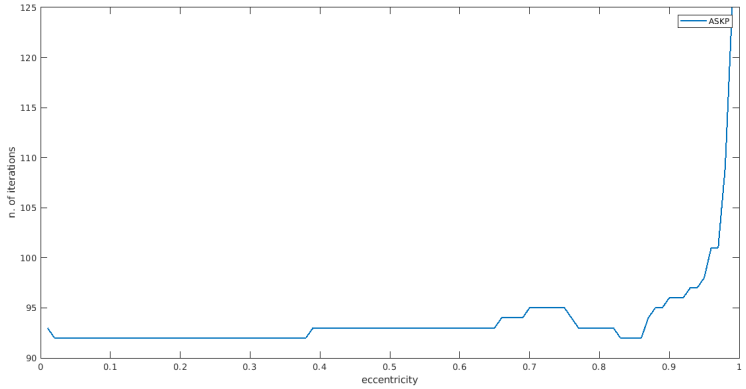

Figure 4: Number of iterations to reach the optimum changing the eccentricity of the matrix Q on a problem with n=100, using ASKP_m.

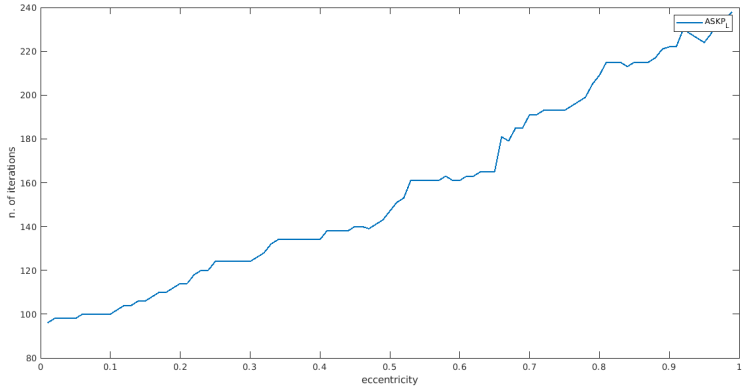

Figure 5: Number of iterations to reach the optimum changing the eccentricity of the matrix Q on a problem with n=100, using ASKP_L.

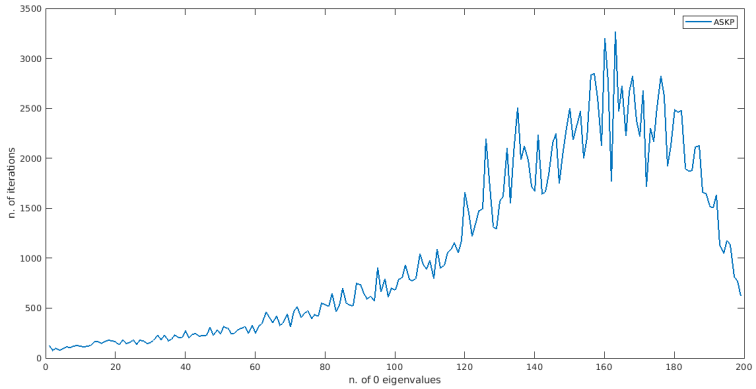

Figure 6: Number of iterations to reach the optimum changing the number of 0 eigenvalues of the singular matrix Q, with n=200, using ASKP_m.

	ecc=0.100	**ecc=0.500**	**ecc=0.999**
n=100	0.25s, 100 iter.	0.25s, 147 iter.	0.45s, 273 iter.
n=500	6.67s, 532 iter.	6.27s, 794 iter.	20.52s, 1431 iter.
n=1000	83.62s, 1096 iter.	61.26s, 1651 iter.	189.82s, 2970 iter.

Table 4: Total computation time and number of iterations of ASKP_L with different eccentricity of the non-singular matrix Q and $b = 10000$.

References

[1] Beresford N. Parlett James R. Bunch, Linda Kaufman. Decomposition of a symmetric matrix, 1976. doi:10.1007/BF01399088.

A Factorization and kernel plots

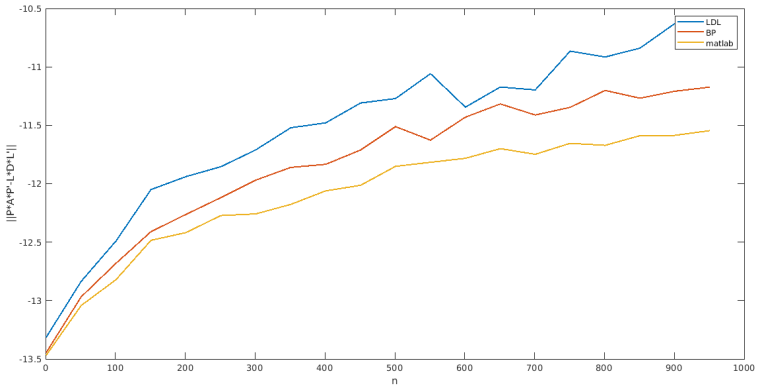

Figure 7: Comparison of accuracy of the factorization of an indefinite symmetric matrix $A \in \mathbb{R}^{n \times n}$, computed using LDL factorization with pivoting, Bunch-Parnell algorithm and matlab 'ldl' function, on a logarithmic scale.

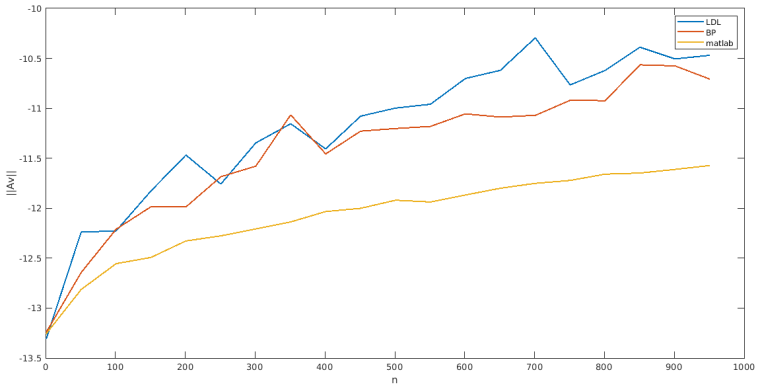

Figure 8: Comparison of accuracy of the first kernel vector v of a singular positive semi-defininte matrix $A \in \mathbb{R}^{n \times n}$ with one zero eigenvalue, computed using LDL factorization, Bunch-Parnell algorithm and matlab 'null' function on a logarithmic scale.

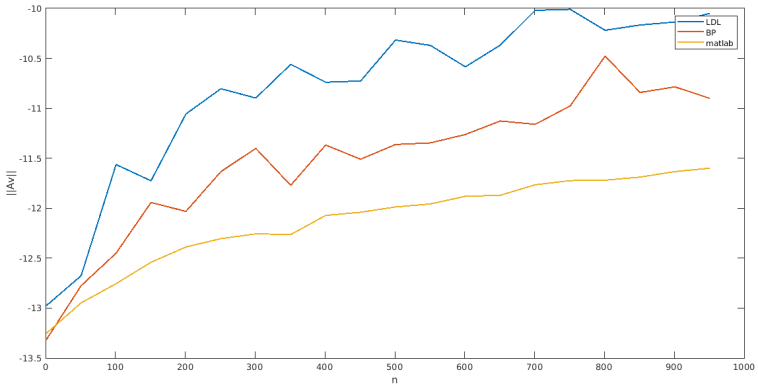

Figure 9: Comparison of accuracy of the first kernel vector v of a singular positive semi-defininte matrix $A \in \mathbb{R}^{n \times n}$ with 10% zero eigenvalues, computed using LDL factorization, Bunch-Parnell algorithm and matlab 'null' function on a logarithmic scale.

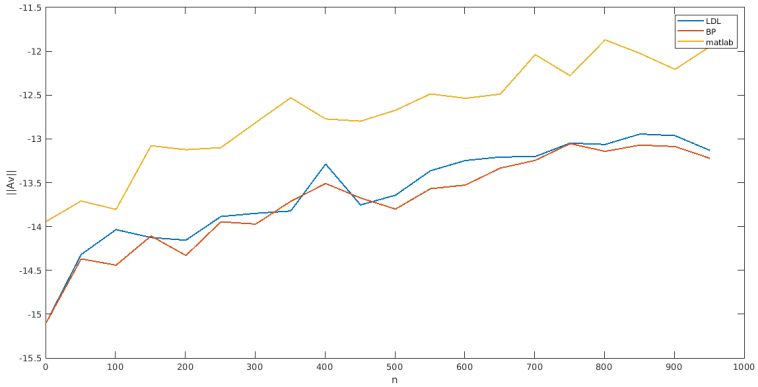

Figure 10: Comparison of accuracy of the first kernel vector v of a singular positive semi-defininte matrix $A \in \mathbb{R}^{n \times n}$ with 90% zero eigenvalues, computed using LDL factorization, Bunch-Parnell algorithm and matlab 'null' function on a logarithmic scale.