Computational Mathematics for Learning and Data Analysis

Non_ML project number 21, 2021/22

Edoardo Federici 616667, Carlo Tosoni 644824

January 2022

1 Introduction

This report describes our work on the assigned final project for the Computational Mathematics for Learning and Data Analysis course (academic year 2021/2022).

2 Framing the Problem

Given P a sparse linear system of the form

$$\begin{bmatrix} D & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$$

Where D is a diagonal positive-definite matrix, hence its eigenvalues are strictly greater than 0, and E is the edge-node adjacency matrix of a given directed graph. These problems arise as the KKT system of the convex quadratic separable Min-Cost Flow problem.

The project consists in the development of two algorithms and in their analysis, they might be denominated in this relation as **(A1)** and **(A2)**.

(A1) consists in the application of the conjugate gradient for linear systems in the reduced linear system. Indeed, the second equation states $Ex = c$ and therefore it is possible to reduce the system by eliminating the variable x, thus obtaining $(ED^{-1}E^T)y = ED^{-1}b - c$. The algorithm should rely on a callback function that computes the product $ED^{-1}E^Tv$ for a given vector v, without computing explicitly the matrix $ED^{-1}E^T$.

(A2) The second algorithm consists of applying GMRES/MINRES to the original system.

Off-the-shelf solvers are not allowed. In the first part of the relation there is an introduction to the topics covered in the project.

3 GMRES and MINRES

The Generalized Minimal Residual Method is an iterative method for the analytical solution of a system of liner equations. It involves the computation of an approximation of the solution vector in the Krylov subspace with minimal residual.

3.1 Iterative algorithms

In linear algebra the importance of iterative methods derives fro the fact that direct algorithms have a complexity equal to $O(m^3)$, where $Ax = b$ with $A \in R^{m \times n}$ and $b \in R^{n \times 1}$. This complexity becomes too large when m is big, hence iterative methods are able to cut the $O(m^3)$ bottleneck of direct methods to $O(m^2)$. However this result can be achieved for some matrices, but not for others.

3.2 Sparsity

A possible structure that a matrix might have is sparsity, i.e. preponderance of zero entries. In many applications of real life there could be matrices where only few element for each row are different from zero.

The sparsity structure of matrices can be exploited by iterative methods; these algorithms use a matrix in the form of a black box. Iterative algorithms require nothing more than the ability to determine Ax for any x, this calculation is computed by a black box (or oracle) and there is no need to worry about how these methods work. In other words, we have to focus only on designing the iterative methods, without worrying about oracles' functioning. Sparse matrices can be stored through a list of nonzero entries, in the format (i, j, A_{ij}), below there is the pseudocode of a function able to compute the product matrix-vector directly in this format.

Algorithm 1 Function w = compute_product(A, v):

```
w ← zeros(size(A, 1)) //create an empty vector of dimension m
for (i, j, A_ij) in A : do
    w[i] += A_ij * v[j] //update the vector w
end for
return w
```

3.3 Krylov Space

Given a matrix A and a vector b, the associated Krylov sequence is the set of vectors $b, Ab, A^2b, ...$, which can be computed by an oracle (black box) function in the form $b, Ab, A(Ab), ...$ The corresponding Krylov subspaces are the spaces spanned by successively larger groups of these vectors.
Given a nonsingular $A \in C^{N\times N}$ and $b \neq 0$, the n^{th} Krylov (sub)space $K_n(A, b)$ generated by A from b is

$$K_n(A, b) = span(b, Ab, A^2b, ..., A^{n-1}b)$$

It is clear that every vector in $K_n(A, b)$ can be uniquely expressed by the linear combination of the basis $\{b, Ab, ..., A^{n-1}b\}$ and that $K_1 \subseteq K_2 \subseteq ... \subseteq K_n - 1$.

Starting from $x_0 = 0$, iterative algorithms compute at each k-th step the vector x_k, which belongs to the order k-th Krylov subspace, i.e. $x_k \in span(b, Ab, A^2b, ..., A^{k-1}b)$. The sequence of vectors x_0, x_1, x_2, ..., x_n converges to the solution of the linear system $Ax = b$. Indeed in the vector space $K_n(A, b)$ we can find n scalars α_0, α_1, ..., α_{n-1} such that the vector $x_n = \alpha_0 b$, $\alpha_1 Ab$, ..., $\alpha_{n-1}A^{n-1}b$ is the minimum of the function

$$\|Ax - b\| \quad with\, x \in K_n(A, b)$$

or equivalently we can find the vector $y \in R^n$ that minimizes the function

$$\|AVy - b\| \quad with\, V = [b|Ab|...|A^{n-1}b]$$

3.4 Arnoldi

A problem related to Krylov subspaces is that the columns of the the matrix $V = [b|Ab|...|A^{n-1}b]$ tend to be multiples of the same vectors as n increases. To avoid this issue, the Arnoldi algorithm is used to compute an orthonormal basis of the krylan subspace $K_n(A, b)$. At each step, Arnoldi computes the vector q_j given the vectors q_1, q_2, ..., q_{j-1} such that q_1, q_2, ..., q_{j-1}, q_j is an orthonormal basis of $K_j(A, b)$, where $q_j \in K_j(A, b)\backslash K_{j-1}(A, b)$.

The algorithm operates in the following way:

Base step: Firstly, we have to find an orthonormal basis of $span(b)$, i.e. $K_1(A, b)$. This is actually easy, indeed $q_0 = b/\|b\|$

Generic step ($j \to j + 1$): computing $q_{j+1} \in K_{j+1}(A, b)\backslash K_j(A, b)$,

1) We compute the vector w.
$w = Aq_j = \beta_1 q_1 + \beta_2 q_2 + ... + \beta_{j+1} q_{j+1}$ ($w \in K_{j+1}(A, b)$ but $w \notin K_j(A, b)$).
All the vectors q_1, q_2, ..., q_j are known, in fact only the vector q_{j+1} is unknown in the sequence used to compute w.

2) substract the q_1 component exploiting orthonormality.
$q_1^T w = q_1^T q_1 \beta_1 + q_1^T q_2 \beta_2 + ... + q_1^T q_{j+1} \beta_{j+1}$

we know that $q_i^T q_j = 0$ if $i \neq j$ or $= 1$ if $i = j$.
hence $q_1^T w = \beta_1$.
so we can update w in this way:
$w \leftarrow w - q_1(q_1^T w) = \beta_2 q_2 \ + \ ... \ + \beta_{j+1} q_{j+1}$
3) Repeat the previous step until $w = \beta_{j+1} q_{j+1}$.
At this point it is actually easy to compute the vector q_{j+1}, indeed $q_{i+1} = w/\|w\|$.

Algorithm 2 Arnoldi's pseudocode

```
Q = zeros(len(b), n) //initialization of the matrix Q
Q(:, 1) ← b/‖b‖ //first vector of the Krylov subspace
for j = 1 : n do
    w = A * Q(:, j) //compute the vector w
    for i = 1 : j do
        H(i, j) = Q(:, i)′ * w //computing the coefficient β_{i,j}
        w = w − Q(:, i) * H(i, j) //updating the vector w
    end for
    H(j + 1, j) = ‖w‖ //computing the coefficient β_{j+1,j}
    Q(:, j + 1) = w/H(j + 1, j)
end for
return [Q, H]
```

Factorization
It is possible to summarize what we have done in the previous algorithm by this formula:

$$AQ_j = Q_{j+1} H_j$$

Where Q_{j+1} is the matrix $[q_1|q_2|...|q_{j+1}]$ and the matrix H_j is formed by the coefficients previously computed, hence:

$$H_j = \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} & \beta_{15} & \dots & \beta_{1(j-1)} & \beta_{1j} \\ \beta_{21} & \beta_{22} & \beta_{23} & \beta_{24} & \beta_{25} & \dots & \beta_{2(j-1)} & \beta_{2j} \\ 0 & \beta_{32} & \beta_{33} & \beta_{34} & \beta_{35} & \dots & \beta_{3(j-1)} & \beta_{3j} \\ 0 & 0 & \beta_{43} & \beta_{44} & \beta_{45} & \dots & \beta_{4(j-1)} & \beta_{4j} \\ 0 & 0 & 0 & \beta_{54} & \beta_{55} & \dots & \beta_{5(j-1)} & \beta_{5j} \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & \beta_{(j+1)j} \end{bmatrix}$$

These are the dimensions of the matrices involved:
$A \in R^{m \times m}$ (the matrix of the system)
$Q_j \in R^{m \times j}$ (a long and thin matrix)
$Q_{j+1} \in R^{m \times j+1}$ (a long and thin matrix)
$H_j \in R^{(j+1) \times j}$ (a rectangular matrix)
Notice also that the expression $AQ_j = Q_{j+1} H_{j+1}$ is not equivalent to $A = Q_{j+1} H_j Q_j^T$, because the matrix Q_j is rectangular and therefore $Q_j Q_j^T \neq I$

Variant: to have the same matrix Q_n to both sides, it is possible to rewrite the same expression as: $AQ_n = Q_n H_n + q_{n+1} \beta_{n+1} e_n^T$ where $e_n^T = [0\,, 0\,, ..., 1]$ and H_n is formed by just the first n rows of the matrix H.

Lucky Breakdown
At some step it may happen that $\beta_{j+1} = 0$, hence $Aq_j = \beta_1 q_1 + \beta_2 q_2 + ... + \beta_j q_j + 0 q_{j+1}$ thus contradicting $Aq_{j+1} \in K_{j+1}(A, b) \backslash K_j(A, b)$, it means that the vector q_{j+1} is no longer uniquely determined. After the breakdown, the factorization is equal to $AQ_n = Q_n H_n$, because $q_{n+1} \beta_{n+1,n} e_n^T$ is equal to 0 (since $\beta_{n+1,n} = 0$).

The breakdown might happen at any step of the iterative algorithm, for example it could even happen in the first step if the vector b is an eigenvector of the matrix A, indeed in this case, we can easily find that β_2 must be equal to 0.

If we get a "lucky breakdown", we can compute the exact solution of the linear system $Ax = b$. However usually we want to stop the iterations much early, when $n \ll m$.

3.5 GMRES

Now that we have the Krylov subspace and the Arnoldi iteration tools at our disposal, we can use them to solve a related problem.
Let $r_0 = b - Ax_0$ be the residual error given an initial guess $x_0 \neq 0$. It follows that $r_0 = b$ if $x_0 = 0$. GMRES approximates the exact solution of $Ax = b$ by the vector $x_n \in K_n$ that minimizes the Euclidean norm of the residual $r_n = b - Ax_n$. We can't use the vectors $r_0, Ar_0, ..., A^{n-1}r_0$ as a basis for K_n because they could be linearly dependent; we instead rely on the orthonormal vectors $q_1, q_2, ..., q_n$ found by the Arnoldi iteration. When framing the problem this way we see that $x_n = Q_n y$. From this it follows that

$$\min_{x_n \in K_n(A,b)} \|Ax_n - b\| = \min_{y \in R^n} \|AQ_n y - b\|$$

The right hand side can be interpreted as a Least Squares problem with matrix $AQ_n \in R^{n \times m}$. We can further develop it:

$$\min_{y \in R^n} \|AQ_n y - b\| = \left\|Q_{n+1}\underline{H_n} y - Q_{n+1} e_1 \|b\|\right\| = \left\|Q_{n+1}(\underline{H_n} y - e_1 \|b\|)\right\| = \left\|\underline{H_n} y - e_1 \|b\|\right\|$$

which now is a Least Squares problem with the (upper Hessenberg) matrix $\underline{H_n} \in R^{(n+1)\times n}$. What can we do with this? Well, let's go back to the residual:

$$\|r_n\| = \|b - Ax_n\| = \left\|b - A(x_0 + Q_n y_n)\right\| = \|r_0 - AQ_n y_n\| = \|\beta q_1 - AQ_n y_n\| =$$
$$\left\|\beta q_1 - Q_{n+1}\underline{H_n} y_n\right\| = \left\|Q_{n+1}(\beta e_1 - \underline{H_n} y_n\right\| = \left\|\beta e_1 - \underline{H_n} y_n\right\|$$

where $e_1 = (1, 0, ..., 0)^T$ is the first vector in the standard basis of R^{n+1} and $\beta = \|r_0\|$.
Now x_n can be found by minimizing the Euclidean norm of the residual $r_n = \underline{H_n} y_n - \beta e_1$, which is a linear Least Squares problem.
To recap, at each step n of the iteration, we need to compute q_n with the Arnoldi method, find the y_n which minimizes $\|r_n\|$ and compute $x_n = x_0 + Q_n y_n$ until the residual is small enough.

Algorithm 3 Pseudocode of the algorithm GMRES

$q_1 \leftarrow b/\|b\|$
for $n = 1, 2, 3, ...$ **do**
  (Step n-th of Arnoldi iteration)
  Find y to minimize $\left\|H_n y - \|b\| e_1\right\|$ $(= \|r_n\|)$
  $x_n \leftarrow Q_n y$
end for

3.6 Complexity

The complexity to solve the least squares problem $\left\|H_n y - e_1 \|b\|\right\|$ is equal to $O(n^3)$, since $H_n \in R^{(n+1)\times n}$, however, due to the special structure of H, we can compute qr(H) with $O(n^2)$, not $O(n^3)$.

However Arnoldi is the part of the algortihm with the highest complexity; at each step of Arnoldi we have to compute a matrix-vector product whose complexity is equal to $O(m^2)$ in the worst case, but this complexity can be reduced down to $O(m)$ if the matrix is sparse. Hence, since the iterations are n, the complexity is $O(mn)$.

Moreover, we have to consider that at each step the vector w must be update through repeated scalar products, the complexity of this operation is equal to $O(mn^2)$, since, in the worst case, the scalar product must be computed n times for each step (where n is the number of steps computed). Hence, the total complexity of the algorithms GMRES and Arnoldi is equal to $O(mn^2 + n\, mat - vect)$, where $mat - vect$ is the complexity of the matrix-vector product.

3.7 Convergence of GMRES

Considering a vector $x_n \in K_n(A, b)$, it implies that $x_n = p(A)b$ for some polynomial $p(t)$ of degree $< n$. $p(A) = (\alpha_0 I + \alpha_1 A + \alpha_2 A^2 + ... + \alpha_{n-1} A^{n-1})b$.

Hence, to solve the linear system we are looking for.

$$\min_{x_n = p(A)b} \|Ax_n - b\| = \min_{p\ of\ degree < n} \left\|Ap(A) - I)b\right\| \leq \min_{p\ of\ degree < n} \left\|Ap(A) - I\right\| \|b\|$$

GMRES finds the best polynomial for us.
given A a matrix diagonalizable ($A = VDV^{-1}$), then

$$A^k = VDV^{-1}VDV^{-1}VDV^{-1}...VDV^{-1} = VD^kV^{-1}$$

$$A^k = V \begin{bmatrix} \lambda_1^k & & 0 \\ & \ddots & \\ 0 & & \lambda_m^k \end{bmatrix} V^{-1}$$

And so,

$$Ap(A) - I = V \begin{bmatrix} \lambda_1 p(\lambda_1) - 1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m p(\lambda_m) - 1 \end{bmatrix} V^{-1}$$

Here if A has at most n distinctive eigenvalues, it means that must exist a polynomial of degree $p < n$ such that $p(\lambda_i) = 1/\lambda_i$ for each λ of A, hence $\lambda_i p(\lambda_i) - 1$ must be equal to 0, namely $\min_{x_n = p(A)b} \|Ax_n - b\| = 0$, so GMRES can reach the exact solution in n steps.

This argument can be extended to matrices that have clustered eigenvalues; if a matrix A has n groups of clustered eigenvalues, it means that exists a polynomial $p(x)$ of degree $p < n$ such that the roots of the function $q(x) = xp(x) - 1$ approximate (more or less) precisely the the clusters of eigenvalues of A. However, we can expect that $\min_{p(t)} max_{i \in [1,m]} \left\| q(\lambda_i) \right\| = \epsilon$ to be small enough. Hence

$$\left\| Ap(A) - I \right\| \leq \|V\| \left\| \begin{bmatrix} \lambda_1 p(_1) - 1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m p(_m) - 1 \end{bmatrix} \right\| \left\| V^{-1} \right\| \leq \epsilon K(V)$$

where $K(V)$ is the condition number of the matrix V and it is equal to $|\lambda_{max}| / |\lambda_{min}|$ (if V is normal). Moreover we can derive that the approximation of the polynomial $q(x) = xp(x) - 1$ is more precise if the eigenvalues are distant from the origin, indeed in this case the value of ϵ would be smaller compared to an analogous system with the eigenvalues closer to the origin. It is even possible to figure out a matrix P and to solve the system $PAx = Pb$ in order to have better locations for the system's eigenvalues.

3.8 Computing eigenvalues with Arnoldi

Arnoldi's iterations can be also used to compute some eigenvalue of a matrix A (with a low complexity). Firstly, if a breakdown occurs after n steps, then it means that we can write the factorization form as, $AQ_n = Q_n H_n$. Here it is easy to prove that the eigenvalues of H_n are a subset of the eigenvalues of A, indeed considering the pairs $(\lambda_i, w_i) i \in [1, n]$ respectively the eigenvalues and eigenvectors of H_n, then $(\lambda_i, Q_n w_i) i \in [1, n]$ are n of the eigenvalues and eigenvectors of A (the proof $AQ_n w_i = Q_n H_n w_i = Q_n H_n \lambda_i$). The pairs $(\lambda_i, Q_n w_i)$ are called Ritz pairs of A, an interesting result is that even if we do not reach any breakdown, the pairs $(\lambda_i, Q_n W_i)$ are good approximation of the real Ritz pairs of A and the more Arnoldi iterates, the more accurate is that approximation. Arnoldi converges to the eigenvalues with the highest modulus of the matrix A, hence it is a good method to compute the eigenvalues with the highest modulus for a sparse matrix. **AGGIUNGERE CONSIDERAZIONE SUL CALCOLARE AUTOVALORI PIÙ PICCOLI**

3.9 MINRES

When the matrix A is symmetric, the algorithm takes the name of MINRES. In this case also the matrix of H_n is symmetric, hence, since in the lower part there are only zeros (H_n is almost an upper triangular matrix), the only values different from zero are located in the "tridiagonal" of the matrix. Namely

$$H_n = \begin{bmatrix} * & * & & & & & & \\ * & * & * & & & & & \\ & * & * & * & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & * & * & * & \\ & & & & & * & * & * \\ & & & & & & * & * \end{bmatrix}$$

It is actually easy to prove it, indeed given A a symmetric matrix, exploiting the orthogonality of the matrix Q_n, we can write

$$AQ_n = Q_n H_n + q_{n+1} h_{n+1,n} e_n^T \Rightarrow q_i^T A q_j = H_{ij}$$

because if we multiple q_i^T on the left end e_j, on the right (here $i \leq n$ and $j < n$), we obtain:

$$q_i^T A Q_n e_j = q_i^T Q_n H_n e_j \Rightarrow q_i^T A q_j = q_i^T Q_n \begin{bmatrix} H_{1j} \\ \vdots \\ H_{nj} \end{bmatrix} \Rightarrow q_i^T A q_j = e_i^T \begin{bmatrix} H_{1j} \\ \vdots \\ H_{nj} \end{bmatrix} = H_{ij}$$

and here if A is symmetric $H_{ij} = q_i^T A q_j = (q_j^T A q_i)^T = H_{ji}$ and so H is symmetric. Here it is clear that most of Arnoldi's iterations are completely useless, because only three or two entrie per row are different from zero in H_n. This result implies that the complexity of the algorithm MINRES is slightly lower than the complexity of GMRES, indeed the complexity decreases from $O(mn^2 + n\,mat - vec)$ to $O(mn + n\,mat - vec)$, because we do not have to compute all the scalar products required to compute the entries of H_n and to update the vector w. From the previous formulas it results that the complexity of MINRES is lower than the complexity of GMRES only if the iterations of Arnoldi are sufficiently few and so the biggest contribute to the complexity of the algorithms is due to the matrix-vector product. Finally, the iterations of Arnoldi for simmetric matrices are called Lanczos iterations.

4 Conjugate Gradient

The conjugate gradient method is an algorithm used to approximate the solution of linear systems whose matrix is positive-definite.

4.1 Introduction

In mathematics, a symmetric matrix A (i.e. $A = A^T$) with real entries is positive-definite if the number $x^T Ax$ is greater than 0 for each vector x different from zero; in other words, a real matrix A is positive-definite if and only if it is symmetric and its eigenvalues are strictly greater than 0. the definition of positive-definite matrix can be enlarged also for complex square matrices, in this case a Hermitian matrix A (A Hermitian $\iff$ $a_{ij} = \overline{a_{ji}}$) if the real number $x^* Ax$ is strictly greater than zero for every nonzero complex vector x.

The A-norm of a vector $u \in R^n$ is defined as $\sqrt{u^T Au}$, moreover we say that two vectors u and v are A-orthogonal if $u^T Av = 0$, since A is a positive-definite matrix, the left-hand side defines an inner product $u^T Av = \langle u, v \rangle_A \coloneqq \langle Au, v \rangle = \langle u, A^T v \rangle = \langle u, Av \rangle$. Two vectors are conjugate if and only if they are conjugate with respect to this inner product, this is a symmetric relation (if u is conjugate to v then v is conjugate to u). Given a set of vectors $D = (d_1, ..., d_n)$ its vectors are mutually conjugate vectors with respect to A if $d_i A d_j = 0$ for all $i \neq j$; these vectors form a basis for the vector space R^n and it is possible to express the solution of the system $Ax = b$ as linear combination of the basis D.

4.2 The Conjugate method as a direct method

The solution of the system is given by $x_* = \sum_{i=1}^{n} \alpha_i d_i$, hence to calculate x_*, we can compute the coefficients α_i in this way

$$x_* = \sum_{i=1}^{n} \alpha_i d_i \Rightarrow Ax_* = \sum_{i=1}^{n} \alpha_i A d_i \Rightarrow d_k^T A x_* = \sum_{i=1}^{n} \alpha_i d_k^T A d_i = \alpha_k d_k^T A d_k = \alpha_k \langle d_k, d_k \rangle_A$$

Since $d_k^T A x_* = d_k^T b$, the coefficient α_i is equal to $\alpha_k = \langle d_k, b \rangle / \langle d_k, d_k \rangle_A$. This is a way to use the conjugate gradient as a direct method.

4.3 The conjugate method as an iterative method

4.3.1 A first implementation

Actually it is possible also to implement an iterative version of the conjugate gradient, this allow us to approximately solve linear systems without computing all the vectors d_i. The conjugate gradient as iterative method requires only to compute a good set of conjugate vectors d_i in order to obtain a good approximation of the system's exact solution. The main idea is to apply an optimization algorithm to understand the which is the direction of the exact solution x_x, that solution is also the unique minimum of the quadratic form $f(x) = 1/2x^T Ax - x^T b$, which is a strictly convex function if A is positive-definite (and so x_* is the unique minimizer of $f(x)$).

Firstly, to approximate the minimum of the quadratic function, we can exploit the gradient of the quadratic function to find its minimum. Hence, we can compute the first vector of the conjugate basis, i.e. d_0, as $-\nabla f(x) = b - Ax$ at $x = x_0$, namely $d_0 = b - Ax_0$. Then, the other vectors of the basis will be conjugate to the gradient, hence the name conjugate gradient method. In this first step, we can also notice that the residual (r_0) is equal to d_0, therefore $r_0 = b - Ax_0$. Finally we assume, without any limitation, that $x_0 = 0$, hence $d_0 = r_0 = b$. At the generic k-th step, the residual r_k is equal to $b - Ax_x$, in the gradient descent method we would move in the direction r_k, but in the conjugate gradient method, the vector d_k must be conjugate to all the other vectors of the basis, hence we have to find a way to compute that conjugate vector. To do that, we can exploit the Gram-Schmidt process, this is a method to orthonormalize a set of vector in a given inner product. Given a finite set of linearly independent set of vectors $S = \{v_1, ..., v_k\}$ Gram-Schmidt returns an orthogonal set of vector $S' = \{u_1, ..., u_k\}$. The Gram-Schmidt method works through the projection operator, which is defined as $proj_u = \frac{\langle u,v \rangle}{\langle u,u \rangle} u$, hence

$$u_k = v_k - \sum_{j=1}^{k-1} proj_{u_j}(v_k)$$

.
Therefore, if we apply this process to our sets of vectors r_i and d_i, we obtain

$$p_k = r_k - \sum_{i<k} \frac{p_i^T A r_k}{p_i^T A p_k}$$

Then finally with can compute α_k studying the function $f(x_k + \alpha_k d_k)$ (from the previous computation) and in particular the derivative with respect to α_k. The final result is

$$\alpha_k = \frac{p_k^T r_k}{p_k^T A p_k}$$

Finally, we can compute x_{k+1}

$$x_{k+1} = x_k + \alpha_k d_k$$

4.3.2 Another better implementation

The implementation of the conjugate gradient method described above is too computationally expensive. Firstly, we need to store a lot of vectors to iteratively compute the set of vectors r_i and d_i, besides there are a lot of matrix-vector products to compute. However, A deeper analysis of the algorithm suggests that we can cut the cost of that algorithm. Indeed the vector r_i is orthogonal to r_j with $i \neq j$ and d_i is A-orthogonal to d_j with $i \neq j$. This can be regarded that as the algorithm progresses, d_i and r_i span the same Krylov subspace; $K_j(A,b) = span(x_1, x_2, x_3, ..., x_j) = span(d_0, d_1, ..., d_{j-1}) = span(r_0, r_1, ..., r_{j-1})$ and so $(r_0, ..., r_{j-1})$ is a orthogonal basis of the vector space $K_j(A,b)$ with respect to the conventional inner product and $(d_0, ..., d_{j-1})$ is a orthogonal basis of the vector $K_j(A,b)$ with respect to the inner product defined by the matrix A. Finally, we can define r_j in this way; $r_j = b - Ax_j = b - Ax_{j-1} - A(x_j - x_{j-1}) = r_{j-1} - \alpha_j A d_{j-1}$ (the last equivalence derives from $x_j = x_{j-1} + \alpha_j d_{j-1}$), this way to compute the residuals is less expensive, since the vector Ad_{j-1} is already compute to evaluate α_k.

Now we are able to write a better equivalent pseudocode of the algorithm, which is less expensive and requires to store only 3 vectors at each step.

Algorithm 4 Pseudocode of the conjugate gradient method

$x_0 = 0, r_0 = b, d_0 = b$ //Initializing the variables
for $j = 1, 2, 3, ..., n$ **do**
  $\alpha_j \leftarrow \frac{r_{j-1}^T r_{j-1}}{d_{j-1}^T A d_{j-1}}$
  $x_j \leftarrow x_{j-1} + \alpha_j d_{j-1}$
  $r_j \leftarrow r_{j-1} - \alpha_j A d_{j-1}$
  $\beta_j \leftarrow \frac{r_j^T r_j}{r_{j-1} r_{j-1}}$
  $d_j \leftarrow r_j + \beta_j d_{j-1}$
end for

We can stop the iterations when the residual is sufficiently small, hence when we have found a good approximation of the linear system's solution.

4.4 Proving the orthogonality and the A-orthogonality

In this section we want to prove that given

$$\alpha_j = \frac{r_{j-1}^T r_{j-1}}{d_{j-1}^T A d_{j-1}}, \quad \beta = \frac{r_j^T r_j}{r_{j-1}^T r_{j-1}}$$

The set of vectors r_j and d_j are respectively orthogonal are A-orthogonal.

Theorem $r_i^T r_j = 0$ **and** $d_i^T A d_j = 0$ **if** $i \neq j$

1) Base Case (j=1)
We want to prove that $r_0^T r_1 = 0$. We know that $r_0 = b$ and

$$r_1 = r_0 - \alpha_1 A d_0 = b - \frac{b^T b}{b^T A b} A b = b - \frac{|b|^2}{b^T A b} A b \Rightarrow r_0^T r_1 = |b|^2 - \frac{|b|^2}{A b} A b = |b|^2 - |b|^2 = 0$$

2) Inductive hypothesis, we assume that the thesis is true for $j - 1$.

3) Proving the thesis for a generic j

First case, $i < j - 1$

$$r_i^T r_j = r_i^T (r_{j-1} - \alpha_j A d_{j-1}) = r_i^T r_{j-1} - \alpha_j r_i^T A d_{j-1}$$

Where $r_i^T r_{j-1} = 0$ for hypothesis and $r_i \in K_{j-1}$ therefore we can write r_i as a linear combination of the vectors $(d_0, ..., d_{j-2})$, and so for inductive hypothesis even the second term is equal to 0.

Second case, $i = j - 1$

$$r_{j-1}^T r_j = r_{j-1}^T r_{j-1} - \alpha_j r_{j-1}^T A d_{j-1}, \;\; \alpha_j = \frac{r_{j-1}^T r_{j-1}}{d_{j-1}^T A d_{j-1}}$$

The result is equal to 0 if

$$\frac{r_{j-1}^T r_{j-1}}{d_{j-1}^T A d_{j-1}} = \frac{r_{j-1}^T r_{j-1}}{r_{j-1}^T A d_{j-1}} \Rightarrow d_{j-1}^T A d_{j-1} = r_{j-1}^T A d_{j-1}$$

but we know that,

$$d_{j-1}^T A d_{j-1} = (r_{j-1} + \beta_{j-1} d_{j-2})^T A d_{j-1} = r_{j-1} A d_{j-1} + \beta_{j-1} d_{j-2}^T A d_{j-1}$$

Where $\beta_{j-1} d_{j-2}^T A d_{j-1} = 0$.

In a similar way we can also prove that $d_i^T A d_j = 0$ with $i < j$.

4.5 Complexity

It is actually easy to prove that the complexity of the conjugate gradient method is asymptotically equal to MINRES, hence $O(mn + n\, mat - vec)$.

4.6 Conjugate Gradient as a Krylov space method

The Conjugate Gradient method compute, implicitly, an orthonormal basis of $K_j(A, b)$, which is equal to the orthonormal basis computed in GMRES (up to a sign), $q_j = \pm r_j / \|r_j\|$.
Which is now the equation to solve to interpret the conjugate gradient method as a Krylov space method? MINRES would derive at each step x_j as $y_j = \underline{H_j}^T e_1 \|b\|$, but the conjugate gradient does something slightly different, indeed here $y_j = H_j^{-1} e_1 \|b\|$. ($x_j = Q_j y_j$)

Where $\underline{H_j} \in R^{(j+1)xj}$ and $H_j \in R^{jxj}$, namely the matrix H_j is a square matrix, and so we can compute its inverse.
This happens because at each step we obtain $r_j = b - Ax_j$ which is orthogonal to all the previous vectors computed and these vectors $(r_0, r_1, ..., r_{j-1})$ are an orthogonal basis of the vector space $K_j(A, b)$. It follows that they are also orthogonal to the columns of the matrix Q_j computed by Arnoldi, therefore

$$0 = Q_j^T r_j = Q_j^T (b - Ax_j) = \begin{bmatrix} \|b\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} - Q_j^T A x_j = \begin{bmatrix} \|b\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} - Q_j^T A Q_j y_j = \begin{bmatrix} \|b\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} - H_j y_j$$

The last equality derives from

$$AQ_j = Q_j H_j + q_{j+1} ... \Rightarrow Q_j^T A Q_j = H_j + 0$$

and so

$$0 = \|b\| e_1 - H_j y_j$$

4.7 The best approximation property

About the conjugate gradient method there is a theorem that states

Theorem x_j is the best approximation of the exact solution x of the linear system $Ax = b$ inside $K_j(A, b)$ in the A norm, namely

$$x_j = \underset{z \in K_j(A,b)}{\arg\min} \|z - x\|_A = \underset{z \in K_j(A,b)}{\arg\min} (z - x)^T A (z - x)$$

Proof: a generic $z \in K_j(A, b)$ can be written as $x_j + y$, $y \in K_j(A, b)$

$$(z - x)^T A (z - x) = (x_j + y - x)^T A (x_j + y - x) = y^T A y + (x_j - x)^T A y + y^T A (x - x_j) + (x - x_j)^T A (x - x_j)$$

Here we claim that $y^T A (x_j - x) = (x_j - x)^T A y$ and that they are both equal to 0. This is true because, $A(x_j - x) = Ax_j - Ax = Ax_j - b = -r_j$ and we have already demonstrated that the vector r_j is orthogonal to all the vectors in $K_j(A, b) = span(r_0, r_1, ..., r_{j-1})$. Hence it remains only

$$(z-x)^T A(z-x) = (x_j - x)^T A(x_j - x) + y^T Ay \Rightarrow \|z-x\|_A = \|x_j - x\|_A + (a\ positive\ quantity)$$

it follows that x_j is the vector that approximates better the solution of the system in the A norm.

With the best approximation theorem is also possible to prove an another theorem that states
Theorem at each step of the conjugate gradient

$$x_j = \underset{z \in K_j(A,b)}{\arg\min} \frac{1}{2} z^T Az - b^T z + c$$

Hence at each step x_j is the vector that minimizes the quadratic form among all the vectors belonging to $K_j(A,b)$.
To prove this theorem we have to consider that the quadratic form is the A norm of the error.
Proof

$$\|z-x\|_A = (z-x)^T A(z-x) = z^T Az - x^T Az - z^T Ax + x^T Ax$$

$$x^T Az = z^T Ax = b^T z = z^T b \Rightarrow \|z-x\|_A = z^T Az - 2b^T z + costant$$

which is exactly the quadratic form. Hence since we have already demonstrated the best approximation property, it follows that we have also demonstrated this second theorem.

4.8 Convergence of the Conjugate Gradient

It is actually possible to prove that

$$\|x_n - x\|_A \leq (\frac{\sqrt{\lambda_{max}} - \sqrt{\lambda_{min}}}{\sqrt{\lambda_{max}} + \sqrt{\lambda_{min}}})^n \|x_0 - x\|_A$$

Where λ_{max} and λ_{min} are respectively the biggest and the smallest eigenvalues of the matrix A.

The number $\frac{\sqrt{\lambda_{max}} - \sqrt{\lambda_{min}}}{\sqrt{\lambda_{max}} + \sqrt{\lambda_{min}}}$ is related to $K(A)$ (where $K(A) = \frac{\lambda_{max}}{\lambda_{min}}$).

$$\frac{\sqrt{\lambda_{max}} - \sqrt{\lambda_{min}}}{\sqrt{\lambda_{max}} + \sqrt{\lambda_{min}}} = \frac{\frac{\sqrt{\lambda_{max}}}{\sqrt{\lambda_{min}}} - 1}{\frac{\sqrt{\lambda_{max}}}{\sqrt{\lambda_{min}}} + 1} = \frac{\sqrt{K(A)} - 1}{\sqrt{K(A)} + 1} \in [0, 1)$$

The is an important bound about the convergence of the conjugate gradient. Firstly, the smaller is $K(A)$ the faster is the convergence of the algorithm. Secondly, this convergence rate is faster than the convergence of the algorithms GMRES and MINRES, beucause here we have a guaranteed regular steady convergence which is actually not present in the other two algorithms. Finally, it is also important to say that usually the convergence of CG is even faster (it depends on the clusters of the eigenvalues), however, in the worst case, we can converge to the solution with this speed.

We can also apply the same reasoning of GMRES and MINRES to the conjugate gradient method
$x_n \in K_n(A,b) \rightarrow x_n = p(A)b$ where $p(x)$ is a polynomial of degree $p < n$.
CG computes the minimizer $\|x_n - x\|_A = \min_{p(t)} \|p(A)b - x\|_A = \|p(A)Ax - x\|_A = \|(p(A)A - I)x\|_A$ if $A = VDV^T$ (diagonalizable) then $p(A)A - I = V(p(D)D - I)V^{-1} =$

$$= V \begin{bmatrix} \lambda_1 p(\lambda_1) - 1 & & \\ & \ddots & \\ & & \lambda_n p(\lambda_n) - 1 \end{bmatrix} V^{-1}$$

Again, if A as at most $n < m$ different eigenvalues, then we can find p such that $p(\lambda_i) = 1/\lambda_i$ and so we reach the exact solution of the system. Otherwise we can notice that we can find a better approximation of the system if its eigenvalues are clustered and far away from zero.

5 KKT systems

Since this topic is not important for our project like the others previously described, we will treat it only briefly, describing only the most important features related to KKT systems and the Karush-Kuhn-Tucker conditions.

5.1 Introduction to optimization

Mathematical optimization deals with the problem of finding numerically minimums of a function, this function is usually called cost function. Hence, the optimization problem consists of minimizing a function by systematically choosing input values from within an allowed set and computing the value of the function.

5.2 Convex optimization

Convex optimization is a subfield of mathematical optimization that studies the problem of minimizing convex functions over convex sets.

A function is convex if the line segment between any two points on the graph of the function does not lie below the graph between the two points.

A set is convex if, given any two points of the set, the set contains the whole line segment that joins them.

Many classes of convex optimization problems admit polynomial-time algorithms, instead mathematical optimization is generally a NP-hard problem.

5.3 The Karush-Kuhn-Tucker conditions

The KKT conditions are important to solve problems of convex optimization. If these conditions are satisfied, then it means that, essentially, you can reduce solving an optimization problem to solving a bunch of equations and inequalities.

Let's consider an optimization problem, hence we have to find: $\min f(x)$, moreover we have also some constrains on the domain, in particular we have both inequality constraint functions $(g_i(i = 1, ..., m))$ and equality constraint functions $(h_i(i = 1, ..., l))$. Now, since we are dealing with convex optimization problems, the functions $f(x)$ and $g_i(x)(i = 1, ..., m)$ must be convex, while the functions $h_i(x)(i = 1, ..., l)$ must be linear. So the problem that we have to solve is

$$\min_{\substack{g_i(x)\leq 0\ \forall i\in[1,m]:\\ h_i(x)=0\ \forall i\in[1,l]}} f(x)$$

to get rid of these constraint functions, applying the method of Lagrange multipliers, we can rewrite the problem in this way

$$\max_{u_i\geq 0, v_i} \min_x f(x) + \sum_i v_i h_i(x) + \sum_j u_j g_j(x)$$

This is exactly the dual problem of the first one (which is the primal). In this way we are exactly rewriting the constraint functions and so the optimal value will be chosen in the acceptable domain of the function. This dual form is particularly interesting because firstly, we compute the x required to minimize the function, therefore we compute the minimum of the function according to the variables v_i and u_i, and only in a second time we find the values of these variables to maximize the function.

Here the function $f(x) + \sum_i v_i h_i(x) + \sum_j u_j g_j(x)$ is called the Lagrangian function, and we can easily identify which are the possible candidate minimums studying the gradient of the function. Indeed, the candidate minimums are the saddle points of the function, hence we have to find the points x^* such that

$$\nabla f(x^*) + \sum_{j=1}^{l} v_i h_i(x^*) + \sum_{i=1}^{m} u_i \nabla g_i(x^*) = 0$$

However we need additional assumptions to determine if x^* is an optimal solution for the problem. These conditions take the name of Karush-Kuhn-Tucker conditions and they are

1) Primal feasibility
   $g_i(x^*) \leq 0,\ for\ i = 1, ..., m$
   $h_j(x^*) = 0,\ for\ j = 1, ..., l$

2) Dual feasibility
$u_i \geq 0, \ for \ i = 1, ..., m$

3) Complementary slackness
$\sum_{i=1}^{m} \mu_i g_i(x^*) = 0$

The last condition derives from the fact that x and u_i must be chosen properly, without affecting the optimal value of the minimization problem.

Here it finishes the theorical part of the relation.

6 Algorithms

Below we have described the algorithms that we have implemented, highlighting the results obtained, the issues encountered and how we have solved them. Please, read the file README.dm for a full description of the functions implemented.

6.1 Version

This report is related to the first delivery of the project.

6.2 Initialization of the system

Creation of the directed graph

First of all, it is required a method to compute the directed graph $G = (N, A)$; G is required to get the matrix E which represent the edge-node adjacency matrix of the graph G. The graph is computed by the function *graph_initialization* which requires three parameters, n the number of graph's node, e i.e. the number of edges and s a seed to generate random numbers. There is not any assumption on the graph, therefore it could not connected or cyclic or there may be round edges (from and to the same node). For these reasons the function is pretty simple, it just creates two arrays that represent the sources and destinations of the edges, after that the function creates the graph.

Creation of the matrices E, D, b and c

The function *system_initialization* creates the matrices required to outline the system, i.e. the matrices E, D, b and c. The function requires only the directed graph G and a seed s to generate random numbers. given n, the number of nodes in G then $E \in R^{nxn}, D \in R^{nxn}, b \in R^{nx1} mc \in R^{nx1}$. E can be retrieved directly from the graph G, since E is the edge-node adjacency matrix of G, the cell $E_{u,v}$ is equal to 1 if exists a direct edge from u to v, otherwise it is equal to 0. D is just a positive-definite diagonal matrix, hence its eigenvalues are strictly greater than 0; these eigenvalues are randomly created by *system_initialization*, then it is possible to define the matrix D. The vectors b and c, instead are just the solutions of the system, the function *system_initialization* creates them randomly.

Creation of the system

We the matrices E, D, b, c we can finally define the system of our algorithm, which is defined in the following way

$$\begin{bmatrix} D & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$$

The function *system_assembly* takes the matrices previously computed to create that system, returning a matrix A and a vector s, where

$$A = \begin{bmatrix} D & E^T \\ E & 0 \end{bmatrix} \qquad s = \begin{bmatrix} b \\ c \end{bmatrix}$$

These matrices are required to apply MINRES/GMRES to our system (**A2**).

6.3 The Conjugate Gradient method (A1)

The function *conjugate_gradient* of our project implements a CG algorithm. This function requires as inputs the edge-node E adjacency graph of the previously computed directed graph G, the diagonal matrix D of the system, the first two vectors b and c of the constant terms of the original system (before the reduction), a function handle f to compute a matrix product and the number of iterations n that the algorithm has to compute.

The algorithm initializes the first column of x (the best approximation of the system's resolution in $K_0(A, b)$ in A norm) to be zero and the first column of both r and d (respectively the residual and the direction) to s, which is the sign-changed gradient of the quadratic form when $x = 0$. Then the iteration starts: at each step we want to find the coefficients a and b such that we can update the values of x, r and d. This approach lets us retain the A-orthogonality of d and the (classical) orthogonality of r between the values in each subsequent iteration, while the algorithm updates the projection of the solution x to the appropriate Krylov subspace.

At the end the algorithm returns the matrix x of the approximated solutions (for each step), the matrix r of the residuals and the matrix d of the conjugate vectors.

6.4 MINRES/GMRES (A2)

The function *minres* of our project implements the algorithm MINRES. This function requires the matrices A and s which are respectively the matrix of the system and the value of each equation of the system, moreover the function requires a third argument, which is the number of Arnoldi iterations that the function has to compute.

Firstly, the function *minres* creates the first vector of the orthonormal basis by dividing b by its norm, then the algorithm initializes the matrices H and Q (in our algorithm the matrix Q1 represents what we have previously called Q_n, while Q2 represents Q_{n+1}). Then the algorithms computes n steps of Arnoldi iterations, where at each step the matrices H and Q are updated by concatenating the old version of these matrices with the new vectors computed by Arnoldi (at the k-th iteration, these vector are the (k+1)-th vector of the orthonormal basis, i.e. q_{k+1}, for the matrix Q and the vector of the coefficients to write Aq_k as a linear combination of the orthonormal basis Q_{k+1} for the matrix H). The Arnoldi iterations are computed by the function *arnoldi_iter* of our project.

At this point, after each step of Arnoldi, the algorithm exploits the factorization to find the vector $x_n \in K_n(A, b)$ that minimizes $\|Ax_n - b\|$, indeed the algorithm firstly computes $y = H_n \backslash e_1\|b\|$, namely the vector $y \in R^n$ that minimizes $\left\|H_n y - e_1\|b\|\right\|$ and then it computes $x_n = Q_n y$.

At the end of the algorithm execution, the function *minres* returns not only the final matrices Q and H, but also the residuals r obtained after each step of Arnoldi, hence r is a vector of dimension n, where the i-th entry of that vector is equal to $\|Ax_i - b\|$ with $x_i \in K_i(A, b)$. Through the vector r, we can study the trend of the residuals in order to see how fast they are decreasing at each step.

6.5 Complexity

The complexity of these algorithms is simply equal to the complexity of MINRES and the Conjugate Gradient method (already discussed in the theoretical part of the relation). Hence we can derive that the complexity of both (A1) and (A2) is equal to $O(mn + n\,mat - vec)$.

7 Experiments

8 Conclusions