LINEAR ALGEBRA V1.0

TABLE OF CONTENTS

1. INTRODUCTION

The Linear algebra module is a collection of various decomposition and factorization algorithms used to solve for overdetermined homogeneous linear system

2. PROBLEM STATEMENT

There is a need to compile all the various decomposition and factorization algorithms to compute the least-square errors and least square error norms to help solve the optimization problem of the linear systems of equations “**Ax=b**”

The main goal of this project is to create a Linear System class which is reusable and scalable and can be integrated in a machine learning module package to help with complex data transformations.

3. LINEAR ALGEBRA MODULE

The Linear Algebra supports the following decomposition and factorization algorithms to solve for the Least Squares optimization

1. Cholesky Factorization method

- The Cholesky factorization of a Hermitian positive-definite matrix **A**, is a factorization of the form where **L** is a lower triangular matrix with real and positive diagonal entries, and **L*** denotes the conjugate transpose of **L**
- **A** must be a symmetric and positive definite matrix
- **A** can be decomposed as shown in figure 1. Where A=L@L.T (@ =matrix multiplication)

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{pmatrix}$$

$$= \begin{pmatrix} L_{11}^2 & & \text{(symmetric)} \\ L_{21}L_{11} & L_{21}^2 + L_{22}^2 & \\ L_{31}L_{11} & L_{31}L_{21} + L_{32}L_{22} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{pmatrix},$$

we obtain the following:

$$\mathbf{L} = \begin{pmatrix} \sqrt{A_{11}} & 0 & 0 \\ A_{21}/L_{11} & \sqrt{A_{22} - L_{21}^2} & 0 \\ A_{31}/L_{11} & \left(A_{32} - L_{31}L_{21}\right)/L_{22} & \sqrt{A_{33} - L_{31}^2 - L_{32}^2} \end{pmatrix}$$

Figure 1. Cholesky Factorization

2. QR Decomposition method

- In QR decomposition, any real square matrix A may be decomposed as $\boldsymbol{A = QR}$ where Q is an orthogonal matrix (its columns are orthogonal unit vectors meaning) and R is an upper triangular matrix (also called right triangular matrix)
- The QR decomposition method can represented using the equation below, where A is a rectangular n x m matrix with n>=m

$$A = QR = Q\begin{bmatrix} R_1 \\ 0 \end{bmatrix} = [\,Q_1 \quad Q_2\,]\begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

- We use the "Gram-Schmidt" procedure to solve the QR decomposition shown in figure 2 below

$$A = \left[\, \mathbf{a}_1 \mid \mathbf{a}_2 \mid \cdots \mid \mathbf{a}_n \,\right].$$

Then,

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{a}_1, \qquad \mathbf{e}_1 = \frac{\mathbf{u}_1}{||\mathbf{u}_1||}, \\ \mathbf{u}_2 &= \mathbf{a}_2 - (\mathbf{a}_2 \cdot \mathbf{e}_1)\mathbf{e}_1, \qquad \mathbf{e}_2 = \frac{\mathbf{u}_2}{||\mathbf{u}_2||}. \\ \mathbf{u}_{k+1} &= \mathbf{a}_{k+1} - (\mathbf{a}_{k+1} \cdot \mathbf{e}_1)\mathbf{e}_1 - \cdots - (\mathbf{a}_{k+1} \cdot \mathbf{e}_k)\mathbf{e}_k, \qquad \mathbf{e}_{k+1} = \frac{\mathbf{u}_{k+1}}{||\mathbf{u}_{k+1}||}. \end{aligned}$$

Note that $||\cdot||$ is the L_2 norm.

Figure 2. Gram Schmidt Procedure for QR Decomposition

- The resulting QR decomposition is shown with the help of the equation below

$$A = \left[\, \mathbf{a}_1 \mid \mathbf{a}_2 \mid \cdots \mid \mathbf{a}_n \,\right] = \left[\, \mathbf{e}_1 \mid \mathbf{e}_2 \mid \cdots \mid \mathbf{e}_n \,\right] \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{e}_1 & \mathbf{a}_2 \cdot \mathbf{e}_1 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_1 \\ 0 & \mathbf{a}_2 \cdot \mathbf{e}_2 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_n \end{bmatrix} = QR.$$

3. Single Value Decomposition (SVD) method

The Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into three matrices. It has some interesting algebraic properties and conveys important geometrical and theoretical insights about linear transformations.

The SVD of n x m matrix A is given by the formula :
A = U @ Diagonal(Sigma) @ VT, where,

- U: nxm matrix of the orthonormal eigenvectors of .
- VT: transpose of a mxm matrix containing the orthonormal eigenvectors of A^{T}A.
- Sigma: a diagonal matrix of the singular values which are the square roots of the eigenvalues of A.T@A.

4. METHODOLOGY

4.1 Create and Initialize the Matrix A

The Matrix A is created and initialized using the constraint_matrix_initializer service, where we initialize the instance of Matrix class as a matrix object "A" and then extract the metadata for A as follows:

Metadata={matrix_rows, matrix_cols, matrix_dimension}

4.2 Create a Linear System

The next step is to create a Linear System class to precompute the decompositions/factorization of matrix A using the following methods

1. Cholesky factorization

2. QR decomposition

3. SVD decomposition

4.3. Save the Linear System

The Linear System class initializes the decomposition methods mentioned above and precomputes the decomposition results and finally **save** their respective decomposition results as transformation files in a pickle format.

4.4. Load the Linear System

The create_linear_system workflow invokes the methods of the Linear System Class and Loads the linear system variables and the precomputed decomposition results from transformation pickle files.

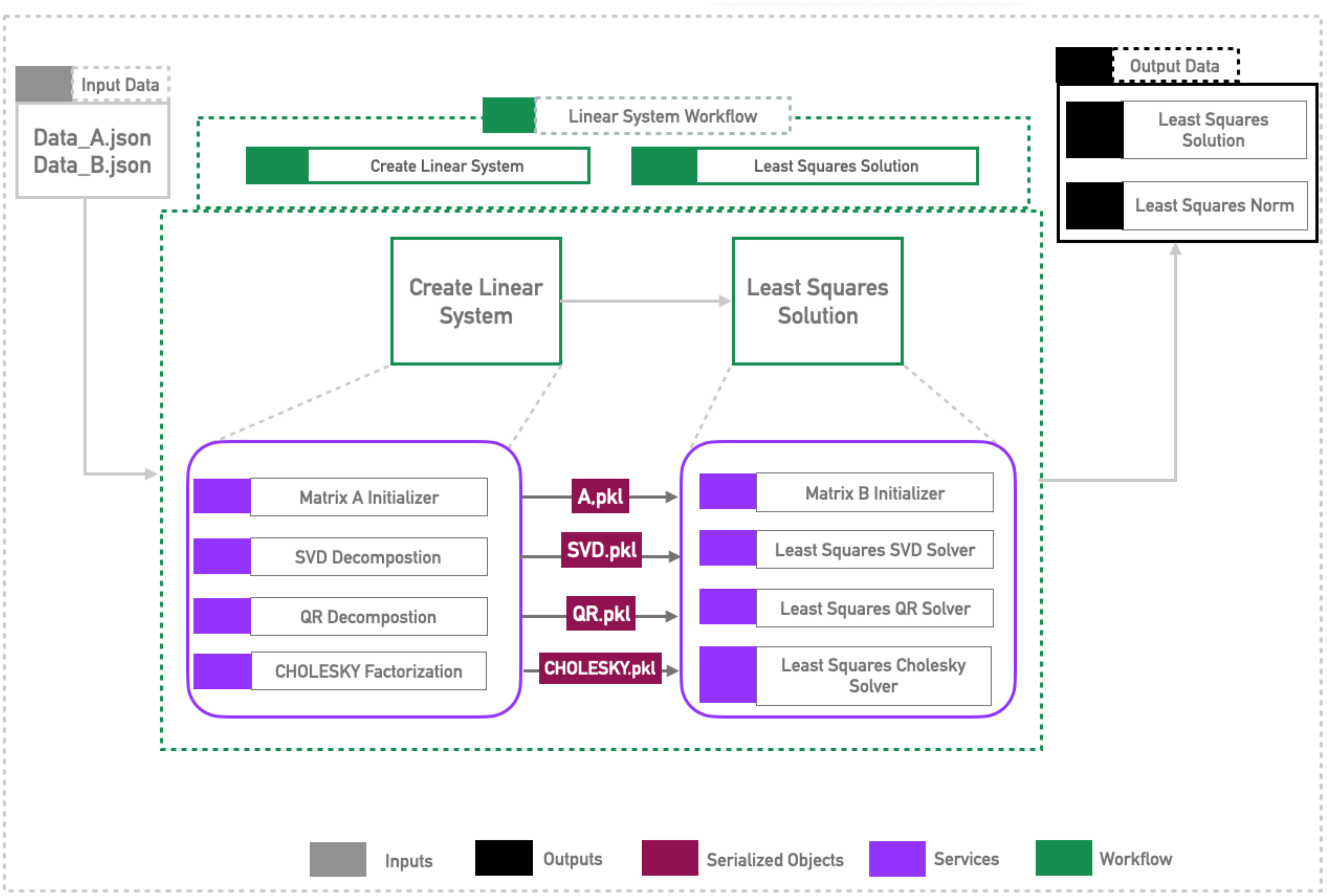
4.5. Compute the Least Squares Method

In this step, we load the linear system variables and the precomputed decomposition results and also instantiate the Right hand side ("B") to solve the linear system ("**Ax=b**") using least squares method and get "**x_lcs**" which is the least squares solution for x.

4.6. Compute the Least Squares Norm

Finally, we compute the least squares norm $||Ax-B||_2$ solution and we measure the $||x\_lcs||_2$ to understand the impact of minimizing the least squares to reach an optimal solution

4.7 Software Design Architecture



The figure 3 demonstrates the software design architecture.

- The software design pattern is built on the philosophy of modularization and packaging. The software is designed with a collection of modules and services to achieve code reusability and extensibility

4.8. Big O notation for decomposition methods

1. Cholesky Factorization

The cost of flops (floating point operations per second) for Cholesky factorization for A (n x n) matrix is explained as follows:

```
        for k in range(n):
            # diagonal elements
            l[k,k]=round(csqrt(l[k,k] - np.dot(l[k,0:k],l[k,0:k])),2)
            # non-diagonal elements
            for i in range(k+1,n):
                l[i,k]=(l[i,k]-np.dot(l[i,0:k],l[k,0:k]))/l[k,k]
```

Outer Loop k runs from 1 to n costing (n-1) divisions, $\frac{1}{3}n*n^2-1$ multiplications and $\frac{1}{3}n*n^2-1$ subtractions,

Hence, the Big O Notation is $O(\frac{1}{3}n^3)$

2. QR Decomposition

The cost of flops (floating point operations per second) for QR factorization for A (n x m) matrix is explained as follows:

```
        Q= np.zeros((self.n,self.m))
        for i, col in enumerate(self.A.T):
            Q[:,i]=col
            for prev in Q.T[:i]:
                Q[:,i]-=(prev@col)/(prev@prev)*prev
        Q=Q/(linalg.norm(Q,axis=0))
```

For the outer loop, we enumerate for Matrix A transpose as m x n and perform n multiplications, n-1 additions and n divisions and inner loop we perform m multiplications, n-1 additions and n subtractions,

so, the total FLOP count is given by

$$\frac{1}{2}m(m+1)(n-1)+\frac{1}{2}m(n)(m-1)+nm^2+mn$$

Hence, the Big O Notation is $O(2nm^2)$

5. FUTURE WORK

The future motivation is to generalize the workflows and make them scalable. The future steps are as follows.

- Add more functionality to the decomposition workflows to make it generic
- Add support for polar and NNMF decomposition algorithms
- Create a distributed architecture to process multiple RHS ("B") to optimize using parallel computing

6. REFERENCES

[1] https://en.wikipedia.org/wiki/QR_decomposition

[2] https://www.math.ucla.edu/~yanovsky/Teaching/Math151B/handouts/Gram-Schmidt.pdf

[3]https://en.wikipedia.org/wiki/Cholesky_decomposition

[4] https://en.wikipedia.org/wiki/Singular_value_decomposition

[5] https://en.wikipedia.org/wiki/Semi-orthogonal_matrix

[6] https://python.quantecon.org/svd_intro.html

[7] https://numpy.org/doc/stable/reference/routines.linalg.html

7. APPENDIX

1. GitHub Project URL : https://github.com/akshaydarora/linear_algebra