Computational Mathematics for Learning and Data Analysis

Report: Project 18

Member: Pham Tran Huong Giang (606579)

TABLE OF CONTENTS

1. Problem

(P) is low rank approximation of a matrix $A \in \Re^{mxn}$, i.e.,

$$min\| A - UV \| \text{ with } U \in \Re^{mxk}, V \in \Re^{kxn}$$

(A) alternating optimization: first take an initial $V = V_0$ and compute $U_1 = argmin_U\| A - UV_0 \|$ then use it to compute $V_1 = argmin_V\| A - U_1V \|$ then $U_2 = argmin_U\| A - UV_1 \|$ and so on, until convergence. Use the normal equations with Cholesky factorization to solve these sub-problems.

2. Outline of the algorithm

This problem can be transformed to the least square problem as following:

With the first step when we want to find $U_1 = argmin_U\| A - UV_0 \|$ with $V = V_0$, that means we want to find a matrix U_1 s.t $A \approx U_1V_0$. In other word,

$$\begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_m \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ \cdots \\ u_m \end{bmatrix} * [v_1 v_2 \dots v_n]$$

with a_i is row vector with n dimension, u_i is row vector with k dimension and v_i is column vector with k dimension.

We can find matrix U_1 by finding each row of U_1 with:

$$u_iV = a_i \Leftrightarrow (u_iV)^T = a_i^T \Leftrightarrow V^Tu_i^T = a_i^T \Leftrightarrow VV^Tu_i^T = Va_i^T$$

With $M = V^T \in \Re^{nxk}, u_i^T = x, a_i^T = b$ the above equation becomes: $M^TMx = M^Tb$. This can be solved by Cholesky factorization method.

Firstly, decompose: $M^TM = R^TR$ with R is an upper triangular matrix. Then solve the lower triangular system: $R^Ty = M^Tb$ to find vector y. Finally, solve the upper triangular matrix $Rx = y$. By this method we can form U_1 row by row with m rows.

For V, carry on the same process with $A^T \approx V^TU^T$

So on, we can find $V_1, U_2, V_2 \dots$ by the same way until we reach a stage that $\| U_{t+1}V_{t+1} - U_tV_t \| \leq \epsilon$.

3. Expectation from the algorithm

Convergence

Stationary point of the subproblems

Minimizing $f(U,V) = \| A - UV \|$ that also means minimizing $g(U,V) = \frac{1}{2}\| A - UV \|^2$

When finding U to minimize $g(U) = \frac{1}{2}\| A - UV \|^2$, we have:

$\nabla g(U) = (A - UV)V^T = AV^T - UVV^T$

$\nabla^2 g(U) = VV^T$

In order to be a local minimal, U need to satisfied:

1. $\nabla g(U) = 0$
2. $\nabla^2 g(U) \succ 0.$

$\nabla g(U) = 0 \Leftrightarrow (A - UV)V^T \Leftrightarrow AV^T = UVV^T \Leftrightarrow U = AV^T(VV^T)^{-1}$ (with VV^T is positive definite). This is also the result of m least square problems stated in session 2 to find U. Besides, with VV^T is positive definite $\nabla^2 g(U) \succ 0$ already holds.

So by solving least square problems to find U, we will get at least to the local minima.

The same thing holds for V.

Convergence of the algorithm

Because $U_t = argmin_U \| A - UV_{t-1} \|$ and $V_t = argmin_V \| A - U_t V \|$ we will have a decrease sequence:

$f(U_0, V_0) \geq f(U_1, V_1) \geq \ldots \geq f(U_t, V_t).$

Since the function f calculates the norm of a suppression of matrixes A and UV, in which A has rank $r = min(m,n)$ and UV has rank k and $k < r$. So function f is bounded below by σ_{k+1} ($f_t \geq \sigma_{k+1}$) which is the $(k+1)^{th}$ singular value of A (in l2-norm, but it works for both l2-norm and Frobenius norm so we compare in l2-norm). Thus, eventually, the sequence $\{f_t\}$ will converge to a limit point $\bar{f}$.

Indeed, in [1] and [2], L. Grippo , M. Sciandrone proved that the result of this approach will converge to a critical point under some assumptions as below:

1. *Every subproblem has an optimal solution*
2. *The sequence $\{(U_tV_t)\}$ generated by the algorithm admits a limit point.*

If our problem satisfies these conditions then we can reach a stationary point of f even though function f is not convex according to **Corollary 2 in [1]**:

Suppose that the sequence $\{x^t\}$ generated by 2-block GS method has limits points. Then every limit point $\bar{x}$ of $\{x^t\}$ is a critical point of problem minimize $f(x) = f(x_1, x_2)$ with $x \in (X_1 \times X_2)$.

Let us examine whether our problem satisfies the above conditions:

1. Every subproblem has an optimal solution

   As mentioned in the previous subsection, we find the solution for subproblem (assume U) by minimizing $g(U) = \frac{1}{2}\| A - UV \|^2$. For this, we form the matrix U row by row by solving the normal equation $M^TMx = M^Tb$ (with $M = V^T \in \mathfrak{R}^{nxk}, u_i^T = x, a_i^T = b$ are the i^{th} row of matrix U, A, respectively) for each row to minimize the function $h = min \| Mx - b \|^2$. If each x is the optimal solution for each row of U then U is also the optimal solution for this subproblem.

   $h = min \| Mx - b \|^2$ is a convex problem and always solvable so by this way the algorithm satisfies this condition.

2. The sequence $\{(U_tV_t)\}$ generated by the algorithm admits a limit point.

   For this condition, let us examine the sequence $\{(U_tV_t)\}$ generated by the algorithm.

   Assume that we have the best k-rank approximation A^k of matrix A and $A^k = \bar{U}\bar{V}$.

   Start with an initial U_0V_0, we will have a sequence $U_0V_0, U_1V_1, U_2V_2, \ldots U_tV_t$ with:

   $U_0V_0 = A_0, U_1V_1 = A_1, \ldots U_tV_t = A_t$ and $f_0 = \| A - A_0 \|$, $f_1 = \| A - A_1 \|$, … and, $f_t = \| A - A_t \|$.

   As we mentioned before, f is bounded below by σ_{k+1} (in l2-norm) which is the $(k+1)^{th}$ singular value of A and also $\sigma_{k+1} = \| A - A^k \|_2$.

That means $lim_{t\to\infty} f_t = \| A - A_t \| = \| A - A^k \| \Rightarrow lim_{t\to\infty} A_t = A^k \Rightarrow lim_{t\to\infty} U_t V_t = \bar{U}\bar{V}$.

Thus, the sequence $\{(U_t V_t)\}$ generated by the algorithm has at least a limit point $\bar{U}\bar{V}$ which make our problem satisfies this condition.

Since the algorithm meets all the conditions of **Corollary 2 in [1],** we can say that the result obtained by this algorithm will converge a stationary point of f.

However, at this point we cannot say anything about where the function will converge (a global minima, a local minima or even worse a saddle point) because we cannot be sure that the Hessian of function $g(U,V) = \frac{1}{2}\| A - UV \|^2$ which in the form $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$ is positive definite or not.

Convergence rate

For the convergence rate, in **Chapter 7 of [4]**, Yuejie Chi, Yue M. Lu and Yuxin Chen mentioned that the Alternating Minimization algorithm obtains ϵ-accuracy within $O(log(\frac{1}{\epsilon}))$ step.

To be more specific, the authors of [4] claim that if we update two matrix U, V sequentially as follow:

$U = argmin_U \| \alpha(A - UV) \|_F^2$

$V = argmin_V \| \alpha(A - UV) \|_F^2$

with α is linear operator satisfies the Restricted Isometry property (RIP)

$(1 - \delta_r) \| X \|_F^2 \leq \| \alpha(X) \|_F^2 \leq (1 + \delta_r) \| X \|_F^r$ with $\delta_r < 1$ is a constant,

then the Alternating Minimization algorithm will converge with the rate $O(log(\frac{1}{\epsilon}))$.

In our case, α is identity (that means $\alpha(M) = M$). The above property hold with:

$(1 - \delta_r) \| X \|_F^2 \leq \| X \|_F^2 \leq (1 + \delta_r) \| X \|_F^r$

Thus our algorithm will converge at the rate of $O(log(\frac{1}{\epsilon}))$.

Complexity

For each iteration (of updating both updating both U and V):

Firstly, before processing Cholesky factorization, we have to calculate:

(i) $M^T M$, that costs $O(nk^2)$ and $O(mk^2)$ for processing V and U, respectively,

(ii) then compute Cholesky factorization 2 times, costs $\frac{1}{3}k^3 + \frac{1}{3}k^3 = \frac{2}{3}k^3$

Then, we have to solve $(m+n)$ least square problems $M^T Mx = M^T b$ after have Cholesky Factorization $M^T M = R^T R$. To solve this, we need to calculate $M^T b$, $R^T y = M^T b$ and $Rx = y$. For each least square problem

- $M^T b$ cost $O(kn)$ for U and $O(km)$ for V
- Solving y from $R^T y = M^T b$ with R^T lower triangular matrix costs $O(k^2)$
- Solving $Rx = y$ also costs $O(k^2)$

To finding U, we need to solve m least square problems, in total it costs $O(mk^2 + mnk)$

And n least square problems for V costs $O(nk^2 + mnk)$

Thus, each iteration of alternating algorithm (updating both U and V) costs $O(nk^2 + mk^2 + k^3 + mnk)$ in total.

Stability

We are using Cholesky Factorization to solve the least square subproblems so in this part we will focus on the stability of using Cholesky Factorization to solve the least square problems. In [3], Higham worked on error analysis of Cholesky factorization and he proposed two theorems:

Theorem 10.3 in [3]*: If Cholesky factorization applied to the symmetric positive definite matrix $A \in \mathfrak{R}^{nxn}$ runs to completion then the computed factor $\widehat{R}$ satisfies:*

$$\widehat{R}^T \widehat{R} = A + \Delta A, |\Delta A| \leq \gamma_{n+1} |\widehat{R}^T| |\widehat{R}|$$

Theorem 10.4 in [3]*: Let $A \in \mathfrak{R}^{nxn}$ be symmetric positive definite and suppose Cholesky factorization produces a computed factor $\widehat{R}$ and a computed solution $\hat{x}$ to $Ax = b$. Then:*

$$(A + \Delta A)\hat{x} = b, |\Delta A| \leq \gamma_{3n+1} |\widehat{R}^T| |\widehat{R}|$$

With $\gamma_n = \frac{nu}{1-nu}$ where u is unit rounding ($\epsilon_{machine}$) and with an implicit assumption that $nu \leq 1$.

Thus, Cholesky factorization is backward stable. Next, we will examine the stability of solving the least square problems with Cholesky factorization (with reference from [5]).

By solving least square problem by normal equations $M^T Mx = M^T b$, we have the matrix $M^T M$, this matrix has the condition number of κ^2, not just κ. Thus, even Cholesky factorization is backward stable, normal equation is always gives the relative error:

$$\frac{\| \tilde{x} - x \|}{\| x \|} = O(\kappa^2 \epsilon_{machine})$$

3. Input data

For the input data we need a full rank matrix A, which can be generated randomly.

We employ Singular value decomposition to make sure matrix A is full rank. That means, we will generate 2 orthogonal matrixes $U \in \mathfrak{R}^{mxm}$, $V \in \mathfrak{R}^{nxn}$ and a diagonal matrix $\Sigma \in \mathfrak{R}^{mxn}$. The entries on the diagonal of Σ are ordered decreasingly.

Then we will have $A = U \Sigma V^T$.

By this way, we have the information about the $(k+1)^{th}$ singular value σ_{k+1} of matrix A to verify our result in next step.

4. Implementation

In this section, we process to find the general formula for each sub-task and then from these formulas we come up with the pseudocode for each task as describe in the Appendix.

Cholesky factorization

From a square symmetric matrix $X \in \mathfrak{R}^{(k+1)*(k+1)}$ (in this section, we use $k+1$ instead of k for the sake of array index representation in python) we decompose $X = R^T R$. Thus,

$$\begin{bmatrix} x_{00} & x_{01} & x_{02} \dots & x_{0k} \\ x_{10} & x_{11} & x_{12} \dots & x_{1k} \\ x_{20} & x_{21} & x_{22} \dots & x_{2k} \\ & & \dots & \\ x_{k0} & x_{k1} & x_{k2} \dots & x_{kk} \end{bmatrix} = \begin{bmatrix} r_{00} & 0 & 0 & \dots 0 \\ r_{01} & r_{11} & 0 & \dots 0 \\ r_{02} & r_{12} & r_{22} & \dots 0 \\ & & \dots & \\ r_{0k} & r_{1k} & r_{2k} \dots & r_{kk} \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \dots & r_{0k} \\ 0 & r_{11} & r_{12} \dots & r_{1k} \\ 0 & 0 & r_{22} \dots & r_{2k} \\ & & \dots & \\ 0 & 0 & 0 \dots & r_{kk} \end{bmatrix}$$

That means:

$$r_{00}^2 = x_{00} \quad r_{00} * r_{01} = x_{01} \quad r_{00} * r_{02} = x_{02} \quad \dots \quad r_{00} * r_{0k} = x_{0k}$$

$$r_{01}^2 + r_{11}^2 = x_{11} \quad r_{01} * r_{02} + r_{11} * r_{12} = x_{12} \ \ldots \quad r_{01} * r_{0k} + r_{11} * r_{1k} = x_{1k}$$

$$r_{02}^2 + r_{12}^2 + r_{22}^2 = x_{22}; \ \ldots \qquad r_{02} * r_{0k} + r_{12} * r_{1k} + r_{22} * r_{2k} = x_{2k}$$

$$\ldots$$

$$r_{0k}^2 + r_{1k}^2 + .. + r_{kk}^2 = x_{kk}$$

In general:

$$x_{ii} = \sum_{t=0}^{t=i} r_{ti}^2 \text{ and } x_{ij} = \sum_{t=0}^{t=i} r_{ti} * r_{tj}$$

$$\Leftrightarrow r_{ii} = (x_{ii} - \sum_{t=0}^{t=i-1} r_{ti}^2)^{1/2} \text{ and } r_{ij} = \frac{1}{r_{jj}}(x_{ij} - \sum_{t=0}^{i-1} r_{ti} * r_{tj})$$

See ***function chol(X)*** in the Appendix for pseudocode.

Solving lower triangular system

We have lower triangular system $R^T y = M^T b = v$ with $R^T \in \Re^{(k+1)*(k+1)}$ is a lower triangular matrix, and vector $v \in \Re^{(k+1)*1}$, that means:

$$\begin{bmatrix} r_{00} & 0 & 0 & \ldots 0 \\ r_{01} & r_{11} & 0 & \ldots 0 \\ r_{02} & r_{12} & r_{22} & \ldots 0 \\ & & \ldots & \\ r_{0k} & r_{1k} & r_{2k} & \ldots r_{kk} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \ldots \\ y_k \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \ldots \\ v_k \end{bmatrix}$$

$$y_0 = \frac{v_0}{r_{00}} \qquad y_1 = \frac{v_1 - r_{01} * y_0}{r_{11}} \qquad y_2 = \frac{v_2 - (r_{02} * y_0 + r_{12} * y_1)}{r_{22}}$$

$$y_k = \frac{v_k - (r_{0k} * y_0 + r_{1k} * y_1 + .. + r_{k-1,k} * y_{k-1})}{r_{kk}}$$

Thus, in general:

$$y_i = \frac{v_i - \sum_{t=0}^{i-1} r_{ti} * y_t}{r_{ii}}$$

See ***function slts(R, v)*** in the Appendix for pseudocode.

Solving upper triangular system

After finding vector y by solving lower triangular system, we need to solve the upper triangular system: $Rx = y$ with $R \in \mathfrak{R}^{(k+1)*(k+1)}$ is an upper triangular matrix.

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} \dots & r_{0k} \\ 0 & r_{11} & r_{12} \dots & r_{1k} \\ 0 & 0 & r_{22} \dots & r_{2k} \\ & & \dots & \\ 0 & 0 & 0 \dots & r_{kk} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_k \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \dots \\ y_k \end{bmatrix}$$

We have:

$$x_k = \frac{y_k}{r_{kk}} \qquad x_{k-1} = \frac{y_{k-1} - x_k * r_{k-1,k}}{r_{k-1,k-1}} \qquad \dots \; x_1 = \frac{y_1 - (r_{1k} * x_k + r_{1,k-1} * x_{k-1} + .. + r_{12} * x_2)}{r_{11}} \text{ and}$$

$$x_0 = \frac{y_0 - (r_{0k} * x_k + r_{0,k-1} * x_{k-1} + .. + r_{01} * x_1)}{r_{00}}$$

In general:

$$x_i = \frac{y_i - \sum_{t=k}^{i+1} r_{it} x_t}{r_{ii}}$$

See ***function suts(R, y)*** in the Appendix for pseudocode.

Find the solution for subproblem

Next step, we will use Cholesky factorization, Lower and Upper triangular system to solve the subproblem $U_t = argmin_U \| A - UV_{t-1} \|$ and $V_t = argmin_V \| A - U_t V \|$.

As mentioned in the second section, we can solve these problems by a linear system to find each matrix row by row.

For finding each row we carry the below flow:

- Calculate Cholesky factorization
- Solve the lower triangular system
- Solve the upper triangular system

See ***function argmin(A, M)*** in the Appendix for pseudocode.

5. Testing and result

Unit test

For testing our algorithm, firstly, we will test every single task including: function to calculate Cholesky factorization, function to solve lower triangular system and function to solve upper triangular system.

For Cholesky factorization: We randomly generate a matrix M_{test} with random size, then we calculate $X_{test} = M_{test}^T M_{test}$. By applying Cholesky factorization on X_{test} we get a lower triangular matrix R_{test}. To verify the precision of our Cholesky factorization, we calculate the norm $\frac{\| X_{test} - R_{test} R_{test}^t \|}{\| R_{test} \|}$

For the lower triangular system: We use the matrixes above to compute y_{test} such that $R_{test} * y_{test} = M_{test}^t * b_{test}$, with b_{test} is a random vector with the appropriate size. Then we calculate the norm $\frac{\| R_{test} * y_{test} - M_{test}^t * b_{test} \|}{\| y_{test} \|}$ to check the correctness of our system.

We carry out the same procedure to examine our solver of the upper triangular system $R_{test}^t * x_{test} = y_{test}$. After forming the vector x_{test} the norm $\frac{\| R_{test}^t * x_{test} - y_{test} \|}{\| x_{test} \|}$ is used to verify the precision of our calculation.

We run the test ten times and report the result in the table below:

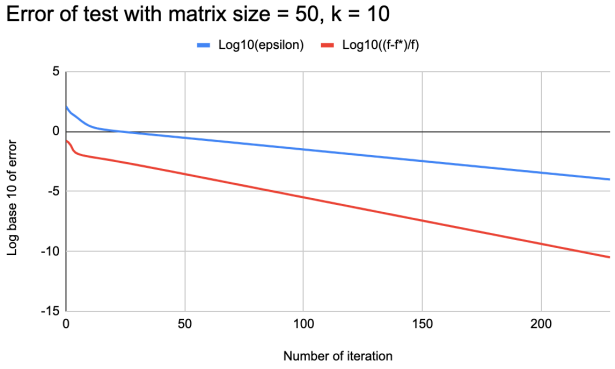
Idx	Size of matrix (M)	$\frac{\| X_{test} - R_{test} R_{test}^t \|}{\| R_{test} \|}$	$\frac{\| R_{test} * y_{test} - M_{test}^t * b_{test} \|}{\| y_{test} \|}$	$\frac{\| R_{test}^t * x_{test} - y_{test} \|}{\| x_{test} \|}$
1	859x589	9.39E-14	2.3E-13	7.34E-15
2	900x868	1.66E-13	3.37E-13	1.33E-14
3	944x131	7.23E-15	5.33E-14	1.43E-14
4	831x286	3.12E-14	1.17E-13	1.17E-14
5	655x446	6.33E-14	1.89E-13	5.53E-15
6	535x448	7.1E-14	1.68E-13	7.39E-15
7	996x766	1.58E-13	2.74E-13	1.21E-14
8	833x503	7.21E-14	2.07E-13	6.54E-15
9	756x683	1.26E-13	2.66E-13	9.45E-15
10	470x373	2.77E-14	9.98E-14	5.17E-15

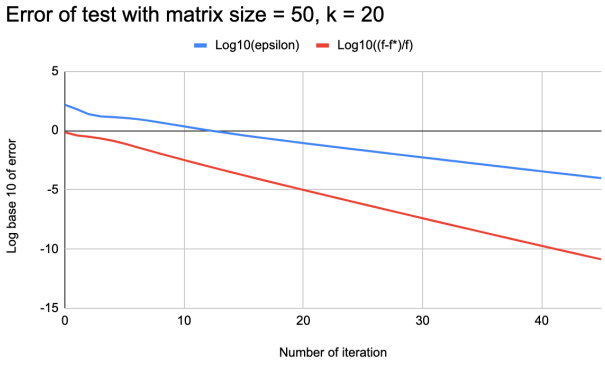
Experience and result

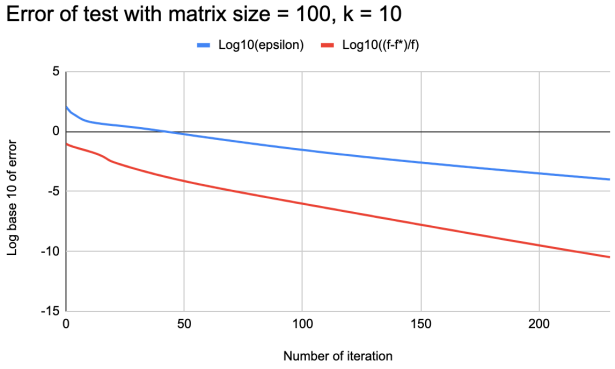
Our ultimate purpose is to find out 2 matrixes U, V such that UV will be the best $k-rank$ approximation of matrix A.

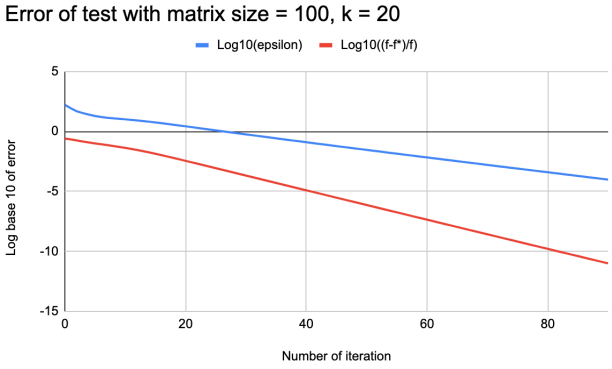
Thus, to test our program, besides checking the error $e = \| U_t V_t - U_{t-1} V_{t-1} \|$, we will also want to compare the result we get with the best $k-rank$ approximation of A so we also plot the value $\frac{f-\bar{f}}{\bar{f}} = \frac{\| A - UV \|_2 - \sigma_{k+1}}{\sigma_{k+1}}$ with σ_{k+1} is the best result we can get as proved in previous section.

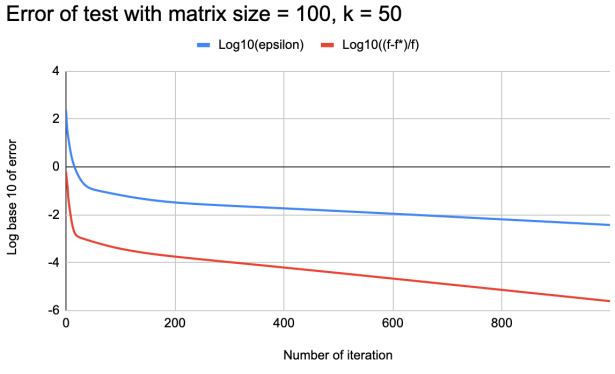
The following charts show how the error reduce with a number of tests.

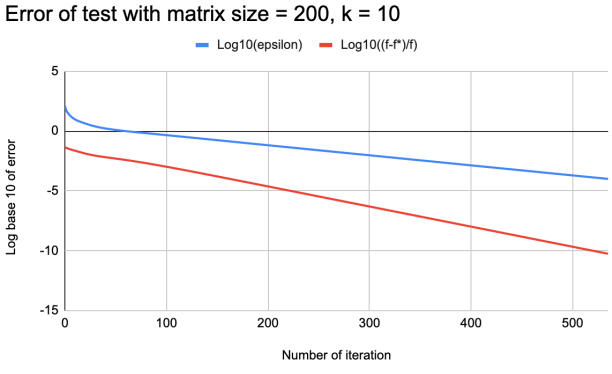


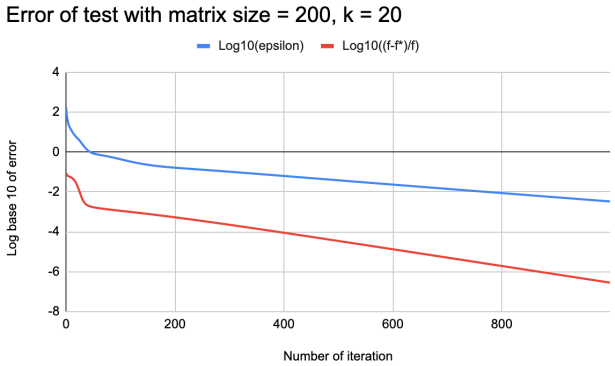


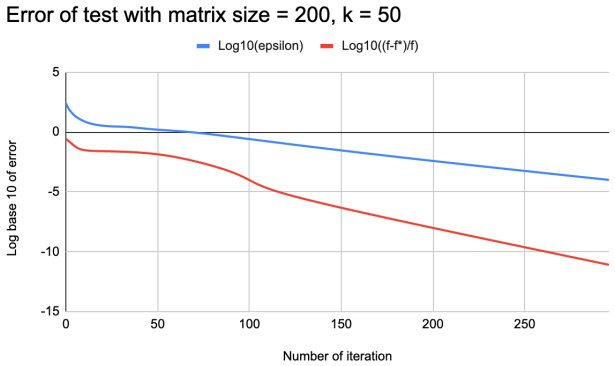


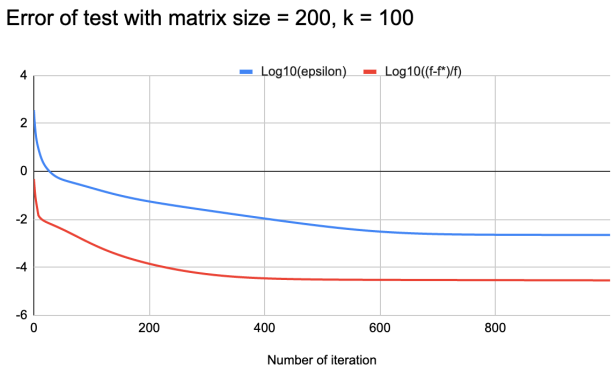


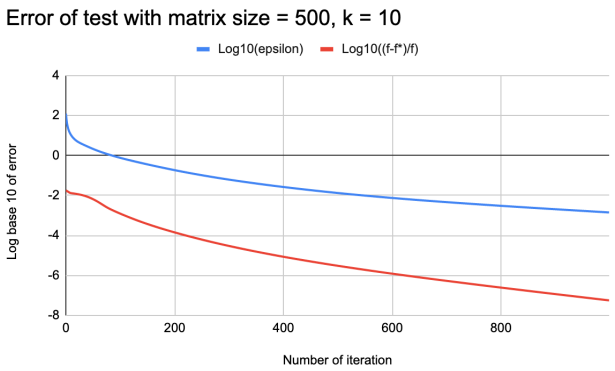


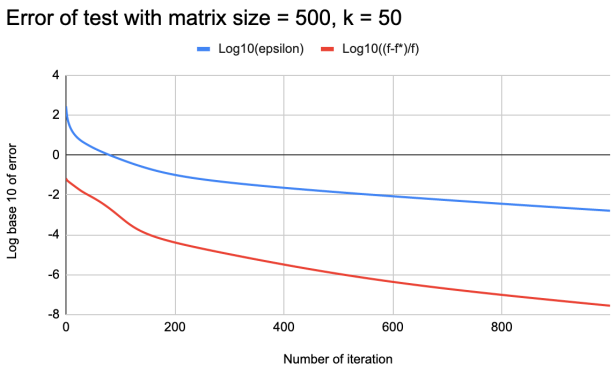


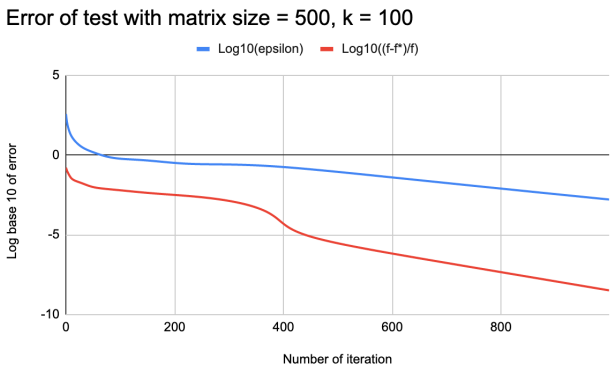


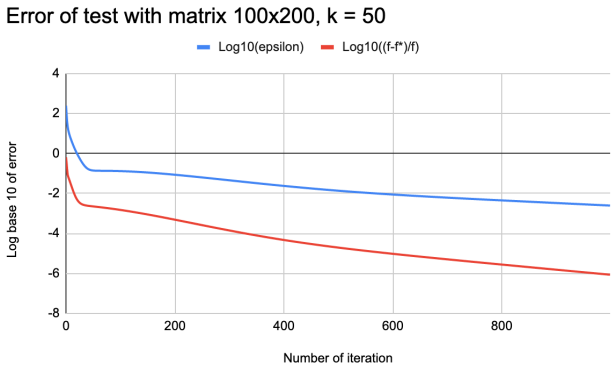


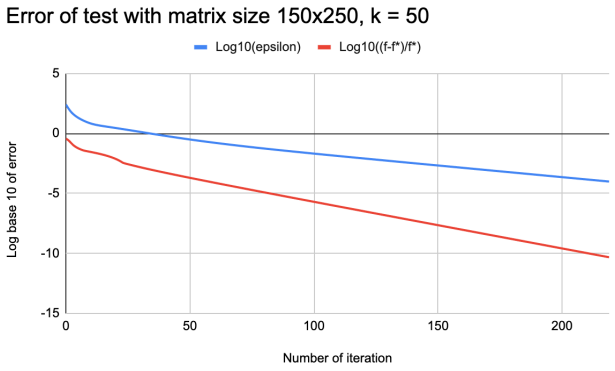












According to [4], the result will converge with the rate of $O(log(\frac{1}{\epsilon}))$, that means the error line should be straight. From these above charts, it is clearly that the general trend of the residual is a steep decrease at the first few iterations followed by a straight line in the most cases.

Running time

All the experiences in this part are conducted on the device MacBook Air 2018, with 1.6GHz Dual-Core Intel Core i5.

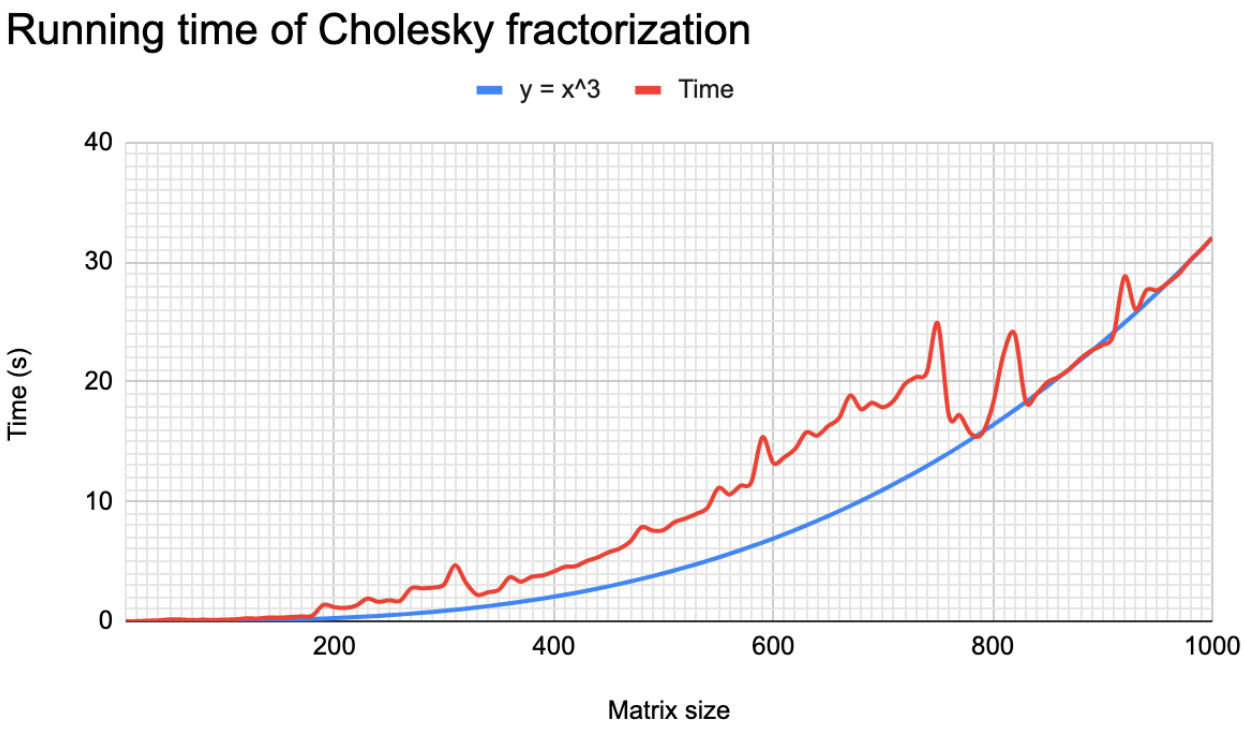
Cholesky factorization

In this subsection, we examine running time and the scale ability of Cholesky function.

We test our function with matrixes of different size in range from 10, 20, 30 … to 1000. With each size we run the function 10 times and measure the average running time. The result is shown in the figure below:

The red line represents the running time of our Cholesky function.

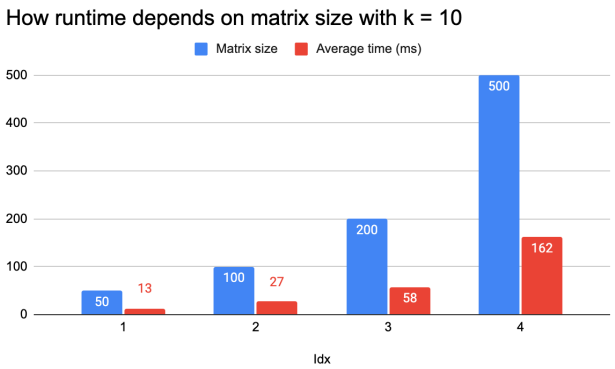
The blue line shows the function $y = x^3$ in appropriate range and scale in relative to our data. The

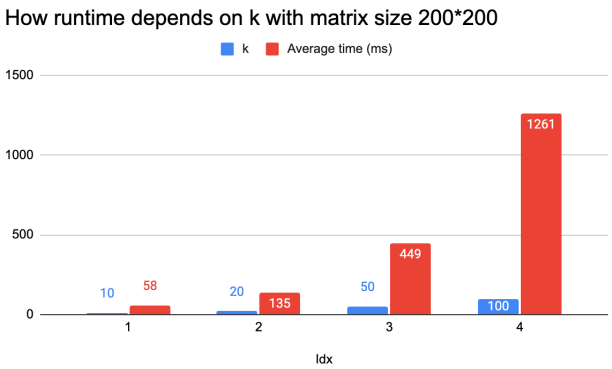


complexity of Cholesky algorithm is $O(n^3)$ so we plot this line here to give the reader a glance on the comparison.

Time of each iteration

The charts below show how the time of each iteration depends on the size of the matrix and number k.





6. Reference

1. L. Grippo , M. Sciandrone. On the convergence of the block nonlinear Gauss–Seidel method under convex constraints, September, 1999.

2. Luigi Grippo and Marco Sciandrone. Globally convergent block-coordinate techniques for unconstrained optimization, February, 1998.

3. Nicholas J. Higham. Accuracy and Stability of Numerical Algorithms, second edition.

4. Yuejie Chi, Yue M. Lu and Yuxin Chen. Nonconvex Optimization Meets Low-Rank Matrix Factorization: An Overview, September, 2019.

5. Lloyd N. Trefethen, David Bau III. Numerical Linear Algebra, SIAM, 1997.

Appendix A: Pseudocode

Cholesky factorization:

function chol(X):

for i from 0 to k :

for j from 0 to i - 1:

sum = 0

for t from 0 to j - 1:

$sum \mathrel{+}= r_{it} * r_{jt}$

$$r_{ij} = \frac{1}{r_{jj}}(x_{ij} - sum)$$

sum = 0

for t from 0 to i - 1:

$sum \mathrel{+}= r_{it}^2$

$$r_{ii} = sqrt(x_{ii} - sum)$$

Lower triangular system

function slts(R, v):

for i from 0 to k:

sum = 0

for t from 0 to i - 1:

$sum \mathrel{+}= r_{it} * y_t$

$$y_i = \frac{v_i - sum}{r_{ii}}$$

Upper triangular system

function suts(R, y):

for i from k to 0:

sum = 0

for t from k to i + 1:

$sum \mathrel{+}= r_{it} * x_t$

$$x_i = \frac{y_i - sum}{r_{ii}}$$

Solution for subproblem

function argmin(A, M):

N: matrix consists of m rows

$X = M^T M$

$R = chol(X)$ *# lower triangular matrix kxk*

for i from 0 to m - 1:

$b = A_{i,0:n-1}$

$y = slts(R, M^T b)$ *# vector kx1*

$x = suts(R^T, y)$ *# vector kx1*

$N_{i,0:k-1}$ = x

Appendix B: How to run the code

We implement the algorithm in python language with jupyter notebook.

The notebook contains many cells (which are marked in order), in which the first 5 cells are the sub tasks of the algorithm (Cholesky factorization, lower triangular system, upper triangular system…). The main part of the algorithm is in the 6th cell.

In the 7th cell, we generate the input data (matrix A) for the function call of the algorithm in 8th cell.

To run the 8th cell, you should run all the first 7 cells first. In the 8th cell, you can set the value for m, n, k and then run the algorithm.

The output of the cell will show:

- k^{th} singular value of matrix (A)
- #iteration, epsilon and $\| A - UV \|_2$
- Total number of iterations until the algorithm reaches the stopping condition
- The final value of epsilon
- Average time of each iteration
- The final value of $\| A - UV \|_2$

All the other cells are for testing purpose.

Appendix C: Verify the complexity of Cholesky factorization

In this appendix, we will verify the complexity of our algorithm to calculate Cholesky factorization From the pseudo code above:

1. function chol(X):

2. *for i from 0 to k :*

3. *for j from 0 to i - 1:*

4. *sum = 0*

5. *for t from 0 to j - 1:*

6. $sum \mathrel{+}= r_{it} * r_{jt}$

7. $r_{ij} = \frac{1}{r_{jj}}(x_{ij} - sum)$

8. *sum = 0*

9. *for t from 0 to i - 1:*

10. $sum \mathrel{+}= r_{it}^2$

11. $r_{ii} = sqrt(x_{ii} - sum)$

To simplify, we only count the number of multiplication and addition in the for loop of line 5 because this is the heaviest work of this algorithm.

For each value of j in line 3, we have to calculate j multiplication and j addition with $j = 0 : i - 1$ so we will need to calculate $\sum_{j=0}^{i-1} j = \frac{i * (i-1)}{2}$ multiplications and $\sum_{j=0}^{i-1} j = \frac{i * (i-1)}{2}$ additions,

so for both, we need to calculate: $i * (i - 1)$ operators.

In most outer for loop in line 2, we have $i = 0 : k$, thus in total we need to calculate $\sum_{i=0}^{k} i(i-1)$

operations.

With, $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ we can come to a conclusion that the complexity of our algorithm is $O(\frac{1}{3}n^3)$.