CENG327 Project – Numeric solution of sparse linear systems with Kaczmarz and Cimmino methods

Aim: Analyzing the performance of numeric solution methods for different sparse linear systems in different real-world problems

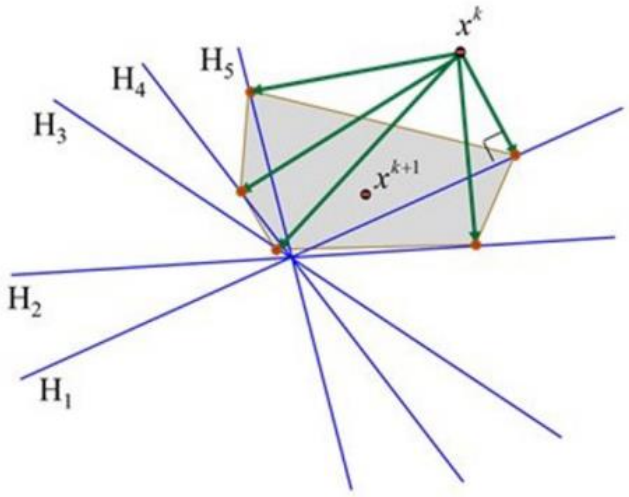
Due Date: 24/12/2023 @23:55

In this project, you will analyze the execution time of different solution methods for linear systems of equations. For a set of systems from the SuiteSparse Matrix Collection (https://sparse.tamu.edu/), you must use different solvers in the Scipy package and compare the running times of them with the running time of your method that you implemented in Python.

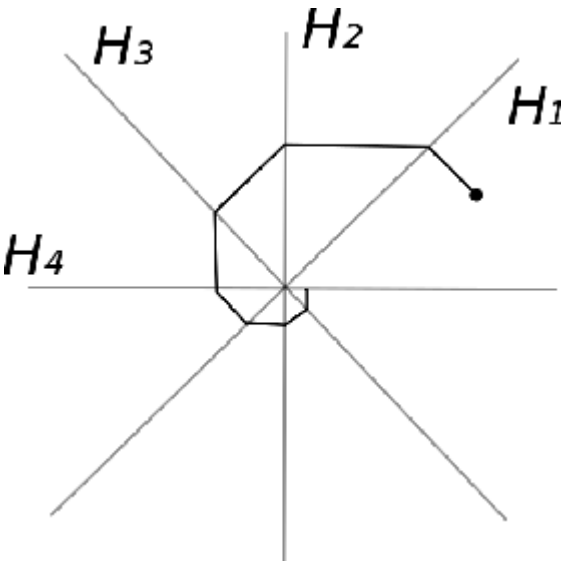
For solving linear systems of equations, there are mainly two categories: direct methods and iterative methods. The hybrid method is a new category that emerges later. Hybrid methods aim to leverage the advantages of direct methods and iterative methods. The block Kaczmarz and block Cimmino methods are examples of the hybrid methods which have increasing popularity in the scientific computing society.

In the project, you will implement the block Kaczmarz and block Cimmino methods by using the Python programming language. In these methods, the number of iterations for the convergence is affected by the number of blocks (parts) used in the methods. After implementing the method, you need to find the optimum number of parts that gives the lowest execution time for each system specifically. You will plot the behavior of the methods for each matrix to see the execution time changes with respect to different numbers of parts.

Block Kaczmarz and block Cimmino are row projection methods that requires the computation of the orthogonal projections $P_i x = A_i (A_i^T A_i)^{-1} A_i^T x$ of a vector x of onto range (A_i), for i=1,2, …, m. By using these projections, the solution is updated iteratively. The next figure shows a simpler case in the 2-D domain where projections onto 5 subspaces and the updated solution (x^{k+1}) of the next iteration are shown.



For Kaczmarz, iterations can be shown as in 2 dimension :



In both methods, a linear system of equations of the form

$$Ax = f$$

is divided into K row blocks as follows:

$$\begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_K \end{pmatrix} x = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_K \end{pmatrix}$$

After the partitioning, the method solves K independent systems to obtain projections individually and sums up the projections to obtain a better approximation for the solution. The next figure shows the main steps of the block Cimmino algorithm.

1: Choose $x^{(0)}$
2: **while** $t = 0, 1, 2, \ldots,$ until convergence **do**
3: **for** $i = 1, \ldots, K$ **do**
4: $\delta_i = A_i^+(f_i - A_i x^{(t)})$
5: **end for**
6: $x^{(t+1)} = x^{(t)} + \omega \sum_{i=1}^{K} \delta_i$
7: **end while**

Block Kaczmarz algorithm: The block Kaczmarz method obtains its next iterate through a product of projections of the current iterate such that, we have the following computation.

$$\delta^{(i+1)} = \delta^{(i)} + A_i^+ \left(b_i - A_i \delta^{(i)}\right) \quad \text{for } i = 1 \text{ to } p$$

where $\delta^{(1)} = x^{(k)}$ and A_i^+ is the Moore-Penrose pseudoinverse of A_i. Then the next iterate is computed as

$$x^{(k+1)} = \delta^{(p+1)}.$$

In the algorithms, A_i^+ and δ_i show the Moore–Penrose pseudoinverse of A_i and the computed projection for A_i, respectively. ω is a relaxation parameter, you can take it one.

If there are fewer equations than unknowns, the system is called underdetermined. For underdetermined systems like $A_i\delta_i = f_i - A_i x$, there are infinitely many solutions that satisfy this system for different δ_i. Only the minimum 2-norm solution leads to a solution in the block Cimmino method. There are many ways to compute the minimum norm solution.

1. QR factorization:

$$A_i^T = QR$$

$$(QR)^T x = f_i \Rightarrow R^T Q^T x = f_i \Rightarrow Q^T x = (R^T)^{-1} f_i \Rightarrow x = Q\,(R^T)^{-1} f_i$$

First you need to solve $(R^T)^{-1} f_i$ then multiply the solution with Q

2. Seminormal equations

$$A_i^T = QR$$

$$x = A_i^T\ (R^T R)^{-1} f_i$$

First you need to solve $(R^T R)^{-1} f_i$ then multiply the solution with A_i^T.

3. Normal equations

$$x = A_i^T (A_i A_i^T)^{-1}\, f_i$$

First you need to solve $(A_i A_i^T)^{-1} f_i$ then multiply the solution with A_i^T.

4. With an augmented system approach

$$\begin{pmatrix} I & A_i^T \\ A_i & 0 \end{pmatrix} \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \begin{pmatrix} 0 \\ r_i \end{pmatrix}$$

then the solution of the above system gives $\delta_i = u_i = A_i^T (A_i A_i^T)^{-1} r_i$ and $v_i = -(A_i A_i^T)^{-1} r_i$, where $r_i = (f_i - A_i\, x)$. Here, you do not need v_i , you only need the upper part of the solution vector, that is u_i in the above system.

The algorithm: after you create the augmented system, you can use a sparse linear solver, such as spsolve of scipy to compute the solution of the square augmented system.

In fact, here x is the solution of the optimization problem: minimize ||x|| subject to Ax = y .

Steps for Linear system solvers:

1. Download ` poisson3Da` problem from the SuiteSparse Collection. It is a computational fluid dynamics problem, and you can find the RHS of the system in the file 'poisson3Da_b.mtx'
2. With the following code you can read the matrix and RHS from python

```
from scipy.io import mmread
A = mmread('poisson3Da.mtx')
f = mmread('poisson3Da_b.mtx')
```

3. ‘A’ is a sparse matrix in COOrdinate format and ‘f’ is a ndarray. By using ‘A’ and ‘f’ you need to find x that satisfies Ax=b.
4. Here you must measure the time and relative residuals of different methods with different parameters (in iterative methods, you can set maximum number of iterations to 1000). relative residual: ||Ax-f|| / ||f||, threshold = 1e-5
    a. Solve the system with spsolve (sparse LU) with the following fill-in reducing orderings: NATURAL, MMD_ATA, COLAMD
    b. Solve the system with QR factorization. You can use the dense QR method as follows or you can use a wrapper if spqr for sparse QR factorization is installed in your computer.

```
Q, R = linalg.qr(A.todense()) # QR decomposition with qr function
y = np.dot(Q.T, B) # Let y=Q'.B using matrix multiplication
x = linalg.solve(R, y)
```

    c. Solve the system with bicg-stab
    d. Solve the system with ilu preconditioned bicg-stab

```
import scipy.sparse.linalg as sla
B = sla.spilu(A, drop_tol=1e-12, fill_factor=1)#,
permc_spec='NATURAL')
Mz = lambda r: B.solve(r)
Minv = sla.LinearOperator(A.shape, Mz)
bicg-stab(A, f, …..,  M=Minv)
```

5. Make a table that reports your finding of time and residual for each method. If a method gives an error you should note this in the table.
6. Run your block Kaczmarz and block Cimmino algorithms with K=2,4,6 and 8. Measure the time and residual for each K.
7. Compare and analyze your results with the results of one of the below Scipy methods.
   Checking methods:
    a. least square method of scipy
    b. pseudo-inverse of scipy

8. Fit a polynomial for your results in block Cimmino without K=6, and estimate the time result for K=6. How far is your estimation from the actual result of K=6?
9. Fit a polynomial for your results in block Cimmino including all results, and extrapolate the time result for K=16. How far is your estimation from the actual result of K=16? Here, you need to run your algorithm for K=16 as well.
10. Apply the steps [6-9] for the second matrix whose name is given next to your project group for finding the minimum 2-norm solution of an underdetermined system in block Cimmino.

Questions:

1- What are the advantages and disadvantages of direct methods for solving linear equations?
2- Give 3 examples of direct methods and compare their properties with one another.
3- What are the advantages and disadvantages of iterative methods for solving linear equations?

4- Give 3 examples of iterative methods and compare their properties with one another.