Activity 05 : *Numerical resolution of linear Systems*

Gauss Method & Gauss-Seidel Method

1. Gauss elimination method :

a. Within your folder, create a new notebook and name it: ***TP05.ipynb***

b. Start your notebook by the first cell of importing necessary modules:

```python
import numpy as np
import matplotlib.pyplot as plt
```

c. After that, within a new cell code, write the program allowing to proceed to the gauss elimination of a linear system $\boldsymbol{A}.\boldsymbol{x} = \boldsymbol{y}$, where $A = n \times n$ matrix, and y is a constant vector with n components

```python
def gauss_elimination(A, y):
    A = np.array(A, dtype=float) # A matrix of dimension n x n
    y = np.array(y, dtype=float) # y The constant vector dim(y)=n
    n = len(y)
    M = np.hstack((A, y.reshape((n, 1)))) # Form the augmented matrix M = [A|y]
    print('the new matrix M = [A|y] = \n', M, '\n')

    for i in range(n): # Forward Elimination
        # Find pivot row
        if M[i, i] == 0:
            print("Error: Pivot element is zero")
            return None
        for j in range(i + 1, n): # the factor to eliminate from the current element
            factor = M[j, i] / M[i, i]
            M[j, :] -= factor * M[i, :] # Subtract 'factor' times the current row
    # Back Substitution
    x = np.zeros(n, dtype=float)
    for i in range(n - 1, -1, -1):
        sum_ax = np.dot(M[i, i+1:n], x[i+1:n]) # Calculate sum of terms with known x values
        x[i] = (M[i, n] - sum_ax) / M[i, i] # Solve for x[i]
    return x
```

d. Apply the method on a given linear system

```python
# Coefficients matrix A
A = [[4, 1, 2],
     [3, 5, 1],
     [1, 1, 3]]

# Constants vector y
y = [4, 7, 3]

# Solve the system
x = gauss_elimination(A, y)
print('Solution is x =', x)
```

Activity 05 : *Numerical resolution of linear Systems*

Gauss Method & Gauss-Seidel Method

2. **Gauss-Seidel Method :** to use the iterative Gauss-Seidel method:

$$\boxed{x_i = \frac{1}{a_{i,i}}\left[y_i - \sum_{j=1,j\neq i}^{j=n} a_{i,j}x_j\right]}$$

a. First, we need to check if the matrix A is diagonally dominant, using the following code

```python
def diag_dominant(A):
    A = np.abs(np.asarray(A))
    diagonal = np.diag(A) # Extract diagonal elements
    others_sum = np.sum(A, axis=1) - diagonal # Sum off-diag elements - diagonal
    if np.all(diagonal >= others_sum) == True:
        print(' A is diagonally dominant')
    else :
        print(' A is not diagonally dominant')
```

b. Then, let's define the Gauss-Seidel method with python:

```python
def gauss_seidel(A, y, e=1e-8, max_iter=100):
    A = np.array(A, dtype=float) # A matrix of dimension n x n
    y = np.array(y, dtype=float) # y The constant vector dim(y)=n
    n = len(y) # dimension of y
    x = np.zeros(n, dtype=float) # initial guess

    for k in range(max_iter):
        x_old = x.copy()
        # Loop over rows (equations)
        for i in range(n):
            # Calculate the sum of terms for other variables using the most recent x values
            sum_except_i = np.dot(A[i, :i], x[:i]) + np.dot(A[i, (i+1):], x_old[(i+1):])
            # Update x[i] using the Gauss-Seidel formula
            x[i] = (y[i] - sum_except_i) / A[i, i]
        # Check for convergence (L-infinity norm of the difference)
        max_diff = np.max(np.abs(x - x_old))
        if max_diff < e:
            print(f"Converged after {k + 1} iterations.")
            return x
    print(f"Did not converge within {max_iter} iterations.")
    return x
```

c. Finally check the method with the same example use in the first section

```python
A = [[4, 1, 2], [3, 5, 1], [1, 1, 3]]
y = [4, 7, 3]

# Solve the system
x = gauss_seidel(A, y)
print("Solution:", x)
```