Numerik ÜB 2

👥 Pauline Thiele, Konstantin Regenhardt, Johannes Malinowski, Sophie Mena-Chavez, Nina Böhm

Multiplikation

```
#Multipliziert 2 Matrizen und überprüft auf Gültigkeit der Dimensionen
    def get_products(self, matrix_a, matrix_b):
        rows_a = len(matrix_a)
        cols_a = len(matrix_a[0])
        rows_b = len(matrix_b)
        cols_b = len(matrix_b[0])

        if cols_a != rows_b:
            message = "Matrizen können nicht multipliziert werden. Inkorrekte Dimensionen."
            return message

        result_matrix = [[0 for row in range(cols_b)] for col in range(rows_a)]

        for i in range(rows_a):
            for j in range(cols_b):
                for k in range(cols_a):
                    result_matrix[i][j] += matrix_a[i][k] * matrix_b[k][j]
        return result_matrix
```

Gauss

```
"""Die Funktion lu_decomposition zerlegt eine übergebene A Matrix in eine L und R Matrix und rundet
    das Ergebnis auf die 4. Nachkommastelle"""
    def lu_decomposition(self, a_matrix):
        # Überprüft, ob die übergebene Matrix quadratisch ist
        n = len(a_matrix)
        for x in range(n):
            m = len(a_matrix[x])
            if m != n:
                raise ValueError("Die Matrix ist nicht quadratisch.")

        # Erstellt eine leere untere Dreiecksmatrix
        l_matrix = [0.] * n
        for y in range(n):
            l_matrix[y] = [0.] * n
        for z in range(n):
            l_matrix[z][z] = 1.

        # Zerlegt a_matrix in L und R Matrix mit Typ 3 der Elementare Zeilenumformungen
        # (R Matrix im Code weiterhin als a_matrix bezeichnet)
        row, column = 0, 0
        while row < n and column < n:
```

```
            for i in range(row + 1, n):
                l_matrix[i][column] = round(a_matrix[i][column] / a_matrix[row][column], 4)
                # Die Rechnung vor der For-Schleife wir mit Hilfe der Funktion pair_items
                # auf die zu modifizierende Zeile angewandt.
                a_matrix[i] = [round(modified_row - l_matrix[i][column]
                               * pivot_row, 4) for modified_row, pivot_row in self.pair_items(a_matrix[i], a_matrix[row])]
            row += 1
            column += 1

        joint_matrix = self.util.add_matrices(l_matrix, a_matrix)
        return l_matrix, a_matrix, joint_matrix


    """pair_items verbindet das i-te Item aus einer Liste part1 mit dem i-ten Item aus einer Liste part2, und giebt
     das Ergebnis als einzelne Liste zurück. Die Einzelnen Listen stehen jeweils für eine Reihe aus der A Matrix"""
    def pair_items(self, part1: tuple, part2: tuple):
        if len(part1) >= len(part2):
            count = len(part2)
        else:
            count = len(part1)

        result = []
        for i in range(count):
            temp = [part1[i], part2[i]]
            result.append(temp)

        return result
```

Addition

```
def add_matrices(self, matrix1, matrix2):
        result = [[0]*len(matrix1) for i in range(len(matrix1))]
        for row in range(len(matrix1)):
            for col in range(len(matrix1)):
                if row == col:
                    result[row][col] = matrix2[row][col]
                else:
                    result[row][col] = matrix1[row][col] + matrix2[row][col]
        return result
```