PA3 Part C Documentation

Program Description

Decomposes a matrix into its R, Λ, and R^{-1} components and checks if their product results in the original matrix. R^T will not be used for the decomposition as [it only works for real symmetric matrices].

Important Library Details

- Eigen
  - Library path: the headers for the Eigen library are located in /usr/include/eigen3 on my Linux machine.
  - Library version: I have installed Eigen version 3.4.0.

Marginal Cases

- Invalid inputs:
  - The quadratic determinant of the characteristic polynomial of the matrix is less than 0 results in no real solutions.
  - The eigenvalues are 0: supposedly eigenvectors can be calculated in this case, however I don't know how to.
- Invalid computations:
  - All important computations in the Eigen implementation methods were handled by Eigen, and the outputs have been checked.
  - All custom implementation methods simply added two doubles and stored them in the corresponding double element of a result matrix. The outputs have also been checked.

Design Choices

- The characteristic polynomial of the matrix($p(\lambda) = a\lambda^2 + b\lambda + c$ where $a = 1$, $b = -a_{11} - a_{22}$ and $c = a_{11}a_{22} - a_{12}a_{21} = det(A)$) will be used to calculate the eigenvalues and invalid solutions.
- The quadratic determinant of the characteristic polynomial($d = b^2 - 4ac$) will determine the output of the program:
  - Quadratic determinant is less than 0: No eigen decomposition for A exists.
  - Quadratic determinant is 0: Both eigenvalues are equal to $\frac{-b}{2a}$.

- ○ Quadratic determinant is greater than 0: Eigenvalues are different and will be calculated with $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$(or since d is already calculated, $\frac{-b\pm\sqrt{d}}{2a}$), then sorted for dominance.
- To solve for eigenvectors, the equation $(A - \lambda_n I)\,\vec{r}_n = \vec{0}$ where $\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$ will be used. The homogeneous system solver from part A of this assignment will be used for each eigenvector before assembling them into an eigen matrix: $R = \begin{bmatrix} \vec{r}_1 & \vec{r}_2 \end{bmatrix}$.
  - ○ Optimization: for cases where the quadratic determinant is 0, only the linear system of one vector needs to be solved(this will be $\vec{r}_1$) and then r can be constructed by doubling that vector($R = \begin{bmatrix} \vec{r}_1 & \vec{r}_1 \end{bmatrix}$).

Pseudocode

```
Matrix(2,6) GetInputAsMatrix(const string &input_path)

Int main():
        CALL SolveFile() for each file

        RETURN 0

void SolveFile(const string &input_path, const string &output_path):
        DECLARE 2x6 matrix raw_input as CALL of GetInputAsMatrix with input_path
        OPEN output_file at output_path

        CREATE mat from first 2 columns of raw_input

        CREATE eigenvalue_mat
        CREATE eigenvector_mat

        IF !SolveEigenvalueMat(mat, eigenvalue_mat):
                PRINT “no eigenvalues” to output_file
        ELSE IF !SolveEigenvectorMat(mat, eigenvalue_mat, eigenvector_mat):
                PRINT “cannot compute” to output_file
        ELSE:
                PRINT eigenvector_mat to output_file
                PRINT eigenvalue_mat to output_file
                PRINT eigenvector_mat * eigenvalue_mat * eigenvector_mat.inverse() to
                                output_file

                IF EqualsWithinTolerance(mat, eigenvector_mat * eigenvalue_mat *
                                eigenvector_mat.inverse():
```

```
            PRINT 1 to output_file
        ELSE:
            PRINT 0 to output_file

    CLOSE output_file

bool SolveEigenvalueMat(const matrix &mat, matrix &eigenvalue_mat):
    CALCULATE a, b and c

    CALCULATE quadratic_det as b*b - 4*a*c

    IF EqualsWithinTolerance(0, quadratic_det):
        DECLARE double eigenvalue as (-b) / (2 * a)

        IF EqualsWithinTolerance(0, eigenvalue):
            RETURN false
        ELSE
            SET eigenvalue_mat = 
```
$$\begin{bmatrix} eigenvalue & 0 \\ 0 & eigenvalue \end{bmatrix}$$
```
            RETURN true
    ELSE IF quadratic_det < 0:
        RETURN false
    ELSE:
        DECLARE double eigenvalue_1 as (-b + sqrt(quadratic_det)) / (2 * a)
        DECLARE double eigenvalue_2 as (-b - sqrt(quadratic_det)) / (2 * a)

        IF abs(eigenvalue_1) > abs(eigenvalue_2):
            SET eigenvalue_mat = 
```
$$\begin{bmatrix} eigenvalue\_1 & 0 \\ 0 & eigenvalue\_2 \end{bmatrix}$$
```
        ELSE:
            SET eigenvalue_mat = 
```
$$\begin{bmatrix} eigenvalue\_2 & 0 \\ 0 & eigenvalue\_1 \end{bmatrix}$$
```
        RETURN true

bool SolveEigenvectorMat(const matrix &mat, const matrix &eigenvalue_mat, matrix
        &eigenvector_mat):
    DECLARE eigenvector_1, eigenvector_2

    IF !SolveEigenvector(mat, eigenvalue_mat(0,0), eigenvector_1):
        RETURN false
    IF !SolveEigenvector(mat, eigenvalue_mat(1,1), eigenvector_2):
        RETURN false
```

```
    SET eigenvector_mat to [eigenvector_1, eigenvector_2]
    RETURN true

bool SolveEigenvector(const matrix &mat, const double &eigenvalue, vector
        &eigenvector):
    RETURN SolveHomogeneousSystem(mat - eigenvalue * 2dIdentityMatrix, eigenvector)
```