ID2090 Assignment 4 Report

Arivoli Ramamoorthy (ee23b008)

April 2024

Abstract

This report documents Assignment 4 for ID2090.

1 Breaking The Spectre

1.1 Task

Your task is to create a script, which takes as an argument, a YAML serialized file, containing a Matrix object inside it (consider this matrix as $\boldsymbol{A}$) and gives as output the following values:

- The eigenvalues of the matrix $\boldsymbol{A}$.
- The unitary matrix $\boldsymbol{U}$ that diagonalizes $\boldsymbol{A}$ (that is, $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^*$, where $\boldsymbol{D}$ is a diagonal matrix).
- The diagonal matrix $\boldsymbol{D}$ resulting from the previous step.
- The spectral decomposition of the matrix $\boldsymbol{A}$.
- The classification of the matrix $\boldsymbol{A}$ into any one of these special matrices (Hermitian, unitary, positive semidefinite, positive definite or just normal).

1.2 Solution Overview

I have initiated the analysis by loading the matrix data from the **YAML** file. Subsequently, I have converted the loaded data into a **SymPy** matrix or a **DomainMatrix**. Following this initial step, I proceeded to perform the essential computations to derive various properties of the matrix. These computations involved calculating the eigenvalues and eigenvectors, forming the unitary matrix, obtaining spectral decomposition, and finally, classifying the matrix based on specific criteria.

1.3 Loading YAML Data

- The `load_yaml_data` function loads data from a YAML file specified by the file path.
- It returns the loaded YAML data.

```python
def load_yaml_data(file_path):
    with open(file_path, "r") as file:
        yaml_data = yaml.load(file, Loader=yaml.Loader)
    return yaml_data
```

1.4 Matrix Conversion

- The `sympyconv` function converts the loaded YAML data to a SymPy matrix.
- The `domainconv` function converts the loaded YAML data to a DomainMatrix.

```python
def sympyconv(input):
    if ("a" in input
        and "b" in input["a"]
        and "c" in input["a"]["b"]
        and "d" in input["a"]["b"]["c"]
    ):
        rows, columns = input["rows"], input["columns"]
        matrix_data = input["a"]["b"]["c"]["d"]
        matrix_list = [
            [
                complex(matrix_data[i][j])
                if isinstance(matrix_data[i][j], Number)
                else matrix_data[i][j]
                for j in range(columns)
            ]
            for i in range(rows)
        ]
        return MutableDenseMatrix(matrix_list)
    else:
        return None


def domainconv(input):
    if "args" in input and input["args"]:
        args = input["args"][0]
        rows, cols = input["prev"]["state"]["dimensions"]
        matrix = DomainMatrix(args, rows=rows, dimensions=(rows,
    cols))
        return matrix
    else:
        return None
```

1.5 Matrix Operations

Spectral decomposition is a fundamental concept in linear algebra that provides a powerful representation of square matrices. Consider an $n \times n$ matrix A with distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and corresponding linearly independent eigenvectors $v_1, v_2, \ldots, v_n$. The spectral decomposition of A expresses it as the product of three matrices:

- The matrix Q, composed of the eigenvectors of A, forms the columns of Q.
- The diagonal matrix Λ, containing the eigenvalues of A, sits at the heart of the decomposition.
- The inverse of Q, denoted as Q^{-1}, facilitates the reconstruction of A from its spectral components.

Mathematically, this decomposition is expressed as:

$$A = Q\Lambda Q^{-1}$$

Here, Q is orthogonal ($Q^TQ = I$), ensuring that its inverse is simply its transpose ($Q^{-1} = Q^T$). This decomposition unveils essential structural insights into A and is widely utilized in diverse fields, including quantum mechanics, signal processing, and data analysis.

- The `decompose_matrix` function decomposes the matrix into eigenvalues, eigenvectors, and diagonal matrix.
- It then prints the eigenvalues, eigenvectors, and diagonal matrix as the output as required by the assignment.

```

def decompose_matrix(matrix):
    eigen = matrix.eigenvects(simplify=True)
    eigenvecs = []
    eigenvalslist = []
    for eigenvalue, mult, vecs in eigen:
        eigenvecs.extend(vecs[:mult])
        eigenvalslist.extend([eigenvalue] * mult)

    eigenvals = Matrix([eigenvalslist])
    print("Eigenvalues:")
    pprint(eigenvals)
    print("\n")

    eigenvecsmatrix = eigenvecs[0]
    for x in eigenvecs[1:]:
        eigenvecsmatrix = eigenvecsmatrix.row_join(x)

    Uc = Matrix.orthogonalize(*eigenvecs, normalize=True)
    U = Uc[0]
    for i in Uc[1:]:
        U = U.row_join(i)
    print("U:")
    pprint(U)
    print("\n")

    Ut = U.conjugate().transpose()
    D = Ut * matrix * U
    print("D:")
    pprint(D)
    print("\n")

    out = {}
    for i in range(3):
        if eigenvals[i] not in out:
            out[eigenvals[i]] = Uc[i] * Uc[i].conjugate().transpose
    ()
        else:
            out[eigenvals[i]] += Uc[i] * Uc[i].conjugate().
    transpose()

    print("Decomposition:")
    for key in out:
        print(key, "*")
        pprint(out[key])
```

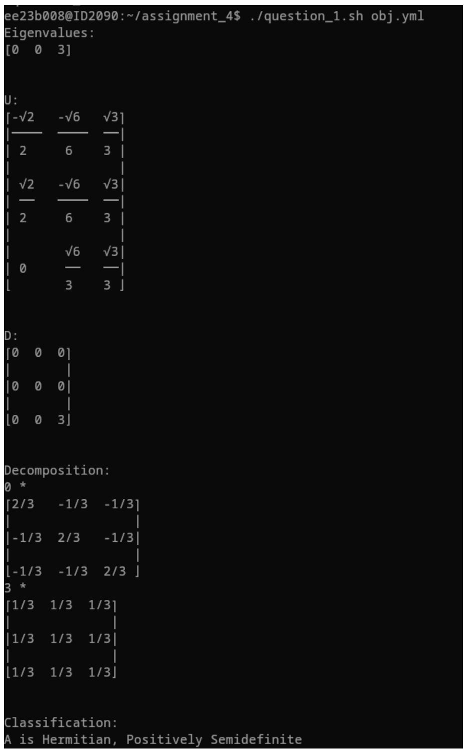
1.6 Execution

- The script first loads the YAML data from the file `$a`(command line argument).
- It uses, yaml.load (safe_load didn't seem to work in my code)
- It then converts the YAML data to a SymPy matrix or DomainMatrix.
- Finally, it performs matrix operations and classification on the loaded matrix.

```
yaml_data = load_yaml_data($a)
if isinstance(yaml_data, MutableDenseMatrix):
    matrix = yaml_data
elif isinstance(yaml_data, dict):
    matrix = sympyconv(yaml_data)
    if matrix is None:
        matrix = domainconv(yaml_data)
else:
    matrix = None
if matrix is not None:
    decompose_matrix(matrix)
    classify_matrix(matrix)
```

1.7 Conclusion and Output

The spectral decomposition stands as a cornerstone in linear algebra, providing a powerful method to express matrices in terms of their eigenvalues and



```
ee23b008@ID2090:~/assignment_4$ ./question_1.sh obj.yml
Eigenvalues:
[0  0  3]


U:
⎡-√2   -√6   √3⎤
⎢────  ────  ──⎥
⎢ 2     6    3 ⎥
⎢              ⎥
⎢ √2   -√6   √3⎥
⎢ ──   ────  ──⎥
⎢ 2     6    3 ⎥
⎢              ⎥
⎢       √6   √3⎥
⎢ 0     ──   ──⎥
⎣       3    3 ⎦


D:
⎡0  0  0⎤
⎢       ⎥
⎢0  0  0⎥
⎢       ⎥
⎣0  0  3⎦


Decomposition:
0 *
⎡2/3   -1/3  -1/3⎤
⎢                ⎥
⎢-1/3  2/3   -1/3⎥
⎢                ⎥
⎣-1/3  -1/3  2/3 ⎦
3 *
⎡1/3  1/3  1/3⎤
⎢             ⎥
⎢1/3  1/3  1/3⎥
⎢             ⎥
⎣1/3  1/3  1/3⎦


Classification:
A is Hermitian, Positively Semidefinite
```

Figure 1: Output

eigenvectors. Its significance reverberates across various domains, encompassing matrix diagonalization, dynamical systems' stability analysis, and pivotal dimensionality reduction techniques like Principal Component Analysis in data science.

Moreover, spectral decomposition serves as a fundamental tool for computing matrix functions and offers profound insights into the behavior of linear transformations. Its applicability extends to the analysis of Hermitian and unitary matrices, playing a crucial role in quantum mechanics and signal processing applications.

The concepts of eigenvectors and related mathematics form the bedrock of linear algebra, rendering it an indispensable tool not only in engineering disciplines but also in the realm of machine learning. The versatility and utility of spectral decomposition underscore its enduring importance in various scientific and computational endeavors.

2 Optimaze

2.1 Task:

You are given a set of (x, y, z) coordinates of points in points.csv (Program files are available in */var/home/Jan24/assignments/assignment_4*). Your task is to find the optimal equation of the plane, fitting the behavior of the system of points, to the best possible extent. The equation of the plane is taken to be ax + by + cz = 1, for the rest of the discussion. Here in the following discussion,

1. A refers to matrix A, and b refers to a vector b,
2. The superscript t refers to the t-th time step,
3. θ_t refers to vector (a, b, c) and $\Delta\theta_t$ refers to vector $(\Delta a, \Delta b, \Delta c)$.

1. Initialise random values to θ_0. Report this value when the program starts executing.
2. Compute the objective function (L) which is to be minimised. You can use the sum of squares of distances of the plane from the points to be that objective (error) function.
3. Find $\Delta\theta_t$ using Newton's steepest descent method:

   $$\Delta\theta_t = H_t^{-1} g_t$$

   where H is the Hessian and g is the Gradient which are computed as:

   $$H_{ij} = \frac{\partial^2 L}{\partial\theta_i \partial\theta_j}, \quad g_i = \frac{\partial L}{\partial\theta_i}$$

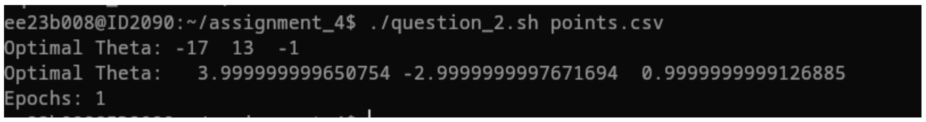
4. Update the parameters θ_t using the following update rule:

   $$\theta_{t+1} = \theta_t - \Delta\theta_t$$

5. Keep track of the error at the t-th step. This can be computed in a similar manner as the objective function.
6. Repeat the following steps until this error is less than a threshold, say δ. Then print the optimal parameters with the number of steps (epochs) taken for the solution to converge to θ^*.

2.2 Solution Overview

The provided problem involves implementing Newton's steepest descent method to minimize the objective function L by updating parameters iteratively until the error falls below a specified threshold. The key steps include initializing random values to θ_0, computing the objective function, finding $\Delta\theta_t$ using Newton's method, updating the parameters θ_t, tracking the error at each step, and repeating the process until convergence.



```
ee23b008@ID2090:~/assignment_4$ ./question_2.sh points.csv
Optimal Theta: -17  13  -1
Optimal Theta:   3.999999999650754 -2.9999999997671694  0.9999999999126885
Epochs: 1
```

Figure 2: Output

2.3 Initializing Random Values to θ_0:

- Random values are generated for θ_0 using the `randrange()` function within a specified range.
- The initial value of θ_0 is printed when the program starts executing.

```
#Initialize theta with random values
inttheta = matrix([[randrange(-23, 23)], [randrange(-23, 23)], [
    randrange(-23, 23)]])
```

2.4 Computing the Objective Function L and Output img:

- The objective function L is computed as the sum of squares of distances of the plane from the given points.
- The distance of a point from the plane is calculated using the formula $(x \cdot \theta_1 + y \cdot \theta_2 + z \cdot \theta_3 - 1)^2$, where x, y, and z are the coordinates of the point and θ_1, θ_2, and θ_3 are the parameters.

```
#distance of point from plane
def distance(x, y, z, points):
    d2 = [(x * point[0] + y * point[1] + z * point[2] - 1) ** 2 for
     point in points]
    return sum(d2)
```

2.5 Finding $\Delta\theta_t$ Using Newton's Method:

- The Hessian matrix H and the Gradient matrix G are computed.
- $\Delta\theta_t$ is calculated as $H_t^{-1}G_t$, where H is the Hessian matrix and G is the Gradient matrix.
- Newton's steepest descent method is used to find the optimal step size for parameter update.
- The parameters θ_t are updated using the formula $\theta_{t+1} = \theta_t - \Delta\theta_t$.

```
# Calculate delta theta and update theta
dtheta = hmatrix.inverse() * gmatrix
theta -= dtheta
iter += 1
```

2.6 Repeating Steps Until Convergence:

- The entire process is repeated until the error is less than the threshold δ.
- The optimal parameters θ^* along with the number of epochs taken for convergence are printed as outputs.

```
while distance(theta[0, 0], theta[1, 0], theta[2, 0], p) > $sigma:
```

3 References

1. https://web.stanford.edu/class/math114/lecture_notes/intro_opt.pdf
2. https://amsi.org.au/ESA_Senior_Years/SeniorTopic3/3j/3j_2content_2.html
3. https://docs.sympy.org/latest/modules/matrices/dense.html
4. Wikipedia
5. Credits to friends for helping :D