Lab 08 - Eigen Value and Corresponding Vector Using Power Method

Objectives

1. Calculate eigen value using power method.
2. Calculate corresponding vector using power method.

Theory

The power method is an iterative numerical technique used to estimate the dominant eigenvalue (the one with the largest absolute value) and its corresponding eigenvector of a square matrix. It is particularly useful for large matrices where analytical methods are impractical.

The method starts with an initial non-zero vector and repeatedly multiplies it by the matrix. After each multiplication, the resulting vector is normalized. As the process continues, the vector converges to the eigenvector associated with the dominant eigenvalue, and the ratio of the components of successive vectors provides an estimate of the eigenvalue.

Mathematically, for a matrix A and an initial vector x_0, the iteration is:

$$x_{k+1} = \frac{Ax_k}{\|Ax_k\|}$$

The approximate dominant eigenvalue at each step can be found using the Rayleigh quotient:

$$\lambda_k = \frac{x_k^T A x_k}{x_k^T x_k}$$

The power method is simple to implement and converges rapidly if the dominant eigenvalue is well separated from the others. However, it may fail or converge slowly if the matrix has multiple eigenvalues of the same magnitude or if the initial vector is orthogonal to the dominant eigenvector.

Algorithm

1. **Start** with a square matrix A and an initial non-zero vector x_0.
2. **Normalize** x_0 (e.g., divide by its largest absolute entry).
3. **Repeat** for a fixed number of iterations or until convergence:
    - Multiply: $y = Ax_k$
    - Estimate eigenvalue: $\lambda_k =$ largest absolute value entry in y
    - Normalize: $x_{k+1} = \frac{y}{\|y\|}$ (or divide by the entry used for λ_k)
    - Check for convergence (e.g., if x_{k+1} changes very little from x_k)
4. **Output** the estimated dominant eigenvalue λ_k and eigenvector x_{k+1}.

Source Code

```c
#include <stdio.h>
#include <math.h>

#define SIZE 10 // Size of the array
#define TOLERANCE 0.00001 // Tolerance for  comparison of successive iterations
#define MAX_ITERATIONS 1000 // Maximum number of iterations for convergence

// Macro to find the array size
#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))

// Function to find the maximum value in a vetor
double max_absolute_value(const double arr[SIZE], size_t size) {
    double max_val = fabs(arr[0]);
    for (size_t i = 1; i < size; i++) {
        if (fabs(arr[i]) > max_val) {
            max_val = fabs(arr[i]);
        }
    }
    return max_val;
}

// Function to user input a square matrix
void inputMatrix(double A[SIZE][SIZE], size_t size) {
    printf("Enter the elements of the matrix (%zu x %zu):\n", size, size);
    for (size_t i = 0; i < size; i++) {
        for (size_t j = 0; j < size; j++) {
            scanf("%lf", &A[i][j]);
        }
    }
}

// Function to user input an initial guess for the eigenvector
void inputInitialGuess(double X[SIZE], size_t size) {
    printf("Enter the initial guess for the eigenvector (size %zu):\n", size);
    for (size_t i = 0; i < size; i++) {
        scanf("%lf", &X[i]);
    }
}

// Function for matrix-vector multiplication
void matrixVectorMultiply(const double A[SIZE][SIZE], const double X[SIZE], double Y[SIZE], size_t size) {
    for (size_t i = 0; i < size; i++) {
        Y[i] = 0.0f;
        for (size_t j = 0; j < size; j++) {
            Y[i] += A[i][j] * X[j];
        }
    }
}

// Function to normalize a matrix
void normalize(double X[SIZE], size_t size) {
    // Find the maximum absolute value in the vector
    double max_val = max_absolute_value(X, size);
    if (max_val == 0.0) return; // Avoid division by zero
    for (size_t i = 0; i < size; i++) {
        X[i] /= max_val;
    }
}

// Power method to find the dominant eigenvalue and eigenvector
void powerMethod(const double A[SIZE][SIZE], double X[SIZE], size_t size  ) {
    double Y[SIZE];
    int iterations = 0;

    printf("***************************************************************************\n");
    printf("Power Method to find the dominant eigenvalue and eigenvector:\n");

    // Iterate until convergence or maximum iterations reached
    while (iterations < MAX_ITERATIONS) {
        // Multiply the matrix A with the vector X
        matrixVectorMultiply(A, X, Y, size);

        // Find the value of lambda (dominant eigenvalue)
        double lambda = max_absolute_value(Y, size);

        // Normalize the resulting vector Y
        normalize(Y, size);

        // Print the current iteration and the resulting vector in table format
        printf("Iteration %d: ", iterations + 1);
        for (size_t i = 0; i < size; i++) {
            printf("%lf ", Y[i]);
        }
        printf("\t: lambda = %lf", lambda);
        printf("\n");

        // Check for convergence
        double norm_diff = 0.0f;
        for (size_t i = 0; i < size; i++) {
            norm_diff += fabs(Y[i] - X[i]);
        }

        // If the change is within the tolerance, we have converged
        if (norm_diff < TOLERANCE) {
            break;
        }

        // Update X to be the new vector Y
        for (size_t i = 0; i < size; i++) {
            X[i] = Y[i];
        }

        iterations++;
    }

    printf("***************************************************************************\n");

    printf("Converged after %d iterations.\n", iterations);
    printf("Dominant eigenvector:\n");
    for (size_t i = 0; i < size; i++) {
        printf("%lf ", X[i]);
    }
    printf("\n");
}

// Driver code
int main(int argc, char const *argv[])
{
    double A[SIZE][SIZE], X[SIZE], Y[SIZE];
    int n;
    // Take user input for the size of the matrix
    printf("Enter the size of the matrix (max %d): ", SIZE);
    scanf("%d", &n);

    // Check if the user provided a valid size for the matrix
    if (n > SIZE || n <= 0) {
        printf("Invalid size. Please enter a value between 1 and %d.\n", SIZE);
        return 1;
    }

    // Input the matrix
    inputMatrix(A, n);

    // Initialize the first guess for the eigenvector
    inputInitialGuess(X, n);

    // Perform the power method to find the dominant eigenvalue and eigenvector
    powerMethod(A, X, n);

    return 0;
}
```

Output

```
Enter the size of the matrix (max 10): 2
Enter the elements of the matrix (2 x 2):
1 2
3 4
Enter the initial guess for the eigenvector (size 2):
1 1
***************************************************************************
Power Method to find the dominant eigenvalue and eigenvector:
Iteration 1: 0.428571 1.000000  : lambda = 7.000000
Iteration 2: 0.459459 1.000000  : lambda = 5.285714
Iteration 3: 0.457286 1.000000  : lambda = 5.378378
Iteration 4: 0.457437 1.000000  : lambda = 5.371859
Iteration 5: 0.457426 1.000000  : lambda = 5.372311
Iteration 6: 0.457427 1.000000  : lambda = 5.372279
***************************************************************************
Converged after 5 iterations.
Dominant eigenvector:
0.457426 1.000000
```

Discussion

The power method is a practical and efficient algorithm for finding the dominant eigenvalue and its corresponding eigenvector, especially for large matrices where analytical solutions are not feasible. In this lab, the method was implemented in C, allowing for user input of both the matrix and the initial guess vector. The iterative process demonstrated rapid convergence to the dominant eigenvalue, as seen in the sample output, where the method stabilized within a few iterations.

One important observation is that the choice of the initial vector can affect the speed of convergence, but as long as it is not orthogonal to the dominant eigenvector, the method will generally succeed. The normalization step ensures numerical stability and prevents overflow or underflow during iterations. However, the method may not work well if the matrix has multiple eigenvalues of the same magnitude or if the dominant eigenvalue is not well separated from the others.

Overall, the lab provided hands-on experience with an essential numerical technique, reinforcing concepts related to eigenvalues, eigenvectors, and iterative algorithms.

Conclusion

The power method offers a straightforward approach to approximating the dominant eigenvalue and eigenvector of a matrix. Through this lab, its implementation and practical utility were demonstrated, highlighting its effectiveness and limitations. This experience deepened understanding of iterative numerical methods in linear algebra.

Name: Kushal Prasad Joshi
Roll No: 230345
Faculty: Science and Technology
Program: BE Computer
Group: B