Welcome to Python Programming!

Teaching Assistant of Introduction to Numerical Analysis Class

Class Date: 2024-2025

Instructor: Dr. Mina Zarei

Class Link: Telegram group.

Exercise class time: **Monday at 9:30 P002 Class** (determined by student vote)

Session 1 (Install Python in VS Code)

Follow the instructions in the GitHub link.

link: www.github.com/AliSeif96/Introduction-to-Numerical-Analysis/).

---

session 2 (Basic training)

1. Variables and Data Types
2. Taking Output and Input from the User
3. Conditionals (if, elif, else)
4. Loops
5. Functions

---

session 3 (Basic training 2)

6. Lists
7. Arrays
8. Matrices
9. Using `NumPy`

---

session 4 (Root finding)

10.1. Bisection Method (روش تصنیف)

10.2. False Position Method (روش نابجایی)

10.3. Newton-Raphson Method (روش نیوتن رافسون)

10.4. Secant Method (روش وتری)

10.5. Fixed-Point Iteration (روش تکرار ساده)

10.6. Brent's Method

---

session 5 (Solving and displaying the Kuramoto model)

11. Matplotlib
12. Kuramoto model

---

session 6 (File Handling and Runge-Kutta 4th)

13. File Handling
14. Runge-Kutta 4th

Session 7 (Eigenvalues and Eigenvectors)

15. Computing Eigenvalues and Eigenvectors
16. Gaussian Elimination Method

Eigenvalue and Eigenvector Calculation for a 2x2 Matrix

This Python code computes the **eigenvalues** and **eigenvectors** of a 2x2 matrix. The code follows a basic procedure for finding eigenvalues by solving the characteristic equation and computes the corresponding eigenvectors using a direct method.

Steps Involved:

1. **Compute Eigenvalues**: The eigenvalues of the matrix (A) are found by solving the characteristic equation:

$$\det(A - \lambda I) = 0$$

   This is equivalent to solving the quadratic equation:

$$\lambda^2 - \mathrm{trace}(A)\lambda + \det(A) = 0$$

   where:

   - **(trace(A))** is the sum of the eigenvalues of matrix (A).
   - **(det(A))** is the product of the eigenvalues of matrix (A).

   The roots of this equation give the eigenvalues.

2. **Compute Eigenvectors**: Once the eigenvalues are found, the corresponding eigenvectors are calculated by solving the system

$$((A - \lambda I)x = 0)$$

   .

   The eigenvector is computed by manipulating the matrix to reduce it to a form that allows direct calculation of one element, with the other element set to 1. The result is then normalized to ensure it has a unit length.

Code Functions:

- `compute_eigenvalues(A)` : Computes the eigenvalues of matrix (A) by solving the characteristic equation.
- `compute_eigenvector(A, lambda)` : Computes an eigenvector corresponding to the eigenvalue (lambda).
- `main()` : The main function where matrix (A) is defined, eigenvalues and eigenvectors are computed, and results are displayed.

Example:

Given the matrix:

$$A = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix}$$

The code will compute the eigenvalues and eigenvectors for this matrix.

Usage:

To use the code, simply define the matrix (A) in the `main()` function, and call the `compute_eigenvalues` and `compute_eigenvector` functions to get the eigenvalues and eigenvectors, respectively.

Output:

The code will print the eigenvectors corresponding to each eigenvalue.

```python
import numpy as np

def compute_eigenvalues(A):
    """
    Computes the eigenvalues of a 2x2 matrix A by solving the characteristic equation.
    """
    a, b = A[0, 0], A[0, 1]  # استخراج عناصر سطر اول ماتریس A
    c, d = A[1, 0], A[1, 1]  # استخراج عناصر سطر دوم ماتریس A

    # Compute trace and determinant
    trace = a + d  # محاسبه trace که مجموع قطر اصلی ماتریس است
    determinant = (a * d) - (b * c)  # محاسبه دترمینان ماتریس A

    # Solve the quadratic equation λ^2 - trace*λ + determinant = 0
    discriminant = trace**2 - 4 * determinant  # محاسبه دیسکریمانت معادله درجه دوم

    if discriminant < 0:
        raise ValueError("Complex eigenvalues detected, this code only handles real eigenvalues.")
        # در صورتی که دیسکریمانت منفی باشد، نشان‌دهنده وجود مقادیر تخمینی مختلط است

    λ1 = (trace + discriminant**0.5) / 2  # محاسبه اولین مقدار ویژه
    λ2 = (trace - discriminant**0.5) / 2  # محاسبه دومین مقدار ویژه

    return [λ1, λ2]  # بازگشت مقادیر ویژه

def compute_eigenvector(A, λ):
    """
    Computes an eigenvector corresponding to the given eigenvalue λ.
    """
    A_minus_lambdaI = A - λ * np.eye(A.shape[0])  # A-λI, ساخت ماتریس A منهای λI
    # Solve (A - λI)x = 0
    if A_minus_lambdaI[0, 0] != 0:  # اگر عنصر قطر اصلی غیر صفر باشد
        x1 = -A_minus_lambdaI[0, 1] / A_minus_lambdaI[0, 0]  # محاسبه اولین عنصر بردار ویژه
        eigvec = np.array([x1, 1])  # ایجاد بردار ویژه
    else:  # اگر عنصر قطر اصلی صفر باشد
        x2 = -A_minus_lambdaI[1, 0] / A_minus_lambdaI[1, 1]  # محاسبه عنصر دیگر بردار ویژه
        eigvec = np.array([1, x2])  # ایجاد بردار ویژه
    # Normalize eigenvector
    return eigvec / np.linalg.norm(eigvec)  # نرمال‌سازی بردار ویژه

def main():
    """
    Main function to compute eigenvalues and eigenvectors of a given matrix.
    """
    # Define matrix A
    A = np.array([[4, 1],  # تعریف ماتریس A
                  [2, 3]])

    # Compute eigenvalues
    eigvals = compute_eigenvalues(A)  # محاسبه مقادیر ویژه

    # Compute eigenvectors for each eigenvalue
    eigvecs = [compute_eigenvector(A, λ) for λ in eigvals]  # محاسبه بردارهای ویژه برای هر مقدار ویژه

    # Display eigenvectors
    for i, vec in enumerate(eigvecs):  # نمایش بردارهای ویژه
        print(f"Eigenvector corresponding to λ={eigvals[i]}: {vec}")  # چاپ بردار ویژه مربوط به هر مقدار ویژه

if __name__ == "__main__":  # اگر این فایل به عنوان اسکریپت اصلی اجرا شود
    main()  # اجرای تابع اصلی
```

```
Eigenvector corresponding to λ=5.0: [0.70710678 0.70710678]
Eigenvector corresponding to λ=2.0: [-0.4472136   0.89442719]
```

Gaussian Elimination for Solving Linear Systems

This Python code implements the **Gaussian Elimination** method for solving a system of linear equations (Ax = b). The process consists of three main steps: pivoting, forward elimination, and back substitution. Each step is broken down into a separate function to make the code modular and understandable.

Steps Involved:

1. **Create Augmented Matrix**: Combine the coefficient matrix (A) and the right-hand side vector (b) to form an augmented matrix ([A | b]).
2. **Pivoting**: Swap rows to ensure that the element with the largest absolute value in the current column becomes the pivot element.
3. **Forward Elimination**: Perform Gaussian elimination to transform the augmented matrix into an upper triangular form.
4. **Back Substitution**: Once the matrix is in upper triangular form, solve for the unknowns by substituting values back into the equations.

Code Functions:

- `create_augmented_matrix(A, b)` : Combines matrix (A) and vector (b) to create the augmented matrix.
- `pivoting(augmented_matrix, i)` : Performs pivoting by swapping rows based on the largest absolute value in the current column.
- `forward_elimination(augmented_matrix, x)` : Eliminates entries below the pivot to create an upper triangular matrix.
- `back_substitution(augmented_matrix)` : Performs back substitution to find the solution vector (x).
- `gaussian_elimination(A, b)` : The main function that ties everything together and solves the system using Gaussian elimination.

Example:

Given the system of equations:

$$\begin{aligned} 2x_1 + x_2 - x_3 &= 8 \\ -3x_1 - x_2 + 2x_3 &= -11 \\ -2x_1 + x_2 + 2x_3 &= -3 \end{aligned}$$

The code will compute the solution for (x_1), (x_2), and (x_3) using Gaussian elimination.

Usage:

To use the code, define the coefficient matrix (A) and the right-hand side vector (b), then call the `gaussian_elimination` function with these inputs. The solution will be returned as a vector (x).

Output:

The code will print the augmented matrix and the solution to the system.

```python
import numpy as np

def create_augmented_matrix(A, b):
    """
    Creates the augmented matrix [A | b] by combining the coefficient matrix A with the right-hand side vector b.
    """
    n = len(b)  # تعداد معادلات
    augmented_matrix = []  # ماتریس گسترش یافته
    for i in range(n):
        augmented_matrix.append(list(A[i]) + [b[i]])  # اضافه کردن b به A
    return augmented_matrix  # بازگشت ماتریس گسترش یافته


def pivoting(augmented_matrix, i):
    """
    Performs pivoting to swap rows based on the largest absolute value in the current column.
    """
    n = len(augmented_matrix)
    max_row = i  # فرض می‌کنیم بزرگترین مقدار در سطر i است
    for j in range(i+1, n):  # جستجو در سطرهای زیر
        if abs(augmented_matrix[j][i]) > abs(augmented_matrix[max_row][i]):  # مقایسه مقادیر
            max_row = j  # ذخیره سطر جدید با بزرگترین مقدار
    augmented_matrix[i], augmented_matrix[max_row] = augmented_matrix[max_row], augmented_matrix[i]  # جابجایی سطرها
    return augmented_matrix  # بازگشت ماتریس پس از پیووتینگ


def forward_elimination(augmented_matrix, x):
    """
    Performs forward elimination to create an upper triangular matrix.
    """
    n = len(augmented_matrix)
    divisor = augmented_matrix[x][x]  # مقدار قطر اصلی سطر معین x

    # یک کردن مقدار قطر اصلی سطر معین x
    for i in range(x, n+1):  # همه ی سطر معین نورمال شود
        augmented_matrix[x][i] /= divisor  # [سطر][ستون]

    # حذف مقادیر پایین‌تر از قطری
    for j in range(x+1, n):  # برای هر سطر زیر سطر x
        factor = augmented_matrix[j][x]  # مقدار ضرب‌شونده برای حذف
        for k in range(x, n+1):  # اصلاح تمام مقادیر سطر
            augmented_matrix[j][k] -= factor * augmented_matrix[x][k]
    return augmented_matrix  # بازگشت ماتریس پس از حذف گاوسی


def back_substitution(augmented_matrix):
    """
    Performs back substitution to find the solution vector x.
    """
    n = len(augmented_matrix)
    x = np.zeros(n)  # ایجاد یک بردار صفر برای جواب‌ها
    for i in range(n-1, -1, -1):  # از سطر آخر به سمت بالا حرکت می‌کنیم
        x[i] = augmented_matrix[i][n]  # جواب از آخرین عنصر سطر
        for j in range(i+1, n):  # برای هر متغیر بعدی
            x[i] -= augmented_matrix[i][j] * x[j]  # حل معادله برای متغیر i
    return x  # بازگشت جواب‌ها


def gaussian_elimination(A, b):
    """
    Solves the system of linear equations Ax = b using Gaussian Elimination.
    """
    augmented_matrix = create_augmented_matrix(A, b)  # ساخت ماتریس گسترش یافته

    # انجام حذف گاوسی
    n = len(b)
    for i in range(n):  # برای هر ستون
        augmented_matrix = pivoting(augmented_matrix, i)  # انجام پیووتینگ
        augmented_matrix = forward_elimination(augmented_matrix, i)  # انجام حذف گاوسی
    # انجام بازگشتی
    x = back_substitution(augmented_matrix)  # یافتن جواب‌ها
    return x  # بازگشت جواب‌ها


def main():
    # سیستم معادلات نمونه:
    A = np.array([[2, 1, -1],
                  [-3, -1, 2],
                  [-2, 1, 2]], dtype=float)  # ماتریس ضرایب A
    b = np.array([8, -11, -3], dtype=float)  # بردار سمت راست معادلات


    # حل سیستم معادلات
    solution = gaussian_elimination(A, b)  # فراخوانی تابع حذف گاوسی

    print("[A|b]:\n",np.hstack([A, b.reshape(-1, 1)]))
    print("\nSolution:\n", solution)  # نمایش جواب‌ها

if __name__ == "__main__":  # اگر این فایل به عنوان اسکریپت اصلی اجرا شود
    main()  # اجرای تابع main
```

```
[A|b]:
 [[  2.   1.  -1.   8.]
 [ -3.  -1.   2. -11.]
 [ -2.   1.   2.  -3.]]

Solution:
 [ 2.  3. -1.]
```

تمرین