

C语言大作业报告：矩阵与向量操作库

C语言大作业报告：矩阵与向量操作库

一、需求分析

二、函数设计

0. 导入导出

0.1 从txt导入

0.2 导出到txt

1. 单个向量操作

1.1 显示与切片

1.1.1 显示向量

1.1.2 显示向量元素

1.1.3 向量切片

1.2 初始化与变换

1.2.1 初始化列向量

1.2.2 向量绝对值

1.2.3 向量元素取倒数

1.2.4 向量元素平方

1.2.5 向量归一化

1.3 统计与排序

1.3.1 查找向量中的最大元素

1.3.2 查找向量中的最小元素

1.3.3 查找向量中最大的c1个元素的索引

1.3.4 查找向量中最小的c1个元素的索引

1.3.5 从大到小排序向量

1.3.6 从小到大排序向量

1.3.7 向量范数

1.4 算术操作

1.4.1 向量除以标量

1.4.2 向量乘以标量

2. 单个矩阵操作

2.1 显示与切片

2.1.1 显示矩阵

2.1.2 显示矩阵元素

2.1.3 矩阵行切片

2.1.4 矩阵列切片

2.1.5 矩阵根据位置矩阵切片

2.2 变换

2.2.1 矩阵元素取倒数

2.2.2 矩阵元素取平方

2.2.3 替换矩阵的指定列

2.2.4 矩阵取转置

2.3 统计

2.3.1 矩阵所有元素求和

2.3.2 查找矩阵中的最大元素

2.3.3 每行的最大元素

2.3.4 每行的最小元素

2.3.5 整行元素都为0的行编号

2.3.6 矩阵范数

2.3.7 矩阵列向量间的夹角Cos值

2.4 算术操作

2.4.1 矩阵乘以标量

2.5 特殊矩阵

2.5.1 生成随机对称矩阵

2.6 特征值求解

2.6.1 幂法

3. 向量与向量操作

3.1 向量加法

3.2 向量减法

3.3 向量按元素除法

3.4 向量按元素乘法

3.5 余弦

3.6 点积

3.7 向量外积

4. 向量与矩阵操作

4.1 向量左乘矩阵

4.2 向量右乘矩阵

5. 矩阵与矩阵操作

5.1 矩阵加法

5.2 矩阵减法

5.3 AB型矩阵相乘

5.4 ATBA型矩阵相乘

5.5 矩阵按元素乘法

5.6 矩阵按元素除法

6. 线性方程

6.1 上三角矩阵的逆

6.2 上三角矩阵方程求解

6.3 一般线性方程求解（高斯消去法）

7. 辅助函数

7.1 向量

7.2 矩阵

7.3 方程

7.4 其他

8. 主函数及子菜单

8.1 主函数

8.2 子菜单

三、界面设计

1. 主菜单

2. 子菜单

四、测试和排错

1. 测试

1.1 主菜单测试

1.2 单个向量操作

1.3 单个矩阵操作

1.4 向量与向量操作

1.5 向量与矩阵操作

1.6 矩阵与矩阵操作

1.7 线性方程

2. 排错

五、总结与体会

1. 得意之处

2. 不足

3. 总结

4. 表格

六、附录

函数全部代码

0. 导入导出

1. 单个向量操作

2. 单个矩阵操作

3. 向量与向量操作

4. 向量与矩阵操作

5. 矩阵与矩阵操作

一、需求分析

• 功能需求

1. 界面操作：
 - 用户可以通过选择文件来导入矩阵和向量数据。
 - 界面提供运算选项，如矩阵乘法、向量运算等。
 - 界面显示运算结果。
2. 数据导入导出：
 - 支持从txt文件中导入矩阵和向量数据。
 - 支持将运算结果导出为txt文件。
3. 基本运算：
 - 支持求矩阵或向量的特定元素、切片操作。
 - 支持向量的基本运算，如点乘、点除、归一化等。
 - 支持矩阵的基本运算，如加法、减法、乘法（包括矩阵与矩阵、矩阵与向量）。
4. 其他运算：
 - 支持线性方程组的求解（如高斯消元法）。
 - 支持矩阵特征值的求解（如幂法）。

• 非功能需求

- 易用性：界面简洁明了，操作直观。
- 准确性：运算结果应准确无误。
- 可靠性：软件应能正确处理错误输入和异常情况。

二、函数设计

规定索引从0开始！

设计思路：

- 数据结构设计：

```
typedef struct {  
    double* data; // 数据  
    int size; // 大小  
    int is_init; // 是否初始化  
} Vector;  
typedef struct {  
    double* data; // 数据  
    int rows; // 行数  
    int cols; // 列数  
    int is_init; // 是否初始化  
} Matrix;
```

向量和矩阵都使用了一维数组来存数据。

- 具体函数设计:

按功能需求分为功能函数与辅助函数:

- 功能函数:

按涉及向量/矩阵数量分类, 将功能函数大致分为6类

- 0. 导入导出

- 1. 单个向量

- 2. 单个矩阵

- 3. 向量与向量

- 4. 向量与矩阵

- 5. 矩阵与矩阵

- 6. 线性方程

并按这6类功能设计6个子菜单(1-6), 这样方便用户操作,同时也方便内存管理。

- 非功能函数:

- 1. 子菜单

- 2. 其他辅助函数

子菜单会展示具体选项并强制导入, 然后用switch-case选择不同功能; 其他辅助函数包括算向量内积长度或者冒泡排序之类的辅助函数。

下面是具体的函数设计:

(注_1: 这里只给出了函数声明、设计思路及各函数的调用关系, [全部代码](#)放在了附录里, 单击链接跳转至对应位置。(群里说不需要源代码, 但是附录里已经分类并且加格式写好了, 就不删了))

(注_2: 导入导出函数及其初始化函数每个子菜单每个选项都会调用, 下面不再赘述。)

(注_3: 有些函数的参数功能和返回值不言自明, 下面不再赘述。)

0. 导入导出

0.1 从txt导入

- `Vector* import_vector_from_txt(char* filename);`

规定向量的格式为一个文件只有一个向量, 每行只有一个元素, 还有其他则忽略。

以“r”只读打开文件, 从头统计向量vec大小size, 用 `fgets()` 读取每行数据到line, 去掉line的行尾换行符, 跳过'\0', 遇到有效数据 `size++`; 初始化vec, 回开头填充vec, 用 `stdrod()` 转换字符为double。

- `Matrix* import_matrix_from_txt(char* filename);`

规定矩阵格式为一个文件只有一个矩阵, 每行元素数相同, 以第一行元素数为列数, 其他行元素数大于列数则忽略,其他行元素数小于列数则补为0。

以“r”只读打开文件, 从头统计矩阵mat列数cols, 用 `fgets()` 读取每行数据到line, 用 `strtok()` 以空格、制表符、换行符分割数据, 遇到有效列数 `cols++`; 再回开头读行数rows, 初始化mat, 回开头填充mat, 用 `stdrod()` 转换字符为double。

0.2 导出到txt

用"w"创建文件，用fprintf()写入。

- `void export_vector_to_txt(Vector* vector, char* filename);`
- `void export_matrix_to_txt(Matrix* matrix, char* filename);`

1. 单个向量操作

对于单个向量和单个矩阵的操作，尽量使用了void型函数，便于内存分配和释放。

1.1 显示与切片

1.1.1 显示向量

参数功能：指向要操作对象的指针

- `void print_vector(Vector* vector);`

1.1.2 显示向量元素

参数功能：指向要操作对象的指针,向量元素的对应索引

- `void print_vector_element(Vector* vector, int index);`

1.1.3 向量切片

参数功能：指向要操作对象的指针,切片索引和切片长度；返回值：已切片的向量

要先给切片分内存 `Vector* slice_vector = (Vector*)malloc(sizeof(Vector));`

- `Vector* slice_vector(Vector* vector, int* indices, int indices_length);`

1.2 初始化与变换

1.2.1 初始化列向量

调用init_vector函数（辅助函数），参数功能：大小和指定初始化的值；返回值：指定初始化的向量

- `Vector init_column_vector(int size, double value);`

1.2.2 向量绝对值

参数功能：指向要操作对象的指针，下同

- `void abs_vector(Vector* vector);`

1.2.3 向量元素取倒数

- `void reciprocal_elements_vector(Vector* vector);`

1.2.4 向量元素平方

- `void square_elements_vector(Vector* vector);`

1.2.5 向量归一化

用2范数，即模长规范化,调用 `double norm_vector(Vector* vector, int option);`

- `void normalize_vector(Vector* vector);`

1.3 统计与排序

1.3.1 查找向量中的最大元素

冒泡排序，找最值和排序都会调用冒泡排序函数，不再赘述

- `double max_element_vector(Vector vector);`

1.3.2 查找向量中的最小元素

- `double min_element_vector(Vector vector);`

1.3.3 查找向量中最大的c1个元素的索引

参数功能：要操作的对象，c1，指向c1个元素的索引的指针

新建一个两个元素的结构体

```
typedef struct{
    double vaule;
    int index;
}index_vaule;
```

用来保存元素对应的索引信息，然后对vaule排序，indices就可以从index中获得。

调用辅助函数 `void bubbleSort_struct_index_vaule(index_vaule* arr, int n)` 排序

调用辅助函数 `void export_indices_to_txt(int* indices, int c1, char* filename)` 导出

- `void top_c1_max_indices(Vector vector, int c1, int* indices);`

1.3.4 查找向量中最小的c1个元素的索引

- `void top_c1_min_indices(Vector vector, int c1, int* indices);`

1.3.5 从大到小排序向量

- `void sort_vector_desc(Vector* vector);`

1.3.6 从小到大排序向量

- `void sort_vector_asc(Vector* vector);`

1.3.7 向量范数

参数功能：指向要操作对象的指针，选项；返回值：范数

- `double norm_vector(Vector* vector, int option);`

1.4 算术操作

1.4.1 向量除以标量

参数功能：指向要操作对象的指针，标量

- `void divide_vector_by_scalar(Vector* vector, double scalar);`

1.4.2 向量乘以标量

- `void multiply_vector_by_scalar(Vector* vector, double scalar);`

2. 单个矩阵操作

2.1 显示与切片

2.1.1 显示矩阵

参数功能：指向要操作对象的指针，下同

- `void print_matrix(Matrix* mat);`

2.1.2 显示矩阵元素

- `void print_matrix_element(Matrix* matrix, int row, int col);`

2.1.3 矩阵行切片

- `double* slice_matrix_row(Matrix* matrix, int row);`

2.1.4 矩阵列切片

- `double* slice_matrix_col(Matrix* matrix, int col);`

2.1.5 矩阵根据位置矩阵切片

参数功能：指向要操作对象的指针，切片行数，切片列数，切片列索引，切片行索引

- `Matrix* slice_matrix_m(Matrix* matrix, int row, int col, int * indices_col, int * indices_row);`

2.2 变换

2.2.1 矩阵元素取倒数

- `void reciprocal_elements_matrix(Matrix* matrix);`

2.2.2 矩阵元素取平方

- `void square_elements_matrix(Matrix* matrix);`

2.2.3 替换矩阵的指定列

参数功能：指向要操作对象的指针，指向新列的元素的指针，要替换的列号

- `void replace_column(Matrix* matrix, double* new_column, int col_index);`

2.2.4 矩阵取转置

- `void transpose_matrix(Matrix* matrix);`

2.3 统计

2.3.1 矩阵所有元素求和

- `double sum_of_matrix(Matrix* matrix);`

2.3.2 查找矩阵中的最大元素

- `void max_elements_matrix(Matrix* matrix);`

2.3.3 每行的最大元素

- `void max_elements_per_row(Matrix* matrix, double* max_values);`

2.3.4 每行的最小元素

- `void min_elements_per_row(Matrix* matrix, double* min_values);`

2.3.5 整行元素都为0的行编号

- `void find_zero_rows(Matrix* matrix, int* num_zero_rows);`

2.3.6 矩阵范数

参数功能：指向要操作对象的指针，选项；返回值：范数

- `double norm_matrix(Matrix* matrix, int option);`

2.3.7 矩阵列向量间的夹角Cos值

参数功能：指向要操作对象的指针；返回值：所有的列向量余弦值指针

调用 `double* slice_matrix_col(Matrix* matrix, int col);` 取矩阵的列向量切片

调用 `double calculate_cos(double* vec1, double* vec2, int size);` 算余弦值

- `double* calculate_all_cos(Matrix* matrix);`

2.4 算术操作

2.4.1 矩阵乘以标量

参数功能：指向要操作对象的指针；返回值：结果

- `Matrix multiply_matrix_by_scalar(Matrix* matrix, double scalar);`

2.5 特殊矩阵

2.5.1 生成随机对称矩阵

参数功能：生成矩阵的大小；返回值：指向生成矩阵的指针

- `Matrix* generate_random_symmetric_matrix(int rows);`

2.6 特征值求解

2.6.1 幂法

数学原理：

假设：

1. A 有一个主特征值 λ_1 ，其模严格大于其他特征值的模。
2. 初始向量 x_0 在 x_1 方向上有非零分量（即 $c_1 \neq 0$ ）。

假设矩阵 A 是一个 $n \times n$ 的实矩阵，且存在一个主特征值 λ_1 ，其对应的特征向量为 x_1 。

此外，假设 $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \dots \geq |\lambda_n|$ ，其中 $\lambda_2, \lambda_3, \dots, \lambda_n$ 是 A 的其他特征值。

首先，可以将初始向量 x_0 表示为 A 的所有特征向量的线性组合（矩阵 A 有 n 个线性无关的特征向量并不是幂法收敛的必要条件）：

$$x_0 = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

其中 c_1, c_2, \dots, c_n 是标量系数。

现在，考虑第 k 次迭代后的向量 x_k ：

$$x_k = \frac{A^k x_0}{\|A^k x_0\|}$$

将 x_0 的展开式代入上式，得到：

$$x_k = \frac{A^k (c_1 x_1 + c_2 x_2 + \cdots + c_n x_n)}{\|A^k (c_1 x_1 + c_2 x_2 + \cdots + c_n x_n)\|}$$

由于 $A^k x_i = \lambda_i^k x_i$ （这是特征值和特征向量的定义），可以将上式重写为：

$$x_k = \frac{c_1 \lambda_1^k x_1 + c_2 \lambda_2^k x_2 + \cdots + c_n \lambda_n^k x_n}{\|c_1 \lambda_1^k x_1 + c_2 \lambda_2^k x_2 + \cdots + c_n \lambda_n^k x_n\|}$$

现在，考虑当 k 趋于无穷大时的情况。由于 $|\lambda_1| > |\lambda_i|$ 对于所有 $i > 1$ ，因此 λ_1^k 的增长速度将远远超过其他项 λ_i^k 。因此，当 k 很大时，上式中的分母主要由 $c_1 \lambda_1^k x_1$ 项决定。

同样地，分子中的 $c_1 \lambda_1^k x_1$ 项也将占据主导地位。因此，当 k 趋于无穷大时， x_k 将趋近于：

$$x_k \approx \frac{c_1 \lambda_1^k x_1}{\|c_1 \lambda_1^k x_1\|} = \pm x_1$$

这里的 \pm 取决于 c_1 的符号，因为特征向量可以乘以任意非零常数，所以可以说 x_k 收敛到了 x_1 的方向。

更具体地说，设 $x_{k+1} = \frac{Ax_k}{\|Ax_k\|}$ ，其中 x_k 是第 k 次迭代后的向量。那么，随着 k 的增加， x_{k+1} 将越来越接近 x_1 的方向，即 x_{k+1} 将收敛到 x_1 ，有：

$$Ax_{k+1} \approx \lambda_1 x_{k+1}$$

则有：

$$\lambda_1 \approx \frac{x_{k+1}^T Ax_{k+1}}{x_{k+1}^T x_{k+1}}$$

程序实现：

1. 初始化：

- 选择一个非零的初始向量 v_0 。
- 设定一个容差 ε 作为收敛的阈值。
- 设定最大迭代次数 N 。

2. 迭代过程：

对于 $k = 0, 1, 2, \dots, N$ ，执行以下步骤： a. 计算 $w_k = Av_k$ 。 b. 计算 $m_k = \|w_k\|$ （向量的2-范数）。 c. 更新向量 $v_{k+1} = \frac{w_k}{m_k}$ 。 d. 如果 $\|v_{k+1} - v_k\| < \varepsilon$ ，则认为已经收敛，停止迭代。

3. 提取特征值和特征向量：

- 特征值近似为 $\lambda \approx m_k$ 。
- 特征向量为最后一次迭代的 v_{k+1} 。

参数功能:

指向要操作对象的指针,切片索引和切片长度

lambda: 指向主特征值的指针

x: 指向主特征向量的指针

epsilon: 收敛阈值

N: 最大迭代次数

返回值: 0: 未收敛; 1: 已收敛

- `int power_method(Matrix* A, double* lambda, double* x, double epsilon, int N)`

3. 向量与向量操作

检查向量大小是否匹配

3.1 向量加法

参数功能: 要操作的对象; 返回值: 操作后的向量; 下同

- `Vector add_vectors(Vector v1, Vector v2);`

3.2 向量减法

- `Vector subtract_vectors(Vector v1, Vector v2);`

3.3 向量按元素除法

检查除数不为零, 若有除数为0, 则将对结果记为0

- `Vector elementwise_divide(Vector v1, Vector v2);`

3.4 向量按元素乘法

`Vector elementwise_multiply(Vector v1, Vector v2);`

3.5 余弦

调用 `double calculate_cos(double* vec1, double* vec2, int size)`

- `double cosine_similarity(Vector v1, Vector v2);`

3.6 点积

参数功能: 要操作的对象; 返回值: 结果; 下同

- `double dot_product(Vector* v1, Vector* v2);`

3.7 向量外积

- `Matrix vector_outer_product(Vector* vec1, Vector* vec2);`

4. 向量与矩阵操作

检查向量和矩阵大小是否匹配

4.1 向量左乘矩阵

- `Vector multiply_matrix_by_vector(Matrix* matrix, Vector* vector);`

4.2 向量右乘矩阵

- `Vector multiply_matrix_by_vector(Matrix* matrix, Vector* vector);`

5. 矩阵与矩阵操作

检查矩阵大小是否匹配

5.1 矩阵加法

参数功能：要操作的对象指针；返回值：结果；下同

- `Matrix add_matrices(Matrix* m1, Matrix* m2);`

5.2 矩阵减法

- `Matrix subtract_matrices(Matrix* m1, Matrix* m2);`

5.3 AB型矩阵相乘

- `Matrix multiply_AB(Matrix* A, Matrix* B);`

5.4 ATBA型矩阵相乘

调用 `transpose_matrix(A)`；取转置

调用 `multiply_AB(B, A)`；算矩阵相乘

- `Matrix multiply_ATBA(Matrix* A, Matrix* B);`

5.5 矩阵按元素乘法

- `Matrix elementwise_multiply_matrices(Matrix* m1, Matrix* m2);`

5.6 矩阵按元素除法

检查除数不为零，若有除数为0，则对应结果记为0

- `Matrix elementwise_divide_matrices(Matrix* m1, Matrix* m2);`

6. 线性方程

6.1 上三角矩阵的逆

参数功能：指向要操作对象的指针；返回值：逆矩阵

`Matrix invert_upper_triangular_matrix(Matrix* m);`

1. 逆矩阵的对角线元素取倒数即可，下面是计算非对角线元素的思路：

在计算逆矩阵的非对角线元素时，我们基于以下方程：

$$\sum_{k=i}^n A_{ik}(A^{-1})_{kj} = I_{ij}$$

由于 I_{ij} 是单位矩阵的元素，当 $i \neq j$ 时， $I_{ij} = 0$ 。

同时，因为 A 和 A^{-1} 都是上三角矩阵，当 $k > j$ 时， $(A^{-1})_{kj} = 0$ 。因此，考虑 k 的范围从 i 到 j ，方程变为：

$$\sum_{k=i}^j A_{ik}(A^{-1})_{kj} = 0$$

将 $k = j$ 的情况单独分离出来：

$$A_{ij}(A^{-1})_{jj} + \sum_{k=i}^{j-1} A_{ik}(A^{-1})_{kj} = 0$$

解出 $(A^{-1})_{ij}$ ：

$$(A^{-1})_{ij} = -\frac{1}{A_{jj}} \sum_{k=i}^{j-1} A_{ik}(A^{-1})_{kj}$$

在代码中实现时，计算累加和：

$$\text{sum} = \sum_{k=i}^{j-1} A_{ik}(A^{-1})_{kj}$$

然后使用这个累加和来计算 $(A^{-1})_{ij}$ ：

$$(A^{-1})_{ij} = -\text{sum} \times (A^{-1})_{jj}$$

2. 下面是函数思路：

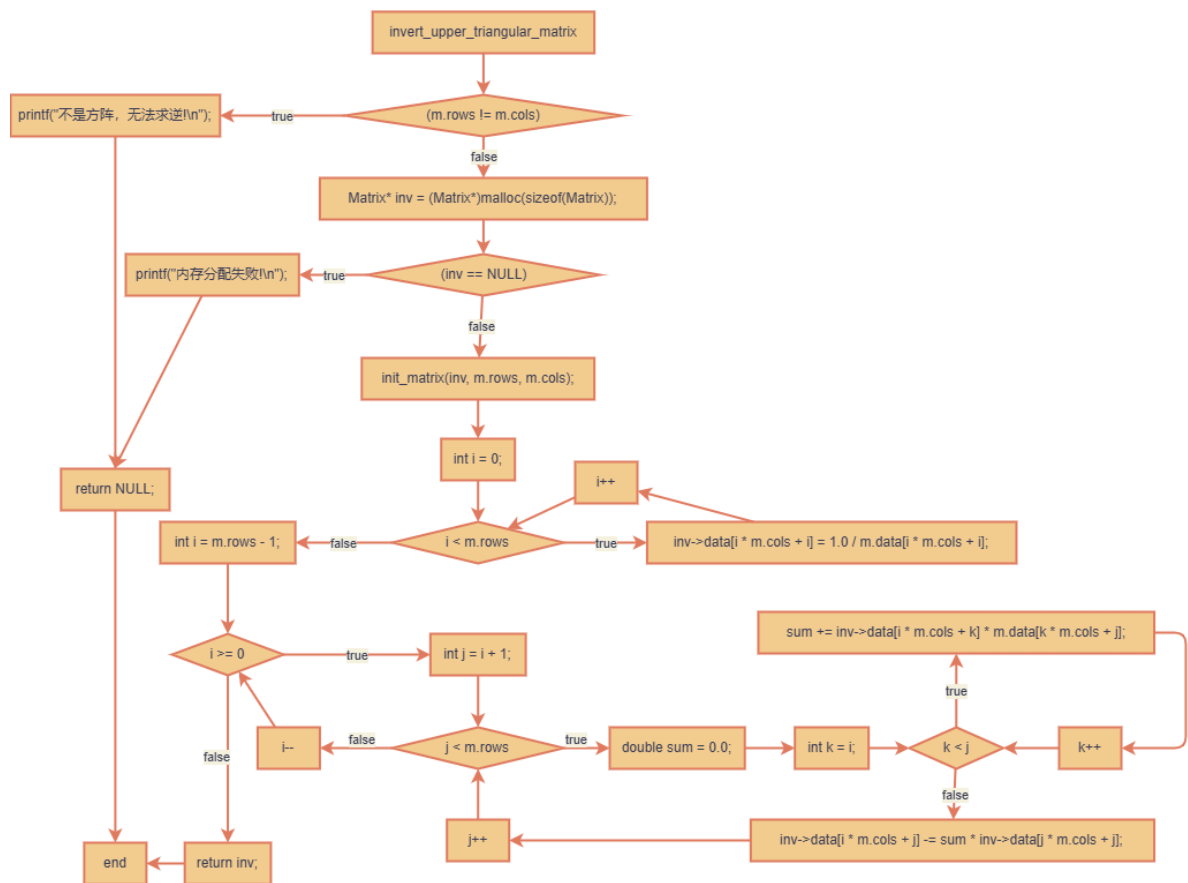
先将逆矩阵的对角线元素取为倒数 `inv[i][i] = 1 / matrix[i][i]`

然后计算逆矩阵的上三角部分的非对角线元素

从最后一行开始，逐行向上求

```
for i from n-1 to 0 do
  for j from i+1 to n-1
    sum = 0
    for k from i to j-1 do
      sum = sum + inv[i][k] * m[k][j]
    inv[i][j] = inv[i][j] - sum * inv[j][j]
```

下面是[流程图](#)：



6.2 上三角矩阵方程求解

参数功能:

指针:

A: 系数上三角矩阵

b: Ax=b的b

x: 解向量

下同

1. 从最后一个方程开始求解

$$x_n = \frac{b_n}{a'_{nn}}$$

2. 逐个回代求解 对于 $i = n - 1, n - 2, \dots, 1$:

$$x_i = \frac{b_i - (a_{i,i+1} \cdot x_{i+1} + a_{i,i+2} \cdot x_{i+2} + \dots + a_{i,n} \cdot x_n)}{a_{ii}}$$

```
void solve_upper_triangular_equation(Matrix* A, Vector* b, Vector* x);
```

6.3 一般线性方程求解 (高斯消去法)

通过行变换, 将线性方程组的系数矩阵转换为上三角矩阵。

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \xrightarrow{\text{高斯消去法}} \begin{pmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a'_{nn} \end{pmatrix}$$

下面是具体过程：

对于线性方程组 $Ax = b$ ，其中 A 是系数矩阵， x 是未知数向量， b 是常数项向量。

先选主元，选主元的过程是为了确保数值稳定性和避免除以零。

选主元：

假设当前正在处理第 k 列，从第 k 列的第 k 行到最后一行的元素中，选择绝对值最大的元素作为新的主元。

如果该元素不在第 k 行，则将其所在的行与第 k 行进行交换。

消元

消元过程是将系数矩阵 A 通过行变换转换为上三角矩阵。

1. 对于 $k = 1, 2, \dots, n - 1$ ，执行以下步骤：

2. 对于 $i = k + 1, k + 2, \dots, n$ ，进行消元：

◦ 计算消元乘数：

$$m_{ik} = \frac{a_{ik}}{a_{kk}}$$

◦ 使用该乘数更新第 i 行的元素，使第 i 行第 k 列的元素变为 0：

$$a_{ij}^{\text{new}} = a_{ij}^{\text{old}} - m_{ik} \cdot a_{kj}^{\text{old}}, \quad j = k, k + 1, \dots, n$$

◦ 同时更新增广向量 b 的对应元素：

$$b_i^{\text{new}} = b_i^{\text{old}} - m_{ik} \cdot b_k^{\text{old}}$$

经过上述步骤，原矩阵 A 被转换为一个上三角矩阵 U ，增广向量 b 也相应更新。此时，原方程组 $Ax = b$ 转换为 $Ux = b^{\text{new}}$ ，可通过上三角矩阵方程求解。

调用 `void solve_upper_triangular_equation(Matrix* A, Vector* b, Vector* x);`

`void solve_linear_equations(Matrix* A, Vector* b, Vector* x);`

7. 辅助函数

7.1 向量

- 初始化

参数功能：指向要操作对象的指针，要初始化的向量大小；返回值：0：失败；1：成功

`int init_vector(Vector* vec, int size)`

- 释放

`void free_vector(Vector* vec)`

- 是否导出

参数功能：指向要操作对象的指针，导出文件名

调用 `void export_vector_to_txt(Vector* vector, char* filename);`

这个是在导出函数的基础上加了一个if语句询问用户是否导出

`void if_export_vec_result(Vector* vec, char* filename)`

- 向量模长

`double vector_norm(double* vec, int size)`

- 向量点积

`double dot_product(double* vec1, double* vec2, int size)`

- 向量cos

调用上面两个函数

`double calculate_cos(double* vec1, double* vec2, int size)`

7.2 矩阵

参数功能等与向量时类似

- 初始化

`int init_matrix(Matrix* mat, int rows, int cols)`

- 释放

`void free_matrix(Matrix* mat)`

- 是否导出

`void if_export_matrix_result(Matrix* mat, char* filename)`

7.3 方程

- 导入

参数功能：指向* vec, * mat的指针，因为要修改指针的内容，所以用二级指针

`void import_linear_equation(Vector** vec, Matrix** mat)`

7.4 其他

- 排序

- 冒泡

`void bubbleSort(double arr[], int n)`

`void elbbubSort(double arr[], int n)` (冒泡反排序)

- 查找向量中最大的c1个元素的索引用

使用方法见1. 单个向量操作->统计与排序->[查找向量中最大的c1个元素的索引](#)

参数功能：指向要操作对象的指针，c1

对结构体排序

`void bubbleSort_struct_index_vaule(index_vaule* arr, int n)`

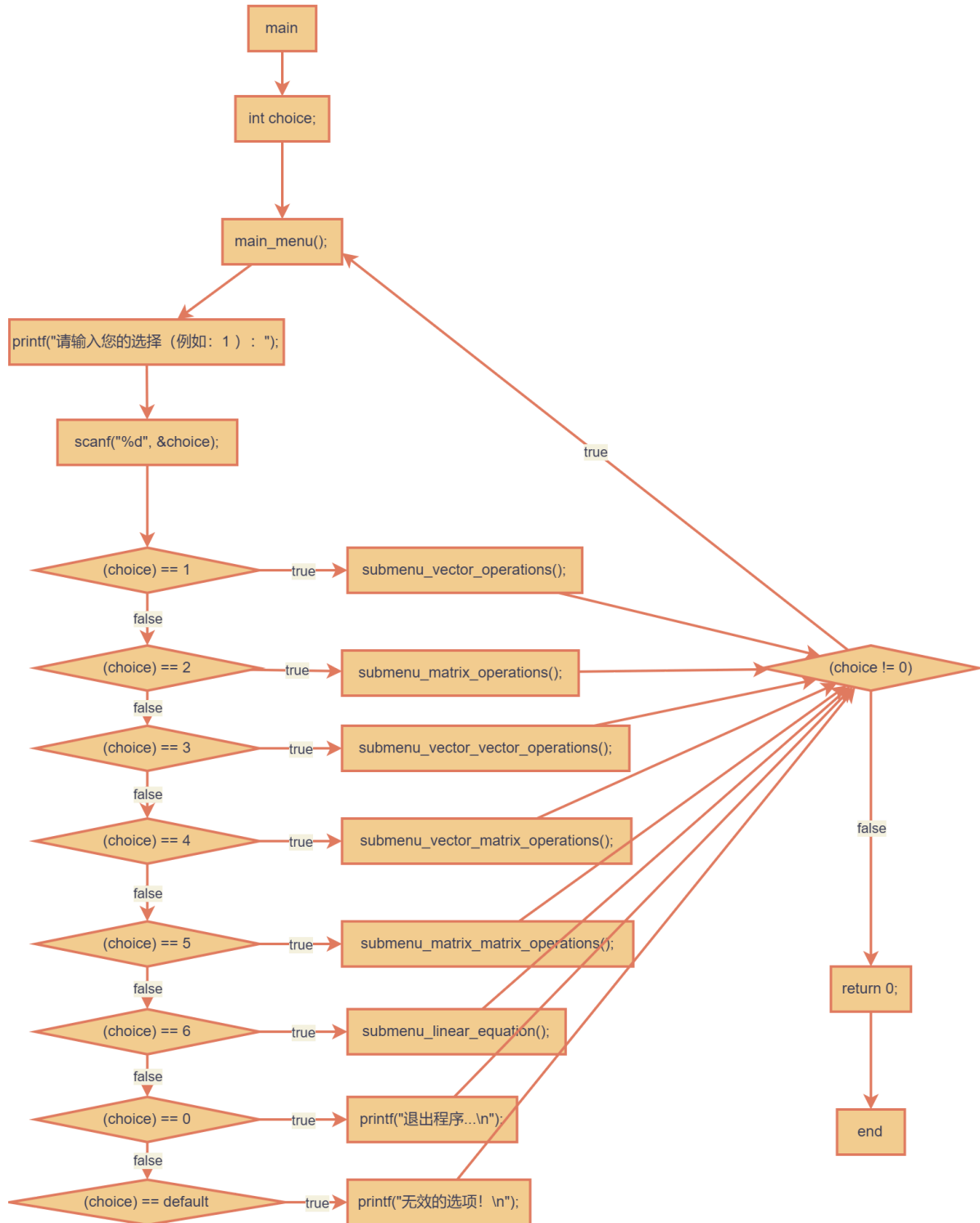
参数功能：指向要操作对象的指针，c1，文件名

c1索引导出

```
void export_indices_to_txt(int* indices, int c1, char* filename)
```

8. 主函数及子菜单

8.1 主函数



8.2子菜单

思路：进入每个子菜单会强制导入，然后用switch-case选择不同选项。

每个子菜单只在开始时开对应需要的内存，在子菜单结束时释放。

例如单个向量操作子菜单只在开始时导入开内存，在子菜单结束时释放，中间如果有选项需要内存就临时分配并释放。

- 单个向量操作

```
void submenu_vector_operations();
```

- 单个矩阵操作

```
void submenu_matrix_operations();
```

- 向量与向量操作

```
void submenu_vector_vector_operations();
```

- 向量与矩阵操作

```
void submenu_vector_matrix_operations();
```

- 矩阵与矩阵操作

```
void submenu_matrix_matrix_operations();
```

- 线性方程

```
void submenu_linear_equation_operations();
```

三、界面设计

1.主菜单

```
void main_menu() {  
    printf("\n***** 主菜单 *****\n");  
    printf("    1. 单个向量操作\n");  
    printf("    2. 单个矩阵操作\n");  
    printf("    3. 向量与向量操作\n");  
    printf("    4. 向量与矩阵操作\n");  
    printf("    5. 矩阵与矩阵操作\n");  
    printf("    6. 线性方程\n");  
    printf("    0. 退出\n");  
    printf("*****\n");  
}
```

```

***** 主菜单 *****
1. 单个向量操作
2. 单个矩阵操作
3. 向量与向量操作
4. 向量与矩阵操作
5. 矩阵与矩阵操作
6. 线性方程
0. 退出
*****
请输入您的选择（例如：1）：|

```

2.子菜单

• 单个向量操作子菜单

```

void show_submenu_vector_operations() {
    printf("\n===== 单个向量操作 =====\n");
    printf(" 1. 导入导出\n");
    printf("    1.1. 导入向量数据 (A)\n");
    printf("    1.2. 导出向量数据 (B)\n");
    printf(" 2. 显示与切片\n");
    printf("    2.1. 显示向量 (C)\n");
    printf("    2.2. 显示向量元素 (D)\n");
    printf("    2.3. 向量切片 (E)\n");
    printf(" 3. 初始化与变换\n");
    printf("    3.1. 初始化列向量 (F)\n");
    printf("    3.2. 向量绝对值 (G)\n");
    printf("    3.3. 向量元素取倒数 (H)\n");
    printf("    3.4. 向量元素平方 (I)\n");
    printf("    3.5. 向量归一化 (J)\n");
    printf(" 4. 统计与排序\n");
    printf("    4.1. 查找向量中的最大元素 (K)\n");
    printf("    4.2. 查找向量中的最小元素 (L)\n");
    printf("    4.3. 查找向量中最大的c1个元素的索引 (M)\n");
    printf("    4.4. 查找向量中最小的c1个元素的索引 (N)\n");
    printf("    4.5. 从小到大排序向量 (O)\n");
    printf("    4.6. 从大到小排序向量 (P)\n");
    printf("    4.7. 向量范数 (Q)\n");
    printf(" 5. 算术操作\n");
    printf("    5.1. 向量除以标量 (R)\n");
    printf("    5.2. 向量乘以标量 (S)\n");
    printf(" 0. 退出子菜单\n");
    printf("-----\n");
}

```

```

===== 单个向量操作 =====
1. 导入导出
    1.1. 导入向量数据 (A)
    1.2. 导出向量数据 (B)
2. 显示与切片
    2.1. 显示向量 (C)
    2.2. 显示向量元素 (D)
    2.3. 向量切片 (E)
3. 初始化与变换
    3.1. 初始化列向量 (F)
    3.2. 向量绝对值 (G)
    3.3. 向量元素取倒数 (H)
    3.4. 向量元素平方 (I)
    3.5. 向量归一化 (J)
4. 统计与排序
    4.1. 查找向量中的最大元素 (K)
    4.2. 查找向量中的最小元素 (L)
    4.3. 查找向量中最大的c1个元素的索引 (M)
    4.4. 查找向量中最小的c1个元素的索引 (N)
    4.5. 从小到大排序向量 (O)
    4.6. 从大到小排序向量 (P)
    4.7. 向量范数 (Q)
5. 算术操作
    5.1. 向量除以标量 (R)
    5.2. 向量乘以标量 (S)
0. 退出子菜单
-----
请先导入向量，输入向量所在文件名：|

```

• 单个矩阵操作子菜单

```

void show_submenu_matrix_operations() {
    printf("\n===== 单个矩阵操作 =====\n");
    printf(" 1. 导入导出\n");
    printf("    1.1. 导入矩阵数据 (A)\n");
    printf("    1.2. 导出矩阵数据 (B)\n");
    printf(" 2. 显示与切片\n");
    printf("    2.1. 显示整个矩阵 (C)\n");
    printf("    2.2. 显示矩阵元素 (D)\n");
    printf("    2.3. 矩阵行切片 (E)\n");
    printf("    2.4. 矩阵列切片 (F)\n");
    printf("    2.5. 矩阵根据位置矩阵切片 (G)\n");
    printf(" 3. 变换\n");
    printf("    3.1. 矩阵元素取倒数 (H)\n");
    printf("    3.2. 矩阵元素平方 (I)\n");
    printf("    3.3. 替换矩阵指定列 (J)\n");
    printf("    3.4. 矩阵转置 (K)\n");
    printf(" 4. 统计\n");
}

```

```
printf("    4.1. 矩阵所有元素求和 (L)\n");
printf("    4.2. 查找矩阵中的最大元素 (M)\n");
printf("    4.3. 每行的最大元素 (N)\n");
printf("    4.4. 每行的最小元素 (O)\n");
printf("    4.5. 整行元素都为0的行编号 (P)\n");
printf("    4.6. 矩阵范数及特征值 (Q)\n");
printf("    4.7. 矩阵列向量间的夹角Cos值 (R)\n");
printf("  5. 算术操作\n");
printf("    5.1. 矩阵乘以标量 (S)\n");
printf("  6. 特殊矩阵生成\n");
printf("    6.1. 生成随机对称矩阵 (T)\n");
printf("  a. 刷新子菜单 (a)\n");
printf("  0. 退出子菜单\n");
printf("-----\n");
}
```

```

===== 单个矩阵操作 =====
1. 导入导出
    1.1. 导入矩阵数据 (A)
    1.2. 导出矩阵数据 (B)
2. 显示与切片
    2.1. 显示整个矩阵 (C)
    2.2. 显示矩阵元素 (D)
    2.3. 矩阵行切片 (E)
    2.4. 矩阵列切片 (F)
    2.5. 矩阵根据位置矩阵切片 (G)
3. 变换
    3.1. 矩阵元素取倒数 (H)
    3.2. 矩阵元素平方 (I)
    3.3. 替换矩阵指定列 (J)
    3.4. 矩阵转置 (K)
4. 统计
    4.1. 矩阵所有元素求和 (L)
    4.2. 查找矩阵中的最大元素 (M)
    4.3. 每行的最大元素 (N)
    4.4. 每行的最小元素 (O)
    4.5. 整行元素都为0的行编号 (P)
    4.6. 矩阵范数及特征值 (Q)
    4.7. 矩阵列向量间的夹角Cos值 (R)
5. 算术操作
    5.1. 矩阵乘以标量 (S)
6. 特殊矩阵生成
    6.1. 生成随机对称矩阵 (T)
a. 刷新子菜单 (a)
0. 退出子菜单

-----
请先导入矩阵，输入矩阵所在文件名：|

```

• 向量与向量操作子菜单

```

void show_submenu_vector_vector_operations() {
    printf("\n==== 向量与向量操作 ====\n");
    printf("1. 算术操作\n");
    printf("    1.1 向量加法 (A)\n");
    printf("    1.2 向量减法 (B)\n");
    printf("    1.3 向量按元素除法 (C)\n");
    printf("    1.4 向量按元素乘法 (D)\n");
    printf("2. 余弦与点积\n");
    printf("    2.1 余弦 (E)\n");
    printf("    2.2 点积 (F)\n");
    printf("3. 外积\n");
    printf("    3.1 向量外积 (G)\n");
    printf("a. 导入新向量 (a)\n");
}

```

```

printf("b. 打印当前向量 (b)\n");
printf("0. 退出子菜单\n");
printf("-----\n");
}

```

==== 向量与向量操作 ====

1. 算术操作

1.1 向量加法 (A)

1.2 向量减法 (B)

1.3 向量按元素除法 (C)

1.4 向量按元素乘法 (D)

2. 余弦与点积

2.1 余弦 (E)

2.2 点积 (F)

3. 外积

3.1 向量外积 (G)

a. 导入新向量 (a)

b. 打印当前向量 (b)

0. 退出子菜单

请导入第一个向量，输入向量1所在文件名：|

• 向量与矩阵操作子菜单

```

void show_submenu_vector_matrix_operations() {
printf("\n==== 向量与矩阵操作 ==== \n");
printf("1. 矩阵乘以向量 (M)\n");
printf("2. 向量乘以矩阵 (N)\n");
printf("a. 导入新向量和矩阵 (a)\n");
printf("b. 打印当前向量和矩阵 (b)\n");
printf("0. 退出子菜单\n");
printf("-----\n");
}

```

==== 向量与矩阵操作 ====

1. 矩阵乘以向量 (M)

2. 向量乘以矩阵 (N)

a. 导入新向量和矩阵 (a)

b. 打印当前向量和矩阵 (b)

0. 退出子菜单

请先导入矩阵，输入矩阵所在文件名：|

- 矩阵与矩阵操作子菜单

```
void show_submenu_matrix_matrix_operations() {  
    printf("\n==== 矩阵与矩阵操作 ==== \n");  
    printf(" 1.矩阵加法 (A)\n");  
    printf(" 2.矩阵减法 (B)\n");  
    printf(" 3.矩阵按元素乘法 (C)\n");  
    printf(" 4.矩阵按元素除法 (D)\n");  
    printf(" 5.AB型矩阵相乘 (E)\n");  
    printf(" 6.ATBA型矩阵相乘 (F)\n");  
    printf(" a. 导入新矩阵 (a)\n");  
    printf(" b. 打印当前矩阵 (b)\n");  
    printf(" 0. 退出子菜单\n");  
    printf("-----\n");  
}
```

==== 矩阵与矩阵操作 ====

- 1.矩阵加法 (A)
- 2.矩阵减法 (B)
- 3.矩阵按元素乘法 (C)
- 4.矩阵按元素除法 (D)
- 5.AB型矩阵相乘 (E)
- 6.ATBA型矩阵相乘 (F)
- a. 导入新矩阵 (a)
- b. 打印当前矩阵 (b)
- 0. 退出子菜单

请导入第一个矩阵，输入矩阵1所在文件名：|

- 线性方程子菜单

```
void show_submenu_linear_equation_operations() {  
    printf("\n==== 线性方程 ==== \n");  
    printf("1. 上三角矩阵的逆 (A)\n");  
    printf("2. 上三角矩阵方程求解 (B)\n");  
    printf("3. 一般线性方程求解 (C)\n");  
    printf("a. 导入新方程 (a)\n");  
    printf("0. 退出子菜单\n");  
    printf("-----\n");  
}
```

```
==== 线性方程 ====
1. 上三角矩阵的逆 (A)
2. 上三角矩阵方程求解 (B)
3. 一般线性方程求解 (C)
a. 导入新方程 (a)
0. 退出子菜单
-----
请输入您的选择 (例如: A ) : |
```

四、测试和排错

1.测试

在Windows 11 系统下测试：

1.1主菜单测试

***** 主菜单 *****

1. 单个向量操作
2. 单个矩阵操作
3. 向量与向量操作
4. 向量与矩阵操作
5. 矩阵与矩阵操作
6. 线性方程
0. 退出

请输入您的选择（例如：1）：q
无效的选项！

***** 主菜单 *****

1. 单个向量操作
2. 单个矩阵操作
3. 向量与向量操作
4. 向量与矩阵操作
5. 矩阵与矩阵操作
6. 线性方程
0. 退出

请输入您的选择（例如：1）：1

===== 单个向量操作 =====

1. 导入导出

1.1. 导入向量数据 (A)

***** 主菜单 *****

1. 单个向量操作
2. 单个矩阵操作
3. 向量与向量操作
4. 向量与矩阵操作
5. 矩阵与矩阵操作
6. 线性方程
0. 退出

请输入您的选择（例如：1）：0
退出程序...

Process exited after 14.18 seconds w

Press ANY key to exit...|



1.2 单个向量操作

- 3.4. 向量元素平方 (I)
- 3.5. 向量归一化 (J)
- 4. 统计与排序
 - 4.1. 查找向量中的最大元素 (K)
 - 4.2. 查找向量中的最小元素 (L)
 - 4.3. 查找向量中最大的c1个元素的索引 (M)
 - 4.4. 查找向量中最小的c1个元素的索引 (N)
 - 4.5. 从小到大排序向量 (O)
 - 4.6. 从大到小排序向量 (P)
 - 4.7. 向量范数 (Q)
- 5. 算术操作
 - 5.1. 向量除以标量 (R)
 - 5.2. 向量乘以标量 (S)
- 0. 退出子菜单

请先导入向量，输入向量所在文件名：v1
无法打开文件 v1
向量导入失败！
检查文件名是否正确或文件是否为空
请先导入向量，输入向量所在文件名：v1.txt
向量导入成功！
请输入您的选择（例如：A）：C
当前向量：

5.00
2.00
8.00
请输入您的选择（例如：A）：D
查看向量第i个元素，输入i：0
向量的第0个元素是 5.000000
请输入您的选择（例如：A）：E
输入切片长度：2
输入切片索引：0
1
向量切片成功

5.00
2.00
是否导出？(y/n)：y
向量数据成功导出至 sliced_vec.txt
切片向量导出至：sliced_vec.txt

- 4.3. 查找向量中最大的c1个元素的索引
- 4.4. 查找向量中最小的c1个元素的索引
- 4.5. 从小到大排序向量 (O)
- 4.6. 从大到小排序向量 (P)
- 4.7. 向量范数 (Q)
- 5. 算术操作
 - 5.1. 向量除以标量 (R)
 - 5.2. 向量乘以标量 (S)
- 0. 退出子菜单

请先导入向量，输入向量所在文件名：v1.txt
向量导入成功！
请输入您的选择（例如：A）：F
请输入列向量的大小：6
请输入要初始化的值：2.6
列向量初始化成功！
2.60
2.60

sliced_vec.txt		
文件	编辑	查看
5.00		
2.00		

```
2.60
2.60
2.60
2.60
是否要将列向量导出到文件? (y/n): y
请输入导出向量所至文件名: v3.txt
向量数据成功导出至 v3.txt
```

v3.txt

文件	编辑	查
2.60		
2.60		
2.60		
2.60		
2.60		
2.60		

```
请先导入向量, 输入向量所在文件名: v1
无法打开文件 v1
向量导入失败!
检查文件名是否正确或文件是否为空
请先导入向量, 输入向量所在文件名: v1.txt
向量导入成功!
请输入您的选择 (例如: A ): C
当前向量:
  5.00
  2.00
  8.00
 -4.00
请输入您的选择 (例如: A ): H
向量元素已取倒数
请输入您的选择 (例如: A ): C
当前向量:
  0.20
  0.50
  0.12
 -0.25
请输入您的选择 (例如: A ): C
当前向量:
  5.00
  2.00
  8.00
 -4.00
请输入您的选择 (例如: A ): G
向量已取绝对值
请输入您的选择 (例如: A ): C
当前向量:
  5.00
  2.00
  8.00
  4.00
```

请先导入向量，输入向量所在文件名：v1.txt

向量导入成功！

请输入您的选择（例如：A）：C

当前向量：

5.00

2.00

8.00

-4.00

请输入您的选择（例如：A）：I

向量元素已取平方

请输入您的选择（例如：A）：C

当前向量：

25.00

4.00

64.00

16.00

请输入您的选择（例如：A）：B

请输入导出向量所至文件名：v0.txt

向量数据成功导出至 v0.txt



```

请先导入向量，输入向量所在文件名： v1.txt
向量导入成功！
请输入您的选择（例如：A）： C
当前向量：
  5.00
  2.00
  8.00
 -4.00
请输入您的选择（例如：A）： K
向量中的最大元素是： 8.000000
请输入您的选择（例如：A）： L
向量中的最小元素是： -4.000000
请输入您的选择（例如：A）： M
请输入要查找的最大元素个数c1: 2
第0大的元素位置： 2
第1大的元素位置： 0
是否要将最大的2个元素的索引导出到文件？(y/n): n
请输入您的选择（例如：A）： N
请输入要查找的最小元素个数c1: 22
c1无效！
是否要将最小的22个元素的索引导出到文件？(y/n): n
请输入您的选择（例如：A）： N
请输入要查找的最小元素个数c1: 2
第0小的元素位置： 3
第1小的元素位置： 1
是否要将最小的2个元素的索引导出到文件？(y/n): y
请输入导出文件名： top_2.txt
索引已成功导出到 top_2.txt

```



```

请输入您的选择（例如：A）： 0
向量已按从小到大排列
请输入您的选择（例如：A）： C
当前向量：
 -4.00
  2.00
  5.00
  8.00
请输入您的选择（例如：A）： Q
请选择范数类型（1: L1范数，2: L2范数，3: 无穷范数）： 1
向量范数为： 19.000000
请输入您的选择（例如：A）： Q
请选择范数类型（1: L1范数，2: L2范数，3: 无穷范数）： 2
向量范数为： 10.440307
请输入您的选择（例如：A）： Q
请选择范数类型（1: L1范数，2: L2范数，3: 无穷范数）： 3
向量范数为： 8.000000

```

```
请输入您的选择（例如：A）：C
当前向量：
  5.00
  2.00
  8.00
 -4.00
请输入您的选择（例如：A）：R
请输入标量：0
标量为0.000000，无法除以标量！
请输入您的选择（例如：A）：R
请输入标量：2
向量已除以标量
请输入您的选择（例如：A）：C
当前向量：
  2.50
  1.00
  4.00
 -2.00
请输入您的选择（例如：A）：C
当前向量：
  5.00
  2.00
  8.00
 -4.00
请输入您的选择（例如：A）：S
请输入标量：4
向量已乘以标量
请输入您的选择（例如：A）：C
当前向量：
 20.00
  8.00
 32.00
-16.00
```

1.3 单个矩阵操作

0. 退出子菜单

```
-----  
请先导入矩阵，输入矩阵所在文件名： 2  
无法打开文件 2  
矩阵导入失败！  
检查文件名是否正确或文件是否为空  
请先导入矩阵，输入矩阵所在文件名： m1.txt  
矩阵导入成功！  
请输入您的选择（例如： A ）： C  
1.00 0.50  
0.50 0.50  
请输入您的选择（例如： A ）： D  
请输入要显示的行和列（例如： 0 1 ）： 0 0  
第0行第0列的元素是： 1.00  
请输入您的选择（例如： A ）： E  
请输入要切片的行号： 0  
矩阵行切片成功  
1.00 0.50  
是否要导出切片到文件？（Y/N）： y  
切片已导出到row_slice.txt
```



row_slice.txt

文件 编辑 查看

|1.00
0.50

请输入您的选择（例如：A）：G
请输入切片的大小：
请输入要切片的行数：
0
请输入要切片的列数：
0
切片的大小有误！
请输入切片的大小：
请输入要切片的行数：
1
请输入要切片的列数：
2
输入切片行索引：
0 1
输入切片列索引：
1

0.50 0.50
是否要导出结果到文件？（Y/N）：n

请输入您的选择（例如：A）：F
请输入要切片的列号：0
矩阵列切片成功
1.00
0.50

是否要导出切片到文件？（Y/N）：y
切片已导出到col_slice.txt



col_slice.txt

文件

编辑

查看

1.00

0.50

请输入您的选择（例如：A）：C
1.00 0.50
0.50 0.50
请输入您的选择（例如：A）：H
矩阵元素已取倒数
请输入您的选择（例如：A）：C
1.00 2.00
2.00 2.00
请输入您的选择（例如：A）：I
矩阵元素已取平方
请输入您的选择（例如：A）：C
1.00 4.00
4.00 4.00
请输入您的选择（例如：A）：J
请输入要替换的列号：
1
请输入新列的元素（输入2个值）：
5 8
矩阵第1列已替换
请输入您的选择（例如：A）：C
1.00 5.00
4.00 8.00
请输入您的选择（例如：A）：K
矩阵已转置
请输入您的选择（例如：A）：C
1.00 4.00
5.00 8.00
请输入您的选择（例如：A）：B
请输入导出矩阵所至文件名：m0.txt
矩阵数据成功导出至 m0.txt



请输入您的选择 (例如: A) : C

1.00 0.50

0.50 0.50

请输入您的选择 (例如: A) : L

矩阵所有元素的和为: 2.50

请输入您的选择 (例如: A) : M

矩阵中的最大元素是: 1.00

请输入您的选择 (例如: A) : N

每行的最大元素为:

1.00

0.50

请输入您的选择 (例如: A) : O

每行的最小元素为:

0.50

0.50

请输入您的选择 (例如: A) : P

没有0行!

请输入您的选择 (例如: A) : A

矩阵已导入, 是否导入一个新矩阵覆盖当前矩阵? (y/n): y

请输入新矩阵所在文件名: m2.txt

请输入您的选择 (例如: A) : C

2.00 4.00 9.00

8.00 0.00 6.00

0.00 0.00 0.00

3.00 2.00 5.00

0.00 0.00 0.00

请输入您的选择 (例如: A) : P

整行元素都为0的行编号为:

2

4

是否要导出整行元素都为0的行编号到文件? (Y/N) : n

请输入您的选择 (例如: A) : Q

请选择类型 (1: L1范数, 2: L2范数, 3: 无穷范数, 4: 特征值) : 1

1范数为: 20.000000

请输入您的选择 (例如: A) : Q

请选择类型 (1: L1范数, 2: L2范数, 3: 无穷范数, 4: 特征值) : 2

2范数为: 1.959874

请输入您的选择 (例如: A) : Q

请选择类型 (1: L1范数, 2: L2范数, 3: 无穷范数, 4: 特征值) : 3

无穷范数为: 13.000000

请输入您的选择 (例如: A) : Q

请选择类型 (1: L1范数, 2: L2范数, 3: 无穷范数, 4: 特征值) : 4

A不是方阵! 未求得特征值!

请输入您的选择 (例如: A) : C

4.00 1.00

2.00 3.00

请输入您的选择 (例如: A) : Q

请选择类型 (1: L1范数, 2: L2范数, 3: 无穷范数, 4: 特征值) : 4

主特征值为: 5.000000

对应特征向量为:

0.707107

0.707107

请输入您的选择（例如：A）：C

2.00	4.00	9.00	9.00	4.00
8.00	0.00	6.00	4.00	7.00
0.00	0.00	0.00	1.00	0.00
3.00	2.00	5.00	4.00	5.00
0.00	0.00	0.00	7.00	3.00

请输入您的选择（例如：A）：R

矩阵列向量间的夹角Cos值为：

0.36

0.77

0.55

0.90

0.86

0.77

0.58

0.82

0.87

0.83

请输入您的选择（例如：A）：S

请输入标量：9

矩阵已乘以标量

请输入您的选择（例如：A）：C

18.00	36.00	81.00	81.00	36.00
72.00	0.00	54.00	36.00	63.00
0.00	0.00	0.00	9.00	0.00
27.00	18.00	45.00	36.00	45.00
0.00	0.00	0.00	63.00	27.00

请输入您的选择（例如：A）：T

请输入对称矩阵大小：3

随机对称矩阵生成成功

-0.32 -0.60 -0.86

-0.60 -0.06 -0.89

-0.86 -0.89 -0.66

是否导出？(y/n)：y

随机对称矩阵数据成功导出至 random_sym_mat.txt

请输入您的选择（例如：A）：T

请输入对称矩阵大小：3

随机对称矩阵生成成功

0.26 0.20 0.35

0.20 -0.96 0.20

0.35 0.20 0.55

是否导出？(y/n)：y

随机对称矩阵数据成功导出至 random_sym_mat.txt

请输入您的选择（例如：A）：T

请输入对称矩阵大小：3

随机对称矩阵生成成功

0.26 0.20 0.35

0.20 -0.96 0.20

0.35 0.20 0.55

是否导出？(y/n)：y

随机对称矩阵数据成功导出至 random_sym_mat.txt

请输入您的选择（例如：A）：T

请输入对称矩阵大小：3

随机对称矩阵生成成功

0.26 0.20 0.35

0.20 -0.96 0.20

0.35 0.20 0.55


1.4 向量与向量操作

```
==== 向量与向量操作 ====
1. 算术操作
    1.1 向量加法 (A)
    1.2 向量减法 (B)
    1.3 向量按元素除法 (C)
    1.4 向量按元素乘法 (D)
2. 余弦与点积
    2.1 余弦 (E)
    2.2 点积 (F)
3. 外积
    3.1 向量外积 (G)
a. 导入新向量 (a)
b. 打印当前向量 (b)
0. 退出子菜单
-----
请导入第一个向量，输入向量1所在文件名： v1.txt
向量1导入成功！
请导入第二个向量，输入向量2所在文件名： v2.txt
两个向量大小不同，请重新导入
请导入第二个向量，输入向量2所在文件名： v3.txt
向量2导入成功！
向量1向量2导入成功！
请输入您的选择（例如：A）： b
向量1:
5.00
2.00
8.00
-4.00
向量2:
1.00
0.00
7.00
-8.00
```

```

请输入您的选择（例如：A）：A
6.00
2.00
15.00
-12.00
是否要导出结果到文件？（Y/N）：y
请输入导出文件名：v_1_add_3.txt
向量数据成功导出至 v_1_add_3.txt
请输入您的选择（例如：A）：B
4.00
2.00
1.00
4.00
是否要导出结果到文件？（Y/N）：n
请输入您的选择（例如：A）：C
除数不能为零！
5.00
0.00
1.14
0.50
是否要导出结果到文件？（Y/N）：n
请输入您的选择（例如：A）：D
5.00
0.00
56.00
32.00
是否要导出结果到文件？（Y/N）：n

```

 v_1_add_3.txt

文件 编辑 查看

```


6.00
2.00
15.00
-12.00

```

```

请输入您的选择（例如：A）：E
向量1向量2夹角余弦为：0.834290
请输入您的选择（例如：A）：F
向量1向量2点积为：93.000000
请输入您的选择（例如：A）：G
5.00 0.00 35.00 -40.00
2.00 0.00 14.00 -16.00
8.00 0.00 56.00 -64.00
-4.00 -0.00 -28.00 32.00
是否要导出结果到文件？（Y/N）：Y
请输入导出文件名：m_v_1_3.txt
矩阵数据成功导出至 m_v_1_3.txt

```

 m_v_1_3.txt

文件 编辑 查看

```

5.00 0.00 35.00 -40.00
2.00 0.00 14.00 -16.00
8.00 0.00 56.00 -64.00
-4.00 -0.00 -28.00 32.00

```

1.5 向量与矩阵操作

```
请先导入矩阵，输入矩阵所在文件名： m1.txt
矩阵导入成功！
请先导入向量，输入向量所在文件名： v1.txt
向量导入成功！
请输入您的选择（例如： M ）： b
向量：
5.00
2.00
8.00
-4.00
矩阵：
4.00  1.00  8.00
2.00  3.00  0.00
9.00  6.00  4.00
9.00  8.00  5.00
请输入您的选择（例如： M ）： M
矩阵的列数与向量的大小不同！
请输入您的选择（例如： M ）： N
78.00 -20.00 53.00 0.00 是否要导出结果到文件？（Y/N）： y
请输入导出文件名： result.txt
向量数据成功导出至 result.txt
```



```
请输入您的选择（例如：M）：a
是否导入新向量和矩阵1？（y/n）：y
请导入新矩阵，输入新矩阵所在文件名：m1.txt
新矩阵导入成功！
请导入新向量，输入新向量所在文件名：v2.txt
新向量导入成功！
请输入您的选择（例如：M）：N
向量的大小与矩阵的行数不同！
请输入您的选择（例如：M）：b
向量：
1.00
0.00
-8.00
矩阵：
  4.00  1.00  8.00
  2.00  3.00  0.00
  9.00  6.00  4.00
  9.00  8.00  5.00
请输入您的选择（例如：M）：M
-60.00
2.00
-23.00
-31.00
是否要导出结果到文件？（Y/N）：y
请输入导出文件名：result.txt
向量数据成功导出至 result.txt
```



1.6 矩阵与矩阵操作

==== 矩阵与矩阵操作 ====

- 1.矩阵加法 (A)
- 2.矩阵减法 (B)
- 3.矩阵按元素乘法 (C)
- 4.矩阵按元素除法 (D)
- 5.AB型矩阵相乘 (E)
- 6.ATBA型矩阵相乘 (F)
- a. 导入新矩阵 (a)
- b. 打印当前矩阵 (b)
0. 退出子菜单

请导入第一个矩阵，输入矩阵1所在文件名： m1.txt
矩阵1导入成功！

请导入第二个矩阵，输入矩阵2所在文件名： m2.txt
矩阵2导入成功！

请输入您的选择（例如： A ）： b

矩阵1:

4.00	1.00	8.00
2.00	3.00	0.00
9.00	6.00	4.00
9.00	8.00	5.00

矩阵2:

2.00	4.00	9.00	9.00	4.00
8.00	0.00	6.00	4.00	7.00
0.00	0.00	0.00	1.00	0.00
3.00	2.00	5.00	4.00	5.00
0.00	0.00	0.00	7.00	3.00

请输入您的选择（例如： A ）： A

矩阵大小不同！请输入您的选择（例如： A ）： |

```
矩阵大小不同！请输入您的选择（例如：A）：a
是否导入两个新矩阵，覆盖当前矩阵？(y/n): y
请导入第一个矩阵，输入矩阵1所在文件名：m1.txt
矩阵1导入成功！
请导入第二个矩阵，输入矩阵2所在文件名：m2.txt
矩阵2导入成功！
请输入您的选择（例如：A）：b
矩阵1:
  4.00  1.00  8.00
  2.00  3.00  0.00
  9.00  6.00  4.00
矩阵2:
  2.00  4.00  9.00
  8.00  0.00  6.00
  0.00  0.00  0.00
请输入您的选择（例如：A）：A
  6.00  5.00  17.00
 10.00  3.00  6.00
  9.00  6.00  4.00
是否要导出结果到文件？(Y/N)：n
请输入您的选择（例如：A）：B
  2.00 -3.00 -1.00
 -6.00  3.00 -6.00
  9.00  6.00  4.00
是否要导出结果到文件？(Y/N)：n
请输入您的选择（例如：A）：C
  8.00  4.00  72.00
 16.00  0.00  0.00
  0.00  0.00  0.00
是否要导出结果到文件？(Y/N)：n
请输入您的选择（例如：A）：D
  2.00  0.25  0.89
  0.25  0.00  0.00
  0.00  0.00  0.00
是否要导出结果到文件？(Y/N)：y
请输入导出文件名：result.txt
矩阵数据成功导出至 result.txt
```



result.txt

文件 编辑 查看

```
2.00 0.25 0.89
0.25 0.00 0.00
0.00 0.00 0.00
```

```

请输入您的选择（例如：A）：E
16.00 16.00 42.00
28.00 8.00 36.00
66.00 36.00 117.00
是否要导出结果到文件？（Y/N）：n
请输入您的选择（例如：A）：F
24.00 16.00 48.00
26.00 4.00 27.00
16.00 32.00 72.00
是否要导出结果到文件？（Y/N）：Y
请输入导出文件名：result.txt
矩阵数据成功导出至 result.txt

```



result.txt

文件 编辑 查看

```

24.00 16.00 48.00
26.00 4.00 27.00
16.00 32.00 72.00

```

1.7 线性方程

```

==== 线性方程 ====
1. 上三角矩阵的逆 (A)
2. 上三角矩阵方程求解 (B)
3. 一般线性方程求解 (C)
a. 导入新方程 (a)
b. 打印当前线性方程 (b)
0. 退出子菜单
-----
请输入您的选择（例如：A）：A
请导入上三角矩阵，输入上三角矩阵所在文件名：m2.txt
矩阵导入成功！
2.00 4.00 9.00
0.00 4.00 6.00
0.00 6.00 5.00
该矩阵不是上三角矩阵！
请导入上三角矩阵，输入上三角矩阵所在文件名：m1.txt
矩阵导入成功！
上三角矩阵为：
2.00 4.00 9.00
0.00 4.00 6.00
0.00 0.00 5.00
上三角矩阵的逆为：
0.50 -0.50 -0.30
0.00 0.25 -0.30
0.00 0.00 0.20
是否要导出结果到文件？（Y/N）：y
请输入导出文件名：invert.txt
矩阵数据成功导出至 invert.txt

```

```
请输入您的选择（例如：A）：B
请先导入上三角矩阵方程
请导入矩阵，输入矩阵所在文件名：m1.txt
矩阵导入成功！
请导入向量，输入向量所在文件名：v1.txt
向量导入成功！
  2.00  4.00  9.00
  0.00  4.00  6.00
  0.00  0.00  5.00
* x =
5.00
2.00
8.00
方程解为：
-0.90
-1.90
1.60
是否要导出结果到文件？（Y/N）：y
请输入导出文件名：result.txt
向量数据成功导出至 result.txt
```

 result.txt

文件 编辑

-0.90
-1.90
1.60

```

请输入您的选择（例如：A）：B
请先导入上三角矩阵方程
请导入矩阵，输入矩阵所在文件名：m2.txt
矩阵导入成功！
请导入向量，输入向量所在文件名：v1.txt
向量导入成功！
该矩阵不是上三角矩阵！
请输入您的选择（例如：A）：C
请先导入线性方程
请导入矩阵，输入矩阵所在文件名：m2.txt
矩阵导入成功！
请导入向量，输入向量所在文件名：v2.txt
向量导入成功！
  2.00  4.00  9.00
  1.00  4.00  6.00
  5.00  6.00  5.00
* x =
1.00
0.00
6.00
-8.00
方程解为：
1.52
-0.12
-0.17
0.00
是否要导出结果到文件？（Y/N）：y
请输入导出文件名：result.txt
向量数据成功导出至 result.txt

```

result.txt		
文件	编辑	查看
1.52		
-0.12		
-0.17		
0.00		

2. 排错

时间有限，只写了其中几个错误及排错过程。

- ✓ 在主函数里用 `int choice` 记录选项，导致输入字母时会导致异常

```

int main() {
    int choice;
    do {
        main_menu();
        printf("请输入您的选择（例如：1）：");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                //...
            case 6:
                submenu_linear_equation_operations();
                break;
            case 0:
                printf("退出程序...\n");
                break;
            default:
                printf("无效的选项！\n");
                break;
        }
    } while (choice != 0);
}

```

```

    return 0;
}

```

修改为 char

```

int main() {
    char choice;
    do {
        main_menu();
        printf("请输入您的选择（例如：1）：");
        scanf(" %c", &choice);
        switch (choice) {
            case '1':
                submenu_vector_operations();
                break;
            //...
            default:
                printf("无效的选项！\n");
                break;
        }
    } while (choice != '0');

    return 0;
}

```

正常！

- ☑ 在 submenu_vector_vector_operations();、submenu_linear_equation(); 子菜单里设计了导入函数，用于一次性导入两个结构体。

开始时导入函数是这样设计的：

```

void import_linear_equation(Matrix* mat, Vector* vec) {
    while (mat == NULL || mat->is_init == 0) {
        printf("请导入系数矩阵，输入系数矩阵所在文件名：");
        scanf("%49s", filename);
        mat = import_matrix_from_txt(filename);
        if (mat != NULL && mat->is_init == 1) {
            printf("系数矩阵导入成功！\n");
        } else {
            printf("系数矩阵导入失败！\n");
            printf("检查文件名是否正确或文件是否为空\n");
        }
    }
    while (vec == NULL || vec->is_init == 0) {
        printf("请导入列向量，输入列向量所在文件名：");
        scanf("%49s", filename);
        vec = import_vector_from_txt(filename);
        if (vec != NULL && vec->is_init == 1) {
            printf("列向量导入成功！\n");
        } else {
            printf("列向量导入失败！\n");
            printf("检查文件名是否正确或文件是否为空\n");
        }
    }
}
}

```

看似没有什么问题，但是

调用之前是这样定义的：

```
Matrix* mat = (Matrix*)malloc(sizeof(Matrix));
Vector* vec = (Vector*)malloc(sizeof(Vector));
```

导致在子菜单选择b时会提示 "请先导入线性方程！\n"

```
case 'b':
if (mat == NULL || mat->is_init == 0 || vec == NULL || vec->is_init == 0) {
    printf("请先导入线性方程！\n");
} else {
    print_matrix(mat);
    printf("* x =\n");
    print_vector(vec);
}
break;
```

然后检查哪里出了问题

```
case 'b':
if (mat == NULL || mat->is_init == 0 || vec == NULL || vec->is_init == 0) {
    printf("请先导入线性方程！\n");
    if (mat == NULL) {
        printf("1");
    }
    if (mat->is_init == 0) {
        printf("2");
    }
    if (vec == NULL) {
        printf("3");
    }
    if (vec->is_init == 0) {
        printf("4");
    }
} else {
    print_matrix(mat);
    printf("* x =\n");
    print_vector(vec);
}
break;
```

发现 `mat->is_init == 0` 和 `vec->is_init == 0` 是成立的，经过反复验证，是导入函数的问题

导入函数用一级指针无法修改结构体指针的内容，所以修改为二级指针

```
void import_linear_equation(Vector** vec, Matrix** mat){
    while (*mat == NULL || (*mat)->is_init == 0) {
        printf("请导入矩阵，输入矩阵所在文件名：");
        scanf("%49s", filename);
        *mat = import_matrix_from_txt(filename);
        if (*mat != NULL && (*mat)->is_init == 1) {
            printf("矩阵导入成功！\n");
        } else {
            printf("矩阵导入失败！\n");
        }
    }
}
```

```

        printf("请检查文件名是否正确或文件是否为空\n");
        if (*mat != NULL) {
            free_matrix(*mat);
            *mat = NULL;
        }
    }
}
while (*vec == NULL || (*vec)->is_init == 0) {
    printf("请导入向量，输入向量所在文件名: ");
    scanf("%49s", filename);
    *vec = import_vector_from_txt(filename);
    if (*vec != NULL && (*vec)->is_init == 1) {
        printf("向量导入成功! \n");
    } else {
        printf("向量导入失败! \n");
        printf("请检查文件名是否正确或文件是否为空\n");
        if (*vec != NULL) {
            free_vector(*vec);
            *vec = NULL;
        }
    }
}
}
}

```

正常使用!

- ☑ 开始在幂法中初始化用了已写好的初始化函数

```

Vector v[N]; //用于迭代的向量
for (int i = 0; i < N; i++) {
    init_vector(&v[i], n); // 初始化所有向量
}

```

修改后:

```

int power_method(Matrix* A, double* lambda, double* x, double epsilon, int
N) {
    if (A->cols != A->rows) {
        printf("A不是方阵! ");
        return 0;
    }
    int n = A->cols;
    Vector v[N]; //用于迭代的向量
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < n; j++) {
            v[i].data[j] = 1;
        }
        v[i].is_init = 1; // 初始化所有向量
    }
    double m[N]; //保存向量大小，用于规范化
    for (int k = 0; k < N - 1; k++) {
        Vector w;
        init_vector(&w, n);
        w = multiply_matrix_by_vector(A, &v[k]);
        m[k] = vector_norm(w.data, w.size);
    }
}

```



```

        v[k + 1].size = v[k].size;
        for (int j = 0; j < n; j++) {
            v[k + 1].data[j] = w.data[j] / m[k];
        }
        double temp = fabs( vector_norm(subtract_vectors(v[k], v[k +
1])).data, n) );
        if (temp <= epsilon) { // 收敛
            *lambda = m[k];
            for (int i = 0; i < n; i++) {
                x[i] = v[k + 1].data[i];
            }
            return 1;
        }
    }

    // 如果达到N未收敛
    for (int i = 0; i < N; i++) {
        free_vector(&(v[i])); // 释放向量内存
    }
    return 0;
}

```

中间用了N个v,m,不太好, 修改后:

```

int power_method(Matrix* A, double* lambda, double* x, double epsilon, int
N) {
    if (A->cols != A->rows) {
        printf("A不是方阵! ");
        return 0;
    }

    int n = A->cols;
    Vector v, w;
    init_vector(&v, n); // 初始化迭代向量v
    for (int i = 0; i < n; i++) {
        v.data[i] = 1.0; // 将v的所有元素初始化为1
    }

    double m; // 保存向量大小, 用于规范化
    for (int k = 0; k < N - 1; k++) {
        w = multiply_matrix_by_vector(A, &v); // 计算Av
        m = vector_norm(w.data, w.size); // 计算向量范数

        if (m == 0) {
            printf("向量的范数为零, 无法继续进行迭代。\\n");
            free_vector(&v);
            free_vector(&w);
            return 0;
        }

        // 规范化向量w
        for (int i = 0; i < n; i++) {
            w.data[i] /= m;
        }

        double temp = fabs(vector_norm(subtract_vectors(v, w).data, n)); //
计算v和w的差的范数
    }
}

```

```

        if (temp <= epsilon) { // 判断是否收敛
            *lambda = m; // 设置特征值
            for (int i = 0; i < n; i++) {
                x[i] = w.data[i]; // 设置特征向量
            }
            free_vector(&v);
            free_vector(&w);
            return 1; // 收敛, 返回1
        }

        v = w; // 为下一次迭代准备, 将w赋值给v
    }

    // 如果达到N未收敛
    printf("达到最大迭代次数, 算法未收敛。 \n");
    free_vector(&v);
    free_vector(&w);
    return 0; // 未收敛, 返回0
}

```

只使用两个 `v` 和 `w`, 减少了内存占用

五、总结与体会

1.得意之处

1.1 数据设计: 都设计为一维数组, 内存连续分配, 提高访问速度

```

typedef struct {
    double* data; // 数据
    int size; // 大小
    int is_init; // 是否初始化
} Vector;
typedef struct {
    double* data; // 数据
    int rows; // 行数
    int cols; // 列数
    int is_init; // 是否初始化
} Matrix;

```

1.2 函数分类

```

int main() {
    int choice;
    do {
        main_menu();
        printf("请输入您的选择 (例如: 1 ) : ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                submenu_vector_operations();
                break;
            case 2:
                submenu_matrix_operations();

```

```

        break;
    case 3:
        submenu_vector_vector_operations();
        break;
    case 4:
        submenu_vector_matrix_operations();
        break;
    case 5:
        submenu_matrix_matrix_operations();
        break;
    case 6:
        submenu_linear_equation();
        break;
    case 0:
        printf("退出程序...\n");
        break;
    default:
        printf("无效的选项! \n");
        break;
    }
} while (choice != 0);

return 0;
}

```

如此，只需要在子菜单开头和结束时管理内存

以单个向量操作为例

```

void submenu_vector_operations() {
    show_submenu_vector_operations();
    char choice;
    Vector* vec = NULL;
    while (vec == NULL || vec->is_init == 0) {
        printf("请先导入向量，输入向量所在文件名: ");
        scanf("%49s", filename);
        vec = import_vector_from_txt(filename);
        if (vec != NULL && vec->is_init == 1) {
            printf("向量导入成功! \n");
        } else {
            printf("向量导入失败! \n");
            printf("检查文件名是否正确或文件是否为空\n");
        }
    }

    do {
        printf("请输入您的选择（例如: A ）: ");
        scanf(" %c", &choice);

        switch (choice) {
            case '1':
                printf("1. 导入导出\n");
                printf("1.1. 导入向量数据 (A)\n");
                printf("1.2. 导出向量数据 (B)\n");
                break;
            //...
            case '0':

```

```

        printf("退出子菜单\n");
        break;
    default:
        printf("无效的选项! \n");
        break;
    }

} while (choice != '0');
if (vec != NULL) {
    free_vector(vec);
}
}

```

类似这样，只要在导入时分配内存，结束时释放，很方便。

1.3 辅助函数

将向量和矩阵的初始化和释放，打印等功能封装辅助函数，方便调用。

1.4 注释

写了很多注释（必要的时候）。

1.5 菜单分类

菜单分类很详细，而且界面很清晰。

2. 不足

2.1 未检查错误

没检查很多下面简单列一些，可能也不全

- 导入时，向量如果一行有多个元素或矩阵一行元素数大于列数未判断
- 切片时，如果切片索引一样未判断

2.2 索引从0开始

本程序所有第1个都是第0个，不想改了

2.3 代码重复

存在很多重复的地方，没时间再重删了，其实是设计的时候没想清楚就开始写代码了，导致很多代码重复

2.4 文件名默认

有些文件名为了方便直接的默认了，比如向量切片函数文件名的默认值为"sliced_vec.txt"

3. 总结

3.1 跳过空白字符的办法：

- `scanf(" %c", &export_choice);` 前面的空格用于跳过任何残留的换行符或其他空白
- `while ((clean = getchar()) != '\n' && clean != EOF);`

3.2 void

很多情形下可以写void型函数，可以对参数进行检查，并且可以简化函数接口从而专注于操作而非结果。

进而可以用int型函数改写，通过返回0或1或其他值表示不同情况

3.3 想改变指针的值，要用二级指针

3.4 在子函数开内存？

正如前文所言，该程序只在导入时分配内存，结束时释放，很方便，但是可能也会导致一些问题，比如子函数开的内存如果忘记在调用后释放会造成泄露。

3.5 错误处理：

在涉及文件操作、内存分配等步骤时，检查返回值以确保操作成功完成。

例如，在 `fopen` 和 `malloc` 后检查返回值是否为 `NULL`。

3.6 测试和排错的技巧

- 在适当位置printf
- 设置断点
- 查看变量值
- 单步执行代码
- 把可能有问题的代码注释起来，找到问题所在
- 查报错信息

4.表格

	6.4 – 6.7	6.8 – 6.12	6.13 – 6.14	6.15 – 6.17
孙文	基本结构设计，导入导出函数	除线性方程和特征值的子菜及其功能函数	线性方程和幂法	报告

六、附录

函数全部代码

0. 导入导出

- 从txt导入
 - `Vector* import_vector_from_txt(char* filename);`

```
// 从txt文件导入：
Vector* import_vector_from_txt(char* filename) {
    FILE* file = fopen(filename, "r");
    //如果打开失败
    if (file == NULL) {
        printf( "无法打开文件 %s \n", filename);
        return NULL;
    }
    //统计维数
    int size = 0;
    char line[100];
    while (fgets(line, sizeof(line), file)) {
        size_t len = strlen(line);
        if (len > 0 && line[len - 1] == '\n') {
            line[len - 1] = '\0';
        } //去行尾换行符
    }
```

```

        if (line[0] == '\0') {
            continue;
        }
        size++;
    }
    if (size == 0) {
        fclose(file);
        return NULL;
    }
    Vector* vec = (Vector*)malloc(sizeof(Vector));
    if (!vec) {
        printf("为Vector分配内存失败");
        fclose(file);
        return NULL;
    }
    if (!init_vector(vec, size)) {
        free(vec);
        fclose(file);
        return NULL;
    }
    //从头开始读数据
    rewind(file);
    int i = 0;
    while (fgets(line, sizeof(line), file)) {
        size_t len = strlen(line);
        if (len > 0 && line[len - 1] == '\n') {
            line[len - 1] = '\0';
        } //去行尾换行符
        if (line[0] == '\0') {
            continue;
        }
        vec->data[i++] = strtod(line, NULL); //类型转换为double, 且不关心数
字外的其他字符
    }
    fclose(file);
    return vec;
}

```

o Matrix* import_matrix_from_txt(char* filename);

```

Matrix* import_matrix_from_txt(char* filename) {
    FILE* file = fopen(filename, "r");
    //如果打开失败
    if (file == NULL) {
        printf("无法打开文件 %s \n", filename);
        return NULL;
    }
    int cols = 0;
    int rows = 0;
    char line[1024];
    //读第一行求列数
    if (fgets(line, sizeof(line), file)) {
        char* token = strtok(line, "\n\t ");
        while (token != NULL) {
            cols++;
        }
    }
}

```

```

        token = strtok(NULL, "\n\t ");
    }
} else {
    fclose(file);
    return NULL;
}
//回开头求行数
rewind(file);
while (fgets(line, sizeof(line), file)) {
    rows++;
}
Matrix* mat = (Matrix*)malloc(sizeof(Matrix));
if (!mat) {
    fclose(file);
    printf("为Matrix内存分配失败");
    return NULL;
}

if (!init_matrix(mat, rows, cols)) {
    fclose(file);
    free(mat);
    return NULL;
}
//回开头读数据
rewind(file);
double* current_row = mat->data;

while (fgets(line, sizeof(line), file)) {
    char* token = strtok(line, "\n\t ");
    int col_i = 0;
    while (token != NULL && col_i < cols) {
        current_row[col_i++] = strtod(token, NULL);
        token = strtok(NULL, "\n\t ");
    }
    current_row += cols;
}

mat->is_init = 1;
return mat;
}

```

- 导出到txt

- `void export_vector_to_txt(Vector* vector, char* filename);`

```

void export_matrix_to_txt(Matrix* matrix, char* filename) {
    if (matrix == NULL || matrix->rows <= 0 || matrix->cols <= 0 ||
matrix->data == NULL || filename == NULL) {
        printf("提供参数有误");
        return;
    }
    FILE* file = fopen(filename, "w");
    //如果打开失败
    if (file == NULL) {
        printf("无法打开文件 %s \n", filename);
    }
}

```

```

        return;
    }
    //写入数据
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            fprintf(file, "%.21f ", matrix->data[i * matrix->cols + j]);
        }
        fprintf(file, "\n");
    }
    fclose(file);
    printf("矩阵数据成功导出至 %s\n", filename);
}

```

○ `void export_matrix_to_txt(Matrix* matrix, char* filename);`

```

void export_matrix_to_txt(Matrix* matrix, char* filename) {
    if (matrix == NULL || matrix->rows <= 0 || matrix->cols <= 0 ||
matrix->data == NULL || filename == NULL) {
        printf( "提供参数有误");
        return;
    }
    FILE* file = fopen(filename, "w");
    //如果打开失败
    if (file == NULL) {
        printf( "无法打开文件 %s \n", filename);
        return;
    }
    //写入数据
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            fprintf(file, "%.21f ", matrix->data[i * matrix->cols + j]);
        }
        fprintf(file, "\n");
    }
    fclose(file);
    printf("矩阵数据成功导出至 %s\n", filename);
}

```

1.单个向量操作

- 显示与切片
 - 显示向量


```
//打印vector
void print_vector(Vector* vector) {
    if (vector == NULL) {
        printf("向量指针为空! \n");
        return;
    }
    for (int i = 0; i < vector->size; i++) {
        printf("%.21f\n", vector->data[i]);
    }
}
```

- 显示向量元素

```
// 显示向量元素:
void print_vector_element(Vector* vector, int index) {
    if (vector == NULL) {
        printf("向量指针为空! \n");
        return;
    }
    if (index < 0 || index >= vector->size) {
        printf("索引 i 越界! \n");
        return;
    }
    printf("向量的第%d个元素是 %1f\n", index, vector->data[index]);
}
```

- 向量切片

```
// 向量的切片:
Vector* slice_vector(Vector* vector, int* indices, int indices_length) {
    if (vector == NULL || indices == NULL || indices_length <= 0) {
        return NULL;
    }
    Vector* slice_vector = (Vector*)malloc(sizeof(Vector));
    if (slice_vector == NULL) {
        return NULL;
    }
    slice_vector->data = (double*)malloc(indices_length *
sizeof(double));
    if (slice_vector->data == NULL) {
        free(slice_vector);
        return NULL;
    }
    for (int i = 0; i < indices_length; i++) {
        int index = indices[i];
        if (index >= 0 && index < vector->size) {
            slice_vector->data[i] = vector->data[index];
        } else {
            free(slice_vector->data);
            free(slice_vector);
            return NULL;
        }
    }
    slice_vector->size = indices_length;
    slice_vector->is_init = 1;
    return slice_vector;
}
```

```
}
```

- 初始化与变换

- 初始化列向量

```
Vector* init_column_vector(int size, double value) {  
    Vector* vec = (Vector*)malloc(sizeof(Vector));  
    if (!vec) {  
        printf("为Vector分配内存失败\n");  
        return NULL;  
    }  
    init_vector(vec, size);  
    for (int i = 0; i < size; i++) {  
        vec->data[i] = value;  
    }  
    return vec;  
}
```

- 向量绝对值

```
void abs_vector(Vector* vector) {  
    if (vector == NULL || vector->data == NULL) {  
        return;  
    }  
    for (int i = 0; i < vector->size; i++) {  
        vector->data[i] = fabs(vector->data[i]);  
    }  
}
```

- 向量元素取倒数

```
void reciprocal_elements_vector(Vector* vector) {  
    if (vector == NULL || vector->data == NULL) {  
        return;  
    }  
    for (int i = 0; i < vector->size; i++) {  
        double element = vector->data[i];  
        if (element != 0.0) {  
            vector->data[i] = 1.0 / element;  
        } else {  
            vector->data[i] = 0;  
        }  
    }  
}
```

- 向量元素平方

```
void square_elements_vector(Vector* vector) {
    if (vector == NULL || vector->data == NULL) {
        return;
    }
    for (int i = 0; i < vector->size; i++) {
        vector->data[i] *= vector->data[i];
    }
}
```

- 向量归一化

```
//向量归一化
void normalize_vector(Vector* vector) {
    double norm = norm_vector(vector, 2);
    if (norm != 0) {
        for (int i = 0; i < vector->size; i++) {
            vector->data[i] /= norm;
        }
        printf("向量已归一化\n");
    } else {
        printf("向量范数为0, 无法归一化\n");
    }
}
```

- 统计与排序

- 查找向量中的最大元素

```
double max_element_vector(Vector vector) {
    double max = vector.data[0];
    for (int i = 1; i < vector.size; i++) {
        if (max < vector.data[i]) {
            max = vector.data[i];
        }
    }
    return max;
}
```

- 查找向量中的最小元素

```
double min_element_vector(Vector vector) {
    double min = vector.data[0];
    for (int i = 1; i < vector.size; i++) {
        if (min > vector.data[i]) {
            min = vector.data[i];
        }
    }
    return min;
}
```

- 查找向量中最大的c1个元素的索引

```
void top_c1_max_indices(Vector vector, int c1, int* indices) {
    if (c1 <= 0 || c1 > vector.size) {
```

```

        printf("c1无效!\n");
        return;
    }
    index_valeur* i_v = (index_valeur*)malloc(vector.size *
sizeof(index_valeur));
    for (int i = 0; i < vector.size; i++) {
        i_v[i].index = i;
        i_v[i].valeur = vector.data[i];
    }
    bubbleSort_struct_index_valeur(i_v, vector.size);
    for (int i = 0; i < c1; i++) {
        indices[i] = i_v[vector.size - i - 1].index;
    }
    for (int i = 0; i < c1; i++) {
        printf("第%d大的元素位置: %d\n", i, indices[i]);
    }
}

```

- 查找向量中最小的c1个元素的索引

```

void top_c1_min_indices(Vector vector, int c1, int* indices) {
    if (c1 <= 0 || c1 > vector.size) {
        printf("c1无效!\n");
        return;
    }
    index_valeur* i_v = (index_valeur*)malloc(vector.size *
sizeof(index_valeur));
    for (int i = 0; i < vector.size; i++) {
        i_v[i].index = i;
        i_v[i].valeur = vector.data[i];
    }
    bubbleSort_struct_index_valeur(i_v, vector.size);
    for (int i = 0; i < c1; i++) {
        indices[i] = i_v[i].index;
    }
    for (int i = 0; i < c1; i++) {
        printf("第%d小的元素位置: %d\n", i, indices[i]);
    }
}

```

- 从大到小排序向量

```

//从大到小排序向量
void sort_vector_desc(Vector* vector) {
    elbbubSort(vector->data, vector->size);
}

```

- 从小到大排序向量

```

//从小到大排序向量
void sort_vector_asc(Vector* vector) {
    bubbleSort(vector->data, vector->size);
}

```

- 向量范数

```

//向量范数
double norm_vector(Vector* vector, int option) {
    if (vector == NULL || vector->data == NULL) {
        return -1;
    }
    switch (option) {
        case 1: {
            double sum_1 = 0;
            for (int i = 0; i < vector->size; i++) {
                sum_1 += fabs(vector->data[i]);
            }
            return sum_1;
        }
        case 2: {
            double sum_2 = vector_norm(vector->data, vector->size);
            return sum_2;
        }
        case 3:
            double max = max_element_vector(*vector);
            return max;
        default:
            return -1;
    }
}

```

- 算术操作

- 向量除以标量

```

//向量除以标量
void divide_vector_by_scalar(Vector * vector, double scalar) {
    if (vector == NULL || vector->data == NULL) {
        return;
    }
    if (scalar == 0.0) {
        printf("%1f为0! ", scalar);
        return;
    }
    for (int i = 0; i < vector->size; i++) {
        vector->data[i] /= scalar;
    }
}

```

- 向量乘以标量

```

//向量乘以标量
void multiply_vector_by_scalar(Vector * vector, double scalar) {
    for (int i = 0; i < vector->size; i++) {
        vector->data[i] *= scalar;
    }
}

```

2. 单个矩阵操作

- 显示与切片

- 显示矩阵

```
//打印matrix
void print_matrix(Matrix* mat) {
    for (int i = 0; i < mat->rows; i++) {
        for (int j = 0; j < mat->cols; j++) {
            printf(" %.21f ", mat->data[i * mat->cols + j]);
        }
        printf("\n");
    }
}
```

- 显示矩阵元素

```
void print_matrix_element(Matrix* matrix, int row, int col) {
    printf("请输入要显示的行和列（例如：0 1）：");
    scanf("%d %d", &row, &col);
    if(row >= 0 && row < matrix->rows && col >= 0 && col < matrix->cols){
        printf("第%d行第%d列的元素是： %.21f\n", row, col, matrix->data[row
* matrix->cols + col]);
    }else{
        printf("索引越界！\n");
    }
}
```

- 矩阵行切片

```
double* slice_matrix_row(Matrix* matrix, int row) {
    if (row >= 0 && row < matrix->rows) {
        double* slice_matrix_row = (double*)malloc(matrix->cols *
sizeof(double));
        for (int i = 0; i < matrix->cols; i++) {
            slice_matrix_row[i] = matrix->data[row * matrix->cols + i];
        }
        return slice_matrix_row;
    } else {
        printf("行索引越界！\n");
        return NULL;
    }
}
```

- 矩阵列切片

```
double* slice_matrix_col(Matrix* matrix, int col) {
    if (col >= 0 && col < matrix->cols) {
        double* slice_matrix_col = (double*)malloc(matrix->rows *
sizeof(double));
        for (int i = 0; i < matrix->rows; i++) {
            slice_matrix_col[i] = matrix->data[i * matrix->cols + col];
        }
        return slice_matrix_col;
    } else {
        printf("列索引越界! \n");
        return NULL;
    }
}
```

- 矩阵根据位置矩阵切片

```
Matrix* slice_matrix_m(Matrix* matrix, int row, int col, int *
indices_col, int * indices_row) {
    Matrix * slice_matrix = (Matrix*)malloc(sizeof(Matrix)) ;
    if (!init_matrix(slice_matrix, row, col)) {
        printf("矩阵切片初始化失败! \n");
    }

    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            slice_matrix->data[i * col + j] = matrix-
>data[indices_row[i] * matrix->cols + indices_col[j] ];
        }
    }
    return slice_matrix;
}
```

- 变换

- 矩阵元素取倒数

```
void reciprocal_elements_matrix(Matrix* matrix) {
    if (matrix == NULL || matrix->data == NULL) {
        return;
    }
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            double element = matrix->data[i * matrix->cols + j];
            if (element != 0.0) { // 避免除以零
                matrix->data[i * matrix->cols + j] = 1.0 / element;
            }
        }
    }
}
```

- 矩阵元素取平方

```

void square_elements_matrix(Matrix* matrix) {
    if (matrix == NULL || matrix->data == NULL) {
        return;
    }
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            matrix->data[i * matrix->cols + j] *= matrix->data[i *
matrix->cols + j];
        }
    }
}

```

- 替换矩阵的指定列

```

void replace_column(Matrix* matrix, double* new_column, int col_index) {
    if (matrix == NULL || matrix->data == NULL || new_column == NULL ||
col_index < 0 || col_index >= matrix->cols) {
        return;
    }
    for (int i = 0; i < matrix->rows; i++) {
        matrix->data[i * matrix->cols + col_index] = new_column[i];
    }
}

```

- 矩阵取转置

```

void transpose_matrix(Matrix* matrix) {
    if (matrix == NULL || matrix->data == NULL) {
        return;
    }
    int new_rows = matrix->cols;
    int new_cols = matrix->rows;
    double* new_data = (double*)malloc(sizeof(double) * new_rows *
new_cols);
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            new_data[j * new_rows + i] = matrix->data[i * matrix->cols +
j];
        }
    }
    free(matrix->data);
    matrix->data = new_data;
    matrix->rows = new_rows;
    matrix->cols = new_cols;
}

```

- 统计

- 矩阵所有元素求和


```
double sum_of_matrix(Matrix* matrix) {
    if (matrix == NULL || matrix->data == NULL) {
        return 0.0;
    }
    double sum = 0.0;
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            sum += matrix->data[i * matrix->cols + j];
        }
    }
    return sum;
}
```

- 查找矩阵中的最大元素

```
void max_elements_matrix(Matrix* matrix) {
    if (matrix == NULL || matrix->data == NULL) {
        return;
    }
    double max = matrix->data[0];
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            if (matrix->data[i * matrix->cols + j] > max) {
                max = matrix->data[i * matrix->cols + j];
            }
        }
    }
    printf("矩阵中的最大元素是: %.21f\n", max);
}
```

- 每行的最大元素

```
//找出每行的最大元素
void max_elements_per_row(Matrix* matrix, double* max_values) {
    if (matrix == NULL || matrix->data == NULL || max_values == NULL) {
        return;
    }
    for (int i = 0; i < matrix->rows; i++) {
        double max = matrix->data[i * matrix->cols];
        for (int j = 1; j < matrix->cols; j++) {
            if (matrix->data[i * matrix->cols + j] > max) {
                max = matrix->data[i * matrix->cols + j];
            }
        }
        max_values[i] = max;
    }
}
```

- 每行的最小元素

```
//找出每行的最小元素
void min_elements_per_row(Matrix* matrix, double* min_values) {
    if (matrix == NULL || matrix->data == NULL || min_values == NULL) {
        return;
    }
}
```

```

    }
    for (int i = 0; i < matrix->rows; i++) {
        double min = matrix->data[i * matrix->cols];
        for (int j = 1; j < matrix->cols; j++) {
            if (matrix->data[i * matrix->cols + j] < min) {
                min = matrix->data[i * matrix->cols + j];
            }
        }
        min_values[i] = min;
    }
}

```

- 整行元素都为0的行编号

```

//找整行元素都为0的行编号
void find_zero_row_indices(Matrix* matrix, int* zero_row_indices, int*
max_indices) {
    if (matrix == NULL || matrix->data == NULL || zero_row_indices ==
NULL) {
        return;
    }
    int zero_row_count = 0;
    for (int i = 0; i < matrix->rows && zero_row_count < matrix->rows;
i++) {
        int is_zero_row = 1;
        for (int j = 0; j < matrix->cols; j++) {
            if (matrix->data[i * matrix->cols + j] != 0.0) {
                is_zero_row = 0;
                break;
            }
        }
        if (is_zero_row) {
            zero_row_indices[zero_row_count++] = i;
        }
    }
    *max_indices = zero_row_count;
}

```

- 矩阵范数及特征值

```

void norm_matrix(Matrix* matrix, int option) {
    if (matrix == NULL || matrix->data == NULL) {
        return ;
    }
    double result;
    double sum;
    //特征值用
    double lambda; // 存储计算出的特征值
    double x[matrix->cols]; // 存储计算出的特征向量
    double epsilon = 1e-6; // 收敛阈值
    int N = 1000; // 最大迭代次数
    switch (option) {
        case 1:
            for (int j = 0; j < matrix->cols; ++j) {
                sum = 0.0;
                for (int i = 0; i < matrix->rows; ++i) {

```

```

        sum += fabs(matrix->data[i * matrix->cols + j]);
    }
    if (sum > result) {
        result = sum;
    }
}
printf("1范数为: %f\n", result);

return;
case 2:
    Matrix temp = *matrix;
    transpose_matrix(matrix);
    Matrix t = multiply_AB(matrix, &temp);
    if (power_method(&t, &lambda, x, epsilon, N)) {
        printf("2范数为: %f\n", sqrt(lambda));
    }
    return;
case 3:
    for (int i = 0; i < matrix->rows; ++i) {
        sum = 0.0;
        for (int j = 0; j < matrix->cols; ++j) {
            sum += fabs(matrix->data[i * matrix->cols + j]);
        }
        if (sum > result) {
            result = sum;
        }
    }
    printf("无穷范数为: %f\n", result);
    return;
case 4:
    if (power_method(matrix, &lambda, x, epsilon, N)) {
        printf("主特征值为: %f\n", lambda);
        printf("对应特征向量为: \n");
        for (int i = 0; i < matrix->cols; i++) {
            printf("%f\n", x[i]);
        }
    } else {
        printf("未求得特征值! \n");
    }
    return ;
default:
    printf("无效的选项! \n");
    return;
}
}

```

- 矩阵列向量间的夹角Cos值

```

double* calculate_all_cos(Matrix * matrix) {
    if (matrix == NULL || matrix->data == NULL ) {
        return NULL;
    }
    if ( matrix->cols < 2) {
        printf("矩阵列数小于2! ");
        return NULL;
    }
}

```

```

int num_all_cos = matrix->cols * (matrix->cols - 1) / 2; // 组合数
C(n, 2)
double* all_cos = (double*)malloc(num_all_cos * sizeof(double));
if (all_cos == NULL) {
    return NULL;
}
int index = 0;
for (int i = 0; i < matrix->cols - 1; i++) {
    for (int j = i + 1; j < matrix->cols; j++) {
        double* vec1 = slice_matrix_col(matrix, i);
        double* vec2 = slice_matrix_col(matrix, j);
        all_cos[index++] = calculate_cos(vec1, vec2, matrix->rows);
    }
}
return all_cos;
}

```

- 算术操作

- 矩阵乘以标量

```

void multiply_matrix_by_scalar(Matrix * matrix, double scalar) {
    if (matrix == NULL || matrix->data == NULL) {
        return;
    }
    for (int i = 0; i < matrix->rows; i++) {
        for (int j = 0; j < matrix->cols; j++) {
            matrix->data[i * matrix->cols + j] *= scalar;
        }
    }
}

```

- 特殊矩阵

- 生成随机对称矩阵

```

Matrix* generate_random_symmetric_matrix(int rows) {
    Matrix* m = (Matrix*)malloc(sizeof(Matrix));
    if (!m) {
        printf("内存分配失败");
        return NULL;
    }
    init_matrix(m, rows, rows);
    srand(time(NULL));
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < rows; j++) {
            if (j <= i) { // 只处理上三角和对角线
                double val = (rand() % 201 - 100) / 100.0; // [-1, 1]
                m->data[i * rows + j] = val;
                if (i != j) {
                    m->data[j * rows + i] = val;
                }
            }
        }
    }
    return m;
}

```

- 特征值求解

- 幂法

```
int power_method(Matrix* A, double* lambda, double* x, double epsilon,
int N) {
    if (A->cols != A->rows) {
        printf("A不是方阵! ");
        return 0;
    }

    int n = A->cols;
    Vector v, w;
    init_vector(&v, n); // 初始化迭代向量v
    for (int i = 0; i < n; i++) {
        v.data[i] = 1.0; // 将v的所有元素初始化为1
    }

    double m; // 保存向量大小, 用于规范化
    for (int k = 0; k < N - 1; k++) {
        w = multiply_matrix_by_vector(A, &v); // 计算Av
        m = vector_norm(w.data, w.size); // 计算向量范数

        if (m == 0) {
            printf("向量的范数为零, 无法继续进行迭代。\\n");
            free_vector(&v);
            free_vector(&w);
            return 0;
        }

        // 规范化向量w
        for (int i = 0; i < n; i++) {
            w.data[i] /= m;
        }

        double temp = fabs(vector_norm(subtract_vectors(v, w).data, n));
        // 计算v和w的差的范数
        if (temp <= epsilon) { // 判断是否收敛
            *lambda = m; // 设置特征值
            for (int i = 0; i < n; i++) {
                x[i] = w.data[i]; // 设置特征向量
            }
            free_vector(&v);
            free_vector(&w);
            return 1; // 收敛, 返回1
        }

        v = w; // 为下一次迭代准备, 将w赋值给v
    }

    // 如果达到N未收敛
    printf("达到最大迭代次数, 算法未收敛。\\n");
    free_vector(&v);
    free_vector(&w);
    return 0; // 未收敛, 返回0
}
```

3. 向量与向量操作

- 向量加法

```
Vector add_vectors(Vector v1, Vector v2) {
    Vector result;
    if (v1.size != v2.size) {
        printf("向量大小不匹配! \n");
        init_vector(&result, 0);
        return result;
    }
    init_vector(&result, v1.size);
    for (int i = 0; i < v1.size; i++) {
        result.data[i] = v1.data[i] + v2.data[i];
    }
    return result;
}
```

- 向量减法

```
// 向量减法
Vector subtract_vectors(Vector v1, Vector v2) {
    Vector result;
    if (v1.size != v2.size) {
        printf("向量大小不匹配! \n");
        init_vector(&result, 0);
        return result;
    }
    init_vector(&result, v1.size);
    for (int i = 0; i < v1.size; i++) {
        result.data[i] = v1.data[i] - v2.data[i];
    }
    return result;
}
```

- 向量按元素除法

```
// 向量按元素除法
Vector elementwise_divide(Vector v1, Vector v2) {
    Vector result;
    if (v1.size != v2.size) {
        printf("向量大小不匹配! \n");
        init_vector(&result, 0);
        return result;
    }
    init_vector(&result, v1.size);
    for (int i = 0; i < v1.size; i++) {
        if (v2.data[i] != 0.0) {
            result.data[i] = v1.data[i] / v2.data[i];
        } else {
            printf("除数不能为零! \n");
            result.data[i] = 0.0;
        }
    }
}
```

```
    return result;
}
```

- 向量按元素乘法

```
// 向量按元素乘法
Vector elementwise_multiply(Vector v1, Vector v2) {
    Vector result;
    if (v1.size != v2.size) {
        printf("向量大小不匹配! \n");
        init_vector(&result, 0);
        return result;
    }
    init_vector(&result, v1.size);
    for (int i = 0; i < v1.size; i++) {
        result.data[i] = v1.data[i] * v2.data[i];
    }
    return result;
}
```

- 余弦

```
// 余弦
void cos_v1_v2(Vector v1, Vector v2) {
    if (v1.size != v2.size) {
        printf("向量大小不匹配! \n");
        return ;
    }
    double cos = calculate_cos(v1.data, v2.data, v1.size);
    printf("向量1向量2夹角余弦为: %1f\n", cos);
}
```

- 点积

```
// 点积
void dot_product_vectors(Vector* v1, Vector* v2) {
    if (v1->size != v2->size) {
        printf("向量大小不匹配! \n");
        return ;
    }
    double dot=dot_product(v1->data, v2->data, v1->size);
    printf("向量1向量2点积为: %1f\n", dot);
}
```

- 向量外积

```
// 向量外积
Matrix vector_outer_product(Vector* vec1, Vector* vec2) {
    Matrix result;
    init_matrix(&result, vec1->size, vec2->size);
    for (int i = 0; i < vec1->size; i++) {
        for (int j = 0; j < vec2->size; j++) {
            result.data[i * vec2->size + j] = vec1->data[i] * vec2->data[j];
        }
    }
    return result;
}
```

4.向量与矩阵操作

- 向量左乘矩阵

- ```
// 向量左乘矩阵
Vector multiply_matrix_by_vector(Matrix* matrix, Vector* vector) {
 Vector result;
 if (matrix->cols != vector->size) {
 printf("矩阵的列数必须与向量的大小不同! \n");
 init_vector(&result, 0);
 return result;
 }
 init_vector(&result, matrix->rows);
 for (int i = 0; i < matrix->rows; i++) {
 double sum = 0.0;
 for (int j = 0; j < matrix->cols; j++) {
 sum += matrix->data[i * matrix->cols + j] * vector->data[j];
 }
 result.data[i] = sum;
 }
 return result;
}
```

- 向量右乘矩阵

- ```
// 向量右乘矩阵
Vector multiply_vector_by_matrix(Vector* vector, Matrix* matrix) {
    Vector result;
    if (vector->size != matrix->rows) {
        printf("向量的大小与矩阵的行数不同! \n");
        init_vector(&result, 0 );
        return result;
    }
    init_vector(&result, matrix->cols);
    for (int i = 0; i < matrix->cols; i++) {
        double sum = 0.0;
        for (int j = 0; j < matrix->rows; j++) {
            sum += matrix->data[i * matrix->cols + j] * vector->data[j];
        }
        result.data[i] = sum;
    }
    return result;
}
```


5. 矩阵与矩阵操作

- 矩阵加法

```
Matrix add_matrices(Matrix* mat1, Matrix* mat2) {
    Matrix result;
    if (mat1->rows == mat2->rows && mat1->cols == mat2->cols) {
        init_matrix(&result, mat1->rows, mat1->cols);
        for (int i = 0; i < mat1->rows; i++) {
            for (int j = 0; j < mat1->cols; j++) {
                result.data[i * mat1->cols + j] = mat1->data[i * mat1->cols + j] + mat2->data[i * mat1->cols + j];
            }
        }
    } else {
        init_matrix(&result, 0, 0);
        printf("矩阵大小不同! ");
    }
    return result;
}
```

- 矩阵减法

```
Matrix subtract_matrices(Matrix* mat1, Matrix* mat2) {
    Matrix result;
    if (mat1->rows == mat2->rows && mat1->cols == mat2->cols) {
        init_matrix(&result, mat1->rows, mat1->cols);
        for (int i = 0; i < mat1->rows; i++) {
            for (int j = 0; j < mat1->cols; j++) {
                result.data[i * mat1->cols + j] = mat1->data[i * mat1->cols + j] - mat2->data[i * mat1->cols + j];
            }
        }
    } else {
        init_matrix(&result, 0, 0);
        printf("矩阵大小不同! ");
    }
    return result;
}
```

- AB型矩阵相乘

```
Matrix multiply_AB(Matrix* A, Matrix* B) {
    Matrix result;
    if (A->cols != B->rows) {
        printf("这两个矩阵不可AB乘");
        init_matrix(&result, 0, 0);
        return result;
    }
    init_matrix(&result, A->rows, B->cols);
    for (int i = 0; i < result.rows; i++) {
        for (int j=0; j < result.cols; j++) {
```

```

        double* A_rowi = slice_matrix_row(A, i);
        double* B_colj = slice_matrix_col(B, j);
        result.data[i * result.cols + j] = dot_product(A_rowi,
B_colj, A->cols);

        free(A_rowi);
        free(B_colj);
    }
}

return result;
}

```

- ATBA型矩阵相乘)

```

    Matrix multiply_ATBA(Matrix* A, Matrix* B) {
        Matrix result, temp;
        if (B->cols != A->rows || B->cols != B->rows) {
            printf("这两个矩阵不可ATBA乘\n");
            init_matrix(&result, 0, 0);
            return result;
        }
        init_matrix(&temp, A->rows, B->cols);
        init_matrix(&result, A->cols, A->cols);
        temp = multiply_AB(B, A);
        transpose_matrix(A);
        result = multiply_AB(A, B);
        free_matrix(&temp);
        return result;
    }

```

- 矩阵按元素乘法

```

    Matrix elementwise_multiply_matrices(Matrix* mat1, Matrix* mat2) {
        Matrix result;
        if (mat1->rows == mat2->rows && mat1->cols == mat2->cols) {
            init_matrix(&result, mat1->rows, mat1->cols);
            for (int i = 0; i < mat1->rows; i++) {
                for (int j = 0; j < mat1->cols; j++) {
                    result.data[i * mat1->cols + j] = mat1->data[i * mat1->cols + j] * mat2->data[i * mat1->cols + j];
                }
            }
        } else {
            init_matrix(&result, 0, 0);
            printf("矩阵大小不同! ");
        }
        return result;
    }

```

- 矩阵按元素除法

```

○ Matrix elementwise_divide_matrices(Matrix* mat1, Matrix* mat2) {
    Matrix result;
    if (mat1->rows == mat2->rows && mat1->cols == mat2->cols) {
        init_matrix(&result, mat1->rows, mat1->cols);
        for (int i = 0; i < mat1->rows; i++) {
            for (int j = 0; j < mat1->cols; j++) {
                if (mat2->data[i * mat1->cols + j] != 0) {
                    result.data[i * mat1->cols + j] = mat1->data[i *
mat1->cols + j] / mat2->data[i * mat1->cols + j];
                } else {
                    result.data[i * mat1->cols + j] = 0;
                }
            }
        }
    } else {
        init_matrix(&result, 0, 0);
        printf("矩阵大小不同!");
    }
    return result;
}

```

6. 线性方程

- 上三角矩阵的逆

```

Matrix* invert_upper_triangular_matrix(Matrix m) {
    if (m.rows != m.cols) {
        printf("不是方阵，无法求逆!\n");
        return NULL;
    }

    Matrix* inv = (Matrix*)malloc(sizeof(Matrix));
    if (inv == NULL) {
        printf("内存分配失败!\n");
        return NULL;
    }
    init_matrix(inv, m.rows, m.cols);

    // 从上向下
    for (int i = 0; i < m.rows; i++) {
        // 对角线元素直接取倒数
        inv->data[i * m.cols + i] = 1.0 / m.data[i * m.cols + i];
    }

    // 从最后一行开始，逐行向上求逆矩阵的非对角线元素
    for (int i = m.rows - 1; i >= 0; i--) {
        for (int j = i + 1; j < m.rows; j++) {
            double sum = 0.0;
            for (int k = i; k < j; k++) {
                sum += inv->data[i * m.cols + k] * m.data[k * m.cols + j];
            }
            inv->data[i * m.cols + j] -= sum * inv->data[j * m.cols + j];
        }
    }
    return inv;
}

```

```
}
```

- 上三角矩阵方程求解

```
void solve_upper_triangular_equation(Matrix* A, Vector* b, Vector* x) {
    // 从最后一个方程开始解
    for (int i = A->rows - 1; i >= 0; --i) {
        // 初始化x=b
        x->data[i] = b->data[i];
        // 减去已经解出的x乘以对应的系数
        for (int j = i + 1; j < A->rows; ++j) {
            x->data[i] -= A->data[i * A->cols + j] * x->data[j];
        }

        // 除以对角线上的系数，得到x
        if (A->data[i * A->cols + i] == 0) {
            printf("在第 %d 行有0元，方程不可解\n", i);
            return;
        }
        x->data[i] /= A->data[i * A->cols + i];
    }
}
```

- 一般线性方程求解（高斯消去法）

```
void solve_linear_equations(Matrix* A, Vector* b, Vector* x) {
    int n = A->rows;
    // 高斯消元
    for (int i = 0; i < n; ++i) {
        // 寻找当前列的主元（绝对值最大的元素）
        double pivot = fabs(A->data[i * n + i]); // 初始假设对角线上的元素为主元
        int pivot_row = i; // 主元所在的行
        for (int k = i + 1; k < n; ++k) {
            if (fabs(A->data[k * n + i]) > pivot) {
                pivot = fabs(A->data[k * n + i]);
                pivot_row = k;
            }
        }

        // 如果找到了更合适的主元，则交换对应的行
        if (pivot_row != i) {
            for (int j = i; j < n; ++j) {
                double tmp = A->data[i * n + j];
                A->data[i * n + j] = A->data[pivot_row * n + j];
                A->data[pivot_row * n + j] = tmp;
            }
            double tmp_b = b->data[i];
            b->data[i] = b->data[pivot_row];
            b->data[pivot_row] = tmp_b;
        }

        // 对下方的行进行消元，使得下方行的当前列元素变为0
    }
}
```

```

        for (int k = i + 1; k < n; ++k) {
            double factor = A->data[k * n + i] / A->data[i * n + i]; // 消元
            for (int j = i; j < n; ++j) {
                A->data[k * n + j] -= factor * A->data[i * n + j]; // 更新下方
            }
            b->data[k] -= factor * b->data[i]; // 更新右侧向量的对应元素
        }
    }

    //上三角
    solve_upper_triangular_equation(A, b, x);
}

```

7.辅助函数

- 向量

- 初始化

```

int init_vector(Vector* vec, int size) {
    vec->data = (double*) malloc(size * (sizeof(double)));
    if (!vec->data) {
        perror("Memory allocation failed");
        return 0;
    }
    for (int i = 0; i < size; i++) {
        vec->data[i] = 0;
    }
    vec->size = size;
    vec->is_init = 1;
    return 1;
}

```

- 释放

```

void free_vector(Vector* vec) {
    if (vec->is_init) {
        free(vec->data);
        vec->data = NULL;
        vec->size = 0;
        vec->is_init = 0;
    }
}

```

- 导入两个

```

void import_vector_2(Vector** vec1, Vector** vec2) {

    while (1) {
        // 导入第一个向量
        while (*vec1 == NULL || (*vec1)->is_init == 0) {

```

```

        printf("请导入第一个向量, 输入向量1所在文件名: ");
        scanf("%49s", filename);
        *vec1 = import_vector_from_txt(filename);
        if (*vec1 != NULL && (*vec1)->is_init == 1) {
            printf("向量1导入成功! \n");
        } else {
            printf("向量1导入失败! \n");
            printf("检查文件名是否正确或文件是否为空\n");
        }
    }

    // 导入第二个向量
    Vector* tempVec2 = NULL;
    while (1) {
        printf("请导入第二个向量, 输入向量2所在文件名: ");
        scanf("%49s", filename);
        tempVec2 = import_vector_from_txt(filename);
        if (tempVec2 != NULL && tempVec2->is_init == 1) {
            if ((*vec1)->size == tempVec2->size) {
                printf("向量2导入成功! \n");
                break; // 大小匹配, 退出循环
            } else {
                printf("两个向量大小不同, 请重新导入\n");
                // 释放不匹配的向量内存
                free_vector(tempVec2);
            }
        } else {
            printf("向量2导入失败! \n");
            printf("检查文件名是否正确或文件是否为空\n");
        }
    }
    *vec2 = tempVec2;
    printf("向量1向量2导入成功! \n");
    break;
}
}

```

- 是否导出

```

void if_export_vec_result(Vector* vec, char* filename) {
    printf("是否要导出结果到文件? (Y/N): ");
    char export_choice;
    scanf(" %c", &export_choice);
    if (export_choice == 'Y' || export_choice == 'y') {
        printf("请输入导出文件名: ");
        scanf("%49s", filename);
        export_vector_to_txt(vec, filename);
    }
}

```

- 向量模长

```
double vector_norm(double* vec, int size) {
    double sum_of_squares = 0.0;
    for (int i = 0; i < size; i++) {
        sum_of_squares += vec[i] * vec[i];
    }
    return sqrt(sum_of_squares);
}
```

- 向量点积

```
double dot_product(double* vec1, double* vec2, int size) {
    double result = 0.0;
    for (int i = 0; i < size; i++) {
        result += vec1[i] * vec2[i];
    }
    return result;
}
```

- 向量cos

```
double calculate_cos(double* vec1, double* vec2, int size) {
    double dot = dot_product(vec1, vec2, size);
    double norm1 = vector_norm(vec1, size);
    double norm2 = vector_norm(vec2, size);
    return dot / (norm1 * norm2);
}
```

• 7.2 矩阵

- 初始化

```
int init_matrix(Matrix* mat, int rows, int cols) {
    mat->data = (double*) malloc(rows * cols * (sizeof(double)));
    if (!mat->data) {
        printf("内存分配失败");
        return 0;
    }
    if (rows > 0 && cols > 0) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                mat->data[i * cols + j] = 0;
            }
        }
        mat->is_init = 1;
    } else {
        mat->data = NULL;
        mat->is_init = 0;
    }
    mat->rows = rows;
    mat->cols = cols;

    return 1;
}
```

- 释放

```

void free_matrix(Matrix* mat) {
    if (mat->is_init) {
        free(mat->data);
        mat->data = NULL;
        mat->rows = 0;
        mat->cols = 0;
        mat->is_init = 0;
    }
}

```

- 是否导出

```

void if_export_matrix_result(Matrix* mat, char* filename) {
    printf("是否要导出结果到文件? (Y/N): ");
    char export_choice;
    scanf(" %c", &export_choice);
    if (export_choice == 'Y' || export_choice == 'y') {
        printf("请输入导出文件名: ");
        scanf("%49s", filename);
        export_matrix_to_txt(mat, filename);
    }
}

```

• 7.3 方程

- 导入

```

void import_linear_equation(Vector** vec, Matrix** mat) {
    while (*mat == NULL || (*mat)->is_init == 0) {
        printf("请导入矩阵, 输入矩阵所在文件名: ");
        scanf("%49s", filename);
        *mat = import_matrix_from_txt(filename);
        if (*mat != NULL && (*mat)->is_init == 1) {
            printf("矩阵导入成功! \n");
        } else {
            printf("矩阵导入失败! \n");
            printf("请检查文件名是否正确或文件是否为空\n");
            if (*mat != NULL) {
                free_matrix(*mat);
                *mat = NULL;
            }
        }
    }
    while (*vec == NULL || (*vec)->is_init == 0) {
        printf("请导入向量, 输入向量所在文件名: ");
        scanf("%49s", filename);
        *vec = import_vector_from_txt(filename);
        if (*vec != NULL && (*vec)->is_init == 1) {
            printf("向量导入成功! \n");
        } else {
            printf("向量导入失败! \n");
            printf("请检查文件名是否正确或文件是否为空\n");
            if (*vec != NULL) {
                free_vector(*vec);
            }
        }
    }
}

```



```

        *vec = NULL;
    }
}
}
}

```

• 7.4 其他

◦ 排序

■ 冒泡

```

void bubbleSort(double arr[], int n) {
    int i, j;
    double temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void elbbubSort(double arr[], int n) {
    int i, j;
    double temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] < arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

```

■ 查找向量中最大的c1个元素的索引用

```

typedef struct {
    double vaule;
    int index;
} index_vaule;

void bubbleSort_struct_index_vaule(index_vaule* arr, int n) {
    int i, j;
    index_vaule temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j].vaule > arr[j + 1].vaule) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

```

```

    }
}
}

```

```

//c1索引导出
void export_indices_to_txt(int* indices, int c1, char* filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("无法打开文件 %s\n", filename);
        return;
    }
    for (int i = 0; i < c1; i++) {
        fprintf(file, "%d\n", indices[i]);
    }
    fclose(file);
    printf("索引已成功导出到 %s\n", filename);
}

```

8.主函数及子菜单

- 主函数

```

int main() {
    int choice;
    do {
        main_menu();
        printf("请输入您的选择（例如：1）：");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                submenu_vector_operations();
                break;
            case 2:
                submenu_matrix_operations();
                break;
            case 3:
                submenu_vector_vector_operations();
                break;
            case 4:
                submenu_vector_matrix_operations();
                break;
            case 5:
                submenu_matrix_matrix_operations();
                break;
            case 6:
                submenu_linear_equation();
                break;
            case 0:
                printf("退出程序...\n");
                break;
            default:
                printf("无效的选项! \n");
                break;
        }
    } while (choice != 0);
}

```

```
    return 0;
}
```

- 主菜单

```
void main_menu() {
    printf("\n***** 主菜单 *****\n");
    printf("    1. 单个向量操作\n");
    printf("    2. 单个矩阵操作\n");
    printf("    3. 向量与向量操作\n");
    printf("    4. 向量与矩阵操作\n");
    printf("    5. 矩阵与矩阵操作\n");
    printf("    6. 线性方程\n");
    printf("    0. 退出\n");
    printf("*****\n");
}
```

- 子菜单

- 单个向量操作

```
void submenu_vector_operations() {
    show_submenu_vector_operations();
    char choice;
    Vector* vec = NULL;
    while (vec == NULL || vec->is_init == 0) {
        printf("请先导入向量，输入向量所在文件名: ");
        scanf("%49s", filename);
        vec = import_vector_from_txt(filename);
        if (vec != NULL && vec->is_init == 1) {
            printf("向量导入成功! \n");
        } else {
            printf("向量导入失败! \n");
            printf("检查文件名是否正确或文件是否为空\n");
        }
    }

    do {
        printf("请输入您的选择（例如: A ）: ");
        scanf(" %c", &choice);

        switch (choice) {
            case '1':
                printf("1. 导入导出\n");
                printf("1.1. 导入向量数据 (A)\n");
                printf("1.2. 导出向量数据 (B)\n");
                break;
            case 'A': // 导入向量数据
                if (vec != NULL && vec->is_init == 1) {
                    printf("向量已导入，是否导入一个新向量覆盖当前向量? (y/n): ");

                    char overwrite;
                    scanf(" %c", &overwrite);

                    if (overwrite == 'y' || overwrite == 'Y') {
                        free_vector(vec);

```

```

        // 重新导入向量
        printf("请输入新向量所在文件名: ");
        scanf("%49s", filename);
        vec = import_vector_from_txt(filename);
        if (vec == NULL || vec->is_init == 0) {
            printf("向量导入失败! \n");
        } else {
            printf("新向量导入成功! \n");
        }
    } else {
        printf("保留当前向量\n");
    }
} else {
    printf("错误: 向量未初始化 \n");
}
break;
case 'B': // 导出向量数据
    if (vec != NULL && vec->is_init == 1) {
        printf("请输入导出向量所至文件名: ");
        scanf("%49s", filename);
        export_vector_to_txt(vec, filename);
    } else {
        printf("没有向量可导出! \n");
    }
    break;
case '2':
    printf("2. 显示与切片\n");
    printf("2.1. 显示向量 (C)\n");
    printf("2.2. 显示向量元素 (D)\n");
    printf("2.3. 向量切片 (E)\n");
    break;
case 'C': // 显示向量
    printf("当前向量:\n ");
    print_vector(vec);
    break;
case 'D': // 显示向量第i个元素
    int index;
    printf("查看向量第i个元素, 输入i: ");
    scanf("%d", &index);
    print_vector_element(vec, index);
    break;
case 'E': // 做向量的切片
    int indices_length;
    printf("输入切片长度: ");
    scanf("%d", &indices_length);
    if (indices_length <= 0) {
        printf("退出向量切片\n");
        break;
    }
    int* indices = (int*)malloc(indices_length *
sizeof(int));
    printf("输入切片索引: ");
    for (int i = 0; i < indices_length; i++) {
        scanf("%d", &indices[i]);
    }
    Vector* sliced_vec = slice_vector(vec, indices,
indices_length);
    if (sliced_vec != NULL && sliced_vec->is_init == 1) {

```

```

        printf("向量切片成功\n ");
        print_vector(sliced_vec);
        printf("是否导出? (y/n): ");
        char export_slice;
        scanf(" %c", &export_slice);
        if (export_slice == 'y' || export_slice == 'Y') {
            if (sliced_vec != NULL && sliced_vec->is_init
== 1) {
                export_vector_to_txt(sliced_vec,
"sliced_vec.txt");
                printf("切片向量导出至: sliced_vec.txt\n");
            }
        }
    } else {
        printf("向量切片失败! \n");
    }
    free_vector(sliced_vec);
    free(indices);
    break;
case '3':
    printf("3. 初始化与变换\n");
    printf("3.1. 初始化列向量 (F)\n");
    printf("3.2. 向量绝对值 (G)\n");
    printf("3.3. 向量元素取倒数 (H)\n");
    printf("3.4. 向量元素平方 (I)\n");
    printf("3.5. 向量归一化 (J)\n");
    break;
case 'F': // 初始化列向量
    int size;
    double value;
    printf("请输入列向量的大小: ");
    scanf("%d", &size);
    printf("请输入要初始化的值: ");
    scanf("%lf", &value);
    Vector* new_vec = init_column_vector(size, value);
    if (new_vec != NULL && new_vec->is_init == 1) {
        printf("列向量初始化成功! \n");
        print_vector(new_vec);
        printf("是否要将列向量导出到文件? (y/n): ");
        char export_choice;
        scanf(" %c", &export_choice);
        if (export_choice == 'y' || export_choice == 'Y') {
            char filename[50];
            printf("请输入导出向量所至文件名: ");
            scanf("%49s", filename);
            export_vector_to_txt(new_vec, filename);
        }

        free_vector(new_vec);
    } else {
        printf("列向量初始化失败! \n");
    }
    break;
case 'G': // 向量绝对值
    if (vec != NULL && vec->is_init == 1) {
        abs_vector(vec);
        printf("向量已取绝对值\n");
    } else {

```

```

        printf("向量为空! \n");
    }
    break;

case 'H': // 向量元素取倒数
    if (vec != NULL && vec->is_init == 1) {
        reciprocal_elements_vector(vec);
        printf("向量元素已取倒数\n");
    } else {
        printf("向量为空! \n");
    }
    break;

case 'I': // 向量元素平方
    if (vec != NULL && vec->is_init == 1) {
        square_elements_vector(vec);
        printf("向量元素已取平方\n");
    } else {
        printf("向量为空! \n");
    }
    break;

case 'J': // 向量归一化, 用2范数
    if (vec != NULL && vec->is_init == 1) {
        normalize_vector(vec);
    } else {
        printf("向量为空! \n");
    }
    break;

case '4':
    printf("4. 统计与排序\n");
    printf("4.1. 查找向量中的最大元素 (K)\n");
    printf("4.2. 查找向量中的最小元素 (L)\n");
    printf("4.3. 查找向量中最大的c1个元素的索引 (M)\n");
    printf("4.4. 查找向量中最小的c1个元素的索引 (N)\n");
    printf("4.5. 从小到大排序向量 (O)\n");
    printf("4.6. 从大到小排序向量 (P)\n");
    printf("4.7. 向量范数 (Q)\n");
    break;

case 'K': // 查找向量中的最大元素
    if (vec != NULL && vec->is_init == 1) {
        double max = max_element_vector(*vec);
        printf("向量中的最大元素是: %lf\n", max);
    } else {
        printf("向量为空! \n");
    }
    break;

case 'L': // 查找向量中的最小元素
    if (vec != NULL && vec->is_init == 1) {
        double min = min_element_vector(*vec);
        printf("向量中的最小元素是: %lf\n", min);
    } else {
        printf("向量为空! \n");
    }
    break;

```

```

c1);

case 'M': // 查找向量中最大的c1个元素的索引
if (vec != NULL && vec->is_init == 1) {
    int c1;
    printf("请输入要查找的最大元素个数c1: ");
    scanf("%d", &c1);
    int* indices = (int*)malloc(c1 * sizeof(int));
    top_c1_max_indices(*vec, c1, indices);
    printf("是否要将最大的%d个元素的索引导出到文件? (y/n): ",
c1);

    char export_choice;
    scanf(" %c", &export_choice);
    if (export_choice == 'y' || export_choice == 'Y') {
        char filename[50];
        printf("请输入导出文件名: ");
        scanf("%49s", filename);
        export_indices_to_txt(indices, c1, filename);
        free(indices);
    }
} else {
    printf("向量为空! \n");
}
break;

case 'N': // 查找向量中最小的c1个元素的索引
if (vec != NULL && vec->is_init == 1) {
    int c1;
    printf("请输入要查找的最小元素个数c1: ");
    scanf("%d", &c1);
    int* indices = (int*)malloc(c1 * sizeof(int));
    top_c1_min_indices(*vec, c1, indices);
    printf("是否要将最小的%d个元素的索引导出到文件? (y/n): ",
c1);

    char export_choice;
    scanf(" %c", &export_choice);
    if (export_choice == 'y' || export_choice == 'Y') {
        char filename[50];
        printf("请输入导出文件名: ");
        scanf("%49s", filename);
        export_indices_to_txt(indices, c1, filename);
        free(indices);
    }
} else {
    printf("向量为空! \n");
}
break;

case 'O': // 从小到大排序向量
if (vec != NULL && vec->is_init == 1) {
    sort_vector_asc(vec);
    printf("向量已按从小到大排列\n");
} else {
    printf("向量为空! \n");
}
break;

case 'P': // 从大到小排序向量
if (vec != NULL && vec->is_init == 1) {
    sort_vector_desc(vec);

```

```

        printf("向量已按从大到小排列\n");
    } else {
        printf("向量为空! \n");
    }
    break;

case 'Q': // 向量范数
    if (vec != NULL && vec->is_init == 1) {
        int option;
        printf("请选择范数类型 (1: L1范数, 2: L2范数, 3: 无穷范数) : ");

        scanf("%d", &option);
        double norm = norm_vector(vec, option);
        printf("向量范数为: %lf\n", norm);
    } else {
        printf("向量为空! \n");
    }
    break;

case '5':
    printf("5. 算术操作\n");
    printf("5.1. 向量除以标量 (R)\n");
    printf("5.2. 向量乘以标量 (S)\n");
    break;
case 'R': // 向量除以标量
    if (vec != NULL && vec->is_init == 1) {
        double scalar;
        printf("请输入标量: ");
        scanf("%lf", &scalar);
        divide_vector_by_scalar(vec, scalar);
        if (scalar) {
            printf("向量已除以标量\n");
        }
    } else {
        printf("向量为空! \n");
    }
    break;

case 'S': // 向量乘以标量
    if (vec != NULL && vec->is_init == 1) {
        double scalar;
        printf("请输入标量: ");
        scanf("%lf", &scalar);
        multiply_vector_by_scalar(vec, scalar);
        printf("向量已乘以标量\n");
    } else {
        printf("向量为空! \n");
    }
    break;
case '0':
    printf("退出子菜单\n");
    break;
default:
    printf("无效的选项! \n");
    break;
}

} while (choice != '0');

```



```

    if (vec != NULL) {
        free_vector(vec);
    }
}

```

○ 单个矩阵操作

```

void submenu_matrix_operations() {
    show_submenu_matrix_operations();
    char choice;
    Matrix* mat = NULL;
    while (mat == NULL || mat->is_init == 0) {
        printf("请先导入矩阵，输入矩阵所在文件名: ");
        scanf("%49s", filename);
        mat = import_matrix_from_txt(filename);
        if (mat != NULL && mat->is_init == 1) {
            printf("矩阵导入成功! \n");
        } else {
            printf("矩阵导入失败! \n");
            printf("检查文件名是否正确或文件是否为空\n");
        }
    }

    char export_answer;
    double new_column[mat->rows];
    double max_values[mat->rows];
    double min_values[mat->rows];
    int zero_row_indices[mat->rows]; // 假设最多有mat->rows个零行
    do {
        printf("请输入您的选择（例如: A ）: ");
        char clean;
        while ((clean = getchar()) != '\n' && clean != EOF); // 清除输入缓冲区
        choice = getchar();

        switch (choice) {
            case '1':
                printf("1. 导入导出\n");
                printf("1.1. 导入矩阵数据 (A)\n");
                printf("1.2. 导出矩阵数据 (B)\n");
                break;
            case 'A': // 导入矩阵数据
                if (mat != NULL && mat->is_init == 1) {
                    printf("矩阵已导入，是否导入一个新矩阵覆盖当前矩阵? (y/n): ");

                    char overwrite;
                    while ((clean = getchar()) != '\n' && clean != EOF);
                    overwrite = getchar();

                    if (overwrite == 'y' || overwrite == 'Y') {
                        // 释放当前矩阵
                        free_matrix(mat);
                        printf("请输入新矩阵所在文件名: ");
                        scanf("%49s", filename);
                        mat = import_matrix_from_txt(filename);

                        while (mat == NULL || mat->is_init == 0) {

```

```

        printf("新矩阵导入失败! \n");
        printf("检查文件名是否正确或文件是否为空\n");
        printf("导入新矩阵, 请输入矩阵所在文件名: ");
        scanf("%49s", filename);
        mat = import_matrix_from_txt(filename);
        if (mat != NULL && mat->is_init == 1) {
            printf("矩阵导入成功! \n");
        }
    }

    } else {
        printf("保留当前矩阵\n");
    }
} else {
    printf("错误: 矩阵未初始化 \n");
}
break;
case 'B': // 导出矩阵数据
    if (mat != NULL && mat->is_init == 1) {
        printf("请输入导出矩阵所至文件名: ");
        scanf("%49s", filename);
        export_matrix_to_txt(mat, filename);
    } else {
        printf("没有矩阵可导出! \n");
    }
    break;
case '2':
    printf("2. 显示与切片\n");
    printf("2.1. 显示整个矩阵 (C)\n");
    printf("2.2. 显示矩阵元素 (D)\n");
    printf("2.3. 矩阵行切片 (E)\n");
    printf("2.4. 矩阵列切片 (F)\n");
    break;
case 'C': // 显示整个矩阵
    print_matrix(mat);
    break;
case 'D': // 显示矩阵元素
    int row, col;
    print_matrix_element(mat, row, col);
    break;
case 'E': // 矩阵行切片
    int row_index;
    printf("请输入要切片的行号: ");
    scanf("%d", &row_index);
    double* row_slice = slice_matrix_row(mat, row_index);
    if (row_slice != NULL) {
        printf("矩阵行切片成功\n");
        for (int i = 0; i < mat->cols; i++) {
            printf("%.21f ", row_slice[i]);
        }
        printf("\n");
        printf("是否要导出切片到文件? (Y/N): ");
        scanf(" %c", &export_answer);
        if (export_answer == 'Y' || export_answer == 'y') {
            FILE *file = fopen("row_slice.txt", "w");
            if (file != NULL) {
                for (int i = 0; i < mat->cols; i++) {
                    fprintf(file, "%.21f\n", row_slice[i]);
                }
            }
        }
    }
}

```

```

        }
        fclose(file);
        printf("切片已导出到row_slice.txt\n");

    } else {
        printf("无法打开文件以进行写入\n");
    }
}
free(row_slice);
}
break;
case 'F': // 矩阵列切片
    int col_index;
    printf("请输入要切片的列号: ");
    scanf("%d", &col_index);
    double* col_slice = slice_matrix_col(mat, col_index);
    if (col_slice != NULL) {
        printf("矩阵列切片成功\n");
        for (int i = 0; i < mat->rows; i++) {
            printf("%.21f\n", col_slice[i]);
        }
        printf("\n");
        printf("是否要导出切片到文件? (Y/N): ");
        scanf(" %c", &export_answer);
        if (export_answer == 'Y' || export_answer == 'y') {
            FILE *file = fopen("col_slice.txt", "w");
            if (file != NULL) {
                for (int i = 0; i < mat->rows; i++) {
                    fprintf(file, "%.21f\n", col_slice[i]);
                }
                fclose(file);
                printf("切片已导出到col_slice.txt\n");
            } else {
                printf("无法打开文件以进行写入\n");
            }
        }
        free(col_slice);
    }
    break;
case 'G':
    printf("请输入切片的大小: \n");
    printf("请输入要切片的行数: \n");
    scanf("%d", &row);
    printf("请输入要切片的列数: \n");
    scanf("%d", &col);
    while (row <= 0 || row > mat->rows || col <= 0 || col >
mat->cols) {

        printf("切片的大小有误! \n");
        printf("请输入切片的大小: \n");
        printf("请输入要切片的行数: \n");
        scanf("%d", &row);
        printf("请输入要切片的列数: \n");
        scanf("%d", &col);
    }
    int * indices_row = (int *)malloc(row * sizeof(int));
    int * indices_col = (int *)malloc(col * sizeof(int));
    printf("输入切片行索引: \n");
    for (int i = 0; i < row; i++) {

```

```

scanf("%d", &indices_row[i]);
if (indices_row[i] < 0 || indices_row[i] > mat-
>cols) {

    printf("行索引越界! \n");
    free(indices_col);
    free(indices_row);
}
}
printf("输入切片列索引: \n");
for (int i = 0; i < col; i++) {
    scanf("%d", &indices_col[i]);
    if (indices_col[i] < 0 || indices_col[i] > mat-
>rows) {

        printf("列索引越界! \n");
        free(indices_col);
        free(indices_row);
    }
}

Matrix * slice_matrix = (Matrix*)malloc(sizeof(Matrix));
if (!init_matrix(slice_matrix, row, col)) {
    printf("矩阵切片初始化失败! \n");
}
slice_matrix = slice_matrix_m(mat, row, col,
indices_col, indices_row);
printf("\n");
print_matrix(slice_matrix);
if_export_matrix_result(slice_matrix, filename);
free_matrix(slice_matrix);
free(indices_col);
free(indices_row);
break;
case 'H': // 矩阵元素取倒数
    reciprocal_elements_matrix(mat);
    printf("矩阵元素已取倒数\n");
    break;
case 'I': // 矩阵元素平方
    square_elements_matrix(mat);
    printf("矩阵元素已取平方\n");
    break;
case 'J': // 替换矩阵指定列
    int col_to_replace;
    printf("请输入要替换的列号: \n");
    scanf("%d", &col_to_replace);
    printf("请输入新列的元素 (输入%d个值): \n", mat->rows);
    for (int i = 0; i < mat->rows; i++) {
        scanf("%lf", &new_column[i]);
    }
    replace_column(mat, new_column, col_to_replace);
    printf("矩阵第%d列已替换\n", col_to_replace);
    break;
case 'K': // 矩阵转置
    transpose_matrix(mat);
    printf("矩阵已转置\n");
    break;
case 'L': // 矩阵所有元素求和
    double sum = sum_of_matrix(mat);
    printf("矩阵所有元素的和为: %.2lf\n", sum);

```

```

        break;
    case 'M': // 查找矩阵中的最大元素
        max_elements_matrix(mat);
        break;
    case 'N': // 每行的最大元素
        max_elements_per_row(mat, max_values);
        printf("每行的最大元素为: \n");
        for (int i = 0; i < mat->rows; i++) {
            printf("%.21f\n", max_values[i]);
        }
        break;
    case 'O': // 每行的最小元素
        min_elements_per_row(mat, min_values);
        printf("每行的最小元素为: \n");
        for (int i = 0; i < mat->rows; i++) {
            printf("%.21f\n", min_values[i]);
        }
        break;
    case 'P': // 整行元素都为0的行编号
        int max_indices = 0;
        find_zero_row_indices(mat, zero_row_indices,
&max_indices);
        if (zero_row_indices != NULL && max_indices != 0) {
            printf("整行元素都为0的行编号为: \n");
            for (int i = 0; i < max_indices; i++) {
                printf("%d\n", zero_row_indices[i]);
            }
            printf("是否要导出整行元素都为0的行编号到文件? (Y/N): ");
            char export_answer;
            scanf(" %c", &export_answer);
            if (export_answer == 'Y' || export_answer == 'y') {
                FILE *file = fopen("zero_row_indices.txt", "w");
                if (file != NULL) {
                    for (int i = 0; i < max_indices; i++) {
                        fprintf(file, "%d\n",
zero_row_indices[i]);
                    }
                    fclose(file);
                    printf("切片已导出到zero_row_indices.txt\n");
                } else {
                    printf("无法打开文件以进行写入\n");
                }
            }
        } else {
            printf("没有0行! \n");
        }
        break;
    case 'Q': // 矩阵范数特征值
        if (mat != NULL && mat->is_init == 1) {
            int option;
            printf("请选择类型 (1: L1范数, 2: L2范数, 3: 无穷范数, 4:
特征值): ");

            scanf("%d", &option);
            norm_matrix(mat, option);
        } else {
            printf("矩阵为空! \n");
        }
    }
}

```

```

    }
    break;
case 'R': // 矩阵列向量间的夹角Cos值
    double* all_cos = calculate_all_cos(mat);
    if (all_cos != NULL) {
        printf("矩阵列向量间的夹角Cos值为: \n");
        for (int i = 0; i < mat->cols * (mat->cols - 1) / 2;
i++) {
            printf("%.21f\n", all_cos[i]);
        }
        free(all_cos); // 释放内存
    }
    break;
case 'S': // 矩阵乘以标量
    double scalar;
    printf("请输入标量: ");
    scanf("%lf", &scalar);
    multiply_matrix_by_scalar(mat, scalar);
    printf("矩阵已乘以标量\n");
    break;
case 'T': // 生成随机对称矩阵
    printf("请输入对称矩阵大小: ");
    int size;
    scanf("%d", &size);
    Matrix* random_sym_mat =
generate_random_symmetric_matrix(size);
    if (random_sym_mat != NULL) {
        printf("随机对称矩阵生成成功\n ");
        print_matrix(random_sym_mat);
        printf("是否导出? (y/n): ");
        char export_random_sym_mat;
        while ((clean = getchar()) != '\n' && clean != EOF);
        export_random_sym_mat = getchar();
        if (export_random_sym_mat == 'y' ||
export_random_sym_mat == 'Y') {
            printf("随机对称");
            export_matrix_to_txt(random_sym_mat,
"random_sym_mat.txt");
        }
    } else {
        printf("随机对称矩阵生成失败! \n");
    }
    free_matrix(random_sym_mat);
    break;
default:
    printf("无效的选项! \n");
    break;
}
} while (choice != '0');
if (mat != NULL) {
    free_matrix(mat);
}
}
}

```

◦ 向量与向量操作

```

void submenu_vector_vector_operations() {

```

```

Vector* vec1 = NULL;
Vector* vec2 = NULL;
import_vector_2(&vec1, &vec2);
Vector result;
init_vector(&result, vec1->size);
char choice;
do {
    show_submenu_vector_vector_operations();
    printf("请输入您的选择（例如：A）：");
    scanf(" %c", &choice);
    switch (choice) {
        case '1':
            printf("1. 算术操作\n");
            printf("1.1 向量加法 (A)\n");
            printf("1.2 向量减法 (B)\n");
            printf("1.3 向量按元素除法 (C)\n");
            printf("1.4 向量按元素乘法 (D)\n");
            break;
        case 'A': // 向量加法
            result = add_vectors(*vec1, *vec2);
            if (result.is_init) {
                print_vector(&result);
                if_export_vec_result(&result, filename); // 导出结果
                free_vector(&result);
            }
            break;
        case 'B': // 向量减法
            result = subtract_vectors(*vec1, *vec2);
            if (result.is_init) {
                print_vector(&result);
                if_export_vec_result(&result, filename); // 导出结果
                free_vector(&result);
            }
            break;
        case 'C': // 向量按元素除法
            result = elementwise_divide(*vec1, *vec2);
            if (result.is_init) {
                print_vector(&result);
                if_export_vec_result(&result, filename); // 导出结果
                free_vector(&result);
            }
            break;
        case 'D': // 向量按元素乘法
            result = elementwise_multiply(*vec1, *vec2);
            if (result.is_init) {
                print_vector(&result);
                if_export_vec_result(&result, filename); // 导出结果
                free_vector(&result);
            }
            break;
        case '2':
            printf("2. 余弦与点积\n");
            printf("2.1 余弦 (E)\n");
            printf("2.2 点积 (F)\n");
            break;
        case 'E': // 余弦
            cos_v1_v2(*vec1, *vec2);
            break;
    }
}

```

矩阵结果

```
case 'F': // 点积
    dot_product_vectors(vec1, vec2);
    break;
case '3':
    printf("3. 外积\n");
    printf("    3.1 向量外积 (G)\n");
    break;
case 'G': // 向量外积
    Matrix outer_prod ;
    init_matrix(&outer_prod, vec1->size, vec1->size);
    outer_prod = vector_outer_product(vec1, vec2);
    print_matrix(&outer_prod);
    if_export_matrix_result(&outer_prod, filename); // 导出矩阵结果

    free_matrix(&outer_prod);
    break;
case '0':
    printf("退出子菜单\n");
    break;
case 'a':
    printf("是否导入两个新向量,覆盖当前向量? (y/n): ");
    char overwrite;
    scanf(" %c", &overwrite);
    if (overwrite == 'y' || overwrite == 'Y') {
        // 释放当前向量并重新导入
        free_vector(vec1);
        free_vector(vec2);
        import_vector_2(&vec1, &vec2);
    } else {
        printf("保留当前向量\n");
    }
    break;
case 'b':
    printf("向量1: \n");
    print_vector(vec1);
    printf("向量2: \n");
    print_vector(vec2);
    break;
default:
    printf("无效的选项! \n");
    break;
}
} while (choice != '0');
free_vector(vec1);
free_vector(vec2);
free_vector(&result);
}
```

◦ 向量与矩阵操作

```
void submenu_vector_matrix_operations() {
    show_submenu_vector_matrix_operations();
    Matrix* mat = NULL;
    Vector* vec = NULL;
    while (mat == NULL || mat->is_init == 0) {
        printf("请先导入矩阵, 输入矩阵所在文件名: ");
        scanf("%49s", filename);
```



```

mat = import_matrix_from_txt(filename);
if (mat != NULL && mat->is_init == 1) {
    printf("矩阵导入成功! \n");
} else {
    printf("矩阵导入失败! \n");
    printf("检查文件名是否正确或文件是否为空\n");
}
}

while (vec == NULL || vec->is_init == 0) {
    printf("请先导入向量, 输入向量所在文件名: ");
    scanf("%49s", filename);
    vec = import_vector_from_txt(filename);
    if (vec != NULL && vec->is_init == 1) {
        printf("向量导入成功! \n");
    } else {
        printf("向量导入失败! \n");
        printf("检查文件名是否正确或文件是否为空\n");
    }
}

char choice;
Vector result;
init_vector(&result, vec->size);

do {
    printf("请输入您的选择 (例如: M ): ");
    scanf(" %c", &choice);
    switch (choice) {
        case 'M': // 矩阵乘以向量
            result = multiply_matrix_by_vector(mat, vec);
            if (result.is_init) {
                print_vector(&result);
                if_export_vec_result(&result, filename);
            }
            break;
        case 'N': // 向量乘以矩阵
            result = multiply_vector_by_matrix(vec, mat);
            if (result.is_init) {
                for (int i = 0; i < result.size; i++) {
                    printf(" %.2lf ", result.data[i]);
                }
                if_export_vec_result(&result, filename);
            }
            break;
        case 'a':
            printf("是否导入新向量和矩阵1? (y/n): ");
            char overwrite;
            scanf(" %c", &overwrite);
            if (overwrite == 'y' || overwrite == 'Y') {
                // 释放当前并重新导入
                free_matrix(mat);
                free_vector(vec);
                while (mat == NULL || mat->is_init == 0) {
                    printf("请导入新矩阵, 输入新矩阵所在文件名: ");
                    scanf("%49s", filename);
                    mat = import_matrix_from_txt(filename);
                    if (mat != NULL && mat->is_init == 1) {
                        printf("新矩阵导入成功! \n");
                    }
                }
            }
        }
    }
}

```

```

        } else {
            printf("新矩阵导入失败! \n");
            printf("检查文件名是否正确或文件是否为空\n");
        }
    }
    while (vec == NULL || vec->is_init == 0) {
        printf("请导入新向量, 输入新向量所在文件名: ");
        scanf("%49s", filename);
        vec = import_vector_from_txt(filename);
        if (vec != NULL && vec->is_init == 1) {
            printf("新向量导入成功! \n");
        } else {
            printf("新向量导入失败! \n");
            printf("检查文件名是否正确或文件是否为空\n");
        }
    }
} else {
    printf("保留当前向量和矩阵\n");
}
break;
case 'b':
    printf("向量: \n");
    print_vector(vec);
    printf("矩阵: \n");
    print_matrix(mat);
    break;
case '0':
    printf("退出子菜单\n");
    break;
default:
    printf("无效的选项! \n");
    break;
}
} while (choice != '0');
free_vector(vec);
free_vector(&result);
free_matrix(mat);
}

```

○ 矩阵与矩阵操作

```

void submenu_matrix_matrix_operations() {
    show_submenu_matrix_matrix_operations();
    char choice;
    Matrix* mat1 = NULL;
    Matrix* mat2 = NULL;
    import_matrix_2(&mat1, &mat2);
    Matrix result;
    do {
        printf("请输入您的选择 (例如: A ): ");
        scanf(" %c", &choice);
        switch (choice) {
            case 'A': // 矩阵加法
                init_matrix(&result, mat1->rows, mat1->cols);
                result = add_matrices(mat1, mat2);
                if (result.is_init) {
                    print_matrix(&result);
                }
            }
        }
    } while (choice != '0');
}

```

```

        if_export_matrix_result(&result, filename);
    }
    break;
case 'B': // 矩阵减法
    init_matrix(&result, mat1->rows, mat1->cols);
    result = subtract_matrices(mat1, mat2);
    if (result.is_init) {
        print_matrix(&result);
        if_export_matrix_result(&result, filename);
    }
    break;
case 'E': // AB
    init_matrix(&result, mat1->rows, mat2->cols);
    result = multiply_AB(mat1, mat2);
    if (result.is_init) {
        print_matrix(&result);
        if_export_matrix_result(&result, filename);
    }
    break;
case 'F': // ATBA
    init_matrix(&result, mat1->cols, mat1->cols);
    result = multiply_ATBA(mat1, mat2);
    if (result.is_init) {
        print_matrix(&result);
        if_export_matrix_result(&result, filename);
    }
    break;
case 'C': // 矩阵按元素乘法
    init_matrix(&result, mat1->rows, mat1->cols);
    result = elementwise_multiply_matrices(mat1, mat2);
    if (result.is_init) {
        print_matrix(&result);
        if_export_matrix_result(&result, filename);
    }
    break;
case 'D': // 矩阵按元素除法
    init_matrix(&result, mat1->rows, mat1->cols);
    result = elementwise_divide_matrices(mat1, mat2);
    if (result.is_init) {
        print_matrix(&result);
        if_export_matrix_result(&result, filename);
    }
    break;
case 'a':
    printf("是否导入两个新矩阵,覆盖当前矩阵? (y/n): ");
    char overwrite;
    scanf(" %c", &overwrite);
    if (overwrite == 'y' || overwrite == 'Y') {
        // 释放当前并重新导入
        free_matrix(mat1);
        free_matrix(mat2);
        import_matrix_2(&mat1, &mat2);
    } else {
        printf("保留当前矩阵\n");
    }
    break;
case 'b':
    printf("矩阵1: \n");

```



```

        printf("矩阵导入失败! \n");
        printf("检查文件名是否正确或文件是否为空\n");
    }
}
Matrix* inv_mat = invert_upper_triangular_matrix(*mat);
if (inv_mat == NULL) {
    printf("无法计算逆矩阵。 \n");
} else {
    printf("上三角矩阵为:\n");
    print_matrix(mat);
    printf("上三角矩阵的逆为:\n");
    print_matrix(inv_mat);
    if_export_matrix_result(mat, filename);
    free_matrix(inv_mat);
}
free_matrix(mat);
break;
case 'B': // 上三角矩阵方程求解
    if (mat == NULL || mat->is_init == 0 || vec == NULL ||
vec->is_init == 0) {
        printf("请先导入上三角矩阵方程\n");
        import_linear_equation( &vec, &mat);
        int is_upper_triangular = 1;
        for (int i = 1; i < mat->rows &&
is_upper_triangular; i++) {
            for (int j = 0; j < i; j++) {
                if (mat->data[i * mat->cols + j] != 0) { //
如果下三角部分有非零元素
                    is_upper_triangular = 0; // 标记矩阵不是上
三角的
                }
            }
        }
        if (!is_upper_triangular) {
            printf("该矩阵不是上三角矩阵! \n");
            free_vector(vec);
            free_vector(&result);
            free_matrix(mat);
            break;
        }
    }
    print_matrix(mat);
    printf("* x =\n");
    print_vector(vec);
    init_vector(&result, vec->size);
    solve_upper_triangular_equation(mat, vec, &result);
    if (result.is_init) {
        printf("方程解为: \n");
        print_vector(&result);
        if_export_vec_result(&result, filename);
    } else {
        printf("求解失败! ");
    }
    free_vector(vec);
    free_vector(&result);
    free_matrix(mat);
    break;

```

```

case 'C': // 一般线性方程求解
    printf("请先导入线性方程\n");
    import_linear_equation( &vec, &mat);
    print_matrix(mat);
    printf("* x =\n");
    print_vector(vec);
    init_vector(&result, vec->size);
    solve_linear_equations(mat, vec, &result);
    if (result.is_init) {
        printf("方程解为: \n");
        print_vector(&result);
        if_export_vec_result(&result, filename);
    } else {
        printf("求解失败! ");
    }
    free_vector(vec);
    free_vector(&result);
    free_matrix(mat);
    break;
case 'a':
    printf("是否导入新方程? (y/n): ");
    char overwrite;
    scanf(" %c", &overwrite);
    if (overwrite == 'y' || overwrite == 'Y') {
        // 释放当前并重新导入
        free_matrix(mat);
        free_vector(vec);
        printf("请导入新方程\n");
        import_linear_equation( &vec, &mat);
        printf("%d\n", mat->is_init);
        print_matrix(mat);
    } else {
        printf("保留当方程\n");
    }
    break;
case '0':
    printf("退出子菜单\n");
    break;
default:
    printf("无效的选项! \n");
    break;
    }
} while (choice != '0');
}

```

开发工具说明

C语言:

- [RedPandaIDE \(小熊猫C .3.0.winxp.mingw32.绿色版\)](#)
- VS code

流程图:

- [未命名绘图 - debug996.com](#) (<https://debug996.com/draw/draw.html#>)
- [draw.io \(diagrams.net\)](#) ([draw.io \(diagrams.net\)](https://draw.io))

报告:

- Typora

结语

写这些代码时正值高考，想到去年今日我也已高考结束，感慨颇多。完成大作业虽然耗时很久，但也获益良多。愿今日的我，能够不负以往的期望，继续前行