Computer algebra systems and Representations of vectors and matrices

1. Short introduction:

A computer algebra system (CAS) is any mathematical software that can process mathematical expressions in a manner like traditional manual calculations by mathematicians and scientists.

2. Application field:

1. Symbolic calculations and exact results
2. Equation solving and analytical calculations
3. Mathematical Derivation and Proof
4. Multi-field application

3. Vector:

1. vector representation:
    1. Array:
    2. List:
    3. Specialized Vector Data Types
2. vector operation (in python):

    Suppose there are two vectors $\mathbf{v} = [1, 2, 3]$ and $\mathbf{w} = [4, 5, 6]$

```python
# Define vectors
v = vector([1, 2, 3])
w = vector([4, 5, 6])

# Vector addition
result_add = v + w
print("Vector addition result:", result_add)

# Vector subtraction
result_sub = v - w
print("Vector subtraction result:", result_sub)

# Dot product
result_dot = v.dot_product(w)
print("Dot product result:", result_dot)

# Cross product
result_cross = v.cross_product(w)
print("Cross product result:", result_cross)

# Scalar multiplication
scalar = 2
result_scalar_mult = scalar * v
print("Scalar multiplication result:", result_scalar_mult)
```

4. Matrices:

1. Matrix representation:

    1. Array:

Arrays are represented using one-dimensional or two-dimensional arrays. Each array element corresponds to an element of the matrix. For two-dimensional arrays, the elements of the matrix can be accessed by row searching and list indexing.

2. list nesting:

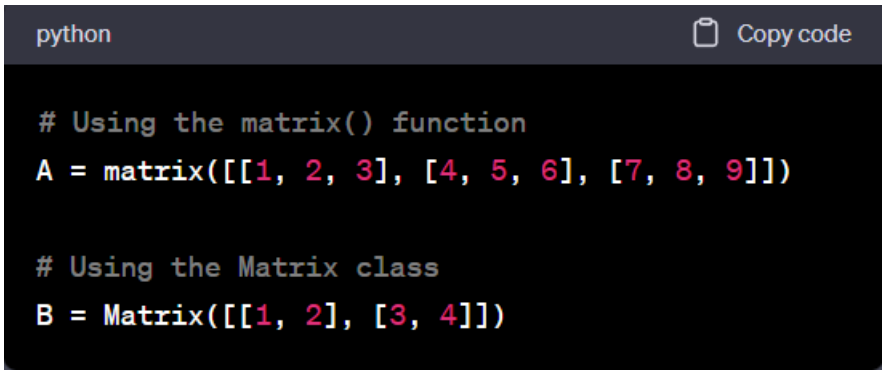
The outer list represents the rows of the matrix, and the inner list represents the columns of the matrix. Elements of a matrix can be accessed by row index and column index.

3. specialized array data structure

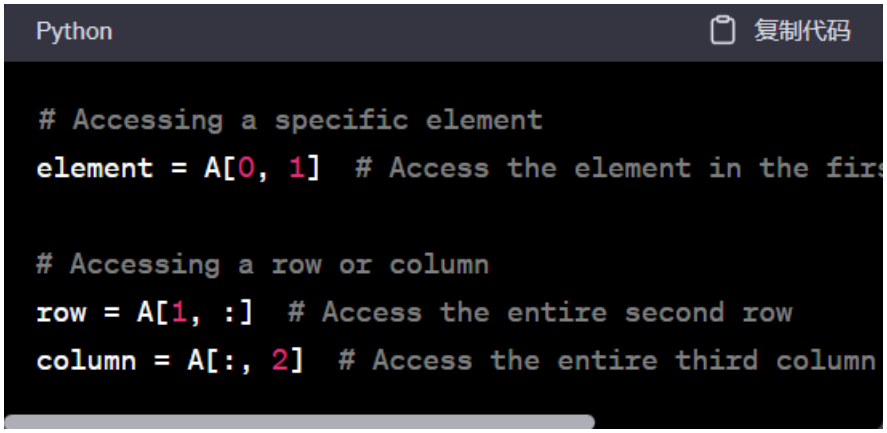
Many CASs provide specialized matrix data structures for representing and manipulating matrices. These data structures usually have rich functions and methods that can be used for various operations and calculations on matrices.

2. Matrix Operations (in python):

1. Creating a matrix:



2. Access matrix elements:



3. perform matrix operations:

Python 复制代码

```python
# Matrix addition
C = A + B

# Matrix subtraction
D = A - B

# Matrix multiplication
E = A * B

# Matrix transpose
F = A.transpose()

# Matrix inverse
G = A.inverse()

# Determinant calculation
det = A.det()
```

4.Gaussian reduction

```python
import numpy as np
def gaussian_elimination(A, b):
    # Concatenate the augmented matrix
    augmented_matrix = np.concatenate((A, b.reshape(-1, 1)), axis=1)
    # Perform Gaussian elimination
    rows, cols = augmented_matrix.shape
    for i in range(rows):
        # Find the row with the maximum element in the current column
        max_row = i
        for j in range(i + 1, rows):
            if abs(augmented_matrix[j, i]) > abs(augmented_matrix[max_row, i]):
                max_row = j
        # Swap the rows to bring the maximum element to the current row
        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]
        # Perform row operations to eliminate the elements below the current row
        for j in range(i + 1, rows):
            factor = augmented_matrix[j, i] / augmented_matrix[i, i]
            augmented_matrix[j] -= factor * augmented_matrix[i]
```