functions

unknown

Oct 20, 2022

CONTENTS

CHAPTER

ONE

FUNCTIONS

This module implement basic matrix operation, such as multiplication, determinant, inverse, solve linear equation, adjugate

See `functions.mult()` , `functions.inv()` ,:func:*functions.solve* , `functions.det()` , `functions.adj()`

def det(a):

“”” Given a martix *a*, return its determinat or *None* if its determinant does not exist.

a

[np.array or list of lists] ‘n x m’ array

det

[np.float64 or None] The determinant of *a*.

```
>>> a = [[2, 0, -1], [0, 5, 6], [0, -1, 1]]
>>> d = det(a)
>>> d
22
>>> b = [[2, 2, -3], [1, 5, 3], [2, -4, 1]]
>>> c = det(b)
>>> c
86
```

See https://en.wikipedia.org/wiki/Gaussian_elimination for further details. “”” a = np.array(a) n = np.shape(a)[0] p = np.shape(a)[1] tot = 0 if n == 1:

return a[0][0]

if n != p:

return None

for i in range(p):

newrow = np.delete(a, 1, 0) newrow = np.delete(newrow, i, 1) tot += (-1) ** (i+1) * det(newrow) * a[1, i]

return tot

def mult(A, B):

“”” Given two matrix A and B , return its multiplication or None if the column length of A are not equal to row length of B .

A

[np.array or list of lists] ‘n x m’ array

B

[np.array or list of lists] ‘m x k ‘ array

mult
[np.array or None] The multiplication of A and B , the size should be 'n x k' .

```
>>> A = np.array([[1, 2],[3, 4]])
>>> B = np.array([[5], [6]])
>>> D = mult(A, B)
>>> D
array([[17.],
       [39.]])
"""
rowA, rowB = np.size(A, 0), np.size(B, 0)
colA, colB = np.size(A, 1), np.size(B, 1)
if colA != rowB:
    return None
new_array = np.zeros((rowA, colB))
for i in range(rowA):
    for j in range(colB):
        for k in range(rowB):
            new_array[i][j] += A[i][k] * B[k][j]
return new_array
```

def adj(A):
"""" Given a martix *A*, return its adjugate matrix or *None* if its matrix size is not square

A
[np.array or list of lists] 'n x m' array

new
[np.array or None] The adjugate of A.

```
>>> a = [[1, 0, -1], [-2, 3, 0], [1, -3, 2]]
>>> d = adj(a)
>>> d
array([[6., 3., 3.],
       [4., 3., 2.],
       [3., 3., 3.]])
"""
A = np.array(A)
n = np.shape(A)[0]
m = np.shape(A)[1]
if n != m:
    return None
new = np.zeros((n, m))
for i in range(n):
    for j in range(m):
        newrow = np.delete(A, i, 0)
        newrow = np.delete(newrow, j, 1)
        new[i][j] = (-1) ** (i + j) * det(newrow)
```

new = np.transpose(new) return new

def inv(a):
"""" Given a martix *a*, return its inverse matrix or *None* if its matrix size is not square

a
[np.array or list of lists] 'n x m' array

new
[np.array or None] The inverse of a.

```
>>> a = [[1, 0, -1], [-2, 3, 0], [1, -3, 2]]
>>> d = inv(a)
>>> d
array([[2.        , 1.        , 1.        ],
       [1.33333333, 1.        , 0.66666667],
       [1.        , 1.        , 1.        ]])
"""
A = np.array(a)
n = np.shape(A)[0]
m = np.shape(A)[1]
if n != m or det(a) == 0:
    return None
```

new = 1/det(a) * adj(a)

return new

def solve(a, b):
“”” Given martix *a* and *b*, when ax = b, return x or None if a is square matrix or det(a) is zero

a
[np.array or list of lists] ‘n x m’ array

new
[np.array or None] The inverse of a.

```
>>> a = [[1, 0, -1], [-2, 3, 0], [1, -3, 2]]
>>> b = [[1],[2],[-1]]
>>> d = solve(a, b)
>>> d
array([[3.        ],
       [2.66666667],
       [2.        ]])
"""
n = np.shape(a)[0]
m = np.shape(a)[1]
b = np.array(b)
a = np.array(a)
res = np.zeros((n, 1))
if n != m:
    return None
if det(a) != 0:
    for j in range(n):
        c = np.array(a)
        c[(slice(None, None, 1), slice(j, j + 1, 1))] = b
        res[j, :] = det(c)
    result = 1 / det(a) * res
    return result
else:
    return None
```