

Если не хватает мозгов ,чтобы портировать на мобилки уже давно написанные NumPy ,SciPy и т.д.,то можно написать свой собственный модуль [mls.py](#) ,решающий практически любые задачи линейной алгебры и тем самым ,хотя бы отчасти оправдать свое жалкое существование...

Модуль [mls.py](#) действительно функционален - об этом можно судить обзорев его состав:

```
>>>import mls
```

```
>>>dir(mls)
```

```
['M', '__builtins__', '__doc__', '__file__', '__name__', '__package__', 'cond', 'det', 'eigenvv', 'gd', 'gdc', 'gt', 'inv', 'kA', 'mix', 'mnr', 'mult', 'mx', 'norm', 'replace', 'solver', 'srt', 'st', 'sub', 'sum', 'tran']
```

Среди этого бессмысленного набора наименований особого внимания заслуживают класс M и функции eigenvv,solver. Поговорим о них подробнее. Сначала посмотрим как работать с классом M:

```
>>> from mls import*
>>> e=M([[1,0,0],[0,1,0],[0,0,1]]) #создана ед. матрица
>>> a=M([[2,-3,9],[3,-1,7],[5,0,2]]) # матрица создается как вложенный массив по столбцам
>>> b=M([[5,1,-1],[0,9,1],[7,0,3]])
>>> (a+b)*(a**2-a*b+b**2)
[[-73, 116, 406], [109, 630, 665], [152, 10, 390]]
>>> tran(_) # транспонирование пред. матрицы
[[-73, 109, 152], [116, 630, 10], [406, 665, 390]]
```

```
>>> det(a-3*e)
62
>>> c=M([[1+4j,2-3j,3],[9-4.5j,6.8j,2+4j],[2,9j,3-5j]])
>>> a*c-b*a
[(-50-1j), (20-9j), (7+15j)], [(-36+31.399999999999999j), (-21+6.7000000000000002j),
(68+15.100000000000001j)], [(-20+2j), (-11-9j), (23+53j)]]
>>> det(_)
(41922.399999999994+24906.200000000004j)
```

Функция `eigenvv` определяет собственные числа и векторы квадратных матриц порядков 2,3,4.

```
>>> eigenvv(a)
```

```
eigenvalue= (7.1397902+0j)
eigenvector= [(0.800580284728583+0j), (-0.29506176607960644+0j), 1.0]
```

```
eigenvalue= (1.2053502+0j)
eigenvector= [(-0.7351167449927356+0j), 1.0, (-0.48316792512297457+0j)]
```

```
eigenvalue= (-5.3451405+0j)
eigenvector= [(-0.53098737261804496+0j), (-0.36660773520537138+0j), 1.0]
```

```
([(7.1397902000000002+0j), (1.2053502+0j), (-5.3451405000000003+0j)],
[(0.800580284728583+0j), (-0.29506176607960644+0j), 1.0], [(-0.7351167449927356+0j), 1.0,
(-0.48316792512297457+0j)], [(-0.53098737261804496+0j), (-0.36660773520537138+0j), 1.0]])
>>> eigenvv(a-c)
```

```
eigenvalue= (2.7461883-1.7488979j)
eigenvector= [1.0, (-0.025776960513960057-0.095347314019651561j),
```

```
(0.38748789462228034+0.59833818557634011j)]
```

```
eigenvalue= (3.4141302-10.582526j)  
eigenvector= [(-0.79888176082178219-0.43755011457163828j), 1.0,  
(0.39707013650005551-0.099214328021895337j)]
```

```
eigenvalue= (-7.1603186+6.5314239j)  
eigenvector= [(-0.5107648357925445-0.24363014942593963j),  
(-0.66954796666311345+0.063002584846649612j), 1.0]
```

```
([(2.7461883-1.7488979j), (3.4141301999999998-10.582526j),  
(-7.1603186000000001+6.5314239000000001j)], [[1.0,  
(-0.025776960513960057-0.095347314019651561j),  
(0.38748789462228034+0.59833818557634011j)],  
[(-0.79888176082178219-0.43755011457163828j), 1.0,  
(0.39707013650005551-0.099214328021895337j)],  
[(-0.5107648357925445-0.24363014942593963j),  
(-0.66954796666311345+0.063002584846649612j), 1.0]])
```

И наконец, с помощью функции `solver` можно решить любую линейную систему $AX=B$. Приводим соответствующие примеры.

```
>>> from mls import*  
>>> a=M([[1-5.6j,8,9+2.j],[2,3,0],[9.2-5j,9,2.2+5.4j]])  
>>> det(a)  
(-54.279999999999973+8.6400000000000148j)  
>>> inv(a)  
[[-0.072254985819114867-0.3099536307567643j),  
(-1.2112276594786844+0.27146265700265593j),  
(0.46796920722099689+0.18502678611623846j)],  
[(0.048169990546076576+0.20663575383784283j),  
(1.1408184396524563-0.18097510466843714j),  
(-0.31197947148066457-0.12335119074415898j)],  
[(0.21539056355446345-0.24206016084910537j),
```

```
(-1.1309725889185192-0.37162128165541686j),  
(0.18553258425776106+0.33903834797323257j)]]  
>>> a*_  
[[ (1+1.1102230246251565e-16j), -4.4408920985006262e-16j,  
 (3.3306690738754696e-16-4.4408920985006262e-16j), [5.5511151231257827e-17j,  
 (1.0000000000000004+4.4408920985006262e-16j), (-1.1102230246251565e-16+0j)],  
 [(-4.4408920985006262e-16-4.4408920985006262e-16j),  
 (-2.2204460492503131e-16-4.4408920985006262e-16j), (1.0000000000000004+0j)]]  
>>> b=a[0]  
>>> b  
[(1-5.5999999999999996j), 8, (9+2j)]  
>>> solver(a,b)  
Rang A= 3  
Rang A|B= 3  
X0=[(1-4.4408920985006262e-16j), 0j,  
(1.6653345369377348e-16+1.6653345369377348e-16j)]  
  
[[ (1-4.4408920985006262e-16j), 0j, (1.6653345369377348e-16+1.6653345369377348e-16j)]]  
>>> A=[[1,2],[2,3],[3,4],[3,5],[9,0]]  
>>> B=[1,5]  
>>> solver(A,B)  
Rang A= 2  
Rang A|B= 2  
X0=[0.0, 0.0, 0.0, 1.0, -0.2222222222222232]  
X1=[0.0, 0.0, 1.0, -0.79999999999999982, -0.066666666666666763]  
X2=[0.0, 1.0, 0.0, -0.59999999999999964, -0.02222222222222338]  
X3=[1.0, 0.0, 0.0, -0.39999999999999991, 0.02222222222222199]  
  
[[ (0.0, 0.0, 0.0, 1.0, -0.2222222222222232), [0.0, 0.0, 1.0, -0.79999999999999982,  
 -0.066666666666666763], [0.0, 1.0, 0.0, -0.59999999999999964, -0.02222222222222338],  
 [1.0, 0.0, 0.0, -0.39999999999999991, 0.02222222222222199]]
```

В последнем примере решена система уравнений:

$$x_1 + 2x_2 + 3x_3 + 3x_4 + 9x_5 = 1$$

$$2x_1 + 3x_2 + 4x_3 + 5x_4 = 5$$

Найдено частное решение X_0 и построен базис пространства решений однородной системы X_1, X_2, X_3 .