Matrix multiplication, solve Ax = b solve for x

So I was given a homework assignment that requires solving the coefficients of cubic splines. Now I clearly understand how to do the math on paper as well as with MatLab, I want to solve the problem with Python. Given an equation Ax = b where I know the values of A and b, I want to be able to solve for x with Python and I am having trouble finding a good resource to do such a thing.

Ex.

```
A = |1 0 0|
    |1 4 1|
    |0 0 1|

x = Unknown 3x1 matrix

b = |0 |
    |24|
    |0 |
```

Solve for x

python numpy matrix-multiplication

edited Mar 4 '14 at 8:18
Bach
2,996 2 13 35

asked Mar 4 '14 at 4:42
Scalahansolo
821 3 13 29

@MattDMo: OP already tagged `numpy` . – Amadan Mar 4 '14 at 4:43

I have looked at NumPy a little but its a lot to get through, do you know which NumPy function(s), or at least which area of NumPy would handle this best? – Scalahansolo Mar 4 '14 at 4:44

2 If `Ax = B` , `x = (A^-1)B` . Take a look at `inv` and `dot` functions. – Amadan Mar 4 '14 at 4:45

1 as a general reference, take a look at the NumPy for Matlab Users page if you haven't come across it already. Scrolling down, there's a big list of linear algebra equivalents that may be helpful, as well as a variety of other comparisons to help you from Matlab to the exciting world of Python :) – MattDMo Mar 4 '14 at 4:48

1 Extra marks for disclosing this is a homework assignment. That lets us treat it accordingly, giving you the benefit of the work you have to do to understand the solutions. – holdenweb Mar 4 '14 at 5:32

3 Answers

In a general case, use `solve` :

```
>>> import numpy as np
>>> from scipy.linalg import solve
>>>
>>> A = np.random.random((3, 3))
>>> b = np.random.random(3)
>>>
>>> x = solve(A, b)
>>> x
array([ 0.98323512,  0.0205734 ,  0.06424613])
>>>
>>> np.dot(A, x) - b
array([ 0.,  0.,  0.])
```

If your problem is banded (which cubic splines it often are), then there's
http://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.solve_banded.html

To comment on some of the comments to the question: better *not* use `inv` for solving linear systems. `numpy.lstsq` is a bit different, it's more useful for fitting.

As this is homework, you're really better off at least reading up on ways of solving tridiagonal linear systems.

edited Sep 6 '14 at 10:21
jeffery_the_wind
5,061 15 63 115

answered Mar 4 '14 at 8:41
ev-br
12.4k 3 30 56

Numpy is the main package for scientific computing in Python. If you are windows user then download it here: http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy else follow these instructions: http://www.scipy.org/install.html.

```
import numpy
A = [[1,0,0],[1,4,1],[0,0,1]]
b = [0,24,0]
x = numpy.linalg.lstsq(A,b)
```

answered Mar 4 '14 at 5:00

BushMinusZero
506 4 12

In addition to the code of Zhenya, you might also find it intuitive to use the np.dot function:

```
import numpy as np
A = [[1,0,0],
    [1,1,1],
    [6,7,0]]
b = [0,24,0]
# Now simply solve for x
x = np.dot(np.linalg.inv(A), b)
#np.linalg.inv(A)  is simply the inverse of A, np.dot is the dot product
print x

Out[27]: array([  0.,   0.,  24.])
```

answered Aug 4 '14 at 19:25

moldovean
1,622 20 22