Python (NumPy, SciPy), finding the null space of a matrix

I'm trying to find the null space (solution space of Ax=0) of a given matrix. I've found two examples, but I can't seem to get either to work. Moreover, I can't understand what they're doing to get there, so I can't debug. I'm hoping someone might be able to walk me through this.

The documentation pages (`numpy.linalg.svd` , and `numpy.compress`) are opaque to me. I learned to do this by creating the matrix `C = [A|0]` , finding the reduced row echelon form and solving for variables by row. I can't seem to follow how it's being done in these examples.

Thanks for any and all help!

Here is my sample matrix, which is the same as the wikipedia example:

```
A = matrix([
    [2,3,5],
    [-4,2,3]
    ])
```

Method (found here, and here):

```
import scipy
from scipy import linalg, matrixr
def null(A, eps=1e-15):
    u, s, vh = scipy.linalg.svd(A)
    null_mask = (s <= eps)
    null_space = scipy.compress(null_mask, vh, axis=0)
    return scipy.transpose(null_space)
```

When I try it, I get back an empty matrix:

```
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import scipy
>>> from scipy import linalg, matrix
>>> def null(A, eps=1e-15):
...    u, s, vh = scipy.linalg.svd(A)
...    null_mask = (s <= eps)
...    null_space = scipy.compress(null_mask, vh, axis=0)
...    return scipy.transpose(null_space)
...
>>> A = matrix([
...     [2,3,5],
...     [-4,2,3]
...     ])
>>>
>>> null(A)
array([], shape=(3, 0), dtype=float64)
>>>
```

numpy matrix scipy linear-algebra svd

edited Nov 9 '15 at 22:21
ali_m
24.4k 4 51 100

asked May 4 '11 at 19:56
Nona Urbiz
1,724 11 38 73

2 The wikipedia page you linked to actually gives a very nice explanation of why you should use an SVD to calculate the null space (or solve) of a matrix when you're dealing with floating point values. en.wikipedia.org/wiki/… The approach you describe (solving for variables row-by-row) will amplify any rounding errors, etc. (This is the same reason you should almost never explicitly calculate the inverse of a matrix...) – Joe Kington May 4 '11 at 20:29

5 Answers

It appears to be working okay for me:

```
A = matrix([[2,3,5],[-4,2,3],[0,0,0]])
A * null(A)
>>> [[  4.02455846e-16]
>>>  [  1.94289029e-16]
>>>  [  0.00000000e+00]]
```

answered May 4 '11 at 20:29

Bashwork
1,366 6 13

I'm sure I'm missing something, but Wikipedia suggests that the values should be `[ [-.0625], [-1.625], [1] ]` ? – Nona Urbiz May 4 '11 at 20:36

Moreover, it's returning an empty matrix for me `[]` . What could be wrong? – Nona Urbiz May 4 '11 at 20:58

6 @Nona Urbiz - It's returning an empty matrix because you're not putting in a row of zeros, as Bashwork (and wikipedia) does above. Also, the null space values returned (`[-0.33, -0.85, 0.52]`) are normalized so that the magnitude of the vector is 1. The wikipedia example is not normalized. If you just take `n = null(A)` and have a look at `n / n.max()` , you'll get `[-.0625, -1.625, 1]` . – Joe Kington May 4 '11 at 21:08

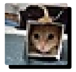
3 @Bashwork - How would I know to programmatically add a row of zeroes? Does the matrix have to be square? – Coder Sep 2 '12 at 20:27

You get the SVD decomposition of the matrix `A` . `s` is a vector of eigenvalues. You are interested in almost zero eigenvalues (see $A*x=\lambda*x$ where $\abs(\lambda)<\epsilon$), which is given by the vector of logical values `null_mask` .

Then, you extract from the list `vh` the eigenvectors corresponding to the almost zero eigenvalues, which is exactly what you are looking for: a way to span the null space. Basically, you extract the rows and then transpose the results so that you get a matrix with eigenvectors as columns.

edited May 4 '11 at 20:17

answered May 4 '11 at 20:05



Wok
2,786 2 17 41

Thank you very much for taking the time to reply and help me. Your answer was very helpful to me, but I accepted Bashworks answer as ultimately, it brought me to the solution. The only reason I am able to understand the solution though, is your response. – Nona Urbiz May 4 '11 at 21:16

No worry, I thought your problem was something else. – Wok May 5 '11 at 10:03

Sympy makes this straightforward.

```
>>> from sympy import Matrix
>>> A = [[2, 3, 5], [-4, 2, 3], [0, 0, 0]]
>>> A = Matrix(A)
>>> A * A.nullspace()[0]
Matrix([
[0],
[0],
[0]])
>>> A.nullspace()
[Matrix([
[-1/16],
[-13/8],
[    1]])]
```

edited Nov 11 '15 at 2:16

answered Nov 10 '14 at 2:09

Idr
1,447 2 16 34

Your method is almost correct. The issue is that the shape of s returned by the function scipy.linalg.svd is (K,) where K=min(M,N). Thus, in your example, s only has two entries (the singular values of the first two singular vectors). The following correction to your null function should allow it to work for any sized matrix.

```
import scipy
import numpy as np
from scipy import linalg, matrix
def null(A, eps=1e-12):
...    u, s, vh = scipy.linalg.svd(A)
...    padding = max(0,np.shape(A)[1]-np.shape(s)[0])
...    null_mask = np.concatenate(((s <= eps), np.ones((padding,),dtype=bool)),axis=0)
...    null_space = scipy.compress(null_mask, vh, axis=0)
...    return scipy.transpose(null_space)
A = matrix([[2,3,5],[-4,2,3]])
print A*null(A)
>>>[[  4.44089210e-16]
>>> [  6.66133815e-16]]
A = matrix([[1,0,1,0],[0,1,0,0],[0,0,0,0],[0,0,0,0]])
print null(A)
>>>[[ 0.         -0.70710678]
>>> [ 0.          0.        ]
>>> [ 0.          0.70710678]
>>> [ 1.          0.        ]]
print A*null(A)
>>>[[ 0.  0.]
>>> [ 0.  0.]
>>> [ 0.  0.]
>>> [ 0.  0.]]
```

edited Dec 19 '14 at 16:56

answered Oct 23 '14 at 19:25

Thomas Wentworth
31 4

I have been using this code in my work and I noticed a problem. An eps value of 1e-15 seems to be too small. Notably, consider the matrix A = np.ones(13,2). This code will report that this matrix has a rank 0 null space. This is due to the scipy.linalg.svd function reporting that the second singular value is above 1e-15. I don't know much about the algorithms behind this function, however I suggest using eps=1e-12 (and perhaps lower for very large matrices) unless someone with more knowledge can chime in. (In infinite precision the second singular value should be 0). – Thomas Wentworth Dec 19 '14 at 16:55

A faster but less numerically stable method is to use a rank-revealing QR decomposition, such as `scipy.linalg.qr` with `pivoting=True` :

```
import numpy as np
from scipy.linalg import qr

def qr_null(A, tol=None):
    Q, R, P = qr(A.T, mode='full', pivoting=True)
    tol = np.finfo(R.dtype).eps if tol is None else tol
    rnk = min(A.shape) - np.abs(np.diag(R))[::-1].searchsorted(tol)
    return Q[:, rnk:].conj()
```

For example:

```
A = np.array([[ 2, 3, 5],
              [-4, 2, 3],
              [ 0, 0, 0]])
Z = qr_null(A)

print(A.dot(Z))
#[[  4.44089210e-16]
# [  6.66133815e-16]
# [  0.00000000e+00]]
```

edited Nov 10 '15 at 2:19

answered Nov 9 '15 at 22:15

ali_m
24.4k 4 51 100