In [148]:
```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
```

In [14]:
```
training = pd.read_csv('training.csv')
```

In [15]:
```
training.head()
```

Out[15]:

	first	second	third	label
0	3.852831	-4.096736	-2.165801	0.0
1	-1.386546	1.499733	-4.099774	0.0
2	-4.556334	-3.665592	2.501289	0.0
3	-3.110257	4.206114	3.859427	0.0
4	-0.689583	-0.305288	2.745201	0.0

In [16]:
```
print(training.columns)
first = training['first']
second = training['second']
third = training['third']
label = training['label']
```

```
Index(['first', 'second', 'third', 'label'], dtype='object')
```

In [30]:
```
test = pd.read_csv('test.csv')
first2 = test['first']
second2 = test['second']
third2 = test['third']
label2 = test['label']
```

In [31]:
```
first[0]
```

Out[31]:
```
3.8528309999999997
```

In [32]:
```
points = list(zip(first, second, third))
points2 = list(zip(first2, second2, third2))
```

In [139]:
```
d = [1, 2, 3, 5]
C_arr = [0.0001, 0.001, 0.01, 0.1, 1]
```

In [140]:
```
arr = []
for k in range(5):
    arr.append(list(np.zeros(4)))
arr2 = []
for k in range(5):
    arr2.append(list(np.zeros(4)))
```

In [141]:
```
arr2
```

Out[141]:
```
[[0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0]]
```

In [142]:
```
for i in range(5):
    for j in range(4):
        clf = SVC(C = C_arr[i], degree = d[j])
        clf.fit(points, label)
        arr[i][j] = 1 - clf.score(points, label)
        arr2[i][j] = 1 - clf.score(points2, label2)
```

In [143]:
```
arr
```

Out[143]:
```
[[0.088999999999999968,
  0.088999999999999968,
  0.088999999999999968,
  0.088999999999999968],
 [0.088999999999999968,
  0.088999999999999968,
  0.088999999999999968,
  0.088999999999999968],
 [0.088999999999999968,
  0.088999999999999968,
  0.088999999999999968,
  0.088999999999999968],
 [0.088999999999999968,
  0.088999999999999968,
  0.088999999999999968,
  0.088999999999999968],
 [0.021000000000000019,
  0.021000000000000019,
  0.021000000000000019,
  0.021000000000000019]]
```

In [144]:
```
arr2
```

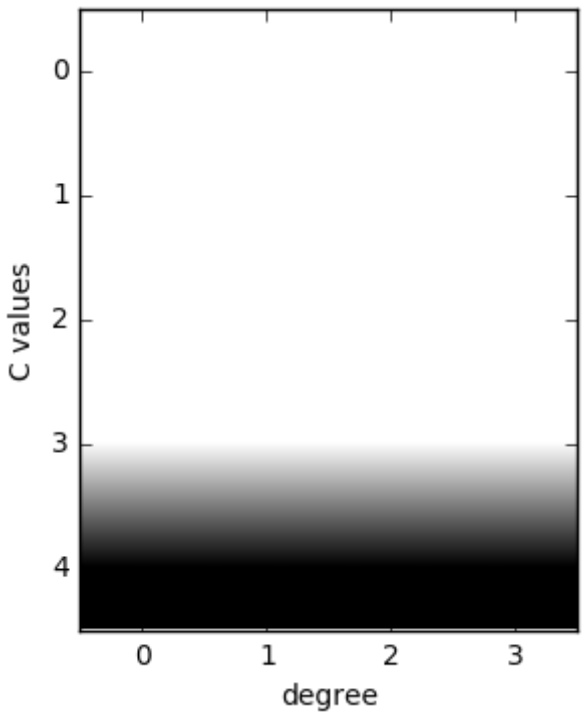
Out[144]:
```
[[0.44166666666666665,
  0.44166666666666665,
  0.44166666666666665,
  0.44166666666666665],
 [0.44166666666666665,
  0.44166666666666665,
  0.44166666666666665,
  0.44166666666666665],
 [0.44166666666666665,
  0.44166666666666665,
  0.44166666666666665,
  0.44166666666666665],
 [0.44166666666666665,
  0.44166666666666665,
  0.44166666666666665,
  0.44166666666666665],
 [0.44133333333333336,
  0.44133333333333336,
  0.44133333333333336,
  0.44133333333333336]]
```

In [ ]:
```

```

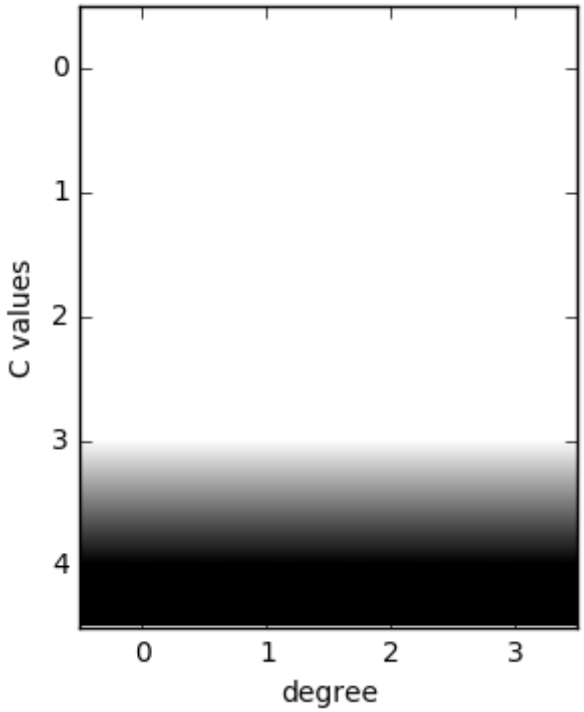
In [145]:
```
print("Training errors:") #because plt.title with xticks doesn't work?
plt.imshow(arr, interpolation = None, cmap = 'gray')
plt.xticks([0, 1, 2, 3])
plt.xlabel("degree")
plt.ylabel("C values")
plt.show()
```

```
Training errors:
```



In [146]:
```
print("Test errors:")
plt.imshow(arr2, interpolation = None, cmap = 'gray')
plt.xticks([0, 1, 2, 3])
plt.xlabel("degree")
plt.ylabel("C values")
plt.show()
```

```
Test errors:
```



In [147]:
```
for x, y in zip(C_arr, range(5)):
    print(str(x) + " is represented by " + str(y))
```

```
0.0001 is represented by 0
0.001 is represented by 1
0.01 is represented by 2
0.1 is represented by 3
1 is represented by 4
```

In [92]:
```
for x, y in zip(d, range(4)):
    print(str(x) + " is represented by " + str(y))
```

```
1 is represented by 0
2 is represented by 1
3 is represented by 2
5 is represented by 3
```

In [100]:
```
On what basis would you decide that a hyperparameter setting is optimal? Whi
```

```
Object `results` not found.
```

1. On what basis would you decide that a hyperparameter setting is optimal? Which setting of (C, d) gave the optimal results?

The hyperparameter setting is optimal when C is 0.1 and 1. The choice of d did not matter.

2.You'll notice that between C and d, one factor mattered far more than the other. What can you conclude about the structure of the datasets and how they were generated?

```
C mattered far more than d. The choice of 0.1 and 1 for C suggests that a
higher penalty for misclassified points was necessary, implying that the
data probably doesn't have much variation to begin with (i.e. easier to
classify into sets).
```

3.With as much granularity as possible, which hyperparameter settings are underfitting and which are overfitting? What allows you to make this claim?

```
The C settings are underfitting when C is less than 0.1, since there is a
less strict penalty for misclassified points (allows for more variation
than we care about). All of
```

In [106]:
```
gammas = [0.001, 0.001, 0.1, 1, 10, 100, 1000]
```

In [165]:
```
errors = []
```

In [166]:
```
for g in gammas:
    clf = SVC(gamma = g)
    clf.fit(points, label)
    errors.append([1- score for score in cross_val_score(clf, points, label,
```

In [167]:
```
errors
```

Out[167]:
```
[[0.09027777777777779,
  0.090909090909090939,
  0.090909090909090939,
  0.090909090909090939,
  0.090909090909090939,
  0.084507042253521125,
  0.084507042253521125],
 [0.09027777777777779,
  0.090909090909090939,
  0.090909090909090939,
  0.090909090909090939,
  0.090909090909090939,
  0.084507042253521125,
  0.084507042253521125],
 [0.03472222222222221,
  0.055944055944055937,
  0.013986013986013957,
  0.048951048951048959,
  0.041958041958041981,
  0.021126760563380254,
  0.028169014084507005],
 [0.05555555555555558,
  0.062937062937062915,
  0.041958041958041981,
  0.069930069930069894,
  0.041958041958041981,
  0.042253521126760618,
  0.028169014084507005],
 [0.09027777777777779,
  0.090909090909090939,
  0.083916083916083961,
  0.090909090909090939,
  0.090909090909090939,
  0.077464788732394374,
  0.077464788732394374],
 [0.09027777777777779,
  0.090909090909090939,
  0.090909090909090939,
  0.090909090909090939,
  0.090909090909090939,
  0.084507042253521125,
  0.084507042253521125],
 [0.09027777777777779,
  0.090909090909090939,
  0.090909090909090939,
  0.090909090909090939,
  0.090909090909090939,
  0.084507042253521125,
  0.084507042253521125]]
```

In [168]:
```
gamma_table = pd.DataFrame()
pairs = list(zip(gammas, errors))
```

In [169]:
```
for g,e in pairs:
    gamma_table[g] = e
```

In [170]:
```
gamma_table
```

Out[170]:

	0.001	0.1	1.0	10.0	100.0	1000.0
0	0.090278	0.034722	0.055556	0.090278	0.090278	0.090278
1	0.090909	0.055944	0.062937	0.090909	0.090909	0.090909
2	0.090909	0.013986	0.041958	0.083916	0.090909	0.090909
3	0.090909	0.048951	0.069930	0.090909	0.090909	0.090909
4	0.090909	0.041958	0.041958	0.090909	0.090909	0.090909
5	0.084507	0.021127	0.042254	0.077465	0.084507	0.084507
6	0.084507	0.028169	0.028169	0.077465	0.084507	0.084507

In [115]:
```

```

Out[115]:
```
[(0.001, 0.088999999999999968),
 (0.001, 0.088999999999999968),
 (0.1, 0.02200000000000002),
 (1, 0.014000000000000012),
 (10, 0.0030000000000000027),
 (100, 0.0),
 (1000, 0.0)]
```

In [ ]:
```

```