Over at NHLnumbers.com, Josh W has done some work to determine how much of the outcome of an NHL game is determined by "luck." I think this is a pretty great read and you should go check it out (http://nhlnumbers.com/2013/8/6/theoretical-predictions-in-machine-learning-for-the-nhl-part-ii). His definition of luck is derived from the following logic. The distribution of win-loss records in the NHL seems to be Gaussian and it can be compared to two other distributions - the "pure luck" distribution in which teams win completely at random and the "pure skill" distribution in which teams win according to a predetermined ranking. If you look at Josh's article you'll see exactly what I mean. Josh then builds a model that basically attempts to recreate a Gaussian that matches the NHL's distribution that mixes the luck and the skill distributions.

The short story here is that while I really liked his work, I think a better definition of luck makes sense. His definition is essentially as follows: once the above model is fit to the NHL data, the factor of that blend which comes from the luck distribution should be the amount that NHL games actually depend on luck. I think there are a few problems with this. For one thing, his model implies that the #1 team and the #29 team have the same chance of beating the #30 team but teams don't exhibit this type of parity. So, while this is a successful model on a population level for generating the appropriate win-loss records, I don't think it tells the whole story.

I think a better way of describing luck relies on observing how the best teams fare against the worst teams on the spectrum of skill (always win) to luck (50-50). With that in mind, I'm going to take the win-loss data since 2000-2001 and crunch it according to how times finished the regular season.

In [4]:
```
import pandas as pd
import matplotlib
from sklearn import linear_model
from os import getcwd
```

Here's our data loading. Not efficient but sufficient.

In [7]:
```
gamesDFs = []
standingsDFs = []
yearList = ['01','02','03','04','06','07','08','09','10','11','12','13']
for year in yearList:
    gamesDFs.append(pd.read_csv(getcwd()+'/data/games'+year+'.csv', parse_
    standingsDFs.append(pd.read_csv(getcwd()+'/data/standings'+year+'.csv'
```

Here's a little transformation that puts converts the dictionary of standings into a dict of the form ranks[year_index][team] = rank in final standings. We have to spend some effort standardizing the team names.

In [8]:
```
ranks = []
for standings in standingsDFs:
    rank = {}
    for s in standings:
        standings[s] = standings[s].split('\xa0')[-1].lower()
        if standings[s][:5] == 'montr':
            standings[s] = 'montreal'
        rank[standings[s]] = s
    ranks.append(rank)

# test
ranks[9]['washington']
```

Out[8]: 1

Filling in the game results year-by-year.

In [27]:
```
gameResults = []
for year in gamesDFs:
    X = year[year['.'].isnull()][['Home Team','Away Team','Score','Score.1
    X['HomeWin'] = X['Score'] > X['Score.1']
    gameResults.append(X)

#test
gameResults[0][:10]
```

Out[27]:

	Home Team	Away Team	Score	Score.1	HomeWin
1	Phoenix	St. Louis	4	1	True
2	Calgary	Detroit	3	4	False
3	Philadelphia	Vancouver	6	3	True
4	Buffalo	Chicago	4	2	True
6	San Jose	St. Louis	1	4	False
7	Anaheim	Minnesota	3	1	True
8	Edmonton	Detroit	2	1	True
11	Washington	Los Angeles	1	4	False
12	New Jersey	Montreal	8	4	True
13	Phoenix	Minnesota	4	1	True

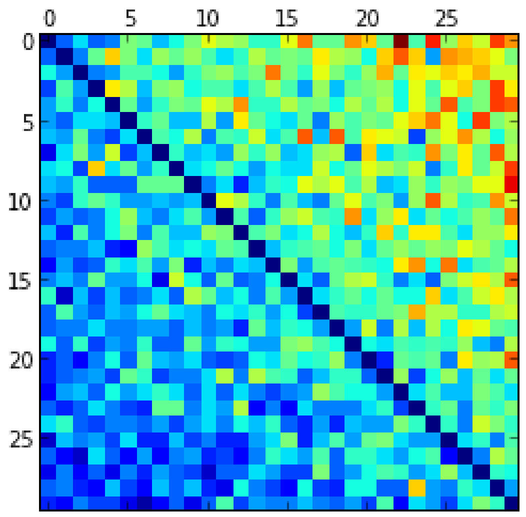
Now we build an array that records how the teams did over all observed seasons, stripping away the team identity and merging them according to how they finished the season (ie 1st to 30th).

In [28]:
```
teamArray = np.zeros((30,30))
for year in range(len(gameResults)):
    for record in gameResults[year].values:
        team1 = ranks[year][record[0].lower()]
        team2 = ranks[year][record[1].lower()]
        if record[-1]:
            winner = team1
            loser =  team2
        else:
            winner = team2
            loser =  team1
        teamArray[winner,loser]+=1
```

Here's a matrix plot of how the various teams fared against each other. The top row shows how the best team each year fared against the 2nd through 30th teams in terms of total wins or losses.

In [30]:
```
pylab.matshow(teamArray)
```

Out[30]:
```
<matplotlib.image.AxesImage at 0x64592f0>
```

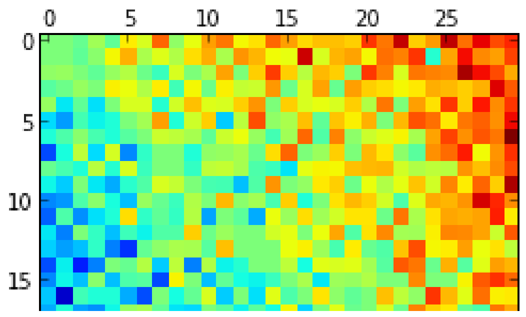


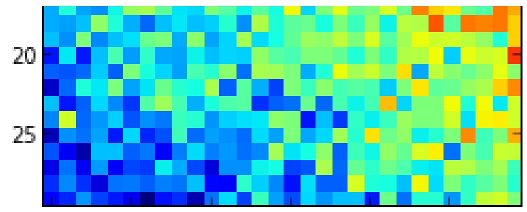
We now replace total wins / losses with winning percentage. ratioArray[team1,team2] = team1's winning % vs team2.

In [31]:
```
ratioArray = 0.5*np.ones((30,30))
for i in range(30):
    for j in range(0,30):
        if j != i:
            ratioArray[i,j] = teamArray[i,j]/(teamArray[j,i]+teamArray[i,j
```

In [32]:
```
pylab.matshow(ratioArray)
```

Out[32]:
```
<matplotlib.image.AxesImage at 0x64fd390>
```

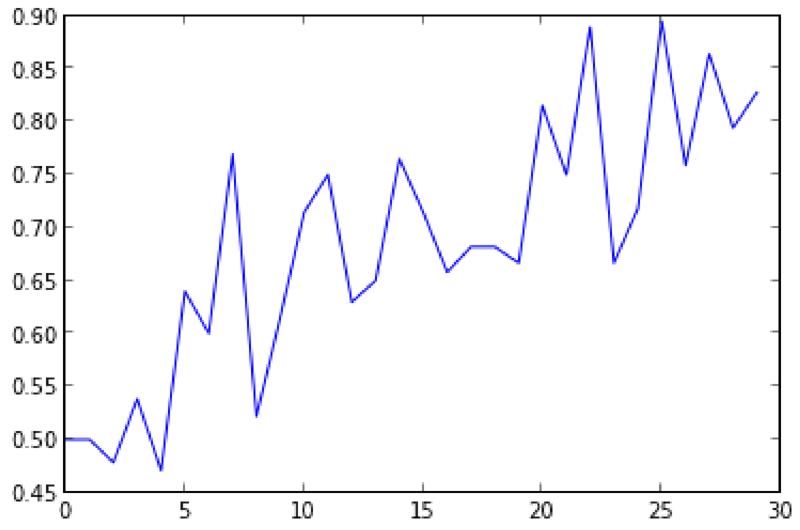




This is the actual winning %s of the best team each year against every other team, sorted by end-of-season standings.

In [33]:
```
plot(ratioArray[0])
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x60a5e70>]
```



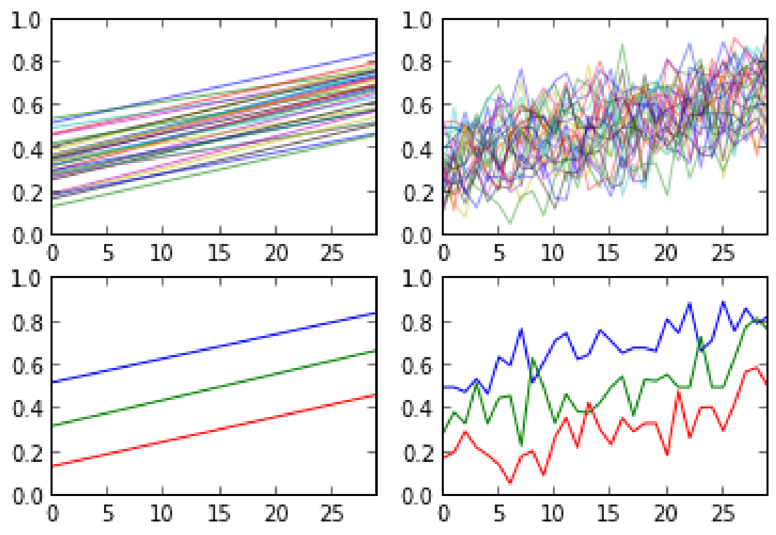
We fit a linear model to each rank's data.

In [34]:
```
X = arange(30).reshape(30,1)
models = []
for team in ratioArray:
    lrmodel = linear_model.LinearRegression()
    lrmodel.fit(X,team)
    models.append(lrmodel.predict(X))
```

In [37]:
```
pyplot.figure(1)

pyplot.subplot(221)
pyplot.axis([0,29,0,1])
for i in models:
    plot(i,alpha=0.6)

pyplot.subplot(222)
pyplot.axis([0,29,0,1])
for i in range(30):
    plot(ratioArray[i],alpha=0.4)
```

```
pyplot.subplot(223)
pyplot.axis([0,29,0,1])
plot(models[0])
plot(models[15])
plot(models[29])

pyplot.subplot(224)
pyplot.axis([0,29,0,1])
plot(ratioArray[0])
plot(ratioArray[15])
plot(ratioArray[29])
```

Out[37]:
```
[<matplotlib.lines.Line2D at 0x6a2e350>]
```

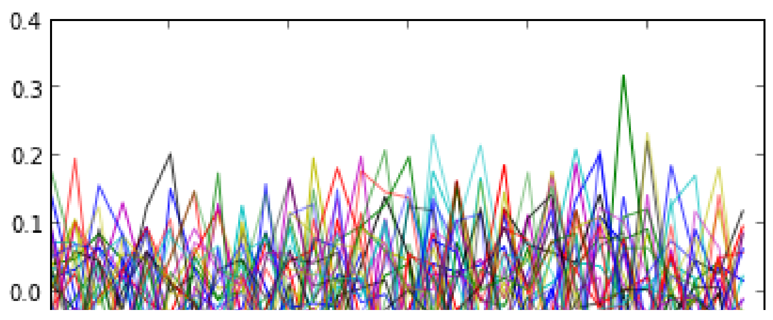


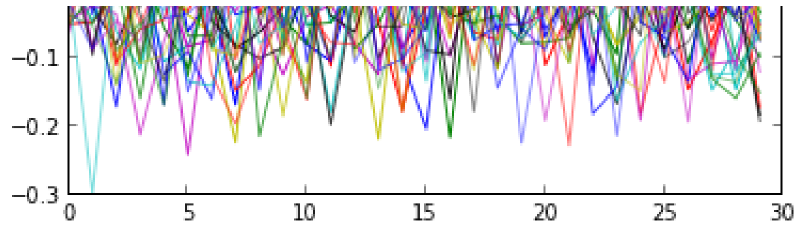
So how much do the data deviate from the models?

In [17]:
```
predicted = array(models)
deviation = (predicted - ratioArray)
mean(abs(deviation))
```

Out[17]:
```
0.071525572902002441
```

In [20]:
```
pyplot.figure(1)
for i in range(30):
    plot(deviation[i],alpha=(30-i)*0.02+0.4)
```

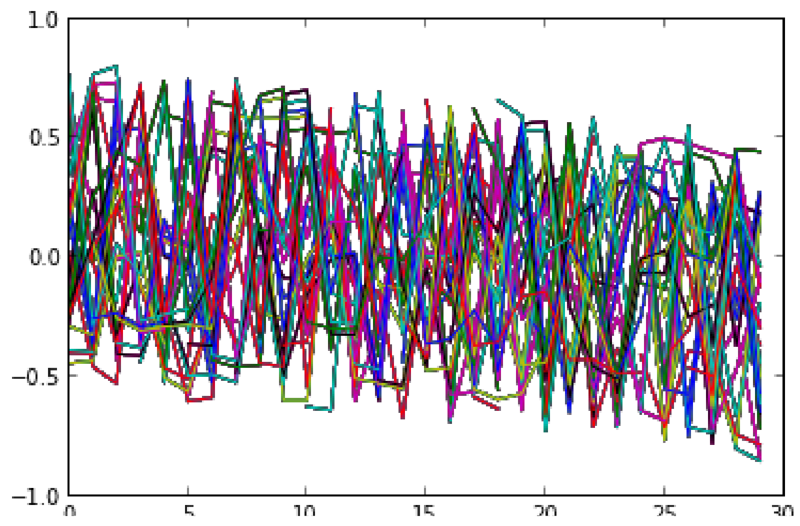




This is over the course of 11.5 seasons so there's a lot of smoothing happening. Let's try to take a look at a single year.

In [38]:
```python
results09 = np.zeros((30,30))

for record in gameResults[7].values:
    team1 = ranks[9][record[0].lower()]
    team2 = ranks[9][record[1].lower()]
    if record[-1]:
        winner = team1
        loser =  team2
    else:
        winner = team2
        loser =  team1
    results09[winner,loser]+=1

    ratio09 = 0.5*np.ones((30,30))

for i in range(30):
    for j in range(0,30):
        if j != i:
            ratio09[i,j] = results09[i,j]/(results09[j,i]+results09[i,j])
```

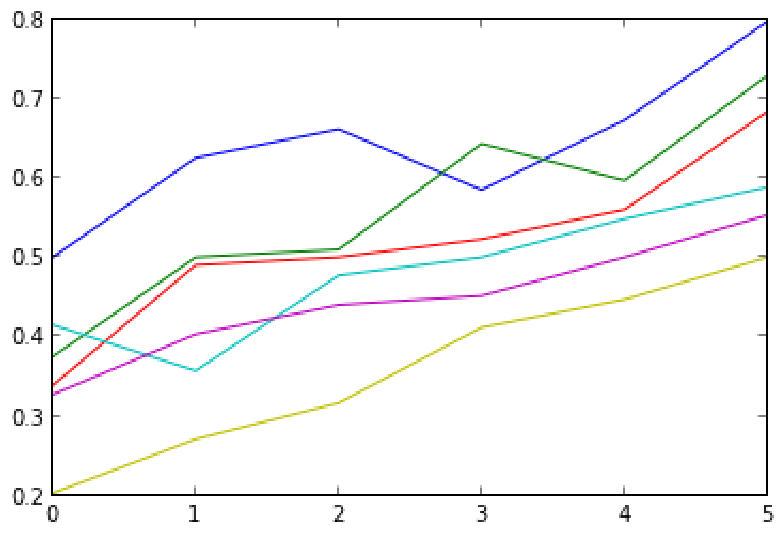
In [39]:
```python
pyplot.figure(2)
deviation09 = predicted - ratio09
for i in range(30):
    plot(deviation09)
```



Yikes! Still a lot of the craziness comes from playing some teams only once or twice - huge variance there. Teams should not be too sensitive to the difference between say the #1 team and the #2 team, so let's try to bin these teams and see how the data looks.

In [40]:
```
results09b = np.zeros((6,6))

for record in gameResults[7].values:
    team1 = ranks[7][record[0].lower()]
    team2 = ranks[7][record[1].lower()]
    if record[-1]:
        winner = team1
        loser =  team2
    else:
        winner = team2
        loser =  team1
    results09b[winner//5,loser//5]+=1

ratio09b = 0.5*np.ones((6,6))

for i in range(6):
    for j in range(6):
        if j != i:
            ratio09b[i,j] = results09b[i,j]/(results09b[j,i]+results09b[i,
```

In [41]:
```
pyplot.figure(1)
for i in range(6):
    plot(ratio09b[i])
```



It seems that you can say that at this coarse level teams perform about as well as expected.

In [ ]: