Neuroinformatics Toolbox Documentation

Release 0.1

Jacob Huth

June 14, 2013

CONTENTS

The NI Toolbox contains python versions of commonly used functions to deal with spike data. Get the pdf version here

CHAPTER

ONE

FRAGEN

- **splines** - was für constraints müssen erfüllt sein?
- Welche Modelle sind sinnvoll zu vergleichen für generieren und fitten?

Contents:

1.1 Tutorial

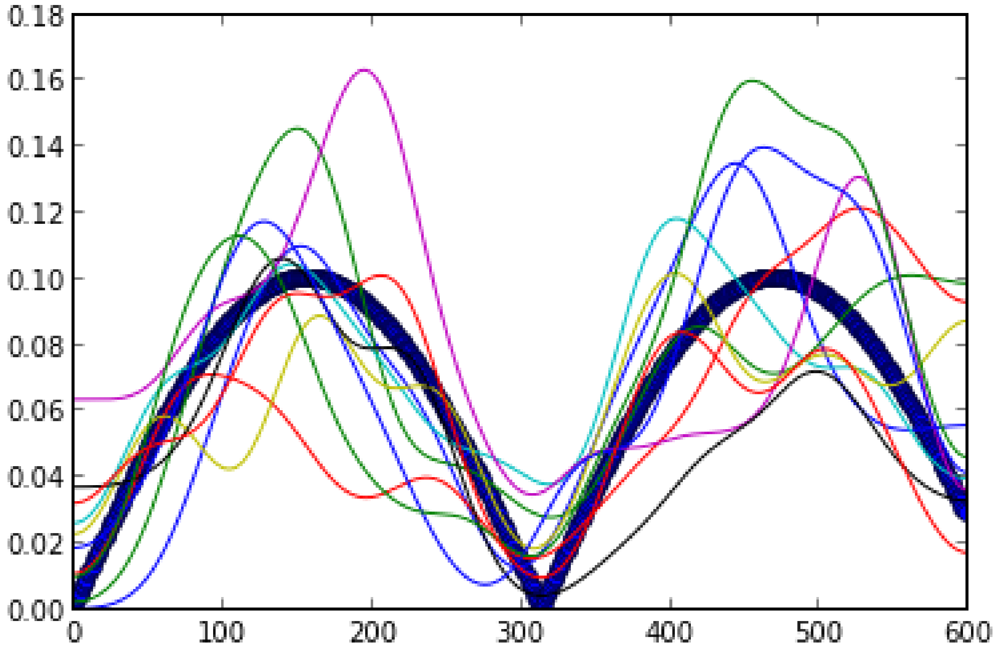
Something here

1.2 Some Examples

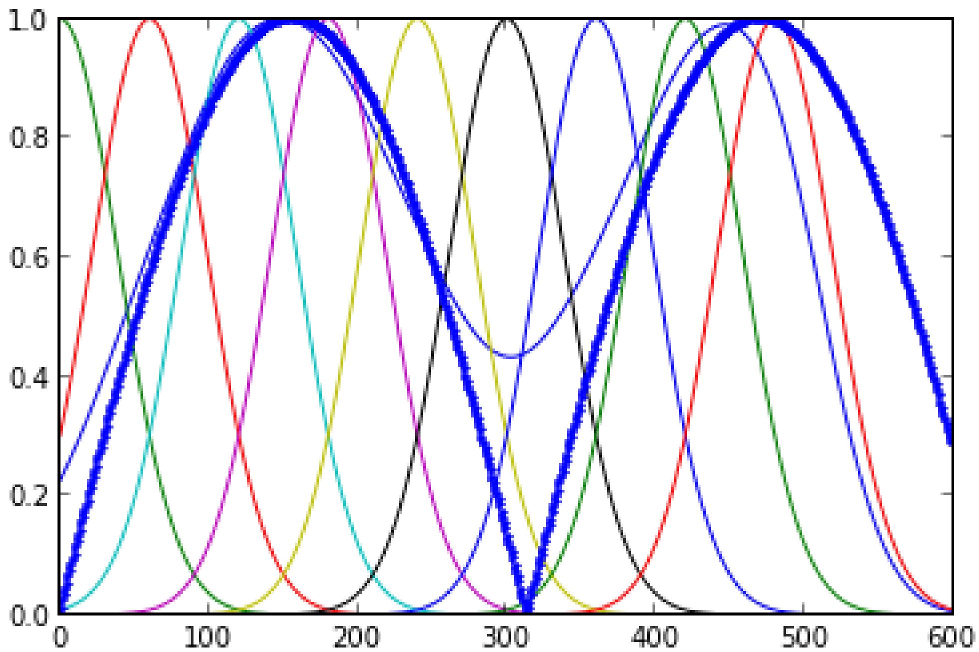
This code can be executed in an **ipython notebook –pylab inline** in browser maltlab like environment.

```
cd uni/MT/EIC/py

import pandas as pd
import ni.model.pointprocess
reload(ni.model.pointprocess)
p1 = ni.model.pointprocess.createPoisson(0.1,1000)

p2 = ni.model.pointprocess.createPoisson(sin(numpy.array(range(0,200))*0.01)*0.5- 0.2,1000)

import matplotlib
matplotlib.rcParams['savefig.dpi'] = 72

from scikits.statsmodels.genmod import generalized_linear_model
from scikits.statsmodels.genmod.families.family import Binomial

spikes = np.array(ni.model.pointprocess.createPoisson(sin(numpy.array(range(0,200))*0.01)*0.5- 0.2,1

design_matrix = [[exp(-1*(((i-j*100))**2)/1000) for j in range(0,9)] for i in linspace(0,999,1000)]
glm = generalized_linear_model.GLM(spikes,design_matrix, family = Binomial())
res = glm.fit()

import ni.model.glm as glm
import ni.tools.bootstrap as bs
reload(glm)
reload(bs)
```

```
length = 600

target = abs(sin(numpy.array(range(0,length))*0.01)*0.1)
spikes = np.array([ni.model.pointprocess.createPoisson(target,length).getCounts() for i in range(0,1
#design_matrix = [[exp(-1*(((i-j*60))**2)/10000) for j in range(0,9)] for i in linspace(0,length-1,le
design_matrix = [[exp(-1*(((i-j*60))**2)/3000) for j in range(0,9)] for i in linspace(0,length-1,leng
design_matrix2 = []
for i in range(0,10):
    design_matrix2.append(design_matrix)
design_matrix = vstack(design_matrix2)
plot(target,'bo')
for i in range(0,10):
    ni.model.pointprocess.plotGaussed(spikes[i],30)
#bs.evaluate(glm.GLM(spikes,design_matrix),spikes,10)
```




```
from scikits.statsmodels.genmod import generalized_linear_model
from scikits.statsmodels.genmod.families.family import Binomial

Data = reshape(spikes,(length * 10,1))
#plot(Data,'bo')
#plot(design_matrix[:,3])

dev = np.array([abs((design_matrix[find(Data.transpose()),i])) for i in range(0,9)])
prediction = sum(design_matrix * mean(dev,1),1)
plot(prediction[0:length]/max(prediction))
plot(design_matrix[0:length,:])
plot(target/max(target),'b.')
#g = generalized_linear_model.GLM(Data,design_matrix[:,1:5], family = Binomial())
#res = g.fit()

#model = glm.GLM(Data,design_matrix)
#m = model.fit(Data)
```




```
g = generalized_linear_model.GLM(Data,design_matrix, family = Binomial())
res = g.fit()
```

Note: Generates an ERROR

ValueError Traceback (most recent call last) /home/plogic/uni/MT/EIC/py/<ipython-input-298-79fc06ff0302> in <module>()

1 g = generalized_linear_model.GLM(Data,design_matrix, family = Binomial())

—-> 2 res = g.fit()

/usr/lib/pymodules/python2.7/scikits/statsmodels/genmod/generalized_linear_model.pyc in fit(self, maxiter, method, tol, scale)
404 wlsendog = eta + self.family.link.deriv(mu) * (self.endog-mu) 405 - offset

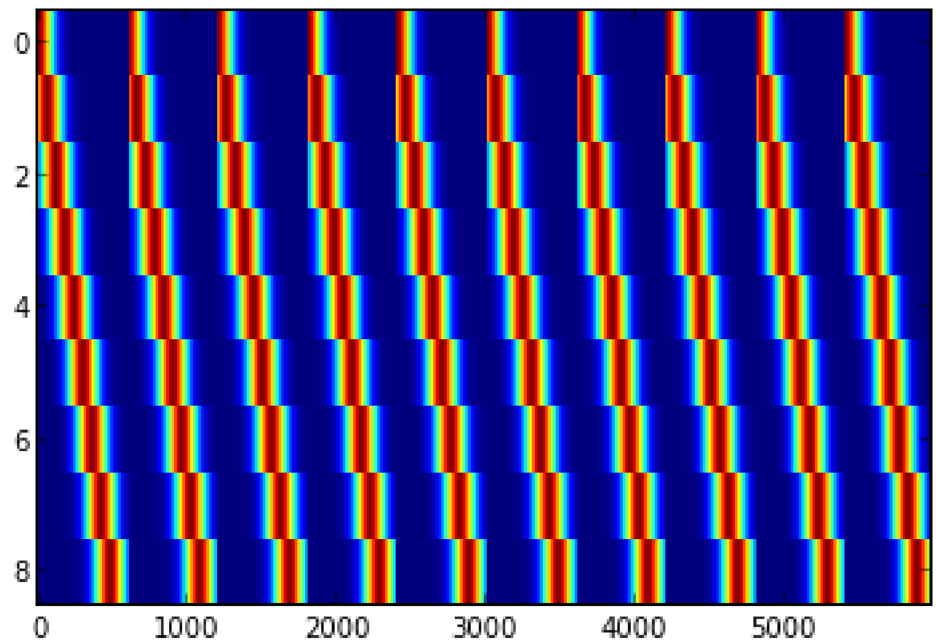
–> 406 wls_results = WLS(wlsendog, wlsexog, self.weights).fit() 407 eta = np.dot(self.exog, wls_results.params) + offset 408 mu = self.family.fitted(eta)

/usr/lib/pymodules/python2.7/scikits/statsmodels/regression/linear_model.pyc in __init__(self, endog, exog, weights)
383 weights.size == design_rows) : 384 raise ValueError(

–> 385 'Weights must be scalar or same length as design') 386 self.weights = weights.reshape(design_rows) 387 super(WLS, self).__init__(endog, exog)

ValueError: Weights must be scalar or same length as design

```
design_matrix = [[exp(-1*(((i-j*30))**2)/10000) for j in range(0,9)] for i in linspace(0,299,300)]
design_matrix2 = []
for i in range(0,10):
    design_matrix2.append(design_matrix)
#design_matrix2
design_matrix = vstack(design_matrix2)

img = imshow(np.array(design_matrix).transpose(), aspect='auto')
img.set_interpolation('nearest')
```




```
res.summary()
```

Note:

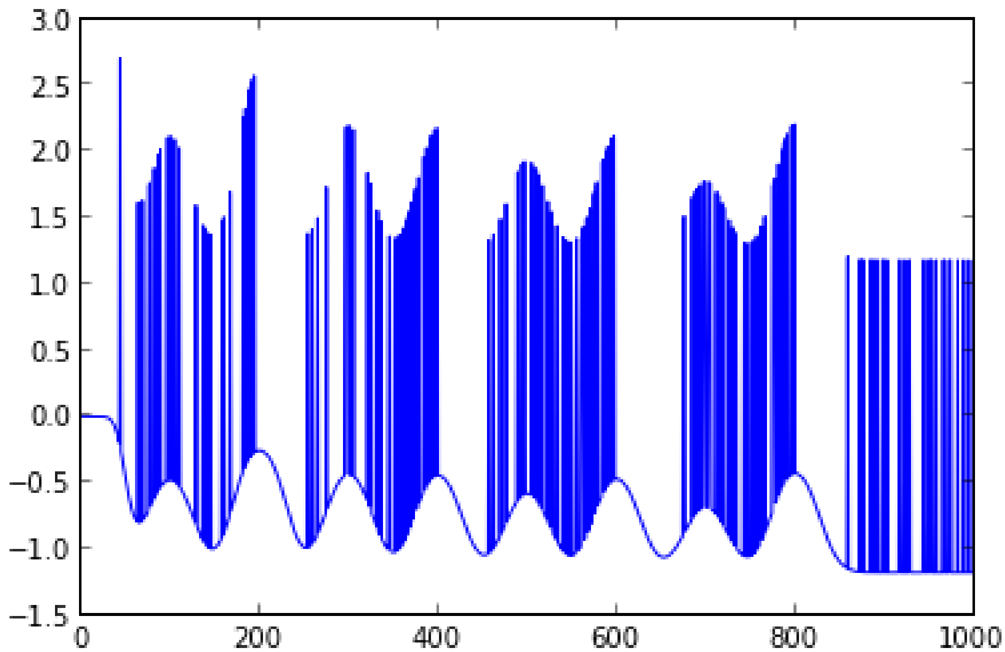
Generalized linear model

Model Family: Binomial # of obs: 1000 Method: IRLS Df residuals: 991 Dependent Variable: Y Df model: 8 Date: Thu, 30 May 2013 Scale: 1.0000 Time: 17:29:30 Log likelihood: -521.6848

coefficient stand errors t-statistic Conf. Interval

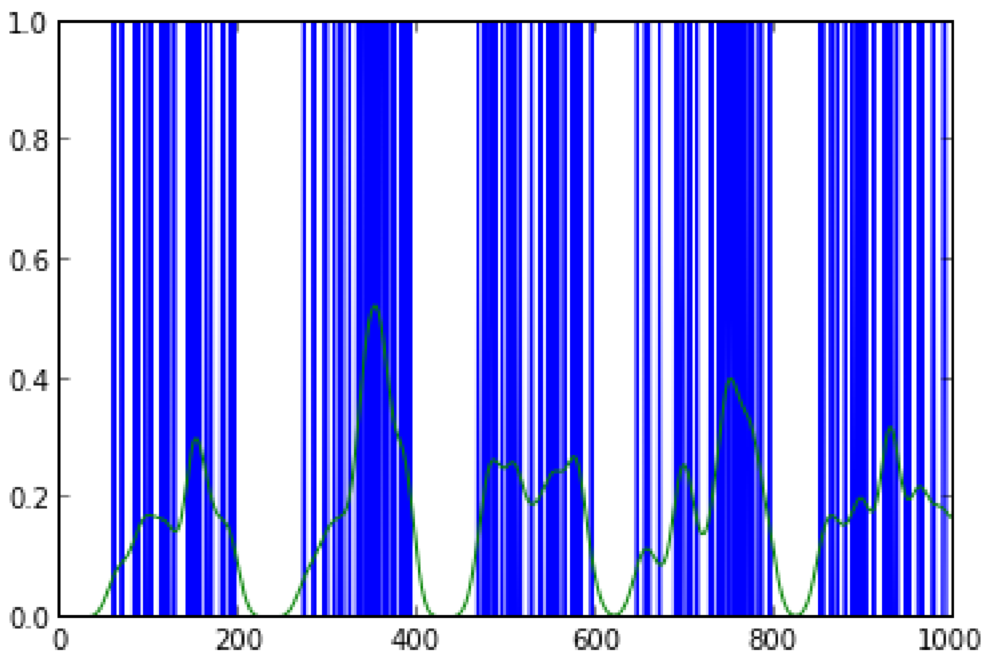
x0 -24.4281 -2.4888 [-43.666, -5.190] x1 -2.1134 -4.7164 [-2.992, -1.235] x2 -3.3877 -5.3690 [-4.624, -2.151] x3 -2.2992 -4.9481 [-3.210, -1.388] x4 -2.2641 -4.9220 [-3.166, -1.362] x5 -1.6907 -4.2519 [-2.470, -0.911] x6 -2.1377 -4.8080 [-3.009, -1.266] x7 -1.3351 -3.6306 [-2.056, -0.614] x8 -2.3364 -4.9854 [-3.255, -1.418]

```
plot(res.resid_deviance)
```

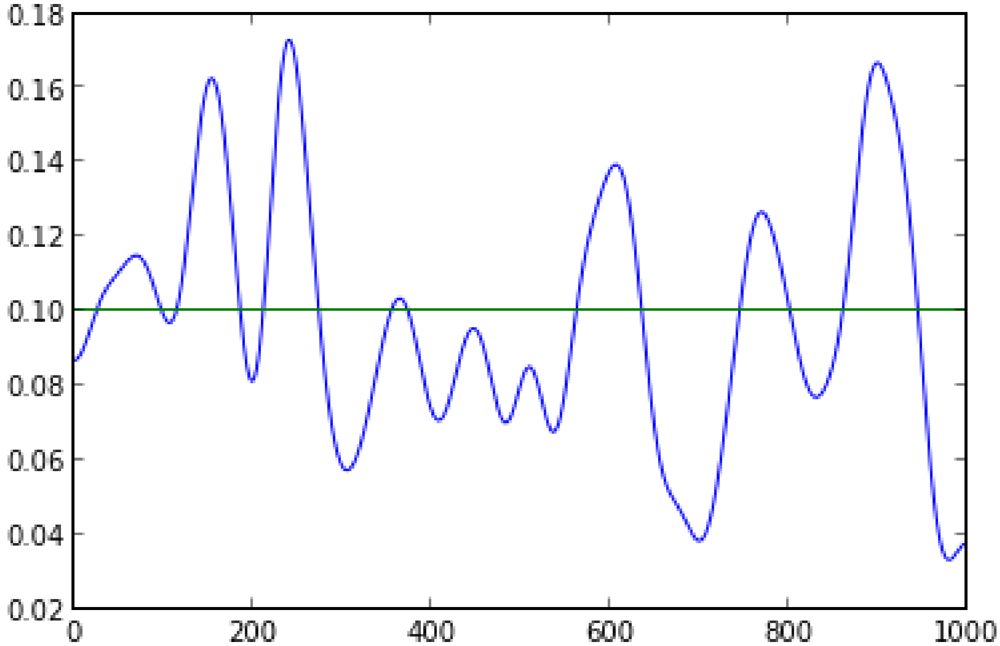



```
p2.plot()
```

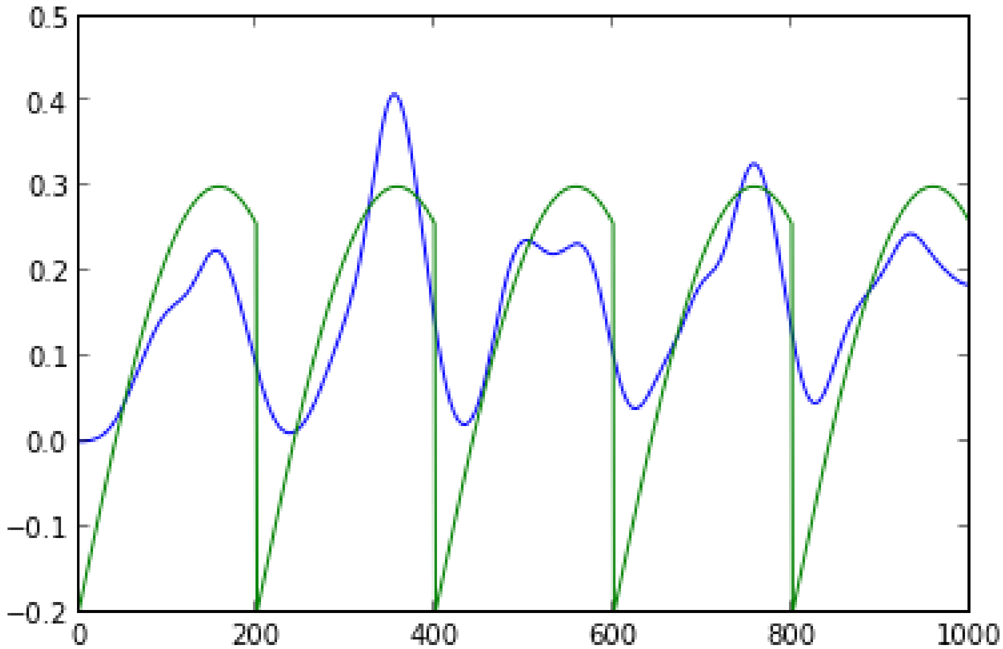
```
p2.plotGaussed(10)
```




```
p1.plotGaussed(20)
plot(p1.frate)
```

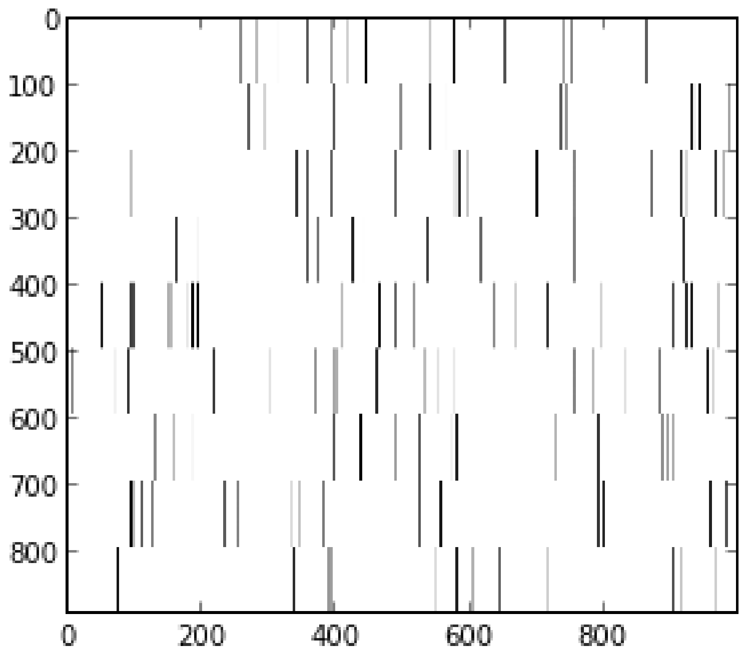



```
p2.plotGaussed(20)
plot(p2.frate)
```

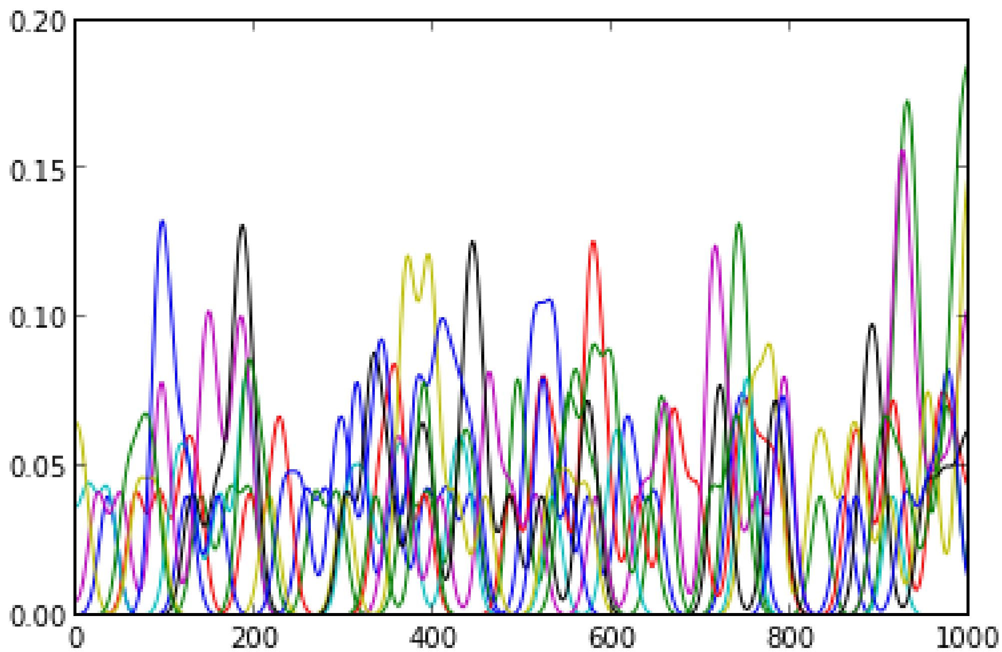



```
frate = (numpy.array(range(0,200))*0.001)*0.2+0.01
channels = 9

dists = [ni.model.pointprocess.createPoisson(frate,1000) for i in range(0,channels)]
#for i in range(0,9): dists[i].plotGaussed(10)
import itertools
spks = np.array([dists[i].getCounts() for i in range(0,channels) for j in range(0,99) ])
imshow(-1*spks)
```

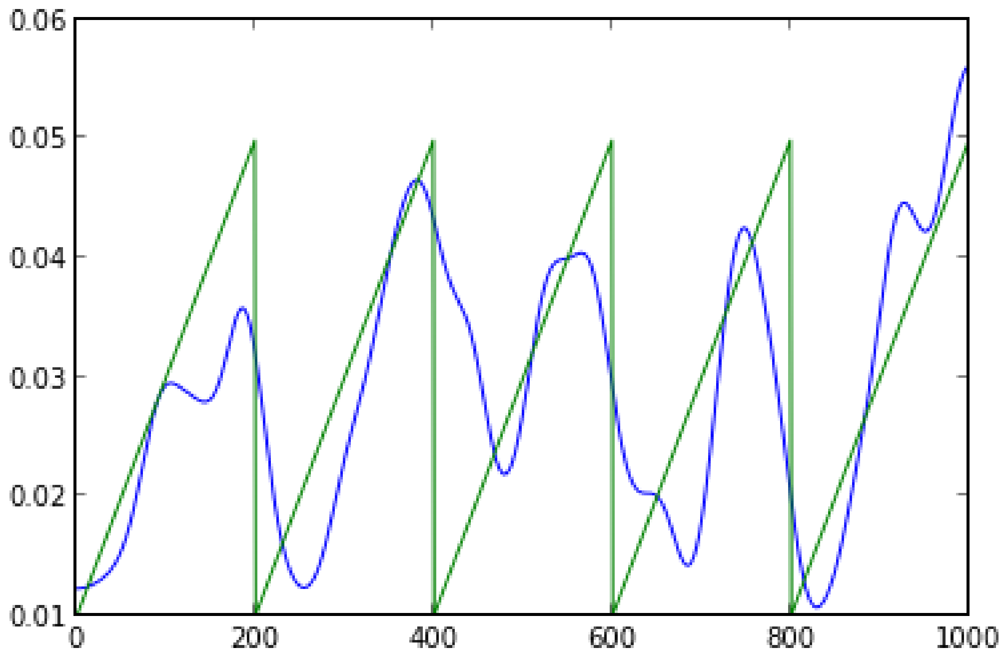
```
set_cmap('gray')
```



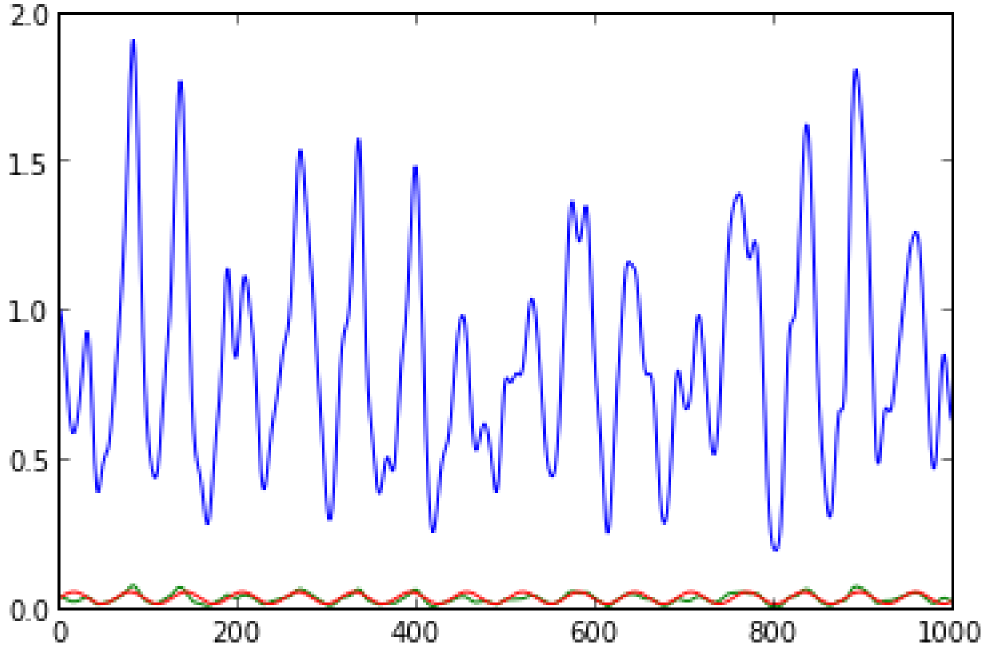
```
for i in range(0,channels): dists[i].plotGaussed(10)
```



```
ni.model.pointprocess.plotGaussed(np.array([dists[i].getCounts() for i in range(0,channels)]).mean(a
plot(dists[0].frate)
```

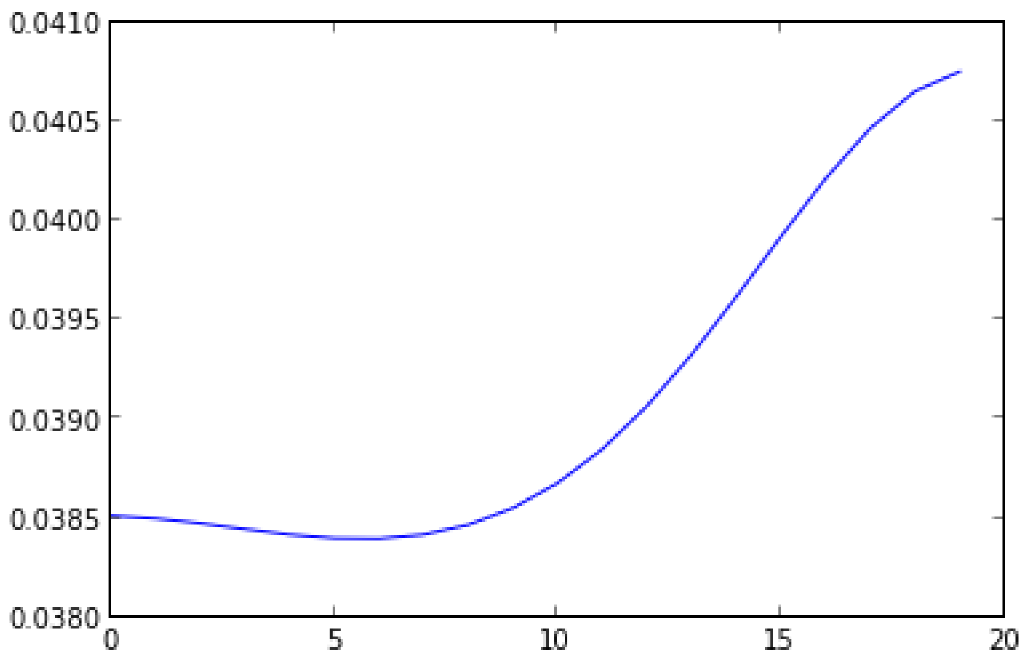



```
reload(ni.model.pointprocess)

frate = sin(numpy.array(range(0,1000))*0.1)*0.02+0.04
mc = ni.model.pointprocess.MultiChannelPointProcess(20)
mc.set(lambda x: ni.model.pointprocess.createPoisson(frate,1000))
#m = ni.model.pointprocess.createPoisson(0.1,1000)
#mc.set(lambda x: m)

ni.model.pointprocess.plotGaussed(mc.getMeanCounts(),5)
c = np.array(mc.get(lambda x: x.getCounts())).var(axis=0)

ni.model.pointprocess.plotGaussed(c,5)
plot(frate)
```



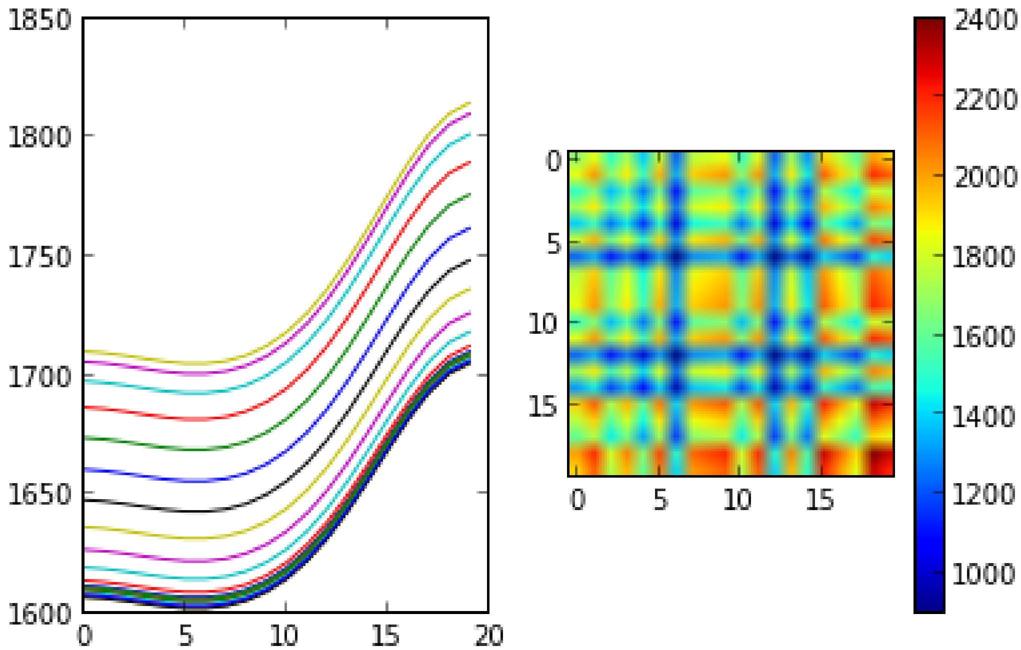

```
c = np.array(mc.get(lambda x: x.getCounts())).var(axis=1)
```

```
ni.model.pointprocess.plotGaussed(c,5)
```



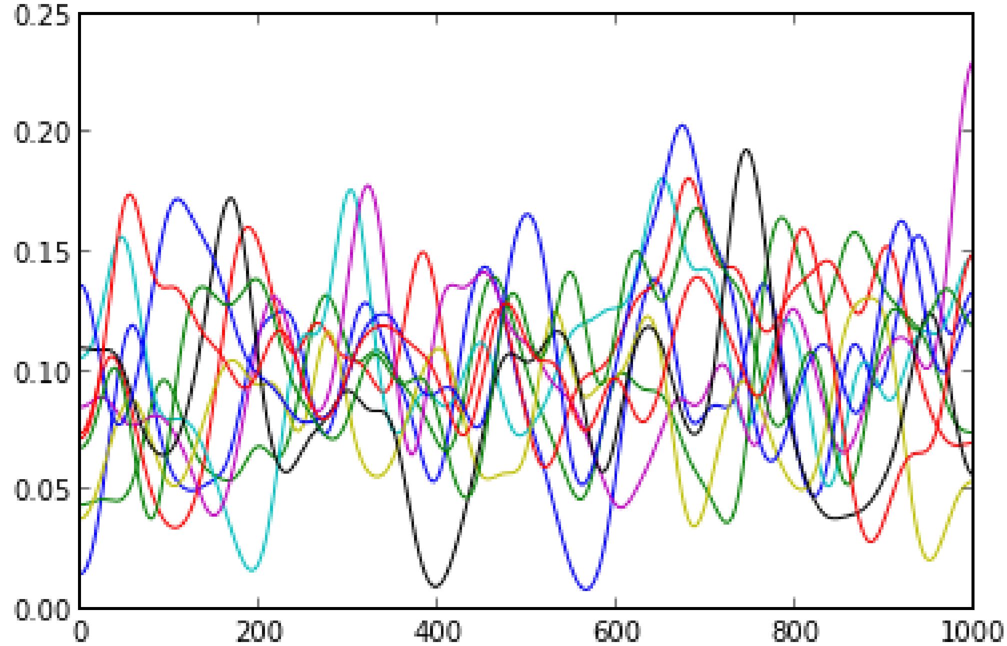

```
c2 = np.array(mc.get2(lambda x,y: np.convolve(x.getCounts(),y.getCounts()).sum()))
subplot(1,2,1)
ni.model.pointprocess.plotGaussed(c2,5)
subplot(1,2,2)

imshow(c2)
colorbar()
```

```
set_cmap('jet')
```

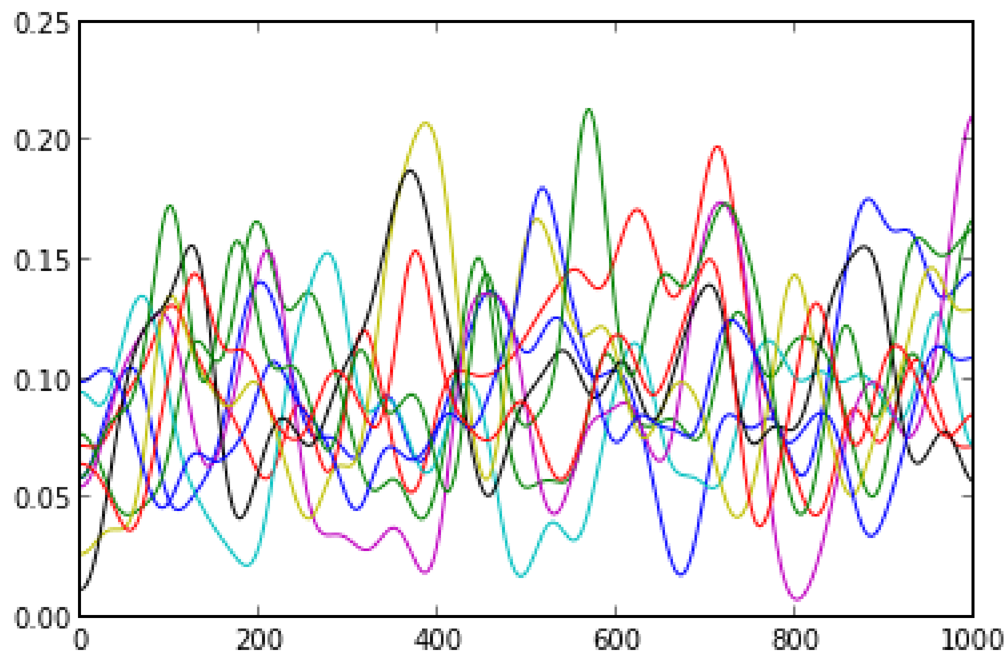



```
reload(ni.model.pointprocess)

ppc = ni.model.pointprocess.PPContainer()
ppc.setData([ni.model.pointprocess.createPoisson(0.1,1000).getCounts() for i in range(0,10)])
ppc.data.columns
[ni.model.pointprocess.plotGaussed(ppc.data[i],20) for i in range(0,10)]
```

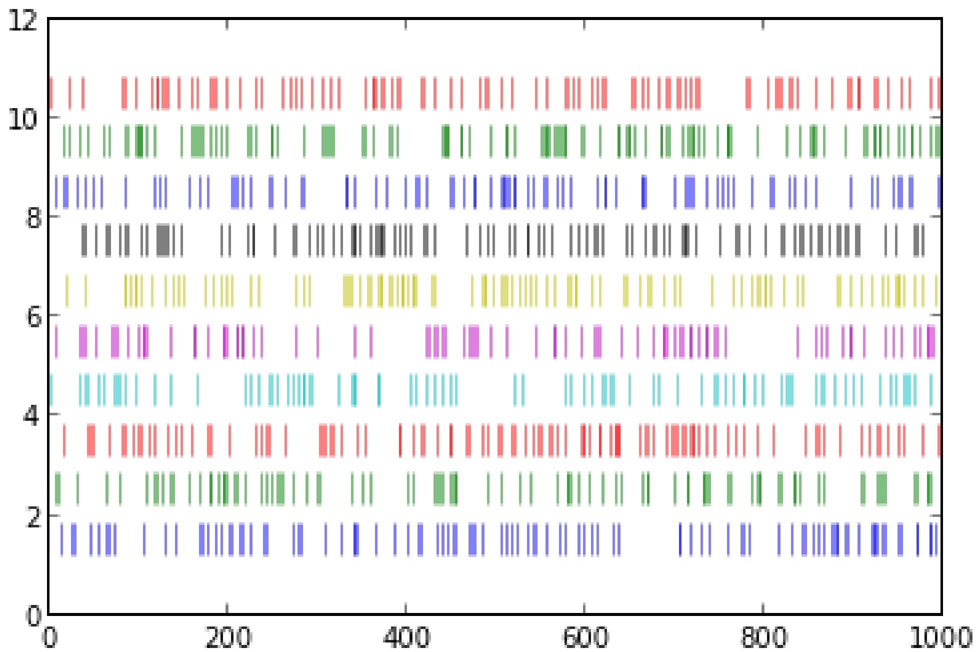
Note: [None, None, None, None, None, None, None, None, None, None]




```
cont = pd.DataFrame([ni.model.pointprocess.createPoisson(0.1,1000).getCounts() for i in range(0,10)])
[ni.model.pointprocess.plotGaussed(cont.T[i],20) for i in cont.T.columns]
```




```
ni.model.pointprocess.plotMultiSpikes(cont.T)
```

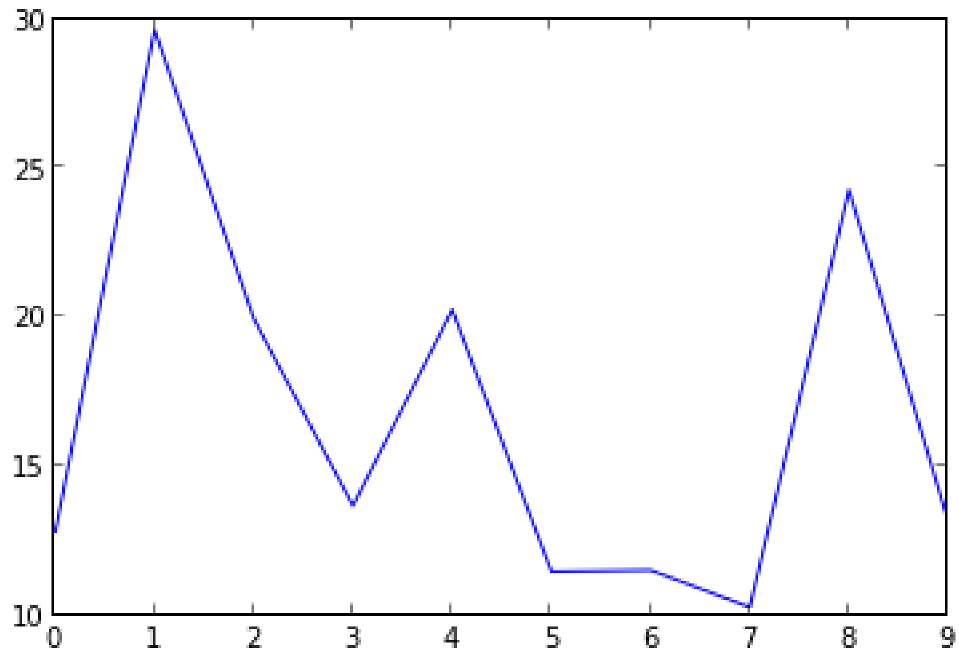



```
import ni.model.net_sim
reload(ni.model.net_sim)
c = ni.model.net_sim.SimulationConfiguration()
c.Nneur = 10
net = ni.model.net_sim.Net(c)
print net
net.plot_firing_rates()
```

Note: 'ni.model.net_sim' Simulation Setup Timerange: (250, 10250)

10 channels with firing rates: [12.815928361, 29.6328550796, 19.9415819867, 13.6710936491, 20.242131795, 11.4661487294, 11.5071338947, 10.2727521514, 24.2587596858, 13.1497981307]

Firing Rates plot




```
connections = [[abs(net.Jall[i,j,:].sum()) < 0.001 for i in range(0,10)] for j in range(0,10)]
#imgplot = imshow(connections)
#imgplot.set_interpolation('nearest')
import pydot
d = pydot.graph_from_edges(np.argwhere(connections), directed=True)
#d.create_png()
d.set_layout('neato')
d.set_mindist(10)
d.write_png("test2.png")

d2 = pydot.graph_from_edges(connections, directed=True)
#d.create_png()
d2.set_layout('neato')
d2.write_png("test1.png")

for i in range(1,11):
    print i
    res1 = net.simulate() #ni.model.net_sim.simulate(c)
    res1.plot_firing_rates()
    print res1
```

Note: 0 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 5.07s to compute. Timerange: (0, 10250) 2359 Spikes in 10 channels:

[242, 277, 131, 336, 326, 199, 260, 141, 233, 214]

1 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 5.32s to compute. Timerange: (0, 10250) 2305 Spikes in 10 channels:

[246, 329, 134, 299, 303, 199, 273, 123, 201, 198]

2 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 5.17s to compute. Timerange: (0, 10250) 2449 Spikes in 10 channels:

[251, 334, 125, 326, 324, 235, 261, 162, 214, 217]

3 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 9.69s to compute. Timerange: (0, 10250) 2436 Spikes in 10 channels:

[241, 343, 131, 310, 324, 210, 277, 131, 235, 234]

4 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 6.74s to compute. Timerange: (0, 10250) 2352 Spikes in 10 channels:

[251, 334, 121, 332, 306, 206, 278, 113, 225, 186]

5 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 5.3s to compute. Timerange: (0, 10250) 2342 Spikes in 10 channels:

[253, 332, 126, 325, 321, 216, 240, 114, 198, 217]

6 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 5.39s to compute. Timerange: (0, 10250) 2395 Spikes in 10 channels:

[263, 340, 121, 328, 305, 207, 268, 135, 216, 212]

7 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 5.72s to compute. Timerange: (0, 10250) 2340 Spikes in 10 channels:

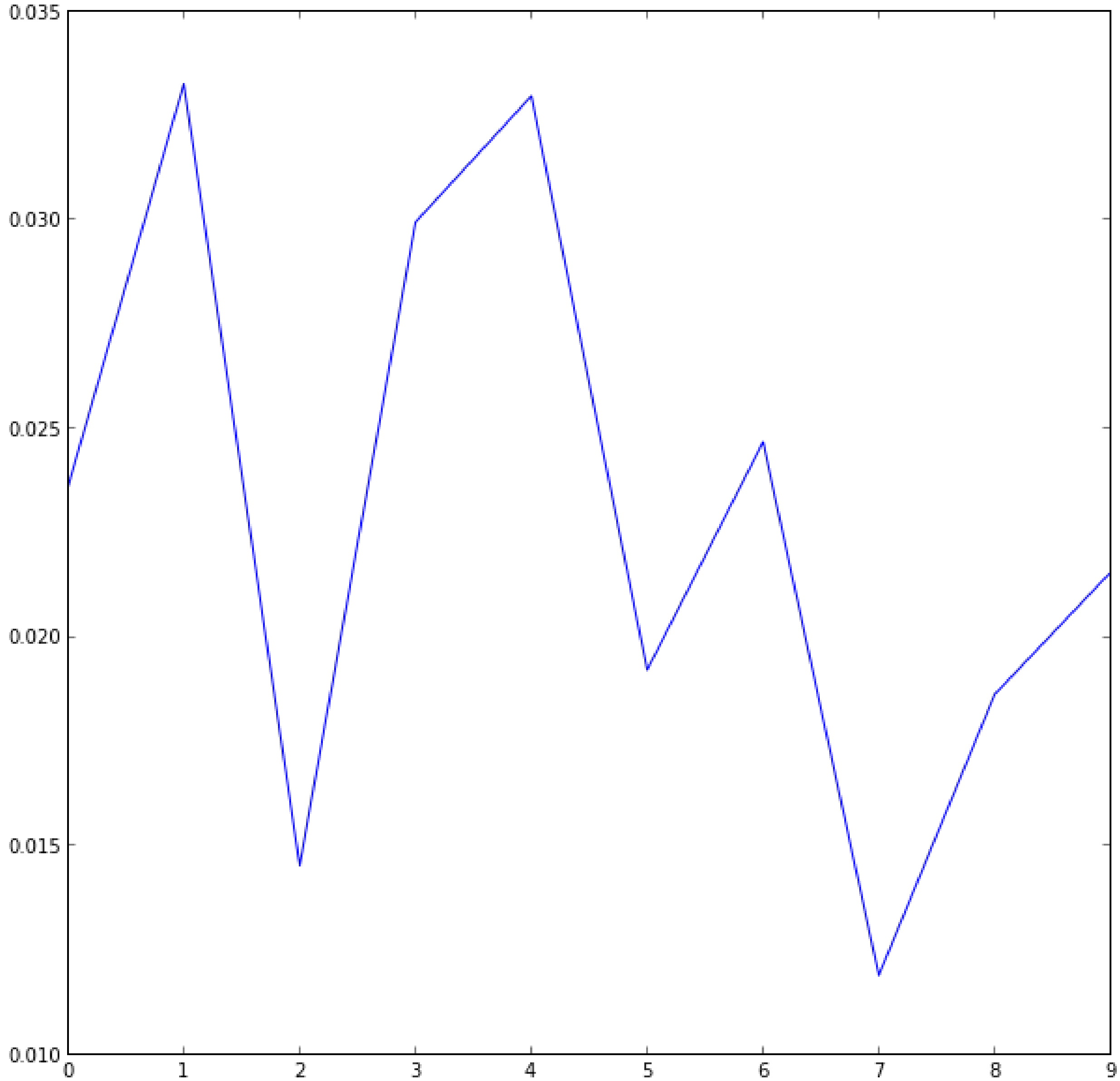
[248, 308, 137, 317, 317, 189, 277, 116, 225, 206]

8 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 5.64s to compute. Timerange: (0, 10250) 2357 Spikes in 10 channels:

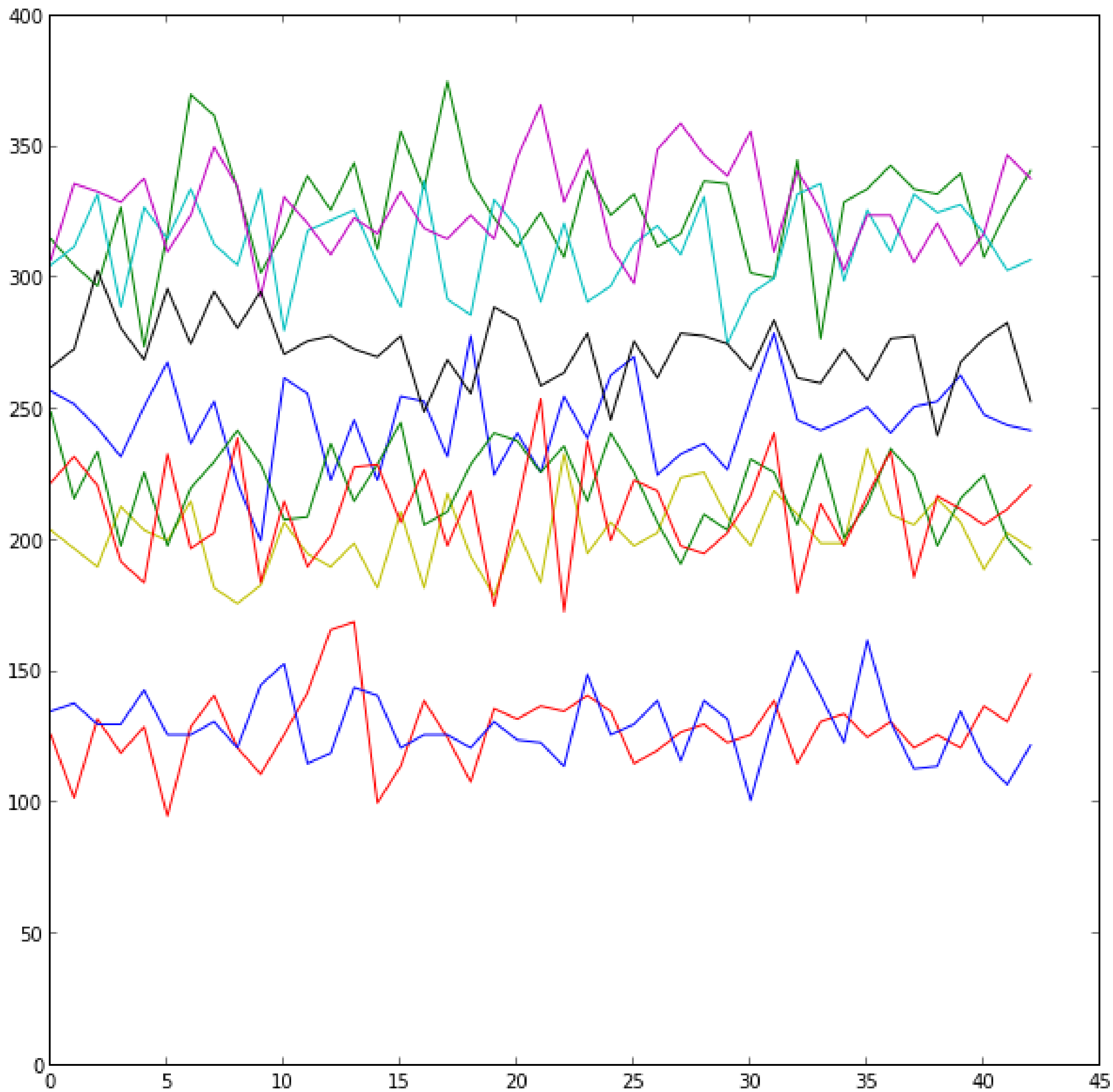
[244, 326, 131, 303, 347, 203, 283, 107, 201, 212]

9 Firing Rates plot 'ni.model.net_sim' Simulation Result Took 5.71s to compute. Timerange: (0, 10250) 2361 Spikes in 10 channels:

[242, 341, 149, 307, 338, 197, 253, 122, 191, 221]



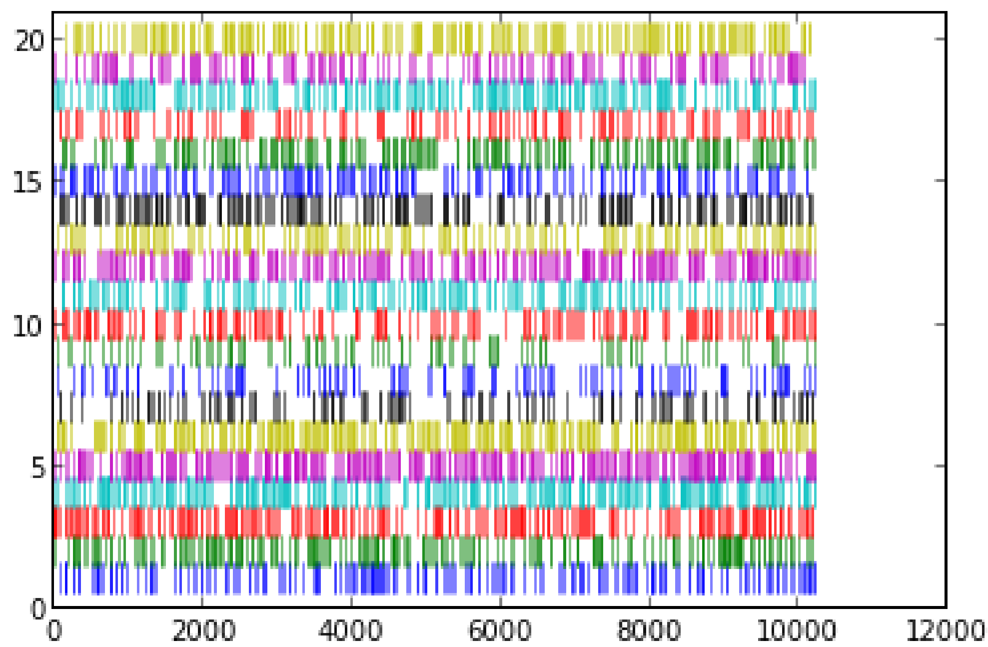

```
#[r.plot_firing_rates() for r in net.results]
plot(numpy.array([r.num_spikes_per_channel for r in net.results]))
#plot([r.spikes.T.mean() for r in net.results].T)
plot([0]*len(net.results))
```




```
res2 = net.simulate()
res2.plot()
print res2
```

Note: 'ni.model.net_sim' Simulation Result Took 11.85s to compute. Timerange: (0, 10250) 4883 Spikes in 20 channels:

[219, 258, 299, 343, 379, 346, 153, 139, 106, 220, 223, 310, 193, 274, 212, 252, 163, 291, 202, 301]

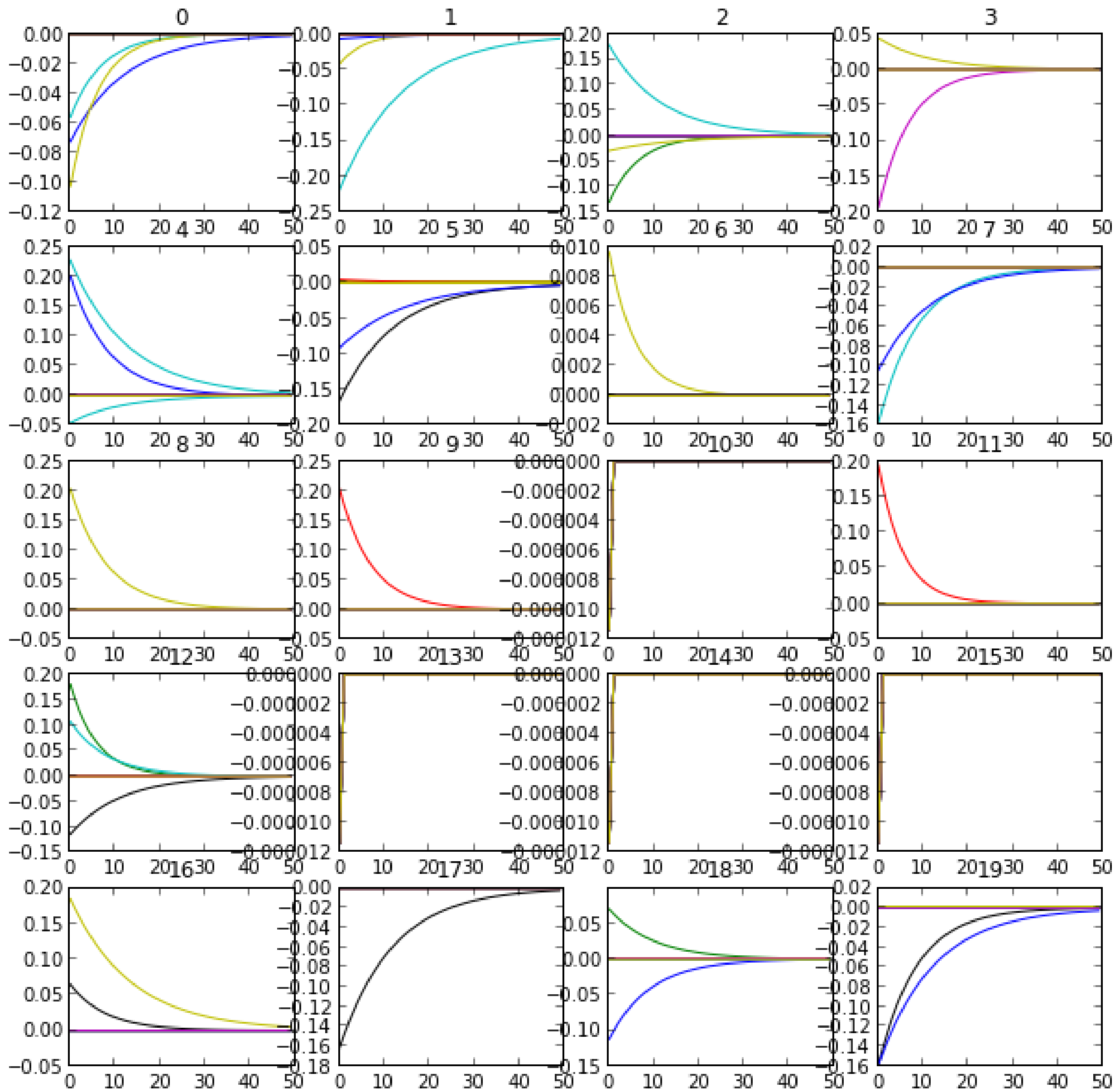



```
print res1
len(net.Jall.nonzero()[0])
print "Interaction plot"
pyplot.figsize(10,10)
for i in range(0,net.config.Nneur):
  subplot(5,net.config.Nneur/5,i+1)
  p = plot(net.Jall[i,:,:].T)
  print i, net.Jall[i,:,:].sum()
  #p.set_cmap('hot')
  title(str(i))
```

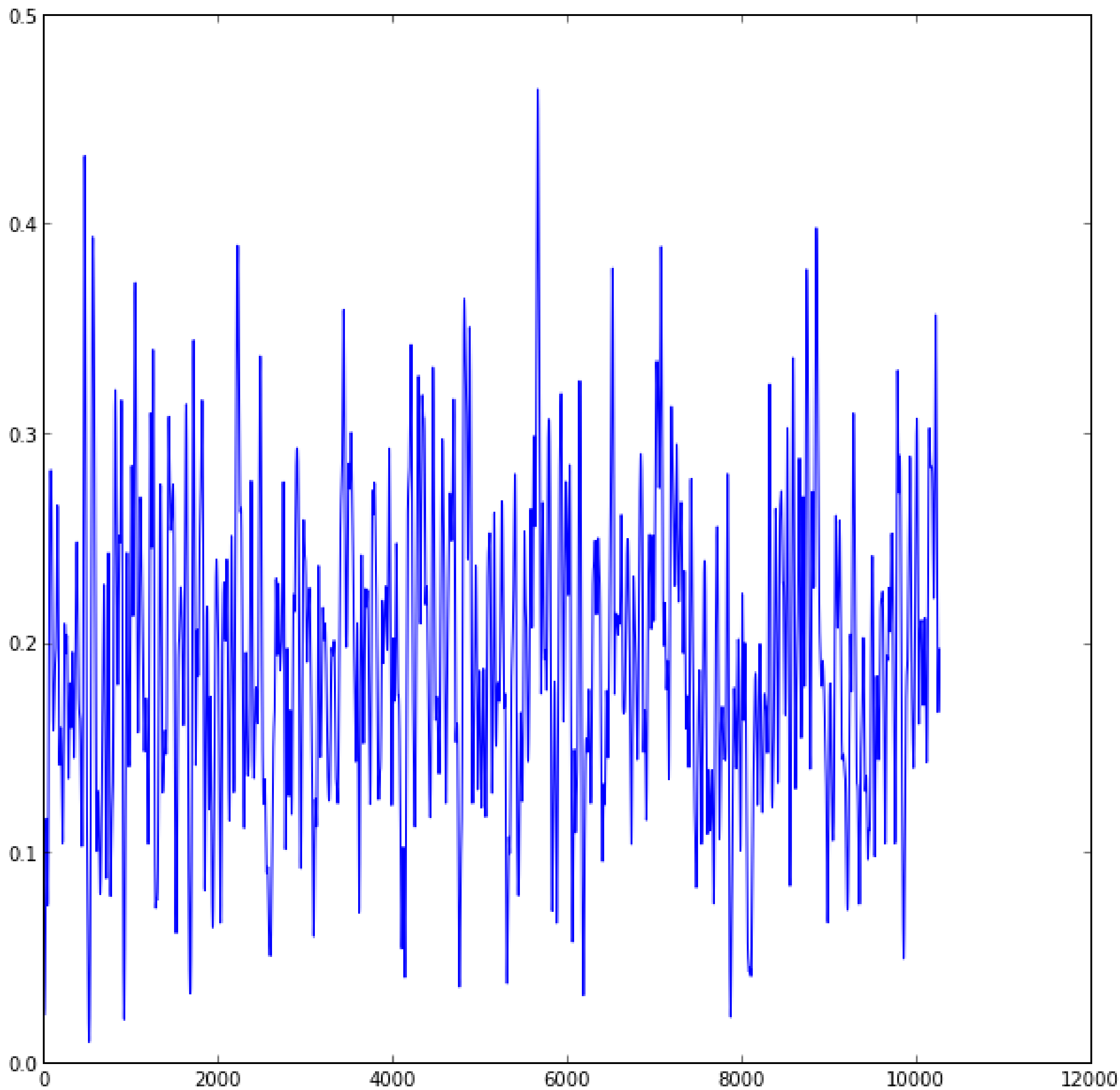
Note: 'ni.model.net_sim' Simulation Result Took 18.46s to compute. Timerange: (0, 10250) 4801 Spikes in 20 channels:

[202, 280, 241, 307, 347, 358, 140, 149, 110, 210, 206, 323, 185, 281, 239, 255, 174, 262, 248, 284]

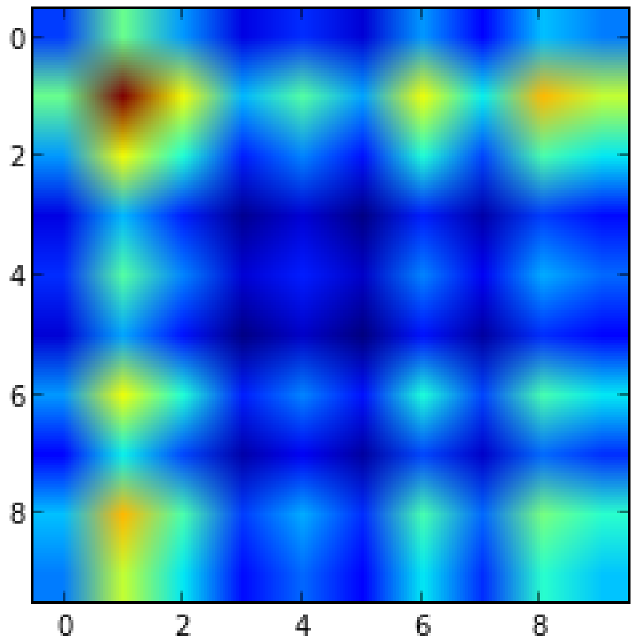
Interaction plot 0 -2.03759911202 1 -3.44761233133 2 0.723006313412 3 -0.952023565109 4 4.25574511824 5 -3.3453963685 6 0.0627568073968 7 -2.69221647701 8 1.76568144941 9 1.52325958899 10 -0.000215659457296 11 1.16132044427 12 0.784372891918 13 -0.000215659457296 14 -0.000215659457296 15 -0.000215659457296 16 3.06368154242 17 -1.95556680782 18 -0.349520835524 19 -3.41784456774



```
ni.model.pointprocess.plotGaussed(res.spikes.T.sum(),10)
```



```
#conv = [[np.convolve(p1,p2) for p1 in res.spikes] for p2 in res.spikes]
imshow([[np.convolve(res.spikes[j],res.spikes[i]).sum() for i in range(0,10)] for j in range(0,10)])
```




```
res.spikes.info
```

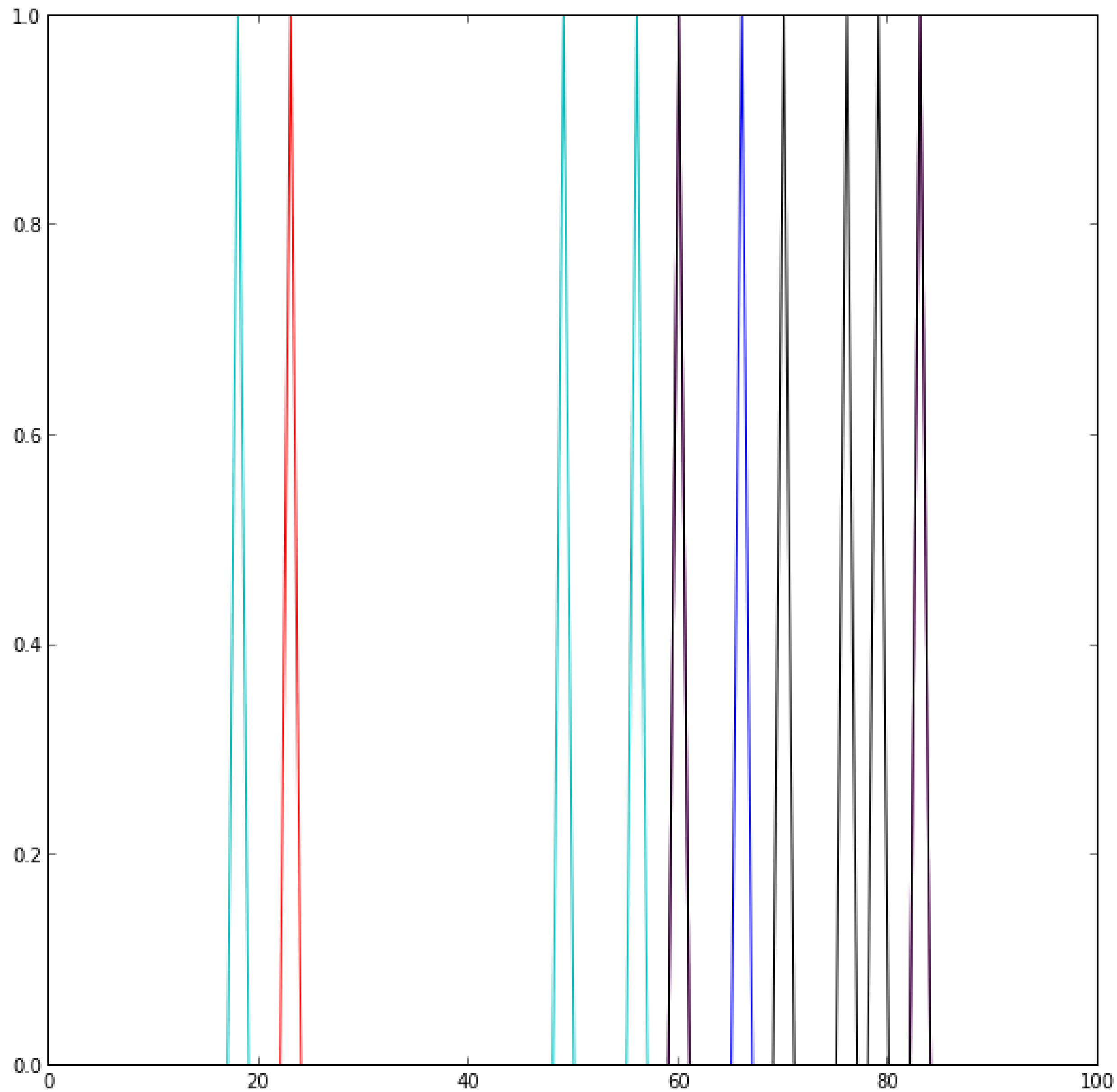
Note: <bound method DataFrame.info of <class 'pandas.core.frame.DataFrame'> Int64Index: 10250 entries, 0 to 10249 Data columns: 0 10250 non-null values 1 10250 non-null values 2 10250 non-null values 3 10250 non-null values 4 10250 non-null values 5 10250 non-null values 6 10250 non-null values 7 10250 non-null values 8 10250 non-null values 9 10250 non-null values dtypes: float64(10)>

```
import sklearn

enet = sklearn.linear_model.ElasticNet()
X = np.array(res.spikes)[0:100,0:8]
y = np.array(res.spikes)[0:100,9]
enet.fit(X,y)
enet.predict([4,5,6])
```

Note:

ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000, normalize=False, precompute='auto', rho=0.5, tol=0.0001)

```
plot(np.array(res.spikes)[0:100,0:8])
```



1.3 model Package

1.3.1 `glm` Module

Not yet functional, instead the statmodels glm can be used directly:

```
from scikits.statsmodels.genmod import generalized_linear_model
from scikits.statsmodels.genmod.families.family import Binomial

spikes = np.array(ni.model.pointprocess.createPoisson(sin(numpy.array(range(0,200))*0.01)*0.5- 0.2,1

design_matrix = [[exp(-1*(((i-j*100))**2)/1000) for j in range(0,9)] for i in linspace(0,999,1000)]
glm = generalized_linear_model.GLM(spikes,design_matrix, family = Binomial())
```

```
res = glm.fit()
```

Todo

Talk with Robert about his code

Todo

create different models that use glm in various ways / designs

class `ni.model.glm.`**`GLM`**`(`*X*, *Y*`)`
Undocumented

`fit``(`*X*`)`
Undocumented

class `ni.model.glm.`**`GLM_model`**`(`*m*, *coef*`)`
Undocumented

`predict``(`*X*`)`
Undocumented

1.3.2 `net_sim` Module

The Net Simulator is divided into a Configuration, Net and a Result object.

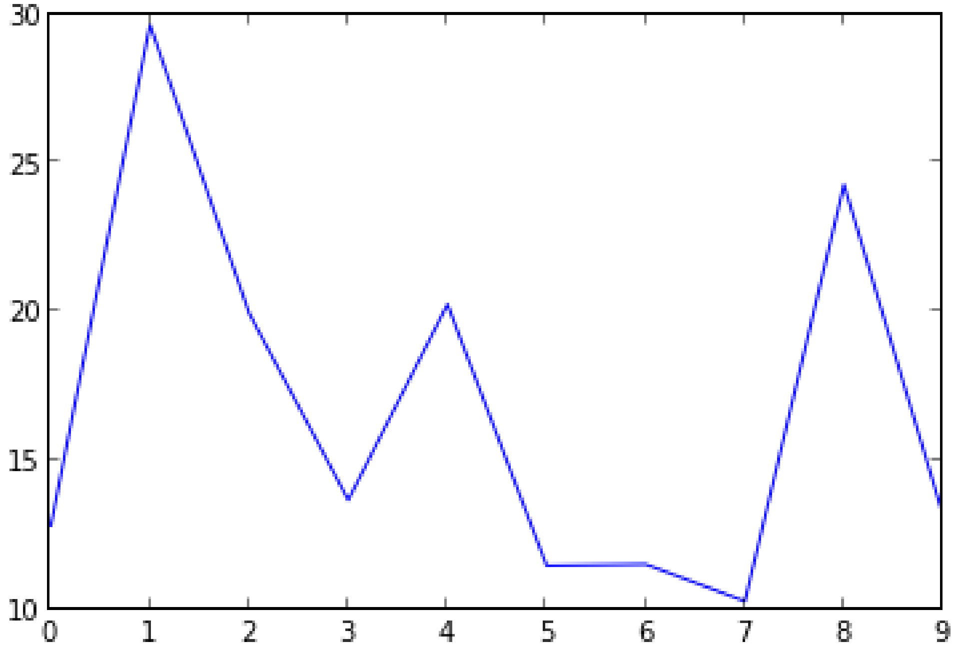
After configuration of the network it can be instantiated by calling *Net(conf)* with a valid configuration *conf*. This creates eg. random connectivity so that the simulation with the same network can be repeated multiple times.

Todo

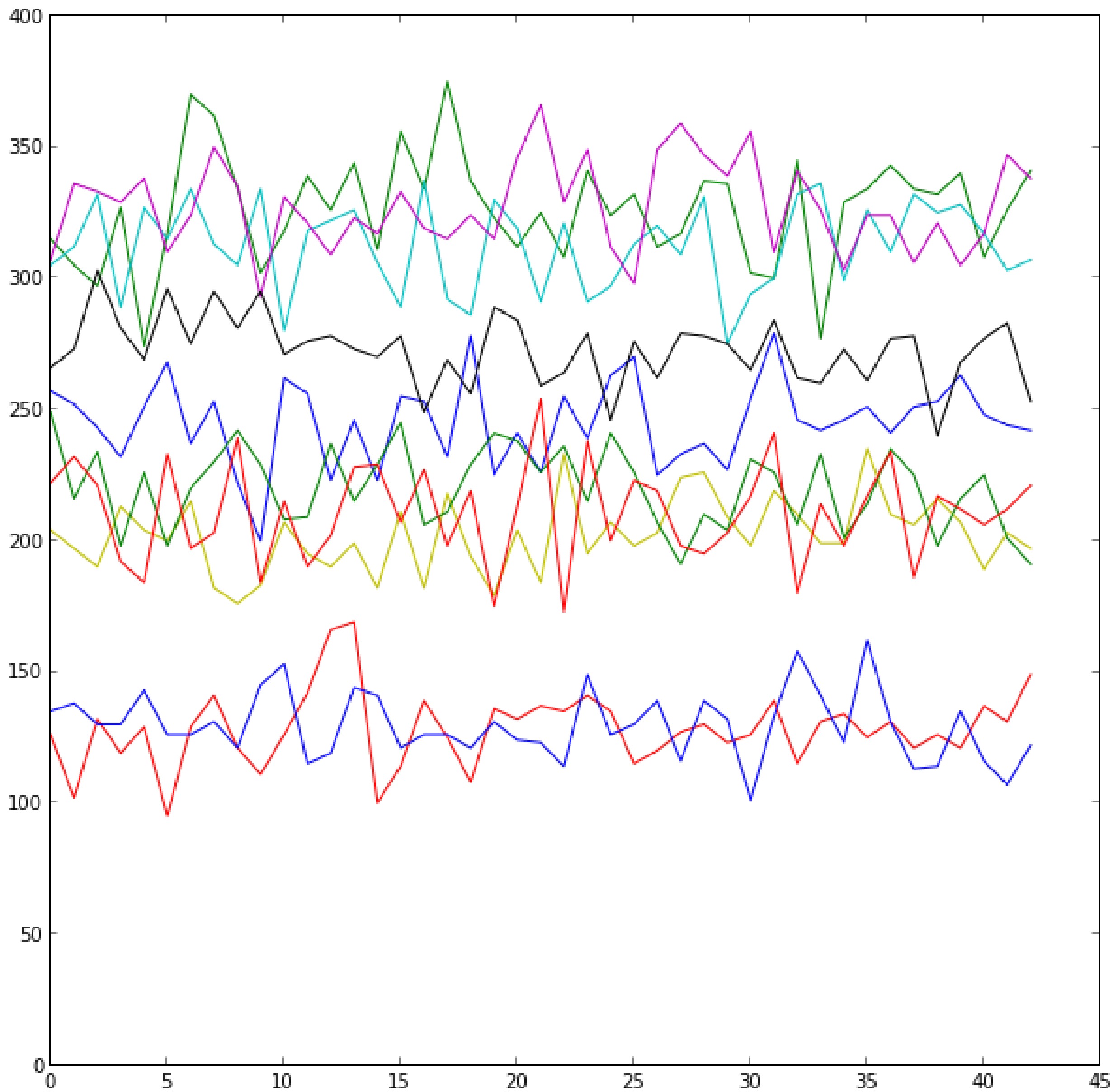
Add options for random number generator seeds, so that the *exact* same trial can be run over and over again.

```
c = ni.model.net_sim.SimulationConfiguration()
c.Nneur = 10
net = ni.model.net_sim.Net(c)
print net
net.plot_firing_rates()
```

```
'ni.model.net_sim' Simulation Setup
Timerange: (250, 10250)
 10 channels with firing rates:
        [12.815928361, 29.6328550796, 19.9415819867, 13.6710936491, 20.242131795, 11.4661487294, 11.5
Firing Rates plot
```




```
for i in range(1,11):
    print i
    res1 = net.simulate()
    res1.plot_firing_rates()

plot(numpy.array([r.num_spikes_per_channel for r in net.results]))
plot([0]*len(net.results))
```



class `ni.model.net_sim.`**`Net`**(*config*)

Undocumented

`load`(*filename*)

Undocumented

`plot_firing_rates`()

Undocumented

`plot_interaction`()

Undocumented

`save`(*filename*)

Undocumented

`simulate`()

Undocumented

class `ni.model.net_sim.`**`SimulationConfiguration`**
Undocumented

class `ni.model.net_sim.`**`SimulationResult`**
Undocumented

`plot`()
Undocumented

`plot_firing_rates`()
Undocumented

`stopTimer`()
Undocumented

`store`(*data*)
Undocumented

`ni.model.net_sim.`**`simulate`**(*config*)
Undocumented

1.3.3 `pointprocess` Module

Todo

Use different internal representations, depending on use. Ie. Spike times vs. binary array

---

class `ni.model.pointprocess.`**`MultiChannelPointProcess`**(*channels*)

`get`(*fun*)

`get2`(*fun*)

`getMeanCounts`()

`set`(*fun*)

`set2`(*fun*)

class `ni.model.pointprocess.`**`PPContainer`**

`setData`(*d*)

class `ni.model.pointprocess.`**`PointProcess`**(*dimensionality*)
A Point Process container.

Usually generated by loading from a file or via `ni.model.pointprocess.createPoisson()`

`addSpike`(*t*)
Undocumented

`getCounts`()
Undocumented

`getProbability`(*t_from*, *t_to*)
Undocumented

`plot`()
Undocumented

plotGaussed (*width*)
Undocumented

class ni.model.pointprocess.**SimpleFiringRateModel**
Uses just the firing rate as a predictor

fit (*Data*)

loglikelihood (*Data*, *Prediction*)

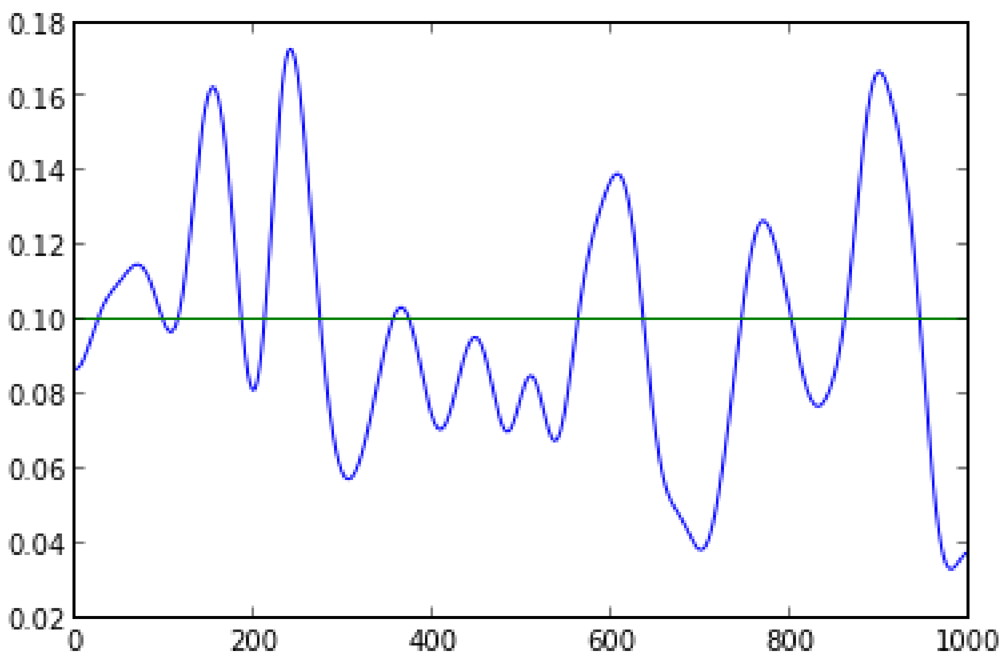
predict (*Data*)

ni.model.pointprocess.**createPoisson** (*p*, *l*)
This generates a spike sequence of length *l* according to either a fixed firing rate *p*, or a repeated sequence of firing rates if *type(p) == np.ndarray*.

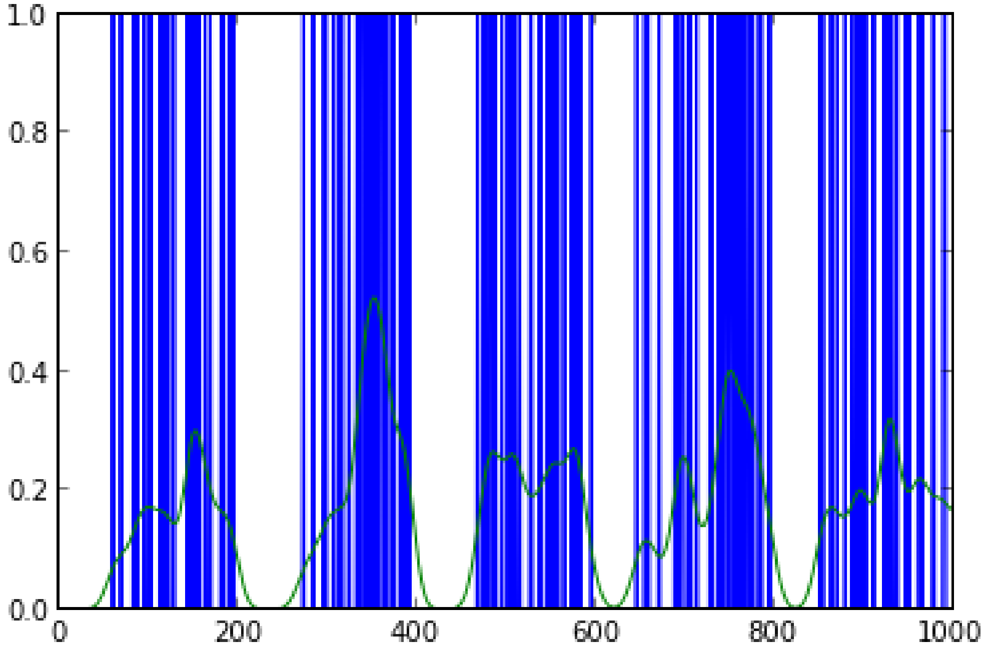
It creates a ni.model.pointprocess.PointProcess

Example 1:

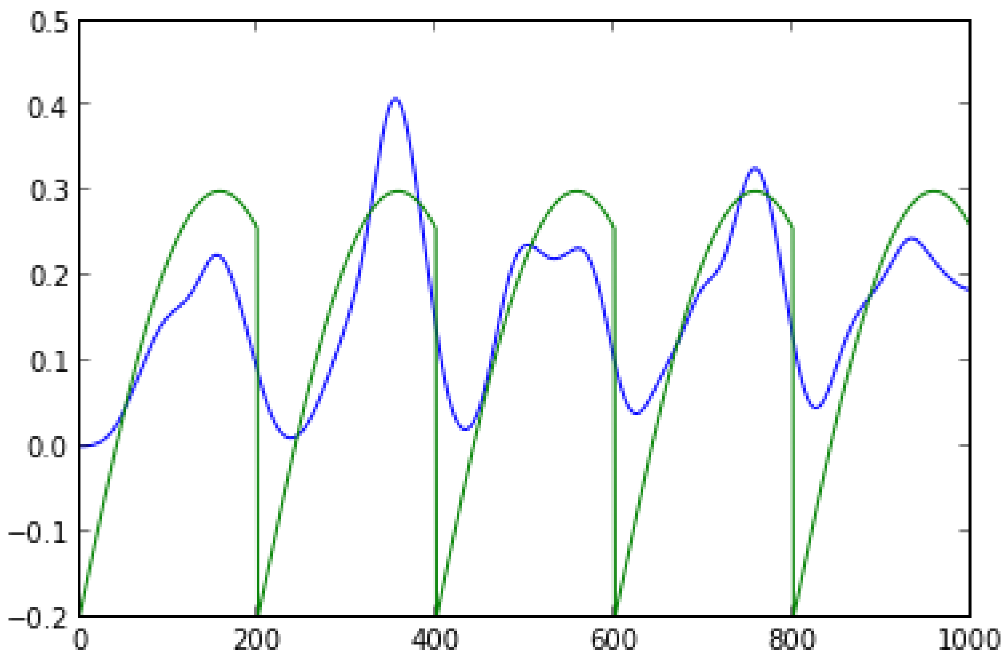
```
p1 = ni.model.pointprocess.createPoisson(0.1,1000)
p1.plotGaussed(20)
plot(p1.frate)
```




```
p2 = ni.model.pointprocess.createPoisson(sin(numpy.array(range(0,200))*0.01)*0.5- 0.2,1000)
p2.plot()
p2.plotGaussed(10)
```




```
p2.plotGaussed(20)
plot(p2.frate)
```



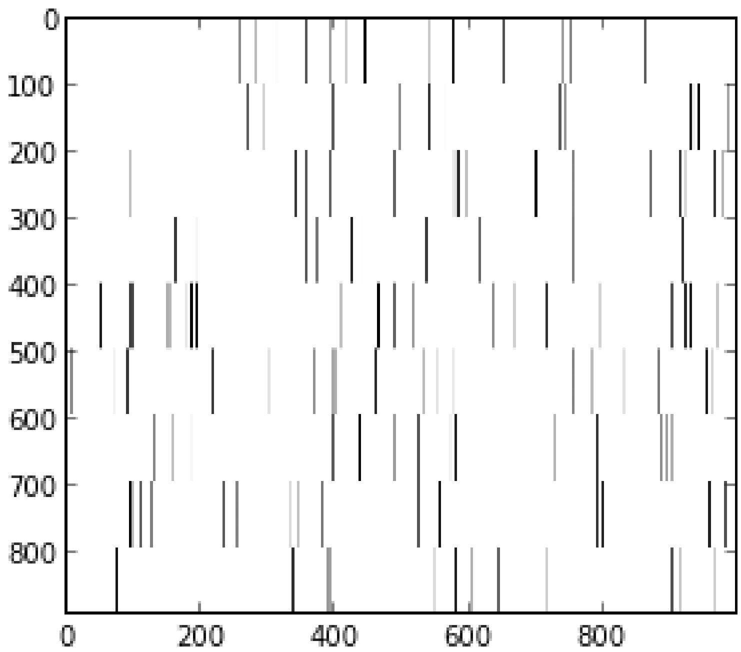
Example with multiple channels:

```
frate = (numpy.array(range(0,200))*0.001)*0.2+0.01
channels = 9

dists = [ni.model.pointprocess.createPoisson(frate,1000) for i in range(0,channels)]
#for i in range(0,9): dists[i].plotGaussed(10)
import itertools
```

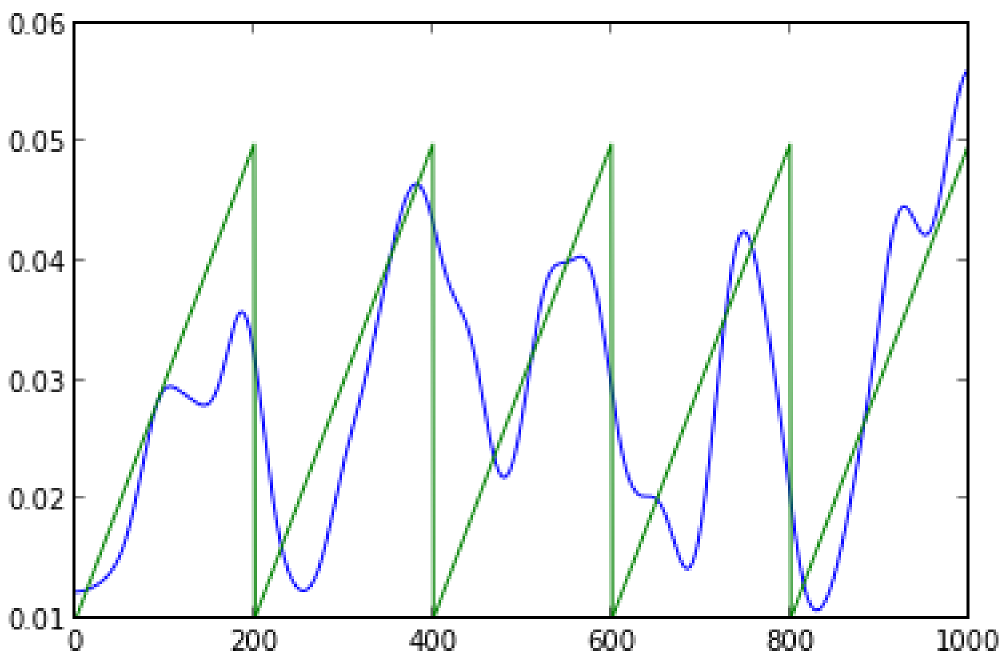
```
spks = np.array([dists[i].getCounts() for i in range(0,channels) for j in range(0,99) ])
imshow(-1*spks)
set_cmap('gray')
```

Will generate:

```
(A plot of spikes)
```

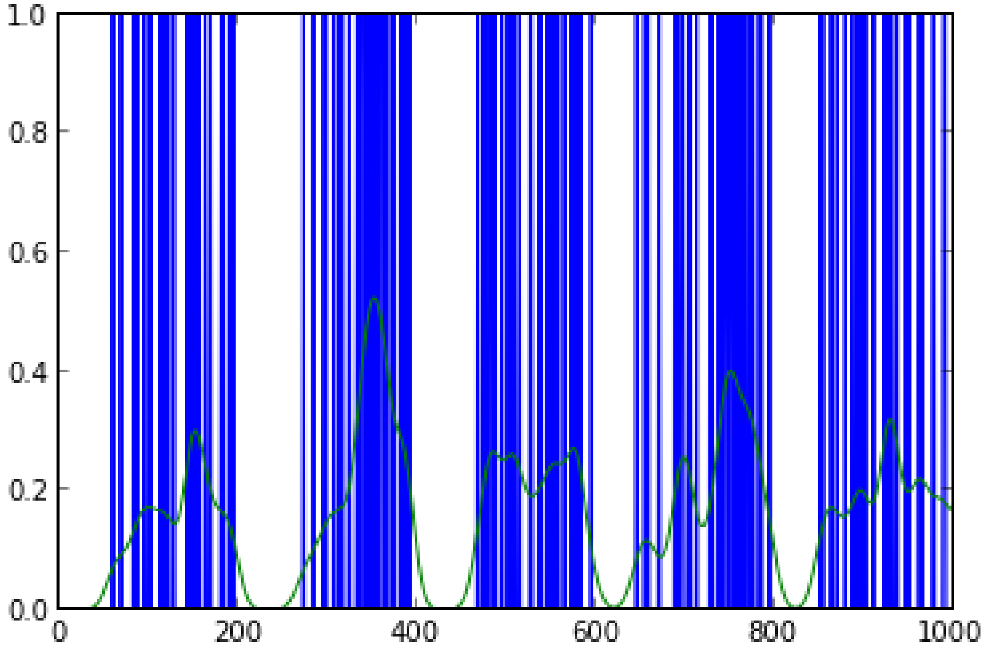



```
ni.model.pointprocess.plotGaussed(np.array([dists[i].getCounts() for i in range(0,channels)]).me
plot(dists[0].frate)
```



ni.model.pointprocess.**plotGaussed**(*data*, *width*)

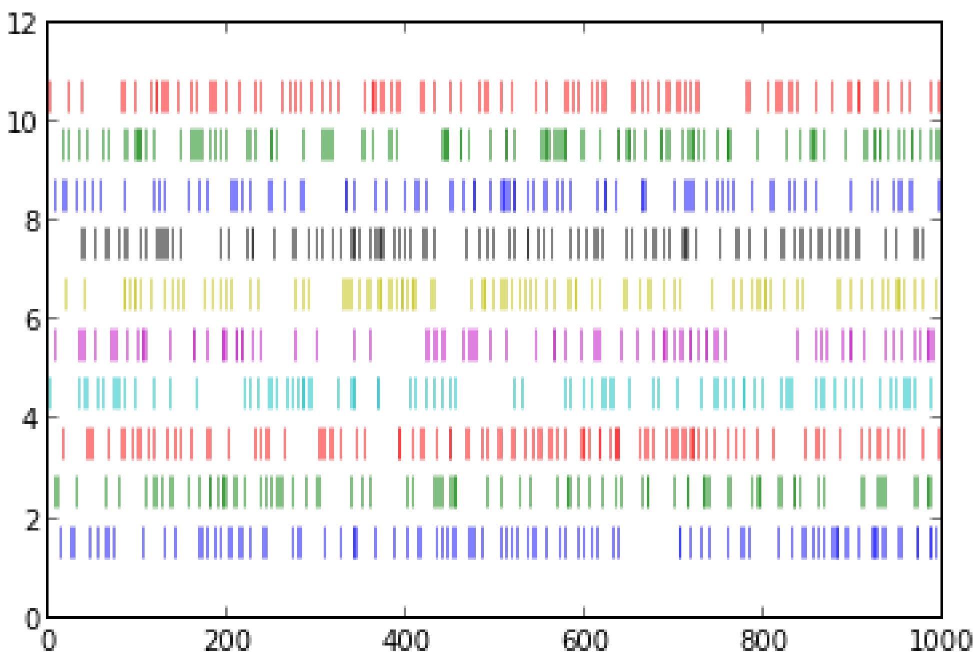
```
p2 = ni.model.pointprocess.createPoisson(sin(numpy.array(range(0,200))*0.01)*0.5- 0.2,1000)
p2.plot()
p2.plotGaussed(10)
```



ni.model.pointprocess.**plotMultiSpikes**(*spikes*)

- *spikes* is a binary 2d matrix

Generates something like:



1.4 tools Package

1.4.1 bootstrap Module

ni.tools.bootstrap.**evaluate**(*Model*, *Data*, *bootstrap_repetitions*, *return_all=False*)

Executes a certain number of bootstrap repetitions to calculate the bias of the likelihood he model computes

Model

A model object that is capable of loglikelihood estimation

Data

Data that is to be reshuffled. A bootstrap sample is drawn from this Data of the same length with each Element of Data being equally probable of being included.

bootstrap_repetitions

Number of repetitions

return_all

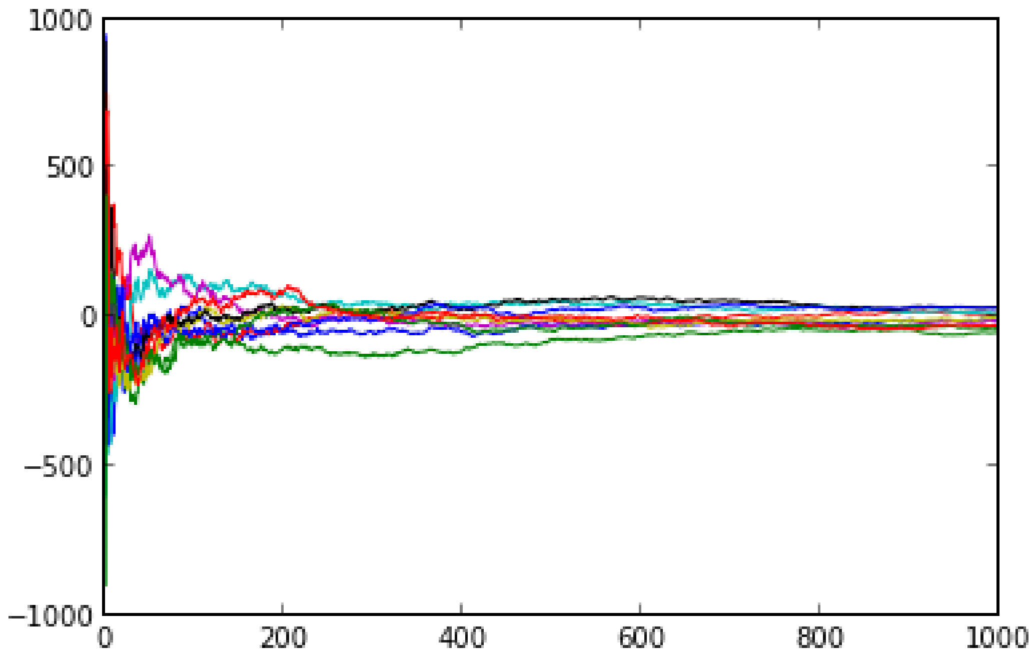
Default: False. Whether an array of all bootstrap biases should be returned or just the mean.

Example:

```
import ni.tools.bootstrap
reload(ni.tools.bootstrap)
import ni.model.pointprocess
reload(ni.model.pointprocess)
p1 = np.array([ni.model.pointprocess.createPoisson(0.1,1000).getCounts() for i in range(0,10)])
p2 = np.array([ni.model.pointprocess.createPoisson(sin(numpy.array(range(0,200))*0.01)*0.5- 0.2,

m1 = ni.model.pointprocess.SimpleFiringRateModel()

ni.tools.bootstrap.evaluate(m1,p1,10000)
```

Or to see the effect of increasing bootstrap size:

```
[plot(np.cumsum(ni.tools.bootstrap.evaluate(m1,p1,1000,return_all=True))/range(1,1001)) for i in
```



`ni.tools.bootstrap.`**`likelihood_Fun`**(*y*, *x*, *mu*)
Calculates the likelihood for a binary vector and a predicted firing rate

$$-(size(y)/2) \cdot log(2 \cdot \pi \cdot x^2) - (1/x^2 \cdot (y - mu)^2)$$

1.4.2 `progressbar` Module

`ni.tools.progressbar.`**`progress`**(*a*, *b*)
Undocumented

`ni.tools.progressbar.`**`progress_end`**()
Undocumented

`ni.tools.progressbar.`**`progress_init`**()
Undocumented

1.4.3 `project` Module

NI Project Management

- All steps in a configuration / simulation process will be logged to some folder structure
- after the simulation and even after changing the original code, the results should still be viewable / interpretable with a project viewer
- batches of runs should be easy to batch interpret (characteristic plots etc.)
- **metadata should contain among others:** date software versions configuration options manual comments
- saving of plots/data should be done by the project manager

class `ni.tools.project.`**`Project`**(*name*, *folder*)
Undocumented

CHAPTER

TWO

LIST OF TODO ITEMS

Todo

Talk with Robert about his code

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/model/glm.py:docstring of ni.model.glm, line 19.)

Todo

create different models that use glm in various ways / designs

---

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/model/glm.py:docstring of ni.model.glm, line 23.)

Todo

Add options for random number generator seeds, so that the *exact* same trial can be run over and over again.

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/model/net_sim.py:docstring of ni.model.net_sim, line 11.)

Todo

Use different internal representations, depending on use. Ie. Spike times vs. binary array

---

(The *original entry* is located in /home/plogic/uni/MT/EIC/py/ni/model/pointprocess.py:docstring of ni.model.pointprocess, line 7.)

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

n

INDEX