In [2764]:

```
%matplotlib inline
from __future__ import (print_function,
                        unicode_literals,
                        division)
from future.builtins import str, open, range, dict
import matplotlib
from mpl_toolkits.axes_grid1 import make_axes_locatable
import numpy as np
import matplotlib.pyplot as plt
from pprint import pprint

def plot_in_range(fig, ax, spikes, start_t, end_t, color='blue', mar
kersize=1):
    for i, times in enumerate(spikes):
        times = np.array(times)
        whr = np.where(np.logical_and(times >= start_t,  times <= en
d_t))[0].astype('int')
        if len(whr):
            plot_times = times[whr]
            plt.plot(plot_times, i*np.ones_like(plot_times), '.', co
lor=color,
                     markersize=markersize, markeredgewidth=0)


def pop_rate_per_second(spikes, total_t, dt):
    ms_to_s = 1./1000.
    n_neurons = len(spikes)
    end = 0
    rates = []
    for start in range(0, total_t, dt):
        end = start + dt
        spike_count = 0
        for times in spikes:
            times = np.array(times)
            whr = np.where(np.logical_and(times >= start, times < en
d))[0]
            spike_count += len(whr)
        rate = float(spike_count) / float(n_neurons * dt * ms_to_s)
#         rate = float(spike_count) / float(dt * ms_to_s)
        rates.append(rate)

    return rates


def active_neurons_per_pattern(spikes, start_time, end_time, sample_
indices, start_indices_index, config):
    t_per_sample = config['time_per_sample']
    n_samples = config['n_samples']
    n_patterns = config['n_patterns']

    posts = []
    active_neurons = {idx: set() for idx in range(n_patterns)}
    st = start_time
    for idx in range(start_indices_index, len(sample_indices)):
        pat_id = sample_indices[idx] // n_samples
        et = st + t_per_sample
        posts[:] = []
        for post_id, times in enumerate(spikes):
            times = np.array(times)
            find = np.where(np.logical_and(times >= st, times < et))
[0]
            if len(find):
                posts.append(post_id)

#         print(pat_id, st, et, posts)
```

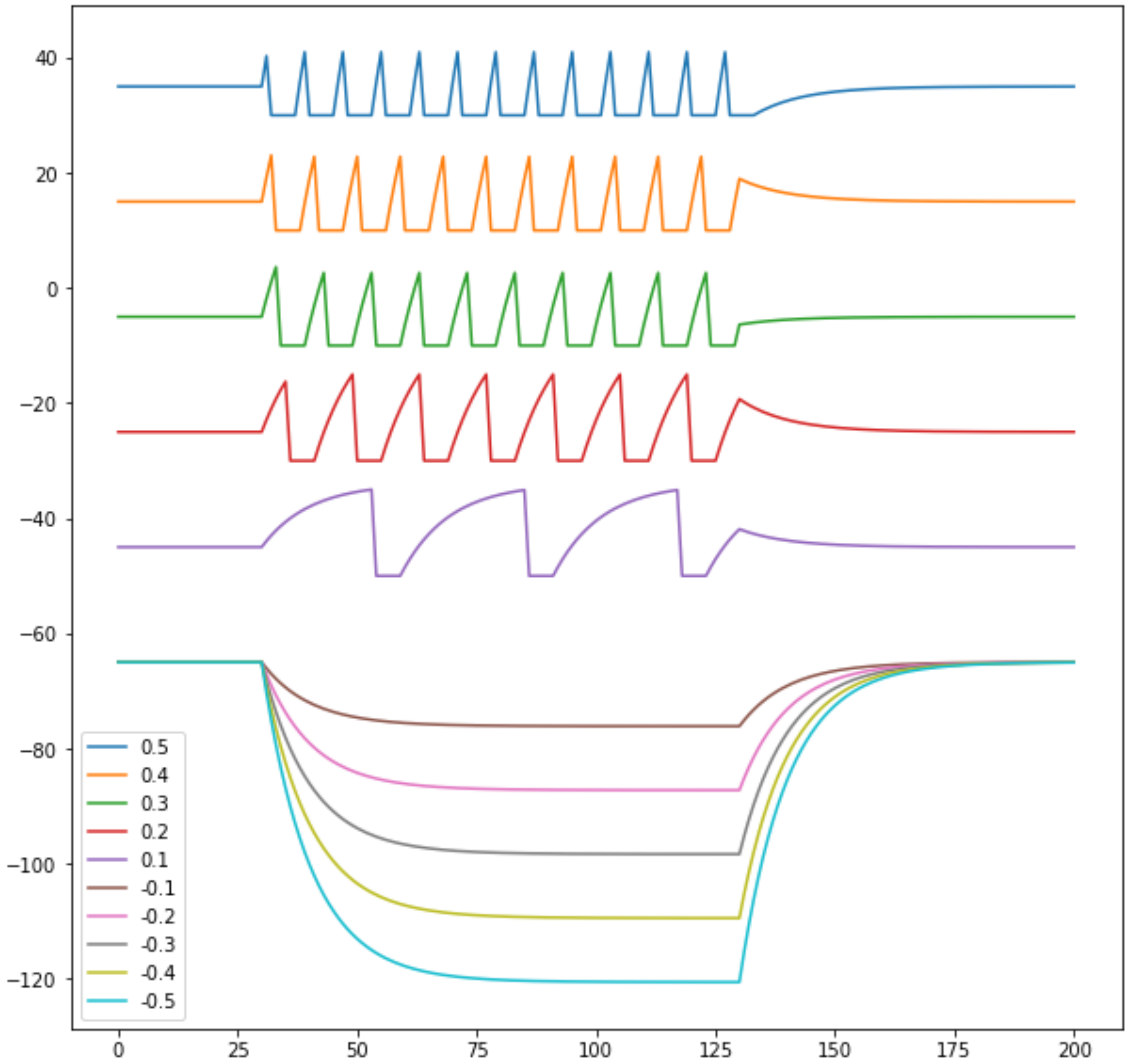
Neuron parameters:

Param	Kenyon	Horn	Decision	Units
C_m	0.25			nF
V_{reset}	-70			mV
V_{rest}	-65			mV
V_{thresh}	-20			mV
$e_{rev,E}$	0			mV
$e_{rev,I}$	-92			mV
τ_m	10			ms
τ_{refrac}	1			ms
τ_{syn_E}	1.0	1.0	5.0	ms
τ_{syn_I}	1.5	5.0	2.5	ms

In [2765]:
```
1e100
```

Out[2765]: 1e+100

In [2766]:
```
import glob

dc_data = {}
for f in glob.glob('response*.npz'):
    d = np.load(f)
    dc_data[d['dc'].item()] = d

fig = plt.figure(figsize=(10, 10))
for dc in sorted(dc_data.keys(), reverse=True):
    offset = 200*dc if dc > 0 else 0
    plt.plot(dc_data[dc]['voltage']+offset, label=dc_data[dc]['dc'])
plt.legend()
plt.show()
```

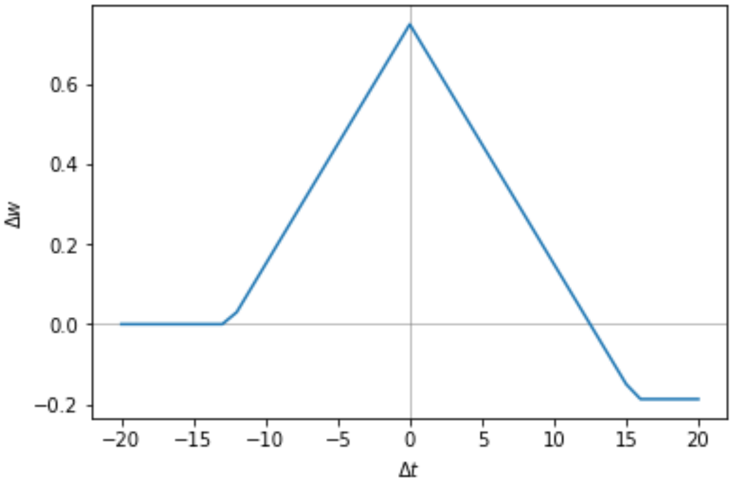


In [2766]:

In [2766]:

In [2767]:

```
c10 = 10.0**5
c01 = 20.0
c11 = 5.0
tl = 25.0
tsh = 10.0
td = 10.0**5
g0 = 1.25
gmax = 3.75
t_minus =  -((1.0/c10 + 1.0/c11)*tl*c11)/2.0
t_plus = ((1.0/c01 + 1.0/c11)*tl*c11)/2.0
a_plus = -2.0*gmax/(tl*c11)
a_minus = -a_plus
y_minus = -gmax/c10
y0 = gmax/c11
y_plus = -gmax/c01

max_dt = 20.0
dt = np.arange(-max_dt, max_dt + 1)
y = np.zeros_like(dt)
y[dt < t_minus] = y_minus
w = np.where(np.logical_and(t_minus < dt, dt <= 0))
y[w] = a_minus * dt[w] + y0
w = np.where(np.logical_and(0 < dt, dt <= t_plus))
y[w] = a_plus * dt[w] + y0
y[dt > t_plus] = y_plus

plt.figure()
ax = plt.subplot(1, 1, 1)
plt.axhline(0, color='gray', linewidth=0.5)
plt.axvline(0, color='gray', linewidth=0.5)
plt.plot(dt, y)

ax.set_ylabel('$\Delta w$')
ax.set_xlabel('$\Delta t$')
# plt.yscale('log')
plt.show()
```



In [2767]:

In [2768]:
```
fname = "mbody-nKC=1000_nDN=100_probAL2KC=0p15_gScale=0p025_nAL=100_
nPatternsAL=10_probAL=0p2_w2s=0p25_probNoiseSamplesAL=0p1_probKC2DN=
0p2_nLH=20_inactiveScale=0p1_randomizeSamplesAL=True_nSamplesAL=1000
_backend=genn"
fname = "mbody-nKC=1000_nDN=100_probAL2KC=0p15_gScale=0p025_nAL=100_
nPatternsAL=10_probAL=0p2_w2s=1p0_probNoiseSamplesAL=0p1_probKC2DN=0
p2_nLH=20_inactiveScale=0p1_randomizeSamplesAL=True_nSamplesAL=1_bac
kend=genn"
fname = "mbody-nKC=1000_nDN=100_probAL2KC=0p15_gScale=0p025_nAL=100_
nPatternsAL=10_probAL=0p2_w2s=0p5_probNoiseSamplesAL=0p1_probKC2DN=0
p2_nLH=20_inactiveScale=0p1_randomizeSamplesAL=True_nSamplesAL=1000_
backend=genn"

fname = "mbody-experiment"
fname = "mbody-experiment-10k-working"
# fname = "mbody-experiment-5k-working"
fname += ".npz"
```

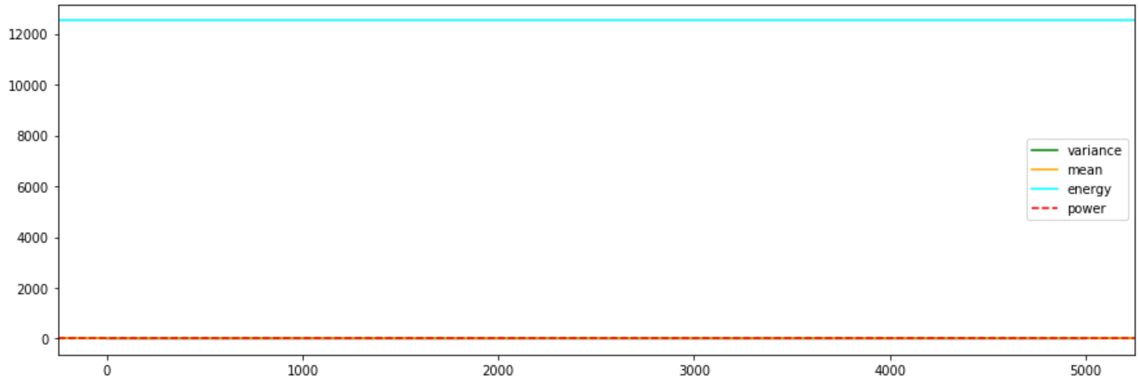
In [2769]:
```
data = np.load(fname)
pprint(data.keys())
args = data['args'].item()
pprint(data['static_weights'].item())
pprint(args)
pprint(data['stdp_params'].item())
```

```
['output_start_connections',
 'max_rand_dt',
 'kenyon_spikes',
 'sample_dt',
 'stdp_params',
 'static_weights',
 'args',
 'horn_spikes',
 'lateral_horn_connections',
 'input_vectors',
 'sample_indices',
 'sim_time',
 'input_samples',
 'neuron_parameters',
 'decision_spikes',
 'start_dt',
 'input_spikes',
 'output_end_weights']
{u'AL to KC': 0.0025,
 u'AL to LH': 0.01625,
 u'DN to DN': 0.0025,
 u'KC to DN': 0.00375,
 u'KC to KC': 0.00025,
 u'LH to KC': 0.0048125}
Namespace(backend='genn', gScale=0.025, inactiveScale=0.1, nAL=100,
nDN=100, nKC=2500, nLH=20, nPatternsAL=10, nSamplesAL=10000, probAL=
0.2, probAL2KC=0.15, probAL2LH=0.5, probKC2DN=0.2, probNoiseSamplesA
L=0.1, randomizeSamplesAL=True, renderSpikes=False, w2s=0.0025)
{u'timing_dependence': {u'name': u'SpikePairRule',
                        u'params': {u'A_minus': 0.05,
                                    u'A_plus': 0.01,
                                    u'tau_minus': 33.7,
                                    u'tau_plus': 16.8}},
 u'weight_dependence': {u'name': u'MultiplicativeWeightDependence',
                        u'params': {u'w_max': 0.00375, u'w_min': 0.0
}}}
```

In [2769]:
```
```

In [2770]:
```
out_spikes = data['decision_spikes']
dt = 1000
total_t = int(data['sim_time'])
print(total_t)
out_rates = np.array(pop_rate_per_second(out_spikes, total_t, dt))
out_mean = avg_mean(out_rates)
out_e = energy(out_rates)
out_p = power(out_rates)
out_var = variance(out_rates)

plt.figure(figsize=(15, 5))
plt.plot(out_rates)
plt.axhline(out_var, color='green', label='variance')
plt.axhline(out_mean, color='orange', label='mean')
plt.axhline(out_e, color='cyan', label='energy')
plt.axhline(out_p, color='red', linestyle='--', label='power')
plt.legend()
plt.show()
```

```
5000000
```

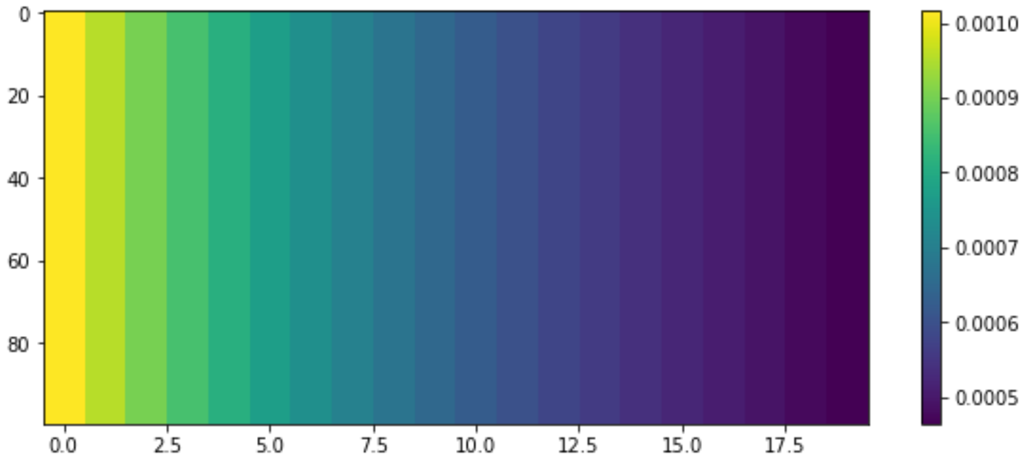


In [2770]:

In [2775]:
```
min_w = 0.00005
end_w = np.around(data['output_end_weights'], decimals=6).reshape((a
rgs.nKC, args.nDN))
# end_w[:] = end_w
if bool(1):
    start_conn = data['output_start_connections']
    start_w = np.zeros((args.nKC, args.nDN))
    for pre, post, w, d in start_conn:
        start_w[pre, post] = w
    start_w[:] = np.round(start_w, decimals=6)
else:
    start_w = np.around(data['output_start_connections'][:, 2], deci
mals=9).reshape((args.nKC, args.nDN))
    diff = np.setdiff1d(start_w, end_w)
    print(len(diff))
    print(diff)

max_w = np.max(end_w)
max_start_w = np.max(start_w)
```

AL to LH weights:

In [2776]:
```
nrows, ncols = args.nAL, args.nLH
weights = np.zeros((nrows, ncols))
for pre, post, w, d in data['lateral_horn_connections']:
    weights[pre, post] = w
plt.figure(figsize=(10,4))
ax = plt.subplot(1, 1, 1)
plt.imshow(weights)
plt.colorbar()
ax.set_aspect(0.1)
plt.show()
```

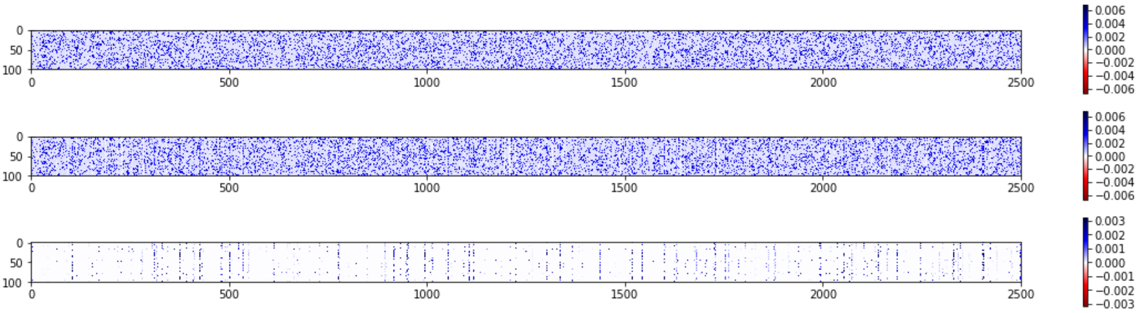


In [2777]:
```
# for i in range(10):
#     print(start_w[0][i], end_w[0][i])
```

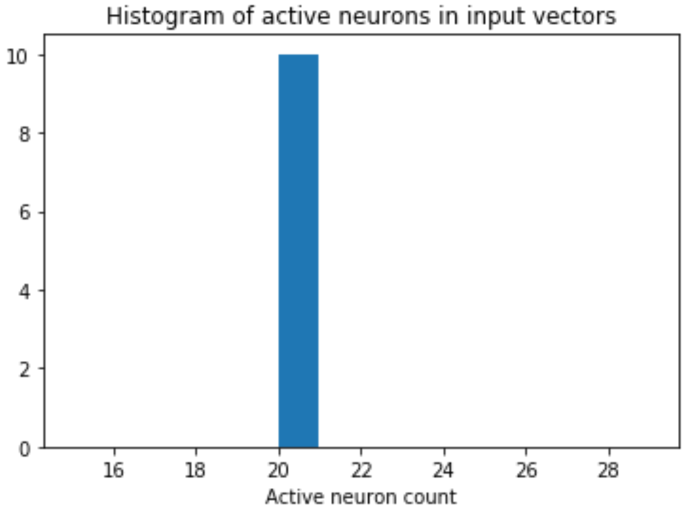
In [2778]:

```
args = data['args'].item()
print(args)
print(args.nKC, args.nDN)
cmap = 'seismic_r'
# cmap = 'Greys_r'
plt.figure(figsize=(20, 5))
ax = plt.subplot(3, 1, 1)
im = plt.imshow(start_w.reshape((args.nKC, args.nDN)).transpose(), c
map=cmap, vmin=-max_start_w, vmax=max_start_w)
plt.colorbar(im, ax=ax)

ax = plt.subplot(3, 1, 2)
im = plt.imshow(end_w.reshape((args.nKC, args.nDN)).transpose(), cma
p=cmap, vmin=-max_w, vmax=max_w)
plt.colorbar(im, ax=ax)


ax = plt.subplot(3, 1, 3)
diff = end_w - start_w
diff[diff <= 0.0001] = 0.0
max_diff = np.abs(diff).max()
im = plt.imshow((diff).reshape((args.nKC, args.nDN)).transpose(), cm
ap=cmap, vmin=-max_diff, vmax=max_diff)
plt.colorbar(im, ax=ax)
plt.savefig("weight_change.pdf")
plt.show()
```

```
Namespace(backend='genn', gScale=0.025, inactiveScale=0.1, nAL=100,
nDN=100, nKC=2500, nLH=20, nPatternsAL=10, nSamplesAL=10000, probAL=
0.2, probAL2KC=0.15, probAL2LH=0.5, probKC2DN=0.2, probNoiseSamplesA
L=0.1, randomizeSamplesAL=True, renderSpikes=False, w2s=0.0025)
2500 100
```

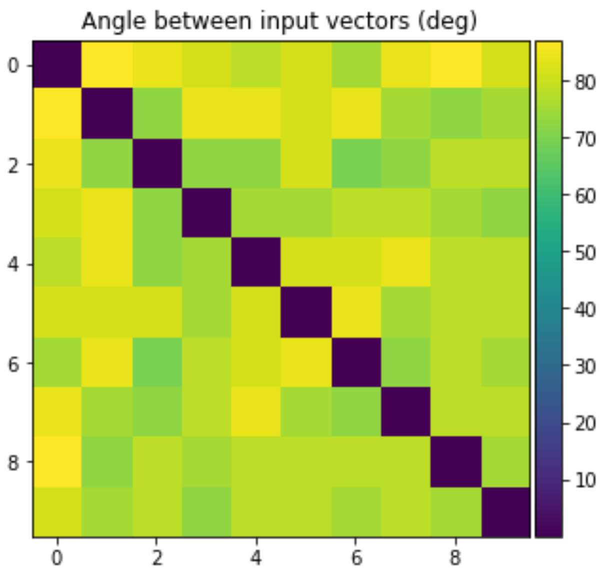


In [2779]:
```
plt.figure()
ax = plt.subplot(1, 1, 1)
ax.set_title('Histogram of active neurons in input vectors')
sums = data['input_vectors'].sum(axis=1)
plt.hist(sums, bins=range(15, 30))
ax.set_xlabel('Active neuron count')
plt.show()
```

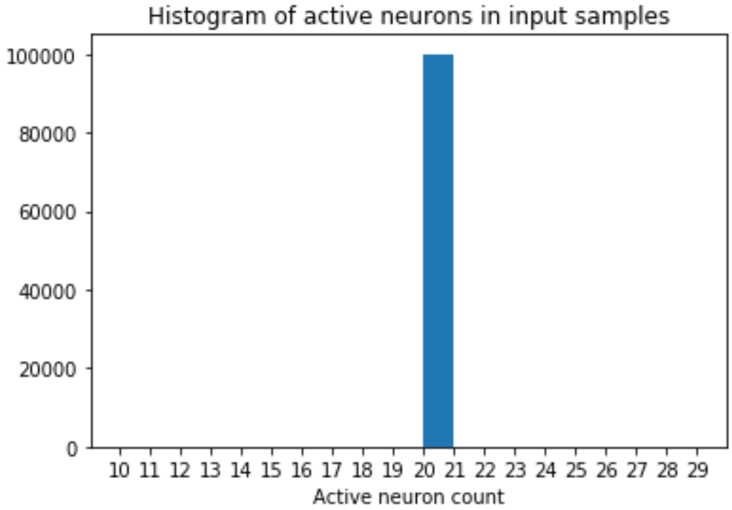


In [2780]:
```
d = np.zeros((len(data['input_vectors']), len(data['input_vectors'])
))
for i in range(len(data['input_vectors'])):
    a = data['input_vectors'][i]
    na = np.sqrt(np.dot(a, a))
    for j in range(len(data['input_vectors'])):
        b = data['input_vectors'][j]
#         diff =  a - b
#         d[i, j] = np.sqrt(np.dot(diff, diff))
        nb = np.sqrt(np.dot(b, b))
        d[i, j] = np.rad2deg( np.arccos( np.dot(a, b)/(na*nb) ) )

fig = plt.figure(figsize=(5, 5))
ax = plt.subplot(1, 1, 1)
im = plt.imshow(d, interpolation='none')
div = make_axes_locatable(ax)
cax = div.append_axes('right', size='5%', pad=0.05)
cbar = plt.colorbar(im, cax=cax)
ax.set_title('Angle between input vectors (deg)')
```

Out[2780]: Text(0.5,1,u'Angle between input vectors (deg)')



In [2781]:
```
plt.figure()
ax = plt.subplot(1, 1, 1)
ax.set_title('Histogram of active neurons in input samples')
sums = data['input_samples'].sum(axis=1)
plt.hist(sums, bins=range(10, 30))
ax.set_xlabel('Active neuron count')
ax.set_xticks(range(10, 30))
plt.show()
```



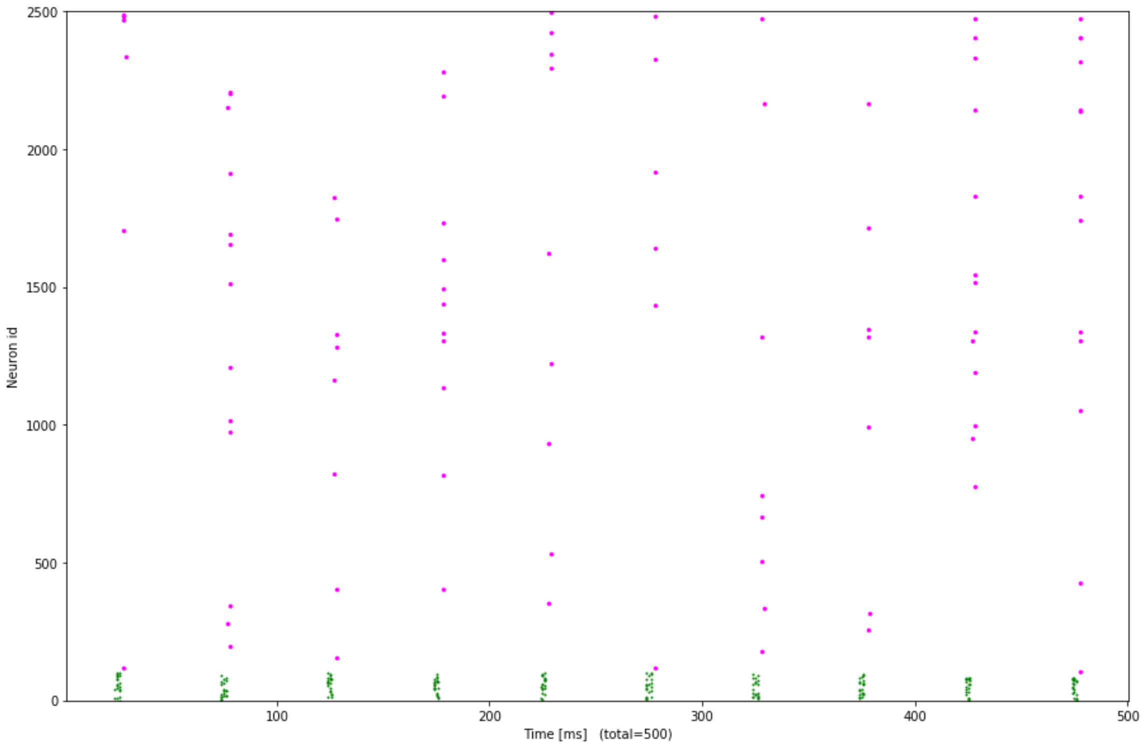
In [2782]:
```
out_spikes = data['decision_spikes']
k_spikes = data['kenyon_spikes']
h_spikes = data['horn_spikes']
in_spikes = data['input_spikes']
total_t = int(data['sim_time'])
```

In [2783]:
```
start_t = 0
end_t = total_t
end_t = 500

if args.nSamplesAL <= 100:
    # start_t = int(total_t*0.999)
    fig = plt.figure(figsize=(15, 10))
    ax = plt.subplot(1, 1, 1)
    plot_in_range(fig, ax, in_spikes,  start_t=start_t, end_t=end_t,
markersize=4, color='green')
    ax.set_ylabel('Neuron id')
    ax.set_xlabel('Time [ms]   (total={})'.format(total_t - start_t)
)
    ax.set_ylim(-1, 101)

    plt.show()
```

Kenyon cell response - start

In [2784]:
```
# start_t = int(total_t*0.999)
start_t = 0
end_t = 500
# end_t = total_t
print(start_t, total_t)
fig = plt.figure(figsize=(15, 10))
ax = plt.subplot(1, 1, 1)
plot_in_range(fig, ax, in_spikes,  start_t=start_t, end_t=end_t, markersize=4, color='green')
plot_in_range(fig, ax, k_spikes,  start_t=start_t, end_t=end_t, markersize=7, color='magenta')

ax.set_ylabel('Neuron id')
ax.set_xlabel('Time [ms]   (total={})'.format(end_t - start_t))
ax.set_ylim(-1, args.nKC + 1)

plt.show()
```

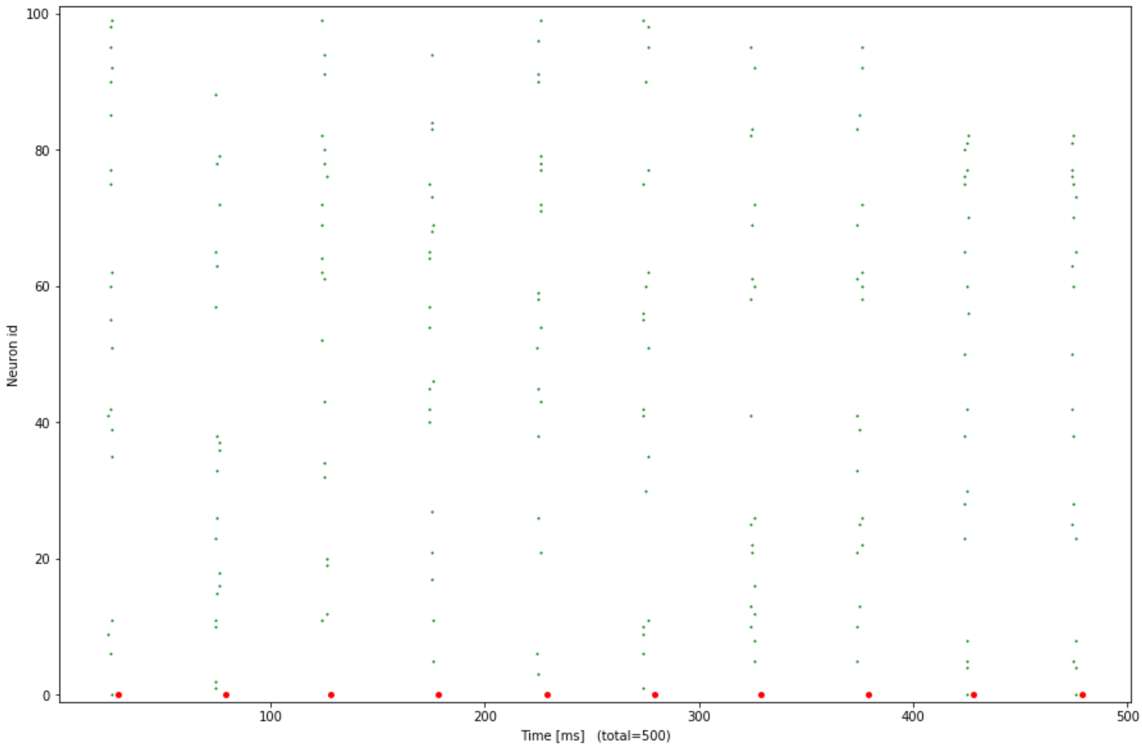
```
0 5000000
```



Horn spikes - start

In [2785]:
```
# start_t = int(total_t*0.999)
start_t = 0
end_t = 500
# end_t = total_t
print(start_t, total_t)
fig = plt.figure(figsize=(15, 10))
ax = plt.subplot(1, 1, 1)
plot_in_range(fig, ax, in_spikes,  start_t=start_t, end_t=end_t, markersize=4, color='green')
plot_in_range(fig, ax, h_spikes,  start_t=start_t, end_t=end_t, markersize=10, color='red')

ax.set_ylabel('Neuron id')
ax.set_xlabel('Time [ms]   (total={})'.format(end_t - start_t))
ax.set_ylim(-1, args.nAL + 1)

plt.show()
```

```
0 5000000
```

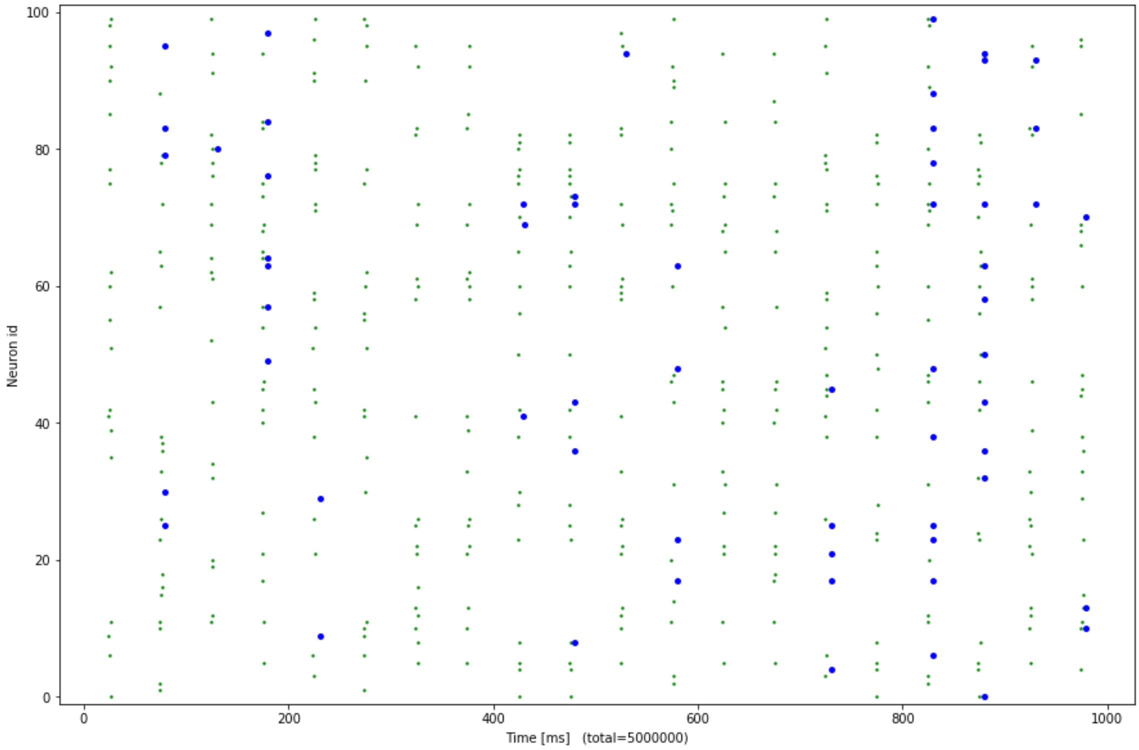


Decision neurons - start

In [2786]:

```
# start_t = int(total_t*0.999)
start_t = 0
end_t = 1000
print(start_t, end_t)
fig = plt.figure(figsize=(15, 10))
ax = plt.subplot(1, 1, 1)

plot_in_range(fig, ax, in_spikes,  start_t=start_t, end_t=end_t, markersize=5, color='green')
plot_in_range(fig, ax, out_spikes, start_t=start_t, end_t=end_t, markersize=10)


ax.set_ylabel('Neuron id')
ax.set_xlabel('Time [ms]   (total={})'.format(total_t - start_t))
ax.set_ylim(-1, 101)

plt.show()
```

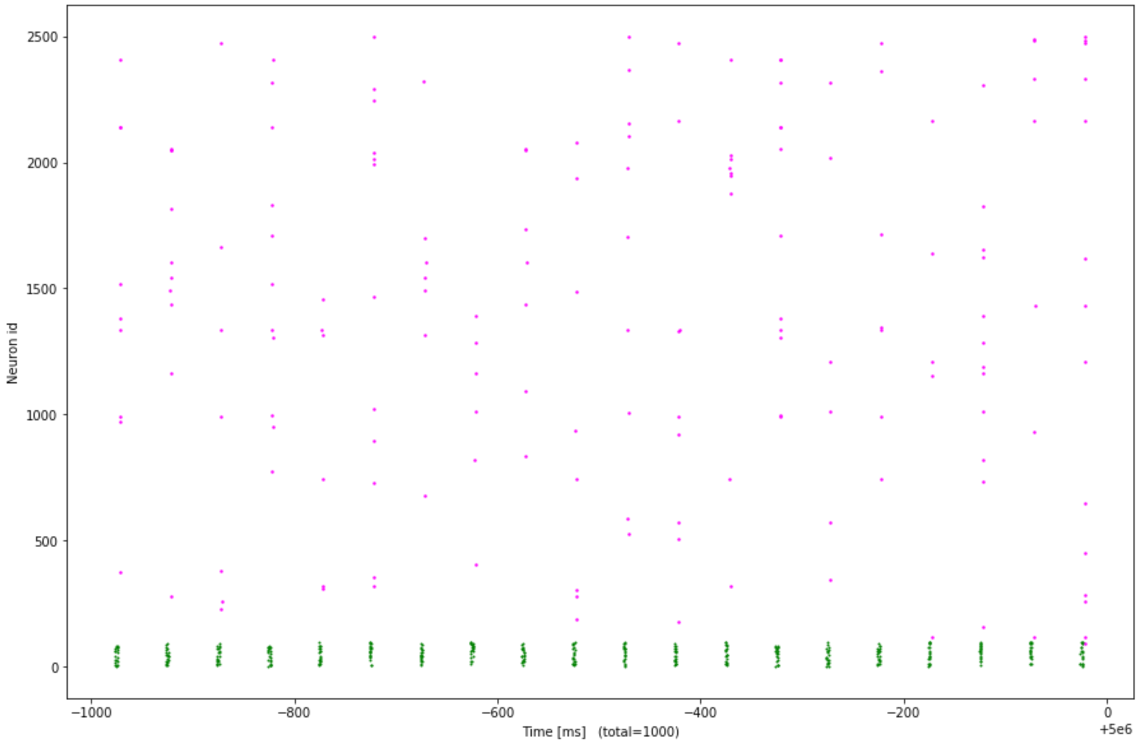
```
0 1000
```



Kenyon cells - end

In [2787]:
```
start_t = total_t - 1000
end_t = total_t
print(start_t, total_t)
fig = plt.figure(figsize=(15, 10))
ax = plt.subplot(1, 1, 1)
plot_in_range(fig, ax, in_spikes,  start_t=start_t, end_t=end_t, mar
kersize=4, color='green')
plot_in_range(fig, ax, k_spikes,  start_t=start_t, end_t=end_t, mark
ersize=5, color='magenta')

ax.set_ylabel('Neuron id')
ax.set_xlabel('Time [ms]   (total={})'.format(total_t - start_t))

plt.show()
```

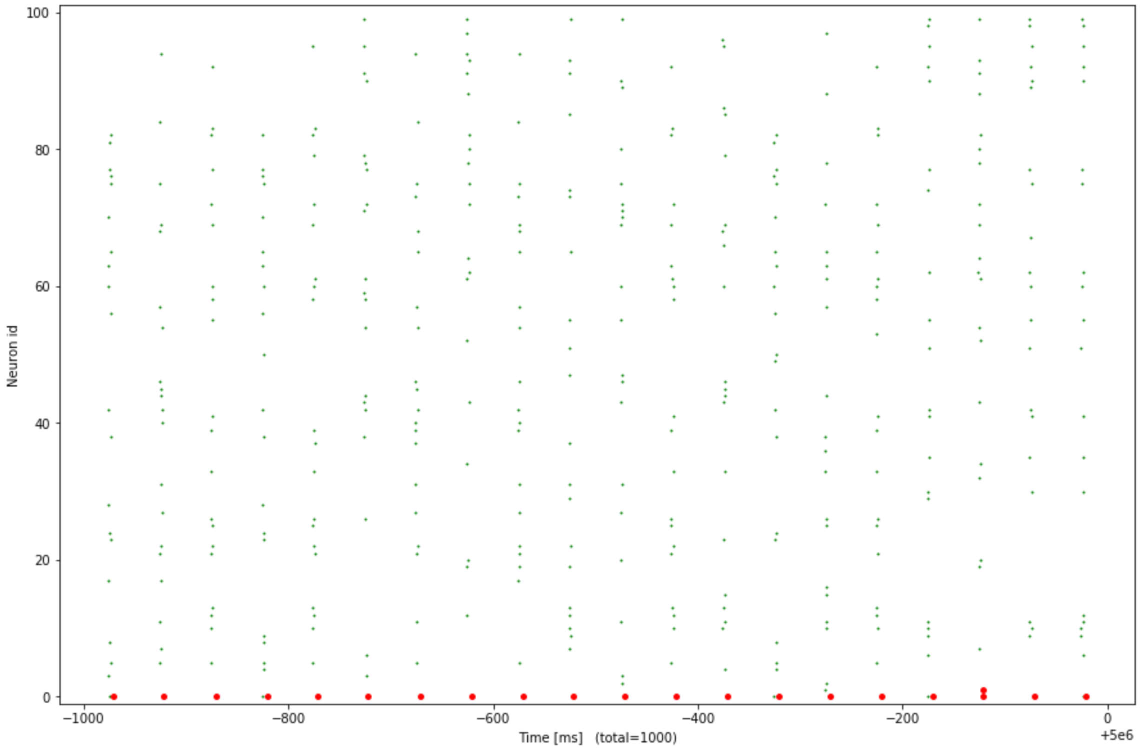
```
4999000 5000000
```



Lateral Horn - end

In [2788]:
```
start_t = total_t - 1000
end_t = total_t
print(start_t, total_t)
fig = plt.figure(figsize=(15, 10))
ax = plt.subplot(1, 1, 1)
plot_in_range(fig, ax, in_spikes,  start_t=start_t, end_t=end_t, markersize=4, color='green')
plot_in_range(fig, ax, h_spikes,  start_t=start_t, end_t=end_t, markersize=10, color='red')

ax.set_ylabel('Neuron id')
ax.set_xlabel('Time [ms]   (total={})'.format(total_t - start_t))
ax.set_ylim(-1, 101)

plt.show()
```

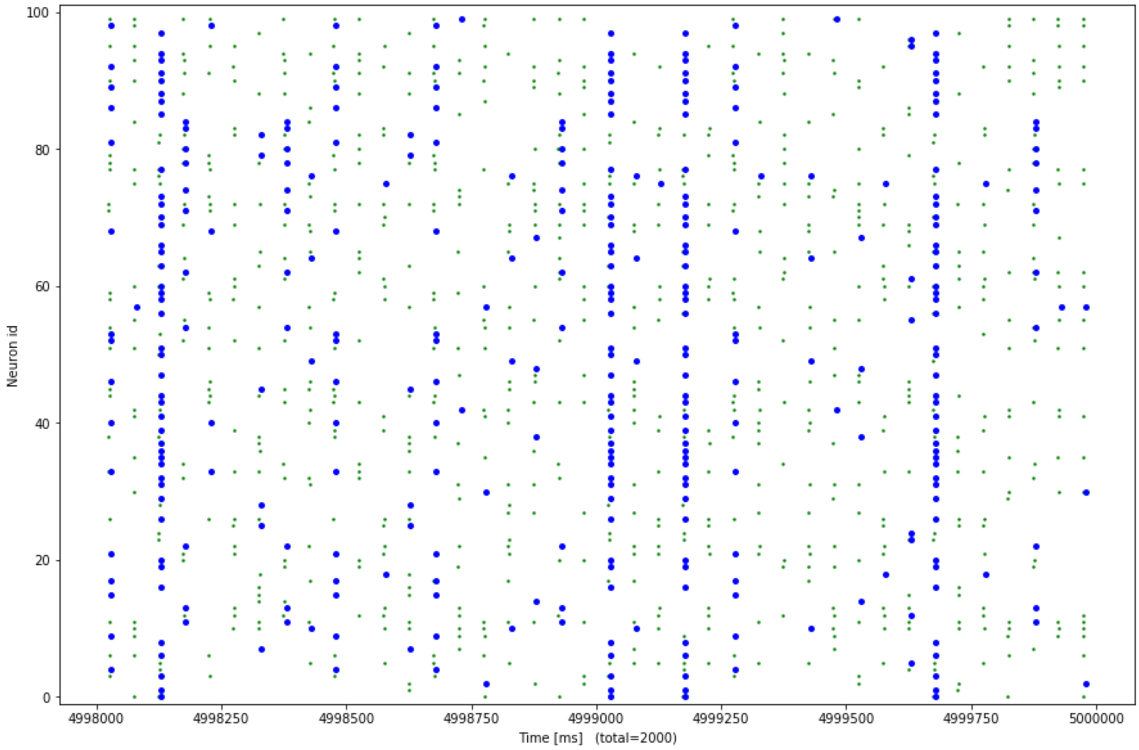
```
4999000 5000000
```



Decision neurons - end

In [2789]:
```
# start_t = int(total_t*0.999)
total_t = int(data['sim_time'])
start_t = total_t - 2000
print(start_t, total_t)
fig = plt.figure(figsize=(15, 10))
ax = plt.subplot(1, 1, 1)

plot_in_range(fig, ax, in_spikes,  start_t=start_t, end_t=total_t, m
arkersize=5, color='green')
plot_in_range(fig, ax, out_spikes, start_t=start_t, end_t=total_t, m
arkersize=10)


ax.set_ylabel('Neuron id')
ax.set_xlabel('Time [ms]   (total={})'.format(total_t - start_t))
ax.set_ylim(-1, 101)

plt.show()
```

```
4998000 5000000
```



In [2790]:
```
# start_t = int(total_t*0.999)
if bool(0):
    start_t = 19000
    end_t = start_t + 1000
    print(start_t, total_t)
    fig = plt.figure(figsize=(15, 10))
    ax = plt.subplot(1, 1, 1)

    plot_in_range(fig, ax, in_spikes,  start_t=start_t, end_t=end_t,
markersize=5, color='green')
    plot_in_range(fig, ax, out_spikes, start_t=start_t, end_t=end_t,
markersize=10)


    ax.set_ylabel('Neuron id')
    ax.set_xlabel('Time [ms]   (total={})'.format(total_t - start_t)
)
    ax.set_ylim(-1, 101)

    plt.show()
```

In [2790]:

In [2790]:

Decision neuron population rate

In [2790]:

In [2791]:
```
print(out_var)
```

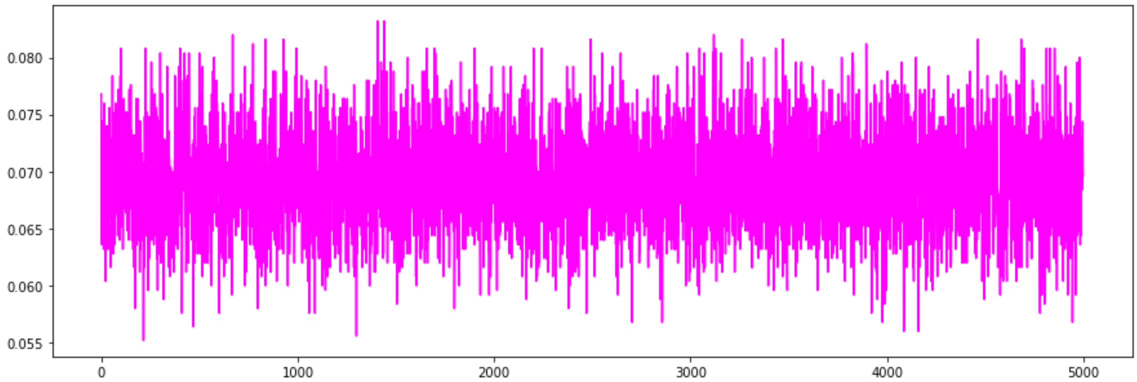
```
0.385436112764
```

In [2791]:

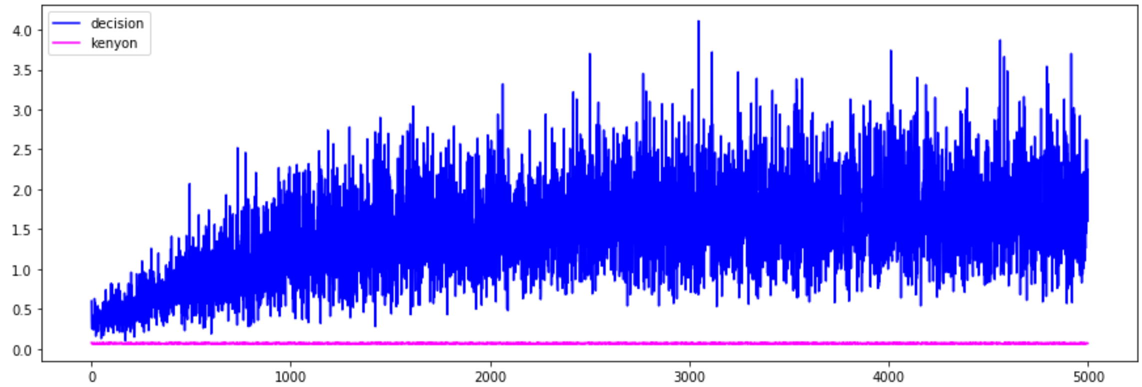
Kenyon cell population rate

In [2792]:
```
k_rates = pop_rate_per_second(k_spikes, total_t, dt)
```

In [2793]:
```
plt.figure(figsize=(15, 5))
plt.plot(k_rates, color='magenta')
plt.show()
```



In [2794]:
```
plt.figure(figsize=(15, 5))
plt.plot(out_rates, label='decision', color='blue')
plt.plot(k_rates, label='kenyon', color='magenta')
plt.legend()
plt.show()
```



In [2794]:

In [2794]:

In [2795]:
```
args = data['args'].item()
print(args)
```

```
Namespace(backend='genn', gScale=0.025, inactiveScale=0.1, nAL=100,
nDN=100, nKC=2500, nLH=20, nPatternsAL=10, nSamplesAL=10000, probAL=
0.2, probAL2KC=0.15, probAL2LH=0.5, probKC2DN=0.2, probNoiseSamplesA
L=0.1, randomizeSamplesAL=True, renderSpikes=False, w2s=0.0025)
```

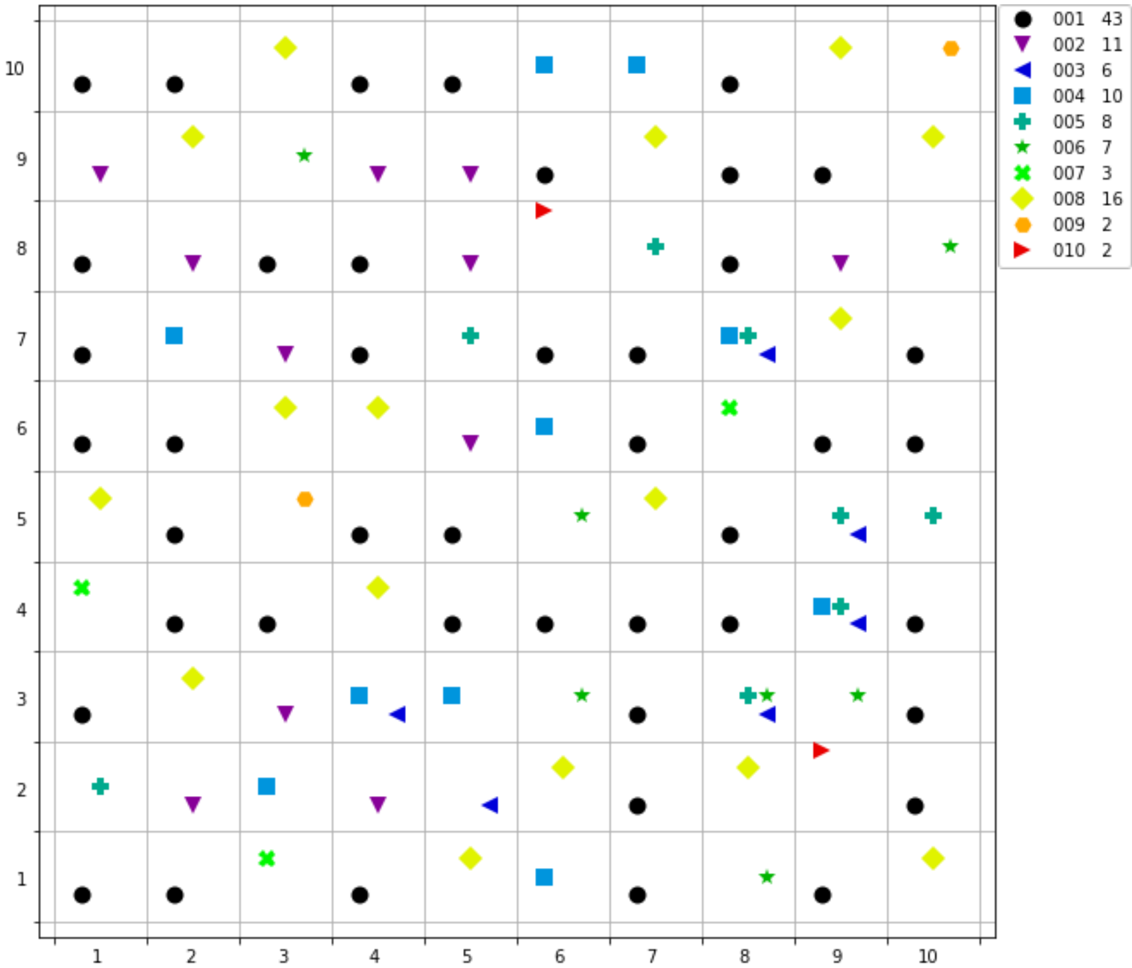
In [2803]:
```
config = {
    'n_patterns': args.nPatternsAL,
    'n_samples': args.nSamplesAL,
    'time_per_sample': float(data['sample_dt']),
}
start_idx =  args.nPatternsAL * args.nSamplesAL - (1000 * args.nPatt
ernsAL)
start_t = start_idx * float(data['sample_dt'])
end_t = total_t
print(config)
print(start_idx)
print(start_t, end_t)
```

```
{u'time_per_sample': 50.0, u'n_patterns': 10, u'n_samples': 10000}
90000
4500000.0 5000000
```

In [2804]:
```
act_neurons = active_neurons_per_pattern(out_spikes, start_t, end_t,
                data['sample_indices'], start_idx, config)
```

In [2805]:
```
# gr = matplotlib.cm.gist_rainbow
gr = matplotlib.cm.nipy_spectral
cmap_dc = gr.N//args.nPatternsAL
cmap = np.array([gr(i*cmap_dc) for i in range(args.nPatternsAL)] )
# print(help(cmap))
#
cmap[:, 3] = 1.0
```

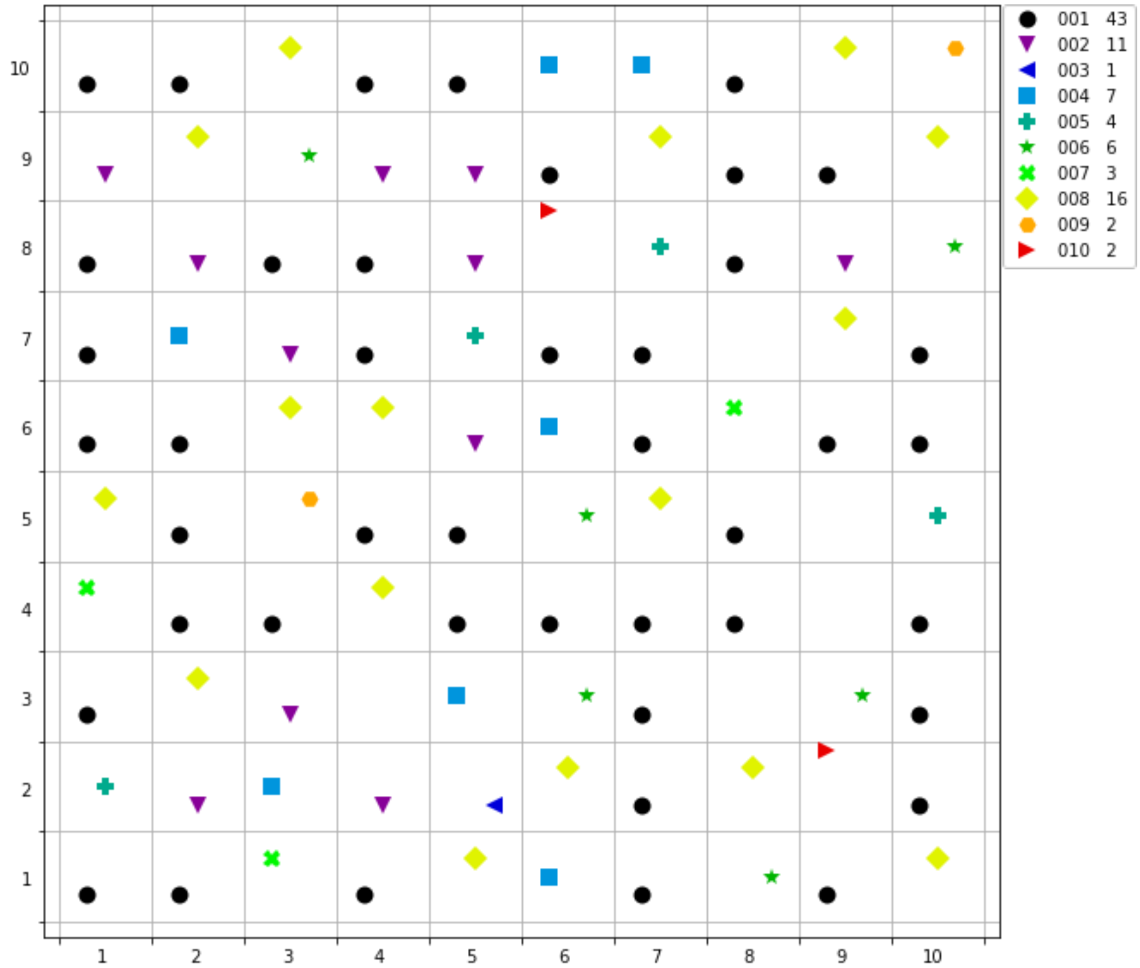
In [2806]:
```
markers = ['o', 'v', '<', 's', 'P', '*', 'X', 'D', 'H', '>']
plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1)

pats_per_neuron = {nid: set() for nid in range(args.nDN)}
first_times = [True for _ in act_neurons]

for pat_id in act_neurons:
    color = cmap[pat_id]
    marker = markers[pat_id]
    for n_id in act_neurons[pat_id]:
        pats_per_neuron[n_id] |= set([pat_id])
        label = "{:03d}   {}".format(pat_id + 1, len(act_neurons[pat
_id])) if first_times[pat_id] else None
        if first_times[pat_id]:
            first_times[pat_id] = False
        x = n_id % 10
        y = n_id // 10
        dx = ((pat_id % 3) - 1) * 0.2
        dy = ((pat_id // 3) - 1) * 0.2
        plt.plot(x+dx, y+dy, '.', markerfacecolor=color,
                 marker=marker, markeredgewidth=0, markersize=10,
                 label=label)

ax.set_xticks([i-0.5 for i in range(11)])
ax.set_yticks([i-0.5 for i in range(11)])
ax.set_xticklabels([i for i in range(1, 11)])
ax.set_yticklabels([i for i in range(1, 11)])

dx = 0.25
dy = 0.0
offset = matplotlib.transforms.ScaledTranslation(dx, dy, fig.dpi_sca
le_trans)
for label in ax.xaxis.get_majorticklabels():
    label.set_transform(label.get_transform() + offset)

dx = 0
dy = 0.25
offset = matplotlib.transforms.ScaledTranslation(dx, dy, fig.dpi_sca
le_trans)
for label in ax.yaxis.get_majorticklabels():
    label.set_transform(label.get_transform() + offset)

handles, labels = ax.get_legend_handles_labels()
labels, handles = zip(*sorted(zip(labels, handles), key=lambda t: t[
0]))
ax.legend(handles, labels, bbox_to_anchor=(1.15, 1.01))

# plt.legend()
plt.grid()
plt.show()
```



001 43
002 11
003 6
004 10
005 8
006 7
007 3
008 16
009 2
010 2

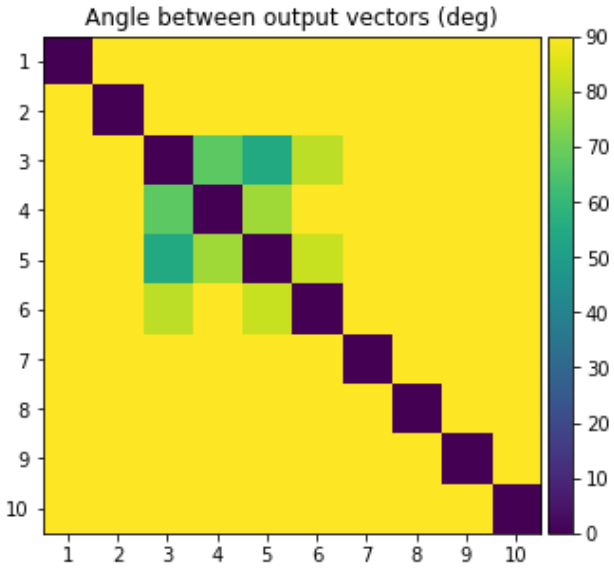
In [2807]:
```
markers = ['o', 'v', '<', 's', 'P', '*', 'X', 'D', 'H', '>']
plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1)

first_times = [True for _ in act_neurons]
unique_act_neurons = {pat_id: [] for pat_id in act_neurons}
for n_id in pats_per_neuron:
    if len(pats_per_neuron[n_id]) != 1:
        continue
    for pat_id in pats_per_neuron[n_id]:
        unique_act_neurons[pat_id].append(n_id)


no_unique = 0
for pat_id in unique_act_neurons:
    no_unique += (len(unique_act_neurons[pat_id]) > 0)

no_unique = (no_unique == 0)
print("There are no neurons responding uniquely to an input pattern?
%s"%no_unique)

if not no_unique:
    for pat_id in unique_act_neurons:
        color = cmap[pat_id]
        marker = markers[pat_id]
        for n_id in unique_act_neurons[pat_id]:

            label = "{:03d}   {}".format(pat_id + 1, len(unique_act_
neurons[pat_id])) if first_times[pat_id] else None
            if first_times[pat_id]:
                first_times[pat_id] = False
            x = n_id % 10
            y = n_id // 10
            dx = ((pat_id % 3) - 1) * 0.2
            dy = ((pat_id // 3) - 1) * 0.2
            plt.plot(x+dx, y+dy, '.', markerfacecolor=color,
                     marker=marker, markeredgewidth=0, markersize=10
,
                     label=label)

    ax.set_xticks([i-0.5 for i in range(11)])
    ax.set_yticks([i-0.5 for i in range(11)])
    ax.set_xticklabels([i for i in range(1, 11)])
    ax.set_yticklabels([i for i in range(1, 11)])

    dx = 0.25
    dy = 0.0
    offset = matplotlib.transforms.ScaledTranslation(dx, dy, fig.dpi
_scale_trans)
    for label in ax.xaxis.get_majorticklabels():
        label.set_transform(label.get_transform() + offset)

    dx = 0
    dy = 0.25
    offset = matplotlib.transforms.ScaledTranslation(dx, dy, fig.dpi
_scale_trans)
    for label in ax.yaxis.get_majorticklabels():
        label.set_transform(label.get_transform() + offset)

    handles, labels = ax.get_legend_handles_labels()
    labels, handles = zip(*sorted(zip(labels, handles), key=lambda t
: t[0]))
    ax.legend(handles, labels, bbox_to_anchor=(1.15, 1.01))

    plt.grid()
    plt.show()
```

```
There are no neurons responding uniquely to an input pattern? False
```



In [2807]:

In [2808]:
```
angles = np.zeros((args.nPatternsAL, args.nPatternsAL))
dots = np.zeros((args.nPatternsAL, args.nPatternsAL))
weights = np.zeros((args.nPatternsAL, args.nPatternsAL))
divs = np.zeros((args.nPatternsAL, args.nPatternsAL))
for i in range(args.nPatternsAL):
    out_i = np.zeros(args.nDN)
    out_i[list(act_neurons[i])] = 1
    w_i = np.sqrt(np.dot(out_i, out_i))

    for j in range(args.nPatternsAL):
        out_j = np.zeros(args.nDN)
        out_j[list(act_neurons[j])] = 1
        w_j = np.sqrt(np.dot(out_j, out_j))
        dots[i, j] = np.dot(out_i, out_j)
        weights[i, j] = (w_i*w_j)
        divs[i, j] = dots[i, j]/weights[i, j]
#         divs[np.isnan(divs)] = 0.
        angles[i, j] = np.rad2deg( np.arccos( divs[i, j] ) )

#         angles[] = 0.
#         diff = out_i - out_j
#         angles[i, j] = np.sqrt(np.dot(diff, diff))

angles[np.isclose(divs, 1.0)] = 0.

fig = plt.figure(figsize=(5, 5))
ax = plt.subplot(1, 1, 1)
im = plt.imshow(angles, interpolation='none')
div = make_axes_locatable(ax)
cax = div.append_axes('right', size='5%', pad=0.05)
cbar = plt.colorbar(im, cax=cax)
ax.set_xticks(range(args.nPatternsAL))
ax.set_yticks(range(args.nPatternsAL))
ax.set_xticklabels(['{}'.format(i+1) for i in range(args.nPatternsAL
)])
ax.set_yticklabels(['{}'.format(i+1) for i in range(args.nPatternsAL
)])
ax.set_title('Angle between output vectors (deg)')
plt.show()
```



In [2808]:

In [2808]:

In [2802]:
```
[3] * 10
```
Out[2802]: [3, 3, 3, 3, 3, 3, 3, 3, 3, 3]

In [2802]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: