In [1]:
```python
# import the required packages
import numpy as np
import aphla as ap
import datetime, scipy
import matplotlib.pylab as plt
```

In [2]:
```python
print "aphla version:", ap.__version__, "@ %s" % str(datetime.datetime.now())
```

```
aphla version: 0.3.0b1 @ 2012-04-02 16:33:27.829074
```

In [3]:
```python
# initialize for the NSLS2 Virtual Storage Ring.
# it loads lattice/elements with their EPICS PV names and the
# orbit response matrix for orbit correction
ap.initNSLS2VSR()
```

```
Creating lattice from '/home/lyyang/devel/venv/local/lib/python2.7/site-
packages/aphla-0.3.0b1-py2.7.egg/aphla/conf/nsls2.csv'
Using ORM: /home/lyyang/.hla/orm.hdf5
```

In [4]:
```python
# since we are using virtual accelerator (Tracy-II/EPICS)
# we have the twiss available. Otherwise we can load the twiss data
# from an external file
ap.initNSLS2VSRTwiss()
```

```
Elements in lattice: 1389
```

In [5]:
```python
# there are several set of lattice structure available
ap.machines.lattices()
```

Out[5]:
```
{u'LTB': u'/home/lyyang/devel/venv/local/lib/python2.7/site-packages/aphla-0.3.0b1-
py2.7.egg/aphla/conf/nsls2.csv',
 u'LTD1': u'/home/lyyang/devel/venv/local/lib/python2.7/site-packages/aphla-0.3.0b1-
py2.7.egg/aphla/conf/nsls2.csv',
 u'LTD2': u'/home/lyyang/devel/venv/local/lib/python2.7/site-packages/aphla-0.3.0b1-
py2.7.egg/aphla/conf/nsls2.csv',
 u'SR': u'/home/lyyang/devel/venv/local/lib/python2.7/site-packages/aphla-0.3.0b1-
py2.7.egg/aphla/conf/nsls2.csv'}
```

In [6]:
```python
# get all elements with name pattern 'BPM', if it is a family name,
# e.g. 'BPM', 'QUAD', it returns all the elements of that type.
# similar for 'C02' the elements in cell 2, 'G2' the elements on girder 2.
# 'A' elements with symmetry A
bpms = ap.getElements('BPM')
```

In [23]:
```python
# print a list of elements, the first 5 only
print bpms[:5]
```

```
[PH1G2C30A:BPM @ sb=4.935000, PH2G2C30A:BPM @ sb=7.460020, PM1G4C30A:BPM @
sb=13.144600, PM1G4C30B:BPM @ sb=15.377300, PL2G6C30B:BPM @ sb=20.247200]
```

In [9]:
```python
print bpms[0].name, bpms[0].cell, bpms[0].girder, bpms[0].length
```

```
PH1G2C30A C30 G2 0.0
```

In [10]:
```python
b0 = bpms[0]
```

In [13]:
```python
# configured in channel finder or csv file, the element can have fields.
# typically for BPM and corrector, it has 'x', 'y', and quadrupole 'k1'
```

```
b0.fields()
```

Out[13]: ['y', 'x']

In [14]:
```
t0 = ap.getElements('HCOR')[0]
```

In [15]:
```
t0.fields()
```

Out[15]: ['x']

In [21]:
```
# all pvs of this HCOR element
print t0.name, t0.pv()
```

```
CXHG2C30A ['SR:C30-MG:G02A{HCor:H}Fld-SP', 'SR:C30-MG:G02A{HCor:H}Fld-I']
```

In [18]:
```
# get pvs associated with field 'x'
t0.pv(field='x')
```

Out[18]: ['SR:C30-MG:G02A{HCor:H}Fld-SP', 'SR:C30-MG:G02A{HCor:H}Fld-I']

In [19]:
```
# the readback pv
t0.pv(field='x', handle='readback')
```

Out[19]: ['SR:C30-MG:G02A{HCor:H}Fld-I']

In [22]:
```
#the setpoint PV
t0.pv(field='x', handle='setpoint')
```

Out[22]: ['SR:C30-MG:G02A{HCor:H}Fld-SP']

In [24]:
```
# this element belongs to the following groups,
# if call getElements() with any of these group name, it should be
# in the return list
ap.getGroups(t0.name)
```

Out[24]: ['C30', 'G2', 'A', 'HCOR']

In [26]:
```
# if we want BPMs in cell 10
bpmc10 = ap.getGroupMembers(['C10', 'BPM'])
print bpmc10
```

```
[PH1G2C10A:BPM @ sb=268.921000, PH2G2C10A:BPM @ sb=271.446000, PM1G4C10A:BPM @
sb=277.131000, PM1G4C10B:BPM @ sb=279.363000, PL2G6C10B:BPM @ sb=284.233000,
PL1G6C10B:BPM @ sb=286.797000]
```

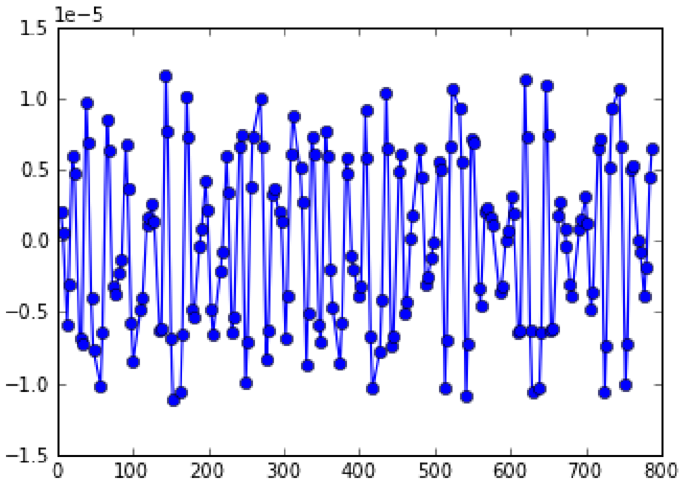
In [28]:
```
# see what is the orbit in cell 10
# print out the bpm name, s location
for bpm in bpmc10: print bpm.name, bpm.sb, bpm.x, bpm.y
```

```
PH1G2C10A 268.921 9.99539289913e-06 4.41133317807e-06
PH2G2C10A 271.446 6.63825383355e-06 5.94967813001e-06
PM1G4C10A 277.131 -8.30429941132e-06 2.10502660486e-06
PM1G4C10B 279.363 -6.1320661967e-06 9.47593322926e-07
PL2G6C10B 284.233 3.20179486237e-06 -5.79074844962e-07
PL1G6C10B 286.797 3.84843170539e-06 -1.73850197563e-06
```

In [30]:
```
bpms = ap.getElements('BPM')
s = [ b.sb for b in bpms]
x = [ b.x for b in bpms]
```

```
plt.plot(s, x, '-o')
```

Out[30]: [<matplotlib.lines.Line2D at 0x2997690>]

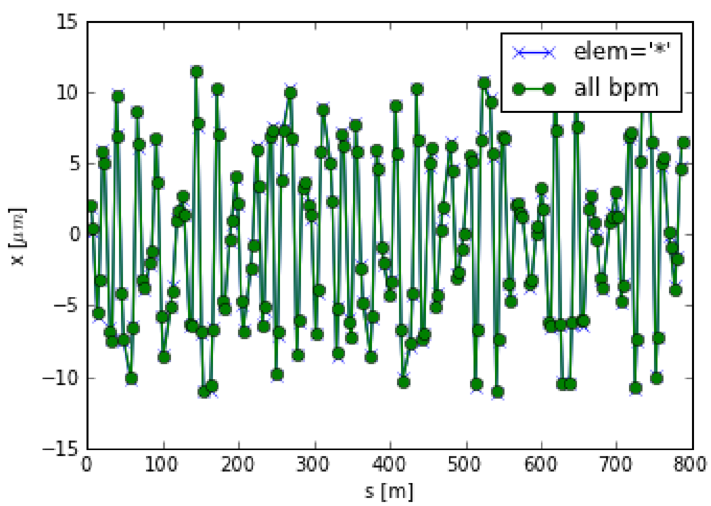


In [31]:
```
obt0 = ap.getOrbit('*', spos = True)
```

In [32]:
```
obt1=ap.getOrbit(spos=True)
```

In [34]:
```
plt.plot(obt0[:,-1], obt0[:,0]*1e6, '-x', label="elem='*'")
plt.plot(obt1[:,-1], obt1[:,0]*1e6, '-o', label="all bpm")
plt.legend()
plt.xlabel("s [m]")
plt.ylabel(r"x [$\mu m$]")
```

Out[34]: <matplotlib.text.Text at 0x421fbd0>

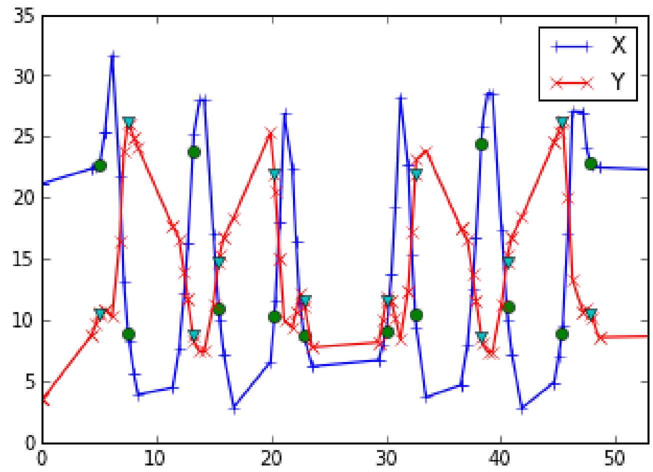


In [6]:
```
s0 = ap.getLocations('*')
beta0=ap.getBeta('*')
s1 = ap.getLocations('BPM')
beta1=ap.getBeta('BPM')
```

In [23]:
```
plt.plot(s0, beta0[:,0], '-+', label='X')
plt.plot(s1, beta1[:,0], 'o')
plt.plot(s0, beta0[:,1], '-x', label='Y')
plt.plot(s1, beta1[:,1], 'v')
```

```
plt.legend()
plt.xlim(s0[0], s0[-1]/15.0)
```

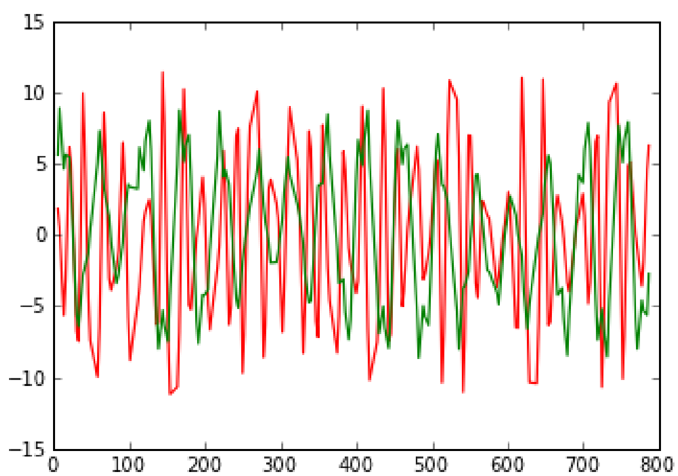
Out[23]: (0.0, 52.7972)



In [9]:
```
# initial orbit
obt0a = ap.getOrbit()
ap.hlalib._reset_trims()
ap.waitStableOrbit(obt0a)
obt0 = ap.getOrbit(spos=True)
plt.plot(obt0[:,-1], obt0[:,0]*1e6, 'r-')
plt.plot(obt0[:,-1], obt0[:,1]*1e6, 'g-')
```

DONE

Out[9]: [<matplotlib.lines.Line2D at 0x41dfa50>]



In [7]:
```
bpms = ap.getElements('BPM')
bpms1 = [b.name for b in bpms if b.cell in ['C02', 'C03', 'C04', 'C20', 'C21', 'C22']
cors = ap.getElements('HCOR') + ap.getElements('VCOR')
cors1 = [c.name for c in cors]
```

In [10]:
```
obt = [ap.getOrbit(spos=True)]
for i in range(20):
    obtt = ap.getOrbit()
    ap.correctOrbit(bpms1, cors1)
    ap.waitStableOrbit(obtt)
    obt.append(ap.getOrbit(spos=True))
```

```
print "DONE"
```

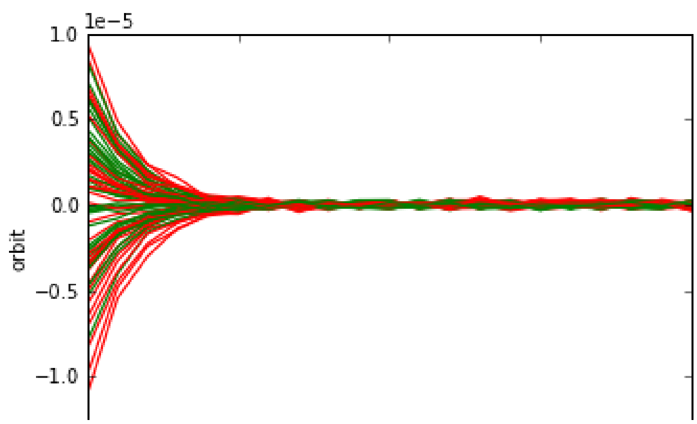
```
Euclidian norm: predicted/realized 1.96771191493e-05 1.9995778432e-05
Euclidian norm: predicted/realized 9.97149406941e-06 9.7716378406e-06
Euclidian norm: predicted/realized 4.9622188598e-06 5.07947399087e-06
Euclidian norm: predicted/realized 2.5624982819e-06 2.70563837776e-06
Euclidian norm: predicted/realized 1.29197338301e-06 1.80506666317e-06
Euclidian norm: predicted/realized 9.33999803598e-07 1.4612792765e-06
Euclidian norm: predicted/realized 6.48603871929e-07 1.2331231848e-06
Euclidian norm: predicted/realized 6.0636446952e-07 1.20503320318e-06
Euclidian norm: predicted/realized 5.41949138201e-07 1.02558359795e-06
Euclidian norm: predicted/realized 5.12791798976e-07 1.12535149732e-06
Failed to reduce orbit distortion, restoring... 1.02558359795e-06 1.12535149732e-06
Euclidian norm: predicted/realized 5.63330770468e-07 1.07127404887e-06
Euclidian norm: predicted/realized 4.65913058085e-07 1.07869554442e-06
Failed to reduce orbit distortion, restoring... 9.31826116171e-07 1.07869554442e-06
Euclidian norm: predicted/realized 6.72817057552e-07 1.32053578979e-06
Euclidian norm: predicted/realized 7.09409644895e-07 9.69700205177e-07
Euclidian norm: predicted/realized 5.98054637206e-07 1.07581560517e-06
Euclidian norm: predicted/realized 5.8748509724e-07 1.42786970504e-06
Failed to reduce orbit distortion, restoring... 1.17497019448e-06 1.42786970504e-06
Euclidian norm: predicted/realized 5.95771184042e-07 1.28976777851e-06
Failed to reduce orbit distortion, restoring... 1.19154236808e-06 1.28976777851e-06
Euclidian norm: predicted/realized 6.08162090189e-07 1.31073706275e-06
Failed to reduce orbit distortion, restoring... 1.21632418038e-06 1.31073706275e-06
Euclidian norm: predicted/realized 6.46123038142e-07 1.5319891126e-06
Failed to reduce orbit distortion, restoring... 1.29224607628e-06 1.5319891126e-06
Euclidian norm: predicted/realized 5.43182942433e-07 1.20725780561e-06
Failed to reduce orbit distortion, restoring... 1.08636588487e-06 1.20725780561e-06
DONE
```

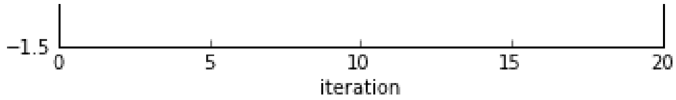
In [16]:

```
# plotting only the 'C02', ..., 'C22' part of the orbit
m, n = np.shape(obt[0])
idx = [i for i in range(m) if bpms[i].cell in ['C02', 'C03', 'C04', 'C20', 'C21', 'C2
print idx
for i in range(m):
    x = [obt[j][i,0] for j in range(len(obt)) if i in idx]
    plt.plot(x, 'r-')
    y = [obt[j][i,1] for j in range(len(obt)) if i in idx]
    plt.plot(y, 'g-')
plt.xlabel("iteration")
plt.ylabel("orbit")
```

```
[12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 120, 121,
122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137]
```

Out[16]: <matplotlib.text.Text at 0x5c29ad0>





In [6]:
```
# a hidden function for HLA development
ap.hlalib._reset_trims()
```

```
DONE
```

In [7]:
```
trims = ap.getElements('HCOR')
t0 = trims[0]
t0.enableTrace('x')
t0.mark('x')
trims[0].x = 1e-6
```

In [8]:
```
trims[0].x = 1e-7
```

In [9]:
```
t0.revert('x')
```

In [10]:
```
print t0.status
```

```
<bound method CaElement.status of CXHG2C30A:HCOR @ sb=5.432500>
```

In [11]:
```
t0.x
```

Out[11]:
```
9.999610513466087e-07
```

In [12]:
```
t0.revert('x')
t0.x
```

Out[12]:
```
0.0
```

In [13]:
```
t0.revert('x')
```

In [14]:
```
t0.x
```

Out[14]:
```
0.0
```

In [ ]:
```

```