Demo of the `electrode` package

This demo is similar to the "SurfacePattern demo: backward - finite - electric" of Roman Schmied, see
http://atom.physik.unibas.ch/people/romanschmied/code/SurfacePattern.php

In [1]:
```
from numpy import *
from matplotlib import pyplot as plt
from scipy import constants
from electrode.transformations import euler_matrix
from electrode import (System,
        PointPixelElectrode, PolygonPixelElectrode,
        PatternValueConstraint, PatternRangeConstraint)

set_printoptions(precision=2)
```

Define two functions to build and optimize the pattern electrode:

In [2]:
```
def hextess(n, points=False):
    """returns a hexagonal pixel electrode with unit radius
    and n pixels per unit length in a hexagonal pattern

    if points is True, each pixel is approximated as a point
    else each pixel is a hexagon"""
    x = vstack(array([[i+j*.5, j*3**.5*.5, 0]
        for j in range(-n-min(0, i), n-max(0, i)+1)])
        for i in range(-n, n+1))/(n+.5)
    if points:
        a = ones((len(x),))*3**.5/(n+.5)**2/2
        return PointPixelElectrode(points=x, areas=a)
    else:
        a = 1/(3**.5*(n+.5)) # edge length
        p = x[:, None, :] + [[[a*cos(phi), a*sin(phi), 0] for phi in
            arange(pi/6, 2*pi, pi/3)]]
        return PolygonPixelElectrode(paths=list(p))
```

In [3]:
```
def threefold(n, h, d, H, nmax=1, points=True):
    """returns a System instance with a single hexagonal rf pixel electrode.
```

```
    The pixel factors (whether a pixel is grounded or at rf) are optimized
    to yield three trapping sites forming an equilateral triangle with

       n pixels per unit length,
       ion separation d,
       ion height h, and
       trapping frequencies with a ratio 2:1:1 (radial being the strongest).

    The effect of a grounded shielding cover at height H is accounted for
    up to nmax components in the expansion in h/H.
    """
    s = System()
    rf = hextess(n, points)
    rf.voltage_rf = 1.
    rf.name = "rf"
    rf.cover_height = H
    rf.nmax = nmax
    s.electrodes.append(rf)

    ct = []
    ct.append(PatternRangeConstraint(min=0, max=1.))
    for p in 0, 4*pi/3, 2*pi/3:
        x = array([d/3**.5*cos(p), d/3**.5*sin(p), h])
        r = euler_matrix(p, pi/2, pi/4, "rzyz")[:3, :3]
        ct.append(PatternValueConstraint(d=1, x=x, r=r,
            v=[0, 0, 0]))
        ct.append(PatternValueConstraint(d=2, x=x, r=r,
            v=2**(-1/3.)*eye(3)*[1, 1, -2]))
    rf.pixel_factors, c = rf.optimize(ct)
    return s, c
```

Create and optmimize the system:

In [4]:
```
points=False
n=50
h=1/8.
d=1/4.
H=25/8.

if True:
    # take a simpler, approximate problem
    n=12
    points=True

s, c = threefold(n, h, d, H, nmax=1, points=points)
```

```
variables: 469
inequalities 938
equalities 23
     pcost       dcost       gap    pres   dres   k/t
 0:  1.5111e-01 -4.9189e+02  5e+02  3e-13  3e-15  1e+00
 1:  1.2944e-01 -4.7416e+01  5e+01  3e-14  4e-15  1e-01
 2: -2.2217e-01 -1.7542e+01  2e+01  3e-13  2e-15  2e-02
 3: -3.3689e-01 -2.7317e+00  2e+00  2e-13  1e-15  3e-03
 4: -3.9776e-01 -2.5075e+00  2e+00  3e-13  1e-15  2e-03
 5: -6.1144e-01 -9.8938e-01  4e-01  2e-13  3e-16  4e-04
```

```
 6: -7.2995e-01 -8.7165e-01  1e-01  8e-14  1e-16  2e-04
 7: -7.2838e-01 -8.5589e-01  1e-01  2e-13  2e-16  1e-04
 8: -7.4559e-01 -8.4467e-01  1e-01  9e-13  2e-16  1e-04
 9: -7.7307e-01 -8.0655e-01  3e-02  3e-13  2e-16  4e-05
10: -7.8831e-01 -7.8882e-01  5e-04  6e-14  1e-16  5e-07
11: -7.8852e-01 -7.8855e-01  3e-05  9e-12  1e-16  3e-08
12: -7.8853e-01 -7.8853e-01  3e-07  2e-12  9e-17  3e-10
Optimal solution found.
```

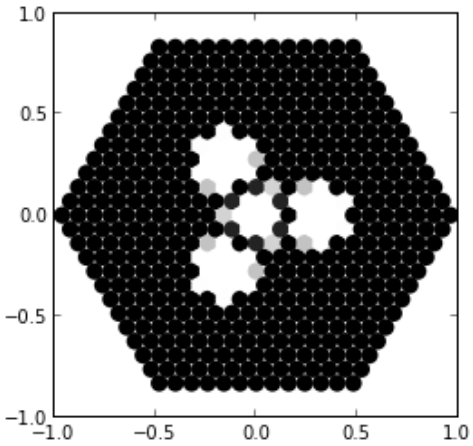
Analysis of the result:

In [5]:
```
x0 = array([d/3**.5, 0, h])
# optimal strength of the constraints
print "c*h**2*c:", h**2
# rf field should vanish
print "rf'/c:", s.electrode("rf").potential(x0, 1)[0][:, 0]/c
# rf curvature should be (2, 1, 1)/2**(1/3)
print "rf''/c:", s.electrode("rf").potential(x0, 2)[0][(0, 1, 2), (0, 1, 2), 0]/c
```

```
c*h**2*c: 0.015625
rf'/c: [ -3.04e-16  -5.26e-16  -3.96e-15]
rf''/c: [-1.59  0.79  0.79]
```

Plot the electrode pattern, white is ground, black/red is rf:

In [6]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_xlim((-1,1))
ax.set_ylim((-1,1))
s.plot_voltages(ax, u=array([1.]))
```



Get some physical quantities for a specific implementation:

In [7]:
```
l = 320e-6 # length scale, hexagon radius
u = 20. # peak rf voltage
o = 2*pi*50e6 # rf frequency
m = 24*constants.atomic_mass # ion mass
q = 1*constants.elementary_charge # ion charge
```

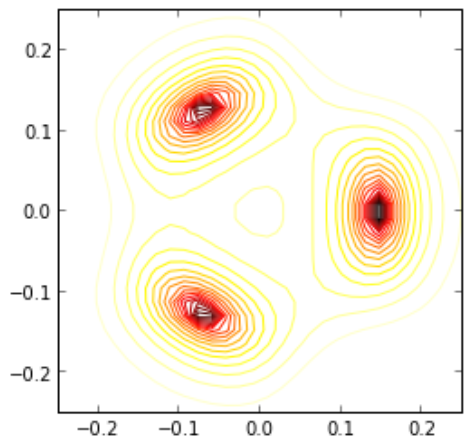
```
for line in s.analyze_static(x0, l=l, u=u, o=o, m=m, q=q):
```

```
u = 20 V, f = 50 MHz, m = 24 amu, q = 1 qe, l = 320 µm, axis=(0, 1, 2)
analyze point: [ 0.14  0.    0.12] ([ 46.19   0.    40.  ] µm)
 minimum is at offset: [ 0.  0.  0.]
 rf, dc potentials: 1.8e-27, 0 (7e-29 eV, 0 eV)
 saddle offset, height: [  8.41e-04  -8.69e-07   3.15e-02], 0.011 (0.00043 eV)
 pp+dc normal curvatures: [ 138.48  138.48  553.92]
 motion is bounded: True
 pseudopotential modes:
  2.341 MHz, [ -1.34e-14   9.35e-01  -3.54e-01]
  2.341 MHz, [  2.49e-14   3.54e-01   9.35e-01]
  4.681 MHz, [  1.00e+00   3.68e-15  -2.80e-14]
  euler angles: [-172.51   90.   -118.2 ]
 mathieu modes:
  2.349 MHz, [ -4.20e-15   7.21e-01  -6.75e-01]
  2.349 MHz, [  1.04e-15   6.77e-01   7.20e-01]
  4.748 MHz, [  9.61e-01   1.97e-15  -3.27e-15]
  euler angles: [-148.88   90.   -166.15]
 heating for 1 nV²/Hz white on each electrode:
  field-noise psd: [  9.95e-41   2.97e-40   1.68e-38] V²/(m² Hz)
  pot-noise psd: 2.20598584281e-19 V²/Hz
  2.341 MHz: ndot=3.989e-25/s, S_E*f=8.992e-33 (V² Hz)/(m² Hz)
  2.341 MHz: ndot=1.686e-24/s, S_E*f=3.8e-32 (V² Hz)/(m² Hz)
  4.681 MHz: ndot=5.164e-27/s, S_E*f=4.656e-34 (V² Hz)/(m² Hz)
```

Plot the horizontal logarithmic pseudopotential at the ion height:

In [8]:
```
n = 30
xyz = mgrid[-d:d:1j*n, -d:d:1j*n, h:h+1]
xyzt = xyz.transpose((1, 2, 3, 0)).reshape((-1, 3))
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.contour(xyz[0].reshape((n,n)), xyz[1].reshape((n,n)),
           log(s.potential(xyzt)).reshape((n,n)),
           20, cmap=plt.cm.hot)
```

Out[8]:
```
<matplotlib.contour.QuadContourSet instance at 0x4dd7f80>
```



Plot the logarithmic pseudopotential and the separatrix in the xz plane.

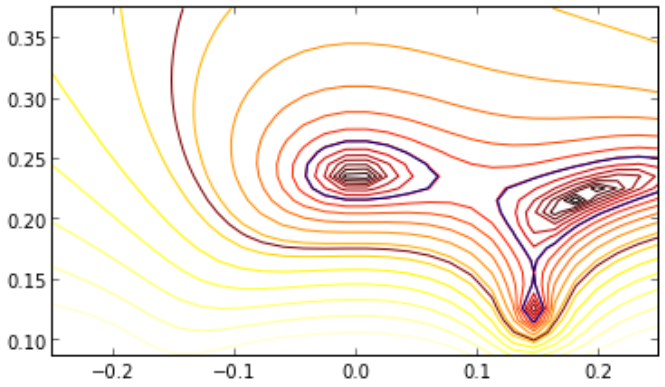
In [9]:
```
xs1, ps1 = s.saddle(x0+1e-2)
```

```
xs0, ps0 = s.saddle([0, 0, .8])
print "main saddle:", xs0, ps0

n = 30
xyz = mgrid[-d:d:1j*n, 0:1, .7*h:3*h:1j*n]
xyzt = xyz.transpose((1, 2, 3, 0)).reshape((-1, 3))
p = s.potential(xyzt)
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.contour(xyz[0].reshape((n,n)), xyz[2].reshape((n,n)),
           log(p).reshape((n,n)),
           20, cmap=plt.cm.hot)
ax.contour(xyz[0].reshape((n,n)), xyz[2].reshape((n,n)),
           log(p).reshape((n,n)),
           [log(ps0), log(ps1)], color="black")
```

```
main saddle: [  6.06e-08   2.99e-07   5.50e-01] 0.166220037083
```

Out[9]: <matplotlib.contour.QuadContourSet instance at 0x4f84830>



In [ ]: