Optimization with PyMC/Python

This notebook demonstrates how to do continuous optimization with PyMC, the pymc.MAP class, and the pymc.MAP.fit method.

In [1]:
```
import pylab as pl
import pymc as mc
import scipy.optimize
```

Powell's Method

This method has worked very well for me in practice, and is reasonably simple theoretically. It was introduced by MJD Powell, in the 1964 paper "An efficient method for finding the minimum of a function of several variables without calculating derivatives", The Computer Journal (1964) 7 (2): 155-162.

In Section 7, of Powell's 1964 paper, there is an example application of his method to a function of three variables:

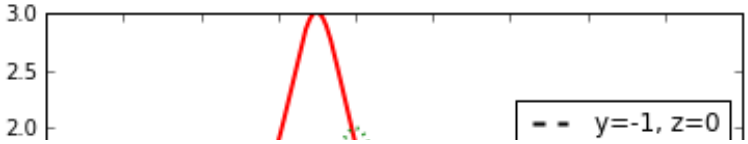
$$f = \frac{1}{1+(x-y)^2} + \sin\left(\frac{1}{2}\pi yz\right) + \exp\left(-\left(\frac{x+z}{y}-2\right)^2\right)$$

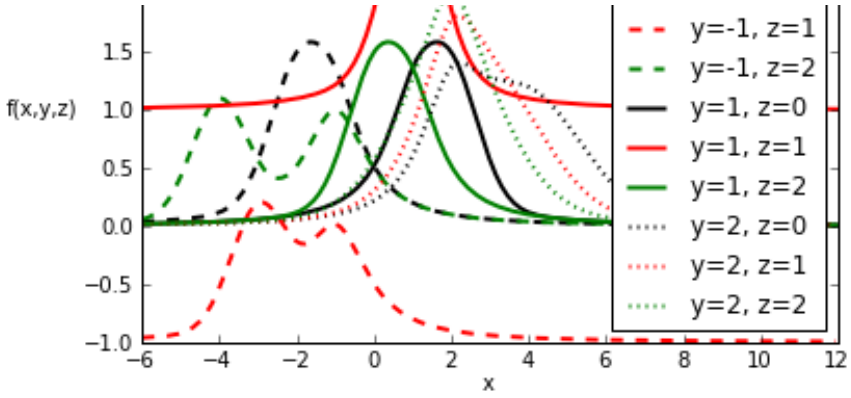
Replicating this is where I will begin.

In [2]:
```
def f(x, y, z):
    result = 0
    result += 1 / (1 + (x - y)**2)
    result += pl.sin(.5 * pl.pi * y * z)
    result += pl.exp(-((x + z) / y - 2)**2)
    return result
```

In [3]:
```
xx = pl.arange(-6,12,.01)
for i, y in enumerate([-1., 1., 2.]):
    for j, z in enumerate([0., 1., 2.]):
        yy = [f(x,y,z) for x in xx]

        plot(xx, yy,
             label='y=%.0f, z=%.0f'%(y,z),
             linestyle=['--', '-', ':'][i],
             color='krg'[j],
             linewidth=2)
xlabel('x')
ylabel('f(x,y,z)', rotation='horizontal')
legend(loc='lower right')
```

Out[3]: <matplotlib.legend.Legend at 0xa69262c>





Here is a (tiny) PyMC model with MAP values corresponding to f. Using PyMC to optimize f isn't necessary, but it is convenient.

In [4]:
```
X = mc.Uninformative('X', value=[0., 1., 2.])
@mc.potential
def objective(X=X):
    return f(*X)
```

In [5]:
```
map = mc.MAP([X, objective])
```

In [6]:
```
map.fit(method='fmin_powell')
```

In [7]:
```
X.value, objective.logp
```

Out[7]:
```
(array([ 0.99866318,  0.9985752 ,  1.00360469]), 2.9999678944996173)
```

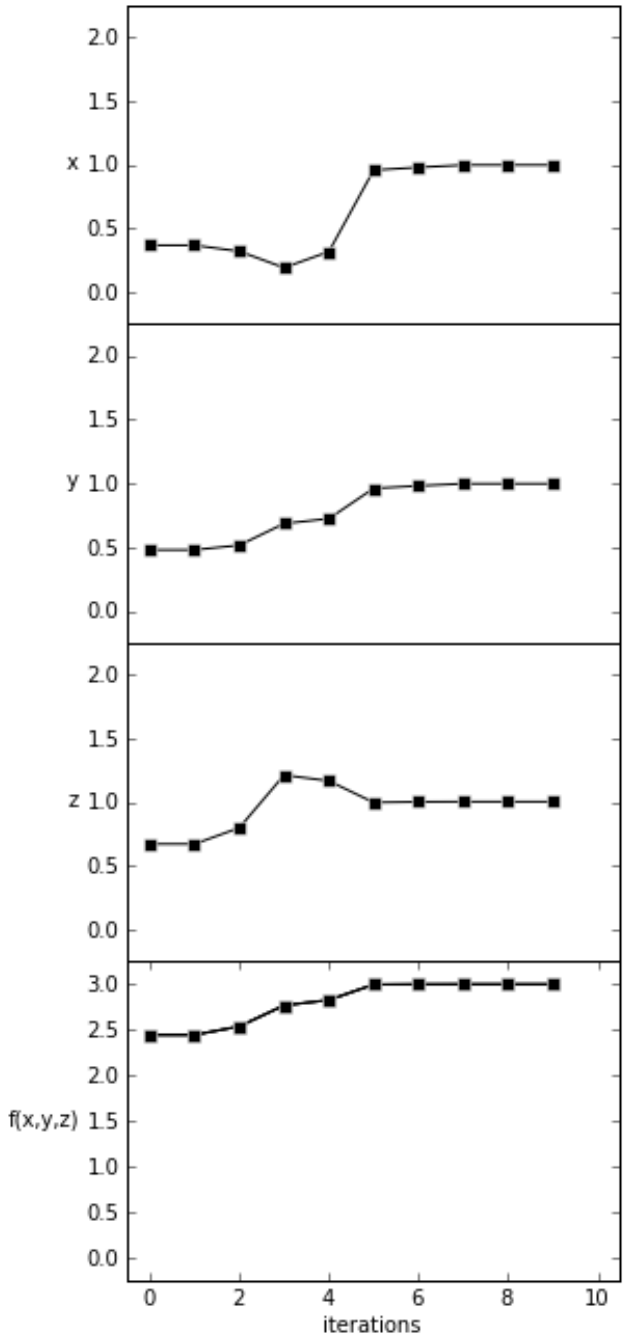
And here is what it is doing, when it does that:

In [8]:
```
def fit_for(iterlim=1, initial_value=[0., 1., 2.], method='fmin_powell'):
    X = mc.Uninformative('X', value=initial_value)
    @mc.potential
    def objective(X=X):
        return f(*X)

    map = mc.MAP([X, objective])
    map.fit(method=method, iterlim=iterlim)
    return dict(X=X.value, objective=objective.logp)
```

In [9]:
```
results = []
for i in range(10):
    results.append(fit_for(i))
```

In [10]:
```
def plot_result(results):
    figure(figsize=(4.25, 11))
    subplots_adjust(hspace=0)
    for j in range(3):
        subplot(4,1,j+1)
        plot([r_i['X'][j] for r_i in results], 'sk-', mec='w')
        ylabel('xyz'[j], rotation=0)
```

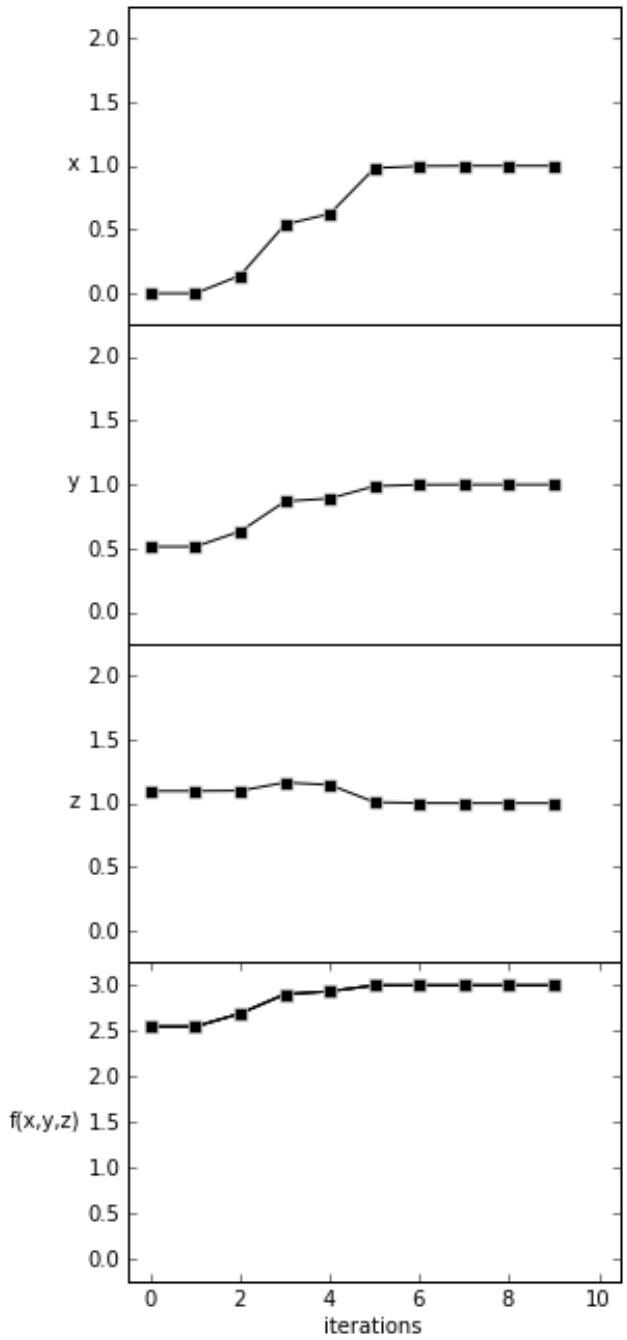
```
        xticks([])
        yticks([0,.5,1, 1.5, 2.])
        axis([-.05*len(results), 1.05*len(results), -.25, 2.25])

        subplot(4,1,4)
        plot([r_i['objective'] for r_i in results], 'sk-', mec='w')
        ylabel('f(x,y,z)', rotation=0)
        yticks([0,.5,1, 1.5, 2., 2.5, 3.])
        axis([-.05*len(results), 1.05*len(results), -.25, 3.25])
        xlabel('iterations')

plot_result(results)
```



It is lovely that this takes 6 iterations to converge, just like the example in Powell's paper.

What about the same thing for different initial values?

In [11]:
```
result = []
for i in range(10):
    result.append(fit_for(i, initial_value=[0., 0., 1.]))

plot_result(result)
```

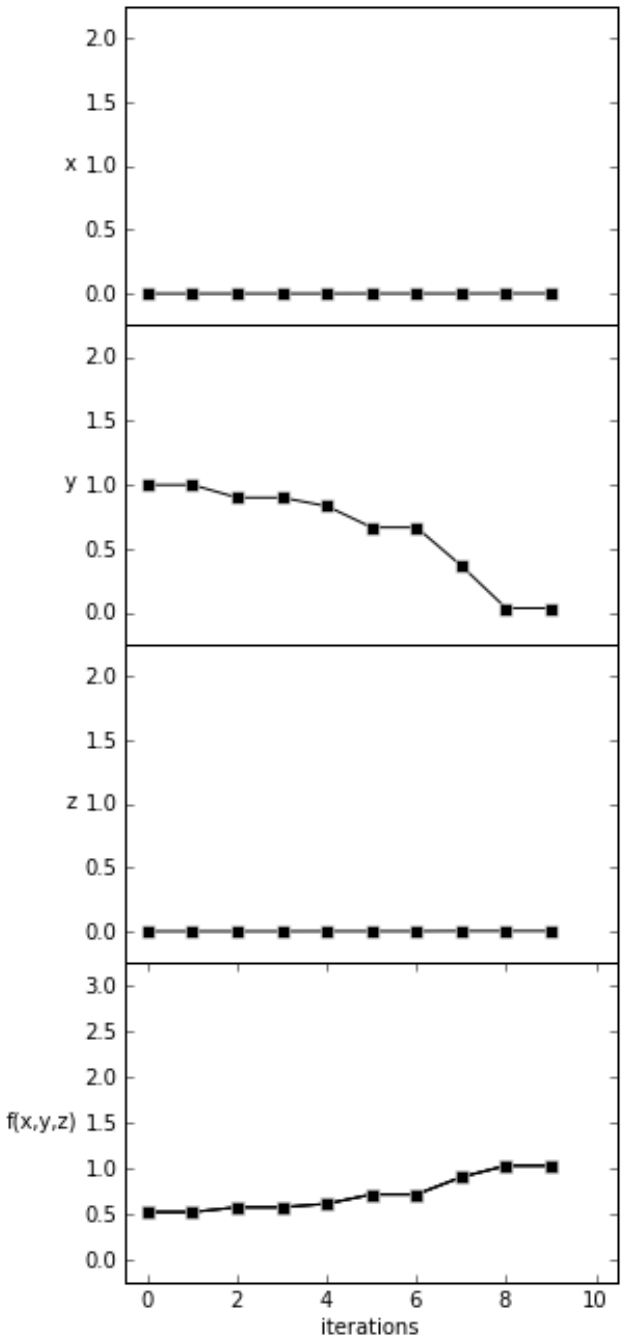


That only took 5 iterations.

Here is one that Powell's gets to a global optimum, but Nelder-Mead does not.

In [12]:
```
result = []
for i in range(10):
```

```
    result.append(fit_for(i, initial_value=[0., 1., 0.], method='fmin'))

plot_result(result)
```



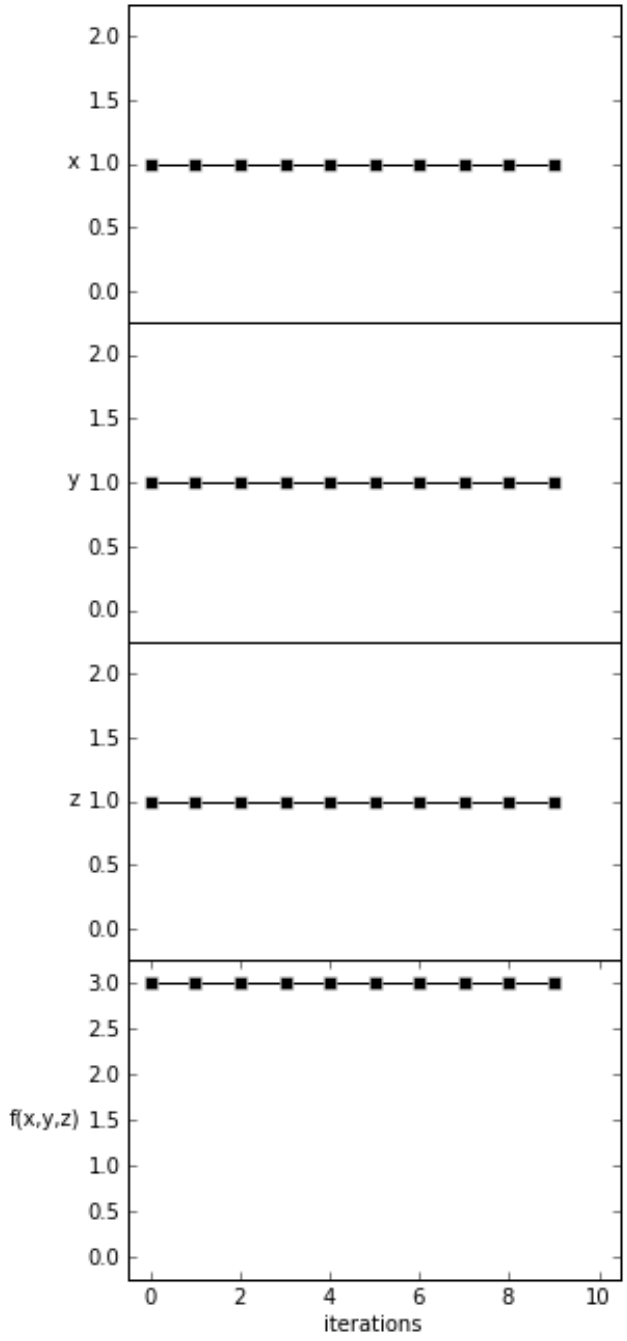
Other methods

PyMC provides access to several algorithms in `scipy.optimize` besides Powell's method.

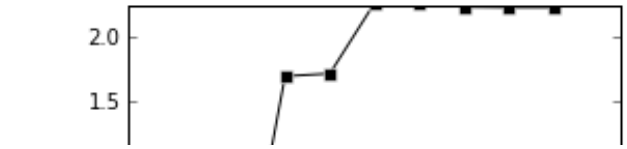
Here are a few to compare:

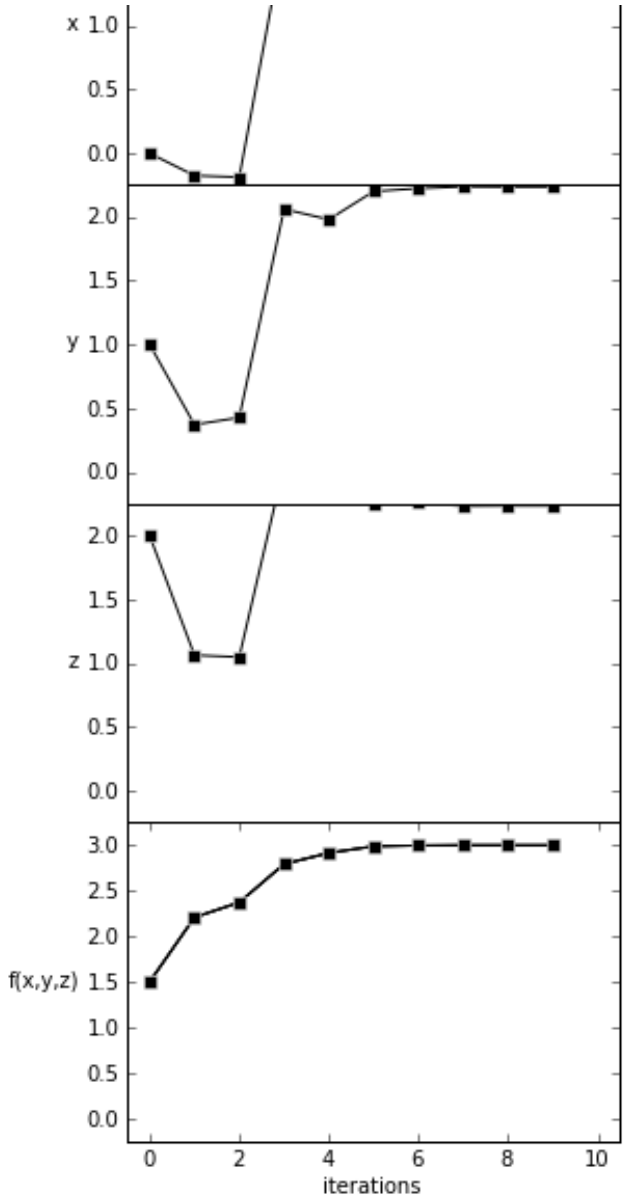
In [13]:
```
result = []
for i in range(10):
    result.append(fit_for(i, initial_value=[0., 1., 2.], method='fmin_l_bfgs_
```

```
plot_result(result)
```

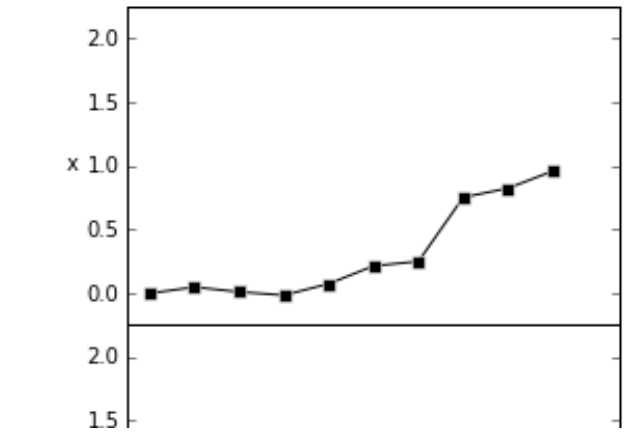


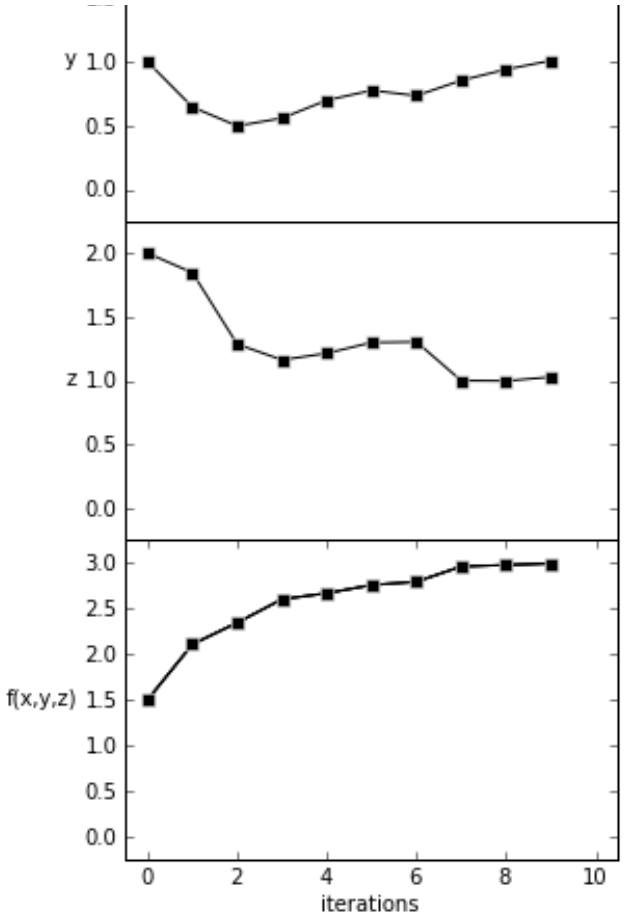
In [14]:
```
result = []
for i in range(10):
    result.append(fit_for(i, initial_value=[0., 1., 2.], method='fmin_ncg'))

plot_result(result)
```



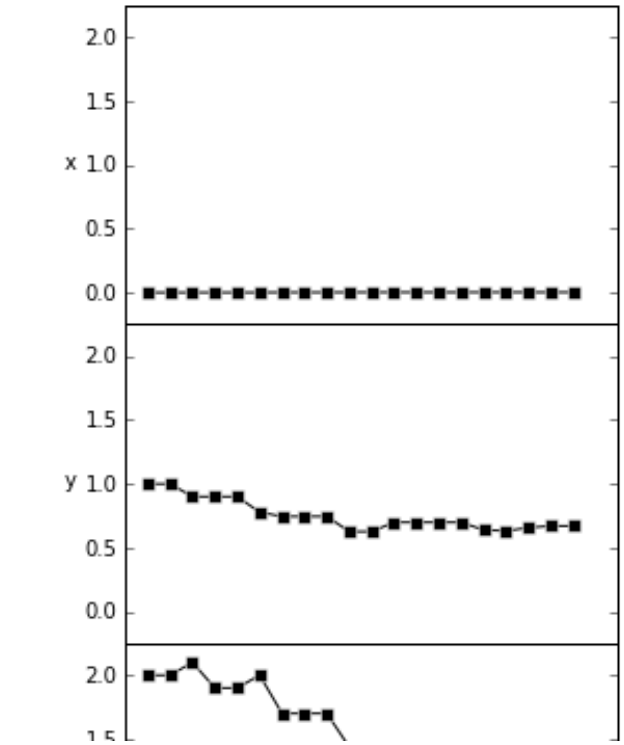


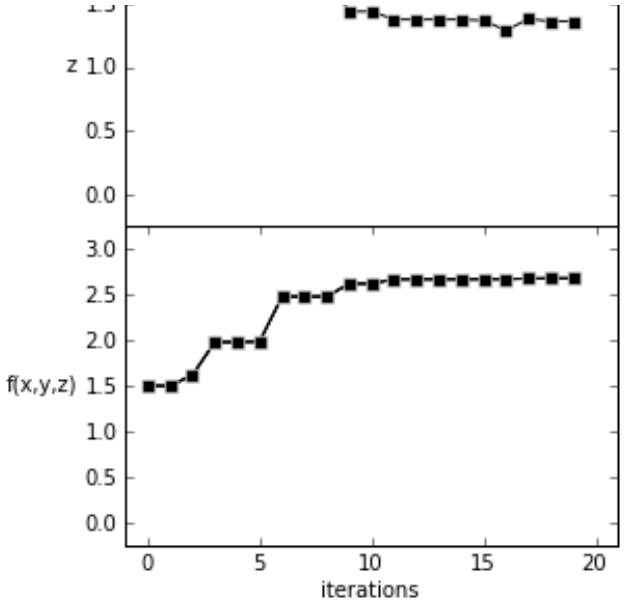
In [15]:
```
result = []
for i in range(10):
    result.append(fit_for(i, initial_value=[0., 1., 2.], method='fmin_cg'))

plot_result(result)
```





In [16]:
```
result = []
for i in range(20):
    result.append(fit_for(i, initial_value=[0., 1., 2.], method='fmin'))

plot_result(result)
```





In [ ]: