GVAR AND LSQFIT

Casey E. Berger

August 21st, 2017

Contents

1 GVAR

1.1 *Dependencies*

No dependencies, but numpy is needed for array manipulation.

1.2 *Basics*

Creating gvar variables

To create a single gvar:

```
import gvar as gv
#create a variable with mean 1.0 and standard deviation 0.2
x = gv.gvar(1.0,0.2)
x_mean = x.mean
x_std = x.sdev
y = gv.gvar(2.0,0.5)
y_mean = y.mean
y_std = y.sdev
print x
print y
print x + y
print x - y
print x*y
print x/y
print y*y
```

Variables created like this can be added, subtracted, multiplied, or divided, and the error will be automatically propagated.

You can also create a gvar by feeding in a string formatted in one of two ways:

```
import gvar as gv
import numpy as np
#create a single gvar by feeding in a string formatted in one of two ways
s = "1.00(20)"
gv_s = gv.gvar(s)
print gv_s
s2 = "1.00 +- 0.2"
gv_s2 = gv.gvar(s2)
print gv_s2
```

Arrays or dictionaries of numbers can also be averaged together to get a gvar:

```
import gvar as gv
import numpy as np
#create a gvar by averaging an array of floats
a = np.array([1.0,2.0,3.0,4.0,5.0])
gv_a = gv.dataset.avg_data(a)
print gv_a
```

The covariance matrix can also be extracted using gvar:

```
import gvar as gv
import numpy as np

#extract a covariance matrix
b =
    np.array([gv.gvar(1.0,0.2),gv.gvar(1.5,0.3),gv.gvar(1.2,0.2),gv.gvar(2.0,0.4)])
cov_b = gv.evalcov(b)
print cov_b
```

In addition to using an array, a dictionary can be used to store the data.

An array of gvar can be generated more quickly in a few different ways:

```
import gvar as gv
import numpy as np

#create an array of gvar by feeding in the means and the covariance
    matrix
data = gv.gvar([1.0, 1.5,1.2,2.0], cov_b)
print data

#create an array of gvar by feeding in the means and the sdevs
data2 = gv.gvar([1.0, 1.5,1.2,2.0], [0.2,0.3,0.2,0.4])
print data2
```

You can also feed in a dictionary or an array.

1.3 *Frequent or Cryptic Error Messages*

2 LSQFIT

2.1 *Dependencies*

This package requires numpy, gvar, and cython to run.

2.2 *Basics*

This fitter works for anything that can be fit using χ^2 minimization.

lsqfit's nonlinear fit function takes input y data in the form of an array of gvar objects, input x data in the form of an array of floats, priors in the form of an array of gvar objects, and a fit function, as well as other optional inputs and fits the function to the data.

Here is an example of a simple exponential fit:

```
import numpy as np
import gvar as gv
import lsqfit

def priors():
   p = dict()
   p['A0'] = [0.1,0.05]
   p['B0'] = [5.,0.5]
   p['C0'] = [3.5,0.5]
   prior = dict()
   for k in p.keys():
      prior[k] = gv.gvar(p[k][0], p[k][1])
   return prior

class fit_function():
   #def __init__(self):
   #  return None
   def A(self,n,p):
      return p['A%s' %n]
   def B(self,n,p):
      return p['B%s' %n]
   def C(self,n,p):
      return p['C%s' %n]
   def func(self,t,p):
      r = 0
      for n in range(0,1):
         An = self.A(n,p)
         Bn = self.B(n,p)
         Cn = self.C(n,p)
         r += An + Bn*np.exp(-Cn*t)
```

```
     return r

x = np.arange(1,5)
y = gv.gvar([0.26,0.099,0.10,0.09],[0.05,0.07,0.005,0.000007])
p = priors()
fitc = fit_function()
fit = lsqfit.nonlinear_fit(data=(x,y),prior=p,fcn=fitc.func)
print fit
```

2.3 *Frequent or Cryptic Error Messages*