posterior

April 22, 2020

In [ ]:
```
import numpy as np
from scipy.interpolate import interp1d
from tqdm import trange

class posterior:

    def lnlike(self, p, arg):

        cl_theo = p[0]*(self.cl_scal(p[1]) + p[2]*self.cl_tens)

#        cl_theo = p[0]*(self.cl_scal_ns1 + p[1]*self.cl_tens)

        l = np.arange(2,self.lmax+1)
        like = -np.sum((2.*l+1.)/2.*(self.cl_data[l]/cl_theo[l]+np.log(cl_theo[l])))

        return like

    def lnprior(self, p, arg):
        if p[0]>0. or p[1]>arg[0] or p[1]<arg[1] or p[2]>0.:
            return 0.
        else:
            return -np.inf

    def lnpos(self, p, arg_p, arg_l):

        lnprior = self.lnprior(p, arg_p)

        if lnprior == -np.inf:
            return -np.inf
        else:
            return self.lnlike(p, arg_l)+lnprior

    def load_theory(self, file_scal, file_tens, lmax, ns_range):

        self.lmax = lmax

        ns = np.arange(ns_range[0], ns_range[1], ns_range[2])
```

```
        data_scal = np.load(file_scal)

        l = np.arange(1, lmax+1)
        fl = np.append(1., l*(l+1)/(2.*np.pi))

        self.ns = ns
        self.data_scal = data_scal[:,:lmax+1]/fl

#         self.cl_scal_ns1 = data_scal[30,:lmax+1]/fl

#         self.interp = []
#         for l in range(lmax+1):
#             self.interp.append(interp1d(ns, data_scal[:,l]/fl[l]))

        del data_scal

        self.cl_tens = np.load(file_tens)[:lmax+1]/fl

    def cl_scal(self, ns):
        out = []
        for l in range(self.lmax+1):
#             out.append(self.interp[l](ns))
            out.append(np.interp(ns, self.ns, self.data_scal[:,l]))
        return np.array(out)

    def load_data(self, file_data, lmax):

        self.cl_data = np.load(file_data)

        l = np.arange(1, lmax+1)
        fl = np.append(1., l*(l+1)/(2.*np.pi))

        self.cl_data = self.cl_data[:lmax+1]/fl

    def calc_evidence(self, intervals, tol, max_n, arg_p, arg_l):

        min_n = 1000

        ndim = len(intervals)
        delta = np.array([x[1]-x[0] for x in intervals])
        p_min = np.array([x[0] for x in intervals])

        old_lnev = np.inf

        lnlk = []
        ite = trange(max_n)
        for i in ite:
```

```
            p = np.random.rand(ndim)
            p = delta*p+p_min

            lnlk.append(self.lnlike(p, arg_l))

            if i % 100 or i == 0:
                continue

            lnlk_max = max(lnlk)
            lnev = np.log(np.mean([np.exp(l-lnlk_max) for l in lnlk]))+lnlk_max
            err =np.abs((old_lnev-lnev)/lnev)

            ite.set_description('LnEv: %f | err: %7.1e ' % (lnev, err))

            converged = err < tol
            converged &= i >= min_n
            if converged:
                print ' Converged!'
                ite.close()
                return lnev
            old_lnev = lnev

        print 'Evidence not converged: err = %7.1e' % err
        return None
```