ASSIGNMENT 5 SOLUTIONS

Note: Uploaded slightly late due to technical issues with RAM overloading on MCMC high-step count computations.

Q1)

Using some preset parameters return a Chi2 of 15268 for 2501 degrees of freedom. As our fitted Chi2 is way too far from the DOF, this fit is unacceptable with a survival of zero…

$H_0 = 69,\ \Omega_b h^2 = 0.22, \Omega_c h^2 = 0.12,\ \tau = 0.06,\ A_s = 2.1e-9,\ n_s = 0.95$

Using the second set of parameters shown above, we extrapolate a value of 3272 for the Chi2 with a survival of 10e-24... This new fit remains unacceptable roughly 10+ sigmas away from it!

The mean is 2501 ± 71 which agrees with the predicted $n \pm \sqrt{2n}$ value where n is the DOF.

Q2)

For this question here, we just apply the LM many times. As such, we may minimize to find a reasonable Chi2 value… A damping factor gets lower and lower as we approach our new committed Chi2 target value. And as such, the parameters are recalculated along each step to find reach a new set of optimized values.

NEW OPTIMIZED FIT:

$H_0 = 68.0 \pm 1.18$

$\Omega_b h^2 = 2.24e-2 \pm 2.32e-4$

$\Omega_c h^2 = 1.19e-1 \pm 2.65e-3$

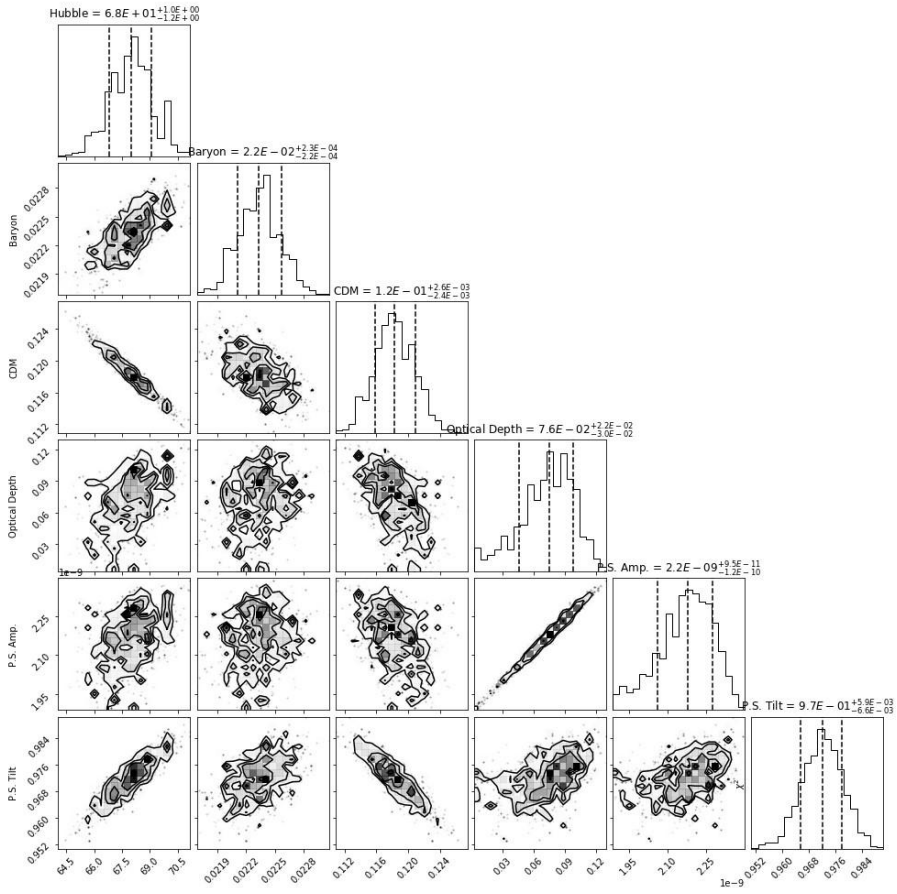
$\tau = 5.82e-2 \pm 3.66e-2$

$A_s = 2.11e-9 \pm 1.47e-10$

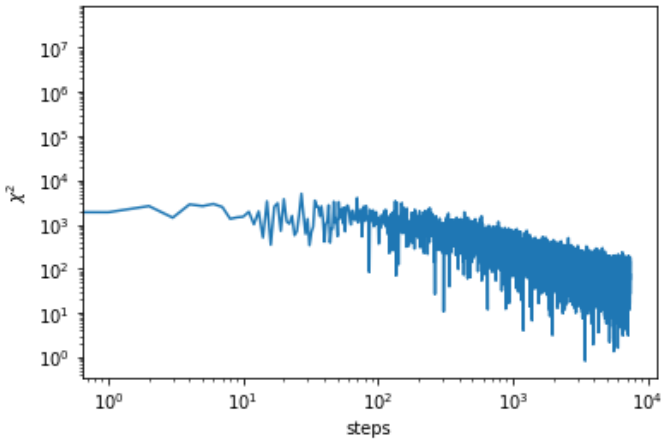
$n_s = 9.7e-1 \pm 6.40e-3$

We store the optimized values and cov-matrix for later use in MCMC in text files and later chain files.

Q3) We here implement numerical derivates. Applying the curvature matrix from before. As discussed with other classmates, learning how to code the Levenberg-Marquardt from their examples, I differentiated along what seems to be six directions in three dimensions. A damping process was introduced. Subsequently, we save the optimized results on SSD for later usage. We here run continuous chains using previous data. Fusing these results together over roughly 3000 MCMC steps, we obtain the following Corner plot:

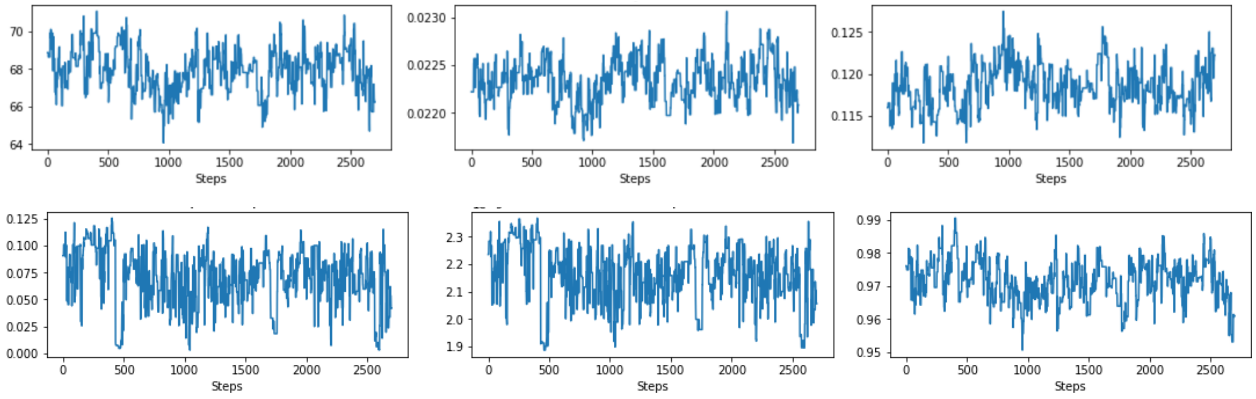


The chain seems to be doing quite well. No odd patterns have emerged necessarily. Looking at the FFT Power Spectrum, we note that chain convergence is evident. And the corner plot presents interesting correlations to indicate further convergence.

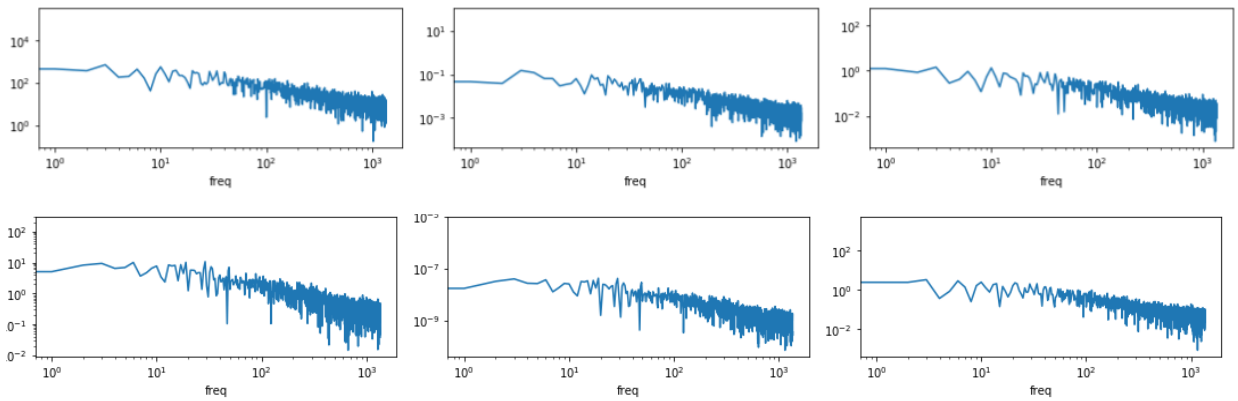


Again, this chain does seem to converge… Corner plots are shown below with minor skewing, which was not observed in the previous corner plots. This may also be due to the substantially higher step count (which could have revealed skewing that was not visible previously). Other classmates did observe similar results.

Here are the parameter plots for Hubble, Baryon, CDM, Optical Depth, Amp, and Tilt from left to right…



FFT Spectra for Hubble, Baryon, CDM, Optical Depth, Amp, and Tilt from left to right…

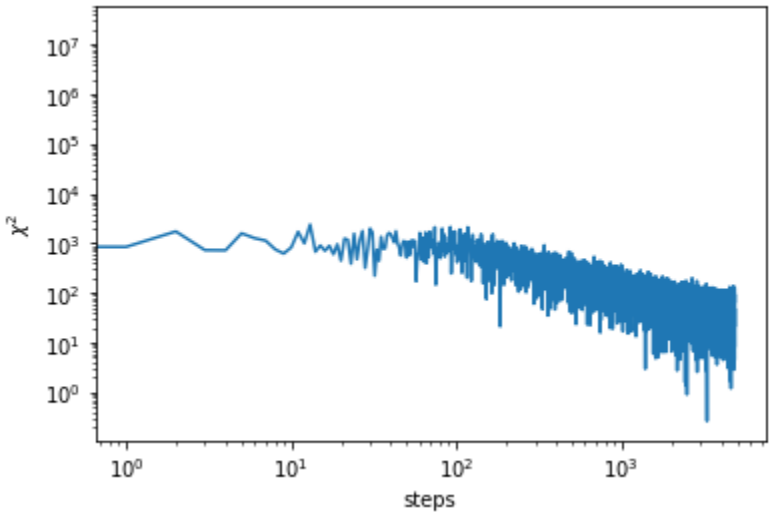


We additionally have a calculated value for Dark Energy of 0.70 ± 0.02

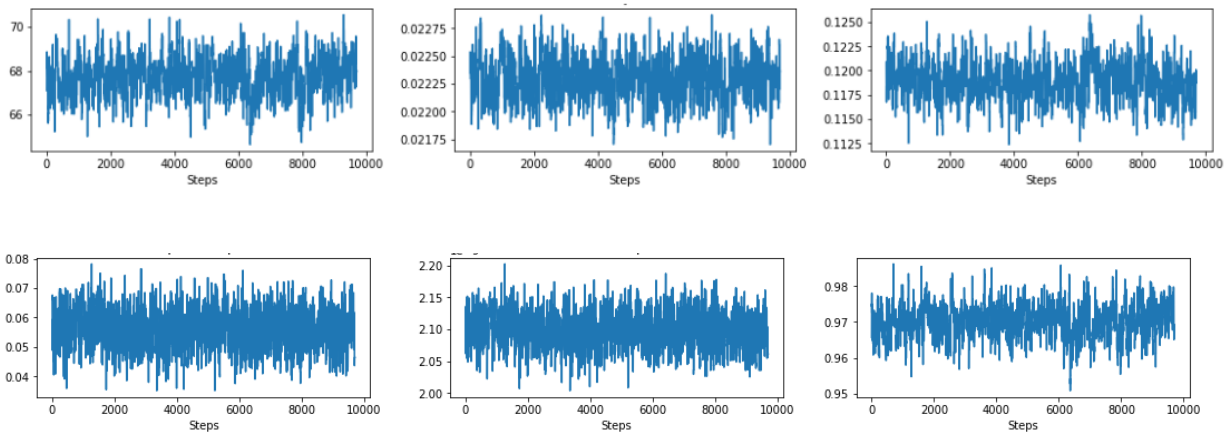
Q4)

Now, we do an MCMC with data solely taken from the covariance importance sampling rather than raw data. This time, running over 10,000 steps.

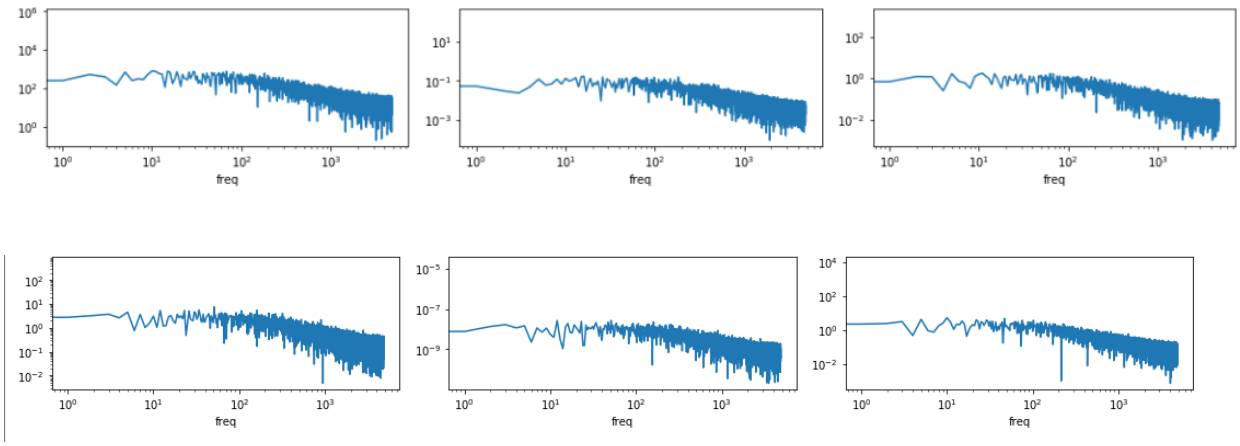
Below is the FFT Power Spectrum:



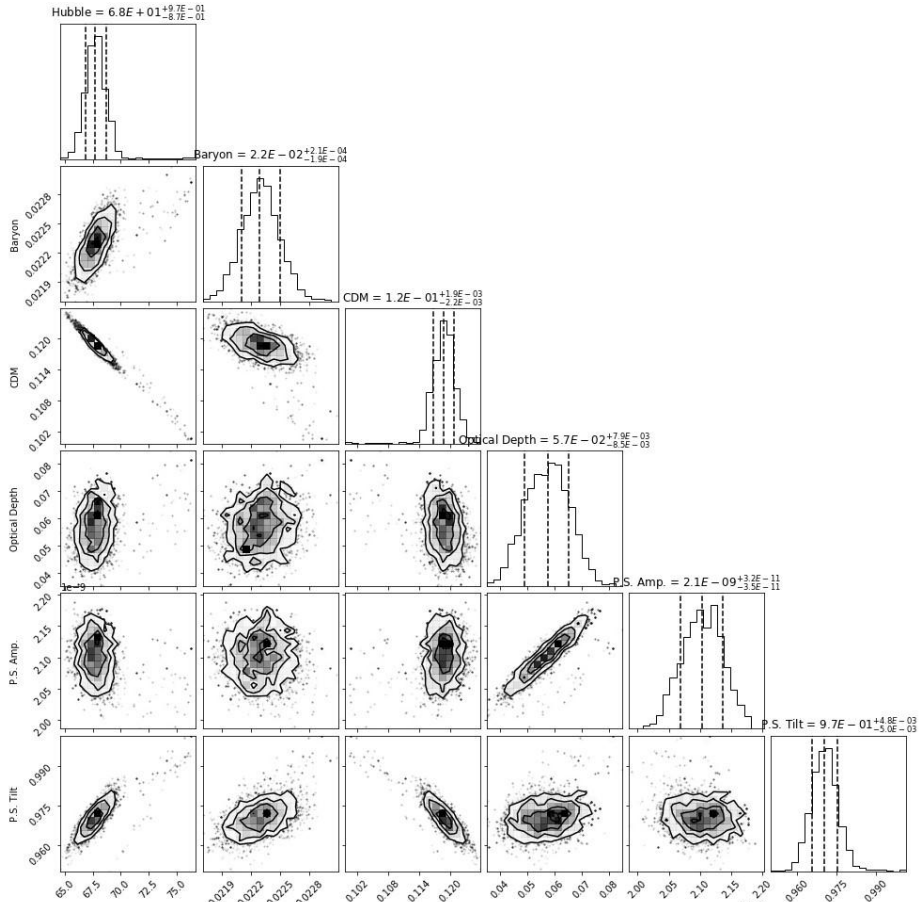
Here are the **parameter plots** for *Hubble, Baryon, CDM, Optical Depth, Amp, and Tilt* from left to right…



FFT Spectra for *Hubble, Baryon, CDM, Optical Depth, Amp, and Tilt* from left to right…



Below are the corner plots.



The **newly acquired MCMC parameters** (optimized from the importance sampling) are

$H_0 = 67.8 \pm 1.25$

$\Omega_b h^2 = 2.23e-2 \pm 1.99e-4$

$\Omega_c h^2 = 1.19e-1 \pm 2.69e-3$

$\tau = 5.73e-2 \pm 8.01e-2$

$A_s = 2.10e-9 \pm 3.25e-10$

$n_s = 9.71e-1 \pm 5.51e-3$

Whereas, the **importance sampling** was:

$H_0 = 68.0 \pm 9.55e-1$

$\Omega_b h^2 = 2.24e-2 \pm 2.43e-4$

$\Omega_c h^2 = 1.18e-1 \pm 2.14e-3$

$\tau = 5.76e-2 \pm 1.27e-2$

$A_s = 2.10e-9 \pm 5.31e-11$

$n_s = 9.71e-1 \pm 5.17e-3$

Clearly, these do not completely match in comparison. This is not as expected. Quite odd, some type of overshooting. They do agree, however, within a few sigma intervals with roughly similar errors. An exact replica should not be fully expected either.

Based on recommendations from other classmates, weighing the tau factor is critical here in getting accurate results. Not sure if this is fully visible in this computation and ran out of time to further investigate the root of the skewing and other factors.

Thank you for taking the time to grade my PSET. This was much more difficult than previous psets (to me at least). Worked through most of it along with classmates and eventually ran into hardware difficulties with RAM usage which halted progress for a while. I hope my explanations are sufficient.

DEFINED FUNCTIONS and loops of Interest…
(all code can be found in PSET5SolutionCode.py)
(accidentally introduced a bug in amend() at the end trying to clean code up, no time to fix)

```python
def calculateChiVal(y):
    return np.sum((y - spec)**2/errors**2)

def get_spectrum(pars,lmax=3000):

    H0 = pars[0]
    ombh2 = pars[1]
    omch2 = pars[2]
    tau = pars[3]
    As = pars[4]
    ns = pars[5]

    #CAMB Library (took a while to understand... with help from classmates)
    pars = camb.CAMBparams()
    pars.set_cosmology(H0=H0, ombh2=ombh2, omch2=omch2, mnu=0.06, omk=0, tau=tau)
    pars.InitPower.set_params(As=As,ns=ns,r=0)
    pars.set_for_lmax(lmax,lens_potential_accuracy=0)

    results=camb.get_results(pars)

    powers = results.get_cmb_power_spectra(pars,CMB_unit='muK')

    cmb = powers['total']
    tt = cmb[:,0]

    return tt[2:]

#Taken from Previous PSET
def Ndf(f, x, dxArr):
    diffs = []
    for i in range(len(x)):
        iter1 = np.zeros(len(x))
        iter1[i] += 1
        dx = dxArr[i]

        m1 = x.copy()
        m2  = x.copy()
        p1  = x.copy()
        p2 = x.copy()

        m2 -= 2 *dx * iter1
        m1  -= dx * iter1
        p1  += dx * iter1
        p2 += 2*dx * iter1
        diffs.append((f(m2) + 8 * f(p1) - 8 * f(m1) - f(p2))/(12 * dx))
    return np.array(diffs)


def calculateSpectrum(params, lmax = 3000):

    f = lambda params: get_spectrum(params, lmax)
    y = f(params)
    gradient = np.zeros([lmax, len(params)])
    gradient = Ndf(f, params, dx).transpose()
    return y, gradient

#Very Confused about this, helped by classmates
def amend(dampFac, succ):
    if succ:
        dampFac *= .3
    if dampFac <= 0.1:
        dampFac = 0
    else:
        if dampFac == 0:
            dampFac = 1
        else:
            dampFac *= 2
    return dampFac
```

```
#Got Help from Davia for this step
for i in range(stepCount):

    print("Damping Factor: ".rjust(30), dampFac)
    print("Parameter Values: ".rjust(30), p)

    pred, gradient = calculateSpectrum(p)
    pred = pred[:len(spec)]
    gradient = np.matrix(gradient)[:len(spec),:]

    r = spec - pred
    r = np.matrix(r).transpose()

    lhs = gradient.transpose() @ invN @ gradient
    curv_mat = np.linalg.inv(lhs)

    lhs += dampFac * np.diag(np.diag(gradient.transpose() @ invN @ gradient))
    rhs = gradient.transpose() @ invN @ r

    dp = np.linalg.inv(lhs)@(rhs)
    newPValu = p.copy()

    for j in range(len(p)):
        newPValu[j] = p[j] + dp[j]

    chiSqNew = calculateChiVal(get_spectrum(newPValu)[:len(spec)])

    if chiSqNew < chisq:

        succ = True
        chisq = chiSqNew

        p = newPValu
        dampFac = amend(dampFac, succ)

    else:
        succ = False
        dampFac = amend(dampFac, succ)

#print("Done For Loop 1")
```

```

p = np.load('obj/fit_p.npy')
curv_mat = np.load('obj/fit_curvmat.npy')

paramNames = ['Hubble', "Baryon", "CDM", "Optical Depth", "Primordial Amp.", "Primordial Tilt"]
paramError = np.sqrt(np.diag(curv_mat))
for i, n in enumerate(paramNames):
    print(f"{n} = ".ljust(20), f"{p[i]:.2E}", "+/-", f"{paramError[i]:.2E}")

fit_string = []
paramNames = ['Hubble', "Baryon", "CDM", "Optical Depth", "Primordial Amp.", "Primordial Tilt"]
paramError = np.sqrt(np.diag(curv_mat))

for i, n in enumerate(paramNames):
    fit_string.append(f"{n} = ".ljust(20)+f"{p[i]:.2E}"+ " +/- "+ f"{paramError[i]:.2E}\n")

#np.savetxt('planck_fit_params.txt', fit_string, delimiter="\n", fmt="%s")

ppChain = np.load('obj/chain.npy')[-1]
stepCount = 3000
scaleFactor = .75

```

```
201
202 chisqr = np.load('obj/chiChain.npy')[-1]
203
204 chain = np.zeros([stepCount, len(ppChain)])
205 chiChain = np.zeros(stepCount)
206
207 for s in np.arange(stepCount):
208
209     newPValu = ppChain + np.random.multivariate_normal(mean = np.zeros(len(curv_mat)), cov = curv_mat) * scaleFactor
210     chi2New = calculateChiVal(get_spectrum(newPValu)[:len(spec)])
211     del_chi = chi2New - chisqr
212     take = None
213     if del_chi >= 0 :
214         if np.random.rand() < np.exp(- 0.5 * del_chi):
215             11
216             take = True
217         else:
218             take = False
219     else:
220         take = True
221
222     if take == True:
223         ppChain = newPValu
224         chisqr = chi2New
225
226     chiChain[s] = chisqr
227     chain[s, :] = ppChain
228
229 #print("Done For Loop 2")
230
```

```
326 #Next Run
327 chain = np.zeros([stepCount, len(ppChain)])
328 chiChain = np.zeros(stepCount)
329
330 for s in np.arange(stepCount):
331
332     newPValu = ppChain + np.random.multivariate_normal(mean = np.zeros(len(curv_mat)), cov = prevCov) * scaleFactor
333
334     chi2New = calculateChiVal(get_spectrum(newPValu)[:len(spec)])
335     del_chi = chi2New - chisqr + (newPValu[3] - tau)**2/tauError**2
336
337     take = None
338     if del_chi >= 0 :
339         if np.random.rand() < np.exp(- 0.5 * del_chi):
340             take = True
341         else:
342             take = False
343     else:
344         take = True
345
346     if take == True:
347         ppChain = newPValu
348         chisqr = chi2New
349     chiChain[s] = chisqr
350     chain[s, :] = ppChain
```