Klion_Lab3

November 22, 2022

[1]:
```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

%matplotlib inline
```

[2]:
```
def TruncatedPoisson(mu, kmin, nsamples = 1):
    """Truncated Poisson, values >=k; mu is same as lambda
    This effectively uses x = F^{-1}(u) technique;
    exploits built-in python functions"""

    # normalization factor. Subtract pbty of truncated part
    nrm = 1.0 - stats.poisson.cdf(kmin-1, mu)

    # u = values between cdf(k) and 1; the second term is the random part
    yr = stats.poisson.cdf(kmin-1, mu) + np.random.rand(nsamples)*(nrm)

    # inverse CDF
    xr = stats.poisson.ppf(yr, mu)

    # maps them to integers
    return xr.astype(int)

def GelmanRubin(A, M, n):
    """A is a matrix with n columns and M rows
    Aij = ith sample from jth chain"""

    sj2 = np.zeros(n); aj = np.zeros(n)

    for j in range(n):
        sj2[j] = np.var(A[:,j])
        aj[j] = np.mean(A[:,j])
    W = np.mean(sj2) # within-chain
    B = M * np.var(aj)
    s = (1. - 1./M)*W + 1./M * B # inter-chain
    R = np.sqrt(s/W)

    return R, s, W, B
```

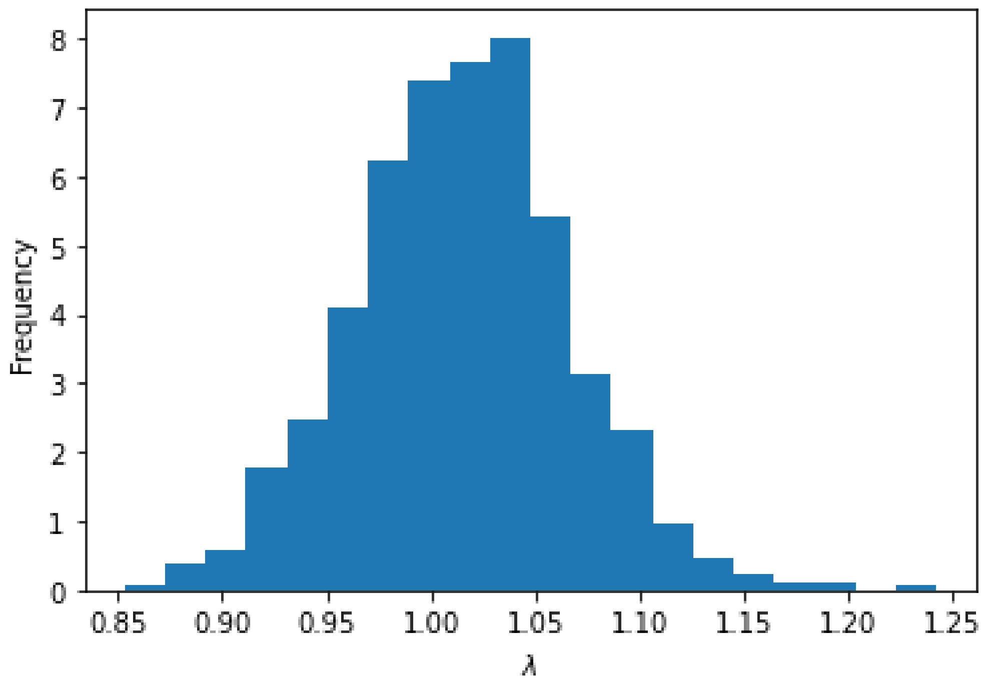
[3]:
```
'''x, z = np.empty(352), np.empty(13)
#j = 0
xj0 = np.empty(142)
for i in range(142):
    xj0[i] = 0

#j = 1
xj1 = np.empty(129)
for i in range(129):
    xj1[i] = 1

#j = 2
xj2 = np.empty(56)
for i in range(56):
    xj2[i] = 2

#j = 3
xj3 = np.empty(25)
for i in range(25):
    xj3[i] = 3

#j = 4
for i in range(13):
    z[i] = np.random.randint(4, 50)

#join all the x arrays together
x = np.concatenate((xj0, xj1, xj2, xj3))
#shuffle the contents of x using built-in function
np.random.shuffle(x)'''
```

[3]:
```
'x, z = np.empty(352), np.empty(13)\n#j = 0\nxj0 = np.empty(142)\nfor i in
range(142):\n    xj0[i] = 0\n\n#j = 1\nxj1 = np.empty(129)\nfor i in
range(129):\n    xj1[i] = 1\n\n#j = 2\nxj2 = np.empty(56)\nfor i in range(56):\n
xj2[i] = 2\n\n#j = 3\nxj3 = np.empty(25)\nfor i in range(25):\n    xj3[i] =
3\n\n#j = 4\nfor i in range(13):\n    z[i] = np.random.randint(4, 50)\n
\n#join all the x arrays together \nx = np.concatenate((xj0, xj1, xj2,
xj3))\n#shuffle the contents of x using built-in function\nnp.random.shuffle(x)'
```

[4]:
```
niters = 1000

#create 5 copies basically for diff starting lambdas
#Z_0 and lambda_0
lamb = np.empty(niters, float)
lamb1, lamb2, lamb3, lamb4 = np.empty(niters, float), np.empty(niters, float),␣
 ↪np.empty(niters, float), np.empty(niters, float)
lamb[0] = 1.
lamb1[0] = 5.
```

```
lamb2[0] = 0.1
lamb3[0] = 10.
lamb4[0] = 50.



Z = np.empty((13, niters), float)
Z1, Z2, Z3, Z4 = np.empty((13, niters), float), np.empty((13, niters), float),␣
 ↪np.empty((13, niters), float), np.empty((13, niters), float)
Z[:, 0] = TruncatedPoisson(lamb[0], 4, nsamples = 13)
Z1[:, 0] = TruncatedPoisson(lamb1[0], 4, nsamples = 13)
Z2[:, 0] = TruncatedPoisson(lamb2[0], 4, nsamples = 13)
Z3[:, 0] = TruncatedPoisson(lamb3[0], 4, nsamples = 13)
Z4[:, 0] = TruncatedPoisson(lamb4[0], 4, nsamples = 13)

for i in range(1, niters):
    Z[:, i] = TruncatedPoisson(lamb[i-1], 4, nsamples = 13)
    Z1[:, i] = TruncatedPoisson(lamb1[i-1], 4, nsamples = 13)
    Z2[:, i] = TruncatedPoisson(lamb2[i-1], 4, nsamples = 13)
    Z3[:, i] = TruncatedPoisson(lamb3[i-1], 4, nsamples = 13)
    Z4[:, i] = TruncatedPoisson(lamb4[i-1], 4, nsamples = 13)

    lamb[i] = stats.gamma.rvs((316 + np.sum(Z[:, i])), scale = 1/365)
    lamb1[i] = stats.gamma.rvs((316 + np.sum(Z1[:, i])), scale = 1/365)
    lamb2[i] = stats.gamma.rvs((316 + np.sum(Z2[:, i])), scale = 1/365)
    lamb3[i] = stats.gamma.rvs((316 + np.sum(Z3[:, i])), scale = 1/365)
    lamb4[i] = stats.gamma.rvs((316 + np.sum(Z4[:, i])), scale = 1/365)
```

[5]:
```
plt.hist(lamb[:-20], 20, density = True)
plt.xlabel("$\lambda$")
plt.ylabel("Frequency")
plt.show()
```

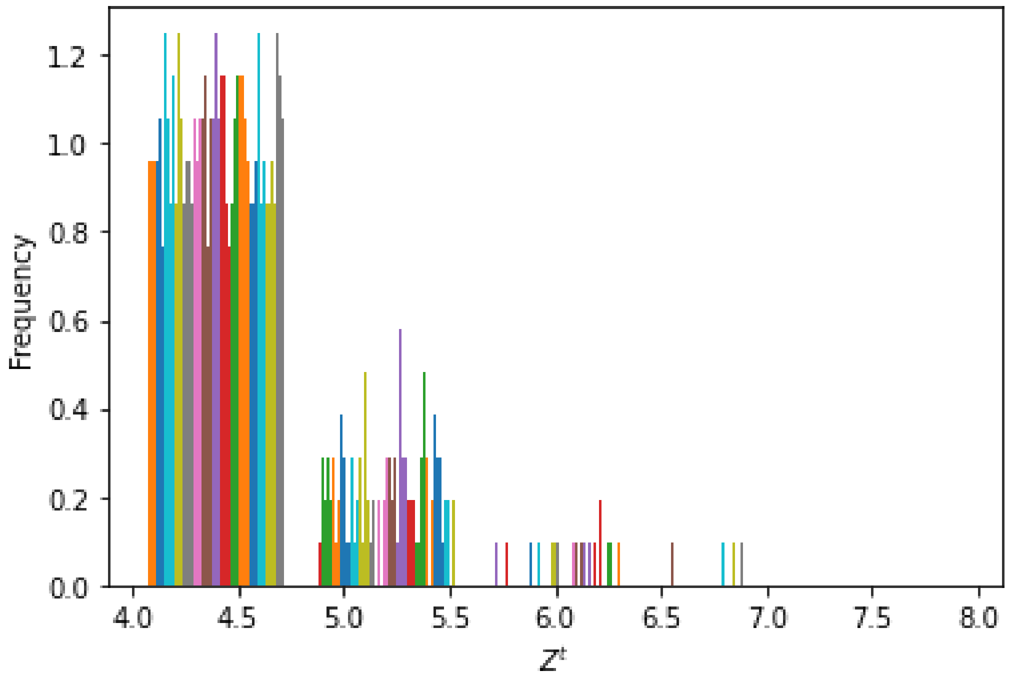


[6]:
```
allLambs = np.vstack((lamb, lamb1, lamb2, lamb3, lamb4))
GelmanRubin(allLambs, 5, niters//2)
```

[6]:
```
(1.106110772970685, 0.8613324920500058, 0.7040015026178184, 1.4906564497787558)
```

R $\approx$ 1.1061, which is less than 1.2, so the amount of iterations is fine for convergence.

[7]:
```
#for i in range(niters):
plt.hist(Z, 5, density = True)
plt.xlabel("$Z^{t}$")
plt.ylabel("Frequency")
plt.show()
```

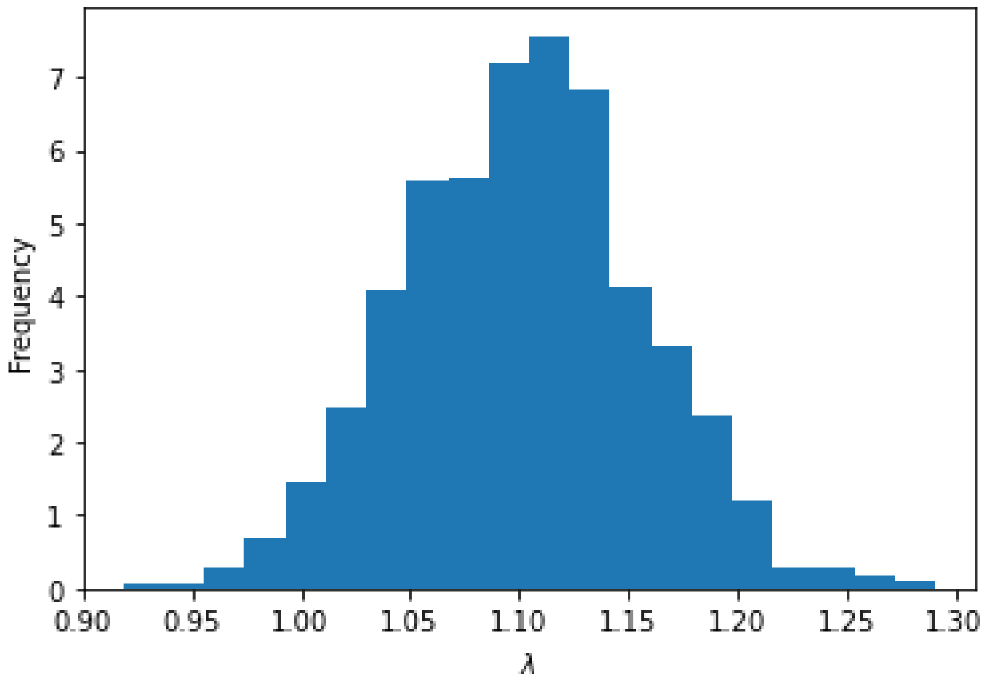


1 3+ = 38 now

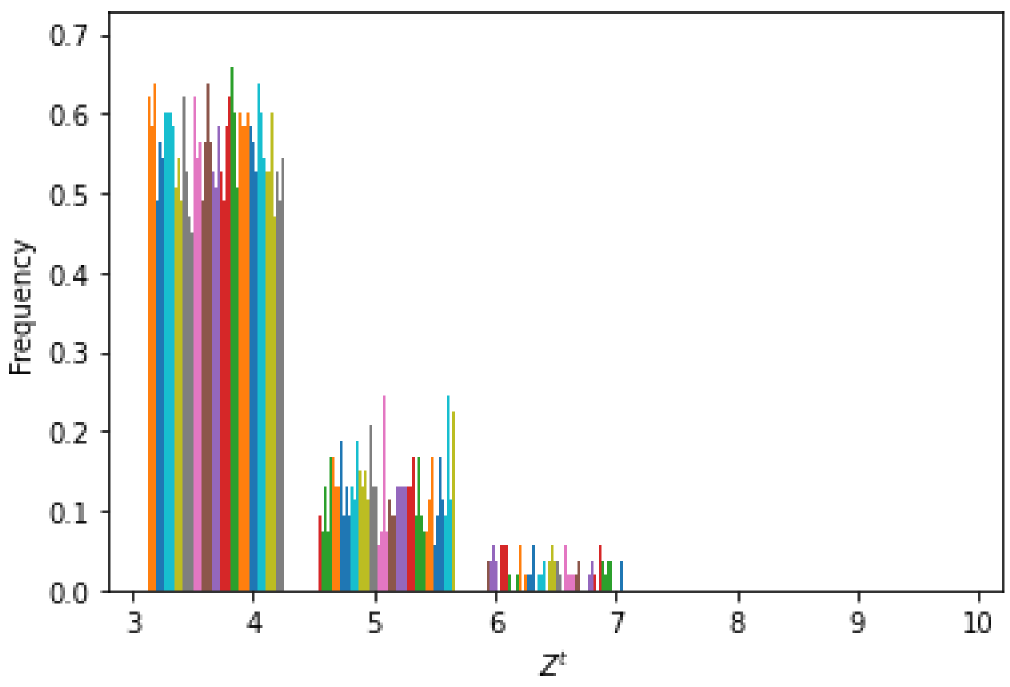
[8]:
```
#create 5 copies basically for diff starting lambdas
#Z_0 and lambda_0
lamb = np.empty(niters, float)
lamb1, lamb2, lamb3, lamb4 = np.empty(niters, float), np.empty(niters, float),␣
 ↪np.empty(niters, float), np.empty(niters, float)
lamb[0] = 1.
lamb1[0] = 5.
lamb2[0] = 0.1
lamb3[0] = 10.
lamb4[0] = 50.


Z = np.empty((38, niters), float)
Z1, Z2, Z3, Z4 = np.empty((38, niters), float), np.empty((38, niters), float),␣
 ↪np.empty((38, niters), float), np.empty((38, niters), float)
Z[:, 0] = TruncatedPoisson(lamb[0], 3, nsamples = 38)
Z1[:, 0] = TruncatedPoisson(lamb1[0], 3, nsamples = 38)
Z2[:, 0] = TruncatedPoisson(lamb2[0], 3, nsamples = 38)
Z3[:, 0] = TruncatedPoisson(lamb3[0], 3, nsamples = 38)
Z4[:, 0] = TruncatedPoisson(lamb4[0], 3, nsamples = 38)
```

```
for i in range(1, niters):
    Z[:, i] = TruncatedPoisson(lamb[i-1], 4, nsamples = 38)
    Z1[:, i] = TruncatedPoisson(lamb1[i-1], 4, nsamples = 38)
    Z2[:, i] = TruncatedPoisson(lamb2[i-1], 4, nsamples = 38)
    Z3[:, i] = TruncatedPoisson(lamb3[i-1], 4, nsamples = 38)
    Z4[:, i] = TruncatedPoisson(lamb4[i-1], 4, nsamples = 38)

    lamb[i] = stats.gamma.rvs((241 + np.sum(Z[:, i])), scale = 1/365)
    lamb1[i] = stats.gamma.rvs((241 + np.sum(Z1[:, i])), scale = 1/365)
    lamb2[i] = stats.gamma.rvs((241 + np.sum(Z2[:, i])), scale = 1/365)
    lamb3[i] = stats.gamma.rvs((241 + np.sum(Z3[:, i])), scale = 1/365)
    lamb4[i] = stats.gamma.rvs((241 + np.sum(Z4[:, i])), scale = 1/365)
```

[9]:
```
plt.hist(lamb, 20, density = True)
plt.xlabel('$\lambda$')
plt.ylabel('Frequency')
plt.show()
```



[10]:
```
plt.hist(Z, 5, density = True)
plt.xlabel("$Z^{t}$")
plt.ylabel("Frequency")
plt.show()
```



```
[11]: allLambs2 = np.vstack((lamb, lamb1, lamb2, lamb3, lamb4))
      GelmanRubin(allLambs2, 5, niters//2)
```

```
[11]: (1.1031414762433547, 0.8640077114673383, 0.709994838346942, 1.4800592039489238)
```

R $\approx$ 1.1031, which is less than 1.2, so the amount of iterations is fine for convergence.