# 之前的结果

具体的运行方式为在pytorch-coopnets里运行run\_whitenoise.sh和run\_fbm.sh得到的图片存在result\_image文件夹下，选其中较好的生成图片放在spectrum文件夹下,运行test\_welch.m即可得到生成图片和原始数据的功率谱密度对比图如下:



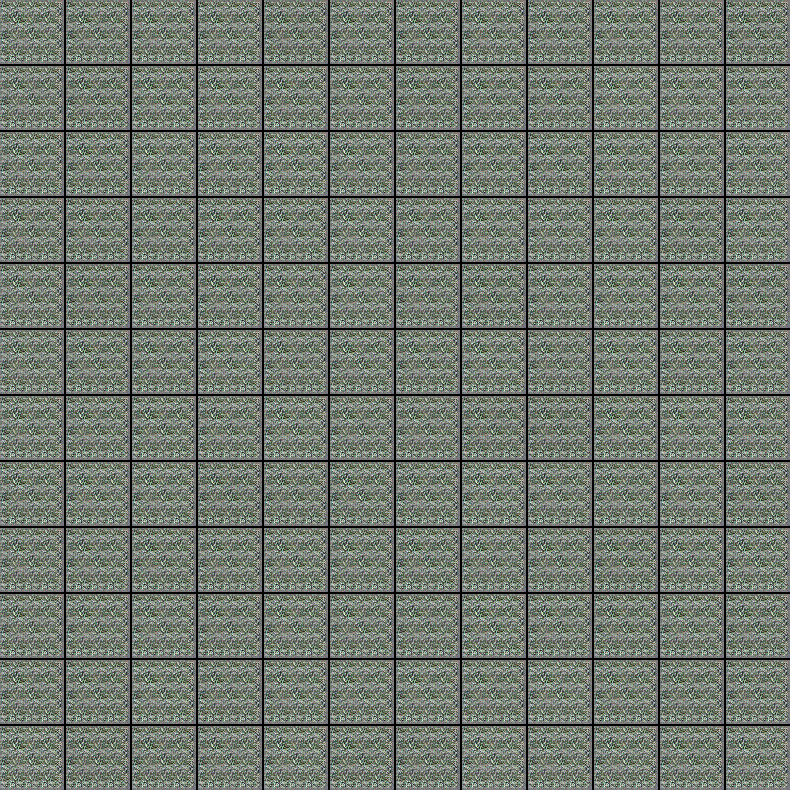


而这个湍流图片使用的暂时是之前的tensorflow源码的结果，还没做到湍流部分所以没在pytorch源码中运行。



附相应的高斯白噪声和fbm生成图片如下：

高斯白噪声：



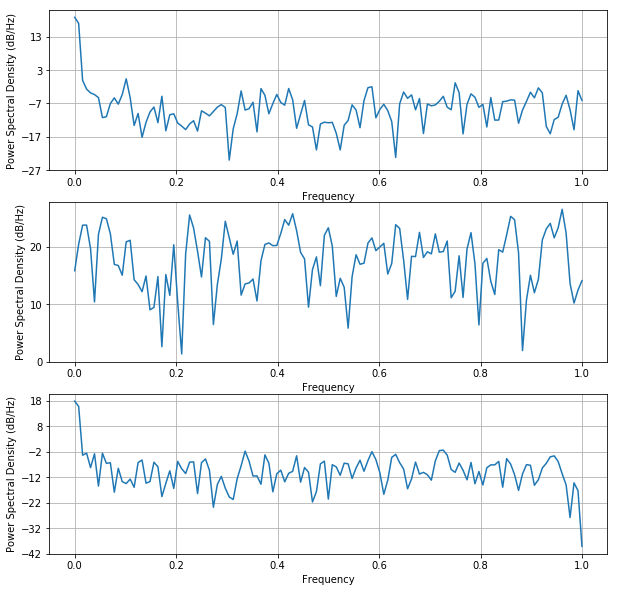
Fbm：



# 补充之前组会中说的自己做的3个部分：

1. 以下是用MCMC采样fbn得到的psd初步结果，第一个是使用协方差矩阵直接求导的结果，第二个是高斯白噪声，第三个是使用autograd对logPy求导的结果得到的功率谱密度：

具体的代码是pytorch-CoopNets库下的pytorch-CoopNets/gauss\_mcmc.ipynb中的第一个和第二个执行块，其中gauss\_omega\_mcmc\_simulate是使用协方差矩阵得到fbn采样的函数，gauss\_omega\_mcmc\_simulate\_omega是使用autograd对logPy求导进行mcmc的结果。



2.补充pytorch-coopnet仿真的Cosmodata功率谱密度对比和生成的图片：

具体方式是在pytorch-CoopNets中运行run\_cosmodata.sh得到生成的图片在result\_image文件夹下，取看起来比较好的生成结果移动到spectrum库中，运行spectrum/test\_welch.m即可得到下边的功率谱对比图片，其中对应的代码块是最后的%%号下边的可执行块。





3.之后做了在已知E|X|=miu1,EX\*X=miu2的时候对应的FRAME模型ME（最大熵）和MLE(极大似然估计)的仿真问题，我使用的方式有些问题，也许要调整先从gauss情况做起。我是假定存在有参数beta1，beta2的分布从中采样得到的结果作为输入的data,期望在一个其他的初始值的情况下,使用MLE估计可以梯度下降到预设的beta1，beta2值上。

具体代码在pytorch-CoopNets库下的pytorch-CoopNets/gauss\_mcmc.ipynb中的最后一个代码块，其中mcmc\_simulate\_beta是在已知beta情况下进行mcmc采样的函数，我用于生成输入数据和MLE中计算梯度时的采样，mle\_simulate\_beta是MLE估计beta值的函数，给定输入的初始值beta0使用梯度下降方式迭代得到估计的beta值。

具体结果暂时只有在调参数很细致时才会得到差不多结果，其余情况都看起来不会收敛。给定预设beta值是0.5，0.5.迭代初始值是0.4,0.4。得到类似下边的输出，逐步梯度下降到0.5，0.5而且很容易超过或者说不收敛而且算的很慢。

"0.4685170989334586 0.43731656999886065\n",

"0.4687488379329446 0.4374321701154116\n",

"0.46895221291482475 0.4375439932867888\n",

"0.46915393899381186 0.43765438503772053\n",

"0.46939071753621125 0.43777549667656457\n",

"0.46960024288296726 0.4378892287909988\n",

"0.46977657192945504 0.4379917091578248\n",

"0.4699295744299891 0.43808654901385335\n",

"0.47011504548788097 0.438185934737325\n",

"0.47027085021138215 0.4382702509164813\n",

"0.4704305318295958 0.43836614306271104\n",

"0.47065265506505993 0.43847960797697333\n",

"0.47085720449686075 0.43858979636430767\n",

"0.47106350255012536 0.43869685775041606\n",

"0.4712256305217745 0.4387945381850007\n",

"0.47147042176127457 0.4389196214377883\n",

"0.4716229093372824 0.439002151206136\n",

"0.4718507745563986 0.4391206479147079\n",

"0.47201612713933017 0.4392171871438625\n",

"0.4721723625361922 0.43930828213691736\n",

"0.4723418378829959 0.43940340148657586\n",

"0.47253851628303556 0.43951425866037636\n",

"0.4727363835275176 0.43961879086494476\n",

"0.4729376552999023 0.43972181856632264\n",

"0.473114439159632 0.4398182814717296\n",

"0.4733514914959672 0.43993848130852015\n",

"0.47354679124057325 0.4400436973348263\n",

"0.4737487095445398 0.44015968868881494\n",

"0.47388252492249044 0.4402460022345188\n",

"0.4740837454646829 0.4403564316332343\n",

"0.474302254661918 0.4404703995212915\n",

"0.47446664933860333 0.44056842614710356\n",

"0.4746583545356992 0.44067612794786715\n",

"0.474831360176206 0.4407681840434673\n",

"0.47507773500680955 0.4408917074352505\n",

"0.47527067607641255 0.44099617016315484\n",

"0.4754448710083965 0.4410954402983191\n",

"0.47563546827435527 0.4412056217491629\n",

"0.4758264447748664 0.4413125115260484\n",

"0.47603017869591746 0.4414234935864808\n",

"0.4762066374719146 0.4415265301838519\n",

"0.4764026408493522 0.4416379875242712\n",

"0.47657035771012335 0.44173398595303315\n",

"0.47678976973891285 0.44185025970637815\n",

"0.4769810747206214 0.4419579459279777\n",

"0.4771617325246337 0.4420632180869581\n",

"0.4773796595484021 0.44217746520042434\n",

"0.4775705524832013 0.44227708458900467\n",

"0.47777120377123383 0.44238953895866884\n",

"0.47793272928893593 0.4424824417904021\n",

"0.4780977549999955 0.44257572924345745\n",

"0.4782932398766282 0.4426826602444054\n",

"0.47848358343541647 0.44278333009779464\n",

"0.4786580934375527 0.4428835379630328\n",

"0.47886392967402963 0.44298864555358897\n",

"0.4790827901512387 0.443106603935361\n",

"0.47926086385548144 0.44320421852916486\n",

"0.4794436273723844 0.4433050988093019\n",

"0.4796148117333654 0.4434051852598787\n"