Simulation Assignment II

Ze Yang (zey@andrew.cmu.edu)

January 31, 2018

```
In [2]: from abc import ABCMeta, abstractmethod
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        plt.style.use('ggplot')
        plt.rc('text', usetex=True)
        plt.rc('font', family='serif', size=15)
        %matplotlib inline

        import statsmodels.api as sm
        import scipy.stats as stats
        from scipy.special import gamma
        from scipy.stats import weibull_min, norm, t
        from scipy.linalg import expm
        from progressbar import ProgressBar
```

---

1 Cholesky Decomposition

$$\boldsymbol{\Sigma} = \begin{pmatrix} 1 & \rho & \rho^2 \\ \rho & 1 & \rho \\ \rho^2 & \rho & 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{pmatrix} \begin{pmatrix} a & b & d \\ 0 & c & e \\ 0 & 0 & f \end{pmatrix} = \begin{pmatrix} a^2 & ab & ad \\ ab & b^2+c^2 & bd+ce \\ ad & bd+ce & d^2+e^2+f^2 \end{pmatrix} \tag{1}$$

The second leading principal minor is exactly the same as the one we discussed in class. So a, b, c should be the same: $a = 1, b = \rho, c = \sqrt{1-\rho^2}$. In the third row we have:

$$\begin{cases} ad = \rho^2 \\ bd + ce = \rho \\ d^2 + e^2 + f^2 = 1 \end{cases} \quad \Rightarrow \quad \begin{cases} d = \rho^2 \\ e = \rho\sqrt{1-\rho^2} \\ f = \sqrt{1-\rho^2} \end{cases} \tag{2}$$

Therefore, to simulate a sequence of trivariate normal random variables with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, we do the following:

1. Simulate three independent standard normal $\boldsymbol{Z} = (Z_1 \ \ Z_2 \ \ Z_3)^\top$.

2. Let $\boldsymbol{X} = \boldsymbol{A}\boldsymbol{Z} + \boldsymbol{\mu}$, where

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 \\ \rho & \sqrt{1-\rho^2} & 0 \\ \rho^2 & \rho\sqrt{1-\rho^2} & \sqrt{1-\rho^2} \end{pmatrix}$$

Then $\boldsymbol{X}$ follows the desired trivariate normal distribution.

---

2 Normal Generation Methods

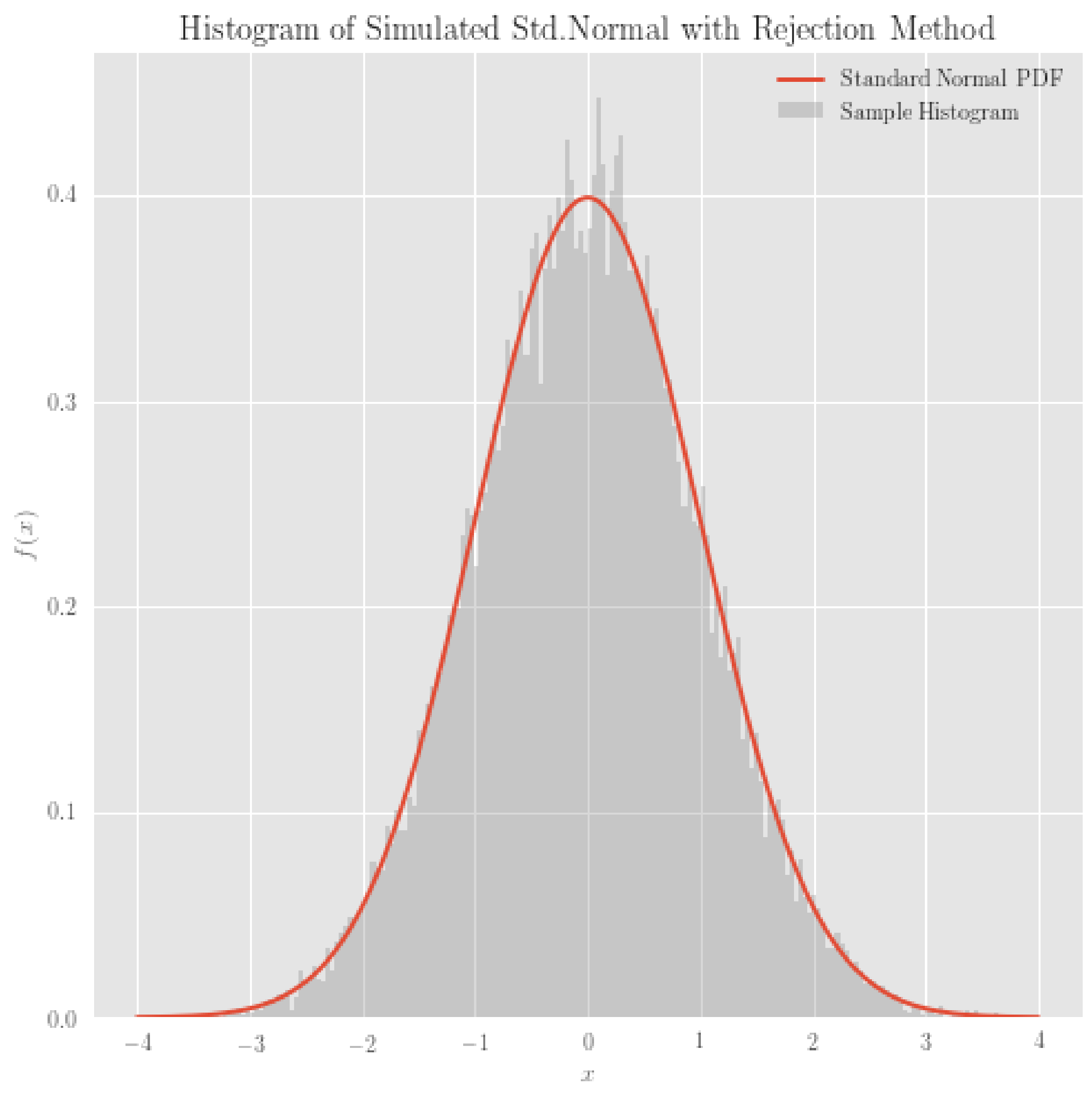
```
In [6]: # code from hw1
        class BaseRNG(object):
            __metaclass__ = ABCMeta

            @abstractmethod
            def draw(self, n=1):
                raise NotImplementedError

            def marginal_plot(self, ax, n_sample, bin_width,
                              truncate=(-np.inf, np.inf),
                              label='_nolegend_', which=None):
                if which is None:
                    # univariate rng
                    sample = self.draw(n_sample)
                else:
                    # multivariate rng
                    sample = self.draw(n_sample)[:,which]
                lb, ub = truncate
                bins = np.arange(max(min(sample), lb), (
                    min(max(sample), ub)+bin_width), bin_width)
                ax.hist(sample, bins, normed=1,
                        alpha=0.3, color="grey", label=label)
                return ax, sample


        class ProbIntegralGenerator(BaseRNG):

            def __init__(self, inv_cdf):
                self.inv_cdf = inv_cdf

            def draw(self, n=1):
                if n==1: return self.inv_cdf(np.random.uniform())
                u = np.random.uniform(size=n)
                return np.array([self.inv_cdf(p) for p in u])
        class RejectionGenerator(BaseRNG):

            def __init__(self, inducing_rng, rej_func):
                self.inducing_rng = inducing_rng
                self.rej_func = rej_func

            def draw(self, n):
                container, i = np.zeros(n), 0
                while i < n:
                    y = self.inducing_rng.draw()
                    u = np.random.uniform()
                    if u <= self.rej_func(y):
                        container[i] = y
                        i += 1
                if n == 1: return container[0]
                return container


        class GoldmanSachesMixture(BaseRNG):

            def draw(self, n):
                u = np.random.uniform(size=n)
                z = np.random.normal(size=n)
                z[u<=0.82] *= 0.6
                z[u>0.82] *= 1.98
                return z
```

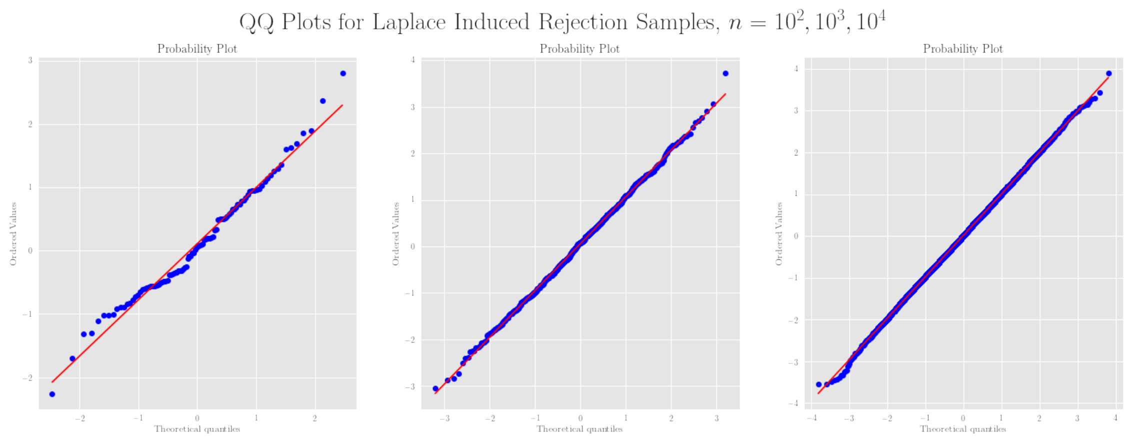
```
In [7]: # inverse cdfs
        def laplace_inv(b):
            return lambda u: np.random.choice([1,-1])*b*np.log(u)

        def generalized_lambda_inv(l1, l2, l3, l4):
            return lambda u: l1 + (1/l2)*(u**l3 - (1-u)**l4)

        # rejection threshold function (g(x)'s)
        def laplace_induced_normal_rej(c):
            return lambda x: (1/c)*np.sqrt(2*np.e/np.pi)*np.exp(
                -0.5*(x-np.sign(x))**2)
```

2.1 Rejection Method

```
In [8]: # rejection sampler
        laplace1_sampler = ProbIntegralGenerator(laplace_inv(1))
        std_normal_rejection_sampler = RejectionGenerator(
            inducing_rng=laplace1_sampler,
            rej_func=laplace_induced_normal_rej(c=np.sqrt(2*np.e/np.pi))
        )
```

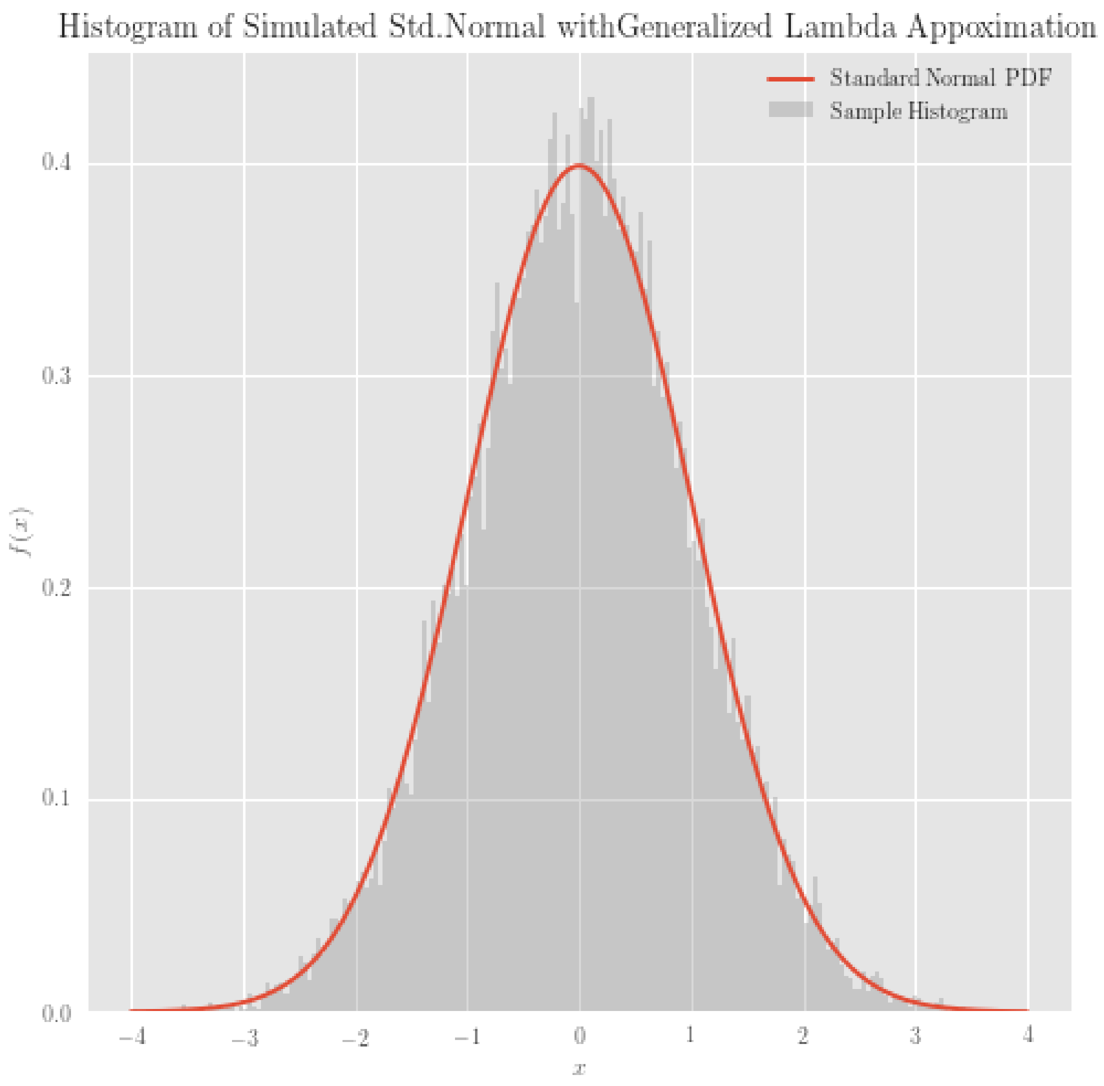
```
fig, ax = plt.subplots(1, 1, figsize=(8,8))
ax, sample = std_normal_rejection_sampler.marginal_plot(
    ax, 20000, 0.04, label='Sample Histogram')
x = np.linspace(-4, 4, 200)
ax.plot(x, norm.pdf(x), linewidth=2,
        label='Standard Normal PDF')
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$f(x)$')
ax.set_title('Histogram of Simulated Std.Normal with Rejection Method')
ax.legend()

f, axes = plt.subplots(1,3,figsize=(24, 8))
for j in range(3):
    normal_sample = std_normal_rejection_sampler.draw(10**(j+2))
    qq = stats.probplot(
        normal_sample, dist="norm", plot=axes[j])
_ = plt.suptitle(r'QQ Plots for Laplace Induced Rejection Samples,'
                 +r' $n=10^2, 10^3, 10^4$', size=30)
```


Histogram of Simulated Std.Normal with Rejection Method
Standard Normal PDF
Sample Histogram
$f(x)$
x
0.0
0.1
0.2
0.3
0.4
−4
−3
−2
−1
0
1
2
3
4

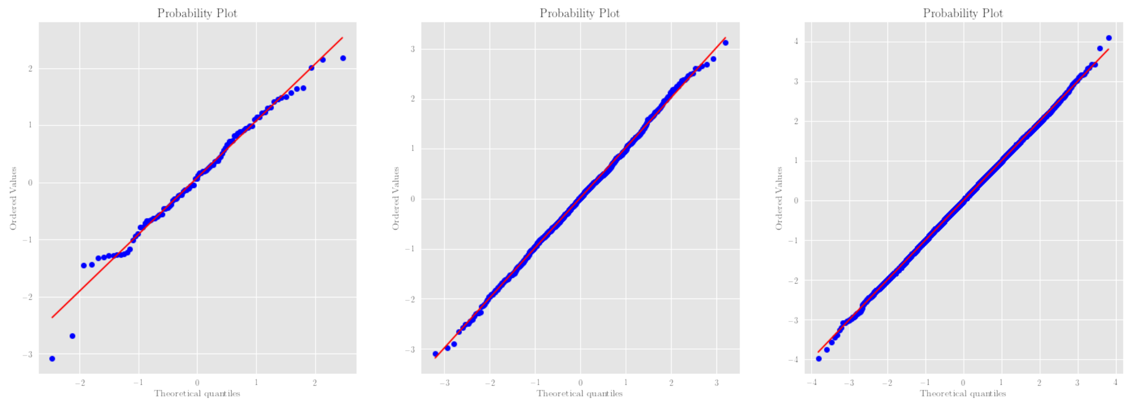

QQ Plots for Laplace Induced Rejection Samples, $n = 10^2, 10^3, 10^4$
Probability Plot
Probability Plot
Probability Plot
Ordered Values
Theoretical quantiles

2.2 Generalized Lambda Distribution

```
In [10]: # generalized lambda approximation sampler
         gen_lambda_approx_sampler = ProbIntegralGenerator(
             generalized_lambda_inv(0, 0.1975, 0.1349, 0.1349)
         )

         fig, ax = plt.subplots(1, 1, figsize=(8,8))
         ax, sample = gen_lambda_approx_sampler.marginal_plot(
             ax, 20000, 0.04, label='Sample Histogram')
         x = np.linspace(-4, 4, 200)
         ax.plot(x, norm.pdf(x), linewidth=2,
                 label='Standard Normal PDF')
         ax.set_xlabel(r'$x$')
         ax.set_ylabel(r'$f(x)$')
         ax.set_title('Histogram of Simulated Std.Normal with' +
                      'Generalized Lambda Appoximation')
         ax.legend()

         f, axes = plt.subplots(1,3,figsize=(24, 8))
         for j in range(3):
             normal_sample = gen_lambda_approx_sampler.draw(10**(j+2))
             qq = stats.probplot(
                 normal_sample, dist="norm", plot=axes[j])
         _ = plt.suptitle(r'QQ Plots for Generalized Lambda Samples,'
                      +r' $n=10^2, 10^3, 10^4$', size=30)
```

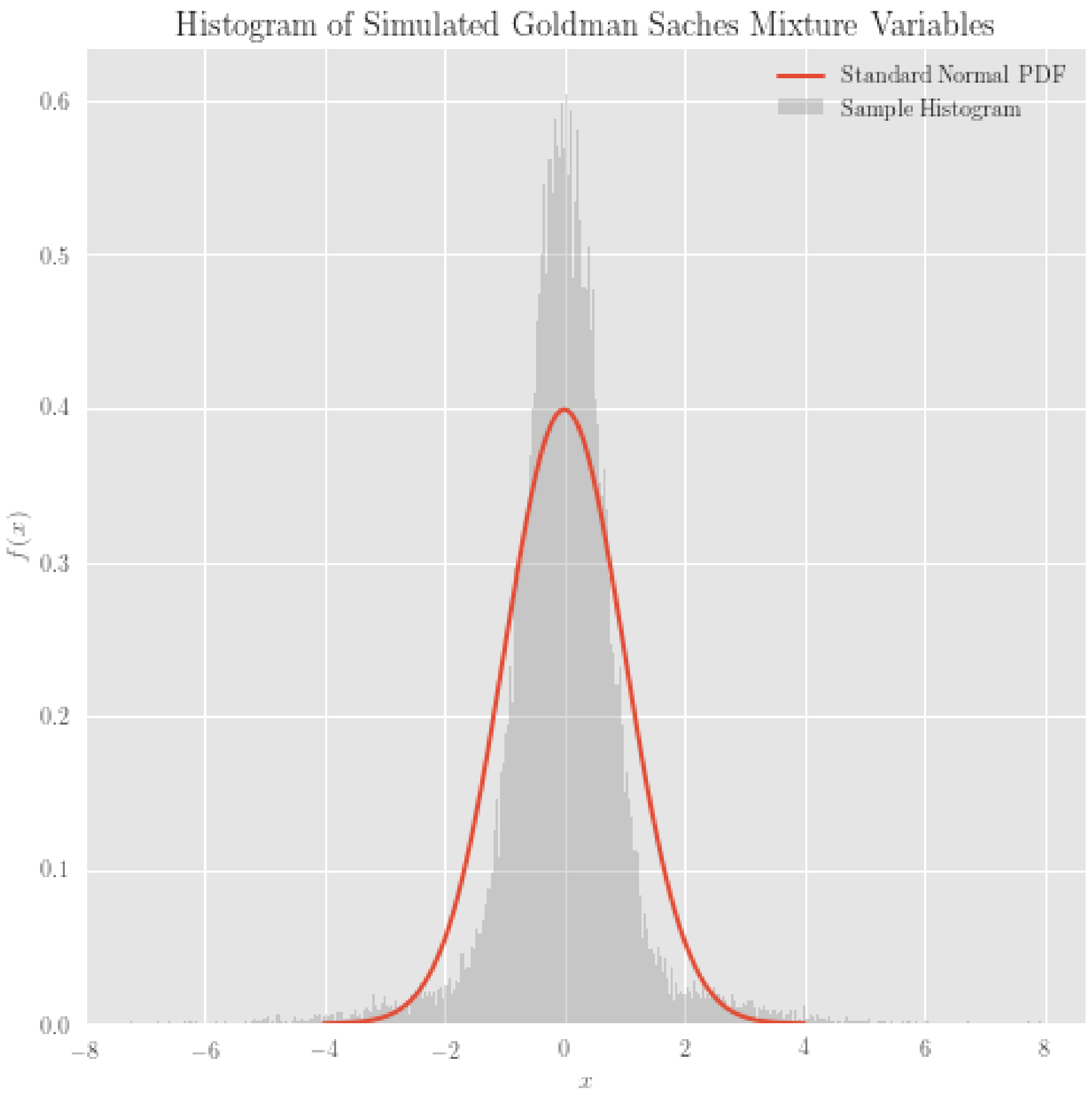


QQ Plots for Generalized Lambda Samples, $n = 10^2, 10^3, 10^4$

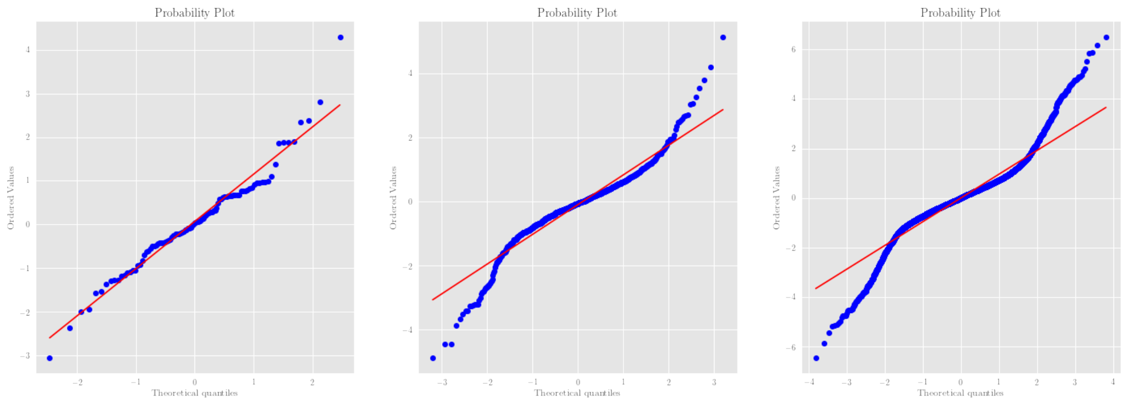


2.3 "Goldman Sachs Weighted Normal"

```
In [11]: # goldman heavy tail sampler
         gs_heavy_tail_sampler = GoldmanSachesMixture()
         fig, ax = plt.subplots(1, 1, figsize=(8,8))
         ax, sample = gs_heavy_tail_sampler.marginal_plot(
             ax, 20000, 0.04, label='Sample Histogram')
         x = np.linspace(-4, 4, 200)
         ax.plot(x, norm.pdf(x), linewidth=2,
                 label='Standard Normal PDF')
         ax.set_xlabel(r'$x$')
         ax.set_ylabel(r'$f(x)$')
         ax.set_title('Histogram of Simulated ' +
                      'Goldman Saches Mixture Variables')
         ax.legend()

         f, axes = plt.subplots(1,3,figsize=(24, 8))
         for j in range(3):
             normal_sample = gs_heavy_tail_sampler.draw(10**(j+2))
             qq = stats.probplot(
                 normal_sample, dist="norm", plot=axes[j])
         _ = plt.suptitle(r'QQ Plots for GS Heavy Tail Samples,'
                          +r' $n=10^2, 10^3, 10^4$', size=30)
```



QQ Plots for GS Heavy Tail Samples, $n = 10^2, 10^3, 10^4$



Comments:

- The Rejection and the Generalized Lambda integral transform methods generate very decent normal samples, in that the probability plot looks like a nice line.
- The "Goldman Sachs Weighted Normal" method is intended to sample from a distribution that has heavier tails than normal. Therefore, we can see from both the histogram and the qq plots that the sample has indeed a thicker tail part.

---

3 Bivariate Copula

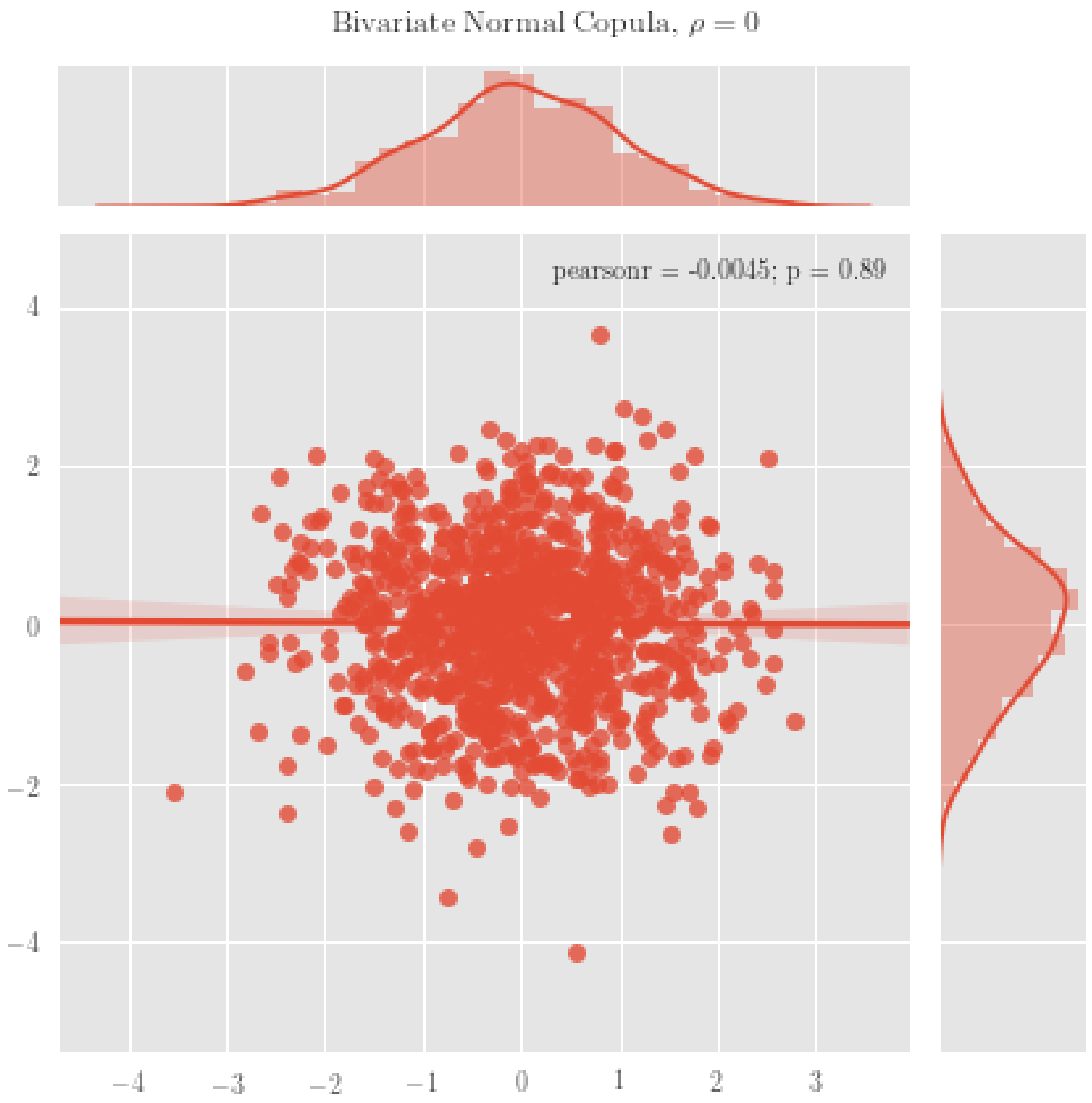
```
In [57]: class BaseCopula(BaseRNG):
             __metaclass__ = ABCMeta

             def reset_cov(self, cov_matrix):
                 assert len(cov_matrix) == self.d
                 self.cov = cov_matrix
                 self.std = np.sqrt(np.diag(self.cov))
                 self.A = np.linalg.cholesky(self.cov).T

             def bivariate_plot(self, n_sample, d1=0, d2=1):
                 X = self.draw(n_sample)
                 g = sns.jointplot(X[:,d1], X[:,d2], kind="reg")
                 return g


         class GaussianCopula(BaseCopula):

             def __init__(self, cov_matrix, marginal_inv_cdfs=None):
                 self.inv_cdfs = marginal_inv_cdfs
                 self.cov = cov_matrix
                 self.std = np.sqrt(np.diag(self.cov))
                 self.d = len(self.cov)
                 self.A = np.linalg.cholesky(self.cov).T

             def draw(self, n):
                 Z = np.random.normal(size=(n, self.d))
                 Y = Z.dot(self.A)
                 if not self.inv_cdfs: return Y
                 U = norm.cdf(Y/self.std)
                 return np.apply_along_axis(self.inv_cdfs, 1, U)


         class StudentTCopula(GaussianCopula):
```

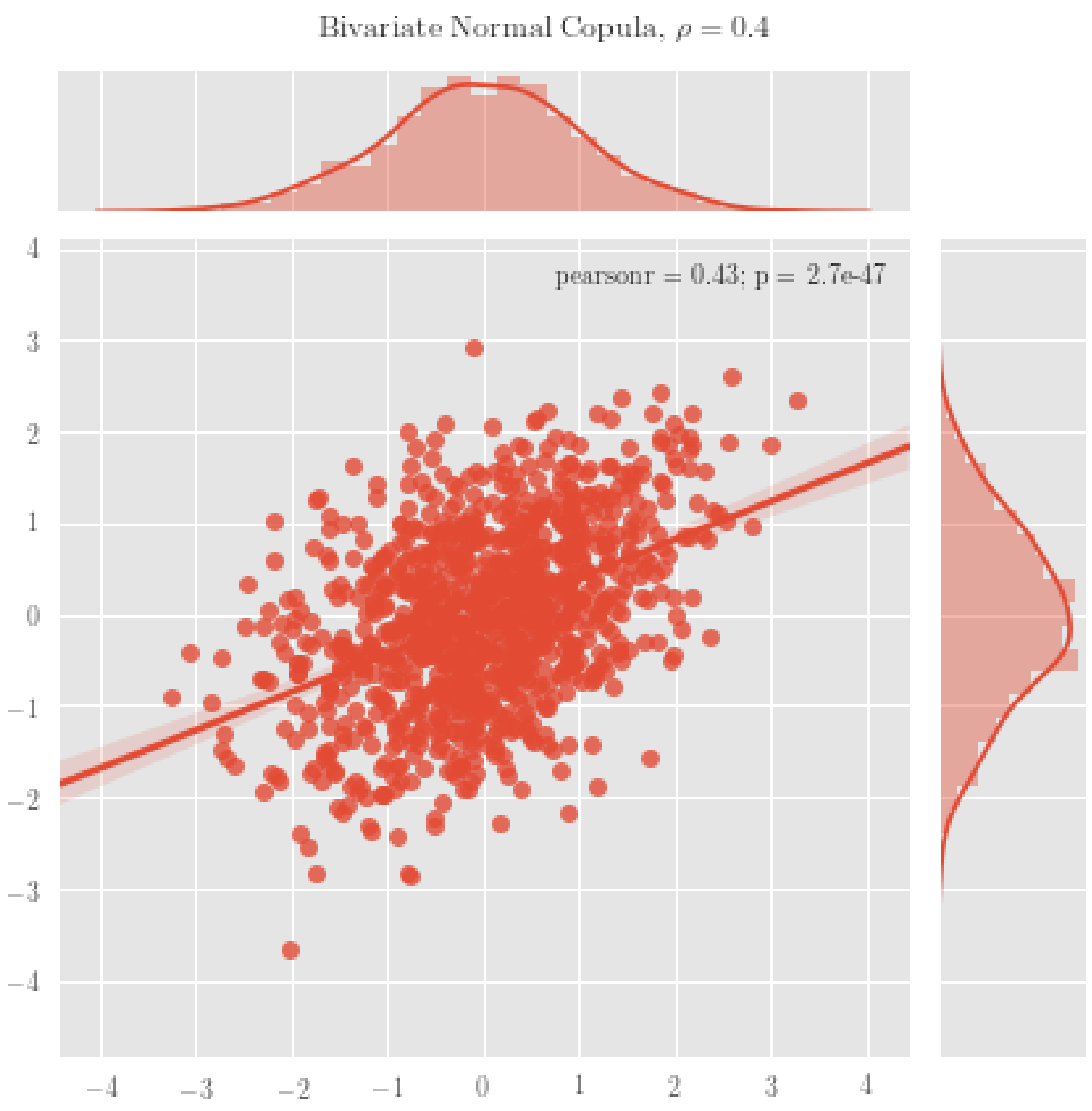
```
    def __init__(self, df, cov_matrix, marginal_inv_cdfs=None):
        super(StudentTCopula, self).__init__(
            cov_matrix, marginal_inv_cdfs)
        self.df = df

    def draw(self, n):
        Z = np.random.normal(size=(n, self.d))
        Y = Z.dot(self.A)
        S = np.random.chisquare(df=self.df, size=n)
        T = Y*np.sqrt(self.df/S).reshape(n,1)
        if not self.inv_cdfs: return T
        U = t.cdf(T/self.std, df=self.df)
        return np.apply_along_axis(self.inv_cdfs, 1, U)
```

```
In [17]: # inverse marginal cdfs
         def exponential_marginal_inv(b_vec):
             return lambda u: np.array(
                 [b*np.log(u[i]) for i,b in enumerate(b_vec)])
```

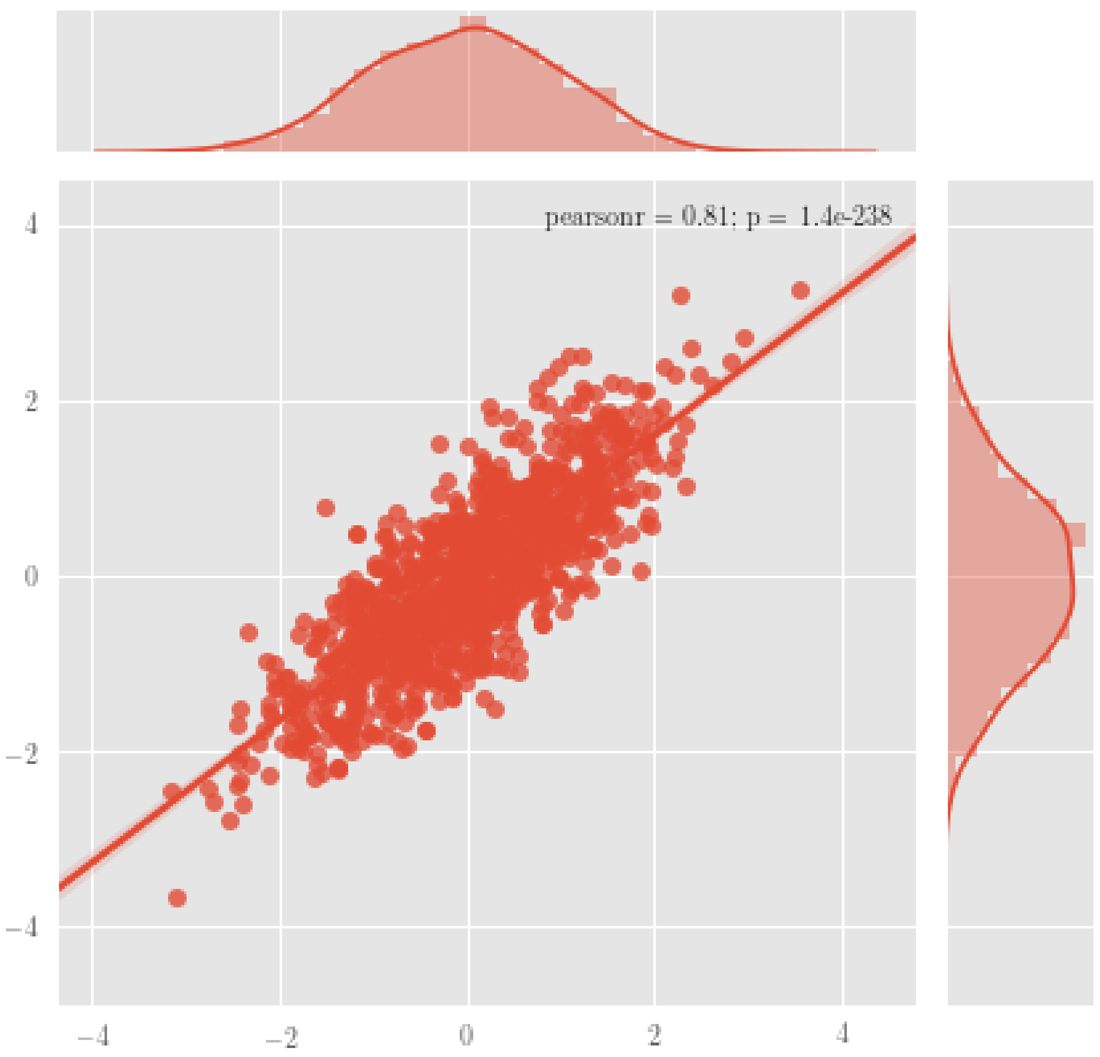
3.1 Bivariate Normal

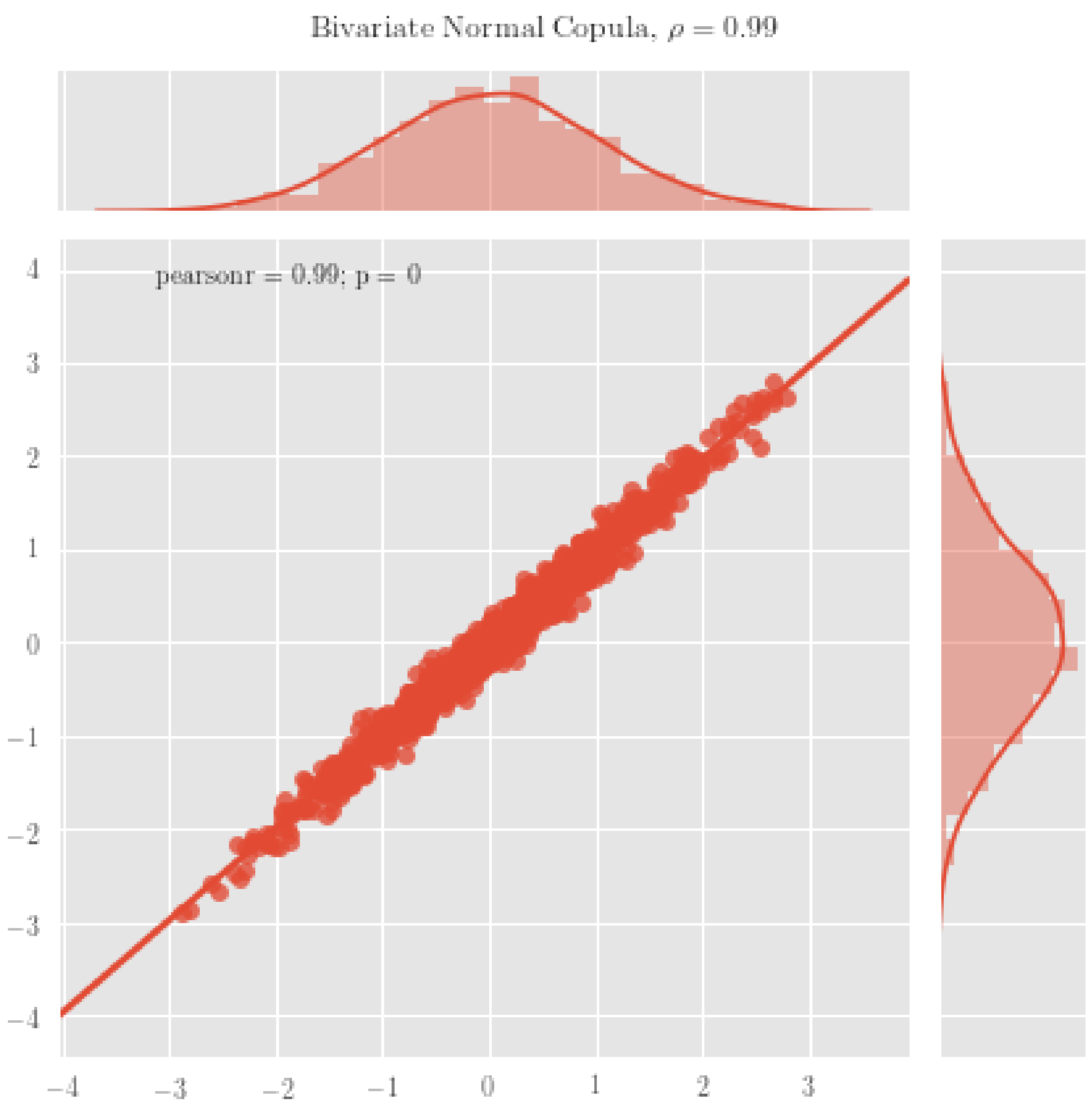
```
In [48]: gs_copula_sampler = GaussianCopula(
             np.array([[1,0], [0,1]])
         )
         for rho in [0, 0.4, 0.8, 0.99]:
             gs_copula_sampler.reset_cov(
                 np.array([[1,rho], [rho,1]])
             )
             g = gs_copula_sampler.bivariate_plot(1000)
             plt.subplots_adjust(top=0.93)
             g.fig.suptitle('Bivariate Normal Copula, '
                            +r'$\rho={}$'.format(rho))
```


Bivariate Normal Copula, $\rho = 0$
pearsonr = -0.0045; p = 0.89
4
2
0
−2
−4
−4
−3
−2
−1
0
1
2
3


Bivariate Normal Copula, $\rho = 0.4$
pearsonr = 0.43; p = 2.7e-47
−4
−3
−2
−1
0
1
2
3
4

Bivariate Normal Copula, $\rho = 0.8$

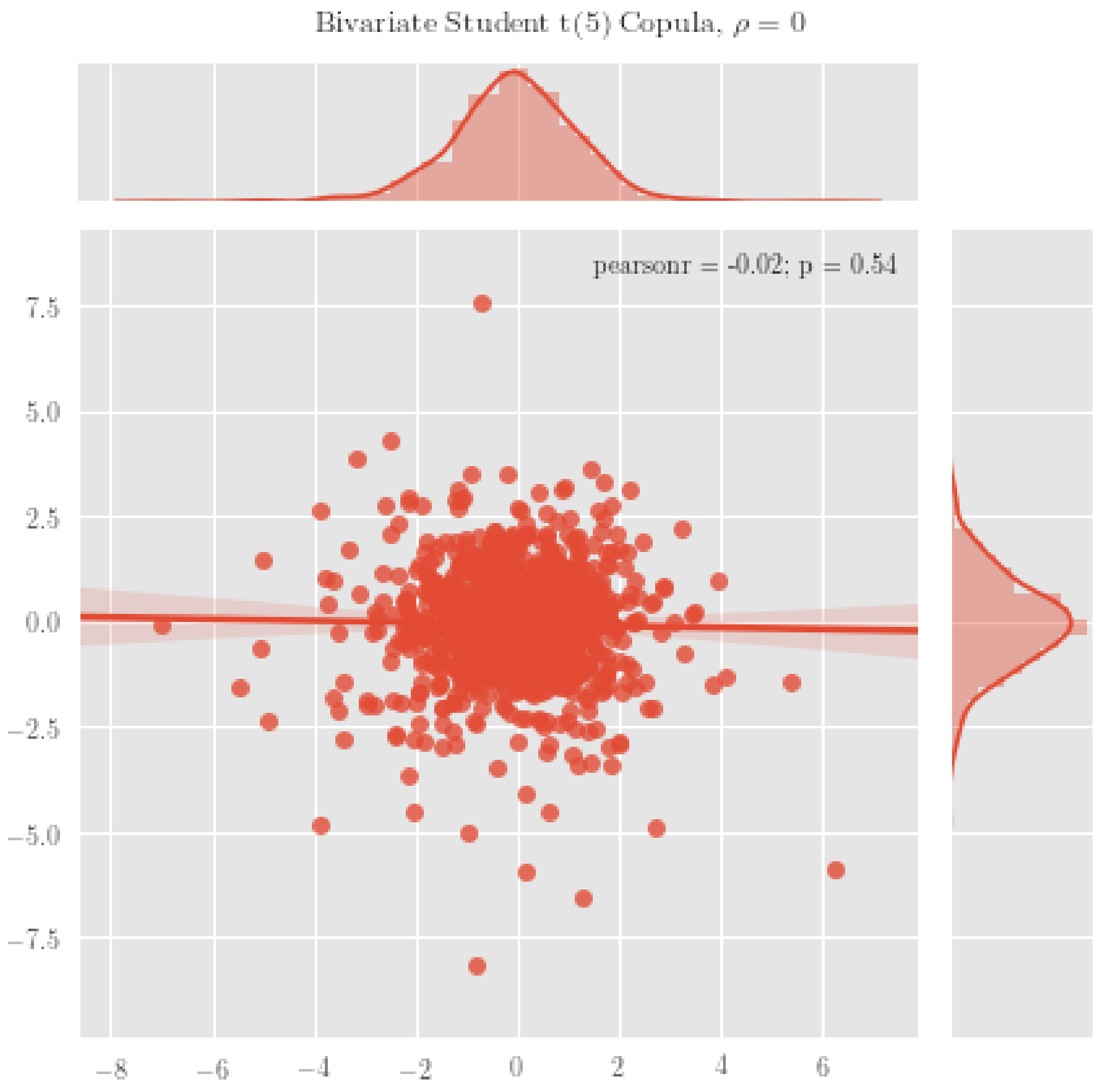


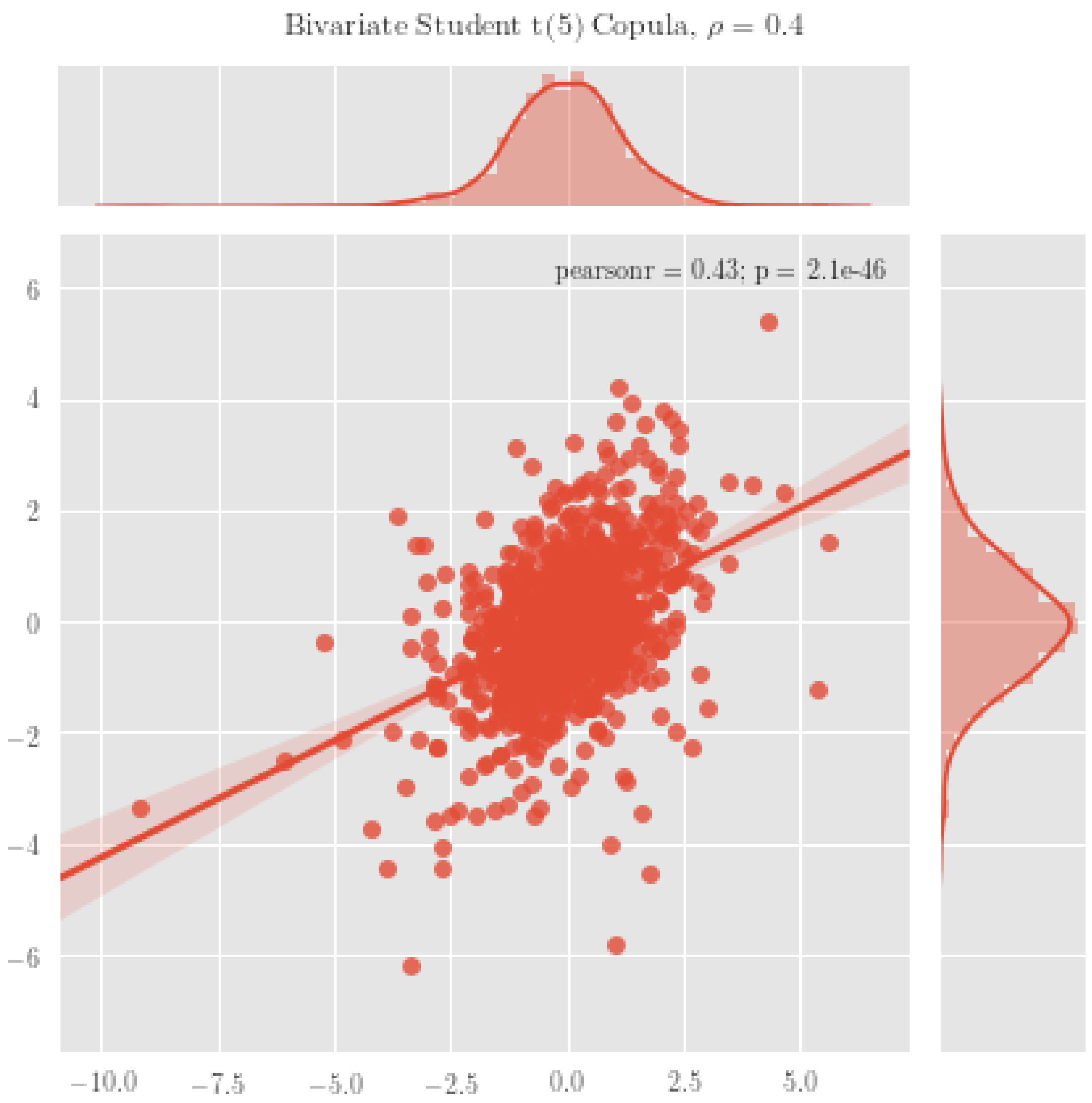


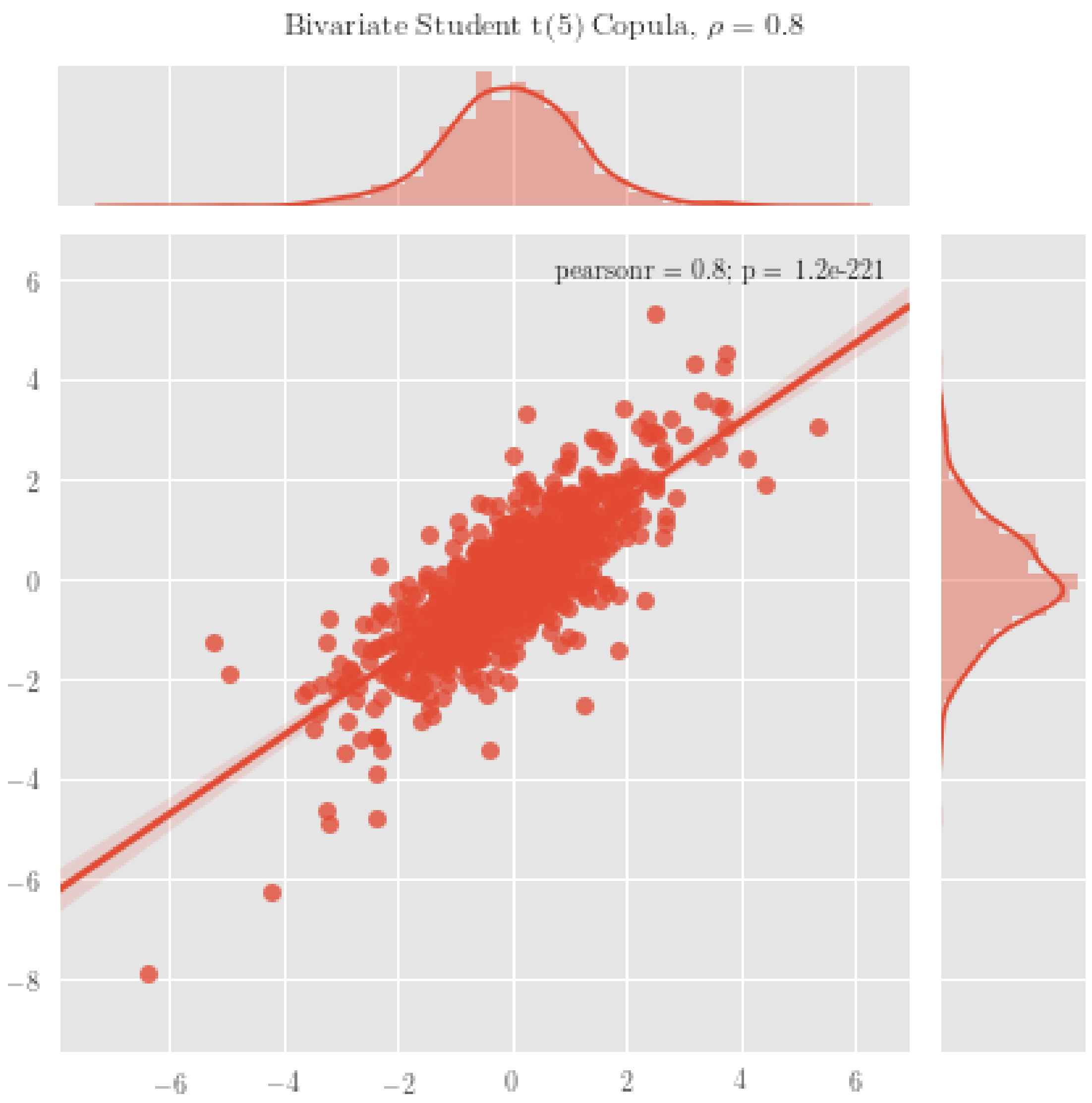
3.2 Student T(5) Copula

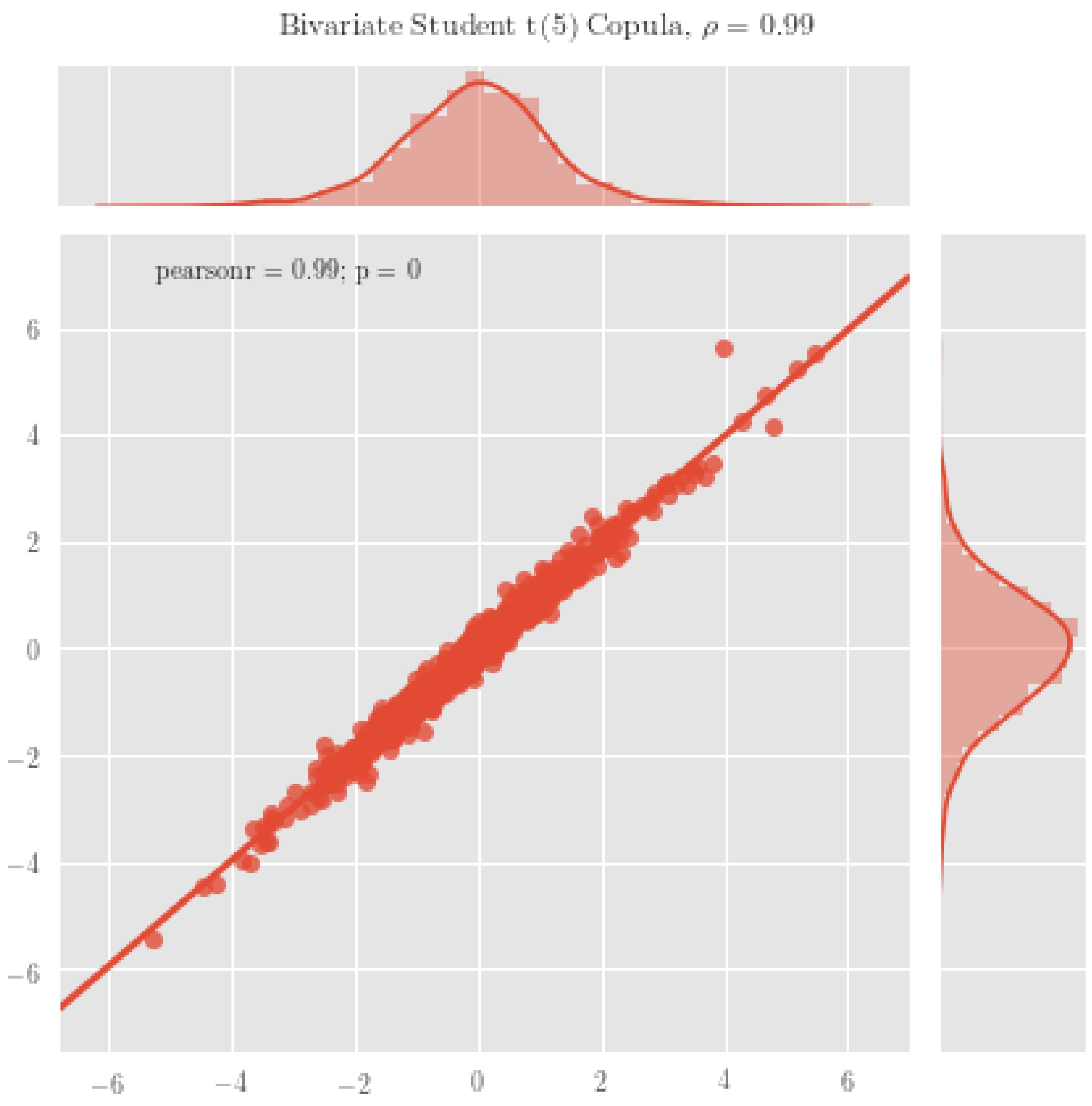
```
In [50]: t_copula_sampler = StudentTCopula(
             df=5,
             cov_matrix=np.array([[1,0], [0,1]])
         )
         for rho in [0, 0.4, 0.8, 0.99]:
             t_copula_sampler.reset_cov(
                 np.array([[1,rho], [rho,1]])
             )
             g = t_copula_sampler.bivariate_plot(1000)
             plt.subplots_adjust(top=0.93)
```

```
g.fig.suptitle('Bivariate Student t(5) Copula, '
                  +r'$\rho={}$'.format(rho))
```




Bivariate Student t(5) Copula, $\rho = 0.4$
pearsonr = 0.43; p = 2.1e-46
6
4
2
0
−2
−4
−6
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0

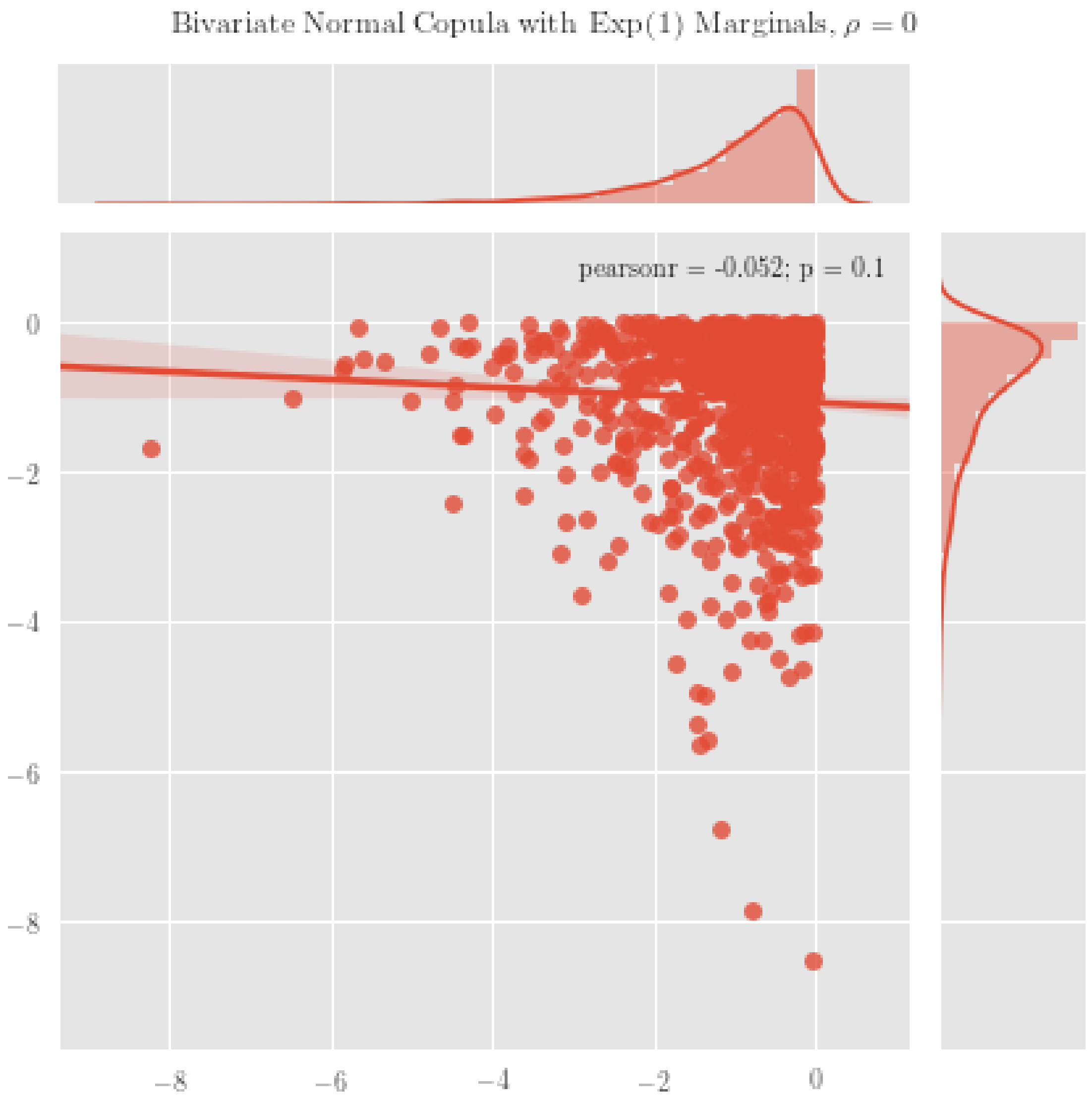

Bivariate Student t(5) Copula, $\rho = 0.8$
pearsonr = 0.8; p = 1.2e-221
6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6

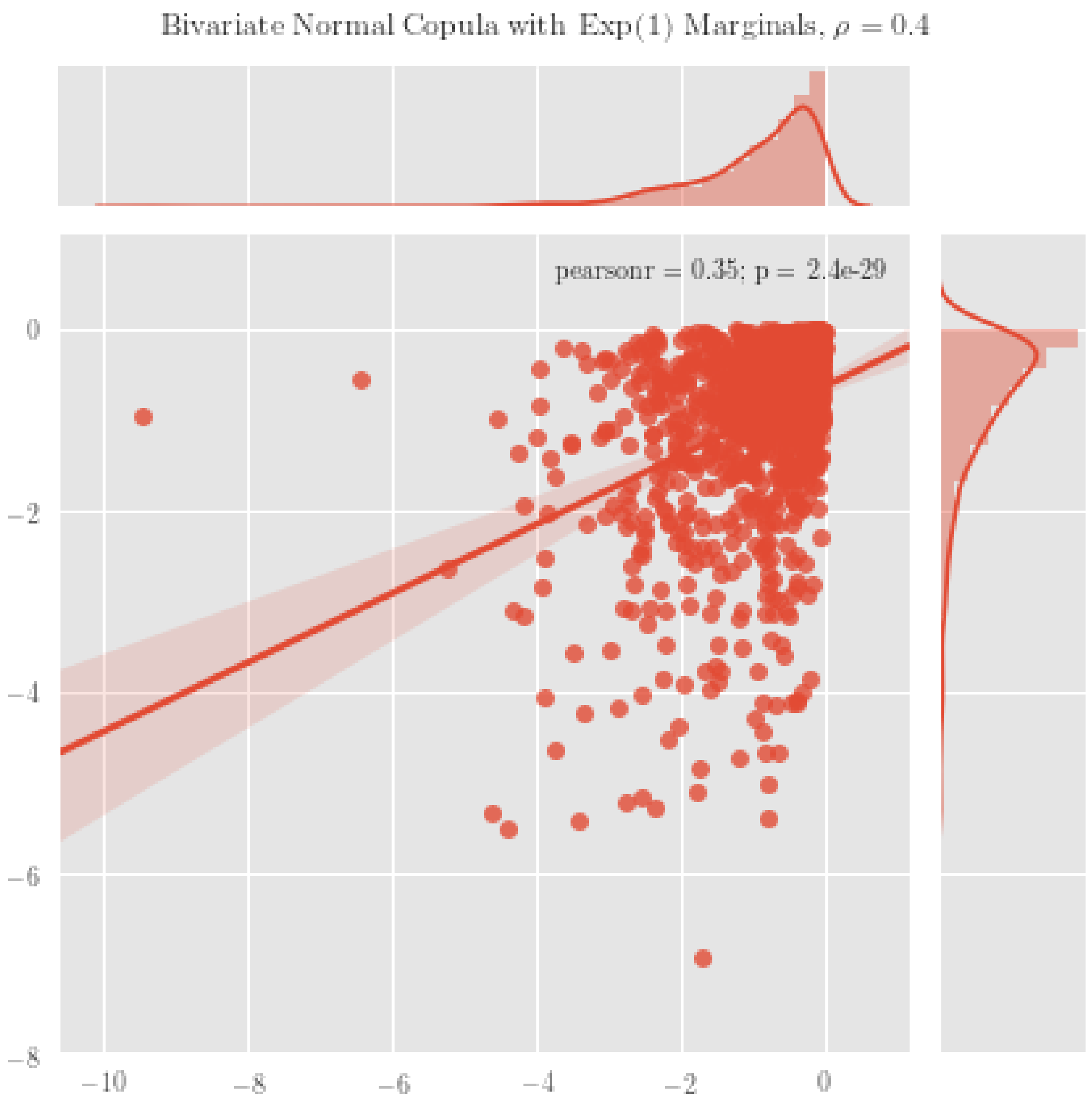


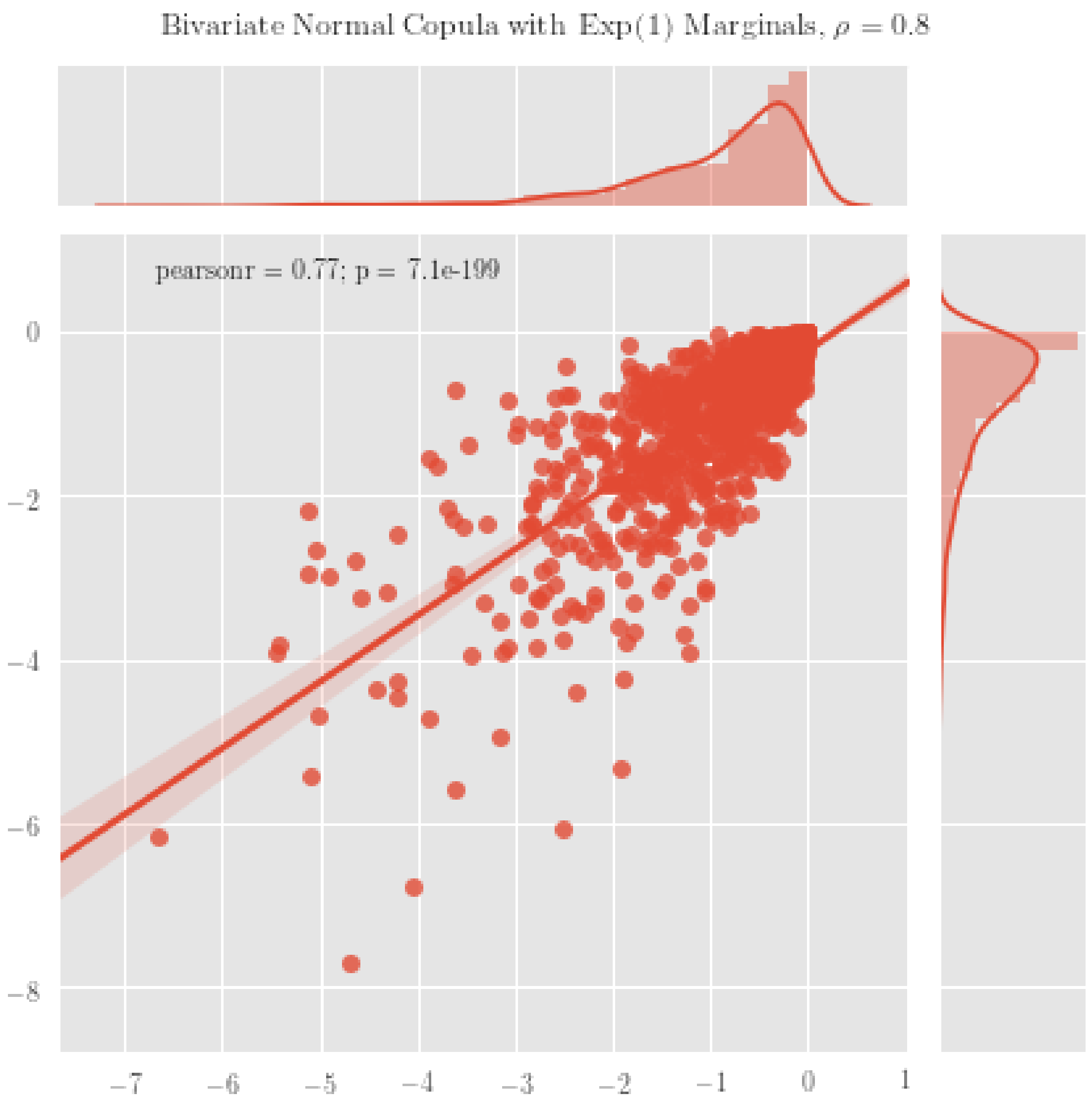
3.3 Gaussian Copula, Exponential(1) Marginal

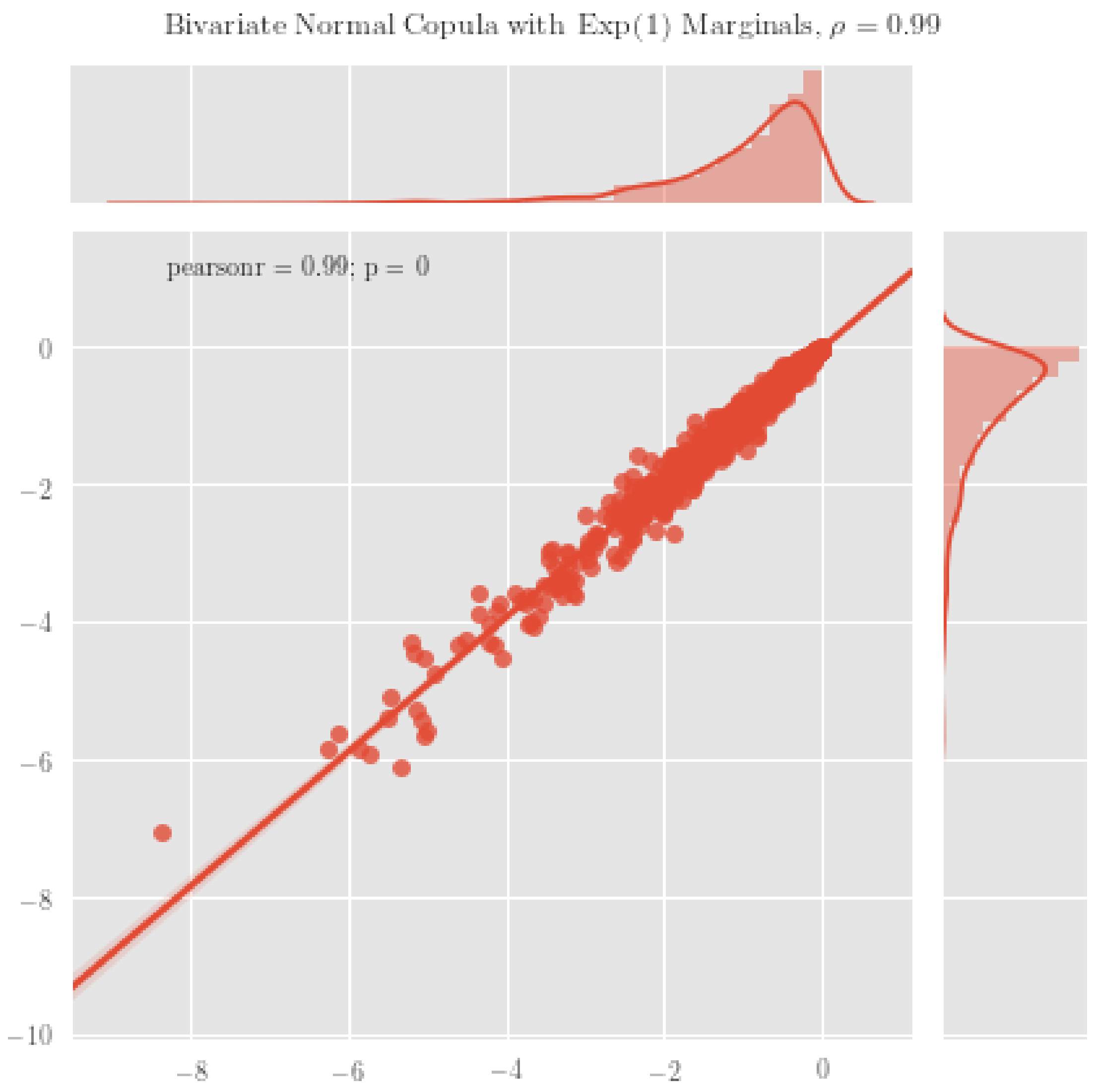
```
In [64]: gsexp_copula_sampler = GaussianCopula(
             cov_matrix=np.array([[1,0], [0,1]]),
             marginal_inv_cdfs=exponential_marginal_inv([1,1])
         )
         for rho in [0, 0.4, 0.8, 0.99]:
             gsexp_copula_sampler.reset_cov(
                 np.array([[1,rho], [rho,1]])
             )
             g = gsexp_copula_sampler.bivariate_plot(1000)
             plt.subplots_adjust(top=0.93)
```

```
g.fig.suptitle('Bivariate Normal Copula with Exp(1) Marginals, '
                  +r'$\rho={}$'.format(rho))
```




Bivariate Normal Copula with Exp(1) Marginals, $\rho = 0.4$
pearsonr = 0.35; p = 2.4e-29
0
−2
−4
−6
−8
−10
−8
−6
−4
−2
0

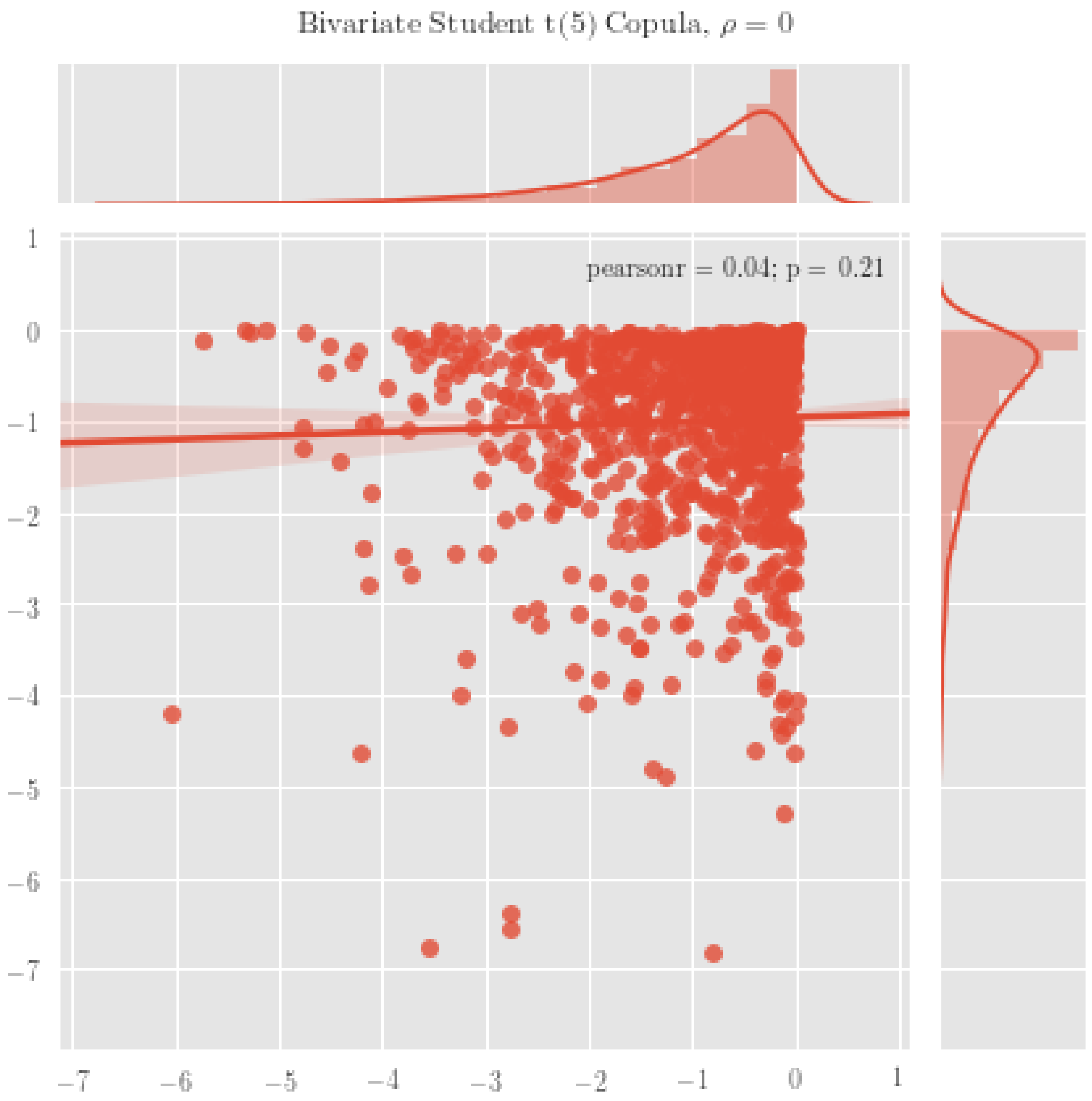

Bivariate Normal Copula with Exp(1) Marginals, $\rho = 0.8$
pearsonr = 0.77; p = 7.1e-199
0
−2
−4
−6
−8
−7
−6
−5
−4
−3
−2
−1
0
1



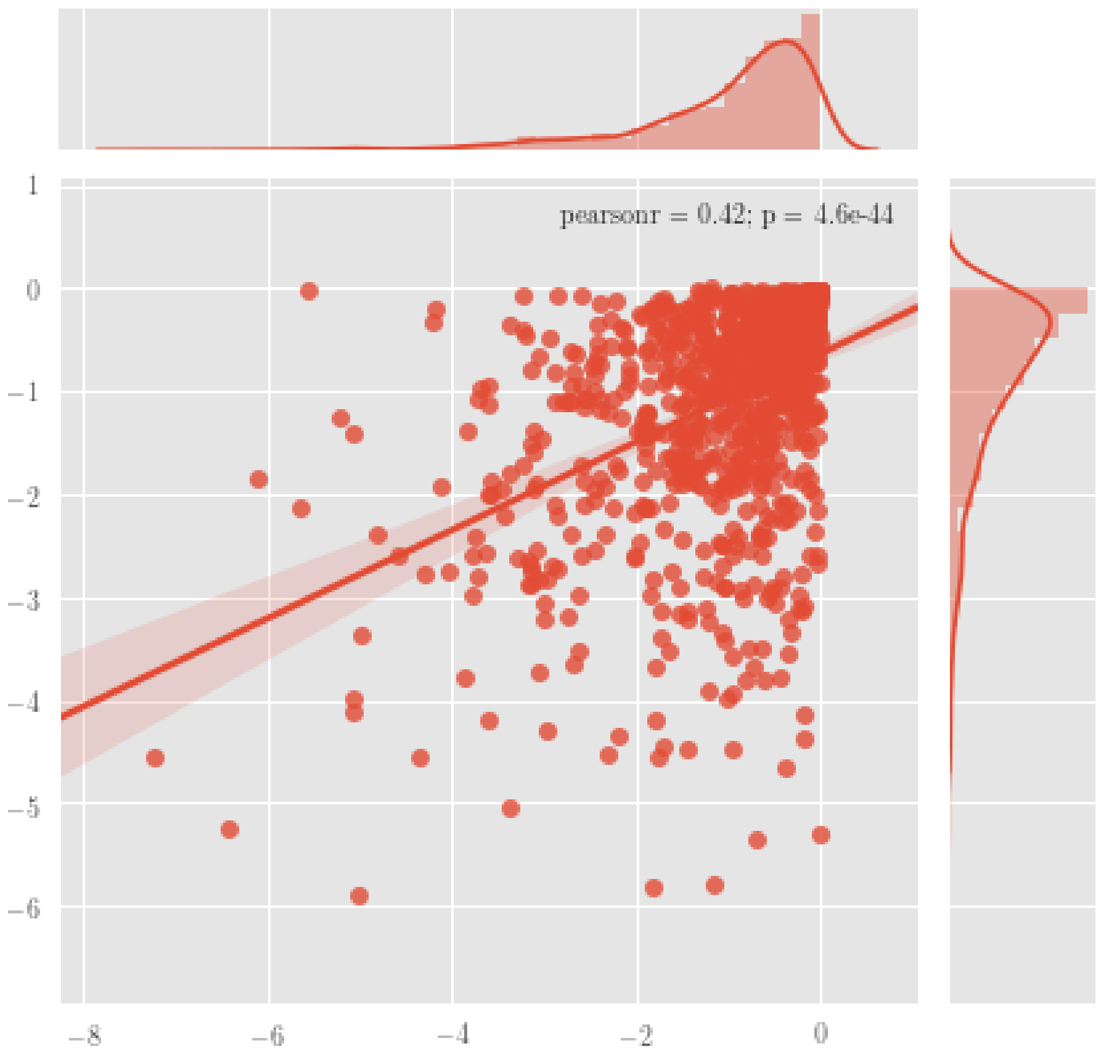
3.4 Student T(5) Copula, Exponential(1) Marginal

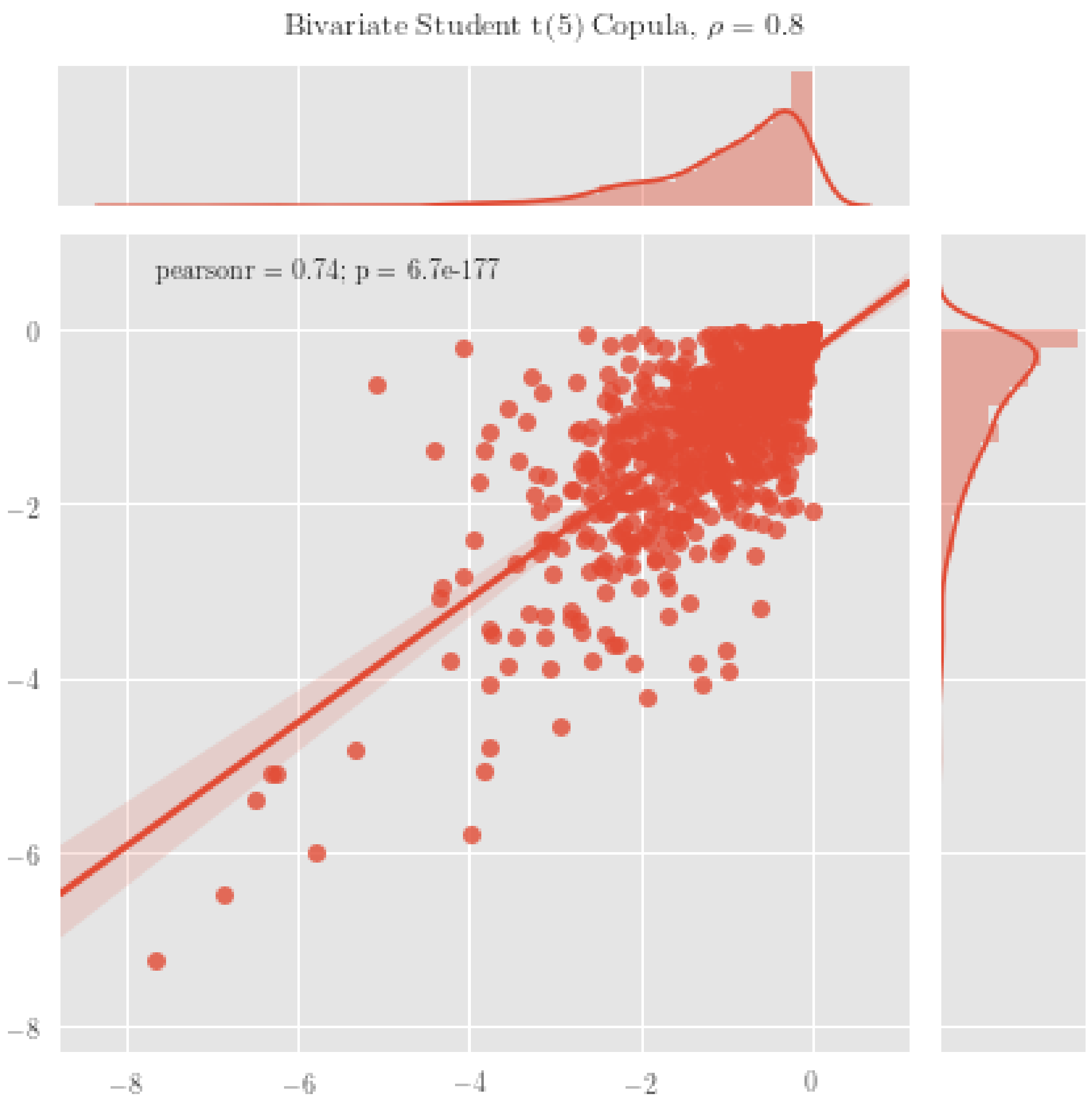
```
In [61]: t_copula_sampler = StudentTCopula(
             df=5,
             cov_matrix=np.array([[1,0], [0,1]]),
             marginal_inv_cdfs=exponential_marginal_inv([1,1])
         )
         for rho in [0, 0.4, 0.8, 0.99]:
             t_copula_sampler.reset_cov(
                 np.array([[1,rho], [rho,1]])
             )
             g = t_copula_sampler.bivariate_plot(1000)
             plt.subplots_adjust(top=0.93)
```

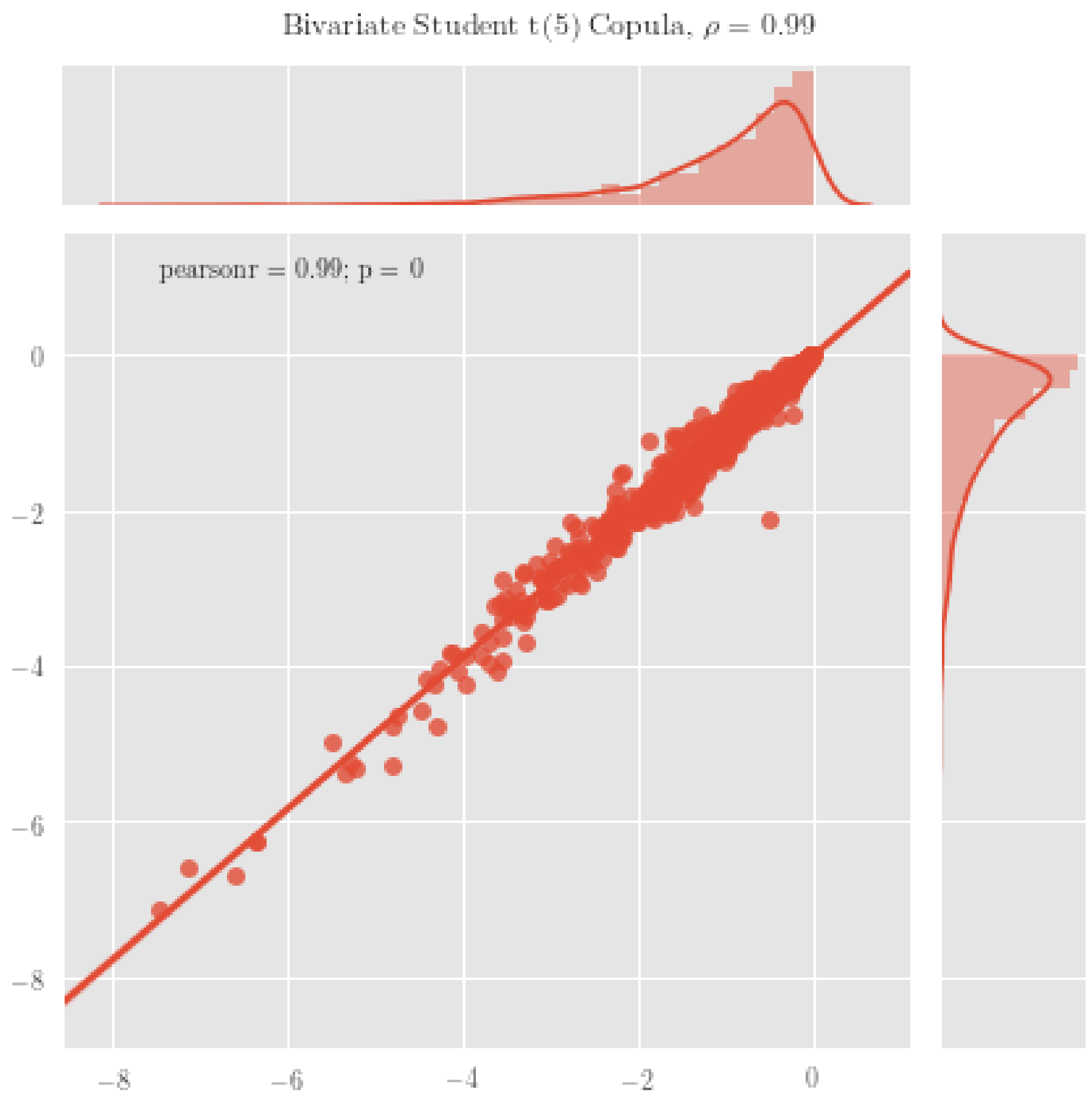
```
g.fig.suptitle('Bivariate Student t(5) Copula, '
              +r'$\rho={}$'.format(rho))
```



Bivariate Student t(5) Copula, $\rho = 0.4$




Bivariate Student t(5) Copula, $\rho = 0.8$
pearsonr = 0.74; p = 6.7e-177
0
−2
−4
−6
−8
−8
−6
−4
−2
0



Comments:

- In all the four cases, the scatters become more aligned together toward the regression line as the correlation coefficient increases. And we perceive more and more elliptical joint distribution in the middle subplot when ρ increases.

- In every single case study, X_1 and X_2, however, have the same marginal distribution no matter what ρ is. That is, the Copula method has successfully separated the marginal distribution apart from the correlation as it should have. In case **(a)**, the marginal distribution of X_1 and X_2 is Normal$(0, 1)$, in case **(b)**, it's Student $t(5)$, and finally in case **(3)** and **(4)**, the marginals are Exponential(1).

- Although in case **(3)** and **(4)** we used the same marginal distribution, the joint scatter plots look quite different. The different is in the dependence structure: t-copola seems to have more scatters around the boundary ($X_1 = 0$ and $X_2 = 0$) and the line $X_1 = X_2$ than the Gaussian

Copula. Such difference is more evident if we simulate $t(1)$ Copula. Not surprisingly, t(ν) copula approaches Gaussian copula when me make ν larger.

---

4 T Variables with Rejection Method

(a) The Cauchy pdf is $f_1(x) = \frac{1}{\pi}(1+x^2)^{-1}$. Therefore

$$\begin{aligned} g_n(x) = \frac{f_n(x)}{C_n f_1(x)} &= \frac{1}{C_n} \cdot \underbrace{\frac{\sqrt{\pi}\Gamma(\frac{n+1}{2})}{\sqrt{n}\Gamma(\frac{n}{2})}}_{\text{constant}} \cdot \frac{1+x^2}{(1+\frac{x^2}{n})^{\frac{n+1}{2}}} \\ &:= \frac{1}{C_n} A_n \psi_n(x) \in (0,1] \end{aligned} \tag{3}$$

The first constant part is easy to deal with. Moreover, everything is positive here, so $C_n > 0$ will make $g_n(x) > 0$ for all $n \in \mathbb{N}^+, x \in \mathbb{R}$. Now we consider how to make it less than or equal to 1. We calculate

$$\begin{aligned} \frac{\partial \psi_n(x)}{\partial x} &= \frac{2x(1+\frac{x^2}{n})^{\frac{n+1}{2}} - \frac{x(n+1)}{n}(1+x^2)(1+\frac{x^2}{n})^{\frac{n+1}{2}-1}}{(1+\frac{x^2}{n})^{n+1}} \\ &= x\left(1+\frac{x^2}{n}\right)^{-\frac{n+1}{2}} \left[2 - \frac{(n+1)(1+x^2)}{n(1+\frac{x^2}{n})}\right] \\ &= x\left(1+\frac{x^2}{n}\right)^{-\frac{n+1}{2}} \left[1 - \frac{1+nx^2}{n+x^2}\right] \end{aligned} \tag{4}$$

Clearly, for $n > 1$: $x^2 > 1 \iff (n-1)x^2 > n-1 \iff 1+nx^2 > n+x^2 \iff \left(1 - \frac{1+nx^2}{n+x^2}\right) < 0$. Therefore, $\frac{\partial \psi_n(x)}{\partial x} < 0$ whenever $x^2 > 1$, and $\frac{\partial \psi_n(x)}{\partial x} > 0$ whenever $x^2 < 1$. Hence,

$$\sup_{x\in\mathbb{R}} \psi_n(x) = \psi_n(\pm 1) = \frac{2}{(1+\frac{1}{n})^{\frac{n+1}{2}}}$$

Our choice of C_n satisfies $\frac{1}{C_n} A_n \psi_n(x) \le 1$, $\forall x \in \mathbb{R}$. Therefore, the smallest C_n that satisfies the above condition is determined by

$$C_n^* = A_n \sup_{x\in\mathbb{R}} \psi_n(x) = \frac{\sqrt{\pi}\Gamma(\frac{n+1}{2})}{\sqrt{n}\Gamma(\frac{n}{2})} \frac{2}{(1+\frac{1}{n})^{\frac{n+1}{2}}} \tag{5}$$

(b) With the help of programs:

$$C_2 \approx 1.20920; \quad C_3 \approx 1.29904; \quad C_5 \approx 1.38029$$

Since we define C_∞ with

$$g_\infty(x) = \frac{\phi(x)}{C_\infty f_1(x)} = \frac{1}{C_\infty}\sqrt{\frac{\pi}{2}} \frac{1+x^2}{e^{\frac{x^2}{2}}} = \frac{1}{C_\infty}\sqrt{\frac{\pi}{2}}\eta(x) \in (0,1] \tag{6}$$

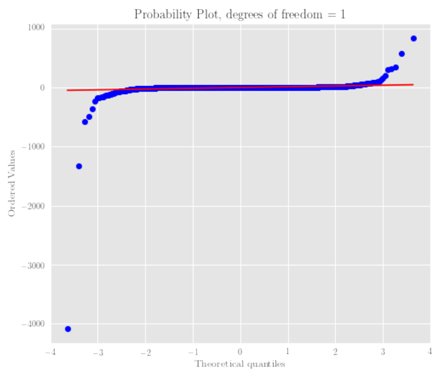
To be consistent with the C_n^* we derived in part **(a)**, $\eta(x)$ also takes its maximum at $x^2 = 1$, and it's easy to be verified by taking derivatives: $\frac{d\eta(x)}{dx} = xe^{-x^2/2}(1-x^2)$. Therefore, we have

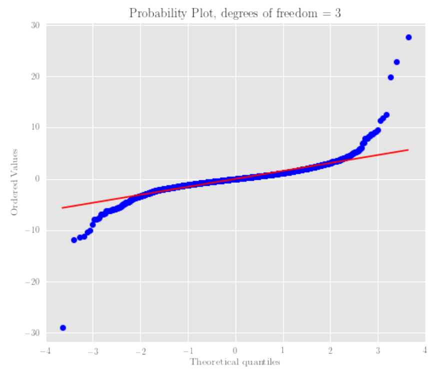
$$C_\infty = \sqrt{\frac{\pi}{2}}\eta(\pm 1) = \sqrt{\frac{2\pi}{e}} \approx 1.520347$$

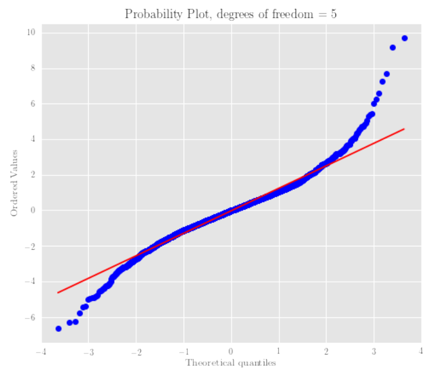
(c)

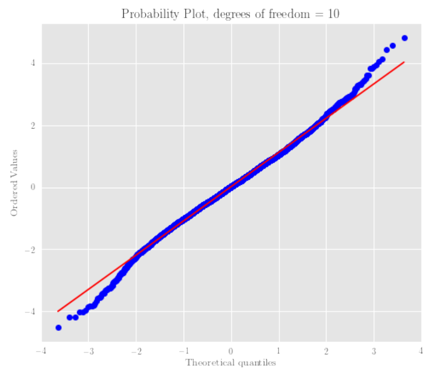
```
In [5]: def t_scaler(n):
            if n > 340: return np.sqrt(2*np.pi/np.e)
            return (gamma(n/2+0.5)/(np.sqrt(n/np.pi)*gamma(n/2))) * (
                2/((1+1/n)**(n/2+0.5)))

        def cauchy_inv(x0, gamma):
            return lambda u: gamma*np.tan(np.pi*u-np.pi/2)+x0

        def cauchy_induced_t_rej(df):
            C = t_scaler(df)
            return lambda x: (1/C)*t.pdf(x, df)*np.pi*(1+x**2)
```

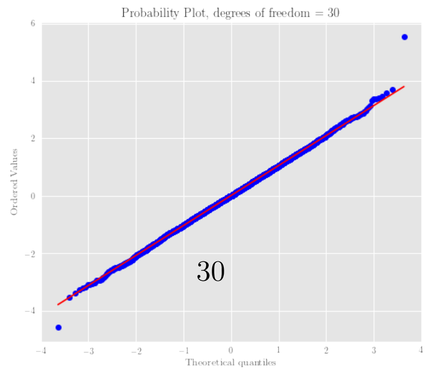
```
In [9]: f, axes = plt.subplots(5,1,figsize=(8, 40))
        for i, df in enumerate([1, 3, 5, 10, 30]):
            cauchy_sampler = ProbIntegralGenerator(cauchy_inv(0,1))
            t_rejection_sampler = RejectionGenerator(
                inducing_rng=cauchy_sampler,
                rej_func=cauchy_induced_t_rej(df=df)
            )
            t_sample = t_rejection_sampler.draw(5000)
            qq = stats.probplot(
                t_sample, dist="norm", plot=axes[i])
            axes[i].set_title(
                'Probability Plot, degrees of freedom = {}'.format(df))
```


Probability Plot, degrees of freedom = 1
Ordered Values
Theoretical quantiles


Probability Plot, degrees of freedom = 3
Ordered Values
Theoretical quantiles


Probability Plot, degrees of freedom = 5
Ordered Values
Theoretical quantiles


Probability Plot, degrees of freedom = 10
Ordered Values
Theoretical quantiles


Probability Plot, degrees of freedom = 30
Ordered Values
Theoretical quantiles

```
In [ ]:
```