Homework #4: Monte Carlo Stimulation

姓名：葉于廷

學號：410878048

目標：

- Part.1 認識蒙地卡羅方法。
- Part.2 檢視峰度、偏度為標準常態分佈、進而延伸 G1, G2 為標準常態分布、JB 統計量為卡方自由度為二的分布
- Part.3 寫 Jarque-beta 檢定並檢測是否具有檢定效果
- Part.4 比較不同檢定(Kolmogorow、 Anderson)在相異樣本的檢定效果
- Part.5 三門問題
- Part.6 結語

Part.1 Introduction

此方法興起是因電腦的出現，加速了計算與更多想法的萌生。

使用數統機率的概念來進行模擬的計畫，像機率可用抽很多次的情況中了幾次來計算

模擬的結果應與先計算的結果應一致，又或是不會計算，先模擬看結果，進而對想探討的目標有更進一步的認識

In [ ]:
```
# install packages
from scipy.stats import skew, kurtosis, norm, chi2, t, kstest, anderson, jarque_bera
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
from statsmodels.distributions.empirical_distribution import ECDF
```

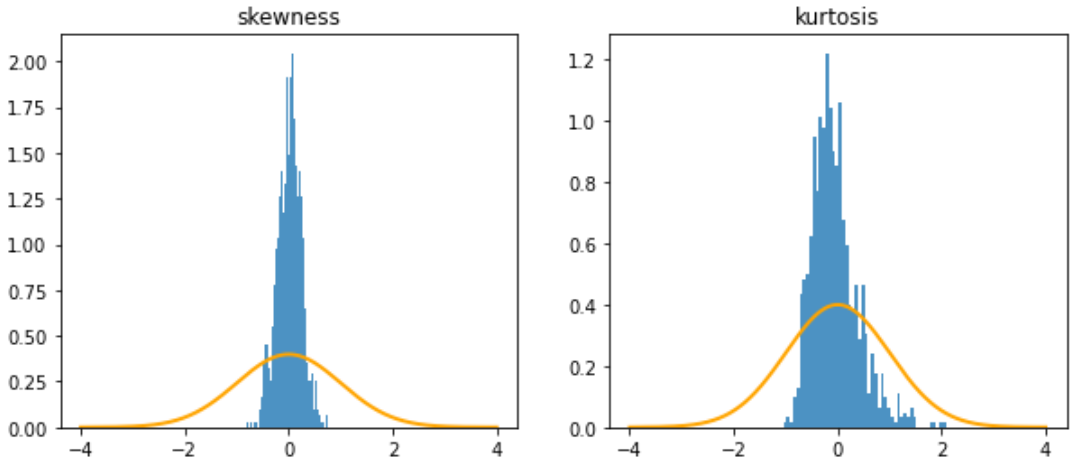
Part.2 Jarque–Bera statistics

想要觀測一個分布是否為常態，可看他的偏度是否是為**0**，但只要是對稱分布偏度都會是 **0** 我們在額外觀察峰度樣貌，常態理論峰度為 **0**。

[圖一]具體的分配我們多次抽樣 **N(0, 1)** 觀察 **skewness**、**kurtosis**，如我們預期中心點在 **0**， 但變異數似乎較小，不像標準常態。

In [ ]:
```
n = 100
N = 1000
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

# simulate the sample from normal dist.
norms = norm.rvs(loc=0, scale=1, size=(n, N))

#  watch skewness
g1 = skew(norms)
ax[0].hist(g1, bins=50, density=True, alpha=0.8)
x = np.linspace(-4, 4, 1000)
ax[0].plot(x, norm.pdf(x), lw=2, color="orange")
```

```
ax[0].set_title("skewness")

# watch kurtosis
g2 = kurtosis(norms, fisher=False) - 3
ax[1].hist(g2, bins=50, density=True, alpha=0.8)
x = np.linspace(-4, 4, 1000)
ax[1].plot(x, norm.pdf(x), lw=2, color="orange")
ax[1].set_title("kurtosis")

plt.show()
```



因此我們對 **skewness kurtosis** 做一個調整使統計量服從標準常態

模擬乘上常數的分配

$$G_1 = \sqrt{\frac{n}{6}}\hat{s}, \quad G_2 = \sqrt{\frac{n}{24}}(\hat{k} - 3)$$

In [ ]:

```
def G1(n):
    m = 50000
    def g1(x): return np.sqrt(len(x)/6) * skew(x)
    g1s = [0] * m
    for i in range(m):
        x_normal = np.random.normal(size=n, loc=0, scale=1)
        g1s[i] = g1(x_normal)
    return g1s


def G2(n):
    m = 50000
    def g2(x): return np.sqrt(n / 24) * (kurtosis(x, fisher=False) - 3)
    g2s = [0] * m
    for i in range(m):
        x_normal = np.random.normal(size=n, loc=0, scale=1)
        g2s[i] = g2(x_normal)
    return g2s
```

我們將透過四種方法檢定標準常態

1. 利用 **hist** 與 **normal pdf** 對照

2. 利用 **qqplot** 觀察點是否落在線上

3. 利用 **Kolmogorov test** 檢定 是否常態

4. 利用 **ecdf** 對照

In [ ]:

```
def obnormal(data, n, statistic):
```

```
    fig, ax = plt.subplots(1, 2, figsize=(10, 3.5))

    # histogram
    ax[0].hist(data, density=True, alpha=0.7, color="orange", bins = 100)

    # normal line
    x = np.linspace(-4, 4, 1000)
    y = norm.pdf(x, loc=0, scale=1)
    ax[0].plot(x, y, color="red", lw=3, alpha=0.6, label="theory")
    ax[0].set_title("Hist")
    ax[0].legend(loc="upper left")

    # ecdf
    ycdf = norm.cdf(x, loc=0, scale=1)
    dataecdf = ECDF(data)
    ax[1].plot(x, ycdf, alpha=0.5, lw=3, label="theory")
    ax[1].plot(dataecdf.x, dataecdf.y, alpha=0.5, lw=3, label="sampling")
    ax[1].set_title("cdf")
    ax[1].legend(loc="upper left")
    fig.suptitle("The " + str(n) + " samples from " +
                 statistic.__name__ + " distribution")
    plt.show()


# function pack
def gstat(func, n):
    data = func(n)
    obnormal(data, n, func)
    return(np.array(data))
```

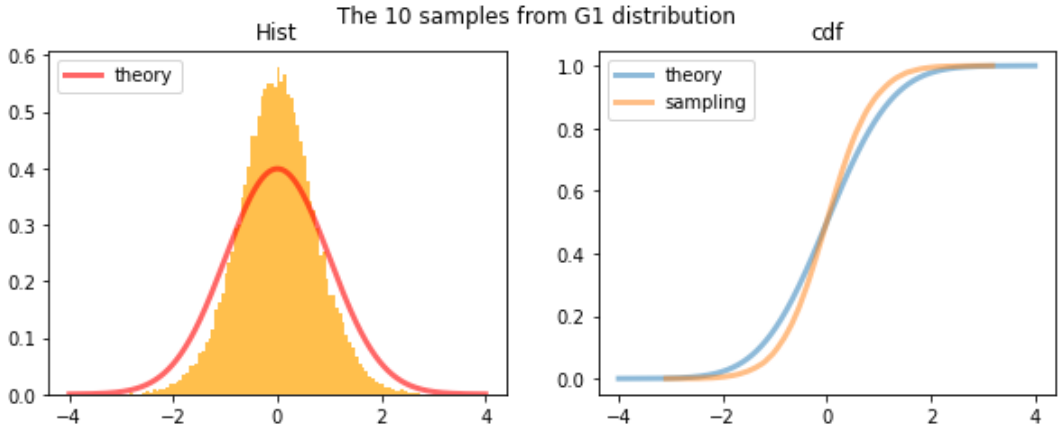
一開始我們模擬 **G1 n = 10, 20, 30, 50, 100, 300, 500**

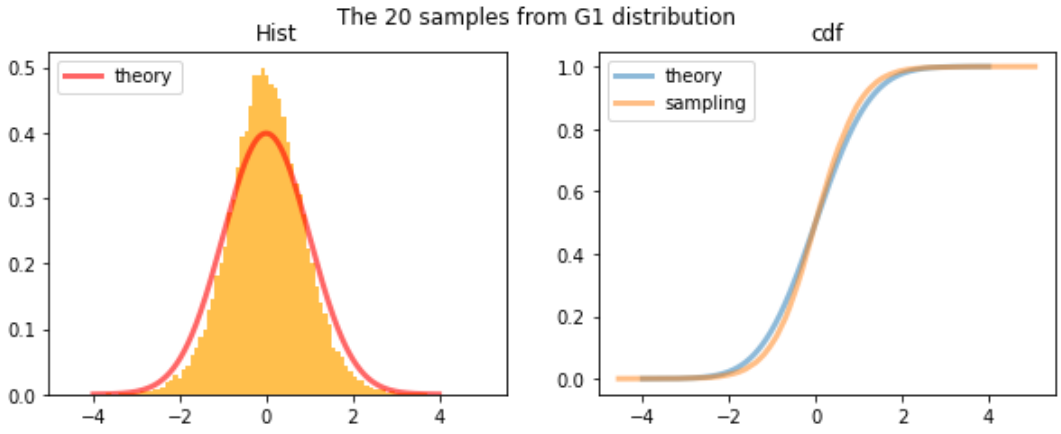
hist 靠近理論線 兩個 **cdf** 也相當靠近

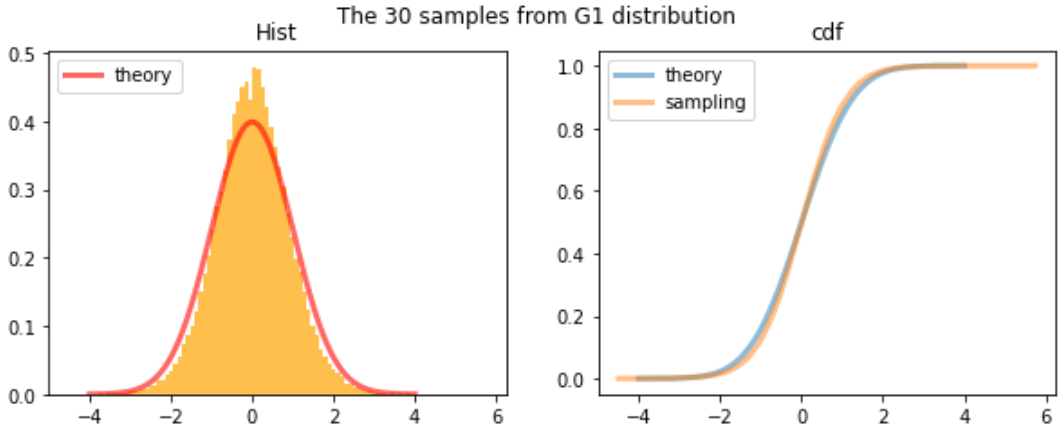
可認為 **G1** 的確近似標準常態

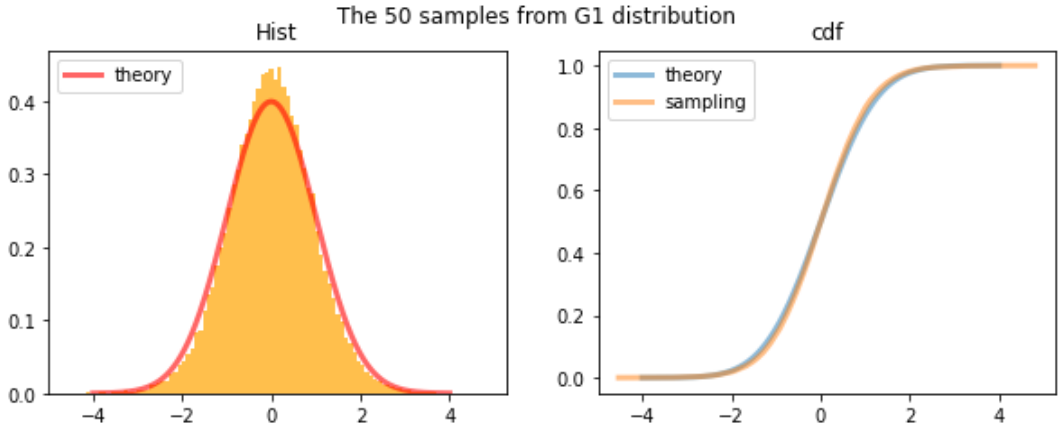
In [ ]:

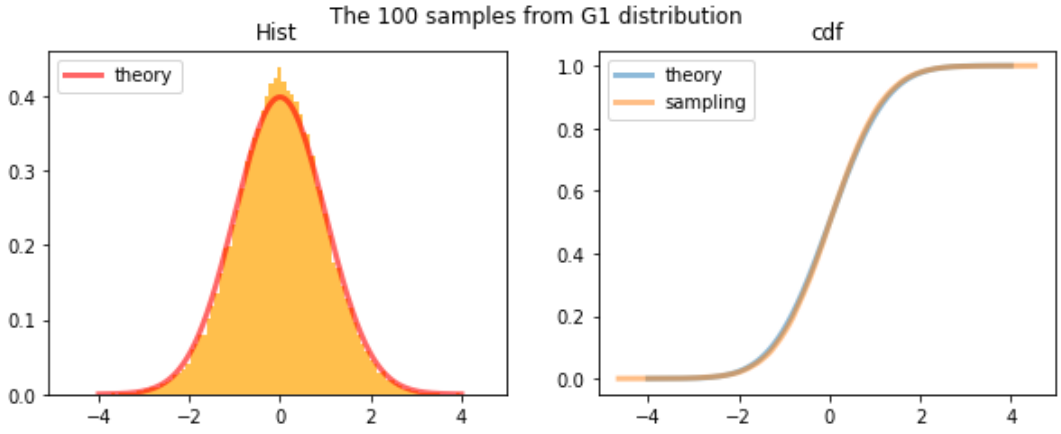
```
# %%
g110 = gstat(G1, 10)
g120 = gstat(G1, 20)
g130 = gstat(G1, 30)
g150 = gstat(G1, 50)
g1100 = gstat(G1, 100)
g1300 = gstat(G1, 300)
g1500 = gstat(G1, 500)
```

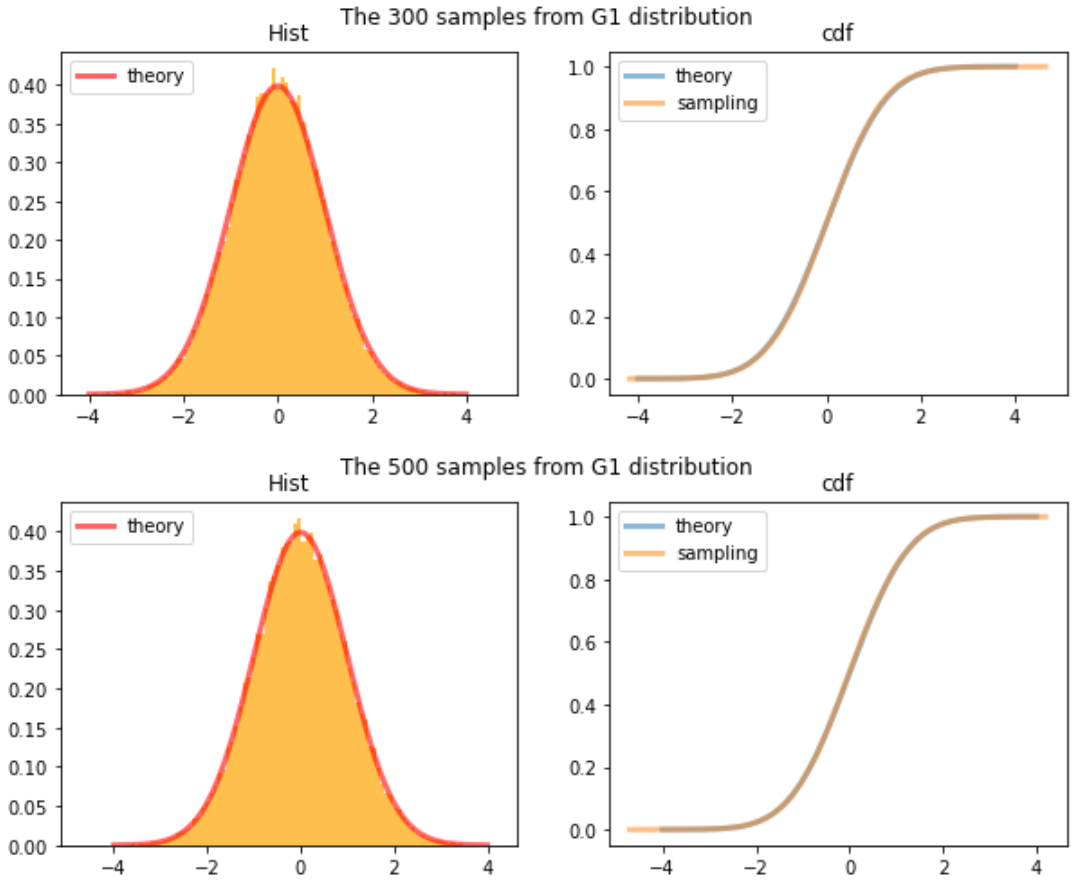



The 20 samples from G1 distribution
Hist
theory
cdf
theory
sampling


The 30 samples from G1 distribution
Hist
theory
cdf
theory
sampling


The 50 samples from G1 distribution
Hist
theory
cdf
theory
sampling


The 100 samples from G1 distribution
Hist
theory
cdf
theory
sampling

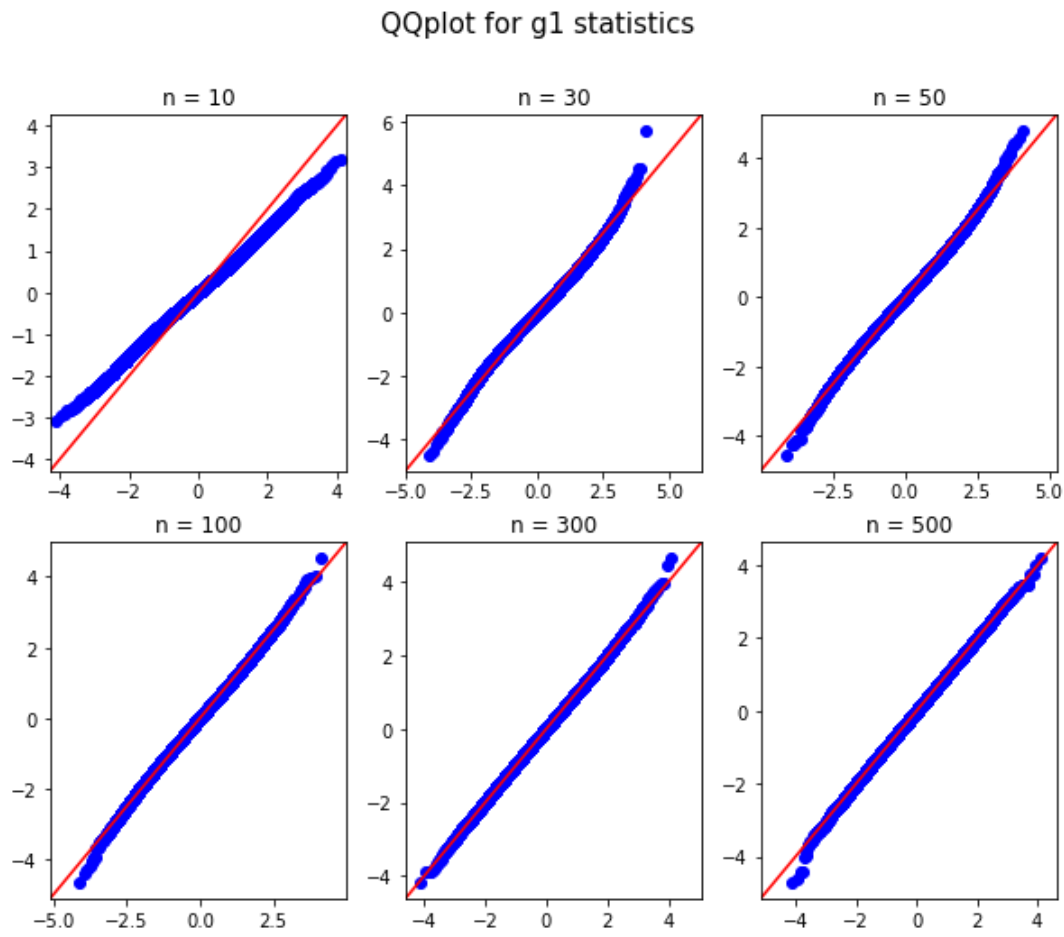


qqplot: 點相當貼其線 代表越像常態

看得出 **G1** 隨樣本數越大越貼齊線 越像常態

In [ ]:
```
fig, ax = plt.subplots(2, 3, figsize = (10, 8))
g1df = pd.DataFrame(np.array([g110, g130, g150, g1100, g1300, g1500]).T)
n = [10, 30, 50, 100, 300, 500]
k = 0
for i in range(2):
    for j in range(3):
        sm.qqplot(g1df[k], line = "45", ax = ax[i, j])
        ax[i, j].set_ylabel('')
        ax[i, j].set_xlabel('')
        ax[i, j].set_title('n = '+ str(n[k]))
        k = k + 1
plt.suptitle("QQplot for g1 statistics", fontsize = 15)
```

Out[ ]:
```
Text(0.5, 0.98, 'QQplot for g1 statistics')
```



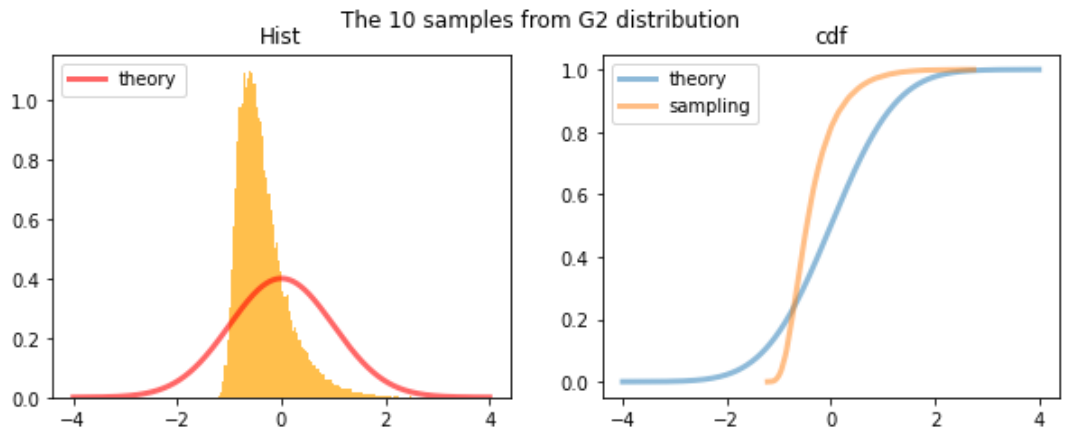
接著我們模擬 **G2 n = 10, 20, 30, 50, 100, 300, 500**

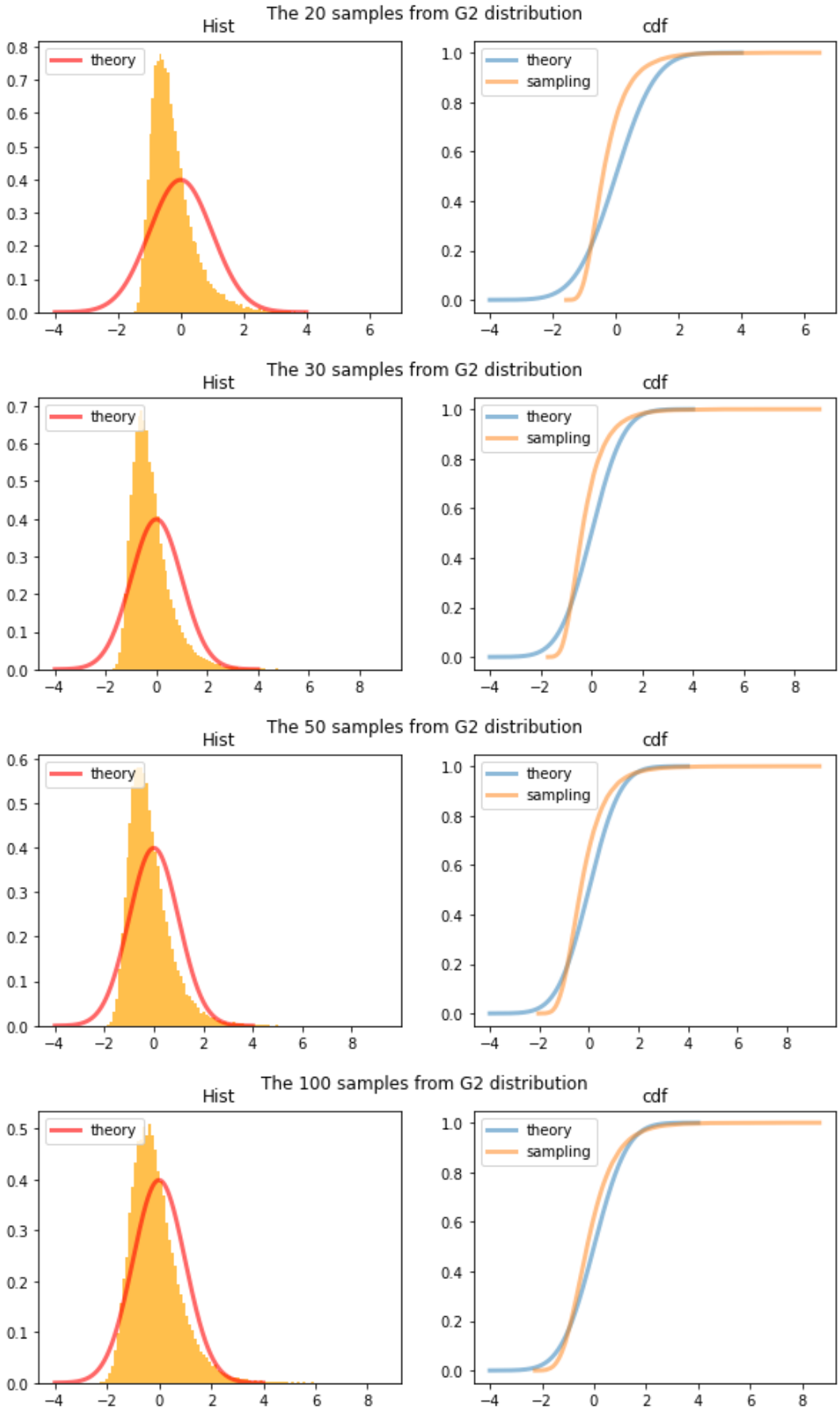
發現 **hist** 越來越靠近線 **cdf** 也逐漸對齊

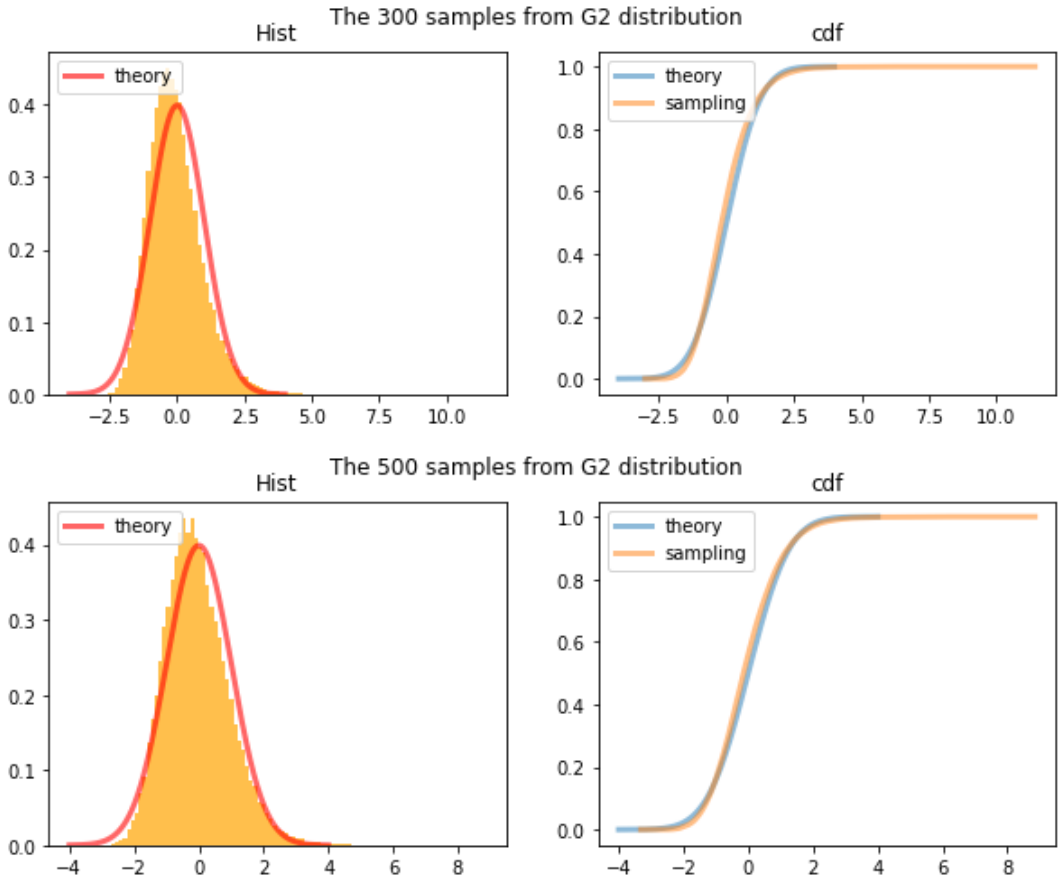
我們認定 **G2** 樣本數 **n** 很大會越來越大時看起來有符從標準常態的趨勢

In [ ]:

```
# %%
g210 = gstat(G2, 10)
g220 = gstat(G2, 20)
g230 = gstat(G2, 30)
g250 = gstat(G2, 50)
g2100 = gstat(G2, 100)
g2300 = gstat(G2, 300)
g2500 = gstat(G2, 500)
```



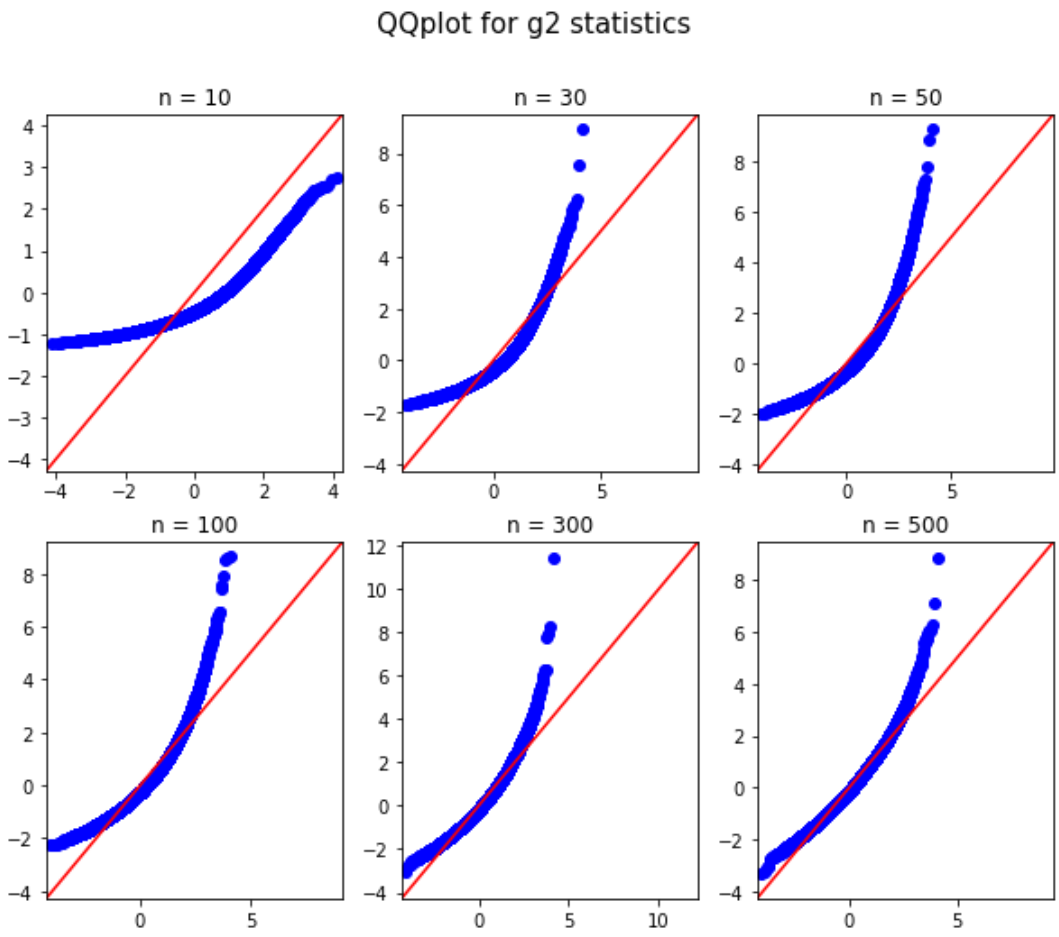

The 20 samples from G2 distribution
Hist
cdf
theory
sampling
The 30 samples from G2 distribution
Hist
cdf
theory
sampling
The 50 samples from G2 distribution
Hist
cdf
theory
sampling
The 100 samples from G2 distribution
Hist
cdf
theory
sampling



G2 qqplot 在樣本數 500 時不像標準常態 點大致沒有貼齊線

In [ ]:
```
fig, ax = plt.subplots(2, 3, figsize = (10, 8))
g2df = pd.DataFrame(np.array([g210, g230, g250, g2100, g2300, g2500]).T)
n = [10, 30, 50, 100, 300, 500]
k = 0
for i in range(2):
    for j in range(3):
        sm.qqplot(g2df[k], line = "45", ax = ax[i, j])
        ax[i, j].set_ylabel('')
        ax[i, j].set_xlabel('')
        ax[i, j].set_title('n = '+ str(n[k]))
        k = k + 1
plt.suptitle("QQplot for g2 statistics", fontsize = 15)
```

Out[ ]:
```
Text(0.5, 0.98, 'QQplot for g2 statistics')
```



Kolmogorov 檢定

$$H0：\text{此分配為常態} \quad Ha：\text{此分配不為常態}$$

這是一個可以檢視資料是否為常態的檢定

概念是將 **ecdf** 對照 找出垂直差距最大的資料點位置

對照 **K** 分配決定統計量

其中 **n = 500** 時 檢定已認為 **G1** 服從標準常態

G2 則都沒有顯著 不認為這時候的 **G2** 為標準常態

In [ ]:
```
g1stat = np.empty(6)
g1p = np.empty(6)
g2stat = np.empty(6)
g2p = np.empty(6)
for i in range(6):
    g1stat[i], g1p[i] = kstest(g1df[i], 'norm')
    g2stat[i], g2p[i] = kstest(g2df[i], 'norm')


koldf = pd.DataFrame(np.array([g1stat, g1p, g2stat, g2p]).T, columns = ['G1_stat', 'G1_p', 'G2_stat',
koldf["G1_pvalue"] = [">= 0.05"  if i >= 0.05 else "< 0.05" for i in koldf.G1_p]
koldf["G2_pvalue"] = [">= 0.05"  if i >= 0.05 else "< 0.05" for i in koldf.G2_p]
koldf.drop(columns = ['G2_p', 'G1_p'], inplace = True)
koldf['n'] = [10, 30, 50, 100, 300, 500]
koldf.set_index("n", inplace = True)
print("G1 and G2 are under Kolmogorov test")
koldf.loc[:, ['G1_stat', "G1_pvalue", 'G2_stat', "G2_pvalue"]]
```

```
G1 and G2 are under Kolmogorov test
```

Out[ ]:

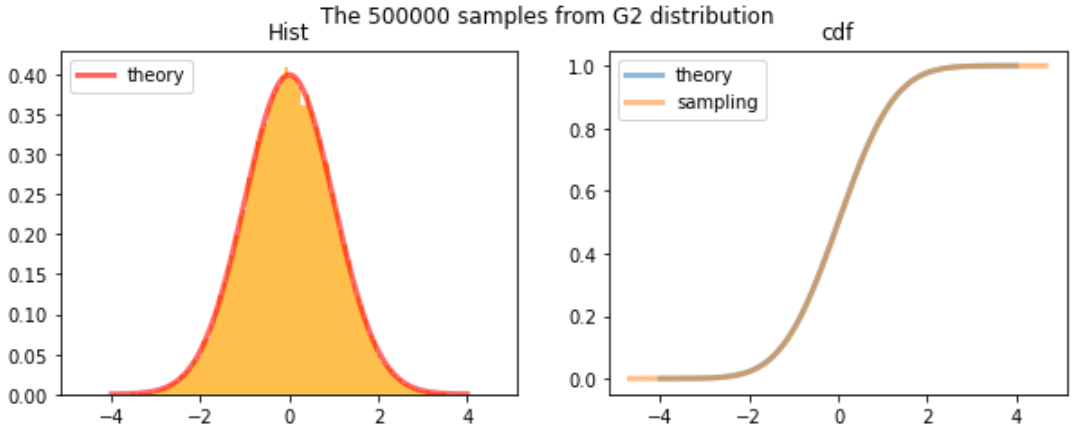
n	G1_stat	G1_pvalue	G2_stat	G2_pvalue

n	G1_stat	G1_pvalue	G2_stat	G2_pvalue
n				
10	0.077559	< 0.05	0.313053	< 0.05
30	0.034855	< 0.05	0.200942	< 0.05
50	0.023590	< 0.05	0.162234	< 0.05
100	0.014479	< 0.05	0.121791	< 0.05
300	0.006887	< 0.05	0.078792	< 0.05
500	0.005239	>= 0.05	0.060537	< 0.05

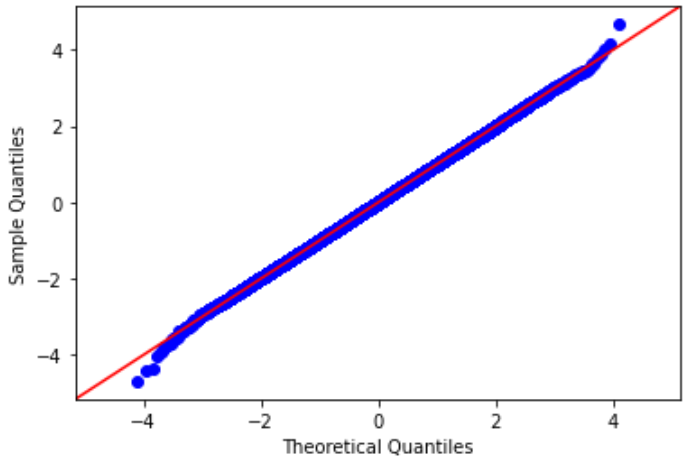
將 **G2** 拉大到樣本數 **500000** 檢定顯著 抽如此多 其原因可能是不對稱使要抽取的樣本要更多。

HIST 有越來越靠近紅線的趨勢 加上檢定 認定是漸進標準常態!

In [ ]:
```
g2100000 = gstat(G2, 500000)
sm.qqplot(g2100000, line = '45')
s, pvalue = kstest(g2100000, 'norm')
pvalue = ">= 0.05" if pvalue > 0.05 else "< 0.05"
t = "Kolmogorov test : statistics = {}, p-value {}".format(round(s, 5), pvalue)
print(t)
```




```
Kolmogorov test : statistics = 0.00249, p-value >= 0.05
```



在峰度和偏度互為獨立下，進而推得 G_3 為卡方自由度為 **2** 的分配

$$G_3 = {G_1}^2 + {G_2}^2,$$

$$where \;\; G_1 = \sqrt{\frac{n}{6}}\,\hat{s}, \;\; G_2 = \sqrt{\frac{n}{24}}\,(\hat{k} - 3)$$

常態的偏度和峰度為 **0**，可推測 **JB** 的的數值欲低我們會更有證據說明常態的性質

In [ ]:
```
# make G3
def G3(n):
    m = 10000
    def g2(x): return np.sqrt(len(x) / 24) * kurtosis(x)
    def g1(x): return np.sqrt(len(x) / 6) * skew(x)
    def g3(x): return g2(x) ** 2 + g1(x) ** 2
    g3s = [0] * m
    for i in range(m):
        x_normal = np.random.normal(size=n, loc=0, scale=1)
        g3s[i] = g3(x_normal)
    return g3s

# check chisquare
def obchisquare(m, g3s):
    # %%
    fig, ax = plt.subplots(1, 2, figsize=(10, 4))
    n = 10000
    y = np.random.chisquare(df=2, size=n)

    ax[0].plot(np.linspace(0, 20, 1000), chi2.pdf(np.linspace(0, 20, 1000), 2), alpha=0.5, lw=3, labe
    ax[0].hist(g3s, alpha=0.5, lw=3, label="G3", density=True, bins = 100)

    ax[0].set_ylabel("density")
    ax[0].legend()
    ax[0].set_title("Histogram")
    necdf = ECDF(y)
    dataecdf = ECDF(g3s)
    ax[1].plot(necdf.x, necdf.y, alpha=0.5, lw=3, label="$\chi^2_{(2)}$")
    ax[1].plot(dataecdf.x, dataecdf.y, alpha=0.5, lw=3, label="G3")
    ax[1].set_ylabel("cdf")
    ax[1].legend(loc="upper right")
    ax[1].set_title("ECDF")
    plt.suptitle("The " + str(m) + " samples from G3 distribution", fontsize = 15)

# pack the functions into the function
def g3stat(n):
    data = G3(n)
    obchisquare(n, data)
    return(np.array(data))
```

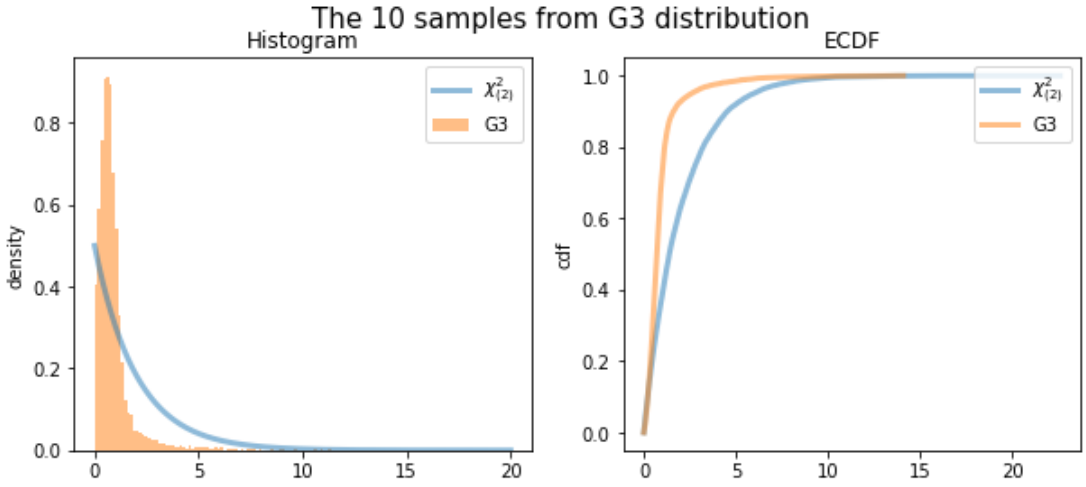
下面我們觀察 不同樣本大小的 **G3**

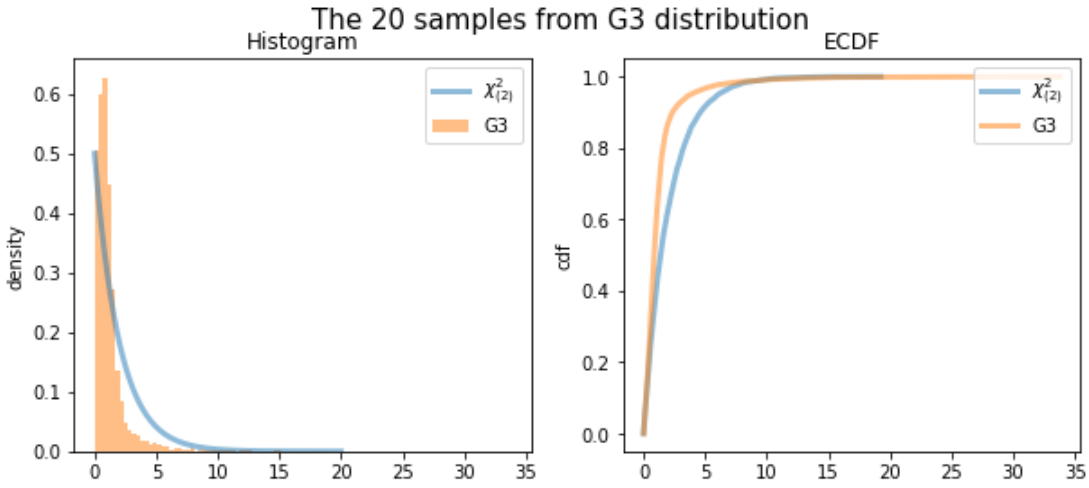
ecdf 隨 樣本變大越貼近線

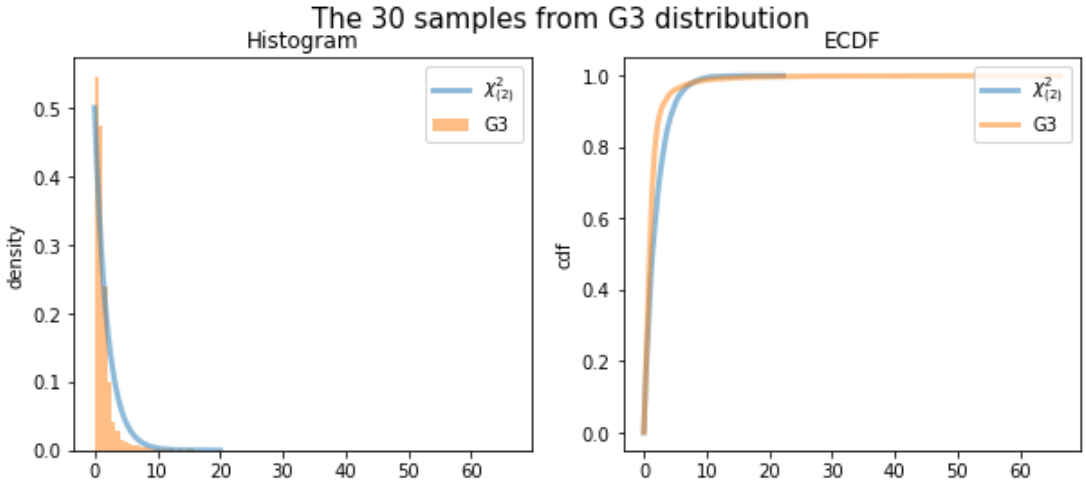
這裡要大一點的樣本才有卡方的樣子

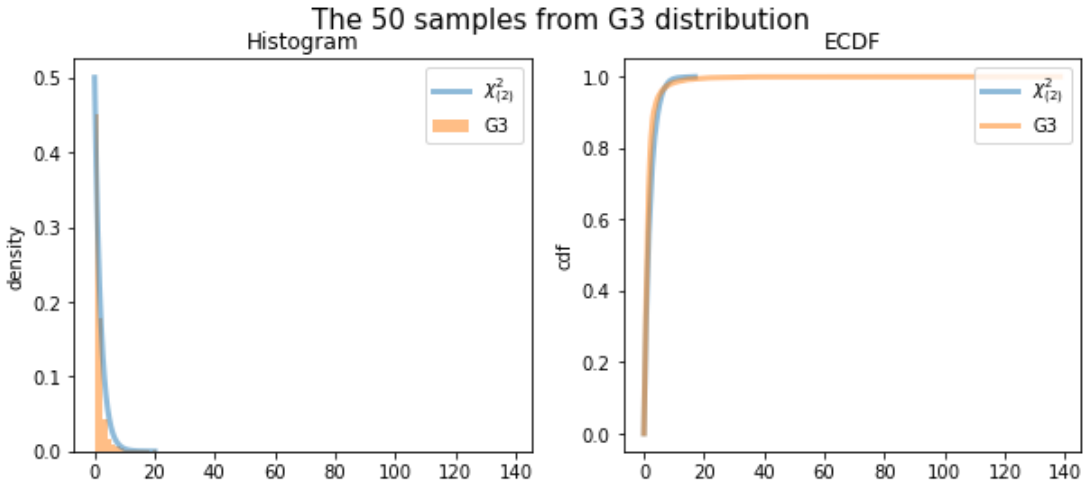
In [ ]:
```
g3stat(10)
g3stat(20)
g3stat(30)
g3stat(50)
g3stat(100)
g3stat(300)
g3stat(500)
```

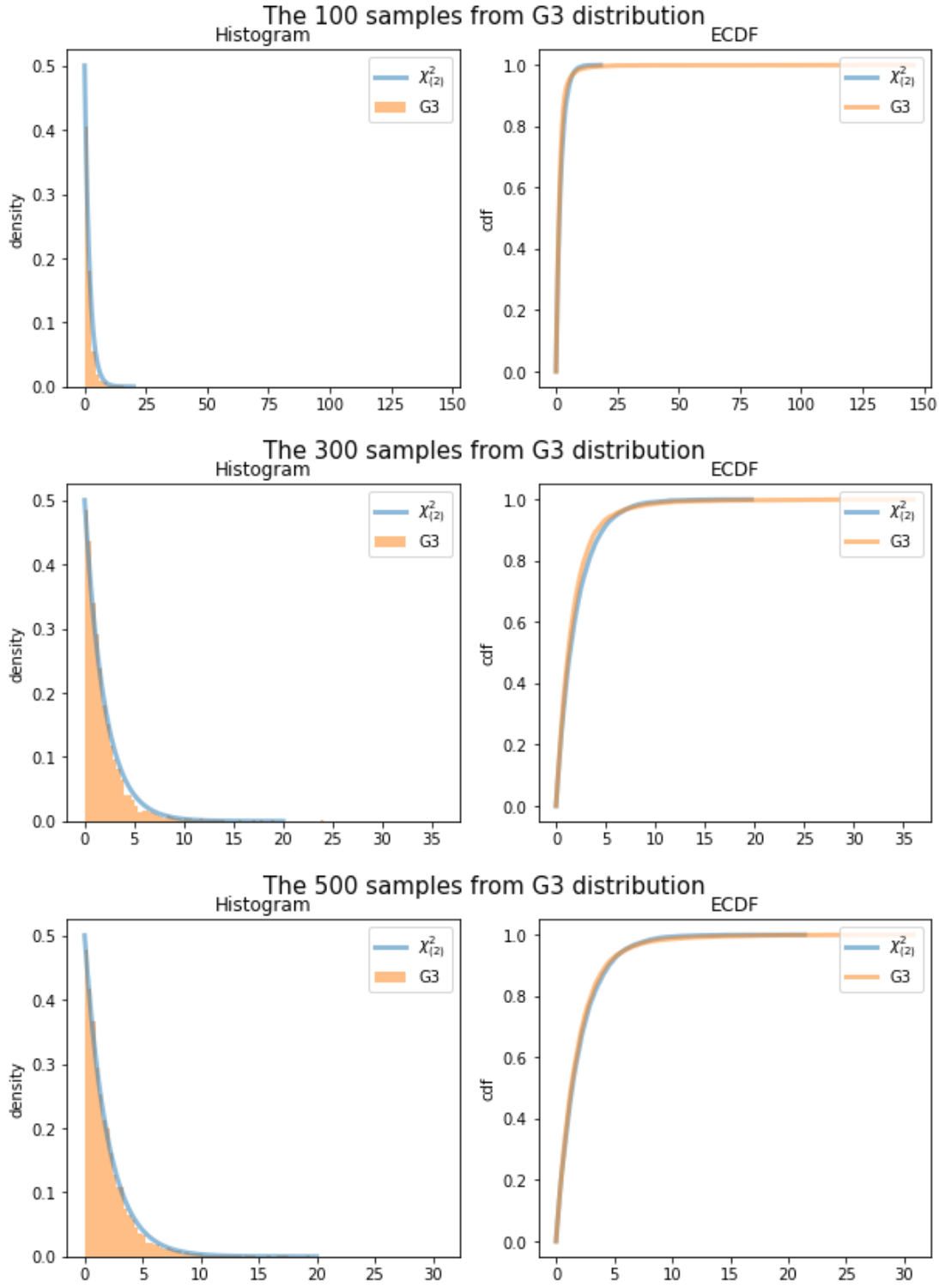
Out[ ]:
```
array([3.14708045, 6.93742896, 1.2972594 , ..., 2.22085916, 0.52462877,
       1.79698849])
```


The 10 samples from G3 distribution
Histogram
ECDF
density
cdf
$\chi^2_{(2)}$
G3


The 20 samples from G3 distribution
Histogram
ECDF
density
cdf
$\chi^2_{(2)}$
G3


The 30 samples from G3 distribution
Histogram
ECDF
density
cdf
$\chi^2_{(2)}$
G3


The 50 samples from G3 distribution
Histogram
ECDF
density
cdf
$\chi^2_{(2)}$
G3



將 **G3** 拉大到樣本數 **10000**

這時的 **qqplot** 和 **ecdf** 點較貼齊理論值的線 支持我們認為的卡方自由度二結果。

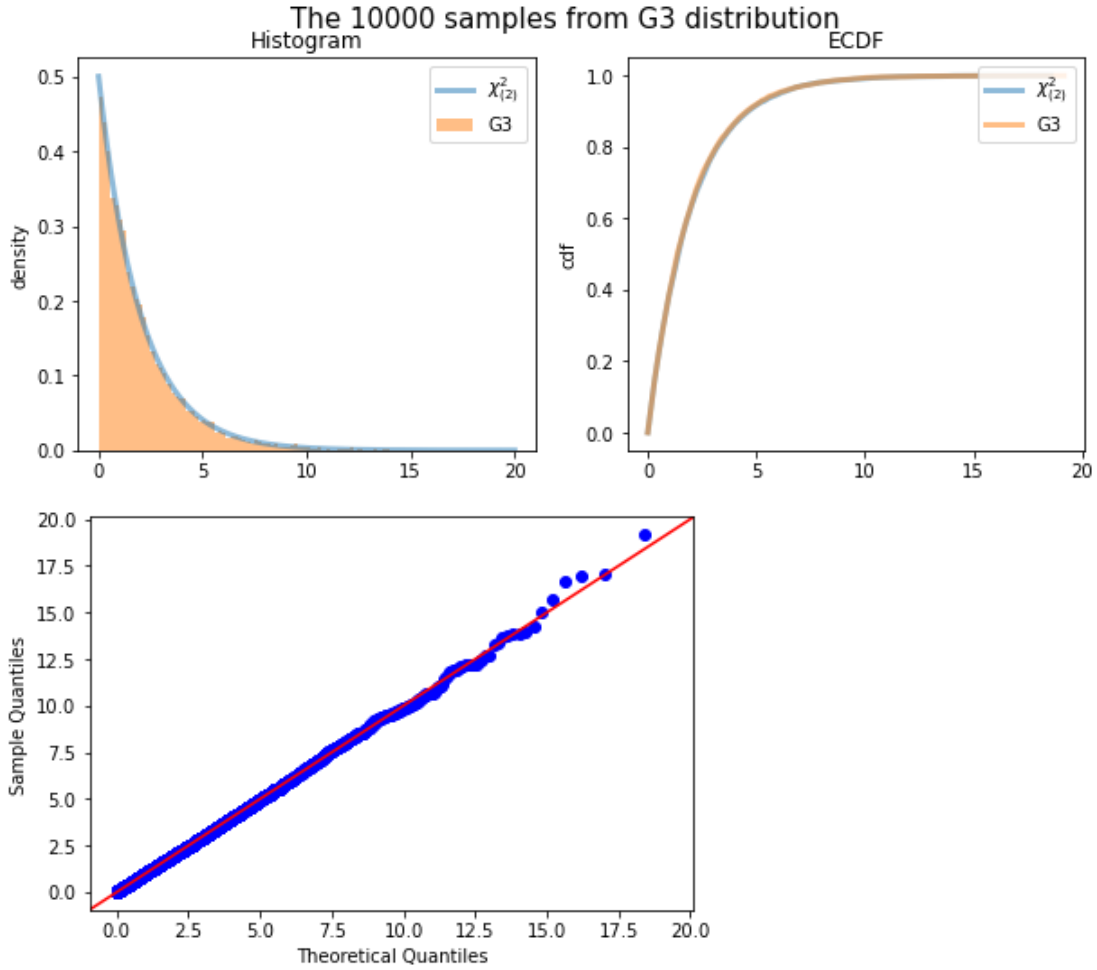
KS 檢定 **p-value** 大於 **0.05** 顯著 有足夠證據支持卡方自由度二。

因此我們認定 **G3** 近似卡方自由度為二的分配

In [ ]:

```
g310000 = g3stat(10000)
sm.qqplot(g310000, dist = chi2, distargs = (2, ), line = '45')
s, pvalue = kstest(g310000, 'chi2', args = (2, ))
pvalue = ">= 0.05" if pvalue > 0.05 else "< 0.05"
t = "Kolmogorov test : statistics = {}, p-value {}".format(round(s, 5), pvalue)
print(t)
```

```
Kolmogorov test : statistics = 0.01075, p-value >= 0.05
```



Part.3 Jarque_Beta test

上述我們得知，**G3** 統計量服從卡方自由度為 **2** 的分配

底下會用幾個分布來探究 **power** 藉以檢驗這個檢定是否有效

建立如此的常態假設

$$H0：\text{此分配為常態} \quad Ha：\text{此分配不為常態}$$

則 **power** 可定義為:在分配不為常態時 有多少機率可拒絕虛無假設

以下呈現的是 各分配不同大小樣本模擬出的 **power** 值 利用的是 **power** 可有多少證據告訴我們此分布不為常態

這裡每個抽取試驗模擬 **m = 50000** 遍，其目的是要呈現分布的長相，而 **n** 是我們想要討論的樣本大小介於**[10, 20, 30, 50, 100, 300, 500]**。

下方是 **power** 和不同分配及樣本大小的 **dataframe**

In [ ]:

```
# test function
def JBtest(x):
    g2 = lambda x: np.sqrt(len(x)/ 24) * (kurtosis(x, fisher = False) - 3)
    g1 = lambda x: np.sqrt(len(x)/ 6) * skew(x)
    g3 = lambda x: g2(x) ** 2 + g1(x) ** 2
    stat = g3(x)
    p_value = 1 - chi2.cdf(stat, df = 2)
    return stat, p_value

#  select distribution
def randomdist(n, dist, arg):
    if dist == "normal":
        return np.random.normal(size = n, loc = arg[0], scale = arg[1])
```

```
    if dist == "t":
        return np.random.standard_t(size = n, df = arg[0])
    if dist == "uniform":
        return np.random.uniform(size = n, low=arg[0], high= arg[1])
    if dist == "chisquare":
        return np.random.chisquare(size = n, df = arg[0])

# calculate power
def calpower(JBdata):
    crivalue = chi2.ppf(0.95, df = 2)
    rej = np.sum(JBdata > crivalue)
    return(rej/len(JBdata))

# power for dif. dist.
powerarr = np.empty(42)
i = 0
for j,k in zip(["normal", "t", "t", "t", "uniform", "chisquare"], [[0, 1], [3], [10], [30], [0, 1], [
    for n in [10, 20, 30, 50, 100, 300, 500]:
        distdata = randomdist(n * 50000, j,k).reshape(n, 50000)
        stat, _ = JBtest(distdata)
        powerarr[i] = calpower(stat)
        i = i + 1

powerdf = pd.DataFrame(powerarr.reshape(6, 7).T)
powerdf.columns = ["normal(0,1)", "t(3)", "t(10)", "t(30)", "uniform(0,1)", "chisquare(8)"]
powerdf["n"] = [10, 20, 30, 50, 100, 300, 500]
print(powerdf.set_index('n'))
```

```
     normal(0,1)     t(3)    t(10)    t(30)  uniform(0,1)  chisquare(8)
n
10       0.00836  0.10136  0.02088  0.01144       0.00216       0.04008
20       0.02520  0.31114  0.07566  0.03900       0.00044       0.16574
30       0.02944  0.46402  0.11724  0.05150       0.00024       0.27762
50       0.03678  0.66604  0.17642  0.07010       0.00016       0.49268
100      0.04268  0.89744  0.28760  0.09380       0.56908       0.86460
300      0.04782  0.99940  0.56606  0.15262       1.00000       0.99998
500      0.04718  1.00000  0.74302  0.19868       1.00000       1.00000
```

藉由圖形 可更清楚看到趨勢

從剛剛的模擬中可知 會希望樣本大一點使分配更像卡方 才具有我們想討論的檢定

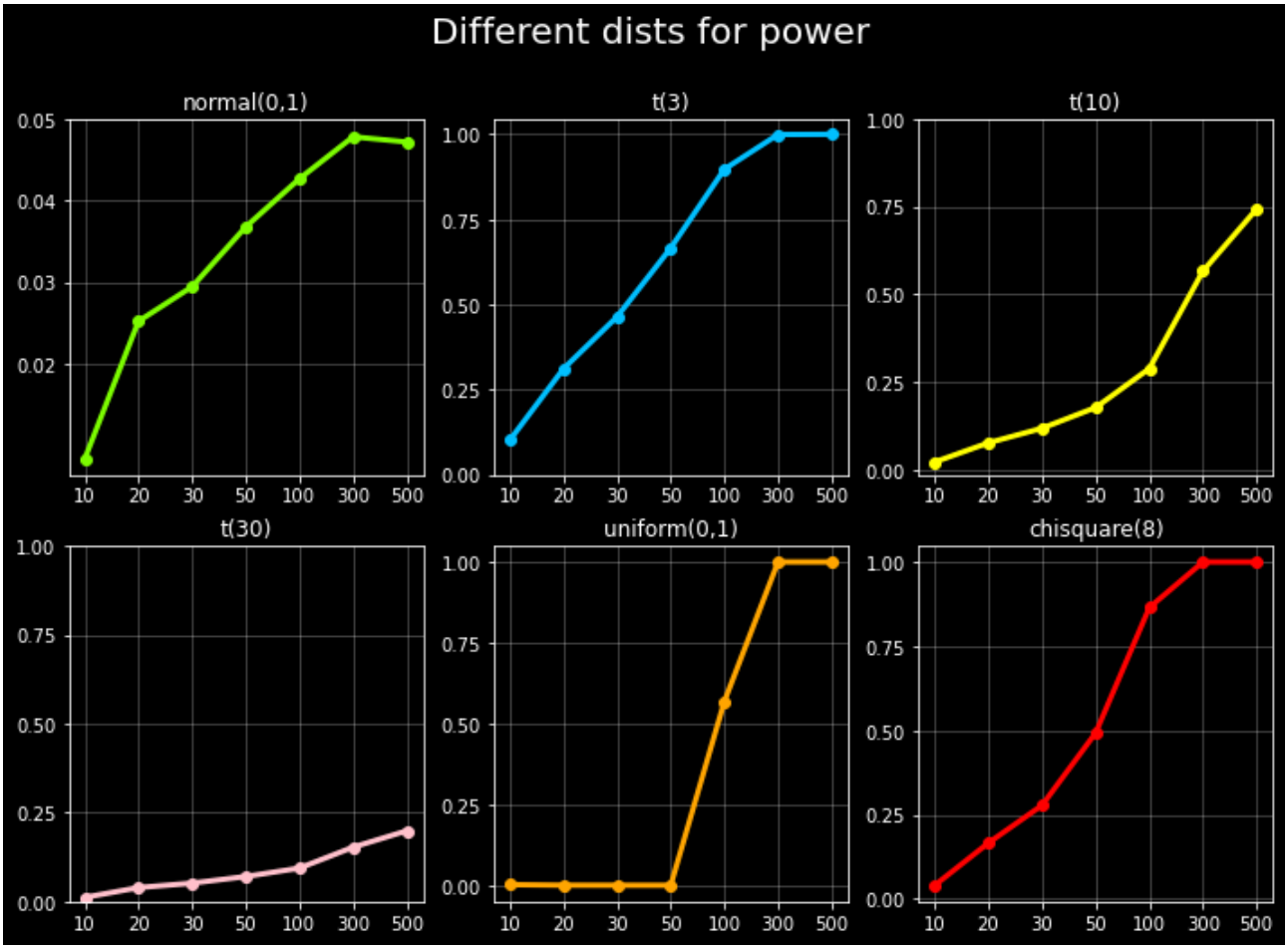
對應到的是下方的圖 是常態的分配因樣本數變大 **power** 趨近 **0.5** 不是常態的因樣本變大 **power** 趨近一

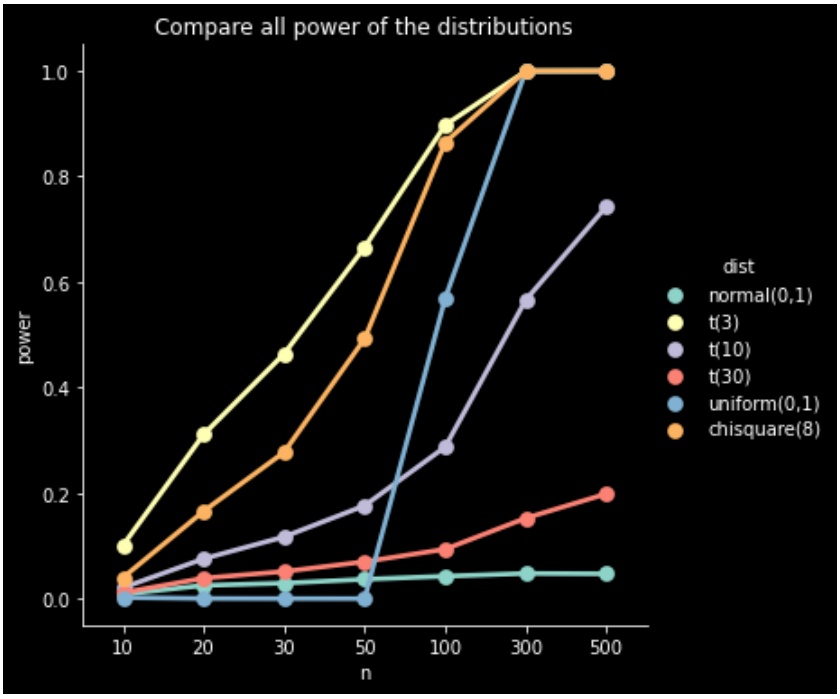
t 分配的自由度增加時 **power** 在同一個樣本數 **n** 下時減少，可印證為 **t** 分配在自由度愈大時很愈像常態。

In [ ]:

```
plt.style.use("dark_background")

# setting
colnames = ["normal(0,1)", "t(3)", "t(10)", "t(30)", "uniform(0,1)", "chisquare(8)"]
colors = ["lawngreen", "deepskyblue", "yellow", "pink", "orange", "red"]

# start plotting lines
fig, ax = plt.subplots(2, 3, figsize = (12, 8))
k = 0
for i in range(2):
    for j in range(3):
        ax[i, j].plot(powerdf.n.astype('str'), powerdf[colnames[k]], color = colors[k], lw= 3, marker
        ax[i, j].set_title(colnames[k])
        ax[i, j].grid(True, alpha = 0.3)
        if i+j == 0:
            ax[i, j].set_yticks( [0.02, 0.03, 0.04, 0.05])
        else:
            ax[i, j].set_yticks( [0, 0.25, 0.5, 0.75, 1])
        k = k+1
plt.suptitle('Different dists for power',fontsize =20)
```

```
plt.show()
sns.set_style({'axes.grid' : False})

# combine all lines
dfmelt = powerdf.melt("n", var_name='dist', value_name='vals')
p = sns.catplot(x="n", y="vals", hue='dist', data=dfmelt, kind='point')
p.set(xlabel = "n", ylabel = "power", title = "Compare all power of the distributions")
plt.show()
```





Part 4. Compare different test for normality

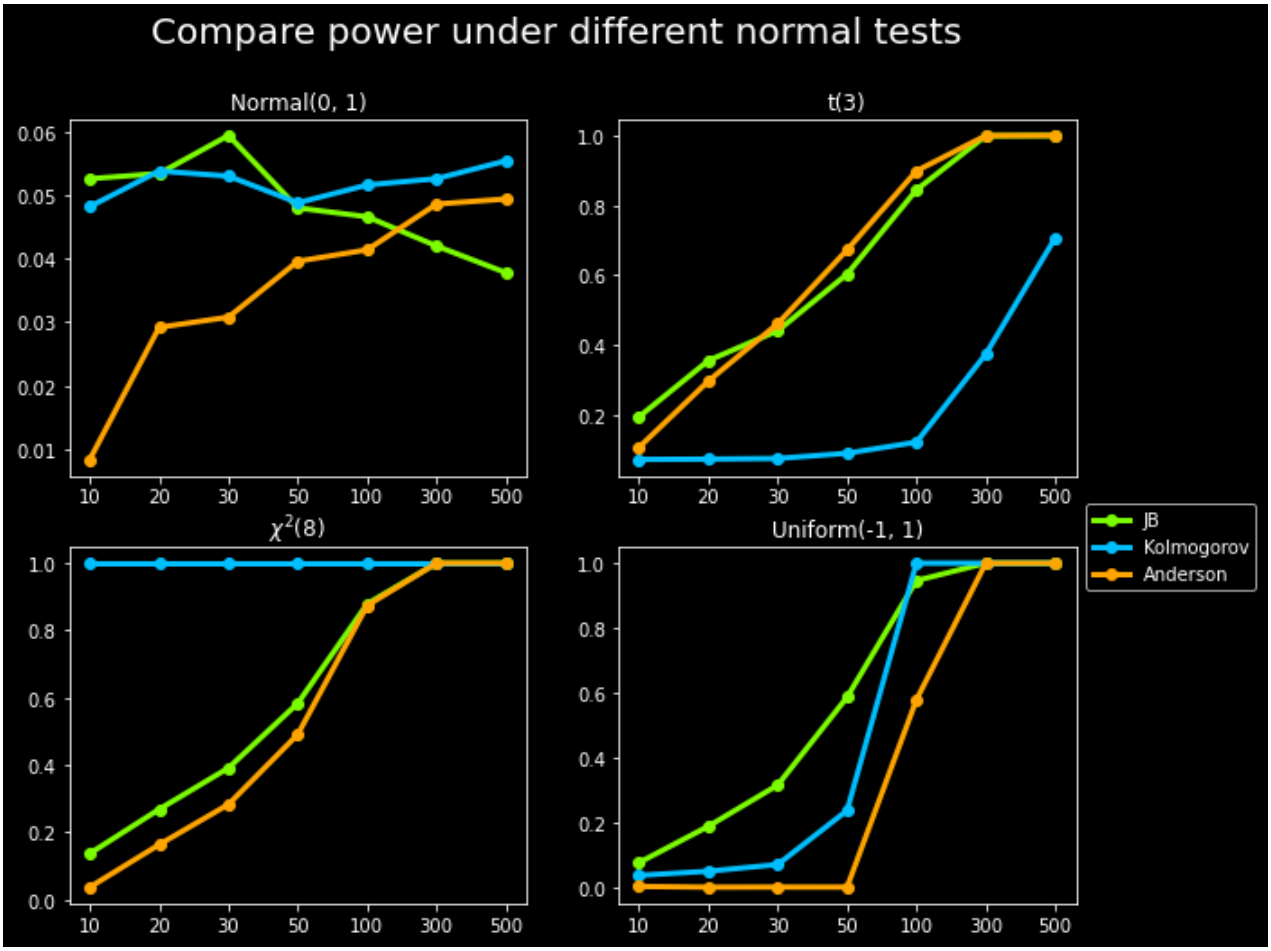
這裡看看不同的常態檢定 **Kolmogorow** 和 **Anderson darling**

此外， **python** 上也有 **Jarque_beta** 的程式

用這三個套件來比較不同樣本大小對於 **power** 的影響

In [ ]:
```
def normaltest(func, dist, args):
    n = [10, 20, 30, 50, 100, 300, 500]
    m = 5000
    power = np.empty(len(n))
    for i in range(len(n)):
        count = 0
        if func == 'anderson':
            for j in range(m):
                ns = randomdist(n[i], dist,  args)
                stat, cv, _= anderson(ns)
                if stat > cv[2]:
                    count = count + 1
        if func == 'kolmogorov':
            for j in range(m):
                ns = randomdist(n[i], dist,  args)
                _, pv= kstest(ns, 'norm')
                if pv < 0.05:
                    count = count + 1
        if func == 'jarque_beta':
            for j in range(m):
                ns = randomdist(n[i], dist,  args)
                _, pv= jarque_bera(ns)
                if pv < 0.05:
                    count = count + 1
        power[i] = count/m
    return(power)
```

In [ ]:
```
fig, ax = plt.subplots(2, 2, figsize= (10, 8))
n = np.array([10, 20, 30, 50, 100, 300, 500])
titles = ["Normal(0, 1)", "t(3)", r'$\chi^2(8)$', 'Uniform(-1, 1)']
dist = ["normal", "t", 'chisquare', 'uniform']
param = [[0, 1], [3], [8], [-1, 1]]
k = 0
for i in range(2):
    for j in range(2):
        ax[i, j].plot(n.astype(str), normaltest('anderson', dist[k], param[k]), label = "JB", lw = 3,
        ax[i, j].plot(n.astype(str), normaltest('kolmogorov', dist[k], param[k]), label = "Kolmogorov
        ax[i, j].plot(n.astype(str), normaltest('jarque_beta', dist[k], param[k]), label = "Anderson"
        ax[i, j].set_title(titles[k])
        k = k + 1
plt.legend(loc='center left', bbox_to_anchor=(1, 1))
plt.suptitle("Compare power under different normal tests", fontsize = 20)
plt.style.use('default')
```



討論

1. normal 的 power 在不同檢定 固定在 0.05 之間遊蕩 代表這三個檢定方法對於常態是有檢定的效果存在的。
2. 其他不是常態的分配 power會愈來愈接近一
3. 從剛剛對於 G3 要樣本數大才進似卡分自由度二的分配 發現 JBtest 適用於大一點的樣本數。
4. 卡方自由度八的 kolmogorov test power 都保持在一 可去探討是否kolmogorov test 對於不對稱分配會格外敏感。
5. 安德森檢定在各分配的表現相比其他檢定來的慢 ，可能是套件有些瑕疵，可去探討。

Part 5. 三門問題

在一個電視節目中，有提供三道門，供玩家選擇，其中一道門會有一台車，其他兩道後面是山羊，選中車即中獎

一開始玩家會先選定一道門，然後主持人會從其他兩道門中開出其中一個沒有獎品的門

接著玩家可選擇是否要換成另一個沒開的門，換了會比較高嗎?

根據直覺，兩道門中其中之一有車子，那換跟不換獲獎機率都應該是 1/2。

我們以蒙地卡羅模擬玩家重複玩了很多次的結果，再以比率來近似機率

In [ ]:

```
# replicate n games
n = 10000

# If we change, ...
change_hit = 0
for i in range(n):
    # set award
```

```
    award = np.random.choice([1, 2, 3], 1)
    # set first choice
    choose = np.random.choice([1, 2, 3], 1)
    # open door of non-award
    open = np.random.choice(list(set([1, 2, 3]) - set(award) - set(choose)), 1)
    # change and hit
    if list(set([1, 2, 3]) - set(choose) - set(open)) == [award]:
        change_hit = change_hit + 1

# result
print('模擬 10000 次 換門有車的結果為 ' , change_hit , ' 次' +'\n'+"比率為", change_hit/n)
```

```
模擬 10000 次 換門有車的結果為  6705  次
比率為 0.6705
```

事實上,並不是我們直覺的 **1/2**

我們將換門情形拆開看

假如一開始選到車，則換門一定會失敗。

假如一開始選到羊，則換門一定會成功。

如此可得一開始選到羊的機率就是換門成功的機率也就是 **2/3**，程式也印照了我們模擬的結果

Part.6 結語

蒙地卡羅處理數統的分布逼近相當直觀，可印證數學上抽象的數字或觀念

在這裡不僅實現了 **JB** 統計量的構思想法

且利用多次模擬去驗證檢定是否有實際的檢定力存在

除了數學理論，也可利用在生活的一些例子中

像最後提到的三門問題的應用 容易去模擬那些情況 進而推得機率

其廣泛的應用，令人嘆為觀止。