In [1]:
```
import torch
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from scipy.stats import chi2
from scipy.stats import chisquare
from scipy.stats import binom
from scipy.stats import beta
from scipy.stats import gamma
from scipy.stats import norm
from scipy.stats import invgamma


import statistics as stat

%matplotlib inline

mpl.rcParams['pdf.fonttype'] = 42
mpl.rcParams['ps.fonttype'] = 42
fig_dpi       = 300
fig_typeface = 'Helvetica'
fig_family    = 'monospace'
fig_style     = 'normal'
```

1.a

Sample X, Y from N(0,1).

In [2]:
```
class Drunkman(object):
    def __init__(self, start_point:list = [0,0], steps:int = 50):
        self.start_point = start_point
        self.steps = steps
    def __repr__(self):
        return "I am a drunk man."
    def walk(self, start_point = [0,0]):
        self.start_point = start_point
        oneTrack = [self.start_point]
        for i in range(self.steps):
            x = np.random.normal(0,1)
            y = np.random.normal(0,1)
            self.start_point = [self.start_point[0] + x,  self.start_point[1
            oneTrack.append(self.start_point)
        return oneTrack
    def getLocation(self):
        # Use one hot to show the final position
        location = np.array([0,0,0,0], dtype = "float64")
        oneTrack = self.walk()
        if oneTrack[-1][0] >= 0:
            if oneTrack[-1][1] > 0:
                location[0] += 1
        if oneTrack[-1][0] <= 0:
            if oneTrack[-1][1] < 0:
                location[2] += 1
        if oneTrack[-1][0] > 0:
            if oneTrack[-1][1] <= 0:
                location[3] += 1
        if oneTrack[-1][0] < 0:
            if oneTrack[-1][1] >= 0:
```

```
                location[1] += 1
        return location

    def oneTrack_plot(self):
        oneTrack  = self.walk()
        track = np.array(oneTrack)
        X = track[:, 0]
        Y = track[:, 1]
        x_range = [-np.max(np.abs(X)), np.max(np.abs(X))]
        y_range = [-np.max(np.abs(Y)), np.max(np.abs(Y))]
        f, ax = plt.subplots(1, 1, figsize=(3, 3), facecolor='white', dpi=30
        ax.plot(0, 0, ".",3, c = "blue", zorder = 1)
        ax.plot(X[-1], Y[-1], ".",3, c = "darkgreen", zorder = 1)
        ax.plot(X, Y, "-",1., c = "darkred", zorder = 0)
        ax.tick_params(axis='both', which='both', labelsize='xx-small', righ

        ax.vlines(0,y_range[0], y_range[1], ls = '--', color = 'black',lw =
        ax.hlines(0,x_range[0], x_range[1], ls = '--', color = 'black',lw =

        ax.set_xlabel("X", size='x-small')
        ax.set_ylabel("Y", size='x-small')
        ax.set_xlim(x_range)
        ax.set_ylim(y_range)
#         ax.legend(loc = 1 ,fontsize = 3,markerscale = 2,ncol = 3,scatterpo
        plt.show()

    def Trials(self, manNum : int = 10000):
        initial = np.array([0,0,0,0], dtype = "float64")
        for _ in range(manNum):
            finalLocation = self.getLocation()
            initial += finalLocation
        return initial
```

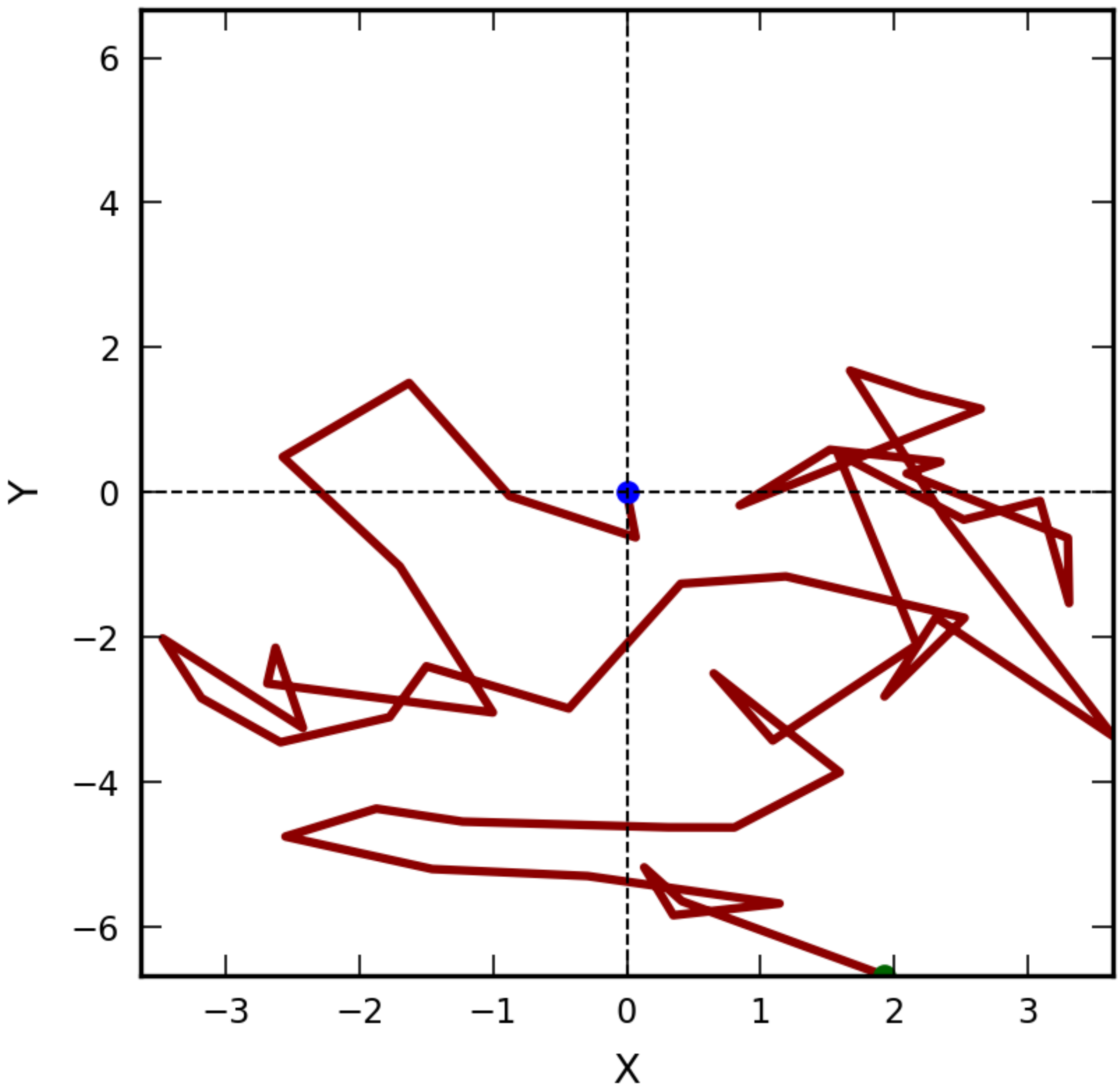
In [3]:

```
Walk_man = Drunkman()
```

1.a.(1) Plot 1 random walk. of 50. steps

In [4]:

```
Walk_man.oneTrack_plot()
```



1.a.(2) Report the number that ends in each quadrant

In [5]:
```
num_counts = Walk_man.Trials()
num_counts
```

Out[5]:
```
array([2557., 2398., 2509., 2536.])
```

1.a.(3) What values would you expect for these proportions? Do the observed proportions vary significantly from the expected values?

I think $p_1 = p_2 = p_3 = p_4 = \frac{1}{4}$.

And I plan to do Chi-square test using the results showed above.

$H_0 : p_1 = p_2 = p_3 = p_4 = \frac{1}{4}$.

H_a : It's not the case and they are not equal to each other. Then the test Statistic $\sum_{i=1}^{i=4} \frac{\frac{f_i}{N} - \frac{1}{4}}{1/4}$ follows $\chi^2(3)$

In [6]:
```
H_null = np.array([1/4, 1/4, 1/4, 1/4])
chisq, p = chisquare(num_counts/np.sum(num_counts), f_exp=H_null, ddof=0)
print(f"The p-value is {p:4.5f}, so we accept the null hypothesis.")
```

```
The p-value is 1.00000, so we accept the null hypothesis.
```

1.b (steps goes to 500)

In [7]:
```
Walk_man500 = Drunkman(steps = 500)
```

1.b.(1) Report the number that ends in each quadrant

In [8]:
```
num_counts500 = Walk_man500.Trials()
num_counts500
```

Out[8]:
```
array([2543., 2461., 2459., 2537.])
```

1.b.(3) What values would you expect for these proportions? Do the observed proportions vary significantly from the expected values?

I think $p_1 = p_2 = p_3 = p_4 = \frac{1}{4}$.
And I plan to do Chi-square test using the results showed above.
$H_0 : p_1 = p_2 = p_3 = p_4 = \frac{1}{4}$.
H_a : It's not the case and they are not equal to each other. Then the test Statistic
$\sum_{i=1}^{i=4} \frac{\frac{f_i}{N} - \frac{1}{4}}{1/4}$ follows $\chi^2(3)$

In [9]:
```
H_null = np.array([1/4, 1/4, 1/4, 1/4])
chisq, p = chisquare(num_counts500/np.sum(num_counts500), f_exp=H_null, ddof
print(f"The p-value is {p:4.5f}, so we accept the null hypothesis.")
```

```
The p-value is 1.00000, so we accept the null hypothesis.
```

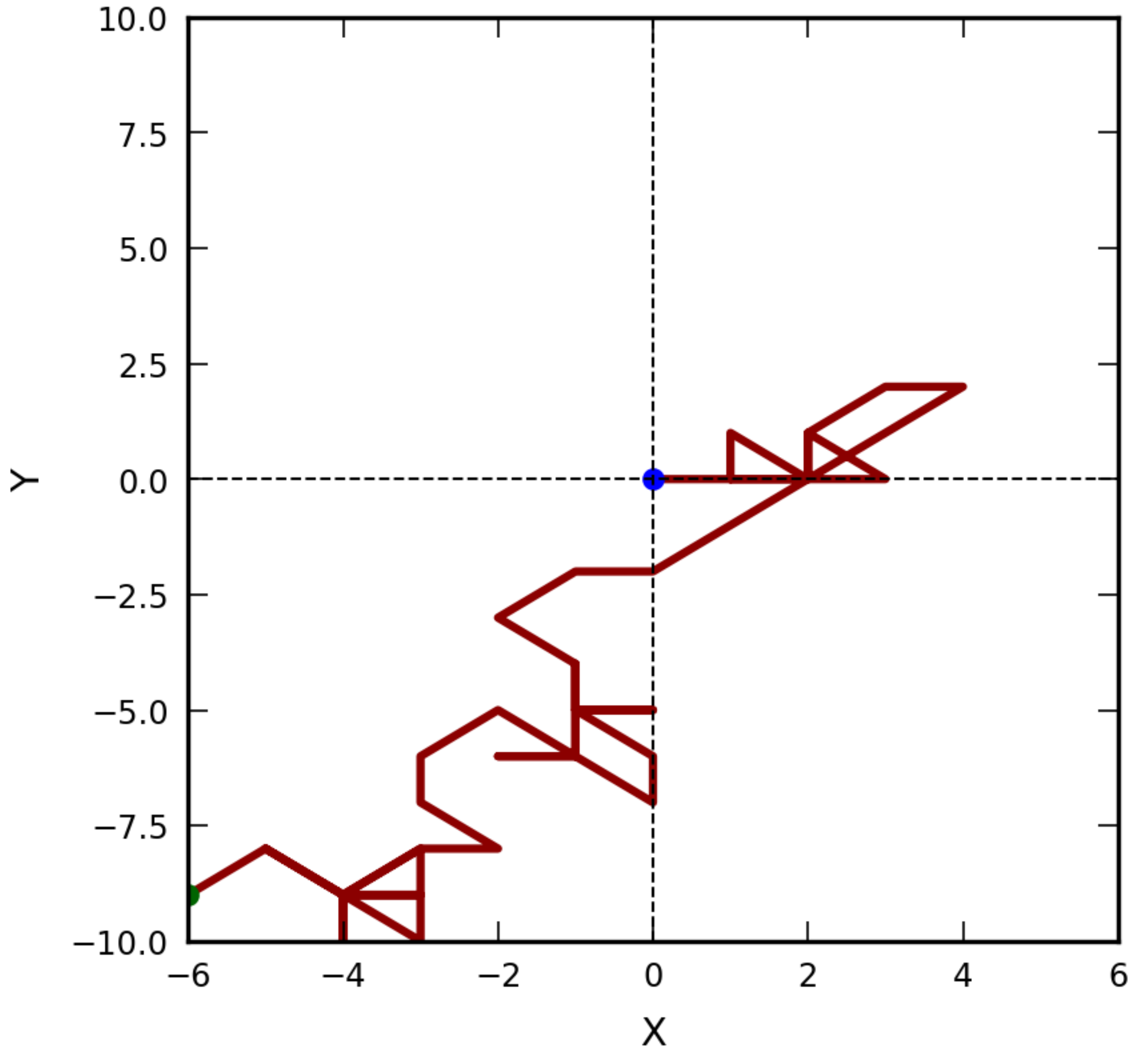
1.Repeat 1a and 1b using random walks on the lattice in R^2

In [10]:
```
class DrunkmanOnLattice1(Drunkman):
    def __init__(self, start_point:list = [0,0], steps:int = 50):
        super(Drunkman,self).__init__()
        self.steps = steps
        self.start_point = start_point

    def walk(self, start_point = [0,0]):
        step_choice = [[1,0], [1,1],[0,1],[-1,1], [-1,0], [-1,-1],[0,-1],[1,
        self.start_point = start_point
        oneTrack = [self.start_point]
        for i in range(self.steps):
            step = step_choice[np.random.randint(0,8)]
            self.start_point = [self.start_point[0] + step[0],  self.start_p
            oneTrack.append(self.start_point)
        return oneTrack
```

In [11]:
```
class DrunkmanOnLattice2(Drunkman):
    def __init__(self, start_point:list = [0,0], steps:int = 50):
        super(Drunkman,self).__init__()
        self.steps = steps
        self.start_point = start_point

    def walk(self, start_point = [0,0]):
```

```
        step_choice = [[1,0],[0,1], [-1,0],[0,-1]]
        self.start_point = start_point
        oneTrack = [self.start_point]
        for i in range(self.steps):
            step = step_choice[np.random.randint(0,4)]
            self.start_point = [self.start_point[0] + step[0],  self.start_p
            oneTrack.append(self.start_point)
        return oneTrack
```

In [12]:
```
# He can walk along the diagonals.
Walk_manL1 = DrunkmanOnLattice1()
Walk_manL1.oneTrack_plot()
numL_counts = Walk_manL1.Trials()
numL_counts
```



Out[12]:
```
array([2519., 2525., 2399., 2511.])
```

I think $p_1 = p_2 = p_3 = p_4 = \frac{1}{4}$.

And I plan to do Chi-square test using the results showed above.

$H_0 : p_1 = p_2 = p_3 = p_4 = \frac{1}{4}$.

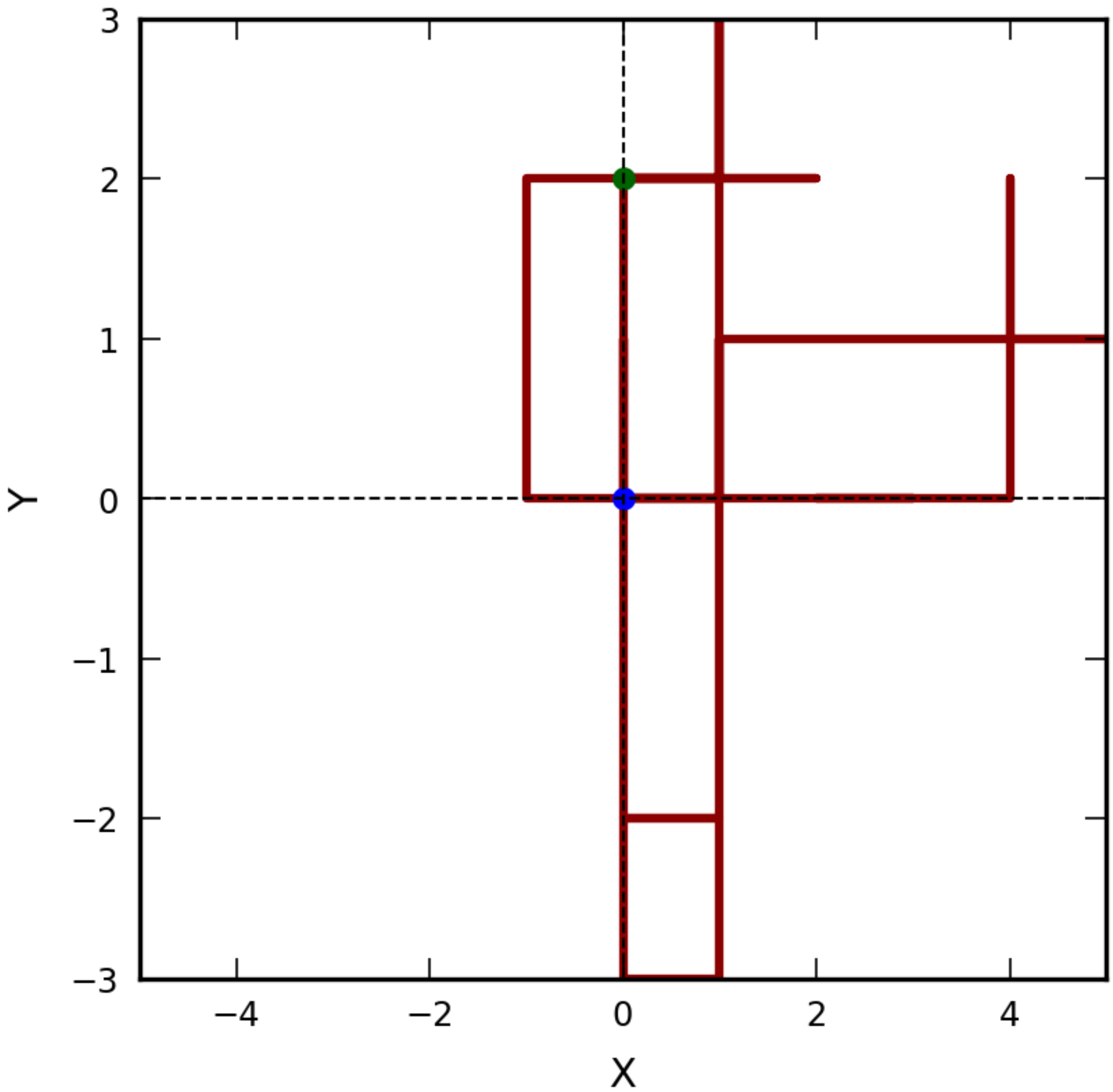
H_a : It's not the case and they are not equal to each other. Then the test Statistic $\sum_{i=1}^{i=4} \frac{\frac{f_i}{N} - \frac{1}{4}}{1/4}$ follows $\chi^2(3)$

In [13]:
```
H_null = np.array([1/4, 1/4, 1/4, 1/4])
chisq, p = chisquare(num_counts500/np.sum(num_counts500), f_exp=H_null, ddof
```

```
print(f"The p-value is {p:4.5f}, so we accept the null hypothesis.")
```

The p-value is 1.00000, so we accept the null hypothesis.

In [16]:
```
# He can just walk along the grids.
Walk_manL2 = DrunkmanOnLattice2()
Walk_manL2.oneTrack_plot()
numL_counts = Walk_manL2.Trials()
numL_counts
```



Out[16]: array([2436., 2475., 2491., 2474.])

In [17]:
```
H_null = np.array([1/4, 1/4, 1/4, 1/4])
chisq, p = chisquare(num_counts500/np.sum(num_counts500), f_exp=H_null, ddof
print(f"The p-value is {p:4.5f}, so we accept the null hypothesis.")
```

The p-value is 1.00000, so we accept the null hypothesis.

As for the 500 steps, it is the same but with the initial DrunkmanOnLattice(500). They have the same conclusion for the χ^2 test

2. Sample from Binomial(14,9); Beta(5,5); Gamma(12,2); IG(12,2); $\chi^2(2)$; $\chi^{-2}(2)$

In [3]:
```
pointNums = [1e2,1e4, 1e5, 1e6, 1e7]
```

In [30]:
```
table1E4 = pd.DataFrame()
Distributions = ["Binomial", "Beta", "Gamma","InverseGamma", "ChiSquare","In
for pointNum in pointNums:
    BinPoints = np.random.binomial(14, .9, int(pointNum))
    BetaPoints = np.random.beta(.5, .5, int(pointNum))
    GammaPoints = np.random.gamma(12, 1/2, int(pointNum)) # Here I use 1/bet
    IGPoints = 1/np.random.gamma(12, 1/2, int(pointNum)) # 1/X ~ Gamma(12,2)
    ChiPoints = np.random.chisquare(2, int(pointNum))
    IChiPoints = 1/np.random.chisquare(2, int(pointNum))# 1/X ~ ChiSq(2); th
#     IChiPoints = 1/np.random.gamma(1,1/2, int(pointNum))
    table = pd.DataFrame({"Binomial": BinPoints, "Beta":BetaPoints, "Gamma":
                         "InverseGamma": IGPoints, "ChiSquare":ChiPoints, "I
    exec("Table%s = table"%str(int(pointNum)))
    print("Table%s created"%str(int(pointNum)))
```

```
Table100 created
Table10000 created
Table100000 created
Table1000000 created
Table10000000 created
```

In [31]:
```
2/11/11/5
```

Out[31]:
```
0.003305785123966942
```

In [32]:
```
STDs = []
EPs = []
for pointNum in pointNums:
    exec("table = Table%s"%str(int(pointNum)))
    std = []
    ep = []
    for col in table.columns:
        std.append(np.std(table[col]))
        ep.append(np.mean(table[col]))
    STDs.append(std)
    EPs.append(ep)
STDs = np.array(STDs)
EPs = np.array(EPs)
```

In [33]:
```
STD_tabel = pd.DataFrame(STDs,columns = Distributions)
STD_tabel.index = pd.Series(["1e2","1e4", "1e5", "1e6", "1e7"])
EP_tabel = pd.DataFrame(EPs,columns = Distributions)
EP_tabel.index = pd.Series(["1e2","1e4", "1e5", "1e6", "1e7"])
```

In [34]:
```
STD_tabel
```

Out[34]:

	Binomial	Beta	Gamma	InverseGamma	ChiSquare	InverseChiSquare
1e2	1.068597	0.347124	1.685416	0.063718	1.786718	16.519434
1e4	1.115702	0.354414	1.736821	0.057843	2.006398	241.289256
1e5	1.120014	0.354066	1.734220	0.057736	2.003261	284.759514
1e6	1.121812	0.353827	1.732433	0.057495	2.002551	1904.222821
1e7	1.122769	0.353615	1.732307	0.057521	2.000062	5270.254600

In [35]:

```
EP_tabel
```

Out[35]:

	Binomial	Beta	Gamma	InverseGamma	ChiSquare	InverseChiSquare
1e2	12.590000	0.463507	5.681483	0.186208	1.633425	8.113601
1e4	12.597000	0.495860	6.012049	0.182295	1.997652	18.958098
1e5	12.597200	0.501169	5.995845	0.181720	1.999858	18.603301
1e6	12.601645	0.499771	6.000658	0.181763	2.001047	24.873609
1e7	12.599799	0.500024	6.000174	0.181820	2.000061	29.114923

The true values of expectation and variance are shown below in the table according to the formulas. The simulation results approximate the real values

Distributions/Properties	Expectation	$\sqrt{Variance} = std$
Binomial(14, .9)	12.6	$\sqrt{1.26} = 1.122$
Beta(.5, .5)	0.5	$\sqrt{0.125} = 0.354$
Gamma(12, 2)	6	$\sqrt{0.125} = 1.732$
IG(12, 2)	0.1818	$\sqrt{0.003306} = 0.058$
$\chi^2(2)$	2	$\sqrt{4} = 2$
$\chi^{-2}(2)$	~	~

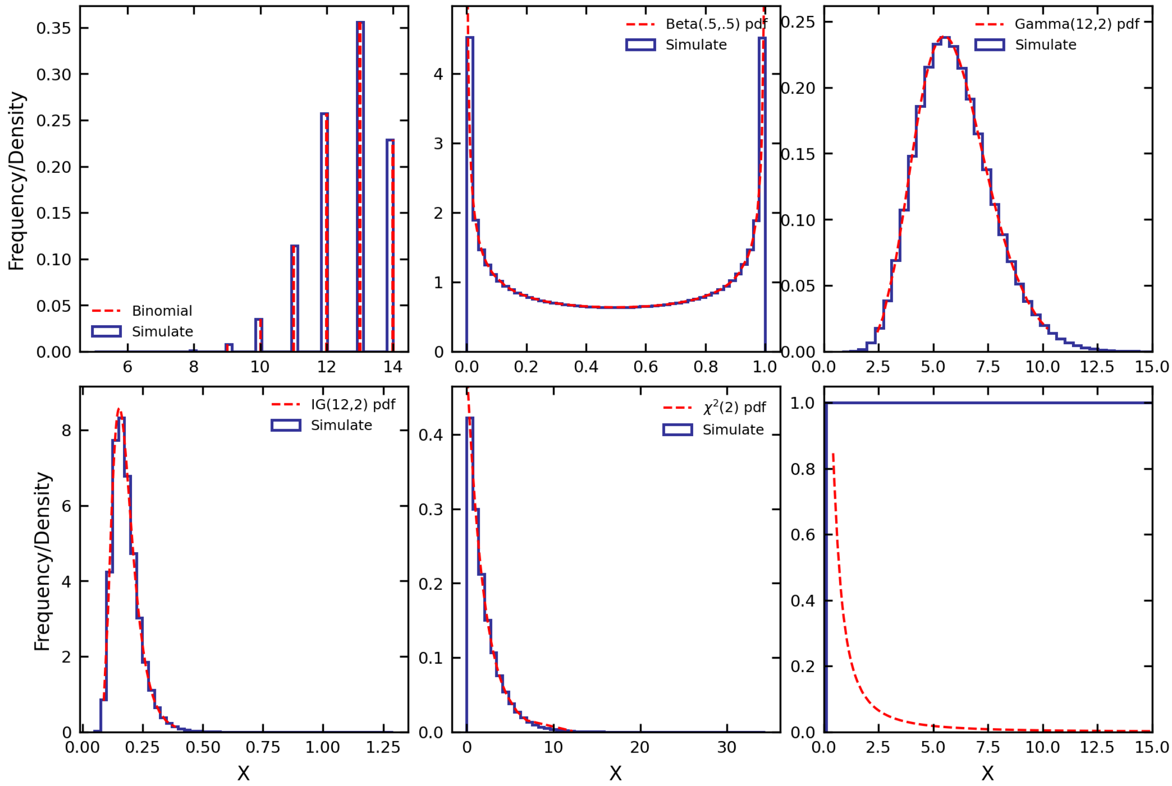
In [36]:

```
f,ax = plt.subplots(figsize=(6.5, 4.8), dpi=300)
gs1 = gridspec.GridSpec(2, 1)
gs1.update(left=0.01, right=0.31, bottom=0.05, top=0.95, hspace=0.1, wspace=
ax1 = plt.subplot(gs1[0])
ax4 = plt.subplot(gs1[1])


gs2 = gridspec.GridSpec(2, 1)
gs2.update(left=0.35, right=0.65, bottom=0.05, top=0.95, hspace=0.1, wspace=
ax2 = plt.subplot(gs2[0])
ax5= plt.subplot(gs2[1])


gs3 = gridspec.GridSpec(2, 1)
gs3.update(left=0.69, right=0.99, bottom=0.05, top=0.95, hspace=0.1, wspace=
```

```
ax3 = plt.subplot(gs3[0])
ax6 = plt.subplot(gs3[1])

table = Table10000000
# x1 = np.arange(binom.ppf(0.0, 14, .9),
#                binom.ppf(1.0, 14, .9))
for i in range(len(Distributions)):
    col = Distributions[i]
    data = np.array(Table100[col])
    data.sort()
    exec("x%d = data"%(i+1))
ax1.vlines(x1, 0, binom.pmf(x1, 14, .9), colors='red', linestyles='--', lw=1
ax2.plot(x2, beta.pdf(x2, .5, .5), 'r--', lw=1, label='Beta(.5,.5) pdf', zor
ax3.plot(x3, gamma.pdf(x3, 12,0, 1/2), 'r--', lw=1, label='Gamma(12,2) pdf',
ax4.plot(x4, invgamma.pdf(x4, 12,0, 2), 'r--', lw=1, label='IG(12,2) pdf', z
ax5.plot(x5, chi2.pdf(x5, 2), 'r--', lw=1, label=r'$\chi^2(2)$ pdf', zorder
ax6.plot(x6, 1/2*x6**(-2)*np.exp(-1/(2*x6)), 'r--', lw=1, label=r'$\chi^2(2)

for i in range(len(Distributions)):
    exec("ax = ax%d"%(i+1))
    col = Distributions[i]
    data = table[col]
    if i ==0:
        ax.hist(data, density= 0,
            weights=np.ones(len(data))/len(data), bins=50, histtype='step',l
        ax.legend(loc = 3 ,fontsize = 6,markerscale = 1,ncol = 1,scatterpoin
    elif i==5:
        ax.hist(data, density= 0,
                weights=np.ones(len(data))/len(data), bins=50, histtype='ste
    else:
        hist, bins =np.histogram(data, density = 1, bins =50)
        ax.hist(data, density= 1,
                weights=np.ones(len(data))/len(data), bins=50, histtype='ste
        ax.legend(loc = 1 ,fontsize = 6,markerscale = 1,ncol = 1,scatterpoin
        ax.set_ylim(0, 1.1*max(hist))
    ax.tick_params(axis='both', which='both',labelleft = True, labelsize='x-
ax3.set_xlim(0,15)
ax6.set_xlim(0,15)
ax1.legend(loc = 3 ,fontsize = 6,markerscale = 1,ncol = 1,scatterpoints= 1,f
ax4.set_xlabel('X', size='small'); ax5.set_xlabel('X', size='small');  ax6.s
ax1.set_ylabel('Frequency/Density', size = "small"); ax4.set_ylabel('Frequen
```

```
/var/folders/dy/y_4bw3nj3nl7cw3b482fcf_c0000gn/T/ipykernel_81596/1902577999.
py:4: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is depr
ecated since 3.6 and will be removed two minor releases later; explicitly ca
ll ax.remove() as needed.
  ax1 = plt.subplot(gs1[0])
```

Out[36]: `Text(0, 0.5, 'Frequency/Density')`



3a. Draw a sample of size 10000 from the trivariate normal distribution with mean (0,0,0) and variance-covariance matrxi:

$$\begin{bmatrix} 1 & 4.5 & 9.0 \\ 4.5 & 25 & 49 \\ 9 & 49 & 100 \end{bmatrix}$$

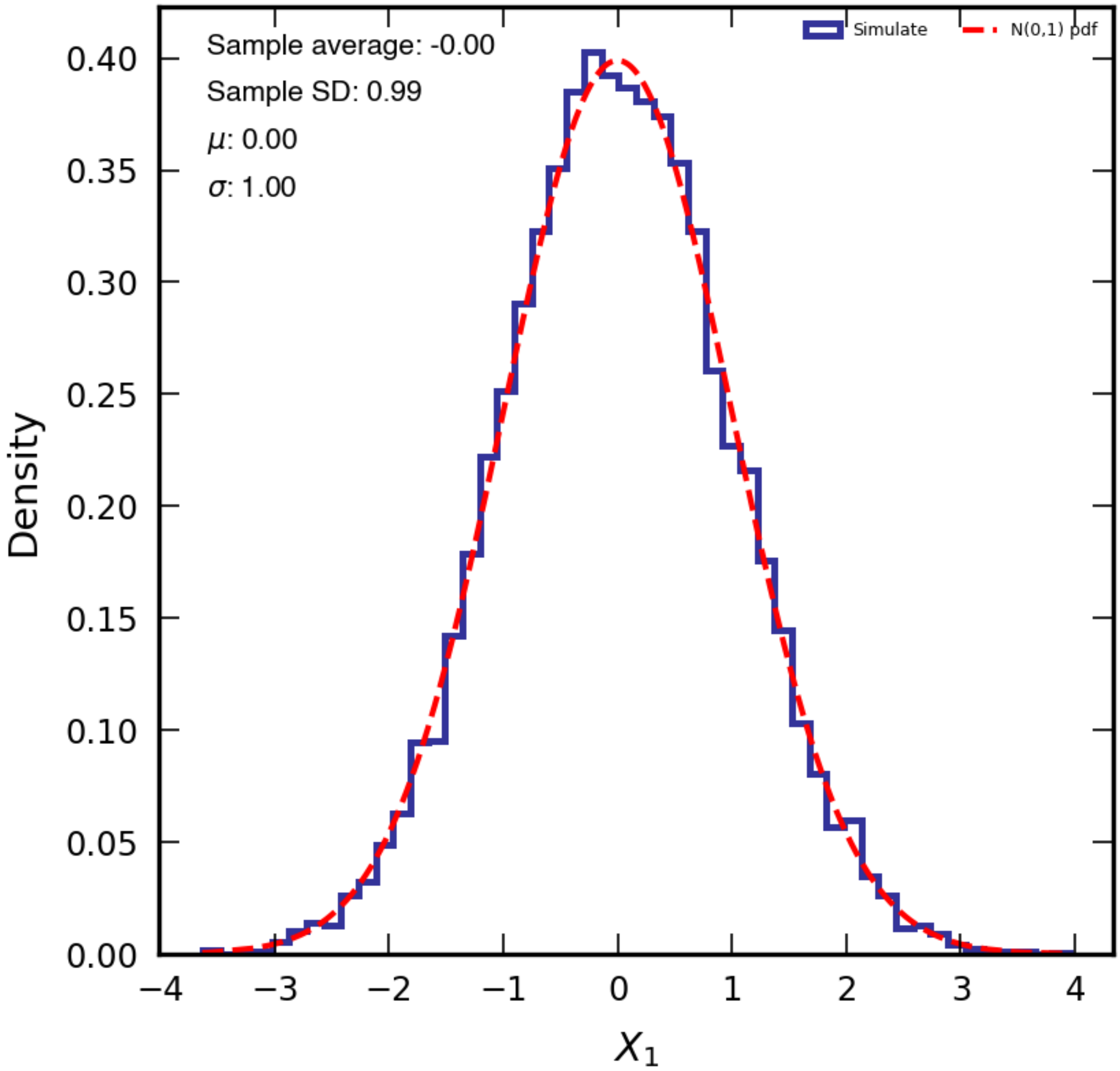
In [3]:
```
MEAN = np.array([0, 0, 0])
COV = np.array([[1, 4.5, 9.0],
                [4.5, 25, 49],
                [9, 49, 100]])
X = np.random.multivariate_normal(MEAN, COV, size=int(1e4), check_valid='war
X1 = X[:,0]
X2 = X[:,1]
X3 = X[:,2]
```

3b. Draw a histogram of the X_1 deviates, along with the true marginal. Compute the sample average and sample SD and compare these numbers to the true values.

X_1 ~ N(0,$\epsilon_{1,1}$) = N(0,1)

In [4]:
```
f, ax = plt.subplots(1, 1, figsize=(3, 3), facecolor='white', dpi=300, grids
X1.sort()
ax.hist(X1, density= 1, bins=50, histtype='step',lw= 1.2,color='navy', alpha
```

```
ax.plot(X1, norm.pdf(X1, 0, 1), 'r--', lw=1, label='N(0,1) pdf', zorder = 1)
ax.text(0.05, 0.95, "Sample average: %.2f"%(np.mean(X1)), size=5, weight = '
ax.text(0.05, 0.9, "Sample SD: %.2f"%(np.std(X1)), size=5, weight = 'bold',s
ax.text(0.05, 0.85, r"$\mu$: %.2f"%(0), size=5, weight = 'bold',style=fig_st
ax.text(0.05, 0.8, r"$\sigma$: %.2f"%(1), size=5, weight = 'bold',style=fig_
ax.tick_params(axis='both', which='both', labelsize='xx-small', right=True,
ax.set_xlabel(r"$X_1$", size='x-small')
ax.set_ylabel("Density", size='x-small')
#         ax.set_xlim(x_range)
#         ax.set_ylim(y_range)
ax.legend(loc = 1 ,fontsize = 3,markerscale = 2,ncol = 3,scatterpoints= 1,fr
plt.show()
```



3c. Draw a sample of size 10,000 from the conditional distribution p($X_1|X_2,\ X_3$), take $X_2 = X_3 = 1$. Compute the sample mean and sample SD and compare these numbers to the true values. Draw a histogram of the simulated values, along with the true conditional distribution.

The covaraince martix can be written as :

$$\begin{bmatrix} \sigma^2_{X1} & \rho_{X1,X2}\sigma_1\sigma_2 & \rho_{X1,X3}\sigma_1\sigma_3 \\ \rho_{X1,X2}\sigma_1\sigma_2 & \sigma^2_{X2} & \rho_{X2,X3}\sigma_2\sigma_3 \\ \rho_{X1,X3}\sigma_1\sigma_3 & \rho_{X2,X3}\sigma_2\sigma_3 & \sigma^2_{X3} \end{bmatrix} =$$

$$\begin{bmatrix} 1^2 & 0.9 \times 1 \times 5 & 0.9 \times 1 \times 10 \\ 0.9 \times 1 \times 5 & 5^2 & 0.98 \times 5 \times 10 \\ 0.9 \times 1 \times 10 & 0.98 \times 5 \times 10 & 10^2 \end{bmatrix}$$

E[$X_1|X_2 = x_2, X_3 = x_3$] = μ_1 + $\frac{\sigma 1(\sigma 3(x2-\mu 2)(\rho 23\rho 13-\rho 12)-\sigma 2(x3-\mu 3)(\rho 13-\rho 23\rho 12)}{(\rho_{23}^2-1)\sigma 2\sigma 3}$

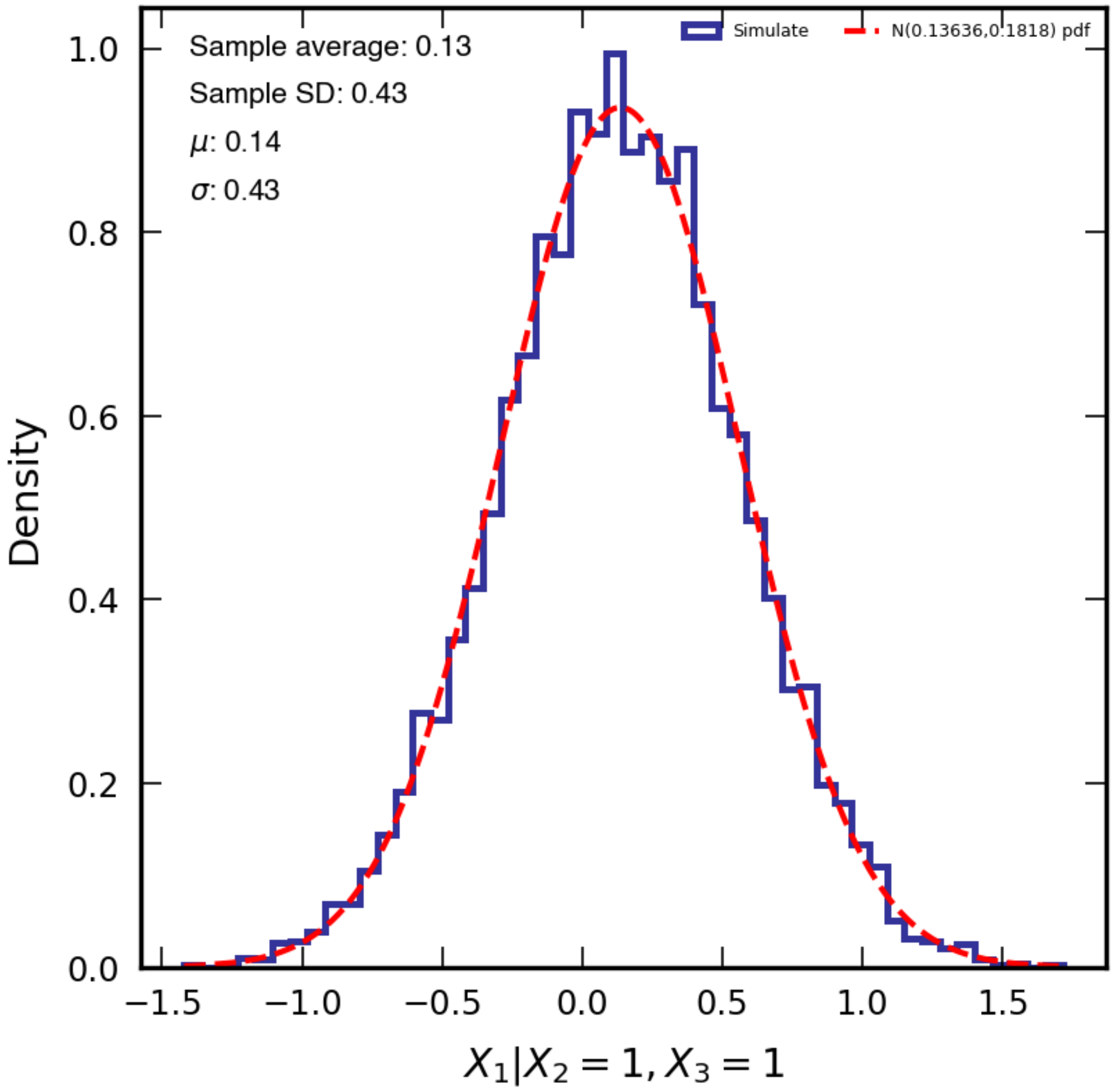
Var($X_1|X_2 = x_2, X_3 = x_3$) = $\frac{\sigma_1^2(\rho_{23}^2-2\rho 23\rho 13\rho 12+\rho_{13}^2+\rho_{12}^2-1)}{\rho_{23}^2-1}$

Hence, E[$X_1|X_2 = 1, X_3 = 1$] = 0.1363636363636362

Var($X_1|X_2 = 1, X_3 = 1$) = 0.18181818181818385

STD($X_1|X_2 = 1, X_3 = 1$) = 0.4263

In [5]:
```
f, ax = plt.subplots(1, 1, figsize=(3, 3), facecolor='white', dpi=300, grids
X1_11 = np.random.normal(0.13636, np.sqrt(0.1818), int(1e4))
X1_11.sort()
ax.hist(X1_11, density= 1, bins=50, histtype='step',lw= 1.2,color='navy', al
ax.plot(X1_11, norm.pdf(X1_11, 0.13636, np.sqrt(0.1818)), 'r--', lw=1, label
ax.text(0.05, 0.95, "Sample average: %.2f"%(np.mean(X1_11)), size=5, weight
ax.text(0.05, 0.9, "Sample SD: %.2f"%(np.std(X1_11)), size=5, weight = 'bold
ax.text(0.05, 0.85, r"$\mu$: %.2f"%(0.13636), size=5, weight = 'bold',style=
ax.text(0.05, 0.8, r"$\sigma$: %.2f"%(np.sqrt(0.1818)), size=5, weight = 'bo
ax.tick_params(axis='both', which='both', labelsize='xx-small', right=True,
ax.set_xlabel(r"$X_1|X_2=1,X_3 = 1$", size='x-small')
ax.set_ylabel("Density", size='x-small')
#         ax.set_xlim(x_range)
#         ax.set_ylim(y_range)
ax.legend(loc = 1 ,fontsize = 3,markerscale = 2,ncol = 3,scatterpoints= 1,fr
plt.show()
```



3d. Use the sample from trivariate joint distributionto obtain the conditional mean and SD of X_1 given $X_2 = 1, X_3 = 1$. i.e Look at all the tripes whose second and third components are within an ϵ of .5, .05, .005. Then do a compare.

In [ ]:
```
# It is hard to generate such a number of random numbers, I use GPU to run t
epsilons = [.5, .05, .005]
pointNums = [1e3, 1e4, 1e5]
pool_1 = np.array([])
pool_2 = np.array([])
pool_3 = np.array([])
batch = int(1e7)
while(len(pool_3)<int(1e5)):
    X = np.random.multivariate_normal(MEAN, COV, size=batch, check_valid='wa
#     ----------------------------------
    afterX2_1 = X[np.abs(X[:,1]-1)<.5]
    afterX3_1 = afterX2_1[np.abs(afterX2_1[:,2]-1)<.5]
    X1_1 = afterX3_1[:,1]
    pool_1 = np.append(pool_1, X1_1)
    pool_1 = pool_1.flatten()
#     ----------------------------------
    afterX2_2 = X[np.abs(X[:,1]-1)<.05]
    afterX3_2 = afterX2_2[np.abs(afterX2_2[:,2]-1)<.05]
    X1_2 = afterX3_2[:,1]
```

```
    pool_2 = np.append(pool_2, X1_2)
    pool_2 = pool_2.flatten()
#     ----------------------------------
    afterX2_3 = X[np.abs(X[:,1]-1)<.005]
    afterX3_3 = afterX2_3[np.abs(afterX2_3[:,2]-1)<.005]
    X1_3 = afterX3_3[:,1]
    pool_3 = np.append(pool_3, X1_3)
    pool_3 = pool_3.flatten()

DF1 = pd.DataFrame(pool_1)
DF2 = pd.DataFrame(pool_2)
DF3 = pd.DataFrame(pool_3)
DF1.write("./pool1.csv")
DF2.write("./pool2.csv")
DF3.write("./pool3.csv")
```

In [37]:

```
# epsilons = [.5, .05, .005]
# pointNums = [1e3, 1e4, 1e5]
# pool_1 = np.array([])
# pool_2 = np.array([])
# pool_3 = np.array([])
# batch = int(1e7)
# while(len(pool_3)<int(1e5)):
#     MN = torch.distributions.multivariate_normal.MultivariateNormal(MEAN_t
#     X = MN.sample_n(batch).cuda()
#     afterX2_1 = X[torch.abs(X[:,1]-1)<.5]
#     afterX3_1 = afterX2_1[torch.abs(afterX2_1[:,2]-1)<.5]
#     X1 = np.array(afterX3_1[:,1].cpu())
#     pool_1 = np.append(pool_1, X1)
#     pool_1 = pool_1.flatten()
# #     ----------------------------------
#     afterX2_2 = X[torch.abs(X[:,1]-1)<.05]
#     afterX3_2 = afterX2_2[torch.abs(afterX2_2[:,2]-1)<.05]
#     X2 = np.array(afterX3_2[:,1].cpu())
#     pool_2 = np.append(pool_2, X2)
#     pool_2 = pool_2.flatten()
# #     ----------------------------------
#     afterX2_3 = X[torch.abs(X[:,1]-1)<.005]
#     afterX3_3 = afterX2_3[torch.abs(afterX2_3[:,2]-1)<.005]
#     X3 = np.array(afterX3_3[:,1].cpu())
#     pool_3 = np.append(pool_3, X3)
#     pool_3 = pool_3.flatten()
# #     ----------------------------------
#     time.sleep(0.001)
#     print(len(pool_3)/1e5, end = "\r")
# DF1 = pd.DataFrame(pool_1)
# DF2 = pd.DataFrame(pool_2)
# DF3 = pd.DataFrame(pool_3)
# DF1.to_csv("./output/pool1.csv")
# DF2.to_csv("./output/pool2.csv")
# DF3.to_csv("./output/pool3.csv")
```

In [58]:

```
pool_1 = np.array(pd.read_csv("./pool1.csv"))[:,1]
pool_2 = np.array(pd.read_csv("./pool2.csv"))[:,1]
pool_3 = np.array(pd.read_csv("./pool3.csv"))[:,1]
```

In [86]:

```
l3_1 = np..random.choice(pool_1, int(1e3))
l3_2 = np..random.choice(pool_2, int(1e3))
l3_3 = np..random.choice(pool_3, int(1e3))
l4_1 = np..random.choice(pool_1, int(1e4))
```

```
l4_2 = np..random.choice(pool_2, int(1e4))
l4_3 = np..random.choice(pool_3, int(1e4))
l5_1 = np..random.choice(pool_1, int(1e5))
l5_2 = np..random.choice(pool_2, int(1e5))
l5_3 = np..random.choice(pool_3, int(1e5))
# l3_1 = np.random.choice(pool_1, int(1e2))
# l3_2 = np.random.choice(pool_2, int(1e2))
# l3_3 = np.random.choice(pool_3, int(1e2))
# l4_1 = np.random.choice(pool_1, int(1e3))
# l4_2 = np.random.choice(pool_2, int(1e3))
# l4_3 = np.random.choice(pool_3, int(1e3))
# l5_1 = np.random.choice(pool_1, int(1e4))
# l5_2 = np.random.choice(pool_2, int(1e4))
# l5_3 = np.random.choice(pool_3, int(1e4))
SDTable = pd.DataFrame({"0.5": [np.std(l3_1), np.std(l4_1), np.std(l5_1)],
                        "0.05": [np.std(l3_2), np.std(l4_2), np.std(l5_2)],
                        "0.005": [np.std(l3_3), np.std(l4_3), np.std(l5_3)]})
MEANTable = pd.DataFrame({"0.5": [np.mean(l3_1), np.mean(l4_1), np.mean(l5_1
                        "0.05": [np.mean(l3_2), np.mean(l4_2), np.mean(l5_2)]
                        "0.005": [np.mean(l3_3), np.mean(l4_3), np.mean(l5_3)
SDTable.index = pd.Series(["1e3","1e4", "1e5"])
MEANTable.index = pd.Series(["1e3","1e4", "1e5"])
```

In [87]:
```
SDTable
```

Out[87]:

	0.5	0.05	0.005
1e3	0.441542	0.399323	0.473322
1e4	0.409784	0.437535	0.432946
1e5	0.421195	0.430867	0.426066

In [88]:
```
MEANTable
```

Out[88]:

	0.5	0.05	0.005
1e3	0.119986	0.179721	0.162527
1e4	0.148155	0.131988	0.144291
1e5	0.138369	0.131974	0.140682

In [ ]: