Inverse transformation method

1 Non uniform random numbers

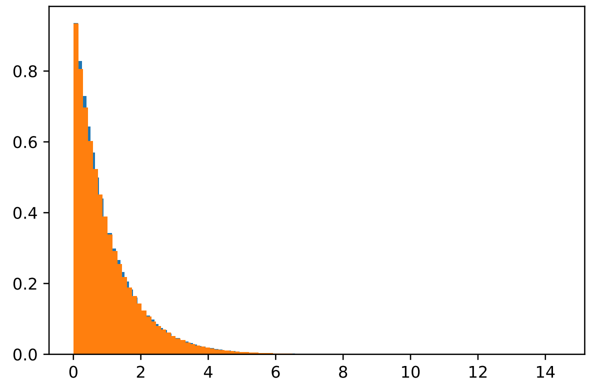
We illiustrate with Python the method of the Inverse Transformation method (and as usual the efficiency of the Python code by using the module time) .

1. Write a Python code which stores in an array 1000000 random numbers by using two methods : the member function exponential of the module np.random and the member function uniform (with a scale equal to 1) and using the inverse transformation uniform. Compare the elasped times of the two methods and display the histograms associated with the two methods on the same graph.

Solution:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 15 11:06:18 2021

@author: viot
"""

import numpy as np
import time
import matplotlib.pyplot as plt

start=time.time()
a=np.random.uniform(0,1,1000000)
b=-np.log(a)
end=time.time()
print(end-start)
start=time.time()
c=np.random.exponential(1,1000000)
end=time.time()
print(end-start)
plt.hist(b,bins=100,density=True)
plt.hist(c,bins=100,density=True)
plt.savefig('exponential.pdf')
```

The difference between two methods are very small but slightly in favor of the Inverse Transformation method. With a large number of random numbers, the normalized histograms are very close.



2. We want to illustrate with Python the method of the Box Müller method (and as usual the efficiency of the Python code by using the module time). Write a Python code which stores in an array 1000000 random numbers by using two methods : the member function randn of the module np.random and the member function uniform (with a scale equal to 1) and using the inverse transformation uniform. Compare the elapsed times of the two methods and display the histograms associated with the two methods on the same graph. By using some trickn obtain a ratio of the elapsed times less than 1.5

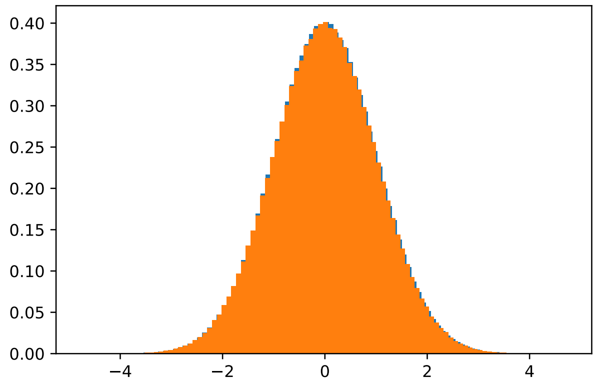
Solution:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 15 11:58:18 2021

@author: viot
"""

import numpy as np
import time
import matplotlib.pyplot as plt

start=time.time()
a=np.random.uniform(0,1,size=(2,500000))
b0=np.sqrt(-2*np.log(a[1]))
b=np.cos(2*np.pi*a[0])*b0
b1=np.sin(2*np.pi*a[0])*b0
b= np.concatenate((b,b1))
end=time.time()
```

```
print(end-start)

start=time.time()
c=np.random.randn(1000000)
end=time.time()
print(end-start)
plt.hist(b,bins=100,density=True)
plt.hist(c,bins=100,density=True)
plt.savefig('gaussian.pdf')
```

Because the numbers generated with a sin and a cos are independant, the number of uniform random numbers to generate is the same than the total number.

The concatenation slows down a little bit the code



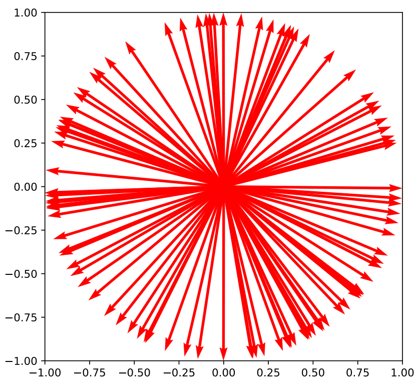
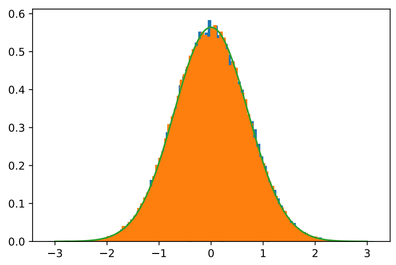
3. Generate 100 two-dimensional unit vectors uniformly. Plot the vectors by using the function **quiver** of **matplotlib.pyplot**.
4. By using 10000 independent runs, compute the average of each run (along 0x and 0y) and store results in two arrays. Build and plot the two histograms. Show that the distributions are well fitted by Gaussian distribution with a variance 1/2.

Solution:

The script python has two functions corresponding to the two answers

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 15 15:04:58 2021

@author: viot
"""
```

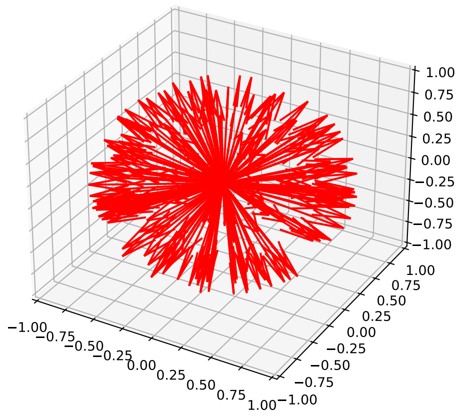
```python
import numpy as np
import time
import matplotlib.pyplot as plt
nbre=100

def gaussian(x, sig):
    return np.exp(-x*x / (2 *sig*sig)) /np.sqrt(2*np.pi)/sig

def plotvec(nbre):
    a=np.random.uniform(0,2*np.pi,size=(nbre))
    b=np.array([np.cos(a),np.sin(a)])
    plt.figure(figsize=(6,6))
    for i in np.arange(nbre):
        plt.quiver(0,0,b[0,i],b[1,i],angles='xy', scale_units='xy', scale
            ↪ =1,color='red')
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.savefig('vector2D.pdf')
    plt.show()

def average(nbre,nsim):
    sumx=np.zeros(nsim)
    sumy=np.zeros(nsim)
    for i in np.arange(nsim):
        a=np.random.uniform(0,2*np.pi,size=(nbre))
        sumx[i]=np.sum(np.cos(a))
        sumy[i]=np.sum(np.sin(a))
    plt.hist(sumx/np.sqrt(nbre),bins=100,density=True)
    plt.hist(sumy/np.sqrt(nbre),bins=100,density=True)
    x=np.linspace(-3,3,200)
    plt.plot(x,gaussian(x,1/np.sqrt(2)))
    plt.savefig("fluctua.pdf")
    plt.show()


plotvec(nbre)
start=time.time()
average(nbre,70000)
end=time.time()
print("elapsed time",end-start)
```

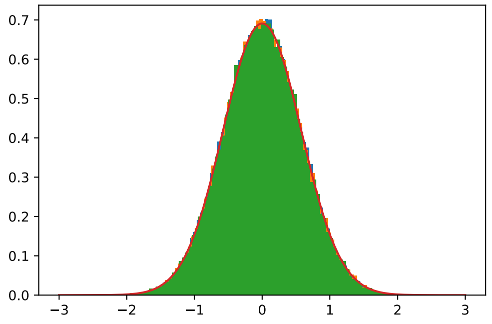



5. Generate 100 three-dimensional unit vectors uniformly. Plot the vectors by using the function **quiver** of **matplotlib.pyplot**.
6. By using 10000 independent runs, compute the average of each run (along 0x and 0y) and store results in three arrays. Build and plot the two histograms. Show that the distributions are well fitted by Gaussian distribution with a variance 1/3.

Solution:
The script python has two functions corresponding to the two answers

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 15 17:36:52 2021

@author: viot
"""

import numpy as np
import time
import matplotlib.pyplot as plt
nbre=200

def gaussian(x, sig):
    return np.exp(-x*x / (2 *sig*sig)) /np.sqrt(2*np.pi)/sig

def plotvec(nbre):
    phi=np.random.uniform(0,2*np.pi,size=(nbre))
    z=np.random.uniform(-1,1,size=(nbre))
    b=np.array([np.sqrt(1-z*z)*np.cos(phi),np.sqrt(1-z*z)*np.sin(phi),z])
    ax=plt.figure(figsize=(6,6)).add_subplot(projection='3d')

    for i in np.arange(nbre):
        ax.quiver(0,0,0,b[0,i],b[1,i],b[2,i],length=1, normalize=True,color='
            ↪ red')
    ax.set_xlim(-1,1)
    ax.set_ylim(-1,1)
    ax.set_zlim(-1,1)
    plt.savefig('vector3D.pdf')
    plt.show()

def average(nbre,nsim):
    sumx=np.zeros(nsim)
    sumy=np.zeros(nsim)
    sumz=np.zeros(nsim)
    for i in np.arange(nsim):
        phi=np.random.uniform(0,2*np.pi,size=(nbre))
        z=np.random.uniform(-1,1,size=(nbre))
        b=np.array([np.sqrt(1-z*z)*np.cos(phi),np.sqrt(1-z*z)*np.sin(phi),z])
        sumx[i]=np.sum(b[0])
        sumy[i]=np.sum(b[1])
        sumz[i]=np.sum(b[2])
    plt.hist(sumx/np.sqrt(nbre),bins=100,density=True)
    plt.hist(sumy/np.sqrt(nbre),bins=100,density=True)
    plt.hist(sumz/np.sqrt(nbre),bins=100,density=True)
    x=np.linspace(-3,3,200)
    plt.plot(x,gaussian(x,1/np.sqrt(3)))
    plt.savefig("fluctua3d.pdf")
    plt.show()


#plotvec(nbre)
start=time.time()
average(nbre,70000)
end=time.time()
print("elapsed time",end-start)
```


−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−3
−2
−1
0
1
2
3