4

Write programs to generate Halton, Sobol, or Faure sequences in dimensions up to 50

在创建sequence前需要先创建b-ary函数，因为下列各个sequences的创建都会使用到以下两个函数

In [27]:
```
def B_ARY(k, b):
    a=0
    if k > 0:
        jmax = np.floor(np.log(k)/np.log(b))
        a = np.repeat(0, jmax+1)
        q = b**jmax
        for j in range(int(jmax+1)):
            a[j] = np.floor(k/q)
            k = k-q*a[j]
            q = q/b
    return(a)
```

In [45]:
```
def NEXTB_ARY(ain, b):
    m = len(ain)
    carry = True
    aout = np.repeat(0, m)
    for i in range(m):
        if (carry):
            if (ain[m-1-i] == b-1):
                aout[m-1-i] = 0
            else:
                aout[m-1-i] = ain[m-1-i] + 1
                carry = False
        else:
            aout[m-1-i] = ain[m-1-i]
    if (carry):
        aout = aout.tolist()
        aout.insert(0,1)
    return (aout)
```

4.1 Halton sequence

首先创建$Van\ de\ Corput$函数来产生数据点，以下函数内，n_0 = 起始点、$npts$ = 点数量、b = 基数

In [69]:
```
def Van_de_Corput(n0, npts, b):
    nmax = n0 + npts -1
    a = B_ARY(n0-1, b)
    x = np.repeat(0, npts)
    x = x.tolist()
    for i in range(n0,nmax+1):
        a = NEXTB_ARY(a, b)
        m = len(a)
        s=0
        q = 1/b
        for j in range(m):
            s = s + q * a[m-1-j]
            q = q/b
        x[i-n0] = s
    return (x)
```

接着就能产生$Halton\ sequence$，以下函数内，n_0 = 起始点、$npts$ = 点数量、$bvec$ = 基数、d = 维度
下面列出五个维度的$Halton\ sequence$，可再调整$bvec$和d再拓展至50维

In [89]:
```
def HaltonPts(n0, npts, d, bvec):
    P = np.zeros(npts*d).reshape(npts,d)
    for i in range(d):
        P[:,i] = Van_de_Corput(n0, npts, bvec[i])
    return (P)
```

In [94]:
```
HaltonPts(2, 10, 5, [2,3,5,7,13])
```

Out[94]:
```
array([[0.25      , 0.66666667, 0.4       , 0.28571429, 0.15384615],
       [0.75      , 0.11111111, 0.6       , 0.42857143, 0.23076923],
       [0.125     , 0.44444444, 0.8       , 0.57142857, 0.30769231],
       [0.625     , 0.77777778, 0.04      , 0.71428571, 0.38461538],
       [0.375     , 0.22222222, 0.24      , 0.85714286, 0.46153846],
       [0.875     , 0.55555556, 0.44      , 0.02040816, 0.53846154],
       [0.0625    , 0.88888889, 0.64      , 0.16326531, 0.61538462],
       [0.5625    , 0.03703704, 0.84      , 0.30612245, 0.69230769],
       [0.3125    , 0.37037037, 0.08      , 0.44897959, 0.76923077],
       [0.8125    , 0.7037037 , 0.28      , 0.59183673, 0.84615385]])
```

4.2 Sobol sequence

接着来产生$Sobol\ Sequence$，下面函数$cvec$ = 多项式的参数向量、$minit$ = 初始向量、r = 矩阵大小，以此产生矩阵

In [33]:
```
def SobolMat(cvec, minit, r):
    q = len(cvec) - 1
    if (q == 0):
        V = np.identity(r)
    if (q > 0):
        V = np.zeros(r*r).reshape(r,r)
        mvec = np.repeat(0, r-q).tolist()
        for i in range(len(minit)):
            mvec.insert(0,minit[2-i])
        mstate = minit
        for i in range(q+1,r+1,1):
            mnext = np.repeat(0, r).tolist()
            for j in range(q):
                t=B_ARY(2**(j+1) * cvec[j + 1] * mstate[q-1-j], 2).toli
st()
                t1 = np.repeat(0, r -len(t)).tolist()
                for k in range(len(t)):
                    t1.append(t[k])
                t = t1
                mnext = np.abs(np.array(mnext) - np.array(t))
            mn1 = np.repeat(0, r-len(B_ARY(mstate[0],2))).tolist()
            mn2 = B_ARY (mstate[0],2).tolist()
            for l in range(len(mn2)):
                mn1.append(mn2[l])
            mnext = np.abs(mn1)
            mnext = sum(mnext*np.array([2**(r-p-1) for p in range(r)]))
            mvec[i-1] = mnext
            mstate[1:].append(mnext)

        mbin=[]
        for i in range(r):
            mbin.append(B_ARY(mvec[i], 2))
            mbin = [item for sublist in mbin for item in sublist]
            k = len(mbin)
            for j in range(k):
                V[i + 2 -j, i] = mbin[k - 1 - j]
    return(V)
```

接着用以下函数产生$sequence$，$n0$ = 初始点, $npts$ = 点数量, d = 维度, $pvec$ = 多项式向量, $mmat$ = 初始值

In [34]:
```
def GARYCODE(n):
    x1 = B_ARY(n, 2)
    x2 = B_ARY(np.floor(n/2), 2)
    l1 = len(x1)
    return(np.abs(np.array(x1)-np.array(np.repeat(0, l1 - len(x2)).toli
st().extend(x2))))
```

In [41]:
```
def SobolPts(n0, npts, d, pvec, mmat):
    nmax = n0 + npts - 1
    rmax = 1 + np.floor(np.log(nmax)/np.log(2))
    r=1
    P = np.zeros(npts*d).reshape(npts, d)
    y = np.zeros(rmax*d).reshape(rmax, d)
    if (n0 > 1):
        r = 1 + np.log(n0-1)/np.log(2)
    qnext = 2**r
    a = B_ARY(n0-1, 2)
    g = GARYCODE(n0-1)
    b_pwrs = (0.5)**(range(rmax))

    #用pvec的multinomial创建矩阵
    V = np.zeros(rmax*rmax*d).reshape(rmax, rmax, d)
    for i in range(d):
        q = np.floor(log(pvec[i])/log(2))
        cvec = B_ARY(pvec[i], 2)
        V[:,:,i] = SobolMat(cvec, mmat[i,1:q], rmax)


    #用graycode来计算点
    for i in range(d):
        for m in range(r):
            for n in range(r):
                y[m,i] = (y[m, i] + V[m, n, i] * g[r+1-n]) % 2

    for k in range(n0, nmax, 1):
        if (k == qnext):
            r=r+1
            g = [p for p in range(g)]
            l=1
            qnext = 2*qnext
        else:
            l=0
            for i in range(len(a)):
                if (a[i] == 0):
                    l=i
            g[l] = 1 - g[l]

    a = NEXTB_ARY(a, 2)

    #递回来计算k点
    for i in range(d):
        for m in range(r):
            y[m,i] = (y[m,i] + V[m, r+1-l, i]) % 2
            for j in range(r):
                P[k+1-n0, i] = P[k+1-n0, i] + b_pwrs[j] * y[j,i]
    return(P)
```

4.3 Faure sequence

接着产生$Faure\ sequence$，以下函数$n0$ = 初始点、$npts$ = 点数量、d = 维度、b = 基数

```
In [42]: def FaureMat(r, i):
    C = np.zeros(r*r).reshape(r, r)
    C[0,0] = 1
    if (r >= 2):
        for m in range(1,r,1):
            C[m, m] = 1
            C[0, m] = i*C[0, m-1]

    if(r >=3):
        for n in range(2,r,1):
            for m in range(1,n-1,1):
                C[m, n] = C[m-1, n-1] + i * C[m, n-1]

    return(C)
```

```
In [43]: def FaurePts(n0, npts, d, b):
    nmax = n0 + npts -1
    rmax = 1 + np.floor(np.log(nmax)/np.log(b))
    rmax = int(rmax)
    P = np.zeros(npts*d).reshape(npts, d)
    y = np.repeat(0, rmax)
    r = 1 + np.floor(np.log(max(1, n0-1))/np.log(b))
    qnext = b**r
    a = B_ARY(n0-1, b)
    C = np.zeros(rmax*rmax*(d-1)).reshape(rmax, rmax, d-1)
    for i in range(d-1):
        C[:,:,i] = FaureMat(rmax, i)

    b_pwrs = (1/b)**np.array([p for p in range(rmax)])
    for k in range(n0,nmax+1,1):
        a = NEXTB_ARY(a, b)
        if (k == qnext):
            r = r+1
            qnext = b*qnext
        for j in range(int(r)):
            P[k-n0, 0] = P[k-n0, 0] + b_pwrs[j] * a[int(r)-j-1]
        for i in range(1,d,1):
            for m in range(int(r)):
                for n in range(int(r)):
                    y[m] = y[m] + C[m,n,i-2]*a[int(r)-n-1]
            y[m] = y[m] % b
            e = b_pwrs*y
            P[k-n0, i] = P[k-n0, i] + e[m]
            y[m] = 0
    return(P)
```

接着就能产生$Faure\ sequence$，以下函数内，n_0 = 起始点、$npts$ = 点数量、$bvec$ = 基数、d = 维度
下面列出五个维度的$Faure\ sequence$，可再调整$bvec$和d再拓展至50维

```
In [61]: FaurePts(5, 10, 5, 3)

Out[61]: array([[2.33333333, 0.33333333, 0.33333333, 0.33333333, 0.33333333],
       [0.66666667, 0.66666667, 0.66666667, 0.66666667, 0.66666667],
       [1.66666667, 0.66666667, 0.66666667, 0.66666667, 0.66666667],
       [2.66666667, 0.66666667, 0.66666667, 0.66666667, 0.66666667],
       [0.11111111, 0.11111111, 0.11111111, 0.11111111, 0.11111111],
       [1.11111111, 0.11111111, 0.11111111, 0.11111111, 0.11111111],
       [2.11111111, 0.11111111, 0.11111111, 0.11111111, 0.11111111],
       [0.44444444, 0.11111111, 0.11111111, 0.11111111, 0.11111111],
       [1.44444444, 0.11111111, 0.11111111, 0.11111111, 0.11111111],
       [2.44444444, 0.11111111, 0.11111111, 0.11111111, 0.11111111]])
```

5

Empirically compare the L2 discrepancies of Halton, Sobol, or Faure sequences, and compare with the expected L2 discrepancy

Faure表现最好而Halton与Sobol差不多

In [1]:
```
import tensorflow as tf
import tensorflow_probability as tfp
import numpy as np
import openturns as ot
```

In [74]:
```
#create L2 discrepency
def square_dis(d,npts):
    x = np.random.uniform(0,1,d)
    y = np.random.uniform(0,1,d)
    volumn = np.prod(abs(x-y))
    s=0
    for i in range(npts):
        s = s + np.prod(i*(sample[i,:]<max(np.concatenate((x, y)))) & (
sample[i,:]>=min(np.concatenate((x, y)))))

    sresult = ((s/npts - volumn)**2)
    return sresult

def L2_discrepency(P):
    M = 1000
    npts = P.shape[0]
    d = P.shape[1]
    square_dis(d,npts)
    simu = []
    for i in range(M):
        simu.append(square_dis(d,npts))
    return(np.sqrt(np.mean(simu)))
```

In [120]:
```
# determine dim
num_results = 1024
dim = 5

#Halton
Halton = tfp.mcmc.sample_halton_sequence(
  dim,
  num_results=num_results,
  seed=127)
Halton = Halton.numpy()

#Sobol
Sobol = tf.math.sobol_sample(
    dim, num_results, skip=0, dtype=tf.dtypes.float32, name=None
)
Sobol = Sobol.numpy()

#Faure
sequence = ot.FaureSequence(5)
Faure = np.array(sequence.generate(5))
```

In [121]:
```
print('Halton\'s L2 discrepency',L2_discrepency(Halton))
print('Sobol\'s L2 discrepency',L2_discrepency(Sobol))
print('Faure\'s L2 discrepency',L2_discrepency(Faure))
```

```
Halton 0.23691619328228194
Sobol 0.2378269889141603
Faure 0.16893876452031778
```

6

Compare the efficiency of Halton, Sobol, or Faure sequences for high-dimensional financial problems (say, option pricing) and compare their efficiency with MC

以下面条件来设定option price:

In [244]:
```
r = 0.05
S0 = 50
sigma = 0.3
total_T = 1
d = 50
dt = total_T/d
t = [i/100 for i in range(d+1)][1:]
#t = t[-1]
w = np.repeat(1/d, d)
K = 50
```

接着分别创建Halton、Sobol、Faure的option函数

In [182]:
```
from scipy.stats import norm
import statistics
```

In [145]:
```
def Halton_option(N):
    G = np.repeat(0, N)
    for k in range(N):
        Halton = tfp.mcmc.sample_halton_sequence(d,num_results=N,seed=1
27)
        Halton = Halton.numpy()
        Z = norm.ppf(Halton)
        W = Z.sum(axis=0) * np.sqrt(dt)
        X = (r - 0.5 *sigma**2) * np.array(t) + sigma * W
        S = S0 * np.exp(X)
        G[k] = np.exp((-r)*total_T) * (np.prod(S**(1/d)) - K)
    return (G)
```

In [200]:
```
def Sobol_option(N):
    G = np.repeat(0, N)
    for k in range(N):
        Sobol = tf.math.sobol_sample(d, N, skip=0, dtype=tf.dtypes.floa
t32, name=None)
        Sobol = Sobol.numpy()
        Z = norm.ppf(Sobol)
        W = Z.sum(axis=0) * np.sqrt(dt)
        X = (r - 0.5 *sigma**2) * np.array(t) + sigma * W
        S = S0 * np.exp(X)
        G[k] = np.exp((-r)*total_T) * (np.prod(S**(1/d)) - K)
    return (G)
```

```
In [263]: def Faure_option(N):
              G = np.repeat(0, N)
              for k in range(N):
                  sequence = ot.FaureSequence(d)
                  Faure = np.array(sequence.generate(N))
                  Z = norm.ppf(Faure)
                  W = Z.sum(axis=0) * np.sqrt(dt)
                  X = (r - 0.5 *sigma**2) * np.array(t) + sigma * W
                  S = S0 * np.exp(X)
                  G[k] = np.exp((-r)*total_T) * (np.prod(S**(1/d)) - K)
              return (G)
```

以及一个最原始的MC option function

```
In [253]: def MC_option(N):
              G = np.repeat(0, N)
              for k in range(N):
                  Z = np.random.normal(0,1,d)
                  W = Z * np.sqrt(dt)
                  X = (r - 0.5 *sigma**2) * np.array(t) + sigma * W
                  S = S0 * np.exp(X)
                  G[k] = np.exp((-r)*total_T) * (np.prod(S**(1/d)) - K)
              return (G)
```

分别求variance来做比较，在高维度中其实变异都不是非常大

```
In [195]: H = Halton_option(500)
```

```
In [202]: S = Sobol_option(500)
```

```
In [264]: F = Faure_option(500)
```

```
In [254]: M = MC_option(500)
```

```
In [256]: print('Halton_option',statistics.variance(H))
          print('Sobol_option',statistics.variance(S))
          print('Faure_option',statistics.variance(F))
          print('MC_option',statistics.variance(M))
```

```
Halton_option 1
Sobol_option 0
Faure_option 0
MC_option 0
```

7

Using Halton, Sobol, or Faure sequences, in conjunction with random work construction, Brownian bridge construction and PCA construction of Brownian motion, for option pricing

以下分别创建RW、BB、PCA，并把 Halton、Sobol、Faure融入函数内

```
In [297]: from sklearn.decomposition import pca
```

In [277]:

```
#RW
def RW(n, d, total_T, method):
    BM = np.zeros(n*d).reshape(n, d)
    for i in range(n):
        if (method == 'halton'):
            Halton = tfp.mcmc.sample_halton_sequence(d,num_results=n,se
ed=127)
            P = Halton.numpy()
        if (method == 'sobol'):
            Sobol = tf.math.sobol_sample(d, n, skip=0, dtype=tf.dtypes.
float32, name=None)
            P = Sobol.numpy()
        if (method == 'faure'):
            sequence = ot.FaureSequence(d)
            P = np.array(sequence.generate(n))

        Z = norm.ppf(P)
        dt = total_T/d
        BM[i,:] = Z.sum(axis=0) * np.sqrt(dt)
    return (BM)
```

In [290]:

```
#BB
def BB(n, d, total_T, method):
    m =np.floor(np.log(d)/np.log(2))
    BM = np.zeros(n*(d+1)).reshape(n, d+1)
    t = [i/(d+1) for i in range(total_T)]
    for i in range(n):
        if (method == 'halton'):
            Halton = tfp.mcmc.sample_halton_sequence(d,num_results=n,se
ed=127)
            P = Halton.numpy()
        if (method == 'sobol'):
            Sobol = tf.math.sobol_sample(d, n, skip=0, dtype=tf.dtypes.
float32, name=None)
            P = Sobol.numpy()
        if (method == 'faure'):
            sequence = ot.FaureSequence(d)
            P = np.array(sequence.generate(n))

        Z = norm.ppf(P)
        j=1
        BM[i,int(2**m)] = np.sqrt(t[int(2**m)])*Z[i, j]
        delta = 2**m
        p_max = 1
        for k in range(m):
            s_min = delta/2
            s = s_min
            l=0
            r = delta
            for p in range(p_max):
                j=j+1
                a = ((t[r+1]-t[s+1])*BM[i,l+1] + (t[s+1]-t[l+1])*BM[i,r
+1])/(t[r+1]-t[l+1])
                b = sqrt((t[s+1]-t[l+1]) * (t[r+1]-t[s+1]) / (t[r+1] -
t[l+1]))
                BM[i, s+1] = a + b * Z[i,j]
                s = s + delta
                l = l + delta
                r = r + delta

            p_max = 2 * p_max
            delta = s_min

        if (d > 2**m):
            for k in range((2**m+1),d,1):
                j=j+1
                BM[i,k+1] = sqrt(t[k+1] - t[k]) * Z[i, j]
    return(BM)
```

In [302]:

```
#PCA
def PCA(n, d, total_T, method):
    BM = np.zeros(n*d).reshape(n, d)
    dt = total_T/d
    C = np.zeros(d*d).reshape(d, d)
    for i in range(d):
        if (method == 'halton'):
            Halton = tfp.mcmc.sample_halton_sequence(d,num_results=n,se
ed=127)
            P = Halton.numpy()
        if (method == 'sobol'):
            Sobol = tf.math.sobol_sample(d, n, skip=0, dtype=tf.dtypes.
float32, name=None)
            P = Sobol.numpy()
        if (method == 'faure'):
            sequence = ot.FaureSequence(d)
            P = np.array(sequence.generate(n))
        Z = norm.ppf(P)
        for j in range(d):
                C[i, j] = min(i,j) * dt

    cpca = pca.score(C, y=None)
    for k in range(n):
         BM[k,:] = np.dot(cpca,np.array(Z[k,]).reshape( d, 1))
    return(BM)
```

使用方式如下，不同的路径和方法以此类推：

In [278]:

```
RW(5,3,1,'halton')
```

Out[278]:

```
array([[-1.48826058, -2.20629799,  0.01321616],
       [ 0.31973544,  0.42830123,  1.26648916],
       [ 0.16986964,  0.59755812,  0.78399475],
       [ 0.91223935,  0.09704928, -0.60806908],
       [-0.45806392,  0.85682255,  0.31984927]])
```

8

Using good lattice rules (say, Korobov lattice rules), in conjunction with BB or PCA, for option pricing

In [68]:

```python
#使用BB
def korobov_BB(n, d, generator, total_T):
    P = korobov(n, d, generator)
    Z = norm.ppf(P)
    m =np.floor(np.log(d)/np.log(2))
    BM = np.zeros(n*(d+1)).reshape(n, d+1)
    t = [i/d for i in range(total_T)]
    t = t[1:]
    for i in range(n):
        j=1
        BM[i,2**m+1] = np.sqrt(t[2**m + 1])*Z[i, j]
        delta = 2**m
        p_max = 1
        for k in range(m):
            s_min = delta/2
            s = s_min
            l=0
            r = delta
            for p in range(p_max):
                j=j+1
                a = ((t[r+1]-t[s+1])*BM[i,l+1] + (t[s+1]-t[l+1])*BM[i,r+1])/(t[r+1]-t[l+1])
                b = np.sqrt((t[s+1]-t[l+1]) * (t[r+1]-t[s+1]) / (t[r+1] - t[l+1]))
                BM[i, s+1] = a + b * Z[i,j]
                s = s + delta
                l = l + delta
                r = r + delta

            p_max = 2 * p_max
            delta = s_min
        if (d > 2**m):
            for k in range(2**(m+1),d,1):
                j=j+1
                BM[i,k+1] = sqrt(t[k+1] - t[k]) * Z[i, j]
    return(BM)
```

In [63]:

```python
#使用PCA
def korobov_PCA(n, d, generator, total_T):
    P = korobov(n, d, generator)
    Z = norm.ppf(P)
    BM = np.zeros(n*d).reshape(n, d)
    dt = total_T/d
    C = np.zeros(d*d).reshape(d, d)
    for i in range(d):
        for j in range(d):
            C[i, j] = min(i,j) * dt
    pca = pca.score(C, y=None)
    for k in range(n) :
        BM[k,:] =  np.dot(pca,np.array(Z[k,]).reshape( d, 1))
    return(BM)
```