Peer Assessments (https://class.coursera.org/smac-001/human_grading/)
/ Homework Session 4: Sampling and integration in high dimensions
Help (https://class.coursera.org/smac-001/help/peergrading?url=https%3A%2F%2Fclass.coursera.org%2Fsmac-001%2Fhuman_grading%2Fview%2Fcourses%2F971628%2Fassessments%2F8%2Fresults%2Fmine)

Submission Phase

1. Do assignment ☑ (/smac-001/human_grading/view/courses/971628/assessments/8/submissions)

Evaluation Phase

2. Evaluate peers ☑ (/smac-001/human_grading/view/courses/971628/assessments/8/peerGradingSets)

Results Phase

3. See results ☑ (/smac-001/human_grading/view/courses/971628/assessments/8/results/mine)

Your effective grade is 20

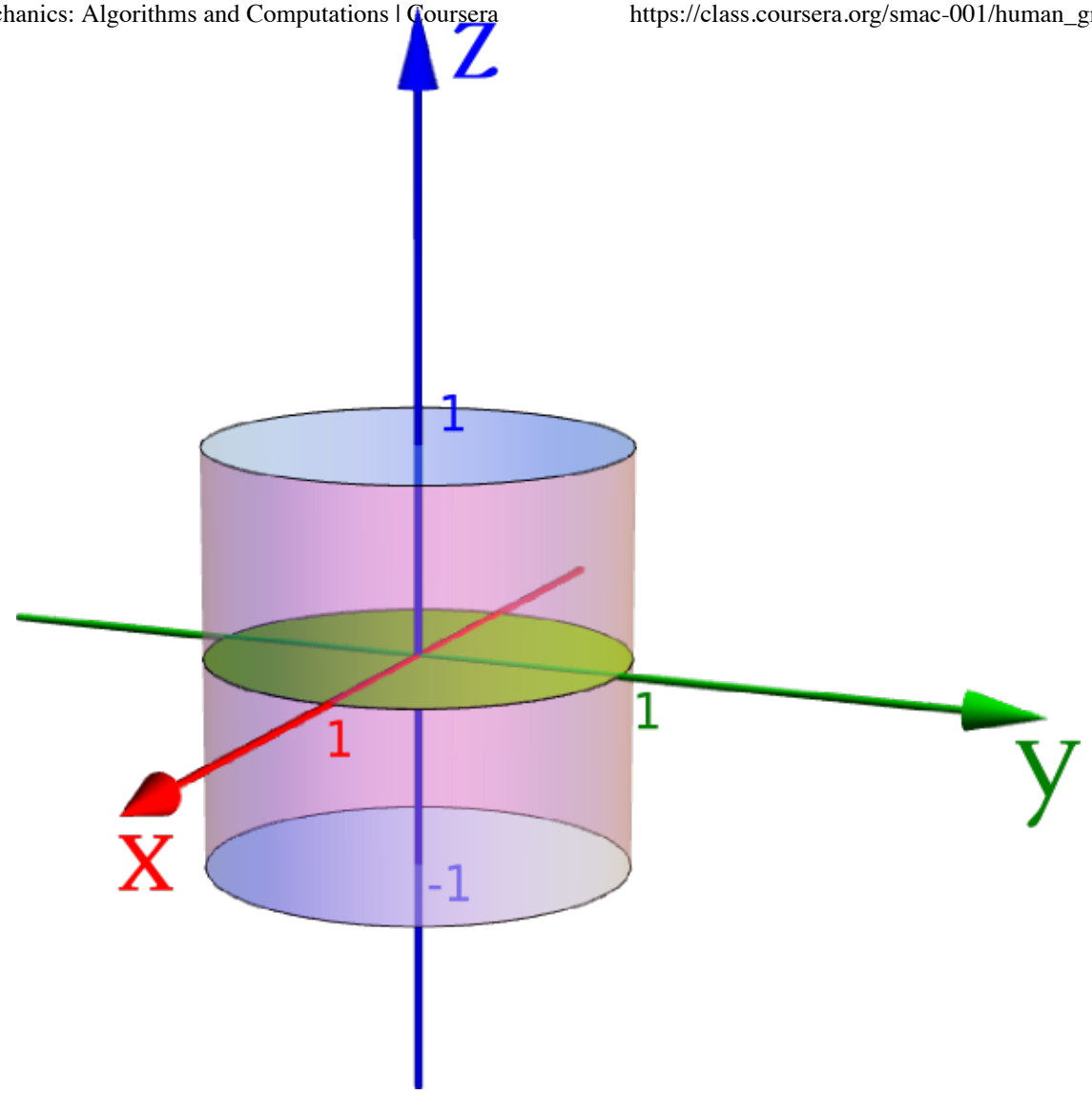
Your unadjusted grade is 20, which is simply the grade you received from your peers.

See below for details.

In this homework session 4 of Statistical Mechanics: Algorithms and Computations, we experience the power of Markov-chain Monte Carlo algorithms, and the intimate relation between sampling and integration.

The story of this homework involves hypherspheres, hypercubes and hypercylinders (see glossary):

- Sampling a point in a hypercube is easy (and computing its volume trivial). This allows us to sample a point in the hypersphere, and to compute the hypersphere volume, as we already did in week 1, but in high dimension, this recipe is no good.
- Sampling a point in a (d+1)-dimensional hypercylinder is easy for all d, if we know how to sample a point in the d-dimensional hypersphere. This allows us to compute the ratio of volumes of the (d+1)-dimensional hypersphere to the d-dimensional hypersphere.
- A bit of "accordion playing" then allows us (allows you!) to compute the volume of the d-dimensional hypersphere.



Summary: We first provide a glossary of terms and a preparation program:

GLOSSARY OF TERMS:

Sphere of radius r: a three-dimensional object of all points x, y, z with x^2 + y^2 + z^2 < r^2. Its volume is Vol_s(r, d=3).

Unit sphere: sphere of radius 1. Its volume is Vol1_s(d=3).

Unit hypersphere in d dimensions: d-dimensional object of all points x_0, ..., x_{d-1} with x_0^2 + ... + x_{d-1} ^2 < 1. Its volume is Vol1_s(d).

Unit hypercube in d dimensions: d-dimensional object of all points x_0, ..., x_{d-1} with -1 < x_0 < 1,.... -1 < x_{d-1} < 1. Its volume is Vol1_cube(d).

Unit cylinder: a three-dimensional object of all points x, y, z with x^2 + y^2 < 1 and -1 < z < 1. Its volume is Vol1_cyl(d=3).

Unit hypercylinder in d dimensions: d-dimensional object of all points

x_0, ..., x_{d-1} with x_0^2 + ... + x_{d-2} ^2 < 1 and -1 < x_{d-1} < 1. Its volume is Vol1_cyl(d).

PREPARATION PROGRAM

```
import random, math

def Vol1_s(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.0)

for dimension in range(1,20):
    print dimension, Vol1_s(dimension)
```

A1

Study the **glossary of terms**. Then answer the following questions

1. What is the volume of the three-dimensional sphere of radius r, Vol_s(r,3)? (give formula)
2. What is the volume of the three-dimensional unit sphere? (give formula)
3. What is the volume of the d-dimensional unit hypersphere? (give formula) (NB: see PREPARATION PROGRAM)
4. What is the volume of the (d+1)-dimensional unit hypercylinder Vol1_cyl(d+1)?(give formula involving Vol1_s(d)). **MAKE SURE TO ANSWER THIS QUESTION**!
5. What is the ratio of volumes of the (d+1)-dimensional unit hypercylinder the d-dimensional unit hypersphere? (give exact result) **MAKE SURE TO ANSWER THIS QUESTION**!
6. What is the volume of the d-dimensional unit hypercube? (give formula)

1. Vol_s(r,3) = $\frac{4\pi r^3}{3}$

2. Vol1_s(3) = $\frac{4\pi}{3}$

3. Vol1_s(d) = $\frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)}$

4. Vol1_cyl(d+1) = 2 * Vol1_s(d))

5. Vol1_cyl(d+1)/Vol1_s(d)) = 2

6. Vol1_cube(d) = $1^d = 1$

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

A1 Here, you are asked to evaluate your fellow student's understanding of a glossary of terms in simple geometry. There are 6 aspects (questions)
An example of an answer that would receive full score is as follows:
1/ 4/3 pi r^3
2/ 4/3 pi (or 4.188)

3/ pi^(d/2) / Gamma(d/2 + 1), where Gamma is the Gamma function (an answer with factorials is also OK)
4/ 2.0 * Vol1_s(d)
5/ 2 (NB: this question is equivalent to question 4)
6/ 2^d

POINTS - DESCRIPTION
Give 0 points if 0 questions are treated correctly
Give 1 points if 1 or 2 aspects are treated correctly
Give 2 points if 3 or 4 aspects are treated correctly
Give 3 points if 5 or 6 aspects are treated correctly

Score from your peers: **3**

peer 1 → Attention, the lenght of a side of the unit hypercube is 2 !

peer 2 → *[This area was left blank by the evaluator.]*

A2
Download (cut-and-paste) and run the **Preparation Program**. Then do the following

1/ **Can you confirm** that this program computes the volume of the unit hypersphere in dimensions from 1 to 19? (Yes/No answer)
2/ **Explain how to modify** this program to compute the volume of the hypersphere of radius r.
3/ Extend the **Preparation Program** into a graphics (movie) version, and **plot the quantity** "Vol1_s" for all dimensions d from 1 through 200. Use linear scale in x and logarithmic scale in y. **Upload the modified program** and **upload a figure** showing your results, and make sure that it presents the correct scaling (linear in x, logarithmic in y), has correctly labelled axes and a descriptive title. **Compute and communicate** the numerical values Vol1_s(5), Vol1_s(20), Vol1_s(200), as you will need them later.

1. Yes

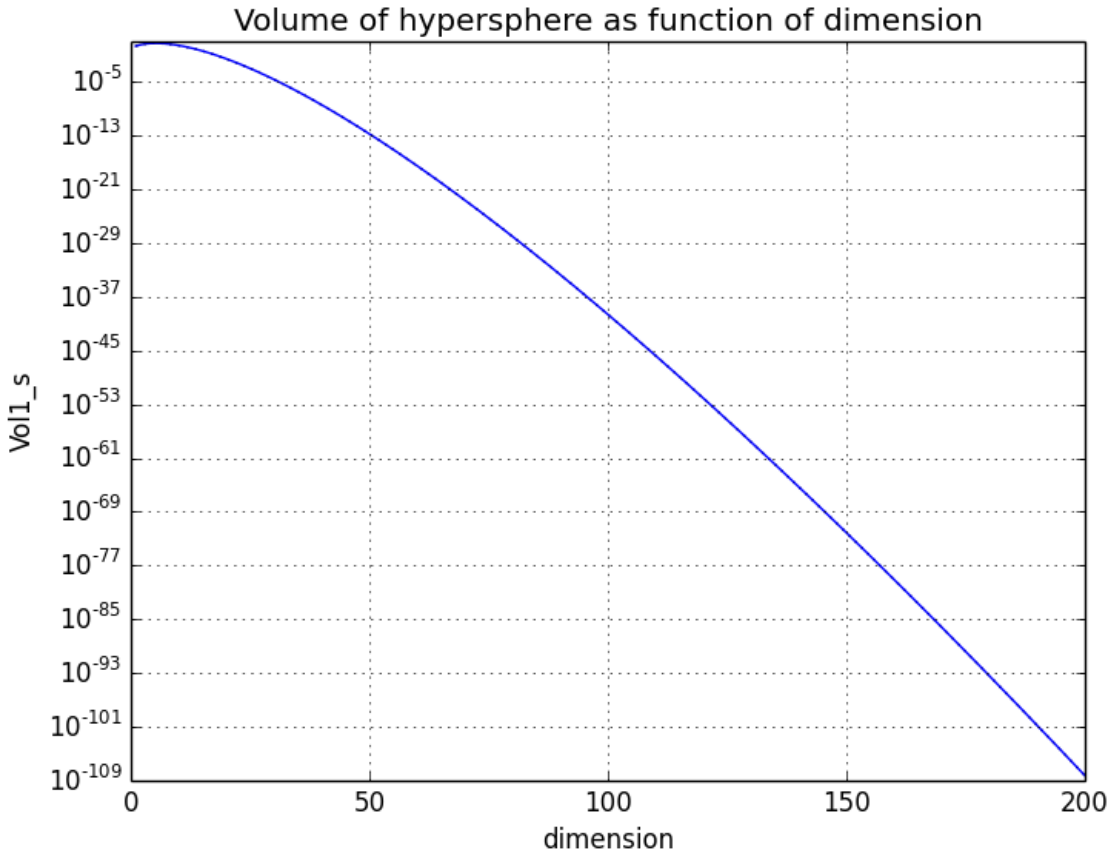
2. Vol_s(r, d) = $\frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} r^d$

The code should be changed as below:

```
def Vol_s(radius, dimension):
    return (radius**dimension) * math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2
.0 + 1.0)
```

`3. Code is as follows:`

```
import random, math, pylab

def Vol1_s(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.0)

dimensions_list = []
Vol1_s_list = []

for dimension in range(1,201):
    dimensions_list.append(dimension)
    Vol1_s_list.append(Vol1_s(dimension))
    print dimension, Vol1_s(dimension)

pylab.plot(dimensions_list, Vol1_s_list, ':k')
pylab.plot(dimensions_list, Vol1_s_list)
pylab.xlabel('dimension')
pylab.ylabel('Vol1_s')
pylab.yscale('log')
pylab.grid()
pylab.title('Volume of hypersphere as function of dimension')
pylab.savefig('Vol1_s_dim.png')
pylab.show()
```



Vol1_s(5) = 5.26378901391
Vol1_s(20) = 0.02580689139
Vol1_s(200) = 5.55883284203e-109

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

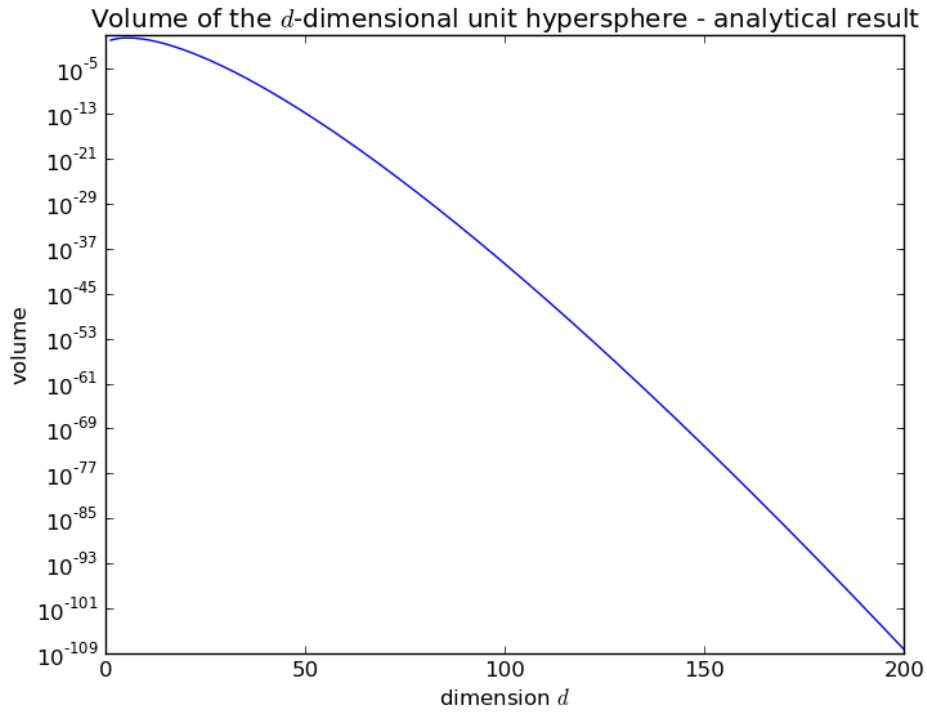
Here, a mixture of simple questions and some programming of a Python program in five aspects.

You can give a total of 3 points.
Give 1 points if questions 1 and 2 are both answered correctly
Give 1 additional point if the program is correct and the correct plot is uploaded
Give 1 additional point if the volumes for d=5, d=20 d=200 are computed correctly

An example of an answer that would receive full score is as follows:

The answer to question 1 is YES
The answer to question 2 is: multiply with r ** dimension (or similar answer)
A correct program and the corresponding plot is shown here.

```
import random, math, pylab

def Vol1_s(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.
0)

x_data = []
y_data = []

for dimension in range(1,201):
    x_data.append(dimension)
    y_data.append(Vol1_s(dimension))
dimension = 20; print dimension, Vol1_s(dimension)
dimension = 50; print dimension, Vol1_s(dimension)
dimension = 200; print dimension, Vol1_s(dimension)
pylab.plot(x_data, y_data)
pylab.xlabel('dimension $d$')
pylab.ylabel('volume')
pylab.title('Volume of the $d$-dimensional unit hypersphere - analytical
 result')
pylab.yscale('log')
pylab.savefig('vol_unit_hypersphere.png')
pylab.show()
```



The volumes for d=20, 50, 200 are as follows:

d= 20 Vol1(d) = 0.02580689139

d= 50 Vol1(d) = 1.73021924584e-13

d=200 Vol1(d) = 5.55883284203e-109

POINTS - DESCRIPTION

POINTS ARE ADDITIVE:

Give 1 points if **questions 1 and 2 are both answered correctly**

Give 1 point if the **program is correct and the correct plot is uploaded**

Give 1 point if the **volumes for d=5, d=20 d=200 are computed correctly**

Score from your peers: **2.5**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

B

Summary:

You first write a **direct sampling algorithm** for points in the d-dimensional unit hypercube. This generalizes what we did in week 1.

You then **compute Vol1_s(d)/Vol1_cube(d)** from the ratio of hits to trials in generalization of what you already did in homework session 1.

You then write a **Markov-chain algorithm inside the unit hypersphere in d dimensions.**

You then **compare the (d+1)-dimensional unit hypersphere to the (d+1)-dimensional unit-hypercylinder.**

Finally, some music! You will **play the accordion** to compute the volume of the 200-dimensional unit hypersphere

B1

Generalize the program direct_pi_multirun.py from d-2 dimensions into a python program in d dimensions. This allows you to sample points in the d-dimensional unit hypercube and to compute Vol1_s(d)/Vol1_cube(d) from the ratio of hits to trials.

In your algorithm, implement the tabula-rasa rule, that is, break out of the construction of the vector x_0, x_1,..., x_k, ... x_{d-1} as soon as x_0^2 + ... + x_k^2 > 1. (You may also sample all the x_0,..., x_{d-1}, and check later, but this is less good.).

Use n_trials=1000000 pebbles for each dimension d-1,2,3,...,12. You may repeatedly run the program for dimensions d=1,2,3... or implement a loop over dimensions... Stop as soon as, for a given dimension, you have zero hits for n_trials trials (this will appear quite rapidly).

Provide a table presenting

```
-------------------------------------------------------
n_trials used for all
d | estimation of Vol1_s(d) | Vol1_s(d) (exact) | n_hits
-------------------------------------------------------
1   ...                        ..                ....
2
...
```

Upload your python program, and the **table (as a simple text file)**. Don't forget to **indicate the number of trials** in the title of your table

```
import random, math

def Vol1_s(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.0)

def direct_pi(N,d):
    n_hits = 0
    for i in range(N):
        sum_x2 = 0.0
        flag = 0

        for j in range(d):
            x = random.uniform(-1.0, 1.0)
            sum_x2 += (x ** 2.0)
            if sum_x2 > 1.0:
                flag = 1
                break

        if flag == 0:
            n_hits += 1

    return n_hits

n_trials = 1000000
d = 1
f = open('hypersphere_volumes.txt', 'w')
f.write('d | Vol1_s(d) [estimate] | Vol1_s(d) [exact] | n_hits\n')

while True:
    Vol1_s_exa = Vol1_s(d)

    n_hits = direct_pi(n_trials,d)
    Vol1_cube = float(2 ** d)
    Vol1_s_est = Vol1_cube * (float(n_hits) / float(n_trials))

    f.write(str(d)+'\t'+str(Vol1_s_est)+'\t\t'+str(Vol1_s_exa)+'\t\t'+str(n_hits)+'\n')

    if n_hits > 0:
        d += 1
    else:
        break
```

```
Number of trials = 1000000

d | Vol1_s(d) [estimate] | Vol1_s(d) [exact] | n_hits
1     2.0                    2.0                 1000000
2     3.140852               3.14159265359        785213
3     4.195904               4.18879020479        524488
4     4.93248                4.93480220054        308280
5     5.275456               5.26378901391        164858
6     5.163008               5.16771278005         80672
7     4.70912                4.72476597033         36790
8     4.004864               4.05871212642         15644
9     3.33824                3.29850890274          6520
10    2.494464               2.55016403988          2436
11    1.941504               1.88410387939           948
12    1.437696               1.33526276885           351
13    0.917504               0.910628754783          112
14    0.638976               0.599264529321           39
15    0.425984               0.381443280823           13
16    0.0                    0.235330630359            0
```

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here you are asked to check that a submitted Python program is OK.
There are two aspects to check:
1/ Monte Carlo results should correspond to the analytical results.
2/ The program should really sample 1000000 points in the hypercube,
then check whether the point is in the hypersphere.

A version of the program where the number of hits is fixed, and not the number of trials is inexact.

You can give a total of 2 points.
Give 0 points if 0 aspects are treated correctly
Give 1 points if 1 aspects is treated correctly
Give 2 points if 2 aspects are treated correctly

NB: Note that this exercise is non-trivial. Take away points only if there are serious problems with the solution.

An example of an answer that would receive full score is as follows:

```
import random, math

def volume_analytic(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.
0)

dimension_max = 20
n_trials = 1000000
for dimension in range(1, dimension_max + 1):
    n_reject = 0
    for trial in range(n_trials):
        hyperradius_square = 0.0
        for d in range(dimension):
            hyperradius_square += random.uniform(-1.0, 1.0) ** 2
            if hyperradius_square > 1.0:
                n_reject += 1
                break
    n_accept = n_trials - n_reject
    volume = 2.0 ** dimension * n_accept / float(n_trials)
    if n_accept > 0:  print dimension , volume, \
                          volume_analytic(dimension), n_accept
    else:
        print dimension , 'no sample', volume_analytic(dimension)
```

Here the table:
n_trials = 1000000
d | estimation of Vol1_s(d) | Vol1_s(d) (exact) | n_hits
---
1 2.0 2.0 1000000
2 3.140236 3.14159265359 785059
3 4.186336 4.18879020479 523292
4 4.9432 4.93480220054 308950
5 5.257856 5.26378901391 164308
6 5.168192 5.16771278005 80753
7 4.756736 4.72476597033 37162
8 4.06528 4.05871212642 15880
9 3.218432 3.29850890274 6286
10 2.54464 2.55016403988 2485
11 1.794048 1.88410387939 876
12 1.45408 1.33526276885 355
13 0.876544 0.910628754783 107
14 0.688128 0.599264529321 42
15 0.425984 0.381443280823 13
16 no sample 0.235330630359

THIS IS DIFFICULT MATERIAL, SUBSTRACT POINTS ONLY if there are severe problems with the graphics or the program.

POINTS - DESCRIPTION
Give 0 points if 0 aspects are treated correctly
Give 1 points if 1 aspects is treated correctly
Give 2 points if 2 aspects are treated correctly

Score from your peers: **1.5**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

B2
Generalize markov_pi.py to implement a Markov-chain Monte Carlo algorithm to sample point inside the d-dimensional unit hypersphere:

Represent the configuration point as a list: x = [x_0, x_1,...., x_k, ..., x_{d - 1}]

Start at the origin [0.0, 0.0, 0.0, ... 0.0] (program this as: x = [0] * d). Instead of changing all dimensions at a time, as we did in markov_pi.py,
for all iterations i = 0, 1, 2,...., n_trials:
sample ONE dimension k and propose a change of x[k] only:

```
k = random.randint(0, d - 1)
x_new_k = x[k] + random.uniform(-1.0, 1.0)
```

ACCEPT this move if the new radius < 1, otherwise REJECT and remain where you are.
Test this program for d=2. More precisely:
1/ Produce a scatter plot of positions (for yourself, do not upload). There is no need to produce a graphics file, as it would not show the "piles".
2/ Compute the mean value of r^2 = x[0] **2 + x[1] ** 2. In d=2, this value should be equal to 1/2.
3/ Explain why <r^2> = 1/2 (hint: compute two integrals)
4/ Upload your program for general d. It should be correct, but there is **no need yet for it to be efficient**.

1. Scatterplot not uploaded as asked.

2. The mean value of r^2 is indeed around 0.5.

3. It's easier to analyze this in polar coordinate system.

$\langle r^2 \rangle = \int_0^{2\pi} \int_0^1 r^2 \ f(r, \phi) \ r dr \ d\phi$ where $f(r, \phi)$ is the probability density function.

The sampling is uniformly random in cartesian coordinate system. So, it's also random in the polar coordinate system which implies that random r and ϕ are chosen uniformly independent of each other.

$f(r, \phi) = \frac{1}{1-0} \frac{1}{2\pi - 0} C = \frac{C}{2\pi}$ where C is normalization constant.

Since we are sampling all the time in the 2D disk, we have $\int_0^{2\pi} \int_0^1 f(r, \phi) \ r dr \ d\phi = 1$ since the total probability should be 1.

Thus, $\int_0^{2\pi} \int_0^1 \frac{C}{2\pi} \ r dr \ d\phi = 1$ yielding C = 2. Thus, $f(r, \phi) = \frac{1}{\pi}$.

Finally, $\langle r^2 \rangle = \int_0^{2\pi} \int_0^1 r^2 \frac{1}{\pi} r dr d\phi = \frac{1}{\pi} \int_0^{2\pi} d\phi \int_0^1 r^3 dr = 2 \left. \frac{R^4}{4} \right|_0^1 = \frac{1}{2} = 0.5$

4. Program code:

```
import random, pylab

def markov_pi (N,d):
        n_hits = 0
        x = [0.0] * d
        radius_sq = 0.0
        sum_radius_sq = 0.0
        x0_list= []
        x1_list=[]
        x0_list.append(x[0])
        x1_list.append(x[1])

        for i in range(n_trials):
                k = random.randint(0, d - 1)
                x_new_k = x[k] + random.uniform(-1.0, 1.0)

                radius_sq_new = radius_sq - (x[k] ** 2.0) + (x_new_k ** 2.0)

                if radius_sq_new < 1:
                        x[k] = x_new_k
                        radius_sq = radius_sq_new
                        sum_radius_sq += radius_sq
                        n_hits += 1
                else:
                        sum_radius_sq += radius_sq

                x0_list.append(x[0])
                x1_list.append(x[1])

        pylab.plot(x0_list, x1_list, 'o')
        pylab.xlabel('x0')
        pylab.ylabel('x1')
        pylab.grid()
        pylab.title('x0 vs x1 scatterplot')
        pylab.savefig('x0_x1_scatterplot.png')
        pylab.show()

        return (n_hits, sum_radius_sq)

n_trials = 1000000
d = 2
(n_hits, sum_radius_sq) = markov_pi(n_trials,d)
print 4.0 * n_hits / float(n_trials), sum_radius_sq / float(n_trials)
```

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here you are asked to check three issues:

1/ The correct observation \sim 0.5
2/ The explanation in terms of an integral
3/ The program should be correct

An example of an answer that would receive full score is as follows:
< r^2 > is very close to 0.5
The explanation is the following:
= \int dx dy x^2 + y ^2/ \int dx dy (integration over unit disk0
= \int r dr r^2 / int dr r (between 0 and 1)
= \int r^3 / \int r = (1/4) / (1/2) = 1/2

```
import random, math

def volume_analytic(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dime
nsion / 2.0 + 1.0)

dimension_max = 20
n_trials = 1000000
for dimension in range(1, dimension_max + 1):
    n_reject = 0
    for trial in range(n_trials):
        hyperradius_square = 0.0
        for d in range(dimension):
            hyperradius_square += random.uniform(-1.0, 1
.0) ** 2
            if hyperradius_square > 1.0:
                n_reject += 1
                break
    n_accept = n_trials - n_reject
    volume = 2.0 ** dimension * n_accept / float(n_trial
s)
    if n_accept > 0:  print dimension , volume, \
               volume_analytic(dimension), n_accept
    else:
        print dimension , 'no sample', volume_analytic(d
imension)
```

POINTS - DESCRIPTION

Give 0 points if 0 aspects are treated correctly

Give 1 points if 1 aspects is treated correctly

Give 2 points if 2 aspects are treated correctly

Give 3 points if 3 aspects are treated correctly

Score from your peers: **2.5**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

B3

Extend the python program of section B2: Complement the random sample x_0...x_{d-1} in the d-dimensional hypersphere to a random sample in the (d+1)-dimensional unit HYPERCYLINDER, by simply drawing one additional random number x_supp = random.uniform(-1.0, 1.0).
x_0,...x_{d-1},x_supp
Now, define an observable Q
Q = 1 if x_0^2 + ... + x_{d - 1} ^2 + x_supp ^2 < 1
Q = 0 otherwise.

Show the remarkable formula

2 * <Q> = Vol1_s(d + 1) / Vol1_s(d)

Upload your program, it should be correct but does not have to be efficient (see B4)

Upload your results for Vol1_s(d + 1) / Vol1_s(d) for d = 1, 2.

By sampling *x_supp* using random.uniform(-1.0, 1.0), we are sampling points in (d+1) hypercylinder. Then, we are checking if the entire set of points i.e. (x_0,...x_{d-1}, x_supp) also lies in the (d+1) hypersphere. The probability of a point being in both (d+1) hypercylinder and (d+1) hypersphere is given by <Q>.

Therefore, <Q> = Vol1_s(d+1)/Vol1_cyl(d+1) ..... (1)
Now, as in A1.4, Vol1_cyl(d+1) = 2 * Vol1_s(d) .... (2)

Combining (1) and (2), we get **Vol1_s(d+1)/Vol1_s(d) = 2 * <Q>**

```
import random, pylab, math

def Vol1_s(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.0)

def markov_pi (N, d):
        x = [0.0] * d
        radius_sq = 0.0
        sum_Q = 0

        for i in range(n_trials):
                k = random.randint(0, d - 1)
                x_new_k = x[k] + random.uniform(-1.0, 1.0)
                radius_sq_new = radius_sq - (x[k] ** 2.0) + (x_new_k ** 2.0)

                if radius_sq_new < 1:
                        x[k] = x_new_k
                        radius_sq = radius_sq_new

                x_supp = random.uniform(-1.0, 1.0)
                if radius_sq + x_supp ** 2 < 1:
                        sum_Q +=1

        return (float(sum_Q)/float(N))

n_trials = 1000000
d = 1
Q_average = markov_pi (n_trials, d)

print 2 * Q_average, Vol1_s(d + 1) / Vol1_s(d)
```

d=1:
2*<Q> = 1.570432 [Estimate of Vol1_s(2) / Vol1_s(1)],
Vol1_s(2) / Vol1_s(1) = 1.57079632679 [Exact]

d=2:
2*<Q> = 1.333124 [Estimate of Vol1_s(3) / Vol1_s(2)]
Vol1_s(3) / Vol1_s(2) = 1.33333333333 [Exact]

Clearly, the <Q> method provides accurate estimations of the ratio "Vol1_s(d + 1) / Vol1_s(d)"

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here you are asked to check three aspects:
1/ that a formula is derived correctly,

2/ that a submitted Python program is OK, and
3/ that the results are given correctly for d=1 and 2.

You can give a total of 3 points.
Give 0 points if 0 aspects are treated correctly
Give 1 points if 1 aspects is treated correctly
Give 2 points if 2 aspects are treated correctly
Give 3 points if 3 aspects are treated correctly

An example of an answer that would receive full score is as follows:
1/ Discussion of formula:
The ratio of volumes of the (d+1)-dimensional unit hypercylinder to the d-dimensional unit hypersphere is

Vol1_cyl(d + 1) = 2 Vol1_s(d)

The ratio of volumes of the (d+1)-dimensional unit hypersphere to the (d+1)-dimensional unit hypercylinder is

Vol1_s(d+1) / Vol1_cyl(d + 1) = <Q>

This yields the formula in the assigment.

```
import math, random

Vol_3 = 4.0 / 3.0 * math.pi
Vol_2 = math.pi
Vol_1 = 2.0
dimension = 2
n_samples = 1000000
x = [0] * dimension
n_accept = 0
for sample in range(n_samples):
    k = random.randint(0, dimension - 1)
    x_old_k = x[k]
    x[k] += random.uniform(-1.0, 1.0)
    new_hyperradius_square = sum(a ** 2 for a in x)
    if new_hyperradius_square > 1.0:
        x[k] = x_old_k
    x_cyl = random.uniform(-1.0,1.0)
    radius_square = sum(a ** 2 for a in x) + x_cyl **2
    if radius_square < 1.0: n_accept += 1
    ratio = 2.0 * n_accept / float(n_samples)
if dimension == 1:
    print "ratio Vol1_s(2)/ Vol1_s(1)", ratio, Vol_2 / Vol_1
if dimension == 2:
    print "ratio Vol1_s(3)/ Vol1_s(2)", ratio, Vol_3 / Vol_2
```

3/ Results:
Result for d=1: 1.5713
Result for d=2: 1.3344

POINTS - DESCRIPTION
Give 0 points if 0 aspects are treated correctly
Give 1 points if 1 aspect is treated correctly
Give 2 points if 2 aspects are treated correctly
Give 3 points if 3 aspects are treated correctly

Score from your peers: **2.5**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

C

In this section you get our program into shape for computing the ratio of volumes Vol1_s(200)/Vol_s(199) to high precision, and even V_h(200), from a series of calculations. Finally you do some error estimations.

C1

Modify your program of section B (if necessary) to run in high dimensions: Each iteration should take a constant number of operations, independent of d:
1/ Don't compute the squared radius x[0]^2 + ... + x[d-1]^2 from scratch each time. Calculate it from the previous configuration!
2/ Don't compute x_0^2 + ... + x_{d - 1} ^2 + x_supp ^2 (that is needed to evaluate <Q>) from scratch each time. Calculate it from what you already have available.
Explain in a few words your strategy for programming this fast program,
Upload the program.
Compute Vol1_s(200) / Vol1_s(199)**,** and **compare** to the exact result from section A.

1. This is also done in code used in B2 as shown below:

```
radius_sq_new = radius_sq - (x[k] ** 2.0) + (x_new_k ** 2.0)
```

2. This is also done in code used in B3 as shown below:

```
if radius_sq + x_supp ** 2 < 1:
```

The program code is as below:

```
import random, pylab, math

def Vol1_s(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.0)

def markov_pi (N,d):
        x = [0.0] * d
        radius_sq = 0.0
        sum_Q = 0

        for i in range(n_trials):
                k = random.randint(0, d - 1)
                x_new_k = x[k] + random.uniform(-1.0, 1.0)
                radius_sq_new = radius_sq - (x[k] ** 2.0) + (x_new_k ** 2.0)

                if radius_sq_new < 1:
                        x[k] = x_new_k
                        radius_sq = radius_sq_new

                x_supp = random.uniform(-1.0, 1.0)
                if radius_sq + x_supp ** 2 < 1:
                        sum_Q +=1

        return (float(sum_Q)/float(N))

n_trials = 1000000
d = 200
Q_average = markov_pi (n_trials, d)

print d, 2 * Q_average, Vol1_s(d + 1) / Vol1_s(d)
```

d=200:
2*<Q> = 0.176556, Vol1_s(201) / Vol1_s(200) = 0.176584158635

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here you are asked to evaluate three aspects:

1/ There should be a short (10-30) word explanation of the strategy
2/ There should be a correct program.
Note that it is possible that your fellow student already presented the efficient program in section B.
3/ There should be the correct value for Vol1_s(200)/Vol1_s(199)

You can give a total of 3 points.
Give 0 points if 0 aspects are treated correctly

Give 1 points if 1 aspects is treated correctly
Give 2 points if 2 aspects are treated correctly
Give 3 points if 3 aspects are treated correctly

An example of an answer that would receive full score is as follows:
1/ In the program, we keep track of the square of the radius of the point,
and update this quantity by subtracting the value x[k] **2 and adding
the new value, if accepted
2/ Here is the program

```
import random, math

def volume_analytic(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.
0)

dimension = 199
n_samples = 100000
n_accept = 0
x = [0] * dimension
hyperradius_square = 0.0
for sample in range(n_samples):
    k = random.randint(0, dimension - 1)
    x_new_k = x[k] + random.uniform(-1.0, 1.0)
    new_hyperradius_square = hyperradius_square - x[k] ** 2 + x_new_k **
 2
    if new_hyperradius_square < 1.0:
        x[k] = x_new_k
        hyperradius_square = new_hyperradius_square
    if hyperradius_square + random.uniform(-1.0, 1.0) ** 2 < 1.0:
        n_accept += 1
ratio = 2.0 * n_accept / float(n_samples)
print ratio, volume_analytic(dimension + 1) / volume_analytic(dimension)
```

3/ The numerical value is 0.17574, while the exact value is 0.177023967696

POINTS - DESCRIPTION
Give 0 points if 0 aspects are treated correctly
Give 1 points if 1 aspects is treated correctly
Give 2 points if 2 aspects are treated correctly
Give 3 points if 3 aspects are treated correctly

Score from your peers: **3**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

C2

Now "play accordion": Modify your program from C1 so that it performs independent calculations for d=1, 2, 3,..., 199, each time computing Vol1_s(d+1) / Vol1_s(d). Each of these calculations should start at the origin x = [0] * d. Put these results together with the analytical result of Vol1_s(1) to obtain Vol1_s(d)

Explain in a few words your strategy for computing Vol1_s(d), and **upload the program.**

Test your program by computing Vol1_s(d=5) and compare with the exact result from section A. Then, GRAND FINALE, **Produce a plot** (graphics file) of Vol1_(d) as a function of d for d=2,....,200. **Communicate** your results for Vol1_s(5) and Vol1_s(200).

Strategy:

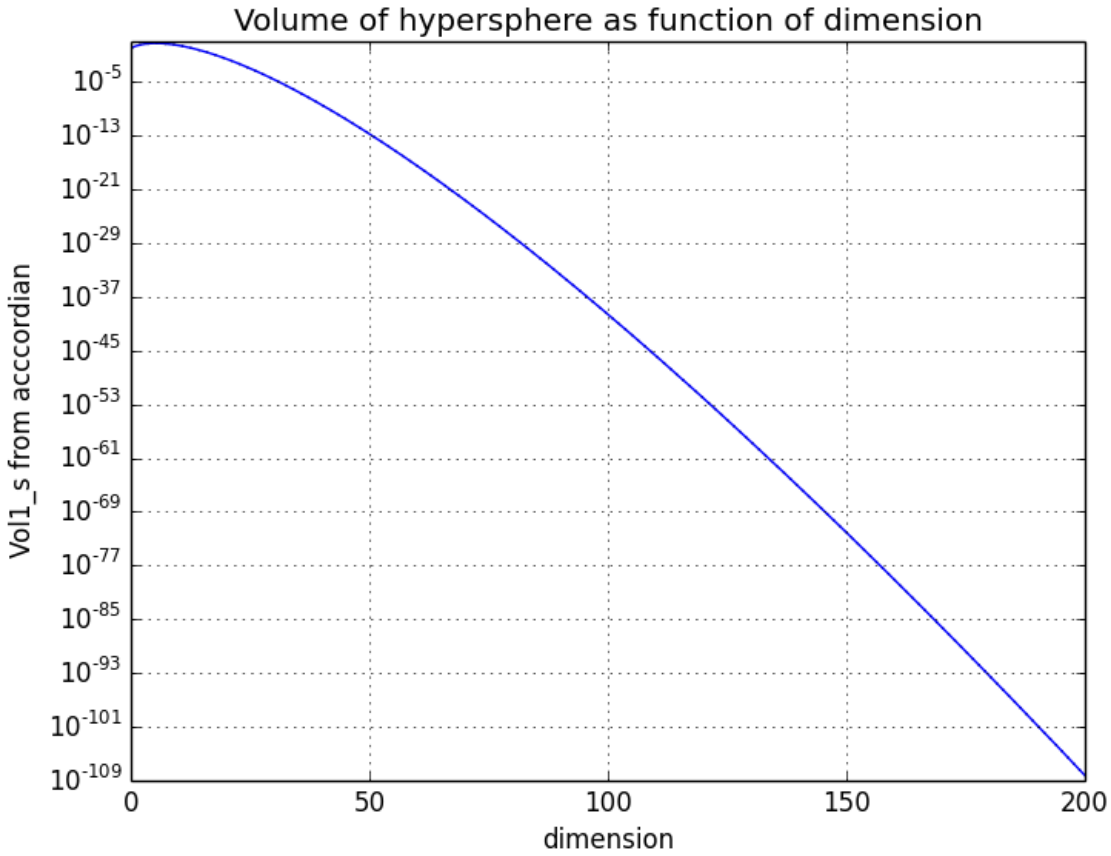
We know from analytical calculations that Vol1_s(1) is 2.0. We can run program from C1 at d=1 to obtain Vol1_s(2)/Vol1_s(1). Since we know Vol1_s(1), we can derive Vol1_s(2).

Next, we run for d=2 and obtain Vol1_s(3)/Vol1_s(2), and derive Vol1_s(3) since we know Vol1_s(2) from previous step. Repeating this for d=3, 4, 5, ... 199, we can obtain Vol1_s(4), Vol1_s(5), ... Vol1_s(200).

Program code:

```
import random, pylab, math

def Vol1_s_exact(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.0)

def markov_pi (N,d):
        x = [0.0] * d
        radius_sq = 0.0
        sum_Q = 0

        for i in range(n_trials):
                k = random.randint(0, d - 1)
                x_new_k = x[k] + random.uniform(-1.0, 1.0)
                radius_sq_new = radius_sq - (x[k] ** 2.0) + (x_new_k ** 2.0)

                if radius_sq_new < 1:
                        x[k] = x_new_k
                        radius_sq = radius_sq_new

                x_supp = random.uniform(-1.0, 1.0)
                if radius_sq + x_supp ** 2 < 1:
                        sum_Q +=1

        return (float(sum_Q)/float(N))

n_trials = 1000000
Vol1_s = []

# 1d volume is 1.0 and 2d volume is 2.0
Vol1_s.append(1.0)
Vol1_s.append(2.0)

for d in range(1, 200):
        Q_average = markov_pi (n_trials, d)
        print d, '/ 200 done'
        Vol1_s.append(Vol1_s[d] * 2 * Q_average)

print 'Vol1_s(5) estim = ', Vol1_s[5]
print 'Vol1_s(5) exact = ', Vol1_s_exact(5)

print 'Vol1_s(200) estim = ', Vol1_s[200]
print 'Vol1_s(200) exact = ', Vol1_s_exact(200)

pylab.plot(range(0,201), Vol1_s, ':k')
pylab.plot(range(0,201), Vol1_s)
pylab.xlabel('dimension')
pylab.ylabel('Vol1_s from acccordian')
pylab.yscale('log')
pylab.grid()
pylab.title('Volume of hypersphere as function of dimension')
pylab.savefig('Vol1_s_accordion.png')
```

```
pylab.show()
```



Estimated vs exact Vol1_s (d) for d=5 and d=200:

Vol1_s(5) estim = 5.27617575658
Vol1_s(5) exact = 5.26378901391

Vol1_s(200) estim = 5.62700548745e-109
Vol1_s(200) exact = 5.55883284203e-109

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here you are asked to evaluate three aspects:

1/ It should be explained that

Vol1_s(d) = Vol1_s(1) x Vol1_s(2)/Vol1_s(1) x ... x Vol1_s(d)/Vol1_s(d-1)

2/ There should be a program implementing this strategy
3/ The results for d=5 and d=200 should be given and there should be a plot.

You can give 3 points
Give 0 points if 0 aspects are treated correctly
Give 1 points if 1 aspects is treated correctly
Give 2 points if 2 aspects are treated correctly
Give 3 points if 3 aspects are treated correctly

Below an answer that would received full score.

1/ Clearly;

Vol1_s(d) = Vol1_s(1) x Vol1_s(2)/Vol1_s(1) x ... x Vol1_s(d)/Vol1_s(d-1)

2/ Here is the program

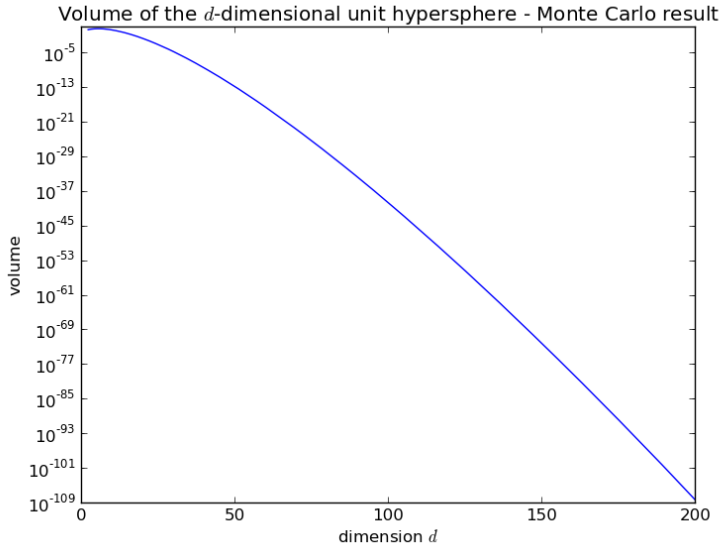
```
import random, math, pylab

def volume_analytic(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.
0)

dimension_max = 200
n_samples = 100000
volume = 2.0
x_data = []
y_data = []
for dimension in xrange(1, dimension_max):
    n_accept = 0
    x = [0] * dimension
    hyperradius_square = 0.0
    for sample in xrange(n_samples):
        k = random.randint(0, dimension - 1)
        x_new_k = x[k] + random.uniform(-1.0, 1.0)
        new_hyperradius_square = hyperradius_square - \
                            x[k] ** 2 + x_new_k ** 2
        if new_hyperradius_square < 1.0:
            x[k] = x_new_k
            hyperradius_square = new_hyperradius_square
        if hyperradius_square + random.uniform(-1.0, 1.0) ** 2 < 1.0:
            n_accept += 1
    volume *= 2.0 * n_accept / float(n_samples)
    print dimension + 1 , volume , volume_analytic(dimension + 1)
    x_data.append(dimension + 1)
    y_data.append(volume)
pylab.plot(x_data, y_data)
pylab.xlabel('dimension $d$')
pylab.ylabel('volume')
pylab.title('Volume of the $d$-dimensional unit hypersphere - Monte Carl
o result')
pylab.yscale('log')
pylab.savefig('vol_unit_hypersphere_MC.png')
pylab.show()
```

3/ For n_trials = 100000

d=5 MC = 5 5.29386127935 Exact = 5.26378901391

d=200 MC = 6.11130861137e-109 Exact = 5.55883284203e-109

Here is the plot:



x

This plot is very similar to the exact solution...Note that we did not ask you to compare the two.

POINTS - DESCRIPTION
Give 0 points if 0 aspects are treated correctly
Give 1 points if 1 aspect is treated correctly
Give 2 points if 2 aspects are treated correctly
Give 3 points if 3 aspects are treated correctly

Score from your peers: **3**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

C3 As Monte Carlo calculators you should control your error. To save time, you will do this for d=20.

Compute the Monte Carlo estimate for Vol1_s(20) twenty times, and determine the mean value <Vol1_s(20)>, the mean square <Vol1_s(20)^2> and the Error = \sqrt{ <Vol1_s(20)^2> - <Vol1_s(20)>^2> } / sqrt{20}. This is your final result, and it only depends on n_trials.

Produce the following table:
n_trials | <Vol1_s(20)> | Error | Vol1_s(20) (exact result) |
1 | ...
10 | ...
100
1000
10000
100000

Is the error bar consistent with the exact result for very short Markov-chain calculations? Comment on strategies of how to use a better starting point for the Markov chain calculation in dimension d, given that

you already computed in dimension d, but there is no need to implement them in this homework session.

Congrats! Chapeau! Alle Achtung! Bravissimo if you got this far, or anything near!

The code is as follows:

```
import random, math

def Vol1_s_exact(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.0)

def markov_pi (N,d):
        x = [0.0] * d
        radius_sq = 0.0
        sum_Q = 0

        for i in range(n_trials):
                k = random.randint(0, d - 1)
                x_new_k = x[k] + random.uniform(-1.0, 1.0)
                radius_sq_new = radius_sq - (x[k] ** 2.0) + (x_new_k ** 2.0)

                if radius_sq_new < 1:
                        x[k] = x_new_k
                        radius_sq = radius_sq_new

                x_supp = random.uniform(-1.0, 1.0)
                if radius_sq + x_supp ** 2 < 1:
                        sum_Q +=1

        return (float(sum_Q)/float(N))

n_trials_list = [1, 10, 100, 1000, 10000, 100000, 1000000]

f = open('Monte_Carlo_errors.txt', 'w')
f.write('n_trials | <Vol1_s(20)> |  Error  | Vol1_s(20) (exact result)\n')
f.flush()

for n_trials in n_trials_list:
        print 'n_trials =', n_trials, 'starting'

        sum_Vol1_s_20 = 0.0
        sum_Vol1_s_20_sq = 0.0

        for run in range(20):
                Vol1_s = []

                # 1d volume is 1.0 and 2d volume is 2.0
                Vol1_s.append(1.0)
                Vol1_s.append(2.0)

                for d in range(1, 20):
                        Q_average = markov_pi (n_trials, d)
                        Vol1_s.append(Vol1_s[d] * 2 * Q_average)

                sum_Vol1_s_20 += Vol1_s[20]
                sum_Vol1_s_20_sq += Vol1_s[20] ** 2.0
```

```
                print 'run = ', run+1, '/ 20 done'

        mean_Vol1_s_20 = sum_Vol1_s_20/20.0
        mean_Vol1_s_20_sq = sum_Vol1_s_20_sq/20.0

        error_Vol1_s_20 = math.sqrt( (mean_Vol1_s_20_sq - (mean_Vol1_s_20 ** 2.0))/20.0
)

        f.write(str(n_trials)+'\t'+str(mean_Vol1_s_20)+'\t'+str(error_Vol1_s_20)+'\t')
        f.write(str(Vol1_s_exact(20))+'\n')
        f.flush()
        print 'n_trials =', n_trials, 'done\n'
```

The generated table is as below:

```
n_trials | <Vol1_s(20)>    |   Error            |  Vol1_s(20) (exact result)
1          52428.8             51101.2731345       0.02580689139
10         3.60864674961       1.23483074673       0.02580689139
100        0.0506850833212     0.0120034727487     0.02580689139
1000       0.0254197965806     0.00147171750605    0.02580689139
10000      0.0264757319055     0.000400377190583   0.02580689139
1000000    0.0258479272294     4.10937358498e-05   0.02580689139
```

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

In this final part, you are asked to evaluate that the error calculation has been done correctly. There are two aspects:

1/ Your fellow student should correctly evaluate the error, and produce a table similar to the one shown below

2/ Your fellow student should realize that there is a systematic error associated to the initial condition. Maybe he/she will remark that it would be better if one constructed the starting configuration at dimension d+1 from the final configuration at dimension d (In fact, one should create it, by trial and error, from the configuration in the (d+1)-dimensional hypercylinder.

You can give 2 points
Give 0 points if 0 aspects are treated correctly
Give 2 points if any of the aspects 1/ and 2/ has been treated with some care.

Here, for your information, is a program that computed the error

```
import random, math

def volume_analytic(dimension):
    return math.pi ** (dimension / 2.0)/ math.gamma(dimension / 2.0 + 1.
0)

dimension_max = 20
n_samples = 1
for grand_iter in range(20):
    data = []
    data_sq = []
    for iter in range(5):
        volume = 2.0
        for dimension in range(1, dimension_max):
            n_accept = 0
            x = [0] * dimension
            hyperradius_square = 0.0
            for sample in xrange(n_samples):
                k = random.randint(0, dimension - 1)
                x_new_k = x[k] + random.uniform(-1.0, 1.0)
                new_hyperradius_square = hyperradius_square - \
                                x[k] ** 2 + x_new_k ** 2
                if new_hyperradius_square < 1.0:
                    x[k] = x_new_k
                    hyperradius_square = new_hyperradius_square
                if hyperradius_square + random.uniform(-1.0, 1.0) ** 2 <
 1.0:
                    n_accept += 1
            volume *= 2.0 * n_accept / float(n_samples)
        data.append(volume)
        data_sq.append(volume ** 2)
#       print dimension + 1 , volume , volume_analytic(dimension + 1)
    mean_volume = sum(data)/len(data)
    mean_volume_sq = sum(data_sq)/len(data)
    error = math.sqrt(mean_volume_sq - mean_volume ** 2)/ \
                        math.sqrt(float(len(data)))
    print n_samples, mean_volume, error, volume_analytic(dimension_max)
    n_samples *= 10
```

but note that the students were not asked to upload this program.

Below an answer that would received full score.

TABLE:

```
    1 0.0          0.0          0.02580689139 (output not well defined)
   10 3.49157635456 1.67071516567 0.02580689139
  100 0.0500323383413 0.0170032912322 0.02580689139
 1000 0.0295954736964 0.0039847272968 0.02580689139
10000 0.0261182324108 0.00125900744043 0.02580689139
100000 0.0259585826301 0.000277723163221 0.02580689139
```

DISCUSSION:
2/ There is a systematic error associated to the initial condition, and
for small n_trials, the error estimate is inconsistent with the
exact result.
A better starting configuration for d+1 can be constructed from the
final configuration at dimension d.

POINTS - DESCRIPTION
Give 0 points if 0 aspects are treated correctly
Give 2 points if the table and/or the discussion has been done.

NB: Note that **this is VERY DIFFICULT material. So give 2 points if there is some reasonable understanding of errors.**

Score from your peers: **2**

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

Overall evaluation/feedback

Note: this section can only be filled out during the evaluation phase.

peer 1 → Very nice work !

peer 2 → *[This area was left blank by the evaluator.]*