Peer Assessments (https://class.coursera.org/smac-001/human_grading/)
/ Homework Session 1: From the one-half rule to the bunching method
Help (https://class.coursera.org/smac-001/help/peergrading?url=https%3A%2F%2Fclass.coursera.org%2Fsmac-001%2Fhuman_grading%2Fview%2Fcourses%2F971628%2Fassessments%2F4%2Fresults%2Fmine)

Submission Phase

1. Do assignment ☑ (/smac-001/human_grading/view/courses/971628/assessments/4/submissions)

Evaluation Phase

2. Evaluate peers ☑ (/smac-001/human_grading/view/courses/971628/assessments/4/peerGradingSets)

Results Phase

3. See results ☑ (/smac-001/human_grading/view/courses/971628/assessments/4/results/mine)

Your effective grade is 13

Your unadjusted grade is 13, which is simply the grade you received from your peers.

See below for details.

Introduction

This homework is divided into three parts. **In the first part**, you estimate π by using a **direct-sampling algorithm** and determine the statistical error of your numerical calculation. **In the second part**, you again estimate π, but this time for a **Markov-chain algorithm**. The precision of the result depends on the rejection rate of the algorithm, which itself depends on the step size δ. You will see that performance is best when δ is chosen such that about half of the proposed moves are accepted. This is called the "**1/2 rule**". It is an approximate rule, **a rule of thumb**, not a mathematical law. Finally, **in the third part**, you quantify the error associated with the Markov-chain algorithm using the **bunching algorithm**.

You will create plots to analyze the output of your programs graphically. We encourage your use of pylab (inside Python) for graphics output, but if you have problems with pylab, you may use other graphics programs to render your results. Please take special care that the axes in your plots are labeled correctly and that they show the data that is asked for.

(A) Direct-sampling algorithm
(A1)
Run the direct-sampling algorithm "direct_pi_multirun.py", provided on the Coursera website and convince yourself that for large enough n_trials, the output of each of the n_runs runs converges to π

=3.1415926.... Determine the root mean square (rms) error, given by:

$$\text{rms error} = \sqrt{\frac{1}{\text{n\_runs}} \sum_{i=0}^{\text{n\_runs}-1} (\pi_i^{est} - \pi)^2},$$

where π^est is the output of each single run. Note that this gives the actual precision of your calculation, as you compare with the exact mathematical value of π. Take n_runs=500 and modify "direct_pi_multirun.py" in order to compute the rms error according to the above formula for n_trials = 2^4, 2^5,...,2^12. Produce a log-log-plot of the error as a function of n_trials.

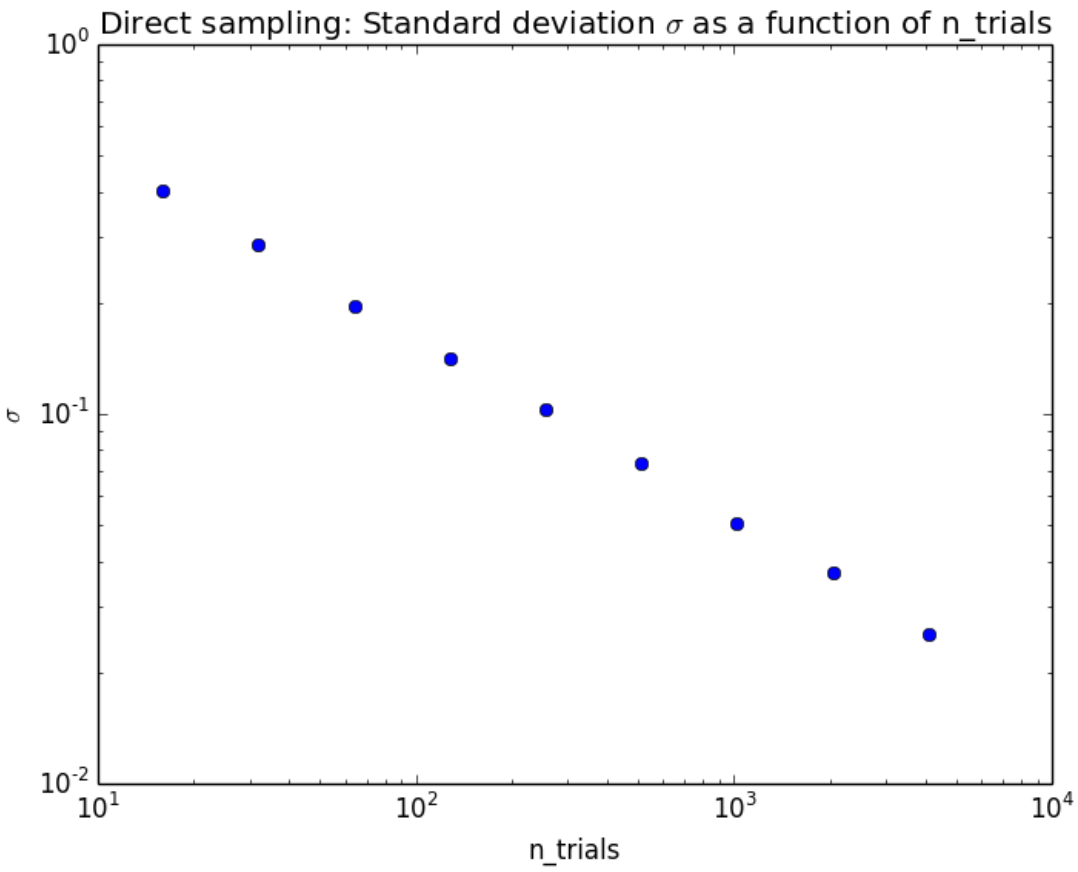
Below, we provide the code of the modified program (this is a special goodie of week 1). **Cut and paste it into a file**, then run it. Submit the plot as a graphics file (pdf, png or jpeg format). This is quite easy, once you have Python installed on your machine!

```
import random, math, pylab

def direct_pi(N):
    n_hits = 0
    for i in range(N):
        x, y = random.uniform(-1.0, 1.0), random.uniform(-1.0, 1.0)
        if x ** 2 + y ** 2 < 1.0:
            n_hits += 1
    return n_hits

n_runs = 500
n_trials_list = []
sigmas = []
for poweroftwo in range(4, 13):
    n_trials = 2 ** poweroftwo
    sigma = 0.0
    for run in range(n_runs):
        pi_est = 4.0 * direct_pi(n_trials) / float(n_trials)
        sigma += (pi_est - math.pi) ** 2
    sigmas.append(math.sqrt(sigma/(n_runs)))
    n_trials_list.append(n_trials)

pylab.plot(n_trials_list, sigmas, 'o')
pylab.gca().set_xscale('log')
pylab.gca().set_yscale('log')
pylab.xlabel('n_trials')
pylab.ylabel('$\sigma$')
pylab.title('Direct sampling: Standard deviation $\sigma$ as a function of n_trials')
pylab.savefig('direct_sampling_statistical_error.png')
pylab.show()
```



Evaluation/feedback on the above work

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

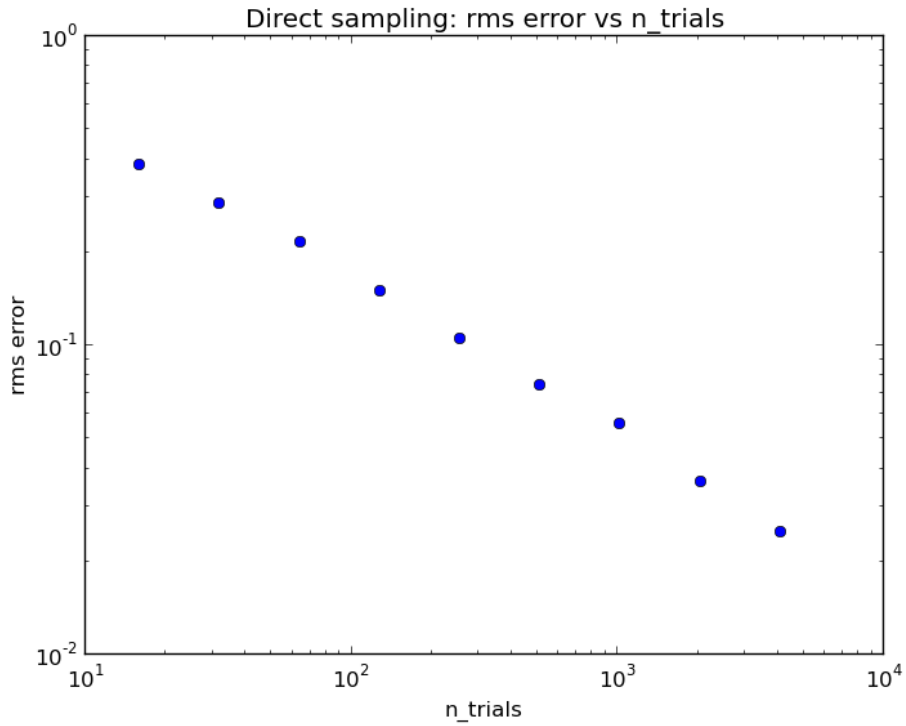
Here, you are asked to evaluate a scientific diagram. You can give a total of **2 points**.

Give **1 point** if the diagram meets the following criteria of graphical quality (which will be the general criteria for the quality of diagrams in this course):

- The diagram shows exactly the quantities that were asked for (here: "rms error" and "n_trials")
- All axes are correctly labeled (here: "rms error" and "n_trials". Equivalent labels like "ERROR" or "The error" or "NTRIALS" or "ntrials" etc are also acceptable.)
- All axes have the correct scaling (here: both x-axis and y-axis are logarithmic).
- The diagram shows at least the range of data points that was asked for (here: n_trials=2^4, 2^5,...,2^12).

Give **1 point** if the diagram represents the correct results: the plot will show an approximately straight line in the log-log plot.

If the plot meets the criteria of graphical quality and shows the correct results, the student earns a total of **2 points**. An example that would receive the full score is shown here:



Score from your peers: **2**

(A2)
By referring to your figure, show that the rms error is proportional to n_trials^(-α) and estimate α.
Enter the empirical value of α in the below box with a precision of 1 digit after the decimal point. Further explanation of your result is not necessary. Example: If you determined α = 0.8783, simply write "0.9" into the editor box below.

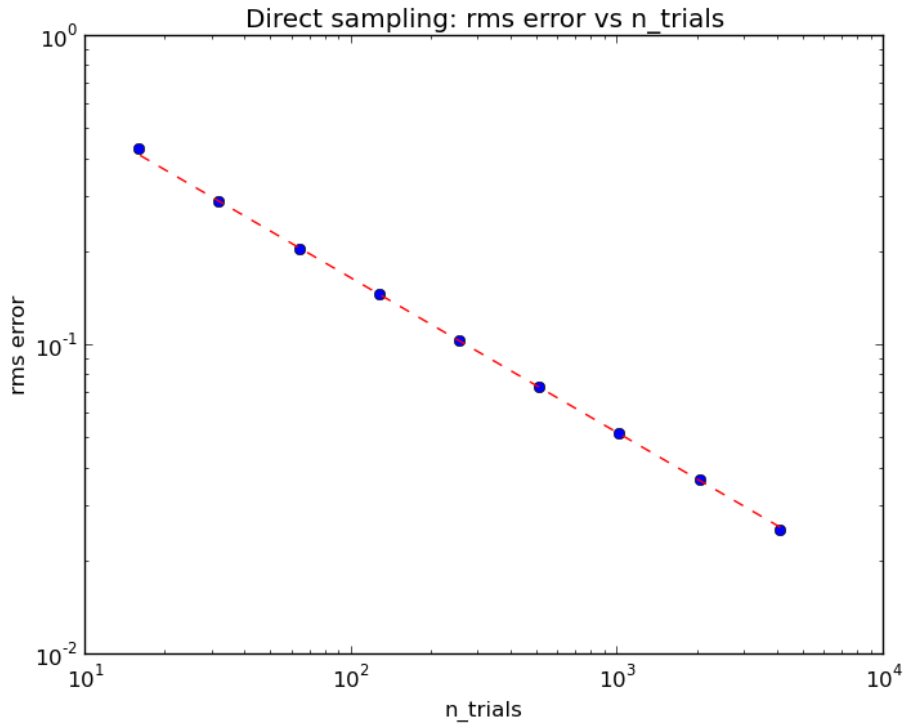
0.5

Evaluation/feedback on the above work

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

In this question, the students were asked to determine a number, the exponent α. Give **1 point** for the correct answer.

The correct answer to this question is 0.5 as can be readily determined from the diagram that was created in the solution of (A1):

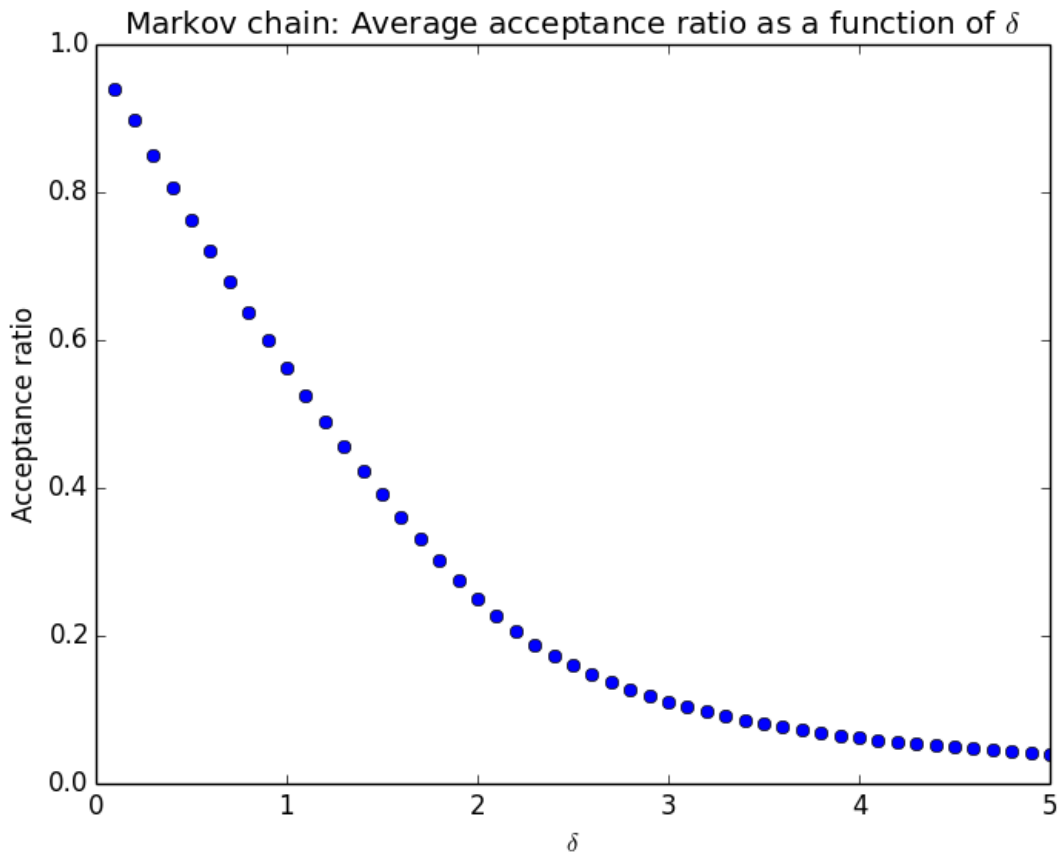


The red dashed line corresponds to rms_error ∝ n_trials^(-1/2).

Score from your peers: **1**

(B) Markov chain algorithm and the "1/2 rule of thumb"

(B1) Run the Markov-chain algorithm "markov_pi_multirun.py", provided on the Coursera website and **convince yourself that for large enough n_trials, the output of each of the n_runs runs approaches the value of** π. Then, set the number of runs to n_runs = 500 and the number of trials to n_trials = 1000. **Modify the program** so that it calculates the **acceptance ratio**: the number of the moves that are not rejected, divided by n_trials. Run the new program and compute the acceptance ratio for the following values of δ ("delta" in "markov_pi_multirun.py"): δ=0.1, δ=0.2, δ=0.3,..., δ=5.0 . Plot the computed **acceptance ratio** as a function of δ and submit the plot as a graphics file (pdf, png or jpeg format). Use **linear scaling** on both axes.



Evaluation/feedback on the above work

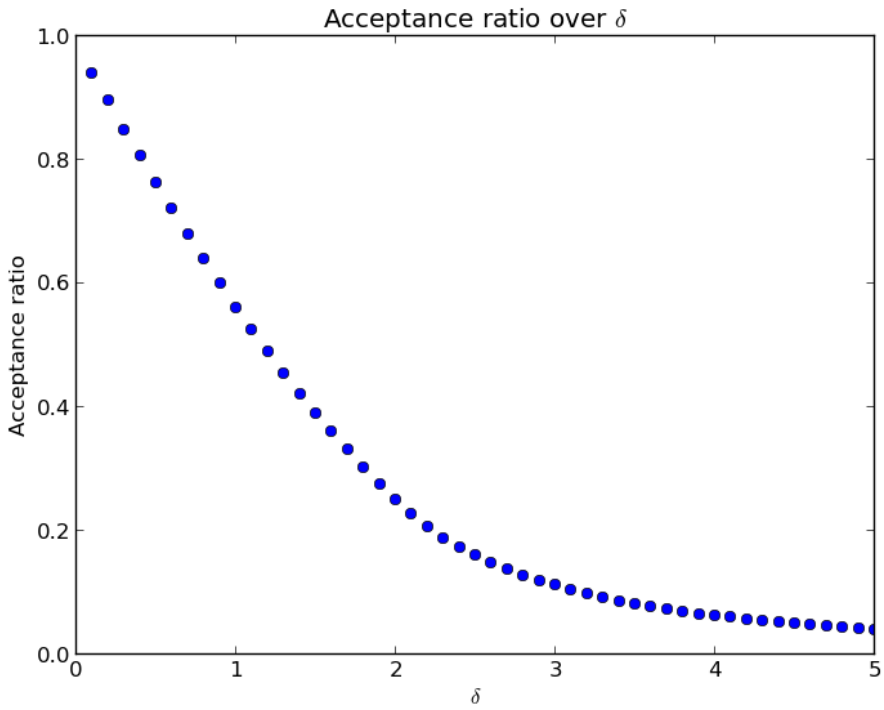
Note: this section can only be filled out during the evaluation phase.

Here, you are asked to evaluate a scientific diagram. You can give a total of **2 points.**

Give **1 point**, if the diagram meets the following criteria of graphical quality:

- The diagram shows exactly the quantities that were asked for (here: the acceptance ratio and δ)
- All axes are correctly labelled (here: "Acceptance ratio" and "δ". Equivalent labels like "p_{acc}" for the acceptance ratio or "delta" for "δ" are also acceptable.)
- All axes have the correct scaling (here: both x-axis and y-axis are linear)
- The diagram shows at least the range of data points that was asked for (here: $\delta = 0.1, \delta = 0.2, \delta = 0.3, \ldots, \delta = 5.0$)

Give **1 point**, if the diagram represents the correct results: the plot will show the behavior shown in the figure below.



Score from your peers: **2**

(B2)
The "**1/2 thumb rule**" predicts that the best performance of your Markov Chain Monte Carlo algorithm is for an acceptance ratio of approximately 1/2: half the moves are accepted, and half of them are rejected.

In (B1) you computed the acceptance ratios for δ=0.1,δ=0.2,δ=0.3,... , δ=5.0 (Note that δ can be larger than the box!!)
Find the δ-interval in which the acceptance ratio equals 1/2.
Example: If the interval where you expect the acceptance ratio of 1/2 is between 3.5 and 3.6, you write "3.5-3.6". Intervals significantly larger than 0.1, like "3.0-4.0" are considered as wrong results.

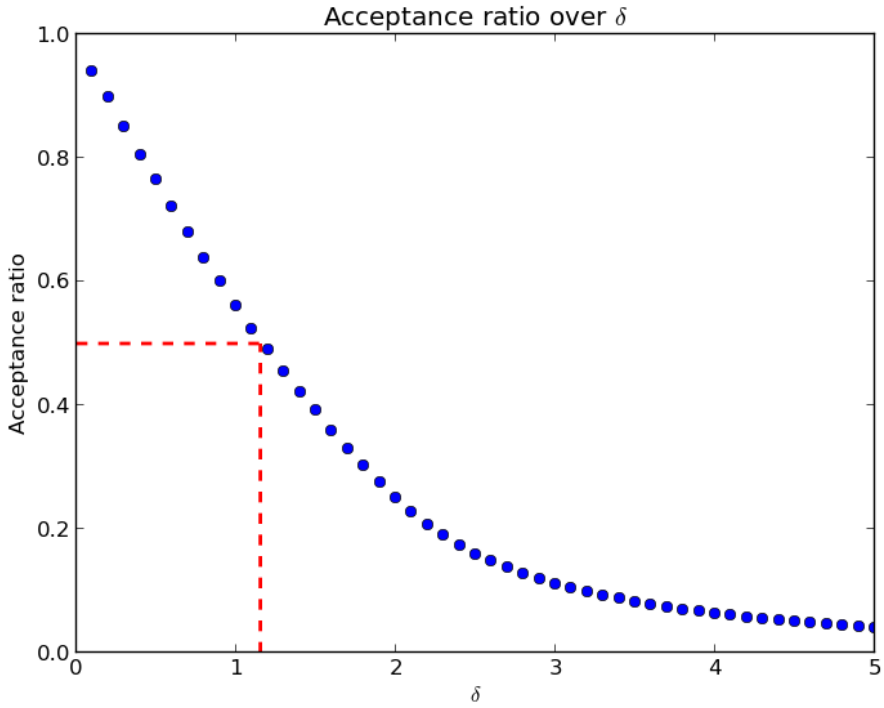
1.1-1.2

Evaluation/feedback on the above work

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

In this question, the students were asked to determine an interval, the interval of δ-values between which the acceptance ratio equals 1/2. Give **1 point** for the correct answer.

The correct answer to this question is 1.1-1.2 as can be readily determined from the numerical output or from the diagram that was created in the solution of (B1):



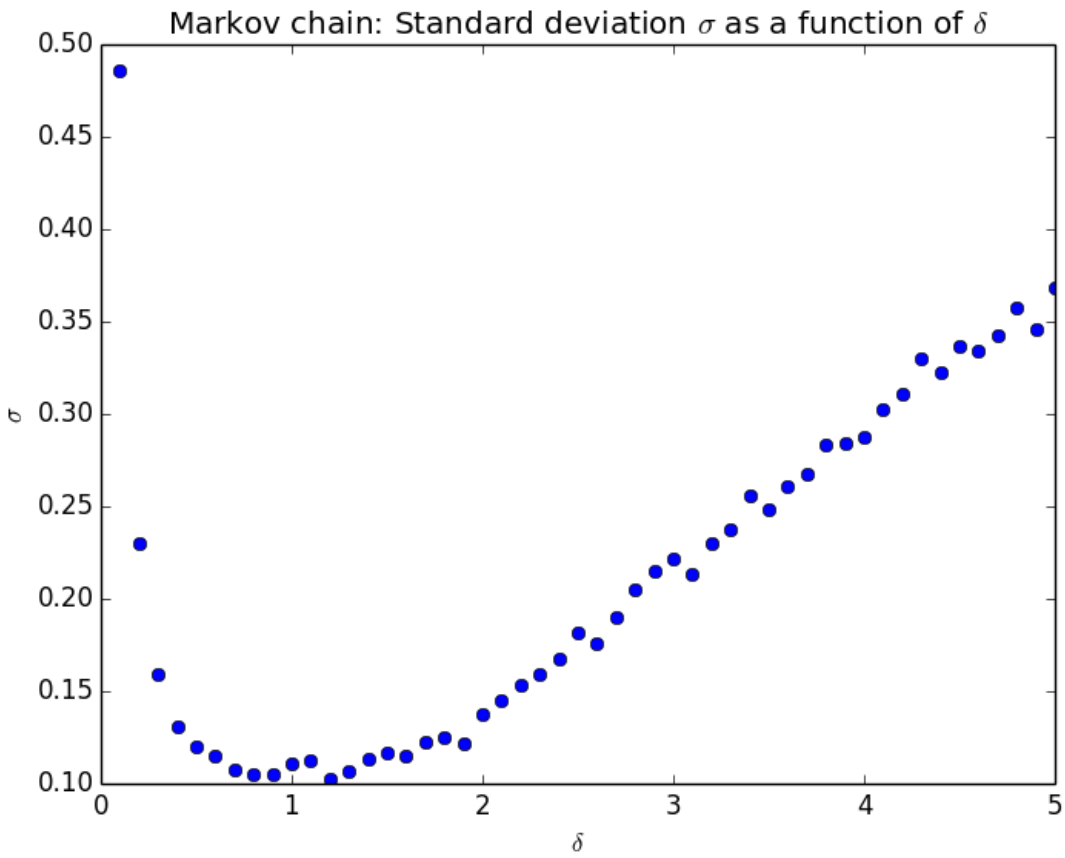
The δ-value corresponding to an acceptance ratio of 1/2 lies between 1.1 and 1.2. In order to give a more precise value, one could decrease the increment of δ from 0.1 to lower values. Although this is beyond the scope of this exercise, you should feel encouraged to try this out.

Score from your peers: **1**

(B3)
Now study the performance of "markov_pi_multirun.py" as a function of δ. Again, use n_runs = 500 and n_trials = 1000.
Modify the program so that it computes the rms error, exactly as we did for the direct sampling algorithm, using again the formula

$$\text{rms error} = \sqrt{\frac{1}{\text{n\_runs}} \sum_{i=0}^{\text{n\_runs}-1} (\pi_i^{est} - \pi)^2}$$

for values of δ=0.1,δ=0.2, δ=0.3,.... ,δ=5.0 .
Plot the rms error as a function of δ and submit the plot as a graphics file (pdf, png or jpeg format). Note that the rms error gives the actual precision of your calculation, as you compare with the exact mathematical value of π.



Evaluation/feedback on the above work

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

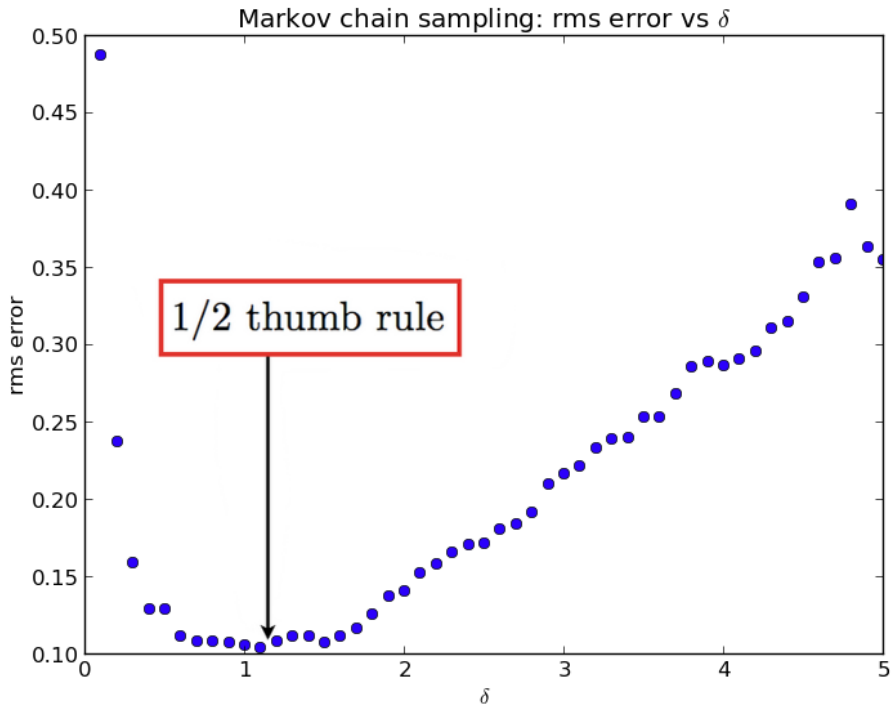
Here, you are asked to evaluate a scientific diagram. You can give a total of **2 points.**

Give **1 point** if the diagram meets the following criteria of graphical quality:

- The diagram shows exactly the quantities that were asked for (here: the standard deviation and δ)
- All axes are correctly labelled (here: "rms error" and "δ". Equivalent labels like "error" for the rms error or "delta" for "δ" are also acceptable.
- All axes have the correct scaling (here: both x-axis and y-axis are linear)
- The diagram shows at least the range of data points that was asked for (here: δ=0.1, δ=0.2, δ=0.3, ..., δ=5.0).

Give **1 point** if the diagram represents the correct behavior. If the Python program has been modified correctly, the standard deviation curve will have a clear minimum around δ=1.1 - 1.2 (the value that we estimated in (B2)).

An example diagram that would receive full score is shown below. Note, that the label "1/2 thumb rule" and the arrow pointing to the minimum are not mandatory and have been added here only to clarify our explanation.



Score from your peers: **2**

(B4)
Comment shortly

1. whether the acceptance rate 1/2 indeed gives better accuracy than other choices, by referring to your figures, and compare it to the performance of the algorithm for other values of δ.
2. why $\delta \approx 0$ and $\delta \to \infty$ give worse results than the intermediate value of δ that corresponds to the 1/2 rule.

1. Acceptance rate is 1/2 for δ=1.1-1.2. The rms is lower for δ=1.1-1.2 than other values of δ.

2. $\delta \approx 0$, the changes are too small. $\delta \to \infty$, the changes are too large. In first case, almost all moves are accepted (resulting in small changes per move) and in second case, almost all moves are rejected (a big change happens once in a while). In both cases, the position x,y evolves very slowly. We have sampling issues resulting in worse results than for intermediate δ values, where this issue is less pronounced.

Evaluation/feedback on the above work

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

To answer question 1:

- The student is required to relate the two diagrams, rms error versus δ and acceptance ratio versus δ, that have been used in parts (B1) to (B3). The student is expected to explain that the minimum of the rms error, i.e. the highest accuracy, is achieved at a δ that corresponds approximately to an acceptance ratio of 1/2. It can be readily seen that the mimimum is indeed found close to the δ that was obtained in part (a). If this is explained correctly in the student's reply, please evaluate the reply with **1 point**, if not please give **0 points**.

To answer question 2:

- The student should explain that for very small δ, the heliport square is swept out very slowly because the stepsize is so small. On the other hand, for very large δ, the heliport square is again swept out very slowly because the acceptance ratio is so small. If this is explained correctly in the student's reply, please evaluate the reply with **1 point**, if not please give **0 points**.

If both questions are answered correctly, give a total of **2 points**, and if one of the two questions is answered correctly, give a total of **1 point**.

An acceptable answer would address the two aspects of the question, for example:

1. The value of δ obtained using the acceptance ratio 0.5 is close to the minimum of the error as a function of δ.
2. For very small δ, the heliport square is swept out very slowly because the stepsize is so small. On the other hand, for very large δ, the heliport square is again swept out very slowly because the acceptance ratio is so small.

Score from your peers: **2**

(C) The bunching algorithm

Real Markov-chain simulations present two differences with respect to what we presented in section B:

1. It is not possible to compare the numerical data to the exact result, here π = 3.1415..., as we simply do not know the result. In the rms error, where we up to now compared the output of our calculation with the exact value π = 3.1415..., we must therefore replace the mathematical value π by an estimate.
2. it is usually not possible to have many independent runs (it is usually not possible to put n_runs >> 1 in our example) that each evolve for very long times (it is usually not possible to set n_trials =1000 in our example). One needs to estimate the error from a single run.

For independent data x_0, x_1, x_2....., x_(n_trials -1), as those produced by "direct_pi.py", this is easy, and the error is given by

$$\text{error} = \frac{\sqrt{\langle x^2\rangle - \langle x\rangle^2}}{\sqrt{\text{n\_trials}}},$$

where

$$\langle x^2\rangle = \sum_i x_i^2 / \text{ n\_trials}$$

and

$$\langle x\rangle = \sum_i x_i / \text{n\_trials}.$$

Note that in the above error formula we no longer compare with the exact value of π (that we do not

know), and that the x_i, in our example are either 0 (in the square, outside the circle) or 4 (inside the circle).

For independent data, we expect that with about 68% probability the expectation value of x, that is, in our example π = 3.1415, is in the interval

$$[\langle x\rangle - \text{error}, \langle x\rangle + \text{error}].$$

(C1) Convince yourself that this "naive error" formula

$$\text{naive error} = \frac{\sqrt{\langle x^2\rangle - \langle x\rangle^2}}{\sqrt{\text{n\_trials}}}$$

CANNOT be used for Markov-chain data x_0, x_1, x_2....., x_(n_trials -1). To this end, use the following program (with n_trials = 2 ** 14
delta = 0.1, n_parties = 100), to estimate the probability with which the true value of π = 3.1415... lies inside the "naive error interval"

$$[\langle x\rangle - \text{naive error}, \langle x\rangle + \text{naive error}].$$

```
import random, math

def markov_pi_all_data(N, delta):
    x, y = 1.0, 1.0
    data_sum = 0.0
    data_sum_sq = 0.0
    for i in range(N):
        del_x, del_y = random.uniform(-delta, delta), random.uniform(-delta, delta)
        if abs(x + del_x) < 1.0 and abs(y + del_y) < 1.0:
            x, y = x + del_x, y + del_y
        if x ** 2 + y ** 2 < 1.0:
            data_sum += 4.0
            data_sum_sq += 4.0 ** 2
    return data_sum / float(N), data_sum_sq / float(N)

n_trials = 2 ** 14
delta = 0.1
n_parties = 100
inside_error_bar = 0
for iteration in range(n_parties):
    mean, mean_square = markov_pi_all_data(n_trials, delta)
    naive_error = math.sqrt(mean_square  - mean ** 2) / math.sqrt(n_trials)
    error =  abs(mean - math.pi)
    if error < naive_error: inside_error_bar += 1
    print mean, error, naive_error
print inside_error_bar / float(n_parties),'fraction: error bar including pi'
```

Cut and paste this program into a file, then run it.

How large is the probability with which the true value of π = 3.1415... lies inside the "naive error interval", approximately for the above program with n_trials = 2^14 and delta = 0.1?

1. π = 3.1415... lies in the naive error interval with 100% probability, because it corresponds to the Gaussian error, that is 100% correct
2. π = 3.1415... lies in the naive error interval with approximately 68% probability, corresponding to one

standard deviation.

3. π = 3.1415... lies in the naive error interval with approximately 10% probability.
4. π = 3.1415... never lies inside the naive error interval, because Markov-chain data simply differ from independent data.

NB: Note that we provide the program for convenience, and given that we are just starting in this course.

3

Evaluation/feedback on the above work

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

The correct answer is answer 3.

Give **1 point** if the diagram if this answer is provided. Otherwise give 0 points.

Score from your peers: **1**

(C2) In order to improve the error estimation for the Markov chain, the "bunching algorithm" does use the independent error analysis for the Markov-chain data x_0, x_1, x_2....., x_(n_trials -1), but then analyzes these data again, using the independent error analysis, after grouping them into pairs

(x_0 + x_1)/2, (x_2 + x_3)/2, (x_4 + x_5)/2, ... (x_(n_trials -2) + x_(n_trials -1))/2

This process is then repeated for several iterations, making pairs of pairs, then pairs of pairs of pairs, etc. This very successful algorithm is implemented for a long sequence of data produced by markov_pi.py (x_i = 4 if it corresponds to a hit, and x_i = 0 otherwise) in the below program, that is again provided for convenience. The program produces a plot of error against iteration.

```
import random, pylab, math

def markov_pi_all_data(N, delta):
    x, y = 1.0, 1.0
    data = []
    for i in range(N):
        del_x, del_y = random.uniform(-delta, delta), random.uniform(-delta, delta)
        if abs(x + del_x) < 1.0 and abs(y + del_y) < 1.0:
            x, y = x + del_x, y + del_y
        if x ** 2 + y ** 2 < 1.0:
            data.append(4.0)
        else:
            data.append(0.0)
    return data

poweroftwo = 14
n_trials = 2 ** poweroftwo
delta = 0.1
data = markov_pi_all_data(n_trials, delta)
errors  = []
bunches = []
for i in range(poweroftwo):
    new_data = []
    mean = 0.0
    mean_sq = 0.0
    N = len(data)
    while data != []:
        x = data.pop()
        y = data.pop()
        mean += x + y
        mean_sq += x ** 2 + y ** 2
        new_data.append((x + y) / 2.0 )
    errors.append(math.sqrt(mean_sq / N - (mean / N) ** 2) / math.sqrt(N))
    bunches.append(i)
    data = new_data[:]
pylab.plot(bunches, errors, 'o')
pylab.xlabel('iteration')
pylab.ylabel('naive error')
pylab.title('Bunching: naive error vs iteration number')
pylab.savefig('apparent_error_bunching.png', format='PNG')
pylab.show()
```

The observed error is found to increase with the iterations, and exhibits a plateau. This plateau is an excellent estimation of the true error of Markov-chain output. Run this program and answer three questions:

1. Which point on the plot produced by the program of this section corresponds to the naive error of section C1?
2. Why does the error initially increase with the iterations?
3. Why is there a plateau, at all?

1. Iteration=1

2. Data remains correlated for the initial bunching iterations. However, the correlation begins to decrease. The error underestimation starts to therfore get lowered leading to increase in the error.

3. After a certain number of bunching iterations, the data becomes uncorrelated and therefore, the true error gets calculated. This leads to a plateau since the error becomes constant for the next rounds of bunching iterations. However, soon, the number of data points starts to become small. We then have sampling problem and the error is no more the true error.

Evaluation/feedback on the above work

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

To answer question 1:

- The student should show that, at iteration = 0 (that is for the first point on the left), the naive error is evaluated, as in section **C1**. If this is explained correctly in the student's reply, please evaluate the reply with **1 point**, if not please give **0 points**

To answer question 2:

- The student should argue that the bunching makes the data less correlated, and the underestimation of the correct error less severe. If this is explained correctly in the student's reply, please evaluate the reply with **1 point**, if not please give **0 points**.

To answer question 3:

- The student should argue that after a given number of iterations, the bunching does not make the data less correlated, as they are already uncorrelated. If this is explained correctly in the student's reply, please evaluate the reply with **1 point**, if not please give **0 points**.

Give a total of **1, 2 or 3 points**, corresponding to the number of correct answers.

An acceptable answer can be short. It would address the three aspects of the question, for example:

1. The first point to the left
2. Larger bunches are less correlated, so the independent error is better.
3. Bunching has no significant effect on uncorrelated data, as those corresponding to large bunches.

Score from your peers: **2**