(http://help.coursera.org/customer/widget/emails/new)

Peer Assessments (https://class.coursera.org/smac-001/human_grading/)
/ Homework session 6: Path sampling: A firework of algorithms of
Help (https://class.coursera.org/smac-001/help/peergrading?url=https%3A%2F%2Fclass.coursera.org%2Fsmac-001%2Fhuman_grading%2Fview%2Fcourses%2F971628%2Fassessments%2F10%2Fresults%2Fmine)

Submission Phase

1. Do assignment ☑ (/smac-001/human_grading/view/courses/971628/assessments/10/submissions)

Evaluation Phase

2. Evaluate peers ☑ (/smac-001/human_grading/view/courses/971628/assessments/10/peerGradingSets)

Results Phase

3. See results ☑ (/smac-001/human_grading/view/courses/971628/assessments/10/results/mine)

Your effective grade is 19

Your unadjusted grade is 19, which is simply the grade you received from your peers.

See below for details.

In this homework session 6 of Statistical Mechanics: Algorithms and Computations, we study **path-sampling methods** for the harmonic oscillator. The point is to **improve on** the naive path-sampling methods represented by **naive_harmonic_path.py**. We start with a **preparation program** that reminds us of the convenient way lists can manipulated in Python.

Then, in the harmonic oscillator, we will use the harmonic Levy construction in two ways in the trap, then the free Levy construction. Finally, we apply the free and the harmonic Levy construction in an anharmonic potential down to quite low temperatures.

A Summary: In this section we provide a simple **preparation program**, shown here, and then analyze it.

```
#
# Part one
#
L = range(10)
for k in range(10):
    print L
    L = L[3:] + L[:3]
print
#
# Part two
#
K = range(10)
for i in range(10):
    print K
    dummy = K.pop()
    K = [dummy] + K
print
#
# Part three
#
J = range(10)
for i in range(10):
    print K
    dummy = K.pop(0)
    K = K + [dummy]
print
#
# Part four
#
I = range(10)
weight = sum(a ** 2 for a in I)
```

A1 Download (cut-and-paste) the **Preparation program 1**. Run it and analyze its output.

1/ Explain what the "Part one" of the **Preparation program 1** is good for. **Attention, there are two aspects to this subquestion**, they are related to **list slicing** and to **list merging (concatenation)** (add prints where needed).
NB: You may consider looking up the references http://stackoverflow.com/questions/509211/pythons-slice-notation (http://stackoverflow.com/questions/509211/pythons-slice-notation) **and** http://stackoverflow.com/questions/1720421/merge-two-lists-in-python (http://stackoverflow.com/questions/1720421/merge-two-lists-in-python)
2/ Explain what the "Part two"of the **Preparation program 1** does (add prints where needed).
3/ Explain what the "Part three"of the **Preparation program 1** does (add prints where needed).
4/ Explain the concept of list comprehension underlying "Part four" of the **Preparation program 1** (add prints where needed).

NB: Answer each of the above questions briefly. Remember what you learned, especially in part one and part four, in the subsequent exercises.

1. L[3:] - Selects all elements of list L **except** the first three elements of list L and L[:3] picks the first three elements of list L (list slicing). Next, these are merged together and stored in L (list merging). L = L[3:] + L[:3]. Therefore, **with every iteration, the first three elements get moved to the back of list**. Output is as below:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[3, 4, 5, 6, 7, 8, 9, 0, 1, 2]
[6, 7, 8, 9, 0, 1, 2, 3, 4, 5]
[9, 0, 1, 2, 3, 4, 5, 6, 7, 8]
[2, 3, 4, 5, 6, 7, 8, 9, 0, 1]
[5, 6, 7, 8, 9, 0, 1, 2, 3, 4]
[8, 9, 0, 1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
[4, 5, 6, 7, 8, 9, 0, 1, 2, 3]
[7, 8, 9, 0, 1, 2, 3, 4, 5, 6]
```

2. dummy = K.pop() removes the last element from list K and stores it in dummy variable. Next, K = [dummy] + K merges the list with element dummy to K. **The net result is that the last element of K moves to the front with every update of K.** The output is:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[9, 0, 1, 2, 3, 4, 5, 6, 7, 8]
[8, 9, 0, 1, 2, 3, 4, 5, 6, 7]
[7, 8, 9, 0, 1, 2, 3, 4, 5, 6]
[6, 7, 8, 9, 0, 1, 2, 3, 4, 5]
[5, 6, 7, 8, 9, 0, 1, 2, 3, 4]
[4, 5, 6, 7, 8, 9, 0, 1, 2, 3]
[3, 4, 5, 6, 7, 8, 9, 0, 1, 2]
[2, 3, 4, 5, 6, 7, 8, 9, 0, 1]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

3. dummy = K.pop(0) removes the first element from list K and stores it in dummy variable. Next, K = K + [dummy] merges the list with element dummy to K. **The net result is that the first element of K moves to the llas with every update of K**. The output is:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
[2, 3, 4, 5, 6, 7, 8, 9, 0, 1]
[3, 4, 5, 6, 7, 8, 9, 0, 1, 2]
[4, 5, 6, 7, 8, 9, 0, 1, 2, 3]
[5, 6, 7, 8, 9, 0, 1, 2, 3, 4]
[6, 7, 8, 9, 0, 1, 2, 3, 4, 5]
[7, 8, 9, 0, 1, 2, 3, 4, 5, 6]
[8, 9, 0, 1, 2, 3, 4, 5, 6, 7]
[9, 0, 1, 2, 3, 4, 5, 6, 7, 8]
```

4. I=range(10) stores the list [0,1,2,3,4,5,6,7,8,9] in I. Next, sum(a ** 2 for a in I) loops through each element of list I, squares it and compute the sum. Essentially, sum(a ** 2 for a in I) = I[0]^2 + I[1]^2 + ... + I[9]^2. The resulting sum is stored in variable **weight**.

Evaluation/feedback on the above work

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

A1 Here, you should evaluate your fellow student's good understanding of slicing and concatenation (merging) in Python, but notice that this is a preparation program in a statistical physics class, and not a Python course, so some approximation is OK. Students don't really have to mention that "pop", etc, is called a "method".

There are four issues
1/ slicing and concatenation (merging)
2/ pop operation (method), taking off the LAST element of a list, followed by merge
3/ pop operation (method), (taking off the FIRST element of a list), followed by merge
4/ List comprehension

An answer obtaining full score is as follows:
1/ slicing and concatenation (merging), leading to a roll, or a shifting of indices
2/ pop operation (method), taking off the last element of a list, followed by merge leading to the shifting of indices.
3/ pop operation (method), (taking off the first element of a list), followed by merge leading to the shifting of indices.
4/ List comprehension, summing the squares.

POINTS - DESCRIPTION
Give from 0 to 4 points according to the number of issues addressed correctly.
1 point: slicing and concatenation.
1 point: pop of last element.
1 point: pop of first element.
1 point: list comprehension for summing squares.

Score from your peers: **4**

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

peer 2 → nicely explained

peer 3 → *[This area was left blank by the evaluator.]*

B In this part, you explore sampling methods starting from naive_harmonic_path.py.

B1 Download (cut-and-paste) and run the below program, a variant of naive_harmonic_path.py

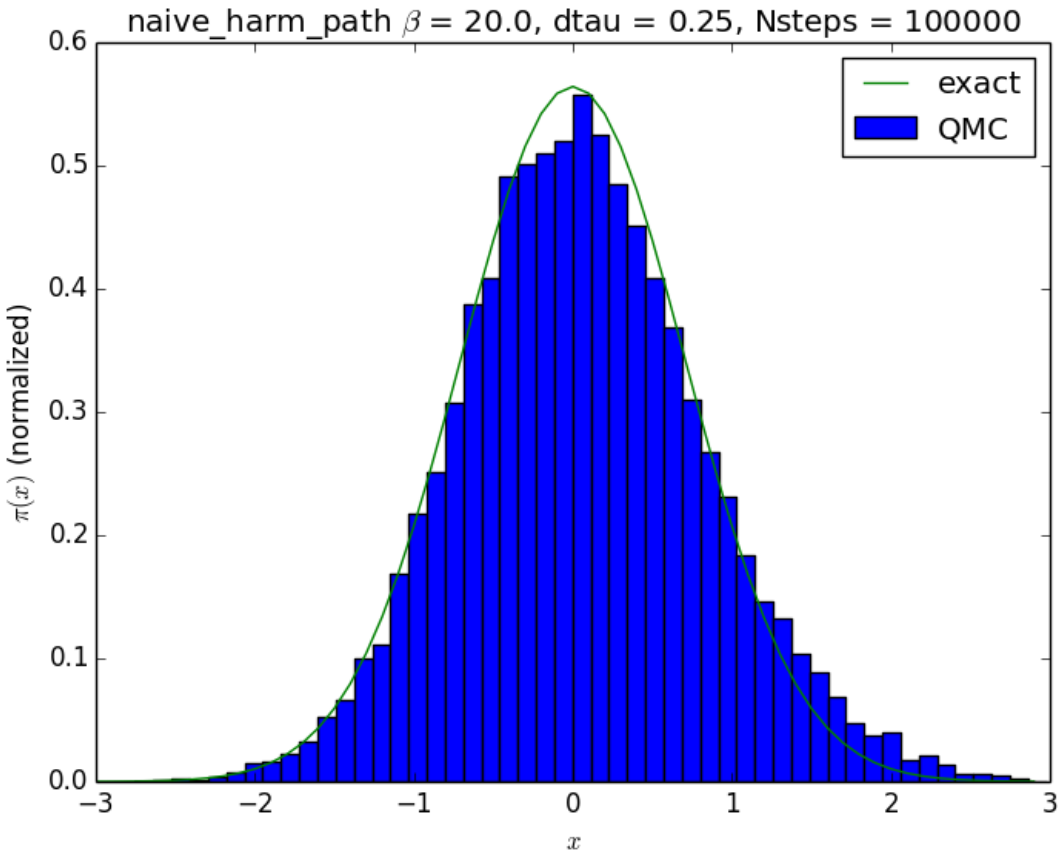
```
import math, random, pylab

def rho_free(x, y, beta):
    return math.exp(-(x - y) ** 2 / (2.0 * beta))

beta = 20.0
N = 80
dtau = beta / N
delta = 10.0
n_steps = 100000
x = [2.0] * N
data = []
for step in range(n_steps):
    k = random.randint(0, N - 1)
    knext, kprev = (k + 1) % N, (k - 1) % N
    x_new = x[k] + random.uniform(-delta, delta)
    old_weight  = (rho_free(x[knext], x[k], dtau) *
                   rho_free(x[k], x[kprev], dtau) *
                   math.exp(-0.5 * dtau * x[k] ** 2))
    new_weight  = (rho_free(x[knext], x_new, dtau) *
                   rho_free(x_new, x[kprev], dtau) *
                   math.exp(-0.5 * dtau * x_new ** 2))
    if random.uniform(0.0, 1.0) < new_weight / old_weight:
        x[k] = x_new
        acc += 1
    if step % 100 == 0:
        k = random.randint(0, N - 1)
        data.append(x[k])
pylab.hist(data, bins=50, normed=True, label='QMC')
x_values = [0.1 * a for a in range (-30, 30)]
y_values = [math.sqrt(math.tanh(beta / 2.0)) / math.sqrt(math.pi) * \
                  math.exp( - xx **2 * math.tanh( beta / 2.0)) for xx in x_values]
pylab.plot(x_values, y_values, label='exact')
pylab.xlabel('$x$')
pylab.ylabel('$\\pi(x)$ (normalized)')
pylab.axis([-3.0, 3.0, 0.0, 0.6])
pylab.legend()
ProgType = 'naive_harm_path'
pylab.title(ProgType + ' beta = ' + str(beta) + ', dtau = ' + str(dtau) + ', Nsteps = '+
str(n_steps))
pylab.savefig(ProgType + str(beta) + '.png')
pylab.show()
```

Modify the sampling rate (at " if step%100 == 0: "), and the stepsize (at "delta = ...") but **NOT the initial condition**, and **NOT the number of iterations**, and **upload** the best approximation you obtained for pi(x) (**graphics file**). **Discuss** in a few words what you observe. **Explain the three lines of this code containing the word "ProgType".**

step % 100 == 0 changed to step % 1 == 0 to store all the results.
delta = 10.0 changed to delta = 1.0. This choice of delta gives the acceptance ratio (acc/n_steps)

as 0.50954.

Lowering the sampling frequency and delta lead to improvement in the sampling leading to better approximation of QMC result to the analytical exact result.




```
1 ProgType = 'naive_harm_path'
2 pylab.title(ProgType + ' beta = ' + str(beta) + ', dtau = ' + str(dtau) + ', \
            Nsteps = '+ str(n_steps))
3 pylab.savefig(ProgType + str(beta) + '.png')
```

ProgType is initialized as 'naive_harm_path in line 1. Next, it's used to create title of the plot in line 2 and to file name of the graphic file in line 3.

Evaluation/feedback on the above work

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

Here, you should evaluate your fellow student's working knowledge of naive harmonic path.

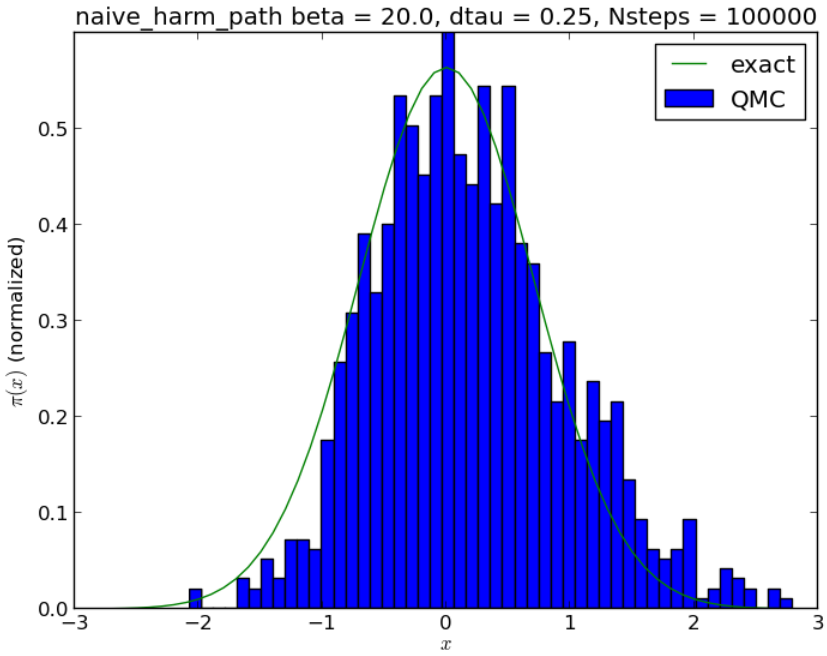
There are two issues
1/ It should come out that the program is quite slow, that the step size of 10 is much too big, and that the optimal step size is about 1, based on an analysis of the rejection rate. A file corresponding to the file shown here should be uploaded.

2/ It should be said that there are string concatenations involving ProgType that allow to produce good titles and filenames with minimal

effort.

An answer obtaining full score is as follows:

1/ the program is quite slow, and the step size of 10 is much too big. I estimate that he optimal step size is about 1, based on an analysis of the rejection rate, and here is a a file corresponding to the best data obtained, they are not great:



2/ string concatenations involving ProgType that allow to produce good titles and filenames with minimal effort.

POINTS - DESCRIPTION
Give from 0 to 2 points according to the number of issues addressed correctly

1 point - Slowness + data file
1 point understanding of ProgType

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.]*

peer 3 → *[This area was left blank by the evaluator.]*

B2 Now modify your program by introducing levy_harmonic_path.
For simplicity, we **provide the function** levy_harmonic_path that you should use.

```
def levy_harmonic_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        Ups1 = 1.0 / math.tanh(dtau) + \
               1.0 / math.tanh(dtau_prime)
        Ups2 = x[k - 1] / math.sinh(dtau) + \
               xend / math.sinh(dtau_prime)
        x.append(random.gauss(Ups2 / Ups1, \
               1.0 / math.sqrt(Ups1)))
    return x
```

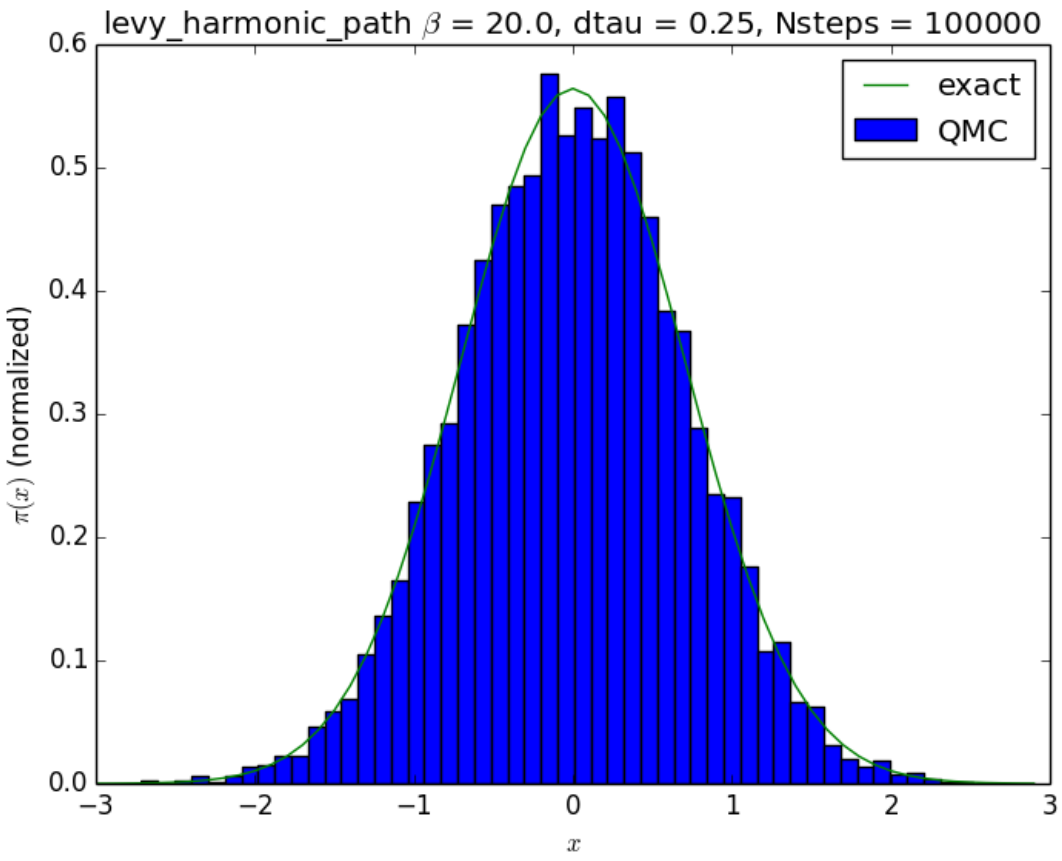
This function (which **returns** a list of **N elements**) should replace the naive construction for the N elements of the path (both xstart and xend should be the bead "0" of the path). **Make sure (print) that your path always has N elements**: **The final point x[N] is NOT used**. It agrees with x[0].

Run your program first for very small N. Notice that the position x[0] never changes.

To fix this problem, use "Part 1" of the **Preparation program 1** to wrap the path around the tau-axis **after each Levy construction** (replace "3" in the **Preparation program 1** for example by "N / 2"). **Don't forget to update ProgType.**

Now, **check** that this nice program recovers the exact result for pi(x) for beta = 20. **Upload** your program, the output (**upload graphics file**), and **give a short explanation** of what this program does. **Explain whether this is a Markov-chain algorithm or not**.

```
import math, random, pylab

def levy_harmonic_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        Ups1 = 1.0 / math.tanh(dtau) + 1.0 / math.tanh(dtau_prime)
        Ups2 = x[k - 1] / math.sinh(dtau) + xend / math.sinh(dtau_prime)
        x.append(random.gauss(Ups2 / Ups1, 1.0 / math.sqrt(Ups1)))
        wrap = random.randint(0, N - 1)
        x = x[wrap:] + x[:wrap]
    return x

def rho_free(x, y, beta):
    return math.exp(-(x - y) ** 2 / (2.0 * beta))

beta = 20.0
N = 80
dtau = beta / N
n_steps = 100000
x = [2.0] * N
data = []

for step in range(n_steps):
    x = levy_harmonic_path(x[0], x[0], dtau, N)

    if step % 10 == 0:
        k = random.randint(0, N - 1)
        data.append(x[k])

pylab.hist(data, bins=50, normed=True, label='QMC')
x_values = [0.1 * a for a in range (-30, 30)]
y_values = [math.sqrt(math.tanh(beta / 2.0)) / math.sqrt(math.pi) * \
                  math.exp( - xx **2 * math.tanh( beta / 2.0)) for xx in x_values]
pylab.plot(x_values, y_values, label='exact')
pylab.xlabel('$x$')
pylab.ylabel('$\\pi(x)$ (normalized)')
pylab.axis([-3.0, 3.0, 0.0, 0.6])
pylab.legend()
ProgType = 'levy_harmonic_path'
pylab.title(ProgType + ' $\\beta$ = ' + str(beta) + ', dtau = ' + str(dtau) + ', Nsteps
 = '+ str(n_steps))
pylab.savefig(ProgType + str(beta) + '.png')
pylab.show()
```



The direct path sampling using Levy construction is being used. It does a stochastic interpolation between points x_0 and x_N. The Levy construction satisfies a local construction principle: the path $x(\tau)$, in any interval $[\tau_1, \tau_2]$, is the stochastic interpolation of its end points $x(\tau_1)$ and $x(\tau_2)$, but the behavior of the path outside the interval plays no role.

The algorithm is Markov Chain since the next state only depends on the current state and not on the sequence of events that preceded it.

Evaluation/feedback on the above work

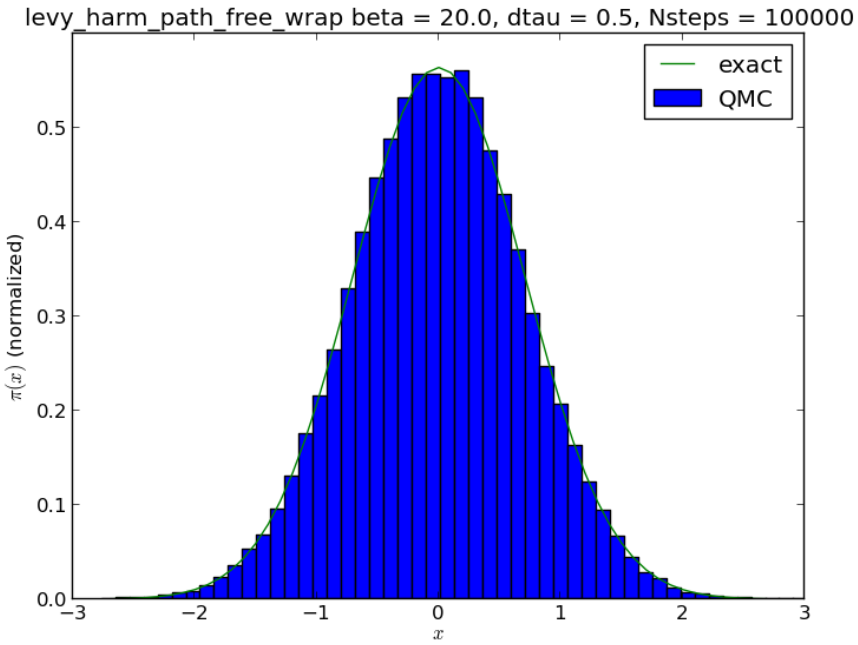
Note: this section can only be filled out during the evaluation phase.

Here, you should evaluate your fellow student's ability to implement the levy_harmonic_path, together with the wrapping operation and to get it running.

There are three issues:
1/ Data file upload
2/ Program upload (it should resemble the below program, Check that the ProgType variable is changed from "naive" to something
else.)
3/ **E**xplanation should mention that the program samples harmonic Levy path from x_0 -> x_0 then shifts the path. Should mention that this is a Markov-chain Monte Carlo program (it is not necessary that the student explicitly mention that detailed balance is satisfied, although this is of course true).

An answer obtaining full score is as follows:

1/ Here are the results obtained:



2/ Here is the program (note that the ProgType variable is changed from "naive")

```
import math, random, pylab

def levy_harmonic_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        Ups1 = 1.0 / math.tanh(dtau) + \
               1.0 / math.tanh(dtau_prime)
        Ups2 = x[k - 1] / math.sinh(dtau) + \
               xend / math.sinh(dtau_prime)
        x.append(random.gauss(Ups2 / Ups1, \
               1.0 / math.sqrt(Ups1)))
    return x

beta = 20.0
N =  80
Ncut = N / 2
dtau = beta / N
delta = 1.0
n_steps = 100000
x = [0.0] * N
data = []
for step in range(n_steps):
    x = levy_harmonic_path(x[0], x[0], dtau, N)
    x = x[Ncut:] + x[:Ncut]
    k = random.randint(0, N - 1)
    data.append(x[k])
pylab.hist(data, bins=50, normed=True, label='QMC')
x_values = [0.1 * a for a in range (-30,30)]
y_values = [math.sqrt(math.tanh(beta / 2.0)) / math.sqrt(math.pi) * \
                  math.exp( - xx **2 * math.tanh( beta / 2.0)) for xx in
 x_values]
pylab.plot(x_values, y_values, label='exact')
pylab.xlabel('$x$')
pylab.ylabel('$\\pi(x)$ (normalized)')
pylab.axis([-3.0, 3.0, 0.0, 0.6])
pylab.legend()
ProgType = 'levy_harm_path_wrap'
pylab.title(ProgType + ' beta = ' + str(beta) + ', dtau = ' + str(dtau)
+ ', Nsteps = '+ str(n_steps))
pylab.savefig(ProgType + str(beta) + '.png')
pylab.show()
```

3/ Explanation: This program samples harmonic Levy path from x_0 -> x_0, then shifts the path. This is a Markov-chain Monte
Carlo program. [It satisfies detailed balance] (last part optional).

POINTS - DESCRIPTION

Give from 0 to 3 points according to the number of issues addressed correctly

1 point: figure
1 point: program

1 point: short explanation including realization that this is a Markov-chain algorithm

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.]*

peer 3 → *[This area was left blank by the evaluator.]*

B3 Now **modify** the program of section **B2**: As we know the distribution of x[0], we may **sample from this distribution**. Therefore, **determine the standard deviation** of the exact distribution pi(x) \propto .exp(- x **2 * tanh(beta / 2.0) (pay attention to the difference between standard deviation and variance).

Your new program should thus contain a line

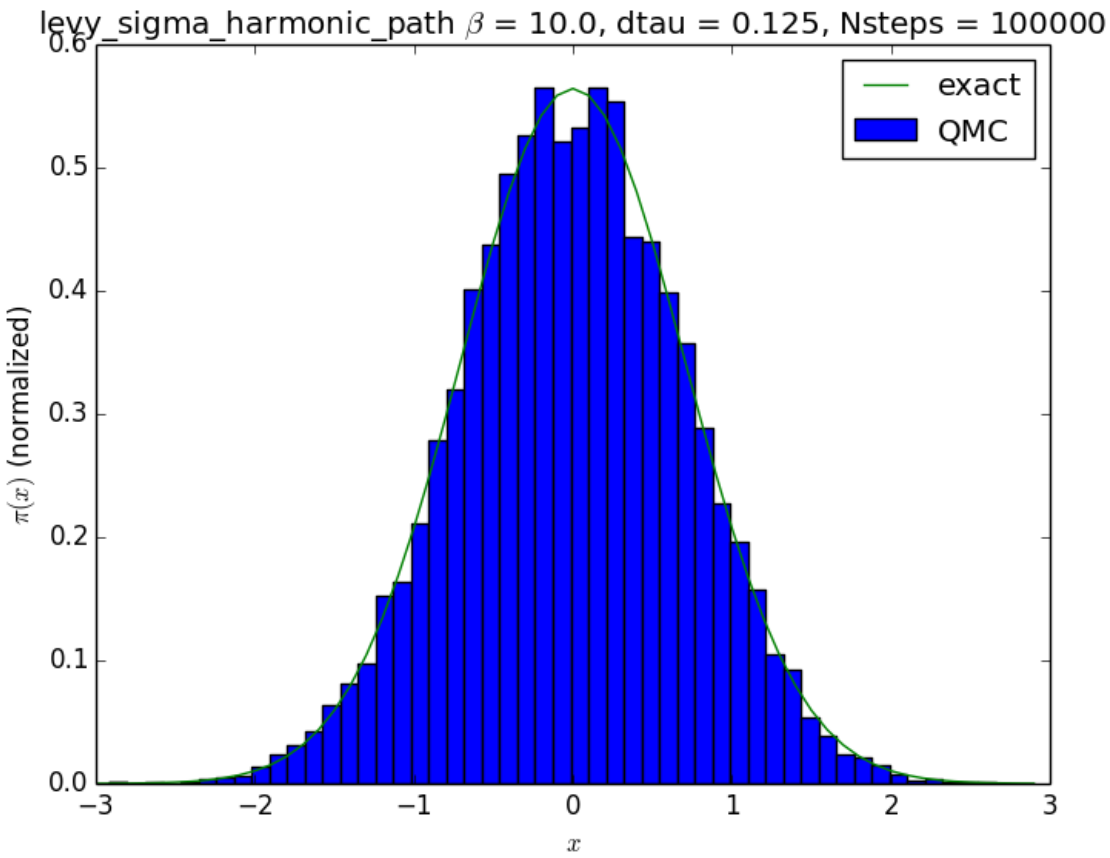
```
x[0] = random.gauss(0.0, sigma)
```

(or equivalent), where sigma is the standard deviation of pi(x), and **this line should be followed** by the harmonic Levy construction. **No "wrapping"** needed. **Don't forget to update ProgType**.

Now, **check that this beautiful program recovers the exact result for pi(x)**. **Upload your program**, **upload the output** for beta = 10 (graphics file), and **give a short explanation** of what this program does. **Explain whether this is a Markov-chain algorithm or not**.

NB: This algorithm is a **crucial ingredient** of the week 7 "Bose-Einstein condensation" program. Outstanding as it is, it is less general than the Program of section B2.

```
import math, random, pylab

def levy_harmonic_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        Ups1 = 1.0 / math.tanh(dtau) + 1.0 / math.tanh(dtau_prime)
        Ups2 = x[k - 1] / math.sinh(dtau) + xend / math.sinh(dtau_prime)
        x.append(random.gauss(Ups2 / Ups1, 1.0 / math.sqrt(Ups1)))
    return x

def rho_free(x, y, beta):
    return math.exp(-(x - y) ** 2 / (2.0 * beta))

beta = 10.0
N = 80
dtau = beta / N
n_steps = 100000
x = [2.0] * N
data = []
sigma = math.sqrt( math.tanh( beta / 2.0 ) / 2.0 )
for step in range(n_steps):
    x[0] = random.gauss(0.0, sigma)
    x = levy_harmonic_path(x[0], x[0], dtau, N)

    if step % 10 == 0:
        k = random.randint(0, N - 1)
        data.append(x[k])

pylab.hist(data, bins=50, normed=True, label='QMC')
x_values = [0.1 * a for a in range (-30, 30)]
y_values = [math.sqrt(math.tanh(beta / 2.0)) / math.sqrt(math.pi) * \
                  math.exp( - xx **2 * math.tanh( beta / 2.0)) for xx in x_values]
pylab.plot(x_values, y_values, label='exact')
pylab.xlabel('$x$')
pylab.ylabel('$\\pi(x)$ (normalized)')
pylab.axis([-3.0, 3.0, 0.0, 0.6])
pylab.legend()
ProgType = 'levy_sigma_harmonic_path'
pylab.title(ProgType + ' $\\beta$ = ' + str(beta) + ', dtau = ' + str(dtau) + ', \
            Nsteps = '+ str(n_steps))
pylab.savefig(ProgType + str(beta) + '.png')
pylab.show()
```



We do a small modification by sampling x_0 from the exact distribution. Other aspects remain same as in B2. This is also a Markov-chain algorithm.

Evaluation/feedback on the above work

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

Here, you should evaluate your fellow student's ability to implement the levy_harmonic_path, together with the correct sampling of x_0

There are four issues:
1/ sigma is 1.0 / sqrt(2.0 * tanh(beta / 2.0), or in other words (Python):

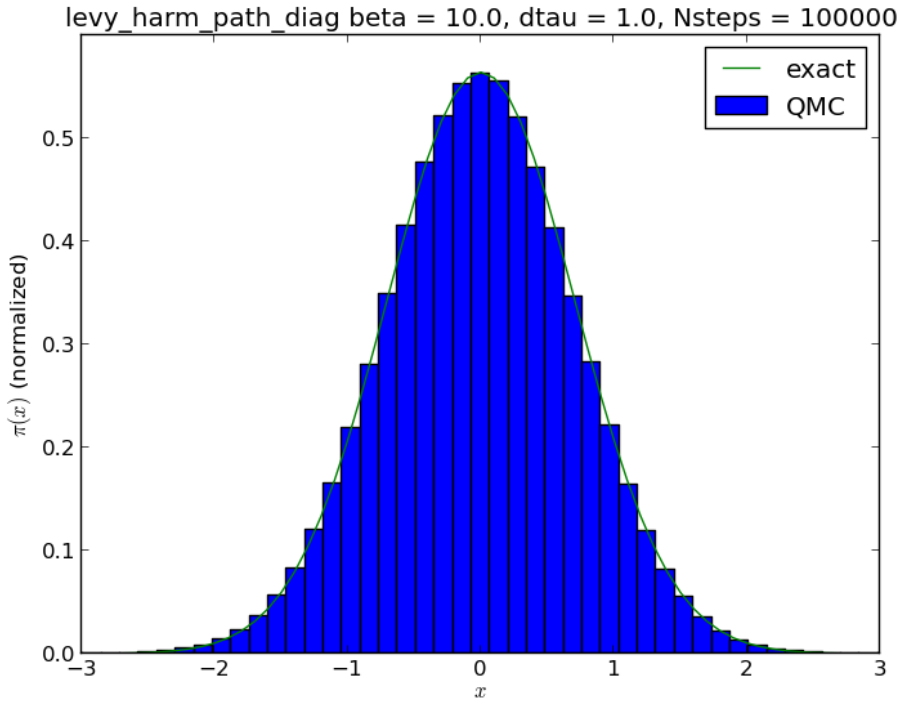
```
xxx = random.gauss(0.0, 1.0 / math.sqrt(2.0 * math.tanh( beta / 2.0)))
```

2/ Upload of results file, that should resemble about what is shown here
There's no problem if another temperature was chosen than beta = 10.

3/ Upload of program, that should resemble the one here

4/ It should be mentioned that this is a Direct-sampling algorithm.
It is not required that the student explains that there is NO Trotter error in this program, although this is of course the case

An answer obtaining full score is as follows:
1/ sigma = 1.0 / sqrt(2.0 * tanh(beta / 2.0)
2/ Here are the results



3/ Here the program

```
import math, random, pylab

def levy_harmonic_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        Ups1 = 1.0 / math.tanh(dtau) + \
               1.0 / math.tanh(dtau_prime)
        Ups2 = x[k - 1] / math.sinh(dtau) + \
               xend / math.sinh(dtau_prime)
        x.append(random.gauss(Ups2 / Ups1, \
               1.0 / math.sqrt(Ups1)))
    return x

beta = 10.0
N = 10
dtau = beta / N
delta = 1.0
n_steps = 100000
x = [0.0] * N
data = []
for step in range(n_steps):
    xxx = random.gauss(0.0, 1.0 / math.sqrt(2.0 * math.tanh( beta / 2.0)
))
    x = levy_harmonic_path(xxx, xxx, dtau, N)
    k = random.randint(0, N - 1)
    data += x
pylab.hist(data, bins=50, normed=True, label='QMC')
x_values = [0.1 * a for a in range (-30,30)]
y_values = [math.sqrt(math.tanh(beta / 2.0)) / math.sqrt(math.pi) * \
                  math.exp( - xx ** 2 * math.tanh( beta / 2.0)) for xx i
n x_values]
pylab.plot(x_values, y_values, label='exact')
pylab.xlabel('$x$')
pylab.ylabel('$\\pi(x)$ (normalized)')
pylab.axis([-3.0, 3.0, 0.0, 0.6])
pylab.legend()
ProgType = 'levy_harm_path_diag'
pylab.title(ProgType + ' beta = ' + str(beta) + ', dtau = ' + str(dtau)
+ ', Nsteps = '+ str(n_steps))
pylab.savefig(ProgType + str(beta) + '.png')
pylab.show()
```

4/ This is a direct-sampling algorithm, not a Markov-chain algorithm

POINTS - DESCRIPTION

Give from 0 to 4 points according to the number of issues addressed correctly.

1 point: correct sigma
1 point: correct result
1 point: correct program
1 point: Understanding that this is direct sampling

Score from your peers: **3**

peer 1 → I guess your definition of sigma as 'math.sqrt(math.tanh(beta / 2.0) / 2.0)' is different from the answer '1.0 / math.sqrt(2.0 * math.tanh(beta / 2.0))', although numerically these two give similar results.

peer 2 → Non-Markov. I personally don't understand how they got such a perfect curve - my result was similar to yours.

peer 3 → *[This area was left blank by the evaluator.]*

C Here we go back to the wrapping algorithm of section B2 (NOT B3!!). We now use the free Levy construction in the harmonic potential.

C1 Add the function levy_free_path to the wrapping algorithm of section **B2**. For your convenience, it is given here:

```
def levy_free_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        x_mean = (dtau_prime * x[k - 1] + dtau * xend) / \
                 (dtau + dtau_prime)
        sigma = math.sqrt(1.0 / (1.0 / dtau + 1.0 / dtau_prime))
        x.append(random.gauss(x_mean, sigma))
    return x
```

Note that it outputs a path of length N (not N+1 as in the lecture). **Do not erase** the function levy_harmonic_path, you will need it later.

Check that for a path obtained from the Free levy construction, the statistical weight is given by the celebrated exponential of the
"Feynman action" exp(- \sum delta_tau \sum_i V(i)) (in Python, by

```
Weight_trott = math.exp(sum(-a **2/ 2.0 * dtau for a in x))
```

(see **Preparation program**, Part 4)).

Therefore, **replace the harmonic Levy construction** of section **B2** by the **free Levy construction**, but new paths should be accepted with the Metropolis algorithm.

The **program of this section should**
1/ **Propose** a new path between x[0] and x[0], from Leey_free_path.
2/ **Compute** its Weight (modify the above line... x[..] will have to be replaced).
3/ **Accept** the new path with probability min(1, Weight_new/ Weight_old).
4/ **Update** the Weight (if move accepted).
5/ **Update** the path (if move accepted).

6/ Wrap the path, as in section **B2**.

NB: Attention in part 5/ **x = x_new** leads to desaster, **x = x_new[:]** is much better.

Check your program for small values of beta. At large beta (around beta = 20), the acceptance probability will have fallen to zero, but this can be cured in a single line...

```
x_new = levy_free_path(x[0], x[Ncut], dtau, Ncut) + x[Ncut:]
```

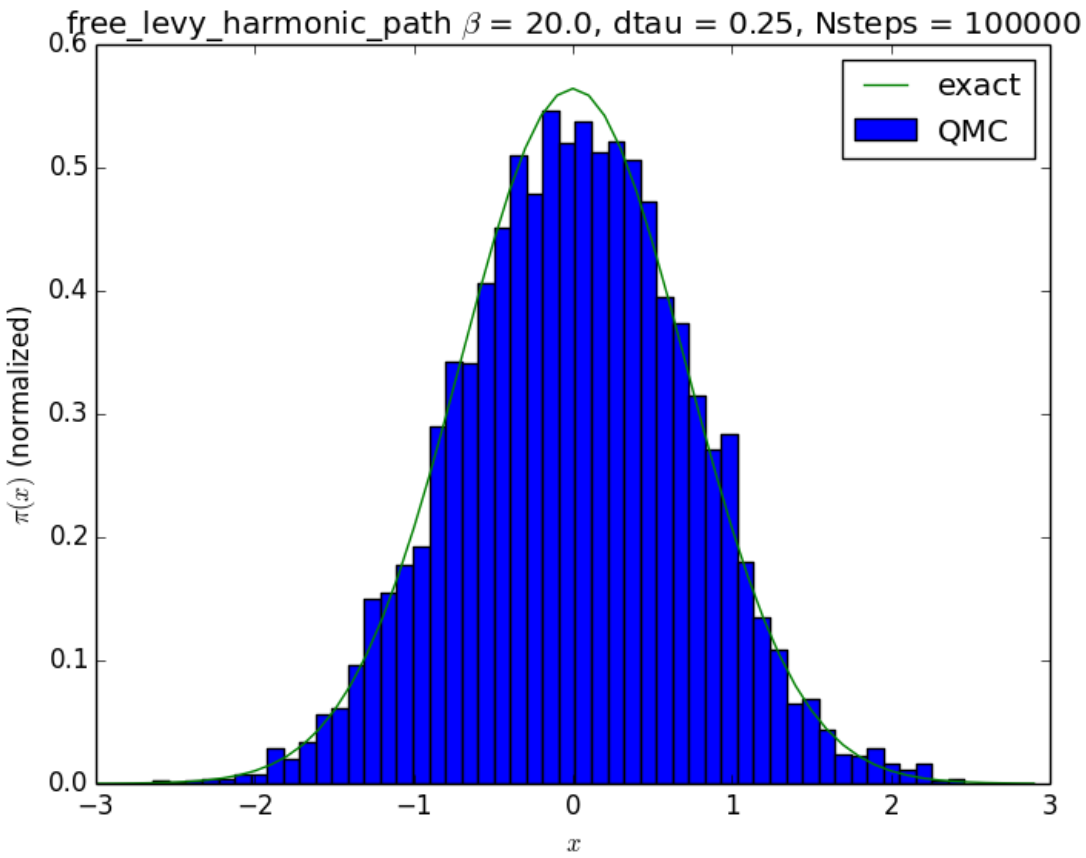
...where Ncut is chosen suitably. We thus construct a new path between 0 and Ncut-1, and the old path from Ncut through N - 1.

Dont forget to update ProgType. Run your program at **beta = 20**, **upload results** (graphics file), **upload the program.**

```
import math, random, pylab

def levy_free_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        x_mean = (dtau_prime * x[k - 1] + dtau * xend) / \
                 (dtau + dtau_prime)
        sigma = math.sqrt(1.0 / (1.0 / dtau + 1.0 / dtau_prime))
        x.append(random.gauss(x_mean, sigma))
        wrap = random.randint(0, N - 1)
        x = x[wrap:] + x[:wrap]
    return x

def levy_harmonic_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        Ups1 = 1.0 / math.tanh(dtau) + 1.0 / math.tanh(dtau_prime)
        Ups2 = x[k - 1] / math.sinh(dtau) + xend / math.sinh(dtau_prime)
        x.append(random.gauss(Ups2 / Ups1, 1.0 / math.sqrt(Ups1)))
        wrap = random.randint(0, N - 1)
        x = x[wrap:] + x[:wrap]
    return x

def rho_free(x, y, beta):
    return math.exp(-(x - y) ** 2 / (2.0 * beta))

beta = 20.0
N = 80
dtau = beta / N
n_steps = 100000
x = [1.0] * N
data = []
Weight_old = math.exp(sum(-a ** 2/ 2.0 * dtau for a in x))
acc = 0
for step in range(n_steps):
        Ncut = random.randint(0,N/2)
        x_new = levy_free_path(x[0], x[Ncut], dtau, Ncut) + x[Ncut:]
        Weight_new = math.exp(sum(-a ** 2/ 2.0 * dtau for a in x_new))

        if random.uniform(0.0, 1.0) < Weight_new / Weight_old:
                x = x_new[:]
                acc += 1
                Weight_old = Weight_new

        #print x
        if step % 10 == 0:
                k = random.randint(0, N - 1)
                data.append(x[k])
```

```
print 'acc_ratio = ', acc/float(n_steps)
pylab.hist(data, bins=50, normed=True, label='QMC')
x_values = [0.1 * a for a in range (-30, 30)]
y_values = [math.sqrt(math.tanh(beta / 2.0)) / math.sqrt(math.pi) * \
                 math.exp( - xx **2 * math.tanh( beta / 2.0)) for xx in x_values]
pylab.plot(x_values, y_values, label='exact')
pylab.xlabel('$x$')
pylab.ylabel('$\\pi(x)$ (normalized)')
pylab.axis([-3.0, 3.0, 0.0, 0.6])
pylab.legend()
ProgType = 'free_levy_harmonic_path'
pylab.title(ProgType + ' $\\beta$ = ' + str(beta) + ', dtau = ' + str(dtau) + ', Nsteps
 = '\
            + str(n_steps))
pylab.savefig(ProgType + str(beta) + '.png')
pylab.show()
```

acc_ratio = 0.54505



Evaluation/feedback on the above work

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

C1 Here, you should evaluate your fellow student's ability to implement a reweighting scheme for levy_free_path for the harmonic oscillator

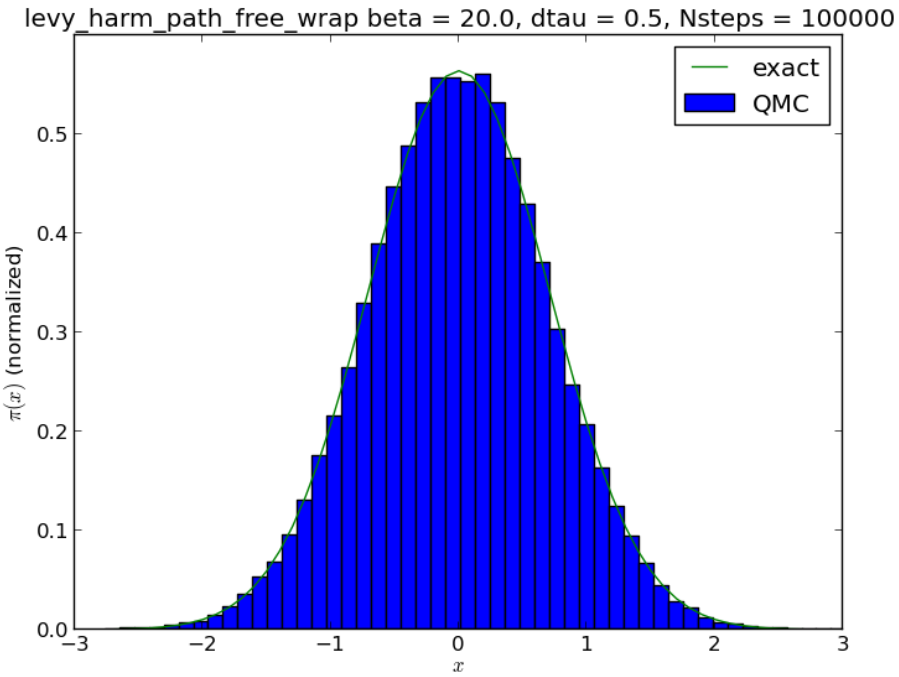
There are three issues:

1/ Reweighting by
new_Weight_trott = math.exp(sum(-a ** 2 / 2.0 * dtau for a in x_new)) (or equivalent) should be implemented. This weight can be taken over the whole path (as shown here) or only about the elements that change.

2/ Program, that should resemble the one shown here In particular, check that the Trotter weight is correctly implemented, and that the Metropolis algorithm is implemented involving a line resembling

if random.uniform(0.0, 1.0) < new_Weight_trott / Weight_trott

3/ Results, that should correspond to the exact result:



It is not required that the student indicates that there is some Trotter error, although this is of course the case.

An answer obtaining full score is as follows:
Here is the program

```
import math, random, pylab

def levy_free_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        x_mean = (dtau_prime * x[k - 1] + dtau * xend) / \
                 (dtau + dtau_prime)
        sigma = math.sqrt(1.0 / (1.0 / dtau + 1.0 / dtau_prime))
        x.append(random.gauss(x_mean, sigma))
    return x

beta = 20.0
N = 40
Ncut = N / 2
dtau = beta / N
delta = 1.0
n_steps = 100000
x = [0.1] * N
Weight_trott = math.exp(sum(-a ** 2 / 2.0 * dtau for a in x))
data = []
n_acc = 0
for step in range(n_steps):
    x_new = levy_free_path(x[0], x[Ncut], dtau, Ncut) + x[Ncut:]
    new_Weight_trott = math.exp(sum(-a ** 2 / 2.0 * dtau for a in x_new)
)
    if random.uniform(0.0, 1.0) < new_Weight_trott / Weight_trott:
        n_acc += 1
        Weight_trott = new_Weight_trott
        x = x_new[:]
    x = x[1:] + x[:1]
    k = random.randint(0, N - 1)
    data.append(x[k])
print n_acc / float (n_steps), 'acceptance rate'
pylab.hist(data, bins=50, normed=True, label='QMC')
x_values = [0.1 * a for a in range (-30, 30)]
y_values = [math.sqrt(math.tanh(beta / 2.0)) / math.sqrt(math.pi) * \
                  math.exp( - xx ** 2 * math.tanh( beta / 2.0)) for xx i
n x_values]
pylab.xlabel('$x$')
pylab.ylabel('$\\pi(x)$ (normalized)')
pylab.axis([-3.0, 3.0, 0.0, 0.6])
pylab.plot(x_values, y_values, label='exact')
pylab.legend()
ProgType = 'levy_harm_path_free_wrap'
pylab.title(ProgType + ' beta = ' + str(beta) + ', dtau = ' + str(dtau)
+ ', Nsteps = '+ str(n_steps))
pylab.savefig(ProgType + str(beta) + '.png')
pylab.show()
```

POINTS - DESCRIPTION

Give from 0 to 3 points according to the number of issues addressed correctly

1 point for reweighting (see above)
1 point for program (see above)
1 point for figure (see above)

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.]*

peer 3 → *[This area was left blank by the evaluator.]*

C2 Adapt the program of section **C1** for the anharmonic potential implemented by the function

V(x) = x ** 2 / 2 + cubic * x ** 3 + quartic * x ** 4

with cubic = -1, quartic = 1.

Modify your program so that it samples the anharmonic path integral at beta = 20, starting from
a/ the **free Levy construction**.
b/ the **harmonic Levy construction**.

Naturally, the Feynman action (the Weight) is different in both cases, because **a term counted in the "measure" of the probability distribution should not be counted in the "statistical weight" of the paths**. **Upload your program** (one program is enough, but it should be able to run for both cases, maybe using a **LevyType** variable to choose the potential and to modify ProgType). **Upload the two distributions** pi(x) (separate files OK). They should be very similar, at sufficiently small dtau.

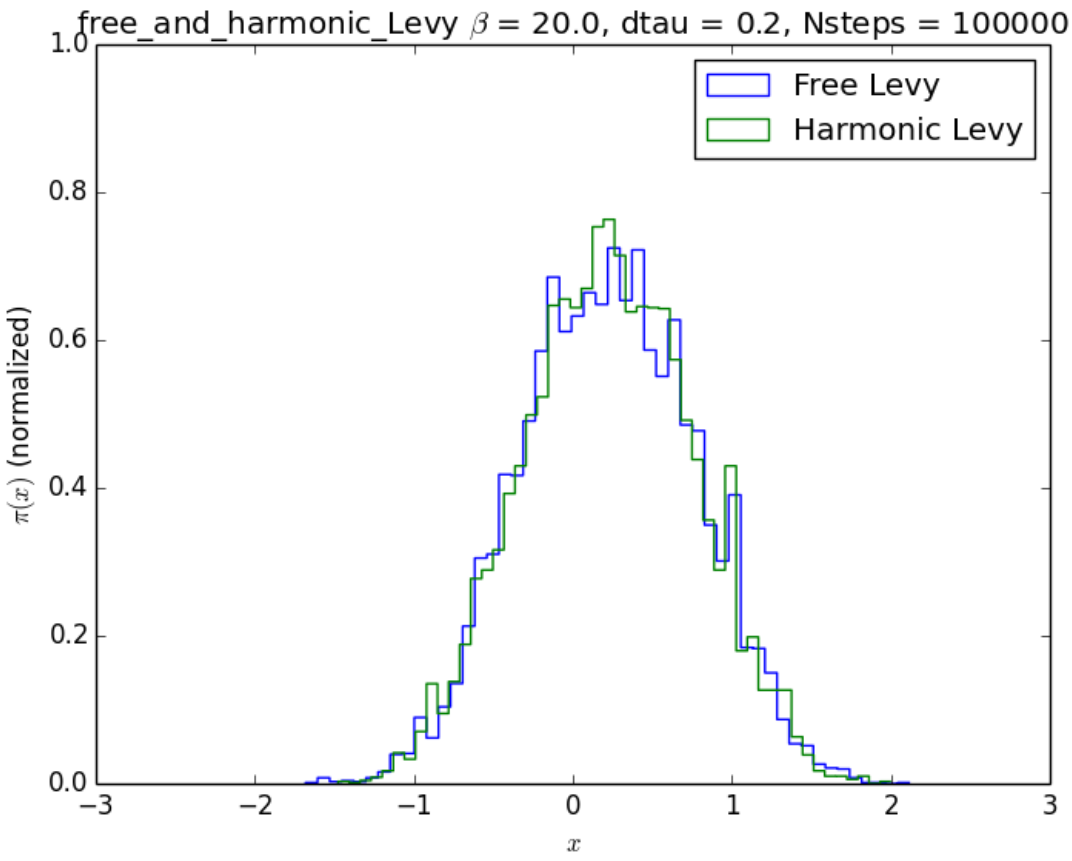
```
import math, random, pylab

def V(x):
    pot =  x ** 2 / 2 + cubic * x ** 3 + quartic * x ** 4
    return pot

def levy_free_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        x_mean = (dtau_prime * x[k - 1] + dtau * xend) / (dtau + dtau_prime)
        sigma = math.sqrt(1.0 / (1.0 / dtau + 1.0 / dtau_prime))
        x.append(random.gauss(x_mean, sigma))
        wrap = random.randint(0, N - 1)
        x = x[wrap:] + x[:wrap]
    return x

def levy_harmonic_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        Ups1 = 1.0 / math.tanh(dtau) + 1.0 / math.tanh(dtau_prime)
        Ups2 = x[k - 1] / math.sinh(dtau) + xend / math.sinh(dtau_prime)
        x.append(random.gauss(Ups2 / Ups1, 1.0 / math.sqrt(Ups1)))
        wrap = random.randint(0, N - 1)
        x = x[wrap:] + x[:wrap]
    return x

def rho_free(x, y, beta):
    return math.exp(-(x - y) ** 2 / (2.0 * beta))

beta = 20.0
N = 100
dtau = beta / N
n_steps = 100000

cubic = -1
quartic = 1

for LevyType in range(2):
        acc = 0
        data = []
        x = [1.0] * N

        #Free Levy
        if LevyType == 0:
                Weight_old = math.exp(sum(-V(a) * dtau for a in x))
        #Harmonic Levy
        else:
                Weight_old = math.exp(sum(-(V(a) - a ** 2 / 2.0) * dtau for a in x))
```

```
        for step in range(n_steps):
                if LevyType == 0:
                        Ncut = random.randint(0,N/2)
                        x_new = levy_free_path(x[0], x[Ncut], dtau, Ncut) + x[Ncut:]
                        Weight_new = math.exp(sum(-V(a) * dtau for a in x_new))
                else:
                        Ncut = random.randint(0,N/2)
                        x_new = levy_harmonic_path(x[0], x[Ncut], dtau, Ncut) + x[Ncut:
]
                        Weight_new = math.exp(sum(-(V(a) - a ** 2 / 2.0) * dtau for a i
n x_new))

                if random.uniform(0.0, 1.0) < Weight_new / Weight_old:
                        x = x_new[:]
                        acc += 1
                        Weight_old = Weight_new

                if step % 10 == 0:
                        k = random.randint(0, N - 1)
                        data.append(x[k])

        print 'LevyType =', LevyType, 'acc_ratio = ', acc/float(n_steps)
        if LevyType == 0:
                mylabel = 'Free Levy'
        else:
                mylabel = 'Harmonic Levy'
        pylab.hist(data, bins=50, normed=True, label=mylabel, histtype='step')

pylab.xlabel('$x$')
pylab.ylabel('$\\pi(x)$ (normalized)')
pylab.axis([-3.0, 3.0, 0.0, 1.0])
pylab.legend()
ProgType = 'free_and_harmonic_Levy'
pylab.title(ProgType + ' $\\beta$ = ' + str(beta) + ', dtau = ' + str(dtau) + ', Nsteps
 = '+ str(n_steps))
pylab.savefig(ProgType + str(beta) + '.png')
pylab.show()
```

dtau = 20/100 = 0.2

LevyType = 0 [Free Levy] acc_ratio = 0.35638
LevyType = 1 [Harmonic Levy] acc_ratio = 0.56401

Both the histograms (**method a: Free Levy and method b: Harmonic Levy**) have been plotted on the same graph using the histtype = 'step' to facilitate comparison. As suggested in the exercise explanation, both the histograms are quite similar to each other for small dtau (=0.2 in this case).



Evaluation/feedback on the above work

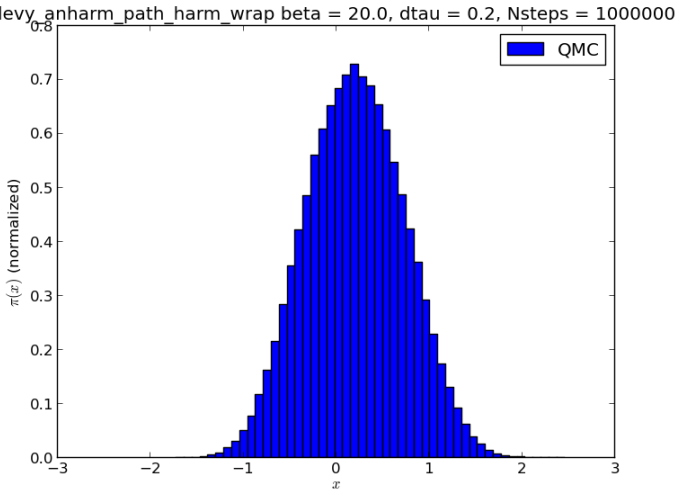
Note: this section can only be filled out during the evaluation phase.

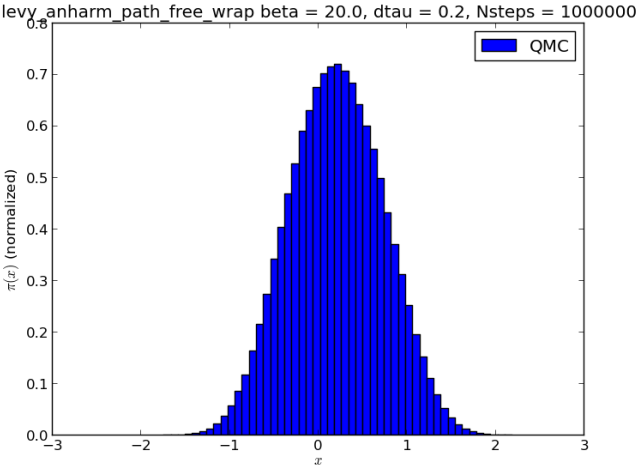
C2 Here, you should evaluate your fellow student's ability to implement a reweighting scheme for levy_free_path for the anharmonic oscillator

There are three issues:
1/ Reweighting by the anharmonic part of the potential for levy_harmonic in the program
2/ Reweighting by the full potential for levy_free in the potential
3/ Implementation as in section **C1**, of the Metropolis algorithm

```
   if LevyType == 'free':
        x_new = levy_free_path(x[0], x[Ncut], dtau, Ncut) + x[Ncut:]
    elif LevyType == 'harm':
        x_new = levy_harmonic_path(x[0], x[Ncut], dtau, Ncut) + x[Ncut:]
    new_Corr_trott = math.exp(sum(- V(a) * dtau for a in x_new))
    if LevyType == 'free':
        new_Corr_trott *= math.exp(sum(- a ** 2 / 2 * dtau for a in x_ne
w))
```

or equivalent
4/ Upload the the two figures, that should resemble those shown here.





An answer obtaining full score is as follows:

1/ The free Levy construction needs reweighting with the entire potential (see code)
2/ The harmonic Levy construction needs reweighting with the anharmonic potential only (see code)
3/ Here is the program
4/ Two Figures, as shown above

```
import math, random, pylab

def V(x): # using only the anharmonic part
    pot =  cubic * x ** 3 + quartic * x ** 4
    return pot

def levy_harmonic_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        Ups1 = 1.0 / math.tanh(dtau) + \
               1.0 / math.tanh(dtau_prime)
        Ups2 = x[k - 1] / math.sinh(dtau) + \
               xend / math.sinh(dtau_prime)
        x.append(random.gauss(Ups2 / Ups1, \
               1.0 / math.sqrt(Ups1)))
    return x

def levy_free_path(xstart, xend, dtau, N):
    x = [xstart]
    for k in range(1, N):
        dtau_prime = (N - k) * dtau
        x_mean = (dtau_prime * x[k - 1] + dtau * xend) / \
                 (dtau + dtau_prime)
        sigma = math.sqrt(1.0 / (1.0 / dtau + 1.0 / dtau_prime))
        x.append(random.gauss(x_mean, sigma))
    return x

LevyType = 'free'   # set this ether to 'free' or to 'harm'
beta = 20.0
cubic = -1.0
quartic = 1.0
N = 100
Ncut = N / 4
dtau = beta / N
delta = 1.0
n_steps = 1000000
x = [0.1] * N
Corr_trott = math.exp(sum(- V(a) * dtau for a in x))
if LevyType == 'free':
    Corr_trott *= math.exp(sum(- a ** 2 / 2 * dtau for a in x))
data = []
n_acc = 0
for step in range(n_steps):
    if LevyType == 'free':
        x_new = levy_free_path(x[0], x[Ncut], dtau, Ncut) + x[Ncut:]
    elif LevyType == 'harm':
        x_new = levy_harmonic_path(x[0], x[Ncut], dtau, Ncut) + x[Ncut:]
    new_Corr_trott = math.exp(sum(- V(a) * dtau for a in x_new))
    if LevyType == 'free':
        new_Corr_trott *= math.exp(sum(- a ** 2 / 2 * dtau for a in x_ne
```

```
w))
    if random.uniform(0.0, 1.0) < new_Corr_trott / Corr_trott:
        n_acc += 1
        Corr_trott = new_Corr_trott
        x = x_new[:]
    x = x[1:] + x[:1]
    k = random.randint(0, N - 1)
    data.append(x[k])
print n_acc / float (n_steps), 'acceptance rate', LevyType
pylab.hist(data, bins=50, normed=True, label='QMC')
pylab.xlabel('$x$')
pylab.ylabel('$\\pi(x)$ (normalized)')
pylab.xlim(-3.0, 3.0)
pylab.legend()
ProgType = 'levy_anharm_path_' + LevyType + '_wrap'
pylab.title(ProgType + ' beta = ' + str(beta) + ', dtau = ' + str(dtau)
+ ', Nsteps = '+ str(n_steps))
pylab.savefig(ProgType + str(beta) + '.png')
pylab.show()
```

POINTS - DESCRIPTION

Give from zero to four points depending on the number of issues treated correctly**:**

1/ Program with correct Reweighting by the anharmonic part of the potential for levy_harmonic.
2/ Program with correct Reweighting by the full potential for levy_free.
3/ Correct implementation of the Metropolis algorithm.
4/ Correct figures uploaded.

Score from your peers: **4**

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

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

peer 3 → *[This area was left blank by the evaluator.]*