Question 1:

CODING: Students in PHYS150 were tasked to measure the local acceleration of gravity g by dropping an object from height H, measuring the time t it takes to reach the ground, and using the formula,

$$H = \frac{1}{2} g t^2. \tag{1}$$

The datafile **`GravityMeasurements.dat`** contains measurements of the value of H in meters, its uncertainty σ_{H_i}, time t in seconds, and its uncertainty σ_{t_i} made independently by $N = 350$ students. The file has five columns -- (1) index $i = 0, \ldots, N-1$, (2) measurement $\{H_i\}$, (3) its corresponding uncertainty $\{\sigma_{H_i}\}$, (4) measurement $\{t_i\}$, and (5) its corresponding uncertainty $\{\sigma_{t_i}\}$ from each student.

```
0   20.066  1.120   2.092  0.084
1   20.363  0.389   2.023  0.050
2   21.498  0.706   1.939  0.115
3   19.709  0.791   1.963  0.118
4   21.192  1.225   2.132  0.097
5   16.631  1.479   1.877  0.112
...
```

(a) (1 pt) Calculate the values of acceleration of gravity $\{g_i\}$.

(b) (2 pts) Calculate the corresponding uncertainties $\{\sigma_{g_i}\}$ using the proper error propagation formulae.

(c) (2 pts) In this part and the next one, ignore the uncertainties $\{\sigma_{g_i}\}$, but rather estimate it directly from the measurements by computing the standard deviation of $\{g_i\}$. What is the best estimate of this value $\tilde{\sigma}_g$?

(d) (2 pts) Using this value of $\tilde{\sigma}_g$, calculate the maximum-likelihood estimate (MLE) of the mean of $\{g_i\}$ and the uncertainty in the mean from the measurements.

(e) (2 pts) Now compute the inverse-variance weighted MLE of the mean of $\{g_i\}$ using the actual uncertainty $\{\sigma_{g_i}\}$ on each measurement. Also compute the uncertainty in the mean to show that it is smaller than the value $\tilde{\sigma}_g$ estimated in (a).

In [1]:
```python
# Problem 1a
def calculate_g(data):
    """Given measurements of H and t, calculate the inferred gravitational acce

    The input height (H) and time (t) are given as data['H'] and data['t'].

    Returns an array with the corresponding g values.
    """
    ### BEGIN SOLUTION
    H = data['H']
```

```
    t = data['t']
    return 2*H / t**2
    ### END SOLUTION
```

In [2]:
```
import numpy as np

data = np.genfromtxt('GravityMeasurements.dat', names=['ID','H','sigma_H','t','

print('The first few rows are:')
for i in range(6):
    print(data[i])
print()

g = calculate_g(data)

print('The first few g values are: ',g[:5])
print('The lowest estimate is ',np.min(g))
print('The highest estimate is ',np.max(g))
print('The mean estimate is ',np.mean(g))

# None of the students had measurements that were more than 30% off.
assert np.allclose(g, 9.8, rtol=0.3)
### BEGIN HIDDEN TESTS

# Not much to do here, just compare to the right answer
#print('Their answer: ',g)
true_g = 2*data['H'] / data['t']**2
#print('Correct answer: ',true_g)

np.testing.assert_allclose(g, true_g)
### END HIDDEN TESTS
```

```
The first few rows are:
(0., 20.066, 1.12, 2.092, 0.084)
(1., 20.363, 0.389, 2.023, 0.05)
(2., 21.498, 0.706, 1.939, 0.115)
(3., 19.709, 0.791, 1.963, 0.118)
(4., 21.192, 1.225, 2.132, 0.097)
(5., 16.631, 1.479, 1.877, 0.112)

The first few g values are:  [ 9.16996004  9.95130395 11.43595496 10.22949009
9.32454266]
The lowest estimate is  7.357945689904639
The highest estimate is  12.505430961137918
The mean estimate is  9.81354685188228
```

In [3]:
```
# Problem 1b
def propagate_sigma_g(data):
    """Calculate the uncertainties in the g measurement based on the reported u

    The input height (H) and time (t) are given as data['H'] and data['t'].
    Their respective estimated uncertainties are data['sigma_H'] and data['sigm

    Returns an array with the corresponding sigma_g values.
    """
    ### BEGIN SOLUTION
    # The formula for g is g = 2H/t^2
```

```
    # The basic error propagation formula is
    # sigma_g^2 = (dg/dH)^2 sigma_H^2 + (dg/dt)^2 sigma_t^2
    #           = (2/t^2)^2 sigma_H^2 + (-4H/t^3)^2 sigma_t^2
    #           = 1/t^6 (4 t^2 sigma_H^2 + 16 H^2 sigma_t^2)
    H = data['H']
    t = data['t']
    # Since the sigmas always show up squared, it's easiest to work with sigsq
    # At the end, we will take the square root.
    sigsq_H = data['sigma_H']**2
    sigsq_t = data['sigma_t']**2

    sigsq_g = (4 * t**2 * sigsq_H + 16 * H**2 * sigsq_t) / t**6
    return np.sqrt(sigsq_g)
    ### END SOLUTION
```

In [4]:

```
sigma_g = propagate_sigma_g(data)

print('The first few calculated uncertainties are: ',sigma_g[:10])
print('The range of sigma_g values is ',np.min(sigma_g),np.max(sigma_g))

# The "pull" is sometimes a useful quantity to look at.
# It is the difference of each measurement from the mean (or expected value) di
# For Gaussian errors, you should expect around 68% of values to have pull betw
# If this isn't the case, it could mean your errors are poorly estimated, or th
# Gaussian, or both.

pull = (g-np.mean(g)) / sigma_g
print('The first few pulls are: ',pull[:5])
frac_lt_1 = np.sum(np.abs(pull) < 1) / len(pull)
print(f'The fraction of points with |pull| < 1 = {frac_lt_1:0.3f}')
### BEGIN HIDDEN TESTS

# Again, just compare to the right answer
#print('Their answer: ',sigma_g)
t = data['t']
H = data['H']
true_sigma_g = (4 * data['t']**2 * data['sigma_H']**2
                + 16 * data['H']**2 * data['sigma_t']**2)**0.5 / data['t']**3
#print('Correct answer: ',true_sigma_g)

np.testing.assert_allclose(sigma_g, true_sigma_g)
### END HIDDEN TESTS
```

```
The first few calculated uncertainties are:  [0.89680364 0.52736391 1.40753685
1.29654819 1.0052088  1.40511361
 0.52327863 1.30351406 0.71850238 0.82814092]
The range of sigma_g values is  0.24308647305001324 2.0710334154258025
The first few pulls are:  [-0.71764518  0.26121829  1.15265764  0.32080816 -0.
48647026]
The fraction of points with |pull| < 1 = 0.671
```

In [5]:

```
# Problem 1c
def estimate_unweighted_sigma_g(data):
    """Estimate an overall estimate of the uncertainties in the g values based
    standard deviation.

    This estimate ignores the students' own estimates of the uncertainties on H
```

```
    just uses their calculated g values.

    Returns a single (not array) value sigma_g.
    """
    # Hints: 1. Use your calculate_g function to get the array of g values.
    #        2. Feel free to use appropriate numpy or scipy functions.

    ### BEGIN SOLUTION
    g = calculate_g(data)
    # Here we calculate it by hand, but the np.std function is fine to use.
    # It's also fine if you don't use the N-1 in the denominator.
    # Using it is technically more correct, but the difference rarely matters i
    var_g = np.sum((g-np.mean(g))**2) / (len(g)-1)
    return np.sqrt(var_g)
    ### END SOLUTION
```

In [6]:

```
unweighted_sigma_g = estimate_unweighted_sigma_g(data)

print(f'The estimated unweighted sigma_g is {unweighted_sigma_g:.4f}')
print(f'Compare this to the mean propagated sigma_g: {np.mean(sigma_g):.4f}')
### BEGIN HIDDEN TESTS

# Note: Allow either N or N-1 in the denominator.
g = calculate_g(data)
num = np.sqrt(np.sum((g - np.mean(g))**2))
true_sigma_g_1 = num / (len(g)-1)**0.5
true_sigma_g_2 = num / (len(g))**0.5

print('Their answer: ',unweighted_sigma_g)
print('Valid correct answers: ',true_sigma_g_1, true_sigma_g_2)
assert np.isclose(unweighted_sigma_g, true_sigma_g_1) or np.isclose(unweighted_

### END HIDDEN TESTS
```

```
The estimated unweighted sigma_g is 0.8511
Compare this to the mean propagated sigma_g: 0.8096
Their answer:  0.8510884935927929
Valid correct answers:  0.8510884935927928 0.8498717831871401
```

In [7]:

```
# Problem 1d
def calculate_unweighted_mle_meang(data):
    """Estimate the unweighted maximum-likelihood estimate of <g> and its uncer

    This estimate ignores the students' own estimates of the uncertainties on h

    Returns a tuple of two values: <g>, sigma_<g>
    """
    # Hints: 1. Use calculate_g to get the array of g values.
    #        2. Use estimate_unweighted_sigma_g for sigma_g.
    #        3. Feel free to use appropriate numpy or scipy functions.

    ### BEGIN SOLUTION
    g = calculate_g(data)
    sigma_g = estimate_unweighted_sigma_g(data)

    meang = np.mean(g)
    sigma_meang = sigma_g / len(g)**0.5
```

```
    return meang, sigma_meang
    ### END SOLUTION
```

In [8]:
```
meang, sigma_meang = calculate_unweighted_mle_meang(data)

print('Using unweighted maximum likelihood,')
print(f'the mean estimate of g is {meang:0.4f} +- {sigma_meang:0.4f}')
### BEGIN HIDDEN TESTS

# This code snippet lets us just log the test failures so we can see all the th
# bomb out on the first failure.  Then at the end, we just assert that there we
# automatic grading to give points.
nfail=0
from contextlib import contextmanager
import traceback
@contextmanager
def log_assert():
    global nfail
    try:
        yield
    except AssertionError as e:
        print('Failed assert:')
        print(traceback.format_stack()[-3].split('\n')[1])
        print('    msg =',str(e))
        nfail += 1

print('Their answer: ',meang, sigma_meang)
g = calculate_g(data)
unweighted_sigma_g = estimate_unweighted_sigma_g(data)
true_meang = np.mean(g)
print('Correct answer for mean: ',true_meang)
with log_assert():
    assert np.isclose(meang, true_meang), (meang, true_meang)

# Note: Allow either N or N-1 in the denominator.
true_sigma_meang_1 = unweighted_sigma_g / (len(g)-1)**0.5
true_sigma_meang_2 = unweighted_sigma_g / len(g)**0.5

print('Valid correct answers for sigma: ',true_sigma_meang_1, true_sigma_meang_
with log_assert():
    assert np.isclose(sigma_meang, true_sigma_meang_1) or np.isclose(sigma_mear
        (sigma_meang, true_sigma_meang_1, true_sigma_meang_2)

print(f"\nTotal of {nfail} test failures")
assert nfail == 0
### END HIDDEN TESTS
```

```
Using unweighted maximum likelihood,
the mean estimate of g is 9.8135 +- 0.0455
Their answer:  9.81354685188228 0.04549259355499685
Correct answer for mean:  9.81354685188228
Valid correct answers for sigma:  0.04555772256981239 0.04549259355499685

Total of 0 test failures
```

In [9]:
```
# Problem 1e
def calculate_weighted_mle_meang(data):
```

```
    """Estimate the inverse-variance weighted maximum-likelihood estimate of <g

    This estimate uses the students' own estimates of the uncertainties on H ar
    into a separate estimate of sigma_g for each data point.

    Returns a tuple of two values: <g>, sigma_<g>
    """
    # Hints: 1. Use calculate_g to get the array of g values.
    #        2. Use propagate_sigma_g for sigma_g.
    #        3. Feel free to use appropriate numpy or scipy functions.

    ### BEGIN SOLUTION
    g = calculate_g(data)
    sigma_g = propagate_sigma_g(data)

    # The formula is <g> = Sum(g/sigma_g^2) / Sum(1/sigma_g^2)
    num = np.sum(g/sigma_g**2)
    denom = np.sum(1/sigma_g**2)
    meang = num / denom

    # The uncertainty in the mean is sigma_<g>^2 = 1/Sum(1/sigma_g^2))
    sigma_meang = 1. / np.sum(1. / sigma_g**2)**0.5
    return meang, sigma_meang
    ### END SOLUTION
```

In [10]:

```
meang, sigma_meang = calculate_weighted_mle_meang(data)

print('Using weighted maximum likelihood,')
print(f'The mean estimate of g is {meang:0.4f} +- {sigma_meang:0.4f}')

### BEGIN HIDDEN TESTS

print('Their answer: ',meang, sigma_meang)
g = calculate_g(data)
sigma_g = propagate_sigma_g(data)
num = np.sum(g/sigma_g**2)
denom = np.sum(1/sigma_g**2)
true_meang = num / denom
true_sigma_meang = 1. / np.sum(1. / sigma_g**2)**0.5

print('Correct answer for mean: ',true_meang)
with log_assert():
    assert np.isclose(meang, true_meang), meang

print('Correct answer for sigma: ',true_sigma_meang)
with log_assert():
    assert np.isclose(sigma_meang, true_sigma_meang), sigma_meang

print(f"\nTotal of {nfail} test failures")
assert nfail == 0
### END HIDDEN TESTS
```

```
Using weighted maximum likelihood,
The mean estimate of g is 9.7531 +- 0.0306
Their answer:  9.753074085349953 0.03057998018874849
Correct answer for mean:  9.753074085349953
Correct answer for sigma:  0.03057998018874849

Total of 0 test failures
```

Question 2:

CODING: Using the same dataset as above:

(a) (3 pts) Calculate the χ^2 value defined as:

$$\chi^2 = \sum_{i=0}^{N-1} \left(\frac{g_i - \mu'}{\sigma_{g_i}} \right)^2 \tag{2}$$

where μ' is the inverse-variance weighted MLE of the mean of $\{g_i\}$ from 1(c).

(b) (4 pts) Now calculate the χ^2 value as a function of μ' from $\mu' = 9.6$ to 10.0 in steps of 0.001 and show that the value of μ' that minimizes the χ^2 is indeed given by the answer from 1(c).

(c) (4 pts) Finally, determine the lower and upper values of μ' at which the χ^2 is larger than the minimum value by 1.00.
(NOTE: These values should match the $\pm 1\sigma$ range from the MLE in part 1(e) and is another way to perform parameter estimation using the χ^2 statistic.)

(d) **EXTRA CREDIT**: (2 pts) Make a single plot that shows all of these facts with appropriate labels to show the important features.

In [11]:

```python
# Problem 2a
def calculate_mle_chisq(data):
    """Calculate chisq for the inverse-variance weighted MLE estimate of mu'.

    The sigma_g values are based on the students' estimated uncertainties sigma

    Returns chisq
    """
    # Hints: 1. Use calculate_g to get the array of g values.
    #        2. Use propagate_sigma_g for sigma_g.
    #        3. Use calculate_weighted_mle_meang to get mu'.

    ### BEGIN SOLUTION
    g = calculate_g(data)
    sigma_g = propagate_sigma_g(data)

    # Note: a common python-ism for ignoring one or more return values in a tup
    # In this case, we don't need sigma_meang, so just ignore it.
    mu, _ = calculate_weighted_mle_meang(data)

    chisq = np.sum(((g-mu)/sigma_g)**2)
```

```
    return chisq
    ### END SOLUTION
```

In [12]:
```
chisq = calculate_mle_chisq(data)

print(f'The chisq estimate for the MLE estimate of <g> is {chisq:.2f}')
print('This should be roughly comparable to the number of data points: ',len(da

### BEGIN HIDDEN TESTS

print('Their answer: ',chisq)
g = calculate_g(data)
sigma_g = propagate_sigma_g(data)
mu, _ = calculate_weighted_mle_meang(data)
true_chisq = np.sum(((g-mu)/sigma_g)**2)

print('Correct answer: ',true_chisq)
assert np.isclose(chisq, true_chisq), (chisq, true_chisq)
### END HIDDEN TESTS
```

```
The chisq estimate for the MLE estimate of <g> is 376.27
This should be roughly comparable to the number of data points:  350
Their answer:  376.27029481662527
Correct answer:  376.27029481662527
```

In [13]:
```
# Problem 2b
def calculate_chisq_range(data, min_mu, max_mu):
    """Calculate chisq over a range of mu values from min_mu to max_mu in steps

    This function will generate an array of mu values over the given range.
    For each value of mu, it will calculate the corresponding chisq value.

    The sigma_g values are based on the students' estimated uncertainties sigma

    Note: The output mu values should be monotonically increasing.

    Returns mu_array, chisq_array as arrays of equal length.
    """
    # Hints: 1. Use calculate_g to get the array of g values.
    #        2. Use propagate_sigma_g for sigma_g.

    ### BEGIN SOLUTION
    g = calculate_g(data)
    sigma_g = propagate_sigma_g(data)
    delta = 0.001

    npoints = int(np.rint((max_mu-min_mu)/delta + 1))  # rint rounds to the nea
    mu_array = np.linspace(min_mu, max_mu, npoints, endpoint=True)
    chisq_array = np.array([np.sum(((g-mu)/sigma_g)**2) for mu in mu_array])
    return mu_array, chisq_array
    ### END SOLUTION

def find_minimum_chisq(mu_array, chisq_array):
    """Find the minimum chisq and its corresponding mu, given arrays of each.

    Returns mu_minimum, chisq_minimum
    """
```

```
    # Hint: np.min(chisq_array) will return the value of the minimum.  There is
    #       numpy function that will instead give you the index of the minimum,
    #       will let you access the corresponding element from mu_array.

    ### BEGIN SOLUTION
    min_index = np.argmin(chisq_array)
    mu_minimum = mu_array[min_index]
    chisq_minimum = chisq_array[min_index]
    return mu_minimum, chisq_minimum
    ### END SOLUTION
```

In [14]:

```
mu_array, chisq_array = calculate_chisq_range(data, 9.6, 10.0)
mu_minimum, chisq_minimum = find_minimum_chisq(mu_array, chisq_array)

print(f'The minimum chisq in steps of 0.001 is {chisq_minimum:.4f}')
print(f'mu at the minimum is {mu_minimum:.3f}')
assert chisq_minimum == np.min(chisq_array)

# Do this again for comparison.
chisq_mle = calculate_mle_chisq(data)

print('\nFor comparison:')
print(f'chisq at the MLE solution is {chisq_mle:.3f}')
print(f'The MLE estimate of mu is {calculate_weighted_mle_meang(data)[0]:.3f}')
assert np.isclose(chisq_minimum, chisq_mle, rtol=1.e-3)
### BEGIN HIDDEN TESTS

# Check that mu_array has right range and step size
with log_assert():
    assert np.isclose(np.min(mu_array), 9.6), np.min(mu_array)

with log_assert():
    assert np.isclose(np.max(mu_array), 10.0), np.max(mu_array)

with log_assert():
    assert np.allclose(np.diff(mu_array), 1.e-3), np.diff(mu_array)

# Check that chisq is correct
g = calculate_g(data)
sigma_g = propagate_sigma_g(data)
correct_chisq_array = np.array([np.sum(((g-mu)/sigma_g)**2) for mu in mu_array]
#print('Their chisq_array = ',chisq_array[:10])
#print('Correct chisq_array = ',correct_chisq_array[:10])
with log_assert():
    assert np.allclose(chisq_array, correct_chisq_array, rtol=1.e-3), chisq_arr

# Check that mu_minimum is right.
with log_assert():
    assert np.all(chisq_minimum <= chisq_array)
with log_assert():
    assert mu_minimum in mu_array
#print('mu_array = ',mu_array)
i_min = np.where(mu_array == mu_minimum)
#print('i_min = ',i_min)
assert mu_array[i_min] == mu_minimum  # This is what where is supposed to mean.
with log_assert():
    assert np.isclose(chisq_minimum, chisq_array[i_min]), (chisq_minimum, chisc
```

```
print(f"\nTotal of {nfail} test failures")
assert nfail == 0
### END HIDDEN TESTS
```

```
The minimum chisq in steps of 0.001 is 376.2703
mu at the minimum is 9.753

For comparison:
chisq at the MLE solution is 376.270
The MLE estimate of mu is 9.753

Total of 0 test failures
```

In [15]:

```
# Problem 2c:
def find_one_sigma_range(mu_array, chisq_array):
    """Find a range of mu values where chisq < min_chisq + 1.0, given arrays of

    Note: the input values in mu_array may be assumed to be monotonically incre

    Returns min_mu, max_mu.
    """
    ### BEGIN SOLUTION
    min_chisq = np.min(chisq_array)
    good_mus = mu_array[chisq_array < min_chisq + 1.0]
    return good_mus[0], good_mus[-1]
    ### END SOLUTION
```

In [16]:

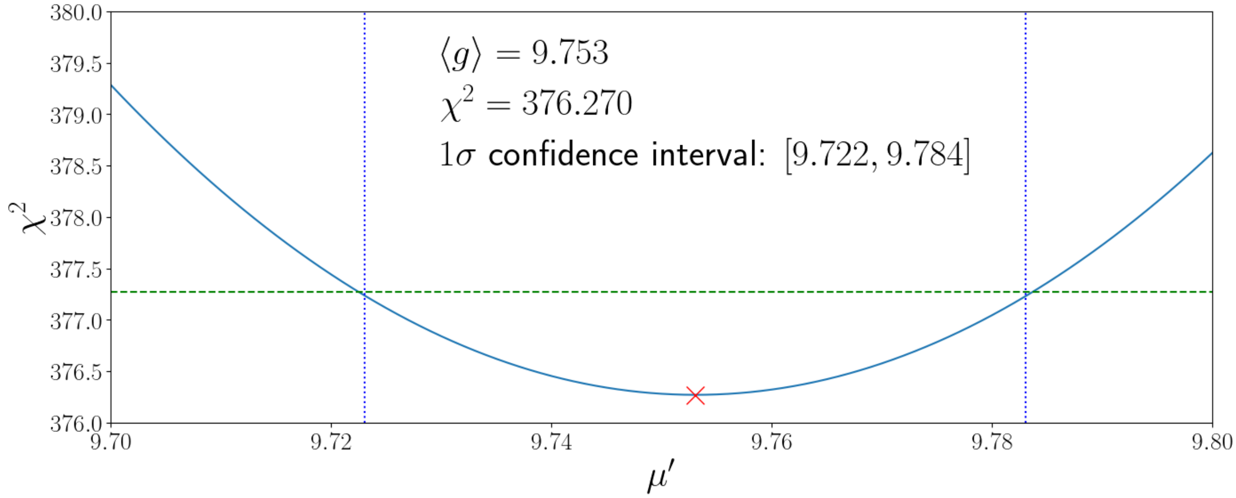
```
min_mu, max_mu = find_one_sigma_range(mu_array, chisq_array)
print(f'Range where chiq < ({chisq_minimum:.2f} + 1) is {min_mu:.3f} < mu < {ma

print('\nFor comparison with 1(e):')
meang, sigma_meang = calculate_weighted_mle_meang(data)
print(f'The MLE estimate would predict {meang-sigma_meang:.3f} < mu < {meang+si
### BEGIN HIDDEN TESTS

print('Their answers = ',min_mu, max_mu)
# Check that all mu in their range have
with log_assert():
    assert np.isclose(np.min(mu_array), 9.6), np.min(mu_array)

with log_assert():
    assert np.isclose(np.max(mu_array), 10.0), np.max(mu_array)

with log_assert():
    assert np.allclose(np.diff(mu_array), 1.e-3), np.diff(mu_array)

# Check that chisq is correct
g = calculate_g(data)
sigma_g = propagate_sigma_g(data)
correct_chisq_array = np.array([np.sum(((g-mu)/sigma_g)**2) for mu in mu_array]
#print('Their chisq_array = ',chisq_array[:10])
#print('Correct chisq_array = ',correct_chisq_array[:10])
with log_assert():
    assert np.allclose(chisq_array, correct_chisq_array, rtol=1.e-3), (chisq_ar

# Check that mu_minimum is right.
```

```
with log_assert():
    assert np.all(chisq_minimum <= chisq_array)
with log_assert():
    assert mu_minimum in mu_array
i_min = np.where(mu_array == mu_minimum)
assert mu_array[i_min] == mu_minimum  # This is what where means!
with log_assert():
    assert np.isclose(chisq_minimum, chisq_array[i_min]), (chisq_minimum, chisc

print(f"\nTotal of {nfail} test failures")
assert nfail == 0
### END HIDDEN TESTS
```

```
Range where chiq < (376.27 + 1) is 9.723 < mu < 9.783

For comparison with 1(e):
The MLE estimate would predict 9.722 < mu < 9.784
Their answers =  9.722999999999999 9.783

Total of 0 test failures
```

In [17]:

```
# Problem 2d (EXTRA CREDIT)

### BEGIN SOLUTION
# Note to TAs: They don't need to use latex for their labels, but they should g
# And since this was open ended, allow some variety in how they label things.
# some way the minimum chisq and 1 sigma range.

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

# These setup bits just make the plot a bit nicer.  They aren't required to get
matplotlib.rc('xtick', labelsize=20) # Increase some of the font sizes from the
matplotlib.rc('ytick', labelsize=20)
matplotlib.rc('font', size=30)
matplotlib.rc('text', usetex=True)  # Allow LaTex in plot labels and in-plot te

fig, ax = plt.subplots(1, figsize=(16,6))

ax.set_xlim(9.70, 9.80)
ax.set_ylim(376, 380)
ax.plot(mu_array, chisq_array)
# Note: You don't need to use latex for the labels, but you do need to give som
ax.set_xlabel(r'$\mu^\prime$')
ax.set_ylabel(r'$\chi^2$')
ax.plot(mu_minimum, chisq_minimum, color='red', marker='x', markersize=15)  # D
ax.axhline(chisq_minimum+1, color='green', ls='--')  # Draw horizontal line at
ax.axvline(min_mu, color='blue', ls=':')  # Draw vertical lines at min_mu, max_
ax.axvline(max_mu, color='blue', ls=':')
ax.text(9.73, 379.5, r"$\langle g \rangle = {:0.3f}$".format(meang))
ax.text(9.73, 379, r"$\chi^2 = {:0.3f}$".format(chisq_mle))
ax.text(9.73, 378.5, r"$1 \sigma$ confidence interval: $[{:0.3f}, {:0.3f}]$".fo

plt.show()
### END SOLUTION
```


$\langle g \rangle = 9.753$
$\chi^2 = 376.270$
1σ confidence interval: $[9.722, 9.784]$
χ^2
μ'
380.0
379.5
379.0
378.5
378.0
377.5
377.0
376.5
376.0
9.70
9.72
9.74
9.76
9.78
9.80

```
In [ ]:
```