In [1]:
```python
import torch
import numpy as np
import json
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import importlib

from evaluation_based_sampling import evaluate, evaluate_program
from daphne import daphne

from graph_based_sampling import sample_from_joint
```

## Problem 2

10k samples in 1.59s implies 384k samples

In [2]:
```python
from load_helper import ast_helper, graph_helper
```

### Importance sampling

In [3]:
```python
import parse
import importance_sampling
import importlib
importlib.reload(parse)
```

Out[3]: `<module 'parse' from '/Users/geoffwoollard/repos/prob_prog/hw/hw3/parse.py'>`

In [4]:
```python
fname = '2.daphne'
ast = ast_helper(fname)
```

In [5]:
```python
%%time
num_samples=384000
samples, sigmas = parse.time_samples(num_samples,ast=ast)
```

```
CPU times: user 10min 28s, sys: 919 ms, total: 10min 29s
Wall time: 10min 29s
```

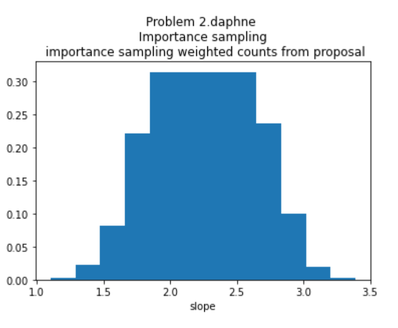
In [6]:
```python
samples = np.array([sample.tolist() for sample in samples])
```

In [7]:
```python
posterior_mean, probs = importance_sampling.weighted_average(samples,sigmas,reshape_probs=(-1,1),axis=0)
posterior_mean
```

Out[7]:
```
array([ 2.15899032, -0.53834021])
```

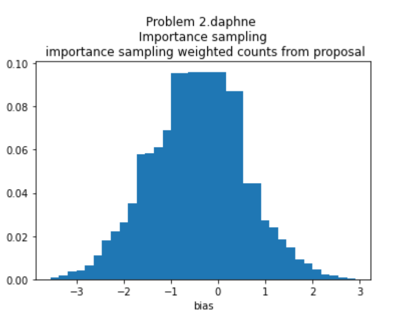
In [8]:
```python
counts_bins = np.histogram(samples[:,0], weights=probs, bins=500)
counts, bins = counts_bins[0], counts_bins[1]
idx = (counts > np.quantile(counts,1-.05))
plt.bar(bins[1:][idx],counts[idx])
plt.title('Problem {} \n Importance sampling \n importance sampling weighted counts from proposal'.format(fname.split('.')[0]))
plt.ylabel('Counts')
plt.xlabel('slope')
```

Out[8]: `Text(0.5, 0, 'slope')`



In [9]:
```python
counts_bins = np.histogram(samples[:,1], weights=probs, bins=500)
counts, bins = counts_bins[0], counts_bins[1]
idx = (counts > np.quantile(counts,1-.05))
plt.bar(bins[1:][idx],counts[idx])
plt.title('Problem {} \n Importance sampling \n importance sampling weighted counts from proposal'.format(fname.split('.')[0]))
plt.ylabel('Counts')
plt.xlabel('bias')
```

Out[9]: `Text(0.5, 0, 'bias')`



In [10]:
```python
expectation_samples_2, probs = importance_sampling.weighted_average(samples**2,sigmas,reshape_probs=(-1,1),axis=0)
posterior_variance = expectation_samples_2 - posterior_mean**2
posterior_variance
```

Out[10]:
```
array([0.05573231, 0.79199933])
```

In [11]:
```python
expectation_samples_xy, probs = importance_sampling.weighted_average(samples[:,0]*samples[:,1],sigmas)
covariance = expectation_samples_xy - posterior_mean[0]*posterior_mean[1]
covariance
```

Out[11]:
```
-0.18441316191640498
```

In [12]:
```python
for result in [
    "{} importance sampling: posterior mean slope {:1.3f} | variance slope {:1.3e}".format(fname,posterior_mean[0],posterior_variance[0]),
    "{} importance sampling: posterior mean bias {:1.3f} | variance bias {:1.3e}".format(fname,posterior_mean[1],posterior_variance[1]),
    "{} importance sampling: posterior covariance of slope and bias variance bias {:1.3e}".format(fname,covariance),

]:
    print(result)
```

```
2.daphne importance sampling: posterior mean slope 2.159 | variance slope 5.574e-02
2.daphne importance sampling: posterior mean bias -0.538 | variance bias 7.920e-01
2.daphne importance sampling: posterior covariance of slope and bias variance bias -1.844e-01
```

Numpy contains it's own method for computing this, and we can check it agrees with our results (where things are spelt out a bit more for learning purposes).

In [13]:
```python
np.cov(samples.T,aweights=probs,ddof=0)
```

Out[13]:
```
array([[ 0.05573231, -0.18443133],
       [-0.18443133,  0.79199933]])
```

### MH Gibbs

5k samples in 21.3s implies 140k samples in 10 min

In [12]:
```python
import mh_gibbs
from mh_gibbs import mh_gibbs_wrapper, compute_log_joint_prob
importlib.reload(mh_gibbs)
```

Out[12]: `<module 'mh_gibbs' from '/Users/geoffwoollard/repos/prob_prog/hw/hw3/mh_gibbs.py'>`

In [13]:
```python
fname = '2.daphne'
graph = graph_helper(fname)
graph
```

Out[13]:

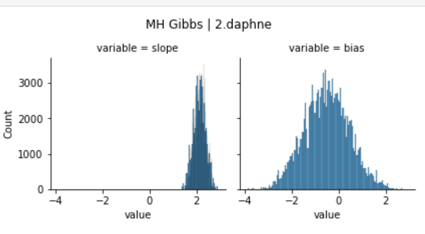
```
[{'observe-data': ['fn',
   ['_', 'data', 'slope', 'bias'],
   ['let',
    ['xn', ['first', 'data']],
    ['let',
     ['yn', ['second', 'data']],
     ['let',
      ['zn', ['+', ['*', 'slope', 'xn'], 'bias']],
      ['let',
       ['dontcare9', ['observe', ['normal', 'zn', 1.0], 'yn']],
       ['rest', ['rest', 'data']]]]]]]},
 {'V': ['observe3',
   'observe6',
   'observe4',
   'observe7',
   'sample2',
   'sample1',
   'observe8',
   'observe5'],
  'A': {'sample2': ['observe3',
    'observe6',
    'observe4',
    'observe7',
    'observe8',
    'observe5'],
   'sample1': ['observe3',
    'observe6',
    'observe4',
    'observe7',
    'observe8',
    'observe5']},
  'P': {'sample1': ['sample*', ['normal', 0.0, 10.0]],
   'sample2': ['sample*', ['normal', 0.0, 10.0]],
   'observe3': ['observe*',
    ['normal', ['+', ['*', 'sample1', 1.0], 'sample2'], 1.0],
    2.1],
   'observe4': ['observe*',
    ['normal', ['+', ['*', 'sample1', 2.0], 'sample2'], 1.0],
    3.9],
   'observe5': ['observe*',
    ['normal', ['+', ['*', 'sample1', 3.0], 'sample2'], 1.0],
    5.3],
   'observe6': ['observe*',
    ['normal', ['+', ['*', 'sample1', 4.0], 'sample2'], 1.0],
    7.7],
   'observe7': ['observe*',
    ['normal', ['+', ['*', 'sample1', 5.0], 'sample2'], 1.0],
    10.2],
   'observe8': ['observe*',
    ['normal', ['+', ['*', 'sample1', 6.0], 'sample2'], 1.0],
    12.9]},
  'Y': {'observe3': 2.1,
   'observe4': 3.9,
   'observe5': 5.3,
   'observe6': 7.7,
   'observe7': 10.2,
   'observe8': 12.9}},
 ['vector', 'sample1', 'sample2']]
```


In [14]:
```python
%%time
num_steps=140000
return_list, samples_whole_graph = mh_gibbs.mh_gibbs_wrapper(graph,num_steps)
```

```
CPU times: user 10min 14s, sys: 2.36 s, total: 10min 16s
Wall time: 10min 20s
```

In [15]:
```python
samples = np.array([sample.tolist() for sample in return_list])
```

In [16]:
```python
burn_in = int(0.01*num_steps)
df = pd.DataFrame(samples[burn_in:])
df.columns = ['slope','bias']
df['iteration'] = df.index
tall = pd.melt(df, id_vars='iteration')

g = sns.FacetGrid(tall, col="variable")
g.map(sns.histplot, "value")

plt.suptitle('MH Gibbs | {}'.format(fname))
plt.subplots_adjust(top=0.8)
```



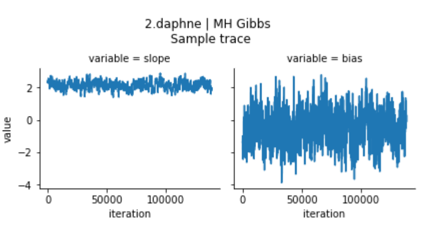
In [17]:
```python
posterior_mean = samples[burn_in:].mean(0)
cov_matrix = np.cov(samples[burn_in:].T,ddof=0)
posterior_variance = samples[burn_in:].var(0)
covariance = cov_matrix[0,1]
assert np.isclose(cov_matrix[0,0],posterior_variance[0])
assert np.isclose(cov_matrix[1,1],posterior_variance[1])
```

In [18]:
```python
for result in [
    "{} MH Gibbs: posterior mean slope {:1.3f} | variance slope {:1.3e}".format(fname,posterior_mean[0],posteri
    "{} MH Gibbs: posterior mean bias {:1.3f} | variance bias {:1.3e}".format(fname,posterior_mean[1],posterior
    "{} MH Gibbs: posterior covariance of slope and bias variance bias {:1.3e}".format(fname,covariance),

]:
    print(result)
```

```
2.daphne MH Gibbs: posterior mean slope 2.149 | variance slope 5.918e-02
2.daphne MH Gibbs: posterior mean bias -0.515 | variance bias 8.965e-01
2.daphne MH Gibbs: posterior covariance of slope and bias variance bias -2.070e-01
```

In [19]:
```python
g = sns.FacetGrid(tall, col="variable")
g.map(plt.plot, "iteration","value")

plt.suptitle('MH Gibbs | {}'.format(fname))
plt.subplots_adjust(top=0.75)
plt.suptitle('{} | MH Gibbs \n Sample trace'.format(fname))
```

Out[19]: `Text(0.5, 0.98, '2.daphne | MH Gibbs \n Sample trace')`

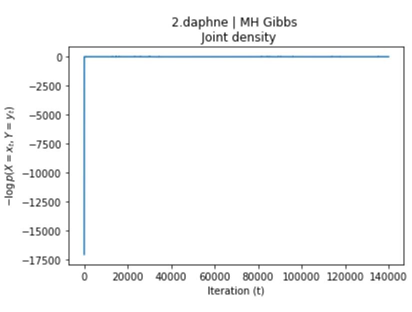


In [20]:
```python
G = graph[1]
Y = G['Y']
Y = {key:evaluate([value])[0] for key,value in Y.items()}
P = G['P']
```

In [21]:
```python
size = len(samples_whole_graph)
jll = np.zeros(size)
for idx in range(size):
    jll[idx] = compute_log_joint_prob(samples_whole_graph[idx],Y,P)
```

In [22]:
```python
pd.Series(jll).plot()
plt.xlabel('Iteration (t)')
plt.ylabel(r'$-\log p(X=x_t,Y=y_t)$')
plt.title('{} | MH Gibbs \n Joint density'.format(fname))
```

Out[22]: `Text(0.5, 1.0, '2.daphne | MH Gibbs \n Joint density')`



### HMC

4.2s/ 200 samples implies 28.5k samples in 10 min 1k samples in 21.3 s implies 28k samples

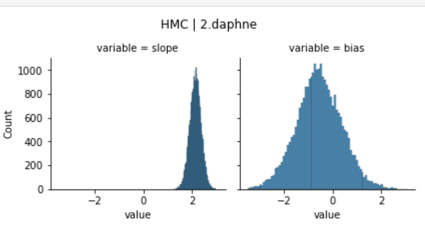
In [143…
```python
fname = '2.daphne'
graph = graph_helper(fname)
```

In [144…
```python
import hmc
importlib.reload(hmc)
from hmc import hmc_wrapper
```

In [ ]:
```python
num_samples=28000
return_list, samples_whole_graph = hmc_wrapper(graph,num_samples,T=20,epsilon=0.01)
```

In [154…
```python
samples = np.array([sample.tolist() for sample in return_list])
```

In [155…
```python
burn_in = int(0.01*num_samples)
df = pd.DataFrame(samples[burn_in:])
df.columns = ['slope','bias']
df['iteration'] = df.index
tall = pd.melt(df, id_vars='iteration')

g = sns.FacetGrid(tall, col="variable")
g.map(sns.histplot, "value")

plt.suptitle('HMC | {}'.format(fname))
plt.subplots_adjust(top=0.8)
```



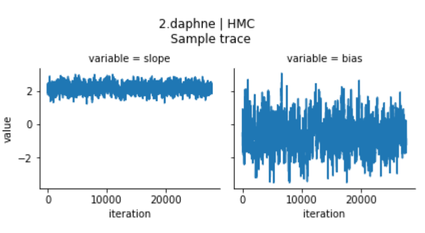
In [156…
```python
posterior_mean = samples[burn_in:].mean(0)
cov_matrix = np.cov(samples[burn_in:].T,ddof=0)
posterior_variance = samples[burn_in:].var(0)
covariance = cov_matrix[0,1]
assert np.isclose(cov_matrix[0,0],posterior_variance[0])
assert np.isclose(cov_matrix[1,1],posterior_variance[1])
```

In [157…
```python
for result in [
    "{} HMC: posterior mean slope {:1.3f} | variance slope {:1.3e}".format(fname,posterior_mean[0],posterior_va
    "{} HMC: posterior mean bias {:1.3f} | variance bias {:1.3e}".format(fname,posterior_mean[1],posterior_vari
    "{} HMC: posterior covariance of slope and bias variance bias {:1.3e}".format(fname,covariance),

]:
    print(result)
```

```
2.daphne HMC: posterior mean slope 2.161 | variance slope 5.652e-02
2.daphne HMC: posterior mean bias -0.556 | variance bias 8.609e-01
2.daphne HMC: posterior covariance of slope and bias variance bias -1.984e-01
```

In [158…
```python
g = sns.FacetGrid(tall, col="variable")
g.map(plt.plot, "iteration","value")

plt.suptitle('HMC | {}'.format(fname))
plt.subplots_adjust(top=0.75)
plt.suptitle('{} | HMC \n Sample trace'.format(fname))
```

Out[158… `Text(0.5, 0.98, '2.daphne | HMC \n Sample trace')`



In [159…
```python
G = graph[1]
Y = G['Y']
Y = {key:evaluate([value])[0] for key,value in Y.items()}
P = G['P']
```

In [160…
```python
size = len(samples_whole_graph)
jll = np.zeros(size)
for idx in range(size):
    jll[idx] = compute_log_joint_prob(samples_whole_graph[idx],Y,P)
```

In [161…
```python
pd.Series(jll).plot()
plt.xlabel('Iteration (t)')
plt.ylabel(r'$-\log p(X=x_t,Y=y_t)$')
plt.title('{} | HMC \n Joint density'.format(fname))
```

Out[161… `Text(0.5, 1.0, '2.daphne | HMC \n Joint density')`