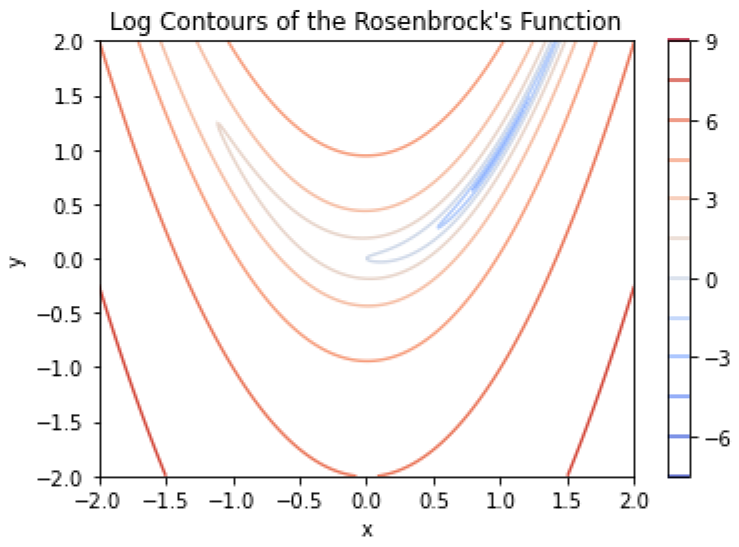
In [373…
```
#Nessecary libraries
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as op
from chainconsumer import ChainConsumer
import corner
import os
import logging
import pickle
import stan
```

Question 1

In [374…
```
#create a grid from -2 to 2 in both directions with 250 points
x = np.linspace(-2, 2, 250)
y = np.linspace(-2, 2, 250)

xx, yy = np.meshgrid(x, y)


#define the func
def rosen(x,y):
    return (1-x)**2 +100*(y - x**2)**2

#evaluate it
z = rosen(xx,yy)

#plot the contours
fig, ax = plt.subplots()
contour = ax.contour(xx, yy, np.log(z), levels=10, cmap='coolwarm')
fig.colorbar(contour)
ax.set_title("Log Contours of the Rosenbrock's Function")
ax.set_xlabel("x")
ax.set_ylabel("y")
plt.show()
```



Question 2

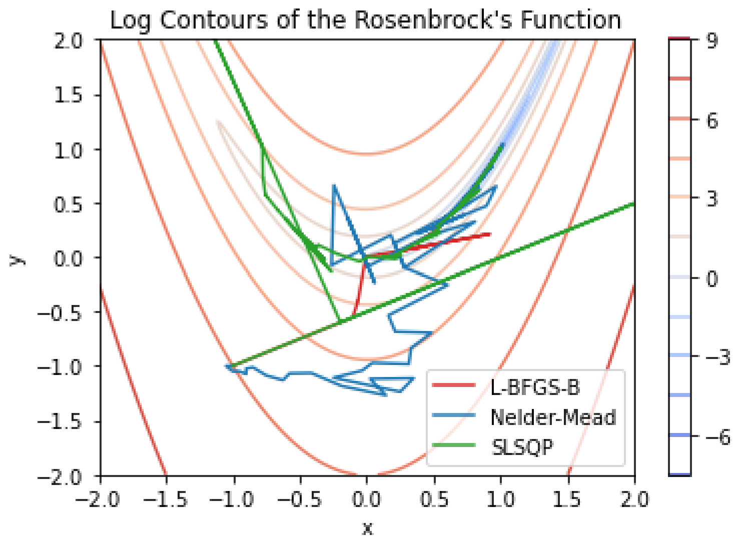
In [375…
```
#prepare to record theta.
#such that theta is appended to a list when called
```

```
#use this in the optimisation.


thetas = []
def objective_function(theta):
    thetas.append(theta)
    return rosen(*theta)
```

In [376…

```
#define method and colours used in the loop
methods = ['L-BFGS-B', 'Nelder-Mead', 'SLSQP']
colour = ['tab:red', 'tab:blue', 'tab:green']

#set up the background of the plot
fig, ax = plt.subplots()
contour = ax.contour(xx, yy, np.log(z), levels=10, cmap='coolwarm')
fig.colorbar(contour)
ax.set_title("Log Contours of the Rosenbrock's Function")
ax.set_xlabel("x")
ax.set_ylabel("y")

#loop optimisation for each method
for i in np.arange(3):
    #optimise for initial position (-1,-1)
    thetas = []
    result = op.minimize(
        objective_function,
        [-1,-1],
        method=methods[i],
        bounds=[
            (None, None),
            (None, None)
        ]
    )
    minimum = rosen(*result.x)
    #print the results
    print(f'The minimum of the Rosenbrocks function using the {methods[i]} was found to
    thetas = np.array(thetas).T
    #plot the paths
    ax.plot(thetas[0], thetas[1], label=methods[i], c=colour[i])

plt.xlim(-2,2)
plt.ylim(-2,2)
plt.legend()
plt.show()
```

```
The minimum of the Rosenbrocks function using the L-BFGS-B was found to be 9.12925010194
1865e-12 corresponding to the points [0.99999698 0.99999395] (x,y)
The minimum of the Rosenbrocks function using the Nelder-Mead was found to be 5.30934391
8637161e-10 corresponding to the points [0.99999886 0.99999542] (x,y)
The minimum of the Rosenbrocks function using the SLSQP was found to be 6.36627528136865
2e-09 corresponding to the points [0.9999628  0.99991854] (x,y)
```



Question 3

The code below generates fake data that is drawn from

$$y \sim \mathcal{N}\left(\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3, \sigma_y\right)$$

In [377…

```
# this generates random data

np.random.seed(0)
N = 30
x = np.random.uniform(0, 100, N)
theta = np.random.uniform(-1e-3, 1e-3, size=(4, 1))
# Define the design matrix.
A = np.vstack([
np.ones(N),
x,
x**2,
x**3
]).T
y_true = (A @ theta).flatten()
y_err_intrinsic = 10 # MAGIC number!
y_err = y_err_intrinsic * np.random.randn(N)
y = y_true + np.random.randn(N) * y_err
y_err = np.abs(y_err)
```

Now assume that the data was generated from each of the following model:

$$y \sim \mathcal{N}\left(\theta_0, \sigma_y\right)$$

$$y \sim \mathcal{N}\left(\theta_0 + \theta_1 x, \sigma_y\right)$$

$$y \sim \mathcal{N}\left(\theta_0 + \theta_1 x + \theta_2 x^2, \sigma_y\right)$$

$$y \sim \mathcal{N}\left(\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3, \sigma_y\right)$$

$$y \sim \mathcal{N}\left(\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4, \sigma_y\right)$$

$$y \sim \mathcal{N}\left(\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5, \sigma_y\right)$$

Recal that the linear algebra solution follows:

$$X = \left[A^\top C^{-1} A\right]^{-1} \left[A^\top C^{-1} Y\right] \quad .$$

Where

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \cdots & \cdots & \cdots & \ddots & \cdots \\ 1 & x_N & x_N^2 & \cdots & x_N^n \end{bmatrix}$$

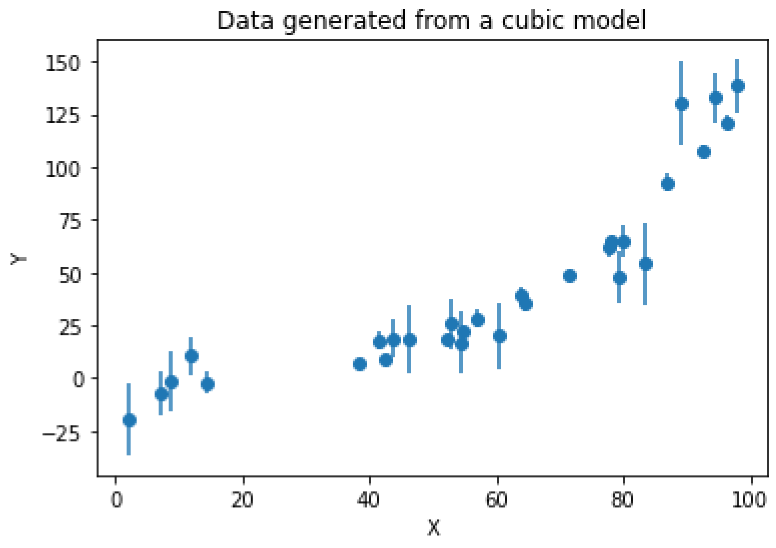
$$C = \begin{bmatrix} \sigma_{y1}^2 & 0 & \cdots & 0 \\ 0 & \sigma_{y2}^2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & \sigma_{yN}^2 \end{bmatrix}$$

The design matrix A will depend on the chosen model

First lets just plot the data

In [378…
```
plt.errorbar(x, y, yerr=y_err, fmt='o')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Data generated from a cubic model')
```

Out[378]:
```
Text(0.5, 1.0, 'Data generated from a cubic model')
```



In [379…
```
Y = np.atleast_2d(y).T
C = np.diag(y_err * y_err)
C_inv = np.linalg.inv(C)


#define the design matrix for each model.
A1 = np.vstack([np.ones_like(x)]).T
A2 = np.vstack([np.ones_like(x), x]).T
```

```
A3 = np.vstack([np.ones_like(x), x, x**2]).T
A4 = np.vstack([np.ones_like(x), x, x**2, x**3]).T
A5 = np.vstack([np.ones_like(x), x, x**2, x**3, x**4]).T
A6 = np.vstack([np.ones_like(x), x, x**2, x**3, x**4,x**5]).T

#combine
Atot = [A1,A2,A3,A4,A5,A6]

linalg_theta = []
for A in Atot:

    G = np.linalg.inv(A.T @ C_inv @ A)
    X = G @ (A.T @ C_inv @ Y)
    linalg_theta.append(X.T)
    print(X.T)

#we need to defin the log-liklihood for each model
```

```
[[21.67873104]]
[[-67.1179982    1.66024109]]
[[14.01680306 -1.0743762   0.02236534]]
[[-4.66104908e+00  2.46702201e-01 -4.57796601e-03  1.63422420e-04]]
[[-5.63444708e+00  3.66362413e-01 -8.75324378e-03  2.19927933e-04
  -2.60997016e-07]]
[[-2.01344523e+01  2.91357063e+00 -1.40030564e-01  3.12276501e-03
  -2.92565557e-05  1.07324902e-07]]
```

The Bayesian Information Critera is defined by

$$BIC = Dlog(N) - 2log\hat{\mathcal{L}}(\mathrm{y}|\hat{\theta})$$

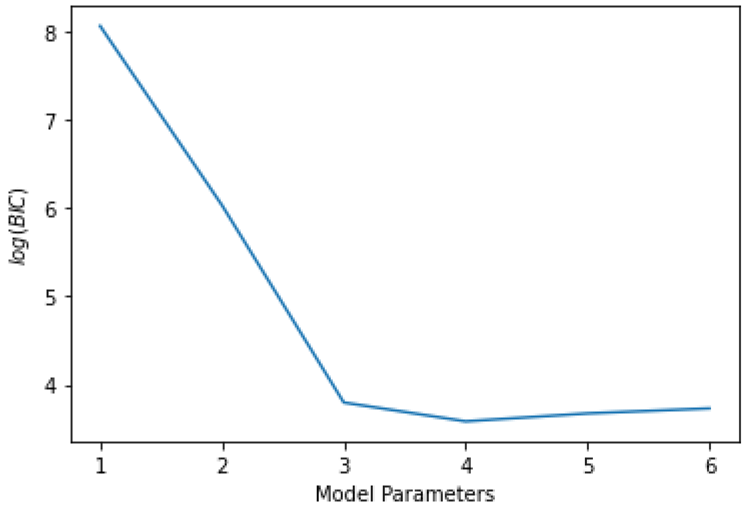
Where D is the number of model parameters, N is the is the number of data points. $\hat{\mathcal{L}}(\mathrm{y}|\hat{\theta})$ is the maximum log-likelihood of the model.

In [380…
```
#define the log-liklihoods of each model

def ln_likelihood(theta, x, y, y_err, i):
    if i == 0:
        b = theta
        return -0.5 * np.sum((y - b)**2 / y_err**2)
    elif i==1:
        b, a1 = theta
        return -0.5 * np.sum((y - a1 * x - b)**2 / y_err**2)
    elif i ==2:
        b, a1, a2 = theta
        return -0.5 * np.sum((y - a2*x**2 - a1 * x - b)**2 / y_err**2)
    elif i ==3:
        b, a1, a2, a3 = theta
        return -0.5 * np.sum((y - a3*x**3 - a2*x**2 - a1 * x - b)**2 / y_err**2)
    elif i == 4:
        b, a1, a2, a3, a4 = theta
        return -0.5 * np.sum((y - a4*x**4 - a3*x**3 - a2*x**2 - a1 * x - b)**2 / y_err**
    elif i ==5:
        b, a1, a2, a3, a4, a5 = theta
        return -0.5 * np.sum((y - a5*x**5 - a4*x**4 - a3*x**3 - a2*x**2 - a1 * x - b)**2
```

In [381…
```
# now create a list of BIC with each model

BIC = []

#list of the number of model parameters
D = np.arange(1,7,1)
#number of data points
N = len(x)
```

```
for i in np.arange(6):
    max_log_l = ln_likelihood(linalg_theta[i].T,x,y,y_err,i)
    BIC.append(D[i]*np.log(N) - 2*max_log_l)

print(BIC)

plt.plot(D,np.log(BIC))
plt.xlabel('Model Parameters')
plt.ylabel('$log(BIC)$')
plt.show()
```

```
[3150.7446245108035, 415.2095641283359, 44.616906558247884, 36.066508244237454, 39.45035
509776402, 41.74576218404091]
```



It is easy to increase the maximum log-likelihood of a model by increasing the number of paramters. The BIC is designed to take into account this issue when comparing models and lower BIC implies a better model!

In our case the model with 4 parameters is favoured which is consitent with the generated data However typically we would need lower BIC to be confident that such a model is indeed favoured.

Question 4

The data is drawn from a 2D gaussian centered around a single point with uncorrelated x and y uncertainties. Such that the log likelihood of the model is:

$$\log \mathcal{L} \propto -\sum_{i=1}^{N} \frac{[y_i - \hat{y}_i]^2}{2\sigma_{yi}^2} + \frac{[x_i - \hat{x}_i]^2}{2\sigma_{xi}^2} + log(\sigma_{yi}\sigma_{xi})$$

Such that

$$U = -\log \mathcal{L} \propto \sum_{i=1}^{N} \frac{[y_i - \hat{y}_i]^2}{2\sigma_{yi}^2} + \frac{[x_i - \hat{x}_i]^2}{2\sigma_{xi}^2} + log(\sigma_{yi}\sigma_{xi})$$

and

$$\frac{dU}{d\hat{x}_i} \propto -\sum_{i=1}^{N} \frac{x_i - \hat{x}_i}{\sigma_{xi}^2}$$

$$\frac{dU}{d\hat{y}_i} \propto -\sum_{i=1}^{N} \frac{y_i - \hat{y}_i}{\sigma_{yi}^2}$$

In [382…

```
#define the log-likelihood of the model
def ln_likelihood(theta, x, y, x_err, y_err):

    mu_x, mu_y = theta

    return -np.sum(np.log(y_err*x_err) + (x-mu_x)**2/(2*x_err**2) + (y-mu_y)**2/(2*y_err

#define the log-prior
#perhaps we dont need to define the prior
def ln_prior(theta):
    mu_x, mu_y = theta
    if not (1 > mu_x > 0)\
    or not (1 > mu_y > 0):
        return -np.inf
    return 1.0

def ln_probability(theta, x, y, x_err, y_err):
    return  ln_likelihood(theta, x, y, x_err, y_err)

def U(theta, x, y, x_err,y_err):
    return - ln_probability(theta, x, y, x_err,y_err)

def dU_dx(theta, x, y,  x_err, y_err):

    mu_x, mu_y = theta
    dU_dmux = -np.sum((x - mu_x)/x_err**2)
    dU_dmuy = -np.sum((y - mu_y)/y_err**2)

    return np.array([dU_dmux, dU_dmuy])
```

Question 5

In [384…

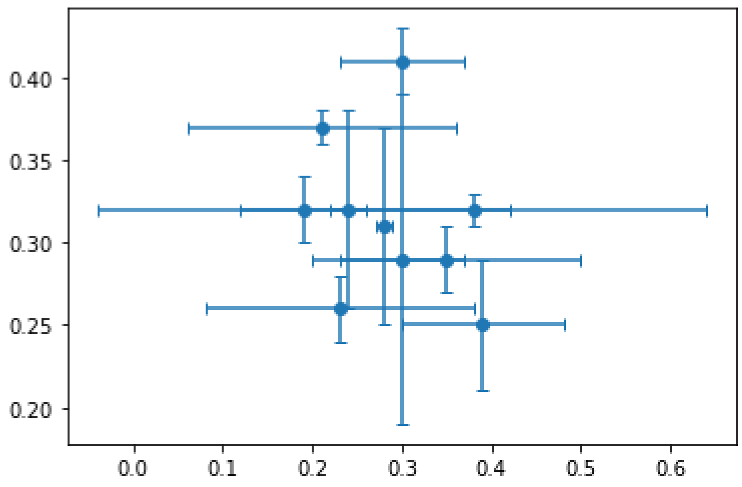
```
def leapfrog_integration(theta, p, dU_dx, n_steps, step_size):
    """
    Integrate a particle along an orbit using the Leapfrog integration scheme.
    """

    #append initial positions and enrgy
    total_energy = U(theta, x, y, x_err,y_err) + p.T@p/2
    energy.append(total_energy)#record positions and total energy
    positions.append(theta)

    theta = np.copy(theta)
    p = np.copy(p)
    # Take a half-step first.
    p -= 0.5 * step_size * dU_dx(theta, x, y,  x_err, y_err)
    for step in range(n_steps):
        theta += step_size * p
        p -= step_size * dU_dx(theta, x, y,  x_err, y_err)
            #append intermediate positions and enrgy
        total_energy = U(theta, x, y, x_err,y_err) + p.T@p/2

        energy.append(np.copy(total_energy))#record positions and total energy
        positions.append(np.copy(theta))


    theta += step_size * p
    p -= 0.5 * step_size * dU_dx(theta, x, y,  x_err, y_err)

      #append intermediate positions and enrgy
    total_energy = U(theta, x, y, x_err,y_err) + p.T@p/2
    energy.append(total_energy)#record positions and total energy
    positions.append(theta)

    return (theta, -p)
```

In [385…

```
#define data
x, y, x_err, y_err =  np.array(
    [[0.38, 0.32, 0.26, 0.01],
    [0.30, 0.41, 0.07, 0.02],
    [0.39, 0.25, 0.09, 0.04],
    [0.30, 0.29, 0.07, 0.10],
    [0.19, 0.32, 0.23, 0.02],
    [0.21, 0.37, 0.15, 0.01],
    [0.28, 0.31, 0.01, 0.06],
    [0.24, 0.32, 0.02, 0.06],
    [0.35, 0.29, 0.15, 0.02],
    [0.23, 0.26, 0.15, 0.02]]
        ).T

plt.errorbar(x,y,xerr=x_err,yerr=y_err,fmt='o', capsize=3)
```

Out[385]:

```
<ErrorbarContainer object of 3 artists>
```

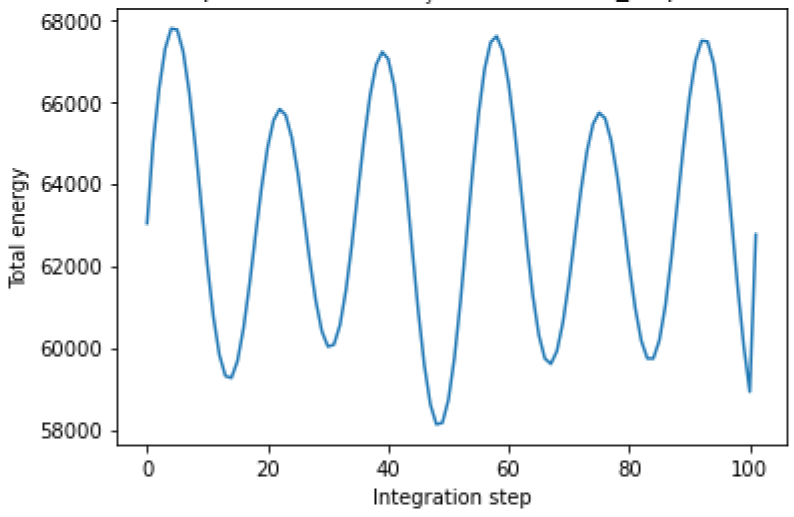


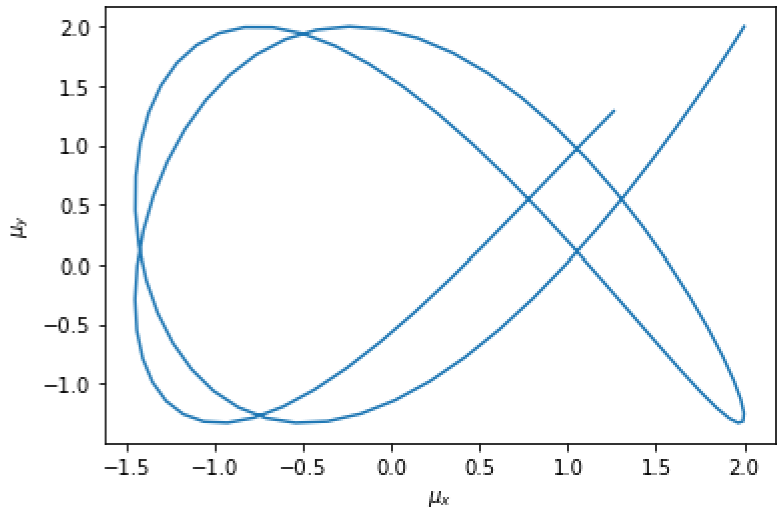
In [386…

```
initial_theta = [(2.0,2.0),(0.5,0.5),(0.1,0.1)]
N_list = [100,500,1000]

# np.random.seed(1)
p = np.random.normal(size = 2) #draw initial momentum

print(p)


for i in range(3):
    positions = []
    energy = []

    leapfrog_integration(initial_theta[i], p, dU_dx, N_list[i], 0.001)

    dom = range(len(energy))
    plt.title(f'Initial position $mu_x$ = $mu_y$ = {initial_theta[i]}, N_steps = {N_list
    plt.plot(dom,energy)
    plt.ylabel('Total energy')
    plt.xlabel('Integration step')
    plt.show()
    mux, muy = np.array(positions).T

    plt.plot(mux,muy)
    plt.xlabel('$\mu_x$')
    plt.ylabel('$\mu_y$')
    plt.show()
```

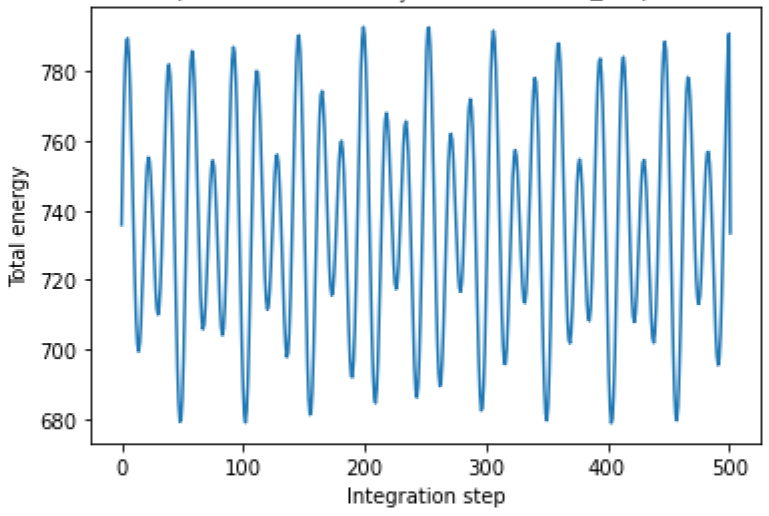
```
[ 1.17877957 -0.17992484]
```

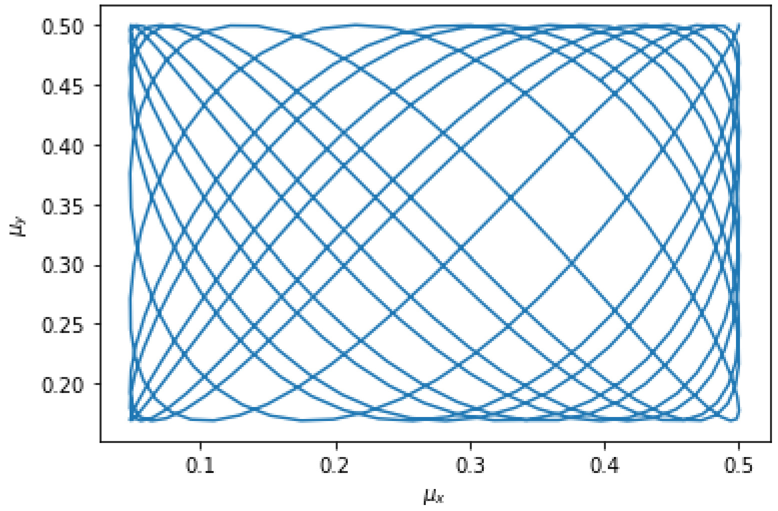
Initial position mu_x = mu_y = (2.0, 2.0), N_steps = 100

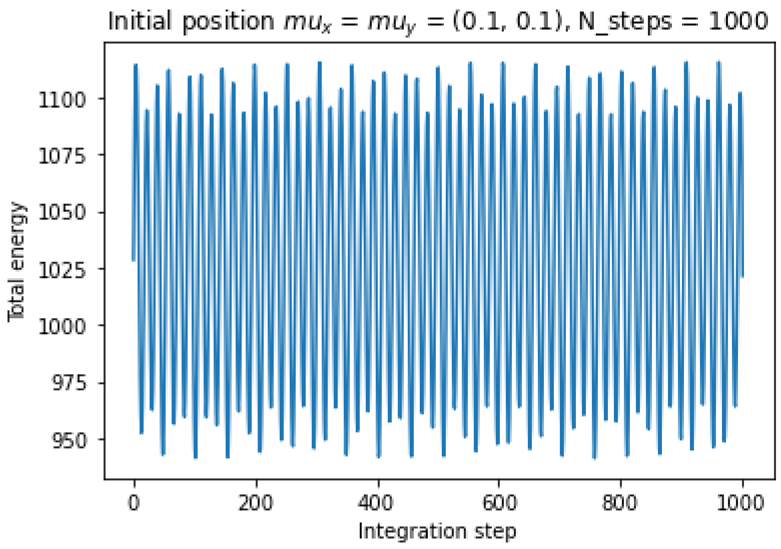


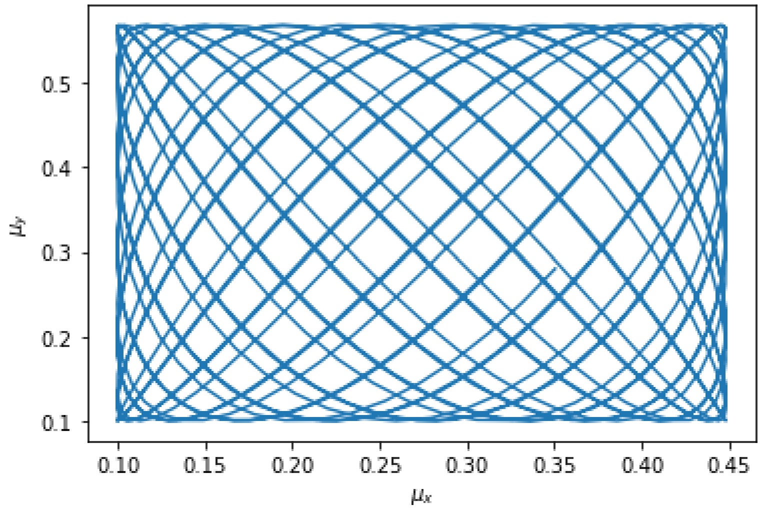


Initial position mu_x = mu_y = (0.5, 0.5), N_steps = 500









The Leap-frog integrator is a time-reversible and volume-preserving scheme. Hence the hamiltonian remains constant and total energy is therefore conserved. \ This is why leapfrog is used in conjunction with the Hamiltonian MC, since other schemes such as RK4 will not conserve energy and cause HMC to fail.

Question 6

In [387…

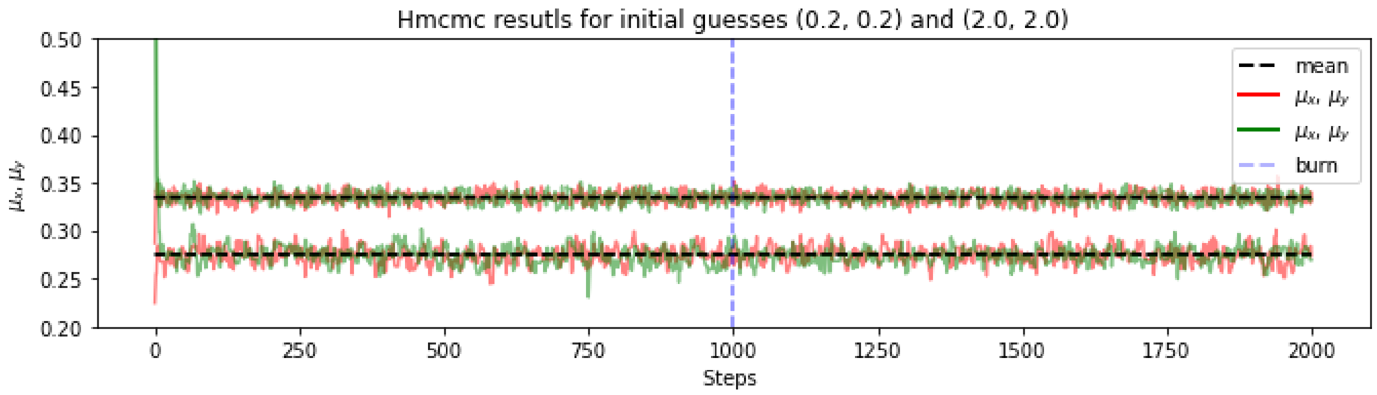
```
#defining the hamiltonian MCMC
def h_mcmc(theta, N):
    # step size
    dx = 0.01
    # no. of steps
    L = 10
    # initial guess
    theta0 = theta
    chain = []
    for i in range(N):
        print(f"Running step {i} of {N}", end='\r')
        # 1. draw from momentum distribution.
        p0 = np.random.normal(size = 2)
        # 2. integrate for L steps.
        theta1, p1 = leapfrog_integration(theta0, p0, dU_dx, L, dx)
        p1 = -p1

        alpha = np.exp(-U(theta1, x, y, x_err, y_err) + U(theta0, x, y, x_err, y_err) -
        u = np.random.uniform(0., 1.)
        if alpha >= u:
            # accept
            theta0 = theta1
        chain.append(theta0)
    return np.array(chain)
```

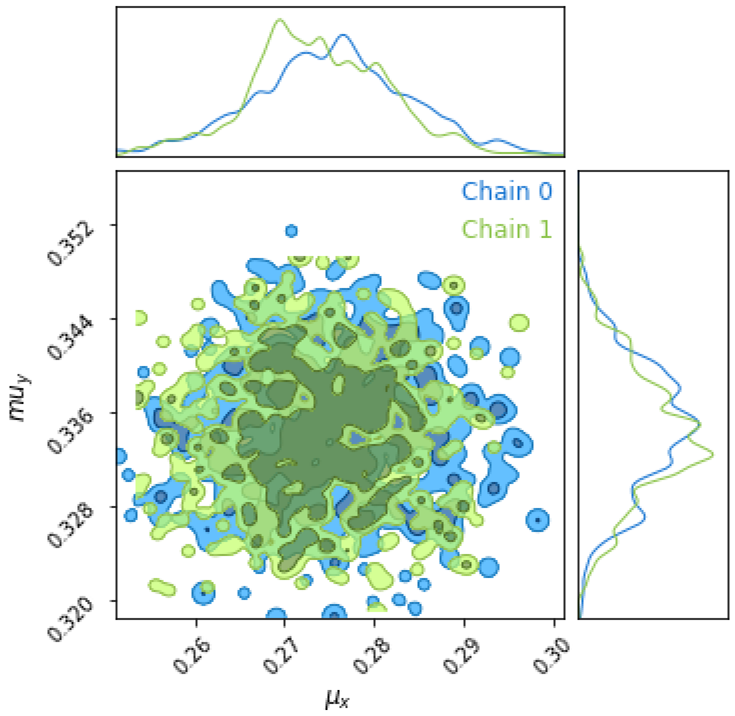
In [395…

```
initial_theta = [(0.2,0.2),(2.0,2.0)]
colour = ['r', 'g']

# chain_list = []
# for i in range(2):
#     chain = h_mcmc(initial_theta[i], 2000)
#     chain_list.append(chain)

fig, ax = plt.subplots(figsize=(10, 3))
ax.legend(loc = 'right')
ax.set_ylabel(r"$\mu_x, \mu_y$")
ax.set_xlabel("Steps ")
ax.set_ylim(0.2,0.5)
plt.title(f'Hmcmc resutls for initial guesses {initial_theta[0]} and {initial_theta[1]}'

for i in range(2):
    mux, muy = chain_list[i].T
    mean_mux = np.mean(mux)
    mean_muy = np.mean(muy)

    mean_x = np.full(len(mux),mean_mux)
    mean_y = np.full(len(muy),mean_muy)

    ax.plot(mux, c=colour[i], label="$\mu_y$ HMC chain", alpha = 0.5)
    ax.plot(muy, c=colour[i], label="$\mu_y$ HMC chain", alpha = 0.5)
    ax.plot(mean_x, c='k', label="Mean $\mu_y$",linestyle = '--')
    ax.plot(mean_y, c='k', label="Mean $\mu_y$", linestyle = '--')
    plt.axvline(x=1000, color='blue', linestyle='--', alpha = 0.3, label= 'burn')

legend_elements = [
    plt.Line2D([0], [0], color='k', lw=2, label='mean', linestyle = '--'),
    plt.Line2D([0], [0], color='r', lw=2, label='$\mu_x$, $\mu_y$'),
    plt.Line2D([0], [0], color='g', lw=2, label='$\mu_x$, $\mu_y$'),
    plt.Line2D([0], [0], color='blue', lw=2, label='burn',linestyle = '--', alpha = 0.3)
]

ax.legend(handles=legend_elements)
fig.tight_layout()

c = ChainConsumer()
for i in range(2):
    c.add_chain(chain_list[i][1000:], parameters=["$\mu_x$", "$mu_y$"])
fig = c.plotter.plot(filename="example.png", figsize="column")
```

```
#     # plot corner plot
#     fig = corner.corner(chain, labels=["$\mu_x$", "$\mu_y$"], show_titles=True, title_


#     for ax in fig.get_axes():
#         ax.tick_params(axis='both', size=3)
#         ax.set_title(ax.get_title())
#         ax.title.set_position([0.5, 0.95])
#     plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that arti
sts whose label start with an underscore are ignored when legend() is called with no arg
ument.
WARNING:chainconsumer:Parameter $\mu_x$ in chain Chain 0 is not constrained
WARNING:chainconsumer:Parameter $\mu_x$ in chain Chain 1 is not constrained
WARNING:chainconsumer:Parameter $mu_y$ in chain Chain 0 is not constrained
WARNING:chainconsumer:Parameter $mu_y$ in chain Chain 1 is not constrained
```





The convergence of the chains that had different initialisation points is a promising sign that the HMC shceme is working corectly. Although we see a general 'bumbyness' in the guassian curves this could be smoothened out with increasing more steps. The predicted values for the model parameters are also consistent to our expectations bye 'eye'. We could furthur improve this scheme if we were to tune the parameters for each step allowing for a more effective sampler.

Quesiton 7

In [ ]:
```
# ###############THIS IS MY .stan FILE ################

# data {
#     int<lower=1> N_data; // number of data points

#     // x-values of the data is uncertain.
#     vector[N_data] x;
#     vector[N_data] sigma_x;

#     // y-values of the data is uncertain.
#     vector[N_data] y;
#     vector[N_data] sigma_y;

# }

# parameters {
#     // Mean value of the x-guassian.
#     real<lower=-2.0, upper=2.0> mu_x;

#     // Mean value of the y-guassian.
#     real<lower=-2.0, upper=2.0> mu_y;

# }

# model {
#     for (i in 1:N_data) {
#         x[i] ~ normal(
#           mu_x,
#           sigma_x[i]
#         );
#       y[i] ~ normal(
#            mu_y,
#           sigma_y[i]
#         );
#     }
# }
```

In [329…
```
from cmdstanpy import CmdStanModel, install_cmdstan
install_cmdstan()
```

```
Installing CmdStan version: 2.32.1
Install directory: /home/lewis/.cmdstan
Downloading CmdStan version 2.32.1
Download successful, file: /tmp/tmpisgq_ekf
Extracting distribution
DEBUG:cmdstanpy:cmd: make build -j1
cwd: None
Unpacked download as cmdstan-2.32.1
Building version cmdstan-2.32.1, may take several minutes, depending on your system.
DEBUG:cmdstanpy:cmd: make examples/bernoulli/bernoulli
cwd: None
Test model compilation
Installed cmdstan-2.32.1
```

Out[329]:
```
True
```

In [396…
```
model = CmdStanModel(stan_file='PS2-1.stan')

# Data.
data = dict(
    N_data = 10,
    x=x,
    y=y,
    sigma_x = x_err,
    sigma_y = y_err,
)

fit1 = model.sample(data = data)
fit2 = model.sample(data = data)


#run samples
```

```
17:56:57 - cmdstanpy - INFO - compiling stan file /home/lewis/Documents/Honours/Machine
learning/notebooks/probelm_sets/PS2-1.stan to exe file /home/lewis/Documents/Honours/Mac
hine learning/notebooks/probelm_sets/PS2-1
INFO:cmdstanpy:compiling stan file /home/lewis/Documents/Honours/Machine learning/notebo
oks/probelm_sets/PS2-1.stan to exe file /home/lewis/Documents/Honours/Machine learning/n
otebooks/probelm_sets/PS2-1
DEBUG:cmdstanpy:cmd: make /tmp/tmpijtcybf8/tmpvbcqks4l
cwd: /home/lewis/.cmdstan/cmdstan-2.32.1
DEBUG:cmdstanpy:Console output:

--- Translating Stan model to C++ code ---
bin/stanc  --o=/tmp/tmpijtcybf8/tmpvbcqks4l.hpp /tmp/tmpijtcybf8/tmpvbcqks4l.stan

--- Compiling, linking C++ code ---
g++ -std=c++1y -pthread -D_REENTRANT -Wno-sign-compare -Wno-ignored-attributes      -I s
tan/lib/stan_math/lib/tbb_2020.3/include    -O3 -I src -I stan/src -I stan/lib/rapidjson
_1.1.0/ -I lib/CLI11-1.9.1/ -I stan/lib/stan_math/ -I stan/lib/stan_math/lib/eigen_3.4.0
-I stan/lib/stan_math/lib/boost_1.78.0 -I stan/lib/stan_math/lib/sundials_6.1.1/include
-I stan/lib/stan_math/lib/sundials_6.1.1/src/sundials    -DBOOST_DISABLE_ASSERTS
  -c -Wno-ignored-attributes   -x c++ -o /tmp/tmpijtcybf8/tmpvbcqks4l.o /tmp/tmpijtcybf
8/tmpvbcqks4l.hpp
g++ -std=c++1y -pthread -D_REENTRANT -Wno-sign-compare -Wno-ignored-attributes      -I s
tan/lib/stan_math/lib/tbb_2020.3/include    -O3 -I src -I stan/src -I stan/lib/rapidjson
_1.1.0/ -I lib/CLI11-1.9.1/ -I stan/lib/stan_math/ -I stan/lib/stan_math/lib/eigen_3.4.0
-I stan/lib/stan_math/lib/boost_1.78.0 -I stan/lib/stan_math/lib/sundials_6.1.1/include
-I stan/lib/stan_math/lib/sundials_6.1.1/src/sundials    -DBOOST_DISABLE_ASSERTS
       -Wl,-L,"/home/lewis/.cmdstan/cmdstan-2.32.1/stan/lib/stan_math/lib/tbb" -Wl,-rpat
h,"/home/lewis/.cmdstan/cmdstan-2.32.1/stan/lib/stan_math/lib/tbb"        /tmp/tmpijtcyb
f8/tmpvbcqks4l.o src/cmdstan/main.o       -Wl,-L,"/home/lewis/.cmdstan/cmdstan-2.32.1/st
an/lib/stan_math/lib/tbb" -Wl,-rpath,"/home/lewis/.cmdstan/cmdstan-2.32.1/stan/lib/stan_
math/lib/tbb"     stan/lib/stan_math/lib/sundials_6.1.1/lib/libsundials_nvecserial.a sta
n/lib/stan_math/lib/sundials_6.1.1/lib/libsundials_cvodes.a stan/lib/stan_math/lib/sundi
als_6.1.1/lib/libsundials_idas.a stan/lib/stan_math/lib/sundials_6.1.1/lib/libsundials_k
insol.a  stan/lib/stan_math/lib/tbb/libtbb.so.2 -o /tmp/tmpijtcybf8/tmpvbcqks4l
rm -f /tmp/tmpijtcybf8/tmpvbcqks4l.o

17:57:08 - cmdstanpy - INFO - compiled model executable: /home/lewis/Documents/Honours/M
achine learning/notebooks/probelm_sets/PS2-1
INFO:cmdstanpy:compiled model executable: /home/lewis/Documents/Honours/Machine learnin
g/notebooks/probelm_sets/PS2-1
DEBUG:cmdstanpy:input tempfile: /tmp/tmpyqnp6xyb/a6te2rrr.json
DEBUG:cmdstanpy:cmd: /home/lewis/Documents/Honours/Machine learning/notebooks/probelm_se
ts/PS2-1 info
cwd: None
17:57:08 - cmdstanpy - INFO - CmdStan start processing
INFO:cmdstanpy:CmdStan start processing
```

```
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status
```

```
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:idx 1
DEBUG:cmdstanpy:idx 2
DEBUG:cmdstanpy:idx 3
DEBUG:cmdstanpy:running CmdStan, num_threads: 1
DEBUG:cmdstanpy:running CmdStan, num_threads: 1
DEBUG:cmdstanpy:running CmdStan, num_threads: 1
```

```
DEBUG:cmdstanpy:running CmdStan, num_threads: 1
DEBUG:cmdstanpy:CmdStan args: ['/home/lewis/Documents/Honours/Machine learning/notebook
s/probelm_sets/PS2-1', 'id=1', 'random', 'seed=94233', 'data', 'file=/tmp/tmpyqnp6xyb/a6
te2rrr.json', 'output', 'file=/tmp/tmpyqnp6xyb/PS2-18t_kb9sf/PS2-1-20230511175709_1.cs
v', 'method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
DEBUG:cmdstanpy:CmdStan args: ['/home/lewis/Documents/Honours/Machine learning/notebook
s/probelm_sets/PS2-1', 'id=2', 'random', 'seed=94233', 'data', 'file=/tmp/tmpyqnp6xyb/a6
te2rrr.json', 'output', 'file=/tmp/tmpyqnp6xyb/PS2-18t_kb9sf/PS2-1-20230511175709_2.cs
v', 'method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
DEBUG:cmdstanpy:CmdStan args: ['/home/lewis/Documents/Honours/Machine learning/notebook
s/probelm_sets/PS2-1', 'id=3', 'random', 'seed=94233', 'data', 'file=/tmp/tmpyqnp6xyb/a6
te2rrr.json', 'output', 'file=/tmp/tmpyqnp6xyb/PS2-18t_kb9sf/PS2-1-20230511175709_3.cs
v', 'method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
DEBUG:cmdstanpy:CmdStan args: ['/home/lewis/Documents/Honours/Machine learning/notebook
s/probelm_sets/PS2-1', 'id=4', 'random', 'seed=94233', 'data', 'file=/tmp/tmpyqnp6xyb/a6
te2rrr.json', 'output', 'file=/tmp/tmpyqnp6xyb/PS2-18t_kb9sf/PS2-1-20230511175709_4.cs
v', 'method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
```

```
17:57:09 - cmdstanpy - INFO - CmdStan done processing.
INFO:cmdstanpy:CmdStan done processing.
DEBUG:cmdstanpy:runset
RunSet: chains=4, chain_ids=[1, 2, 3, 4], num_processes=4
 cmd (chain 1):
        ['/home/lewis/Documents/Honours/Machine learning/notebooks/probelm_sets/PS2-1',
'id=1', 'random', 'seed=94233', 'data', 'file=/tmp/tmpyqnp6xyb/a6te2rrr.json', 'output',
'file=/tmp/tmpyqnp6xyb/PS2-18t_kb9sf/PS2-1-20230511175709_1.csv', 'method=sample', 'algo
rithm=hmc', 'adapt', 'engaged=1']
 retcodes=[0, 0, 0, 0]
 per-chain output files (showing chain 1 only):
 csv_file:
        /tmp/tmpyqnp6xyb/PS2-18t_kb9sf/PS2-1-20230511175709_1.csv
 console_msgs (if any):
        /tmp/tmpyqnp6xyb/PS2-18t_kb9sf/PS2-1-20230511175709_0-stdout.txt
DEBUG:cmdstanpy:Chain 1 console:
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    save_warmup = 0 (Default)
    thin = 1 (Default)
    adapt
      engaged = 1 (Default)
      gamma = 0.050000000000000003 (Default)
      delta = 0.80000000000000004 (Default)
      kappa = 0.75 (Default)
      t0 = 10 (Default)
      init_buffer = 75 (Default)
      term_buffer = 50 (Default)
      window = 25 (Default)
    algorithm = hmc (Default)
      hmc
        engine = nuts (Default)
          nuts
            max_depth = 10 (Default)
        metric = diag_e (Default)
        metric_file =  (Default)
        stepsize = 1 (Default)
        stepsize_jitter = 0 (Default)
    num_chains = 1 (Default)
id = 1 (Default)
data
```

```
  file = /tmp/tmpyqnp6xyb/a6te2rrr.json
init = 2 (Default)
random
  seed = 94233
output
  file = /tmp/tmpyqnp6xyb/PS2-18t_kb9sf/PS2-1-20230511175709_1.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)
  sig_figs = -1 (Default)
  profile_file = profile.csv (Default)
num_threads = 1 (Default)


Gradient evaluation took 3.9e-05 seconds
1000 transitions using 10 leapfrog steps per transition would take 0.39 seconds.
Adjust your expectations accordingly!


Iteration:    1 / 2000 [  0%]  (Warmup)
Iteration:  100 / 2000 [  5%]  (Warmup)
Iteration:  200 / 2000 [ 10%]  (Warmup)
Iteration:  300 / 2000 [ 15%]  (Warmup)
Iteration:  400 / 2000 [ 20%]  (Warmup)
Iteration:  500 / 2000 [ 25%]  (Warmup)
Iteration:  600 / 2000 [ 30%]  (Warmup)
Iteration:  700 / 2000 [ 35%]  (Warmup)
Iteration:  800 / 2000 [ 40%]  (Warmup)
Iteration:  900 / 2000 [ 45%]  (Warmup)
Iteration: 1000 / 2000 [ 50%]  (Warmup)
Iteration: 1001 / 2000 [ 50%]  (Sampling)
Iteration: 1100 / 2000 [ 55%]  (Sampling)
Iteration: 1200 / 2000 [ 60%]  (Sampling)
Iteration: 1300 / 2000 [ 65%]  (Sampling)
Iteration: 1400 / 2000 [ 70%]  (Sampling)
Iteration: 1500 / 2000 [ 75%]  (Sampling)
Iteration: 1600 / 2000 [ 80%]  (Sampling)
Iteration: 1700 / 2000 [ 85%]  (Sampling)
Iteration: 1800 / 2000 [ 90%]  (Sampling)
Iteration: 1900 / 2000 [ 95%]  (Sampling)
Iteration: 2000 / 2000 [100%]  (Sampling)

 Elapsed Time: 0.017 seconds (Warm-up)
               0.043 seconds (Sampling)
               0.06 seconds (Total)


DEBUG:cmdstanpy:input tempfile: /tmp/tmpyqnp6xyb/6wzbmb65.json
DEBUG:cmdstanpy:cmd: /home/lewis/Documents/Honours/Machine learning/notebooks/probelm_se
ts/PS2-1 info
cwd: None
17:57:09 - cmdstanpy - INFO - CmdStan start processing
INFO:cmdstanpy:CmdStan start processing
```

```
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status
```

```
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:idx 1
DEBUG:cmdstanpy:idx 2
DEBUG:cmdstanpy:idx 3
DEBUG:cmdstanpy:running CmdStan, num_threads: 1
DEBUG:cmdstanpy:running CmdStan, num_threads: 1
DEBUG:cmdstanpy:running CmdStan, num_threads: 1
DEBUG:cmdstanpy:running CmdStan, num_threads: 1
```

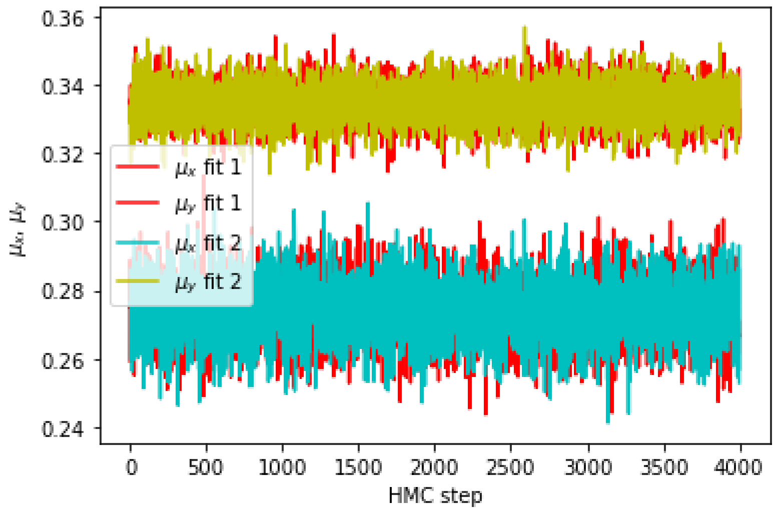
```
DEBUG:cmdstanpy:CmdStan args: ['/home/lewis/Documents/Honours/Machine learning/notebook
s/probelm_sets/PS2-1', 'id=1', 'random', 'seed=20855', 'data', 'file=/tmp/tmpyqnp6xyb/6w
zbmb65.json', 'output', 'file=/tmp/tmpyqnp6xyb/PS2-1vm1wrn63/PS2-1-20230511175709_1.cs
v', 'method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
DEBUG:cmdstanpy:CmdStan args: ['/home/lewis/Documents/Honours/Machine learning/notebook
s/probelm_sets/PS2-1', 'id=2', 'random', 'seed=20855', 'data', 'file=/tmp/tmpyqnp6xyb/6w
zbmb65.json', 'output', 'file=/tmp/tmpyqnp6xyb/PS2-1vm1wrn63/PS2-1-20230511175709_2.cs
v', 'method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
DEBUG:cmdstanpy:CmdStan args: ['/home/lewis/Documents/Honours/Machine learning/notebook
s/probelm_sets/PS2-1', 'id=3', 'random', 'seed=20855', 'data', 'file=/tmp/tmpyqnp6xyb/6w
zbmb65.json', 'output', 'file=/tmp/tmpyqnp6xyb/PS2-1vm1wrn63/PS2-1-20230511175709_3.cs
v', 'method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
DEBUG:cmdstanpy:CmdStan args: ['/home/lewis/Documents/Honours/Machine learning/notebook
s/probelm_sets/PS2-1', 'id=4', 'random', 'seed=20855', 'data', 'file=/tmp/tmpyqnp6xyb/6w
zbmb65.json', 'output', 'file=/tmp/tmpyqnp6xyb/PS2-1vm1wrn63/PS2-1-20230511175709_4.cs
v', 'method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
```

```
17:57:09 - cmdstanpy - INFO - CmdStan done processing.
INFO:cmdstanpy:CmdStan done processing.
DEBUG:cmdstanpy:runset
RunSet: chains=4, chain_ids=[1, 2, 3, 4], num_processes=4
 cmd (chain 1):
        ['/home/lewis/Documents/Honours/Machine learning/notebooks/probelm_sets/PS2-1',
'id=1', 'random', 'seed=20855', 'data', 'file=/tmp/tmpyqnp6xyb/6wzbmb65.json', 'output',
'file=/tmp/tmpyqnp6xyb/PS2-1vm1wrn63/PS2-1-20230511175709_1.csv', 'method=sample', 'algo
rithm=hmc', 'adapt', 'engaged=1']
 retcodes=[0, 0, 0, 0]
 per-chain output files (showing chain 1 only):
 csv_file:
        /tmp/tmpyqnp6xyb/PS2-1vm1wrn63/PS2-1-20230511175709_1.csv
 console_msgs (if any):
        /tmp/tmpyqnp6xyb/PS2-1vm1wrn63/PS2-1-20230511175709_0-stdout.txt
DEBUG:cmdstanpy:Chain 1 console:
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    save_warmup = 0 (Default)
    thin = 1 (Default)
    adapt
      engaged = 1 (Default)
      gamma = 0.050000000000000003 (Default)
      delta = 0.80000000000000004 (Default)
      kappa = 0.75 (Default)
      t0 = 10 (Default)
      init_buffer = 75 (Default)
      term_buffer = 50 (Default)
      window = 25 (Default)
    algorithm = hmc (Default)
      hmc
        engine = nuts (Default)
          nuts
            max_depth = 10 (Default)
        metric = diag_e (Default)
        metric_file =  (Default)
        stepsize = 1 (Default)
        stepsize_jitter = 0 (Default)
    num_chains = 1 (Default)
id = 1 (Default)
data
  file = /tmp/tmpyqnp6xyb/6wzbmb65.json
```

```
init = 2 (Default)
random
  seed = 20855
output
  file = /tmp/tmpyqnp6xyb/PS2-1vm1wrn63/PS2-1-20230511175709_1.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)
  sig_figs = -1 (Default)
  profile_file = profile.csv (Default)
num_threads = 1 (Default)


Gradient evaluation took 8e-06 seconds
1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
Adjust your expectations accordingly!


Iteration:    1 / 2000 [  0%]  (Warmup)
Iteration:  100 / 2000 [  5%]  (Warmup)
Iteration:  200 / 2000 [ 10%]  (Warmup)
Iteration:  300 / 2000 [ 15%]  (Warmup)
Iteration:  400 / 2000 [ 20%]  (Warmup)
Iteration:  500 / 2000 [ 25%]  (Warmup)
Iteration:  600 / 2000 [ 30%]  (Warmup)
Iteration:  700 / 2000 [ 35%]  (Warmup)
Iteration:  800 / 2000 [ 40%]  (Warmup)
Iteration:  900 / 2000 [ 45%]  (Warmup)
Iteration: 1000 / 2000 [ 50%]  (Warmup)
Iteration: 1001 / 2000 [ 50%]  (Sampling)
Iteration: 1100 / 2000 [ 55%]  (Sampling)
Iteration: 1200 / 2000 [ 60%]  (Sampling)
Iteration: 1300 / 2000 [ 65%]  (Sampling)
Iteration: 1400 / 2000 [ 70%]  (Sampling)
Iteration: 1500 / 2000 [ 75%]  (Sampling)
Iteration: 1600 / 2000 [ 80%]  (Sampling)
Iteration: 1700 / 2000 [ 85%]  (Sampling)
Iteration: 1800 / 2000 [ 90%]  (Sampling)
Iteration: 1900 / 2000 [ 95%]  (Sampling)
Iteration: 2000 / 2000 [100%]  (Sampling)

 Elapsed Time: 0.016 seconds (Warm-up)
               0.038 seconds (Sampling)
               0.054 seconds (Total)
```

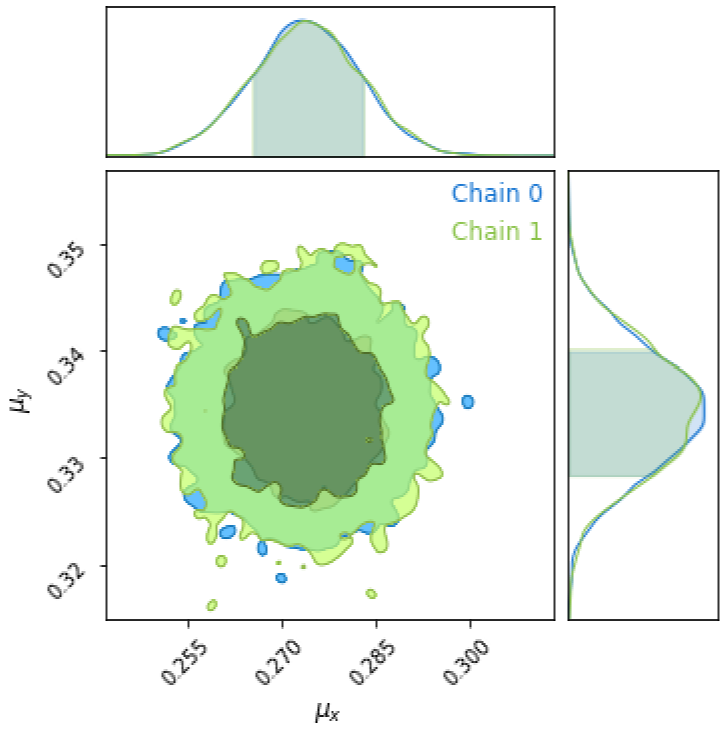
In [397…

```
plt.xlabel('HMC step')
plt.ylabel('$\mu_x$, $\mu_y$')
plt.plot(fit1.mu_x, c = 'r', label = '$\mu_x$ fit 1')
plt.plot(fit1.mu_y, c = 'r',label = '$\mu_y$ fit 1')
plt.plot(fit2.mu_x, c = 'c',label = '$\mu_x$ fit 2')
plt.plot(fit2.mu_y, c = 'y',label = '$\mu_y$ fit 2')
plt.legend()
plt.show()
```



In [398…

```
chain1 = [fit1.mu_x,fit1.mu_y]
chain2 = [fit2.mu_x, fit2.mu_y]
c = ChainConsumer()
c.add_chain(chain1, parameters=["$\mu_x$","$\mu_y$"])
c.add_chain(chain2, parameters=["$\mu_x$","$\mu_y$"])
fig = c.plotter.plot(filename="example.png", figsize="column")
```



As one might of expected the results look alot smoother than my scheme. This is because Stan tunes paramters in each step such as the number of steps and step size for the intergration. This allows for a more effective and faster sampler.

In [ ]: