Question 1:

Let us assume a simple model of the form:

$$y_i \sim N(mx_i + b,\ \sigma_{y_i})$$

Cast this problem into matrix form and show how the frequency distribution is related to the χ^2 statistic

Solution.

The frequency distribution for the uncertainty in the data points (x_i, y_i) of the (linear) model denoted by $p(y_i|x_i, \sigma_y, m, b)$ is given by:

$$p(y_i|x_i,\sigma_y,m,b) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y_i - mx_i - b)^2}{2\sigma_y^2}\right) \quad (1)$$

One can cast the above problem into matrix form by first defining the following matrices:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad A = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & \vdots \\ 1 & x_n \end{bmatrix}, \quad C = \begin{bmatrix} \sigma_{y_1}^2 & 0 & \cdots & 0 \\ 0 & \sigma_{y_2}^2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_{y_n}^2 \end{bmatrix}$$

$$X = \begin{bmatrix} m \\ b \end{bmatrix}$$

where Y consists of all of the measured values of the dependent variable (y_i), A is the design matrix, C is the covariance matrix, consisting of the values of the uncertainties for each y_i, and X is a column vector representing the model parameters (m and b), which correspond to the slope and the y-axis intercept of the linear model. Hence, Equation (1) in matrix form becomes:

$$p(y_i|x_i,\sigma_y,m,b) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{1}{2}(Y - AX)^T C^{-1} (Y - AX)\right)$$

If one assumes that the data points are drawn independently of one another, one can attempt to optimize the parameters that maximize the total probability given the model by defining the total likelihood $\mathcal{L}$, which represents the product of all conditional probabilities of each data point. Hence:

$$\mathcal{L} = \prod_{i=1}^{n} p(y_i|x_i,\sigma_y,m,b)$$

Computing the total likelihood may involve taking the product of small terms. Hence, it is more practical to take the natural logarithm of both sides of the above definition and convert the product into a sum as follows:

$$\begin{aligned}
\log(\mathcal{L}) &= \log\left(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{1}{2}(Y - AX)^T C^{-1} (Y - AX)\right)\right) \\
&= \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma_y^2}}\right) + \sum_{i=1}^{n} \log\left(\exp\left[-\frac{1}{2}(Y - AX)^T C^{-1} (Y - AX)\right]\right) \\
&= -\frac{1}{2}\sum_{i=1}^{n} \log\left(2\pi\sigma_y^2\right) + \sum_{i=1}^{n} -\frac{1}{2}(Y - AX)^T C^{-1} (Y - AX) \\
&= -\sum_{i=1}^{n} \log\left(2\pi\sigma_y\right) - \frac{1}{2}\sum_{i=1}^{n} (Y - AX)^T C^{-1} (Y - AX) \\
&= -\frac{n}{2}\log(2\pi) - \sum_{i=1}^{n} \log(\sigma_y) - \frac{1}{2}\sum_{i=1}^{n} (Y - AX)^T C^{-1} (Y - AX)
\end{aligned}$$

Since the terms $\frac{n}{2}\log(2\pi)$ and $\sum_{i=1}^{n}\log(\sigma_y)$ are both constant, the above expression can be written as:

$$\log(\mathcal{L}) \sim k\sum_{i=1}^{n} (Y - AX)^T C^{-1} (Y - AX) + K$$

where $K = -\frac{n}{2}\log(2\pi) - \sum_{i=1}^{n}\log(\sigma_y)$ is the constant term. Hence:

$$\begin{aligned}
\log(\mathcal{L}) &\sim -\frac{1}{2}\sum_{i=1}^{n} (Y - AX)^T C^{-1} (Y - AX) + K \\
&\sim \frac{-1}{2}\chi^2 + K
\end{aligned}$$

Hence, it is evident that

$$\chi^2 = \sum_{i=1}^{n} (Y - AX)^T C^{-1} (Y - AX) = \sum_{i=1}^{n} \frac{(y_i - mx_i - b)^2}{\sigma_y^2}$$ } result provided in the lecture notes

Question 2:

Write the definition of χ^2 in matrix form and take the first derivative with respect to X, the vector of unknown parameters, and show that the solution to $Y = AX$, with a covariance matrix C and numerous implicit assumptions is:

$$X = (A\,C^{-1}A)^T\,(A\,C^{-1}Y)$$

Solution.

In matrix form, χ^2 can be written as:

$$\chi^2 = (Y - AX)^T\ C^{-1}\ (Y - AX)$$

where the matrices Y, A, X and C^{-1} are defined as given in Question 1

One can expand the above expression as follows:

$$\begin{aligned}
\chi^2 &= (Y - AX)^T C^{-1} (Y - AX) \\
&= (Y^T - (AX)^T) C^{-1} (Y - AX) \\
&= (Y^T - X^T A^T) C^{-1} (Y - AX) \\
&= (Y^T C^{-1} - X^T A^T C^{-1})(Y - AX) \\
&= Y^T C^{-1} Y - Y^T C^{-1} AX - X^T A^T C^{-1} Y + X^T A^T C^{-1} AX
\end{aligned}$$

Since we wish to find the best fit values for X, this corresponds to the minimization of χ^2 $\Rightarrow \frac{\partial \chi^2}{\partial X} = 0$

$$\begin{aligned}
\frac{\partial \chi^2}{\partial x} &= \frac{\partial}{\partial x}(Y^T C^{-1} Y) - \frac{\partial}{\partial x}(Y^T C^{-1} AX) - \frac{\partial}{\partial x}(X^T A^T C^{-1} Y) + \frac{\partial}{\partial x}(X^T A^T C^{-1} AX) = 0 \\
&\Rightarrow 0^T C^{-1} A - \frac{\partial}{\partial x}(X^T A^T C^{-1} Y) + \frac{\partial}{\partial x}(X^T A^T C^{-1} AX) = 0 \\
&= -Y^T C^{-1} A - \frac{\partial}{\partial x}(X^T A^T C^{-1} Y) + X^T\left((A^T C^{-1} A)^T + (A^T C^{-1} A)\right) = 0
\end{aligned}$$

where the identity $\frac{\partial}{\partial x}(X^T A x) = X^T(A + A^T)$ has been used in the last term (as $A^T C^{-1} A$ is independent of the model parameters m and b)

$$\begin{aligned}
&\Rightarrow -Y^T C^{-1} A - \frac{\partial}{\partial x}(X^T A^T C^{-1} Y) + X^T\left((C^{-1})^T A^T A\right) + (A^T C^{-1} A)) = 0 \\
&\Rightarrow -Y^T C^{-1} A - \frac{\partial}{\partial x}(X^T A^T C^{-1} Y) + X^T\left(A^T C^{-1} A + A^T C^{-1} A\right) = 0 \\
&\Rightarrow -Y^T C^{-1} A - \frac{\partial}{\partial x}(X^T A^T C^{-1} Y) + A^T C^{-1} A X + A^T C^{-1} A X = 0 \\
&\Rightarrow -Y^T C^{-1} A - \frac{\partial}{\partial x}(X^T A^T C^{-1} Y) + 2A^T C^{-1} A X = 0 \\
&\Rightarrow -Y^T C^{-1} A - \frac{\partial}{\partial x}(Y^T C^{-1} A X) + 2A^T C^{-1} A X = 0
\end{aligned}$$

where the property $\frac{\partial}{\partial x}(X^T B) = \frac{\partial}{\partial x}(B^T X)$ has been used in the second term

$$\begin{aligned}
&\therefore -Y^T C^{-1} A - Y^T C^{-1} A + 2A^T C^{-1} A X = 0 \\
&\therefore 2\,Y^T C^{-1} A = 2A^T C^{-1} A X \\
&\therefore Y^T C^{-1} A = A^T C^{-1} A X \\
&\therefore X = \left[A^T C^{-1} A\right]^{-1}\left[Y^T C^{-1} A\right] \text{ - best fit parameters for the model} \Rightarrow \text{QED}
\end{aligned}$$

Question 3:

Show that the matrix to project cartesian scatter λ perpendicular to the line $y = mx + b$ is:

$$\Lambda = \frac{\lambda^2}{1 + m^2}\begin{bmatrix} m^2 & -m \\ -m & 1 \end{bmatrix}$$

Solution.

The projection of any vector u onto another vector v is given by

$$\begin{aligned}
\text{Proj}_v(u) &= \frac{u \cdot v}{v \cdot v} v \\
&= \frac{v\,u}{\|v\|^2} v
\end{aligned}$$

The above expression can be rewritten in terms of matrix products as follows:

$$\begin{aligned}
\text{Proj}_v(u) &= \frac{v^T u}{v^T v} v = \frac{1}{v^T v}\, v\,(v^T u) \\
&= \frac{1}{v^T v}(v\,v^T)\,u \\
&= \frac{v\,v^T}{v^T v} u = P\,u \quad \text{where } P \text{ denotes the projection matrix that draws forms the vector } u
\end{aligned}$$

One can let u be the vector that points in the direction of the line perpendicular to the line $y = mx + b$. The equation of this line is given by

$$y = \frac{-1}{m}x + b \quad , \ b \in \mathbb{R}$$

Note that the value of b is unimportant as one is only interested in the direction of the unit vector that spans the line, which is specified by the gradient. Hence a possible u could be:

$$u = \begin{bmatrix} 1 \\ -1/m \end{bmatrix}$$

Hence, the projection matrix of the cartesian scatter, P can be determined by applying the above relationship

$$P = \frac{u\,u^T}{u^T u} \begin{matrix} \leftarrow \text{outer product} \\ \leftarrow \text{inner product} \\ \leftarrow 1 + \frac{1}{m^2} \end{matrix} = \left(1 + \frac{1}{m^2}\right)^{-1} \begin{bmatrix} 1 \\ -1/m \end{bmatrix} \begin{bmatrix} 1 & -1/m \end{bmatrix} = \begin{bmatrix} 1 & -1/m \\ -1/m & 1/m^2 \end{bmatrix} \cdot \frac{m^2}{m^2+1}$$

$$= \frac{1}{m^2+1}\begin{bmatrix} m^2 & -m \\ -m & 1 \end{bmatrix}$$

One must also include the square of the intrinsic scatter parameter, λ^2 as a projector. Hence the projection matrix for intrinsic scatter perpendicular to the line perpendicular to y_i mark is given by:

$$\Lambda = \frac{\lambda^2}{m^2+1}\begin{bmatrix} m^2 & -m \\ -m & 1 \end{bmatrix} \Rightarrow \text{QED}$$

Question 4:

The model parameters of this model are:

- m (slope of the linear model)
- b (y-axis intercept of the linear model)
- $\log(\lambda)$ (logarithm of the intrinsic scatter parameter of the linear model)

One can assume a flat prior on m and $\log(\lambda)$ of the form:

$$p(m,\log\lambda) \propto (1+m^2)^{-3/2}$$

Additionally, one can choose a uniform prior on λ in log space (as this will be invariant under a change of coordinates)

$$p(\ln(\lambda)) \sim U(-10, 10)$$

One can also introduce a parameter $\hat{x}_i$, which represents the true value of x for each data point. One can adopt an improper uniform prior on this parameter of the form:

$$p(y_i, x_i | m, b, C_i, \ln(\lambda)) = \int_{\infty}^{\infty} \frac{1}{\sqrt{(2\pi)^2 \det(C_i)}} \exp\left(-\frac{1}{2}\,\delta_i^T C_i^{-1} \delta_i\right) d\hat{x}_i$$

$$\delta_i = \text{residual vector} = \begin{bmatrix} x_i - \hat{x}_i \\ y_i - mx_i - b \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} \sigma_{x_i}^2 & \lambda_{xy} \sigma_{x_i} \sigma_{y_i} \\ \lambda_{xy}\sigma_{x_i}\sigma_{y_i} & \sigma_{y_i}^2 \end{bmatrix}$$

Hence $\hat{x}_i$ can be marginalised out (i.e integrated out). The log prior function for this model is of the form

$$-\frac{3}{2}\log(1 + m^2)$$

while the log likelihood function is given by

$$\sum_{i=1}^{n} -\log(\Sigma_i^2) - \frac{\Delta_i^2}{2\Sigma_i^2}$$

where $\Sigma_i^2 = V^T C_i V$, $\Delta_i = y_i - mx_i - b$ and V is the projection vector.

$$V = \begin{bmatrix} -m \\ 1 \end{bmatrix}$$

Question 5:

Since this model is required to account for outliers in addition to intrinsic scatter and correlations between the uncertainties in individual data points, one can employ a mixture model that employs the following model parameters and associated priors:

(1) m and b, for which there is a prior of $p(m,b) \propto (1+m^2)^{-3/2}$
(2) λ, which has a uniform log prior of the form $p(\ln(\lambda)) \sim U(-10, 10)$
(3) Q, which relates to whether a given data point is an outlier or not. This is assigned a prior value of Q for the straight line model (foreground model) and $1-Q$ for the outlier model
(4) σ_o, which represents the scatter of the outlier model, which has a uniform prior and has been chosen to assume values between 0 and 200
(5) Y_o, which represents the variance of the outlier model, which has a uniform prior in log space between -5 and 5 (this is related to the standard deviation of the outlier model

One must introduce a parameter q_i, which indicates whether each data point is an outlier or not. This can then be marginalised (integrated over) in the same way that $\hat{x}_i$ was in Question (4). The log prior function is given by:

$$p(\theta) \sim -\frac{3}{2}\log(1 + m^2)$$

The log likelihood function is given by:

$$\prod_{i=1}^{n}\left[\frac{Q}{\sqrt{2\pi\Sigma_i^2}} \exp\left(\frac{-(y_i - mx_i - b)^2}{2\Sigma_i^2}\right) + \frac{1-Q}{\sqrt{2\pi(\Sigma_i^2 + \sigma_o^2)}} \exp\left(-\frac{(y_i - mx)^2}{2(\Sigma_i^2 + \sigma_o^2)}\right)\right]$$

$$\therefore \mathcal{L} \propto \left(-\frac{1}{2}\sum_{i=1}^{n}\log(2\pi\Sigma_i^2) - \frac{1}{2}\sum_{i=1}^{n}\frac{(y_i - mx_i - b)^2}{\Sigma_i^2} + \ln\left(\frac{1-Q}{\sqrt{2\pi\sigma_o^2}} \exp\left(\frac{(y_i - m_o)^2}{2(\Sigma_i^2 + V_o)}\right)\right)\right)$$

where $\Sigma_i^2 = V^T C_i V$ and V = projection vector $[-m, 1]$

these account for intrinsic scatter

The log posterior probability is given by the sum of the log likelihood function and the log prior functions

Question 4

Use the example data given above. Assume that the y values are generated by a straight line model where there is some intrinsic scatter to the line, and there are uncertainties in the x and y -direction, but there is no correlation between the x-and y-uncertainties.

You need to fully specify this model. That includes specifying the model parameters, the priors on those parameters, any marginalisations, the log likelihood function, and the log prior function.

In [6]:

```
#Loading the dataset and useful libraries and packages
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
from matplotlib.patches import Ellipse
import scipy.optimize as op
from corner import corner
import emcee
from matplotlib.patches import Ellipse
# For reproducibility
np.random.seed(0)

# Load the data.
_, x, y, y_err, x_err, rho_xy = data = np.array([
    [1,  201, 592, 61,  9, -0.84],
    [2,  244, 401, 25,  4, +0.31],
    [3,   47, 583, 38, 11, +0.64],
    [4,  287, 402, 15,  7, -0.27],
    [5,  203, 495, 21,  5, -0.33],
    [6,   58, 173, 15,  9, +0.67],
    [7,  210, 479, 27,  4, -0.02],
    [8,  202, 504, 14,  4, -0.05],
    [9,  198, 510, 30, 11, -0.84],
    [10, 158, 416, 16,  7, -0.69],
    [11, 165, 393, 14,  5, +0.30],
    [12, 201, 442, 25,  5, -0.46],
    [13, 157, 317, 52,  5, -0.03],
    [14, 131, 311, 16,  6, +0.50],
    [15, 166, 400, 34,  6, +0.73],
    [16, 160, 337, 31,  5, -0.52],
    [17, 186, 423, 42,  9, +0.90],
    [18, 125, 334, 26,  8, +0.40],
    [19, 218, 533, 16,  6, -0.78],
    [20, 146, 344, 22,  5, -0.56],
]).T
```

Question 5

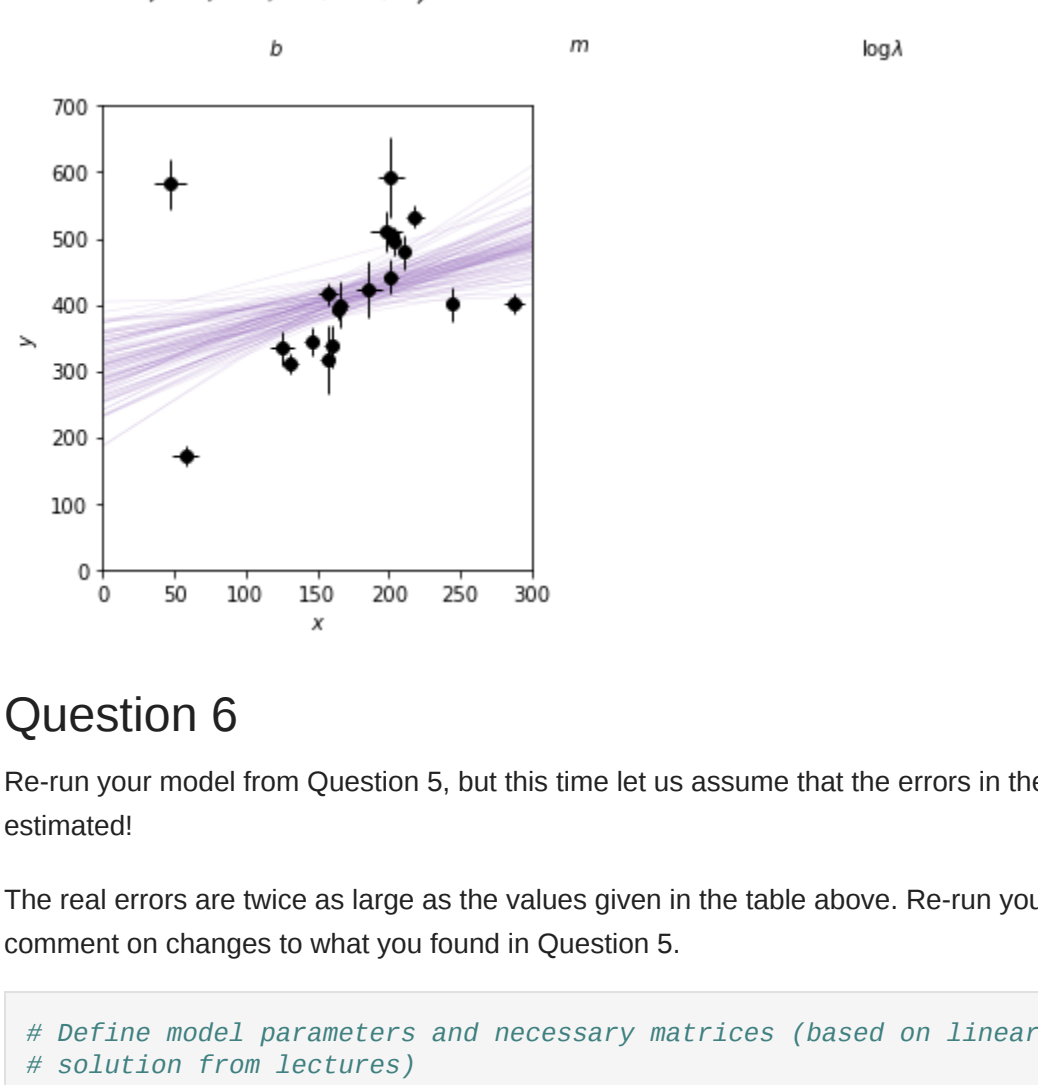
Implement the model specified in Question 4 in a programming language of your choice.

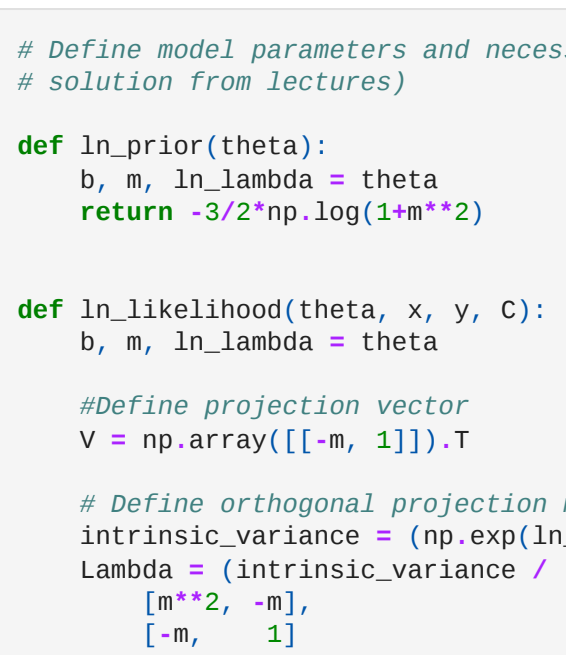
Specify a sensible initial guess for the model parameters, and then optimise those parameters using an optimisation algorithm of your choice. Use an off-the-shelf Markov-chain Monte Carlo package to sample the model posteriors.

Plot the posterior distributions of the model parameters, and plot the posterior predictions of the model compared to the data.

In [7]:

```
# Define model parameters and necessary matrices (based on linear algebra
# solution from lectures)

def ln_prior(theta):
    b, m, ln_lambda = theta
    return -3/2*np.log(1+m**2)


def ln_likelihood(theta, x, y, C):
    b, m, ln_lambda = theta

    #Define projection vector
    V = np.array([[-m, 1]]).T

    # Define orthogonal projection matrix.
    intrinsic_variance = (np.exp(ln_lambda)**2)
    Lambda = (intrinsic_variance / (1 + m**2)) * np.array([
        [m**2, -m],
        [-m,    1]
    ])

    Delta = (y - m*x -b)
    Sigma = (V.T @ (C + Lambda) @ V).flatten()


    return np.sum(-np.log(Sigma)-0.5*Delta**2/Sigma)

def ln_probability(theta, x, y, C):
    lp = ln_prior(theta)
    if not np.isfinite(lp):
        return lp
    return lp + ln_likelihood(theta, x, y, C)

Y = np.atleast_2d(y).T
A = np.vstack([np.ones_like(x), x]).T
C = np.diag(y_err * y_err)
C_inv = np.linalg.inv(C)
G = np.linalg.inv(A.T @ C_inv @ A)
X = G @ (A.T @ C_inv @ Y)


#Define covariance matrix (note that off diagonal terms = 0 since the
#x and y uncertainties are uncorrelated)
covs = np.array([[[x_e**2, x_e*y_e*rho*0],
                  [x_e*y_e*rho*0, y_e**2]] for y_e, x_e, rho in zip(y_err, x_err, rho_xy)])


#Optimise while taking lambda = -3 for initialisation
args = (x,y,covs)
initial_theta = np.hstack([X.T[0], -3])

result = op.minimize(lambda *args: -ln_probability(*args), initial_theta,
                     args=args,
                     method="L-BFGS-B",
                     bounds = [(None, None), (None, None), (-10, 10)])



#Sample model posteriors
ndim, nwalkers = (result.x.size, 32)
p0 = [result.x + 1e-5 * np.random.randn(ndim) for k in range(nwalkers)]

sampler = emcee.EnsembleSampler(
    nwalkers,
    ndim,
    ln_probability,
    args=args
)

# Run the burn-in.
pos, *_ = sampler.run_mcmc(p0, 500)
sampler.reset()

# Run production.
sampler.run_mcmc(pos, 1000)

# Make a corner plot.
chain = sampler.chain.reshape((-1, ndim))

fig = corner(
    chain,
    labels=(r"$b$", r"$m$", r"$\log{\lambda}$")
)

#Make posterior predictions
fig, ax = plt.subplots(figsize=(4, 4))

ax.scatter(x, y, c="k", s=10)
ax.errorbar(x, y,
            xerr=x_err, yerr=y_err,
            fmt="o", lw=1, c="k")

xlim = np.array([0, 300])
ax.set_xlabel(r"$x$")
ax.set_ylabel(r"$y$")

ax.xaxis.set_major_locator(MaxNLocator(6))
ax.yaxis.set_major_locator(MaxNLocator(7))

# Plot draws of the posterior.
for index in np.random.choice(chain.shape[0], size=100):
    b, m, ln_lambda = chain[index]
    ax.plot(
        xlim,
        m * xlim + b,
        "-",
        c="tab:purple",
        alpha=0.2,
        lw=0.5,
        zorder=-1
    )

ax.set_xlim(*xlim)
ax.set_ylim(0, 700)

fig.tight_layout()
```





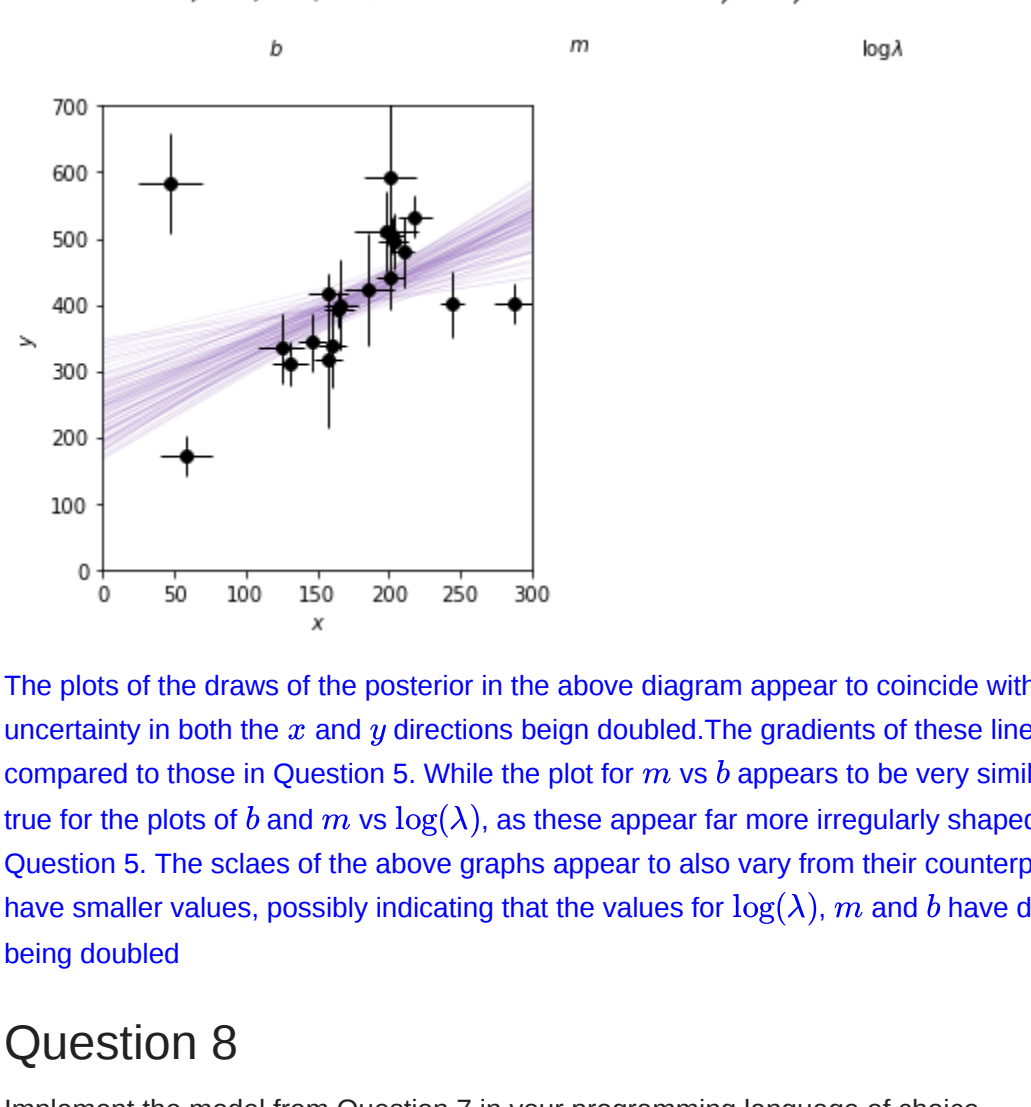
Question 6

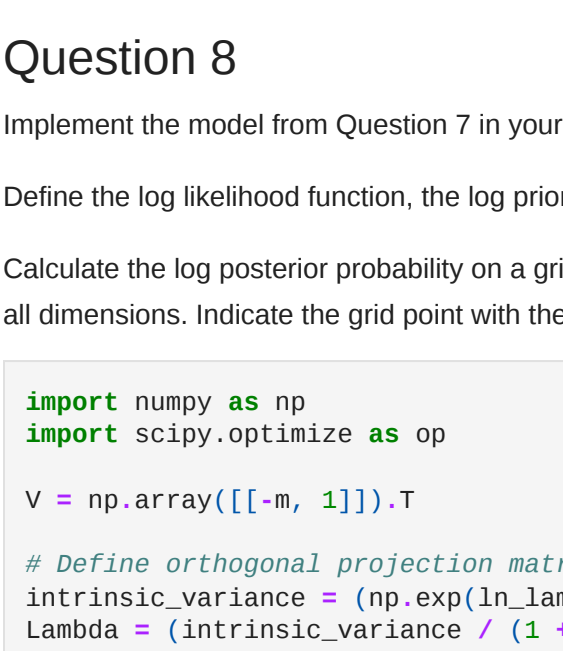
Re-run your model from Question 5, but this time let us assume that the errors in the x and y direction were incorrectly under-estimated!

The real errors are twice as large as the values given in the table above. Re-run your inferences with the real error values, and comment on changes to what you found in Question 5.

In [10]:

```
# Define model parameters and necessary matrices (based on linear algebra
# solution from lectures)

def ln_prior(theta):
    b, m, ln_lambda = theta
    return -3/2*np.log(1+m**2)


def ln_likelihood(theta, x, y, C):
    b, m, ln_lambda = theta

    #Define projection vector
    V = np.array([[-m, 1]]).T

    # Define orthogonal projection matrix.
    intrinsic_variance = (np.exp(ln_lambda**2))
    Lambda = (intrinsic_variance / (1 + m**2)) * np.array([
        [m**2, -m],
        [-m,    1]
    ])

    Delta = (y - m*x -b)
    Sigma = (V.T @ (C + Lambda) @ V).flatten()


    return np.sum(-np.log(Sigma)-0.5*Delta**2/Sigma)

def ln_probability(theta, x, y, C):
    lp = ln_prior(theta)
    if not np.isfinite(lp):
        return lp
    return lp + ln_likelihood(theta, x, y, C)

Y = np.atleast_2d(y).T
A = np.vstack([np.ones_like(x), x]).T
C = np.diag(y_err * y_err)
C_inv = np.linalg.inv(C)
G = np.linalg.inv(A.T @ C_inv @ A)
X = G @ (A.T @ C_inv @ Y)


#Define covariance matrix (note that off diagonal terms = 0 since the
#x and y uncertainties are uncorrelated)
covs = np.array([[[4*(x_e**2), x_e*y_e*rho*0],
                  [x_e*y_e*rho*0, 4*(y_e**2)]] for y_e, x_e, rho in zip(y_err, x_err, rho_xy)])


#Optimise while taking lambda = -3 for initialisation
args = (x,y,covs)
initial_theta = np.hstack([X.T[0], -3])

result = op.minimize(lambda *args: -ln_probability(*args), initial_theta,
                     args=args,
                     method="L-BFGS-B",
                     bounds = [(None, None), (None, None), (-10, 10)])



#Sample model posteriors
ndim, nwalkers = (result.x.size, 32)
p0 = [result.x + 1e-5 * np.random.randn(ndim) for k in range(nwalkers)]

sampler = emcee.EnsembleSampler(
    nwalkers,
    ndim,
    ln_probability,
    args=args
)

# Run the burn-in.
pos, *_ = sampler.run_mcmc(p0, 1000)
sampler.reset()

# Run production.
sampler.run_mcmc(pos, 1000)

# Make a corner plot.
chain = sampler.chain.reshape((-1, ndim))

fig = corner(
    chain,
    labels=(r"$b$", r"$m$", r"$\log{\lambda}$")
)

#Make posterior predictions
fig, ax = plt.subplots(figsize=(4, 4))

ax.scatter(x, y, c="k", s=10)
ax.errorbar(x, y,
            xerr=2*x_err, yerr=2*y_err,
            fmt="o", lw=1, c="k")

xlim = np.array([0, 300])
ax.set_xlabel(r"$x$")
ax.set_ylabel(r"$y$")

ax.xaxis.set_major_locator(MaxNLocator(6))
ax.yaxis.set_major_locator(MaxNLocator(7))

# Plot draws of the posterior.
for index in np.random.choice(chain.shape[0], size=100):
    b, m, ln_lambda = chain[index]
    ax.plot(
        xlim,
        m * xlim + b,
        "-",
        c="tab:purple",
        alpha=0.2,
        lw=0.5,
        zorder=-1
    )

ax.set_xlim(*xlim)
ax.set_ylim(0, 700)

fig.tight_layout()
```





The plots of the draws of the posterior in the above diagram appear to coincide with a larger number of data points despite the uncertainty in both the x and y directions beign doubled.The gradients of these linear graphs appear to approach a consistent value as compared to those in Question 5. While the plot for m vs b appears to be very similarly shaped to that of in Question 5, the same is not true for the plots of b and m vs $\log(\lambda)$, as these appear far more irregularly shaped compared to their elliptical counterparts in Question 5. The sclaes of the above graphs appear to also vary from their counterparts in Question 5 and in general, these appear to have smaller values, possibly indicating that the values for $\log(\lambda)$, m and b have decreased due to the uncertainties in both directions being doubled

Question 8

Implement the model from Question 7 in your programming language of choice.

Define the log likelihood function, the log prior function, and the log posterior probability function.

Calculate the log posterior probability on a grid in θ using sensible step sizes in each dimension, and plot the log posterior probability in all dimensions. Indicate the grid point with the highest log probability.

In [16]:

```
import numpy as np
import scipy.optimize as op

V = np.array([[-m, 1]]).T

# Define orthogonal projection matrix.
intrinsic_variance = (np.exp(ln_lambda)**2)
Lambda = (intrinsic_variance / (1 + m**2)) * np.array([
        [m**2, -m],
        [-m,    1]
    ])

Delta = (y - m*x -b)
Sigma = (V.T @ (C + Lambda) @ V).flatten()

np.random.seed(0)

_, x, y, y_err, x_err, rho_xy = np.array([
    [1,  201, 592, 61,  9, -0.84],
    [2,  244, 401, 25,  4, +0.31],
    [3,   47, 583, 38, 11, +0.64],
    [4,  287, 402, 15,  7, -0.27],
    [5,  203, 495, 21,  5, -0.33],
    [6,   58, 173, 15,  9, +0.67],
    [7,  210, 479, 27,  4, -0.02],
    [8,  202, 504, 14,  4, -0.05],
    [9,  198, 510, 30, 11, -0.84],
    [10, 158, 416, 16,  7, -0.69],
    [11, 165, 393, 14,  5, +0.30],
    [12, 201, 442, 25,  5, -0.46],
    [13, 157, 317, 52,  5, -0.03],
    [14, 131, 311, 16,  6, +0.50],
    [15, 166, 400, 34,  6, +0.73],
    [16, 160, 337, 31,  5, -0.52],
    [17, 186, 423, 42,  9, +0.90],
    [18, 125, 334, 26,  8, +0.40],
    [19, 218, 533, 16,  6, -0.78],
    [20, 146, 344, 22,  5, -0.56],
]).T




#Define covariance matrix with correlation between x and y uncertainties
C = np.array([[[x_e**2, x_e*y_e*rho],
                  [x_e*y_e*rho, y_e**2]] for
              y_e, x_e, rho in zip(y_err, x_err, rho_xy)])


def ln_prior(theta):
    b, m, Q, mu_o, ln_sigma_v, ln_lambda = theta
    if not (1 > Q > 0) \
    or not (700 > mu_o > 0) \
    or not (5 > ln_sigma_v > -5) \
    or not (10 > ln_lambda > -10):
        return -np.inf

    return -3/2 * np.log(1 + m**2)

def ln_likelihood_fg(theta, x, y, C):
    b, m, Q, mu_o, ln_sigma_v,ln_lambda = theta

    return -0.5*((np.log(Sigma)-(Delta**2/Sigma)))


def ln_likelihood_bg(theta, x, y, y_err):
    b, m,  Q, mu_o, ln_sigma_v,ln_lambda = theta
    total_variance = y_err**2 + (np.exp(ln_sigma_v))**2
    return -0.5 * ((y - mu_o)**2 / total_variance) - 0.5 * np.log(total_variance)

def ln_probability_q8(theta, x, y, y_err, C):
    lp = ln_prior(theta)
    if not np.isfinite(lp):
        return (-np.inf, np.nan * np.ones((2, x.size)))

    b, m, Q, mu_o, ln_sigma_v, ln_lambda = theta
    # Compute weighted foreground likelihoods for each data point.
    # (They are weighted by the log(Q) term)
    ll_fg = np.log(Q) + ln_likelihood_fg(theta, x, y, C)

    # Compute weighted background likelihoods for each data point.
    ll_bg = np.log(1 - Q) + ln_likelihood_bg(theta, x, y, y_err)

    # Sum the log of the sum of the exponents of the log likelihoods.
    ll = np.sum(np.logaddexp(ll_fg, ll_bg))
    return (lp + ll, np.vstack([ll_fg, ll_bg]))


args = (x, y, y_err, C)
initial_theta = np.array([250, 1, 0.5, 400, -3])
# Use optimize.brute to determine the -ve log likelihood is minimised
result = op.brute(
    lambda *args: -ln_probability_q8(*args)[0],
    args=args,
    ranges=[
        (0, 1000),
        (0, 1000),
        (0, 1),
        (0, 700),
        (-5, 5),
        (-10, 10)
    ], Ns = 10
)

#Unsure of how to plot and log_likelihood background function appears to produce
# a value error when I input the Sigma and Delta matrices from the intrinsic scatter
# model

for i in range(len(*args[0])):
    plt.contour(-ln_probability_q8(*args, C)[0][i], -ln_probability_q8)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-8f3351e6a012> in <module>
     12
     13 Delta = (y - m*x -b)
---> 14 Sigma = (V.T @ (C + Lambda) @ V).flatten()
     15
     16 np.random.seed(0)

ValueError: operands could not be broadcast together with shapes (20,20) (2,2)
```

In [ ]: