solutions to the exercises in "Data analysis recipes: Fitting a model to data"

Julia implementation of solutions to the exercises in David Hogg's 2010 tutorial paper on fitting a model to data

PTA

Table of contents

preliminary

source code of this document: https://github.com/phineas-pta/hogg2010/blob/main/hogg2010.qmd

this document's computations were carried out with `julia` 1.11.5, and its rendering was achieved via `quarto` 1.7.31

references & resources

reference: **David W. Hogg**, **Jo Bovy**, **Dustin Lang**. 2010. *Data analysis recipes: Fitting a model to data*

- paper: https://arxiv.org/pdf/1008.4686
- source code: https://github.com/davidwhogg/DataAnalysisRecipes/blob/master/straightline/straightline.tex
- lecture: https://www.youtube.com/playlist?list=PLBB44462E5201DD1E

you will also find here other papers from the *Data analysis recipes* series by David W. Hogg, and their original TeX files are available in the GitHub repository linked above

- https://arxiv.org/pdf/2005.14199
- https://arxiv.org/pdf/1710.06068
- https://arxiv.org/pdf/1205.4446
- https://arxiv.org/pdf/0807.4820

additional resources: the following are various python implementations i've come across, but it's worth noting that they often do not include solutions for every exercise

- https://astrowizici.st/teaching/phs5000/
- https://www.pymc.io/projects/examples/en/latest/generalized_linear_models/GLM-robust-with-outlier-detection.html
- https://github.com/binado/fitaline/blob/main/fitaline.ipynb
- https://github.com/wdconinc/data-analysis-recipes-fitting-a-model-to-data/blob/master/Exercises_in_%22Data_Analysis_Recipes_Fitting_a_Model_to_Data%22.ipynb
- https://github.com/nhuntwalker/data_analysis_recipes/blob/master/worked_exercises.ipynb
- https://github.com/jimbarrett27/hogg2010/blob/master/solutions.ipynb
- https://github.com/nadanai263/fittingmodeltodata/blob/master/Fitting%20a%20model%20to%20data%20(Hogg%2C%20Bovy%2C%20Lang%202010).ipynb
- https://github.com/Lachimax/hblfit/blob/main/notebooks/1-standard-practise.ipynb

data

data to be used throughout the tutorial:

Table 1: data

x	y	σ_x	σ_y	ρ_{xy}
201	592	9	61	-0.84
244	401	4	25	0.31
47	583	11	38	0.64
287	402	7	15	-0.27
203	495	5	21	-0.33

x	y	σ_x	σ_y	ρ_{xy}
58	173	9	15	0.67
210	479	4	27	-0.02
202	504	4	14	-0.05
198	510	11	30	-0.84
158	416	7	16	-0.69
165	393	5	14	0.30
201	442	5	25	-0.46
157	317	5	52	-0.03
131	311	6	16	0.50
166	400	6	34	0.73
160	337	5	31	-0.52
186	423	9	42	0.90
125	334	8	26	0.40
218	533	6	16	-0.78
146	344	5	22	-0.56

The full uncertainty covariance matrix for each data point is given by $\begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix}$ with $\sigma_{xy} = \rho_{xy}\sigma_x\sigma_y$

given the authors' academic credentials, these data most likely stem from astrophysics, with each measurement of x and y possessing uncertainties and correlations, and potentially subject to some additional manipulation

prepare Julia environment

```
import Pkg
Pkg.add(["Turing", "StatsPlots", "LogExpFunctions", "KernelDensity", #="MarkdownTables"=#])
```

```
using Statistics: mean, var, quantile
using LinearAlgebra: Diagonal, dot, logabsdet, eigen, transpose, isposdef
using LogExpFunctions: logaddexp, logsumexp
using KernelDensity: kde
using Turing, StatsPlots
# using MarkdownTables: markdown_table
setprogress!(false); # disable progress bar in Turing.jl
```

```
[ Info: [Turing]: progress logging is disabled globally
[ Info: [AdvancedVI]: global PROGRESS is set as false
```

data entry

```
const N = 20;
const x    = [201., 244.,  47., 287., 203.,  58., 210., 202., 198., 158., 165., 201., 157., 131., 166.,
↪  160., 186., 125., 218., 146.];
const y    = [592., 401., 583., 402., 495., 173., 479., 504., 510., 416., 393., 442., 317., 311., 400.,
↪  337., 423., 334., 533., 344.];
const σ_x  = [  9.,   4.,  11.,   7.,   5.,   9.,   4.,   4.,  11.,   7.,   5.,   5.,   5.,   6.,   6.,
↪  5.,   9.,   8.,   6.,   5.];
const σ_y  = [ 61.,  25.,  38.,  15.,  21.,  15.,  27.,  14.,  30.,  16.,  14.,  25.,  52.,  16.,  34.,
↪  31.,  42.,  26.,  16.,  22.];
const ρ_xy = [-.84,  .31,  .64, -.27, -.33,  .67, -.02, -.05, -.84, -.69,   .3, -.46, -.03,   .5,  .73,
↪  -.52,   .9,   .4, -.78, -.56];
```

data but in tensor format, to be useful in sections 7 & 8

```
const Z = [[x[i], y[i]] for i ∈ 1:N];
const S = [ # covariance tensor
    [
        σ_x[i]^2                ρ_xy[i]*σ_x[i]*σ_y[i];
        ρ_xy[i]*σ_x[i]*σ_y[i]   σ_y[i]^2
    ]
    for i ∈ 1:N
];
```

data points 5 through 20 in the table, which exclude outliers, to be used in various exercises

```
const N′ = 16;
const x′    = x[   5:end];
const y′    = y[   5:end];
const σ_x′  = σ_x[ 5:end];
const σ_y′  = σ_y[ 5:end];
const ρ_xy′ = ρ_xy[5:end];
const Z′    = Z[   5:end];
const S′    = S[   5:end];
```

options to run MCMC chains

```
const N_samples = 5000;
const N_chains = 4;
const N_warmup = 1000;
const N_thinning = 5;
const N_bins = max(isqrt(N_samples), N_samples ÷ 10); # in histogram
```

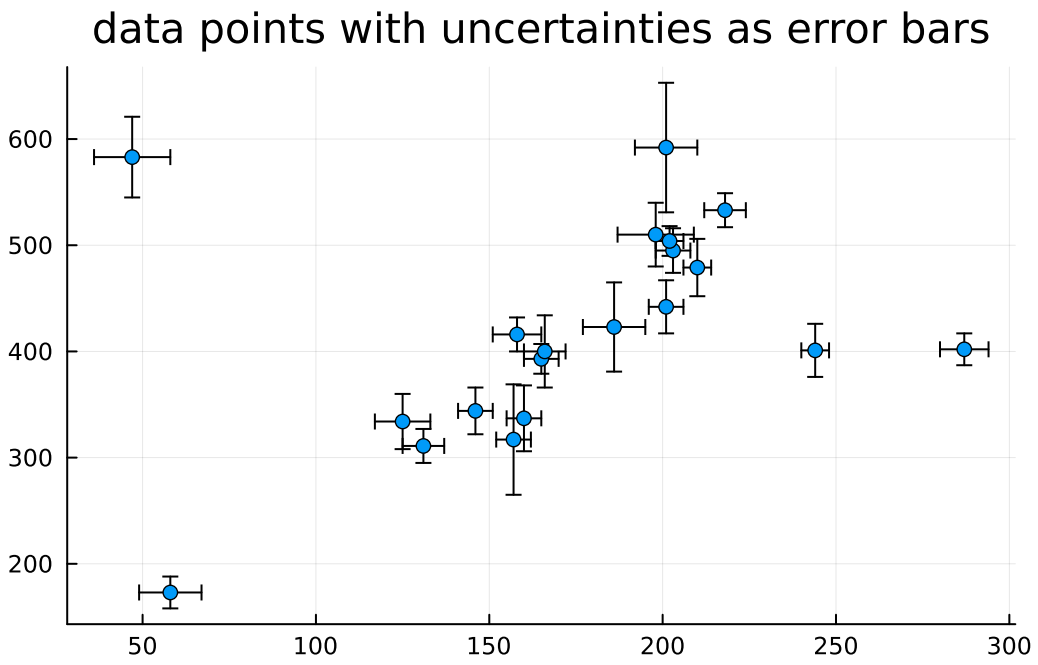
helper functions

```
"""print value with its uncertainty"""
function print_uncertainty(𝕩, δ𝕩; digits=2)
    return "$(round(𝕩; digits=digits)) ± $(round(δ𝕩; digits=digits))"
end

"""scatter plot with uncertainties as ellipses"""
function plot_ellipses(ℕ, 𝕩, 𝕪, ℤ, 𝕊; n_σ=1)
    p = scatter(𝕩, 𝕪, label=nothing)
    for i ∈ 1:ℕ
        covellipse!(p, ℤ[i], 𝕊[i], n_std=n_σ, label=nothing)
    end
    return p
end

"""flatten MCMC chain from matrix with N_samples rows and N_chains columns to a vector of length
↪  N_samples × N_chains"""
function flatten_chains(mcmc_chain)
    return reduce(vcat, mcmc_chain.data)
end

#=
# formatter to use with MarkdownTables
value_to_markdown(𝕩::Number) = "$(round(𝕩; digits=3))"
value_to_markdown(𝕩::AbstractString) = 𝕩
value_to_markdown(𝕩) = "`$𝕩`"
=#
```

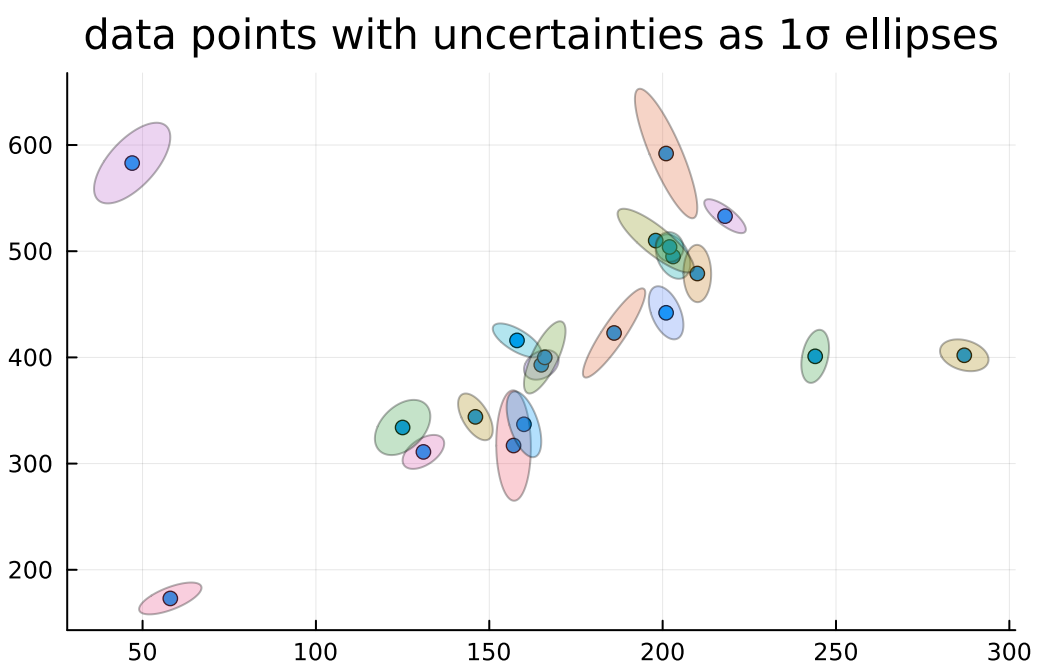
```

"""print summary of MCMC chains with interesting values"""
function print_summary(mcmc_chains)
    # return markdown_table(summarystats(mcmc_chains)[:, [:mean, :std, :mcse, :rhat]];
    ↪ formatter=value_to_markdown)
    return summarystats(mcmc_chains)[:, [:mean, :std, :mcse, :rhat]]
end

function print_information_criterion(data_model, mcmc_chains)
    # posterior log likehood
    post_log𝓛 = loglikelihood(data_model, mcmc_chains) # shape: N_samples × N_chains

    # Deviance Information Criterion
    # copied from
    ↪ https://github.com/StatisticalRethinkingJulia/ParetoSmoothedImportanceSampling.jl/blob/master/src/dic.j
    # deviance = (-2) .* post_log𝓛
    # DIC = mean(deviance) + var(deviance) / 2
    # given 𝔼(a+b𝕏) = a+b𝔼(𝕏) and 𝕍ar(a+b𝕏) = b²𝕍ar(𝕏) we can simplify the formula for DIC
    DIC = 2 * (var(post_log𝓛) - mean(post_log𝓛))

    # Widely Applicable Information Criterion / Watanabe–Akaike information criterion
    # copied from
    ↪ https://github.com/StatisticalRethinkingJulia/ParetoSmoothedImportanceSampling.jl/blob/master/src/waic.
    lppd = dropdims(logsumexp(post_log𝓛 .- log(size(post_log𝓛)[1]); dims=1); dims=1) # log pointwise
↪   predictive density
    pD = dropdims(var(post_log𝓛; dims=1, corrected=false); dims=1) # overfitting penalty
    WAIC = (-2) * sum(lppd - pD)

    println("Deviance Information Criterion: $DIC")
    println("Widely Applicable Information Criterion: $WAIC")
    println("Log predictive density: $(sum(lppd))")
    println("effective number of parameters: $(sum(pD))")
end

const x_min, x_max = extrema(x);
const y_min, y_max = extrema(y);
const angle90 = π / 2; # useful in sections 7 & 8
```

quick look at data

```
scatter(x, y, xerror=σ_x, yerror=σ_y, label=nothing, title="data points with uncertainties as error
↪   bars")
```




```
plot_ellipses(N, x, y, Z, S)
title!("data points with uncertainties as 1σ ellipses")
```



1 Standard practice

quick summary

- **Objective**: Fit a straight line $y = m\,x + b$ to data using χ^2 minimization.
- **Assumptions**:
  - Negligible uncertainties in x-direction.
  - Known, Gaussian uncertainties in y-direction.
  - Linear relationship with no intrinsic scatter.
- **Method**:
  - Solve using matrix algebra.
  - Uncertainties in m and b derived from the covariance matrix.
- **Limitations**: Rarely valid in practice due to real-world data complexities (e.g., outliers, unknown uncertainties).

conventional method for fitting a straight line $y = m\,x + b$ to data points where only the y-values have known Gaussian uncertainties

Construct the matrices:

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

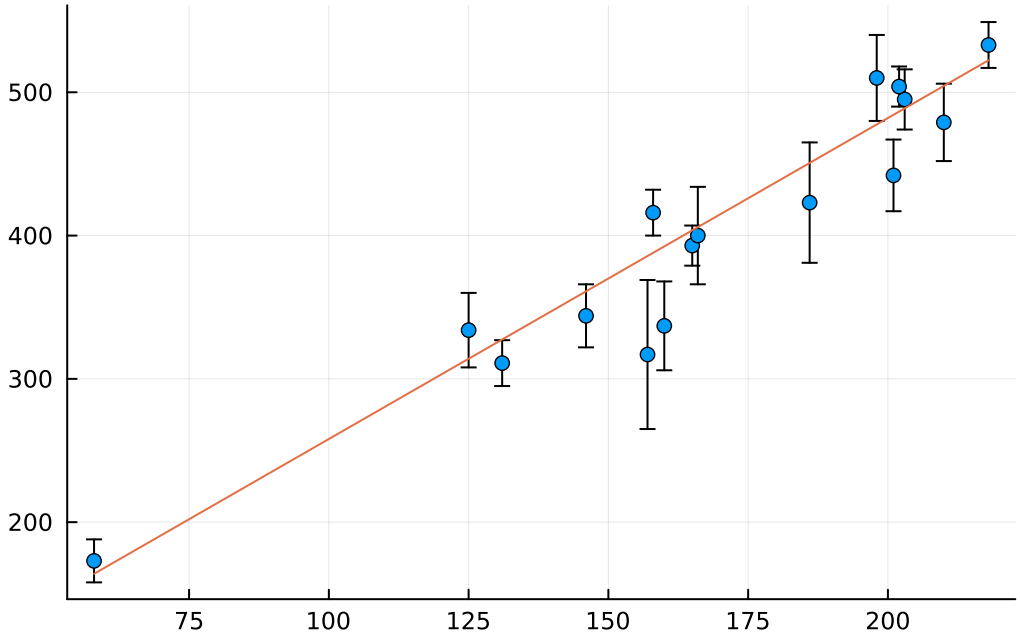
best-fit slope m and intercept b is given by: $\begin{bmatrix} b \\ m \end{bmatrix} = X = \left[A^\top C^{-1} A\right]^{-1} \left[A^\top C^{-1} Y\right]$

1.1 exercise 01

> Using the standard linear algebra method, fit the straight line $y = m\,x + b$ to the x, y, and σ_y values for data points 5 through 20 in the table. That is, ignore the first 4 data points, and also ignore the columns for σ_x and ρ_{xy}. Make a plot showing the points, their uncertainties, and the best-fit line. What is the standard uncertainty variance σ_m^2 on the slope of the line?

```
Yex01 = reshape(y′, :, 1); # vector to 1-column matrix
Aex01 = hcat(ones(N′), x′);
Cex01 = Diagonal(σ_y′ .^ 2);

tmp = transpose(Aex01) * inv(Cex01); # intermediary result
cov_bm = inv(tmp * Aex01); # covariance matrix of b & m
Xex01 = cov_bm * tmp * Yex01;
bex01, mex01 = Xex01;
```

```
scatter(x′, y′, yerror=σ_y′, label=nothing)
plot!(𝕩 -> mex01*𝕩 + bex01, label=nothing)
```



The "standard uncertainty variance" is to be found in the diagonals of the matrix: $\begin{bmatrix} \sigma_b^2 & \sigma_{mb} \\ \sigma_{mb} & \sigma_m^2 \end{bmatrix} = \left[A^\top C^{-1} A\right]^{-1}$

```
cov_bm
```

```
2×2 Matrix{Float64}:
 332.923    -1.88954
  -1.88954   0.0116166
```

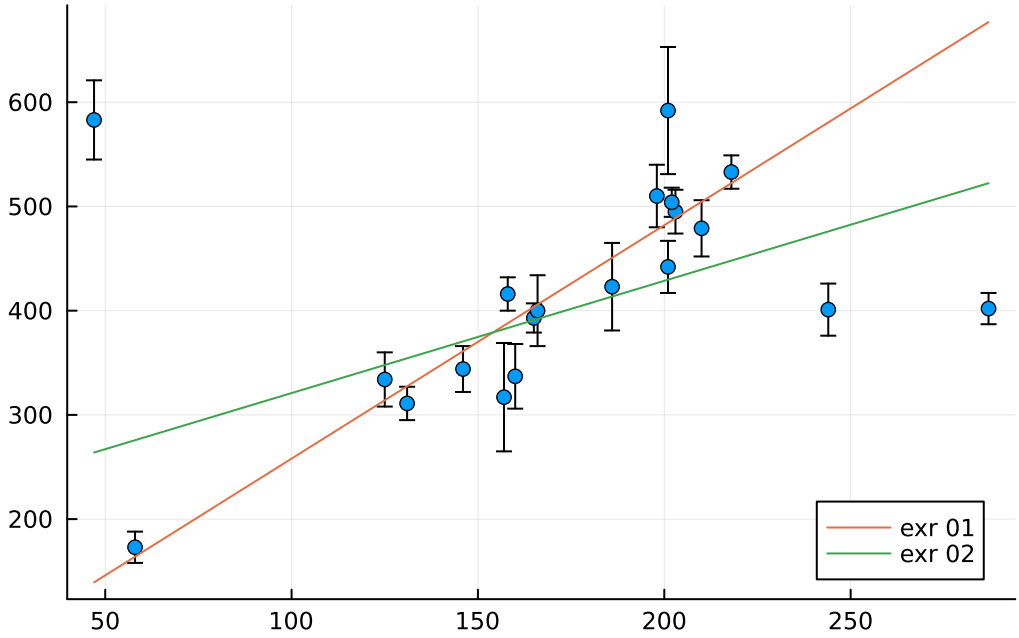
standard uncertainty (square root of variance) on b is 18.246166749268173, on m is 0.10778047654050076

i.e. $b = 34.05 \pm 18.25$ and $m = 2.24 \pm 0.11$

1.2 exercise 02

> Repeat exercise 01 but for all the data points in the table. What is the standard uncertainty variance σ_m^2 on the slope of the line? Is there anything you don't like about the result? Is there anything different about the new points you have included beyond those used in exercise 01?

```
Yex02 = reshape(y, :, 1); # vector to 1-column matrix
Aex02 = hcat(ones(N), x);
Cex02 = Diagonal(σ_y .^ 2);

tmp = transpose(Aex02) * inv(Cex02); # intermediary result
cov_bm = inv(tmp * Aex02); # covariance matrix of b & m
Xex02 = cov_bm * tmp * Yex02;
bex02, mex02 = Xex02;

scatter(x, y, yerror=σ_y, label=nothing)
plot!(𝕩 -> mex01*𝕩 + bex01, label="exr 01")
plot!(𝕩 -> mex02*𝕩 + bex02, label="exr 02")
```



new values for the fitted line: $b = 213.27 \pm 14.39$ and $m = 1.08 \pm 0.08$

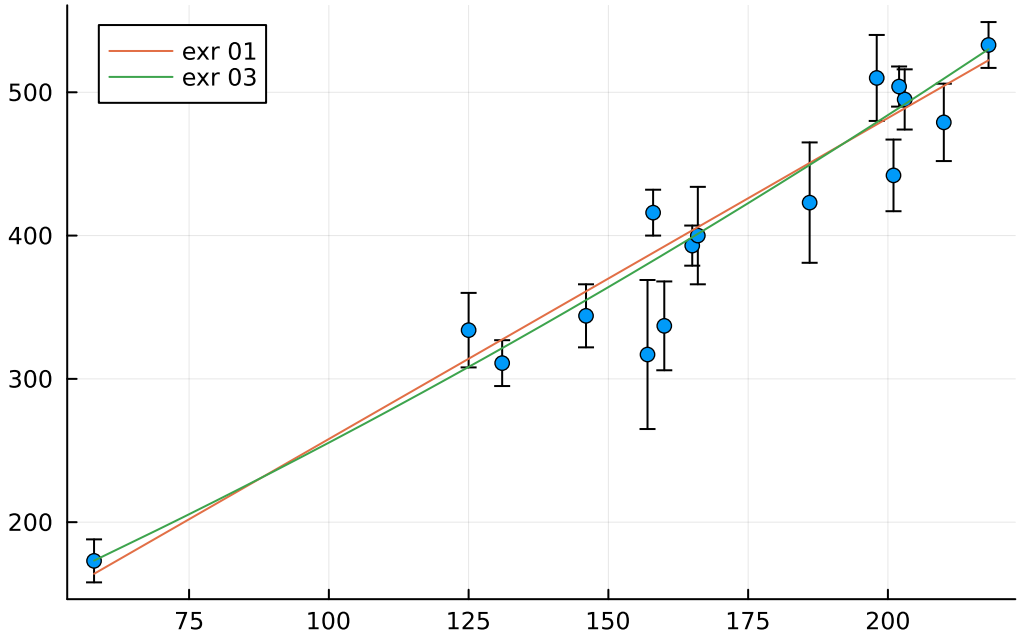
the outliers drastically affect the fit:

- the slope is reduced by more than half and the intercept is adjusted by ~200 to compensate.
- uncertainties actually decrease

1.3 exercise 03

> Generalize the method of this section to fit a general quadratic (2nd order) relationship. Add another column to matrix A containing the values x_i^2, and another element to vector X (call it q). Then re-do exercise 01 butfitting for and plotting the best quadratic relationship $q\,x^2 + m\,x + b$

```
Yex03 = reshape(y', :, 1); # vector to 1-column matrix
Aex03 = hcat(ones(N'), x', x'.^2);
Cex03 = Diagonal(σ_y' .^ 2);

tmp = transpose(Aex03) * inv(Cex03); # intermediary result
cov_bmq = inv(tmp * Aex03); # covariance matrix of b, m & q
Xex03 = cov_bmq * tmp * Yex03;
bex03, mex03, qex03 = Xex03;

scatter(x', y', yerror=σ_y', label=nothing)
plot!(𝕩 -> mex01*𝕩 + bex01, label="exr 01")
plot!(𝕩 -> qex03*𝕩^2 + mex03*𝕩 + bex03, label="exr 03")
```



$b = 72.89 \pm 38.91$ and $m = 1.6 \pm 0.58$ and $q = 0.0023 \pm 0.002$

2 The objective function

quick summary

objective function: to measure how well a model fits the data → optimize this function to find best-fit model

for the standard line-fitting case (Gaussian errors in y), best-fit model = maximizing the likelihood of observing the data given the line parameters (m, b) which also minimizes the χ^2 value, which represents the weighted sum of squared differences between the data and the fitted line

2.1 exercise 04

> Imagine a set of N measurements t_i, with uncertainty variances $\sigma_{t_i}^2$, all of the same (unknown) quantity T. Assuming the generative model that each t_i differs from T by a Gaussian-distributed offset, taken from a Gaussian with zero mean and variance $\sigma_{t_i}^2$, write down an expression for the log likelihood $\log \mathcal{L}$ for the data given the model parameter T. Take a derivative and show that the maximum likelihood value for T is the usual weighted mean.

Gaussian likelihood:

$$\mathcal{L} = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma_{t_i}^2}} \exp\left(-\frac{(t_i - T)^2}{2\sigma_{t_i}^2}\right)$$

log-likelihood: when we are faced with the situation of taking the product of (potentially many) values close to zero, it's a good idea to take the logarithm and sum the values instead:

$$\log \mathcal{L} = -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log \sigma_{t_i}^2 - \sum_{i=1}^{N} \frac{(t_i - T)^2}{2\sigma_{t_i}^2}$$

bonus: $\log \mathcal{L} = \mathrm{K} - \frac{1}{2}\chi^2$ with K some constant

derivative:

$$\frac{\partial \log \mathcal{L}}{\partial T} = -\frac{\partial}{\partial T}\sum_{i=1}^{N} \frac{(t_i - T)^2}{2\sigma_{t_i}^2} = \sum_{i=1}^{N} \frac{(t_i - T)}{\sigma_{t_i}^2}$$

This takes its maximum when the derivative is 0

$$\sum_{i=1}^{N} \frac{(t_i - T)}{\sigma_{t_i}^2} = 0 = \sum_{i=1}^{N} \frac{t_i}{\sigma_{t_i}^2} - T\sum_{i=1}^{N} \frac{1}{\sigma_{t_i}^2}$$

we get the weighted mean as expected

$$\hat{T} = \frac{\sum_{i=1}^{N} \frac{t_i}{\sigma_{t_i}^2}}{\sum_{i=1}^{N} \frac{1}{\sigma_{t_i}^2}}$$

2.2 exercise 05

> Take the matrix formulation for χ^2 given in *equation (7)* in the paper and take derivatives to show that the minimum is at the matrix location given in *equation (5)* in the paper

Equation (7) is

$$\chi^2 = [Y - AX]^\top C^{-1}[Y - AX]$$

By the product rule, we have

$$\frac{\partial \chi^2}{\partial X} = \left(\frac{\partial}{\partial X}[Y - AX]^\top\right) C^{-1}[Y - AX] + [Y - AX]^\top \left(\frac{\partial}{\partial X} C^{-1}[Y - AX]\right)$$

Differentiating, we get

$$\frac{\partial \chi^2}{\partial X} = -A^\top C^{-1}[Y - AX][Y - AX]^\top C^{-1}[-A]$$

Multiplying out

$$\frac{\partial \chi^2}{\partial X} = -A^\top C^{-1} Y + A^\top C^{-1} AX - Y^\top C^{-1} A + [AX]^\top C^{-1} A$$

Since $-A^\top C^{-1} Y$ is a scalar, it equals its own transpose

$$\frac{\partial \chi^2}{\partial X} = -2A^\top C^{-1} Y + A^\top C^{-1} AX + X^\top A^\top C^{-1} A$$

The two terms in X are also simply transposes of one another, and can thus be trivially added together

$$\frac{\partial \chi^2}{\partial X} = -2A^\top C^{-1} Y + 2A^\top C^{-1} AX$$

X takes its maximum likelihood value when the derivative is 0:

$$-2A^\top C^{-1} Y + 2A^\top C^{-1} A\hat{X} = 0$$

Rearranging:

$$A^\top C^{-1} A\hat{X} = 2A^\top C^{-1} Y$$

And so

$$\hat{X} = \left(A^\top C^{-1} A\right)^{-1} A^\top C^{-1} Y$$

As given in *equation (5)*

3 Pruning outliers

quick summary

- **Generative Model**:
  - Treat data as a mixture of "good" (on the line) and "bad" (outliers) points.
  - Likelihood combines foreground (line) and background (outlier) distributions.
- **Priors**:
  - Flat priors for m, b; informative priors for outlier parameters (e.g., amplitude $P_{\rm bad}$, background variance $V_{\rm bad}$).
- **Marginalization**:
  - Use MCMC to integrate over nuisance parameters $(P_{\rm bad}, Y_{\rm bad}, V_{\rm bad})$ and obtain marginalized posterior for m, b.
  - Avoids arbitrary data rejection (e.g., sigma clipping) and provides robust uncertainties.

importance of modeling outliers (data points that don't fit the assumed model) rather than simply ignoring them

one of the well-known methods to deal with outliers: sigma clipping (iteratively removing points 1 /3 /5 far from the fit then re-fit then remove until stable) → a procedure which works very well but doesn't an objective function to be optimized, so not statiscally justifiable

authors' proposal: mixture model

- outliers come from a distribution with probability $P_{\rm bad}$, example of distribution: $\mathcal{N}(Y_{\rm bad}, V_{\rm bad})$
- inliers come from straight line with probability $1 - P_{\rm bad}$, therefore distribution: $\mathcal{N}\left(mx_i + b, \sigma_{y_i}^2\right)$

We now model our data as being drawn from a mixture of 2 Gaussians, one which is the 'true' relation and one which is a distribution of outliers. This mixture model is generated by marginalising an 'exponential' model which contains N latent classifying labels q_i for each data point

$$\mathcal{L} \propto \prod_{i=1}^{N} \left[\frac{1 - P_{\rm bad}}{\sqrt{2\pi\sigma_{y_i}^2}} \exp\left(-\frac{[y_i - mx_i - b]^2}{2\sigma_{y_i}^2} \right) + \frac{P_{\rm bad}}{\sqrt{2\pi[V_{\rm bad} + \sigma_{y_i}^2]}} \exp\left(-\frac{[y_i - Y_{\rm bad}]^2}{2[V_{\rm bad} + \sigma_{y_i}^2]} \right) \right]$$

3.1 exercise 06

Using the mixture model proposed in the paper — that treats the distribution as a mixture of a thin line containing a fraction $[1 - P_{\rm bad}]$ of the points and a broader Gaussian containing a fraction $P_{\rm bad}$ of the points — find the best-fit (the maximum a posteriori) straight line $y = m\,x + b$ for the x, y, and σ_y for the data.
Before choosing the MAP line, marginalize over parameters $(P_{\rm bad}, Y_{\rm bad}, V_{\rm bad})$. That is, if you take a sampling approach, this means sampling the full 5-dimensional parameter space but then choosing the peak value in the histogram of samples in the 2-dimensional parameter space (m, b).
Make one plot showing this 2-dimensional histogram, and another showing the points, their uncertainties, and the MAP line.
How does this compare to the standard result you obtained in exercise 02? Do you like the MAP line better or worse?
For extra credit, plot a sampling of 10 lines drawn from the marginalized posterior distribution for (m, b) (marginalized over $(P_{\rm bad}, Y_{\rm bad}, V_{\rm bad})$) and plot the samples as a set of light grey or ransparent lines.

```
# DRAFT (SHOULDN'T USE): manually modify log likehood
@model function model_ex06_draft(N, x, y, σ_y)
    b ~ Normal(0, 5)
    m ~ Normal(0, 5)

    # for outliers
    P_bad ~ Uniform(0, 1)
    Y_bad ~ Uniform(y_min, y_max) # mean for all outliers
    V_bad ~ InverseGamma(.001, .001) # additional variance for outliers

    for i ∈ 1:N
        ŷ_i = b + m * x[i]
        σ_bad_i = sqrt(σ_y[i]^2 + V_bad)

        # faster computation: marginalize discrete param
         inlier = log1p(-P_bad) + logpdf(Normal(ŷ_i    , σ_y[i]), y[i])
        outlier = log(   P_bad) + logpdf(Normal(Y_bad, σ_bad_i), y[i])
        Turing.@addlogprob! logaddexp(inlier, outlier)

    end
end
```

```
# recommendation: use convenient constructors for mixture models
@model function model_ex06(N, x, y, σ_y)
    b ~ Normal(0, 5)
    m ~ Normal(0, 5)

    # for outliers
    P_bad ~ Uniform(0, 1)
    Y_bad ~ Uniform(y_min, y_max) # mean for all outliers
    V_bad ~ InverseGamma(.001, .001) # additional variance for outliers

    for i ∈ 1:N
        ŷ_i = b + m * x[i]
        σ_bad_i = sqrt(σ_y[i]^2 + V_bad)
        y[i] ~ MixtureModel(Normal, [(ŷ_i, σ_y[i]), (Y_bad, σ_bad_i)], [1 - P_bad, P_bad])
    end
end

data_model_ex06 = model_ex06(N, x, y, σ_y);
chains_ex06 = sample(data_model_ex06, NUTS(), MCMCThreads(), N_samples, N_chains; num_warmup=N_warmup,
↪   thinning=N_thinning);
```

```
┌ Info: Found initial step size
└   ϵ = 0.2
┌ Info: Found initial step size
└   ϵ = 0.2
┌ Info: Found initial step size
└   ϵ = 0.2
┌ Info: Found initial step size
└   ϵ = 0.2
```

diagnostics of chains:

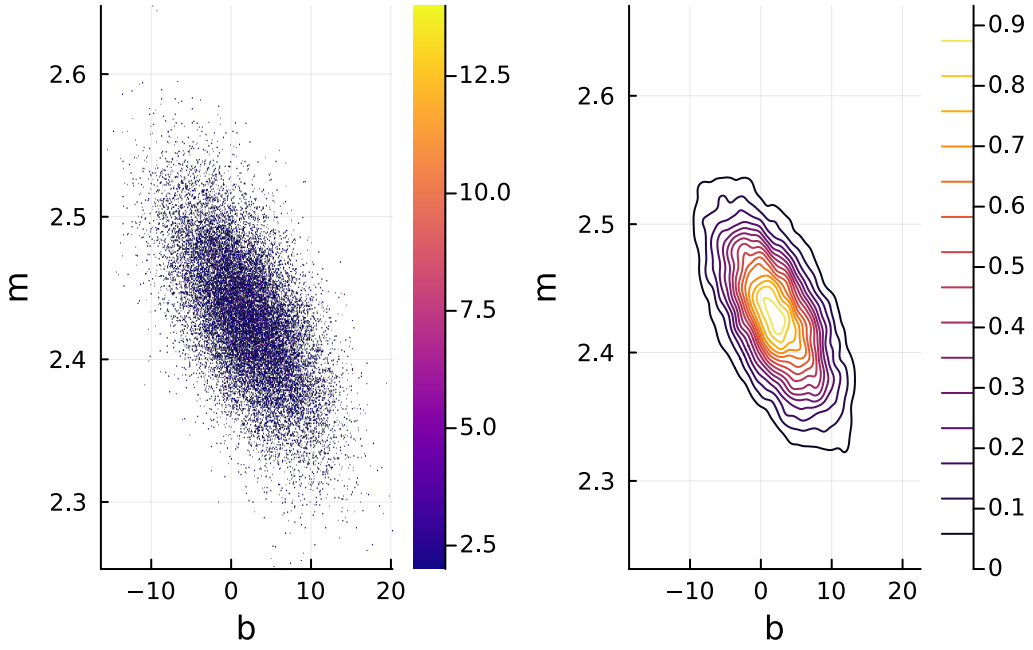
```
print_information_criterion(data_model_ex06, chains_ex06)
print_summary(chains_ex06)
```

```
Deviance Information Criterion: 221.7352952317259
Widely Applicable Information Criterion: 881.058088349788
Log predictive density: -431.39938552609465
effective number of parameters: 9.129658648799358

Summary Statistics
  parameters         mean          std       mcse      rhat
      Symbol      Float64      Float64    Float64   Float64

           b       2.0360       4.8289     0.0345    1.0001
           m       2.4291       0.0454     0.0003    1.0001
       P_bad       0.2955       0.1259     0.0009    1.0000
       Y_bad     442.6367      51.3189     0.3730    1.0000
       V_bad   13962.1324   63748.2562   456.7060    1.0000
```

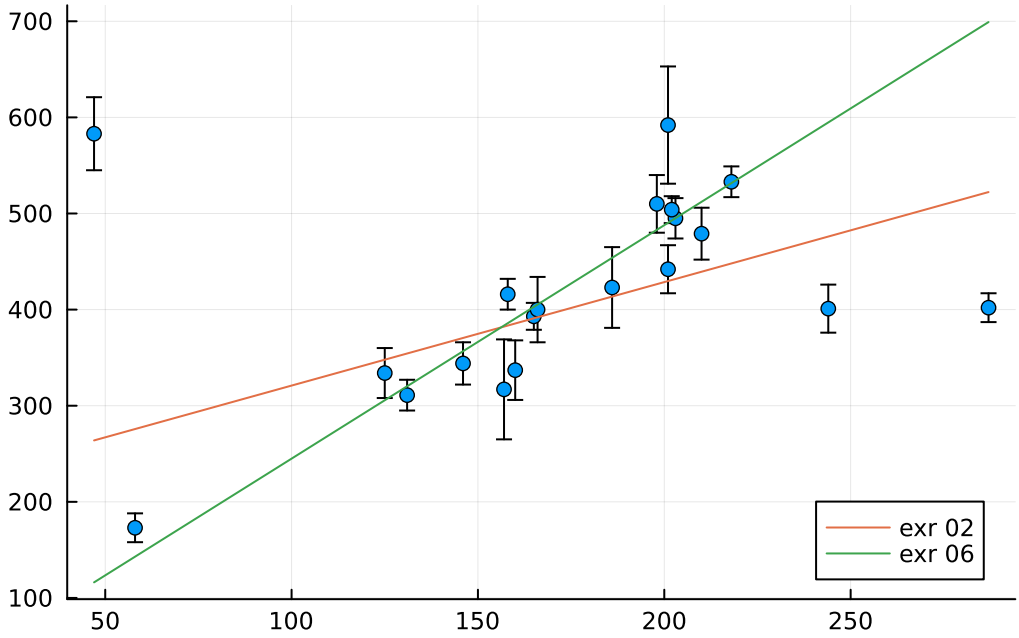
all chains seem converged, we're good!

good thing about MCMC is that it samples directly from marginalized distribution so no need to compute any integral after running

```
p = plot(layout=2)
histogram2d!(p, chains_ex06[:b], chains_ex06[:m], bins=N_bins, normalize=:pdf, color=:plasma, subplot=1, xlabel="b", ylabel="m")
plot!(p, kde((flatten_chains(chains_ex06[:b]), flatten_chains(chains_ex06[:m]))), subplot=2, xlabel="b", ylabel="m")
```




```
b_ex06 = mean(chains_ex06[:b]);
m_ex06 = mean(chains_ex06[:m]);
scatter(x, y, yerror=σ_y, label=nothing)
plot!(𝕩 -> m_ex02*𝕩 + b_ex02, label="exr 02")
plot!(𝕩 -> m_ex06*𝕩 + b_ex06, label="exr 06")
```



3.2 exercise 07

Solve exercise 06 but now plot the fully marginalized (over $m, b, Y_{\text{bad}}, V_{\text{bad}}$) posterior distribution function for parameter P_{bad}. Is this distribution peaked about where you would expect, given the data?
Now repeat the problem, but dividing all the data uncertainty variances σ_{yi}^2 by 4 (or dividing the uncertainties σ_{yi} by 2). Again plot the fully marginalized posterior distribution function for parameter P_{bad}. Discuss.

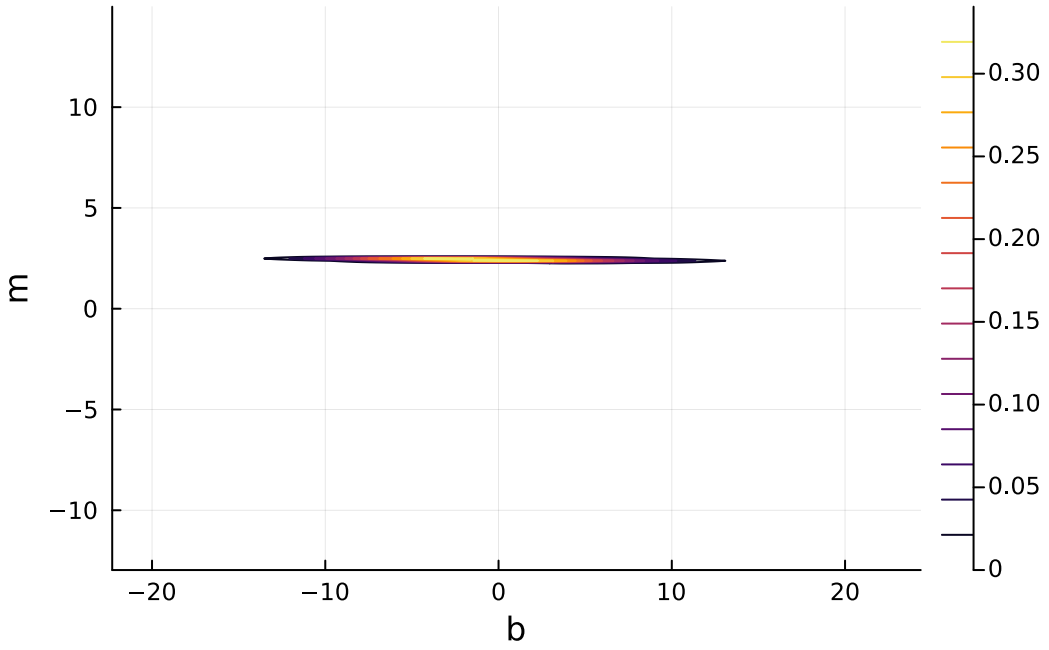
```
data_model_ex07 = model_ex06(N, x, y, σ_y ./ 2);
chains_ex07 = sample(data_model_ex07, NUTS(), MCMCThreads(), N_samples, N_chains; num_warmup=N_warmup,
↪  thinning=N_thinning);
println('=' ^ 80)
print_information_criterion(data_model_ex07, chains_ex07)
print_summary(chains_ex07)
```

```
┌ Info: Found initial step size
└   ϵ = 0.018750000000000003
┌ Info: Found initial step size
└   ϵ = 0.05
┌ Info: Found initial step size
└   ϵ = 0.05
┌ Info: Found initial step size
└   ϵ = 0.025
```

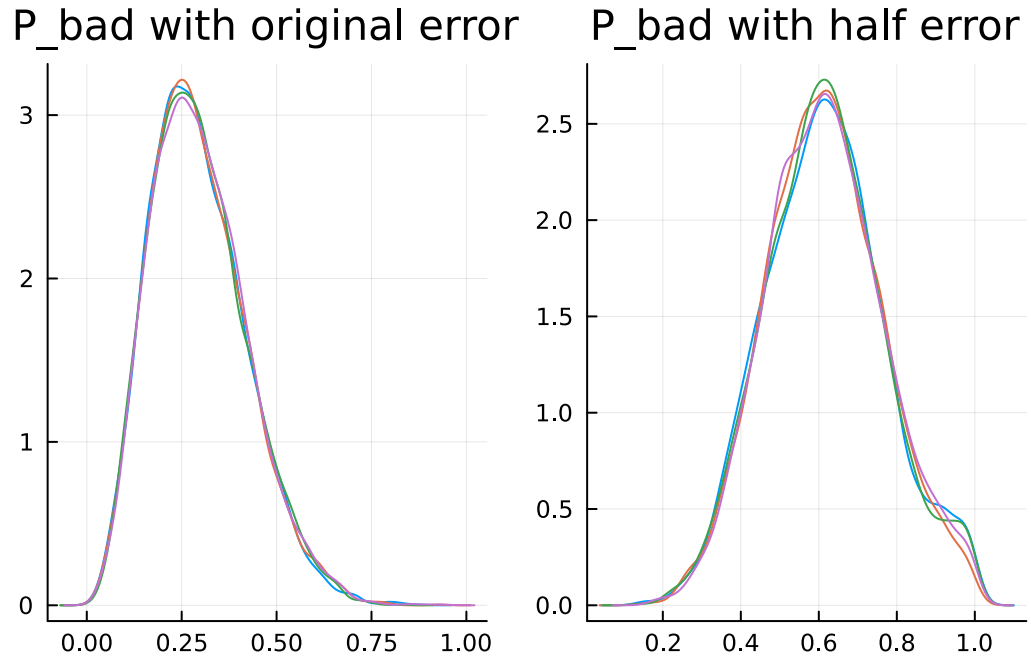
```
Deviance Information Criterion: 240.1627443618504
Widely Applicable Information Criterion: 953.061520252609
Log predictive density: -462.2800752497833
effective number of parameters: 14.250684876521213

Summary Statistics
  parameters         mean         std      mcse      rhat
      Symbol      Float64     Float64   Float64   Float64

           b      -0.5160      5.3314    0.0381    1.0000
           m       2.4016      0.9425    0.0251    1.0013
       P_bad       0.6163      0.1512    0.0028    1.0004
       Y_bad     417.2441     37.9225    0.2856    1.0000
```

```
V_bad    15136.0479    9283.7038    76.0338    1.0002
```

```
plot(kde((flatten_chains(chains_ex07[:b]), flatten_chains(chains_ex07[:m]))), xlabel="b", ylabel="m")
```




```
p = plot(layout=2)
density!(p, chains_ex06[:P_bad], subplot=1, label=nothing, title="P_bad with original error")
density!(p, chains_ex07[:P_bad], subplot=2, label=nothing, title="P_bad with half error")
```



with original error: peak at 0.2, arguably 3-4 outliers in the data, so it makes sense

with ½ original error: bigger value for the peak meaning that many more points have been considered outliers

little extra

maximum likelihood estimates:

```
maximum_likelihood(data_model_ex06)
```

```
ModeResult with maximized lp of -4066.13
[-8.324643296432741, 0.044615203183588636, 0.0818541357432142, 239.83585277205523, Inf]
```

maximum *a posteriori* estimates:

```
maximum_a_posteriori(data_model_ex06)
```

```
ModeResult with maximized lp of -Inf
[0.7959712976074101, -9.16487460309687, 0.8256362265152977, 497.5010434363468, Inf]
```

sampling from the prior distribution:

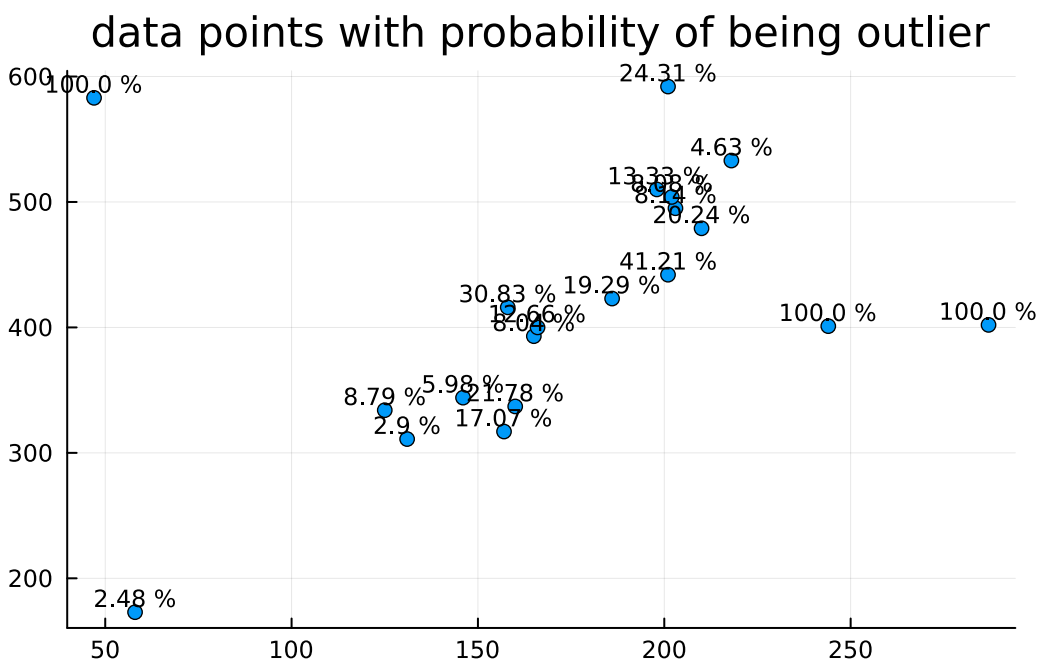
```
print_summary(sample(data_model_ex06, Prior(), N_samples))
```

```
Summary Statistics
  parameters       mean        std      mcse      rhat
      Symbol    Float64    Float64   Float64   Float64

           b    -0.1233     5.0757    0.0725    0.9998
           m     0.1104     4.9841    0.0705    1.0006
       P_bad     0.5036     0.2871    0.0042    1.0000
       Y_bad   381.8850   121.5185    1.7256    0.9998
       V_bad        Inf        NaN       NaN    1.0739
```

compute probability of each point to be classified as outlier:

```
# Compute the un-normalized log probabilities for each cluster
log_prob_assign_outliers = zeros(N_chains, N_samples, N);
Threads.@threads for step ∈ 1:N_samples
    for chain ∈ 1:N_chains
        b = chains_ex06[:b][step, chain]
        m = chains_ex06[:m][step, chain]
        V_bad = chains_ex06[:V_bad][step, chain]
        P_bad = chains_ex06[:P_bad][step, chain]
        Y_bad = chains_ex06[:Y_bad][step, chain]
        for i ∈ 1:N
            ŷ_i = b + m * x[i]
            inliers_log_prob = log1p(-P_bad) + logpdf(Normal(ŷ_i, σ_y[i]), y[i])

            σ_bad_i = sqrt(σ_y[i]^2 + V_bad)
            outliers_log_prob = log(P_bad) + logpdf(Normal(Y_bad, σ_bad_i), y[i])

            # Bayes rule to compute the assignment probability: ℙ(outliers | data) ∝ ℙ(data | outliers) ℙ(outliers)
            log_prob_assign_outliers[chain, step, i] = outliers_log_prob - logaddexp(inliers_log_prob, outliers_log_prob)
        end
    end
end

log_prob_assign_outliers_bis = dropdims(logsumexp(log_prob_assign_outliers; dims=2); dims=2); # shape: N_chains × N
proba_outlier = exp.(log_prob_assign_outliers_bis .- log(N_samples));

# Average across the MCMC chain
proba_outlier_bis = dropdims(mean(proba_outlier; dims=1); dims=1);
```

```julia
p = scatter(x, y, label=nothing, title="data points with probability of being outlier");
for i ∈ 1:N
    proba = round(100 * proba_outlier_bis[i]; digits=2)
    annotate!(p, x[i], y[i], text("$proba %", 8, :bottom))
end
p
```



4 Uncertainties in the best-fit parameters

quick summary

- **Standard χ^2 Uncertainties**:
  - Valid only under strict assumptions (correct uncertainties, no outliers).
  - Often underestimate true uncertainties in real data.
- **Bayesian Posterior**:
  - Uncertainties derived from posterior distribution (e.g., variance, credible intervals).
  - Accounts for model complexity and outliers.
- **Empirical Methods**:
  - **Bootstrap**: Resample data with replacement to estimate parameter distributions.
  - **Jackknife**: Leave-one-out resampling to assess sensitivity to individual points.
  - Both methods useful when uncertainties are unknown or unreliable.

This brief section reiterates how to determine the uncertainties (often represented by a covariance matrix) associated with the best-fit parameters obtained from the fitting process.

It stresses that the validity of these uncertainty estimates depends heavily on the correctness of the initial assumptions about the data's error properties and the appropriateness of the model itself.

4.1 exercise 08

> Compute the standard uncertainty σ_m^2 obtained for the slope of the line found by the standard fit you did in exercise 02. Now make jackknife (20 trials) and bootstrap estimates for the uncertainty σ_m^2. How do the uncertainties compare and which seems most reasonable, given the data and uncertainties on the data?

$$\begin{bmatrix} \sigma_b^2 & \sigma_{mb} \\ \sigma_{mb} & \sigma_m^2 \end{bmatrix} = \left[A^\top C^{-1} A\right]^{-1} =$$

```
inv(transpose(A_ex02) * inv(C_ex02) * A_ex02)
```

```
2×2 Matrix{Float64}:
 207.188    -1.05427
  -1.05427   0.00599181
```

bootstrap (sample with replacement)

```
M = 30; # number of bootstrap samples
b_ex08_bootstrap = zeros(M);
m_ex08_bootstrap = zeros(M);
Threads.@threads for j ∈ 1:M # bootstrap sample
    idx_bs = rand(1:N, N) # bootstrap indices
    Y_bs = reshape(y[idx_bs], :, 1) # vector to 1-column matrix
    A_bs = hcat(ones(N), x[idx_bs])
```

```
    C_bs = Diagonal(σ_y[idx_bs] .^ 2)

    tmp = transpose(A_bs) * inv(C_bs) # intermediary result
    b_ex08_bootstrap[j], m_ex08_bootstrap[j] = inv(tmp * A_bs) * tmp * Y_bs
end
```

$$\sigma^2_{b_{\text{bootstrap}}} = \frac{1}{M}\sum_{j=1}^{M}\left(b_{j_{\text{bootstrap}}} - b_{\text{best-fit}}\right)^2 =$$

```
mean((b_ex08_bootstrap .- b_ex02) .^ 2)
```

```
19755.282590887535
```

$$\sigma^2_{m_{\text{bootstrap}}} = \frac{1}{M}\sum_{j=1}^{M}\left(m_{j_{\text{bootstrap}}} - m_{\text{best-fit}}\right)^2 =$$

```
mean((m_ex08_bootstrap .- m_ex02) .^ 2)
```

```
0.5609441065406167
```

$$\sigma_{bm_{\text{bootstrap}}} = \frac{1}{M}\sum_{j=1}^{M}\left(b_{j_{\text{bootstrap}}} - b_{\text{best-fit}}\right)\left(m_{j_{\text{bootstrap}}} - m_{\text{best-fit}}\right) =$$

```
mean((b_ex08_bootstrap .- b_ex02) .* (m_ex08_bootstrap .- m_ex02))
```

```
-103.63569649977154
```

jackknife (leave-one-out)

```
b_ex08_jackknife = zeros(M);
m_ex08_jackknife = zeros(M);
Threads.@threads for i ∈ 1:N # jackknife sample
    idx_jk = fill(true, N) # jackknife indices
    idx_jk[i] = false # remove j-th point
    Y_jk = reshape(y[idx_jk], :, 1) # vector to 1-column matrix
    A_jk = hcat(ones(N-1), x[idx_jk])
    C_jk = Diagonal(σ_y[idx_jk] .^ 2)

    tmp = transpose(A_jk) * inv(C_jk) # intermediary result
    b_ex08_jackknife[i], m_ex08_jackknife[i] = inv(tmp * A_jk) * tmp * Y_jk
end
```

$$\sigma^2_{b_{\text{jackknife}}} = \left(1 - \frac{1}{N}\right)\sum_{i=1}^{N}\left(b_{i_{\text{jackknife}}} - b_{\text{best-fit}}\right)^2 =$$

```
(1 - 1/N) * sum((b_ex08_jackknife .- b_ex02) .^ 2)
```

```
456153.12364972197
```

$$\sigma^2_{m_{\text{jackknife}}} = \left(1 - \frac{1}{N}\right) \sum_{i=1}^{N} \left(m_{i_{\text{jackknife}}} - m_{\text{best-fit}}\right)^2 =$$

```
(1 - 1/N) * sum((m_ex08_jackknife .- m_ex02) .^ 2)
```

```
11.778148123343657
```

$$\sigma_{bm_{\text{jackknife}}} = \left(1 - \frac{1}{N}\right) \sum_{i=1}^{N} \left(b_{i_{\text{jackknife}}} - b_{\text{best-fit}}\right) \left(m_{i_{\text{jackknife}}} - m_{\text{best-fit}}\right) =$$

```
(1 - 1/N) * sum((b_ex08_jackknife .- b_ex02) .* (m_ex08_jackknife .- m_ex02))
```

```
2048.741485924347
```

4.2 exercise 09

> Re-do exercise 06 — the mixture-based outlier model — but just with the "inlier" points 5 through 20 from the data. Then do the same again, but with all measurement uncertainties reduced by a factor of 2 (uncertainty variances reduced by a factor of 4). Plot the marginalized posterior probability distributions for line parameters (m, b) in both cases.
> Did these posterior distributions get smaller or larger with the reduction in the data-point uncertainties? Compare this with the dependence of the standard uncertainty estimate $\left[A^\top C^{-1} A\right]^{-1}$.

```
data_model_ex09 = model_ex06(N', x', y', σ_y');
chains_ex09 = sample(data_model_ex09, NUTS(), MCMCThreads(), N_samples, N_chains; num_warmup=N_warmup,
    thinning=N_thinning);
println('=' ^ 80)
print_information_criterion(data_model_ex09, chains_ex09)
print_summary(chains_ex09)
```

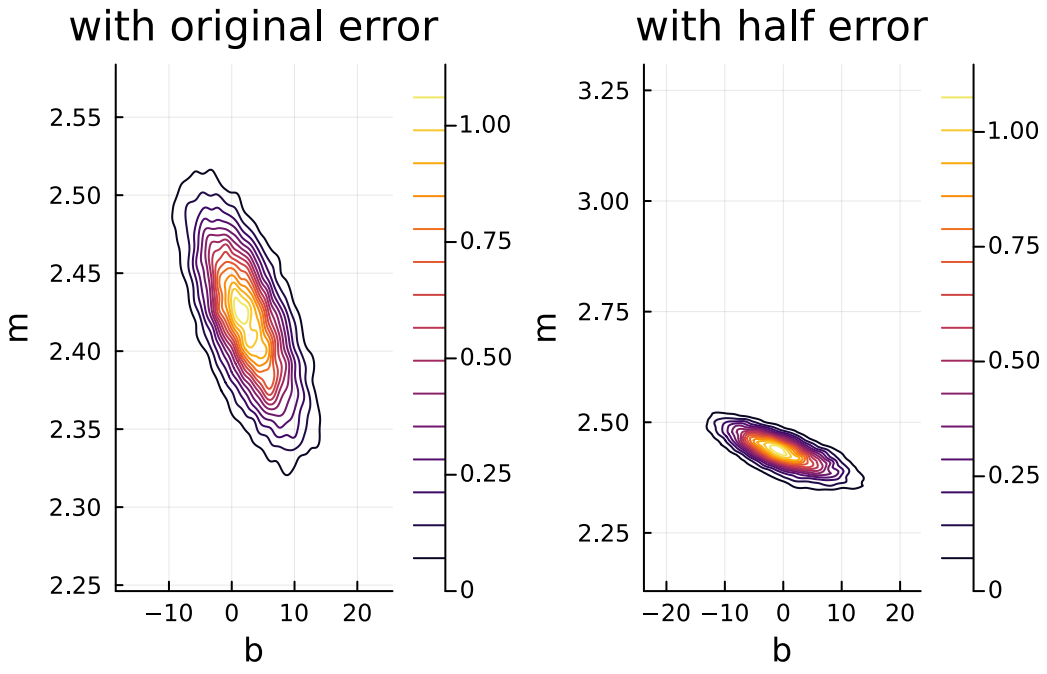
```
┌ Info: Found initial step size
└   ϵ = 0.05
┌ Info: Found initial step size
└   ϵ = 0.2
┌ Info: Found initial step size
└   ϵ = 0.2
┌ Info: Found initial step size
└   ϵ = 0.2
```

```
================================================================================
Deviance Information Criterion: 157.69638955957026
Widely Applicable Information Criterion: 626.5719652376915
Log predictive density: -306.96896785400037
effective number of parameters: 6.317014764845421

Summary Statistics
  parameters       mean        std      mcse      rhat
      Symbol    Float64    Float64   Float64   Float64

           b     2.4314     4.8047    0.0410    1.0000
           m     2.4195     0.0405    0.0003    1.0001
      P_bad     0.0563     0.0550    0.0006    1.0005
      Y_bad   380.9167   119.6706    0.8939    1.0003
      V_bad        Inf        Inf       NaN    1.0002
```

```
data_model_ex09_½err = model_ex06(N′, x′, y′, σ_y′ ./ 2);
chains_ex09_½err = sample(data_model_ex09_½err, NUTS(), MCMCThreads(), N_samples, N_chains;
    num_warmup=N_warmup, thinning=N_thinning);
print_information_criterion(data_model_ex09_½err, chains_ex09_½err)
print_summary(chains_ex09_½err)
```

```
┌ Info: Found initial step size
└   ϵ = 0.036865234375000014
┌ Info: Found initial step size
└   ϵ = 0.05
┌ Info: Found initial step size
└   ϵ = 0.05
┌ Info: Found initial step size
└   ϵ = 0.05
Deviance Information Criterion: 181.33037188830056
Widely Applicable Information Criterion: 718.9223116410654
Log predictive density: -349.0744541335065
effective number of parameters: 10.386701687026314

Summary Statistics
  parameters            mean              std            mcse      rhat
      Symbol         Float64          Float64         Float64   Float64

           b         -0.1162           5.6647          0.0450    1.0000
           m          2.4327           0.0453          0.0006    1.0000
       P_bad          0.4148           0.1514          0.0013    1.0005
       Y_bad        366.5573          54.5760          0.4160    1.0004
       V_bad   24548078.6263   3461841875.5036   24521508.9123    1.0004
```

```
p = plot(layout=2)
plot!(p, kde((flatten_chains(chains_ex09[     :b]), flatten_chains(chains_ex09[     :m]))), subplot=1,
    xlabel="b", ylabel="m", title="with original error")
plot!(p, kde((flatten_chains(chains_ex09_½err[:b]), flatten_chains(chains_ex09_½err[:m]))), subplot=2,
    xlabel="b", ylabel="m", title="with half error")
```



The posterior distributions got much larger with the reduction in uncertainties. They're way underestimated.

The posterior distributions aren't Gaussian anymore, now they're some "islands", which correspond to different choices of outliers

If individual data-points have been estimated by some means that effectively relies on shared information, then there will be large covariances among the data points. These covariances bring off-diagonal elements into the covariance matrix C, which was previously trivially constructed under the assumption that all covariances are precisely 0.

Once the off-diagonal elements are non-zero, χ^2 must be computed by the matrix expression: $\chi^2 = [Y - AX]^\top C^{-1}[Y - AX]$

5 Non-Gaussian uncertainties

no exercise in this section

quick summary: can somehow model as gaussian anyway …

- **Non-Gaussian Uncertainties**:
  - Replace Gaussian likelihood with appropriate distributions (e.g., biexponential, Student's t).
  - Model upper/lower limits by truncating likelihoods (e.g., Poisson for binned data).
- **Avoid Ad-Hoc Methods**:
  - Softening χ^2(e.g., Huber loss) is discouraged without generative justification.
  - Use MCMC to infer parameters of complex likelihoods.

6 Goodness of fit and unknown uncertainties

quick summary

- χ^2 **as a Metric**:
  - Expected value $\chi^2 \approx N - 2$ for a good fit.
  - Large deviations suggest model inadequacy or incorrect uncertainties.
- **Bayesian Perspective**:
  - Cannot reject a model outright; compare models using Bayes factors.
  - Infer unknown uncertainties by marginalizing over them in the posterior.

standard (frequentist) paradigm: χ^2 to measure goodness of fit

with a straight line: χ^2 expect to be in between $[N-2] \pm \sqrt{2[N-2]}$ with 2 is model parameters count (m & b)

6.1 exercise 10

> Assess the χ^2 value for the fit performed in exercise 01 (do that problem first if you haven't already). Is the fit good? What about for the fit performed in exercise 02?

$\chi^2 = [Y - AX]^\top C^{-1} [Y - AX]$

```
tmp = Yex01 - Aex01 * Xex01; # intermediary result
χ²ex01 = dot(tmp, inv(Cex01), tmp);
χ²ex01 / (N′ - 1) # χ² per degrees of freedom
```

```
1.2453846607493875
```

This is a pretty good fit. Aside from the plot showing a linear fit that generally follows the points (whether the points themselves are linear or not), the reduced χ^2 metric is pretty close to 1. This is indicating that on average the squared difference between my model and each point is roughly on par (actually slightly larger) than the measured inherent variance of each point. Can't ask for too much better than that when you're assuming the functional form of a distribution.

```
tmp = Yex02 - Aex02 * Xex02; # intermediary result
χ²ex02 = dot(tmp, inv(Cex02), tmp);
χ²ex02 / (N - 1) # χ² per degrees of freedom
```

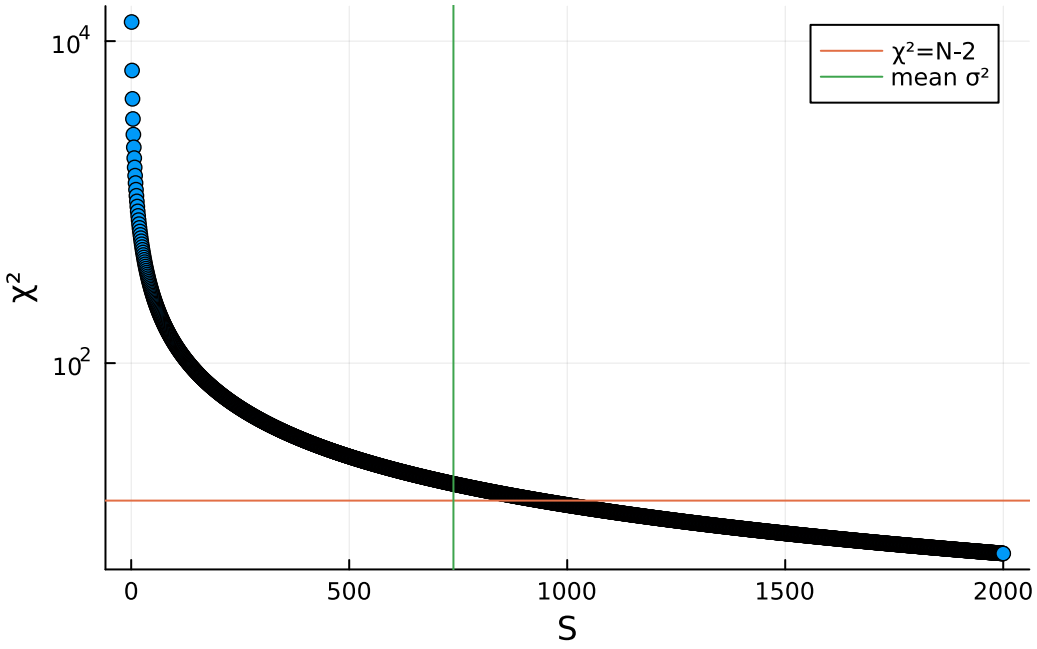
```
15.261248567473642
```

This is a horrid fit. Squared differences 15× larger than what's described by measured variances.

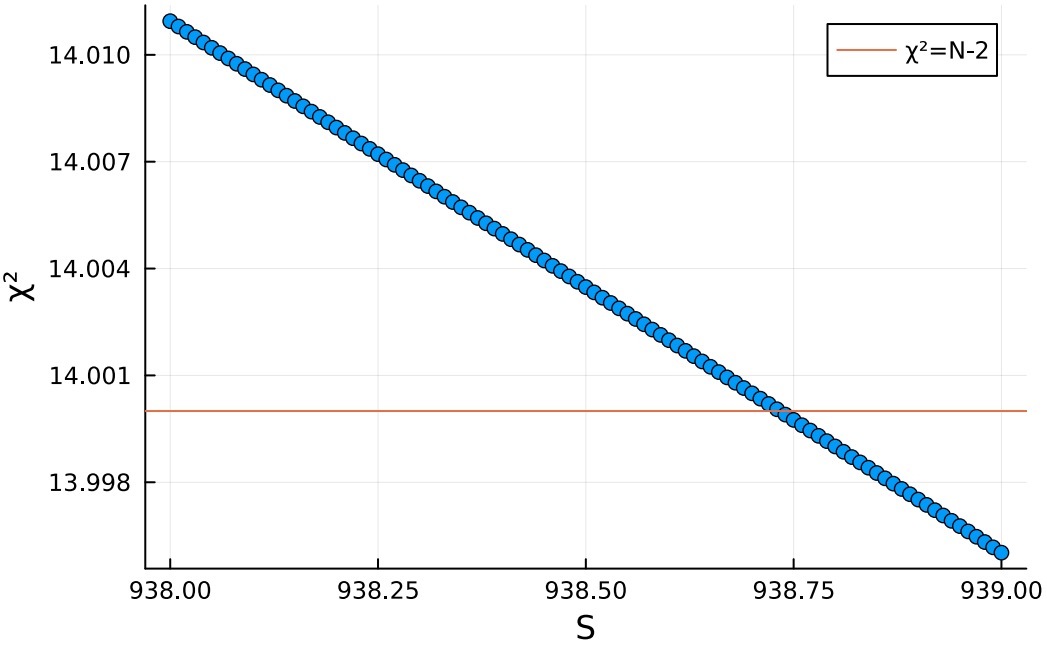
6.2 exercise 11

Re-do the fit of exercise 01 but setting all $\sigma_{y_i}^2 = S$, that is, ignoring the uncertainties and replacing them all with the same value S. What uncertainty variance S would make $\chi^2 = N - 2$? How does it compare to the mean and median of the uncertainty variances $\left(\sigma_{y_i}^2\right)_{i=1}^{N}$?

```
Yex11 = reshape(y′, :, 1); # vector to 1-column matrix
Aex11 = hcat(ones(N′), x′);

Sex11 = 1:2000;
χ²ex11 = map(Sex11) do s
    σ²ex11 = fill(s, N′)
    Cex11 = Diagonal(σ²ex11)
    tmp1 = inv(Cex11)
    tmp2 = transpose(Aex11) * tmp1
    Xex11 = inv(tmp2 * Aex11) * tmp2 * Yex11
    tmp3 = Yex11 - Aex11 * Xex11
    return dot(tmp3, tmp1, tmp3)
end

subsetNm2 = N′ - 2
scatter(Sex11, χ²ex11, yaxis=:log, xlabel="S", ylabel="χ²", label=nothing)
hline!([subsetNm2], label="χ²=N-2")
vline!([mean(σ_y′ .^ 2)], label="mean σ²")
```



value S would be between: 938 and 939



mean variances $\left(\sigma_{y_i}^2\right)_{i=1}^N$ of the true data set: 739.0625

The uniform offset S necessary to make $\chi^2 = N - 2$ is quite larger than both the mean and median variances $\left(\sigma_{y_i}^2\right)_{i=1}^N$ of the true data set. If the data were in fact linear with some measurement uncertainties, perhaps the uncertainties themselves are underestimated.

6.3 exercise 12

> Flesh out and write all equations for the Bayesian uncertainty estimation and marginalization procedure described in this section. Note that the inference and marginalization would be very expensive without excellent sampling tools! Make the additional (unjustified) assumption that all the uncertainties have the same variance $\sigma_{y_i}^2 = S$ to make the problem tractable.
> Apply the method to the x and y values for points 5 through 20 in the data. Make a plot showing the points, the maximum a posteriori value of the uncertainty variance as error bars, and the maximum a posteriori straight line.
> For extra credit, plot 2 straight lines, one that is maximum a posteriori for the full posterior and one that is the same but for the posterior after the uncertainty variance S has been marginalized out. Also plot 2 sets of error bars, one that shows the maximum for the full posterior and one for the posterior after the line parameters (m, b) have been marginalized out.

Gaussian likelihood:

$$\mathcal{L} = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi \mathrm{S}}} \exp\left(-\frac{(y_i - mx_i - b)^2}{2\mathrm{S}}\right)$$

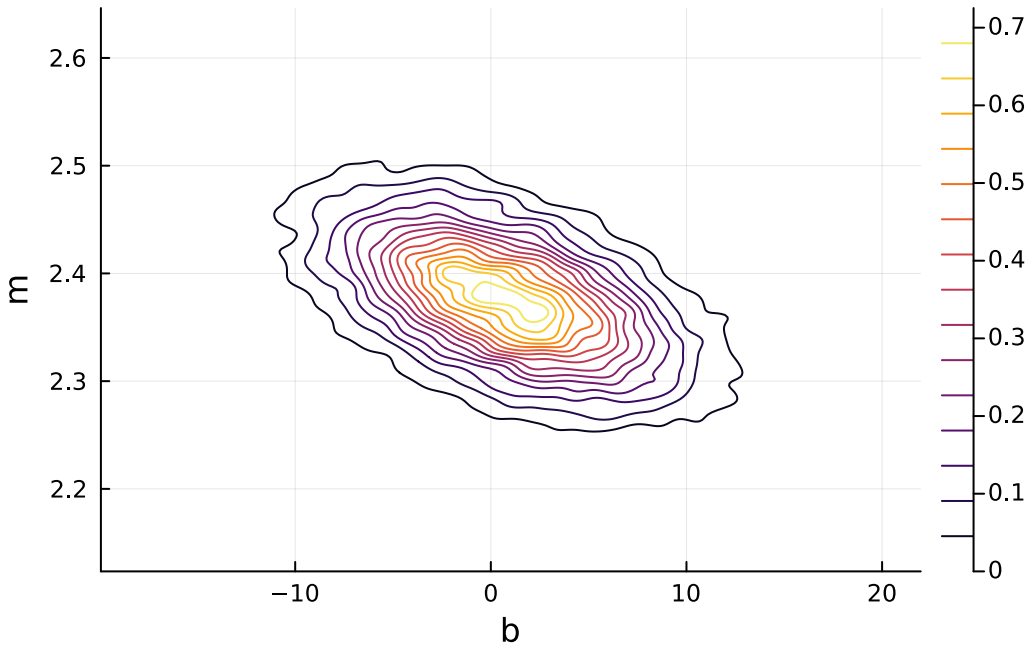
```
@model function model_ex12(N, x, y)
    b ~ Normal(0, 5)
    m ~ Normal(0, 5)
    S ~ InverseGamma(.001, .001) # common variance for all data

    for i ∈ 1:N
        y[i] ~ Normal(b + m * x[i], sqrt(S))
    end
end

data_model_ex12 = model_ex12(N', x', y')
chains_ex12 = sample(data_model_ex12, NUTS(), MCMCThreads(), N_samples, N_chains; num_warmup=N_warmup,
↪   thinning=N_thinning);
```

```
13 println('=' ^ 80)
14 print_information_criterion(data_model_ex12, chains_ex12)
15 print_summary(chains_ex12)
```

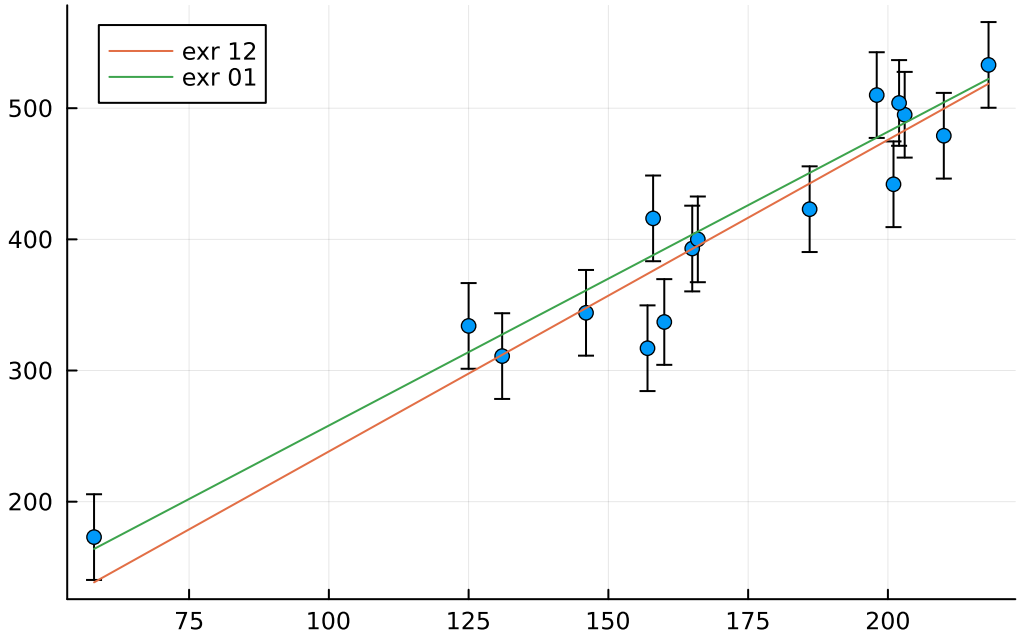
```
┌ Info: Found initial step size
└   ϵ = 0.00078125
┌ Info: Found initial step size
└   ϵ = 0.0015625
┌ Info: Found initial step size
└   ϵ = 0.00078125
┌ Info: Found initial step size
└   ϵ = 0.00078125
```

```
Deviance Information Criterion: 158.01676347170306
Widely Applicable Information Criterion: 629.2810782107024
Log predictive density: -310.03761918999885
effective number of parameters: 4.602919915352328

Summary Statistics
  parameters        mean        std      mcse      rhat
      Symbol     Float64    Float64   Float64   Float64

           b      0.6802     4.9422    0.0348    1.0001
           m      2.3760     0.0548    0.0004    1.0000
           S   1067.2937   459.6777    3.2433    0.9999
```

```
1 plot(kde((flatten_chains(chains_ex12[:b]), flatten_chains(chains_ex12[:m]))), xlabel="b", ylabel="m")
```




```
1 b_ex12 = mean(chains_ex12[:b]);
2 m_ex12 = mean(chains_ex12[:m]);
3 S_ex12 = mean(chains_ex12[:S]);
4 scatter(x′, y′, yerror=fill(sqrt(S_ex12), N′), label=nothing)
5 plot!(𝕩 -> m_ex12*𝕩 + b_ex12, label="exr 12")
6 plot!(𝕩 -> m_ex01*𝕩 + b_ex01, label="exr 01")
```


exr 12
exr 01
500
400
300
200
75
100
125
150
175
200

7 Arbitrary 2-dimensional uncertainties

quick summary

- **Generative Model for 2D Uncertainties**:
  - Extend to account for uncertainties in x and y directions.
  - Project uncertainties onto the line's direction to compute likelihood.
- **Comparison to PCA**:
  - PCA is invalid for noisy data; generative model accounts for measurement errors.

more complex scenario of fitting a line when both the x and y measurements have uncertainties: using a covariance tensor for each data point to represent the uncertainties in x, the uncertainties in y, and any correlation between the x and y errors

covariance tensor $S_i = \begin{bmatrix} \sigma_{x_i}^2 & \sigma_{xy_i} \\ \sigma_{xy_i} & \sigma_{y_i}^2 \end{bmatrix}$ with $\sigma_{xy} = \rho_{xy}\sigma_x\sigma_y$

likelihood

$$\mathcal{L} = \prod_{i=1}^{N} \frac{1}{2\pi\sqrt{\det(S_i)}} \exp\left(-\frac{1}{2}\left[Z_i - Z\right]^\top S_i^{-1}\left[Z_i - Z\right]\right)$$

with

$$Z_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad ; \quad Z = \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \end{bmatrix} = \begin{bmatrix} \hat{x}_i \\ m\hat{x}_i + b \end{bmatrix}$$

instead of slope m, use unit vector $\vec{v}$ orthogonal to the line (*i.e.* normal vector)

$$\vec{v} = \frac{1}{\sqrt{1+m^2}} \begin{bmatrix} -m \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}$$

with $\theta = \arctan m$ angle made between the line and the x axis

orthogonal displacement Δ_i of each data point (x_i, y_i) from the line: $\Delta_i = \vec{v}^\top Z_i - b\cos\theta$

each data point's covariance tensor S_i projects down to an orthogonal variance $\Sigma_i^2 = \vec{v}^\top S_i \vec{v}$

then the log likelihood can be written as

$$\log\mathcal{L} = K - \frac{1}{2}\sum_{i=1}^{N}\left[\frac{\Delta_i^2}{\Sigma_i^2} + \log|\Sigma_i^2| + \log\left(1+m^2\right)\right]$$

where K is some constant

the authors suggest: performing the fit or likelihood maximization not in terms of (m, b) but rather $(\theta, b_\perp)$, where $b_\perp = b\cos\theta$ the perpendicular distance of the line from the origin

> In the astrophysics literature, there is a tradition, when there are uncertainties in both directions, of fitting the "forward" and "reverse" relations — that is, fitting y as a function of x and then x as a function of y — and then splitting the difference between the 2 slopes so obtained, or treating the difference between the slopes as a systematic uncertainty. This is unjustified.

Another common method for finding the linear relationship in data when there are uncertainties in both directions is principal components analysis. The method of PCA does return a linear relationship for a data set, in the form of the dominant principal component. However, this is the dominant principal component of the observed data, not of the underlying linear relationship that, when noise is added, generates the observations. For this reason, the output of PCA will be strongly drawn or affected by the individual data point noise covariances S_i.

example of mixture model with outlier

$$Z_{\text{bad}} = \begin{bmatrix} X_{\text{bad}} \\ Y_{\text{bad}} \end{bmatrix} \quad ; \quad V_{\text{bad}} = \begin{bmatrix} V_{\text{bad}_x} & \rho_{\text{bad}}\sqrt{V_{\text{bad}_x} V_{\text{bad}_y}} \\ \rho_{\text{bad}}\sqrt{V_{\text{bad}_x} V_{\text{bad}_y}} & V_{\text{bad}_y} \end{bmatrix}$$

$$\mathcal{L} \propto \prod_{i=1}^{N} \left[\frac{1 - P_{\text{bad}}}{\sqrt{2\pi\Sigma_i^2}} \cos(\theta) \exp\left(-\frac{\Delta_i^2}{2\Sigma_i^2} \right) + \frac{P_{\text{bad}}}{\sqrt{2\pi \det\left(V_{\text{bad}} + S_i\right)}} \exp\left(-\frac{1}{2} \left[Z_i - Z_{\text{bad}}\right]^\top \left[V_{\text{bad}} + S_i\right]^{-1} \left[Z_i - Z_{\text{bad}}\right] \right) \right]$$

7.1 exercise 13

> Using the method of this section, fit the straight line $y = m\,x + b$ to the x, y, σ_x^2, σ_{xy}, and σ_y^2 values of points 5 through 20 taken from the data. Make a plot showing the points, their 2-dimensional uncertainties (show them as 1 ellipses), and the best-fit line.

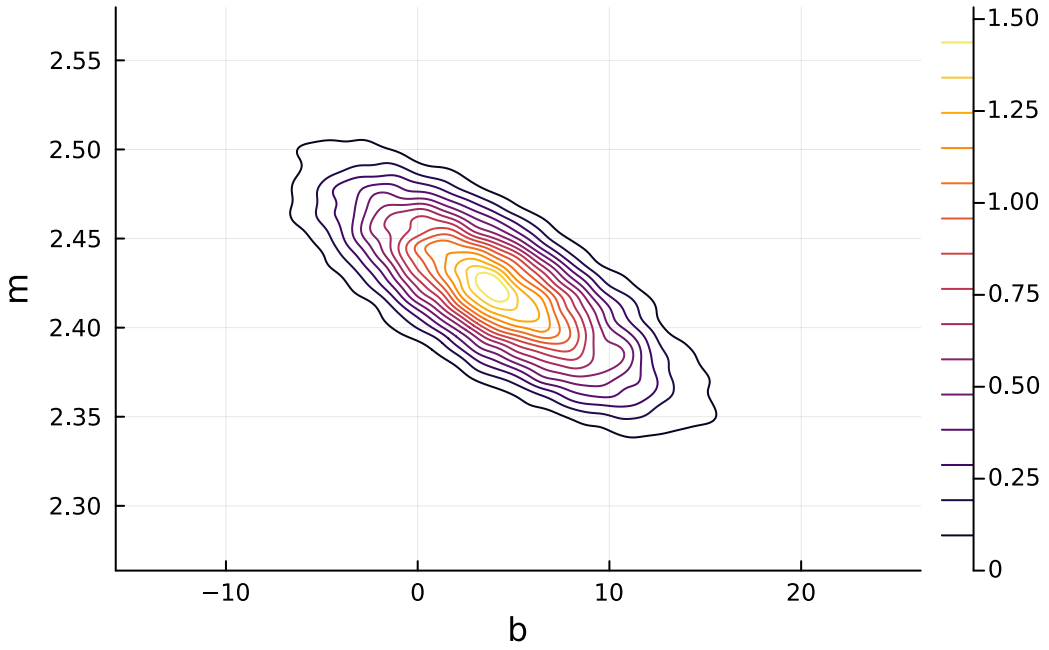
```julia
# DRAFT (SHOULDN'T USE): manually modify log likehood
@model function modelₑₓ₁₃_draft(ℕ, ℤ, 𝕊)
    b ~ Normal(0, 5) # intercept
    θ ~ Uniform(-angle90, angle90) # angle of the fitted line, use this instead of slope
    v⃗ = [-sin(θ), cos(θ)] # unit normal vector
    m := tan(θ) # `:=` operator add variable to the chain

    for i ∈ 1:ℕ
        Δᵢ = dot(ℤ[i], v⃗) - b*v⃗[2] # orthogonal displacement of each data point from the line
        Σᵢ² = dot(v⃗, 𝕊[i], v⃗) # orthogonal variance of projection of each data point to the line
        Turing.@addlogprob! -.5*(Δᵢ^2 / Σᵢ² + log(abs(Σᵢ²)) + log1p(m^2))
    end
end
```

```julia
# recommendation: multivariate normal distribution
@model function modelₑₓ₁₃(ℕ, ℤ, 𝕊)
    b ~ Normal(0, 5)
    m ~ Normal(0, 5)

    for i ∈ 1:ℕ
        xᵢ = ℤ[i][1]
        ℤ̂ᵢ = [xᵢ, b + m * xᵢ]
        ℤ[i] ~ MvNormal(ℤ̂ᵢ, 𝕊[i])
    end
end

data_modelₑₓ₁₃ = modelₑₓ₁₃(N′, Z′, S′);
chainsₑₓ₁₃ = sample(data_modelₑₓ₁₃, NUTS(), MCMCThreads(), N_samples, N_chains; num_warmup=N_warmup,
    thinning=N_thinning);
println('=' ^ 80)
print_information_criterion(data_modelₑₓ₁₃, chainsₑₓ₁₃)
print_summary(chainsₑₓ₁₃)
```

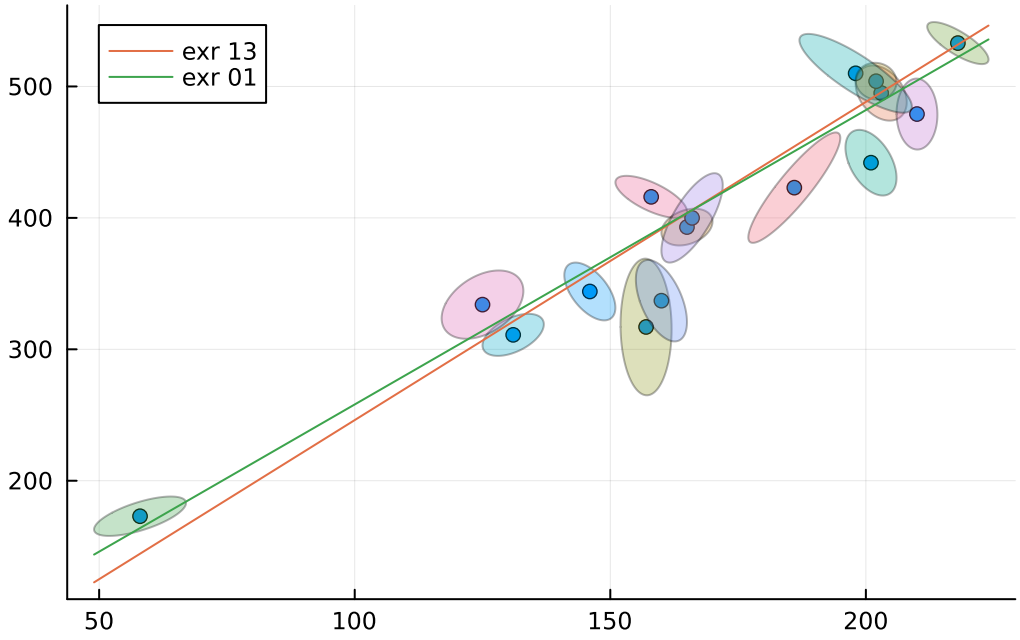
```
┌ Info: Found initial step size
└   ϵ = 0.025
┌ Info: Found initial step size
└   ϵ = 0.025
┌ Info: Found initial step size
└   ϵ = 0.025
┌ Info: Found initial step size
└   ϵ = 0.025
```

```
Deviance Information Criterion: 247.00888431179794
Widely Applicable Information Criterion: 984.7350833469969
Log predictive density: -488.041225247589
effective number of parameters: 4.326316425909415

Summary Statistics
  parameters      mean       std      mcse      rhat
      Symbol   Float64   Float64   Float64   Float64

           b    4.0385    4.7008    0.0353    1.0001
           m    2.4210    0.0355    0.0003    1.0001
```

```
plot(kde((flatten_chains(chains_ex13[:b]), flatten_chains(chains_ex13[:m]))), xlabel="b", ylabel="m")
```




```
b_ex13 = mean(chains_ex13[:b]);
m_ex13 = mean(chains_ex13[:m]);

plot_ellipses(N′, x′, y′, Z′, S′)
plot!(𝕩 -> m_ex13*𝕩 + b_ex13, label="exr 13")
plot!(𝕩 -> m_ex01*𝕩 + b_ex01, label="exr 01")
```



7.2 exercise 14

Repeat exercise 13, but using all of the data. Some of the points are now outliers, so your fit may look worse. Follow the fit by a robust procedure analogous to the Bayesian mixture model with bad-data probability P_{bad} described in section 3. Use something sensible for the prior probability distribution for (m, b).
Plot the 2 results with the data and uncertainties.
For extra credit, plot a sampling of 10 lines drawn from the marginalized posterior distribution for (m, b) and plot the samples as a set of light grey or transparent lines. For extra extra credit, mark each data point on your plot with the fully marginalized probability that the point is bad (that is, rejected, or has $q = 0$).

```
# DRAFT (SHOULDN'T USE): manually modify log likehood
@model function modelₑₓ₁₄_draft(ℕ, ℤ, 𝕊)
    b ~ Normal(0, 5) # intercept
    θ ~ Uniform(-angle90, angle90) # angle of the fitted line, use this instead of slope
    v⃗ = [-sin(θ), cos(θ)] # unit normal vector
    m := tan(θ) # `:=` operator add variable to the chain

    # for outliers
    P_bad ~ Uniform(0, 1)
    X_bad ~ Uniform(x_min, x_max)
    Y_bad ~ Uniform(y_min, y_max)
    σ_bad_x ~ InverseGamma(.001, .001)
    σ_bad_y ~ InverseGamma(.001, .001)
    ρ_bad ~ Uniform(-1, 1)

    Z_bad = [X_bad, Y_bad]
    V_bad_xy = ρ_bad * σ_bad_x * σ_bad_y
    V_bad = [σ_bad_x^2 V_bad_xy; V_bad_xy σ_bad_y^2]

    for i ∈ 1:ℕ
        Δᵢ = dot(ℤ[i], v⃗) - b*v⃗[2] # orthogonal displacement of each data point from the line
        Σᵢ² = dot(v⃗, 𝕊[i], v⃗) # orthogonal variance of projection of each data point to the line
        tmp1 = ℤ[i] - Z_bad
        tmp2 = 𝕊[i] + V_bad
        inlier = log1p(-P_bad) - .5*log(abs(Σᵢ²)) + log(abs(v⃗[2])) - .5 * Δᵢ^2 / Σᵢ²
        outlier = log(P_bad) - .5 * (logabsdet(tmp2)[1] + dot(tmp1, tmp2, tmp1))
```

```
        Turing.@addlogprob! logaddexp(inlier, outlier)
    end
end
```

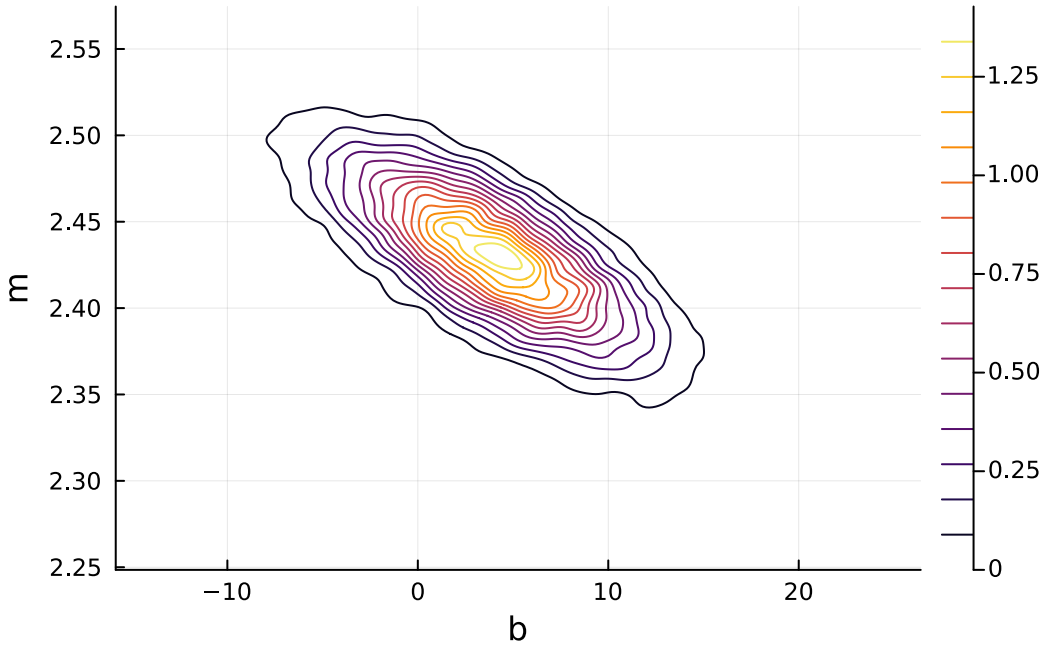
```
@model function model_ex14(ℕ, ℤ, 𝕊)
    b ~ Normal(0, 5)
    m ~ Normal(0, 5)

    # for outliers
    P_bad ~ Uniform(0, 1)
    X_bad ~ Uniform(x_min, x_max)
    Y_bad ~ Uniform(y_min, y_max)
    σ_bad_x ~ InverseGamma(.001, .001)
    σ_bad_y ~ InverseGamma(.001, .001)
    ρ_bad ~ Uniform(-1, 1)

    Z_bad = [X_bad, Y_bad]
    V_bad_xy = ρ_bad * σ_bad_x * σ_bad_y
    V_bad = [σ_bad_x^2 V_bad_xy; V_bad_xy σ_bad_y^2]

    # maybe more computationally efficient if use LKJ Cholesky for correlation matrix
    # see: https://discourse.julialang.org/t/singular-exception-with-lkjcholesky/85020/25

    for i ∈ 1:ℕ
        Σ_badᵢ = 𝕊[i] + V_bad
        if !isposdef(Σ_badᵢ) # sometimes isn't positive-definitive
            Turing.@addlogprob! -Inf # force sampler to reject a sample
            break
        else
            xᵢ = ℤ[i][1]
            ẑᵢ = [xᵢ, b + m * xᵢ]
            ℤ[i] ~ MixtureModel(MvNormal, [(ẑᵢ, 𝕊[i]), (Z_bad, Σ_badᵢ)], [1 - P_bad, P_bad])
        end
    end
end

data_model_ex14 = model_ex14(N, Z, S);
chains_ex14 = sample(data_model_ex14, NUTS(), MCMCThreads(), N_samples, N_chains; num_warmup=N_warmup,
↪  thinning=N_thinning);
println('=' ^ 80)
print_information_criterion(data_model_ex14, chains_ex14)
print_summary(chains_ex14)
```

```
┌ Info: Found initial step size
└   ϵ = 0.025
┌ Info: Found initial step size
└   ϵ = 0.05
┌ Info: Found initial step size
└   ϵ = 0.05
┌ Info: Found initial step size
└   ϵ = 0.025
```

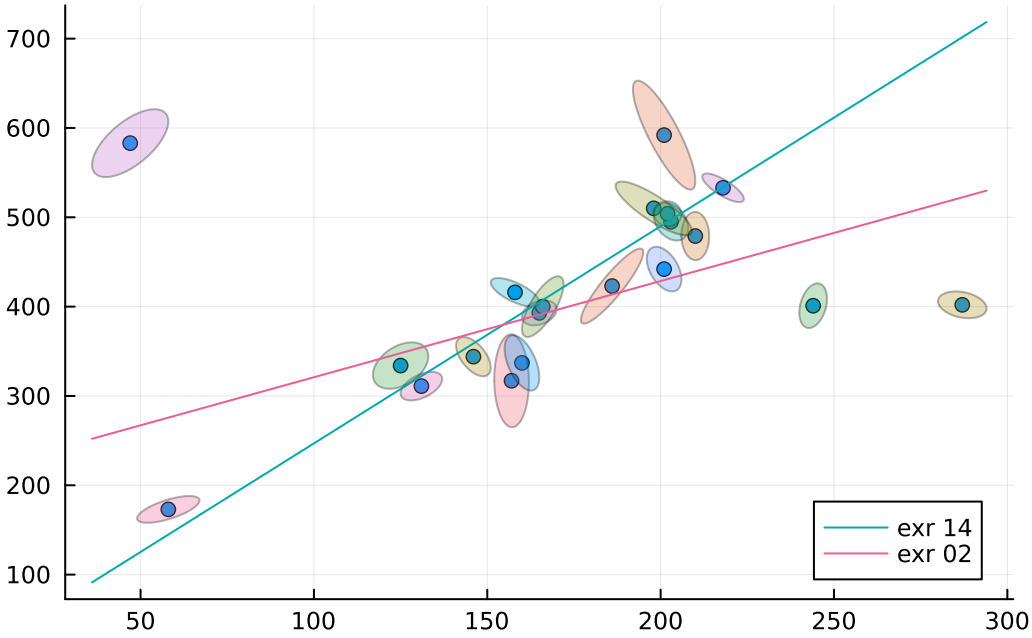
```
Deviance Information Criterion: 369.79616331784376
Widely Applicable Information Criterion: 1463.3422277068032
Log predictive density: -709.1683608352579
effective number of parameters: 22.502753018143757

Summary Statistics
```

```
parameters       mean       std      mcse      rhat
    Symbol    Float64   Float64   Float64   Float64

         b     3.7896    4.7017    0.0377    1.0000
         m     2.4312    0.0359    0.0003    1.0002
     P_bad     0.1942    0.0861    0.0007    1.0000
     X_bad   185.3930   52.9724    0.4222    1.0001
     Y_bad   457.8542   56.7519    0.4168    1.0002
   σ_bad_x   141.0835   96.3230    0.7048    0.9999
   σ_bad_y   111.8610   87.5325    0.7000    1.0001
     ρ_bad    -0.5655    0.3885    0.0036    1.0002
```

```
plot(kde((flatten_chains(chains_ex14[:b]), flatten_chains(chains_ex14[:m]))), xlabel="b", ylabel="m")
```

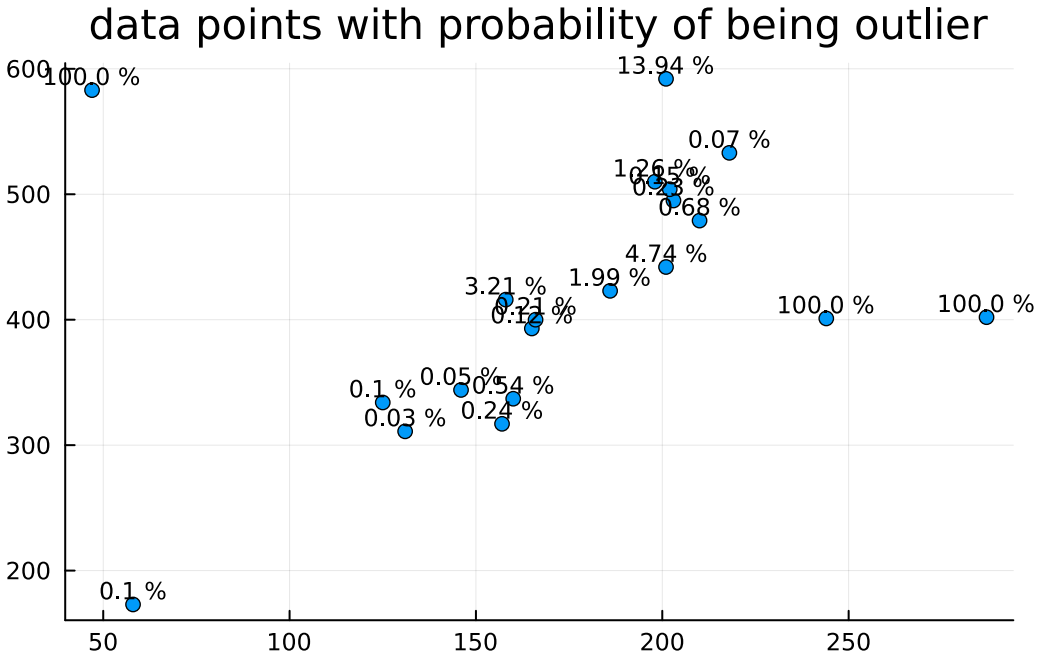



```
b_ex14 = mean(chains_ex14[:b]);
m_ex14 = mean(chains_ex14[:m]);

plot_ellipses(N, x, y, Z, S)
plot!(𝕩 -> m_ex14*𝕩 + b_ex14, label="exr 14")
plot!(𝕩 -> m_ex02*𝕩 + b_ex02, label="exr 02")
```



little extra

compute probability of each point to be classified as outlier:

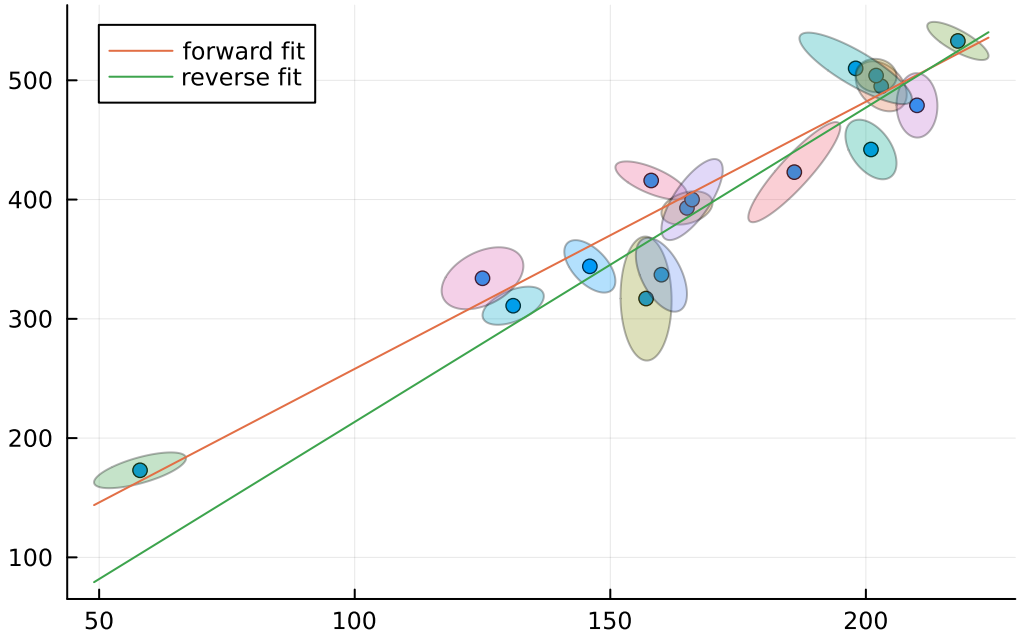


7.3 exercise 15

Perform the abominable forward-reverse fitting procedure on points 5 through 20 from the data.
That is, fit a straight line to the y values of the points, using the y-direction uncertainties σ_y^2 only, by the standard method described in section 1.
Now transpose the problem and fit the same data but fitting the x values using the x-direction uncertainties σ_x^2 only.
Make a plot showing the data points, the x-direction and y-direction uncertainties, and the 2 best-fit lines.
Comment.

```
Yex15 = reshape(x′, :, 1); # vector to 1-column matrix
Aex15 = hcat(ones(N′), y′);
Cex15 = Diagonal(σ_x′ .^ 2);
tmp = transpose(Aex15) * inv(Cex15); # intermediary result
bex15, mex15 = inv(tmp * Aex15) * tmp * Yex15;
```

```
plot_ellipses(N′, x′, y′, Z′, S′)
plot!(𝕩 -> mex01*𝕩 + bex01, label="forward fit")
plot!(𝕩 -> (𝕩 - bex15) / mex15, label="reverse fit")
```



- forward fit: slope = 2.24 and intercept = 34.05
- reverse fit: slope = 2.64 and intercept = -49.94

7.4 exercise 16

Perform principal components analysis on points 5 through 20 from the data. That is, diagonalize the 2×2 matrix Q given by $Q = \sum_{i=1}^{N} \left[Z_i - \bar{Z}\right] \left[Z_i - \bar{Z}\right]^\top$ with $\bar{Z} = \frac{1}{N} \sum_{i=1}^{N} Z_i$. Find the eigenvector of Q with the largest eigenvalue.
Now make a plot showing the data points, and the line that goes through the mean $\bar{Z}$ of the data with the slope corresponding to the direction of the principal eigenvector. Comment.

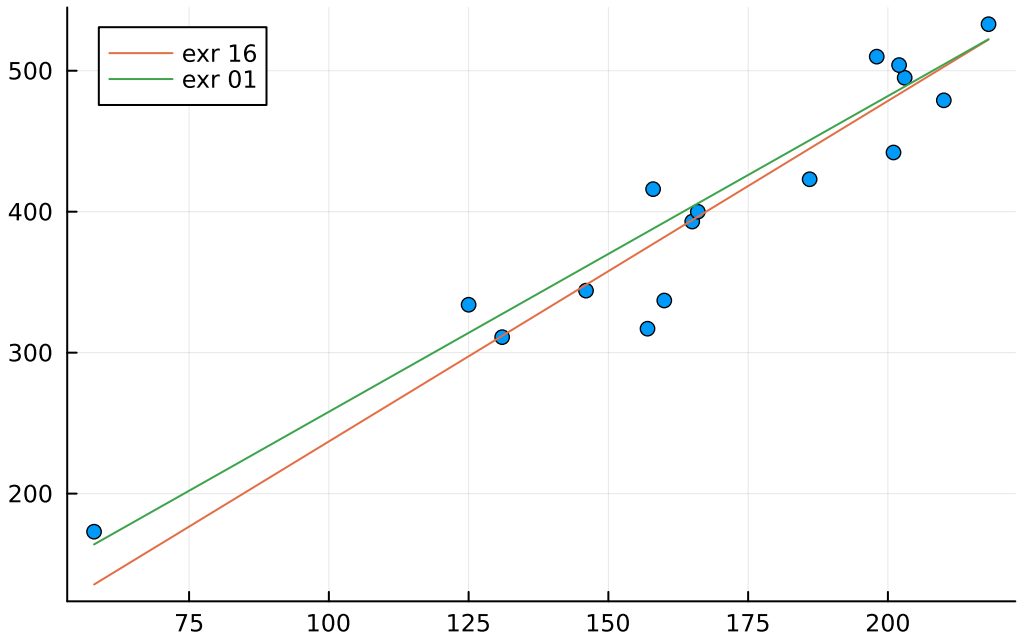
```
Z̄′ = [mean(x′), mean(y′)];
Z′_centered = [Z′[i] - Z̄′ for i ∈ 1:N′];
Q = sum([z*transpose(z) for z ∈ Z′_centered])
```

```
2×2 Matrix{Float64}:
 25057.0  55542.8
 55542.8      1.36261e5
```

```
_, eigvecs = eigen(Q);
eigvecs_max = eigvecs[:, end] # direction vector of 1st principal component
```

```
2-element Vector{Float64}:
 0.3824362551881647
 0.923981877916257
```

```
m_ex16 = eigvecs_max[end] / eigvecs_max[begin];
b_ex16 = Z̄'[2] - m_ex16 * Z̄'[1]; # all principal components go through the means

scatter(x', y', label=nothing)
plot!(𝕩 -> m_ex16*𝕩 + b_ex16, label="exr 16")
plot!(𝕩 -> m_ex01*𝕩 + b_ex01, label="exr 01")
```



slope = 2.42 and intercept = -4.6

8 Intrinsic scatter

quick summary

- **Intrinsic Scatter**:
  - Add a parameter $ V $ to model intrinsic variance (e.g., perpendicular to the line).
  - Model scatter perpendicular to the line, marginalizing over true positions.
  - Avoids ad-hoc "scatter subtraction" and provides principled uncertainty estimates.

introduce an intrinsic Gaussian variance V, orthogonal to the line

the log likelihood can be written as

$$\log \mathcal{L} = K - \frac{1}{2} \sum_{i=1}^{N} \left[\frac{\Delta_i^2}{\Sigma_i^2 + V} + \log |\Sigma_i^2 + V| + \log (1 + m^2) \right]$$

where K is some constant

limitations: it models only the distribution orthogonal to the relationship

8.1 exercise 17

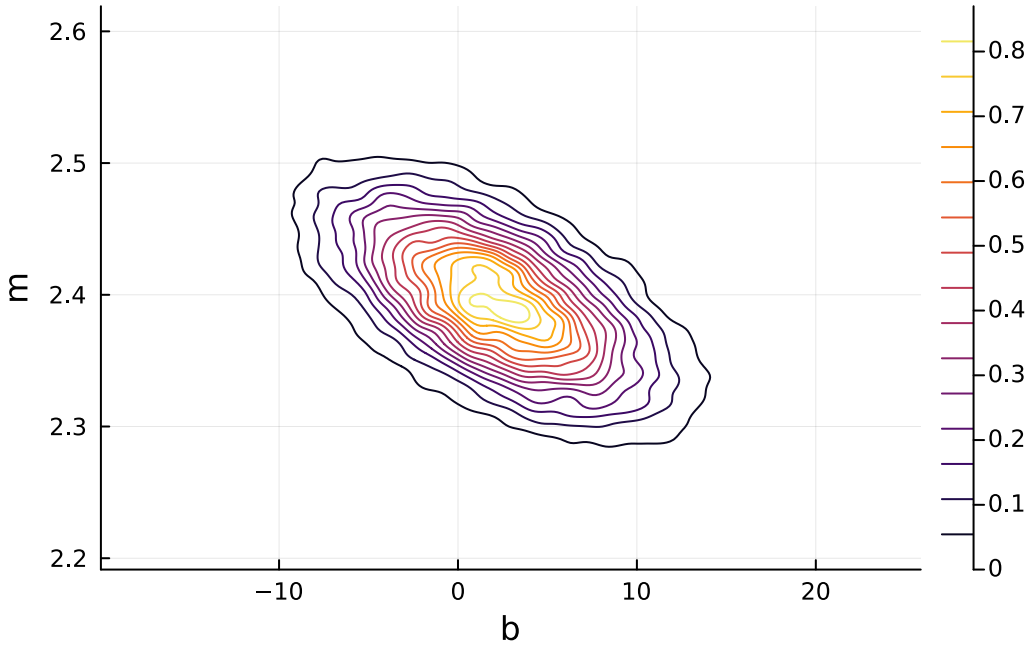
> Re-do exercise 13, but now allowing for an orthogonal intrinsic Gaussian variance V and only excluding data point 3. Re-make the plot, showing not the best-fit line but rather the $\pm\sqrt{V}$ lines for the maximum-likelihood intrinsic relation.

```
@model function model_ex17(ℕ, ℤ, 𝕊)
    b ~ Normal(0, 5)
    θ ~ Uniform(-angle90, angle90) # angle of the fitted line, use this instead of slope
    v⃗ = [-sin(θ), cos(θ)] # unit normal vector
    m := tan(θ) # `:=` operator add variable to the chain
    V ~ InverseGamma(.001, .001) # intrinsic Gaussian variance orthogonal to the line

    for i ∈ 1:ℕ
        Δᵢ = dot(ℤ[i], v⃗) - b*v⃗[2] # orthogonal displacement of each data point from the line
        Σᵢ² = dot(v⃗, 𝕊[i], v⃗) # orthogonal variance of projection of each data point to the line
        tmp = Σᵢ² + V # intermediary result
        Turing.@addlogprob! -.5*(Δᵢ^2 / tmp + log(abs(tmp)) + log1p(m^2))
    end
end

data_model_ex17 = model_ex17(N′, Z′, S′);
chains_ex17 = sample(data_model_ex17, NUTS(), MCMCThreads(), N_samples, N_chains; num_warmup=N_warmup,
↪   thinning=N_thinning);
println('=' ^ 80)
print_information_criterion(data_model_ex17, chains_ex17)
print_summary(chains_ex17)
```

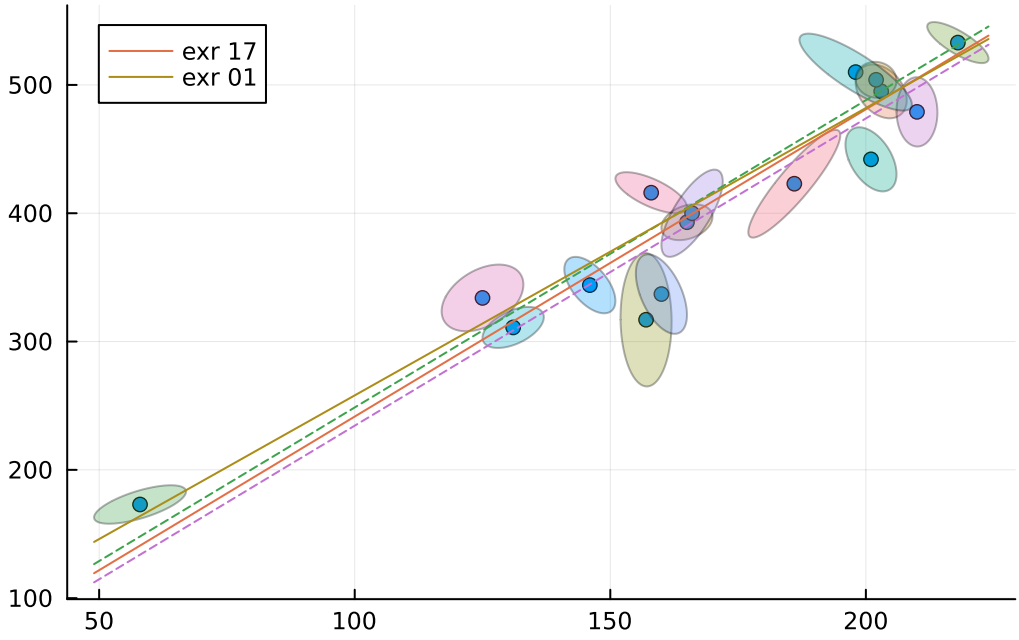
```
┌ Info: Found initial step size
└   ϵ = 0.025
┌ Info: Found initial step size
└   ϵ = 0.025
┌ Info: Found initial step size
└   ϵ = 0.025
┌ Info: Found initial step size
└   ϵ = 0.025
```

```
Deviance Information Criterion: 123.47737305379421
Widely Applicable Information Criterion: 491.75357604976875
Log predictive density: -242.6821637024135
effective number of parameters: 3.194624322470837

Summary Statistics
  parameters      mean       std      mcse      rhat
      Symbol   Float64   Float64   Float64   Float64

           b    2.0692    4.9119    0.0354    1.0001
           θ    1.1751    0.0070    0.0001    1.0002
           m    2.3946    0.0474    0.0003    1.0002
           V    7.5046   20.5960    0.1584    1.0000
```

```
plot(kde((flatten_chains(chains_ex17[:b]), flatten_chains(chains_ex17[:m]))), xlabel="b", ylabel="m")
```



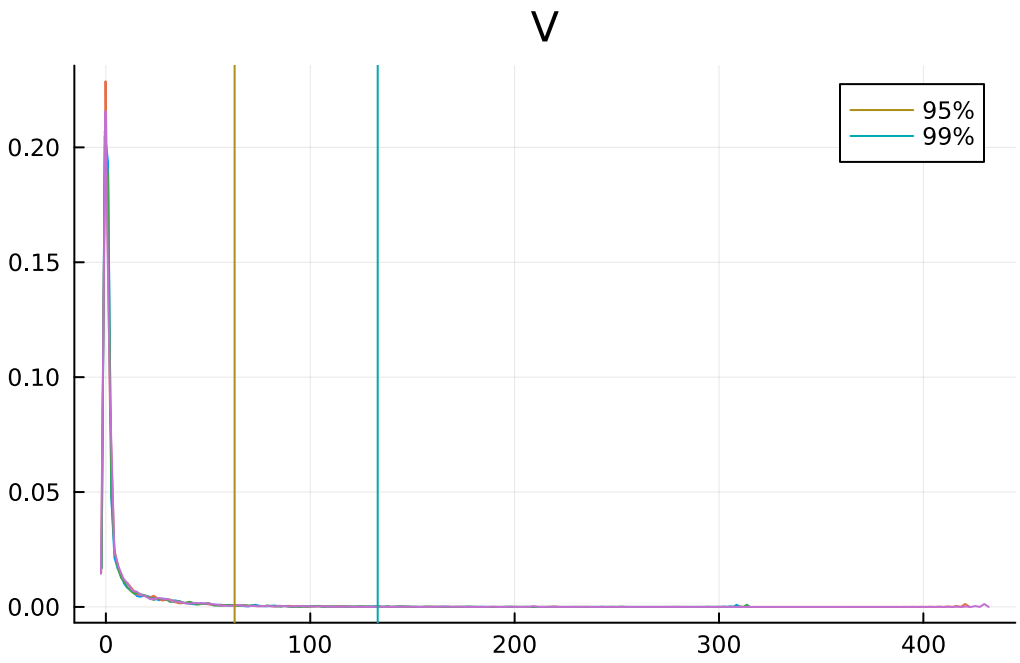

```
b_ex17 = mean(chains_ex17[:b]);
m_ex17 = mean(chains_ex17[:m]);
move_up = sqrt(mean(chains_ex17[:V])) / cos(mean(chains_ex17[:θ]));

plot_ellipses(N′, x′, y′, Z′, S′)
plot!(𝕩 -> m_ex17*𝕩 + b_ex17, label="exr 17")
plot!(𝕩 -> m_ex17*𝕩 + b_ex17 + move_up, linestyle=:dash, label=nothing)
plot!(𝕩 -> m_ex17*𝕩 + b_ex17 - move_up, linestyle=:dash, label=nothing)
plot!(𝕩 -> m_ex01*𝕩 + b_ex01, label="exr 01")
```



8.2 exercise 18

Re-do exercise 17 but as a Bayesian, with sensible Bayesian priors on $(\theta, b_\perp, V)$. Find and marginalize the posterior distribution over $(\theta, b_\perp)$ to generate a marginalized posterior probability for the intrinsic variance parameter V. Plot this posterior with the 95% and 99% upper limits on V marked. Why did we ask only for upper limits?

```
# confidence level α → 2 quantiles: 1/2 ± α/2
level9x = quantile(chains_ex17[:V].data, [.5+.95/2 .5+.99/2]) # use matrix not vector for vline() later
density(chains_ex17[:V], title="V", label=nothing)
vline!(level9x, label=["95%" "99%"])
```



the posterior is very skewed, so the upper limit is much more informative than the lower limit which isn't very informative because it's very close to 0.