Project 1

Problem 1

A higher-order fit. We did a linear fit ($y_{pred} = b + mx$)

- Try extending the code to do a quadratic fit: $y_{pred} = b + mx + qx^2$
- Does the fit improve? Repeat the jack-knife analysis (and plot up the answers produced by each run of the jack-knife), and investigate the scatter in the " q " values. -Can you conclude anything about whether you need the " q " term for a good fit?

Solution:

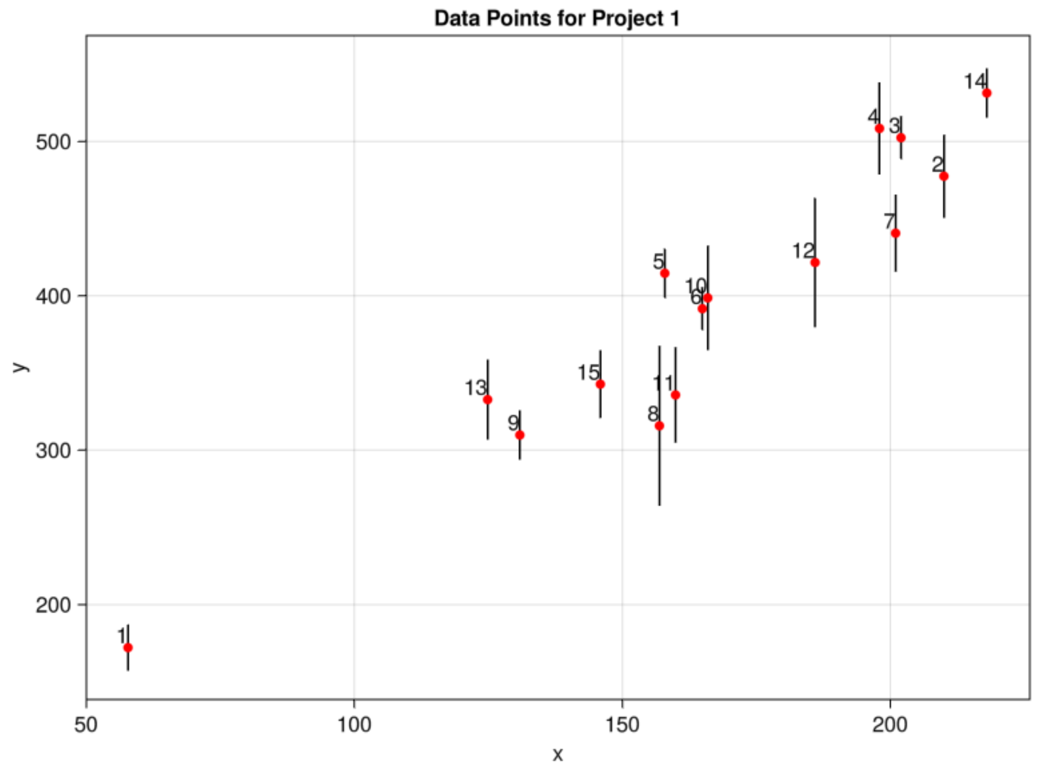
In [1]:
```
using CSV
using DataFrames
using WGLMakie
using Optim
```

In [2]:
```
alldata = CSV.read("data.csv", DataFrame);
data = alldata[6:end, :];
```

Visualization of cropped data

In [3]:
```
f = Figure()
Axis(f[1, 1],xlabel="x", ylabel="y", title="Data Points for Project 1")
errorbars!(data.x, data.y, data.sigma_y)
scatter!(data.x, data.y, markersize=10, color=:red)
for i in 1:length(data.x)
    text!(data.x[i], data.y[i], text = string(i), align = (:right, :bottom))
end
f
```

Out[3]:



Objective function for Quadractic Fit

This function takes the input $(x, y \ \& \ \sigma_y)$ tuple as input in `args` variable and initial parameters in `parameters`. It computes the log-likelihood function as follows:

$$-\sum_{i=0}^{n}\left(-log(2\pi\sigma_i) - \frac{1}{2}\frac{(y_i - mx_i - b)^2}{\sigma_i^2}\right)$$

In [5]:
```
function objective_quadratic(parameters, args)
    (x, y, sigma) = args
    b = parameters[1]
    m = parameters[2]
    q = parameters[3]
    y_pred = b .+ m .* x + q .* x .* x
    return -sum(-log.(sigma * sqrt(2 * π)) .-0.5 .* (y .- y_pred).^2 / sigma.^2)
end;
```

In [21]:
```
m = 2.0; b = 10.0; q = 1.0;
starting_params = [b, m, q]
arguments = (data.x, data.y, data.sigma_y)
result = optimize(p -> objective_quadratic(p, arguments), starting_params)
@assert Optim.converged(result)
bq_abs, mq_abs, q_abs = Optim.minimizer(result)
```

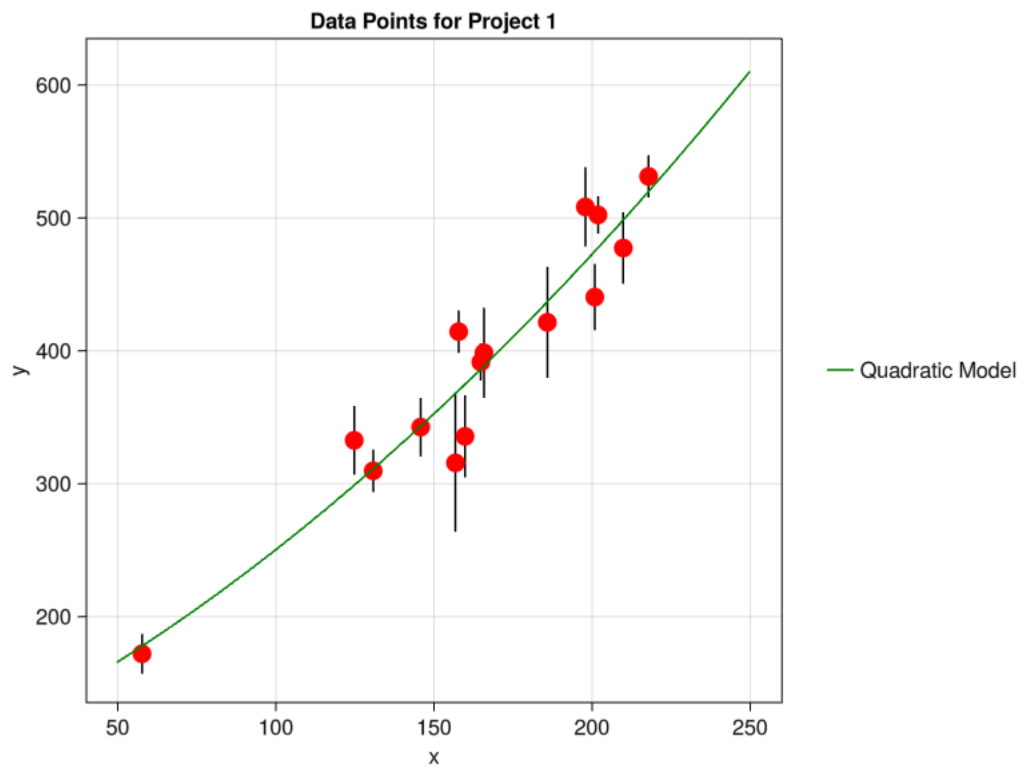
Out[21]:
```
3-element Vector{Float64}:
 99.75559490606929
  1.1551829282654742
  0.0035791194886326235
```

Plot of the estimated quadratic model

In [22]:
```
f = Figure()
Axis(f[1, 1],xlabel="x", ylabel="y",title="Data Points for Project 1")
errorbars!(data.x, data.y, data.sigma_y)
```

```
scatter!(data.x, data.y, markersize=20, color=:red)
xx = LinRange(50, 250, 25);
yy_abs = xx .* xx .* q_abs .+ xx .* mq_abs .+ bq_abs
lin = lines!(xx, yy_abs, color=:green)
Legend(f[1, 2], [lin], ["Quadratic Model"], framevisible = false)
f
```

Out[22]:



Jack-Knife function defintion

In [8]:

```
function jack_knife!(data_x, data_y, data_sigma_y, obj_func, initial_pos, final_re
    """
    This is a simple implemetation of *jack-knife* process for curve fitting.

    * x: x axis data for some data set (list/array)
    * y: y axis data for the same data set (list/array)
    * sigma_y: uncertainty of y axis data (list/array)
    * obj_func: a function that computes the error or cost for particular function
               It will get called like this:
           cost = obj_func(parameters, obj_args)
    * starting_params: initial values for the optimizer (list/array)
    * final_result: it the container for the output of the parameter values (b,m,q

    Returns final_result

    Usage:

    ```

    ndata = size(data,1)
    final_result1 = zeros(ndata,3)
    initial_parameter = [b, m, q]
    final_result1 = jack_knife!(data.x, data.y, data.sigma_y, objective_quadratic,
    (B_jack, M_jack, Q_jack) = (final_result1[:,1], final_result1[:,2], final_resu

    ```

    """
    ndata = size(data_x,1)
```

```
    nparam = size(initial_pos,1)
    final_result = zeros(ndata,nparam)
    for i in 1:ndata

        xcopy = copy(data_x)
        deleteat!(xcopy, i)

        ycopy = copy(data_y)
        deleteat!(ycopy, i)

        scopy = copy(data_sigma_y)
        deleteat!(scopy, i)

        result = optimize(p -> obj_func(p, (xcopy, ycopy, scopy)), starting_params
        @assert Optim.converged(result)
        optim_val = Optim.minimizer(result)
        for j in 1:nparam
            final_result[i,j] = optim_val[j] # write it in one line
        end
    end
    return final_result
end;
```

In [9]:

```
ndata = size(data,1)
final_result1 = zeros(ndata,3)
final_result1 = jack_knife!(data.x, data.y, data.sigma_y, objective_quadratic,[b, 
(B_jack, M_jack, Q_jack) = (final_result1[:,1], final_result1[:,2], final_result1[
```

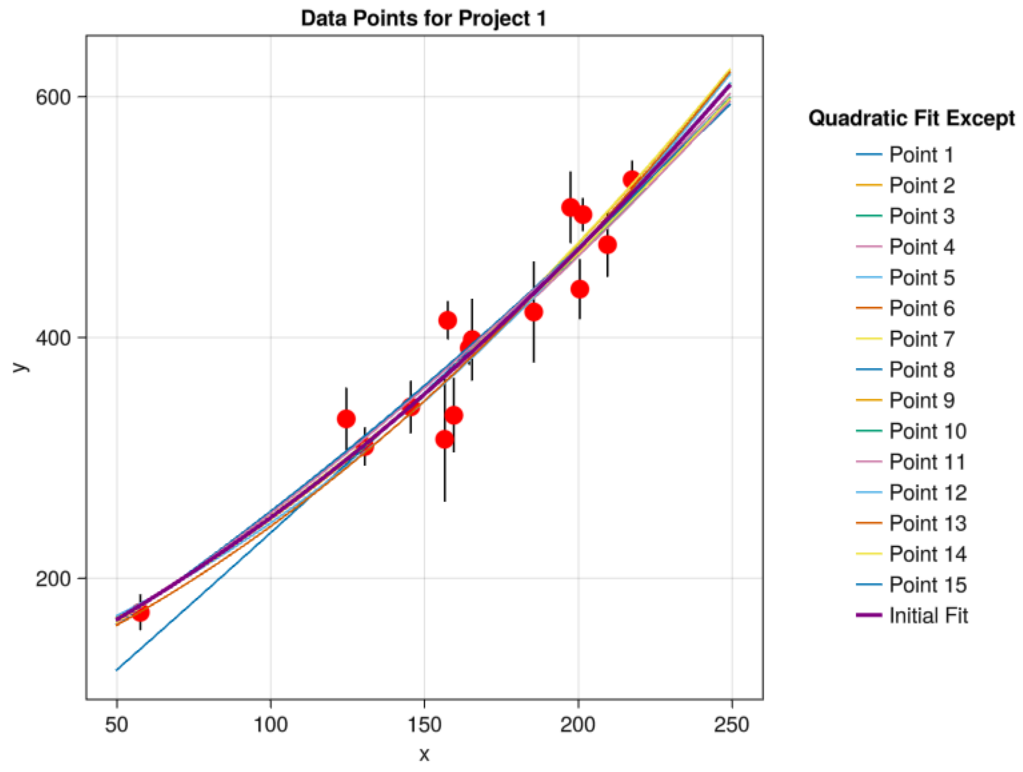
In [10]:

```
f = Figure()
ax = Axis(f[1, 1],xlabel="x", ylabel="y", title="Data Points for Project 1")
errorbars!(data.x, data.y, data.sigma_y)
scatter!(data.x, data.y, markersize=20, color=:red)

for i in 1:ndata
    yy_jack = xx .* xx .* Q_jack[i] .+ xx .* M_jack[i] .+ B_jack[i]
    lines!(xx, yy_jack,label = "Point " * string(i) )
end

lines!(xx, yy_abs, color=:purple, linewidth=3, label = "Initial Fit")
f[1, 2] = Legend(f, ax, "Quadratic Fit Except", framevisible = false)
f
```

Out[10]:

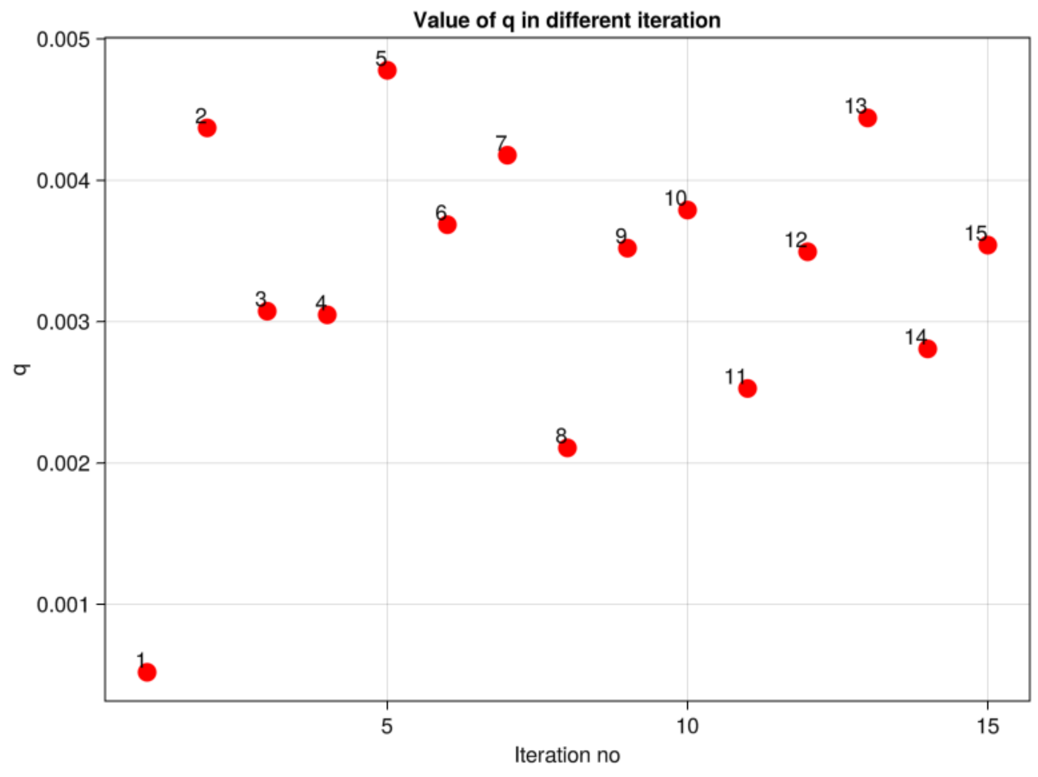


Plot of q values for each jack-knife process

In [12]:

```
f = Figure()
Axis(f[1, 1], ylabel = "q", xlabel = "Iteration no", title = "Value of q in differe
datt = LinRange(1.0,ndata,ndata)
scatter!(datt, Q_jack, markersize=20, color=:red)
for i in 1:length(data.x)
    text!(datt[i], Q_jack[i], text = string(i), align = (:right, :bottom))
end
f
```

Out[12]:



From the scale of the plot we can clearly see that because of the presence of point 1 in data, q values is little higher. However, the value of q is close to zero compared to other parameters. Therefore, the parameter q might not be needed here.

- **Stretch 1.** We were using (negative of) the log-likelihood for our objective function to optimize. Part of that log-likelihood term is called the "χ-squared" term (or "sum of χ-squared"), and a change in χ-squared ("δ-χsq") is often used as a quick way of talking about how much a fit is improved by adding a parameter. Read up about χ-squared and describe how much adding a quadratic term improved the fit, in terms of δ-χ-squared. Finally, look up the Akaike Information Criterion, which is a numerical implementation of Occam's Razor that is supposed to tell you whether you are justified in adding the quadratic parameter to your model. Does the AIC improve when you go from the linear to the quadratic model? Optionally, look up the Bayesian Information Criterion (just a different formula for doing the same job); what does it say?

Solution:

For this problem description χ-squared objective function would look like the following:

$$\chi^2(m,b) = \sum_{i=0}^{n} \frac{(y_i - mx_i - b)^2}{\sigma_i^2}$$

Objective Function for Linear fit

In [13]:
```
function objective_linear(parameters, args)
    (x, y, sigma) = args
    b = parameters[1]
    m = parameters[2]
    y_pred = b .+ m .* x
    return -sum(-log.(sigma * sqrt(2 * π)) .-0.5 .* (y .- y_pred).^2 / sigma.^2)
end;
```

In [14]:
```
starting_params = [b, m]
arguments = (data.x, data.y, data.sigma_y)
result = optimize(p -> objective_linear(p, arguments), starting_params)
@assert Optim.converged(result)
b_abs, m_abs = Optim.minimizer(result)
```

Out[14]:
```
2-element Vector{Float64}:
 32.00184588666802
  2.1910407280118998
```

Now we can look at the χ-squared value for linear and quardatic fit using our objective functions. The log-likelihood function is as follows.

$$L(m,b) = -\sum_{i=0}^{n} \left(-log(2\pi\sigma_i) - \frac{1}{2}\frac{(y_i - mx_i - b)^2}{\sigma_i^2} \right)$$

We can see clearly that using χ-squared definition we can obtain the following:

$$L(m,b) = -\sum_{i=0}^{n} (-log(2\pi\sigma_i)) - \frac{1}{2}\chi^2(m,b)$$

This implies:

$$\chi^2(m,b) = 2\sum_{i=0}^{n} log(2\pi\sigma_i) - 2L(m,b)$$

Now we compute the $2\sum_{i=0}^{n} log(2\pi\sigma_i)$ as a new variable ζ and use our already defined varible to find the value of $\chi^2(m,b)$.

In [24]:
```
ζ = 2 *sum( log.(2π .* data.sigma_y))
chi_squared_linear =  2*ζ - 2* objective_linear([b_abs, m_abs], arguments)
chi_squared_quadratic = 2*ζ -2*objective_quadratic([bq_abs, mq_abs, q_abs], argume
@show chi_squared_linear, chi_squared_quadratic;
```

```
(chi_squared_linear, chi_squared_quadratic) = (-1540.402402369019, -1539.689952421
4396)
```

It is very clear that the quadratic fit improves the χ-squared value very tiny amount. Now we can apply Akaike information criterion for this case. The formula for this is

$$AIC = 2k - 2ln(\hat{L})$$

where k is the number of estimated parameters in the model and $\hat{L}$ is the maximum value of log-likelihood function of the model.

In [31]:
```
AIC_linear = 2 * 2 - 2 *log(objective_linear([b_abs, m_abs], arguments))
AIC_quadratic = 2 * 3 - 2 *log(objective_quadratic([bq_abs, mq_abs, q_abs], argume
@show AIC_linear, AIC_quadratic;
```

```
(AIC_linear, AIC_quadratic) = (-9.648255965578139, -7.647481223140014)
```

We know that if the value of AIC is minimum the model is better. Therefore, the linear fit model is better that the quadratic fit model. Next we can put the same test using Bayesian Information Criterion. The key formula here is

$$BIC = k\ ln(n) - 2ln(\hat{L})$$

where n is the number of data point in x and k and $\hat{L}$ is same as for AIC.

In [33]:
```
BIC_linear =  2 * log(ndata)- 2 *log(objective_linear([b_abs, m_abs], arguments))
BIC_quadratic = 3 * log(ndata) - 2 *log(objective_quadratic([bq_abs, mq_abs, q_abs
@show BIC_linear, BIC_quadratic;
```

```
(BIC_linear, BIC_quadratic) = (-8.23215556337372, -5.523330619833384)
```

We get the same conclusion from the BIC , since linear fit gives the lowest value. Hence we can say that linear fit is better for this data set using these information theoretic criteria.

Problem 2

Investigate the m, b plane. In class we plotted up our solutions in the m, b coordinate plane. Here, I'd like you to investigate a bit more. Try making a contour plots of the (Gaussian) objective function (in terms of m, b), but giving it only a single data point. (I think you should find that there is a diagonal locus of m, b values that have a maximum log-likelihood (so minimum objective-function value), because those lines go right through that single data point. Now try the same thing with a different data point. (Plot both contours on the same plot. Then also try plotting the contour for the two data points combined.) Now try all data points. You can make a contour plot with code like this -- in this case, just the first data point because I'm pulling out elements [1:1]:

```
# Range of m,b values to plot
bvals = LinRange(0., 100., 100)
mvals = LinRange(2.0, 2.5, 100);
# Compute the objective function for each point in a grid
og = [objective_gauss([b,m], data.x[1:1], data.y[1:1],
data.sigma_y[1:1])
      for b in bvals, m in mvals]
f = Figure()
Axis(f[1, 1])
contour!(bvals, mvals, og, levels=20)
f
```

I've found this list of Makie plots to be a handy reference: https://docs.makie.org/stable/examples/plotting_functions/

Next, try plotting the contours for our outlier-rejecting version of the objective function. Try expanding the range of the m and b parameters and see if you can find the "punk" minimum we found in class. Are there other minima you can see also?

Solution:

In [34]:
```
ndata = size(data,1)
final_result = zeros(ndata,2)
final_result = jack_knife!(data.x, data.y, data.sigma_y, objective_linear,[b, m],
(B_jack, M_jack) = (final_result[:,1], final_result[:,2]);
```

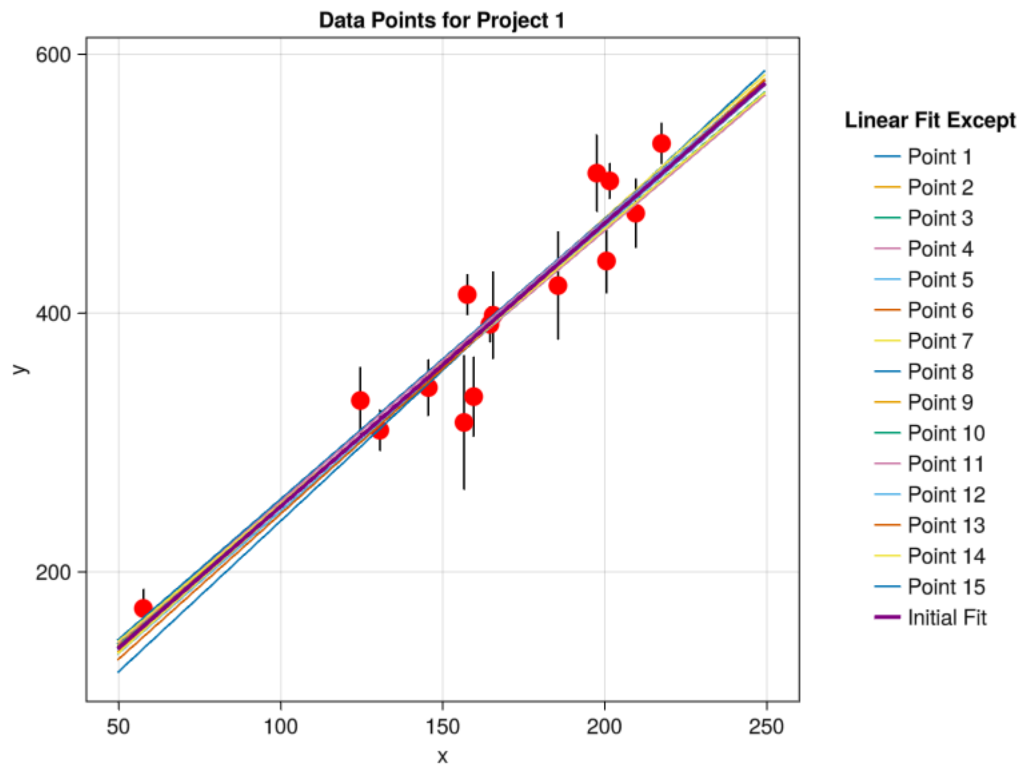
In [35]:
```
f = Figure()
ax = Axis(f[1, 1],xlabel="x", ylabel="y", title="Data Points for Project 1")
errorbars!(data.x, data.y, data.sigma_y)
scatter!(data.x, data.y, markersize=20, color=:red)

for i in 1:ndata
    yy_jack = xx .* M_jack[i] .+ B_jack[i]
    lines!(xx, yy_jack,label = "Point " * string(i) )
end

yy_abs = xx .* m_abs .+ b_abs

lines!(xx, yy_abs, color=:purple, linewidth=3, label = "Initial Fit")
f[1, 2] = Legend(f, ax, "Linear Fit Except", framevisible = false)
f
```

Out[35]:



In [36]:

```
function objective_gauss(parameters, x, y, sigma)
    b = parameters[1]
    m = parameters[2]
    y_pred = b .+ m .* x
    return -sum(-log.(sigma * sqrt(2 * π)) .-0.5 .* (y .- y_pred).^2 / sigma.^2)
end;
```

Function to plot contour for different set of data points

In [37]:

```
function cplots(x,y,sigma, obj_func, ranges)
    """
    This function does the contour plot in parameter space (m vs b)
    and find the minimum point in the terrain and display it in the plot

    * x: x axis data for some data set (list/array)
    * y: y axis data for the same data set (list/array)
    * sigma: uncertainty of y axis data (list/array)
    * obj_func: a function that computes the error or cost for particular function
                It will get called like this:
        cost = obj_func(parameters, obj_args)
    * ranges: it is a tuple that contains the range of the contour like this (xmin

    Returns

    * f: the handle for the plot
    * val: the minimum value within the contour plot range
    * bvals: the value for intercept at the minimum value within the contour plot r
    * mvals: the value for slope at the minimum value within the contour plot range

    Usuage:

    ```

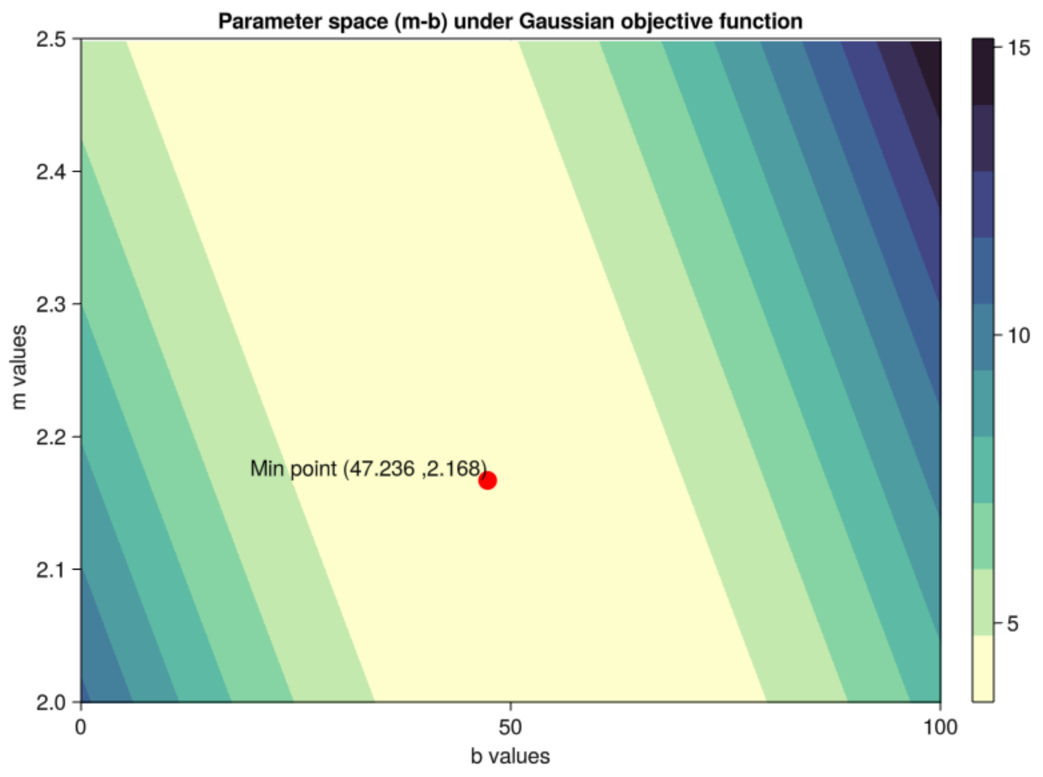
    ranges = (0,100,2,2.5)
    f, val, bvalue, mvalue = cplots(data.x,data.y,data.sigma_y,objective_gauss,rang
    ```
```

```
    """
    # Range of m,b values to plot
    f = Figure()
    (xmin, xmax, ymin, ymax) = ranges
    Axis(f[1, 1], xlabel = " b values ", ylabel = " m values",
        title = "Parameter space (m-b) under Gaussian objective function")
    bvals = LinRange(xmin, xmax, 200)
    mvals = LinRange(ymin, ymax, 200);
    # Compute the objective function for each point in a grid
    og = [obj_func([b,m], x, y, sigma)
          for b in bvals, m in mvals]
    val, arg = findmin(og)
    co = contourf!(bvals, mvals, og, colormap = :deep)
    Colorbar(f[1, 2], co)
    scatter!( bvals[arg[1]], mvals[arg[2]], markersize=20, color=:red)
    text!(bvals[arg[1]], mvals[arg[2]],
        text = "Min point (" * string(round(bvals[arg[1]],digits=3))*
        " ," * string(round(mvals[arg[2]],digits=3))* ")" , align = (:right, :bott
    return f , val, bvals[arg[1]], mvals[arg[2]]
end;
```

Contour plot for single data point

In [38]:
```
ranges = (0,100,2,2.5)
f, val, bvalue, mvalue = cplots(data.x[1:1],data.y[1:1],data.sigma_y[1:1],objectiv
f
```

Out[38]:



- As expected we got totally parallel contour, because we are only using one data point as the input for the objective function.

Contour plot with randomly selected two data points

- At first we randomly select two different points from the data set

In [39]:
```
indices = [];
first_index = rand(1:ndata,1);
second_index = rand(1:ndata,1);
while second_index == first_index
    second_index = rand(1:ndata,1)';
end
indices = hcat(first_index, second_index)
@show indices;
```

```
indices = [11 12]
```

- Next we find the slope and intercept of the straight line that goes through the middle of two points.

$$y_1 = mx_1 + b$$
$$y_2 = mx_2 + b$$

This gives us,

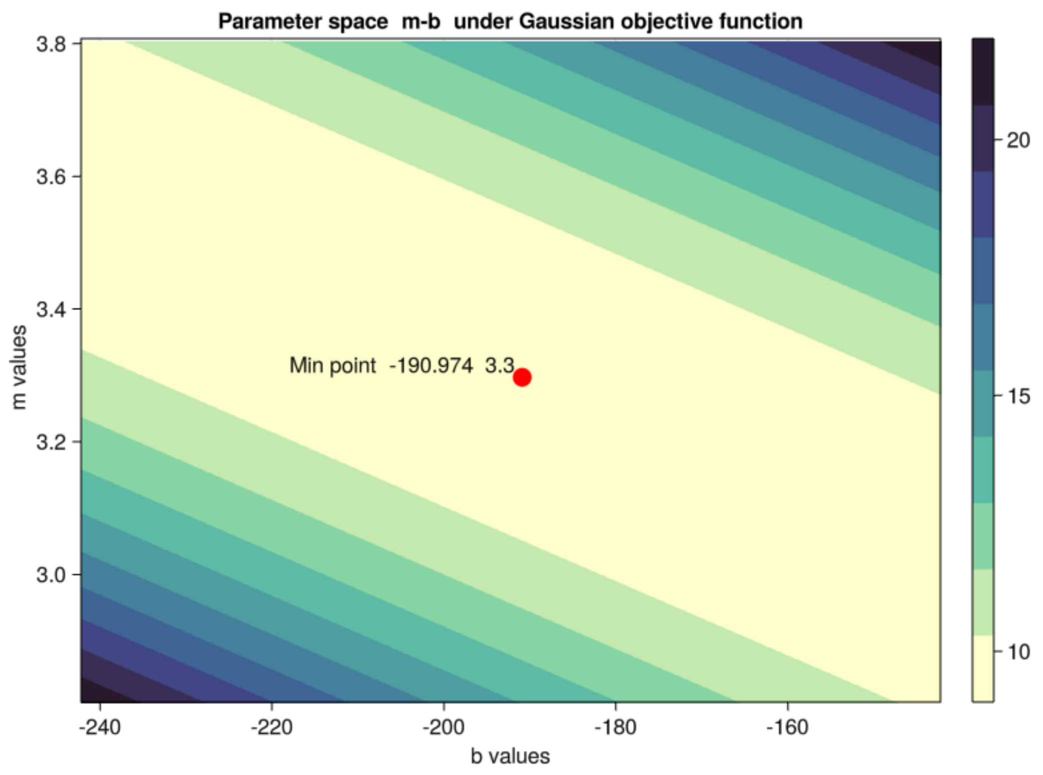
$$m = \frac{y_1 - y_2}{x_1 - x_2}$$
$$b = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1}$$

In [40]:
```
m_actual  = (data.y[indices][2]-data.y[indices][1])/(data.x[indices][2]-data.x[ind
b_actual  = (data.y[indices][1]*data.x[indices][2]-data.y[indices][2]*data.x[indic
    -data.x[indices][1])
@show m_actual, b_actual;
```

```
(m_actual, b_actual) = (3.3076923076923075, -192.23076923076923)
```

In [41]:
```
ranges = (b_actual-50,b_actual+50,m_actual-0.5,m_actual+0.5)
f, val, bvalue, mvalue = cplots(data.x[indices],data.y[indices],data.sigma_y[indic
f
```

Out[41]:



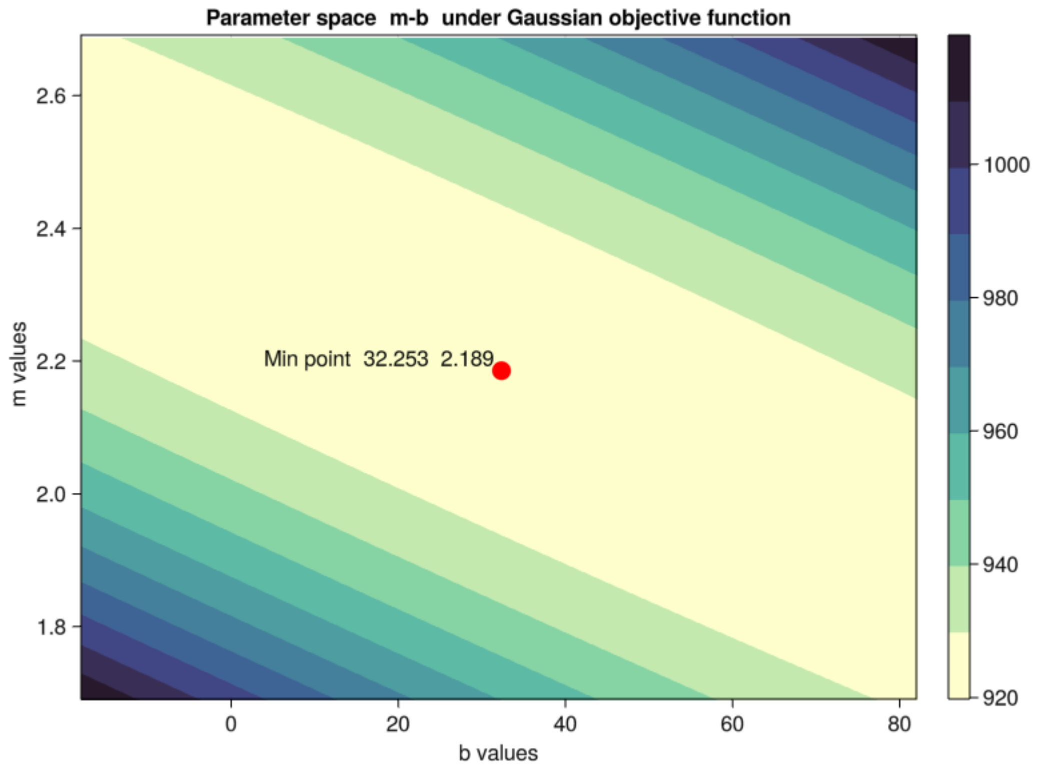
Contour plot with all the data points

In [42]:
```
m = 2.0; b = 10.0;
starting_params = [b, m]
arguments = (data.x, data.y, data.sigma_y)
result = optimize(p -> objective_linear(p, arguments), starting_params)
@assert Optim.converged(result)
b_abs, m_abs = Optim.minimizer(result)
```

Out[42]:
```
2-element Vector{Float64}:
 32.00184588666802
  2.1910407280118998
```

In [43]:
```
ranges = (b_abs-50,b_abs+50,m_abs-0.5,m_abs+0.5)
f, val, bvalue, mvalue = cplots(data.x,data.y,data.sigma_y,objective_gauss,ranges)
f
```

Out[43]:



Objective function considering the outliers

In [44]:
```
function objective_outliers(parameters, x, y, sigma)
    b = parameters[1]
    m = parameters[2]

    frac_bad = 0.01
    like_bad = frac_bad * (1. / 600.)

    y_pred = b .+ m .* x
    like_good = (1. - frac_bad) * 1 ./(sqrt(2*π) .* sigma) .* exp.(-0.5 * (y .- y_
    like = like_bad .+ like_good
    loglike = log.(like)

    return -sum(loglike)
end;
```

In [45]:
```
starting_params = [b + 0., m]
result = optimize(p -> objective_outliers(p, alldata.x, alldata.y, alldata.sigma_y
                    starting_params)
@assert Optim.converged(result)
b_out, m_out = Optim.minimizer(result)
```

Out[45]:
```
2-element Vector{Float64}:
 32.71681761931607
  2.252635451863153
```

In [46]:
```
starting_params = [700 + 0., -0.5]
result = optimize(p -> objective_outliers(p, alldata.x, alldata.y, alldata.sigma_y
                  starting_params)
@assert Optim.converged(result)
b_punk, m_punk = Optim.minimizer(result)
```

Out[46]:
```
2-element Vector{Float64}:
 627.9728284643311
  -0.7590088958472745
```

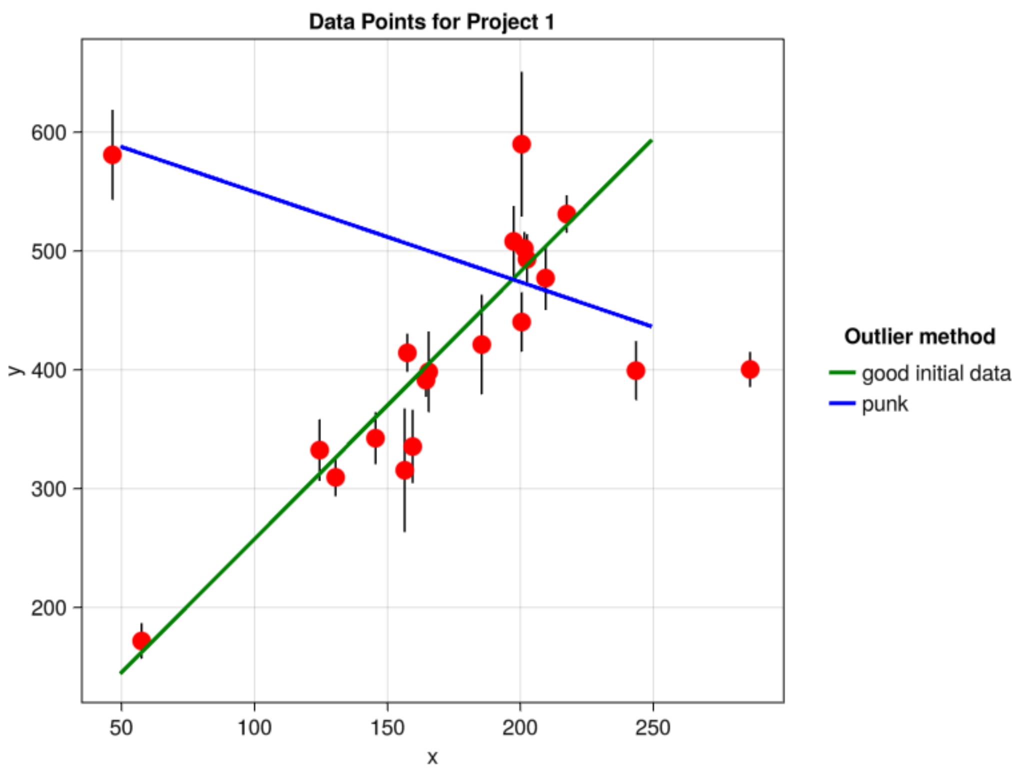
In [47]:
```
f = Figure()
ax=Axis(f[1, 1],xlabel="x", ylabel="y", title="Data Points for Project 1")
errorbars!(alldata.x, alldata.y, alldata.sigma_y)
scatter!(alldata.x, alldata.y, markersize=20, color=:red)

yy_out = xx .* m_out .+ b_out
lines!(xx, yy_out, color=:green, linewidth=3,label="good initial data")

yy_punk = xx .* m_punk .+ b_punk
lines!(xx, yy_punk, color=:blue, linewidth=3,label="punk")
f[1, 2] = Legend(f, ax, "Outlier method", framevisible = false)

f
```

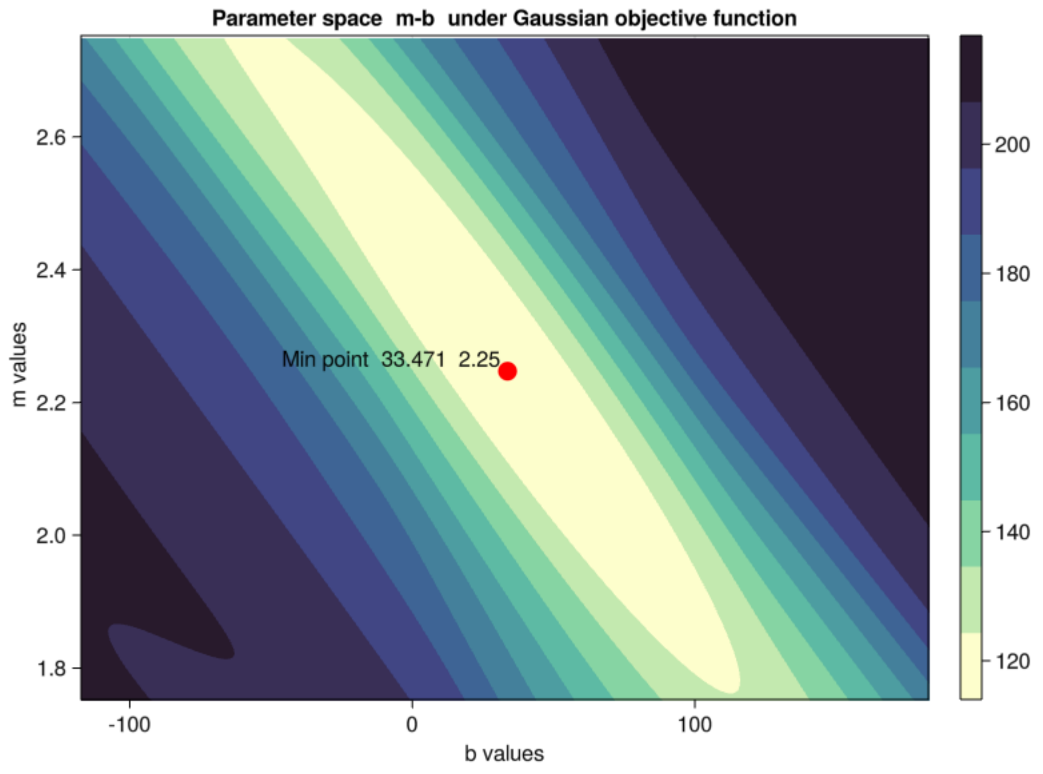
Out[47]:



Outlier objective function plot with minima highlighted

In [48]:
```
ranges = (b_out-150,b_out+150,m_out-0.5,m_out+0.5)
f, val, bvalue, mvalue = cplots(alldata.x,alldata.y,alldata.sigma_y,objective_outl
f
```

Out[48]:

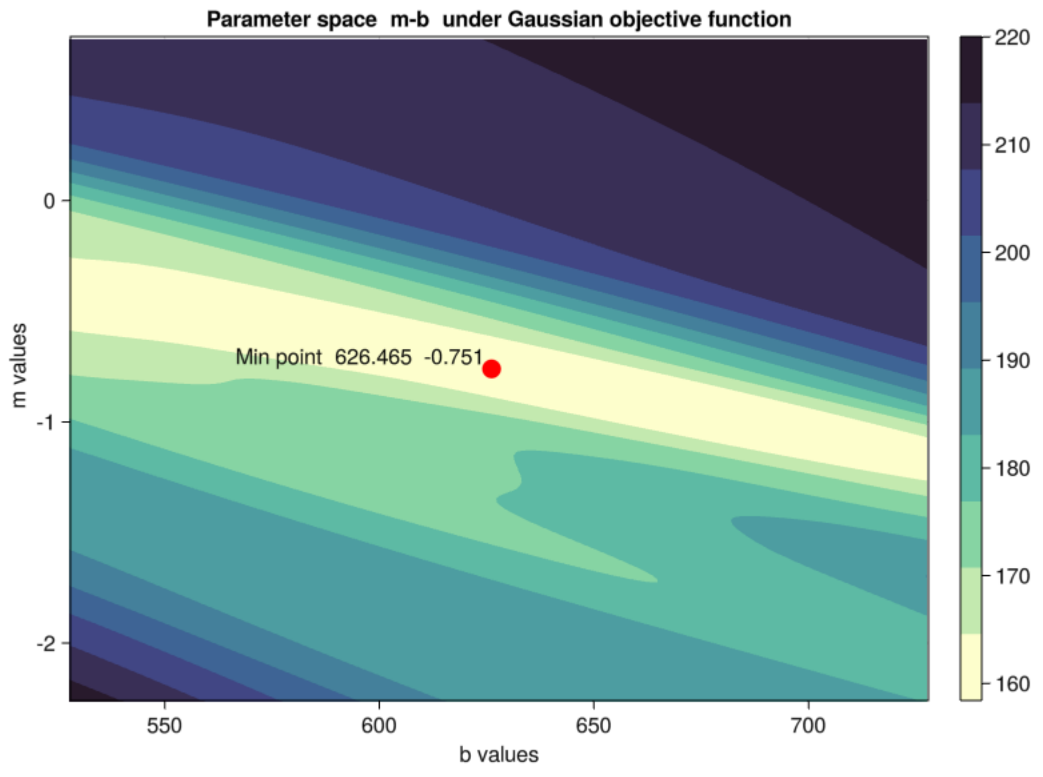


"Punk" minima highlighted :p

In [49]:

```
ranges = (b_punk-100,b_punk+100,m_punk-1.5,m_punk+1.5)
f, val, bvalue, mvalue = cplots(alldata.x,alldata.y,alldata.sigma_y,objective_outl
f
```

Out[49]:



Search for another punk minima

In [50]:

```
starting_params = [400 + 0., 0.25]
result = optimize(p -> objective_outliers(p, alldata.x, alldata.y, alldata.sigma_y
                        starting_params)
@assert Optim.converged(result)
b_punk2, m_punk2 = Optim.minimizer(result)
```

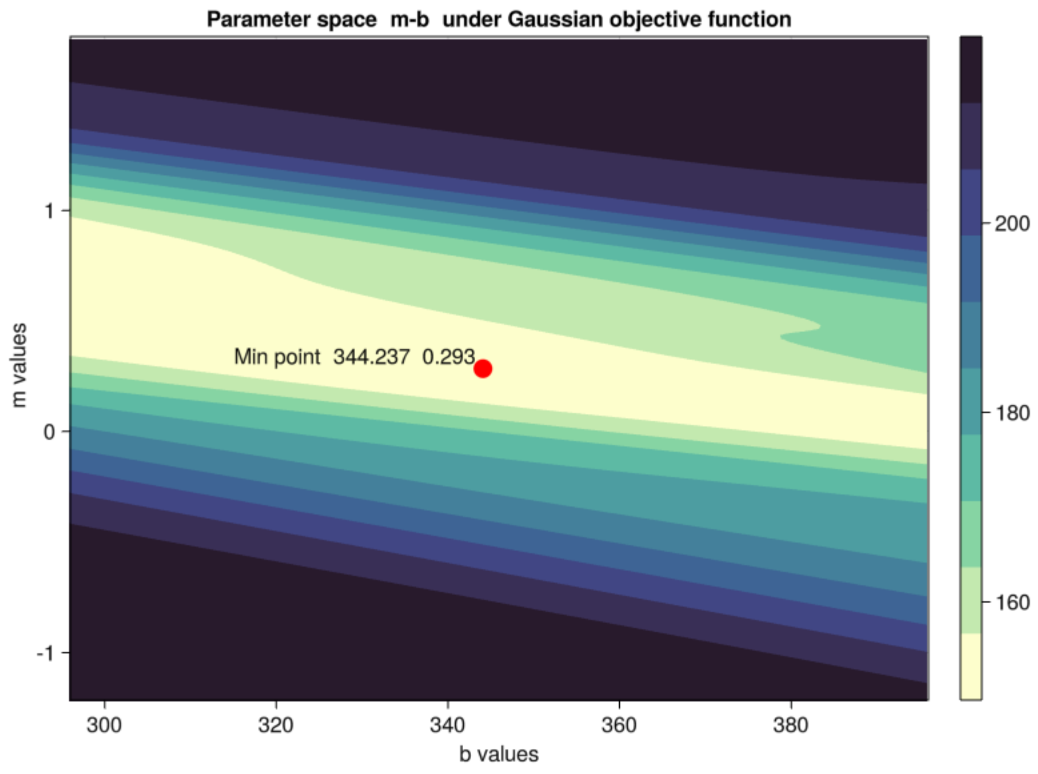
Out[50]:

```
2-element Vector{Float64}:
 345.9957926487757
   0.2858209586946322
```

In [51]:

```
ranges = (b_punk2-50,b_punk2+50,m_punk2-1.5,m_punk2+1.5)
f, val, bvalue, mvalue = cplots(alldata.x,alldata.y,alldata.sigma_y,objective_outl
f
```

Out[51]:



Stretch 2. Record the m and b values that the `optimize()` function tries before it finds a minimum. Plot those points over the contours (it'll probably be clearest to plot them with dots and lines connecting the dots). Try starting it from different initial guesses. You might want to save the m,b values passed to the objective function in lists created outside the function, eg

```
M = []
B = []
function objective_gauss(parameters, x, y, sigma)
  b = parameters[1]
  m = parameters[2]
  push!(M, m)
  push!(B, b)
  ...
end
# Convert from an array of Any to an array of Float64...
M = Float64.(M)
B = Float64.(B)
# I am sure there is a better way to do that!
```

Solution:

In [52]:
```
M = []
B = []
function objective_gauss_modified(parameters, args)
    (x, y, sigma) = args
    b = parameters[1]
    m = parameters[2]
    push!(M,m)
    push!(B,b)
    y_pred = b .+ m .* x
    return -sum(-log.(sigma * sqrt(2 * π)) .-0.5 .* (y .- y_pred).^2 / sigma.^2)
end;
```

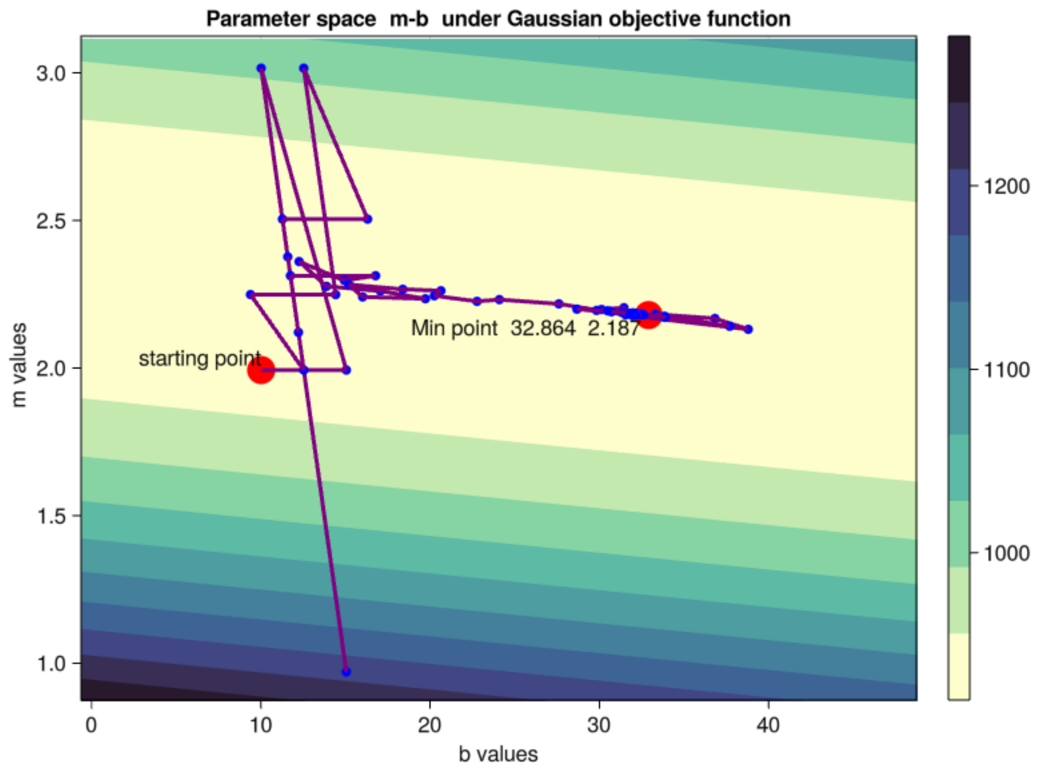
In [53]:
```
starting_params = [b, m]
arguments = (data.x, data.y, data.sigma_y)
result = optimize(p -> objective_gauss_modified(p, arguments), starting_params)
@assert Optim.converged(result)
b_abs, m_abs = Optim.minimizer(result);
M = Float64.(M)
B = Float64.(B);
```

We need to have a function for drawing lines in between the points (x_1, y_1) and (x_2, y_2). So we are going to use previously established formulas for this case. However, to implement the function we need to check that whether the x_1 is equal to x_2 or not. So the function is defined in the following piece of code.

In [54]:
```
function draw_lines(x1,x2,y1,y2)
    if x1>x2
        x = LinRange(x1,x2,10)
        y = (y1-y2)/(x1-x2).*x .+ (y1*x2-y2*x1)/(x2-x1)
    elseif x1<x2
        x = LinRange(x2,x1,10)
        y = (y1-y2)/(x1-x2).*x .+ (y1*x2-y2*x1)/(x2-x1)
    else
        y = LinRange(y1,y2,10)
        x = x1*ones(10)
    end
    return x,y
end;
```

In [55]:
```
f = Figure()
(xmin, xmax, ymin, ymax) = (minimum(B)-10,maximum(B)+10,minimum(M)-0.1,maximum(M)+
Axis(f[1, 1], xlabel = " b values ", ylabel = " m values",
    title = "Parameter space (m-b) under Gaussian objective function")
bvals = LinRange(xmin, xmax, 200)
mvals = LinRange(ymin, ymax, 200);
# Compute the objective function for each point in a grid
og = [objective_gauss([b,m], data.x,data.y,data.sigma_y)
      for b in bvals, m in mvals]
val, arg = findmin(og)
co = contourf!(bvals, mvals, og, colormap = :deep)
Colorbar(f[1, 2], co)
scatter!( bvals[arg[1]], mvals[arg[2]], markersize=30, color=:red)
text!(bvals[arg[1]], mvals[arg[2]],
    text = "Min point (" * string(round(bvals[arg[1]],digits=3))*
    " ," * string(round(mvals[arg[2]],digits=3))* ")" , align = (:right, :top))
iterations = length(M)-1
scatter!( B[1], M[1], markersize=30, color=:red)
text!(B[1], M[1], text = "starting point", align = (:right, :bottom))
for i in 1:iterations
    xx1,yy1 = draw_lines(B[i],B[i+1],M[i],M[i+1])
    scatter!( B[i+1], M[i+1], markersize=10, color=:blue)
    lines!(xx1,yy1,color=:purple, linewidth=3)
end
f
```

Out[55]:



Problem 3

Remember that we found that our m, b estimates were anti-correlated. Can you rephrase or reparameterize the equation for a line so that the variables are less correlated? First, you can try subtracting the average x value from all the x values, and find m', b' for that modified data set. When you re-run the jack-knife test, does the covariance for the new m', b' values look better? What happens if you try to convert your m', b' values back to m, b values for the original data set?

Solution:

The anti-correlation between m and b values has been shown using plots.

In [56]:

```
using Statistics
```

In [57]:

```
Mean_of_data_x = mean(data.x)
newdata_x = data.x .- Mean_of_data_x;
```

In [63]:

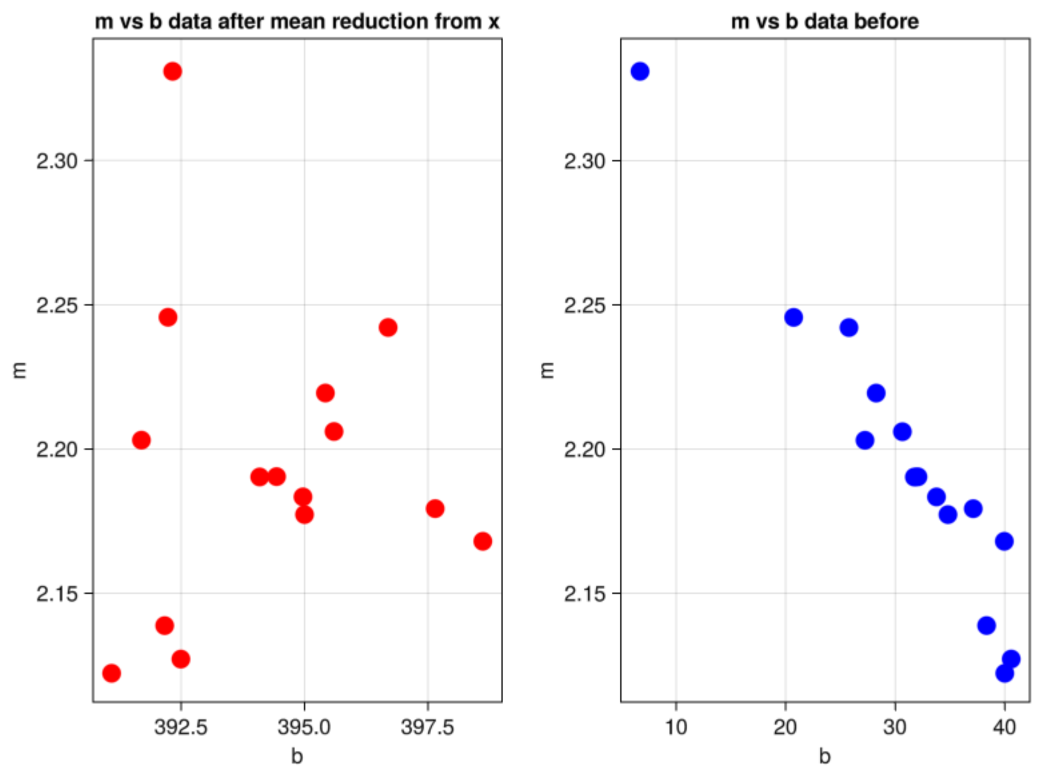
```
ndata = size(data,1)
final_result = zeros(ndata,2)
final_result = jack_knife!(newdata_x, data.y, data.sigma_y, objective_linear,[b, m
(Bnew_jack, Mnew_jack) = (final_result[:,1], final_result[:,2]);
```

In [64]:

```
f = Figure()
Axis(f[1, 1],xlabel="b", ylabel="m", title="m vs b data after mean reduction from 
scatter!(Bnew_jack, Mnew_jack, markersize=20, color=:red)
```

```
Axis(f[1, 2],xlabel="b", ylabel="m", title="m vs b data before")
scatter!(B_jack, M_jack, markersize=20, color=:blue)
f
```

Out[64]:



In [60]:

```
@show cov(B_jack,M_jack), cov(Bnew_jack,Mnew_jack);
```

```
(cov(B_jack, M_jack), cov(Bnew_jack, Mnew_jack)) = (-0.4540983764816002, 0.0034437
647291590606)
```

It is clear form the plot and covariance value of m and b that the amount of correlation has been reduced by removing the mean from the data in x corrdinate. Now if we want to covert back to our previous data points we need to update the values of b' only using corresponding m'. The update formula is just

$$b = b' - \bar{x}m'$$
$$m = m'$$

In [61]:

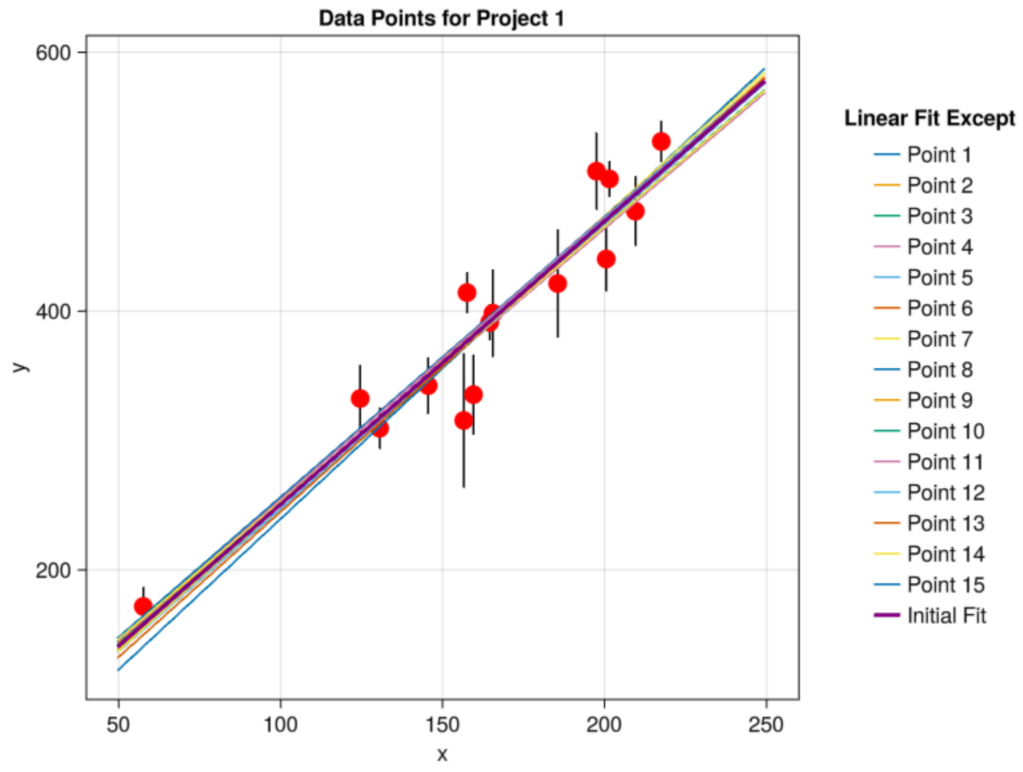
```
f = Figure()
ax = Axis(f[1, 1],xlabel="x", ylabel="y", title="Data Points for Project 1")
errorbars!(data.x, data.y, data.sigma_y)
scatter!(data.x, data.y, markersize=20, color=:red)

Bnew_updated_jack = Bnew_jack .- Mean_of_data_x .* Mnew_jack

for i in 1:ndata
    yy_jack =  xx .* Mnew_jack[i] .+ Bnew_updated_jack[i]
    lines!(xx, yy_jack,label = "Point " * string(i) )
end

lines!(xx, yy_abs, color=:purple, linewidth=3, label = "Initial Fit")
f[1, 2] = Legend(f, ax, "Linear Fit Except", framevisible = false)
f
```

Out[61]:



- **Stretch 3.** Is there a different line parameterization you can think of that might work better? Eg, what about polar coordinates?

Solution:

Polar coordinate may not be a better choice than slope-intercept model, because expressing straight line using polar coordinate is sort of unnatural. However, for equation of circle polar coordinate may be a better choice. Other form of line parameterization is somewhat similar to $y = mx + b$ formula. However, the slope-intercept formula is affine and better suited for evaluating cost function because the operations are addition and multiplication (no division here). The presence of division might cause some problem in this type of optimization problem. That is why I think this slope-intercept parameterization of straight line is best.

- **Stretch 4.** Throughout the notebook, try doing some "software engineering" as Erik described. When I wrote the notebook in class, I copy-pasted a lot of code. Once I copy-pasted a block of code three or four times, I probable should have gone to the effort of turning it into a proper function. For example, calling the optimizer and then fetching the minimizer() values might make a good function. Or the jack-knife procedure would probably make a good function. Be sure to include some documentation about your functions, including some examples of how to call them. Optionally, are there tests you can think of that will exercise your new functions?

Solution:

The `jack_knife!` and `cplots` functions are implemented as per suggestions. I don't know how to implement tests for this kind of functions because I don't know any other way of writing these functions to crosscheck it. It may be possible to compactify the optimization process call, but it is just one or two lines. That is why I didn't bother to write a function to combine those.

Acknowledgement:

In this project, I didn't collaborate with anyone.