Heritability simulations:

This repository contains a julia script that performs simulations of genetic and environmental effects on phenotypic traits.

The main features of the script include:

Simulation of genetic variation: The script simulates parental genotypes for a specified number of loci, considering both additive and dominant effects. It then generates offspring genotypes based on parental genotypes and calculates resulting phenotypic values. Environmental variation: Environmental effects on phenotypic traits are simulated by specifying a range of environmental values. Phenotypic values are calculated by incorporating both genetic and environmental effects. Regression analysis: Linear regression models are fitted to the simulated data to explore the relationship between parental and offspring phenotypic values. The script calculates regression coefficients, intercepts, and R-squared values to quantify the strength of the relationship. Visualization: The script generates visualizations, including scatter plots of parental-offspring phenotypic values, histograms of regression coefficients and intercepts, and a combined plot displaying regression lines for multiple simulations. The script provides insights into the interaction between genetic and environmental factors in determining phenotypic variation, making it useful for educational purposes and research in quantitative genetics.

How to use it?

1. Download this GitHub repository.
2. Install julia and Pluto.jl.
3. Select the parameters using the sliders below
4. Visualize the scatterplots, the heritability is the slope of the graph

```julia
begin
    using Markdown
    using InteractiveUtils
    using DataFrames
    using Random
    using Statistics
    using GLM
    using RCall
    using PlutoUI
    using Images
end
```

```julia
begin
    @rimport ggplot2 as ggplot2
    @rimport gridExtra as gridExtra
end
```

In our classroom, comprising 30 groups of 4 students each, every group is tasked with handling 4 pumpkins. These pumpkins possess genotypes characterized by 6 loci, each consisting of zeroes and ones, dictating the diameter of the pumpkins.

During the initial exercise, students study the concept of additivity within a selection experiment. They start by computing the phenotype of each pumpkin, which involves aggregating the effects of alleles and incorporating environmental variation introduced through a die roll (-3, -2, -1, +1, +2, +3) cm.

Subsequently, students select the pumpkins exhibiting the highest and lowest phenotypes and generate gametes. They accomplish this by assigning values for homozygotes or flipping a coin for heterozygotes at each locus.

These gametes are then combined to produce new offspring, with students calculating the phenotype of each offspring in the same manner.

Finally, the class collaboratively conducts a regression analysis on the midparent and midoffspring averages to extract insights from the experiment.

In the second activity, students replicate the process but adhere to complete dominance rules, where (1,0)=(1,1)=2cm and (0,0)=0cm per loci.

Each simulation represents a semester for the class Ecological Genetics in Stony Brook University.

50

```
@bind num_simulations Slider(1:500,50,true)
```

Each group has 4 students (4 pumpkins each), the typical number of groups is 30.

30

```
@bind num_groups Slider(1:30,30,true)
```

The baseline diameter for a pumpkin is 10 cm.

10

```
@bind baseline Slider(-20:20,10,true)
```

We can simulate n number of loci:

```
md"""
We can simulate n number of loci:
"""
```

6

```
@bind number_loci Slider(0:15,6,true)
```

With a minimum number of loci being heterozygotes.

3

```
@bind min_num_hetero Slider(0:15,3,true)
```

Different number of dominant loci can be selected

2

```
@bind number_of_dom Slider(0:15,2,true)
```

The effect of A1 is +1cm. An A1A1 individual would have +2cm.

1

```
@bind effect_A1 Slider(-3:3,1,true)
```

0

```julia
@bind effect_A2 Slider(-3:3,0,true)
```

The effect of dominance can be toggled to simulate partial dominance, total dominance, underdominance and overdominance. d is the effect over the mid-effect value, mean(A1,A2) d=0 (additivity), d=1 (total dominance)

0

```julia
@bind effect_d Slider(-2:2,0,true)
```

In the activity we model a uniform distribution by rolling a die, however it may be useful to explore a normal distribution of environmental effects.

Uniform

```julia
@bind env_type Select(["Uniform","Normal"])
```

env_effect is a multiplier of the environmental range.

1

```julia
@bind env_effect Slider(-3:3,1,true)
```

env_sd is a parameter to model the normal distribution variance.

2.0

```julia
@bind env_sd Slider(-3.0:3.0,2.0,true)
```

Epistasis can also be modeled by adding a value if a minimum of loci (epistasis_level) has a value of A1 per loci equal or greater than N.

false

```julia
@bind epistasis Select([false,true])
```

3.0

```julia
@bind epistasis_level Slider(0.0:6.0,3.0,true)
```

10.0

```julia
@bind epistasis_value Slider(-10.0:10.0,10.0,true)
```

```
RObject{VecSxp}
TableGrob (2 x 2) "arrange": 3 grobs
  z     cells    name           grob
1 1 (1-1,1-1) arrange gtable[layout]
2 2 (1-1,2-2) arrange gtable[layout]
3 3 (2-2,1-1) arrange gtable[layout]
```

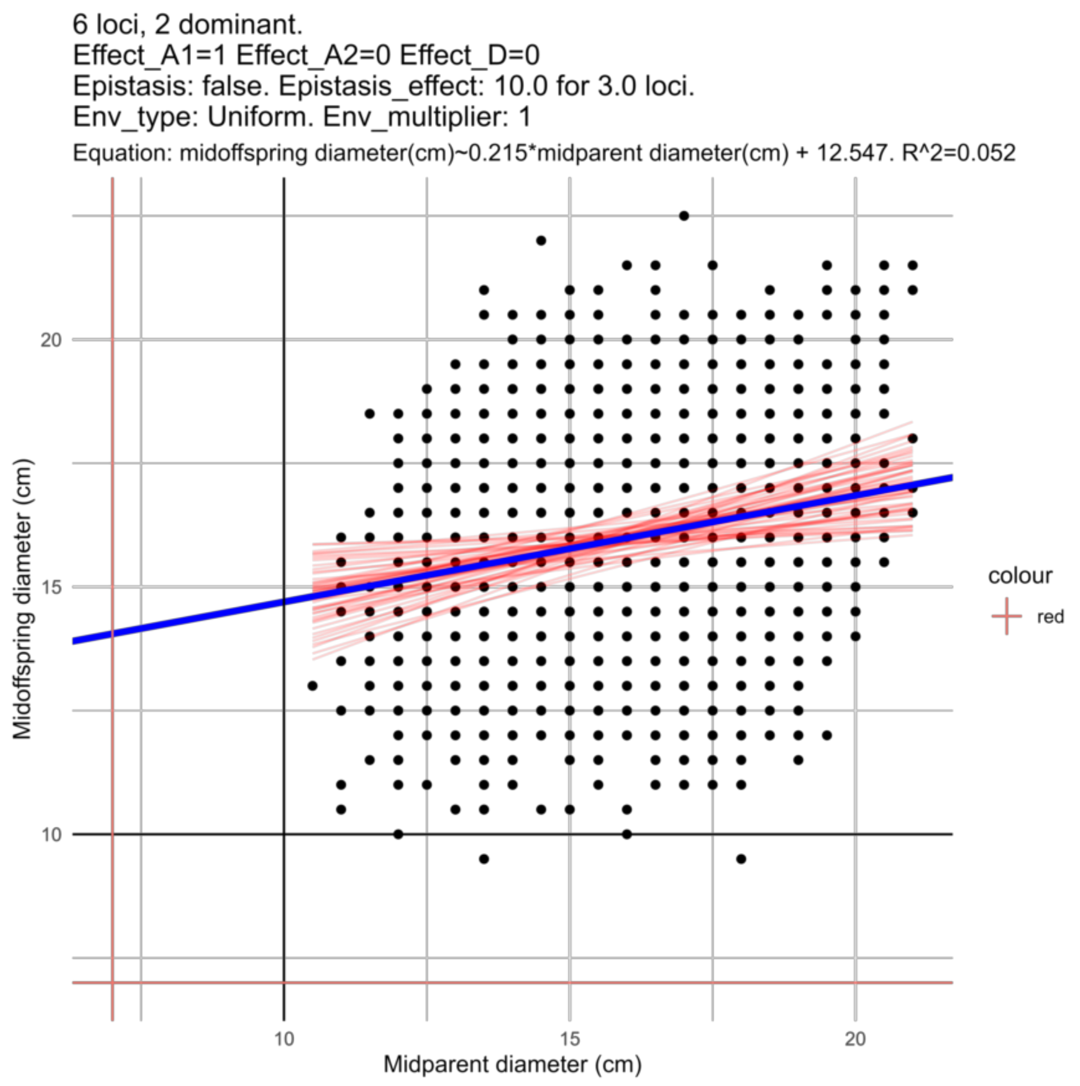
```
begin
    mid_effect = mean([effect_A1, effect_A2])
    effect_a1 = effect_A1 - mid_effect
    effect_a2 = effect_A2 - mid_effect
    possible_num_homo = 0:(number_loci - min_num_hetero)
    number_of_add = number_loci - number_of_dom
    add_vec = fill("ADD", number_of_add)
    dom_vec = fill("DOM", number_of_dom)
    all_effect = vcat(add_vec, dom_vec)

        if env_type == "Uniform"
            env_range = vcat(-3:-1, 1:3)
        elseif env_type == "Normal"
            env_range = round.(randn(1000) .* env_sd, digits=2)
        else
        println("error")
        end

        minimum_pheno_value = baseline + env_effect * minimum(env_range) + 2 *
        number_loci * minimum([effect_A1, effect_A2])

    # Define functions:
    # Function to sample genotypesfunction sample_genotype(num_loci)
    function sample_genotype(num_loci)
        possible_num_homo = 0:(num_loci - min_num_hetero)
        hom = rand(possible_num_homo)
        het = num_loci - hom
        hom_vec = reshape(fill("HOM", hom),1,:)
        het_vec = reshape(fill("HET", het),1,:)
        all_loci = hcat(hom_vec, het_vec)
        all_loci = shuffle(all_loci)
        chromosome1 = reshape(rand(["A1", "A2"], num_loci), 1, :)
        chromosome2 = reshape(Vector{String}(undef, num_loci),1,:)

        for i in 1:length(all_loci)
        if all_loci[i] == "HOM"
            chromosome2[i] = chromosome1[i]
        elseif all_loci[i] == "HET"
            chromosome2[i] = (chromosome1[i] == "A1" ? "A2" : "A1")
        else
            println("error")
        end
    end

        genotype = vcat(chromosome1, chromosome2)
        return genotype
    end

function get_phenotype(geno, baseline, effect_a1, effect_a2, effect_d,effects)
    effect_val = []
        for e in 1:length(effects) # For every loci
            if effects[e] == "DOM" # If the locus is dominant
                if geno[1, e] == "A1" && geno[2, e] == "A1"
                    push!(effect_val, 2 * (mid_effect + effect_a1))
                elseif (geno[1, e] == "A1" && geno[2, e] == "A2") || (geno[1, e] ==
    "A2" && geno[2, e] == "A1")
                    push!(effect_val, mid_effect + effect_d + mid_effect + effect_d)
                elseif geno[1, e] == "A2" && geno[2, e] == "A2"
                    push!(effect_val, 2 * (mid_effect + effect_a2))
                else
                    println("error")
                end
            elseif effects[e] == "ADD" # If the locus is additive
```

```julia
            if geno[1, e] == "A1" && geno[2, e] == "A1"
                push!(effect_val, 2 * (mid_effect + effect_a1))
            elseif (geno[1, e] == "A1" && geno[2, e] == "A2") || (geno[1, e] == "A2" && geno[2, e] == "A1")
                push!(effect_val, mid_effect + effect_a1 + mid_effect + effect_a2)
            elseif geno[1, e] == "A2" && geno[2, e] == "A2"
                push!(effect_val, 2 * (mid_effect + effect_a2))
            else
                println("error")
            end
        else
            println("error")
        end
    end

# Epistasis term
    if epistasis == true # If epistasis is present
        epis_val = sum(effect_val .>= 1) >= epistasis_level
        geno_val = epis_val ? sum(effect_val) + epistasis_value : sum(effect_val)
    else
        # No epistasis
        geno_val = sum(effect_val)
    end

    env_val = rand(env_range) # Sample the environmental range
    pheno = baseline + env_effect * env_val + geno_val
    return [baseline env_val geno_val pheno]
end

    # Function to get offspring given two parental genotypes
function get_offspring(genotype1, genotype2)
    gamete1 = []
    for i in 1:size(genotype1, 2)
        if genotype1[1, i] == genotype1[2, i]
            push!(gamete1, genotype1[1, i])
        elseif genotype1[1, i] != genotype1[2, i]
            push!(gamete1, rand([genotype1[1, i], genotype1[2, i]]))
        end
    end

    gamete2 = []
    for i in 1:size(genotype2, 2)
        if genotype2[1, i] == genotype2[2, i]
            push!(gamete2, genotype2[1, i])
        elseif genotype2[1, i] != genotype2[2, i]
            push!(gamete2, rand([genotype2[1, i], genotype2[2, i]]))
        end
    end

    gamete1=reshape(gamete1,1,:)
    gamete2=reshape(gamete2,1,:)

    new_genotype = vcat(gamete1, gamete2)
    return new_genotype
end

slope_vec = Vector{Float64}()
intercept_vec = Vector{Float64}()
R_squared_vec =Vector{Float64}()
plot_vec = []
all_sim = []
p1 = []
for simulation in 1:num_simulations
    println("Simulation: $simulation")
    all_data = [] # List with all_data per simulation

for group in 1:num_groups
    println("Group: $group")

    geno_par = Dict(
        "par1" => sample_genotype(number_loci),
        "par2" => sample_genotype(number_loci),
        "par3" => sample_genotype(number_loci),
        "par4" => sample_genotype(number_loci))

    pheno_par1 = get_phenotype(geno_par["par1"], baseline, effect_a1, effect_a2, effect_d,all_effect)
    pheno_par2 =  get_phenotype(geno_par["par2"], baseline, effect_a1, effect_a2, effect_d,all_effect)
    pheno_par3 =  get_phenotype(geno_par["par3"], baseline, effect_a1, effect_a2, effect_d,all_effect)
    pheno_par4 =  get_phenotype(geno_par["par4"], baseline, effect_a1, effect_a2, effect_d,all_effect)

    pheno_table=DataFrame(vcat(pheno_par1, pheno_par2, pheno_par3, pheno_par4),:auto)
    name_par = ["par1", "par2", "par3", "par4"]
    pheno_table = hcat(pheno_table, name_par,makeunique=true)
    rename!(pheno_table, [:baseline, :environment_val, :genotype_val, :phenotype_val, :name])
    sort!(pheno_table, :phenotype_val)

    small_parents = pheno_table[1:2, :]
    parentsmall_name1 = small_parents[1, :name]
    parentsmall_name2 = small_parents[2, :name]
    large_parents = pheno_table[3:4, :]
    parentlarge_name1 = large_parents[1, :name]
    parentlarge_name2 = large_parents[2, :name]

    geno_off = Dict(
        "off1_small" => get_offspring(geno_par[parentsmall_name1], geno_par[parentsmall_name2]),
        "off2_small" => get_offspring(geno_par[parentsmall_name1], geno_par[parentsmall_name2]),
        "off1_large" => get_offspring(geno_par[parentlarge_name1], geno_par[parentlarge_name2]),
        "off2_large" => get_offspring(geno_par[parentlarge_name1], geno_par[parentlarge_name2]))

    pheno_off1_small =  get_phenotype(geno_off["off1_small"], baseline, effect_a1, effect_a2, effect_d,all_effect)
    pheno_off2_small =  get_phenotype(geno_off["off2_small"], baseline, effect_a1, effect_a2, effect_d,all_effect)
    pheno_off1_large = get_phenotype(geno_off["off1_large"], baseline, effect_a1, effect_a2, effect_d,all_effect)
    pheno_off2_large =  get_phenotype(geno_off["off2_large"], baseline, effect_a1, effect_a2, effect_d,all_effect)

    all_data_df = DataFrame(
        simulation_n=simulation,
        group_n = group,
        largep_env1=large_parents[1,2],
        largep_geno1=large_parents[1,3],
        largep_pheno1=large_parents[1,4],
        largep_env2=large_parents[2,2],
        largep_geno2=large_parents[2,3],
        largep_pheno2=large_parents[2,4],
        midparent_large = mean(large_parents.phenotype_val),
        largeo_env1=pheno_off1_large[2],
        largeo_geno1=pheno_off1_large[3],
        largeo_pheno1=pheno_off1_large[4],
        largeo_env2=pheno_off2_large[2],
        largeo_geno2=pheno_off2_large[3],
        largeo_pheno2=pheno_off2_large[4],
        midoffspring_large = mean([pheno_off1_large[4] pheno_off2_large[4]]),
        smallp_env1=small_parents[1,2],
            smallp_geno1=small_parents[1,3],
            smallp_pheno1=small_parents[1,4],
            smallp_env2=small_parents[2,2],
            smallp_geno2=small_parents[2,3],
            smallp_pheno2=small_parents[2,4],
            midparent_small = mean(small_parents.phenotype_val),
            smallo_env1=pheno_off1_small[2],
            smallo_geno1=pheno_off1_small[3],
            smallo_pheno1=pheno_off1_small[4],
            smallo_env2=pheno_off2_small[2],
            smallo_geno2=pheno_off2_small[3],
            smallo_pheno2=pheno_off2_small[4],
            midoffspring_small = mean([pheno_off1_small[4], pheno_off2_small[4]]),
        )

        large_df = all_data_df[:,1:16]
        large_df = hcat(large_df, DataFrame(type = "large"))
        rename!(large_df,[:simulation,:group,:p_env1,:p_geno1,:p_pheno1,
        :p_env2,:p_geno2,:p_pheno2,
        :midpar,
        :o_env1,:o_geno1,:o_pheno1,
        :o_env2,:o_geno2,:o_pheno2,
        :midoff,:type])

        small_df = all_data_df[:,[1; 2; 17:30]]
        small_df = hcat(small_df, DataFrame(type = "small"))
        rename!(small_df,[:simulation,:group,:p_env1,:p_geno1,:p_pheno1,
        :p_env2,:p_geno2,:p_pheno2,
        :midpar,
        :o_env1,:o_geno1,:o_pheno1,
        :o_env2,:o_geno2,:o_pheno2,
        :midoff,:type])

        all_data_df2 = vcat(small_df,large_df)
        push!(all_data, all_data_df2)
    end


    # Concatenate into single dataframe:
    all_g_df = vcat(all_data...)

    mod1 = lm(@formula(midoff ~ midpar), all_g_df)
    intercept1 = round(coef(mod1)[1],digits=3)
    slope1 = round(coef(mod1)[2],digits=3)
    r_squared1 = round(r2(mod1),digits=3)

    push!(slope_vec, slope1)
    push!(intercept_vec, intercept1)
    push!(R_squared_vec, r_squared1)

    subtitle1 = "y~" * string(slope1) * "*x + " * string(intercept1) * ". R^2=" *
    string(r_squared1)

    # Store list of dataframes:
    push!(all_sim, all_g_df)

    end

    # Combine all data into single df:
    all_sim_df = vcat(all_sim...)

    mod2 = lm(@formula(midoff ~ midpar), all_sim_df)
    intercept2 = round(coef(mod2)[1],digits=3)
    slope2 = round(coef(mod2)[2],digits=3)
    r_squared2 = round(r2(mod2),digits=3)

    title2 = string(number_loci) * " loci, " * string(number_of_dom) * " dominant.\n"
    * "Effect_A1=" * string(effect_A1) * " Effect_A2=" * string(effect_A2) * "
    Effect_D=" * string(effect_d) * "\n" * "Epistasis: " * string(epistasis) * ".
    Epistasis_effect: " * string(epistasis_value) * " for " * string(epistasis_level)
    * " loci.\n" * "Env_type: " * env_type * ". Env_multiplier: " * string(env_effect)

    subtitle2 = "Equation: midoffspring diameter(cm)~" * string(slope2) * "*midparent
    diameter(cm) + " * string(intercept2) * ". R^2=" * string(r_squared2)

    # Initialize an empty DataFrame to store regression lines
    regression_lines_df = DataFrame(x = Float64[], y = Float64[], simulation =
    String[])

    # Loop through each slope and intercept combination
    for i in 1:length(slope_vec)
        # Calculate the endpoints of the line using the range of x values
        x_range_ln = extrema(all_sim_df.midpar)
        y_range_ln = slope_vec[i] .* x_range_ln .+ intercept_vec[i]

        xy_tuples = [(x_range_ln[i], y_range_ln[i]) for i in 1:2]
        df = DataFrame(x = [xy[1] for xy in xy_tuples], y = [xy[2] for xy in
    xy_tuples])
        df.simulation .= "Simulation $i"

        # Add this line to the DataFrame storing all regression lines
        append!(regression_lines_df, df)
    end

    "Heritability less than 0=" * string(sum(slope_vec.<=0)/length(slope_vec))

    "Heritability less than 0.45=" * string(sum(slope_vec .<= 0.45)/length(slope_vec))

    "Average Heritability" * string(mean(slope_vec))

    df2= DataFrame(slope=slope_vec,intercept=intercept_vec)

p2=ggplot2.ggplot(all_sim_df, ggplot2.aes(x=:midpar, y=:midoff)) +
ggplot2.geom_point()+  ggplot2.geom_line(data = regression_lines_df,
                ggplot2.aes(x = :x, y = :y, group = :simulation),
                color = "red", alpha = 0.2)+
    ggplot2.geom_abline(slope=slope2,
                intercept=intercept2,
                col="blue",lwd=1.5)+
      ggplot2.xlab("Midparent diameter (cm)")+
      ggplot2.ylab("Midoffspring diameter (cm)")+
      ggplot2.theme_minimal() +
      ggplot2.labs(title=title2,
              subtitle=subtitle2)+
    ggplot2.geom_vline(ggplot2.aes(xintercept=10))+
      ggplot2.geom_hline(ggplot2.aes(yintercept=10))+
      ggplot2.geom_vline(ggplot2.aes(xintercept=minimum_pheno_value,col="red"))+
      ggplot2.geom_hline(ggplot2.aes(yintercept=minimum_pheno_value,col="red"))

    scatter=ggplot2.ggplot(df2,ggplot2.aes(x=:slope,y=:intercept))+
      ggplot2.theme_minimal()+
      ggplot2.geom_hex()

    hist_bottom = ggplot2.ggplot()+
        ggplot2.geom_histogram(data=df2,ggplot2.aes(:slope))+
        ggplot2.theme_minimal()

    hist_right = ggplot2.ggplot()+
        ggplot2.geom_histogram(data=df2,ggplot2.aes(:intercept))+
        ggplot2.coord_flip()+
        ggplot2.theme_minimal()

    p3 = gridExtra.grid_arrange(scatter,hist_right,hist_bottom,
    ncol=2, nrow=2, widths=(4, 1), heights=(4, 1))
```

```
Simulation: 1
Group: 1
```

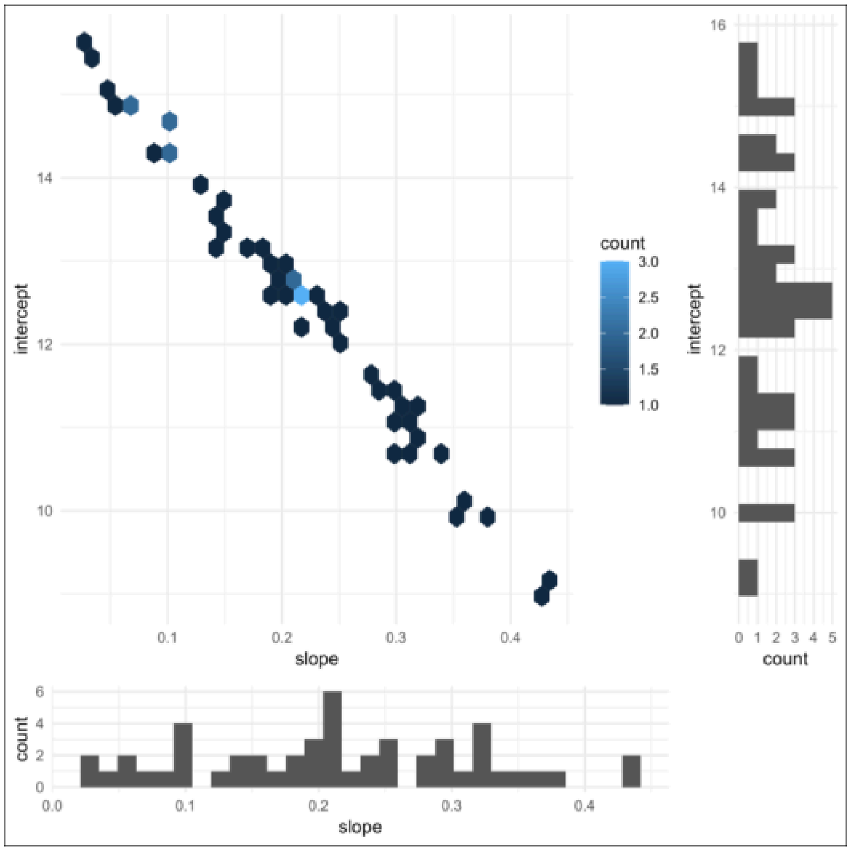
```
Group: 2
Group: 3
Group: 4
Group: 5
Group: 6
Group: 7
Group: 8
Group: 9
Group: 10
Group: 11
Group: 12
Group: 13
Group: 14
Group: 15
Group: 16
Group: 17
Group: 18
Group: 19
Group: 20
Group: 21
Group: 22
Group: 23
Group: 24
Group: 25
Group: 26
Group: 27
Group: 28
Group: 29
Group: 30
Simulation: 2
Group: 1
Group: 2
Group: 3
Group: 4
Group: 5
Group: 6
Group: 7
```

The slope is the heritability for all simulations (blue line). Each red line represent and individual simulation.



```
begin
        ggplot2.ggsave(p2,file="p2.png",dpi=1000,height=7,width=7)
        im1 = Images.load("p2.png")
end
```

This plot shows the slopes and intercepts of all simulations.



```
begin
	ggplot2.ggsave(p3,file="p3.png",dpi=600)
	im2 = Images.load("p3.png")
end
```

```
Saving 7 x 7 in image
```