Отчёт по лабораторной работе №5

Построение графиков

Гань Чжаолун

Цель работы

Основная цель работы — освоить синтаксис языка Julia для построения графиков.

Выполнение лабораторной работы

5.2. Предварительные сведения

5.2.1. Основные пакеты для работы с графиками в Julia

```
using Pkg
Pkg.add("Plots")
Pkg.add("PyPlot")
Pkg.add("Plotly")
Pkg.add("UnicodePlots")

# подключаем для использования Plots:
using Plots
```

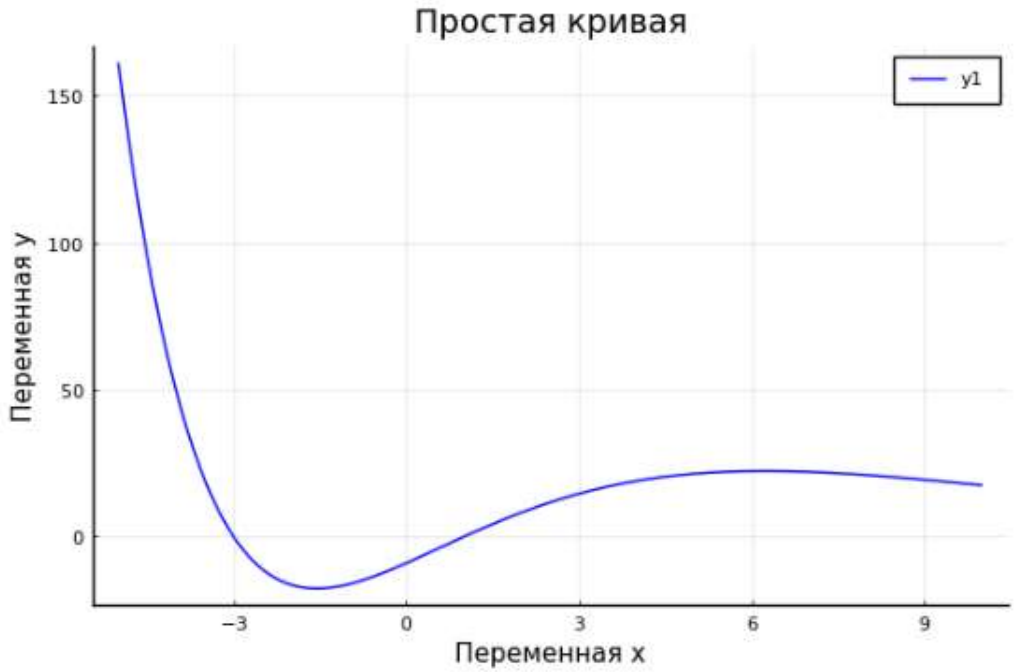
```
   Updating registry at `~/.julia/registries/General`
  Resolving package versions...
No Changes to `~/.julia/environments/v1.5/Project.toml`
No Changes to `~/.julia/environments/v1.5/Manifest.toml`
  Resolving package versions...
No Changes to `~/.julia/environments/v1.5/Project.toml`
No Changes to `~/.julia/environments/v1.5/Manifest.toml`
  Resolving package versions...
No Changes to `~/.julia/environments/v1.5/Project.toml`
No Changes to `~/.julia/environments/v1.5/Manifest.toml`
  Resolving package versions...
No Changes to `~/.julia/environments/v1.5/Project.toml`
No Changes to `~/.julia/environments/v1.5/Manifest.toml`
```

Построение графика функции

```
# задание функции:
f(x) = (3x.^2 + 6x .- 9).*exp.(-0.3x)

# генерирование массива значений x в диапазоне от -5 до 10 с шагом 0,1
# (шаг задан через указание длины массива):
x = collect(range(-5,10,length=151))
# генерирование массива значений y:
y = f(x)

# указывается, что для построения графика используется gr():
gr()
# задание опций при построении графика
# (название кривой, подписи по осям, цвет графика):
plot(x,y,
    title="A simple curve",
    xlabel="Variable x",
    ylabel="Variable y",
    color="blue")
```

```
using Plots
# указывается, что для построения графика используется pyplot():
pyplot()
# задание опций при построении графика
# (название кривой, подписи по осям, цвет графика):
plot(x,y,
    title="Простая кривая",
    xlabel="Переменная x",
    ylabel="Переменная y",
    color="blue")
```

Рисунок 1.1 Основные пакеты для работы с графиками в Julia 1




```
using Plots
# указывается, что для построения графика используется plotly():
plotly()
# задание опций при построении графика
# (название кривой, подписи по осям, цвет графика):
plot(x,y,
    title="Простая кривая",
    xlabel="Переменная x",
    ylabel="Переменная y",
    color="blue")
```

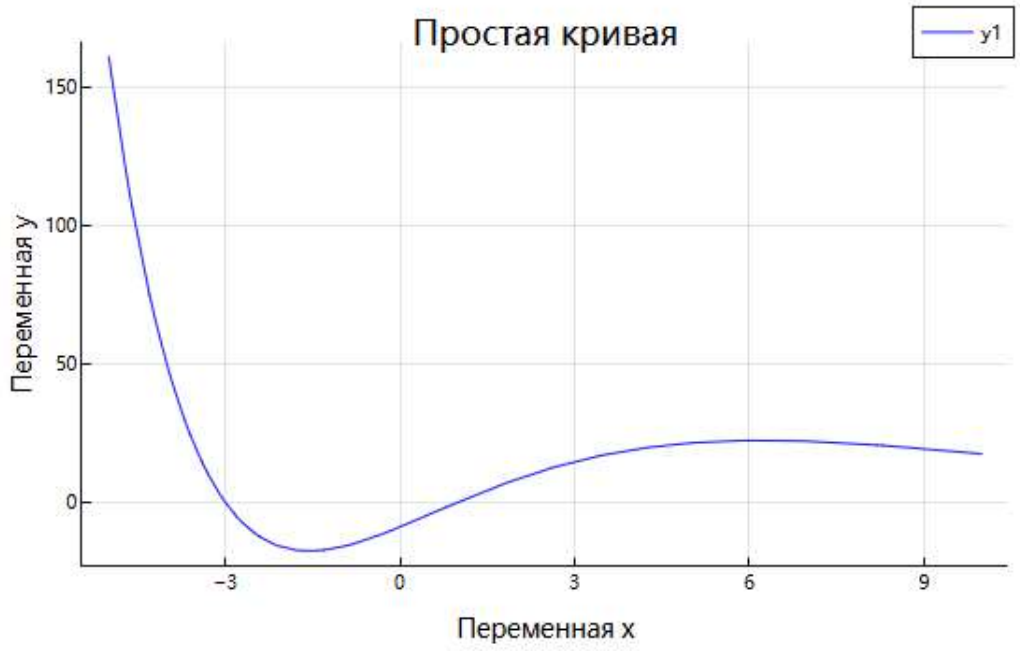


Рисунок 1.2 Основные пакеты для работы с графиками в Julia 2

```
# using Plots
# unicodeplots()
using UnicodePlots #т.к. использование просто в качестве бэкенда не сработало, вызываю функции собственно пакета
# задание опций при построении графика
# (название кривой, подписи по осям, цвет графика):
lineplot(x, y,
    title="Простая кривая",
    xlabel="Переменная x",
    ylabel="Переменная y",
    color= :blue)
```

```
WARNING: using UnicodePlots.histogram in module Main conflicts with an existing identifier.
```

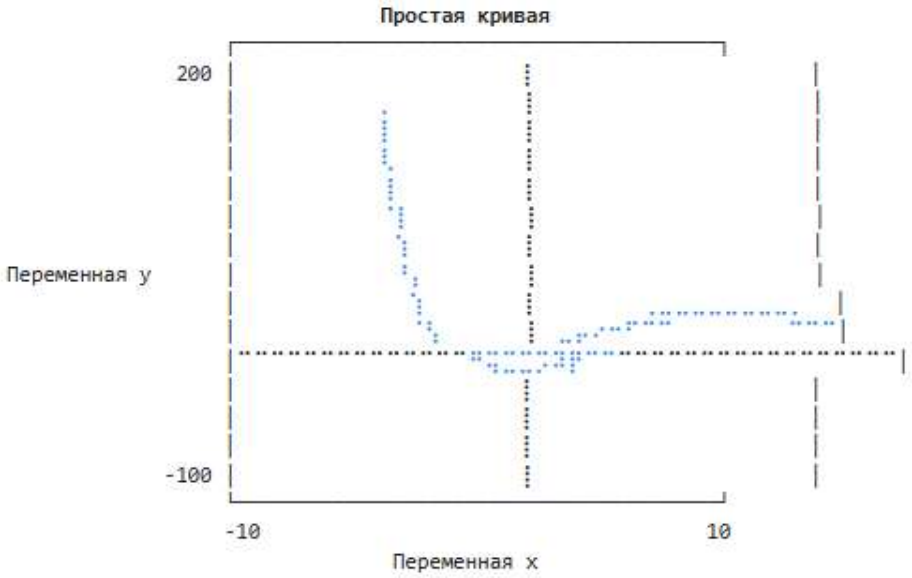
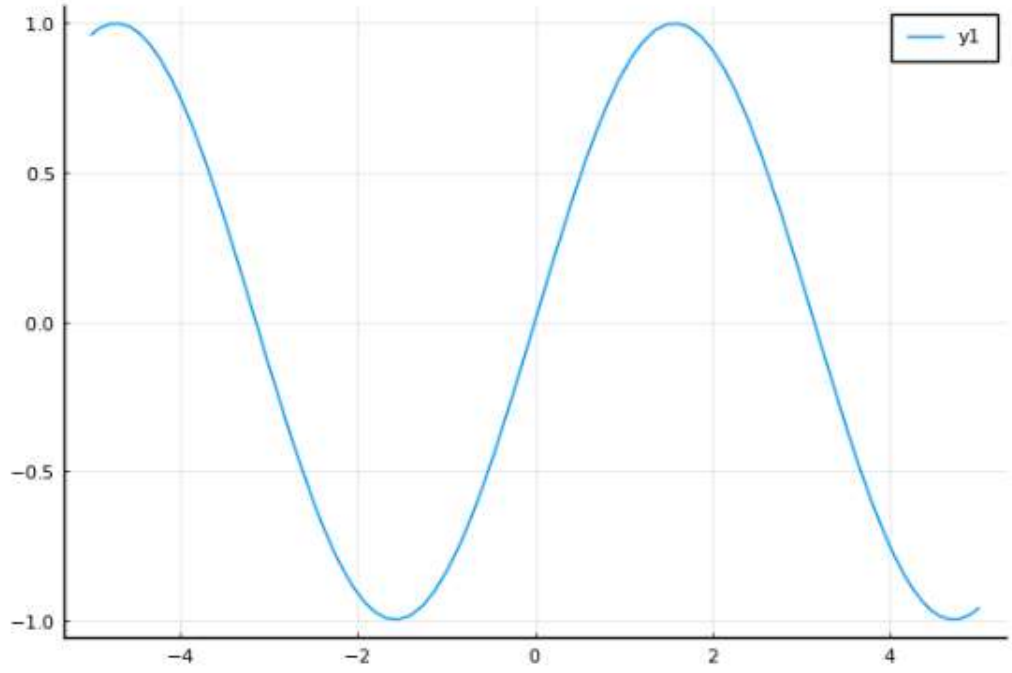


Рисунок 1.3 Основные пакеты для работы с графиками в Julia 3

5.2.2. Опции при построении графика

```
# рассмотрим дополнительные возможности пакетов для работы с графиками
using Plots
# указывается, что для построения графика используется pyplot():
pyplot()
# задание функции sin(x):
sin_theor(x) = sin(x)

# построение графика функции sin(x):
plot(sin_theor)
```




```
pyplot()
sin_taylor(x) = [(-1)^i*x^(2*i+1)/factorial(2*i+1) for i in 0:4] |> sum
# построение графика функции sin_taylor(x):
plot(sin_taylor)
```

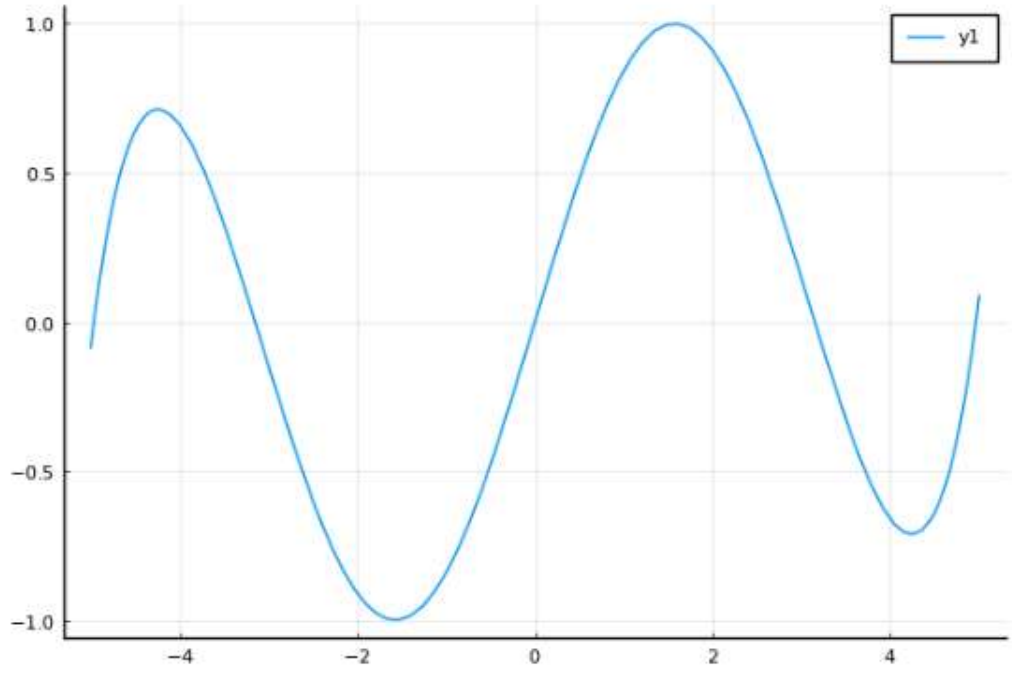
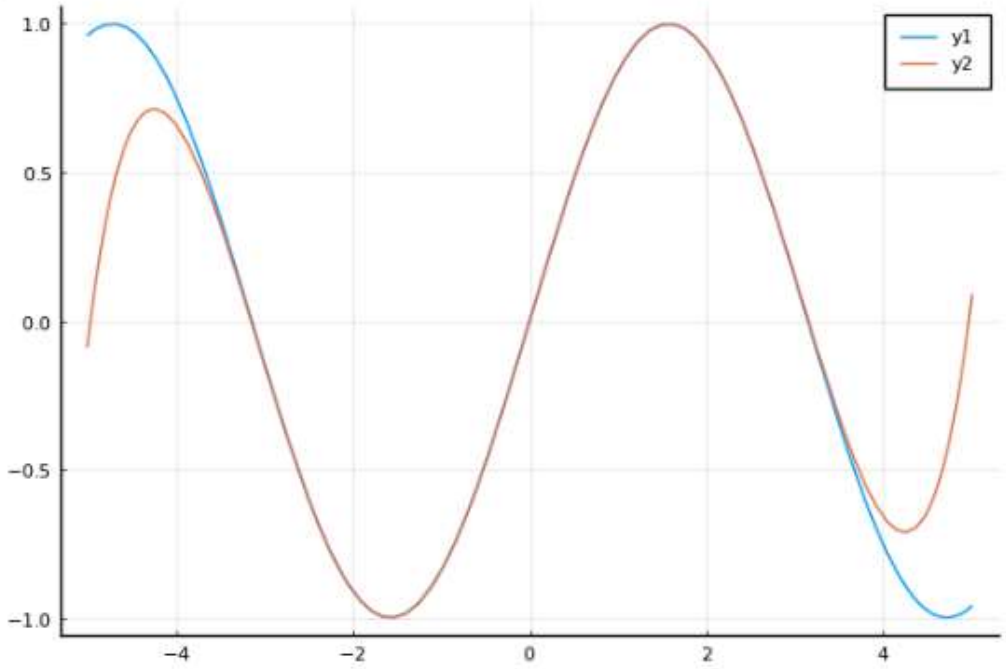


Рисунок 1.4 Опции при построении графика 1

```
# построение двух функций на одном графике:
plot(sin_theor)
plot!(sin_taylor)
```




```
plot(
    # функция sin(x):
    sin_taylor,
    # подпись в легенде, цвет и тип линии:
    label = "sin(x), разложение в ряд Тейлора",
    line=(:blue, 0.3, 6, :solid),
    # размер графика:
    size=(800, 500),
    # параметры отображения значений по осям
    xticks = (-5:0.5:5),
    yticks = (-1:0.1:1),
    xtickfont = font(12, "Times New Roman"),
    ytickfont = font(12, "Times New Roman"),
    # подписи по осям:
    ylabel = "y",
    xlabel = "x",
    # название графика:
    title = "Разложение в ряд Тейлора",
    # поворот значений, заданный по оси x:
    xrotation = rad2deg(pi/4),
    # заливка области графика цветом:
    fillrange = 0,
    fillalpha = 0.5,
    fillcolor = :lightgoldenrod,
    # задание цвета фона:
    background_color = :ivory
)
plot!(
    # функция sin_theor:
    sin_theor,
    # подпись в легенде, цвет и тип линии:
    label = "sin(x), теоретическое значение",
    leg=:topright,
    line=(:black, 1.0, 2, :dash)
)
```

Рисунок 1.5 Опции при построении графика 2


```
# сохранение графика в файле в формате pdf или png:
savefig("taylor.pdf")
savefig("taylor.png")
```

Рисунок 1.6 Опции при построении графика 3

5.2.3. Точечный график

5.2.3.1. Простой точечный график

```
# параметры распределения точек на плоскости:
x = range(1,10,length=10)
y = rand(10)
# параметры построения графика:
plot(x, y,
    seriestype = :scatter,
    title = "Точечный график",
    xlabel = "x",
    ylabel = "y",
    # подпись в легенде, цвет и тип линии:
    label = "y1, теоретическое значение",
    leg = :topright
)
```

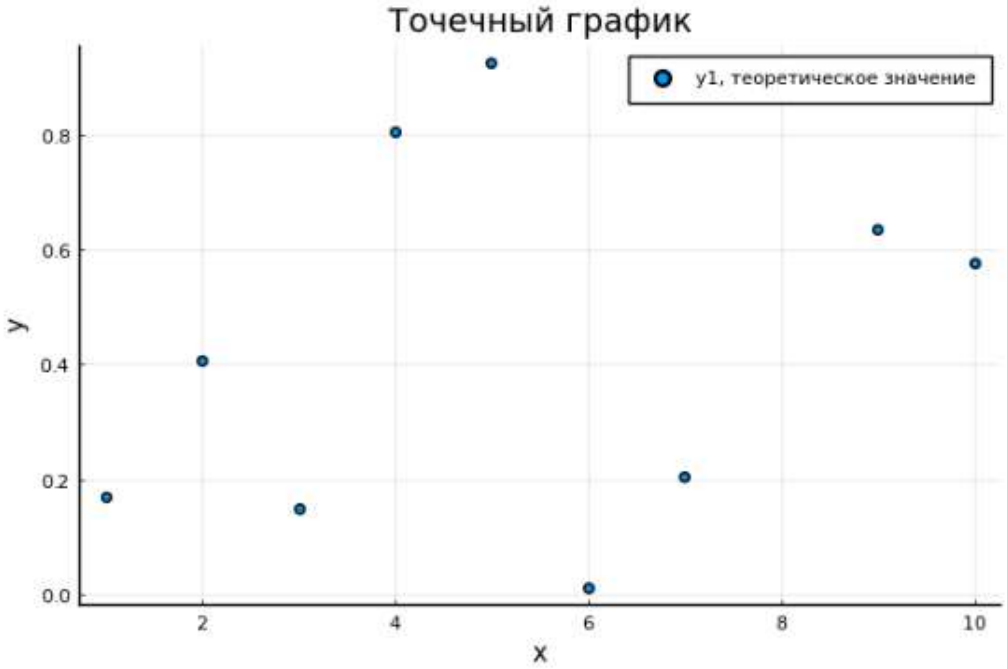


Рисунок 1.7 Точечный график 1

5.2.3.2. Точечный график с кодированием значения размером точки

```
# параметры распределения точек на плоскости:
n = 50
x = rand(n)
y = rand(n)
ms = rand(50) * 30
# параметры построения графика:
scatter(x, y,
    markersize=ms,
    # подписи по осям:
    xlabel = "x",
    ylabel = "y",
    # подпись в легенде, цвет и тип линии:
    label = "y1, теоретическое значение",
    title = "Точечный график")
```

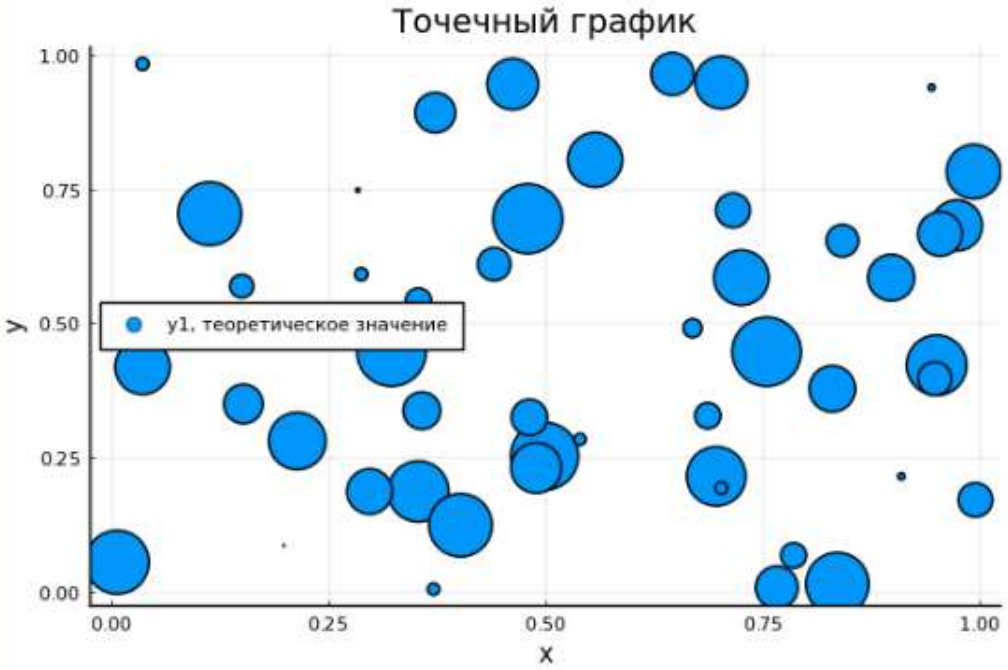


Рисунок 1.8 Точечный график 2

5.2.3.3. мерный точечный график с кодированием значения размером точки

```
# параметры распределения точек в пространстве:
n = 50
x = rand(n)
y = rand(n)
z = rand(n)
ms = rand(50) * 30
# параметры построения графика:
scatter(x, y, z,
    markersize=ms,
    xlabel = "x",
    ylabel = "y",
    zlabel = "z",
    # подпись в легенде, цвет и тип линии:
    label = "sin(x), теоретическое значение",
    title = "Точечный график")
```

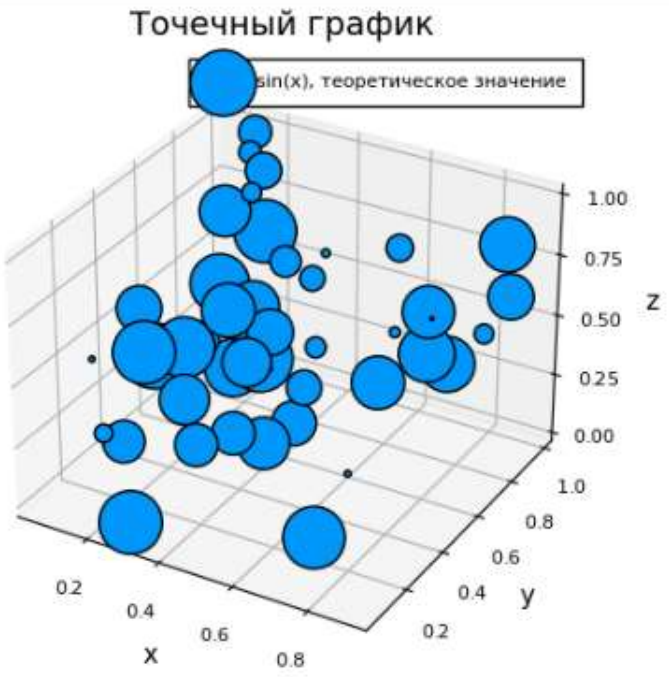
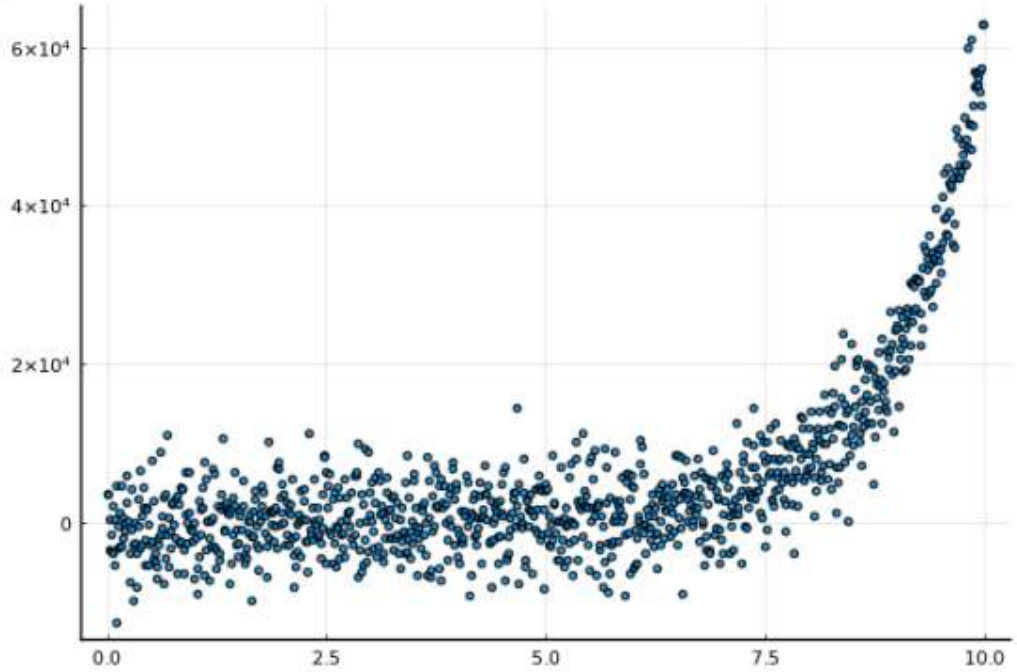


Рисунок 1.9 Точечный график 3

5.2.4. Аппроксимация данных

```
# массив данных от 0 до 10 с шагом 0.01:
x = collect(0:0.01:9.99)
# экспоненциальная функция со случайным сдвигом значений:
y = exp.(ones(1000)+x) + 4000*randn(1000)
# построение графика:
scatter(x,y,markersize=3,alpha=.8,legend=false)
```




```
# определение массива для нахождения коэффициентов полинома:
A = [ones(1000) x x.^2 x.^3 x.^4 x.^5]
# решение матричного уравнения:
c = A\y
# построение полинома:
f1 = c[1]*ones(1000) + c[2]*x + c[3]*x.^2 + c[4]*x.^3 + c[5]*x.^4 + c[6]*x.^5

# построение графика аппроксимирующей функции:
plot!(x, f1, linewidth=3, color=:red)
```

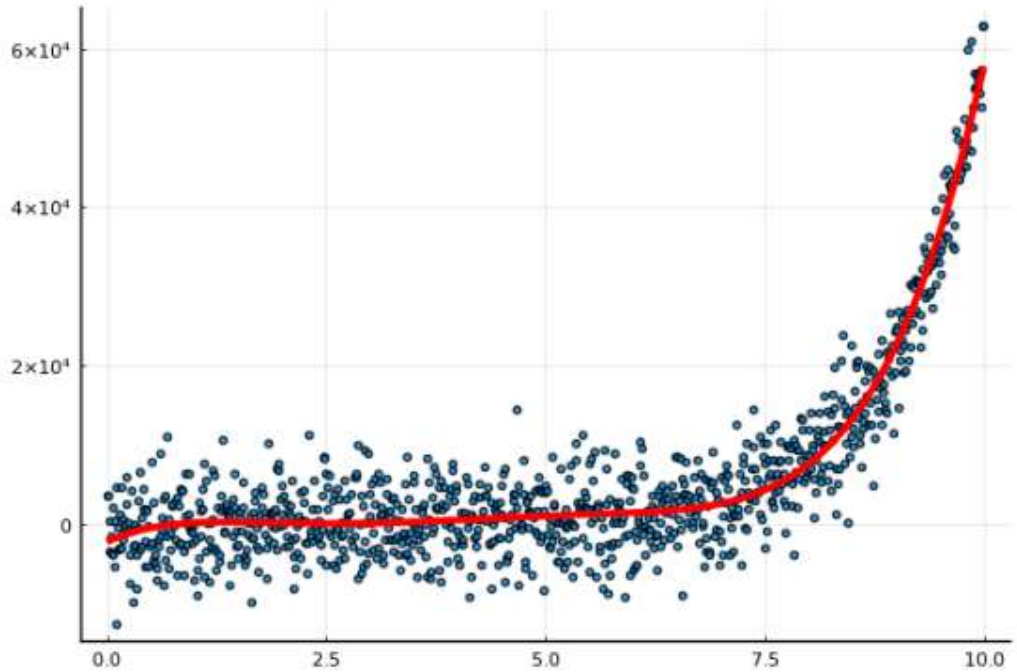
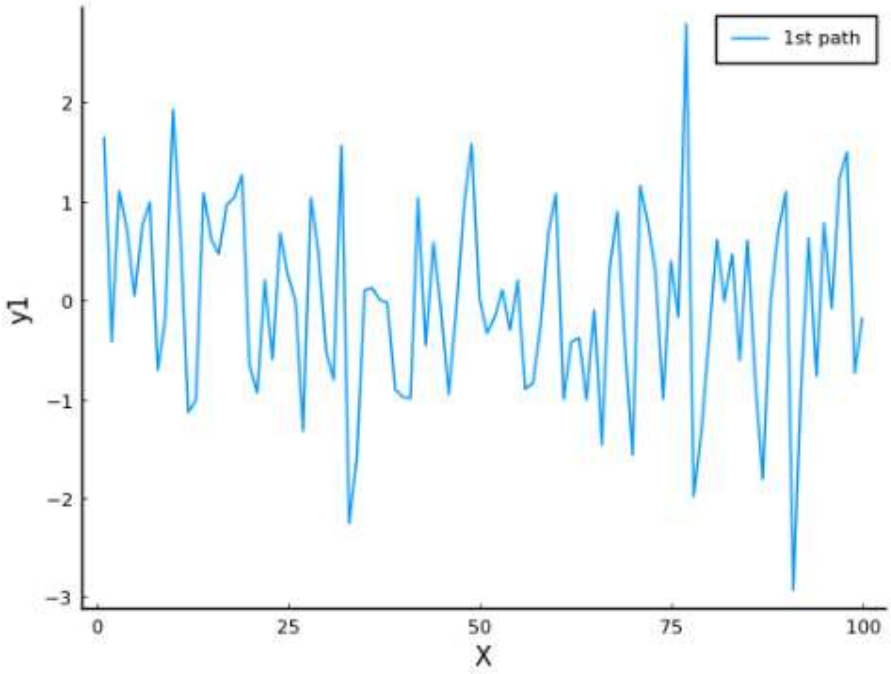


Рисунок 1.10 Аппроксимация данных

5.2.5. Две оси ординат

```
using Plots.PlotMeasures
# пример случайной траектории
# (заданы обозначение траектории, легенда вверху справа, без сетки)
plot(randn(100),
    xlabel = "X",
    ylabel="y1",
    label = "1st path",
    leg=:topright,
    grid = :off,
    right_margin = 20mm
)
```




```
# пример добавления на график второй случайной траектории
# (задано обозначение траектории и её цвет, легенда снизу справа, без сетки)
# задана рамка графика
plot!(twinx(), randn(100)*10,
    c=:red,
    ylabel="y2",
    label = "2nd path",
    leg=:bottomright,
    grid = :off,
    box = :on,
    size=(600, 400)
)
```

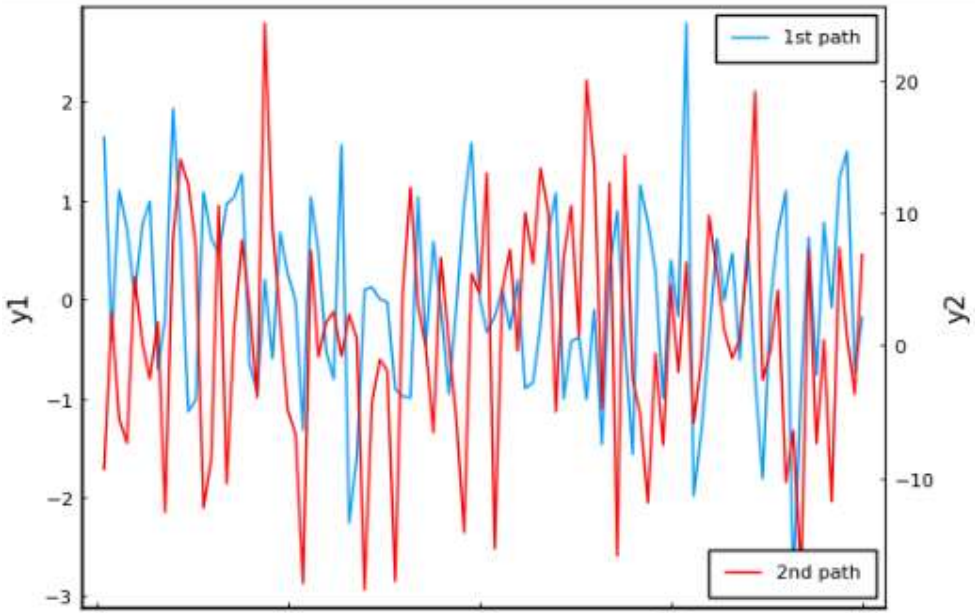


Рисунок 1.11 Две оси ординат

5.2.6. Полярные координаты

```
# функция в полярных координатах:
r(θ) = 1 + cos(θ) * sin(θ)^2
# полярная система координат:
θ = range(0, stop=2π, length=50)
# график функции, заданной в полярных координатах:
plot(θ, r.(θ),
    proj=:polar,
    lims=(0,1.5)
)
```

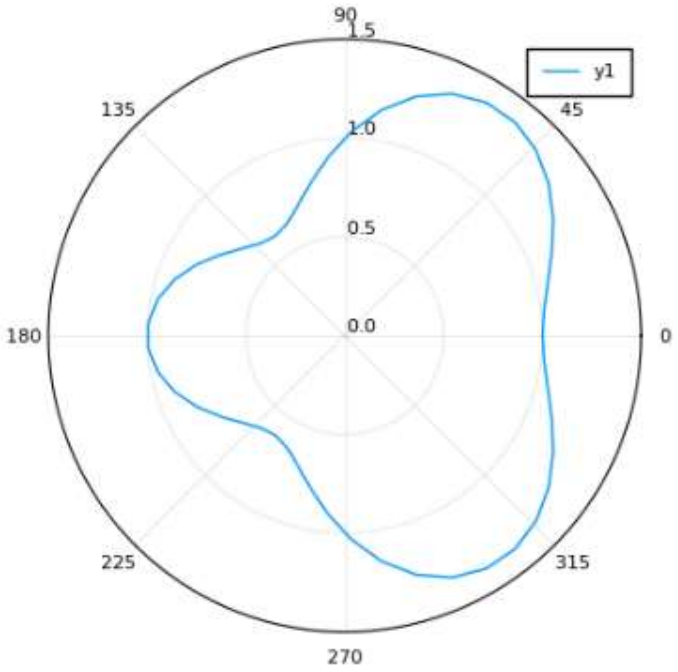


Рисунок 1.12 Полярные координаты

5.2.7. Параметрический график

5.2.7.1. Параметрический график кривой на плоскости

```
# параметрическое уравнение:
x1(t) = sin(t)
y1(t) = sin(2t)
# построение графика:
plot(x1, y1, 0, 2π, leg=false, fill=(0,:orange))
```

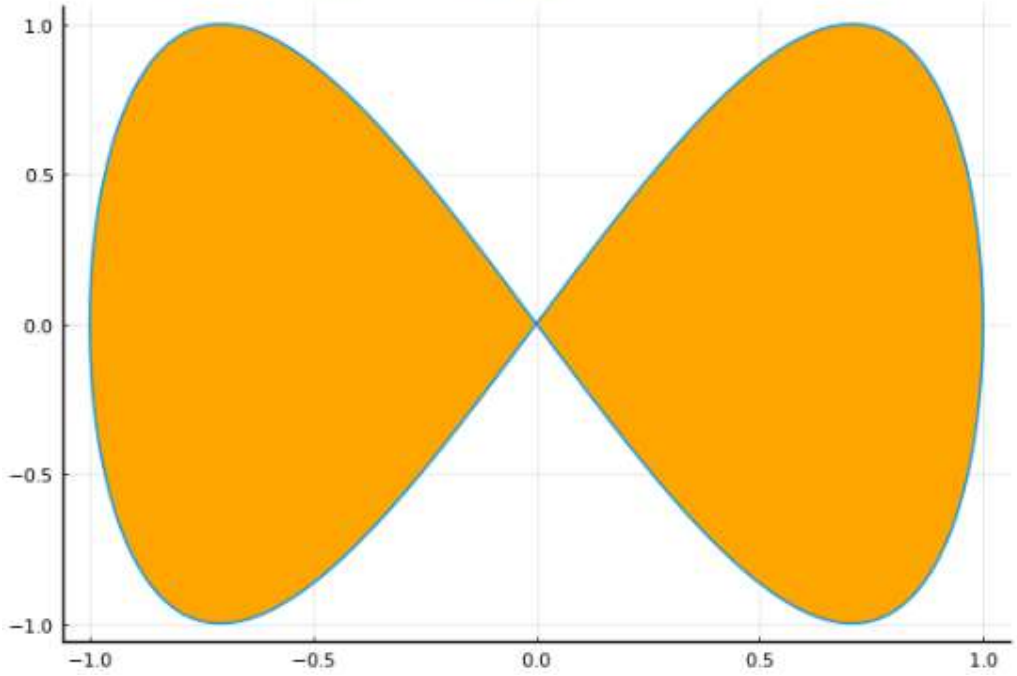


Рисунок 1.13 Параметрический график 1

5.2.7.2. Параметрический график кривой в пространстве

```
# параметрическое уравнение
t = range(0, stop=10, length=1000)
x = cos.(t)
y = sin.(t)
z = sin.(5t)
# построение графика:
plot(x, y, z)
```

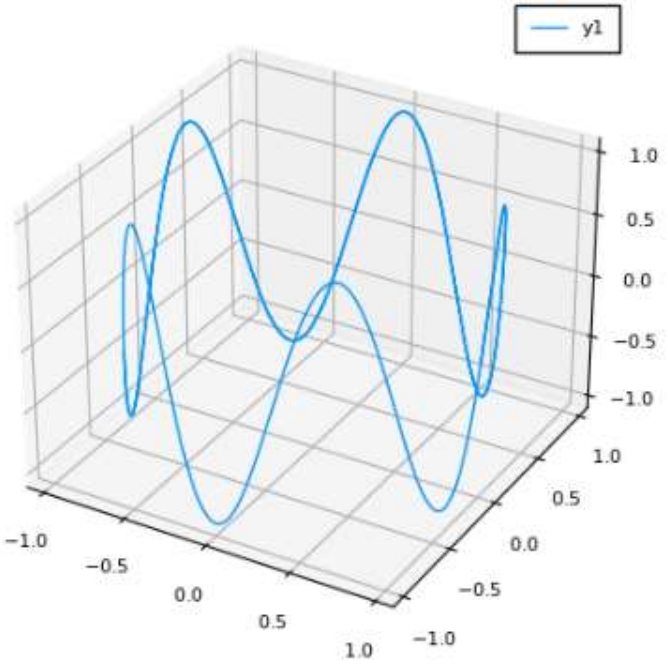
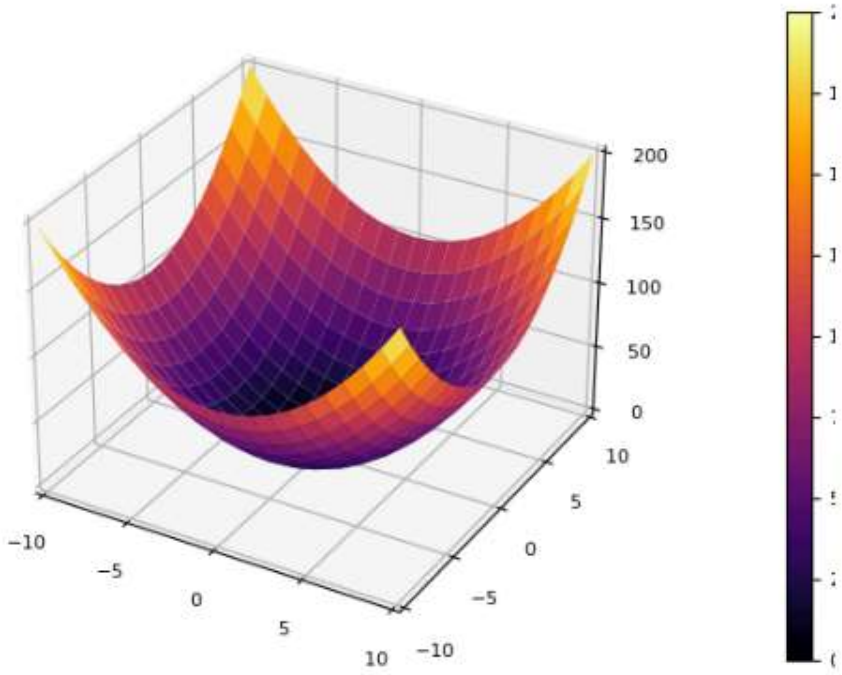


Рисунок 1.14 Параметрический график 2

5.2.8. График поверхности

```
# построение графика поверхности:
f(x,y) = x^2 + y^2
x = -10:10
y = x
surface(x, y, f)
```




```
# построение графика поверхности:
f(x,y) = x^2 + y^2
x = -10:10
y = x
plot(x, y, f,
    linetype=:wireframe
)
```

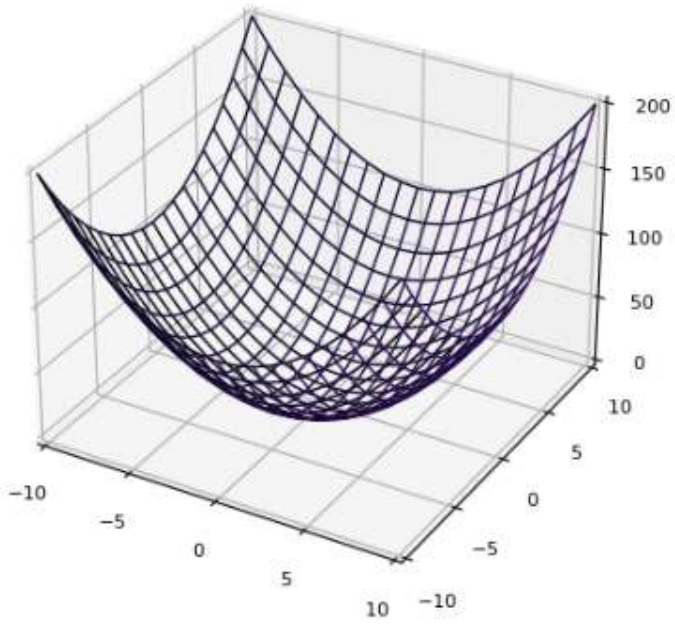
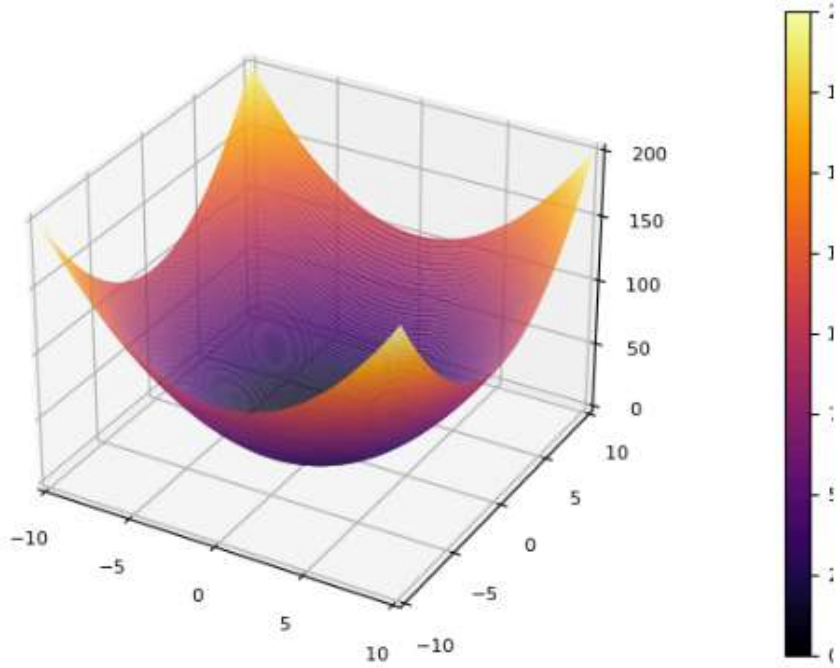


Рисунок 1.15 График поверхности 1

```
f2(x,y) = x^2 + y^2
x = -10:0.1:10
y = x
plot(x, y, f2,
linetype = :surface
)
```




```
x = range(-2,stop=2,length=100)
y = range(sqrt(2),stop=2,length=100)
f3(x,y) = x*y-x-y+1
plot(x, y, f3,
    linetype = :surface,
    c=cgrad([:red,:blue]),
    camera=(-30,30),
)
```

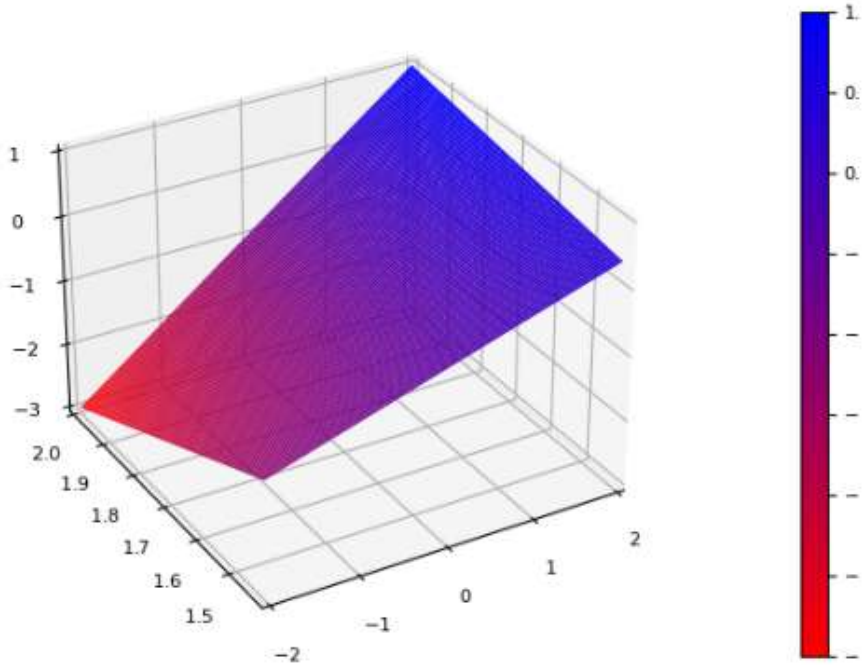
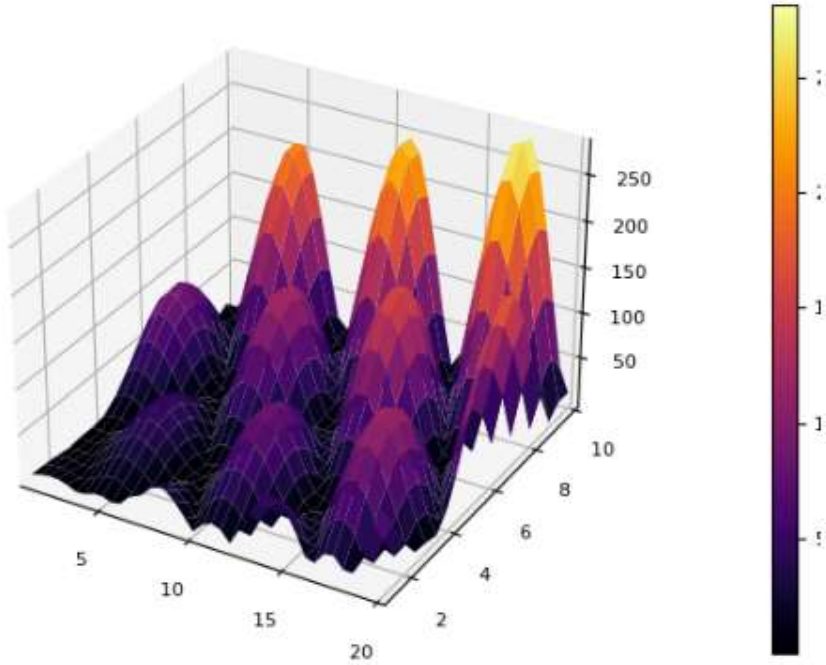


Рисунок 1.16 График поверхности 2

5.2.9. Линии уровня

```
x = 1:0.5:20
y = 1:0.5:10
g(x, y) = (3x + y ^ 2) * abs(sin(x) + cos(y))
plot(x, y, g,
    linetype = :surface,
)
```




```
contour(x, y, g)
```

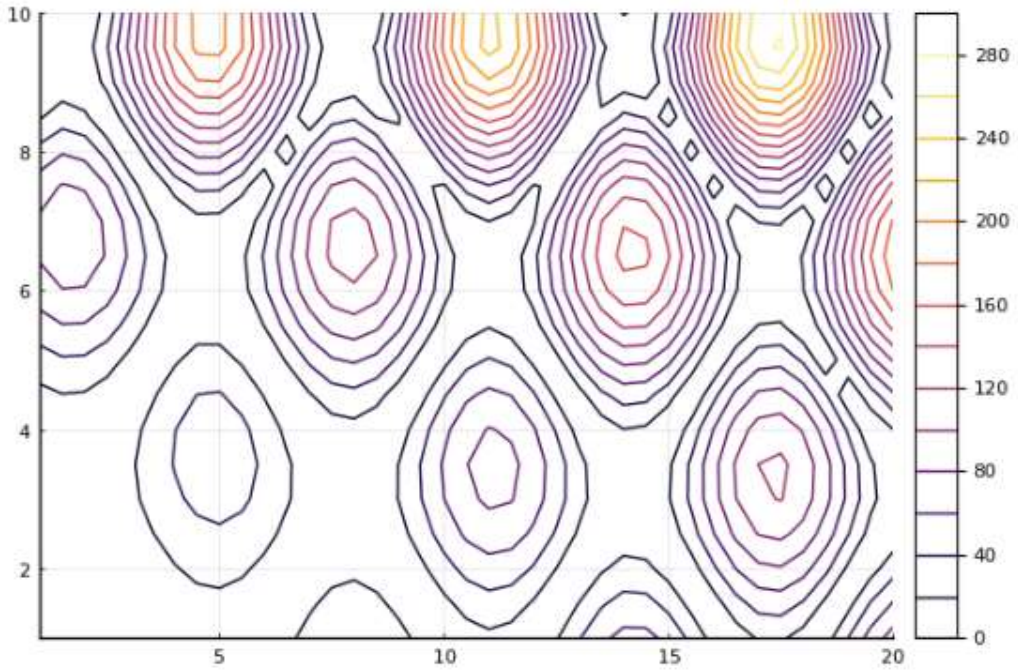


Рисунок 1.17 Линии уровня 1

```
p = contour(x, y, g,
fill=true)
plot(p)
```

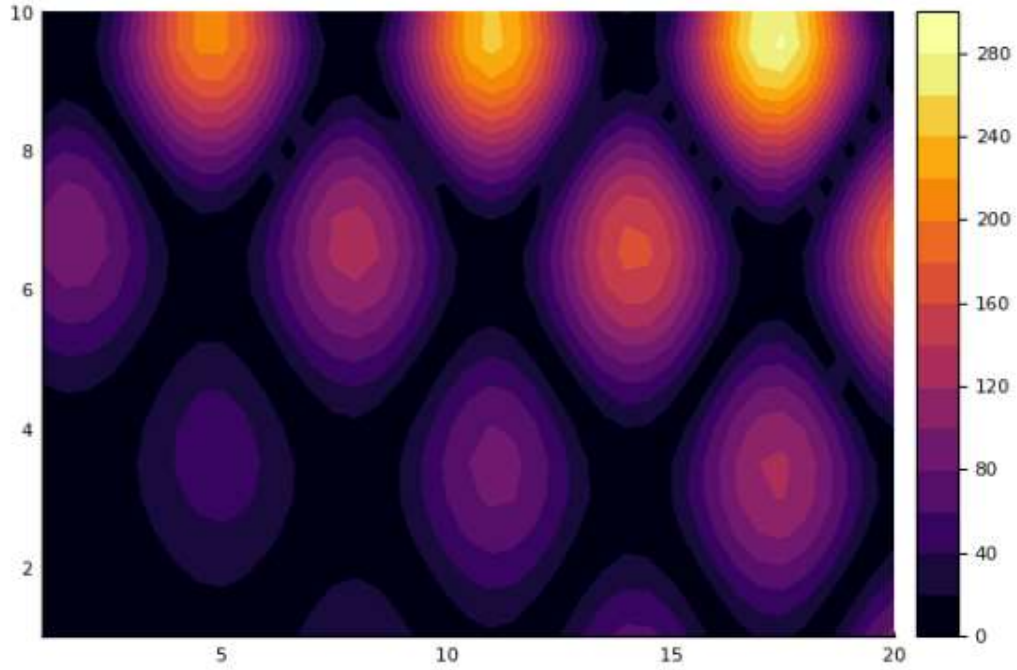
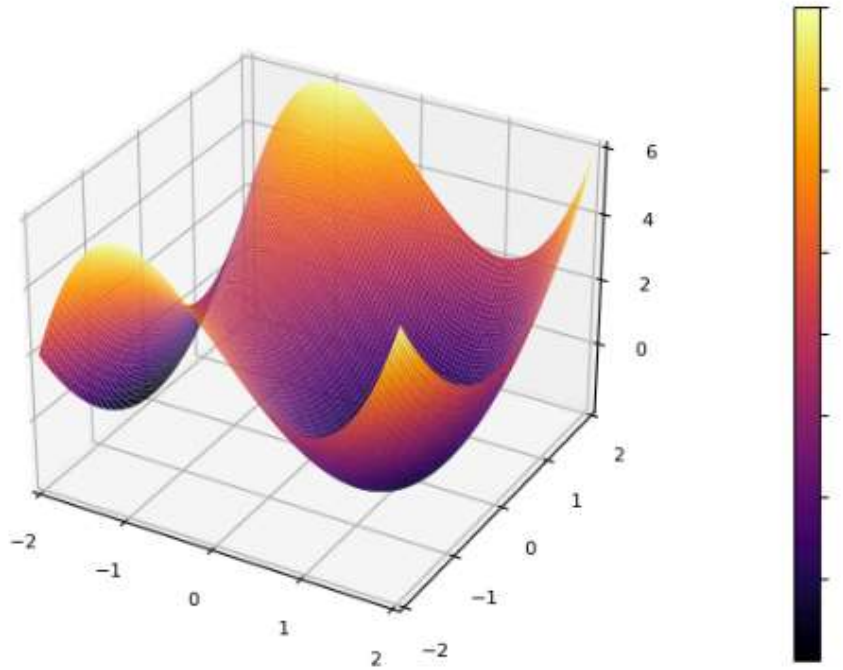


Рисунок 1.18 Линии уровня 2

5.2.10. Векторные поля

```
# определение переменных:
X0 = range(-2, stop=2, length=100)
Y0 = range(-2, stop=2, length=100)
# определение функции:
h(x, y) = x^3 - 3x + y^2
# построение поверхности:
plot(X0, Y0, h,
    linetype = :surface
)
```




```
# построение линий уровня:
contour(X0, Y0, h)
```

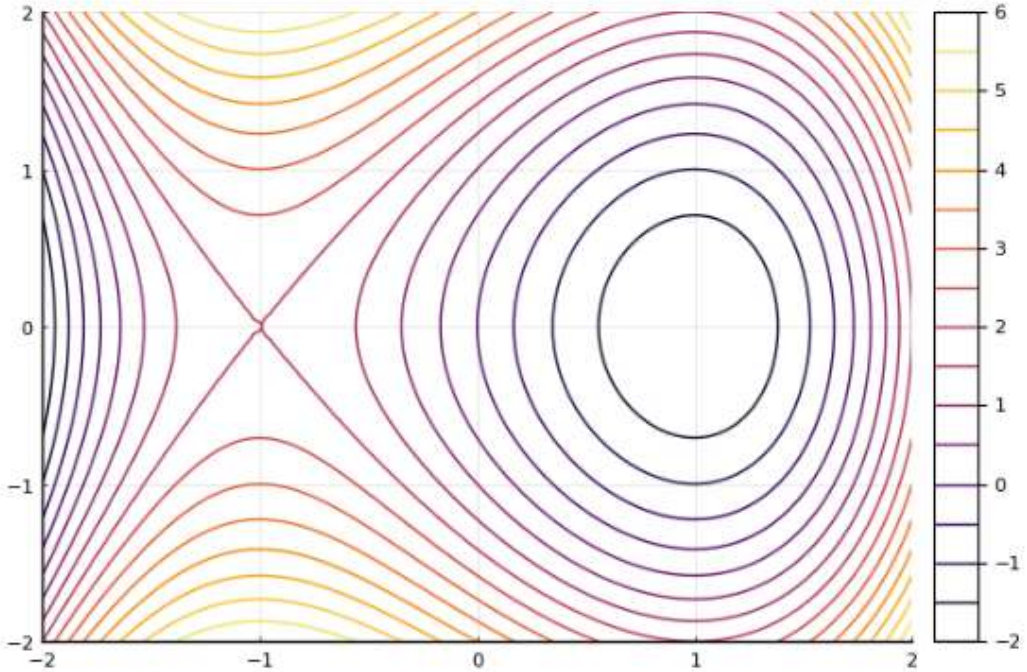


Рисунок 1.19 Векторные поля 1

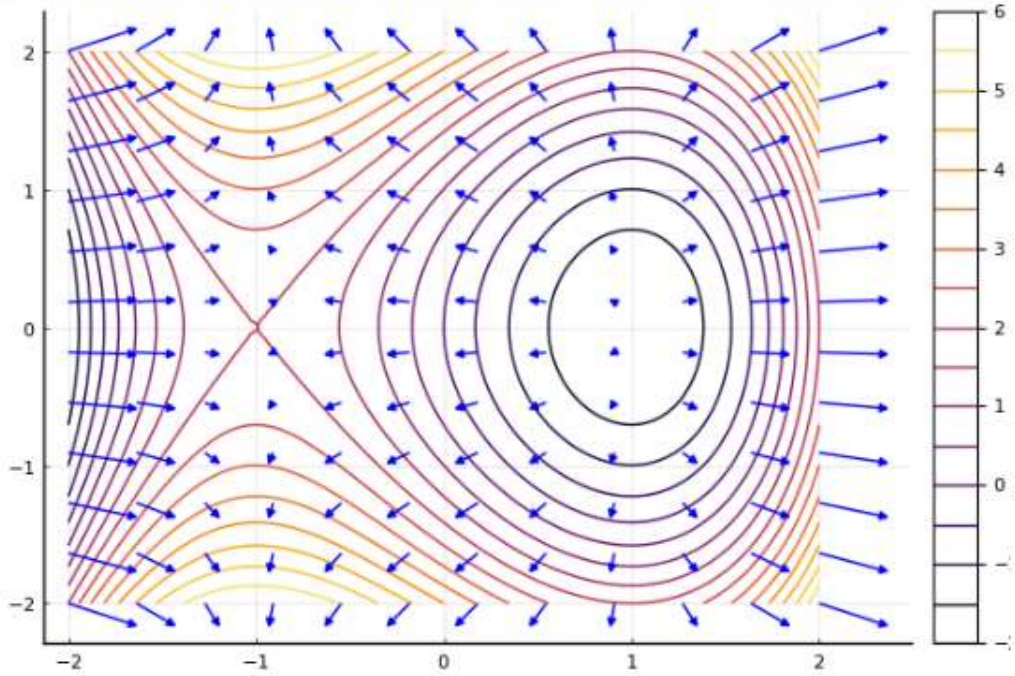
```
# градиент:
xs = range(-2, stop=2, length=12)
ys = range(-2, stop=2, length=12)
```

```
-2.0:0.36363636363636365:2.0
```

```
# производная от исходной функции:
dh(x, y) = [3x^2-3; 2y]/25
```

```
dh (generic function with 1 method)
```

```
xxs = [x for x in xs for y in ys]
yys = [y for x in xs for y in ys]

quiver!(xxs, yys, quiver=dh, c=:blue)
```




```
# коррекция области видимости графика:
xlims!(-2, 2)
ylims!(-2, 2)
quiver!(xxs, yys, quiver=dh, c=:blue)
```

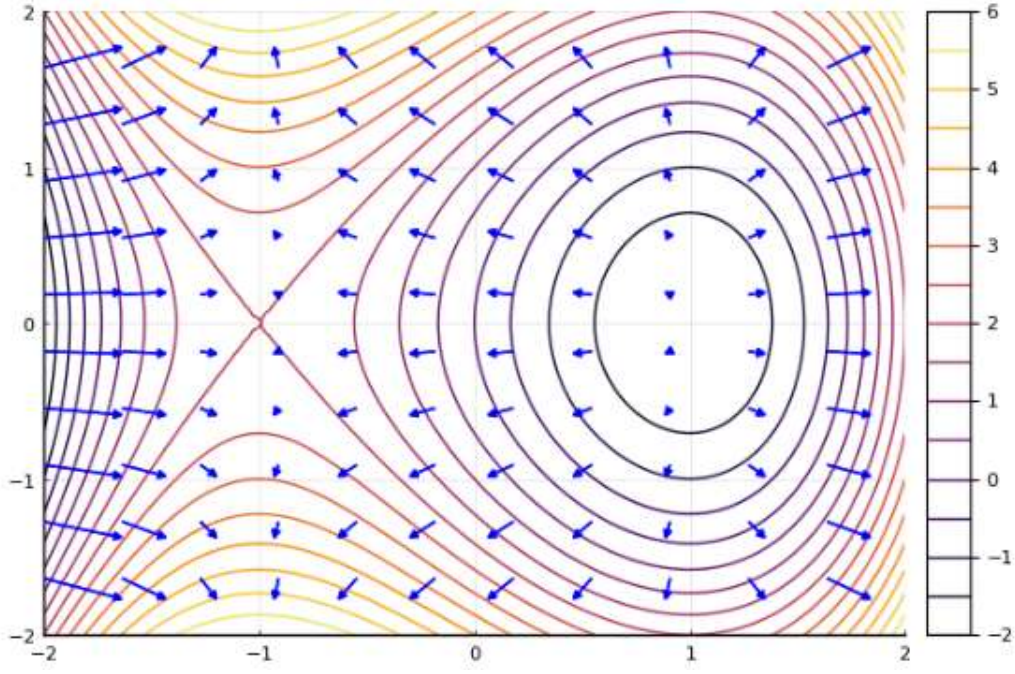
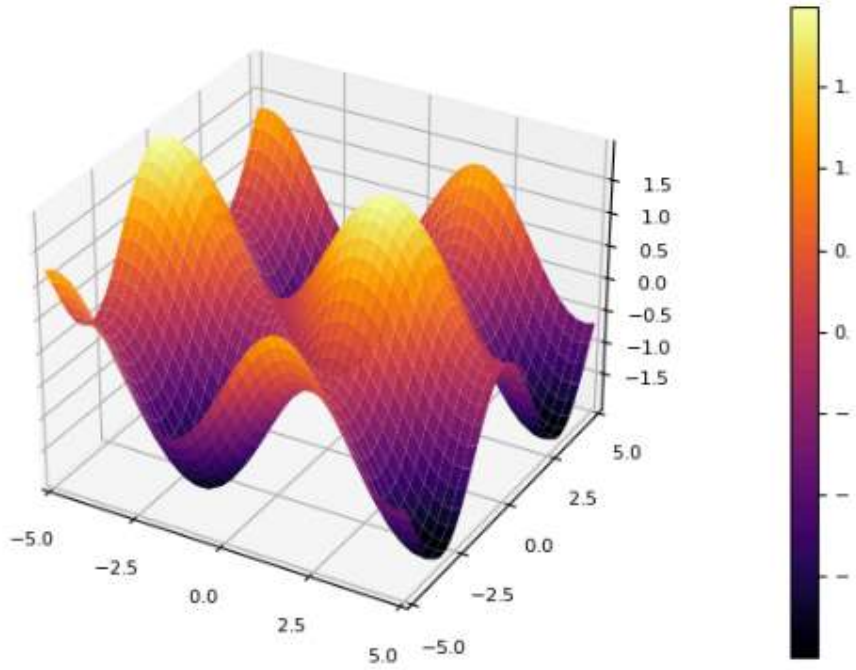


Рисунок 1.20 Векторные поля 2

5.2.11. Анимация

5.2.11.1. Gif-анимация

```
pyplot()
# построение поверхности:
i = 0
X = Y = range(-5,stop=5,length=40)
surface(X, Y, (x,y) -> sin(x+10sin(i))+cos(y))
```




```
# анимация:
X = Y = range(-5, stop=5, length=40)
@gif for i in range(0, stop=2π, length=100)
surface(X, Y, (x, y) -> sin(x + 10sin(i)) + cos(y))
end
```

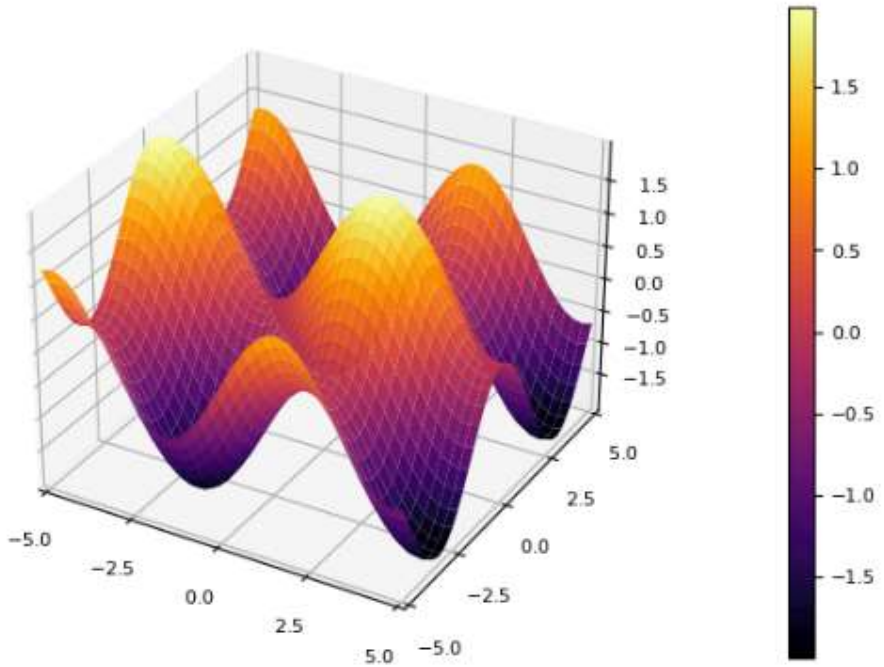


Рисунок 1.21 Векторные поля 3

```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/tmp.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

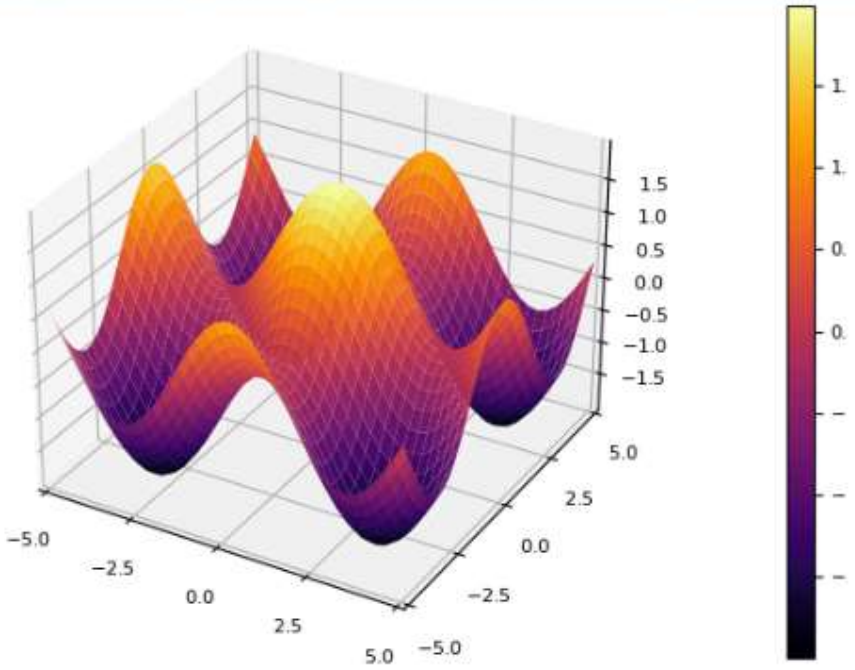


Рисунок 1.22 Векторные поля 4

5.2.11.2. Гипоциклоида

```
# радиус малой окружности:
r1 = 1
# коэффициент для построения большой окружности:
k = 3
# число отсчётов:
n = 100
# массив значений угла θ:
# theta from 0 to 2pi ( + a little extra)
θ = collect(0:2*π/100:2*π+2*π/100)
# массивы значений координат:
X = r1*k*cos.(θ)
Y = r1*k*sin.(θ)
# задаём оси координат:
plt = plot(5, xlim=(-4,4), ylim=(-4,4), c=:red, aspect_ratio=1, legend=false, framestyle=:origin)
# большая окружность:
plot!(plt, X,Y, c=:blue, legend=false)
```

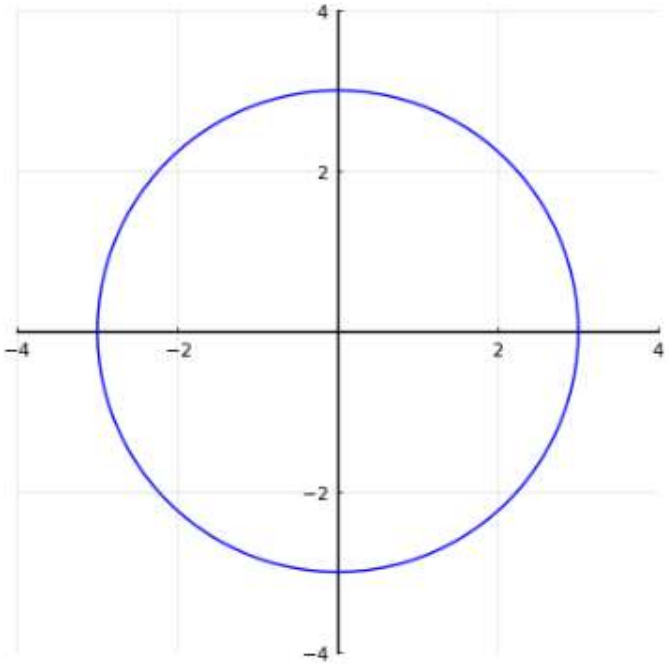
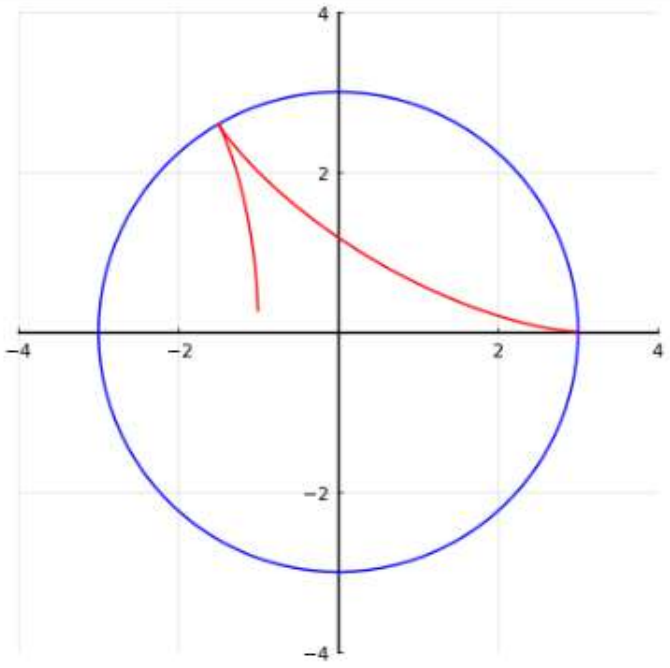


Рисунок 1.23 Векторные поля 5

```
i = 50
t = θ[1:i]
# гипоциклоида:
x = r1*(k-1)*cos.(t) + r1*cos.((k-1)*t)
y = r1*(k-1)*sin.(t) - r1*sin.((k-1)*t)
plot!(x,y, c=:red)
```




```
# малая окружность:
xc = r1*(k-1)*cos(t[end]) .+ r1*cos.(θ)
yc = r1*(k-1)*sin(t[end]) .+ r1*sin.(θ)
plot!(xc,yc,c=:black)
```

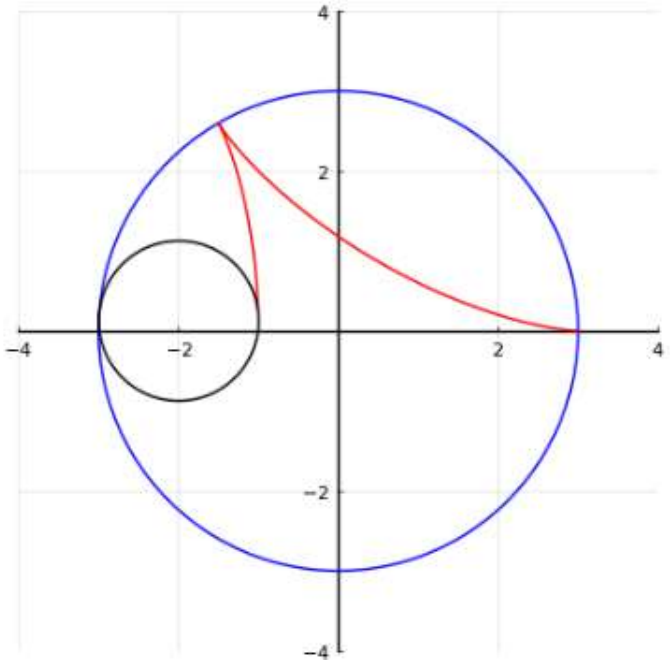


Рисунок 1.24 Векторные поля 6

```
# радиус малой окружности:
xl = transpose([r1*(k-1)*cos(t[end]) x[end]])
yl = transpose([r1*(k-1)*sin(t[end]) y[end]])
plot!(xl, yl, markershape=:circle, markersize=4, c=:black)
scatter!([x[end]], [y[end]], c=:red, markerstrokecolor=:red)
```

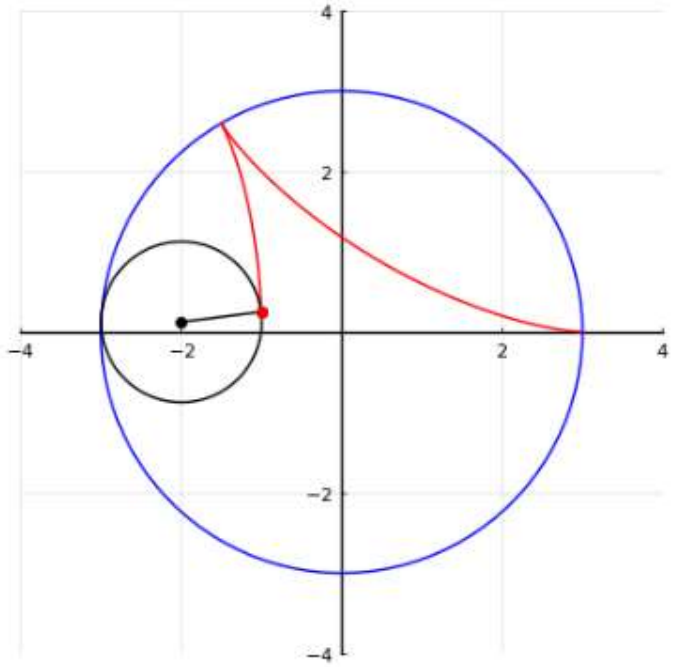


Рисунок 1.25 Векторные поля 7

```
anim = @animate for i in 1:n
# задаём оси координат:
plt=plot(5, xlim=(-4,4), ylim=(-4,4), c=:red, aspect_ratio=1, legend=false, framestyle=:origin)
# большая окружность:
plot!(plt, X, Y, c=:blue, legend=false)
t = θ[1:i]

# гипоциклоида:
x = r1*(k-1)*cos.(t) + r1*cos.((k-1)*t)
y = r1*(k-1)*sin.(t) - r1*sin.((k-1)*t)
plot!(x,y, c=:red)

# малая окружность:
xc = r1*(k-1)*cos(t[end]) .+ r1*cos.(θ)
yc = r1*(k-1)*sin(t[end]) .+ r1*sin.(θ)
plot!(xc, yc, c=:black)

# радиус малой окружности:
xl = transpose([r1*(k-1)*cos(t[end]) x[end]])
yl = transpose([r1*(k-1)*sin(t[end]) y[end]])
plot!(xl,yl,markershape=:circle, markersize=4, c=:black)
scatter!([x[end]], [y[end]], c=:red, markerstrokecolor=:red)
end

gif(anim,"hypocycloid.gif")
```

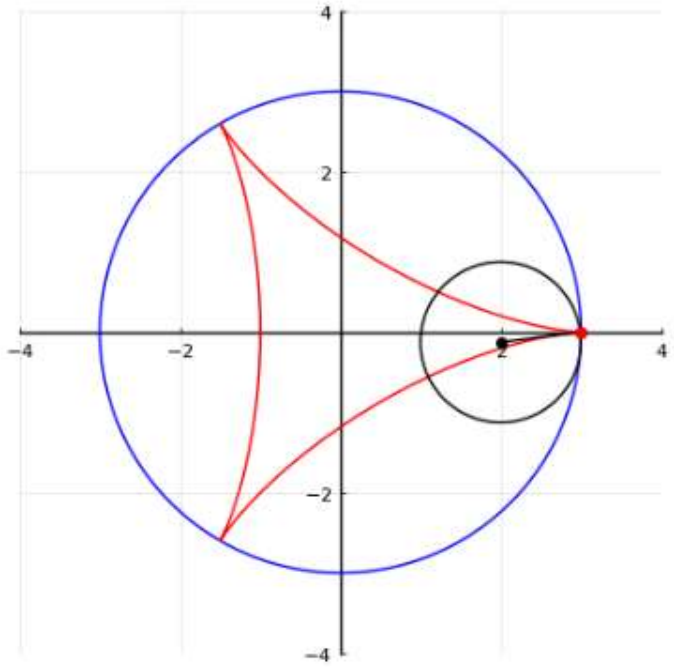


Рисунок 1.26 Векторные поля 8

```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/hypocycloid.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

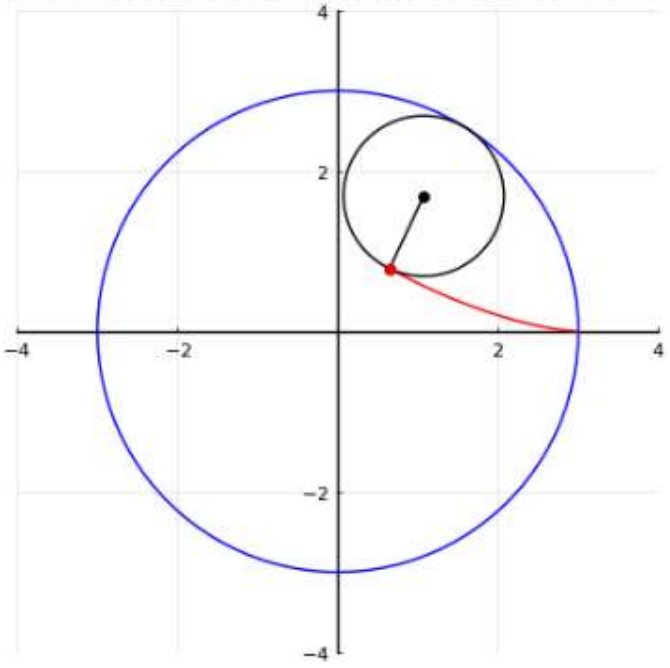
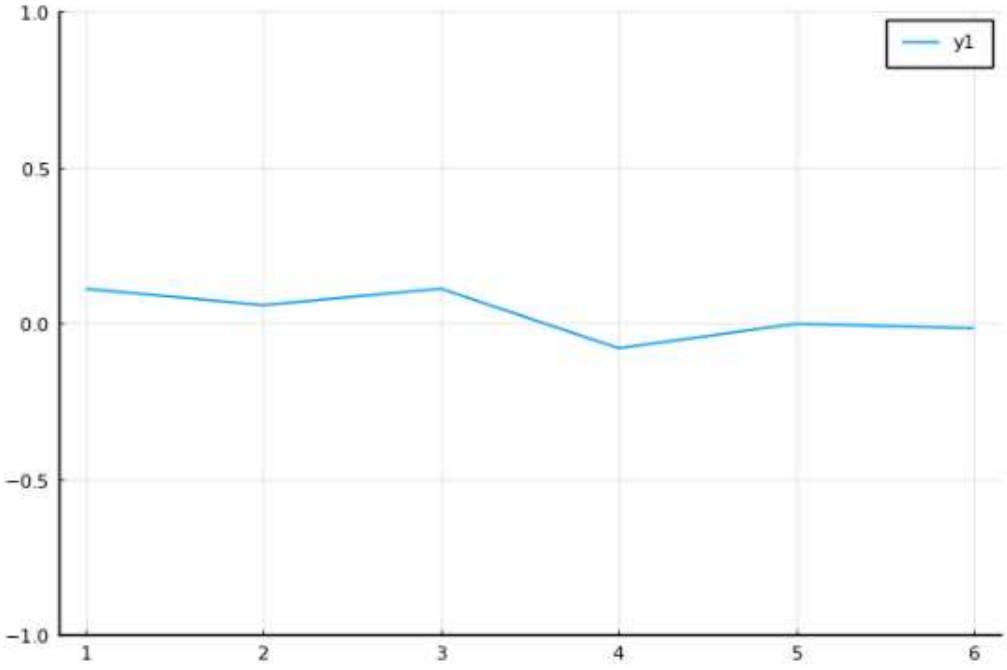


Рисунок 1.27 Векторные поля 9

5.2.12. Errorbars

```
using Statistics
sds = [1, 1/2, 1/4, 1/8, 1/16, 1/32]
n = 10
y = [mean(sd*randn(n)) for sd in sds]
errs = 1.96 * sds / sqrt(n)
plot(y,
    ylims = (-1,1),
)
```




```
plot(y,
ylims = (-1,1),
err = errs
)
```

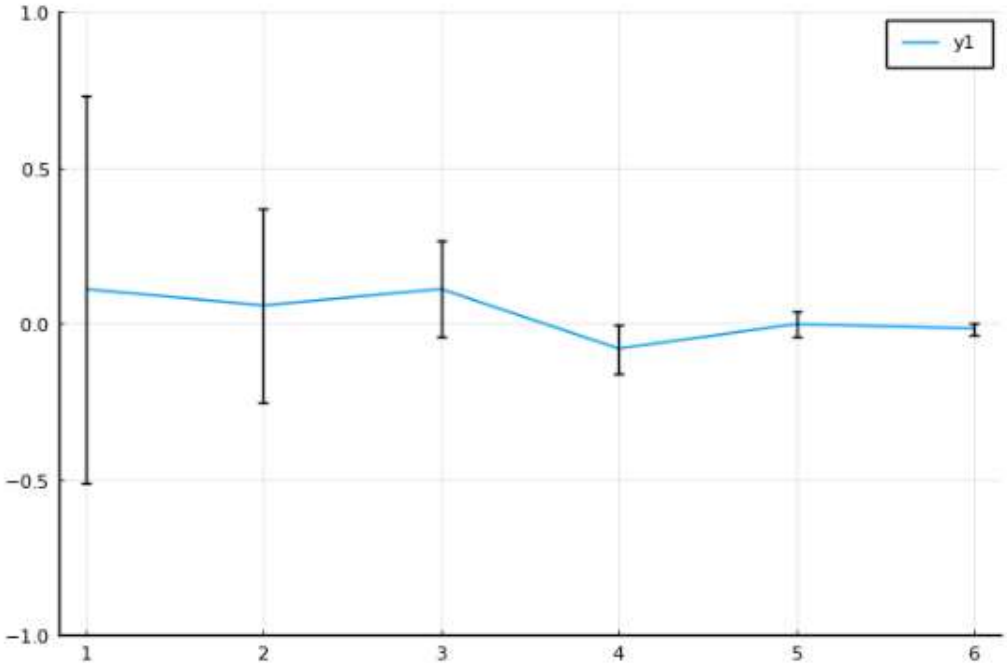
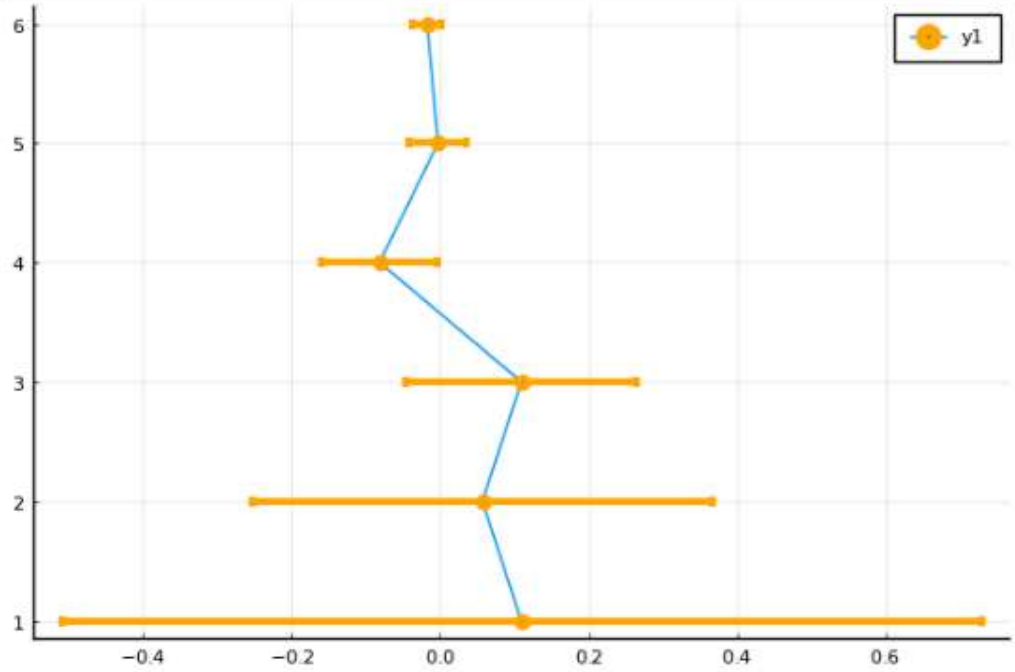


Рисунок 1.28 Errorbars 1

```
plot(y, 1:length(y),
    xerr = errs,
    marker = stroke(3,:orange)
)
```




```
plot(y,
    ribbon=errs,
    fill=:cyan
)
```

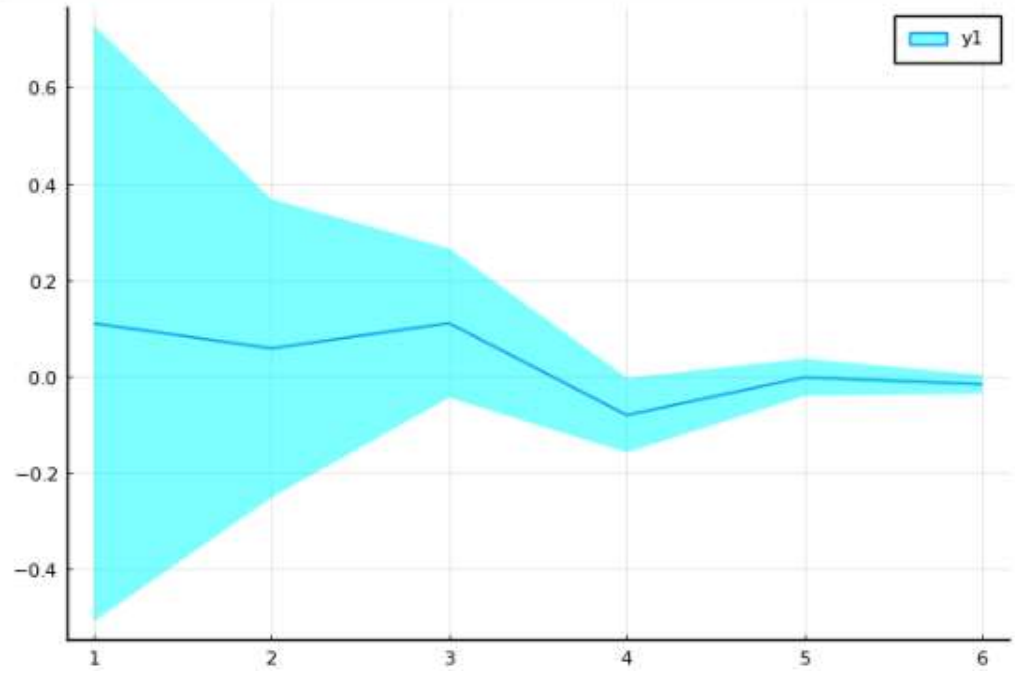
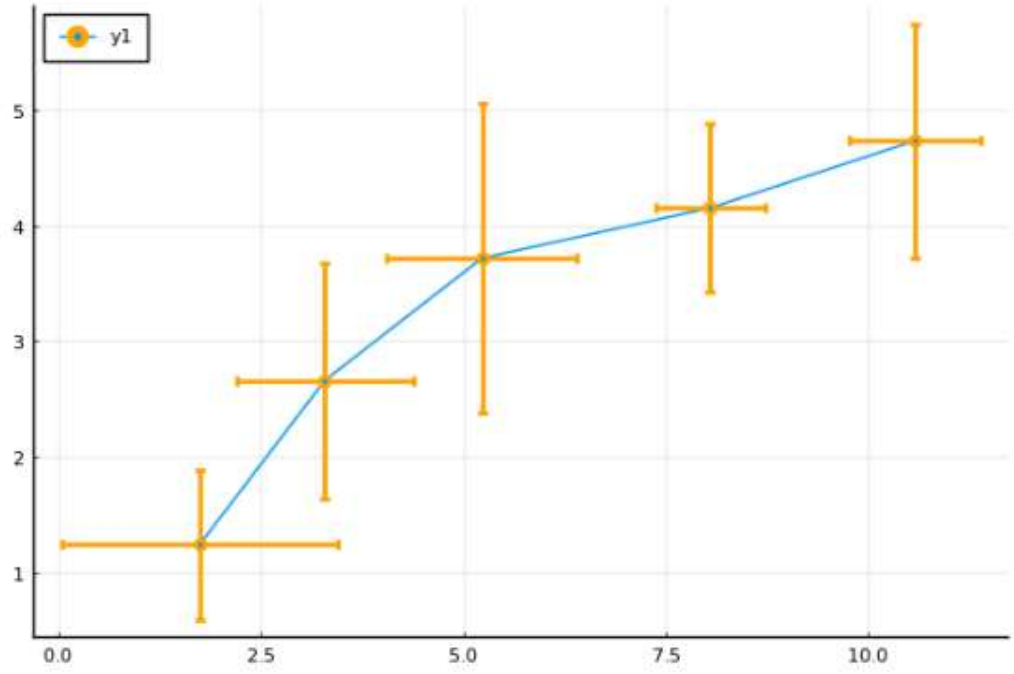


Рисунок 1.29 Errorbars 2

```
n = 10
x = [(rand()+1) .* randn(n) .+ 2i for i in 1:5]
y = [(rand()+1) .* randn(n) .+ i for i in 1:5]
f(v) = 1.96std(v) / sqrt(n)
xerr = map(f, x)
yerr = map(f, y)
x = map(mean, x)
y = map(mean, y)
plot(x, y,
    xerr = xerr,
    yerr = yerr,
    marker = stroke(2, :orange)
)
```




```
plot(x, y,
    xerr = (0.5xerr,2xerr),
    yerr = (0.5yerr,2yerr),
    marker = stroke(2, :orange)
)
```

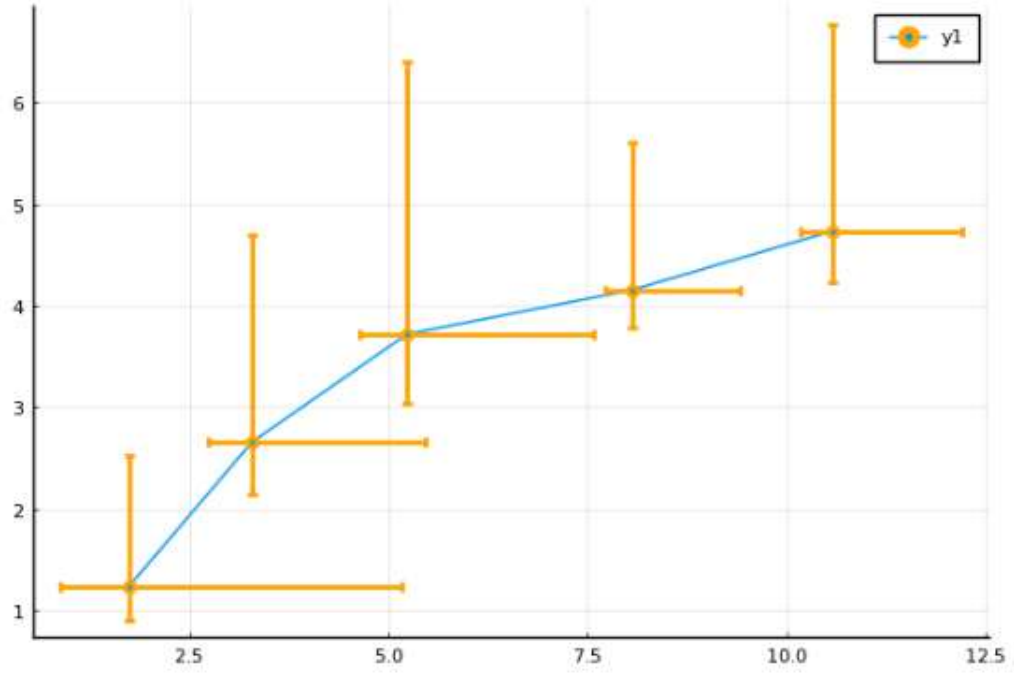


Рисунок 1.30 Errorbars 3

5.2.13. Использование пакета Distributions

```
import Pkg
Pkg.add("Distributions")
using Distributions
```

```
  Resolving package versions...
No Changes to `~/.julia/environments/v1.5/Project.toml`
No Changes to `~/.julia/environments/v1.5/Manifest.toml`
```

```
pyplot()
ages = rand(15:55, 1000)
histogram(ages)
```

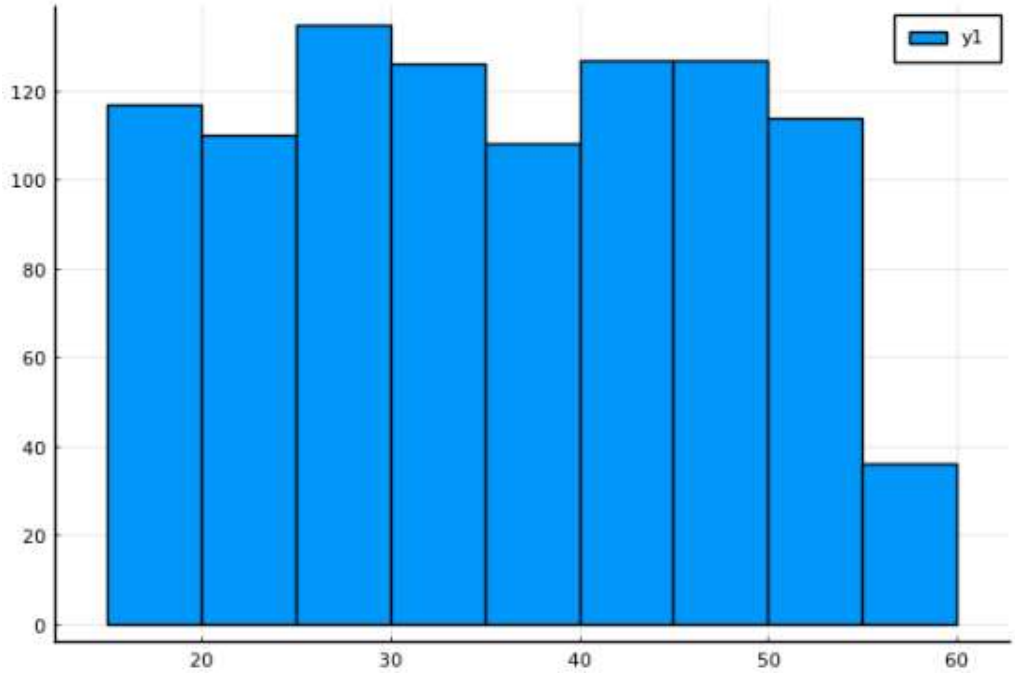
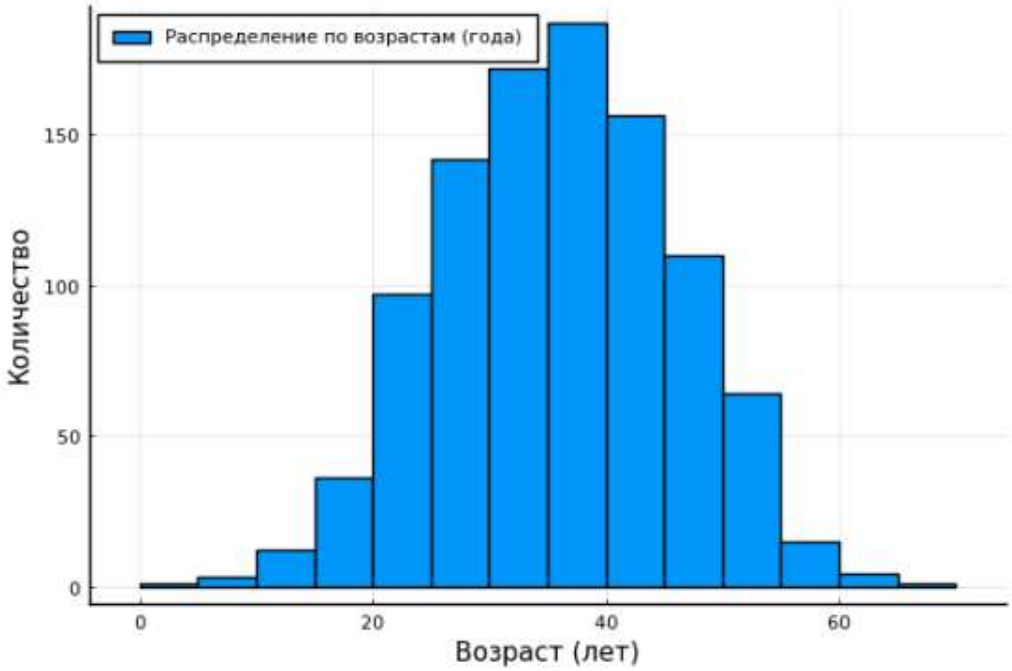


Рисунок 1.31 Использование пакета Distributions 1

```
d = Normal(35.0, 10.0)
ages = rand(d, 1000)
histogram(
    ages,
    label="Распределение по возрастам (года)",
    xlabel = "Возраст (лет)",
    ylabel= "Количество"
)
```



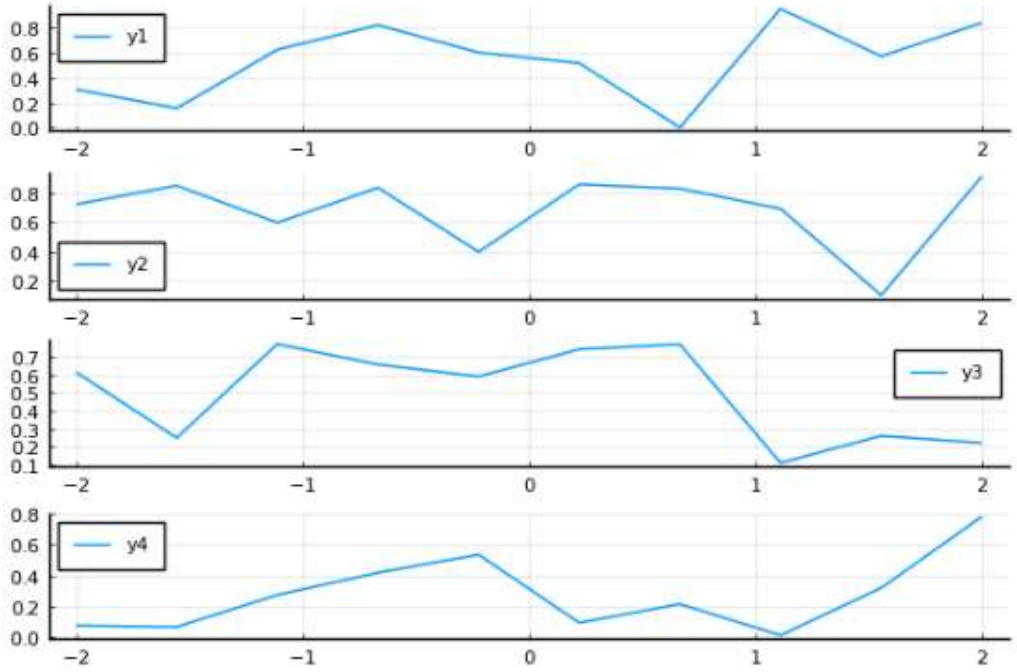

```
plotly()
d1=Normal(10.0,5.0);
d2=Normal(35.0,10.0);
d3=Normal(60.0,5.0);
N=1000;
ages = (Float64)[];
ages = append!(ages,rand(d1,Int64(ceil(N/2))));
ages = append!(ages,rand(d2,N));
ages = append!(ages,rand(d3,Int64(ceil(N/3))));
histogram(
    ages,
    bins=50,
    label="Распределение по возрастам (года)",
    xlabel = "Возраст (лет)",
    ylabel= "Количество",
    title = "Распределение по возрастам (года)"
)
```

Возраст (лет)

Рисунок 1.32 Использование пакета Distributions 2

5.2.14. Подграфики

```
# подгружаем pyplot():
pyplot()
# построение серии графиков:
x=range(-2,2,length=10)
y = rand(10,4)
plot(x,y,
    layout=(4,1)
)
```




```
plot(x,y,
layout=4
)
```

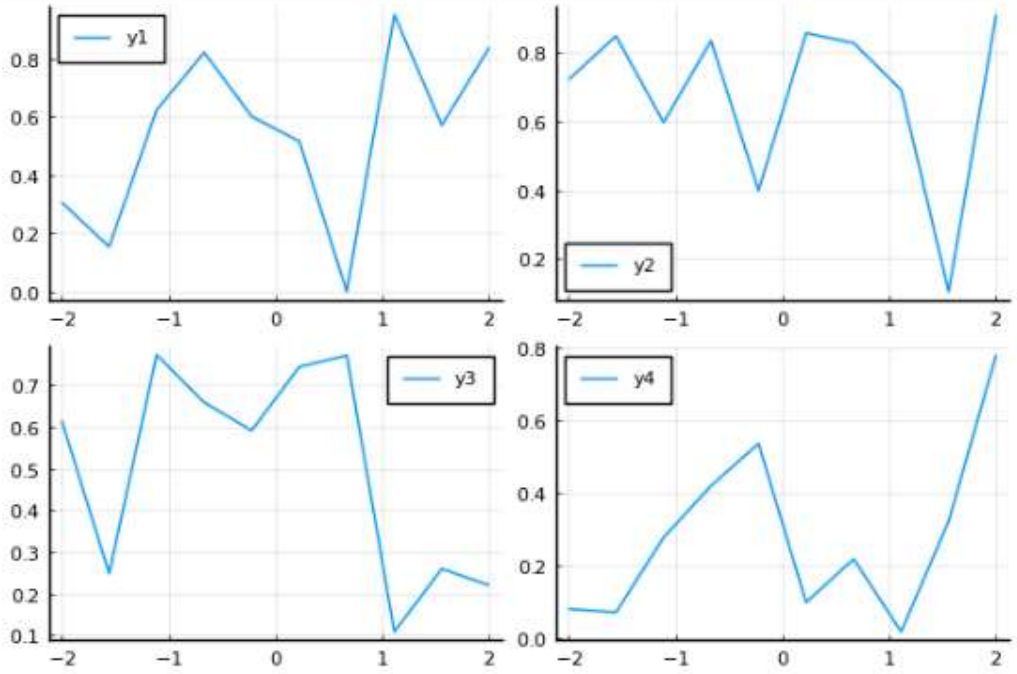
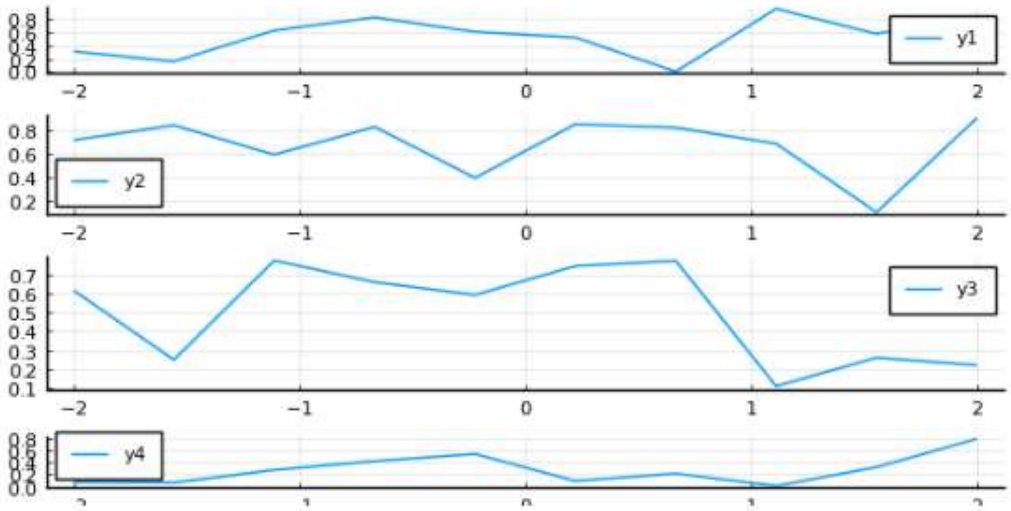
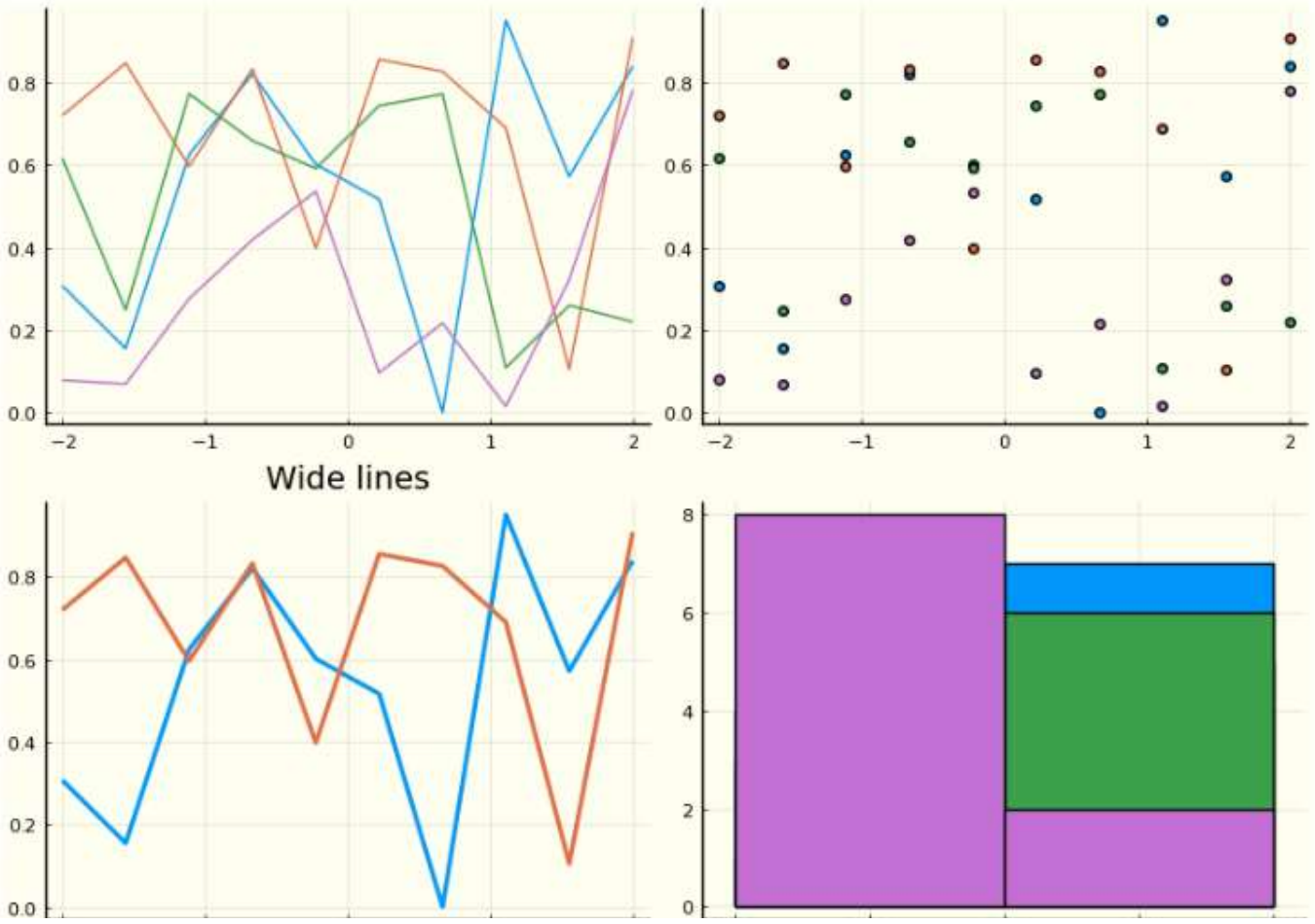


Рисунок 1.33 Подграфики 1

```
plot(x,y,
    size=(600,300),
    layout = grid(4,1,heights=[0.2,0.3,0.4,0.15])
)
```




```
# график в виде линий:
p1 = plot(x,y)
# график в виде точек:
p2 = scatter(x,y)
# график в виде линий с оформлением:
p3 = plot(x,y[:,1:2],xlabel="Labelled plot of two columns",lw=2,title="Wide lines")
# 4 гистограммы:
p4 = histogram(x,y)

plot(
    p1,p2,p3,p4,
    layout=(2,2),
    legend=false,
    size=(800,600),
    background_color = :ivory
)
```



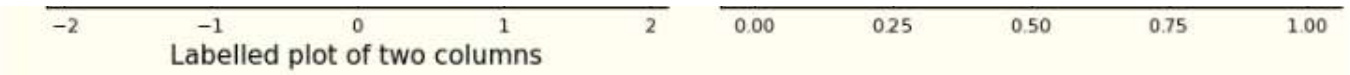
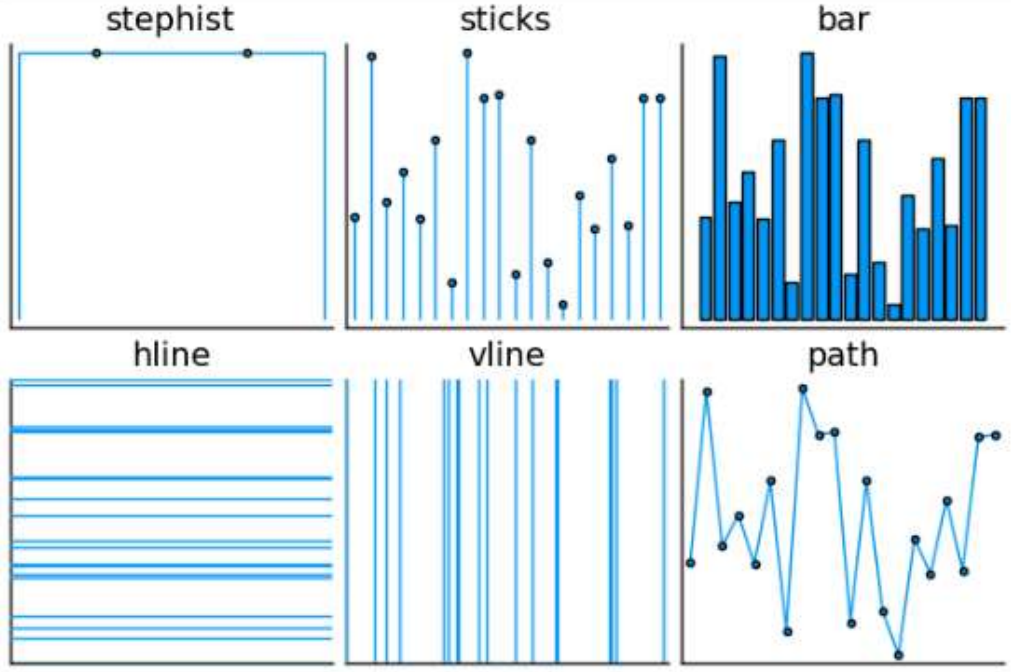


Рисунок 1.34 Подграфики 2

```
seriestypes = [:stephist, :sticks, :bar, :hline, :vline, :path]
titles =["stephist" "sticks" "bar" "hline" "vline" "path"]
plot(rand(20,1), st = seriestypes,
    layout = (2,3),
    ticks=nothing,
    legend=false,
    title=titles,
    m=3
)
```




```
l = @layout [ a{0.3w} [grid(3,3)
b{0.2h} ]]
plot(
    rand(10,11),
    layout = l, legend = false, seriestype = [:bar :scatter :path],
    title = ["($i)" for j = 1:1, i=1:11], titleloc = :right, titlefont = font(8)
)
```

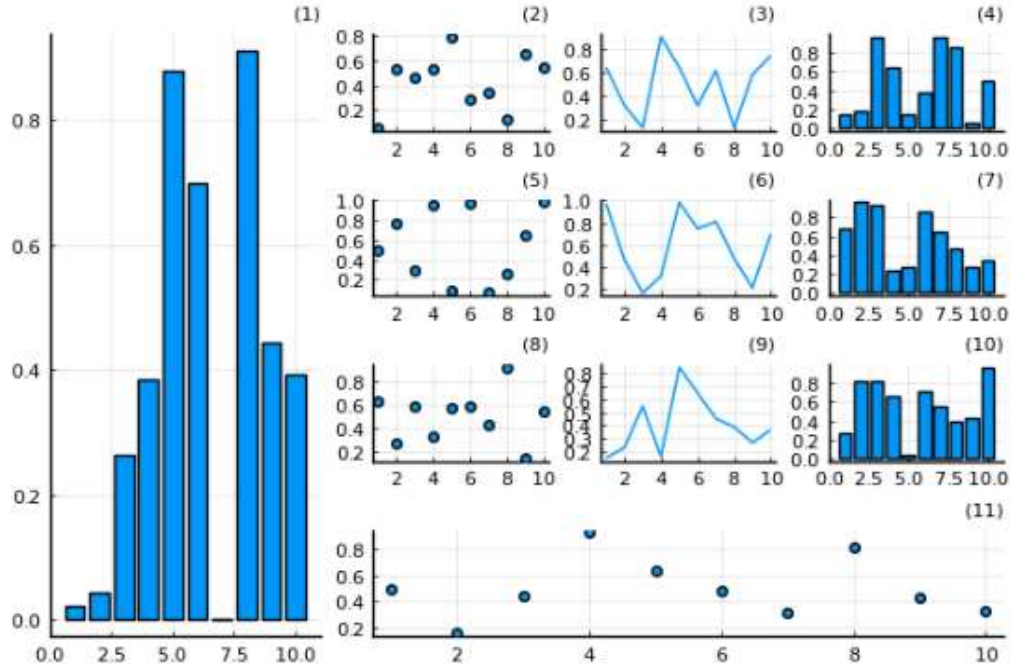


Рисунок 1.35 Подграфики 3

Задания для самостоятельного выполнения

5.4. Задания для самостоятельного выполнения

1. Постройте все возможные типы графиков (простые, точечные, гистограммы и т.д.) функции $y = \sin(x)$, $x = 0, 2\pi$. Отобразите все графики в одном графическом окне.

```
f4(x) = sin.(x)
# сгенерируем масссив значений x в диапазоне от 0 до П
x = collect(range(0, 2*pi, length=200))
# зададим функцию
y = f4(x)

fig1 = plot(x, y,
    title="Simple graphic",
    xlabel="X",
    ylabel="Y",
    color="red")
fig2 = scatter(x, y,
    title="Point graphic",
    xlabel="X",
    ylabel="Y",
    color="red")
fig3 = histogram(f4(x),
    title="Histogram",
    xlabel="X",
    ylabel="Y",
    color="purple")
fig4 = histogram(f4(x),
    bins = 10,
    title="Histogram (bins=10)",
    xlabel="X",
    ylabel="Y",
    color="green")
plot(fig1, fig2, fig3, fig4, layout=(2, 2), legends=false, size=(800, 600))
```

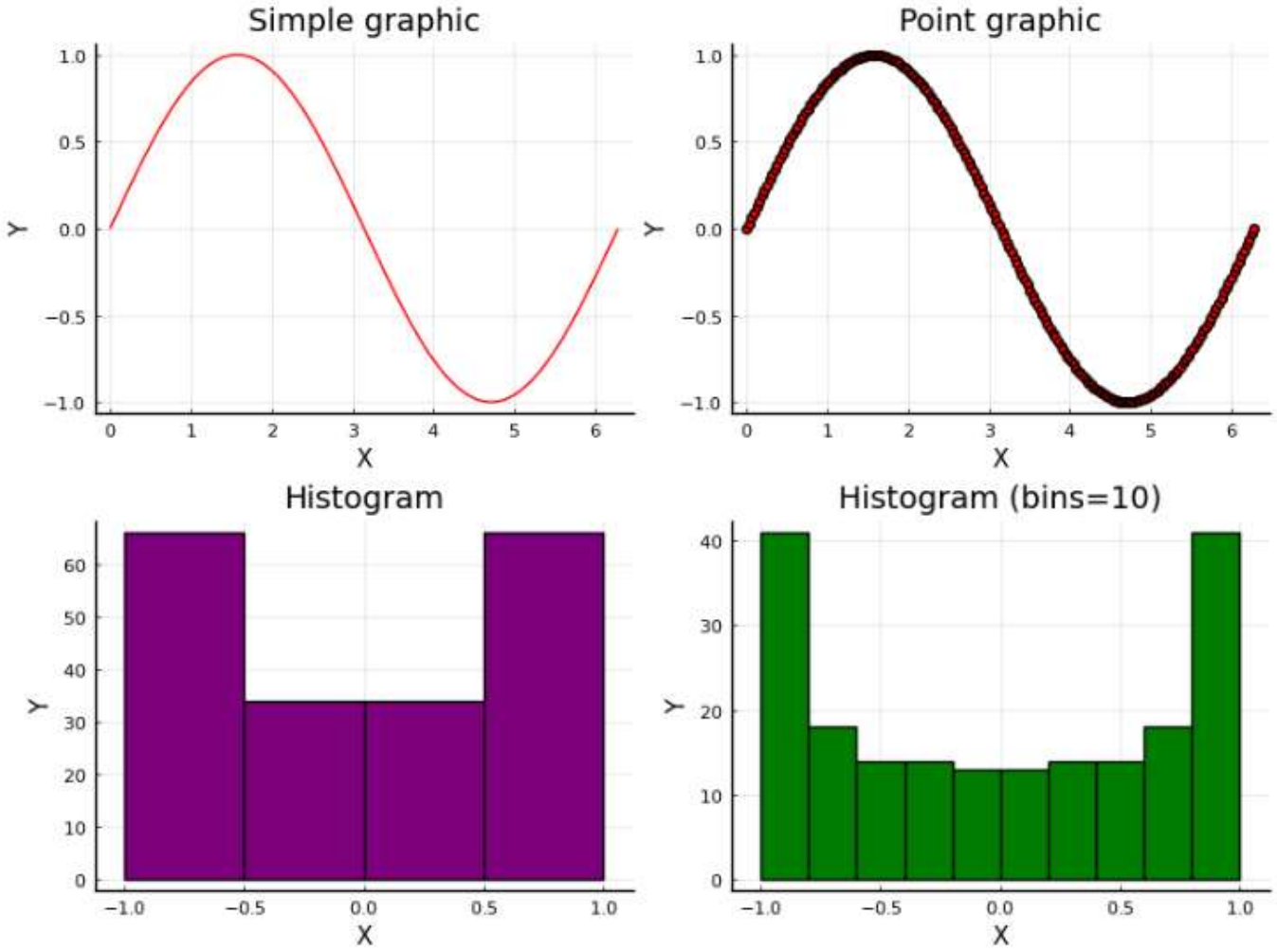
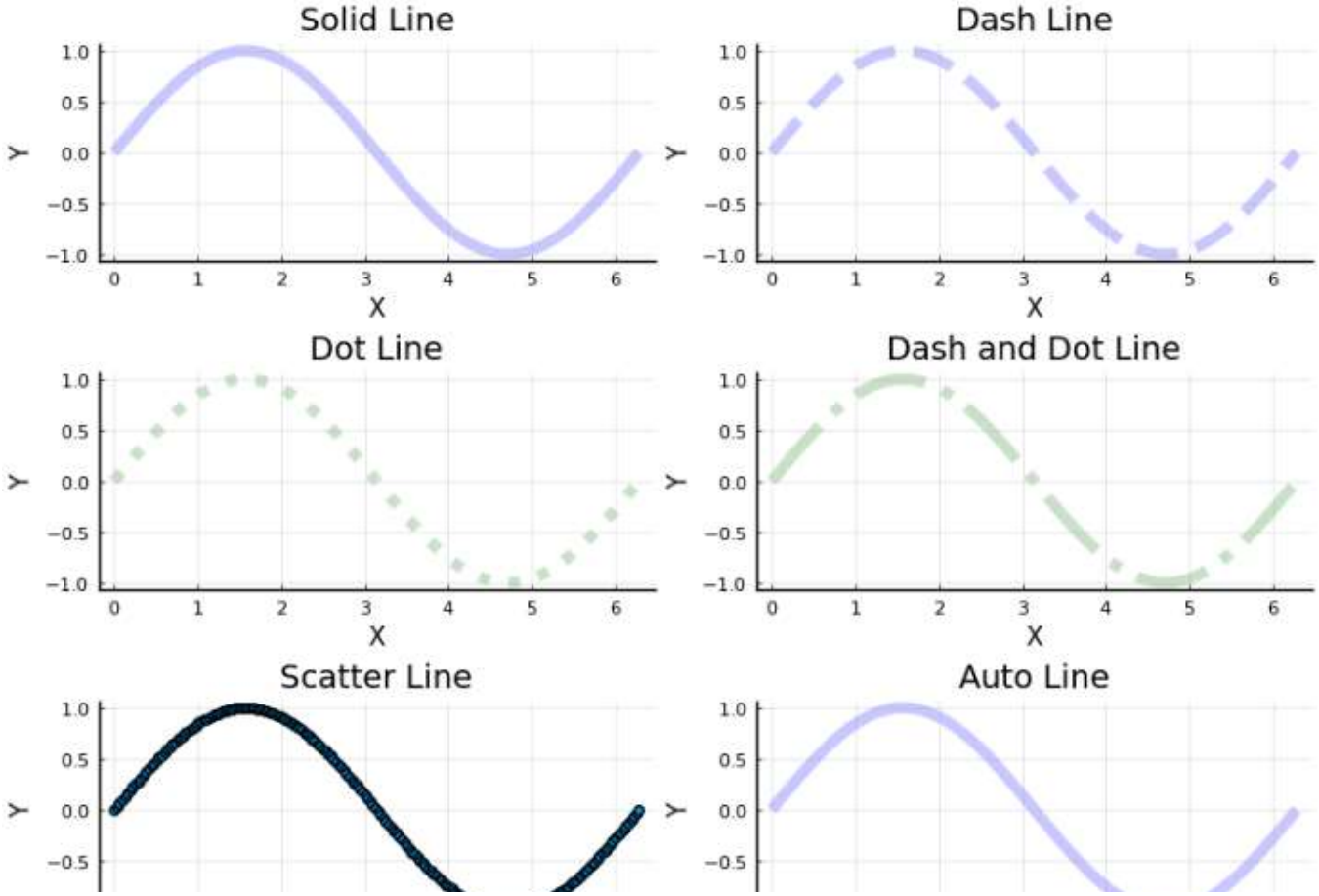


Рисунок 2.1 Код и результат Задания 1

Моя основная идея заключалась в том, чтобы сначала определить набор данных для функции `y = sin(x)` в диапазоне от `0` до `2π`. Затем я последовательно построил несколько типов графиков на основе этих данных: простой линейный график (plot), точечный график (scatter) и две гистограммы (histogram) с разным количеством столбцов (bins). Наконец, я совместил все эти графики в одном общем оконном представлении, используя функцию `plot` с параметром `layout`. Такой подход позволяет наглядно сравнить различные типы визуализации для одних и тех же данных.

2. Постройте графики функции $y = \sin(x)$, $x = 0, 2\pi$ со всеми возможными (сколько сможете вспомнить) типами оформления линий графика. Отобразите все графики в одном графическом окне.

```
f4(x) = sin.(x)
# сгенерируем масссив значений x в диапазоне от 0 до П с шагом 0.1
x = collect(range(0, 2*pi, length=200))
# зададим функцию
y = f4(x)

fig1 = plot(x, y,
    title="Solid Line",
    line = (:blue, 0.2, 5, :solid),
    xlabel="X",
    ylabel="Y")
fig2 = plot(x, y,
    title="Dash Line",
    line = (:blue, 0.2, 5, :dash),
    xlabel="X",
    ylabel="Y")
fig3 = plot(x, y,
    line = (:green, 0.2, 5, :dot),
    title="Dot Line",
    xlabel="X",
    ylabel="Y")
fig4 = plot(x, y,
    line = (:green, 0.2, 5, :dashdot),
    title="Dash and Dot Line",
    xlabel="X",
    ylabel="Y")
fig5 = plot(x, y,
    line = (:blue, 0.2, 5, :scatter),
    title="Scatter Line",
    xlabel="X",
    ylabel="Y")
fig6 = plot(x, y,
    line = (:blue, 0.2, 5, :auto),
    title="Auto Line",
    xlabel="X",
    ylabel="Y")
plot(fig1, fig2, fig3, fig4, fig5, fig6, layout=(3, 2), legends=false, size=(800, 600))
```



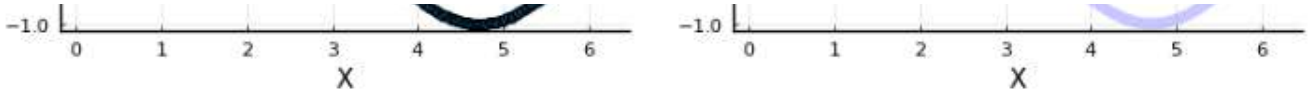


Рисунок 2.2 Код и результат Задания 2

Я сначала определил массив точек для функции `y = sin(x)` в заданном диапазоне `0` до `2π`, а затем построил несколько графиков, меняя типы и стили линий — от сплошной и пунктирной до точечной и комбинированной ("dashdot"). После этого я расположил все полученные графики в одном окне, чтобы можно было наглядно сравнить различные варианты оформления линий.

3. Постройте график функции $y(x) = \pi x$^2 ln(x), назовите оси соответственно. Пусть цвет рамки будет зелёным, а цвет самого графика — красным. Задайте расстояние между надписями и осями так, чтобы надписи полностью умещались в графическом окне. Задайте шрифт надписей. Задайте частоту отметок на осях координат.

```
using Plots.PlotMeasures
f5(x) = (pi*x.^2).*log.(x)
# сгенерируем масссив значений x в диапазоне от 0 до 2П
x = collect(range(0, 2*pi, length=200))
# зададим функцию
y = f5(x)
plotly()
plot(x,y, color="red", box = :on, foreground_color="green",
    xaxis = ("Ось x", (0, 6.5), 6.5:-0.5:0), yaxis = ("Ось y", (0, 250), 250:-25:0),
    leg=:right,
    title="График функции (задание №3)",
    titlefont = (15, "montserrat", :black),
    top_margin = 10mm,
    right_margin = 5mm,
    legendfontsize = 8,
    guidefont = (12, "montserrat", :black),
    tickfont = (8, "montserrat",:black),
    xrotation = 90)
```

Рисунок 2.3 Код и результат Задания 3

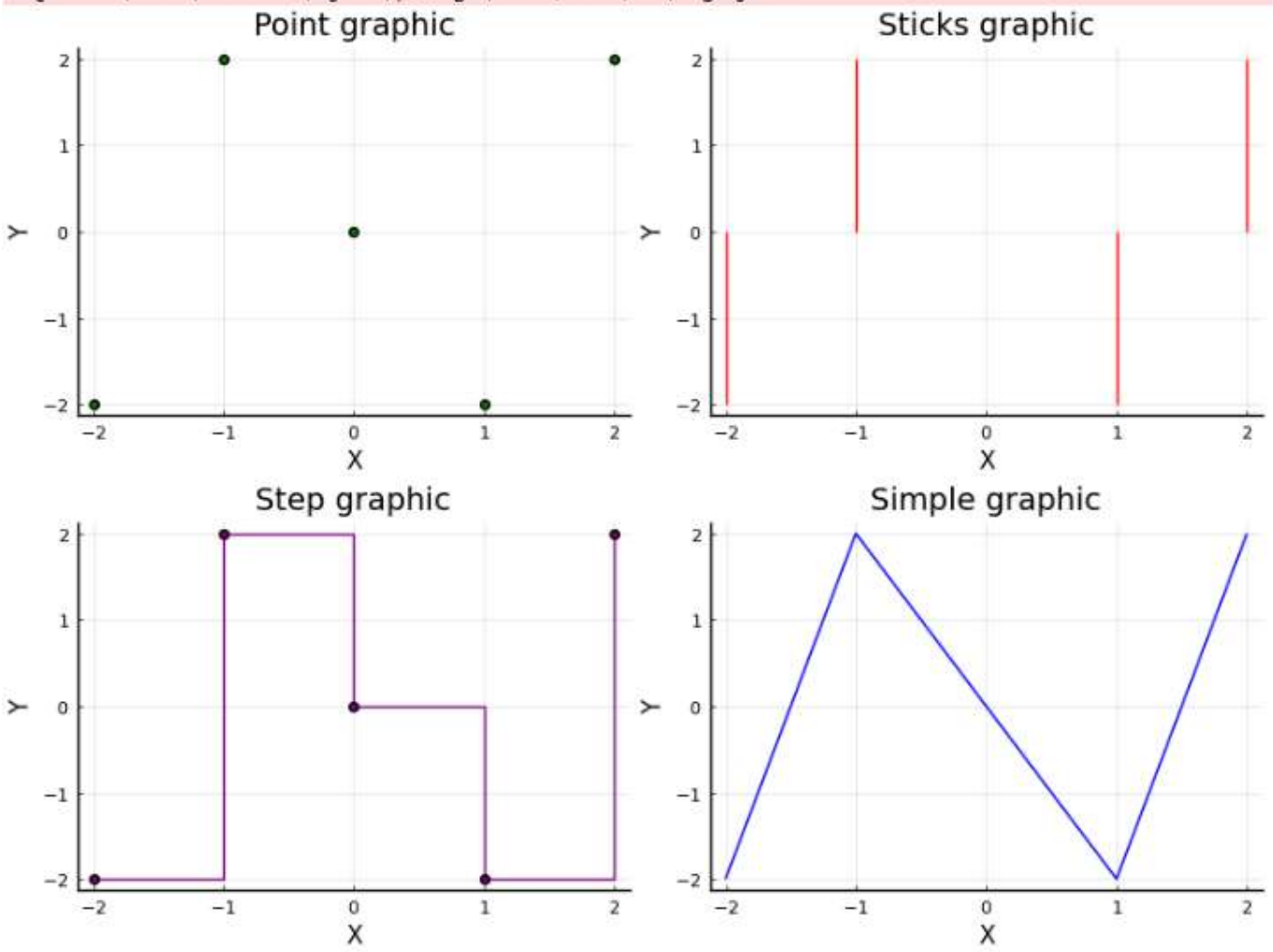
Я сначала определил функцию и сгенерировал точки по оси в требуемом интервале. Затем построил график, придав ему все необходимые свойства: цвет линии — красный, цвет рамки (границы графической области) — зелёный, соответствующие подписи осей с заданным шрифтом, отступы и частоту отметок на осях. В итоге, все настройки визуализации были применены так, чтобы надписи и оси были аккуратно расположены и удобно читаемы.

4. Задайте вектор x = (−2, −1, 0, 1, 2). В одном графическом окне (в 4-х подокнах)изобразите графически по точкам x значения функции $y(x)$ = x^3 − 3x в виде:
   - точек,
   - линий,
   - линий и точек,
   - кривой.

```
# зададим функцию
f(x) = x.^3 - 3*x
x = [-2, -1, 0, 1, 2]
y = f(x)
using Plots.PlotMeasures
pyplot()
fig1 = scatter(x, y,
    title="Point graphic",
    xlabel="X",
    ylabel="Y",
    color="green")
fig2 = plot(x, y,
    title="Sticks graphic",
    seriestype = :sticks,
    xlabel="X",
    ylabel="Y",
    color="red")
fig3 = plot(x, y,
    title="Step graphic",
    marker = '.',
    seriestype = :step,
    xlabel="X",
    ylabel="Y",
    color="purple")
fig4 = plot(x, y,
    title="Simple graphic",
    xlabel="X",
    ylabel="Y",
    color="blue")
plot(fig1, fig2, fig3, fig4, layout=(2, 2), legends=false, size=(800, 600))
```

Рисунок 2.4.1 Код и результат Задания 4.1

```
┌ Warning: Skipped marker arg ..
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/args.jl:862
```




```
# Сохраню полученное изображение
savefig("figure_solomko.png")
```

Рисунок 2.4.2 Код и результат Задания 4.2

Я сначала определил функцию и выбрал набор точек для оценки этой функции. Затем я построил несколько вариантов графиков, используя один и тот же набор данных: в одном подграфике я изобразил точки (scatter), в другом — линии (sticks), в третьем — сочетание линий и точек (step с маркерами), и, наконец, в четвёртом — простую кривую (line). После этого я расположил все четыре вида графических представлений в одном общем окне в виде сетки из четырёх подграфиков. Такой подход позволил мне наглядно сравнить различные способы визуализации одних и тех же данных.

5. Задайте вектор x = (3, 3.1, 3.2, ... , 6). Постройте графики функций $y1(x) = \pi x$ и $y2(x) = \exp(x)\cos(x)$ в указанном диапазоне значений аргумента x следующим образом:
   - постройте оба графика разного цвета на одном рисунке, добавьте легенду и сетку для каждого графика; укажите недостатки у данного построения;
   - постройте аналогичный график с двумя осями ординат.

```
f_1(x) = pi*x
f_2(x) = exp.(x).*cos.(x)
x = [i for i in 3:0.1:6]
y_1 = f_1(x)
y_2 = f_2(x)

pyplot()
plot(x, y_1,
    title="График функций (Задание №5)",
    xlabel="X",
    ylabel="Y",
    leg=:topleft,
    color="blue",
    grid = (:y, :orange, :dot),
    right_margin = 20mm)

plot!(x, y_2,
    color="purple",
    leg=:bottomright,
    grid = :on,
    box = :on)
```

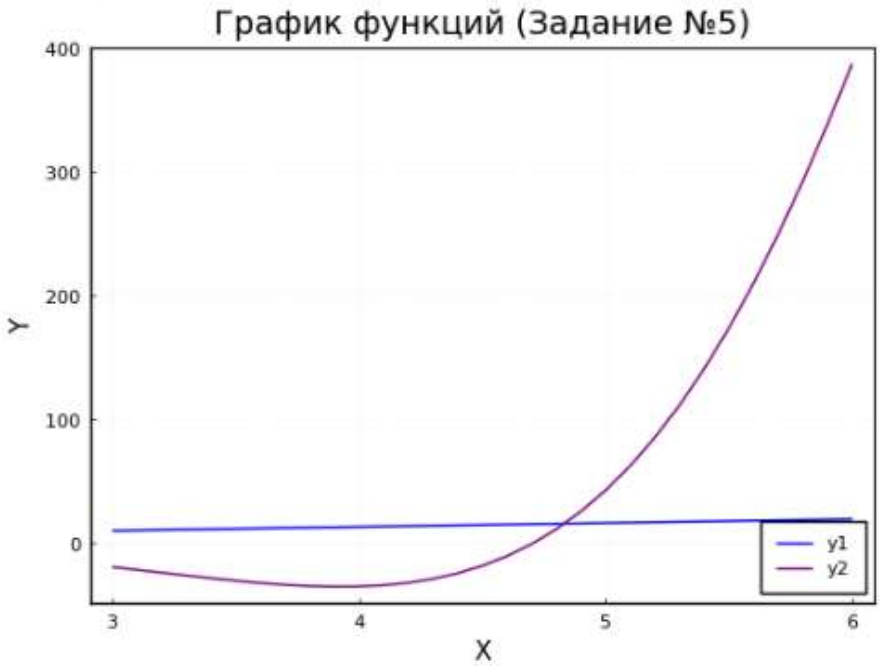


Рисунок 2.5.1 Код и результат Задания 5.1

```
plot(x, y_1,
    title="График функций (Задание №5)",
    xlabel="X",
    ylabel="Y1",
    leg=:topleft,
    color="blue",
    grid = (:y, :orange, :dot),
    right_margin = 20mm)

plot!(twinx(), x, y_2,
    ylabel = "Y2",
    color="purple",
    leg=:bottomright,
    grid = :on,
    box = :on)
```

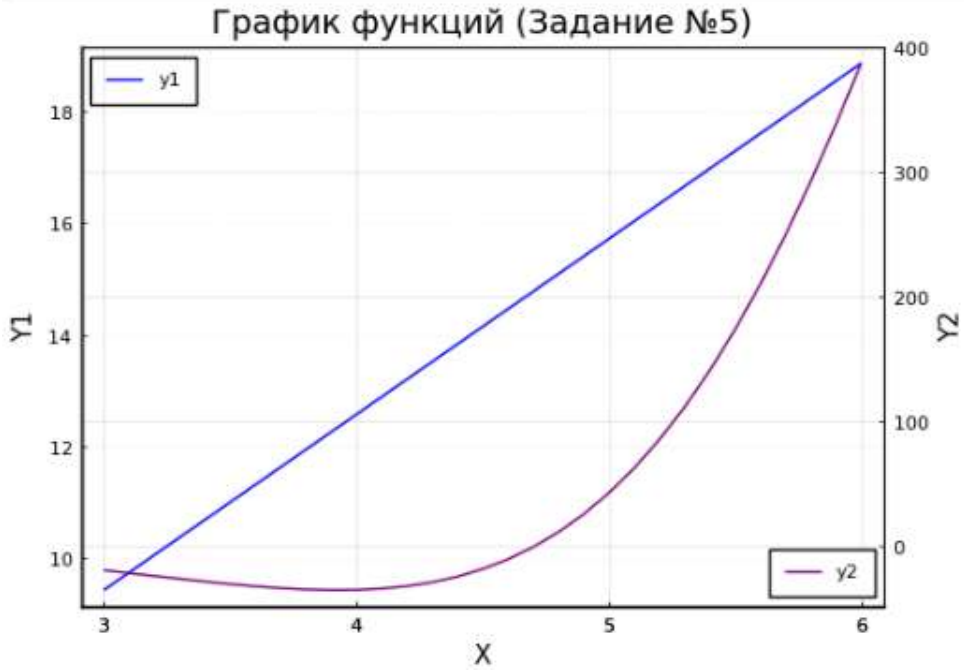


Рисунок 2.5.2 Код и результат Задания 5.2

Мой основной подход заключался в том, чтобы сначала определить функции , а затем построить их на одном наборе точек . Я создал два графика. В первом варианте оба графика были отображены на одной координатной сетке с общей осью ординат, использовал разный цвет и легенду для каждой функции, а также включил сетку. Во втором варианте я воспользовался двумя осями ординат: одну оставил для (y*1), другую добавил с помощью* `twinx()` *для (y*2). Таким образом, каждый график имел свою вертикальную шкалу, что позволило более наглядно сравнивать их.

6. Постройте график некоторых экспериментальных данных (придумайте сами), учитывая ошибку измерения

```
f_3(x) = x.^2 - 2*x
x = rand(20)
y_3 = f_3(x)
n = 20

error = 1.23 * y_3 / sqrt(n)

plot(y_3, ylims=(-5, 5), err = error)
```

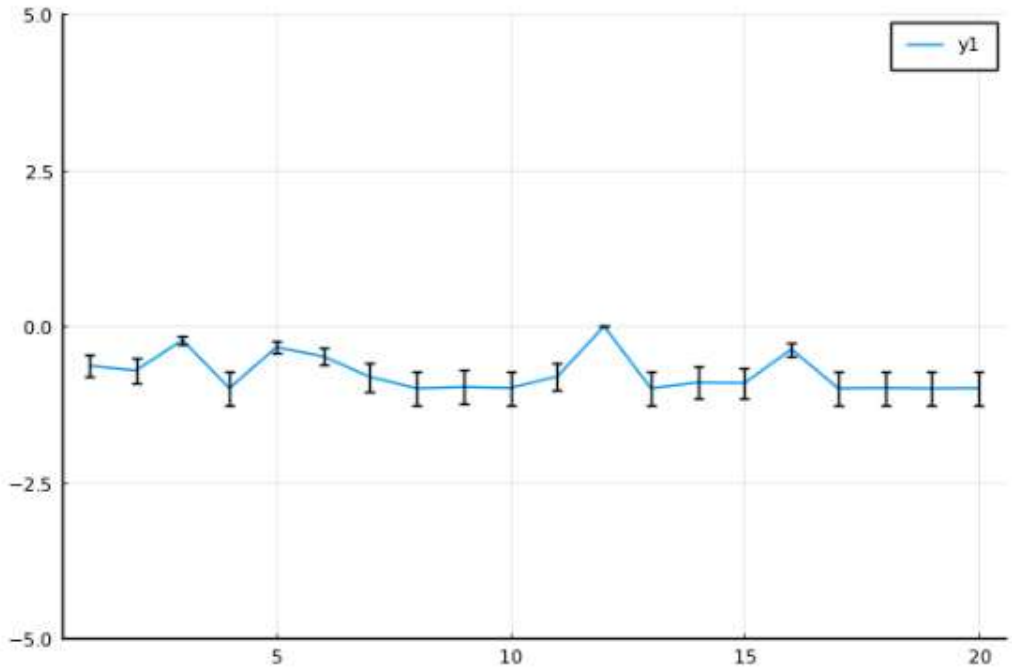


Рисунок 2.6 Код и результат Задания 6

Я решил сгенерировать набор экспериментальных данных, основываясь на простой функции , и использовать случайные значения (x) для придания имитации неопределённости. Затем я вычислил значения (y_3) и задал ошибку измерения как некоторую пропорциональную величину от значения функции, делённую на корень из количества точек. В конце я построил график с использованием ошибок, чтобы визуализировать разброс данных и показать, как может выглядеть экспериментальная неопределённость измерений.

7. Постройте точечный график случайных данных. Подпишите оси, легенду, название графика.

```
x = rand(20)
y = rand(20)

scatter(x, y,
    title = "Точечный график случайных чисел",
    label = "Точки (x; y)",
    leg = :topright,
    xlabel="X",
    ylabel="Y")
```

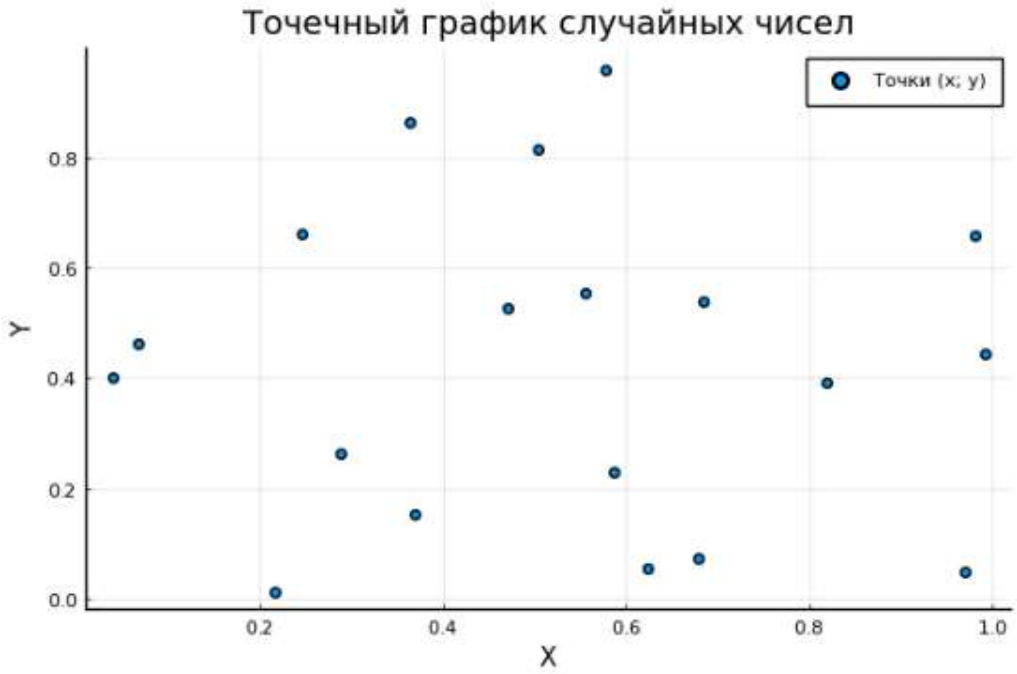


Рисунок 2.7 Код и результат Задания 7

Я сгенерировал два набора случайных чисел для осей (x) и (y), затем построил точечный график, отобразив каждую пару (*xi, yi*) в виде отдельной точки. Добавил подписи к осям, легенду и заголовок, чтобы график был понятен и информативен. Такой подход даёт быстрое визуальное представление о распределении случайных данных.

8. Постройте 3-мерный точечный график случайных данных. Подпишите оси, легенду, название графика.

```
x = rand(60)
y = rand(60)
z = rand(60)

scatter(x, y, z,
    xlabel = "x",
    ylabel = "y",
    zlabel = "z",
    label = "Точки (x; y; z)",
    title = "Точечный 3D-график")
```

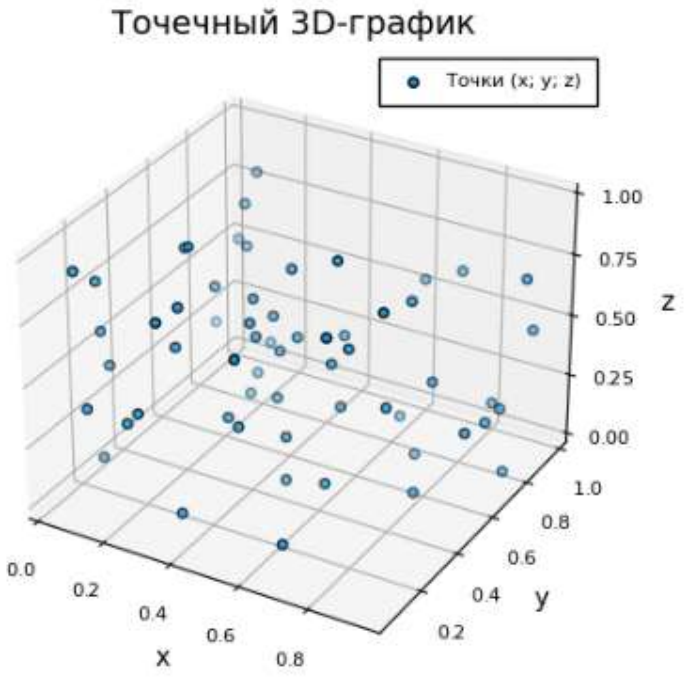
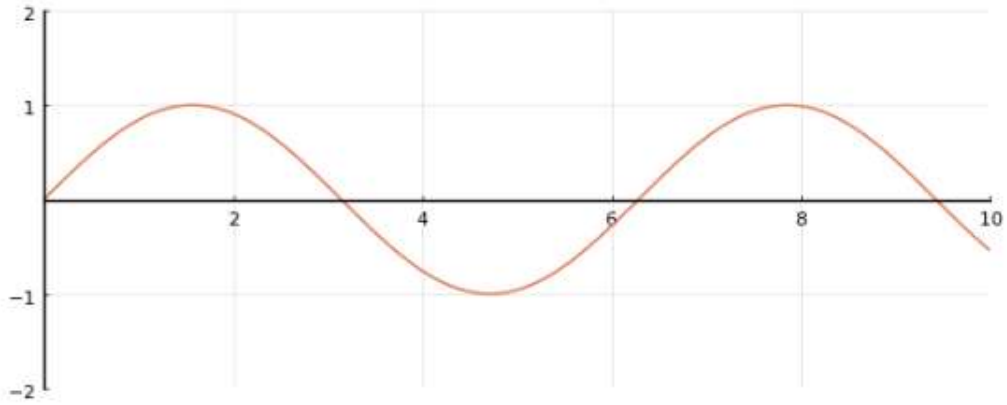


Рисунок 2.8 Код и результат Задания 8

Я сгенерировал три набора случайных чисел для координат (x), (y) и (z), а затем построил трёхмерный точечный график. Для удобства интерпретации я добавил подписи к осям, легенду и заголовок графика. Таким образом, я получил наглядное 3D-представление случайных данных, показывающее их распределение в пространстве.

9. Создайте анимацию с построением синусоиды. То есть вы строите последовательность графиков синусоиды, постепенно увеличивая значение аргумента. После соедините их в анимацию.

```
n = 300
x = collect(0*pi : 2*pi/100 : 10*pi+pi/100)

anim = @animate for i in 1:n
    fig = plot(1,
               xlim=(0, 10),
               ylim=(-2, 2),
               c=:purple,
               aspect_ratio=1,
               legend=false,
               framestyle=:origin)
    step = x[1 : i]
    y = sin.(step)
    plot!(step, y)
end
#Сохранила анимацию в gif-файл
gif(anim, "sinusoida.gif")
```




```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/sinusoida.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

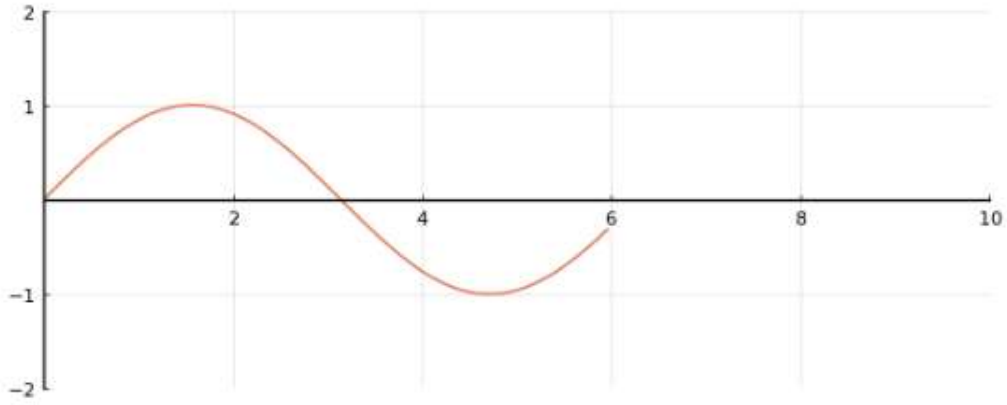


Рисунок 2.9 Код и результат Задания 9

Я захотел создать анимацию, показывающую, как синусоида «нарастает» с увеличением аргумента. Для этого я сначала сформировал вектор значений (x), затем пошагово — от кадра к кадру — добавлял всё больше точек на график, вычисляя для них значение sin(x). С помощью цикла и макроса `@animate` я генерировал последовательность кадров, а затем превратил её в GIF-анимацию. Таким образом, по мере продвижения по циклу синусоида постепенно удлиняется, наглядно демонстрируя процесс построения кривой.

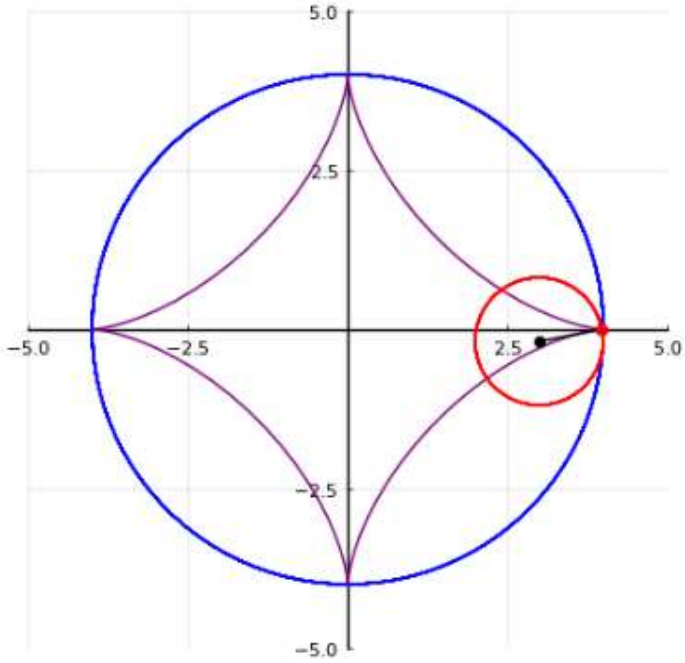
10. Постройте анимированную гипоциклоиду для 2 целых значений модуля k и 2 рациональных значений модуля k.

```
function hypocycloid(x, r0, n)
    # радиус малой окружности:
    r0 = r0
    # коэффициент для построения большой окружности:
    k = x
    # число отсчётов:
    count = n
    # массив значений угла θ:
    θ = collect(0: 2*π/100 : 10*π+2*π/count)

    # массивы значений координат:
    x_1 = r0 * k * cos.(θ)
    y_1 = r0 * k * sin.(θ)
    #В конце сделаем анимацию получившегося изображения
    anim = @animate for i in 1:count
        # задаём оси координат:
        plt=plot(5,
            xlim=(-k-1, k+1),
            ylim=(-k-1, k+1),
            color=:red,
            aspect_ratio=1,
            legend=false,
            framestyle=:origin)
        # большая окружность:
        plot!(plt, x_1, y_1, c=:blue, legend=false)
        t = θ[1 : i]

        # гипоциклоида:
        x = r0 * (k-1) * cos.(t) + r0 * cos.((k-1) * t)
        y = r0 * (k-1) * sin.(t) - r0 * sin.((k-1) * t)
        plot!(x,y, color=:purple)

        # малая окружность:
        x_r0 = r0*(k-1)*cos(t[end]) .+ r0*cos.(θ)
        y_r0 = r0*(k-1)*sin(t[end]) .+ r0*sin.(θ)
        plot!(x_r0, y_r0, color=:red)

        # радиус малой окружности:
        xl_r0 = transpose([r0*(k-1)*cos(t[end]) x[end]])
        yl_r0 = transpose([r0*(k-1)*sin(t[end]) y[end]])
        plot!(xl_r0, yl_r0,
            markershape=:circle,
            markersize=4,
            color=:black)
        scatter!([x[end]],
            [y[end]],
            color=:red,
            markerstrokecolor=:red)
    end
    gif(anim,"hypocycloid.gif")
end
```

```
hypocycloid (generic function with 1 method)
```

Рисунок 2.10.1 Код и результат Задания 10.1

```
# первый вариант
hypocycloid(4, 1, 100)
```




```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/hypocycloid.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

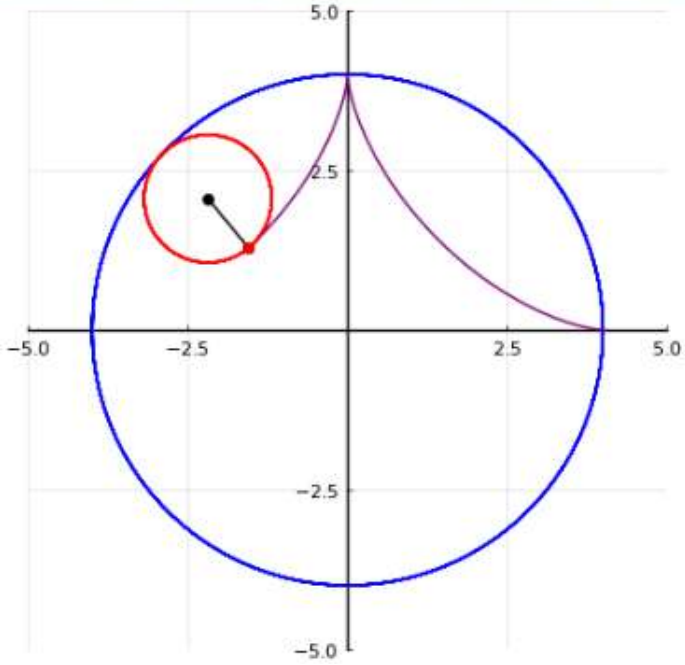
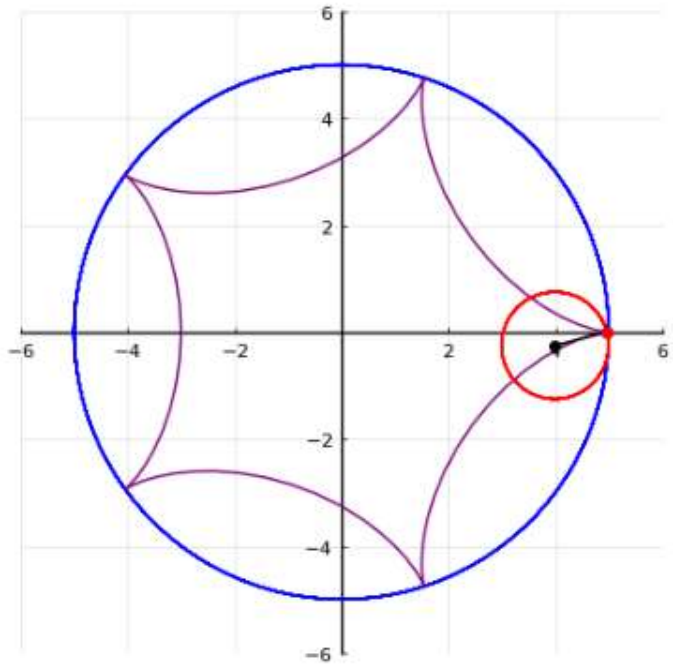


Рисунок 2.10.2 Код и результат Задания 10.2

```
# второй вариант
hypocycloid(5, 1, 100)
```




```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/hypocycloid.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

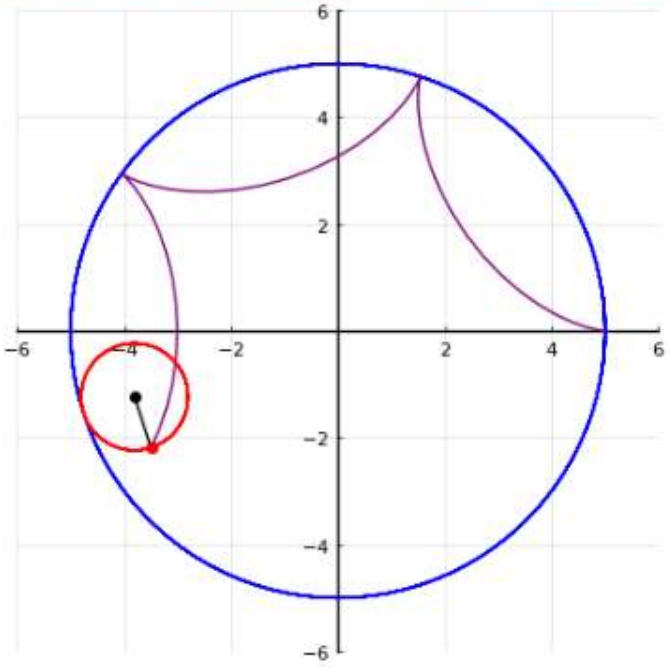
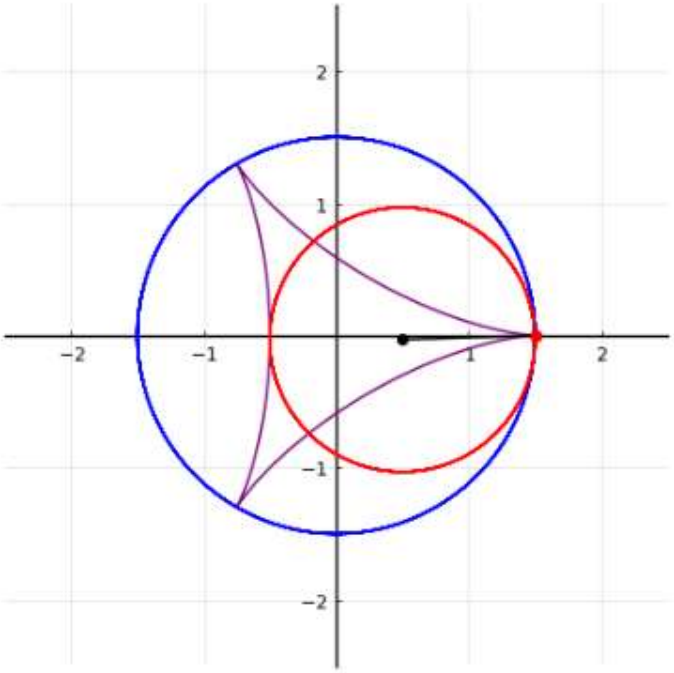


Рисунок 2.10.3 Код и результат Задания 10.3

```
# третий вариант
hypocycloid(1.5, 1, 200)
```




```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/hypocycloid.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

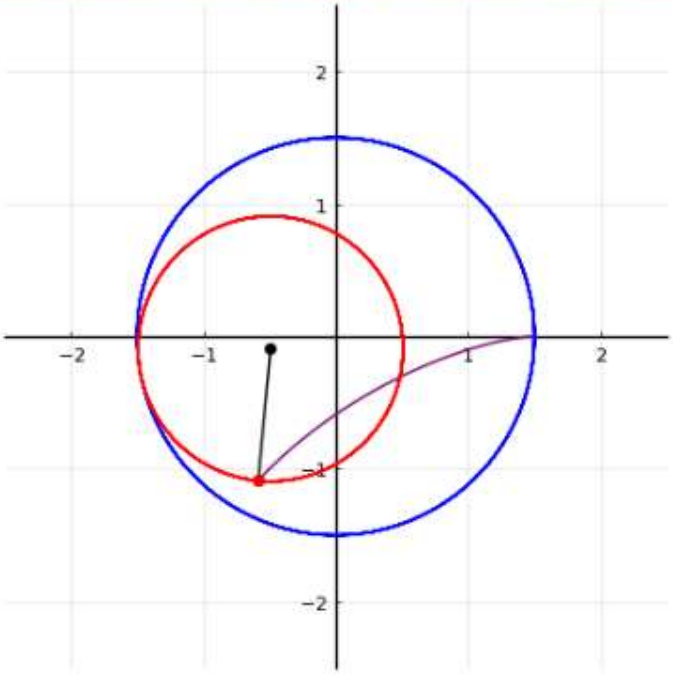
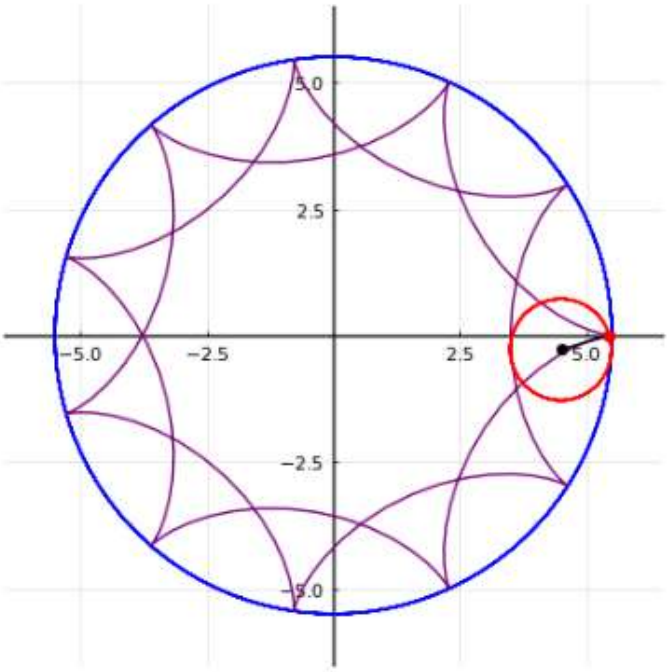


Рисунок 2.10.4 Код и результат Задания 10.4

```
# четвертый вариант
hypocycloid(5.5, 1, 200)
```




```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/hypocycloid.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

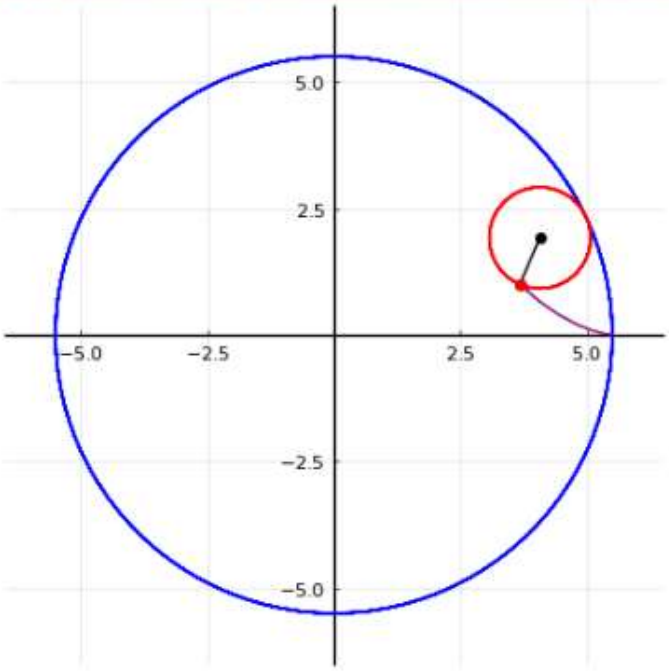


Рисунок 2.10.5 Код и результат Задания 10.5

Я решил визуализировать гипоциклоиду, постепенно «прорисовывая» её траекторию и показывая процесс вращения маленькой окружности по внутренней стороне большой. Для этого я создал функцию, которая на заданном интервале угла строит координаты большой окружности, малой окружности и самой гипоциклоиды. Затем я использовал цикл анимации (`@animate`), который по шагам добавляет новые точки на траектории, позволяя увидеть, как форма развивается во времени. Поменяв параметр (k) (отношение радиусов окружностей), я получил разные варианты гипоциклоид. В итоге, каждая анимация сохраняется в GIF-файл, наглядно демонстрируя процесс построения фигуры для разных значений (k).

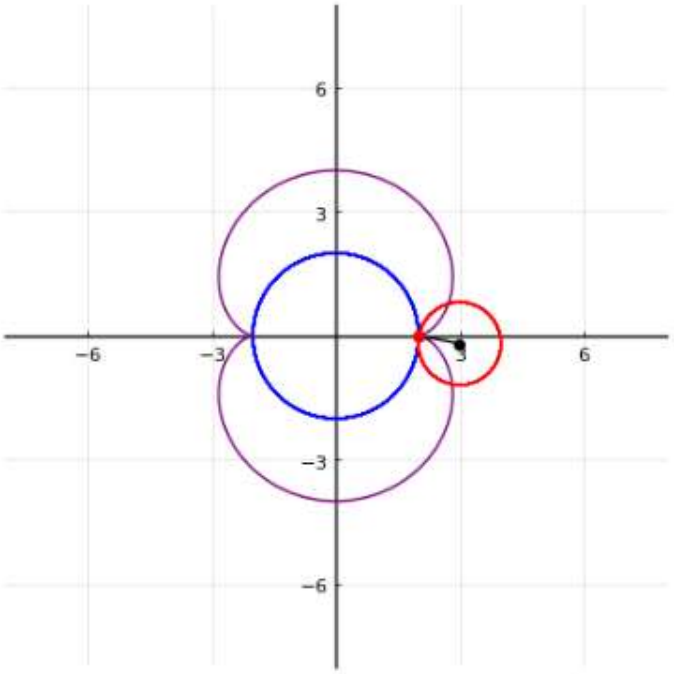
11. Постройте анимированную эпициклоиду для 2 целых значений модуля k и 2 рациональных значений модуля k.

```
function epicycloid(x, r0, n)
    # радиус малой окружности:
    r0 = r0
    # коэффициент для построения большой окружности:
    k = x
    # число отсчётов:
    count = n
    # массив значений угла θ:
    θ = collect(0: 2*π/100 : 10*π+2*π/count)

    # массивы значений координат:
    x_1 = r0 * k * cos.(θ)
    y_1 = r0 * k * sin.(θ)
    #В конце сделаем анимацию получившегося изображения
    anim = @animate for i in 1:count
        # задаём оси координат:
        plt=plot(5,
            xlim=(-2*r0*k*2, 2*r0*k*2),
            ylim=(-2*r0*k*2, 2*r0*k*2),
            color=:red,
            aspect_ratio=1,
            legend=false,
            framestyle=:origin)
        # большая окружность:
        plot!(plt, x_1, y_1, c=:blue, legend=false)
        t = θ[1 : i]

        # гипоциклоида:
        x = r0 * (k + 1) * cos.(t) - r0 * cos.((k + 1) * t)
        y = r0 * (k + 1) * sin.(t) - r0 * sin.((k + 1) * t)
        plot!(x,y, color=:purple)

        # малая окружность:
        x_r0 = r0*(k + 1)*cos(t[end]) .- r0*cos.(θ)
        y_r0 = r0*(k + 1)*sin(t[end]) .- r0*sin.(θ)
        plot!(x_r0, y_r0, color=:red)

        # радиус малой окружности:
        xl_r0 = transpose([r0*(k + 1)*cos(t[end]) x[end]])
        yl_r0 = transpose([r0*(k + 1)*sin(t[end]) y[end]])
        plot!(xl_r0, yl_r0,
            markershape=:circle,
            markersize=4,
            color=:black)
        scatter!([x[end]],
            [y[end]],
            color=:red,
            markerstrokecolor=:red)
    end
    gif(anim,"epicycloid.gif")
end
```

```
epicycloid (generic function with 1 method)
```

Рисунок 2.11.1 Код и результат Задания 11.1

```
# первый вариант
epicycloid(2, 1, 100)
```




```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/epicycloid.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

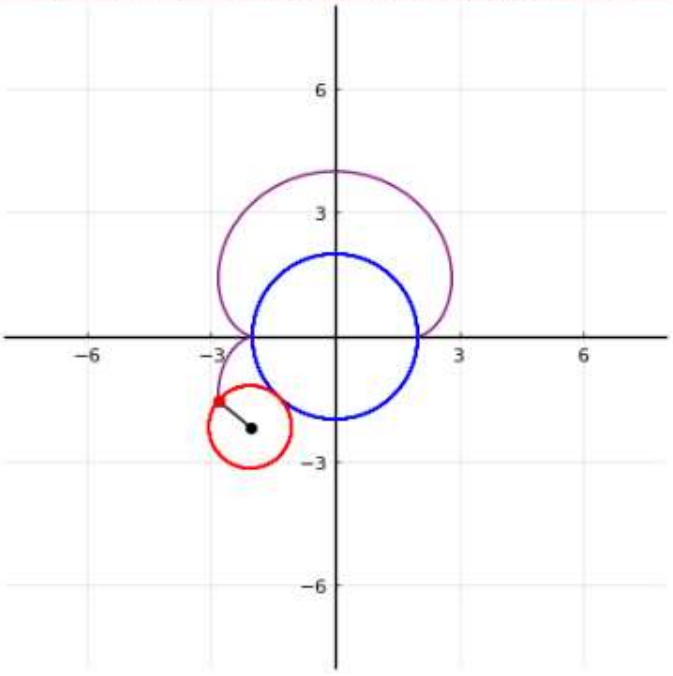
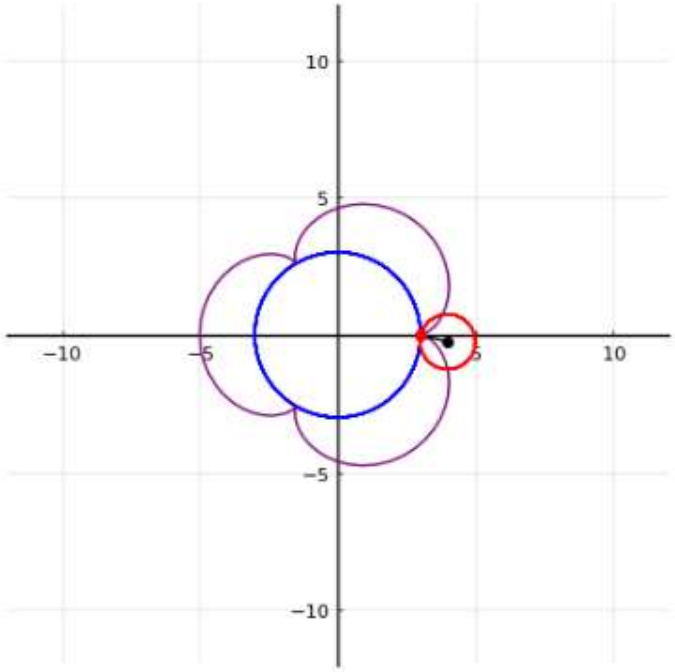


Рисунок 2.11.2 Код и результат Задания 11.2

```
# второй вариант
epicycloid(3, 1, 100)
```




```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/epicycloid.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

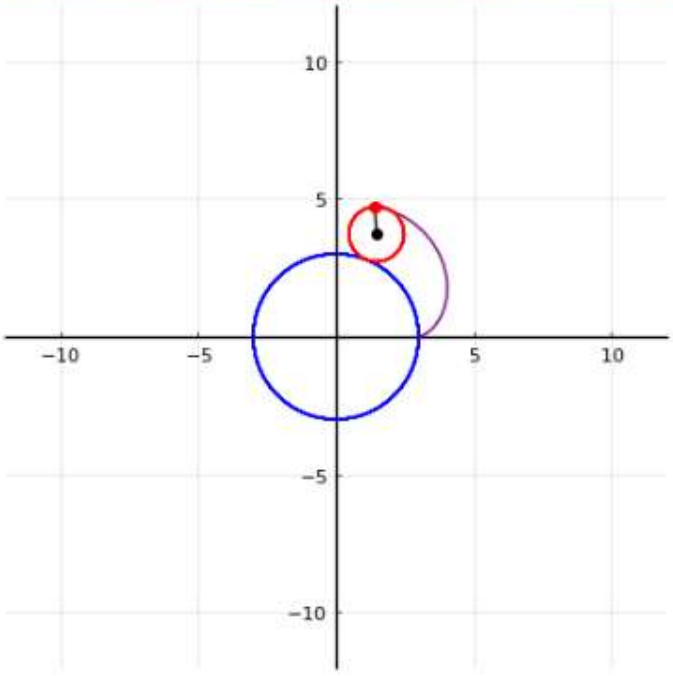
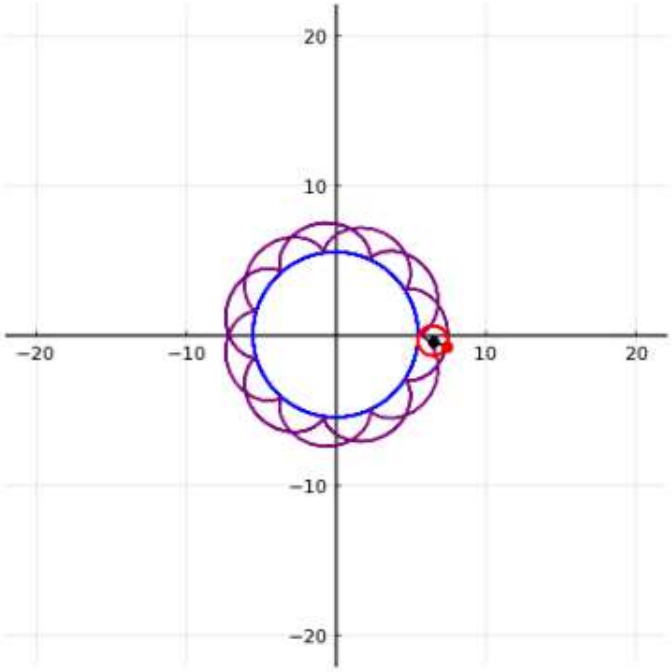


Рисунок 2.11.3 Код и результат Задания 11.3

```
# третий вариант
epicycloid(5.5, 1, 500)
```




```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/epicycloid.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

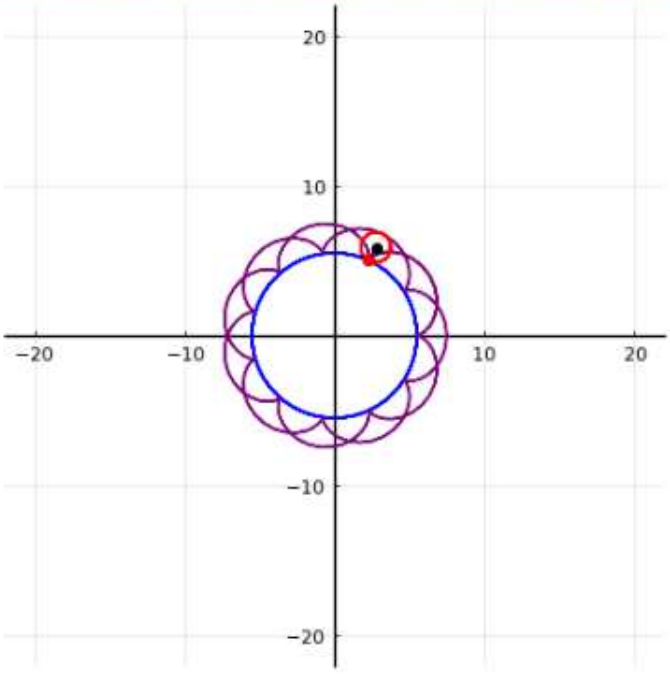
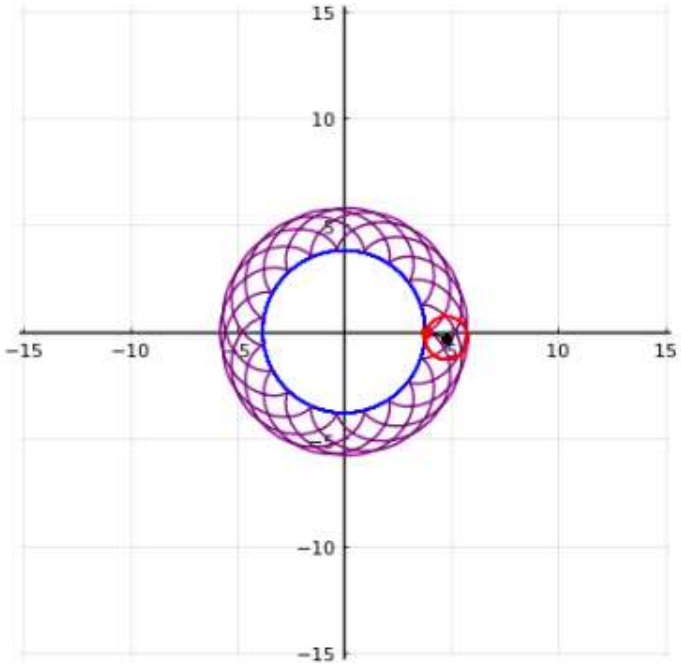


Рисунок 2.11.4 Код и результат Задания 11.4

```
# четвертый вариант
epicycloid(3.8, 1, 500)
```




```
┌ Info: Saved animation to
│   fn = /Users/anastasia/Desktop/Учеба универ/Практикум по телекоммуникациям/5 lab/epicycloid.gif
└ @ Plots /Users/anastasia/.julia/packages/Plots/YdauZ/src/animation.jl:104
```

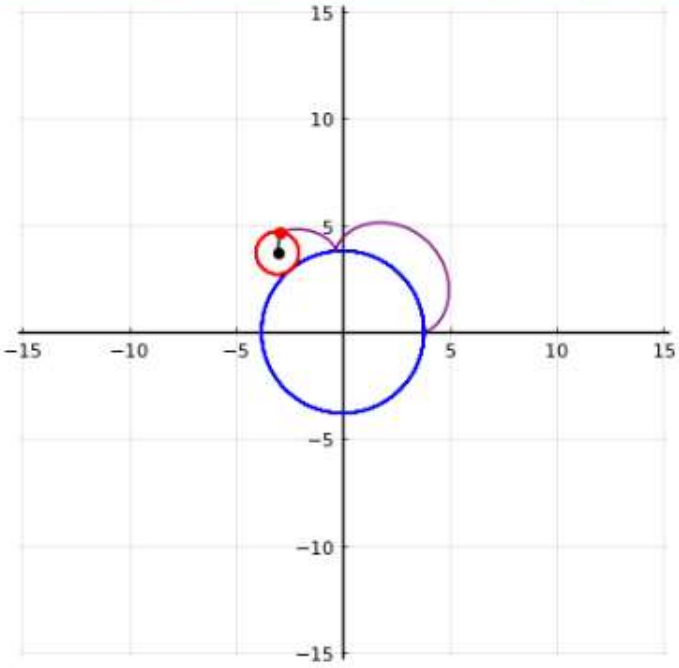


Рисунок 2.11.5 Код и результат Задания 11.5

Я захотел наглядно показать построение эпициклоиды при различных значениях параметра (k). Для этого я написал функцию, которая для заданных (k), радиуса (r_0) и количества шагов (n) генерирует точки большой и малой окружностей, а также координаты эпициклоиды на каждом шаге, постепенно «прорисовывая» её траекторию. В цикле анимации на каждом кадре я добавляю всё больше точек, показывая, как фигура формируется во времени. Изменяя значение (k), я получил различные характерные формы эпициклоиды. После завершения построения всех кадров я сохранил анимацию в виде GIF-файла.

Вывод

Я освоил синтаксис языка Julia для построения графиков.