Отчет по лабораторной работе 5

Петрушов Дмитрий, 1032212287

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Основные пакеты для работы с графиками в Julia

Julia поддерживает несколько пакетов для работы с графиками. Использование того или иного пакета зависит от целей, преследуемых пользователем при построении. Стандартным для Julia является пакет Plots.jl.

Рассмотрим построение графика функции f(x) = (3x2 + 6x - 9)e-0,3x разными способами (рис. [2.1] - (рис. [2.2]):

```
# 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")
```

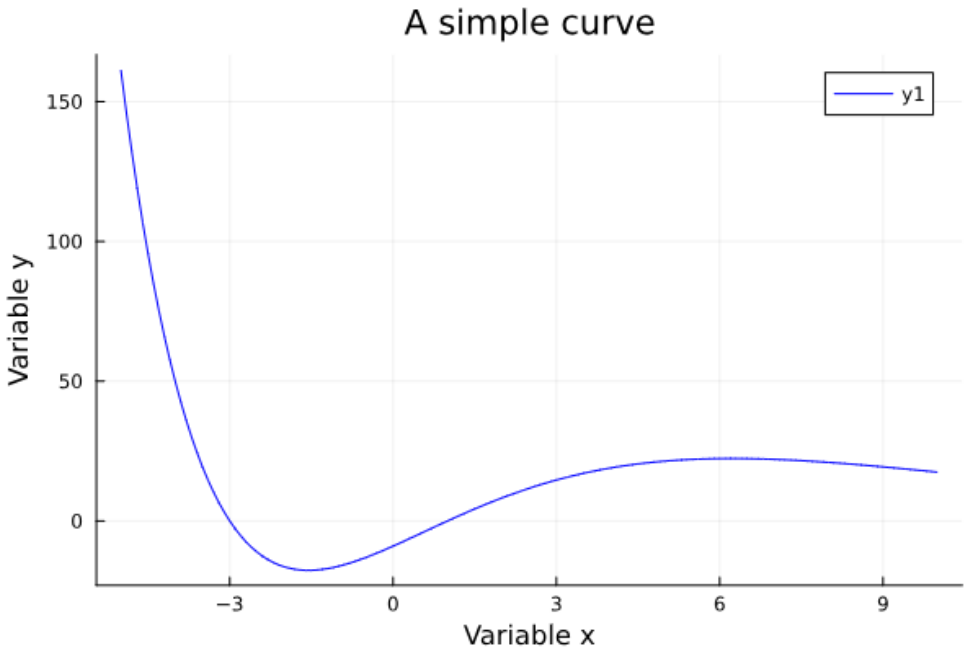

Рис. 2.1: График функции, построенный при помощи gr()

```
# 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")
```

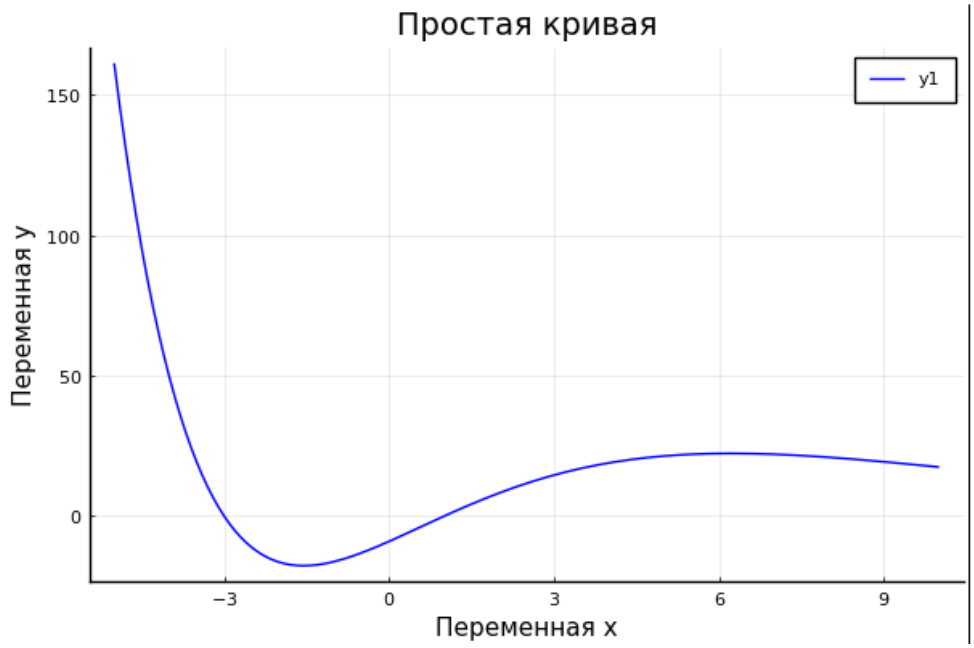

Рис. 2.2: График функции, построенный при помощи pyplot()

2.2 Опции при построении графика

На примере графика функции sin(x) и графика разложения этой функции в ряд Тейлора рассмотрим дополнительные возможности пакетов для работы с графикой (рис. [2.3] - рис. [2.5]):

[7]:
```
# 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)
```

[7]:

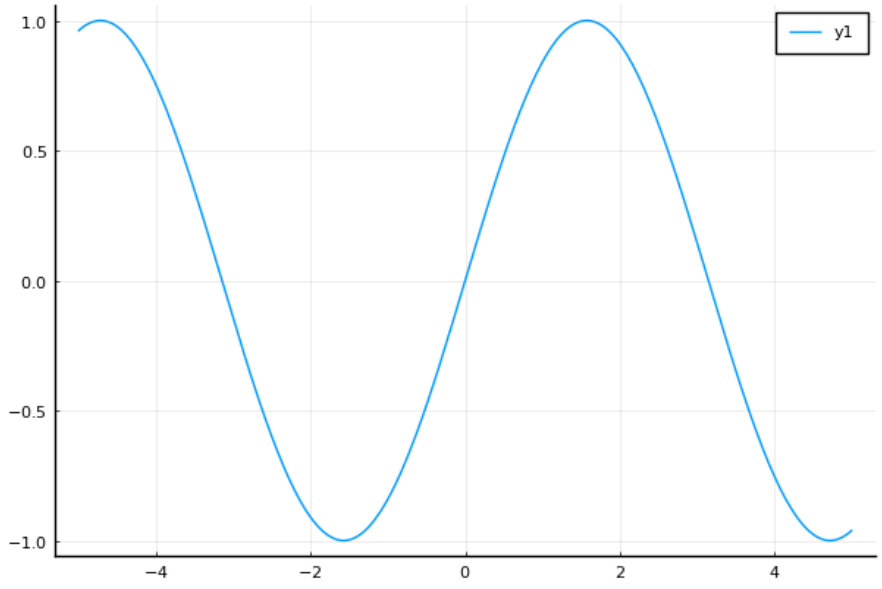

Рис. 2.3: График функции sin(x)

```
# задание функции разложения исходной функции в ряд Тейлора:
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)
# построение двух функций на одном графике:
plot(sin_theor)
plot!(sin_taylor)
```

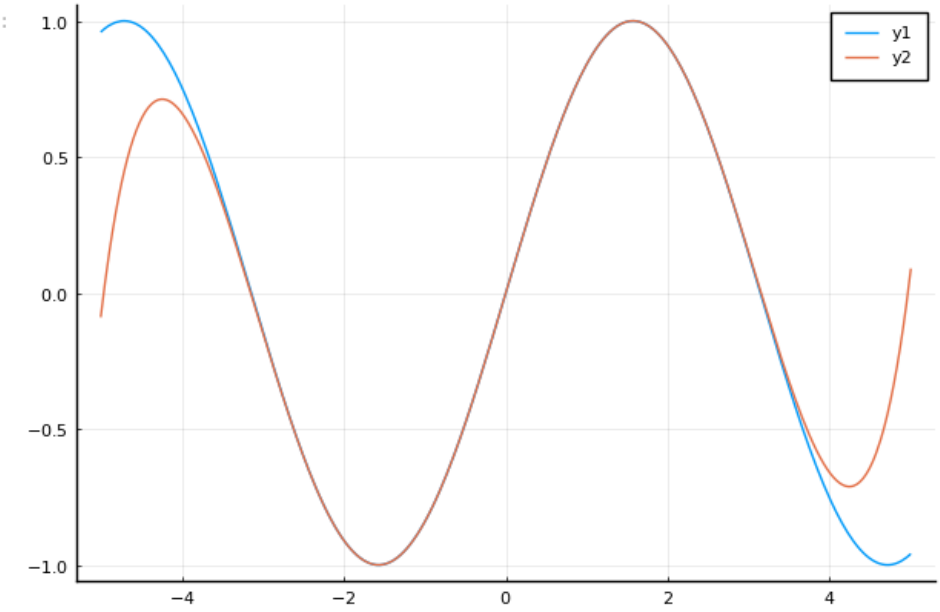

Рис. 2.4: График функции разложения исходной функции в ряд Тейлора

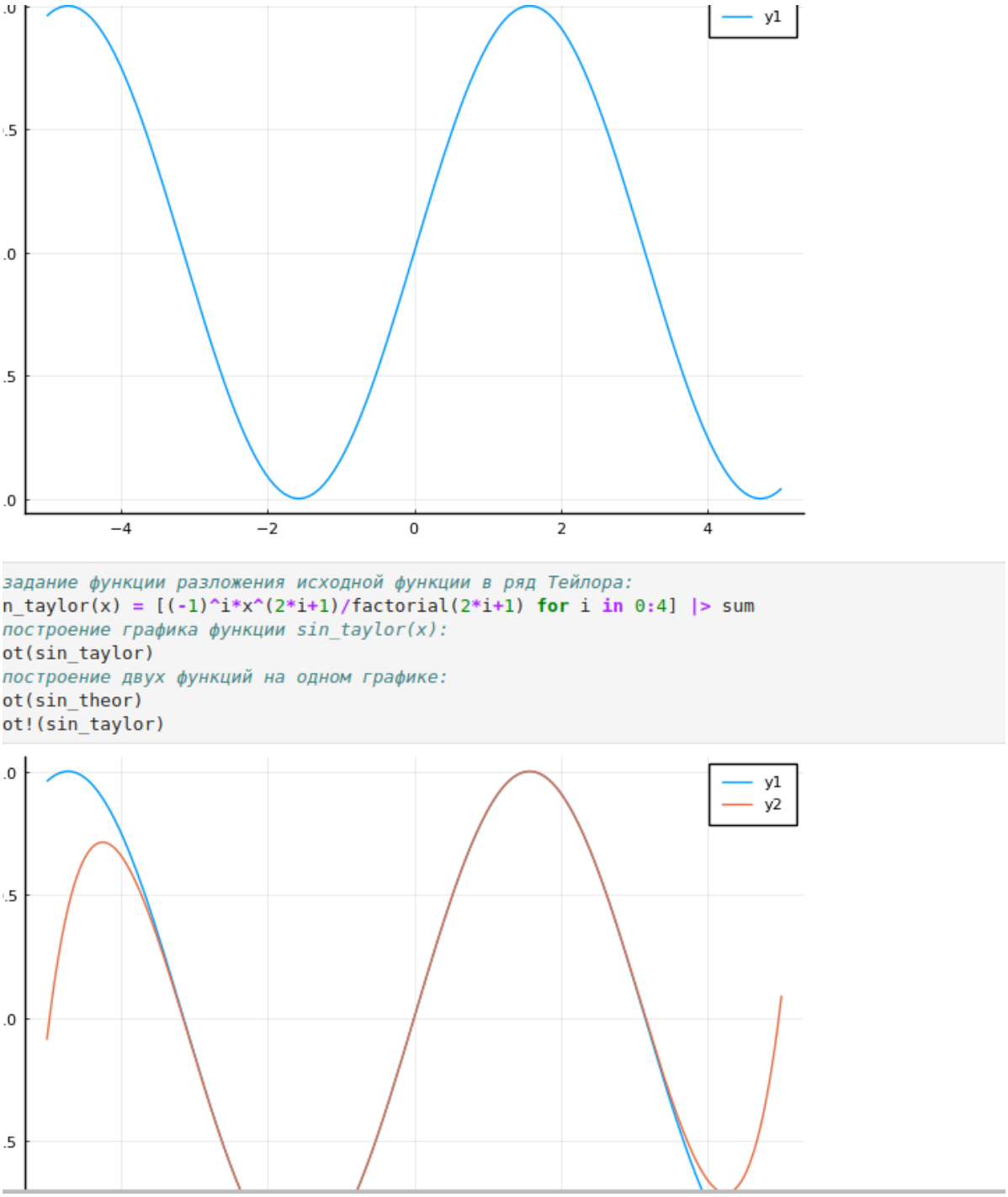

Рис. 2.5: Графики исходной функции и её разложения в ряд Тейлора

Затем добавим различные опции для отображения на графике (рис. [2.6]):

```
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",
    line=(:black, 1.0, 2, :dash))
```

Рис. 2.6: Вид графиков после добавления опций при их построении

2.3 Точечный график

Как и при построении обычного графика для точечного графика необходимо задать массив значений x, посчитать или задать значения y, задать опции построения графика (рис. [2.7]):

```
# 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"
)
```

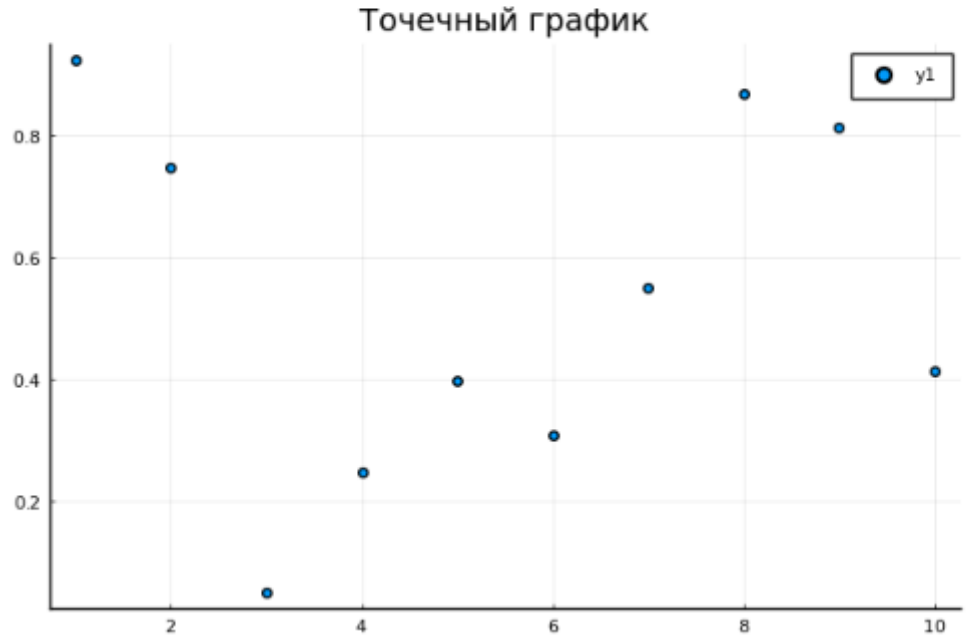

Рис. 2.7: График десяти случайных значений на плоскости (простой точечный график)

Для точечного графика можно задать различные опции, например размер маркера, его тип, цвет и и т.п. (рис. [2.8]):

```
# 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)
```

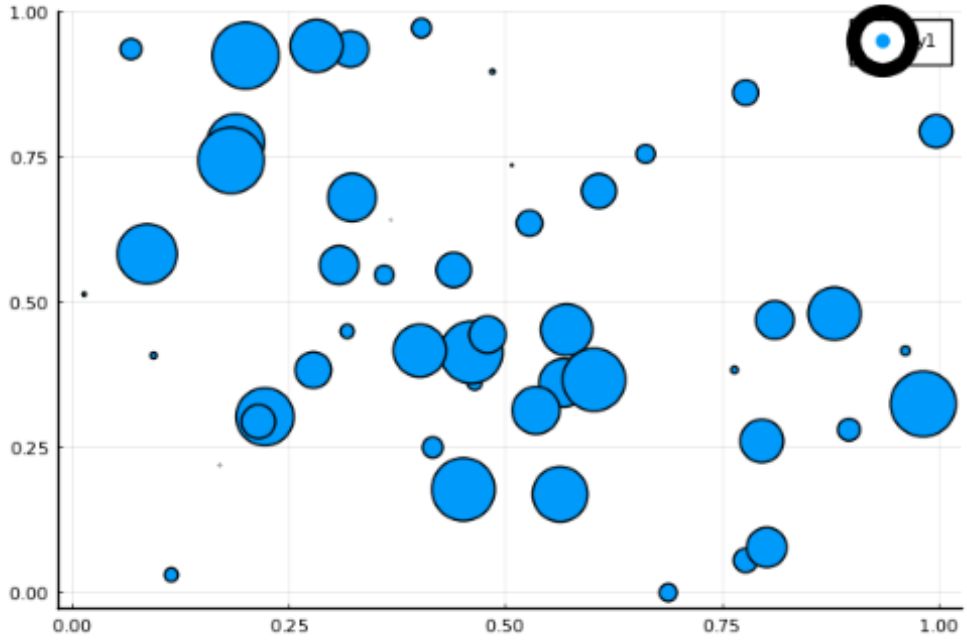

Рис. 2.8: График пятидесяти случайных значений на плоскости с различными опциями отображения (точечный график с кодированием значения размером точки)

Также можно строить и 3-мерные точечные графики (рис. [2.9]):

```
# 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)
```

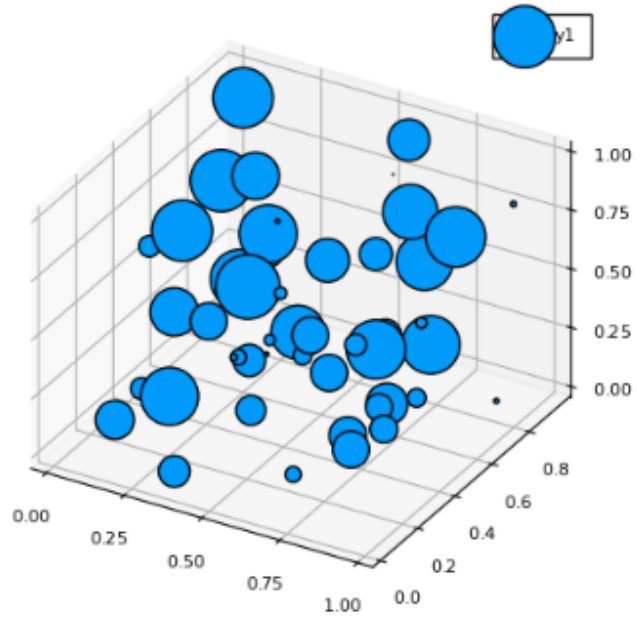

Рис. 2.9: График пятидесяти случайных значений в пространстве с различными опциями отображения (3-мерный точечный график с кодированием значения размером точки)

2.4 Аппроксимация данных

Аппроксимация — научный метод, состоящий в замене объектов их более простыми аналогами, сходными по своим свойствам.

Для демонстрации зададим искусственно некоторую функцию, в данном случае похожую по поведению на экспоненту (рис. [2.10]):

```
# 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)
```

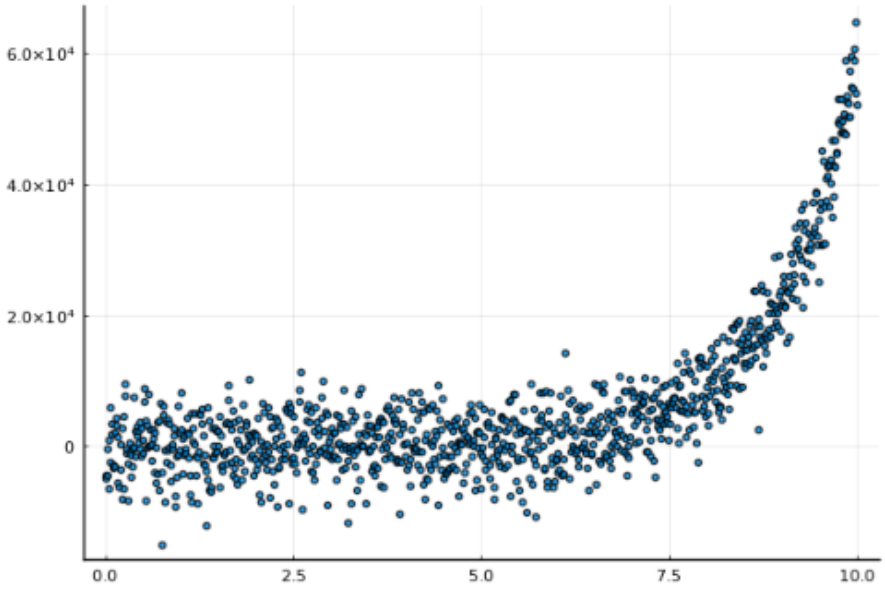

Рис. 2.10: Пример функции

Аппроксимируем полученную функцию полиномом 5-й степени (рис. [2.11]):

```
# 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)
```

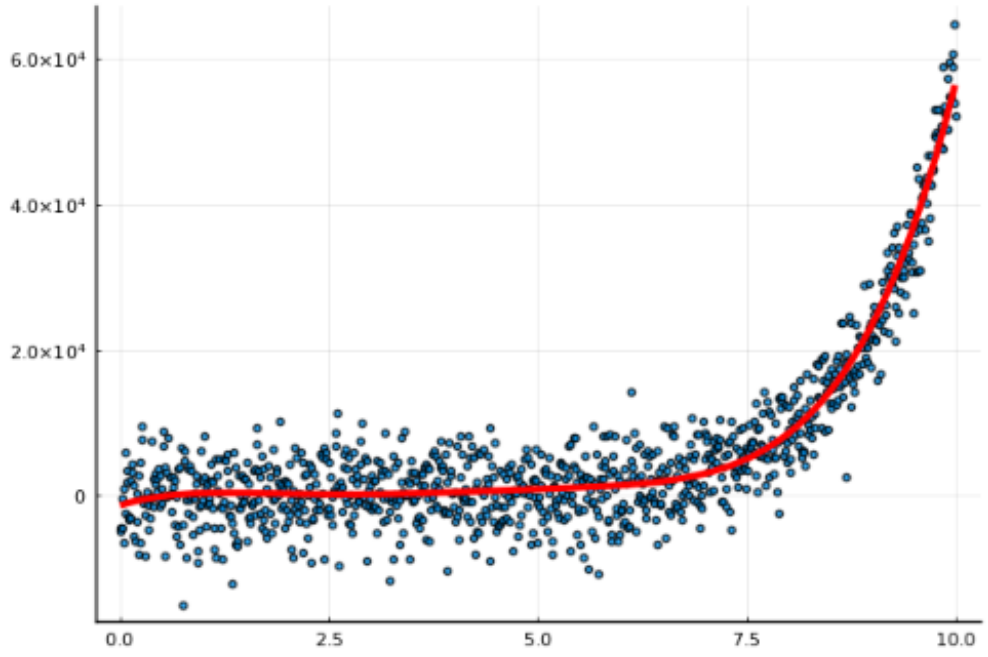

Рис. 2.11: Пример аппроксимации исходной функции полиномом 5-й степени

2.5 Две оси ординат

Иногда требуется на один график вывести несколько траекторий с существенными отличиями в значениях по оси ординат.

Пример первой траектории (рис. [2.12]):

```
# 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",
leg=:bottomright,
grid = :off,
box = :on,
# size=(600, 400)
)
```

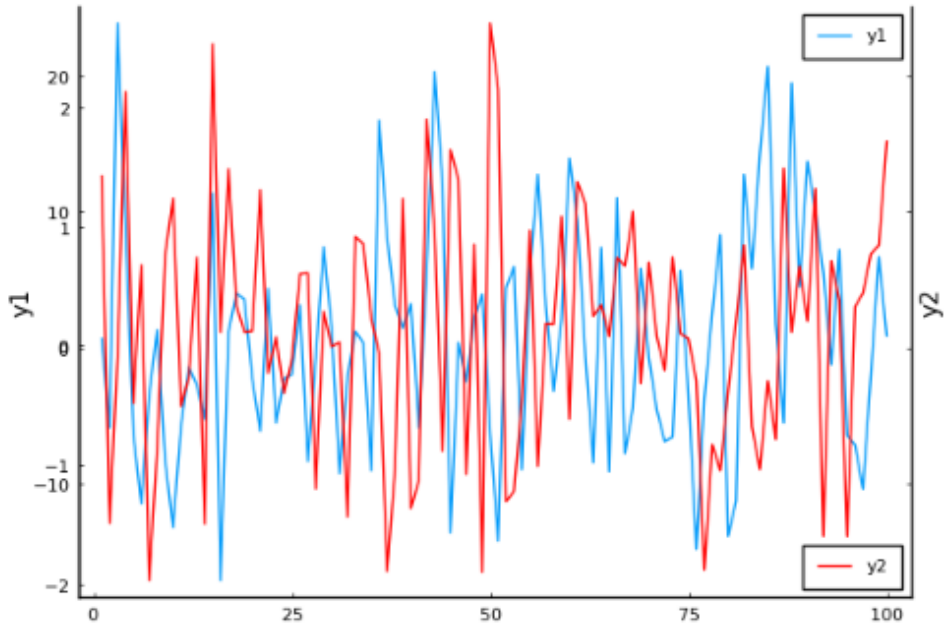

Рис. 2.12: Пример отдельно построенной траектории

2.6 Полярные координаты

Приведём пример построения графика функции в полярных координатах (рис. [2.13]):

```
# 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)
)
```

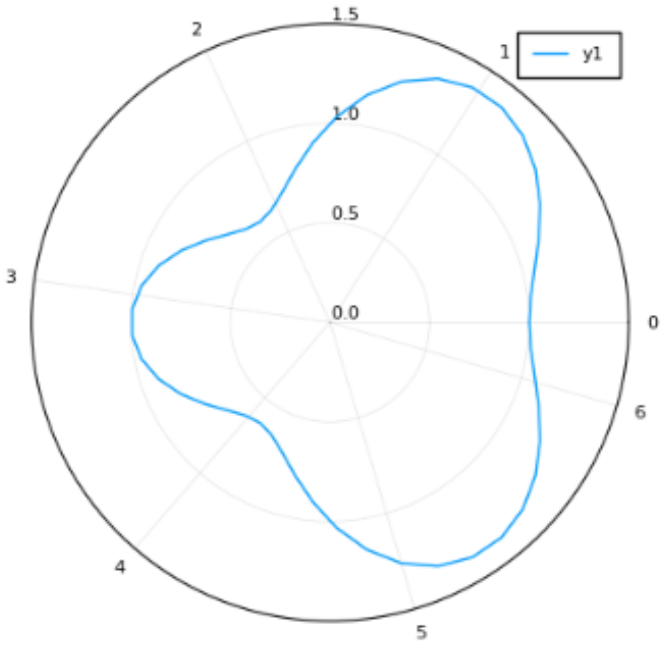

Рис. 2.13: График функции, заданной в полярных координатах

2.7 Параметрический график

Приведём пример построения графика параметрически заданной кривой на плоскости (рис. [2.14]):

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

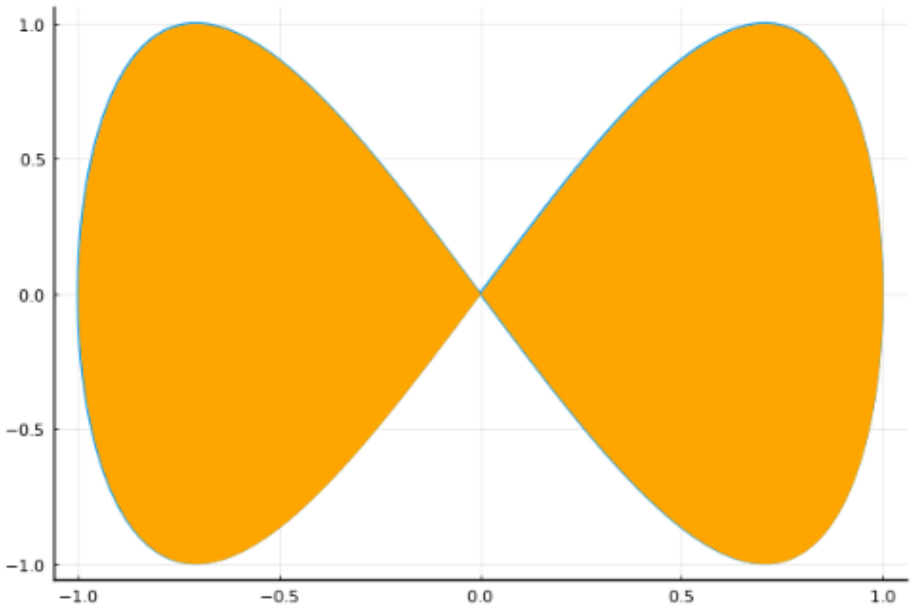

Рис. 2.14: Параметрический график кривой на плоскости

Далее приведём пример построения графика параметрически заданной кривой в пространстве (рис. [2.15]):

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

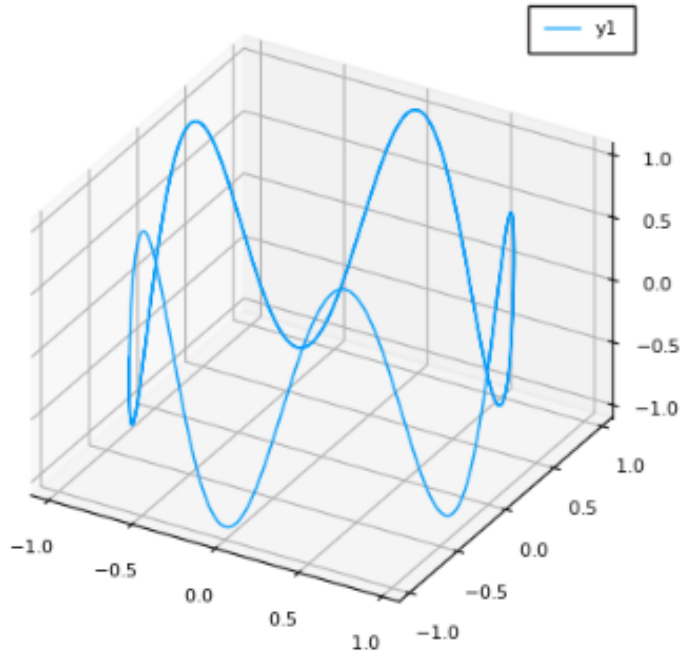

Рис. 2.15: Параметрический график кривой в пространстве

2.8 График поверхности

Для построения поверхности, заданной уравнением f(x, y) = x2 + y2, можно воспользоваться функцией surface() (рис. [2.16]):

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

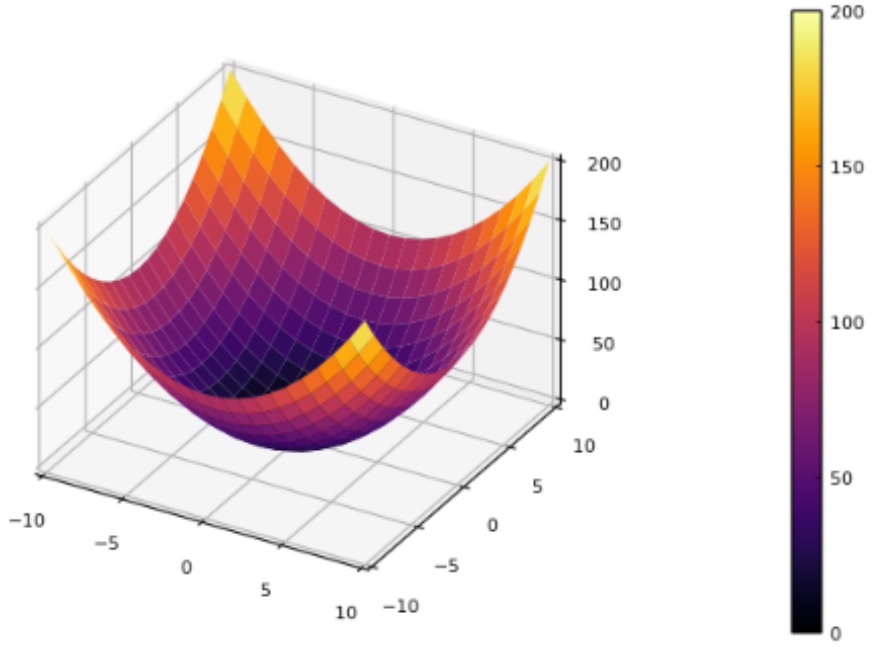

Рис. 2.16: График поверхности (использована функция surface())

Также можно воспользоваться функцией plot() с заданными параметрами (рис. [2.17]):

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

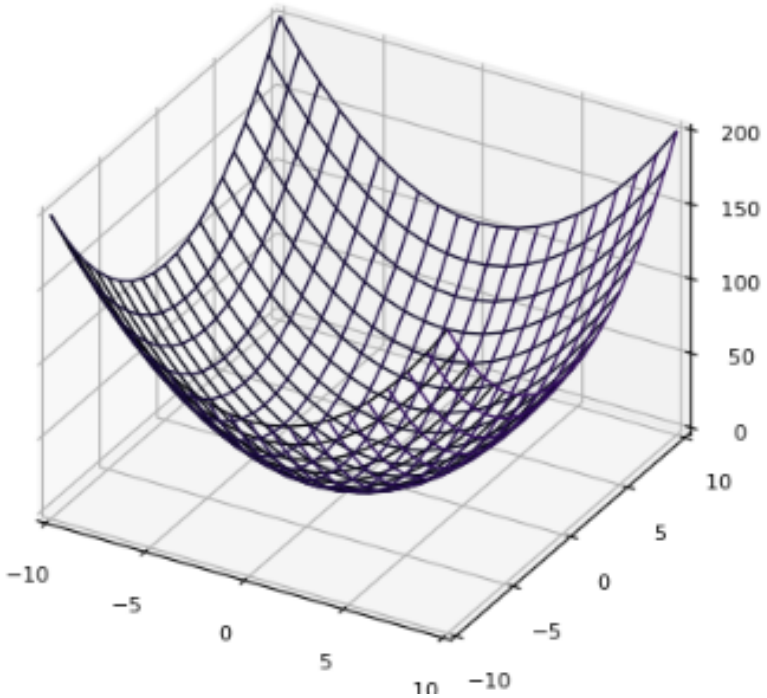

Рис. 2.17: График поверхности (использована функция plot())

Можно задать параметры сглаживания (рис. [2.18]):

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

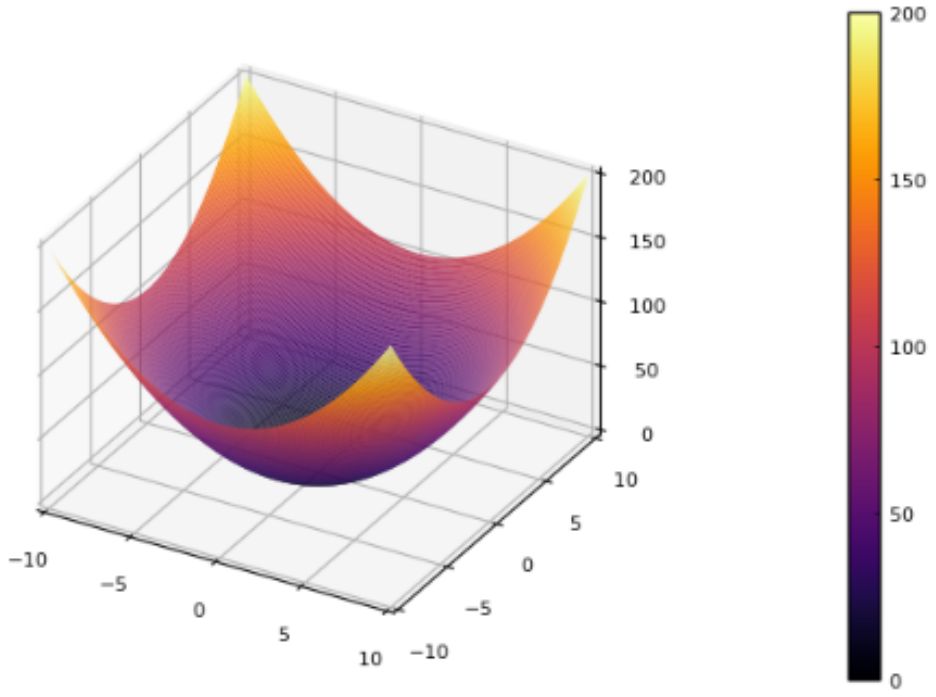

Рис. 2.18: Сглаженный график поверхности

Можно задать определённый угол зрения (рис. [2.19]):

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

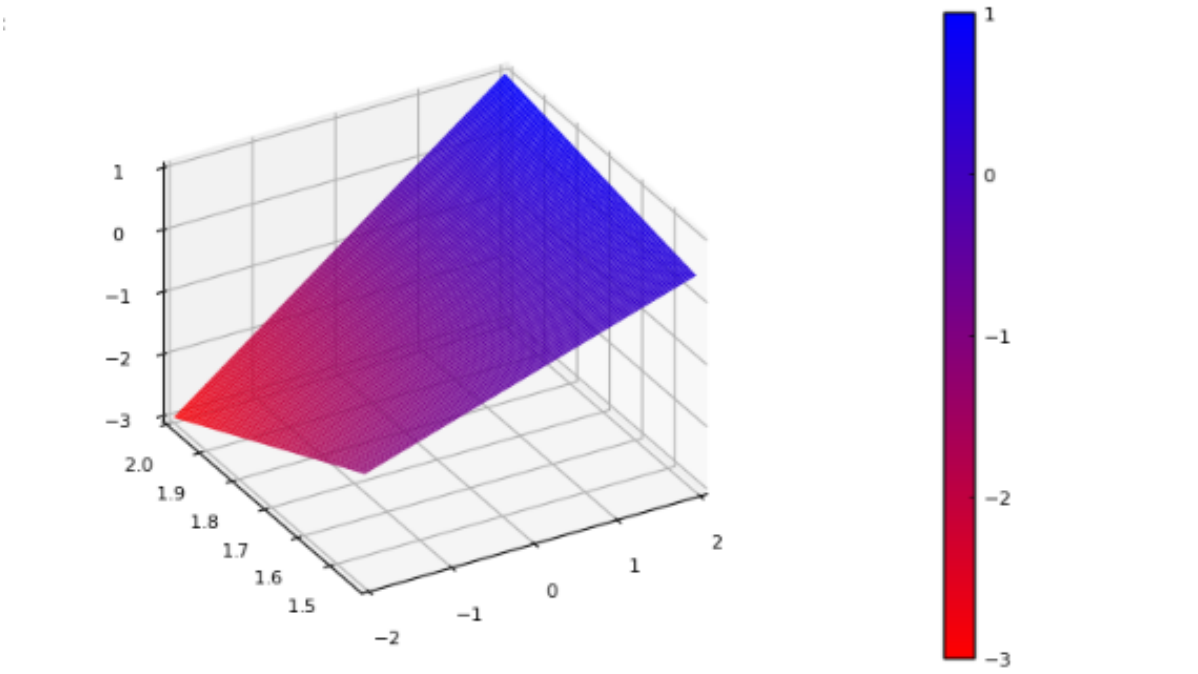

Рис. 2.19: График поверхности с изменённым углом зрения

2.9 Линии уровня

Линией уровня некоторой функции от двух переменных называется множество точек на координатной плоскости, в которых функция принимает одинаковые значения. Линий уровня бесконечно много, и через каждую точку области определения можно провести линию уровня.

С помощью линий уровня можно определить наибольшее и наименьшее значение исходной функции от двух переменных. Каждая из этих линий соответ-

ствует определённому значению высоты.

Поверхности уровня представляют собой непересекающиеся пространственные поверхности.

Рассмотрим поверхность, заданную функцией g(x, y) = (3x + y2)| sin(x) + cos(y)| (рис. [2.20]):

```
]: 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,
   )
```

]:

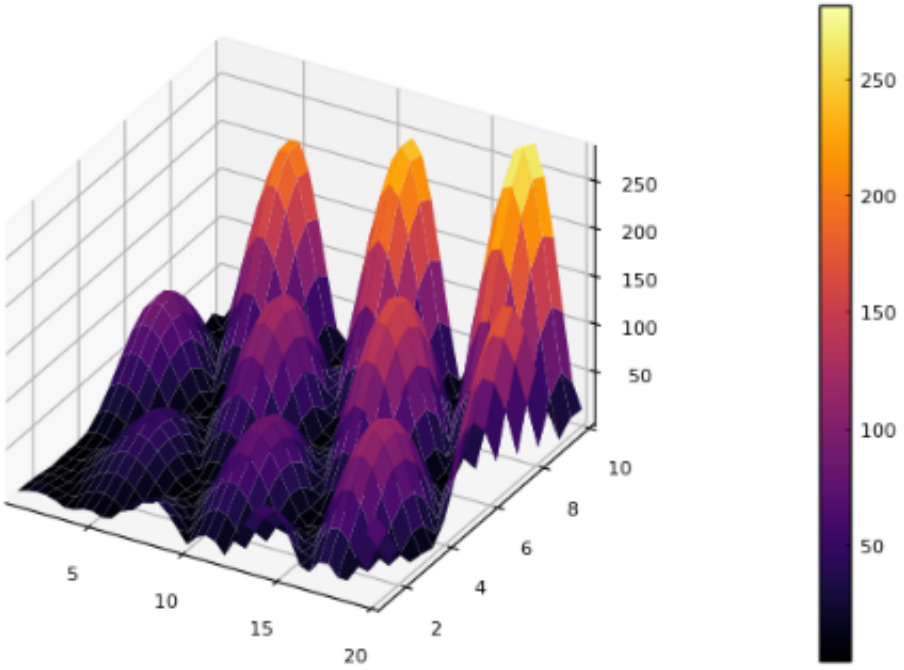

Рис. 2.20: График поверхности, заданной функцией g(x, y) = (3x + y2)| sin(x) + cos(y)|

Линии уровня можно построить, используя проекцию значений исходной функции на плоскость (рис. [2.21]):

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

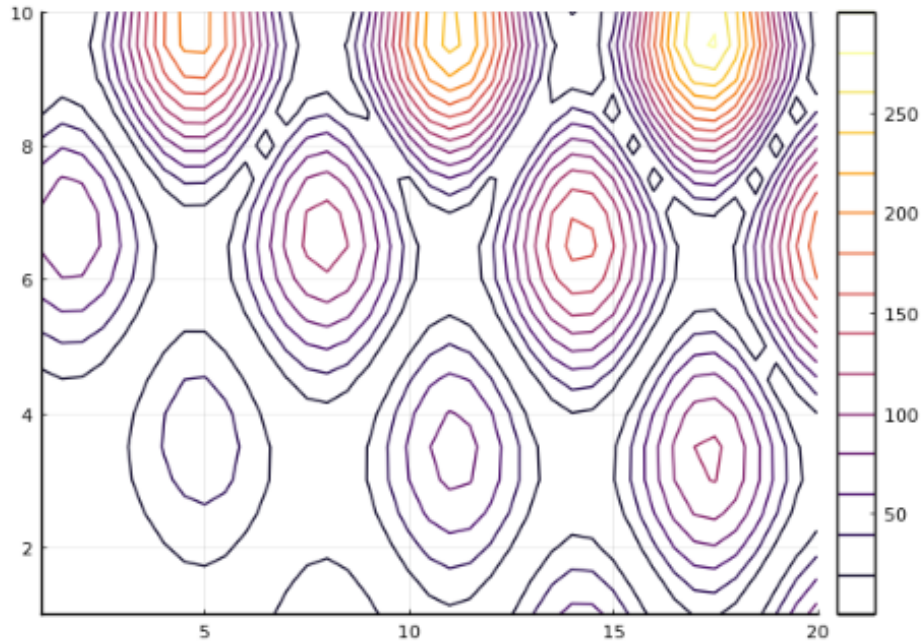

Рис. 2.21: Линии уровня

Можно дополнительно добавить заливку цветом (рис. [2.22]):

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

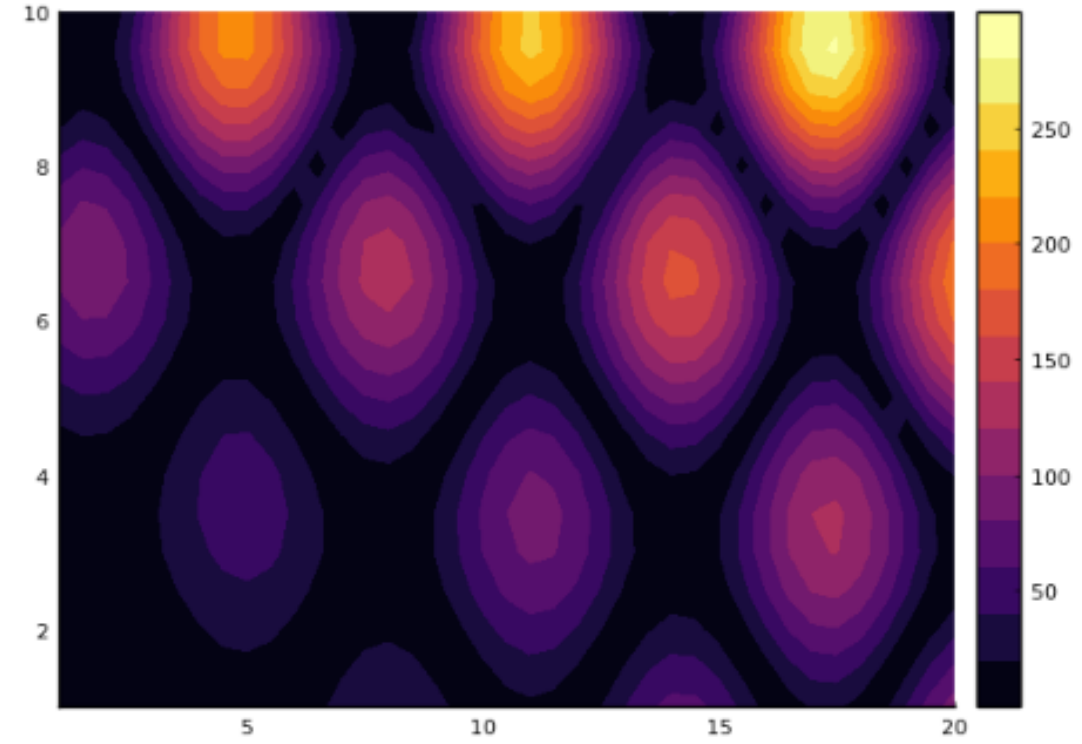

Рис. 2.22: Линии уровня с заполнением

2.10 Векторные поля

Если каждой точке некоторой области пространства поставлен в соответствие вектор с началом в данной точке, то говорят, что в этой области задано векторное поле.

Векторные поля задают векторными функциями.

Для функции h(x, y) = x3 - 3x + y2 сначала построим её график (рис. [2.23]) и линии уровня (рис. [2.24]):

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

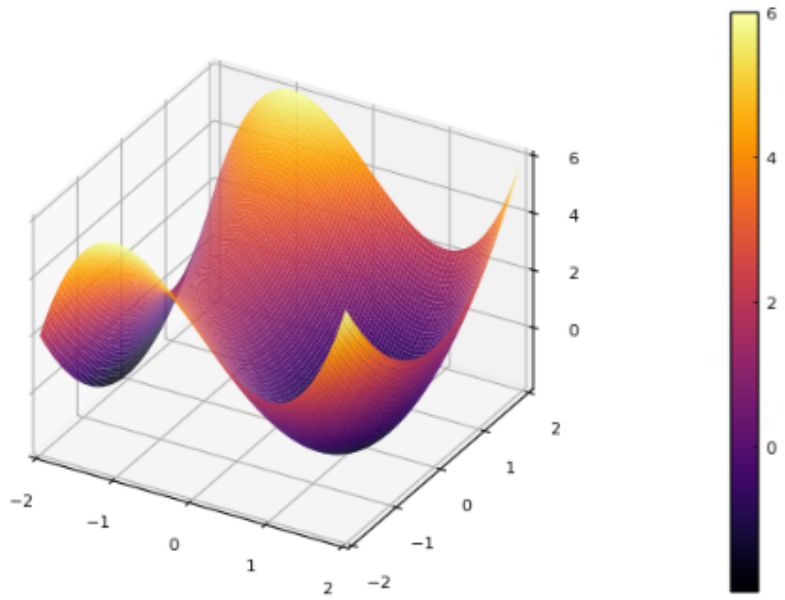

Рис. 2.23: График функции h(x, y) = x3 - 3x + y2

```
# построение линий уровня:
contour(X, Y, h)
```

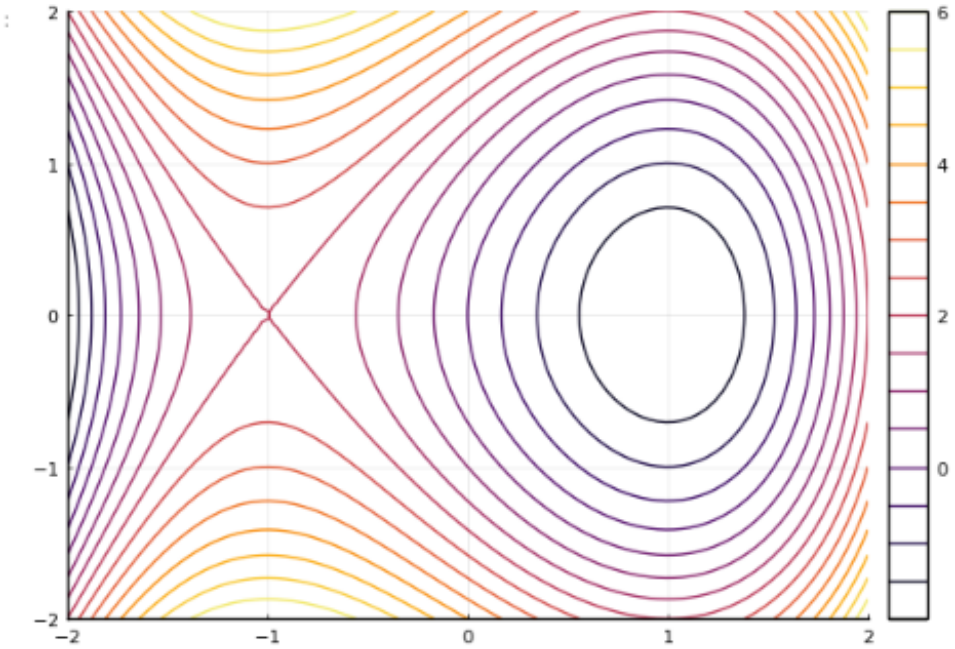

Рис. 2.24: Линии уровня для функции h(x, y) = x3 - 3x + y2

Векторное поле можно охарактеризовать векторными линиями. Каждая точка векторной линии является началом вектора поля, который лежит на касательной в данной точке.

Для нахождения векторной линии требуется решить дифференциальное уравнение.

2.11 Анимация

Технически анимированное изображение представляет собой несколько наложенных изображений (или построенных в разных точках графиках) в одном файле.

В Julia рекомендуется использовать gif-анимацию в pyplot().

Строим поверхность (рис. [2.25]):

```
# 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
```

```
[ Info: Saved animation to /home/hobo/tmp.gif
```

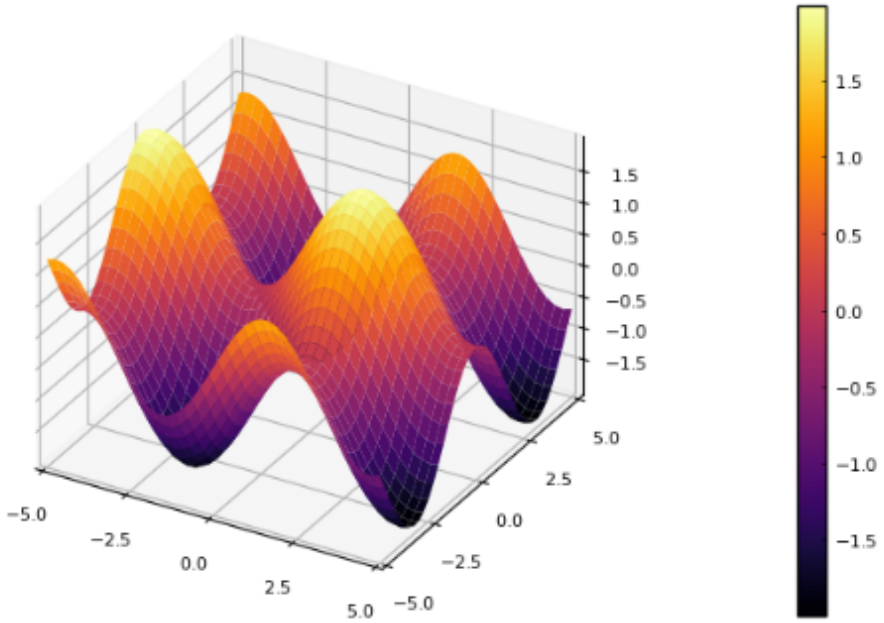

Рис. 2.25: Статичный график поверхности

Добавляем анимацию (рис. [2.26]):

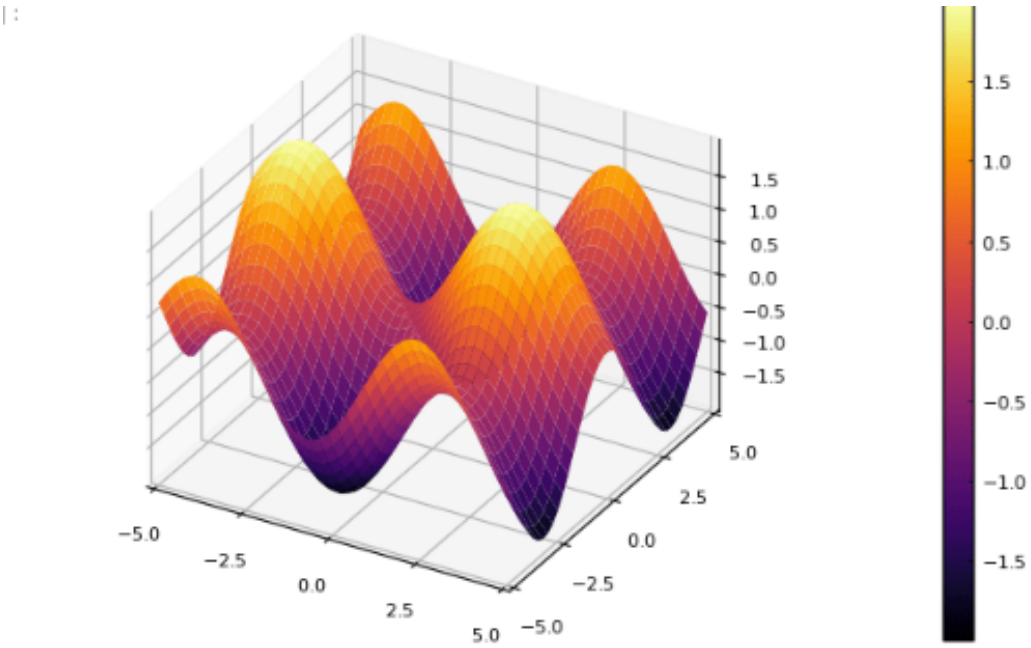

Рис. 2.26: Анимированный график поверхности

2.12 Гипоциклоида

Гипоциклоида — плоская кривая, образуемая точкой окружности, катящейся по внутренней стороне другой окружности без скольжения.

Построим большую окружность (рис. [2.27]):

```
# радиус малой окружности:
r1 = 1
# коэффициент для построения большой окружности:
k = 3
# число отсчётов:
n = 100
# массив значений угла θ:
# theta from θ 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)
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)
```

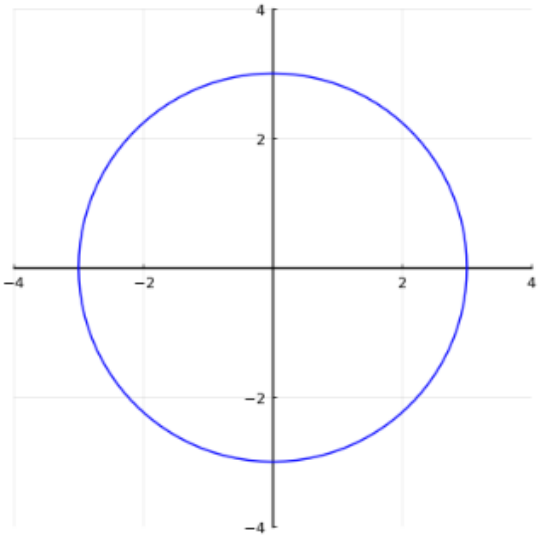

Рис. 2.27: Большая окружность гипоциклоиды

Для частичного построения гипоциклоиды будем менять параметр t (рис. [2.28]):

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

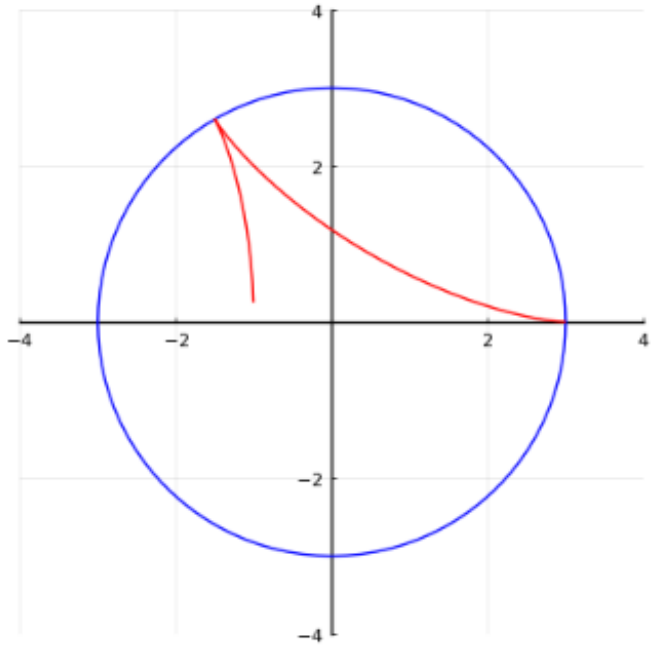

Рис. 2.28: Половина пути гипоциклоиды

Добавляем малую окружность гипоциклоиды (рис. [2.29]):

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

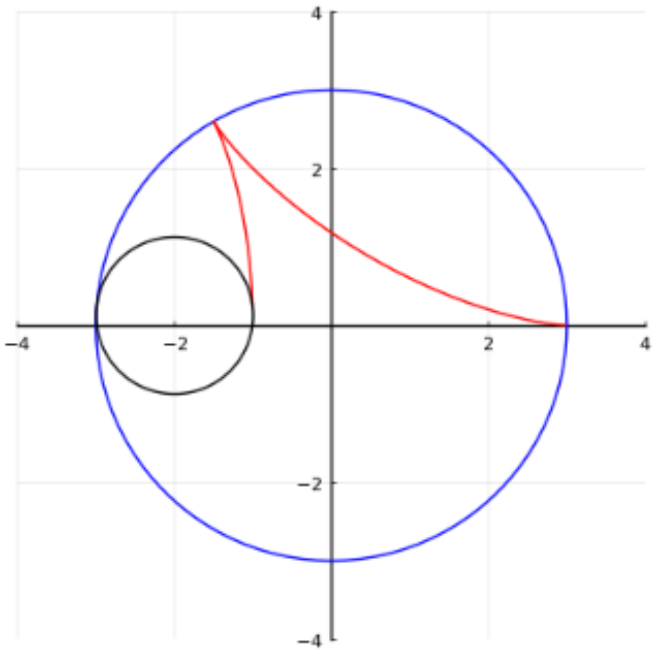

Рис. 2.29: Малая окружность гипоциклоиды

Добавим радиус для малой окружности (рис. [2.30]):

```
# 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)
```

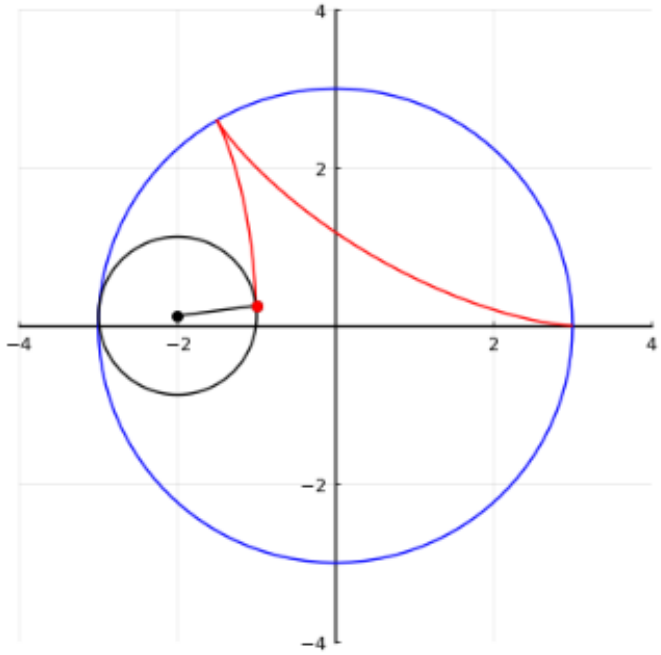

Рис. 2.30: Малая окружность гипоциклоиды с добавлением радиуса

В конце сделаем анимацию получившегося изображения (рис. [2.31]):

```
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")
```

```
[ Info: Saved animation to /home/hobo/hypocycloid.gif
```

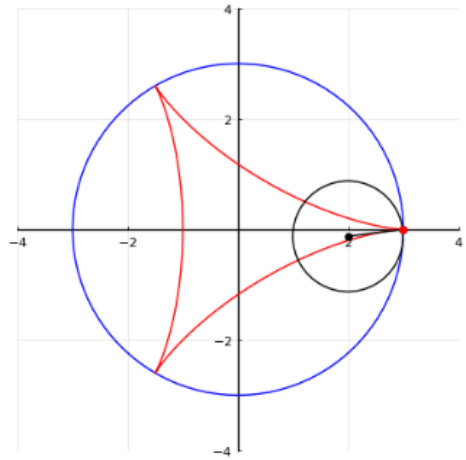

Рис. 2.31: Малая окружность гипоциклоиды с добавлением радиуса

2.13 Errorbars

В исследованиях часто требуется изобразить графики погрешностей измерения.

Построим график исходных значений (рис. [2.32]):

```
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),
)
```

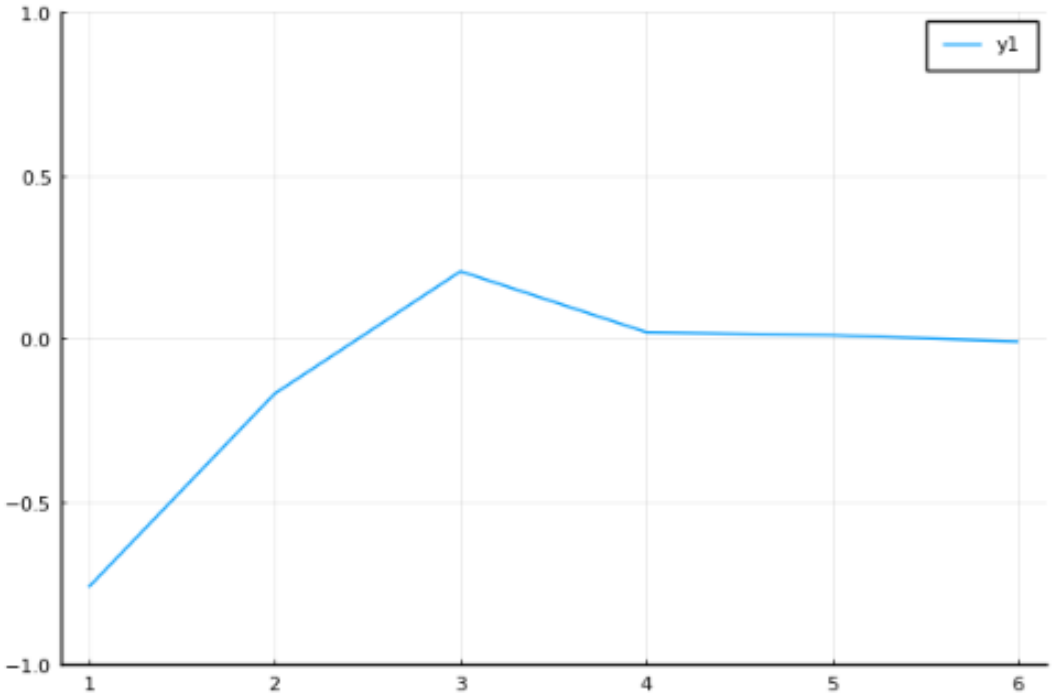

Рис. 2.32: График исходных значений

Построим график отклонений от исходных значений (рис. [2.33]):

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

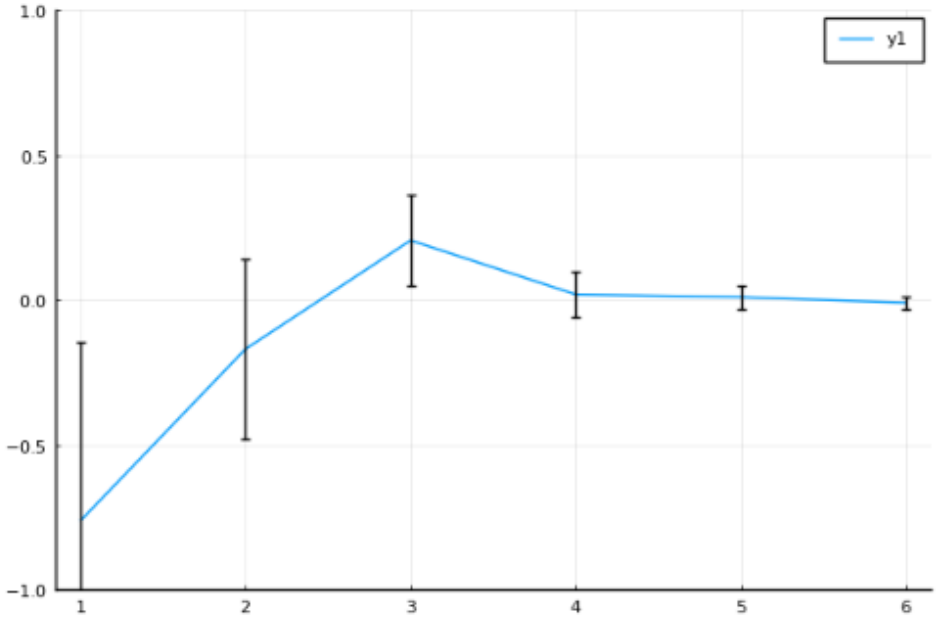

Рис. 2.33: График исходных значений с отклонениями

Повернём график (рис. [2.34]):

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

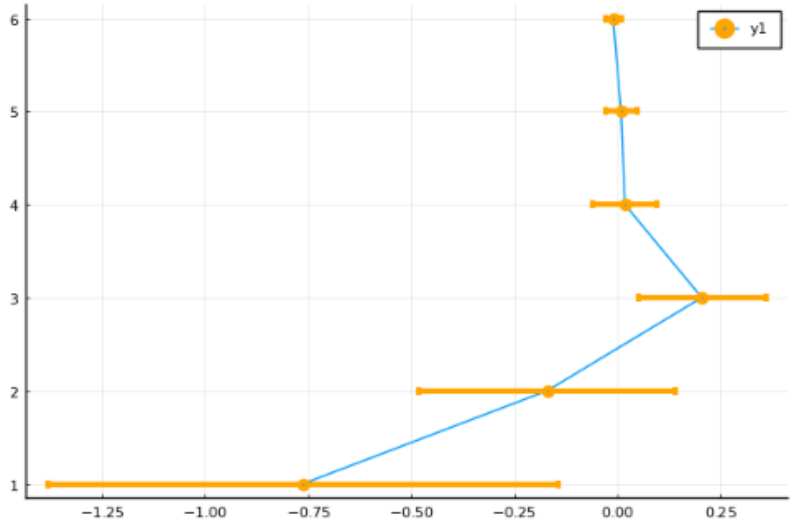

Рис. 2.34: Поворот графика

Заполним область цветом (рис. [2.35]):

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

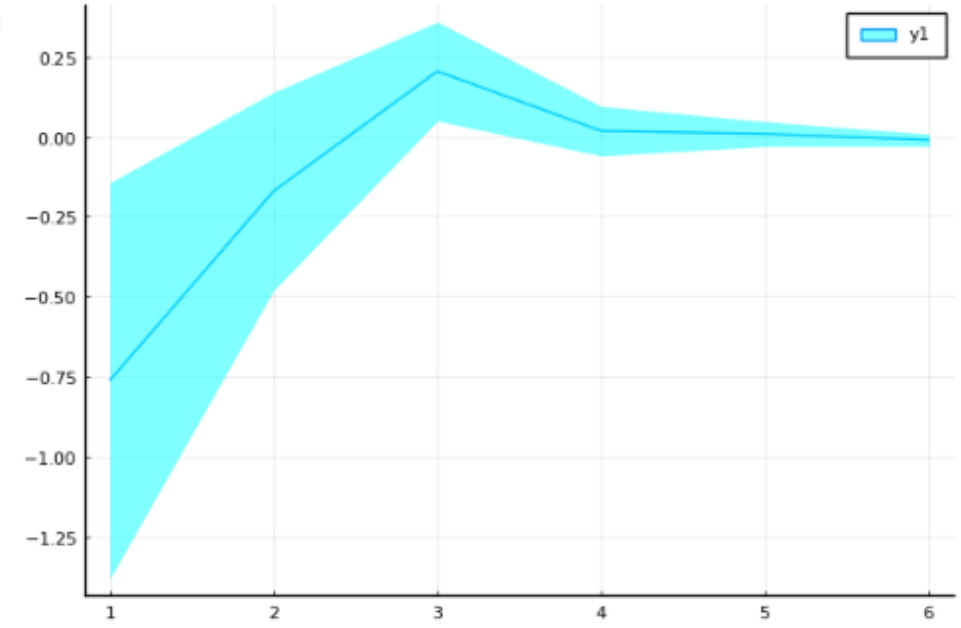

Рис. 2.35: Заполнение цветом

Можно построить график ошибок по двум осям (рис. [2.36]):

```
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)
)
```

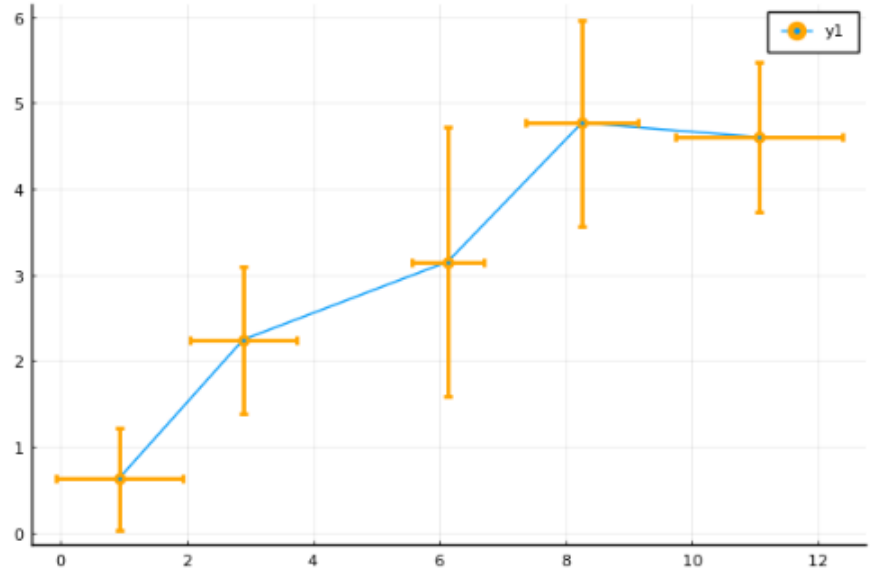

Рис. 2.36: График ошибок по двум осям

Можно построить график асимметричных ошибок по двум осям (рис. [2.37]):

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

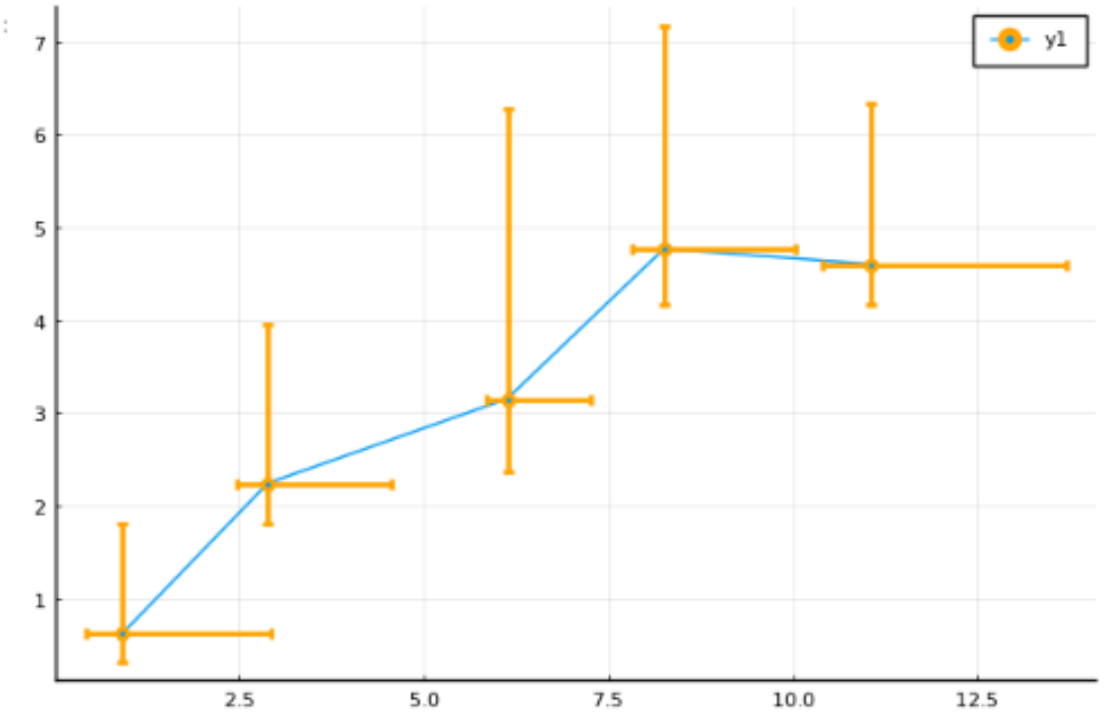

Рис. 2.37: График асимметричных ошибок по двум осям

2.14 Использование пакета Distributions

Строим гистограмму.

Задаём нормальное распределение и строим гистограмму.

Далее применим для построения нескольких гистограмм распределения людей по возрастам на одном графике plotly().

Ничего из этого не получилось сделать.

2.15 Подграфики

Определим макет расположения графиков. Команда layout принимает кортеж layout = (N, M), который строит сетку графиков NxM. Например, если задать layout = (4,1) на графике четыре серии, то получим четыре ряда графиков (рис. [2.38]):

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

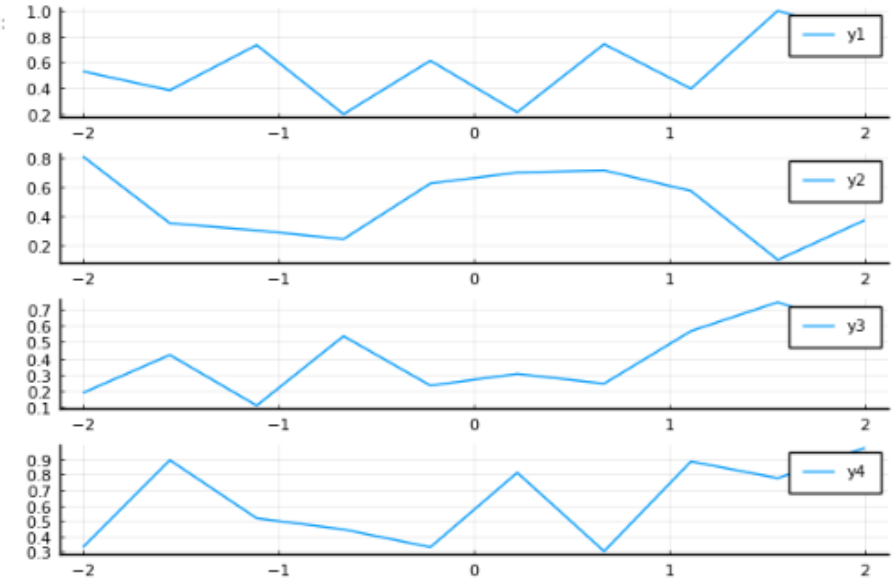

Рис. 2.38: Серия из 4-х графиков в ряд

Для автоматического вычисления сетки необходимо передать layout целое число (рис. [2.39]):

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

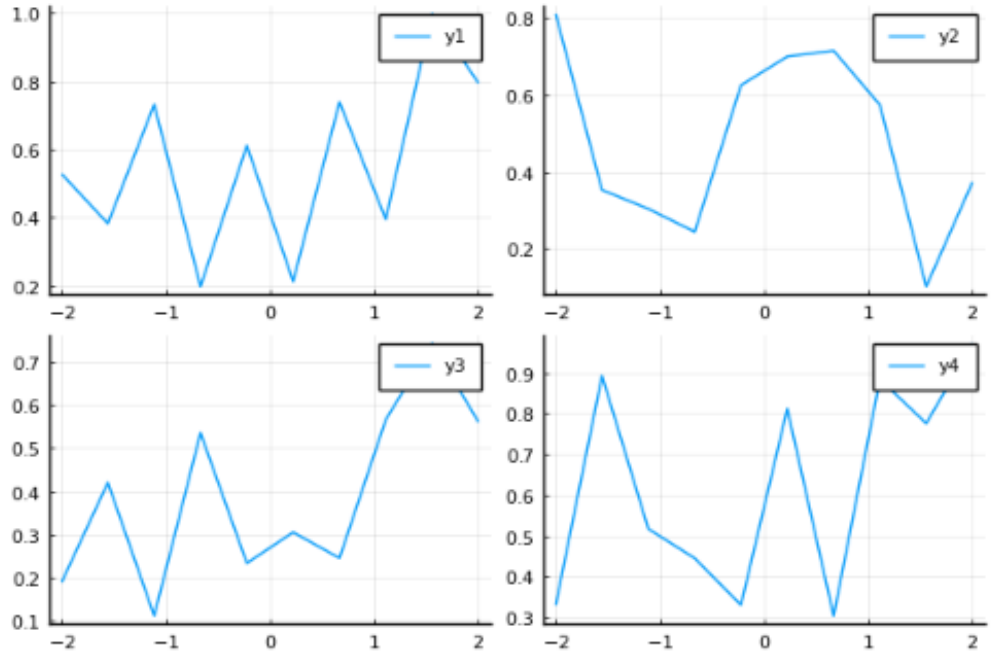

Рис. 2.39: Серия из 4-х графиков в сетке

Аргумент heights принимает в качестве входных данных массив с долями желаемых высот. Если в сумме дроби не составляют 1,0, то некоторые подзаголовки могут отображаться неправильно.

Можно сгенерировать отдельные графики и объединить их в один, например, в сетке 2 × 2 (рис. [2.40]):

```
plot(x,y,
    size=(600,300),
    layout = grid(4,1,heights=[0.2,0.3,0.4,0.1])
)
```

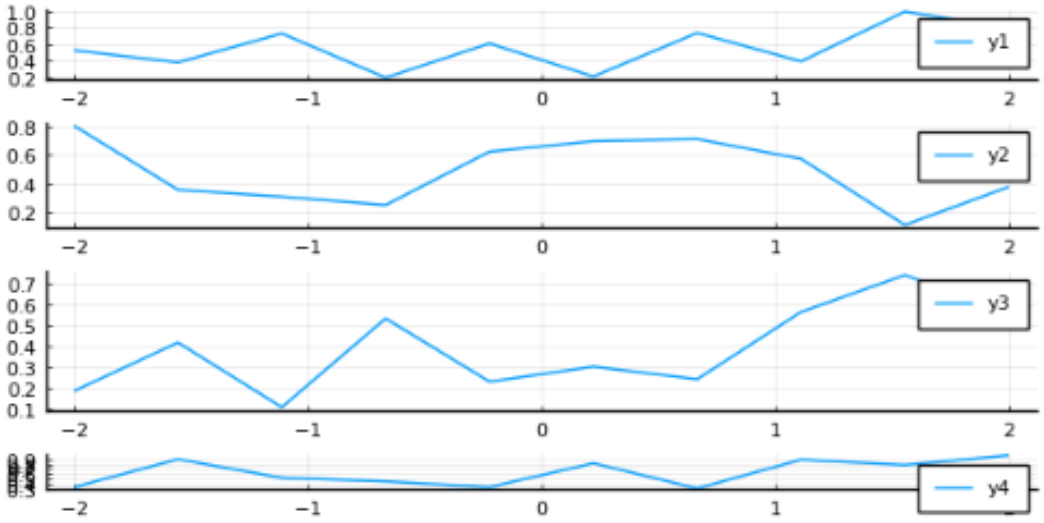

Рис. 2.40: Объединение нескольких графиков в одной сетке

Обратите внимание, что атрибуты на отдельных графиках применяются к отдельным графикам, в то время как атрибуты в последнем вызове plot применяются ко всем графикам.

Разнообразные варианты представления данных (рис. [2.41]):

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

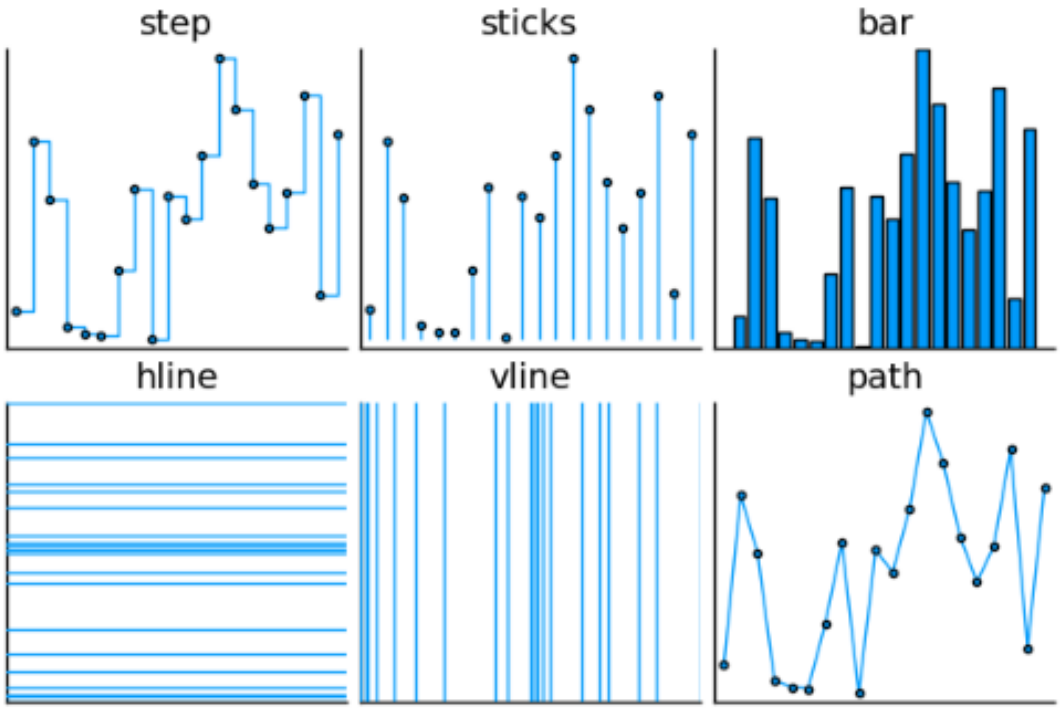

Рис. 2.41: Разнообразные варианты представления данных

Применение макроса [**layout?**] наиболее простой способ определения сложных макетов. Точные размеры могут быть заданы с помощью фигурных скобок, в противном случае пространство будет поровну разделено между графиками (рис. [2.42]):

```
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)
)
```

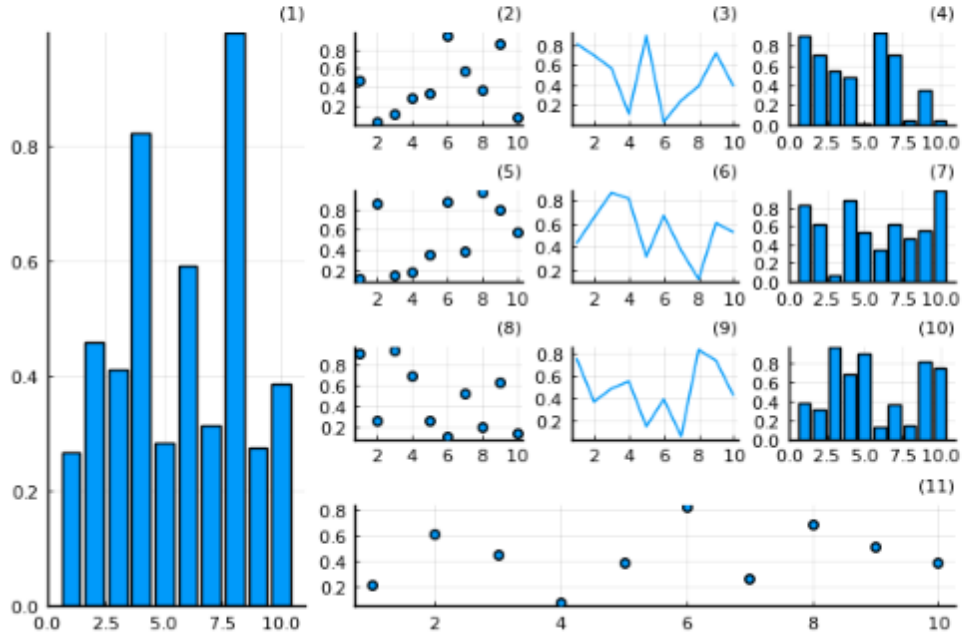

Рис. 2.42: Демонстрация применения сложного макета для построения графиков

3 Вывод

В ходе выполнения лабораторной работы был освоен синтаксис языка Julia для построения графиков.

Список литературы