

Лабораторная работа № 5. Построение графиков

5.1. Цель работы

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

5.2. Предварительные сведения

5.2.1. 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`.

Перед использованием графических пакетов следует их установить и подключить в Julia:

```
using Pkg
Pkg.add("Plots")
Pkg.add("PyPlot")
Pkg.add("Plotly")
Pkg.add("UnicodePlots")
# подключаем для использования Plots:
using Plots
```

Одним из преимуществ `Plots.jl` является то, что он позволяет легко менять вызовы библиотек работы с графикой: `gr()`, `pyplot()`, `plotlyjs()`.

Далее рассмотрим построение графика функции $f(x) = (3x^2 + 6x - 9)e^{-0.3x}$ разными способами. Фактически для построения графика требуется иметь массив соответствующих значений x и y .

Зададим исходную функцию (используются поэлементные операции над векторами):

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

Определим массив значений x :

```
# 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))
```

Определим массив значений y :

```
# генерирование массива значений y:
y = f(x)
```

Далее показано сравнение построения графиков при использовании разных программно-аппаратных частей `Plots.jl`.

По умолчанию используется `gr()`:

```
# указывается, что для построения графика используется gr():
gr()
```

Стандартным образом можно задать различные опции при построении графика:

```
# 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")
```

В результате получим график исходной функции (рис. 5.1).

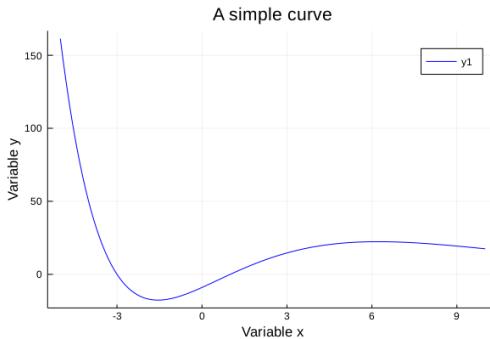


Рис. 5.1. График функции $f(x) = (3x^2 + 6x - 9)e^{-0.3x}$, построенный при помощи gr()

Если требуется, чтобы на графике были надписи на русском языке, то лучше воспользоваться pyplot() (рис. 5.2):

```
# указывается, что для построения графика используется pyplot():
pyplot()
```

Коррекция содержания опций при построении графика:

```
# 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")
```

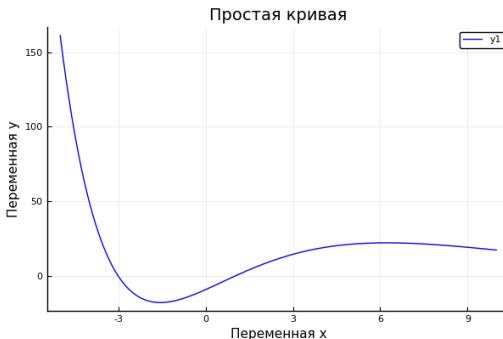


Рис. 5.2. График функции $f(x) = (3x^2 + 6x - 9)e^{-0.3x}$, построенный при помощи pyplot()

В качестве упражнения постройте график функции $f(x) = (3x^2 + 6x - 9)e^{-0.3x}$ при помощи `plotly()` и `unicodeplots()`.

5.2.2. Опции при построении графика

Далее на примере графика функции $\sin(x)$ и графика разложения этой функции в ряд Тейлора:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}, \quad x \in \mathbb{C}$$

рассмотрим дополнительные возможности пакетов для работы с графикой.

Используем `pyplot()`:

```
# указывается, что для построения графика используется pyplot():
pyplot()
```

Задаём функцию:

```
# задание функции sin(x):
sin_theor(x) = sin(x)
```

Строим график функции (рис. 5.3):

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

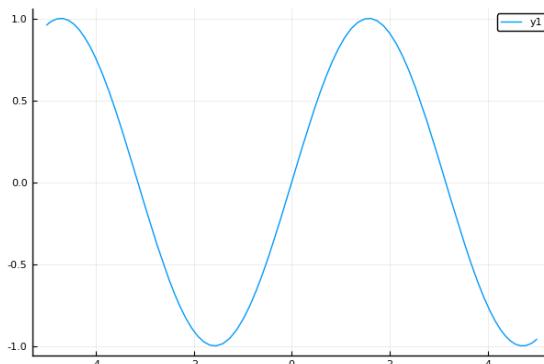


Рис. 5.3. График функции $\sin(x)$

Задаём разложение исходной функции в ряд Тейлора:

```
# 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
```

Строим график разложения исходной функции в ряд Тейлора (рис. 5.4):

```
# построение графика функции sin_taylor(x):
plot(sin_taylor)
```

Выводим две функции на один график (рис. 5.5):

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

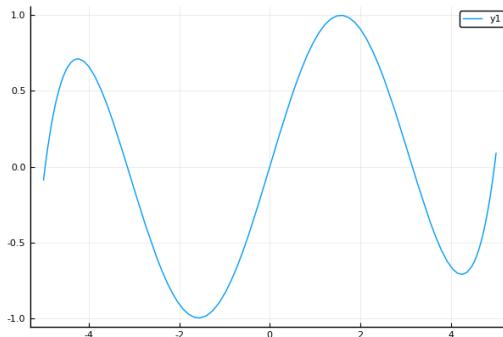


Рис. 5.4. График функции разложения исходной функции в ряд Тейлора

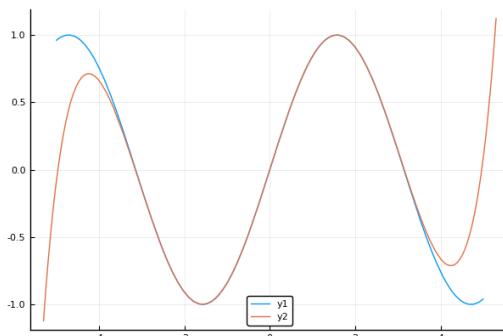


Рис. 5.5. Графики исходной функции и её разложения в ряд Тейлора

Далее добавим различные опции для отображения на графике (рис. 5.6).

```
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))

```

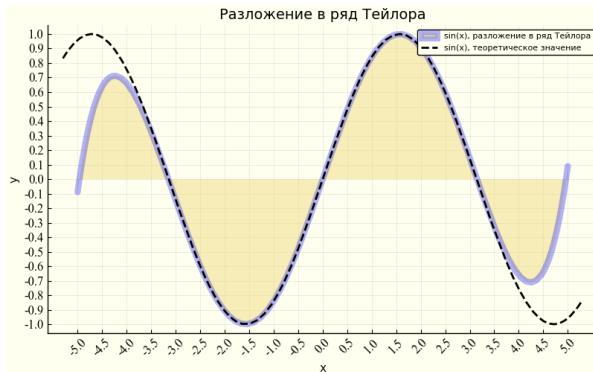


Рис. 5.6. Вид графиков из рис. 5.5 после добавления опций при их построении

Для сохранения построенного графика в определённом формате можно использовать функцию `savefig()`:

```

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

```

5.2.3. Точечный график

Графики в виде точек на плоскости или в пространстве часто используются в статистических исследованиях.

5.2.3.1. Простой точечный график

Как и построении обычного графика для точечного графика необходимо задать массив значений x , посчитать или задать значения y , задать опции построения графика:

```
# 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"
)
```

В результате получим график случайных десяти значений на плоскости (рис. 5.7).

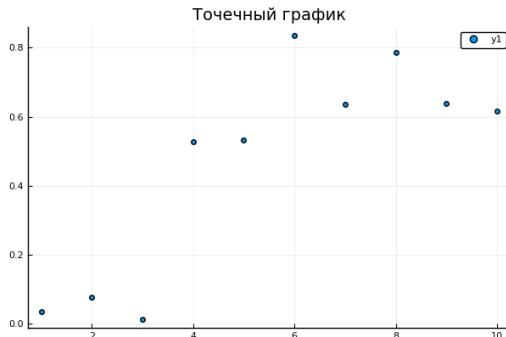


Рис. 5.7. График десяти случайных значений на плоскости

5.2.3.2. Точечный график с кодированием значения размером точки

Для точечного графика можно задать различные опции, например размер маркера, его тип, цвет и т.п. (рис. 5.8).

```
# 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)
```

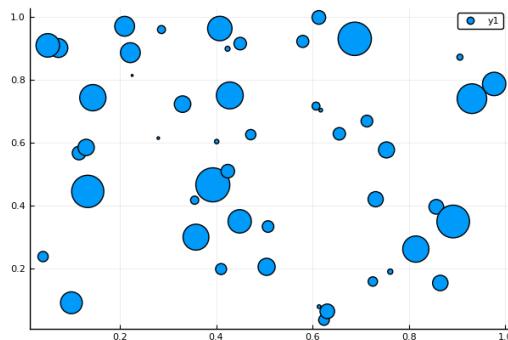


Рис. 5.8. График пятидесяти случайных значений на плоскости с различными опциями отображения

5.2.3.3. 3-мерный точечный график с кодированием значения размером точки

Можно строить и 3-мерные точечные графики (рис. 5.9)

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)

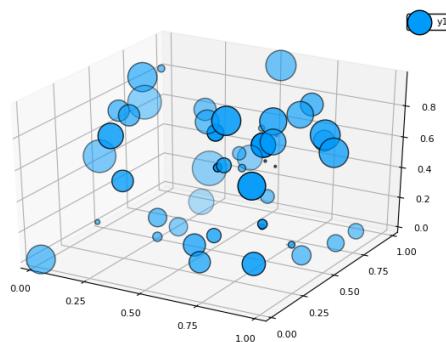


Рис. 5.9. График пятидесяти случайных значений в пространстве с различными опциями отображения

В качестве упражнения дополните код построения точечных графиков, подписьав оси, добавив легенду и название.

5.2.4. Аппроксимация данных

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

Пусть на некотором отрезке в точках $x_0, x_1, x_2, \dots, x_N$ известны значения некоторой функции $f(x)$, а именно $y_0, y_1, y_2, \dots, y_N$. Требуется определить параметры a_i многочлена вида $F(x) = a_0 + a_1x + a_2x^2 + \dots + a_kx^k$, где $k < N$ такое, что сумма квадратов отклонений значений y от значений функции $F(y)$ в заданных точках x минимальна, т.е.

$$S = \sum_0^N [y_i - F(x_i, a_0, a_1, \dots, a_k)]^2 \rightarrow \min$$

Геометрически это означит, что нужно найти кривую $y = F(x)$, полином которой проходит как можно ближе к каждой из заданных точек.

Такая задача может быть решена, если решить систему уравнений вида: $A\vec{x} = \vec{y}$, где A — матрица коэффициентов многочлена $F(x)$, \vec{x}, \vec{y} — вектора соответствующих значений.

Для демонстрации зададим искусственно некоторую функцию, в данном случае похожую по поведению на экспоненту (рис. 5.10):

```
# 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)
```

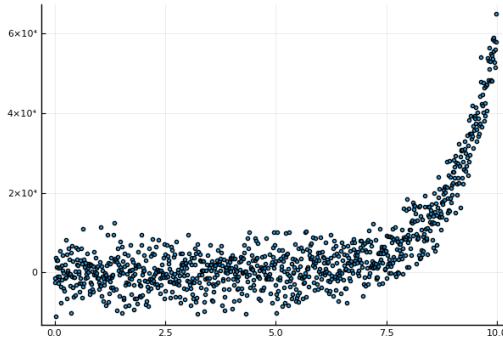


Рис. 5.10. Пример функции

Аппроксимируем полученную функцию полиномом 5-й степени (рис. 5.11):

```
# 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:
f = 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,f,linewidth=3, color=:red)
```

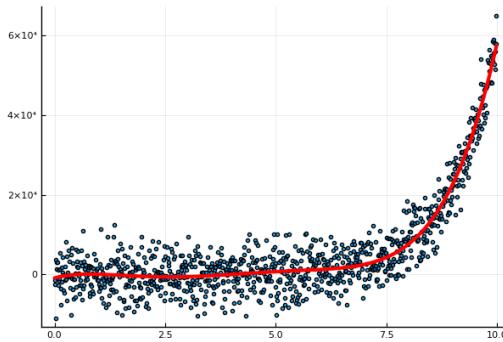


Рис. 5.11. Пример аппроксимации исходной функции полиномом 5-й степени

5.2.5. Две оси ординат

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

Пример первой траектории (рис. 5.12):

```
# пример случайной траектории
# (заданы обозначение траектории, легенда вверху справа, без сетки)
plot(randn(100),
      ylabel="y1",
      leg=:topright,
      grid = :off,
    )
```

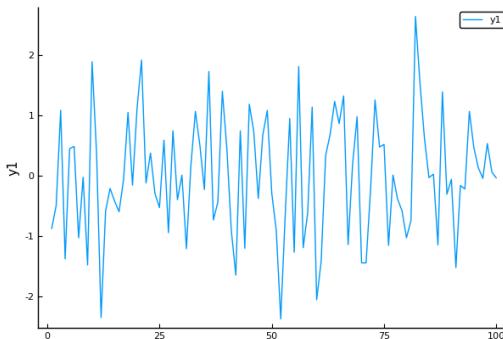


Рис. 5.12. Пример отдельно построенной траектории

Далее на существующий график добавляется вторая траектория (рис. 5.13) с дополнительными элементами для улучшения восприятия информации:

```
# 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)
    )
```

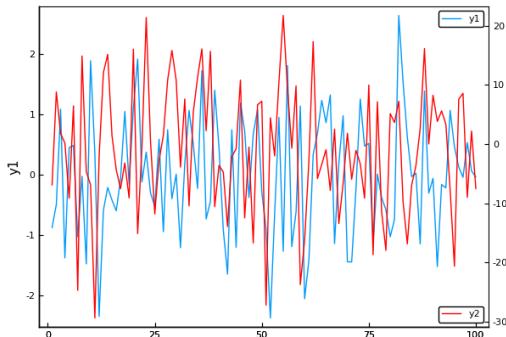


Рис. 5.13. Пример двух траекторий на одном графике с двумя осями ординат

5.2.6. Полярные координаты

Приведём пример построения графика функции

$$r(\vartheta) = 1 + \cos \vartheta \sin^2 \vartheta$$

в полярных координатах (рис. 5.14):

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

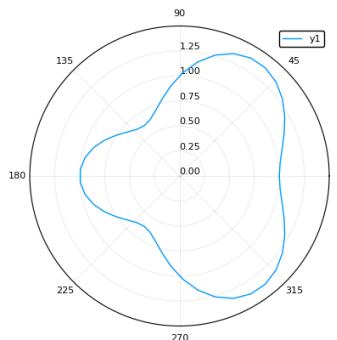


Рис. 5.14. График функции, заданной в полярных координатах

5.2.7. Параметрический график

При параметрическом представлении графика некоторой функции координаты на графике задаются как функции от некоторого набора свободных параметров. В случае одного параметра получим параметрическое уравнение кривой. Выражая координаты точек поверхности через два свободных параметра, получим параметрическое задание поверхности.

5.2.7.1. Параметрический график кривой на плоскости

Приведём пример построения графика параметрически заданной кривой на плоскости (рис. 5.15). Кривая задана выражениями $x(t) = \sin(t)$, $y(t) = \sin(2t)$, $0 \leq t \leq 2\pi$:

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

5.2.7.2. Параметрический график кривой в пространстве

Приведём пример построения графика параметрически заданной кривой в пространстве (рис. 5.16). Кривая задана выражениями $x(t) = \cos(t)$, $y(t) = \sin(t)$, $z(t) = \sin(5t)$:

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

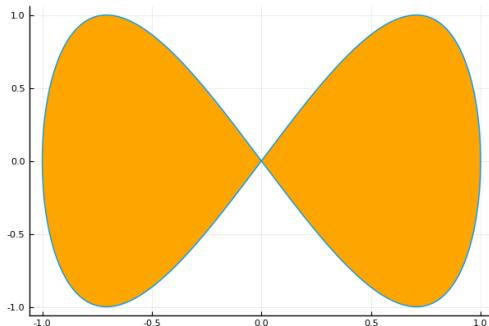


Рис. 5.15. Параметрический график кривой на плоскости

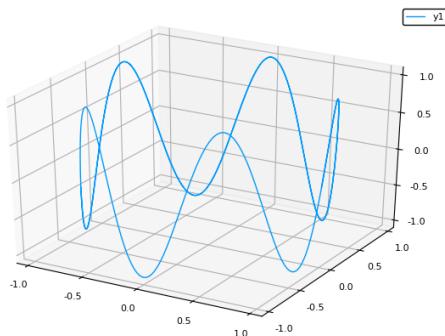


Рис. 5.16. Параметрический график кривой в пространстве

5.2.8. График поверхности

Для построения поверхности, заданной уравнением $f(x, y) = x^2 + y^2$, можно воспользоваться функцией `surface()` (рис. 5.17):

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

Также можно воспользоваться функцией `plot()` с заданными параметрами (рис. 5.18):

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

Можно задать параметры сглаживания (рис. 5.19):

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

Можно задать определённый угол зрения (рис. 5.20):

```
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),
)
```

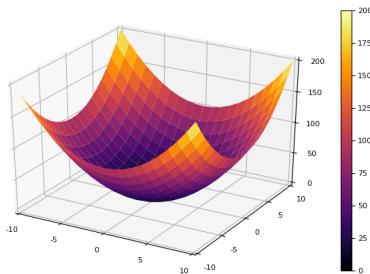


Рис. 5.17. График поверхности
(использована функция surface())

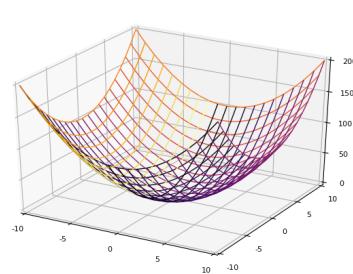


Рис. 5.18. График поверхности
(использована функция plot())

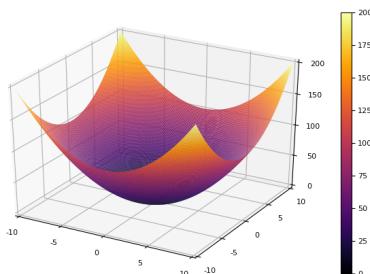


Рис. 5.19. Сглаженный график поверхности

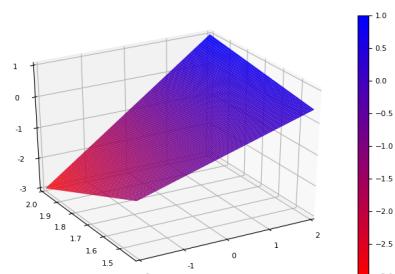


Рис. 5.20. График поверхности
с изменённым углом зрения

5.2.9. 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 соответствует определённому значению высоты.

Поверхности уровня представляют собой непересекающиеся пространственные поверхности.

Рассмотрим поверхность, заданную функцией $g(x, y) = (3x + y^2)|\sin(x) + \cos(y)|$ (рис. 5.21):

```
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,
)
```

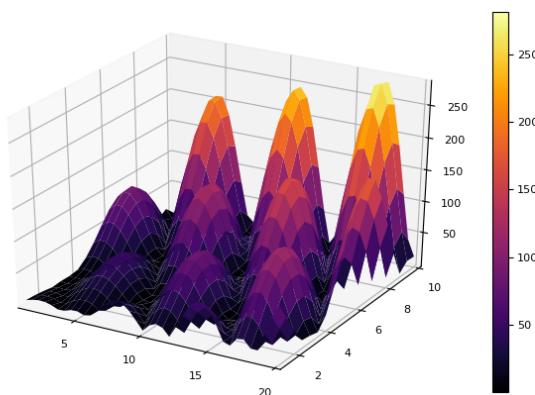


Рис. 5.21. График поверхности, заданной функцией $g(x, y) = (3x + y^2)|\sin(x) + \cos(y)|$

Линии уровня можно построить, используя проекцию значений исходной функции на плоскость (рис. 5.22):

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

Можно дополнительно добавить заливку цветом (рис. 5.23):

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

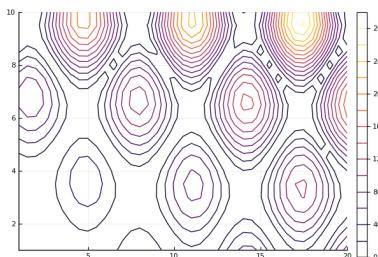


Рис. 5.22. Линии уровня

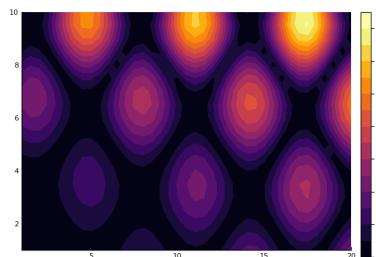


Рис. 5.23. Линии уровня с заполнением

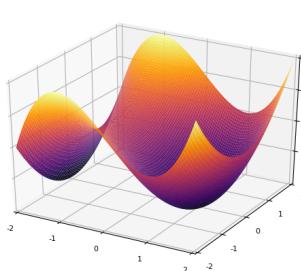
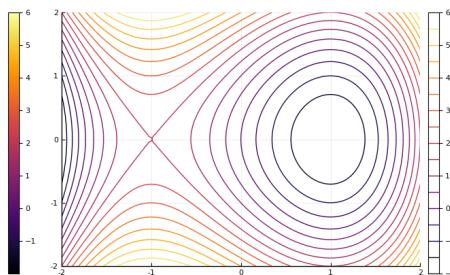
5.2.10. Векторные поля

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 $h(x, y) = x^3 - 3x + y^2$ сначала построим её график (рис. 5.24) и линии уровня (рис. 5.25):

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

Рис. 5.24. График функции
 $h(x, y) = x^3 - 3x + y^2$ Рис. 5.25. Линии уровня для функции
 $h(x, y) = x^3 - 3x + y^2$

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.

Для заданной функции построим векторное поле (рис. 5.26, рис. 5.27):

```
# градиент:
x = range(-2, stop=2, length=12)
y = range(-2, stop=2, length=12)
# производная от исходной функции:
dh(x, y) = [3x^2 - 3; 2y] / 25
# построение векторного поля:
quiver!(x, y', quiver=dh, c=:blue)
# коррекция области видимости графика:
xlims!(-2, 2)
ylims!(-2, 2)
```

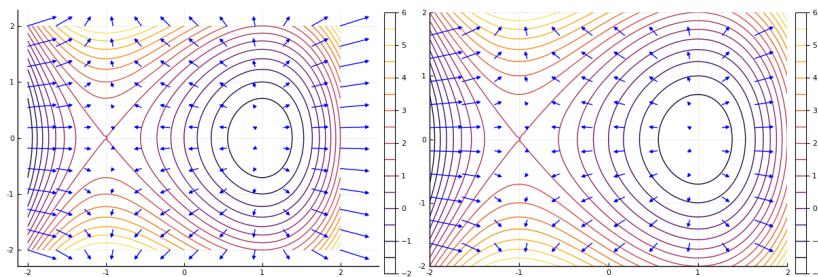


Рис. 5.26. Векторное поле функции
 $h(x, y) = x^3 - 3x + y^2$

Рис. 5.27. Векторное поле функции
 $h(x, y) = x^3 - 3x + y^2$ (скорректирована
область видимости)

5.2.11. Анимация

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

5.2.11.1. Gif-анимация

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

```
pyplot()
```

Строим поверхность (рис. 5.28):

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

Добавляем анимацию (рис. 5.29):

```
# 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
```

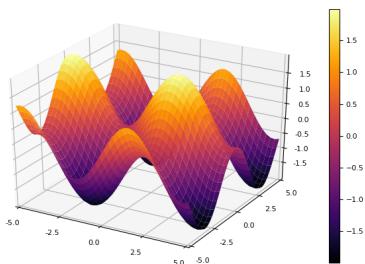


Рис. 5.28. Статичный график поверхности

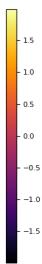


Рис. 5.29. Анимированный график поверхности

5.2.11.2. Гипоциклоида

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

$$\begin{cases} x = r(k-1) \left(\cos t + \frac{\cos((k-1)t)}{k-1} \right), \\ y = r(k-1) \left(\sin t - \frac{\sin((k-1)t)}{k-1} \right), \end{cases}$$

где $k = \frac{R}{r}$, R — радиус неподвижной окружности, r — радиус катящейся окружности.

Модуль величины k определяет форму гипоциклоиды.

Построим гипоциклоиду. Сначала зададим параметры:

```
# радиус малой окружности:  
r = 1  
# коэффициент для построения большой окружности:  
k = 3  
# число отсчётов:  
n = 100
```

Затем зададим массивы необходимых значений:

```
# 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 = r*k*cos.(θ)  
Y = r*k*sin.(θ)
```

Построим оси координат:

```
# задаём оси координат:  
plt=plot(5,xlim=(-4,4),ylim=(-4,4), c=:red, aspect_ratio=1,  
         legend=false, framestyle=:origin)
```

Построим большую окружность (рис. 5.30):

```
# большая окружность:
plot!(plt, X,Y, c=:blue, legend=false)
```

Для частичного построения гипоциклоиды будем менять параметр t (рис. 5.31):

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

Добавляем малую окружность гипоциклоиды (рис. 5.32):

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

Добавим радиус для малой окружности (рис. 5.33):

```
# радиус малой окружности:
xl = transpose([r*(k-1)*cos(t[end]) x[end]])
yl = transpose([r*(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)
```

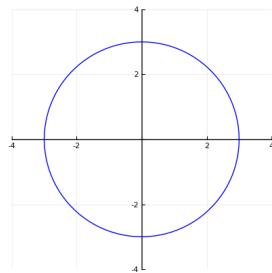


Рис. 5.30. Большая окружность гипоциклоиды

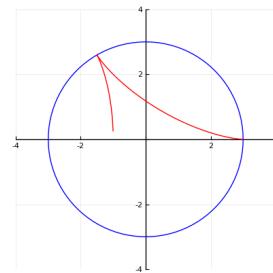


Рис. 5.31. Половина пути гипоциклоиды

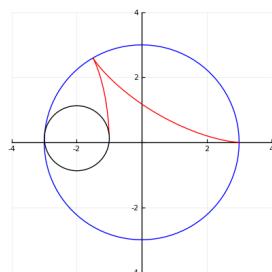
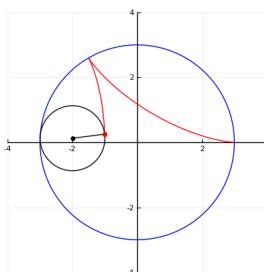


Рис. 5.32. Малая окружность гипоциклоиды Рис. 5.33. Малая окружность гипоциклоиды с добавлением радиуса



В конце сделаем анимацию получившегося изображения (рис. 5.34):

```
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]

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

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

    # радиус малой окружности:
    xl = transpose([r*(k-1)*cos(t[end]) x[end]])
    yl = transpose([r*(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
```

Сохраним анимацию в gif-файл:

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

Рис. 5.34. Анимация движения гипоциклоиды

5.2.12. Errorbars

В исследованиях часто требуется изобразить графики погрешностей измерения.

Подключим пакет Statistics:

```
# подключение пакета Statistics:
```

```
import Pkg
```

```
Pkg.add("Statistics")
```

```
using Statistics
```

Зададим массив значений:

```
sds = [1, 1/2, 1/4, 1/8, 1/16, 1/32]
```

Затем генерируем массив ошибок (отклонений от исходных значений):

```
n = 10
```

```
y = [mean(sd*randn(n)) for sd in sds]
```

```
errs = 1.96 * sds / sqrt(n)
```

Построим график исходных значений (рис. 5.35):

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

Построим график отклонений от исходных значений (рис. 5.36):

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

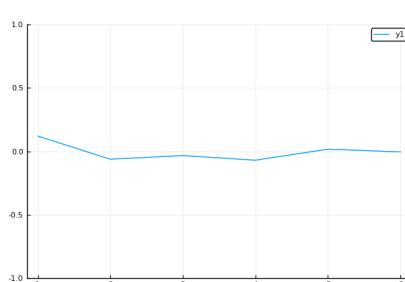


Рис. 5.35. График исходных значений

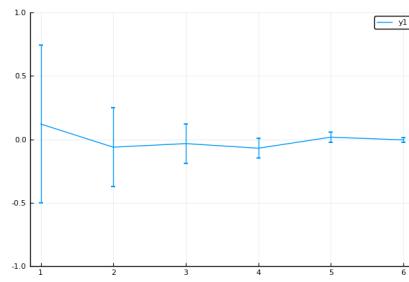


Рис. 5.36. График исходных значений с отклонениями

Повернём график (рис. 5.37):

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

Заполним область цветом (рис. 5.38):

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

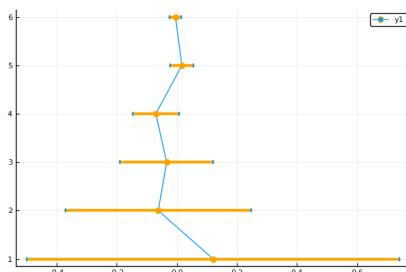


Рис. 5.37. Поворот графика

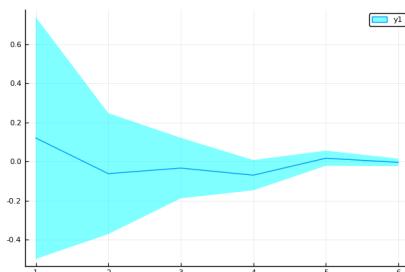


Рис. 5.38. Заполнение цветом

Можно построить график ошибок по двум осям (рис. 5.39):

```
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))
)
```

Можно построить график асимметричных ошибок по двум осям (рис. ??):

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

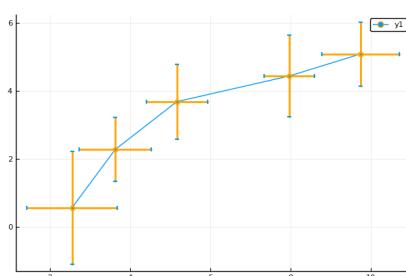


Рис. 5.39. График ошибок по двум осям

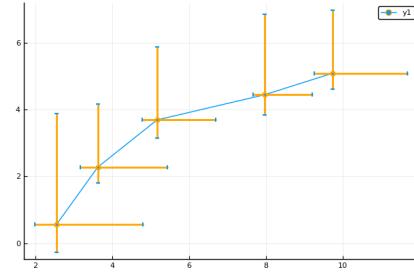


Рис. 5.40. График асимметричных ошибок по двум осям

5.2.13. Использование пакета Distributions

Подгружаем пакет распределений:

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

Подгружаем pyplot():
pyplot()

Задаём массив случайных чисел:
ages = rand(15:55,1000)

Строим гистограмму (рис. 5.41):
histogram(ages)

Задаём нормальное распределение и строим гистограмму (рис. 5.42):

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

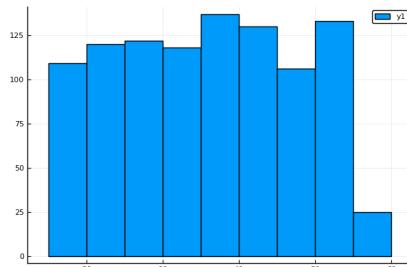


Рис. 5.41. Гистограмма, построенная на массиву случайных чисел

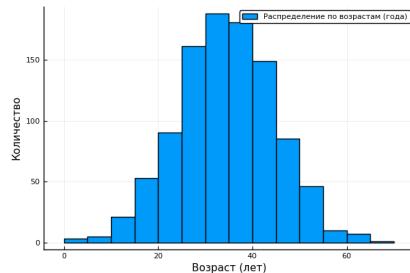


Рис. 5.42. Гистограмма нормального распределения

Далее применим для построения нескольких гистограмм распределения людей по возрастам на одном графике plotly() (рис. 5.43):

```
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)"
)

```

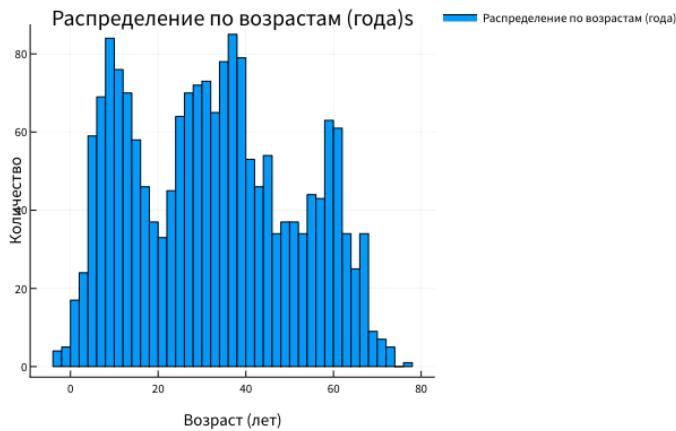


Рис. 5.43. Гистограмма распределения людей по возрастам

5.2.14. 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. 5.44):

```

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

```

Для автоматического вычисления сетки необходимо передать `layout` целое число (рис. 5.45):

```

plot(x,y,
      layout=4
)

```

Чуть более сложные макеты сетки могут быть созданы с помощью конструктора `grid(...)`. Например, в следующем макете создаются участки разной высоты (рис. 5.46).

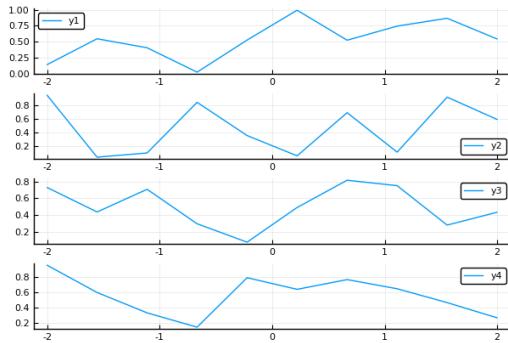


Рис. 5.44. Серия из 4-х графиков в ряд

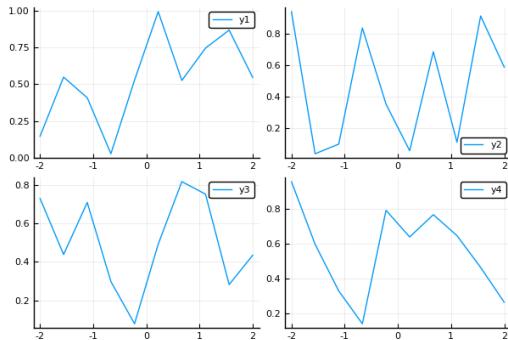


Рис. 5.45. Серия из 4-х графиков в сетке

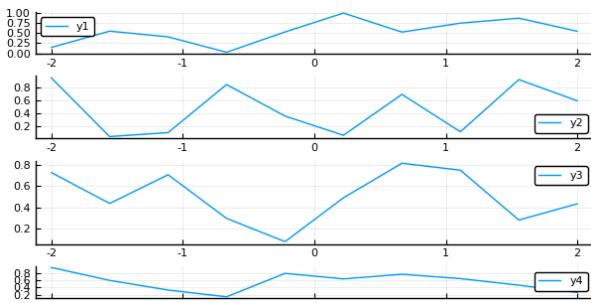


Рис. 5.46. Серия из 4-х графиков разной высоты в ряд

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:

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

Можно сгенерировать отдельные графики и объединить их в один, например, в сетке 2×2 (рис. 5.47):

```
# 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
)
```

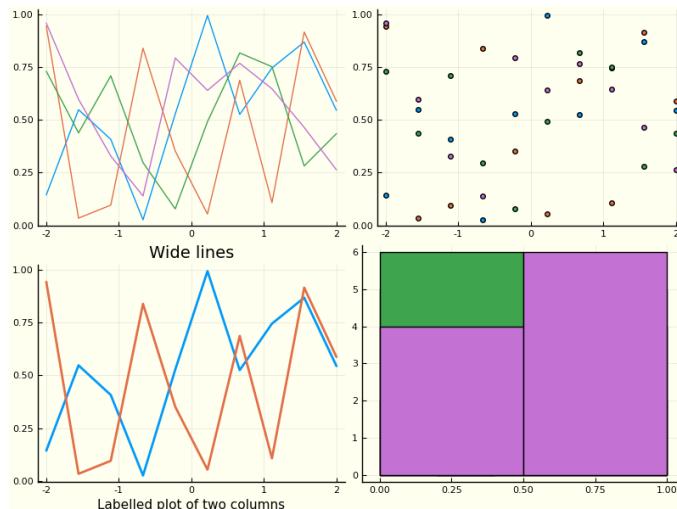


Рис. 5.47. Объединение нескольких графиков в одной сетке

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. 5.48):

```
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
)
```

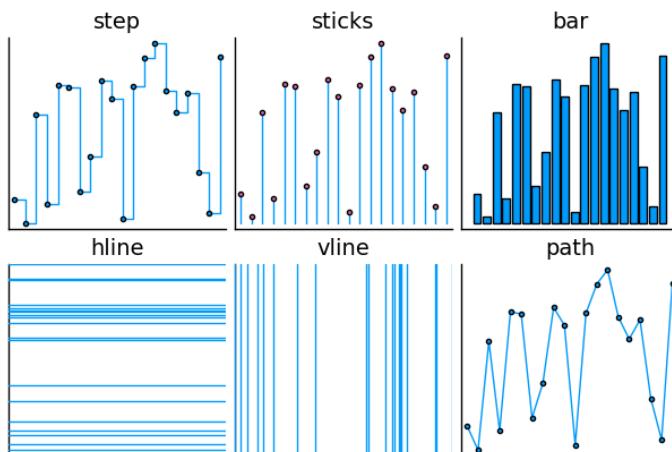


Рис. 5.48. Разнообразные варианты представления данных

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. 5.49):

```
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)
)
```

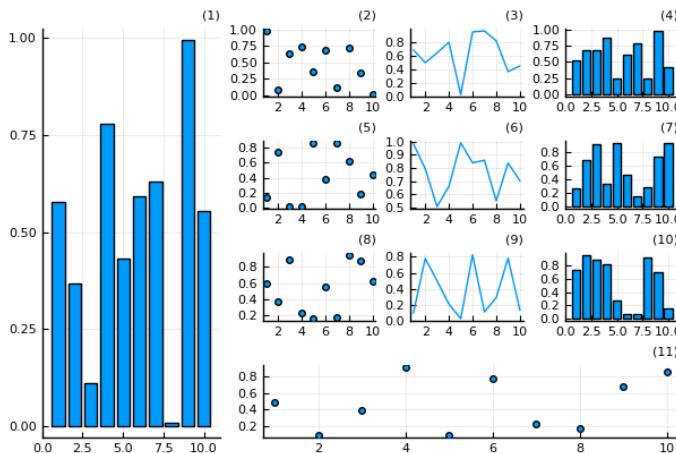


Рис. 5.49. Демонстрация применения сложного макета для построения графиков

5.3. Задание

- Используя Jupyter Lab, повторите примеры из раздела 5.2. При этом дополните графики обозначениями осей координат, легендой с названиями траекторий, названиями графиков и т.п.
- Выполните задания для самостоятельной работы (раздел 5.4).

5.4. Задания для самостоятельного выполнения

- 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.
- 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.
- 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.
- 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.

Сохраните полученные изображения в файле `figure_familiya.png`, где вместо `familiya` укажите вашу фамилию.

5. Задайте вектор $x = (3, 3.1, 3.2, \dots, 6)$. Постройте графики функций $y_1(x) = \pi x$ и $y_2(x) = \exp(x) \cos(x)$ 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.
6. Постройте график некоторых экспериментальных данных (придумайте сами), учитывая ошибку измерения.
7. Постройте точечный график случайных данных. Подпишите оси, легенду, название графика.
8. Постройте 3-мерный точечный график случайных данных. Подпишите оси, легенду, название графика.
9. Создайте анимацию с построением синусоиды. То есть вы строите последовательность графиков синусоиды, постепенно увеличивая значение аргумента. После соедините их в анимацию.
10. Постройте анимированную гипотиклоиду для 2 целых значений модуля k и 2 рациональных значений модуля k .
11. Постройте анимированную эпиклоиду для 2 целых значений модуля k и 2 рациональных значений модуля k .

5.5. Содержание отчёта

1. Титульный лист с указанием номера лабораторной работы и ФИО студента.
2. Формулировка задания работы.
3. Описание выполнения задания:
 - подробное пояснение выполняемых в соответствии с заданием действий;
 - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
 - листинги (исходный код) программ и результаты его выполнения;
4. Выводы, согласованные с заданием работы.