ВСР 2.2
Аналитическая подборка по языку программирования Julia
(направление: научные вычисления и анализ данных)

1) Введение

Julia — современный язык для научных вычислений с JIT-компиляцией, множественной диспетчеризацией и богатой экосистемой. В подборке приведены аннотированные источники и «борды» — врезки кода с пояснениями, покрывающие типовые задачи анализа данных, моделирования и оптимизации.

2) Аннотированная библиография

- Официальная документация Julia — базовый справочник по синтаксису, производительности, параллелизму, менеджеру пакетов.
- Pkg (stdlib): окружения, Project/Manifest, воспроизводимость экспериментов.
- DataFrames.jl: табличные операции (groupby/combine/transform), reshape, join.
- CSV.jl: быстрый парсер CSV/TSV с интеграцией в DataFrames.jl.
- DifferentialEquations.jl: единый интерфейс для ОДУ/СДУ/ДУ/DAE с адаптивными солверами.
- JuMP.jl: декларативная формулировка задач оптимизации (LP/MIP/NLP/SDP) поверх разных солверов.
- Optim.jl: градиентные/безградиентные методы для численной оптимизации.
- Flux.jl: нативный ML-фреймворк с автодифом и поддержкой GPU.
- Makie.jl: современная визуализация 2D/3D/интерактив.
- BenchmarkTools.jl: честные замеры производительности (@btime/@benchmark).

Вышеуказанные источники составляют «минимальный костяк» экосистемы для учебно-исследовательских задач на Julia.

3) Примеры («борды») с комментариями

3.1. Подготовка воспроизводимого окружения

```
import Pkg

Pkg.activate("proj-julia-analytics")

Pkg.add([

   "DataFrames", "CSV", "BenchmarkTools",

   "Makie", "DifferentialEquations", "JuMP", "HiGHS",
```

```
    "Optim", "Flux"
])
```

Фиксируем зависимости в Project.toml/Manifest.toml, чтобы код собирался одинаково на всех машинах.

3.2. Загрузка и агрегирование данных (CSV → DataFrame)

```
using CSV, DataFrames

df = DataFrame(CSV.File("sales.csv"; missingstring="NA"))
first(df, 5)

select!(df,
    :region,
    :product,
    :qty    => ByRow(x -> isnothing(x) ? missing : x) => :qty,
    :price  => :price,
    [:qty, :price] => ByRow(*) => :revenue
)

g = groupby(df, [:region, :product])
agg = combine(g, :revenue => mean => :avg_rev, nrow => :n)
sort!(agg, :avg_rev, rev=true)
```

CSV.jl быстро читает данные; DataFrames.jl предоставляет привычные операции группировки и агрегации.

3.3. Визуализация с Makie (hist + scatter)

```
using Makie

xs = agg.avg_rev
fig = Figure()
```

```
ax1 = Axis(fig[1,1], title = "Распределение среднего дохода")

hist!(ax1, xs, bins=20)


ax2 = Axis(fig[1,2], title = "Средний доход vs количество")

scatter!(ax2, agg.n, agg.avg_rev)

fig
```

Makie даёт управляемые Figure/Axis и качественные графики для публикаций и интерактива.

3.4. Моделирование динамики (ОДУ) в DifferentialEquations.jl

```
using DifferentialEquations


function lotka!(du, u, p, t)

    α, β, δ, γ = p

    x, y = u

    du[1] = α*x - β*x*y

    du[2] = δ*x*y - γ*y

end


u0 = [1.0, 1.0]

p  = (1.5, 1.0, 1.0, 3.0)

tspan = (0.0, 15.0)

prob = ODEProblem(lotka!, u0, tspan, p)

sol = solve(prob, Tsit5(); reltol=1e-8, abstol=1e-8)

sol(5.0)
```

Единый интерфейс постановки задачи (ODEProblem) и адаптивные солверы (например, Tsit5).

3.5. Линейная оптимизация в JuMP + HiGHS

```
using JuMP, HiGHS
```

```
model = Model(HiGHS.Optimizer)

@variable(model, x >= 0)

@variable(model, y >= 0)

@objective(model, Max, 3x + 5y)

@constraint(model, 2x + y <= 14)

@constraint(model, x + 3y <= 18)

optimize!(model)

(value(x), value(y), objective_value(model))
```

Декларативно задаём переменные, ограничения и целевую функцию; решаем HiGHS’ом.

3.6. Численная оптимизация (Optim.jl)

```
using Optim


rosenbrock(x) = (1.0 - x[1])^2 + 100.0*(x[2] - x[1]^2)^2

x0 = [-1.2; 1.0]

res = optimize(rosenbrock, x0, BFGS(); autodiff=:forward)

(res.minimizer, res.minimum, res.iterations)
```

Компактный вызов для «чистых» численных задач без явного задавания модели.

3.7. Бейзлайн-ML с Flux.jl

```
using Flux


X = collect(range(-2, 2; length=201)) |> x -> reshape(x, 1, :)

Y = 2 .* X .- X.^3

model = Chain(Dense(1, 16, relu), Dense(16, 1))

loss(x, y) = Flux.Losses.mse(model(x), y)

opt = Descent(1e-2)
```

```
for epoch in 1:2000
    Flux.train!(loss, Flux.params(model), [(X, Y)], opt)
end
```

Flux — тонкий фреймворк на автодифе; подходит для учебных и исследовательских экспериментов.

3.8. Измерение производительности (BenchmarkTools)

```
using BenchmarkTools

f(n) = sum(i^2 for i in 1:n)

@btime f(10^6)

@benchmark f(10^6)
```

`@btime` и `@benchmark` учитывают прогрев JIT и шум, что даёт честные метрики.

4) Достоинства и ограничения экосистемы

- Скорость и выразительность: один язык от прототипирования до HPC.
- Воспроизводимость: окружения Pkg и фиксация зависимостей.
- Богатый численный стек (DiffEq, JuMP/Optim) и визуализация (Makie).
- Культура измерений (BenchmarkTools).
- Ограничения: молодость некоторых узких пакетов; для продвинутых BI может потребоваться интеграция; ML-экосистема меньше, чем у PyTorch/TF.

5) Рекомендации по применению

1. Активируйте проект и фиксируйте зависимости (Pkg.activate).
2. Начинайте с DataFrames/CSV для чистки и первичного анализа.
3. Для динамики/оптимизации используйте DifferentialEquations и JuMP/Optim.
4. Измеряйте производительность BenchmarkTools и следите за регрессиями.
5. Визуализируйте результаты в Makie, готовя графики к публикации.

6) Заключение

Для научных вычислений и анализа данных Julia обеспечивает цельную цепочку: ввод данных → моделирование/оптимизация → визуализация → измерения. При дисциплине с окружениями и бенчмарками язык сочетает скорость C и удобство высокоуровневого синтаксиса, что делает его эффективным выбором для учебных и исследовательских проектов.