Пример решения задачи оптимизации портфеля через генетические алгоритмы в Julia

Генетические алгоритмы (ГА) — эвристические методы оптимизации, имитирующие естественный отбор. Основные этапы:

1. Генерация начальной популяции
2. Расчет приспособленности особей
3. Селекция лучших родителей
4. Кроссовер и мутация
5. Формирование нового поколения

В портфельной оптимизации:

- Особи: векторы распределения капитала
- Приспособленность: комбинация доходности и риска

Постановка задачи

Инвестор хочет распределить $10,000 между 3 активами:

Актив	Ожидаемая доходность	Риск (σ)	Минимальная доля	Максимальная доля
A	8%	10%	10%	50%
B	12%	15%	10%	50%
C	6%	5%	10%	50%

Ковариационная матрица: - Cov(A,B) = 0.5% - Cov(A,C) = -0.2% - Cov(B,C) = 0.3%

Цель: максимизация E - λVar, где - E = Ожидаемая доходность портфеля - Var = Дисперсия (риск) портфеля - λ = Коэффициент риск-доходность (λ=1.0 означает равный баланс между доходностью и риском)

[ ]:
```
ASSETS = [:A, :B, :C]
EXPECTED_RETURNS = [0.08, 0.12, 0.06]
COV_MATRIX = [
    0.10^2   0.005   -0.002;
    0.005    0.15^2   0.003;
    -0.002   0.003    0.05^2
]
MIN_SHARE = 0.10
MAX_SHARE = 0.50
LAMBDA = 1.0  # Коэффициент риск-доходность

# Параметры алгоритма
```

```
POPULATION_SIZE = 100
GENERATIONS = 50
MUTATION_RATE = 0.1
TOURNAMENT_SIZE = 3
MUTATION_STRENGTH = 0.05
SEED = 42
```

[ ]:
```
42
```

Инициализация окружения

Подключим необходимые пакеты:

[ ]:
```
# import Pkg
# Pkg.add("Random")
```

[ ]:
```
using Random
```

Генерация особи

Особи — нормализованные векторы, удовлетворяющие ограничениям. Алгоритм:

1. Случайная генерация в [0,1]
2. Нормализация суммы к 1
3. Повтор, пока все доли в [10%, 50%]

[ ]:
```
function generate_individual()
    while true
        x = rand(3)
        x /= sum(x)  # Нормализация суммы к 1
        if all(MIN_SHARE .<= x .<= MAX_SHARE)
            return x
        end
    end
end
```

[ ]:
```
generate_individual (generic function with 1 method)
```

Инициализация популяции

Популяция — массив из POPULATION_SIZE особей

[ ]:
```
function generate_population(pop_size)
    [generate_individual() for _ in 1:pop_size]
```

```
end
```

[ ]: generate_population (generic function with 1 method)

Функция приспособленности

Целевая функция: E - λVar, где:

- E = Σ(ожидаемая доходность[i] * доля[i])
- Var = ΣΣ(ковариация[i,j] * доля[i] * доля[j])

[ ]:
```
function fitness(x)
    # Расчет ожидаемой доходности
    E = sum(EXPECTED_RETURNS .* x)

    # Расчет дисперсии портфеля
    Var = COV_MATRIX[1,1]*x[1]^2 +
          COV_MATRIX[2,2]*x[2]^2 +
          COV_MATRIX[3,3]*x[3]^2 +
          2*COV_MATRIX[1,2]*x[1]*x[2] +
          2*COV_MATRIX[1,3]*x[1]*x[3] +
          2*COV_MATRIX[2,3]*x[2]*x[3]

    E - LAMBDA * Var  # Целевая функция для максимизации
end
```

[ ]: fitness (generic function with 1 method)

Селекция родителей

Турнирный отбор: случайный выбор TOURNAMENT_SIZE кандидатов, выбор лучшего

[ ]:
```
function tournament_selection(population, fitnesses)
    candidates = rand(1:POPULATION_SIZE, TOURNAMENT_SIZE)
    best_idx = argmax(fitnesses[candidates])
    population[candidates[best_idx]]
end
```

[ ]: tournament_selection (generic function with 1 method)

Кроссовер

Арифметический кроссовер: линейная комбинация родителей. При нарушении ограничений — генерация новой особи

[ ]:
```
function crossover(p1, p2)
    α = rand()  # Коэффициент смешивания
    child = p1 .+ α.*(p2 .- p1)
    child ./= sum(child)
    all(MIN_SHARE .<= child .<= MAX_SHARE) ? child :␣
  ↪generate_individual()
end
```

[ ]:
```
crossover (generic function with 1 method)
```

Мутация

Гауссово возмущение с адаптацией. Мутированные гены фиксируются в [10%, 50%]

[ ]:
```
function mutate(x)
    mutated = copy(x)
    for i in 1:3
        if rand() < MUTATION_RATE
            # Добавляем шум с нормальным распределением
            mutated[i] = clamp(
                mutated[i] + randn()*MUTATION_STRENGTH,
                MIN_SHARE,
                MAX_SHARE
            )
        end
    end
    mutated ./= sum(mutated)
    all(MIN_SHARE .<= mutated .<= MAX_SHARE) ? mutated : x
end
```

[ ]:
```
mutate (generic function with 1 method)
```

Основной алгоритм

Этапы поколения:

1. Расчет приспособленности
2. Сохранение элиты
3. Селекция, кроссовер и мутация для заполнения популяции

[ ]:
```
function genetic_algorithm()
    Random.seed!(SEED)
    population = generate_population(POPULATION_SIZE)
    best = (fitness=-Inf, individual=zeros(3))
```

```
    for gen in 1:GENERATIONS
        # Расчет приспособленности
        fitnesses = [fitness(ind) for ind in population]
        best_idx = argmax(fitnesses)

        # Сохранение лучшей особи
        if fitnesses[best_idx] > best.fitness
            best = (fitness=fitnesses[best_idx],␣
↪individual=population[best_idx])
        end

        # Формирование нового поколения
        new_pop = [population[best_idx]]  # Элитизм

        while length(new_pop) < POPULATION_SIZE
            parent1 = tournament_selection(population, fitnesses)
            parent2 = tournament_selection(population, fitnesses)
            child = crossover(parent1, parent2)
            push!(new_pop, mutate(child))
        end

        population = new_pop[1:POPULATION_SIZE]
    end
    best
end
```

[ ]:
```
genetic_algorithm (generic function with 1 method)
```

Результаты оптимизации

Запускаем основной алгоритм и выводим результаты

[ ]:
```
result = genetic_algorithm()

# Расчет дополнительных метрик
portfolio_return = sum(EXPECTED_RETURNS .* result.individual)
portfolio_variance = let x = result.individual
    COV_MATRIX[1,1]*x[1]^2 + COV_MATRIX[2,2]*x[2]^2 +␣
↪COV_MATRIX[3,3]*x[3]^2 +
    2*COV_MATRIX[1,2]*x[1]*x[2] + 2*COV_MATRIX[1,3]*x[1]*x[3] +␣
↪2*COV_MATRIX[2,3]*x[2]*x[3]
end

println("Оптимальное распределение:")
println("A: ", round(result.individual[1]*100, digits=2), "%")
```

```
println("B: ", round(result.individual[2]*100, digits=2), "%")
println("C: ", round(result.individual[3]*100, digits=2), "%")

println("\nМетрики портфеля:")
println("Ожидаемая доходность: ", round(portfolio_return, digits=4) *␣
  ↪100, "%")
println("Дисперсия риска: ", round(portfolio_variance, digits=5))
println("Значение целевой функции (E - λVar): ", round(result.
  ↪fitness, digits=4))
```

```
Оптимальное распределение:
A: 39.91%
B: 50.0%
C: 10.09%

Метрики портфеля:
Ожидаемая доходность: 9.8%
Дисперсия риска: 0.00938
Значение целевой функции (E - λVar): 0.0886
```