Пример решения задачи оптимизации портфеля через нелинейное программирование в Julia

Нелинейное программирование (НЛП) — задача оптимизации с нелинейной целевой функцией или ограничениями.

Основная форма:

$$minf(x)$$

при условии

$$g_i(x) \leq 0, h_j(x) = 0$$

$$x \in \mathbb{R}^n$$

В портфельной оптимизации: - Цель: максимизация доходности - Ограничения: риск (нелинейная функция), распределение капитала

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%

Дополним задачу нелинейным ограничением на максимально допустимый риск.

[ ]:
```
EXPECTED_RETURNS = [0.08, 0.12, 0.06]  # A, B, C
RISKS = [0.10, 0.15, 0.05]             # Стандартные отклонения
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
MAX_VARIANCE = 0.008  # Максимально допустимая дисперсия
```

[ ]:
```
0.008
```

Инициализация окружения

Подключим необходимые пакеты:

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

[ ]:
```
using JuMP, Ipopt
```

Инициализация модели

Создаем модель с нелинейным решателем Ipopt. Отключаем подробный вывод.

[ ]:
```
model = Model(Ipopt.Optimizer)
set_optimizer_attribute(model, "print_level", 0)  # Уменьшаем вывод␣
  ↪логов
```

Переменные решения

Доли активов с ограничениями через константы

[ ]:
```
@variable(model, MIN_SHARE <= x[1:3] <= MAX_SHARE, base_name =␣
  ↪["x_A", "x_B", "x_C"]);
```

Базовые ограничения

Полное инвестирование капитала

[ ]:
```
@constraint(model, sum(x[i] for i in 1:3) == 1.0);
```

Нелинейное ограничение на риск

Дисперсия портфеля вычисляется через квадратичную форму

[ ]:
```
@NLconstraint(model,
    x[1]^2 * COV_MATRIX[1,1] +
    x[2]^2 * COV_MATRIX[2,2] +
    x[3]^2 * COV_MATRIX[3,3] +
    2x[1]x[2]*COV_MATRIX[1,2] +
    2x[1]x[3]*COV_MATRIX[1,3] +
    2x[2]x[3]*COV_MATRIX[2,3] <= MAX_VARIANCE
);
```

Целевая функция

Максимизация ожидаемой доходности портфеля

[ ]:
```
@objective(model, Max, sum(x[i] * EXPECTED_RETURNS[i] for i in 1:3));
```

Решение задачи

Запуск оптимизации и проверка корректности решения

[ ]:
```
optimize!(model)

if termination_status(model) != MOI.LOCALLY_SOLVED
    error("Решение не найдено: ", termination_status(model))
end
```

Результаты оптимизации

Вывод оптимальных долей и ключевых метрик портфеля

[ ]:
```
println("Оптимальные доли:")
println("A: ", round(value(x[1])*100, digits=2), "%")
println("B: ", round(value(x[2])*100, digits=2), "%")
println("C: ", round(value(x[3])*100, digits=2), "%")

# Расчет фактической дисперсии
portfolio_variance = value(x[1])^2 * COV_MATRIX[1,1] +
                     value(x[2])^2 * COV_MATRIX[2,2] +
                     value(x[3])^2 * COV_MATRIX[3,3] +
                     2value(x[1])*value(x[2])*COV_MATRIX[1,2] +
                     2value(x[1])*value(x[3])*COV_MATRIX[1,3] +
                     2value(x[2])*value(x[3])*COV_MATRIX[2,3]

portfolio_return = sum(value(x[i]) * EXPECTED_RETURNS[i] for i in 1:3)
portfolio_risk = sqrt(portfolio_variance)

println("\nМетрики портфеля:")
println("Ожидаемая доходность: ", round(portfolio_return*100,␣
  ↪digits=2), "%")
println("Дисперсия: ", round(portfolio_variance, digits=5))
println("Стандартное отклонение: ", round(portfolio_risk*100,␣
  ↪digits=2), "%")
```

```
Оптимальные доли:
A: 32.01%
B: 47.29%
C: 20.7%

Метрики портфеля:
Ожидаемая доходность: 9.48%
```

Дисперсия: 0.008
Стандартное отклонение: 8.94%