Пример решения задачи оптимизации портфеля через квадратичное программирование в Julia

Квадратичное программирование (КП) — задача оптимизации с квадратичной целевой функцией и линейными ограничениями. Основная форма:

$\min (1/2)x^TQx + c^Tx$

при условии $Ax \leq b$
$x \in \mathbb{R}^n$

В портфельной оптимизации: - Цель: минимизация риска (дисперсии) - Ограничения: доходность, 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%

Цель: минимизация риска (дисперсии)

[ ]:
```
ASSETS = [:A, :B, :C]  # Идентификаторы активов
N = length(ASSETS)

# Ожидаемые доходности (в порядке A, B, C)
EXPECTED_RETURNS = [0.08, 0.12, 0.06]

# Матрица ковариаций (в порядке A, B, C)
COV_MATRIX = [
    0.10^2   0.005   -0.002;
    0.005    0.15^2   0.003;
    -0.002   0.003    0.05^2
]

# Ограничения на доли
MIN_SHARE = 0.10
MAX_SHARE = 0.50
```

[ ]:
```
0.5
```

Инициализация окружения

Подключим необходимые пакеты:

[ ]:
```
# Установка пакетов (раскомментировать при первом запуске)
# import Pkg
# Pkg.add("JuMP")
# Pkg.add("Ipopt")
```

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

Инициализация модели

Создаем модель оптимизации с использованием решателя Ipopt, специализированного для нелинейных задач

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

Переменные решения

Вводим переменные для долей капитала с ограничениями снизу и сверху

[ ]:
```
@variable(model, MIN_SHARE <= x_A <= MAX_SHARE)
@variable(model, MIN_SHARE <= x_B <= MAX_SHARE)
@variable(model, MIN_SHARE <= x_C <= MAX_SHARE)
```

[ ]:
x_C

Базовые ограничения

Главное ограничение — полное инвестирование капитала

[ ]:
```
@constraint(model, total_investment, x_A + x_B + x_C == 1.0)
```

[ ]:
x_A + x_B + x_C = 1

Целевая функция

Рассчитываем дисперсию портфеля по формуле:

$$Var = \Sigma(w_i^2 \sigma_i^2) + 2\Sigma(w_i w_j Cov_i j)$$

[ ]:
```
@objective(model, Min,
    x_A^2 * COV_MATRIX[1,1] +
    x_B^2 * COV_MATRIX[2,2] +
    x_C^2 * COV_MATRIX[3,3] +
    2x_A*x_B*COV_MATRIX[1,2] +
    2x_A*x_C*COV_MATRIX[1,3] +
    2x_B*x_C*COV_MATRIX[2,3]
)
```

[ ]:
$0.010000000000000002x\_A^2 + 0.0225x\_B^2 + 0.0025000000000000005x\_C^2 + 0.01x\_A \times x\_B - 0.004x\_A \times x\_C + 0.006x\_B \times x\_C$

Решение задачи

Запускаем оптимизацию и проверяем статус решения

[ ]:
```
optimize!(model)

if termination_status(model) == MOI.LOCALLY_SOLVED
    println("Решение найдено успешно")
else
    println("Проблемы при решении: ", termination_status(model))
end
```

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear␣
 ↪optimization.
 Ipopt is released as open source code under the Eclipse Public␣
 ↪License (EPL).
         For more information visit https://github.com/coin-or/Ipopt
******************************************************************************

Решение найдено успешно
```

Результаты оптимизации

Выводим оптимальные доли и ключевые метрики портфеля

[ ]:
```
println("\nОптимальные доли:")
println("A: ", round(value(x_A)*100, digits=2), "%")
println("B: ", round(value(x_B)*100, digits=2), "%")
println("C: ", round(value(x_C)*100, digits=2), "%")

portfolio_return = sum([
    value(x_A)*EXPECTED_RETURNS[1],
```

```
    value(x_B)*EXPECTED_RETURNS[2],
    value(x_C)*EXPECTED_RETURNS[3]
])

portfolio_variance = objective_value(model)
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: 40.0%
B: 10.0%
C: 50.0%

Метрики портфеля:
Ожидаемая доходность: 7.4%
Дисперсия: 0.00235
Стандартное отклонение: 4.85%
```