Пример решения задачи оптимизации портфеля через стохастическое программирование в Julia

Стохастическое программирование — подход к оптимизации с учетом неопределенности в данных. Основная идея:

- Параметры модели представляются случайными величинами
- Используется генерация сценариев для аппроксимации распределений
- Целевая функция включает риск-метрики (CVaR, VaR)

В портфельной оптимизации:

- Цель: баланс между доходностью и риском
- Особенность: учет экстремальных сценариев через CVaR

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%

Особенности задачи:

- Генерация 100 сценариев доходности методом Монте-Карло
- Комбинированная цель: максимизация ожидаемой доходности - 5%*CVaR

[ ]:
```
ASSETS = [:A, :B, :C]
EXPECTED_RETURNS = [0.08, 0.12, 0.06]
RISKS = [0.10, 0.15, 0.05]
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
N_SCENARIOS = 100
CVAR_ALPHA = 0.95  # Уровень доверия для VaR
CVAR_BETA = 1 - CVAR_ALPHA   # Вес CVaR в целевой функции
```

[ ]:
```
0.050000000000000044
```

Инициализация окружения

Подключим необходимые пакеты:

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

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

Генерация сценариев

Используем многомерное нормальное распределение для моделирования доходностей

[ ]:
```
Random.seed!(42)  # Фиксируем seed для воспроизводимости

# Создание распределения и генерация данных
mvnormal = MvNormal(EXPECTED_RETURNS, COV_MATRIX)
returns_scenarios = rand(mvnormal, N_SCENARIOS)';
```

Инициализация модели

Создаем модель с нелинейным решателем Ipopt

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

Переменные решения

Целочисленные переменные для каждого актива с явными границами

[ ]:
```
@variable(model, MIN_SHARE <= x[1:3] <= MAX_SHARE)
```

[ ]:
```
3-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
```

Базовые ограничения

Полное инвестирование капитала

[ ]:
```
@constraint(model, sum(x) == 1);
```

Параметры CVaR

Вводим вспомогательные переменные для расчета Conditional Value-at-Risk

[ ]:
```
@variable(model, VaR)
@variable(model, Z[1:N_SCENARIOS] >= 0);
```

Ограничения для CVaR

Формализуем условия для расчета ожидаемых потерь

[ ]:
```
@constraint(model, [i=1:N_SCENARIOS],
    Z[i] >= VaR - sum(x[j] * returns_scenarios[i, j] for j in 1:3)
);
```

Целевая функция

Комбинируем ожидаемую доходность и риск через CVaR

[ ]:
```
@objective(model, Max,
    sum(mean(returns_scenarios[:, j]) * x[j] for j in 1:3) -
    CVAR_BETA * (VaR - (1/(1-CVAR_ALPHA)) * sum(Z))/N_SCENARIOS
);
```

Решение задачи

Запуск оптимизации и проверка корректности решения

[ ]:
```
optimize!(model)
```

Результаты оптимизации

Выводим оптимальные доли и ключевые метрики

[ ]:
```
println("Оптимальные доли:")
for (i, asset) in enumerate(ASSETS)
    println("$asset: ", round(value(x[i])*100, digits=2), "%")
```

```
end

portfolio_return = sum(value(x[i]) * EXPECTED_RETURNS[i] for i in 1:3)
portfolio_cvar = value(VaR) - (sum(value.(Z))/N_SCENARIOS)/
  ↪(1-CVAR_ALPHA)

println("\nКлючевые метрики:")
println("Ожидаемая доходность: ", round(portfolio_return*100,␣
  ↪digits=2), "%")
```

```
Оптимальные доли:
A: 38.82%
B: 49.84%
C: 11.34%

Ключевые метрики:
Ожидаемая доходность: 9.77%
```