СБЕР

Математика на чистой Java,

или о внедрении математических моделей в бизнес-логику enterprise-проекта

Александр Эйдлин
Telegram @AlexanderEydlin

О чём этот доклад?

1

Откуда в «рядовом» проекте может ВНЕЗАПНО появиться математика

2

Какие в принципе есть варианты внедрения математических моделей в бизнес-логику Java-проекта

3

Какие библиотеки с GitHub в этом могут помочь

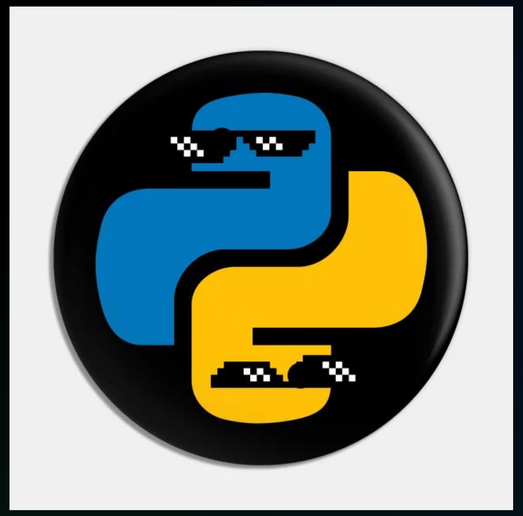
4

Как на чистой Java удалось реализовать большую систему с множеством математических моделей

Математика на Java. Но зачем?!

Автор иллюстрации – Борис Могилин

Неужели нельзя сделать на Python / R?



- ✓ Более привычные языки для математики
- ✗ Нужны компетенции – полбеды
- ✗ Нужна интеграция – со всеми вытекающими последствиями (инфраструктура, DevOps, сопровождение)

Ради единственной математической модели?
Серьёзно?

JBreak 2017. Алексей Зиновьев
Kafka льёт, а Spark разгребает

Joker 2023. Дмитрий Морозовский
Путешествие из Java в Python:
два мира — один JEP

* QR-коды кликабельны

Некоторые реальные кейсы

Нужен результат в разумные сроки с помощью знакомого и привычного инструмента

- Студенческая курсовая или дипломная работа
- Быстрая реализация MVP

Расчёты составляют небольшую часть функционала продукта

- Пример: прогноз расходов по бюджетным статьям в финансовой системе

Математические модели глубоко интегрированы в бизнес-логику продукта

- Скоринговая система
- Система оптимизации кассовой ликвидности Сбера

Типовые задачи, которые нужно решать

Статистический анализ и обработка данных

- Простейшие стат. показатели: медиана, мода, среднеквадратичное отклонение
- Анализ главных компонентов (PCA), спектральный анализ

Матрицы и операции над ними

Математическая оптимизация

- Классические оптимизационных задач (градиентные методы, одномерная оптимизация, симплекс-метод, метод Нелдера-Мида, дискретная оптимизация и т.д.)
- Стохастическая оптимизация (методы случайного поиска)

Численные методы

Машинное обучение

- Классификация и регрессия
- Кластерный анализ
- Нейронные сети

~~Кто виноват?~~
Что делать?

Варианты решения

- **Spark ML – мощный инструмент, заточенный под big data и machine learning**

JBreak 2018

Алексей Зиновьев

Смузи ML вместе со Spark MLlib

Joker 2023. Николай Бутаков AutoML на Spark: миф, ставший реальностью

Варианты решения

Свой «велосипед» (или порт с другого языка)

✓ Минимум зависимостей, простота нахождения багов и оптимизации производительности

✗ Сложность реализации может оказаться неприемлемой

Встроенные возможности реляционной СУБД

- От простейших агрегатных функций SQL до хранимых процедур
- Вариант интеграции, использующий JDBC в качестве API

Поискать математические библиотеки на GitHub

Немного предметной области

Процесс управления кассовой ликвидностью

Заявка на обслуживание точки (банкомат или доп. офис) – центральный артефакт кассовой ликвидности.

Аналитик формирует заявку и отправляет её в **кассовый центр**.

В кассовом центре **кассир** готовит сумки / кассеты и сопроводительную документацию.

Инкассаторы исполняют заявку, выезжая к точке и совершая операции сдачи и/или подкрепления.

Идея: автоматизировать работу аналитиков, рекомендуя оптимальные суммы и даты обслуживания точек.

12 тыс.

дополнительных офисов, включая передвижные

49 тыс.

устройств самообслуживания 7 различных видов

Оптимизация кассовой ликвидности

Входящие остатки, прогнозы

Для мультивалютных объектов (доп. офисов) – по всем валютам

Затраты

Стоимость кассовых операций

Стоимость выезда инкассаторов

Ставка фондирования

Курсы валют

Интервалы обслуживания

Когда возможно обслуживание точки (в зависимости от режима доступа, графика работы подразделений кассы и инкассации)

Ресурс кассового центра

Количество заявок, которые кассовый центр может исполнить (по дням)

ervin-x. Прикручиваем ИИ.

Оптимизация работы банкоматов

Основные этапы расчёта оптимальных заявок

Joker 2023. Дмитрий Бугайченко
Демистифицируем машинное обучение –
из разработчика в ML-инженеры

Задача регрессии

По значениям переменных (предикторов) и известным выходным значениям восстановить их связь и научиться предсказывать выходные значения на новых данных.

Площадь общая	Площадь кухни	Число комнат	Этаж	Вид из окна	…	Стоимость
44,7	12,2	1	4	двор		11 345 000
71,5	11,3	1	7	шоссе		15 221 000
38,1	6,5	2	2	парк		10 719 000
46,3	15,3	1	8	улица		?

желаемые выходы

прогнозируемое значение

входные признаки (предикторы)

$y = f(x) = f(x_1, \dots, x_n)$ – неизвестная зависимость,

попытаемся смоделировать: $y \sim \hat{f}(x_1, \dots x_n)$

на основании известных значений $y^{(1)}, \dots, y^{(m)}$, соответствующих векторам $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

Прогноз оборотов как задача регрессии

Хотим спрогнозировать выдачу клиентам из банкомата на горизонт до 45 дней (к примеру).

День t	$T-p$	$T-p+1$	…	$T-2$	$T-1$	T	$T+1$	…	$T+\tau$
Сумма $y(t)$	313 100	442 300		657 000	92 900	$y(T)$	$y(T+1)$		$y(T+\tau)$

- **Минимальный вариант – авторегрессия:**

  $y(t) \sim \hat{f}\,(y(t-1), y(t-2), \dots)$: предикторы – предыдущие значения

- **Более качественный прогноз требует учёта дополнительных факторов:**

$$y(t) \sim \hat{f}\,(x_1, \dots, x_m),$$

где x_i – предыдущие значения, флаги выходного дня, усреднённые обороты за период, факторы повышенного спроса и т.д.

Математическая оптимизация

Подобрать такой вектор X, которое обеспечит наименьшее или наибольшее значение целевой функции $F(X)$ и при этом будет удовлетворять ограничениям задачи.

Пример: нефтеперерабатывающий завод может произвести до 500 тыс. тонн в месяц бензина двух марок:

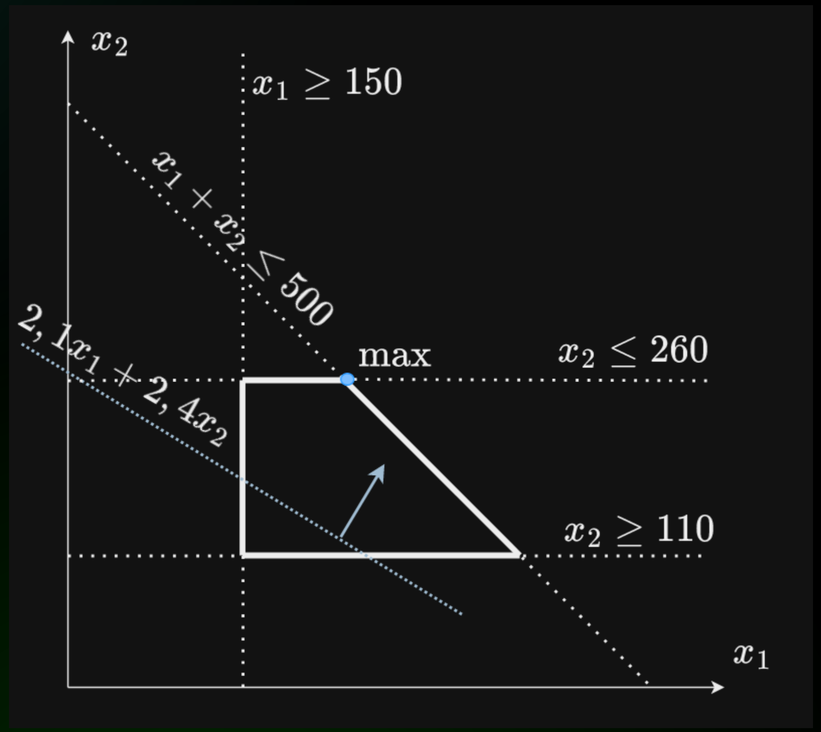
	Обязательства перед клиентами, тыс. тонн	Максимум, тыс. тонн	Маржа, млн. ₽ / тыс. т	Объём производства, тыс. тонн
АИ-95	150	неограниченно	2,1	x_1
АИ-98	110	260	2,4	x_2

Сколько бензина x_1 и x_2 каждого вида произвести, чтобы обеспечить максимальную прибыль?

Математическая оптимизация

Постановка в виде задачи линейного программирования:

$$\begin{cases} \underbrace{F(x_1, x_2) = 2{,}1x_1 + 2{,}4x_2 \to \max}_{\text{целевая функция (кост-функция, фитнес-функция и т.д.)}} \\ \left.\begin{array}{c} x_1 + x_2 \leq 500 \\ 150 \leq x_1 \\ 110 \leq x_2 \leq 260 \\ x_1 \geq 0, x_2 \geq 0 \end{array}\right\} \text{система ограничений} \end{cases}$$



- Целевая функция и ограничения могут быть нелинейными.
- Аналитический вид может быть неизвестен.
- Переменные могут быть непрерывными и дискретными, их число может достигать тысяч и больше.

Некоторые подходы к решению задач оптимизации

Переменные	Целевая функция	Ограничения	Метод решения
Непрерывные	Непрерывно дифференцируемая	Равенства, причём $F_i(X)$ – непрерывно дифференцируемые	Множители Лагранжа
Единственная непрерывная	Рассчитывается «дорого» или экспериментально	$\mathrm{a} \leq x \leq \mathrm{b}$	**Методы одномерного поиска**
Непрерывные	Дифференцируемая	Отсутствуют	Градиентный спуск и модификации
Непрерывные	Любая, в т.ч. рассчитываемая экспериментально	$\mathrm{m_j} \leq x_j \leq \mathrm{M_j}, \mathrm{j} = \overline{1, N}$	Нелдера-Мида, **BOBYQA**
Непрерывные	Линейная	Линейные	Симплекс-метод
Непрерывные	Выпуклая	Выпуклые	Методы выпуклого программирования
Целочисленные и/или непрерывные, булевы*	Линейная	Линейные	Отсечения (Гомори), **ветвей и границ**
Любые	Произвольная	Произвольные	**Стохастические (рандомизированные)**

А теперь — к делу

Какие библиотеки нам помогли?

Apache commons-math

SMILE

XGBoost4J

ojAlgo

Основные этапы расчёта оптимальных заявок

Стартер-пак математика – Apache commons-math

Сайт

GitHub

Стартер-пак математика – Apache commons-math

Линейная алгебра и матрицы

Операции над матрицами и векторами,
собственные значения и вектора,
решение СЛАУ,
разложение матриц...

Математическая статистика

Стандартные статистические функции,
линейная регрессия,
проверка статистических гипотез...

Математическая оптимизация

Симплекс-метод,
одномерный поиск,
методы нулевого порядка (Нелдера-Мида, BOBYQAO, CMA-ES...),
градиентные методы

Численные методы

Интегрирование,
дифференцирование,
дифференциальные уравнения...

Стартер-пак математика – Apache commons-math

Генетические алгоритмы

Машинное обучение

Кластеризация K-means и модификации
Кластеризация DBSCAN

Нейронные сети

Одномерные и двумерные самоорганизующиеся карты Кохонена (SOM)

Разное

Фильтры Калмана
Функции Бесселя

Что не так с commons-math?

☹ **Проект развивается крайне медленно**

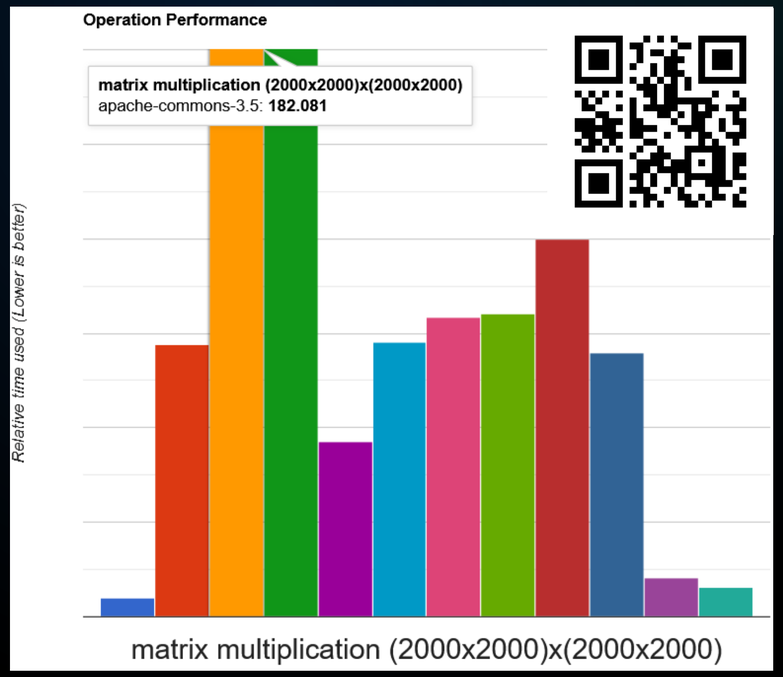
- **Последняя стабильная версия 3.6.1 вышла восемь (!) лет назад.**
- **Work in progress – 4.0-beta1 вышла в декабре 2022.**
- **Документация на сайте поломалась и её не чинят.**
- **Низкая активность на GitHub.**

Что не так с commons-math?

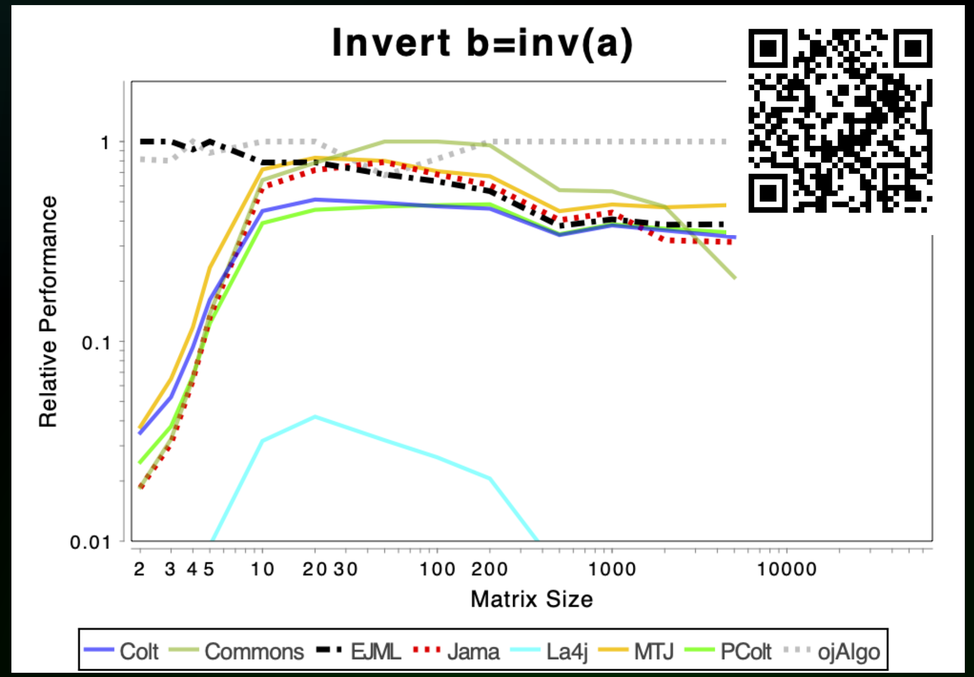
☹ **Не хватает функционала, особенно в части ML**

и непохоже, что ситуация изменится.

☹ **Проблемы с производительностью матричных операций**



Время перемножения матриц (меньше – лучше)



Скорость инвертирования матрицы от размера (больше – лучше)

Smile – «Scikit-learn на JVM»

Сайт

GitHub

Smile – «Scikit-learn на JVM»

Классификация

Support vector machines,
multilayer perceptron,
Naïve Bayes,
decision tree…

Регрессия

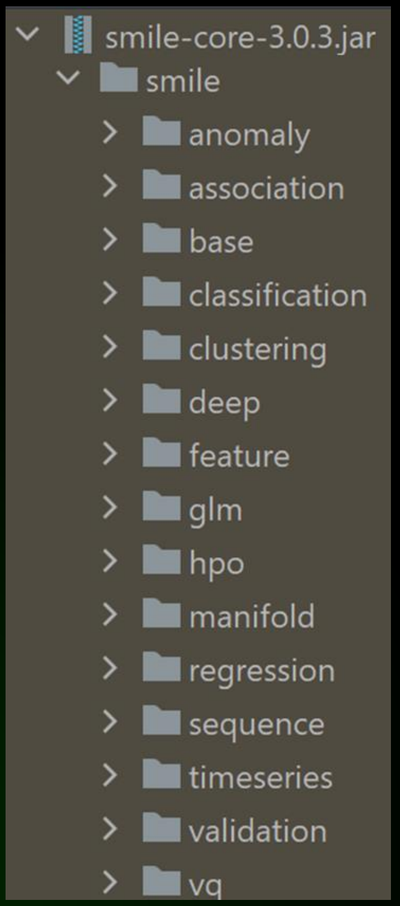
LASSO, random forest,
ridge regression, ElasticNet,
gradient tree boosting,
RBF network…

Кластеризация

Иерархические методы
(single link, complete link…),
DBSCAN, CLARANS,
K-means и вариации,
deterministic annealing…

Прочее

Векторная квантизация (SOM, neural gas),
детекция аномалий,
глубокое обучение,
извлечение и анализ признаков,
матрицы, генетические алгоритмы…



Ансамбль ~~песни и пляски~~ прогнозных моделей

Ансамбль ~~песни и пляски~~ прогнозных моделей

XGBoost – секретное оружие дата сатанистов

dmlc XGBoost

Сайт

GitHub

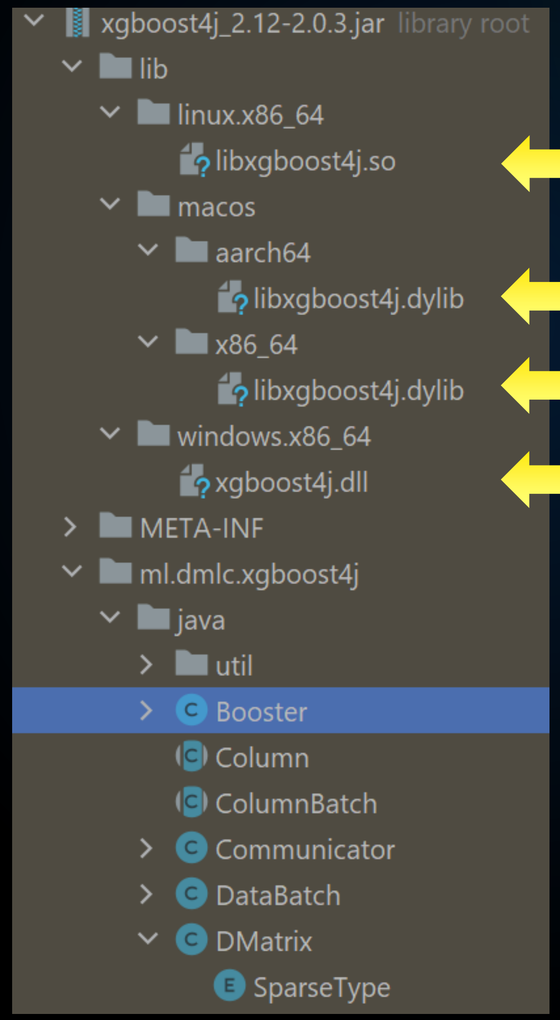
Java API Tutorial

Описание алгоритма

XGBoost – секретное оружие дата сатанистов

- Реализует одноимённый эффективный метод eXtreme Gradient Boosting, предназначенный для решения задач кластеризации и регрессии.
- Поддерживает распределённые вычисления и расчёты на GPU (CUDA).
- Умеет работать с Apache Spark, Hadoop, Flink.

Заглянем в исходники XGBoost4J




```
private void init(DMatrix[] cacheMats) throws XGBoostError {
    long[] handles = null;
    if (cacheMats != null) {
        handles = dmatrixsToHandles(cacheMats);
    }
    long[] out = new long[1];
    XGBoostJNI.checkCall(XGBoostJNI.XGBoosterCreate(handles, out));

    handle = out[0];
}

@Override
protected void finalize() throws Throwable {
    super.finalize();
    dispose();
}

public synchronized void dispose() {
    if (handle != 0L) {
        XGBoostJNI.XGBoosterFree(handle);
        handle = 0;
    }
}
```

XGBoost4J – нативная библиотека. Что из этого следует?

Выделил память – приберись за собой, вызови dispose()!

Booster, DMatrix, QuantileDMatrix

Данные, помещаемые в DMatrix, нужно преобразовать в float[]:

```
List<SortedMap<String /* имя колонки */, Double>> xTrain;
List<Double> yTrain;
...
// вытянем предикторы (входы) в плоский массив
DMatrix train = new DMatrix(xToFloatArray(xTrain), xTrain.size(), featureCount, 0.0f);
// зададим разметку (желаемые выходы)
train.setLabel(yToFloatArray(yTrain));

Booster booster = XGBoost.train(train, ...);
```

JPoint 2020. Иван Углянский
В нативный код из уютного мира Java:
Путешествие туда и обратно

Основные этапы расчёта оптимальных заявок

Постановка задачи оптимизации обслуживания

Дано:

D – длина горизонта в днях,

M – количество оптимизируемых объектов кассового центра (банкоматов / клиентских офисов),

C_i – стоимость единичного обслуживания i-го объекта,

доступные временные интервалы обслуживания объектов по дням,

r – ставка фондирования, % годовых (стоимость отвлечения денежных средств),

Требуется:

Составить такой план обслуживания, который обеспечит:

1) **наименьшие расходы** на обслуживание и фондирование;

2) **выполнение SLA по доступности** (деньги не должны кончаться, банкоматы не должны переполняться);

3) **соблюдение прочих ограничений** (ресурс кассового центра, режим доступа инкассаторов, физическая вместимость кассет, рисковые показатели и т.д.).

Постановка задачи* оптимизации обслуживания

Введём переменные:

$$x_{ij} = \begin{cases} 1, \text{если } i - \text{й объект обслуживается в } j - \text{й день горизонта,} \\ 0 \text{ иначе.} \end{cases}$$

Получим матрицу инкассаций, которую и будем оптимизировать:

$$X = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

M объектов

День 1 День 2 ... День D

* значительно упрощённая

Постановка задачи оптимизации обслуживания

Выразим целевую функцию:

Суммарная стоимость горизонта по объектам:

$$Cost(X) = \sum_{i=1}^{M} Cost_i(X) \to \min,$$

Стоимость объекта – стоимость обслуживания + расходы на фондирование:

$$Cost_i(X) = Cost_i^{serv}(X) + Cost_i^{fund}(X)$$

Избавимся ограничений с помощью метода штрафных функций:

За нарушение k-го ограничения к общей стоимости будем прибавлять штраф:

$$Cost_i(X) \mathrel{+}= Cost_i(X) + Cost_i^{Penalty_j}(X)$$

Для поиска оптимального решения применим метаэвристику «имитация отжига».

Метод имитации отжига. Свой «велосипед»

```
// константы подбираются опытным путём
private static final int MAX_ITERATIONS = 1000;
private static final double INITIAL_TEMPERATURE = 1000;
private static final double DECREASE_FACTOR = 1 - 10. / MAX_ITERATIONS;

public Solution minimize() {
    double temperature = INITIAL_TEMPERATURE;
    Solution currentSolution = buildInitialSolution();
    double currentCost = calculateCost(currentSolution);

    for (int i = 0; i < MAX_ITERATIONS; i++) {
        temperature = temperature * DECREASE_FACTOR;

        Solution newSolution = mutate(currentSolution);
        double newCost = calculateCost(newSolution);

        if (newCost < currentCost
            || (new Random()).nextDouble() < Math.exp(-(newCost - currentCost) / temperature)) {

            currentSolution = newSolution;
            currentCost = newCost;
        }
    }
    return currentSolution;
}
```

Метод имитации отжига. Свой «велосипед»

```
// константы подбираются опытным путём
private static final int MAX_ITERATIONS = 1000;
private static final double INITIAL_TEMPERATURE = 1000;
private static final double DECREASE_FACTOR = 1 - 10. / MAX_ITERATIONS;

public Solution minimize() {
    double temperature = INITIAL_TEMPERATURE;
    Solution currentSolution = buildInitialSolution();
    double currentCost = calculateCost(currentSolution);

    for (int i = 0; i < MAX_ITERATIONS; i++) {
        temperature = temperature * DECREASE_FACTOR;

        Solution newSolution = mutate(currentSolution);
        double newCost = calculateCost(newSolution);

        if (newCost < currentCost
            || (new Random()).nextDouble() < Math.exp(-(newCost - currentCost) / temperature)) {

            currentSolution = newSolution;
            currentCost = newCost;
        }
    }
    return currentSolution;
}
```

Построим начальное решение, рассчитаем стоимость

Метод имитации отжига. Свой «велосипед»

```
// константы подбираются опытным путём
private static final int MAX_ITERATIONS = 1000;
private static final double INITIAL_TEMPERATURE = 1000;
private static final double DECREASE_FACTOR = 1 - 10. / MAX_ITERATIONS;

public Solution minimize() {
    double temperature = INITIAL_TEMPERATURE;
    Solution currentSolution = buildInitialSolution();
    double currentCost = calculateCost(currentSolution);

    for (int i = 0; i < MAX_ITERATIONS; i++) {
        temperature = temperature * DECREASE_FACTOR;

        Solution newSolution = mutate(currentSolution);
        double newCost = calculateCost(newSolution);

        if (newCost < currentCost
            || (new Random()).nextDouble() < Math.exp(-(newCost - currentCost) / temperature)) {

            currentSolution = newSolution;
            currentCost = newCost;
        }
    }
    return currentSolution;
}
```

Рандомно модифицируем решение, рассчитываем новую стоимость

Метод имитации отжига. Свой «велосипед»

```
// константы подбираются опытным путём
private static final int MAX_ITERATIONS = 1000;
private static final double INITIAL_TEMPERATURE = 1000;
private static final double DECREASE_FACTOR = 1 - 10. / MAX_ITERATIONS;

public Solution minimize() {
    double temperature = INITIAL_TEMPERATURE;
    Solution currentSolution = buildInitialSolution();
    double currentCost = calculateCost(currentSolution);

    for (int i = 0; i < MAX_ITERATIONS; i++) {
        temperature = temperature * DECREASE_FACTOR;

        Solution newSolution = mutate(currentSolution);
        double newCost = calculateCost(newSolution);

        if (newCost < currentCost
            || (new Random()).nextDouble() < Math.exp(-(newCost - currentCost) / temperature)) {

            currentSolution = newSolution;
            currentCost = newCost;
        }
    }
    return currentSolution;
}
```

Решение с меньшей стоимостью акцептуем сразу

Метод имитации отжига. Свой «велосипед»

```
// константы подбираются опытным путём
private static final int MAX_ITERATIONS = 1000;
private static final double INITIAL_TEMPERATURE = 1000;
private static final double DECREASE_FACTOR = 1 - 10. / MAX_ITERATIONS;

public Solution minimize() {
    double temperature = INITIAL_TEMPERATURE;
    Solution currentSolution = buildInitialSolution();
    double currentCost = calculateCost(currentSolution);

    for (int i = 0; i < MAX_ITERATIONS; i++) {
        temperature = temperature * DECREASE_FACTOR;

        Solution newSolution = mutate(currentSolution);
        double newCost = calculateCost(newSolution);

        if (newCost < currentCost
            || (new Random()).nextDouble() < Math.exp(-(newCost - currentCost) / temperature)) {

            currentSolution = newSolution;
            currentCost = newCost;
        }
    }
    return currentSolution;
}
```

Температура снижается с каждой итерацией

Решение с большей стоимостью (менее выгодное) **можем** акцептовать, но вероятность тем ниже, чем:
1) хуже решение;
2) ниже температура («остывание»).

Основные этапы расчёта оптимальных заявок

Раскладка суммы по номиналам – простая задача. Или нет?

Некоторую сумму (например, 1 500 000) разложить на фиксированные доступные номиналы (например, 100, 500, 1000, 5000).

Вместимость кассеты банкомата ограничена (например, 2000 купюр).

Количество купюр в кассете не должно быть ниже порогового значения (например, 400).

Кассовый центр работает не с отдельными купюрами, а упаковками по 10, 100, 1000 и т.д., при этом для разных номиналов могут быть разные упаковки. (Покупюрно тоже может быть.).

Нужно постараться соблюсти номинальные пропорции, например:
5000 р – 90% от суммы, 1000 р – 5%, 500 р – 2%, 100 р – 3%.

Раскладка суммы по номиналам – простая задача. Или нет?

☹ Пример 1 – невозможно соблюсти номинальные пропорции

Номинал	min	max	Упаковка	Доля
100	200	2000	100	2%
500	200	2000	100	3%
1000	200	2000	100	5%
5000	200	2000	100	90%
Сумма: 1 500 000 р				

Номинал 5000: 1 500 000 * 90% = 1 350 000 (270 купюр).

Округлим вверх – сразу набрали 1 500 000 (не подходит), округлим вниз – 1 000 000.

Номинал 1000: 1 500 000 * 5% = 75 000 (75 купюр).

Округлим вверх до 200 купюр – 200 000.

Номинал 500: 1 500 000 * 3% = 45 000 (90 купюр).

Округлим вверх до 200 купюр – 100 000.

Номинал 100: 1 500 000 * 2% = 30 000 (300 купюр)

Сумма: 1 000 000 + 200 000 + 100 000 + 30 000 = 1 330 000 – не набрали ☹

Раскладка суммы по номиналам – простая задача. Или нет?

😕 Пример 2 – невозможно соблюсти номинальные пропорции

Номинал	min	max	Упаковка	Доля
100	100	1000	100	10%
500	100	1000	100	15%
1000	100	1000	100	15%
5000	100	1000	100	50%
Сумма: 2 500 000 р				

Номинал 100: 2 500 000 * 10% = 250 000 (2500 купюр).

Округлим вниз до 1000 купюр – набрали 100 000.

Номинал 500: 2 500 000 * 15% = 375 000 (750 купюр).

Округлим вверх до 800 купюр – набрали 400 000.

Номинал 1000: 2 500 000 * 15% = 375 000 (375 купюр).

Округлим вверх до 400 купюр – набрали 400 000.

Номинал 5000: 2 500 000 * 50% = 1 250 000 (250 купюр)

Округлим вверх до 300 купюр – набрали 1 500 000.

Сумма: 100 000 + 400 000 + 400 000 + 1 500 000 = 2 400 000 – опять не набрали. ☹

Раскладка суммы по номиналам – простая задача. Или нет?

☹ Пример 3 – невозможно разложить сумму по номиналам в принципе

Номинал	min	max	Упаковка	Доля
200			100	
500			100	
2000			100	
5000			100	
Сумма: 2 500 000 р				

1 упаковка номинала 200 – это 20 000.

2 530 000 некратно 20 000.

Как составить алгоритм раскладки, учитывающий всё многообразие кейсов?

В каком порядке перебирать номиналы?

В какую сторону округлять сумму?

Что делать, если полученная сумма больше или меньше целевой?

Попробуем поставить как оптимизационную задачу

Входные данные

Номинал	min	max	Упаковка	Доля
d_1	m_1	M_1	p_1	r_1
d_2	m_2	M_2	p_2	r_2
…				
d_N	m_N	M_N	p_N	r_N
Сумма: C				

$$0 \le r_j \le 1 \text{ и } r_1 + \cdots + r_N = 1$$

Введём переменные

x_j – количество упаковок номинала d_j.

По смыслу $x_j \ge 0$ и целые.

Попробуем поставить как оптимизационную задачу

Ограничения

1) Хотим набрать сумму не меньше запрошенной:

$$d_1 p_1 x_1 + \cdots + d_N p_N x_N \geq C$$

2) По числу листов номинала:

$$m_j \leq p_j x_j \leq M_j$$

3) По соблюдению номинальных пропорций (l – допустимый «коридор» в упаковках):

$$d_j p_j (x_j - l) \leq C_j r_j \leq d_j p_j (x_j + l)$$

Целевая функция

Вместо точного равенства потребуем минимизировать превышение:

$$F(x_1, \ldots, x_N) = d_1 p_1 x_1 + \cdots + d_N p_N x_N \to \min$$

Номинальная раскладка как задача оптимизации

Задача линейного **целочисленного** программирования с N неотрицательными переменными и $4N + 1$ ограничениями.

- 😕 Переменные целочисленные – симплекс-метод из commons-math не годится.
- 🙂 Стохастическая оптимизация – «из пушки по воробьям».
- 🙂 Метод отсечения (Гомори) – теплее, но долго.
- 💪 Метод ветвей и границ (Лэнд и Дойг).

ojAlgo – лёгкая оптимизационная библиотека

oj! OPTIMATIKA

Сайт

GitHub

ojAlgo – лёгкая оптимизационная библиотека

Поддерживает следующие виды оптимизации:

- классическая линейная (непрерывная);
- классическая дискретная;
- выпуклая квадратичная (с линейными ограничениями).

Предварительно упрощает оптимизационную задачу и самостоятельно выбирает подходящий алгоритм оптимизации (solver).

Быстрые (по сравнению с commons-math) операции на матрицах.

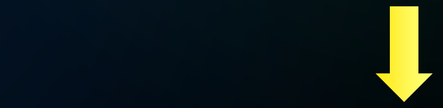
Написана на чистой Java без внешних зависимостей и нативного кода.

Реализуем номинальную раскладку на ojAlgo

```
@AllArgsConstructor
public class LayoutOptimizer {

    @AllArgsConstructor
    static class DenominationDto {

        . . .
    }

    long amount; // сумма, которую нужно разложить

    List<DenominationDto> denominations;

    int packAllowedRange; // ширина коридора

    public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {

      // TODO рассчитать и вернуть раскладку
    }
}
```

Реализуем номинальную раскладку на ojAlgo

Номинал	min	max	Упаковка	Доля
d_1	m_1	M_1	p_1	r_1
d_2	m_2	M_2	p_2	r_2
…				
d_N	m_N	M_N	p_N	r_N



```
@AllArgsConstructor
class DenominationDto {
    int denomination; // номинал
    int minNotes;     // минимальное число купюр
    int maxNotes;     // максимальное число купюр
    int packSize;     // объём упаковки
    double ratio;     // доля номинала
}
```

Реализуем номинальную раскладку на ojAlgo

```
public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {

    ExpressionsBasedModel model = new ExpressionsBasedModel();

    // если расчёт затягивается, нас устроит субоптимальное решение
    model.options.time_suffice = CalendarDateUnit.MINUTE.toDurationInMillis();
    model.options.iterations_suffice = 1000;

    // набрать не меньше требуемой суммы
    Expression amountLowerConstraint = model.addExpression("Amount")
                                            .lower(amount);

    ...
}
```

Реализуем номинальную раскладку на ojAlgo

```
public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {

    ...
    Map<Integer /*номинал*/, Variable /*переменная ojAlgo*/> denominationToVariable = new HashMap<>();


    for (DenominationDto dto : denominations) {
        Variable variable = model.addVariable("X_" + dto.denomination)
                                 // вес в целевой функции
                                 .weight((long) dto.denomination * dto.packSize)
                                 .integer(); // целочисленная переменная

        denominationToVariable.put(dto.denomination, variable);

        // вес в ограничении на сумму такой же
        amountLowerConstraint.set(variable, (long) dto.denomination * dto.packSize);

        model.addExpression("NoteCount_" + dto.denomination)
             .set(variable, dto.packSize)
             .lower(dto.minNotes).upper(dto.maxNotes);

        model.addExpression("Ratio_" + dto.denomination)
             .set(variable, (long) dto.packSize * dto.denomination)
             .lower(dto.ratio * amount - packAllowedRange * dto.packSize * dto.denomination)
             .upper(dto.ratio * amount + packAllowedRange * dto.packSize * dto.denomination);
    }

    ...
}
```

Реализуем номинальную раскладку на ojAlgo

```
public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {

    ...
    Map<Integer /*номинал*/, Variable /*переменная ojAlgo*/> denominationToVariable = new HashMap<>();


    for (DenominationDto dto : denominations) {
        Variable variable = model.addVariable("X_" + dto.denomination)
                                 // вес в целевой функции
                                 .weight((long) dto.denomination * dto.packSize)
                                 .integer(); // целочисленная переменная

        denominationToVariable.put(dto.denomination, variable);

        // вес в ограничении на сумму такой же
        amountLowerConstraint.set(variable, (long) dto.denomination * dto.packSize);

        model.addExpression("NoteCount_" + dto.denomination)
             .set(variable, dto.packSize)
             .lower(dto.minNotes).upper(dto.maxNotes);

        model.addExpression("Ratio_" + dto.denomination)
             .set(variable, (long) dto.packSize * dto.denomination)
             .lower(dto.ratio * amount - packAllowedRange * dto.packSize * dto.denomination)
             .upper(dto.ratio * amount + packAllowedRange * dto.packSize * dto.denomination);
    }

    ...
}
```

Введём переменные, сложим в map

Реализуем номинальную раскладку на ojAlgo

```
public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {

    ...
    Map<Integer /*номинал*/, Variable /*переменная ojAlgo*/> denominationToVariable = new HashMap<>();


    for (DenominationDto dto : denominations) {
        Variable variable = model.addVariable("X_" + dto.denomination)
                                 // вес в целевой функции
                                 .weight((long) dto.denomination * dto.packSize)
                                 .integer(); // целочисленная переменная

        denominationToVariable.put(dto.denomination, variable);

        // вес в ограничении на сумму такой же
        amountLowerConstraint.set(variable, (long) dto.denomination * dto.packSize);

        model.addExpression("NoteCount_" + dto.denomination)
             .set(variable, dto.packSize)
             .lower(dto.minNotes).upper(dto.maxNotes);

        model.addExpression("Ratio_" + dto.denomination)
             .set(variable, (long) dto.packSize * dto.denomination)
             .lower(dto.ratio * amount - packAllowedRange * dto.packSize * dto.denomination)
             .upper(dto.ratio * amount + packAllowedRange * dto.packSize * dto.denomination);
    }

    ...
}
```

Ограничение снизу
на сумму раскладки

Реализуем номинальную раскладку на ojAlgo

```
public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {

    ...
    Map<Integer /*номинал*/, Variable /*переменная ojAlgo*/> denominationToVariable = new HashMap<>();


    for (DenominationDto dto : denominations) {
        Variable variable = model.addVariable("X_" + dto.denomination)
                                 // вес в целевой функции
                                 .weight((long) dto.denomination * dto.packSize)
                                 .integer(); // целочисленная переменная

        denominationToVariable.put(dto.denomination, variable);

        // вес в ограничении на сумму такой же
        amountLowerConstraint.set(variable, (long) dto.denomination * dto.packSize);

        model.addExpression("NoteCount_" + dto.denomination)
             .set(variable, dto.packSize)
             .lower(dto.minNotes).upper(dto.maxNotes);

        model.addExpression("Ratio_" + dto.denomination)
             .set(variable, (long) dto.packSize * dto.denomination)
             .lower(dto.ratio * amount - packAllowedRange * dto.packSize * dto.denomination)
             .upper(dto.ratio * amount + packAllowedRange * dto.packSize * dto.denomination);
    }

    ...
}
```

Ограничение на min
и max число купюр номинала

Реализуем номинальную раскладку на ojAlgo

```
public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {

    ...
    Map<Integer /*номинал*/, Variable /*переменная ojAlgo*/> denominationToVariable = new HashMap<>();


    for (DenominationDto dto : denominations) {
        Variable variable = model.addVariable("X_" + dto.denomination)
                                 // вес в целевой функции
                                 .weight((long) dto.denomination * dto.packSize)
                                 .integer(); // целочисленная переменная

        denominationToVariable.put(dto.denomination, variable);

        // вес в ограничении на сумму такой же
        amountLowerConstraint.set(variable, (long) dto.denomination * dto.packSize);

        model.addExpression("NoteCount_" + dto.denomination)
             .set(variable, dto.packSize)
             .lower(dto.minNotes).upper(dto.maxNotes);

        model.addExpression("Ratio_" + dto.denomination)
             .set(variable, (long) dto.packSize * dto.denomination)
             .lower(dto.ratio * amount - packAllowedRange * dto.packSize * dto.denomination)
             .upper(dto.ratio * amount + packAllowedRange * dto.packSize * dto.denomination);
    }

    ...
}
```

Ограничение на долю номинала в раскладке

Реализуем номинальную раскладку на ojAlgo

```
public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {
    ...

    Optimisation.State state = model.minimise().getState();

    if (state != INFEASIBLE) {
        Map<Integer, Integer> result = new HashMap<>();

        for (DenominationDto dto : denominations) {
            int packCount = denominationToVariable.get(dto.denomination)
                                                  .getValue() // BigDecimal
                                                  .setScale(0, RoundingMode.HALF_UP)
                                                  .intValue();

            result.put(dto.denomination, packCount * dto.packSize);
        }

        return result;
    } else {
        ...
    }
}
```

Реализуем номинальную раскладку на ojAlgo

```
public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {
    ...

    Optimisation.State state = model.minimise().getState();

    if (state != INFEASIBLE) {
        Map<Integer, Integer> result = new HashMap<>();

        for (DenominationDto dto : denominations) {
            int packCount = denominationToVariable.get(dto.denomination)
                                                  .getValue() // BigDecimal
                                                  .setScale(0, RoundingMode.HALF_UP)
                                                  .intValue();

            result.put(dto.denomination, packCount * dto.packSize);
        }

        return result;
    } else {
        ...
    }
}
```

Запускаем оптимизацию, анализируем результат

Реализуем номинальную раскладку на ojAlgo

```
public Map<Integer /*номинал*/, Integer /*число купюр*/> getLayout() {
    ...

    Optimisation.State state = model.minimise().getState();

    if (state != INFEASIBLE) {
        Map<Integer, Integer> result = new HashMap<>();

        for (DenominationDto dto : denominations) {
            int packCount = denominationToVariable.get(dto.denomination)
                                                  .getValue() // BigDecimal
                                                  .setScale(0, RoundingMode.HALF_UP)
                                                  .intValue();

            result.put(dto.denomination, packCount * dto.packSize);
        }

        return result;
    } else {
        ...
    }
}
```

Вытаскиваем результат расчёта

Ранее рассмотренный пример

Номинал	min	max	Упаковка	Доля
100	200	2000	100	2%
500	200	2000	100	3%
1000	200	2000	100	5%
5000	200	2000	100	90%
Сумма: 1 500 000 р				

```
@Test
public void test() {
    List<LayoutOptimizer.DenominationDto> denominations = new ArrayList<>();

    denominations.add(new LayoutOptimizer.DenominationDto(100, 200, 2000, 100, 0.02));
    denominations.add(new LayoutOptimizer.DenominationDto(500, 200, 2000, 100, 0.03));
    denominations.add(new LayoutOptimizer.DenominationDto(1000, 200, 2000, 100, 0.05));
    denominations.add(new LayoutOptimizer.DenominationDto(5000, 200, 2000, 100, 0.9));

    Map<Integer, Integer> layout = new LayoutOptimizer(1500000L, denominations, 2).getLayout();
    assertNotNull(layout);
    System.out.println(layout); // 100 - 200 купюр, 500 - 200 купюр, 1000 - 200 купюр, 5000 - 300 купюр
    // Полученная сумма 20000 (1,1%) + 100000 (5,5%) + 200000 (11%) + 1500000 (82,4%) = 1820000.
}
```

Заключение

Что осталось за кадром

- **OptaPlanner** – мультитул для математической оптимизации.
- **Jenetics** – всё про генетические алгоритмы и эволюционные вычисления. Умеет в многокритериальную оптимизацию.
- **EJML** – операции с матрицами.
- **Symja** – символьные вычисления на Java (RIP 2016, но других нет).

✕ **RIP – библиотеки, которые давно умерли, но чат-боты их рекомендуют:**

- **JAMA** – одна из старейших, окончательно загнулась в 2010.
- **SuanShu / NM.dev** – не развивается с 2022.
- **JScience** – RIP 2011.
- **Colt / Parallel Colt** – RIP 2010.

Выводы

💪 **Даже в системе с большим количеством математических моделей реально обойтись одной лишь Java.**

- ✓ Возможно, Apache commons-math сразу решит все ваши задачи.
- ✓ Если нужен ML и вы не хотите лезть в Spark, то стоит присмотреться к Smile.
- ✓ Если хочется GPU и распределённых вычислений, то можно попробовать XGBoost4J. (Осторожно, native!)
- ✓ Многие задачи оптимизации эффективно решает легковесная ojAlgo. В матричные операции она тоже неплохо умеет.