Квартические методы первого порядка для минимизации низкого ранга

Раду-Александру Драгомир, Александр д'Аспремон, Жером Больт

Содержание

Аннотация

Мы исследуем общую невыпуклую формулировку для задач минимизации низкого ранга. Используя последние результаты по неевклидовым методам первого порядка, мы предлагаем эффективные и масштабируемые алгоритмы. Наш подход использует геометрию, задаваемую дивергенцией Брэгмана для тщательно выбранных функций ядра, вводя новое семейство граммовых квартических ядер для задач без ограничений. Численные эксперименты демонстрируют передовую производительность в задаче восстановления матрицы расстояний Евклида и симметричной неотрицательной матричной факторизации.

1 Введение

Минимизация низкого ранга является центральной задачей оптимизации с приложениями в таких областях, как машинное обучение, обработка сигналов и вычислительная биология. Она включает в себя нахождение аппроксимации матрицы низкого ранга при сохранении её ключевых характеристик, что важно для задач, таких как восстановление матриц (например, в рекомендательных системах), робастный метод главных компонент (PCA) и компрессированный сбор данных.

Распространённый подход — это факторизация Бурера-Монтейру, где положительно полуопределённая матрица Y заменяется на $Y = XX^T$, что уменьшает размер задачи, но вводит невыпуклость. Невыпуклость усложняет оптимизацию из-за наличия множества локальных минимумов, требуя тщательно разработанных методов для обеспечения сходимости к значимому решению.

Основная цель данной работы — решить эти проблемы, используя неевклидовую геометрию, которая обеспечивает гибкость при адаптации к структурам задачи. Традиционные евклидовы методы часто не справляются со сложностью задач минимизации низкого ранга из-за неспособности в полной мере учитывать их геометрию.

Предложенный подход использует дивергенцию Брэгмана — обобщение мер расстояния, адаптированных к специфической геометрии задачи. В сочетании с квартическими ядрами это позволяет эффективно проводить оптимизацию в неевклидовых пространствах. Динамическая адаптация геометрии к структуре задачи обеспечивает более высокие скорости сходимости и улучшенную масштабируемость, особенно на больших наборах данных.

В данной статье вводится семейство квартических ядер, включая нормовые и граммовые ядра, которые повышают производительность методов первого порядка. Эти ядра захватывают более богатую геометрическую информацию об области оптимизации, позволяя алгоритмам более эффективно справляться с невыпуклыми задачами. Широкие численные эксперименты демонстрируют, что предложенные методы превосходят современные подходы по скорости сходимости и качеству решений.

2 Теоретические основы

2.1 Определения и ключевые концепции

- **Относительная гладкость:** Обобщение условия Липшица для градиента. Относительная гладкость позволяет алгоритмам оптимизации работать в неевклидовых геометриях, используя свойства задачи через дивергенцию Брэгмана вместо глобальной константы Липшица для градиента. Это обеспечивает плавное изменение функции относительно выбранного ядра $h(X)$.

- **Дивергенция Брэгмана:** Определяется через строго выпуклую функцию ядра $h(X)$ и измеряет различие между двумя точками X и Y в пространстве оптимизации:

$$D_h(X, Y) = h(X) - h(Y) - \langle \nabla h(Y), X - Y \rangle.$$

В отличие от евклидовых расстояний, дивергенция Брэгмана может адаптироваться к геометрии задачи, обеспечивая более эффективную оптимизацию.

2.2 Формулировка и предположения

Задача минимизации низкого ранга переформулирована с использованием факторизации Бурера-Монтейру $Y = XX^T$, что сокращает размерность задачи, но вводит невыпуклость. Оптимизационная задача выражается как:

$$\min \Psi(X) = F(XX^T) + g(X),$$

где:

- $F(XX^T)$: Гладкая выпуклая функция, отражающая основную цель (например, ошибка восстановления).
- $g(X)$: Регуляризующий член, обеспечивающий дополнительную структуру (например, разреженность или неотрицательность).

Основное предположение состоит в том, что F является относительно гладкой относительно выбранного ядра h, а $g(X)$ достаточно прост для эффективного вычисления на каждом шаге итерации.

3 Предложенные методы

3.1 Квартические ядра

Предлагаются два типа квартических ядер, предназначенных для захвата различных геометрических свойств задачи оптимизации:

- **Нормовое ядро $h_N(X)$:** Базовое ядро, которое зависит только от фробениусовой нормы X:
$$h_N(X) = \frac{\alpha}{4}\|X\|^4 + \frac{\sigma}{2}\|X\|^2,$$
где α и σ — параметры, регулирующие вес квартического и квадратичного членов.
- **Граммовое ядро $h_G(X)$:** Более сложное ядро, которое включает дополнительную структуру через граммову матрицу X^TX:
$$h_G(X) = h_N(X) + \frac{\beta}{4}\|X^TX\|^2,$$
где β регулирует влияние граммового члена, который учитывает корреляции между столбцами X.

3.2 Алгоритмы

Предлагаемые алгоритмы основаны на структуре Dyn-NoLips, которая адаптивно регулирует шаги и использует свойства квартических ядер:

- **Dyn-NoLips с нормовым ядром:** Для этого варианта оптимизация использует нормовое ядро $h_N(X)$. Итеративное обновление имеет вид:
$$X_{k+1} = \text{argmin}_U \left\{ g(U) + \langle \nabla F(X_k), U - X_k \rangle + \frac{1}{\lambda_k} D_{h_N}(U, X_k) \right\},$$
где λ_k — адаптивный шаг.
- **Dyn-NoLips с граммовым ядром:** Для этого варианта используется более богатое граммовое ядро $h_G(X)$. Итеративное обновление аналогично, но с $D_{h_G}(U, X_k)$:
$$X_{k+1} = \text{argmin}_U \left\{ g(U) + \langle \nabla F(X_k), U - X_k \rangle + \frac{1}{\lambda_k} D_{h_G}(U, X_k) \right\}.$$

3.3 Сравнение ядер

- **Нормовое ядро:** Простое и вычислительно дешевое. Эффективно для общих задач минимизации низкого ранга, но может быть менее подходящим для данных со сложной структурой.
- **Граммовое ядро:** Захватывает более богатую геометрическую информацию, обеспечивая лучшую производительность в задачах с корреляциями между переменными. Однако оно вычислительно более затратное из-за $\|X^T X\|^2$.

Благодаря динамической адаптации геометрии задачи с использованием этих ядер предложенные методы достигают более высоких скоростей сходимости и лучшей масштабируемости по сравнению с традиционными евклидовыми подходами.

4 Сравнительный анализ

Для оценки эффективности предложенных методов были использованы различные алгоритмы, включая как современные подходы, так и предложенный Dyn-NoLips. В этом разделе кратко описаны основные алгоритмы, использованные в сравнении, их принципы работы и вычислительные сложности.

4.1 Используемые алгоритмы

- **Dyn-NoLips:** Динамический метод первого порядка, основанный на дивергенции Брэгмана. Использует адаптивный шаг λ_k, чтобы обеспечить достаточное уменьшение функции. Итеративное обновление имеет вид:

$$T_\lambda(X) = \arg\min_U \left\{ g(U) + \langle \nabla f(X), U - X \rangle + \frac{1}{\lambda} D_h(U, X) \right\},$$

  где $D_h(U, X)$ определяется через выбранное ядро (нормовое h_N или граммовое h_G). Сложность одной итерации: $O(nr^2)$ для h_N и $O(nr^2 + r^3)$ для h_G.

- **Beta-SNMF:** Мультипликативный метод обновления для задач симметричной неотрицательной матричной факторизации (SymNMF). Использует фиксированный параметр β для регулирования обновлений. Формула обновления:

$$X_{ij} \leftarrow X_{ij} \frac{(MX)_{ij}^{\beta}}{(XX^T X)_{ij}^{\beta}}.$$

  Отличается простотой реализации, но требует подбора параметра β.

- **PG (Projected Gradient):** Метод проекционного градиента, обновляющий переменные с проекцией на допустимое множество. Итерация имеет вид:

$$X_{k+1} = \text{proj}_{\text{constraint}}(X_k - \alpha_k \nabla f(X_k)),$$

  где α_k выбирается с помощью поиска по Армихо. Сложность определяется проекцией: $O(nr)$ для каждой итерации.

- **CD (Coordinate Descent):** Метод координатного спуска, минимизирующий функцию по одной переменной за раз. Формула обновления:

$$X_{ij} \leftarrow \arg\min_{x \geq 0} f(X_{ij} \mid \text{фиксированы остальные элементы}).$$

  Эффективен для задач с большой размерностью, где сложность одной итерации пропорциональна числу координат.

- **SymANLS:** Чередующийся метод наименьших квадратов для SymNMF. На каждой итерации решается подзадача:
$$X \leftarrow \arg\min_{X \geq 0} \|M - XX^T\|_F^2.$$
Сложность итерации: $O(nr^2)$.

- **SymHALS:** Улучшенный чередующийся метод, обновляющий одну строку или столбец за раз. Формула обновления:
$$X_{i,:} \leftarrow \arg\min_{x \geq 0} \|M - XX^T\|_F^2.$$
Более эффективен на больших наборах данных. Сложность итерации аналогична SymANLS.

4.2 Сравнение

Для оценки методов были использованы следующие метрики:

- **Время сходимости:** время, необходимое для достижения заданного критерия сходимости.
- **Разрыв функции цели:**
$$f(X_k) - f_{\min},$$
где $f_{\min}$ — минимальное значение функции цели среди всех инициализаций.

Dyn-NoLips продемонстрировал вторую лучшую производительность после SymHALS на задачах SymNMF, превосходя другие методы по стабильности и отсутствию необходимости в настройке гиперпараметров. В задаче восстановления евклидовых матриц расстояний (EDMC) метод Dyn-NoLips-Gram показал наилучшую сходимость, что иллюстрирует преимущества граммовой геометрии.

5 Код для тестирования

В этом разделе описан пример кода на языке Julia, который используется для запуска экспериментов. Приведенный код включает модули и функции для различных алгоритмов, подготовки данных и их обработки.

5.1 Структура кода

Код состоит из следующих частей:

1. **Загрузка библиотек**: загрузка необходимых модулей и пакетов.
2. **Инициализация**: определение основных параметров алгоритмов.
3. **Подготовка данных**: загрузка и обработка данных в зависимости от выбранного набора данных.
4. **Запуск алгоритмов**: выполнение алгоритмов с заданными параметрами и сохранение результатов.
5. **Анализ результатов**: визуализация потерь и значения финальной точности.

5.2 Пример кода

Ниже приведен код, который выполняет вышеуказанные шаги:

Листинг 1: Загрузка библиотек

```
include("utils.jl")
include("PG.jl") # projected gradient
include("BPG.jl") # Bregman proximal gradient algorithms (Nolips and
    variants)
include("Beta.jl")
include("CD.jl")
include("SymHALS.jl")
include("SymANLS.jl")

include("SymNMF.jl") # wrapper for all SymNMF solvers

using LinearAlgebra
using PyPlot
using Random
using NPZ
using SparseArrays
using JLD
```

Листинг 2: Инициализация параметров

```
algos = [:pga, :nolips, :acc_nolips, :dyn_nolips, :beta, :cd, :
   sym_hals]
algo_names = Dict(:pga => "PG", :dyn_nolips => "Dyn-Nolips",
    :beta => "Beta", :cd => "CD",
    :sym_hals => "SymHALS", :sym_anls => "SymANLS")

algo_params = Dict()

algo_params[:pga] = (step = 1., beta = 0.1, sigma = 0.01,
   max_inner_iter = 20)
algo_params[:dyn_nolips] = (step = 1., gamma_inc = 2., gamma_dec =
   2.)
algo_params[:beta] = (beta = 0.99,)
algo_params[:cd] = ()
algo_params[:sym_hals] = (mu = 1e-2,)
```

Листинг 3: Загрузка данных

```
possible_choices = [:synth500, :synth1000, :cbcl, :coil20, :tdt2, :
   reuters]

dataset = :tdt2 # CHANGE HERE

max_iter = 12000
max_time = 10.
monit_acc = true;
y_true = [0]
```

```
function load_dataset(ds_name)
    file_content = load("data/$ds_name.jld")
    return file_content["M"], file_content["labels"]
end

if dataset == :synth500 # synthetic dataset
    mu = 1e1
    max_time = 5.
    n, r = 500, 20
    M, r = synthetic_SNMF(n, r), r;
    monit_acc = false

elseif dataset == :synth1000
    mu = 1e1
    max_time = 10.
    n, r = 1000, 30
    M, r = synthetic_SNMF(n, r), r;
    monit_acc = false

elseif dataset == :cbcl
    mu = 1e-2
    max_time = 10.
    r = 20
    M, y_true = load_dataset("cbcl")
    monit_acc = false

elseif dataset == :coil20
    mu = 1e-2
    max_time = 20.
    M, y_true = load_dataset("coil20")
    r = 20

elseif dataset == :tdt2
    mu = 1e-1
    max_time = 20.
    M, y_true = load_dataset("tdt2")
    r = 30

elseif dataset == :reuters
    mu = 1e-1
    max_time = 20.
    M, y_true = load_dataset("reuters")
    r = 25

elseif dataset == :orl
    mu = 1e-2
    max_time = 10.
    M, y_true = load_dataset("orl")
    r = 40
end
algo_params[:sym_hals] = (mu = mu,)
```

```
algo_params[:sym_anls] = (mu = mu,)

monitoring_interval = max_time / 20.;
```

Листинг 4: Запуск алгоритмов

```
function run_algo(algo::Symbol, A0::Matrix{Float64}, algo_params)
    SymNMF(M, r; algo = algo,
        max_iter = max_iter,
        max_time = max_time,
        monitoring_interval = monitoring_interval,
        A_init = A0,
        monitor_accuracy = monit_acc,
        true_labels = y_true,
        algo_params...);
end;

n_runs = 5 # number of runs to average

algos_to_run = [:pga, :dyn_nolips, :beta, :cd,
    :sym_hals, :sym_anls]

all_losses = Dict()
clust_accs = Dict()
n_measures = Dict()
min_loss = Inf

for t = 1:n_runs
    println("Run number $t / $n_runs ...")
    A0 = random_init_sym(M, r);

    for algo = algos_to_run
        params, name = algo_params[algo], algo_names[algo]

        A, losses = run_algo(algo, A0, params)

        # updating minimal loss
        min_loss = min(min_loss, minimum(losses[:,2]))

        if t == 1
            all_losses[algo] = losses
            clust_accs[algo] = [losses[end,:4]]
        else
            push!(clust_accs[algo], losses[end,:4])

            # a procedure to ensure that all losses measures have
               same size,
            # in order to average them
            n_meas_old = size(all_losses[algo],1)
            n_meas_new = min(n_meas_old, size(losses,1))

            trimmed_old_losses = all_losses[algo][1:n_meas_new,:]
```

```
            trimmed_losses = losses[1:n_meas_new,:]

            all_losses[algo] = ((t - 1) * trimmed_old_losses +
               trimmed_losses) / t
        end
    end
end

all_losses[:min_loss] = min_loss
```

Листинг 5: Графики и анализ

```
fig, ax = subplots()

markers = Dict(:pga => ".", :nolips => "v", :acc_nolips => "s",
    :dyn_nolips => "D", :cd => "None",
    :sym_hals => "^", :sym_anls => "h", :dyn_acc_nolips => "s", :
       beta => "None")


algos_to_show = [:dyn_nolips, :beta, :pga, :cd, :sym_anls, :sym_hals
   ]

linestyles = Dict()
for algo = algos_to_show
   linestyles[algo] = "-"
end

linestyles[:beta] = "--"
linestyles[:cd] = "-."

function plot_loss(axis, losses, label, marker, ls)
    min_loss = all_losses[:min_loss]
    axis.plot(losses[:,1], losses[:,2] .- min_loss, label = label,
        marker = marker, linestyle = ls, linewidth = 2)
end

for algo = algos_to_show
   plot_loss(ax, all_losses[algo], algo_names[algo], markers[algo],
        linestyles[algo])
end

fontsize = 12
ax.set_xlabel("Time (seconds)", fontsize = fontsize)

ax.set_ylabel("f(X^k) - f_min", fontsize = fontsize)

ax.legend(fontsize = fontsize, loc = 1);
ax.set_yscale("log");

n = size(M, 1)
title("Results on dataset $dataset, with n = $n, r = $r. Averaged
```

```
    over $n_runs runs")
# ax1[:set_xlim](0, 5.)

println("Clustering accuracies \n")
for algo = algos_to_run
    println(algo_names[algo], "\t", mean(clust_accs[algo]))
end
```

5.3 Используемые библиотеки

Для работы кода требуется установить следующие библиотеки Julia:

- **LinearAlgebra**: операции с матрицами.
- **PyPlot**: построение графиков.
- **Random**: генерация случайных чисел.
- **NPZ**: работа с файлами в формате .npz.
- **SparseArrays**: работа с разреженными матрицами.
- **JLD**: работа с файлами формата .jld.

6 Обсуждение сторонних библиотек

6.1 Общие сведения

Код основывается на библиотеке **SymNMF**, которая предоставляет реализацию различных алгоритмов для симметричной неотрицательной матричной факторизации. Подробнее:

- **SymNMF/SymNMF.jl**: оболочка для тестирования и мониторинга всех алгоритмов.
- **SymNMF/BPG.jl**: алгоритм Dyn-NoLips.
- **SymNMF/Beta.jl**: алгоритм Beta-SNMF.
- **SymNMF/CD.jl**: алгоритм координатного спуска.
- **SymNMF/PG.jl**: проекционный градиент.
- **SymNMF/utils.jl**: функции для инициализации и оценки.

Подробная документация находится в репозитории QuarticLowRankOptimization.

6.2 Используемые датасеты и решаемые задачи

Эксперименты проводились на следующих датасетах:

- **:synth500, :synth1000:** Синтетические данные размером 500 и 1000 строк, используемые для проверки масштабируемости алгоритмов на данных с известным ранком.
- **:cbcl:** Набор данных с изображениями лиц, применяемый для задач восстановления матриц.
- **:coil20:** Набор изображений объектов, используемый для кластеризации и анализа структуры данных.
- **:tdt2, :reuters:** Текстовые датасеты, предназначенные для задач кластеризации документов и моделирования тем.

7 Экспериментальные результаты

- **Наборы данных:** CBCL (лица), Coil-20 (объекты), TDT2 и Reuters (тексты).
- **Метрики:** Время сходимости, нормализованный разрыв функции цели и среднеквадратичная ошибка.
- **Наблюдения:** Dyn-NoLips превосходит существующие методы по масштабируемости и настройке параметров.

7.1 Результаты на датасете CBCL

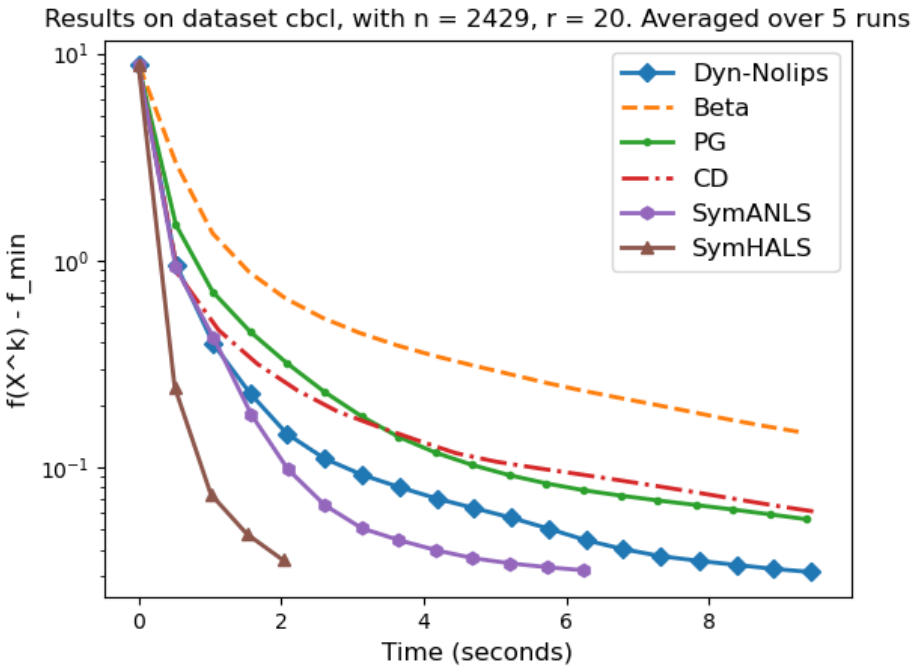

Рис. 1: Зависимость ошибки от времени для CBCL

Таблица 1: Кластерная точность и другие данные для CBCL

Алгоритм	Точность	Количество итераций	Время (секунды)
PG	0.000	19	9.366
Dyn-Nolips	0.000	19	9.433
Beta	0.000	19	9.302
CD	0.000	18	9.522
SymHALS	0.000	5	2.026
SymANLS	0.000	13	6.246

7.2 Результаты на датасете Coil-20

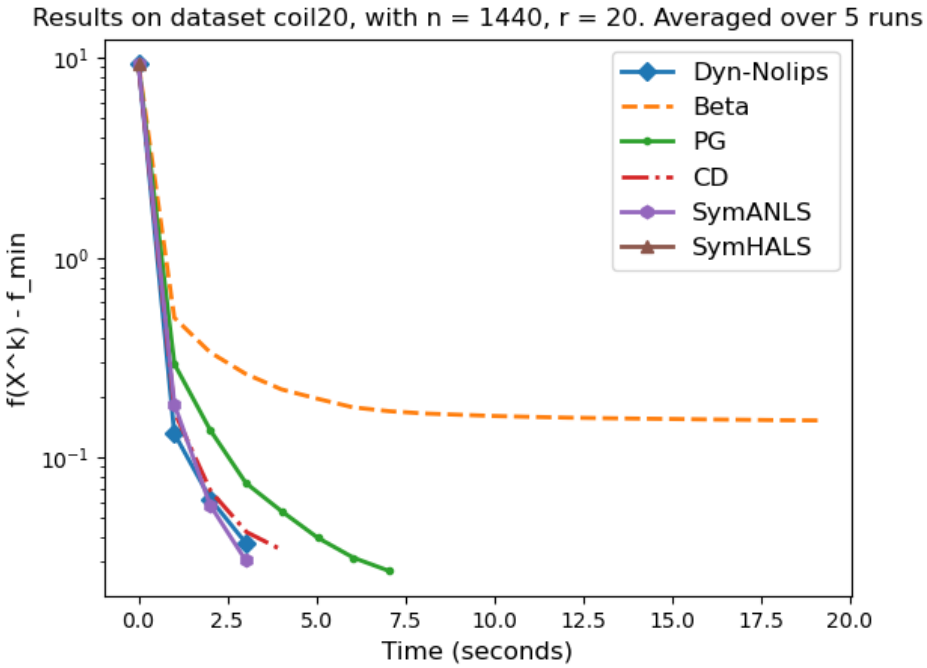

Рис. 2: Зависимость ошибки от времени для Coil-20

Таблица 2: Кластерная точность и другие данные для Coil-20

Алгоритм	Точность	Количество итераций	Время (секунды)
PG	0.744	8	7.05808
Dyn-Nolips	0.730	4	3.03284
Beta	0.627	20	19.10743
CD	0.758	5	4.05985
SymHALS	0.653	1	0.00003
SymANLS	0.773	4	3.02128

7.3 Результаты на датасете TDT2

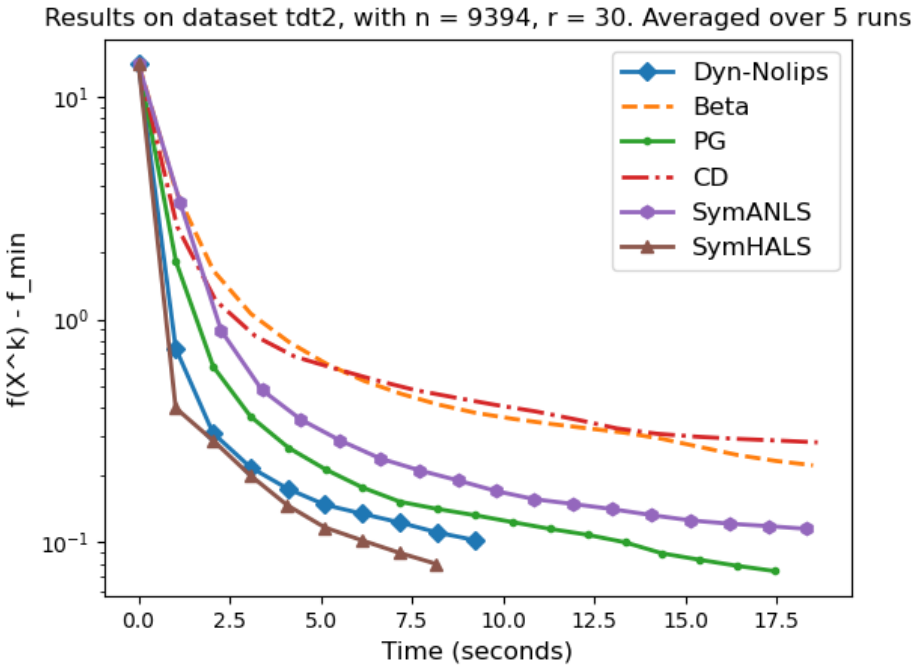

Рис. 3: Зависимость ошибки от времени для TDT2

Таблица 3: Кластерная точность и другие данные для TDT2

Алгоритм	Точность	Количество итераций	Время (секунды)
PG	0.855	18	17.427
Dyn-Nolips	0.858	10	9.235
Beta	0.782	19	18.490
CD	0.785	18	18.616
SymHALS	0.885	9	8.161
SymANLS	0.835	18	18.291

7.4 Результаты на датасете Reuters

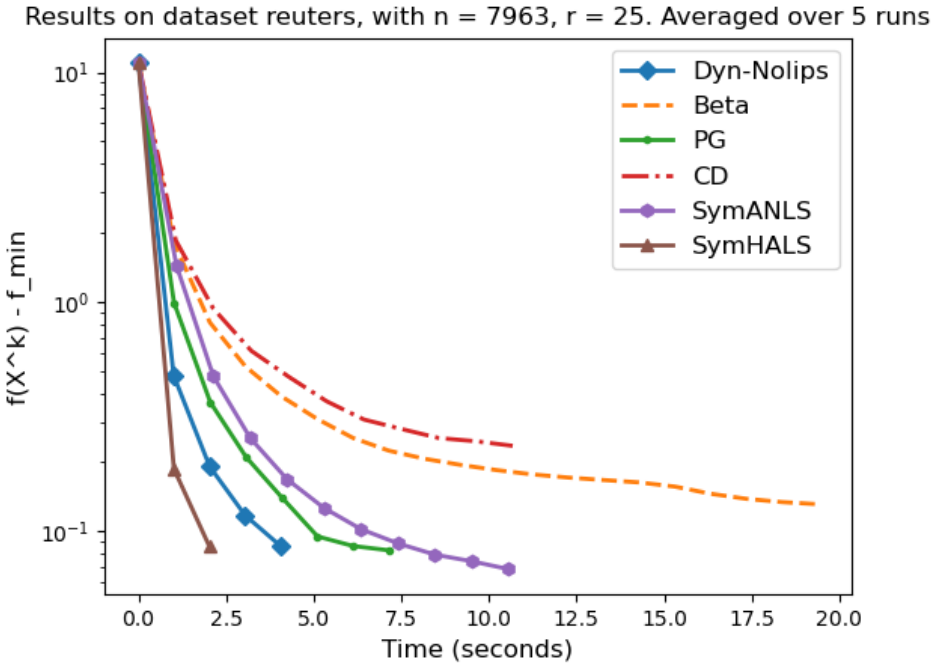

Рис. 4: Зависимость ошибки от времени для Reuters

Таблица 4: Кластерная точность и другие данные для Reuters

Алгоритм	Точность	Количество итераций	Время (секунды)
PG	0.436	8	7.151
Dyn-Nolips	0.439	5	4.071
Beta	0.456	20	19.377
CD	0.400	11	10.678
SymHALS	0.426	3	2.024
SymANLS	0.449	11	10.561

8 Выводы

- Разработаны масштабируемые алгоритмы для невыпуклой минимизации низкого ранга.
- Предложены новые квартические ядра для улучшения скоростей сходимости.
- Будущая работа: инерционные варианты и новые дизайны ядер для более широких приложений.

Список литературы

- Candès, E.J., Recht, B.: Exact Matrix Completion via Convex Optimization. Found. Comput. Math. (2009).
- Burer, S., Monteiro, R.D.C.: Local Minima in Low-Rank Semidefinite Programming. Math. Program. (2005).

- Bolte, J., Teboulle, M.: First-Order Methods Beyond Lipschitz Gradient Continuity. Math. Oper. Res. (2017).