Методы первого порядка с полиномиальными ядрами четвёртой степени для минимизации низкого ранга

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

Содержание

Аннотация

Мы изучаем невыпуклую формулировку для минимизации низкого ранга. Используя методы первого порядка с дивергенцией Брэгмана, предлагаем эффективные алгоритмы с новой геометрией, задаваемой полиномиальными ядрами четвертой степени. Вводим нормовые и ядра Грама, последнее из которых используется для задач без ограничений. Численные эксперименты показывают высокую производительность в симметричной неотрицательной матричной факторизации и восстановлении матриц.

1 Введение

Минимизация низкого ранга — важная задача оптимизации с приложениями в машинном обучении, обработке сигналов и биологии. Она направлена на нахождение низкоранговых аппроксимаций матриц с сохранением ключевых характеристик, что важно для задач, таких как восстановление матриц (например, рекомендательные системы) и устойчивый PCA.

Факторизация Бурера-Монтейру, заменяющая матрицу Y на $Y = XX^T$, уменьшает размер задачи, но вводит невыпуклость, усложняя оптимизацию. Мы предлагаем методы, основанные на дивергенции Брэгмана, адаптированной к геометрии задачи, что позволяет решать её эффективнее, чем стандартные евклидовы методы.

Наш подход использует полиномиальные ядра четвертой степени: нормовые для задач с ограничениями и ядра Грама для задач без ограничений. Эти ядра учитывают геометрические особенности пространства оптимизации, обеспечивая быструю сходимость и масштабируемость на больших наборах данных. Численные эксперименты подтверждают эффективность методов, особенно на разреженных и крупномасштабных данных.

2 Теоретические основы

2.1 Определения и ключевые концепции

- **Относительная гладкость:** Обобщение условия Липшица для градиента. Относительная гладкость позволяет учитывать свойства задачи через дивергенцию Брэгмана вместо использования глобальной константы Липшица для градиента в евклидовой метрике. Это позволяет адаптировать оптимизацию к геометрии задачи, заданной ядром $h(X)$.

- **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 является асимметричной и 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 усложняет анализ из-за её невыпуклости. 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, неотрицательность или другие ограничения).

Основное предположение состоит в том, что F является относительно гладкой относительно выбранного ядра h, а $g(X)$ позволяет эффективно вычислять операции оптимизации (например, проекции или обновления) на каждом шаге итерации.

3 Предложенные методы

3.1 Полиномиальные ядра четвертой степени

Предлагаются два типа полиномиальных ядер четвертой степени, 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 членов второй и четвертой степени.

- **Ядро Грама $h_G(X)$:** 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 полиномиальных ядер четвертой степени:

- **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 ядром Грама:** Для этого варианта используется более богатое ядро Грама $h_G(X)$. 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.

- **Ядро Грама:** Захватывает более богатую геометрическую информацию, обеспечивая лучшую производительность в задачах с корреляциями между переменными. Однако оно вычислительно более затратное из-за $\|X^TX\|^2$.

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 (нормы h_N или Грама 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:

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

  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.
- **Целевая разность:**
$$f(X_k) - f_{\min},$$
где $f_{\min}$ — минимальное значение целевой функции среди всех инициализаций.

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 Основной код

5.1.1 загрузка библиотек

Загрузка необходимых модулей и пакетов.

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
```

5.1.2 Инициализация

Определение основных параметров алгоритмов.

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,)
```

5.1.3 Подготовка данных

Загрузка и обработка данных в зависимости от выбранного набора данных.

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.;
```

5.1.4 Запуск алгоритмов

Выполнение алгоритмов с заданными параметрами и сохранение результатов.

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
```

5.1.5 Анализ результатов

Визуализация потерь и значения финальной точности.

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
```

5.2 Стандартные 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.

5.3 Сторонние библиотеки

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.

5.4 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. (Не представлены в данной работе)
- **: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.

6 Экспериментальные результаты

- **Наборы данных:** CBCL (лица), Coil-20 (объекты), TDT2 и Reuters (тексты).
- **Метрики:** Время сходимости, целевая разность, кластерная точность и количество итераций.

6.1 Результаты на датасете CBCL

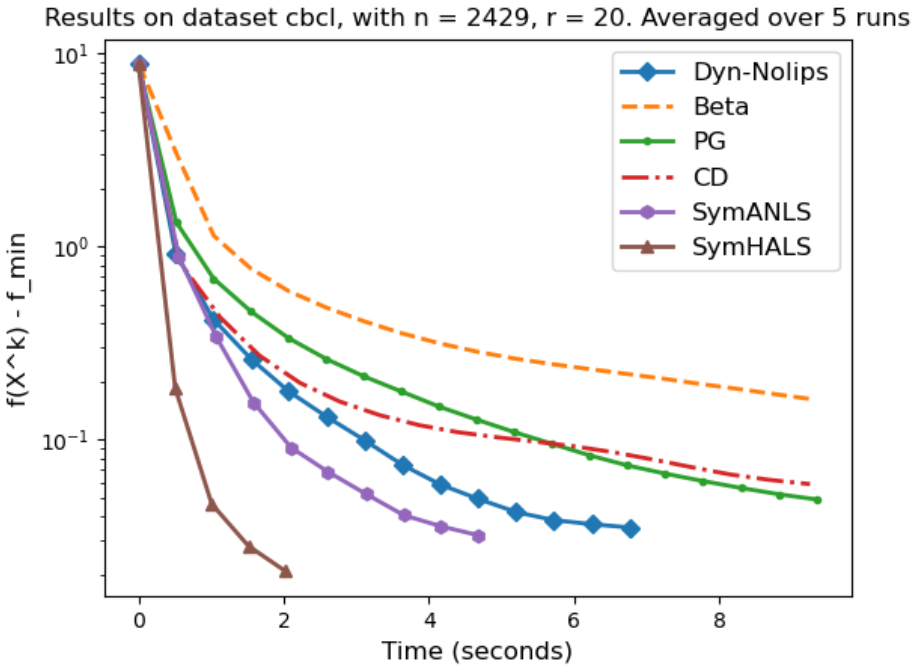

Рис. 1: Зависимость ошибки от времени для CBCL

Таблица 1: Данные для CBCL

Алгоритм	Количество итераций	Время (секунды)	Целевая разность
PG	19	9.351	0.049
Dyn-Nolips	14	6.767	0.035
Beta	19	9.288	0.162
CD	18	9.246	0.059
SymHALS	5	2.02	0.021
SymANLS	10	4.686	0.032

6.2 Результаты на датасете Coil-20

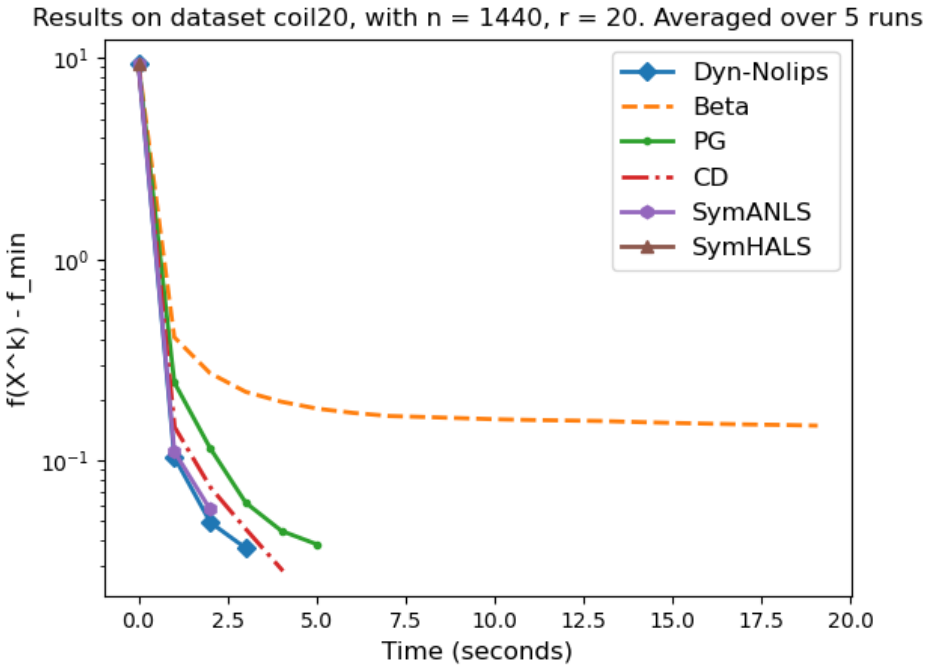

Рис. 2: Зависимость ошибки от времени для Coil-20

Таблица 2: Данные для coil20

Алгоритм	Кластерная точность	Количество итераций	Время (секунды)	Целевая разность
PG	0.717	6	5.029	0.038
Dyn-Nolips	0.72	4	3.026	0.037
Beta	0.616	20	19.088	0.149
CD	0.759	5	4.062	0.028
SymHALS	0.492	1	0.0	9.326
SymANLS	0.727	3	2.012	0.057

6.3 Результаты на датасете TDT2

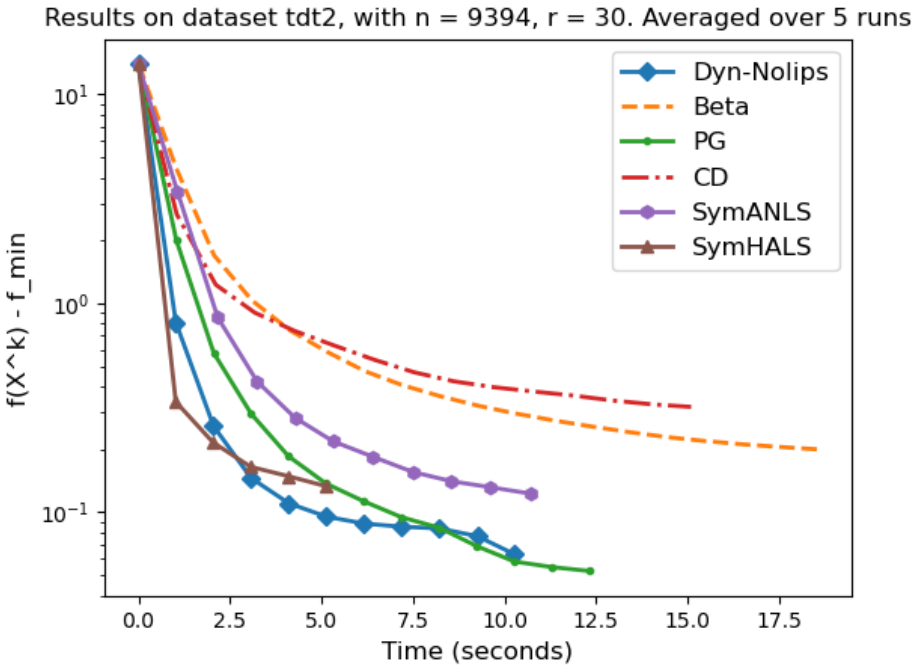

Рис. 3: Зависимость ошибки от времени для TDT2

Таблица 3: Данные для TDT2

Алгоритм	Кластерная точность	Количество итераций	Время (секунды)	Целевая разность
PG	0.849	13	12.332	0.052
Dyn-Nolips	0.855	11	10.284	0.063
Beta	0.781	19	18.557	0.2
CD	0.801	15	15.102	0.319
SymHALS	0.841	6	5.105	0.134
SymANLS	0.828	11	10.709	0.123

6.4 Результаты на датасете Reuters

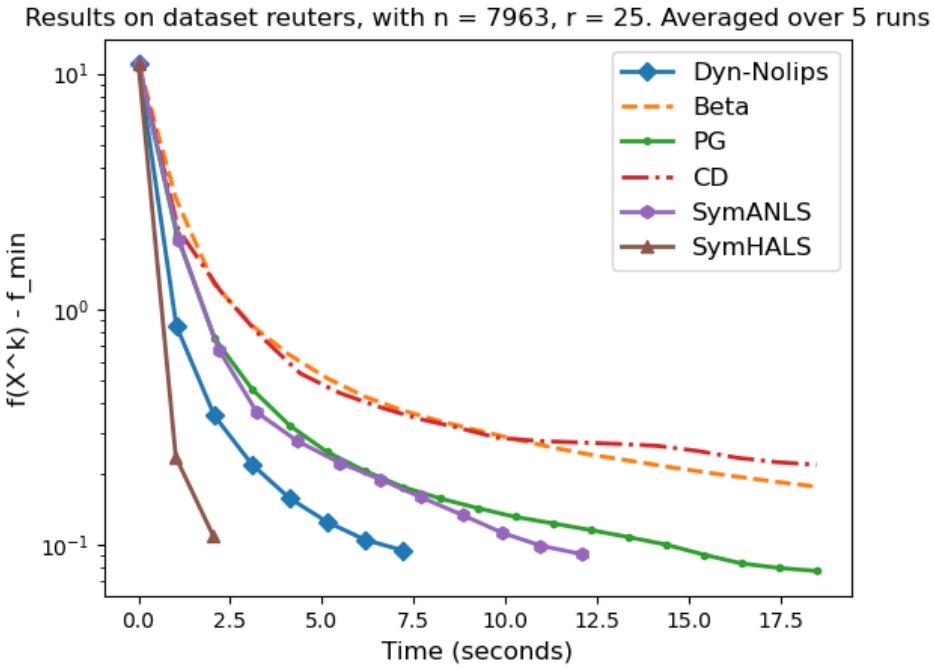

Рис. 4: Зависимость ошибки от времени для Reuters

Таблица 4: Данные для reuters

Алгоритм	Кластерная точность	Количество итераций	Время (секунды)	Целевая разность
PG	0.488	19	18.507	0.078
Dyn-Nolips	0.48	8	7.213	0.095
Beta	0.383	19	18.518	0.177
CD	0.376	18	18.491	0.219
SymHALS	0.452	3	2.037	0.11
SymANLS	0.486	12	12.084	0.091

7 Заключение

Dyn-NoLips — это надёжный и универсальный метод, особенно подходящий для больших и разреженных задач, таких как текстовая кластеризация (TDT2, Reuters). Хотя он уступает SymHALS в скорости сходимости, его способность работать без подбора параметров делает его предпочтительным выбором в ситуациях, где требуется высокая автоматизация и устойчивость.

Если точность кластеризации имеет наивысший приоритет, SymANLS может быть предпочтительным выбором, однако это достигается ценой более сложной настройки.