Проект по непрерывной оптимизации

Самородова Екатерина, Кускаров Тагир, Ровник Юлия

4 июня 2020 г.

Содержание

1 Введение

Данный проект состоит в программной реализации одной из модификаций метода Гаусса-Ньютона — известного численного метода оптимизации. Метод Гаусса-Ньютона используется для итерационного численного решения задач нелинейной регрессии [2]. Однако оригинальный метод требует вычисления обратной матрицы к гессиану на каждой итерации, что является очень трудоемкой операцией в случае больших размерностей. “Гибкая” модификация метода позволяет использовать некоторые особенности внешнего вида матрицы для ускорения алгоритма, переход к стохастической версии градиентного спуска, а также вариьирование длины шага на каждой итерации с помощью метода экстраполяции и правила Армихо.

2 Алгоритм Гаусса-Ньютона

Алгоритм Гаусса-Ньютона — это итеративный численный метод нахождения решения задачи наименьших квадратов отклонений. Так, пусть задано m функций в n-мерном пространстве $\mathbb{R}^n \to \mathbb{R} := g_k(x)$, тогда задача состоит в отыскании точки

$$x^* = \arg\min_{x \in \mathbb{R}^n} \left\{ f(x) := \sum_{k=1}^{m} g_k^2(x) \right\}.$$

Пусть $F(x) = [g_k^2(x)]_{k=1}^m$ – вектор-столбец, а $F' = J \in \mathbb{R}^{m \times n}$ — матрица Якоби, определяющаяся следующим образом: $J_{ij} = \frac{\partial g_i(x)}{\partial x_j}$. Тогда итерация алгоритма выглядит так:

$$x_{t+1} = x_t - \left[J^T(x_t) J(x_t) \right]^{-1} J^T(x_t) F(x_t)$$

Преимущество данного метода перед классическим методом Ньютона состоит в том, что нет необходимости вычислять матрицу вторых производных (гессиан) на каждом шаге, что может быть весьма затруднительно. Рассматриваемая в данном случае модицификация предлагает стохастический метод и более гибкий подход к выбору промежуточных параметров.

3 Модифицированная версия алгоритма

Текущая постановка задачи:

$$f(x) := ||F(x)||_2 \to \min,\ F(x) = (F_1(x), \ldots, F_m(x))$$

Дополнительно нормализуем целевую функцию:

$$\hat{f}_1 = \frac{f^2(x)}{\sqrt{m}} = \frac{||F(x)||^2}{\sqrt{m}} = \left(\frac{f_2(x)}{m} \right)^2;\ \hat{f}_2 = \frac{f^2(x)}{m}$$

Нормализация позволяет разрешить $m \to \infty$. Тогда стохатическая версия алгоритма [4] будет выглядеть следующим образом [1].

Algorithm 1 Стохастический метод наименьших квадратов

Зафиксируем $L_0 > 0$ и размер батча $p \in \{0, \dots, m\}$
for $k \geq 0$ **do**
$I_k \subseteq \{1, \dots, m\}$, где $|I_k| - p$
$G_k = \{F_i'(x_k),\ i \in I_k\}$
$\phi_k(y) = \hat{f}_1(x_k) + \langle \hat{f}_1'(x_k), y - x_k \rangle + \frac{1}{2\hat{f}_1(x_k)} \left(\frac{1}{p} ||G_k^T(y - x_k)||^2 \right)$
Найдем наименьший $i_k \geq 0$, для которого выполнено:
Если $T_{ik} = \text{argmin}\ \{\psi_{ik}(y) = \phi_k(y) + 2^{i_k - 1} L_k ||y - x_k||^2\}$, то $\hat{f}_1(T_{ik}) \leq \psi_{ik}(T_{ik})$
$x_{k+1} = T_{ik}$
$L_{k+1} = 2^{i_k - 1} L_k$

4 Детали программной реализации

4.1 Минимизация $\hat{\psi}(x, i, L)$

Одним из пунктов алгоритма, описанного в предыдущем пункте, является вычисления на каждой итерации точки

$$T_{i_k} = \arg\min_y \{\psi_{i_k}(y, x_k, i_k, L_k).\}$$

Данная точка может быть найдена явно из условия первого порядка для функции ψ:

$$F'(x)^T \left[F(x) + F'(x)(T - x) \right] + L f_1(x)(T - x) = 0,$$

что можно переписать как

$$\left[F'(x)^T F'(x) + L f_1(x) I \right] T = L f_1(x) \cdot x - F'(x)^T F(x).$$

Это выражение является системой линейных уравнений относительно неизвестного вектора T и имеет единственное решение

$$T = \left[F'(x)^T F'(x) + L f_1(x) I \right]^{-1} \cdot \left[L f_1(x) \cdot x - F'(x)^T F(x) \right].$$

Обращение матрицы является вычислительно сложной задачей, однако её структура позволяет применить некоторые оптимизации. Допустим, что матрицу Якоби $F'(x)$ удалось разложить в произведение $U^T B V$, где U и V — ортогональные матрицы, а B — *бидиагональная* матрица, то есть такая матрица, у которой все ненулевые элементы находятся на главной диагонали, а также на диагонали, соседней с ней.

В таком случае можно преобразовать выражение для матрицы, которую нужно обратить:

$$\begin{aligned}\left[F'(x)^T F'(x) + Lf_1(x)I\right]^{-1} &= \left[(U^T BV)^T U^T BV + Lf_1(x)I\right]^{-1} = \\ &= \left[V^T B^T \underbrace{UU^T}_{=I} BV + Lf_1(x)\underbrace{V^T IV}_{=I}\right]^{-1} = \\ &= \left[V^T\left(B^T B + Lf_1(x)I\right)V\right]^{-1} = \\ &= V^T\left[B^T B + Lf_1(x)I\right]^{-1} V\end{aligned}$$

Матрица $B^T B$ является *тридиагональной* и симметричной, так же как и $B^T B + Lf_1(x)I$, и может быть обращена эффективно.

Известно, что тридиагональная матрица может быть обращена за квадратичную сложность [3]. В статье [4] указывается, что алгоритм обращения может быть улучшен до линейной сложности $O(n)$, однако подтверждений этому найдено не было.

Кроме того, заметим, что в рамках одной итерации нужно вычислять T для разных значений i и L, но матрица Якоби зависит только от точки x_k и не меняется. Следовательно, ее бидиагонализацию можно проводить лишь один раз за итерацию.

4.2 Алгоритм бидиагонализации матрицы

В этом разделе опишем метод Голуба-Кахана [1], использующийся для получения вышеупомянутого разложения матрицы Якоби по алгоритму [2], псевдокод которого представлен ниже.

Algorithm 2 Бидиагонализация Голуба-Кахана

A $\in Mat_{p\times q}(\mathbb{R})$ – исходная матрица
$n = \min(p, q)$ – количество элементов на главной диагонали
$v_1 = e_n$ – единичный вектор длины n
$\beta_0 = 0$
for k in $1 \dots n$ **do**
    $u_k = \mathrm{A}v_k$ - $\beta_{k-1} \cdot u_{k-1}$
    $\alpha_k = ||u_k||_2$
    $u_k = \frac{u_k}{\alpha_k}$
    $v_{k+1} = \mathrm{A}^T u_k$ - $\alpha_k \cdot v_k$
    if $k + 1 \leq n$ **then**
        $\beta_k = ||v_{k+1}||_2$
        $v_{k+1} = \frac{v_{k+1}}{\beta_k}$

Результатом работы алгоритма является набор значений $\alpha_1 \dots \alpha_n$, которые задают главную диагональ матрицы B, $\beta_1 \dots \beta_{n-1}$, которые задают значения на диагональ выше (в данном случае используется верхнедиагональный вид), а также полученные матрицы U и V. При условии, что все вычисления происходят абсолютно точно, выполняется равенство $B = U^T \cdot A \cdot V$.

Так как арифметические операции с дробными числами в компьютере имеют погрешность, которая накапливается со временем, необходимо убедиться в его достаточной численной устой-

чивости, прежде чем применять его на практике.

Для этого алгоритм был протестирован на случайных матрицах различного размера, где значения элементы $A_{ij} \in (0, 1]$ распределены равномерно. Для каждого эксперимента был посчитан максимум поэлементой разницы $B - U^T \cdot A \cdot V$, затем была построена гистограмма таких значений, полученных за 1000 тестов 1.

Получается, что абсолютное большинство тестов дает примерно нулевую разницу, при этом с уменьшением n (например, для длинных и узких прямоугольных матриц) “хвост” 1 распределения становится меньше.

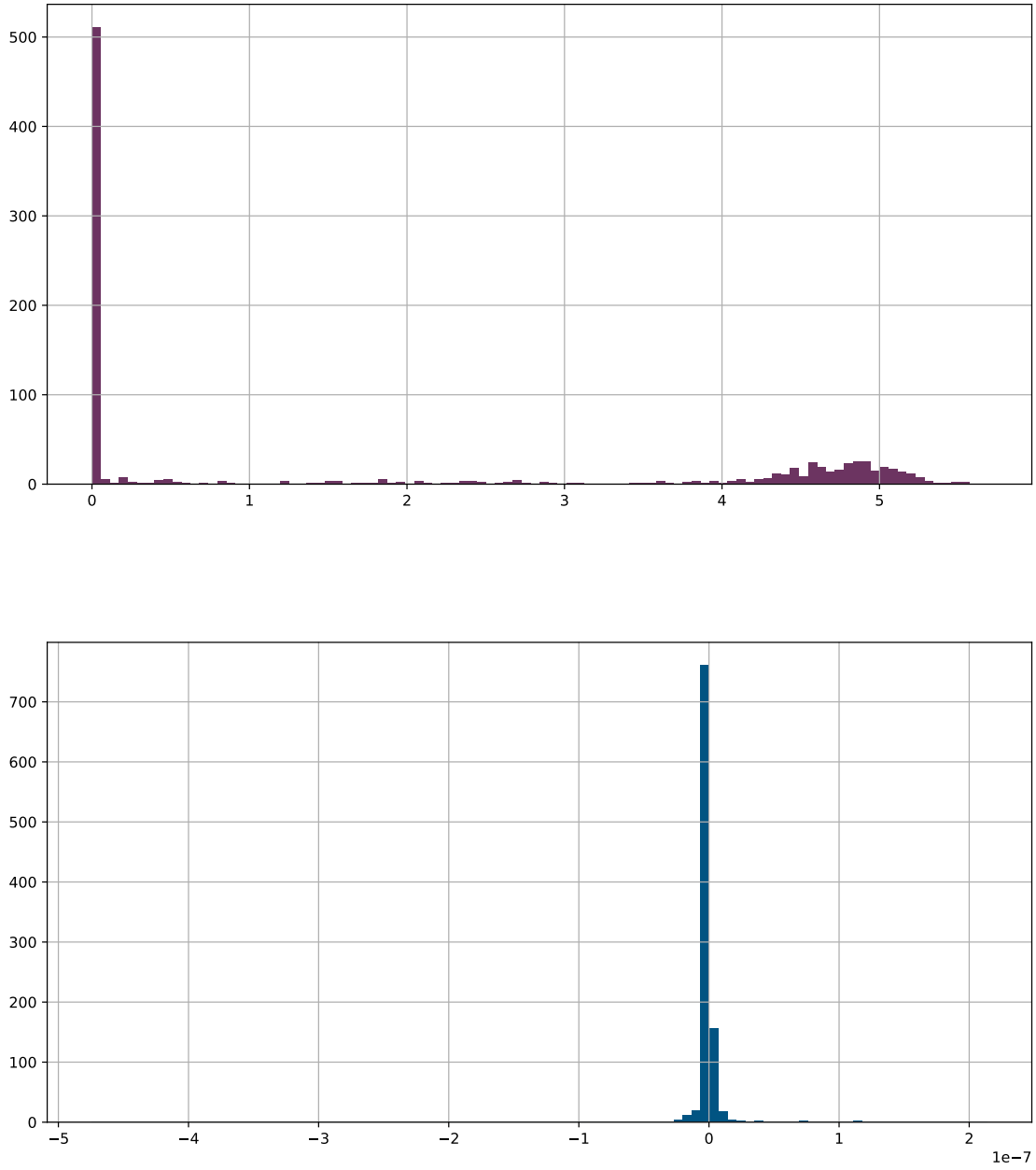

Рис. 1: Распределение максимума поэлементной разницы для тысячи случайных матриц 10x10 и средней разницы для тысячи случайных матриц 6x8

4.3 Выбор длины шага

На каждой итерации алгоритма выбирается длина шага в направлении антиградиента. Для этого могут использоваться разные стратегии – например, константный шаг ($\tau = const$) или заранее заданная последовательность ($\tau = \frac{1}{\sqrt{k+1}}$, где k - это номер итерации). В данном случае был использован принцип экстраполяции и правило Армихо.

- **Экстраполяция** — стратегия удволения τ_k, начиная с $\tau = 1$ с последовательным сокращением после локализации одномерной стационарной точки. При этом необходимо поддерживать мотононность функционала [4].

  Эта процедура пытается сделать длинный шаг экстраполяции. Данная стратегия очень простая, в дальнейшем предполагается попробовать более тяжеловесные алгоритмы, однако легкость данной версии позволяет использовать её в других стратегиях выбора длины шага (например, в правиле Армихо) на каждой итерации, которые, однако, могут делать более короткие шаги. Выбор оптимального шага осуществляется по алгоритму 3.

Algorithm 3 Метод экстраполяции для выбора оптимального шага

$\tau_0 = 0$, $\tau_1 = 1$
while $\psi'(\tau_k) < 0$ and $\psi(\tau_k) \leq \psi(\tau_{k+1})$ **do**
  $\tau_{k+1} = 2\tau_k$

- **Армихо** — стратегия увеличения или уменьшения τ (начиная с $\tau_0 = 1$) в зависимости от знака $\psi_k(1)$. Данное правило, возможно, в конечном итоге дает меньший шаг [4].

  Зафиксируем $0 < \alpha < \beta < 1$ (например, $\alpha = \frac{1}{3}$, $\beta = \frac{2}{3}$). Тогда нужно найти такое $\hat{\tau}$, что выполнено

$$\psi(0) + \beta\psi'(0)\hat{\tau} \leq \psi(\hat{\tau}) \leq \psi(0) + \alpha\psi'(0)\hat{\tau} \tag{1}$$

  При условии, что $\psi'(0) \neq 0$, это неравенство влечет за собой условие $\psi'(0)\hat{\tau} < 0$. Таким образом обеспечивается уменьшение значения функции.

  Правило Армихо состоит из двух этапов — *локализации* и *спецификации*.

  Локализация. Цель данной стадии — найти такие τ_1 и τ_2, что выполнено

$$\psi(0) + \beta\psi'(0)\tau_1 > \psi(\tau_1) \tag{2}$$

$$\psi(0) + \alpha\psi'(0)\tau_2 < \psi(\tau_2) \tag{3}$$

  Стартуем с $\tau = 1$, если (1) выполнено, остановка (нужное значение $\hat{\tau}$ уже найдено). Иначе выполнено одно из условий — (2) или (3).

  Если выполнено первое (2), то пробуем зафиксировать $\tau_1 = 1$ и $\tau_2 = 2\tau_1$, если оказалось верным изначальное условие (1), остановка. Иначе продолжаем поиск.

  Если выполнено второе (3), то пробуем зафиксировать $\tau_2 = 1$ и $\tau_1 = \frac{1}{2}\tau_2$. Если (1) выполнено, то остановимся. Иначе продолжим.

Псевдокод данной процедуре описан в алгоритме [4].

Algorithm 4 Стадия локализации в правиле Армихо

function CONDITION1(τ)
  return $\psi(0) + \beta\psi'(0)\tau \leq \psi(\tau) \leq \psi(0) + \alpha\psi'(0)\tau$
function CONDITION2.1(τ_1)
  return $\psi(0) + \beta\psi'(0)\tau_1 > \psi(\tau_1)$
function CONDITION2.2(τ_2)
  return $\psi(0) + \alpha\psi'(0)\tau_2 < \psi(\tau_2)$

$\alpha = 1/3$, $\beta = 2/3$
$\tau = 1$
if CONDITION1(τ) **then**
  return τ
if condition1.2 **then**τ
  $\tau_1 = 1$, $\tau_2 = 2\tau_1$
else
  $\tau_2 = 1$, $\tau_1 = \frac{1}{2}\tau_2$

while NOT (CONDITION2.1(τ_1) AND CONDITION2.2(τ_2)) **do**
  if CONDITION2.1(τ_1) **then**
    $\tau_1 = \tau_2$
    $\tau_2 = 2\tau_1$
  else
    $\tau_2 = \tau_1$
    $\tau_1 = \frac{1}{2}\tau_2$

Спецификация. Если на предыдущей стадии не было найдено подходящее $\hat{\tau} = 1$, то была получена пара чисел τ_1 и τ_2, удовлетворяющих условиям (2) и (3). Тогда найдем бинпоиском такое $\hat{\tau} \in [\tau_1, \tau_2]$, что выполнено (1) с помощью алгоритма [5], при этом важно на каждом шаге сохранять условия (2) и (3) верными.

Algorithm 5 Стадия спецификации в правиле Армихо

while True **do**
  $\tau = \frac{\tau_1+\tau_2}{2}$
  if CONDITION1(τ) **then**
    return $\hat{\tau} = \tau$
  if CONDITION2.1(τ) **then**
    $\tau_1 = \tau$
  else ▷ CONDITION2.2(τ) is True
    $\tau_2 = \tau$

Понятно, что поиск $\hat{\tau}$ по данному правилу занимает логарифмическое время.

5 Анализ работы алгоритма

Были проведены следующие эксперименты:

1. Полученный код был запущен на функции $f : \mathbb{R} \to \mathbb{R}$, $f(x) = x^2 - x$ из $x_0 = 20000$ для нахождения точки минимума модуля, т.е. оптимальными точками в данном случае являются 1 и 0. По итогам работы был построен график 2 зависимости значения найденной точки x_k функции от номера итерации k.

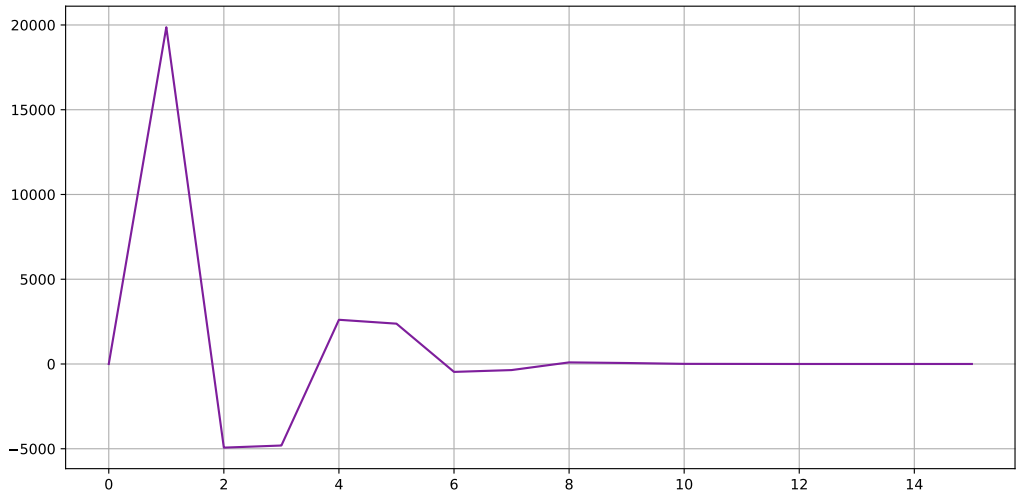

Рис. 2: Значение координаты x для функции $x^2 - x$ в зависимости от итерации алгоритма Гаусса-Ньютона из начальной точки $x_0 = 20000$ с использованием метода экстраполяции

2. Функция $f : \mathbb{R}^2 \to \mathbb{R}$, $f(x, y) = (x^2 - 2x + 1) + (y^2 + 5y - 6) + 10 \Rightarrow$ оптимальными являются точки, в которых $(x^2 - 2x + 1) + (y^2 + 5y - 6) = -10$. График 3 показывает зависимость функции от номера итерации.

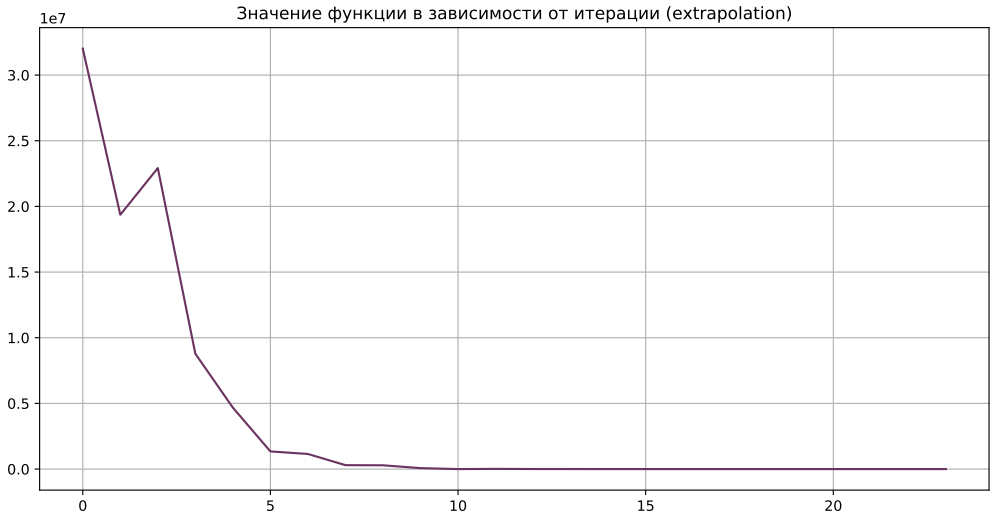

Рис. 3: Значение функции $(x^2 - 2x + 1) + (y^2 + 5y - 6) + 10$ в зависимости от итерации алгоритма Гаусса-Ньютона из точки $(x_0, y_0) = (400, 400)$ с использованием метода экстраполяции

3. Функция $f : \mathbb{R}^2 \to \mathbb{R}$, $f(x, y) = x^4 - x + y^4 - 2y^2 + 1$ $\Rightarrow$ одной из оптимальных точек является $(0, -1)$, модуль функции достигает нуля, что видно по графику 4. При этом при достаточно больших начальных точках значение функции оказывается слишком большим из-за $x^4 + y^4$ (практически устремляется к бесконечности), и метод не сходится.

Рис. 4: Значение функции $x^4 - x + y^4 - 2y^2 + 1$ в зависимости от итерации алгоритма Гаусса-Ньютона из начальной точки $(x_0, y_0) = (5, 5)$ с использованием метода экстраполяции

4. Также было показано, что для $f(x) = x^2 - x$ метод экстраполяции работает для любых значений L_0, что видно по графику 5. При этом правило Армихо работает при некоторых заданных значениях L_0 и по какой-то причине падает из-за ошибки в стадии локализации — в цикле получаются значения τ_1 и τ_2, не удовлетворяющие условиям (2) и (3).

Рис. 5: Результат работы метода в зависимости от начального параметра L_0

5. Для $f(x) = x^2 - x$, $L_0 = 5$ и $x_0 = 20000$ данный “гибкий” метод с выбором длина шага по правилу Армихо сходится быстрее (22 итерации вместо 27), чем метод с использованием экстраполяции при тех же входных данных 6.

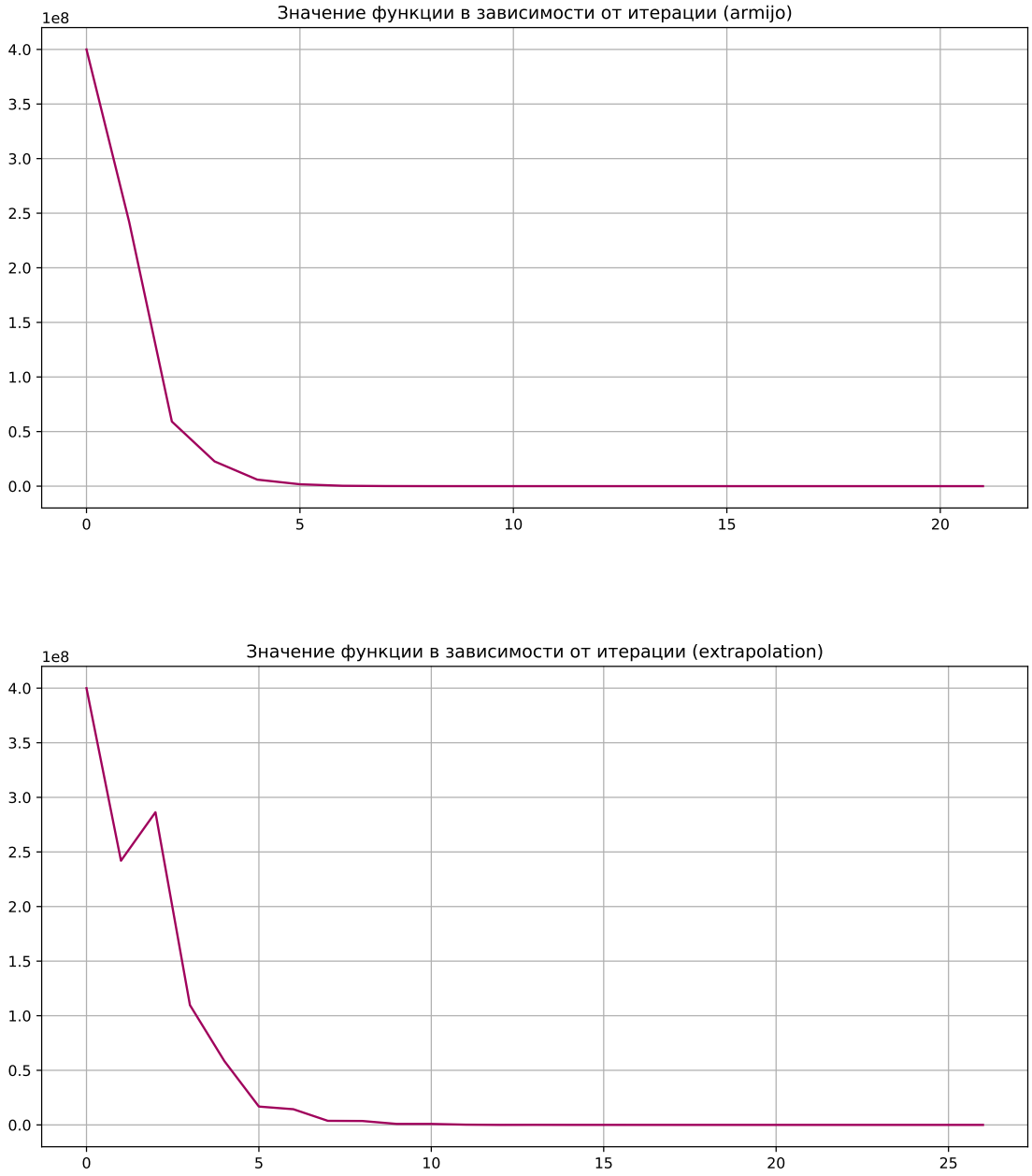

Рис. 6: Сравнение скорости сходимости правила Армихо и экстраполяции на примере функции $x^2 - x$ для $x_0 = 20000$, $L_0 = 5$

6 Заключение

В процессе работы над проектом было изучено множество материалов, по итогам которых была получена программная реализация “гибкой” модификации метода Гаусса-Ньютона с применением метода экстраполяции для выбора оптимальной длины шага. Полученная реализация была протестирована на различных функция (например, $x^2 - x$ и $x^4 - x + y^4 - 2y^2 + 1$), алгоритм сходится к оптимальной точке, задающей минимум модуля исследуемой функции за небольшое количество итераций. Однако на сходимость метода достаточно сильно влияет задание начальной точки.

Правило Армихо также было реализовано, однако результат работы данного метода сильно зависит от заданного параметра L_0. При достаточно больших значениях L_0 (> 25) метод перестает работать, тогда как при небольших значениях данного параметра метод показывает даже лучшую сходимость, чем метод с выбором длины шага с помощью экстраполяции.

Исходный код выложен по ссылке `https://github.com/kuskarov/hse-optimization-project/blob/master/implementation.ipynb`.

Список литературы

[1] Golub-Kahan-Lanczos bidiagonalization procedure. http://www.netlib.org/utk/people/JackDongarra/etemplates/node198.html.

[2] Метод Ньютона-Гаусса. http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%9D%D1%8C%D1%8E%D1%82%D0%BE%D0%BD%D0%B0-%D0%93%D0%B0%D1%83%D1%81%D1%81%D0%B0.

[3] C.M. Da Fonseca. On the Eigenvalues of Some Tridiagonal Matrices. 2005.

[4] Yurii Nesterov. Flexible Modification of Gauss-Newton Method. 2020.

[5] Бакушинский А. Б. К проблеме сходимости интеративно-регуляризованного метода Гаусса-Ньютон. *Журнал вычислительной математики имат*ематической *физики*, 1992.

[6] Колесников Е. В. SVD-разложение и его практические приложения. https://docplayer.ru/26987730-Svd-razlozhenie-i-ego-prakticheskie-prilozheniya.html.