МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ярославский государственный университет им. П.Г. Демидова»

Кафедра математического анализа

Курсовая работа

Применение алгоритмов условной оптимизации для вычисления минимальной нормы интерполяционного проектора

(Магистерская программа «Математическое моделирование и численные методы» 01.04.02 Прикладная математика и информатика)

Научный руководитель
к. ф-м. н.
_____________ А.Ю. Ухалов
« ___ » _____________ 2019 г.

Студент группы ПМИ-11МО
_____________ А.В. Лютенков
« ___ » ______________2019 г.

Ярославль, 2019 г.

Содержание

Введение

В данной работе рассматривается задача о построении минимального проектора (проектора, имеющего минимальную норму) при интерполяции непрерывной на шаре функции с помощью полиномов n переменных степени не выше единицы. Неравенство Лебега связывает норму проектора с величиной наилучшего приближения функции многочленами соответствующей степени. Этим, в частности, и обусловлен интерес к изучению минимальных проекторов и к получению оценок их норм. Также рассматривается вопрос условной оптимизации функции многих переменных и их программная реализация.

Также описывается реализованая в рамках данной работы компьютерная программа, которая решает задачу условной оптимизации функции многих переменных для нахождения верхних оценок минимальной нормы проектора.

1. Задача линейной интерполяции на n-мерном шаре

Положим B_n — n-мерный шар. $\Pi_1(\mathbb{R}^n)$ — совокупность многочленов n переменных степени $\leqslant 1$. Пусть S — невырожденный сиплекс в $\mathbb{R}^n$, вершины симплекса зададаются, как $x^{(j)} = (x_1^{(j)}, ..., x_n^{(j)}), j = 1, \ldots, n$. Рассмотрим матрицу A:

$$A := \begin{pmatrix} x_1^{(1)} & \ldots & x_n^{(1)} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_1^{(n+1)} & \ldots & x_n^{(n+1)} & 1 \end{pmatrix}. \tag{1.1}$$

Скажем, что набор точек $x^{(j)}$ — допустим для интерполяции многочленами из $\Pi_1(\mathbb{R}^n)$. Это условие эквивалентно тому, что матрица A является невырожденной.
$\Delta := det(A)$, определитель, который получается из Δ заменой j-й строки на строку $(x_1, \ldots, x_n, 1)$. Многочленый $\lambda_j(x) := \Delta_j(x)/\Delta$ из $\Pi_1(\mathbb{R}^n)$ называются базисными многочленами Лагранжа симплекса S и обладают свойством $\lambda_j(x^k) = \delta_j^k$, где δ_j^k — символ Кронакера. $\lambda_j = l_{1j}x_1 + \cdots + l_{nj}x_n + l_{n+1j}$, коэффициентны l_{ij} составляют столбцы матрицы

$$A^{-1} = \begin{pmatrix} \ldots & l_{1,j} & \ldots \\ \vdots & \vdots & \vdots \\ \ldots & l_{n,j} & \ldots \\ \ldots & l_{n+1,j} & \ldots \end{pmatrix}. \tag{1.2}$$

Так как $\lambda_j(x^k) = \delta_j^k$ любой многочлен $p \in \Pi_1(\mathbb{R}^n)$ удовлетворяет равенству

$$p(x) = \sum_{j=1}^{n+1} p(x^{(j)})\lambda_j(x). \tag{1.3}$$

Так как $det(A) \neq 0$, то для любой $f \in C(B_n)$, где $C(B_n)$ — совокупность $f : B_n \to \mathbb{R}$ найдется единственный многочлен $p \in \Pi_1(\mathbb{R}^n)$ удовлетворяющий условиям:

$$p(x^{(j)}) = f(x^{(j)}). \tag{1.4}$$

1.1. Интерполяционный проектор

Введем в рассмотрение оператор $P : C(B_n) \to \Pi_1(\mathbb{R}^n)$, который далее будем называть интеполяционным проектором. Интерполяционный проектор по системе узлов $x^{(j)}$ определяется с помощью равенств:

$$Pf(x^{(j)}) = f_j := f(x^{(j)}), j = 1, \ldots, n+1. \tag{1.1.1}$$

Из этих равенств следует, что данный оператор является линейным и справедлив следующий аналог интерполяционной формулы Лагранжа:

$$Pf(x^{(j)}) = p(x) = \sum_{j=1}^{n+1} f_j\lambda_j(x). \tag{1.1.2}$$

1.2. Норма интерполяционного проектора. Минимальная норма проектора

Обозначим $||P||$ норму оператора P. Эта величина зависит от от узлов $x^{(j)}$.

Лемма. *Для любого интерполяционного проектора $P : C(B_n) \to \Pi_1(\mathbb{R}^n)$ и симплекса S с вершинами в его узлах имеет место равенство*

$$||P|| = \max_{x \in B_n} \sum_{j=1}^{n+1} |\lambda_j(x)| \qquad (1.2.1)$$

Доказательство этого утверждения можно найти в монографии [1].

Обозначим через θ_n минимальну норму проектора, при условии, что все узлы принадлежат шару B_n:

$$\theta_n := \min_{x^{(j)} \in B_n} ||P|| \qquad (1.2.2)$$

Интерполяционный проектор P^* с нормой $||P^*|| = \theta_n$ назовем минимальным.

2. Компьютерная программа для расчета θ_n

В рамках данной работы была реализована компьютерна программа для численной минимизации функции многих переменных. Норма проектора вычисляется по формуле(1.2.1), зная это, зададим целевую функцию для минимизации.

$$F(A) = \max_{x \in B_n} \sum_{j=1}^{n+1} |\lambda_j(x)| \qquad (2.1.1)$$

Где A — марица, которая имеет вид (1.1).

Программа реализованна на языке Python с использованием библиотек SciPy и NumPy. Программа реализована в виде консольного приложения. Приложение зависит от выше указанных программных библиотек. Листинг кода программы приводится для условной оптимизации функции многих переменных указан в Приложении 1.

2.1. NumPy

NumPy — библиотека с открытым исходным кодом для языка программирования Python. Возможности:

поддержка многомерных массивов (включая матрицы); поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами. Математические алгоритмы, реализованные на интерпретируемых языках (например, Python), часто работают гораздо медленнее тех же алгоритмов, реализованных на компилируемых языках (например, Фортран, Си, Java). Библиотека NumPy предоставляет реализации вычислительных алгоритмов (в виде функций и операторов), оптимизированные для работы с многомерными массивами. В результате любой алгоритм, который может быть выражен в виде последовательности операций над массивами (матрицами) и реализованный с использованием NumPy, работает так же быстро, как эквивалентный код, выполняемый в MATLAB.

2.2. SciPy

SciPy — библиотека для языка программирования Python с открытым исходным кодом, предназначенная для выполнения научных и инженерных расчётов.

Возможности:

- поиск минимумов и максимумов функций;
- вычисление интегралов функций;
- поддержка специальных функций;
- обработка сигналов;
- обработка изображений;
- работа с генетическими алгоритмами;
- решение обыкновенных дифференциальных уравнений;
- и др.

2.3. SciPy, оптимизация с условиями

Общий интерфейс для решения задач как условной, так и безусловной оптимизации в пакете scipy.optimize предоставляется функцией [2]

```
minimize()
```

Однако известно, что универсального способа для решения всех задач не существует, поэтому выбор адекватного метода как всегда ложится на плечи исследователя. [2]

Подходящий алгоритм оптимизации задается с помощью аргумента функции [2]

```
minimize(..., method="")
```

Для условной оптимизации функции нескольких переменных доступны реализации следующих методов: [2]

- trust-constr
- SLSQP
- TNC
- L-BFGS-B
- COBYLA

В зависимости от выбранного метода, по-разному задаются условия и ограничения для решения задачи: [2]

- объектом класса Bounds, для методов L-BFGS-B, TNC, SLSQP, trust-constr
- списком (min, max), для этих же методов L-BFGS-B, TNC, SLSQP, trust-constr
- объектом или списком объектов LinearConstraint, NonlinearConstraint для методов COBYLA, SLSQP, trust-constr
- словарем или списком словарей
- COBYLA

Пример:

```
from scipy.optimize import minimize
from scipy.optimize import rosen, rosen_der, rosen_hess, rosen_hess_prod

x0 = np.array([0.5, 0])
res = minimize(rosen, x0, method='trust-constr', jac=rosen_der, hess=rosen_hess,
constraints=[linear_constraint, nonlinear_constraint],
options={'verbose': 1}, bounds=bounds)
print(res.x)
```

2.3.1. Алгоритм Бройдена-Флетчера-Голдфарба-Шанно (BFGS)

Для решения текущей задачи был выбран метод L-BFGS-B, который хорошо себя зарекомендавал на подобных задачах.

При численной оптимизации алгоритм Бройдена-Флетчера-Голдфарба-Шанно (BFGS) является итерационным методом решения неограниченных задач нелинейной оптимизации [3].

Метод BFGS относится к квази-ньютоновским методам, классу методов оптимизации восходящего подъема, которые ищут стационарную точку (предпочтительно дважды непрерывно дифференцируемой) функции. Для таких задач необходимым условием оптимальности является то, что градиент равен нулю. Метод Ньютона и методы BFGS не гарантируют сходимости, если функция не имеет квадратичного разложения Тейлора вблизи оптимума. Тем не менее, BFGS доказал свою хорошую производительность даже для негладкой оптимизации [4].

В квази-ньютоновских методах матрицу гессиана вторых производных не нужно оценивать напрямую. Вместо этого матрица гессиана аппроксимируется с использованием обновлений, определяемых оценками градиента (или приблизительными оценками градиента). Квази-ньютоновские методы являются обобщениями метода секущих для нахождения корня первой производной для многомерных задач. В многомерных задачах уравнение секущей не определяет уникальное решение, а квази-ньютоновские методы отличаются тем, как они ограничивают решение. Метод BFGS является одним из самых популярных членов этого класса [5]. Также широко используется L-BFGS, который представляет собой версию BFGS с ограниченной памятью, которая особенно подходит для задач с очень большим количеством переменных (например,$>$ 1000). Вариант BFGS-B обрабатывает простые ограничения например шаром.

Алгоритм назван в честь Чарльза Джорджа Бройдена, Роджера Флетчера, Дональда Голдфарба и Дэвида Шанно.

Схема алгоритма:

дано ϵ, x_0
инициализировать C_0
$k = 0$
while $||\nabla f_k|| > \epsilon$
найти направление $p_k = -C_k \nabla f_k$
вычислить $x_{k+1} = x_k + \alpha_k p_k \alpha_k$
обозначить $s_k = x_{k+1} - x_k y_k = \nabla f_{k+1} - \nabla f_k$
вычислить C_{k+1}
$k = k + 1$
end

3. Заключение

Были опробованы методы условной оптимизации помтавляемые библиотекой SciPy. Написаны тестовые программы для вычисления минимальной нормы интерполяционного проектора на n-мерном шаре. Получены численные оценки для некоторых n. Определены методы для дальнейшей разработки программы для получения оценок минимальной нормы проектора, с помощью предложенных методов.

Список литературы

[1] Невский М. В. Геометрические оценки в полиномиальной интерполяции. Ярославль: ЯрГУ, 2012.

[2] https://www.pvsm.ru/python/314870

[3] Fletcher, Roger (1987), Practical methods of optimization (2nd ed.), New York: John Wiley & Sons, ISBN 978-0-471-91547-8

[4] Lewis, Adrian S.; Overton, Michael (2009), Nonsmooth optimization via BFGS

[5] Nocedal, Jorge; Wright, Stephen J. (2006), Numerical Optimization (2nd ed.), Berlin, New York: Springer-Verlag, ISBN 978-0-387-30303-1

Приложение 1

Листинг программы

```
import scipy as scp
import numpy as np
import random as rnd
from scipy.optimize import minimize
from scipy.optimize import rosen, rosen_der, rosen_hess, rosen_hess_prod
from scipy.optimize import Bounds
from scipy.optimize import SR1
from scipy.optimize import BFGS
from scipy.optimize import LinearConstraint
from scipy.optimize import NonlinearConstraint

x0 = np.array([])
dim = 5
context_vect = []

rnd_max = 10000

class settings:
        def __init__(self):
                pass

        def setMatrinx(self, matrix):
                self.context_matrix = matrix

def cons_f(x):
        return sum([i**2 for i in x])

nonlinear_constraint = NonlinearConstraint (cons_f, 0, 1, jac = '2-point', hess

def getRandomVect(vec_len):
        rnd_vec = rnd.sample(range(rnd_max), vec_len)
        rnd_vec = [float(i)/rnd_max for i in rnd_vec]
        return np.array(rnd_vec)

def getMatrixByVect(x):
        vect = np.array([1 for i in range(dim + 1)])
        x = np.concatenate((x,vect), axis = None)
        matrix = np.array(x).reshape(dim+1, dim+1)
        matrix = matrix.transpose()
        return matrix

def getVectByMatrix(matrix):
        return np.asarray([matrix[i,0:dim-1] for i in range(dim-1)])
```

```
def getNorm(x, params):
        sum = 0.0
        for i in range(dim):
                cur_sum = 0.0
                for j in range(dim):
                        cur_sum += params[0][i][j] * x[j]
                sum += abs(cur_sum)
        return -sum

def func(x):
        matrix = np.linalg.inv(getMatrixByVect(x))

        x0_sphere = getRandomVect( dim*(dim+1) )
        max_t = minimize(getNorm, x0_sphere, [matrix],method='L-BFGS-B',  jac="2
        constraints=[nonlinear_constraint], options={'eps': 1e-10})
        res = max_t.x
        return -getNorm(res, [matrix])


x0 = getRandomVect(dim*(dim+1))

res = minimize(func, x0, method='trust-constr',  jac="2-point", hess=SR1(),
        constraints=[nonlinear_constraint],
        options={'verbose': 1})
        print(res)

```