Федеральное агентство по образованию
Государственное образовательное учреждение
высшего профессионального образования
«Нижегородский государственный университет
им. Н.И. Лобачевского»

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных
технологий математики и механики

Отчет по лабораторной работе

Многошаговая схема решения двумерных задач глобальной оптимизации

Выполнил:
студент группы 381506-2
Антонова Л. А.

Проверил:
ассистент каф. МОСТ,
ИИТММ
Козинов Е. А.

Нижний Новгород
2018 г.

Оглавление

Постановка задачи

Задача многомерной многоэкстремальной оптимизации может быть определена как проблема поиска наименьшего значения действительной функции $\varphi(y)$

$$\varphi(y^*) = min\{\varphi(y) : y \epsilon D\},$$
$$D = \{y \in R^N : a_i \le y_i \le b_i, 1 \le i \le N\},$$

где $a, b \in R^N$ есть заданные векторы.

Численное решение задачи сводится к построению оценки $y_k \in D$, отвечающей некоторому понятию близости к точке y^* (например, $||y^* - y_k^*|| \le \varepsilon$, где $\varepsilon > 0$ есть заданная точность) на основе конечного числа k вычислений значений оптимизируемой функции. Относительно класса рассматриваемых задач предполагается выполнение двух важных условий.

Во-первых, предполагается, что оптимизируемая функция $\varphi(y)$ может быть задана не аналитически, а некоторым алгоритмом вычисления ее значений в точках области D; при этом *испытание* (вычисление одного значения) является вычислительно-трудоемкой операцией.

Во-вторых, будем предполагать, что $\varphi(y)$ удовлетворяет условию Липшица

$$|\varphi(y_1) - \varphi(y_2)| \le L\|y_1 - y_2\|, \ \ y_1, y_2 \epsilon D \ , 0 < L < \infty$$

что соответствует ограниченности изменения значений функции при ограниченной вариации аргумента. Это предположение можно интерпретировать (применительно к прикладным задачам) как отражение ограниченности мощностей, порождающих изменения в моделируемой системе.

Задачи многоэкстремальной оптимизации имеют существенно более высокую трудоемкость решения по сравнению с другими типами оптимизационных задач, т.к. глобальный оптимум является интегральной характеристикой решаемой задачи и требует исследования всей области поиска. Как результат, поиск глобального оптимума сводится к построению некоторого покрытия (сетки) в области параметров, и выборе наилучшего значения функции на данной сетке.

В этой работе будет рассмотрена базовая многошаговая схема решения двумерных задач глобальной оптимизации.

Метод решения

Рассмотрим в качестве поисковой информации множество

$$\omega = \omega_k = \{(x_i, z_i), 1 \le i \le k\}$$

Согласно алгоритму, два первых испытания проводятся на концах отрезка $[a, b]$, т.е. $x^1 = a,\ x^2 = b$, вычисляются значения функции $z^1 = \varphi(a),\ z^2 = \varphi(b)$, и количество k проведенных испытаний полагается равным 2.

Пусть проведено $k \ge 2$ испытаний и имеется информация о текущем ω_k. Для выбора точки x^{k+1} нового испытания необходимо выполнить следующие действия.

1. Пронумеровать нижним индексом (начиная с нулевого значения) точки $x^i, 1 \le i \le k$, из множества в порядке возрастания, т.е.

$$a = x_0 < x_1 < \dots < x_k = b$$

2. Полагая $z_i = \varphi(x_i),\ \ 1 \le i \le k$ вычислить величину

$$M = \max_{1 \le i \le k} \left| \frac{z_i - z_{i-1}}{x_i - x_{i-1}} \right|$$

и положить

$$m = \begin{cases} rm, & M > 0 \\ 1, & M = 0 \end{cases}$$

где $r > 1$ является заданным параметром метода.

3. Для каждого интервала $(x_i - x_{i-1}),\ 1 \le i \le k-1$ вычислить характеристику

$$R(i) = m(x_i - x_{i-1}) + \frac{(z_i - z_{i-1})^2}{m(x_i - x_{i-1})} - 2(z_i + z_{i-1})$$

4. Найти интервал $(x_t - x_{t-1})$, которому соответствует максимальная характеристика

$$R(t) = \max\{R(i) : 1 \le i \le k-1\}$$

5. Провести новое испытание в точке

$$x^{k+1} = \frac{1}{2}(x_t + x_{t-1}) - \frac{(z_t - z_{t-1})}{2m}$$

Вычислить значение $z^{k+1} = \varphi(x^{k+1})$ и увеличить номершага поиска на единицу $k = k + 1$.

Операция пунктов 1-5 описывают решающее правило алгоритма глобального поиска.

Правило остановки задается в форме

$$H_k(\Phi, \omega_k) = \begin{cases} 0, & x_t - x_{t-1} \le \varepsilon \\ 1 & x_t - x_{t-1} > \varepsilon \end{cases}$$

где $\varepsilon > 0$ — заданная точность поиска (по координате).

Наконец, в качестве оценки экстремума выбирается пара

$$e^k = (\varphi_k^*, x_k^*),$$

где φ_k^* − минимальное вычисленное значение функции, т.е.

$$\varphi_k^* = \min_{1 \le i \le k} \varphi(x^i),$$

а x_k^* − координата этого значения:

$$x_k^* = \arg\min_{1 \le i \le k} \varphi(x^i)$$

Для того чтобы обобщить данный алгоритм на двумерный случай: будем выполнять оптимизацию по одной переменной, например по переменной x. Проблема с которой мы столкнемся это невозможность вычисления значения функции $z_i = \varphi(x_i\,, y_i)$, так как множество x_i нам известно, а y_i нет. Для того, что его вычислить зафиксируем x_i и при фиксированном x_i выполним оптимизацию по y. Алгоритм вернет нам некоторую точку минимума (y^*, z^*), тогда $y_i = y^*$, $z_i = z^*$. Все остальные шаги алгоритма выполняются аналогично одномерному случаю.

Схема распараллеливания

В качестве схемы распараллеливания необходимо было осуществить разделение области поиска. В данной работе выполнено разделение области на полосы:

1. Выберем область $Q: \{a \le x \le b, g \le y \le k\}$.
2. Будем разделять на полосы по координате x:
   Пусть у нас имеется ProcNum число процессов, каждый из которых имеет уникальный ранг ProcRank. Число потоков и ранг каждого потока определяются с помощью функций omp_get_num_threads() и omp_get_thread_num() соответственно.
   Тогда деление области поиска на процессы можно представить следующим образом:

$$d_1 = a + \frac{b-a}{ProcNum} * ProcRank - \text{левая граница полосы}$$

$$d_2 = a + \frac{b-a}{ProcNum} * (ProcRank + 1) - \text{правая граница полосы}$$

$$a \le d_i \le b$$

d_1 и d_2 у каждого процесса имеют свои значения.

Внутри каждого процесса задача подразделяется на потоки, каждому потоку достается полоса:

$$N = \text{omp\_get\_num\_threads()}$$

$$R = \text{omp\_get\_thread\_num()}$$

$$s_1 = a + \frac{b-a}{\mathrm{N}} * R - \text{левая граница полосы}$$

$$s_2 = a + \frac{b-a}{\mathrm{N}} * (R + 1) - \text{правая граница полосы}$$

Опять же, s_1 и s_2 у каждого потока имеют свои значения.

Таким образом, разделение области поиска выглядит так, как показано

Внутри каждой полосы выполняется последовательный многошаговый алгоритм глобального поиска, после чего формируется результат на каждом процессе, а затем путем редукции общий результат (найденный глобальный минимум) остается на нулевом процессе (ProcRank=0).

Описание программной реализации

Последовательная версия

struct Point – описывает значение координат точки (x,y) и значение функции f(x,y)=z в этой точке.

Основные функции:

GlobalMinCalculation(TPostfix func, double a, double b) – на вход подается функция в постфиксной форме записи, и границы области в которой ищется глобальный минимум

1) Инициализация начальных значений

```
points[0].x = a;
points[1].x = b;
points[0].y = CalculateYMin(func, a, b, a);
points[1].y = CalculateYMin(func, a, b, b);
points[0].z = func.Calculate(a, points[0].y);
points[1].z = func.Calculate(b, points[1].y);
```

2) Пока не получим больше 1000 точек или удовлетворительную точность

- Вычисляем M для всех имеющихся на данном шаге промежутков

```
M = CalculateMBigMax(points, k, true);
```

- Для вычисленного M вычисляем m

```
m = CalculateMSmall(M);
```

- Для каждого промежутка вычисляем вероятность нахождения минимума

```
Ri = CalculateRs(m, points, k, true);
```

- Ищем номер промежутка с наибольшей вероятностью (первый в случае нескольких наибольших)

```
t = FindIntNumber(Ri);
```

- Вычисление условия остановки

```
stop_flag = points[t].x - points[t - 1].x;
```

- Генерируем следующее значение координаты и добавляем точку с такой координатой и вычисленным значением целевой функции в вектор точек с его последующей пересортировкой

temp = InsertXNext(func,points, t, m, a, b);

points = temp;

- Количество точек в векторе увеличивается на 1

k++;

2) Пересчитываем номер нужного нам промежутка для последней сработавшей итерации

Вычисляем M для всех имеющихся на данном шаге промежутков

M = CalculateMBigMax(points, k, true);

Для вычисленного M вычисляем m

m = CalculateMSmall(M);

Для каждого промежутка вычисляем вероятность нахождения минимума

Ri = CalculateRs(m, points, k, true);

Ищем номер промежутка с наибольшей вероятностью (первый в случае нескольких наибольших)

t = FindIntNumber(Ri);

возвращаем один из концов интервала(точку) с наибольшей вероятностью нахождения в нем минимума, полученный на последнем шаге перед остановкой цикла

return points[t];

InsertXNext(TPostfix func, vector<Point> p, int t, double m, double a, double b) - Вычисляет следующее значение координаты с вычислением оптимального y и вставляет новую точку в вектор с учетом пересортировки

```
Point new_point;
new_point.x = (p[t - 1].x + p[t].x) / 2 - (p[t].z - p[t - 1].z) / (2 * m);
new_point.y = CalculateYMin(func,a, b, new_point.x);
new_point.z = func.Calculate(new_point.x, new_point.y);
vector<Point> temp = InsertSort(new_point, p, true);
return temp;
```

double CalculateYMin(TPostfix func, double a, double b, double _x)- Вычисляет координату y, соотв. минимальному значению функции при фиксированном x, алгоритм работы аналогичен **GlobalMinCalculation()**

Генератор тестов

На вход генератору тестов подается номер теста, который будет сгенерирован.

1) f(x,y)=4*x^2+8*y^3+16*x Границы поиска: [-15,15]
2) f(x,y)= x^3-y^6+2*x*y-10 Границы поиска: [-20,20]
3) f(x,y)= cos(4*y-1)*x Границы поиска: [-5,5]
4) f(x,y)= cos(0.1*y)+12*x-x^2+6 Границы поиска: [-5,5]
5) f(x,y)= 4*x^2+8*y^3+16*x Границы поиска: [-10,10]
6) f(x,y)= (x+2*y-7)^2+(2*x+y-5)^2 Границы поиска: [-1000,1000]
7) f(x,y)= six(x+y)+(x-y)^2-1.5*x+2.5*y+1 Границы поиска: [-1000,1000]
8) f(x,y)= 100*(y-x^2)^2+(1-x^2) Границы поиска: [-1000,1000]
9) f(x,y)= 100*(y-x^3)^2-(1-x)^2 Границы поиска: [-1000,1000]
10) f(x,y)= 2*x^2-1.05*x^4+(x^6)/6+x*y+y^2 Границы поиска: [-1000,1000]

Для записи открывается бинарный файл «test_номер_теста.tst». Затем в него записывается функция и левая и правая граница области поиска.

Проверка результата

На вход для проверки подается две строки: путь к файлу с правильным ответом и путь к файлу с результатом работы программы. Из файла с результатами работы программы считывается res=f(x,y), где f(x,y) – минимум. Из файла с ответами считывается answ. Если абсолютное значение res-answ меньше 0.001 тест считается пройденным.

OpenMp

Для реализации параллельной версии с использованием OpenMp использовались те же функции, что и для последовательной версии. Параллельная часть программы выглядит следующим образом

#pragma omp parallel private(p,left_b, right_b,ProcRank,result)

shared(left_border,right_border,ProcNum)

- Для каждого потока локальными считаются переменные содержащие локальный результат, ранг процесса, левая и правая границы поиска для данного потока и функция. Глобальными объявляются левая и правая граница поиска, количество потоков.

```
p.PutInfix(function); p.ToPostfix();
ProcRank = omp_get_thread_num();
left_b = left_border + (right_border - left_border) / ProcNum*ProcRank;
right_b = left_border + (right_border - left_border) / ProcNum*(ProcRank + 1);
result = GlobalMinCalculation(p, left_b, right_b);
```

Вычисление границ поиска для каждого потока.

#pragma omp barrier -синхронизация всех потоков

#pragma omp single

global_res = result; - один из потоков записывает в переменную global_res результат вычислений на своём отрезке

#pragma omp critical –критическая секция в которой может находится только один поток

```
if (result.z < global_res.z)
    global_res = result;
```

- сравнивается локальный результат потока,попавшего в критическую секцию с текущим значением глобального минимума, если он меньше, то global_res=result.

TBB

Для реализации TBB версии реализован класс **myTbb**.

Основными являются функции:

operator()(const blocked_range<double>& r)

-для каждого из потоков выполняется последовательная версия глобального поиска минимума

```
int left_border = r.begin();

int right_border = r.end();
```

```
res = GlobalMinCalculation(func, left_border, right_border);
```

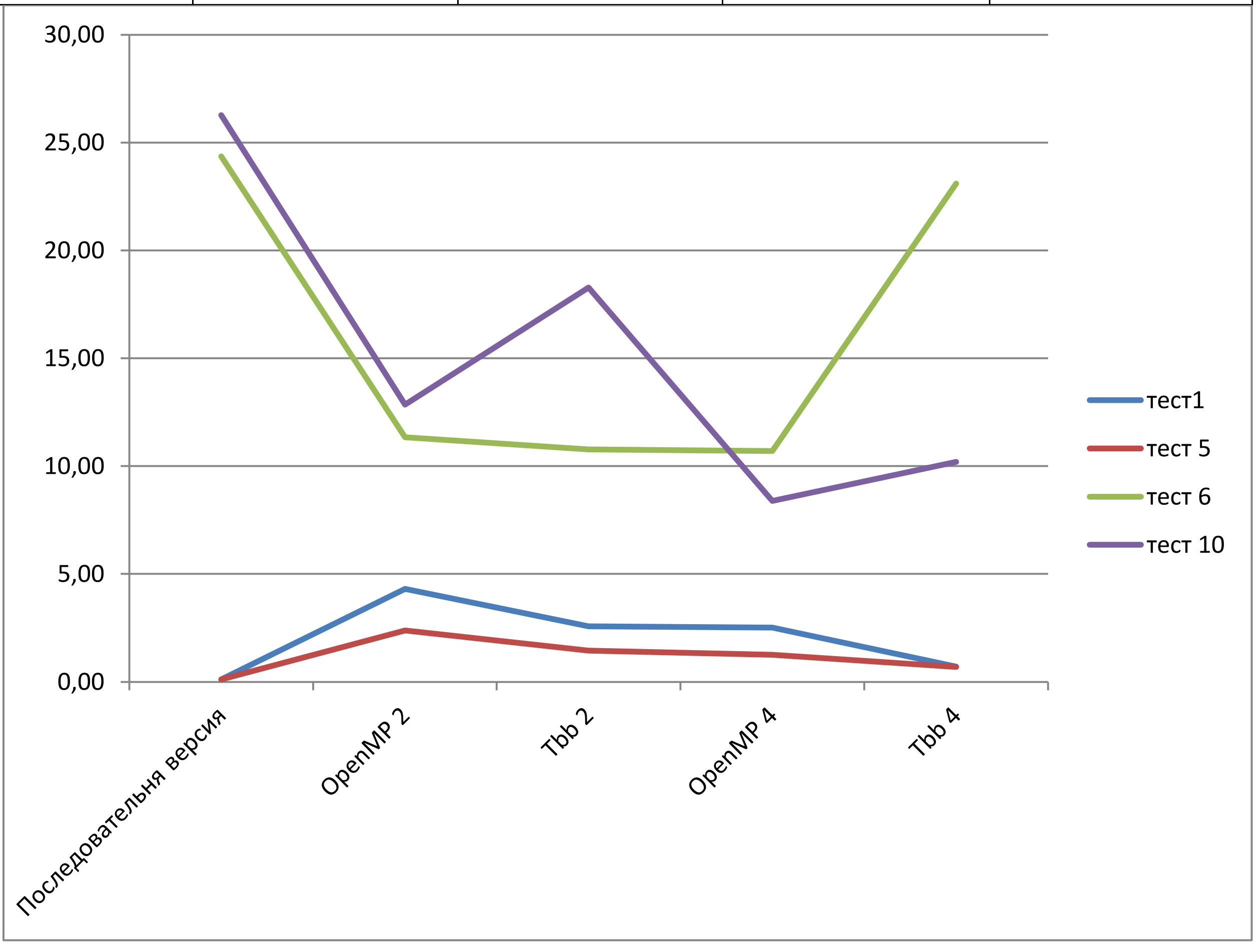
join(const myTbb& rhs) – редукция, ищется минимальный результат на каждом из потоков

```
if (rhs.res.z < res.z)
    res = rhs.res;
```

Результаты экспериментов по оценке масштабируемости.

Для оценки масштабируемости были выбраны тесты на которых наиболее заметна разница в производительности

№	Последовательная версия	OpenMp		TBB	
		2	4	2	4
1	0,11	4.31	2,51	2,58	0,70
5	0,11	2,39	1,26	1,44	0,69
6	24,36	11,34	10,70	10,78	23,10
10	26,27	12,85	8,39	18,28	10,19



Вывод

В ходе лабораторной работы была написана программа с использованием библиотек OpenMP и TBB, выполняющая поиск глобального минимума двумерной функции $\varphi(x,y)$в заданной области $Q\colon \{x_1 \le x \le x_2, y_1 \le y \le y_2\}$.

По итогам работы можно сделать вывод о значительном ускорении работы алгоритма при распараллеливании. Такое ускорение получается, поскольку внутри программы происходит разделение области поиска минимума заданной функции между процессами и потоками. Чем меньше ширина полосы, тем меньше итераций алгоритма требуется для решения задачи. Кроме того, алгоритм достаточно быстро сходится к минимальному значению функции, если это значение присутствует в исследуемой области. Снижение величины ускорения с увеличением числа процессов и потоков можно объяснить тем,что внутри полос, не содержащих минимум функции, требуется большое количество итераций алгоритма, а значит и увеличение времени работы программы, что приводит к замедлению роста величины ускорения.