# **2.2 Особенности последовательной реализации алгоритмов.**

Суть любого градиентного метода состоит в продвижении к точке экстремума по направлению градиента в случае поиска максимума или антиградиента - при поиске минимума.

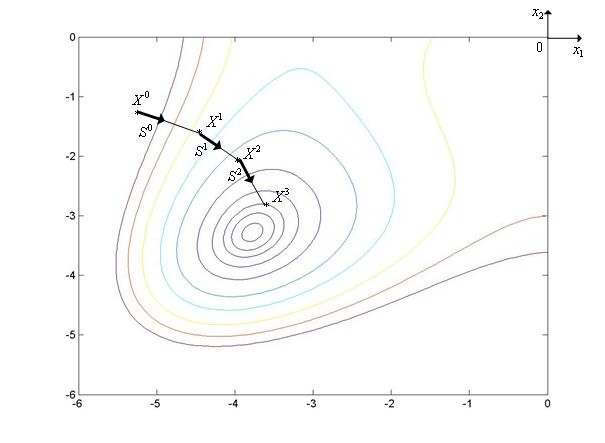


Рис.2.3 Градиентный спуск

Однако найденное значение необязательно будет являться глобальным минимумом или максимумом функции в заданной области. Зачастую градиентные методы приводят к некой стационарной точке или локальному экстремуму, что не удовлетворяет интересам решающего задачу человека.

Проблему нахождения глобального экстремума при наличии нескольких точек максимума/минимума можно обойти с помощью разбиения области поиска на части - наложения на неё сетки,- и вычисления экстремума градиентным спуском из каждого узла сетки.

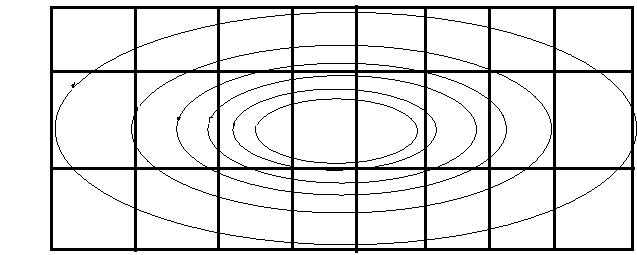


Рис.2.4. Наложение на область сетки

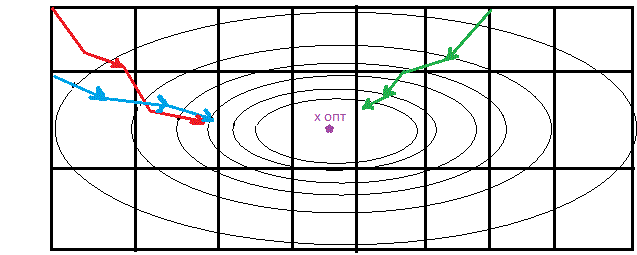


Рис.2.5 Спуск из каждого узла на примере трех градиентных спусков

Чем мельче берется сетка, тем выше вероятность попадания в глобальный экстремум. Однако стоит обратить внимание на то, что в этом случае поиск глобального экстремума обращается в трудоемкую задачу, требующую длительного времени для нахождения её решения, что не под силу человеку.

На помощь приходят вычислительные системы, позволяющие значительно снизить время обработки области поиска экстремума. Современные характеристики вычислительных систем растут с каждым годом, в частности это касается роста числа процессоров в компьютерах, что позволяет непросто свалить тягостные расчеты на машину, но и оптимизировать их за счет распараллеливания.

# **2.3 Параллельные алгоритмы нахождения экстремума градиентными методами**

При разработке параллельного алгоритма я основывалась на следующих фактах:

1) Вычисления в каждой части области осуществляются независимо от других расчетов..

2) Основным требованием является минимизация области поиска одним потоком.

Распараллеливание позволяет дополнительно разделить покрытую сеткой область на части согласно количеству используемых потоков (в проекте предусмотрено равномерное распределение).

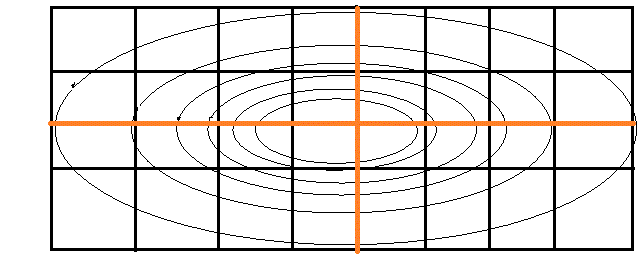


Рис.2.6. Разбиение на части согласно числу потоков n=4

Таким образом мы уменьшим количество начальных точек, рассматриваемых программой и минимизируем время за счет одновременной обработки области поиска. Каждый поток пройдет по доступному ему числу точек и выберет минимальное значение из всех найденных, после чего по завершению своей работы проведет сравнение с предполагаемым глобальным значением минимума и заменит его в случае наиболее лучшего результата целевой функции. Чтобы потоки одновременно не пытались изменить значение общей переменной , в программе используется синхронизация через lock.

lock (locker)

{

if (mink <= min)

{

min = mink; for (int j = 0; j < n; j++) otv[j] = ask[sd][j];

}

Общая блок- схема параллельных алгоритмов имеет вид:

Прием данных 1-ым потоком

Поиск решения на 1-ом потоке

Сравнение полученного значения с предполагаемым минимумом в крит. секции

Прием данных 2-ым потоком

Прием данных n-ым потоком

Вывод результата

Поиск решения на 2-ом потоке

Сравнение полученного значения с предполагаемым минимумом в крит. секции

Поиск решения на n-ом потоке

Сравнение полученного значения с предполагаемым минимумом в крит. секции

Ввод исходных данных

Разбиение области на части и определение начальных точек

Создание потоков и передача им данных

**.............**

Рис.2.7 . Блок-схема работы параллельного алгоритма