Криворожский государственный педагогический институт
Кафедра информатики и прикладной математики

А.П. Полищук, С.А. Семериков

ЧИСЛЕННЫЕ МЕТОДЫ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ МЕТОДОЛОГИИ

Учебное пособие

Раздел 6: Системы нелинейных уравнений и задачи оптимизации

Кривой Рог

1998

Полищук А.П., Семериков С.А. Численные методы в объектно-ориентированной методологии. Раздел «Системы нелинейных уравнений и задачи оптимизации». Учебное пособие. – Кривой Рог: КГПИ, 1998. – 29 с.

Авторы:

Полищук А.П.	к. т. н., с. н. с., доцент кафедры информатики и прикладной математики.
Семериков С.А.	магистр математики.

Рецензенты:

Рашевский Н.А.	к. ф.-м. н., доцент кафедры математики (КГПИ)
Теплицкий И.А.	учитель-методист, зам. директора по научной работе (Центрально-Городская гимназия)

Под общей редакцией доктора физико-математических наук, профессора В.Н. Соловьёва.

Рекомендовано к печати на заседании кафедры информатики КГПИ, протокол №2 от 17.09.98 г.

Підп. до друку 24.12.98
Друк №3. Друк офсетний
Умовн. фарбо-відб. 1,45
Тираж 300

Формат 80x84 1/16.
Умовн. друк. арк. 1,6
Зам. №12-1911

КДПІ, 324086, Кривий Ріг-86, пр. Гагаріна, 54

Криворізька міська друкарня
324050, Кривий Ріг-50, пр. Металургів, 28.

Оглавление

6. Численные методы решения систем нелинейных уравнений (СНУ) и поиска экстремумов функций

6.1. Общая постановка задачи

Мы объединили эти два класса задач в одном разделе потому, что они являются «сопряженными» – методы, используемые для вычисления корней, можно использовать для определения экстремумов и наоборот.

Система нелинейных уравнений с p неизвестными может быть записана в виде:

$$f_i(x_1, \ldots, x_p)=0,\ i=1, \ldots, p$$

в которых хотя бы одна функция f_i нелинейна.

С этой системой мы можем связать некоторую неотрицательную функцию $\Phi(x_1, \ldots, x_p)$ такую, что ее нулевой минимум является решением системы нелинейных уравнений, например:

$$\Phi(x_1, \ldots, x_p)=\sum_{i=1}^{p} f_i^2(x_1,...,x_p)$$

и решать задачу поиска корней методами поиска экстремума.

С другой стороны, при заданной для поиска минимума функции многих переменных можно, дифференцируя ее по каждой из переменных и приравнивая частные производные нулю, составить систему уравнений, решение которой даст координаты искомого минимума:

$$\frac{\partial \Phi}{\partial x_i}=0,\ i=1, \ldots, p$$

и выполнить определение координат минимума решением системы уравнений.

Оба класса задач распадаются на подклассы, выделяемые, например, по размерности вектора аргументов – одномерный или многомерный поиск (корней или экстремума), или по наличию или отсутствию существенных погрешностей в определении значения функций – поиск (корней или экстремума) в условиях помех или при их отсутствии, или по выполнению предварительной локализации искомого объекта – определение экстремума унимодальной или многоэкстремальной функции в заданном диапазоне (с одним или многими корнями в интервале неопределенности в случае вычисления корней).

Мы начнем рассмотрение методов с наиболее простых задач – для функций одной переменной при отсутствии помех в определении значений унимодальной функции в заданных точках.

6.2. Решение нелинейных уравнений

Функция одной переменной при отсутствии помех

Даже этот простейший случай связан с рядом проблем и первая из них – возможность наличия комплексных корней нелинейного или трансцендентного уравнения $f(x)=0$. В разделе, посвященном работе с полиномами, мы привели метод Ньютона для вычисления корней полинома (в том числе комплексных) и не будем к этому возвращаться. Вычисление корней для полиномиальных уравнений связано с большими удобствами – прежде всего, возможностью вычисления всех корней без предварительной процедуры их отделения (локализации корня в некотором диапазоне значений). Эта возможность обеспечивается процедурой последовательного понижения степени полинома делением его на полином первого порядка $(x–x^*)$, где x^* – значение уже вычисленного корня. Такое удаление из вычислительной процедуры уже вычисленных корней оказывается невыполнимым для других классов нелинейных функций и мы можем «натыкаться» на уже вычисленный корень многократно, если не выполним предварительного определения достаточно узких диапазонов размещения всех корней.

К сожалению, кроме табулирования функции и фиксации границ смены знаков или построения графиков функций современная теория ничего не предлагает. Отделение корней – это еще искусство, так как общее количество подлежащих локализации корней, скорее всего, неизвестно. Существенное упрощение задачи для полиномиальных уравнений подсказывает идею полиномиальной аппроксимации нелинейной функции с последующим вычислением корней, но мы не встречали в литературе подобных рекомендаций и не апробировали идею сами – предполагается, что неизбежные при аппроксимации погрешности не позволят получить достоверный результат.

В тех случаях, когда корни вещественны и осуществлено их предварительное отделение в конечном интервале значений (интервале неопределенности), используют

Методы последовательного сокращения интервала неопределенности

Все эти методы основаны на том или ином выборе точки (очередного кандидата на уточненное значение корня) внутри текущего интервала неопределенности и только этим методом выбора точки и отличаются друг от друга. После выбора точки определяется значение функции в этой точке и выясняется, на какой из границ интервала неопределенности значение функции совпадает по знаку со значением во внутренней точке – эта граница и переносится на место выбранной внутренней точки, приводя к уменьшению ширины интервала неопределенности. Процесс выбора внутренней точки продолжается до достижения шириной интервала заданной погрешности.

Трудности могут возникнуть для корней, которые образуются касанием функцией оси абсцисс без ее пересечения – этот вариант отслеживается либо параллельным контролем значений функции и фиксацией значения корня при входе значения функции в зону, близкую к машинному нулю, либо дополнительным поиском нулевого минимума квадрата функции в том же начальном интервале. Теперь об алгоритмах выбора внутренней точки. Помимо уже разобранного в разделе полиномов метода Ньютона, известны следующие:

Метод дихотомии (половинного деления или метод Больцано) предполагает выбор внутренней точки посредине текущего интервала неопределенности. Его эффективность в среднем трудно оспорить другими предложениями, по крайней мере, при известной начальной ширине интервала и заданной конечной ширине всегда известно, сколько всего потребуется итераций до завершения поиска – метод уменьшает длину интервала вдвое на каждом шаге.

Метод хорд – точка внутри интервала выбирается на пересечении с осью абсцисс линии, соединяющей два крайних (положительное и отрицательное) значения функции. Координаты этой точки вычисляются легко из простых геометрических соображе-

ний на подобных треугольниках. По существу, метод основан на линейной интерполяции функции по границам интервала неопределенности.

При желании вы можете использовать параболическую интерполяцию по трем точкам, взяв третью, например, тоже в середине интервала и вычислив координату точки пересечения параболы с осью абсцисс. Если значения аргумента на границах интервала c и d, то координата внутренней точки при линейной аппроксимации

$$x_{вн} = c - \frac{d-c}{f(d)-f(c)} f(c).$$

Итерации прекращаются, когда расстояние между соседними приближениями станет меньше заданной погрешности.

Метод случайного поиска – координаты точки внутри текущего интервала неопределенности генерируются с помощью генератора случайных чисел.

Рекуррентные методы уточнения текущей оценки значения корня

Метод секущих – модификация метода Ньютона, связанная с заменой производной функции отношением приращения функции к приращению аргумента. Для старта процесса уточнения необходимо задать два близких друг к другу приближения x_0 и x_1, а каждое новое приближение получим по формуле:

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k).$$

Скорость сходимости несколько ниже метода Ньютона, но не требуется вычисление производной.

Метод простых итераций может использоваться как для систем уравнений, так и для одиночного уравнения. Дадим для разнообразия его запись в векторной форме.

Если система уравнений $f(x)=0$ приведена к виду $x=\varphi(x)$ и выбрано начальное приближение x_0, то последующие приближения методом простой итерации находятся по формуле

$$x_{m+1} = \varphi(x_m),\ m = 0, 1, \ldots$$

Если последовательность векторов x_m сходится к вектору x^*, а функции $\varphi_i(x)$ непрерывны, то вектор x^* является приближенным решением системы.

Уточнение корня функции одной переменной в условиях помех

Если значения функции при заданных значениях аргумента определяются с погрешностями, которыми нельзя пренебречь и которые существенно искажают результаты поиска корня в заданном интервале, используют так называемые

Методы стохастической аппроксимации

В приложении к задаче поиска корня этот метод реализуется известной ***процедурой Роббинса-Монро:***

$$x_{n+1}=x_n-a_n z(x_n),$$

где x_n, x_{n+1} – значения аргумента на соответствующих индексам шагах, $z(x_n)$ – полученная с погрешностью оценка функции $f(x)$ на n-м шаге, a_n – некоторый член последовательности положительных чисел, удовлетворяющих *условиям сходимости Дворецкого*:

1. Предел членов последовательности сходится: $\lim a_n=0$ при $n\to\infty$.
2. Сумма членов последовательности расходится: $\sum_{n=1}^{\infty} a_n =\infty$.
3. Сумма квадратов членов – сходится: $\sum_{n=1}^{\infty} a_n^2 < \infty$.

Этим условиям удовлетворяет, например, гармоническая последовательность $1/n$: 1, 1/2, 1/3, …, обеспечивающая наиболее быстрое сокращение шага по сравнению с другими последовательностями вида n^{-p}.

Роббинс и Монро показали, что при выполнении этих требований их процедура сходится в среднеквадратическом смысле – математическое ожидание квадрата отклонения от истинного значения корня стремится к нулю: $\lim \{E[(x_n-x^*)^2]\}=0$ при $n\to\infty$.

Впоследствии Блюм показал, что при $n\to\infty$ последовательность x_n сходится к x^* с вероятностью единица: $p\{\lim x_n=x^*\}=1$.

6.3. Методы поиска экстремума функций

Унимодальные функции одного аргумента при отсутствии помех

Метод дихотомии

Нетрудно доказать, что если в нашем распоряжении только 2 опыта, то лучшее, что можно сделать для получения информации с целью максимального приближения к экстремуму – это провести эти 2 опыта в середине интервала по возможности ближе друг к другу. После переноса одной из границ в более удаленную от экстремума точку длина интервала будет больше половины первоначального как раз на расстояние между опытами. Следующие опыты можно осуществить снова в середине интервала и т. д. – эта простейшая и достаточно эффективная стратегия активного эксперимента называется *методом половинного деления (дихотомии)* – аналог метода Больцано в задаче поиска корня.

Метод Фибоначчи

Существует более совершенный метод, чем метод дихотомии – это предложенный в 1953 г. Кифером ε-минимаксный алгоритм, основанный на использовании чисел Фибоначчи для определения коэффициента деления текущего интервала неопределенности. Пусть в нашем распоряжении n экспериментов для исследования на экстремум исходного интервала единичной длины. Предположим, что все эксперименты, кроме одного, уже проведены и оставшийся интервал имеет длину L_{n-1}, а внутри его находится оставшийся опыт, давший лучшее приближение к экстремуму и поэтому не ставший новой границей интервала. Так как у нас остался единственный опыт в запасе, то не видно лучшего решения, как провести его симметрично уже имеющемуся относительно середины интервала. Чтобы эта симметрия соблюдалась и на более ранних опытах, длины интервалов должны удовлетворять условию $L_{j-1}=L_j+L_{j+1}$, что возможно, если первый опыт провести в 2-х точках, отстоящих от противоположных концов исходного интервала в доле его длины, равной F_{n-1}/F_n, где F_{n-1}, F_n – числа Фибоначчи соответствующих номеров. Одна из

внутренних точек станет новой его границей, а вторая окажется внутри интервала и следующий опыт необходимо провести просто симметрично этой оставшейся точке относительно середины нового интервала.

Метод золотого сечения

Начиная поиск экстремума, экспериментатор часто не знает, сколько опытов ему для этого понадобится – обычно он собирается проводить свои эксперименты до достижения поставленной цели, то есть пока интересующий его критерий не будет удовлетворен. Однако каждый хотел бы использовать быстро сходящийся метод, экономящий время и средства на экспериментирование. Как мы уже знаем, наиболее эффективным теоретически является метод Фибоначчи, но воспользоваться им можно, только заранее зная число испытаний, так как первый опыт рассчитывается по числам Фибоначчи, номера которых зависят от общего числа опытов. Существует метод, почти столь же эффективный, как и метод Фибоначчи и не зависящий от количества запланированных опытов, то есть со свободным завершением. По этому методу предлагается выдерживать постоянным отношение длин последовательных интервалов неопределенности $L_{j-1}/L_j=L_j/L_{j+1}=t$, откуда приходим к квадратному уравнению $t^2=t+1$, имеющему 1 положительный корень 1.618033989... – *золотое сечение*. По результатам 2-х экспериментов устанавливается, в каком из сегментов интервала опыты будут продолжены. В этом оставшемся интервале уже есть 1 предыдущий эксперимент и для продолжения достаточно провести симметричный ему опыт. После n испытаний приходим к интервалу неопределенности $L_n=1/t^{n-1}$. Метод предложен Кифером и Джонсоном и при большом числе опытов практически совпадает по эффективности с методом Фибоначчи.

Многомерный поиск экстремума

Метод координатного спуска

Алгоритм координатного спуска используется в многомерных задачах экспериментальной оптимизации и заключается в сведении многомерной задачи к последовательности одномерных, ре-

шаемых методами минимизации функции одной переменной, например золотого сечения.

Вначале в заданной области определения функции всем координатам, кроме одной, присваиваются фиксированные значения и целевая функция делается зависимой только от одной переменной. Далее ищется условный минимум функции, вариацией одной свободной переменной. Полученная координата условного минимума фиксируется в найденном значении и ищется условный минимум вариацией следующей переменной. После использования всех координат процесс снова продолжается с первоначальной переменной и т. д.

Если в области минимума функция цели достаточно гладкая, то процесс спуска по координатам будет линейно сходиться к минимуму. В сходящемся процессе расстояния между соседними точками однокоординатных минимумов будут стремиться к 0,что можно использовать для формулировки условия завершения итерационного процесса. Недостатком метода является чрезмерно большой объем экспериментов и опасность группирования опытов вокруг ложного экстремума при сложном рельефе исследуемой поверхности.

Метод градиента (наискорейшего спуска или крутого восхождения или метод Бокса-Уилсона)

Турист, желающий подняться на вершину холма, достигнет цели, если будет непрерывно подниматься вверх, а если он торопится, то ему придется двигаться по самым крутым направлениям.

Реализация этого интуитивно понятного метода носит название метода крутого восхождения или метода градиента. Помимо направления, в математической реализации метода необходимо иметь и алгоритм выбора шага поиска – при слишком большом шаге неизбежны потери на рыскание вокруг экстремума, а при слишком маленьком движение к цели будет медленным. Градиент функции $f(x_1, x_2, ..., x_n)$ в каждой точке направлен в сторону наибольшего локального возрастания этой функции.

$$\operatorname{grad} f(x)=\left(\frac{\partial f(x)}{\partial x_1},....,\frac{\partial f(x)}{\partial x_p}\right)^T.$$

Если выбрано начальное приближение $x=x_0$, то каждое очередное вычисляется так:

$$x_{m+1}=x_m-a_m\text{grad}\, f(x_m),$$

где значение a_m определяет величину шага и может быть вычислено как наименьший положительный корень скалярного уравнения

$$\frac{\partial}{\partial a}(x_m-a_m\text{grad}\, f(x_m))=0.$$

Для поиска максимума необходимо двигаться в направлении градиента, а для поиска минимума – в противоположном. Для определения направления движения (приращений текущих координат независимых переменных) придется вычислять производные целевой функции по каждой координате (составляющие градиента) и нормировать их по модулю вектора градиента, чтобы выделить в чистом виде направление (направляющие косинусы вектора градиента или, что то же самое, составляющие вектора единичной длины с тем же направлением, что и вектор градиента).

Теперь еще о величине шага. *При наличии погрешностей* в определении значения целевой функции шаг должен убывать по мере приближения к цели, т.е. его величина должна быть членом последовательности беззнаковых чисел, удовлетворяющей условию $\lim a_n=0$ при $n\to\infty$. Сумма членов этой последовательности должна расходиться $\sum_{n=1}^{\infty} a_n=\infty$, а сумма квадратов – сходиться $\sum_{n=1}^{\infty} a_n^2<\infty$. Мы уже рассматривали эти условия Дворецкого при изучении методов поиска корня в условиях помех. Кестен показал, что асимптотическая сходимость может быть заменена затухающим колебательным процессом, если длину шага уменьшать по гармоническому закону только после того, как соответствующая составляющая градиента поменяет знак.

Но шаг целесообразно делать переменным и в процессе поиска в отсутствии помех – вдали от экстремума можно двигаться с большим шагом, а по мере приближения к цели его следует уменьшать для повышения точности в определении экстремума.

Последовательный симплексный поиск (ПСМ) субоптимальной области в многомерном пространстве

Вводные замечания

Метод был предложен в 1962 году Спиндлеем, Херстом и Химсуорсом [см. А.П. Дамбраускас. Симплексный поиск. – М.: Энергия, 1979]. Оригинальность метода состоит в том, что движение к оптимуму в многомерном пространстве независимых переменных осуществляется последовательным отражением вершин симплекса. В k-мерном евклидовом пространстве *симплексом* называют фигуру, образованную k+1 точками (вершинами), не принадлежащими одновременно ни одному пространству меньшей размерности. В одномерном пространстве симплекс есть отрезок прямой, в двумерном – треугольник, в трёхмерном – тетраэдр и т. д. Симплекс называется *регулярным*, если расстояния между его вершинами равны. В ПСМ используются регулярные симплексы.

Из любого симплекса, отбросив одну его вершину, можно получить новый симплекс, если к оставшимся вершинам добавить всего одну точку. Это замечательное свойство и было использовано авторами метода при построении алгоритма движения симплекса в сторону искомой цели.

Для оценки направления движения во всех вершинах симплекса необходимо определить значения целевой функции Y_j. При поиске максимума наиболее целесообразным будет движение от вершины v_s с наименьшим значением Y_s к противоположной грани симплекса. Шаг поиска выполняется переходом из некоторого симплекса S_{n-1} в новый симплекс S_n путем исключения вершины v_s и построения ее зеркального отображения $v_{нз}$ относительно грани, общей обоим симплексам. Многократное отражение худших вершин приводит к шаговому движению центра симплекса к цели по траектории некоторой ломаной линии. Если не учитывать эксперименты в вершинах исходного симплекса, то на каждый шаг поиска требуется всего одно определение целевой функции.

Алгоритм последовательного симплексного метода

Исходные данные:

- количество независимых переменных k;
- предельные значения каждой i-й независимой переменной x_{imin}, x_{imax}. Эти предельные значения для реальных объектов иссле-

дования определяются априорными сведениями, условиями безопасности при проведении экспериментов и т. д.

- допустимая ошибка в определении координат оптимума ε;
- предполагается также возможность определения значения целевой функции $Y(x_{1j}, x_{2j}, ..., x_{ij}, ..., x_{kj})$ для каждой j-й вершины симплекса.

Подготовительные операции

1. Прежде всего необходимо определить стартовую точку для начала поисковых процедур; при отсутствии дополнительных априорных данных естественно расположить ее в центре области, ограниченной предельными значениями независимых переменных:

$$x_{1c}=(x_{1max}-x_{min})/2,\ x_{2c}=(x_{2max}-x_{2min})/2,\ ...,\ x_{kc}=(x_{kmax}-x_{kmin})/2.$$

Перенос в эту точку начала координат облегчит последующие вычислительные процедуры (достигается вычитанием).

2. Определяются координаты вершин начального симплекса. Из множества возможных ориентаций начального симплекса на практике используют 2 варианта:

а) Центр симплекса располагается в начале координатной системы, а одна из вершин ((n+1)-я) – на оси x_n. Остальные вершины при этом расположатся симметрично относительно координатных осей.

Координаты вершин вычисляются в этом варианте с помощью матрицы:

Номер вершины	Координаты вершин					
	x_1	x_2	x_3	...	x_{n-1}	x_n
1	$-r_1$	$-r_2$	$-r_3$	...	$-r_{n-1}$	$-r_n$
2	R_1	$-r_2$	$-r_3$	...	$-r_{n-1}$	$-r_n$
3	0	R_2	$-r_3$	...	$-r_{n-1}$	$-r_n$
...	...	...	...	...	...	…
n	0	0	0	...	R_{n-1}	$-r_n$
n+1	0	0	0	...	0	R_n

где при единичной длине ребра симплекса $r_i=\dfrac{1}{\sqrt{2i(i+1)}}$,

$R_i=\dfrac{1}{\sqrt{2(i+1)}}$, i=1, 2, ..., n.

б) Во втором варианте одна из вершин симплекса размещается в начале координат, а исходящие из нее ребра образуют одинаковые углы с соответствующими осями. Вспомогательная расчетная матрица для координат вершин начального симплекса имеет вид:

Номер вершины	Координаты вершин					
	x_1	x_2	x_3	...	x_{n-1}	x_n
1	0	0	0	...	0	0
2	p	q	q	...	q	q
3	q	p	q	...	q	q
4	q	q	p	...	q	q
...	...	...	...	...	...	…
n+1	q	q	q	...	q	p

где при единичной длине ребра $p=n-1+\frac{\sqrt{n+1}}{n\sqrt{2}}$, $q=\frac{\sqrt{n+1}-1}{n\sqrt{2}}$.

2. Получаем значения функции отклика в вершинах исходного симплекса и на этом завершаем подготовительные операции.

Алгоритм поиска

В цикле с выходом по заданному условию выполняем:

Отбрасываем вершину с наихудшим значением критерия оптимальности (наименьшим при поиске максимума или наибольшим при поиске минимума).

Вычисляем координаты вершины, зеркально отображаемой отброшенной относительно противоположной ей грани симплекса:

$$x_i=x_i(2(x_{1i}+x_{2i}+...+x_{j-1,\,i}+x_{j+1,\,i}+...+x_{n+1,\,i})/n-x_{ji},$$

где j – номер отбрасываемой вершины, i – номер координаты.

Проводим эксперимент в новой точке для получения значения целевой функции.

Условием выхода из цикла может быть малое приращение целевой функции на протяжении заданного числа опытов или при сохранении одной из вершин своего присутствия в симплексе заданное число раз и т. п.

При приближении к области оптимума точность может быть повышена уменьшением размера симплекса, но при наличии погрешностей в определении значений целевой функции необходимо ограничить размер симплекса снизу, чтобы избежать блужданий под действием случайных шумов измерений.

Преимущества метода

Число необходимых опытов для определения направления движения мало по сравнению с другими методами.

Легко учитываются ограничения на область изменения варьируемых при поиске факторов.

Эффективность метода растет с увеличением мерности пространства поиска.

Малый объем вычислений на каждом шаге.

Отсутствие высоких требований к точности оценки значения целевой функции – достаточно возможности проранжировать значения качественно по принципу «больше-меньше».

Метод пригоден для преследования дрейфующей цели (максимума или минимума), что делает его применимым в адаптивных алгоритмах.

Возможность изменять мерность пространства на ходу изменением количества вершин симплекса.

Недостатки метода

Отсутствие данных о влиянии каждого фактора на целевую функцию.

Трудность интерпретации характера поверхности отклика по данным реализуемых в методе опытов.

6.4. Программная реализация класса подпрограмм для поиска экстремума унимодальных в заданном интервале функций одной переменной

```
#include<conio.h>
#include "matrix.h"

typedef matrix<double> dmatrix;
typedef vector<double> dvector;

//Определим вначале некоторые вспомогательные подпрограммы
//Для вычисления числа Фибоначчи с заданным номером
long Fib(long n)
{
  long F[3]={0,1};
  if(n<0)
    throw xmsg("Некорректный номер числа");
  if(n<2)
    return F[n];
```

```
  for(long i=2;i<=n;i++)
  {
    F[2]=F[0]+F[1];
    F[0]=F[1];
    F[1]=F[2];
  }
  return F[2];
}
```

```
//Простая парабола для тестирования одномерного поиска
double gv(double x)
{
  return (4*x*x+5*x+1);
}
```

```
//Функциональный класс объектов для поиска экстремума; мы включили
//в него и некоторые функции определения значения вещественного корня
class MonoExtremum
{
  double c, d;//Границы для одномерного поиска
  double eps;//Для допустимой погрешности поиска
/*Указатель на подпрограмму, возвращающую значение исследуемой функции*/
  double (*getv)(double x);
 public:
  //Конструктор для одномерного поиска
  MonoExtremum(double C, double D, double EPS,
double (*GETV)(double X)):
c(C),d(D),eps(EPS),getv(GETV) { }
  //Прототипы функций поиска вещественного корня
  double RealRoot(long cod); /*Сокращением интервала неопределенности*/
  double DNewton(); //Метод секущих
  double Stoh(); //Стохастическая аппроксимация
  //Прототипы функций решения задачи условной оптимизации
  double Dihot(); //метод дихотомии
  double Fibon(long cnt); //метод Фибоначчи
  double Gold(); //метод золотого сечения
  dvector Coord(); //метод покоординатного спуска
```

```
  dvector Grad(); //метод градиента
  dvector PSM(); //последовательный симплекс-метод
};


//Обобщенная функция уточнения корня последовательным сокращением
//интервала неопределенности
double MonoExtremum::RealRoot(long cod=0)
{
  double ct=c,dt=d,x,vc,vd,vx,k;
  for(;;)
  {
    switch(cod)
    {
       case 0: //Больцано
         k=0.5;
         vc=getv(ct);
         break;
       case 1: //Хорды
         vc=getv(ct);
         vd=getv(dt);
         k=vc/(vd+vc);
         if(k<0)
           k=-k;
         break;
       case 2: //Случайный поиск
         randomize();
         k=double(1+random(1111))/1111.0;
         vc=getv(ct);
         break;
    }
    x=ct+k*(dt-ct);
    vx=getv(x);
    if(fabs(vx)<eps)
      return x;
    if(vx*vc>0)
      ct=x;
    else
      dt=x;
  }
}
```

//Метод секущих - дискретный аналог метода Ньютона

```
double MonoExtremum::DNewton()
{
  double sx[3];
  sx[0]=(c+d)/2;
  sx[1]=sx[0]+1e-2;
  sx[2]=sx[1]-(getv(sx[1]))*(sx[1]-sx[0])/(getv(sx[1])-
getv(sx[0]));
  while(1)
  {
    if(getv(sx[2])<eps)
      return sx[2];
    sx[0]=sx[1];
    sx[1]=sx[2];
   sx[2]=sx[1]-(getv(sx[1]))*(sx[1]-sx[0])/(getv(sx[1])-
getv(sx[0]));
  }
}
```

//Стохастическая аппроксимация

```
double MonoExtremum::Stoh()
{
  double scnt=1;
  //Для определения знака приращения
  double sx[3],fa[3];
  sx[0]=(c+d)/2.0;
  fa[0]=fabs(getv(sx[0]));
  if(fa[0]<eps)
    return sx[0];
  sx[1]=sx[0]+1e-2;
  fa[1]=fabs(getv(sx[1]));
  if(fa[1]<eps)
    return sx[1];
  for(scnt=1;;scnt++)
  {
    sx[2]=sx[1]-(20.0/scnt)*fa[1]*(sx[1]-sx[0])/(fa[1]-fa[0]);
    fa[2]=fabs(getv(sx[2]));
    if(fa[2]<eps)
      return sx[2];
    sx[0]=sx[1];
    sx[1]=sx[2];
    fa[0]=fa[1];
```

```
    fa[1]=fa[2];
    scnt++;
  }
}
```

//Реализация функций одномерного поиска экстремума
//Метод дихотомии

```
double MonoExtremum::Dihot()
{
  double xl,xr,yl,yr,dt=d,ct=c;
  for(;(dt-ct)>eps;)
  {
```

//задаем координаты 2-х точек эксперимента вблизи средины интервала

```
    xl=(ct+dt)/2-0.01*(dt-ct);
    xr=(ct+dt)/2+0.01*(dt-ct);
    //Вычисляем значения функции в этих точках
    yl=getv(xl);
    yr=getv(xr);
    if(sign(yl-yr)>0)
    //Бывшая правая - теперь левая, а правая - симметрично
    {
      ct=xl;
      xl=xr;
      xr=ct+dt-xl;
    }
    else
    {
      dt=xr;
      xr=xl;
      xl=ct+dt-xr;
    }
  }
  return (ct+dt)/2;
}
```

//Метод Фибоначчи для поиска экстремума

```
double MonoExtremum::Fibon(long cnt)
{
  double k=(double)Fib(cnt-1)/(double)Fib(cnt);
```

//вычисляем координаты 2-х первых точек эксперимента

```
  double ct=c,dt=d,xl=d-k*(d-c), xr=c+k*(d-c),yl,yr;
```
/*Далее в цикле в худшую точку переносим ближайшую границу, а добавочную точку определяем симметрично оставшейся относительно середины интервала */
```
  for(long i=cnt;i>1;i--)
  {
    yl=getv(xl);
    yr=getv(xr);
    if(sign(yl-yr)>0)
    {
      ct=xl;
      xl=xr;
      xr=ct+dt-xl;
    }
    else
    {
      dt=xr;
      xr=xl;
      xl=ct+dt-xr;
    }
  }
  return (ct+dt)/2;
}
```

//Метод золотого сечения
```
double MonoExtremum::Gold()
{
  double k=0.618033989;
```
  //вычисляем координаты 2-х первых точек эксперимента
```
  double ct=c,dt=d,xl=d-k*(d-c), xr=c+k*(d-c),yl,yr;
```
/*Далее в цикле в худшую точку переносим ближайшую границу, а добавочную точку определяем симметрично оставшейся относительно середины интервала*/
```
  for(;(dt-ct)>eps;)
  {
    yl=getv(xl);
    yr=getv(xr);
    if(sign(yl-yr)>0)
    {
      ct=xl;
      xl=xr;
      xr=ct+dt-xl;
```

```
    }
    else
    {
      dt=xr;
      xr=xl;
      xl=ct+dt-xr;
    }
  }
  return (ct+dt)/2;
}


//Программа тестирования методов уточнения корней и
//одномерного поиска экстремума
main()
{
  /*Тестирование методов уточнения корня последовательным сужением интервала неопределенности*/
MonoExtremum root1(-0.5,0.5,1e-3,gv);//Сконструируем объект
  double res0=root1.RealRoot(0); //0-метод Больцано
  cout<<endl<<"Bolzano:     "<<res0<<"     Func     :
"<<gv(res0);
  double res1=root1.RealRoot(1); //1-метод хорд
  cout<<endl<<"Chord    :    "<<res1<<"    Func    :
"<<gv(res1);
  double res2=root1.RealRoot(2); //2-метод Монте _ Карло
  cout<<endl<<"Rand :"<<res2<<" Func : "<<gv(res2);

  //Тестирование поиска корня методом секущих
  double resDNewton=root1.DNewton();
  cout<<endl<<"DNewton : "<<resDNewton<<" Func : "
      <<gv(resDNewton);

  //Тестирование поиска корня методом стохастической аппроксимации
  double resStoh=root1.Stoh();
  cout<<endl<<"Stoh : "<<resStoh<<" Func : "
      <<gv(resStoh);

  //Тестирование методов поиска экстремума
  MonoExtremum extr1(-3.0,+5.0,1e-3,gv);/*Сконструируем
объект*/
```

```
  //Вызываем методы решения
  double resDihot=extr1.Dihot(), //Дихотомии
         resFib=extr1.Fibon(25), //Фибоначчи
         resGold=extr1.Gold(); //Золотого сечения
  cprintf("\r\nDihot : %lf Fibon : %lf Gold : %lf ",
          resDihot,resFib,resGold);
  getch();
}
```

6.5. Программная реализация класса подпрограмм для многомерного поиска экстремума унимодальных функций

```
#include <conio.h>
#include <iostream.h>
#include "matrix.h"

typedef matrix<double> dmatrix;
typedef vector<double> dvector;


//Определим некоторые вспомогательные подпрограммы
//Возвращает код знака числа
inline long sign(double x)
{
  return (x>0)?1:((x<0)?-1:0);
}


//Простой параболоид для тестирования многомерного поиска
double gv(dvector x)
{
  double sum=0;
  int i;
  int r=x.getm();
  for(i=0;i<r;i++)
    sum+=10.0*(x[i]-4)*(x[i]-4);
  return sum;
}
```

/*Простейшая функция для численного дифференцирования, возвращающая значение вектора градиента в заданной точке при заданной матрице ограничений на составляющие аргумента*/

```
dvector GetGrad(dvector x,dmatrix rm)
{
  int r=x.getm();
  dvector dx(r),grad(r);
  double _yold=gv(x),//Текущее значение функции
         _ynew;
  for(int i=0;i<r;i++)
  {
    //Приращение аргумента возьмем в малую долю его интервала
    dx[i]=fabs(rm[i][1]-rm[i][0])*1e-10;
    if((x[i]+dx[i])>rm[i][1])
      dx[i]=-dx[i];
    _ynew=(gv(x+dx)); //Новое значение функции
    grad[i]=(_ynew-_yold)/(x[i]+dx[i]);/*Составляющая
вектора градиента*/
  }
  grad=(~grad); //Нормируем по модулю
  return grad; //Возвратим вектор градиента
}
```

```
//Функциональный класс объектов для многомерного поиска экстремума
class MultiExtremum
{
  //Размерность вектора аргументов исследуемой функции
  long range;
  //матрица ограничений на значения составляющих вектора аргументов
  dmatrix rm;
  dvector veps;//Вектор допустимых погрешностей по аргументу
  /*Указатель на подпрограмму, возвращающую значение исследуемой
функции при заданном векторе ее аргументов*/
  double (*getv)(dvector x);
 public:
/*Конструктор для многомерного поиска примет аргументы - мерность
пространства, матрицу ограничений, вектор допустимых погрешностей,
указатель на подпрограмму вычисления значений функции*/
  MultiExtremum(long RANGE, dmatrix RM, dvector
VEPS, double (*GETV)(dvector XM)): range(RANGE),
rm(RM), veps(VEPS), getv(GETV) {}
```

```
  //Прототипы функций решения задачи условной оптимизации
  void  MultiDihot(dvector&  x,int  n);/*вспомогательный
метод дихотомии*/
  dvector Coord(); //метод покоординатного спуска
  dvector Grad(); //метод градиента
  dvector PSM(); //последовательный симплекс-метод
};


/*Реализация локального поиска экстремума по одной координате ме-
тодом дихотомии*/
void MultiExtremum::MultiDihot(dvector& x,int n)
{
  double xl,xr,yl,yr,dt=rm[n][1],ct=rm[n][0];
  for(;(dt-ct)>veps[n];)
  {
    x[n]=xl=(ct+dt)/2-0.01*fabs(dt-ct);
    yl=getv(x);
    x[n]=xr=(ct+dt)/2+0.01*fabs(dt-ct);
    yr=getv(x);
    if(sign(yl-yr)>0)
    {
      ct=xl;
      xl=xr;
      xr=ct+dt-xl;
    }
    else
    {
      dt=xr;
      xr=xl;
      xl=ct+dt-xr;
    }
  }
  x[n]=(ct+dt)/2;
}


//Методы многомерного поиска
//Координатный спуск
dvector MultiExtremum::Coord()
{
  //Вначале станем в средину допустимой области поиска
  dvector xtp(range),xtn(range),xeps(range);
```

```
  int i,flag;
  for(int i=0;i<range;i++)
    xtp[i]=xtn[i]=(0.5*(rm[i][0]+rm[i][1]));
  for(flag=0;;)
  {
    for(i=0;i<range;i++)
      MultiDihot(xtn,i);
    xeps=xtn-xtp;
    for(i=0;i<range;i++)
      if(fabs(xeps[i])>fabs(veps[i]))
        flag++;
    if(!flag)
      break;
    else
      xtp=xtn;
  }
  return xtn;
}
```

//Метод градиента

```
dvector MultiExtremum::Grad()
{
  double k=5.0,vn,vp;
  dvector
xtp(range),xtn(range),xeps(range),gr(range);
  //Вначале станем в средину допустимой области поиска
  for(int i=0;i<range;i++)
    xtp[i]=(0.5*(rm[i][0]+rm[i][1]));
  for(;;)
  {
    //Вычислим градиент
    gr=GetGrad(xtp,rm);
   //Корректируем аргумент в направлении, противоположном градиен-
ту
    xtn=xtp-k*gr;
    vn=getv(xtn);
    vp=getv(xtp);
    //Если шаг слишком большой - уменьшаем его
    for(;vn>vp;)
    {
      k/=1.2;
      xtn=xtp-k*gr;
```

```
      vn=getv(xtn);
      vp=getv(xtp);
    }
    /*Если шаг по градиенту не изменяет существенно значение функции - прекращаем поиск*/
    if(fabs(getv(xtn)-getv(xtp))<1e-8)
      break;
    xtp=xtn;
  }
  return xtn;
}


/*Последовательный симплекс-метод.
Этот метод достаточно громоздок в программном исполнении, особенно при использовании переменного размера поискового симплекса, защиты от вращения симплекса вокруг одной из вершин и других манипуляций, необходимых для его эффективной работы. Мы приводим упрощенный, а, следовательно, не очень точный в вычислениях пример программы, демонстрирующий принцип работы алгоритма */
dvector MultiExtremum::PSM()
{
  double sum;
  int i,j;

  dvector  yv(range+1);/*Вектор значений функции в вершинах симплекса*/
/*Формируем исходный регулярный симплекс в пространстве размерности range с центром в начале координат */
  dmatrix simp(range+1,range);//матрица для вершин симплекса
  dvector stmp(range);//
  const double L=2.0L;
  //заполняем матрицу симплекса
  //Сначала нулевую строку
  for(j=0;j<range;j++)
    simp[0][j]=0.0;
  //Затем остальные
  double pk=(L/(range*sqrt(2)))*
            (sqrt(range+1)+double(range)-1.0);
  double qk=L*(pk-0.5*sqrt(2));
  for(i=1;i<(range+1);i++)
    for(j=0;j<range;j++)
```

```
      simp[i][j]=(j==(i-1))?pk:qk;
  int pmax=-1,//предыдущая плохая
      cicl;
  int vmin=0,vmax=0;
  //Бесконечный цикл поиска
  for(cicl=1;/*cicl<100000*/;cicl++)
  {
    //Вычисляем значения функции во всех вершинах
    for(i=0;i<(range+1);i++)
      yv[i]=getv(simp[i]);
    //Отыскиваем номера наихудшей и наилучшей вершин
    for(i=1;i<(range+1);i++)
    {
      if(yv[i]>yv[vmax])
        vmax=i;
      if(yv[i]<yv[vmin])
        vmin=i;
    }
    //Если наилучшая нас устраивает - прекращаем поиск
    if(fabs(yv[vmin])<0.05)
      break;//Грубая оценка попадания в зону экстремума
    /*В этом месте вместо break можно было бы уменьшить размер симплекса и условие достижения экстремума*/
    //Если плохая не сменила номер - ищем ближайшую плохую
    if((cicl>1)&&(vmax==pmax))
      for(i=1;i<(range+1);i++)
      {
        if((yv[i]>yv[vmax])&&(i!=pmax))
          vmax=i;
      }
    pmax=vmax;//Запоминаем как предыдущую плохую
    //меняем плохую вершину на зеркально отраженную
    for(j=0;j<range;j++)//j-номер координаты
    {
      /*Для каждой координаты вычислим ее среднее значение по всем вершинам*/
      sum=0;
      for(int k=0;k<(range+1);k++)
        sum+=simp[k][j];//k- номер вершины
      sum*=(2.0/(range+1));
      /*Из среднего вычитаем плохое значение и результат сохраняем в промежуточном векторе*/
```

```
      stmp[j]=sum-simp[vmax][j];
    }
    /*Новый вектор координат переносим на место наихудшей вершины*/
    simp[vmax]=stmp;
  }//бесконечный цикл
  return simp[vmin];
}


//Программа тестирования методов поиска экстремума
main()
{
  //Тестируем многомерные методы поиска экстремума
  //Формируем размерность, матрицу ограничений и вектор погрешностей
  int r=5;
  dmatrix m(r,2);dvector e(R);
  for(int i=0;i<r;i++)
  {
    m[i][0]=-8.0;
    m[i][1]=15.0;
    e[i]=0.0001;
  }
  MultiExtremum extM(r,m,e,gv);//Конструируем объект
  //Вызываем для этого объекта методы поиска минимума
  dvector resCoord=extM.Coord(); /*Метод покоординатного спуска*/
  cout<<endl<<gv(resCoord);
  cout<<endl<<resCoord;
  dvector resGrad=extM.Grad(); /*Метод градиента (наискорейшего спуска)*/
  cout<<endl<<gv(resGrad);
  cout<<endl<<resGrad;

  dvector resPSM=extM.PSM(); /*Последовательный симплекс-метод*/
  cout<<endl<<gv(resPSM);
  cout<<endl<<resPSM;
  getch();
}
```