МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

кафедра молекулярной физики

ПОИСК НАИЛУЧШЕГО ПРИБЛИЖЕНИЯ НАБОРА ДВУХМЕРНЫХ ДАННЫХ *N*-ПАРАМЕТРИЧЕСКОЙ ФУНКЦИЕЙ

ТЕХНИЧЕСКОЕ ОПИСАНИЕ

070500 Т00000 001 ТО

Разработчик _____ Александров О.Е.
к. ф.-м. н., программист 13 разряда __.__.98

Екатеринбург 2003

РЕФЕРАТ

Александров О.Е. *ПОИСК НАИЛУЧШЕГО ПРИБЛИЖЕНИЯ НАБОРА ДВУХМЕРНЫХ ДАННЫХ N-ПАРАМЕТРИЧЕСКОЙ ФУНКЦИЕЙ*: Техническое описание. Екатеринбург: УГТУ, 2003. 21 с.

Библиогр. 0 назв. Рис. 2. Табл. 0. Прил. 2.

Издание первое.

ПРИБЛИЖЕНИЕ ДАННЫХ ФУНКЦИЕЙ, ПОИСК МИНИМУМА, МЕТОД НАИМЕНЬШИХ КВАДРАТОВ.

Описывает структуру и принципы построения процедуры поиска наилучшего приближения двумерного набора данных (X_i, Y_i), $i = 1, \ldots M$ заданной N-параметрической функцией $Y = F(X, P_1, \ldots P_N)$. Включает описание реализации обобщенной процедуры поиска экстремума функции и описание способа задания пользовательской аппроксимирующей функции. Дает рекомендации по использованию при написании пользовательских программ.

Язык программы - Borland Pascal 7.0. Модуль модифицирован, с сохранением DOS-совместимости, и протестирован для работы в среде Delphi 3.0.

Использованы процедуры вычисления специальных функций (функция ошибок *erf* и гамма-функция Γ) из библиотеки *Numerical Recipes*, © Copr. 1986-92 Numerical Recipes Software .5-28.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Процедура поиска наилучшего приближения двумерного набора данных (X_i, Y_i), $i = 1, \ldots M$ заданной N-параметрической функцией $Y = F(X, P_1, \ldots P_N)$ была разработана для нужд автоматизации измерений на масс-спектрометре МИ1201-АГМ.

1. ИЗМЕРЕНИЕ ИНТЕНСИВНОСТИ СИГНАЛА

1.1. Для чего это нужно

Измерение сигнала в точке, соответствующей центру пика на масс-спектре необходимо для корректного сравнения интенсивности сигналов и определения доли вещества в пробе.

Точное измерение интенсивности особенно важно при анализе микропримесей.

1.2. Как это можно сделать

Измерение интенсивности сигнала в пике может быть разделено на два этапа[1)]:

1) Определение положения центра пика[2)]. Центром пика считается точка сигнала максимальной интенсивности при отсутствии шумов[3)].
2) Измерение интенсивности сигнала в центре пика достаточно длительное время[4)]. Длительное время измерения сигнала позволяет увеличить точность измерения, за счет усреднения шумовой составляющей сигнала.

Простейшим вариантом поиска мог бы быть простое измерение сигнала с необходимой длительностью при каждом допустимом значении магнитного поля (т.е. с минимальным шагом развертки). Недостаток такого подхода заключается в чрезмерных затратах времени на измерение.

[1)] Ниже будет предложена и иная схема измерения.

[2)] Положение центра пика может быть задано различно, например, значением тока электромагнита, в единицах счетчика импульсов развертки или в единицах отношения массы иона к заряду. Для дальнейшего это несущественно, но мы будем оперировать числом импульсов контроллера развертки, поскольку это, фактически, минимальный шаг, на который мы можем изменять эл.-магн. поле.

[3)] На этот счет существуют и другие мнения.

[4)] Предполагается, что мы имеем возможность установить любое требуемое значение положения с необходимой точностью (фактически эта точность определяется контроллером развертки и кое-какими привходящими обстоятельствами).

2. УПРОЩЕННАЯ ПРОЦЕДУРА ПОИСКА ЦЕНТРА ПИКА

2.1. Исходные данные и математическая формулировка задачи

В качестве исходных данных выступает некоторая последовательность точек (x_i, y_i), $i = -N, -(N+1), \ldots, -1, 0, 1, \ldots M$ и в общем случае $N \neq M$. При этом все y_i удовлетворяют условию $y_i \geq y_{min}$, где y_{min} некоторый априорно заданный уровень сигнала.

Величина x - характеристика интенсивности магнитного поля и обычно изменяется по закону: $x_i = x_0 + \Delta x \cdot i$, где Δx - некоторый постоянный шаг развертки (ШР) с которым изменяется напряженность поля электромагнита (ЭМ). Величина ШР удовлетворяет условию $\Delta x << \sigma$, где 2σ - ширина пика на половине его высоты. Значение x_0 соответствует максимальному значению y в последовательности ($y_0 > y_i$ для любого $i \subset [-N .. M]$) [5]. Значение x_0 не обязательно совпадает с истинным центром пика (экстремумом).

Величина y_i - характеристика интенсивности выходного сигнала масс-спектрометра (ионный ток), измеренная в точке x_i.

Предполагается, что мы можем получить такую последовательность в произвольного интервала шкалы x масс-спектра.

Необходимо найти значение x_{max}, при котором y_{max} соответствует истинному экстремуму пика.

2.2. Упрощенный алгоритм поиска центра для симметричного пика

Если мы предположим, что пик симметричен относительно x_{max}, то можно отыскать x_{max} следующим образом:

$$x_{\max} = \frac{\sum_{i=-N}^{-1}\left(x_i + x_i'\right) + \sum_{i=1}^{M}\left(x_i + x_i'\right)}{2\left(N + M\right)}, \qquad (2.1)$$

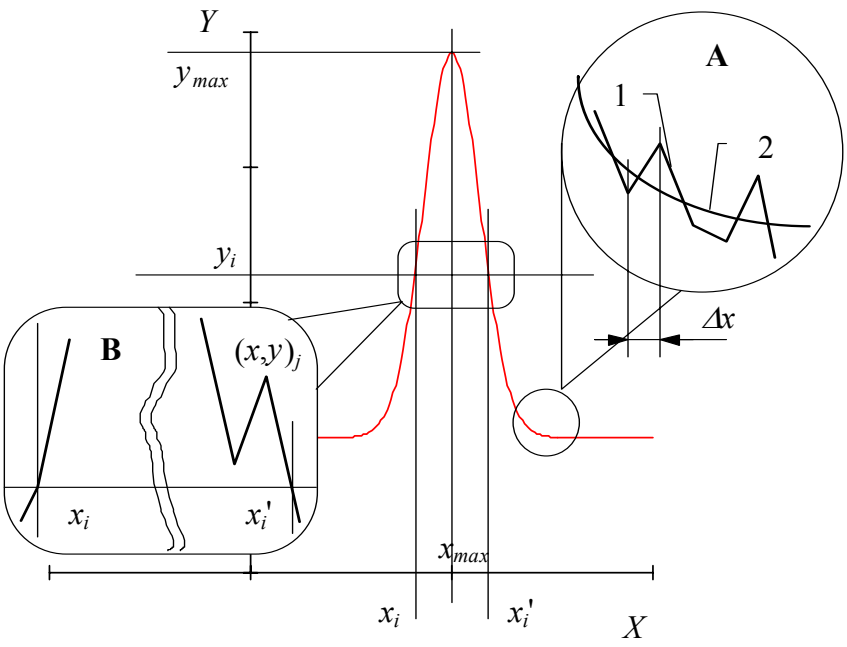
где x_i - точка исходной последовательности; x_i' - соответствует точке, в которой интенсивность сигнала $y = y_i$, но расположенной на противоположном склоне

[5]) Если существует несколько $y_j = y_{max}, j = j_1 \ldots j_2$ и $j \subset [-N .. M]$ то в качестве искомого y_i принимается $i = (j_1 + j_2)/2$.

пика, как это изображено на рис. 2.1. Поскольку такой точки в исходной последовательности скорее всего нет, то x_i' следует вычислить как пересечение прямой $y = y_i$ с соответствующим сегментом ломаной линии, определяемой точками $(x, y)_j$ и $(x, y)_{j+1}$ такими, что $y_j > y_i$ и $y_{j+1} < y_i$ (см. рис. 2.1 выноска В).

Недостаток этого алгоритма — явная неработоспособность при частичном наложении пиков (несимметричный пик).

ОТЫСКАНИЕ ПОЛОЖЕНИЯ ЦЕНТРА СИММЕТРИЧНОГО ПИКА



A - вид экспериментальной 1 и реальной 2 кривой. Экспериментальная кривая это ломаная линия, вершины которой не совпадают с реальной кривой из-за случайных помех.
B - схема вычисления x_i'.

Рис. 2.1

3. ПОИСК ЦЕНТРА ПИКА НА ОСНОВЕ АППРОКСИМАЦИИ ЭКСПЕРИМЕНТАЛЬНЫХ ТОЧЕК ФУНКЦИЕЙ

Основными преимуществами восстановления функции по экспериментальным точкам являются возможности: а) восстановления одновременно центра пика и интенсивности сигнала в центре; в) восстановление параметров для перекрывающихся пиков.

3.1. Эквивалентность усреднения в пространстве и времени

Альтернативой алгоритму пункта 2.2 является отыскание центра пика методом восстановления истинной[6)] кривой.

Для этого, как минимум, необходим выбор функционального описания пика: $y = f(x, \mathbf{P})$, где $\mathbf{P} = (p_1, p_2, \ldots p_K)$ - вектор параметров функции, который надо определить по экспериментальным данным из условия наилучшего совпадения.

Прямолинейный подход состоял бы в использовании метода наименьших квадратов (МНК), однако не следует торопиться.

Основной недостаток МНК сложность применения для функций, которые невозможно линеаризовать.

Вспомним, что МНК, лишь способ устранить случайные возмущения, возникающие при экспериментальном измерении. Он использует множество точек $(x, y)_i$ для восстановления параметров функции.

В некотором приближении это можно интерпретировать как усреднение в пространстве x_i, для точек с ограниченным временем измерения (чем меньше время измерения в точке, тем больше относительная случайная погрешность).

Но с другой стороны, мы можем повысить достоверность измерения одной точки, за счет уменьшения их числа. В частности мы можем потратить все время измерения для отыскания K точек на пике, где K - число параметров выбранной функции. В этом случае, указанные K точек будут определены с большей достоверностью.

Если принять гипотезу, что погрешность измерения интенсивности сигнала в точке обратно пропорциональна времени измерения, то затратив равное время на измерение меньшего числа точек, мы не ухудшим совокупную погрешность нового набора исходных данных.

Здесь есть тонкость, связанная с тем, что если мы потратим все время на измерение M - точках для $M < K$, то мы явно ухудшаем результат — восстановить функцию станет невозможно. Аналогичный результат может быть достигнут при выборе неудачных точек для измерения, например, на одной стороне пика или очень близко друг к другу. Поэтому тезис об эквивалентности усреднения в пространстве и времени может быть неверен.

Восстановление искомой функции $y = f(x, \mathbf{P})$ по K точкам сводится к решению системы K уравнений:

$$y_j = f(x_j, \mathbf{P}), j = 1, 2, K \qquad (3.1)$$

[6)] Имеется ввиду, что есть возможность установить вид функциональной зависимости для пика теоретически.

относительно неизвестных параметров **P**. Такое решение может оказаться проще (для небольшого числа параметров), чем применение МНК.

3.2. Измерение при наложении пиков

Для случая изолированного пика алгоритм восстановления функции не дает существенного выигрыша. Единственный плюс — не надо проводить дополнительное точное измерение интенсивности в центре пика, поскольку автоматически восстанавливается и амплитуда сигнала.

Существенно большее преимущество дает этот метод при измерении в условиях наложения S пиков[7)]. В этом случае искомой функцией будет суперпозиция функций $f(x, \mathbf{P}_s)$:

$$y = \sum_{s=1}^{S} f\left(x, \mathbf{P}_s\right), \tag{3.2}$$

содержащая $S \cdot K$ параметров.

Реально число параметров должно возрастать медленнее, поскольку можно предположить, что все пики геометрически подобны и отличаются только амплитудой. Тогда, число параметров при наложении S пиков составит $K + (S\text{-}1)$, т.е. потребует для разрешения S пиков дополнительно замерить S точек.

3.3. Аппроксимирующая функция для пика

Поскольку распределение плотности ионов в пучке, падающем нам приемную щель детектора ионов (рис. 3.1) может быть аппроксимировано нормальным распределением (график 1 на рис. 3.1)

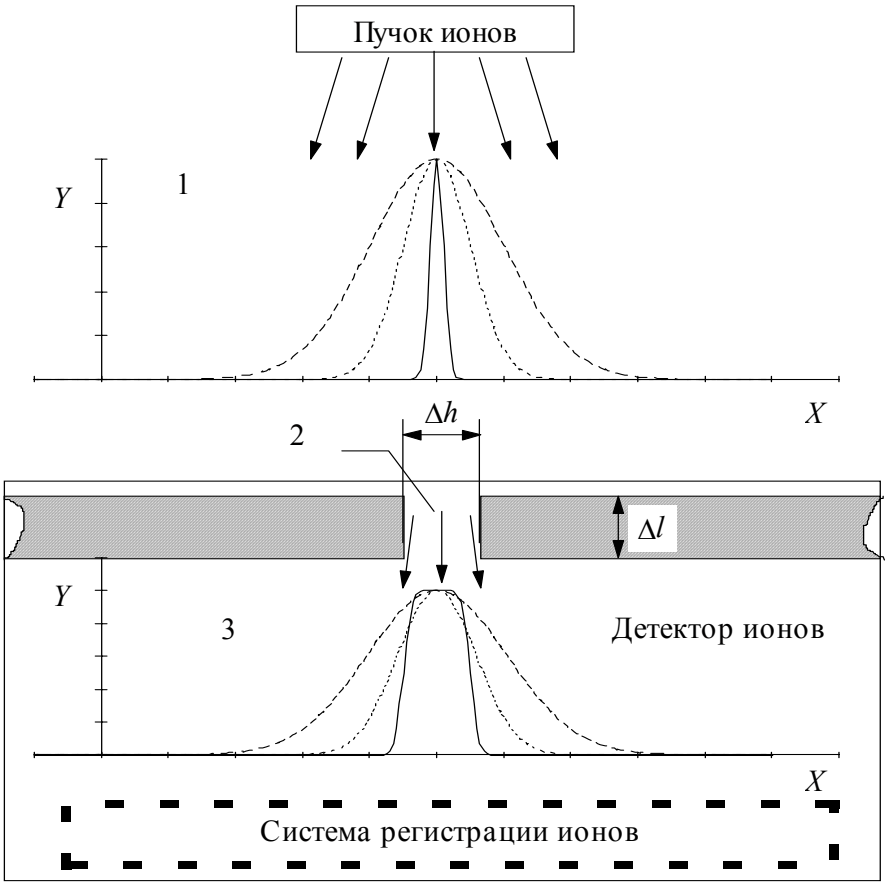
$$y_0\left(x\right) = \frac{A_0}{\sigma\sqrt{2\pi}} e^{-\frac{(x-x_0)^2}{2\sigma^2}} + B_0, \tag{3.3}$$

здесь A_0 - амплитуда пика, B_0 - фон пика, σ - полуширина пика, x_0 - положение центра пика.

Распределение в форме (3.3) означает принятие гипотезы, что на распределение плотности ионов в пучке влияет значительное число независимых случайных сил, с постоянным распределением плотности вероятности.

[7)] Наложение пиков в реальном масс-спектре существует всегда, но им обычно пренебрегают если расстояние между центрами пика много больше или просто больше их суммарной ширины.

СВЯЗЬ РАСПРЕДЕЛЕНИЯ ПЛОТНОСТИ ПОТОКА ИОНОВ В ПУЧКЕ И ИНТЕНСИВНОСТИ РЕГИСТРИРУЕМОГО СИГНАЛА



1 - распределение интенсивности сигнала до приемной щели детектора;
2 - приемная щель детектора;
3 - интегральная[8)] интенсивность сигнала после приемной щели детектора.
Показано три различных по ширине распределения плотности ионов в пучке и соответствующие им интенсивности регистрируемого сигнала

Рис. 3.1

Детектор ионов представляет собой премную щель 2 (см. рис. 3.1) шириной Δh в пластине толщиной Δl, которая вырезает из пучка ионов некоторую его часть и пропускает ее к системе регистрации ионов. Высота щели полагается бесконечно большой.

Для случая бесконечно тонкой пластины ($\Delta l = 0$) интенсивность, регистрируемая детектором будет иметь вид распределения 3 (см. рис. 3.1)

$$y(x) = \frac{A_0}{\sigma\sqrt{2\pi}} \int_{x-\frac{\Delta h}{2}}^{x-\frac{\Delta h}{2}} e^{-\frac{(x-x_0)^2}{2\sigma^2}} dx + B_0 \Delta h . \qquad (3.4)$$

Очевидно, что при стремлении ширины щели Δh к нулю или для $\Delta h << \sigma$ распределение интенсивности после щели стремится к (3.3). При сравнимой щели Δh и ширине пика до щели σ, распределение интенсивности регистрируемого

[8)] Предполагается, что все ионы, прошедшие щель, регистрируются.

сигнала приобретает плоскую вершину в окрестности максимума и стремится к прямоугольной форме (см. рис. 3.1).

Функция (3.3) может быть представлена в форме

$$y(x) = A \cdot \left\{ erf\left(\frac{x - x_0 + \Delta h/2}{\sigma\sqrt{2}} \right) - erf\left(\frac{x - x_0 - \Delta h/2}{\sigma\sqrt{2}} \right) \right\} + B \qquad (3.5)$$

где $erf(x)$ - функция ошибок $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2} dx$, это широко известная специальная функция и для нее есть хорошие аппроксимации и методы численного вычисления, например, см. прил. 2.

Вычисление функции необходимо производить быстро. Для сложных функций, не реализованных аппаратно в процессоре, применяют либо интерполяцию по табличным значениям, либо метод суммирования ряда (хотя ряды обычно плохо сходятся). Пример применения обеих методов см., например, прил. 2.

3.4. Восстановление функции по данным измерения

В этом разделе рассмотрен вопрос применения МНК для восстановления параметров функции произвольного вида.

3.4.1. Общая постановка задачи

Пусть дана последовательность:

$$(x_i, y_i),\ i = 1, 2, \ldots N, \qquad (3.6)$$

и функция

$$y = F(x, \mathbf{P}), \qquad (3.7)$$

где $\mathbf{P}$ - вектор параметров, $\mathbf{P} = (p_1, p_2, \ldots p_K)$.

Необходимо найти такой $\mathbf{P}_0$, чтобы функционал

$$\Phi(\mathbf{P} = \mathbf{P}_0) = \sum_{i=1}^{N} \left[F(x_i, \mathbf{P}_0) - y_i \right]^2 \qquad (3.8)$$

принимал минимальное значение.

Необходимое условие экстремума в точке $\mathbf{P}_0$:

$$\frac{\partial \Phi(\mathbf{P}_0)}{\partial P_k} = 0,\ k = 1, \ldots N \qquad (3.9)$$

или

$$2\sum_{i=1}^{N} \frac{\partial F(x_i, \mathbf{P}_0)}{\partial P_k} \cdot \left[F(x_i, \mathbf{P}_0) - y_i \right] = 0,\ k = 1, \ldots N$$

Таким образом есть альтернатива, либо решать уравнение

$$\Phi(\mathbf{P}_0) = \min, \tag{3.10}$$

либо систему N уравнений

$$\sum_{i=1}^{N} \frac{\partial F(x_i, \mathbf{P}_0)}{\partial P_k} \cdot [F(x_i, \mathbf{P}_0) - y_i] = 0, \; k = 1, \ldots N \tag{3.11}$$

Поскольку приближение для пика, описанное в пункте 3.3, не является абсолютно точным, более того экспериментальные измерения дают несимметричные пики (хвост в сторону увеличения массы иона), то является разумным ориентироваться на восстановление произвольной функции. В этом случае, невозможно использовать особенности конкретного вида функции и линеаризовать или другим образом упростить уравнения (3.10) или (3.11). Поэтому приходится ориентироваться на численный расчет.

3.4.2. Общие принципы численного решения уравнений

Обычная схема численного решения уравнения может быть представлена в виде:

1) Выбор начального приближения для решения $\mathbf{P}_0$ в области допустимых значений $\mathbf{P}$. Пусть начальное приближение выбрано и равно $\mathbf{P}_{00}$.
2) Отыскание следующего приближения $\mathbf{P}_{01}$ такого, чтобы расстояние от $\mathbf{P}_{01}$ до $\mathbf{P}_0$, было меньше чем расстояние от $\mathbf{P}_{00}$ до $\mathbf{P}_0$.
3) Повтор операции 2) для вычисления последовательности $\mathbf{P}_{0j}$, $j = 1, 2, \ldots$, сходящейся к $\mathbf{P}_0$.
4) Вычисление $\mathbf{P}_{0j}$ до $j = M$ такого, что расстояние от $\mathbf{P}_{0M}$ до $\mathbf{P}_0$ будет меньше заданной величины ε.
5) Величина $\mathbf{P}_{0M}$ принимается как искомая величина $\mathbf{P}_0$.

Наиболее важным этапом является выбор алгоритма вычисления $\mathbf{P}_{0j}$. Этот алгоритм должен: а) обеспечивать сходимость к $\mathbf{P}_0$; б) обеспечивать максимальную скорость сходимости. т.е. минимальное число M.

3.4.3. Метод наискорейшего спуска

Применяется для поиска экстремумов, т.е. в нашем случае пригоден для решения уравнения (3.10).

В этом случае выбирают:

$$\mathbf{P}_{0\,j+1} = \mathbf{P}_{0\,j} - S_j \cdot \frac{d\Phi(\mathbf{P}_{0\,j})}{d\mathbf{P}} \tag{3.12}$$

где S - некоторая произвольная положительная величина. В общем случае S может изменяться от шага к шагу для достижения оптимальной сходимости.

Вычисление по (3.12) выполняет смещение точки в направлении против градиента функции, т.е. в сторону уменьшения значения функции или в направлении наискорейшего спуска. Величину S следует определить из условий: а) обеспечения сходимости, б) максимальной скорости сходимости.

В нашем случае, когда известна точная нижняя граница функции $\Phi(\mathbf{P})$, равная нулю, выбор S можно сделать так: новая точка $\mathbf{P}_{0j+}1$ определяется как пересечение прямой сонаправленной с вектором $\mathbf{grad}_{\mathbf{P}}(\Phi(\mathbf{P}_{0j}))$ и проведенной через точку $\Phi(\mathbf{P}_{0j})$ с плоскостью $\mathbf{P} = 0$. Что дает выражение для S

$$S_j = \frac{\Phi\left(\mathbf{P}_{0\,j}\right)}{\left|\frac{\partial\Phi\left(\mathbf{P}_{0\,j}\right)}{\partial\mathbf{P}}\right|} \tag{3.13}$$

Скорее всего этот способ вычисления неприемлем. Простейший аргумент — при приближении к точке минимума мы получим увеличение шага и деление на ноль в точке экстремума

3.4.4. Модификация метода наискорейшего спуска для гарантии сходимости

Основной проблемой всех способов выбора следующего значения приближения является обеспечение сходимости в любых условиях. Для улучшения этого показателя любую методику выбора следующего значения приближения можно дополнить следующими шагами:

1) Вычисляется значение функции в новой точке $\Phi(\mathbf{P}_{0j+1})$.
2) Сравнивается со значением в предыдущей точке $\Phi(\mathbf{P}_{0j})$.
3) Если значение $\Phi(\mathbf{P}_{0j+1}) \geq \Phi(\mathbf{P}_{0j})$, то $\mathbf{P}_{0j+1}$ корректируется, новое значение вычисляется по формуле $\mathbf{P}_{0j+1}' = (\mathbf{P}_{0j+1} + \mathbf{P}_{0j})/2$.
4) Шаги 1)-3) повторяются пока не выполнится неравенство $\Phi(\mathbf{P}_{0j+1}) < \Phi(\mathbf{P}_{0j})$.

Это усовершенствование почти гарантирует сходимость, при этом скорость сходимости ухудшается незначительно.

3.4.5. Параболическое уточнение шага

Вычисление следующего значения приближения по формуле (3.13) является весьма грубым и может существенно сказываться на скорости сходимости, ухудшая ее.

Для ускорения сходимости можно использовать аппроксимацию многочленом второй степени для более точного вычисления следующего значения $\mathbf{P}_{0j+1}$.

В этом случае предполагается, что функцию вблизи экстремума можно представить приближенно в виде:

$$\Phi(\mathbf{P}) = a\cdot\mathbf{P}^2 + \mathbf{b}\cdot\mathbf{P} + c. \tag{3.14}$$

Такое приближение является весьма точным для ближайшей окрестности минимума (это следует из разложения в ряд Тейлора).

Если мы знаем коэффициенты в (3.14), то можно вычислить приближение для точки экстремума, решив уравнение

$$2a\cdot\mathbf{P} + \mathbf{b} = 0. \qquad (3.15)$$

Сами коэффициенты a и $\mathbf{b}$ могут быть вычислены, например, по значению функции $\Phi(\mathbf{P}_{0j})$, ее производной $\frac{d\Phi(\mathbf{P}_{0j})}{d\mathbf{P}}$ в точке $\mathbf{P}_{0j}$ и значению функции в предыдущей точке $\Phi(\mathbf{P}_{0j-1})$.

Однако, поскольку у начального приближения $\mathbf{P}_{00}$ нет предыдущей точки, то параболическая аппроксимация далее используется по следующему алгоритму:

1) Вычисляется $\mathbf{P}_{0j+1}$ по методу описанному в пунктах 3.4.3 и 3.4.4. При этом оказываются известными $\Phi(\mathbf{P}_{0j})$, $\frac{d\Phi(\mathbf{P}_{0j})}{d\mathbf{P}}$ и $\Phi(\mathbf{P}_{0j+1})$.
2) На основе $\Phi(\mathbf{P}_{0j})$, $\frac{d\Phi(\mathbf{P}_{0j})}{d\mathbf{P}}$ и $\Phi(\mathbf{P}_{0j+1})$ восстанавливаются коэффициенты параболической аппроксимации (3.14) и вычисляется уточненное значение $\mathbf{P'}_{0j+1}$ по формуле (3.15).

Вычисление $\mathbf{P'}_{0j+1}$ на основе параболического уточнения может быть выполнено по следующему алгоритму:

1) Модифицируем (3.14) к виду: $\Phi(\mathbf{P}) = a\cdot(\mathbf{P} - \mathbf{P}_{0j})^2 + \mathbf{b}\cdot(\mathbf{P} - \mathbf{P}_{0j+1}) + c$. Это не ухудшает общности, но упрощает восстановление коэффициентов.
2) Для вычисления значений коэффициентов следует решить систему уравнений:

$$\Phi_{0j} = \mathbf{b}\cdot(\mathbf{P}_{0j} - \mathbf{P}_{0j+1}) + c$$

$$\frac{d\Phi(\mathbf{P}_{0j})}{d\mathbf{P}} = \mathbf{b} \qquad (3.16)$$

$$\Phi_{0j+1} = a\cdot(\mathbf{P}_{0j+1} - \mathbf{P}_{0j})^2 + c$$

3) Решая уравнение (3.15), с учетом модификации, получим: $\mathbf{P'}_{0j+1} = -\frac{\mathbf{b}}{2a} + \mathbf{P}_{0j}$. Откуда для уточненного S' получим:

$$S'_j = \frac{1}{2}\frac{(\mathbf{P}_{0j+1} - \mathbf{P}_{0j})^2}{\Phi_{0j+1} - \Phi_{0j} - \mathbf{grad}(\Phi(\mathbf{P}_{0j}))\cdot(\mathbf{P}_{0j+1} - \mathbf{P}_{0j})}. \qquad (3.17)$$

Вычисление $\mathbf{P'}_{0j+1}$ можно провести по формуле (3.12).

С учетом предистории вычисления $\mathbf{P}_{0j+1}$ по формуле (3.12), можно записать вместо (3.17)

$$S'_j = \frac{1}{2} \frac{\left(S_j \cdot \mathbf{grad}\left(\Phi\left(\mathbf{P}_{0j}\right)\right)\right)^2}{\Phi_{0j+1} - \Phi_{0j} - S_j \cdot \left(\mathbf{grad}\left(\Phi\left(\mathbf{P}_{0j}\right)\right)\right)^2} \cdot \qquad (3.18)$$

3.5. О программе для численного решения уравнения

Здесь приведены рассуждения о возможной реализации метода наискорейшего спуска. Для других методов суть подхода не меняется.

Реализация программы должна обеспечивать применения алгоритма для произвольной аппроксимирующей функции. В качестве примера для подражания можно взять реализацию функции «genfit» в пакете MathCAD.

Программа должна позволять передать процедуре численного поиска минимума: 1) начальные значения приближения для параметров — $\mathbf{P}_{00}$, 2) процедуру вычисления значения функции в произвольной точке $\mathbf{P}$ — $\Phi(\mathbf{P})$, 3) процедуру вычисления вектора из частных производных функции $\Phi(\mathbf{P})$ в точке $\mathbf{P}$ — $\mathbf{grad}_{\mathbf{P}}(\Phi(\mathbf{P}))$, 4) желаемую точность вычисления ε.

Программа должна вернуть значение $\mathbf{P}_0$ с указанной точностью или информировать о невозможности отыскания такового. Приемлемым вариантом может быть возврат значения $\mathbf{P}_0$ с указанием достигнутой точности.

Передачу процедур вычисления функции и параметрического градиента функции следует сделать через функциональные параметры типа *F:function(…);* или *F:procedure(…)*.

После реализации такой процедуры поиск приближений для новых аппроксимирующих функций будет заключаться только в программировании процедур вычисления значения функции и градиента функции.

4. ОПИСАНИЕ ПРОЦЕДУРЫ *GeneralFit*

Алгоритм обобщенного поиска наилучшего приближения набора двухмерных данных *N*-параметрической функцией реализован в модуле *uGenFit* (см. файл *uGenFit.pas*).

4.1. Основная процедура модуля *uGenFit* и ее аргументы

Основные процедуры модуля: *GeneralFit* и *GeneralFitX*. Они различаются тем, что *GeneralFitX* дополнительно возвращает значение достигнутой точности вычислений.

Процедуры выполняют поиск значений параметров произвольной функции, при которых наилучшим образом (минимум суммы квадратов отклонений) аппроксимируется набор заданных точек x_i, y_i.

Их заголовки определены следующим образом:

procedure *GeneralFit(aF:tFunc; aGradF:tGradF; var P:tParameters; const X,Y:tData; Eps:Double; var ErrCode:tErrorCode);*

Описание аргументов см. *GeneralFitX.*

procedure *GeneralFitX(aF:tFunc; aGradF:tGradF; var P:tParameters; const X,Y:tData; Eps:Double; var ReachedEps:Double; var ErrCode:tErrorCode);*

Аргументы процедуры имеют следующее значение: *aF* - вычисляет значение аппроксимирующей функции в точке *x* при параметрах **P**, тип определен как *tFunc = function (const P:tParameters; X:Double):Double; aGradF* - вычисляет значения всех частных производных аппроксимирующей функции по всем P_i в точке *x* при параметрах **P**, тип определен как *tGradF = procedure(const P:tParameters; X:Double; var GradF: tParameters)*; *P* - ВХОД: значения начального приближения для параметров, ВЫХОД: значения найденного наилучшего приближения для параметров, типы определены так

```
tParameterNumber=1..cMaxParameterSize;
tParametersArray=array[tParameterNumber] of Double;
tParameters=record
        Size:tParameterNumber;
        Par:tParametersArray;
end;
```

X, *Y* - массивы данных x_i и y_i, типы определены так

```
tDataNumber=1..cMaxDataSize;
tDataArray=array[tDataNumber] of Double;
tData=record
        Size:tDataNumber;
        Data:tDataArray;
end;
```

Eps - желаемая погрешность вычислений; *ReachedEps* - возвращает реально достигнутую погрешность, в любом случае; *ErrCode* - возвращает код ошибки или 0 — если нет ошибки.

4.2. Вспомогательные процедуры модуля *uGenFit*

procedure ***AllocateP(var pP:tPtrParameters; Size:tParameterNumber);***

procedure ***AllocateD(var pD:tPtrData; Size:tDataNumber);***

Размещают в памяти структуру *tParameters* или *tData* размером *Size* и устанавливают значение *Size* в поле *Size* структуры (см. определение типа *tParameters* и *tData*). Размер — это число элементов массива РЕАЛЬНО РАЗМЕЩЕНЫХ В ПАМЯТИ. Обычно *Size* < *cMaxParameterSize* или *cMaxDataSize*. НЕ РЕКОМЕНДУЕТСЯ изменять значение поля *Size* структуры.

procedure ***FreeP(var pP:tPtrParameters);***

procedure ***FreeD(var pD:tPtrData);***

Освобождают память, занимаемую структурой *tParameters* или *tData*, размер освобождаемой памяти берется из поля *Size* структуры (см. определение типа *tParameters* и *tData*).

procedure ***MoveP(const p1:tParameters; var p2:tParameters);***

procedure ***MoveD(const d1:tData; var d2:tData);***

Присваивает *p2* (*d2*) значение *p1* (*d1*) с учетом их размера. Возбуждает ошибку *RunError*(201), если размеры структур не совпадают. Размер берется из поля *Size* структуры (см. определение типа *tParameters* и *tData*).

4.3. Процедуры подготовки данных

Для увеличения скорости и точности вычислений процедуры *GeneralFit* рекомендуется нормализовать исходные данные с помощью процедур модуля *uDataPreparation*, см. файл *uDataPre.pas*.

Модуль содержит процедуру нормализации данных, которая приводит все точки исходных массивов к диапазону [0..1] и возвращает заполненную структуру данных для обратного преобразования (денормализации). Алгоритм нормализации вектора данных:

1) В массиве (векторе) ищется наименьшее значение, запоминается в структуре данных обратного преобразования.
2) Из всех элементов массива вычитается наименьшее значение, найденное на шаге 1).
3) Ищется наибольшее значение в массиве, запоминается в структуре данных обратного преобразования.
4) Все элементы массива делятся на наибольшее значение, найденное на шаге 3).

procedure *NormalizeVector(var NP:tNormalizeParameter; var V:tData);*

Нормализует вектор данных *V* и возвращает параметры преобразования в структуре *NP*. Тип *tNormalizeParameter* определен так:

```
tNormalizeParameter=record
        Origin, Scale:Double;
end;
```

procedure *NormalizeData(var ND:tNormalizeParameters; var X,Y:tData);*

Нормализует данные *X*, *Y* и возвращает параметры преобразования в структуре *ND*. Тип *tNormalizeParameters* определен так:

```
tNormalizeParameters=record
        x,y:tNormalizeParameter;
end;
```

function *GetUnNormalized(const NP:tNormalizeParameter; const X:Double):Double;*

Возвращает значение *X* преобразованное к ненормализованному значению в соответствии с данными *NP*.

4.4. Задание аппроксимирующих функций

ВНИМАНИЕ! Данная реализация процедуры *GeneralFit* принимает в качестве функциональных параметров аппроксимирующую функцию $F(x, \mathbf{P})$, а не функционал $\Phi(\mathbf{P})$ (см. пункт 3.4). Функционал $\Phi(\mathbf{P})$ вычисляется внутри процедуры *GeneralFit*.

Функциональные параметры *aF* и *aGradF* передают процедуре *GeneralFit* указатели на процедуры вычисления аппроксимирующей функции и ее параметрического градиента. Пользователь должен самостоятельно запрограммировать эти процедуры.

В качестве примера можно использовать модуль *uFuncDef* (см. файл *uFuncDef.pas*), где содержится код вычисления аппроксимирующей функции и ее параметрического градиента описанных в пункте 3.3.

Процедуры должны иметь заголовки следующего типа:

tFunc = function(const P:tParameters; X:Double):Double;

При вызове процедура типа *tFunc* должна вычислить и вернуть значение аппроксимирующей функции в точке *X* при значении параметров *P*.

tGradF = procedure(const P:tParameters; X:Double; var GradF:tParameters);

При вызове процедура типа *tGradF* должна вычислить и вернуть в переменной *GradF* значение параметрического градиента аппроксимирующей функции в точке *X* при значении параметров *P*. В элементе массива

данных структуры *GradF* должна возвращаться производная по соответствующему ему параметру. Порядок следования параметров внутри структуры несущественен для работы процедуры *GeneralFit*, но пользовательские функции не должны менять его в процессе вычислений.

4.5. Порядок работы с процедурой *GeneralFit*

При использовании процедуры *GeneralFit* следует придерживаться следующего порядка:

1) Запрограммировать и протестировать процедуры вычисления аппроксимирующей функции и ее параметрического градиента. Следует позаботиться чтобы при вычислении не возникало переполнения или деления на ноль в особых точках функции. Для этого в особых точках следует возвращать числовой результат. Например, в особой точке, где функция стремится к бесконечности, следует возвращать максимальное число для математического сопроцессора (максимальное значение для типа *Double* равно $\sim 10^{300}$).
2) Разместить в памяти необходимые структуры данных нужного размера: 1) массивы данных *X, Y* и 2) массив параметров *P*. Рекомендуется динамическое размещение процедурами *AllocateD* и *AllocateP*.
3) Заполнить массивы данных *X, Y* значениями. Порядок следования точек в массиве несущественен, но x_i и y_i, относящиеся к одной точке должны находится в элементах массивов с одинаковым индексом i.
4) Рекомендуется нормализовать данные (см. пункт 4.3).
5) Заполнить массив параметров начальными приближениями. Здесь имеет смысл постараться выбрать наилучшее приближение — чем ближе исходная точка **P** к истинному значению $\mathbf{P_0}$ — тем быстрее сосчитает *GeneralFit*.
6) Вызвать процедуру *GeneralFit*, передав ей подготовленные аргументы.
7) После возврата из *GeneralFit* проверить код ошибки и, если он равен нулю, то в переменной *P* возвращены значения параметров.

ПРИЛОЖЕНИЕ 1

ВЫЧИСЛЕНИЕ ФУНКЦИИ *ERF*

Данный пример взят из библиотеки численных методов «Numerical Recipes»[9].

```
      FUNCTION erf(x)
      REAL erf,x
CU    USES gammp
      REAL gammp
       if(x.lt.0.)then
          erf=-gammp(.5,x**2)
       else
          erf=gammp(.5,x**2)
      endif
       return
       END
```

C (C) Copr. 1986-92 Numerical Recipes Software .5-28.

[9]) Библиотека «Numerical Recipes» (© Copr. 1986-92 Numerical Recipes Software .5-28.) содержит исходный код большого числа процедур, реализующих различные численные методы вычислений, код представлен на ФОРТРАНЕ 77 и СИ.

ВЫЧИСЛЕНИЕ ВСПОМОГАТЕЛЬНЫХ ФУНКЦИЙ

Данный пример взят из библиотеки численных методов «Numerical Recipes»[10].

```
      FUNCTION gammp(a,x)
      REAL a,gammp,x
CU    SES gcf,gser
      REAL gammcf,gamser,gln
      if(x.lt.0..or.a.le.0.)pause 'bad
arguments in gammp'
      if(x.lt.a+1.)then
        call gser(gamser,a,x,gln)
        gammp=gamser
      else
        call gcf(gammcf,a,x,gln)
        gammp=1.-gammcf
      endif
      return
      END
```

C (C) Copr. 1986-92 Numerical Recipes Software .5-28.

```
      SUBROUTINE gcf(gammcf,a,x,gln)
      INTEGER ITMAX
      REAL a,gammcf,gln,x,EPS,FPMIN
      PARAMETER
(ITMAX=100,EPS=3.e-7,FPMIN=1.e-30)
CU    USES gammln
      INTEGER i
      REAL an,b,c,d,del,h,gammln
      gln=gammln(a)
      b=x+1.-a
      c=1./FPMIN
      d=1./b
      h=d
      do 11 i=1,ITMAX
        an=-i*(i-a)
        b=b+2.
        d=an*d+b
        if(abs(d).lt.FPMIN)d=FPMIN
        c=b+an/c
        if(abs(c).lt.FPMIN)c=FPMIN
        d=1./d
        del=d*c
        h=h*del
        if(abs(del-1.).lt.EPS)goto 1
11    continue
```

[10]) Библиотека «Numerical Recipes» (© Copr. 1986-92 Numerical Recipes Software .5-28.) содержит исходный код большого числа процедур, реализующих различные численные методы вычислений, код представлен на ФОРТРАНЕ 77 и СИ.

```
      pause 'a too large, ITMAX too small
in gcf'
1     gammcf=exp(-x+a*log(x)-gln)*h
      return
      END
```



```
      SUBROUTINE gser(gamser,a,x,gln)
      INTEGER ITMAX
      REAL a,gamser,gln,x,EPS
      PARAMETER
(ITMAX=100,EPS=3.e-7)
CU    USES gammln
      INTEGER n
      REAL ap,del,sum,gammln
      gln=gammln(a)
      if(x.le.0.)then
        if(x.lt.0.)pause 'x < 0 in gser'
        gamser=0.
        return
      endif
      ap=a
      sum=1./a
      del=sum
      do 11 n=1,ITMAX
        ap=ap+1.
        del=del*x/ap
        sum=sum+del
        if(abs(del).lt.abs(sum)*EPS)goto 1
11    continue
      pause 'a too large, ITMAX too small
in gser'
1     gamser=sum*exp(-x+a*log(x)-gln)
      return
      END
```



```
      FUNCTION gammln(xx)
      REAL gammln,xx
      INTEGER j
      DOUBLE PRECISION
ser,stp,tmp,x,y,cof(6)
      SAVE cof,stp
      DATA cof,stp/76.18009172947146d0,-
86.50532032941677d0,
     *24.01409824083091d0,-
1.231739572450155d0,.1208650973866179d-
2,
     *-.5395239384953d-
5,2.5066282746310005d0/
      x=xx
      y=x
      tmp=x+5.5d0
      tmp=(x+0.5d0)*log(tmp)-tmp
      ser=1.000000000190015d0
      do 11 j=1,6
      y=y+1.d0
      ser=ser+cof(j)/y
11    continue
      gammln=tmp+log(stp*ser/x)
      return
      END
```