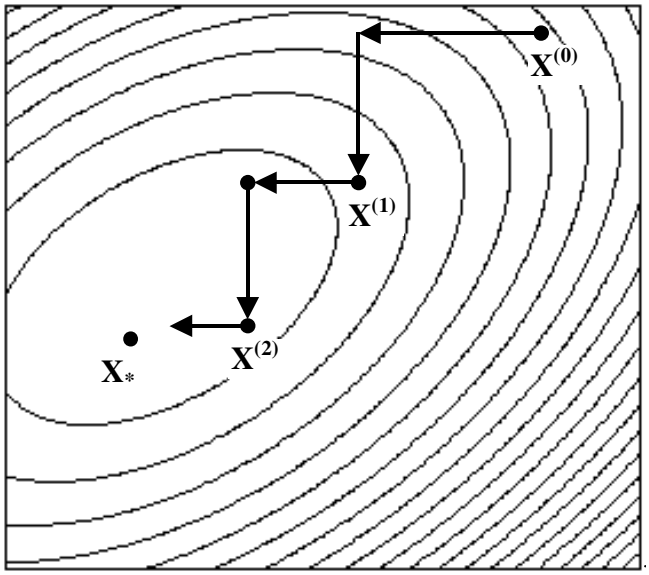
В.В. Зайцев, В.М. Трещев

ЧИСЛЕННЫЕ МЕТОДЫ ДЛЯ ФИЗИКОВ. НЕЛИНЕЙНЫЕ УРАВНЕНИЯ И ОПТИМИЗАЦИЯ



Самара
2006

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

В.В.Зайцев, В.М.Трещев

ЧИСЛЕННЫЕ МЕТОДЫ ДЛЯ ФИЗИКОВ. НЕЛИНЕЙНЫЕ УРАВНЕНИЯ И ОПТИМИЗАЦИЯ

Учебное пособие

Издательство «Самарский университет»

2006

УДК 519.6
ББК 22.19
З 12

Зайцев, В.В.

З 12 Численные методы для физиков. Нелинейные уравнения и оптимизация: учебное пособие / В.В.Зайцев, В.М.Трещев. – Самара: Изд-во «Самарский университет», 2006. – 86 с.: ил. – 250 экз.

ISBN 5-86465-240-7

Учебное пособие посвящено одному из разделов курса численных методов – решению нелинейных уравнений и оптимизации. Изложение проведено на «физическом» уровне строгости. Основное внимание уделено описанию численных алгоритмов и ограничениям и проблемам, возникающим при их применении.

Приведены примеры реализации численных алгоритмов с использованием пакета MathCAD.

УДК 519.6
ББК 22.19

Р е ц е н з е н т ы : докт. физ.-мат. наук, проф. Крутов А.Ф.,
докт. физ.-мат. наук, проф. Яровой Г.П.

ISBN 5-86465-240-7

ПРЕДИСЛОВИЕ

Учебное пособие написано на основе лекций, читаемых авторами студентам-физикам Самарского государственного университета, и посвящено одному из разделов курса численных методов – решению нелинейных уравнений и методам оптимизации.

В пособии приведены основные сведения о численных методах решения нелинейных уравнений и систем уравнений, методах минимизации функций одной и нескольких переменных, необходимые для практического использования этих методов при анализе нелинейных моделей систем различной физической природы. Рассмотрены как традиционные, классические численные методы, так и методы, вошедшие в вычислительную практику сравнительно недавно.

Изложение проводится на «физическом» уровне строгости. Математические обоснования большинства методов даны на основе элементарных результатов математического анализа и должны быть понятны студентам младших курсов. Основное внимание уделено практической стороне использования численных методов, а также ограничениям и проблемам, возникающим при их применении. Там, где это возможно, описание математических результатов сопровождается физической интерпретацией процессов, лежащих в основе численных алгоритмов.

Приведены многочисленные примеры реализации рассмотренных численных алгоритмов с использованием пакета MathCAD. Предполагается, что студенты прослушали курс программирования и имеют навыки составления программ. Детальный разбор программных модулей MathCAD, имеющих «прозрачную» структуру, часто способных заменить стандартные блок-схемы алгоритмов, рекомендуется для наиболее полного понимания сути изучаемых методов. Кроме того, использование MathCAD дает возможность студентам совершенствоваться в научном программировании.

ГЛАВА 1

ЧИСЛЕННОЕ РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

1.1. Введение

Необходимость решения нелинейных уравнений возникает при анализе очень многих физических систем. Например, положение равновесия материальной точки при ее перемещении вдоль оси Ox определяется из условия равенства нулю равнодействующей $F(x)$ приложенных сил, т.е. путем решения уравнения $F(x)=0$. При этом зависимость силы F от координаты x задается физической моделью системы.

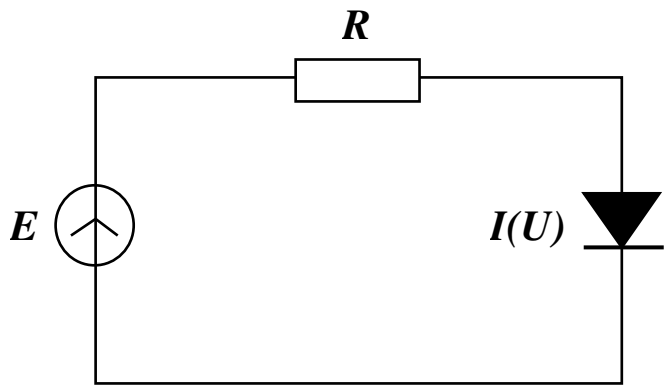


Рис. 1.1. Схема электрической цепи

Еще один пример из области электроники выглядит следующим образом. Пусть нелинейный двухполюсник с вольт-амперной характеристикой $I(U)$ включен в электрическую цепь последовательно с линейным резистором R и источником напряжения E по схеме, приведенной на рис. 1.1. Требуется определить ток в цепи и напряжения на ее элементах. Математическая запись второго закона Кирхгофа для рассматриваемой замкнутой цепи имеет вид

$$RI(U)+U=E.$$

Это уравнение относительно напряжения на нелинейном двухполюснике. Его можно записать в двух эквивалентных формах:

$$RI(U)+U-E=0 \text{ и } U=E-RI(U).$$

И ту, и другую мы будем использовать в дальнейшем, считая, тем не менее, первую основной.

Итак, общая форма записи нелинейного уравнения имеет вид

$$f(x)=0. \qquad (1.1)$$

Если функция $f(x)$ – полином, то уравнение называется *алгебраическим*. Если же левая часть (1.1) содержит тригонометрические функции, логарифм, экспоненту или специальные функции, то уравнение носит

название *трансцендентного*. В большинстве случаев алгебраические и трансцендентные уравнения решаются одними и теми же численными методами. Хотя для решения алгебраических уравнений разработаны и специальные методы, основанные на использовании свойств корней полиномов.

В подавляющем большинстве методы решения нелинейных уравнений являются итерационными, т.е. алгоритм решения заключается в многократном повторении некоторой вычислительной процедуры. Полученное таким образом решение всегда является приближенным, но может быть сделано сколь угодно близким к точному. Для начала расчетов итерационным методом требуется задание начального приближения корня или границ области его локализации. Обычно эта проблема решается на основе физических соображений или графически.

Отметим также, что в вычислительной практике блок решения нелинейных уравнений часто является составной частью более сложных алгоритмов.

1.2. Метод простой итерации

Рассмотрение методов численного решения нелинейных уравнений начнем с простейшего – метода простой итерации. Для его применения уравнение (1.1) представляется в виде

$$x = g(x),$$

где $g(x) = x + f(x)$, и итерации проводятся по формуле

$$x_{n+1} = g(x_n). \qquad (1.2)$$

Простота программной реализации алгоритма (1.2) является главным достоинством метода простой итерации.

На рис. 1.2 дана геометрическая интерпретация метода. В системе координат (x, y) построены графики функций $y = g(x)$ и $y = x$. Абсциссы точек пересечения этих линий соответствуют корням уравнения (1.1). Если имеется начальное приближение x_0, то на графике $y = g(x)$ легко находится точка $(x_0, g(x_0))$. Проведя через нее прямую $y = x_1$ до пересечения с прямой $y = x$, получим точку $(x_1, g(x_0))$. Абсцисса этой точки x_1 дает первое приближение к решению. Затем на графике $y = g(x)$ находится точка $(x_1, g(x_1))$, а на графике $y = x$ – точка $(x_2, g(x_1))$. Ее абсцисса является вторым приближением. Итерации продолжаются до тех пор, пока величина $| x_{n+1} - x_n |$ не достигнет заданной точности решения.

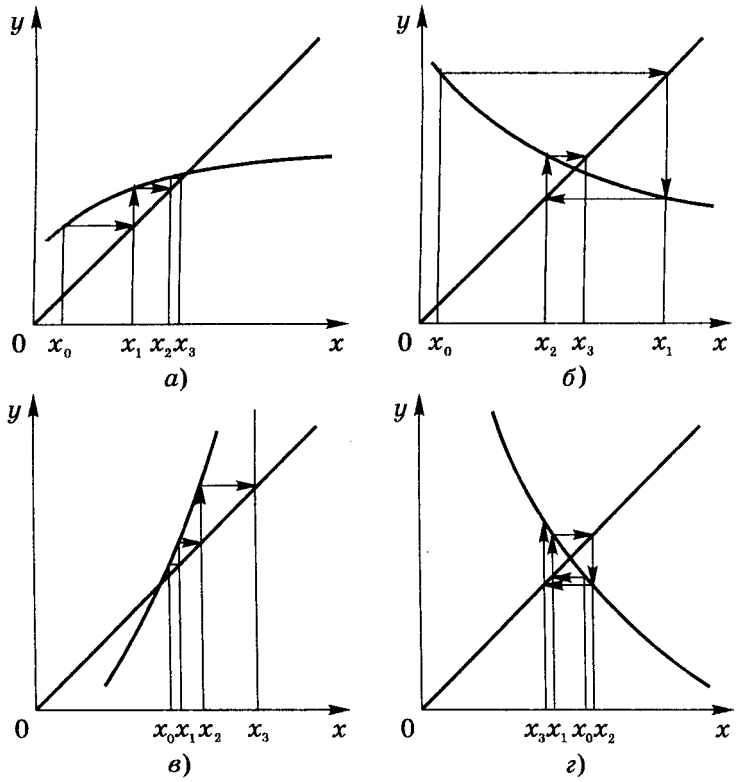


Рис. 1.2. Метод простой итерации

На рис. 1.2 изображено поведение последовательных приближений для случаев: $0 < g'(x) < 1$ (*а*), $-1 < g'(x) < 0$ (*б*), $g'(x) > 1$ (*в*) и $g'(x) < -1$ (*г*). Как видно из рисунков, сходимость последовательности x_n к истинному решению x_r имеет место лишь при выполнении условия $-1 < g'(x_r) < 1$ (рис. 1.2 *а*, *б*). Для функции $f(x)$ это условие имеет вид

$$-2 < f'(x_r) < 0. \qquad (1.3)$$

Если же $g'(x_r) > 1$ или $g'(x_r) < -1$, то итерационный процесс расходится (рис. 1.2 *в*, *г*).

Наличие жесткого условия сходимости – существенный недостаток метода. Для любой программы, в которой используется алгоритм простой итерации, необходимо предусматривать контроль сходимости и прекращать вычисления, если она не обеспечивается. При этом проверить выполнение неравенства (1.3) на практике не удается, так как истинное значение корня x_r или его достаточно точные оценки до проведения вычислений неизвестны. Поэтому используются косвенные критерии сходимости. Например, в блок-схеме алгоритма простой итерации, представленной на рис. 1.3, вычисления прекращаются, если число итераций превысило наперед заданное значение $I_{\max}$.

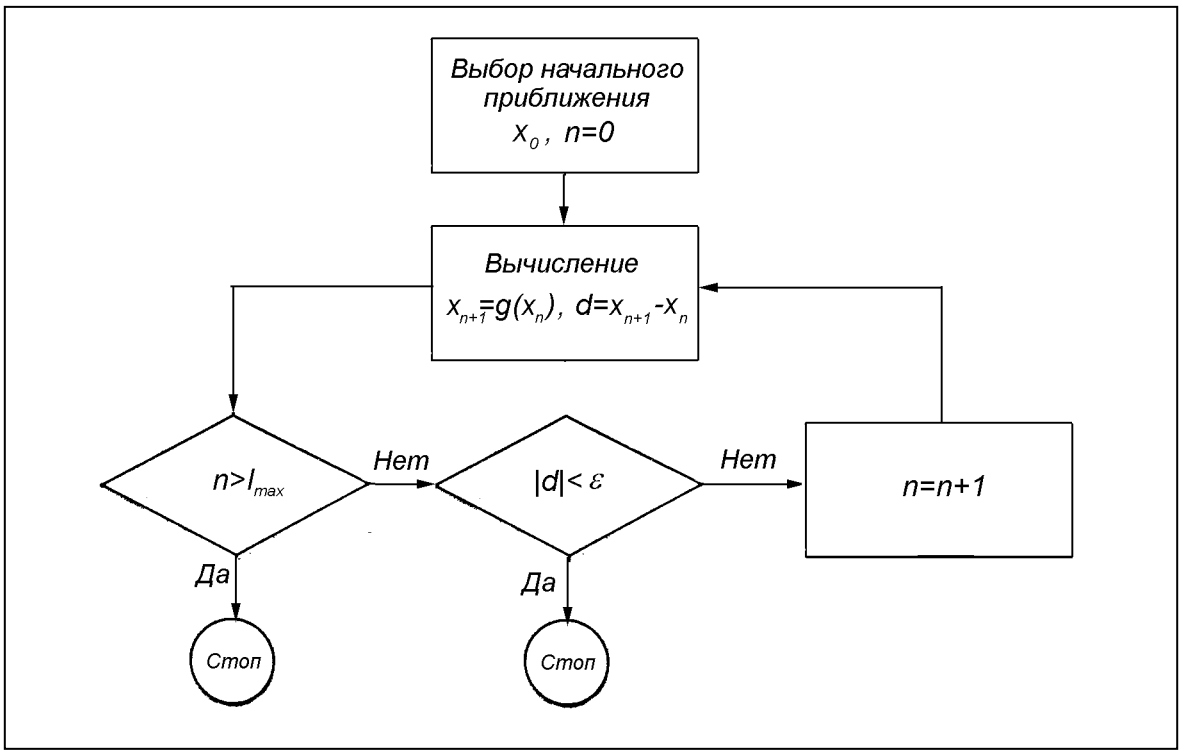


Рис. 1.3. Блок-схема алгоритма метода простой итерации

Как обобщение простой итерации может рассматриваться алгоритм, определяемый формулой

$$x_{n+1} = x_n + \lambda f(x_n), \tag{1.4}$$

где λ – константа, знак и абсолютная величина которой выбираются так, чтобы обеспечить сходимость итерационного процесса. Из сопоставления этой формулы с формулой (1.2) следует, что здесь $g(x) = x + \lambda f(x)$. Поэтому условие сходимости $-1 < g'(x_r) < 1$ для функции $f(x)$ сводится к неравенству

$$-2 < \lambda f'(x_r) < 0. \tag{1.5}$$

Ясно, что, имея возможность варьировать константу λ, теоретически мы всегда можем выполнить условие (1.5). В частности, можно было бы положить $\lambda = -1/f'(x_r)$, но практически это невыполнимо, так как значение корня неизвестно. Если считать, что приближение x_n близко к корню x_r и производные $f'(x_n)$ и $f'(x_r)$ различаются незначительно, то, положив $\lambda = -1/f'(x_n)$, мы получим итерационный алгоритм

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

известный как метод Ньютона. Но вопрос о сходимости алгоритма все же остается открытым.

1.3. Методы половинного деления и ложного положения

Эти методы гарантируют сходимость итерационного процесса, если найден интервал локализации корня, т.е. отрезок $[x_n, x_{n+1}]$ такой, что значения $f(x_n)$ и $f(x_{n+1})$ имеют разные знаки. Поиск интервала локализации, как правило, не вызывает затруднений и проводится путем вычисления значений функции $f(x_n)$ для последовательности значений аргумента $x_n = x_0 + nh$ до тех пор, пока не будут найдены две точки, удовлетворяющие условию $f(x_n)f(x_{n+1}) < 0$. Следует лишь отметить, что шаг поиска h не должен быть слишком велик, чтобы не пропустить два близко расположенных корня, если, конечно, таковые имеются. В то же время, шаг h не может быть и очень мал, иначе поиск займет слишком много времени. Начальная точка поиска x_0 часто выбирается, исходя из априорной информации о расположении корней уравнения.

Пусть интервал локализации корня найден. Обозначим его как $[a,b]$. Далее итерационный процесс метода половинного деления состоит из следующих шагов.

1. Вычисляется среднее значение $c = (a+b)/2$ аргумента функции $f(x)$ на отрезке $[a,b]$ и значение $f(c)$.

2. Если знак $f(c)$ совпадает со знаком $f(a)$, то левая граница интервала локализации корня смещается в точку c, т.е. значению a присваивается значение c, а значению $f(a)$ – значение $f(c)$, и поиск следующего приближения продолжается с шага 1.

3. Если же знак $f(c)$ не совпадает со знаком $f(a)$, то в точку c смещается правая граница интервала локализации корня, т.е. значению b присваивается значение c, а значению $f(b)$ – значение $f(c)$, после чего следует возврат к шагу 1.

Итерационный процесс продолжается до тех пор, пока вычисляемая на каждой итерации ширина интервала локализации корня не уменьшится до заданной величины погрешности решения или текущее значение $|f(c)|$ не станет меньше заданной малой величины ε. В качестве приближенного значения корня берется последнее вычисленное значение c. Блок-схема алгоритма половинного деления с первоначально заданным интервалом локализации корня и выходом из итераций по условию $|f(c)| < \varepsilon$ приведена на рис. 1.4.

Гарантируя сходимость последовательности приближений к истинному решению, метод половинного деления не обладает высокой вычислительной эффективностью. Действительно, ширина интервал локализации корня после N итераций уменьшается в 2^N раз. Следовательно, для уточнения одного десятичного знака в решении

требуется более трех итераций. Это очень невысокая скорость сходимости, так как, например, более эффективный метод Ньютона в окрестности корня позволяет за одну итерацию удваивать число верных десятичных знаков.

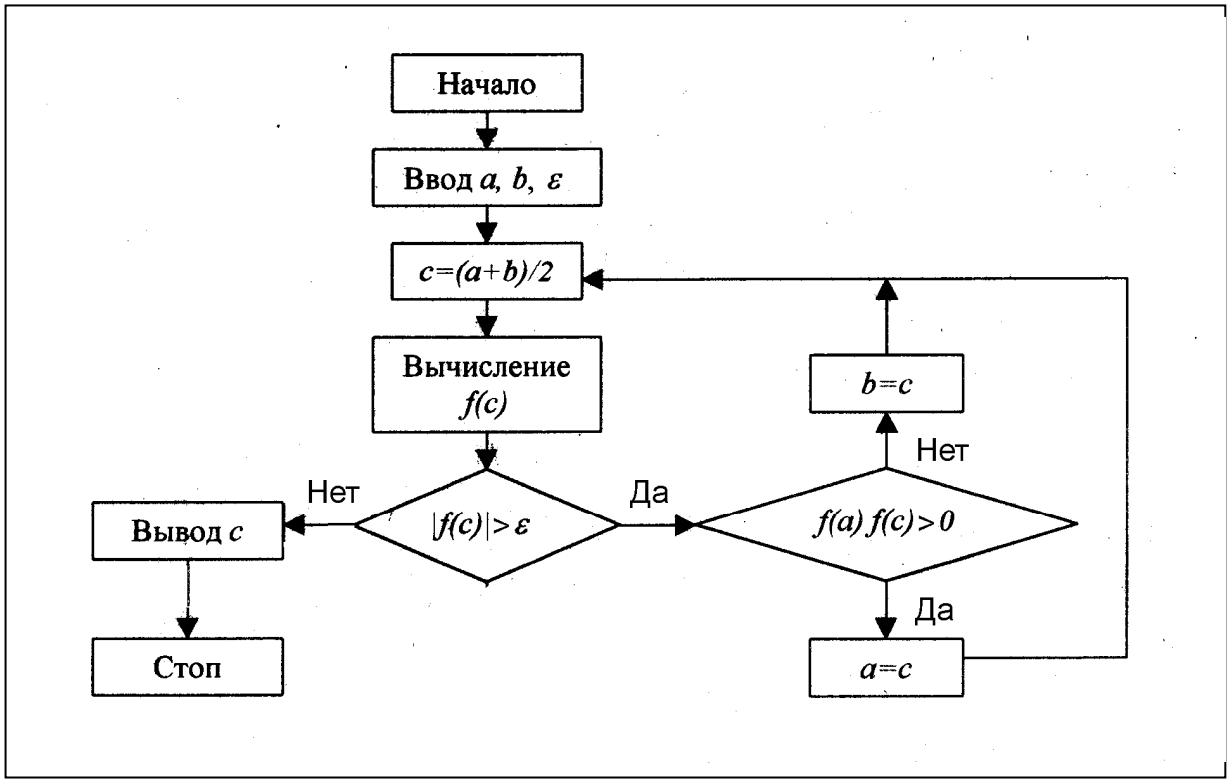


Рис. 1.4. Блок-схема алгоритма метода половинного деления

Поскольку скорость сходимости итерационного процесса к корню уравнения играет очень важную роль при сравнении эффективности различных методов, дадим ее количественное определение. В общем случае в ходе итерационного процесса погрешности решения на двух последовательных итерациях связаны неравенством

$$\left|x_{n+1} - x_r\right| \le C\left|x_n - x_r\right|^{\alpha}, \qquad (1.6)$$

где C – некоторая константа. Показатель степени α в этом неравенстве и есть скорость сходимости. Если $\alpha = 1$, то сходимость называется *линейной*; если $1 < \alpha < 2$ – *сверхлинейной*; если $\alpha = 2$ – *квадратичной*. Чаще всего скорость сходимости α более важна, чем константа C в неравенстве (1.6). Тем не менее, константа C также может играть важную роль. В частности, из двух алгоритмов с одинаковыми скоростями α быстрее сходится тот, что характеризуется меньшим значением C. Далее, линейно сходящийся метод с $C \approx 0$ вначале может сходиться быстрее, чем квадратично сходящийся метод с большой константой C. Таким образом, хотя большие

значения α в конечном счете обеспечивают быструю сходимость, линейная скорость может быть вполне приемлемой, если константа C мала. Но если константа C близка к единице, то линейная сходимость является недопустимо медленной.

Метод половинного деления имеет линейную скорость сходимости и значение $C \approx 0.5$.

Несколько ускорить сходимость метода половинного деления, уменьшив константу C, можно, если в качестве приближения к решению брать не середину отрезка $[a,b]$, а точку нуля линейной функции, интерполирующей $f(x)$ по узлам a и b, как это показано на рис. 1.5. Из рисунка видно, что такая точка может дать лучшее приближение к решению, чем середина отрезка $[a,b]$.

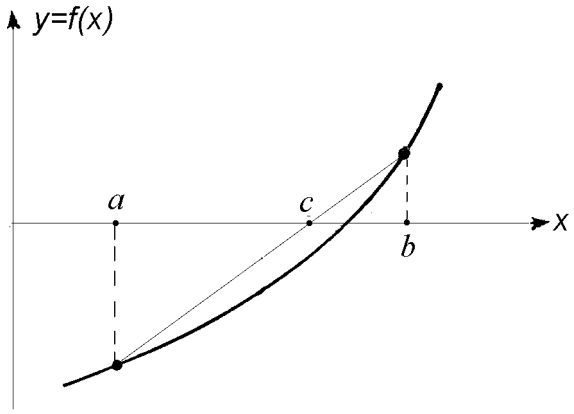


Рис. 1.5. Метод ложного положения

Интерполяционный полином первой степени для функции $f(x)$ имеет вид

$$P_1(x) = \frac{x-b}{a-b} f(a) + \frac{x-a}{b-a} f(b).$$

Его корень, как нетрудно установить, вычисляется по формуле

$$c = a - f(a)\frac{b-a}{f(b)-f(a)}. \qquad (1.7)$$

Именно эта формула используется на первом шаге итерационного процесса в методе ложного положения. Остальные шаги те же, что и в методе половинного деления. Блок-схема алгоритма ложного положения имеет тот же вид, что и схема, приведенная на рис. 1.4. В последней следует лишь заменить блок вычисления средней точки на блок вычислений по формуле (1.7).

Отметим, что метод половинного деления имеет второе название – *метод бисекции*, а метод ложного положения – *метод хорд*.

Методы половинного деления и ложного положения помимо медленной сходимости обладают и другими недостатками. Очевидно, что они не позволяют найти корни, в которых функция $f(x)$ не пересекает ось x, а лишь касается ее. Напротив, этими методами вместе с корнями определяются и точки разрыва функции, если при переходе через них функция меняет знак. В этом случае программа с выходом из итерационного цикла по условию $|f(c)|<\varepsilon$ должна предусматривать прекращение вычислений при превышении числом итераций максимально допустимого значения $I_{\max}$. Если же в программе используется условие уменьшения ширины интервала локализации корня до заданной погрешности решения, то после окончания итераций следует убедиться в малости значения $|f(c)|$. В противном случае найденное значение c следует считать точкой разрыва функции.

Тем не менее, гарантированная сходимость обеспечивает методам половинного деления и ложного положения достаточно широкую область применения. Например, многие программы решения нелинейных уравнений начинают вычисления методом половинного деления, а затем, когда корень локализован в достаточно узкой области, уточняют его положение быстро сходящимся методом.

1.4. Метод Ньютона и метод секущих

Метод Ньютона является, пожалуй, самым популярным методом решения нелинейных уравнений. В нем для вычисления каждого следующего приближения к корню используется экстраполяция функции с помощью касательной к кривой в текущей точке.

Пусть x_n – текущее приближение к корню x_r и $\Delta x = x_r - x_n$. Тогда можно записать следующее разложение функции $f(x)$ в ряд Тейлора:

$$f(x_r) = f(x_n + \Delta x) = f(x_n) + f'(x_n)\Delta x + \frac{1}{2} f''(x_n)\Delta x^2 + \dots . \quad (1.8)$$

С учетом того, что $f(x_r) = 0$, получим

$$f(x_n) + f'(x_n)\Delta x + \frac{1}{2} f''(x_n)\Delta x^2 + \dots = 0 . \quad (1.9)$$

Выражение (1.9) представляет собой одну из форм записи уравнения (1.1). Она удобна тем, что можно находить приближенные решения, ограничиваясь конечным числом слагаемых в левой части (1.9). С учетом двух слагаемых находим приближение вида

$$\Delta x^{(1)} = x_r^{(1)} - x_n = -\frac{f(x_n)}{f'(x_n)}.$$

Если теперь точку $x_r^{(1)}$ взять в качестве следующего за x_n уточнения корня, то получим итерационную формулу метода Ньютона:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \qquad (1.10)$$

Аналогичная формула была получена Рафсоном, но чуть позже Ньютона – в 1690 году. Поэтому метод часто называется методом *Ньютона–Рафсона*.

Итерационный процесс по формуле (1.10) продолжается до тех пор, пока разность $|x_{n+1} - x_n|$ не достигнет заданной погрешности решения или значение $|f(x_{n+1})|$ не уменьшится до заданной величины. Блок-схема алгоритма метода Ньютона имеет тот же вид, что и приведенная на рис. 1.3 блок-схема алгоритма метода простой итерации. В ней лишь следует считать, что

$$g(x_n) = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Нетрудно показать, что геометрически x_{n+1} интерпретируется как точка пересечения оси x касательной к кривой $y = f(x)$ в точке x_n (рис. 1.6). Отсюда и второе название метода Ньютона – метод касательных.

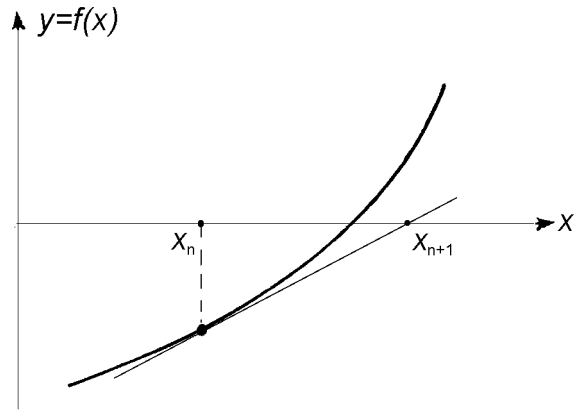


Рис. 1.6. Метод Ньютона

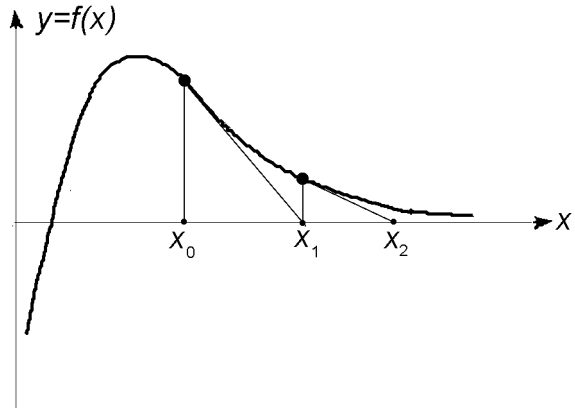


Рис. 1.7. Расходимость метода Ньютона

Совершенно очевидно, что сходимость метода в большой мере зависит от удачного выбора начального приближения x_0. Рис. 1.7 отражает ситуацию, когда итерации по методу Ньютона уводят от корня уравнения. Можно представить и другие ситуации, например, осцилляции последовательных приближений в окрестности точки, не являющейся корнем.

Тем не менее, если начальное приближение выбрано в ближайшей окрестности корня, то метод обеспечивает квадратичную сходимость итерационного процесса. Именно это свойство придает методу Ньютона особое значение.

Наличие у метода квадратичной сходимости можно установить следующим образом. Вновь воспользуемся разложением функции $f(x)$ в ряд Тейлора в окрестности точки x_n:

$$f(x) = f(x_n) + f'(x_n)(x - x_n) + \frac{1}{2} f''(x_*)(x - x_n)^2 .$$

Это разложение, в отличие от (1.8), является точным, т.к. в нем используется некоторая точка x_*, про которую известно лишь то, что она расположена между x и x_n. Положив в нем $x = x_r$, получим

$$f(x_n) + f'(x_n)(x - x_n) + \frac{1}{2} f''(x_*)(x - x_n)^2 = 0 .$$

Разделим теперь это равенство на $f'(x_n)$ и преобразуем результат к виду

$$x_r - \left(x_n - \frac{f(x_n)}{f'(x_n)} \right) = -\frac{f''(x_*)}{2f'(x_n)} (x_r - x_n)^2 .$$

Левая часть полученного выражения, согласно итерационной формуле метода (1.10), есть $x_r - x_{n+1}$. Тогда мы можем записать

$$\left| x_r - x_{n+1} \right| = C \left| x_r - x_n \right|^2 ,$$

где $C = \left| f''(x_*) \right| / 2 \left| f'(x_n) \right|$. А это равенство совершенно определенно указывает на квадратичную сходимость итерационного процесса (1.10).

Однако необходимо уточнение: метод Ньютона имеет квадратичную скорость сходимости в окрестности простого (однократного) корня. Если корень кратный, то метод Ньютона сходится гораздо медленнее – линейно.

Помимо возможных затруднений с выбором начального приближения у метода Ньютона есть еще один недостаток – необходимость задания производной $f'(x)$ в аналитической форме. Если функция $f(x)$ сложна, то аналитическое дифференцирование сопряжено со значительными техническими трудностями. Еще большие проблемы возникают, когда $f(x)$ вообще не задана аналитически, например, ее значения являются результатом вычислений по некоторому алгоритму. Этот недостаток ограничивает область практического применения метода Ньютона. Один из способов его преодоления состоит в аппроксимации производной $f'(x_n)$ конечными разностями. Например, можно использовать аппроксимацию центральной разностью:

$$f'(x_n) = \frac{f(x_n + h) - f(x_n - h)}{2h} + O(h^2), \qquad (1.11)$$

которая имеет второй порядок точности по приращению аргумента h. Построенный таким способом метод будет вести себя почти так же, как и метод с точными значениями производной, но при этом потребуются дополнительные вычисления функции, что снижает вычислительную эффективность алгоритма.

Дополнительных вычислений функции можно избежать, если при аппроксимации производной использовать ранее вычисленные значения. При таком подходе $f'(x_n)$ в формуле (1.10) заменяется приближенным выражением

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}, \qquad (1.12)$$

и мы приходим к итерационной формуле

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}, \qquad (1.13)$$

составляющей основу *метода секущих*. Геометрическая интерпретация одного шага итераций дана на рис. 1.8. Как и в методе Ньютона, вычисления заканчиваются, если последовательные значения x_n и x_{n+1} совпадают с заданной точностью или если достигнуто достаточно близкое к нулю значение $f(x_{n+1})$.

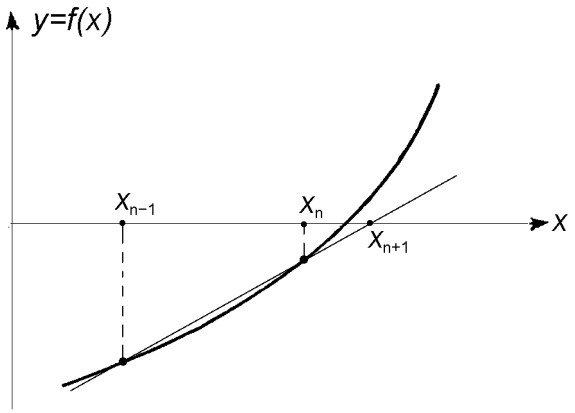


Рис. 1.8. Метод секущих

Проблема выбора начального приближения для метода секущих столь же актуальна, что и для метода Ньютона. Причем, в отличие от метода Ньютона (1.10), использующего для вычисления x_{n+1} только одно предыдущее приближение x_n и поэтому являющегося одношаговым, метод секущих, как это следует из формулы (1.13), – двухшаговый метод. Для начала поиска корня теперь требуется уже не одна начальная точка x_0, а две – x_0 и x_1. Однако это усложнение задачи не является существенным. Как правило, если выбор точки x_0 обеспечивает сходимость метода Ньютона, то за x_1 можно взять любую близкую точку.

Скорость сходимости итерационного процесса (1.13) ниже, чем у итераций по методу Ньютона. Показано, что метод секущих сходится к простому корню сверхлинейно со скоростью $\alpha = (1+\sqrt{5})/2 = 1.618$. Тем не менее, эта скорость достаточно высока для того, чтобы обеспечить методу широкую область применения. Современные компьютерные программы часто основаны на полиалгоритмах, представляющих собой комбинации двух или более методов. Например, вначале используется метод половинного деления, с помощью которого вычисления ведутся до тех пор, пока приближение не окажется настолько близко к корню, что будет сходиться метод секущих. После этого для ускорения сходимости вычисления продолжаются методом секущих.

1.5. Метод Стеффенсена и ускорение сходимости Эйткена

Скорость сходимости метода секущих можно повысить, если вместо аппроксимации производной $f'(x_n)$ выражением (1.12) использовать выражение

$$f'(x_n) \approx \frac{f(x_{n+1}) - f(x_n)}{x_{n+1} - x_n}. \qquad (1.14)$$

Его можно интерпретировать как аппроксимацию правой разностью, если формально считать, что в (1.12) используется левая разность.

В выражение (1.14) входит еще не найденное значение x_{n+1}. Вычислим его, применив алгоритм простой итерации (1.2):

$$x_{n+1} = g(x_n) = x_n + f(x_n).$$

В результате мы получим аппроксимацию

$$f'(x_n) \approx \frac{f\big(x_n + f(x_n)\big) - f\big(x_n\big)}{f\big(x_n\big)},$$

использование которой в методе Ньютона (1.10) приводит к новому итерационному алгоритму

$$x_{n+1} = x_n - \frac{f(x_n)}{f\left(x_n + f(x_n)\right) - f\left(x_n\right)} f(x_n), \qquad (1.15)$$

известному в численном анализе как *метод Стеффенсена*.

Метод Стеффенсена имеет квадратичную сходимость, но высокая скорость сходимости достигается за счет дополнительного вычисления значения $f\left(x_n + f(x_n)\right)$. На каждой итерации метод требует двух вычислений функции. По этому критерию метод Стеффенсена менее эффективен, чем метод секущих.

Итерационный алгоритм (1.15) можно получить также в рамках предложенного Эйткеном способа ускорения сходимости линейно сходящихся последовательностей.

Рассмотрим последовательность

$$z_n = z + Cq^n. \qquad (1.16)$$

При $|q|<1$ эта последовательность сходится к пределу z. Нетрудно построить преобразование, которое позволяло бы выразить предельное значение z через три элемента последовательности z_{n-1}, z_n и z_{n+1}. Действительно, два очевидных равенства,

$$\frac{z_n - z}{z_{n-1} - z} = q \quad \text{и} \quad \frac{z_{n+1} - z}{z_n - z} = q,$$

приводят к равенству $(z_{n+1} - z)(z_{n-1} - z) = (z_n - z)^2$, из которого следует

$$z = \frac{z_{n+1} z_{n-1} - z_n^2}{z_{n+1} - 2z_n + z_{n-1}}.$$

В соответствии с этим результатом рассмотрим предложенное Эйткеном преобразование произвольной последовательности z_n в другую последовательность

$$\xi_{n+1} = \frac{z_{n+1} z_{n-1} - z_n^2}{z_{n+1} - 2z_n + z_{n-1}}. \qquad (1.17)$$

Если применить это преобразование к любой последовательности вида (1.16), то при любом значении n будет иметь место равенство $\xi_n = z = \lim_{n\to\infty} z_n$. Можно ожидать, что, если последовательность x_n будет иметь близкий к (1.16) тип сходимости, то преобразование (1.17), хотя и не будет при любом n давать ее предел, но все же приведет к новой последовательности, сходящейся к z быстрее, чем исходная.

Для примера применения ускорения Эйткена рассмотрим последовательность приближений к двукратному корню $x_r = 2$ уравнения $x^3 - x^2 - 8x + 12 = 0$, полученную методом Ньютона. Значения элементов последовательности размещены во втором столбце таблицы 1.1.

Таблица 1.1

n	x_n	$\lvert x_n - x_r \rvert / \lvert x_{n-1} - x_r \rvert$	ξ_n	$\lvert \xi_n - x_r \rvert / \lvert \xi_{n-1} - x_r \rvert$
0	0.5	-	-	-
1	1.454545	0.363636	-	-
2	1.745059	0.467391	1.872159	-
3	1.876049	0.486197	1.983607	0.128232
4	1.938822	0.493563	1.996588	0.208141
5	1.969602	0.496884	1.999213	0.230676
6	1.984847	0.498466	1.999811	0.240656
7	1.992435	0.499239	1.999954	0.245400
8	1.996221	0.499621	1.999988	0.247717
9	1.998111	0.499811	1.999997	0.248863
10	1.999056	0.499905	1.999999	0.249432
11	1.999528	0.499953	2.000000	0.249717
12	1.999764	0.499976	2.000000	0.249856
13	1.999882	0.499988	2.000000	0.249948
14	1.999941	0.499994	2.000000	0.250448

Далее, в третьем столбце таблицы приведены значения константы C в неравенстве (1.6), рассчитанные для каждой итерации в предположении о сходимости со скоростью $\alpha = 1$. Как видно из приведенных результатов, константа C мало меняется в ходе итерационного процесса и очень близка к значению $C = 0.5$. Следовательно, предположение о линейной сходимости метода Ньютона к двукратному корню справедливо.

Применив к линейно сходящейся последовательности x_n формулу ускорения (1.17), получим значения ξ_n, помещенные в четвертый столбец таблицы 1.1. Из сопоставления второго и четвертого столбцов таблицы следует вполне однозначный вывод о том, что ускорение сходимости достигнуто. Действительно, дойдя, например, до седьмой итерации метода Ньютона и применив ускорение Эйткена, мы получим примерно тот же результат, что и после четырнадцати итераций метода Ньютона. Пятый столбец таблицы 1.1 указывает на то, что в данном примере ускорение сходимости получено не за счет роста показателя скорости α (он по-прежнему равен единице), а за счет уменьшения константы C до значения $C = 0.25$.

Проанализируем теперь возможность ускорения сходимости последовательности приближений к корню в методе простой итерации. Используя разложение в ряд Тейлора, правую часть итерационной формулы $x_{n+1} = g(x_n)$ можно представить в виде

$$g(x_n) = g\big(x_r + (x_n - x_r)\big) = x_r + g'(x_r)(x_n - x_r) + O\big((x_n - x_r)^2\big),$$

после чего формула записывается как

$$x_{n+1} - x_r = g'(x_r)(x_n - x_r) + O\big((x_n - x_r)^2\big).$$

Таким образом, с точностью до квадрата ошибки $e_n = x_n - x_r$ на каждой итерации можно записать приближенное равенство

$$x_{n+1} - x_r \approx g'(x_r)(x_n - x_r),$$

из которого следует, что последовательность x_n задается формулой

$$x_n \approx x_r + [g'(x_r)]^n (x_0 - x_r)$$

и имеет почти тот же тип сходимости, что и последовательность (1.16). Значит, последовательность приближений к корню в методе простой итерации вполне подходит для применения процедуры ускорения сходимости.

При применении процедуры ускорения целесообразно каждое улучшенное значение сразу же вводить в расчеты, чтобы в последующих вычислениях оно уже было учтено. На каждом шаге итерации это выглядит следующим образом. Пусть вычисления доведены до значения x_n; по нему вычисляем два вспомогательных значения $x_n^{(1)} = g(x_n)$ и $x_n^{(2)} = g\big(g(x_n)\big)$. К трем значениям x_n, $x_n^{(1)}$ и $x_n^{(2)}$ применяем формулу ускорения (1.17) и результат принимаем за следующее приближение x_{n+1}:

$$x_{n+1} = \frac{x_n g\big(g(x_n)\big) - g^2(x_n)}{g\big(g(x_n)\big) - 2g(x_n) + x_n}. \qquad (1.18)$$

Нетрудно показать, что полученное равенство есть просто одна из форм записи итерационной формулы Стеффенсена (1.15).

Формулу (1.18) можно применить и к решению задачи о поиске двукратного корня уравнения $x^3 - x^2 - 8x + 12 = 0$. Положив $g(x) = x - f(x)/f'(x)$, что соответствует итерациям Ньютона, в результате расчетов по формуле (1.18) получим последовательность {0.5; 1.87215909; 1.99916211; 1.99999996; 2.00000000}. Сравнив ее с последовательностью ξ_n из четвертого столбца таблицы 1.1, видим, что эффект возрастает, если ускорение вводить не в последовательность, а в алгоритм с помощью которого она получена.

1.6. Метод обратной квадратичной интерполяции

Как следует из названия, в этом методе для определения нового приближения к корню уравнения $f(x)=0$ используется квадратичная интерполяция функции $x=g(y)$, обратной по отношению к функции $y=f(x)$. Так как для интерполяции полиномом второй степени необходимы три точки на плоскости (x,y), то выберем их из трех последовательных приближений к корню:

$$(x_{k-2}, y_{k-2}=f(x_{k-2})),\ (x_{k-1}, y_{k-1}=f(x_{k-1}))\ \text{и}\ (x_k, y_k=f(x_k)).$$

В таком случае интерполяционный полином Лагранжа записывается в виде

$$P_2(y)=\frac{x_k(y-y_{k-1})(y-y_{k-2})}{(y_k-y_{k-1})(y_k-y_{k-2})}+\frac{x_{k-1}(y-y_k)(y-y_{k-2})}{(y_{k-1}-y_k)(y_{k-1}-y_{k-2})}+ \\ +\frac{x_{k-2}(y-y_k)(y-y_{k-1})}{(y_{k-2}-y_k)(y_{k-2}-y_{k-1})}. \qquad (1.19)$$

Дальнейшее использование этого полинома основано на следующих соображениях. Если бы была известна функция $x=g(y)$, то нахождение корня свелось бы к простому вычислению

$$x_r=g(0).$$

Но задача поиска обратной функции почти всегда более сложна, чем решение уравнения. Можно лишь предложить замену $g(y)$ полиномом $P_2(y)$ в окрестности корня. Тогда есть основания ожидать, что, вычислив значение $P_2(0)$, мы получим новое приближение к корню исходного уравнения: $x_{k+1}=P_2(0)$.

Таким образом, в соответствии с выражением (1.19), итерации в методе обратной квадратичной интерполяции следует проводить по формуле

$$x_{k+1}=\frac{y_{k-1}y_{k-2}x_k}{(y_k-y_{k-1})(y_k-y_{k-2})}+\frac{y_k y_{k-2}x_{k-1}}{(y_{k-1}-y_k)(y_{k-1}-y_{k-2})}+ \\ +\frac{y_k y_{k-1}x_{k-2}}{(y_{k-2}-y_k)(y_{k-2}-y_{k-1})}. \qquad (1.20)$$

Скорость сходимости итерационного процесса в методе обратной квадратичной интерполяции равна $\alpha=1.839$, что несколько выше, чем в методе секущих. Однако для начала расчетов необходимо выбрать три начальных значения x_0, x_1, x_2, и если они выбраны неудачно, то поведение алгоритма может оказаться хаотическим.

Квадратичная интерполяция используется также в *методе Мюллера*. Но при этом интерполируется не обратная функция $x=g(y)$, а прямая –

$y = f(x)$. Затем корень интерполяционного полинома считается новым приближением к корню исходного уравнения. Но, т.к. полином второй степени имеет два корня, приходится решать проблему выбора одного из них. Определенные трудности возникают и при появлении у полинома комплексных корней. Поэтому метод Мюллера для однократных корней, по-видимому, не имеет преимуществ перед методом обратной квадратичной интерполяции. Он может применяться для поиска двукратных корней, в которых функция $y = f(x)$ не пересекает ось x, а лишь касается ее.

1.7. Численные методы поиска комплексных корней полиномов

Нелинейные уравнения, содержащие только суммы целых степеней аргумента x, называются нелинейными алгебраическими уравнениями. В общем виде их можно записать как

$$P_n(x) = x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0 = 0. \qquad (1.21)$$

Решения нелинейных алгебраических уравнений называют также корнями полиномов. При отыскании корней полезно иметь в виду, что полином степени n имеет n корней (с учетом кратности), которые могут быть действительными или комплексными. Если все коэффициенты полинома a_i действительны, то комплексные корни образуют комплексно-сопряженные пары. Информация о числе корней весьма полезна при численных решениях.

Общие алгоритмы решения нелинейных уравнений, описанные выше, можно использовать и для нахождения корней полиномов. При этом практическую реализацию алгоритмов следует проводить в рамках арифметики комплексных чисел. Тем не менее, существует ряд специальных методов для отыскания всех (действительных и комплексных) корней полиномов. Рассмотрим некоторые из них.

Большинство методов основано на процедуре выделения из исходного полинома квадратичного множителя:

$$P_n(x) = (x^2 + px + q)(x^{n-2} + b_{n-3}x^{n-3} + ... + b_1 x + b_0). \qquad (1.22)$$

А так как корни квадратичного полинома $x^2 + px + q = 0$ находятся аналитически, то порядок уравнения (1.21) понижается на два, и новым уравнением, подлежащим решению, будет уравнение

$$x^{n-2} + b_{n-3}x^{n-3} + ... + b_1 x + b_0 = 0,$$

из левой части которого снова выделяется квадратичный множитель. Процесс выделения множителей заканчивается тогда, когда остается квадратное уравнение $x^2 + b_1 x + b_0 = 0$ либо линейное уравнение $x + b_0 = 0$.

Задача о выделении квадратичного множителя, т.е. об отыскании n коэффициентов $p, q, b_{n-3}, b_{n-2}, ..., b_1, b_0$, решается следующим образом. Приравнивая коэффициенты при одинаковых степенях x в двух формах (1.21) и (1.22) полинома $P_n(x)$, получим две системы равенств:

$$\begin{aligned} & b_{n-3} + p = a_{n-1}, \\ & b_{n-4} + pb_{n-3} + q = a_{n-2}, \\ & b_{n-5} + pb_{n-4} + qb_{n-3} = a_{n-3}, \\ & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ & b_1 + pb_2 + qb_3 = a_3, \\ & b_0 + pb_1 + qb_2 = a_2 \end{aligned} \tag{1.23}$$

$$\text{и} \quad q = \frac{a_0}{b_0}, \; p = \frac{a_1 - qb_1}{b_0}. \tag{1.24}$$

Если задать произвольные значения коэффициентов p и q, то путем последовательного исключения из системы (1.23) неизвестных коэффициентов $b_{n-3}, b_{n-2}, ..., b_2$ можно рассчитать значения b_1 и b_0, т.е. равенства (1.23) определяют две функции двух аргументов:

$$b_0 = b_0(p, q) \text{ и } b_1 = b_1(p, q). \tag{1.25}$$

Теперь функции (1.25) следует использовать в правых частях равенств (1.24), сформировав таким образом систему двух нелинейных уравнений

$$\begin{aligned} q &= \frac{a_0}{b_0(p, q)}, \\ p &= \frac{a_1 - qb_1(p, q)}{b_0(p, q)}. \end{aligned} \tag{1.26}$$

Заметим, что в системе (1.26) правые части уравнений заданы не в аналитической форме, а в форме алгоритма расчета значений b_1 и b_0 для каждой пары значений аргументов (p, q). Подобная ситуация часто встречается в вычислительной практике. Отсутствие аналитической формы записи уравнений не является препятствием для их решения

Если система уравнений (1.26) решается с помощью алгоритма простой итерации[1], то численный метод для исходного уравнения (1.21) называется *методом Лина*. При этом итерации от начального приближения (p_0, q_0) проводятся по формулам

[1] Численные методы решения систем нелинейных уравнений подробно рассмотрены в следующем разделе.

$$q_{k+1} = \frac{a_0}{b_0(p_k, q_k)}, \quad p_{k+1} = \frac{a_1 - q_k b_1(p_k, q_k)}{b_0(p_k, q_k)}. \tag{1.27}$$

Вместо простой итерации можно использовать алгоритм Зейделя. Тогда вторая из итерационных формул (1.27) будет выглядеть следующим образом:

$$p_{k+1} = \frac{a_1 - q_{k+1} b_1(p_k, q_{k+1})}{b_0(p_k, q_{k+1})}.$$

Итерационный процесс прекращается либо при выполнении условия

$$\sqrt{(p_{k+1} - p_k)^2 + (q_{k+1} - q_k)^2} \le \varepsilon,$$

где малая величина ε определяет точность решения, либо если число итераций превысит допустимый уровень.

Если вместо (1.27) использовать модифицированные итерационные формулы вида

$$\begin{aligned} q_{k+1} &= (1-\lambda) q_k + \lambda \frac{a_0}{b_0(p_k, q_k)}, \\ p_{k+1} &= (1-\lambda) p_k + \lambda \frac{a_1 - q_{k+1} b_1(p_k, q_{k+1})}{b_0(p_k, q_{k+1})}, \end{aligned} \tag{1.28}$$

то подбором величины параметра λ часто удается улучшить сходимость итерационного процесса.

Литература

1. Бахвалов, Н.С. Численные методы [Текст] / Н.С.Бахвалов, Н.П.Жидков, Г.М Кобельков. – М.: ФИЗМАТЛИТ, 2000.
2. Крылов, В.И. Вычислительные методы. Том I [Текст] / В.И.Крылов, В.В.Бобков, П.И.Монастырный. – М.: Наука, 1976.
3. Турчак, Л.И. Основы численных методов [Текст] / Л.И.Турчак, П.В.Плотников. – М.: ФИЗМАТЛИТ, 2002.
4. Каханер, Д. Численные методы и программное обеспечение [Текст] / Д.Каханер, К.Моулер, С.Нэш. – М.: Мир, 1998.
5. Мэтьюз, Д. Численные методы. Использование MATLAB [Текст] / Д.Мэтьюз, К.Финк. – М.: Издательский дом «Вильямс», 2001.
6. Ортега, Д. Итерационные методы решения нелинейных систем уравнений со многими неизвестными [Текст] / Д.Ортега, В.Рейнболдт. – М.: Мир, 1975.

Приложение 1

Программы решения нелинейных уравнений

Ниже приведены тексты MathCAD-программ, предназначенных для решения нелинейных уравнений общего вида (1.1). Они имеют структуру программных модулей (рис. П.1.1, рис. П.1.3 – П.1.5) с заголовками

FunZero_Sec(a,b,F,ε), ***FunZero_Stff(a,b,F,ε)*** и ***FunZero_I(a,b,F,ε)***,

где a и b – левая и правая границы интервала локализации корня, F – имя функции в левой части уравнения, ε – точность поиска корня.

Программные модули возвращают трехмерный вектор-столбец с компонентами: найденное значение корня, значение функции при найденном значении аргумента, количество потребовавшихся для поиска итераций.

Программа *FunZero_Sec* реализует полиалгоритм, состоящий из алгоритмов бисекции и секущих, программа *FunZero_Stff* – полиалгоритм, состоящий из алгоритмов бисекции и Стеффенсона, а программа *FunZero_I* – алгоритм обратной параболической интерполяции.

Пример обращения к программным модулям приведен на рис. П.1.1.

$F(x) := \exp(-x) - x \quad FunZero\_Sec(0,1,F,10^{-6})^T = [0.567143 \;\; -6.84075\cdot 10^{-12} \;\; 7]$

$FunZero\_Stff(0,1,F,10^{-6})^T = [0.567143 \;\; 0 \;\; 6] \quad FunZero\_Stff(0,1,F,10^{-6})^T = [0.567143 \;\; 0 \;\; 3]$

Рис. П.1.1. Обращения к программам решения нелинейных уравнений

На рис. П.1.2 приведен пример обращения к программе поиска корней полинома. Программа реализует вычисленииния по методу Лина с итерациями (1.28). Текст программы, состоящей из двух модулей, приведен на рис. П.1.6 и рис. П.1.7.

$x\cdot(x+1)\cdot(x-3)\cdot(x+4)\cdot(x-5)\ \text{expand},x \rightarrow x^5 - 3\cdot x^4 - 21\cdot x^3 + 43\cdot x^2 + 60\cdot x$

$\lambda \equiv 0.2 \quad A := (0 \;\; 60 \;\; 43 \;\; -21 \;\; -3 \;\; 1)^T$

$\text{PolyRoots\_L}(A)^T = \left[-2.9931582\cdot 10^{-6} \;\; -0.999999 \;\; 3.0000015 \;\; -3.9999995 \;\; 5.0000008\right]$

Рис. П.1.2. Обращения к программе поиска корней полинома

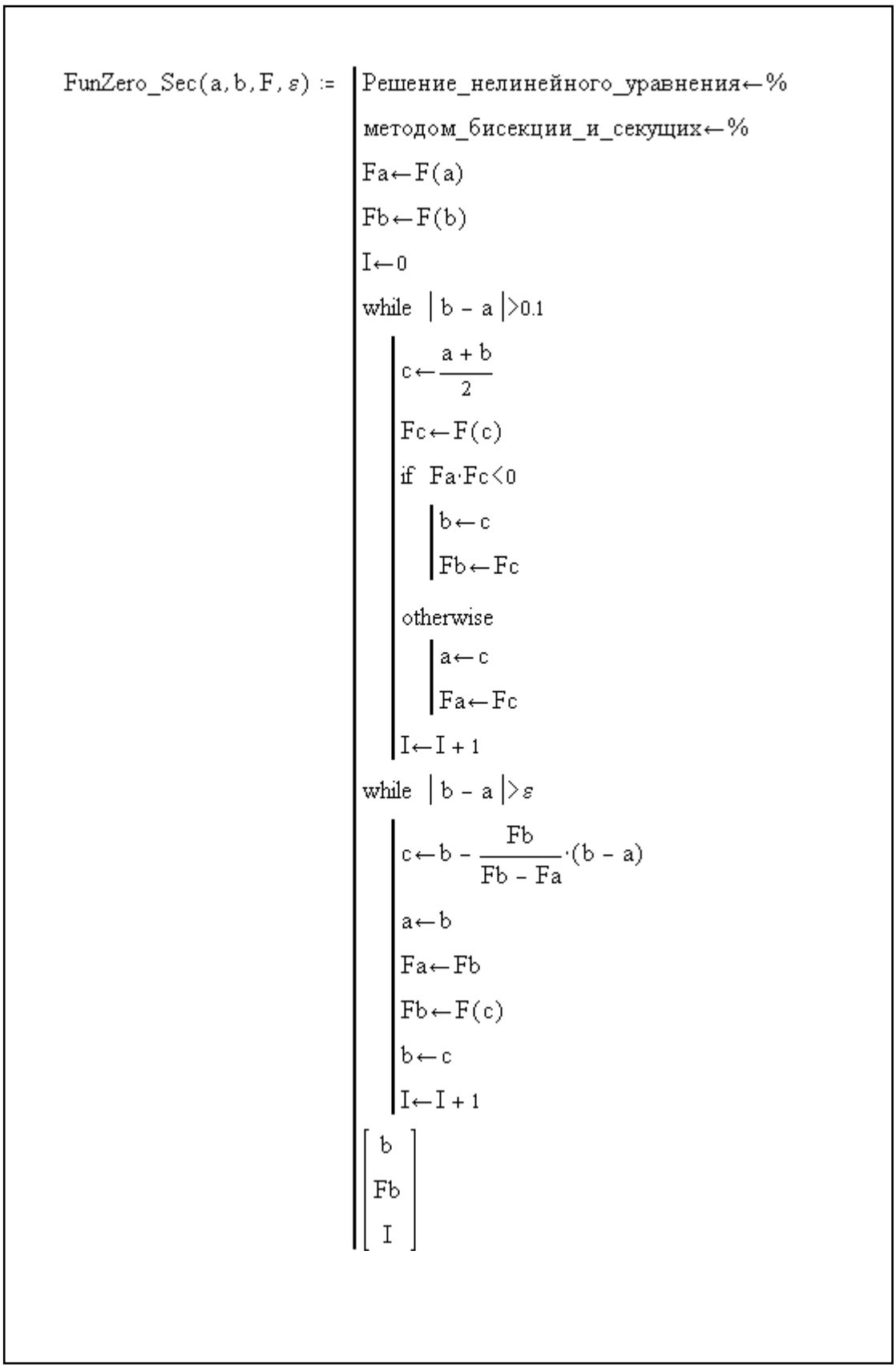


Рис. П.1.3. Программа решения нелинейных уравнений методом бисекции и секущих

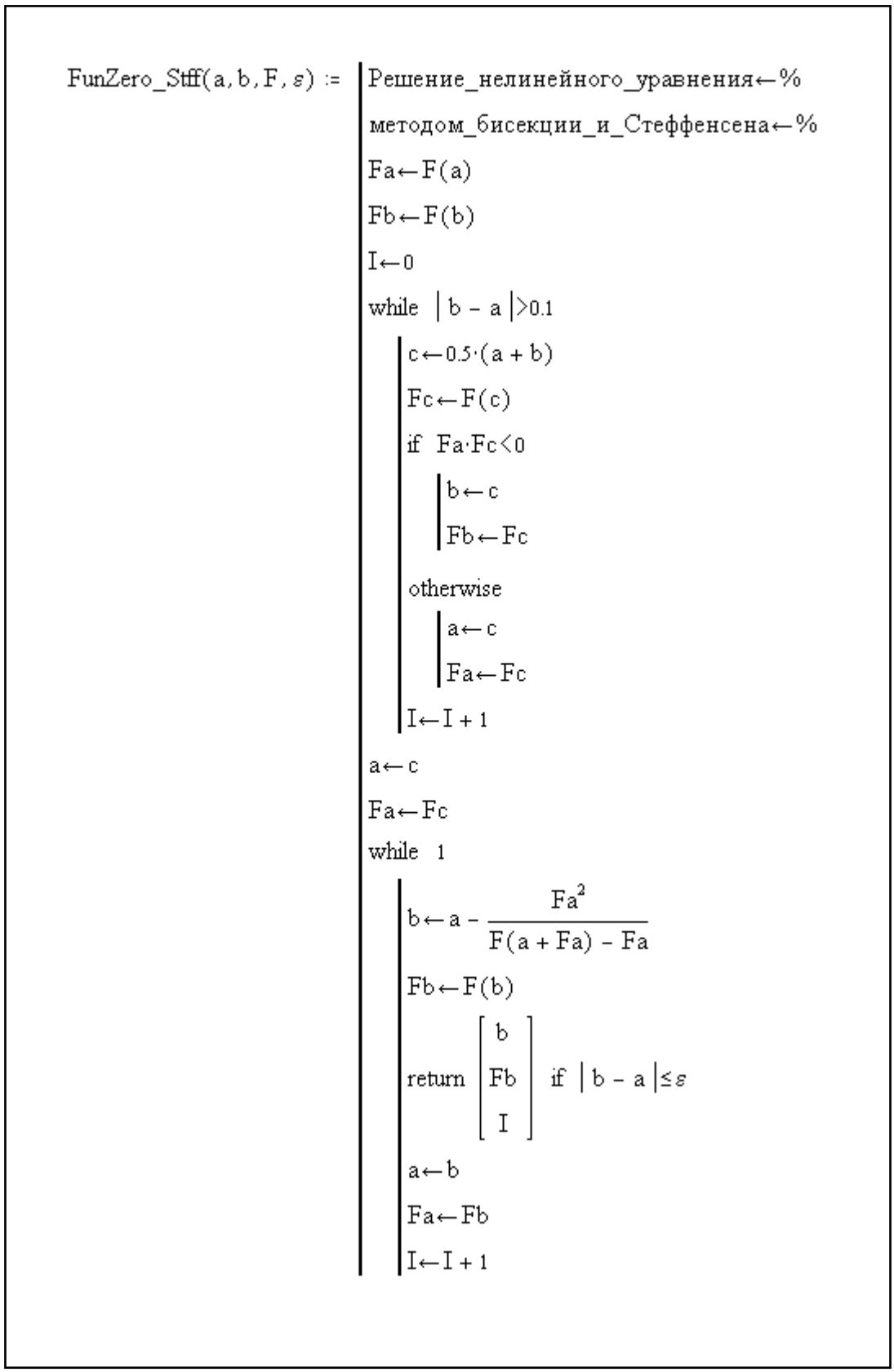


Рис. П.1.4. Программа решения нелинейных уравнений методом бисекции и Стеффенсена

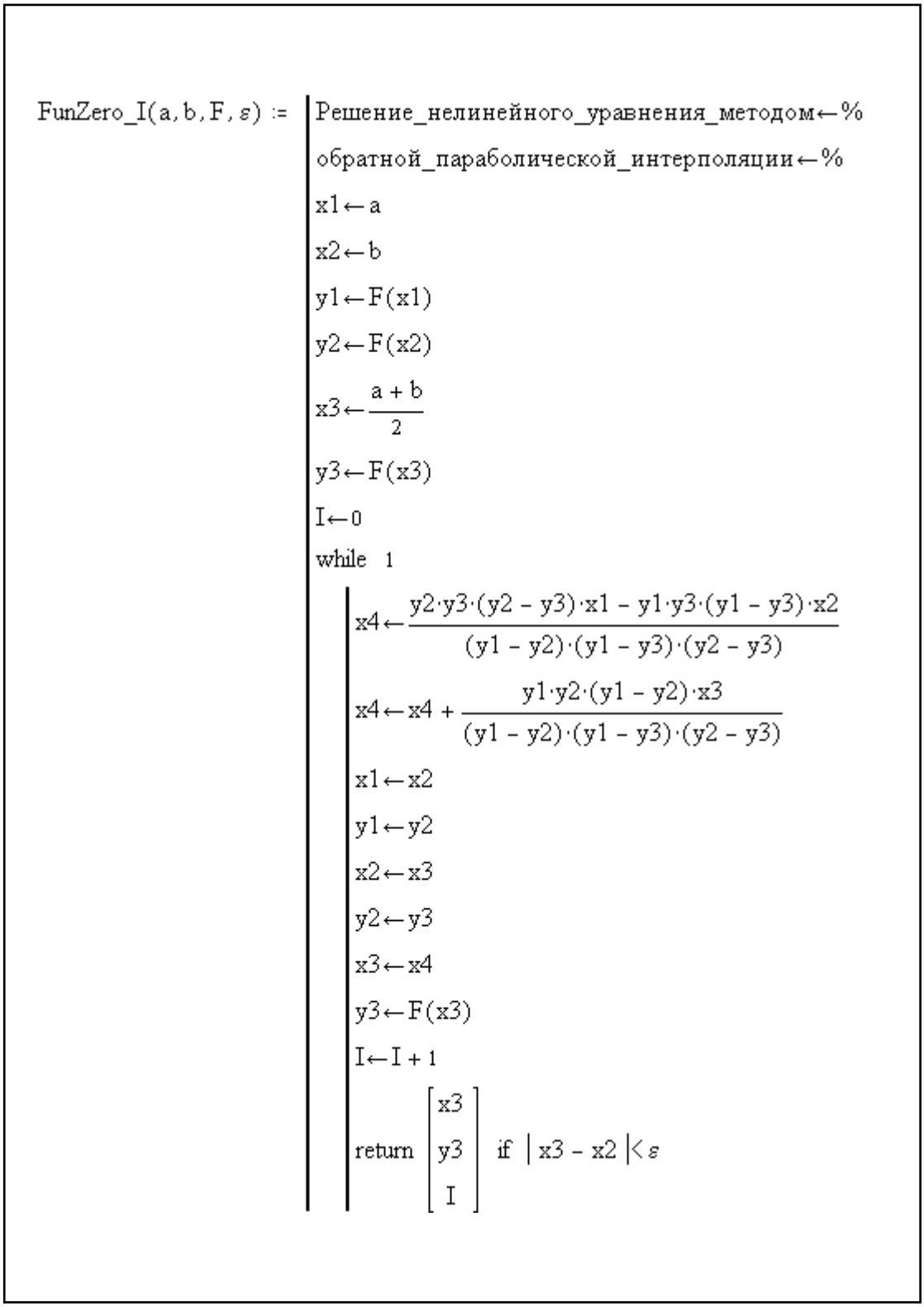


Рис. П.1.5. Программа решения нелинейных уравнений методом обратной параболической интерполяции

```
PolyFactor(A, N, p, q) := | Программа_выделения_квадратичного_множителя ← %
                          | return A  if  N ≤ 2
                          | I ← 0
                          | while 1
                          |     B_1 ← 1  if  N = 3
                          |     B_{N-3} ← A_{N-1} - p
                          |     B_{N-4} ← A_{N-2} - p·B_{N-3} - q  if  N > 3
                          |     for  n ∈ N - 5 .. 0                if  N ≥ 5
                          |       B_n ← A_{n+2} - p·B_{n+1} - q·B_{n+2}
                          |     Q ← (1 - λ)·q + λ·A_0 / B_0
                          |     P ← (1 - λ)·p + λ·(A_1 - Q·B_1) / B_0
                          |     ε ← sqrt((p - P)^2 + (q - Q)^2)
                          |     I ← I + 1
                          |     Out_N ← P
                          |     Out_{N-1} ← Q
                          |     Out_{N-2} ← 1
                          |     for  n ∈ 0 .. N - 3
                          |       Out_n ← B_n
                          |     return Out  if  ε ≤ 10^-6
                          |     return ["Problem"; I]  if  I > 1000
                          |     p ← P
                          |     q ← Q
```

Рис. П.1.6. Программа выделения из полинома квадратичного множителя

PolyRoots_L(A) :=

Программа_вычисления_корней_полинома ← %

$N \leftarrow \text{length}(A) - 1$

$A \leftarrow \frac{A}{A_N}$

if N = 2

$x_0 \leftarrow \frac{-A_{N-1}}{2} + \sqrt{\left(\frac{A_{N-1}}{4}\right)^2 - A_{N-2}}$

$x_1 \leftarrow \frac{-A_{N-1}}{2} - \sqrt{\left(\frac{A_{N-1}}{4}\right)^2 - A_{N-2}}$

return x

$p \leftarrow 1$

$q \leftarrow 1$

$n \leftarrow 0$

while 1

$B \leftarrow \text{PolyFactor}(A, N, p, q)$

return $\begin{bmatrix} \text{"Problem"} \\ N \end{bmatrix}$ if $B_0 = \text{"Problem"}$

$x_n \leftarrow \frac{-B_N}{2} + \sqrt{\left(\frac{B_N}{4}\right)^2 - B_{N-1}}$

$x_{n+1} \leftarrow \frac{-B_N}{2} - \sqrt{\left(\frac{B_N}{4}\right)^2 - B_{N-1}}$

$n \leftarrow n + 2$

$N \leftarrow N - 2$

$x_n \leftarrow -B_{N-1}$ if N = 1

if N = 2

$x_n \leftarrow \frac{-B_{N-1}}{2} + \sqrt{\left(\frac{B_{N-1}}{4}\right)^2 - B_{N-2}}$

$x_{n+1} \leftarrow \frac{-B_{N-1}}{2} - \sqrt{\left(\frac{B_{N-1}}{4}\right)^2 - B_{N-2}}$

return x if $N \le 2$

$A \leftarrow B$

Рис. П.1.7. Программа вычисления корней полинома

ГЛАВА 2

РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

2.1. Введение

С системами нелинейных уравнений при решении физических задач приходится встречаться не менее часто, чем с одним нелинейным алгебраическим или трансцендентным уравнением. Общая форма записи системы из *n* нелинейных алгебраических или трансцендентных уравнений с *n* неизвестными имеет вид

$$\begin{aligned} f_1(x_1, x_2, ..., x_n) &= 0, \\ f_2(x_1, x_2, ..., x_n) &= 0, \\ &\cdots\cdots\cdots \\ f_n(x_1, x_2, ..., x_n) &= 0. \end{aligned} \tag{2.1}$$

Подобные системы уравнений могут возникать, например, при численном моделировании нелинейных физических систем на этапе поиска их стационарных состояний. В ряде случаев системы вида (2.1) получаются опосредованно, в процессе решения некоторой другой вычислительной задачи. К примеру, пытаясь минимизировать функцию нескольких переменных, можно искать те точки многомерного пространства, в которых градиент функции равен нулю. При этом приходится решать систему уравнений (2.1) с левыми частями – проекциями градиента на координатные оси.

В векторных обозначениях систему (2.1) можно записать в более компактной форме:

$$\mathbf{f}(\mathbf{x}) = 0, \tag{2.2}$$

где $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ – вектор-столбец аргументов, $\mathbf{f} = (f_1, f_2, ..., f_n)^T$ – вектор-столбец функций; символом $(.)^T$ обозначена операция транспонирования.

Поиск решений системы нелинейных уравнений – это задача намного более сложная, чем решение одного нелинейного уравнения. Обобщения некоторых численных методов, пригодных для одного уравнения, на системы уравнений требуют слишком большого объема вычислений и не используются на практике. В частности, это относится к методу половинного деления. Тем не менее, ряд итерационных методов решения нелинейных уравнений может быть распространен и на системы нелинейных уравнений.

2.2. Метод простой итерации

Метод простой итерации для систем нелинейных уравнений по существу является обобщением одноименного метода для одного уравнения. Он основан на том, что система уравнений (2.1) приводится к виду

$$\begin{aligned} x_1 &= g_1(x_1, x_2, ..., x_n), \\ x_2 &= g_2(x_1, x_2, ..., x_n), \\ &\cdots\cdots\cdots\cdots \\ x_n &= g_n(x_1, x_2, ..., x_n), \end{aligned}$$

и итерации проводятся по формулам

$$\begin{aligned} x_1^{(k+1)} &= g_1\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right), \\ x_2^{(k+1)} &= g_2\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right), \\ &\cdots\cdots\cdots\cdots \\ x_n^{(k+1)} &= g_n\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right). \end{aligned} \qquad (2.3)$$

Здесь верхний индекс указывает на номер приближения. Итерационный процесс (2.3) начинается с некоторого начального приближения $\left(x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)}\right)$ и продолжается до тех пор, пока модули приращений всех аргументов после одной итерации не станут меньше заданного ε:

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\|_\infty = \max_{1\le i\le n}\left\{\left|x_i^{(k+1)} - x_i^{(k)}\right|\right\} < \varepsilon .$$

С равным успехом можно использовать условие

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\|_2 = \left|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right| = \sqrt{\left(x_i^{(k+1)} - x_i^{(k)}\right)^2} < \varepsilon .$$

Метод простой итерации прямо ведет к решению и легко программируется. Но два существенных недостатка ограничивают область его применения. Один из них – довольно жесткое условие сходимости, состоящее в том, что для всех значений $\mathbf{x}$ из некоторой окрестности решения $\mathbf{X}$ должно выполняться неравенство

$$\left\|\mathbf{G}'(\mathbf{x})\right\|_\infty < 1, \qquad (2.4)$$

где $\mathbf{G}'$ – матрица Якоби вектор-функции $\mathbf{G}$, а символом $\|.\|_\infty$ обозначена матричная норма:

$$\left\|\mathbf{G}'(\mathbf{x})\right\|_\infty = \max_{1\le i\le n}\left\{\sum_{j=1}^{n}\left|\partial G_i / \partial x_j\right|\right\}.$$

Если же точка начала итераций выбрана далеко от истинного решения, то сходимость метода, даже при выполнении условия (2.4), не гарантирована. Ясно, что проблема выбора начального приближения, непростая уже для одного уравнения, для нелинейных систем становится весьма сложной.

Другой недостаток метода простой итерации – медленная сходимость.

2.3. Метод Зейделя

Одной из модификаций метода простой итерации, направленной на ускорение сходимости, является модификация, носящая название *метода Зейделя*. В этом методе итерационный процесс описывается формулами

$$\begin{aligned} x_1^{(k+1)} &= g_1\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right), \\ x_2^{(k+1)} &= g_2\left(x_1^{(k+1)}, x_2^{(k)}, ..., x_n^{(k)}\right), \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ x_n^{(k+1)} &= g_n\left(x_1^{(k+1)}, ..., x_{n-1}^{(k+1)}, x_n^{(k)}\right). \end{aligned} \qquad (2.4)$$

Здесь следует обратить внимание на то, что уточненное значение x_1 сразу же используется для уточнения x_2. Затем по новым значениям x_1 и x_2 вычисляется x_3, и т.д. Скорость сходимости у итерационного процесса (2.4) несколько выше, чем у простой итерации (2.3), но проблема выбора начального приближения остается по-прежнему острой. Иногда эту проблему удается решить с помощью рассматриваемого далее метода *возмущения параметров*.

Наряду с итерационным процессом (2.4) можно также рассмотреть процесс, в котором компоненты вектора приближений определяются из уравнений

$$\begin{aligned} f_1\left(\xi, x_2^{(k)}, ..., x_n^{(k)}\right) &= 0, \\ f_2\left(x_1^{(k+1)}, \xi, ..., x_n^{(k)}\right) &= 0, \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ f_n\left(x_1^{(k+1)}, ..., x_{n-1}^{(k+1)}, \xi\right) &= 0. \end{aligned} \qquad (2.5)$$

Каждое из них представляет собой уравнение с одним неизвестным ξ. Значение ξ_1, являющееся корнем первого из уравнений совокупности (2.5), рассматривается в качестве нового приближения для компоненты x_1: $x_1^{(k+1)} = \xi_1$. Затем корень ξ_2 второго уравнения считается новым приближением для x_2: $x_2^{(k+1)} = \xi_2$ и т.д. Привлекательность такого подхода состоит в возможности использования сравнительно простых методов

решения одного уравнения. Но на практике это может привести к очень большому объему вычислений.

2.4. Метод возмущения параметров

Суть этого метода состоит в следующем. Сначала, наряду с системой уравнений (2.1), рассматривается некоторая дополнительная система

$$\begin{aligned} &h_1^{(0)}(x_1, x_2, ..., x_n) = 0, \\ &h_2^{(0)}(x_1, x_2, ..., x_n) = 0, \\ &\quad ..... \\ &h_n^{(0)}(x_1, x_2, ..., x_n) = 0, \end{aligned} \tag{2.6}$$

которая подбирается так, чтобы было известно ее решение. Это может быть, например, система линейных алгебраических уравнений. Затем, «деформируя» левые части уравнений системы (2.6), превратим их в левые части уравнений исходной системы (2.1) с помощью конечного числа K последовательных изменений:

$$\begin{aligned} &h_1^{(k+1)}(x_1, x_2, ..., x_n) = h_1^{(k)}(x_1, x_2, ..., x_n) + \left[f_1(x_1, x_2, ..., x_n) - h_1^{(k)}(x_1, x_2, ..., x_n)\right]\frac{k+1}{K}, \\ &h_2^{(k+1)}(x_1, x_2, ..., x_n) = h_2^{(k)}(x_1, x_2, ..., x_n) + \left[f_2(x_1, x_2, ..., x_n) - h_2^{(k)}(x_1, x_2, ..., x_n)\right]\frac{k+1}{K}, \\ &..... \\ &h_n^{(k+1)}(x_1, x_2, ..., x_n) = h_n^{(k)}(x_1, x_2, ..., x_n) + \left[f_n(x_1, x_2, ..., x_n) - h_n^{(k)}(x_1, x_2, ..., x_n)\right]\frac{k+1}{K} \end{aligned} \tag{2.7}$$

где $k = 0,1,..., K-1$. При большом значении K последовательные изменения функций (2.7) будут малыми. После каждого изменения итерационным методом решается возмущенная система уравнений

$$\begin{aligned} &h_1^{(k+1)}(x_1, x_2, ..., x_n) = 0, \\ &h_2^{(k+1)}(x_1, x_2, ..., x_n) = 0, \\ &\quad ..... \\ &h_n^{(k+1)}(x_1, x_2, ..., x_n) = 0. \end{aligned} \tag{2.8}$$

Известное решение системы (2.6) используется как начальное приближение для итерационного решения системы (2.8) при $k = 0$. Так как уравнения этой системы мало отличаются от уравнений системы (2.6), весьма вероятно, что сходимость итерационного процесса будет обеспечена. Затем полученное решение рассматривается как начальное

приближение для решения системы (2.8) при $k=1$ и т.д. В конце счета, когда k становится равным *K-1*, последняя решаемая система уравнений (2.8) становится эквивалентной исходной системе (2.1).

Таким образом, проблема выбора начального приближения в методе возмущений параметров решена. Однако для превращения системы уравнений (2.6) в решаемую систему может потребоваться большое число шагов (от десятков до сотен). Поэтому применение данного метода часто связано со значительными затратами машинного времени. Положительную роль играет то, что при малых «деформациях» (2.7) решение каждой из систем уравнений (2.8) может быть получено всего за несколько итераций.

2.5. Итерации Пикара

В ряде случаев система уравнений (2.1) имеет специальный вид и в векторно-матричной форме обозначений записывается как

$$\mathbf{Ax} - \mathbf{G}(\mathbf{x}) = 0, \tag{2.9}$$

где $\mathbf{A}$ – заданная невырожденная матрица, а $\mathbf{G}$ – нелинейная векторная функция. К системам уравнений такого вида приводят, в частности, конечно-разностные методы решения нелинейных граничных задач.

Для системы (2.9) естественна следующая итерационная процедура

$$\mathbf{x}^{(k+1)} = \mathbf{A}^{-1}\mathbf{G}(\mathbf{x}^{(k)}), \tag{2.10}$$

которая часто называется итерациями *Пикара*. Использование обратной матрицы $\mathbf{A}^{-1}$ в формуле (2.10) имеет целью лишь компактную запись итерационного алгоритма. На самом деле, на каждом шаге итераций решается система линейных алгебраических уравнений (СЛАУ)

$$\mathbf{Ax}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)}).$$

Итерации Пикара можно рассматривать как частный случай более общего итерационного процесса:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{BF}(\mathbf{x}^{(k)}), \tag{2.11}$$

где $\mathbf{B}$ – заданная невырожденная матрица. Легко видеть, что если $\mathbf{F}(\mathbf{x}) = \mathbf{Ax} - \mathbf{G}(\mathbf{x})$ и $\mathbf{B} = \mathbf{A}^{-1}$, то (2.11) сводится к (2.10). При другом выборе матрицы $\mathbf{B}$ мы получим еще несколько алгоритмов, в частности, алгоритмы метода Ньютона и многомерного метода секущих.

2.6. Метод Ньютона

Как и в случае одного нелинейного уравнения, метод Ньютона для системы нелинейных уравнений имеет существенное преимущество в скорости сходимости перед методом простой итерации. Этим обусловлена широкая популярность метода Ньютона и его модификаций.

В основе метода Ньютона для систем уравнений лежит представление левых частей всех уравнений системы (2.1) рядами Тейлора. Пусть $\mathbf{x}^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right)^T$ – текущее приближение корня $\mathbf{X} = \left(X_1, X_2, ..., X_n\right)^T$ и $\mathbf{\Delta x} = \mathbf{X} - \mathbf{x}^{(k)}$. Тогда можно записать следующее покомпонентное разложение функции **f(X)** в ряды Тейлора:

$$f_1\left(X_1, X_2, ..., X_n\right) = f_1\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right) + \frac{\partial f_1}{\partial x_1}\Delta x_1 + ... + \frac{\partial f_1}{\partial x_n}\Delta x_n + ...,$$

$$f_2\left(X_1, X_2, ..., X_n\right) = f_2\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right) + \frac{\partial f_2}{\partial x_1}\Delta x_1 + ... + \frac{\partial f_2}{\partial x_n}\Delta x_n + ...,$$

.....

$$f_n\left(X_1, X_2, ..., X_n\right) = f_n\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right) + \frac{\partial f_n}{\partial x_1}\Delta x_1 + ... + \frac{\partial f_n}{\partial x_n}\Delta x_n + ...$$

Здесь все производные вычисляются при $\mathbf{x} = \mathbf{x}^{(k)}$. С учетом того, что $\mathbf{f(X)} = \mathbf{0}$, получим систему уравнений, эквивалентную системе (2.1):

$$\begin{aligned}
&f_1\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right) + \frac{\partial f_1}{\partial x_1}\Delta x_1 + ... + \frac{\partial f_1}{\partial x_n}\Delta x_n + ... = 0,\\
&f_2\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right) + \frac{\partial f_2}{\partial x_1}\Delta x_1 + ... + \frac{\partial f_2}{\partial x_n}\Delta x_n + ... = 0,\\
&.....\\
&f_n\left(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right) + \frac{\partial f_n}{\partial x_1}\Delta x_1 + ... + \frac{\partial f_n}{\partial x_n}\Delta x_n + ... = 0.
\end{aligned} \qquad (2.12)$$

Если теперь в левых частях уравнений (2.12) оставить лишь слагаемые, содержащие нулевую и первую степени приращений Δx_i, то исходную нелинейную задачу удается свести к решению системы линейных уравнений вида

$$\begin{aligned}
&\frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 + ... + \frac{\partial f_1}{\partial x_n}\Delta x_n = -f_1,\\
&\frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 + ... + \frac{\partial f_2}{\partial x_n}\Delta x_n = -f_2,\\
&.....\\
&\frac{\partial f_n}{\partial x_1}\Delta x_1 + \frac{\partial f_n}{\partial x_2}\Delta x_2 + ... + \frac{\partial f_n}{\partial x_n}\Delta x_n = -f_n.
\end{aligned} \qquad (2.13)$$

В этой системе все функции и их производные вычисляются в точке $\mathbf{x}^{(k)}$. Так как полученная система линейных уравнений (2.13) приближенна, результат ее решения не является коренем исходной нелинейной системы (2.1), но может использоваться для расчета нового приближенного значения корня:

$$x_i^{(k+1)} = x_i^{(k)} + \Delta x_i,\ i = 1, 2, ..., n. \qquad (2.14)$$

Итерационный процесс, определяемый формулами (2.14), в которых приращения Δx_i вычисляются путем решения СЛАУ (2.13), представляет собой алгоритм метод Ньютона для системы нелинейных уравнений (2.1). Когда модули всех корректирующих приращений Δx_i становятся достаточно малыми:

$$\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\|_\infty < \varepsilon,$$

итерации прекращаются. Если абсолютные величины корней $X_1, X_2, ..., X_n$ значительно отличаются друг от друга, то для прекращения счета следует пользоваться нормированными корректирующими приращениями.

Систему уравнений (2.13) можно записать в векторно-матричной форме:

$$\mathbf{J}(\mathbf{x}^{(k)})\,\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}^{(k)}), \qquad (2.15)$$

если ввести в рассмотрение *матрицу Якоби* $\mathbf{J(x)}$ с элементами

$$J_{i,j}(x) = \partial f_i(x) \Big/ \partial x_j.$$

Формальная запись решения системы (2.15) имеет вид

$$\Delta\mathbf{x} = -\mathbf{J}(\mathbf{x}^{(k)})^{-1}\mathbf{f}(\mathbf{x}^{(k)}),$$

где $\mathbf{J}^{-1}$– матрица, обратная матрице Якоби. Воспользовавшись этим выражением, итерационные формулы (2.14) запишем как

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{J}(\mathbf{x}^{(k)})^{-1}\mathbf{f}(\mathbf{x}^{(k)}). \qquad (2.16)$$

В таком компактном виде формула (2.16) сильно напоминает одномерную форму метода Ньютона и часто используется при описании его многомерной формы. Однако в большинстве случаев вычисление обратной матрицы $\mathbf{J}^{-1}$ не является ни необходимым, ни желательным; предпочтение как раз следует отдавать решению СЛАУ (2.13).

При обсуждении эффективности многомерного метода Ньютона следует учитывать значительный объем вычислений при решении линейной системы (2.13) в случае, когда число уравнений в системе велико. Но при небольших n эта операция не требует существенных затрат машинного времени. Например, при $n = 2$ система (2.13) имеет простое аналитическое решение с небольшим числом арифметических операций. Этот случай целесообразно рассмотреть подробнее, т.к. система из двух уравнений очень часто встречается в вычислительной практике. К ней, в частности, сводится весьма распространенная задача о нахождении комплексных корней нелинейного уравнения $F(z) = 0$. Действительно, если ввести в рассмотрение функции

$$f_1(x_1, x_2) = \mathrm{Re}\big(F(x_1 + jx_2)\big) \text{ и } f_2(x_1, x_2) = \mathrm{Im}\big(F(x_1 + jx_2)\big),$$

то вещественная часть x_1 и мнимая часть x_2 комплексного корня z находятся из системы уравнений

$$f_1(x_1, x_2) = 0, \quad f_2(x_1, x_2) = 0.$$

Будем предполагать, что для данной системы определитель матрицы Якоби (якобиан) отличен от нуля на каждой итерации:

$$|\mathbf{J}| = \begin{vmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} \end{vmatrix} = \frac{\partial f_1}{\partial x_1}\frac{\partial f_2}{\partial x_2} - \frac{\partial f_1}{\partial x_2}\frac{\partial f_2}{\partial x_1} \neq 0.$$

Тогда система имеет решение

$$\Delta x_1 = \frac{1}{|\mathbf{J}|}\left(f_2 \frac{\partial f_1}{\partial x_2} - f_1 \frac{\partial f_2}{\partial x_2} \right),$$

$$\Delta x_2 = \frac{1}{|\mathbf{J}|}\left(f_1 \frac{\partial f_2}{\partial x_1} - f_2 \frac{\partial f_1}{\partial x_1} \right).$$

Теперь формулы итераций (2.14) можно записать в явном виде:

$$x_1^{(k+1)} = x_1^{(k)} - \frac{1}{|\mathbf{J}|}\left(f_2 \frac{\partial f_1}{\partial x_2} - f_1 \frac{\partial f_2}{\partial x_2} \right),$$
$$x_2^{(k+1)} = x_2^{(k)} - \frac{1}{|\mathbf{J}|}\left(f_1 \frac{\partial f_2}{\partial x_1} - f_2 \frac{\partial f_1}{\partial x_1} \right). \qquad (2.17)$$

Все функции и их производные в правых частях этих формул вычисляются в точке $X^{(k)} = \left(x_1^{(k)}, x_1^{(k)}\right)$.

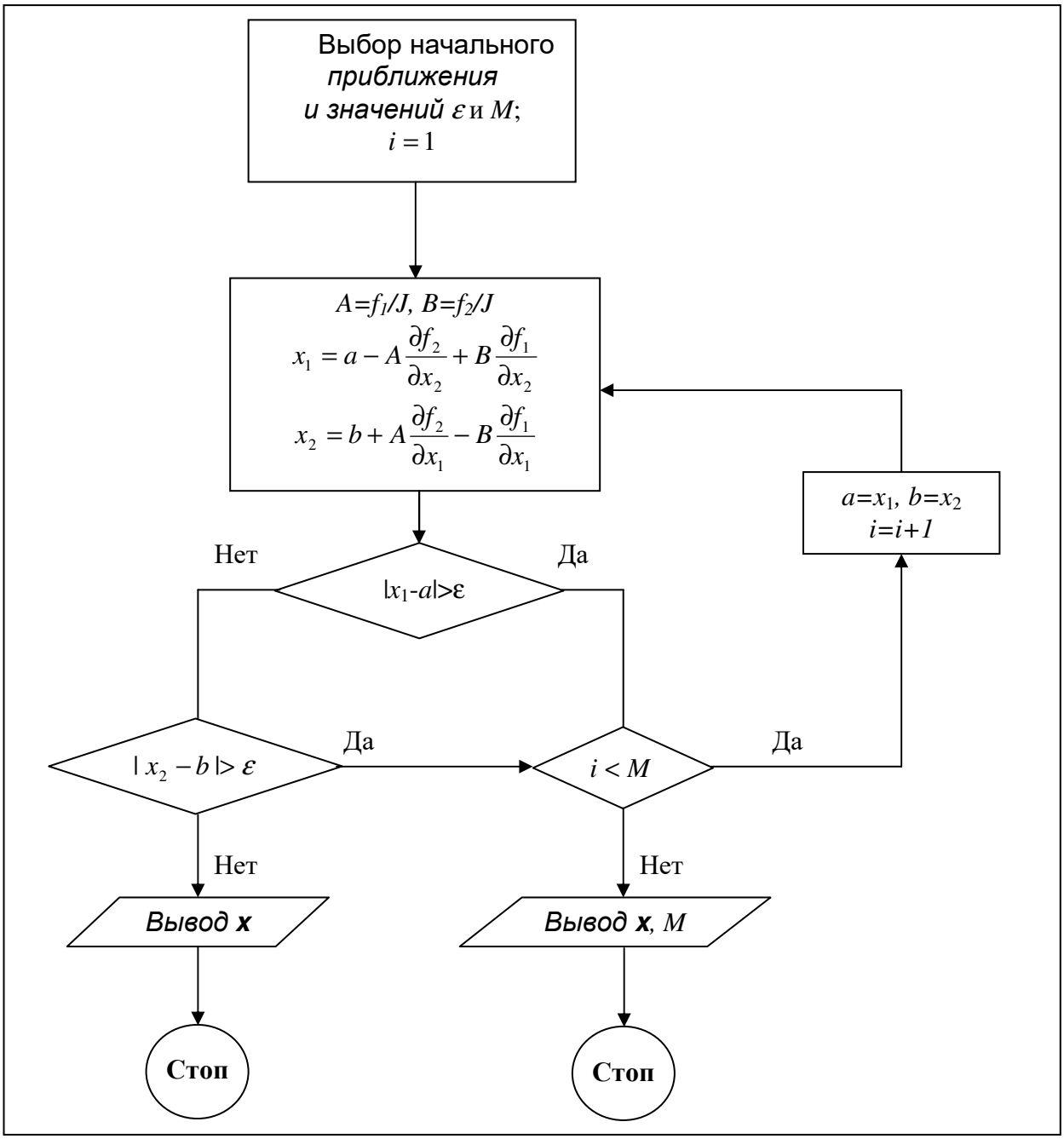


Рис. 2.1. Блок-схема метода Ньютона для системы двух уравнений

Блок-схема итерационного алгоритма (2.17) приведена на рис. 2.1.

При условии, что в окрестности корня вектор-функция $\mathbf{f}(\mathbf{x})$ дважды непрерывно дифференцируема по всем аргументам и матрица $\mathbf{J}$ невырождена, многомерный метод Ньютона сходится квадратично:

$$\left|\mathbf{x}^{(k+1)} - \mathbf{X}\right| < C\left|\mathbf{x}^{(k)} - \mathbf{X}\right|^2.$$

Однако для обеспечения сходимости метода весьма важен удачный выбор начального приближения. Область сходимости сужается с увеличением числа уравнений и ростом их сложности.

Процесс сходимости итераций метода Ньютона от начального приближения $\mathbf{x}^{(0)} = (2,2)^T$ к корню $\mathbf{X} = (1,1)^T$ системы уравнений

$$\begin{aligned} x_1^5 + x_2^3 - x_1 x_2 - 1 = 0, \\ x_1^2 x_2 + x_2 - 2 = 0 \end{aligned} \qquad (2.18)$$

иллюстрируется таблицей 2.1. Во второй и третий столбцы помещены компоненты вектора приближений, полученные в ходе итераций. Погрешность приближений занесена в четвертый столбец таблицы.

Таблица 2.1

k	$x_1^{(k)}$	$x_2^{(k)}$	$\left\|\mathbf{x}^{(k)} - \mathbf{X}\right\|$	$\left\|\Delta\mathbf{x}^{(k)}\right\| \Big/ \left\|\Delta\mathbf{x}^{(k-1)}\right\|^2$
0	2.000000000	2.000000000	1.414213562	-
1	1.693548387	0.890322581	0.702167004	0.351
2	1.394511613	0.750180529	0.466957365	0.947
3	1.192344147	0.822840986	0.261498732	1.199
4	1.077447418	0.918968807	0.112089950	1.639
5	1.022252471	0.976124950	0.032637256	2.598
6	1.002942200	0.996839728	4.317853366E-3	4.054
7	1.000065121	0.999930102	9.553233627E-5	5.124
8	1.000000033	0.999999964	4.871185259E-8	5.337
9	1.000000000	1.000000000	1.272646866E-14	5.363

Как видно, итерации сходятся довольно быстро – результат с семью верными цифрами после десятичной точки получается после восьми итераций. Заметим, что при решении этой же задачи итерационным методом (2.11) с матрицей

$$\mathbf{B} = \begin{pmatrix} 0.032 & 0 \\ 0 & 0.9 \end{pmatrix}$$

для получения результата с сопоставимой погрешностью потребовалось 247 итераций.

Данные пятого столбца таблицы 2.1 подтверждают положение о квадратичной сходимости метода. Правда, зависимость $\left|\Delta\mathbf{x}^{(k)}\right| \approx C\left|\Delta\mathbf{x}^{(k-1)}\right|^2$ справедлива в довольно малой окрестности корня, а константа C достаточно велика: $C \approx 5.4$.

Фактором, снижающим вычислительную эффективность метода Ньютона при большом числе уравнений, является трудоемкость вычисления матрицы Якоби. В одномерном случае можно не без оснований предполагать, что «стоимость» вычисления $f'(x)$ та же, что и вычисления $f(x)$. При n измерениях требуется уже n^2 вычислений $f_i'(\mathbf{x})$, что во много раз более трудоемко, чем n вычислений $f_i(\mathbf{x})$. Поэтому метод Ньютона часто модифицируют таким образом, что новая матрица Якоби вычисляется не на каждой итерации, а через некоторое число шагов итерационного процесса. При этом ряд итераций проводится с одной и той же матрицей. Увеличение числа итераций, сопровождающее такую модификацию, компенсируется меньшей «стоимостью» одной итерации.

Если производные в матрице Якоби трудно или невозможно задать аналитически, то это будет серьезным препятствием в использовании метода Ньютона. В таком случае можно попытаться аппроксимировать частные производные конечными разностями, используя приближения, полученные на предыдущих итерациях. Например, формулы двухточечной аппроксимации производных левыми разностями в точке $\mathbf{x}^{(k)}$ имеют вид

$$\frac{\partial f_i}{\partial x_j} = \frac{f_i\left(x_1^{(k)},...,x_j^{(k)},...,x_n^{(k)}\right) - f_i\left(x_1^{(k)},...,x_j^{(k)}-h,...,x_n^{(k)}\right)}{h}.$$

Вычисленные таким образом производные можно использовать в уравнениях (2.13), проводя итерации в соответствии с общей схемой метода Ньютона. Однако следует иметь в виду, что приближенная матрица Якоби может оказаться плохо обусловленной.

Наконец заметим, что если все же желательно использовать аналитическую запись матрицы Якоби, то рутинную работу по дифференцированию функций и последующему программированию полученных выражений можно облегчить, используя компьютерные программы символьного дифференцирования.

2.6. Метод Бройдена

Поскольку прямое вычисление матрицы Якоби не всегда возможно, а ее конечно-разностная аппроксимация зачастую дает неудовлетвори-

тельные результаты, заслуживают внимания итерационные алгоритмы вида (2.16), в которых вместо матрицы Якоби используются ее некоторым способом составленные приближения. Такие алгоритмы носят название *квазиньютоновских*. Наиболее популярным из них является алгоритм, предложенный Бройденом в 1965 году.

Одна из возможных интерпретаций формулы Ньютона (2.16) заключается в следующем. На каждом шаге итерационного процесса, после того как найдено приближение $\mathbf{x}^{(k)}$, исходная векторная функция $\mathbf{f(x)}$ заменяется линейной моделью

$$\mathbf{F}^{(k)}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^{(k)}) + \mathbf{A}_k(\mathbf{x} - \mathbf{x}^{(k)}), \tag{2.19}$$

где $\mathbf{A}_k = \mathbf{J}(\mathbf{x}^{(k)})$. Решение $\mathbf{x}^{(k+1)}$ системы уравнений $\mathbf{F}^{(k)}(\mathbf{x}) = 0$, полученное в рамках данной модели, принимается за новое приближение к корню. В итоге итерационная формула принимает вид

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{A}_k^{-1}\mathbf{f}(\mathbf{x}^{(k)}). \tag{2.20}$$

Предположим теперь, что матрица Якоби недоступна, например, не может быть задана аналитически, и потребуем чтобы

$$\mathbf{F}^{(k)}(\mathbf{x}^{(k-1)}) = \mathbf{f}(\mathbf{x}^{(k-1)}),$$

т.е. чтобы модель, построенная для точки $\mathbf{x}^{(k)}$, позволяла вычислять значения векторной функции в предыдущей точке $\mathbf{x}^{(k-1)}$. Это приводит к так называемому соотношению секущих:

$$\mathbf{A}_k\left(\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right) = \mathbf{f}(\mathbf{x}^{(k)}) - \mathbf{f}(\mathbf{x}^{(k-1)}). \tag{2.21}$$

В общем случае матрица $\mathbf{A}_k$ содержит n^2 элементов, в то время как (2.21) представляет собой систему из n уравнений. Поэтому совершенно очевидно, что соотношение секущих полностью определяет $\mathbf{A}_k$ только при решении одного нелинейного уравнения $f(x) = 0$:

$$A_k = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}.$$

В методе Бройдена недостающие для однозначного определения матрицы $\mathbf{A}_k$ условия подбираются из следующих соображений. По аналогии с линейной моделью (2.19) для точки $\mathbf{x}^{(k)}$ нетрудно записать модель для предыдущей точки $\mathbf{x}^{(k-1)}$:

$$\mathbf{F}^{(k-1)}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^{(k-1)}) - \mathbf{A}_{k-1}(\mathbf{x} - \mathbf{x}^{(k-1)}). \tag{2.22}$$

Разность $\Delta\mathbf{F}^{(k)}(\mathbf{x}) = \mathbf{F}^{(k)}(\mathbf{x}) - \mathbf{F}^{(k-1)}(\mathbf{x})$ с учетом соотношения секущих (2.21) можно представить в форме

$$\Delta\mathbf{F}^{(k)}(\mathbf{x}) = \left(\mathbf{A}_k - \mathbf{A}_{k-1}\right)\left(\mathbf{x} - \mathbf{x}^{(k-1)}\right). \qquad (2.23)$$

Матрицу $\mathbf{A}_k$ предлагается выбирать так, чтобы модуль разности (2.23) был минимален для всех $\mathbf{x}$:

$$\rho(\mathbf{x}) = \sqrt{\left(\Delta\mathbf{F}^{(k)}(\mathbf{x})\right)^T \Delta\mathbf{F}^{(k)}(\mathbf{x})} = \min.$$

При этом матрица $\mathbf{A}_{k-1}$ считается заданной.

Для упрощения исследования функции $\rho(\mathbf{x})$ перейдем к новым координатам точки $\mathbf{x}$, таким что

$$\mathbf{x} - \mathbf{x}^{(k-1)} = \alpha\,\Delta\mathbf{x} + \mathbf{t},$$

где $\Delta\mathbf{x} = \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}$, $\mathbf{t}$ – произвольный вектор, перпендикулярный вектору $\Delta\mathbf{x}$ (это означает, что $\Delta\mathbf{x}^T\mathbf{t} = 0$), и $0 \le \alpha \le 1$. В координатах $(\alpha, \mathbf{t})$

$$\rho^2(\alpha, \mathbf{t}) = \alpha^2 \Delta\mathbf{x}^T \Delta\mathbf{A}^T \Delta\mathbf{A}\Delta\mathbf{x} + \alpha\left(\Delta\mathbf{x}^T \Delta\mathbf{A}^T \Delta\mathbf{A}\mathbf{t} + \mathbf{t}^T \Delta\mathbf{A}^T \Delta\mathbf{A}\Delta\mathbf{x}\right) + \mathbf{t}^T \Delta\mathbf{A}^T \Delta\mathbf{A}\mathbf{t}.$$

В этом выражении первое и третье слагаемые заведомо неотрицательны. Кроме того, величина $\Delta\mathbf{x}^T \Delta\mathbf{A}^T \Delta\mathbf{A}\Delta\mathbf{x}$ не зависит от матрицы $\mathbf{A}_k$, т.к. из соотношения секущих (2.21) следует, что

$$\Delta\mathbf{A}\Delta\mathbf{x} = \left(\mathbf{A}_k - \mathbf{A}_{k-1}\right)\Delta\mathbf{x} = \mathbf{f}(\mathbf{x}^{(k)}) - \mathbf{f}(\mathbf{x}^{(k-1)}) - \mathbf{A}_{k-1}\Delta\mathbf{x}. \qquad (2.24)$$

Поэтому, чтобы минимизировать $\rho^2(\alpha, \mathbf{t})$ при любых α и $\mathbf{t}$ матрицу $\mathbf{A}_k$ следует выбирать так, чтобы $\Delta\mathbf{A}\mathbf{t} = \left(\mathbf{A}_k - \mathbf{A}_{k-1}\right)\mathbf{t} = 0$. Равенство всегда будет иметь место, если

$$\Delta\mathbf{A} = \mathbf{v}\,\Delta\mathbf{x}^T,$$

где вектор $\mathbf{v}$ должен быть таким, чтобы удовлетворялось соотношение секущих. Подставив полученное выражение для $\Delta\mathbf{A}$ в соотношение секущих в форме (2.24), найдем вектор $\mathbf{v}$

$$\mathbf{v} = \frac{\mathbf{f}(\mathbf{x}^{(k)}) - \mathbf{f}(\mathbf{x}^{(k-1)}) - \mathbf{A}_{k-1}\Delta\mathbf{x}}{\Delta\mathbf{x}^T\Delta\mathbf{x}},$$

а затем и матрицу

$$\mathbf{A}_k = \mathbf{A}_{k-1} + \frac{\mathbf{f}(\mathbf{x}^{(k)}) - \mathbf{f}(\mathbf{x}^{(k-1)}) - \mathbf{A}_{k-1}\Delta\mathbf{x}}{\Delta\mathbf{x}^T\Delta\mathbf{x}}\Delta\mathbf{x}^T. \qquad (2.25)$$

Таким образом, получена итерационная формула (2.25), позволяющая по некоторой первоначально заданной матрице $\mathbf{A}_0$ построить последовательность матриц $\mathbf{A}_k$, удовлетворяющих соотношению секущих (2.21). Каждая из этих матриц может быть использована на соответствующем шаге итерационного процесса (2.20). Формула (2.25) носит название

формулы секущих, а основанный на ее использовании алгоритм называется *методом Бройдена*.

На каждом шаге метода Бройдена проводятся следующие вычисления.

1. Решается СЛАУ $\mathbf{A}_k \Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}^{(k)})$.
2. Определяется новое приближение $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}$ и вычисляется значение функции $\mathbf{f}(\mathbf{x}^{(k+1)})$.
3. По формуле (2.25) определяется матрица $\mathbf{A}_{k+1}$, необходимая для выполнения нового шага.

Алгоритм содержит неопределенность в выборе начального приближения $\mathbf{A}_0$. На практике для обеспечения начала итерационного процесса здесь единственный раз можно использовать конечные разности для аппроксимации матрицы Якоби $\mathbf{J}(\mathbf{x}^{(0)})$, положив затем $\mathbf{A}_0 = \mathbf{J}(\mathbf{x}^{(0)})$.

Исходя из процедуры конструирования алгоритма, можно сделать вывод о том, что метод Бройдена является многомерным обобщением метода секущих, точнее, одним из возможных вариантов такого обобщения. Скорость сходимости у метода Бройдена, как и у любого метода секущих, несколько ниже, чем у метода Ньютона. Это подтверждает таблица 2.2, содержащая результаты применения метода Бройдена к системе уравнений (2.18), которая ранее уже решались методом Ньютона (см. табл. 2.1). Например, результат с шестью верными цифрами после десятичной точки достигается за тринадцать итераций, в то время как методу Ньютона на это требуется восемь итераций.

Таблица 2.2

k	$x_1^{(k)}$	$x_2^{(k)}$	$\left\|\mathbf{x}^{(k)} - \mathbf{X}\right\|$
0	2.000000000	2.000000000	1.414213562
1	1.694513211	0.889023252	0.703323850
2	1.532940994	0.835742461	0.557679695
3	1.330935487	0.770464391	0.402746685
4	1.251500757	0.804076528	0.318808151
5	1.139841409	0.866849425	0.193092452
6	1.087198127	0.913245001	0.123003835
7	1.039140157	0.958904664	0.056751903
8	1.016525113	0.982554663	0.024029547
9	1.003700722	0.996037640	0.005421774
10	1.000537288	0.999428320	0.000784536
11	1.000005832	0.999993444	8.774735736E-6
12	1.000000808	0.999999157	1.167386327E-6
13	0.999999806	1.000000202	2.795316564E-7
14	1.000000000	1.000000000	3.994662952E-10

Тем не менее, метод Бройдена весьма популярен и находит широкое применение в вычислительной практике.

Литература

1. Бахвалов, Н.С. Численные методы [Текст] / Н.С.Бахвалов, Н.П.Жидков, Г.М Кобельков. – М.: ФИЗМАТЛИТ, 2000.
2. Пирумов, У.Г. Численные методы [Текст] / У.Г.Пирумов. – М.: Дрофа, 2003.
3. Турчак, Л.И. Основы численных методов [Текст] / Л.И.Турчак, П.В.Плотников. – М.: ФИЗМАТЛИТ, 2002.
4. Дэннис, Д. Численные методы безусловной оптимизации и решения нелинейных уравнений [Текст]/ Д.Дэннис, Р.Шнабель. – М.: Мир, 1988.
5. Мэтьюз, Д. Численные методы. Использование MATLAB [Текст] / Д.Мэтьюз, К.Финк. – М.: Издательский дом «Вильямс», 2001.
6. Ортега, Д. Итерационные методы решения нелинейных систем уравнений со многими неизвестными [Текст]/ Д.Ортега, В.Рейнболдт. – М.: Мир, 1975.

Приложение 2.1
Решение систем уравнений, зависящих от параметра

Во многих физических задачах встречаются уравнения и системы уравнений, содержащие функции, зависящие от параметра. По аналогии с записью (2.2), такие системы нелинейных уравнений можно представить в виде

$$\mathbf{f}(\mathbf{x};\lambda) = 0\,, \qquad (\text{П.2.1})$$

где λ – параметр системы. Ясно, что решения $\mathbf{x}_r$ системы (П.2.1) будут зависеть от величины параметра λ. При этом, что очень важно, зависимость $\mathbf{x}_r = \mathbf{x}_r(\lambda)$ часто является непрерывной.

Типичной задачей этого класса является, например, задача о волноводной дисперсии. В этом случае корни дисперсионных уравнений – волновые числа – зависят от частоты гармонической волны.

Допустим, что необходимо найти решения системы (П.2.1) для ряда значений параметра $\lambda_0 < \lambda_1 < ... < \lambda_M$, причем при $\lambda = \lambda_0$ решение $\mathbf{x}_r^{(0)}$ известно или система решается достаточно просто. Тогда, если величина приращения параметра $|\lambda_1 - \lambda_0|$ мала, в силу непрерывности $\mathbf{x}_r = \mathbf{x}_r(\lambda)$ есть основания ожидать, что разность $|\mathbf{x}_r^{(1)} - \mathbf{x}_r^{(0)}|$ также мала. Поэтому $\mathbf{x}_r^{(0)}$ может быть хорошим начальным приближением при решении системы $\mathbf{f}(\mathbf{x};\lambda_1) = 0$. Повторяя эту процедуру, используем каждое найденное решение предыдущей задачи в качестве начального приближения при

решении следующей задачи. Такой способ построения последовательности решений известен под названием *метод продолжения по параметру*. Рассмотренный в основной части данной главы метод возмущения параметров является его частным вариантом.

С методом продолжения по параметру тесно связан *метод дифференцирования по параметру* или *метод Давиденко*. Допустим, что система уравнений (П.2.1) в некотором интервале изменения λ определяет непрерывное и непрерывно дифференцируемое решение $\mathbf{x}_r(\lambda)$. Продифференцируем уравнения системы (П.2.1) по λ, придерживаясь правила дифференцирования сложной функции. Получим векторное равенство

$$\mathbf{J}\left(\mathbf{x}_r(\lambda);\lambda\right)\frac{d\mathbf{x}_r}{d\lambda}+\frac{\partial\mathbf{f}\left(\mathbf{x}_r(\lambda);\lambda\right)}{\partial\lambda}=0,$$

которое, предполагая невырожденность матрицы Якоби $\mathbf{J}\left(\mathbf{x}(\lambda);\lambda\right)$, преобразуем к системе дифференциальных уравнений вида

$$\frac{d\mathbf{x}_r}{d\lambda}=-\left[\mathbf{J}\left(\mathbf{x}_r(\lambda);\lambda\right)\right]^{-1}\frac{\partial\mathbf{f}\left(\mathbf{x}_r(\lambda);\lambda\right)}{\partial\lambda}. \qquad (П.2.2)$$

Пусть нами также получено решение $\mathbf{x}_r^{(0)}$ системы (П.2.1) для некоторого значения λ_0 параметра λ из интервала непрерывности функции $\mathbf{x}_r(\lambda)$. Тогда это решение можно использовать в качестве начального условия

$$\mathbf{x}_r(\lambda_0)=\mathbf{x}_r^{(0)} \qquad (П.2.3)$$

для системы дифференциальных уравнений (П.2.2).

Решение сформулированной задачи Коши (П.2.2), (П.2.3) даст, как можно надеяться, требуемое решение исходной системы нелинейных уравнений (П.2.1) в анализируемом диапазоне значений параметра λ. По существу, это непрерывный вариант метода продолжения по параметру.

Зависимость от параметра можно вводить в систему уравнений искусственно, что зачастую помогает в поиске ее решений. Например, пусть $\mathbf{x}^{(0)}$ – начальное приближение к решению системы $\mathbf{f}(\mathbf{x})=0$, но оно не может обеспечить сходимость описанных выше итерационных методов. Введем в рассмотрение новую вектор-функцию, зависящую от параметра:

$$\mathbf{F}(\mathbf{x};\lambda)=\mathbf{f}(\mathbf{x})+(\lambda-1)\mathbf{f}(\mathbf{x}^{(0)}). \qquad (П.2.4)$$

Совершенно очевидно, что при $\lambda=0$ система уравнений $\mathbf{F}(\mathbf{x};0)=\mathbf{f}(\mathbf{x})-\mathbf{f}(\mathbf{x}^{(0)})=0$ имеет решение $\mathbf{x}^{(0)}$, а при $\lambda=1$ она переходит в исходную систему: $\mathbf{F}(\mathbf{x};1)=\mathbf{f}(\mathbf{x})=0$. Следовательно, к системе $\mathbf{F}(\mathbf{x};\lambda)=0$ целесообразно применить метод дифференцирования по параметру, в рамках которого нетрудно получить систему дифференциальных уравнений

$$\frac{d\mathbf{x}_r}{d\lambda} = -\left[\mathbf{J}\left(\mathbf{x}_r(\lambda)\right)\right]^{-1}\mathbf{f}\left(\mathbf{x}^{(0)}\right)$$

с начальным условием $\mathbf{x}_r(0) = \mathbf{x}^{(0)}$. Решение этой задачи Коши при $\lambda = 1$ даст решение исходной системы уравнений $\mathbf{f}(\mathbf{x}) = 0$.

На практике задачи Коши, конечно, придется интегрировать численно, и для этого, в принципе, можно использовать любой из явных алгоритмов. В этом отношении метод дифференцирования по параметру выглядит весьма привлекательно. Но встречаются ситуации, когда плохая обусловленность или даже вырождение матрицы Якоби снижают надежность метода.

Приложение 2.2
Программы решения систем нелинейных уравнений

Здесь представлены три программных модуля вычислительной системы MathCAD, предназначенные для решения систем нелинейных уравнений вида (2.2). Заголовки модулей:

NSys_Z(x,F,ε), ***NSys_N(x,F,J,ε)***, и ***NSys_B(x,F,ε)***,

где ***x*** –вектор-столбец начального приближения, ***F*** – имя векторной функции левых частей уравнений, ***J*** – имя матрицы Якоби, ε – точность поиска решения.

Программные модули возвращают многомерный вектор-столбец решения. Кроме того, модуль *NSys_Z* возвращает максимальное число итераций, если решение не найдено.

Программа *NSys_Z* реализует алгоритм Зейделя, программа *NSys_N* – алгоритм Ньютона, а программа *NSys_B* – алгоритм Бройдена. Программа *NSys_B* содержит обращение к вспомогательному модулю ***Der(x,F,ε)***, который численно формирует матрицу Якоби в точке ***x***.

Пример обращения к программным модулям приведен на рис. П.2.1.

$$F(x) := \begin{bmatrix} x_0 \cdot x_1 - (x_1)^3 - (x_0)^5 + 1 \\ (x_0)^2 \cdot x_1 + x_1 - 2 \end{bmatrix} \qquad J(x) := \begin{bmatrix} x_1 - 5\cdot(x_0)^4 & x_0 - 3\cdot(x_1)^2 \\ 2\cdot x_0 \cdot x_1 & (x_0)^2 + 1 \end{bmatrix}$$

$$x := \begin{bmatrix} 2 \\ 2 \end{bmatrix} \quad NSys\_B(x, F, 10^{-4}) = \begin{bmatrix} 1.000001339 \\ 0.999998612 \end{bmatrix} \quad NSys\_N(x, F, J, 10^{-4}) = \begin{bmatrix} 1.000000033 \\ 0.999999964 \end{bmatrix}$$

Рис. П.2.1. Обращения к программам решения систем нелинейных уравнений

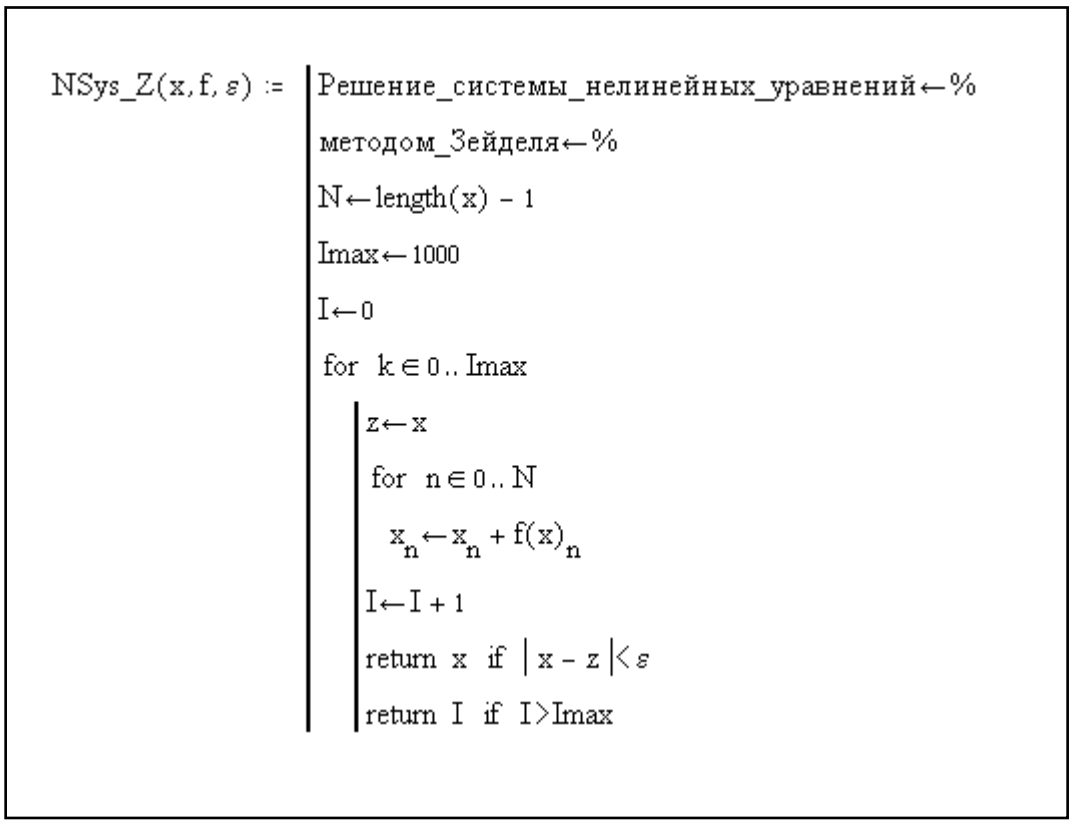


Рис. П.2.2. Программа решения системы нелинейных уравнений методом Зейделя

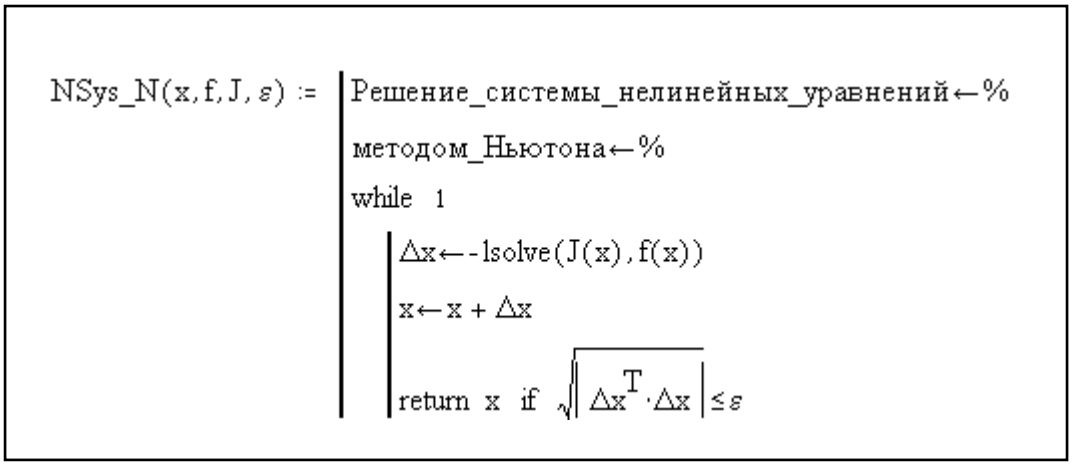


Рис. П.2.3. Программа решения системы нелинейных уравнений методом Ньютона

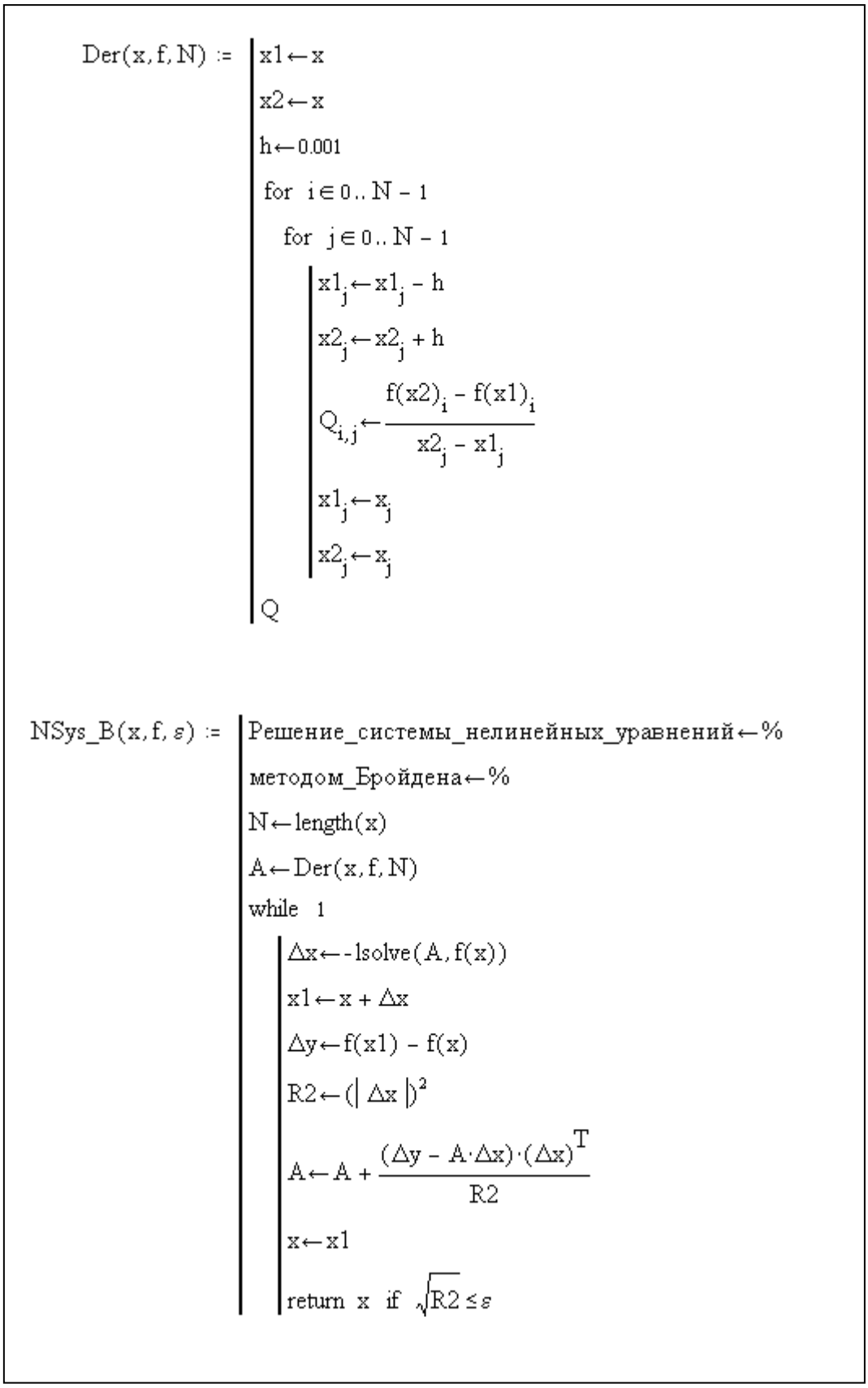


Рис. П.2.4. Программа решения системы нелинейных уравнений методом Бройдена

ГЛАВА 3

МЕТОДЫ ОДНОМЕРНОЙ ОПТИМИЗАЦИИ

3.1. Введение

Термин *оптимизация* означает минимизацию или максимизацию функции. Задачи оптимизации часто возникают в физических исследованиях и технических разработках. Например, минимум энергии физической системы определяет ее стационарное состояние. Поиск наиболее экономичной конструкции радиоэлектронного устройства проводится минимизацией энергопотребления, выраженного в виде функции параметров системы. Иногда оптимизационные задачи появляются опосредованно, как средство решения каких-либо других задач. Так, краевую задачу для системы обыкновенных дифференциальных уравнений можно решать методом пристрелки, отыскивая нулевой минимум целевой функции, сконструированной из граничных условий.

В общем случае задача оптимизации формулируется как задача нахождения на множестве S n-мерного пространства таких значений аргументов функции n вещественных переменных $f(x_1, x_2, ..., x_n)$, при которых эта функция достигает минимума или максимума. Минимизируемая (или максимизируемая) функция часто называется целевой функцией. Если множество S представляет собой все n-мерное пространство, то говорят, что решается задача *безусловной оптимизации* (оптимизации без ограничений). В противном случае речь идет о задаче *оптимизации с ограничениями* в виде условий, определяющих множество S. В дальнейшем мы будем обсуждать в основном методы безусловной оптимизации и оптимизации на интервале. Более сложные задачи оптимизации с ограничениями являются предметом специальных курсов линейного и нелинейного программирования.

В данном разделе мы рассмотрим методы оптимизации функции одной переменной. При этом для определенности везде будем решать задачу о минимизации функции. Задачу о поиске максимума легко превратить в задачу минимизации, поменяв знак функции на противоположный.

Предварительно определим еще несколько терминов. Будем говорить, что функция $f(x)$ имеет *глобальный минимум* в точке x_*, если точка x_* допустима (принадлежит множеству S) и $f(x_*) \le f(x)$ для всех допустимых точек. Аналогично, $f(x)$ имеет *локальный минимум* в точке x_*, если точка x_* допустима и $f(x_*) \le f(x)$ для всех допустимых точек из окрестности x_*. Большинство численных методов оптимизации использует информацию о поведении функции в окрестности предполагаемой точки ее минимума и, поэтому, ориентировано на поиск локальных минимумов.

Наконец, функция $f(x)$ называется *унимодальной* на отрезке $[a,b]$, если на этом отрезке она имеет единственный минимум и при $x \le x_*$ строго убывает, а при $x \ge x_*$ строго возрастает.

Методы одномерной оптимизации являются итерационными методами и их можно разделить на две группы. К одной из них относятся методы, использующие значения производных для достижения высокой скорости сходимости итерационного процесса, к другой – методы, оперирующие только со значениями функции. Последние обладают гарантированной сходимостью к минимуму унимодальной функции, но являются довольно медленными.

3.2. Метод Ньютона

Среди методов одномерной оптимизации, использующих значения производных функции, наиболее известен метод Ньютона. Ранее он рассматривался в контексте решения нелинейных уравнений. Теперь применим его для минимизации функции $f(x)$. Будем предполагать, что функция имеет по крайней мере три непрерывные производные и ограничена снизу.

Для функции общего вида не существует формулы для вычисления точки, минимизирующей $f(x)$. Идея метода Ньютона состоит в аппроксимации $f(x)$ в окрестности предполагаемой точки минимума x_0 более простой функцией, для минимизации которой можно воспользоваться аналитическими выражениями. Точка минимума x_1 этой аппроксимирующей функции дает новое приближение для точки минимума исходной функции $f(x)$. Затем процесс повторяется до тех пор, пока модуль разности между двумя последовательными приближениями не достигнет заданной погрешности минимизации. Так как линейная функция не имеет конечного минимума, простейшей аппроксимацией является квадратичная

Чтобы получить формулу итерационного процесса, предположим, что точка x_n – текущее приближение к истинному положению минимума x_*, и рассмотрим разложение функции $f(x)$ в ряд Тейлора в окрестности точки x_n, ограничившись тремя членами ряда:

$$q(h) = f(x_n + h) = f(x_n) + hf'(x_n) + \frac{1}{2}h^2 f''(x_n).$$

Теперь задача состоит в минимизации квадратичной функции $q(h)$. При этом предполагается, что точка минимума h_* функции $q(h)$ дает лучшее

приближение к значению x_*: $x_{n+1} = x_n + h_*$. Для минимизации функции $q(h)$ вычислим производную $q'(h)$ и приравняем ее к нулю, что дает

$$h_* = -\frac{f'(x_n)}{f''(x_n)} .$$

Таким образом, алгоритм метода Ньютона описывается итерационной формулой

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}. \qquad (3.1)$$

Условие сходимости итерационного процесса (3.1) можно сформулировать как следующее утверждение. Пусть $f'(x_*) = 0,\ f''(x_*) \neq 0$ и $f'''(x_*)$ непрерывна. Тогда существует окрестность точки x_* такая, что если начальное приближение x_0 принадлежит этой окрестности, то последовательность значений x_n, вычисляемых по формуле (3.1), сходится к x_* при $n \to \infty$. Для функций не являющихся унимодальными при $f''(x_n) > 0$ итерации (3.1) сходятся к локальному минимуму, а при $f''(x_n) < 0$ – к локальному максимуму.

Когда метод Ньютона сходится, скорость сходимости квадратична. Это означает, что если точка x_n достаточно близка к x_*, то

$$\left|x_{n+1} - x_*\right| \le C\left|x_n - x_*\right|^2,$$

где C – неотрицательная константа, зависящая от вида минимизируемой функции. Проще говоря, число верных значащих цифр приближения x_n удваивается с каждой новой итерацией.

Главный недостаток метода Ньютона заключается в том, что он требует вычисления производных функции $f(x)$. В сложных практических задачах это может оказаться трудным или попросту невозможным. Кроме того, для сходимости метода начальное приближение x_0 необходимо выбирать достаточно близко к искомой точке локального минимума. На практике предварительная оценка положений минимумов может вызвать затруднения. В этом случае единственная возможность оценки – вычислить значения функции в некоторой последовательности точек и выбрать наименьшее значение.

Отметим, что одномерная оптимизация тесно связана с задачей решения нелинейных уравнений. Так, формула (3.1) получается, если методом Ньютона решать, уравнение

$$f'(x) = 0,$$

которое представляет собой необходимое условие экстремума функции. Однако на практике способ решения оптимизационной задачи путем сведения ее к нелинейному уравнению используется сравнительно редко, так как он требует вычисления производной.

С другой стороны, нелинейное уравнение $g(x)=0$ можно решить, найдя точку нулевого минимума функции

$$f(x)=g^2(x).$$

Этот прием часто используется в вычислительной практике.

3.3. Метод последовательной параболической интерполяции

Итерационный алгоритм метода Ньютона был разработан путем квадратичной аппроксимации функции с использованием значений ее двух первых производных. Еще один итерационный метод, основанный на квадратичной интерполяции, но использующий только значения минимизируемой функции, можно получить следующим образом.

Предположим, что функция $f(x)$ унимодальна. Выберем три произвольных значения аргумента x_1, x_2 и x_3 и проведем квадратичную интерполяцию $f(x)$ по узловым точкам $\big(x_1, f(x_1)\big)$, $\big(x_2, f(x_2)\big)$ и $\big(x_3, f(x_3)\big)$. Пусть значение x_4 минимизирует интерполяционный полином. Теперь исключим из рассмотрения значение x_1 и второй шаг итераций проведем с узловыми точками $\big(x_2, f(x_2)\big)$, $\big(x_3, f(x_3)\big)$ и $\big(x_4, f(x_4)\big)$. Продолжая итерационный процесс, на шаге с номером $k-2$ имеем узловые точки $\big(x_{k-2}, f(x_{k-2})\big)$, $\big(x_{k-1}, f(x_{k-1})\big)$ и $\big(x_k, f(x_k)\big)$. Их интерполирует полином Лагранжа

$$P_2(x)=f_{k-2}\frac{(x-x_{k-1})(x-x_k)}{(x_{k-2}-x_{k-1})(x_{k-2}-x_k)}+f_{k-1}\frac{(x-x_{k-2})(x-x_k)}{(x_{k-1}-x_{k-2})(x_{k-1}-x_k)}+$$
$$+f_k\frac{(x-x_{k-2})(x-x_{k-1})}{(x_k-x_{k-2})(x_k-x_{k-1})},$$

где $f_i=f(x_i)$. Значение x_{k+1}, минимизирующее этот полином, найдем, решив уравнение $P_2'(x_{k+1})=0$:

$$x_{k+1}=\frac{1}{2}\frac{f_{k-2}(x_k^2-x_{k-1}^2)-f_{k-1}(x_k^2-x_{k-2}^2)+f_k(x_{k-1}^2-x_{k-2}^2)}{f_{k-2}(x_k-x_{k-1})-f_{k-1}(x_k-x_{k-2})+f_k(x_{k-1}-x_{k-2})}. \qquad (3.2)$$

Вычисления по формуле (3.2) заканчиваются, когда модуль разности между двумя последовательными приближениями достигает заданной малой величины. Описанный алгоритм называется *последовательной*

параболической интерполяцией. Доказано, что в области, где $f''(x) > 0$, он сходится к точке минимума со скоростью приблизительно равной 1,324, т.е. $|x_{n+1} - x_*| \le C|x_n - x_*|^{1,324}$.

Для функций, не являющихся унимодальными, метод последовательной параболической интерполяции может сходиться к точкам как минимумов, так и максимумов. Кроме того, возможна локализация точек перегиба функции, в которых $f'(x) = 0$ и $f''(x) = 0$. В этом случае, как и в методе Ньютона, начальные приближения следует выбирать вблизи точки искомого минимума.

3.4. Метод золотого сечения

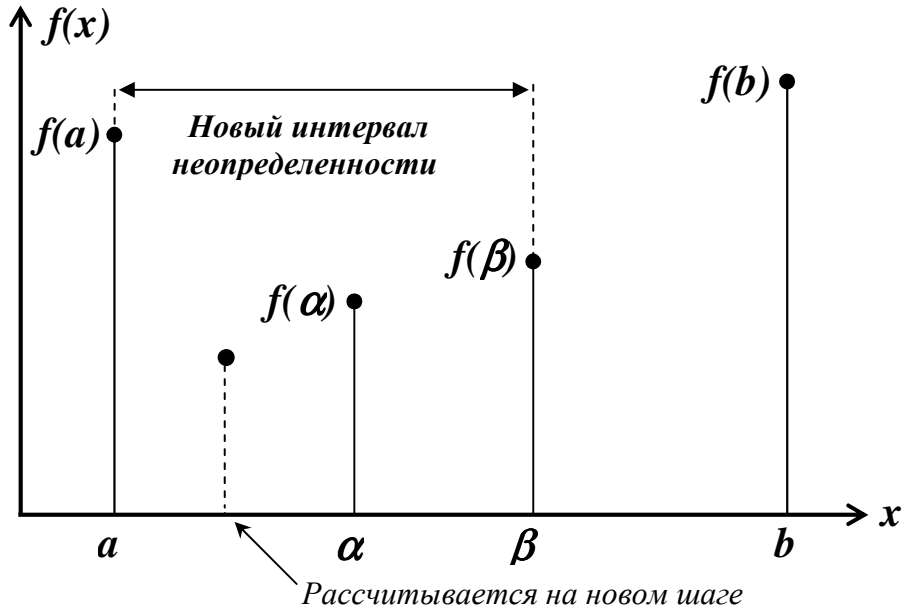
Помимо рассмотренных выше методов, основанных на аппроксимации минимизируемой функции, существует также группа методов прямого поиска, в основе которых лежит определенный способ систематического сужения интервала значений аргумента, заключающего точку минимума – *интервала неопределенности*. Для унимодальных функций методы прямого поиска характеризуются гарантированной сходимостью, хотя и являются достаточно медленными. Из них мы рассмотрим наиболее широко используемый метод *золотого сечения*.

Пусть минимизируемая функция $f(x)$ является унимодальной на отрезке $[a_0, b_0]$, который можно рассматривать как исходный интервал неопределенности. Вычислив значения $f(\alpha)$ и $f(\beta)$ в двух его внутренних точках $\alpha < \beta$, можно сузить интервал неопределенности. Действительно, если $f(\alpha) > f(\beta)$, то точка минимума унимодальной функции находится на отрезке $[\alpha, b_0]$. Теперь он является новым интервалом неопределенности. В противном случае интервал неопределенности – отрезок $[a_0, \beta]$. В любом случае одна из точек α или β остается внутренней точкой нового интервала неопределенности. Ее целесообразно использовать на следующем шаге итерационного процесса локализации минимума, что позволяет вычислять уже не два новых значения функции, а только одно. Это возможно, если новый интервал неопределенности точками α или β разбивается на отрезки в том же отношении, что и исходный интервал.

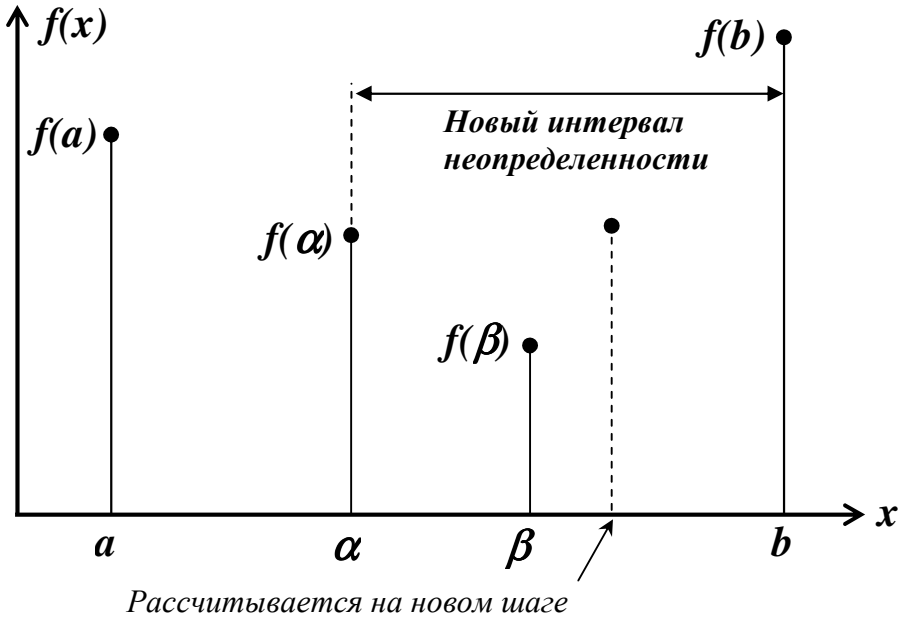
Вопрос о разбиении интервала неопределенности является центральным, поэтому его следует рассмотреть подробнее. Как было сказано выше, после первого шага итераций интервал неопределенности стягивается в отрезок $[\alpha, b]$ или $[a, \beta]$. Так как точка минимума с равной вероятностью может оказаться на каждом из этих отрезков, то коэффициент дробления интервала неопределенности для них должен быть одним и тем же:

$$\frac{b-\alpha}{b-a}=\frac{\beta-a}{b-a}=\xi\,. \tag{3.3}$$

Далее без потери общности можно допустить, что $f(\alpha)<f(\beta)$, т.е. отрезок $[a,\beta]$ – новый интервал неопределенности, а точка α – внутренняя точка интервала. Рис. 3.1 *а* схематически отображает данную ситуацию.



а)



б)

Рис. 3.1. Варианты деления интервала неопределенности

Так как после второго шага итераций интервал неопределенности может стянуться в отрезок $[a,\alpha]$, то коэффициент дробления следует записать в виде отношения

$$\frac{\alpha - a}{\beta - a} = \xi. \tag{3.4}$$

Равенства (3.3) и (3.4) позволяют однозначно определить величину ξ. Для этого числитель дроби (3.4) преобразуем к виду

$$\alpha - a = b - a - (b - \alpha) = b - a - (\beta - a).$$

Здесь второе равенство записано с учетом того, что $b - \alpha = \beta - a$, как это следует из соотношения (3.3). Теперь вместо (3.4) нетрудно записать выражение

$$\frac{1}{\xi} - 1 = \xi,$$

которое после приведения к общему знаменателю принимает стандартную форму квадратного уравнения относительно коэффициента дробления интервала неопределенности:

$$\xi^2 + \xi - 1 = 0.$$

Уравнение имеет положительный корень

$$\xi = \frac{\sqrt{5} - 1}{2} \approx 0{,}618.$$

Теперь, воспользовавшись равенствами (3.3), нетрудно определить положение точек α и β:

$$\begin{aligned} \alpha &= \xi a + (1 - \xi) b, \\ \beta &= (1 - \xi) a - \xi b. \end{aligned} \tag{3.5}$$

Таким образом, основными в итерационном процессе метода золотого сечения являются следующие этапы.

1. Если $f(\alpha) < f(\beta)$, то $b = \beta$, $\beta = \alpha$, $f(\beta) = f(\alpha)$, а значение α вычисляется по первой формуле (3.5) (см. рис. 3.1 *а*).
2. Если $f(\alpha) > f(\beta)$, то $a = \alpha$, $\alpha = \beta$, $f(\alpha) = f(\beta)$, а значение β вычисляется по второй формуле (3.5) (см. рис. 3.1 *б*).

После выполнения *N* итераций, на которых придется вычислить *N*+1 значение минимизируемой функции, ширина интервала неопределенности d_N будет равна

$$d_N = 0{,}618^N d_0,$$

где d_0 – ширина исходного интервала. Итерации прекращаются при достижении величиной d_N заданной точности вычислений. В качестве оценки точки минимума функции можно принять середину последнего интервала неопределенности. Полная блок-схема алгоритма метода золотого сечения приведена на рис. 3.2.

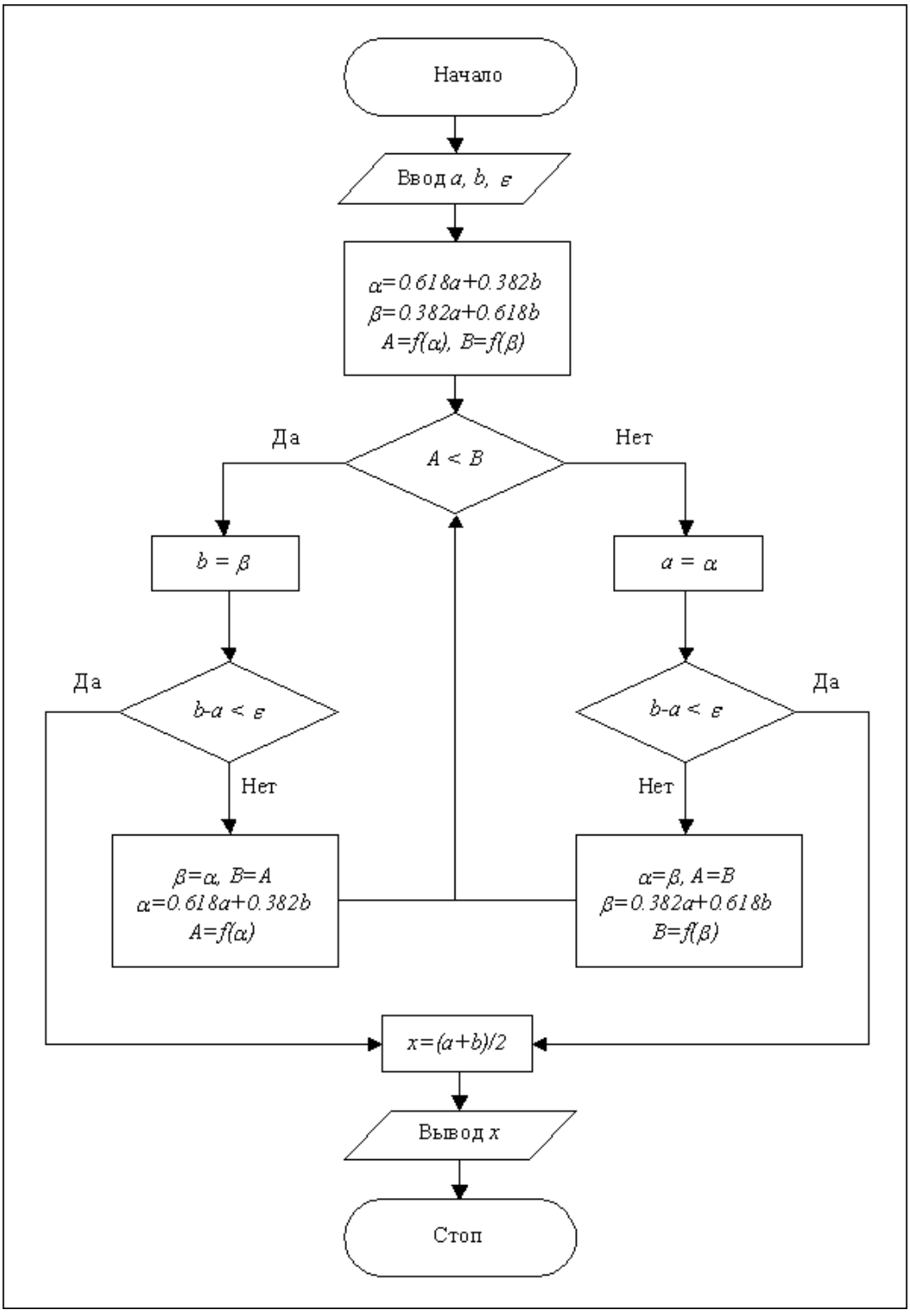


Рис. 3.2. Блок-схема алгоритма метода золотого сечения

Заметим, что золотым сечением называется такое деление отрезка на две неравные части, при котором отношение длины наименьшей части к длине наибольшей равно отношению длины наибольшей части к длине всего отрезка. Пользуясь формулами (3.5), нетрудно показать, что на каждой итерации α и β являются точками золотого сечения для интервала неопределенности. Это обстоятельство и определило название описанного метода оптимизации.

Итерации по методу золотого сечения гарантированно сходятся к минимуму унимодальной функции, однако эта сходимость довольно медленная (линейная). При разработке компьютерных программ оптимизации хорошим решением является комбинация метода золотого сечения с более «быстрым», но не всегда сходящимся методом.

3.5. Оптимизация методом установления

Если рассматривать точку минимума функции $f(x)$ как координату положения равновесия некоторой физической системы, можно предложить физически обоснованный метод оптимизации, который заключается в исследовании процесса перехода системы из начального состояния в состояние равновесия, т.е. процесса установления равновесия.

Рассмотрим плоское движение материальной точки массы m по кривой $y = f(x)$ в поле силы тяжести с ускорением свободного падения **g**. Вектор **g** направлен противоположно оси y, как показано на рис. 3.3.

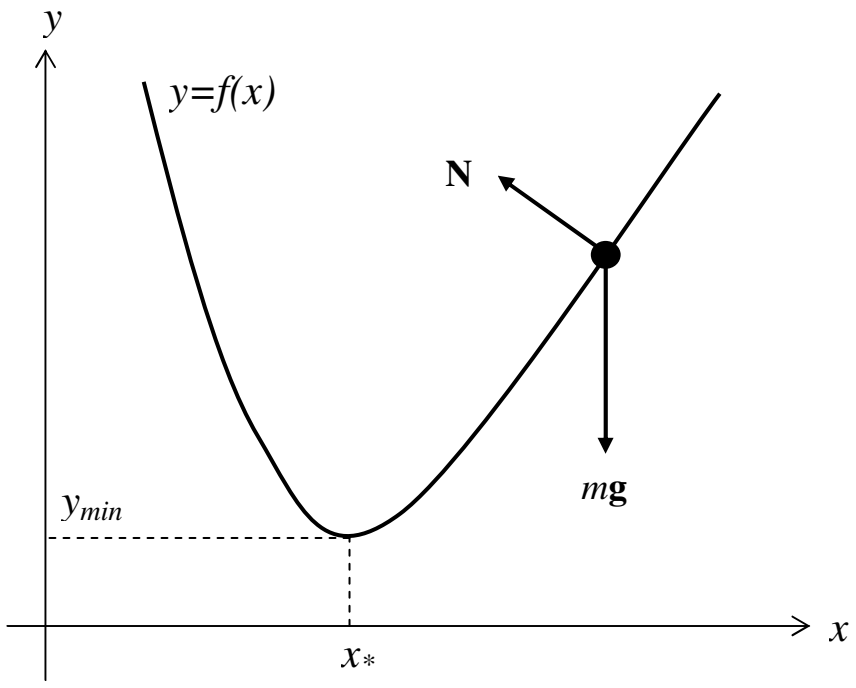


Рис. 3.3. Движение материальной точки по плоской кривой

Кроме силы тяжести $m\mathbf{g}$ на материальную точку действует также сила реакции связи **N**. Поэтому уравнение движения точки имеет вид

$$m\frac{d^2\mathbf{r}}{dt^2} = m\mathbf{g} + \mathbf{N}, \tag{3.6}$$

где **r** – радиус-вектор точки.

Движение по оси x описывает проекция уравнения (3.6) на соответствующую ось:

$$m\frac{d^2x}{dt^2} = -N\sin\alpha. \tag{3.7}$$

Здесь α – угол, который составляет с осью x касательная к кривой $y = f(x)$ в текущей точке. Величина силы реакции связи находится из условия отсутствия перемещений по нормали к кривой $y = f(x)$: $N = mg\cos\alpha$. Учитывая, что

$$\sin\alpha = \frac{f'(x)}{\sqrt{1+[f'(x)]^2}} \quad \text{и} \quad \cos\alpha = \frac{1}{\sqrt{1+[f'(x)]^2}},$$

уравнение движения (3.7) преобразуем к виду

$$\frac{d^2x}{dt^2} + g\frac{f'(x)}{1+[f'(x)]^2} = 0.$$

В это уравнение добавим еще слагаемое $2\delta\, dx/dt$, учитывающее силу трения, пропорциональную скорости. Тогда оно примет вид

$$\frac{d^2x}{dt^2} + 2\delta\frac{dx}{dt} + g\frac{f'(x)}{1+[f'(x)]^2} = 0. \tag{3.8}$$

В окрестности точки минимума функции ее производная $f'(x) << 1$, и вместо (3.8) можно использовать более простое уравнение:

$$\frac{d^2x}{dt^2} + 2\delta\frac{dx}{dt} + gf'(x) = 0. \tag{3.9}$$

Физический опыт показывает, что с течением времени рассматриваемая система примет положение равновесия в точке минимума функции $f(x)$. Это можно установить и строго математически, исследуя уравнение (3.9). Действительно, умножив уравнение (3.9) на производную dx/dt и перегруппировав слагаемые, получим соотношение

$$\frac{d}{dt}\left[\frac{1}{2}\left(\frac{dx}{dt}\right)^2 + gf(x)\right] = -2\delta\left(\frac{dx}{dt}\right)^2 < 0,$$

из которого следует, что в процессе движения по кривой $y = f(x)$ значения функции

$$W(x, \dot{x}) = \frac{1}{2}\left(\frac{dx}{dt}\right)^2 + gf(x)$$

непрерывно уменьшаются. Но минимум функции $W(x)$ достигается в точке минимума функции $f(x)$ при нулевой скорости dx/dt:

$$W_{\min} = gf_{\min} = gf(x_*) = W(x_*, 0).$$

Следовательно, движение материальной точки приводит ее в точку x_*, где она будет находиться в состоянии покоя. Заметим, что введенная в рассмотрение функция $W(x, \dot{x})$ есть не что иное, как полная механическая энергия материальной точки.

Итак, положение минимума функции $f(x)$ можно найти, решая дифференциальное уравнение (3.9). При этом от реального физического времени t в уравнении целесообразно перейти к безразмерному времени $\tau = \omega_0 t$, используя для нормировки частоту колебаний около положения равновесия $\omega_0 = \sqrt{gf''(x_*)}$. После нормировки времени уравнение (3.9) примет вид

$$\frac{d^2x}{d\tau^2} + \frac{1}{Q}\frac{dx}{d\tau} + \frac{f'(x)}{f''(x_*)} = 0, \qquad (3.10)$$

где $Q = \omega_0/(2\delta)$ – добротность малых колебаний в окрестности точки x_*.

Для малых колебаний удается определить оптимальную величину добротности, обеспечивающую наибольшую скорость установления состояния равновесия в системе (3.10). В этом случае применима аппроксимация

$$f(x) = f(x_*) + \frac{1}{2}f''(x_*)(x - x_*)^2,$$

с учетом которой уравнение (3.10) становится линейным:

$$\frac{d^2x}{d\tau^2} + \frac{1}{Q}\frac{dx}{d\tau} + x = x_*.$$

При начальных условиях $x(0) = x_0$, $\dot{x}(0) = 0$ оно имеет решение

$$x(\tau) = x_* + \frac{x_0 - x_*}{\lambda_2 - \lambda_1}\left(\lambda_2 \exp(\lambda_1\tau) - \lambda_1 \exp(\lambda_2\tau)\right). \qquad (3.11)$$

Здесь $\lambda_1 = -\frac{1}{2Q} + \sqrt{\frac{1}{4Q^2} - 1}$ и $\lambda_2 = -\frac{1}{2Q} - \sqrt{\frac{1}{4Q^2} - 1}$ – корни характеристического уравнения

$$\lambda^2 + \frac{1}{Q}\lambda + 1 = 0.$$

Скорость установления процесса (3.11) определяется показателем

$$\gamma(Q) = \max\{\mathrm{Re}(\lambda_1), \mathrm{Re}(\lambda_2)\}.$$

Нетрудно найти, что наибольшая скорость соответствует значению

$$\gamma_{\min} = \min_{Q}(\gamma(Q)) = -1,$$

которое достигается при $Q = 0{,}5$.

Однако этот результат, указывающий на существование оптимального значения добротности, важен скорее в теоретическом плане. А так как величина $f''(x_*)$ в уравнении (3.10) заранее неизвестна, то на практике можно положить $f''(x_*) = 1$ и численно решать уравнение

$$\frac{d^2x}{d\tau^2} + \frac{1}{Q}\frac{dx}{d\tau} + f'(x) = 0, \qquad (3.12)$$

экспериментально подбирая значение Q, обеспечивающее приемлемую скорость сходимости. Именно уравнение (3.12) и является основой одного из вариантов метода установления.

Другой вариант метода получим, предположив, что сила трения при движении материальной точки по кривой $y = f(x)$ намного превосходит силу инерции. Это означает, что Q очень мало и в уравнении (3.12) можно пренебречь первым слагаемым по сравнению со вторым. При этом дифференциальное уравнение метода установления примет вид

$$\frac{dx}{d\tau} = -Qf'(x). \qquad (3.13)$$

Это уравнение первого порядка и его численное решение можно получить с меньшим количеством вычислений, чем решение уравнения (3.12). Но может оказаться, что скорость сходимости численного решения к стационарному значению для уравнения (3.13) ниже, чем для (3.12).

Вопрос о решении дифференциальных уравнений (3.12), (3.13) – это отдельная большая тема численного анализа. Здесь мы ограничимся решениями с использованием метода конечных разностей. Разностная аппроксимация уравнения (3.12) на временной сетке с постоянным шагом $\Delta\tau$ имеет вид

$$\frac{x_{n+1}-2x_n+x_{n-1}}{\Delta\tau^2}+\frac{1}{Q}\frac{x_{n+1}-x_{n-1}}{2\Delta\tau}+f'(x_n)=0.$$

Это выражение после приведения подобных членов и группировки слагаемых преобразуется в итерационную формулу

$$x_{n+1}=x_n+\nu(x_n-x_{n-1})+\mu f'(x_n) \tag{3.14}$$

с коэффициентами

$$\nu=\frac{2-Q^{-1}\Delta\tau}{2+Q^{-1}\Delta\tau},\quad \mu=-\frac{2\Delta\tau^2}{2+Q^{-1}\Delta\tau}.$$

На значения коэффициентов ν и μ помимо добротности Q влияет также величина шага $\Delta\tau$. При выборе $\Delta\tau$ следует учитывать, что очень малые шаги в процессе установления приводят к слишком большому количеству вычислений. С другой стороны, при большой величине шага численное решение (3.14) дифференциального уравнения (3.12) может оказаться слишком неточным или вообще неустойчивым. Фактически вопрос о выборе подходящей величины шага должен решаться в каждом конкретном случае.

Разностная аппроксимация уравнения (3.13) приводит к итерационной формуле

$$x_{n+1}=x_n+\sigma f'(x_n)$$

с коэффициентом

$$\sigma=-Q\Delta\tau,$$

которая может рассматриваться как частный случай формулы (3.14). Например, выбрав параметры Q и $\Delta\tau$ так, что $\nu=0$, получим $\mu=\sigma=-\Delta\tau^2/2$.

Если при расчетах по методу (3.14) желательно не использовать аналитического выражения для производной $f'(x)$, то ее можно вычислять так же, как и в методе секущих (1.13). В таком случае вместо (3.14) получим формулу

$$x_{n+1}=x_n+\nu(x_n-x_{n-1})+\mu\frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}. \tag{3.15}$$

Метод установления применим не только к задачам оптимизации, но и к другим стационарным задачам, в частности, к решению нелинейных уравнений $f(x)=0$. В этом случае можно минимизировать функцию $F(x)=f^2(x)$, точка нулевого минимума которой совпадает с корнем исходного уравнения. Итерационная формула при этом принимает вид

$$x_{n+1}=x_n+\nu(x_n-x_{n-1})+2\mu f(x_n)\frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}}.$$

Завершая обсуждение метода установления отметим, что его использование для решения задач одномерной оптимизации, возможно, не вляется полностью оправданным. Эти довольно простые задачи при современном уровне развития вычислительной техники с успехом могут быть решены прямыми методами, например, методом золотого сечения. Но для нас, метод установления интересен тем, что демонстрирует возможность построения математических алгоритмов на основе теоретических представлений о процессах, протекающих в физических системах. А для многомерных задач алгоритмы установления интересны и с точки зрения практического применения.

Литература

1. Бахвалов, Н.С. Численные методы [Текст] / Н.С.Бахвалов, Н.П.Жидков, Г.М.Кобельков. – 3-е изд., доп. и перераб. – М.: БИНОМ. Лаборатория знаний, 2004.
2. Васильев, Н.Ф. Численные методы решения экстремальных задач [Текст] / Н.Ф.Васильев. – М.: Наука, 1980.
3. Турчак, Л.И. Основы численных методов [Текст] / Л.И.Турчак, П.В.Плотников. – М.: ФИЗМАТЛИТ, 2002.
4. Банди, Б. Методы оптимизации. Вводный курс [Текст] / Б.Банди. – М.: Радио и связь, 1988.
5. Мэтьюз, Д. Численные методы. Использование MATLAB [Текст] / Д.Мэтьюз, К.Финк. – М.: Издательский дом «Вильямс», 2001.
6. Шуп, Т. Решение инженерных задач на ЭВМ: Практическое руководство [Текст] / Т.Шуп. – М.: Мир, 1982.

Приложение 3
Программы одномерной оптимизации

Приведем несколько программ, позволяющих решать задачи минимизации функции одной переменной в среде MathCAD.

Программа *Interval* предназначена для поиска интервала неопределенности, содержащего локальный минимум функции. Заголовок программного модуля имеет вид

***Interval(F,x0,h)*,**

где F – имя минимизируемой функции, $x0$ – начальная точка поиска, h – шаг поиска. Текст модуля приведен на рис. П.3.1. Программа возвращает двумерный вектор-столбец, компонентами которого являются левая и правая границы интервала неопределенности шириной $2h$.

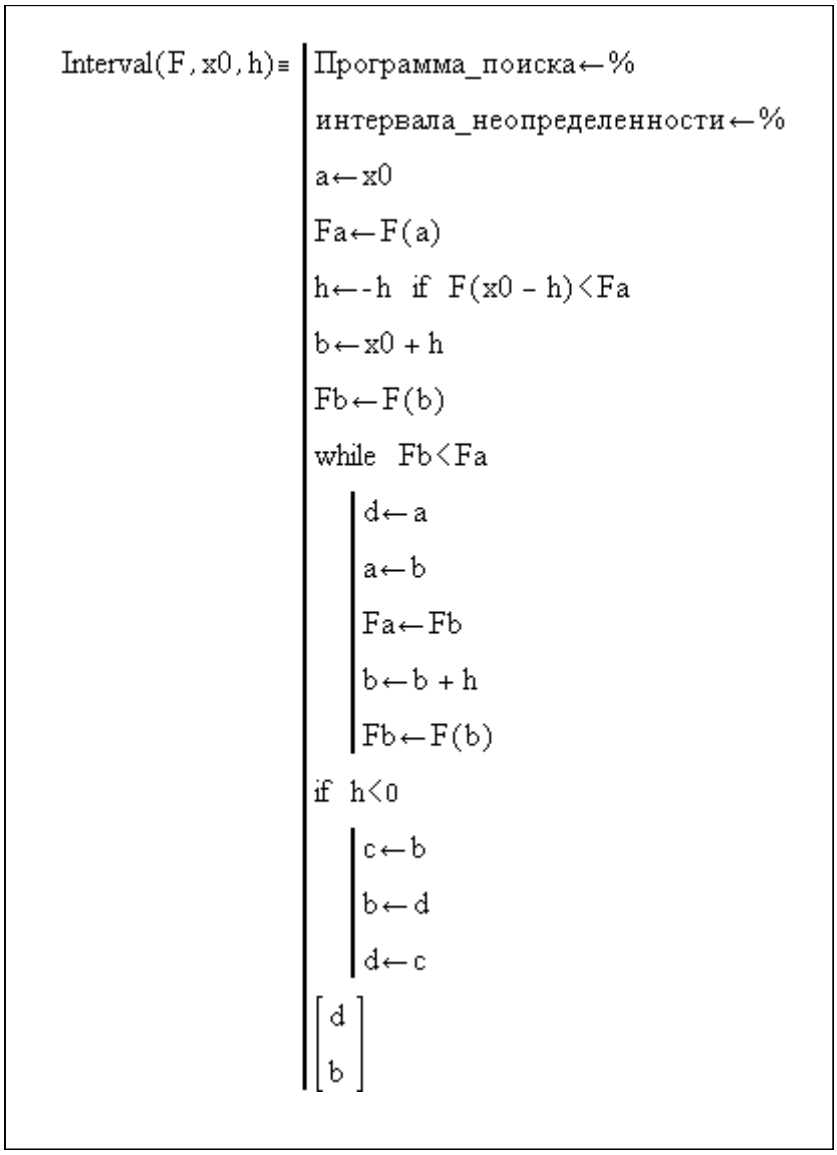


Рис. П.3.1. Программа поиска интервала неопределенности

Программные модули

Min_GS(F,x0,ε) и ***Min_SPI(F,x0,ε)***

реализуют вычисления по методу золотого сечения и методу обратной параболической интерполяции. Аргументы модулей: *F* – имя минимизируемой функции, *x0* – начальное приближение к точке минимума, ε – точность. Программы возвращают приближенное значение координаты минимума функции. Тексты программных модулей представлены на рис. П.3.2 и рис. П3.3.

Программа *Setting* реализует алгоритм метода установления (3.15). Заголовок программного модуля имеет вид

Setting(F,x0,Q,Δτ),

где аргументы *F* и *x0* имеют тот же смысл, что и ранее, а добротность *Q* и шаг $\Delta\tau$ задаются в соответствии с изложенным в п. 3.5. Текст программного модуля содержит рис. П.3.4.

```
Min_GS(F, x0, ε) := | Программа_минимизации ← %
                    | методом_золотого_сечения ← %
                    | h ← 0.101
                    | ξ ← (√5 − 1) / 2
                    | A ← Interval(F, x0, h)
                    | a ← A_0
                    | b ← A_1
                    | α ← ξ·a + (1 − ξ)·b
                    | β ← (1 − ξ)·a + ξ·b
                    | Fα ← F(α)
                    | Fβ ← F(β)
                    | while 1
                    |     if Fα < Fβ
                    |         b ← β
                    |         β ← α
                    |         Fβ ← Fα
                    |         α ← ξ·a + (1 − ξ)·b
                    |         Fα ← F(α)
                    |     if Fα > Fβ
                    |         a ← α
                    |         α ← β
                    |         Fα ← Fβ
                    |         β ← (1 − ξ)·a + ξ·b
                    |         Fβ ← F(β)
                    |     return (a + b) / 2  if |b − a| ≤ ε
```

Рис. П.3.2. Программа одномерной оптимизации методом золотого сечения

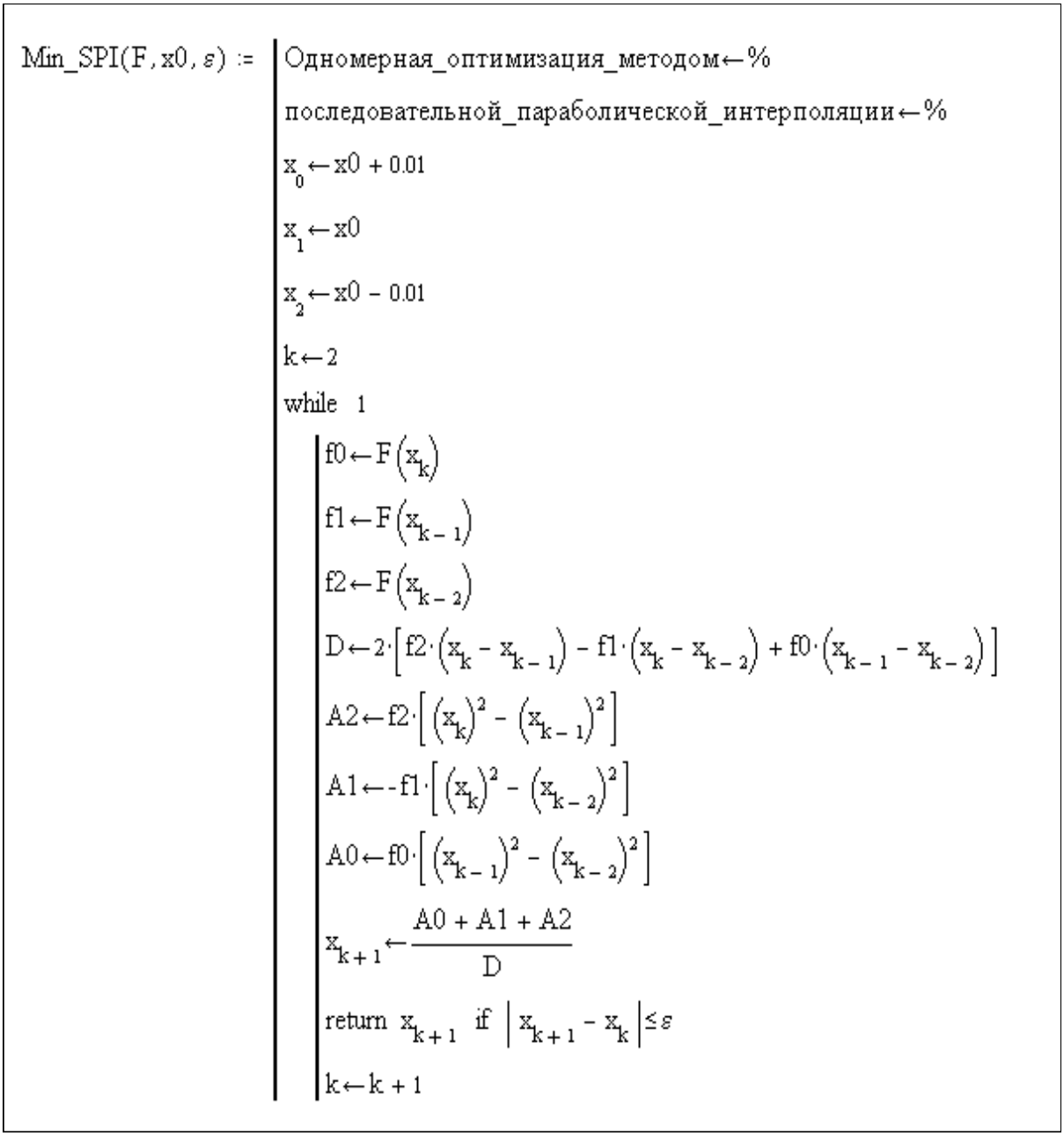


Рис. П.3.3. Программа одномерной оптимизации методом последовательной параболической интерполяции

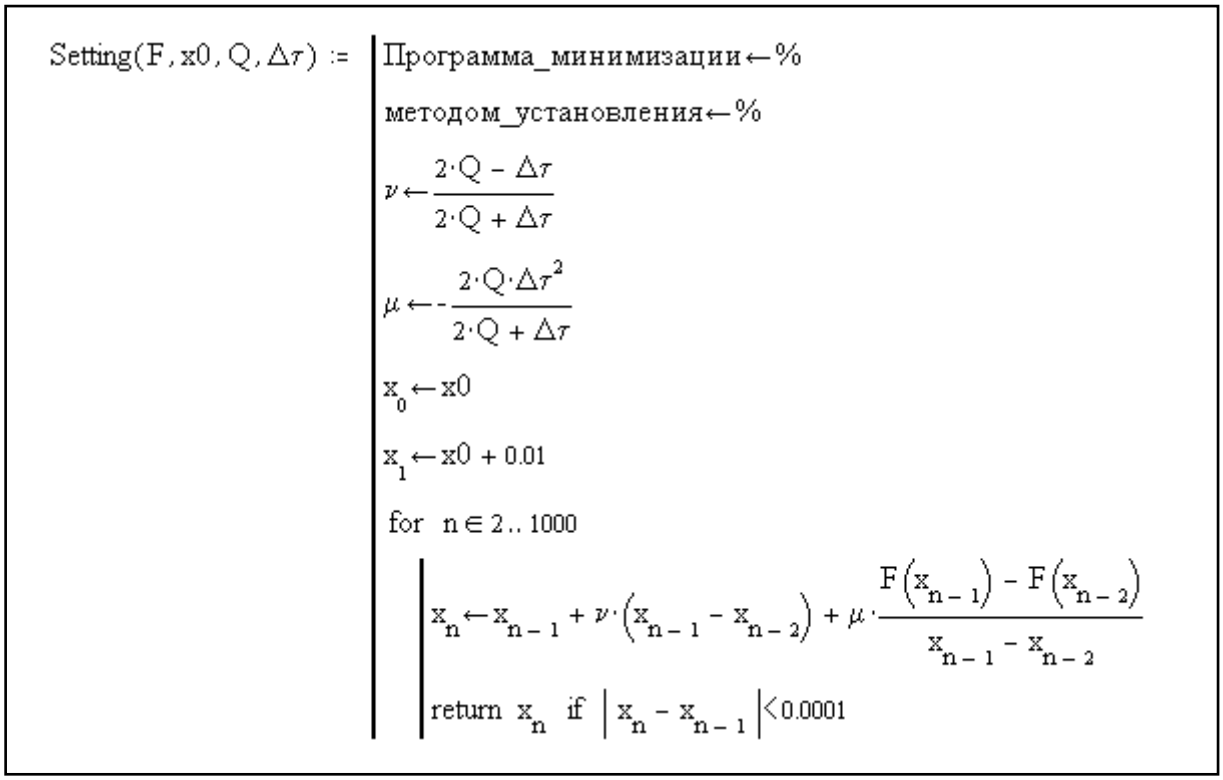


Рис. П.3.4. Программа одномерной оптимизации методом установления

Пример обращения к программным модулям приведен на рис. П.3.5.

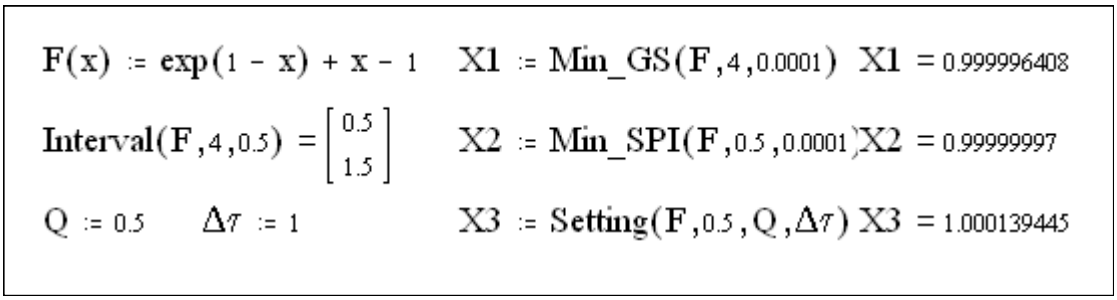


Рис. П.3.5. Обращения к программам одномерной оптимизации

ГЛАВА 4

МНОГОМЕРНАЯ ОПТИМИЗАЦИИ

4.1. Введение

В практике научных исследований и технических разработок задачи многомерной оптимизации, состоящие в поиске минимумов функций нескольких переменных, встречаются гораздо чаще, чем задачи, сводящиеся к минимизации функции одного аргумента. Можно привести множество примеров задач из различных отраслей науки и техники, при решении которых методы оптимизации играют центральную роль в получении конкретного числового результата.

Мы ограничимся здесь всего лишь одним примером из области обработки результатов физического эксперимента методом наименьших квадратов. Для определенности будем рассматривать результаты измерения амплитудно-частотной характеристики резонатора. Данные измерений представлены в виде ряда значений амплитуд колебаний a_i на частотах ν_i. Теоретически амплитуда вынужденных колебаний высокодобротного резонатора описывается выражением

$$a(\nu) = \frac{a_{in}}{2\sqrt{\left(1-\frac{\nu}{\nu_r}\right)^2 + \frac{1}{4Q^2}}},$$

где ν_r и Q – резонансная частота и добротность колебательной системы, a_{in} – амплитуда внешнего воздействия. Значения этих трех параметров теоретической модели неизвестны *a priori* и должны быть определены по результатам эксперимента. Для решения этой задачи подходит метод наименьших квадратов. В рамках метода составляется целевая функция

$$f(a_{in}, \nu_r, Q) = \sum_{m=1}^{M} \left(a(\nu_m) - a_m\right)^2 ,$$

минимизируемая в трехмерном пространстве параметров a_{in}, ν_r, Q.

Традиционно методы многомерной оптимизации, т.е. поиска положения минимума (или максимума) функции многих аргументов делят на две группы – прямые и градиентные. В прямых методах для поиска минимума сравниваются вычисляемые значения целевой функции в различных точках многомерного пространства. Градиентные методы основаны на использовании дополнительной информации о положении точки минимума, содержащейся в значениях производных целевой функции. В данной главе мы рассмотрим наиболее распространенные

алгоритмы решения задач многомерной оптимизации, как прямые, так и градиентные.

4.2. Метод покоординатного спуска

Этот метод относится к группе прямых методов и основан на многократном применении алгоритмов одномерной оптимизации. Стратегия метода – постепенное приближение к точке минимума функции путем последовательных вариаций одной из координат при фиксированных значениях остальных. Рассмотрим алгоритм метода подробнее.

Пусть в *n*–мерном пространстве задана точка $X^{(0)}$ с координатами $x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)}$, являющаяся точкой начального приближения к минимуму функции $f(x_1, x_2, ..., x_n)$. Зафиксируем все координаты, кроме первой – x_1. Получим $f_1(x_1) = f(x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})$ – функцию одной переменной. Для функции $f_1(x_1)$ решим задачу одномерной оптимизации и найдем значение $x_1^{(1)}$ – первую координату точки первого приближения к минимуму. Зафиксируем теперь все координаты, кроме x_2, и решим задачу одномерной оптимизации для функции $f_2(x_2) = f(x_1^{(1)}, x_2, ..., x_n^{(0)})$. Результатом решения будет $x_2^{(1)}$ – вторая координата точки первого приближения к минимуму. Продолжая описанный процесс перебора переменных, получим все координаты $x_1^{(1)}, x_2^{(1)}, ..., x_n^{(1)}$ точки первого приближения $X^{(1)}$. На этом первая итерация алгоритма завершается.

Вторая и все последующие итерации алгоритма организуются аналогичным образом. Итерационный процесс завершается при выполнении условия близости точек, найденных на двух последовательных итерациях с номерами *I* и *I+1*:

$$\sqrt{\sum_{k=1}^{n} (x_k^{(i+1)} - x_k^{(i)})^2} \le \varepsilon, \qquad (4.1)$$

где ε – малая величина, характеризующая точность расчетов.

Задача одномерной оптимизации в рамках метода покоординатного спуска может быть решена одним из описанных ранее методов, например, методом последовательной параболической интерполяции или методом золотого сечения.

При минимизации функции двух переменных рассматриваемый метод легко проиллюстрировать геометрически. В этом случае функция $f(x_1, x_2)$ описывает некоторую поверхность в трехмерном пространстве. На рис. 4.1 изображены линии уровня этой поверхности. Процесс минимизации выглядит следующим образом. Точка $X^{(0)}$ задает начальное приближение. Двигаясь сначала параллельно оси x_1 (спуск по координате x_1), а затем параллельно оси x_2 (спуск по координате x_2), мы попадаем в точку первого приближения $X^{(1)}$. Проведя на второй итерации еще два покоординатных спуска, окажемся в точке второго приближения $X^{(2)}$ и т.д.

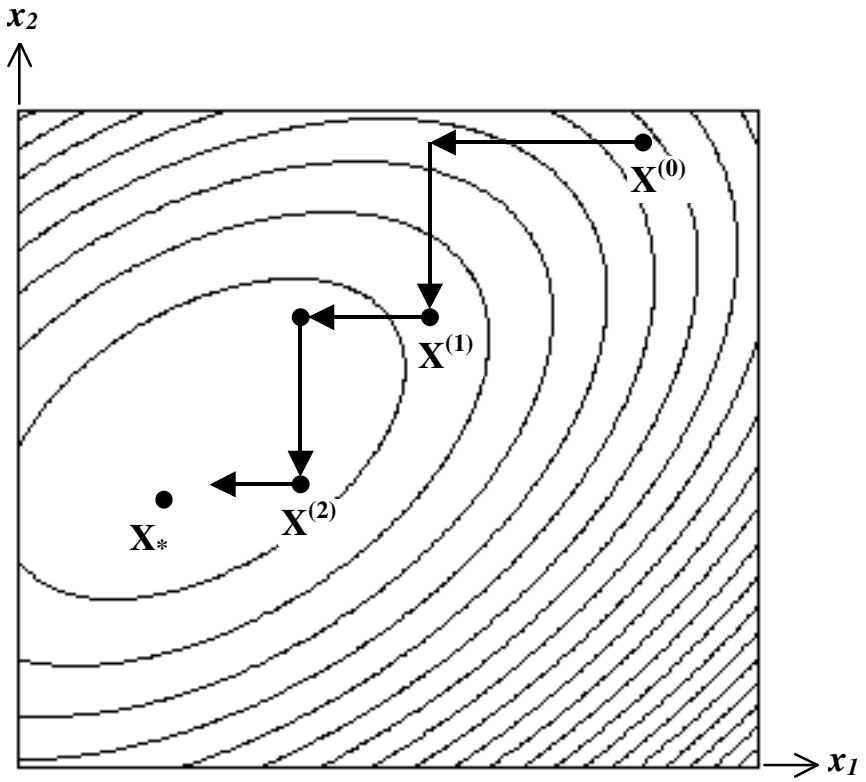


Рис. 4.1. Геометрическая иллюстрация метода покоординатного спуска

Важным является вопрос о сходимости описанного итерационного процесса. Ответ на него зависит от конкретного вида целевой функции и удачного выбора начального приближения. Для гладких функций при начальном приближении из окрестности локального минимума последовательность $X^{(0)}, X^{(1)}, X^{(2)}, \ldots$ сходится к оптимальной точке. Однако и здесь применение метода затруднено наличием так называемых *оврагов*.

Овраг представляет собой впадину, линии уровня которой приближенно имеют форму эллипсов с различающимися во много раз полуосями. При наличии оврага траектория спуска имеет вид зигзагообразной линии с малым шагом, вследствие чего результирующая скорость спуска к минимуму сильно замедляется. Алгоритм теряет

вычислительную эффективность. Характерным примером овражной целевой функции является функция Розенброка

$$f_R(x_1, x_2) = (x_1 - 1)^2 + 100(x_2 - x_1^2)^2, \qquad (4.2)$$

которая часто используется при тестировании алгоритмов оптимизации.

Метод покоординатного спуска совершенно неприменим в тех случаях, когда линии уровня целевой функции имеют точки излома. Одним из многочисленных примеров может служить функция

$$f(x_1, x_2) = \left| x_1^2 - x_2 + 0{,}5 \right| + (x_2 - 1)^2$$

с линиями уровня, изображенными на рис. 4.2. Для метода покоординатного спуска точки излома являются тупиковыми, т.е. из них невозможно продвижение в направлениях, параллельных координатным осям. В то же время, рис. 4.2 показывает, что движение вдоль линии, соединяющей точки излома, – линии излома, наиболее быстро приводит в точку минимума. Это свойство линий излома используют некоторые методы многомерной оптимизации, в частности, метод Хука–Дживса и метод конфигураций Розенброка.

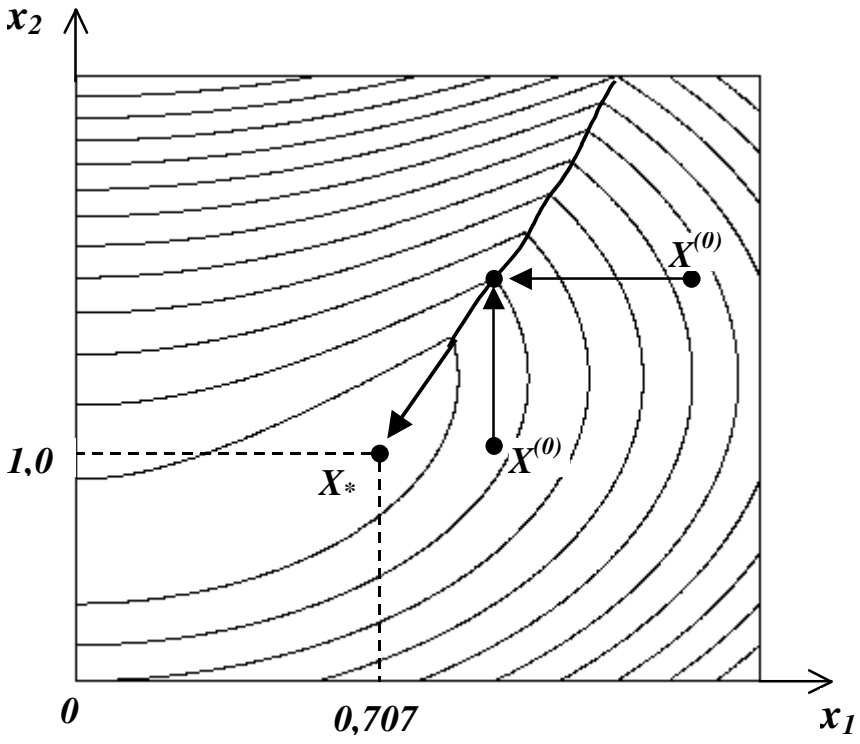


Рис. 4.2. Изломы линий уровня

Достоинство метода покоординатного спуска в том, что он позволяет применять сравнительно простые алгоритмы одномерной оптимизации,

недостаток – медленная сходимость. Поэтому его часто используют на начальном этапе решения задачи, переходя затем к более сложным, но и более быстрым методам.

4.3. Градиентные методы. Метод наискорейшего спуска

Алгоритмы оптимизации, использующие для поиска минимума не только значения целевой функции, но и значения ее градиента составляют основу так называемых градиентных методов. Напомним, что вектор градиента

$$\nabla f = grad\ f = \frac{\partial f}{\partial x_1} + \frac{\partial f}{\partial x_2} + ... + \frac{\partial f}{\partial x_n}$$

перпендикулярен линии (поверхности) уровня и направлен в сторону возрастания функции. Направление градиента в рассматриваемой точке есть направление наибольшего возрастания функции.

Один из градиентных методов, носящий название метода *градиентного спуска*, основан на смещении на постоянный шаг в направлении, противоположном направлению градиента, с последующим вычислением значения целевой функции в полученной точке. Если это значение оказывается меньше предыдущего, то вычисляется градиент в новой точке, и все действия повторяются. При этом величина шага может быть увеличена. Если же значение целевой функции возрастает или не изменяется, то шаг смещения из предыдущей точки уменьшается, и все вычисления повторяются. Итерации продолжаются до тех пор, пока расстояние между точками, полученными на двух последовательных итерациях, не снизится до заданной величины, либо не будет достигнута заданная близость значений целевой функции:

$$\left| f(x_1^{(i+1)}, x_2^{(i+1)}, ..., x_n^{(i+1)}) - f(x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)}) \right| \le \varepsilon . \qquad (4.3)$$

Если компоненты вектора градиента не удается определить аналитическим способом, то можно найти их приближенные значения в рассматриваемой точке, аппроксимируя производные центральными разностями:

$$\frac{\partial f}{\partial x_k} = \frac{f(x_1, x_2, ..., x_k + \Delta, ..., x_n) - f(x_1, x_2, ..., x_k - \Delta, ..., x_n)}{2\Delta},$$

где малое приращение переменной x_k.

Вариантом градиентного спуска является алгоритм, реализованный в методе *наискорейшего спуска*. В этом алгоритме градиент вычисляется далеко не в каждой точке траектории спуска, а лишь там, где достигнут

минимум функции при движении в направлении, заданном на предыдущей итерации. Более детально это выглядит следующим образом.

Спуск из точки начального приближения $X^{(0)}$ производится в направлении $-\,grad\, f(X^{(0)})$ вдоль прямой, заданной уравнением

$$\mathbf{x}(h) = \mathbf{x}^{(0)} - h\, grad\, f(X^{(0)}).$$

В процессе спуска минимизируется функция одного аргумента

$$g^{(0)}(h) = f\left(\mathbf{x}^{(0)} - h\, grad\, f(X^{(0)})\right).$$

Результат одномерной оптимизации – точка $X^{(1)}$ – служит начальной точкой спуска вдоль прямой

$$\mathbf{x}(h) = \mathbf{x}^{(1)} - h\, grad\, f(X^{(1)})$$

на втором шаге метода, когда минимизируется функция

$$g^{(1)}(h) = f\left(\mathbf{x}^{(1)} - h\, grad\, f(X^{(1)})\right).$$

Итерационный процесс продолжается до выполнения условия (4.1). Можно также завершить итерации, если достигнута близость значений целевой функции (4.3). Кроме того, в точке минимума градиент целевой функции равен нулю. Поэтому малость модуля градиента также может служить признаком окончания процесса многомерной оптимизации.

Отметим, что метод наискорейшего спуска, по-видимому, является одним из старейших методов минимизации функций многих аргументов. Его идея была предложена Коши еще в 1845 году. При минимизации функций, поверхности уровня которых близки к многомерным эллипсоидам с полуосями одного порядка величины, метод наискорейшего спуска, равно как и другие градиентные методы, как правило, сходится быстрее, чем метод покоординатного спуска.

Общий недостаток рассмотренных здесь градиентных методов – медленная сходимость при наличии оврагов на поверхности минимизируемой функции. Дело в том, что этом случае локальные градиенты, с которыми оперируют рассмотренные алгоритмы, даже приблизительно не указывают направление к точке минимума. Векторы градиентов сильно осциллируют относительно направления на минимум, в результате чего траектория спуска представляет собой зигзагообразную линию с малым шагом в направлении минимума.

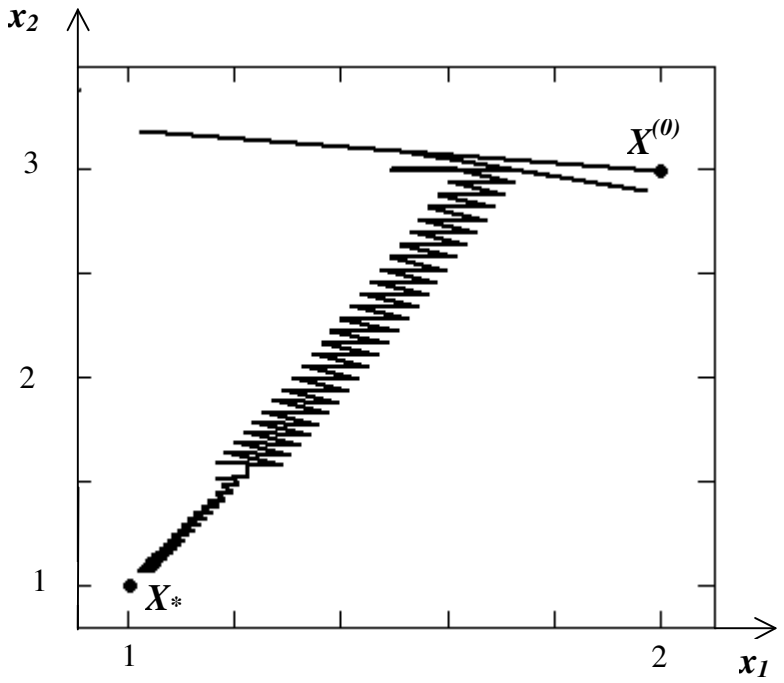


Рис. 4.3. Траектория градиентного спуска

На рис. 4.3 показана траектория градиентного спуска из начальной точки $X^{(0)} = (2;3)$ к минимуму функции $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - x_1^2)^2$, расположенному в точке $X_* = (1;1)$. Поверхность данной целевой функции содержит овраг, правда не столь узкий, как овраг функции Розенброка (4.2). Но и при этом отчетливо проявляется зигзагообразность траектории. Точка $X^{(I)} = (1{,}0002;1{,}0006)$ в рассматриваемом примере достигнута за 163 итерации. Здесь явно возникает задача ускорения сходимости.

4.4. Метод Ньютона. Квазиньютоновские методы

Метод Ньютона для решения задач минимизации функций многих аргументов является обобщением рассмотренного ранее одноименного метода одномерной оптимизации. При некоторых предположениях о характере минимизируемой функции он сходится значительно быстрее, чем методы градиентного спуска.

Обоснование алгоритма многомерного метода Ньютона проведем, рассматривая вначале двумерный случай. Разложение целевой функции $f(x_1, x_2)$ в ряд Тейлора в окрестности точки текущего приближения $X^{(k)}$ имеет вид

$$f(x_1^{(k)}+\xi_1, x_2^{(k)}+\xi_2) = f(x_1^{(k)}, x_2^{(k)}) + \xi_1 \frac{\partial f}{\partial x_1} + \xi_2 \frac{\partial f}{\partial x_2} + \\ + \frac{1}{2}\left(\xi_1^2 \frac{\partial^2 f}{\partial x_1^2} + \xi_1\xi_2 \frac{\partial^2 f}{\partial x_1 \partial x_2} + \xi_2\xi_1 \frac{\partial^2 f}{\partial x_2 \partial x_1} + \xi_2^2 \frac{\partial^2 f}{\partial x_2^2} \right) + \dots . \qquad (4.4\ a)$$

Здесь все производные вычисляются в точке $X^{(k)}$. Выражение (4.4 *а*) можно записать в векторно-матричной форме, если ввести в рассмотрение квадратную матрицу $\mathbf{H}$ (*матрицу Гессе*) с элементами

$$h_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

и вектор-столбец смещений $\boldsymbol{\xi} = (\xi_1, \xi_2)^T$:

$$f(\mathbf{x}^{(k)} + \boldsymbol{\xi}) = f(\mathbf{x}^{(k)}) + \boldsymbol{\xi}^T \nabla f(\mathbf{x}^{(k)}) + \frac{1}{2}\boldsymbol{\xi}^T \mathbf{H}(\mathbf{x}^{(k)})\boldsymbol{\xi} + \dots . \qquad (4.4\ б)$$

Несмотря на то, что этот результат получен для двумерного случая, его векторно-матричная форма (4.4 *б*) сохраняется и для *n* измерений.

Если в выражении (4.4) сохранить лишь слагаемые до второго порядка по $\boldsymbol{\xi}$ включительно, то это означает, что истинную целевую функцию $f(\mathbf{x})$ мы заменяем квадратичной формой

$$Q(\boldsymbol{\xi}) = f(\mathbf{x}^{(k)}) + \boldsymbol{\xi}^T \nabla f(\mathbf{x}^{(k)}) + \frac{1}{2}\boldsymbol{\xi}^T \mathbf{H}(\mathbf{x}^{(k)})\boldsymbol{\xi} \ .$$

Чтобы получить новое приближение к точке минимума целевой функции, минимизируем $Q(\boldsymbol{\xi})$, вычисляя ее градиент по $\boldsymbol{\xi}$:

$$\nabla Q(\boldsymbol{\xi}) = \nabla f(\mathbf{x}^{(k)}) + \mathbf{H}(\mathbf{x}^{(k)})\boldsymbol{\xi} \ ,$$

и приравнивая его к нулю. В результате получаем систему линейных алгебраических уравнений относительно компонент вектора $\boldsymbol{\xi}_*$ – вектора смещения из точки текущего приближения $X^{(k)}$ в точку нового приближения $X^{(k+1)}$:

$$\mathbf{H}(\mathbf{x}^{(k)})\boldsymbol{\xi}_* = -\nabla f(\mathbf{x}^{(k)}) . \qquad (4.5)$$

Уравнения (4.5) называются уравнениями Ньютона. Их решение позволяет определить новое приближение как

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\xi}_* . \qquad (4.6)$$

Если решение уравнений Ньютона (4.5) формально записать через матрицу $\mathbf{H}(\mathbf{x}^{(k)})^{-1}$, обратную матрице Гессе, то

$$\boldsymbol{\xi}_* = -\mathbf{H}(\mathbf{x}^{(k)})^{-1} \nabla f(\mathbf{x}^{(k)}) ,$$

и итерационная формула многомерного метода Ньютона будет выглядеть следующим образом:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{H}(\mathbf{x}^{(k)})^{-1} \nabla f(\mathbf{x}^{(k)}). \qquad (4.7)$$

Однако это не более чем формальная запись алгоритма. Практическая реализация вычислений по методу Ньютона предполагает решение системы уравнений (4.5) без формирования обратной матрицы Гессе в явном виде.

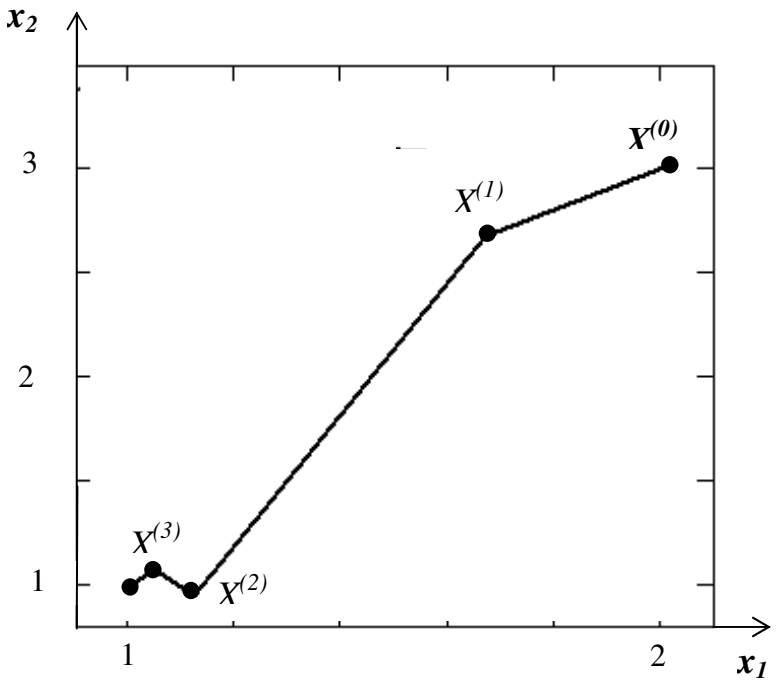


Рис. 4.4. Траектория спуска метода Ньютона

Метод Ньютона для многих измерений обладает тем же свойством квадратичной сходимости в окрестности минимума, что и в одномерном случае. На рис. 4.4 приведена траектория спуска методом Ньютона к минимуму функции $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - x_1^2)^2$, рассмотренной в предыдущем разделе. На переход из точки $X^{(0)} = (2;3)$ в точку $X^{(4)} = (1{,}0005; 0{,}999)$ в этом случае требуется всего четыре итерации.

Однако необходимость вычисления матрицы Гессе существенно затрудняет практическое применение метода Ньютона. Конечно, как и в одномерном случае, можно аппроксимировать вторые производные конечными разностями, но эта операция для n измерений требует n^2 дополнительных вычислений функции, что слишком дорого, если не принимать во внимание простейшие, легко вычисляемые функции.

Еще один существенный недостаток метода заключается в необходимости решения на каждой итерации системы из n линейных

алгебраических уравнений (4.5) с затратой примерно $n^3/6$ арифметических операций. При больших n это неприемлемый объем работы на одну итерацию.

К настоящему времени разработаны модификации метода Ньютона, сохраняющие высокую скорость сходимости, но не требующие вычислений и обращений матрицы Гессе. Эта группа методов носит название *квазиньютоновских*. Широко известен относящийся к указанной группе *метод Дэвидона–Флетчера–Пауэла*. В нем, в отличие от алгоритма (4.7), спуск на каждой итерации проводится в направлении $-\mathbf{B}_k \nabla f(\mathbf{x}^{(k)})$, где $\mathbf{B}_k$ – положительно определенная симметричная матрица, которая обновляется на каждом шаге и в пределе стремится к обратной матрице Гессе.

4.5. Симплексный метод Нелдера–Мида

Симплексом называется n-мерная геометрическая фигура, ребрами которой являются прямые линии, пересекающиеся в $n+1$ вершине. В двумерном пространстве симплексом является треугольник, в трехмерном – тетраэдр. Прямые методы поиска, получившие свое название от этого геометрического объекта, основаны на исследовании значений целевой функции в вершинах симплекса и построении последовательности симплексов с центрами, систематически смещающимися в направлении точки минимума.

Из группы симплексных методов наиболее известен метод Нелдера–Мида. Он считается одним из самых эффективных методов минимизации функций при числе переменных $n \le 6$.

В методе Нелдера–Мида симплекс деформируется и перемещается в пространстве с помощью трех операций: *отражения*, *растяжения* и *сжатия*. Смысл этих операций поясним при рассмотрении шагов алгоритма.

1. Находим значения целевой функции в вершинах симплекса:

$$f_1 = f(\mathbf{x}_1), f_2 = f(\mathbf{x}_2), ..., f_{n+1} = f(\mathbf{x}_{n+1}).$$

2. Среди вершин симплекса находим точку X_L с наименьшим значением функции f_L, точку X_G со значением f_G, следующим за наименьшим, и точку X_H с наибольшим значением функции. f_H. Точку X_L назовем наилучшей, точку X_G – хорошей, а точку X_H – наихудшей (см. рис. 4.5 *а*).

3. Определяем центр тяжести $\mathbf{x}_M$ всех вершин симплекса, за исключением наихудшей и вычисляем значение функции в центре тяжести:

$$\mathbf{x}_M = \frac{1}{n}\sum_{i \ne H} \mathbf{x}_i, \; f_M = f(\mathbf{x}_M). \qquad (4.8)$$

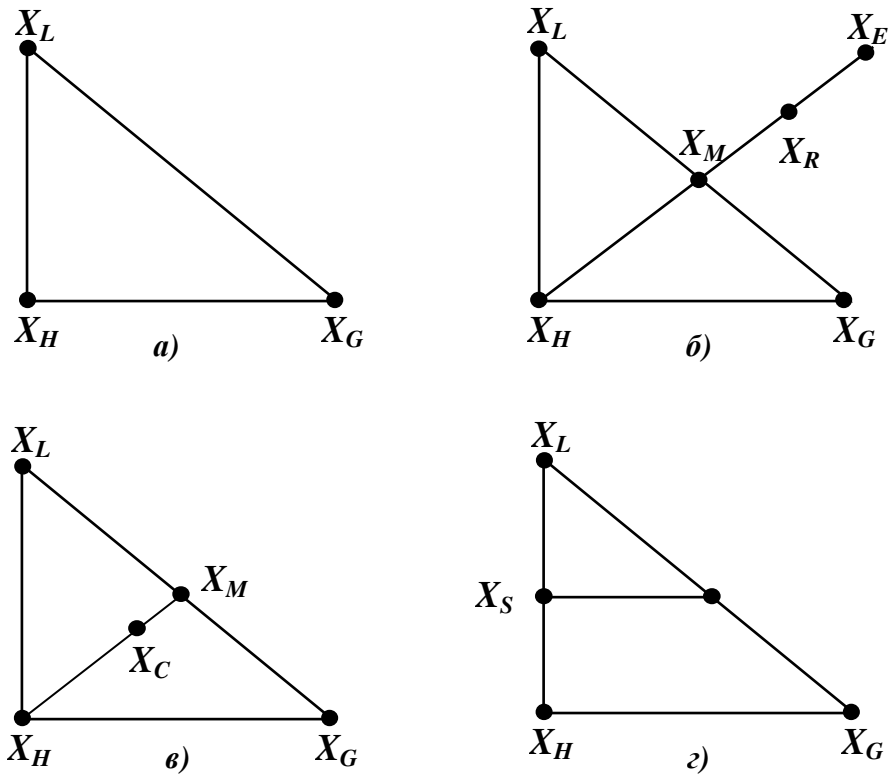


Рис. 4.5. Преобразования двумерного симплекса

4. Проводим отражение точки X_H относительно точки X_M и получаем точку X_R со значением функции $f_R = f(\mathbf{x}_R)$. Формула отражения при этом имеет вид

$$\mathbf{x}_R = (1+\alpha)\mathbf{x}_M - \alpha\mathbf{x}_H . \qquad (4.9)$$

Замечание. Обосновать операцию отражения можно, рассматривая двумерный вариант метода. В этом случае операция отражения иллюстрируется рис. 4.5 *б*. Целевая функция убывает при движении вдоль стороны треугольника от вершины X_H к вершине X_L так же, как и при движении от X_H к X_G. Следовательно, велика вероятность того, что функция принимает наименьшее значение в точках, которые лежат вдали от вершины X_H за противоположной стороной симплекса. Коэффициент $\alpha > 0$ определяет изменение длины вектора $\mathbf{x}_M - \mathbf{x}_H$ при отражении:

$$\mathbf{x}_R - \mathbf{x}_M = \alpha(\mathbf{x}_M - \mathbf{x}_H),$$

отсюда следует формула (4.9).

5. Сравниваем значения f_R и f_G.

Если $f_R < f_G$, то проводим растяжение в выбранном направлении и получаем точку X_E. Ее радиус-вектор выражается формулой

$$\mathbf{x}_E = (1-\gamma)\mathbf{x}_M + \gamma\mathbf{x}_R. \qquad (4.10)$$

Вычисляем значение $f_E = f(\mathbf{x}_E)$.

Замечание. Операция растяжения проводится на том основании, что при выполнении условия $f_R < f_G$ отражение дает правильное направление в сторону минимума. Возможно, минимум находится несколько дальше, чем точка X_R. Поэтому целесообразно продвинуться в выбранном направлении за точку X_R (рис. 4.5 *б*). Коэффициент $\gamma > 0$ определяет растяжение вектора $\mathbf{x}_R - \mathbf{x}_M$, полученного при отражении:

$$\mathbf{x}_E - \mathbf{x}_M = \gamma(\mathbf{x}_R - \mathbf{x}_M).$$

6. Сравниваем значения f_E и f_L.

Если $f_E < f_L$, то заменяем точку X_H точкой X_E и значение функции f_H – значением f_E. После чего проверяем условие сходимости к минимуму и заканчиваем итерационный процесс или возвращаемся на шаг 2.

Если $f_E > f_L$, то точку X_E не принимаем во внимание, а заменяем точку X_H точкой X_R и значение функции f_H – значением f_R. После этого проверяем итерационный процесс на сходимость и либо заканчиваем его, либо возвращаемся на шаг 2.

7. Если найденная на шаге 4 точка X_R такова, что $f_R > f_G$, то сравниваем значения f_R и f_H.

Если $f_R > f_H$, то переходим на шаг 8 – шаг сжатия.

Если $f_R < f_H$, то заменяем точку X_H точкой X_R и значение функции f_H значением f_R, после чего переходим к сжатию симплекса.

8. Проводим операцию сжатия, вычисляя радиус-вектор точки X_C по формуле

$$\mathbf{x}_C = (1-\beta)\mathbf{x}_M + \beta\mathbf{x}_H \qquad (4.11)$$

и значение целевой функции в точке сжатия $f_C = f(\mathbf{x}_C)$.

Замечание. Получив при отражении точку X_R и установив, что $f_R > f_H$, мы приходим к выводу о том, что переместились слишком далеко от точки X_H и пытаемся исправить ситуацию, расположив новую вершину симплекса в точке X_C, лежащей ближе к наилучшей и хорошей точкам, чем X_R (рис. 4.5 *в*).

9. Сравниваем значения f_C и f_H.

Если $f_C < f_H$, то заменяем точку X_H точкой X_C и значение функции f_H значением f_C. Проверяем сходимость и, если она не достигнута, то возвращаемся на шаг 2.

Если $f_C > f_H$, то попытки найти точку со значением целевой функции меньшим, чем f_H, закончились неудачей, поэтому переходим к *стягиванию* симплекса.

10. Проводим стягивание симплекса к наилучшей вершине путем перемещения всех вершин X_k в новые точки с радиус-векторами $(\mathbf{x}_k + \mathbf{x}_L)/2$, т.е. в точки, лежащие на серединах ребер симплекса (рис.4.5 *г*). Затем вычисляем значения функции во всех новых вершинах симплекса и проверяем сходимость. Если сходимость не достигнута, то возвращаемся на шаг 2.

Проверка сходимости в симплексном методе основана на вычислении стандартного отклонения целевой функции в вершинах симплекса:

$$\sigma = \sqrt{\frac{1}{n}\sum_{k=1}^{n+1}\left(f_k - \bar{f}\right)^2},$$

где $\bar{f} = \dfrac{1}{n+1}\sum_{k=1}^{n+1} f_k$ – среднее значение функции в вершинах. Если $\sigma < \varepsilon$, то все значения функции очень близки друг к другу, и поэтому они, наверное, лежат вблизи минимума, приближенно находящегося в наилучшей точке X_L.

Коэффициенты α, β и γ на основании многочисленных экспериментов с различными комбинациями значений Нелдер и Мид рекомендуют выбрать следующим образом: $\alpha = 1,\ \beta = 0{,}5,\ \gamma = 2$. Начальный симплекс может быть произвольным.

Для наглядности шаги описанного алгоритма представлены блок-схемой на рис. 4.6.

4.6. Оптимизация с ограничениями. Метод штрафных функций

Задача оптимизации с ограничениями состоит в поиске минимума целевой функции $f(x_1, x_2, ..., x_n)$, зависящей от n аргументов, при условии выполнения m соотношений связи между аргументами. Связи могут быть заданы в виде равенств:

$$g_i(x_1, x_2, ..., x_n) = 0,\ \ i = 1, 2, ..., I, \qquad (4.12)$$

и неравенств:

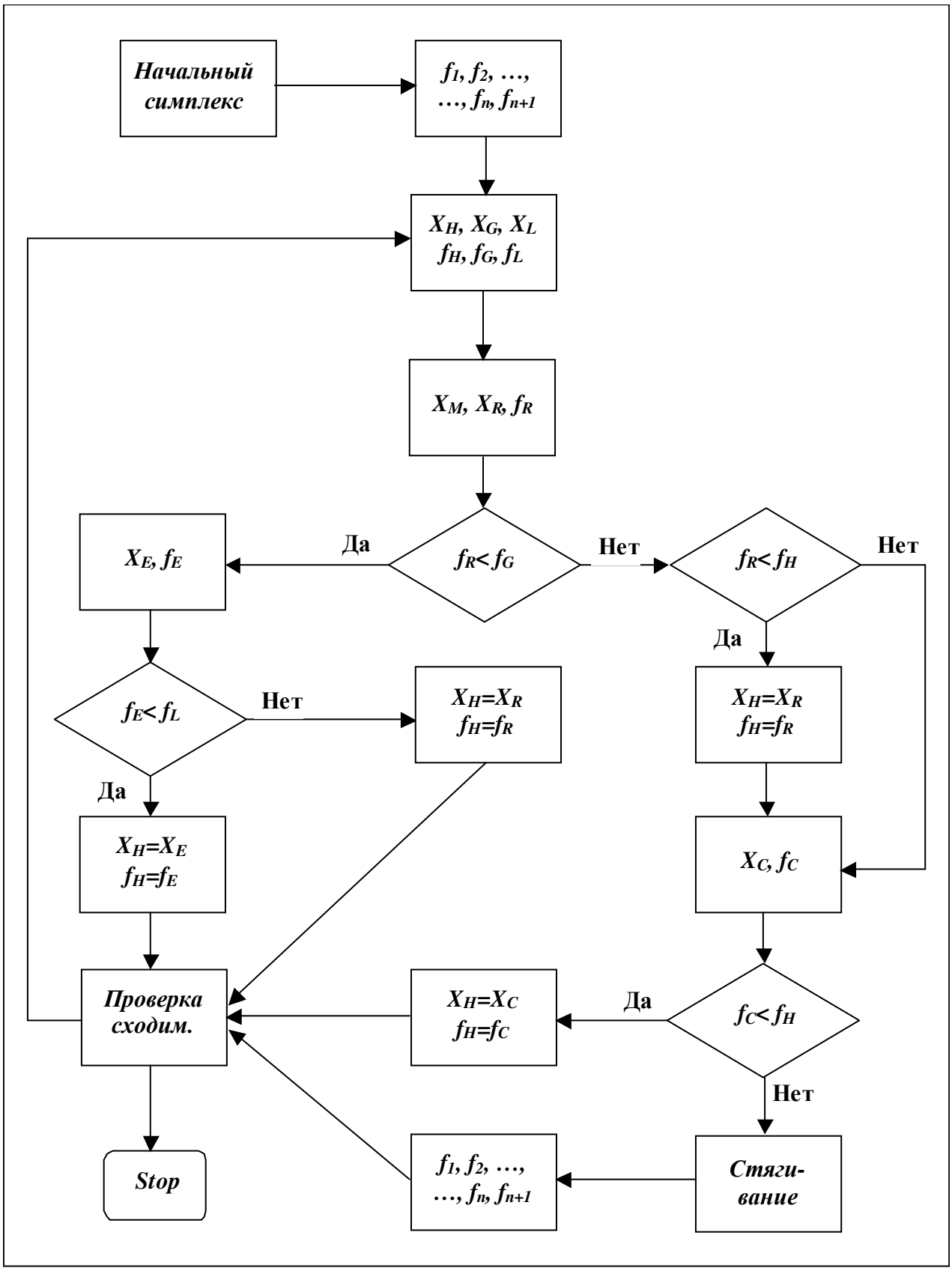


Рис. 4.6. Блок-схема алгоритма симплексного метода

$$q_j(x_1, x_2, ..., x_n) \geq 0, \quad j = 1,2,...,J, \; I + J = m. \tag{4.13}$$

Методы решения задач условной оптимизации составляют содержание специальных курсов линейного и нелинейного программирования. Здесь мы рассмотрим лишь метод *штрафных функций*, позволяющий использовать алгоритмы безусловной оптимизации для решения задач оптимизации с ограничениями. При этом вводится в рассмотрение новая целевая функция

$$F(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{I} \varphi_i[g_i(\mathbf{x})] + \sum_{j=1}^{J} \varphi_j[q_i(\mathbf{x})], \tag{4.14}$$

$\varphi_m[.]$ – штрафные функции, учитывающие ограничения, заданные равенствами и неравенствами. Основное требование к штрафным функциям состоит в том, что они должны принимать нулевые значения, если условия (4.12), (4.13) выполняются, и резко возрастать в противном случае.

Например, можно выбрать функцию (4.14) в виде

$$F(\mathbf{x}, \lambda) = f(\mathbf{x}) + \frac{1}{\lambda}\sum_{i=1}^{I}[g_i(\mathbf{x})]^2 + \frac{1}{\lambda}\sum_{j=1}^{J}\left[\max\left(q_i(\mathbf{x}),0\right)\right]^2, \; \lambda > 0. \tag{4.15}$$

При малых значениях параметра λ функция $F(\mathbf{x}, \lambda)$ вне области *S*, ограниченной условиями (4.12), (4.13), резко возрастает. Минимум функции $F(\mathbf{x}, \lambda)$ может находиться либо внутри области *S*, либо вне ее. В первом случае минимумы функций $f(\mathbf{x})$ и $F(\mathbf{x}, \lambda)$ совпадают, т.к. штрафные функции равны нулю внутри *S*. Во втором – минимум целевой функции $f(\mathbf{x})$ лежит на границе области. При этом можно построить последовательность $\lambda_k \to 0$ такую, что последовательность минимумов функции $F(\mathbf{x}, \lambda_k)$ стремится к минимуму функции $f(\mathbf{x})$.

Казалось бы, задачу можно решить в один этап при малом λ. Но в этом случае линии уровня функции $F(\mathbf{x}, \lambda)$ представляют собой «овраги» с крутыми склонами. Выше отмечалось, что минимизация таких функций затруднительна. Поэтому легче решать последовательность задач оптимизации, принимая минимум, найденный на очередном этапе, за начальное приближение следующего этапа.

Таким образом, задача минимизации целевой функции $f(\mathbf{x})$ с ограничениями (4.12), (4.13) свелась к последовательности задач безусловной оптимизации для вспомогательной функции (4.15).

Литература

1. Бахвалов, Н.С. Численные методы [Текст] / Н.С.Бахвалов, Н.П.Жидков, Г.М.Кобельков. – 3-е изд., доп. и перераб. – М.: БИНОМ. Лаборатория знаний, 2004.
2. Турчак, Л.И. Основы численных методов [Текст] / Л.И.Турчак, П.В.Плотников. – М.: ФИЗМАТЛИТ, 2002.
3. Каханер, Д. Численные методы и программное обеспечение [Текст] / Д.Каханер, К.Моулер, С.Нэш. – М.: Мир, 1998.
4. Мэтьюз, Д. Численные методы. Использование MATLAB [Текст] / Д.Мэтьюз, К.Финк. – М.: Издательский дом «Вильямс», 2001.
5. Банди, Б. Методы оптиимзации. Вводный курс [Текст] / Б.Банди. – М.: Радио и связь, 1988.
6. Шуп, Т. Решение инженерных задач на ЭВМ: Практическое руководство [Текст] / Т.Шуп. – М.: Мир, 1982.

Приложение 4
Программы многомерной оптимизации

Приведем ряд программ для решения задач многомерной оптимизации в среде MathCAD.

Программа *NMin_G* реализует алгоритм метода градиентного спуска. Заголовок программного модуля имеет вид

$$NMin\_G(F,x0,h,\varepsilon),$$

где F – имя минимизируемой функции, x0 и h – начальная точка и начальный шаг поиска, ε – точность. Программа возвращает многомерный вектор-столбец, состоящий из координат точки приближенного минимума, значения функции в этой точке и количества проведенных итераций. Текст модуля приведен на рис. П.4.1. Программа *NMin_G* использует вспомогательный модуль

$$Grad(F,x,\Delta),$$

вычисляющий единичный вектор в направлении градиента функции F в текущей точке x; Δ – приращение аргумента, используемое для вычисления производных по формуле центральных разностей. Текст программного модуля представлен на рис. П.4.2.

Программа

$$NMin\_SPI(F,x0,\varepsilon),$$

проводит поиск минимума функции методом покоординатного спуска. При этом минимизация в направлении спуска осуществляется методом последовательной параболической интерполяции. Программный модуль *NMin_SPI(F,x0,ε)* имеет те же аргументы, что и представленный выше

модуль *NMin_G*, за исключением аргумента *h*, который в данном случае не используется. Текст модуля *NMin_SPI* приведен на рис. П.4.3.

Программный модуль

Simplex(F,x0,ε),

реализует вычисления симплексным методом Нелдера–Мида. Он имеет те же аргументы, что и модули, представленные выше; возвращает вектор-столбец координат точки приближенного минимума. Текст модуля – на рис. П.4.4.

```
Min_G(F, x0, h, ε) := | Программа_многомерной_оптимизации←%
                      | методом_градиентного_спуска←%
                      | N←length(x0)
                      | I←0
                      | while 1
                      |     g←Grad(F, x0, 0.0001)
                      |     for n∈0..N - 1
                      |       x_n←x0_n - h·g_n
                      |     I←I + 1
                      |     Out_0←F(x)
                      |     Out_1←I
                      |     return stack(x, Out) if h<ε
                      |     if F(x)≥g_N
                      |         | h←h/2
                      |         | x←x0
                      |     x0←x
```

Рис. П.4.1. Программа многомерной оптимизации методом градиентного спуска

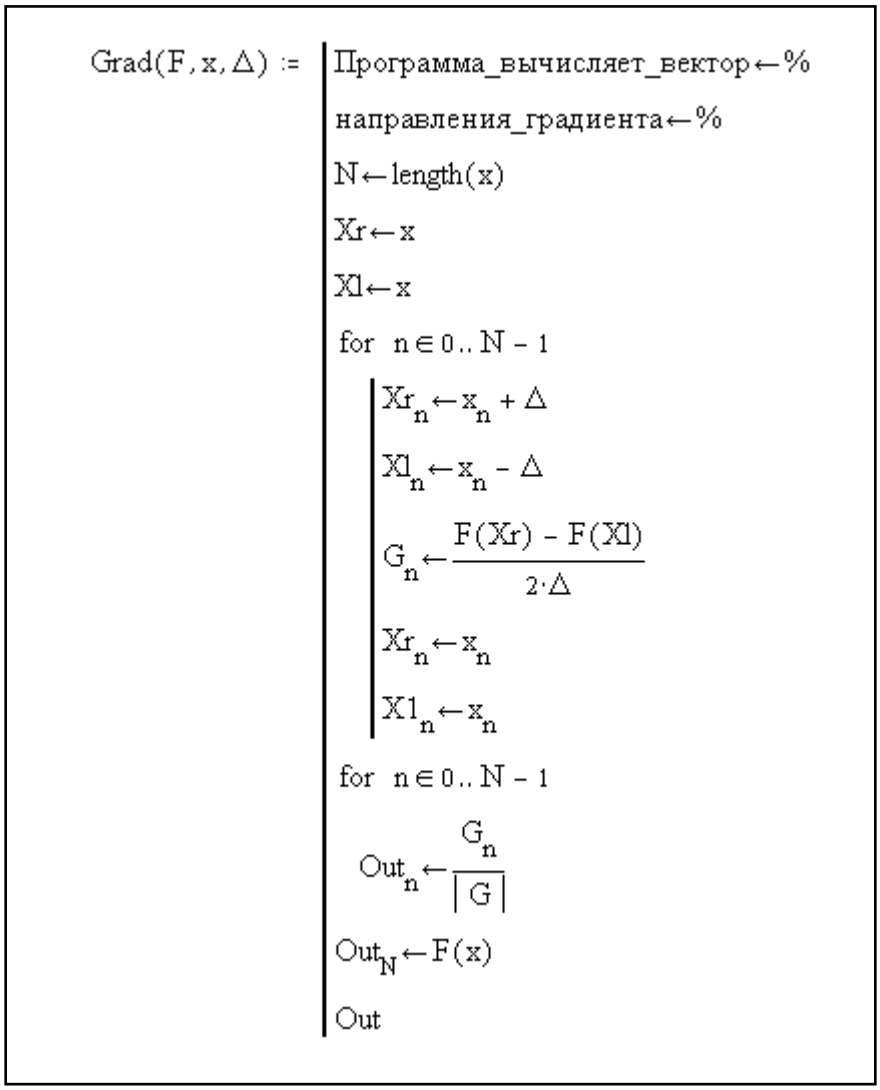


Рис. П.4.2. Программа вычисления единичного вектора в направлении градиента и значения функции в текущей точке

```
NMin_SPI(F, x0, ε) := | Многомерная_оптимизация_методом ← %
                      | последовательной_параболической_интерполяции ← %
                      | по_каждой_переменной ← %
                      | N ← length(x0) - 1
                      | X2 ← x0
                      | X1 ← x0
                      | X0 ← x0
                      | while 1
                      |     Z ← X2
                      |     for n ∈ 0..N
                      |         X0_n ← X2_n + 0.01
                      |         X1_n ← X2_n
                      |         X2_n ← X2_n - 0.01
                      |         f0 ← F(X0)
                      |         f1 ← F(X1)
                      |         while |X2_n - X1_n| ≥ ε
                      |             f2 ← F(X2)
                      |             A0 ← f0·[(X2_n)^2 - (X1_n)^2]
                      |             A1 ← f1·[(X2_n)^2 - (X0_n)^2]
                      |             A2 ← f2·[(X1_n)^2 - (X0_n)^2]
                      |             A ← (A0 - A1 + A2) / (2·[f2·(X1_n - X0_n) - f1·(X2_n - X0_n) + f0·(X2_n - X1_n)])
                      |             f0 ← f1
                      |             f1 ← f2
                      |             X0 ← X1
                      |             X1 ← X2
                      |             X2_n ← A
                      |     return X2 if |X2 - Z| ≤ ε·√(N + 1)
```

Рис. П.4.3. Программа многомерной оптимизации методом покоординатного спуска с последовательной параболической интерполяцией

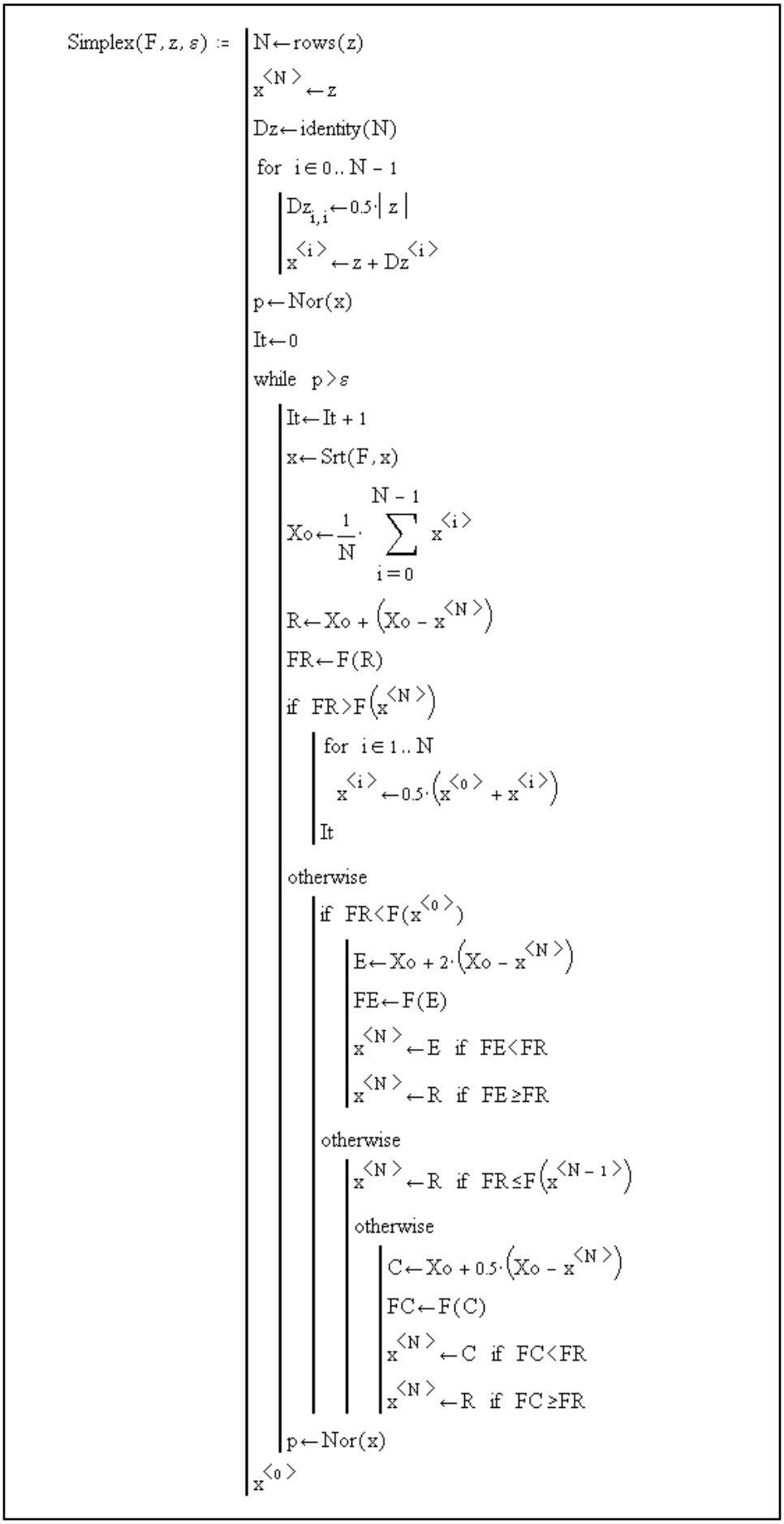


Рис. П.4.4. Программа многомерной оптимизации симплексным методом

Программа *Simplex* использует вспомогательные модули

Str(F,x) и ***Nor(x)***,

первый из которых проводит сортировку координат вершин симплекса, а второй – вычисляет векторную норму.

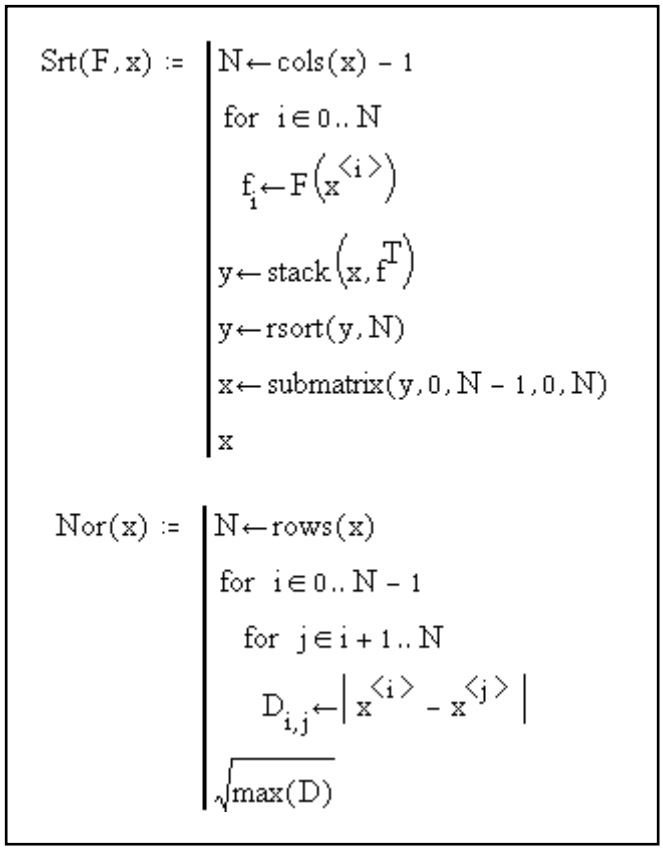


Рис. П.4.5. Вспомогательные программы для модуля Simplex

Пример обращения к программным модулям при минимизации функции Розенброка приведен на рис. П.4.6.

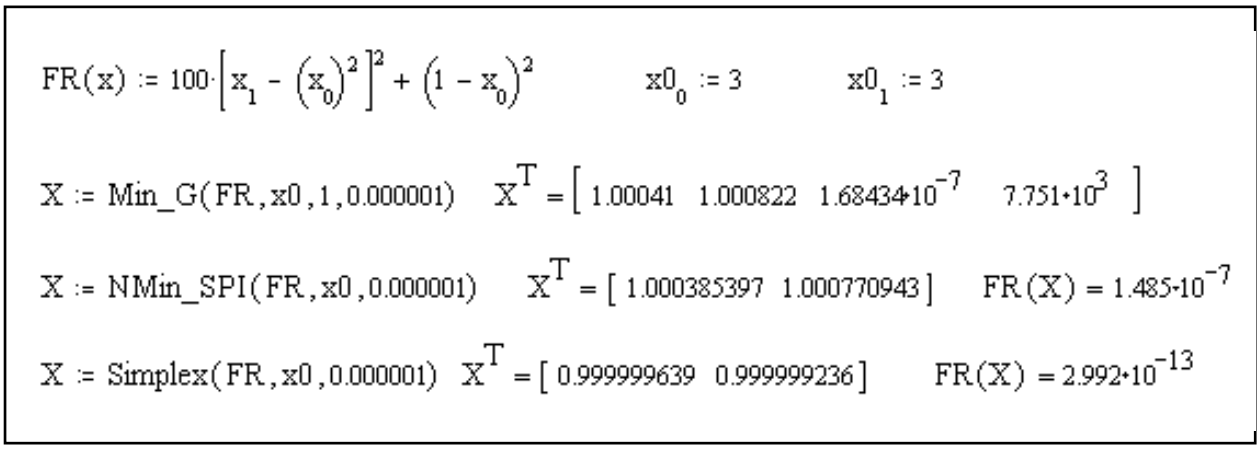


Рис. П.4.6. Обращения к программам многомерной оптимизации

СОДЕРЖАНИЕ

	Предисловие ……...	3
1.	**ЧИСЛЕННОЕ РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ**	4
1.1.	Введение ……..…….……...	4
1.2.	Метод простой итерации ……..…….	5
1.3.	Методы половинного деления и ложного положения …….	8
1.4.	Метод Ньютона и метод секущих ……..……...	11
1.5.	Метод Стеффенсена и ускорение сходимости Эйткена ……..……	15
1.6.	Метод обратной квадратичной интерполяции ……..….	19
1.7.	Численные методы поиска комплексных корней полиномов …..……...	20
	Литература ……..……..	22
	Приложение 1. Программы решения нелинейных уравнений ……...….	23
2.	**РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ**	29
2.1.	Введение ……..……..	29
2.2.	Метод простой итерации ……..…….	30
2.3.	Метод Зейделя …….	31
2.4.	Метод возмущения параметров …….……	32
2.5.	Итерации Пикара …….	33
2.6.	Метод Ньютона ……..……..	34
2.7.	Метод Бройдена ……..	39
	Литература ……...	43
	Приложение 2.1. Решение систем уравнений, зависящих от параметра	43
	Приложение 2.2. Программы решения систем нелинейных уравнений	45
3.	**МЕТОДЫ ОДНОМЕРНОЙ ОПТИМИЗАЦИИ**	48
3.1.	Введение ……..……..	48
3.2	Метод Ньютона ……..……..	49
3.3	Метод последовательной параболической интерполяции ……..	51
3.4	Метод золотого сечения …….……	52
3.5	Оптимизация методом установления …….……..	56
	Литература ……..	61
	Приложение 3. Программы одномерной оптимизации ……..…….	61
4.	**МНОГОМЕРНАЯ ОПТИМИЗАЦИЯ**	66
4.1.	Введение ……..…….	66
4.2.	Метод покоординатного спуска …….	67
4.3.	Градиентные методы. Метод наискорейшего спуска ……..	70
4.4.	Метод Ньютона. Квазиньютоновские методы …….	72
4.5.	Симплексный метод Нелдера–Мида …….	75
4.6.	Оптимизация с ограничениями. Метод штрафных функций …….	78
	Литература ……..……..	81
	Приложение 4. Программы многомерной оптимизации …….	83

Учебное издание

Зайцев Валерий Васильевич

Трещев Владимир Михайлович

ЧИСЛЕННЫЕ МЕТОДЫ ДЛЯ ФИЗИКОВ.

НЕЛИНЕЙНЫЕ УРАВНЕНИЯ И ОПТИМИЗАЦИЯ

Учебное пособие

Редактор Ю.В.Яценко
Корректор Ю.В.Яценко
Компьютерная верстка, макет И.К.Зайцевой

Подписано в печать 12.04.06
Формат 60x84/16. Бумага офсетная. Усл.печ.л. 5,1; уч.-изд.л. 5,5.
Тираж 250 экз. Заказ № 476
Издательство «Самарский университет», 443011, г.Самара, ул. Акад. Павлова, 1.
Отпечатано ООО «Универс-групп»