Численные методы решения нелинейных уравнений

В лекции будут рассматриваться уравнения вида $f(x^*) = 0$, где $x, f \in \mathbb{R}$, а также $x^* = \varphi(x^*)$. Подразумевается, что корень у функции один и существует. Кроме того подразумевается, что функция f нелинейная.

Представление корня будет искаться итеративно. Так, будет некоторое начальное приближение x_0, а методы будут вычислять последовательность, которая сходится к x^*:

$\lim_{n\to\infty} x_n = x^*$

Найти точное значение корня в общем случае не представляется возможным, так как

- Точный корень может не иметь арифметического выражения
- В машинной арифметике присутствуют погрешности

Введем понятия прямой и обратной ошибки.

Прямая ошибка: $\Delta f = |f(x_n)|$

Обратная ошибка: $\Delta x = |x_n - x^*|, \epsilon_x = \Delta x / x$ (относительная обратная ошибка)

Метод половинного деления

Метод деления пополам позволяет исключать в точности половину интервала на каждой итерации.

При использовании метода считается, что функция непрерывна и имеет на концах интервала разный знак, что гарантирует наличие хотя бы одного корня на интервале.

> Метод каждый раз делит отрезок на два равных (добавляется новая серединная точка) и для следующей итерации выбирает тот, в котором значения функции на концах отличаются по знаку. Так продолжается до тех пор, пока в серединной точке значение функции не станет по модулю меньше, чем установленная погрешность.

Рассмотрим процесс построения последовательности. За начальную точку берется середина некоторого отрезка $[L_0, R_0]$, который удовлетворяет условию: $\text{sign}(f(L_0)) \neq \text{sign}(f(R_0))$.

$$x_0 = \frac{1}{2}(L_0 + R_0)$$

Далее возможны три случая на каждой итерации:

- Если $\text{sign}(f(L_n)) \neq \text{sign}(f(x_n))$

$$\begin{cases} x_{n+1} = \frac{1}{2}(L_n + x_n) \\ R_{n+1} = x_n \\ L_{n+1} = L_n \end{cases}$$

- Если $\text{sign}(f(x_n)) \neq \text{sign}(f(R_n))$

$$\begin{cases} x_{n+1} = \frac{1}{2}(x_n + R_n) \\ L_{n+1} = x_n \\ R_{n+1} = R_n \end{cases}$$

- $x^* = x_n$

Метод половинного деления может работать достаточно долго, так как на каждом шаге отрезок, на котором может быть корень, уменьшается всего в два раза. Иначе говоря, обратная ошибка на шаге n:

$$|x_n - x^*| < \frac{R_0 - L_0}{2^{n+1}}$$

Такая сходимость является *линейной*, то есть *сходимостью порядка 1*.

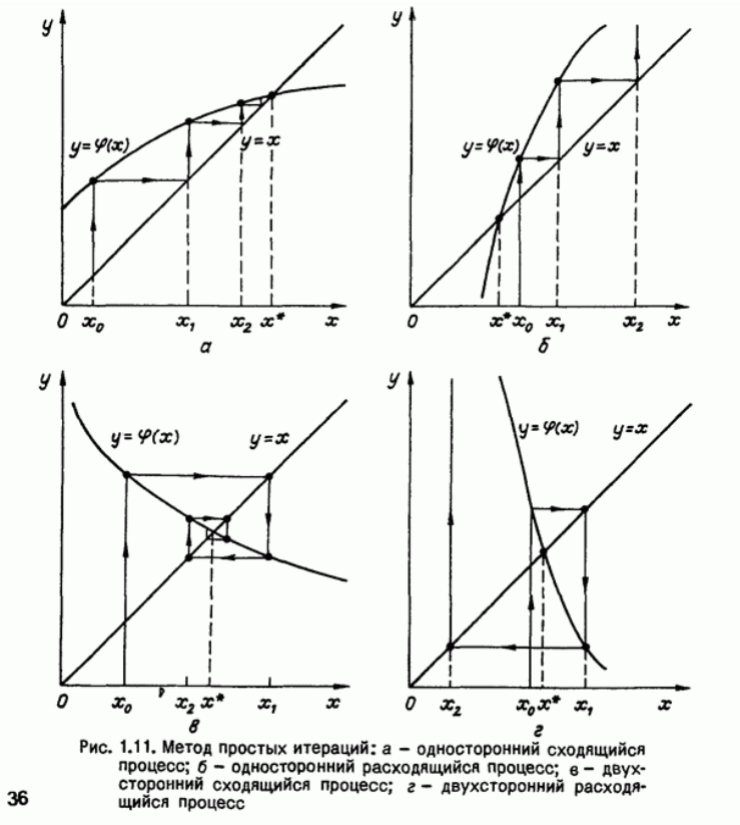
При использовании мантиссы преставления корня размером 52 бит, в худшем случае потребуется 52 итерации, чтобы получить корень. Это медленно, поэтому метод применяется крайне редко.

Метод простой итерации

Вспомним представление нелинейного уравнения как $x^* = \varphi(x^*)$. Построение последовательности в методе простой итерации эту формулу напоминает:

$x_{n+1} = \varphi(x_n)$

> Берется некоторое начальное приближение. На каждом следующем шаге применяем функцию φ к предыдущей аппроксимации. Визуально это выглядит как «ступеньки» между функцией $y = \varphi(x)$ и $y = x$. Не всегда сходится.



> Источник изображения: Мудров А. Е. Численные методы для ПЭВМ на языках Бейсик, Фортран и Паскаль. - Томск: МП "РАСКО", 1991. - 272 с.

Если функция $\varphi(x)$ – сжимающее отображение, то есть $\varphi'(x) \leq k < 1$, то $\lim_{n\to\infty} x_n = x^*$.

Пример: уравнение Кеплера для эллипса

$E = M + e \sin E \; [= \varphi(E)], e < 1$

$\varphi' = e \cos E \leq e < 1$

E – эксцентрическая аномалия

M – средняя аномалия

Тогда метод будет рассчитывать последовательность:

$$\begin{aligned} E_0 &= M \\ E_1 &= M + e \sin E_0 \\ E_2 &= M + e \sin E_1 \\ \dots \end{aligned}$$

Иллюстрация «а» на рисунке 1.11 выше иллюстрирует работу метода на подобном уравнении.

Поговорим о сходимости метода.

Учитывая, что $\varphi'(x) \leq k < 1$, получаем, что

$$|x_{n+1} - x^*| = |\varphi(x_n) - \varphi(x^*)| \leqslant k|x_n - x^*| \leqslant k^n |x_0 - x^*|$$

Таким образом, последовательность убывает, то есть действительно $\lim_{n\to\infty} x_n = x^*$.

Порядок сходимости q:

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|^q} = \mu$$

- Если φ' определена и непрерывна:

$$\begin{aligned} \lim \frac{|x_{n+1}-x^*|}{|x_n-x^*|} = \lim \frac{|\varphi(x_n)-\varphi(x^*)|}{|x_n-x^*|} &= \lim |\varphi'(\xi_n)|,\ \xi_n \in [x_n,\ x^*] \\ &= |\varphi'(x^*)| \end{aligned}$$

Таким образом, порядок сходимости – 1.

- Если $\varphi'(x^*) = 0$, φ'' определена и непрерывна:

Используем разложение в ряд Тейлора:

$$\varphi(x_n) = \varphi(x^*) + \varphi'(x^*)(x_n - x^*) + \frac{1}{2}\varphi''(\xi_n)(x_n - x^*)^2,\ \xi_n \in [x_n, x^*]$$

Тогда, полагая $q = 2$:

$$\begin{aligned} \lim \frac{|x_{n+1}-x^*|}{|x_n-x^*|^2} = \lim \frac{|\varphi(x_n)-\varphi(x^*)|}{|x_n-x^*|^2} &= \lim \frac{|\varphi''(\xi_n)|}{2},\ \xi_n \in [x_n,\ x^*] \\ &= \frac{|\varphi''(x^*)|}{2} \end{aligned}$$

Предел существует, значит при соблюдении вышеупомянутых условий порядок сходимости метода – 2.

Метод Ньютона

Один из наиболее часто используемых методов, обладающий порядком сходимости 2.

Определим функцию $\varphi(x)$:

$$\varphi(x) = x - \frac{f(x)}{f'(x)}$$

Несложно заметить, что ее производная:

$$\varphi'(x) = \frac{f(x)f''(x)}{f'(x)^2}$$

Тогда, если $f'(x) \neq 0$: $\varphi(x^*) = x^*, \varphi(x^*) = 0$

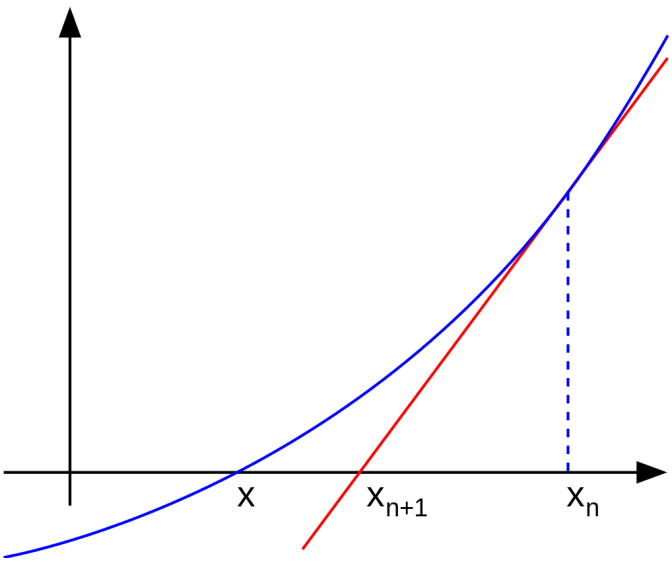
Условия квадратичной сходимости $q = 2$

- $f'(x) \neq 0$, при нарушении метод не сможет продолжить работу
- $f''(x)$ непрерывна
- $f'''(x)$ ограничена
- Начальное приближение достаточно близко к x^*

Эти ограничения, однако, достаточно серьезные, но квадратичная скорость сходимости оправдывает применение метода.

При нарушении последний трех нарушений метод может либо не сойтись вообще, либо сойтись с меньшим порядком сходимости.

Графически один шаг метода выглядит так:



> Метод проводит касательную к функции в точке x_n, а за x_{n+1} берется точка, в которой эта касательная пересекается с 0

Примеры нарушений условий

1. $f(x) = \sqrt{x}$: $x_{n+1} = x_n - \frac{\sqrt{x_n}}{-1/(2\sqrt{x_n})} = -x_n$

Прежде всего, нарушается условие $f'(x) \neq 0$, так как $f'(x^*) = 0, x^* = 0$.

Если начинать итерации с 0, то продолжить их мы не сможем. С другой стороны, это и не потребуется, так как 0 есть корень.

Если начать итерации с положительного числа, метод на первом же шаге попадет в отрицательную область (где, в общем-то, значения функции уже не определены). Но метод этого «не заметит» и продолжит вычислять последовательность, которая будет состоять из двух чисел, одинаковых по модулю и разных по знаку. Метод расходится.

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

Если начинать итерации с 0, то продолжить их мы не сможем. С другой стороны, это и не потребуется, так как 0 есть корень.

Если начинать с положительного числа, то, как видно из выражения выше, на каждом шаге метод будет вычислять значение, вдвое меньшее значению на предыдущем шаге. Таким образом корень будет достигнут, но порядок сходимости – 1.

Машинные вычисления

Машинные вычисления отличаются от вычислений на бумаге тем, что числа представляются в виде дробей со знаменателем, равным степени двойки. Таким образом, представление чисел уже несет в себе некоторую погрешность. Кроме того, процессор имеет ограниченное количество арифметических операций и функций.

Примитивы:

- fadd, fsub, fmul, fdiv (+, -, *, /)
- fsqrt ($\sqrt{x}$)
- fsin, fcos
- fyl2x, fy2lxp1 ($y \cdot \log_2(x), y \cdot \log_2(x+1)$)
- rsqrtss (SSE) ($1/\sqrt{x}$)

На самом деле, в некоторых процессорах нет даже всех этих примитивов. В этом случае, например, $sin(x)$ можно посчитать через ряд Тейлора:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

При больших значениях x требуется вычислить достаточно большой ряд, что не оптимально, поэтому x переносится в окрестность 0, учитывая периодичность функции $\sin(x)$.

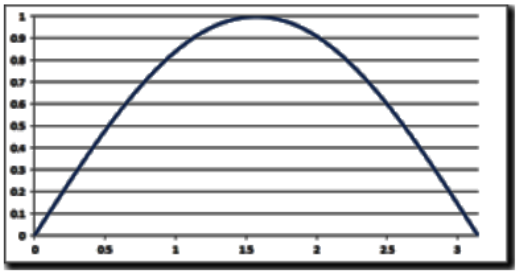
В процессорах Intel была реализована такая операция уменьшения аргумента.

$\sin(\pi + \varepsilon) = -\sin(\varepsilon)$

Но это было сделано без учета достаточного количества значащих цифр в числе π, и в 2014 году было обнаружено, что все реализации $\sin$ в процессорах Intel имеют ошибку, которая проявляется при аргументах, близких к π.

Intel Underestimates Error Bounds by 1.3 quintillion

Posted on October 9, 2014



Intel's manuals for their x86/x64 processor clearly state that the *fsin* instruction (calculating the trigonometric *sine)* has a maximum error, in round-to-nearest mode, of one unit in the last place. This is not true. It's not even close.

The worst-case error for the *fsin* instruction for small inputs is actually about 1.37 quintillion units in the last place, leaving fewer than four bits correct. For huge inputs it can be much worse, but I'm going to ignore that.

I was shocked when I discovered this. Both the *fsin* instruction and Intel's documentation are hugely inaccurate, and the inaccurate documentation has led to poor decisions being made.

The great news is that when I shared an early version of this blog post with Intel they reacted quickly and the documentation is going to get fixed!

I discovered this while playing around with my favorite mathematical identity. If you add a double-precision approximation for *pi* to the *sin()* of that same value then the sum of the two values (added by hand) gives you a quad-precision estimate (about 33 digits) for the value of *pi*. This works because the *sine* of a number very close to *pi* is almost equal to the error in the estimate of *pi*. This is just calculus 101, and a variant of Newton's method, but I still find it charming.

В связи с этим, синус теперь считается через ряд Тейлора (речь идёт о стандартной библиотеке языка Си – libc).

Вычисление квадратного корня

Забудем, что такой примитив есть в процессоре. Тогда можно использовать ряд Тейлора (учитывая, что в окрестности 0 ряд не определен):

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{x^3}{16} - \frac{5x^4}{128} + \frac{7x^5}{256} + \cdots$$

Обеспечивается сходимости порядка 1 в $|x| \leq 1$

Ограниченность интервала сходимости не является проблемой – в функциях часто аргумент приводится к интервалу, где сходимость быстрее.

Теперь запишем уравнение в виде $x = \sqrt{a}$. Тогда $x = \varphi(x) = \frac{a}{x}$. Но метод неподвижной точки не сходится.

Запишем уравнение в другом виде:

$f(x) = a - x^2 = 0$

Решим уравнение методом Ньютона:

$$\varphi(x) = x - \frac{f(x)}{x'(x)} = x + \frac{a - x^2}{2x} = 1/2\left(x + \frac{a}{x}\right)$$

Данный метод называется Вавилонским методом. Сходимость порядка 2, но на каждой итерации происходит деление - операция крайне нежелательная в вычислительных алгоритмах по двум причинам:

- Деление занимает намного больше процессорного времени, чем умножение
- Деление может порождать численные артефакты, если знаменатель очень близок к 0

Вычисления обратного квадратного корня

Сформулируем задачу иначе:

$x = 1/\sqrt{a}$

$f(x) = ax^2 - 1 = 0$

Решим уравнение методом Ньютона:

$$\varphi(x) = x - \frac{ax^2 - 1}{2ax} = x - \frac{x}{2}(ax^2 - 1)$$

Метод сходится с порядком 2 и не содержит делений. Если начальная точка задана в диапазоне от 0 до 2-х, то метод сходится примерно за 3 итерации, то есть достигает точности, сравнимой с машинной точностью представления числа.

Остается лишь умножить на a полученное значение:

$$\sqrt{a} = a \cdot \frac{1}{\sqrt{a}} = ax$$