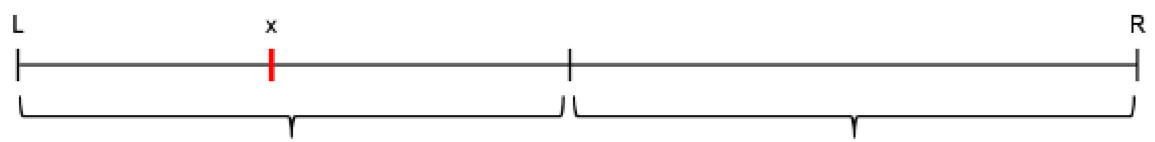
Алгоритми та складність

I семестр

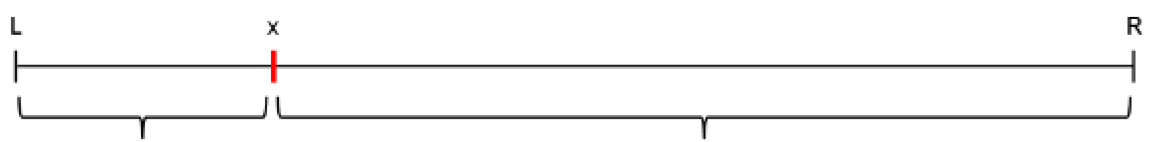
Лекція 7

Інтерполяційний пошук

- Припустимо, ми хочемо знайти певну людину в телефонній книзі чи слово у словнику.
- Зрозуміло, прізвище «Бойко» ми станемо шукати ближче до початку книги, а не її середини (як приміром, для «Мельник») чи кінця (у випадку «Шевченко»).
- Принцип бінарного пошуку в цьому випадку не виглядає найефективнішим:



- Бажано було б отримати поділ, схожий на цей:

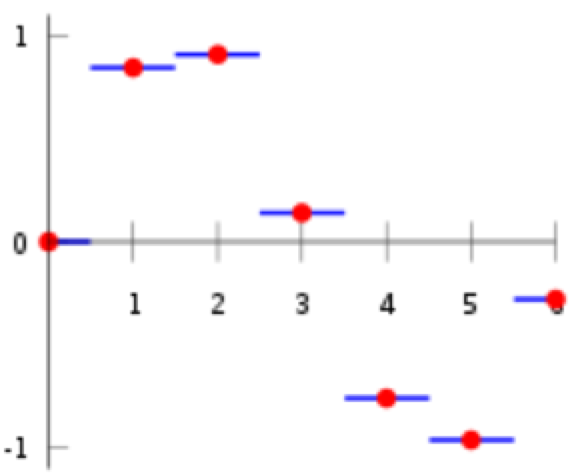


Інтерполяційний пошук

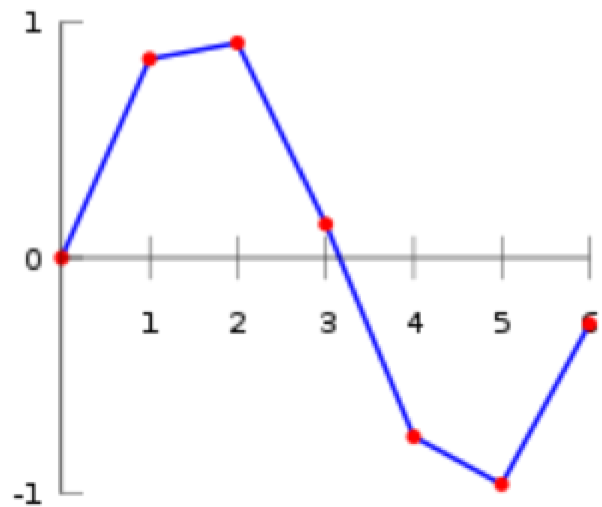
- Інтерполяційний пошук (W.W.Peterson, 1957) – алгоритм для пошуку за заданим ключем у впорядкованому масиві.
- Він враховує значення шуканого ключа при розбитті.
- Відомо, що значення в масиві зростають (строго кажучи, не спадають). Припустимо, вони зростають за певним відомим законом. Тоді точкою поділу масиву буде очікувана позиція значення ключа.
- Інтерполяційний пошук належить до алгоритмів зі змінним зменшенням розміру задачі.

Інтерполяційний пошук

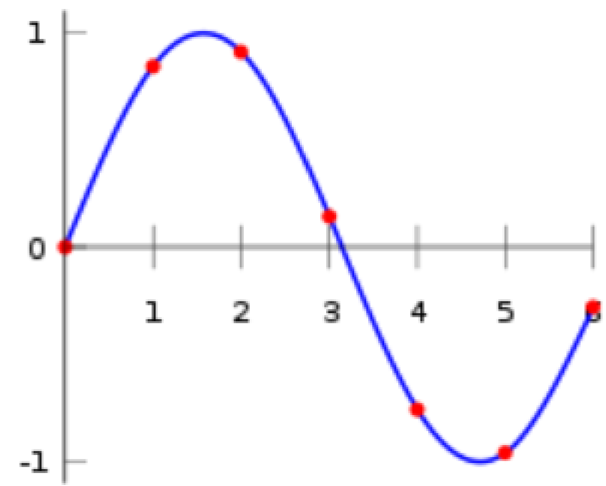
- Інтерполяція – спосіб знаходження невідомих проміжних значень певної функції за заданим дискретним набором її значень.
- Термін ввів англієць John Wallis (1656 р.).



Інтерполяція методом найближчого сусіда

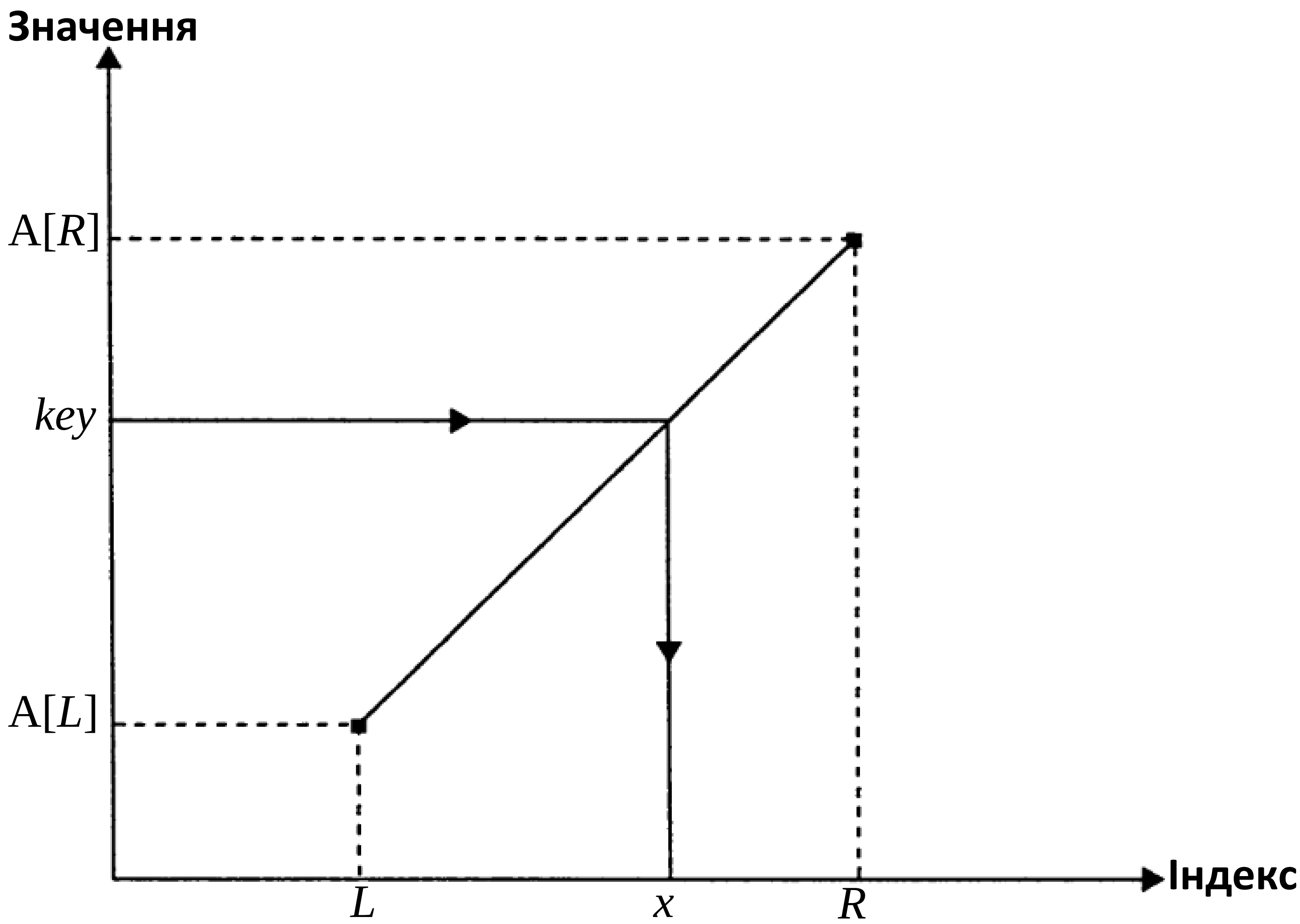


Лінійна інтерполяція



Інтерполяція кубічними сплайнами

Інтерполяційний пошук



Обчислення індексу при інтерполяційному пошуку

Інтерполяційний пошук

- Нехай виконується ітерація пошуку в масиві A між елементами A[*L*] та A[*R*].
- Вважається, що значення в масиві зростають лінійно.
- Значення ключа пошуку *key* порівнюється з елементом, індекс якого обчислюється як абсциса точки на прямій, що проходить через точки (*L*, A[*L*]) та (*R*, A[*R*]) і має ординату *key*.
- Використовуючи рівняння прямої, отримаємо

$$x = L + \left\lfloor \frac{(key - \mathrm{A}[L])(R - L)}{\mathrm{A}[R] - \mathrm{A}[L]} \right\rfloor.$$

Інтерполяційний пошук

- В припущенні, що дані рівномірно розподілені по шкалі інтерполяції, алгоритм має ефективність O(log log N). (Точніше, це менше $\log_2 \log_2 N+1$ порівнянь ключів в середньому.)
- Однак в найгіршому випадку пошук вироджується в лінійний.
- Реалізації алгоритму можуть працювати некоректно при наявності однакових елементів.
- Інтерполяційний пошук матиме сенс, якщо зменшення кількості порівнянь важливіше за ускладнення обчислень при кожному з них (наприклад, при зверненні до великих об’ємів даних у зовнішній пам’яті).
- Алгоритм може використовуватись для пошуку у відсортованому, але не індексованому наборі даних.

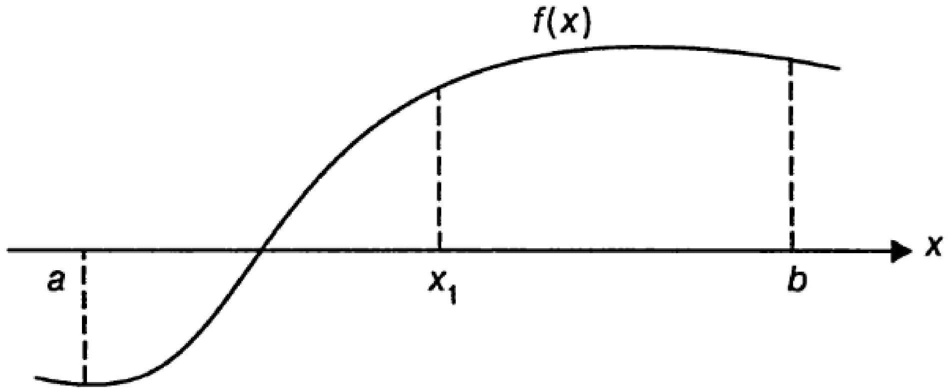
Розв'язання нелінійних рівнянь

- Пару алгоритмів розв'язання нелінійних рівнянь можна співвіднести з алгоритмами бінарного та інтерполяційного пошуків.
- Шукається розв'язок рівняння вигляду
$$f(x) = 0.$$
- Не існує формули для визначення точних розв'язків такого рівняння загального вигляду, тому потрібні алгоритми для наближеного розв'язання.
- Будемо інтерпретувати розв'язок рівняння як точку, в якій графік функції $f(x)$ перетинає вісь абсцис.
- Функція може мати один, декілька чи нескінченну кількість коренів, або їх не мати взагалі, тому бажано спочатку на графіку оцінити розташування коренів та виділити інтервали, які їх містять.

Метод дихотомії

- Інші назви – метод поділу навпіл, метод бісекції.
- В основі алгоритму лежить факт, що графік неперервної функції має перетнути вісь абсцис між двома точками *a* та *b* хоча б один раз, якщо значення функції в цих точках має різні знаки (теорема Больцано-Коші про проміжне значення неперервної функції).
- Починаючи з відрізка [a,b], на кінцях якого $f(x)$ має протилежні знаки, знаходиться середня точка $x_{mid} = (a + b)/2$.
- Якщо $f(x_{mid}) = 0$, корінь знайдено.
- Інакше алгоритм продовжує пошук кореня на тому з відрізків $[a, x_{mid}]$ чи $[x_{mid}, b]$, на кінцях якого функція має різні знаки.

Метод дихотомії



Перша ітерація методу. Точка x_1 – середина [a, b]

- Процес завершується, коли відрізок $[a_n,b_n]$, що містить корінь x*, настільки малий, що абсолютна похибка наближення x* величиною $x_n = (a_n + b_n)/2$ менша за вибране значення $\varepsilon > 0$, тобто при

$$\frac{b_n - a_n}{2} < \varepsilon \quad (\text{або ж} \quad x_n - a_n < \varepsilon).$$

Метод дихотомії

- Легко довести, що

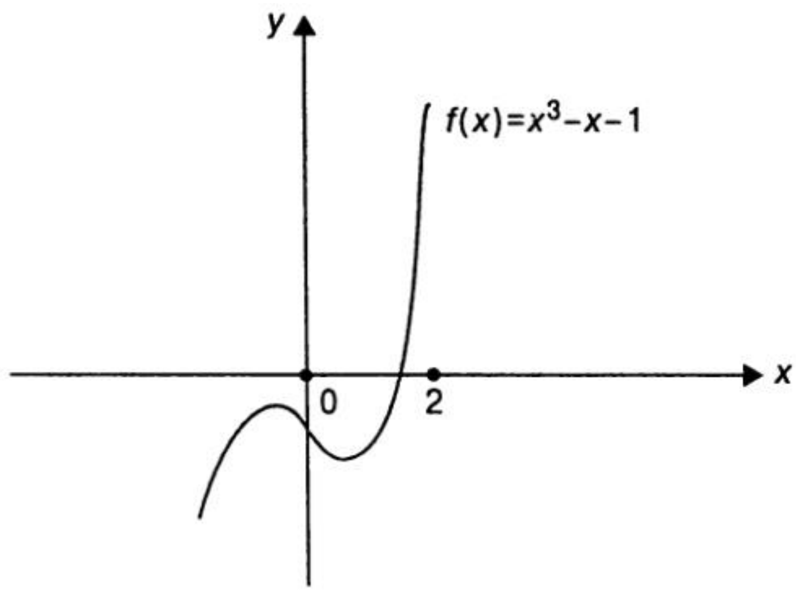
$$|x_n - x^*| \le \frac{b_n - a_n}{2^n} \quad \text{для } n = 1,2, \dots .$$

- Це означає, що послідовність наближень $\{x_n\}$ збігається до кореня x*.
- Вибираючи ε, слід враховувати машинні обмеження при обчисленні значень з плаваючою комою.
- В реалізацію бажано включити обмеження на кількість ітерацій. З попередньої формули можна вивести n, (теоретично) достатнє для досягнення необхідної точності ε: $n > \log_2 \frac{b_1 - a_1}{\varepsilon}$.

Метод дихотомії

<u>Приклад</u>. Розглянемо рівняння $x^3 - x - 1 = 0$.

Воно має єдиний дійсний корінь. Оскільки $f(0)<0$ та $f(2)>0$, будемо шукати корінь в інтервалі (0, 2).



Вибравши похибку $\varepsilon = 10^{-2}$, оцінимо потрібну кількість ітерацій: $n > \log_2(2/10^{-2})$, тобто $n \geq 8$.

Метод дихотомії

Приклад (закінчення). $x^3 - x - 1 = 0.$

n	a_n	b_n	x_n	$f(x_n)$
1	0.0−	2.0+	1.0	−1.0
2	1.0−	2.0+	1.5	0.875
3	1.0−	1.5+	1.25	−0.296875
4	1.25−	1.5+	1.375	0.224609
5	1.25−	1.375+	1.3125	−0.051514
6	1.3125−	1.375+	1.34375	0.082611
7	1.3125−	1.34375+	1.328125	0.014576
8	1.3125−	1.328125+	1.3203125	−0.018711

Покрокове виконання алгоритму дихотомії. Знаки після чисел показують знак f(x) у відповідній точці.

- Знайдене наближене значення x*= x_8 = 1.3203125:

$$|1.3203125 - x^*| < 10^{-2}.$$

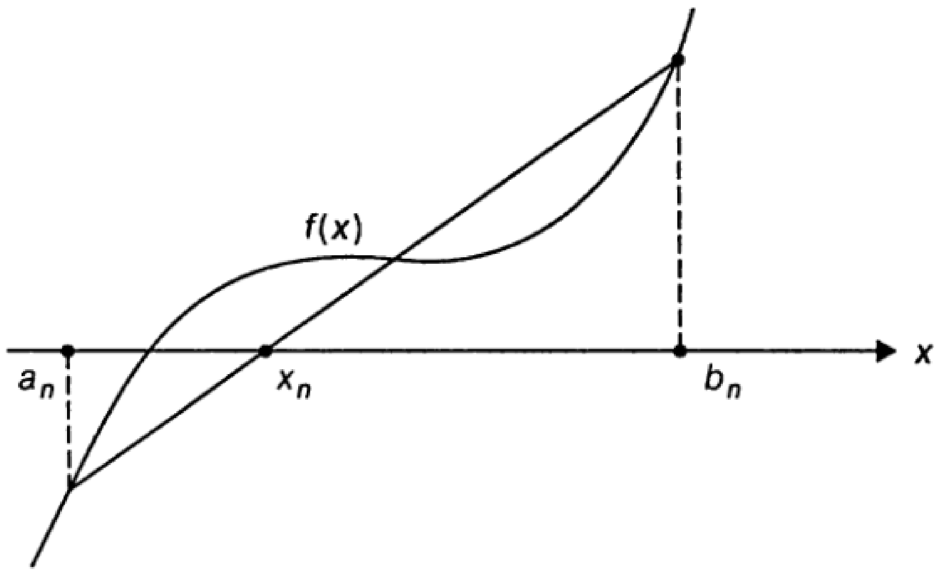
Метод дихотомії

- На практиці метод поділу навпіл для розв'язання рівнянь майже не застосовують через його повільну порівняно з іншими методами збіжність.
- Окрім того, його не можна розповсюдити на рівняння загальнішого вигляду чи системи рівнянь.
- Але метод має і свої переваги: він завжди збігається до кореня при будь-якому коректному виборі початкового інтервала та не використовує похідну (як деякі ефективніші методи).
- На відміну від бінарного пошуку, який вимагає попередньої відсортованості даних, метод дихотомії не потребує монотонності від функції.

Метод хорд

- Також: метод січних, метод хибного положення, метод лінійної інтерполяції, regula falsi.
- Метод працює схожим на метод дихотомії чином.
- Маємо неперервну функцію $f(x)$ та відрізок [a,b], на кінцях якого вона приймає різні знаки. Але в якості проміжної точки береться не середина відрізку, а точка перетину осі абсцис з прямою, що проходить через точки $(a, f(a))$ та $(b, f(b))$.
- По суті, функція $f(x)$ інтерполюється на відрізку многочленом першого степеня, а за чергове наближення береться його корінь (схожість з інтерполяційним пошуком).
- Метод завжди збігається.

Метод хорд



Вибір проміжної точки методом хорд

- Формула обчислення точки перетину прямої з віссю абсцис при черговій ітерації:

$$x_n = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}.$$

Метод хорд

Приклад. Ілюстрація для $x^3 - x - 1 = 0$.

n	a_n	b_n	x_n	$f(x_n)$
1	0.0−	2.0+	0.333333	−1.296296
2	0.333333−	2.0+	0.676471	−1.366909
3	0.676471−	2.0+	0.960619	−1.074171
4	0.960619−	2.0+	1.144425	−0.645561
5	1.144425−	2.0+	1.242259	−0.325196
6	1.242259−	2.0+	1.288532	−0.149163
7	1.288532−	2.0+	1.309142	−0.065464
8	1.309142−	2.0+	1.318071	−0.028173

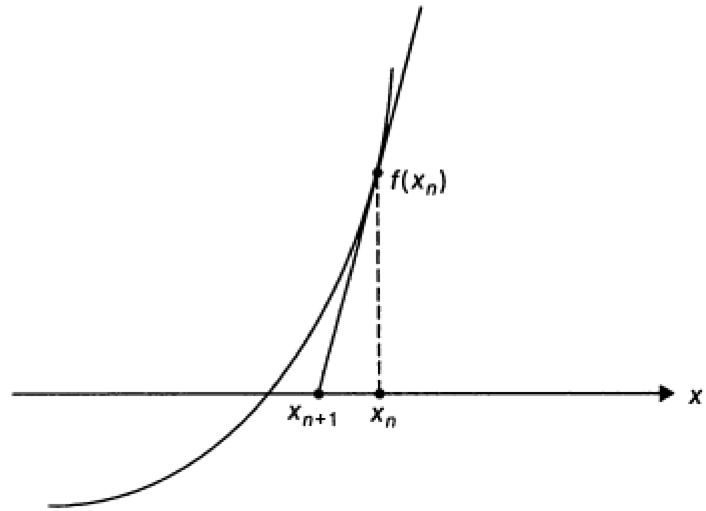
Перші вісім кроків ітерації методу січних

- Зазвичай метод швидко збігається, однак можливі вироджені випадки, коли він працюватиме повільніше за дихотомію (як в прикладі).

Метод Ньютона

- Також: метод Ньютона-Рафсона, метод дотичних.
- Швидко збіжний метод, застосовний до багатьох типів рівнянь і систем рівнянь.
- В нашому випадку черговий елемент x_{n+1} послідовності наближень розв’язку рівняння f(x)=0 визначається як перетин дотичної до графіка функції f(x) в точці x_n з віссю абсцис.
- В більшості випадків метод гарантує швидку збіжність при виборі “достатньо близького” до кореня початкового наближення x_0.
- Втім послідовність може збігатися і при далекому від кореня початковому наближенні.

Метод Ньютона



Ітерація методом Ньютона

- Формула для елементів послідовності наближень методом Ньютона:

$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)} \quad \text{для}\ n = 0,1,\ldots.$$

Метод Ньютона

Приклад. Ілюстрація для $x^3 - x - 1 = 0$.

Послідовність наближень буде обчислюватись так:

$$x_{n+1} = x_n + \frac{x_n^3 - x_n - 1}{3x_n^2 - 1}.$$

Початковим наближенням виберемо $x_0 = 2$.

n	x_n	x_{n+1}	$f(x_{n+1})$
0	2.0	1.545455	1.145755
1	1.545455	1.359615	0.153705
2	1.359615	1.325801	0.004625
3	1.325801	1.324719	$4.7 \cdot 10^{-6}$
4	1.324719	1.324718	$5.0 \cdot 10^{-12}$

Кроки ітерації методу Ньютона.

Послідовність збігається до кореня набагато швидше.

Метод Ньютона

- На кожній ітерації потрібно обчислювати нове значення функції та її похідної.
- Отримання похідної додатково може бути непростою задачею.
- Значення похідної в точці не повинно бути 0.
- Метод не вказує границь, в яких лежить корінь.
- При довільному виборі початкового наближення можна отримати розбіжну послідовність.
- Тому при застосуванні методу слід попередньо проаналізувати функцію.
- При виконанні всіх умов метод збігається дуже швидко.

Швидке множення чисел

- Існує ряд задач (зокрема в криптології), в яких треба оперувати дуже великими цілими числами з розміром понад 100 десяткових цифр.
- Зрозуміло, такі числа не вдасться розмістити в одному машинному слові і для їх обробки необхідні спеціальні підходи.
- Розглянемо алгоритм множення подібних великих чисел.
- Якщо використати звичайний алгоритм множення в стовпчик, то результат перемноження двох n-значних чисел (різна кількість цифр вирівнюється приписуванням спереду нулів) отримується за n^2 множень (кожна з n цифр першого числа множиться на кожну з n цифр другого).

Алгоритм Карацуби

- Довгий час вважалося, що нижньою оцінкою подібного множення є $\Omega(n^2)$ – гіпотезу сформулював А. Колмогоров і вона вважалася правдоподібною.
- Однак невдовзі А. Карацуба винайшов новий спосіб множення двох n-значних цілих, який працював швидше ніж за n^2.
- В основі методу лежить декомпозиція.
- Це був перший з алгоритмів швидкого множення, згодом стали з’являтися ще швидші методи.
- В основі їх, а також і методу швидкого множення матриць Штрассена та швидкого перетворення Фур’є так само лежить ідея А. Карацуби.

Алгоритм Карацуби

- Продемонструємо ідею на прикладі множення конкретної пари двоцифрових чисел.
- Візьмемо числа 23 та 14. Представимо їх так:

$$23 = 2 \cdot 10^1 + 3 \cdot 10^0, \quad 14 = 1 \cdot 10^1 + 4 \cdot 10^0.$$

- Перемножимо їх:

$$23 \cdot 14 = \left(2 \cdot 10^1 + 3 \cdot 10^0\right) \cdot \left(1 \cdot 10^1 + 4 \cdot 10^0\right) =$$
$$= (2 \cdot 1) \cdot 10^2 + (3 \cdot 1 + 2 \cdot 4) \cdot 10^1 + (3 \cdot 4) \cdot 10^0.$$

- Але середній член можна обчислити лише за одне множення! Ми скористаємося вже відомими обчисленими добутками $(2 \cdot 1)$ та $(3 \cdot 4)$:

$$3 \cdot 1 + 2 \cdot 4 = (2 + 3) \cdot (1 + 4) - (2 \cdot 1) - (3 \cdot 4)\,.$$

Алгоритм Карацуби

- В загальному вигляді для пари двоцифрових чисел $a = a_1 a_0$ та $b = b_1 b_0$ їх добуток

$$c = a \cdot b = c_2 \cdot 10^2 + c_1 \cdot 10^1 + c_0,$$

де:

$$c_2 = a_1 \cdot b_1$$

$$c_0 = a_0 \cdot b_0$$

$$c_1 = (a_1 + a_0) \cdot (b_1 + b_0) - (c_2 + c_0)$$

- Узагальнимо отримане на добуток двох *n*-значних цілих, якщо *n* – парне додатне число.
- В записі вигляду $a = a_1 a_0$ будемо розуміти під $\boldsymbol{a_1}$ першу половину цифр числа $\boldsymbol{a}$, під $\boldsymbol{a_0}$ – другу.

Алгоритм Карацуби

- Таким чином $a = a_1 \cdot 10^{n/2} + a_0,$
  $$b = b_1 \cdot 10^{n/2} + b_0.$$
- Добуток:

$$c = a \cdot b = \left(a_1 \cdot 10^{n/2} + a_0\right) \cdot \left(b_1 \cdot 10^{n/2} + b_0\right) =$$
$$= (a_1 \cdot b_1) \cdot 10^n + (a_1 \cdot b_0 + a_0 \cdot b_1) \cdot 10^{n/2} + (a_0 \cdot b_0) =$$
$$= c_2 \cdot 10^n + c_1 10^{n/2} + c_0,$$

де

$$c_2 = a_1 \cdot b_1$$
$$c_0 = a_0 \cdot b_0$$
$$c_1 = (a_1 + a_0) \cdot (b_1 + b_0) - (c_2 + c_0)$$

Алгоритм Карацуби

- Якщо $n/2$ парне, то значення c_2, c_1 та c_0 можна обчислити за цим же алгоритмом.
- Отже, за умови $n = 2^k$, отримаємо рекурсивний алгоритм для обчислення добутку пари n-значних цілих чисел.
- Рекурсію можна завершити і до досягнення $n = 1$ – коли числа стануть достатньо малими для безпосереднього їх перемноження.
- Оцінимо кількість множень в алгоритмі:

$$M(n) = 3M(n/2) \text{ при } n > 1,\ M(1) = 1.$$

- Розв'язок рекурентного співвідношення за основною теоремою $\Theta\left(n^{\log_2 3}\right) \approx \Theta(n^{1,585})$.

Алгоритм Карацуби

- Алгоритм буде працювати швидше за класичний для великих чисел (довжиною в сотні цифр).
- В той же час при числах середньої довжини він може працювати навіть довше за рахунок більшого вкладу констант.
- Для порівняння, нехай множаться два 1024-цифрових числа ($n = 1024 = 2^{10}$).
  - Класичний алгоритм вимагає множень

    $(2^{10})^2 = 1.048.576$.
  - Алгоритму Карацуби досить множень

    $3^{10} = 59.049$.

Швидке множення матриць

- Так само, як і у випадку з множенням великих чисел, виявилося, що класичний спосіб множення матриць за n^3 часу не є оптимальним.
- Першим на це вказав Ф. Штрассен і запропонував відповідний швидший алгоритм.
- В основі його – ідея швидкого множення із застосуванням підходу «розділяй та владарюй».
- Спробуємо оцінити, за рахунок чого можна скоротити кількість множень.

Швидке множення матриць

- Для більш зручного розбиття матриць розміром *n* x *n* припустимо, що *n* = 2^k (за необхідності матрицю можна доповнити нулями до потрібної розмірності).
- Тоді кожну з матриць (A, B та їх добуток C) можна розділити на 4 блоки-підматриці розміром *n*/2 x *n*/2:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- Добуток C = AB запишеться так:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} .$$

Швидке множення матриць

- Обчислення добутку зведеться до визначення

$$
\begin{aligned}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \ , \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \ , \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \ , \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \ .
\end{aligned}
$$

- При обчисленні кожного з 4 блоків відбувається 2 множення і 1 додавання матриць розміру $n/2$ x $n/2$.
- На основі цих співвідношень опишемо базовий прямолінійний рекурсивний алгоритм множення матриць.

Швидке множення матриць

SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

```
n = A.rows
Пусть C – новая матрица размером n × n
if n == 1
    c11 = a11 · b11
else разбиение A, B и C
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

Швидке множення матриць

- Визначимо рекурентне співвідношення для оцінки часу роботи алгоритму T(n).
- База рекурсії – константний час.
- Відбувається 8 рекурсивних викликів для задач розміром $n/2$ – часовий вклад T($n/2$) кожен.
- Додавання матриць по $n^2/4$ елементів кожна дають вклад $\Theta(n^2)$.
- Розбиття матриць може відбуватися як через обчислення індексів (час $\Theta(1)$), так і шляхом копіювання матриць (час $\Theta(n^2)$).
- В будь-якому випадку на операції розбиття-злиття піде $\Theta(n^2)$ часу (різниця лише в розмірах константи).

Швидке множення матриць

- Рекурентне співвідношення матиме вигляд

$$T(n) = \begin{cases} \Theta(1)\ , & n = 1\ , \\ 8T(n/2) + \Theta(n^2)\ , & n > 1\ . \end{cases}$$

- Розв'язок його за основною теоремою $\Theta(n^3)$.
- Тут найкритичніший вклад дає кількість рекурсивних викликів.
- Ключовим моментом є спроба зменшити кількість рекурсивних множень.
- Ф. Штрассен запропонував виконувати не 8, а 7 множень матриць розміру $n/2$ x $n/2$ за рахунок збільшення на постійне значення числа додавань матриць (при цьому зросте константа в оцінці $\Theta(n^2)$).

Метод Штрассена

- Введемо наступні матриці:

$$M_1 = (\mathrm{A}_{11} + \mathrm{A}_{22}) \cdot (\mathrm{B}_{11} + \mathrm{B}_{22})$$
$$M_2 = (\mathrm{A}_{21} + \mathrm{A}_{22}) \cdot B_{11}$$
$$M_3 = A_{11} \cdot (\mathrm{B}_{12} - \mathrm{B}_{22})$$
$$M_4 = A_{22} \cdot (\mathrm{B}_{21} - \mathrm{B}_{11})$$
$$M_5 = (\mathrm{A}_{11} + \mathrm{A}_{12}) \cdot B_{22}$$
$$M_6 = (\mathrm{A}_{21} - \mathrm{A}_{11}) \cdot (\mathrm{B}_{11} + \mathrm{B}_{12})$$
$$M_7 = (\mathrm{A}_{12} - \mathrm{A}_{22}) \cdot (\mathrm{B}_{21} + \mathrm{B}_{22})$$

- Тоді результуючі блоки обчислюються так:

$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 - M_2 + M_3 + M_6$$

Метод Штрассена

- Таким чином, обмінявши одне матричне множення на фіксовану кількість додавань матриць вдалося зменшити число рекурсивних викликів функції множення.
- Нове рекурентне співвідношення:

$$T(n) = \begin{cases} \Theta(1)\ , & n = 1\ , \\ 7T(n/2) + \Theta(n^2)\ , & n > 1\ . \end{cases}$$

- Тепер його розв'язок $\Theta\left(n^{\log_2 7}\right) \approx \Theta(n^{2{,}807})$.
- Замість 8 множень і 4 додавань базового прямолінійного алгоритму отримали 7 множень і 18 додавань/віднімань.

Метод Штрассена

- Алгоритм дає виграш на великих щільних матрицях.
- При невеликих розмірах матриць розмір прихованої константи переважує кубічну складність класичного способу множення, тому на практиці при досягненні «точки перетину» (в сенсі розмірності матриць; залежить від конкретної системи) відбувається перехід від методу Штрассена до стандартного множення.
- Для множення розріджених матриць є спеціальні ефективніші алгоритми.
- В деяких випадках метод може накопичувати більші числові похибки через обмежену точність машинних обчислень (менша чисельна стійкість).

Метод Штрассена

- На сьогодні існують ще швидші алгоритми (Копперсміта-Вінограда та ряд його покращень) з часом $O(n^{2,37})$, але як ще більше наблизитися до оптимальної роботи за $\Omega(n^2)$ поки не відомо.
- Ці алгоритми, на відміну від методу Штрассена, становлять чисто теоретичний інтерес, оскільки дають перевагу на настільки великих матрицях, що їх навіть сучасна техніка ще не в змозі обробити.

Атомний удар по горобцям

- Розглянемо ще більш загальний метод розв’язання рекурентних співвідношень на основі підходу «розділяй та владарюй» (Акра-Баззі).
- Нехай маємо рекурентне співвідношення у формі

$$T(n) = \sum_{i=1}^{k} a_i T(n/b_i) + f(n),$$

де k – додатна ціла константа,

$a_i > 0$, $b_i > 1$ – константи,

$f(n) = \Omega(n^c)$, $f(n) = O(n^d)$ для деяких констант $0 < c \leq d$ (умови поліноміального зростання),

$T(\Theta(1)) = \Theta(1)$.

Метод Акра-Баззі

- Розв'язок такого співвідношення матиме вигляд

$$T(n) = \Theta\left(n^\rho\left(1 + \int_1^n \frac{f(u)}{u^{\rho+1}}\,du\right)\right),$$

де ρ – єдине дійсне число (воно завжди існуватиме), що задовольняє

$$\sum_{i=1}^{k} a_i / b_i^\rho = 1$$

- Метод розв'яже майже всяке рекурентне співвідношення типу «розділяй та владарюй». (Приклад винятку: $T(n) = \sqrt{n}\mathrm{T}(\sqrt{n}) + n$.)
- Основний метод є частковим випадком теореми Акра-Баззі.

Метод Акра-Баззі

Приклад 1. Розглянемо рекурентне співвідношення

$$T(n) = T(3n/4) + T(n/4) + n$$

Рівняння

$$(3/4)^{\rho} + (1/4)^{\rho} = 1$$

має єдиний розв'язок $\rho = 1$ і тому

$$T(n) = \Theta\left(n\left(1 + \int_1^n \frac{1}{u}\,du\right)\right) = O(n \log n).$$

Метод Акра-Баззі

Приклад 2. Розглянемо рекурентне співвідношення

$$T(n) = T(n/5) + T(7n/10) + n$$

Рівняння

$$(1/5)^\rho + (7/10)^\rho = 1$$

не має аналітичного розв'язку. Однак можна помітити, що функція $(1/5)^x + (7/10)^x$ спадна, тому $0 < \rho < 1$. Отримуємо

$$\int_1^n \frac{f(u)}{u^{\rho+1}}\,du = \int_1^n u^{-\rho}\,du = \frac{u^{1-\rho}}{1-\rho}\bigg|_{u=1}^{n} = \frac{n^{1-\rho}-1}{1-\rho} = \Theta(n^{1-\rho}),$$

$$T(n) = \Theta(n^\rho \cdot (1 + \Theta(n^{1-\rho}))) = \Theta(n).$$