НАУЧНЫЕ ВЫЧИСЛЕНИЯ

Использование внешних библиотек и стандартных солверов. Точность вычислений. Поиск корней уравнений.

К. Владимиров, Yadro, 2024
mail-to: konstantin.vladimirov@gmail.com

СЕМИНАР 8.1

Матричные вычисления и линейное программирование

Мотивирующий пример: определитель

- Что такое определитель?
- Допустим все коэффициенты целые. Может ли определитель не быть целым?
- Как вы подсчитаете определитель для матрицы $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$?
- А что насчёт $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$?
- Как вы запрограммируете такое вычисление?
- Какая будет алгоритмическая сложность предлагаемого решения?
- Как вы будете тестировать ваш алгоритм?

Генерация интересных матриц

- В среднем генерируя матрицы со случайными элементами мы будем получать очень плохие тесты.
- Как получить матрицу желаемого размера $N \cdot N$ с заданным определителем?
- Полезные свойства:
  - Определитель треугольной матрицы равен произведению диагональных элементов.
  - Сложение строки со строкой умноженной на коэффициент не меняет определитель.
- Попробуйте написать такую программу.

Вычисление определителя

- Разложение по строке (столбцу) это рекурсивное вычисление определителя меньшей матрицы и умножение на элемент строки с правильным знаком

$$det\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = a_{11} \cdot det\begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix} - a_{12} \cdot det\begin{pmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{pmatrix} + \cdots$$

- Оцените алгоритмическую сложность этого метода?

LU-декомпозиция

- Основан на свойстве определителя, что прибавление столбца или вычитание столбца ничего не меняет.

$$det\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = det\begin{pmatrix} a_{11} & a_{12} \\ 0 & a_{22} - a_{12}\frac{a_{21}}{a_{11}} \end{pmatrix}$$

- У треугольной матрицы определитель равен произведению элементов на главной диагонали.
- Промежуточный тип таким образом разумно делать с плавающей точкой даже для целочисленных матриц.
- Большая часть научных вычислений производится с плавающей точкой.

Внезапная проблема

- Что если первый элемент слишком маленький?
- Примените ваш алгоритм к $\begin{pmatrix} 0.00001 & 100.00 \\ 100.00 & 100.00 \end{pmatrix}$
- Попробуйте увеличивать и уменьшать первый элемент и наблюдать размер ошибки.
- К слову, а что если он вообще нулевой?

Алгоритм D – full pivoting

```
// Input: M -- matrix NxN

for (current = 0; current < N; ++current) {
  // max from [(N - current) x (N - current)] submatrix
  (max, col, row) = max_submatrix_element(M, current);
  swap_columns(M, current, col); // sometimes * (-1)
  swap_rows(M, current, row);
  pivot = diagonal_element(M, current);
  if (fabs(pivot) < epsilon)
    abort(); // elimination not possible
  eliminate(M, current, pivot);
}

// Output: product of the main diagonal
```

Problem DT – определитель матрицы

- На входе матрица в обычном представлении.
- На выходе её определитель, подсчитанный методом Гаусса-Жордана с полным пивотингом.

Внезапное решение problem DT

- На контесте оно не зайдёт, но можно попробовать локально.

```
lapack_int LAPACKE_dgetrf(int matrix_layout,
  lapack_int m, lapack_int n, double *a,
  lapack_int lda, lapack_int *p);
```

- Библиотека LAPACK и её C-интерфейс LAPACKE является одной из старейших и наиболее уважаемых научных библиотек.
- DGETRF сохраняет в матрице A разложение на LU, а в массиве P перестановки.

$$\begin{matrix} * & * & * \\ * & * & * \\ * & * & * \end{matrix} = P \times \begin{matrix} 1 & 0 & 0 \\ * & 1 & 0 \\ * & * & 1 \end{matrix} \times \begin{matrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{matrix}$$

test_dgetrf.c

Библиотека BLAS

- Первый уровень: операции над векторами.

Операция	Смысл
`SAXPY, DAXPY,` `CAXPY, ZAXPY`	$y = ax + y$
`SDOT, DDOT, ...`	dot product
`SNRM2, DNRM2,`	Euclidean norm
`SASUM, DASUM,`	Sum of modules
`ISAMAX, IDAMAX`	Index of max abs
`SROTG, DROTG,`	Given's rotation setup
`SROT, DROT, ...`	Given's rotation apply

S: Single	**D**: Double
C: Single Complex	**Z**: Double Complex

$$G_{1,2} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$G_{2,3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

https://www.netlib.org/blas

Библиотека BLAS

- Второй уровень: операции вектор-матрица.

Операция	Смысл
`SGER, DGER, ...`	$A = \alpha x y^T + A$
`SSYR, DSYR, ...`	$A = \alpha x x^T + A$
`SSYR2, ...`	$A = \alpha x y^T + \alpha y x^T + A$

`SGEMV, DGEMV,`
`SSYMV, DSYMV,`
`...`
	$y = \alpha A x + \beta y$
	$y = \alpha A^T x + \beta y$

S: Single	**D**: Double
C: Single Complex	**Z**: Double Complex

GE	General
SY	Symmetric
TR	Triangular
GB	General banded
TB	Triangular banded
SP	Symmetric packed
TP	Triangular packed

- Операции оптимизированы под свой тип матриц. Иногда отличается интерфейс как в SSYR vs SGER.

Библиотека BLAS

- Третий уровень: операции матрица-матрица.

Операция	Смысл
SGEMM, DGEMM, SSYMM, DSYMM, STRMM, ...	$C = \alpha AB + \beta C$ $C = \alpha A^T B + \beta C$ $C = \alpha AB^T + \beta C$
STRSM, DTRSM, CTRSM, ZTRSM	$Ax = \alpha B,\ A^T x = \alpha B$ $xA = \alpha B,\ xA^T = \alpha B$
CSYRK, ...	$C = \alpha AA^T + \beta C$
CSYR2K, ...	$C = \alpha AB^T + \alpha BA^T + \beta C$

S: Single	**D**: Double
C: Single Complex	**Z**: Double Complex

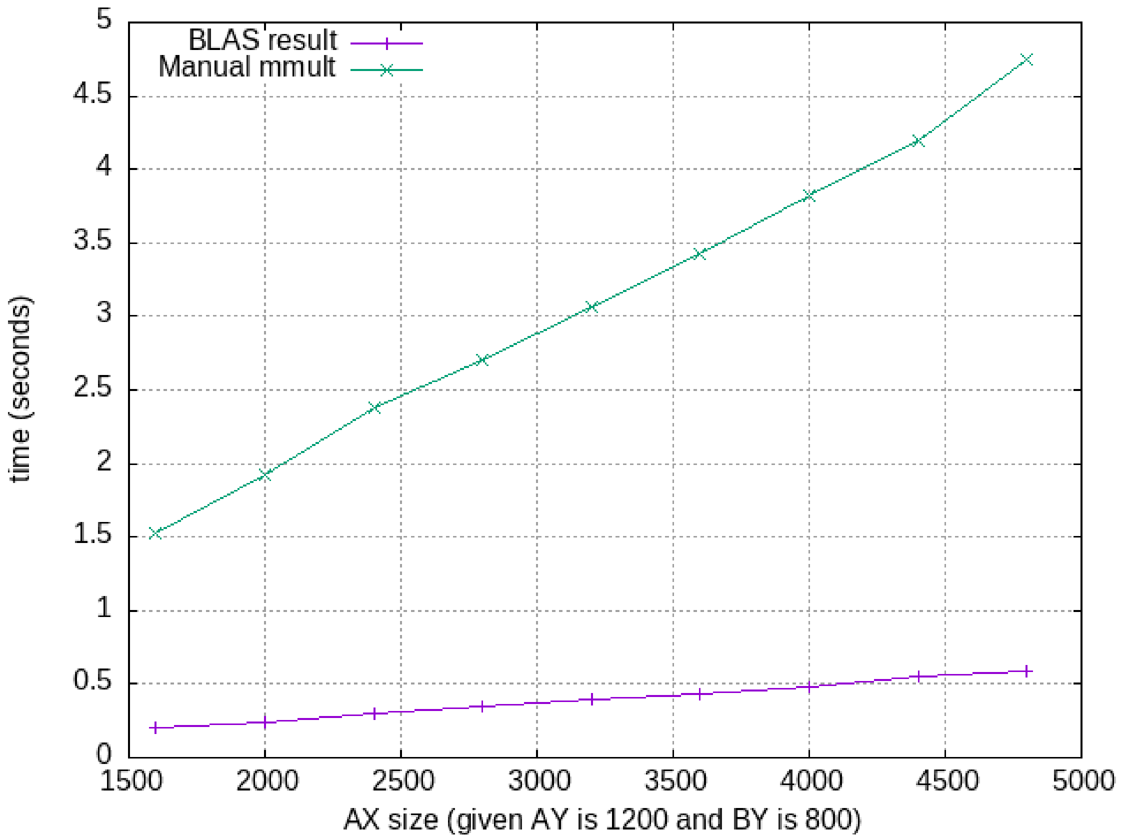
GE	General
SY	Symmetric
TR	Triangular
GB	General banded
TB	Triangular banded
SP	Symmetric packed
TP	Triangular packed

- Ещё специальные вещи вроде CHER2K где также происходит комплексное сопряжение.

test_mmult.c

Замеры: BLAS против нашего mmult

- На прошлом семинаре мы улучшили mmult с помощью транспонирования. Попробуем погоняться с библиотекой BLAS.
- Для замеров использовался тот же бенчмаркинг.
- Сам график был получен shell-скриптом.
- И построен утилитой gnuplot, которая также очень полезна.



mmult-blas.c

Основы gnuplot

```
set term png
set grid
set key left top
set xlabel "AX size (given AY is 1200 and BY is 800)"
set ylabel "time (seconds)"

# Blas vs manual impl

set output "sgemm_blas.png"
plot 'mm-blas.dat' with linespoints title 'BLAS result',\
     'mm-trans.dat' with linespoints t 'Manual mmult'
```

Перенесёмся в 1939 год, СССР

- Имеется n складов некоего однородного груза (например песок или руда) и m заводов. Известно:
- Сколько груза находится на i-м складе a_i.
- Сколько нужно груза j-му заводу b_j.
- Расстояние от склада до завода c_{ij}. Стоимость перевозки пропорциональна расстоянию и количеству груза: $x_{ij}c_{ij}$.
- Необходимо: составить наиболее дешёвый план перевозки.
- Как бы вы начали решать эту задачу?

Линейное программирование

- В общем случае: максимизировать $J(v) = C^T v$
- При ограничениях вида $Av \leq B$, где A это матрица и b это вектор-столбец.
- Также допустимы ограничения вида $Av = B$
- **Например для случая заводов**
- Минимизировать $c_{11}x_{11} + c_{12}x_{12} + \cdots + c_{nm}x_{nm}$
- При ограничениях:
- $x_{ij} \geq 0$, $x_{i1} + x_{i2} + \cdots + x_{im} \leq a_i$, $x_{1j} + x_{2j} + \cdots + x_{mj} \geq b_j$

Связь минмимума и максимума

- Если мы минимизируем (максимизируем) функционал $J(x)$ то:

$$\max\big(J(x)\big) = -\min(-J(x))$$

- То есть на самом деле задача максимизации и минимизации связаны и часто взаимозаменяемы.

$$x + y \leq 20$$

$$3x + 4y \leq 72$$

- Максимизировать $J(x, y) = 4x + 5y$, решение: 92
- Минимизировать $J(x, y) = -4x - 5y$, решение: -92

Простая модельная задача

- Максимизировать $J(x, y) = 4x + 5y$
- При ограничениях: $x \geq 0$, $y \geq 0$,

$x + y \leq 20$, $3x + 4y \leq 72$

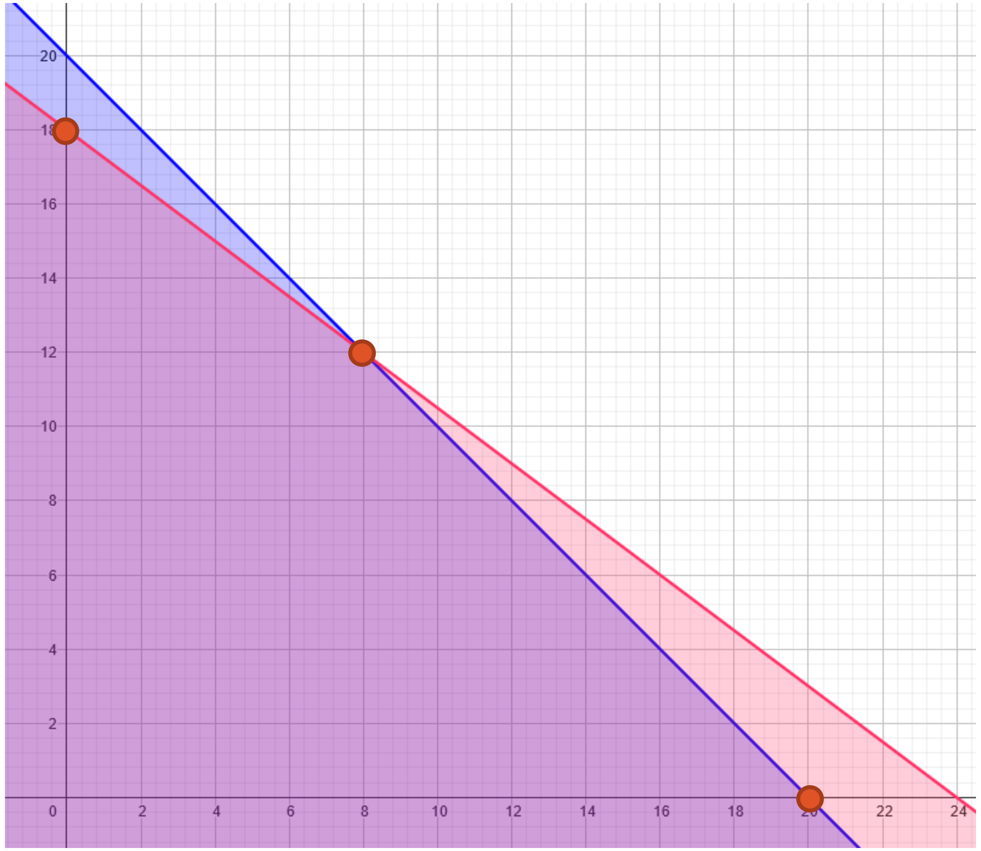
- Три особых точки:

$(0,18)$: $J(x, y) = 90$

$(8,12)$: $J(x, y) = 92$

$(20,0)$: $J(x, y) = 80$

- Поскольку задача линейная, максимум всегда в особой точке.



https://www.geogebra.org/calculator/wt2cbfay

Обсуждение

- Всё это очень хорошо пока размерность маленькая. Но что если вектор из десятков элементов? А если из тысяч?
- В общем случае мы имеем дело с задачей минимизации на многомерном выпуклом политопе (с каким-то неглупым перебором его вершин).
- Существуют самые разные методы решения. Реализация даже самых простых методов (например классического симплекс-метода) даже с помощью Lapack довольно сложна и неочевидна.

Линейное программирование: clp

- Минимизировать:

$x + 4y + 9z$

- При ограничениях:

$x + y \leq 5$

$x + z \geq 10$

$-y + z = 7$

$x \leq 4$

$-1 \leq y \leq 1$

```
NAME          TESTPROB
ROWS
 N  COST
 L  LIM1
 G  LIM2
 E  MYEQN
COLUMNS
    X      COST  1  LIM1   1  LIM2  1
    Y      COST  4  LIM1   1  MYEQN -1
    Z      COST  9  LIM2   1  MYEQN 1
RHS
    RHS1   LIM1  5  LIM2  10  MYEQN 7
BOUNDS
 UP BND1   X     4
 LO BND1   Y    -1
 UP BND1   Y     1
ENDATA
```


https://en.wikipedia.org/wiki/MPS_(format)

СЕМИНАР 8.2

Решение уравнений и вычисления функций (а также немного фракталов).

Работа с плавающими числами

- Избегайте сравнения на равенство.
- Будьте очень аккуратным с ошибками сложения.
- Учитывайте конечный размер плавающих чисел.
- Имейте в виду, что операции над числами не всегда возвращают числа.

Избегайте сравнивать на равенство

- Казалось бы сравнение должно выполняться, но увы.

```
d1 = 10.0;
d2 = sqrt(d1);
d3 = d2 * d2;

if (d1 == d3) {
  //  сюда мы можем и не попасть (зависит от округления)
}
```

- Правильный способ сравнивать: в пределах некоей погрешности.

```
if (fabs(d1 - d3) < tolerance)
```

Аккуратнее с ошибками сложения

- В следующем примере мы пытаемся вычислить как можно более точную производную, измельчая шаг до предела `double` диапазона

```
double h, cosval;

for (i = 1; i < 20; ++i) {
  h = pow(10.0, -i);
  cosval = (sin(1.0 + h) - sin(1.0)) / h;
  printf("%d:\t%.15lf\n", i, cosval); // всё лучше и лучше?
}

printf("True result: %.15lf\n", cos(1.0));
```

- Результаты при этом могут быть неожиданны.

Помните о конечности диапазона

- Даже числа одинарной точности предоставляют гигантские диапазоны. Но конечные.
- Это заметно при сложении с очень большими числами.

```
f = 16777216.0f; // 2^24
nextf = f + 1.0f; // побитово не отличается от f
```

- И при сложении с очень малыми.

```
fone = 1.0f;
feps = 0.00000005f;
fenext = fone + feps; // побитово не отличается от fone
```

- И тут встаёт вопрос: а с насколько маленькими можно складывать?

Ваш результат это не всегда число

- Следующий код позволяет получить бесконечность за конечное время.

```
double d = 1.79e+308;
double infd = 2.0 * d;

printf("d: %le\tinfd: %le\n", d, infd);
```

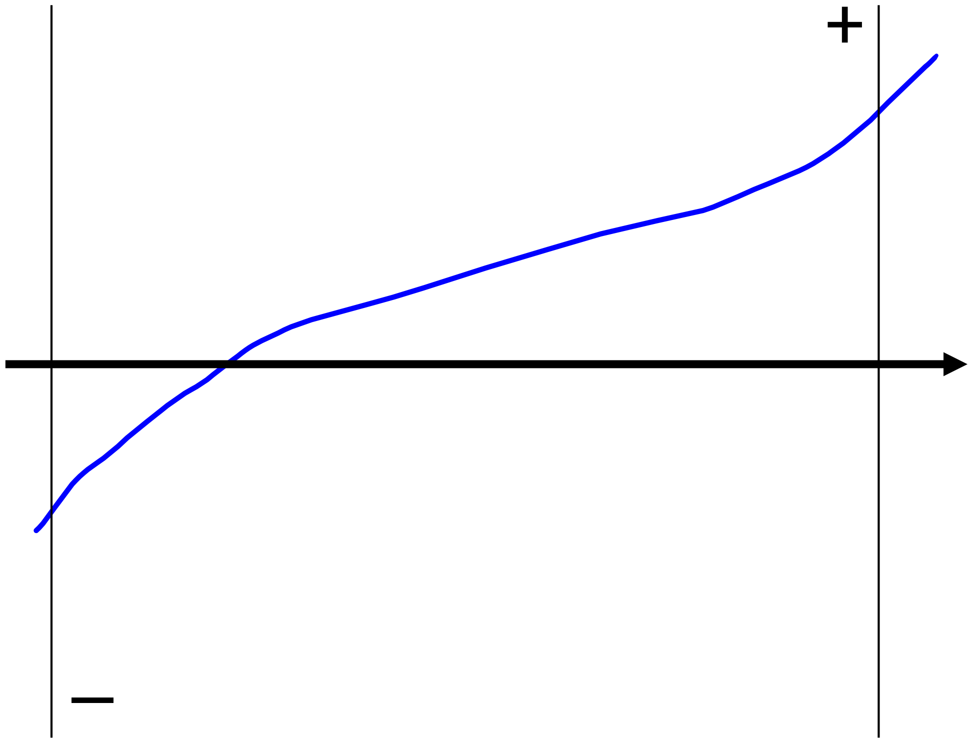
- Дальше над результатом будут работать другие правила.

Брекетинг корня

- Допустим вы хотите найти корень уравнения

$$x^2 * \sin(x) - 5x + 7 = 0$$

- Вы знаете, что он лежит где-то в диапазоне от -3 до 3
- Как вы его найдёте?



Дихотомия

- Допустим вы хотите найти корень уравнения.

$x^2 * \sin(x) - 5x + 7 = 0$

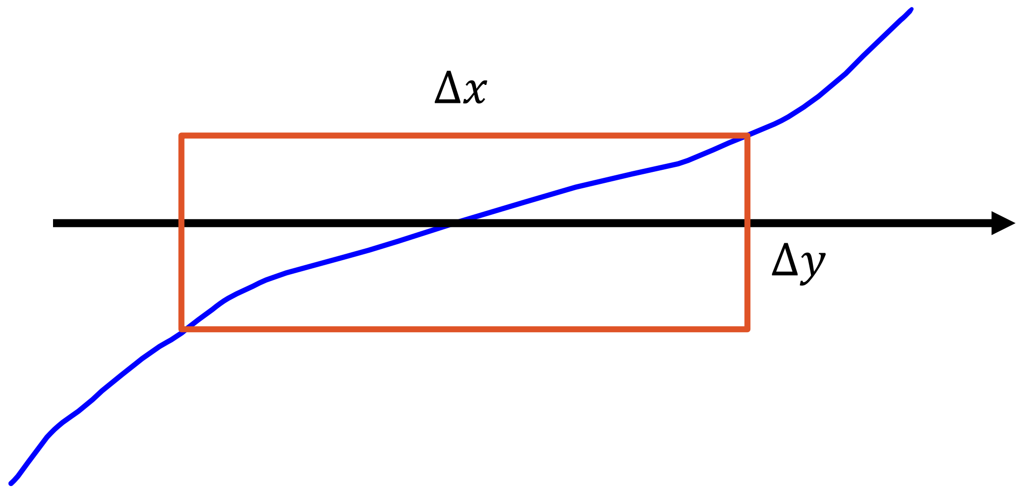
- Вы знаете, что он лежит где-то в диапазоне от -3 до 3.
- Как вы его найдёте?
- В данном случае нам повезло: $f(-3) > 0$ и $f(3) < 0$.
- Для решения можно воспользоваться дихотомией: на каждом шаге делить отрезок пополам и если там значение совпадает по знаку с левым, то брать правый интервал и наоборот.
- Это очень похоже на бинарный поиск в сортированном массиве.

Обсуждение

- Что вы будете сравнивать с выбранной вами точностью?

```
while (fabs(нечто) < epsilon) {
  дихотомия
}
```

- Вариантов собственно два: Δx и Δy.
- Опишите плюсы и минусы каждого из них.



Problem DH: дихотомия уравнений

- Дано уравнение

$$x^2 * \sin(x) - 5x + 7 = 0$$

- Найдите делением пополам корень, лежащий в интервале от `-3` до `3`
- Попробуйте теперь найти один из семи корней, лежщих в интервале от `-13.5` до `13`
- Творческая задача: сможете ли вы написать программу, которая находит все семь? Как вы сделаете чтобы она работала в общем случае?
- Проверить численный ответ можно здесь: https://www.wolframalpha.com/input/?i=x%5E2*sin(x)+-+5x+%2B+7+%3D+0

Дихотомия в целых числах?

- Может ли нам дихотомия помочь в поиске целочисленного квадратного корня?

```
unsigned isqrt(unsigned x); // return ⌊√x⌋

isqrt(0) == 0;

isqrt(1) == 1;
isqrt(2) == 1;
isqrt(3) == 1;

isqrt(4) == 2;
....
```

Целочисленный квадратный корень

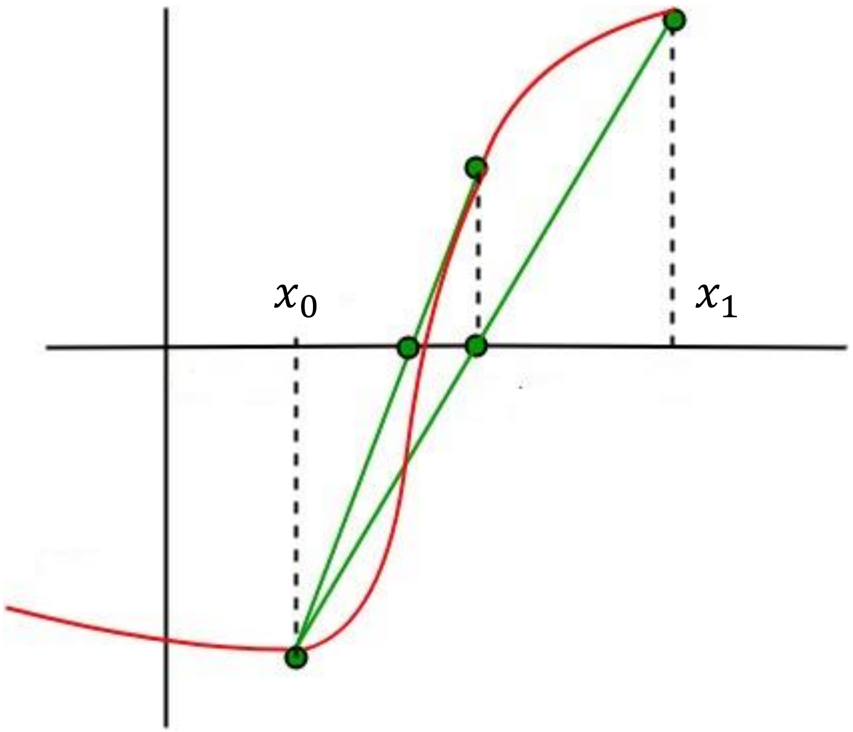
```
unsigned isqrt(unsigned x) {
  unsigned l = 0, r = x, mid;
  while (l < r) {
    mid = (l + r + 1) / 2;
    if (x < mid * mid)
      r = mid - 1;
    else
      l = mid;
  }
  return l;
}
```

```
unsigned isqrtd(unsigned x) {
  double d = sqrt(x);
  fesetround(FE_DOWNWARD);
  return rint(d);
}
```

- Как вы думаете, если забенчмаркать, то что быстрее?

Метод ложной позиции

- Пусть мы располагаем точками x_{k-1} и x_k, при этом $x_k > x_{k-1}$ и $s[f(x_{k-1})] \neq s[f(x_k)]$
- Посчитаем новую точку x.
- $x = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$
- А далее в зависимости от знака $s[f(x)]$ обновим:
- либо $x_{k+1} = x_k, x_k = x$
- либо $x_{k+1} = x, x_k = x_{k-1}$



Regula falsi для целых чисел?

- Группа людей вместе покупает некую вещь.
- Если все люди платят 7 монет, то им не хватает 4 монеты.
- Если все люди платят 8 монет, то избыток 3 монеты.
- Найти сколько стоит эта вещь и сколько людей её покупает.
- Конечно тут можно составить линейное диофантово уравнение и рещить алгоритмом Евклида.
- Сможете ли вы придумать как приложить метод ложной позиции к этой задаче?

Обсуждение: нарушим брекетинг?

- Исследуя regula falsi, люди обнаружили интересную вещь. Из-за того, что в худшем случае один её конец в итоге зафиксирован, сходимость у этого метода не лучше, чем у дихотомии.
- Идея метода секущих в том, что мы забываем про брекетинг и всегда делаем:

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

- То есть обновляем обе границы в любом случае.
- Да мы можем потеряться и разойтись, но зато скорость сходимости улучшается в разы.

Алгоритм SC – метод Риддерса

- Метод Риддерса основан на улучшенном fals position, но сходится быстрее.

```
typedef double (*func_t)(double x);

double fsgn(double x) { return signbit(x) ? -1.0 : 1.0; }

double secant(func_t f, double xleft, double xright) {
  assert(fsgn(f(xleft)) != fsgn(f(xright)));
  // далее в цикле:
  // xmid = (xleft + xright) / 2.0;
  // fl = f(xleft); fr = f(xright); fm = f(xmid);
  // xnew = xmid + (xmid-xleft) * fsgn(fl - fr) * fm / sqrt(fm * fm – fl * fr);
  // заменяем xleft = xnew или xright = xnew в зависимости от знака f(xnew)
  // проверяем условие выхода из цикла fabs(f(xnew)) < precision
  return xnew;
}
```

Problem EC – исследование сходимости

- Замерьте количество итераций методом Риддерса и дихотомией для уравнения.

$$x^2 * \sin(x) - 5x + 7 = 0$$

- Попробуйте разные начальные интервалы.
- Попробуйте float и double precision.
- Подтверждают ли ваши результаты теоретическое превосходство метода?

Обсуждение

- Рассмотрим уравнение

$$x^2 + e^x - 0.827185 = 0$$

- У него два действительных корня, но довольно сложно выбрать два значения, в которых функция принимала бы разные знаки (попробуйте!)
- Что делать в этом случае?

Внезапная идея

- У нас уже был метод где мы забыли про брекетинг.

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

- Может быть мы просто его запустим с любой точки?
- Это так себе идея, но её можно сделать рабочей, заменив разность на производную.

Алгоритм N – метод Ньютона

```
struct func_deriv { double func; double der; };

// возвращает значение функции и производной в точке x
typedef struct func_deriv (*fder_t) (double x);

double newton(fder_t f, double x) {

  // реализуйте самостоятельно
```

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

```
  return x;
}
```

Problem EN: решение методом Ньютона

- Даны два уравнения

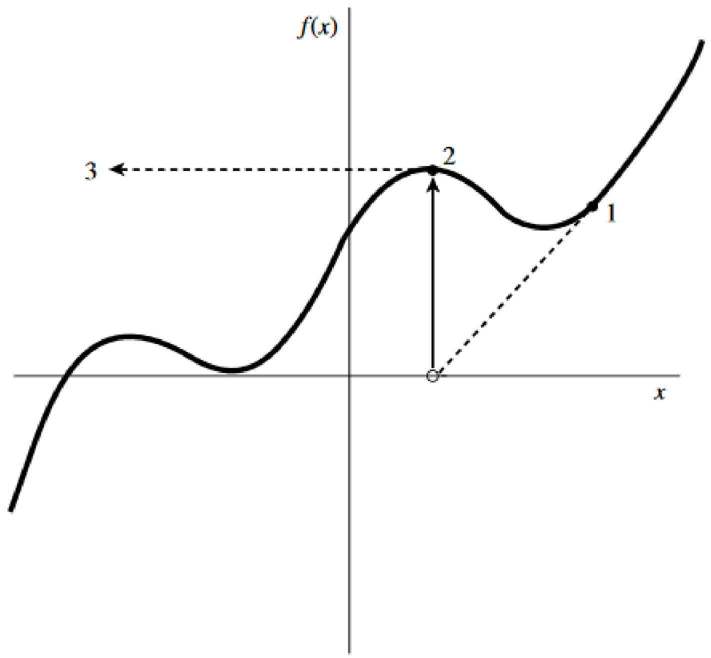
$x^2 + e^x - 0.827185 = 0$

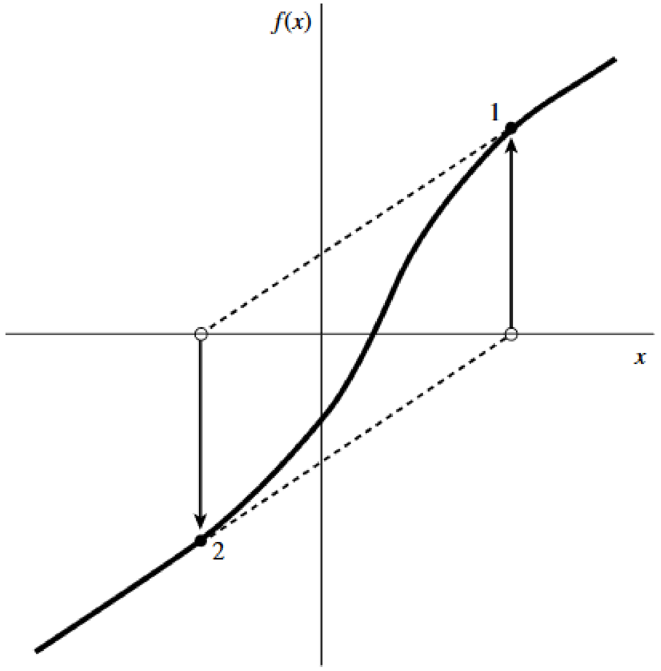
$x^2 * \sin(x) - 5x + 7 = 0$

- Решите оба используя алгоритм N
- Были ли у вас какие-нибудь сложности со вторым?

Проблемы со сходимостью

- У метода Ньютона есть проблемы со сходимостью (картинки из $[Numrec]$)





Вычисление функций

- Рассмотрим идеально простой пример: вычислить квадратный корень.
- Дано: число a.
- Нужно найти такой x, что $x^2 = a$ т.е. найти ноль для функции $f(x) = x^2 - a$
- $x_{k+1} = x_k - \frac{{x_k}^2 - a}{2x_k} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$
- Это очень быстро сходится от правильного начального приближения.

Быстрый приближённый логарифм

- В следующей таблице приведены значения чисел и их логарифмов по основанию 2

Число	Представление	Представление минус `0x3f800000`	Логарифм
`1.0f`	`0x3f800000`	`0x00000000`	`0.0f`
`2.0f`	`0x40000000`	`0x00800000`	`1.0f`
`4.0f`	`0x40800000`	`0x01000000`	`2.0f`

- Из этого следует интересная формула
- `log(x) ≃ ([x as bits] - 0x3f800000) / 0x00800000`

Быстрые приближения

- Можно реализовать быстрый логарифм на базе формулы

```
log(x) ≃ (float) (as_uint(x) - 0x3f800000) / (float) 0x00800000
```

- Давайте потратим некоторое время на анализ того как это вообще может работать

Быстрые приближения

- Можно реализовать быстрый логарифм на базе формулы

`log(x) ≃ (float) ([x as bits] - 0x3f800000) / (float) 0x00800000`

- Реализуйте также быстрое возведение двойки в данную степень

2^x `≃ [((unsigned) (x * (float)0x00800000) + 0x3f800000) as float]`

- Подобная же магия возможна и для квадратного корня

$\sqrt{x}$ `≃ [(([x as bits] >> 1) + (0x3f800000 >> 1)) as float]`

- Сравните с функциями логарифма, возведения в степень и извлечения корня стандартной библиотеки. Имеет ли на вашей системе эта магия реальный смысл?

Problem RI – инверсный корень

- Пусть дано a и надо найти $x = \frac{1}{\sqrt{a}}$
- Это всё равно, что решить уравнение $f(x) = \frac{1}{x^2} - a = 0$
- Напишите функцию

```
double inv_sqrt(double a);
```

- Не используйте при реализации стандартную функцию sqrt и деление, воспользуйтесь методом Ньютона
- Как вы будете тестировать вашу функцию?

Магический инверсный корень

- В качестве приближённого решения предыдущей проблемы будет работать следующая магическая процедура

```
float magic_inv_sqrt (float y) {
  double x2 = 0.5f * y;
  long i = to_long(y);
  i = 0x5f3759df - (i >> 1); // Magic!
  y = *(float *) &i;
  y  = y * (1.5f - (x2 * y * y)); // one additional Newton step
  return y;
}
```

- Догадываетесь ли вы как это работает?

Подробнее: https://en.wikipedia.org/wiki/Fast_inverse_square_root

Фракталы

- Несмотря на трудности которые создают проблемы со сходимостью, они же порождают фрактальную структуру
- Например рассмотрим в комплексных числах уравнение

$$z^3 - 1 = 0$$

- Для него есть такие z_0 для которых метод Ньютона сходится и такие, для которых нет
- Из-за нестабильного поведения около локальных максимумов, область сходимости образует самоподобную кривую, то есть собственно фрактал
- Рисунки таких фракталов на комплексной плоскости могут быть крайне красивы

Подробнее: https://en.wikipedia.org/wiki/Newton_fractal

Реализация фрактала Ньютона

- Функции в комплексных числах удобнее реализовать в комплексных числах.

```
static complex double next(complex double z) {
  complex double numerator = z * z * z - 1;
  complex double denominator = 3 * z * z;
  return z - numerator / denominator;
}
```

- Но что это за тип данных?

Работа с комплексными числами

```
complex double a, b, c;
double re, im;

a = CMPLX(1.0, 2.0);
b = CMPLX(3.0, 4.0);
c = a * b;

re = creal(c);
im = cimag(c);

printf("%lf + %lf * i\n", re, im);
```

Голоморфная динамика

- Во фрактале Ньютона мы рассмтариваем нечто вроде

$z \to \frac{z^3-1}{3z^2}$

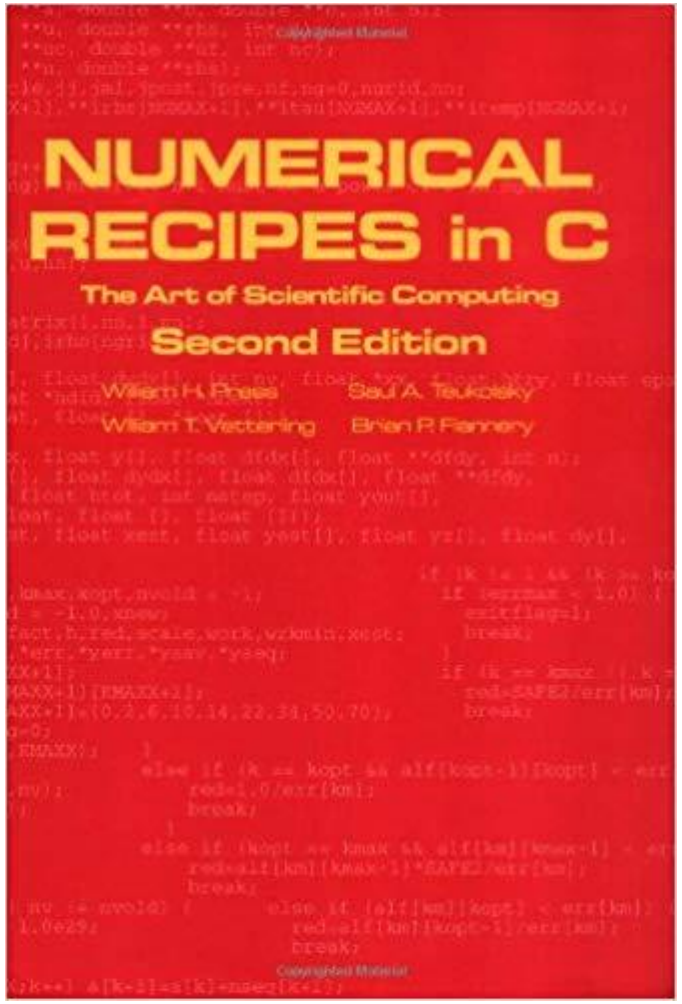
- Рассмотрим просто функции вида:

$z \to z^2 + c$

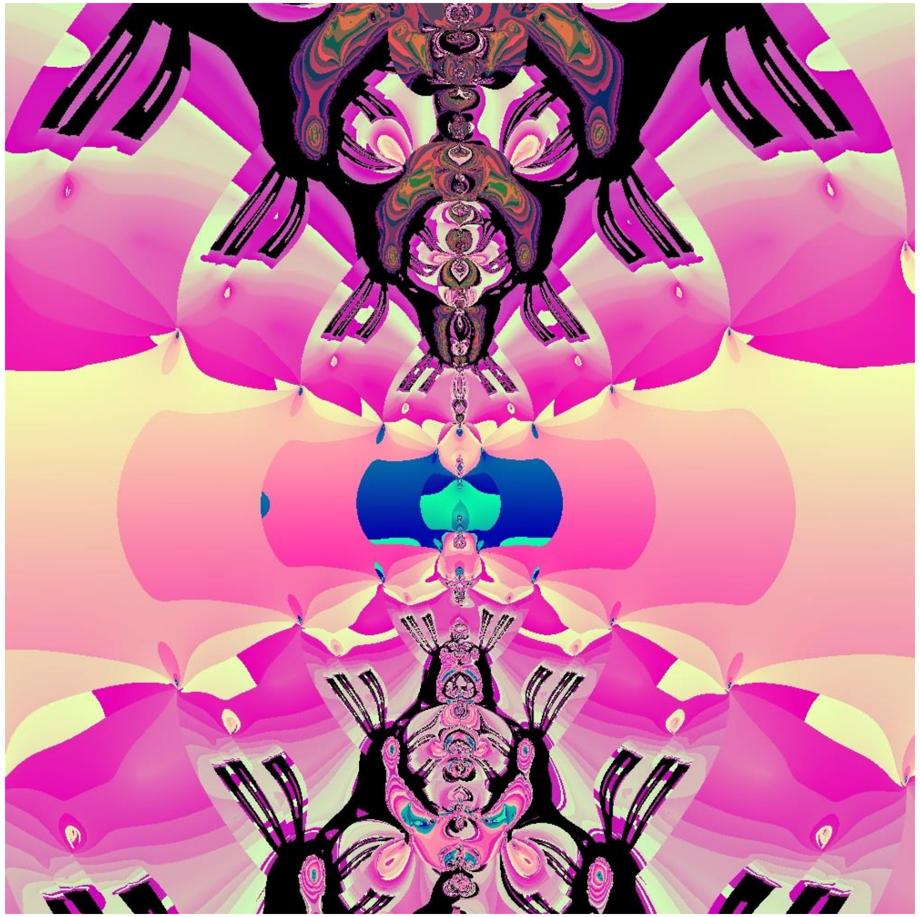
- Здесь c это какая-то комплексная константа.
- Как это отображение будет себя вести на комплексной плоскости: расходиться или сходиться и за сколько итераций?

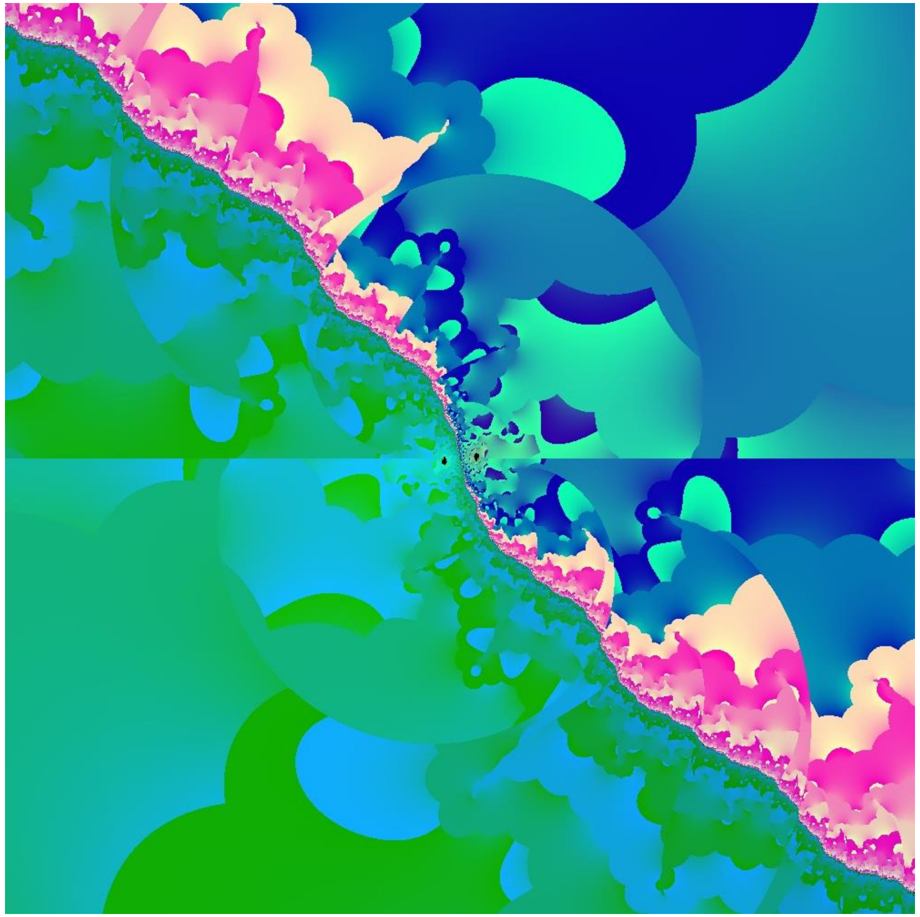
Литература

- $[C11]$ ISO/IEC – "Information technology – Programming languages – C", 2011
- $[IEEE]$ ANSI/IEEE Std 754 – "Standard for Binary Floating-Point Arithmetic", 1985
- $[K\&R]$ Brian W. Kernighan, Dennis Ritchie – The C programming language, 1988
- $[Numrec]$ W. Press, S. Teukolsky – Numerical Recipes in C, 2nd edition, 1992
- $[TIPS]$ John D. Cook – Five Tips for Floating Point Programming, 2014
- $[TRICKS]$ James F. Blinn – Floating point tricks, 1997



Примеры для более сложных функций





Источник: https://vk.com/fracgen