Численные методы и математическое моделирование

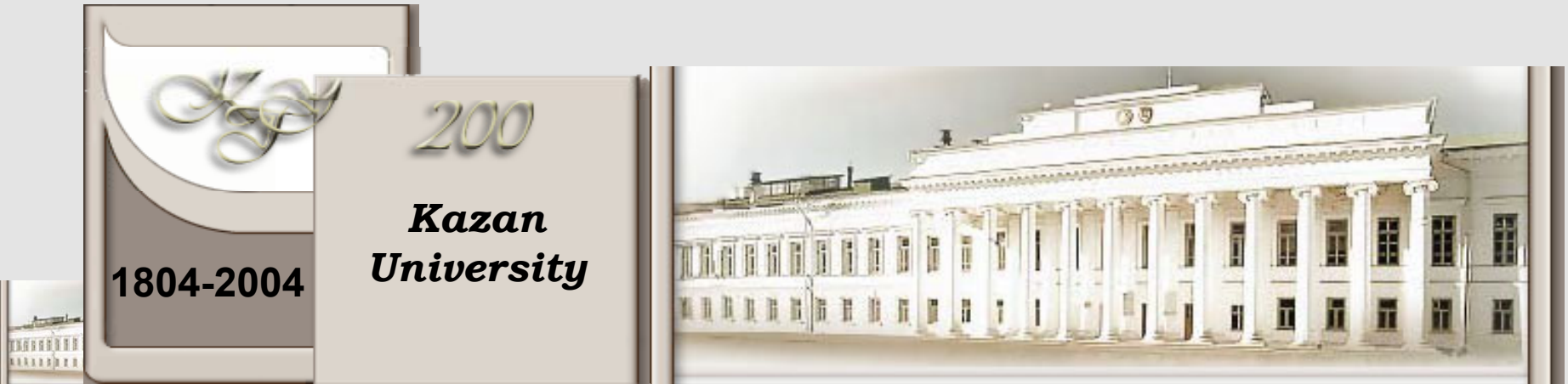
Ю.Н. Прошин

кафедра теоретической физики

Казанского государственного университета

yurii.proshin@ksu.ru

2004-2011, Казань



Численное решение: алгоритмы, методы и неприятности …

КУЛЬТУРА ВЫЧИСЛЕНИЙ НА ЭВМ

- До сих пор => Постановки задач и алгоритмы их решения.
- Однако, мы имеем цепочку «модель — алгоритм — программа».
- Одна из возможных причин несовпадения желаемого и получаемого

  => несовпадение машинной арифметики с обычной из-за конечности разрядной сетки ЭВМ.

Возникающие ошибки могут привести к большим неприятностям, если их не контролировать и не соблюдать некоторые элементарные правила организации вычислений. Правила эти неформальны и напоминают правила хорошего тона. Уровень их выполнения определяет уровень вычислительной культуры пользователя ЭВМ. Поясним на примерах основные из этих правил.

Пример 1.
Коммутативность, ассоциативность, …

Вычислим (на обычном калькуляторе или на ЭВМ)

$10^{16} + 1 - 10^{16}$, $10^{32} + 10^{10} - 10^{32}$

$10^{16} - 10^{16} + 1$, $10^{32} - 10^{32} + 10^{10}$

```
>> 10^16 + 1 - 10^16
ans =
   0
>> 10^32 + 10^10 - 10^32
ans =
   0
>>
```

```
>> single(10^8) + 1 - single(10^8)
ans =
   0
```

Таким образом, в машинной арифметике **нарушаются** законы **коммутативности и ассоциативности** действий.

Применимость основных выводов элементарной математики ставится под сомнение.

Пример 1.
Коммутативность, ассоциативность, …

Вычислим (на обычном калькуляторе или на ЭВМ)

$10^{16} + 1 - 10^{16}$, $10^{32} + 10^{10} - 10^{32}$

$10^{16} - 10^{16} + 1$, $10^{32} - 10^{32} + 10^{10}$

```
>> 10^16 + 1 - 10^16
ans =
   0
>> 10^32 + 10^10 - 10^32
ans =
   0
>>
```

```
>> 10^16 - 10^16 + 1
ans =
   1
>> 10^32 - 10^32 + 10^10
ans =
   1.0000e+010
>>
```

Таким образом, в машинной арифметике **нарушаются** законы **коммутативности и ассоциативности** действий.

Применимость основных выводов элементарной математики ставится под сомнение.

Пример 2. Предел.

Известно, что

$$\lim_{n\to\infty} e_n = e, \quad \text{где} \quad e_n = (1+1/n)^n, \quad e \approx 2.718281828\ldots$$

```
>> n=single([1 1e5 1e7 1.1e7 1.2e7 2e7 3e7])
n =
        1    100000    10000000    11000000    12000000    20000000    30000000

>> (1+1./n).^n
ans =
   2.0000    2.7220    3.2940    3.7110    4.1808    1.0000    1.0000
```

Вывод: при вычислениях с ЭВМ применимость основного понятия высшей математики — **предела** — также ставится под сомнение.

Правило 1.

Определение. Машинным эпсилоном называется наименьшее представимое в ЭВМ число ε, удовлетворяющее условию

$$1+\varepsilon>1, \text{ т. е. } \varepsilon_м=\min\{\varepsilon : 1+\varepsilon>1\}.$$

Правило 1.

Величина $\varepsilon_м$ характеризует наименьшую относительную погрешность вычислений и зависит от конкретной ЭВМ и разрядности вычислений (single, double,...).
Требовать **БОЛЬШЕГО** невозможно!

Задача
и комментарий к правилу 1.

Задача. Найти минимальное число ε_M, используемое компьютером при вычислениях по умолчанию, определить количество значащих цифр используемых при численных расчетах, выяснить возможности его увеличения (уменьшения). (На примере любого языка программирования **Си, Паскаль, Фортран, Дельфи** или вычислительного пакета **MatLab, Maple, Mathematica, MathCad, Origin, Derive**).

- Очевидно, если $\varepsilon_M > 10^{-k}$, то на данной ЭВМ нельзя гарантировать, что в результатах будет содержаться **не менее k** верных значащих цифр.
- Напомним, что цифра числа называется верной, если абсолютная погрешность числа не превосходит половины единицы того разряда, в котором эта цифра находится.

Чувствительность к исходным данным. **Корни полинома**

- Задача может быть *чувствительна* к малым ошибкам, допущенным при представлении исходных данных.
- Пример, корни уравнения $p(x) = (x-2)^2 = 0$ равны двум, при изменении свободного члена на малую величину $\varepsilon = 10^{-6} \Rightarrow (x-2)^2 = \varepsilon$ изменение в корнях много больше: $x_{1,2} = 2 \pm 10^{-3}$.
- Этот тип неустойчивости еще более выражен у полиномов более высокой степени. Корни следующего полиномиального уравнения $p(x) = 0$, где $p(x) = (x-1)\cdot(x-2)\cdot\ldots\cdot(x-20) = x^{20} - 210x^{19} + \ldots$ суть *реальные* числа от 1 до 20 и хорошо разделены.

Задача: Измените коэффициент при x^{19}: $(-210) \Rightarrow (-210 + 10^{-7})$ и проследите численно за катастрофическим изменением решения. [**Уилкинсон, 1963**]

Чувствительность к исходным данным. **Корни полинома** (MatLab, Кондрашов)

Пусть `vn=1:n`, где `n>1` - целочисленный параметр, и
`pn=poly(vn')` - полином с корнями `1:n`, которые хорошо отделены друг от друга,
`wn=roots(pn)` - вектор-столбец с вычисленными корнями полинома `pn`.
Проведем сравнение `vn'` и `wn` для различных `n`.

```
>> n=2; vn=1:n; pn=poly(vn'); wn=roots(pn); [vn',wn]
ans =     1     2
          2     1
```

откуда видно, что элементы в `wn` нужно упорядочить.

```
>> n=3; vn=1:n; pn=poly(vn'); wn=roots(pn); R=[vn',sort(wn)]
R =     1.0000    1.0000
        2.0000    2.0000
        3.0000    3.0000
```

Появившиеся после десятичной точки цифры 0 в первом столбце R как бы "наведены" значениями из второго столбца (шероховатость команды `format`).
Погрешность в втором столбце R еще очень мала:

```
>> ((R(:,2)-R(:,1))./R(:,1))'
ans =   1.0e-015 *
                  0.2220   -0.9992    0.5921
```

дает относительную ошибку для корней
Это означает, что в численном результате верны примерно 14 знаков.

Чувствительность к исходным данным. Корни полинома

Для n=10 выполнение "программы" дает

```
>> n=10; vn=1:n; pn=poly(vn'); wn=roots(pn); R=[vn',sort(wn)]; R1=(R(:,2)-R(:,1))./R(:,1); R1'
ans =    1.0e-009 *
  -0.0000   0.0011  -0.0129   0.0711  -0.2270   0.4464  -0.5501   0.4132  -0.1725   0.0306
```

говорит о том, что точность результата постепенно падает.

```
>> me=max(abs(R1))
me =
  5.5015e-010   ,
```

т.е. теперь для всех корней верны только 9 знаков (me – max. error). Но корни еще остаются вещественными, поскольку

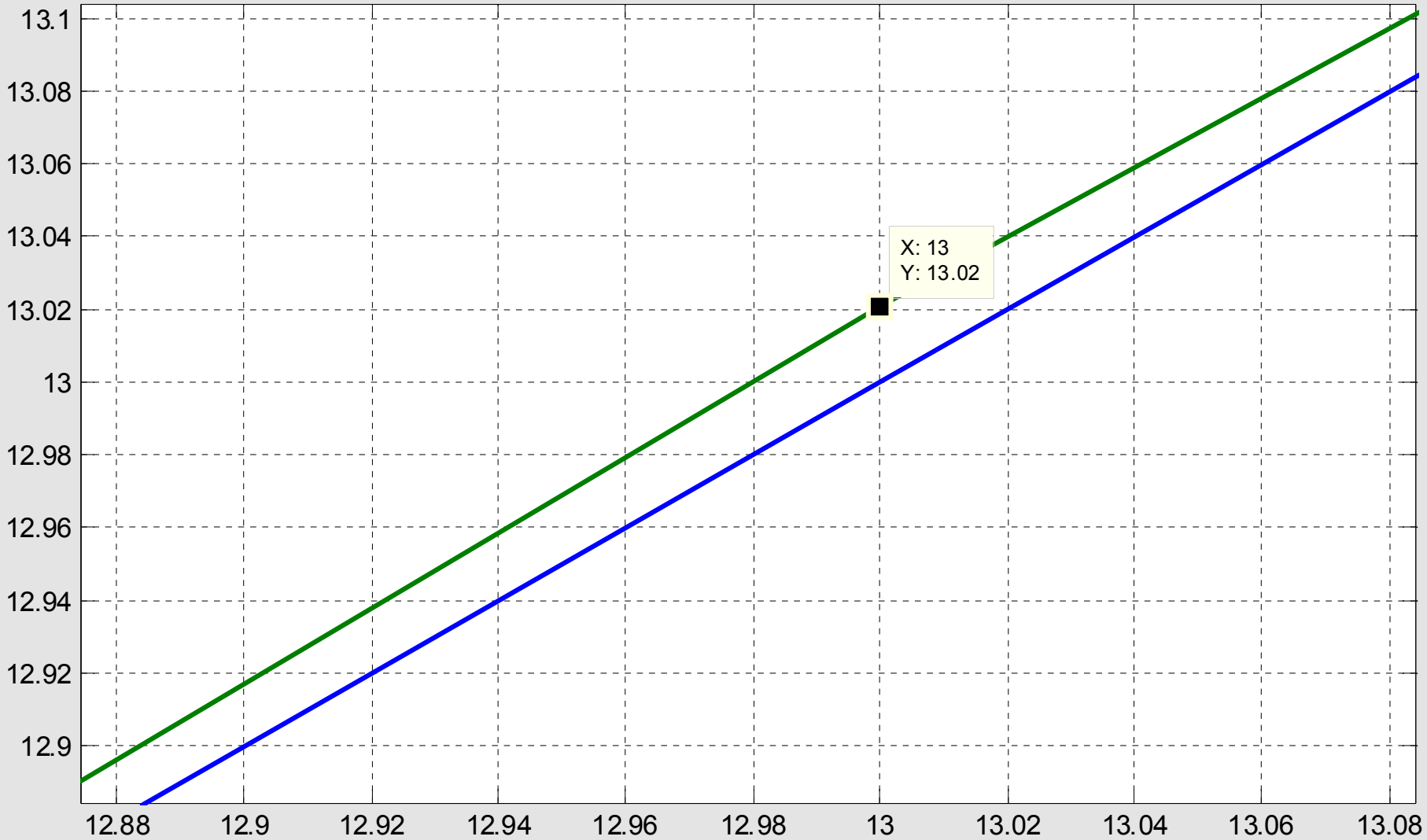
```
>> iwn=sum(abs(imag(wn)))
iwn =
   0
```

Повторим для n=20 и найдем максимальную относительную ошибку me=0.0016, так что теперь для всех корней верны только 2 знака, но результат пока еще остается вещественным, поскольку iwn=0. Сравним точные и вычисленные корни графически:

```
>> plot(R), grid
```

Чувствительность к исходным данным.

Корни полинома



Чувствительность к исходным данным. Корни полинома

Пусть n=20 pn(2)= -210 (это коэффициент при x^{19}). Прибавим к нему 1e-7, т.е. внесем в него малое возмущение примерно в 10-м знаке, и повторим расчет :

```
>> n=20; vn=1:n; pn=poly(vn'); pn(2)=pn(2)+1e-7; wn=roots(pn); R=[vn',sort(wn)], plot(R,'*'), grid
R =  …
```

Несмотря на такое малое возмущение в коэффициенте pn(2), некоторые корни стали комплексными. Это видно, во-первых, из выдачи на экране (их мнимые части достигают по модулю 2.7), во-вторых, из того, что теперь iwn=sum(abs(imag(wn)))=18.6631, и из графика.

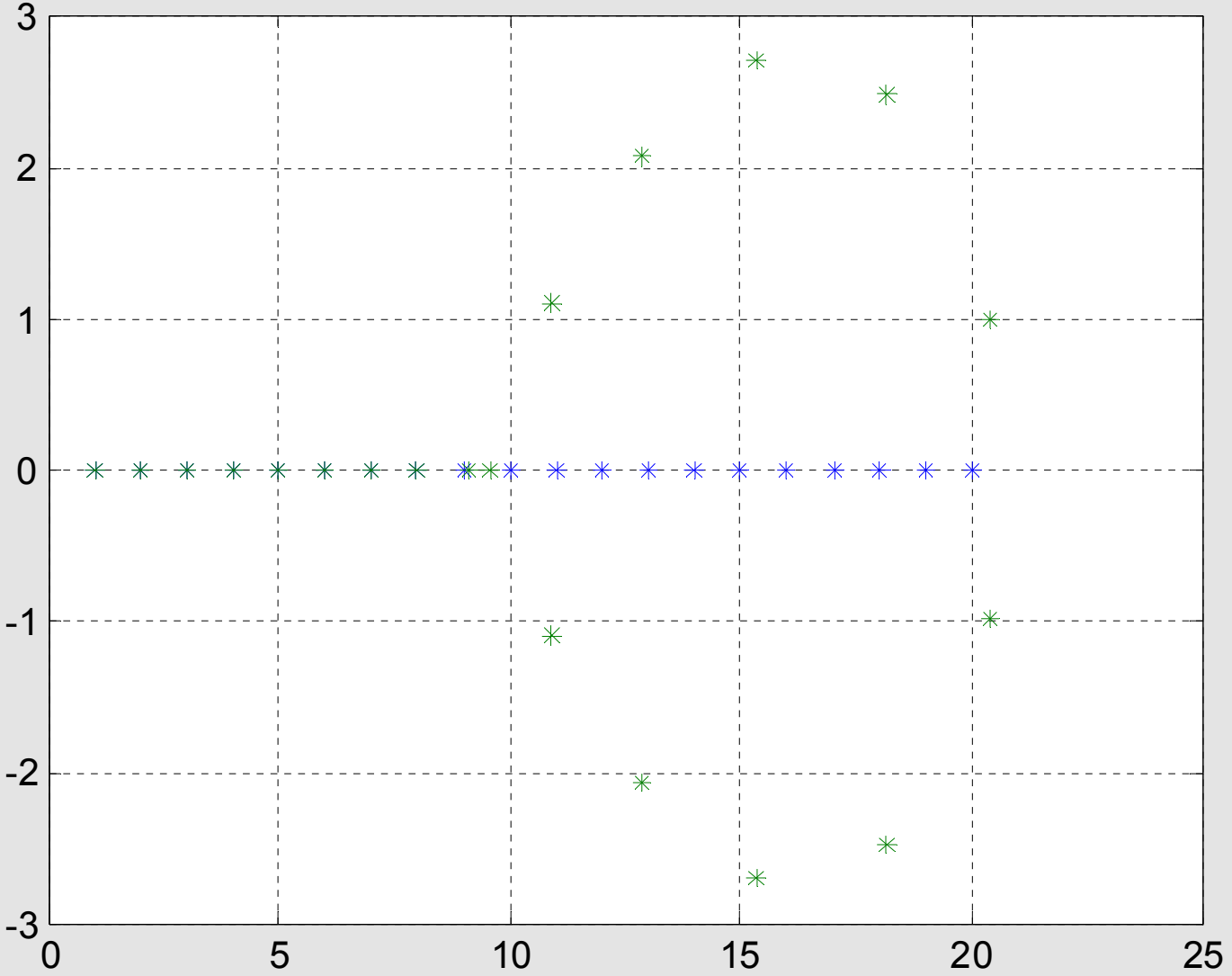
Чувствительность к исходным данным.
Корни полинома

R =

1.0000	1.0000
2.0000	2.0000
3.0000	3.0000
4.0000	4.0000
5.0000	5.0000
6.0000	6.0000
7.0000	7.0001
8.0000	7.9948
9.0000	9.1074
10.0000	9.5767
11.0000	10.9206 - 1.1013i
12.0000	10.9206 + 1.1013i
13.0000	12.8460 - 2.0622i
14.0000	12.8460 + 2.0622i
15.0000	15.3147 - 2.6987i
16.0000	15.3147 + 2.6987i
17.0000	18.1572 - 2.4702i
18.0000	18.1572 + 2.4702i
19.0000	20.4220 - 0.9992i
20.0000	20.4220 + 0.9992i

Чувствительность к исходным данным.

Корни полинома



Чувствительность к исходным данным. Корни полинома

Выясним, почему так сильно изменились результаты при внесении столь малого возмущения. Обозначим через p(x) наш *невозмущенный* полином pn при n=20 и через a его второй коэффициент:

p(x)=prod(x-k),k=1:20, или $p(x)=x^{20}+ax^{19}+ \dots +20!$, a=-210.

Тогда для корней x=1:20 по теореме о производной неявной функции будем иметь

$$\partial p / \partial x * dx / da + \partial p / \partial a = 0$$, откуда $$dx / da = -\partial p / \partial a \,/\, \partial p / \partial x$$

У нас $\partial p / \partial a = x^{19}$, а полином $\partial p / \partial x$ находится как polyder(pn). Поэтому для вычисления **dxda** на множестве **vn** наших корней сначала выполним строку

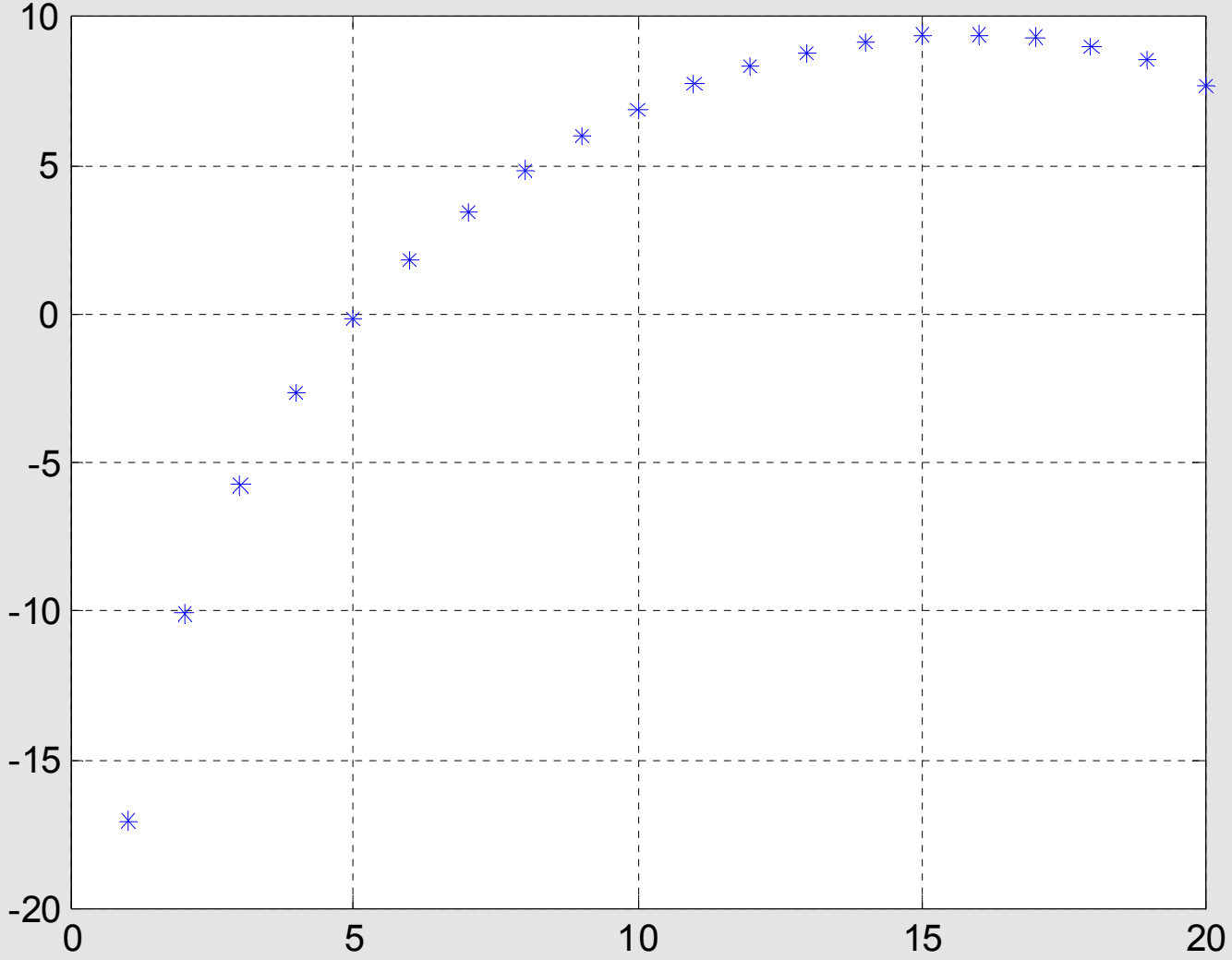
```
>> n=20; vn=1:n; pn=poly(vn');
```

а затем строку (на графике значения функции определены только для **x=1:20**)

```
>> dpn=polyder(pn); dxda=-(vn.^19)./polyval(dpn,vn); plot(log10(abs(dxda)),'*'), grid
```

и увидим, что уже при x=8 $dx/da=10^5$ и будет еще больше с ростом x. Поэтому внесение в коэффициент a возмущения 10^{-7} должно в обязательном порядке заметно сказаться на значениях некоторых корней, каким бы методом они ни находились.

Чувствительность к исходным данным.
Корни полинома



Пример 3. Эквивалентные формулы – разные результаты ?!

Пусть $\varepsilon_M = 10^{-2}$ и требуется решить уравнение: $x^2 + 9{,}9x - 1 = 0$

Округляем до двух значащих цифр.
Формулы для решения уравнения

$$x^2 + px + q = 0 \qquad x_{1,2} = (-p \pm \sqrt{D})/2 \qquad D^2 = p^2 - 4q$$

Результат: $\sqrt{D} \approx 10,\ x_1 \simeq 0{,}05;\ x_2 \approx -9{,}95 \approx -10$

Верный ответ $x_1 \approx 0{,}1$ -- ошибка получена при вычитании близких чисел.

Эквивалентная формула $x_{1,2} = -2q/(p \pm \sqrt{D})$

получаем $x_1 \approx 2/(9{,}9 + 10) \approx 0{,}1$

Правда, по этой формуле уже x_2 будет вычислено с двукратной погрешностью.

Пример 4. Эквивалентные формулы – разные результаты ?!

Оценка дисперсии случайной величины по измерениям :

$$\hat{\sigma}_2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \overline{x^2}, \text{ где } \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (1)$$

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - x)^2 \quad (2)$$

Пусть $x_1 = 12345.1$, $x_2 = 12345.2$, $x_3 = 12345.3$.

Находим среднее

И считаем СКО (1)

```
>> x= single([12345.1 12345.2 12345.3])
x =   1.0e+004 *     1.2345100   1.2345200   1.2345300
>> x1=mean(x)
x1 =  1.2345200e+004
>> y1=sum((x).^2)/3, y2=x1^2
y1 = 152403952
y2 = 152403968
>> y1-y2
ans = -16
>> y22=sum((x-x1).^2)/3
y22 =  0.0066799
>> y = std(x,1)^2
y =  0.0066799
```

Чушь!

Верный ответ

Истинно так!

а СКО (2) ?!

Правило 2.

Правило 2.

При выборе формулы и порядка вычислений избегать вычитания близких чисел и деления на малые величины.

Пример 5. Решения СЛАУ

Система 1:	Решение:
$\begin{cases} x+5y=17, \\ 1{,}5x+7{,}501y=25{,}5. \end{cases}$	$\begin{cases} x=17. \\ y=0. \end{cases}$
Система 2:	**Решение:**
$\begin{cases} x+5y=17, \\ 1{,}5x+7{,}499y=\mathbf{25{,}503}. \end{cases}$	$\begin{cases} x=32, \\ y=-3. \end{cases}$

Столь большие отличия в ответах возникли из-за того, что матрицы коэффициентов систем 1 и 2 плохо обусловлены: определители $\Delta_i = \det|a_i|$ малы. Действительно, $\Delta_1 = 0.001$, $\Delta_2 = -0.001$.

Правило 3.

Избегать плохо обусловленных матриц.
Использовать специальные методы.

Переполнение, исчезновение порядка,…

Пример 6. Числа, представимые в ЭВМ, лежат в диапазоне

$$minD \le |x| \le maxD.$$

При выходе результата за **minD** => **underflow** (исчезновение порядка), при выходе за **maxD** => **overflow** (переполнение).

- При переполнении обычно говорят, что плохи исходные данные, а при исчезновении порядка полагают результат равным нулю.
- Не следует торопиться. Пусть $10^{-78} \le |x| \le 10^{76}$ и вычисляется величина $x = ab/(cd)$ при $a = 10^{-30}$, $b = 10^{-60}$, $c = 10^{-40}$, $d = 10^{-50}$.
- Если $x = a \cdot b/c/d$ => $10^{-30-60}/c/d$ => **underflow**;
- Если $x = 1/c/d \cdot a \cdot b$ => $10^{+40+50} \cdot a \cdot b$ => **overflow**.
- Если $x = a/c \cdot b/d$ => $10^{10} \cdot 10^{-60}/10^{-50}$ => **правильный ответ** $x = 1$.
- Этот же ответ можно получить, если отмасштабировать переменные, например, умножив на 10^{40}.

Правило 4.

Правило 4.

При переполнении или исчезновении порядка следует попытаться изменить последовательность действий, ввести масштабные множители и т. д.

При исчезновении порядка не всегда следует обнулять результат.

Суммы, произведения…

Пример 7. Пусть $\varepsilon_м = 10^{-2}$ и требуется найти $S = 100 + \underbrace{0.1 + \ldots + 0.1}_{\text{2000 слагаемых}}$

- Если вести суммирование слева направо, то с учетом округления до двух значащих цифр => $S = 100$
- Если вычислять справа налево, то после тысячи слагаемых => 100, и дальнейшее {+0.1+0.1+…} **ничего не изменит**. Результат $S = 200$
- Правильный результат ***S = 300*** !? Как получить?!
- Сложим 1000 чисел по 0.1, затем еще 1000 чисел по 0.1, а **потом** сложим промежуточные суммы.

Правило 5.

Правило 5.

При сложении следует располагать слагаемые в порядке возрастания абсолютных величин, стараясь, чтобы при каждом сложении порядки величин различались мало. При необходимости цикл суммирования разбивается на несколько более коротких.

Аналогичное правило действует при перемножении большого числа сомножителей.

Последовательные приближения

Пример 8. При расчетах методами последовательных приближений часто ведут вычисления до тех пор, пока поправка (разность между текущими и последующими приближениями) не станет меньше заданного порога. При этом, как правило, не обеспечивается заданная погрешность результата.

$$S = \sum_{k=1}^{\infty} \frac{1}{k^2}$$

Найти S с точностью до 10^{-3}.

Если вести вычисления до тех пор, пока общий член ряда $1/k^2$ не станет меньше 10^{-3}, т. е. до $k_{обрыв} = 32$, и $S = 1.610$.

Правильно => $S = \pi^2/6 = 1{,}650\ldots$.

Если же приближенную сумму ряда

$$\sum_{k=1}^{\infty} \frac{1}{k}$$

то погрешность останется бесконечной, как бы мала ни становилась величина $1/k$, поскольку ряд расходится!

Правило 6.

Правило 6.

Нужно помнить, что остановка итерационного процесса $x_1, x_2, \ldots$. по косвенному критерию (например, по

$$|x_n - x_{n-1}| < \varepsilon$$ или $$|F(x_n)| < \varepsilon$$

в задаче решения уравнения $F(x) = 0$,

по критерию $$\left\| \frac{\partial f}{\partial x}(x_n) \right\| < \varepsilon$$ в задаче

оптимизации $f(x)$ и т. д.)

не гарантирует достижения заданной погрешности

$$|x_n - \lim_{n \to \infty} x_n| < \varepsilon$$

Пример 9. **Неустойчивость алгоритмов**

- Проверить *неустойчивость алгоритмов (погрешность действия)* на примере вычисления интеграла $$E_n = \int_0^1 x^n e^{x-1} dx$$ (n = 1, 2, 3,...)

при помощи рекуррентной формулы

$$E_n = \int_0^1 x^n e^{x-1} dx = x^n e^{x-1}\Big|_0^1 - n\int_0^1 x^{n-1} e^{x-1} dx = 1 - n \cdot E_{n-1}$$, (n = 2, 3,...),

- E_0 вычислить аналитически и построить таблицу значений E_n при n = 1, 2,..., 24. Оценить возникающую ошибку.

Пример 9. **Неустойчивость алгоритмов**

Оценим (вычислим ТОЧНО!) начальное значение E_0

$$E_n = \int_0^1 x^n e^{x-1} dx = x^n e^{x-1}\Big|_0^1 - n\int_0^1 x^{n-1} e^{x-1} dx = 1 - n \cdot E_{n-1}$$

$$E_1 = \int_0^1 x\ e^{x-1} dx = x\ e^{x-1}\Big|_0^1 - \int_0^1 e^{x-1} dx = 1 - E_0 \approx 0.367879441171442$$

$$E_0 = \int_0^1 e^{x-1} dx = e^{-1}\left(1-e\right) = 1 - e^{-1} \approx$$

$$1 - 0.367879441171442 \approx 0.632120558828558$$

Пример 9. Неустойчивость алгоритмов

n	$E_n = 1 - nE_{n-1}$
1	0.3679
2	0.2642
3	0.2073
4	0.1709
5	0.1455
6	0.1268
7	0.1124
8	0.1009
9	0.0916
10	0.0839
11	0.0774
12	0.0718

n	$E_n = 1 - nE_{n-1}$
13	0.0669
14	0.0627
15	0.0590
16	0.0555
17	0.0572
18	-0.0295
19	1.5596
20	-30.1924
21	635.0403
22	-1.3970e+004
23	3.2131e+005
24	-7.7114e+006

Уже для n=18 получен <u>бессмысленный</u> результат !
Причина в том, что начальная ошибка округления быстро накапливается: при вычислении n=18 она умножается на 2, затем на 3, 4,..., 18

Пример 9. **Неустойчивость алгоритмов**

- Проверить *неустойчивость алгоритмов (погрешность действия)* на примере вычисления интеграла $$E_n = \int_0^1 x^n e^{x-1} dx$$ (n = 1, 2, 3,...)

при помощи рекуррентной формулы

$$E_n = \int_0^1 x^n e^{x-1} dx = x^n e^{x-1}\Big|_0^1 - n\int_0^1 x^{n-1} e^{x-1} dx = 1 - n \cdot E_{n-1}$$, (n = 2, 3,...),

- E_0 вычислить аналитически и построить таблицу значений E_n при n = 1, 2,..., 24. Оценить возникающую ошибку.
- Повторить вычисления, изменив алгоритм на ***устойчивый*** $\Rightarrow$

  $E_{n-1} = (1 - E_n) / n.$

  Аналитически и численно оценить ошибку при вычислениях E_n для n = 24, 23,..., 1 при выборе начального значения E_{25} = 0 (показать, что начальная ошибка $\delta E_{24} < 1/25$ и далее ***уменьшается*** !).

Пример 9. Неустойчивость алгоритмов

n	$E_n = 1 - nE_{n-1}$	n	$E_n = 1 - nE_{n-1}$	$E_{n-1} = (1 - E_n)/n$
1	0.3679	13	0.0669	0.0669
2	0.2642	14	0.0627	0.0627
3	0.2073	15	0.0590	0.0590
4	0.1709	16	0.0555	0.0557
5	0.1455	17	0.0572	0.0528
6	0.1268	18	-0.0295	0.0501
7	0.1124	19	1.5596	0.0477
8	0.1009	20	-30.1924	0.0455
9	0.0916	21	635.0403	0.0436
10	0.0839	22	-1.3970e+004	0.0417
11	0.0774	23	3.2131e+005	0.0400
12	0.0718	24	-7.7114e+006	0.0400

Пример 9. Неустойчивость алгоритмов

После первого шага начальная ошибка уменьшится в 24 раза, после второго — еще в 23 раза, для n=20 мы получим все шесть значащих цифр верных. Формула (2), в отличие от (1), определяет устойчивый вычислительный процесс: погрешность результата каждого шага меньше погрешности исходных данных.

n	$E_{n-1} = (1 - E_n)/n$	$E_{n-1} = (1 - E_n)/n$
1	0.3679	0.0669
2	0.2642	0.0627
3	0.2073	0.0590
4	0.1709	0.0557
5	0.1455	0.0528
6	0.1268	0.0501
7	0.1124	0.0477
8	0.1009	0.0455
9	0.0916	0.0436
10	0.0839	0.0417
11	0.0774	0.0400
12	0.0718	0.0400

Правило 6.

Правило 6.

Пользуйтесь только устойчивыми численными алгоритмами!

Литература

- Д. Поттер, Вычислительные методы в физике.
- Н. Н. Калиткин, Численные методы.
- Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков, Численные методы.

The End