Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Московский институт электронной техники»

Лабораторная работа «Распространение ошибок в вычислительных процедурах»

Работу выполнил
Учащийся группы ПИН-33
Карпеченков Михаил Владимирович
Под руководством
Ярошевича Владимира Александровича

Москва 2023

① Начнём со второго пункта. Рассмотрим многочлен Уилкинсона

$$P(x) = (x-1)(x-2)...(x-20) = x^{20} - 210x^{19} + 20615x^{18} + \dots$$

Очевидно, корнями его являются $x_1 = 1$, $x_2 = 2, \dots, x_{20} = 20$. Выполните в MATLAB команду `p=poly(1:20)`, которая позволяет получить коэффициенты полинома, корнями которого является аргумент функции. Наберите `roots(p)` и убедитесь, что корни полинома найдены верно. Измените значение, например, второго коэффициента на малую величину 10^{-7} (составляет $\approx 5 \cdot 10^{-8}\%$ от числа) и снова выполните эту команду - половина корней стала комплексными числами! Исходная задача оказалась неустойчивой к входным данным (их малое изменение ведет к сильному изменению решения), в результате чего получился абсурдный результат.

```
format long
p=poly(1:20);
vpa(p,20)
roots(p)
p(2) = p(2) + 10^(-7)
roots(p)
```

ans =

[1.0, -210.0, 20615.0, -1256850.0, 53327946.0, -1672280820.0, 40171771630.0, -756111184500.0, 11310276995381.0, -135585182899530.0, 1307535010540395.0, -10142299865511450.0, 63030812099294896.0, -311333643161390656.0, 1206647803780373248.0, -3599979517947607040.0, 8037811822645052416.0, -12870931245150988288.0, 13803759753640704000.0, -8752948036761600000.0, 2432902008176640000.0]

ans =

19.999874055724192

19.001295393676987

17.993671562737585

17.018541647321989

15.959717574548915

15.059326234074415

13.930186454760916

13.062663652011070

11.958873995343460

11.022464271003383

9.991190949230132

9.002712743189727

7.999394310958664

7.000096952230211

5.999989523351082

5.000000705531480

3.999999973862455

3.000000000444877

1.999999999998383

0.999999999999949

ans =

20.421952157241002 + 0.999204441836253i

20.421952157241002 - 0.999204441836253i

18.157179851897560 + 2.470210137828049i

18.157179851897560 - 2.470210137828049i

15.314665874595322 + 2.698627782369154i

15.314665874595322 - 2.698627782369154i

12.845907764519843 + 2.062027828451370i

12.845907764519843 - 2.062027828451370i

10.920734092000639 + 1.100999586499169i

10.920734092000639 - 1.100999586499169i

9.576245848460163 + 0.000000000000000i

9.108084834750644 + 0.000000000000000i

7.994608540228951 + 0.000000000000000i

7.000179586708066 + 0.000000000000000i

6.000002117174314 + 0.000000000000000i

4.999999458560572 + 0.000000000000000i

4.000000035082699 + 0.000000000000000i

2.999999998492783 + 0.000000000000000i

2.000000000033340 + 0.000000000000000i

0.999999999999706 + 0.000000000000000i

① Рассмотрим представление числа в компьютере. Чаще всего в MATLAB производятся операции над числами с двойной точностью. Число формата double — 64-разрядное число (8 байт), в котором 1 бит — знаковый, 52 отводятся под мантиссу и 11 под порядок числа. ($D = \pm(1+m)\cdot 2^n$, $m = 0, m_1 m_2 \ldots m_k$, $m_1 \neq 0$ — мантисса числа, n — порядок). Так как в порядке тоже есть знаковый бит, то множество его значений лежит от $-2^{10}+1 = -1023$ до $2^{10} = 1024$. В командном окне MATLAB наберите `2^1023`. Получилось число с десятичным порядком +308 (его легко оценить аналитически: $2^{1024} \approx 2^{1000} = 1024^{100} \approx (10^3)^{100} = 10^{300}$). Теперь выполните `2^1024` — получили `Inf` т.е. машинную бесконечность. Команда `realmax` как раз и выдаёт максимальное число, которое можно представить в MATLAB, команда `realmin` — минимальное (по абсолютной

величине).

① Теперь посмотрим на мантиссу: $2^{52} \approx 4,5 \cdot 10^{15}$, таким образом, мантисса содержит 15-16 десятичных знаков; все, что вылезет за эти пределы, будет отброшено. Установите формат отображения с плавающей точкой: `format long e`, выполните `sqrt(2)` (квадратный корень из 2). Посчитайте число выданных цифр.

```
>> 2^1023
ans =
  8.9885e+307
>> 2^1024
ans =
   Inf


realmin
realmax
format long e
sqrt(2)
```

ans =

  2.225073858507201e-308

ans =

  1.797693134862316e+308

ans =

   1.414213562373095e+00

Получили ответ в виде десятичного числа с 16-ю цифрами.

① Казалось бы, такая точность представления способна удовлетворить любые нужды исследователя. Однако, необходимо помнить, что, например, перед тем как два числа будут сложены они должны быть приведены к единому порядку, а то, что при сложении выйдет за мантиссу будет отброшено! Поэтому, если сложить `10^8+10^-7` в мантиссу уложатся 15 десятичных знаков и будет получен верный результат, а если выполнить `10^8+10^-8`, то малое число выйдет за разрядную сетку, и получим те же `10^8` (проверьте).

```
10^8+10^(-7)
10^8+10^(-8)
```

ans =

1.000000000000001e+08

ans =

1.000000000000000e+08

① Получается парадоксальная ситуация: давайте прибавим к 1 малое число `10^-16`, но последовательно `10^17` раз[1](такие вычисления характерны для задач ЧМ — вычисление рядов, разнообразные итерационные, разностные задачи...). Легко найти ответ 11, однако, машинная арифметика даёт 1 — ошибка 1000%! Если бы мы начали с конца, т.е. сначала нашли сумму малых, а затем прибавили к единице, то получили бы верный результат. Таким образом, машинное сложение, в общем случае, не коммутативно.

1-ый способ (слагаемые расположены по невозрастанию):

```
clear; clc;

format long e;
a=1;
b=1/10^(16);
for i=1:1:10^(6)
    i
    a=a+1/10^(16);
end
a
```

```
Command Window
i =
       999990
i =
       999991
i =
       999992
i =
       999993
i =
       999994
i =
       999995
i =
       999996
i =
       999997
i =
       999998
i =
       999999
i =
     1000000
a =
     1
fx >>
```

2-ой способ (слагаемые расположены по неубыванию)

```
format long e;
a=1;
b=1/10^(16);
for i=1:1:10^(6)
    i
    b=b+1/10^(16);
end
a=b+a
```

```
Command Window
i =
       999990
i =
       999991
i =
       999992
i =
       999993
i =
       999994
i =
       999995
i =
       999996
i =
       999997
i =
       999998
i =
       999999
i =
     1000000
a =
     1.000000000100000e+00
fx >>
```

Проверила на 10^6 итераций (этого достаточно, чтобы увидеть разницу), действительно, сложение в Matlab некоммутативно. (в первом способе получили ответ 1, а правильный ответ представлен во втором способе)

① Знание этих фактов позволяет протестировать компьютер на погрешность вычислении. Найдем значение выражения $\frac{1+\varepsilon-1}{\varepsilon}$ при $\varepsilon = 2^{-n}$, $n = 0, 1, 2, \ldots$. С точки зрения аналитической математики значение этого

выражения постоянно и равно единице для любых n. Сделайте предположения, при каком значении n это выражение перестанет отличаться от единицы, чему будет равно; напишите программу, реализующую этот алгоритм с пошаговой выдачей номера шага, значения ε и результата выражения.

Из-за размеров мантиссы скорее всего будет ошибка в округлении при n>=53

```
n=0;
ans=1;
eps=0;
while(ans==1)
    eps=2^(n)
    ans=(1+eps-1)/eps;
    fprintf('Номер шага: %d\nОтвет: %16.16f\n', n, ans);
    n=n+1;
end
```

eps =

1

Номер шага: 0

Ответ: 1.0000000000000000

eps =

2

Номер шага: 1

Ответ: 1.0000000000000000

eps =

4

Номер шага: 2

Ответ: 1.0000000000000000

eps =

8

Номер шага: 3

Ответ: 1.0000000000000000

eps =

16

Номер шага: 4

Ответ: 1.0000000000000000

eps =

32

Номер шага: 5

Ответ: 1.0000000000000000

eps =

64

Номер шага: 6

Ответ: 1.0000000000000000

eps =

128

Номер шага: 7

Ответ: 1.0000000000000000

eps =

256

Номер шага: 8

Ответ: 1.0000000000000000

eps =

512

Номер шага: 9

Ответ: 1.0000000000000000

eps =

1024

Номер шага: 10

Ответ: 1.0000000000000000

eps =

2048

Номер шага: 11

Ответ: 1.0000000000000000

eps =

4096

Номер шага: 12

Ответ: 1.0000000000000000

eps =

8192

Номер шага: 13

Ответ: 1.0000000000000000

eps =

16384

Номер шага: 14

Ответ: 1.0000000000000000

eps =

32768

Номер шага: 15

Ответ: 1.0000000000000000

eps =

65536

Номер шага: 16

Ответ: 1.0000000000000000

eps =

131072

Номер шага: 17

Ответ: 1.0000000000000000

eps =

262144

Номер шага: 18

Ответ: 1.0000000000000000

eps =

524288

Номер шага: 19

Ответ: 1.0000000000000000

eps =

1048576

Номер шага: 20

Ответ: 1.0000000000000000

eps =

2097152

Номер шага: 21

Ответ: 1.0000000000000000

eps =

4194304

Номер шага: 22

Ответ: 1.0000000000000000

eps =

8388608

Номер шага: 23

Ответ: 1.0000000000000000

eps =

16777216

Номер шага: 24

Ответ: 1.0000000000000000

eps =

33554432

Номер шага: 25

Ответ: 1.0000000000000000

eps =

67108864

Номер шага: 26

Ответ: 1.0000000000000000

eps =

134217728

Номер шага: 27

Ответ: 1.0000000000000000

eps =

268435456

Номер шага: 28

Ответ: 1.0000000000000000

eps =

536870912

Номер шага: 29

Ответ: 1.0000000000000000

eps =

    1.073741824000000e+09

Номер шага: 30

Ответ: 1.0000000000000000

eps =

    2.147483648000000e+09

Номер шага: 31

Ответ: 1.0000000000000000

eps =

    4.294967296000000e+09

Номер шага: 32

Ответ: 1.0000000000000000

eps =

    8.589934592000000e+09

Номер шага: 33

Ответ: 1.0000000000000000

eps =

    1.717986918400000e+10

Номер шага: 34

Ответ: 1.0000000000000000

eps =

    3.435973836800000e+10

Номер шага: 35

Ответ: 1.0000000000000000

eps =

    6.871947673600000e+10

Номер шага: 36

Ответ: 1.0000000000000000

eps =

    1.374389534720000e+11

Номер шага: 37

Ответ: 1.0000000000000000

eps =

2.748779069440000e+11

Номер шага: 38

Ответ: 1.0000000000000000

eps =

5.497558138880000e+11

Номер шага: 39

Ответ: 1.0000000000000000

eps =

1.099511627776000e+12

Номер шага: 40

Ответ: 1.0000000000000000

eps =

2.199023255552000e+12

Номер шага: 41

Ответ: 1.0000000000000000

eps =

4.398046511104000e+12

Номер шага: 42

Ответ: 1.0000000000000000

eps =

8.796093022208000e+12

Номер шага: 43

Ответ: 1.0000000000000000

eps =

1.759218604441600e+13

Номер шага: 44

Ответ: 1.0000000000000000

eps =

3.518437208883200e+13

Номер шага: 45

Ответ: 1.0000000000000000

eps =

7.036874417766400e+13

Номер шага: 46

Ответ: 1.0000000000000000

eps =

    1.407374883553280e+14

Номер шага: 47

Ответ: 1.0000000000000000

eps =

    2.814749767106560e+14

Номер шага: 48

Ответ: 1.0000000000000000

eps =

    5.629499534213120e+14

Номер шага: 49

Ответ: 1.0000000000000000

eps =

    1.125899906842624e+15

Номер шага: 50

Ответ: 1.0000000000000000

eps =

    2.251799813685248e+15

Номер шага: 51

Ответ: 1.0000000000000000

eps =

    4.503599627370496e+15

Номер шага: 52

Ответ: 1.0000000000000000

eps =

    9.007199254740992e+15

Номер шага: 53

Ответ: 0.9999999999999999

② Рассмотрим погрешности численных методов. Построим алгоритм вычисления интеграла $I_n = \int_0^1 x^n e^{x-1}\,dx$, $n = 1, 2, 3, \ldots$ Интегрируя по частям, находим:

$$I_1 = \int\limits_0^1 xe^{x-1}\,dx = xe^{x-1}|_0^1 - \int\limits_0^1 e^{x-1}\,dx = \tfrac{1}{e}$$

$$I_2 = \int\limits_0^1 x^2e^{x-1}\,dx = x^2e^{x-1}|_0^1 - 2\int\limits_0^1 xe^{x-1}\,dx = 1 - 2I_1$$

$$\ldots$$

$$I_n = \int\limits_0^1 x^ne^{x-1}\,dx = x^ne^{x-1}|_0^1 - n\int\limits_0^1 x^{n-1}e^{x-1}\,dx = 1 - nI_{n-1}$$

Вычислите значения интегралов, до $n = 30$. Заметьте, что подинтегральная функция на всем отрезке интегрирования неотрицательна, следовательно, и значение интеграла — положительное число. Более того, подынтегральная функция на данном интервале ограничена функцией $y = 1$, т.е. значение интеграла не может превышать единицы. Посмотрите, как это согласуется с полученными результатами. Попытайтесь на основе полученных сведений о погрешностях объяснить данный феномен.

```
function [answ] = CaLcMyInTeGrAl(n)
    syms x;
    func=x^n*exp(x-1);
    if(n==1)
        answ=1/exp(1);
        %answ=int(func,x,0,1)
    else
        if(n>1)
            answ=1-n*CaLcMyInTeGrAl(n-1)
        end
        if(n<1)
            fprintf('Ошибка входных данных!');
        end
    end
    plot(n,answ,'or');
end
```

```
CaLcMyInTeGrAl(30)
```

answ =

  2.642411176571154e-01

answ =

  2.072766470286537e-01

answ =

```
   1.708934118853853e-01

answ =

   1.455329405730734e-01

answ =

   1.268023565615595e-01

answ =

   1.123835040690837e-01

answ =

   1.009319674473304e-01

answ =

   9.161229297402684e-02

answ =

   8.387707025973157e-02

answ =

   7.735222714295276e-02

answ =

   7.177327428456692e-02

answ =

   6.694743430062999e-02

answ =

   6.273591979118009e-02

answ =

   5.896120313229858e-02

answ =

   5.662074988322274e-02

answ =

   3.744725198521337e-02

answ =

   3.259494642661593e-01

answ =

  -5.193039821057027e+00

answ =

   1.048607964211405e+02

answ =
```

-2.201076724843952e+03

answ =

4.842468794656693e+04

answ =

-1.113766822771040e+06

answ =

2.673040474650495e+07

answ =

-6.682601176626236e+08

answ =

1.737476306022821e+10

answ =

-4.691186026251618e+11

answ =

1.313532087350553e+13

answ =

-3.809243053316594e+14

answ =

1.142772915994978e+16

ans =

1.142772915994978e+16

Можно заметить, что находить по проделанному выше алгоритму нельзя, так как получим неверный ответ. Оно и видно, ведь если записать общее выражение для I_n и попытаться понять, что останется после раскрытия скобок, то можно заметить, что останется $\sum_1^{n-1}(n-1)!$ (примерно) + $\frac{n!}{e} * (-1)^{n-1}$ $(n \geq 2)$. При $n \to \infty$ разность по модулю между вторым и первым слагаемым будет значительно увеличиваться → получим неверный ответ.

③ Напишите функцию вычисления значения синуса в виде конечной суммы ряда[2] ($\sin x = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+1)!}$). По признаку Лейбница погрешность вычисления сходящегося знакопеременного ряда не превышает по абсолютной величине первого из отброшенных членов. Вычисления членов ряда проводите до вычисления члена по модулю не превышающего

10^{-17}. Вычислите значение синуса в точках $0, \frac{\pi}{3}, \frac{\pi}{2}, \pi, 2\pi$, убедитесь, что полученные значения с высокой степенью точности совпадают с действительным значением синуса в этих точках. Далее проведите вычисления в точках $12\pi, 13\pi, 14\pi$, выводя результаты по шагам (посчитанный член и частичную сумму ряда). Полученные результаты объясняются погрешностями округления, в реальных программах значение аргумента приводится к отрезку $[0; \frac{\pi}{2}]$.

```
function [result] = Sinx(a)
    syms x;
    n=0;
    answ=x;
    sum=0;
    b=true;
    t=0;
    result=0;
    while(b)
        sum=sum+answ;
        t=double(subs(answ,x,a))
        result = result + vpa(t,16);
        coeff=(-x^2)/(2*n+2)/(2*n+3);
        answ=answ*coeff;
        if(abs(t)>=10^(-17))
            n=n+1;
        else
            b=false;
        end
    end
    result = vpa(subs(sum,x,a),16)
end


Sinx(0);
Sinx(pi/3);
Sinx(pi/2);
Sinx(pi);
Sinx(2*pi);
```

result =

0.0

result =

0.8660254037844386

result =

1.0

result =

-2.908631335271852e-21

result =

-7.06306858677339e-21

```
Sinx(12*pi);
Sinx(13*pi);
Sinx(14*pi);
```

Далее идут промежуточные результаты значений n-ого члена (на n-ом шаеге алгоритма), хотел вставить и сумму ряда в отчёт, но он бы вышел слишком громоздким(алгоритм по 150+ итераций совершал) (На примере промежуточных результатов всё равно видно, что из-за нарушения условия сходимости данного ряда и ошибок с погрешностями результат получается неправильным)

result =

37.699111843077518861551678845613

result =

-8892.1085720832710264941706293133

result =

625670.30979867630925786290029354

result =

-20847061.246034316929906489516699

result =

403007670.48768497237291943821768

result =

-5073284217.7559502052393852492823

result =

44817947648.575737722006708500718

result =

-292832375736.83014606705579149928

result =

1471419157170.5006644798192085007

result =

-5860144924434.0637886451807914993

result =

18948873518996.701836354819208501

result =

-50733238055514.673163645180791499

result =

114323131962004.6393363548192085

result =

-219839140028301.4856636451807915

result =

365036621012524.1393363548192085

result =

-528768651543711.3606636451807915

result =

674163773693274.3893363548192085

result =

-762504521096486.3606636451807915

result =

770397935677845.1393363548192085

result =

-699640040078005.8606636451807915

result =

574294003605857.1393363548192085

result =

-428222278991337.4856636451807915

result =

291373293000300.0143363548192085

result =

-181663618148619.2981636451807915

result =

104174372472998.4518363548192085

result =

-55135249563951.610663645180791499

result =

27018053896866.279961354819208501

result =

-12294460304390.345038645180791499

result =

5209252320208.6139457298192085007

result =

-2060379344999.1350777076807914993

result =

762508078367.40789104231920850072

result =

-264617530386.07526325455579149928

result =

86289819066.32732463606920850071 8

result =

-26491186758.25995563736829149928 2

result =

7670570114.75174403914538037571 77

result =

-2098338485.15292667618665087428 23

result =

543175578.17471194991198193821768

result =

-133253364.31068217030438769068857

result =

31024898.46444455513086926731923 9

result =

-6864756.03110453507575381007333 93

result =

1445375.09700716545161616597036 19

result =

-289939.29805782016079116072794132

result =

55476.49873156673096877131322688 8

result =

-10136.02286968401345598902841739 9

result =

1770.26249836598766909520784682 96

result =

-295.85313186051130993217518263128

result =

47.34592425016974606552687763604 4

result =

-7.2747256611533884521886386313001

result =

1.0616280515552067017242996268449

result =

-0.15954468747242031714640344154232

result =

0.012292819943622022767482952700265

result =

-0.010952889517928132782641732832 6

result =

-0.0079274922905677813346583441373935

result =

-0.0083065933508559751659928630613747

result =

-0.0082608248188305562071628477356297

result =

-0.0082661521973674738395444810045917

result =

-0.0082655539520224825863224036833279

result =

-0.0082656188063370622501505229556351

result =

-0.0082656120149691194906402091009338

result =

-0.0082656127023390490567851914614421

result =

-0.0082656126350590231193718982940492

result =

-0.0082656126414311357792420349747467

result =

-0.0082656126408468652446677654295093

result =

-0.0082656126408987574294998108130822

result =

-0.0082656126408942909585480622878909

result =

-0.0082656126408946637037856565899131

result =

-0.0082656126408946335286821368702521

result =

-0.0082656126408946358993594264348462

result =

40.840704496667312100014301102895

result =

-11312.624273273117724698552278716

result =

935545.43258490615815307520947055

result =

-36667415.072997813883121949204592

result =

834448587.98924064093560363673291

result =

-12374545839.904687886896427613267

result =

128856966161.22760581427544738673

result =

-992899970785.39470863884955261327

result =

5885957481828.9900569861504473867

result =

-27662816501520.091974263849552613

result =

105570799030311.36115073615044739

result =

-333616474132131.13884926384955261

result =

887297167291193.11115073615044739

result =

-2013613025924395.3888492638495526

result =

3945267867808632.6111507361504474

result =

-6742037195574119.3888492638495526

result =

10138673933308116.611150736150447

result =

-13522169767399183.388849263849553

result =

16106520355435652.611150736150447

result =

-17240013466081591.388849263849553

result =

16675102013943268.611150736150447

result =

-14647800886313163.388849263849553

result =

11738788807603944.611150736150447

result =

-8618222801835071.3888492638495526

result =

5818318474580472.6111507361504474

result =

-3624669279450513.3888492638495526

result =

2090336359492990.6111507361504474

result =

-1119232250700974.3888492638495526

result =

557911279765784.86115073615044739

result =

-259567853973111.38884926384955261

result =

112979978237006.98615073615044739

result =

-46107593043879.107599263849552613

result =

17678994824064.189275736150447387

result =

-6381086051439.0997867638495526133

result =

2172052937649.6052913611504473867

result =

-698434104093.40642738884955261327

result =

212499417660.24652671271294738673

result =

-61266986073.545587545099552613267

result =

16763135497.589666361150447386733

result =

-4358475486.7735920006659588632671

result =

1078263503.8534688895317950429829

result =

-254131479.79917812890082214451709

result =

57127025.200182611283961791029783

result =

-12261870.085606386723812561997561

result =

2515724.3889327404770293416889627

result =

-493858.38729666762694333172717496

result =

92849.554372113126445669165464204

result =

-16736.920753190205433298417715 27

result =

2892.181648680413926826274605553 9

result =

-482.4441348574316370157090704028 4

result =

74.8579870242190546325376500303 92

result =

-13.620920195585684637792523864371

result =

-0.10630881784508279921069959662178

result =

-2.0937778427645505519207334886936

result =

-1.8121753042852608750669661562618

result =

-1.8506439902295278268819510346724

result =

-1.8455741142541135734449496462808

result =

-1.8462191460062529667585730407467

result =

-1.8461398732985356819218141681896

result =

-1.8461492896177765593355806199999

result =

-1.8461482079322776689351080618446

result =

-1.846148328164954285948745685354

result =

-1.8461483152266527842912412343414

result =

-1.8461483165752723348431232176409

result =

-1.8461483164390412480391491852621

result =

-1.8461483164523840802074804455194

result =

-1.8461483164511164023599939195606

result =

-1.8461483164512332868225588221931

result =

-1.8461483164512228231592033439767

result =

-1.8461483164512237330228685458299

result =

-1.8461483164512236561424703986663

result =

-1.8461483164512236624575334632244

result =

43.982297150257105338477029239294

result =

-14136.221571306860900269731448258

result =

1357403.2030116865179624688778657

result =

-61813170.655603749141922937616275

result =

1635406835.7300757342717733514462

result =

-28211633607.448455912456742273554

result =

341899792140.16329945863700772645

result =

-3067430007215.1174622601129922736

result =

21179453569720.425506489887007726

result =

-115967333498321.59011851011299227

result =

515705448681284.28488148988700773

result =

-1899185168562681.7151185101129923

result =

5886593085710130.2848814898870077

result =

-15568022383746061.715118510112992

result =

35543700748225410.284881489887008

result =

-70771017062345565.715118510112992

result =

123982491018659906.28488148988701

result =

-192605286943391485.71511851011299

result =

267170119088328642.28488148988701

result =

-332970952904336061.71511851011299

result =

374918299059691202.28488148988701

result =

-383315935617457981.71511851011299

result =

357472195779884354.28488148988701

result =

-305345573496312893.71511851011299

result =

239800190901069378.28488148988701

result =

-173750050261361533.71511851011299

result =

116521773733074434.28488148988701

result =

-72540226952148893.715118510112992

result =

42036703559179858.284881489887008

result =

-22733161904593461.715118510112992

result =

11500056142097158.284881489887008

result =

-5453909710802381.7151185101129923

result =

2429857466012141.2848814898870077

result =

-1018965236879916.2151185101129923

result =

402933544556551.78488148988700773

result =

-150503352897998.84011851011299227

result =

53186113390654.847381489887007726

result =

-17809481704093.918243510112992274

result =

5658887026220.1169127398870077264

result =

-1708560068909.8215638226129922736

result =

490807601133.07052602113700772645

result =

-134309973335.09268198667549227355

result =

35053329210.730926411762007726446

result =

-8734904423.7385636395075235235538

result =

2080445232.2061594897649374139462

result =

-474093427.43632326780342196105377

result =

103467121.37155784906364346863373

result =

-21645717.412801650748482691034244

result =

4344788.1673727469287004602108735

result =

-837353.41365001576476429580108937

result =

155176.75680598064801956535918895

result =

-27575.242732864647299768313334549

result =

4798.67924840529387069875045005

result =

-722.87685826950845153294833120191

result =

184.45686518060134944573181006825

result =

40.707079562000226085382362601814

result =

62.678926061068770119499074884226

result =

59.43687626321794808458210160432

result =

59.898971665138828584284195006768

result =

59.835312858184391195714216611287

result =

59.843793870532323713159253781777

result =

59.842700572490754327278836639445

result =

59.842837019081457812104110642246

result =

59.842820524388360969993097000081

result =

59.84282245680325524745096337089

result =

59.842822237299706986903702044228

result =

59.842822261486138090203850696186

result =

59.842822258899777706273500167597

result =

59.842822259168303111506232091568

result =

59.842822259141223195638611519223

result =

59.842822259143876921025558161652

result =

59.842822259143624115004055893169

result =

59.842822259143647536397868075798

result =

59.84282225914364542534863110443

result =

59.842822259143645610533797287328

result =

59.842822259143645594717901088196

result =

59.842822259143645596033473034872

Вывод:

В этой лабораторной работе я изучил и проверил на практике распространенные ошибки при различных вычислениях в программе Matlab.