Answers to Homework 1: Computer Arithmetics

---

1. The number $\pi = 3.14159265358979\ldots$ If we use the approximation $\pi \approx 3.14$, what is the absolute error? Express your answer using chopping to a decimal normalized floating-point representation with 5 significant digits.

   Answer: The absolute error is $|\pi - \pi^*| = 0.00159265358979\ldots \approx 0.15926 \times 10^{-2}$.

2. The hexadecimal (base 16) counting system uses letters A, B, C, D, E, and F in addition to decimal digits (from 0 to 9) to represent digits for 10, 11, 12, 13, 14, and 15, correspondingly. Find the hexadecimal representation of the decimal number 2989.

   Answer: $(2989)_{10} = (BAD)_{16}$.

   Solution: Dividing 2989 by 16, we find that $2989 = 186 \cdot 16 + 13$. The remainder gives us the last digit: D, since $(13)_{10} = D_{16}$. Repeating the division, we get $186 = 11 \cdot 16 + 10$, therefore $2989 = (11 \cdot 16 + 10) \cdot 16 + 13 = 11 \cdot 16^2 + 10 \cdot 16^1 + 13 \cdot 16^0 = (BAD)_{16}$.

3. In the IEEE double-precision floating-point standard, 64 bits (binary digits) are used to represent a real number: 1 bit for the sign, 11 bits for the exponent, and 52 bits for the mantissa. A double-precision normalized non-zero number x can be written in this standard as $x = \pm(1.d_1 d_2 \cdots d_{52})_2 \times 2^{n-1023}$, with $1 \le n \le 2046$, and $0 \le d_k \le 1$ for $k = 1, 2, \ldots, 52$.

   (a) What is the smallest positive number in this system?

   (b) What is the smallest negative number in this system?

   (c) How many real numbers are in this system?

   Note: You may express your answers with formulas. Note that the correct answers are different from the ones in the textbook.

   Answer:

   (a) $2^{-1022} \approx 2.2 \times 10^{-308}$.

   (b) $-(2 - 2^{-52})\, 2^{1023} \approx -1.8 \times 10^{308}$.

   (c) $2 \cdot 2046 \cdot 2^{52} + 1 \approx 1.8 \times 10^{19}$.

   Solution:

   (a) The smallest positive number has the sign $+$, the smallest possible mantissa with digits $d_1 = d_2 = \cdots = d_{52} = 0$, and the smallest possible exponent $n = 1$. Therefore, $x = (1.0)_2 \times 2^{-1022} = 2^{-1022}$.

   (b) The smallest negative number has the sign $-$, the largest possible mantissa with digits $d_1 = d_2 = \cdots = d_{52} = 1$, and the largest possible exponent $n = 2046$. Therefore, $x = -(1.11\ldots 1)_2 \times 2^{2046-1023} = -(2 - 2^{-52}) \times 2^{1023}$.

(c) The sign can take two possible values. Hence, the first factor of 2. The exponent can take 2046 possible values, which gives us the second factor. Next, we need to multiply by 2^{52} possible combinations for the mantissa, and add one to account for number zero: $N = 2 \cdot 2046 \cdot 2^{52} + 1$.

Subtlety: We have not talked about it in class, but the IEEE standard actually allows for numbers that are smaller than the minimum normalized floating-point number. These so-called "subnormal" or "unnormalized" numbers appear when the stored exponent n is equal to zero and the mantissa is different from zero. In the double-precision standard, an unnormalized number x can be written as $x = \pm(0.d_1 d_2 \cdots d_{52})_2 \times 2^{-1022}$, where $0 \le d_k \le 1$ for $k = 1, 2, \ldots, 52$ and $d_1 \cdot d_2 \cdot \cdots d_{52} \neq 0$. The smallest positive unnormalized number is $2^{-52} \cdot 2^{-1022} = 2^{-1074} \approx 5 \times 10^{-324}$. The total number of unnormalized numbers is 2 (for the sign) multiplied by $(2^{52} - 1)$ (we need to subtract the zero combination from all possible digit combinations): $2 \cdot (2^{52} - 1) = 2^{53} - 2 \approx 9 \times 10^{15}$.

4. (Programming) In this assignment, you will evaluate the accuracy of Stirling's famous approximation

$$n! \approx n^n e^{-n} \sqrt{2\pi n} \,. \tag{1}$$

Write a program to output a table of the form

n	$n!$	Stirling's approximation	Absolute error	Relative error

for $n = 1, 2, \ldots, 10$.

Judging from the table, does the accuracy increase or decrease with increasing n?

Hint: If your computer system does not have a predefined value for π, use $\pi =$ `acos(-1.0)` or $\pi =$ `4.0*atan(1.0)`.

Answer:

n	$n!$	Stirling's approximation	Absolute error	Relative error
1	1	0.922137	0.077863	0.077863
2	2	1.919004	0.080996	0.040498
3	6	5.836210	0.163790	0.027298
4	24	23.506175	0.493825	0.020576
5	120	118.019168	1.980832	0.016507
6	720	710.078185	9.921815	0.013780
7	5040	4980.395832	59.604168	0.011826
8	40320	39902.395453	417.604547	0.010357
9	362880	359536.872842	3343.127158	0.009213
10	3628800	3598695.618741	30104.381259	0.008296

Your numbers may be slightly different depending on the computer system and the precision used.

According to the table, the absolute error increases, and the relative error decreases with the increase in n.

Solution:

(a) C program (using double precision)

```c
#include <stdio.h> /* for output */
#include <math.h>  /* for mathematical functions */

int main (void)
{
  int n;
  long nf;
  double stirling, abs_error, rel_error, pi;

  pi = acos(-1.0); /* the number pi */
  nf = 1; /* starting value for n! */

  for (n=1; n <= 10; n++) {
    /* compute n! recursively */
    nf *= n;
    /* compute Stirling's approximation */
    stirling = sqrt(2.0*pi*n)*exp(-n)*pow(n,n);
    /* compute absolute error */
    abs_error = fabs(nf-stirling);
    /* compute relative error */
    rel_error = abs_error/nf;
    /* print out the table */
    printf("n=%d n!=%ld stirling=%f abs_error=%g rel_error=%g\n",
   n, nf, stirling, abs_error, rel_error);
  }

  return 0;
}
```

(b) Fortran-90 program (using single precision)

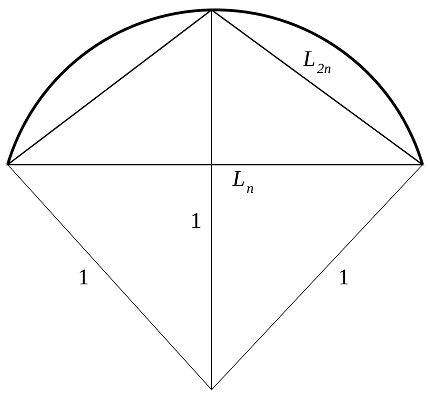
```fortran
program StirlingTable
  integer :: n, nf
  real    :: stirling, abs_error, rel_error, pi

  pi = acos(-1.0) ! the number pi
  nf = 1          ! starting value for n!
  do n=1, 10
     ! compute n! recursively
     nf = nf*n
     ! compute Stirling's approximation
     stirling = sqrt(2.0*pi*n)*exp(-real(n))*(real(n)**n)
     ! compute absolute error
     abs_error = abs(nf-stirling)
     ! compute relative error
     rel_error = abs_error/nf
     ! print out the table
     print *, "n=", n, "n!=", nf, "stirling=", stirling, &
```

```
            "abs_error=", abs_error, "rel_error=", rel_error
  end do
end program StirlingTable
```

5. (Programming) In this assignment, you will compute the number π using an iterative method. An equilateral regular polygon, inscribed in a circle of radius 1, has the perimeter $n L_n$, where n is the number of sides of the polygon, and L_n is the length of one side. This can serve as an approximation for the circle perimeter 2π. Therefore, $\pi \approx \frac{n L_n}{2}$. From trigonometry (see the figure), a polynom with twice as many sides, inscribed in the same circle, has the side length

$$L_{2n} = \sqrt{2 - \sqrt{4 - L_n^2}}\,. \tag{2}$$



   (a) Write a program to iteratively compute approximations for π using equation (2) and starting from $n = 6$ and $L_6 = 1$ (regular hexagon). You need to do the computations using double precision floating-point numbers. Output a table of the form

n	L_n	Absolute error in approximating π

   for $n = 6, 6 \times 2, 6 \times 4, \ldots, 6 \times 2^{20}$.

   (b) Use the formula $b - \sqrt{b^2 - a} = \frac{a}{b + \sqrt{b^2 - a}}$ to derive a different form of equation (2).

   (c) Modify your program using the new equation and repeat the computation to produce a new table.

   (d) Compare the tables and explain the source of the difference.

   Answer:

(a)

n	L_n	Absolute error
6	1.000000e+00	1.415927e-01
12	5.176381e-01	3.576411e-02
24	2.610524e-01	8.964040e-03
48	1.308063e-01	2.242451e-03
96	6.543817e-02	5.607027e-04
192	3.272346e-02	1.401813e-04
384	1.636228e-02	3.504568e-05
768	8.181208e-03	8.761441e-06
1536	4.090613e-03	2.190353e-06
3072	2.045307e-03	5.475467e-07
6144	1.022654e-03	1.370016e-07
12288	5.113269e-04	3.494900e-08
24576	2.556635e-04	8.268577e-09
49152	1.278317e-04	8.268577e-09
98304	6.391587e-05	8.268577e-09
196608	3.195793e-05	8.268577e-09
393216	1.597896e-05	3.497781e-07
786432	7.989482e-06	3.497781e-07
1572864	3.994734e-06	5.813935e-06
3145728	1.997367e-06	5.813935e-06
6291456	9.987114e-07	8.161143e-05

Your numbers may be different depending on the computer system and the precision used.

(b) The modified equation is

$$L_{2n} = \frac{L_n}{\sqrt{2+\sqrt{4-L_n^2}}}\,. \tag{3}$$

(c)

n	L_n	Absolute error
6	1.000000e+00	1.415927e-01
12	5.176381e-01	3.576411e-02
24	2.610524e-01	8.964040e-03
48	1.308063e-01	2.242451e-03
96	6.543817e-02	5.607027e-04
192	3.272346e-02	1.401813e-04
384	1.636228e-02	3.504568e-05
768	8.181208e-03	8.761441e-06
1536	4.090613e-03	2.190362e-06
3072	2.045307e-03	5.475905e-07
6144	1.022654e-03	1.368976e-07
12288	5.113269e-04	3.422441e-08
24576	2.556635e-04	8.556102e-09
49152	1.278317e-04	2.139026e-09
98304	6.391587e-05	5.347562e-10
196608	3.195793e-05	1.336895e-10
393216	1.597897e-05	3.342304e-11
786432	7.989483e-06	8.355983e-12
1572864	3.994742e-06	2.088996e-12
3145728	1.997371e-06	5.222489e-13
6291456	9.986854e-07	1.305622e-13

(d) The first method loses accuracy at large n, because it suffers from the loss of precision when subtracting two positive numbers of similar magnitude (when L_n is small, $\sqrt{4-L_n^2}$ is close to 2). In the second method, the accuracy steadily increases with n.

Solution:

(a) C program

```
#include <stdio.h> /* for output */
#include <math.h>  /* for mathematical functions */

int main (void)
{
  const int METHOD=2;
  int k;
  long n;
  double ln, ln2, pi, approx, error;

  pi = acos(-1.0); /* number pi */
  ln2 = 1.0; /* first approximation for the side length squared */
  n = 6;
  for (k=0; k < 21; k++) {
    ln = sqrt(ln2);
    /* compute the approximation for pi */
    approx = 0.5*n*ln;
```

```
    /* compute the error */
    error = fabs(pi-approx);
    /* print out the result */
    printf("n=%ld ln=%e error=%e\n", n, ln, error);

    n *= 2; /* double the number of sides */
    if (METHOD==1) { /* use the first method */
      ln2 = 2.0 - sqrt(4.0-ln2);
    } else {         /* use the second method */
      ln2 = ln2/(2.0 + sqrt(4.0-ln2));
    }
  }

  return 0;
}
```

(b) Fortran-90 program

```
program PiApprox
  integer, parameter :: method=2
  integer            :: n, k
  double precision   :: ln, ln2, pi, approx, error

  pi = acos(-1.0) ! number pi
  ln2 = 1.0 ! first approximation for the side length squared
  n = 6
  do k=1,21
     ln=sqrt(ln2)
     ! compute the approximation for pi
     approx = 0.5*n*ln
     ! compute the error
     error = abs(pi-approx)
     ! print out the result
     print *, "n=", n, "ln=", ln, "error=", error

     n = n*2 ! double the number of sides
     if (method==1) then ! use the first method
        ln2 = 2.0 - sqrt(4.0-ln2)
     else           ! use the second method
        ln2 = ln2/(2.0 + sqrt(4.0-ln2))
     end if
  end do
end program PiApprox
```