Answers to Homework 2: Nonlinear Equations: Bisection and Regula Falsi

---

1. Prove that the equation

$$f(x) = x + a\cos x = 0 \tag{1}$$

has at least one solution (for every real a).

Hint: Use the intermediate value theorem. You don't have to prove that $f(x)$ is continuous.

Solution: First, we note that $|a\cos x| \le |a|$ for all x. Therefore,

$$f(2|a|) = 2|a| + a\cos(2|a|) \ge 2|a| - |a\cos(2|a|)| \ge 2|a| - |a| = |a| \ge 0$$

and

$$f(-2|a|) = -2|a| + a\cos(2|a|) \le -2|a| + |a\cos(2|a|)| \le -2|a| + |a| = -|a| \le 0$$

If $a = 0$ then the equation $f(x) = 0$ has the trivial solution $x = 0$. If $a \neq 0$ then

$$f(2|a|) \ge |a| > 0 \text{and} f(-2|a|) \le -|a| < 0\,.$$

According to the intermediate theorem, there must be at least one real number c, $-2|a| \le c \ge 2|a|$ such that $f(c)$ takes the intermediate value $f(c) = 0$.

2. If the bisection method is applied on the initial interval from $a = 14$ to $b = 16$, how many iterations will be required to guarantee that the root is located to the maximum accuracy of the IEEE double-precision standard?

Hint: A number in the IEEE double-precision standard can have the maximum of 53 significant digits in the binary format.

Answer: 50 iterations or more.

Solution: Writing the numbers in the binary format, we see that

$$a = 14 = 8 + 4 + 2 = 1\times 2^3 + 1\times 2^2 + 1^1 = (1110)_2$$

$$b = 16 = 1\times 2^4 = (10000)_2$$

The root is between these two numbers. Therefore, it has the form

$$c = (111x.xxxx\ldots)_2$$

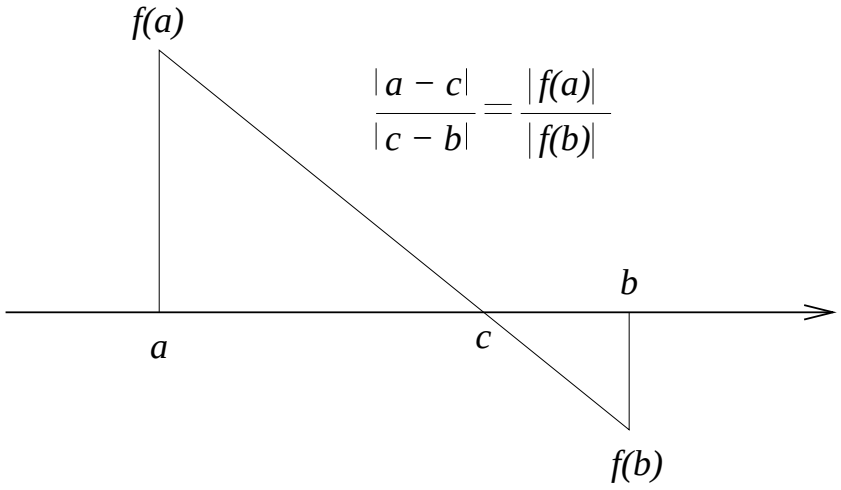
where x stands for unknown digits. Thus, we already know three significant binary digits in the root. Each iteration of bisection will bring us at least one additional binary digit of precision. $53 - 3 = 50$ iterations will guarantee that the root is located to the maximum precision. For safety, we can add one additional iteration.

3. The bisection method can be implemented with the following schematic algorithm

```
BISECTION(f(x), a, b, xtol, ftol)
 1  fa ← f(a)
 2  fb ← f(b)
 3  ASSERT : SIGN(fa) ≠ SIGN(fb)
 4  for n ← 1, 2, ...
 5  do
 6     d ← b − a
 7     c ← a + d/2
 8     fc ← f(c)
 9     if |d| < xtol and |fc| < ftol
10       then return c
11     if SIGN(fc) = SIGN(fa)
12       then
13            a ← c
14            fa ← fc
15       else
16            b ← c
```

In this assignment, you will design an analogous algorithm for a different method. In this method, the input and the general strategy are the same as those in bisection, but instead of dividing the interval in half, we will divide it in proportion $|f(a)|/|f(b)|$ (see the figure.) This method is known as the method of chords or the method of false position or *regula falsi*.

Modify the algorithm above to transform it to an algorithm for *regula falsi*.



Answer:

```
REGULA-FALSI(f(x),a,b,xtol, ftol)
 1  fa ← f(a)
 2  fb ← f(b)
 3  ASSERT : SIGN(fa) ≠ SIGN(fb)
 4  for n ← 1,2,...
 5  do
 6     d ← b − a
 7     c ← a + d · fa/(fa − fb)
 8     fc ← f(c)
 9     if |d| < xtol and |fc| < ftol
10       then return c
11     if SIGN(fc) = SIGN(fa)
12       then
13            a ← c
14            fa ← fc
15       else
16            b ← c
17            fb ← fc
```

Solution:

We only need to modify line 7 of the algorithm to adopt a different formula. According the figure, the next iteration is found by intersecting a straight line with the x axis. The straight line equation is

$$L(x) = f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a},$$

and its root is given by

$$c = \frac{b\,f(a) - a\,f(b)}{f(a) - f(b)} = a + f(a)\,\frac{b-a}{f(a) - f(b)}.$$

This is the new equation that we need to use instead of $c = (a+b)/2 = a + (b-a)/2$ in bisection.

4. (Programming) Implement the bisection algorithm in the programming language of your choice and use it to solve the following three problems. In each case, perform 20 iterations and output your results in a table

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$

(a) To test your program, start by computing something familiar. How about π? Solve the equation

$$\sin x = 0 \tag{2}$$

using the initial interval $a = 2$ and $b = 4$.

Determine (experimentally) the number of iterations required to compute the number π with six significant decimal digits.

(b) The equation

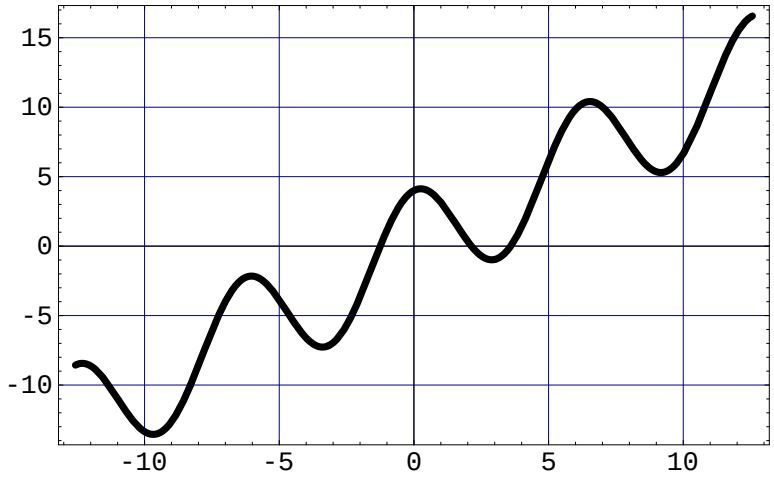
$$x + e^x = 0 \tag{3}$$

has the root around $x \approx -0.6$. Find a better approximation using the initial interval $a = -1$ and $b = 0$.

(c) The equation

$$x + 4\cos x = 0 \tag{4}$$

has three roots. Locate all of them, selecting an appropriate initial interval for each. The function $f(x) = x + 4\cos x$ is plotted below.



Answer:

(a)

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	2.000000	4.000000	3.000000	1.411200e-01
1	3.000000	4.000000	3.500000	3.507832e-01
2	3.000000	3.500000	3.250000	1.081951e-01
3	3.000000	3.250000	3.125000	1.659189e-02
4	3.125000	3.250000	3.187500	4.589122e-02
5	3.125000	3.187500	3.156250	1.465682e-02
6	3.125000	3.156250	3.140625	9.676534e-04
7	3.140625	3.156250	3.148438	6.844793e-03
8	3.140625	3.148438	3.144531	2.938592e-03
9	3.140625	3.144531	3.142578	9.854713e-04
10	3.140625	3.142578	3.141602	8.908910e-06
11	3.140625	3.141602	3.141113	4.793723e-04
12	3.141113	3.141602	3.141357	2.352317e-04
13	3.141357	3.141602	3.141479	1.131614e-04
14	3.141479	3.141602	3.141541	5.212625e-05
15	3.141541	3.141602	3.141571	2.160867e-05
16	3.141571	3.141602	3.141586	6.349879e-06
17	3.141586	3.141602	3.141594	1.279516e-06
18	3.141586	3.141594	3.141590	2.535182e-06
19	3.141590	3.141594	3.141592	6.278330e-07

Experimentally, 18 iterations are required to compute π with 6 significant digits.

(b)

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	-1.000000	-0.000000	-0.500000	1.065307e-01
1	-1.000000	-0.500000	-0.750000	2.776334e-01
2	-0.750000	-0.500000	-0.625000	8.973857e-02
3	-0.625000	-0.500000	-0.562500	7.282825e-03
4	-0.625000	-0.562500	-0.593750	4.149755e-02
5	-0.593750	-0.562500	-0.578125	1.717584e-02
6	-0.578125	-0.562500	-0.570312	4.963760e-03
7	-0.570312	-0.562500	-0.566406	1.155202e-03
8	-0.570312	-0.566406	-0.568359	1.905360e-03
9	-0.568359	-0.566406	-0.567383	3.753492e-04
10	-0.567383	-0.566406	-0.566895	3.898588e-04
11	-0.567383	-0.566895	-0.567139	7.237912e-06
12	-0.567383	-0.567139	-0.567261	1.840599e-04
13	-0.567261	-0.567139	-0.567200	8.841203e-05
14	-0.567200	-0.567139	-0.567169	4.058732e-05
15	-0.567169	-0.567139	-0.567154	1.667477e-05
16	-0.567154	-0.567139	-0.567146	4.718446e-06
17	-0.567146	-0.567139	-0.567142	1.259729e-06
18	-0.567146	-0.567142	-0.567144	1.729360e-06
19	-0.567144	-0.567142	-0.567143	2.348157e-07

A better approximation for the root is $c \approx -0.567143$.

(c) We can see from the graph that the first root is located between $a = -2$ and $b = -1$, the second root is between $a = 2$ and $b = 3$, and the third root is between $a = 3$ and $b = 4$.

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	-2.000000	-1.000000	-1.500000	1.217051e+00
1	-1.500000	-1.000000	-1.250000	1.128945e-02
2	-1.500000	-1.250000	-1.375000	5.968092e-01
3	-1.375000	-1.250000	-1.312500	2.907649e-01
4	-1.312500	-1.250000	-1.281250	1.391801e-01
5	-1.281250	-1.250000	-1.265625	6.379864e-02
6	-1.265625	-1.250000	-1.257812	2.621701e-02
7	-1.257812	-1.250000	-1.253906	7.454270e-03
8	-1.253906	-1.250000	-1.251953	1.919981e-03
9	-1.253906	-1.251953	-1.252930	2.766548e-03
10	-1.252930	-1.251953	-1.252441	4.231343e-04
11	-1.252441	-1.251953	-1.252197	7.484608e-04
12	-1.252441	-1.252197	-1.252319	1.626726e-04
13	-1.252441	-1.252319	-1.252380	1.302285e-04
14	-1.252380	-1.252319	-1.252350	1.622260e-05
15	-1.252380	-1.252350	-1.252365	5.700283e-05
16	-1.252365	-1.252350	-1.252357	2.039008e-05
17	-1.252357	-1.252350	-1.252354	2.083730e-06
18	-1.252354	-1.252350	-1.252352	7.069436e-06
19	-1.252354	-1.252352	-1.252353	2.492854e-06

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	2.000000	3.000000	2.500000	7.045745e-01
1	2.000000	2.500000	2.250000	2.626945e-01
2	2.000000	2.250000	2.125000	1.993466e-02
3	2.125000	2.250000	2.187500	1.258968e-01
4	2.125000	2.187500	2.156250	5.406023e-02
5	2.125000	2.156250	2.140625	1.732620e-02
6	2.125000	2.140625	2.132812	1.239179e-03
7	2.132812	2.140625	2.136719	8.059875e-03
8	2.132812	2.136719	2.134766	3.414426e-03
9	2.132812	2.134766	2.133789	1.088641e-03
10	2.132812	2.133789	2.133301	7.501460e-05
11	2.133301	2.133789	2.133545	5.068770e-04
12	2.133301	2.133545	2.133423	2.159471e-04
13	2.133301	2.133423	2.133362	7.047023e-05
14	2.133301	2.133362	2.133331	2.271190e-06
15	2.133331	2.133362	2.133347	3.409977e-05
16	2.133331	2.133347	2.133339	1.591435e-05
17	2.133331	2.133339	2.133335	6.821596e-06
18	2.133331	2.133335	2.133333	2.275207e-06
19	2.133331	2.133333	2.133332	2.009649e-09

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	3.000000	4.000000	3.500000	2.458267e-01
1	3.500000	4.000000	3.750000	4.677626e-01
2	3.500000	3.750000	3.625000	8.333466e-02
3	3.500000	3.625000	3.562500	8.837434e-02
4	3.562500	3.625000	3.593750	4.276545e-03
5	3.593750	3.625000	3.609375	3.909324e-02
6	3.593750	3.609375	3.601562	1.729897e-02
7	3.593750	3.601562	3.597656	6.483811e-03
8	3.593750	3.597656	3.595703	1.096777e-03
9	3.593750	3.595703	3.594727	1.591599e-03
10	3.594727	3.595703	3.595215	2.478398e-04
11	3.595215	3.595703	3.595459	4.243614e-04
12	3.595215	3.595459	3.595337	8.823404e-05
13	3.595215	3.595337	3.595276	7.980955e-05
14	3.595276	3.595337	3.595306	4.210568e-06
15	3.595276	3.595306	3.595291	3.779991e-05
16	3.595291	3.595306	3.595299	1.679478e-05
17	3.595299	3.595306	3.595303	6.292131e-06
18	3.595303	3.595306	3.595304	1.040788e-06
19	3.595304	3.595306	3.595305	1.584888e-06

Solution:

(a) C program (double precision)

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

#define SIGN(a) ((a) >= 0.) /* sign function as a macro */

/* function: bisect
   ----------------
   Implements the bisection algorithm.
   func       - a pointer to a function
   a, b       - initial interval
   xtol, ftol - tolerance in position and value
   nmax       - maximum number of iterations
*/
double bisect(double (*func)(double), double a, double b,
      double xtol, double ftol, int nmax)
{
  int n;
  double fa, fb, fc, c, d;

  fa = func(a);
  fb = func(b);
  /* assert that signs are different */
  assert (SIGN(fa) != SIGN(fb));
  for (n=0; n < nmax; n++) {
    d = b-a;
    c = a+d*0.5;
    fc = func(c);

    /* print out the table */
    printf("n=%d a=%f b=%f c=%f |f(c)|=%e\n",n,a,b,c,fabs(fc));

    /* return if the root is located to the tolerance */
    if (fabs(d) <= xtol && fabs(fc) <= ftol) return c;

    if (SIGN(fc) == SIGN(fa)) {
      a = c;
      fa = fc;
    } else {
      b = c;
    }
  }

  fprintf(stderr,"Warning: Exact root is not found after %d iterations\n",
  nmax);
  return c;
}

/* function from assignment 4. (b) */
static double function_b (double x)
{
  return (x + exp(x));
}

/* function from assignment 4. (c) */
static double function_c (double x)
{
  return (x + 4.*cos(x));
}
```

```
int main (void)
{
  int nmax=20; /* maximum number of iterations */
  double xtol=0., ftol=0., a, b, c;

  /* assignment 4. (a) */
  a=2.;  b=4.;  c = bisect(&sin, a,b, xtol, ftol, nmax);

  /* assignment 4. (b) */
  a=-1.; b=0.;  c = bisect(&function_b, a,b, xtol, ftol, nmax);

  /* assignment 4. (c) */
  a=-2.; b=-1.; c = bisect(&function_c, a,b, xtol, ftol, nmax);
  a=2.;  b=3.;  c = bisect(&function_c, a,b, xtol, ftol, nmax);
  a=3.;  b=4.;  c = bisect(&function_c, a,b, xtol, ftol, nmax);

  return 0;
}
```

(b) Fortran-90 program (single precision)

```
! sign function (different from Fortran intrinsic)
integer function sign(x)
  real, intent (in) :: x
  if (x >= 0.) then
     sign = 1
  else
     sign = 0
  end if
end function sign

! function: bisect
! ----------------
! Implements the bisection algorithm.
function bisect(func, a, b, xtol, ftol, nmax) result (c)
  integer, intent (in)     :: nmax        ! maximum number of iterations
  real,    intent (in)     :: xtol, ftol  ! tolerance in position and value
  real,    intent (in out) :: a, b        ! interval
  interface
     real function func (x)               ! function interface
       real, intent (in) :: x
     end function func
  end interface
  integer  :: n, sign
  real     :: fa, fb, fc, c, d
  external :: sign

  fa = func(a)
  fb = func(b)
  ! assert that signs are different
  if (sign(fa) == sign(fb)) then
     write (0,*) "Assertion failed."
     stop
  end if
  do n=0, nmax-1
     d = b-a
     c = a+d*0.5
     fc = func(c)
```

```
      ! print out the table
      print *, "n=", n, "a=", a, "b=", b, "c=", c, "|f(c)|=", abs(fc)

      ! return if the root is located to the tolerance
      if (abs(d) <= xtol .and. abs(fc) <= ftol) return

      if (sign(fc) == sign(fa)) then
         a = c
         fa = fc
      else
         b = c
      end if
   end do

   write (0,*) "Warning: Exact root is not found after", nmax, "iterations"
end function bisect

! function from assignment 4. (a)
real function function_a (x)
   real, intent (in) :: x
   function_a = sin(x)
end function function_a

! function from assignment 4. (b)
real function function_b (x)
   real, intent (in) :: x
   function_b = x + exp(x)
end function function_b

! function from assignment 4. (c)
real function function_c (x)
   real, intent (in) :: x
   function_c = x + 4.*cos(x)
end function function_c

program Main
   integer  :: nmax=20 ! maximum number of iterations
   real     :: xtol=0., ftol=0., a, b, c
   real     :: function_a, function_b, function_c, bisect
   external :: function_a, function_b, function_c, bisect

   ! assignment 4. (a)
   a=2.;  b=4.;  c = bisect(function_a, a,b, xtol, ftol, nmax)

   ! assignment 4. (b)
   a=-1.; b=0.;  c = bisect(function_b, a,b, xtol, ftol, nmax)

   ! assignment 4. (c)
   a=-2.; b=-1.; c = bisect(function_c, a,b, xtol, ftol, nmax)
   a=2.;  b=3.;  c = bisect(function_c, a,b, xtol, ftol, nmax)
   a=3.;  b=4.;  c = bisect(function_c, a,b, xtol, ftol, nmax)
end program Main
```

5. (Programming) Implement your *regulafalsi* algorithm and repeat the computation for the three problems above. Compare the tables. Which of the two methods appears to converge faster?

   Answer:

(a)

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	2.000000	4.000000	3.091528	5.004366e-02
1	3.091528	4.000000	3.147875	6.282262e-03
2	3.091528	3.147875	3.141590	2.295634e-06
3	3.141590	3.147875	3.141593	1.509491e-11
4	3.141590	3.141593	3.141593	1.224647e-16
5	3.141593	3.141593	3.141593	1.224647e-16

Experimentally, 3 iterations are required to compute π with 6 significant digits.

(b)

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	-1.000000	0.000000	-0.612700	7.081395e-02
1	-0.612700	0.000000	-0.572181	7.888273e-03
2	-0.572181	0.000000	-0.567703	8.773920e-04
3	-0.567703	0.000000	-0.567206	9.757273e-05
4	-0.567206	0.000000	-0.567150	1.085062e-05
5	-0.567150	0.000000	-0.567144	1.206646e-06
6	-0.567144	0.000000	-0.567143	1.341853e-07
7	-0.567143	0.000000	-0.567143	1.492210e-08
8	-0.567143	0.000000	-0.567143	1.659415e-09
9	-0.567143	0.000000	-0.567143	1.845355e-10
10	-0.567143	0.000000	-0.567143	2.052125e-11
11	-0.567143	0.000000	-0.567143	2.282174e-12
12	-0.567143	0.000000	-0.567143	2.537970e-13
13	-0.567143	0.000000	-0.567143	2.831069e-14
14	-0.567143	0.000000	-0.567143	3.108624e-15
15	-0.567143	0.000000	-0.567143	3.330669e-16
16	-0.567143	0.000000	-0.567143	1.110223e-16

(c)

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	-2.000000	-1.000000	-1.240625	5.619352e-02
1	-2.000000	-1.240625	-1.252094	1.244282e-03
2	-2.000000	-1.252094	-1.252348	2.607892e-05
3	-2.000000	-1.252348	-1.252353	5.459062e-07
4	-2.000000	-1.252353	-1.252353	1.142707e-08
5	-2.000000	-1.252353	-1.252353	2.391951e-10
6	-2.000000	-1.252353	-1.252353	5.006440e-12
7	-2.000000	-1.252353	-1.252353	1.048051e-13
8	-2.000000	-1.252353	-1.252353	2.442491e-15
9	-2.000000	-1.252353	-1.252353	4.440892e-16

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	2.000000	3.000000	2.258929	2.814555e-01
1	2.000000	2.258929	2.140789	1.771434e-02
2	2.000000	2.140789	2.133726	9.392324e-04
3	2.000000	2.133726	2.133353	4.930855e-05
4	2.000000	2.133353	2.133333	2.587287e-06
5	2.000000	2.133333	2.133332	1.357548e-07
6	2.000000	2.133332	2.133332	7.123033e-09
7	2.000000	2.133332	2.133332	3.737450e-10
8	2.000000	2.133332	2.133332	1.961009e-11
9	2.000000	2.133332	2.133332	1.028511e-12
10	2.000000	2.133332	2.133332	5.373479e-14
11	2.000000	2.133332	2.133332	2.664535e-15
12	2.000000	2.133332	2.133332	8.881784e-16
13	2.000000	2.133332	2.133332	4.440892e-16
14	2.133332	2.133332	2.133332	4.440892e-16

n	a_n	b_n	c_n	$\lvert f(c_n)\rvert$
0	3.000000	4.000000	3.409300	4.482199e-01
1	3.409300	4.000000	3.553692	1.114366e-01
2	3.553692	4.000000	3.586918	2.296437e-02
3	3.586918	4.000000	3.593653	4.541847e-03
4	3.593653	4.000000	3.594981	8.909050e-04
5	3.594981	4.000000	3.595241	1.744722e-04
6	3.595241	4.000000	3.595292	3.415727e-05
7	3.595292	4.000000	3.595302	6.686718e-06
8	3.595302	4.000000	3.595304	1.308994e-06
9	3.595304	4.000000	3.595305	2.562484e-07
10	3.595305	4.000000	3.595305	5.016311e-08
11	3.595305	4.000000	3.595305	9.819917e-09
12	3.595305	4.000000	3.595305	1.922345e-09
13	3.595305	4.000000	3.595305	3.763176e-10
14	3.595305	4.000000	3.595305	7.366774e-11
15	3.595305	4.000000	3.595305	1.442135e-11
16	3.595305	4.000000	3.595305	2.823075e-12
17	3.595305	4.000000	3.595305	5.528911e-13
18	3.595305	4.000000	3.595305	1.079137e-13
19	3.595305	4.000000	3.595305	2.087219e-14

In all the examples, the *regula falsi* method appears to converge much faster.

Solution:

(a) C program (double precision)

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

```
#define SIGN(a) ((a) >= 0.) /* sign function as a macro */

/* function: falsi
   ---------------
   Implements the regula falsi algorithm.
   func       - a pointer to a function
   a, b       - initial interval
   xtol, ftol - tolerance in position and value
   nmax       - maximum number of iterations
*/
double falsi(double (*func)(double), double a, double b,
      double xtol, double ftol, int nmax)
{
  int n;
  double fa, fb, fc, c, d;

  fa = func(a);
  fb = func(b);
  /* assert that signs are different */
  assert (SIGN(fa) != SIGN(fb));
  for (n=0; n < nmax; n++) {
    d = b-a;
    c = a+d*fa/(fa-fb);
    fc = func(c);

    /* print out the table */
    printf("n=%d a=%f b=%f c=%f |f(c)|=%e\n",n,a,b,c,fabs(fc));

    /* return if the root is located to the tolerance */
    if (fabs(d) <= xtol && fabs(fc) <= ftol) return c;

    if (SIGN(fc) == SIGN(fa)) {
      a = c;
      fa = fc;
    } else {
      b = c;
      fb = fc;
    }
  }

  fprintf(stderr,"Warning: Exact root is not found after %d iterations\n",
  nmax);
  return c;
}

/* function from assignment 4. (b) */
static double function_b (double x)
{
  return (x + exp(x));
}

/* function from assignment 4. (c) */
static double function_c (double x)
{
  return (x + 4.*cos(x));
}

int main (void)
{
```

```
  int nmax=20; /* maximum number of iterations */
  double xtol=0., ftol=0., a, b, c;

  /* assignment 4. (a) */
  a=2.;  b=4.;  c = falsi(&sin, a,b, xtol, ftol, nmax);

  /* assignment 4. (b) */
  a=-1.; b=0.;  c = falsi(&function_b, a,b, xtol, ftol, nmax);

  /* assignment 4. (c) */
  a=-2.; b=-1.; c = falsi(&function_c, a,b, xtol, ftol, nmax);
  a=2.;  b=3.;  c = falsi(&function_c, a,b, xtol, ftol, nmax);
  a=3.;  b=4.;  c = falsi(&function_c, a,b, xtol, ftol, nmax);

  return 0;
}
```

(b) Fortran-90 program (single precision)

```
! sign function (different from Fortran intrinsic)
integer function sign(x)
  real, intent (in) :: x
  if (x >= 0.) then
     sign = 1
  else
     sign = 0
  end if
end function sign

! function: falsi
! ----------------
! Implements the regula falsi algorithm.
function falsi(func, a, b, xtol, ftol, nmax) result (c)
  integer, intent (in)     :: nmax        ! maximum number of iterations
  real,    intent (in)     :: xtol, ftol  ! tolerance in position and value
  real,    intent (in out) :: a, b        ! interval
  interface
     real function func (x)               ! function interface
       real, intent (in) :: x
     end function func
  end interface
  integer  :: n, sign
  real     :: fa, fb, fc, c, d
  external :: sign

  fa = func(a)
  fb = func(b)
  ! assert that signs are different
  if (sign(fa) == sign(fb)) then
     write (0,*) "Assertion failed."
     stop
  end if
  do n=0, nmax-1
     d = b-a
     c = a+d*fa/(fa-fb)
     fc = func(c)

     ! print out the table
     print *, "n=", n, "a=", a, "b=", b, "c=", c, "|f(c)|=", abs(fc)
```

```
     ! return if the root is located to the tolerance
     if (abs(d) <= xtol .and. abs(fc) <= ftol) return

     if (sign(fc) == sign(fa)) then
        a = c
        fa = fc
     else
        b = c
fb = fc
     end if
  end do

  write (0,*) "Warning: Exact root is not found after", nmax, "iterations"
end function falsi

! function from assignment 4. (a)
real function function_a (x)
  real, intent (in) :: x
  function_a = sin(x)
end function function_a

! function from assignment 4. (b)
real function function_b (x)
  real, intent (in) :: x
  function_b = x + exp(x)
end function function_b

! function from assignment 4. (c)
real function function_c (x)
  real, intent (in) :: x
  function_c = x + 4.*cos(x)
end function function_c

program Main
  integer  :: nmax=20 ! maximum number of iterations
  real     :: xtol=0., ftol=0., a, b, c
  real     :: function_a, function_b, function_c, bisect
  external :: function_a, function_b, function_c, bisect

  ! assignment 5. (a)
  a=2.;  b=4.;  c = falsi(function_a, a,b, xtol, ftol, nmax)

  ! assignment 5. (b)
  a=-1.; b=0.;  c = falsi(function_b, a,b, xtol, ftol, nmax)

  ! assignment 5. (c)
  a=-2.; b=-1.; c = falsi(function_c, a,b, xtol, ftol, nmax)
  a=2.;  b=3.;  c = falsi(function_c, a,b, xtol, ftol, nmax)
  a=3.;  b=4.;  c = falsi(function_c, a,b, xtol, ftol, nmax)
end program Main
```