Ex 0.1

Use the function nest to evaluate $P(x) = 1 + x + \ldots + x^{50}$ at x = 1.00001. (Use the Matlab ones command to save typing.) Find the error of the computation by comparing with the equivalent expression $Q(x) = (x^{51} - 1)/(x - 1)$.

```
degree = 50;
coefficients = ones(51, 1);
x = 1.00001;
y = nest(degree, coefficients, x)
```

```
y =
  51.012752082749991
```

```
y_equi = (x^51-1)/(x-1)
```

```
y_equi =
  51.012752082745230
```

```
error=y-y_equi
```

```
error =
     4.760636329592671e-12
```

From the above results, we can observe that the computation with the function nest introduces error = 4.7606e-12.

Ex 0.4

Calculate the expressions that follow in double precision arithmetic (using Matlab, for example) for $x = 10^{-1}, \ldots, 10^{-14}$. Then, using an alternative form of the expression that doesn't suffer from subtracting nearly equal numbers, repeat the calculation and make a table of results. Report the number of correct digits in the original expression for each x.

(a) $E_1 = \dfrac{1 - \sec x}{\tan^2 x}$

Solution: an alternative form of the expression to (a) that doesn't suffer from subtracting nearly equal numbers is given as $E_2 = \dfrac{-2\sin^2\left(\frac{x}{2}\right)}{\cos x \tan^2 x}$.

```
x = ones(14,1);
for i=1:1:14
    x(i) = 1/10^i;
end
y_original = (1-sec(x))./(tan(x).^2);
y_equivalent = -2*(sin(x/2).^2)./(cos(x).*tan(x).^2);
correct_digits = zeros(14,1);
correct_digits(1:4) = [13,11,8,8];
format long;
table(x,y_original,y_equivalent,correct_digits,...
```

```
    'VariableNames',{'x','E_1','E_2','correct_digits'})
```

ans = 14×4 table

	x	E_1	E_2	correct_digits
1	0.10000...	-0.4987...	-0.4987...	13
2	0.01000...	-0.4999...	-0.4999...	11
3	0.00100...	-0.4999...	-0.4999...	8
4	0.00010...	-0.4999...	-0.4999...	8
5	0.00001...	-0.5000...	-0.4999...	0
6	0.00000...	-0.5000...	-0.4999...	0
7	0.00000...	-0.5107...	-0.4999...	0
8	0.00000...	0	-0.5000...	0
9	0.00000...	0	-0.5000...	0
10	0.00000...	0	-0.5000...	0
11	0.00000...	0	-0.5000...	0
12	0.00000...	0	-0.5000...	0
13	0.00000...	0	-0.5000...	0
14	0.00000...	0	-0.5000...	0

From the above table, we can observe the original expression E_1 brings huge error when the input goes under 10^{-8}, while the equivalent expression which avoid suffering from nearly equal numbers gives the correct output.

$$(\mathrm{b})E_1 = \frac{1-(1-x)^3}{x}$$

Solution: n alternative form of the expression to (a) that doesn't suffer from subtracting nearly equal numbers is given as $E_2 = \frac{3x-3x^2+x^3}{x} = 3-3x+x^2$

```
x = ones(14,1);
for i=1:1:14
    x(i) = 1/10^i;
end
y_original = (1-(1-x).^3)./x;
y_equivalent = 3-3*x+x.^2;
correct_digits = zeros(14,1);
correct_digits(1:14) = [2,4,6,13,10,11,7,8,8,0,0,5,0,3];
format long;
table(x,y_original,y_equivalent,correct_digits,...
    'VariableNames',{'x','E_1','E_2','correct_digits'})
```

ans = 14×4 table

	x	E_1	E_2	correct_digits
1	0.10000...	2.70999...	2.71000...	2

	x	E_1	E_2	correct_digits
2	0.01000...	2.97009...	2.97010...	4
3	0.00100...	2.99700...	2.99700...	6
4	0.00010...	2.99970...	2.99970...	13
5	0.00001...	2.99997...	2.99997...	10
6	0.00000...	2.99999...	2.99999...	11
7	0.00000...	2.99999...	2.99999...	7
8	0.00000...	2.99999...	2.99999...	8
9	0.00000...	2.99999...	2.99999...	8
10	0.00000...	3.00000...	2.99999...	0
11	0.00000...	3.00000...	2.99999...	0
12	0.00000...	2.99993...	2.99999...	5
13	0.00000...	3.00093...	2.99999...	0
14	0.00000...	2.99760...	2.99999...	3

From the above table, we can get similar conclusions that original expression E_1 suffers severe errors while the equivalent expression E_2 avoid the error.

Ex 1.1

Use the Bisection Method to find the root to eight correct decimal places. (a) $x^5 + x = 1$ (b) $\sin x = 6x + 5$ (c) $\ln x + x^2 = 3$

(a) $x^5 + x = 1$

```
f = @(x)x^5+x-1;
solution = bisection(f,0.7,0.8,23);
solution
```

```
solution =
   0.754877668619156
```

(b) $\sin x = 6x + 5$

```
f=@(x)sin(x)-6*x-5;
solution = bisection(f,-1.2,-0.8,24);
solution
```

```
solution =
  -0.970898926258087
```

(c) $\ln x + x^2 = 3$

```
f=@(x) log2(x)+x^2-3;
solution = bisection(f,1.4,1.6,24);
```

```
solution
```

```
solution =
   1.541361361742020
```

Ex 1.3

(a) Use fzero to find the root of $f(x) = 2x\cos x - 2x + \sin x^3$ on [-0.1,0.2]. Report the forward and backward errors.

The corresponding results are given as follows:

```
f = @(x)2*x*cos(x)-2*x+sin(x^3);
x = fzero(f,[-0.1,0.2])
```

```
x =
     1.688148348885743e-04
```

```
forward_error = abs(0-x)
```

```
forward_error =
     1.688148348885743e-04
```

```
backward_error = f(x)
```

```
backward_error =
     0
```

(b) Run the Bisection Method with initial interval [-0.1,0.2] to find as many correct digits as possible, and report your conclusion.

The corresponding results are given as follows:

```
f = @(x)2*x*cos(x)-2*x+sin(x^3);
x = bisection(f,-0.1,0.2,50);
x
```

```
x =
    -8.410091572427412e-05
```

```
x = bisection(f,-0.1,0.2,100);
x
```

```
x =
    -8.410091572435123e-05
```

```
x = bisection(f,-0.1,0.2,200);
x
```

```
x =
    -8.410091572435123e-05
```

```
forward_error = abs(0-x)
```

```
forward_error =
     8.410091572435123e-05
```

```
backward_error = f(x)
```

```
backward_error =
     0
```

Ex 1.4

Each equation has one root. Use Newton's Method to approximate the root to eight correct decimal places.

Solutions: The corresponding results are given as follows:

(a) $x^3 = 2x + 2$

```
f = @(x)x^3-2*x-2;
f_ = @(x)3*x^2-2;
x = 0.2;
x_old = 1;
while abs(x-x_old)>10^(-8)
    x_old = x;
    x = x-f(x)/f_(x);
end
x
```

```
x =
   1.769292354238631
```

(b) $e^x + x = 7$

```
f = @(x)exp(x)+x-7;
f_ = @(x)exp(x)+1;
x = 0.2;
x_old = 1;
while abs(x-x_old)>10^(-8)
    x_old = x;
    x = x-f(x)/f_(x);
end
x
```

```
x =
   1.672821698628907
```

(c) $e^x + \sin x = 4$

```
f = @(x)exp(x)+sin(x)-4;
f_ = @(x)exp(x)+cos(x);
x = 0.2;
x_old = 1;
while abs(x-x_old)>10^(-8)
    x_old = x;
    x = x-f(x)/f_(x);
end
x
```

```
x =
   1.129980498650832
```

Ex 1.5

Use the Secant Method to find the (single) solution of each equation in Exercise 1.

Solutions: The corresponding results are given as follows:

(a) $x^3 = 2x + 2$

```
f = @(x)x^3-2*x-2;
x_0 = 1;
x_1 = 2;
while x_1-x_0>10^(-8)
    x_new = x_1-f(x_1)/((f(x_1)-f(x_0))/(x_1-x_0));
    x_0=x_1;
    x_1=x_new;
end
x_1
```

```
x_1 =
   1.600000000000000
```

(b) $e^x + x = 7$

```
f = @(x)exp(x)+x-7;
x_0 = 1;
x_1 = 2;
while x_1-x_0>10^(-8)
    x_new = x_1-f(x_1)/((f(x_1)-f(x_0))/(x_1-x_0));
    x_0=x_1;
    x_1=x_new;
end
x_1
```

```
x_1 =
   1.578707247902504
```

(c) $e^x + \sin x = 4$

```
f = @(x)exp(x)+sin(x)-4;
x_0 = 1;
x_1 = 2;
while x_1-x_0>10^(-8)
    x_new = x_1-f(x_1)/((f(x_1)-f(x_0))/(x_1-x_0));
    x_0=x_1;
    x_1=x_new;
end
x_1
```

```
x_1 =
   1.092906580116090
```

Appendix: The source codes of the function nest, no_nest and bisection.

```
function y = nest(d, c, x, b)
if nargin<4
    b = zeros(d, 1);
end
y = c(d+1);
for i=d:-1:1
    y = y.*(x-b(i))+c(i);
end
end


function y = no_nest(d, c, x)
x_new = zeros(d+1, 1);
for i = 0:1:d
    x_new(i+1)=x^i;
end
y = c*x_new;
end



function solution = bisection(f, left, right, tolerance)
if sign(f(left))*sign(f(right))>=0
    error('ERROR')
end
fl = f(left);
fr = f(right);
iteration = 0; %最大迭代次数
while iteration<tolerance
    iteration = iteration+1;
    mid = left+(right-left)/2;
    fm = f(mid);
    if fm==0
        break;
    end
    if sign(fl)*sign(fm)<0
        fr =fm;
        right = mid;
    end
    if sign(fm)*sign(fr)<0
        fl = fm;
        left = mid;
    end
end
solution = left+(right-left)/2;
end
```