1. Sketch curves to roughly locate the roots of the following equations.

(i) $2x + \cos x = 0$

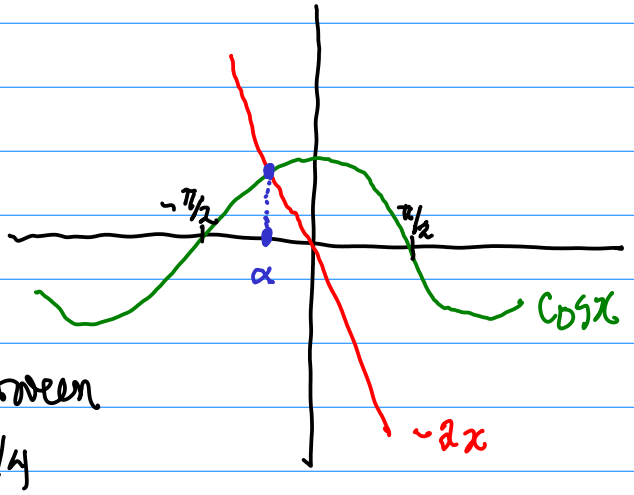
(ii) $x + \ln x = 0$

(iii) $x(x-2) - e^x = 0$

(i) $2x + \cos x = 0$

$g(x) = \cos x$

$h(x) = -2x$

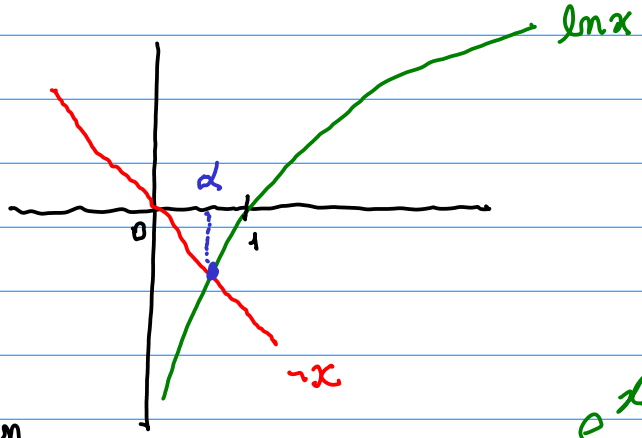


The solution α is somewhere between $-\pi/2$ and 0. Maybe $\alpha \approx -\pi/4$

(ii) $x + \ln x = 0$

$g(x) = \ln x$

$h(x) = -x$

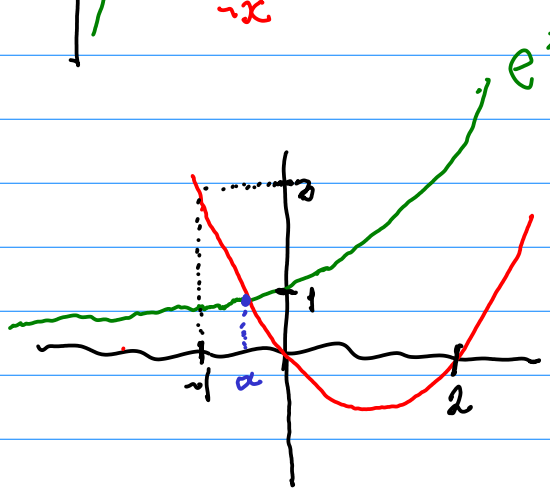


The solution α is somewhere between 0 and 1. Maybe $\alpha \approx 1/2$

(iii) $x(x-2) - e^x = 0$

$g(x) = e^x$

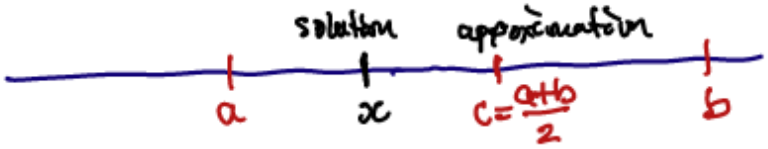
$h(x) = x(x-2)$



Since $h(-1) = (-1)(-3) = 3$ the solution α is somewhere between -1 and 0. Maybe $\alpha \approx -1/2$.

2. What is the maximum error after n iterations of the bisection method?

If the bounding interval of the solution at $n=0$ is $[a,b]$, then without performing any iterations we can approximate the solution x by the midpoint $c = (a+b)/2$.



since x is between a and b, then the distance from c to x satisfies

$$|c-x| \le \max(|c-a|, |c-b|) = \frac{|b-a|}{2}.$$

Now each iteration of bisection cuts the error in half. Therefore the error E_n at the nth iteration satisfies

$$E_n \le \frac{1}{2^n}\frac{|b-a|}{2} = \frac{|b-a|}{2^{n+1}}.$$

In particular, the maximum error must be less than $\frac{|b-a|}{2^{n+1}}$.

3. Each equation in Question 1 has only one root. For each equation use bisection method and a suitable programming language to find the root correct to 4D.

(i) $2x + \cos x = 0$

The Julia program

```
using Printf

f(x)=2*x+cos(x)
a=-pi/2; b=0
@printf("%2d %14s %14s %14s %14s\n",0,"a","b","c","f(c)")
for n=1:50
    c=(a+b)/2
    fc=f(c)
    @printf("%2d %14.6e %14.6e %14.6e %14.6e\n",n,a,b,c,fc)
    if abs(a-c)<0.000005
        break
    end
    if fc>0
        global b=c
    else
        global a=c
    end
end
```

↑ Note that 0.000005 means the accuracy of c is good to 4D. We stop at one magnitude smaller so further enable correctly rounding to 4D. In some borderline cases even that's not enough.

produces the output

	a	b	c	f(c)
1	-1.570796e+00	0.000000e+00	-7.853982e-01	-8.636895e-01
2	-7.853982e-01	0.000000e+00	-3.926991e-01	1.384814e-01
3	-7.853982e-01	-3.926991e-01	-5.890486e-01	-3.466276e-01
4	-5.890486e-01	-3.926991e-01	-4.908739e-01	-9.982644e-02
5	-4.908739e-01	-3.926991e-01	-4.417865e-01	2.041636e-02
6	-4.908739e-01	-4.417865e-01	-4.663302e-01	-3.943602e-02
7	-4.663302e-01	-4.417865e-01	-4.540583e-01	-9.442161e-03
8	-4.540583e-01	-4.417865e-01	-4.479224e-01	5.504067e-03
9	-4.540583e-01	-4.479224e-01	-4.509904e-01	-1.964811e-03
10	-4.509904e-01	-4.479224e-01	-4.494564e-01	1.770688e-03
11	-4.509904e-01	-4.494564e-01	-4.502234e-01	-9.679704e-05
12	-4.502234e-01	-4.494564e-01	-4.498399e-01	8.370115e-04
13	-4.502234e-01	-4.498399e-01	-4.500316e-01	3.701238e-04
14	-4.502234e-01	-4.500316e-01	-4.501275e-01	1.366675e-04
15	-4.502234e-01	-4.501275e-01	-4.501754e-01	1.993626e-05
16	-4.502234e-01	-4.501754e-01	-4.501994e-01	-3.843013e-05
17	-4.501994e-01	-4.501754e-01	-4.501874e-01	-9.246868e-06
18	-4.501874e-01	-4.501754e-01	-4.501814e-01	5.344715e-06
19	-4.501874e-01	-4.501814e-01	-4.501844e-01	-1.951072e-06

and implies $\alpha \approx -.4502$ to 4D accuracy.

(ii) $x + \ln x = 0$

The Julia program

```
using Printf

f(x)=x+log(x)
a=0; b=1
@printf("%2d %14s %14s %14s %14s\n",0,"a","b","c","f(c)")
for n=1:50
    c=(a+b)/2
    fc=f(c)
    @printf("%2d %14.6e %14.6e %14.6e %14.6e\n",n,a,b,c,fc)
    if abs(a-c)<0.000005
        break
    end
    if fc>0
        global b=c
    else
        global a=c
    end
end
```

produces the output

```
 0              a              b              c           f(c)
 1   0.000000e+00   1.000000e+00   5.000000e-01  -1.931472e-01
 2   5.000000e-01   1.000000e+00   7.500000e-01   4.623179e-01
 3   5.000000e-01   7.500000e-01   6.250000e-01   1.549964e-01
 4   5.000000e-01   6.250000e-01   5.625000e-01  -1.286414e-02
 5   5.625000e-01   6.250000e-01   5.937500e-01   7.245308e-02
 6   5.625000e-01   5.937500e-01   5.781250e-01   3.015983e-02
 7   5.625000e-01   5.781250e-01   5.703125e-01   8.741677e-03
 8   5.625000e-01   5.703125e-01   5.664062e-01  -2.037452e-03
 9   5.664062e-01   5.703125e-01   5.683594e-01   3.358017e-03
10   5.664062e-01   5.683594e-01   5.673828e-01   6.617638e-04
11   5.664062e-01   5.673828e-01   5.668945e-01  -6.874732e-04
12   5.668945e-01   5.673828e-01   5.671387e-01  -1.276207e-05
13   5.671387e-01   5.673828e-01   5.672607e-01   3.245240e-04
14   5.671387e-01   5.672607e-01   5.671997e-01   1.558867e-04
15   5.671387e-01   5.671997e-01   5.671692e-01   7.156379e-05
16   5.671387e-01   5.671692e-01   5.671539e-01   2.940122e-05
17   5.671387e-01   5.671539e-01   5.671463e-01   8.319662e-06
18   5.671387e-01   5.671463e-01   5.671425e-01  -2.221183e-06
```

and implies $\alpha \approx 0.5671$ to 4D accuracy. This is a case where it would not be possible to round to 4D if abs(a-c)< 0.00005 were used as the stopping condition.

(iii) $x(x-2) - e^x = 0$

The Julia program

```
using Printf

f(x)=x*(x-2)-exp(x)
a=-1; b=0
@printf("%2d %14s %14s %14s %14s\n",0,"a","b","c","f(c)")
for n=1:50
    c=(a+b)/2
    fc=f(c)
    @printf("%2d %14.6e %14.6e %14.6e %14.6e\n",n,a,b,c,fc)
    if abs(a-c)<0.000005
        break
    end
    if fc<0
        global b=c
    else
        global a=c
    end
end
```

Reverse this inequality since f(c) < 0 means the guess is too large in this case.

produces the output

```
$ julia bisect15.jl
 0              a              b              c           f(c)
 1  -1.000000e+00   0.000000e+00  -5.000000e-01   6.434693e-01
 2  -5.000000e-01   0.000000e+00  -2.500000e-01  -2.163008e-01
 3  -5.000000e-01  -2.500000e-01  -3.750000e-01   2.033357e-01
 4  -3.750000e-01  -2.500000e-01  -3.125000e-01  -8.959379e-03
 5  -3.750000e-01  -3.125000e-01  -3.437500e-01   9.655788e-02
 6  -3.437500e-01  -3.125000e-01  -3.281250e-01   4.364304e-02
 7  -3.281250e-01  -3.125000e-01  -3.203125e-01   1.730295e-02
 8  -3.203125e-01  -3.125000e-01  -3.164062e-01   4.162085e-03
 9  -3.164062e-01  -3.125000e-01  -3.144531e-01  -2.401069e-03
10  -3.164062e-01  -3.144531e-01  -3.154297e-01   8.799023e-04
11  -3.154297e-01  -3.144531e-01  -3.149414e-01  -7.607347e-04
12  -3.154297e-01  -3.149414e-01  -3.151855e-01   5.954592e-05
13  -3.151855e-01  -3.149414e-01  -3.150635e-01  -3.506039e-04
14  -3.151855e-01  -3.150635e-01  -3.151245e-01  -1.455313e-04
15  -3.151855e-01  -3.151245e-01  -3.151550e-01  -4.299330e-05
16  -3.151855e-01  -3.151550e-01  -3.151703e-01   8.276165e-06
17  -3.151703e-01  -3.151550e-01  -3.151627e-01  -1.735860e-05
18  -3.151703e-01  -3.151627e-01  -3.151665e-01  -4.541228e-06
```

and implies $\alpha \approx -0.3152$ to 4D accuracy.

4. Compare the results obtained when the bisection method, the method of false position and the secant method are used with starting values 0 and 1 to solve the equation

$$xe^{-x} = 1 - x.$$

Let $f(x) = xe^{-x} - 1 + x$.

Then bisection

```
using Printf

f(x)=x*exp(-x)-1+x
a=0; b=1
@printf("%2d %14s %14s %14s %14s\n",0,"a","b","c","f(c)")
for n=1:50
    c=(a+b)/2
    fc=f(c)
    @printf("%2d %14.6e %14.6e %14.6e %14.6e\n",n,a,b,c,fc)
    if abs(a-c)<0.000005
        break
    end
    if fc>0
        global b=c
    else
        global a=c
    end
end
```

produces the output

```
 0              a              b              c           f(c)
 1   0.000000e+00   1.000000e+00   5.000000e-01  -1.967347e-01
 2   5.000000e-01   1.000000e+00   7.500000e-01   1.042749e-01
 3   5.000000e-01   7.500000e-01   6.250000e-01  -4.046161e-02
 4   6.250000e-01   7.500000e-01   6.875000e-01   3.319671e-02
 5   6.250000e-01   6.875000e-01   6.562500e-01  -3.291985e-03
 6   6.562500e-01   6.875000e-01   6.718750e-01   1.503517e-02
 7   6.562500e-01   6.718750e-01   6.640625e-01   5.892580e-03
 8   6.562500e-01   6.640625e-01   6.601562e-01   1.305580e-03
 9   6.562500e-01   6.601562e-01   6.582031e-01  -9.918774e-04
10   6.582031e-01   6.601562e-01   6.591797e-01   1.571821e-04
11   6.582031e-01   6.591797e-01   6.586914e-01  -4.172649e-04
12   6.586914e-01   6.591797e-01   6.589355e-01  -1.300207e-04
13   6.589355e-01   6.591797e-01   6.590576e-01   1.358583e-05
14   6.589355e-01   6.590576e-01   6.589966e-01  -5.821616e-05
15   6.589966e-01   6.590576e-01   6.590271e-01  -2.231484e-05
16   6.590271e-01   6.590576e-01   6.590424e-01  -4.364425e-06
17   6.590424e-01   6.590576e-01   6.590500e-01   4.610723e-06
18   6.590424e-01   6.590500e-01   6.590462e-01   1.231543e-07
```

The method of false position

```
using Printf

f(x)=x*exp(-x)-1+x
a=0; b=1
@printf("%2d %14s %14s %14s %14s\n",0,"a","b","c","f(c)")
for n=1:50
    c=b-(b-a)/(f(b)-f(a))*f(b)
    fc=f(c)
    @printf("%2d %14.6e %14.6e %14.6e %14.6e\n",n,a,b,c,fc)
    if abs(fc)<5e-6
        break
    end
    if fc>0
        global b=c
    else
        global a=c
    end
end
```

produces the output

```
 0              a              b              c           f(c)
 1   0.000000e+00   1.000000e+00   7.310586e-01   8.298954e-02
 2   0.000000e+00   7.310586e-01   6.750375e-01   1.872432e-02
 3   0.000000e+00   6.750375e-01   6.626302e-01   4.211924e-03
 4   0.000000e+00   6.626302e-01   6.598510e-01   9.466713e-04
 5   0.000000e+00   6.598510e-01   6.592269e-01   2.127330e-04
 6   0.000000e+00   6.592269e-01   6.590867e-01   4.780264e-05
 7   0.000000e+00   6.590867e-01   6.590552e-01   1.074149e-05
 8   0.000000e+00   6.590552e-01   6.590481e-01   2.413661e-06
```

The secant method

```
using Printf

f(x)=x*exp(-x)-1+x
a=0; b=1
@printf("%2d %14s %14s %14s %14s\n",0,"a","b","c","f(c)")
for n=1:50
    c=b-(b-a)/(f(b)-f(a))*f(b)
    fc=f(c)
    @printf("%2d %14.6e %14.6e %14.6e %14.6e\n",n,a,b,c,fc)
    if abs(fc)<5e-6
        break
    end
    global a=b
    global b=c
end
```

produces the output

0	a	b	c	f(c)
1	0.000000e+00	1.000000e+00	7.310586e-01	8.298954e-02
2	1.000000e+00	7.310586e-01	6.527149e-01	-7.461885e-03
3	7.310586e-01	6.527149e-01	6.591779e-01	1.551291e-04
4	6.527149e-01	6.591779e-01	6.590463e-01	2.901199e-07

Commentary. It took bisection 18 iterations to arrive at an approximation $\alpha^* \approx 6.590462 \times 10^{-1}$ of α which satisfies

$$e_{abs}(\alpha^*) \leq 0.000005$$

however this is a guaranteed error bound.

With the method of false position the approximation $\alpha^* \approx 6.590481 \times 10^{-1}$ with similar accuracy was obtained after only 8 iterations. However the guaranteed error estimate based on the length of the interval at that point was only

$$\frac{|b-a|}{2} \approx 0.3295$$

which, though correct, doesn't provide a useful guaranteed bound on the error. If one switches the stopping criterion back to the length of the interval (as in bisection) it actually takes 26 iterations before the left endpoint is updated and the iteration terminated.

The difficulty of not updating the value of a is remedied by the secant method with the loss of an interval that provides a guaranteed bound on the error. This tradeoff results in the approximation $\alpha^* = 6.5904163 \times 10^{-1}$ after only 4 iterations. That's much faster,

5. Consider the equation $e^x + x = 0$.

(i) Use the iteration $x_{n+1} = -\exp x_n$ with $x_0 = -1$ to find the solution to 3D.

(ii) [Extra Credit and Math 666] Write $x = -e^x$ and add Mx to both sides to obtain

$$(M+1)x = Mx - e^x.$$

Solving for x on the left yields the iteration

$$x_{n+1} = \frac{Mx_n - \exp x_n}{M+1}.$$

Choose a value for M that leads to a faster converging iteration and demonstrate the convergence numerically.

(i) The code

```
using Printf

f(x)=exp(x)+x
xn=-1
@printf("%2s %14s %14s\n","n","xn","f(xn)")
for n=0:25
    @printf("%2d %14.6e %14.6e\n",n,xn,f(xn))
    global xn=-exp(xn)
end
```

produces

n	xn	f(xn)
0	-1.000000e+00	-6.321206e-01
1	-3.678794e-01	3.243212e-01
2	-6.922006e-01	-1.917271e-01
3	-5.004735e-01	1.057700e-01
4	-6.062435e-01	-6.084775e-02
5	-5.453958e-01	3.421655e-02
6	-5.796123e-01	-1.949687e-02
7	-5.601155e-01	1.102765e-02
8	-5.711431e-01	-6.263768e-03
9	-5.648793e-01	3.549378e-03
10	-5.684287e-01	-2.013992e-03
11	-5.664147e-01	1.141904e-03
12	-5.675566e-01	-6.477254e-04
13	-5.669089e-01	3.673203e-04
14	-5.672762e-01	-2.083338e-04
15	-5.670679e-01	1.181517e-04
16	-5.671861e-01	-6.701004e-05
17	-5.671190e-01	3.800394e-05
18	-5.671570e-01	-2.155380e-05
19	-5.671355e-01	1.222405e-05
20	-5.671477e-01	-6.932802e-06

This output clearly suggests that the approximate solution

$x^* \approx -0.567$

is correct to 3D.

(ii) Define $g(x) = \frac{Mx - e^x}{M+1}$. Then the iteration can be written as $x_{n+1} = g(x)$. The smaller g' is near the true solution α the faster this iteration will converge. Since

$$g'(x) = \frac{M - e^x}{M+1}$$

and the solution is near the starting value $x_0 = 1$, then $M = e^{-1}$ is a reasonable choice.

The code

```
using Printf

f(x)=exp(x)+x
g(x)=(M*x-exp(x))/(M+1)
M=exp(-1)
xn=-1
@printf("%2s %14s %14s\n","n","xn","f(xn)")
for n=0:10
    @printf("%2d %14.6e %14.6e\n",n,xn,f(xn))
    global xn=g(xn)
end
```

produces the output

```
 n             xn             f(xn)
 0  -1.000000e+00  -6.321206e-01
 1  -5.378828e-01   4.610049e-02
 2  -5.715850e-01  -6.955207e-03
 3  -5.665003e-01   1.007721e-03
 4  -5.672370e-01  -1.469130e-04
 5  -5.671296e-01   2.139890e-05
 6  -5.671453e-01  -3.117305e-06
 7  -5.671430e-01   4.541078e-07
 8  -5.671433e-01  -6.615152e-08
 9  -5.671433e-01   9.636526e-09
10  -5.671433e-01  -1.403787e-09
```

} This output shows that after 5 iterations the approximation is correct to 7 significant digits.

6. Consider finding $\ln a$ by solving $a = e^x$ for x using Newton-Raphson iterations.

(i) Derive the Newton-Raphson iteration

$$x_{n+1} = x_n - 1 + a\exp(-x_n).$$

(ii) What happens with this iteration if $a = -3$ and $x_0 = 1$?

(iii) [Extra Credit and Math 666] Is it true or false that the iteration always converges for $a > 0$ and $x_0 = 1$? If true explain why; if false provide a counter example.

(i) By definition Newton's method is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

To solve for a we write $f(x) = e^x - a$ so that $f(\alpha) = 0$ implies that $\alpha = \ln a$. Differentiating as $f'(x) = e^x$ yields

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{e^{x_n} - a}{e^{x_n}} = x_n - 1 + a e^{-x_n}$$

which was to be shown.

(ii) If $a = -3$ then we are effectively trying to find $\ln(-3)$. Since there is no real solution to $f(x) = e^x + a$, something is likely to go wrong. By hand we obtain

$x_0 = 1$

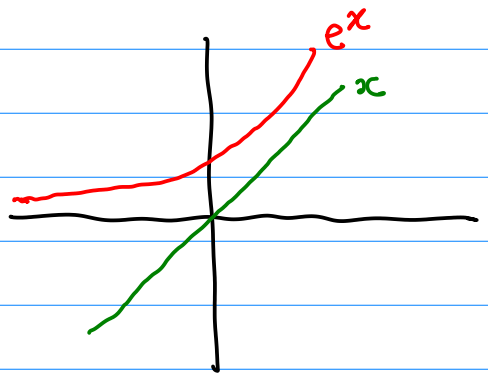
$x_1 = 1 - 1 - 3e^{-1} = -3e^{-1} \approx -1.103638\ldots$

$x_2 = -3e^{-1} - 1 - 3e^{3e^{-1}} = -3(e^{-1} + e^{3e^{-1}}) - 1 \approx -11.148986\ldots$

$x_3 = -3(e^{-1} + e^{3e^{-1}}) - 1 - 3e^{-3(e^{-1}+e^{3e^{-1}})-1} \approx -208492.23\ldots$

It looks like $x_n \to -\infty$ when $a = -3$.

While the numerical evidence is compelling, it's not difficult to use inequalities to show the same. Note that the graphs



show that $e^x > x$ for all x. Consequently $e^{-x} > -x$ or equivalently that $-e^{-x} < x$.

Upon setting $g(x) = x - 1 - 3e^{-x}$ it follows that

$$g(x) = x - 1 - 3e^{-x} < x - 1 + 3x = 4x - 1 < 4x$$

By induction we have

$$x_2 = g(x_1) < 4x_1$$
$$x_3 = g(x_2) < 4x_2 < 4^2 x_1$$
$$\vdots$$
$$x_{n+1} = g(x_n) < 4x_n < 4^n x_1$$

Thus, using the fact that $x_1 = -3e^{-1}$ it follows that

$$x_{n+1} < -4^n \cdot 3e^{-1} \to -\infty \quad \text{as } n \to \infty.$$

Note that the sequence x_n actually converges to $-\infty$ much faster than the above estimates show.

7. Use the Newton–Raphson method to solve to 4D each equation in Question 1.

(i) $2x + \cos x = 0$

We use the starting value $x_0 = -\frac{\pi}{4}$ suggested by the graph in question 1. The code

```
# newton.jl -- Perform five iterations of Newton's method
f(x)=2*x+cos(x)
x0=-pi/4

using Symbolics
@variables t
D(g)=expand_derivatives(Differential(t)(g))
as="df(t)="*string(D(f(t)))
eval(Meta.parse(as))

xn=x0
for n=1:20
    xold=xn
    global xn=xn-f(xn)/df(xn)
    println("n=",n,", xn=",xn)
    if abs(xold-xn)<0.000005
        break
    end
end
```

Could also find f' by hand, but this way reduces typos.

The difference between successive approximations is used to determine when to end the loop.

produces the output

```
n=1, xn=-0.46635291863256073
n=2, xn=-0.45023140545751744
n=3, xn=-0.45018361171715576
n=4, xn=-0.45018361129487355
```

which indicates the desired precision is obtained in 3 iterations

(ii) $x + \ln x = 0$

We use the starting value $x_0 = 1/2$ suggested by the graph in question 1. The code

```
# newton.jl -- Perform five iterations of Newton's method
f(x)=x+log(x)
x0=1/2

using Symbolics
@variables t
D(g)=expand_derivatives(Differential(t)(g))
as="df(t)="*string(D(f(t)))
eval(Meta.parse(as))

xn=x0
for n=1:20
    xold=xn
    global xn=xn-f(xn)/df(xn)
    println("n=",n,", xn=",xn)
    if abs(xold-xn)<0.000005
        break
    end
end
```

produces the output

```
n=1, xn=0.5643823935199818
n=2, xn=0.5671389877150601
n=3, xn=0.5671432903993691
```

which again indicates the desired precision is obtained in 3 iterations

(iii) $x(x-2)-e^x=0$

We use the starting value $x_0 = 1/2$ suggested by the graph in question 1. The code

```
# newton.jl -- Perform five iterations of Newton's method
f(x)=x*(x-2)-exp(x)
x0=-1/2

using Symbolics
@variables t
D(g)=expand_derivatives(Differential(t)(g))
as="df(t)="*string(D(f(t)))
eval(Meta.parse(as))

xn=x0
for n=1:20
    xold=xn
    global xn=xn-f(xn)/df(xn)
    println("n=",n,", xn=",xn)
    if abs(xold-xn)<0.000005
        break
    end
end
```

produces the output

```
n=1, xn=-0.32158217938492784
n=2, xn=-0.3151756028584475
n=3, xn=-0.3151678249557108
n=4, xn=-0.31516782494427475
```

which indicates the desired precision is obtained in 3 iterations.