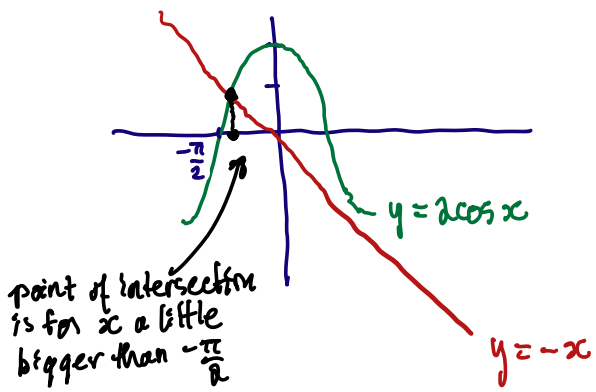
- HW2 from First Steps in Numerical Analysis (Due Oct 14)
  - Step 6 Exercise 2abc
  - Step 7 Checkpoint 3, Exercise 3 (except to 4D)
  - Step 8 Exercise 2abc
  - Step 9 Exercises 1, 3
  - Step 10 Exercises 2, 5

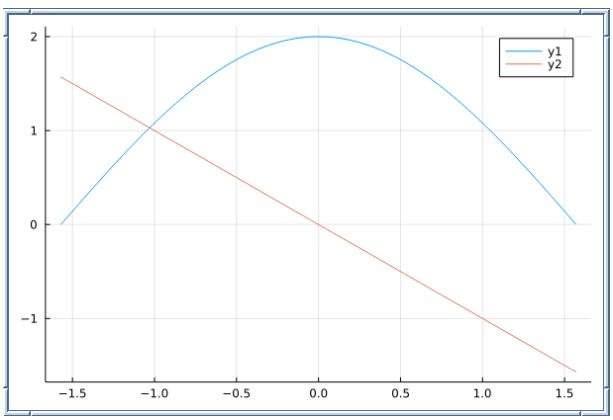
Step 6

2. Use sketch curves to roughly locate all the roots of the following equations.
   (a) $x + 2\cos x = 0$.
   (b) $x + e^x = 0$.
   (c) $x(x-1) - e^x = 0$.
   (d) $x(x-1) - \sin x = 0$.

(a) $2\cos x = -x$

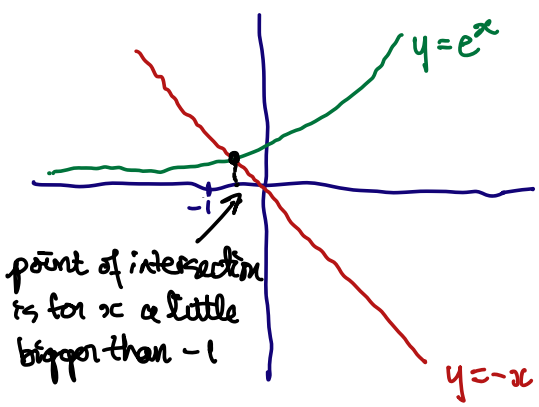


$g(x) = 2\cos x$, $h(x) = -x$

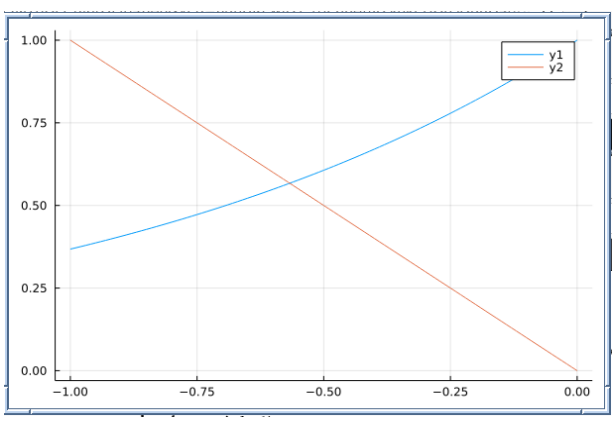



```
julia> using Plots
       xs=-pi/2:0.01:pi/2;
       g(x)=2*cos(x); h(x)=-x;
       plot(xs,g.(xs)); plot!(xs,h.(xs))
```

(b) $e^x = -x$

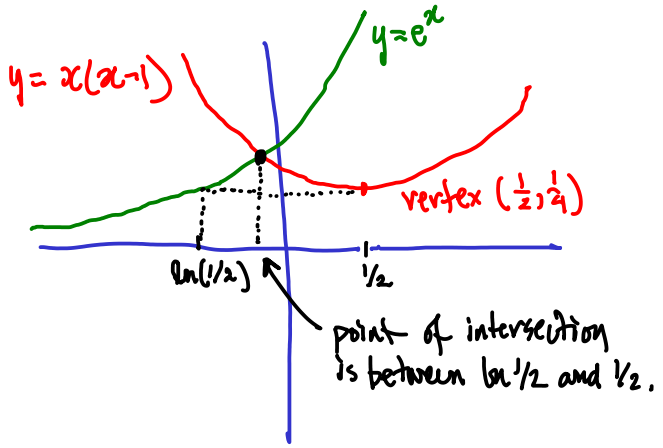


$g(x) = e^x$, $h(x) = -x$

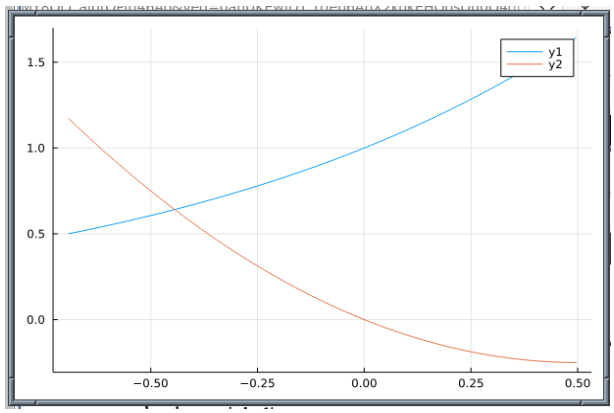



```
julia> using Plots
       xs=-1:0.01:0;
       g(x)=exp(x); h(x)=-x;
       plot(xs,g.(xs)); plot!(xs,h.(xs))
```

(c) $e^x = x(x-1)$



$g(x) = e^x$, $h(x) = x(x-1)$

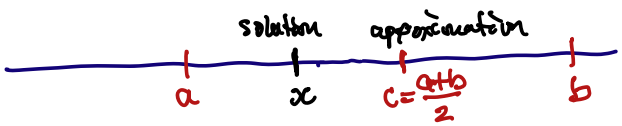



```
julia> using Plots
       xs=log(1/2):0.01:1/2;
       g(x)=exp(x); h(x)=x*(x-1);
       plot(xs,g.(xs)); plot!(xs,h.(xs))
```

Step 7

3. 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 $\frac{|b-a|}{2^{n+1}}$.

Step 7

3. Each equation in Exercises 2(a)–2(c) of Step 6 on page 26 has only one root. For each equation use the bisection method to find the root correct to ~~2D~~ 4D.

For 4D the error E_n must satisfy

$$E_n \leq 0.00005 = 0.5 \times 10^{-4}.$$

4 zeros here.

To perform these calculations we consider the Julia code

```
julia> function bisect4D(a,b)
           n=0
           while true
               c=(a+b)/2
               println("n=$n\n\t[a,b]=[$a,$b]\n\tc=$c")
               if abs(c-a)<0.5e-4
                   return c
               end
               if f(c)<0
                   a=c
               else
                   b=c
               end
               n+=1
           end
       end
```

(a) Based on the graph we know the solution $x \in [-\frac{\pi}{2}, 0]$.

Therefore $a = -\frac{\pi}{2}$, $b = 0$ and solving

$$E_n = \frac{\pi/2}{2^{n+1}} \leq 0.5 \times 10^{-4}$$ for n yields that

$$n \geq \frac{\ln 2/\pi + \ln 0.5 \times 10^{-4}}{\ln 1/2} - 1 \approx 13.9$$

```
julia> (log(2/pi)+log(0.5e-4))/log(1/2)-1
13.939208509021768
```

Consequently we take $n = 14$ and perform 14 bisection steps to guarantee an approximation good to 4D.

Now use the bisect4D function defined above to compute

```
julia> f(x)=2*cos(x)+x;

julia> bisect4D(-pi/2,0)
n=0
    [a,b]=[-1.5707963267948966,0]
    c=-0.7853981633974483
n=1
    [a,b]=[-1.5707963267948966,-0.7853981633974483]
    c=-1.1780972450961724
n=2
    [a,b]=[-1.1780972450961724,-0.7853981633974483]
    c=-0.9817477042468103
n=3
    [a,b]=[-1.1780972450961724,-0.9817477042468103]
    c=-1.0799224746714913
n=4
    [a,b]=[-1.0799224746714913,-0.9817477042468103]
    c=-1.030835089459151
n=5
    [a,b]=[-1.030835089459151,-0.9817477042468103]
    c=-1.0062913968529807
n=6
    [a,b]=[-1.030835089459151,-1.0062913968529807]
    c=-1.0185632431560658
```

```
n=7
    [a,b]=[-1.030835089459151,-1.0185632431560658]
    c=-1.0246991663076084
n=8
    [a,b]=[-1.030835089459151,-1.0246991663076084]
    c=-1.0277671278833798
n=9
    [a,b]=[-1.030835089459151,-1.0277671278833798]
    c=-1.0293011086712653
n=10
    [a,b]=[-1.030835089459151,-1.0293011086712653]
    c=-1.030068099065208
n=11
    [a,b]=[-1.030068099065208,-1.0293011086712653]
    c=-1.0296846038682368
n=12
    [a,b]=[-1.030068099065208,-1.0296846038682368]
    c=-1.0298763514667224
n=13
    [a,b]=[-1.0298763514667224,-1.0296846038682368]
    c=-1.0297804776674795
n=14
    [a,b]=[-1.0298763514667224,-1.0297804776674795]
    c=-1.0298284145671008
-1.0298284145671008
```

Therefore, the desired approximation is -1.0298

(b) Based on the graph $x \in [-1, 0]$. Thus

$$E_n = \frac{1}{2^{n+1}} \leq 0.5 \times 10^{-4}$$

```
julia> log(0.5e-4)/log(1/2)-1
13.287712379549449
```

implies

$$n \geq \frac{\log 0.5 \times 10^{-4}}{\log 1/2} - 1 \approx 13.3$$

Therefore $n = 14$ iterations of the bisection method is enough

```
julia> f(x)=exp(x)+x
f (generic function with 1 method)

julia> bisect4D(-1,0)
n=0
    [a,b]=[-1,0]
    c=-0.5
n=1
    [a,b]=[-1,-0.5]
    c=-0.75
n=2
    [a,b]=[-0.75,-0.5]
    c=-0.625
n=3
    [a,b]=[-0.625,-0.5]
    c=-0.5625
n=4
    [a,b]=[-0.625,-0.5625]
    c=-0.59375
n=5
    [a,b]=[-0.59375,-0.5625]
    c=-0.578125
n=6
    [a,b]=[-0.578125,-0.5625]
    c=-0.5703125
```

```
n=7
    [a,b]=[-0.5703125,-0.5625]
    c=-0.56640625
n=8
    [a,b]=[-0.5703125,-0.56640625]
    c=-0.568359375
n=9
    [a,b]=[-0.568359375,-0.56640625]
    c=-0.5673828125
n=10
    [a,b]=[-0.5673828125,-0.56640625]
    c=-0.56689453125
n=11
    [a,b]=[-0.5673828125,-0.56689453125]
    c=-0.567138671875
n=12
    [a,b]=[-0.5673828125,-0.567138671875]
    c=-0.5672607421875
n=13
    [a,b]=[-0.5672607421875,-0.567138671875]
    c=-0.56719970703125
n=14
    [a,b]=[-0.56719970703125,-0.567138671875]
    c=-0.567169189453125
-0.567169189453125
```

Therefore, the desired approximation is $x = -0.5672$

(C) Based on the graph $x \in [0, 1/2]$. Therefore

$$\tilde{w}_n = \frac{1/2 - \ln 1/2}{2^{n+1}} \leq 0.5 \times 10^{-4}$$

implies

$$n \geq \frac{\log 0.5\times10^{-4} - \log(1/2 - \log 1/2)}{\log 1/2} - 1 \approx 13.5$$

```
julia> (log(0.5e-4)-log(1/2-log(1/2)))/log(1/2)
14.54248439707297
```

Consequently, taking $n = 14$ is sufficient.

```
julia> f(x)=exp(x)-x*(x-1)
f (generic function with 1 method)

julia> bisect4D(log(1/2),1/2)
n=0
    [a,b]=[-0.6931471805599453,0.5]
    c=-0.09657359027997264
n=1
    [a,b]=[-0.6931471805599453,-0.09657359027997264]
    c=-0.39486038541995894
n=2
    [a,b]=[-0.6931471805599453,-0.39486038541995894]
    c=-0.5440037829899521
n=3
    [a,b]=[-0.5440037829899521,-0.39486038541995894]
    c=-0.4694320842049555
n=4
    [a,b]=[-0.4694320842049555,-0.39486038541995894]
    c=-0.43214623481245723
n=5
    [a,b]=[-0.4694320842049555,-0.43214623481245723]
    c=-0.45078915950870635
n=6
    [a,b]=[-0.45078915950870635,-0.43214623481245723]
    c=-0.44146769716058176
```

```
n=7
    [a,b]=[-0.45078915950870635,-0.44146769716058176]
    c=-0.44612842833464406
n=8
    [a,b]=[-0.44612842833464406,-0.44146769716058176]
    c=-0.4437980627476129
n=9
    [a,b]=[-0.44612842833464406,-0.4437980627476129]
    c=-0.4449632455411285
n=10
    [a,b]=[-0.4449632455411285,-0.4437980627476129]
    c=-0.44438065414437067
n=11
    [a,b]=[-0.44438065414437067,-0.4437980627476129]
    c=-0.4440893584459918
n=12
    [a,b]=[-0.44438065414437067,-0.4440893584459918]
    c=-0.4442350062951812
n=13
    [a,b]=[-0.4442350062951812,-0.4440893584459918]
    c=-0.44416218237058647
n=14
    [a,b]=[-0.44416218237058647,-0.4440893584459918]
    c=-0.44412577040828916
-0.44412577040828916
```

Therefore, the desired approximation is $x = -0.4441$

Step 8

2. Compare the results obtained when
   (a) the bisection method,
   (b) the method of false position, and
   (c) the secant method

   are used (with starting values 0.7 and 0.9) to solve the equation

$$3 \sin x = x + \frac{1}{x}$$

The code

```
# step8p2.jl

function bisect4D(a,b)
    n=0
    while true
        c=(a+b)/2
        println("n=$n\n\t[a,b]=[$a,$b]\n\tc=$c")
        if abs(c-a)<0.5e-4
            return c
        end
        if f(c)<0
            a=c
        else
            b=c
        end
        n+=1
    end
end

function falsepos(a,b)
    n=0
    while true
        c=a-f(a)*(b-a)/(f(b)-f(a))
        println("n=$n\n\t[a,b]=[$a,$b]\n\tc=$c")
        if abs(c-a)<0.5e-4 || abs(c-b)<0.5e-4
            return c
        end
        if f(c)<0
            a=c
        else
            b=c
        end
        n+=1
    end
end
```

Same function as step 7

midpoint for bisection

modify to use the secant line to determine c

```
function secant(a,b)
    n=0
    while true
        c=a-f(a)*(b-a)/(f(b)-f(a))
        println("n=$n\n\t[a,b]=[$a,$b]\n\tc=$c")
        if abs(c-a)<0.5e-4
            return c
        end
        a=b
        b=c
        n+=1
    end
end

f(x)=3*sin(x)-x-1/x
a=0.7
b=0.9

println("(a)\tBisection Method")
x=bisect4D(a,b)
println("The solution is ",x)


println("\n(b)\tMethod of False Position")
x=falsepos(a,b)
println("The solution is ",x)


println("\n(c)\tThe Secant Method")
x=secant(a,b)
println("The solution is ",x)
```

remove the conditional and always replace the oldest approximation a by the newer ones. Note that $x \in [a,b]$ is not maintained.

The output from running the above code is

```
julia> include("step8p2.jl")
(a) Bisection Method
n=0
    [a,b]=[0.7,0.9]
    c=0.8
n=1
    [a,b]=[0.7,0.8]
    c=0.75
n=2
    [a,b]=[0.75,0.8]
    c=0.775
n=3
    [a,b]=[0.75,0.775]
    c=0.7625
n=4
    [a,b]=[0.7625,0.775]
    c=0.76875
n=5
    [a,b]=[0.7625,0.76875]
    c=0.765625
n=6
    [a,b]=[0.7625,0.765625]
    c=0.7640625
n=7
    [a,b]=[0.7625,0.7640625]
    c=0.7632812499999999
n=8
    [a,b]=[0.7625,0.7632812499999999]
    c=0.7628906249999999
n=9
    [a,b]=[0.7628906249999999,0.7632812499999999]
    c=0.7630859374999999
n=10
    [a,b]=[0.7630859374999999,0.7632812499999999]
    c=0.7631835937499999
n=11
    [a,b]=[0.7630859374999999,0.7631835937499999]
    c=0.7631347656249998 ← correct to 4D
The solution is 0.7631347656249998
```

```
(b) Method of False Position
n=0
    [a,b]=[0.7,0.9]
    c=0.7732695469940357
n=1
    [a,b]=[0.7,0.7732695469940357]
    c=0.7638348045775717
n=2
    [a,b]=[0.7,0.7638348045775717]
    c=0.7631675526990714 ← last digit rounds wrong
n=3
    [a,b]=[0.7,0.7631675526990714]
    c=0.7631205062201909 ← correct to 4D
The solution is 0.7631205062201909

(c) The Secant Method
n=0
    [a,b]=[0.7,0.9]
    c=0.7732695469940357
n=1
    [a,b]=[0.9,0.7732695469940357]
    c=0.7614285343714231
n=2
    [a,b]=[0.7732695469940357,0.7614285343714231]
    c=0.7631365260379867 ← correct to 4D
n=3
    [a,b]=[0.7614285343714231,0.7631365260379867]
    c=0.7631169759192102
n=4
    [a,b]=[0.7631365260379867,0.7631169759192102]
    c=0.7631169382632903
The solution is 0.7631169382632903
```

ignore

Extra computations that weren't needed to find the correct answer. Maybe my stopping condition was too strict.

The bisection method took $n=11$ iterations to find an approximation that was good to 4D.

The method of false position took $n=3$ iterations.

The secant method took $n=2$ iterations.

Clearly the secant method converged the fastest will the bisection method was the slowest. The method of false position did remarkably well for this problem and has the advantage over the secant method of guaranteed convergence.

Step 9

1. Assuming the initial guess $x_0 = 1$, show by the method of simple iteration that one root of the equation $2x - 1 - 2\sin x = 0$ is 1.4973.

First solve for x as

$2x = 1 + 2\sin x$ and then $x = \frac{1 + 2\sin x}{2}$.

Then define

$$\phi(x) = \frac{1+2\sin x}{2}$$

and iterate as $x_{n+1} = \phi(x_n)$ until subsequent iterations differ by a small tolerance.

```
julia> phi(x)=(1+2*sin(x))/2
phi (generic function with 1 method)

julia> function iterate(xnm1,tol)
           for n=1:30
               xn=phi(xnm1)
               println("x[$n]=$xn")
               if abs(xn-xnm1)<tol
                   return xn
               end
               xnm1=xn
           end
           println("Failed to reach tolerance!")
       end
iterate (generic function with 1 method)

julia> iterate(1,0.5e-6)
x[1]=1.3414709848078965
x[2]=1.4738199802680 82
x[3]=1.495301478075512
x[4]=1.4971516171506023
x[5]=1.4972894537667139
x[6]=1.497299586057349
x[7]=1.4973003301 2862
x[8]=1.497300384765936
1.497300384765936
```

choose tolerance 0.5×10^{-6} small enough to clearly demonstrate the convergence to 1.4973

correct digits

Step 9

3. Use the method of simple iteration to find to 3*D* the root of the equation given in Exercise 2(b) of Step 6 on page 26.

The equation is $e^x + x = 0$. Solving for x yields the possible iteration $\varphi(x) = -e^x$. As demonstrated by the Julia code

```
julia> phi(x)=-exp(x)
phi (generic function with 1 method)

julia> iterate(-1,0.5e-6)
x[1]=-0.36787944117144233
x[2]=-0.6922006275553464
x[3]=-0.5004735005636368
x[4]=-0.6062435350855974
x[5]=-0.545395785975027
x[6]=-0.5796123355033789
x[7]=-0.5601154613610891
x[8]=-0.571143115080177
x[9]=-0.5648793473910495
x[10]=-0.5684287250290607
x[11]=-0.5664147331468833
x[12]=-0.5675566373282834
x[13]=-0.5669089119214953
x[14]=-0.5672762321755696
x[15]=-0.5670678983907884
x[16]=-0.567186050099357
x[17]=-0.5671190400572149
x[18]=-0.5671570440012975
x[19]=-0.5671354902062784
x[20]=-0.5671477142601192
x[21]=-0.567140781458298
x[22]=-0.56714471334657
x[23]=-0.5671424834013071
x[24]=-0.5671437480994115
x[25]=-0.5671430308342419
x[26]=-0.56714343762633
-0.56714343762633
```

slow convergence

This iteration does converge, though somewhat slowly, to the approximation $x \approx -0.567$.

Faster convergence may be obtained by writing

$$x = -e^x \qquad (1+M)x = Mx - e^x \qquad \varphi(x) = \frac{Mx - e^x}{1+M}$$

and choosing the constant M so $|\varphi'(x)| \ll 1$ near the root. Thus

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

suggests taking $M = e^{-1}$ could be a good choice, since $x_0 = -1$ was our initial approximation of the root.

The Julia code

```
julia> phi(x)=(M*x-exp(x))/(1+M)
phi (generic function with 1 method)

julia> M=exp(-1)
0.36787944117144233

julia> iterate(-1,0.5e-6)
x[1]=-0.5378828427399902
x[2]=-0.5715849987225449
x[3]=-0.5665003348813453
x[4]=-0.567237037739373
x[5]=-0.5671296357256952
x[6]=-0.5671452795746927
x[7]=-0.5671430006418975
x[8]=-0.567143332621317
-0.567143332621317
```

Confirms faster convergence and again that $x = -0.567$,

Step 10

2. Derive the Newton-Raphson iteration formula

$$x_{n+1} = x_n - \frac{x_n^k - a}{k x_n^{k-1}}$$

for finding the k-th root of a.

We already know Newton's method is given by

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

To obtain the formula for finding the k^{th} root simply requires choosing f appropriately. By definition the k^{th} root $\sqrt[k]{a}$ is defined as the number such that $x^k = a$. This suggests the natural choice for f as

$$f(x) = x^k - a \quad \text{so that} \quad f'(x) = kx^{k-1}$$

Then

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

Step10

5. Use the Newton-Raphson method to find (to 4*D*) the root of each equation in Exercises 2(a)–2(c) of Step 6 on page 26.

To avoid calculus errors, we use the symbolics package to find the derivatives of f. Slight modifications to the code from computer lab 3 the lead to

```
# step10p5.jl

using Symbolics

function newton(f,x0)
    @variables t
    D(g)=expand_derivatives(Differential(t)(g))
    dfx=D(f(t))
    println("Solving f(t) = 0 by Newton's method where")
    println("\tf(t) = ",string(f(t)))
    println("\tdf(t) = ",string(dfx))
    dfp=eval(build_function(dfx,t))
    df(t)=Base.invokelatest(dfp,t)
    xn=x0
    println("n=0, xn=",xn)
    for n=1:5
        xn=xn-f(xn)/df(xn)
        println("n=",n,", xn=",xn)
    end
    return xn
end

print("(a)\t")
x=newton(x->2*cos(x)+x,-pi/2)
println("The solution is ",x)

print("\n(b)\t")
x=newton(x->exp(x)+x,-1)
println("The solution is ",x)

print("\n(c)\t")
x=newton(x->exp(x)-x*(x-1),0)
println("The solution is ",x)
```

Note: Is was not expected that you write code like this to solve the problem. Even a hand calculator could have been used. However, this code illustrates one way to automate the solution to this problem.

Instead of build_function one could use Meta.parse("t -> "*string(dfx)) similar to how we did things in Lab 3.

This subterfuge was not required in the REPL but is needed inside a function to make sure the correct derivative is JIT compiled.

Now, we can solve each of the three parts easily with just a call to newton.

The output of the above program is

```
julia> include("step10p5.jl")
(a) Solving f(t) = 0 by Newton's method where
    f(t) = t + 2cos(t)
    df(t) = 1 - 2sin(t)
n=0, xn=-1.5707963267948966
n=1, xn=-1.0471975511965979
n=2, xn=-1.0299220484622262
n=3, xn=-1.0298665299069372
n=4, xn=-1.0298665293222589
n=5, xn=-1.0298665293222589
The solution is -1.0298665293222589

(b) Solving f(t) = 0 by Newton's method where
    f(t) = t + exp(t)
    df(t) = 1 + exp(t)
n=0, xn=-1
n=1, xn=-0.5378828427399902
n=2, xn=-0.5669869914054133
n=3, xn=-0.567143285989123
n=4, xn=-0.5671432904097838
n=5, xn=-0.5671432904097838
The solution is -0.5671432904097838

(c) Solving f(t) = 0 by Newton's method where
    f(t) = exp(t) - t*(t - 1)
    df(t) = 1 + exp(t) - 2t
n=0, xn=0
n=1, xn=-0.5
n=2, xn=-0.4449577392259856
n=3, xn=-0.4441304126788834
n=4, xn=-0.4441302288239757
n=5, xn=-0.4441302288239666
The solution is -0.4441302288239666
```

Starting guess $-\frac{\pi}{2}$ based on the graph in step 6(a).

4D at $n=3$

answer ≈ -1.0299

Starting guess based on step 6(b)

4D at $n \approx 3$

answer ≈ -0.5671

starting guess based on step 6(c).

4D at $n \approx 3$

answer ≈ -0.4441

In summary an approximation accurate to 4D was obtained in all three cases after $n \approx 3$ iterations. This convergence depends on the choice for the initial approximation x_0, but should not be very large in any case due to the quadratic convergence of Newton's method.