BEE 4750/5750 Homework 0

Caroline Herzog (crh227)

2022-08-31

Problem 1

Problem 1.1

```
julia> function square_number(x)
           output = x^2
           return output
       end
square_number (generic function with 1 method)
```

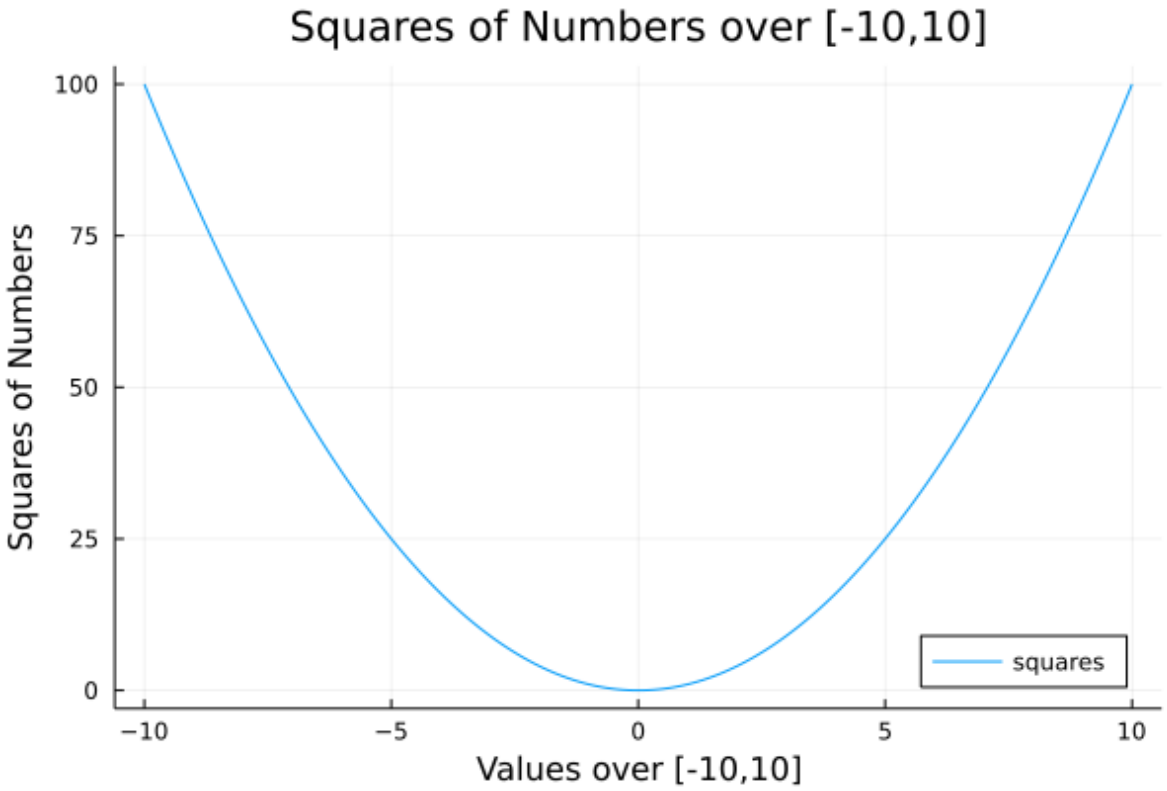
Problem 1.2

We can see that $5^2 = 25$

Problem 1.3

```
julia> # opening "Plots" package
       Pkg.add("Plots")

julia> using Plots

julia> # evaluate square_number over [-10, 10]
       x = -10:0.1:10
-10.0:0.1:10.0

julia> y = square_number.(x)
201-element Vector{Float64}:
 100.0
  98.01
  96.04000000000002
  94.08999999999999
  92.16
  90.25
  88.36000000000001
  86.49000000000001
```

```
 84.63999999999999
 82.80999999999999
  ⋮
 84.63999999999999
 86.49000000000001
 88.36000000000001
 90.25
 92.16
 94.08999999999999
 96.04000000000002
 98.01
100.0

julia> # generate plot
       Plots.plot(x, y, label = "squares", title = "Squares of Numbers over [-1
```



Problem 2

Problem 2.1

$\sqrt{x}$ must be between a and x/a because *a* and *x/a* are a pair of factors that multiply to equal *x*, and must necessarily be either greater or smaller than $\sqrt{x}$. For example, we can think of *x* as being a rectangle with area *x* and side lengths of *a* and *x/a*. If $a = x/a$, our rectangle is actually a square, and $\sqrt{x} = a = x/a$. However, if we begin to make *a* smaller, *x/a* must increase, since the $Area = x$ remains constant.

Problem 2.2

```
julia> function guess_sqrt(x,a)
         # function returns estimate of sqrt(x) given initial guess a
         # given x > 0 - use this algorithm to find sqrt(x)
         to = 0.02    # set tolerance
         diff = 100      # set error

         # set up while loop
         while to <= diff

           # divide x/a, update a to avg.
             a = 0.5*(a + (x/a))

           # calculate new difference
             diff = a - x/a
         end
         return a
       end
guess_sqrt (generic function with 1 method)
```

```
julia> # testing above function for sqrt(2)
       aGuess = guess_sqrt(2,0.3)
1.4170872677117539

julia> println("The approx. sqrt(2) is $aGuess")
The approx. sqrt(2) is 1.4170872677117539
```

Problem 3

Problem 3.1

```
julia> # making a random vector, V, with length 20
       V = rand(20)
20-element Vector{Float64}:
 0.4570905289636782
 0.6021429192502291
 0.2785431322437306
 0.487136759178622
 0.6773894863390858
 0.8603258948871173
 0.8381912398621258
 0.1874854608056773
 0.1890903741824358
 0.36017487974542384
 0.3862741376300338
```

```
0.20491936878410055
0.7396308873648164
0.024877494749318885
0.40990081938909817
0.37368887329612166
0.23571653052339303
0.40910771632430654
0.25281380036555134
0.5824659016376966
```

Problem 3.2

```
julia> function mean(V)
       sumV = 0

         # for loop to calculate mean
         b = length(V)

         for i in 1:b
           sumV = sumV + V[i]
         end

         MnEst = sumV / b
         return MnEst
       end
mean (generic function with 1 method)

julia> function demean(V)

         # running above mean() fxn to obtain est.
         MnEst = mean(V)

         # to avoid mutating old data
         vNew = V
         d = length(vNew)

         for i in 1:d
           vNew[i] = vNew[i] - MnEst
         end

       return vNew
       end
demean (generic function with 1 method)
```

Problem 3.3

```
julia> # create vector, A, of 10 elements
       A = [0 0 1 1 1 1 1 1 0 0]
```

```
1×10 Matrix{Int64}:
 0  0  1  1  1  1  1  1  0  0
```

Problem 3.4

```
julia> # create rand 5x5 matrix, B
       B = rand(5,5)
5×5 Matrix{Float64}:
 0.478225  0.775949    0.0806402  0.780479   0.399922
 0.77923   0.00489479  0.759252   0.388293   0.120052
 0.465787  0.988177    0.879402   0.475588   0.101267
 0.635922  0.655478    0.996101   0.738179   0.0170083
 0.312173  0.929521    0.290679   0.0380429  0.755116

julia> C = zeros(5,5)
5×5 Matrix{Float64}:
 0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0

julia> # find and subtract mean from each column
       for h in 1:5
         colB = B[:,h]

         C[:,h] = demean(colB)
       end

julia> display(C)
5×5 Matrix{Float64}:
 -0.0560425   0.105145   -0.520575   0.296362     0.121249
  0.244963   -0.665909    0.158038  -0.0958231   -0.158621
 -0.0684804   0.317373    0.278187  -0.00852879  -0.177406
  0.101655   -0.0153261   0.394886   0.254063    -0.261665
 -0.222094    0.258717   -0.310536  -0.446074     0.476443
```

Problem 4

Problem 4.1

```
julia> # load Distributions.jl
       import Pkg

julia> Pkg.add("Distributions")
```

```
julia> using Distributions

julia> # create log normal distribution given data
       mu = log(0.03)  # log mean
-3.506557897319982

julia> sig = 0.005     # stand. dev.
0.005

julia> myLogNorm = LogNormal(mu, sig)
Distributions.LogNormal{Float64}(μ=-3.506557897319982, σ=0.005)

julia> # draw 100 samples using rand() function
       ytVal = rand(myLogNorm, 100)
100-element Vector{Float64}:
 0.03000205160895554
 0.03015311005164698
 0.02988696311663252
 0.03015406128627349
 0.030089586275871927
 0.030022645299965983
 0.030282242083511962
 0.03013943826272234
 0.029875293412792603
 0.03004120814120361
 ⋮
 0.030206786438807507
 0.02981422819857077
 0.03010856626824241
 0.030014169222849652
 0.02986944922981434
 0.02954278112556663
 0.030463947520515188
 0.030077289531798047
 0.03019557039613531
```

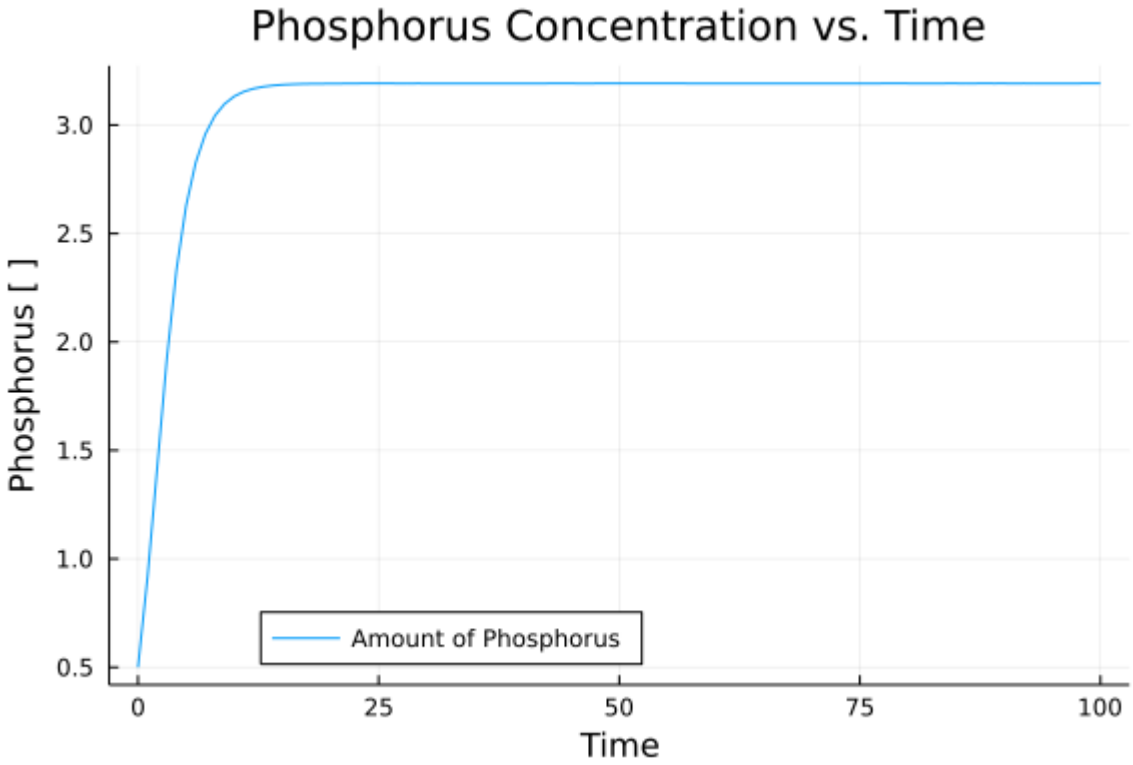
Problem 4.2

```
julia> function PhosLake(a, b, q, T, x0)
       # function takes in constant parameters a, b, q, period T [years],
       # and intial concentration of phophorus x0
       PhosC = zeros(T+1)
       PhosC[1] = x0   # initial P at T = 0

       for i = 1:T
           # setting up parameters
           y = ytVal[i]
           xOld = PhosC[i]
```

```
            # updating new Xt+1 value
            PhosC[i+1] = xOld + a + y + (xOld^q)/(1+xOld^q) - b*xOld
        end
        return PhosC
        end
PhosLake (generic function with 1 method)
```

Problem 4.3

```
julia> # setting up parameters and running function above
       T = 100
100

julia> phosData = PhosLake(0.4, 0.42, 2, T, 0.5)
101-element Vector{Float64}:
 0.5
 0.9200020516089555
 1.4221609531996153
 1.9238929451688715
 2.3333077444365857
 2.6282327888906
 2.82793716411656
 2.9593404608578435
 3.044073696023493
 3.0980327642228005
 ⋮
 3.192041389225218
 3.1918257663719873
 3.191984047064559
 3.191989524855553
 3.1918482613199877
 3.1914324562586702
 3.192091245752161
 3.1921202867455896
 3.1922568921790413

julia> phosTime = 0:1:T
0:1:100

julia> # plot
       import Pkg

julia> Pkg.add("Plots")

julia> using Plots

julia> Plots.plot(phosTime, phosData, label = "Amount of Phosphorus", title = "
```



References

1. https://www.overleaf.com/learn/latex/Mathematical_expressions
2. https://docs.juliahub.com/CalculusWithJulia/AZHbv/0.0.5/precalc/vectors.html

---

Published from solution-template.jmd using Weave.jl v0.10.10 on 2022-08-31.