Nick Largey
Exploration #3
COS574

2.1 Concept Check:

1.a: False
1.b: False
1.c: True
2. It satisfies the conditions that:
   1. A solution to the problem exists.
   2. The solution is unique.
   3. A small perturbation in the problem data results in a small perturbation in the solution.
3. Data error and Computational error
4. Truncation error generally appears when you have a solution that is too precise to store, so bit's are lopped off.
   Round off error happens when you make the decision to reduce precision and round the last number either up or down.
5. Absolute Error: when the y value is large in magnitude.
   Relative Error: When the y value is small and requires more precision
6. If the condition number (the ratio of the forward error to the backward error) is small.
7. A well-conditioned problem allows for small amounts of error, a stable algorithm requires that an input of x results
   in an exact solution, and with a small change in x (called x hat) an exact solution is still the result.
8. Forward error is the difference in the true solution and the estimated solution, Backwards error will show the magnitude
   of how the inputs of the true and estimated solutions will effect those solutions.
9. This means that the input alpha sub n has the rate of convergence of O(Beta sub n)
10. Rate of convergence is important because having an algorithm that converges quickly allows us to calculate an approximate
    solution that is sufficiently accurate while using less computational effort and resources.

2.2.2

beta = 10
p = 5
Base 10: 9.9999
Base 2 = 1.0011

beta = 2
p = 10
Base 10 = 1.998046875
Base 2: 1.11111111

2.2.7

L = 10^-5
U = 10^9
p = 13

2.2.8

```
beta = 10
precision = 5
L = -20
U = 20

Biggest: 9.9999 x 10^20
Smallest: 0.0001 x 10^-20

# 2.2.23
m = 1000
sd = 0.1
n = 1000

rn_num = m .+ sd * randn(n)

function v1(r, n, m)
    x = 0
    for i = 1:n
      x += (r[i] - n)^2
    end
    x *= 1/n

    return x
  end

function v2(r, n, m)
    x = 0
    for i = 1:n
      x += r[i]^2
    end
    x *= 1/n
    x -= m^2
    return x
  end

println(v1(rn_num, n, m))
println(v2(rn_num, n, m)) # much more susceptible to floating-point issues since it's using
                          # two numbers that are squared as opposed to the first which added
                          # together that are then squared.

# 2.2.25
Any digits smaller that 1.0e-17 will result in Catastrophic Cancellation since that will cause
an floating point underflow.
Rewritten: 2x/1-x^2

# 2.2.26
I believe the left side because you would have less of a chance of encountering round off or
truncation error and increase the u-value which would lower the risk of catastrophic cancellation.
```