Homework 8

Due date: Friday, March 18 at 11:59pm in Gradescope

Use the following file name for submitting to Gradescope:

`Homework_8.py`

This assignment is worth 25 points.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
try:
    %matplotlib inline
except:
    pass
```

Problem 1 (3 points)

We can generalize the Fibonacci sequence so that we have the recurrence relationship

$$p_n = ap_{n-1} + bp_{n-2},\ a, b \in \mathbb{R},\ n \geq 2,$$

where we start the sequence p_n with the initial conditions

$$p_0 = s_0,\ p_1 = 1,\ s_0 \in \mathbb{R}.$$

(a)

Using the guess $p_n = \lambda^n$, show that you get two solutions for λ, say $\lambda_{\pm}$, where

$$\lambda_{\pm} = \frac{1}{2}\left(a \pm \sqrt{a^2 + 4b}\right)$$

Remember: the word "show" should indicate to you to show work.

Solution for Problem 1a

$$\frac{\lambda^n}{\lambda^n} = \frac{a\lambda^{n-1} + b\lambda^{n-2}}{\lambda^n}$$

$$1 = a\lambda^{-1} + b\lambda^{-2}$$

$$\lambda^2 = a\lambda + b$$

$$0 = \lambda^2 - a\lambda - b$$

$$\lambda_{\pm} = \frac{a \pm \sqrt{a^2 + 4b}}{2}$$

$$p_n = c_1\lambda_+^n + c_2\lambda_-^n$$

$$p_0 = c_1 + c_2$$

$$p_1 = c_1\lambda_+ + c_2\lambda_-$$

(b)

Writing the general solution as

$$p_n = c_+\lambda_+^n + c_-\lambda_-^n$$

show that when we take our initial conditions into account, we find that the constants c_+ and c_- are given by

$$c_+ = \frac{1 - s_0\lambda_-}{\lambda_+ - \lambda_-},\ c_- = \frac{s_0\lambda_+ - 1}{\lambda_+ - \lambda_-}$$

Solution for Problem 1b

$$(s_0 - c_2)\lambda_+ + c_2\lambda - = 1$$

$$s_0\lambda_+ - c_2\lambda_+ + c_2\lambda_- = 1$$

$$c_2(\lambda_- - \lambda_+) = -s_0\lambda_+$$

$$c_2 = \frac{-s_0\lambda_+}{\lambda_- - \lambda_+}$$

(c)

Let $b = 1$ and $a = 2$.

Determine which of the following is correct (you must show work to obtain full credit):

> A) $|\lambda_+| > 1, \text{ and } |\lambda_-| < 1$
>
> B) $|\lambda_-| > 1, \text{ and } |\lambda_+| < 1$

In [2]:

```
# Solution for Problem 1c
```

$$\lambda_+ = \frac{2 + \sqrt{4 + 4}}{2} = \frac{2 + \sqrt{8}}{2} = 1 + \sqrt{2}$$

$$|\lambda_+| = \left|1 + \sqrt{2}\right| > 1$$

$$\lambda_- = \frac{2 - \sqrt{4 + 4}}{2} = \frac{2 - \sqrt{8}}{2} = 1 - \sqrt{2}$$

$$|\lambda_-| = \left|1 - \sqrt{2}\right| < 1$$

Problem 2 (6 points)

(This problem is autograded.)

The $3n+1$ sequence is generated using the following rules:

- Start with a positive integer n.
- If n = 1, stop.
- If n is even, replace it with n/2.
- If n is odd, replace it with 3n+1.

So if we started with $n = 3$, then we would generate the sequence

$$3, 10, 5, 16, 8, 4, 2, 1$$

The unsolved mathematical problem (called the Collatz conjecture) is whether this code can run forever i.e. it is unknown whether there are any starting values n which generate a sequence which goes on for forever. Thus, an interesting associated quantity we would want to know is how many terms a given value of n generates via the $3n+1$ sequence. We call this number $L(n)$. For example then, using our example of $n = 3$ above, we have that

$$L(3) = 8$$

Write a function called `L` to find $L(n)$ for any given n and then generate a plot of $L(n)$ for $1 \leq n \leq 1000$. Discuss any trends you observe, make sure to label the axes and plot the values with dots or stars instead of lines (This can be done using `'o'` or `'*'` as the third argument given to `plt.plot`). The function should accept a single integer argument n and return a single integer value $L(n)$.

To generate a sequence of integers using `numpy`, it is preferable to use `np.arange`, instead of `np.linspace`. For example, for this problem, you should use

> `n_vals = np.arange(1, 1000+1)`

instead of

> `n_vals = np.linspace(1, 1000, 1000)`

because `np.arange(1, 1000+1)` gives you a sequence of integers by default and `np.linspace(1, 1000, 1000)` does not.

The autograder will grade the function for 4 pts, and the graph will be hand-graded for 2 pts.

In [3]:

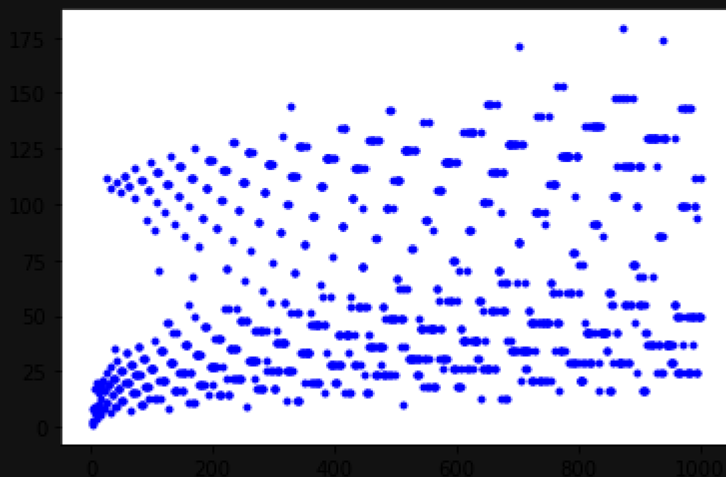
```
#Solution for Problem 2
def L(n):
    count = 1
    #n_vals = np.arange(1, 1000+1)
    #y_vals = [L(i) for i in n_vals]
```

```
    while(n != 1):
        if(n % 2 == 0):
            n /=2
        else:
            n = 1 + (3 * n)
        count += 1
    return count

n_vals = np.arange(1, 1000+1)
y_vals = [L(i) for i in n_vals]

plt.plot(n_vals, y_vals, '.b')
```

Out[3]: `[<matplotlib.lines.Line2D at 0x2bd4e9edf70>]`



Problem 3 (2 points)

(This problem is autograded.)

Write a function called `janken` that plays Rock, Paper, Scissors against your computer.

The way to do this is to assign numbers to Rock, Paper, and Scissors and use `if-elif-else` statements. Let 1 correspond to Rock, 2 correspond to Paper and 3 correspond to Scissors.

The function should accept a single integer argument that is either 1, 2, or 3; each corresponding to the 3 different hands you can play. The function must return a string that indicates who played what and who won with the following format:

> `"You played {your play}; Skynet played {computer's play}. {"You" or "Skynet"} wins!"`

For example, after writing the function, the following code using the function

for _ in range(10): # your_play = random.randint(1,3) # print(janken(your_play)) # # print() # # print(janken(4)) # print(janken(0)) # print(janken(777))

should have the following output:

You played Scissors; Skynet played Paper. You win! # You played Scissors; Skynet played Paper. You win! # You played Scissors; Skynet played Rock. Skynet wins! # You played Scissors; Skynet played Rock. Skynet wins! # You played Scissors; Skynet played Paper. You win! # You played Paper; Skynet played Rock. You win! # You played

Scissors; Skynet played Paper. You win! # You played Rock; Skynet played Paper. Skynet wins! # You played Paper; Skynet played Scissors. Skynet wins! # You played Rock; Skynet played Rock. It's a tie! # # That's not a valid play; Skynet wins! # That's not a valid play; Skynet wins! # That's not a valid play; Skynet wins!

Ignore the leading `#` and space on each line.

Note: Due to "randomness", the outcomes of your 10 games won't match the outcomes of the games in the example; but your function's output should match the **format** of the example.

The following two lines of code may be used to generate random integers between 1 and 3.

> `import random`
>
> `print(random.randint(1,3))`

Use this to program the computer to "choose" it's hands.

In [4]:
```
#Solution for Problem 3
```

Problem 4 (3 points)

(This problem is autograded.)

The sequence $\{a_n\}$ is defined recursively by the equation:

$$n(n-1)a_n = (n-1)(n-2)a_{n-1} - (n-3)a_{n-2}, \quad n > 1,\ a_0 = a_1 = 1$$

Write a function `sum_an` to compute the sum:

$$A(N) = \sum_{n=0}^{N} a_n$$

The funtion should accept a single argument N and return the computed sum up to and including N. Run and display the results of your code for $N = 1, 5, 10, 100$.

In [42]:
```
#Solution for Problem 4
def sum_an(n):
    if(n == 0):
        return 0
    elif n == 1:
        return 1
    else:
        ans = n + sum_an(n-1)
        return ans

sum_an(1), sum_an(5), sum_an(15), sum_an(100)
```

Out[42]:
```
(1, 15, 120, 5050)
```

Problem 5 (5 points)

A sequence that arises in ecology as a model for population growth is defined by the logistic difference equation

$$p_{n+1} = kp_n(1 - p_n),\ n > 0$$

where p_n measures the size of the population of the n-th generation of a single species.

An ecologist is interested in predicting the size of the population as time goes on, and asks these questions: Will it stabilize at a limiting value? Will it change in a cyclical fashion? Or will it exhibit random behavior?

(a)

(This problem is autograded.)

Write a function `p_n` to compute the next n terms of this sequence starting with an initial population $p_0 > 0$ with growth rate $k > 0$. The function should accept 3 arguments ($p_0,\ k,\ n$, in that order) and should return a numpy array containing

$$[p_0,\ p_1,\ p_2,\ \cdots,\ p_{n-1},\ p_n]$$

Note: the length of the returned array should be $n + 1$.

In [6]:

```
# Solution for Problem 5a

def p_n(p0, k, n):
    pN = 0
    pFin = 0
    for i in range(2, n + 1):
        pFin = (k * pN[i - 1] + k * pN[i - 2])
        pN.append(pFin)
    return pN
```

(b)

Calculate 30 terms of the sequence for $p_0 = \frac{1}{2}$ and for two values of k such that $1 < k < 3$. Graph each sequence on the same plot. Do the sequences appear to converge? Repeat for a different value of p_0 between 0 and 1, and graph that on the same plot. Does the limit depend on the choice of p_0? Does it depend on the choice of k? In the spirit of what p_n represents, what is happening to the population? As always, label your axes and include a legend for the plot.

In [23]:

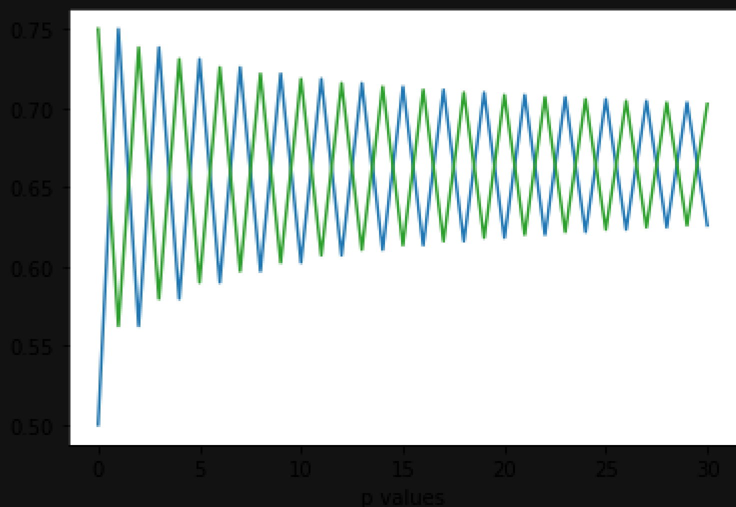
```
# Solution for Problem 5b
p0 = [0.5]
k = 3
n = 30

for i in range(30):
    pNext = k * p0[-1] * (1 - p0[-1])
    p0.append(pNext)
```

```
p1 = [0.75]

for i in range(30):
    p1Next = k * p1[-1] * (1 - p1[-1])
    p1.append(p1Next)

plt.plot(p0)
plt.plot(pNext)
plt.plot(p1)
plt.plot(p1Next)
plt.xlabel('p values')
plt.show()
```



(c)

Calculate the same number of terms of the sequence for a value of k between 3 and 3.4 and plot them. What do you notice about the behavior of the population? Again, label everything in your plot.

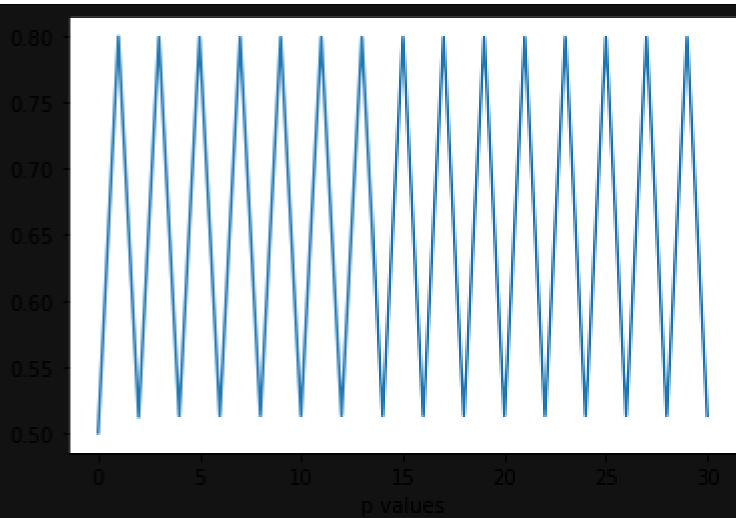
Hint: You may want to plot using stars `'*'` *or circles* `'o'` *instead of lines like you did in Problem 2. It'll be easier to see what's happening. This bit of advice applies to the rest of the problem as well.*

In [28]:

```
# Solution for Problem 5c
p3 = [0.5]
k = 3.2
n = 30

for i in range(30):
    p3Next = k * p3[-1] * (1 - p3[-1])
    p3.append(p3Next)

plt.plot(p3)
plt.plot(p3Next)
plt.xlabel('p values')
plt.show()
```



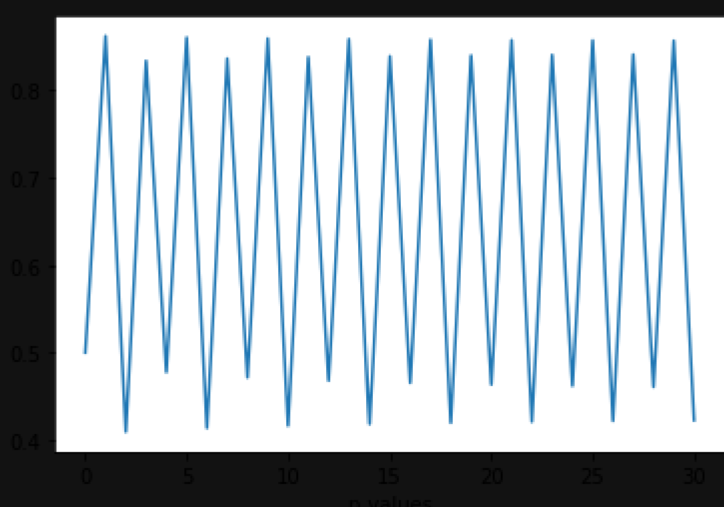
(d)

Experiment with values of k between 3.4 and 3.5 and plot the results. What happens to the population?

In [33]:
```
# Solution for Problem 5d
p4 = [0.5]
k = 3.45
n = 30

for i in range(30):
    p4Next = k * p4[-1] * (1 - p4[-1])
    p4.append(p4Next)

plt.plot(p4)
plt.plot(p4Next)
plt.xlabel('p values')
plt.show()
```

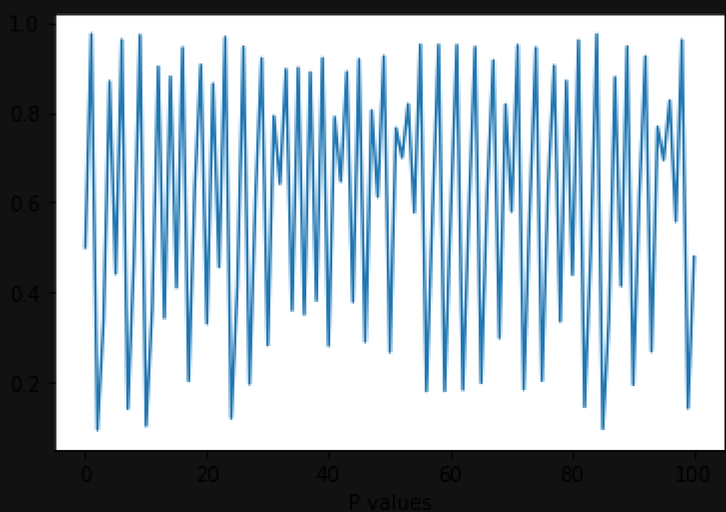


(e)

For values of k between 3.6 and 4, compute and plot at least 100 terms and comment on the behavior of the sequence. What happens if you change p_0 by 0.001? This type of behavior is called *chaotic* and is exhibited by insect populations under certain conditions.

In [34]:
```
# Solution for Problem 5e
p1 = [0.5001]
k1 = 3.9
# Calulating 30 terms for 'p1' and 'p2'
for i in range(100):
   P1next = k1 * p1[-1] * (1 - p1[-1])
   p1.append(P1next)

# Plotting graph for 30 terms
plt.plot(p1)
plt.xlabel('P values')
plt.show()
```



Problem 6 (3 points)

Here we will make plots to study the results of generalized Fibonacci sequence from Problem 1.

(a)

(This problem is autograded.)

Using the recurrence relation from **Problem 1** write a function `lucas` which generates, for $n \geq 2$ and real values a, b, p_0, p_1, the array of points

$$[p_0,\ p_1,\ p_2,\ \cdots,\ p_{n-1},\ p_n]$$

The function should accept 5 arguments: n, a, b, p_0 and p_1 in that order and it should return the array of length $n+1$ detailed above as a numpy array.

In [43]:
```
# Solution for Problem 6a
#+- 1/2 (a+-sqrt(a^2 + 4b))

def lucas(n, a, b, p0, p1):
    pN = [p0, p1]
    for i in range(2, n + 1):
```

```
        pN.append(a * pN[i - 1] + b * pN[i - 2])
    return pN
```

(b)

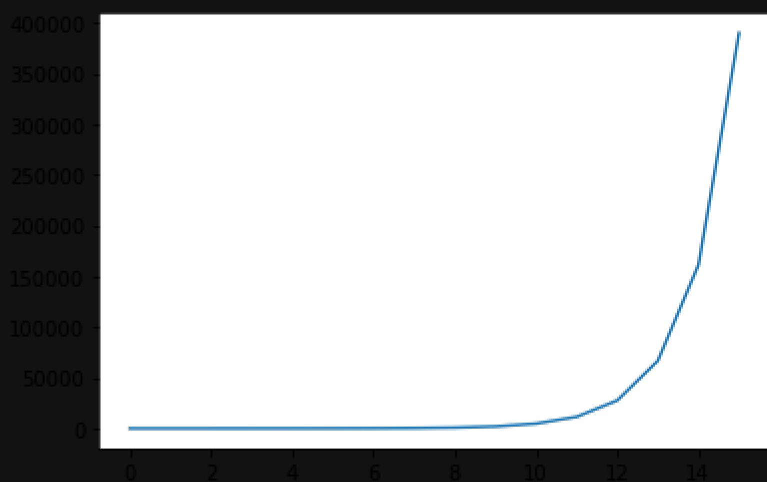
Fixing $n = 15$, $p_0 = 0$, $p_1 = 2$, $a = 2$, $b = 1$ generate a plot of p_n. Make sure axes are appropriately labeled.

In [44]:

```
# Soution for Problem 6b
n = 15
p0 = 0
p1 = 2
a = 2
b = 1

pN = lucas(n, a, b, p0, p1)

plt.plot(range(0, n + 1), pN)
plt.show()
```



(c)

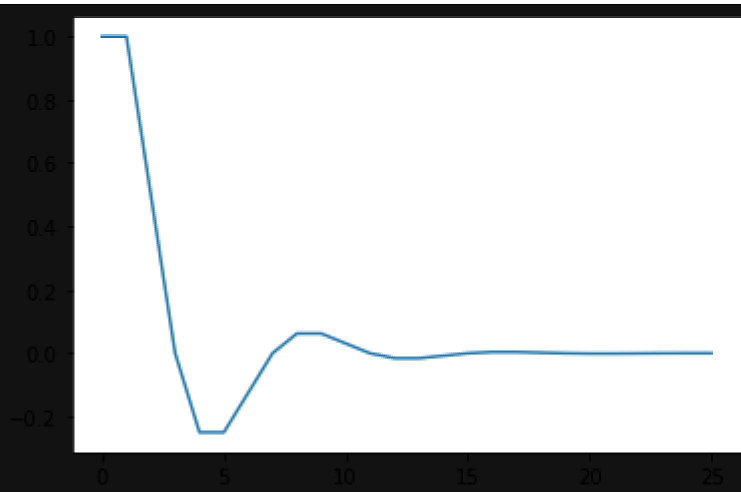
Fixing $n = 25$, $s_0 = 1$, $s_1 = 1$, $a = 1$ choose $b = -1/2$ and generate a plot of p_n. Briefly explain the results you see and how they differ from those in [b]

In [13]:

```
# Solution for Problem 6c
n = 25
p0 = 1
p1 = 1
a = 1
b = -0.5

pN = lucas(n, a, b, p0, p1)

plt.plot(range(0, n + 1), pN)
plt.show()
```



Problem 7 (3 points)

When we write

> `xvals = np.linspace(a,b,int(n)+1)`

we are generating a sequence of points $\{x_j\}$ such that

$$x_j = a + j\delta x, \ \delta x = \frac{b-a}{n}, \ j = 0, \cdots, n.$$

Thus, if I wanted to generate a sequence of points between $a = 4$ and $b = 7$ with spacing $\delta x = .3$, then I would find

$$.3 = \frac{7-4}{n}$$

so that $n = 10$. I could then generate these points via the code

> `xvals = np.linspace(4,7,10+ 1)`

Using the model, write the code which will generate the following sequences of points. In each case, make sure to show your work in $\LaTeX$ for finding n and other relevant parameters. Make sure to print out the sequence once you've figured out how to generate it.

(a)

A sequence of points between $a = -5$ and $b = 3$ with spacing $\delta x = 5^{-3}$.

In [14]:

```
#Solution for Problem 7a
a = -5
b = 3
dx = (5 ** (-3))
n = (b - a) / dx
xvals = np.linspace(a, b, int(n) + 1)
print(xvals)
```

```
[-5.    -4.992 -4.984 ...  2.984   2.992   3.    ]
```

(b)

A sequence of points between $a = 0$ and $b = 25$ with spacing $\delta x = 10^{-m}$, where m is a positive integer that a user would specify. For simplicity, you can write a function that generates the sequence but that isn't necessary.

In [35]:

```
#Solution for Problem 7b
a = 0
b = 25
m = 5
dx = (10 ** (-1 * m))
n = (b - a) / dx

xvals = np.linspace(a, b, int(n) + 1)
print(xvals)
```

```
[0.000000e+00 1.000000e-05 2.000000e-05 ... 2.499998e+01 2.499999e+01
 2.500000e+01]
```

(c)

Using the result from **(a)** and array slicing, what code would I write to find the points x_j such that $-2 \leq x_j \leq 1$? Your answer should be in the form `xvals[n1:n2]` where `n1` and `n2` are two integers you must find and `xvals` is the array you generated in part **(a)**.

In [20]:

```
#Solution for Problem 7c
a = -2
b = 1
dx = (5 ** (-3))
n1 = int((-2 - a)/dx)
n2 = int((1 - a)/dx) + 1

print(xvals[n1:n2])
```

```
[0.00e+00 1.00e-05 2.00e-05 3.00e-05 4.00e-05 5.00e-05 6.00e-05 7.00e-05
 8.00e-05 9.00e-05 1.00e-04 1.10e-04 1.20e-04 1.30e-04 1.40e-04 1.50e-04
 1.60e-04 1.70e-04 1.80e-04 1.90e-04 2.00e-04 2.10e-04 2.20e-04 2.30e-04
 2.40e-04 2.50e-04 2.60e-04 2.70e-04 2.80e-04 2.90e-04 3.00e-04 3.10e-04
 3.20e-04 3.30e-04 3.40e-04 3.50e-04 3.60e-04 3.70e-04 3.80e-04 3.90e-04
 4.00e-04 4.10e-04 4.20e-04 4.30e-04 4.40e-04 4.50e-04 4.60e-04 4.70e-04
 4.80e-04 4.90e-04 5.00e-04 5.10e-04 5.20e-04 5.30e-04 5.40e-04 5.50e-04
 5.60e-04 5.70e-04 5.80e-04 5.90e-04 6.00e-04 6.10e-04 6.20e-04 6.30e-04
 6.40e-04 6.50e-04 6.60e-04 6.70e-04 6.80e-04 6.90e-04 7.00e-04 7.10e-04
 7.20e-04 7.30e-04 7.40e-04 7.50e-04 7.60e-04 7.70e-04 7.80e-04 7.90e-04
 8.00e-04 8.10e-04 8.20e-04 8.30e-04 8.40e-04 8.50e-04 8.60e-04 8.70e-04
 8.80e-04 8.90e-04 9.00e-04 9.10e-04 9.20e-04 9.30e-04 9.40e-04 9.50e-04
 9.60e-04 9.70e-04 9.80e-04 9.90e-04 1.00e-03 1.01e-03 1.02e-03 1.03e-03
 1.04e-03 1.05e-03 1.06e-03 1.07e-03 1.08e-03 1.09e-03 1.10e-03 1.11e-03
 1.12e-03 1.13e-03 1.14e-03 1.15e-03 1.16e-03 1.17e-03 1.18e-03 1.19e-03
 1.20e-03 1.21e-03 1.22e-03 1.23e-03 1.24e-03 1.25e-03 1.26e-03 1.27e-03
 1.28e-03 1.29e-03 1.30e-03 1.31e-03 1.32e-03 1.33e-03 1.34e-03 1.35e-03
 1.36e-03 1.37e-03 1.38e-03 1.39e-03 1.40e-03 1.41e-03 1.42e-03 1.43e-03
 1.44e-03 1.45e-03 1.46e-03 1.47e-03 1.48e-03 1.49e-03 1.50e-03 1.51e-03
 1.52e-03 1.53e-03 1.54e-03 1.55e-03 1.56e-03 1.57e-03 1.58e-03 1.59e-03
 1.60e-03 1.61e-03 1.62e-03 1.63e-03 1.64e-03 1.65e-03 1.66e-03 1.67e-03
 1.68e-03 1.69e-03 1.70e-03 1.71e-03 1.72e-03 1.73e-03 1.74e-03 1.75e-03
```

```
 1.76e-03 1.77e-03 1.78e-03 1.79e-03 1.80e-03 1.81e-03 1.82e-03 1.83e-03
 1.84e-03 1.85e-03 1.86e-03 1.87e-03 1.88e-03 1.89e-03 1.90e-03 1.91e-03
 1.92e-03 1.93e-03 1.94e-03 1.95e-03 1.96e-03 1.97e-03 1.98e-03 1.99e-03
 2.00e-03 2.01e-03 2.02e-03 2.03e-03 2.04e-03 2.05e-03 2.06e-03 2.07e-03
 2.08e-03 2.09e-03 2.10e-03 2.11e-03 2.12e-03 2.13e-03 2.14e-03 2.15e-03
 2.16e-03 2.17e-03 2.18e-03 2.19e-03 2.20e-03 2.21e-03 2.22e-03 2.23e-03
 2.24e-03 2.25e-03 2.26e-03 2.27e-03 2.28e-03 2.29e-03 2.30e-03 2.31e-03
 2.32e-03 2.33e-03 2.34e-03 2.35e-03 2.36e-03 2.37e-03 2.38e-03 2.39e-03
 2.40e-03 2.41e-03 2.42e-03 2.43e-03 2.44e-03 2.45e-03 2.46e-03 2.47e-03
 2.48e-03 2.49e-03 2.50e-03 2.51e-03 2.52e-03 2.53e-03 2.54e-03 2.55e-03
 2.56e-03 2.57e-03 2.58e-03 2.59e-03 2.60e-03 2.61e-03 2.62e-03 2.63e-03
 2.64e-03 2.65e-03 2.66e-03 2.67e-03 2.68e-03 2.69e-03 2.70e-03 2.71e-03
 2.72e-03 2.73e-03 2.74e-03 2.75e-03 2.76e-03 2.77e-03 2.78e-03 2.79e-03
 2.80e-03 2.81e-03 2.82e-03 2.83e-03 2.84e-03 2.85e-03 2.86e-03 2.87e-03
 2.88e-03 2.89e-03 2.90e-03 2.91e-03 2.92e-03 2.93e-03 2.94e-03 2.95e-03
 2.96e-03 2.97e-03 2.98e-03 2.99e-03 3.00e-03 3.01e-03 3.02e-03 3.03e-03
 3.04e-03 3.05e-03 3.06e-03 3.07e-03 3.08e-03 3.09e-03 3.10e-03 3.11e-03
 3.12e-03 3.13e-03 3.14e-03 3.15e-03 3.16e-03 3.17e-03 3.18e-03 3.19e-03
 3.20e-03 3.21e-03 3.22e-03 3.23e-03 3.24e-03 3.25e-03 3.26e-03 3.27e-03
 3.28e-03 3.29e-03 3.30e-03 3.31e-03 3.32e-03 3.33e-03 3.34e-03 3.35e-03
 3.36e-03 3.37e-03 3.38e-03 3.39e-03 3.40e-03 3.41e-03 3.42e-03 3.43e-03
 3.44e-03 3.45e-03 3.46e-03 3.47e-03 3.48e-03 3.49e-03 3.50e-03 3.51e-03
 3.52e-03 3.53e-03 3.54e-03 3.55e-03 3.56e-03 3.57e-03 3.58e-03 3.59e-03
 3.60e-03 3.61e-03 3.62e-03 3.63e-03 3.64e-03 3.65e-03 3.66e-03 3.67e-03
 3.68e-03 3.69e-03 3.70e-03 3.71e-03 3.72e-03 3.73e-03 3.74e-03 3.75e-03]
```

(d)

Using the result from **(b)** and array slicing, what code would I write to find the points x_j such that $4 \leq x_j \leq 13$? Your answer should be in the form `xvals[n1:n2]` where `n1` and `n2` are two integers you must find, though they will be in terms of m. Again you may write a function of m, but it isn't necessary.

In [21]:

```python
#Solution for Problem 7d
a = 4
b = 13
dx = (5 ** (-3))
n1 = int((-2 - a)/dx)
n2 = int((1 - a)/dx) + 1

print(xvals[n1:n2])
```

```
[24.99251 24.99252 24.99253 24.99254 24.99255 24.99256 24.99257 24.99258
 24.99259 24.9926  24.99261 24.99262 24.99263 24.99264 24.99265 24.99266
 24.99267 24.99268 24.99269 24.9927  24.99271 24.99272 24.99273 24.99274
 24.99275 24.99276 24.99277 24.99278 24.99279 24.9928  24.99281 24.99282
 24.99283 24.99284 24.99285 24.99286 24.99287 24.99288 24.99289 24.9929
 24.99291 24.99292 24.99293 24.99294 24.99295 24.99296 24.99297 24.99298
 24.99299 24.993   24.99301 24.99302 24.99303 24.99304 24.99305 24.99306
 24.99307 24.99308 24.99309 24.9931  24.99311 24.99312 24.99313 24.99314
 24.99315 24.99316 24.99317 24.99318 24.99319 24.9932  24.99321 24.99322
 24.99323 24.99324 24.99325 24.99326 24.99327 24.99328 24.99329 24.9933
 24.99331 24.99332 24.99333 24.99334 24.99335 24.99336 24.99337 24.99338
 24.99339 24.9934  24.99341 24.99342 24.99343 24.99344 24.99345 24.99346
 24.99347 24.99348 24.99349 24.9935  24.99351 24.99352 24.99353 24.99354
 24.99355 24.99356 24.99357 24.99358 24.99359 24.9936  24.99361 24.99362
 24.99363 24.99364 24.99365 24.99366 24.99367 24.99368 24.99369 24.9937
 24.99371 24.99372 24.99373 24.99374 24.99375 24.99376 24.99377 24.99378
```

```
 24.99379 24.9938  24.99381 24.99382 24.99383 24.99384 24.99385 24.99386
 24.99387 24.99388 24.99389 24.9939  24.99391 24.99392 24.99393 24.99394
 24.99395 24.99396 24.99397 24.99398 24.99399 24.994   24.99401 24.99402
 24.99403 24.99404 24.99405 24.99406 24.99407 24.99408 24.99409 24.9941
 24.99411 24.99412 24.99413 24.99414 24.99415 24.99416 24.99417 24.99418
 24.99419 24.9942  24.99421 24.99422 24.99423 24.99424 24.99425 24.99426
 24.99427 24.99428 24.99429 24.9943  24.99431 24.99432 24.99433 24.99434
 24.99435 24.99436 24.99437 24.99438 24.99439 24.9944  24.99441 24.99442
 24.99443 24.99444 24.99445 24.99446 24.99447 24.99448 24.99449 24.9945
 24.99451 24.99452 24.99453 24.99454 24.99455 24.99456 24.99457 24.99458
 24.99459 24.9946  24.99461 24.99462 24.99463 24.99464 24.99465 24.99466
 24.99467 24.99468 24.99469 24.9947  24.99471 24.99472 24.99473 24.99474
 24.99475 24.99476 24.99477 24.99478 24.99479 24.9948  24.99481 24.99482
 24.99483 24.99484 24.99485 24.99486 24.99487 24.99488 24.99489 24.9949
 24.99491 24.99492 24.99493 24.99494 24.99495 24.99496 24.99497 24.99498
 24.99499 24.995   24.99501 24.99502 24.99503 24.99504 24.99505 24.99506
 24.99507 24.99508 24.99509 24.9951  24.99511 24.99512 24.99513 24.99514
 24.99515 24.99516 24.99517 24.99518 24.99519 24.9952  24.99521 24.99522
 24.99523 24.99524 24.99525 24.99526 24.99527 24.99528 24.99529 24.9953
 24.99531 24.99532 24.99533 24.99534 24.99535 24.99536 24.99537 24.99538
 24.99539 24.9954  24.99541 24.99542 24.99543 24.99544 24.99545 24.99546
 24.99547 24.99548 24.99549 24.9955  24.99551 24.99552 24.99553 24.99554
 24.99555 24.99556 24.99557 24.99558 24.99559 24.9956  24.99561 24.99562
 24.99563 24.99564 24.99565 24.99566 24.99567 24.99568 24.99569 24.9957
 24.99571 24.99572 24.99573 24.99574 24.99575 24.99576 24.99577 24.99578
 24.99579 24.9958  24.99581 24.99582 24.99583 24.99584 24.99585 24.99586
 24.99587 24.99588 24.99589 24.9959  24.99591 24.99592 24.99593 24.99594
 24.99595 24.99596 24.99597 24.99598 24.99599 24.996   24.99601 24.99602
 24.99603 24.99604 24.99605 24.99606 24.99607 24.99608 24.99609 24.9961
 24.99611 24.99612 24.99613 24.99614 24.99615 24.99616 24.99617 24.99618
 24.99619 24.9962  24.99621 24.99622 24.99623 24.99624 24.99625 24.99626]
```

In [ ]: