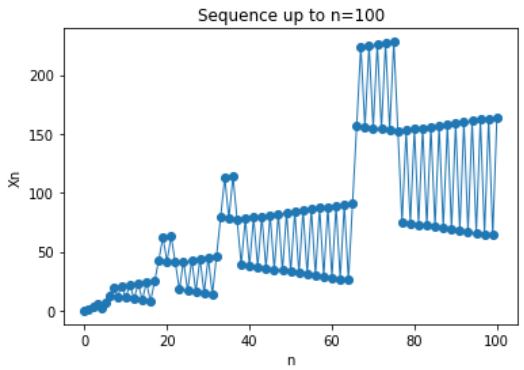
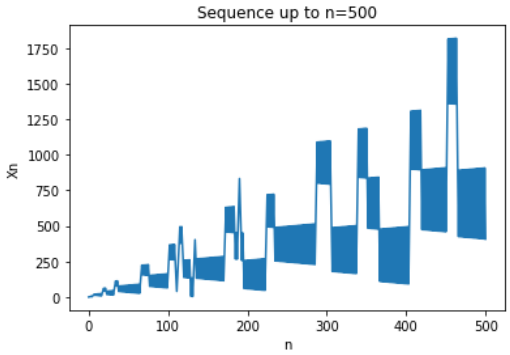
MAS1803 – Assessment 2 – Ewan Walker

Question 1 - B

As you can see for n=500, the line itself is very difficult to see. I have, therefore, also included a graph of the sequence up to n=100 to show why.

Plot





Question 1 - C

Thoughts

What I am starting with is an array 'x' containing all of the values of n for the sequence up until a maximum n.

The equation for a circle is (y-a)^2 + (x+b)^2 = r^2 *for centre (a,b) and radius r.*

This can also be rearranged to y = +-sqrt(r^2 – (x-a)^2) + b *so I will need numpy for np.sqrt.*

I want to plot along the x axis for continuity, meaning I can ignore b ===> y = +-sqrt(r^2 – (x-a)^2)

Each of the variables, in this circumstance, are as follows, starting from n=1:

- n = x[v] for some value v in range(0,len(x)) and v > 0
- r = (n-(n-1)) / 2
- a = n – r

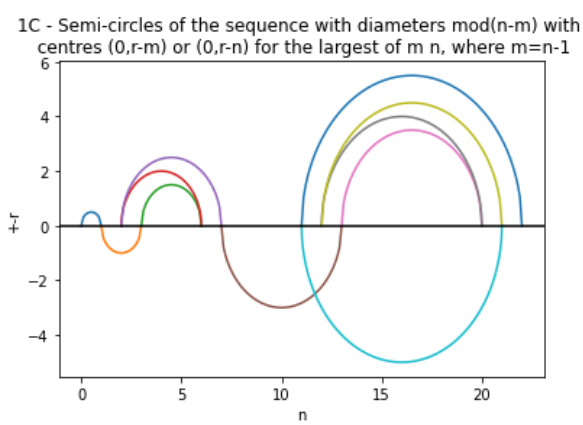
need to account for when n<m

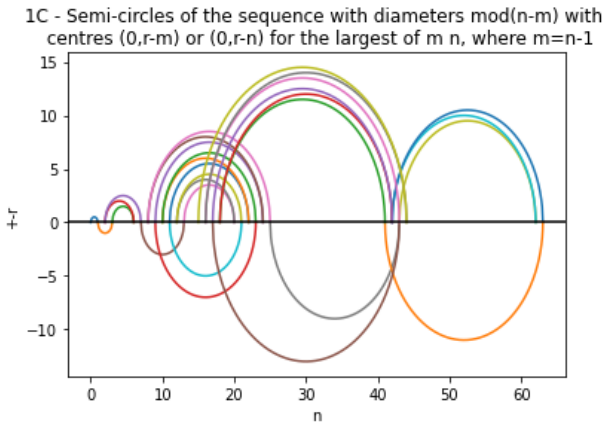
In order to obtain just a semicircle with a horizontal diameter above the axis and below the axis, take the positive and the negative of the equation respectively.

To alternate positive and negative as asked, odd element numbers of the array n are to populate an array (C) as positives and the even element numbers of the array n are to populate an array (C) as the negatives. Note the length of the array doesn't change from x to C – this is so that it is easy to plot.

What I have is an array C populated with all the semi-circle equations, correctly alternating. Lastly, all I must do is plot all the C elements using a 'for' loop to create the requested graph.

Plot





I encountered some problems with my thoughts and have addressed that in the code.

Question 1 - D

To investigate the conjecture that all positive integers will eventually be in the sequence, my thought process proceeded as follows:

1) Take n to be a large number.
2) Create a np.arange 'A' of size max(x) to account for all the integer in list x for the n.
3) Deduct x from A to create array B find integers which don't appear in x.
4) Compare the length of B for increasing size of n.
5) If the value of n/len(B) tends to decrease as n increases, then it can be concluded that eventually all integers may be in the sequence. However, if there is no tendency for the ration of n/len(B) to decrease as n increases, or if it actually increases, then it can be concluded that all positive integers definitely can't be in the sequence.

I ran into type errors. To account for this, I can create a for loop to replace step '3)' like that which produced the sequence initially – whereby I 'A.remove' rater than creating a new list B.. In doing this, I have to amend step 2) to be a list of np.arange.

My results are as follows:

n=501, len(A)=1521 n/len(A) = 0.3294

n=1001, A=3074 n/A = 0.3256

n=1501, A= 4875 n/A = 0.3079

n=2001, A= 5856 n/A = 0.3417

n=2501, A= 8825 n/A = 0.2834

n=100000, A= 546109 n/A= 0.1831

I encountered some problems with my thoughts and have addressed that in the code.

Conclusion

Since the ratio tends to decrease, then yes, eventually all numbers will be in the sequence given that chance of the sequence to span to infinity. Additionally, as the n increases, the smallest number in the sequence continues to get larger.

Question 2 – A

A cuboid has 6 sides. Each of the 3 pieces of ribbon uniquely wrap round only 2 of these sides twice. Therefore, the following set of simultaneous equations can be formulated:

2a+2b= L1

2a+2c= L2

2b+2c= L3

They can be rearranged to isolate and equation for a, b and c in terms of L1, L2 and L3.

A=(L1+L2-L3)/4

B=(L1-L2+L3)/4

C=(-L1+L2+L3)/4

Savanna

Substituting L1=36, L2=76, L3=56 into the A,B,C equations gives:

A=14, B=4, C=24

Total length of ribbon = L1 + L2 + L3 = **168**

Erin

Substituting L1=36, L2=56, L3=56 into the A,B,C equations gives:

A=9, B=9, C=19

Total length of ribbon = L1 + L2 + L3 = **148**

Since the volume of a cuboid can be given by:

V = length x width x height ===> V = A x B x C

Therefore, the volume of Savanna's present is V = 14 x 4 x 24 = **1344**

And the volume of Erin's present is V = 9 x 9 x 19 = **1539**

1344 < **1539** but **168** > **148**

In this case the present with the larger volume was wrapped with the lesser ribbon length. As a result, Erin's hypothesis was incorrect, meaning that, because Savanna disagreed with Erin, Savanna was correct (in this case).

Question 2 - D

The most efficient wrapping method would be where we can wrap the maximized volume with the minimized ribbon length. In other words, what ratio of L1:L2:L3 minimizes the length of ribbon required to wrap the present the most.

Thought process

Find the most efficient shape: maximize volume per ribbon required ('perimeters') using brute force side lengths then plot a graph to find the most efficient values.

Since the side lengths are relative to one another, it makes sense to set one side length to be a constant and vary the other two.

I want to check the volume of each N(r,s,t) and divide that by the same N(r,s,t) 'perimeters' ; then, populate an array O with the value of each.

Plot a graph of O against len(x).

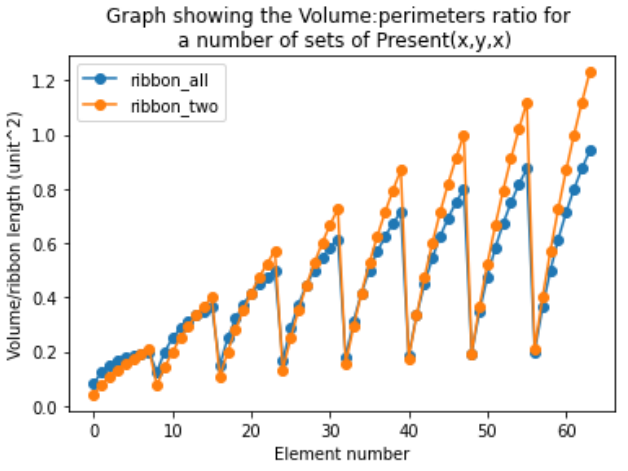
Draw a vertical line at the maximum y-axis value. Where it intersects the x-axis is the value/s of N(r,s,t) with the best ratio (the most efficient cuboid shape). This is the visual representation of a graph asked in the question.

I want to find each set of the values of r, s and t and populate an array with these sets to find the optimum shapes.

Lastly, remove the congruent cuboids from this array to isolate the most efficient unique side lengths of the cuboid whereby the ribbon required to wrap the cuboid is minimized. Modify this to account for the ribbon_all and ribbon_two to answer the question fully.

I encountered some problems with my thoughts and have addressed that in the code.

Plot



Results

Immediately I notice the following characteristics:

- The graph is increasing
- The ratio drops to a local minimum for whatever value I set ‘a’ to be
- The worst possible ratio is a cube (L1=L2=L3)

Conclusion

As the volume increases, the ratio tends to increase. This is no surprise because I have been previously taught the surface area to volume ratio decreases as volume increases. The graph very clearly depicts that where L1=L2, with some **larger L3**, is the most efficient shape (the local maxima); however the local minima represent the least efficient shape whereby L1=L2 but some **equal or smaller L3**. Ribbon_all is only more efficient than the Ribbon_two at the small volumes. Therefore, if you want to save the environment, buy people the longest Tobelrone that you can find for a Christmas present but, if you have to get something shaped as a cube, be sure that its small and is wrapped with ribbon 3 times without the bow!

Code

Q1b

```
import matplotlib.pyplot as plt


x=[0]


for n in range (1,501):
   if  ((x[n-1] - n) in x) != True and (x[n-1] - n) > 0:
      x.append(x[n-1] - n)
   else:
      x.append(x[n-1] + n)


plt.plot(range(0,501),x)
plt.xlabel("n")
plt.ylabel("Xn")
plt.title("Sequence up to n=500")
```

Q1c

```
b=0


for h in range (1,len(x)):
j = round(h/2)      #half of the element number


C=[]
n=x[h]
m=x[h-1]


if n > m:        #Setting the centre from largest of 2 values
r=(n-m)/2
a=n-r
P=np.linspace(m,n,(n-m)*10)
else:
r=(m-n)/2
```

```
a=m-r

P=np.linspace(n,m,(m-n)*100)


for s in range(0,len(P)):

C.append(np.sqrt(pow(r,2)-pow((P[s]-a),2))+b)


if j%2 == 0:

plt.plot(P,C)


else:

M=[]

for t in range(0,len(C)):

M.append(C[t]*(-1))

plt.plot(P,M)




plt.plot(len(x),0)

plt.title("1C - Semi-circles of the sequence with diameters mod(n-m) with \n centres (0,r-m) or (0,r-n) for the largest of m n, where m=n-1")

plt.xlabel("n")

plt.ylabel("+-r")

plt.axhline(0, color="black")
```

Q1e

```
import numpy as np


x=[0]

B=[]



def Q1(p):


  for n in range (1,p):

    if  ((x[n-1] - n) in x) != True and (x[n-1] - n) > 0:

      x.append(x[n-1] - n)

    else:

      x.append(x[n-1] + n)


  #2)
```

```
    A=list(np.arange(0,max(x)))


    #3)
    for z in range (0,p):
        if  z in x != True:
            A.remove(z)


    for i in range(0,len(A)):
        B.append(A[i])


p=132


Q1(p)
```

Q2

```
import matplotlib.pyplot as plt


#2B


class Cuboid:


    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z


    def vol(self):
        return self.x * self.y * self.z


    def surface_area(self):
        return 2*(self.x*self.y + self.x*self.z + self.y*self.z)


    def perimeters(self):
        return sorted([2*(self.x+self.y), 2*(self.x+self.z), 2*(self.y+self.z)])


c= Cuboid(1,2,3)
```

```
#-------------------------------------------------------------

#2C


#Bow ribbon length

B=16


class Present(Cuboid):

colour="purple"


def __init__(self, x, y, z, colour):

super().__init__(x, y, z)

self.colour = colour


def ribbon_all(self):

return sum(super().perimeters())


def ribbon_two(self):

return sum((super().perimeters())[:2],B)


def __repr__(self):

return "{} cuboid present with dimensions 1cm x 2cm x 3cm".format(self.colour)



#p = Present(1,2,3,"blue")


#-------------------------------------------------------------

#2D


L=[]

x=1

a=10


#Setting an array for [L1,L2]

for b in range(0,a):

p=[x]


p.append(b+1)

L.append(p)
```

```
#Setting an array for [L1,L2,L3]
N=[]


def sol(m):
    v=m
    for c in range(0,a):
        j=[v]
        n=[]

        k=j[0]
        l=k[0]
        m=k[1]

        n.append(l)
        n.append(m)
        n.append(c+1)
        N.append(n)

    for b in range(0,a):
        m=L[b]
        sol(m)

#Setting an array of ratios
Oa=[]
Ot=[]

for a in range(0,len(N)):
    j=N[a]

    k=j[0]
    l=j[1]
    m=j[2]

    P = Present(k,l,m,"blue")
    V = P.vol()

    Ra = P.ribbon_all()
```

```
Rt = P.ribbon_two()


oa= V / Ra

ot= V / Rt


Oa.append(oa)

Ot.append(ot)


#Plotting

X=range(0,len(Oa))


plt.plot(X,Oa,marker="o")

plt.plot(X,Ot,marker="o")

plt.xlabel("Element number")

plt.ylabel("Volume/ribbon length (unit^2)")

plt.legend(["ribbon_all","ribbon_two"])

plt.title("Graph showing the Volume:perimeters ratio for \n a number of sets of Present(x,y,x)")
```