Numerical Estimation of Areas

The Monte-Carlo simulation approach

Eden Carbonell
Advisor: Laurent Loosveldt

May 30, 2022

Contents

1 Introduction

The aim of this project is to determine the ency of the Monte Carlo Method, the ”hit or miss method”, a computational algorithm reliant on random sampling, to estimate an area contained in a particular domain. The computational procedures will be on the scripting language python and its corresponding libraries. Our aim is to compute the areas of regular n-gons, a convex hulls, and volumes of spheres. We start by explaining Convex Hull and the QuickHull algorithm, and study its effectiveness for computing areas of polyogons. We proceed by estimating the number π given the curve $x^2 + y^2 = 1$, furthermore we will compute the volume of hyper-spheres by computing the euclidean norm in higher dimensions.

2 Monte-Carlo Method and Randomness

The Monte-Carlo methods are computational algorithms that rely on random sapling to compute a numerical result. Monte-Carlo methods vary but usually follow a particular pattern:

1. Define a domain of possible inputs
2. Generate inputs randomly from a probability distribution over the domain.
3. Performing a deterministic computation on the inputs.
4. Aggregate the results

Absolute Randomness is not necessary in Monte-Carlo simulations, in the majority of cases pseudo-random sequences are sufficient to run successful simulations, in this particular experiment, we will make use of numpy random sampling, numpy.random() is a python function that produces pseudo random numbers through what is known as BitGenerators and Generators. The function has a long period of 2**19937-1, satisfying the Sawilowsky constraint for high quality Monte-Carlo simulation.

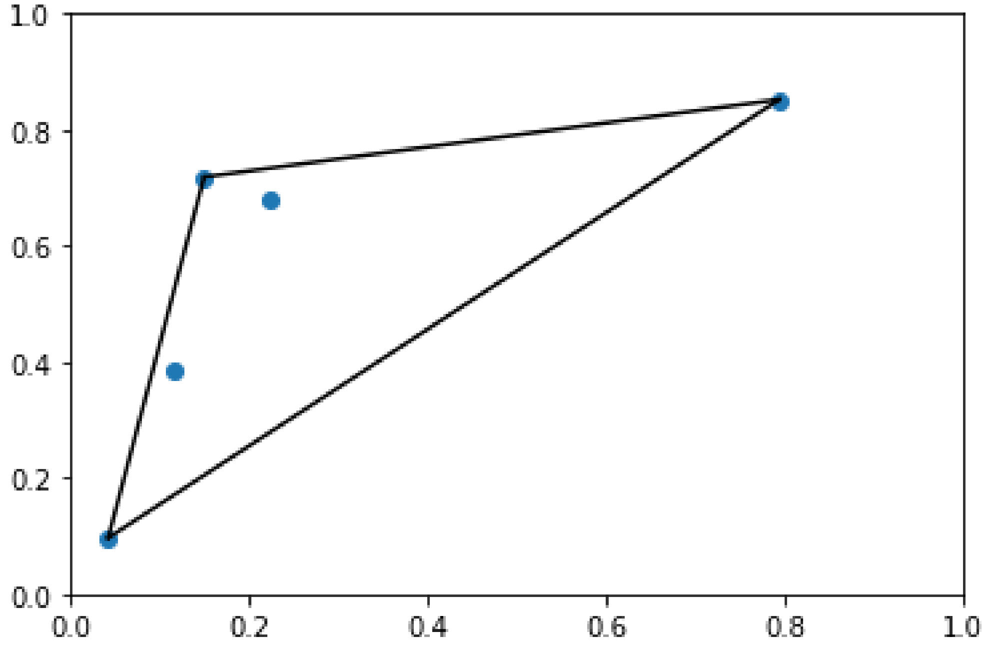
3 Convex Hull

Definition 3.1. *Given a set of points X in a Euclidean Space, the convex hull of X is defined as the intersection of all convex sets containing X.*

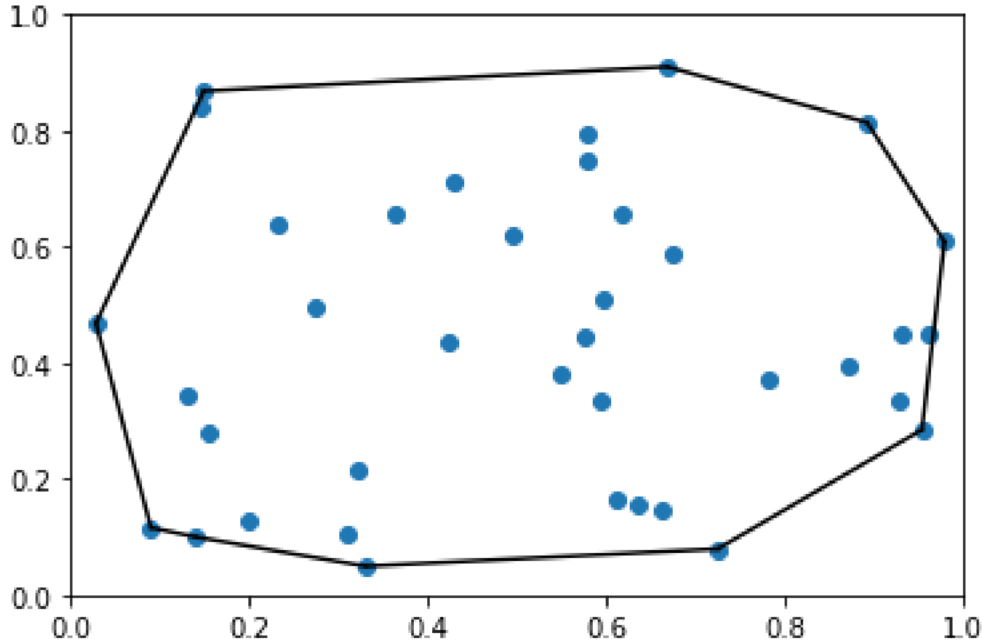
Numerous algorithms are devoted to the construction of convex hulls, Graham scan, output-sensitive algorithms such as Chan’s algorithm and the Kirkpatrick-Seidel algorithm.

In this particular project we are will make use the Scypy library and its ConvexHull function, that is computed using the Qhull library in c++. Lets briefly explain the used algorithm to compute convex hulls, the **Quickhull algorithm**.

3.1 Examples of Convex Hulls



A convex hull given a set of 5 random points



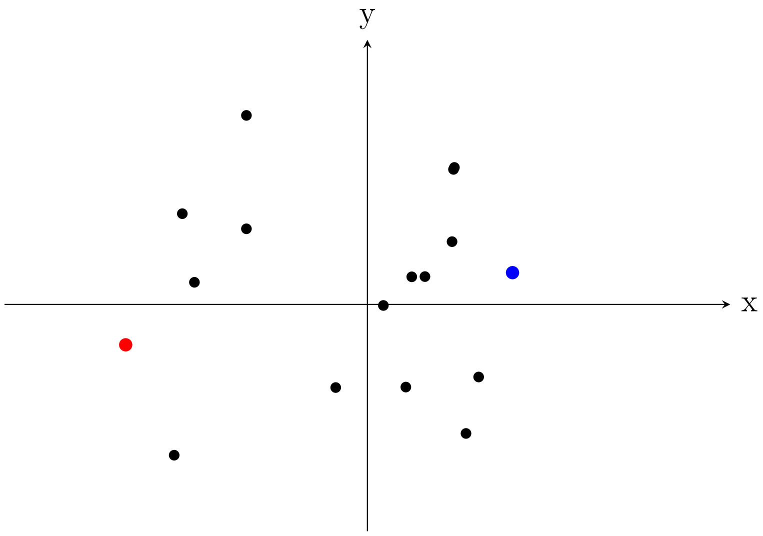
A convex hull given a set of 38 random points

3.2 Quickhull Algorithm

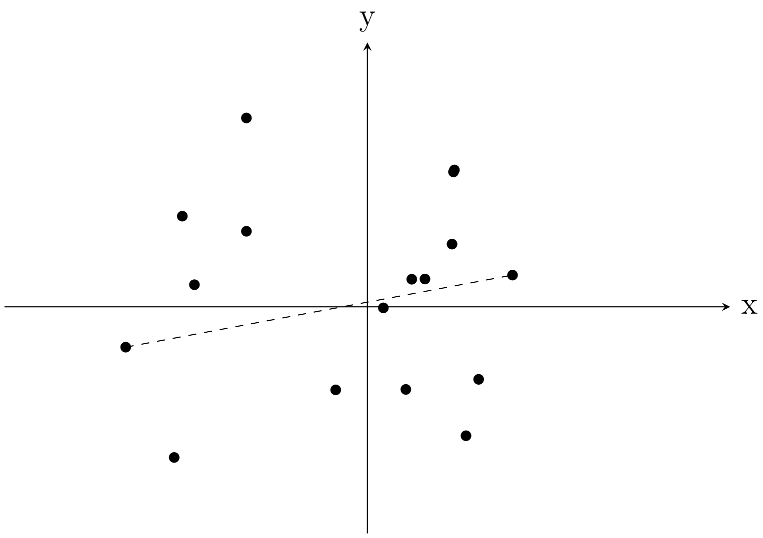
Algorithm 1 Quickhull Algorithm in two dimensions

procedure QUICKHULL(N) ▷ Obtain Convex Hull given a set of N random points in $\mathbb{R}^2$.
    $a \leftarrow min_{x\ axis} N$
    $b \leftarrow max_{x\ axis} N$
    $Line_0 \leftarrow \overline{ab}$ ▷ The control line
    loop: ▷ Repeat until there are no outter points left
    $c \leftarrow$ point at maximum distance from Line
    if $p \in \triangle abc$ **then** ignore p ▷ Search for points facing outwards the control line
    $Line_1 \leftarrow \overline{ac}$ ▷ The two new lines become control lines
    $Line_2 \leftarrow \overline{bc}$

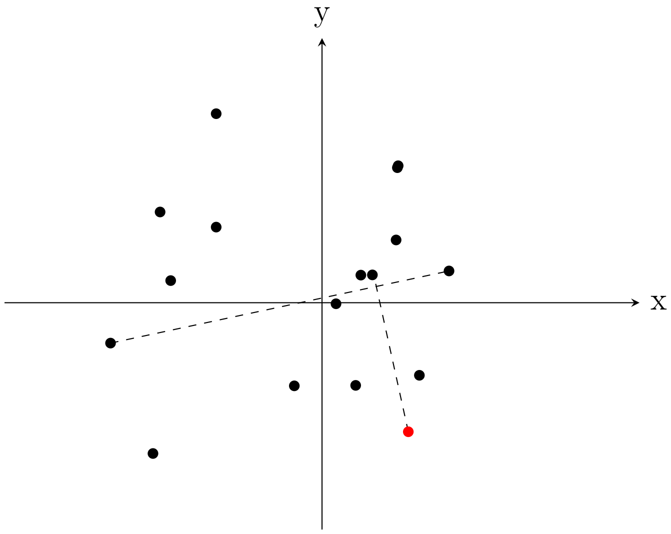
Given a set of points in $\mathbb{R}^2$, we find the points on the leftmost and rightmost side of the x axis.

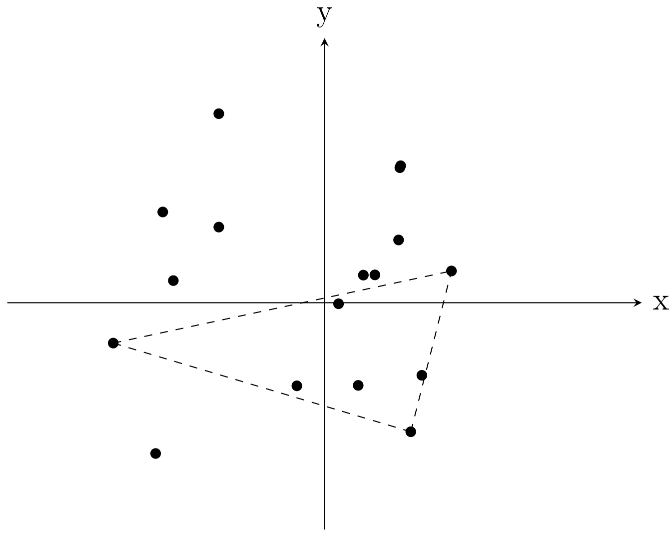


Following the divide and conquer approach, we draw a line between the two points. We proceed either below or over the line.

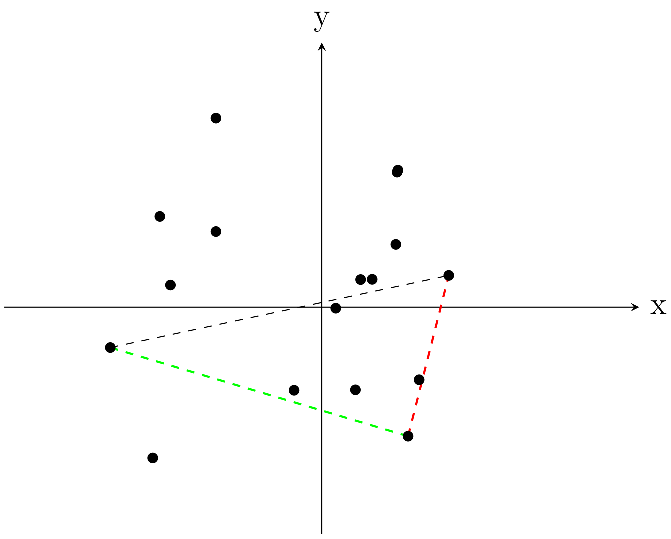


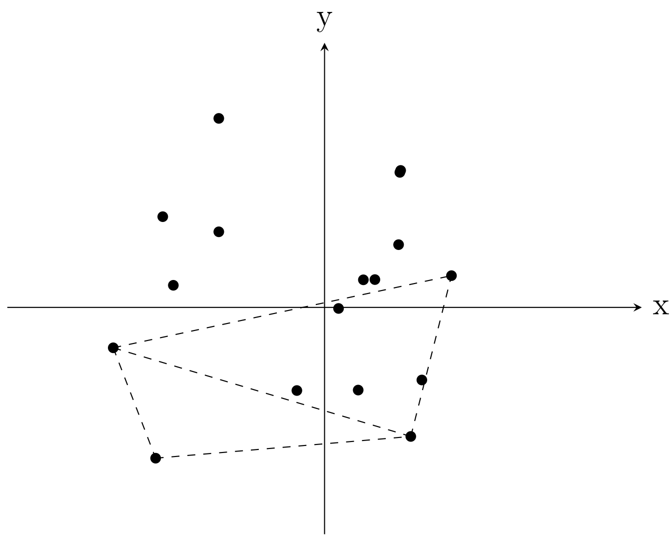
Once the line divides the set of points in two, we find the outward point with the furthest distance from the line, again, either below or over the line, we proceed to create a triangle with the end points from the previous line and the new found point.



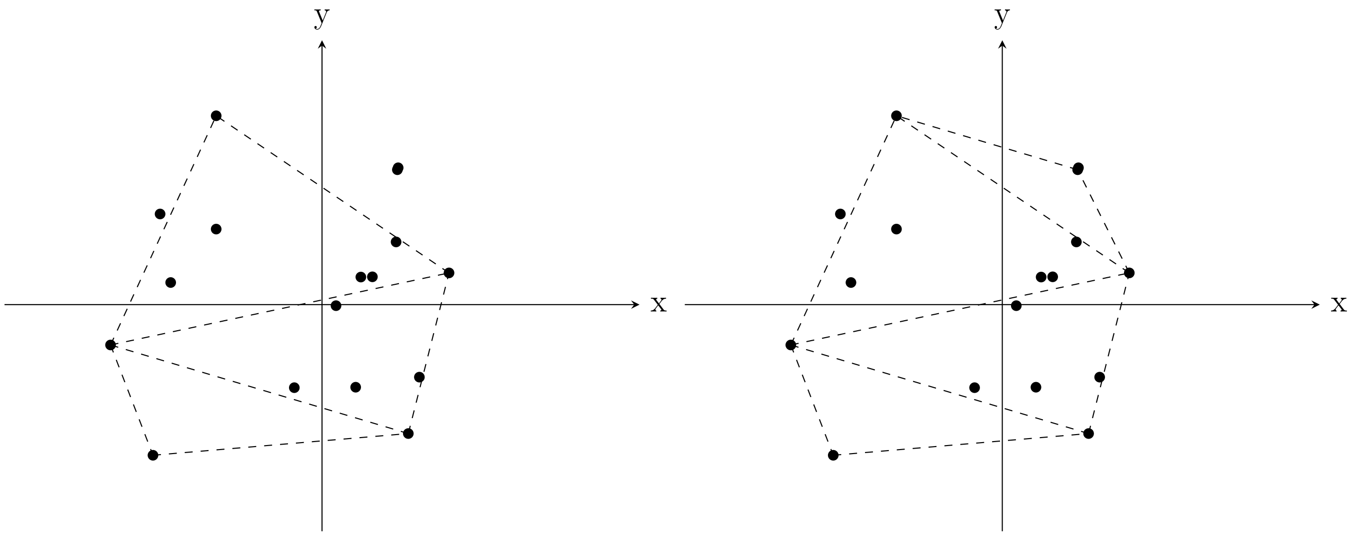


When a triangle is formed, we repeat the process with each of its edges. If a line has not points facing outward, it is skipped. For instance, here the red line is skipped.

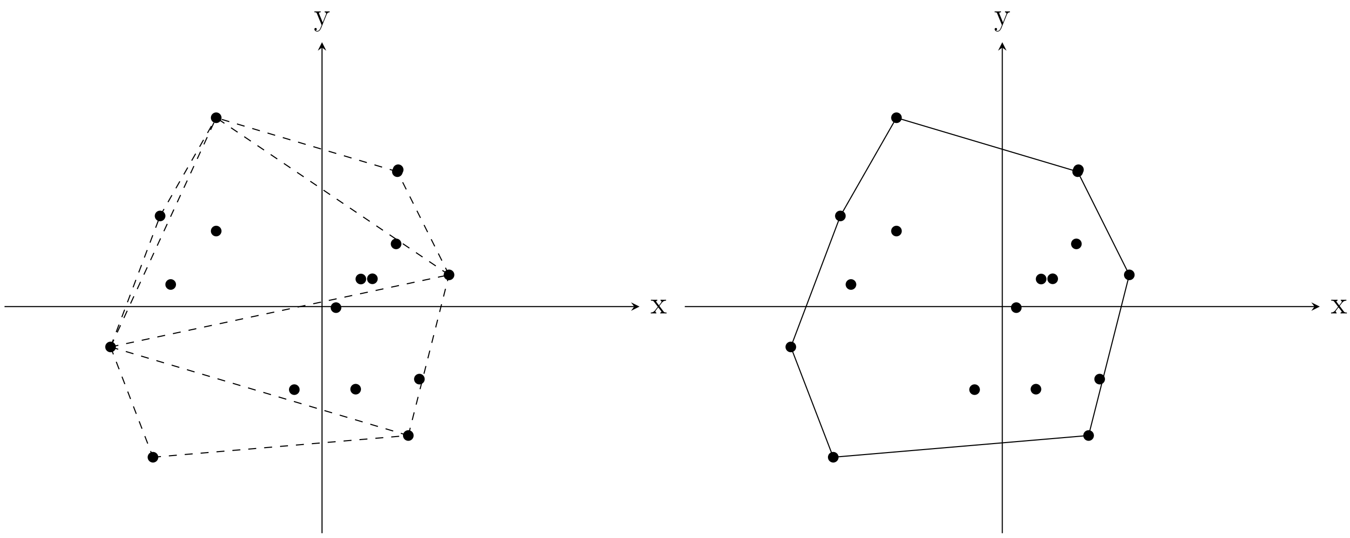




If there are no more points we proceed with the same process in the opposite direction from the initial line.



Finally, once all points have been enveloped, the outer edges of the polygon from the convex hull.



The quick hull algorithm has been proved through smoothed analysis to have a time complexity of $\mathcal{O}\left(\sqrt{\log(n)}\right)$.

4 Numerical Estimation of Convex Hull Areas

Once we have the definition of a convex hull we can compute its area through the Monte-Carlo method. The approach for estimation of areas follows the same process as other Monte-Carlo methods.

1. Define a domain of possible inputs, in this particular case we will define $[-1.5, 1.5]^2$, $[0, 1]^2$ amongst others.

2. Generate inputs randomly from a probability distribution over the domain.

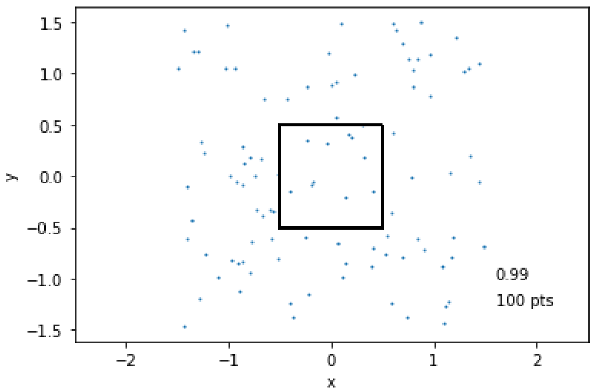
3. Performing a deterministic computation on the inputs. The resulting computation will be dependent on the domain we are operating over:

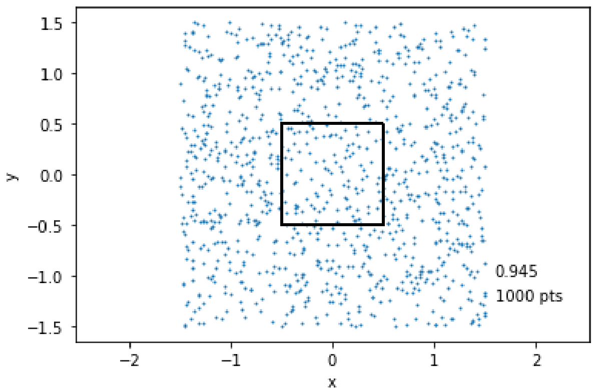
$$\left(\frac{\text{inputs inside the Convex hull}}{\text{total inputs over the domain}}\right) \times \text{the total area of the domain}$$

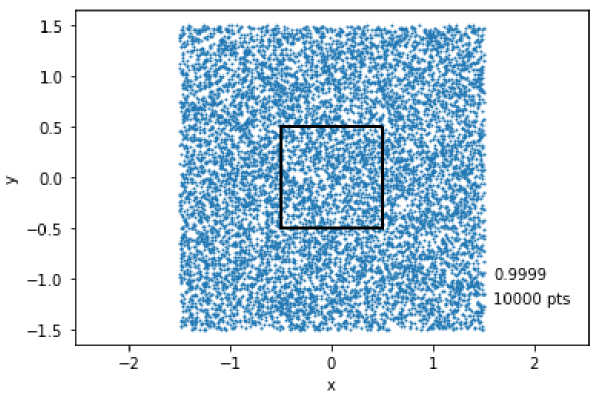
4. Aggregate the results

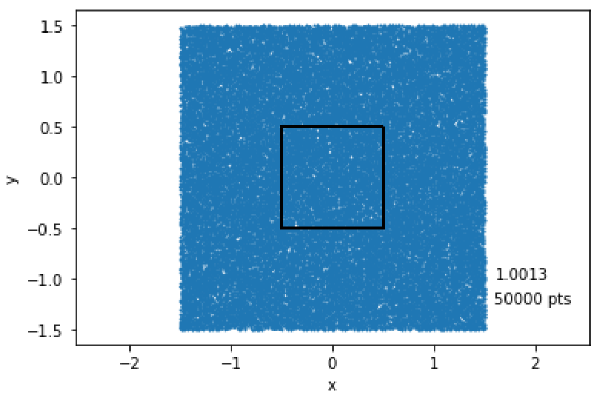
4.1 Example

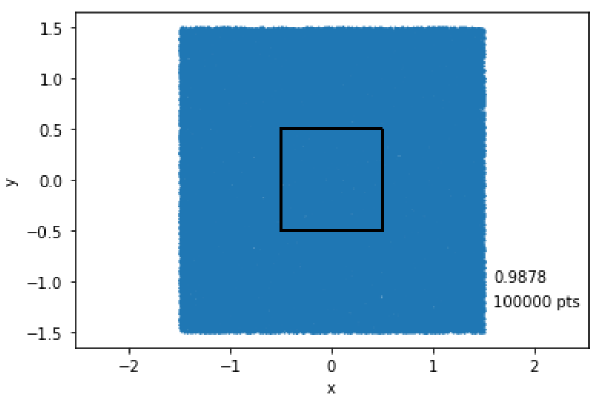
Given the domain $[-1.5, 1.5]^2 \subset \mathbb{R}^2$, we create a convex hull, in this particular case a square of side 1, which we know how to calculate the area, in this case 1. By adding points we observe that the approximated area varies and becomes closer the the known value.

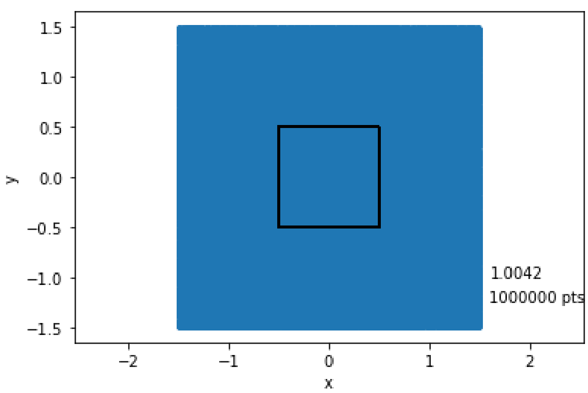

0.99
100 pts
x
y


0.945
1000 pts
x
y

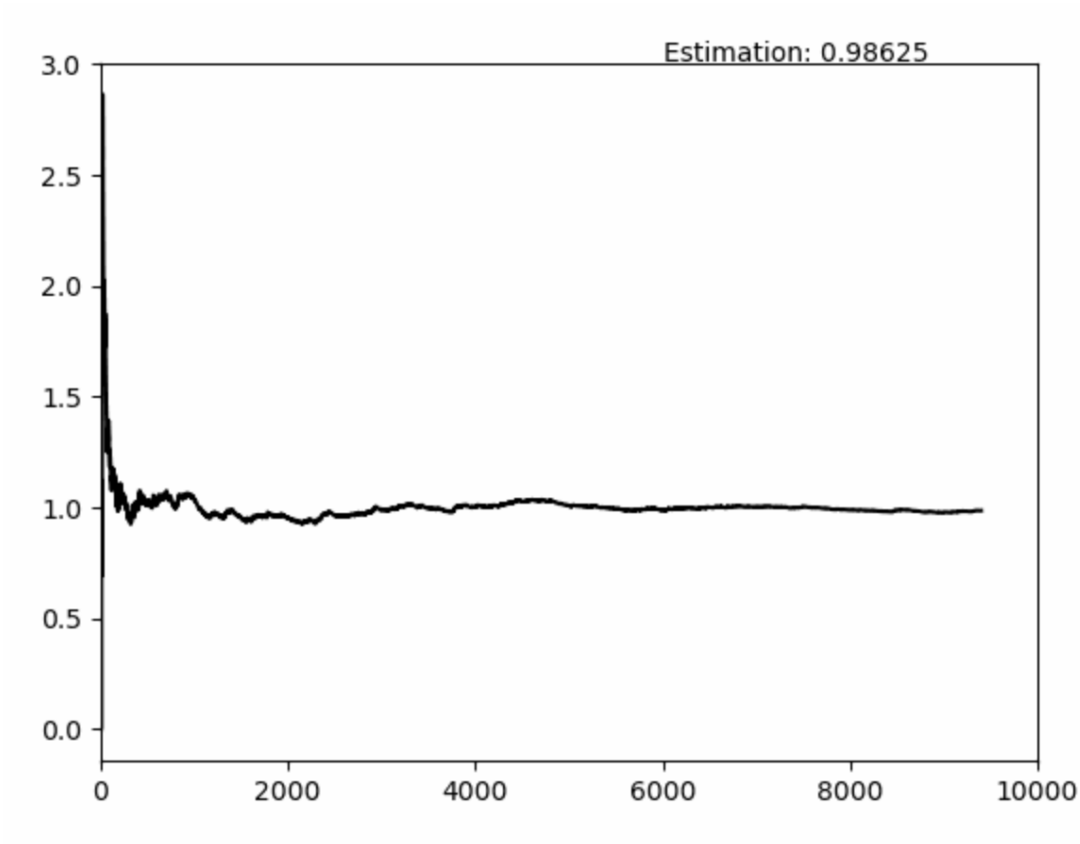

0.9999
10000 pts
x
y


1.0013
50000 pts
x
y


0.9878
100000 pts
x
y


1.0042
1000000 pts
x
y

4.2 Convergence



Square Approximation GIF

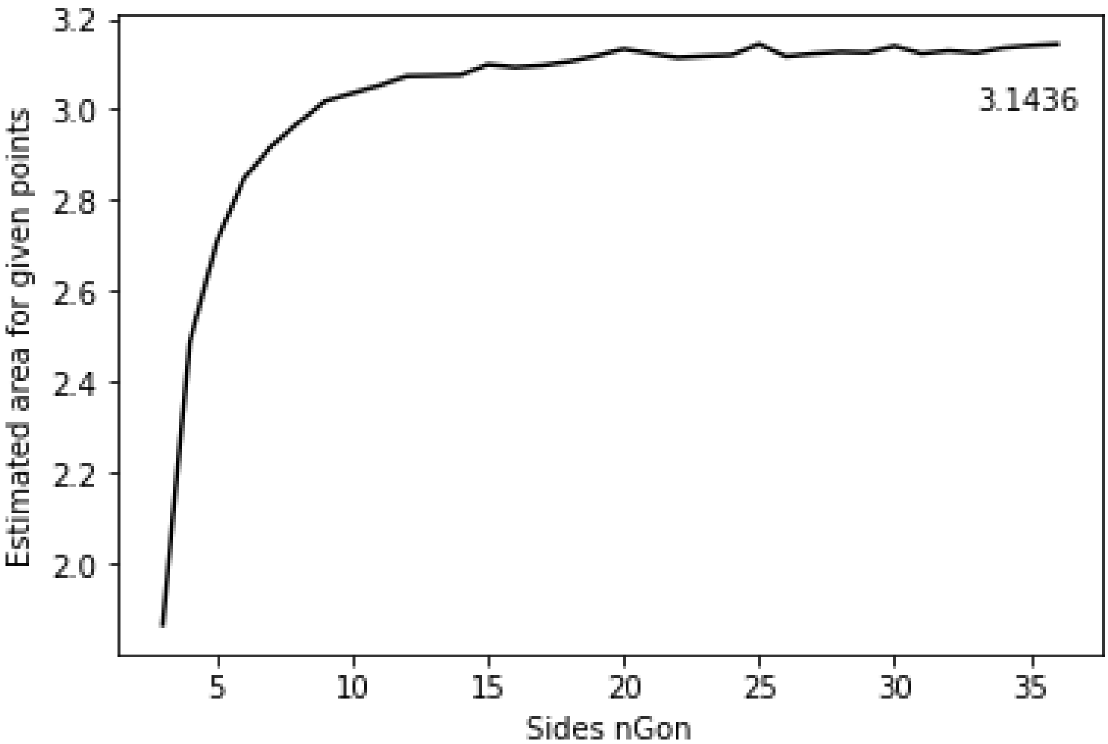
Given approximately 2000 points we observe that the approximation stabilizes.

5 Estimation of Area regular n-Gon

Lets visually represent a 3, 4, 5, 7, 15, and 31-gon. We will use python matplotlib to vusually represent the random distribution of 10000 points and the convex hull, in this case the n-gons over $[-1.5, 1.5]^2$.

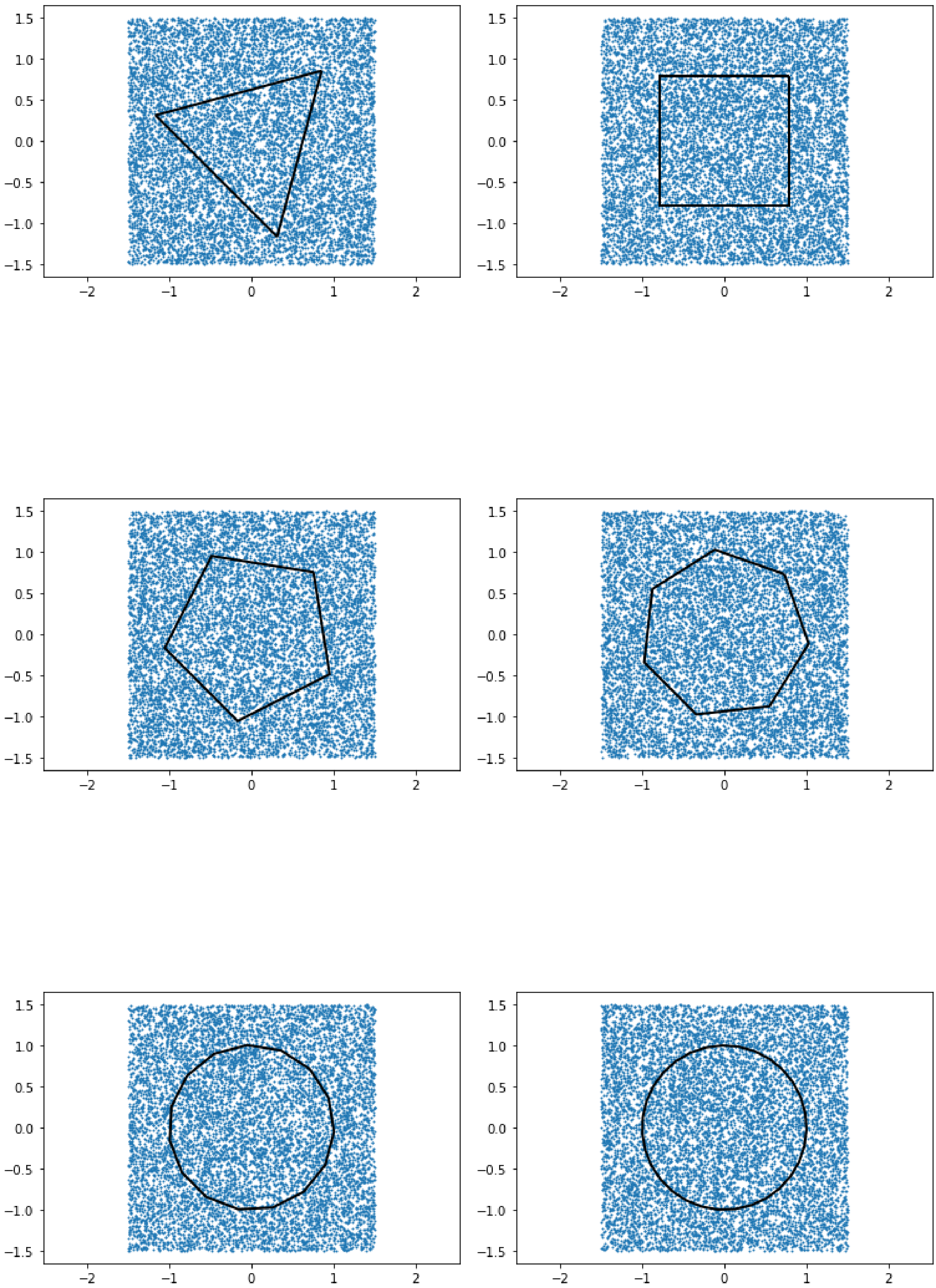
5.1 Method of Exhaustion

We observe through the method of exhaustion that as n increases to a integer large enough, the area of the n-gon converges towards the area of a circle.



Exhaustion Convergence

Due to lack of time and measure theory experience, we will not prove the exhaustion method convergence through probabilistic means, nonetheless the author views this experiment as potentially academically instructive.


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2

6 Estimation of Number π

Through the Monte Carlo methods we are going to approximate the value of pi which is know to be $3.1415926535\cdots$. In order to do so we will explore how the area varies as the number of points increases and observe if there is convergence. Now lets proceed to estimate the area of a circle of radius one, which therefore will have a theoretical area of π.

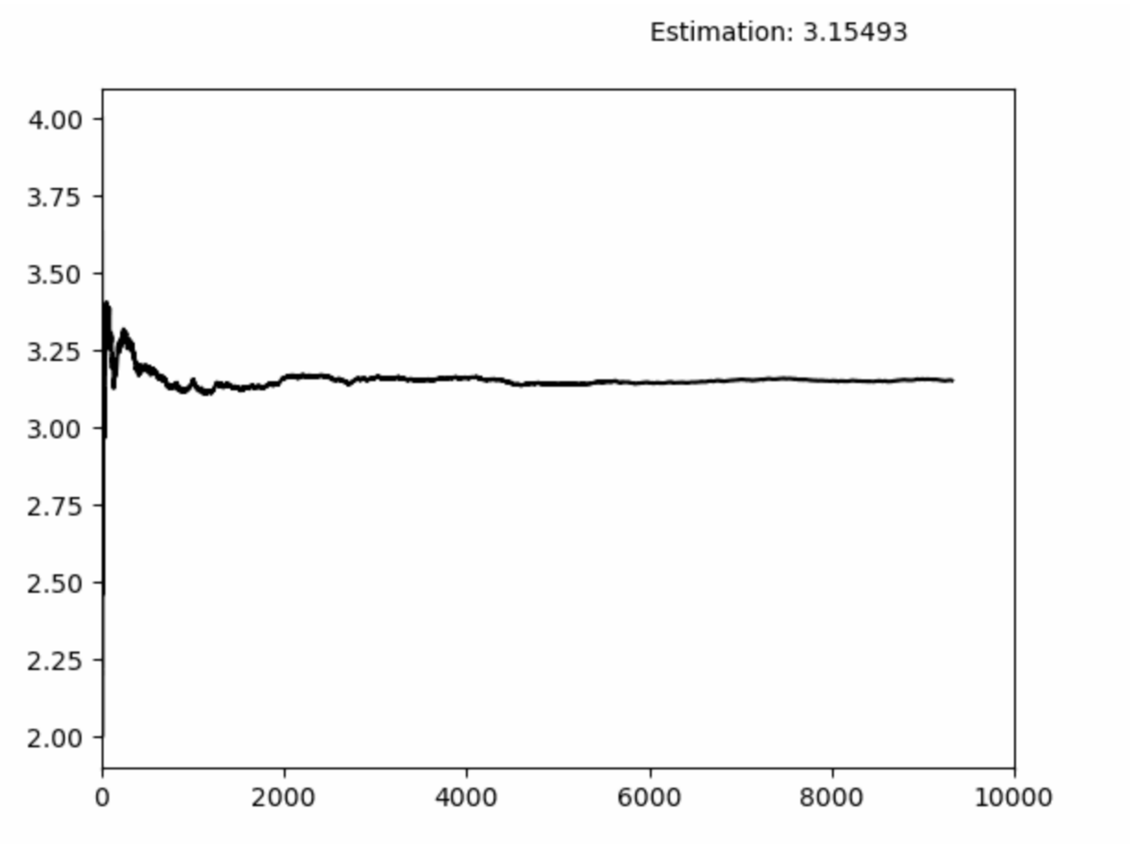
In order to do so we will find all points on a domain $[-1,1]^2 \subset \mathbb{R}^2$ that have an euclidean norm smaller than 1, ie. points $p_j \in \mathbb{R}^2$ such that

$$||p_j||^2 = \sum_{i=0}^{2} p_{j,i}^2 \leq 1. \tag{1}$$

, compute the ratio against the total amount of points on the domain and store it. Then we will plot the ratio against the total points on domain for each iteration.

Lets start by defining a function to determine if a point is of euclidean norm smaller than 1. We will use it to determine if a randomly generated point on $[-1,1]^2 \subset \mathbb{R}^2$ is inside the circle of radius one or not.

Once the euclidean norm function is defined, we iterate over a random sample of points to compute the estimated numerical value of π at any given iterations.



Pi Aproximation GIF

Given enough iterations, we observe that the numerical estimation converges towards 3.14.

6.1 Monte-Carlo method for computing the volume of Hyper-Spheres

Lets see through Monte-Carlo method how the volume of a hyper-sphere varies as we increase the dimensions.

We start with a finite set of points $N = \{p_0, \cdots, p_n\}$, where p_j has a random value between [0,1]. Lets remind the euclidean norm for a point $p_j = (p_{j,0}, \cdots, p_{j,n}) \in [0,1]^{n+1} \subset \mathbb{R}^{n+1}$.

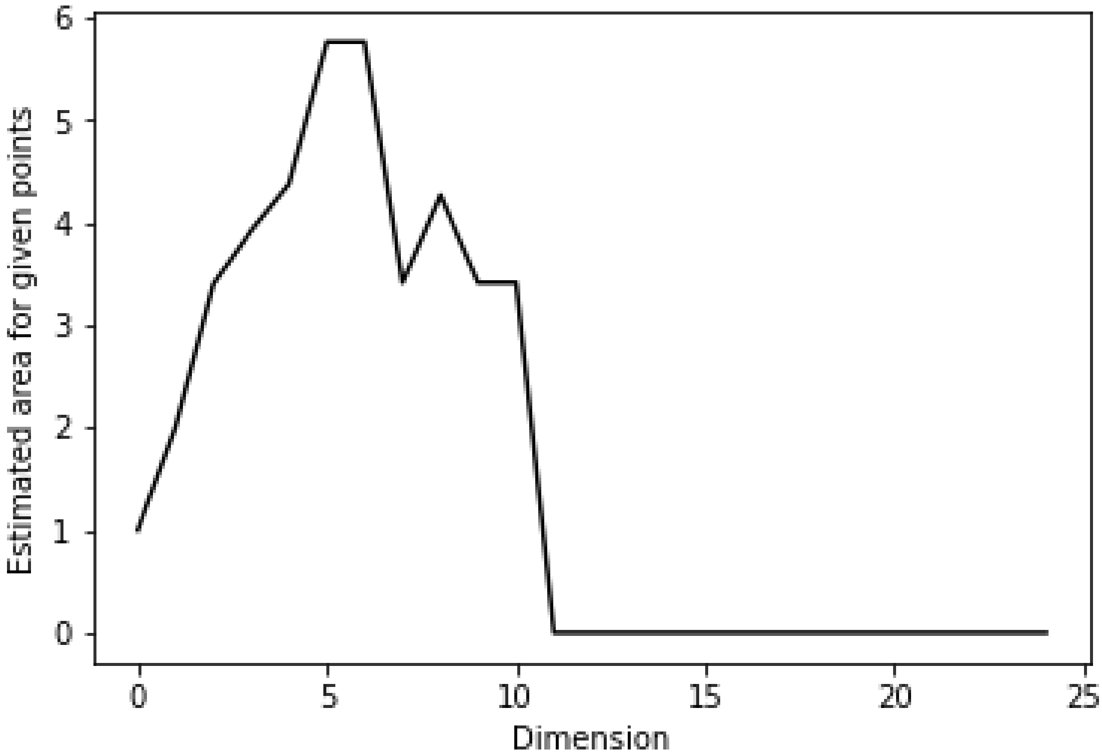
$$||p_j||^2 = \sum_{i=0}^{n} p_{j,i}^2. \tag{2}$$

We define the Hyper-sphere of dimension n and radius 1 as

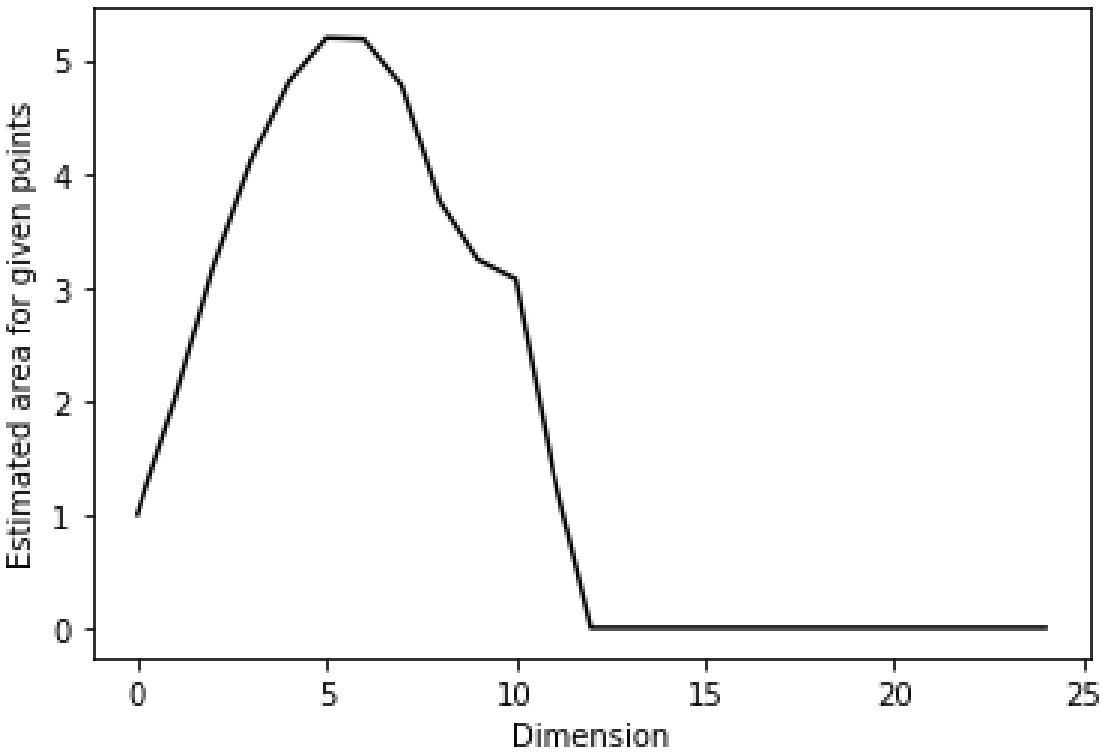
$$S^n = \left\{ x \in \mathbb{R}^{n+1} : ||x|| = 1 \right\}, \tag{3}$$

the point $p_j \in [0,1]^{n+1} \subset \mathbb{R}^{n+1}$ will be inside the sphere (2) if $||p_j|| \leq 1$.

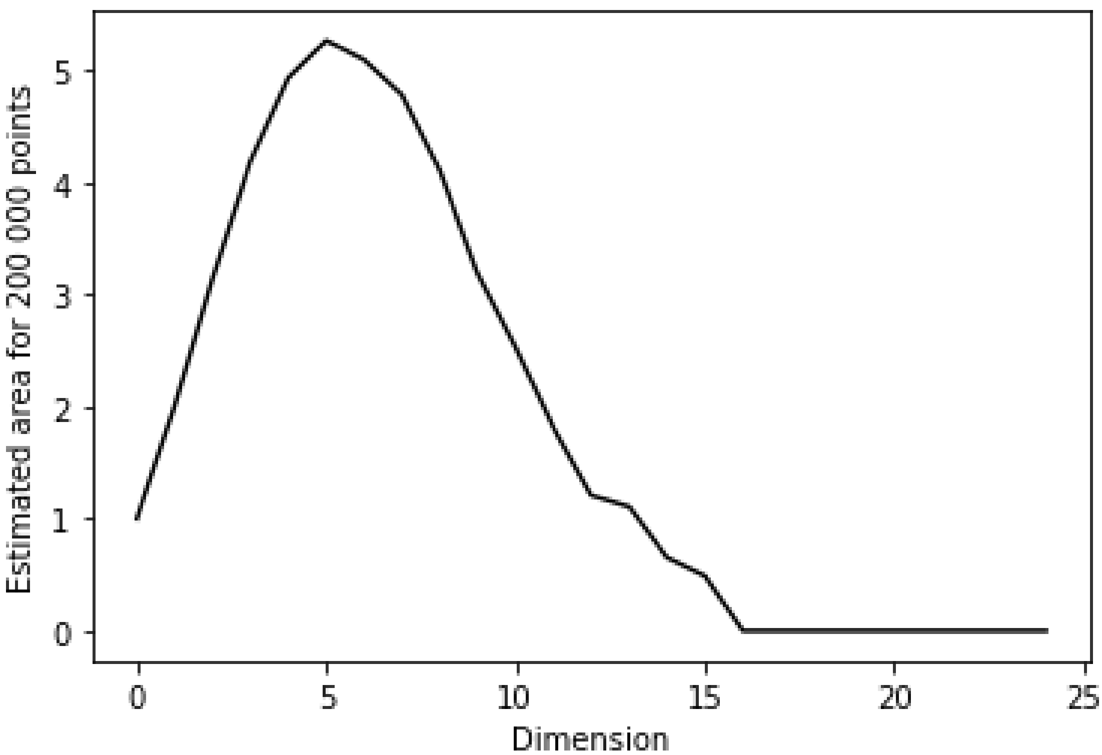
By applying the Monte-Carlo method to compute the volume, we count how many points form N are inside a hyper-sphere S^n and compute the ratio against the carnality fo N. Now lets apply it for all spheres from dimensions ranging from 0 to 30.



The plot for the variation in Volume given 300 random points.



The plot for the variation in Volume given 3000 random points.



The plot for the variation in Volume given 200 000 random points.

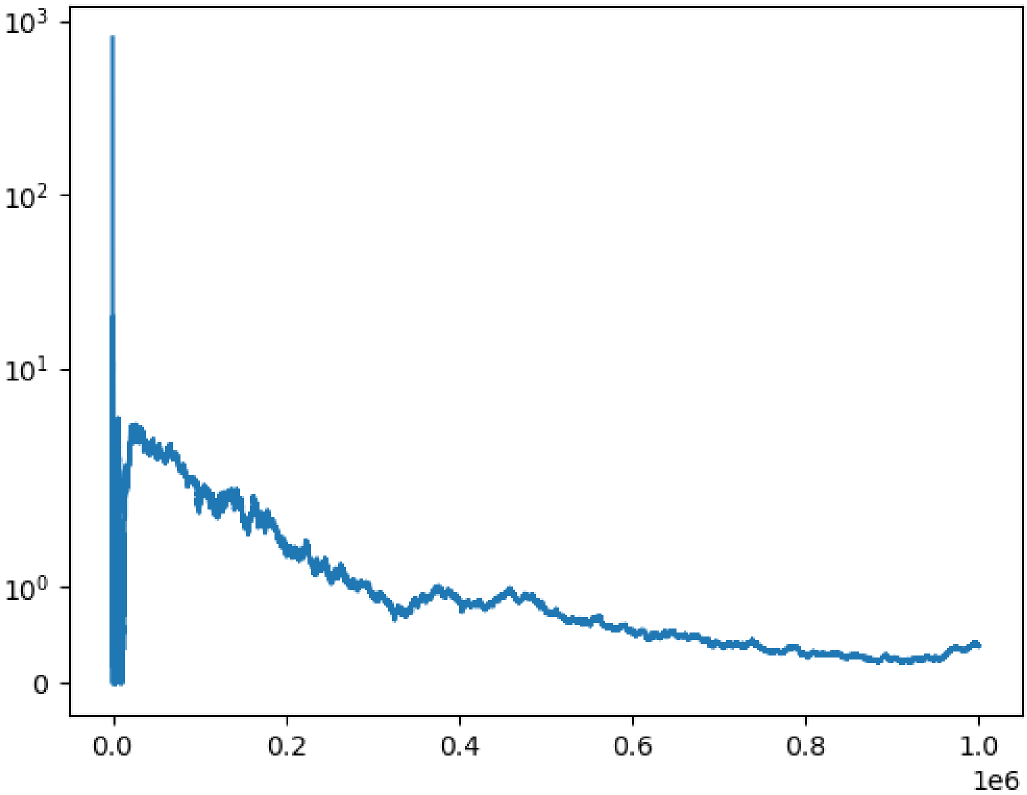
7 Efficiency of the Monte Carlo Method for estimation of Areas

Given the value we want to approximate, the percentage error can be numerically computed trough the simple operation

$$|\,(value - estimation)\,| \times 100. \tag{4}$$

The Monte Carlo method follows the standard error of the mean, $\frac{1}{\sqrt{N}}$, N being the number of points in the sample.

Here is a plot of the error computed using python, showing the error percentage against the number of points.



Square approximation percentage error for $1e^6$ *points*

The approximation is rather inefficient, and a lot of research is conducted on how to improve the Monte Carlo method.

8 References

Wikipedia Méthode de Monte-Carlo French

Wikipedia Monte Carlo Method English

https://www.cs.cornell.edu/courses/cs3220/2008su/slides/montecarlointegration.pdf

Wikipedia Convex hull

https://github.com/EdenForrest/Numerical-Estimation-Areas

Wikipedia Quickhull

https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html

https://en.wikipedia.org/wiki/N-sphere

https://numpy.org/doc/stable/reference/random/index.html

Divide and Conquer

Distance from a line

Wikipedia Monte Carlo integration

9 Code

9.1 Random Convex Hull

```
from matplotlib.path import Path
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull



def pltRdmHull(N):

    #The function is given one argument:
    #N the number of points on which we wish to iterate.
    #The function plots the convexhull from a set of random points on the surface
        [0,1]^2.


    points = np.random.rand(N,2)

    plt.xlim([0,1])
    plt.ylim([0,1])

    plt.plot(points[:,0], points[:,1], 'o')
```

```python

    plt.xlim([0,1])
    plt.ylim([0,1])

    hull = ConvexHull(points)
    hull_path = Path( points[hull.vertices] )

    for simplex in hull.simplices:
       plt.plot(points[simplex, 0], points[simplex, 1],color = 'black')
```

9.2 Animation square convergence

```python
#Enable interactive plot

%matplotlib notebook
from matplotlib.path import Path
from matplotlib.animation import FuncAnimation
from matplotlib.path import Path
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull

N = 10000

#determine vertices of the convex hull
verticesHull = np.array([[-0.5, -0.5], [0.5, -0.5], [0.5, 0.5],[-0.5, 0.5]])

#convexhull path
hull = ConvexHull(verticesHull)
hull_path = Path( verticesHull[hull.vertices] )

#create necessary arrays
storage_array = np.zeros(shape=(N,2))
x = np.arange(0,N)
y = np.zeros(N)

#set initial points to zero
inHull = 0
outHull = 0


#iterate over each point
for i in range(N):
    random_point = np.random.random(2)*3 - 1.5
    #determine if the point is inside the hull
    if hull_path.contains_point(random_point):
        inHull += 1

    #we store areas in array y.
    y[i] = inHull*9/(i+1)


fig = plt.figure()
ax = plt.subplot(1, 1, 1)
data_skip = 20
txt = ax.text(N*0.6, max(y) + 0.25, "")
```

```python

def init_func():
    ax.clear()
    plt.xlabel('n points')
    plt.ylabel('Estimated area')
    plt.xlim((x[0], x[-1]))
    plt.ylim((0, 2 + 0.25))


def update_plot(i):
    ax.clear()
    ax.plot(x[:i + data_skip], y[:i + data_skip], color = 'k')
    ax.scatter(x[i], y[i], color = 'none')

    ax.text(N*0.6, max(y) + 0.25, "Estimation: " + str(round(y[i], 5)))
    ax.set_xlim(0, N)


anim = FuncAnimation(fig,
                     update_plot,
                     frames = np.arange(0, len(x), data_skip),
                     init_func = init_func,
                     interval = 20)

anim.save("squre1Aproximation.gif")
plt.show()
```

9.3 Variation Hyper-Sphere plot

```python
import numpy as np
import math
from matplotlib import pyplot as plt

#We define the norm function.
def inSphere(point):
    #the function is given a point in R^n
    #returns a boolean stating if the norm of the point is smaller than 1.
    if np.sum(np.square(point)) <= 1:
        return True
    else:
        return False


y = []
i = 1
dimensions = 30
points = 200000

for i in range(dimensions):

    inside = 0

    #count how many points are inside sphere of dimension i
    for j in range(points):
        a = np.random.random(i)*2 - 1

```

```
        if inSphere(a):
            inside += 1

    #append the computed volume of sphere of dimension i, y-axis
    y.append((inside*2**i)/points)


#separation for each dimension for the x-axis
x = np.linspace(0,dimensions-1, num = dimensions)

plt.xlabel('Dimension')
plt.ylabel('Estimated area for given points' )
plt.plot(x,y, color = 'black')
```

9.4 NGon plot

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.path as mpltPath


def pltNgonPoints(N, sidepoly, desired_perimeter = 2*np.pi):

    #The function has three arguments: N the number of points on which we wish to
        iterate,
    #the sides of the ngon; desired perimeter of ngon set to default 1.
    #The function plots the random ngon and the scatter points over [-1.5,1.5]^2

    #create random points
    points = np.random.rand(N,2)*3-1.5

    #Set Perimeter
    default_perimeter = 2 * sidepoly * np.sin(np.pi / sidepoly)

    #parametrization of the path, with perimeter adaptation
    polygon = [[np.cos(x+np.pi/4)*(desired_perimeter/default_perimeter),np.sin(x+np.pi
        /4)*(desired_perimeter/default_perimeter)] for x in np.linspace(0, 2*np.pi,
        sidepoly+1)[:sidepoly+1]]


    # Matplotlib mplPath
    path = mpltPath.Path(polygon)
    fig, ax = plt.subplots()
    patch = patches.PathPatch(path, facecolor='none', lw=2)



    plt.scatter(points[:,0],points[:,1], s=0.9)
    ax.add_patch(patch)

    #Equalize axis
    ax.axis('equal')


    plt.plot()
```

```python

pltNgonPoints(1000,36, 2*np.pi)
```

9.5 Exhaustion Plot

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.path as mpltPath


def areaNgon(N, sidepoly, desired_perimeter = 2*np.pi):

    #The function has three arguments: N the number of points on which we wish to
        iterate,
    #the sides of the ngon; desired perimeter of ngon set to default 1.
    #The function plots the random ngon and the scatter points over [-1.5,1.5]^2



    #Set Perimeter
    default_perimeter = 2 * sidepoly * np.sin(np.pi / sidepoly)

    #parametrization of the path, with perimeter adaptation
    polygon = [[np.cos(x+np.pi/4)*(desired_perimeter/default_perimeter),np.sin(x+np.pi
        /4)*(desired_perimeter/default_perimeter)] for x in np.linspace(0, 2*np.pi,
        sidepoly+1)[:sidepoly+1]]


    # Matplotlib mplPath
    path = mpltPath.Path(polygon)
    inHull = 0
    #iterate over each point
    for i in range(N):

        #determine if the point is inside the hull
        if path.contains_point(np.random.random(2)*3 - 1.5):
            inHull += 1
    return inHull/N




y = []
i = 1
sides = 35
points = 30000
i = 3
for i in range(3,sides+1):
    y.append(areaNgon(points, i)*9)

x = np.linspace(3,sides, num = sides-2)

plt.xlabel('Sides nGon')
plt.ylabel('Estimated area for given points' )
plt.plot(x,y, color = 'black')
```

9.6 Animation π convergence

```
#Enable interactive plot
%matplotlib notebook
import math
from matplotlib.path import Path
from matplotlib.animation import FuncAnimation
from matplotlib.path import Path
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull
from matplotlib.artist import Artist

N = 10000

# create necessary arrays
x = np.arange(0, N)
y = np.zeros(N)

# set initial points to zero
inHull = 0


def inCircle(point):
    # the function is given a point in R^n
    # returns a boolean stating if the norm of the point is smaller than 1.
    if np.sum(np.square(point)) <= 1:
        return True
    else:
        return False


# iterate over each point
for i in range(N):
    random_point = np.random.rand(2)*2 - 1

    # determine if the point is inside the hull
    if inCircle(random_point):
        inHull += 1

    # we store areas in array y.
    y[i] = (inHull*4)/(i + 1)

fig = plt.figure()
ax = plt.subplot(1, 1, 1)
data_skip = 20
txt = ax.text(N*0.6, max(y) + 0.25, "")

def init_func():
    ax.clear()
    plt.xlabel('n points')
    plt.ylabel('Estimated area')
    plt.xlim((x[0], x[-1]))
    plt.ylim((0, 4 + 0.5))


def update_plot(i):
    ax.clear()
```

```
    ax.plot(x[:i + data_skip], y[:i + data_skip], color = 'k')
    ax.scatter(x[i], y[i], color = 'none')

    ax.text(N*0.6, max(y) + 0.25, "Estimation: " + str(round(y[i], 5)))
    ax.set_xlim(0, N)


anim = FuncAnimation(fig,
                     update_plot,
                     frames = np.arange(0, len(x), data_skip),
                     init_func = init_func,
                     interval = 20)

anim.save("piAproximation.gif")
plt.show()
```

9.7 Error estimation

```
#iterate over each point
for i in range(N):
    random_point = np.random.random(2)*3 - 1.5
    #determine if the point is inside the hull
    if hull_path.contains_point(random_point):
        inHull += 1

    #we store error % in array y.
    y[i] = abs(inHull*9/(i+1) - 1)*100

fig = plt.figure()
ax = plt.subplot(1, 1, 1)

ax.set_yscale('symlog')
plt.ylabel('Computed percentage error')
plt.xlabel('Points')
plt.plot(x,y)


plt.savefig('errorSquare.png')
```