Ant foraging for food: 2D Simple Random Walk as a Martingale

An ant leaves its anthill in order to forage for food. It moves with the speed of 10 cm per second, but it doesn't know where to go, therefore every second it moves randomly 10 cm directly north, south, east or west with equal probability.

Question 1

If the food is located on east-west lines 20 cm to the north and 20 cm to the south, as well as on north-south lines 20 cm to the east and 20 cm to the west from the anthill, how long will it take the ant to reach it on average?

Solution (All distances expressed in cm and times in sec)

Approach 1: Martingale

- The ant is executing a 2D simple symmetric unbiased random walk $\{S_t\}_{t\geq 1}$ on a regular lattice with a step size (stride) of 10, starting at the origin $S_0 = (0,0)$ and moving north, south, east, west with probabilities $1/4$.
- S_t is a vector in $\mathbb{Z}^2$ denoting the position of the ant at time t. Let $\{X_t\}_{t\geq 1}$ be the sequence of independent random variables (2D vectors representing ant's strides every second) such that $Pr\{X_t = (-10,0)\} = Pr\{X_t = (10,0)\} = Pr\{X_t = (0,10)\} = Pr\{X_t = (0,-10)\} = \frac{1}{4}$ and $S_t = \sum_{i=1}^{t} X_i$.
- We consider that all variables are integrable and define the filtration $\{\mathscr{F}_t\}_{t\geq 1} = \sigma(S_0, X_1, \ldots, X_t); \quad t \geq 1$, then we have (equalities between conditional expectations holding almost surely)
$$\mathbb{E}[S_{t+1}|\mathscr{F}_t] = \mathbb{E}[X_{t+1} + S_t|\mathscr{F}_t] = \mathbb{E}[X_{t+1}|\mathscr{F}_t] + \mathbb{E}[S_t|\mathscr{F}_t]$$
But X_{t+1} is independent of $\mathscr{F}_t$ therefore
$$\mathbb{E}[X_{t+1}|\mathscr{F}_t] = \mathbb{E}[X_{t+1}] = \frac{1}{4}(-10,0) + \frac{1}{4}(10,0) + \frac{1}{4}(0,10) + \frac{1}{4}(0,-10) = (0,0)$$
Also, S_t is $\mathscr{F}_t$-measurable $\implies \mathbb{E}[S_t|\mathscr{F}_t] = S_t$. **This indicates that $\mathbb{E}[S_{t+1}|\mathscr{F}_t] = S_t$ and $\{(S_t, \mathscr{F}_t)\}_{t\geq 1}$ is a martingale.**
- If S_t is a martingale (with bounded increments), then
$$M_t := |S_t|^2 - t$$
is also a martingale. This allows us to use an **Optional Stopping Theorem** on $\{(M_t, \mathscr{F}_t)\}_{t\geq 1}$.
- Defining the stopping time $T = \inf\{t \geq 0 : |S_t| = K\}$ as the time when the ant finds the food, *i.e.*, crosses the boundary at a distance K.
- At time T, we have $|S_t|^2 \geq K^2 \implies M_t \geq K^2 - T$
- Using Optional Stopping Theorem, we have $\mathbb{E}[M_t] = \mathbb{E}[M_0] \implies \mathbb{E}[T] \geq K^2$.
- Considering the distance of food from the anthill (20 cm) and ant's speed (10 cm/s), $\mathbb{E}[T] \geq K^2 \implies \mathbb{E}[T] \geq (20/10)^2 = 4$ sec.

Approach 2: Using properties of a random walk

Following the finite additivity property of the expectation,
$$\mathbb{E}[S_t] = \sum_{i=1}^{t} \mathbb{E}[X_i] = (0,0)$$
A similar calculation, using the independence of the random variables X_t shows that
$$\mathbb{E}[S_t^2] = t\,\mathbb{E}[X_i^2] + 2 \sum_{1\leq i<j\leq t} \mathbb{E}[X_i \cdot X_j] = t\,\mathbb{E}[X_i^2]$$
This shows that the time required to cover the expected translation distance of $\mathbb{E}[|S_t|]$ is
$$t = \frac{\mathbb{E}[S_t^2]}{\mathbb{E}[X_i^2]}$$
For our present scenario, $\mathbb{E}[X_i^2] = \frac{1}{4}(4(100)) = 100$ and $\mathbb{E}[S_i^2] = 20^2 = 400 \implies t = 400/100 = 4$ sec.

Therefore, on an average, the ant will take minimum of 4 seconds to reach the food.

Question 2

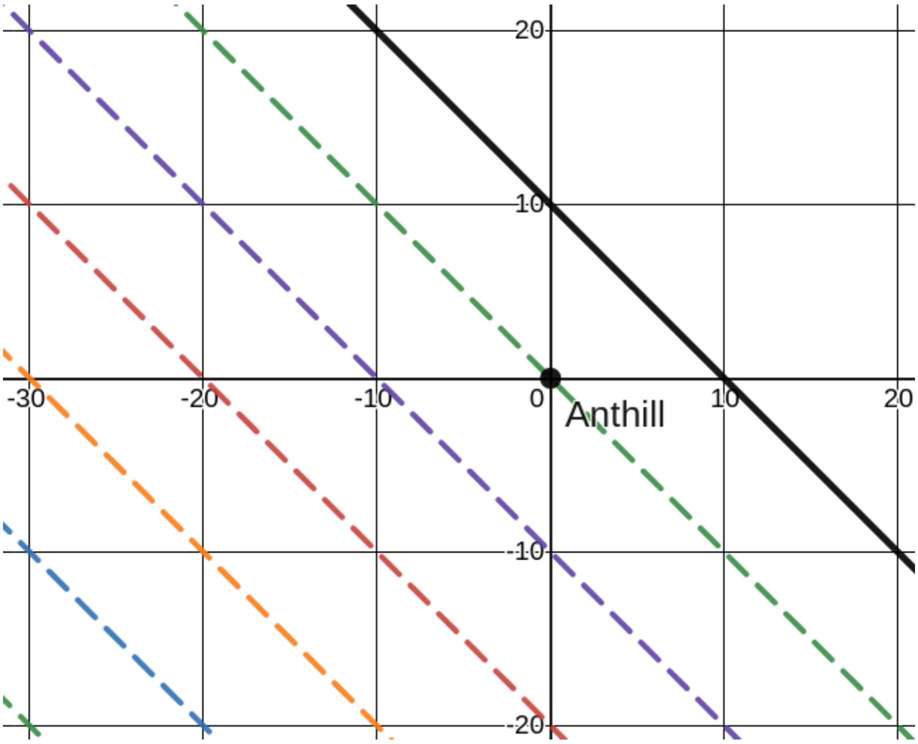
What is the average time the ant will reach food if it is located only on a diagonal line passing through $(10$ cm, 0 cm$)$ and $(0$ cm, 10 cm$)$ points?

Solution

The figure below shows the food line passing through $(10$ cm, 0 cm$)$ and $(0$ cm, 10 cm$)$ points and the anthill. The ant's motion can now be seen as a simple symmetric unbiased 1D random walk Z_t, hopping between the parallel diagonal lines (shown in colors). The diagonal lines are given as
$$Z_t = 10 : \Longrightarrow y = -x + 10 \quad \text{(Food line)}$$
$$Z_t = 0 : \Longrightarrow y = -x \quad \text{(Starting ant location)}$$
$$Z_t = -10 : \Longrightarrow y = -x - 10$$
$$Z_t = -20 : \Longrightarrow y = -x - 20$$
and so on.

```
In [ ]: from IPython.display import Image
        Image('random_walk_question_2.png')
```

Out[ ]:



In this new reduced landscape, the ant, starting at $Z_{t=0} = 0$, goes with the speed of 10 cm/sec to the right (north or east) with probability $1/2$ and to the left (south or west) with the same probability $(1/2)$.

Question 2 can now be changed to:

If $\{Z_t\}_{t\geq 0}$ is a 1D unbiased random walk with $Z_0 = 0$, and we define $T = \inf\{t \geq 0 : Z_t = 10\}$, what is $\mathbb{E}[T]$?

$\{Z_t\}_{t\geq 0}$ here has only upper bound at $Z_t = 10$ while there is no lower bound.

Let's suppose that the lower bound exists at $Z_t = -N$ for some $N \in \mathbb{Z}^+$. Then applying the optional stopping theorem on the martingale Z_t gives

$$\mathbb{E}[Z_t] = p(10) + (1-p)(-N) = Z_0 = 0$$

where $p = Pr\{Z_t = 10\}$, i.e., p is the probability of the ant reaching the food.

Solving for p gives $p = \frac{N}{N+10}$.

Same as in question 1, we now use the property of the martingale $Z_T^2 - T$ to get the expectation of stopping time $\mathbb{E}[T]$.

$$\mathbb{E}[Z_T^2 - T] = Z_0^2 - 0 = 0$$
$$\implies \mathbb{E}[Z_T^2] = p(10)^2 + (1-p)(-N)^2 = \mathbb{E}[T]$$

Substituting for p, we get $\mathbb{E}[T] = 10N$.

This shows that the stopping time equals the product of the lengths of the upper and lower bounds (a well known property of martingales).

However, Z_t is in fact unbounded below ($N = \infty$). Thich indicates that $\mathbb{E}[T] = \infty$.

Therefore, on an average, the ant will take an infinite time to reach the food ($\mathbb{E}[T] = \infty$).

Question 3

Can you write a program that comes up with an estimate of average time to find food for any closed boundary around the anthill? What would be the answer if food is located outside the boundary defined by $\frac{(x-2.5)^2}{30^2} + \frac{(y-2.5)^2}{40^2} < 1$ (lengths in cm) in coordinate system where the anthill is located at ($x = 0$ cm, $y = 0$ cm)? Provide us with a solution rounded to the nearest integer.

Solution

The program below calculates the expectation of stopping time for a 2D unbiased random walk of the ant, starting from ($x = 0$ cm, $y = 0$ cm). Two types of boundaries have been constructed :

1. **Elliptical boundaries**: Closed loops with the equation $\frac{(x-\alpha)^2}{a^2} + \frac{(y-\beta)^2}{b^2} = 1$, where the location ($x = \alpha$ cm, $y = \beta$ cm) is the center, while the lengths $2a$ and $2b$ are the width and height of the ellipse, respectively.
2. **Polygons**: Regular polygons, characterized by the center location, the number of sides and the length of each side, as well as irregular polygons given the vertices.

In [ ]:
```python
# Importing relevant libraries
import random
import numpy as np
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
```

In [ ]:
```python
class BoundaryCurve:
  '''
  A class representing the elliptical boundary of the region for forage.

  Attributes
    ----------
    boundary_center : tuple
        coordinates of the center of the ellipse (default (2.5,2.5))
    a : float
        half the width of the ellipse (default  30.0)
    b : float
        half the height of the ellipse (default 40.0)

    Methods
    -------
    is_inside_boundary(point)
        Returns True if the point is contained within the boundary
  '''
  def __init__(self,
               boundary_center: tuple=(2.5,2.5),
               a: float=30.0,
               b: float=40.0):
    '''
    Parameters
    ----------
    boundary_center : tuple
        coordinates of the center of the ellipse (default (2.5,2.5))
    a : float
        half the width of the ellipse (default  30.0)
    b : float
        half the height of the ellipse (default 40.0)
    '''
    self._boundary_center = boundary_center
    self._a = a
    self._b = b

  @property
  def boundary_center(self):
    return self._boundary_center

  @property
  def a(self):
    return self._a

  @property
  def b(self):
    return self._b

  def is_inside_boundary(self, point: tuple) -> bool:
    '''
    Checks if the point is contained within the boundary

    Parameters
    ----------
    point : tuple
        x and y coordinates of the point under evaluation

    Returns
    ------
        True if point exists within boundary, else False
    '''
    x, y = point
    return ((x-self._boundary_center[0])**2/self._a**2) + ((y-self._boundary_center[1])**2/self._b**2) < 1
```

In [ ]:
```python
class BoundaryPolygon:
  '''
  Parent class representing the Polygon shaped boundary of region under forage

  Methods
  -------
  is_inside_boundary(point)
      Returns True if the point is contained within the boundary
  '''
  def is_inside_boundary(self, point: tuple) -> bool:
    return self._polygon.contains(Point(point))



class RegPolygon(BoundaryPolygon):
  '''
  Subclass of BoundaryPolygon representing polygon with an equiangular and equilateral shape

  Attributes
  ----------
  boundary_center : tuple
      coordinates of the center of the ellipse (default (2.5,2.5))
  n_sides : int
      number of sides of the regular polygon (default 4)
  side_length : float
      length of side (default 20.0)

  Methods
  -------
  create_polygon
      Returns Polygon object
  '''

  def __init__(self,
               boundary_center: tuple=(2.5,2.5),
               n_sides: int=4,
               side_length: float=20.0):
    '''
    Parameters
    ----------
    boundary_center : tuple
        coordinates of the center of the ellipse (default (2.5,2.5))
    n_sides : int
        number of sides of the regular polygon (default 4)
    side_length : float
        length of side (default 20.0)
    '''
    self._boundary_center = boundary_center
    self._n_sides = n_sides
    self._side_length = side_length
    self._polygon = self.create_polygon()

  @property
  def boundary_center(self):
    return self._boundary_center

  @property
  def n_sides(self):
    return self._n_sides

  @property
  def side_length(self):
    return self._side_length

  @property
  def polygon(self):
    return self._polygon

  def create_polygon(self) -> Polygon:
    '''
    Takes the polygon based on boundary center, number of sides and side length

    Parameters
    ----------
    boundary_center : tuple
        coordinates of the center of the ellipse (default (2.5,2.5))
    n_sides : int
        number of sides of the regular polygon (default 4)
    side_length : float
        length of side (default 20.0)

    Returns
    ------
        Polygon object
    '''
    vertices_x, vertices_y = [], []
    for k in range(1, self._n_sides+1):
```

```
      vertices_x.append( np.cos(2*np.pi*k/self._n_sides) )
      vertices_y.append( np.sin(2*np.pi*k/self._n_sides) )
    vertices_x = self._boundary_center[0] + 0.5*self._side_length*np.array(vertices_x)/np.sin(np.pi/self._n_sides)
    vertices_y = self._boundary_center[1] + 0.5*self._side_length*np.array(vertices_y)/np.sin(np.pi/self._n_sides)
    vertices = [(x,y) for (x,y) in zip(vertices_x, vertices_y)]
    return Polygon(vertices)



class IrregPolygon(BoundaryPolygon):
  '''
  Subclass of BoundaryPolygon representing irregular polygon

  Attributes
  ----------
  vertices : list
      list of coordinates of polygon vertices

  Methods
  -------
  create_polygon
      Returns Polygon object
  '''
  def __init__(self, vertices: list):
    self._vertices = vertices
    self._polygon = Polygon(self._vertices)

  @property
  def vertices(self):
    return self._vertices

  @property
  def polygon(self):
    return self._polygon
```

In [ ]:
```python
class FoodSearch:
  '''
  class representing the protocol for calculating average stopping time

  Attributes
  ----------
  anthill_location : tuple (cm)
      coordinates of the location of ant's home (default (0.0, 0.0))
  ant_speed : float (cm/s)
      ant's speed (default 10.0)
  delta_T : float (sec)
      time ant needs to make one stride (default 1.0)
  repeat: int
      number of repetitions of forage to get statistics (default 10000)
  boundary_type: 1, 2, 3
      type of boundary- 1: ellipse, 2: regular polygon, 3: irregular polygon

  Methods
  -------
  anthill_inside_region
      checks if anthill is contained within the region
  make_boundary
      returns boundary object
  forage
      returns one trajectory of forage till ant reaches food
  plot_trajectory
      plots ant's trajectory
  sample_stopping_time
      returns list of stopping times after multiple forages
  stats_stopping_time(input_array)
      returns statistics of input array
  moving_avg(input_array, window_size)
      returns moving average of input array for a given window size
  plot_stopping_times
      plots stopping time vs. forage repetitions
  '''

  def __init__(self,
               anthill_location: tuple=(0.0,0.0),
               ant_speed: float=10.0,
               delta_T: float=1.0,
               repeat: int = 10000,
               boundary_type=1):
    '''
    Parameters
    ----------
    anthill_location : tuple (cm)
        coordinates of the location of ant's home (default (0.0, 0.0))
    ant_speed : float (cm/s)
        ant's speed (default 10.0)
    delta_T : float (sec)
        time ant needs to make one stride (default 1.0)
    repeat: int
        number of repetitions of forage to get statistics (default 10000)
    boundary_type: 1, 2, 3
        type of boundary- 1: ellipse, 2: regular polygon, 3: irregular polygon
    '''
    self._anthill_location = anthill_location
    self.ant_speed = ant_speed
    self.delta_T = delta_T
    self._repeat = repeat
    self._steps = self.delta_T * self.ant_speed * np.array([-1.0, 1.0])
    self._boundary_type = boundary_type
    self._boundary = self.make_boundary()
    self._check_anthill_location = self.anthill_inside_region()

  @property
  def anthill_location(self):
    return self._anthill_location

  @property
  def repeat(self):
    return self._repeat

  @property
  def steps(self):
    return self._steps

  @property
  def boundary_type(self):
    return self._boundary_type

  @property
  def boundary(self):
    return self._boundary
```

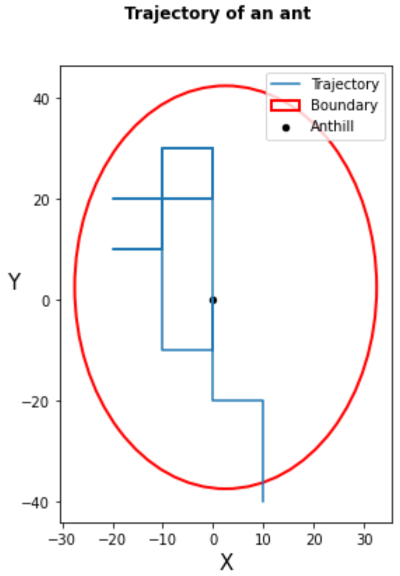
```python
  ## Setters

  @anthill_location.setter
  def anthill_location(self, new_location):
    if isinstance(new_location, tuple):
      self._anthill_location = new_location
    else:
      print(f'Give a tuple of floats for the new anthill location.')

  @repeat.setter
  def repeat(self, value):
    if isinstance(value, int) and value > 0:
      self._repeat = value
    else:
      print(f'Give a positive integer.')

  # Method


  def anthill_inside_region(self) -> bool:
    '''
    Checks if the anthill is contained within the boundary

    Returns
    ------
        True if anthill exists within boundary, else False
    '''
    if not self._boundary.is_inside_boundary(self._anthill_location):
      print('Change the anthill location so that it is inside the bounded region.')
      return False
    else:
      print('The anthill is inside the bounded region.')
      return True


  def make_boundary(self):
    '''
    Creates boundary object depending on boundary type (1: ellipse, 2: regular polygon, 3: irregular polygon)

    Parameters
    ---------
    vertices : list
        list of (x,y) coordinates of vertices of irregular polygon
    boundary_center : tuple
        (x,y) coordinates of center of ellipse or regular polygon
    a,b : float
        half the width (a) and half the height (b) of ellipse
    n_sides : int
        number of sides of regular polygon
    side_length : float
        length of each side of regular polygon

    Returns
    ------
        boundary object
    '''
    if self._boundary_type == 3:
      c = input('Give the coordinates of vertices of the polygon as tuples separated by commas (cm): ')
      print(f'Creating the boundary as an irregular polygon with vertices {c}.')
      vertices = list(eval(c))
      return IrregPolygon(vertices)
    else:
      c = input('Give the coordinates of boundary center with space (cm): ')
      boundary_center = tuple(float(s) for s in c.strip(" ").split(" "))
      if self._boundary_type == 1:
        print('Creating an elliptical boundary.')
        p = input('Give the width (2a) and height (2b) of the ellipse with space (cm): ')
        a, b = tuple(0.5*float(s) for s in p.strip(" ").split(" "))
        print(f'Ellipse ((x - alpha)^2)/a^2 + ((y - beta)^2)/b^2 = 1: (alpha, beta): {boundary_center}, a: {a}, 
b:{b}')
        return BoundaryCurve(boundary_center=boundary_center, a=a, b=b)
      else:
        p = input('Give the number of sides and the side length (cm) of the polygon with space: ')
        n_sides, side_length = tuple(float(s) for s in p.strip(" ").split(" "))
        print(f'Creating the boundary as a regular polygon with center {boundary_center}, {round(n_sides)} sides and 
{side_length} cm each side.')
        return RegPolygon(boundary_center=boundary_center,
                          n_sides=int(n_sides),
                          side_length=side_length)
```

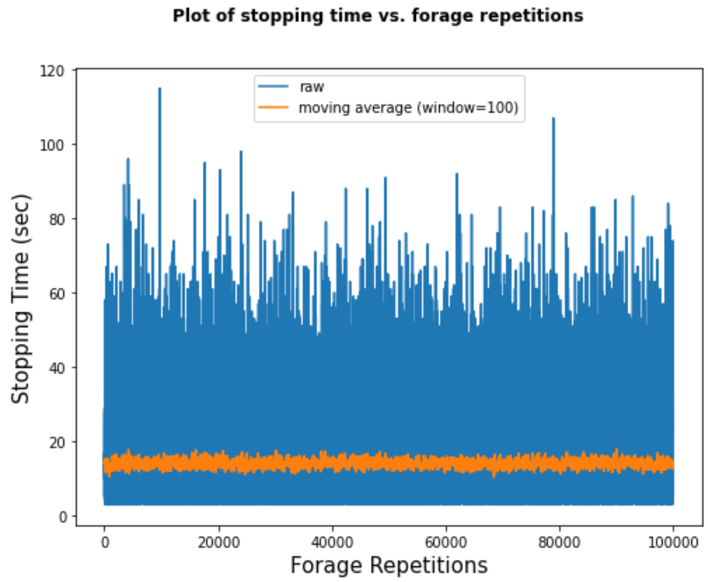
```python
def forage(self, record_steps=False):
    '''
    Execute ant's forage and outputs ant's trajectory and stopping time

    Parameters
    ---------
    record_steps : bool
        create a trajectory of x and y coordinates if True

    Returns
    ------
        stopping time if record_step = False
        tuple of list of x and y coordinates if record_steps = True
    '''
    px, py = self._anthill_location
    if record_steps:
      x, y = [px], [py]
      while self._boundary.is_inside_boundary((px, py)):
        if random.choice([0, 1]) == 0:
          step = random.choice(self._steps)
          px += step
          x.append(px)
          y.append(py)
        else:
          step = random.choice(self._steps)
          py += step
          y.append(py)
          x.append(px)
      return x, y
    else:
      T = 0.0
      while self._boundary.is_inside_boundary((px, py)):
        if random.choice([0, 1]) == 0:
          step = random.choice(self._steps)
          px += step
        else:
          step = random.choice(self._steps)
          py += step
        T += self.delta_T
      return T


def plot_trajectory(self):
    '''
    Plot the trajectory of the ant (array of y coordinates vs array x coordinates of ant's path)
    '''
    hill_x, hill_y = self._anthill_location
    fig, ax = plt.subplots(figsize=(6,6))
    fig.suptitle('Trajectory of an ant', fontweight ="bold")
    ax.scatter([hill_x], [hill_y], s=20, label='Anthill', color='black')
    x, y = self.forage(record_steps=True)
    ax.plot(x, y, label='Trajectory')
    ax.set_aspect('equal', 'box')
    if self._boundary_type == 1:
      ellipse = Ellipse(xy=(self._boundary._boundary_center[0],
                            self._boundary._boundary_center[1]),
                        width=2*self._boundary.a, height=2*self._boundary.b,
                        edgecolor='r', fc='None', lw=2, label='Boundary')
      ax.add_patch(ellipse)
    else:
      ax.plot(*self._boundary.polygon.exterior.xy, label='Boundary', color='r')
    ax.set_xlabel('X', fontsize=15)
    ax.set_ylabel('Y', fontsize=15, rotation=0)
    ax.legend(loc = 'best')
    plt.show()
    plt.close()


def sample_stopping_time(self) -> list:
    '''
    Repeat forage to get statistics of stopping time

    Returns
    ------
        list of stopping times
    '''
    stopping_times = [self.forage() for _ in range(self._repeat)]
    return stopping_times


def stats_stopping_time(self, input_array) -> tuple:
    '''
    Get statistics of stopping time

    Returns
    ------
        tuple containing mean and standard deviation of stopping time
    '''
```

```python
        mean_stopping_time = round(np.mean(input_array))
        std_stopping_time  = round(np.std(input_array))
        return mean_stopping_time, std_stopping_time


    def moving_avg(self, input_array, window_size: int=100) -> np.array:
        '''
        Get moving average of input array

        Parameters
        ------
        input_array
            input array
        window_size
            number of elements to calculate moving average over

        Returns
        ------
            array of moving average
        '''
        return np.convolve(input_array, np.ones(window_size)/window_size, mode='valid')


    def plot_stopping_times(self, window_size: int=100):
        '''
        Plot stopping time vs. number of forage repetitions

        Parameters
        ------
        window_size
            number of elements to calculate moving average over
        '''
        array_stopping_time = self.sample_stopping_time()
        fig, ax = plt.subplots(figsize=(8,6))
        fig.suptitle('Plot of stopping time vs. forage repetitions', fontweight ="bold")
        ax.plot(range(self._repeat), array_stopping_time, label='raw')
        ax.plot(range(self._repeat-window_size+1),
                self.moving_avg(array_stopping_time, window_size=window_size),
                label=f'moving average (window={window_size})')
        ax.set_ylabel('Stopping Time (sec)', fontsize=15)
        ax.set_xlabel('Forage Repetitions', fontsize=15)
        ax.legend(loc = 'best')
        plt.show()
        plt.close()
        print('********************STOPPING TIME STATISTICS***********************')
        stats = self.stats_stopping_time(array_stopping_time)
        print(f'Average : {stats[0]} sec.')
        print(f'Std : {stats[1]} sec.')
        print(f'Repetitions: {self._repeat}')
        print('********************************************************************')
```

```
In [ ]: # Code execution

        # Boundary with equation: ((x - 2.5)^2)/30^2 + ((y - 2.5)^2)/40^2 = 1

        if __name__== "__main__":
          # Start the food search
          foodsearch = FoodSearch(anthill_location=(0.0, 0.0),
                                  ant_speed=10,
                                  delta_T=1.0,
                                  repeat=100000,
                                  boundary_type=1)
          foodsearch.plot_trajectory()     # Plot sample trajectory
          foodsearch.plot_stopping_times() # Plot stopping time
```

```
Give the coordinates of boundary center with space (cm): 2.5 2.5
Creating an elliptical boundary.
Give the width (2a) and height (2b) of the ellipse with space (cm): 60 80
Ellipse ((x - alpha)^2)/a^2 + ((y - beta)^2)/b^2 = 1: (alpha, beta): (2.5, 2.5), a: 30.0, b:40.0
The anthill is inside the bounded region.
```





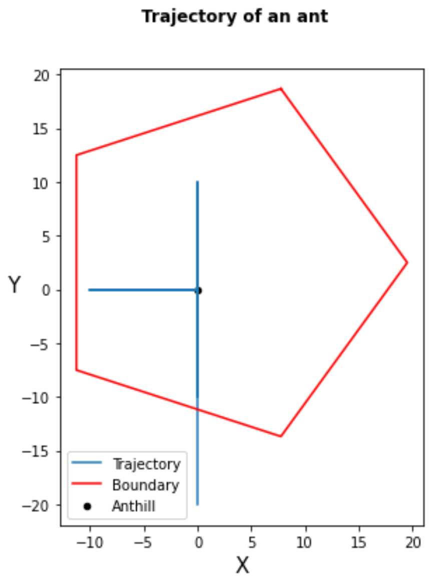

```
******************STOPPING TIME STATISTICS**********************
Average : 14 sec.
Std : 10 sec.
Repetitions: 100000
****************************************************************
```

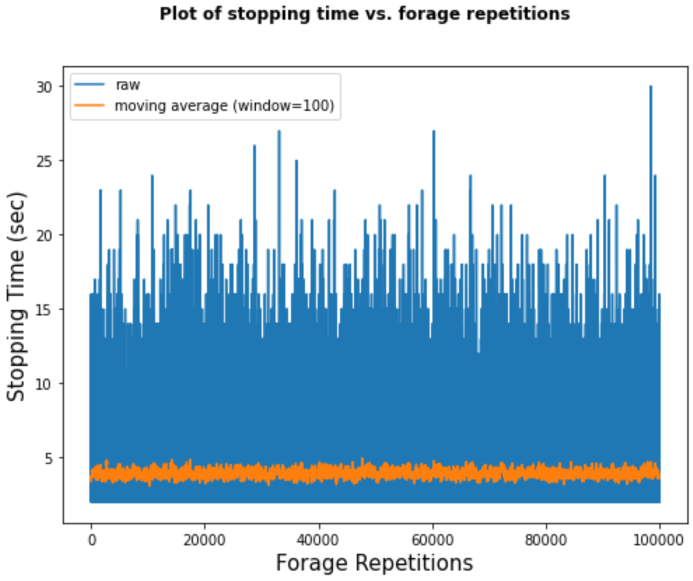
Therefore, an average time to find food, if it is located outside the boundary defined by $\frac{(x-2.5)^2}{30^2} + \frac{(y-2.5)^2}{40^2} < 1$ (lengths in cm) in coordinate system where the anthill is located at $(x = 0$ cm, $y = 0$ cm$)$, is 14 sec.

In [ ]:

```
# Boundary as a regular polygon

foodsearch = FoodSearch(anthill_location=(0.0, 0.0),
                        ant_speed=10,
                        delta_T=1.0,
                        repeat=100000,
                        boundary_type=2)
foodsearch.plot_trajectory()
foodsearch.plot_stopping_times()
```

```
Give the coordinates of boundary center with space (cm): 2.5 2.5
Give the number of sides and the side length (cm) of the polygon with space: 5 20
Creating the boundary as a regular polygon with center (2.5, 2.5), 5 sides and 20.0 cm each side.
The anthill is inside the bounded region.
```

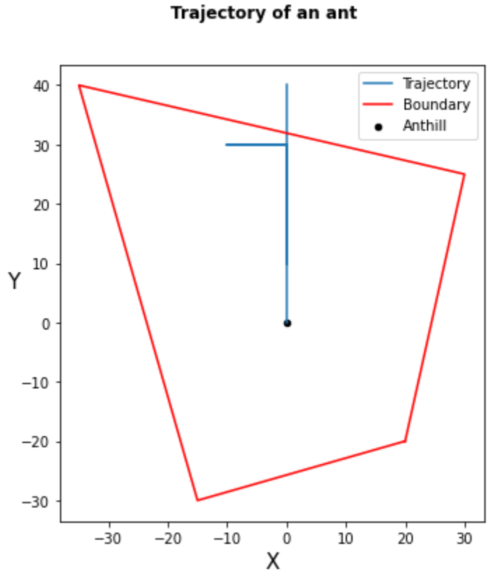


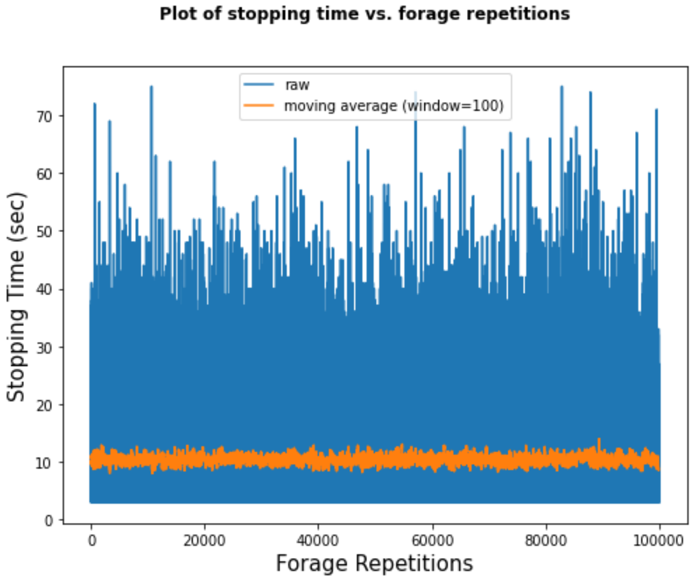



```
*******************STOPPING TIME STATISTICS**********************
Average : 4 sec.
Std : 2 sec.
Repetitions: 100000
*****************************************************************
```

In [ ]:
```
# Boundary as an irregular polygon

foodsearch = FoodSearch(anthill_location=(0.0, 0.0),
                        ant_speed=10,
                        delta_T=1.0,
                        repeat=100000,
                        boundary_type=3)
foodsearch.plot_trajectory()
foodsearch.plot_stopping_times()
```

```
Give the coordinates of vertices of the polygon as tuples separated by commas (cm): (20,-20), (30,25), (-35,40), (-1
5,-30)
Creating the boundary as an irregular polygon with vertices (20,-20), (30,25), (-35,40), (-15,-30).
The anthill is inside the bounded region.
```






```
*******************STOPPING TIME STATISTICS**********************
Average : 10 sec.
Std : 7 sec.
Repetitions: 100000
*****************************************************************
```