# Get Pi with Gaussian and Monta-Calo

Gaussian formula:

Gaussian graph

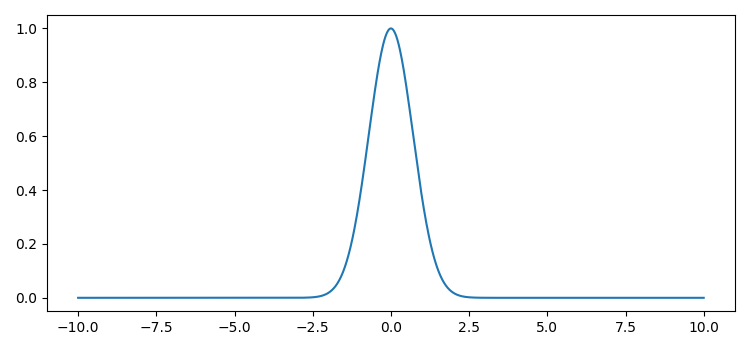


Figure 1：The graph of Gaussian Function

The square of Gaussian formula when x in () is

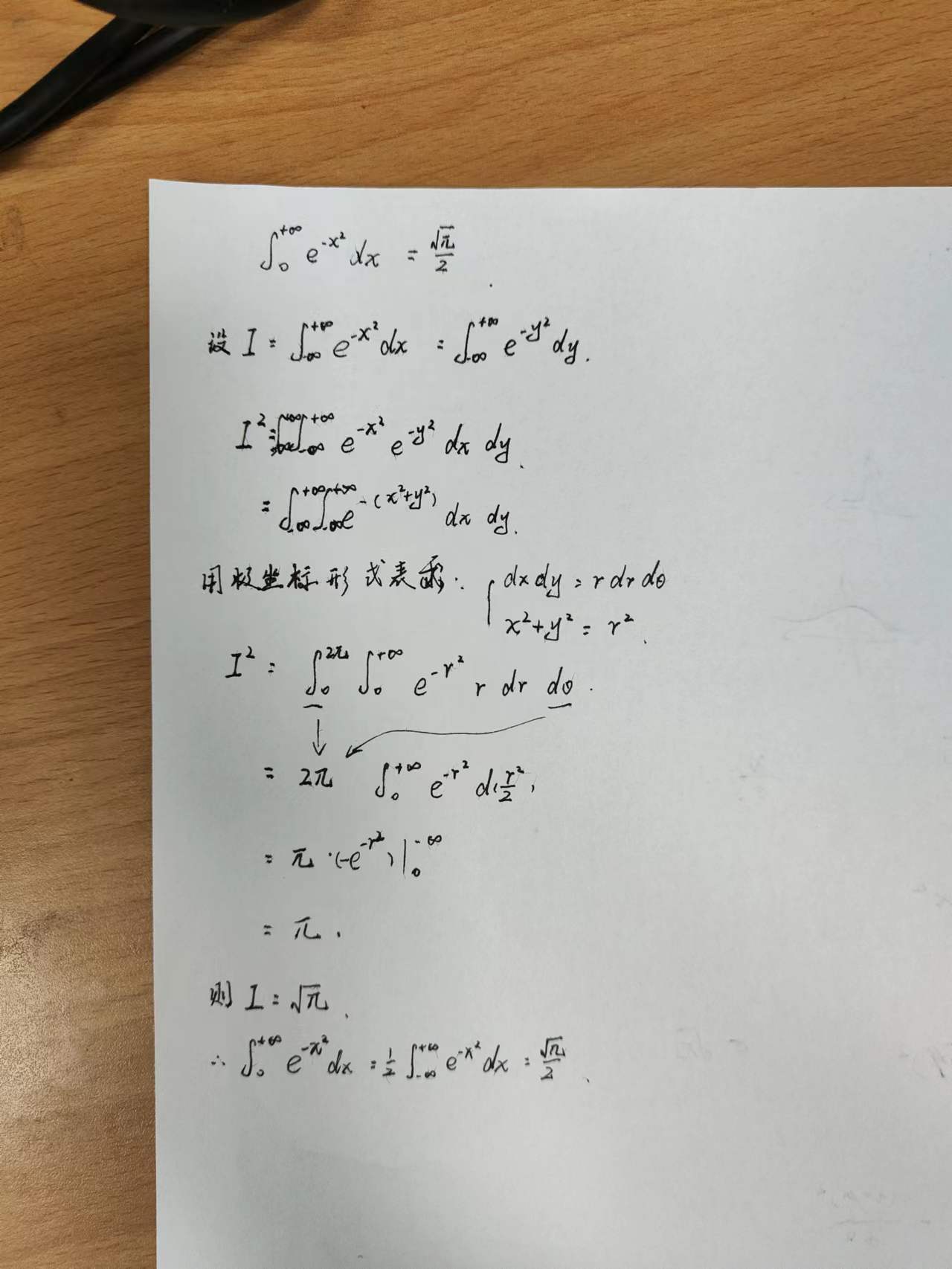


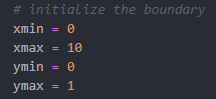
Figure 2：Solve sqaure of Gaussian

Confirm the boundary of simulating.



When the x is 10, is much smaller than the x in 1

So the x can be



Simulate with Monta-Calo method and get pi using language pi

1. import random

2. import numpy as np

3. from time import perf\_counter

4. from multiprocessing.dummy import Pool as ThreadPool

5.

6. # define the Gaussian Function

7. def gaussian(x):

8. return (np.exp(-x\*\*2))

9.

10. # random throw dot and analyze

11. def simulate\_dot\_throwing(xmin, xmax, ymin, ymax):

12. global dot\_in\_area

13. x,y = random.uniform(xmin,xmax),random.uniform(ymin,ymax)

14. if y < gaussian(x):

15. dot\_in\_area = dot\_in\_area + 1

16. return 0

17.

18.

19. # counting program duration

20. start = perf\_counter()

21.

22. # initialize the dot

23. dot\_num = 100000000

24. dot\_in\_area = 0.0

25.

26. # initialize the boundary

27. xmin = 0

28. xmax = 10

29. ymin = 0

30. ymax = 1

31.

32. # Parallel Operation(error)

33. # pool = ThreadPool(4)

34. # pool.map(simulate\_dot\_throwing(xmin, xmax, ymin, ymax), range(1, dot\_num))

35. # pool.close()

36. # pool.join()

37.

38. # Serial Operation

39. for ind in range(1,dot\_num):

40. simulate\_dot\_throwing(xmin, xmax, ymin, ymax)

41.

42. # calculate the result of Pi

43. problem\_space\_square = (xmax - xmin) \* (ymax - ymin)

44. dot\_in\_area\_ratio = dot\_in\_area / dot\_num

45. dot\_in\_area\_square = dot\_in\_area\_ratio \* problem\_space\_square

46. PI = (dot\_in\_area\_square \* 2) \*\* 2

47.

48. duration = perf\_counter() - start

49.

50. print(f'PI 的值为 : {PI}')

51. print("the duration is {:.5f}s".format(duration))

Run result





The relative error is 0.08%, and the accuracy is 99.919% > 99.9%. Thus this method is feasible.