1.

1)

문제에 해당하는 An과 Bn에 해당하는 넓이 값을 함수로 구현하였습니다. 그래서 n에 해당하는 값을 넣어 (ex. A(10)) 함수를 호출하면 그에 해당하는 넓이 값을 반환합니다. 이 값을 통해 Pn과 절대오차를 간단한 수식으로 구할 수 있게 됩니다. 절대 오차값은 코드상에서 to 로 이름지어 구현하였습니다.

While 의 조건문으로 ~(to < tol) 를 작성함으로써 $|\rho n-\pi|<tol$ 를 만족하면 반복문을 종료하게 로직을 구성하였습니다. 먼저 n = 3 으로 시작하였으므로, 그에 해당하는 An, Bn Pn ,... 등을 미리 계산해 두고, while 문 처음의 조건문의 검사를 받습니다. 그 후 while문 내부가 동작하기 때문에 그 다음 n = 4 일 때를 검사하기 위해 n을 먼저 1증가시킨 뒤, An, Bn, Pn, ... 등을 계산해줍니다.

위의 로직은 do while문으로 구현을 하면 더욱 간결하게 구현할 수 있지만, 구글 검색을 통해 MATLAB의 do while문에 대한 문법을 찾지 못하여 이런 방식으로 구현하였습니다.

위와 같은 코드의 결과는 아래와 같습니다.

```
>>
>> Problem_1_1
절대 오차 임계값 입력 : 0.001
가장 작은 정수 n :
    72

이 때의 절대 오차값 :
     9.953394255122205e-04

>>
```

n = 72 일 때 절대 오차값은 0.000995.... 가 나오므로 while문의 0.001보다 작아졌기 때문에 반복분이 종료되고 이러한 결과가 나오게 되었습니다.

2)

이 문제에서는 위 코드에서 몇 가지를 더 추가한 형태입니다. 기본적인 로직의 형태는 같습니다. 우선, |An+1 – An| 과 |Bn+1 – Bn|의 값을 반환하는 함수인 Adf(), Bdf() 함수를 더 추가하여 만들었습니다. 다음으로 While문의 조건문을 ~(Ad < tol || Bd < tol) (Ad = Adf(n), Bd = Bdf(n)입니다.)으로 바꿈으로써 문제의 조건을 만족시킬 때 반복문을 종료하게끔 만들었습니다.

이 코드의 결과는 다음 페이지의 캡쳐본과 같습니다.

```
>> Problem_1_2
절대 오차 임계값 입력 : 0.001
가장 작은 정수 n :
    28

이 때의 절대 오차값 :
   0.006524957294038
```

n = 28 일 때 반복문을 탈출하여 원하는 값을 출력해 내는 것을 볼 수 있습니다.

1) 과 2)의 절대 오차 값을 비교했을 때 0.0009, 0.0065 임을 보았을 때, n이 적게 나온 2) 문제에서는 파이값을 근사화 하는데 1) 문제보다 효율이 떨어짐을 알 수 있었습니다

2.

1)

코드의 전반적인 구조는 x 또는 y좌표가 n의 값과 동일 할 때 while문을 중단시키고, 이 때에 몇 번을 움직여서 도달했는지의 1000번의 합을 구해 1000으로 나누어 평균을 구하였습니다.

우선 8방향을 나타내기 위해 randi([0,7])을 통해 0~7까지 난수를 구해서 각각 숫자마다 시계방향으로 북쪽부터 방향을 정해주었습니다. 이 방향을 간 뒤에는 k값을 1 증가시켜 한 번 이동을 했음을 나타내었습니다. 이렇게 x좌표 또는 y좌표가 n과 같게되어 while문을 빠져나오면, 그러기까지 이동한 총 횟수가 나옵니다. 이 횟수를 1000번 반복하여 누적시켜 평균을 내었습니다. 아래는 n이 각각 5, 10 ,20, 40일 때의 캡쳐본입니다.

```
sum =
       20907

  20.907000000000000
```

```
sum =
       76893

  76.893000000000001

>>
```

```
sum =
      323806

     3.238060000000000e+02

>>
```

```
sum =
     1273138

     1.273138000000000e+03

>>
```

위 캡쳐본을 정리하면 아래 표와 같습니다.

n	평균값
5	약 20
10	약 80
20	약 320
40	약 1270

위 표를 참조해보면, N 값이 2배 커질 때 마다 약 4배씩 평균값이 증가함을 볼 수 있습니다.

아래는 4방향일 때의 표입니다.

n	Average to Boundary
5	28.569
10	120.050
15	263.664
20	458.544
25	738.070
30	1067.283
35	1437.056
40	1860.654
45	2385.057
50	3056.605

(Results based on 1000 trials)

위 두 표를 비교해 보았을 때, 4방향과 8방향중에 8방향의 평균 이동 횟수가 좀 더 짧은 것을 확인 할 수 있었습니다.

2)

이 코드는 위 코드의 전반적인 구조는 동일 합니다. 특히 방향에 관련된 분기는 똑같습니다. 처음에 notCornerSum 과 CornerSum 변수 두 개를 선언 해 줍니다.

While 반복문을 모두 수행하면 평균 이동 횟수를 구할 수 있으므로, 이 구문을 통과 후에 x 와 y좌표의 절대값이 모두 n인지를 검사합니다. 하나라도 x, y의 좌표가 n이 아니라면 완전 코너 부분에 도달 한 것이 아니므로 notCornerSum 에 1을 더하고, 모두 n이라면 cornerSum 에 1을 더 합니다.

로봇이 타일의 끝에 도달하는 경우는 완전 코너 부분 4군데 보다 훨씬 많으므로 당연히 완전 코너 부분에 도달할 확률이 낮다고 예상이 됩니다. 아래는 작성한 코드의 결과를 캡쳐한 화면입니다. (n=5 일 때 입니다.)

```
cornerSum =
    18

   0.018000000000000

notCornerSum =
   982

   0.982000000000000
```

예상대로 완전코너 부분에 도달할 확률은 낮았지만 기대보다 훨씬 낮은 값을 보이고 있습니다.

N 의 값이 커질 경우 완전 코너 부분은 4 개뿐이지만 끝에 해당하는 위치는 더 많아지므로 더욱 더 확률이 떨어질 것임이 예상됩니다. 아래는 n = 10, n= 20 일 때의 캡쳐입니다. N = 40 일때는 완전 코너 부분에 도달할 확률이 거의 0 에 가까워 캡쳐하지 않았습니다.

```
cornerSum =
     6

   0.006000000000000

notCornerSum =
   994

   0.994000000000000
```

```
cornerSum =
     0

     0

notCornerSum =
        1000

     1
```

N = 10 일때는 완전 코너 부분에 도달할 확률이 0 일 경우가 드물었지만, N = 20 인 경우에는 결과의 대부분이 0 이 나왔고, 드물게 1 또는 2 가 나오는 정도였습니다.

위 실험결과를 통해 N 이 증가 할 때 마다 완전 코너 부분에 도달할 확률은 적어지는 것을 알게 되었습니다.

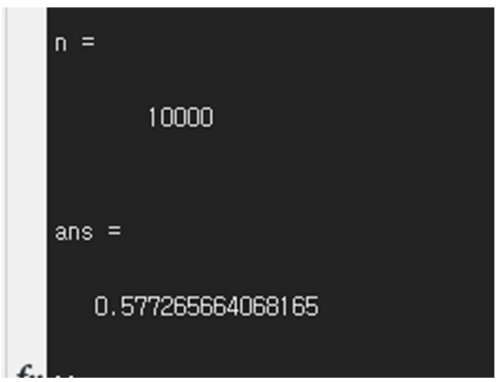
3.

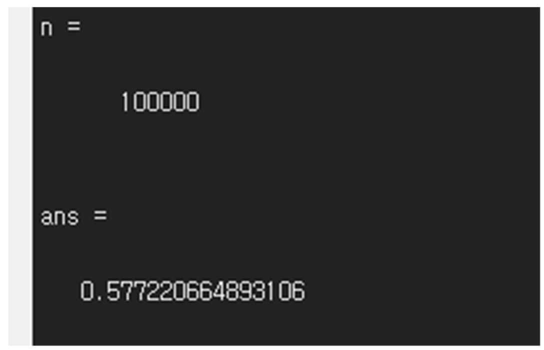
우선 코드를 작성하기 위해 En 에 해당하는 식을 함수를 통해 구현하였습니다. 이로인해 |En+1 – En| 를 반복문의 조건문으로 구현하기 매우 수월해집니다.

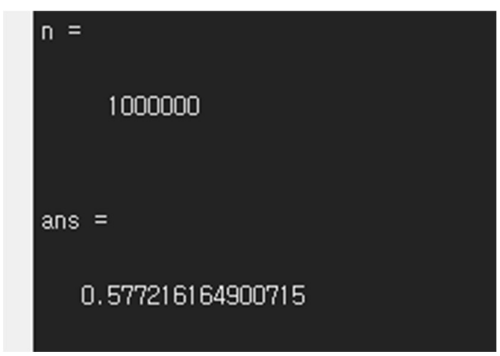
while 문의 조건은 ~(abs(E(n+1) – E(n)) < 0.001 로 설정하였습니다. 그리고 반복문 내에서는 n 의 값을 계속 1 씩 증가시킵니다

위 코드 동작결과 n = 22 의 값이 나옴을 확인하였습니다. 이 때의 En 의 값은 0.600836267039306 이 나옴을 확인하였습니다.

또, 아래 사진은 이 수식의 n 의 값을 계속 해서 늘려 어떤 값으로 수렴하는지 실험한 캡쳐본입니다.







위 캡쳐본과 같이 n 의 값이 증가할 때 마다 특정 값에 수렴하고 있음을 볼 수 있었습니다.