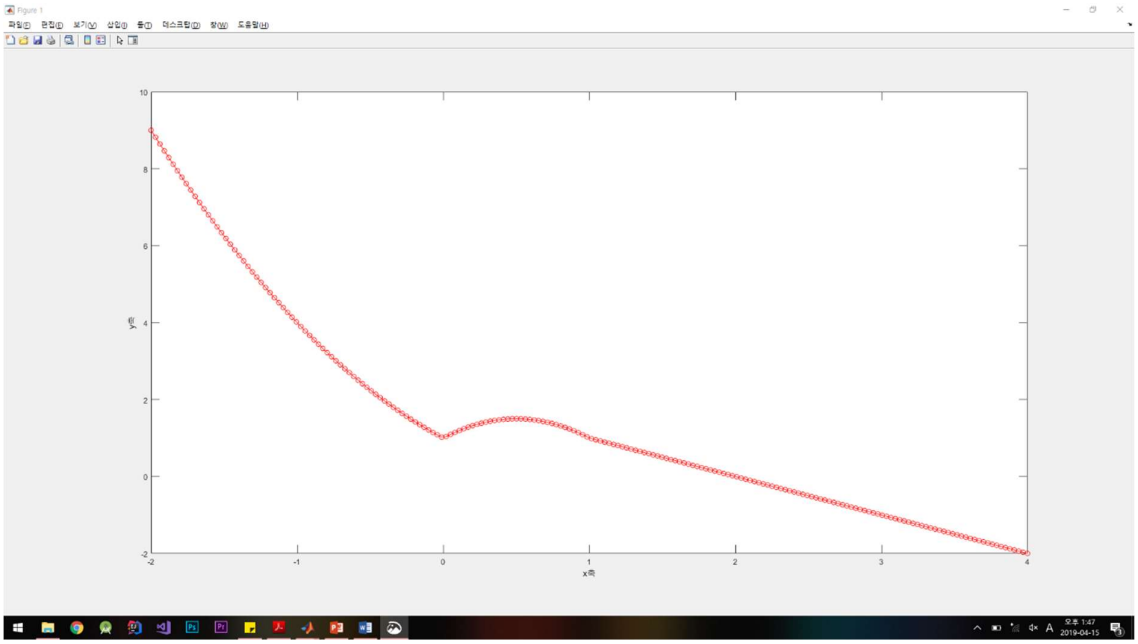
1.

우선 x값을 먼저 정의해줍니다. x값은 -2 부터 4까지 균일하게 200개의 점으로 나누어야 하니 linspace(-2, 4 200) 로 생성해 주었습니다. 그 후, y값도 x벡터 길이만큼 길이를 설정해 주어야하기 때문에 아무렇게나 y = 1 : 200 이라고 생성해줍니다. 그리고 인덱스로 접근할 값 k를 1로 초기화 해줍니다.

각 함수에서 x값에 해당하는 y값을 뽑기 위해 반복문을 선언합니다. 인덱스가 200까지이니 k는 200까지 동작하도록 while 문에 조건문을 작성합니다.

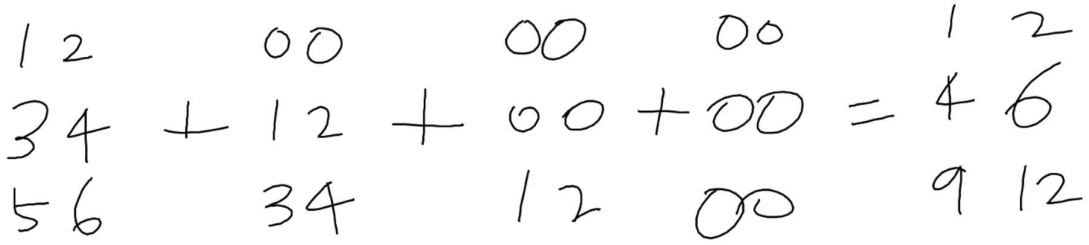
X(k) 값의 범위에 따라 y값이 바뀌기 때문에 이 반복문 안에 세개의 조건문을 작성합니다. X(k)의 범위가 각각 X(k) <= 0, 0 < X(k) <= 1, 1 < X(k) 일 때마다 다른 함수에서의 Y(k)값을 할당 할 수 있도록 만들어 줍니다.

이렇게 x값과 y값을 모두 구한 뒤, plot함수를 이용하여 그래프를 그려줍니다. 문제에서 원 형태의 마커, 빨간색 실선으로 되어있기 때문에 '-or' 옵션을 추가하여 plot(x, y, '-or') 로 그래프를 생성합니다. 또, x축과 y축의 라벨을 달아주기위해 xlabel = 'x축' ylabel = 'y축' 코드를 작성하였습니다.



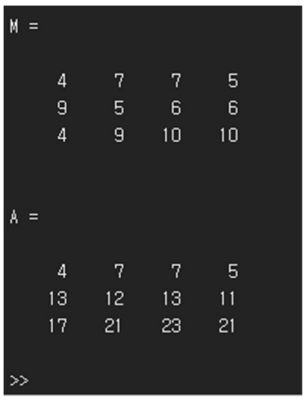
2.

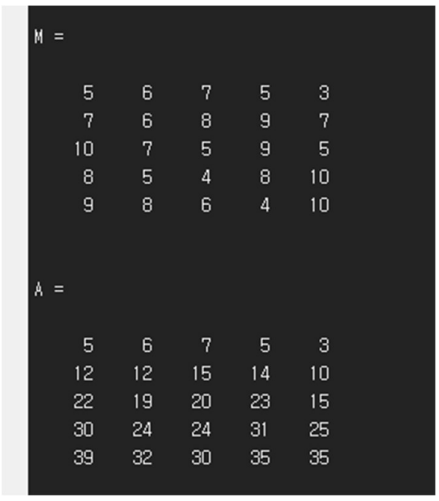
이 문제는 아래와 같은 아이디어를 기본적으로 가지고 로직을 구현하였습니다.
모든 배열 인덱스에 접근하려면 for 문을 두 번 중첩해야 했고, 이 중첩 반복문이 맘에 들지않아 계속 해서 생각해 낸 방법입니다.



첫 배열에다가 변형된 첫 배열의 덧셈을 계속 누적하되, 변형된 배열은 첫 행이 모두 0이 들어가고, 본래 마지막 행이 사라진 형태의 배열입니다.
이 덧셈을 변형된 배열의 형태가 모두 0이 될 때 까지 누적시키면, 문제에서 원하는 함수의 결과값과 같게 나오게 됩니다.

myCumsum 함수의 로직은 아래와 같습니다.
우선 인자로 받은 M의 행수와 열 수를 받습니다. 그리고, 반환해줄 A를 M의 차원과 동일하게 0으로 초기화 시켜줍니다. (A = M 대입 후 모두 0을 곱해줍니다.) 앞으로 첫 행에 채워줄 0으로만 되어있는 1행짜리 벡터를 하나 만들어줍니다. (zeroRow = 1 : col 까지 만든 뒤 모두 0을 곱하여 만들어줍니다.)
반복문은 본래 배열의 행 개수만큼만 반복해주면 충분합니다. 1 부터 row 까지만 반복합니다. 반복문 내의 로직은 아주 간단합니다. A에 M을 더하고, M을 위 그림처럼 변형시킵니다. 이 로직을 row번 반복합니다. M의 변형 형태는 첫 행에 0으로 채워져야 하니 [zeroRow ; 로 시작하며, 그 뒤에 붙일M 변형은 첫 행부터 마지막행 – 1 까지의 데이터들이므로(모든 열을 포함하여야 하므로 ':' 작성 .) M(1 : row – 1, :) 를 붙여줍니다.





3.

1)

인수로 반은 x, y를 각각 식에 대입하여 r과 theta 값을 구해냅니다. 여기서 r값은 더 이상 손봐줄 것이 없지만, theta값에 신경을 써주어야 합니다. Theta = atan(y/x)로 반환된 값을 확인하면, -pi/2 ~ pi/2 에 해당하는 값이 반환됨을 알 수 있습니다. 하지만 극좌표의 theta 값의 범위는 0 ~ 2*pi 이므로 그에 해당하는 각에 맞게 나오도록 값을 설정해주어야 합니다.

우선, tan(-a) = tan(pi – a)공식에 의해서 음수 값의 라디안은 양수 값으로 바꾸어 줄 수 있습니다. 그 변환을 위해서 theta가 음수일 때 pi를 더해주는 조건문을 추가하였습니다.

탄젠트 곡선함수의 주기는 2*pi가 아닌 pi단위이므로 이 반환 값은 0 ~ pi로 아직도 원하는 범위의 각도가 나오지 않습니다. 1, 2 사분면에 찍힌 점의 각도는 3, 4분면에 찍힌 점의 각도와 pi만큼의 차이가 발생합니다. 또한 tan(a) = tan(pi + a) 공식 또한 성립하기 때문에, y좌표 값이 음수인 경우 본래 구한 theta 값에 pi만큼을 더해주는 조건문을 추가하였습니다.

이로서 극좌표계에서 요구하는 범위의 각도를 출력할 수 있게 되었습니다.

2)

먼저, 시뮬레이션을 반복하기위해 몇 번 반복할 것인지에 대한 변수 n 을 설정합니다. 그 다음 평균값을 구하기 위해 던진 횟수의 총합을 저장할 변수를 설정합니다. 그 후 for문을 이용하여 100번의 시뮬레이션을 실행합니다.

시뮬레이션에 사용할 변수 3개를 설정합니다. 점수 총합, hits 개수, 던진 횟수입니다. 시뮬레이션은 점수가 25이상이 되거나 hits가 2번이상이 되면 멈추어야 하기 때문에 while문을 사용합니다. 이 반복 동안 -2 ~ 2 범위의 x, y좌표를 랜덤으로 구합니다. 그 후, 1) 문제에서 만들었던 함수를 이용하여 극좌표를 구합니다.

기본적으로 극좌표의 반지름 값을 이용하여 크게 세개의 if-else 문으로 조건을 나누었습니다. 점수가 10인경우 (r < 0.5), 점수가 -1 또는 -2 인 경우 (r > 1.5) 그리고 나머지인 점수가 1 ~ 8인 경우로 나누었습니다. R < 0.5 인 경우 점수를 더해줄 뿐만 아니라 hits 도 1 증가시켜줍니다. 다음 r > 1.5 인 경우 그 안에 y값의 좌표가 양수일때와 음수일때를 나누는 조건문을 더 추가해줍니다. 이로써 점수가 -1일때와 -2를 구분할 수 있게 됩니다.

위 큰 두 조건문의 나머지는 자연스럽게 점수 1~8까지의 영역이 남게 됩니다. 이 때에는 else 구문 안에 theta의 값을 비교하는 if-else 문 8개를 추가합니다. 각도에 해당하는 점수를 추가하기 위함입니다.

위 조건문들을 모두 통과하면, 던진 횟수를 1 증가시켜줍니다. 그리고 이 시뮬레이션 한 번이 끝나면, 던진 횟수의 총합에 던진 횟수를 더해줍니다.

마지막으로 던진 횟수의 총합에서 반복한 시뮬레이션의 수를 나누어 주면 평균 던진 횟수가 나오는데, 이는 약 19번으로 나타납니다. (17~21 정도의 숫자가 나옵니다.)

```
>> Problem_3_2

ans =

  16.960000000000001

>> Problem_3_2

ans =

  19.670000000000002

>> Problem_3_2

ans =

  18.039999999999999
```