EGR 103L – Fall 2019

Functions and Random Numbers

Marcus Deans (md374)
Lab Section 2, Tuesdays 11:45-2:35
22 September 2019

I understand and have adhered to all the tenets of the Duke Community Standard in completing every part of this assignment. I understand that a violation of any part of the Standard on any part of this assignment can result in failure of this assignment, failure of this course, and/or suspension from Duke University. I will swear to this.

Contents

List of Figures

1 P&E 1.31 - Time

```
Running tests for user md374
Test  1: runs: hms(61870) returns (17, 11, 10)
Test  2: runs: hms(40687) returns (11, 18,  7)
Test  3: runs: hms(46524) returns (12, 55, 24)
Test  4: runs: hms(56475) returns (15, 41, 15)
Test  5: runs: hms(20202) returns ( 5, 36, 42)
Test  6: runs: hms(49126) returns (13, 38, 46)
Test  7: runs: hms(46334) returns (12, 52, 14)
Test  8: runs: hms(75291) returns (20, 54, 51)
Test  9: runs: hms(62367) returns (17, 19, 27)
Test 10: runs: hms(75832) returns (21,  3, 52)
```

2 Reworked P&E 1.31 - More Time

```
Running tests for user md374
Test 1: total_seconds(15) returns 54000
Test 2: total_seconds( 1) returns  3600
Test 3: total_seconds( 6) returns 21600
Test 4: total_seconds(21, 57) returns 79020
Test 5: total_seconds( 4, 51) returns 17460
Test 6: total_seconds(23, 22) returns 84120
Test 7: total_seconds(10, 32,  4) returns 37924
Test 8: total_seconds( 2, 17,  1) returns  8221
Test 9: total_seconds( 9, 36, 29) returns 34589
```

3 P&E 1.35 - Triangles

```
Running tests for user md374
Test 1: runs: triangles( 1,  7,  7) returns (1.430e-01, 1.499e+00, 1.499e+00)
Test 2: runs: triangles( 6,  4,  6) returns (1.231e+00, 6.797e-01, 1.231e+00)
Test 3: runs: triangles( 5,  9,  8) returns (5.857e-01, 1.471e+00, 1.085e+00)
Test 4: runs: triangles( 3,  1,  3) returns (1.403e+00, 3.349e-01, 1.403e+00)
```

4 P&E 1.39 - Football

```
Running tests for user md374
Test  1: runs: ysp(  1,   0,   0,   0,   0) returns  39.58
Test  2: runs: ysp(  6,   6,  27,   4,   0) returns 125.00
Test  3: runs: ysp(  9,   3,  32,   0,   6) returns   5.09
Test  4: runs: ysp(  6,   3,  41,   3,   3) returns  72.22
Test  5: runs: ysp(  2,   2,  16,   0,   0) returns 100.00
Test  6: runs: ysp(  9,   3,  18,   3,   6) returns  42.36
Test  7: runs: ysp(  8,   1,   3,   1,   5) returns  39.58
Test  8: runs: ysp(  1,   0,   0,   0,   1) returns   0.00
Test  9: runs: ysp(  9,   9, 152,   6,   0) returns 158.33
Test 10: runs: ysp(  4,   4,  31,   1,   0) returns 138.54
```

5 Random Integers

```
Running tests for user md374
Test  1: passed: roll_dice(19, 12) returns
[1, 11, 7, 6, 4, 5, 9, 4, 3, 1, 7, 6, 10, 4, 2, 1, 1, 9, 4]
[4, 1, 1, 4, 1, 2, 2, 0, 2, 1, 1, 0]
Test  2: passed: roll_dice(17,  8) returns
[6, 6, 1, 2, 3, 7, 3, 2, 2, 6, 7, 3, 1, 7, 2, 1, 1]
[4, 4, 3, 0, 0, 3, 3, 0]
Test  3: passed: roll_dice( 6, 20) returns
[15, 14, 19, 3, 15, 11]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 2, 0, 0, 0, 1, 0]
Test  4: passed: roll_dice(11,  8) returns
[1, 2, 4, 1, 4, 3, 6, 2, 7, 5, 7]
[2, 2, 1, 2, 1, 1, 2, 0]
Test  5: passed: roll_dice( 7,  8) returns
[6, 2, 3, 1, 5, 5, 4]
[1, 1, 1, 1, 2, 1, 0, 0]
Test  6: passed: roll_dice(19, 10) returns
[1, 2, 8, 6, 4, 2, 8, 2, 5, 1, 8, 2, 6, 1, 8, 5, 7, 8, 3]
[3, 4, 1, 1, 2, 2, 1, 5, 0, 0]
Test  7: passed: roll_dice(16, 20) returns
[15, 10, 15, 3, 10, 14, 11, 5, 16, 11, 10, 3, 19, 2, 18, 9]
[0, 1, 2, 0, 1, 0, 0, 0, 1, 3, 2, 0, 0, 1, 2, 1, 0, 1, 1, 0]
Test  8: passed: roll_dice( 9, 20) returns
[8, 9, 8, 3, 1, 17, 12, 6, 11]
[1, 0, 1, 0, 0, 1, 0, 2, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0]
Test  9: passed: roll_dice( 7, 10) returns
[4, 9, 3, 2, 8, 2, 6]
[0, 2, 1, 1, 0, 1, 0, 1, 1, 0]
Test 10: passed: roll_dice(20, 20) returns
[13, 10, 18, 16, 16, 7, 14, 8, 19, 9, 11, 12, 13, 5, 15, 3, 19, 11, 15, 19]
[0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 2, 1, 2, 1, 2, 2, 0, 1, 3, 0]
```

6 Random Numbers

```
NetID: md374

How many random numbers?: 10000
Uniform: min: 6.24e-05 avg: 5.00e-01 max: 1.00e+00
Normal: min: -4.85e+00 avg: 1.02e-02 max: 3.39e+00
```

A Codes

A.1 keep_time.py

```
# -*- coding: utf-8 -*-
"""
[Keep Time]
[Marcus Deans]
[10 September 2019]

I understand and have adhered to all the tenets of the Duke Community Standard
in creating this code.
Signed: [md374]
"""

# %% get inputs
print('Number of Seconds?: ')
usertime = int(input())
# %% Define function
def hms(intime):

    hours = intime//3600
    intime -= (hours*3600)
    minutes = intime//60
    intime -= (minutes*60)
    seconds = intime
    return hours, minutes, seconds

# %% Tests

print(hms(usertime))
#if __name__ == "__main__":
#    print(hms(0))
#    print(hms(1))
#    print(hms(45))
#    print(hms(60))
#    print(hms(75))
#    print(hms(3600))
#    print(hms(3675))
```

A.2 get_time.py

```
# -*- coding: utf-8 -*-
"""
[Get Time]
[Marcus Deans]
[10 September 2019]

I understand and have adhered to all the tenets of the Duke Community Standard
in creating this code.
Signed: [md374]
"""
# %% get inputs
y = 0
z = 0
x, y, z = [int(x) for x in input("Number of Hours, Minutes, and Seconds: ").split()]
# print (x, y, z, sep='/')
# %% Define function
```

```
17 def total_seconds(hrs, mins=0, secs=0): # does not work yet - inputs not right
18
19     final = 0
20     final += hrs*3600
21     final += mins*60
22     final += secs
23     return final
24
25 # %% test
26 print(total_seconds(x, y, z))
27 #print(total_seconds(1))
28 #print(total_seconds(1, 2))
29 #print(total_seconds(1, 2, 3))
30 #
31 if __name__ == "__main__":
32     print(total_seconds(1))
33     print(total_seconds(1, 2))
34     print(total_seconds(1, 2, 3))
```

A.3 tri_cal.py

```
1  # -*- coding: utf-8 -*-
2  """
3  [Triangle Calculator]
4  [Marcus Deans]
5  [12 September 2019]
6
7  I understand and have adhered to all the tenets of the Duke Community Standard
8  in creating this code.
9  Signed: [md374]
10 """
11 # %% Import modules
12 import math as m
13 import matplotlib.pyplot as plt
14
15 # %% Define function
16
17
18 def triangles(a, b, c, draw=False, fnum=1):
19     # %% Calculate angles
20     A = m.acos(((b**2)+(c**2)-(a**2))/(2*b*c))
21     B = m.acos(((a**2)+(c**2)-(b**2))/(2*a*c))
22     C = m.acos(((a**2)+(b**2)-(c**2))/(2*a*b))
23     # %% Make plot if asked
24     if draw:
25         fig, ax = plt.subplots(num=fnum, clear=True)
26         #plt.plot([0, 1, 0], [0,1,2])
27         width = (c*(m.cos(B)))
28         height = (c*(m.sin(B)))
29         plt.plot([0, a, width, 0], [0, 0, height, 0], '-r')
30         # Calculations and plots
31
32         ax.axis('equal')
33         fig.tight_layout()
34
35     # %% Return angles
36     return A, B, C
37
```

```
38
39 if __name__ == "__main__":
40     print(triangles(3, 7, 4))
41     print(triangles(3, 4, 5))
42     print(triangles(3, 6, 4, True, 5))
```

A.4 football.py

```
1 # -*- coding: utf-8 -*-
2 """
3 [Football Ranking]
4 [Marcus Deans]
5 [12 September 2019]
6
7 I understand and have adhered to all the tenets of the Duke Community Standard
8 in creating this code.
9 Signed: [md374]
10 """
11 import numpy as np
12 # %% Define function
13 def you_shall_pass(pa, pc, py, td, intr):
14     charlie = bounded(float(((100*(pc/pa))-30)/20))
15     tango = bounded(float(20*(td/pa)))
16     yankee = bounded(float(((py/pa)-3)/4))
17     india = bounded(float(2.375-(25*(intr/pa))))
18
19     ranking = (((charlie + tango + yankee + india)*100)/6)
20     return ranking
21
22 # %% Function to return bounded value
23 def bounded(x, low=0, high=2.375):
24     x = np.clip(x, 0, 2.375)
25     return x  # fix this
26
27 if __name__ == "__main__":
28     print(round((you_shall_pass(30, 20, 286, 3, 0)),1))
29     print(round((you_shall_pass(591, 398, 4377, 24, 9)),1))
30     print(round((you_shall_pass(32, 25, 405, 4, 0)),1))
```

A.5 play_game.py

```
1 # -*- coding: utf-8 -*-
2 """
3 [Play Game]
4 [Marcus Deans]
5 [12 September 2019]
6
7 I understand and have adhered to all the tenets of the Duke Community Standard
8 in creating this code.
9 Signed: [md374]
10 """
11 # %% Import modules
12 import numpy as np
13
14 # %% Define function
15 def roll_dice(n_dice=1, n_sides=6, seed=0):
16     np.random.seed(seed) #fill seed
17     first = [] #create string for rolls
```

```
18      quant = [] #create string for quantities
19      for m in range(n_sides): #populate quantity array to corrrect size
20          quant.append(0) #intialize each quantity at 0
21      for a in range(n_dice): #run loop to fill string
22          value = np.random.randint(1,n_sides) #get a random number based on n_sides
23          first.append(value) #add value to string
24          quant[(value-1)] += 1 #add to the respective counter
25      return first, quant #return string of values and quantities
26
27 # %% Testing
28 #if __name__ == "__main__":
29 #     print(roll_dice(10, 6))
30 #     print(roll_dice(9, 12, 2))
```

A.6 gen_rand.py

```
1  # -*- coding: utf-8 -*-
2  """
3  [Generating Random Numbers]
4  [Marcus Deans]
5  [12 September 2019]
6
7  I understand and have adhered to all the tenets of the Duke Community Standard
8  in creating this code.
9  Signed: [md374]
10 """
11
12 # %% Import modules
13 import numpy as np
14 import matplotlib.pyplot as plt
15 import math as m
16
17 # %% Seed based on NetID
18 NetID = input('NetID: ')
19 seed = 0
20 for code in map(ord, NetID):
21     seed = seed + code
22
23 np.random.seed(seed)
24
25 # %% Number of numbers
26 nums = int(input("How many random numbers?: "))  # Remove 1000 and put your code
       here
27
28 # %%Calculate distributions
29 u_d = np.random.uniform(0, 1, nums)  # Remove wrong command and fix
30 n_d = np.random.normal(0, 1, nums)  # Remove wrong command and fix
31
32 # %% Make plots
33 num_bins = m.ceil(10 * m.log10(nums))
34
35 fig, ax = plt.subplots(num=1, clear=True)
36 ax.hist(u_d, num_bins)
37 ax.set_title('Uniform')
38 fig.tight_layout()
39 fig.savefig('UniformPlot.eps')
40
41 fig, ax = plt.subplots(num=2, clear=True)
```

```
42 ax.hist(n_d, num_bins)
43 ax.set_title('Normal')
44 fig.tight_layout()
45 fig.savefig('NormalPlot.eps')
46
47 # %% Print statistics
48 # Your code here
49 uniform_min = '{:.2e}'.format(min(u_d))
50 uniform_avg = '{:.2e}'.format(np.mean(u_d))
51 uniform_max = '{:.2e}'.format(max(u_d))
52 print("Uniform: min:", uniform_min, "avg:", uniform_avg,"max:", uniform_max, sep=" "
       )
53
54 normal_min = '{:.2e}'.format(min(n_d))
55 normal_avg = '{:.2e}'.format(np.mean(n_d))
56 normal_max = '{:.2e}'.format(max(n_d))
57 print("Normal: min:", normal_min, "avg:", normal_avg,"max:", normal_max, sep=" ")
```

B Figures

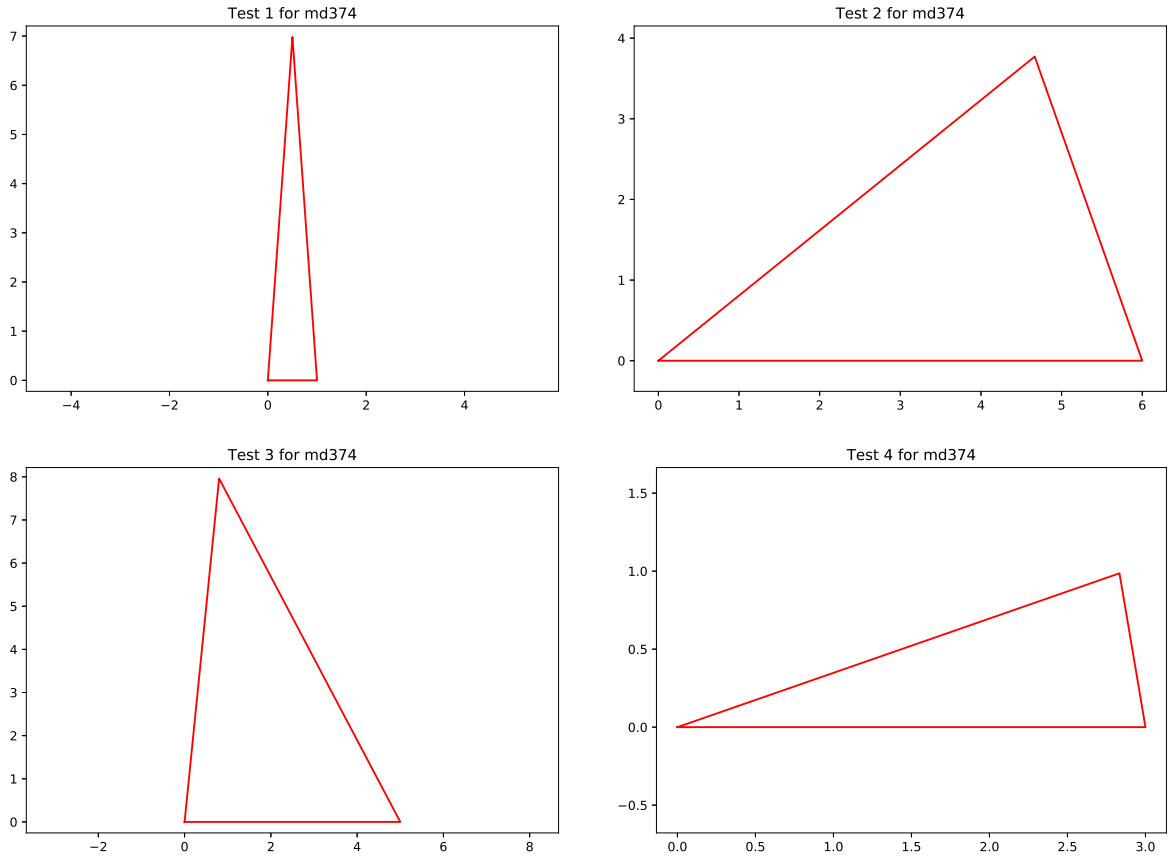

Figure 1: Test Triangles.

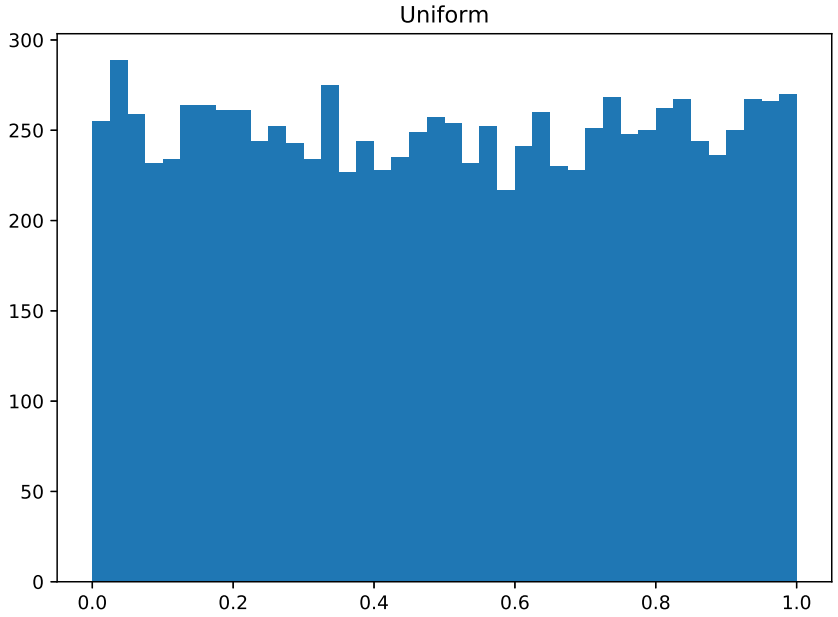

Figure 2: Histogram of Uniformly Distributed Random Numbers.

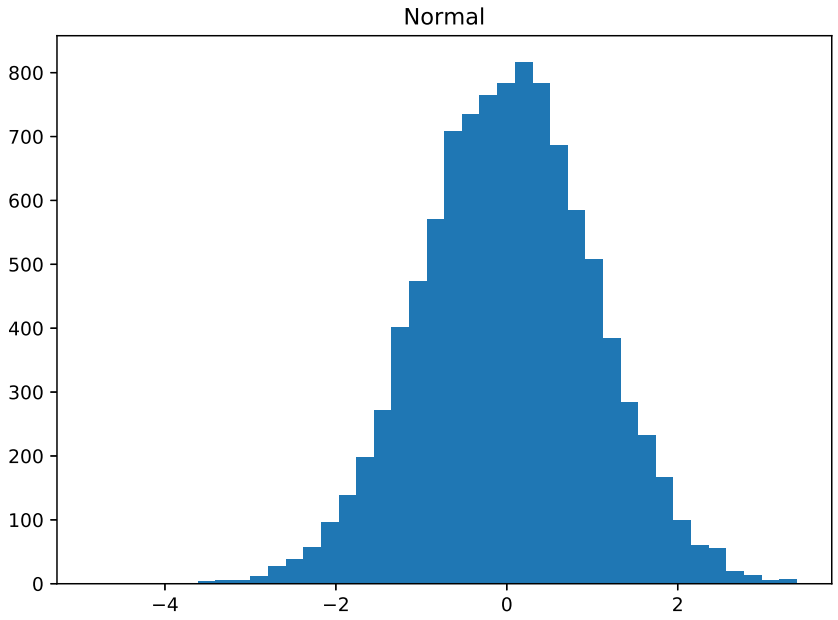

Figure 3: Histogram of Normally Distributed Random Numbers.