

# 移動情報ネットワーク特論

## レポート課題4

電気情報工学専攻 情報工学コース  
F20C004C 太田剛史

## 目次

- [目次](#)
- [はじめに](#)
- [シミュレーションプログラムの作成](#)
- [理論計算のプログラムの作成](#)
- [結果](#)
- [考察](#)

## はじめに

以下にシミュレーションのスク립トを記すが, Github上にシミュレーションのスク립トと描画のためのスク립ト, レポート作成に利用したmarkdownなどを載せてあるため, ネットワーク環境がある場合は下記のURLを参照していただきたい.

[https://github.com/haru1843/mobile\\_info\\_network\\_rep02](https://github.com/haru1843/mobile_info_network_rep02)

@@  
@@

# シミュレーションプログラムの作成

ノードAとノードBの間に $n$ 個のノードが一様かつ独立に分布している場合に、ノードAとノードBが連結可能である確率 $p_n(t)$ を求めるシミュレーションプログラムを作成する。

また今回のシミュレーションにおける各パラメータは以下の通りである。

パラメータ	内容	値
t	ノードAとノードBの距離	100.0
d	電波の届く距離	20.0
n	ノードAとノードBの間にあるノード数	1, 2, 3, ..., 56
-	シミュレーションの実行回数	1e6

プログラムの作成に利用した言語は [Python 3.6.9](#) である。下記にプログラムを記す。

```

1  import numpy as np
2  import os.path as path
3  from typing import List
4  import json
5
6
7  class Simulator:
8      def __init__(self, dist: float, node_num: int, allow_dist: float):
9          """
10             Params
11             -----
12             dist: float
13                 ノードAとノードB間の距離
14             node_num: int
15                 ノードAとノードB間に存在するノード数。(ノードAとノードBを含めない)
16             allow_dist: float
17                 ノード間の通信可能距離
18             """
19             self.dist: float = dist
20             self.node_num: int = node_num
21             self.allow_dist: float = allow_dist
22
23     def run(self, try_num: int = 100000):
24         """
25             課題のシミュレーションを実行し、実行回数に対する通信可能であった割合を返す。
26
27             Param
28             -----
29             try_num: int (default=100000)
30                 シミュレーションの実行回数
31
32             Return
33             -----
34             availability_ratio: float
35                 実行回数に対して、通信可能であった割合。(0~1)
36             """
37             node_mat = np.sort(self.dist * np.random.rand(try_num, self.node_num), axis=1)
38             head = np.zeros((try_num, 1))
39             tail = np.full((try_num, 1), self.dist)

```

```
40         return np.sum(np.sum(
41             np.diff(np.concatenate([head, node_mat, tail], axis=1), axis=1) > self.allow_dist,
42             axis=1
43         ) == 0) / try_num
44
45
46 def main():
47     # 全体のパラメータの設定
48     try_num: int = int(1e6)
49     total_distance: float = 100.0
50     allowable_distance: float = 20.0
51
52     node_num_list: List[int] = list(range(1, 57))
53
54     result_list: List[float] = [0] * len(node_num_list)
55
56     # 各ノード数に対するシミュレーションの実行
57     for idx, node_num in enumerate(node_num_list):
58         sim = Simulator(
59             dist=total_distance,
60             node_num=node_num,
61             allow_dist=allowable_distance
62         )
63         result_list[idx] = sim.run(try_num=try_num)
64
65     # データの保存
66     with open("./output/result.json", mode="w") as f:
67         json.dump({
68             "param": {
69                 "try_num": try_num,
70                 "total_distance": total_distance,
71                 "allowable_distance": allowable_distance,
72             },
73             "x": node_num_list,
74             "y": result_list
75         }, f)
76
77
78 if __name__ == "__main__":
79     main()
80
81
```

# 理論計算のプログラムの作成

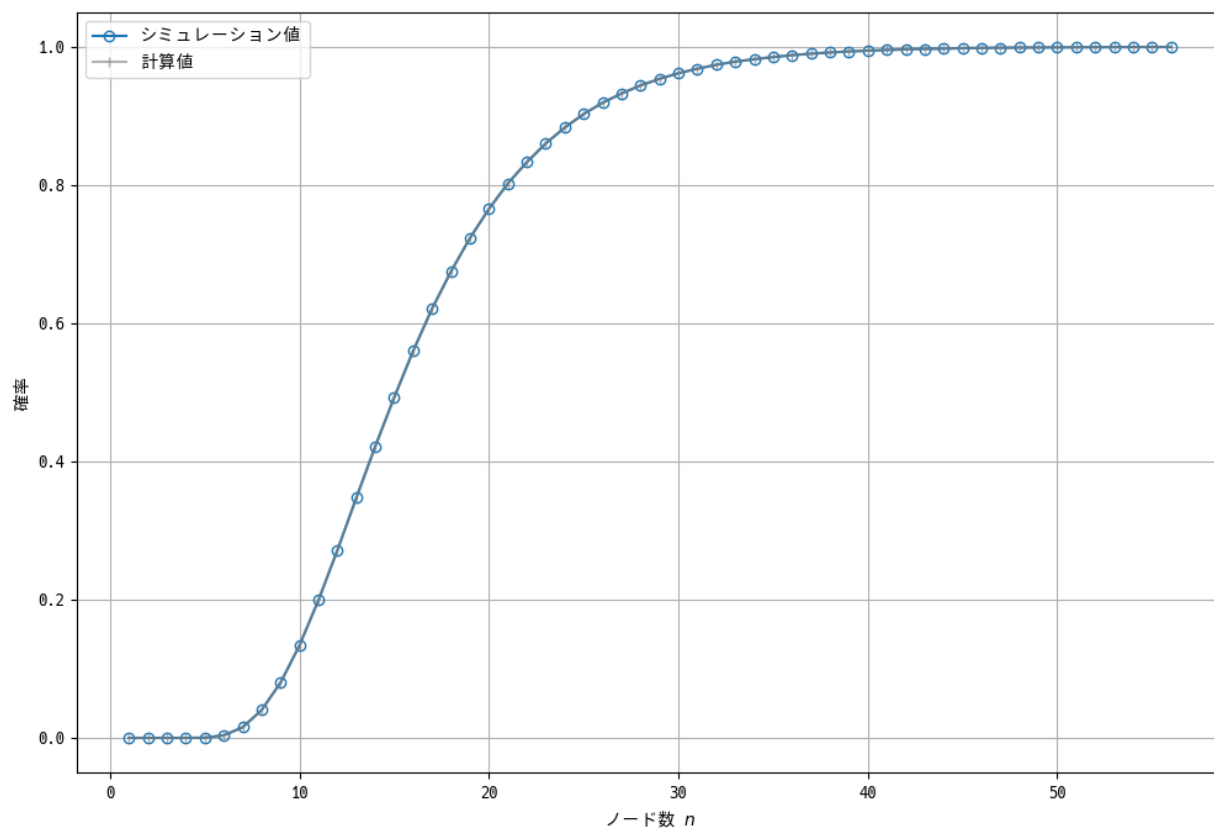
理論計算に利用したプログラムを下記に記す.

また, プログラムの作成に利用した言語は [Python 3.6.9](#) である.

```
1  from scipy.special import comb
2  import numpy as np
3  import json
4
5
6  def pr(n: int, k: int = 0, t: float = 100.0, d: float = 20.0):
7      return comb(n+1, k) * np.sum([
8          ((-1) ** (m-k)) * comb(n+1-k, m-k) * (np.maximum(0, 1-(m*d/t)))**n
9          for m in range(k, (n+1)+1)
10     ])
11
12
13 def main():
14     total_distance: float = 100.0
15     allowable_distance: float = 20.0
16     k: int = 0
17
18     node_num_list: List[int] = list(range(1, 57))
19     result_list = [
20         pr(
21             n=n,
22             k=k,
23             t=total_distance,
24             d=allowable_distance
25         )
26         for n in node_num_list]
27
28     with open("./output/calc_result.json", mode="w") as f:
29         json.dump({
30             "param": {
31                 "total_distance": total_distance,
32                 "allowable_distance": allowable_distance,
33                 "k": k,
34             },
35             "x": node_num_list,
36             "y": result_list
37         }, f, indent=2)
38
39
40 if __name__ == "__main__":
41     main()
42
```

## 結果

上記までのプログラムを用いて算出した結果を、横軸をノード数 $n$ 、縦軸を連結可能である確率としたグラフを作成し、以下に示す。



## 考察

シミュレーション値と理論計算値がほぼ一致していることがわかる。

ノード数が少ないときに確率が0となってしまうことに関して、今回のシミュレーションではノードA/B間の距離が 100.0 であり、電波の届く距離が 20.0 であるため、最低でもノード数が5つないとノードA/B間が連結できないためこのようになる。