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

はじめに

以下にシミュレーションのスクリプトを記すが, Github上にシミュレーションのスクリプトと描画のためのスクリプト, 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_rep04

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 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
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	
d	電波の届く距離	
n	ノードAとノードBの間にあるノード数	
-	シミュレーションの実行回数	

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.

```python
import numpy as np
import os.path as path
from typing import List
import json


class Simulator:
    def __init__(self, dist: float, node_num: int, allow_dist: float):
        """
        Params
        ----------------------------------
        dist: float
            ノードAとノードB間の距離
        node_num: int
            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)
        allow_dist: float
            ノード間の通信可能距離
        """
        self.dist: float = dist
        self.node_num: int = node_num
        self.allow_dist: float = allow_dist

    def run(self, try_num: int = 100000):
        """
        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.

        Param
        ----------------------------------
        try_num: int (default=100000)
            シミュレーションの実行回数

        Return
        ----------------------------------
        availability_ratio: float
            実行回数に対して，通信可能であった割合.(0~1)
        """
        node_mat = np.sort(self.dist * np.random.rand(try_num, self.node_num), axis=1)
        head = np.zeros((try_num, 1))
        tail = np.full((try_num, 1), self.dist)

```

```
        return np.sum(np.sum(
            np.diff(np.concatenate([head, node_mat, tail], axis=1), axis=1) > self.allow_dist,
            axis=1
        ) == 0) / try_num


def main():
    # 全体のパラメータの設定
    try_num: int = int(1e6)
    total_distance: float = 100.0
    allowable_distance: float = 20.0

    node_num_list: List[int] = list(range(1, 57))

    result_list: List[float] = [0] * len(node_num_list)

    # 各ノード数に対するシミュレーションの実行
    for idx, node_num in enumerate(node_num_list):
        sim = Simulator(
            dist=total_distance,
            node_num=node_num,
            allow_dist=allowable_distance
        )
        result_list[idx] = sim.run(try_num=try_num)

    # データの保存
    with open("./output/result.json", mode="w") as f:
        json.dump({
            "param": {
                "try_num": try_num,
                "total_distance": total_distance,
                "allowable_distance": allowable_distance,
            },
            "x": node_num_list,
            "y": result_list
        }, f)


if __name__ == "__main__":
    main()

```

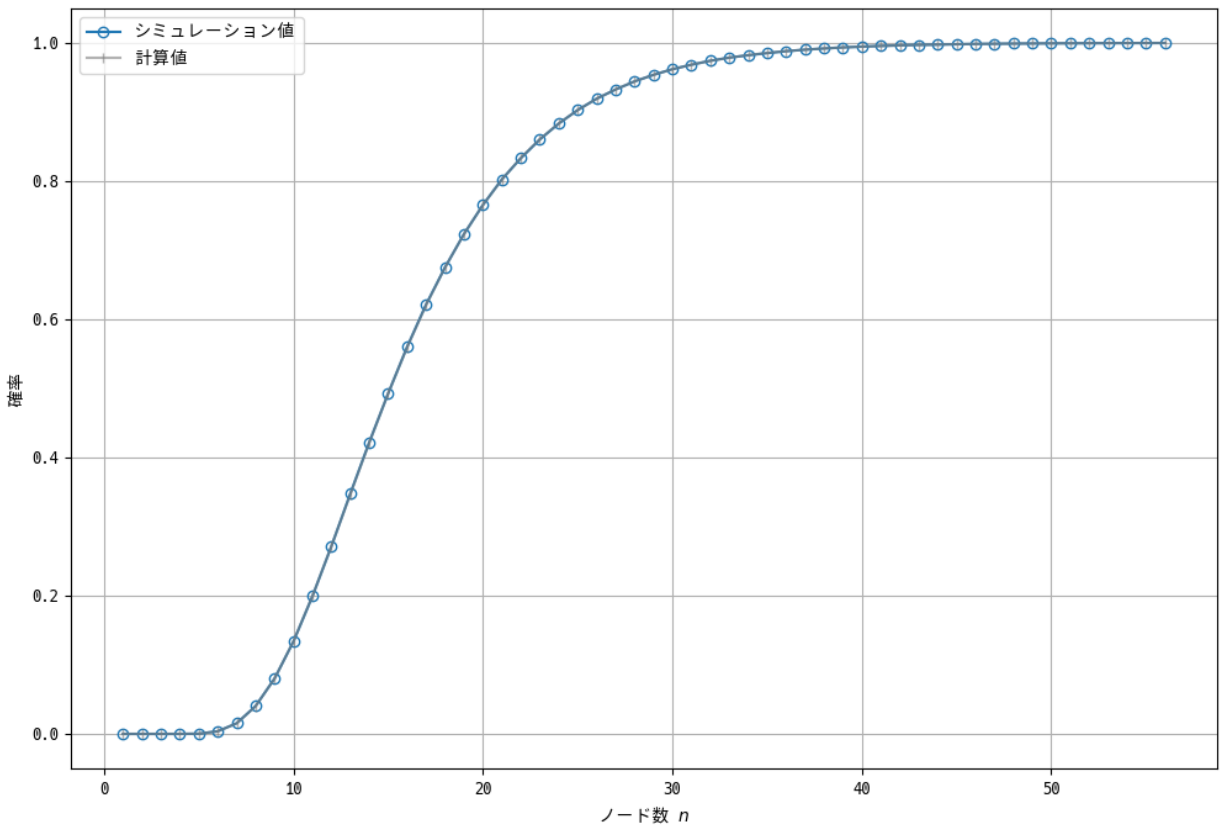
理論計算のプログラムの作成

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.

```python
from scipy.special import comb
import numpy as np
import json


def pr(n: int, k: int = 0, t: float = 100.0, d: float = 20.0):
    return comb(n+1, k) * np.sum([
        ((-1) ** (m-k)) * comb(n+1-k, m-k) * (np.maximum(0, 1-(m*d/t))**n)
        for m in range(k, (n+1)+1)
    ])


def main():
    total_distance: float = 100.0
    allowable_distance: float = 20.0
    k: int = 0

    node_num_list: List[int] = list(range(1, 57))
    result_list = [
        pr(
            n=n,
            k=k,
            t=total_distance,
            d=allowable_distance
        )
        for n in node_num_list]

    with open("./output/calc_result.json", mode="w") as f:
        json.dump({
            "param": {
                "total_distance": total_distance,
                "allowable_distance": allowable_distance,
                "k": k,
            },
            "x": node_num_list,
            "y": result_list
        }, f, indent=2)


if __name__ == "__main__":
    main()

```

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 , 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となってイルことに関して, 今回のシミュレーションではノードA/B間の距離が であり, 電波の届く距離が 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.