ndlib による複雑ネットワーク上の拡散プロセスシミュレーション

実行可能サンプル集付きチュートリアル

河合 勝彦 *

2026 年 4 月 29 日

概要

本チュートリアルは, NetworkX をベースにした拡散プロセスシミュレーションライブラリ **ndlib** の入門資料である. 公式ドキュメント[*1] の構成を踏まえながら, 経済学・社会科学で利用する際に必要になるモデルの考え方, 実行手順, パラメータ設定, 結果の読み取り方を日本語で整理した. 扱う対象は, 基本的なネットワーク生成から, SI/SIS/SIR/SEIR/SEIS/SWIR などの疫学モデル, 閾値・カスケードモデル, 意見ダイナミクス, 動的ネットワーク, Composite モデル, 解析・可視化までを含む. 全コードは `uv` で実行可能な PEP 723 形式のスタンドアロンスクリプトとして同梱し, すべての主要モデルについて実行結果の図表を本文に挿入した. 読者は PDF でモデルの全体像を確認しながら, 同じスクリプトを手元で実行・改変できる.

目次

* 名古屋市立大学大学院経済学研究科 kkawai@econ.nagoya-cu.ac.jp

[*1] https://ndlib.readthedocs.io/en/latest/

1　はじめに

複雑ネットワーク上では, ウイルス感染, 流言, イノベーションの採用, SNS 上の意見変化のような多様な現象が,「ノードの状態が近傍を介して変わる」という共通構造を持つ. ndlib (Network Diffusion library) は, そうした拡散プロセスを宣言的に記述・シミュレーションするための Python パッケージである. 主な特徴は以下のとおりである.

- 30 種類を超える既製モデル (SI, SIS, SIR, SEIR, Voter, Hegselmann – Krause, …)
- NetworkX オブジェクトをそのまま入力として受け取る
- ノード単位・エッジ単位でパラメータを差し替え可能
- Composite モデル機構によるカスタムダイナミクスの構築
- matplotlib / Bokeh による標準可視化
- 動的ネットワーク (`dynetx`) 上のシミュレーション

■本書の使い方　全コード例はリポジトリ `scripts/` 以下に配置されている. 本文中のコードは PDF に埋め込まれているが, 同じファイルがスタンドアロンで実行可能である. まずインストール確認と基本構造を読み, その後に各モデルの説明・コード・結果図を対応させて確認するとよい. 動作確認した後にパラメータを書き換えれば, 自分の研究テーマに合わせた小さな実験へ発展させられる. 個々のスクリプトはリポジトリのルートディレクトリから `uv run scripts/XX_name.py` の形で実行することを想定している.

■用語について　ndlib の多くのモデルでは, 疫学モデル以外でも状態名として `Susceptible`, `Infected`, `Removed` が使われる. 本書では, 病気の文脈では「感染」と呼び, 行動・流行・意見の文脈では「採用」「活性化」「意見」と読み替える. コード中の状態名は ndlib の API に合わせてそのまま示す.

2　インストールと実行環境

本チュートリアルでは Astral 社の `uv` を用いて Python 依存関係を扱う. `uv` は単一ファイルの先頭に PEP 723 形式のメタデータを書いておくと, 依存パッケージを一時環境に解決してそのままスクリプトを起動できる.

2.1　uv のインストール

```
# Linux / macOS
curl -LsSf https://astral.sh/uv/install.sh | sh

# Windows (PowerShell)
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

2.2 PEP 723 形式のスクリプト

本書のサンプルは, ファイル先頭に以下のような**インラインスクリプトメタデータ**を持つ.

```
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
```

このコメントブロックを `uv` が解釈し, 必要なパッケージを一時環境に解決してから Python を呼び出す. したがって `pip install ndlib` をグローバル環境に実行する必要はない. 初回実行時だけ依存関係のダウンロードに時間がかかる.

2.3 動作確認

最初のスクリプトを動かしてみよう.

```
uv run scripts/01_install_check.py
```

ソース: `scripts/01_install_check.py`

```
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "six",
# ]
# ///
"""01_install_check.py — ndlib のインストール確認

実行方法:
    uv run scripts/01_install_check.py

ndlib と NetworkX が正しくインストールされているか確認し,
小規模ネットワーク上で SI モデルを 5 ステップだけ走らせます.

参考文献:
    [1] Rossetti, G., Milli, L., Rinzivillo, S., Sirbu, A., Pedreschi, D., &
        Giannotti, F. (2018). NDlib: a python library to model and analyze
        diffusion processes over complex networks. Int. J. Data Sci. Anal.
    [2] ndlib 公式 Tutorial: https://ndlib.readthedocs.io/en/latest/tutorial.html
"""
from __future__ import annotations

from importlib.metadata import version

import networkx as nx
import ndlib
import ndlib.models.ModelConfig as mc
```

```python
import ndlib.models.epidemics as ep


def main() -> None:
    print(f"NetworkX version: {nx.__version__}")
    print(f"ndlib version   : {version('ndlib')}")

    g = nx.erdos_renyi_graph(n=50, p=0.1, seed=0)
    model = ep.SIModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.05)
    cfg.add_model_parameter("fraction_infected", 0.1)
    model.set_initial_status(cfg)

    iters = model.iteration_bunch(5)
    for it in iters:
        node_count = it["node_count"]
        print(f"step={it['iteration']:>2}  S={node_count[0]:>3}  I={node_count[1]:>3}")


if __name__ == "__main__":
    main()
```

3　ndlib の基本構造

ndlib のシミュレーションは, 基本的には以下の 4 ステップで進む.

1. NetworkX (もしくは `dynetx`) でグラフ `g` を作る
2. モデルを生成する: `model = ep.SIRModel(g)`
3. `ModelConfig.Configuration()` を作り, パラメータと初期条件を設定してモデルへ渡す
4. `model.iteration_bunch(N)` で N ステップ実行し, `build_trends` で集計データを得る

■Configuration の主な API

- `add_model_parameter(name, value)`: モデル全体に作用するパラメータ (β, γ, ...) を設定
- `add_model_initial_configuration(name, list)`: 初期感染ノードの集合 (`Infected`) やブロックノード集合 (`Blocked`) を直接指定
- `add_node_configuration(name, node, value)`: ノード単位の値
- `add_edge_configuration(name, edge, value)`: エッジ単位の値

■パラメータと初期状態の考え方　`fraction_infected` は, 初期状態で感染済み・採用済み・活性化済みのノード割合をランダムに指定するためのモデルパラメータである. 特定のノードを起点にしたい場合は, `add_model_initial_configuration` で状態名 `Infected` とノード集合を明示する. 明示指定は割合指定より優先されるため, ハブから始める場合と周辺ノードから始める場合を比較しやすい. また, 多くのモデルクラスは `seed` 引数を受け取れるが, NetworkX のグラフ生成 seed と ndlib の状態遷移 seed は別物である. 完全に同じ軌跡を再現したいときは, グラフ生成とモデル生成

の両方で乱数 seed を管理する.

■iteration_bunch の戻り値 リストの各要素は `{iteration, status, status_delta, node_count}` を含む辞書である. `node_count` は状態 ID から個数への辞書で, たとえば SIR では `0:S, 1:I, 2:R` を指す. 状態 ID はモデルごとに異なるので, 厳密には `model.get_status_map()` で確認する. `status_delta` は各状態の人数変化であり, `status` は個別ノードの状態を追うための辞書である. 連続値意見モデルや最終状態のネットワーク描画では `iteration_bunch(N, node_status=True)` を使い, 更新されなかったノードは直前値を引き継ぐように復元すると扱いやすい.

4 ネットワークの作成

ndlib の静的モデルは NetworkX のグラフを入力に取る. ランダムグラフ, スケールフリーネットワーク, スモールワールド, 実ネットワーク (Karate Club など) を生成して描画する例を示す. 動的ネットワークを扱う場合は, 後述する `dynetx` の時系列グラフを用いる.

ソース: `scripts/02_network_basics.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "networkx",
#     "matplotlib",
# ]
# ///
"""02_network_basics.py — ndlib で利用する典型的ネットワークの生成

実行方法:
    uv run scripts/02_network_basics.py

NetworkX で生成できる代表的なランダムグラフを比較表示します.
ndlib のモデルはどれも NetworkX のグラフを入力として受け取るため,
基礎となるグラフ生成 API を押さえておくと便利です.

参考文献:
    [1] Erdős, P. & Rényi, A. (1959). On random graphs I.
        Publicationes Mathematicae 6, 290–297.
    [2] Watts, D. J. & Strogatz, S. H. (1998). Collective dynamics of
        'small-world' networks. Nature 393, 440–442.
    [3] Barabási, A.-L. & Albert, R. (1999). Emergence of scaling in
        random networks. Science 286, 509–512.
    [4] Zachary, W. W. (1977). An information flow model for conflict and
        fission in small groups. J. Anthropological Research 33, 452–473.
    [5] NetworkX random graphs:
        https://networkx.org/documentation/stable/reference/generators.html
"""
from __future__ import annotations

import matplotlib.pyplot as plt
import networkx as nx


```

```python
def describe(name: str, g: nx.Graph) -> None:
    n, m = g.number_of_nodes(), g.number_of_edges()
    avg_deg = sum(dict(g.degree()).values()) / n
    print(f"{name:<22} N={n:<5} E={m:<6} <k>={avg_deg:5.2f}")


def main() -> None:
    seed = 42
    graphs = {
        "Erdos-Renyi G(n,p)": nx.erdos_renyi_graph(200, 0.05, seed=seed),
        "Barabasi-Albert"   : nx.barabasi_albert_graph(200, 3, seed=seed),
        "Watts-Strogatz"    : nx.watts_strogatz_graph(200, 6, 0.1, seed=seed),
        "Karate Club"       : nx.karate_club_graph(),
    }

    for name, g in graphs.items():
        describe(name, g)

    fig, axes = plt.subplots(2, 2, figsize=(9, 9))
    for ax, (name, g) in zip(axes.flat, graphs.items()):
        nx.draw_spring(g, ax=ax, node_size=20, with_labels=False, alpha=0.7)
        ax.set_title(name)
    fig.tight_layout()
    fig.savefig("figures/02_network_basics.png", dpi=120)
    print("Saved: figures/02_network_basics.png")


if __name__ == "__main__":
    main()
```

5　疫学モデル

5.1　SI モデル

最もシンプルな感染症モデルである．感染者は永久に感染状態にとどまり，感受性者は近傍の感染者から確率 β で感染する．回復状態がないため，感染者数は単調に増加する．

ソース: scripts/03_si_model.py

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""03_si_model.py — SI(Susceptible-Infected) モデル

実行方法:
    uv run scripts/03_si_model.py

最も単純な感染症モデルである SI モデルを実行します．
感染した個体は永久に感染状態のままで，未感染者と接触すると確率 beta で感染します．

```

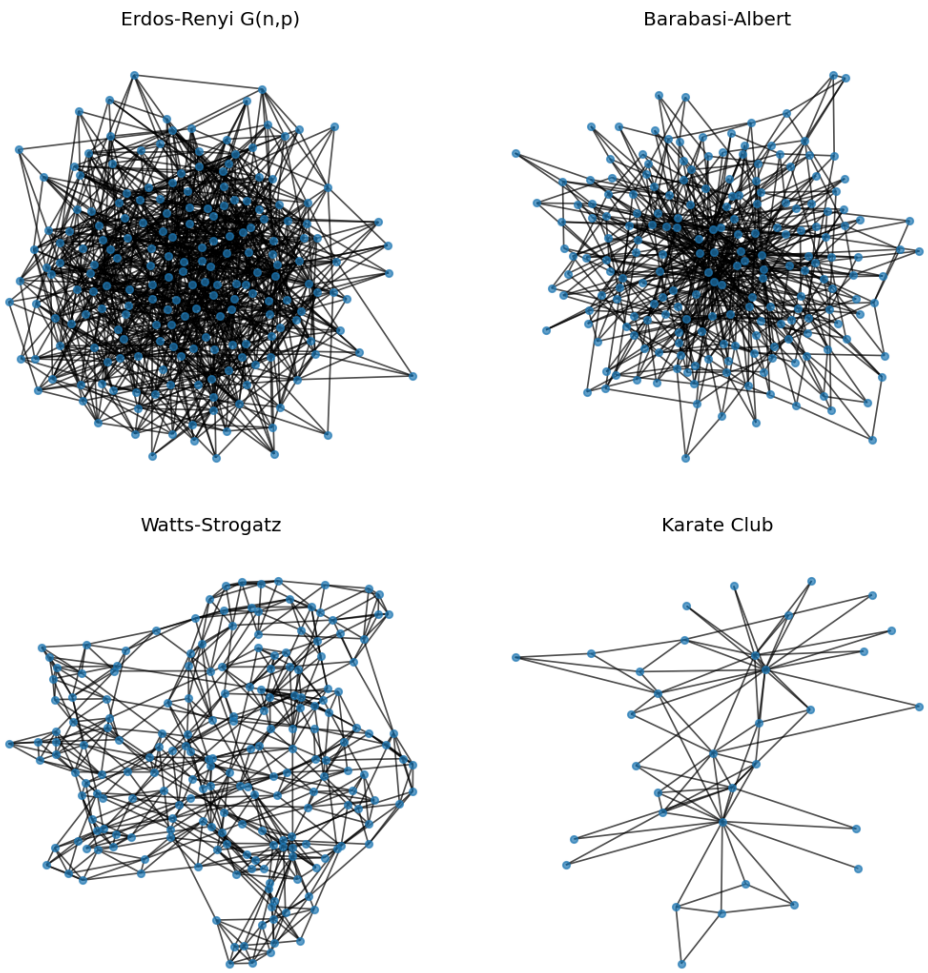

図 1　4 種類の代表的ネットワーク．ER, BA, WS は N=200 で生成しているが, Karate Club は 34 ノードからなる実ネットワークである．

```
参考文献:
    [1] Bailey, N. T. J. (1975). The Mathematical Theory of Infectious Diseases
        and Its Applications (2nd ed.). London: Charles Griffin & Co.
    [2] Pastor-Satorras, R., Castellano, C., Van Mieghem, P. & Vespignani, A.
        (2015). Epidemic processes in complex networks.
        Reviews of Modern Physics 87, 925-979.
    [3] ndlib SI Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/SIm.html
"""
from __future__ import annotations

import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    g = nx.erdos_renyi_graph(1000, 0.01, seed=0)
    model = ep.SIModel(g)
```

```
    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.01)
    cfg.add_model_parameter("fraction_infected", 0.01)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(200)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/03_si_model.png")
    print("Saved: figures/03_si_model.png")


if __name__ == "__main__":
    main()
```

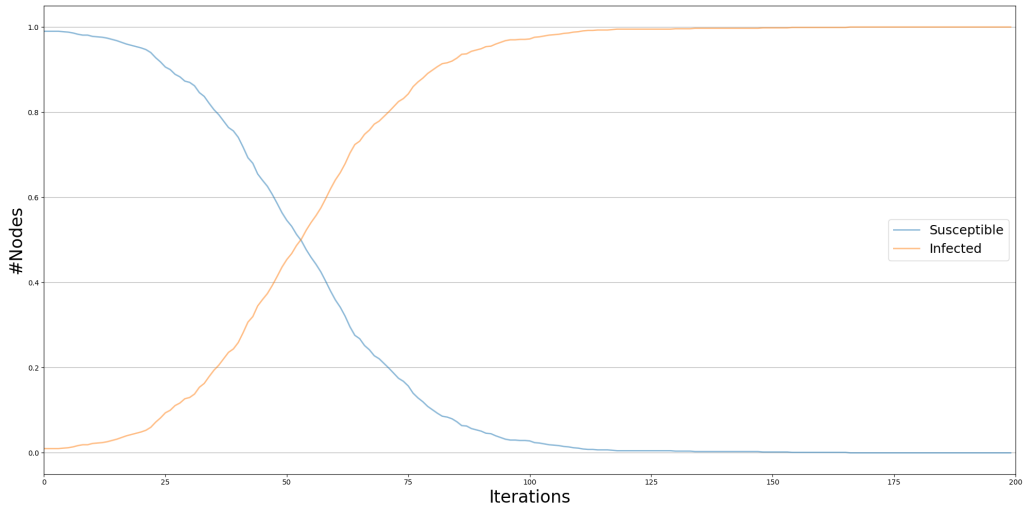

図 2　SI モデルの感染者割合の推移．回復状態がないため，感染者数は単調に増加する．

5.2　SIS モデル

感染後に再び感受性に戻る．パラメータ `lambda` は回復率 λ を表す．感染と回復が繰り返されるため，定常的な感染者割合を観察したい場合に向いている．

ソース: `scripts/04_sis_model.py`

```
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""04_sis_model.py — SIS(Susceptible-Infected-Susceptible) モデル

実行方法:
    uv run scripts/04_sis_model.py
```

```python

感染後に再び感受性に戻るタイプのモデルです．風邪などのように
免疫を獲得しない感染症の挙動を表現します．

参考文献:
    [1] Hethcote, H. W. (2000). The mathematics of infectious diseases.
        SIAM Review 42(4), 599-653.
    [2] Pastor-Satorras, R. & Vespignani, A. (2001). Epidemic spreading in
        scale-free networks. Phys. Rev. Lett. 86, 3200-3203.
    [3] ndlib SIS Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/SIS.html
"""
from __future__ import annotations

import sys
from pathlib import Path

import matplotlib.pyplot as plt
import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep

sys.path.insert(0, str(Path(__file__).parent))
from _plot_helpers import trend_to_ax, prevalence_to_ax  # noqa: E402


def main() -> None:
    g = nx.erdos_renyi_graph(1000, 0.01, seed=1)
    model = ep.SISModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.02)
    cfg.add_model_parameter("lambda", 0.01)
    cfg.add_model_parameter("fraction_infected", 0.05)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(200)
    trends = model.build_trends(iterations)

    fig, axes = plt.subplots(1, 2, figsize=(11, 4))
    trend_to_ax(model, trends, axes[0])
    axes[0].set_title("Diffusion Trend")
    prevalence_to_ax(model, trends, axes[1])
    axes[1].set_title("Diffusion Prevalence")
    fig.tight_layout()
    fig.savefig("figures/04_sis_model.png", dpi=120)
    print("Saved: figures/04_sis_model.png")


if __name__ == "__main__":
    main()
```

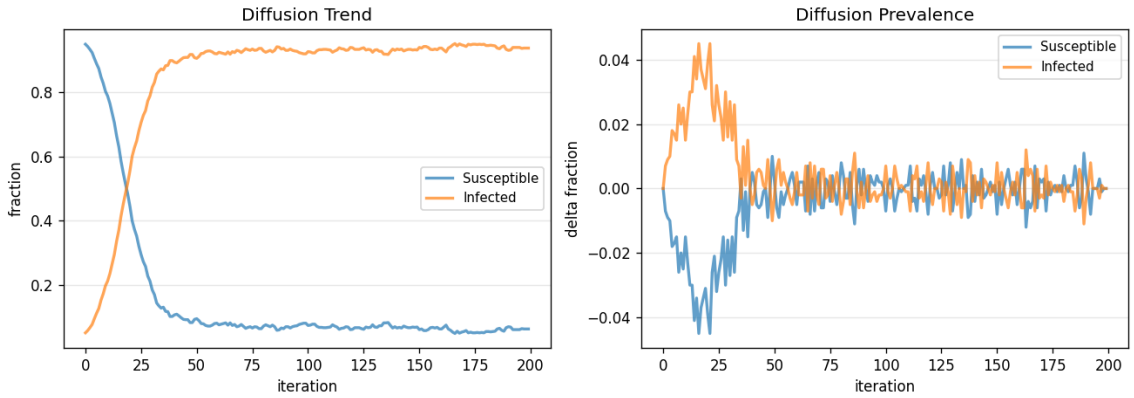

図 3　SIS モデルの状態割合とステップごとの増減．感染と回復が繰り返されるため，感染者割合は一定水準のまわりで推移する．

5.3　SIR モデル

回復 (あるいは死亡) した個体は二度と感染しない．インフルエンザのような短期流行のモデルとして最も基本的である．`beta` は感染率，`gamma` は `Infected` から `Removed` への遷移率である．

ソース: `scripts/05_sir_model.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""05_sir_model.py — SIR(Susceptible-Infected-Removed) モデル

実行方法:
    uv run scripts/05_sir_model.py

回復(または死亡)した個体が二度と感染しない古典的モデルです．
gamma を大きくすると流行は早く収束します．

参考文献:
    [1] Kermack, W. O. & McKendrick, A. G. (1927). A contribution to the
        mathematical theory of epidemics. Proc. Roy. Soc. Lond. A 115, 700-721.
    [2] Newman, M. E. J. (2002). Spread of epidemic disease on networks.
        Phys. Rev. E 66, 016128.
    [3] ndlib SIR Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/SIR.html
"""
from __future__ import annotations

import sys
from pathlib import Path

import matplotlib.pyplot as plt
```

```python
import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep

sys.path.insert(0, str(Path(__file__).parent))
from _plot_helpers import trend_to_ax  # noqa: E402


def run_once(beta: float, gamma: float, seed: int = 0):
    g = nx.erdos_renyi_graph(2000, 0.005, seed=seed)
    model = ep.SIRModel(g)
    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", beta)
    cfg.add_model_parameter("gamma", gamma)
    cfg.add_model_parameter("fraction_infected", 0.01)
    model.set_initial_status(cfg)
    iters = model.iteration_bunch(150)
    return model, model.build_trends(iters)


def main() -> None:
    fig, axes = plt.subplots(1, 3, figsize=(15, 4))
    for ax, gamma in zip(axes, [0.005, 0.02, 0.05]):
        model, trends = run_once(0.02, gamma)
        trend_to_ax(model, trends, ax)
        ax.set_title(f"gamma = {gamma}")
    fig.tight_layout()
    fig.savefig("figures/05_sir_model.png", dpi=120)
    print("Saved: figures/05_sir_model.png")


if __name__ == "__main__":
    main()
```

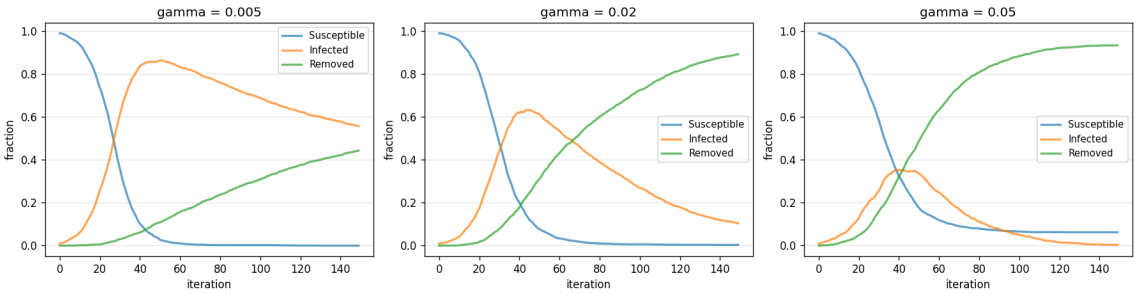

図 4　SIR モデルの γ 依存性．回復率を上げると流行は小さく早く収束する．

5.4　SEIR モデル

潜伏期間 E を持つ．パラメータ `alpha` が $E \to I$ への遷移率に対応する．`beta` は $S \to E$, `gamma` は $I \to R$ に対応する．

ソース: `scripts/06_seir_model.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""06_seir_model.py — SEIR モデル(潜伏期間あり)

実行方法:
    uv run scripts/06_seir_model.py

S (Susceptible) → E (Exposed/潜伏期間) → I (Infected) → R (Removed) と推移します.
COVID-19 のような潜伏期間を持つ感染症のモデル化に向いています.

参考文献:
    [1] Aron, J. L. & Schwartz, I. B. (1984). Seasonality and period-doubling
        bifurcations in an epidemic model. Journal of Theoretical Biology
        110, 665-679. (ndlib SEIR モデルの公式引用)
    [2] Anderson, R. M. & May, R. M. (1991). Infectious Diseases of Humans:
        Dynamics and Control. Oxford University Press.
    [3] ndlib SEIR Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/SEIR.html
"""
from __future__ import annotations

import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    g = nx.barabasi_albert_graph(1500, 3, seed=2)
    model = ep.SEIRModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.02)
    cfg.add_model_parameter("gamma", 0.01)
    cfg.add_model_parameter("alpha", 0.05)
    cfg.add_model_parameter("fraction_infected", 0.005)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(250)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/06_seir_model.png")
    print("Saved: figures/06_seir_model.png")


if __name__ == "__main__":
    main()
```

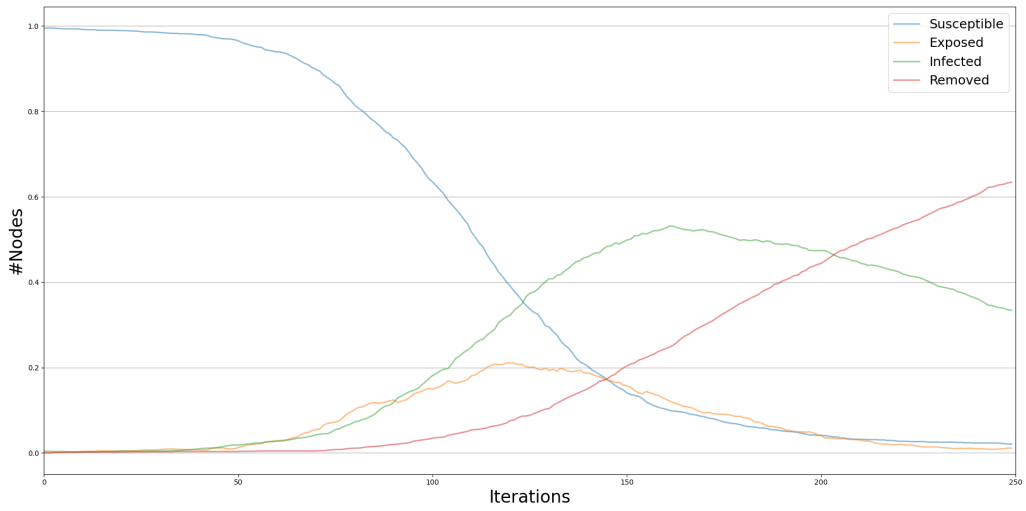

図 5　SEIR モデルの拡散過程．潜伏状態 E を経由してから感染状態 I に移るため，SIR よりも感染ピークが遅れて現れる．

5.5　SEIS モデル

SEIR モデルから免疫獲得を取り去ったモデル．$E \to I$ は `alpha`，$I \to S$ は `lambda` で制御する．慢性的な再感染リスクを伴う感染症の表現に適する．

ソース: `scripts/07_seis_model.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""07_seis_model.py — SEIS モデル（潜伏期間あり，免疫なし）

実行方法:
    uv run scripts/07_seis_model.py

潜伏期間を経て感染し，回復後は再び感受性に戻る点が SEIR との違いです．

参考文献:
    [1] Hethcote, H. W. (1976). Qualitative analyses of communicable disease
        models. Math. Biosciences 28, 335-356.
    [2] Cooke, K. L. & van den Driessche, P. (1996). Analysis of an SEIRS
        epidemic model with two delays. J. Math. Biology 35, 240-260.
    [3] ndlib SEIS Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/SEIS.html
"""
from __future__ import annotations

import networkx as nx
import ndlib.models.ModelConfig as mc
```

```python
import ndlib.models.epidemics as ep
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    g = nx.watts_strogatz_graph(1500, 6, 0.1, seed=3)
    model = ep.SEISModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.02)
    cfg.add_model_parameter("lambda", 0.01)
    cfg.add_model_parameter("alpha", 0.05)
    cfg.add_model_parameter("fraction_infected", 0.01)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(300)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/07_seis_model.png")
    print("Saved: figures/07_seis_model.png")


if __name__ == "__main__":
    main()
```

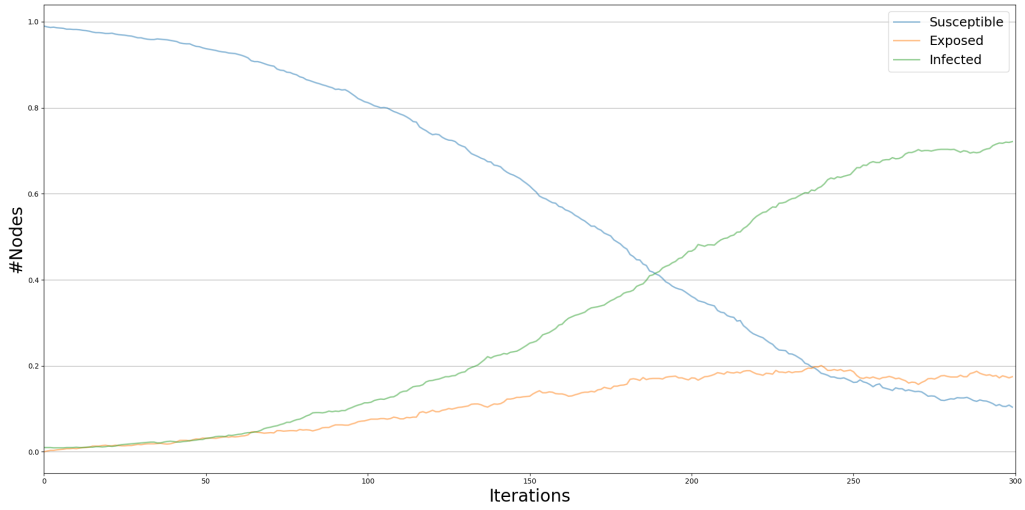

図 6　SEIS モデルの拡散過程．回復後に免疫を獲得せず感受性状態へ戻るため，感染がネットワーク内に残りやすい．

5.6　SWIR モデル

S (Susceptible), W (Weakened), I (Infected), R (Removed) の 4 状態を持つモデルである．感染者に接触した感受性者は，`kappa` により直接感染するか，`mu` により弱化状態へ移る．弱化状態のノードは `nu` により感染へ移る．したがって，「接触後すぐ感染する経路」と「いったん脆弱化してから感染する経路」を同時に表現できる．このサンプルでは弱化・直接感染・弱化後感染の確率をやや高めに置き，短い時間窓で状態遷移が見えるようにしている．

ソース: scripts/08_swir_model.py

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
#     "numpy",
# ]
# ///
"""08_swir_model.py — SWIR モデル(感染前に Weak 状態を経由)

実行方法:
    uv run scripts/08_swir_model.py

S(Susceptible) -> W(Weakened) -> I(Infected) -> R(Removed) と推移します.
免疫が弱まった個体だけが感染する設定をモデル化できます.

参考文献:
    [1] Lee, D., Choi, W., Kertész, J. & Kahng, B. (2017). Universal mechanism
        for hybrid percolation transitions. Scientific Reports 7, 5723.
        (ndlib SWIR モデルの公式引用)
    [2] ndlib SWIR Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/SWIR.html
"""
from __future__ import annotations

import random

import networkx as nx
import numpy as np
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    random.seed(408)
    np.random.seed(408)

    g = nx.erdos_renyi_graph(2000, 0.005, seed=4)
    model = ep.SWIRModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("kappa", 0.03)
    cfg.add_model_parameter("mu", 0.02)
    cfg.add_model_parameter("nu", 0.05)
    cfg.add_model_parameter("fraction_infected", 0.02)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(60)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/08_swir_model.png")
    print("Saved: figures/08_swir_model.png")

```

```python

if __name__ == "__main__":
    main()
```

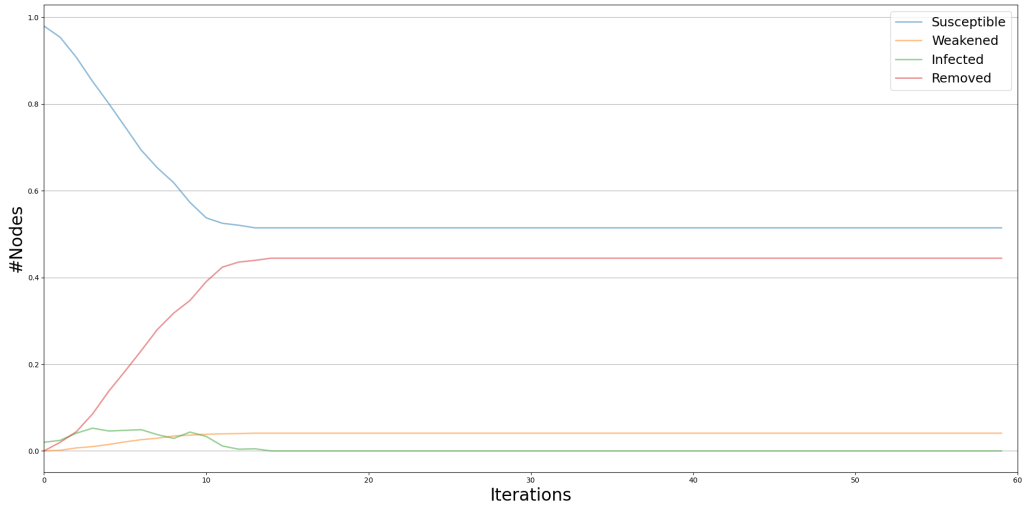

図 7　SWIR モデルの拡散過程．弱化状態 W を導入することで，接触後すぐ感染する経路と，脆弱化してから感染する経路を区別できる．

6　閾値・カスケード・プロファイルモデル

6.1　Granovetter の閾値モデル

近傍の採用割合がノード固有の閾値を超えると，そのノードも採用状態へ移る．ここでの `Infected` は病気ではなく，行動・意見・製品などを採用した状態を表す．社会運動や流行採用などのモデルとして使われる．サンプルでは全員に同じ閾値を置くのではなく，0.05–0.25 の範囲でノードごとに閾値を変え，カスケードが段階的に進む例を示す．

ソース: `scripts/09_threshold_model.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
#     "numpy",
# ]
# ///
"""09_threshold_model.py — 閾値モデル(Granovetter)

実行方法:
    uv run scripts/09_threshold_model.py

各ノードは閾値を持ち，近傍の感染割合が閾値を超えると感染します．
社会運動や行動変容の伝播モデルとして用いられます．

```

```
参考文献:
    [1] Granovetter, M. (1978). Threshold models of collective behavior.
        American Journal of Sociology 83(6), 1420–1443.
    [2] Watts, D. J. (2002). A simple model of global cascades on random
        networks. PNAS 99(9), 5766–5771.
    [3] ndlib Threshold Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/Threshold.html
"""
from __future__ import annotations

import random

import networkx as nx
import numpy as np
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    random.seed(509)
    np.random.seed(509)
    rng = random.Random(509)

    g = nx.erdos_renyi_graph(1000, 0.04, seed=5)
    model = ep.ThresholdModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("fraction_infected", 0.02)
    for n in g.nodes():
        cfg.add_node_configuration("threshold", n, rng.uniform(0.05, 0.25))
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(20)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/09_threshold_model.png")
    print("Saved: figures/09_threshold_model.png")


if __name__ == "__main__":
    main()
```

6.2 Independent Cascades モデル

アクティブになったノードは, 直後の 1 ステップだけ近傍ノードの活性化を試みる. 各エッジに伝搬確率 `threshold` を割り当てるのが特徴である. ノードごとの閾値ではなく, エッジごとの一回限りの伝搬確率である点に注意する. サンプルでは初期活性化率とエッジ伝搬確率を少し高め, アクティブ状態から静止状態へ流れていく過程が見えるようにしている.

ソース: `scripts/10_independent_cascades.py`

```
# /// script
# requires-python = ">=3.10"
```

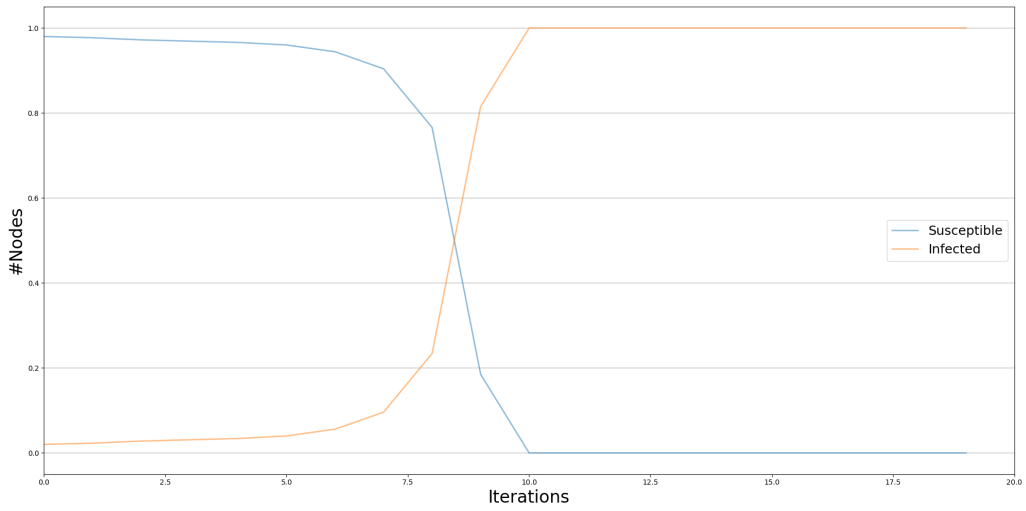

図 8　Granovetter 型閾値モデルの拡散過程．初期採用者から近傍割合の条件を満たすノードへ採用が広がる．

```
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
#     "numpy",
# ]
# ///
"""10_independent_cascades.py — 独立カスケードモデル

実行方法:
    uv run scripts/10_independent_cascades.py

感染ノードはその近傍に対し 1 度だけ感染を試みます．各エッジに割り当てた
確率 threshold で感染が成立し，試行後は静止状態となります．

参考文献:
    [1] Goldenberg, J., Libai, B. & Muller, E. (2001). Talk of the network:
        a complex systems look at the underlying process of word-of-mouth.
        Marketing Letters 12(3), 211-223.
    [2] Kempe, D., Kleinberg, J. & Tardos, É. (2003). Maximizing the spread
        of influence through a social network. Proc. KDD '03, 137-146.
    [3] ndlib IndependentCascades:
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/IndependentCascades.html
"""
from __future__ import annotations

import random

import networkx as nx
import numpy as np
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    random.seed(610)
    np.random.seed(610)
```

```python

    g = nx.barabasi_albert_graph(1000, 3, seed=6)
    model = ep.IndependentCascadesModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("fraction_infected", 0.03)
    for e in g.edges():
        cfg.add_edge_configuration("threshold", e, 0.2)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(30)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/10_independent_cascades.png")
    print("Saved: figures/10_independent_cascades.png")


if __name__ == "__main__":
    main()
```

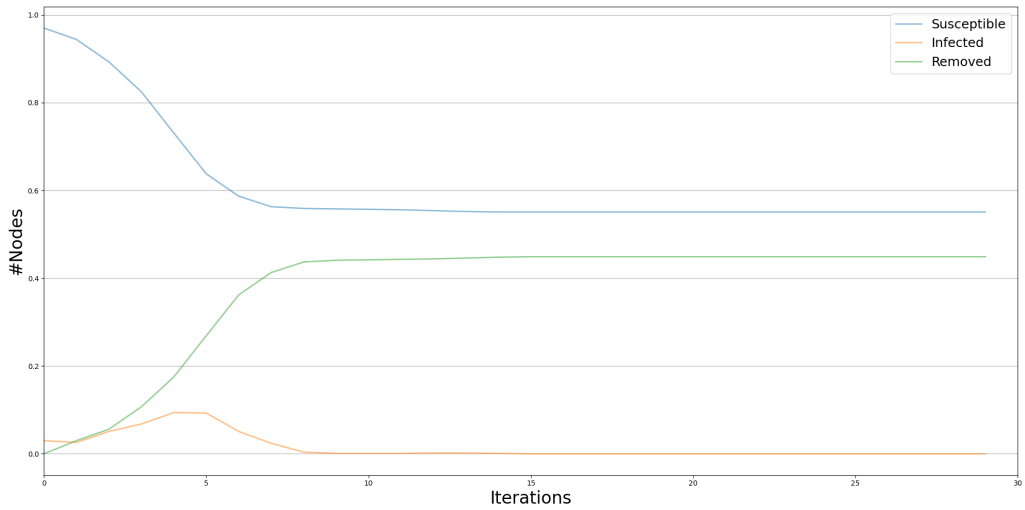

図 9　Independent Cascades モデルの拡散過程. 各アクティブノードは近傍への活性化試行を一度だけ行い, その後は静止状態へ移る.

6.3　プロファイルモデル

各ノードに 0–1 の感受性 (プロファイル値) を持たせ, 近傍に採用済みノードがいれば確率的に状態を切り替える. `blocked` を正にすると採用を拒否したノードがブロック状態へ移る可能性があり, `adopter_rate` は近傍と無関係な自発採用を表す.

ソース: `scripts/11_profile_model.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
```

```python
#     "matplotlib",
# ]
# ///
"""11_profile_model.py — プロファイルモデル

実行方法:
    uv run scripts/11_profile_model.py

各ノードに割り当てられたプロファイル値(個人の感受性)に応じて
感染が成立するモデルです.

参考文献:
    [1] Milli, L., Rossetti, G., Pedreschi, D. & Giannotti, F. (2018).
        Active and passive diffusion processes in complex networks.
        Applied Network Science 3, Article 42.
        DOI: 10.1007/s41109-018-0100-5  (ndlib Profile モデルの公式引用)
    [2] ndlib Profile Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/Profile.html
"""
from __future__ import annotations

import random

import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    rng = random.Random(7)
    g = nx.erdos_renyi_graph(1000, 0.05, seed=7)
    model = ep.ProfileModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("fraction_infected", 0.05)
    cfg.add_model_parameter("blocked", 0.0)
    cfg.add_model_parameter("adopter_rate", 0.0)
    for n in g.nodes():
        cfg.add_node_configuration("profile", n, rng.random())
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(80)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/11_profile_model.png")
    print("Saved: figures/11_profile_model.png")


if __name__ == "__main__":
    main()
```

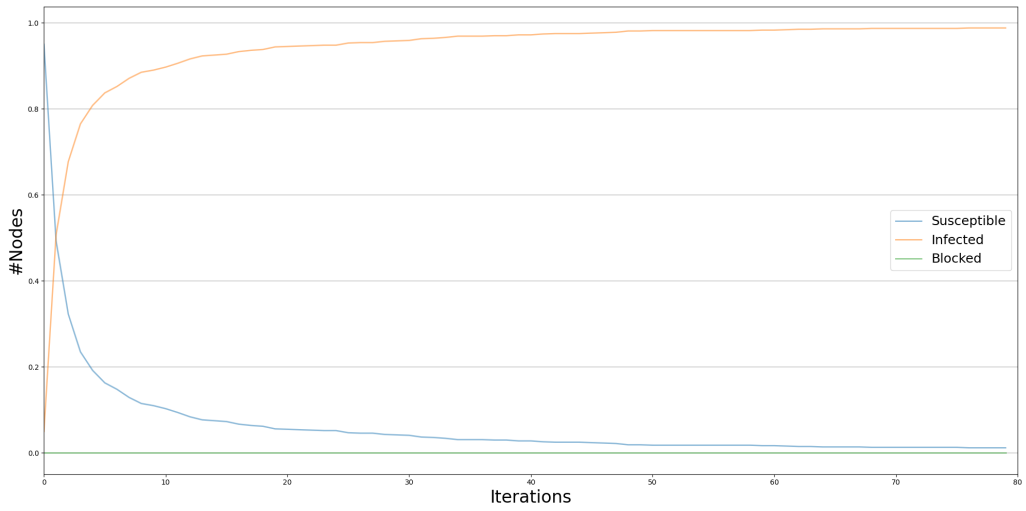

図 10　プロファイルモデルの拡散過程．ノードごとの感受性と自発採用率により，同じ近傍状態でも採用確率が変わる．

6.4　Kertesz 閾値モデル

頑固に状態を変えない **ブロックノード** を一定割合混入できる拡張閾値モデル．`adopter_rate` が自発採用率を，`percentage_blocked` が社会的影響を受けないノードの割合を表す．ブロック層が厚く，自発採用率が低い場合には，局所的な採用で止まる現象が観察できる．

ソース: `scripts/28_kertesz_threshold.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""28_kertesz_threshold.py — Kertesz 閾値モデル

実行方法:
    uv run scripts/28_kertesz_threshold.py

通常の閾値モデルにブロックノード(感染しない頑固者)を導入したものです．
社会運動が一定以上のブロック層では収束しない現象を再現できます．

参考文献:
    [1] Ruan, Z., Iñiguez, G., Karsai, M. & Kertész, J. (2015). Kinetics of
        social contagion. Physical Review Letters 115, 218702.
        (ndlib KerteszThreshold モデルの公式引用)
    [2] Watts, D. J. (2002). A simple model of global cascades on random
        networks. PNAS 99(9), 5766–5771.
    [3] Karsai, M., Iñiguez, G., Kaski, K. & Kertész, J. (2014). Complex
        contagion process in spreading of online innovation.
        Journal of the Royal Society Interface 11, 20140694.
    [4] ndlib KerteszThreshold Model:
```

```
        https://ndlib.readthedocs.io/en/latest/reference/models/epidemics/KThreshold.html
"""
from __future__ import annotations

import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    g = nx.erdos_renyi_graph(1000, 0.05, seed=22)
    model = ep.KerteszThresholdModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("adopter_rate", 0.4)
    cfg.add_model_parameter("percentage_blocked", 0.1)
    cfg.add_model_parameter("fraction_infected", 0.1)
    for n in g.nodes():
        cfg.add_node_configuration("threshold", n, 0.25)
    model.set_initial_status(cfg)

    iters = model.iteration_bunch(60)
    trends = model.build_trends(iters)
    DiffusionTrend(model, trends).plot(filename="figures/28_kertesz_threshold.png")
    print("Saved: figures/28_kertesz_threshold.png")


if __name__ == "__main__":
    main()
```

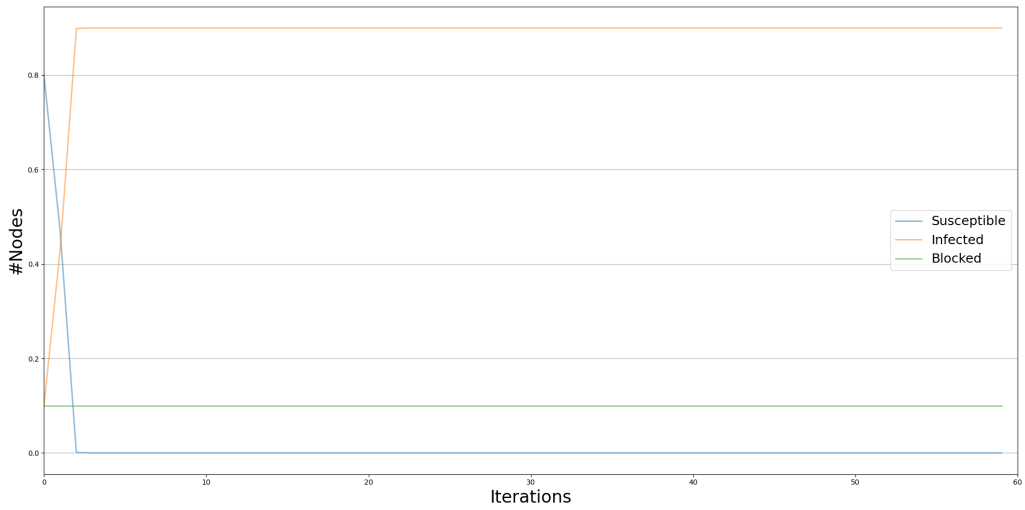

図 11　Kertesz 閾値モデルの拡散過程．ブロックノードを含むため，通常の閾値モデルよりもカスケードが途中で止まりやすい．

7　意見ダイナミクス: 離散値モデル

7.1　Voter モデル

古典的な投票者モデルである．各ステップでノードが近傍の意見を模倣する．このサンプルでは 0/1 の二値状態を半々に初期化する．コード上は `fraction_infected` を使うが，ここでは「意見 1 の初期割合」と読む．有限の連結グラフでは確率的にいずれかの意見へ吸収されるが，収束先と収束時間はネットワーク構造に依存する．

ソース: `scripts/12_voter_model.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "scikit-learn",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""12_voter_model.py — Voter モデル（投票者モデル）

実行方法:
    uv run scripts/12_voter_model.py

各ステップでランダムに選んだノードが，ランダムに選んだ近傍の意見を
模倣します．完備グラフでは合意に至ることが知られています．

参考文献:
    [1] Clifford, P. & Sudbury, A. (1973). A model for spatial conflict.
        Biometrika 60(3), 581-588.
    [2] Holley, R. A. & Liggett, T. M. (1975). Ergodic theorems for weakly
        interacting infinite systems and the voter model.
        Annals of Probability 3(4), 643-663.
    [3] Krapivsky, P. L., Redner, S. & Ben-Naim, E. (2010). A Kinetic View of
        Statistical Physics. Cambridge University Press.
        (ndlib Voter モデルの公式引用)
    [4] Castellano, C., Fortunato, S. & Loreto, V. (2009). Statistical physics
        of social dynamics. Reviews of Modern Physics 81, 591-646.
    [5] ndlib Voter Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/opinion/Voter.html
"""
from __future__ import annotations

import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.opinions as op
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    g = nx.erdos_renyi_graph(1000, 0.1, seed=8)
    model = op.VoterModel(g)

```

```
    cfg = mc.Configuration()
    cfg.add_model_parameter("fraction_infected", 0.5)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(300)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/12_voter_model.png")
    print("Saved: figures/12_voter_model.png")


if __name__ == "__main__":
    main()
```

図 12　Voter モデルの意見割合の推移．確率的な模倣を繰り返し，有限ネットワークではどちらかの意見へ吸収される．

7.2　Q-Voter モデル

q 人の近傍が全員一致のときだけ意見を採用する．q を変えると収束時間や収束先が大きく変わる．q が大きいほど，更新に必要な局所的同調条件が厳しくなる．

ソース: `scripts/13_qvoter_model.py`

```
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "scikit-learn",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""13_qvoter_model.py — Q-Voter モデル

実行方法:
    uv run scripts/13_qvoter_model.py
```

```
14
15 ノードはランダムに q 人の近傍を選び，全員の意見が一致している場合に限り
16 それを採用します．q を変化させると合意形成の挙動が大きく変わります．
17
18 参考文献:
19     [1] Castellano, C., Muñoz, M. A. & Pastor-Satorras, R. (2009).
20         Nonlinear q-voter model. Phys. Rev. E 80, 041129.
21     [2] Nyczka, P., Sznajd-Weron, K. & Cisło, J. (2012). Phase transitions
22         in the q-voter model with two types of stochastic driving.
23         Phys. Rev. E 86, 011105.
24     [3] ndlib QVoter Model:
25         https://ndlib.readthedocs.io/en/latest/reference/models/opinion/QVoter.html
26 """
27 from __future__ import annotations
28
29 import networkx as nx
30 import ndlib.models.ModelConfig as mc
31 import ndlib.models.opinions as op
32 from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend
33
34
35 def main() -> None:
36     g = nx.erdos_renyi_graph(1000, 0.1, seed=9)
37     model = op.QVoterModel(g)
38
39     cfg = mc.Configuration()
40     cfg.add_model_parameter("q", 5)
41     cfg.add_model_parameter("fraction_infected", 0.5)
42     model.set_initial_status(cfg)
43
44     iterations = model.iteration_bunch(300)
45     trends = model.build_trends(iterations)
46
47     DiffusionTrend(model, trends).plot(filename="figures/13_qvoter_model.png")
48     print("Saved: figures/13_qvoter_model.png")
49
50
51 if __name__ == "__main__":
52     main()
```

7.3 Majority Rule モデル

q 人をランダムに選び, 多数派の意見にそろえる集団議論モデル．サンプルでは $q = 3$ とし, 同数で割れる状況を避けている．

ソース: `scripts/14_majority_rule.py`

```
1 # /// script
2 # requires-python = ">=3.10"
3 # dependencies = [
4 #     "ndlib",
5 #     "scikit-learn",
6 #     "networkx",
7 #     "matplotlib",
8 # ]
```

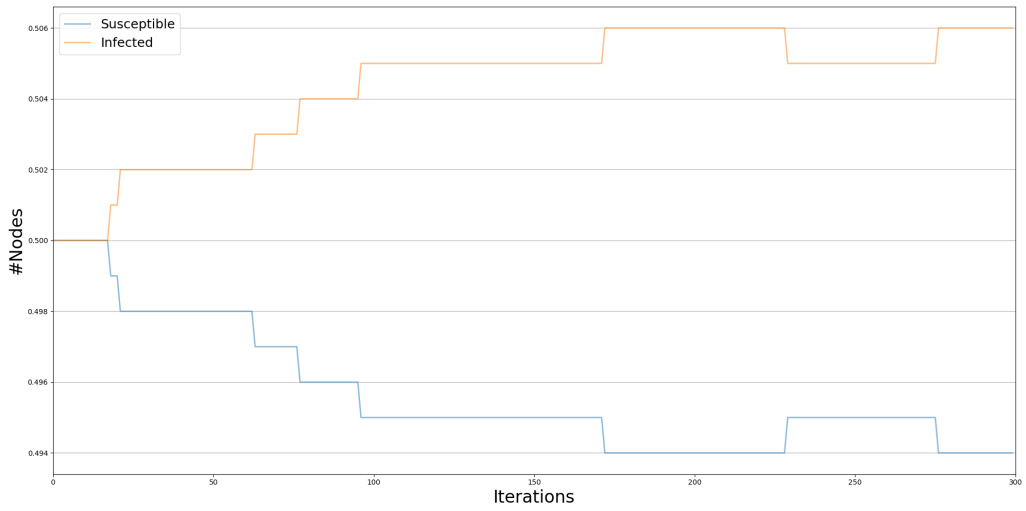

図 13　Q-Voter モデルの意見割合の推移．q 人の近傍が一致した場合だけ更新されるため，局所同調の条件が合意形成を左右する．

```python
# ///
"""14_majority_rule.py — Majority Rule(多数決)モデル

実行方法:
    uv run scripts/14_majority_rule.py

各ステップでランダムに選んだ q 人のグループ内で多数派の意見に
収斂させます．集団内議論を抽象化したモデルです．

参考文献:
    [1] Galam, S. (2002). Minority opinion spreading in random geometry.
        European Physical Journal B 25(4), 403-406.
        (ndlib MajorityRule モデルの公式引用)
    [2] Friedman, R. & Friedman, M. (1984). The Tyranny of the Status Quo.
        Harcourt Brace, Orlando, FL.
    [3] Krapivsky, P. L. & Redner, S. (2003). Dynamics of majority rule in
        two-state interacting spin systems. Phys. Rev. Lett. 90, 238701.
    [4] ndlib MajorityRule Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/opinion/MajorityRule.html
"""
from __future__ import annotations

import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.opinions as op
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    g = nx.erdos_renyi_graph(1000, 0.1, seed=10)
    model = op.MajorityRuleModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("q", 3)
    cfg.add_model_parameter("fraction_infected", 0.5)
    model.set_initial_status(cfg)
```

```
45
46      iterations = model.iteration_bunch(300)
47      trends = model.build_trends(iterations)
48
49      DiffusionTrend(model, trends).plot(filename="figures/14_majority_rule.png")
50      print("Saved: figures/14_majority_rule.png")
51
52
53  if __name__ == "__main__":
54      main()
```

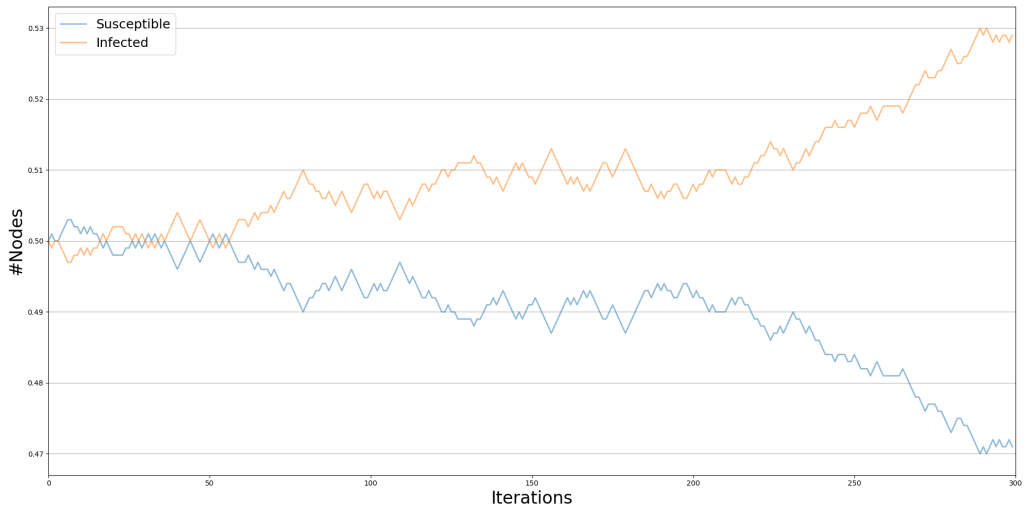

図 14　Majority Rule モデルの意見割合の推移．ランダムに選ばれた小集団内の多数派へ意見がそろう．

7.4　Sznajd モデル

2 人組が一致しているとき, その意見を周囲に押し付ける．社会的影響と同調圧力をシンプルに表現する．Voter モデルが「個人が周囲をまねる」方向の更新であるのに対し, Sznajd モデルは「一致したペアが周囲へ影響する」方向の更新である．

ソース: `scripts/15_sznajd_model.py`

```
1   # /// script
2   # requires-python = ">=3.10"
3   # dependencies = [
4   #     "ndlib",
5   #     "scikit-learn",
6   #     "networkx",
7   #     "matplotlib",
8   # ]
9   # ///
10  """15_sznajd_model.py — Sznajd モデル
11
12  実行方法:
13      uv run scripts/15_sznajd_model.py
14
15  「2 人で意見が一致していれば，周囲を説得できる」という社会的影響の
```

```
ルールを採用したモデルです．

参考文献:
    [1] Sznajd-Weron, K. & Sznajd, J. (2000). Opinion evolution in closed
        community. International Journal of Modern Physics C 11(6), 1157–1165.
        (ndlib Sznajd モデルの公式引用)
    [2] Stauffer, D. (2002). Sociophysics: the Sznajd model and its
        applications. Computer Physics Communications 146(1), 93–98.
    [3] ndlib Sznajd Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/opinion/Snajzd.html
"""
from __future__ import annotations

import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.opinions as op
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    g = nx.erdos_renyi_graph(1000, 0.1, seed=11)
    model = op.SznajdModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("fraction_infected", 0.5)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(500)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/15_sznajd_model.png")
    print("Saved: figures/15_sznajd_model.png")


if __name__ == "__main__":
    main()
```

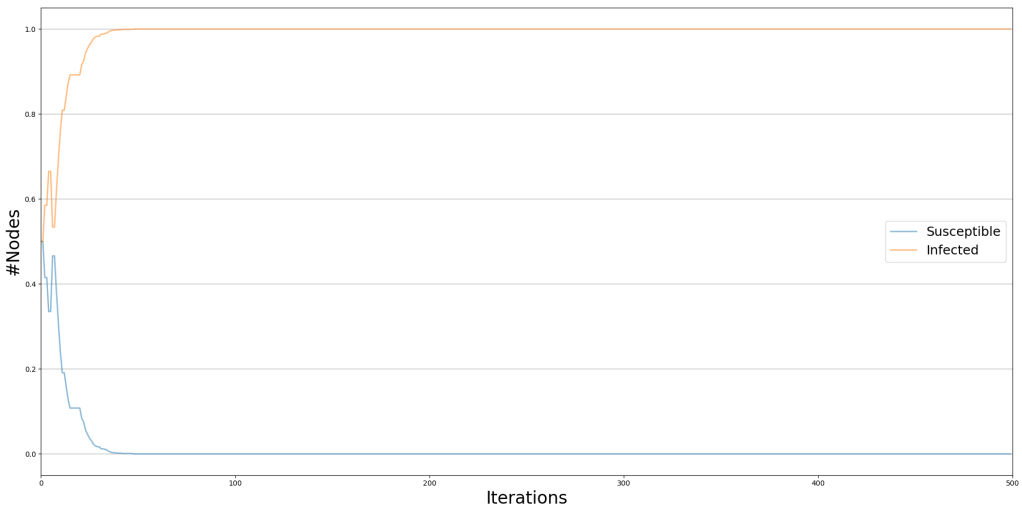

図 15　Sznajd モデルの意見割合の推移．一致したペアが周囲へ影響するため，局所的な同調ペアが拡散の起点になる．

8　意見ダイナミクス: 連続値・認知モデル

8.1　Hegselmann–Krause モデル

連続値の意見を持ち, 自分との意見差が epsilon 以下の近傍と平均を取る. epsilon が小さいと意見クラスタが複数残る (有界信頼). 出力は感染者数ではなく各ノードの連続値意見なので, 本節では DiffusionTrend ではなく軌跡を直接描いている.

ソース: scripts/16_hegselmann_krause.py

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "scikit-learn",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""16_hegselmann_krause.py — Hegselmann-Krause(有界信頼)モデル

実行方法:
    uv run scripts/16_hegselmann_krause.py

連続値の意見を持つエージェントが, 自分との意見差が epsilon 以下の近傍
だけを参照して意見を更新します.

参考文献:
    [1] Hegselmann, R. & Krause, U. (2002). Opinion dynamics and bounded
        confidence: models, analysis and simulation. Journal of Artificial
        Societies and Social Simulation 5(3).
    [2] Lorenz, J. (2007). Continuous opinion dynamics under bounded
        confidence: a survey. International Journal of Modern Physics C
        18(12), 1819-1838.
    [3] ndlib HK Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/opinion/HK.html
"""
from __future__ import annotations

import matplotlib.pyplot as plt
import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.opinions as op


def main() -> None:
    g = nx.erdos_renyi_graph(200, 0.1, seed=12)
    model = op.HKModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("epsilon", 0.32)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(40, node_status=True)
```

```
45
46      # 各ノードの意見軌跡を抽出
47      opinions = {n: [iterations[0]["status"][n]] for n in g.nodes()}
48      for it in iterations[1:]:
49          for n, val in it["status"].items():
50              opinions[n].append(val)
51          # 状態が変化していないノードは前ステップ値を引き継ぐ
52          for n in g.nodes():
53              if len(opinions[n]) < it["iteration"] + 1:
54                  opinions[n].append(opinions[n][-1])
55
56      fig, ax = plt.subplots(figsize=(8, 5))
57      for n, vals in opinions.items():
58          ax.plot(range(len(vals)), vals, color="tab:blue", alpha=0.3, linewidth=0.7)
59      ax.set_xlabel("iteration")
60      ax.set_ylabel("opinion")
61      ax.set_title("Hegselmann-Krause model: opinion trajectories")
62      fig.tight_layout()
63      fig.savefig("figures/16_hegselmann_krause.png", dpi=120)
64      plt.close(fig)
65      print("Saved: figures/16_hegselmann_krause.png")
66
67
68  if __name__ == "__main__":
69      main()
```

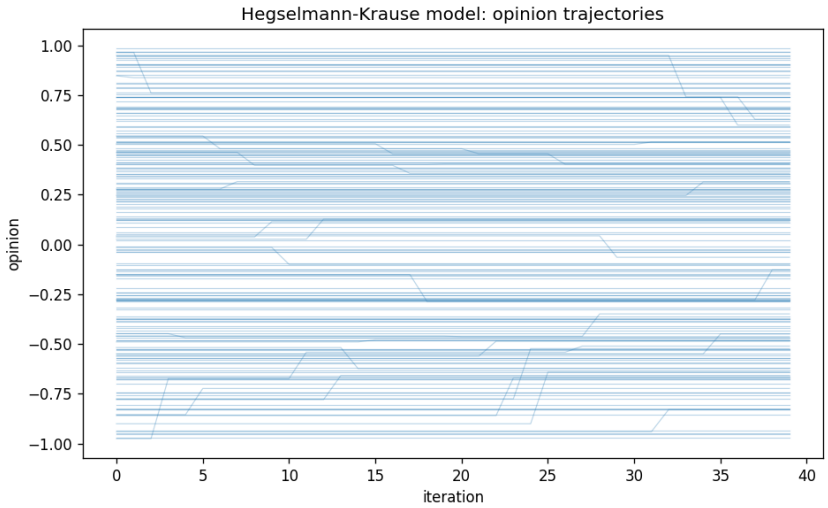

図 16　Hegselmann–Krause モデルにおける意見軌跡．同程度の意見を持つグループへ収束していく．

8.2　Algorithmic Bias モデル

推薦アルゴリズム的な「似た意見ほど見せやすい」バイアス gamma を導入したモデル．gamma が大きくなるほど近い意見同士が相互作用しやすくなり，エコーチェンバーや分極化が形成されやすくなる．epsilon は相互作用できる意見差の幅である．

ソース: scripts/17_algorithmic_bias.py

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "scikit-learn",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""17_algorithmic_bias.py — Algorithmic Bias モデル

実行方法:
    uv run scripts/17_algorithmic_bias.py

epsilon は意見差の許容幅，gamma はバイアスの強さを表します．
gamma を大きくすると意見の似た者同士でクラスターが形成されやすくなり，
SNS 上のエコーチェンバー現象を簡易的に再現できます．

参考文献:
    [1] Sîrbu, A., Pedreschi, D., Giannotti, F. & Kertész, J. (2019).
        Algorithmic bias amplifies opinion fragmentation and polarization:
        a bounded confidence model. PLoS ONE 14(3), e0213246.
    [2] Pariser, E. (2011). The Filter Bubble: What the Internet Is Hiding
        from You. Penguin Press.
    [3] ndlib AlgorithmicBias Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/opinion/AlgorithmicBias.html
"""
from __future__ import annotations

import matplotlib.pyplot as plt
import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.opinions as op


def simulate(gamma: float, ax: plt.Axes) -> None:
    g = nx.complete_graph(100)
    model = op.AlgorithmicBiasModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("epsilon", 0.32)
    cfg.add_model_parameter("gamma", gamma)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(100, node_status=True)
    n_nodes = g.number_of_nodes()
    history = [[iterations[0]["status"].get(i, 0.0)] for i in range(n_nodes)]
    current = [iterations[0]["status"].get(i, 0.0) for i in range(n_nodes)]
    for it in iterations[1:]:
        for k, v in it["status"].items():
            current[k] = v
        for i in range(n_nodes):
            history[i].append(current[i])

    for traj in history:
        ax.plot(range(len(traj)), traj, color="tab:red", alpha=0.3, linewidth=0.6)
    ax.set_title(f"gamma = {gamma}")
```

```
    ax.set_xlabel("iteration")
    ax.set_ylabel("opinion")


def main() -> None:
    fig, axes = plt.subplots(1, 3, figsize=(13, 4))
    for ax, gamma in zip(axes, [0.0, 0.5, 1.5]):
        simulate(gamma, ax)
    fig.tight_layout()
    fig.savefig("figures/17_algorithmic_bias.png", dpi=120)
    plt.close(fig)
    print("Saved: figures/17_algorithmic_bias.png")


if __name__ == "__main__":
    main()
```

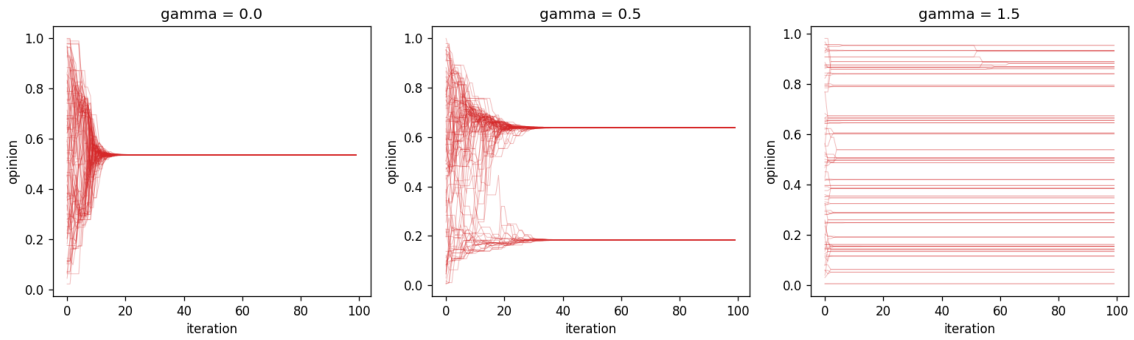

図 17　Algorithmic Bias モデルにおける意見軌跡 (左から $\gamma = 0, 0.5, 1.5$). バイアスが強いほど意見クラスタが先鋭化する.

8.3　Cognitive Opinion Dynamics モデル

外部情報の強さ `I`, バイアス範囲 `B_range_*`, 関与度範囲 `T_range_*`, リスク認知の初期割合 `R_fraction_*` など複数の心理パラメータが相互作用する詳細モデル. サンプルでは, 各ノードの連続値意見が時間とともにどう変化するかを軌跡として可視化している.

ソース: `scripts/29_cognitive_opinion.py`

```
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "scikit-learn",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""29_cognitive_opinion.py — 認知的意見ダイナミクスモデル

実行方法:
    uv run scripts/29_cognitive_opinion.py
```

```
14
15  リスク認知，関与度などのパラメータを通じて，個人の認知バイアスが
16  意見形成に与える影響を観察できます．
17
18  参考文献：
19      [1] Vilone, D., Giardini, F., Paolucci, M. & Conte, R. (2016).
20          Reducing individuals' risk sensitiveness can promote positive and
21          non-alarmist views about catastrophic events in an agent-based
22          simulation. arXiv:1609.04566.
23      [2] Giardini, F. & Vilone, D. (2021). Opinion dynamics and collective
24          risk perception: an agent-based model of institutional and media
25          communication about disasters. Journal of Artificial Societies and
26          Social Simulation 24(1), 4. DOI: 10.18564/jasss.4479
27      [3] Sîrbu, A., Loreto, V., Servedio, V. D. P. & Tria, F. (2017).
28          Opinion dynamics: models, extensions and external effects.
29          In Participatory Sensing, Opinions and Collective Awareness, 363-401.
30      [4] ndlib Cognitive Opinion Dynamics:
31          https://ndlib.readthedocs.io/en/latest/reference/models/opinion/COD.html
32  """
33  from __future__ import annotations
34
35  import matplotlib.pyplot as plt
36  import networkx as nx
37  import ndlib.models.ModelConfig as mc
38  import ndlib.models.opinions as op
39
40
41  def main() -> None:
42      g = nx.erdos_renyi_graph(100, 0.1, seed=23)
43      model = op.CognitiveOpDynModel(g)
44
45      cfg = mc.Configuration()
46      cfg.add_model_parameter("I", 0.15)        # 外部情報の強さ
47      cfg.add_model_parameter("B_range_min", 0.0)
48      cfg.add_model_parameter("B_range_max", 1.0)
49      cfg.add_model_parameter("T_range_min", 0.0)
50      cfg.add_model_parameter("T_range_max", 1.0)
51      cfg.add_model_parameter("R_fraction_negative", 1/3)
52      cfg.add_model_parameter("R_fraction_neutral", 1/3)
53      cfg.add_model_parameter("R_fraction_positive", 1/3)
54      model.set_initial_status(cfg)
55
56      iters = model.iteration_bunch(20, node_status=True)
57      nodes = list(g.nodes())
58      history = {n: [iters[0]["status"].get(n, 0.0)] for n in nodes}
59      cur = dict(iters[0]["status"])
60      for it in iters[1:]:
61          cur.update(it["status"])
62          for n in nodes:
63              history[n].append(cur.get(n, history[n][-1]))
64
65      fig, ax = plt.subplots(figsize=(7, 4.5))
66      for traj in history.values():
67          ax.plot(range(len(traj)), traj, color="tab:purple", alpha=0.3)
68      ax.set_xlabel("iteration")
69      ax.set_ylabel("opinion")
70      ax.set_title("Cognitive opinion dynamics")
71      fig.tight_layout()
```

```
    fig.savefig("figures/29_cognitive_opinion.png", dpi=120)
    plt.close(fig)
    print("Saved: figures/29_cognitive_opinion.png")


if __name__ == "__main__":
    main()
```

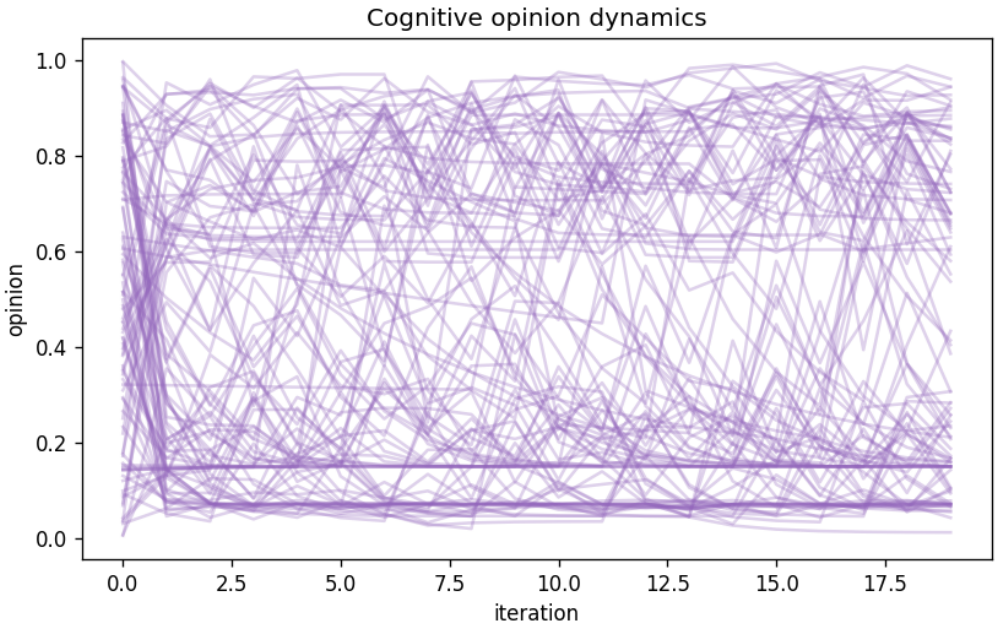

図 18　Cognitive Opinion Dynamics モデルにおける意見軌跡. 外部情報と心理パラメータの相互作用により, ノードごとの連続値意見が変化する.

9　動的ネットワーク上のモデル

エッジが時刻ごとに変わる時系列ネットワーク (`dynetx`) 上での拡散シミュレーションも可能である. 静的グラフ用の `ndlib.models.epidemics` ではなく `ndlib.models.dynamic` のモデルを使い, `iteration_bunch` ではなく `execute_snapshots` でスナップショット列を実行する. 静的モデルでは「誰と誰がつながっているか」が固定されるが, 動的モデルでは接触機会そのものが時間とともに変わる.

ソース: `scripts/20_dynamic_si.py`

```
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "dynetx",
#     "networkx",
#     "matplotlib",
# ]
# ///
```

```python
"""20_dynamic_si.py — 動的ネットワーク上の SI モデル

実行方法:
    uv run scripts/20_dynamic_si.py

dynetx で時刻ごとにスナップショットを持つ動的グラフを構築し，
ndlib の DynSIModel で拡散を観察します．

参考文献:
    [1] Holme, P. & Saramäki, J. (2012). Temporal networks.
        Physics Reports 519(3), 97-125.
    [2] Masuda, N. & Holme, P. (2013). Predicting and controlling infectious
        disease epidemics using temporal networks. F1000Prime Reports 5:6.
    [3] Rossetti, G. (2017). DyNetx: dynamic network library.
        https://dynetx.readthedocs.io/
    [4] ndlib Dynamic SI Model:
        https://ndlib.readthedocs.io/en/latest/reference/models/dynamics/dSI.html
"""
from __future__ import annotations

import random

import networkx as nx
import dynetx as dn
import ndlib.models.ModelConfig as mc
import ndlib.models.dynamic as dm
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    rng = random.Random(15)
    dg = dn.DynGraph()
    for t in range(20):
        snap = nx.erdos_renyi_graph(300, 0.02, seed=rng.randint(0, 10**6))
        dg.add_interactions_from(list(snap.edges()), t=t)

    model = dm.DynSIModel(dg)
    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.05)
    cfg.add_model_parameter("fraction_infected", 0.05)
    model.set_initial_status(cfg)

    iterations = model.execute_snapshots()
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/20_dynamic_si.png")
    print("Saved: figures/20_dynamic_si.png")


if __name__ == "__main__":
    main()
```

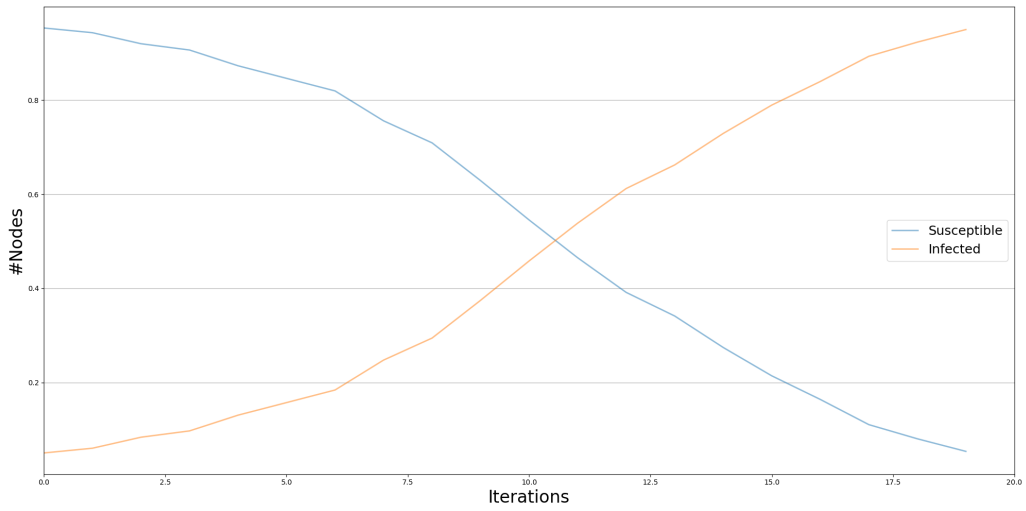

図 19　動的ネットワーク上の SI モデル．エッジ集合が時刻ごとに変わるため，接触機会の変化が感染拡大の速度に影響する．

10　初期条件・ノード単位パラメータ

10.1　ノードごとに異なる閾値

ベータ分布から閾値をサンプリングし，ノード単位で割り当てる例である．同じネットワークでも，低閾値ノードが多いほどカスケードは広がりやすい．第 6 節の閾値モデルを，ノードパラメータ設定の観点からもう一度扱う例と考えればよい．

ソース: `scripts/18_node_specific_params.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""18_node_specific_params.py — ノードごとに異なる閾値を持たせる

実行方法:
    uv run scripts/18_node_specific_params.py

ndlib では add_node_configuration / add_edge_configuration を使って
ノード単位・エッジ単位のパラメータを与えられます．

参考文献:
    [1] Granovetter, M. (1978). Threshold models of collective behavior.
        American Journal of Sociology 83(6), 1420–1443.
    [2] Watts, D. J. (2002). A simple model of global cascades on random
        networks. PNAS 99(9), 5766–5771.
    [3] ndlib ModelConfig (node/edge configuration):
        https://ndlib.readthedocs.io/en/latest/custom/custom.html
```

```python
"""
from __future__ import annotations

import random

import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    rng = random.Random(13)
    g = nx.barabasi_albert_graph(800, 3, seed=13)
    model = ep.ThresholdModel(g)

    cfg = mc.Configuration()
    cfg.add_model_parameter("fraction_infected", 0.05)
    for n in g.nodes():
        cfg.add_node_configuration("threshold", n, rng.betavariate(2, 5))
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(50)
    trends = model.build_trends(iterations)

    DiffusionTrend(model, trends).plot(filename="figures/18_node_specific_params.png")
    print("Saved: figures/18_node_specific_params.png")


if __name__ == "__main__":
    main()
```

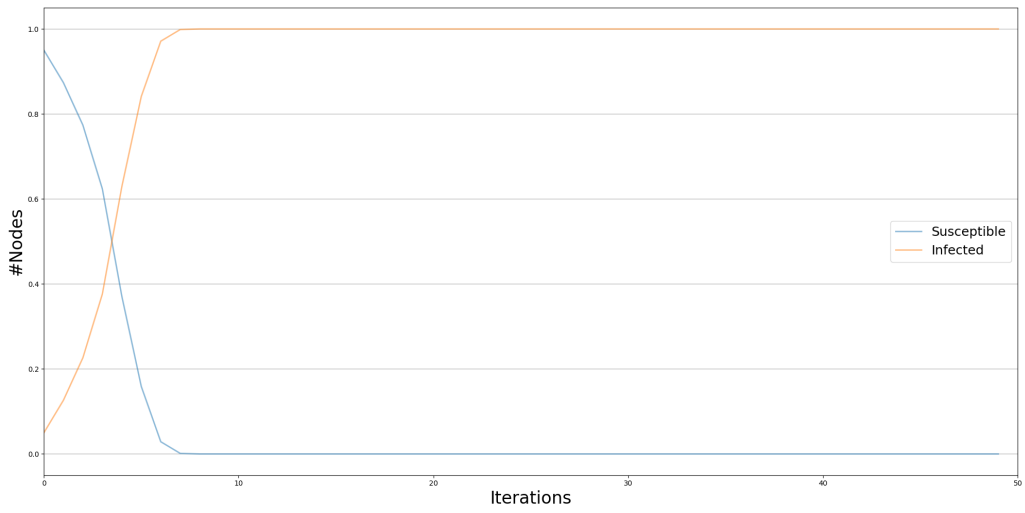

図 20　ノードごとに異なる閾値を持つ閾値モデル．低閾値ノードが多いほど，局所的な採用が全体のカスケードへつながりやすい．

10.2　初期感染ノードを明示的に与える

`add_model_initial_configuration("Infected", […])` を使うと，ハブやブリッジに感染を起こす介入実験ができる．サンプルでは高次数ノード 5 個と葉に近いノード 5 個を起点にして，同じ感染率でも到達規模が変わることを比較している．

ソース: `scripts/19_initial_infected_nodes.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""19_initial_infected_nodes.py — 初期感染ノードの明示的な指定

実行方法:
    uv run scripts/19_initial_infected_nodes.py

fraction_infected は初期感染者の比率をランダムに設定しますが，
代わりに 'Infected' に集合を渡すと特定ノードを起点にできます．
ここではハブ(高次数)ノードを起点とした拡散を観察します．

参考文献:
    [1] Pastor-Satorras, R. & Vespignani, A. (2001). Epidemic spreading in
        scale-free networks. Phys. Rev. Lett. 86, 3200-3203.
    [2] Albert, R., Jeong, H. & Barabási, A.-L. (2000). Error and attack
        tolerance of complex networks. Nature 406, 378-382.
    [3] Kitsak, M. et al. (2010). Identification of influential spreaders
        in complex networks. Nature Physics 6, 888-893.
"""
from __future__ import annotations

import sys
from pathlib import Path

import matplotlib.pyplot as plt
import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep

sys.path.insert(0, str(Path(__file__).parent))
from _plot_helpers import trend_to_ax  # noqa: E402


def run(seed_nodes: list[int], title: str, ax) -> None:
    g = nx.barabasi_albert_graph(800, 3, seed=14)
    model = ep.SIRModel(g)
    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.02)
    cfg.add_model_parameter("gamma", 0.01)
    cfg.add_model_initial_configuration("Infected", seed_nodes)
    model.set_initial_status(cfg)
```

```
    trends = model.build_trends(model.iteration_bunch(100))
    trend_to_ax(model, trends, ax)
    ax.set_title(title)


def main() -> None:
    g = nx.barabasi_albert_graph(800, 3, seed=14)
    degree_sorted = sorted(g.degree, key=lambda kv: kv[1], reverse=True)
    hubs = [n for n, _ in degree_sorted[:5]]
    leaves = [n for n, d in g.degree() if d == 1][:5]
    if not leaves:
        leaves = [n for n, _ in degree_sorted[-5:]]

    fig, axes = plt.subplots(1, 2, figsize=(11, 4))
    run(hubs, "seeds: top-5 hubs", axes[0])
    run(leaves, "seeds: 5 leaves", axes[1])
    fig.tight_layout()
    fig.savefig("figures/19_initial_infected_nodes.png", dpi=120)
    print("Saved: figures/19_initial_infected_nodes.png")


if __name__ == "__main__":
    main()
```

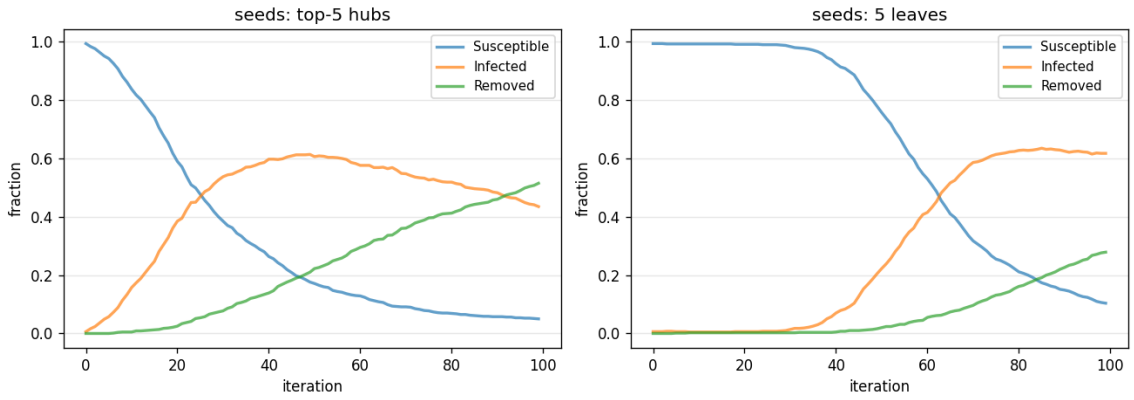

図 21　初期感染ノードの違いによる SIR 拡散の比較．ハブを起点にした場合と低次数ノードを起点にした場合で到達規模が異なる．

10.3　実ネットワーク (Karate Club) 上の SIR

Zachary の Karate Club ネットワークで，ノード 0 (教官) を起点とした感染拡散を観察する．人工的なランダムグラフだけでなく，小規模な実ネットワークに同じモデルを載せ替えられることを示す．

ソース: `scripts/23_real_network_karate.py`

```
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
```

```
#     "networkx",
#     "matplotlib",
# ]
# ///
"""23_real_network_karate.py — Karate Club ネットワーク上の SIR

実行方法:
    uv run scripts/23_real_network_karate.py

NetworkX 同梱の Karate Club ネットワーク(34 ノード)で SIR を実行し,
最終ステップの感染状態をネットワーク図に重ねて描画します.

参考文献:
    [1] Zachary, W. W. (1977). An information flow model for conflict and
        fission in small groups. Journal of Anthropological Research 33(4),
        452-473.
    [2] Girvan, M. & Newman, M. E. J. (2002). Community structure in social
        and biological networks. PNAS 99(12), 7821-7826.
"""
from __future__ import annotations

import matplotlib.pyplot as plt
import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep


COLORS = {0: "#377eb8", 1: "#e41a1c", 2: "#4daf4a"}  # S, I, R


def main() -> None:
    g = nx.karate_club_graph()
    model = ep.SIRModel(g)
    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.1)
    cfg.add_model_parameter("gamma", 0.05)
    cfg.add_model_initial_configuration("Infected", [0])  # 教官から開始
    model.set_initial_status(cfg)

    iters = model.iteration_bunch(40, node_status=True)

    # 最終時点の各ノード状態を再構築
    state = dict(iters[0]["status"])
    for it in iters[1:]:
        state.update(it["status"])

    pos = nx.spring_layout(g, seed=0)
    fig, ax = plt.subplots(figsize=(7, 6))
    node_colors = [COLORS.get(state[n], "#999999") for n in g.nodes()]
    nx.draw_networkx_edges(g, pos, ax=ax, alpha=0.4)
    nx.draw_networkx_nodes(g, pos, ax=ax, node_color=node_colors, node_size=320)
    nx.draw_networkx_labels(g, pos, ax=ax, font_size=8, font_color="white")
    ax.set_title("Karate Club: SIR end-state (S blue / I red / R green)")
    ax.axis("off")
    fig.tight_layout()
    fig.savefig("figures/23_real_network_karate.png", dpi=120)
    print("Saved: figures/23_real_network_karate.png")

```

```python

if __name__ == "__main__":
    main()
```

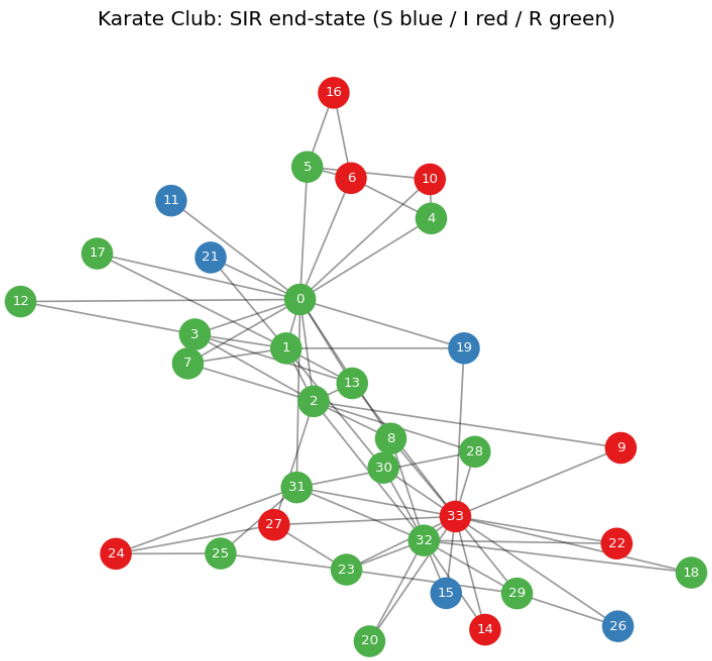

図 22　Karate Club ネットワーク上の SIR 最終状態．青=未感染, 赤=感染中, 緑=回復済．

11　シミュレーションの解析

11.1　モンテカルロによる平均挙動

拡散プロセスは確率的なので, 単一実行ではなく複数回シミュレートし, 平均と分位点を見るのが基本である．本サンプルでは同じグラフ上で乱数 seed を変えながら SIR を繰り返し実行し, 感染者数の平均と 10–90 パーセンタイルを描く．

ソース: `scripts/24_monte_carlo.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
#     "numpy",
# ]
# ///
"""24_monte_carlo.py — モンテカルロ実行による感染規模の推定

実行方法:
    uv run scripts/24_monte_carlo.py
```

```python

ndlib のシミュレーションは確率的なので，単一実行では分散が大きすぎる
場合があります．複数回実行して平均と分位点を取りましょう．

参考文献:
    [1] Metropolis, N. & Ulam, S. (1949). The Monte Carlo method.
        J. Am. Stat. Assoc. 44(247), 335–341.
    [2] Robert, C. P. & Casella, G. (2004). Monte Carlo Statistical Methods
        (2nd ed.). Springer.
    [3] Newman, M. E. J. & Barkema, G. T. (1999). Monte Carlo Methods in
        Statistical Physics. Oxford University Press.
"""
from __future__ import annotations

import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep


def one_run(g: nx.Graph, seed: int, beta: float, gamma: float, steps: int) -> np.ndarray:
    model = ep.SIRModel(g, seed=seed)
    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", beta)
    cfg.add_model_parameter("gamma", gamma)
    cfg.add_model_parameter("fraction_infected", 0.01)
    model.set_initial_status(cfg)
    iters = model.iteration_bunch(steps)
    infected = [it["node_count"][1] for it in iters]
    return np.array(infected)


def main() -> None:
    g = nx.barabasi_albert_graph(800, 3, seed=18)
    n_runs, steps = 30, 100
    runs = np.stack([one_run(g, s, 0.02, 0.01, steps) for s in range(n_runs)])

    mean = runs.mean(axis=0)
    p10 = np.percentile(runs, 10, axis=0)
    p90 = np.percentile(runs, 90, axis=0)
    t = np.arange(steps)

    fig, ax = plt.subplots(figsize=(8, 4.5))
    ax.fill_between(t, p10, p90, alpha=0.3, label="10-90 percentile")
    ax.plot(t, mean, color="black", label="mean")
    ax.set_xlabel("iteration")
    ax.set_ylabel("# infected")
    ax.set_title(f"SIR Monte-Carlo over {n_runs} runs")
    ax.legend()
    fig.tight_layout()
    fig.savefig("figures/24_monte_carlo.png", dpi=120)
    print("Saved: figures/24_monte_carlo.png")


if __name__ == "__main__":
    main()
```

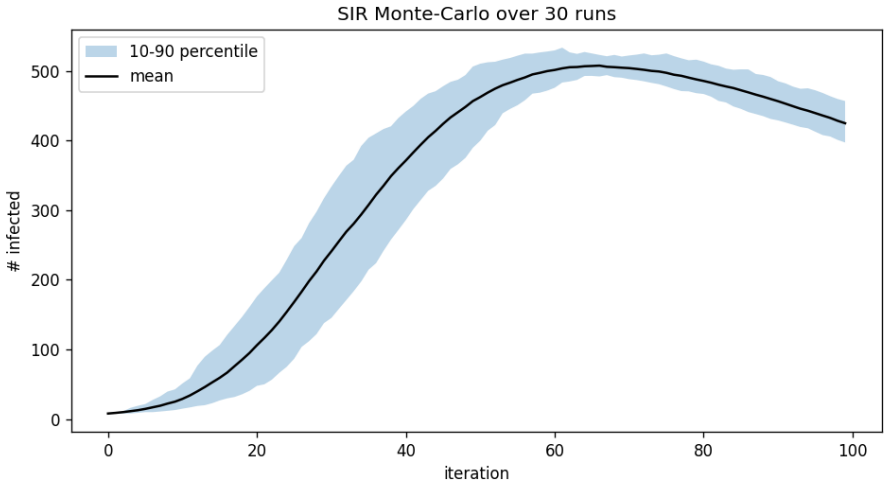

図 23　30 回の SIR モンテカルロ平均と 10–90 パーセンタイル．単発結果に比べて挙動の中心傾向と不確実性を同時に把握できる．

11.2　パラメータスイープ

$\beta \times \gamma$ のグリッド上で最終感染規模を比較し，流行閾値を視覚化する．SIR では流行が終わった後の `Removed` 数が累積感染規模に対応するため，サンプルでは最終ステップの回復者数をヒートマップ化している．

ソース: `scripts/27_parameter_sweep.py`

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
#     "numpy",
# ]
# ///
"""27_parameter_sweep.py — beta x gamma グリッドサーチ

実行方法:
    uv run scripts/27_parameter_sweep.py

beta と gamma の組み合わせを総当たりで動かし，最終感染者数を
ヒートマップで可視化します．

参考文献:
    [1] Diekmann, O., Heesterbeek, J. A. P. & Metz, J. A. J. (1990).
        On the definition and the computation of the basic reproduction
        ratio R0 in models for infectious diseases.
        J. Mathematical Biology 28(4), 365-382.
    [2] Anderson, R. M. & May, R. M. (1991). Infectious Diseases of Humans:
        Dynamics and Control. Oxford University Press. (R0 = beta / gamma)
    [3] Newman, M. E. J. (2002). Spread of epidemic disease on networks.
        Phys. Rev. E 66, 016128.
```

```python
"""
from __future__ import annotations

import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep


def final_recovered(g: nx.Graph, beta: float, gamma: float, steps: int = 100) -> int:
    model = ep.SIRModel(g)
    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", beta)
    cfg.add_model_parameter("gamma", gamma)
    cfg.add_model_parameter("fraction_infected", 0.01)
    model.set_initial_status(cfg)
    iters = model.iteration_bunch(steps)
    return iters[-1]["node_count"][2]


def main() -> None:
    g = nx.barabasi_albert_graph(600, 3, seed=21)
    betas = np.linspace(0.005, 0.05, 8)
    gammas = np.linspace(0.005, 0.05, 8)
    grid = np.zeros((len(betas), len(gammas)))
    for i, b in enumerate(betas):
        for j, gm in enumerate(gammas):
            grid[i, j] = final_recovered(g, b, gm)

    fig, ax = plt.subplots(figsize=(6, 5))
    im = ax.imshow(grid, origin="lower", aspect="auto",
                   extent=(gammas[0], gammas[-1], betas[0], betas[-1]),
                   cmap="viridis")
    fig.colorbar(im, ax=ax, label="final # recovered")
    ax.set_xlabel("gamma")
    ax.set_ylabel("beta")
    ax.set_title("SIR final outbreak size")
    fig.tight_layout()
    fig.savefig("figures/27_parameter_sweep.png", dpi=120)
    print("Saved: figures/27_parameter_sweep.png")


if __name__ == "__main__":
    main()
```

11.3　結果の CSV / JSON エクスポート

node_count と status_delta を表形式で保存しておくと, R, Excel, pandas などで後処理がしやすい. node_count は各時点の状態別人数, status_delta は直前ステップからの増減を表すので, 両方を残しておくと後から流入・流出の解釈を確認できる.

ソース: scripts/26_export_iterations.py

```python
# /// script
```

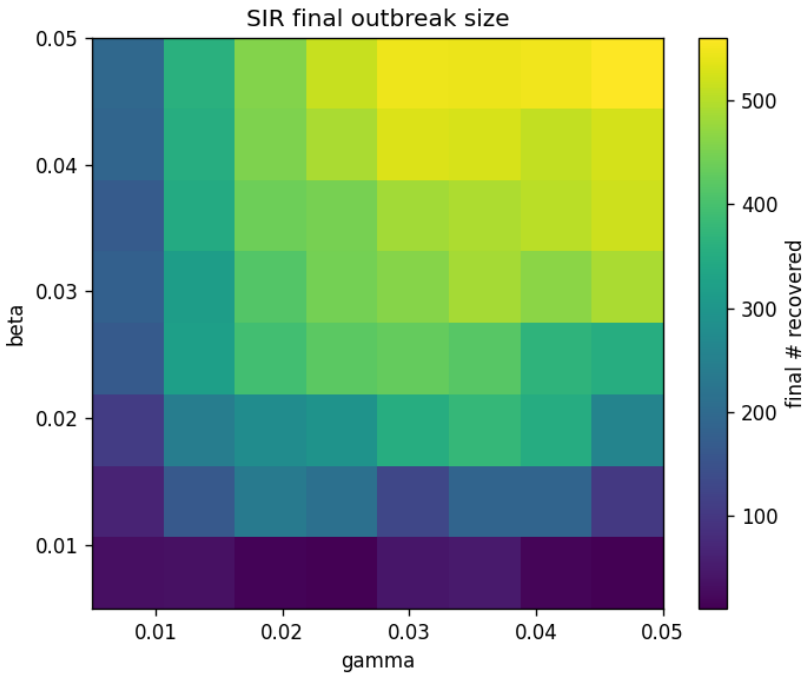

図 24　SIR モデルの最終回復者数を (β, γ) 平面にヒートマップで表示.

```python
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "pandas",
# ]
# ///
"""26_export_iterations.py — シミュレーション結果を CSV/JSON で書き出す

実行方法:
    uv run scripts/26_export_iterations.py

iteration_bunch が返す情報を pandas で整形して保存し,
他のツール(R, Excel など)から読み込みやすくします.

参考文献:
    [1] McKinney, W. (2010). Data structures for statistical computing in
        Python. Proc. 9th Python in Science Conference, 56–61.
    [2] ndlib Iteration / Trends (DiffusionModel API):
        https://ndlib.readthedocs.io/en/latest/developer/ndlib/DiffusionModel.html
"""
from __future__ import annotations

import json
from pathlib import Path

import networkx as nx
import pandas as pd
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep


```

```python
def main() -> None:
    g = nx.barabasi_albert_graph(500, 3, seed=20)
    model = ep.SIRModel(g)
    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.02)
    cfg.add_model_parameter("gamma", 0.01)
    cfg.add_model_parameter("fraction_infected", 0.02)
    model.set_initial_status(cfg)

    iterations = model.iteration_bunch(150)
    rows = [
        {
            "iteration": it["iteration"],
            "S": it["node_count"][0],
            "I": it["node_count"][1],
            "R": it["node_count"][2],
            "delta_S": it["status_delta"].get(0, 0),
            "delta_I": it["status_delta"].get(1, 0),
            "delta_R": it["status_delta"].get(2, 0),
        }
        for it in iterations
    ]
    df = pd.DataFrame(rows)
    out_dir = Path("figures")
    out_dir.mkdir(exist_ok=True)
    df.to_csv(out_dir / "26_iterations.csv", index=False)
    Path(out_dir / "26_iterations.json").write_text(
        json.dumps(rows, indent=2, ensure_ascii=False)
    )
    print(df.tail())
    print("Saved: figures/26_iterations.csv, figures/26_iterations.json")


if __name__ == "__main__":
    main()
```

表 1　figures/26_iterations.csv に出力された SIR シミュレーション結果の一部.

iteration	S	I	R	ΔS	ΔI	ΔR
0	490	10	0	0	0	0
1	490	10	0	0	0	0
2	489	11	0	-1	1	0
148	12	162	326	0	-2	2
149	12	161	327	0	-1	1

12　Composite モデルによるカスタム拡散

ndlib の CompositeModel と compartments を組み合わせると, 状態と遷移ルールを宣言的に書くだけで自分のモデルを作れる. ここでは, 既存の SIR に自発的な除去経路を加えた「ワクチン接種付き SIR」を構築する. サンプルの NodeStochastic は, 対象ノードに感染近傍が少なくとも 1 つあると

きに確率 0.02 で Susceptible から Infected へ遷移させる (triggering_status="Infected"). 感染近傍の人数に比例して確率を増やす設計ではない点に注意する.

ソース: scripts/25_custom_compartment.py

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""25_custom_compartment.py — Compartment 機構によるカスタムモデル

実行方法:
    uv run scripts/25_custom_compartment.py

ndlib.models.CompositeModel と Compartment を組み合わせると,
コードを書かずにオリジナルの感染ダイナミクスを構築できます.
ここでは「ワクチン接種付き SIR」を作成します.
    S --(beta, neighbor=I)--> I
    I --(gamma)--> R
    S --(p_vac)--> R

参考文献:
    [1] Rossetti, G., Milli, L., Rinzivillo, S., Sirbu, A., Pedreschi, D. &
        Giannotti, F. (2018). NDlib: a python library to model and analyze
        diffusion processes over complex networks.
        International Journal of Data Science and Analytics 5(1), 61-79.
    [2] Milli, L., Rossetti, G., Pedreschi, D. & Giannotti, F. (2018).
        Diffusive phenomena in dynamic networks: a data-driven study.
        In International Workshop on Complex Networks (CompleNet 2018),
        Springer Proceedings in Complexity, 151-159.
        DOI: 10.1007/978-3-319-73198-8_13
    [3] ndlib Custom Models (Compartments):
        https://ndlib.readthedocs.io/en/latest/custom/custom.html
"""
from __future__ import annotations

import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.CompositeModel as gc
from ndlib.models.compartments.NodeStochastic import NodeStochastic
from ndlib.viz.mpl.DiffusionTrend import DiffusionTrend


def main() -> None:
    g = nx.erdos_renyi_graph(1000, 0.01, seed=19)
    model = gc.CompositeModel(g)

    model.add_status("Susceptible")
    model.add_status("Infected")
    model.add_status("Removed")

    c1 = NodeStochastic(0.02, triggering_status="Infected")
    c2 = NodeStochastic(0.01)
```

```
    c3 = NodeStochastic(0.005)

    model.add_rule("Susceptible", "Infected", c1)
    model.add_rule("Infected", "Removed", c2)
    model.add_rule("Susceptible", "Removed", c3)

    cfg = mc.Configuration()
    cfg.add_model_parameter("fraction_infected", 0.05)
    model.set_initial_status(cfg)

    trends = model.build_trends(model.iteration_bunch(200))
    DiffusionTrend(model, trends).plot(filename="figures/25_custom_compartment.png")
    print("Saved: figures/25_custom_compartment.png")


if __name__ == "__main__":
    main()
```

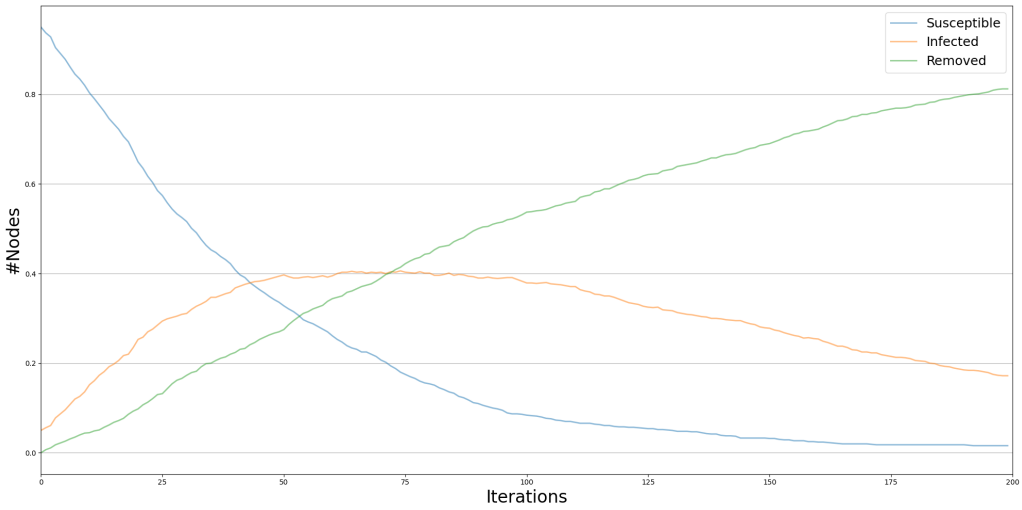

図 25　Composite モデルで作成したワクチン接種付き SIR の拡散過程. 感染, 回復, ワクチン接種による除去をルールとして組み合わせている.

13　可視化

13.1　matplotlib (PNG/PDF)

本書の他の節で多用している標準的な可視化方法. 論文や講義資料へ埋め込む静的図を作る場合は, PNG や PDF として保存できる matplotlib が扱いやすい.

13.2　Bokeh (HTML)

論文付録や Web 配布の都合で対話的グラフが必要な場合は ndlib の `trends` を Bokeh に渡して描画するとよい. 対話的な出力は `figures/21_visualization_bokeh.html`, 本書に埋め込む静的な出力は `figures/21_visualization_bokeh.png` である.

ソース: scripts/21_visualization_bokeh.py

```python
# /// script
# requires-python = ">=3.10"
# dependencies = [
#     "ndlib",
#     "networkx",
#     "bokeh>=3",
#     "matplotlib",
#     "six",
# ]
# ///
"""21_visualization_bokeh.py — Bokeh による対話的可視化

実行方法:
    uv run scripts/21_visualization_bokeh.py

ndlib の trends を Bokeh に渡すと HTML 形式の対話的グラフを書き出せます.
ブラウザで開いて拡大・凡例操作などが可能です.

参考文献:
    [1] Bokeh Development Team (2025). Bokeh: Python library for interactive
        visualization. https://bokeh.org/
    [2] Bokeh User guide:
        https://docs.bokeh.org/en/latest/docs/user_guide.html
"""
from __future__ import annotations

import sys
from pathlib import Path

import matplotlib.pyplot as plt
import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep
from bokeh.io import output_file, save
from bokeh.layouts import gridplot
from bokeh.palettes import Category10
from bokeh.plotting import figure

sys.path.insert(0, str(Path(__file__).parent))
from _plot_helpers import trend_to_ax, prevalence_to_ax  # noqa: E402


def bokeh_panel(model, trends: list, key: str, title: str, ylabel: str):
    statuses = {v: k for k, v in model.available_statuses.items()}
    nn = model.graph.number_of_nodes()
    series_by_status = trends[0]["trends"][key]
    p = figure(
        width=420,
        height=320,
        title=title,
        x_axis_label="iteration",
        y_axis_label=ylabel,
    )
    colors = Category10[10]
    for i, (sid, series) in enumerate(series_by_status.items()):
```

```
        y = [value / nn for value in series]
        p.line(
            list(range(len(series))),
            y,
            line_width=2,
            legend_label=statuses[sid],
            alpha=0.65,
            color=colors[i % len(colors)],
        )
    p.legend.location = "top_right"
    p.legend.click_policy = "hide"
    return p


def main() -> None:
    g = nx.erdos_renyi_graph(1000, 0.01, seed=16)
    model = ep.SIRModel(g)
    cfg = mc.Configuration()
    cfg.add_model_parameter("beta", 0.02)
    cfg.add_model_parameter("gamma", 0.01)
    cfg.add_model_parameter("fraction_infected", 0.05)
    model.set_initial_status(cfg)

    trends = model.build_trends(model.iteration_bunch(150))

    p1 = bokeh_panel(model, trends, "node_count", "Diffusion Trend", "fraction")
    p2 = bokeh_panel(model, trends, "status_delta", "Diffusion Prevalence", "delta fraction")

    output_file("figures/21_visualization_bokeh.html")
    save(gridplot([[p1, p2]]))
    print("Saved: figures/21_visualization_bokeh.html")

    fig, axes = plt.subplots(1, 2, figsize=(11, 4))
    trend_to_ax(model, trends, axes[0])
    axes[0].set_title("Diffusion Trend")
    prevalence_to_ax(model, trends, axes[1])
    axes[1].set_title("Diffusion Prevalence")
    fig.tight_layout()
    fig.savefig("figures/21_visualization_bokeh.png", dpi=120)
    plt.close(fig)
    print("Saved: figures/21_visualization_bokeh.png")


if __name__ == "__main__":
    main()
```

13.3　複数モデルの並列比較

SI, SIS, SIR を同じネットワーク・同じ初期感染割合で並べて比較する．回復状態を持たない SI, 再感染を許す SIS, 免疫獲得を表す SIR の違いを同じ軸で確認できる．

ソース: `scripts/22_multi_model_compare.py`

```
# /// script
# requires-python = ">=3.10"
```

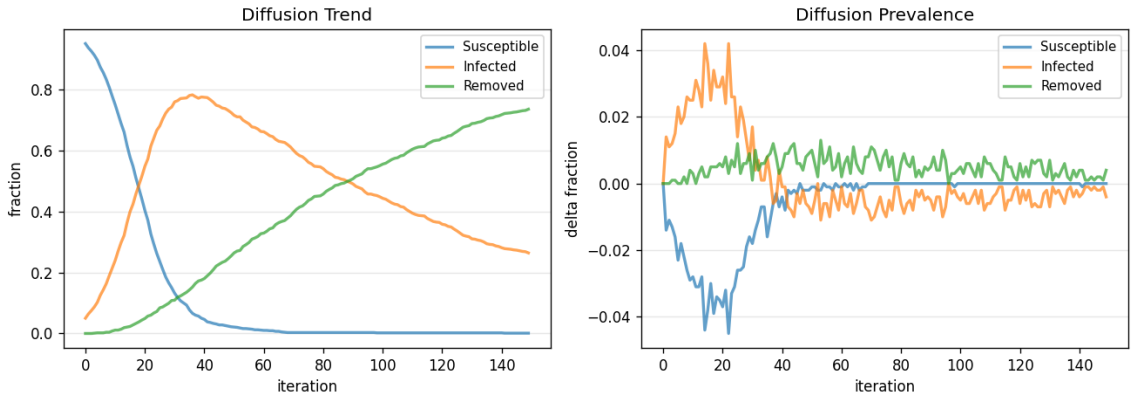

図 26　Bokeh で生成した対話的可視化と同じデータを静的 PNG として書き出したもの．HTML 版では凡例クリックやズーム操作ができる．

```python
# dependencies = [
#     "ndlib",
#     "networkx",
#     "matplotlib",
# ]
# ///
"""22_multi_model_compare.py — 同一ネットワーク上で複数モデルを比較

実行方法:
    uv run scripts/22_multi_model_compare.py

SI / SIS / SIR を同じグラフ・同じ初期条件で実行し，
感染ダイナミクスの違いを並べて表示します．

参考文献:
    [1] Hethcote, H. W. (2000). The mathematics of infectious diseases.
        SIAM Review 42(4), 599-653.
    [2] Keeling, M. J. & Rohani, P. (2008). Modeling Infectious Diseases in
        Humans and Animals. Princeton University Press.
    [3] ndlib MultiPlot:
        https://ndlib.readthedocs.io/en/stable/reference/viz/bokeh/Multiplot.html
"""
from __future__ import annotations

import sys
from pathlib import Path

import matplotlib.pyplot as plt
import networkx as nx
import ndlib.models.ModelConfig as mc
import ndlib.models.epidemics as ep

sys.path.insert(0, str(Path(__file__).parent))
from _plot_helpers import trend_to_ax  # noqa: E402


def configure_si(g: nx.Graph) -> ep.SIModel:
    m = ep.SIModel(g)
    c = mc.Configuration()
```

```
    c.add_model_parameter("beta", 0.02)
    c.add_model_parameter("fraction_infected", 0.05)
    m.set_initial_status(c)
    return m


def configure_sis(g: nx.Graph) -> ep.SISModel:
    m = ep.SISModel(g)
    c = mc.Configuration()
    c.add_model_parameter("beta", 0.02)
    c.add_model_parameter("lambda", 0.01)
    c.add_model_parameter("fraction_infected", 0.05)
    m.set_initial_status(c)
    return m


def configure_sir(g: nx.Graph) -> ep.SIRModel:
    m = ep.SIRModel(g)
    c = mc.Configuration()
    c.add_model_parameter("beta", 0.02)
    c.add_model_parameter("gamma", 0.01)
    c.add_model_parameter("fraction_infected", 0.05)
    m.set_initial_status(c)
    return m


def main() -> None:
    g = nx.erdos_renyi_graph(1500, 0.01, seed=17)

    models = {
        "SI" : configure_si(g),
        "SIS": configure_sis(g),
        "SIR": configure_sir(g),
    }

    fig, axes = plt.subplots(1, 3, figsize=(15, 4))
    for ax, (name, model) in zip(axes, models.items()):
        trends = model.build_trends(model.iteration_bunch(200))
        trend_to_ax(model, trends, ax)
        ax.set_title(name)
    fig.tight_layout()
    fig.savefig("figures/22_multi_model_compare.png", dpi=120)
    print("Saved: figures/22_multi_model_compare.png")


if __name__ == "__main__":
    main()
```

14　すべてを一括実行する

すべてのスクリプトを一度に実行するヘルパが `scripts/30_run_all.py` として用意されている. `figures/` 以下に出力ファイルが揃う. ただし, Bokeh の HTML や CSV/JSON のように PNG 以外を生成するスクリプトも含まれる. いずれかのスクリプトが失敗した場合は, 最後に失敗したファイル名を表示して非ゼロ終了する.

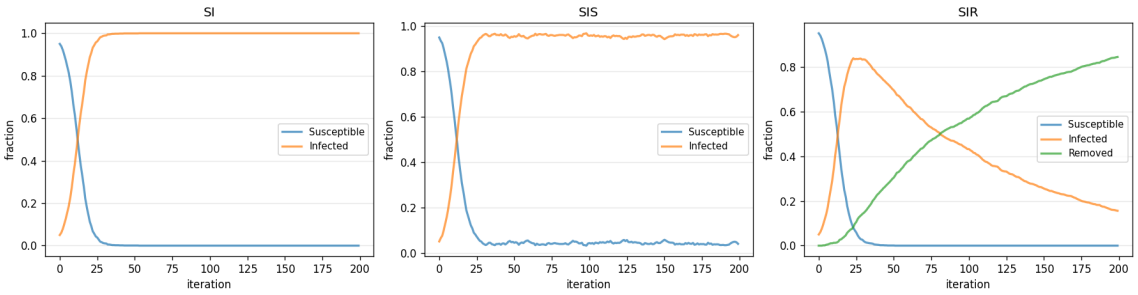

図 27　同一ネットワーク上で実行した SI / SIS / SIR の比較.

ソース: scripts/30_run_all.py

```python
# /// script
# requires-python = ">=3.10"
# dependencies = []
# ///
"""30_run_all.py — 全スクリプトをまとめて実行

実行方法:
    uv run scripts/30_run_all.py

scripts/ 内のサンプルを順次 `uv run` で実行します. 図は figures/ に保存されます.

参考文献:
    [1] Astral. uv: Python package and project manager.
        https://docs.astral.sh/uv/
    [2] PEP 723 — Inline script metadata.
        https://peps.python.org/pep-0723/
"""
from __future__ import annotations

import subprocess
from pathlib import Path


SKIP = {"30_run_all.py", "_plot_helpers.py"}


def main() -> None:
    here = Path(__file__).resolve().parent
    targets = sorted(p for p in here.glob("*.py") if p.name not in SKIP)
    failures: list[str] = []
    for p in targets:
        print(f"\n=== {p.name} ===")
        result = subprocess.run(["uv", "run", str(p)], cwd=here.parent)
        if result.returncode != 0:
            print(f"!! {p.name} exited with {result.returncode}")
            failures.append(p.name)

    if failures:
        failed = ", ".join(failures)
        raise SystemExit(f"Failed scripts: {failed}")

```

```python

if __name__ == "__main__":
    main()
```

15　おわりに

ndlib はモデル数が多いことに加えて, パラメータ・初期条件をきめ細かく制御できる点と Composite モデル機構による拡張性の高さが大きな魅力である. 本書では, まず実行環境と共通 API を確認し, 続いて疫学モデル, 閾値・カスケードモデル, 意見ダイナミクス, 動的ネットワーク, 初期条件の比較, 解析と可視化, Composite モデルへと進んだ. どの例も「グラフを作る, モデルを設定する, 実行する, 結果を読む」という同じ手順で構成している. 本書のサンプルを足がかりに, 経済政策評価, 流言伝播, SNS 上のオピニオン形成, イノベーション拡散などの研究課題へつなげてほしい.

16　用語集

ネットワーク / グラフ
: 人や組織, Web ページなどの対象と, それらの関係をまとめた構造. 数理的にはノードとエッジからなるグラフとして表す.

ノード
: ネットワーク上の点. 感染症モデルでは個体, SNS の例では利用者, イノベーション拡散では企業や消費者に対応する.

エッジ
: ノード同士のつながり. 友人関係, 取引関係, 接触機会などを表す. ndlib では NetworkX のエッジ情報をそのまま利用できる.

次数
: あるノードにつながっているエッジの本数. 次数が大きいノードは多くの相手に影響を与えやすく, ハブと呼ばれることがある.

状態
: 各ノードがその時点で持つ区分. たとえば SIR では感受性状態 S, 感染状態 I, 回復・除去状態 R を使う. 社会的採用モデルでは, `Infected` を「採用済み」や「活性化済み」と読み替える.

拡散プロセス
: ノードの状態が, 近傍ノードとの関係や確率ルールに従って時間とともに変化する過程. 感染, 流言, 意見, 製品採用などを同じ枠組みで扱える.

初期条件
: シミュレーション開始時の状態設定. `fraction_infected` で初期感染・採用割合を指定する方法と, `add_model_initial_configuration` で起点ノードを明示する方法がある.

パラメータ
: モデルの動き方を決める値. たとえば β は感染・伝搬のしやすさ, γ は感染状態から回復・除去状態へ移るしやすさを表す.

乱数 seed

乱数の初期値. 同じ seed を使うと, 同じ条件では同じようなシミュレーション結果を再現しやすい. グラフ生成の seed とモデルの状態遷移に使う seed は区別して考える.

`node_count`

各ステップで状態ごとのノード数を集計した値. 図ではこの値を状態別の時系列として描くことが多い.

`status_delta`

直前ステップから各状態のノード数がどれだけ増減したかを表す値. どのタイミングで流入・流出が起きたかを読むのに役立つ.

SI / SIS / SIR

疫学モデルの基本形. SI は回復しない感染拡大, SIS は感染後に感受性状態へ戻る再感染モデル, SIR は回復後に再感染しない短期流行モデルである.

SEIR / SEIS

潜伏状態 E (Exposed) を含む感染症モデル. 感染してすぐ発症するのではなく, 潜伏期間を経て感染状態へ移る現象を表せる.

SWIR

弱化状態 W (Weakened) を含むモデル. 接触後すぐ感染する経路と, いったん脆弱化してから感染する経路を区別する.

閾値

状態が変わるために必要な境界値. 閾値モデルでは, 近傍の採用割合が自分の閾値を超えると採用状態へ移る.

カスケード

あるノードの状態変化が近傍へ伝わり, さらにその近傍へ連鎖していく現象. 小さな初期変化が大規模な拡散につながる場合がある.

意見ダイナミクス

人々の意見が相互作用によって変化する過程を表すモデル群. 二値意見を扱う Voter モデルから, 連続値意見を扱う Hegselmann–Krause モデルまで幅がある.

有界信頼

自分と近い意見の相手からだけ影響を受けるという考え方. 許容幅が狭いと, 全体が一つにまとまらず複数の意見クラスタが残りやすい.

動的ネットワーク

エッジが時間とともに変わるネットワーク. 接触関係が固定されないため,「いつ誰とつながるか」が拡散結果に影響する.

スナップショット

動的ネットワークをある時点で切り出した静的なネットワーク. ndlib の動的モデルでは, スナップショット列を順に実行する.

モンテカルロ実験

同じ条件のシミュレーションを乱数を変えて何度も実行し, 平均や分位点で典型的な挙動とばらつきを見る方法.

パラメータスイープ

β や γ のようなパラメータを格子状に変えながら何度も実行し, 結果がどの範囲で大きく変わるかを調べる方法.

Composite モデル

ndlib で状態と遷移ルールを組み合わせ, 既製モデルにない拡散過程を宣言的に作るための仕組み.

Compartment

Composite モデルで使う遷移条件の部品. たとえば「感染近傍がいると確率的に感染する」といったルールを表す.

matplotlib

Python の標準的な静的可視化ライブラリ. 論文や講義資料に貼り込む PNG/PDF 図を作る用途に向いている.

Bokeh

Web ブラウザで操作できる対話的な図を作る Python ライブラリ. 凡例クリックやズームを使って結果を探索したい場合に向いている.

PEP 723

Python スクリプトの先頭コメントに依存パッケージなどの実行メタデータを書くための仕様. 本書のサンプルはこの形式を使う.

uv Python の依存関係管理と実行を行うツール. 本書では `uv run scripts/XX_name.py` の形で, 各スクリプトの依存関係を解決して実行する.

■参考文献・リンク

- ndlib 公式ドキュメント https://ndlib.readthedocs.io/en/latest/
- NetworkX 公式ドキュメント https://networkx.org/documentation/stable/
- uv 公式ドキュメント https://docs.astral.sh/uv/
- PEP 723: Inline script metadata https://peps.python.org/pep-0723/
- Rossetti, G., Milli, L., Rinzivillo, S., Sirbu, A., Pedreschi, D., & Giannotti, F. (2018). *NDlib: a python library to model and analyze diffusion processes over complex networks.* International Journal of Data Science and Analytics, 5(1), 61–79. DOI: 10.1007/s41060-017-0086-6.
- Kermack, W. O. & McKendrick, A. G. (1927). A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A*, 115(772), 700–721.
- Granovetter, M. (1978). Threshold models of collective behavior. *American Journal of Sociology*, 83(6), 1420–1443.
- Hegselmann, R. & Krause, U. (2002). Opinion dynamics and bounded confidence: models, analysis and simulation. *Journal of Artificial Societies and Social Simulation*, 5(3), 2.

kkawai@econ.nagoya-cu.ac.jp