Metody Monte Carlo

Laboratorium 5

Zadanie 2

Kod (C++)

Jest to zmodyfikowany skrypt `epidemia.cpp`

```
/*****************************************************************************************
*
*   Program do modelowania rozprzestrzeniania się epidemii na kwadratowej siatce.
*
*   Autor: Dominik Kasprowicz
*   Poprawki: Sebastian Cieślak
*   Ostatnia aktualizacja: 6 maja 2021
*
*****************************************************************************************/

#include <iostream>
#include <fstream>
#include <vector>
#include <list>
#include <iterator> // std::distance()
#include <random>

// Zbiór możliwych stanów osobnika
enum Stan : char
{
    zaszczepiony,
    ozdrowialy,
    podatny,
    chory
};
/*****************************************************************************/

// Położenie osobnika na mapie
struct Koordynaty
{
    int x, y;

    Koordynaty() : x(0), y(0) {}

    Koordynaty(int ax, int ay) : x(ax), y(ay) {}

    Koordynaty sasiad_lewy() const { return Koordynaty(x - 1, y); }

    Koordynaty sasiad_prawy() const { return Koordynaty(x + 1, y); }

    Koordynaty sasiad_gorny() const { return Koordynaty(x, y - 1); }

    Koordynaty sasiad_dolny() const { return Koordynaty(x, y + 1); }

    friend std::ostream &operator<<(std::ostream &ostr, const Koordynaty &wsp)
    {
        ostr << "(" << wsp.x << ", " << wsp.y << ")";
        return ostr;
    }
};
/*****************************************************************************/

struct Wirus
{
    float beta, gamma;

    Wirus() : beta(0.5), gamma(0.5) {}
    Wirus(float b, float g) : beta(b), gamma(g) {}

    friend std::ostream &operator<<(std::ostream &ostr, const Wirus &wirus)
    {
        ostr << "Zaraza o parametrach: beta=" << wirus.beta << ", gamma=" << wirus.gamma <<
std::endl;
```

```
        return ostr;
    }
};
/*****************************************************************************/

// Generator liczb pseudolosowych
class RNG
{
    std::mt19937_64 generator;
    std::uniform_int_distribution<int> losowa_koordynata;
    std::uniform_real_distribution<float> losowa_0_1;

public:
    RNG(int bok_mapy) : generator(132123), losowa_koordynata(0, bok_mapy - 1), losowa_0_1(0, 1) {}

    Koordynaty losuj_koordynaty() { return Koordynaty(losowa_koordynata(generator),
losowa_koordynata(generator)); }
    float losuj_od_0_do_1() { return losowa_0_1(generator); }
};
/*****************************************************************************/

// Populacja jako kwadratowa siatka osobników (a właściwie ich stanów) z przydatnymi metodami.
class Populacja
{
    int bok_mapy; // Długość boku siatki
    // Generator liczb pseudolosowych używany do:
    // 1. losowania współrzędnych osobników zaszczepionych i zarażonych pierwszego dnia,
    // 2. decydowania, czy sąsiad zostanie zarażony (co zachodzi z prawdopodobieństwem beta),
    // 3. decydowania, czy osobnik wyzdrowiał (co zachodzi z prawdopodobieństwem gamma).
    RNG rng;

    // Populacja jako dwuwymiarowa siatka osobników
    std::vector<Stan> osobniki;

    // Pomocnicze listy współrzędnych: osobniki chore i odporne
    std::list<Koordynaty> chorzy_x_y;
    std::list<Koordynaty> odporni_x_y;

    // Parametry epidemii
    Wirus wirus;

    bool jest_na_mapie(const Koordynaty &wsp) const
    {
        return wsp.x >= 0 and wsp.y >= 0 and wsp.x < bok_mapy and wsp.y < bok_mapy;
    }

public:
    Populacja(int n) : bok_mapy(n), rng(bok_mapy) { reset(); }

    void reset()
    {
        osobniki = std::vector<Stan>(bok_mapy * bok_mapy, Stan::podatny);
        chorzy_x_y.clear();
        odporni_x_y.clear();
    }

    int dlugosc_boku() const { return bok_mapy; }

    long liczebnosc() const { return osobniki.size(); }

    long ilu_chorych() const { return chorzy_x_y.size(); }

    long ilu_odpornych() const { return odporni_x_y.size(); }

    long ilu_podatnych() const { return liczebnosc() - ilu_chorych() - ilu_odpornych(); }

    // Funkcja dodana przez Sebastiana Cieślaka.
    long ilu_ozdrowialych()
    {
        long ilu = 0;
        for (Koordynaty &wsp : odporni_x_y)
        {
            if (czy_ozdrowialy(wsp))
                ++ilu;
        }
        return ilu;
    }

    // Funkcja dodana przez Sebastiana Cieślaka.
    long ilu_zaszczepionych()
    {
```

```
        long ilu = 0;
        for (Koordynaty &wsp : odporni_x_y)
        {
            if (czy_zaszczepiony(wsp))
                ++ilu;
        }
        return ilu;
    }

    Stan odczytaj_stan(const Koordynaty &wsp) const { return osobniki[bok_mapy * wsp.x + wsp.y]; }

    void ustaw_stan(const Koordynaty &wsp, const Stan &stan) { osobniki[bok_mapy * wsp.x + wsp.y] =
stan; }

    bool czy_chory(const Koordynaty &wsp) const { return odczytaj_stan(wsp) == Stan::chory; }

    bool czy_zaszczepiony(const Koordynaty &wsp) const { return odczytaj_stan(wsp) ==
Stan::zaszczepiony; }

    bool czy_ozdrowialy(const Koordynaty &wsp) const { return odczytaj_stan(wsp) ==
Stan::ozdrowialy; }

    bool czy_podatny(const Koordynaty &wsp) const { return odczytaj_stan(wsp) == Stan::podatny; }

    bool czy_niepodatny(const Koordynaty &wsp) const { return czy_zaszczepiony(wsp) or
czy_ozdrowialy(wsp); }

    void zamien_osobniki(const Koordynaty &wsp1, const Koordynaty &wsp2)
    {
        Stan stan1 = odczytaj_stan(wsp1);
        ustaw_stan(wsp1, odczytaj_stan(wsp2));
        ustaw_stan(wsp2, stan1);
    }

    // Zwraca listę Koordynat zawierających wyłącznie podatnych sąsiadów
    std::list<Koordynaty> znajdz_podatnych_sasiadow(const Koordynaty &wsp)
    {
        std::list<Koordynaty> sasiedzi;
        for (Koordynaty sasiad : {
                 wsp.sasiad_lewy(),
                 wsp.sasiad_prawy(),
                 wsp.sasiad_dolny(),
                 wsp.sasiad_gorny()})
            if (jest_na_mapie(sasiad) and czy_podatny(sasiad))
                sasiedzi.push_back(sasiad);
        return sasiedzi;
    }

    // Zaszczep losowo wybrane osobniki i zainfekuj inne, również wybrane losowo.
    // Funkcja poprawiona przez Sebastiana Cieślaka.
    void zaraza_przybywa(const Wirus &wir, long ilu_chorych, long ilu_odpornych)
    {
        wirus = wir;

        // Losowe współrzędne osobników odpornych (wyłącznie wśród podatnych).
        for (int i = 0; i < ilu_odpornych; ++i)
        {
            Koordynaty wsp;
            do
            {
                wsp = rng.losuj_koordynaty();
            } while (not czy_podatny(wsp));
            odporni_x_y.push_back(wsp);
            ustaw_stan(odporni_x_y.back(), Stan::zaszczepiony);
        }

        // Losowe współrzędne osobników zarażonych (wyłącznie wśród podatnych).
        for (int i = 0; i < ilu_chorych; ++i)
        {
            Koordynaty wsp;
            do
            {
                wsp = rng.losuj_koordynaty();
            } while (not czy_podatny(wsp));
            chorzy_x_y.push_back(wsp);
            ustaw_stan(chorzy_x_y.back(), Stan::chory);
        }
    }

    // Kolejna "tura" symulacji: chorzy mają szansę wyzdrowieć, podatni mogą się zarazić.
    void kolejny_dzien()
```

```
{
    std::list<Koordynaty> chorzy_nowi;

    // Dla każdego chorego...
    for (Koordynaty &chory_x_y : chorzy_x_y)
    {
        // ...znajdujemy jego podatnych sąsiadów...
        for (Koordynaty &podatny_x_y : znajdz_podatnych_sasiadow(chory_x_y))
        {
            // ... i próbujemy zarazić każdego z nich,
            // co udaje się z prawdopodobieństwem 'beta'.
            if (rng.losuj_od_0_do_1() < wirus.beta)
            {
                // Zainfekuj
                ustaw_stan(podatny_x_y, Stan::chory);
                chorzy_nowi.push_back(podatny_x_y);
            }
        }
    }

    // Osobniki, które były chore już w poprzedniej iteracji, powoli zdrowieją.
    for (Koordynaty &osobnik_x_y : chorzy_x_y)
    {
        if (rng.losuj_od_0_do_1() < wirus.gamma)
        {
            ustaw_stan(osobnik_x_y, Stan::ozdrowialy);
            // Dodaj do listy odpornych
            odporni_x_y.push_back(osobnik_x_y);
        }
    }

    // Usuń ozdrowiałych z listy chorych.
    chorzy_x_y.remove_if([&](const Koordynaty &wsp)
                         { return czy_ozdrowialy(wsp); });

    // Osobniki zarażone w bieżącej iteracji dołączamy do ogólnej puli zarażonych.
    chorzy_x_y.splice(chorzy_x_y.end(), chorzy_nowi);
}

// Zapisuje stan siatki do pliku o podanej nazwie,
bool zapisz_do_pliku(const std::string &nazwa_pliku) const
{
    std::ofstream plik(nazwa_pliku);

    if (not plik.is_open())
    {
        std::cout << " Nie mogę utworzyć pliku '" << nazwa_pliku << "`" << std::endl;
        return false;
    }

    char stan_jako_znak;

    for (int x = 0; x < bok_mapy; ++x)
    {
        for (int y = 0; y < bok_mapy; ++y)
        {
            Stan stan = osobniki[bok_mapy * x + y];
            switch (stan)
            {
            case Stan::chory:
                stan_jako_znak = '3';
                break;
            case Stan::podatny:
                stan_jako_znak = '2';
                break;
            case Stan::ozdrowialy:
                stan_jako_znak = '1';
                break;
            case Stan::zaszczepiony:
                stan_jako_znak = '0';
                break;
            default:
                break;
            }
            plik << stan_jako_znak << '\t';
        }
        plik << std::endl;
    }
    plik.close();
    return true;
}
```

```
};
/*****************************************************************************/

// Kontener do przechowywania dziennych liczności grupy w wybranym stanie, np. zarażonych.
// Udostępnia również podstawowe metody do obróbki statystycznej
// (oczywiście zachęcamy do dodawania własnych).
class Statystyka
{
    // Której grupy dotyczy statystyka: podatnych, zarażonych itp.
    Stan stan;
    std::list<long> grupa;

public:
    Statystyka(const Stan &s) : stan(s) {}

    Stan dotyczy_stanu() const { return stan; }

    void dodaj_dzisiejsze_dane(long ilu) { grupa.push_back(ilu); }

    // Zmiana liczności danej grupy ostatniego dnia.
    long ile_dzisiaj_nowych() const
    {
        if (grupa.size() == 0)
            return 0;
        if (grupa.size() == 1)
            return grupa.back();
        auto ostatni = --grupa.end();
        return *ostatni - *(--ostatni);
    }

    // Maksymalna liczba osobników w danej grupie w czasie trwania eksperymentu.
    long maksimum() const
    {
        auto szczytowy_dzien = grupa.begin();
        for (auto dzis = grupa.begin(); dzis != grupa.end(); ++dzis)
        {
            if (*dzis > *szczytowy_dzien)
                szczytowy_dzien = dzis;
        }
        return *szczytowy_dzien;
    }

    // Na który dzień przypadł szczyt liczności grupy
    int kiedy_maksimum() const
    {
        auto szczytowy_dzien = grupa.begin();
        for (auto dzis = grupa.begin(); dzis != grupa.end(); ++dzis)
        {
            if (*dzis > *szczytowy_dzien)
                szczytowy_dzien = dzis;
        }
        return std::distance(grupa.begin(), szczytowy_dzien);
    }

    // Zlicz dni, w których liczność grupy przekracza podaną wartość
    int ile_dni_powyzej(long ilu) const
    {
        long ile_dni = 0;
        for (auto dzis = grupa.begin(); dzis != grupa.end(); ++dzis)
            if (*dzis > ilu)
                ++ile_dni;
        return ile_dni;
    }

    // Od którego dnia liczność grupy się (aż do końca symulacji)
    // poniżej podanej wartości
    int od_kiedy_ponizej(long ilu) const
    {
        // Będziemy się cofać, poczynając od ostatniego dnia
        auto dzis = grupa.rbegin();
        for (; dzis != grupa.rend(); ++dzis)
            if (*dzis >= ilu)
                break;
        return std::distance(grupa.rbegin(), dzis);
    }

    void wypisz() const
    {
        for (auto x : grupa)
            std::cout << x << '\t';
        std::cout << std::endl;
```

```
    }

    // Zapisuje dane z całego eksperymentu do pliku o podanej nazwie,
    // oddzielając poszczególne rekordy znakiem tabulacji i kończąc
    // znakiem nowego wiersza.
    // Jeśli drugi argument to 'true', nadpisuje plik.
    bool zapisz_do_pliku(const std::string &nazwa_pliku, bool nadpisz = false) const
    {
        std::ofstream plik;

        if (nadpisz)
            plik.open(nazwa_pliku);
        else
            plik.open(nazwa_pliku, std::ostream::app);

        if (not plik.is_open())
        {
            std::cout << " Nie mogę utworzyć pliku '" << nazwa_pliku << "`" << std::endl;
            return false;
        }
        for (auto x : grupa)
            plik << x << '\t';
        plik << std::endl;

        plik.close();
        return true;
    }
};

std::vector<float> argConverter(int argc, char *argv[])
{
    std::vector<float> args;

    for (int count = 1; count < argc; count++)
    {
        std::string s;
        s = argv[count];
        float value = std::stof(s);
        args.push_back(value);
    }

    return args;
}
/*****************************************************************************/

int main(int argc, char *argv[])
{
    std::vector<float> arguments = argConverter(argc, argv);

    // Pierwiastek z liczby osobników (bok kwadratowej siatki).
    // Nie należy bać się liczb rzędu 100 (tysiąca), choć
    // ciekawe ciekawe wyniki można uzyskać i dla 100.
    const int bok_mapy = (int)arguments[0];

    // Liczba osobników zarażonych na początku epidemii.
    const long chorzy_dnia_zero = (long)arguments[5];

    // Liczba osobników zaszczepionych przed nastaniem epidemii.
    const long zaszczepieni_dnia_zero = (long)arguments[6];

    // Prawdopodobieństwo zarażenia każdego z sąsiadów
    // danego osobnika w jednostce czasu.
    const float beta = arguments[2];

    // Prawdopodobieństwo wyzdrowienia w jednostce czasu.
    const float gamma = arguments[3];

    // Liczba niezależnych (!) eksperymentów Monte Carlo.
    const int ile_eksperymentow = (int)arguments[4];

    // Ile dni trwa pojedynczy eksperyment.
    const int ile_dni = (int)arguments[1];

    // Jedno miasto posłuży nam do całej serii eksperymentów Monte Carlo.
    Populacja miasto(bok_mapy);

    for (int eksp_nr = 0; eksp_nr < ile_eksperymentow; ++eksp_nr)
    {
        // std::cout << "Eksperyment " << eksp_nr + 1 << "/" << ile_eksperymentow << std::endl;

        Wirus wirus(beta, gamma);
```

```
        Statystyka chorzy(Stan::chory);
        Statystyka podatni(Stan::podatny);
        Statystyka ozdrowiali(Stan::ozdrowialy);

        // Ten krok (reset) jest konieczny! Wszyscy mieszkańcy stają się na nowo podatni.
        // Stan generatora liczb pseudolosowych NIE jest resetowany, więc kolejny eksperyment
        // będzie miał inny przebieg niż ostatni (i o to chodzi).
        miasto.reset();
        miasto.zaraza_przybywa(wirus, chorzy_dnia_zero, zaszczepieni_dnia_zero);

        // Właściwy eksperyment odbywa się tu.
        for (int dzien = 0; dzien < ile_dni; ++dzien)
        {
            chorzy.dodaj_dzisiejsze_dane(miasto.ilu_chorych());
            podatni.dodaj_dzisiejsze_dane(miasto.ilu_podatnych());
            ozdrowiali.dodaj_dzisiejsze_dane(miasto.ilu_ozdrowialych());
            miasto.kolejny_dzien();
        }

        // UWAGA! Pliki NIE SĄ CZYSZCZONE pomiędzy eksperymentami, co umożliwa
        // zgromadzenie w nich wyników całej serii eksperymentów Monte Carlo.
        // Przed rozpoczęciem nowej serii eksperymentów zaleca się "ręczne" usunięcie plików.
        chorzy.zapisz_do_pliku("chorzy_kazdego_dnia.txt");
        podatni.zapisz_do_pliku("podatni_kazdego_dnia.txt");
        ozdrowiali.zapisz_do_pliku("ozdrowiali_kazdego_dnia.txt");

        // Dla ostatniego z eksperymentów wypiszemy różne charakterystyczne wielkości
        // (głównie po to, żeby pokazać, jak to się robi).
        if (eksp_nr == ile_eksperymentow - 1)
        {

            // UWAGA! Plik z symbolicznie zapisanymi stanami wszystkich osobników może być duży!
            miasto.zapisz_do_pliku("mapa.txt");

            // std::cout << "\n\nPodsumowanie ostatniego z " << ile_eksperymentow << " eksperymentów
Monte Carlo" << std::endl;
            // std::cout << "  Szczyt zachorowań przypada na dzień " << chorzy.kiedy_maksimum() <<
std::endl;
            // std::cout << "  Liczba zarażonych w szczycie to " << chorzy.maksimum();
            // std::cout << " (" << 100 * chorzy.maksimum() / miasto.liczebnosc() << "% populacji)"
<< std::endl;
            // std::cout << "  Liczba zarażonych przekraczała 5% populacji przez "
            //           << chorzy.ile_dni_powyzej(long(0.05 * miasto.liczebnosc())) << " dni" <<
std::endl;
            // std::cout << "  Liczba zarażonych utrzymuje się poniżej 5% populacji od dnia "
            //           << chorzy.od_kiedy_ponizej(long(0.05 * miasto.liczebnosc())) << std::endl;
        }
    }
    return 0;
}
```

Kod (Python)

Odpowiedzialny za testy na skrypcie epidemii, analizę danych oraz do rysowania wykresów.

```python
#!/usr/bin/python

from matplotlib import pyplot as plt
import numpy as np
import subprocess as p


def main():
  # compile()
  # cleanup_results()
  experiment_vacc_sus()
  # experiment_sus_beta()
  # experiment_sus_gamma()


def compile():
  p.call([
      '/usr/bin/g++', '-fdiagnostics-color=always', '-lgsl', '-std=c++20',
      '-lstdc++', '-Wc++11-extensions', '-g', 'epidemia.cpp', '-o', 'epidemia'
  ])


def cleanup_results():
  p.call([
      'rm', 'mapa.txt', 'ozdrowiali_kazdego_dnia.txt',
      'podatni_kazdego_dnia.txt', 'chorzy_kazdego_dnia.txt'
  ])


def flush(text):
  print(f'\r{text}', flush=True, end='')


def simulate(axis, days, beta, gamma, experiments, day0_ill, day0_vacc):
  p.call([
      './epidemia',
      str(axis),
      str(days),
      str(beta),
      str(gamma),
      str(experiments),
      str(day0_ill),
      str(day0_vacc)
  ])


def experiment_vacc_sus():
  # Initial values
  days = 200
  experiments = 5
  day0_ill = 5
  beta = 0.5
  gamma = 0.25
  size = 100
  total_population = size**2

  vacc_population_space = np.linspace(0,
                                      total_population - size,
                                      size + 1,
                                      dtype=int)
  results_space = []

  for index, vacc in enumerate(vacc_population_space):
    flush(f'Vaccinate simulation nr {index}/{len(vacc_population_space) - 1}')
    simulate(axis=size,
             days=days,
             beta=beta,
             gamma=gamma,
             experiments=experiments,
             day0_ill=day0_ill,
             day0_vacc=vacc)

    results = np.loadtxt('./podatni_kazdego_dnia.txt', unpack=True)
    results_space.append(results.min())
```

```
    cleanup_results()

  flush('Simulation completed')
  plt.figure()
  plt.plot([v / size for v in vacc_population_space], results_space, 'r-')
  plt.title('Średnia podatnych osób od procentu populacji osoób zaszczepionych')
  plt.xlabel('Procent zaszczepionych')
  plt.ylabel('Liczba osób podatnych')
  plt.grid()


def experiment_sus_beta():
  # Initial values
  days = 200
  experiments = 5
  day0_ill = 5
  beta = 0.3
  gamma = 0.25
  size = 100
  total_population = size**2

  # Stage 2 - Beta dependency, assuming 30% of vaccined
  beta_space = np.linspace(0.01, 1, size)
  results_space = []

  for index, beta in enumerate(beta_space):
    flush(f'Beta Simulation nr {index}/{len(beta_space) - 1}')
    simulate(axis=size,
             days=days,
             beta=beta,
             gamma=gamma,
             experiments=experiments,
             day0_ill=day0_ill,
             day0_vacc=total_population * 0.3)

    results = np.loadtxt('./podatni_kazdego_dnia.txt', unpack=True)
    results_space.append(results.min())
    cleanup_results()

  flush('Simulation completed')
  plt.figure()
  plt.plot(beta_space, results_space, 'b-')
  plt.title('Średnia podatnych osób od wsp. Beta')
  plt.xlabel('Współczynnik Beta')
  plt.ylabel('Liczba osób podatnych')
  plt.grid()


def experiment_sus_gamma():
  # Initial values
  days = 200
  experiments = 5
  day0_ill = 5
  beta = 0.3
  gamma = 0.25
  size = 100
  total_population = size**2

  # # Stage 3 - Gamma dependency with Beta 0.5 and vaccinated 30%
  gamma_space = np.linspace(0.1, 1, size)
  results_space = []

  for index, gamma in enumerate(gamma_space):
    flush(f'Gamma simulation nr {index}/{len(gamma_space) - 1}')
    simulate(axis=size,
             days=days,
             beta=beta,
             gamma=gamma,
             experiments=experiments,
             day0_ill=day0_ill,
             day0_vacc=total_population * 0.3)

    results = np.loadtxt('./podatni_kazdego_dnia.txt', unpack=True)
    results_space.append(results.min())
    cleanup_results()

  flush('Simulation completed')
  plt.figure()
  plt.plot(gamma_space, results_space, 'g-')
  plt.title('Średnia podatnych osób od wsp. Gamma')
  plt.xlabel('Współczynnik Gamma')
```

```
  plt.ylabel('Liczba osób podatnych')
  plt.grid()


def experiment1():
  _, axis = plt.subplots(2, 2)
  plt.tight_layout(h_pad=2)
  x_scale = 100
  vacc_percent = 0.3

  for day, ax in zip([0, 100, 200, 500],
                     [axis[0, 0], axis[0, 1], axis[1, 0], axis[1, 1]]):
    simulate(axis=x_scale,
             days=day,
             day0_ill=5,
             day0_vacc=x_scale * x_scale * vacc_percent)
    plot_map(ax)
    ax.set_title(f'Day {day}')

  plt.show()


def plot_map(plt=plt, show=False):
  matrix = np.loadtxt('./mapa.txt', unpack=True)

  side = range(len(matrix[0]))
  X, Y = np.meshgrid(side, side)
  Z = [[int(r) for r in row] for row in matrix]

  plt.pcolormesh(X, Y, Z)
  if show:
    plt.show()


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

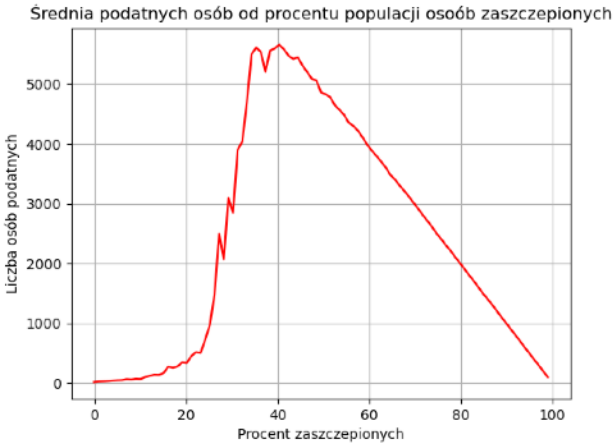
Założenia

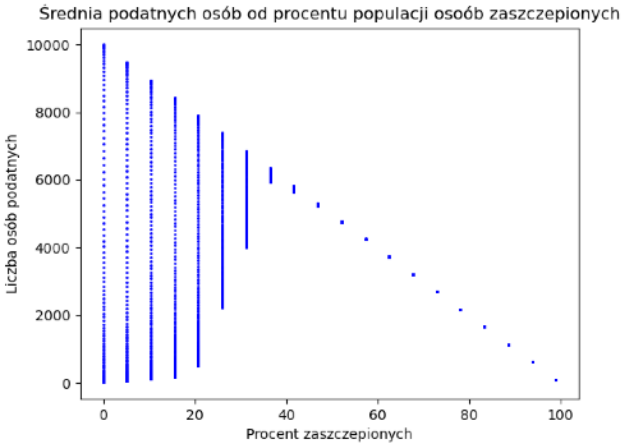
Przyjęte standardowe dane wejściowe (wzorowane na przykładowym pliku epidemia.cpp):

- Rozmiar populacji: **10000**
- Dni: **200**
- Współczynnik beta: **0.5**
- Współczynnik gamma: **0.25**
- Liczba eksperymentów: **5**
- Początkowa liczba osób zarażonych: **5**

Liczba dni jest odpowiednio długa aby pozwolić na pełne rozwinięcie się choroby na przestrzeni całej populacji.
Liczba eksperymentów wynosi 5 aby zredukować wpływ pseudolosowości wyników dziennych. Spośród 5 eksperymentów jest przyjmowana średnia dzienna liczba.
Inicjalizacja choroby zachodzi jedynie na 5 osobnikach. Symuluje to początek epidemii na dostatecznym poziomie realizmu.

Wyniki





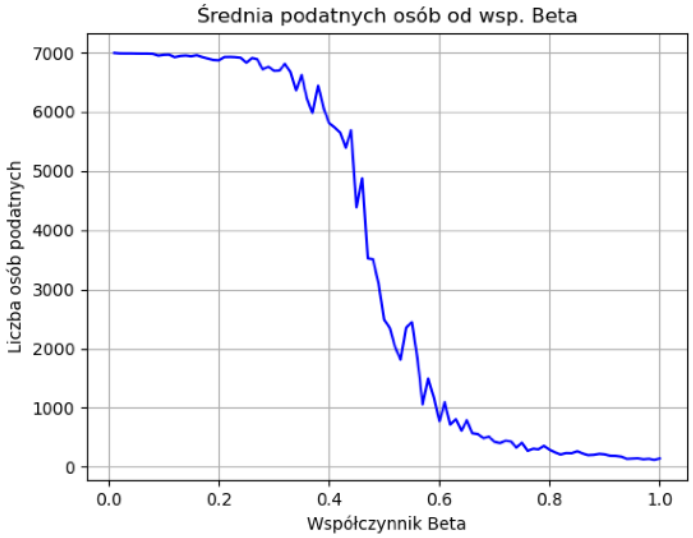
Powyższy wykres (czerwony) został stworzony w oparciu o próbki, rozłożone na przestrzeni 200 dni. Próbki zostały przedstawione na drugim wykresie (niebieskim). W każdej kolumnie znajduje się 200 markerów, odpowiadającym dniom. W perspektywie upływającego czasu populacja osobników podatnych maleje i zbiega do minimalnej wartości. Najważniejszą częścią do analizy wydaje się być właśnie minimalna ilość podatnych osobników, w każdym cyklu symulacji.

Wykres osobników podatnych można podzielić na dwie części. Pierwsza część trwa od początku do szczytu liczby osób podatnych, przy około 40% zaszczepionej populacji. Druga część to odcięcie wykresu z powodu fizycznie mniejszej liczby osób podatnych. Grupa ta jest wypierana przez obiekty zaszczepione odwrotnie proporcjonalnie, dlatego przypomina funkcję liniową.

Prawidłową analizę możemy wykonać w pierwszej części wykresu.
Obserwujemy utrzymywanie się liczby osób podatnych na niskim poziomie do wyszczepienia populacji na poziomie 23-25%. Wartość ta następnie gwałtownie rośnie i osiąga szczyt w okolicach 35 oraz 40%.

Przy założonych danych wejściowych możemy przyjąć, że *wartość progowa* odsetka osób zaszczepionych, powyżej której udaje się uniknąć rozprzestrzenienia patogenu na całą populację wynosi około 25-30%.

Kolejnym etapem symulacji była modyfikacja parametrów beta i gamma, przy założeniu zaszczepienia populacji na poziomie 30%.



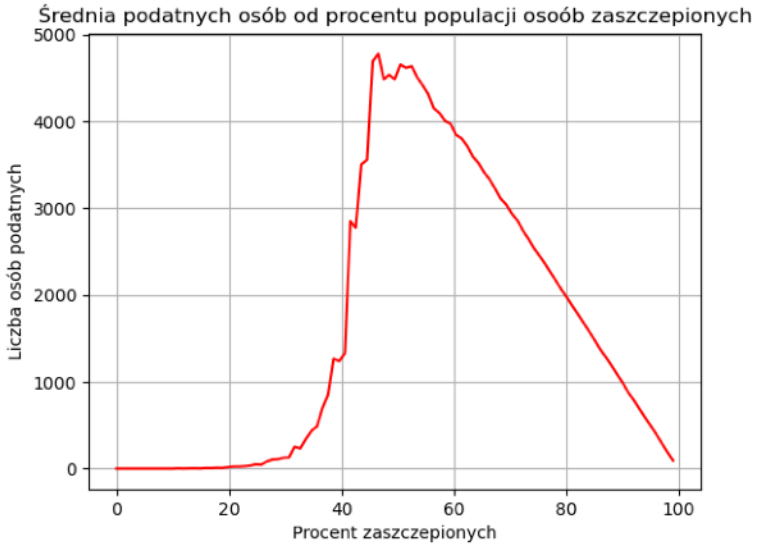
Zakładając optymalną część zaszczepionej populacji na poziomie 30% współczynnik beta potrafi znacznie wpłynąć na przebieg epidemii. W części od 0.0 do 0.4 odnotowuje
się niską transmisję choroby na populację. W przypadku 0.6 do 1.0 przenoszenie choroby jest znacznie częstsze, wyszczepienie 30% nie wystarcza aby większość populacji stanowiły osobniki podatne.
Wartość współczynnika beta 0.5 wydaje się być optymalną wartością pod względem balansu przebiegu symulacji.

Zakładając optymalną część zaszczepionej populacji na poziomie 30% współczynnik gamma także znacząco wpływa na przebieg epidemii. W części od 0.0 do 0.2 odnotowuje
się wysoką transmisję choroby w populacji. W przypadku 0.4 do 1.0 osobniki zarażone zdrowieją na tyle szybko, że zapobiegają rozwojowi epidemii, wyszczepienie 30% w zupełności wystarcza aby większość populacji stanowiły osobniki podatne.
Wartość współczynnika gamma 0.25 wydaje się oddawać balans najmniej wpływający na pozostałe parametry symulacji.

Następna analiza dotyczy wpływu współczynników beta i gamma na *wartość progową* populacji podatnej.

Wariant pesymistyczny Beta = 0.9 Gamma = 0.1

Populacja osobników podatnych jest w strefie ryzyka aż do poziomu około 42-45% populacji zaszczepionej. W tym przypadku osoby podatne stanowią ledwie 5% całej populacji.



Wariant optymistyczny Beta = 0.3 Gamma = 0.3

Populacja osobników podatnych jest bezpieczna już od poziomu około 15-20% populacji zaszczepionej, rozwój choroby jest skutecznie blokowana przez szybko zdrowiejące osobniki i niską transmisję. W tym przypadku osoby podatne stanowią powyżej 80% populacji.