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.

#!/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):

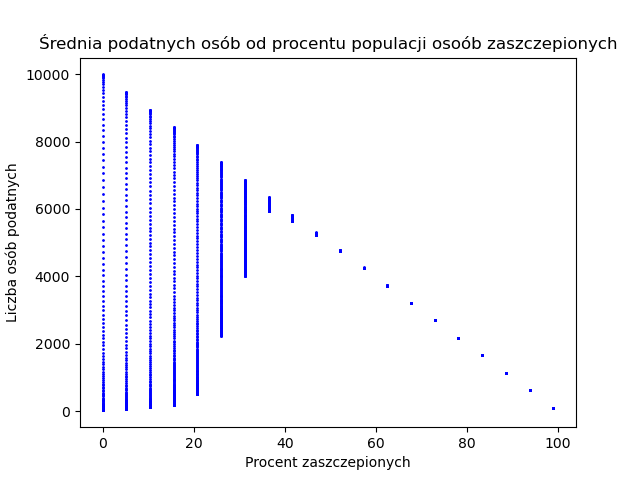
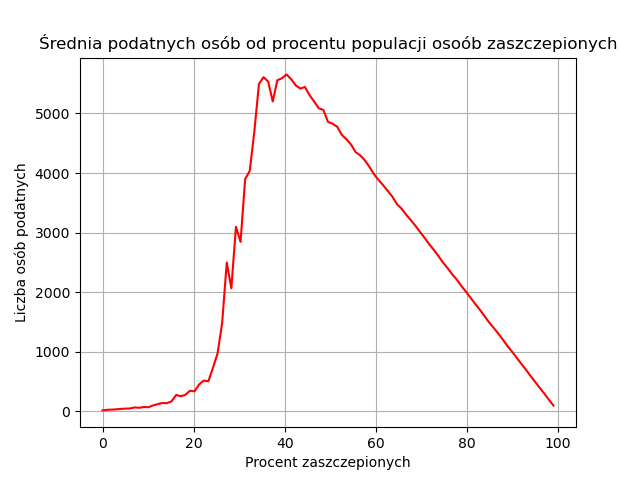
* Rozmiar populacji: **10000**
* Dni: **200**
* Współczynnik beta: **0.5**
* Współczynnik gamma: **0.25**
* Liczba eksperymentów: **5**
* Początkowa liczba osób zarażonych: **5**

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.

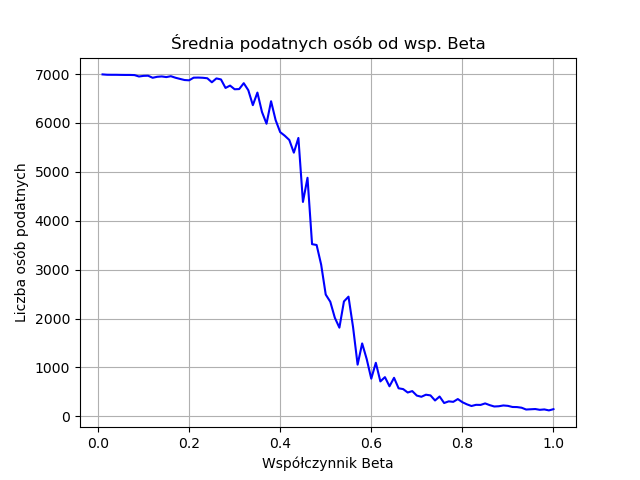
Wykres osobników podatnych można podzielić na dwie części. Pierwsza część trwa od początku do szczytu liczby osób podatnych, przy około 40% zaszczepionej populacji. Druga część to odcięcie wykresu z powodu fizycznie mniejszej liczby osób podatnych. Grupa ta jest wypierana przez obiekty zaszczepione odwrotnie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

Prawidłową analizę możemy wykonać w pierwszej części wykresu.

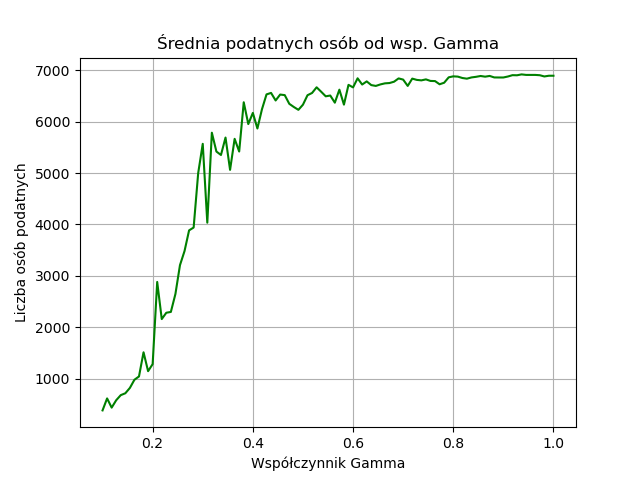
Obserwujemy utrzymywanie się liczby osób podatnych na niskim poziomie do wyszczepienia populacji na poziomie 23-25%. Wartość ta następnie gwałtownie rośnie i osiąga szczyt w okolicach 35 oraz 40%.

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.

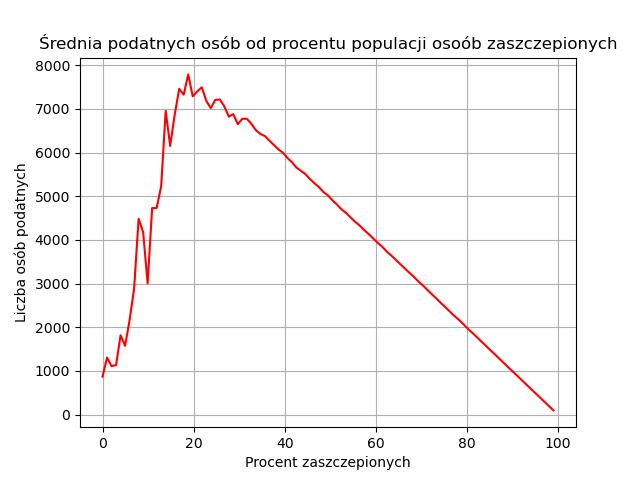
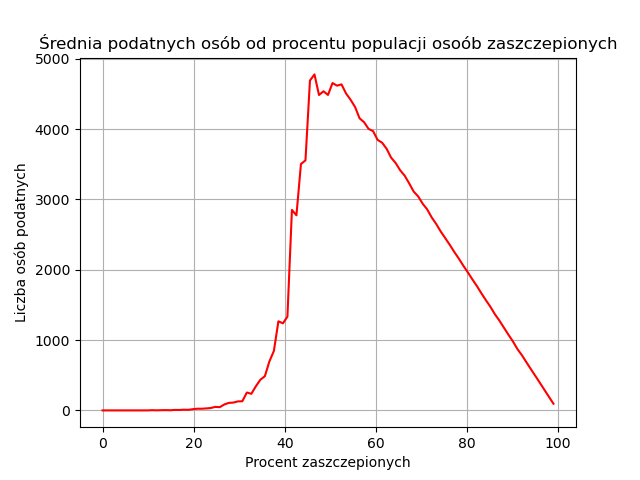


Zakładając optymalną część zaszczepionej populacji na poziomie 30% współczynnik gamma także znacząco wpływa na przebieg epidemii. W części od 0.0 do 0.2 odnotowuje się wysoką transmisję choroby w populacji. W przypadku 0.4 do 1.0 osobniki zarażone zdrowieją na tyle szybko, że zapobiegają rozwojowi epidemii, 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.