Mateusz Jan Mateja
Fizyka Techniczna 4 rok
Wizualizacja Zjawisk Fizycznych (w ramach zaliczenia przedmiotu Symulacje Komputerowe)
Projekt zaliczeniowy Rozwiązanie równanie łańcucha pokarmowego(food chain), które jest uogólnieniem równania Lotki-Volterry na większą liczbę gatunków (dla 3 równań).

Cała aplikacja opiera się na publikacji A Lotka-Volterra Three-species Food Chain (http://math.bd.psu.edu/~jpp4/mathmag243-255.pdf)

Grafika przedstawia model z powyższej publikacji:

The model

The ecosystem that we wish to model is a linear three-species food chain where the lowest-level prey x is preyed upon by a mid-level species y, which, in turn, is preyed upon by a top level predator z. Examples of such three-species ecosystems include: mouse-snake-owl, vegetation-hare-lynx, and worm-robin-falcon. The model we propose to study is

$$\begin{cases} \frac{dx}{dt} = ax - bxy \\ \frac{dy}{dt} = -cy + dxy - eyz \\ \frac{dz}{dt} = -fz + gyz, \end{cases}$$

for $a, b, c, d, e, f, g > 0$, where a, b, c and d are as in the Lotka-Volterra equations and:

- e represents the effect of predation on species y by species z,
- f represents the natural death rate of the predator z in the absence of prey,
- g represents the efficiency and propagation rate of the predator z in the presence of prey.

Since populations are nonnegative, we will restrict our attention to the nonnegative octant $\{(x, y, z) \mid x \geq 0, y \geq 0, z \geq 0\} \subset \mathbb{R}^3$ and the positive octant $\mathbb{R}^3_+ = \{(x, y, z) \mid x > 0, y > 0, z > 0\} \subset \mathbb{R}^3$.

Cały software jest podzielony na dwie części:

1) Kod generujący wyniki w języku C++. Zaimplementowana metoda Rungego-Kutty 4 rzędy(klasyczną) (http://www.phy.davidson.edu/FacHome/dmb/py200/RungeKuttaMethod.htm) i wykonane nią całkowanie. Wyniki zapisywane są w postaci kolumn w pliku .csv o nazwie przybierającej wartości poszczególnych parametrów. (przykładowo 1_1_1_1_1_1_1.6.csv)
2) Kod analizujący dane, w Pythonie (wersja 2.7), który odczytuje wygenerowane pliki .csv, i rysuje wykresy przy pomocy bibliotek Matplotlib, jednocześnie zapisuje ich wersje na dysk w postaci plików .pdf oraz .png (w związku z użyciem bilbioteki Matplotlib można również zapisać w innych formatach). Program również posiada możliwość sprawdzenia wyników przy pomocy tzw. 'solvera' – bilbioteki SciPy.

Instalacja programu krok po kroku jest zawarta w pliku README.md (dla systemów typu UNIX).

Dla kodu generującego wyniku C++ (plik **/src/main.cpp**):

1. Wykorzystanie boost.program_options →bilbioteka pozwalająca dodanie do programu opcji podanych przez użytkownika z linii komend
http://www.boost.org/doc/libs/1_59_0/doc/html/program_options.html

2. Wykorzystanie boost.filesystem →bardzo użyteczna biblioteka w celu wylistowania informacji o plikach i folderach http://www.boost.org/doc/libs/1_59_0/libs/filesystem/doc/tutorial.html

3. Wykorzystanie fstream → w celu obsługi plików
http://www.cplusplus.com/reference/fstream/fstream/

4. Wykorzystanie sstream → w celu operacji na stringach

http://www.cplusplus.com/reference/sstream/stringstream/

Najistotniejsza w tej części całego softwaru jest klasa FoodChain, która zawiera deklaracje poszczególnych parametrów oraz funkcję służącą do obliczania wyników przy pomocy Metody Rungego Kutty IV rzędu:

```
void next_step(double x_prev, double y_prev, double z_prev) {
    K0[0] = f_x_p(x_prev, y_prev, z_prev);
    K0[1] = f_y_p(x_prev, y_prev, z_prev);
    K0[2] = f_z_p(x_prev, y_prev, z_prev);

    K1[0] = f_x_p(x_prev + halfTStep * K0[0], y_prev + halfTStep * K0[1], z_prev + halfTStep * K0[2]);
    K1[1] = f_y_p(x_prev + halfTStep * K0[0], y_prev + halfTStep * K0[1], z_prev + halfTStep * K0[2]);
    K1[2] = f_z_p(x_prev + halfTStep * K0[0], y_prev + halfTStep * K0[1], z_prev + halfTStep * K0[2]);

    K2[0] = f_x_p(x_prev + halfTStep * K1[0], y_prev + halfTStep * K1[1], z_prev + halfTStep * K1[2]);
    K2[1] = f_y_p(x_prev + halfTStep * K1[0], y_prev + halfTStep * K1[1], z_prev + halfTStep * K1[2]);
    K2[2] = f_z_p(x_prev + halfTStep * K1[0], y_prev + halfTStep * K1[1], z_prev + halfTStep * K1[2]);

    K3[0] = f_x_p(x_prev + halfTStep * K2[0], y_prev + halfTStep * K2[1], z_prev + halfTStep * K2[2]);
    K3[1] = f_y_p(x_prev + halfTStep * K2[0], y_prev + halfTStep * K2[1], z_prev + halfTStep * K2[2]);
    K3[2] = f_z_p(x_prev + halfTStep * K2[0], y_prev + halfTStep * K2[1], z_prev + halfTStep * K2[2]);

    X.push_back(x_prev + stepsize / 6.0 * (K0[0] + 2.0*K1[0] + 2.0 * K2[0] + K3[0]));
    Y.push_back(y_prev + stepsize / 6.0 * (K0[1] + 2.0*K1[1] + 2.0 * K2[1] + K3[1]));
    Z.push_back(z_prev + stepsize / 6.0 * (K0[2] + 2.0*K1[2] + 2.0 * K2[2] + K3[2]));
}

void calculate(){
    if(calculated)
        return;
    for (int i = 0; i < T.size() - 1; ++i) {
        next_step(X[i], Y[i], Z[i]);

    }
    calculated = true;
}
```

Możliwe opcje programu wraz z krótkim opisem i domyślnymi wartościami parametrów (./foodchain –help):

```
This program calculates solution to the Foodchain problem(System of equations to be solved)

dx/dt = ax - bxy
dy/dt = -cy + dxy -eyz
dz/dt = -fz + gyz
Allowed options for calculating solution to Lotka-Volterra equation:
  --help                    produce help message
  --steps arg (=1000000)    Number of steps
  --stepsize arg (=0.0001) Stepsize
  --xmin arg (=0.5)         Starting population of prey
  --ymin arg (=0.5)         Starting population of intermediate prey / predator
  --zmin arg (=2)           Starting population of end predator
  --a arg (=1)              a parameter of the equations
  --b arg (=1)              b parameter of the equations
  --c arg (=1)              c parameter of the equations
  --d arg (=1)              d parameter of the equations
  --e arg (=1)              e parameter of the equations
  --f arg (=1)              f parameter of the equations
  --g arg (=0.88)           g parameter of the equations
  --output arg              Output filename / directory (default is current
                            directory)
```

Dla Kodu analizującego dane w Pythonie (plik **/python/foodchain/__init.py__**):

1. Wykorzystanie NumPy → jest podstawowym pakietem wykorzystywanym do obliczeń naukowych w języku Python. Pozwala między innymi na wykonywanie wydajnych operacji na macierzach, obliczenia numeryczne, obliczenia z zakresu algebry liniowej, FFT etc. Stanowi darmową alternatywę dla MATLAB-a. (http://www.numpy.org/)

2. Wykorzystanie Matplotlib →biblioteka do tworzenia wykresów dla języka programowania Python i jego rozszerzenia numerycznego NumPy. (http://matplotlib.org/)

3. Wykorzystanie Pandas → do optymalnego przeczytania plików .csv (http://pandas.pydata.org/)

```
input_file = args['input_file']
with open(input_file, 'r') as csvfile:
    df = pandas.read_csv(csvfile, sep='\t', quotechar='|')
```

4. Wykorzystanie SciPy → do sprawdzenia wyników (http://www.scipy.org/)

```
X0 = np.array([df['X'][0], df['Y'][0], df['Z'][0]])
X = scipy.integrate.odeint(foodchain(a, b, c, d, e, f, g), X0, t)
X = X - np.array([df['X'], df['Y'], df['Z']]).transpose()
X = abs(X)
print "MAXIMUM ERROR: %s" % X.max()
```

5. Wykorzystanie argparse → do parsowania opcji podanych przez użytkownika z linii komend

(https://docs.python.org/3/library/argparse.html)

Możliwe opcje programu (foodchain_py –help):

```
usage: foodchain_py [-h] [--show] [--no-check] input_file output

Front end for C++ foodchain solver

positional arguments:
  input_file  CSV input file beeing output of the C++ program
  output      Output directory, depending on the extension aproperiate files
              will be written

optional arguments:
  -h, --help  show this help message and exit
  --show      If set the output graphs will be also shown in addition to being
              written to disk
  --no-check  If specified loaded data will not be validated against
              scipy.integrate.odeint function
```

Podsumowanie:

Dzieki udanej implementacji rozwiązania równania łańcucha pokarmowego(food chain), które jest

uogólnieniem równania Lotki-Volterry na większą liczbę gatunków (dla 3 równań) zostały potwierdzone wyniki zawarte w publikacji (http://math.bd.psu.edu/~jpp4/mathmag243-255.pdf)

(Te przykładowe pliki .csv oraz wygenerowane wykresy są zawarte w folderze **EXAMPLE SOLUTIONS**)

WYKRES 1:

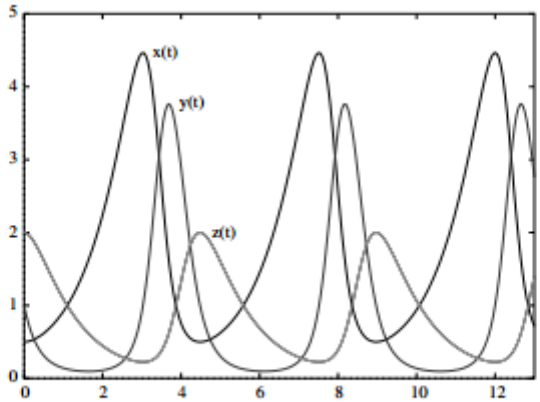
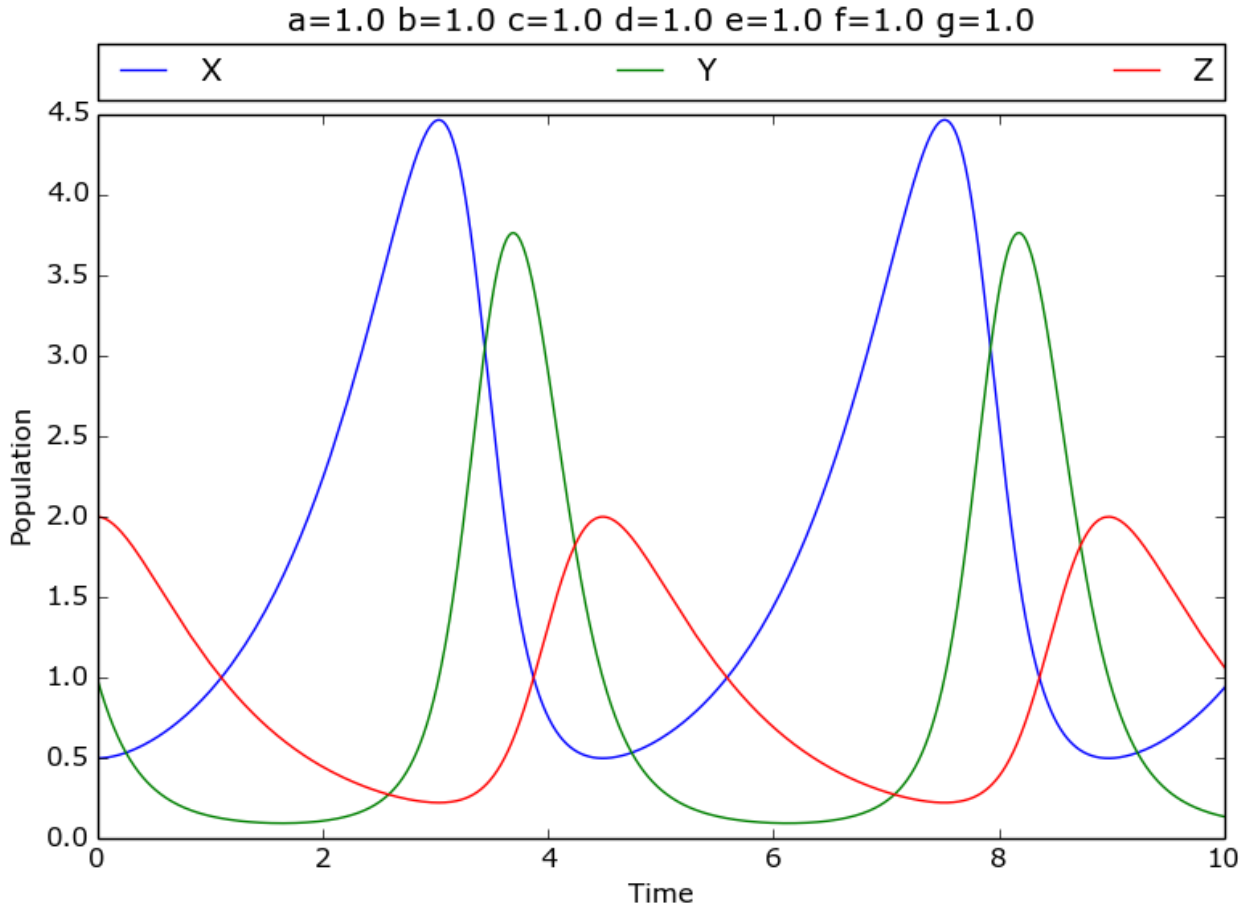


Figure 8 A solution with initial conditions $(x, y, z) = (.5, 1, 2)$ with parameters $a = b = c = d = e = f = g = 1$

Publikacja



Solucja z programu

WYKRES 2:

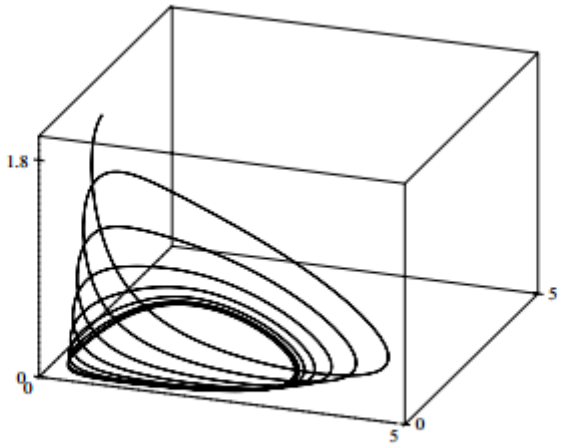
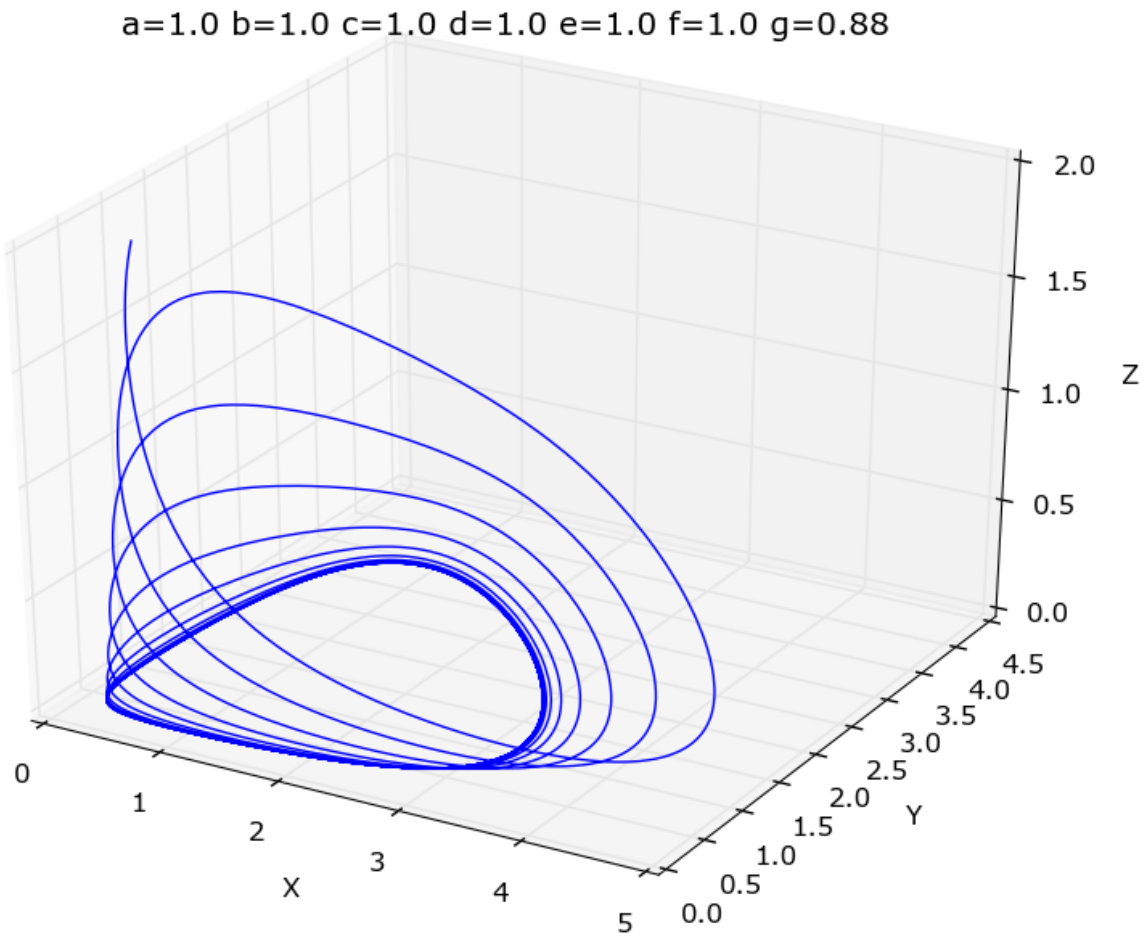


Figure 9 A trajectory in xyz-space with initial conditions $(\frac{1}{2}, 1, 2)$ with $a = b = c = d = e = f = 1$ and $g = 0.88$

Publikacja



Solucja z programu

WYKRES 3:

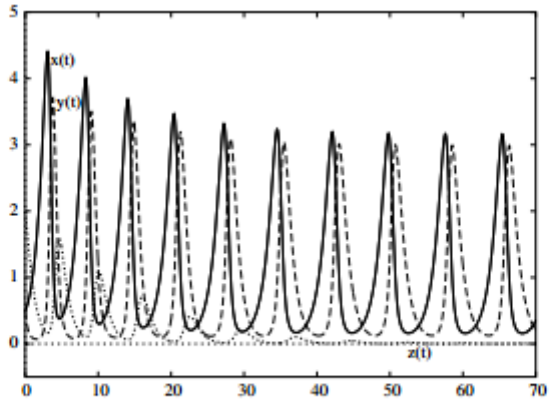
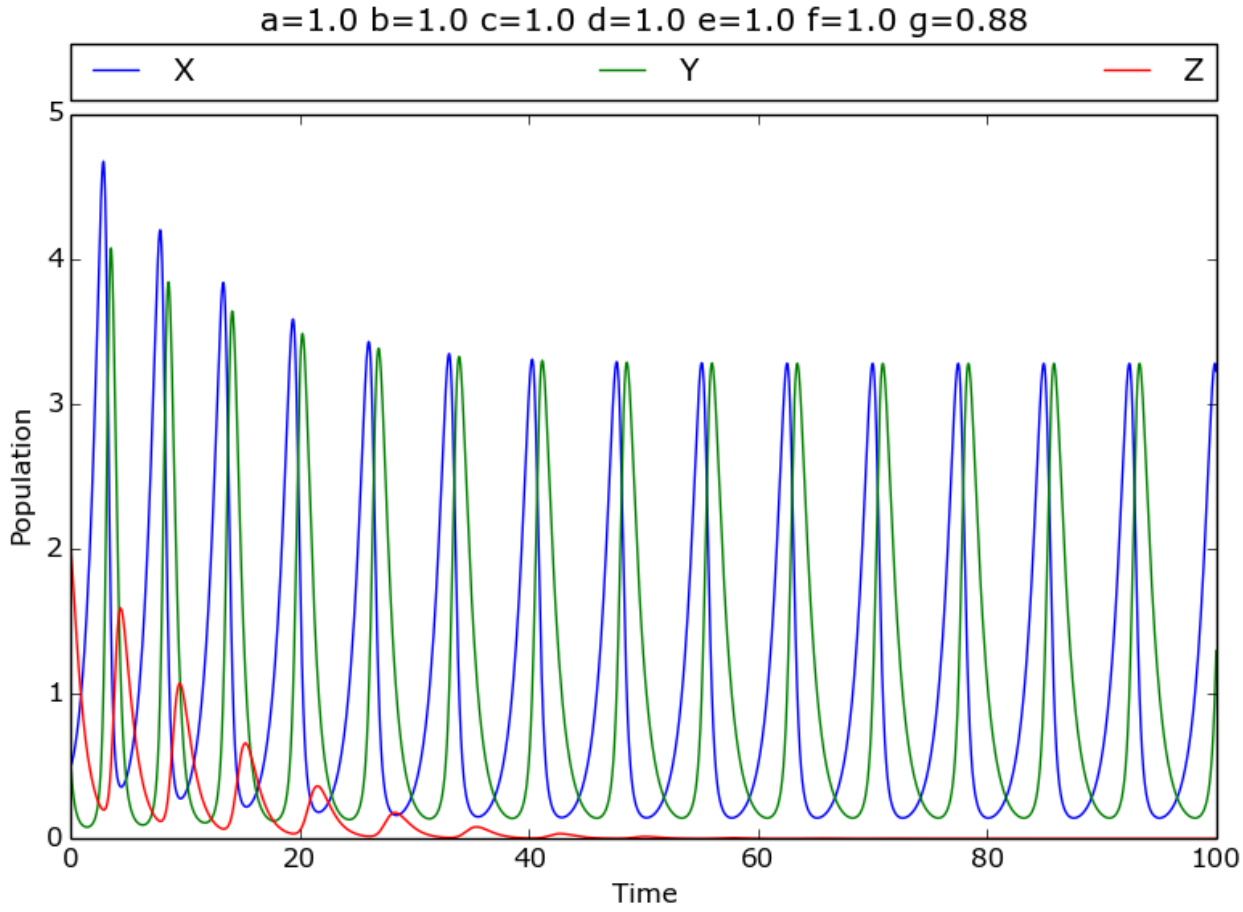


Figure 11 A solution with initial conditions (1/2), (1/2), 2) with $a = b = c = d = e = f = 1$ and $g = 0.88$

Publikacja



Solucja z programu

WYKRES 4:

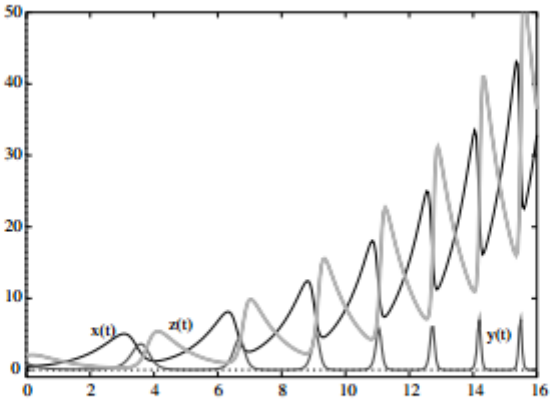
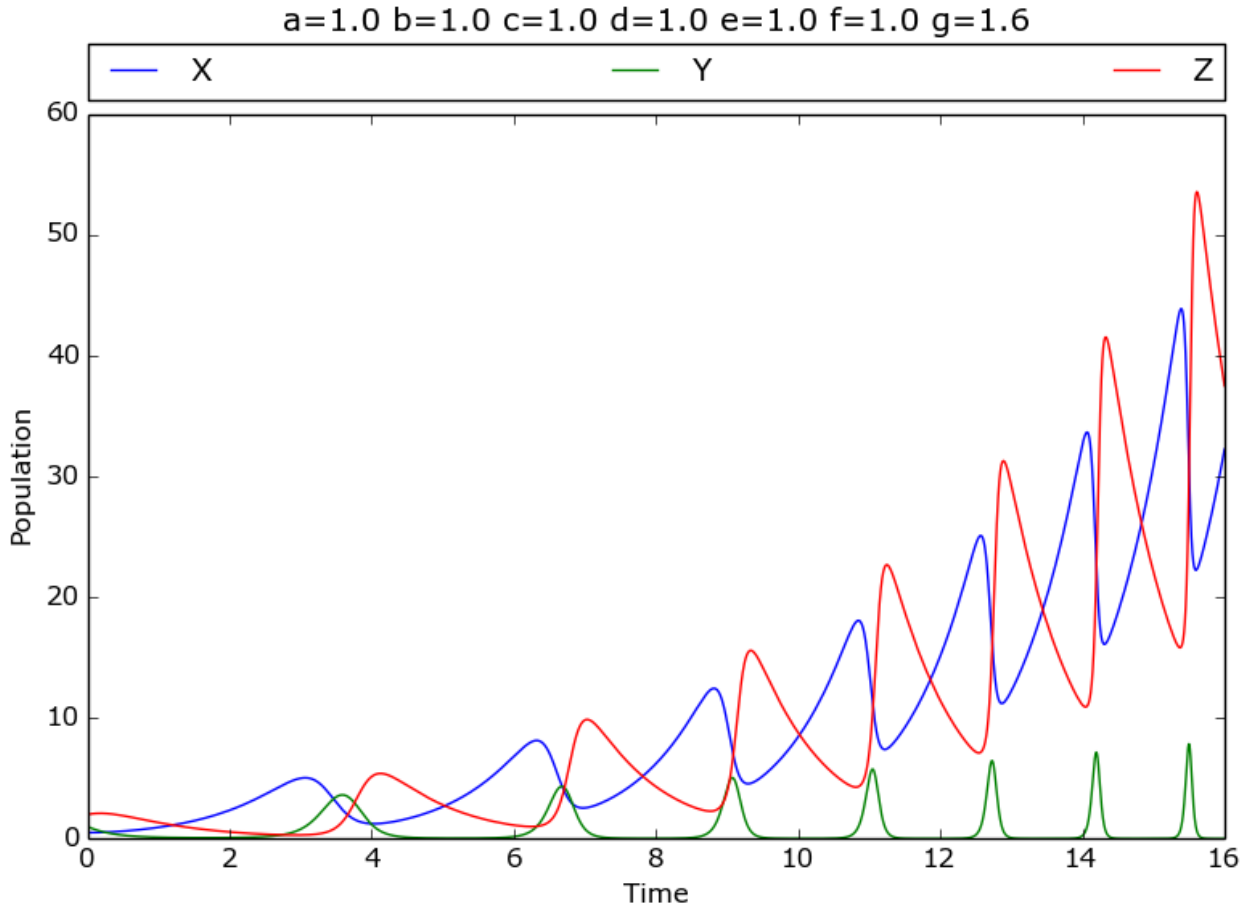


Figure 12 A solution with initial conditions $(x, y, z) = (.5, 1, 2)$ with parameters $g = 1.6$ and $a = b = c = d = e = f = 1$

Publikacja



Solucja z programu