POLITECHNIKA ŚLĄSKA

WYDZIAŁ INŻYNIERII BIOMEDYCZNEJ

Katedra Biosensorów i Przetwarzania Sygnałów Biomedycznych

Modelowanie Struktur i Procesów Biologicznych

Projekt

Podstawowy model Lotki-Volterry i jego modyfikacja z ograniczeniem pojemności środowiska dla ofiar

Suchanek Kamil - PiAIB semestr 1

Prowadząca projekt: Dr Barbara Mika

Bytom, 9 czerwca 2019r.

Streszczenie

Celem poszerzenia horyzontów w dziedzinie modelowania w biologii i medycynie wykonano projekt obejmujący stworzenie aplikacji GUI w języku programowania Ruby oraz dokonania za jej pomocą analizy wpływu parametru śmiertelności drapieżców w biologicznym modelu Lotki-Volterry w formie podstawowej oraz z ograniczeniem pojemności środowiska dla ofiar. W ramach analizy dokonano porównania wpływu zwiększania i zmniejszania wspomnianego parametru na tempo zmian oraz uzyskiwane amplitudy przebiegów symulacji modelu. Przeprowadzone badanie potwierdziło powzięte złożenia, iż parametr śmiertelności drapieżników ma wpływ na tempo wzrostu i zmniejszania się liczebności populacji drapieżników oraz na amplitudę populacji ofiar.

Abstract

In order to broaden the horizons in the field of modeling in biology and medicine, a project involving the implementation of GUI applications in the Ruby programming language and making using it the analysis of the impact of the predator mortality parameter in the biological model of Lotka-Volterra basic model and with the limitation of the environment capacity of the victims. As part of the analysis, the effect of increase and decrease of the parameter on the rate of change as well as the obtained amplitudes of simulations of the model were compared. The study confirmed the submitted assumptions that the predators mortality rate has an impact on the rate of growth and decreasing the population of predators and on the amplitude of the population of the victims.

Spis treści

1. Wstęp - pojęcie modelu matematycznego

Model rozumiany jest jako dwuczęściowa struktura zawierająca opis teoretyczny danego zjawiska na podstawie pewnej wiedzy. Opis teoretyczny nazywany jest modelem heurystycznym, który odzwierciedlany jest przy pomocy zapisu matematycznego stanowiącego drugi element struktury modelu.

Tworzenie modelu heurystycznego polega na dobraniu procesów, czynników mających wpływ na końcowy efekt oraz odrzuceniu tych, które nie wywierają na owy efekt wpływu. Dzięki tej czynności możliwe jest zredukowanie ilości zmiennych i parametrów stosowanych do konstrukcji struktury matematycznej modelu.

Przyjmując formę modelu należy jasno wydzielić na podstawie wiedzy empirycznej co przyjąć za zmienne a co za parametry, zmienne są poszukiwanymi niewiadomymi a parametry wynikami doświadczeń. Wartościowy model służy jako przedmiot badań analitycznych i komputerowych.

Poprawnie skonstruowany model ma jednoznaczne, stabilne względem warunków początkowych i parametrów rozwiązania. Model należy zweryfikować, poprzez przeprowadzenie odpowiednich eksperymentów. O ile wyniki przeprowadzonych prób nie zaprzeczają wnioskom z modelu to model jest poprawny. Niezależnie od liczby eksperymentów potwierdzających wystarczy tylko jeden falsyfikujący aby uznać model za niewłaściwy [1].

2. Model Lotki-Volterry

Pierwszy model opisujący oddziaływania dwóch populacji w ekosystemie dotyczył układu drapieżnik-ofiara. Został on zaproponowany równolegle jako model populacyjny przez Vito Volterrę oraz jako model łańcucha reakcji biochemicznych przez Alfreda Lotkę. Vito zaobserwował z pozoru absurdalne zjawisko tuż po I wojnie światowej. Mianowicie, po ustaniu działań wojennych, gdy ludzie wrócili do wykonywania swych zawodów, rybacy odkryli, że populacja ryb drapieżnych w Adriatyku zwiększyła się. Uznano to za paradoks, zakładając, że populacje ryb powinny ucierpieć w wyniku prowadzonych działań wojennych. Jednak Volterra na bazie swojego modelu wykazał, że wzrost liczebności drapieżników był całkiem naturalny, ponieważ w czasie wojny zaprzestano połowów i dzięki temu populacja drapieżników mogła wrócić do stanu naturalnego. Ponad to, zaproponowany model odzwierciedla znane w ekologii prawo zachowania średnich, stanowiące, że w naturalnych siedliskach zmiany liczebności populacji w czasie zachodzą tak, że zachowana zostaje liczebność średnia. Układ równań różniczkowych dynamiki obu populacji wygląda następująco (1.) [1]:

$$\begin{cases} \dot{V} = rV - aVP \\ \dot{P} = -sP + abVP \end{cases} \tag{1.}$$

gdzie dla uproszczenia została pominięta zmienna niezależna t. Poszczególne składniki układu mają następującą interpretację:

- V oraz P *oznaczają odpowiednio populację ofiar i drapieżników,*
- rV *i* $-sP$ *opisują wewnętrzną dynamikę odpowiednio gatunku ofiary i drapieżnika,*
- r *jest współczynnikiem rozrodczości populacji ofiar,*
- s *jest współczynnikiem śmiertelności populacji drapieżników,*
- aVP *odzwierciedla liczbę losowych spotkań osobników obu gatunków,*

- *a jest współczynnikiem skuteczności polowania, można go interpretować jako biomasę upolowanych ofiar,*
- *współczynnik b odzwierciedla część upolowanej biomasy, którą gatunek drapieżców przeznacza na reprodukcję,*
- *współczynnik ab bywa zastępowany oznaczeniem c*

2.1. Modyfikacja modelu

Podstawowy wariant modelu Lotki-Volterry pomija wiele zależności występujących w naturze, co staje się jednym z powodów krytyki tego modelu, z tego też powodu powstały jego różne modyfikacje.

Jedną z modyfikacji modelu Lotki-Volterry jest ograniczenie pojemności środowiska dla ofiar. Należy założyć, że wewnętrzna dynamika gatunku ofiar rządzi się innymi prawami niż gatunki opisywane w modelu Malthusa, który zakłada nieograniczoną pojemność środowiska. Najprostszym modelem opisującym pożądaną dynamikę jest równanie logistyczne, zatem układ równań będzie się prezentował następująco, przyjmując oznaczenia z (1.):

$$\begin{cases} \dot{V} = rV\left(1 - \frac{V}{K}\right) - aVP \\ \dot{P} = -sP + abVP \end{cases} \tag{2.}$$

gdzie K jest współczynnikiem pojemności środowiska.

3. Cel i założenia

Celem pracy jest stworzenie aplikacji GUI oraz określenie przy jej pomocy wpływu parametru śmiertelności drapieżników na dynamikę modelu Lotki-Volterry. W dalszej części dokumentu parametr śmiertelności drapieżników określono literą s, zgodnie z przyjętym oznaczeniem we wstępie (punkt 2, Model Lotki-Volterry). Aby zrealizować projekt zostaną wykonane czynności:

- Utworzenie aplikacji GUI w języku programowania Ruby. aplikacja ma umożliwić:
  - wygenerowanie przebiegów czasowych symulacji dla modelu podstawowego i zmodyfikowanego,
  - wygenerowanie portretów fazowych dla każdej z symulacji,
  - zapisywanie każdej wykonanej symulacji do późniejszego wglądu.
- Przeprowadzenie analizy dynamiki modelu Lotki-Volterry przy dwóch wariantach:
  - wersji podstawowej modelu,
    - wpływ zmiany parametru s.
  - modyfikacji modelu o ograniczenie pojemności środowiska dla ofiar.
    - wpływ zmiany parametru s przy stałej wartości parametru pojemności środowiska.

Założono, że zmiana parametru śmiertelności drapieżników powinna mieć wpływ na amplitudę oraz tempo zmian liczności obu populacji, w tym:

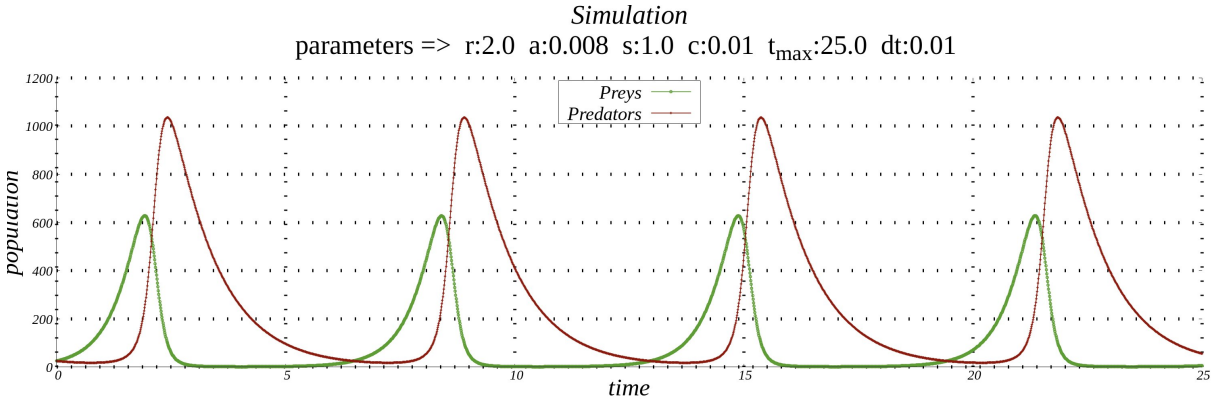
- Zwiększenie parametru s powinno skutkować:
  - zwiększeniem tempa wymierania populacji drapieżców,
  - zmniejszeniem tempa odradzania się populacji drapieżców,
  - zmniejszeniem maksymalnej, lokalnej amplitudy populacji drapieżców,
  - zwiększeniem maksymalnej, lokalnej amplitudy populacji ofiar.
- Zmniejszenie parametru s powinno skutkować:
  - zmniejszeniem tempa wymierania populacji drapieżników,
  - zwiększeniem tempa odradzania się populacji drapieżców,
  - zwiększeniem maksymalnej, lokalnej amplitudy populacji drapieżców,
  - zmniejszeniem maksymalnej, lokalnej amplitudy populacji ofiar.

4. Symulacja i analiza modelu podstawowego

Zostało wykonane porównanie wpływu parametru s na osiągane amplitudy i tempo ich zmian w czasie. W tabeli nr. 1 znajdują się parametry analizowanych symulacji. Oznaczenie parametrów jest zgodne z przyjętym w punkcie 2, Model Lotki-Volterry. Wykonane symulacje znajdują się na rysunkach nr. 1-3.

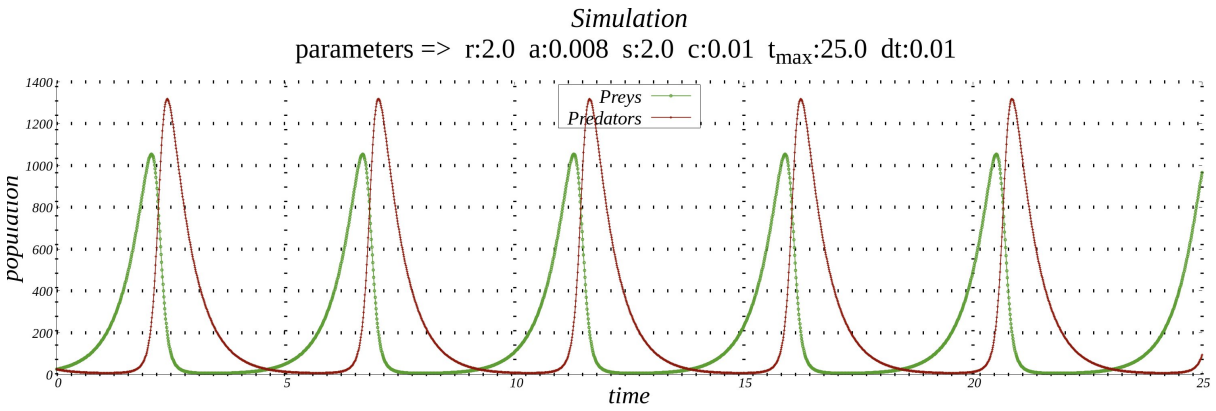
Tabela 1. Parametry symulacji.

Nr. symulacji	r	a	s	c
1	2.0	0.008	1.0	0.01
2	2.0	0.008	2.0	0.01
3	2.0	0.008	3.0	0.01

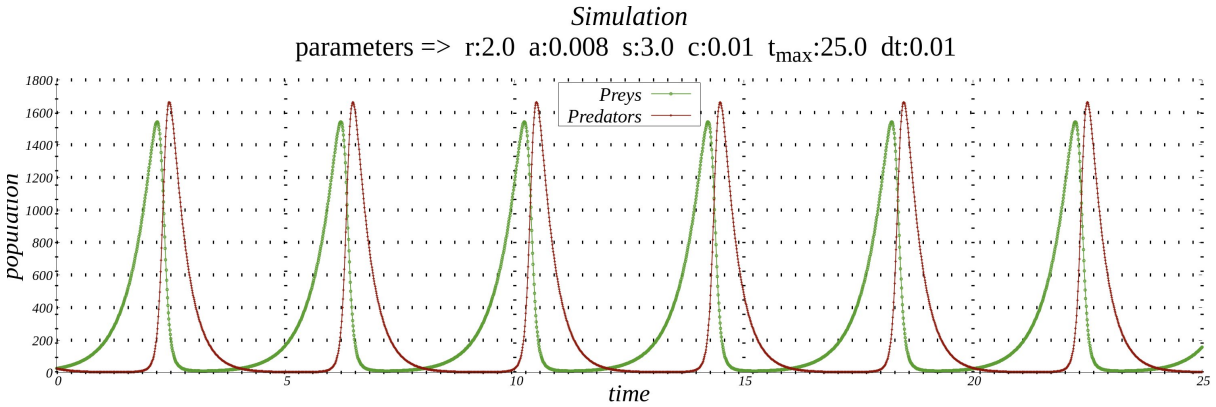


Rys. 1. Przebiegi czasowe symulacji dla s=1.

Na rysunku nr. 1 widać wzrost liczby ofiar, poprzedzający wzrost liczby drapieżców. Zmiany cyklicznie się powtarzają. Na rysunku nr. 2 znajduje się symulacja dla parametru $s = 2$. Zauważalna jest większa amplituda zarówno populacji ofiar jak i populacji drapieżników. Widoczne jest również większe tempo zmian na rysunku nr. 2 w porównaniu do pierwszej symulacji.

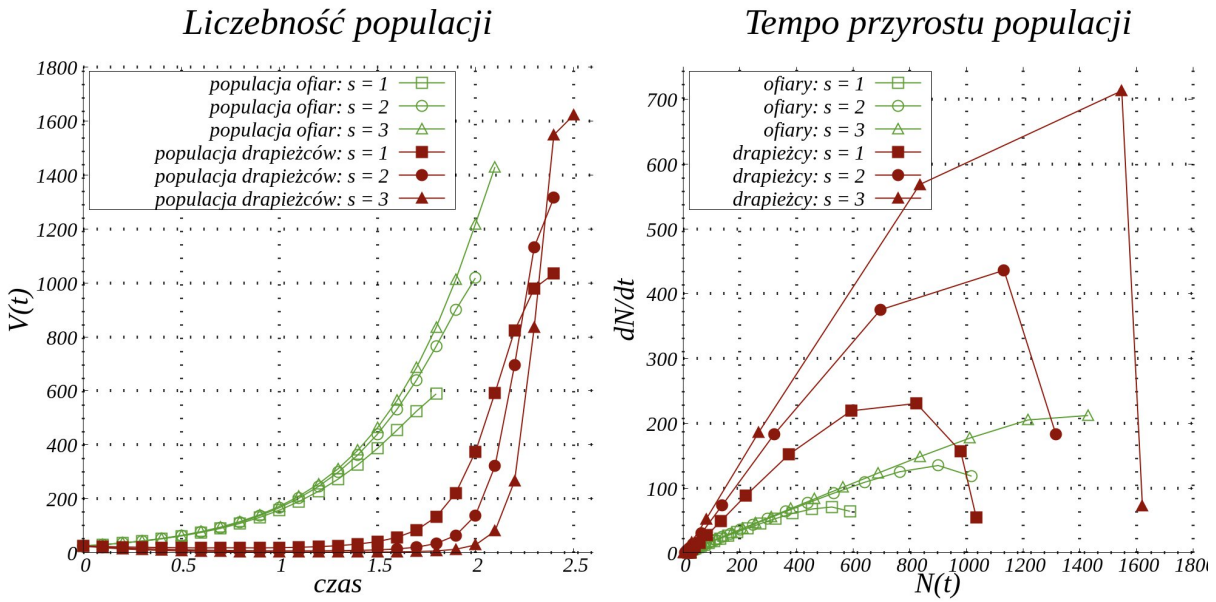


Rys. 2. Przebiegi czasowe symulacji dla s=2.



Rys. 3. Przebiegi czasowe symulacji dla s=3.

Na rysunku nr. 3 pojawiają się zmiany o takim samym charakterze jak na rysunku nr. 2 w stosunku do przebiegów czasowych z pierwszej symulacji. Porównanie zmian liczebności ofiar i drapieżników oraz tempa ich wzrostu w wykonanych symulacjach znajduje się na rysunku nr. 4.



Rys. 4. Liczebność obu populacji oraz tempo ich przyrostu w trakcie trzech symulacji.

Na podstawie zestawienia z rysunku nr. 4 można wywnioskować, że:

- zwiększenie parametru s spowodowało:
  - zwiększenie liczności populacji ofiar,
  - zwiększenie liczności populacji drapieżców,
  - zwiększenie tempa przyrostu populacji drapieżców,
  - zwiększenie tempa przyrost populacji ofiar.

Zestawienie maksymalnych, lokalnych amplitud obu populacji znajduje się w tabeli nr. 2.

Tabela 2. Zestawienie maksymalnych liczebności obu populacji.

Nr. symulacji	s	Maksymalna liczebność populacji ofiar	Maksymalna liczebność populacji drapieżców
1	**1.0**	**589**	**1034**
2	**2.0**	**1018**	**1316**
3	**3.0**	**1430**	**1621**

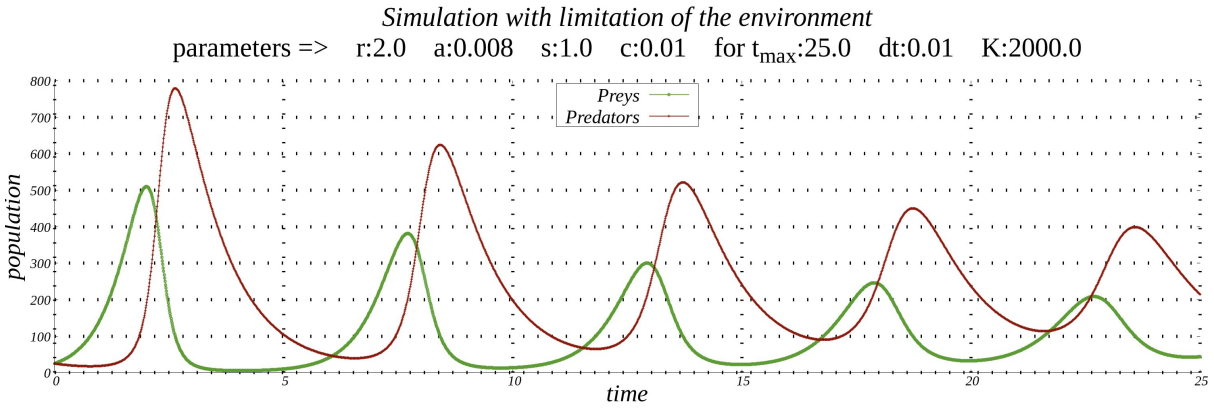
5. Symulacja i analiza modelu z ograniczeniem pojemności środowiska dla ofiar

Wykonano symulację modelu zmodyfikowanego o ograniczenie pojemności środowiska dla ofiar. Tak samo jak w punkcje 5, Symulacja i analiza modelu podstawowego, wykonano porównanie wpływu zmiany wartości parametru s na dynamikę modelu. W tabeli nr. 3 znajduje się zestawienie wykorzystanych do symulacji parametrów. Oznaczenia parametrów są zgodne z użytym w punkcie 2, Model Lotki-Volterry.

Tabela 3. Parametry symulacji modelu z modyfikacją.

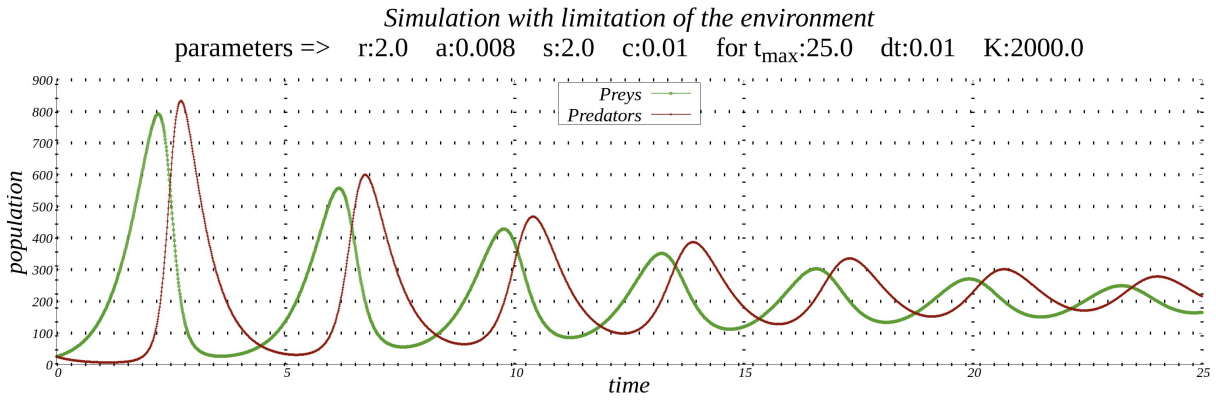
Nr. symulacji	r	a	s	c	K
1	2.0	0.008	1.0	0.01	2000.0
2	2.0	0.008	2.0	0.01	2000.0
3	2.0	0.008	3.0	0.01	2000.0

Przebiegi czasowe wykonanych symulacji znajdują się na rysunkach nr. 5-7. W porównaniu do modelu podstawowego, model z modyfikacją nie cechuje się okresowością rozwiązań.

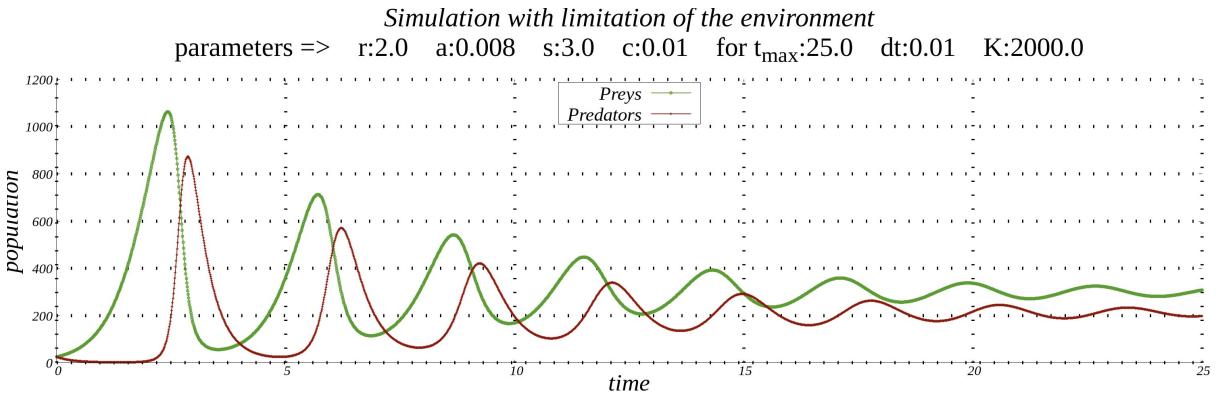


Rys. 5. Przebiegi czasowe symulacji dla s = 1.

W trakcie kolejnych symulacji przedstawionych na rysunkach nr. 5-7 zwiększeniu ulegała liczebność obu gatunków wraz ze zwiększeniem parametru *s*.

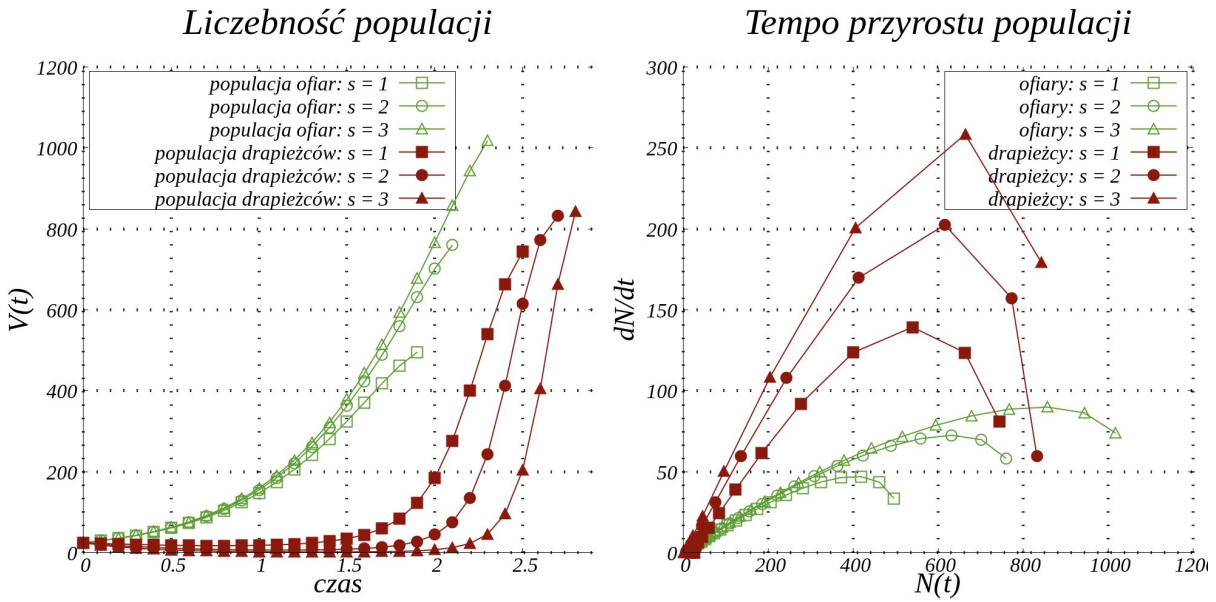


Rys. 6. Przebiegi czasowe symulacji dla s = 2.



Rys. 7. Przebiegi czasowe symulacji dla s = 3.

Rysunek nr. 8 zawiera porównanie osiąganych amplitud przebiegów oraz tempo ich wzrostu dla pierwszego okresu przyrostu populacji.



Rys. 8. Liczebność obu populacji oraz tempo ich przyrostu w trakcie trzech symulacji.

Na podstawie zestawienia z rysunku nr. 8 można wywnioskować, że:

- Wraz z zwiększeniem parametru śmiertelności drapieżców:
  - zwiększeniu ulega tempo przyrostu populacji ofiar,
  - zwiększeniu ulega tempo przyrostu populacji drapieżców,
  - zwiększeniu ulega maksymalna lokalna liczebność populacji ofiar,
  - zwiększeniu ulega maksymalna lokalna liczebność populacji drapieżników.

Tabela nr. 4 zawiera zestawienie maksymalnych liczebności populacji uzyskanych w pierwszym okresie przyrostu populacji.

Tabela 4. Zestawienie maksymalnych liczebności obu populacji.

Nr. symulacji	s	Maksymalna liczebność populacji ofiar	Maksymalna liczebność populacji drapieżców
1	**1.0**	**496**	**744**
2	**2.0**	**760**	**833**
3	**3.0**	**1018**	**842**

6. Wnioski i podsumowanie

Na podstawie przeprowadzonych symulacji w punktach 5 i 6 można stwierdzić, że wyniki zgadzają się z poczynionymi założeniami oprócz:

- liczebność populacji drapieżników nie ulegała zmniejszeniu wraz z zwiększeniem parametru s,
- tempo odradzania się populacji drapieżników nie zmniejszała się wraz z zwiększeniem parametru s

Współczynnik s wywiera znaczący wpływ na dynamikę modelu Lotki-Volterry, a nadal jest to tylko jeden z wielu czynników opisanych i pominiętych mających wpływ na modelowaną zależność drapieżca-ofiara. Modyfikacje modelu podstawowego jedynie wskazują ewentualne możliwości sprecyzowania opisanej zależności międzygatunkowej, co sugeruje niezwykłą złożoność badanych systemów w naukach biologicznych.

7. Aplikacja Small Lotka-Volterra

Aplikacja została napisana jako pojedynczy skrypt dzięki środowisku Shoes, które jest zestawem narzędzi GUI opartym na języku programowania Ruby. Ich autorem jest Jonathan Gillette. Aplikacje Shoes działają na systemach Windows, Mac OSX oraz Linux.

Aplikacje tworzone są w pojedynczych plikach .rb, w blokach funkcjonalnych. Możliwe jest w nich swobodne umieszczanie elementów interfejsu graficznego oraz obarczanie ich pewnymi zdarzeniami. Myślą przewodnią Shoes jest prostota tworzenia aplikacji, którą umożliwia bardzo prosta składnia Rubiego oraz zręczne zaimplementowanie obsługi elementów okna.

Wadami tego środowiska mogą okazać się stopień trudności wdrożenia aplikacji w środowisku produkcyjnym, który zwiększa się wraz z ilością dodatkowego, peryferyjnego oprogramowania, które wykorzystuje aplikacja oraz natywne kreślenie wykresów, które jest bardzo świeżym i niedopracowanym elementem, jednak praktycznie niestosowanym w typowych rozwiązaniach GUI.

Ze względu na niedojrzałe narzędzie do generowania wykresów wykorzystano Gnuplot, multiplatformowe narzędzie graficzne oparte na wierszu poleceń oraz klejnot Numo::Gnuplot będący wrapperem do narzędzia Gnuplot.

Aplikacja została przetestowana na jednym systemie operacyjnym - Ubuntu 18.04. Wymagana jest obecność oprogramowania Gnuplot, wszelkie zasoby Rubiego, czyli interpreter i biblioteki zawarte są wewnątrz środowiska Shoes.

7.1. Specyfikacja wewnętrzna

Aplikacja uruchamiana z poziomu środowiska Shoes korzysta z jej własnego interpretera Ruby MRI, dlatego nie trzeba przejmować się instalacją Rubiego.

7.1.1. Zasoby oprogramowania

Plik główny aplikacji stanowi small_lotka_volterra_shoesapp.rb. Aplikacja korzysta z dwóch folderów:

- image_start, zawiera przebiegi symulacji i portrety fazowe wyświetlane przy starcie aplikacji,
- image_charts, zawiera każde kolejne przebiegi symulacji i portrety fazowe wyświetlane w aplikacji.

Każdy z plików obrazów zapisywanych w folderze image_charts posiada nazwę rozpoczynającą się od:

- phase, obraz zawiera portret fazowy,
- sim, obraz zawiera przebiegi symulacji.

Pozostała część nazwy zawiera dokładne informacje o zastosowanych parametrach przy symulacji. Obecność parametru k na końcu nazwy świadczy o obrazie dotyczącym symulacji modelu zmodyfikowanego. Przy każdym symulowaniu modelu w aplikacji generowane są 4 obrazy o przykładowych nazwach (odpowiadają one symulacji na rysunku 5 w punkcie 4, Symulacja i analiza wyników):

- phase_r:0.1_a:0.008_s:0.1_c:0.01_for_t_m_a_x:250.0_dt:0.1,
- sim_r:0.1_a:0.008_s:0.1_c:0.01_for_t_m_a_x:250.0_dt:0.1,
- phase_r:0.1_a:0.008_s:0.1_c:0.01_for_t_m_a_x:250.0_dt:0.1_k:1000.0,
- sim_r:0.1_a:0.008_s:0.1_c:0.01_for_t_m_a_x:250.0_dt:0.1_k:1000.0.

7.1.2. Program główny

Program skonstruowany jest z bloków kodu charakterystycznych dla środowiska Shoes (Rys. 6). Aplikacja tworzona jest jako obiekt klasy *Shoes*, a parametry podane przy deklaracji odpowiadają za jej globalne cechy, w tym minimalna wysokość i szerokość okna. Wewnątrz aplikacji możliwe jest odwoływanie się do niej samej jako obiektu poprzez *@app* albo słowo kluczowe *self*.

```
@app=Shoes.app(title: "Small Lotka-Volterra App", height: 850, width: 850,
    resizable: true) do
    ##
    # (...Application...)
    ##
end
```

Rys. 6. Blok główny aplikacji.

Zmienne globalne odpowiadają za lokalizację wykresów wyświetlanych przy starcie aplikacji, zapis parametrów symulacji, szerokość każdego z pól tekstowych oraz nastawy potrzebne przy blokowaniu przycisku oraz kontroli animacji wszystkich przycisków.

```
@path_of_phase = "image_start/phase.png"
@path_of_phase_lim_env = "image_start/phase_lim_env.png"
@path_of_simul = "image_start/simul.png"
@path_of_simul_lim_env = "image_start/simul_lim_env.png"
@width_of_edit_line = 55
@Plotting_flag = false
@r, @a, @s, @c, @trange, @dt, @p_init, @v_init, @k =
1.1, 0.01, 0.1, 0.1, [0.0, 100.0], 0.1, 101.0, 125.0, 10000
@button_jumping = @button_save_parameters
```

Rys. 7. Zmienne globalne programu.

Animacje odpowiadają za poruszanie się przycisków, które oznacza gotowość do działania oraz natychmiastowego dostosowywania rozmiarów wyświetlanych wykresów do rozmiarów okna aplikacji. Na rysunku 8 znajduje się animacja przycisków.

```
animate do |i|
  @button_jumping.displace(0, (Math.sin(i) * 6).to_i)
end
```

Rys. 8. Jedna z animacji, zmiana rozmiarów obiektów wykresów.

Aplikacja porządkuje elementy umieszczając je w obiektach *flow* oraz *stack*. Flow układa elementy w nim zawarte możliwie blisko siebie, a gdy się nie zmieszczą obok, przerzucane są do nowego wiersza okna. Stack rozmieszcza obiekty jeden pod drugim. Rozwiązanie takie jest dzielone przez większość rozwiązań graficznego interfejsu użytkownika. Na rysunku 9 znajduje się *stack* z trzema elementami *flow*, w których umieszczono fragmenty panelu edycji parametrów symulacji.

```
@stack_edit_lines = stack do
  flow do # flow of editing parameters
    ###...###
    para "k = ", stroke: white
    @param_k = edit_line width: @width_of_edit_line, height: 30
    @param_k.text = "10000"
  end
  flow do
    ###...###
    para "step = ", stroke: white
    @param_dt = edit_line width: @width_of_edit_line, height: 30
    @param_dt.text = "0.1"
  end
  flow do
    para "Initial value P = ", stroke: white
    ###...###
  end
end
```

Rys. 9. Rozmieszczenie elementów przy pomocy stack i flow.

Przyciski tworzone są takimi samymi blokami *do...end* jak większość elementów Shoes. Wewnątrz bloku przycisku ujęte są czynności wykonywane po jego wciśnięciu. Na rysunku 10 znajduje się przycisk odpowiedzialny za przeprowadzenie symulacji przy pomocy klasy *Miaona*.

```
@button_calculate_simulation = button "Calculate simulation!" do
  s = Miaona.new(r: @r, a: @a, s: @s, c: @c,
                 trange: @trange, dt: @dt,
                 p_init: @p_init, v_init: @v_init, k: @k)
  @v, @p, @t, @v_lim_env, @p_lim_env = s.v, s.p, s.t, s.v_lim_env, s.p_lim_env

  @Plotting_flag = true
  @button_jumping = @button_plot_it
end
```

Rys. 10. Rozmieszczenie elementów przy pomocy stack i flow.

Aplikacja zawiera w swoim oknie przebiegi obu symulacji, natomiast portrety fazowe otwierane są w nowych oknach po kliknięciu danego wykresu. Większość elementów Shoes posiada pole *click*, w której można zawrzeć określone akcje (Rys. 11).

```
@Simulation.click {
  clipboard = @path_of_phase
  window title: "Phase Portrait" do
    @img = image clipboard
    animate do |i|
      @img.width = self.width
      @img.height = self.height
    end
  end
}
```

Rys. 11. Mechanizm otwierania nowego okna dla wykresu portretu fazowego.

Wykresy tworzone za pośrednictwem Gnuplot zapisywane są do plików JPG i następnie wyświetlane jako obiekty *image* (Rys. 11). Składnia wrappera Numo::Gnuplot nie wymusza stanowczej zmiany podstawowej obwoluty Gnuplot, tak więc w celu modyfikacji wykresów można śmiało korzystać z dokumentacji oferowanej przez twórców Gnuplot. Na rysunku 12 znajduje się fragment kodu odpowiedzialny za generowanie wykresu symulacji modelu podstawowego.

```
if @Plotting_flag
  #gnuplot for normal VT#########################################################
  gnuplot = Numo::Gnuplot.new
  #set terminal and output
  @path_of_simul = "image_charts/sim_r:#{@r}_a:#{@a}_s:#{@s}_c:#{@c}
  _for_t_m_a_x:#{@trange[1]}_dt:#{@dt}.png"
  gnuplot.set terminal: :pngcairo, size: [3000, 1000]
  gnuplot.set output: [@path_of_simul]
  #set title
  gnuplot.set title:"Simulation\nparameters =>  r:#{@r}  a:#{@a}  s:#{@s}  c:#{@c}
                     t_m_a_x:#{@trange[1]}  dt:#{@dt} ", font: "Times Italic, 50"
  #set axis
  gnuplot.set grid:true, lw: 8
  gnuplot.set border: 3
  gnuplot.set lmargin: 13
  gnuplot.set bmargin: 5
  gnuplot.set ylabel: "population", font: "Times Italic, 45", offset: [-4,0,0]
  gnuplot.set xlabel: "time", font: "Times Italic, 45"
  gnuplot.set xtics: {font: "Times Italic, 25"}
  gnuplot.set ytics: {font: "Times Italic, 25"}
  gnuplot.set xrange: @trange[0]..@trange[1]
  #set legend
  gnuplot.set key: :center_top_box
  gnuplot.set key_font: "Times Italic, 35"
  #plot data
  gnuplot.plot [@t, @v, w:"linespoints", pt: 6, lt: {rgb: '#5e9c36', lw:2}, t:"Preys"],
               [@t, @p, w:"linespoints", pt: 1, lt: {rgb: '#8b1a0e', lw:2}, t:"Predators"]
```

Rys. 12. Generowanie wykresu przy pomocy Numo::Gnuplot.

7.1.3. Solver

Klasa rozwiązująca układ równań różniczkowych korzysta z algorytmu Runge-Kutty 4-tego rzędu. Składa się ona z metod (Rys. 13):

- *f_lim_env*, obliczanie wartości funkcji zmodyfikowanego modelu,
- *f*, obliczanie wartości funkcji podstawowego modelu,
- *runge_kutty*, obliczanie kolejnych wartości modelów,
- *solveit*, rozwiązywanie układów równań na danym przedziale.

```
def f_lim_env(v, p)
  dvdt =  @r*v*(1.0-(v/@k))-@a*v*p
  dpdt =  -@s*p+@c*v*p #-@s*p+@a*@b*v*p
  return dvdt, dpdt
end

def f(v, p)
  dvdt =  @r*v-@a*v*p
  dpdt =  -@s*p+@c*v*p #-@s*p+@a*@b*v*p
  return dvdt, dpdt
end

def runge_kutty(v, p, v_lim_env, p_lim_env)
  vk1, pk1 = self.f(v, p)
  vk2, pk2 = self.f(v + vk1 * @dt/2.0, p + pk1 * @dt/2.0)
  vk3, pk3 = self.f(v + vk2 * @dt/2.0, p + pk2 * @dt/2.0)
  vk4, pk4 = self.f(v + vk3 * @dt    , p + pk3 * @dt)

  vnew = v + (vk1 + 2.0 * vk2 + 2.0 * vk3 + vk4) * @dt/6.0
  pnew = p + (pk1 + 2.0 * pk2 + 2.0 * pk3 + pk4) * @dt/6.0
  # lim_env
  vk1_lim_env, pk1_lim_env = self.f_lim_env(v_lim_env, p_lim_env)
  vk2_lim_env, pk2_lim_env = self.f_lim_env(v_lim_env + vk1_lim_env * @dt/2.0, p_lim_env + pk1_lim_env * @dt/2.0)
  vk3_lim_env, pk3_lim_env = self.f_lim_env(v_lim_env + vk2_lim_env * @dt/2.0, p_lim_env + pk2_lim_env * @dt/2.0)
  vk4_lim_env, pk4_lim_env = self.f_lim_env(v_lim_env + vk3_lim_env * @dt    , p_lim_env + pk3_lim_env * @dt)
  vnew_lim_env = v_lim_env + (vk1_lim_env + 2.0 * vk2_lim_env + 2.0 * vk3_lim_env + vk4_lim_env) * @dt/6.0
  pnew_lim_env = p_lim_env + (pk1_lim_env + 2.0 * pk2_lim_env + 2.0 * pk3_lim_env + pk4_lim_env) * @dt/6.0

  return vnew, pnew, vnew_lim_env, pnew_lim_env
end

def solveit()
  @v[0] = @V_init
  @p[0] = @P_init
  @v_lim_env[0] = @V_init
  @p_lim_env[0] = @P_init
  @t[0] = 0.0
  0.upto(@trange[1]/@dt) do |iter|
    @v[iter+1], @p[iter+1], @v_lim_env[iter+1], @p_lim_env[iter+1] =
    self.runge_kutty(@v[iter], @p[iter], @v_lim_env[iter], @p_lim_env[iter])
    @t[iter+1] = (1+iter) * @dt
  end
end
```

Rys. 13. Metody klasy solvera.

Klasa napisana jest w taki sposób, aby tworzenie jej nowej instancji wiązało się z obliczaniem symulacji dla zadanych parametrów, czyli metoda odpowiedzialna za rozwiązanie problemu wywoływana jest w konstruktorze klasy (Rys. 14).

```
def initialize(r:, a:, s:, c:, trange:, dt:, p_init:, v_init:, k:)
  @r = r.to_f#1.10   rozrodczość ofiar V
  @a = a.to_f#0.01   skuteczność drapieżnika
  @s = s.to_f#3.10    śmiertelność
  @c = c.to_f#b*a#1.15   biomasa => b * a = wzrost drapieżników p
  @trange = trange#[0.0, 30.0]
  @dt = dt.to_f#0.1
  @P_init = p_init.to_f#101.0
  @V_init = v_init.to_f#125.0
  @v = []
  @p_lim_env = []
  @v_lim_env = []
  @p = []
  @t = []
  @k = k.to_f#pojemność środowiska
  solveit
end
```

Rys. 14. Konstruktor klasy solvera.

7.2. Specyfikacja zewnętrzna

Aplikację uruchamia się z poziomu środowiska Shoes. Środowisko należy zaopatrzyć w klejnot Numo::Gnuplot oraz mieć zainstalowany Gnuplot na komputerze. Aplikacja funkcjonuje sprawnie w systemie Ubuntu 18.04 LTS i do tego systemu odnosi się specyfikacja.

7.2.1. Instalacja

Instalacja oprogramowania Gnuplot z poziomu konsoli znajduje na rysunku 15. Aby system był w stanie pobrać dane ze swych repozytoriów należy mieć stabilne połączenie z internetem. Następnie należy zainstalować środowisko Shoes w wersji 3.3.7 (walkabout) dla Linux 64 bit ze strony shoesrb.com/downloads/. Po pobraniu pliku instalacyjnego należy przejść do folderu pobierania, otworzyć terminal i wystosować polecenia ujęte na rysunku 16. Nazwę zawartą w poleceniach należy dostosować do nazwy pobranego pliku.

```
ciarambola@CiarambolaNaUbuntu:~$ sudo apt-get install gnuplot
Czytanie list pakietów... Gotowe
Budowanie drzewa zależności
Odczyt informacji o stanie... Gotowe
Sugerowane pakiety:
  gnuplot-doc
Zostaną zainstalowane następujące NOWE pakiety:
  gnuplot
0 aktualizowanych, 1 nowo instalowanych, 0 usuwanych i 29 nieaktualizowanych.
Konieczne pobranie 3 816 B archiwów.
Po tej operacji zostanie dodatkowo użyte 31,7 kB miejsca na dysku.
Pobieranie:1 http://pl.archive.ubuntu.com/ubuntu bionic/universe amd64 gnuplot all
 5.2.2+dfsg1-2ubuntu1 [3 816 B]
Pobrano 3 816 B w 0s (14,3 kB/s)
Wybieranie wcześniej niewybranego pakietu gnuplot.
(Odczytywanie bazy danych ... 268873 pliki i katalogi obecnie zainstalowane.)
Przygotowywanie do rozpakowania pakietu .../gnuplot_5.2.2+dfsg1-2ubuntu1_all.deb .
..
Rozpakowywanie pakietu gnuplot (5.2.2+dfsg1-2ubuntu1) ...
Konfigurowanie pakietu gnuplot (5.2.2+dfsg1-2ubuntu1) ...
```

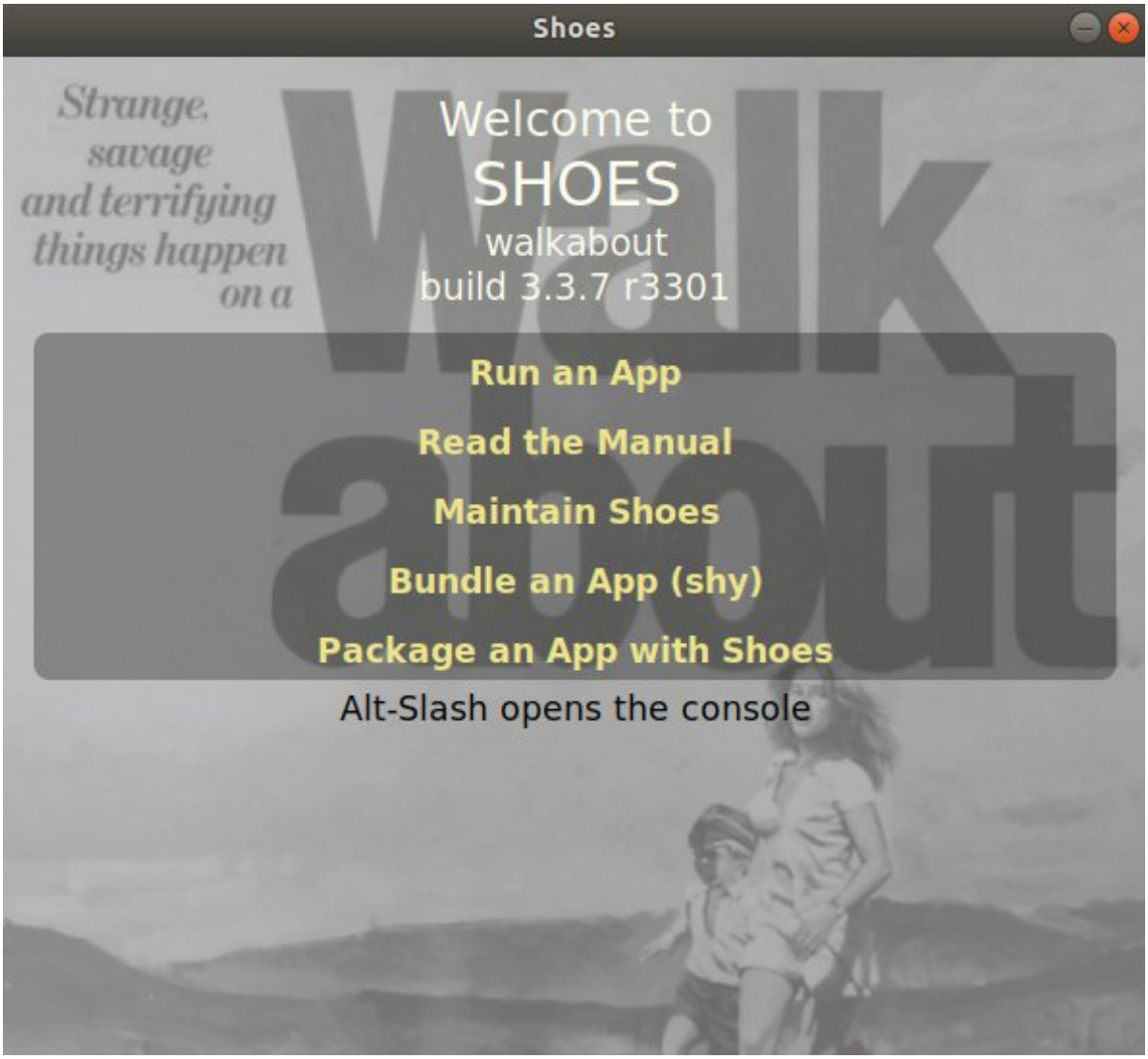
Rys. 15. Instalacja oprogramowania Gnuplot.

```
ciarambola@CiarambolaNaUbuntu:~/Pobrane$ chmod +x shoes-3.3.7-gtk3-x86_64.install
ciarambola@CiarambolaNaUbuntu:~/Pobrane$ ./shoes-3.3.7-gtk3-x86_64.install
Verifying archive integrity... All good.
Uncompressing shoes  100%
Shoes has been copied to /home/ciarambola/.shoes/walkabout. and menus created
If you don't see Shoes in the menu, logout and login
```

Rys. 16. Instalacja środowiska Shoes.

7.2.2. Instrukcja obsługi

Należy odnaleźć program Shoes w wyszukiwarce paska Ubuntu oraz uruchomić środowisko. Pierwszym krokiem w celu uruchomienia aplikacji po włączeniu środowiska jest doinstalowanie klejnotu Numo::Gnuplot udając się na panel konfiguracyjny Shoes, Aby włączyć panel konfiguracyjny należy wybrać opcję *Maintain Shoes* (Rys. 17). Po wybraniu opcji *Maintain Shoes* należy rozwinąć zakładkę *Gems* oraz wybrać opcję *Manage*. Gdy pojawi się okno z pustym polem tekstowym należy wprowadzić nazwę *numo-gnuplot* oraz zatwierdzić przyciskiem *Search Remote*. Następnie wystarczy wybrać opcję install, zatwierdzić ewentualne dodatkowe okna i zamknąć główne okno aplikacji Shoes. Procedura wymaga stabilnego połączenia z internetem.



Rys. 17. Okno główne aplikacji Shoes.

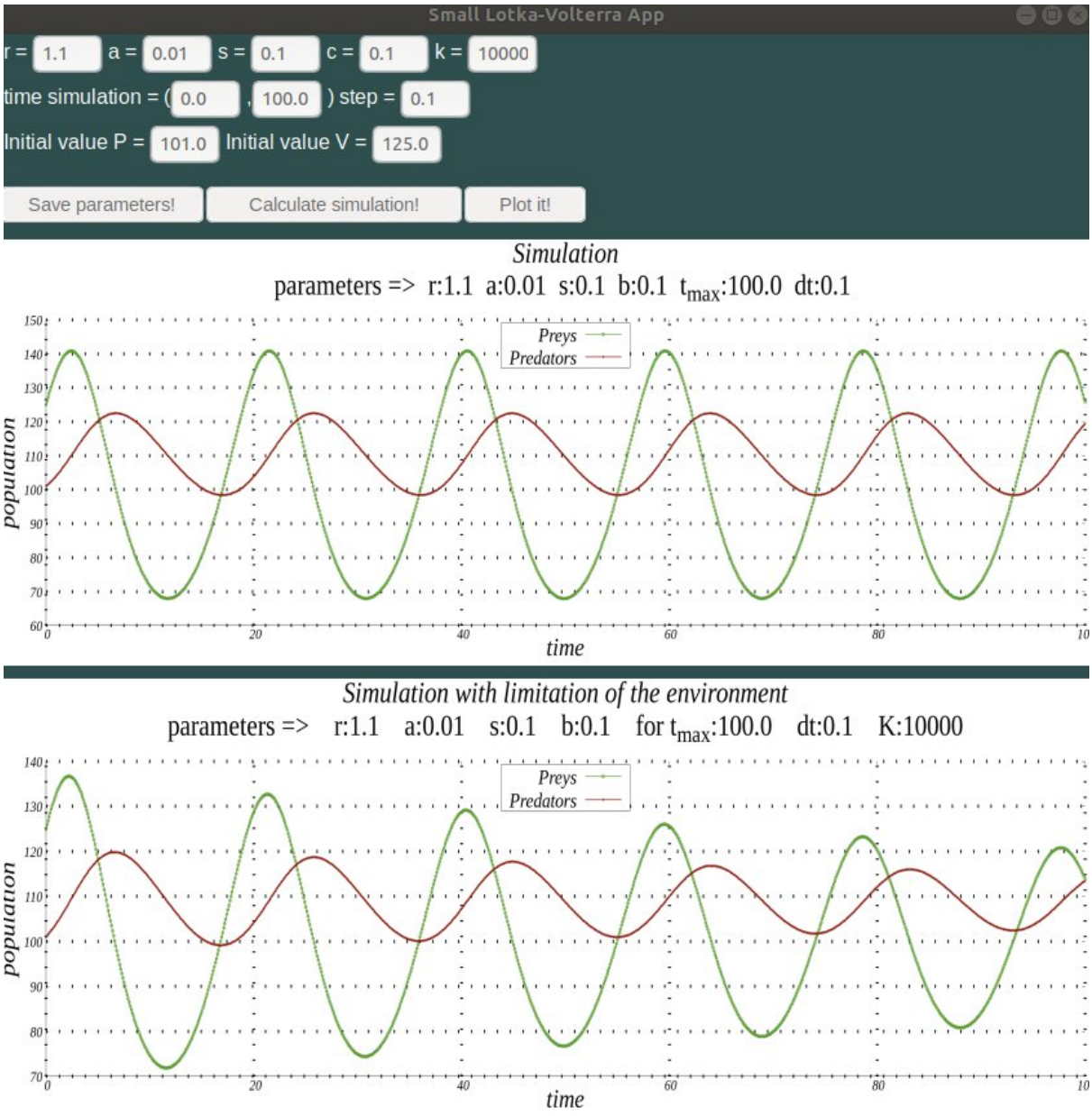
Aby uruchomić aplikację Small Lotka-Volterra należy wybrać opcję *Run an App* (Rys. 17) oraz wybrać rubinowy skrypt wspomniany w punkcie 5.1.1 - Zasoby oprogramowania oraz zatwierdzić wybór. Aplikacja zostanie otwarta w nowym oknie (Rys. 18).

Interfejs składa się z trzech elementów ułożonych w osi pionowej, idąc od góry są to:

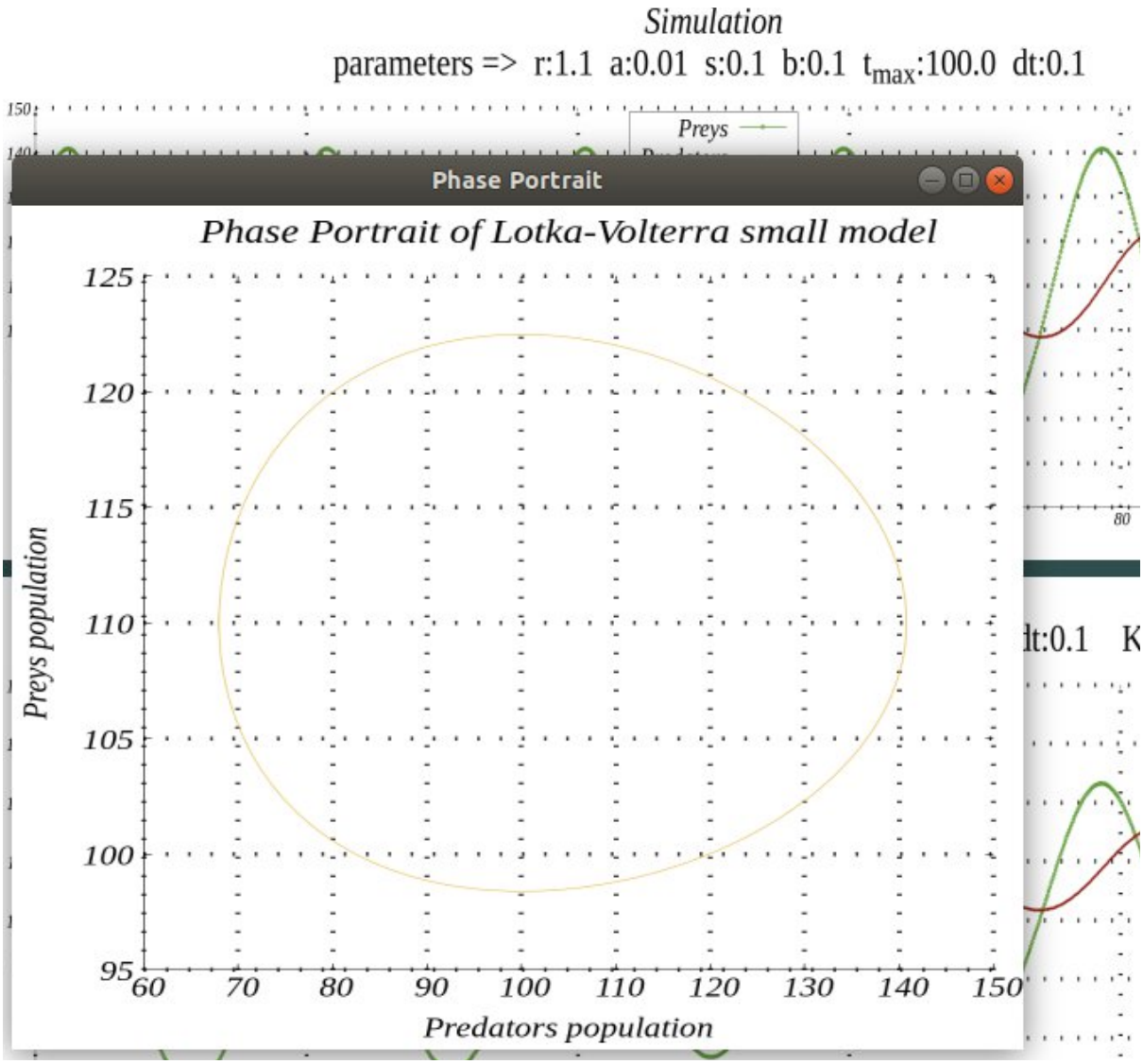
- pola tekstowe do wprowadzania parametrów symulacji, należy wprowadzać jedynie wartości liczbowe, a jako separator dziesiętny stosować kropkę,
- przyciski kontrolne aplikacji, po wprowadzeniu parametrów należy wciskać je zaczynając od lewej strony, czyli od przycisku *Save parameters!*, animacja w postaci podskakiwania kolejnego przycisku oznacza gotowość do działania, taki przycisk można wtedy wcisnąć jako następny, dostępne są przyciski:

- Save parameters!, wciśnięcie skutkuje zapisaniem parametrów do zmiennych globalnych programu,
- Calculate simulation!, wciśnięcie skutkuje zastosowaniem solvera i obliczeniem danych symulacji,
- Plot it!, wciśnięcie skutkuje wygenerowaniem wykresów, zapisem ich do folderu w sposób wspomniany w punkcie 5.1.1, Zasoby oprogramowania oraz wyświetleniem w oknie aplikacji.

❖ Przebiegi czasowe symulacji kolejno modelu podstawowego i zmodyfikowanego, kliknięcie danego przebiegu powoduje otwarcie nowego okna z portretem fazowym (Rys. 19).



Rys. 18. Okno główne aplikacji Small Lotka-Volterra.



Rys. 19. Okno z portretem fazowym modelu podstawowego.

Aplikację można zamknąć w każdej chwili. Wyniki symulacji za każdym razem są zapisywane do pliku i można do nich wrócić poza aplikacją.

Bibliografia

1. Urszula Foryś, Jan Poleszczuk, Modelowanie matematyczne w biologii i medycynie, Uniwersytet Warszawski, 2011.
2. Radosław Grzymkowski, Matematyka dla studentów wyższych uczelni technicznych, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 2000.
3. Leszek Marcinkowski, Numeryczne rozwiązywanie równań różniczkowych, Uniwersytet Warszawski, 2011.
4. http://shoesrb.com
5. http://gnuplot.sourceforge.net/demo