Міністерство освіти та науки України
Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

Звіт до лабораторної роботи №2 на тему:
"Модель Лотки—Вольтерра.
Внутрішньо-видова конкуренція'

Виконав студент групи ОМ-3
Скибицький Нікіта

Київ, 2018

Зміст

1 Модель Лотки—Вольтерра

1.1 Теоретичні відомості

Пригадаємо, що модель Лотки—Вольтерра описується наступною системою диференціальних рівнянь:

$$\begin{cases} \dot{x} = x \cdot (a - b \cdot y), \\ \dot{y} = y \cdot (d \cdot x - c), \end{cases} \tag{1}$$

де a — коефіцієнт розмноження жертв за відсутності хижаків, c — коефіцієнт природної загибелі хижаків, b — інтенсивність зменшення жертв при взаємодії двох популяці, d — інтенсивність нарощування біомаси хижаків при цьому.

Поклавши $\dot{x} = \dot{y} = 0$ знаходимо дві стаціонарні точки:

- тривіальне сідло $(x, y) = (0, 0)$;
- нетривіальний центр $(x, y) = \left(\frac{c}{d}, \frac{a}{b}\right)$.

Також відомо, що загальний інтеграл системи має вигляд

$$V = d \cdot x - c \cdot \ln x + b \cdot y - a \cdot \ln y, \tag{2}$$

де V — певна константа, яка визначається початковими умовами.

1.2 Чисельне моделювання

Було використано мову програмування `Python` і модуль `matplotlib.pyplot`.

1.2.1 Код для фазового портрету

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt

a, b, c, d = 10, 25, 20, 30


def dxdt(x: float, y: float) -> float:
        return x * (a - b * y)


def dydt(x: float, y: float) -> float:
        return y * (d * x - c)


delta = .1

s = 50

x_s, y_s = c / d, a / b
```

```python
for x_l, x_r, y_l, y_r in [(0.01, 1, 0.01, 1),
        (x_s - delta, x_s + delta, y_s - delta, y_s + delta)]:
        x_space, y_space = np.meshgrid(np.arange(x_l, x_r, (x_r - x_l) / s),
                np.arange(y_l, y_r, (y_r - y_l) / s))

        u = np.array([[dxdt(x_space[i, j], y_space[i, j]) \
                for j in range(x_space.shape[1])] for i in range(x_space.shape[0])])

        v = np.array([[dydt(x_space[i, j], y_space[i, j]) \
                for j in range(x_space.shape[1])] for i in range(x_space.shape[0])])

        plt.figure(figsize=(20,20))

        plt.title(f'Phase portrait on $[{x_l:.2f}, {x_r:.2f}] '
                f'\\times [{y_l:.2f}, {y_r:.2f}]$', fontsize=20)

        plt.xlabel('$x$', fontsize=16)
        plt.ylabel('$y$', fontsize=16)

        plt.quiver(x_space, y_space, u, v, np.hypot(u, v))

        plt.scatter(x_s, y_s, s=100, c='k', label=f'Stationary point:\n'
                f'$x = {x_s:.2f}, y = {y_s:.2f}$.')

        plt.legend()

        plt.savefig(f'../tex/phase_{x_l:.2f}_{x_r:.2f}_{y_l:.2f}_{y_r:.2f}_{s}.png')
```

1.2.2 Фазовий портрет

Рис. 1: На квадраті $[0, 1] \times [0, 1]$

phase_square.png

Рис. 2: В околі стаціонарної точки

phase_focus.png

Якщо погано видно, то можна розтягнути на весь екран, вихідні зображення гарної якості.

Як бачимо, отримані результати відповідають теоретичним очікуванням.

1.2.3 Код для 2D і 3D графіків розв'язків

Лістинг коду програми для побудови 2D і 3D графіків розв'язків:

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from typing import Tuple

a, b, c, d = 10, 25, 20, 30


def dxdt(x: float, y: float) -> float:
        return x * (a - b * y)


def dydt(x: float, y: float) -> float:
        return y * (d * x - c)


def f(x: float, y: float) -> Tuple[float, float]:
        return dxdt(x, y), dydt(x, y)
```

```python
x_s, y_s = c / d, a / b

h = 0.01

t = np.arange(0, 3 + h, h)


def k1(x: float, y: float) -> Tuple[float, float]:
        return f(x, y)


def k2(x: float, y: float) -> Tuple[float, float]:
        k1x, k1y = k1(x, y)
        return f(x + (h / 2) * k1x, y + (h / 2) * k1y)


def k3(x: float, y: float) -> Tuple[float, float]:
        k2x, k2y = k2(x, y)
        return f(x + (h / 2) * k2x, y + (h / 2) * k2y)


def k4(x: float, y: float) -> Tuple[float, float]:
        k3x, k3y = k3(x, y)
        return f(x + h * k3x, y + h * k3y)


def rk(x: float, y: float) -> Tuple[float, float]:
        k1x, k1y = k1(x, y)
        k2x, k2y = k2(x, y)
        k3x, k3y = k3(x, y)
        k4x, k4y = k4(x, y)
        return x + (h / 6) * (k1x + 2 * k2x + 2 * k3x + k4x), \
                y + (h / 6) * (k1y + 2 * k2y + 2 * k3y + k4y)


def plot_2d(x0: float, y0: float) -> None:
        x, y = [x0], [y0]

        for _ in range(len(t) - 1):
                _x, _y = rk(x[-1], y[-1])
                x.append(_x)
                y.append(_y)

        plt.figure(figsize=(20,10))

        plt.plot(t, x, 'b-', label=f'$x(t), x(0) = {x0:.2f}$')
        plt.plot(t, y, 'r-', label=f'$y(t), y(0) = {y0:.2f}$')

        plt.title(f'Solution plots with Runge-Kutta on $[{t[0]}, {t[-1]}]$\n'
```

```
                f'$x(0) = {x0:.2f}, y(0) = {y0:.2f}$', fontsize=20)

        plt.xlabel('$t$', fontsize=16)
        plt.ylabel('$x, y$', fontsize=16)

        plt.ylim(0, 1.25)

        plt.legend()

        plt.savefig(f'../tex/plot_2d_{x0:.2f}_{y0:.2f}.png')


def plot_3d(x0: float, y0: float) -> None:
        x, y = [x0], [y0]

        for _ in range(len(t) - 1):
                _x, _y = rk(x[-1], y[-1])
                x.append(_x)
                y.append(_y)

        fig = plt.figure(figsize=(10,10))

        ax = fig.gca(projection='3d')

        ax.plot(x, y, t, 'g-', label=f'$x(0) = {x0:.2f}, y(0) = {y0:.2f}$')

        ax.plot([x_s for _ in t], [y_s for _ in t], t, 'k--',
                label=f'$x(0) = {x_s:.2f}, y(0) = {y_s:.2f}$')

        ax.set_xlim3d(0, 1.25)
        ax.set_ylim3d(0, 1.25)

        plt.title(f'Solution plot with Runge-Kutta on $[{t[0]}, {t[-1]}]$\n'
                f'$x(0) = {x0:.2f}, y(0) = {y0:.2f}$', fontsize=20)

        ax.set_xlabel('$x(t)$', fontsize=16)
        ax.set_ylabel('$y(t)$', fontsize=16)
        ax.set_zlabel('$t$', fontsize=16)

        plt.savefig(f'../tex/plot_3d_{x0:.2f}_{y0:.2f}.png')


for x, y in [(.8, .2), (.5, .5), (.4, .6)]:
        plot_2d(x, y)
        plot_3d(x, y)
```

1.2.4 Розв'язок у 2D

Рис. 3: $x(0) < y(0)$

plot_2d_4_6.png

Рис. 4: $x(0) > y(0)$

plot_2d_8_2.png

Як бачимо, отримані результати відповідають теоретичним очікуванням.

1.2.5 Розв'язок у 3D

Рис. 5: $x(0) < y(0)$

plot_3d_4_6.png

Рис. 6: $x(0) > y(0)$

plot_3d_8_2.png

Як бачимо, отримані результати відповідають теоретичним очікуванням.

2 Внутрішньо-видова конкуренція

2.1 Теоретичні відомості

Пригадаємо, що модель Лотки—Вольтерра яка враховує внутрішньо-видову конкуренцію описується наступною системою диференціальних рівнянь:

$$\begin{cases} \dot{x} = x \cdot (a - b \cdot y - e \cdot x), \\ \dot{y} = y \cdot (d \cdot x - c), \end{cases} \tag{3}$$

де a — коефіцієнт розмноження жертв за відсутності хижаків, c — коефіцієнт природної загибелі хижаків, b — інтенсивність зменшення жертв при взаємодії двох популяці, d — інтенсивність нарощування біомаси хижаків при цьому, і, нарешті, e — коефіцієнт внутрішньо-видової взаємодії серед жертв.

Поклавши $\dot{x} = \dot{y} = 0$ знаходимо три стаціонарні точки:

- тривіальне сідло $(x, y) = (0, 0)$;
- напів-тривіальний стік (eng. *sink*) $(x, y) = \left(\frac{a}{e}, 0\right)$;
- нетривіальний спіральний стік $(x, y) = \left(\frac{c}{d}, \frac{a \cdot d - c \cdot e}{b \cdot d}\right)$.

2.2 Чисельне моделювання

Було використано мову програмування `Python` і модуль `matplotlib.pyplot`.

2.2.1 Код для фазового портрету

Ми не наводимо лістинг коду програми для побудови фазового портрету, оскільки у коді нічого окрім визначення функцій-похідних, визначення стаціонарної точки, і назв файлів у які зберігаються графіки не змінилося.

2.2.2 Фазовий портрет

Рис. 7: На квадраті $[0,1] \times [0,1]$

`phase_square_2.png`

Рис. 8: В околі стаціонарної точки

`phase_focus_2.png`

Якщо погано видно, то можна розтягнути на весь екран, вихідні зображення гарної якості.

Як бачимо, отримані результати відповідають теоретичним очікуванням.

2.2.3 Код для 2D і 3D графіків розв'язків

Ми не наводимо лістинг коду програми для побудови 2D і 3D графіків, оскільки у коді нічого окрім визначення функцій-похідних, визначення стаціонарної точки, і назв файлів у які зберігаються графіки не змінилося.

Зауважимо, що для чисельного інтегрування системи диференціальних рівнянь було використано класичний метод Рунге-Кутти четвертого порядку, адже загальний інтеграл цієї системи невідомий (принаймні мені).

2.2.4 Розв'язок у 2D

Рис. 9: $x(0) < y(0)$

plot_2d_1_9_2.png

Рис. 10: $x(0) > y(0)$

plot_2d_9_1_2.png

Як бачимо, отримані результати відповідають теоретичним очікуванням.

2.2.5 Розв'язок у 3D

Рис. 11: $x(0) < y(0)$

plot_3d_1_9_2.png

Рис. 12: $x(0) > y(0)$

plot_3d_9_1_2.png

Як бачимо, отримані результати відповідають теоретичним очікуванням.