Лабораторная работа 2 по курсу «Нелинейная динамика и её приложения». Отчёт.

Владислав Соврасов
381503м4

1 Построение бифуркационной диаграммы

Рассматривается система авторепрессора с задержкой:

$$\dot{x} = \frac{\alpha}{1 + x^N(t-\tau)} - x$$

Для этой системы необходимо построить бифуркационную диаграмму на плоскости (α, τ). Количество и положение состояний равновесия авторепрессора с задержкой не отличаются от системы без задержки. Комбинация параметров влияет на устойчивость единственного состояния равновесия.

Бифуркационное условие определяется уравнениями:

$$\begin{cases} x_0^{N+1} + x_0 - \alpha = 0 \\ \omega^2 = N^2(1 - \frac{x_0}{\alpha})^2 - 1 \\ cos(\omega\tau) = -\frac{1}{N(1-\frac{x_0}{\alpha})} \end{cases}$$

Задав значение α и решив первое уравнение, можно найти значение ω, подставив во второе уравнение x_0. Затем τ находится из третьего уравнения по явной формуле $\tau = \frac{1}{\omega} arccos(-\frac{1}{N(1-\frac{x_0}{\alpha})})$.

На рис. **??** представленная найденная описанным способом бифуркационная кривая при $N = 2, 4, 6$. Если задать точку (α, τ) выше кривой, то система будет переходить в режим автоколебаний. Если точка ниже кривой, то авторепрессор с задержкой имеет одно устойчивое состояние равновесия.

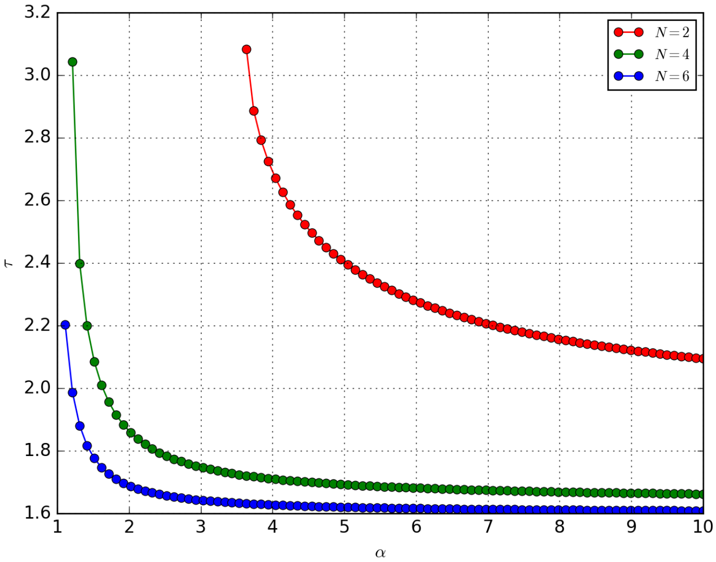

Рис. 1: Бифуркационная диаграмма системы при различных значениях N

2 Исходный код

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
import math
from lab1 import newtonMethod

def main():

    f = lambda x, n, alpha: x**(n + 1) + x - alpha
    f_x = lambda x, n, alpha: (n + 1)*x**n + 1
    alphaGrid = np.linspace(10e-4, 10., 100)
    colors = ['r', 'g', 'b', 'c']

    plt.xlabel('$\\alpha$')
    plt.ylabel('$\\tau$')

    for i in range(1, 4):
        roots = [newtonMethod(alpha, lambda x: f(x, i*2, alpha), \
            lambda x: f_x(x, i*2., alpha))[0] for alpha in alphaGrid]

        wValues = []
        for j, x0 in enumerate(roots):
```

```
            rootArg = (i*2.*(1. - x0 / alphaGrid[j]))**2 - 1.
            if rootArg > 0.:
                wValues.append(math.sqrt(rootArg))
            else:
                wValues.append(np.nan)

        tValues = []
        for j, w in enumerate(wValues):
            acosArg = 1./(i*2.)/(roots[j] / alphaGrid[j] - 1.) / w \
                if w is not np.nan else np.inf
            if acosArg < 1. and acosArg > -1.:
                tValues.append(math.acos(acosArg))
            else:
                tValues.append(np.nan)

        plt.plot(alphaGrid, tValues, colors[i-1]+'-o', label='$N='+str(i*2)+'$')

    plt.grid()
    plt.legend(loc = 'best', fontsize = 10)
    plt.savefig('../pictures/lab2_diagram.png', format = 'png', dpi = 200)

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