Лабораторная работа 1 по курсу «Нелинейная динамика и её приложения». Отчёт.

Владислав Соврасов
381503м4

1 Сравнение скорости сходимости методов поиска корня полинома

Для поиска корня полинома $f(x) = x^3 + x - 1$ были использованы методы дихотомии и Ньютона. Первый метод обладает линейной сходимостью, а второй — квадратичной (т.к. в данном случае производная $f(x)$ нигде не обращается в 0). В качестве начального приближения для метода Ньютона была выбрана точка $x_0 = 1$. В методе дихотомии начальный отрезок был взят $[0, 5; 1]$. Точность обоих методов: 10^{-3}.

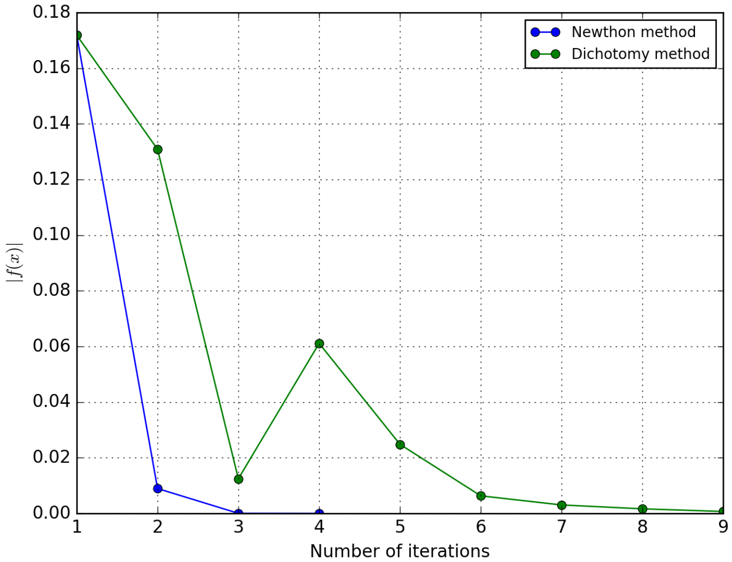

Рис. 1: Зависимость абсолютного значения полинома от номера итерации метода

В результате запуска методов было получено приближённое значение корня $\widetilde{x} = 0.6816$ (значение округлено). Из рис. 1 видно, что метод Нтьютона сходится за 4 итерации, в то время как методу дихотомии необходимо 9 итераций для достижения той же точности.

2 Построение зависимости корня полинома от параметра

Необходимо построить зависимость $x^*(\alpha)$ координаты единственного корня полинома $f(x) = x^{N+1} + x + \alpha$ от α при $N = 2, 4, 6$ на отрезке $[0; 10]$. Из качественного анализа положения корня следует, что при малых α зависимость схожа с линейной, а при больших — близка к функции $\sqrt[N+1]{\alpha}$.

Глядя на рис. 2, можно убедиться в справедливости качественных оценок.

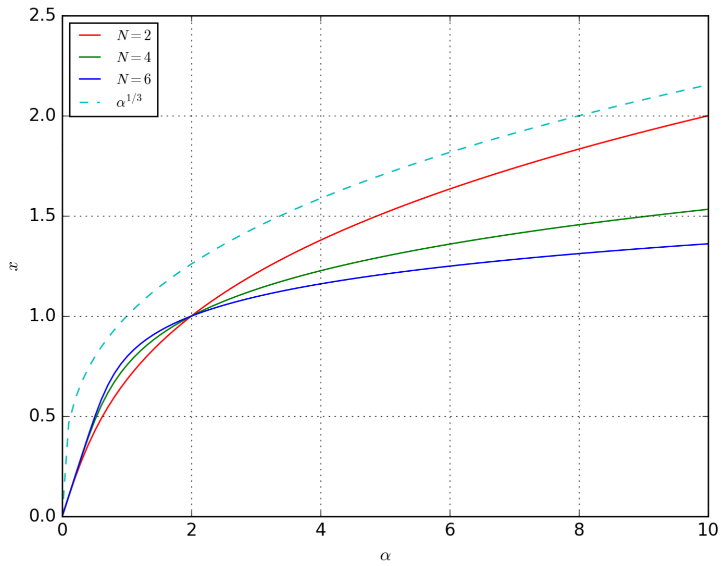

Рис. 2: Зависимость корня полинома $f(x)$ от параметра при различных значениях степени

3 Поиск бифуркационного значения параметра системы

При рассмотрении системы

$$\begin{cases} \dot{x} = \frac{\alpha}{1+y^2} \\ \dot{y} = \frac{\alpha}{1+x^2} \end{cases}$$

было выяснено, что критерием бифуркации является изменение количества корней у полинома $g(x) = x^5 - \alpha x^4 + 2x^3 - 2\alpha x^2 + (\alpha^2 + 1)x - \alpha$. Необходимо обнаружить бифуркацию численно и построить зависимость корней $g(x)$ от α.

Зависимость была построена для $\alpha \in [0; 10]$. При $\alpha = 10$ с помощью запусков метода Ньютона из различных псевдослучайно сгенерированных точек были найдены все корни $g(x)$. Затем, при движении по сетке в направлении $\alpha = 0$, значения корней в предыдущем узле сетки использовались как начальные приближения в методе Ньютона для поиска корней в текущем узле.

Как видно из рис. 3, в окрестности точки $\alpha = 2$ происходит бифуркация и одно состояние равновесия системы превращается в три, которые удаляются друг от друга с ростом α.

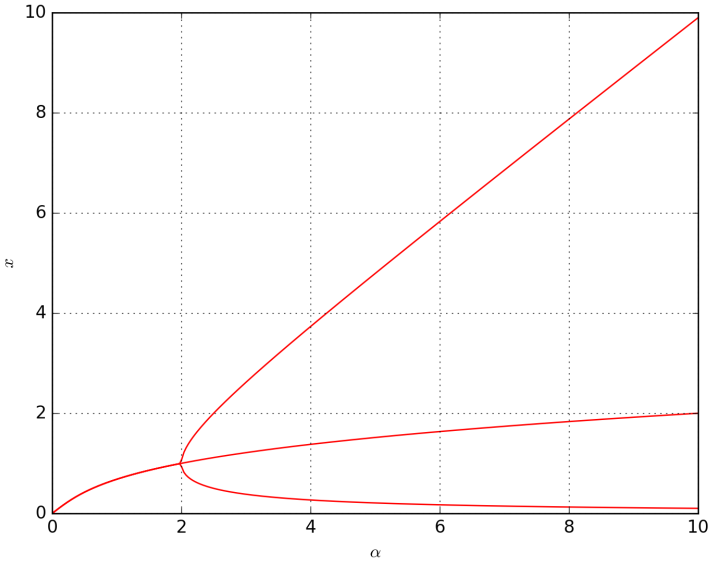

Рис. 3: Зависимость корней полинома $g(x)$ от параметра

4 Исходный код

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

import numpy as np
import matplotlib.pyplot as plt

def newtonMethod(x_0, f, f_x, eps = 0.001):
    x_next = x_0 - f(x_0) / f_x(x_0)
    values_trace = [f(x_next)]

    while abs(x_0 - x_next) > eps:
        x_0, x_next = x_next, x_0
        x_next = x_0 - f(x_0) / f_x(x_0)
        values_trace.append(f(x_next))

    return x_next, values_trace

def dichotomyMethod(x_l, x_r, f, eps = 0.001):
    values_trace = []

    while x_r - x_l > eps:
        x_mid = (x_l + x_r) / 2.
        if f(x_mid)*f(x_l) < 0:
            x_r = x_mid
```

```
        elif f(x_mid)*f(x_r) <= 0:
            x_l = x_mid
        else:
            return None, None
        values_trace.append(f(x_mid))

    return (x_l + x_l) / 2., values_trace

def deleteSimilar(array, value, eps = 0.001):
    filteredArray = []

    for x in array:
        if abs(x - value) > eps:
            filteredArray.append(x)

    return filteredArray

def main():

    f = lambda x: x**3 + x - 1
    f_x = lambda x: 3*x**2 + 1

    #|f(x)|, n=2, alpha=1
    x_opt, iterationsNewthon = newtonMethod(1., f, f_x)
    print('Root found by Newthon method = {}'.format(x_opt))
    x_opt, iterationsDich = dichotomyMethod(0.5, 1., f)
    print('Root found by dichotomy method = {}'.format(x_opt))

    plt.xlabel('Number of iterations')
    plt.ylabel('$|f(x)|$')
    plt.plot(range(1, len(iterationsNewthon) + 1), np.abs(iterationsNewthon), \
        'b-o', label='Newthon method')
    plt.plot(range(1, len(iterationsDich) + 1), np.abs(iterationsDich), 'g-o', \
        label='Dichotomy method')
    plt.grid()
    plt.legend(loc = 'best', fontsize = 10)
    plt.savefig('../pictures/lab1_convergence.png', format = 'png', dpi = 200)

    #x(alpha), n=2,4,6
    f = lambda x, n, alpha: x**(n + 1) + x - alpha
    f_x = lambda x, n, alpha: (n + 1)*x**n + 1
    alphaGrid = np.linspace(0., 10., 100)
    colors = ['r', 'g', 'b', 'c']

    plt.clf()
    plt.xlabel('$\\alpha$')
    plt.ylabel('$x$')

    for i in range(1, 4):
```

```python
        roots = [newtonMethod(alpha, lambda x: f(x, i*2, alpha), \
            lambda x: f_x(x, i*2, alpha))[0] for alpha in alphaGrid]
        plt.plot(alphaGrid, roots, colors[i-1] + '-', label='$N='+str(i*2)+'$')

    plt.plot(alphaGrid, np.power(alphaGrid, [1./3]*len(alphaGrid)), \
        colors[3] + '--', label='$\\alpha^{1/3}$')

    plt.grid()
    plt.legend(loc = 'best', fontsize = 10)
    plt.savefig('../pictures/lab1_roots.png', format = 'png', dpi = 200)

    plt.clf()
    plt.xlabel('$\\alpha$')
    plt.ylabel('$x$')

    f = lambda x, alpha: x**5 - alpha*x**4 + 2*x**3 - 2*alpha*x**2 \
        + (alpha**2 + 1)*x - alpha
    f_x = lambda x, alpha: 5*x**4 - 4*alpha*x**3 + 6*x**2 - \
        4*alpha*x + alpha**2 + 1.

    mid = np.power(alphaGrid[-1], 1./3.)
    initialPoints = [np.random.uniform(mid - 100., mid + 100.) for _ in range(100)]
    roots = [newtonMethod(x0, lambda x: f(x, alphaGrid[-1]), \
        lambda x: f_x(x, alphaGrid[-1]))[0] for x0 in initialPoints]

    filteredRoots = []
    while len(roots) != 0:
        filteredRoots.append(roots[0])
        roots = deleteSimilar(roots, filteredRoots[-1])

    alphaGrid = np.linspace(0., 10., 300)
    filteredRoots = sorted(filteredRoots)
    xValues = np.array([[0.]*len(alphaGrid)]*len(filteredRoots))

    for i in reversed(range(len(alphaGrid))):
        newRoots = []
        for j in range(len(filteredRoots)):
            newRoots.append(newtonMethod(filteredRoots[j], \
                lambda x: f(x, alphaGrid[i]), lambda x: f_x(x, alphaGrid[i]))[0])
            xValues[j][i] = newRoots[j]
        filteredRoots = newRoots

    for i, values in enumerate(xValues):
        plt.plot(alphaGrid, values, colors[0] + '-')

    plt.grid()
    plt.savefig('../pictures/lab1_bifurcation.png', format = 'png', dpi = 200)

if __name__ == '__main__':
```

```
main()
```