МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Курсова робота
з курсу "Чисельні методи"
на тему:
"Мінімаксна апроксимація функцій многочленами"

Виконав:
ст. гр. ПМ-41
Левантович Богдан
Перевірив:
доц. каф. ПМ
Пізюр Я.В.

Львів 2017

Зміст

Вступ

Багатьом із тих, хто стикається з науковими та інженерними розрахунками часто доводиться оперувати наборами значень, отриманих експериментальним шляхом чи методом випадкової вибірки. Як правило, на підставі цих наборів потрібно побудувати функцію, зі значеннями якої могли б з високою точністю збігатися інші отримувані значення. Така задача називається апроксимацією кривої. Інтерполяцією називають такий різновид апроксимації, при якій крива побудованої функції проходить точно через наявні точки даних.

1 Способи задання функцій. Норма похибки

Наближувана функція $f(x)$ у практичних обчисленнях найчастіше задається або в аналітичному вигляді або у вигляді дискретних значень (табличне задання функції). Таблично задану функцію можна представити у вигляді

$$y_k = f_k = f(x_k), k = \overline{1, N},$$

де значення аргумента $X = \{x_k\}_1^N \in [a, b]$. Далі припускатимемо, що аргументи упорядковані за зростанням:

$$a \leq x_1 < x_2 < \ldots < x_N \leq b.$$

При використанні наближених методів на ЕОМ неможливо врахувати значення функції $f(x)$ у всіх точках $[a, b]$, бо кількість N чисел, що може бути представлена на ЕОМ обмежена. Тому обчислювальні методи повинні бути побудовані так, щоб розв'язок задачі на проміжку $[a, b]$ був еквівалентний її розв'язку на характерній підмножині $X_0 \subset X \subset [a, b]$, що складається з обмеженої кількості точок.

Для наближення функції $f(x)$ використовуємо простіший вираз

$$F(A, x) = F(a_0, a_1, \ldots, a_m; x), \tag{1}$$

з $m + 1$ параметром. Частинним випадком виразу (1) є многочлен степеня m

$$P_m(x) = a_0 + a_1 x + \ldots + a_m x^m. \tag{2}$$

Якість наближення функції $f(x)$ за допомогою виразу (1) на проміжку $[a, b]$ характерезується віддалю між цими функціями. Спосіб виміру цієї віддалі визначає норму похибки наближення функції $f(x)$ за допомогою виразу (1) на проміжку $[a, b]$ (або на множині X). Для більшої загальності у виразах для похибки часто використовують зважену віддаль (зважену різницю)

$$\rho(x) = \frac{f(x) - F(A, x)}{w(x)}, \tag{3}$$

де вага $w(x) > 0$ при $x \in [a, b]$, $w_k = w(x_k), k = \overline{1, N}$.

Використання тієї чи іншої норми похибки залежить передусім від конкретних задач, що стоять при наближенні функцій. У теоретичних дослідженнях часто використовується норма похибки L_p.

$$||f-F||_{L_p} = \left(\int_a^b \left| \frac{f(x)-F(A,x)}{w(x)} \right|^p \mathrm{d}x \right)^{\frac{1}{p}} = \left(\sum_{k=1}^{N} \left| \frac{y_k - F(A,x_k)}{w_k} \right|^p \right)^{\frac{1}{p}}.$$

Найбільш вживані частинні випадки цієї норми L_1, L_2 та L_∞. Норму L_1 слід вживати там, де необхідно зменшити суму площ, що обмежуються кривими $y = f(x)$ та $y = F(A,x)$.

$$||f-F||_{L_1} = \int_a^b \frac{|f(x)-F(A,x)|}{w(x)} \mathrm{d}x = \sum_{k=1}^{N} \frac{|y_k - F(A,x_k)|}{w_k}.$$

Норму L_2 або середньоквадратичну похибку найчастіше використовують при обробці дослідних даних.

$$||f-F||_{L_2} = \left(\int_a^b \left(\frac{f(x)-F(A,x)}{w(x)} \right)^2 \mathrm{d}x \right)^{1/2} = \left(\sum_{k=1}^{N} \left(\frac{y_k - F(A,x_k)}{w_k} \right)^2 \right)^{1/2}.$$

Норму L_∞ (її часто називають чебишовською нормою або нормою C) використовують, щоб найточніше представити кожне значення наближуваної функції $f(x)$. Припускається, що ця остання відома достатньо точно:

$$||f-F||_{L_\infty} = ||f-F||_C = \max_{x\in[a,b]} \frac{|f(x)-F(A,x)|}{w(x)} = \max_{x_k \in X} \frac{|y_k - F(A,x)|}{w_k}.$$

При обчисленнях з чебишовською нормою, функцію (3) звуть функцією похибки, її графік - кривою похибки.

2 Найкраще чебишовське наближення

За теоремою Вейєрштрасса для довільних неперервних на обмеженому проміжку $[a,b]$ функцій $f(x)$ та $w(x) > 0$ і довільного $\epsilon > 0$ можна знайти такий многочлен $P_m(x)$, що

$$|\rho(x)| = \frac{|f(x) - P_m(x)|}{w(x)} < \epsilon, \quad x \in [a, b].$$

Ясно, що найменше при цьому значення степеня m многочлена $P_m(x)$ суттєво залежить від способу наближення. Серед усіх способів наближення функцій найменшу похибку а, значить, і найменше m при заданому ϵ, дає найкраще чебишовське наближення.

Вираз $F(A, x) \in F(B, x)$, для якого максимальне значення абсолютної величини зваженої похибки (3) досягає на проміжку $[a, b]$ найменшого значення

$$\min_{c \in B} \max_{x \in [a,b]} \frac{|f(x) - F(C, x)|}{w(x)} = \max_{x \in [a,b]} \frac{|f(x) - F(A, x)|}{w(x)}, \tag{4}$$

звемо найкращим чебишовським зваженим (з вагою $w(x)$) наближенням функції $f(x)$ за допомогою виразу виду $F(A, x)$ на проміжку $[a, b]$.

У цій курсові розглянуто лише найкращі чебишовські наближення. Слова "чебишовські" і "зважені" будемо часом пропускати. При $w(x) = 1$ маємо найкраще абсолютне наближення, при $w(x) = f(x)$ - найкраще відносне.

Величину (4) називатимемо мінімальним (зваженим) відхиленням і позначаємо $E(f, W) \equiv \mu_0$; $E(f, 1) \equiv E(f) \equiv \Delta_0$ - мінімальне абсолютне відхилення; $E(f, f) \equiv \delta_0$ - мінімальне відносне відхилення.

Далі розглянемо властивості найкращих наближень многочленом.

Теорема 1. *Для будь-яких неперервних на проміжку $[a, b]$ функцій $f(x)$ та $w(x) > 0$ і довільного ϵ, існує єдиний многочлен $P_m(x)$ степеня m, що має найменше відхилення $E(f, w)$.*

Теорема 2. *Нехай на проміжку $[a, b]$ задано неперервні функції $f(x)$ та $w(x) > 0$. Тоді для того, щоб деякий многочлен $P_m(x)$ степеня не вище m був многочленом найкращого чебишовського зваженого наближення функції $f(x)$ на проміжку $[a, b]$ необхідно і достатньо, щоб на цьому проміжку знайшлась принаймні одна система з $m + 2$ точок*
$T = \{t_k\}_{k=0}^{m+1}$ $a \le t_0 < t_1 < t_2 < \ldots \le t_{m+1}$, у яких зважена різниця (3) почергово набувала значень різних знаків і досягала за модулем найбільшого на $[a, b]$ значення тобто:

$$\rho(t_0) = -\rho(t_1) = \rho(t_2) = \ldots = (-1)^{m+1} \rho(t_{m+1}) = \pm E(f, W). \tag{5}$$

Система точок T із теореми 2 зветься системою точок (чебишовського альтернансу). Для побудови многочлена найкращого наближення необхідно визначити ці точки. Точно визначити їх значення можна тільки у часткових випадках.

2.1 Схема Ремеза побудови чебишовського наближення

У загальному випадку процес знаходження точок T побудовано на ітераційних методах. Найбільше практичне значення мають методи розроблені українським математиком Є.Я. Ремезом. Коротко розглянемо один з методів. Він складається з таких етапів.

1. З проміжку $[a, b]$ вибираємо початкове наближення T_0 до альтернансу
$$T : t_0^{(0)} < t_1^{(0)} < t_2^{(0)} < \ldots < t_{m+1}^{(0)}.$$
Можна, наприклад, прийняти $t_k^{(0)} = a + \frac{(b-a)k}{m+1}$.

2. Здійснюємо чебишовську інтерполяцію для множини точок $T_j = \{t_k\}_{k=0}^{m+1}, t_k^{(j)} < t_{k+1}^{(j)}, k = \overline{0, m}$, тобто визначаємо коефіцієнти многочлена $P_m^i(x)$ і величину μ_j, для яких виконуються умови $\rho(t_k^{(j)}) = (-1)^k \mu_k \quad k = \overline{0, m+1}$. Для знаходження вказаних величин розв'язуємо систему рівнянь:
$$\begin{cases} f(t_0^{(j)}) - a_0 - a_1 t_0^{(j)} - \ldots - a_m (t_0^{(j)})^m = \mu_j w(t_0^{(j)}), \\ f(t_1^{(j)}) - a_0 - a_1 t_1^{(j)} - \ldots - a_m (t_1^{(j)})^m = -\mu_j w(t_1^{(j)}), \\ \ldots\ldots\ldots\ldots\ldots\ldots \qquad \ldots\ldots\ldots\ldots\ldots\ldots \\ f(t_{m+1}^{(j)}) - a_0 - a_1 t_{m+1}^{(j)} - \ldots - a_m (t_{m+1}^{(j)})^m = (-1)^{m+1} \mu_j w(t_{m+1}^{(j)}). \end{cases} \tag{6}$$
Система є системою $m+2$ алгебраїчних рівнянь з $m+2$ невідомими: $a_0, a_1, \ldots, a_m$ та μ.

3. Перевіряємо виконання рівності

$$|\mu_j| = \max_{x\in[a,b]} |f(x) - P_m^{(j)}(x)|/w(x) \equiv \rho_j. \tag{7}$$

   Якщо рівність виконується, то у відповідності з теоремою 2 многочлен $P_m^{(j)}(x)$ і є шуканий многочлен найкращого наближення. При машинній реалізації алгоритму перевірку рівності заміняють перевіркою нерівності

$$\rho_j - |\mu_j| \leq \epsilon|\mu_j|, \tag{8}$$

   де ϵ - допустима відносна помилка у визначенні похибки наближення. Можна, наприклад, прийняти $\epsilon = 10^{-2}$ чи $\epsilon = 10^{-3}$.

4. Якщо умова 7 чи 8 не виконується, то приймаємо $j := j + 1$ і вибираємо наступне (уточнене) наближення до точок чебишовського альтернансу (наступний V-альтернанс). Далі виконання алгоритму повторюється починаючи з п.2.

   При обчисленнях на ЕОМ у цьому пункті іноді додатково перевіряються умови

$$\left|t_k^{(j-1)} - t_k^j\right| < \eta, \quad k = \overline{0, m+1},$$

   де η - допустима помилка у визначенні точок альтернансу. Якщо остання нерівність справедлива для всіх точок $k = \overline{0, m+1}$, то вважаємо, що многочлен найкращого наближення знайдено.

2.2 Алгоритм Валле-Пуссена заміни точок альтернансу

Існує кілька методів заміни точок альтернансу. Можлива заміна одної або кількох точок одночасно. Найпростішим алгоритмом є алгоритм Є.Я. Ремеза з одноточковою заміною (алгоритм Валлє-Пуссена). Опишемо цей алгоритм.

Нехай при виконанні п.3 знайдена точка $\tilde{x}$, для якої справедливо $\rho_j = |\rho(\tilde{x})|$. Можливі три випадки взаємного розміщення точок V-альтернансу та точки $\tilde{x}$:

1. $t_0^{(j)} < \tilde{x} < t_{m+1}^{(j)}$
2. $\tilde{x} < t_0^{(j)}$
3. $\tilde{x} > t_{m+1}^{(j)}$

Розглянемо спосіб заміни точок V-альтернансу у кожному випадку.

1. Знайдемо ціле число v таке, що $t_v^{(j)} < \tilde{x} < t_{v+1}^{(j)}$. Якщо $\mathrm{sign}(\rho(\tilde{x})) = \mathrm{sign}(\rho(t_{m+1}^{(j)}))$, то приймаємо $t_v^{(j+1)} := \tilde{x}$, у протилежному випадку $t_{v+1}^{(j+1)} := \tilde{x}$. Решту точок V-альтеранансу не змінюємо.

2. Якщо $\mathrm{sign}\,\rho(\tilde{x}) = \mathrm{sign}\,\rho(t_0^{(j)})$, то приймаємо $t_0^{(j+1)} := \tilde{x}$, а решту точок V-альтернансу не змінюємо. Якщо це не так, то заміняємо усі точки альтернансу за формулами:

$$t_0^{(j+1)} := \tilde{x}; \quad t_k^{(j+1)} := t_{k-1}^{(j)}, \quad k = \overline{1, m+1}.$$

   У цьому випадку із V-альтернансу виключається точка $t_{m+1}^{(j)}$

3. Якщо $\mathrm{sign}\,\rho(\tilde{x}) = \mathrm{sign}\,\rho(t_{m+1}^{(j)})$, то приймаємо $t_{m+1}^{(j)} := \tilde{x}$. і решту точок V-альтернансу не змінюємо. Якщо це не так, то замінюємо усі точки V-альтернансу за формулами:

$$t_k^{(j+1)} := t_{k+1}^{(j)}, \quad k = \overline{0, m}; \quad t_{m+1}^{(j+1)} := \tilde{x}.$$

   У цьому випадку із V-альтернансу виключається точка $t_0^{(j)}$.

Отже черговий V-альтернанс відрізняєтся від попереднього тим, що точка $\tilde{x}$, у якій досягається максимум абсолютної величини зваженої похибки, вводиться у V-альтернанс замість однієї із старих точок. Відомо, що алгоритм Валле-Пуссена для заміни точок альтернансу при знаходженні найкращого наближення попередньої функції многочленом на проміжку $[a, b]$ збігається незалежно від початкового наближення до точок альтернансу. Більш того у цьому випадку цей алгоритм збігається зі швидкістю гометричної прогресії у тому сенсі, що знайдуться такі числа A та $0 < q < 1$, що відхилення $E^{(k)}(f,W)$ многочлена $P_m^{(k)}(x)$ від функції $f(x)$ будуть задовольняти нерівності

$$E^{(k)}(f,W) - E(f,W) \leq Aq^k; \quad k = 1, 2, \ldots$$

Фактична швидкість збіжності залежить від диференціальних властивостей функції та використовуваного алгоритму заміни точок альтернансу. Відомо, що коли $f(x) \in C^{m+1}[a,b], w(x) = 1$ або $w(x) = f(x)$ і $f^{(m+1)}(x)$ не змінює знак при $x \in [a,b]$, то граничні точки проміжку $[a,b]$ є точками альтернансу. Тому у цьому випадку алгоритм Валле-Пуссена для наближення многочленами невисоких степенів $m = \overline{0,2}$ практично не програє у швидкості порівняно з іншими алгоритмами типу Є.Я. Ремеза. Зазначимо, що всі перелічені властивості найкращого чебишовського наближення неперервної при $x \in [a,b]$ функції $f(x)$ многочленом справедливі також і для наближення табличної функції. Більш того, при заміні неперервної функції її значенями в точках $x_k = a + \frac{(b-a)k}{N}$ різниця між відповідними відхиленнями при $N \to \infty$ прямує до нуля.

3 Опис програми

Мета програми: знаходження найкращого чебишовського наближення для заданої функції.
Програма написана на мові програмування *Python* з використанням таких бібліотек: *Sympy, Numpy, Plotly.*

3.1 Вхідні дані

1. Початок інтервалу.
2. Кінець інтервалу.
3. Степінь многочлена.
4. Функція для апроксимації.
5. Точність (за замовчуванням 10^{-2}).

3.2 Вихідні дані

1. Коефіцієнти многочлена.
2. Графіки похибок на кожній ітерації.
3. Графік многочлена і функції.

Висновки

У цій курсовій я розглянув найкраще чебишовське наближення многочленами. Написав програму для знаходження коефіцієнтів такого многочлена. Також в програмі реалізув побудову графіків похибок на кожній ітерації, вивід максимальної похибки та значення аргументу при якому ця похибка досягається.

Список використаної літератури

1. Демьянов В.Ф., Малоземов В.Н. Введение в минимакс. -М.: Наука, 1972. - 368 с.

2. Попов Б.А. Равномерное приближение сплайнами. -Киев: Наук. думка, 1989. - 272 с.

3. Попов Б.А., Теслер Г.С. Приближение функций для технических приложений. - Киев: Наук. думка, 1980. - 352 с.

4. Ремез Е.Я. Основы численных методов чебышовского приближения. Киев: Наук. думка, 1969, - 623 с.

5. Попов Б.О. Чисельні методи рівномірного наближення сплайнами. Конспект лекцій. -Львів: ЛДУ, 1992. - 92 с.

6. Самарский А.А., Гулин А.В. Численные методы. -М.: Наука, 1989. - 432 с.

7. https://plot.ly/ - для побудови графіків

8. http://www.sympy.org/ - для розв'язування систем

9. http://www.numpy.org/ - для наукових розрахунків

4 Додатки

4.1 Текст програми

```
import plotly
from plotly.graph_objs import Scatter, Layout
import numpy as np
import sympy

plotly.offline.init_notebook_mode()
x = sympy.Symbol('x')

sympy.init_printing()

start = 1 # start
end = 4 # end
degree = 2 # degree of polynomial

error_on_iteration = 0 # error on each iteration
precision = 1e-2 # precision

alternance = [start + (end-start) * k / float(degree + 1) for k in range(
    degree+2)]

alternance # first alternance

# f = np.e**x # function to approximate
f = sympy.log(x)

def make_eq(coefs, point):
    _f = sympy.lambdify(x, f)
    eq = _f(point)
    for i, c in enumerate(coefs):
        eq -= c*point**i
    return eq

def pol(t):
    global error_on_iteration
    e = sympy.Symbol('e')
    vars_str = ' '.join(['a' + str(i) for i in range(degree+1)])
    variables = sympy.symbols(vars_str)
    eqs = []

    for i in range(degree+2):
        eqs.append(make_eq(variables, t[i]) + e)
        e *= -1
    if (degree + 2) % 2 == 1:
        e *= -1

    solution = sympy.solve(eqs, variables + (e,))
```

```

    error_on_iteration = solution[e]
    polynom = x - x
    for i, v in enumerate(variables):
        polynom += solution[v] * x**i

    return polynom

def max_error():
    polyn = pol(alternance)
    err_fun = np.vectorize(sympy.lambdify(x, f - polyn))
    x_vals = np.linspace(start, end, (end - start) * 1000) # x values to check for maximum
    y_vals = err_fun(x_vals)

    neg_err = min(y_vals)
    pos_err = max(y_vals)

    if abs(neg_err) > pos_err:
        e_max = neg_err
    else:
        e_max = pos_err
    return e_max

def x_of_max_error():
    polyn = pol(alternance)
    err_fun = np.vectorize(sympy.lambdify(x, f - polyn))
    x_vals = np.linspace(start, end, (end - start) * 10000) # x values to check for maximum
    y_vals = err_fun(x_vals)

    absolute_y_vals = list(map(lambda x: abs(x), y_vals))
    e_max = max(absolute_y_vals)

    i = list(absolute_y_vals).index(e_max) # index of max error

    return x_vals[i]


def error():
    return np.vectorize(sympy.lambdify(x, f - pol(alternance)))

def plot_error_function(plot_max_err=False, title="Error"):
    x = sympy.Symbol('x')
    _f = np.vectorize(sympy.lambdify(x, f))
    p = np.vectorize(sympy.lambdify(x, pol(alternance)))
    x_vals = np.linspace(start, end, (end - start) * 1000)


    if plot_max_err == False:
        data = [Scatter(x=x_vals, y = _f(x_vals) - p(x_vals))]
```

```python

    else:
        y_err = max_error()
        x_err = x_of_max_error()
        data = [Scatter(x=x_vals, y = _f(x_vals) - p(x_vals),  name="Error"),
                Scatter(x=[x_err for i in range(100)], y=np.linspace(0, y_err, 100), name="Max error")]

    plotly.offline.iplot({
        "data": data,
        "layout": Layout(title=title)
    })

plot_error_function(plot_max_err=True)


def plot_approximation(plot_max_error=False):

    x = sympy.Symbol('x')
    _f = np.vectorize(sympy.lambdify(x, f))
    p = np.vectorize(sympy.lambdify(x, pol(alternance)))
    x_vals = np.linspace(start, end, (end - start) * 1000)
    data = [Scatter(x=x_vals, y=_f(x_vals), name='f(x)'), Scatter(x = x_vals, y = p(x_vals), name='P(x)')]

    if plot_max_error == True:
        y_err = max_error()
        x_err = x_of_max_error()
        data.append(Scatter(x=[x_err for i in range(100)], y=np.linspace(_f(x_err), p(x_err), 100), name='Error'))

    plotly.offline.iplot({
        "data": data,
        "layout": Layout(title="Function and approximation")
    })

plot_approximation(True)

def sign(x):
    if x > 0: return '+'
    elif x < 0: return '-'
    else: return 0

sign = np.vectorize(sign)

def change_alternance(): # change alternance
    global alternance
    x_err = x_of_max_error()
    temp = alternance[:]
    temp.append(x_err)
    temp.sort()
```

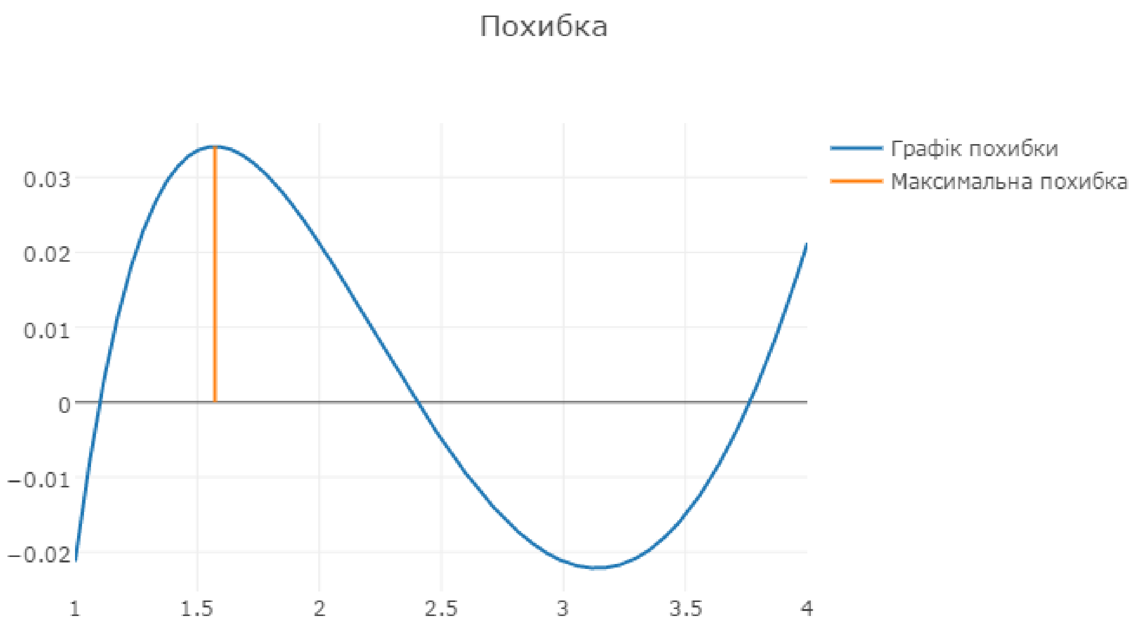
```python
    index_of_x_err = temp.index(x_err)
    if index_of_x_err != 0 and index_of_x_err != (len(temp)-1):
        if sign(error()(temp[index_of_x_err])) == sign(error()(temp[index_of_x_err-1])):
            del temp[index_of_x_err-1]

        else: del temp[index_of_x_err+1]

        alternance = temp[:]
    else: print('Index {}'.format(index_of_x_err))


alternance = [start + (end-start) * k / float(degree + 1) for k in range(degree+2)]
iterations = 1
while abs(abs(max_error()) - abs(error_on_iteration)) / abs(error_on_iteration) > precision:
    print('Alternance before: {}'.format(alternance))
    print('Signs of alternance: {}'.format(sign(error()(alternance))))
    print('Max error: {:.5f}'.format(max_error()))
    print('Error on iteration: {:.5f}'.format(error_on_iteration))
    print('Error in each point of alternance: {}', error()(alternance))
    print('X in which max error: {:.5f}'.format(x_of_max_error()))
    change_alternance()

    print('Alternance before: {}'.format(alternance))
    print('Error in each point of alternance: {}', error()(alternance))
    plot_error_function(True)

    print('\n\n')
    iterations += 1

print('Max error: {}'.format(max_error()))
print('Error on iteration: {}'.format(error_on_iteration))
print('Difference of errors: {}'.format(abs(abs(max_error()) - abs(error_on_iteration)) / abs(error_on_iteration)))
print('Iterations: {}'.format(iterations))
```

4.2 Приклади виконання програми

Приклад 1

Знайдемо чебишовське наближення поліномом степеня 2 для функції $f(x) = ln(x)$ на проміжку $[1, 4]$. Точність ($\epsilon = 0.01$)

Ітерація №1

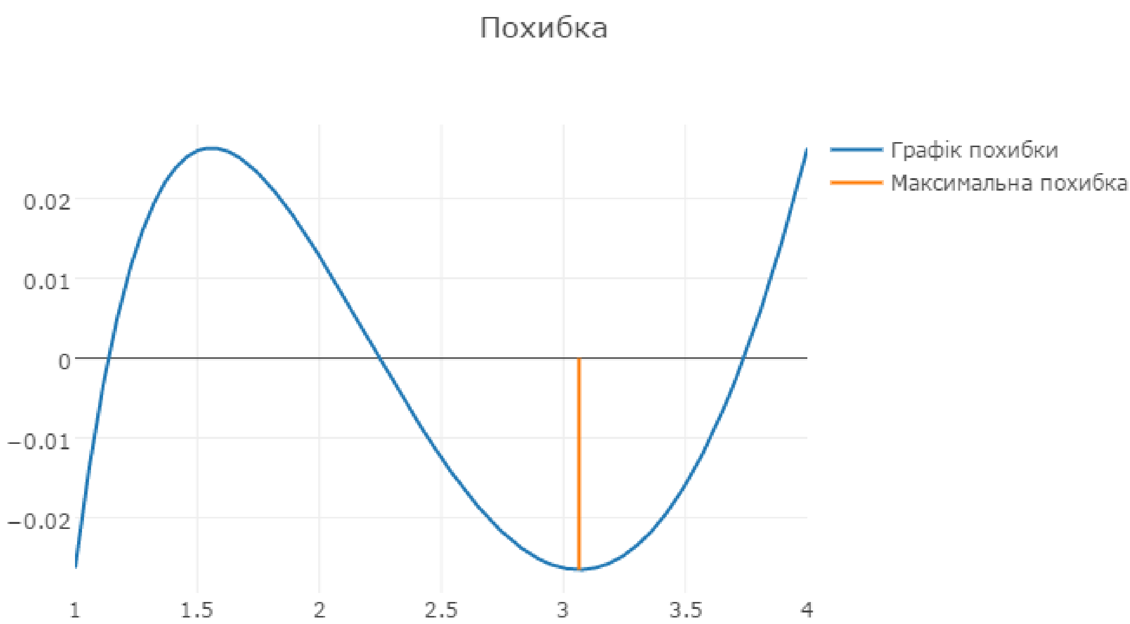


Точки альтернансу	1.0	2.0	3.0	4.0
Похибка	-0.02124	0.02124	-0.02124	0.02124

Максимальна похибка: **0.03411**

Значення x в якому досягається максимальна похибка: **1.57232**

Ітерація №2



Точки альтернансу	1.0	1.57232	3.0	4.0
Похибка	-0.02630	0.02630	-0.02630	0.02630

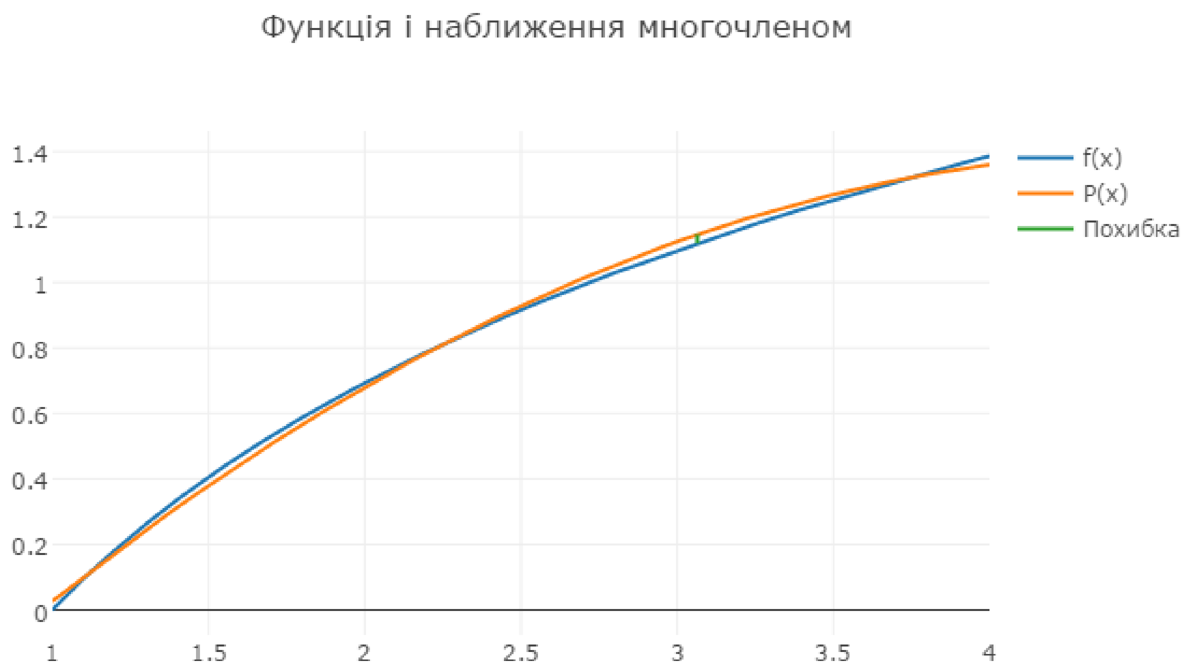
Максимальна похибка: **-0.02651**
Значення x в якому досягається максимальна похибка: **3.06447**

Перевіримо умову завершення алгоритму: $\frac{\rho_j - |\mu_j|}{|\mu_j|} \leq \epsilon$.

$$\frac{0.02651 - 0.02630}{0.02630} = 0.00798 < \epsilon = 0.01$$

Оскільки умова виконується то многочлен чебишовського наближення знайдено. Його вигляд:

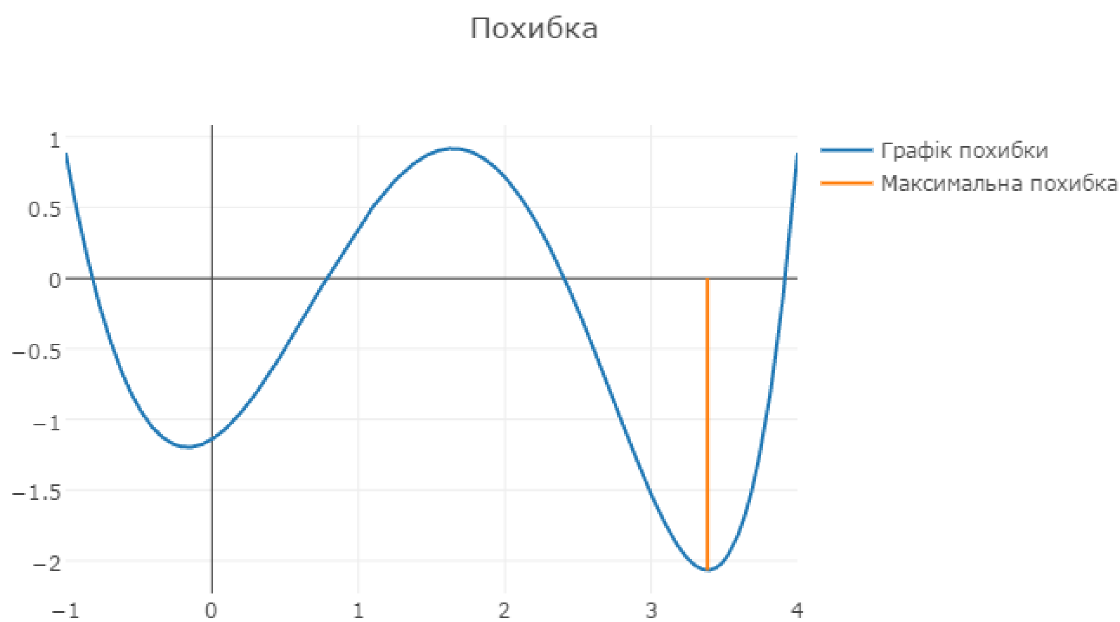
$$P_2(x) = -0.10474x^2 + 0.96826x - 0.83723$$



Приклад 2

Знайдемо чебишовське наближення поліномом степеня 3 для функції $f(x) = e^x$ на проміжку $[-1, 4]$. Точність ($\epsilon = 0.01$)

Ітерація №1

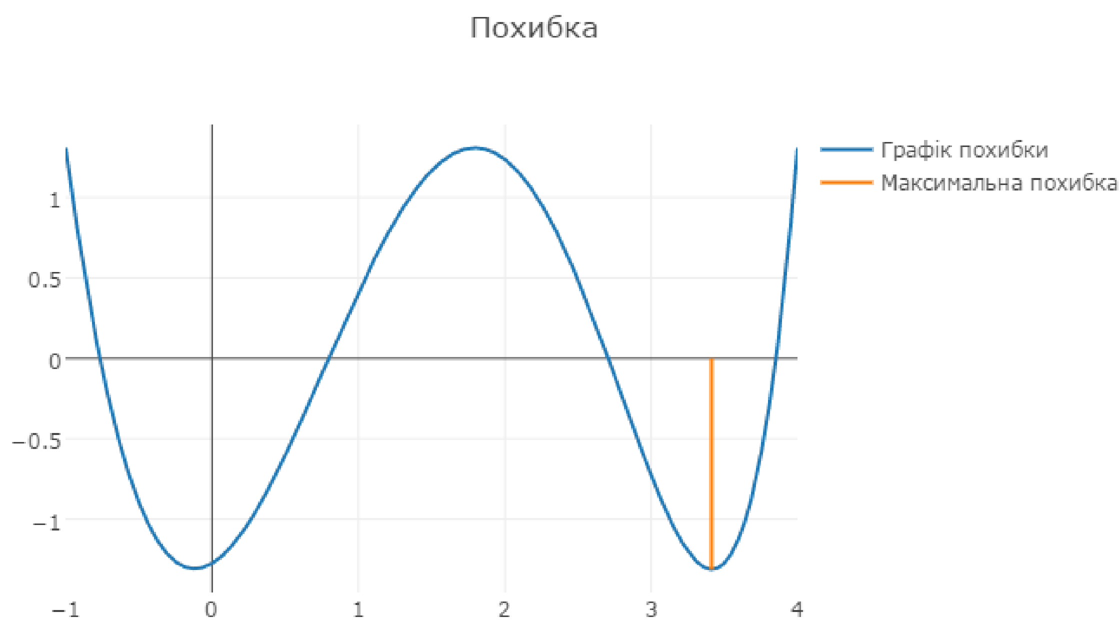


Точки альтернансу	-1.0	0.25	1.5	2.75	4.0
Похибка	0.88435	-0.88435	0.88435	-0.88435	0.88435

Максимальна похибка: **-2.06719**

Значення x в якому досягається максимальна похибка: **3.38529**

Ітерація №4



Точки альтернансу	-1.0	-0.12428	1.79496	3.38529	4.0
Похибка	1.30793	-1.30793	1.30793	-1.30793	1.30793

Максимальна похибка: **-1.31156**
Значення x в якому досягається максимальна похибка: **3.41309**

Перевіримо умову завершення алгоритму: $\frac{\rho_j - |\mu_j|}{|\mu_j|} \le \epsilon$.

$$\frac{1.31156 - 1.30793}{1.30793} = 0.00278 < \epsilon = 0.01$$

Оскільки умова виконується то многочлен чебишовського наближення знайдено. Його вигляд:

$$P_2(x) = 1.16656x^3 - 1.59181x^2 + 0.45627x + 2.27459$$