**Лабораторна робота №1**

**Моделювання неперервно-детермінованих моделей на ЕОМ**

**Мета роботи**: навчитись здійснювати моделювання на ЕОМ неперервно-детермінованих моделей.

**Завдання**

Стан системи описується наступним рівнянням:



з початковими умовами ; .

Виконати моделювання стану системи на інтервалі часу [0, 180] секунд. Представити графіки залежностей  і . Навести значення кроку інтегрування за часом .

Данні для моделювання за варіантами наведені в таблиці 1, а  і  (для варіантів з 1 по 14);  і  (для варіантів з 15 по 28); ; .

Таблиця 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варіанту |  |  |  |  |
| 1, 15 | 0,01 | 0,0 | 0,012 | 0,154 |
| 2, 16 | 0,01 | 0,2 | 0,030 | 0,161 |
| 3, 17 | 0,01 | 1,0 | 0,060 | 0,151 |
| 4, 18 | 0,01 | 2,0 | 0,100 | 0,165 |
| 5, 19 | 0,01 | 3,0 | 0,130 | 0,169 |
| 6, 20 | 0,01 | 1,0 | 0,075 | 0,168 |
| 7, 21 | 0,01 | 3,0 | 0,100 | 0,145 |
| 8, 22 | 0,01 | 1,0 | 0,060 | 0,142 |
| 9, 23 | 0,01 | 3,0 | 0,130 | 0,152 |
| 10, 24 | 0,01 | 1,0 | 0,080 | 0,153 |
| 11, 25 | 0,01 | 3,0 | 0,150 | 0,166 |
| 12, 26 | 0,04 | 0,2 | 0,040 | 0,170 |
| 13, 27 | 0,04 | 1,0 | 0,070 | 0,160 |
| 14, 28 | 0,07 | 0,6 | 0,070 | 0,179 |

Варіант 6: b1 = 0.01; b2 = 1; sigma = 0.075; y0 = 0.168; c1 = 1; c3 = -1; ; 

Короткі теоретичні відомості

Для вирішення диференційного рівняння можна використати чисельний метод рішення – метод Ейлера. За цим методом для рівняння вигляду , де *x* – незалежна змінна, рішення знаходиться за такою формулою:

.

При застосуванні чисельного методу Ейлера можлива похибка рішення за рахунок:

1. похибки, пропорційній кроку ,
2. похибки методу (метод є наближеним),
3. накопичення похибки у процесі обчислення.

Тому треба застосовувати критерії оцінки точності рішення. Наприклад, інтегрований критерій оцінки:

,

де  – множина рішень на *j*-й ітерації,  – відповідна множина рішень на (*j*+1)-й ітерації, причому .

При виконанні умови критерію для рішення, воно вважається знайденим. Інакше – крок  ще раз зменшується і знаходиться нове рішення, для якого знову застосовується критерій оцінювання. Так продовжується доки, поки умова критерію не буде задовільнена.

Метод Ейлера застосовується для рішення рівнянь першого порядку.

Для зниження порядку рівняння вводяться нові змінні та складається система рівнянь з цими змінними. Наприклад, якщо є рівняння другого порядку , то при введенні нових змінних  і , складається система з двох рівнянь першого порядку: , для якої можна застосувати метод Ейлера.

Хід роботи

Подане рівняння – другого порядку, тому треба ввести нові змінні  і .

Складаємо систему рівнянь першого порядку, еквівалентну поданому рівнянню:



Чисельна модель:

Алгоритм знаходження розв’язку рівняння показано на рисунку 1.



Рисунок 1 - Алгоритм знаходження розв'язку рівняння

Розв’язок знайдемо за допомогою мови JavaScript, та на основі результатів побудуємо графік, що дозволить змінювати параметри та відразу бачити як вони впливають на результат.

Текст програми:

*Index.js*

import { debounce } from '../node\_modules/throttle-debounce/esm/index.js';

import findSolution from './findSolution.js';

import { makeResultChart } from './resultChart.js';

const $p = document.querySelector('.page-js');

const $args = $p.querySelector('.args-js');

const $chart = $p.querySelector('.chart-js');

const chart = makeResultChart($chart);

initEvents();

update();

function initEvents() {

const delayedUpdate = debounce(500, update);

$args.addEventListener('submit', (e) => {

e.preventDefault();

update();

});

$args.addEventListener('input', delayedUpdate);

$args.addEventListener('change', delayedUpdate);

}

function update() {

try {

const solution = findSolution(getArgs());

console.log(solution);

updateChart(solution);

} catch (e) {

console.warn('Incorrect data');

console.error(e);

}

}

function updateChart({ x, y1, y2 }) {

chart.data.labels = x.map(y => y.toFixed(4));

chart.data.datasets[0].data = y1.map(y => y.toFixed(4));

chart.data.datasets[1].data = y2.map(y => y.toFixed(4));

chart.update();

}

function getArgs() {

return {

b1: +$args.b1.value,

b2: +$args.b2.value,

sigma: +$args.sigma.value,

y0: +$args.y0.value,

c1: +$args.c1.value,

c3: +$args.c3.value,

w: +$args.w.value,

epsilon: +$args.epsilon.value,

from: +$args.from.value,

to: +$args.to.value,

}

}

*FindSolition.js*

export default function findSolution({ b1, b2, sigma, y0, c1, c3, w, epsilon, from, to }) {

const res = [];

let delta = 2;

while (true) {

const n = Math.floor((to - from) / delta);

const y1 = new Array(n + 1);

const y2 = new Array(n + 1);

const x = new Array(n + 1);

y1[0] = y0;

y2[0] = 0;

for (let i = 1; i <= n; i++) {

const t = x[i] = from + delta \* i;

const y2Prev = y2[i - 1];

const f = sigma \* Math.sin(w \* t) - b1 \* y2Prev -

b2 \* y2Prev \* Math.abs(y2Prev) - c1 \* y1[i - 1] - c3 \* Math.pow(y2Prev, 3);

y2[i] = y2Prev + f \* delta;

y1[i] = y1[i - 1] + y2Prev \* delta;

}

let sum = 0

for (let i = 1; i <= n; i++) {

sum += Math.pow(y1[i] - y1[i - 1], 2);

}

const max = Math.max(...y1);

const rule = Math.sqrt(sum / n / max);

res.push({ delta, rule })

if (rule >= epsilon || Number.isNaN(rule)) {

delta /= 2;

continue;

}

console.log(rule);

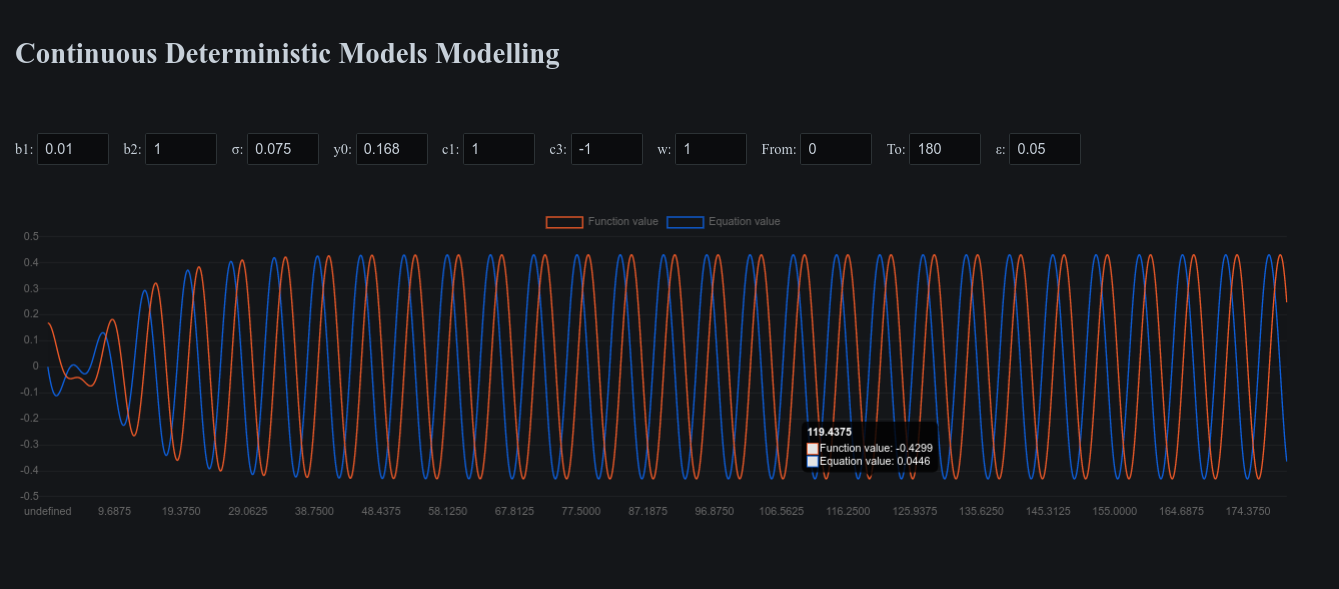
return { x, y1, y2 };

}

}

Результат виконання:

GitHub Pages: <https://dimagashko.github.io/NUK_Projects/systems-modelling/l2/>



**Висновок:** виконавши лабораторну роботу, я набув теоретичних знань та практичних навичок аналізу розрахунків на ЕОМ. За результатами обчислень можна бачити, що тип даних double (float64 у мові GoLang) має більшу точність обчислень, ніж float, що зрозуміло, адже для чисел типу double у оперативній пам’яті виділяється у два рази більше байтів – 4 для float, та 8 для double.