Міністерство освіти і науки україни

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Математичний факультет

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Дисципліна «Автоматизоване тестування»

Лабораторна робота № 3

Виконав студент гр. 8.1214

Ніценко Вадим

Перевірив: Викладач

Лебідь Максим Володимирович

Запоріжжя

2024

**Завдання:**

1. Ознайомитись з програмною реалізацією алгоритму розв’язання СЛАР https://github.com/LiannaAsatryan/gauss\_js
2. Реалізувати mock методи класу Matrix.
3. Створити github репозиторій з тестами.

Метод Гаусса полягає у поступовому перетворенні системи рівнянь до верхньої трикутної матриці з подальшим обчисленням невідомих з кінця.

Хід роботи

Структура програми складається з наступних файлів:

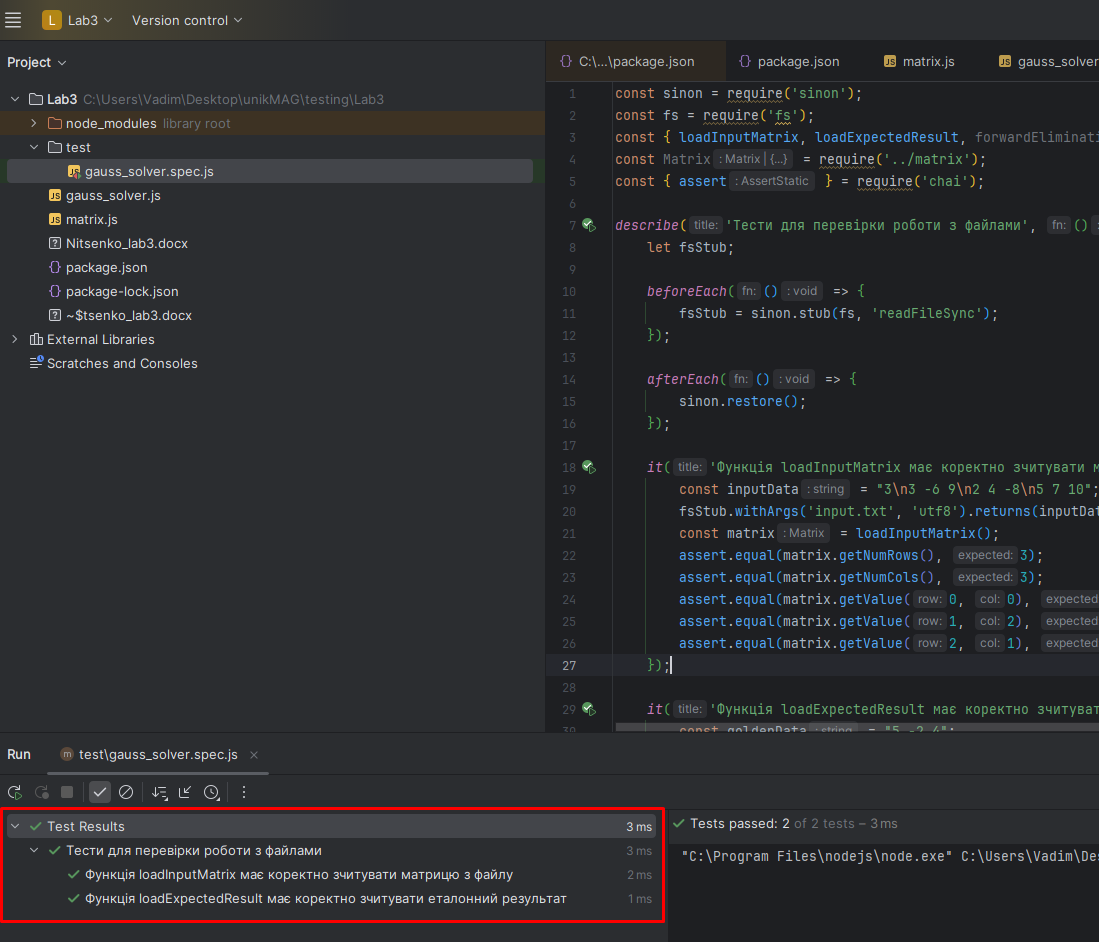
* matrix.js: містить клас Matrix з методами для роботи з матрицями
* gauss\_functions.js: основні функції для розв’язання системи рівнянь
* gauss\_functions.spec.js: файл для тестування основних функцій

**Код matrix.js**

class Matrix {  
 constructor(rows, cols) {  
 this.rows = rows;  
 this.cols = cols;  
 this.data = *Array*.from({ length: rows }, () => *Array*(cols).fill(0));  
 }  
  
 getNumRows() {  
 return this.rows;  
 }  
  
 getNumCols() {  
 return this.cols;  
 }  
  
 setValue(row, col, value) {  
 this.data[row][col] = value;  
 }  
  
 getValue(row, col) {  
 return this.data[row][col];  
 }  
  
  
 addScaledRow(targetRow, sourceRow, scale) {  
 for (let col = 0; col < this.cols; col++) {  
 this.data[targetRow][col] += this.data[sourceRow][col] \* scale;  
 }  
 }  
  
  
 hasInvalidRow() {  
 return this.data.some(row => row.slice(0, this.cols - 1).every(value => value === 0) && row[this.cols - 1] !== 0);  
 }  
}  
  
module.exports = Matrix;

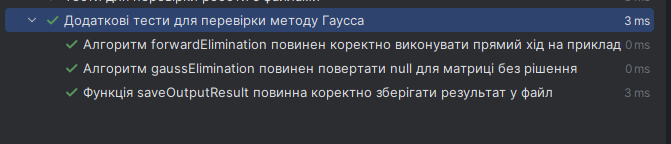
Створюємо тести для перевірки роботи з файлами

*describe*('Тести для перевірки роботи з файлами', () => {  
 let fsStub;  
  
 *beforeEach*(() => {  
 fsStub = sinon.stub(fs, 'readFileSync');  
 });  
  
 *afterEach*(() => {  
 sinon.restore();  
 });  
  
 *it*('Функція loadInputMatrix має коректно зчитувати матрицю з файлу', () => {  
 const inputData = "3\n3 -6 9\n2 4 -8\n5 7 10";  
 fsStub.withArgs('input.txt', 'utf8').returns(inputData);  
 const matrix = loadInputMatrix();  
 assert.equal(matrix.getNumRows(), 3);  
 assert.equal(matrix.getNumCols(), 3);  
 assert.equal(matrix.getValue(0, 0), 3);  
 assert.equal(matrix.getValue(1, 2), -8);  
 assert.equal(matrix.getValue(2, 1), 7);  
 });  
  
 *it*('Функція loadExpectedResult має коректно зчитувати еталонний результат', () => {  
 const goldenData = "5 -2 4";  
 fsStub.withArgs('golden.txt', 'utf8').returns(goldenData);  
 const expectedResult = loadExpectedResult();  
 assert.deepEqual(expectedResult, ["5", "-2", "4"]);  
 });  
});



Cтворюємо наступний тест для перевірки методу Гаусса

*describe*('Додаткові тести для перевірки методу Гаусса', () => {  
 *it*('Алгоритм forwardElimination повинен коректно виконувати прямий хід на прикладі простої матриці', () => {  
 const matrix = new Matrix(3, 4);  
 matrix.setValue(0, 0, 2);  
 matrix.setValue(0, 1, -1);  
 matrix.setValue(0, 2, 3);  
 matrix.setValue(0, 3, 8);  
 matrix.setValue(1, 0, 4);  
 matrix.setValue(1, 1, 2);  
 matrix.setValue(1, 2, -2);  
 matrix.setValue(1, 3, 5);  
 matrix.setValue(2, 0, -3);  
 matrix.setValue(2, 1, 1);  
 matrix.setValue(2, 2, 2);  
 matrix.setValue(2, 3, -7);  
  
 forwardElimination(matrix);  
 assert.equal(matrix.getValue(1, 0), 0);  
 });  
  
 *it*('Алгоритм gaussElimination повинен повертати null для матриці без рішення', () => {  
 const matrix = new Matrix(3, 4);  
 matrix.setValue(0, 0, 0);  
 matrix.setValue(0, 1, 0);  
 matrix.setValue(0, 2, 3);  
 matrix.setValue(0, 3, 4);  
 matrix.setValue(1, 0, 0);  
 matrix.setValue(1, 1, 0);  
 matrix.setValue(1, 2, -2);  
 matrix.setValue(1, 3, 5);  
 matrix.setValue(2, 0, 0);  
 matrix.setValue(2, 1, 0);  
 matrix.setValue(2, 2, 0);  
 matrix.setValue(2, 3, 1);  
  
 const result = gaussElimination(matrix);  
 assert.isNull(result, 'Матриця без рішення повинна повертати null');  
 });  
  
 *it*('Функція saveOutputResult повинна коректно зберігати результат у файл', () => {  
 const outputStub = sinon.stub(fs, 'writeFileSync');  
 saveOutputResult([-3, 2, 7]);  
 assert(outputStub.calledOnceWith('output.txt', '-3 2 7'));  
 outputStub.restore();  
 });  
});



Контрольні запитання

1. Наведіть приклади використання Fake, Stub, Mock.

* Fake — це фейкова (підставна) реалізація функції або об’єкта, яка працює подібно до реального об'єкта, але з мінімальною кількістю функціоналу.
* Приклад: Створення підставної бази даних для тестування, яка веде лог або використовує масив замість справжнього з’єднання з базою даних.
* Stub — це об’єкт, який замінює оригінальну функцію, але замість реальної логіки виконує певні дії (наприклад, повертає заздалегідь визначені значення). Стаб не викликає реального коду, лише симулює поведінку функції. Приклад: У випадку, коли потрібно перевірити функцію, що звертається до бази даних, використовують стаб, щоб повернути заздалегідь підготовлені дані та уникнути доступу до реальної БД.
* Mock — це об’єкт, який дозволяє задавати очікувану поведінку та перевіряти, чи виконувалися виклики у певному порядку. Моки часто використовують для перевірки складної логіки. Приклад: Перевірка виклику функції авторизації з конкретними аргументами, щоб переконатися, що метод викликався з правильними даними.

1. Особливості модульного та приймального тестування.

**Модульне тестування -** це тестування окремих компонентів або функцій програми незалежно від інших частин системи. Основна мета — перевірити коректність роботи окремих частин. Особливість цього тестування що воно використовуються для перевірки логіки конкретних методів чи класів; часто використовують підставні об'єкти.

Приймальне тестування (Acceptance Testing) — це тестування, яке проводиться з метою перевірки, чи відповідає програмний продукт вимогам замовника. Особливість цього тестування в тому що воно зазвичай включає повну перевірку системи в умовах, максимально наближених до реальних; фокусується на тому, щоб система працювала як єдине ціле

1. Призначення бібліотек Sinon та Sinon-Chai.

Sinon — це бібліотека для створення фейкових об’єктів, стабів та моків, яка надає можливість тестування взаємодії компонентів у Node.js та JavaScript. Sinon дозволяє контролювати функціонал без необхідності змінювати реальний код.

* Спрощує тестування ізоляцією частин коду, створенням стабів, моків та підрахунком викликів функцій.

Sinon-Chai — це розширення для Chai, яке додає специфічні асерти до Sinon, спрощуючи синтаксис для перевірки викликів функцій, стабів та моків.

* Забезпечує зручний синтаксис для перевірки станів моків та стабів