Міністерство освіти і науки україни

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Математичний факультет

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Дисципліна «Автоматизоване тестування»

Лабораторна робота № 3

Виконав студент гр. 8.1214

Гавриш Іван Васильович

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.

**Хід роботи:**

1. Наведений репозиторій містить два головні файли для розв’язання СЛАР gauss\_functions.js і matrix.js. Перший містить функції, які відповідають за зчитування матриці та відповідей із текстових файлів, розв’язання СЛАР методом Гаусса, перевірки вхідного файлу та запису розв’язків у файл. Другий є реалізацією класу matrix, який містить основні методи для роботи з матрицями: створення та вивід матриці, отримання та встановлення її елементів, методи для перевірки на наявність нульових та некоректних рядків у матриці тощо.
2. Перейдемо до тестування функцій з файлу gauss\_functions, де замість конкретних реалізацій матриці та об’єкту, для роботи з файловою системою, будуть використовуватись stubs та mocks.

2.1. Створюємо тести до функцій read\_input та read\_golden, які використовують модуль fs для зчитування інформації з файлів. Для імітації процесу зчитування з файлу створюємо stubs, тобто заглушки, для методу readFileSync у методі beforeEach, що буде викликатись перед початком виконання кожного тесту. Для функції read\_input програмуємо stub на повернення об’єкту матриці з елементами з файлу, для read\_golden – повернення списку відповідей.

**describe("reading from files", () => {**

**let fsStub;**

**beforeEach(() => {**

**fsStub = sinon.stub(fs, "readFileSync");**

**});**

**afterEach(() => {**

**fsStub.restore();**

**});**

**it("read\_input should read a file, parse the matrix and return it as the matrix object", () => {**

**const inputFile = "2\n1 2 3\n4 5 6\n";**

**fsStub.withArgs("input.txt", "utf8").returns(inputFile);**

**const matrix = read\_input();**

**console.log(matrix);**

**assert.equal(matrix.get\_rows(), 2);**

**assert.equal(matrix.get\_cols(), 3);**

**});**

**it("read\_golden should read a file with correct answers and return it as a list", () => {**

**const inputFile = "-1 2";**

**fsStub.withArgs("golden.txt", "utf8").returns(inputFile);**

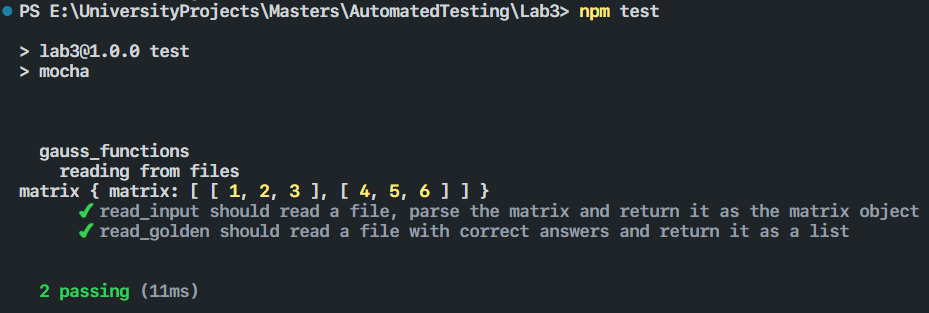
**const golden\_list = read\_golden();**

**assert.deepEqual(golden\_list, ["-1", "2"]);**

**});**

**});**

Виконуємо тести:



Тести виконуються успішно, отже функції, які тестуються, виконують процес зчитування та обробки даних з файлів коректно.

* 1. Протестуємо функцію gauss\_forward, яка отримує матрицю як параметр і виконує елімінацію вперед методом Гаусса. Створюємо mock об’єкт матриці і задаємо для нього поведінку, яку ми очікуємо при виконанні методу, що тестується:

**describe("gauss\_forward", () => {**

**it("performs gauss forward elimination when the pivot element is non-zero", () => {**

**const matrix = {**

**get\_rows: function () {},**

**get\_cols: function () {},**

**get: function () {},**

**mull\_add: function () {},**

**swap\_with\_nonzero\_row: function () {},**

**};**

**const matrixMock = sinon.mock(matrix);**

**matrixMock.expects('get\_rows').once().returns(2);**

**matrixMock.expects('get\_cols').once().returns(3);**

**matrixMock.expects('get').withArgs(0, 0).atLeast(1).returns(1);**

**matrixMock.expects('get').withArgs(1, 0).returns(4);**

**matrixMock.expects('mull\_add').withArgs(1, 0, -4 / 1).once();**

**matrixMock.expects('swap\_with\_nonzero\_row').never();**

**gauss\_forward(matrix);**

**matrixMock.verify();**

**});**

**it("performs row swap when the pivot element is zero", () => {**

**const matrix = {**

**get\_rows: function () {},**

**get\_cols: function () {},**

**get: function () {},**

**mull\_add: function () {},**

**swap\_with\_nonzero\_row: function () {},**

**};**

**const matrixMock = sinon.mock(matrix);**

**matrixMock.expects('get\_rows').once().returns(2);**

**matrixMock.expects('get\_cols').once().returns(3);**

**matrixMock.expects('get').withArgs(0, 0).atLeast(1).returns(0);**

**matrixMock.expects('swap\_with\_nonzero\_row').withArgs(0).once();**

**matrixMock.expects('mull\_add').never();**

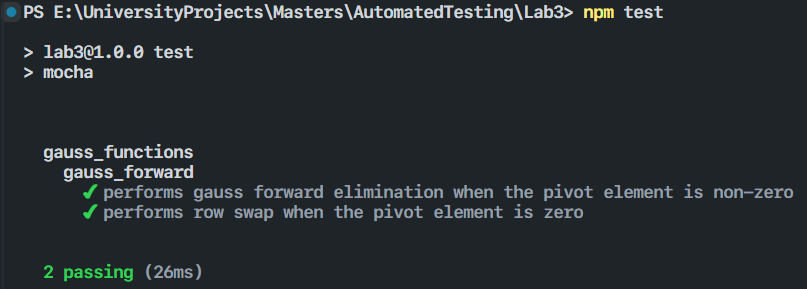
**gauss\_forward(matrix);**

**matrixMock.verify();**

**});**

**});**

Виконуємо тести:



2.3 Протестуємо функцію gauss\_backward, яка отримує матрицю як параметр і виконує елімінацію назад методом Гаусса. Так само створюємо mock об’єкт матриці і задаємо для нього поведінку, яку ми очікуємо при виконанні методу, що тестується:

**describe("gauss\_backward", () => {**

**it("performs gauss backward elimintaion", () => {**

**const matrix = {**

**get\_rows: function () {},**

**get\_cols: function () {},**

**get: function () {},**

**};**

**const matrixMock = sinon.mock(matrix);**

**// Expectations for rows and columns**

**matrixMock.expects('get\_rows').once().returns(2);**

**matrixMock.expects('get\_cols').once().returns(3);**

**matrixMock.expects('get').withArgs(0, 0).returns(1);**

**matrixMock.expects('get').withArgs(0, 1).returns(2);**

**matrixMock.expects('get').withArgs(0, 2).returns(3);**

**matrixMock.expects('get').withArgs(1, 1).returns(5);**

**matrixMock.expects('get').withArgs(1, 2).returns(6);**

**let result = gauss\_backward(matrix);**

**assert.equal(result.length, 2); // equals the number of rows**

**assert.isArray(result);**

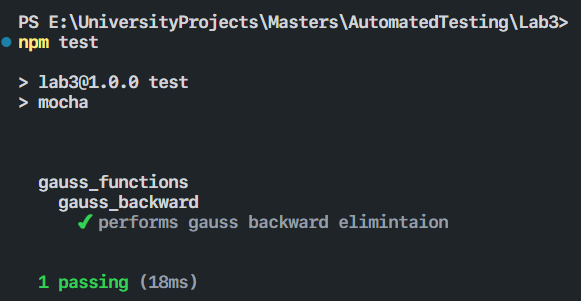
**assert.isNotEmpty(result);**

**matrixMock.verify();**

**});**

**});**

Виконуємо тест:



2.4 Протестуємо функцію gauss, яка отримує матрицю як параметр і виконує розв’язання СЛАР методом Гаусса. Для методів класу matrix створюємо mock відповідники:

**describe("gauss", () => {**

**it("performs gauss elimination", () => {**

**const matrix = {**

**get\_rows: function () {},**

**get\_cols: function () {},**

**get: function () {},**

**mull\_add: function () {},**

**swap\_with\_nonzero\_row: function () {},**

**exists\_wrong\_row: function () {},**

**exists\_zero\_row: function () {}**

**};**

**const matrixMock = sinon.mock(matrix);**

**let result = gauss(matrix);**

**assert.isEmpty(result);**

**matrixMock.verify();**

**});**

**it('should return null when there exists wrong row', () => {**

**const matrix = {**

**get\_rows: function () {},**

**get\_cols: function () {},**

**get: function () {},**

**mull\_add: function () {},**

**swap\_with\_nonzero\_row: function () {},**

**exists\_wrong\_row: function () {},**

**exists\_zero\_row: function () {}**

**};**

**const matrixMock = sinon.mock(matrix);**

**matrixMock.expects('exists\_wrong\_row').returns(true).once();**

**matrixMock.expects('exists\_zero\_row').returns(false).once();**

**const result = gauss(matrix);**

**assert.strictEqual(result, null);**

**});**

**it('should return null when there exists zero row', () => {**

**const matrix = {**

**get\_rows: function () {},**

**get\_cols: function () {},**

**get: function () {},**

**mull\_add: function () {},**

**swap\_with\_nonzero\_row: function () {},**

**exists\_wrong\_row: function () {},**

**exists\_zero\_row: function () {}**

**};**

**const matrixMock = sinon.mock(matrix);**

**matrixMock.expects('exists\_wrong\_row').returns(false).once();**

**matrixMock.expects('exists\_zero\_row').returns(true).once();**

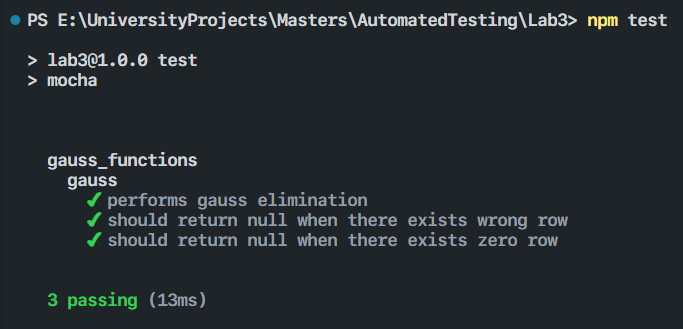
**const result = gauss(matrix);**

**assert.strictEqual(result, null);**

**});**

**});**

Виконуємо тести:



2.5 Виконаємо тестування функції write\_in\_file для запису даних у файл, яка використовуює методи writeFileSyncStub, appendFileSyncStub модулю fs, і для яких будуть створені stub.

**describe('write\_in\_file', () => {**

**let writeFileSyncStub, appendFileSyncStub, readFileSyncStub;**

**beforeEach(function () {**

**writeFileSyncStub = sinon.stub(fs, 'writeFileSync');**

**appendFileSyncStub = sinon.stub(fs, 'appendFileSync');**

**readFileSyncStub = sinon.stub(fs, 'readFileSync');**

**});**

**afterEach(function () {**

**sinon.restore();**

**});**

**it('should append the list values to the file when list\_name is valid', () => {**

**readFileSyncStub.returns("2\n1 2 3\n4 5 6\n");**

**const list\_name = [1.0, 2.0, 3.0];**

**write\_in\_file(list\_name);**

**sinon.assert.calledWithExactly(appendFileSyncStub, "output.txt", "1 ");**

**sinon.assert.calledWithExactly(appendFileSyncStub, "output.txt", "2 ");**

**sinon.assert.calledWithExactly(appendFileSyncStub, "output.txt", "3 ");**

**});**

**it('should write "no solution" to the file when list\_name is null', () => {**

**readFileSyncStub.returns("2\n1 2 3\n4 5 6\n");**

**write\_in\_file(null);  // Call function with null list\_name**

**sinon.assert.calledOnceWithExactly(writeFileSyncStub, "output.txt", "no solution");**

**sinon.assert.notCalled(appendFileSyncStub);  // appendFileSync should not be called**

**});**

**it('should write "wrong input" to the file when check\_input returns false', () => {**

**readFileSyncStub.returns("1 2 3\n4 5 6\n");**

**write\_in\_file([1, 2, 3]);**

**sinon.assert.calledOnceWithExactly(writeFileSyncStub, "output.txt", "wrong imput");**

**sinon.assert.notCalled(appendFileSyncStub);  // appendFileSync should not be called**

**});**

**});**

Виконуємо тести:

