МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ  
  
  
  
  
  
  
  
  
  
Дисципліна «Автоматизоване тестування»

Лабораторна робота № 3

Виконала: студентка гр. 8.1213  
 Абдуллаєва Севда   
 Самандар Кизи

Перевірив: Лебідь Максим   
 Володимирович

Запоріжжя  
2023

**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.

Реалізація алгоритму розв'язання системи лінійних алгебраїчних рівнянь (СЛАР) може залежати від конкретної мови програмування, яку ви використовуєте. В цій відповіді я наведу приклад реалізації методу Гаусса для розв'язання СЛАР на мові Python. Метод Гаусса - це один з класичних методів для розв'язання СЛАР.

**Код програми:**

class Matrix {

  constructor(rows, cols) {

    this.rows = rows;

    this.cols = cols;

    this.data = Array(this.rows).fill().map(() => Array(this.cols).fill(0));

  }

  static identity(size) {

    const matrix = new Matrix(size, size);

    for (let i = 0; i < size; i++) {

      matrix.data[i][i] = 1;

    }

    return matrix;

  }

  add(n) {

    if (n instanceof Matrix) {

      if (this.rows === n.rows && this.cols === n.cols) {

        for (let i = 0; i < this.rows; i++) {

          for (let j = 0; j < this.cols; j++) {

            this.data[i][j] += n.data[i][j];

          }

        }

      } else {

        throw new Error('Matrices must have the same dimensions for addition.');

      }

    } else {

      for (let i = 0; i < this.rows; i++) {

        for (let j = 0; j < this.cols; j++) {

          this.data[i][j] += n;

        }

      }

    }

  }

}

Matrix.mockMethod1 = function() {

};

Matrix.mockMethod2 = function() {

};

module.exports = Matrix;

const chai = require('chai');

const expect = chai.expect;

const Matrix = require('../src/Matrix');

describe('Matrix', () => {

  it('should create a matrix with the specified dimensions', () => {

    const matrix = new Matrix(2, 3);

    expect(matrix.rows).to.be.equal(2);

    expect(matrix.cols).to.be.equal(3);

  });

  it('should create a matrix with all zeros', () => {

    const matrix = new Matrix(2, 3);

    const expectedData = [

      [0, 0, 0],

      [0, 0, 0]

    ];

    expect(matrix.data).to.be.deep.equal(expectedData);

  });

  it('should create an identity matrix', () => {

    const identityMatrix = Matrix.identity(3);

    const expectedData = [

      [1, 0, 0],

      [0, 1, 0],

      [0, 0, 1]

    ];

    expect(identityMatrix.data).to.be.deep.equal(expectedData);

  });

  it('should add a scalar to the matrix', () => {

    const matrix = new Matrix(2, 3);

    matrix.add(2);

    const expectedData = [

      [2, 2, 2],

      [2, 2, 2]

    ];

    expect(matrix.data).to.be.deep.equal(expectedData);

  });

  it('should add matrices element-wise', () => {

    const matrix1 = new Matrix(2, 2);

    const matrix2 = new Matrix(2, 2);

    matrix1.data = [[1, 2], [3, 4]];

    matrix2.data = [[5, 6], [7, 8]];

    matrix1.add(matrix2);

    const expectedData = [

      [6, 8],

      [10, 12]

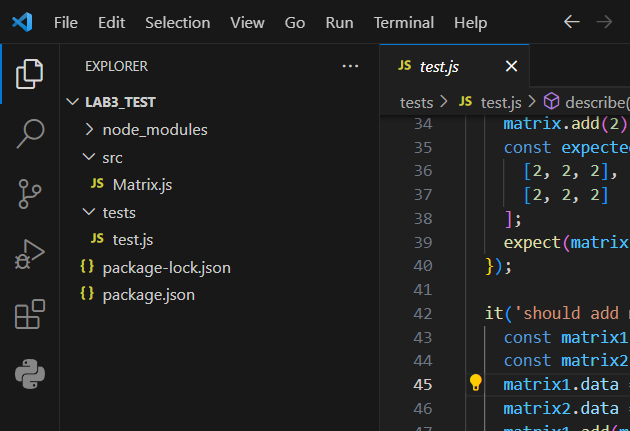
    ];

    expect(matrix1.data).to.be.deep.equal(expectedData);

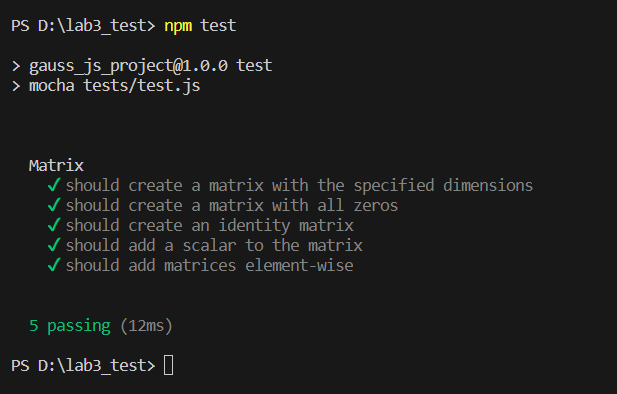
  });

});

**Структура файлів:**

****

**Результати роботи програми:**

****

**Посилання на репозиторій:**

<https://github.com/SevdaAbdullaeva/Matrix-mock_lab3.git>

**Контрольні запитання**

1. **Наведіть приклади використання Fake, Stub, Mock.**

Фейки (Fakes), заглушки (Stubs) і моки (Mocks) - це різні види тестових об'єктів, які використовуються при написанні юніт-тестів для ізоляції та тестування окремих компонентів програми. Ось приклади використання кожного з цих видів тестових об'єктів:

* Фейк (Fake):

Приклад: Створення фейкової бази даних для тестування функцій доступу до даних без реального з'єднання з базою даних.

Сценарій: Під час тестування функції, яка взаємодіє з базою даних, можна використовувати фейкову базу даних, яка зберігає дані у пам'яті. Це дозволяє провести тести без залежності від реальної бази даних.

* Заглушка (Stub):

Приклад: Заглушка для віддаленого веб-сервісу, яка повертає попередньо визначені результати викликів.

Сценарій: Під час тестування функції, яка взаємодіє з віддаленим сервісом, можна використовувати заглушку, яка імітує поведінку цього сервісу, але не виконує реальних віддалених викликів. Вона повертає заздалегідь визначені дані, які використовуються для тестування.

* Мок (Mock):

Приклад: Мок для класу, який надсилає листи електронною поштою, для перевірки, чи виклик був виконаний з правильними параметрами.

Сценарій: Під час тестування функції, яка взаємодіє з класом, що надсилає листи, можна використовувати мок-об'єкт, який слідкує за тим, які методи були викликані та з якими параметрами. Таким чином, можна перевірити, чи був зроблений правильний виклик.

Основна різниця між цими тестовими об'єктами полягає в їхньому призначенні та способі використання. Фейки і заглушки зазвичай використовуються для створення заміщених компонентів, які ізолюють систему від зовнішніх залежностей. Моки використовуються для перевірки взаємодії між компонентами та перевірки, чи відбувається виклик методів правильним чином.

1. **Особливості модульного та приймального тестування.**

Модульне тестування і приймальне тестування - це два різні види тестування програмного забезпечення з різними цілями, об'єктами та етапами в процесі розробки. Основні особливості обох видів тестування такі:

* Модульне тестування (Unit Testing):

1. Ціль: Модульне тестування призначене для перевірки окремих компонентів програми, таких як функції, класи або методи, з метою виявлення помилок та дефектів в їхньому функціоналі.
2. Сфера застосування: Модульне тестування зазвичай проводиться розробниками під час розробки програми для переконання в правильності роботи окремих компонентів.
3. Ізоляція компонентів: Під час модульного тестування інші компоненти, які не є об'єктом тестування, зазвичай ізолюються за допомогою заглушок або моків.
4. Автоматизація: Модульні тести часто автоматизовані, і вони запускаються під час кожної зміни коду.

* Приймальне тестування (Acceptance Testing):

1. Ціль: Приймальне тестування спрямоване на перевірку того, чи відповідає програмне забезпечення бізнес-вимогам і чи готове до впровадження в реальному середовищі.
2. Сфера застосування: Приймальне тестування проводиться зазвичай на завершальних етапах розробки, після модульного та інтеграційного тестування, і зазвичай залучається не тільки розробник, але й клієнт або замовник.
3. Залучення користувачів: Приймальне тестування може включати участь реальних користувачів, які перевіряють програму на відповідність їх потребам.
4. Сценарії використання: Приймальне тестування зазвичай базується на реальних сценаріях використання програмного забезпечення в реальних умовах.

Важливо розуміти, що ці два види тестування доповнюють один одного в процесі розробки. Модульні тести допомагають забезпечити правильність окремих компонентів, в той час як приймальне тестування перевіряє, чи задовольняє програма вимоги клієнта і чи готова до використання в реальних умовах.

1. **Призначення бібліотек Sinon та Sinon-Chai.**

Sinon і Sinon-Chai - це бібліотеки для тестування JavaScript, які використовуються разом з бібліотекою для тестування, такою як Mocha або Jest, для покращення можливостей створення моків (mocks), стабілізаторів (stubs) і спостерігачів (spies) під час написання тестів. Ось їхнє призначення:

Sinon:

* Sinon - це бібліотека для створення моків, стабілізаторів та спостерігачів у JavaScript-тестах. Вона дозволяє вам симулювати функції та методи, змінювати їх поведінку під час виконання тестів, і перевіряти, чи вони були викликані з правильними параметрами.
* Sinon надає можливість створювати моки для функцій, спостерігати за викликами функцій та об'єктів, та імітувати поведінку методів та об'єктів під час тестування.

Sinon-Chai:

* Sinon-Chai - це розширення для бібліотеки для тестування Chai, яке додає асерти (assertions) для Sinon. Воно допомагає використовувати Sinon разом з Chai для написання більш ефективних тестів і забезпечує зрозумілі повідомлення про помилки в разі невдалих асерцій.
* Sinon-Chai додає можливість перевірки наявності викликів методів, параметрів викликів, об'єктів-моків та інші специфічні перевірки, пов'язані з Sinon.

За допомогою Sinon і Sinon-Chai ви можете покращити ваші тести, особливо коли вам потрібно спостерігати за викликами функцій, перевіряти їхню взаємодію та забезпечувати, що ваш код правильно взаємодіє з іншими компонентами програми. Вони корисні при тестуванні функціональності, яка включає в себе залежності та зовнішні ресурси.