|  |
| --- |
| ФГБОУ ВО ЮГОРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ |
| ИНЖЕНЕРНАЯ ШКОЛА ЦИФРОВЫХ ТЕХНОЛОГИЙ |

ОТЧЕТ

по лабораторной работе №2

Тема:

«Интеграционное тестирование»

По дисциплине:

# «Тестирование и отладка программного обеспечения»

Выполнил: студент группы

1521б

Зорин Владислав Сергеевич

Проверил: старший преподаватель

Усманов Руслан Талгатович

г. Ханты-Мансийск, 2024 г.

# Цель работы: закрепить полученные навыки модульного тестирования и получить навыки интеграционного тестирования на языке программирования С#.

# **Индивидуальное задание**

|  |  |
| --- | --- |
| Номер варианта | Задание |
| 1 |  |

# Ход работы

1)Были написаны все необходимые функции (Приложение А).

2)Были написаны тесты (Приложение Б).

3)Все тесты были успешно пройдены (Рис.1).

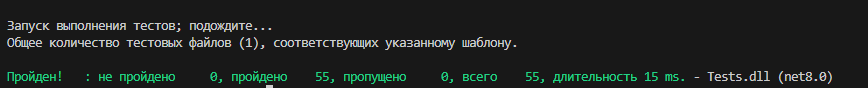


Рисунок 1 – Тестирование.

**Заключение**:

В ходе выполнения лабораторной работы я закрепил навыки модульного тестирования получил опыт в написании интеграционного тестирования на языке программирования C#.

**Приложение А.** Листинг файла Program.cs

using System;

namespace LABA2

{

public class Taylor

{

public double Pow(double x, int n)

{

if (n < 0)

{

return -1;

}

double result = 1;

for (int i = 0; i < n; i++)

{

result \*= x;

}

return result;

}

public double Abs(double x)

{

if (x < 0)

{

return x \* -1;

}

return x;

}

public long Factorial(long n)

{

if (n < 0 || n > 20)

{

return -1;

}

else if (n == 0)

{

return 1;

}

else

return n \* Factorial(n - 1);

}

public double Sin(double x)

{

if (x > 4 || x < -4)

{

return -1;

}

double result = 0;

for (int n = 0; n < 10; n++)

{

double term = Pow(-1, n) \* Pow(x, 2 \* n + 1) / Factorial(2 \* n + 1);

result += term;

}

return result;

}

public double Cos(double x)

{

if (x > 4 || x < -4)

{

return -1;

}

double result = 0;

for (int n = 0; n < 10; n++)

{

double term = Pow(-1, n) \* Pow(x, 2 \* n) / Factorial(2 \* n);

result += term;

}

return result;

}

public double Log(double x)

{

if (x <= 0 || x >= 2)

{

return -1;

}

double result = 0;

for (int n = 1; n <= 200; n++)

{

double term = Pow(-1, n - 1) \* Pow(x - 1, n) / n;

result += term;

}

return result;

}

public double firstStmt(double x)

{

if (x >= 0 || x < -2)

{

return -1;

}

double sin = Sin(x);

double log = Log(Abs(x));

double cos = Cos(log);

double underSqrt = sin \* cos;

if (underSqrt < 0)

{

return -2;

}

return Sqrt(underSqrt);

}

public double Sqrt(double x, double epsilon = 1e-15)

{

if (x < 0)

{

return -1;

}

double guess = x / 2;

while (Abs(guess \* guess - x) > epsilon)

{

guess = (guess + x / guess) / 2;

}

return guess;

}

public double secondStmt(double x)

{

if (x <= 0 || x > 5)

{

return -1;

}

return (1 - Cos(x)) / Sin(x);

}

public double fX(double x)

{

if (x <= 0)

{

return firstStmt(x);

}

return secondStmt(x);

}

static void Main()

{

}

}

}

**Приложение Б.** Листинг файла UnitTest1.cs

using LABA2;

using NUnit.Framework.Interfaces;

namespace Tests;

public class Tests

{

private Taylor obj;

[SetUp]

public void Setup()

{

obj = new Taylor();

}

[TestCase(0.2583,-0.1)]//firstStmt

[TestCase(0.0876,-0.2)]

[TestCase(-1,0)]

[TestCase(-2,-0.3)]

[TestCase(0.255342, 0.5)]//secondStmt

[TestCase(-1, 50)]

[TestCase(-2.18504, 4)]

public void Stmt(double res, double x)

{

Assert.That(obj.fX(x), Is.EqualTo(res).Within(0.0001));

}

[TestCase(-1,-3)]

[TestCase(0.0876,-0.2)]

[TestCase(-1,0)]

[TestCase(-2,-0.3)]

public void firstStmt(double res, double x)

{

Assert.That(obj.firstStmt(x), Is.EqualTo(res).Within(0.0001));

}

[TestCase(0.255342, 0.5)]

[TestCase(-1, 50)]

[TestCase(-1,-4)]

[TestCase(-2.18504, 4)]

public void secondStmt(double res, double x)

{

Assert.That(obj.secondStmt(x), Is.EqualTo(res).Within(0.0001));

}

[TestCase(0.995004, 0.1)]

[TestCase(0.995004, -0.1)]

[TestCase(-0.653644, 4)]

[TestCase(-0.653644, -4)]

[TestCase(-1, -5)]

[TestCase(-1, 100)]

public void Cos(double res, double x)

{

Assert.That(obj.Cos(x), Is.EqualTo(res).Within(0.0001));

}

[TestCase(0.0998334, 0.1)]

[TestCase(-0.0998334,-0.1)]

[TestCase(-0.756802, 4)]

[TestCase(0.756802, -4)]

[TestCase(-1, -6)]

[TestCase(-1, 100)]

public void Sin(double res, double x)

{

Assert.That(obj.Sin(x), Is.EqualTo(res).Within(0.0001));

}

[TestCase(-2.30259,0.1)]

[TestCase(0.641854,1.9)]

[TestCase(-1,5)]

[TestCase(-1,-3)]

public void Log(double res, double x)

{

Assert.That(obj.Log(x), Is.EqualTo(res).Within(0.0001));

}

[TestCase(1,1,2)]

[TestCase(1,123,0)]

[TestCase(-1,20,-2)]

[TestCase(-100000,-10,5)]

public void Pow(double res, double x, int n)

{

Assert.That(obj.Pow(x, n), Is.EqualTo(res).Within(0.0001));

}

[TestCase(2432902008176640000, 20)]

[TestCase(1, 0)]

[TestCase(-1, -1)]

[TestCase(-1, 21)]

public void Factorial(double res, long n)

{

Assert.That(obj.Factorial(n), Is.EqualTo(res).Within(0.0001));

}

[TestCase(1,1)]

[TestCase(1, -1)]

[TestCase(100.5, -100.5)]

[TestCase(10000, 10000)]

public void Abs(double res, double x)

{

Assert.That(obj.Abs(x), Is.EqualTo(res).Within(0.0001));

}

[TestCase(5,25)]

[TestCase(-1, -1)]

[TestCase(3.240370, 10.5)]

[TestCase(10, 100)]

public void Sqrt(double res, double x)

{

Assert.That(obj.Sqrt(x), Is.EqualTo(res).Within(0.0001));

}

[Test]

public void SinfirstStmt()

{

Assert.That(obj.firstStmt(obj.Sin(-0.201)), Is.EqualTo(0.0894).Within(0.0001));

}

[Test]

public void CosfirstStmt()

{

Assert.That(obj.firstStmt(obj.Cos(1.772)), Is.EqualTo(0.0894).Within(0.001));

}

[Test]

public void LogfirstStmt()

{

Assert.That(obj.firstStmt(obj.Log(0.818)), Is.EqualTo(0.0825).Within(0.0001));

}

[Test]

public void SinsecondStmt()

{

Assert.That(obj.secondStmt(obj.Sin(0.5236)), Is.EqualTo(0.2554).Within(0.0001));

}

[Test]

public void CossecondStmt()

{

Assert.That(obj.secondStmt(obj.Cos(1.0472)), Is.EqualTo(0.2554).Within(0.001));

}

[Test]

public void LogsecondStmt()

{

Assert.That(obj.secondStmt(obj.Log(1.6487)), Is.EqualTo(0.2554).Within(0.0001));

}

[Test]

public void fXfirstStmt()

{

Assert.That(obj.fX(obj.firstStmt(-0.1)), Is.EqualTo(0.129).Within(0.001));

}

[Test]

public void fXsecondStmt()

{

Assert.That(obj.fX(obj.secondStmt(0.5)), Is.EqualTo(0.129).Within(0.001));

}

}