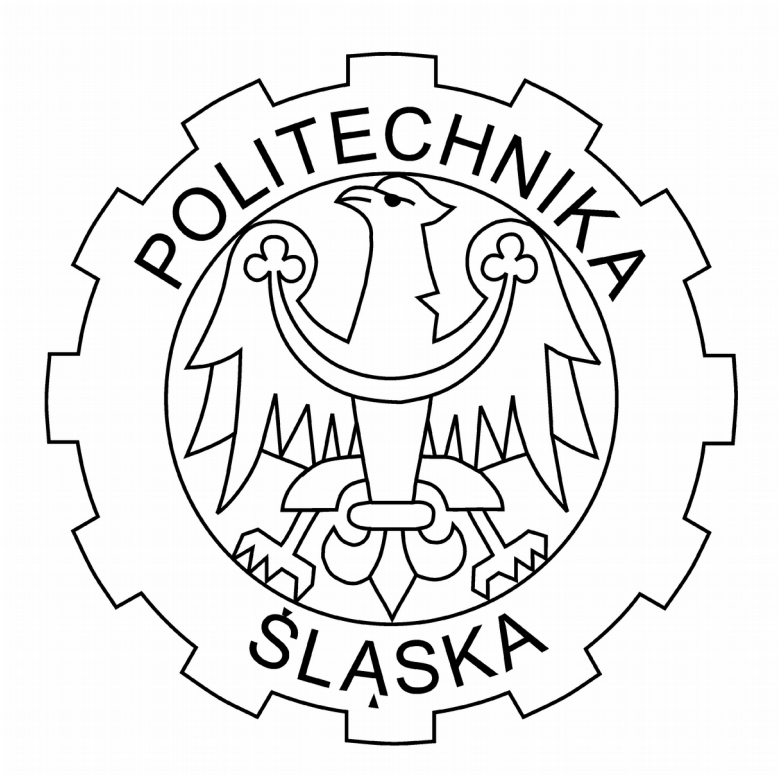
NUMERICAL METHODS

NUMERICAL INTEGRATION



Sergio Barbero, Mateusz Korusiewicz

Introduction to the problem

We will study several differents methods of approximation of integrals and we will study the error

1. Rectangular
2. Trapezoid
3. Simpson formula
4. Chebyshev method

Our problem

We are going to analyze this function: $f(x)=e^{-x}\sin(x)(2\cos(x)-\sin(x))$ in the range $x\epsilon[-3,-1]$

In order to get the analytical result we should perform:

$$\int_{-3}^{-1} f(x)=[\sin^2(-3)e^{(3)}]-[\sin^2(-1)e^{(1)}]=1.5247425170459024$$

Rectangular formula

```
public double Rect(double[] points) {
    double[] f = new double[points.length];
    double aproxArea = 0;
    for (int i = 0, k = -1; i < points.length; i++, k++) {
        f[i] = Math.pow(Math.E, -points[i]) * Math.sin(points[i]) * (2 * Math.cos(points[i]) - Math.sin(points[i]));
        if (i != 0){
                aproxArea += f[k] * (points[i] - points[k]);
        }
    }
    double error = Math.abs(realArea - aproxArea);

    System.out.println("Rectangular aproximate area: " + aproxArea);
    System.out.println("Error: " + error);
    System.out.println();

    return error;
}
```

This is the implementation of the method. We should get the area of the rectangle from one point to the next, and keep adding them to the final result. It’s a very simple method, but not so accurate.

Trapezoid formula

```
public double Trap(double[] points) {
    double[] f = new double[points.length];
    double aproxArea = 0;
    for (int i = 0, k = -1; i < points.length; i++, k++) {
        f[i] = Math.pow(Math.E, -points[i]) * Math.sin(points[i]) * (2 * Math.cos(points[i]) - Math.sin(points[i]));
        if (i != 0) {
            if(f[k] > 0)
                aproxArea += (points[i] - points[k]) * ((f[k] + Math.abs(f[i])) / 2);
            else
                aproxArea += -1* (points[i] - points[k]) * ((Math.abs(f[k]) + Math.abs(f[i])) / 2);
        }
    }
    double error = Math.abs(realArea - aproxArea);

    System.out.println("Trapezoid aproximate area: " + aproxArea);
    System.out.println("Error: " + error);
    System.out.println();
    return error;

}
```

In this case accuracy is a little bit increased, we are taking the area of the point in $f(x_i)$ and the next $f(x_{i+1})$.

Simpson formula

```
public double Simpson(double[] points){
    int n = points.length - 1;
    double h = points[1] - points[0];
    double f[] = new double[points.length];
    double aproxArea = 0; double error = 0;
    if(n % 2 == 0){
        double brackets = 0;
        for(int i=0; i <= n; i++){
            f[i] = Math.pow(Math.E, -points[i]) * Math.sin(points[i]) * (2 * Math.cos(points[i]) - Math.sin(points[i]));
            if(i % 3 == 0){
                brackets += f[i];
            }else if(i % 3 == 1){
                brackets += 4*f[i];
            }else if(i % 3 == 2){
                brackets += 2*f[i];
            }
        }
        aproxArea = (h*brackets)/3;
        error = Math.abs(aproxArea - this.realArea);
        System.out.println("Simpson's aproximate area: " + aproxArea);
        System.out.println("Error: " + error);
        System.out.println();
        return error;
    }else{
        System.out.println("For Sympson's rule number of intervals must be even!");
        System.out.println();
        return 0;
    }
}
```

This is a very good approximation generally. We are taking a quadratic function in every section.

Chebyshev method

```
public double Chebyshev(double[] points){
    int n = points.length;
    double f[] = new double[points.length];
    double x[] = new double[points.length];
    double t = 0;
    double sum = 0;
    double aproxArea = 0; double error = 0;
    for(int index=0, i=1; index < points.length; index++, i++){
        if(n == 2 ){
            if(i == 1)
                t = -0.577350;
            else
                t = 0.577350;
        }else if(n == 3){
            if(i == 1)
                t = -0.707107;
            else if(n == 3)
                t = 0.707107;
            else
                t = 0;
        }else if(n == 4){
            if(i == 1)
                t = -0.794654;
            else if(i == 4)
                t = 0.794654;
            else if(i == 2)
                t = -0.832498;
            else
                t = 0.832498;
        }else if(n == 5){
            if(i == 1)
                t = -0.832498;
            else if(i == 5)
                t = 0.832498;
            else if(i == 2)
                t = -0.374541;
            else if(i == 4)
                t = 0.374541;
            else
```

```
            else
                t = 0;
        }else if(n == 6){
            if(i == 1)
                t = -0.866247;
            else if(i == 6)
                t = 0.866247;
            else if(i == 2)
                t = -0.422519;
            else if(i == 5)
                t = 0.422519;
            else if(i == 3)
                t = -0.266635;
            else
                t = 0.266635;
        }else if(n == 7){
            if(i == 1)
                t = -0.883862;
            else if(i == 7)
                t = 0.883863;
            else if(i == 2)
                t = -0.529657;
            else if(i == 6)
                t = 0.529657;
            else if(i == 3)
                t = -0.323912;
            else if(i == 5)
                t = 0.323912;
            else
                t = 0;
        }
        x[index] = 1*(a+b)/2+1*(a-b)*t/2;
        //System.out.println(" a: " + a + " b: "+ b + " t: "+t + " x" + index + " = " + x[index]);
        f[index] = Math.pow(Math.E, -x[index]) * Math.sin(x[index]) * (2 * Math.cos(x[index]) - Math.sin(x[index]));
        sum += f[index];
    }
    aproxArea = (b - a)*sum / (double) n;
    error = Math.abs(aproxArea - this.realArea);
    System.out.println("Chebyshev's aproximate area: " + aproxArea);
    System.out.println("Error: " + error);
```

As we can observe, we're taking chebyshev nodes t in order to calculate an approximate solution to our problem.

Results

We will perform this methods for two cases and we will evaluate the results:

1. 3 nodes → 2 sections

```
Exact area: 1.5247425170459024
Rectangular aproximate area: 4.694833069464339
Error: 3.170090552418437

Trapezoid aproximate area: 0.40786996940531317
Error: 1.1168725476405892

Simpson's aproximate area: -1.883412143046088
Error: 3.4081546600919905

Chebyshev's aproximate area: 8.190755700539404
Error: 6.666013183493501
```

In this case, rectangular is not good at all, the error is 3 times the real area, it's not a good approximation.

For the Trapezoid approximate area the error is less, 3 times less than the real area.

For the Simpson's formula the error is even bigger than rectangular approximation, and it's also negative.

For Chebyshev's nodes the result is the worst one.

In conclusion, the little accuracy of the results is given for a little quantity of nodes. Let's analyze for more of them

2. 31 nodes → 30 sections

```
Exact area: 1.5247425170459024
Rectangular aproximate area: 1.8364660072839127
Error: 0.3117234902380104

Trapezoid aproximate area: 1.55066846727998
Error: 0.02592595023407762

Simpson's aproximate area: 1.2760568562366845
Error: 0.24868566080921783

Chebyshev's aproximate area: -1.0347532949180716
Error: 2.559495811963974
```

Now we can see an improvement in general terms with the first approximation, Chebyshev's quadrature is the worst one and trapezoid approximation is the best one.