

MERCER UNIVERSITY

SSE-636

Particle Swarm Optimization

Yoseph Abraha

Contents

Introduction.....	3
Double slit interferometer simulation	3
Double slit interferometer simulator source code	3
Sample simulation outputs.....	6
Spectrum analyzer using particle swarm optimization	9
What it does	9
Spectrum analyzer source code.....	9
Sample spectrum analyzer output	15
Indirect time log.....	Error! Bookmark not defined.

Introduction

Particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions (dubbed particles), and moving these particles around according to mathematical formulae. In this project I used this technique to find spectral components of an unknown light wave given its interference pattern. The particles here will be the spectral components and we can start the spectral population with a random population and each particle will move in a direction determined by a mathematical formula that optimizes the solution.

Double slit interferometer simulation

Interferometry makes use of the principle of superposition to combine waves in a way that will cause the result of their combination to have some meaningful property that helps us extract information about the original state of the waves. This works because when two waves with the same wavelength combine, the resulting intensity pattern is determined by the phase difference between the two waves—waves that are in phase will undergo constructive interference while waves that are out of phase will undergo destructive interference. So the pattern we will get in the measuring device will depend on the phase difference of that point's location from the two slits. There will be zero probability at some points where the phase shift is 180° and maximum probability when the phase shift is 0° .

The actual amplitude of the interference depends not only on the phase shift but also on how far the detection point is from the center using the following equation:

$$Intensity = \sin^2\left(\frac{\pi ay}{\lambda D}\right) / \left(\frac{\pi ay}{\lambda D}\right)^2$$

Where a is separation between slits, y distance of the point from center, λ wavelength and D distance between the slits and detection device.

Double slit interferometer simulator source code

Below is a simulator class that gives us an interference pattern for a given double slit interference configuration and wavelength, using the above discussed concepts.

```
class DoubleslitInterferometer
{
    //distance between the apertures
    double distancebetweenSlits;

    //distance from the apertures to the detection board
    double distancetomeasuringDevice;
    //double approxwidthofInterferencePattern=0.1;

    //probability amplitude as array
    double[] outputAsarray = new double[100];
    //points of detection(sampling points)
    double[] outputPoints = new double[100];
}
```

```

//separation between sampling points
double samplingSeparation = 1E-3;
//wavelength of light
double wavelength = 0;

//constructor takes two parameter that determines the interferometer configuration
public DoubleslitInterferometer(double distancebetweenSlits, double
distancetomeasuringDevice)
{
    this.distancebetweenSlits = distancebetweenSlits;
    this.distancetomeasuringDevice = distancetomeasuringDevice;
    //this.approxwidthofInterferencePattern = 700E-9 * distancetomeasuringDevice /
distancebetweenSlits;
    initsamplingPoints();
}

//returns the probability density of interference for a light of certain wavelength in the
configuration set in the constructor
public double[] getInterferencePattern(double wavelength)
{
    this.wavelength = wavelength;
    double[] distancefromSlitone=new double[outputAsarray.Length];
    double[] distancefromSlittwo= new double[outputAsarray.Length];
    for (int i = 0; i < outputPoints.Length; i++)
    {
        distancefromSlitone[i] = Math.Sqrt(Math.Pow(distancetomeasuringDevice, 2) +
Math.Pow((distancebetweenSlits-outputPoints[i]),2));
    }

    for (int i = 0; i < outputPoints.Length; i++)
    {
        distancefromSlittwo[i] = Math.Sqrt(Math.Pow(distancetomeasuringDevice, 2) +
Math.Pow((-distancebetweenSlits - outputPoints[i]), 2));
    }

    outputAsarray = getprobabilityAmplitudes(distancefromSlitone, distancefromSlittwo,
wavelength);
    return outputAsarray;
}

//initializes the sampling points
private void initsamplingPoints()
{
    double maxwidth = (samplingSeparation*outputPoints.Length)/2;
    int points = outputAsarray.Length;
    for (int i = 0; i < points; i++)

```

```

    {
        outputPoints[i] = maxwidth - i * samplingSeparation;
    }
}

//returns interference pattern based on path deferences and wave length
private double[] getprobabilityAmplitudes(double[] path1, double[] path2, double
wavelength)
{
    double[] phaseDifferences = new double[outputPoints.Length];
    double[] relativeProbabilities = new double[outputPoints.Length];
    for (int i = 0; i < outputPoints.Length; i++)
    {
        phaseDifferences[i] = 2*Math.PI*((Math.Abs(path1[i] - path2[i])/wavelength) % 1);
    }
    for (int i = 0; i < outputPoints.Length; i++)
    {
        relativeProbabilities[i] = Math.Pow(Math.Cos(phaseDifferences[i]), 2);
    }
    relativeProbabilities = getNormalizedprobability(relativeProbabilities);
    return relativeProbabilities;
}

//normalizes a given probability distribution and manipulates amplitudes
private double[] getNormalizedprobability(double[] relativeProbability)
{
    double sum = 0;
    double intensity = 0;
    double arg = 0;
    double[] relativeProbabilities = new double[relativeProbability.Length];
    for (int i = 0; i < outputPoints.Length; i++)
    {
        arg = Math.PI * distancebetweenSlits * outputPoints[i] / (wavelength *
distancetomeasuringDevice);
        if (arg == 0)
            intensity = 1;
        else
            intensity = Math.Pow((Math.Sin(arg)/ arg), 2);
        relativeProbabilities[i] = relativeProbability[i] * intensity;
    }

    foreach (double d in relativeProbabilities)
    {
        sum += d;
    }
    for (int i = 0; i < outputPoints.Length; i++)

```

```

        {
            relativeProbabilities[i] /= sum;
        }
        return relativeProbabilities;
    }
}

class Program
{
    static void Main(string[] args)
    {
        //create an double slit interferometer with 50 micro meter separation between slits and 25
        centimeter length
        var interferometer = new DoubleslitInterferometer(100E-6, 100E-2);

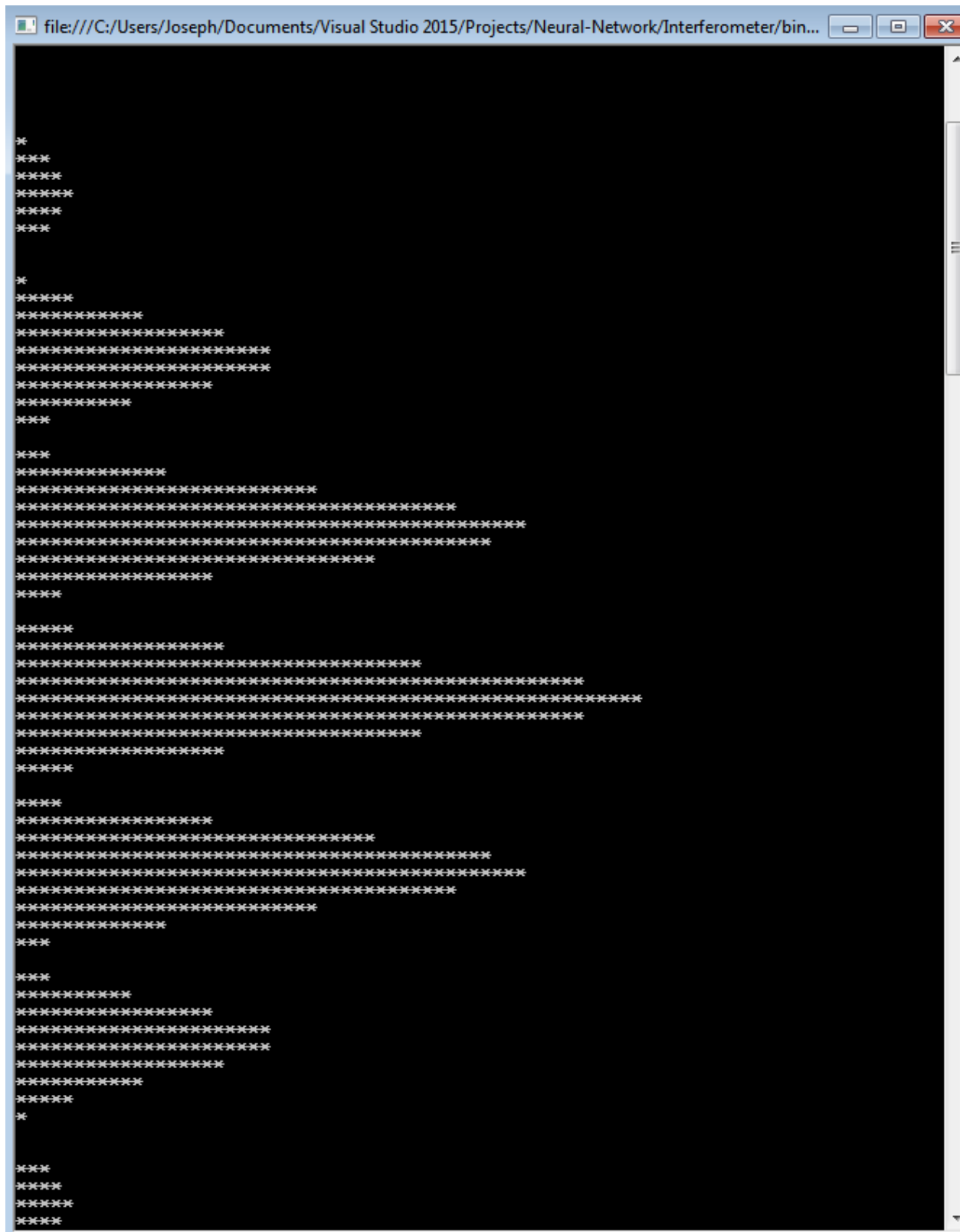
        //get the expected intensity distribution for a red 700 nano meter wavelength light (violet
        light)
        double[] pattern = interferometer.getInterferencePattern(700E-9);
        foreach (double d in pattern)
        {
            printAsterix(d);
        }
        Console.ReadKey();
    }

    //prints asterix based on amplitude at a given point
    private static void printAsterix(double asterixNumber)
    {
        //since the amplitudes are small I scalled them up by 1000 so the interference pattern is
        visible
        int i = (int)(asterixNumber * 1000);
        string asterixes = "";
        for (int j = 0; j < i; j++)
        {
            asterixes+="*";
        }
        Console.WriteLine(asterixes);
    }
}

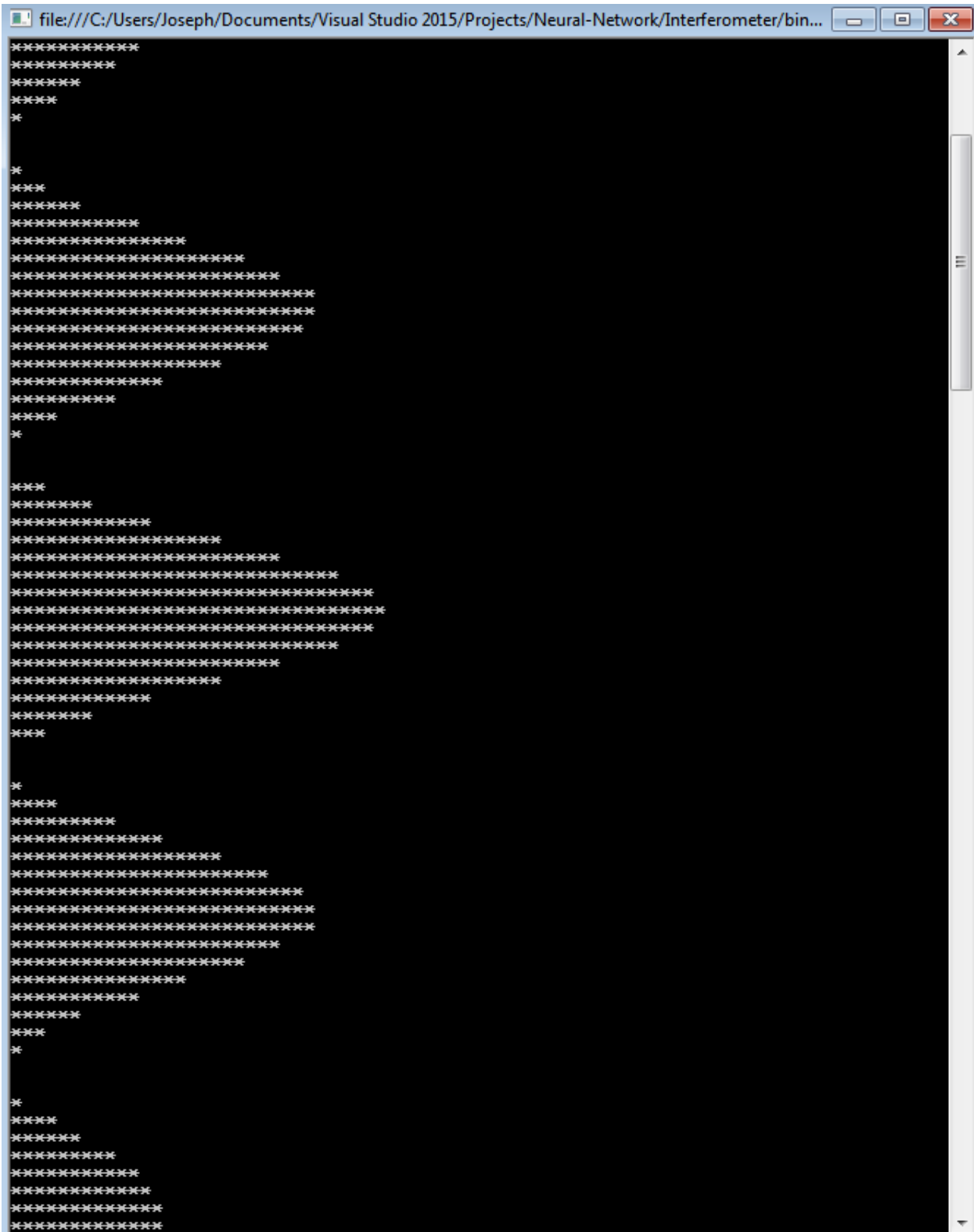
```

Sample simulation outputs

Below is a simulated expected interference pattern using violet color(400nm wavelength) for an interferometer with 100μm separation between slits:



Below is another interference pattern this time for red light(700 nm wavelength):



Spectrum analyzer using particle swarm optimization

What it does

A spectrum analyzer analyzes frequency spectrum of a given signal. It resolves a given signal into discrete components that sum up to give the total signal and also calculates the percentage contribution of each to the total signal. This specific spectral analyzer will get interference pattern of an unknown incoming light and will be tasked to determine the percentages of the different discrete spectral light components that make up the unknown light, using particle swarm optimization algorithm.

Spectrum analyzer source code

Below is the source code for the spectrum analyzer:

```
class VisibleSpectrum
{
    DoubleslitInterferometer interferometer;
    double[] interferencePattern;
    List<SpectralComponent> allComponents;
    double spectralDisperse = 50E-9;

    public VisibleSpectrum(DoubleslitInterferometer interferometer, double[]
interferencePattern)
    {
        this.interferometer = interferometer;
        this.interferencePattern = interferencePattern;
        allComponents= new List<SpectralComponent>();
        initSpectrum();
    }

    public void normalizeComponents()
    {
        double sum = 0;
        foreach (SpectralComponent component in allComponents)
        {
            sum += component.getpercentComponent();
        }

        foreach (SpectralComponent component in allComponents)
        {
            component.setpercentComponent((component.getpercentComponent()) /sum);
        }
    }

    private void initSpectrum()
    {
        for (double wavelength = 400E-9; wavelength <= 700E-9; wavelength +=
spectralDisperse)
        {
```

```

        allComponents.Add(new SpectralComponent(wavelength, this));
    }
    double initialComponent = allComponents.Count;
    initialComponent = 1 / initialComponent;
    foreach (SpectralComponent component in allComponents)
    {
        component.setpercentComponent(initialComponent);
    }
}

public List<SpectralComponent> getFitpopulation()
{
    for (int i = 0; i < 100; i++)
    {
        evolve();
    }
    return allComponents;
}

public double getFitness()
{
    double fitness = 0;
    double temp = 0;
    double[] output = new double[interferencePattern.Length];
    for (int i = 0; i < output.Length; i++)
    {
        foreach (SpectralComponent component in allComponents)
        {
            temp += (component.getpercentComponent() *
interferometer.getInterferencePattern(component.getWavelength())[i]);
        }
        output[i] = temp;
        temp = 0;
    }
    for (int i = 0; i < output.Length; i++)
    {
        fitness += (Math.Pow((interferencePattern[i] - output[i]), 2));
    }
    if (fitness == 0)
        return 10E10;
    fitness = 1 / fitness;
    return fitness;
}

private void evolve()
{

```

```

        foreach (SpectralComponent component in allComponents)
        {
            component.tryOptimization();
        }
    }
}

class SpectralComponent
{
    double wavelength;
    double percentComponent;
    double optimizationSteps = 0.01;
    VisibleSpectrum spectrum;
    public SpectralComponent(double wavelength, VisibleSpectrum spectrum)
    {
        this.spectrum = spectrum;
        this.wavelength = wavelength;
    }

    public void setpercentComponent(double percentComponent)
    {
        this.percentComponent = percentComponent;
    }
    public double getpercentComponent()
    {
        return this.percentComponent;
    }
    public double getWavelength()
    {
        return wavelength;
    }
    public void tryOptimization()
    {
        double initialpercentComponent = this.percentComponent;
        double initialFitness = spectrum.getFitness();
        this.percentComponent += optimizationSteps;
        spectrum.normalizeComponents();
        double newFitness = spectrum.getFitness();
        if (newFitness > initialFitness)
            return;
        this.percentComponent = Math.Abs(percentComponent-2 * optimizationSteps);
        spectrum.normalizeComponents();
        newFitness = spectrum.getFitness();
        if (newFitness > initialFitness)
            return;
        this.percentComponent = initialpercentComponent;
    }
}

```

```

        spectrum.normalizeComponents();
    }
}

class DoubleslitInterferometer
{
    //distance between the apertures
    double distancebetweenSlits;

    //distance from the apertures to the detection board
    double distancetomeasuringDevice;
    //double approxwidthofInterferencePattern=0.1;

    //probability amplitude as array
    double[] outputAsarray = new double[100];
    //points of detection(sampling points)
    double[] outputPoints = new double[100];
    //separation between sampling points
    double samplingSeparation = 1E-3;
    //wavelength of light
    double wavelength = 0;

    //constructor takes two parameter that determines the interferometer configuration
    public DoubleslitInterferometer(double distancebetweenSlits, double
distancetomeasuringDevice)
    {
        this.distancebetweenSlits = distancebetweenSlits;
        this.distancetomeasuringDevice = distancetomeasuringDevice;
        //this.approxwidthofInterferencePattern = 700E-9 * distancetomeasuringDevice /
distancebetweenSlits;
        initsamplingPoints();
    }

    //returns the probability density of interference for a light of certain wavelength in the
configuration set in the constructor
    public double[] getInterferencePattern(double wavelength)
    {
        this.wavelength = wavelength;
        double[] distancefromSlitone=new double[outputAsarray.Length];
        double[] distancefromSlittwo= new double[outputAsarray.Length];
        for (int i = 0; i < outputPoints.Length; i++)
        {
            distancefromSlitone[i] = Math.Sqrt(Math.Pow(distancetomeasuringDevice, 2) +
Math.Pow((distancebetweenSlits-outputPoints[i]),2));
        }
    }
}

```

```

        for (int i = 0; i < outputPoints.Length; i++)
        {
            distancefromSlittwo[i] = Math.Sqrt(Math.Pow(distancefrommeasuringDevice, 2) +
Math.Pow((-distancebetweenSlits - outputPoints[i]), 2));
        }

        outputAsarray = getprobabilityAmplitudes(distancefromSlitone, distancefromSlittwo,
wavelength);
        return outputAsarray;
    }

//initializes the sampling points
private void initsamplingPoints()
{
    double maxwidth = (samplingSeparation*outputPoints.Length)/2;
    int points = outputAsarray.Length;
    for (int i = 0; i < points; i++)
    {
        outputPoints[i] = maxwidth - i * samplingSeparation;
    }
}

//returns interference pattern based on path deferences and wave length
private double[] getprobabilityAmplitudes(double[] path1, double[] path2, double
wavelength)
{
    double[] phaseDifferences = new double[outputPoints.Length];
    double[] relativeProbabilities = new double[outputPoints.Length];
    for (int i = 0; i < outputPoints.Length; i++)
    {
        phaseDifferences[i] = 2*Math.PI*((Math.Abs(path1[i] - path2[i])/wavelength) % 1);
    }
    for (int i = 0; i < outputPoints.Length; i++)
    {
        relativeProbabilities[i] = Math.Pow(Math.Cos(phaseDifferences[i]), 2);
    }
    relativeProbabilities = getNormalizedprobability(relativeProbabilities);
    return relativeProbabilities;
}

//normalizes a given probability distribution and manipulates amplitudes
private double[] getNormalizedprobability(double[] relativeProbability)
{
    double sum = 0;
    double intensity = 0;
    double arg = 0;

```

```

        double[] relativeProbabilities = new double[relativeProbability.Length];
        for (int i = 0; i < outputPoints.Length; i++)
        {
            arg = Math.PI * distancebetweenSlits * outputPoints[i] / (wavelength *
distancetomeasuringDevice);
            if (arg == 0)
                intensity = 1;
            else
                intensity = Math.Pow((Math.Sin(arg)/ arg), 2);
            relativeProbabilities[i] = relativeProbability[i] * intensity;
        }

        foreach (double d in relativeProbabilities)
        {
            sum += d;
        }
        for (int i = 0; i < outputPoints.Length; i++)
        {
            relativeProbabilities[i] /= sum;
        }
        return relativeProbabilities;
    }
}

class SpectrumAnalyzer
{
    private DoubleslitInterferometer interferometer;
    private VisibleSpectrum spectrum;
    public SpectrumAnalyzer(DoubleslitInterferometer interferometer)
    {
        this.interferometer = interferometer;
    }

    public Dictionary<double, double> getspectralComponents(double[] interferencePattern)
    {
        this.spectrum = new VisibleSpectrum(interferometer, interferencePattern);
        Dictionary<double, double> spectralWeights = new Dictionary<double, double>();
        var listofSpectrum = spectrum.getFitpopulation();
        foreach (SpectralComponent component in listofSpectrum)
        {
            spectralWeights.Add(component.getWavelength(),
component.getpercentComponent());
        }
        return spectralWeights;
    }
}

```

Sample spectrum analyzer output

Below is a console program that passes to the spectral analyzer the pattern got from interference simulator of a 500nm wavelength light and tests the analyzer to determine the input wavelengths.

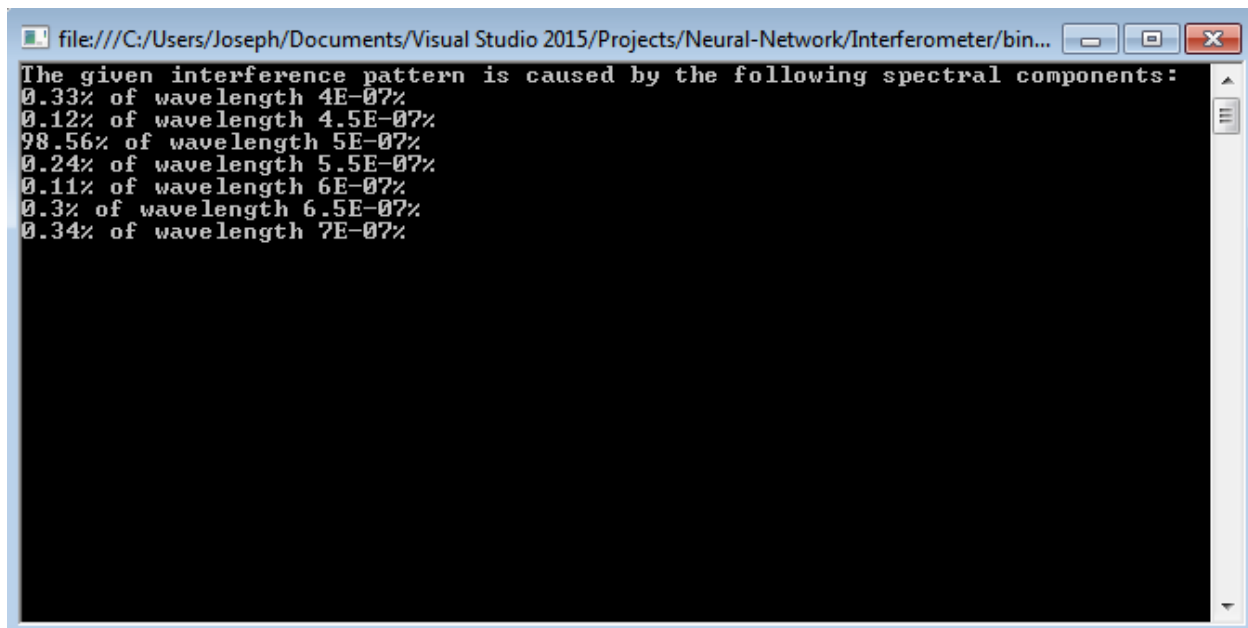
```
class Program
{
    static void Main(string[] args)
    {
        //create an double slit interferometer with 50 micro meter separation between slits and 25
        centimeter length
        var interferometer = new DoubleslitInterferometer(100E-6, 1000E-2);

        //get the expected intensity distribution for a given wavelength light
        double[] pattern = interferometer.getInterferencePattern(500E-9);

        SpectrumAnalyzer spectrumAnalyzer = new SpectrumAnalyzer(interferometer);
        var dict = spectrumAnalyzer.getspectralComponents(pattern);

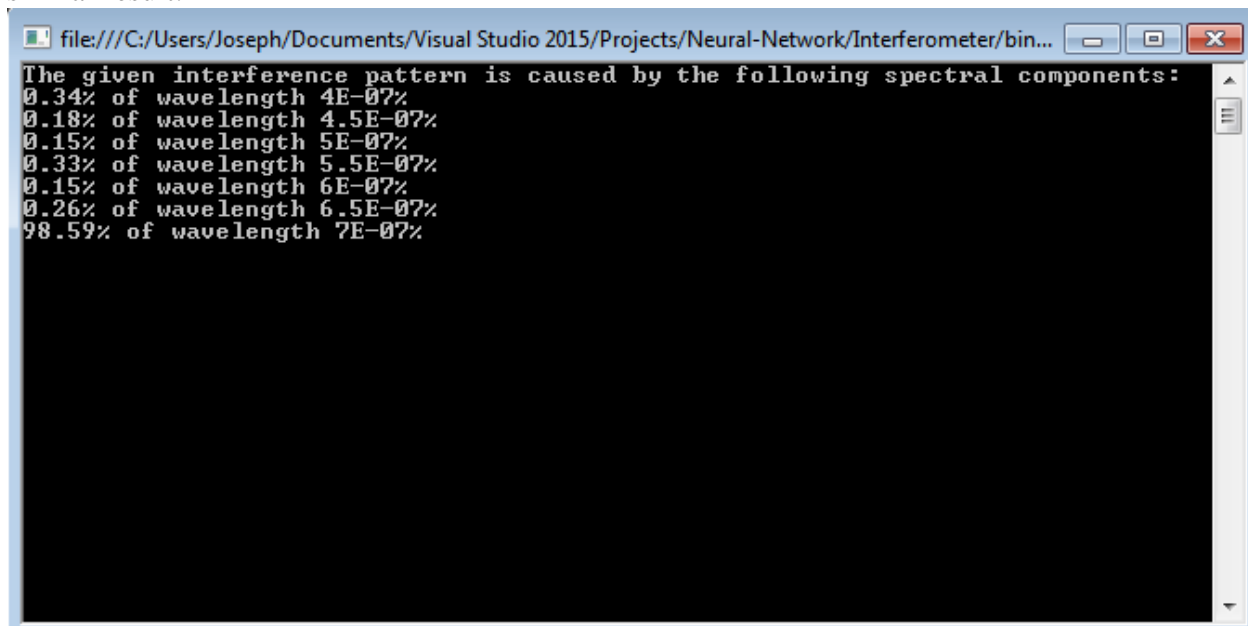
        Console.WriteLine("The given interference pattern is caused by the following spectral
        components:");
        foreach (KeyValuePair<double, double> t in dict)
        {
            Console.WriteLine(String.Format("{1}% of wavelength {0}%", t.Key,
            Math.Round(t.Value * 100, 2)));
        }
        Console.ReadKey();
    }
}
```

Below is the output which says 98.56% of the signal is light of wavelength 500nm, which is almost correct.



```
file:///C:/Users/Joseph/Documents/Visual Studio 2015/Projects/Neural-Network/Interferometer/bin...
The given interference pattern is caused by the following spectral components:
0.33% of wavelength 4E-07%
0.12% of wavelength 4.5E-07%
98.56% of wavelength 5E-07%
0.24% of wavelength 5.5E-07%
0.11% of wavelength 6E-07%
0.3% of wavelength 6.5E-07%
0.34% of wavelength 7E-07%
```

Below I tested using interference data from 700nm wavelength interference and it returned similar result.



```
file:///C:/Users/Joseph/Documents/Visual Studio 2015/Projects/Neural-Network/Interferometer/bin...
The given interference pattern is caused by the following spectral components:
0.34% of wavelength 4E-07%
0.18% of wavelength 4.5E-07%
0.15% of wavelength 5E-07%
0.33% of wavelength 5.5E-07%
0.15% of wavelength 6E-07%
0.26% of wavelength 6.5E-07%
98.59% of wavelength 7E-07%
```

I also passed it interference data that resulted from 50% 500nm & 50% 700nm lights(a mixed light) and below is the console code and output.

```
class Program
{
    static void Main(string[] args)
    {
        //create an double slit interferometer with 50 micro meter separation between slits and 25
        centimeter length
    }
}
```



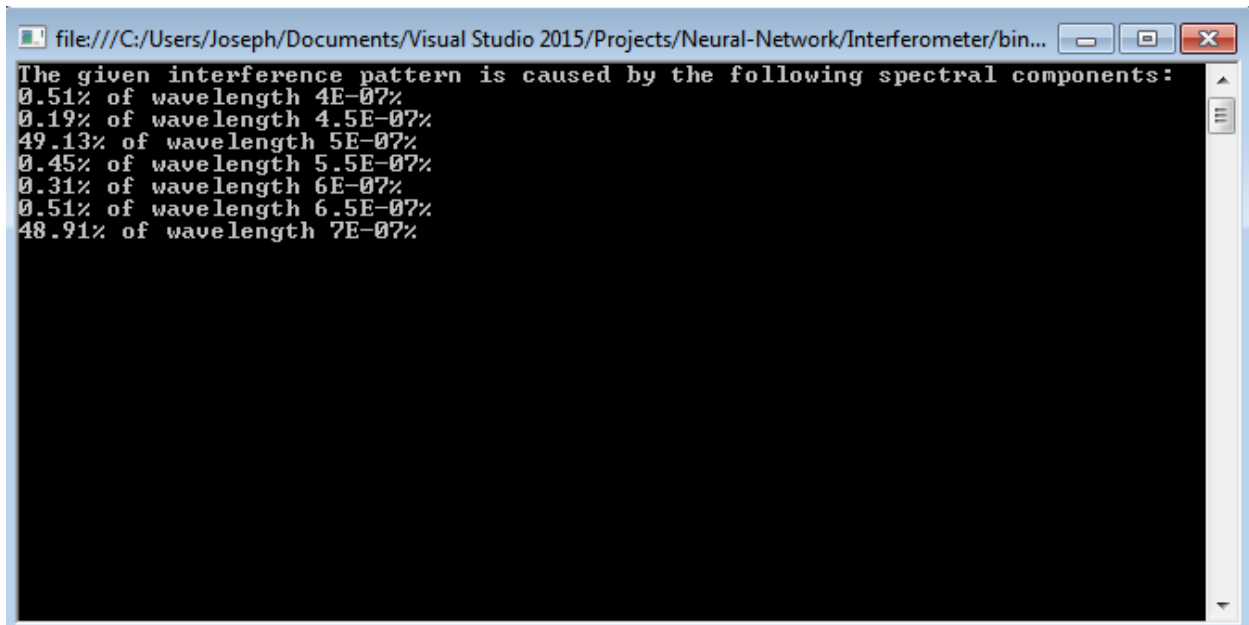
```

var interferometer = new DoubleslitInterferometer(100E-6, 1000E-2);

//get the expected intensity distribution for a given wavelength light
double[] pattern1 = interferometer.getInterferencePattern(700E-9);
double[] pattern2 = interferometer.getInterferencePattern(500E-9);
double[] pattern = new double[pattern1.Length];
for (int i = 0; i < pattern1.Length; i++)
{
    pattern[i] = (0.5 * pattern1[i] + 0.5 * pattern2[i]);
}
SpectrumAnalyzer spectrumAnalyzer = new SpectrumAnalyzer(interferometer);
var dict = spectrumAnalyzer.getspectralComponents(pattern);

Console.WriteLine("The given interference pattern is caused by the following spectral
components:");
foreach (KeyValuePair<double, double> t in dict)
{
    Console.WriteLine(String.Format("{1}% of wavelength {0}%", t.Key,
Math.Round(t.Value * 100, 2)));
}
Console.ReadKey();
}
}

```

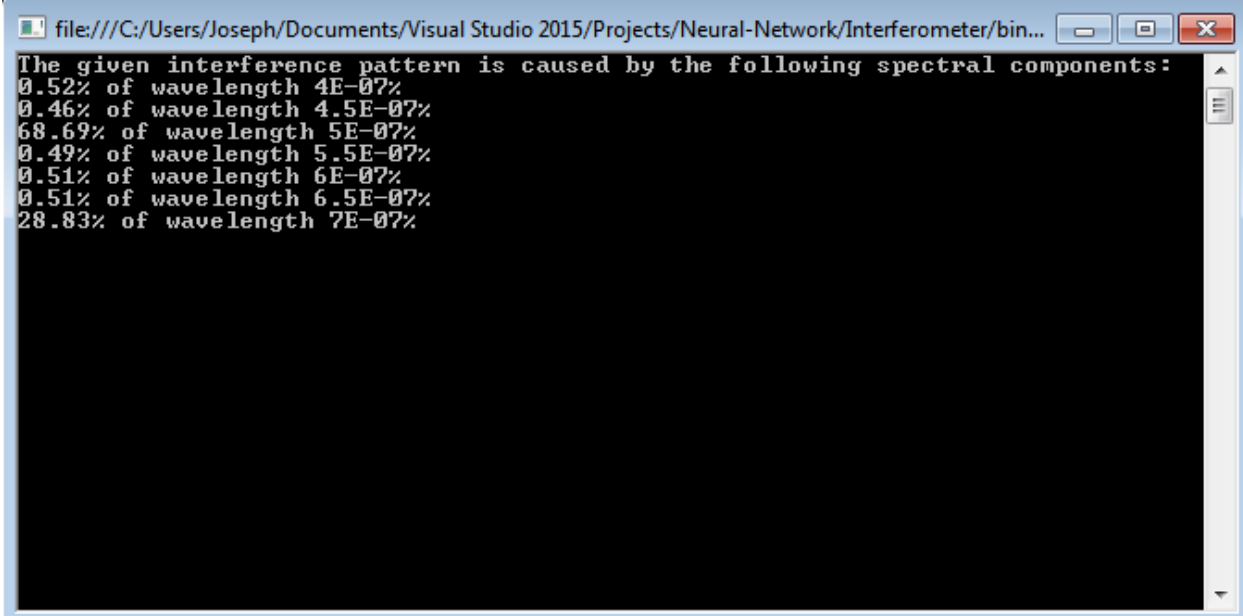


```

file:///C:/Users/Joseph/Documents/Visual Studio 2015/Projects/Neural-Network/Interferometer/bin...
The given interference pattern is caused by the following spectral components:
0.51% of wavelength 4E-07%
0.19% of wavelength 4.5E-07%
49.13% of wavelength 5E-07%
0.45% of wavelength 5.5E-07%
0.31% of wavelength 6E-07%
0.51% of wavelength 6.5E-07%
48.91% of wavelength 7E-07%

```

Finally below is another output where the program successfully determined a 30 & 70% mixed light's components.



```
file:///C:/Users/Joseph/Documents/Visual Studio 2015/Projects/Neural-Network/Interferometer/bin...
The given interference pattern is caused by the following spectral components:
0.52% of wavelength 4E-07%
0.46% of wavelength 4.5E-07%
68.69% of wavelength 5E-07%
0.49% of wavelength 5.5E-07%
0.51% of wavelength 6E-07%
0.51% of wavelength 6.5E-07%
28.83% of wavelength 7E-07%
```