Table of Contents

[1. Project description 4](#_Toc341877361)

[1.1 Introduction 4](#_Toc341877362)

[1.2 Project rationale 4](#_Toc341877363)

[1.3 Design approach 4](#_Toc341877364)

[1.4 Layout of the project 5](#_Toc341877365)

[2. Finite elements analysis 6](#_Toc341877366)

[2.1 Common scheme 6](#_Toc341877367)

[2.2 Problem statement 8](#_Toc341877368)

[2.3 Variational functional of the problem 10](#_Toc341877369)

[2.4 Mixed finite elements method 12](#_Toc341877370)

[3. Implementation 15](#_Toc341877371)

[3.1. Environment 15](#_Toc341877372)

[3.2. Implementation 15](#_Toc341877373)

[3.3. GUI 16](#_Toc341877374)

[4. Verification and testing 19](#_Toc341877375)

[**Test 1.** 19](#_Toc341877376)

[Test 2. 21](#_Toc341877377)

[Test 3. 22](#_Toc341877378)

[**Test 4.** 25](#_Toc341877379)

[**Test 5.** 27](#_Toc341877380)

[Test 6. 29](#_Toc341877381)

[5. Conclusion 31](#_Toc341877382)

[References 32](#_Toc341877383)

[6. Appendices 33](#_Toc341877384)

[6.1. Complex number 33](#_Toc341877385)

[6.2. Matrixes 34](#_Toc341877386)

[6.3. Characteristics 39](#_Toc341877387)

[6.4. Example of saved characteristics. 42](#_Toc341877388)

Project brief

Supervisor: Prof. Gerald Farrell

Phone +353-1-4024577

Mobile:

e-mail: gerald.farrell@dit.ie

School: Electronic and Communication Engineering

Title: Mixed Finite Element Analysis of Real and Complex Magnetodielectrical Waveguide Waves

Objective: To create an application which allows analysing dispersion characteristics of magnetodielectrical waveguide waves

Prerequisites:

Description: This project was completed as a part of collaboration program between DIT(Dublin Institute of Technology) and MSTU MIREA(Moscow State Technical University Institute of Radioengineering Electronics and Automation).

1. Project description
   1. Introduction

The purpose of this project is to implement an application which allows calculating dispersion characteristics and critical values of limited decay of multilayer waveguide waves.

This application is implemented with combining of resources of Visual Studio 2010 and Matlab 7.11. These two systems are powerful enough to do calculations of this project and suitable to work with complicated structures.

The interface, suitable for the user, that is easy to use must be implemented in this project. The system should work in combination with Matlab 7.0+. The user should be able to use system without special preparation.

**Background, what are these devices what are their applications, why simulation, why a gui?**

* 1. Project rationale

Waveguides are one of the means that are used to transmit the waves. According to critical wavelength, which is twice bigger than waveguide’s diameter, they are suitable for microwaves.

Power loss is small enough relatively to other types of transmission lines – that is their obvious advantage.

For more efficient usage of waveguide is better to improve its capacity. To do that waveguides are created with a variable index of refraction. To solve the problem of synthesis of multilayer waveguide it is necessary to calculate the dispersion curves with given refractive index. From mathematical point of view, the problem lies in the solution of Maxwell’s equations in a cylinder with variable index of refraction, which varies along the radios of cylinder.

During solving the problem can appear non-physical solutions and finite elements method helps to get rid of them.

* 1. Design approach

Application consists of 3 parts:

* Entering waveguides characteristics
* Calculation core
* Work with results

All characteristics are entered on one page, validated and used in calculation core.

Calculation core includes different levels of calculations from the simplest to quite complicated. Final calculations lead to computation of dispersion characteristics of waveguide or critical conditions (values) of limited decay according to users options.

Finally, the results are submitted in graphical form with capability of saving in numerical way.

Calculation core (CC) works independently with GUI. This was reached by separation CC and GUI in different threads. That allows working with GUI all the time of calculations and prevents absence of response from application.

* 1. Layout of the project

1. Finite elements analysis
   1. Common scheme

Finite elements analysis is based on idea of approximation continuous function with discrete model, which is based on a set of piecewise continuous functions defined on a finite number of subdomains called finite elements. On every element unknown function is approximated by test function (as general rule polynomial) and boundary conditions coincide with the boundary conditions of initial problem.

We consider usage of finite elements analysis on example of spectral problem in domain Ω:

A (1.1)



on domain boundary (1.2)



Then we reduce the original problem (1.1), (1.2) to a problem in the variational formulation.

To do this, we multiply on right on and take the scalar product of left and right parts of equation, finaly we have:



(A) (1.3)



on domain boundary (1.4)



So we have the problem equivalent to problem (1.1), (1.2). It is variational formulation of spectral problem and is solved in this formulation. Equality of differential equations and variational problems forms the basis of the choice of the computational scheme. Differential equations might be approximated with discrete system, using finite differences, and variational functional can be minimized on finite-dimensional space as in finite elements analysis.

We search the solution of the problem (1.3), (1.4) in form of expansion in the system of basic functions:



Here ci – coefficients of expansion, Ni(G) – basic functions.

Substitute (1.5) in (1.3) and set , then we have:



So, the problem transforms into system of algebraic equations, which looks as matrix:



where λ is eigenvalue and elements of matrices B and C are:

= (A) , = () (1.8)



Finally we have generalised problem of eigenvalues. Now we should define basic functions. In finite element analysis polynomials of different orders are used as basic hosts, they are called finite functions, which are not null in finite domains. In one-dimensional case are used equal segments, in two-dimensional – triangles and rectangles.

Further, we consider one-dimensional scalar problem on segment [a,b]. We set uniform grid {xi}: i = 0,1,..,M, x0 = a, xM = b, xi = ih, h – is grid spacing, M – number of finite elements. As finite elements are meant equal segments of which consists the original segment. Finite functions are defined be equation:

(1.9)



According to attachment x to segment [a,b] and in (1.9) t = (x-xi)/h, we have:

=

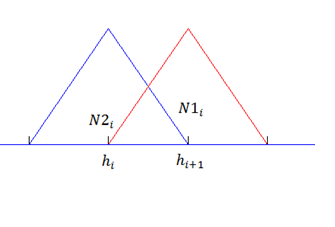


=



In the aggregate these functions are designated as N, which includes both basic functions on one finite element.

Graphically it looks like:



Here are the first-order functions. In the same way are defined polynomials of second, third, etc. order. This will improve accuracy of the method, but at the same time matrices become less sparse and technical implementation of elements of higher order will be more difficult. With modern computers accuracy can be reached on the elements of the first order.

* 1. Problem statement

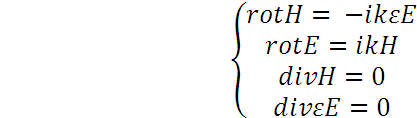
We consider cylindrical waveguide with circular cross-section Ω with unitary radius r=1. In one point on its axis we set cylindrical coordinating system, axis Oz goes along cylinder axis. Let the waveguide be filled with material with characteristics:



ε(r) is piecewise continuous in Ω, the waveguide walls are perfectly conducting.

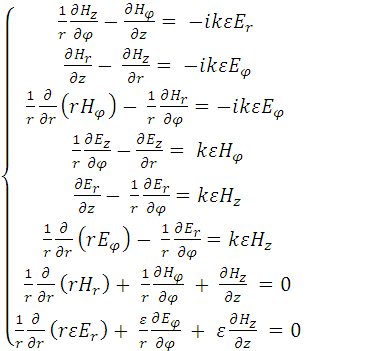
Electromagnetic field inside the waveguide is described by a system of eight Maxwell’s equations for 6 unknowns.

(2.1)



We expand the equation for rotor and rewrite them in cylindrical coordinate system:

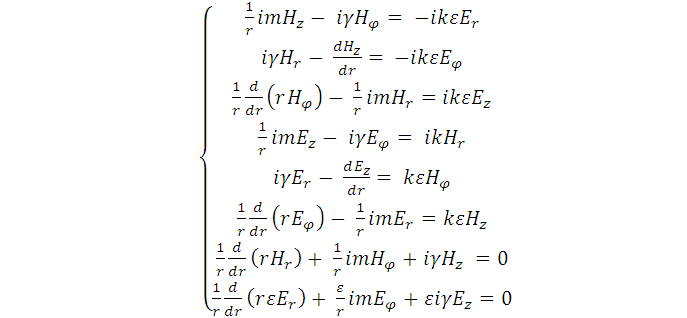
(2.2)



The solution will be in form of normal waves – functions that depend on r, z and φ:

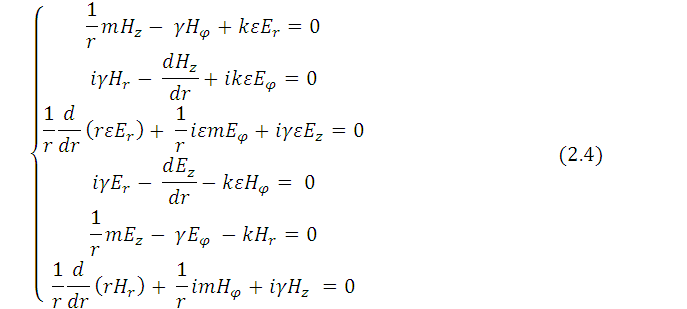
E, H = E(r)eiγz+imφ.

Substituting these normal waves in (2.2), reduce exponential factor – and we come to the problem of finding eigenvalues on segment [0; 1], where γ is eigenvalue:



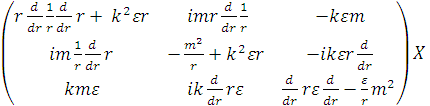
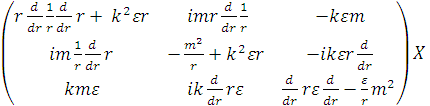
(2.3)

To solve the problem we choose 6 equations:



Then, we set X = (Hr, Hφ, Ez)T = (H┴, Ez)T, Y = (εEr, εEφ, Hz)T = (εE┴, Hz)T

If we substitute Y from first 3 equations (2.4) in last 3, we have the problem of finding eigenvalues:



(2.5)

X belongs to the set of vectors from C∞[0,1] and satisfies boundary conditions:

Hr(0) = 0 and Ez(0) = 0

Hr(1) = 0 and Ez(1) = 0,



and to Maxwell’s equation:



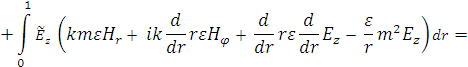
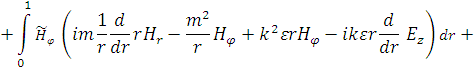
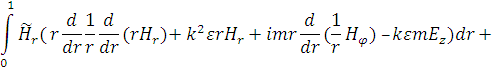
and (in case of distontinued ε) the conditions of conjugation are specified:

s = s = 0, s = s = 0

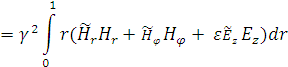


* 1. Variational functional of the problem

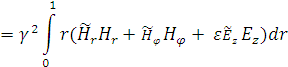
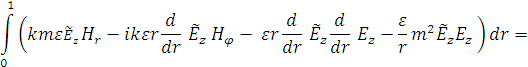
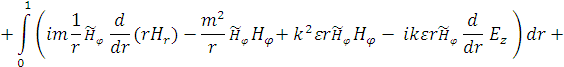
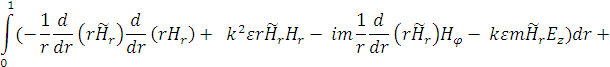
We write down the variational functional (weak formulation) for the original problem. To do this, we multiply (2.5) on the left by arbitrary vector = () and integrate over r from 0 to 1. Finally, we have:



(2.6)



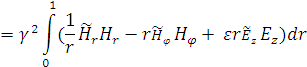
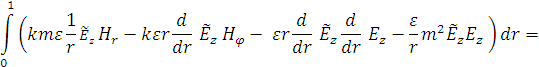
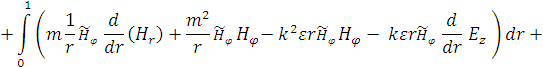
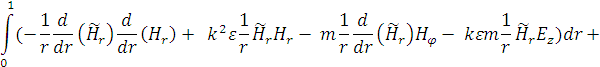
Using boundary conditions, we integrate (2.6) by parts:



We make the following changes to simplify the calculations in the future, as well as get rid of imaginary one:

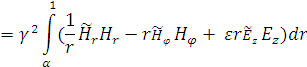
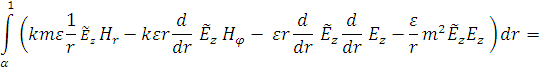
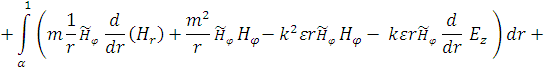
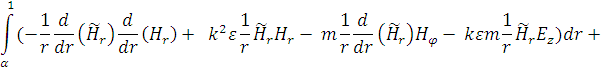


Finally, we have variational functional with complex numbers:



(2.7)

The problem (2.7) has peculiarity in 0. At integration appears indeterminacy – natural logarithm of zero. To avoid it, we put on cylinder axis thin conducting cylinder with radius α and will integrate from α:



(2.8)

Tend α to zero, then, by theorem of Samarsky, eigenvalues of the problem (2.8) will tend to eigenvalues of original problem (2.7).

The final problem is to find eigenvalues γ – the propagation constant – and to construct dispersion curves – dependence of propagation constant on the wavenumber k.

* 1. Mixed finite elements method

The problem has infinite core:

X = ()



where φ is arbitrary function.

While using standard finite elements method operators core approximates in the wrong way, that causes emergence of non-physical solutions, “spirits” of specter, which locate between genuine values – so we can’t tell the difference. Mixed finite elements analysis helps to avoid it. Method consists of approximation components of vector X with polynomials of different order. In our problem we will approximate Hr and Ez by continuous polynomials of first order and Hφ by discontinuous polynomials of zero order.

We set on segment [α, 1] uniform grid {xi}: i = 0,1,..,n, x0 = a, xn = b, xi = ih, h – is grid spacing, n – number of finite elements.

We assume permittivity to be constant on every finite element. Further, we expend in the basic functions every component of vectors X and

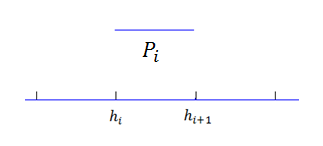
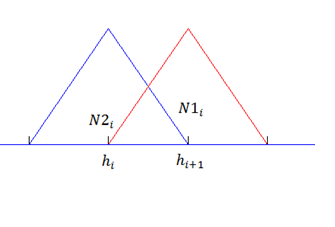


Now we have changes:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

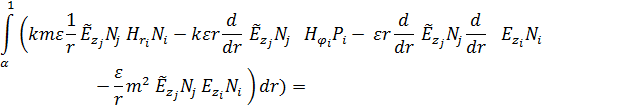
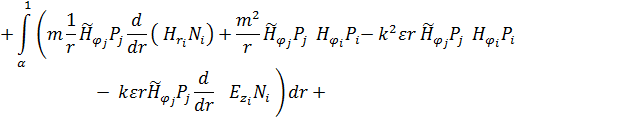
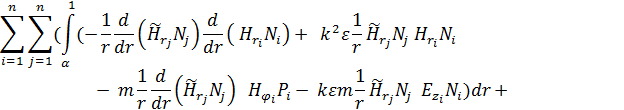
Where Ni = θN1i+σN2i on segment [hi, hi+1] (on finite element with number i).

Graphically it looks as:

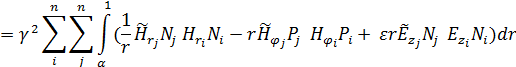


Finite element analysis allows approximate operators core, which tends into zero state, multiplicity of which is about one third of dimension of the matrix eigenvalue problem.

Applying changes, we rewrite variational functional in this way:



(2.9)



In this way have generalized problem of eigenvalues:

AX = γ2 BX

where A and B are matrices of expansion coefficients of X in the basic functions.

1. Implementation

## 3.1. Environment

For implementation was used Visual Studio 2010 with .Net 4.0 Framework and Matlab 7.11. Visual Studio allows using Matlab functions as dll to main project. WHY

The language of the project is C#. This language if powerful and flexible enough for creating wide variety of tools, multilevel hierarchy and simple linking between parts of project. WHY

## 3.2. Implementation

Characteristics which are to be presented for the waveguide are dispersion characteristics. They demonstrate wave propagation in waveguide. More important is to find critical conditions of limited decay. They show limits of waves decay in waveguides and conditions of existence complex waves in multilayer waveguides. Critical values are all the values between conditions. Within the curve of critical values waves decay.

Program consists of several modules which include separately program calculating core and GUI.

In calculating core are modules of:

* Complex numbers with operations (arithmetic, sorting, conversion to decimal numbers and strings)
* Matrices with operations (matrix arithmetic, eigenvalues)
* Final calculations of propagation constant

Source code of main functions of the core are in appendices.

Calculations go according to finite elements analysis. First we should form matrices of expansion coefficients (A and B). Whatever we calculate – dispersion characteristics or conditions of limited decay – we have to compute generalized eigenvalues of A and B. The best way to find them was using m-function from Matlab libraries: e = eig(A,B). This function uses following algorithms and functions [1]:

* Reduction to Hessenberg matrix with orthogonal conversion
* Memorizing all conversions using “orthan” function
* QR-algorithm of Francis and Cublanovskaya for conversion to upper mold Hessenberg matrix
* QZ-algorithm of finding eigenvalues

Eigenvalues are sorted and we make selection to compose dispersion characteristics (app. 6.3). Special selection is made for conditions of limited decay.

Final phase of application is plot assay. On charts you can see the results, detail them and save in both numeric and graphic ways. FORMAT

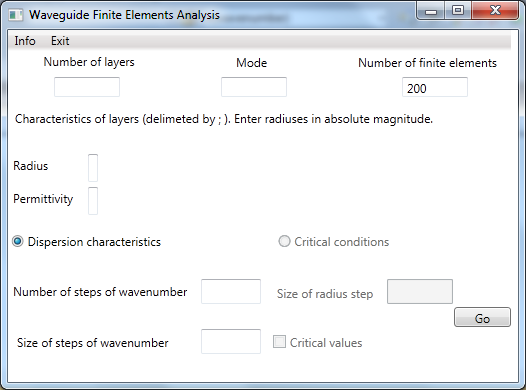
## 3.3. GUI

GUI consists of three windows.

Introducing window (pic.1) allows user to enter all data needed for calculations and get information about program usage and system requirements.

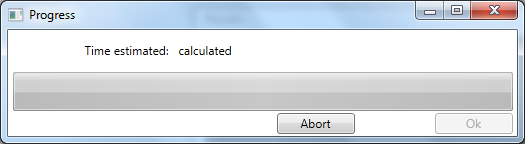
Progress-page shows time estimated for calculations and percentage of completed calculations (pic. 2-3). Calculations works in parallel thread with GUI – that allows user interact with program at the same time.

When calculations are finished appears window with charts containing chosen values: dispersion characteristics or critical conditions/values of limited decay (pic. 4).

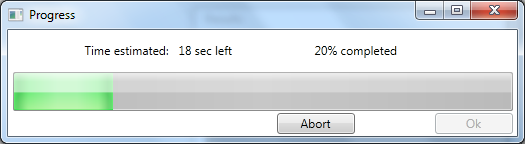


Picture 1. Introducing window.

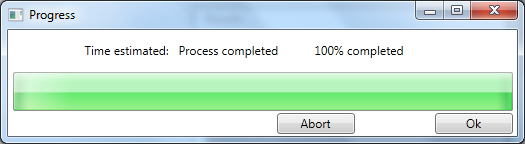
Every field has its own tooltip with information what should be users input, its limits and form. Also, we can see menu with item Info where full information about program usage is.



Picture 2. Progress-page: time is calculated at the beginning.



Picture 3. Calculations in progress.

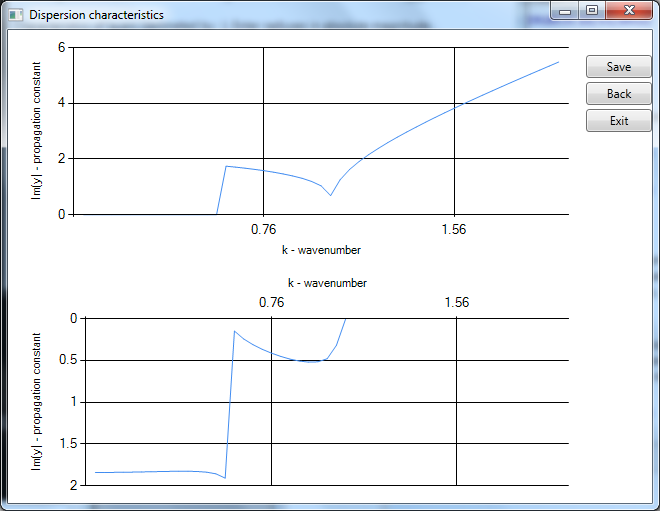


Picture 3.1. Calculations finished.

While calculations are running progress-page shows approximated time estimated and percentage of completed calculations. Also, this page provides ability to abort calculations any time.

When calculations are finished “OK”-button enables and after pressing it results appear. Even pages title changes according to users choice. It would be “Dispersion characteristics”, “Critical conditions” or “Critical values”. You can see two charts showing dependence between propagation constant and wavenumber. Upper one shows real part of constant and lower shows imaginary part.

Dispersion characteristics always appear on two chart (real and imaginary parts of propagation constant). Critical conditions of limited decay by their definition are imaginary, so the only chart is imaginary part of constant. Critical values are all the values between critical conditions for every radius, that’s why the best way to present them is to create its own chart for every radius.



Picture 4. Charts showing dispersion characteristics.

1. Verification and testing

WHY

The program is not real-time. When you need results with good precision it may take long time. FEA program gives better precision than initial Matlab program.

Results of initial program on m-language for 2 layers waveguides were compared with source [2], so we can compare results of FEA with them.

**+**

**Summarise the tests, why choose these?**

**Table**

**Test 1 what? Disp char for 2 layers, why: simple test to verify basic operation**

**Test 2 what?**

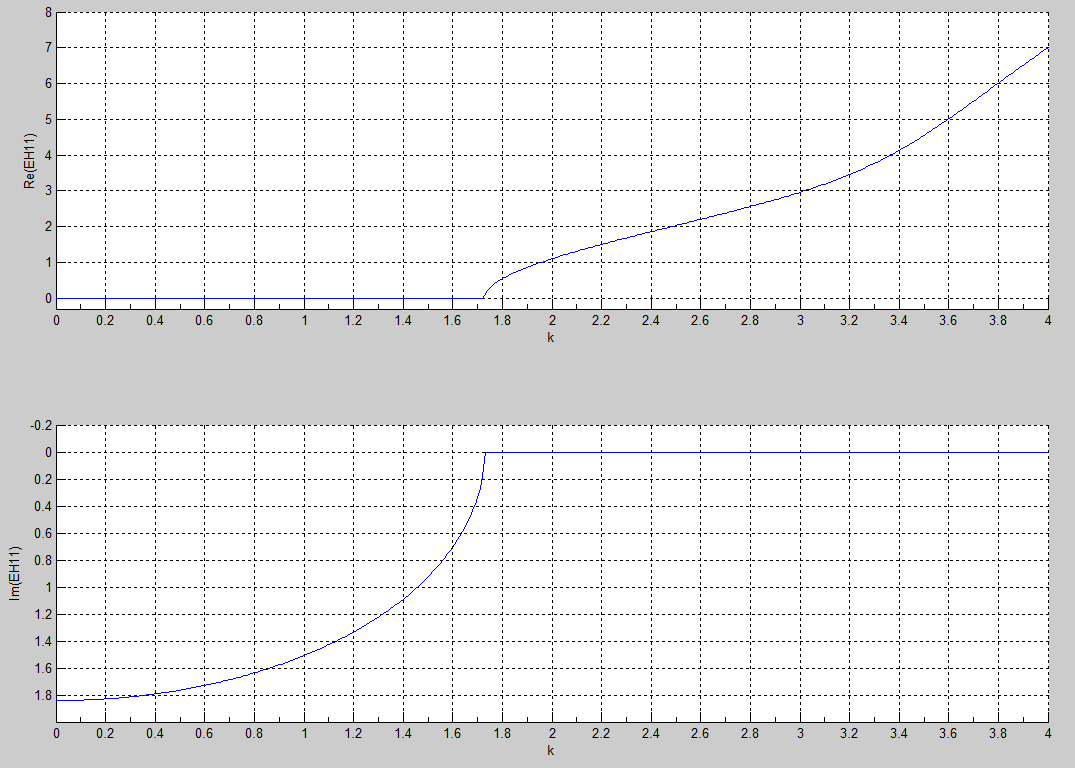
**Test 1.**

Waveguide with 2 layers. Permittivity changes as:

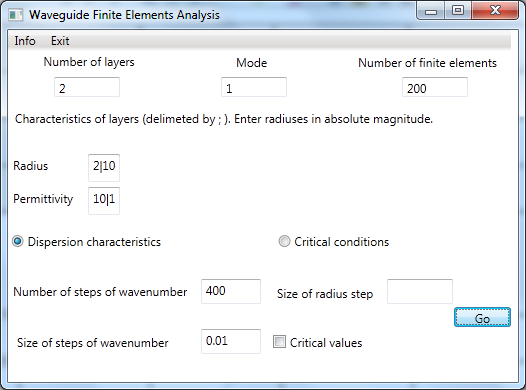
=

Original program results – picture 5. Time – about 20 minutes.

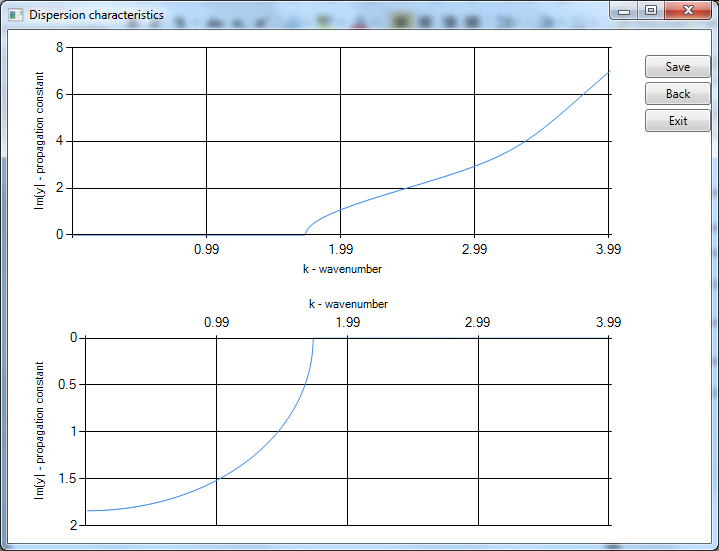
FEA results – picture 6. Time – about 15 minutes.



Picture 5. Original program results. Layers – 2, inside layer radius – 20% of total waveguide radius.



Picture 6. FEA input matching original input.



Picture 7. FEA results. Layers – 2, inside layer radius – 20% of total waveguide radius.

## Test 2.

Dispersion characteristics of waveguide with 6 layers.

Layers characteristics:

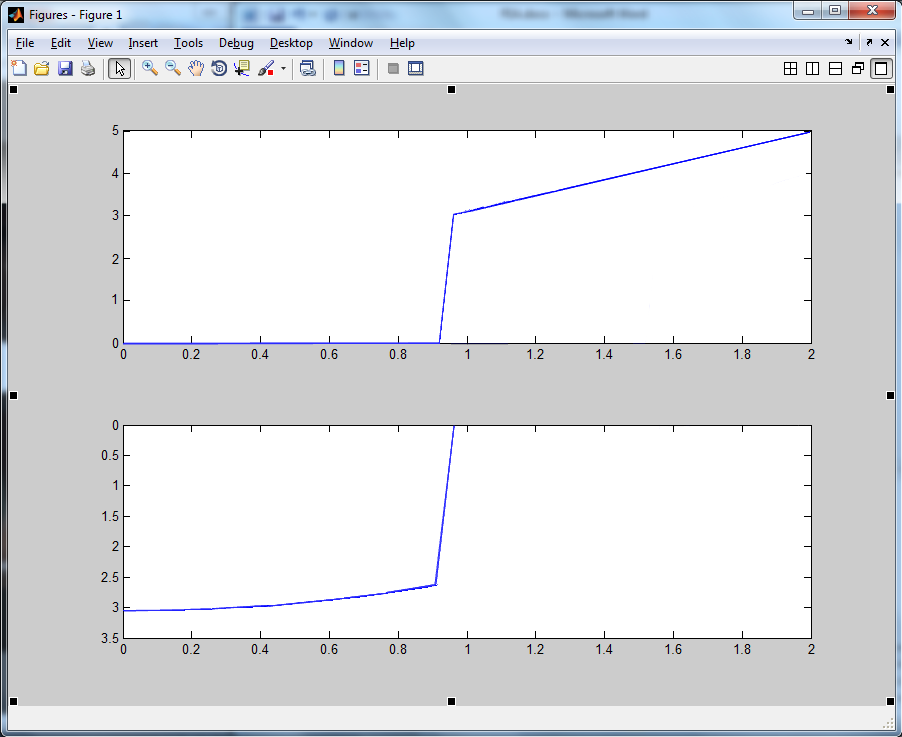
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Radius | 0.2 | 0.4 | 0.6 | 0.8 | 0.9 | 1 |
| Permittivity | 1 | 0.5 | 8 | 2 | 1.9 | 10 |

Mode: 2

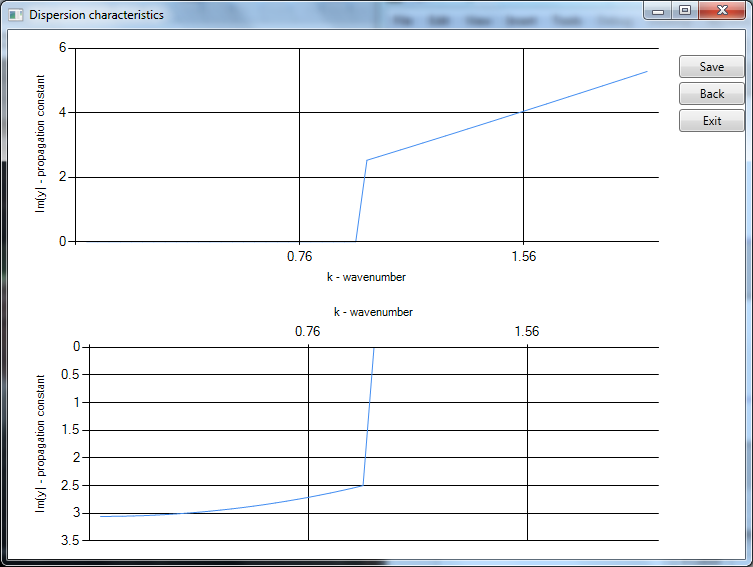
Number of operations: 50.

Original program – time: about 4 minutes of calculation.

FEA – time: about 3 minutes.



Picture 8. Original program results.



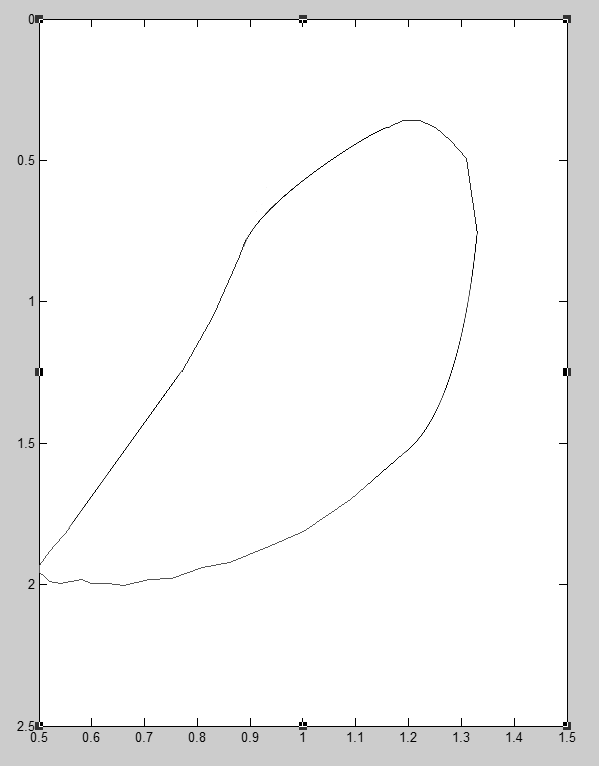
Picture 9. FEA results.

## Test 3.

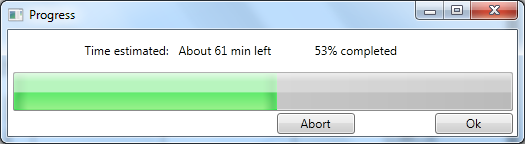
Critical conditions for waveguide with 2 layers.

Original program – picture 10. Time – about 150 minutes of pure calculations. Also it needs manual finding of critical conditions of limiting decay.

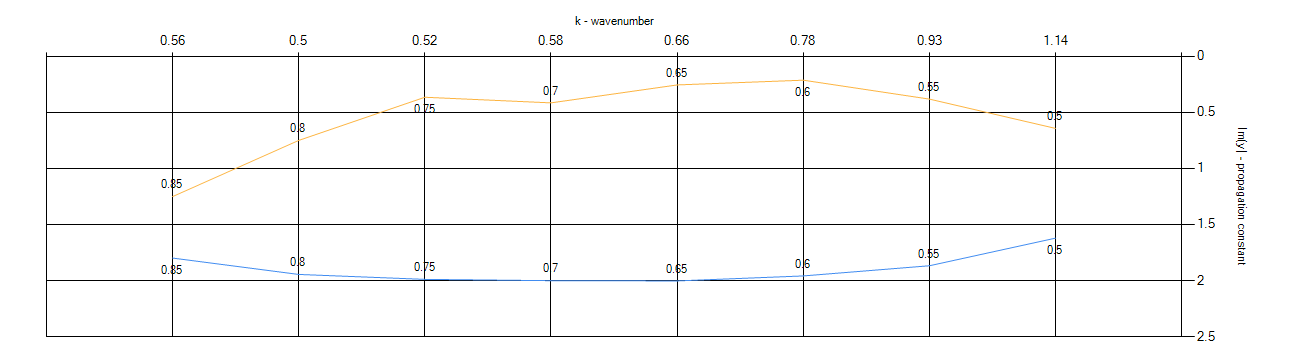
FEA results – picture 12. Time – 133 minutes of pure calculations and about 20-40sec to form charts.



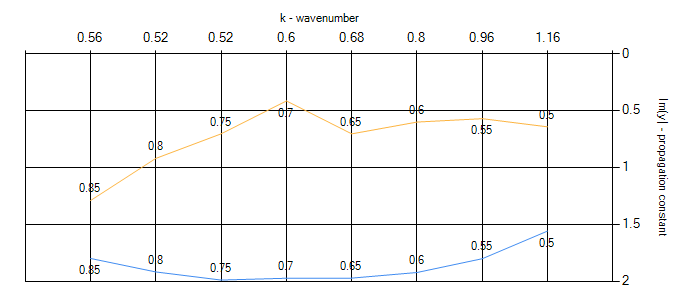
Picture 10. Original program results.



Picture 11. Progressbar while calculations.



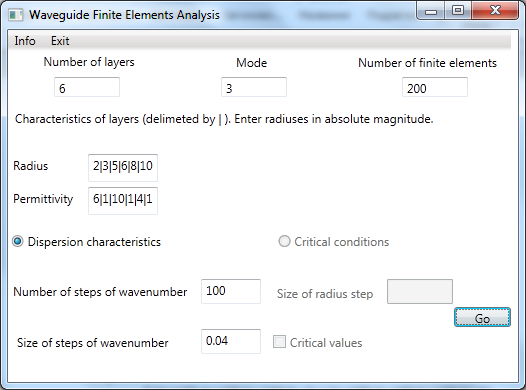
Picture 12a. FEA results(wavenumber step 0.01). Results have numerical and graphical compliance with original program and source [2].



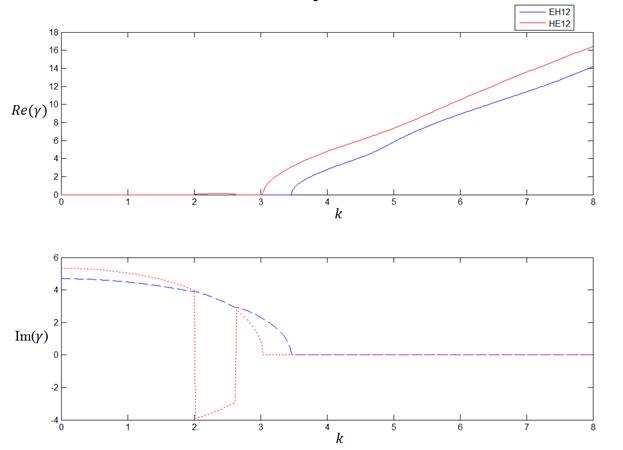
Picture 12b. FEA results(wavenumber step 0.04).

**Test 4.**

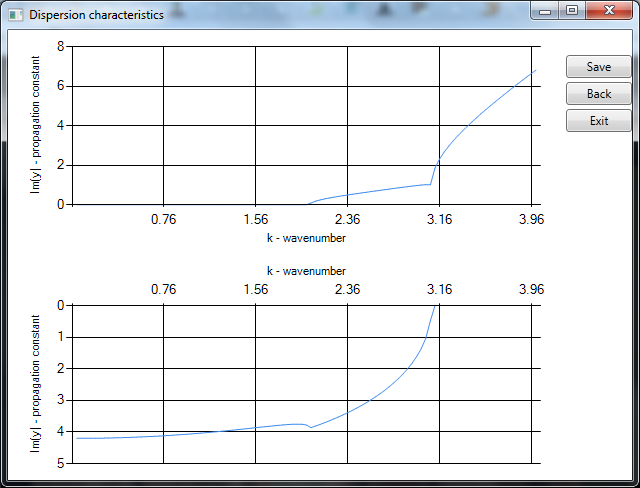
Waveguide with 6 layers. For modes 3 and 4.



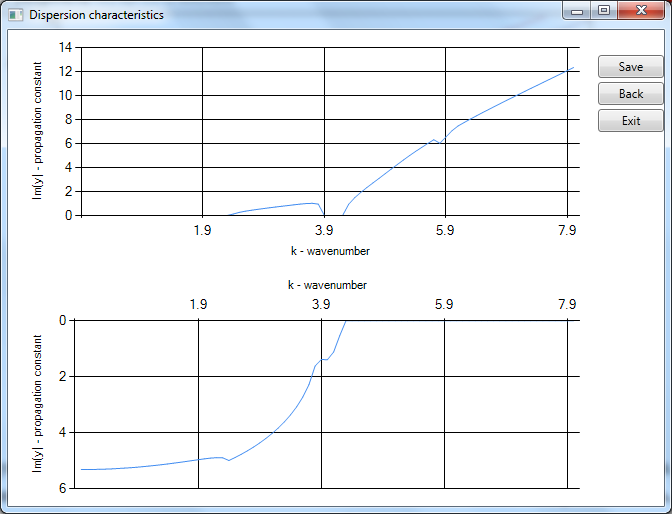
Picture 13. Waveguides characteristics.



Picture 14. Original program result.



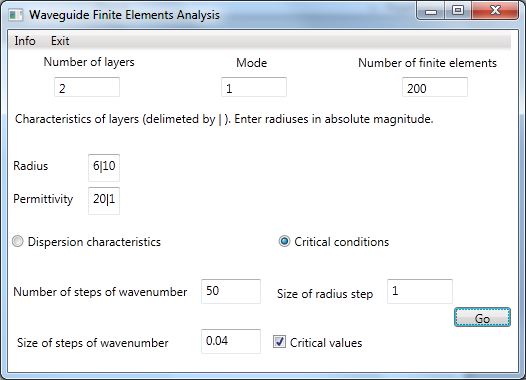
Picture 15.1. FEA results: mode 3.



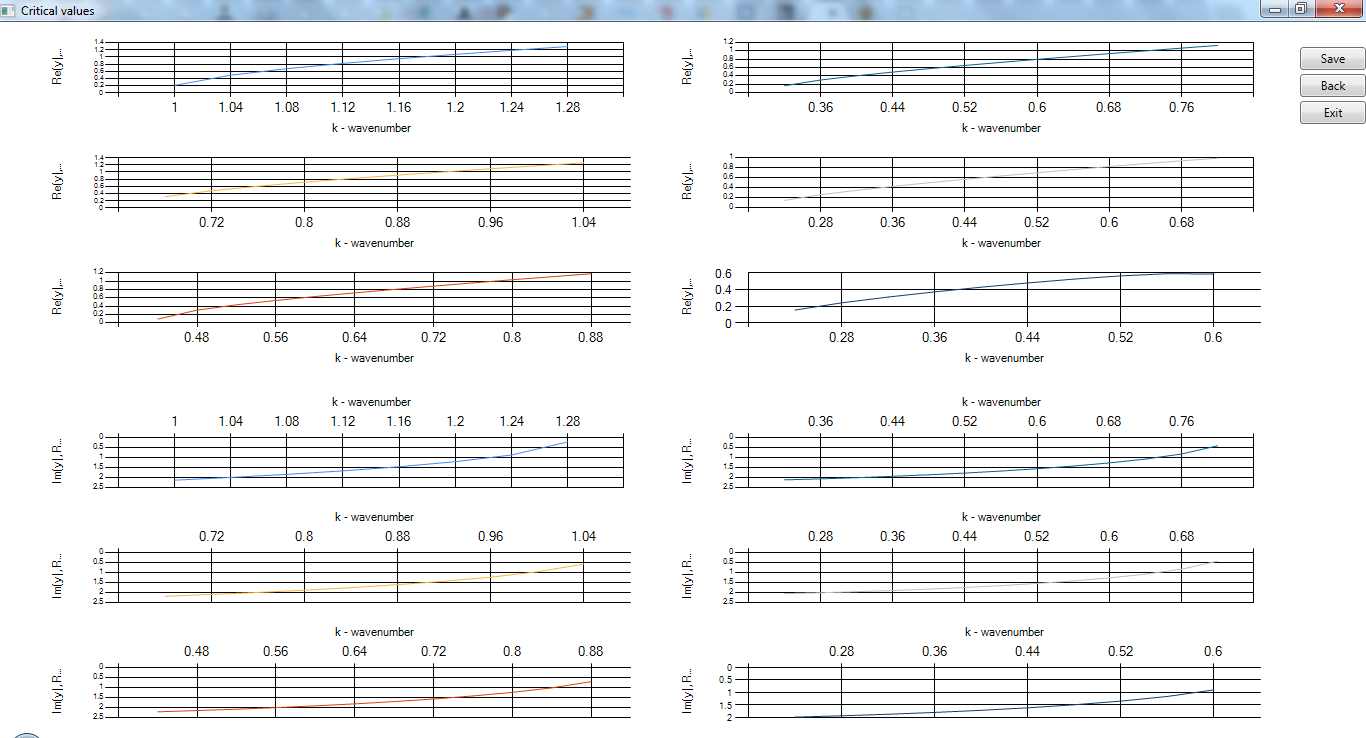
Picture 15.2. FEA results: mode 4.

**Test 5.**

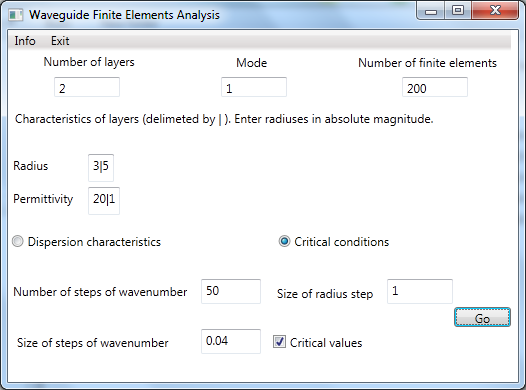
Critical values. Quantity of found critical values depends on initial precision – size of wavenumber steps. As you can see on pic. 16 and 18 all other characteristics are the same, but size and number of steps of wavenumber are changed. The results are represented on pic. 17 and 19.



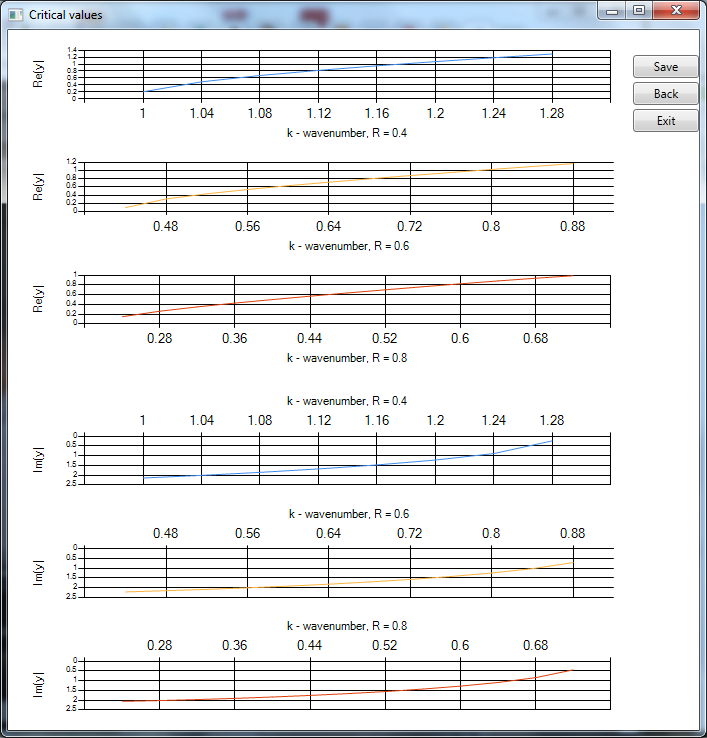
Picture 16.



Picture 17. Critical values.



Picture 18.

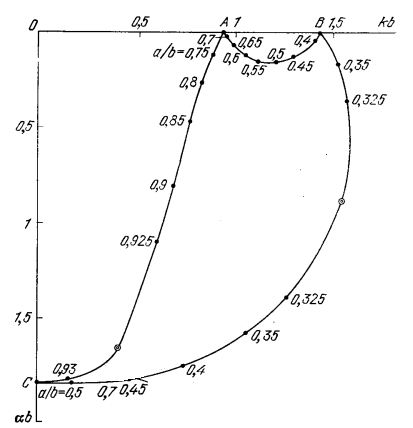


Picture 19. Less critical values.

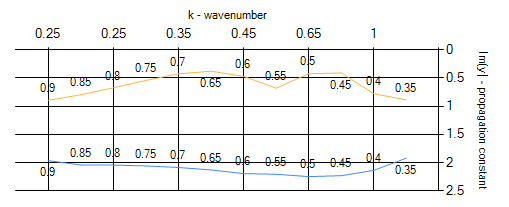
## Test 6.

Critical conditions.

In this test we can see comparison between source and program results.



Picture 20. Original chart from source.



Picture 21. FEA charts. We can see offcet.

Table 1.



1. Conclusion`

References

1. Garbow B. S., Boyle J. M., Dongarra J. J., Moler C. B.. Matrix Eigensystem Routines - EISPACK Guide Extension//Lecture Notes in Computer Science. Berlin, 1977. Vol. 51.
2. Veselov G.I., Raevskij S.B.. Multilayer metal dielectric waveguides. Moscow, 1988.
3. Wait R., Mitchell A.R.. Finite Element Analysis and Applications. 1985.
4. Wait R., Mitchell A.R.. Finite Element Analysis in PDEs. 1981.
5. Vainshtein L.A.. Electromagnetic waves. Moscow, 1988
6. Basharin A.A., Menshih N.L.. Features of the propagation of electromagnetic waves in a waveguide of a metamaterial with losses. (E-magazine) 2010. URL: <http://jre.cplire.ru/jre/nov10/2/text.pdf>
7. Caloz C., Itoh T. Electromagnetic metamaterials: transmission line theory and microwave applications. N.Y.: John Wiley and Sons, Inc., 2006.
8. Appendices

## 6.1. Complex number

Comparison(according to Matlab comparison)

public static bool operator <(Complex c1, Complex c2)

{

double a1 = c1.re;

double a2 = c2.re;

double b1 = c1.im;

double b2 = c2.im;

if (c1.Abs() < c2.Abs()) return true;

if (c1.Abs() > c2.Abs()) return false;

if (c1.Abs() == c2.Abs())

{

if ((a1 > 0 && a2 > 0 && b1 > 0 && b2 > 0) || (a1 < 0 && a2 < 0 && b1 < 0 && b2 < 0))

{

if (c1.Arg() < c2.Arg()) return true;

return false;

}

if ((a1 < 0 && a2 > 0 && b1 > 0 && b2 > 0))

{

return false;

}

if ((a1 > 0 && a2 < 0 && b1 < 0 && b2 < 0))

{

return false;

}

if ((a1 < 0 && a2 < 0 && b1 > 0 && b2 > 0) || (a1 > 0 && a2 > 0 && b1 < 0 && b2 < 0))

{

if (Math.Abs(c1.Arg()) > Math.Abs(c2.Arg())) return true;

return false;

}

if (b1 > 0 && b2 < 0) return true;

if (a1 == 0 || a2 == 0)

{

if (c1.im < c2.im) return true;

return false;

}

if (b1 == 0 || b2 == 0)

{

if (c1.re < c2.re) return true;

return false;

}

return false;

}

return false;

}

## 6.2. Matrixes

#region "Final Matrices"

/// <summary>

/// Setting of matrix A

/// </summary>

/// <param name="n">number of columns and rows</param>

/// <param name="kc">wave number</param>

/// <param name="ec">permittivity</param>

/// <param name="mc">mode</param>

/// <param name="r">Width of layer</param>

public void SetA(int n, double kc, int mc, WorkObject.LAY[] layers)

{

double eps = epsR;

double hc = 1.0/n;

double ec = layers[0].perm;

double ec1 = 0;

if (this.Cols() == 0 || this.Rows() == 0)

{

this.rows = 3\*n-2;

this.cols = 3\*n-2;

this.matrix = new double[3 \* n - 2, 3 \* n - 2];

}

{

this.matrix[0, 0] = (pA22(eps,hc,kc,ec)+ pA11(1,hc,kc,ec));

this.matrix[0, 1] = (pA23(eps, hc, mc));

this.matrix[0, 2] = (pA25(eps, hc, kc, ec, mc) + pA14(1, hc, kc, ec, mc));

this.matrix[1, 0] = (pA32(eps, hc, mc));

this.matrix[1, 1] = (pA33(eps, hc, kc, ec, mc));

this.matrix[1, 2] = (pA35(eps, hc, kc, ec));

this.matrix[2, 0] = (pA52(eps, hc, kc, ec, mc) + pA41(1, hc, kc, ec, mc));

this.matrix[2, 1] = (pA53(eps, hc, kc, ec));

this.matrix[2, 2] = (pA55(eps, hc, ec, mc) + pA44(1, hc, ec, mc));

this.matrix[0, 3] = (pA12(1, hc, kc, ec));

this.matrix[0, 4] = (pA13(1, hc, mc));

this.matrix[0, 5] = (pA15(1, hc, kc, ec, mc));

this.matrix[2, 3] = (pA42(1, hc, kc, ec, mc));

this.matrix[2, 4] = (pA43(1, hc, kc, ec));

this.matrix[2, 5] = (pA45(1, hc, ec, mc));

this.matrix[3, 0] = (pA21(1, hc, kc, ec));

this.matrix[3,2] = (pA24(1,hc,kc,ec,mc));

this.matrix[3,3] = (pA22(1,hc,kc,ec) + pA11(2,hc,kc,ec));

this.matrix[3,4] = (pA23(1,hc,mc));

this.matrix[3,5] = (pA25(1,hc,kc,ec,mc) + pA14(2,hc,kc,ec,mc));

this.matrix[4,0] = (pA31(1,hc,mc));

this.matrix[4,2] = (pA34(1,hc,kc,ec));

this.matrix[4,3] = (pA32(1,hc,mc));

this.matrix[4,4] = (pA33(1,hc,kc,ec,mc));

this.matrix[4,5] = (pA35(1,hc,kc,ec));

this.matrix[5,0] = (pA51(1,hc,kc,ec,mc));

this.matrix[5,2] = (pA54(1,hc,ec,mc));

this.matrix[5,3] = (pA52(1,hc,kc,ec,mc) + pA41(2,hc,kc,ec,mc));

this.matrix[5,4] = (pA53(1,hc,kc,ec));

this.matrix[5,5] = (pA55(1,hc,ec,mc) + pA44(2,hc,ec,mc));

this.matrix[3,6] = (pA12(2,hc,kc,ec));

this.matrix[3,7] = (pA13(2,hc,mc));

this.matrix[3,8] = (pA15(2,hc,kc,ec,mc));

this.matrix[5,6] = (pA42(2,hc,kc,ec,mc));

this.matrix[5,7] = (pA43(2,hc,kc,ec));

this.matrix[5,8] = (pA45(2,hc,ec,mc));

this.matrix[6,3] = (pA21(2,hc,kc,ec));

this.matrix[6,5] = (pA24(2,hc,kc,ec,mc));

this.matrix[6,6] = (pA22(2,hc,kc,ec) + pA11(3,hc,kc,ec));

this.matrix[6,7] = (pA23(2,hc,mc));

this.matrix[6,8] = (pA25(2,hc,kc,ec,mc) + pA14(3,hc,kc,ec,mc));

this.matrix[7,3] = (pA31(2,hc,mc));

this.matrix[7,5] = (pA34(2,hc,kc,ec));

this.matrix[7,6] = (pA32(2,hc,mc));

this.matrix[7,7] = (pA33(2,hc,kc,ec,mc));

this.matrix[7,8] = (pA35(2,hc,kc,ec));

this.matrix[8,3] = (pA51(2,hc,kc,ec,mc));

this.matrix[8,5] = (pA54(2,hc,ec,mc));

this.matrix[8,6] = (pA52(2,hc,kc,ec,mc) + pA41(3,hc,kc,ec,mc));

this.matrix[8,7] = (pA53(2,hc,kc,ec));

this.matrix[8,8] = (pA55(2,hc,ec,mc) + pA44(3,hc,ec,mc));

for (int i1 = 1; i1 < n-3; i1++)

{

for (int ii = 1; ii < layers.Length; ii++)

{

ec1 = ec;

if ((i1 + 3) \* hc > layers[ii-1].R - st2)

ec1 = layers[ii].perm;

if ((i1 + 3) \* hc > layers[ii-1].R - st1)

ec = layers[ii].perm;

}

this.matrix[3 + i1 \* 3, 6 + i1 \* 3] = (pA12(2 + i1, hc, kc, ec));

this.matrix[3 + i1 \* 3, 7 + i1 \* 3] = (pA13(2 + i1, hc, mc));

this.matrix[3 + i1 \* 3, 8 + i1 \* 3] = (pA15(2 + i1, hc, kc, ec, mc));

this.matrix[5 + i1 \* 3, 6 + i1 \* 3] = (pA42(2 + i1, hc, kc, ec, mc));

this.matrix[5 + i1 \* 3, 7 + i1 \* 3] = (pA43(2 + i1, hc, kc, ec));

this.matrix[5 + i1 \* 3, 8 + i1 \* 3] = (pA45(2 + i1, hc, ec, mc));

this.matrix[6 + i1 \* 3, 3 + i1 \* 3] = (pA21(2 + i1, hc, kc, ec));

this.matrix[6 + i1 \* 3, 5 + i1 \* 3] = (pA24(2 + i1, hc, kc, ec, mc));

this.matrix[6 + i1 \* 3, 6 + i1 \* 3] = (pA22(2 + i1, hc, kc, ec) + pA11(3 + i1, hc, kc, ec1));

this.matrix[6 + i1 \* 3, 7 + i1 \* 3] = (pA23(2 + i1, hc, mc));

this.matrix[6 + i1 \* 3, 8 + i1 \* 3] = (pA25(2 + i1, hc, kc, ec, mc) + pA14(3 + i1, hc, kc, ec1, mc));

this.matrix[7 + i1 \* 3, 3 + i1 \* 3] = (pA31(2 + i1, hc, mc));

this.matrix[7 + i1 \* 3, 5 + i1 \* 3] = (pA34(2 + i1, hc, kc, ec));

this.matrix[7 + i1 \* 3, 6 + i1 \* 3] = (pA32(2 + i1, hc, mc));

this.matrix[7 + i1 \* 3, 7 + i1 \* 3] = (pA33(2 + i1, hc, kc, ec, mc));

this.matrix[7 + i1 \* 3, 8 + i1 \* 3] = (pA35(2 + i1, hc, kc, ec));

this.matrix[8 + i1 \* 3, 3 + i1 \* 3] = (pA51(2 + i1, hc, kc, ec, mc));

this.matrix[8 + i1 \* 3, 5 + i1 \* 3] = (pA54(2 + i1, hc, ec, mc));

this.matrix[8 + i1 \* 3, 6 + i1 \* 3] = (pA52(2 + i1, hc, kc, ec, mc) + pA41(3 + i1, hc, kc, ec1, mc));

this.matrix[8 + i1 \* 3, 7 + i1 \* 3] = (pA53(2 + i1, hc, kc, ec));

this.matrix[8 + i1 \* 3, 8 + i1 \* 3] = (pA55(2 + i1, hc, ec, mc) + pA44(3 + i1, hc, ec1, mc));

}

this.matrix[6 + 3 \* (n - 4), 9 + 3 \* (n - 4)] = (pA13(3 + n - 4, hc, mc));

this.matrix[8 + 3 \* (n - 4), 9 + 3 \* (n - 4)] = (pA43(3 + n - 4, hc, kc, ec));

this.matrix[9 + 3 \* (n - 4), 6 + 3 \* (n - 4)] = (pA31(3 + n - 4, hc, mc));

this.matrix[9 + 3 \* (n - 4), 8 + 3 \* (n - 4)] = (pA34(3 + n - 4, hc, kc, ec));

this.matrix[9 + 3 \* (n - 4), 9 + 3 \* (n - 4)] = (pA33(3 + n - 4, hc, kc, ec, mc));

}

}

/// <summary>

/// Setting of matrix B

/// </summary>

/// <param name="n">number of columns and rows</param>

/// <param name="kc">wave number</param>

/// <param name="ec">permitivity</param>

/// <param name="mc">mode</param>

/// <param name="r">Width of layer</param>

public void SetB(int n, double kc, int mc, WorkObject.LAY[] layers)

{

double eps = epsR;

double hc = 1.0/n;

double ec = layers[0].perm;

double ec1 = 0;

if (this.Cols() == 0 || this.Rows() == 0)

{

this.rows = 3 \* n - 2;

this.cols = 3 \* n - 2;

this.matrix = new double[3 \* n - 2, 3 \* n - 2];

}

{

this.matrix[0, 0] = (pB22(eps, hc) + pB11(1, hc));

this.matrix[1, 1] = (pB33(eps, hc));

this.matrix[2, 2] = (pB55(eps, hc, ec) + pB44(1, hc, ec));

this.matrix[0, 3] = (pB12(1, hc));

this.matrix[2, 5] = (pB45(1, hc, ec));

this.matrix[3, 0] = (pB21(1, hc));

this.matrix[3, 3] = (pB22(1, hc) + pB11(2, hc));

this.matrix[4, 4] = (pB33(1, hc));

this.matrix[5, 2] = (pB54(1, hc, ec));

this.matrix[5, 5] = (pB55(1, hc, ec) + pB44(2, hc, ec));

this.matrix[3, 6] = (pB12(2, hc));

this.matrix[5, 8] = (pB45(2, hc, ec));

this.matrix[6, 3] = (pB21(2, hc));

this.matrix[6, 6] = (pB22(2, hc) + pB11(3, hc));

this.matrix[7, 7] = (pB33(2, hc));

this.matrix[8, 5] = (pB54(2, hc, ec));

this.matrix[8, 8] = (pB55(2, hc, ec) + pB44(3, hc, ec));

for (int i1 = 1; i1 < n-3; i1++)

{

for (int ii = 1; ii < layers.Length; ii++)

{

ec1 = ec;

if ((i1 + 3) \* hc > layers[ii - 1].R - st2)

ec1 = layers[ii].perm;

if ((i1 + 3) \* hc > layers[ii - 1].R - st1)

ec = layers[ii].perm;

}

this.matrix[3 + i1\*3,6 + i1\*3] = (pB12(2 + i1,hc));

this.matrix[5 + i1\*3,8 + i1\*3] = (pB45(2 + i1,hc,ec));

this.matrix[6 + i1\*3,3 + i1\*3] = (pB21(2 + i1,hc));

this.matrix[6 + i1\*3,6 + i1\*3] = (pB22(2 + i1,hc) + pB11(3 + i1,hc));

this.matrix[7 + i1\*3,7 + i1\*3] = (pB33(2 + i1,hc));

this.matrix[8 + i1\*3,5 + i1\*3] = (pB54(2 + i1,hc,ec));

this.matrix[8 + i1\*3,8 + i1\*3] = (pB55(2 + i1,hc,ec) + pB44(3 + i1,hc,ec1));

}

this.matrix[6 + 3\*(n-4),9 + 3\*(n-4)] = (pB12(3 + n-4,hc));

this.matrix[9 + 3\*(n-4),6 + 3\*(n-4)] = (pB21(3 + n-4,hc));

this.matrix[9 + 3\*(n-4),9 + 3\*(n-4)] = (pB33(3 + n-4,hc));

}

}

#endregion

#region "Eigenvalues"

//Matlab Arrays for eigenvalues

private MWArray[] res = null;

private MWNumericArray real = null;

private MWNumericArray imag = null;

private Complex[] eigenvalues;

/// <summary>

/// Generalised eigenvalues of A and B (analog to MATLAB M = eig(A,B))

/// </summary>

/// <param name="B">Matrix B</param>

/// <returns>Generalised eigenvalues</returns>

public Complex[] eige(Matrix B)

{

//calling Matlab API

Eig testob = new Eig();

//Matlab Array which gets result of Matlab function eig(A,B)

res = testob.Eigenvalues(2, (MWNumericArray)this.matrix, (MWNumericArray)B.matrix, this.Rows());

//arrays for real and imaginary parts

real = (MWNumericArray)res[0];

imag = (MWNumericArray)res[1];

//copying parts into CSharp arrays

double[,] resCR = (double[,])real.ToArray(MWArrayComponent.Real);

double[,] resCI = (double[,])imag.ToArray(MWArrayComponent.Real);

this.eigenvalues = new Complex[21];

Complex[] eigenbuf = new Complex[this.rows];

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

{

eigenbuf[i] = new Complex(resCR[i, 0], resCI[i,0]);

eigenbuf[i] = eigenbuf[i].Pow(0.5);

}

Complex buf = new Complex();

buf.quickSort(ref eigenbuf, 0, this.rows-1);

for (int i = 0; i < 21; i++)

{

this.eigenvalues[i] = eigenbuf[200 + i - 1];

}

return this.eigenvalues;

}

#endregion

## 6.3. Characteristics

#region "Critical values and conditions"

/// <summary>

/// Function searches critical values and condition. Result depends on param "isCond"

/// </summary>

/// <param name="fe">#finite elements</param>

/// <param name="Cstep">Step for radius checking</param>

/// <param name="Nsteps">Step of wave number</param>

/// <param name="step">Size of step of wave number</param>

/// <param name="mode">Mode #</param>

/// <param name="ec">Permittivity</param>

/// <param name="isCond">Choise of return</param>

/// <returns>

/// If isCond = true: function returns critical conditions, values that are the first and the last complex numbers in all

/// dispersion characteristics

/// If isCond = false: function returns critical values: all the numbers between critical(including them)

/// </returns>

public CRIT[] Crit(int fe, double Cstep, int Nsteps, double step, int mode, LAY[] L, ref System.ComponentModel.BackgroundWorker bg, bool isCond = true)

{

if (L.Length == 2)

{

int N = Convert.ToInt32(1 / Cstep);

LAY[] bufL = new LAY[L.Length];

bufL = L;

CRIT[] buf = new CRIT[N];

CRIT[] precrit = new CRIT[N];

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

{

precrit[i].D = new DISP[2];

}

CRIT[] critVal = new CRIT[N];

bool isChecked = false;

int iniProgress = 0;

int coef = N;

//all the dispersion characteristics for every radius

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

{

buf[i].R = i \* Cstep;

bufL[0].R = i \* Cstep;

if (i == 0) isChecked = false;

else isChecked = true;

buf[i].D = dispersion(fe, Nsteps, step, mode, bufL, ref bg, ref iniProgress, coef, isChecked);

}

#region "Filling critical values and conditions"

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

{

int Beg = 0, End = 0;

for (int ii = 0; ii < Nsteps; ii++)

{

if (buf[i].D[ii].y.isComplex())

{

precrit[i].R = buf[i].R;

precrit[i].D[0] = buf[i].D[ii];

Beg = ii;

break;

}

}

if (Beg != 0)

{

for (int ii = Nsteps - 1; ii > 1; ii--)

{

if (buf[i].D[ii].y.isComplex())

{

precrit[i].R = buf[i].R;

precrit[i].D[1] = buf[i].D[ii];

End = ii;

break;

}

}

}

if (Beg != 0 && End != 0 && !isCond)

{

critVal[i].D = new DISP[End - Beg + 1];

critVal[i].R = buf[i].R;

for (int ii = Beg; ii <= End; ii++)

{

critVal[i].D[ii - Beg] = buf[i].D[ii];

}

}

}

if (!isCond)

{

int counter = 0;

CRIT[] bufcrit = new CRIT[N];

for (int j = 0; j < N; j++)

{

if (!isNull(critVal[j].R))

{

bufcrit[counter] = critVal[j];

counter++;

}

}

critVal = new CRIT[counter];

for (int j = 0; j < counter; j++)

critVal[j] = bufcrit[j];

return critVal;

}

else

{

//deleting "null" in precrit

int counter = 0;

CRIT[] bufcrit = new CRIT[N];

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

{

if (!isNull(precrit[i].R))

{

bufcrit[counter] = precrit[i];

counter++;

}

}

CRIT[] critCond = new CRIT[counter];

//final array of critical conditions

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

{

critCond[i] = bufcrit[i];

}

#endregion

return critCond;

}

}

else

{

CRIT[] c = new CRIT[1];

c[0].R = 0.0;

c[0].D = new DISP[1];

c[0].D[0].k = 0.0;

c[0].D[0].y = new Complex();

return c;

}

}

#endregion

## 6.4. Example of saved characteristics.

