APÉNDICE A

En el siguiente apéndice se dará una explicación muy general del funcionamiento del programa. El programa se ha seccionado en 9 partes importantes, de éste modo será más sencilla su explicación. Para la presente tesis se hizo uso de MATHEMATICA 5.0 para llevar acabo la simulación, basado en el método RK4. El programa solo admite un máximo de dos haces con igual longitud de onda.

Parte [1]. Para simular la muestra se necesitan una serie de parámetros indispensables que caracterizan al material y estructura del arreglo. Dentro de esta lista de valores, los de mayor relevancia son el número de elementos N (Num), la longitud de onda la radiación incidente (λ), la separación entre cada guía de onda (d) y el grosor de las mismas (Δx). No de menos importancia pero dependientes de los primeros, el coeficiente de acoplamiento y la intensidad de la no linealidad quedan definidos en esta parte. También se definen los parámetros de cada uno de los rayos, los valores de la intensidad pico de la función del campo, la media y la fase.

Parte [2]. Determinación de las condiciones iniciales a partir de la determinación de ciertos valores de cada haz. Además el programa elige el haz que será usado para radiar la muestra.

Parte [3]. Se determina la cantidad total de energía suministrada al arreglo.

Parte [4]. En esta parte se seleccionan las condiciones iniciales para hacer uso del método de Runge-Kutta de 4° orden. Además, es en esta parte donde se determinan las condiciones de frontera entre el material y el medio a su entorno.

Propiamente esta es la parte fundamental del programa, pues primero se generan los valores de las cuatro constantes usadas en la estimación de la pendiente. Para un sistema de N ecuaciones diferenciales deben ser generadas 4N constantes. Adicionalmente a esto, en esta sección se lleva a cabo la generación de las N ecuaciones de recursividad para aproximar cada una de las ecuaciones.

Parte [5]. Una vez halladas las funciones de recursividad, se procede a obtener el conjunto de puntos de aproximación, las graficas, así como una función de interpolación de cada lista de puntos únicamente con fines de presentación.

Parte [6]. Basados en el hecho de que el campo dentro de cada guía posee una cierta distribución, pero por simplificación solo será caracterizado por el valor del campo en el centro de cada canal. Se hace uso de funciones uniformes que representen el grosor de cada guía, con esto se logra la generación de la simulación de cada guía de onda.

Parte [7]. Con base en los modelos matemáticos del Hamiltoneano y Conservación de la energía, se hace su aplicación en el programa con el fin de determinar el valor de cada una de estas propiedades para cualquier valor de z. Es importante recordar que para que las simulaciones sean validas, en cada corrida del programa estas funciones deben ser siempre constantes.

Parte [8]. La simulación de la distribución de la luz se hace de dos formas. Para esta parte se toma la aparente distribución de la intensidad de la luz en todo el arreglo por medio de un sistema tridimensional de las distribuciones en cada canal.

Parte [9]. En esta parte se obtienen lo valores de las intensidades a la salida de cada canal. Se obtiene una función de interpolación a partir de los datos obtenidos, solo con fines comparativos.

Parte [10]. Finalmente, haciendo uso de las simulaciones hechas para cada guía en la *Parte* [6], se hace una simulación general de la apariencia real del arreglo y de la distribución que la luz adquiere en su interior.

A continuación se muestra el código del programa computacional.

Obtención de soluciones numéricas para N guía de onda

Método Runge-Kutta

```
Block[{ $RecursionLimit=Infinity},
        Clear[
                E01,E02,m1,m2,ω1,ω2,Δx,ϕ01,ϕ02,Num,λ,d,grosor,gE1,n0,μ,c,k,phase,gE,
                ini1,ini2,end1,end2,zero1,zero2,j,i,f,δz,z,pts,graph,w,graph1,totalPower
        ];
        (*[1]*)
  (*Parametros de arreglo*)
  Num=51;
  d=7*10^-6;
  Δx=4*10^-6;
  grosor=(Num*Δx)+((Num-1)d);
  gE1=Table[0,{Num}];
  n0=3.5;
  μ=3.35*10^-3;
  c=.52;
  λ=1.53*10^-6;
  k=(2Pi n)/λ;
  δz=.1;
  sampleLong=30;
  Print["Grosor = "<>ToString[N[grosor]]<>" m"];
  (*Parámetros de los haces*)
  E01=5;m1=23;ω1=9Δx;ϕ01= 8Pi/16;
  E02=0;m2=1;ω2=.3Δx;ϕ02= 0Pi/2;
        (*[2]*)
  If[E01≠0,
    (*Simulacion de haz 1 con distribucion Gaussiana*)
        normal1[x_]=E01 Exp[-((x-m1)^2)/((ω1/(d+Δx))^2)];
        gE=Table[
                AccountingForm[Chop[N[normal1[j]],0.001],NumberSigns→{"-",""}],
        {j,1,Num,1}];
        zero1=Num-Count[ToExpression[ToString[gE]],0];
        ini1=m1-(zero1-1)/2;
        end1=m1+(zero1+1)/2;
        expo1[x_]=Exp[I x ϕ01];
        phase=Table[N[expo1[j]],{j,1,Num,1}];
```

```
    (*Establecimiento de las primeras condiciones iniciales*)
        For[i=ini1-1,i<end1,
                gE1[[i]]=phase[[i]]*ToExpression[ToString[gE[[i]]]],
        i++];
        arreglo01=gE1;
    (*Número de canales excitados por el haz 1*)
        Print[
                "Guías Excitadas = "<>ToString[zero1]<>
                ", Ancho del haz 1 (ω1)= "<>
                ToString[AccountingForm[N[ω1]]]<>" m"
        ];
   ,normal1[x_]=E01 Exp[-((x-m1)^2)/((ω1/(d+Δx))^2)];
  ];

  If[E02≠0,
    (*Simulacion de haz 2 con distribucion Gaussiana*)
        Clear[gE,phase];
        normal2[x_]=E02 Exp[-((x-m2)^2)/((ω2/(d+Δx))^2)];
        gE=Table[
                AccountingForm[Chop[N[normal2[j]],0.001],NumberSigns→{"-",""}],
        {j,1,Num,1}];
        zero2=Num-Count[ToExpression[ToString[gE]],0];
        ini2=m2-(zero2-1)/2;
        end2=m2+(zero2+1)/2;
        expo2[x_]=Exp[I x ϕ02];
        phase=Table[N[expo2[j]],{j,1,Num,1}];
    (*Establecimiento de las segundas condiciones iniciales*)
        For[i=ini2-1,i<end2,
                gE1[[i]]=phase[[i]]*ToExpression[ToString[gE[[i]]]],
        i++];
        arreglo01=gE1;
    (*Número de canales excitados por el haz 2*)
        Print[
                "Guías Excitadas = "<>ToString[zero2]<>
                ", Ancho del haz 2 (ω2) = "<>
                ToString[AccountingForm[N[ω2]]]<>" m"
        ];
   ,normal2[x_]=E02 Exp[-((x-m2)^2)/((ω2/(d+Δx))^2)];
  ];
        (*[3]*)
  totalPower=Sum[Abs[arreglo01][[i]]^2,{i,Num}];
  Print["Potencia total administrada = "<>ToString[totalPower]]
        (*[4]*)
    (*Generación de parametros iniciales para RK4*)
  For[i=1,i<Num+1,i++,
        f[i][0]=arreglo01[[i]]
  ];
  f[0][j_]=0;f[Num+1][j_]=0;
  w[1][0][j_]=0;w[1][Num+1][j_]=0;
  w[2][0][j_]=0;w[2][Num+1][j_]=0;
  w[3][0][j_]=0;w[3][Num+1][j_]=0;
  w[4][0][j_]=0;w[4][Num+1][j_]=0;
```

```
(*[6]Soluciones numéricas por RK4*)
z[0]=0;
z[j_]:=N[z[j-1]+δz];
For[i=1,i<Num+1,i++,
      ToExpression[
            "w[1]["<>ToString[i]<>"][j_]:=μ*I*f["<>ToString[i]<>
            "][j]*Abs[f["<>ToString[i]<>"][j]]^2+I*c*f["<>ToString[i]<>
            "+1][j]+I*c*f["<>ToString[i]<>"-1][j]"
       ];
      ToExpression[
            "w[2]["<>ToString[i]<>"][j_]:=μ*I*(f["<>ToString[i]<>
            "][j]+.5*δz*w[1]["<>ToString[i]<>"][j])*Abs[(f["<>ToString[i]<>
            "][j]+.5*δz*w[1]["<>ToString[i]<>"][j])]^2+I*c*(f["<>ToString[i]<>
            "-1][j]+.5*δz*w[1]["<>ToString[i]<>"-1][j])+I*c*(f["<>ToString[i]<>
            "+1][j]+.5*δz*w[1]["<>ToString[i]<>"+1][j])"
       ];
      ToExpression[
            "w[3]["<>ToString[i]<>"][j_]:=μ*I*(f["<>ToString[i]<>
            "][j]+.5*δz*w[2]["<>ToString[i]<>"][j])*Abs[(f["<>ToString[i]<>
            "][j]+.5*δz*w[2]["<>ToString[i]<>"][j])]^2+I*c*(f["<>ToString[i]<>
            "-1][j]+.5*δz*w[2]["<>ToString[i]<>"-1][j])+I*c*(f["<>ToString[i]<>
            "+1][j]+.5*δz*w[2]["<>ToString[i]<>"+1][j])"
       ];
      ToExpression[
            "w[4]["<>ToString[i]<>"][j_]:=μ*I*(f["<>ToString[i]<>
            "][j]+δz*w[3]["<>ToString[i]<>"][j])*Abs[(f["<>ToString[i]<>
            "][j]+δz*w[3]["<>ToString[i]<>"][j])]^2+I*c*(f["<>ToString[i]<>
            "-1][j]+δz*w[3]["<>ToString[i]<>"-1][j])+I*c*(f["<>ToString[i]<>
            "+1][j]+δz*w[3]["<>ToString[i]<>"+1][j])"
       ];
      ToExpression[
            "f["<>ToString[i]<>"][j_]:=f["<>ToString[i]<>"][j]=f["<>
            ToString[i]<>"][j-1]+(δz/6)(w[1]["<>ToString[i]<>
            "][j-1]+2w[2]["<>ToString[i]<>"][j-1]+2w[3]["<>ToString[i]<>
            "][j-1]+w[4]["<>ToString[i]<>"][j-1])"
       ];
];
       (*[5]*)
  (*Creación de conjuntos de datos y funciones para graficar*)
  For[i=1,i<Num+1,i++,
      ToExpression[
            "pts["<>ToString[i]<>"]:=Table[{z[i],Abs[f["<>ToString[i]<>
            "][i]]^2},{i,0,(sampleLong/δz)}]"
       ];
      ToExpression[
            "graph["<>ToString[i]<>"]:=ListPlot[pts["<>
            ToString[i]<>"],PlotRange→All,PlotJoined->True,PlotStyle→{"<>
            "RGBColor[1,0,0]},DisplayFunction→Identity]"
       ];
```

```
        ToExpression[
                "graph1["<>ToString[i]<>"]=Interpolation[pts["<>
                ToString[i]<>"]]"
        ];
  ];
        (*[6]*)
  agoa=ToExpression[
                Table[
                        "UnitStep[-(x-"<>ToString[j-.2]<>")(x-"<>ToString[j+.2]<>
                        ")]graph1["<>ToString[j]<>"][z]",{j,1,Num}
                ];
        ];
];
```

Conservación de Energía y Hamiltoneano

```
        (*[7]*)
Clear[hamilton, energy]
Block[{$RecursionLimit = Infinity},
        hamilton =
                Interpolation[
                        Table[
                                {z[j], Total[
                                        Table[c*Abs[f[i][j]]^2 + c*Abs[f[i + 1][j]]^2 -
                                        c*Abs[-f[i + 1][j] + f[i][j]]^2 + (•/2)Abs[f[i][j]]^4,
                                        {i, 0, Num}]
                                ]
                        },{j, 0, (sampleLong/•z)}]
                ];
        energy[z_] = Total[Table[graph1[i][z], {i, 1, Num}]];
]
Plot[{energy[z], hamilton[z]}, {z, 0, sampleLong},
PlotRange -> {0, If[energy[0] -> hamilton[0], 1.5 energy[0], 1.5 hamilton[0]]},
PlotStyle -> {{RGBColor[1, 0, 0], Thickness[.01]},
                {RGBColor[1,0, 0], Dashing[{.02,.03}],Thickness[.01]}},
AxesLabel -> {z, "H, Po"}, TextStyle -> {FontSize -> 12}, ImageSize -> {400, Automatic}];
```

AGOA (intensidades) 3D

```
        (*[8]*)
Block[{$RecursionLimit=Infinity},
        <<Calculus`DSolveIntegrals`
        Needs["Graphics`Graphics3D`"]
        Show[StackGraphics[
                                Table[graph[i],{i,1,Num}]
                        ],
                        BoxRatios→{8Num,8 Num, 5 Num}, Boxed→False,
                        Axes→{True,True, True}, AxesLabel→{"z","N"," |E_N|^2"},
                        PlotRange→ All,AxesStyle→{RGBColor[1,0,0]},
                        ViewPoint->{-2.253, -2.119, 1.372},
                        Ticks→{Automatic,{} ,Automatic},ImageSize→{570,400},
                        TextStyle→{FontFamily→Default,FontSize→14}
        ];
  ]
```

```
Distribución Entrada-Salida y AGOA (intensidades)

Block[{ $RecursionLimit=Infinity},
        Clear[
                discreteDistribution,compareDistributions,sampleSimulation,
                outputAproximation,discreteValues
        ];
        (*[9]*)
  (*Obtención de valores discretos en la salida*)
        discreteValues=
                Table[
                        {i,graph1[i][sampleLong]},{i,Num}
                ];
  (*Generación de gráficas*)
        discreteDistribution=
                Show[
                        Graphics[
                                Table[
                                        Line[
                                                {discreteValues[[i]],{i,0}}
                                        ],{i,1,Num}
                                ],
                        Axes→True,AspectRatio→.9,PlotRange→All
                        ],
                        PlotLabel→"(a)",AxesLabel →{"N","|EN|^2"},
                        DisplayFunction→Identity,
                        TextStyle→{FontFamily→Default,FontSize→14},
                        AxesOrigin→{.5,0}
                ];
  (*Comparación Entrada-Salida*)
        outputAproximation=Interpolation[discreteValues];
        compareDistributions=
                Plot[
                        {outputAproximation[x],normal1[x]^2,normal2[x]^2},{x,1,Num},
                        PlotRange→All,AspectRatio→.9,
                        PlotStyle→{{RGBColor[0,0,1],Thickness[.01]},
                                   {RGBColor[1,0,0],Dashing[{.01,.015}],
                                      If[E01≠0,Thickness[.01],Thickness[0]]
                                   },
                                   {RGBColor[1,0,0],Dashing[{.01,.015}],
                                      If[E02≠0,Thickness[.01],Thickness[0]]
                                  }},
                        AxesLabel →{"N","|EN|^2"},
                        PlotLabel→"(b)",AxesOrigin→{.5,0},
                        TextStyle→{FontFamily→Default,FontSize→14},
                        DisplayFunction→Identity
                ];
        (*[10]*)
  (*Simulación de la muestra*)
```

```
  sampleSimulation=
        ContourPlot[
                        Total[agoa,Num],{x,0,Num+1},{z,0,sampleLong},
                PlotRange→{0,If[E01<E02,.5E02^2,.5E01^2]},
                PlotPoints→250,Contours→40,
                ContourLines→False,Axes→True,AxesLabel→{N,None},
                AspectRatio→1,FrameLabel→{None,"z [mm]","(c)",None},
                FrameTicks→{{1,.5(Num+1),Num},
                            {0,.5sampleLong,sampleLong},
                            False,False},
                TextStyle→{FontFamily→Default,FontSize→14},
                DisplayFunction→Identity
        ];
  (*Presentación en arreglo de los gráficos*)
    Show[
        GraphicsArray[
                {discreteDistribution,
                compareDistributions,
                sampleSimulation}
        ],
        ImageSize→{800,225},Frame→True
    ];
    Print[
        maximo=Max[Table[discreteValues[[i]][[2]],{i,Num}]];
        "Hay un máximo de potencia = "<>ToString[maximo]<>
        " en la Guía numero "<>
        ToString[Position[discreteValues,maximo][[1]][[1]]
    ];
];
```