1 Fundamental Mathematics of Nonlinear-Emission Photonic Glass Fiber and Waveguide Devices

1.1 Introduction

In the design and analysis of nonlinear-emission photonic glass fiber and waveguide devices, one of the most important aspects is to solve the multi-variable rate equation and power-propagation equation groups. In many cases, the equation groups are nonlinear. Currently, there is no commercial software that can be directly used to solve multi-variable nonlinear equation groups. The methods introduced in this chapter include the Newton iteration algorithm and Runge–Kutta algorithm for the initial-value problem and the algorithm for the two-point boundary problem, which are effective numerical techniques for highly doped and co-doped fiber amplifiers and fiber laser systems as well as photonic glass waveguide systems for spectral conversion and white-light generation.

1.2 Newton Iteration Algorithm for Nonlinear Rate Equation Solution

1.2.1 Single-Variable [1]

A nonlinear equation group can be linearized into a linear equation group. At each iteration step, a new linear equation group can be obtained and solved, this method is called linearization method. For example, for a two-dimensional nonlinear population rate equation group:

$$\begin{aligned} \frac{dN_1}{dt} &= aN_1^3 + bN_1N_2 + cN_2^2 + d \\ \frac{dN_2}{dt} &= -aN_1^3 - bN_1N_2 - cN_2^2 - d, \\ N_1 + N_2 &= N \end{aligned} \tag{1.1}$$

where N_1 and N_2 are the population numbers at the lower level and upper level of a two-level laser system with ion–ion nonlinear interaction, respectively. In steady condition, $dN_1/dt = dN_2/dt = 0$, the equation group is nonlinear, its solution has no analytical form. Let us consider the solution of N_1. Equation (1.1) is simplified as Equation (1.2), where x represents N_1:

$$f(x) = 0 \tag{1.2}$$

If x^* is the true root of Equation (1.2) and x_0 is an approximation of x^*, then the linear function through the point $(x_0, f(x_0))$is:

$$L(x) = a_0(x - x_0) + f(x_0) \tag{1.3}$$

where $a_0 = f'(x_0) \neq 0$. If $L(x) \approx f(x)$, then the root of Equation (1.2) can be approximately replaced with the root of $L(x) = 0$, and the new approximate root x_1 is:

$$x_1 = x_0 - \frac{1}{a_0} f(x_0) \tag{1.4}$$

The linear function through the point $(x_1, f(x_1))$ is:

$$L(x) = a_1(x - x_1) + f(x_1) \tag{1.5}$$

If $L(x) \approx f(x)$, then the root of Equation (1.1) can be approximately replaced with the root of $L(x) = 0$, and the new approximate root x_2 is:

$$x_2 = x_1 - \frac{1}{a_1} f(x_1) \tag{1.6}$$

where $a_1 = f'(x_1) \neq 0$. Generally:

$$x_{k+1} = x_k - \frac{1}{a_k} f(x_k), \qquad a_k \neq 0, \quad k = 0, 1, \ldots \tag{1.7}$$

where $a_k = f'(x_k) \neq 0$.

Iteration Equation (1.7) is a linearization method for solving Equation (1.2); its geometric idea is to plot a line through the point $(x_k, f(x_k))$, and the crossing point of the line with the x-axis is considered the new approximation of the root of Equation (1.2), as shown in Figure 1.1.

Actually, with different a_k, a different iteration method can be obtained. The method with $a = f'(x_0)$ is called the Newton iteration algorithm. If we plot a tangent of $y = f(x)$ at the point $(x_k, f(x_k))$ and let x_{k+1} be a root of $L(x) = f'(x_k)(x - x_k) + f(x_k) = 0$, then:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \ldots \tag{1.8}$$

When k is sufficiently large, the root x_{k+1} will approach the true root x^* of Equation (1.2).

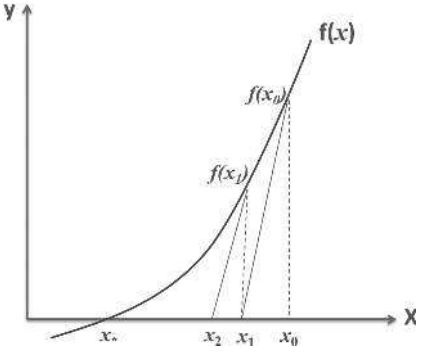

Figure 1.1 Finding procedure of root of nonlinear equation using linearization method.

1.2.2 Multi-Variable

For the equation with n variables $x_1, x_2, \ldots, x_n$:

$$f(x_1, x_2, \ldots, x_n) = 0 \tag{1.9}$$

the solution of the single-variable Equation (1.2) can be used to solve the n-variable equation. It is assumed that $\left(x_1^*, x_2^*, \ldots, x_n^*\right)$ are the true roots of Equation (1.9), and $(x_{10}, x_{20}, \ldots, x_{n0})$ are approximations of the true roots $\left(x_1^*, x_2^*, \ldots, x_n^*\right)$. For simplification, let us use X to represent the n-dimensional variables $(x_1, x_2, \ldots, x_n)$, and use X_0 to represent $(x_{10}, x_{20}, \ldots, x_{n0})$ and X_k to represent $(x_{1k}, x_{2k}, \ldots, x_{nk})$. The linear function through $(X_0, f(X_0))$ is:

$$L_k(X) = A_k(X - X_k) + f(X_k) \tag{1.10}$$

where A_k is an n-order nonsingular matrix; if $L_k(X_k) \approx f(X_k)$, one can use the solution of linear equation group (Equation 1.10) as the approximation of the true root of Equation (1.9).

Let:

$$L_k(X) = A_k(X - X_k) + f(X_k) = 0 \tag{1.11}$$

Then $X = X_{k+1}$ is a new approximation of the true root of Equation (1.9), that is:

$$X_{k+1} = X_k - \frac{f(X_k)}{A_k},\ k = 0, 1, 2, \ldots \tag{1.12}$$

This procedure becomes linear iteration method of the nonlinear equation group $f(X) = 0$. Generally, a different iteration method with different A_k is used. The Newton iteration method uses $A_0 = f'(X_0)$, $A_k = f'(X_k)$ and for $f(X) = 0$:

$$X_{k+1} = X_k - \frac{f(X_k)}{f'(X_k)},\ k = 0, 1, 2, \ldots \tag{1.13}$$

This method is a simplified Newton method.

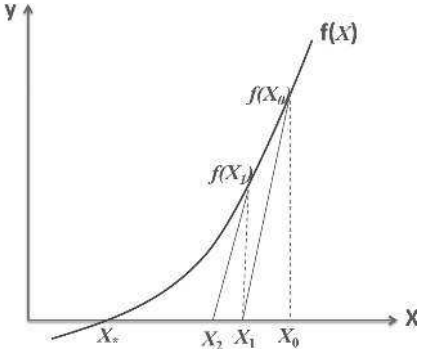

Figure 1.2 Finding procedure of root of multi-variable nonlinear equation using Newton iteration method.

1.3 Runge–Kutta Algorithm for Power-Propagation Equation Solution

1.3.1 Single-Function [2]

The Runge–Kutta algorithm is used to numerically solve differential equations. It is known to be very accurate for a wide range of problems. Consider a single-variable problem:

$$\begin{aligned} &\frac{dy}{dx} = f(x,y), a \le x \le b \\ &y(a) = y_0 \end{aligned} \tag{1.14}$$

with initial condition $y(a) = y_0$, and suppose that y_n is the value of function at the variable x_n. The Runge–Kutta formula takes y_n and x_n to calculate an approximation of y_{n+1} at x_{n+h} by using a weighted average of approximate values of $f(x,y)$ within the interval (x_n, x_{n+h}).

$y(x)$ can be expanded using Taylor series:

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + \frac{(x-x_0)^2}{2}y''(x_0) + \frac{(x-x_0)^3}{6}y'''(x_0) + \dots \tag{1.15}$$

where $y'(x) = dy/dx, y''(x) = d^2y/dx^2, y'''(x) = d^3y/dx^3$.

Equation (1.14) is inserted into Equation (1.15) and Equation (1.16) is obtained:

$$y(x) \approx y(x_0) + (x - x_0)y'(x_0) = y(x_0) + (x - x_0)\, f(x_0, y_0) \tag{1.16}$$

The numerical approximation of Equation (1.16) is:

$$y_1 = y_0 + hf(x_0, y_0) \tag{1.17}$$

where $h = x_1 - x_0$ is a step length, $y_2 = y_1 + hf(x_1, y_1)$, and generally:

$$y_k = y_{k-1} + hf(x_{k-1}, y_{k-1}) \tag{1.18}$$

Thus, y_k can be obtained from y_0 and $f(x_0, y_0)$, and this method is called the Euler method.

According to the mean value theorem of differentials:

$$y(x_{k+1}) - y(x_k) = y'(\xi)(x_{k+1} - x_k) \tag{1.19}$$

Equation (1.20) can be obtained from Equation (1.18):

$$y(x_{k+1}) = y(x_k) + hf(\xi, y(\xi)) \tag{1.20}$$

where $h = x_{k+1} - x_k$, $f(\xi, y(\xi)) = S^*$ is the mean slope of $y(x)$ in the range $[x_k, x_{k+1}]$.

If $f(x_k, y(x_k)) \approx f(x_k, y_k) = S_1$ is an approximation of S^*, then $y_{k+1} = y_k + hS_1$, and this is the first-order Euler equation.

Assume that $f(x_{k+1}, y(x_{k+1})) \approx f(x_k + h, y_k + hf(x_k, y_k)) = S_2$, and the weighted average values of S_1 and S_2 are used as the approximation of S^*, then:

$$\begin{aligned} y_{k+1} &= y_k + \frac{1}{2}h(S_1 + S_2), \\ S_1 &= f(x_k, y_k), \\ S_2 &= f(x_k + h, y_k + hS_1) \end{aligned} \tag{1.21}$$

This is the second-order Euler equation.

Generally, if there are m slope values, such as $S_1, S_2, S_3, \ldots, S_m$ in the range $[x_k, x_{k+1}]$, then:

$$\begin{aligned} y_{k+1} &= y_k + h(\alpha_1 S_1 + \alpha_2 S_2 + \ldots + \alpha_m S_m), \\ S_1 &= f(x_k, y_k), \\ S_2 &= f(x_k + \beta_2 h, y_k + \gamma_2 h) \\ &\ldots \\ S_m &= f(x_k + \beta_m h, y_k + \gamma_m h) \end{aligned} \tag{1.22}$$

This is a general form of the Runge–Kutta algorithm.

Where $0 \le \lambda_k \le 1, y_k + \gamma_k h$ is an approximate value of $y(x_k + \beta_m h)$, and $\alpha_k, \beta_k, \gamma_k$ are undetermined coefficients.

For the third-order Runge–Kutta algorithm:

$$\begin{aligned} y_{k+1} &= y_k + \frac{1}{6}(S_1 + 4S_2 + S_3)h \\ S_1 &= f(x_k, y_k) \\ S_2 &= f\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}S_1 h\right) \\ S_3 &= f(x_k + h, y_k - S_1 h + 2S_2 h) \end{aligned} \tag{1.23}$$

For the fourth-order Runge–Kutta algorithm:

$$\begin{aligned} y_{k+1} &= y_k + \frac{1}{6}(S_1 + 2S_2 + 2S_3 + S_4)h \\ S_1 &= f(x_k, y_k) \\ S_2 &= f\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}S_1 h\right) \\ S_3 &= f\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}S_2 h\right) \\ S_4 &= f(x_k + h, y_k + S_3 h) \end{aligned} \tag{1.24}$$

To run the simulation, it can be started with y_0, and y_1 can be found using the above formula, then y_1 is plugged in to find y_2, and so on.

To solve the power-propagation equations of rare-earth-doped fiber and waveguide systems, y represents pump power, signal power, or amplified spontaneous emission power, and x represents the propagation distance z.

1.3.2 Multi-Functions

With multiple functions, the Runge–Kutta algorithm is similar to the above equations, except that the functions become a vector.

Suppose there are m functions $y_1, y_2, \ldots, y_m$, each of which varies with same variable x. Suppose further that there are m coupled differential equations for these m functions:

$$\begin{aligned}
\frac{dy_1}{dx} &= f_1(x, y_1, y_2, \ldots, y_m) \\
\frac{dy_2}{dx} &= f_2(x, y_1, y_2, \ldots, y_m) \\
&\cdots\cdots\cdots\cdots\cdots\cdots \\
\frac{dy_m}{dx} &= f_n(x, y_1, y_2, \ldots, y_m)
\end{aligned} \tag{1.25}$$

Note there are no derivatives on the right-hand side of those equations, and there are only first derivatives on the left-hand side. These equations can be summarized in vector form as:

$$\frac{dY}{dx} = F(x, Y) \tag{1.26}$$

where $Y = (y_1, y_2, \ldots, y_m)$. Next, let us label the states y_m, y_{m+1}, which are separated by the interval h of variable x. That is, y_m is the value of the function at the variable x_m, and y_{1m} is the value of the first function y_1 at x_m:

$$Y_m = (y_{1,m}, y_{2,m}, \ldots, y_{m,m}) \tag{1.27}$$

$$Y_{m+1} = (y_{1,m+1}; y_{2,m+1} \ldots, y_{m,m+1}) \tag{1.28}$$

To compute the state at a short length h and put the results into x_{m+1}, the fourth-order Runge–Kutta algorithm does the following [2]:

$$Y_{m+1} = Y_m + \frac{h}{6}(S_{1,m} + 2S_{2,m} + 2S_{3,m} + S_{4,m}) \tag{1.29}$$

where:

$$S_{1,m} = F(x_m, Y_m),$$

$$S_{2,m} = F\left(x_m + \frac{h}{2}, Y_m + \frac{h}{2}a_m\right),$$

$$S_{3,m} = F\left(x_m + \frac{h}{2}, Y_m + \frac{h}{2}b_m\right),$$

$$S_{4,m} = F(x_m + h, Y_m + hc_m)$$

The new vector Y_{m+1} gives the values after variable x has passed the small length h.

To solve power-propagation equations of rare-earth-co-doped fiber and waveguide systems, Y represents pump power, signal power, and amplified spontaneous emission power and x represents propagation distance z.

Matlab codes of the Newton iteration method and Runge–Kutta algorithm are attached in the Appendix.

1.4 Two-Point Boundary Problem for Power-Propagation Equations in a Laser Cavity

1.4.1 Principle [3]

In solid and fiber lasers, the pump power, lasing power, and amplified spontaneous emission power-propagation equations form a coupled differential equation group. The pump powers at the two ends of the laser cavity are known and the lasing power at the output end is unknown; thus, this problem is a standard two-point boundary problem.

We need to solve a set of m coupled first-order differential equations, meeting m_1 boundary conditions at starting point z_1, and another set of $m_2 = m - m_1$ boundary conditions at the final point z_2. The differential equation group can be written as follows:

$$\begin{aligned} \frac{dy_1}{dz} &= f_1(z, y_1, y_2, \ldots, y_m) \\ \frac{dy_2}{dz} &= f_2(z, y_1, y_2, \ldots, y_m) \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ \frac{dy_m}{dz} &= f_m(z, y_1, y_2, \ldots, y_m) \end{aligned} \tag{1.30}$$

At z_1, the solution is supposed to meet:

$$B_{1i}(z_1, y_1, y_2, \ldots, y_m) = a_1, a_2, \ldots, a_{m1}, i = 1, \ldots, m_1 \tag{1.31}$$

At z_2, the solution is supposed to meet:

$$B_{2j}(z_2, y_1, y_2, \ldots, y_m) = b_1, b_2, \ldots, b_{m2}, j = 1, \ldots, m_2 \tag{1.32}$$

1.4.2 Shooting Method and Relaxation Method [3]

Shooting and relaxation methods are usually used to solve the two-point boundary problem. One of methods for two-point boundary problem solution is the method which is used to solve numerically the differential equation of initial value problem and the nonlinear algebraic equation. This method is called shooting method. In the shooting method, we can choose values for all dependent variables at one boundary; these values must be consistent with any boundary conditions. The ordinary differential equations are solved by integrating and arriving at the other boundary by using the initial-value method. In general, the difference from the desired boundary value can be found, then we meet a multidimensional root-finding problem, which can be solved by using the

Newton iteration algorithm introduced in the previous section: the variables are adjusted at a boundary point to reduce the difference at the other boundary points. If the differential equations are integrated to follow the trajectory of a shot from gun to a target, then picking the initial conditions corresponds to aiming in Figure 1.3. The shooting algorithm provides a systematic approach to taking a set of ranging shots that allow us to improve our aim systematically.

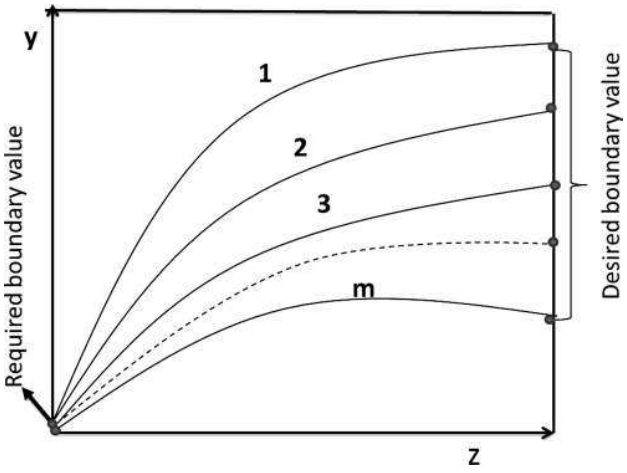

Figure 1.3 Schematic diagram of shooting algorithm for solving *m* coupled differential equations. Trial integration meeting the boundary condition at one endpoint is started. The difference from the desired boundary condition at the final endpoint is used to adjust the starting conditions, until the boundary conditions at the two endpoints are finally met.

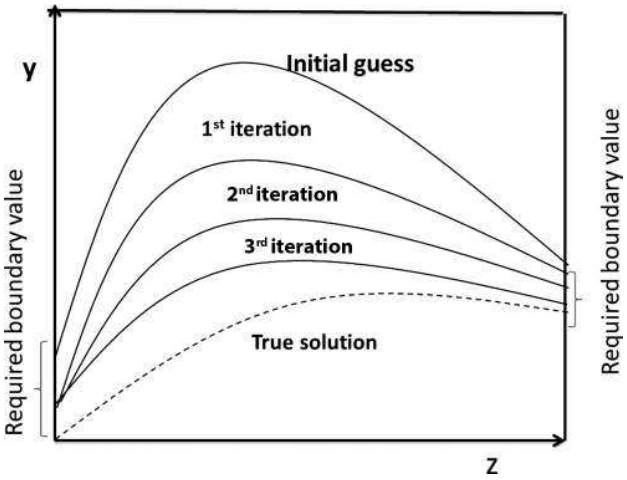

Figure 1.4 Schematic diagram of relaxation algorithm for solving *m* coupled differential equations. A set of initial values is guessed that meets the differential equations and boundary conditions. An iteration process is used to adjust the function to make it close the true solution.

The relaxation algorithm uses a differential approach. The differential equations become finite-difference equations on a mesh of points that covers the range of the integration. A trial solution consists of values for the dependent variable at each mesh point, not meeting the desired finite-difference equations, nor necessarily even meeting the required boundary conditions. The iteration, which is called relaxation, consists of adjusting all values on the mesh in order to bring them into closer agreement with the finite-difference equations and simultaneously with the boundary conditions shown in Figure 1.4. In Matlab, the function bvp4c( ) usually is used to solve the two-point boundary problem described with the equations (1.30)-(1.32). It is assumed that S(z) is an approximate solution of the equations (1.30)-(1.32) and is a continuous function and cubic polynomial at each subrange $[z_n,z_{n+1}]$ in the range $0=z_0<z_1<z_2<\ldots<z_n=L$, and meet the boundary condition:

$$B(S(0), S(L)) = 0;$$

and at end of each subrange, the below differential equations are met:

$$S^{'}(z_n) = f(z_n, S(z_n)),$$

$$S^{'}(z_n + z_{n+1})/2) = f((z_n + z_{n+1})/2), S(z_n + z_{n+1})/2)),$$

$$S^{'}(z_{n+1}) = f(z_{n+1}, S(z_{n+1})).$$

The residual error of the ordinary differential equation is expressed with

$$r(z) = S^{'}(z) - f(z, S(z))$$

Various high-order differential equations can be converted to the first-order differential equations by variable conversion, which can be solved with the function bvp4c( ).

If the true solutions of differential equations are smooth and not highly oscillatory, then the relaxation method works best. If the true solutions are highly oscillatory, the shooting method usually is more efficient because its variable step size of integration can adjust naturally.

Matlab codes of the two-point boundary problem for solid laser and fiber laser are attached in the Appendix.

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, & Brian P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing*, 2nd edn, Cambridge University Press, Cambridge, 2001, 366–72.

[2] William H. Press, Saul A. Teukolsky, William T. Vetterling, & Brian P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing*, 2nd edn, Cambridge University Press, Cambridge, 2001, 715–18.

[3] William H. Press, Saul A. Teukolsky, William T. Vetterling, & Brian P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing*, 2nd edn, Cambridge University Press, Cambridge, 2001, 756–8.