Chapter d02 – Ordinary Differential Equations

1. Scope of the Chapter

This chapter provides functions for the numerical solution of ordinary differential equations. There are two main types of problem: initial-value problems where all boundary conditions are specified at one point, and boundary-value problems where the boundary conditions are distributed between two or more points.

2. Background

For the functions in this chapter a system of ordinary differential equations must be written in the form

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \ldots, y_n), \\ y_2' &= f_2(x, y_1, y_2, \ldots, y_n), \\ &\vdots \\ y_n' &= f_n(x, y_1, y_2, \ldots, y_n), \end{aligned}$$

that is the system must be given in first-order form. The n dependent variables (also, the solution) $y_1, y_2, \ldots, y_n$ are functions of the independent variable x, and the differential equations give expressions for the first derivatives $y_i' = \frac{dy_i}{dx}$ in terms of x and $y_1, y_2, \ldots, y_n$. For a system of n first-order ordinary differential equations, n associated boundary conditions are usually required to define the solution.
A more general system may contain derivatives of higher order, but such systems can almost always be reduced to the first-order form by introducing new variables. For example, taking the third-order equation

$$z''' + zz'' + k(l - z'^2) = 0$$

and writing $y_1 = z$, $y_2 = z'$ and $y_3 = z''$ we can obtain the first-order system

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ y_3' &= -y_1 y_3 - k(l - y_2^2) \end{aligned}$$

For this system $n = 3$ and we require 3 boundary conditions. These conditions must specify values of the dependent variables at certain points. For example, we have an **initial-value problem** if the conditions are:

$$\begin{aligned} y_1 &= 0 \quad \text{at} \quad x = 0 \\ y_2 &= 0 \quad \text{at} \quad x = 0 \\ y_3 &= 2 \quad \text{at} \quad x = 0 \end{aligned}$$

These conditions would enable us to integrate the equations numerically from the point $x = 0$ to some specified end-point. We have a **boundary-value problem** if the conditions are:

$$\begin{aligned} y_1 &= 0 \quad \text{at} \quad x = 0 \\ y_2 &= 0 \quad \text{at} \quad x = 0 \\ y_2 &= 1 \quad \text{at} \quad x = 90 \end{aligned}$$

These conditions could be sufficient to define a solution in the range $0 \leq x \leq 90$ if it exists (see Section 2.2).

2.1. Initial-value Problems

To solve first-order systems, initial values of the dependent variables $y_1, y_2, \ldots, y_n$ must be supplied at a given point a. Also a point b, at which the dependent variables are required, must be specified. The numerical solution is obtained by a step-by-step calculation which approximates the variables

over the range $[a, b]$. The step-length is adjusted automatically to meet specified accuracy tolerances. The accuracy tests are reliable over each individual step but cannot generally be guaranteed over a long range. For many problems there may be no serious accumulation of error, but for unstable systems small perturbations of the solution may lead to rapid divergence of the calculated values from the true values. A simple check is to carry out trial calculations with different tolerances; if the results differ appreciably then the system is probably unstable.

A special class of initial-value problems are those for which the solutions contain rapidly decaying transient terms. Such problems are called **stiff** and require special methods for efficient numerical solution; the methods designed for non-stiff problems when applied to stiff problems tend to be very slow because they need small step-lengths to avoid numerical instability.

In general, for non-stiff first order systems, Runge–Kutta (RK) methods should be used. For the usual requirement of integrating across a range the appropriate functions are nag_ode_ivp_rk_setup (d02pvc) and nag_ode_ivp_rk_range (d02pcc); nag_ode_ivp_rk_setup (d02pvc) is a set up function for nag_ode_ivp_rk_range (d02pcc). For more complex tasks there a further four functions nag_ode_ivp_rk_onestep (d02pdc), nag_ode_ivp_rk_reset_tend (d02pwc), nag_ode_ivp_rk_interp (d02pxc) and nag_ode_ivp_rk_errass (d02pzc). These respectively integrate one step, reset the end of the integration range, perform interpolation and finally supply information concerning error assessment.

When a system is to be integrated over a long range or with relatively high accuracy requirements the variable-order, variable-step Adams method may be more efficient. There is an easy-to-use function nag_ode_ivp_adams_gen (d02cjc) to perform integration over a range; intermediate output in the range can be obtained and the first position where a single function of the solution is first zero can be computed. There is a more flexible function nag_ode_ivp_adams_roots (d02qfc) for general integration purposes; all the positions of zeros of functions of the solution can be computed. An initialisation function nag_ode_ivp_adams_setup (d02qwc) and an interpolation function nag_ode_ivp_adams_interp (d02qzc) to compute the solution at non-step points are provided for use with nag_ode_ivp_adams_roots (d02qfc).

For stiff systems a Backward Differentiation Formula (BDF) variable-order, variable step method should be used. The function nag_ode_ivp_bdf_gen (d02ejc) performs integration over an interval until a user-specified function ($g(x, y)$), if supplied, is zero. The routine also permits the user to define an output routine in order to return the solution at specific points. Its argument list is similar to nag_ode_ivp_adams_gen (d02cjc) except that to solve the equations arising in the BDF method an approximation to the Jacobian $\left(\frac{\partial F_i}{\partial y_j}\right)$ is required. This approximation can be calculated internally but the user may supply an analytic expression. In most cases supplying a correct analytic expression will reduce the amount of computer time used.

2.2. Boundary-value Problems

In general, a system of nonlinear differential equations with boundary conditions given at two or more points cannot be guaranteed to have a solution. The solution has to be determined iteratively if it exists. The functions available in this chapter for this type of problem use finite difference methods. Finite difference equations are set up on a mesh of points and estimated values for the solution on the grid points are chosen. These estimates are used as starting values for a Newton iteration to solve the finite difference equations. The accuracy of the solution is then improved by deferred corrections or the addition of points to the mesh or a combination of both. In some cases the method may require good initial estimates of the solution. The method is unlikely to be successful when the solution varies rapidly over short ranges.

There is an easy-to-use function nag_ode_bvp_fd_nonlin_fixedbc (d02gac) for simple boundary-value problems with assigned boundary values. However, users may find convergence difficult to achieve with nag_ode_bvp_fd_nonlin_fixedbc (d02gac) and instead use the more general purpose function nag_ode_bvp_fd_nonlin_gen (d02rac). nag_ode_bvp_fd_nonlin_gen (d02rac) permits an initial estimate of the solution at all mesh points, caters for more general nonlinear boundary conditions, provides for analytically supplied Jacobians and allows the calculation to be influenced in other ways too. There is also the function nag_ode_bvp_fd_lin_gen (d02gbc) for the general linear two-point boundary-value problem written in standard 'textbook' form. The user is advised to employ interpolation functions from the e01 Chapter to obtain solution values at points not on the final mesh.

3. Available Functions

Integrate system over a range, possibly with intermediate output and root-finding for a single function of the solution, Adams method	`d02cjc`
Integrate a stiff system (using BDF) over a range, with optimal intermediate output and root-finding	`d02ejc`
Solve a system of equations with boundary conditions for simple nonlinear problem	`d02gac`
Solve a system of equations with boundary conditions for general linear problem	`d02gbc`
Integrate a system of equations over a range, using the Runge–Kutta method	`d02pcc`
Integrate a system of equations over single integration step	`d02pdc`
Memory deallocation function for the Runge–Kutta method	`d02ppc`
Set-up function for d02pcc or d02pdc	`d02pvc`
Reset the end-point in an integration performed by d02pdc	`d02pwc`
Computes the solution of a system of ODE using interpolation on an integration step taken by d02pdc	`d02pxc`
Computes details of global error assessment when using d02pcc or d02pdc	`d02pzc`
Integrate a system over a range, possibly determining all the positions where functions of the solution are zero, Adams method	`d02qfc`
Set-up function for d02qfc	`d02qwc`
Memory deallocation function for the Adams method	`d02qyc`
Interpolation function for d02qfc	`d02qzc`
Solve a system of equations with boundary conditions for the general nonlinear problem with a continuation facility	`d02rac`