NAG Library Chapter Introduction

d02 – Ordinary Differential Equations

Contents

1 Scope of the Chapter

This chapter is concerned with the numerical solution of ordinary differential equations. There are two main types of problem: those in which all boundary conditions are specified at one point (initial value problems), and those in which the boundary conditions are distributed between two or more points (boundary value problems and eigenvalue problems). Functions are available for initial value problems and two-point boundary value problems.

2 Background to the Problems

For most of 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 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, suppose we have the third-order equation

$$z''' + zz'' + k\left(l - z'^2\right) = 0.$$

We write $y_1 = z$, $y_2 = z'$, $y_3 = z''$, and the third-order equation may then be written as the system of first-order equations

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

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

$$\begin{aligned} y_1 &= 0 \quad \text{at } x = 0 \\ y_2 &= 0 \quad \text{at } x = 0 \\ y_3 &= 0.1 \quad \text{at } x = 0. \end{aligned}$$

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

$$\begin{aligned} y_1 &= 0 \quad \text{at } x = 0 \\ y_2 &= 0 \quad \text{at } x = 0 \\ y_2 &= 1 \quad \text{at } x = 10. \end{aligned}$$

These conditions would be sufficient to define a solution in the range $0 \leq x \leq 10$, but the problem could not be solved by direct integration (see Section 2.2). More general boundary conditions are permitted in the boundary value case.

2.1 Initial Value Problems

To solve first-order systems, initial values of the dependent variables y_i, for $i = 1, 2, \ldots, n$, must be supplied at a given point, a. Also a point, b, at which the values of the dependent variables are required, must be specified. The numerical solution is then obtained by a step-by-step calculation which approximates values of the variables y_i, for $i = 1, 2, \ldots, n$, at finite intervals over the required range $[a, b]$. The functions in this chapter adjust the step length automatically to meet specified accuracy tolerances. Although the accuracy tests used are reliable over each step individually, in general an accuracy requirement cannot 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 will often lead to rapid divergence of the calculated values from the true values. A simple check for stability is to carry out trial calculations with different tolerances; if the results differ appreciably the system is probably unstable. Over a short range, the difficulty may possibly be overcome by taking sufficiently small tolerances, but over a long range it may be better to try to reformulate the problem.

A special class of initial value problems are those for which the solutions contain rapidly decaying transient terms. Such problems are called **stiff**; an alternative way of describing them is to say that certain eigenvalues of the Jacobian matrix $\left(\frac{\partial f_i}{\partial y_j}\right)$ have large negative real parts when compared to others. These problems 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. A full discussion is given in Hall and Watt (1976) and a discussion of the methods for stiff problems is given in Berzins *et al.* (1988).

2.2 Boundary Value Problems

In general, a system of nonlinear differential equations with boundary conditions at two or more points cannot be guaranteed to have a solution. The solution, if it exists, has to be determined iteratively. A comprehensive treatment of the numerical solution of boundary value problems can be found in Ascher *et al.* (1988) and Keller (1992). The methods for this chapter are discussed in Ascher *et al.* (1979), Ascher and Bader (1987) and Gladwell (1987).

2.2.1 Collocation methods

In the collocation method, the solution components are approximated by piecewise polynomials on a mesh. The coefficients of the polynomials form the unknowns to be computed. The approximation to the solution must satisfy the boundary conditions and the differential equations at collocation points in each mesh sub-interval. A modified Newton method is used to solve the nonlinear equations. The mesh is refined by trying to equidistribute the estimated error over the whole interval. An initial estimate of the solution across the mesh is required.

2.2.2 Finite difference methods

If a boundary value problem seems insoluble by the above method and a good estimate for the solution of the problem is known at all points of the range then a finite difference method may be used. Finite difference equations are set up on a mesh of points and estimated values for the solution at the grid points are chosen. Using these estimated values as starting values a Newton iteration is used 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. Good initial estimates of the solution may be required in some cases and the method is unlikely to be successful when the solution varies very rapidly over short ranges. A discussion is given in Chapters 9 and 11 of Gladwell and Sayers (1980) and Chapter 4 of Gladwell (1979a).

3 Recommendations on Choice and Use of Available Functions

There are no functions which deal directly with complex equations. These may however be transformed to larger systems of real equations of the required form. Split each equation into its real and imaginary parts and solve for the real and imaginary parts of each component of the solution. Whilst this process doubles the size of the system and may not always be appropriate it does make available for use the full range of functions provided presently.

3.1 Initial Value Problems

In general, for non-stiff first-order systems, Runge–Kutta (RK) functions should be used. For the usual requirement of integrating across a range the appropriate functions are nag_ode_ivp_rkts_range (d02pec) and nag_ode_ivp_rkts_setup (d02pqc); nag_ode_ivp_rkts_setup (d02pqc) is a setup function for nag_ode_ivp_rkts_range (d02pec). For more complex tasks there are forward and reverse communication variants (Section 2.3.2 in How to Use the NAG Library and its Documentation) of single step functions with corresponding interpolator; for direct communication these are nag_ode_ivp_rkts_onestep (d02pfc) and nag_ode_ivp_rkts_interp (d02psc), while for reverse communication these are nag_ode_ivp_rk_step_revcomm (d02pgc), nag_ode_ivp_rk_interp_setup (d02phc) and nag_ode_ivp_rk_interp_eval (d02pjc). There are also related utility functions nag_ode_ivp_rkts_reset_tend (d02prc), nag_ode_ivp_rkts_diag (d02ptc) and nag_ode_ivp_rkts_errass (d02puc). When a system is to be integrated over a long range or with relatively high accuracy requirements the variable-order, variable-step Adams' codes may be more efficient. The appropriate function in this case is nag_ode_ivp_adams_gen (d02cjc). For more complex tasks using an Adams' code there are a further four related functions: nag_ode_ivp_adams_roots (d02qfc), nag_ode_ivp_adams_setup (d02qwc), nag_ode_ivp_adams_free (d02qyc) and nag_ode_ivp_adams_interp (d02qzc).

For stiff systems, that is those which usually contain rapidly decaying transient components, the Backward Differentiation Formula (BDF) variable-order, variable-step codes should be used. The appropriate functions in this case are: the simple driver nag_ode_ivp_bdf_gen (d02ejc), or the more comprehensive nag_dae_ivp_dassl_gen (d02nec), based on the DASSL implementation (see Brenan *et al.* (1996)), and its related functions nag_dae_ivp_dassl_cont (d02mcc), nag_dae_ivp_dassl_setup (d02mwc) and nag_dae_ivp_dassl_linalg (d02npc).

If you are not sure how to classify a problem, you are advised to perform some preliminary calculations with nag_ode_ivp_rkts_range (d02pec), which can indicate whether the system is stiff. We also advise performing some trial calculations with nag_ode_ivp_rkts_range (d02pec) (RK), nag_ode_ivp_adams_gen (d02cjc) (Adams) and nag_ode_ivp_bdf_gen (d02ejc) (BDF) so as to determine which type of function is best applied to the problem. The conclusions should be based on the computer time used and the number of evaluations of the derivative function f_i. See Gladwell (1979b) for more details.

3.1.1 Runge–Kutta functions

The basic RK functions are nag_ode_ivp_rkts_onestep (d02pfc) (direct communication) and nag_ode_ivp_rk_step_revcomm (d02pgc) (reverse communication) which take one integration step at a time. An alternative to nag_ode_ivp_rkts_onestep (d02pfc) is nag_ode_ivp_rkts_range (d02pec), which provides output at user-specified points. The initialization of nag_ode_ivp_rkts_range (d02pec), nag_ode_ivp_rkts_onestep (d02pfc) or nag_ode_ivp_rk_step_revcomm (d02pgc) and the setting of optional inputs, including choice of method, is made by a call to the setup function nag_ode_ivp_rkts_setup (d02pqc). Optional output information about error assessment, can be obtained by calls to the function nag_ode_ivp_rkts_errass (d02puc) while integration statistics are returned by the diagnostic function nag_ode_ivp_rkts_diag (d02ptc). nag_ode_ivp_rkts_interp (d02psc) may be used to interpolate on information produced by nag_ode_ivp_rkts_onestep (d02pfc) to give solution and derivative values between the integration points. Similarly nag_ode_ivp_rk_interp_setup (d02phc) may be used to setup an interpolator on information produced by nag_ode_ivp_rk_step_revcomm (d02pgc), and nag_ode_ivp_rk_interp_eval (d02pjc) can evaluate that interpolator to give solution and derivative values between integration points; these functions are recommended when a high-order RK method is specified in the setup function. nag_ode_ivp_rkts_reset_tend (d02prc) may be used to reset the end of the integration range whilst integrating using nag_ode_ivp_rkts_onestep (d02pfc) or nag_ode_ivp_rk_step_revcomm (d02pgc).

3.1.2 Adams' functions

The general Adams' variable-order variable-step function is nag_ode_ivp_adams_roots (d02qfc), which provides a choice of automatic error control and the option of a sophisticated root-finding technique. The initialization of nag_ode_ivp_adams_roots (d02qfc) and the setting of optional inputs is made by a call to the setup function nag_ode_ivp_adams_setup (d02qwc). nag_ode_ivp_adams_interp (d02qzc) may be used to interpolate on information produced by nag_ode_ivp_adams_roots (d02qfc) to give solution and derivative values between the integration points.

There is a simple driving function nag_ode_ivp_adams_gen (d02cjc), which integrates a system over a range and, optionally, computes intermediate output and/or determines the position where a specified function of the solution is zero.

3.1.3 BDF functions

General functions for implicit ordinary differential equations with options for forms of numerical linear algebra are provided by a suite of functions using the DASSL implementation (see Brenan *et al.* (1996)) of the BDF. The main solver in this suite is nag_dae_ivp_dassl_gen (d02nec) is designed for solving systems of the form,

$$F(t, y, y') = 0.$$

These formulations permits solution of differential/algebraic systems (DAEs). Additionally nag_dae_ivp_dassl_gen (d02nec) can be used to solve difficult algebraic problems by continuation; for example, the nonlinear algebraic problem

$$f(x) = 0$$

can be solved by integrating solutions of

$$f(x) + (1 - t)g(x) = 0$$

where the solution to

$$f(x) + g(x) = 0$$

is known.

Options for the solver must be supplied as arguments in an initial call to the setup function nag_dae_ivp_dassl_setup (d02mwc); the Jacobian of the system to be solved can be considered to have a banded structure by a call to nag_dae_ivp_dassl_linalg (d02npc) prior to calling the solver function; and integration can be continued by a call to nag_dae_ivp_dassl_cont (d02mcc) between calls to the solver nag_dae_ivp_dassl_gen (d02nec).

There is a simple driving function nag_ode_ivp_bdf_gen (d02ejc), which integrates a system over a range and, optionally, computes intermediate output and/or determines the position where a specified function of the solution is zero. To solve the equations arising in the BDF method an approximation to the Jacobian $\left(\dfrac{\partial f_i}{\partial y_j}\right)$ is required. This approximation can be calculated internally but you may supply an analytic expression. In most cases supplying a correct analytic expression will reduce the amount of computer time used.

3.2 Boundary Value Problems

In general, for a nonlinear system of mixed order with separated boundary conditions, the collocation method (nag_ode_bvp_coll_nlin_solve (d02tlc) and its associated functions) can be used. Problems of a more general nature can often be transformed into a suitable form for treatment by nag_ode_bvp_coll_nlin_solve (d02tlc), for example nonseparated boundary conditions or problems with unknown parameters (see Section 9 in nag_ode_bvp_coll_nlin_setup (d02tvc) for details).

For simple boundary value problems with assigned boundary values you may prefer to use a code based on the finite difference method for which there is a function with simple calling sequence (nag_ode_bvp_fd_nonlin_fixedbc (d02gac)).

For difficult boundary value problems, where you need to exercise some control over the calculation, and where the collocation method proves unsuccessful, you may wish to try the alternative method of finite differences (nag_ode_bvp_fd_nonlin_gen (d02rac)).

Note that it is not possible to make a fully automatic boundary value function, and you should be prepared to experiment with different starting values or a different function if the problem is at all difficult.

3.2.1 Finite difference methods

nag_ode_bvp_fd_nonlin_fixedbc (d02gac) may be used for simple boundary value problems with assigned boundary values.

You may find that convergence is difficult to achieve using nag_ode_bvp_fd_nonlin_fixedbc (d02gac) since only specifying the unknown boundary values and the position of the finite difference mesh is permitted. In such cases you may use nag_ode_bvp_fd_nonlin_gen (d02rac), which permits specification of an initial estimate for the solution at all mesh points and allows the calculation to be influenced in other ways too. nag_ode_bvp_fd_nonlin_gen (d02rac) is designed to solve a general nonlinear two-point boundary value problem with nonlinear boundary conditions.

A function, nag_ode_bvp_fd_lin_gen (d02gbc), is also supplied specifically for the general linear two-point boundary value problem written in a standard 'textbook' form.

You are advised to use interpolation functions from Chapter e01 to obtain solution values at points not on the final mesh.

3.2.2 Chebyshev integration method

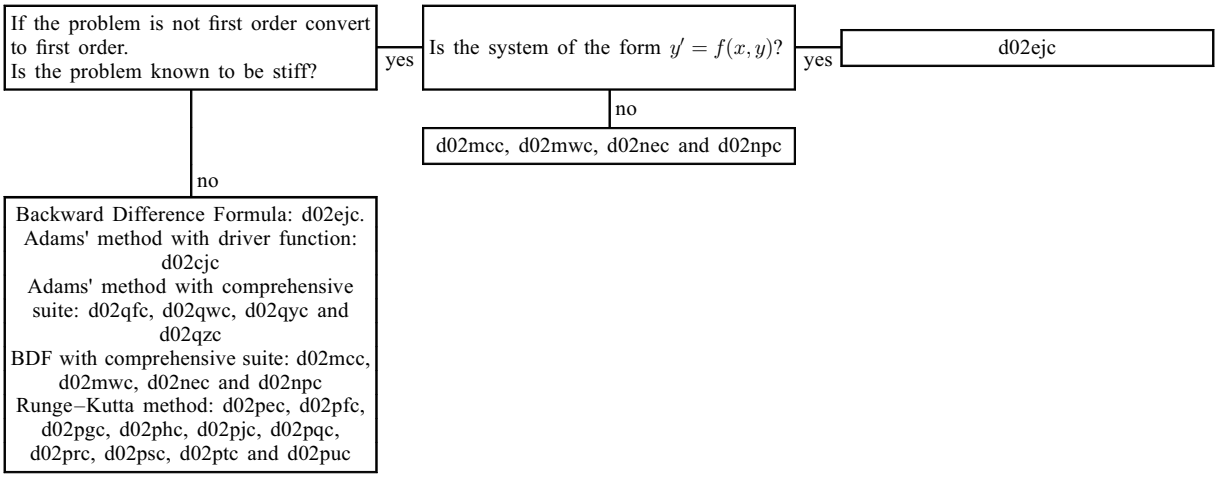
The Chebyshev integration method is an implementation of the Chebyshev collocation method which is fully described and compared against other implementations in Muite (2010). nag_ode_bvp_ps_lin_solve (d02uec) solves a linear constant coefficient boundary value problem using the Chebyshev integration formulation on a Chebyshev Gauss–Lobatto grid and solving in the coefficient space. The required Chebyshev Gauss–Lobatto grid points on a given arbitrary interval $[a, b]$ can first be generated using nag_ode_bvp_ps_lin_cgl_grid (d02ucc). Then nag_ode_bvp_ps_lin_coeffs (d02uac) obtains the Chebyshev coefficients for the right-hand side (of system) function discretized on the obtained Chebyshev Gauss–Lobatto grid. nag_ode_bvp_ps_lin_solve (d02uec) then solves the problem in Chebyshev coefficient space using the integration formulation. Finally nag_ode_bvp_ps_lin_cgl_vals (d02ubc) evaluates the solution (or one of its lower order derivatives) from the set of Chebyshev coefficients returned by nag_ode_bvp_ps_lin_solve (d02uec) on the Chebyshev Gauss–Lobatto grid on $[a, b]$. The set of functions can be used to solve up to fourth order boundary value problems.

3.3 Summary of Recommended Functions

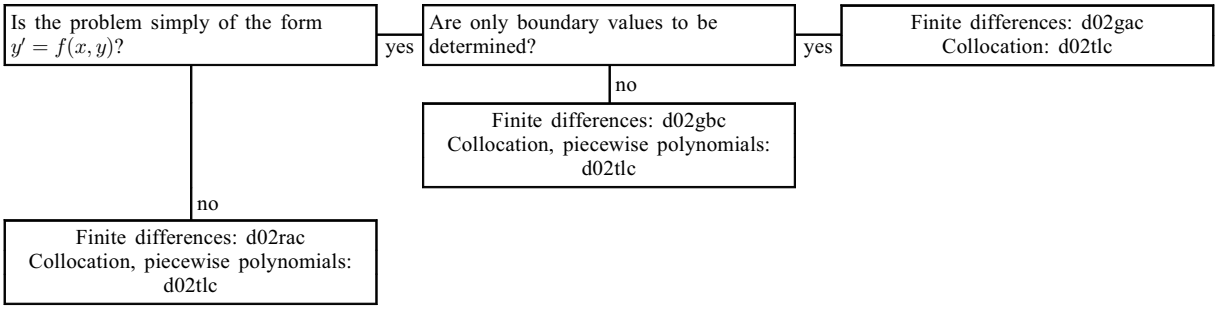
Problem	**Function**		
	RK Method	Adams' Method	BDF Method
Initial Value Problems **Driver Functions**			
Integration over a range with optional intermediate output and optional determination of position where a function of the solution becomes zero		d02cjc	d02ejc
Integration of a range with intermediate output		d02cjc	d02ejc
Integration of a range until function of solution becomes zero		d02cjc	d02ejc
Comprehensive Integration Functions	d02pec, d02pfc, d02pgc, d02phc, d02pjc, d02pqc, d02prc, d02psc, d02ptc and d02puc	d02qfc, d02qwc and d02qzc	d02mcc, d02mwc, d02nec and d02npc
Package for Solving Second-order Systems of Special Form			
Boundary Value Problems **Collocation Method, Mixed Order**	d02tlc, d02tvc, d02txc, d02tyc and d02tzc		
Boundary Value Problems **Finite Difference Method**			
linear problem	d02gbc		
full nonlinear problem	d02rac		
Boundary Value Problems **Chebyshev Collocation, Integration Formulation**			
single linear equation	d02uec with d02uac, d02ubc, d02ucc		

4 Decision Trees

Tree 1: Initial Value Problems



Tree 2: Boundary Value Problems



5 Functionality Index

direct communication ..... nag_ode_ivp_adams_roots (d02qfc)
interpolant ..... nag_ode_ivp_adams_interp (d02qzc)
setup function ..... nag_ode_ivp_adams_setup (d02qwc)
comprehensive integrator functions using Runge–Kutta methods,
diagnostic function ..... nag_ode_ivp_rkts_diag (d02ptc)
diagnostic function for global error assessment ..... nag_ode_ivp_rkts_errass (d02puc)
interpolant, reverse communication ..... nag_ode_ivp_rk_interp_setup (d02phc)
interpolant and interpolation, direct communication ..... nag_ode_ivp_rkts_interp (d02psc)
interpolation, reverse communication ..... nag_ode_ivp_rk_interp_eval (d02pjc)
over a range with intermediate output ..... nag_ode_ivp_rkts_range (d02pec)
over a step, direct communication ..... nag_ode_ivp_rkts_onestep (d02pfc)
over a step, reverse communication ..... nag_ode_ivp_rk_step_revcomm (d02pgc)
reset end of range ..... nag_ode_ivp_rkts_reset_tend (d02prc)
setup function ..... nag_ode_ivp_rkts_setup (d02pqc)
simple driver functions,
variable-order variable-step Adams' method,
until (optionally) a function of the solution is zero, with optional intermediate output
..... nag_ode_ivp_adams_gen (d02cjc)
variable-order variable-step backward differentiation formulae method for stiff systems,
until (optionally) a function of the solution is zero, with optional intermediate output
..... nag_ode_ivp_bdf_gen (d02ejc)

System of ordinary differential equations, boundary value problems,
comprehensive functions using a collocation technique,
continuation function ..... nag_ode_bvp_coll_nlin_contin (d02txc)
diagnostic function ..... nag_ode_bvp_coll_nlin_diag (d02tzc)
general nonlinear problem solver ..... nag_ode_bvp_coll_nlin_solve (d02tlc)
interpolation function ..... nag_ode_bvp_coll_nlin_interp (d02tyc)
setup function ..... nag_ode_bvp_coll_nlin_setup (d02tvc)
finite difference technique with deferred correction,
general linear problem ..... nag_ode_bvp_fd_lin_gen (d02gbc)
general nonlinear problem, with continuation facility ..... nag_ode_bvp_fd_nonlin_gen (d02rac)
simple nonlinear problem ..... nag_ode_bvp_fd_nonlin_fixedbc (d02gac)

6 Auxiliary Functions Associated with Library Function Arguments

None.

7 Functions Withdrawn or Scheduled for Withdrawal

The following lists all those functions that have been withdrawn since Mark 23 of the Library or are scheduled for withdrawal at one of the next two marks.

Withdrawn Function	Mark of Withdrawal	Replacement Function(s)
nag_ode_ivp_rk_range (d02pcc)	26	nag_ode_ivp_rkts_range (d02pec) and associated d02p functions
nag_ode_ivp_rk_onestep (d02pdc)	26	nag_ode_ivp_rkts_onestep (d02pfc) and associated d02p functions
nag_ode_ivp_rk_free (d02ppc)	26	No replacement required
nag_ode_ivp_rk_setup (d02pvc)	26	nag_ode_ivp_rkts_setup (d02pqc)
nag_ode_ivp_rk_reset_tend (d02pwc)	26	nag_ode_ivp_rkts_reset_tend (d02prc)
nag_ode_ivp_rk_interp (d02pxc)	26	nag_ode_ivp_rkts_interp (d02psc)
nag_ode_ivp_rk_errass (d02pzc)	26	nag_ode_ivp_rkts_errass (d02puc)

8 References

Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice–Hall

Berzins M, Brankin R W and Gladwell I (1988) Design of the stiff integrators in the NAG Library *SIGNUM Newsl.* **23** 16–23

Brenan K, Campbell S and Petzold L (1996) *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations* SIAM, Philadelphia

Gladwell I (1979a) The development of the boundary value codes in the ordinary differential equations chapter of the NAG Library *Codes for Boundary Value Problems in Ordinary Differential Equations. Lecture Notes in Computer Science* (eds B Childs, M Scott, J W Daniel, E Denman and P Nelson) **76** Springer–Verlag

Gladwell I (1979b) Initial value routines in the NAG Library *ACM Trans. Math. Software* **5** 386–400

Gladwell I (1987) The NAG Library boundary value codes *Numerical Analysis Report* **134** Manchester University

Gladwell I and Sayers D K (ed.) (1980) *Computational Techniques for Ordinary Differential Equations* Academic Press

Hall G and Watt J M (ed.) (1976) *Modern Numerical Methods for Ordinary Differential Equations* Clarendon Press, Oxford

Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

Muite B K (2010) A numerical comparison of Chebyshev methods for solving fourth-order semilinear initial boundary value problems *Journal of Computational and Applied Mathematics* **234(2)** 317–342

Pryce J D (1986) Error estimation for phase-function shooting methods for Sturm–Liouville problems *IMA J. Numer. Anal.* **6** 103–123