CSC2307H/CSC2322H – The Numerical Solution of ODEs

©
W.H. Enright
Department of Computer Science,
University of Toronto
(enright@cs.utoronto.ca)

Overview and Course Organization

- Overview of problem area
- **CSC2302H: Numerical Methods for IVPs**
  1. Mathematical Setting
  2. General Properties of Numerical Methods
  3. Standard Classes of Methods
     (a) Runge-Kutta
     (b) Multistep
  4. Difficulty of Stiffness
  5. Special Methods for Stiff Problems
  6. Differential/Algebraic Equations
  7. Delay Differential Equations
  8. Validated Numerical Methods
  9. Parallel Methods for IVPs
- **CSC2322H: Numerical Methods for BVPs**
  1. Superposition (For linear problems)
  2. Initial-value based BVP methods
     (a) Shooting methods
  3. Collocation BVP methods
     (a) Derivation
     (b) Mesh selection

4. Finite difference BVP methods
   (a) Derivation
   (b) Extrapolation/deferred correction
5. Runge-Kutta BVP methods
   (a) Derivation
   (b) Implementation
6. Special difficulties
   (a) Convergence for nonlinear problems
   (b) Singular perturbation problems
7. Future topics (to be added later)
   (a) Multiple solutions
   (b) Multipoint

1 Overview of Problem Area

Ordinary differential equations (ODEs) arise in all areas of scientific computation where mathematical simulations of real phenomena are developed and used. Often these models are developed to investigate the evolution of a system over time. Applications are very diverse and include climate modeling, computational biology, computer games, orbit calculations and computational finance.

In order to ensure that the underlying mathematical problem is well-posed we must specify additional constraints (ie. in addition to specifying the ODE). These constraints arise naturally in most applications and are considered part of the specification of the mathematical model. The form of such constraints imposes a classification of ODE problems which is very important in the analysis and approximate solution of the resulting problem. In these notes we will consider different classes of ODEs and focus on initial value problems (IVPs) and on boundary value problems (BVPs). These are two areas where effective software has been developed.

Classification of ODEs:

- **IVP:** An IVP in ODEs is defined by:

$$y' = f(x, y), \;\; y(a) = y_0, \;\; x \in [a, b],$$

  where $y, y_0 \in \Re^n$ and $f : \Re \times \Re^n \to \Re^n$.

  1. A sufficient condition for this IVP to have a unique solution, $y(x)$, is that $f(x, y)$ be continuous on $[a, b] \times \Re^n$ and that it satisfy a Lipschitz condition wrt y.

     Definition: A function $f : \Re \times \Re^n \to \Re^n$ satisfies a Lipschitz condition wrt y iff there exists $L > 0$ such that for all $x \in [a, b]$ and $u, v, \in \Re^n$,
     $\|f(x, u) - f(x, v)\| \leq L\|u - v\|$ for some norm, $\| \;\; \|$.

  2. Higher order systems of ODEs can be reduced to first order systems. For example, $y'' = f(x, y, y')$ can be converted to a first order system of ODEs by letting $z(x) \in \Re^{2n}$ be defined by,

$$z(x) \equiv \begin{bmatrix} z_1(x) \\ z_2(x) \end{bmatrix} \equiv \begin{bmatrix} y(x) \\ y'(x) \end{bmatrix}$$

     We then have,

$$z'(x) = \begin{bmatrix} y'(x) \\ y''(x) \end{bmatrix} = \begin{bmatrix} z_2(x) \\ f(x, z_1(x), z_2(x)) \end{bmatrix} = F(x, z).$$

     Therefore it suffices to consider only numerical methods for first order systems (applies to BVPs as well as IVPs). A consequence is that one approximates $y'(x)$ as well as $y(x)$ whereas a direct second order method may avoid this requirement.

3. Very little can be done (numerically) to exploit linearity (ie. $f(x, y) = A(x)y(x) + h(x)$) where $A(x)$ is an $n \times n$ matrix valued function. (This is not the case for BVPs.)

- **BVP:** A Two Point Boundary Value Problem (TBVP) in ODEs is defined by:

$$y' = f(x, y), \ \ x \in [a, b],$$

with

$$g(y(a), y(b)) = 0, \ \ g : \Re^n \times \Re^n \rightarrow \Re^n.$$

1. Sufficient conditions for this TPBVP to have a solution or for the solution to be unique are very restrictive and not very well understood. Multiple solutions are common. For example, $y'' = y^2$ for $x \in [1, 10]$ with $y(1) = 6, y(10) = .06$ has one solution, $y(x) = 6/x^2$ as well as at least one other 'nearby' solution.
2. Often we consider a TPBVP 'solved' when it can be reduced to an IVP (by determining the 'unknown' vector $y(a)$).
3. Special cases of TPBVPs can often be solved more efficiently or reliably.
   - separated boundary conditions: $g(y(a), y(b)) = 0$ replaced by $g_1(y(a)) = 0, \ \ g_2(y(b)) = 0$.
   - linear problems: $f(x, y) = A(x)y + h(x)$, and $g(y(a), y(b)) = B_a y(a) + B_b y(b)$ or $B_a y(a) = \beta, \ \ B_b y(b) = \alpha$.
   - The theory is better understood for these cases.

- **Generalizations of BVPs**: frequently BVPs that are not TPBVPs arise. We will consider the following generalizations:

1. Eigenvalue problems (parameter determination),

$$y' = f(x, y, \lambda), \ \ \lambda \in \Re^m,$$

   and the number of boundary conditions, $m + n$, is sufficient to resolve the unknowns $y(a)$ and λ (whose solution will reduce the problem to an equivalent IVP).
2. Multipoint Problems – the boundary conditions are specified at more than 2 points (interior constraints are included). These constraints can be specified as

$$h_j(x_j, y(x_j), \lambda) = 0, \ \ \text{for} \ \ j = 1, 2 \cdots J,$$

   where $h_j : \Re \times \Re^n \times \Re^m \rightarrow \Re$.
3. Overdetermined Systems – often arise when sampled solution values are available at a sequence of $x_j's$. Usually only a subset of the solution components are sampled (time series/parameter fitting).

4. Parameter Continuation – we want to determine the solution of,

$$y' = f(x, y, \lambda, \alpha)$$

for a range of $\alpha \in \Re$ (eg. for $\alpha \in [0, 1]$).

2 Numerical Methods for IVPs

2.1 Mathematical Setting:

In the numerical solution of an IVP there are three potential sources of error:

1. Modelling errors (depends on the mathematical model only) which arise when the actual system we are interested in is,

$$u' = g(x, u), \quad u(a) = y_0,$$

were $g()$ is either not completely known or very complicated but $\|g(x, u) - f(x, u)\|$ is small for all (x, u) of interest. That is, $y(x)$ (the exact solution of our IVP satisfies:

$$\begin{aligned} y' &= f(x, y) \\ &= g(x, y) + \beta(x), \quad y(a) = y_0, \end{aligned}$$

where $\beta(x) \equiv f(x, y) - g(x, y)$ is small in norm.

2. Floating Point Errors (depends only on the FP arithmetic) which arise because the IVP is defined by supplying a subroutine, $FCN(x, y, yp)$ to evaluate $f(x, y)$. Each derivative evaluation is computed in FP arithmetic and satisfies:

$$\begin{aligned} yp &= fl(f(x, y)) \\ &= f(x, y) + \alpha(x), \end{aligned}$$

where $\|\alpha(x)\| \leq w\|f\|\mu$, w depends on f and how the subroutine is coded and μ is the unit round-off ($\mu \approx 10^{-15}$ in double precision on IEE FP systems). As a result of these two sources of error, the best we can expect (no matter which numerical method we use), is that the numerical solution, $v(x)$, satisfies:

$$\begin{aligned} v' &= f(x, v) + \alpha(x), \\ &= g(x, v) + \beta(x) + \alpha(x), \quad v(a) = y_0. \end{aligned}$$

3. Discretization or Truncation Error (depends only on the numerical method) and arises because there is no closed form expression for the exact solution of most IVPs and an aproximation must be used. Consider applying a numerical method to approximate the solution of our IVP. It actually appears to the method that

the problem being solved is $v' = f(x,v) + \alpha(x)$, but the user is interested in the solution of $y' = f(x,y)$ or $y' = g(x,y)$. In many cases (as we will see) the numerical method will generate a continuous approximation, $z(x)$ defined on $[a,b]$ satisfying,

$$z' = f(x,z) + \alpha(x) + \delta(x), \quad z(a) = y_0, \tag{1}$$

where $\|\delta(x)\| \leq TOL$ and TOL is a parameter supplied to the method.

Even when $z(x)$ is not directly determined by the method this approach provides an effective way to analyze the associated errors that arise.

Note:

- If $\gamma(x) = \beta(x) + \alpha(x) + \delta(x)$, then we can interpret all three sources of error in terms of perturbations of the desired ODE,

$$\begin{aligned} y' &= g(x,y), \quad y(a) = y_0, \\ z' &= g(x,z) + \gamma(x), \quad z(a) = y_0. \end{aligned}$$

- The analysis of errors is reduced to the question of how small perturbations affect the solution of IVPs. That is, what is the relation between $\|\gamma(x)\|$ and $\|y(x) - z(x)\|$.
- We will assume that $\|\delta(x)\|$ will dominate $\|\beta(x)\|$ and $\|\alpha(x)\|$.

An example of a numerical method which attempts to produce an approximation, $z(x)$, satisfying (1), when applied to $z' = f(x,z) + \alpha(x)$, is the method, DVERK in the Visual Numerics (IMSL) library. This method has the calling sequence,

$$DVERK(N, FCN, X, Y, XEND, TOL, IND, C, NW, W)$$

and determines a partitioning,

$$X = x_0 < x_1 \cdots x_{N_{TOL}} = XEND$$

and corresponding approximations $y_i \approx y(x_i)$ for $i = 0, 1, \cdots N_{TOL}$. This is actually an essential feature of all current adaptive numerical methods for ODEs.

Given this discrete approximation, $(x_i, y_i)_{i=0}^{N_{TOL}}$, this method also generates a continuous interpolant, $\hat{z}(x)$ (a piecewise polynomial or, more precisely a vector of piecewise polynomials), that satisfies:

$$\hat{z}' = f(x, \hat{z}) + \alpha(x) + \hat{\delta}(x),$$

with $\alpha(x)$ as before and,

$$\|\hat{\delta}(x)\| \leq \hat{C}\ TOL,$$

where $\hat{C}$ depends on the problem and the method. Note that provided $TOL >> \mu$, one can 'absorb' $\alpha(x)$ in $\hat{\delta}(x)$. We will see that the local error estimates of a numerical method will be the basis for good approximations to $\hat{\delta}(x)$. That is, bounds on $\|\hat{\delta}(x)\|$ are often computable.

2.1.1 Solution of Perturbed Systems

In this section we develop a mathematical framework for the error analysis of ODE methods using perturbation theory. The approach will allow us to quantify the conditioning of our task and provide us with insight into what we can expect from a numerical method. We will begin with a first order expansion (local linearization) of the error and then provide a more complete rigorous analysis of the error.

1. Let $y(x)$ and $z(x)$ satisfy perturbed systems:

$$\begin{aligned} y' &= f(x,y), \quad y(a) = y_0 & (2)\\ z' &= f(x,z) + \delta(x), \quad z(a) = y_0 & (3) \end{aligned}$$

where $\|\delta(x)\|$ is 'small' for $x \in [a,b]$.
Let $\epsilon(x) = z(x) - y(x)$, then

$$\epsilon'(x) = z'(x) - y'(x) = f(x,z) - f(x,y) + \delta(x),$$

and, using local linearization (ignoring $[\epsilon^2(x)]$ terms) we have,

$$\epsilon' \approx J(x)\epsilon + \delta(x), \quad \epsilon(a) = 0,$$

where

$$J(x) = \frac{\partial f}{\partial y}|_{z(x)}.$$

Note:

- This equation is valid (relevant) only if $\|\epsilon(x)\|$ is small.
- If $v(x)$ satisfies the linear IVP,

$$v' = J(x)v + \delta(x), \quad v(a) = 0,$$

then $\|v(x)\|$ (for any norm) will provide an estimate of the norm of the error, $\|\epsilon(x)\|$. (Sometimes we will estimate or bound $\|\epsilon(x)\|$ directly without estimating $\epsilon(x)$.)

- **An Example**
Consider the IVP,

$$y' = y - 2x/y, \quad y(0) = 1.$$

The known true solution is $y(x) = (1+2x)^{1/2}$.

$$J(x) = \frac{\partial f}{\partial y} = 1 + \frac{2x}{y^2} = 1 + \frac{2x}{1+2x} < 2, \quad \text{for} \ \ x > 0,$$

and therefore the solution of the variational equation (defining $v(x)$) will have a component whose growth is <u>bounded</u> by e^{2x}.

2. An alternative estimate of $\epsilon(x)$ is possible using the approach of defect correction (introduced by Zadainasky, Stetter etc. in the 1970's). It is based on the assumption that the actual error in solving a known 'nearby' problem should be a good estimate of the true error in the original problem. That is, if we know that $y(x)$ and $z(x)$ satisfy (2, 3) (and that $z(x)$ is, computable), then we can apply the same numerical method to the IVP (3) (using the same stepsize sequence) obtaining a sequence $(\hat{z}_i)_{i=0}^N$ as a discrete approximation to $(z(x_i))_{i=0}^N$. We can then use

$$\epsilon(x_i) = z(x_i) - y(x_i) = y_i - y(x_i) \approx \hat{z}_i - z(x_i) = \hat{z}_i - y_i.$$

Note that this approach requires the explicit computation of $\delta(x)$, it assumes that $\delta(x)$ is differentiable and that $\|\delta(x)\|$ dominates $\|\alpha(x)\|$.

2.1.2 Qualitative Behaviour of the Error

To investigate the qualitative behaviour of the error, $\epsilon(x)$, we will first study solutions of linear IVPs of the form,

$$v' = J(x)v + \delta(x), \quad v(a) = 0.$$

1. Consider the special case of $a = 0$ and $\delta(x)$ arbitrary. That is, the linear system:

$$v' = A(x)v + w(x), \quad v(0) = v_0, \tag{4}$$

with $A(x)$ continuous in $[0, x_F]$.

<u>Theorem I:</u> (For example, see Bellman)
There exists a unique solution to (4). To determine the solution, find the matrix solution (Matrizant or Fundamental Matrix) to :

$$\frac{dY}{dx} = A(x)Y(x), \quad Y(0) = I.$$

Exercise: Show that the j^{th} column of $Y(x)$, $y_j(x)$, is the solution of,

$$y_j'(x) = A(x)y_j, \quad y_j(0) = e_j,$$

where e_j is the vector with j^{th} component $= 1$ and all other components $= 0$.

<u>Theorem II:</u>
$Y(x)$ is not singular in $[0, x_F]$. More precisely,

$$det(Y(x)) = e^{(\int_0^x \sum_{i=1}^n a_{ii}(s)\, ds)}.$$

Now consider the homogeneous form (ie., $w(x) = 0$) of (4) with <u>arbitrary</u> initial conditions (specified at $x = a$),

$$v' = A(x)v, \quad v(a) = v_0. \tag{5}$$

We then have,

$$v(x) = Y(x)Y^{-1}(a)v_0.$$

This follows since it satisfies the defining IVP and there is a unique solution (from Theorem I).

To determine the solution of the nonhomogeneous form of (4) with arbitrary initial conditions (specified at $x = a$), let $v(x) = Y(x)u(x)$ for any $u(x)$. Then,

$$\begin{aligned} v' &= Y'u + Yu' \\ &= A(x)Yu + Yu' \\ &= A(x)v + Yu'. \end{aligned}$$

Therefore v is the solution of (4) iff,

$$Yu' = w(x), \text{ and } v(0) = v_0.$$

Note that this first condition implies $u(x) = \int_0^x Y^{-1}(s)w(s)ds + u_0$ while the second condition implies $v(0) = u(0) = u_0 = v_0$.

We then have,

$$v(x) = Y(x)v_0 + \int_0^x Y(x)Y^{-1}(s)w(s)ds.$$

2. For the constant coefficient case ($A(x) \equiv A$, constant) we have,

$$Y' = AY, \ \ Y(0) = I,$$

with the solution,

$$Y(x) = e^{Ax} \equiv \sum_{i=0}^{\infty} \frac{(Ax)^i}{i!}.$$

This equation defines the Matrix exponential. It has the following properties:

(a) It is well defined since

$$\|e^{Ax}\| \leq \sum_{i=0}^{\infty} \frac{\|Ax\|^i}{i!} = e^{\|Ax\|},$$

and this implies uniform convergence in any finite interval.

(b) e^{Ax} satisfies this constant coefficient IVP.

(c) $e^{A(x+s)} = e^{Ax}e^{As}$. That is, $Y(x+s) = Y(x)Y(s)$, $Y^{-1}(x) = Y(-x)$.

(d) e^{Ax} is nonsingular for any A, x.

(e) For the nonhomogeneous constant coefficient IVP:

$$v' = Av + w(x), \quad v(0) = v_0,$$

we have,

$$v(x) = e^{Ax}v_0 + \int_0^x e^{A(x-s)}w(s)ds.$$

(f) Let $A = SDS^{-1}$, D a diagonal matrix,

$$D = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix},$$

then we have,

$$e^{Ax} = Se^{Dx}S^{-1},$$

where

$$e^{Dx} = \begin{bmatrix} e^{\lambda_1 x} & 0 & \cdots & 0 \\ 0 & e^{\lambda_2 x} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & e^{\lambda_n x} \end{bmatrix}.$$

This follows from the definition of the matrix exponential and the fact that $A^k = SD^kS^{-1}$.

(g) With $A = SDS^{-1}$ (as above), the homogeneous IVP, $v' = Av, \quad v(0) = v_0$, has the solution,

$$v(x) = \sum_{j=1}^{n} \alpha_j e^{\lambda_j x} \underline{s}_j,$$

where the $\underline{s}_j$ are the columns of S and satisfy, $A\underline{s}_j = \lambda_j \underline{s}_j$, and $\underline{\alpha} = (\alpha_1, \alpha_2 \cdots \alpha_n)^T$, satisfies, $S\underline{\alpha} = v_0$.

This follows since $e^{Ax} = Se^{Dx}S^{-1}$ implies,

$$v(x) = Y(x)v_0 = Se^{Dx}(S^{-1}v_0) = Se^{Dx}\underline{\alpha}.$$

Note that

$$Se^{Dx} = \begin{bmatrix} \underline{u}_1 & \underline{u}_2 & \cdots & \underline{u}_n \end{bmatrix},$$

where $\underline{u}_j = e^{\lambda_j x}\underline{s}_j$. We then have the desired result:

$$v(x) = \sum_{j=1}^{n} \alpha_j e^{\lambda_j x} \underline{s}_j.$$

(h) When A is not diagonalizable properties analagous to (f) and (g) are more complicated. Consider the 'shift' matrix,

$$U = \begin{bmatrix} 0 & \epsilon_1 & 0 & \cdots & 0 \\ 0 & 0 & \epsilon_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & \epsilon_{n-1} \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix},$$

where $\epsilon_j = 0$ or 1. Exercise: – Show that e^{Ux} is a polynomial in Ux, (what degree?).

Now if $A = [\lambda I + U]$, then

$$e^{Ax} = e^{\lambda I x} e^{Ux} = e^{\lambda x} P(Ux).$$

Therefore for any matrix, $A = TJT^{-1}$ where J is in Jordan Normal Form, we have

$$\begin{aligned} e^{Ax} &= Te^{Jx}T^{-1} \\ &= T(\sum_{j=1}^{m} e^{\lambda_j x} P_j(U_j x))T^{-1}, \end{aligned}$$

where m is the number of distinct eigenvalues of A, U_j is the shift matrix associated with the Jordan block for λ_j and P_j is a polynomial.

(i) $e^{A+B} = e^B e^A$ iff A and B commute ($AB = BA$) .

3. Stability of IVPs:(sensitivity to the initial conditions)
One component of the mathematical conditioning of an IVP is the sensitivity of the solution to perturbations in the initial conditions.

- **Definition**:
The solution, $y(x)$ of the IVP

$$y' = f(x, y), \quad y(a) = y_0,$$

is stable with respect to changes in the initial conditions if, given $\epsilon > 0$, there exists $\delta > 0$ such that if $\|y(a) - \hat{y}(a)\| \leq \delta$ and $y(x)$ and $\hat{y}(x)$ both satisfy $y' = f(x, y)$ then,

$$\|y(x) - \hat{y}(x)\| < \epsilon, \quad \text{for all } \ x > a.$$

This is a strong property since it assumes an infinite interval of interest. If we consider $y(x)$ as a curve in n-space surrounded by 'tubes' of increasing radii; then this states that if $\hat{y}$ ever gets inside a tube it stays inside a slightly larger tube.

- **Definition**:
  The solution, $y(x)$ of the IVP

  $$y' = f(x, y), \quad y(a) = y_0,$$

  is asymptotically stable with respect to changes in the initial conditions if, in addition to being stable, $\|y(x) - \hat{y}(x)\| \to 0$ as $x \to \infty$.
  Now consider the constant coefficient linear IVP,

  $$y' = Ay,$$

  The spectral absicca of A, $\alpha(A)$, is defined as

  $$\alpha(A) = \max_{j=1}^{n} Re(\lambda_j),$$

  (where λ_j are the eigenvalues of A).

- **Theorem**:
  $y(x)$, a solution of this linear problem is stable iff $\alpha(A) \leq 0$ and all the eigenvalues of A with $Re(\lambda_j) = 0$ are simple. (asymptotically stable iff $\alpha(A) < 0$.)
  This follows since,

  $$y(x) = e^{A(x-a)}y_0 = S[\sum_{j=1}^{m} e^{\lambda_j(x-a)} P_j(U_j(x-a))]S^{-1}y_0.$$

  Note that there is no such theorem for the variable coefficient IVP, $y' = A(x)y$. $y(x)$ may be unbounded even if $\alpha(A(x)) < 0$ for all $x > a$. (Sufficient conditions for stability do exist.)

2.1.3 General Error Bounds for Perturbed Systems

In this section we will derive bounds on the growth of solutions of the general linear IVP,

$$v' = A(x)v + w(x),$$

using differential inequalities. In some cases we will be able to compute these bounds without having to compute direct approximations to $v(x)$.

1. **Differential Inequality Theorem** (Coppel) Let $y(x)$ be a solution of the scalar ODE $y' = f(x, y)$ where $f(x, y)$ is continuous. If $u(x)$ is continuous and satisfies $u(a) \leq y(a)$ and

   $$u' \leq f(x, u) \ \ ;\text{on} \ \ [a, b],$$

   then

   $$u(x) \leq y(x) \ \ \text{on} \ \ [a, b].$$

   Similarly, if $s(a) \geq y(a)$ and $s' \geq f(x, s)$, then $s(x) \geq y(x)$ on $[a, b]$. The proofs of these inequalities are given in Coppel, pp. 27-29.

2. **Lemma**: For $u(x)$ differentiable and in $\Re^n$,

$$\frac{d\,\|u(x)\|}{dx} \le \|\frac{du}{dx}\|.$$

Proof:

$$\frac{du}{dx} = \lim_{h\to 0} \frac{u(x+h)-u(x)}{h}.$$

From the triangle inequality for norms we know (for any vectors $w,\ z$),

$$\|w\| \le \|w-z\| + \|z\| \Rightarrow \|w-z\| \ge \|w\| - \|z\|.$$

Therefore we have (with $w = u(x+h)$ and $z = u(x)$),

$$\|\frac{du}{dx}\| = \lim_{h\to 0} \|\frac{u(x+h)-u(x)}{h}\| \ge \lim_{h\to 0}(\frac{\|u(x+h)\| - \|u(x)\|}{h}) = \frac{d\,\|u(x)\|}{dx}.$$

3. **Classical Approach** for bounding solutions to,

$$v' = A(x)v + w(x), \ \text{ on } \ [a,b].$$

$$\frac{d\|v\|}{dx} \le \|\frac{dv}{dx}\| \le \|A\|\ \|v\| \ + \ \|w(x)\| \le L\|v\| + W.$$

(a scalar equation) where $L = \sup_{x\in[a,b]} \|A(x)\|$ and $W = \sup_{x\in[a,b]} \|w(x)\|$ This implies (from the above theorem with $p' = Lp + W,\ \ p(a) = \|v(a)\|$),

$$\|v(x)\| \le p(x) \equiv e^{L(x-a)}\|v(a)\| \ + \ (\frac{e^{L(x-a)}-1}{L})W.$$

4. **Logarithmic Norms** can be used to obtain sharper bounds.

   Definition: The <u>logarithmic norm</u> of a matrix, A, is defined, for any norm subordinate to a vector norm, to be,

$$\mu_{\|\ \|}(A) \equiv \lim_{h\to 0^+} \frac{\|I+hA\|-1}{h}.$$

- For example if $A = \lambda I,\ \lambda \in \Re$,

$$\|I+hA\| = \|(1+h\lambda)I\| = |(1+h\lambda)|\|I\| = |(1+h\lambda)|.$$

  This follows since $\|I\| = 1$ for norms that are subordinate to a vector norm. We then have

$$\mu_{\|\ \|}(A) = \lim_{h\to 0} \frac{|(1+h\lambda)|-1}{h} = \lim_{h\to 0} \frac{(1+h\lambda-1)}{h} = \lambda.$$

  Note that the lognorm of a matrix can be negative.

Now consider $v'(x) = A(x)v + w(x)$,

$$\begin{aligned} v(x+h) &= v(x) + hv'(x) + O(h^2), \\ &= [I + hA(x)]v(x) + hw(x) + O(h^2). \end{aligned}$$

This implies,

$$\|v(x+h)\| \leq \|I + hA(x)\| \; \|v(x)\| + h\|w(x)\| + O(h^2),$$

or (subtracting $\|v(x)\|$ from both sides and dividing by h),

$$\frac{\|v(x+h)\| - \|v(x)\|}{h} \leq \frac{(\|I + hA(x)\| - 1)}{h}\|v(x)\| + \|w(x)\| + O(h).$$

Taking limits we obtain,

$$\begin{aligned} \frac{d\,\|v(x)\|}{dx} &\leq \mu_{\|\,\|}(A(x))\|v(x)\| + \|w(x)\| \\ &\leq M\|v(x)\| + W, \end{aligned}$$

where $M = \sup_{x\in[a,b]} \mu_{\|\,\|}(A(x))$. Therefore, as above (using differential inequalities), and $p' = Mp + W, \;\; p(a) = \|v(a)\|)$, we obtain the <u>bound</u>,

$$\|v(x)\| \leq e^{M(x-a)}\|v(a)\| + (\frac{e^{M(x-a)} - 1}{M})W.$$

Exercise: what about the case $M \leq 0$.

5. **Applications of this Result (Examples):**

$$y' = Ay, \;\; y(0) = y_0.$$

If we assume we have a numerical solution, $z(x)$ such that,

$$\|z' - Az\| \leq TOl, \;\; z(0) = y_0,$$

we can conclude from the above theorem that,

$$\|z(x) - y(x)\| \leq \rho(x) \;\text{ for }\; x \in [0, T].$$

where

$$\rho' = \mu(A)\rho + TOL, \;\; \rho(0) = 0.$$

Therefore

$$\|z(x) - y(x)\| \leq (\frac{e^{\mu(A)x} - 1}{\mu(A)})TOL$$

is a <u>guaranteed bound</u>.

Two examples:

(a)

$$A = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 \end{bmatrix}.$$

In this case $\mu_\infty(A) = 0$ and we can conclude that

$$\|z(x) - y(x)\| \leq xTOL.$$

This follows since $\lim_{s\to 0} \frac{e^{sx}-1}{s} = x$.

(b)

$$A = \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}, \mu_\infty(A) = -1,$$

and

$$\|z(x) - y(x)\| \leq (1 - e^{-x})TOL.$$

6. **Properties of the Lognorm:** ($\mu(A)$ depends on the $\| \|$.)

(a) $\mu(\lambda I) = \lambda$ for $\lambda \in \Re$.

(b) $\mu(zI) = Re(z)$ for complex z.

(c) $\mu(\lambda A) = \lambda\mu(A)$ if $\lambda \in \Re, \quad \lambda > 0$.

(d) $|\mu(A)| \leq \|A\|$.

(e) $\mu(A + B) \leq \mu(A) + \mu(B) \leq \mu(A) + \|B\|$.

(f) $\mu(A) \geq \alpha(A)$.
Proof of this property:
let y be a unit eigenvector of A, such that $Ay = \lambda y$. Then we have,

$$\|(I + hA)y\| - \|y\| = |(1 + h\lambda)| - 1 \to hRe(\lambda)$$

and

$$\|(I + hA)y\| - \|y\| \leq \|I + hA\| - 1 \to h\mu(A).$$

Therefore taking limits $h \to 0$, we obtain for each eigenvalue of A,

$$Re(\lambda) \leq \mu(A).$$

(g) Computing $\mu(A)$ for different norms:

	$\|y\|$	$\|A\|$	$\mu(A)$
l_∞	$\max_{i=1}^n \|y_i\|$	$\max_{i=1}^n (\sum_{j=1}^n \|a_{ij}\|)$	$\max_{i=1}^n (Re(a_{ii}) + \sum_{j=1, j\neq i}^n \|a_{ij}\|)$
l_1	$\sum_{i=1}^n \|y_i\|$	$\max_{j=1}^n (\sum_{i=1}^n \|a_{ij}\|)$	$\max_{j=1}^n (Re(a_{ii}) + \sum_{i=1, j\neq i}^n \|a_{ij}\|)$
l_2	$(\sum_{i=1}^n \|y_i\|^2)^{(1/2)}$	[largest ev of $A^T A]^{(1/2)}$	largest ev of $1/2(A + A^T)$

7. **Theorem**:
   Let $A(x)$ be continuous and $y(x)$ be a solution of $y' = A(x)y$, then

$$-\mu(-A(x))\|y\| \leq \frac{d\|y\|}{dx} \leq \mu(A(x))\|y\|.$$

   Corollaries: (application of differential inequalities)

   - $y(x)$ is stable if,
   $$\lim_{x\to\infty} \int_a^x \mu(A(s))ds < \infty.$$
   - $y(x)$ is asymptotically stable if,
   $$\lim_{x\to\infty} \int_a^x \mu(A(s))ds = -\infty.$$

8. **Lemma**: (Dahlquist)
   If the ODE $u' = J(x,u)u + w(x)$ is valid in an open neighborhood of the point (a, u_0), then
   $$\frac{d\|u\|}{dx} \leq \mu(J(x,u))\|u(x)\| + \|w(x)\|,$$
   is valid in that neighborhood. (Proof is as for the linear case.)

   Recall our Key Interest:

$$z' = f(x,z) + \delta(x), \text{ with } \|\delta(x)\| \leq \rho(x).$$

   for some known function, $\rho(x)$ (eg, $\rho(x) = TOL$), and

$$y' = f(x,y).$$

   We want to derive a bound on $\|z(x) - y(x)\|$ which is not too pessimistic, and which is potentially computable.

9. **Main Theorem**: (Dahlquist)
Let $z(x)$ be a known function which satisfies the differential inequality:

$$\|\frac{dz}{dx} - f(x,z)\| \leq \rho(x), \quad \text{for} \;\; a \leq x \leq b,$$

(note that $\rho(x)$ is a bound on $\|\delta(x)\|$), and let $y(x)$ be a solution to

$$y' = f(x,y), \quad \text{for} \;\; a \leq x \leq b.$$

Assume that for each $x \in [a,b]$, there exists a region $D_x \in \Re^n$ and a function $l(x)$, such that for all $q \in D_x$,

$$\mu[\frac{\partial f}{\partial y}(x,q)] \leq l(x).$$

Then

$$\|z(x) - y(x)\| \leq p(x),$$

where $p(x)$ is the solution of the scalar IVP,

$$p' = l(x)p + \rho(x), \;\; p(a) = p_0 \geq \|z(a) - y(a)\|,$$

provided

$$\{\eta : \|\eta(x) - z(x)\| \leq p(x)\} \in D_x \;\; \text{for} \;\; x \in [a,b].$$

Proof:
Let $u(x) = z(x) - y(x)$. We then have,

$$\frac{du}{dx} = f(x,z) - f(x,y) + \delta(x), \;\; \text{with} \;\; \|\delta(x)\| \leq \rho(x).$$

Let $q(\theta,x) = z(x) - (1-\theta)u(x)$ for $a \leq x \leq b$. We then have $\frac{dq}{d\theta} = u(x)$ and,

$$f(x,z) - f(x,y) = \int_y^z \left[\frac{\partial f(x,q)}{\partial q}\right]_{q=q(\theta,x)} dq.$$

(note that $q = y \Rightarrow \theta = 0$ and $q = z \Rightarrow \theta = 1$). Therefore,

$$f(x,z) - f(x,y) = \int_0^1 \left[\frac{\partial f(x,q)}{\partial q}\right]_{q=q(\theta,x)} u d\theta, = A(x,u)u,$$

where

$$A(x,u) \equiv \int_0^1 \left[\frac{\partial f}{\partial q}\right]_{q=z(x)-(1-\theta)u(x)} d\theta.$$

Therefore we have,

$$u' = A(x,u)u + \delta(x),$$

so we can apply the above lemma with

$$\mu(A(x,u)) \leq \max_{0\leq\theta\leq 1} \mu(\frac{\partial f}{\partial q}(x, q(x,\theta)) \leq l(x)$$

(using $\mu[A(x,u)] \leq \int_0^1 \mu(\frac{\partial f}{\partial q}) d\theta$.)
Note this is only valid if $q(\theta, x) \in D_x$ for $\theta \in [0,1]$. We then have, from the above Lemma,

$$\frac{d\|u\|}{dx} \leq l(x)\|u\| + \|\delta(x)\| \leq l(x)\|u\| + \rho(x),$$

and we can conclude from the theorem on differential inequalities,

$$\|u(x)\| = \|z(x) - y(x)\| \leq p(x) \text{ if } \{\eta : \|\eta(x) - z(x)\| \leq p(x)\} \in D_x.$$

2.2 General Properties of Numerical Methods:

Most numerical methods for IVPs determine a discrete approximation, $(x_i, y_i)_{i=0}^{N_{TOL}}$, to the true solution, $y(x)$, evaluated on a mesh, $a = x_0 < x_1 \cdots < x_{N_{TOL}} = b$. In general the mesh and N_{TOL} will depend on $f(x,y)$ and the specified accuracy tolerance, TOL (adaptive).

2.2.1 Classical Analysis of Numerical Methods

It is not at all clear how to quantify the effectiveness of a numerical method for IVPs. In this section we will review the classical results for methods that apply a formula with a fixed stepsize and then extend these results to allow us to analyze variable stepsize (adaptive) methods. We will justify why the overall quality of a solution can be determined by analyzing the error introduced on a single step of the integration. (That is, how is, the local behaviour of the numerical solution relates to the global behaviour.)

From mathematics we know that our problem,

$$y' = f(x,y), \quad y(a) = y_0,$$

has a unique solution on $[a,b]$, if $f(x,y)$ satisfies certain smoothness conditions (such as a Lipschitz condition).

Definition: $f(x,y)$is locally Lipschitz for our problem iff there exists $L > 0$, such that for all $u, v \in$ a neighborhood of $y(x)$, we have,

$$\|f(x,u) - f(x,v)\| < L\|u - v\|.$$

1. **What is an Acceptable Numerical Solution?**

   - Numerical methods generate a discrete approximate solution, $(x_i, y_i)_{i=0}^{N}$ with $a = x_0 < x_1 \cdots < x_N = b$ where y_i is an approximation to $y(x_i)$.

- Numerical methods will generally determine y_{i+1} only after $(x_j, y_j)_{j=0}^{i}$ have been determined and a suitable stepsize, $h_{i+1} = (x_{i+1} - x_i)$, has been chosen. An underlying k-step formula is then applied,

$$y_{i+1} = y_i + h_{i+1}\Phi(x_i, y_{i+1}, y_i \cdots y_{i-k+1}).$$

  Note that:

  (a) The increment function, Φ, must depend on $f(\ )$.

  (b) Numerical methods are classified according to the value of k:
  $k = 1$ –One-step Methods (Runge-Kutta).
  $k > 1$ –Multistep Methods (Adams, BDF).

  (c) Numerical Methods can also be classified based on whether or not Φ depends on y_{i+1}. If there is a dependence the method is implicit, otherwise it is explicit.

- The error at x_{i+1} will then depend on $y(x_i) - y_i$ as well as on h_{i+1} and Φ. In stepping to x_{i+1} the method cannot control $y(x_i) - y_i$, but h_{i+1} and possibly Φ can be adjusted to improve the local behaviour of the numerical solution.

- Consider the 'Local' IVP associated with step i:

$$y' = f(x, y), \ \ y(x_i) = y_i, \ \ x \in [x_i, b].$$

  Denote the true solution to this local problem, $z_i(x)$.

  (a) If y_i were exact (ie. $y_i = y(x_i)$) then $y(x) \equiv z_i(x)$.

  (b) When we apply a k-step formula, the best we can expect is that y_{i+1} will be a good approximation to $z_i(x_{i+1})$.

- **Definition:** The Global Error at x_{i+1}, GE_{i+1}, is $y(x_{i+1}) - y_{i+1}$.

- **Definition:** The Local Error associated with step i, LE_i, is $z_i(x_{i+1}) - y_{i+1}$.

  The Global Error at x_{i+1} satisfies:

$$y(x_{i+1}) - y_{i+1} = \underbrace{(y(x_{i+1}) - z_i(x_{i+1}))}_{A} + \underbrace{(z_i(x_{i+1}) - y_{i+1})}_{B},$$

where A depends on the global error at x_i and the 'mathematical stability' of the ODE (and is independent of the method) while B is the local error and depends only on h_{i+1} and Φ.

- **Definition:** The Local Truncation Error associated with step i , LTE_i, is

$$z_i(x_{i+1}) - z_i(x_i) - h_{i+1}\Phi(x_i, z_i(x_{i+1}) \cdots z_i(x_{i-k+1})).$$

  For 1-step formulas we have that $LTE = O(h^{p+1})$ iff $LE = O(h^{p+1})$ but this is not the case for multistep formulas. One can interpret the LTE as a direct measure of how well the exact solution of the IVP satisfies the underlying discrete approximation formula.

- One can characterize a numerical IVP (as opposed to the mathematical IVP that is uniquely determined by our problem specification and some smoothness assumptions) by,

$$< f, a, y_0, b, TOL >,$$

  where

  (a) The first four components characterize the "mathematical problem".

  (b) The last component, TOL, is the accuracy component and can be interpreted in several ways. We will consider two possible interpretations:

  Defect Control: There exists $Z(x) \in C[a, b]$ such that $Z(a) = y_0,\ Z(b) = y_N,\ Z'(x)$ exists almost everywhere and

$$Z' = f(x, Z) + \delta(x),$$

  where $\|\delta(X)\| \leq TOL$, for $x \in [a, b]$. Note that $Z(x)$ need not be computable.

  Error per unit Step – EPUS The local error per unit step (LEPUS) is bounded by TOL. That is,

$$\|z_i(x_{i+1}) - y_{i+1}\| \leq TOL(x_{i+1} - x_i), \text{ for } i = 0, 1 \cdots (N-1).$$

  These two interpretations of accuracy are closely related as the following result shows.

  Lemma: If the problem is locally Lipschitz with the associated Lipschitz constant L, the method keeps $|hL| < c$, and $LEPUS \leq TOL$ then there exists $Z(x) \in C[a, b]$ such that $Z'(x)$ exists at all but a finite number of points in $[a, b]$ and $Z'(x) = f(x, Z) + \delta(x)$, with $\|\delta(x)\| \leq (1 + c)TOL$. (That is, if the second criteria is, satisfied then the first is satisfied for a slightly larger TOL.)

  Proof:
  Define $\tilde{Z}(x)$ for $x \in [a, b]$ in a piecewise fashion as:

$$\tilde{Z}(x) \equiv \tilde{z}_i(x) = z_i(x) - \left(\frac{x - x_i}{x_{i+1} - x_i}\right)\epsilon_i, \text{ for } x \in [x_i, x_{i+1}].$$

where ϵ_i is the discrete local error introduced on step i,

$$\epsilon_i = z_i(x_{i+1}) - y_{i+1}.$$

From our assumptions, $\|\epsilon_i\| \leq TOL(x_{i+1} - x_i)$ or $\frac{\|\epsilon_i\|}{(x_{i+1}-x_i)} \leq TOL$. Now $\tilde{Z}(x_i) = z_i(x_i) = y_i$ and

$$\lim_{x \to x_i^-} \tilde{Z}(x) = z_{i-1}(x_i) - \epsilon_{i-1} = y_i.$$

Therefore $\tilde{Z}(x)$ is continuous on $[a, b]$ and $\tilde{Z}'(x)$ exists everywhere except perhaps at the x_i.

Now for $x \in (x_i, x_{i+1})$ we have (from the definition of $\tilde{Z}(x)$),

$$\begin{aligned}
\tilde{Z}'(x) &= z_i'(x) - \frac{1}{x_{i+1} - x_i}\epsilon_i \\
&= f(x, z_i(x)) - \frac{1}{x_{i+1} - x_i}\epsilon_i \\
&= f(x, \tilde{Z}(x)) \underbrace{- \frac{1}{x_{i+1} - x_i}\epsilon_i + [f(x, z_i(x)) - f(x, \tilde{Z}(x))]}_{\delta(x)} \\
&= f(x, \tilde{Z}(x)) + \delta(x),
\end{aligned}$$

where,

$$\begin{aligned}
\|\delta(x)\| &\leq \frac{1}{x_{i+1} - x_i}\|\epsilon_i\| + \|f(x, z_i(x)) - f(x, \tilde{Z}(x))\| \\
&\leq \frac{1}{x_{i+1} - x_i}\|\epsilon_i\| + L\|z_i(x) - \tilde{Z}(x))\| \\
&= \frac{1}{x_{i+1} - x_i}\|\epsilon_i\| + L\frac{x - x_i}{x_{i+1} - x_i}\|\epsilon_i\| \\
&= (1 + L(x - x_i))\,\frac{\|\epsilon_i\|}{(x_{i+1} - x_i)} \\
&\leq (1 + c)TOL \;\;=\;\; \overline{TOL}.
\end{aligned}$$

Note that the converse of this lemma is also true (an easy exercise).

2. **Fixed Stepsize Classical Analysis of the underlying Formula:**

   Note that this type of analysis is still relevant since it identifies a necessary (although not sufficient) criteria for determining suitability of a formula to be used as the basis for a numerical method for solving IVPs.

   - An ODE is <u>well conditioned</u> (or well posed) with respect to initial conditions, y_0 on the interval $[a, b]$ if there exists $K > 0$ and $\bar{\epsilon} > 0$ such that for all $\epsilon < \bar{\epsilon}$ any solution of a perturbed IVP:

     $$z' = f(x, z) + \delta(x), \quad z(a) = y_0 + \epsilon_0,$$

with $\|\delta(x)\| < \epsilon$ for $x \in [a,b]$ and $\|\epsilon_0\| < \epsilon$, satisfies:

$$\|z(x) - y(x)\| < K\epsilon, \text{ for all } x \in [a,b].$$

Theorem (A corollary of earlier results)
If $f(x,y)$ satisfies a local Lipschitz condition then the ODE is well posed with respect to any initial conditions (with the corresponding $K \approx 2e^{L(b-a)}$).

- A formula is stable if for each well conditioned ODE, there exists a constant $h_0 > 0$ such that a change in the starting value(s) by a fixed amount produces a bounded change in the discrete numerical solution that results from from applying the formula with a constant stepsize h, for any $h < h_0$, on the finite interval $[a,b]$.
- A formula is said to converge if any desired degree of accuracy can be achieved for any problem satisfying a Lipschitz condition by choosing a small enough constant stepsize h. (That is, the maximum global error $\to 0$ as $h \to 0$.)

  Note that convergence $\Rightarrow$ stability but the converse is not true (eg. the formula $y_{i+1} = y_i$ is stable but not convergent).
- A formula is of order p iff p is the largest value for which there exists $\bar{h} > 0$, $K > 0$, such that for any $x \in [a,b], i > (x-a)/\bar{h}$, when the formula is applied to an IVP with fixed stepsize, $h = \frac{(x-a)}{i}$ we have (after i steps of size h), $\|y(x_i) - y_i\| < Kh^p$. (Note that after i steps of size h, y_i approximates $y(a+ih)$, where $a + ih = a + i(x-a)/i = x$.)

  Classical Theorem: (see Henrici for proof)
  A discrete numerical formula with local error $O(h^{p+1})$ is of order p. That is, $\|y_{i+1} - z_i(x_i + h)\| = O(h^{p+1}) \Rightarrow$ The formula is of order p.
- A formula is consistent iff order is at least one. The Main Classical Result is that a formula is convergent iff it is stable and consistent. (This theorem applies only to fixed stepsize.) In particular we have

  One-Step Methods Order p $\Leftrightarrow$ local error $= O(h^{p+1}) \Rightarrow$ global error $= O(h^p)$.

  Multistep Methods Order p and numerical stability $\Leftrightarrow$ local error $= O(h^{p+1}) \Rightarrow$ global error $= O(h^p)$.
- The absolute stability of a formula is defined in terms of fixed h, but finite (and not $h \to 0$).
  - Consider the model scalar IVP:

    $$y' = \lambda y, \quad y(x_0) = y_0, \quad \lambda \in \Re, \quad \lambda << 0.$$

    The solution is,

    $$y(x) = e^{\lambda(x-x_0)} y_0, \text{ on any interval } [x_0, x_F],$$

    and therefore $|y(x)| = e^{\lambda(x-x_0)}|y_0|$.

- Consider applying Euler's formula with constant stepsize to this problem (simplest one-step formula),

$$y_{i+1} = y_i + hy'_i = (1 + h\lambda)y_i,$$

When $|\lambda|$ is large, $h\lambda << -2,\ |1 + h\lambda| >> 1$ and,

$$|y_{i+1}| = |1 + h\lambda|^{i+1}|y_0|.$$

This leads to an unstable computation.

- Consider applying the Implicit Euler (Backward Euler) formula with constant stepsize to this problem,

$$y_{i+1} = y_i + hy'_{i+1} = \frac{1}{(1 - h\lambda)}y_i,$$

When $|\lambda|$ is large, $h\lambda << -2,\ |\frac{1}{(1-h\lambda)}| << 1$ and,

$$|y_{i+1}| = |\frac{1}{(1 - h\lambda)}|^{i+1}|y_0|.$$

This leads to a very stable computation as the $|y_i| \to 0$ rapidly.

Now consider the slightly more general model problem,

$$y' = \lambda y, \;\; y(x_0) = y_0, \;\; \text{where } \lambda \in C.$$

A formula, when applied to this problem with constant stepsize h, is considered stable (for the particular values of h and λ) if $|y_i|$ remains bounded as $i \to \infty$ for any initial condition vector, y_0.

Definition
The Region of absolute stability associated with a formula is

$$\mathcal{R} \equiv \{h\lambda : \text{when applied to the model problem with constant } h, |y_i| \to 0\}.$$

- Limitations of constant stepsize and $h \to 0$ analysis:
  - (a) Error bounds can be pessimistic (not necessarily true of error estimates).
  - (b) Problem characteristics often change over $[a, b]$ and one fixed h is not suitable (adaptive choice of stepsize is warranted).
  - (c) ($h \to 0$) analysis ignores round-off error. (Precision must increase as $h \to 0$ if analysis is to be meaningful and convergence observed.)
- Practical convergence as $TOL \to 0$ for EPUS.
  - (a) The global error is difficult to estimate/bound but local error can be estimated and controlled efficiently and reliably. To do this a method must have an associated formula to determine the discrete solution as well as another formula for estimating the local error (or the defect of an associated interpolant, $S(x)$).
  - (b) The stepsize should vary so that $LEPUS \approx TOL$ on each step. We then have from the 'equivalence of the error criteria, that there exists a $Z(x) \in C[a, b]$ satisfying the IVP,

    $$z' = f(x, z) + \delta(x), \;\; z(a) = y_0,$$

    with

    $$\|\delta(x)\| \leq K\, TOL \Rightarrow \|GE\| = O(TOL).$$

2.2.2 Extension of Classical Analysis

Variable stepsize (adaptive) methods are now widely available and are the basis for most reliable and effective general purpose software for IVPs.

1. Often one can associate, with the underlying discrete numerical solution $(x_i, y_i)_{i=0}^{N_{TOL}}$, a 'natural' computable piecewise polynomial, $S(x)$, which interpolates the discrete solution and provides a continuous (possibly $C^1[a,b]$) approximation to $y(x)$. That is, for $x \in [x_i, x_{i+1}]$, $S(x)$ is, a polynomial of degree $\leq k$, (which we will denote by $\hat{z}_i(x)$) interpolating the 'local data' y_i, y_{i+1}.

2. When such an interpolant, $S(x)$, exists then $\hat{z}_i(x) \approx z_i(x)$ over $[x_i, x_{i+1}]$. One can then ask how well $\hat{z}_i(x)$ satisfies the local IVP that defines $z_i(x)$. That is, one can define the approximate (or numerical solution) to be $S(x)$ and ask how well it satisfies the original IVP over [a, b].

3. The <u>defect</u>, $\hat{\delta}(x)$, associated with this particular continuous numerical solution, $S(x)$, is defined for $x \in [a,b]$, to be

$$\hat{\delta}(x) \equiv S'(x) - f(x, S(x)).$$

   Note:

   - $S(x)$ is computable and is a vector of piecewise polynomials (one for each component of $y(x)$).
   - $\hat{\delta}(x)$ will be computable and continuous everywhere except possibly at the meshpoints, $x_1, x_2, \cdots x_{(N_{TOL}-1)}$.

4. The notion of GE and LE extend from the discrete to continuous case in a straightforward way,

$$CGE \equiv y(x) - S(x), \quad \text{for } \ x \in [a,b].$$

$$CLE \equiv z_i(x) - S(x) = z_i(x) - \hat{z}_i(x), \quad \text{for } \ x \in [x_i, x_{i+1}].$$

   The following theoretical results can be shown,

   - $\hat{\delta}(x) = O(h^p) \Rightarrow CLE = O(h^{p+1}) \Rightarrow CGE = O(h^p)$, where h is the maximum stepsize.
   - If $\hat{z}_i(x)$ is computed on each step (and hence $S(x)$ determined globally), then the associated $\hat{\delta}(x)$ can be sampled (at some extra cost) and a bound on $\|\hat{\delta}(x)\|$ over (x_i, x_{i+1}) estimated. An error control strategy can 'accept' (x_{i+1}, y_{i+1}) only if this estimated bound is less than TOL. That is, an attempt is made to ensure $\|\hat{\delta}(x)\| \leq TOL$ for all $x \in [a,b]$ and therefore $S(x)$ will satisfy exactly a slightly pertubed IVP, where the perturbation is bounded by TOL in norm.

2.3 Runge-Kutta methods:

Runge-Kutta methods are one-step methods where the ODE is sampled at s points in the interval $[x_i, x_{i+1}]$ and the associated $\Phi(x_i, y_i)$ is defined as a weighted average of these sampled values.

1. More specifically an s-stage Runge-Kutta formula uses s derivative evaluations and has the form:
$$y_{i+1} = y_i + h_{i+1}(\omega_1 k_1 + \omega_2 k_2 \cdots + \omega_s k_s),$$
where $h_{i+1} = x_{i+1} - x_i$ and, for $j = 1, 2 \cdots s$,
$$k_j = f(x_i + h_{i+1}\alpha_j, y_i + h_{i+1}\sum_{r=1}^{s}\beta_{jr}k_r).$$
This formula is characterized by the tableau,
$$\begin{array}{c|cccc} \alpha_1 & \beta_{11} & \beta_{12} & \dots & \beta_{1s} \\ \alpha_2 & \beta_{21} & \beta_{22} & \dots & \beta_{2s} \\ \vdots & \vdots & \vdots & & \vdots \\ \alpha_s & \beta_{s1} & \beta_{s2} & \dots & \beta_{ss} \\ \hline & \omega_1 & \omega_2 & \dots & \omega_s \end{array}$$
These $s^2 + 2s$ parameters are usually chosen to make implementation easier and/or to maximize the order. The maximum attainable order for an s-stage Runge-Kutta formula is $2s$.

2. Runge-Kutta formulas can be classified into three categories:

   Explicit If $\beta_{jr} = 0$ for $r \geq j$ the formula is <u>explicit</u> and the k_j can be computed sequentially as,
$$k_j = f(x_i + h_{i+1}\alpha_j, y_i + h_{i+1}\sum_{r=1}^{j-1}\beta_{jr}k_r), \text{ for } j = 1, 2 \cdots s.$$

   Semi-implicit If $\beta_{jr} = 0$ for $r > j$ and $\beta_{j,j} \neq 0$ for some j, the formula is called <u>semi-implicit</u> and the k_j can be determined sequentially by solving an order n nonlinear system (at most s such systems per step).

   Implicit Otherwise the formula is <u>implicit</u> and the equations defining the k_j are a fully coupled nonlinear system of order sn.

3. The derivation of maximum order (for a given s) Runge-Kutta formula has received considerable attention in recent years. Recall that a formula will be order p if for all sufficiently differentiable functions $y(x)$ we have,
$$y(x_{i+1}) - y(x_i) - h\Phi(x_i, y(x_i)) = O(h^{p+1}). \tag{6}$$

Consider a formula Φ corresponding to a $\underline{2}$-stage explicit Runge-Kutta formula,

$$\Phi(x_i, y_i) = \omega_1 k_1 + \omega_2 k_2,$$

where,

$$\begin{aligned} k_1 &= f(x_i, y_i), \\ k_2 &= f(x_i + \alpha h, y_i + h\beta k_1). \end{aligned}$$

We determine the parameters $\omega_1, \omega_2, \alpha, \beta$ to obtain as high an order formula as possible. From the definition of order we have order p if

$$y(x_{i+1}) = y(x_i) + h(\omega_1 k_1 + \omega_2 k_2) + O(h^{p+1}) \tag{7}$$

for all sufficiently differentiable functions $y(x)$. To derive such a formula we expand each of $y(x_{i+1}), k_1, k_2$ in a Taylor Series about the point (x_i, y_i), equate like powers of h on both sides of (7), and set $\alpha, \beta, \omega_1, \omega_2$ accordingly.

In what follows we omit arguments when they are evaluated at the point (x_i, y_i). The expansion of the LHS of (7) is:

$$\begin{aligned} LHS &= y(x_{i+1}), \\ &= y(x_i) + hy^{'}(x_i) + \frac{h^2}{2} y^{''}(x_i) + \frac{h^3}{6} y^{'''}(x_i) + O(h^4), \\ &= y(x_i) + hf + \frac{h^2}{2}(f_x + f_y f) \\ & \quad + \frac{h^3}{6}(f_{xx} + 2f_{xy} f + f_{yy} f^2 + f_y f_x + f_y^2 f) + O(h^4). \end{aligned}$$

The expansion of the RHS of (7) is more complicated and first requires the expansions of k_1 and k_2,

$$\begin{aligned} k_1 &= f, \\ k_2 &= f(x_i + \alpha h, y(x_i) + \beta h k_1), \\ &= f(x_i, y(x_i) + \beta h f) + (\alpha h) f_x(x_i, y(x_i) + \beta h f) \\ & \quad + \frac{\alpha^2 h^2}{2} f_{xx}(x_i, y(x_i) + \beta h f) + O(h^3), \\ &= \left[f + \beta h f f_y + \frac{(\beta h f)^2}{2} f_{yy} + O(h^3) \right] \\ & \quad + \left[\alpha h f_x + \alpha \beta h^2 f f_{xy} + O(h^3) \right] + \left[\frac{\alpha^2 h^2}{2} f_{xx} + O(h^3) \right], \\ &= f + (\beta f f_y + \alpha f_x)\ h + (\frac{\beta^2}{2} f^2 f_{yy} + \alpha \beta f f_{xy} + \frac{\alpha^2}{2} f_{xx})\ h^2 + O(h^3). \end{aligned}$$

The expansion of the RHS of (7) then is (with these substitutions for k_1 and k_2)

$$RHS = y(x_i) + h(\omega_1 k_1 + \omega_2 k_2),$$

$$
\begin{aligned}
&= y(x_i) + h\omega_1 f + h\omega_2 [\cdots] + O(h^4), \\
&= y(x_i) + [(\omega_1+\omega_2)f]\ h + [\omega_2(\beta f f_y + \alpha f_x)]\ h^2 \\
&\quad + \left[\omega_2(\frac{\beta^2}{2} f^2 f_{yy} + \alpha\beta f f_{xy} + \frac{\alpha^2}{2} f_{xx})\right]\ h^3 + O(h^4).
\end{aligned}
$$

Finally, these expansions are valid for small h and all sufficiently differentiable $f(x,y)$, so equating like powers of h, in the LHS and RHS expansions, we observe the following:

For order 0 : The coefficients of h^0 always agree and we have order at least zero for any choice of the parameters.

For order 1: If $\omega_1+\omega_2 = 1$ the coefficients of h^1 agree and we have at least order 1.

For order 2: In addition to satisfying the order 1 constraints we must have the coefficient of h^2 the same. That is $\alpha\omega_2 = 1/2$ and $\beta\omega_2 = 1/2$.

For order 3: In addition to satisfying the order 2 constraints we must have the coefficients of h^3 the same. That is we must satisfy the equations,

$$
\begin{aligned}
\omega_2\alpha^2 &= \frac{1}{3}, \\
\omega_2\alpha\beta &= \frac{1}{3}, \\
\omega_2\beta^2 &= \frac{1}{3}, \\
\frac{1}{6} f_x f_y &= ?, \\
\frac{1}{6} f_y^2 f &= ?.
\end{aligned}
$$

Note that there are not enough terms in the coefficient of h^3 in the expansion of the RHS to match the expansion of the LHS. We cannot therefore equate the coefficients of h^3 and the maximum order we can obtain is order 2. Our formula will be order 2 for any choice of $\omega_2 \neq 0$, with $\omega_1 = 1-\omega_2$ and $\alpha = \beta = \frac{1}{2\omega_2}$. This is a one-parameter family of 2^{nd}-order Runge-Kutta formulas.

Three popular choices from this family are:

Modified Euler: $\omega_2 = 1/2$

$$
\begin{aligned}
k_1 &= f(x_i, y_i), \\
k_2 &= f(x_i + h, y_i + hk_1), \\
y_{i+1} &= y_i + \frac{h}{2}(k_1 + k_2).
\end{aligned}
$$

Midpoint: $\omega_2 = 1$

$$\begin{aligned} k_1 &= f(x_i, y_i), \\ k_2 &= f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1), \\ y_{i+1} &= y_i + hk_2. \end{aligned}$$

Heun's Formula: $\omega_2 = 3/4$

$$\begin{aligned} k_1 &= f(x_i, y_i), \\ k_2 &= f(x_i + \frac{2}{3}h, y_i + \frac{2}{3}hk_1), \\ y_{i+1} &= y_i + \frac{h}{4}(k_1 + 3k_2). \end{aligned}$$

Note that the derivations of maximal order formulas can be very messy and tedious, but essentially they follow (as outlined above for the case $s = 2$) by expanding each of the k_j in a bi-variate Taylor series. The maximum attainable order for an s-stage, explicit Runge-Kutta formula is given by the following table:

s	1	2	3	4	5	6	7	8	9	10	11
max order	1	2	3	4	4	5	6	6	7	7	8

An Example – The Classical Fourth Order Runge Formula (1895)

$$\begin{array}{c|cccc} - & - & & & \\ 1/2 & 1/2 & - & & \\ 1/2 & 0 & 1/2 & - & \\ 1 & 0 & 0 & 1 & - \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

4. **Error Estimates:**

   (a) Ideally a method would estimate a bound on the global error and adjust the stepsize, h, to keep the magnitude of the global error less than a tolerance. Such computable bounds are possible but are usually pessimistic and inefficient to implement.

   (b) On the other hand, local errors can be reliably and efficiently estimated and controlled. Consider a method which keeps the magnitude of the local error less than $h\,TOL$ on each step.

   We will consider the derivation of computable local error estimates for such methods.

   As an example, consider the Modified Euler Formula:

$$\begin{array}{c|cc} - & - & \\ 1 & 1 & - \\ \hline & 1/2 & 1/2 \end{array}$$

We have shown

$$\begin{aligned} z_i(x_{i+1}) &= y_i + \frac{h}{2}(k_1 + k_2) \\ &+ \left[\frac{1}{4}f^2 f_{yy} + \frac{1}{2}f f_{xy} + \frac{1}{4}f_{xx} - y'''(x_i)\right] h^3 + O(h^4), \\ &= y_{i+1} + \left[\frac{1}{12}f_{yy}f^2 + \frac{1}{6}f f_{xy} + \frac{1}{12}f_{xx} - f_{xy} - f_y^2 f\right] h^3 + O(h^4), \\ &\equiv y_{i+1} + c(f)h^3 + O(h^4). \end{aligned}$$

It then follows that the local error, LE, satisfies

$$LE = c(f)h^3 + O(h^4),$$

where $c(f)$ is a complicated function of f. There are two general strategies for estimating $c(f)$ – the use of "step halving" and the use of a 3^{rd} order "companion formula".

Step Halving: let $\hat{y}_{i+1}$ be the approximation to $z_i(x_{i+1})$ computed with two steps of size $h/2$. If $c(f)$ is almost constant then we can show

$$z_i(x_{i+1}) = \hat{y}_{i+1} + 2c(f)(\frac{h}{2})^3 + O(h^4)$$

and from above

$$z_i(x_{i+1}) = y_{i+1} + c(f)h^3 + O(h^4).$$

Therefore the local error associated with $\hat{y}_{i+1}$, $\widehat{LE}$, is

$$\begin{aligned} \widehat{LE} &= 2c(f)(\frac{h}{2})^3 + O(h^4), \\ &= \frac{1}{3}(y_{i+1} - \hat{y}_{i+1}) + O(h^4). \end{aligned}$$

The method could then compute $\hat{y}_{i+1}, y_{i+1}$ and accept $\hat{y}_{i+1}$ only if $\frac{1}{3}|y_{i+1} - \hat{y}_{i+1}| < h\,TOL$.

Note that this strategy requires five derivative evaluations on each step and assumes that each of the components of $c(f)$ is slowly varying.

Third Order Companion Formula: To estimate the local error associated with the Modified Euler formula consider the use of a 3-stage, 3^{rd} order Runge-Kutta formula,

$$\begin{aligned} \hat{y}_{i+1} &= y_i + h(\hat{\omega}_1\hat{k}_1 + \hat{\omega}_2\hat{k}_2 + \hat{\omega}_3\hat{k}_3), \\ &= z_i(x_{i+1}) + O(h^4), \end{aligned}$$

We also have

$$\begin{aligned} y_{i+1} &= y_i + \frac{h}{2}(k_1 + k_2), \\ &= z_i(x_{i+1}) - c(f)h^3 + O(h^4). \end{aligned}$$

Subtracting these two equations we have the local error estimate,

$$est_i \equiv (\hat{y}_{i+1} - y_{i+1}) = c(f)h^3 + O(h^4).$$

Note that, for any 3^{rd} order formula, $k_1 = \hat{k}_1$, so we require at most 4 derivative evaluations per step to compute both y_{i+1} and est_i. Furthermore, if $\hat{\alpha}_2 = \alpha_2 = 1$ and $\hat{\beta}_{21} = \beta_{21} = 1$, we have $\hat{k}_2 = k_2$ and the cost is only three derivative evaluations per step. The obvious question is can one derive such a 3-stage 3^{rd} order Runge-Kutta formula ? The answer is yes and the following tableau with $\hat{\alpha}_3 \neq 1$ defines a one-parameter family of such "companion formulas" for the Modified Euler formula:

$$\begin{array}{c|ccc} - & - & & \\ 1 & 1 & - & \\ \hat{\alpha}_3 & \hat{\beta}_{31} & \hat{\beta}_{32} & - \\ \hline & \hat{\omega}_1 & \hat{\omega}_2 & \hat{\omega}_3 \end{array}$$

with

$$\hat{\beta}_{31} = \hat{\alpha}_3^2, \hat{\beta}_{32} = \hat{\alpha}_3 - \hat{\alpha}_3^2,\ \hat{\omega}_2 = \frac{1}{6(\hat{\alpha}_3 - 1)},\ \hat{\omega}_3 = \frac{-1}{6(\hat{\alpha}_3 - 1)},\ \hat{\omega}_1 = 1 - (\frac{1 + 3\hat{\alpha}_3}{6\hat{\alpha}_3}).$$

(c) **Generalization to Higher Order:**

Both of these approaches generalize to higher order Runge-Kutta methods. In particular, the idea of using a "companion formula" of order $p+1$ to estimate the local error of a p^{th} order formula leads to the derivation of s-stage, order $(p, p+1)$ formula pairs with the fewest number of stages. Explicit formula pairs of this type can be characterized by the tableau:

$$\begin{array}{c|ccccc} - & - & & & & \\ \alpha_2 & \beta_{21} & - & & & \\ \vdots & \vdots & & & & \\ \alpha_s & \beta_{s1} & \dots & & \beta_{s-1,s} & - \\ \hline & \omega_1 & \omega_2 & \dots & & \omega_s \\ & \hat{\omega}_1 & \hat{\omega}_2 & \dots & & \hat{\omega}_s \end{array}$$

where

$$\begin{aligned} y_{i+1} &= y_i + h\sum_{r=1}^{s} \omega_r k_r \\ &= z_i(x_{i+1}) - c(f)h^{p+1} + O(h^{p+2}), \end{aligned}$$

$$
\begin{aligned}
\hat{y}_{i+1} &= y_i + h \sum_{r=1}^{s} \hat{\omega}_r k_r \\
&= z_i(x_{i+1}) + O(h^{p+2}), \\
est_i &= (\hat{y}_{i+1} - y_{i+1}) \\
&= c(f) h^{p+1} + O(h^{p+2}).
\end{aligned}
$$

Note that the error estimate is a reliable estimate of the local error associated with the lower order (order p) formula. The following table gives the fewest number of stages required to generate formula pairs of a given order.

order pair	(2,3)	(3,4)	(4,5)	(5,6)	(6,7)	(7,8)	(8,9)
fewest stages	3	4	6	8	10	13	17

The widespread development and use of variable stepsize methods over the last 40-50 years has been accompanied by significant improvements in the effectiveness of the underlying formulas and the associated local error estimates.

- Prior to the 1960's the only real alternative was to use 4^{th} order formulas with step halving to estimate the local error. Two example formulas that were widely used as the basis for software at that time were the classical 4^{th} order formula proposed by Runge in 1895 and the 4^{th} order formula proposed by Gill (and discussed in [Gear pp. 83-84]).
- In 1957 Merson showed that, for $y' = A(x)y + w(x)$, the formula pair,

$$
\begin{array}{c|ccccc}
- & - & & & & \\
1/3 & 1/3 & - & & & \\
1/3 & 1/6 & 1/6 & - & & \\
1/2 & 1/8 & 0 & 3/8 & - & \\
1 & 1/2 & 0 & -3/2 & 2 & - \\
\hline
 & 1/6 & 0 & 0 & 2/3 & 1/6 \\
 & 1/2 & 0 & -3/2 & 2 & 0
\end{array}
$$

satisfies

$$
\begin{aligned}
y_{i+1} &= z_i(x_{i+1}) - \frac{1}{720} h^5 y^{(5)} + O(h^6), \\
\hat{y}_{i+1} &= z_i(x_{i+1}) - \frac{1}{120} h^5 y^{(5)} + O(h^6),
\end{aligned}
$$

That is, this 'formula pair' consists of two 4^{th} order formulas with $s = 5$ that provides an asymptotically correct local error estimate only for linear problems. For such problems one can estimate the local error by,

$$
est_i = \frac{1}{5}(y_{i+1} - \hat{y}_{i+1}) = \frac{1}{720} h^5 y^{(5)} + O(h^6).
$$

- In the 1960s and earlier Fehlberg (working at NASA Houston) derived a large class of Runge Kutta formula pairs that were suitable for all sufficiently smooth problems. An example of a particularly popular order $(4, 5)$ formula pair is:

-	-					
1/4	1/4	-				
3/8	3/32	9/32	-			
12/13	1932/2197	-7200/2197	7296/2197	-		
1	439/216	-8	3680/513	-845/4104	-	
1/2	-8/27	2	-3544/2565	1859/4104	-11/40	-
	25/216	0	1408/2565	2197/4104	-1/5	0
	-16/135	0	6656/12825	28561/56430	-9/50	2/55

- In the last 3 decades Fehlberg and others (Verner, Dormand, Sharp, Bettis, Prince) have proposed such formula pairs with orders up to $(8, 9)$.

5. **Stepsize Control:**

- Step is accepted only if $|est_i| < hTOL$.
- If h is too large, the step will be rejected and the derivative evaluations will be wasted.
- If h is too small, there will be many steps and more function evaluations than necessary.

The usual strategy for choosing the attempted stepsize, h, for the next step is based on 'aiming' at the largest h which will result in an accepted step on the current step. If we assume that $c(f)$ is slowly varying then our current estimate satisfies,

$$|est_i| = |c(f)|h_{i+1}^{p+1} + O(h^{p+2}),$$

and, on the next step attempted step $h_{i+2} = \gamma h_{i+1}$, we want

$$|est_{i+1}| \approx TOL\ h_{i+2}.$$

But

$$\begin{aligned} |est_{i+1}| &\approx |c(f)|(\gamma h_{i+1})^{p+1}, \\ &= \gamma^{p+1}|est_i|. \end{aligned}$$

We can then expect

$$|est_{i+1}| \approx TOL\ h_{i+2},$$

if

$$\gamma^{p+1}|est_i| \approx TOL\ (\gamma h_{i+1}),$$

which is equivalent to

$$\gamma^p|est_i| \approx TOL\ h_{i+1}.$$

The choice of γ to satisfy this heuristic is then,

$$\gamma = \left(\frac{TOL\ h_{i+1}}{|est_i|}\right)^{1/p}.$$

A typical step-choosing heuristic, justified by the above discussion, is to use the formula,

$$h_{i+2} = .9\left(\frac{TOL\ h_{i+1}}{|est_i|}\right)^{1/p} h_{i+1},$$

where .9 is a 'safety factor'. The formula works for use after a rejected step as well but must be modified slightly when round-off errors are significant.

6. Interpolants for RK formula pairs (ie. defining a suitable computable $S(x)$):

   For an s-stage, order $(p, p+1)$ RK formula pair one can determine suitable computable interpolants, $S(x)$, on the mesh $a = x_0 < x_1 \cdots < x_{N_{TOL}} = b$ (associated with the underlying discrete method). There are several alternatives possible and we will consider one of the least expensive alternatives. There is a trade-off that arises between the cost and reliabilty of the resulting implementation.

   A p^{th}-order, s-stage RK formula determines

   $$y_{i+1} = y_i + h_{i+1}\sum_{j=1}^{s}\omega_j k_j,$$

   where

   $$k_j = f(x_i + h_{i+1}\alpha_j, y_i + h_{i+1}\sum_{r=1}^{s}\beta_{jr}k_r).$$

   A Continuous extension (CRK) is determined by adding $(\bar{s} - s)$ extra stages to obtain an order p approximation for $x \in (x_i, x_{i+1})$

   $$u_i(x) = y_i + h_{i+1}\sum_{j=1}^{\bar{s}} b_j(\frac{x - x_i}{h_{i+1}})k_j,$$

   where $b_j(\tau)$ is a polynomial of degree p and $\tau = \frac{x - x_i}{h_{i+1}}$.

   - The $[u_i(x)]_{i=1}^{N_{TOL}}$ define the piecewise polynomial $S(x)$ for $x \in [a, b]$. This will be considered the numerical solution generated by the CRK method.
   - $S(x) \in C^0[a, b]$ and will interpolate the underlying discrete RK values, y_i, if $b_j(1) = \omega_j$ for $j = 1, 2 \cdots s$ and $b_{s+1}(1) = b_{s+2}(1) = \cdots b_{\bar{s}}(1) = 0$.
   - Similarly a simple set of constraints on the $b_j'(\tau)$, will ensure $S'(x)$ interpolates $f(x_i, y_i)$ and therefore $S(x) \in C^1[a, b]$.

Formula	p	s	$\bar{s}$
CRK4	4	4	6
CRK5	5	6	9
CVSS6B	6	7	11
CVSS7	7	9	15
CVSS8	8	13	21

Table 1: Cost per step of some CRK formulas

7. **Local extrapolation** One can use an asymptotically correct estimate of the local error to improve the accuracy (and order) of the numerical solution.

   - Recall, if the local error estimate is asymptotically correct we have for a p^{th}order formula:

     $$est_i = (z_i(x_i + h) - y_{i+1}) + \hat{c}h^{p+2} + O(h^{p+3}),$$

     and therefore,

     $$\hat{y}_{i+1} = y_{i+1} + est_i = z_i(x_i + h) + \hat{c}h^{p+2} + O(h^{p+3}).$$

     That is, 'Adding in the estimate' increases the order by one.
   - For a RK formula pair of order $(p, p+1)$ we have $est_i = \hat{y}_{i+1} - y_{i+1}$, and therefore $y_{i+1} + est_i = \hat{y}_{i+1}$.
   - We sacrifice any 'estimate' of the local error (in $\hat{y}_{i+1}$) if we advance with $\hat{y}_{i+1}$.
   - Several current methods indirectly control the LEPUS of $\hat{y}_{i+1}$, by estimating and bounding the LEPS of y_{i+1}.
     That is, if the error estimate is asymptotically correct and $y_{i+1} = z_i(x_i + h) + ch^{p+1} + O(h^{p+2})$, $\|est_i\| = |c|h^{p+1} + O(h^{p+2})$, then LEPS, $\|est_i\| < TOL$ implies,

     $$|c|h^{p+1} < TOL + O(h^{p+2}) \Rightarrow |c|h^{p+2} < TOLh + O(h^{p+3}).$$

     This then implies,

     $$|\hat{c}|h^{p+2} < |\frac{\hat{c}}{c}|TOLh + O(h^{p+3}),$$

     or the EPUS for $\hat{y}_{i+1} = y_{i+1} + est_i < |\frac{\hat{c}}{c}|TOL + O(h^{p+2})$. Note that this observation applies to any IVP method with an error estimate, but $|\frac{\hat{c}}{c}|$ can vary considerably for different RK formulas and depends on $f(x, y)$.

2.4 An example of software and its use:

Any general purpose method for IVPs which provides a continuous approximation to the IVP can be used in a problem solving environment to investigate interesting questions arising in particular application areas. As an example we will consider a typical question that may arise in the study of a predator-prey relationship in biology.

A predator-prey relationship can be modeled by the IVP:

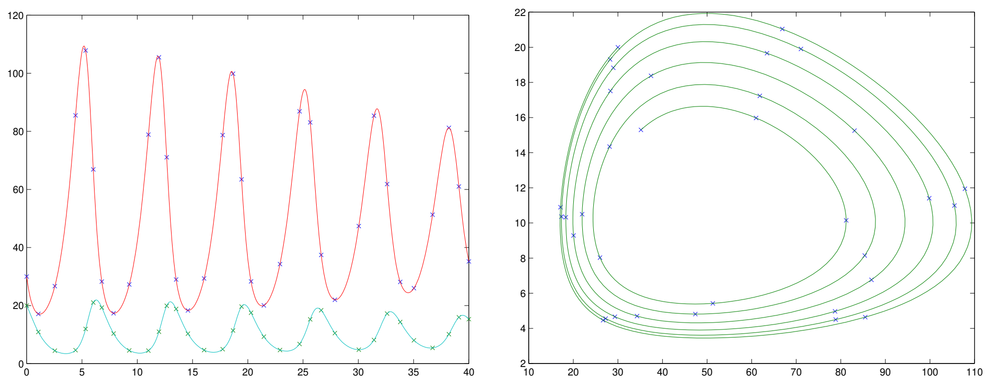
$$y_1' = y_1 - 0.1 y_1 y_2 + 0.02x$$

$$y_2' = -y_2 + 0.02 y_1 y_2 + 0.008x$$

with

$$y_1(0) = 30, \quad y_2(0) = 20.$$

Here $y_1(x)$ represents the 'prey' population at time x and $y_2(x)$ represents the 'predator' population at time x. The solution can then be visualized as a standard x/y solution plot or by a 'phase plane' plot. Figure 1 illustrates the solution to this system. We know that for different initial conditions solutions to this problem exhibit oscillatory behaviour as x increases.



(a) Solution Plot

(b) Phase Plane Plot

Figure 1: Solutions to the predator prey problem for $x \in [0, 40]$

A biologist may be interested in whether the solutions to this equation are 'almost periodic' (in the sense that the difference between successive maximum is constant) and whether the local maxima approach a steady state exponentially. (See Figure 2).

In the next few pages we will illustrate how one can use the methods, DVERK and ode45 (of matlab), to solve this test problem and to investigate this particular question.

We first identify the generic calling sequence used by the DVERK family of IVP methods. The only thing that changes for different members of this family, is the required size of the workspace and the location (within the workspace array, W) of the 'next' discrete solution vector, y_{i+1}.

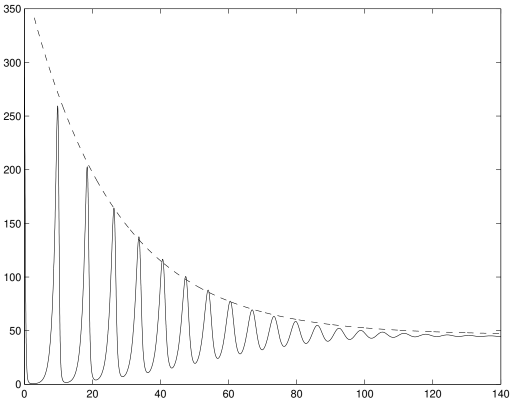


Figure 2: Typical behaviour of prey or predator population and decay to steady state

```
C This file contains the internal documentation only for DVERK. The
C documentation also is relevant to the complete suite of codes developed
C with the same design and calling sequence as DVERK. In this latter case
C the calling sequence is intepreted in the same way, but the underlying
C strategies may have been modified.
C
C     SUBROUTINE DVERK(N, FCN, X, Y, XEND, TOL, IND, C, NW, W)
C
C**********************************************************************
C
C     PURPOSE - THIS IS A RUNGE-KUTTA  SUBROUTINE  BASED  ON  VERNER'S
C FIFTH AND SIXTH ORDER PAIR OF FORMULAS FOR FINDING APPROXIMATIONS TO
C THE SOLUTION OF  A  SYSTEM  OF  FIRST  ORDER ORDINARY  DIFFERENTIAL
C EQUATIONS  WITH  INITIAL  CONDITIONS. IT ATTEMPTS TO KEEP THE GLOBAL
C ERROR PROPORTIONAL TO  A  TOLERANCE  SPECIFIED  BY  THE  USER.  (THE
C PROPORTIONALITY  DEPENDS  ON THE KIND OF ERROR CONTROL THAT IS USED,
C AS WELL AS THE DIFFERENTIAL EQUATION AND THE RANGE OF INTEGRATION.)
C
C     VARIOUS OPTIONS ARE AVAILABLE TO THE USER,  INCLUDING  DIFFERENT
C KINDS  OF  ERROR CONTROL, RESTRICTIONS ON STEP SIZES, AND INTERRUPTS
C WHICH PERMIT THE USER TO EXAMINE THE STATE OF THE  CALCULATION  (AND
C PERHAPS MAKE MODIFICATIONS) DURING INTERMEDIATE STAGES.
C
C     THE PROGRAM IS EFFICIENT FOR NON-STIFF SYSTEMS.  HOWEVER, A GOOD
C VARIABLE-ORDER-ADAMS METHOD WILL PROBABLY BE MORE EFFICIENT IF THE
C FUNCTION EVALUATIONS ARE VERY COSTLY.  SUCH A METHOD WOULD  ALSO  BE
C MORE SUITABLE IF ONE WANTED TO OBTAIN A LARGE NUMBER OF INTERMEDIATE
C SOLUTION VALUES BY INTERPOLATION, AS MIGHT BE THE CASE  FOR  EXAMPLE
C WITH GRAPHICAL OUTPUT.
C
```

```
C      HULL-ENRIGHT-JACKSON   1/10/76
C
C*************************************************************************
C
C     USE - THE USER MUST SPECIFY EACH OF THE FOLLOWING
C
C     N  NUMBER OF EQUATIONS
C
C   FCN  NAME OF SUBROUTINE FOR EVALUATING FUNCTIONS - THE  SUBROUTINE
C          ITSELF MUST ALSO BE PROVIDED BY THE USER - IT SHOULD BE OF
C          THE FOLLOWING FORM
C             SUBROUTINE FCN(N, X, Y, YPRIME)
C             INTEGER N
C             DOUBLE PRECISION X, Y(N), YPRIME(N)
C             ** ETC ***        *
C          AND IT SHOULD EVALUATE YPRIME, GIVEN N, X AND Y
C
C     X  INDEPENDENT VARIABLE - INITIAL VALUE SUPPLIED BY USER
C
C     Y  DEPENDENT VARIABLE - INITIAL VALUES OF COMPONENTS Y(1), Y(2),
C          ..., Y(N) SUPPLIED BY USER
C
C  XEND  VALUE OF X TO WHICH INTEGRATION IS TO BE CARRIED OUT - IT MAY
C          BE LESS THAN THE INITIAL VALUE OF X
C
C   TOL  TOLERANCE - THE SUBROUTINE ATTEMPTS TO CONTROL A NORM OF  THE
C          LOCAL  ERROR  IN  SUCH  A  WAY  THAT  THE  GLOBAL ERROR IS
C          PROPORTIONAL TO TOL. IN SOME PROBLEMS THERE WILL BE ENOUGH
C          DAMPING  OF  ERRORS, AS WELL AS SOME CANCELLATION, SO THAT
C          THE GLOBAL ERROR WILL BE LESS THAN TOL. ALTERNATIVELY, THE
C          CONTROL   CAN   BE VIEWED AS  ATTEMPTING TO  PROVIDE  A
C          CALCULATED VALUE OF Y AT XEND WHICH IS THE EXACT  SOLUTION
C          TO THE  PROBLEM Y' = F(X,Y) + E(X) WHERE THE NORM OF E(X)
C          IS PROPORTIONAL TO TOL.  (THE NORM IS  A  MAX  NORM  WITH
C          WEIGHTS  THAT  DEPEND ON THE ERROR CONTROL STRATEGY CHOSEN
C          BY THE USER.  THE DEFAULT WEIGHT FOR THE K-TH COMPONENT IS
C          1/MAX(1,ABS(Y(K))),  WHICH THEREFORE PROVIDES A MIXTURE OF
C          ABSOLUTE AND RELATIVE ERROR CONTROL.)
C
C   IND  INDICATOR - ON INITIAL ENTRY IND MUST BE SET EQUAL TO EITHER
C          1  OR  2. IF THE USER DOES NOT WISH TO USE ANY OPTIONS, HE
C          SHOULD SET IND TO 1 - ALL THAT REMAINS FOR THE USER TO  DO
C          THEN  IS  TO  DECLARE C AND W, AND TO SPECIFY NW. THE USER
C          MAY ALSO  SELECT  VARIOUS  OPTIONS ON  INITIAL  ENTRY  BY
C          SETTING IND = 2 AND INITIALIZING THE FIRST 9 COMPONENTS OF
```

```
C          C AS DESCRIBED IN THE NEXT SECTION.  HE MAY ALSO  RE-ENTER
C          THE  SUBROUTINE  WITH IND = 3 AS MENTIONED AGAIN BELOW. IN
C          ANY EVENT, THE SUBROUTINE RETURNS WITH IND EQUAL TO
C             3 AFTER A NORMAL RETURN
C             4, 5, OR 6 AFTER AN INTERRUPT (SEE OPTIONS C(8), C(9))
C             -1, -2, OR -3 AFTER AN ERROR CONDITION (SEE BELOW)
C
C     C  COMMUNICATIONS VECTOR - THE DIMENSION MUST BE GREATER THAN OR
C          EQUAL TO 24, UNLESS OPTION C(1) = 4 OR 5 IS USED, IN WHICH
C          CASE THE DIMENSION MUST BE GREATER THAN OR EQUAL TO N+30
C
C    NW  FIRST DIMENSION OF WORKSPACE W -  MUST  BE  GREATER  THAN  OR
C          EQUAL TO N
C
C     W  WORKSPACE MATRIX - FIRST DIMENSION MUST BE NW AND SECOND MUST
C          BE GREATER THAN OR EQUAL TO 9
C
C     THE SUBROUTINE  WILL  NORMALLY  RETURN  WITH  IND  =  3, HAVING
C REPLACED THE INITIAL VALUES OF X AND Y WITH, RESPECTIVELY, THE VALUE
C OF XEND AND AN APPROXIMATION TO Y AT XEND.  THE  SUBROUTINE  CAN  BE
C CALLED  REPEATEDLY  WITH NEW VALUES OF XEND WITHOUT HAVING TO CHANGE
C ANY OTHER ARGUMENT.  HOWEVER, CHANGES IN TOL, OR ANY OF THE  OPTIONS
C DESCRIBED BELOW, MAY ALSO BE MADE ON SUCH A RE-ENTRY IF DESIRED.
C
C     THREE ERROR RETURNS ARE ALSO POSSIBLE, IN WHICH  CASE  X AND  Y
C WILL BE THE MOST RECENTLY ACCEPTED VALUES -
C     WITH IND = -3 THE SUBROUTINE WAS UNABLE  TO  SATISFY  THE  ERROR
C       REQUIREMENT  WITH A PARTICULAR STEP-SIZE THAT IS LESS THAN OR
C       EQUAL TO HMIN, WHICH MAY MEAN THAT TOL IS TOO SMALL
C     WITH IND = -2 THE VALUE OF HMIN  IS  GREATER  THAN  HMAX,  WHICH
C       PROBABLY  MEANS  THAT THE REQUESTED TOL (WHICH IS USED IN THE
C       CALCULATION OF HMIN) IS TOO SMALL
C     WITH IND = -1 THE ALLOWED MAXIMUM NUMBER OF FCN EVALUATIONS  HAS
C       BEEN  EXCEEDED,  BUT  THIS  CAN ONLY OCCUR IF OPTION C(7), AS
C       DESCRIBED IN THE NEXT SECTION, HAS BEEN USED
C
C     THERE ARE SEVERAL CIRCUMSTANCES THAT WILL CAUSE THE CALCULATIONS
C TO  BE  TERMINATED,  ALONG WITH OUTPUT OF INFORMATION THAT WILL HELP
C THE USER DETERMINE THE CAUSE OF  THE      TROUBLE.  THESE  CIRCUMSTANCES
C INVOLVE  ENTRY WITH ILLEGAL OR INCONSISTENT VALUES OF THE ARGUMENTS,
C SUCH AS ATTEMPTING A NORMAL  RE-ENTRY  WITHOUT  FIRST  CHANGING  THE
C VALUE OF XEND, OR ATTEMPTING TO RE-ENTER WITH IND LESS THAN ZERO.
C
C*******************************************************************
C
```

```
C     OPTIONS - IF THE SUBROUTINE IS ENTERED WITH IND = 1, THE FIRST 9
C COMPONENTS OF THE COMMUNICATIONS VECTOR ARE INITIALIZED TO ZERO, AND
C THE SUBROUTINE USES ONLY DEFAULT VALUES  FOR      EACH  OPTION.  IF  THE
C SUBROUTINE  IS  ENTERED  WITH IND = 2, THE USER MUST SPECIFY EACH OF
C THESE 9 COMPONENTS - NORMALLY HE WOULD FIRST SET THEM ALL  TO  ZERO,
C AND  THEN  MAKE  NON-ZERO  THOSE  THAT  CORRESPOND TO THE PARTICULAR
C OPTIONS HE WISHES TO SELECT. IN ANY EVENT, OPTIONS MAY BE CHANGED ON
C RE-ENTRY  TO      THE  SUBROUTINE  -  BUT IF THE USER CHANGES ANY OF THE
C OPTIONS, OR TOL, IN THE COURSE OF A CALCULATION HE SHOULD BE CAREFUL
C ABOUT  HOW  SUCH CHANGES AFFECT THE SUBROUTINE - IT MAY BE BETTER TO
C RESTART WITH IND = 1 OR 2. (COMPONENTS 10 TO 24 OF C ARE USED BY THE
C PROGRAM  -  THE INFORMATION IS AVAILABLE TO THE USER, BUT SHOULD NOT
C NORMALLY BE CHANGED BY HIM.)
C
C  C(1)  ERROR CONTROL INDICATOR - THE NORM OF THE LOCAL ERROR IS  THE
C          MAX  NORM  OF  THE WEIGHTED  ERROR  ESTIMATE  VECTOR, THE
C          WEIGHTS BEING DETERMINED ACCORDING TO THE VALUE OF C(1) -
C             IF C(1)=1 THE WEIGHTS ARE 1 (ABSOLUTE ERROR CONTROL)
C             IF C(1)=2 THE WEIGHTS ARE 1/ABS(Y(K))  (RELATIVE  ERROR
C         CONTROL)
C             IF C(1)=3 THE  WEIGHTS  ARE  1/MAX(ABS(C(2)),ABS(Y(K)))
C         (RELATIVE  ERROR  CONTROL,  UNLESS ABS(Y(K)) IS LESS
C         THAN THE FLOOR VALUE, ABS(C(2)) )
C             IF C(1)=4 THE WEIGHTS ARE 1/MAX(ABS(C(K+30)),ABS(Y(K)))
C         (HERE INDIVIDUAL FLOOR VALUES ARE USED)
C             IF C(1)=5 THE WEIGHTS ARE 1/ABS(C(K+30))
C             FOR ALL OTHER VALUES OF C(1), INCLUDING C(1) = 0,  THE
C         DEFAULT  VALUES  OF  THE  WEIGHTS  ARE  TAKEN  TO BE
C         1/MAX(1,ABS(Y(K))), AS MENTIONED EARLIER
C          (IN THE TWO CASES C(1) = 4 OR 5 THE USER MUST DECLARE  THE
C          DIMENSION OF C TO BE AT LEAST N+30 AND MUST INITIALIZE THE
C          COMPONENTS C(31), C(32), ..., C(N+30).)
C
C  C(2)  FLOOR VALUE - USED WHEN THE INDICATOR C(1) HAS THE VALUE 3
C
C  C(3)  HMIN SPECIFICATION - IF NOT ZERO, THE SUBROUTINE CHOOSES HMIN
C          TO BE ABS(C(3)) - OTHERWISE IT USES THE DEFAULT VALUE
C             10*MAX(DWARF,RREB*MAX(WEIGHTED NORM Y/TOL,ABS(X))),
C          WHERE DWARF IS A VERY SMALL POSITIVE  MACHINE  NUMBER  AND
C          RREB IS THE RELATIVE ROUNDOFF ERROR BOUND
C
C  C(4)  HSTART SPECIFICATION - IF NOT ZERO, THE SUBROUTINE  WILL  USE
C          AN INITIAL  HMAG EQUAL TO ABS(C(4)), EXCEPT OF COURSE FOR
C          THE RESTRICTIONS IMPOSED BY HMIN AND HMAX  -  OTHERWISE IT
C          USES THE DEFAULT VALUE OF HMAX*(TOL)**(1/6)
```

```
C
C  C(5)  SCALE SPECIFICATION - THIS IS INTENDED TO BE A MEASURE OF THE
C          SCALE OF THE PROBLEM - LARGER VALUES OF SCALE TEND TO MAKE
C          THE METHOD MORE RELIABLE, FIRST  BY  POSSIBLY  RESTRICTING
C          HMAX  (AS  DESCRIBED  BELOW) AND SECOND, BY TIGHTENING THE
C          ACCEPTANCE REQUIREMENT - IF C(5) IS ZERO, A DEFAULT  VALUE
C          OF 1  IS  USED.  FOR  LINEAR  HOMOGENEOUS PROBLEMS  WITH
C          CONSTANT COEFFICIENTS, AN APPROPRIATE VALUE FOR SCALE IS A
C          NORM  OF  THE  ASSOCIATED  MATRIX. FOR OTHER PROBLEMS, AN
C          APPROXIMATION TO  AN  AVERAGE  VALUE  OF  A  NORM  OF  THE
C          JACOBIAN ALONG THE TRAJECTORY MAY BE APPROPRIATE
C
C  C(6)  HMAX SPECIFICATION - FOUR CASES ARE POSSIBLE
C          IF C(6).NE.0 AND C(5).NE.0, HMAX IS TAKEN TO BE
C             MIN(ABS(C(6)),2/ABS(C(5)))
C          IF C(6).NE.0 AND C(5).EQ.0, HMAX IS TAKEN TO BE  ABS(C(6))
C          IF C(6).EQ.0 AND C(5).NE.0, HMAX IS TAKEN TO BE
C             2/ABS(C(5))
C          IF C(6).EQ.0 AND C(5).EQ.0, HMAX IS GIVEN A DEFAULT  VALUE
C             OF 2
C
C  C(7)  MAXIMUM NUMBER OF FUNCTION EVALUATIONS  -  IF      NOT  ZERO,  AN
C          ERROR  RETURN WITH IND = -1 WILL BE CAUSED WHEN THE NUMBER
C          OF FUNCTION EVALUATIONS EXCEEDS ABS(C(7))
C
C  C(8)  INTERRUPT NUMBER  1  -  IF  NOT  ZERO,  THE  SUBROUTINE  WILL
C          INTERRUPT THE  CALCULATIONS  AFTER  IT  HAS  CHOSEN  ITS
C          PRELIMINARY VALUE OF HMAG, AND JUST BEFORE CHOOSING HTRIAL
C          AND  XTRIAL  IN  PREPARATION FOR TAKING A STEP (HTRIAL MAY
C          DIFFER FROM HMAG IN SIGN, AND MAY  REQUIRE ADJUSTMENT  IF
C          XEND  IS  NEAR) - THE SUBROUTINE RETURNS WITH IND = 4, AND
C          WILL RESUME CALCULATION AT THE POINT  OF  INTERRUPTION  IF
C          RE-ENTERED WITH IND = 4
C
C  C(9)  INTERRUPT NUMBER  2  -  IF  NOT  ZERO,  THE  SUBROUTINE  WILL
C          INTERRUPT THE  CALCULATIONS  IMMEDIATELY AFTER  IT  HAS
C          DECIDED WHETHER OR NOT TO ACCEPT THE RESULT  OF  THE  MOST
C          RECENT  TRIAL STEP, WITH IND = 5 IF IT PLANS TO ACCEPT, OR
C          IND = 6 IF IT PLANS TO REJECT -  Y(*)  IS  THE  PREVIOUSLY
C          ACCEPTED  RESULT, WHILE W(*,9) IS THE NEWLY COMPUTED TRIAL
C          VALUE, AND W(*,2) IS THE UNWEIGHTED ERROR ESTIMATE VECTOR.
C          THE  SUBROUTINE  WILL  RESUME CALCULATIONS AT THE POINT OF
C          INTERRUPTION ON RE-ENTRY WITH IND = 5 OR 6. (THE USER  MAY
C          CHANGE IND IN THIS CASE IF HE WISHES, FOR EXAMPLE TO FORCE
C          ACCEPTANCE OF A STEP THAT WOULD OTHERWISE BE REJECTED,  OR
```

```
C          VICE VERSA. HE CAN ALSO RESTART WITH IND = 1 OR 2.)
C
C***********************************************************************
C
C  SUMMARY OF THE COMPONENTS OF THE COMMUNICATIONS VECTOR
C
C     PRESCRIBED AT THE OPTION      DETERMINED BY THE PROGRAM
C     OF THE USER
C
C                                    C(10) RREB(REL ROUNDOFF ERR BND)
C     C(1) ERROR CONTROL INDICATOR   C(11) DWARF (VERY SMALL MACH NO)
C     C(2) FLOOR VALUE        C(12) WEIGHTED NORM Y
C     C(3) HMIN SPECIFICATION        C(13) HMIN
C     C(4) HSTART SPECIFICATION      C(14) HMAG
C     C(5) SCALE SPECIFICATION       C(15) SCALE
C     C(6) HMAX SPECIFICATION        C(16) HMAX
C     C(7) MAX NO OF FCN EVALS       C(17) XTRIAL
C     C(8) INTERRUPT NO 1            C(18) HTRIAL
C     C(9) INTERRUPT NO 2            C(19) EST
C                                    C(20) PREVIOUS XEND
C                                    C(21) FLAG FOR XEND
C                                    C(22) NO OF SUCCESSFUL STEPS
C                                    C(23) NO OF SUCCESSIVE FAILURES
C                                    C(24) NO OF FCN EVALS
C
C  IF C(1) = 4 OR 5, C(31), C(32), ... C(N+30) ARE FLOOR VALUES
C
C  RREB and DWARF are machine dependent constants currently set so
C  that they should be appropriate for most machines.  However, it may
C  be appropriate to change them when this program is installed on a
C  new machine.                        K.R.J.   3 Oct 1991.
C
```

Next we give an example of a typical Fortran driver to solve this predator-prey problem using DVERK and the interupt option to compute the associated piecewise polynomial, $S(x)$.

```
C
C  This is the FORTRAN driver to generate a discrete solution and
C  a piecewise polynomial solution from DVERK for the predator-prey
C  problem.
C
      integer i, j, k, ind, n, ref, nw
      double precision TT(200), SD(2,200), y(2), t, tend, c(30), tol,
     +    W(2,30), TTF(2000), SDF(2, 2000), delta
      external fcn
      data c/30*0.d0/
      tol = 1.d-2
      n = 2
      nw = 2
      ref = 20
      y(1) = 30.d0
      y(2) = 20.d0
      t = 0.d0
      tend = 40.d0
      TT(1) = t
      TTF(1) = t
      SD(1,1) = y(1)
      SD(2,1) = y(2)
      SDF(1,1) = SD(1,1)
      SDF(2,1) = SD(2, 1)
      print 110, 1, TTF(1), 1, SDF(1,1), SDF(2,1)
      print 130, 1, TT(1), 1, SD(1,1), SD(2,1)
      ind = 2
      c(9) = 1
      c(6) = 10.d0
      i = 1
      j = 1
 100  call dverk(n, fcn, t, y, tend, tol, ind, c, nw, W)
      if ( ind .eq. 6 ) go to 100
      if ((ind .eq. 3) .or. (ind .lt. 0)) go to 400
C
C The step is to be accepted (ind=5) so we store the discrete solution
C and compute (using intrp) the off-mesh interpolated solution on a fine
C grid of 'ref' equally spaced points.
C
         TT(i+1) = c(17)
         SD(1, i+1) = W(1,13)
```

```
      SD(2, i+1) = W(2,13)
      delta = c(18)/dfloat(ref)
      do 120 k = 1,ref
         j = (i-1)*ref + k + 1
         TTF(j) = t + delta*k
         call intrp(n, t, y, TTF(j), SDF(1,j), c(18), nw, W)
         print 110, j, TTF(j),j, SDF(1,j), SDF(2,j)
110      format(1x, 'TTF(', i4, ') = ', E15.8, ';'/,
   +       1x, 'SDF( 1:2 ,', i4, ') = [', 2E15.8, "]';")
120   continue
      i = i+1
      print 130, i, TT(i), i, SD(1,i), SD(2,i)
130   format(1x, 'TT(', i3, ') = ', E15.8, ';'/,
   +     1x, 'SD( 1:2,', i3, ') = [', 2E15.8, "]';" )
      go to 100
400 print 410, i, j
410 format(1x, 'II = ', i4, ';' /1x, 'JJ = ', i4, ';' )
    stop
    end

    subroutine fcn(n, t, y, yp)
    double precision t, y(2), yp(2)
    yp(1) = y(1) - .1d0*y(1)*y(2) + .02d0*t
    yp(2) = -y(2) + .02d0*y(1)*y(2) + .008d0*t
    return
    end
```

Finally we present a MatLab script that uses a modified version of ode45 (ode45ex) to solve this test problem and answer the question of interest. Note that with eropt = 1, ode45ex produces the same numerical solution as ode45.

```
 global NTOT TOLD YOLD PPT GPPC GPPC2 WOLD GPPC23
 global eropt
 load ART1;
 KK = 7;
  for eropt = 1:3
  for itol = 1:3
  TOL = 10.0^-(2*itol);
 TSPAN = [0.0  100.0];
 NSAMPLE = 1001;
 TSPAN2 = linspace(TSPAN(1), TSPAN(2), NSAMPLE);
 Y0 = [30; 20];
 n = length(Y0);
 options1 = odeset('abstol', TOL, 'reltol',TOL, 'stats', 'on');

 [TT, YY] = ode45ex(@ppde, TSPAN2, Y0, options1);

%
% Generate explicitly the underlying piecewise polynomial associated
% with the approximate solution of this problem.
%
 options2 = odeset(options1,'outputfcn', @GenPP, 'refine', 1, 'stats', 'off');
 if eropt > 1
   options2 = odeset(options2,'outputfcn', @GenPP23);
 end
 [TC1, YC1] = ode45ex(@ppde, TSPAN, Y0, options2);

 N = length(TC1);
%
% The global variables PPT and GPPC now represent  the pp associated
% with the approximate solution of the original problem.
% We can now accurately determine the first 14 local maximum of the first
% component (the prey population) and investigate whether the distance
% between successive local maximum is almost constant. We can also
% investigate whether the magnitude of these local maximums approaches
% a constant value exponentially.
%
%
% To evaluate the piecewise polynomial, S(x) with an underlying
% mesh x_0, x_1, ... x_N, we represent S(x) by the 3-D array, GPPC(1:N, 1:n, 1:6).
% Note that, in this case, 6 is the degree of the local polynomial u_i(x)
% associated with the i^th interval, [x_i, x_{i+1}] and n is problem dimension.
```

```
%
% We first extract the representation of u_i(x), as LPPC(1:n, 1:6)
% (= GPPC(i, 1:n, 1:6) ) and then 'evaluate' S(x) for x in [x_i, x_{i+1}] by
%
%       tau = (x - x_i)/(x_{i+1} - x_i)
%       v1 = tau .^ (0:5)
%       y1 = LPPC * v1
%
%  Similarly we can 'evaluate' the derivative of S(x),  S'(x), by
%
%       tau = (x - x_i)/(x_{i+1} - x_i)
%       v2 = (0:5) .* [0 tau .^ (0:4) ]
%       y1p = LPPC * v2 /(x_{i+1} - x_i)
%
% For the 2 special cases, x = x_i and x = x_{i+1}, (ie., tau = 0, tau = 1)
% the jth component of u_i(x) (for j = 1,2 ... n) satisfies,
%        y_i = u_i(x_i) = LPPC(j,1)
%        y_{i+1} = u_i(x_{i+1}) = LPPC(j,1) + LPPC(j,2) ...+ lPPC(j,6)
%        y_i' = u_i'(x_i) = LPPC(j,2)/(x_{i+1} - x_i)
%        y_{i+1}' = u_i'(x_{i+1}) = [LPPC(j,2) + 2 LPPC(j,3) .. + 5 LPPC(j,6)]/(x_{i+1}
%
% To convert from this representation of u_i(x) and u_i'(x) to the standard
% MATLAB representation we note,
%
% u_i(x) = c_1 x^5 + c_2 x^4 ... c_6
% u_i'(x) = (5 c_1) x^4 + (4 c_2) x^3 .. + (c_5)
%          = d_1 x^4 + d_2 x^3 .. + d_4
%
% Therefore we have
%   c(m) = L(j, 7-m)  for m = 1 .. 6.
% and
%   d(m) = (5-m)c_m for m = 1 ..5.
%
 r = 1;
 diff(1) = 0.e0;
 ratio(1) = 1.e0;
for i = 1:(N-1)
  LPPC = zeros(n,6);
  if eropt == 1
   for k = 1:n
     for m = 1:5
        LPPC(k, m) = GPPC(i,k, m);
     end
     LPPC(k, 6) = 0;
   end
```

```
 end
 if eropt > 1
  for k = 1:n
    for m = 1:6
      LPPC(k, m) = GPPC23(i, k, m);
    end
  end
end
%
% Check for a local maximum only on those intervals where
% y_i' > 0 and y_{i+1}' < 0.
%
   h = PPT(i+1) - PPT(i);
   if LPPC(1,2) > 0 & LPPC(1, :) * (0:5)' < 0
% find the local maximum of the first component of u_i(x).
%
% First determine c(tau) and d(tau) the two local polynomials
% corresponding to the first component of u_i(x_i + tah h),
% and u_i'(x_i + tah h), respectively. Then, using the zeros
% of d(tau), determine the location and value of the maximum
% of u_i(x) in this interval.
%
      for m = 1:5;
         c(m) = LPPC(1,7-m);
         d(m) = (6-m) * c(m);
      end
      c(6) = LPPC(1,1);
      rr = roots(d);
      xr = PPT(i) + h * rr;
%     [i PPT(i) PPT(i+1) LPPC(1,1) xr']
      for m = 1:length(rr) ;
         xt = xr(m);
         if imag(xt) == 0
           if (xt > PPT(i)) & (xt < PPT(i+1))
              TM(r) = xt;
              tau = (xt - PPT(i))/h;
              YM(r) = polyval(c, tau);
              LY(r) = log(YM(r) - 46.e0);
              if r > 1
                 diff(r) = TM(r) - TM(r-1);
                 ratio(r) = (YM(r) - 46.e0)/ (YM(r-1) - 46.e0);
              end
%             [i, r]
%             [PPT(i), PPT(i+1), TM(r), YM(r), diff(r), ratio(r)]
              r = r +1;
```

```
            end
          end
     end
   end
 end
%
% We now consider whether the numerical solution can be identified
% as being 'almost periodic' from the computed results at this tolerance.
% Note that for sufficiently stringent tolerances, the numerical solution
% should be recognized as being 'almost periodic' if the underlying
% true solution is.
%
%
 [PP, SS] = polyfit(KK:15, TM(KK:15), 1)
%
 [P1,S1] = polyfit(TM(2:12), LY(2:12), 1);
 A1 = exp(P1(2));
 B1 = P1(1);
 YL = 46.e0 + A1 * exp(B1*TT);
%
% Visualize the approximate solution using standard x/y plot and
% a phase plane plot of the pp as well as that of the discrete solution.
%
 figure(1);
 plot(TT, YY(:,1), '-', TC1, YC1(:, 1), 'x', TT, YL, '--', TM, YM, 'o')
 figure(2);
 plot(YY(:,1), YY(:,2), '-', YC1(:,1), YC1(:,2), 'x')
 figure(3)
plot(TT, YY(:,1), '-',TM, YM, 'o')
figure(4)
plot(TT, YL, '--', TM, YM, 'o')
ErrTM = abs(TM - TMtrue) ./ TMtrue;
ErrYM = abs(YM - YMtrue) ./ YMtrue;
ErrA1 = abs(A1 - A1true)/abs(A1true);
ErrB1 = abs(B1 - B1true)/abs(B1true);
ErrY = abs(YY - Ytrue) ./ Ytrue ;
 eropt
 TOL
[ErrA1, ErrB1, max(ErrTM), max(ErrYM), max(ErrY) ]
end
end
```

2.5 Multistep Methods:

1. **A Historical Perspective:**

   1883 Bashforth and Adams proposed explicit formulas of the form,

   $$y_{i+1} = y_i + h\sum_{j=1}^{k} \beta_j y^{'}_{i+1-j}$$

   1900 Runge and Kutta developed 4^{th}-order explicit RK formulas.

   1914-1918 Moulton introduced implicit multistep formulas and the idea of a predictor/corrector scheme (in order to effectively solve ballistics equations).

   1926 Milne introduced estimates of the local error based on the difference between the predictor and corrector.

   1956-59 Dahlquist developed the asymptotic ($h \to 0$) stability theory and established stability and convergence theorems for multistep formulas.

   1963 Dahlquist introduced absolute stability and identified the difficulty of stiffness.

   1968-1975 Efficient variable-stepsize, variable order multistep methods appear: (Krogh, Sedgwick, Gear, Shampine).

2. A Linear Multistep Formula (LMF) has the form,

   $$\text{LMF:}\ \ y_{i+1} = \sum_{j=1}^{k} \alpha_j y_{i+1-j} + h\sum_{j=0}^{k} \beta_j y'_{i+1-j}.$$

   This formula is implicit if $\beta_0 \neq 0$ and implicit formulas are usually preferred because of their superior stability and accuracy properties. For implicit formulas we have the RHS of this equation contains the term, $\beta_0 y'_{i+1} = \beta_0 f(x_{i+1}, y_{i+1})$, and therefore a nonlinear system of equations must be solved on each step.

3. To solve the nonlinear systems of equations, a Predictor-Corrector (P-C) iteration is usually used,

   $$\text{P:}\ \ y^0_{i+1} = \sum_{j=1}^{k} \bar{\alpha}_j y_{i+1-j} + h\sum_{j=1}^{k} \bar{\beta}_j y'_{i+1-j}\ \ \text{an explicit formula}$$

   Iterate or 'Correct' for $r = 0, 1 \cdots M$:

   $$\begin{aligned} \text{E:}\ \ y'^{(r)}_{i+1} &= f(x_{i+1}, y^r_{i+1}),\ \ \text{'Evaluate'} \\ \text{C:}\ \ y^{r+1}_{i+1} &= \sum_{j=1}^{k} \alpha_j y_{i+1-j} + h\sum_{j=1}^{k} \beta_j y'_{i+1-j} + h\beta_0 y'^{(r)}_{i+1},\ \ \text{'Correct'} \end{aligned}$$

   - This iteration is denoted $P(EC)^M$ or $P(EC)^M E$.

- If the order of the predictor formula is at least as large as the order of the corrector and $\bar{\alpha}_j = \alpha_j$, for $j = 1, 2 \cdots k$, then an appropriate multiple of their difference can be used to estimate the local error (this idea is due to Milne).
- Convergence of this iteration is guaranteed provided $\|h\frac{\partial f}{\partial y}\| < \frac{1}{|\beta_0|}$. This follows from subtracting the corrector formula (C) from the exact formula (LMF) to obtain,

$$\begin{aligned} y_{i+1} - y_{i+1}^{r+1} &= h\beta_0[f(x_{i+1}, y_{i+1}) - f(x_{i+1}, y_{i+1}^r)] \\ &= h\beta_0 \frac{\partial f}{\partial y}|_\eta (y_{i+1} - y_{i+1}^r). \end{aligned}$$

  This condition is found to be 'almost necessary' in practice.
- $M = 1$ or $M = 2$ has been found to be optimal for most standard problems. With this choice the error estimate can detect any lack of convergence of the iteration (and reduce the stepsize to improve the chances for convergence).

4. Classical Results for uniform mesh (constant stepsize) Multistep Methods:
   - A LMF is Stable (as $h \to 0$) iff the associated characteristic polynomial, $\alpha(z) = z^k - \sum_{j=1}^k \alpha_j z^{k-j}$ has all its roots inside or on the unit circle.
     The necessity part of this result follows from considering $y' = \lambda y$, fixed h, and studying the behaviour of the resulting linear difference equation,

$$[1 - h\lambda\beta_0]y_{i+1} = \sum_{j=1}^k [\alpha_j + h\lambda\beta_j] y_{i+1-j}$$

     The solution of this difference equation can be expressed in terms of the roots of the corresponding characteristic polynomial,

$$\rho(z) = \alpha(z) - (h\lambda)\beta(z), \quad \text{with} \quad \beta(z) = \sum_{j=0}^k \beta_j z^{k-j}.$$

     That is, the y_i satisfy,

$$y_i = \sum_{j=1}^k \mu_j \xi_j^i.$$

   - A LMF is Consistent if $\alpha(1) = 0$ and $\alpha'(1) = \beta(1)$.
   - Key Theorem:– A LMF is Convergent iff it is Consistent and Stable.
   - Order of Convegence for a LMF:
     (a) Recall: The Local Truncation Error (LTE) for a LMF is defined, for sufficiently differentiable $y(x)$, to be,

$$LTE \equiv L(h, y(x)) \equiv y(x_i+h) - \sum_{j=1}^k \alpha_j y(x_i+(1-j)h) - h\sum_{j=0}^k \beta_j y'(x_i+(1-j)h).$$

(b) Recall: A formula is of order p iff $LE = O(h^{p+1})$.

(c) **Key Result:** Stability + $LTE = O(h^{p+1}) \Leftrightarrow LE = O(h^{p+1}) \Rightarrow$ $y(x_i) - y_i = O(h^p)$.

- Connection between Local Error (LE) and LTE for LMF.
  A stable order p LMF satisfies,

$$\begin{aligned} LE &= ch^{p+1} + O(h^{p+2}) \\ LTE &= \hat{c}h^{p+1} + O(h^{p+2}) \end{aligned}$$

where

$$\hat{c} = (k - \sum_{j=1}^{k}(k-j)\alpha_j)c \;=\; \alpha'(1)c \;= \beta(1)c,$$

(if the formula is consistent).

5. Derivation of high order LMFs:

- One can expand the LTE, $L(h, y(x))$, in a power series in h and choose the coefficients $\alpha_1, \alpha_2 \cdots \alpha_k, \beta_0, \beta_1 \cdots \beta_k$ to satisfy $LTE = O(h^{p+1})$, for as large a p as possible, subject to the stability constraint.
  To do this note,

$$y(x_i + (1-j)h) = y(x_{i+1} - jh) = \sum_{r \geq 0} \frac{(-j)^r}{r!} y^{(r)}(x_{i+1})h^r,$$

$$hy'(x_i + (1-j)h) = hy'(x_{i+1} - jh) = \sum_{r \geq 1} \frac{(-j)^{r-1}}{(r-1)!} y^{(r)}(x_{i+1})h^r.$$

Now substituting these expressions into the above definition of LTE (for LMF), we obtain,

$$L(h, y(x)) = D_0 y(x_{i+1}) + D_1 y'(x_{i+1})h + \cdots D_{p+1} y^{(p+1)}(x_{i+1})h^{p+1} + O(h^{p+2}).$$

where each D_r is linear in the unknowns (the α's and the β's). More precisely, the expansion is:

$$\begin{aligned} L(h, y(x)) &= y(x_{i+1}) - \sum_{j=1}^{k}\alpha_j(\sum_{r\geq 0}\frac{(-j)^r}{r!}y^{(r)}(x_{i+1})h^r) - \sum_{j=0}^{k}\beta_j(\sum_{r\geq 1}\frac{(-j)^{r-1}}{(r-1)!}y^{(r)}(x_{i+1})h^r) \\ &= (1 - \sum_{j=1}^{k}\alpha_j)y(x_{i+1}) - \sum_{r\geq 1}\left[\sum_{j=1}^{k}\frac{(-j)^r\alpha_j}{r!} - \sum_{j=0}^{k}\frac{(-j)^{r-1}\beta_j}{(r-1)!}\right]y^{(r)}(x_{i+1})h^r. \end{aligned}$$

and the coefficient D_r of $y^{(r)}(x_{i+1})h^r$ is therefore,

$$D_0 = 1 - \sum_{j=1}^{k}\alpha_j = \alpha(1),$$

$$D_r = -\sum_{j=1}^{k} \alpha_j \frac{(-j)^r}{r!} - \sum_{j=0}^{k} \beta_j \frac{(-j)^{r-1}}{(r-1)!} \text{ for } r \geq 1.$$

- Setting $D_r = 0$ for $r = 0, 1 \cdots$ for as many terms as possible results in a linear system of equations. If each equation is linearly independent we should be able to solve the set $D_r = 0$ for $r = 0, 1 \cdots 2k$. That is, $(2k+1)$ linear equations in the $(2k+1)$ unknowns.
- This system is always non-singular (for any k) and hence we can always find a k-step LMF with $LTE = O(h^{2k+1})$. Will such a formula be stable? Recall that the stability condition is a nonlinear constraint on the α's !
- One could always choose the α's to guarantee stability and then choose the remaining unknowns, $\beta_0, \beta_1 \cdots \beta_k$ to satisfy $D_1 = D_2 \cdots = D_{k+1} = 0$. (Note that $D_0 = \alpha(1)$ is independent of the β's.) The candidates (for $\alpha(z)$) are then,

$$\alpha(z) = (z-1) \prod_{j=1}^{k-1} (z - \eta_j), \text{ where each } |\eta_j| \leq 1.$$

  Such a system will always be nonsingular and we can derive formulas of order $k+1$ in this way. Can we do better?
- Dahlquist's Key Result (1956):
  (a) The maximum order of a k-step, stable LMF is $k+1$ if k is odd and $k+2$ if k is even. For even k, the order $k+2$ formula will only be marginally stable (all the roots of $\alpha(z)$ will be on the unit circle).
  (b) The obvious choice of $\alpha(z) = z^k - z^{k-1}$, leads to Adams Formulas.

6. Special Class of LMF: Adams Formulas.

   Adams Moulton Corrector formula of order $k+1$:

$$y_{i+1} = y_i + h \sum_{j=0}^{k} \beta_j y'_{i+1-j}.$$

   Adams Bashforth Predictor formula of order $k+1$,

$$y_{i+1} = y_i + h \sum_{j=1}^{k+1} \bar{\beta}_j y'_{i+1-j}.$$

   These are the most widely used LMFs for standard IVPs. The corresponding coefficients, β_j's and the $\bar{\beta}_j$'s can be determined by solving the above linear equations. (Recall that Adams Bashforth formulas are explicit and therefore $\bar{\beta}_0 = 0$.)

7. Extension of Adams LMFs to allow variable stepsize:

   - One can write the local solution to an IVP evaluated at $x_i + h$, $z_i(x_i + h)$, as

$$z_i(x_i + h) = y_i + \int_{x_i}^{x_i+h} f(s, z_i(s)) ds.$$

Adams formulas can be interpreted as being based on approximating the integral on the RHS of this expression by the integral of a polynomial that interpolates $f(s, z_i(s))$ at $x_{i+1} - jh$, $(j = 0, 1 \cdots k)$ using $y_{i+1-j} \approx z_i(x_{i+1-j})$. This interpretation yields a closed-form expression for the constant stepsize coefficients, $\beta_r, \bar{\beta}_r$ for the k-step AM and AB formulas. It also suggests a natural extension of these formulas to the case of variable stepsize.

- Let $\bar{P}_{i,k}(s; y'_i, y'_{i-1} \cdots y'_{i-k})$ be the unique polynomial (in s) of degree $\leq k$ satisfying,

$$\bar{P}_{i,k}(x_{i+1-j}; y'_i, y'_{i-1} \cdots y'_{i-k}) = y'_{i+1-j}, = f(x_{i+1-j}, y_{i+1-j}); \quad j = 1, 2 \cdots (k+1),$$

where the x_i's need not be equally spaced. Then, in Lagrange form we have,

$$\bar{P}_{i,k}(s; y'_i, y'_{i-1} \cdots y'_{i-k}) = \sum_{j=0}^{k} \bar{l}_j(s) y'_{i-j},$$

where

$$\bar{l}_j(s) = \frac{\prod_{r=0,\, r\neq j}^{k}(s - x_{i-r})}{\prod_{r=0,\, r\neq j}^{k}(x_{i-j} - x_{i-r})}.$$

Now, letting $\mu = (s - x_i)/h$ we have ,

$$\int_{x_i}^{x_{i+1}} \bar{l}_j(s) ds = h \int_0^1 \bar{l}_j(x_i + \mu h) d\mu,$$

and

$$\bar{l}_j(x_i + \mu h) = \frac{\prod_{r=0,\, r\neq j}^{k}(\mu h + x_i - x_{i-r})}{\prod_{r=0,\, r\neq j}^{k}(x_{i-j} - x_{i-r})},$$

which simplifies, <u>if h is constant</u>, to

$$\bar{l}_j(x_i + \mu h) = \prod_{r=0,\, r\neq j}^{k} \left(\frac{\mu + r - 1}{r - j} \right).$$

Now one can view the Adams Bashforth formula as,

$$\begin{aligned}
y_{i+1} &= y_i + \int_{x_i}^{x_{i+1}} \bar{P}_{i,k}(s; y'_i, y'_{i-1} \cdots y'_{i-k}) ds \\
&= y_i + \int_{x_i}^{x_{i+1}} \left(\sum_{j=0}^{k} \bar{l}_j(s) y'_{i-j} \right) ds \\
&= y_i + \sum_{j=0}^{k} \left(\int_{x_i}^{x_{i+1}} \bar{l}_j(s) ds \right) y'_{i-j} \\
&= y_i + h \sum_{j=0}^{k} \underbrace{\left(\int_0^1 \bar{l}_j(x_i + \mu h) d\mu \right)}_{\bar{\beta}_{j+1}} y'_{i-j},
\end{aligned}$$

and, if h is constant,

$$\bar{\beta}_{j+1} = \int_0^1 \prod_{r=0,\, r\neq j}^{k} \left(\frac{\mu + r - 1}{r - j}\right) d\mu,$$

which is independent of h and the x_i's.

Similarly, for the implicit Adams Moulton formula, let $P_{i,k}(s; y'_{i+1}, y'_i \cdots y'_{i-k})$ be the unique polynomial of degree $\leq k+1$ satisfying,

$$P_{i,k}(x_{i+1-j}; y'_{i+1}, y'_i \cdots y'_{i-k}) = y'_{i+1-j},\ j = 0, 1 \cdots (k+1).$$

Therefore,

$$\begin{aligned} y_{i+1} &= y_i + \int_{x_i}^{x_i+h} P_{i,k}(s; y'_{i+1}, y'_i \cdots y'_{i-k}) ds; \\ &= y_i + h \sum_{j=0}^{k} \underbrace{\left(\int_0^1 l_j(x_i + \mu h) d\mu\right)}_{\beta_j} y'_{i+1-j}, \end{aligned}$$

where

$$l_j(s) = \prod_{r=0,\, r\neq j}^{k+1} \left(\frac{s - x_{i+1-r}}{x_{i+1-j} - x_{i+1-r}}\right),$$

and, if h is constant this simplifies to,

$$\beta_j = \int_0^1 \prod_{r=0,\, r\neq j}^{k+1} \left(\frac{\mu + r - 1)}{r - j}\right) d\mu.$$

- An Efficient $P(EC)^M$ variable step Adams Formula can be implemented as:

$$\text{P:}\ \ y^0_{i+1} = y_i + \int_{x_i}^{x_i+h} \bar{P}_{i,k}(s; y'_i, y'_{i-1} \cdots y'_{i-k}) ds;$$

For $r = 0, 1 \cdots M$:

$$\begin{aligned} \text{E:}\ \ y'^{(r)}_{i+1} &= f(x_{i+1}, y^r_{i+1}), \\ \text{C:}\ \ y^{r+1}_{i+1} &= y_i + \int_{x_i}^{x_i+h} P_{i,k}(s; y'^{(r)}_{i+1}, y'_i \cdots y'_{i-k}) ds; \end{aligned}$$

(a) This iteration will converge ($r \to \infty$) if $\|h\frac{\partial f}{\partial y}\| < \frac{1}{|\beta_0|}$. (Note that this stepsize restriction, $h < \frac{1}{|\beta_0|\, \|\frac{\partial f}{\partial y}\|}$ can be avoided if Newtons method or some modification is used to solve the 'Corrector' equation.)

(b) A Newton basis or Lagrange basis can be used effectively for implementation.

(c) The difference between the predictor and corrector formulas satisfies,

$$y_{i+1}^{r+1} - y_{i+1}^{0} = h\beta_0(y_{i+1}^{'(r)} - y_{i+1}^{'(0)}).$$

- For Adams methods a natural definition for an accurate piecewise polynomial, $S(x)$, valid for all $x \in [a, b]$ is,

$$S(x) \equiv y_i + \int_{x_i}^{x} P_{i,k}(s; y'_{i+1}, y'_i \cdots y'_{i-k})ds \text{ for } x \in [x_i, x_{i+1}].$$

- Error estimates and local extrapolation:
  We have for $M > 1$,

$$\begin{aligned} z_i(x_i + h) - y_{i+1}^{0} &= O(h^{k+1}), \\ z_i(x_i + h) - y_{i+1}^{M} &= O(h^{k+2}). \end{aligned}$$

  Therefore $y_{i+1}^{M} - y_{i+1}^{0}$ is a valid estimate for the local error of y_{i+1}^{0} and we can can control the local error in y_{i+1}^{M} indirectly by keeping,

$$\|est_i\| = \|y_{i+1}^{M} - y_{i+1}^{0}\| < TOL.$$

  Virtually all variable step, variable order Adams methods are implemented this way. Note that there are several such methods available and they can differ on how they store and update the associated piecewise polynomials; whether or not they end on a function evaluation ($P(EC)^M$ or $P(EC)^M\ E$); and how often they consider a change in the stepsize or order.

2.6 Available Software for Nonstiff Problems:

In this section we give some advice on factors to consider when selecting a method for solving a non-stiff IVP. We also identify specific methods and software that are widely available and either in the public domain or supported in Scientific Computing Libraries.

In choosing the particular method to be used on a specific class of problems, factors to consider include:

- For efficiency over a wide range of accuracy requests (values of TOL), one should use a variable order method such as those based on Adams formulas or a carefully chosen fixed-order RK method.

- For a fixed accuracy RK methods will generally be the best choice if the derivatives are relatively inexpensive to evaluate (say less than 25 flops per component). If the derivatives are expensive to evaluate then a varialble order Adams method would be more efficient. This observation is due to the fact that the overhead in taking a step with a multistep method can be significantly greater than that for a one-step method.

- If cost does not increase much as TOL becomes more severe, one should suspect stiffness.
- If costs increase significantly as TOL becomes more severe, one should suspect that the problem may not be smooth and discontinuities may be present.

Relevant numerical methods and their availability:

1. Netlib is a website that maintains a library of state-of the art software for Scientific computing (www.netlib.org/index.html). The methods are in the public domain and the source code (mostly FORTRAN) can be downloaded from this site. The nonstiff IVP methods in this collection include:

   RKSUITE A suite of codes for solving IVPs in ODEs. A choice of RK methods is available. Includes an error assessment facility and a sophisticated stiffness checker. Template programs and example results provided. Supersedes RKF45, DDERKF, D02PAF.

   DVERK by Jackson, Hull, and Enright for ordinary differential equation initial-value problem solver with global error control. Alg: Verner's fifth and sixth order Runge-Kutta pair. Prec: double.

   ODE by Shampine and Gordon for ordinary differential equation initial-value problem solver. Alg: Adam's methods. Prec: double. Rel: excellent.

   RKF45 by Watts and Shampine for ordinary differential equation initial-value problem solver. Alg: Runge-Kutta Fehlberg fourth-fifth order. Prec: double. Rel: good.

   VODE by Brown, Byrne and Hindmarsh for non-stiff or stiff ordinary differential equation initial-value problem solver. Alg: backward differentiation formulae (variable coefficient formulae). Prec: double updated version of epsode

2. Effective IVP methods are also available and supported in the numerical libraries available from NAG, IMSL, HARWELL etc. These methods include:

   IVPRK IMSL

   DERKF SLATEC

   DEABM SLATEC

   IVPAG IMSL

3. A family of different order (up to order 8) Continuos Runge-Kutta methods (similar in form and structure to DVERK) are available locally.

2.7 Stiff ODEs:

Mathematical simulations of systems that have different components evolving on different time scales often lead to ODEs which exhibit numerical difficulties which require special attention and special methods. The associated class of IVPs are called stiff

IVPs and we will present an overview of the cause of the difficulty and software that is available for this class of problems.

1. When solving an IVP numerically there are three possible constraints on the stepsize:

   (a) The stability of the formula. For example for $y' = \lambda y$ one must have $h\lambda \in \mathcal{R}$, the region of absolute stability of the underlying formula.

   (b) The convergence of functional iteration (when an implicit formula is used). For example for a multistep method, one must have $h < \frac{1}{|\beta_0| \, \|\frac{\partial f}{\partial y}\|}$.

   (c) Control of the local error or defect introduced on each step. This usually has the form,

   $$|c| h^{p+1} < TOL\, h.$$

   Definition: AN ODE will be considered <u>Stiff</u> if the stepsize used during integration by <u>conventional</u> methods is determined by stability or convergence of an iteration, rather than by control of the local error.

   Note that by conventional methods we mean standard Adams multistep or standard explicit Runge-Kutta methods. This 'definition' implies that whether a problem is stiff or not will depend on TOL and the length of integration $(b - a)$ and a problem can be stiff for part of the integration.

   - Consider the model scalar IVP:

     $$y' = \lambda y, \quad y(x_0) = y_0, \quad \lambda \in \Re, \quad \lambda << 0.$$

     The solution is,

     $$y(x) = e^{\lambda(x-x_0)} y_0, \quad \text{on any interval } [x_0, x_F],$$

     and therefore $|y(x)| = e^{\lambda(x-x_0)} |y_0|$.

     Let $\hat{x} = x_0 + \ln(\frac{TOL}{|y_0|})/\lambda$ (ie. $|y(\hat{x})| = TOL$). For this model problem we then have,

(a) For $x \in [x_0, \hat{x}]$, $|y(x)| > TOL$ and we require and expect the numerical solution to remain accurate.

(b) For $x > \hat{x}$, $|y(x)| < TOL$, accuracy is no longer important as long as the numerical solution remains stable (ie. $|S(x)| < TOL$).

(c) $[x_0, \hat{x}]$ is the transient region and $[\hat{x}, x_F]$ is the steady state (or 'smooth') region.

- Consider applying Euler's formula to this problem (simplest one-step formula),

$$y_{i+1} = y_i + hy_i' = (1 + h\lambda)y_i,$$

When $|\lambda|$ is large, $h\lambda << -2$, $|1 + h\lambda| >> 1$ and,

$$|y_{i+1}| = |1 + h\lambda|^{i+1}|y_0|.$$

This leads to an unstable computation. To make this computation stable the stepsize would have to be severely restricted ($h < \frac{2}{|\lambda|}$) for all steps.

- Consider applying the Implicit Euler (Backward Euler) formula to this problem,

$$y_{i+1} = y_i + hy_{i+1}' = \frac{1}{(1 - h\lambda)}y_i,$$

When $|\lambda|$ is large, $h\lambda << -2$, $|\frac{1}{(1-h\lambda)}| << 1$ and,

$$|y_{i+1}| = |\frac{1}{(1 - h\lambda)}|^{i+1}|y_0|.$$

This leads to a very stable computation as the $|y_i| \rightarrow 0$ rapidly. That is, numerical stability is satisfied without imposing constraints on the stepsize.

2. **Recall the Notion of Fixed h stability:** Consider the model problem,

$$y' = \lambda y, \;\; y(x_0) = y_0, \;\; \text{where } \lambda \in C.$$

- A formula, when applied to this problem with constant stepsize h, is considered stable (for the particular values of h and λ) if $|y_i|$ remains bounded as $i \to \infty$ for any initial condition vector, y_0.
  (a) When an explicit, s-stage, Runge-Kutta formula is applied to this model problem, the numerical solution can be shown to satisfy,

  $$y_{i+1} = p_s(h\lambda)y_i \;\; = \;\; [p_s(h\lambda)]^{i+1} y_0,$$

  where $p_s(z)$ is a polynomial of degree $\leq s$ whose coefficients are determined by the parameters that define the Runge-Kutta formula. In this case $|y_i|$ is bounded iff $|p_s(h\lambda)| \leq 1$.

  (b) When a k-step LMF is applied to this model problem, the numerical solution satisfies,

  $$y_{i+1} = \sum_{j=1}^{k} c_j \xi_j^{i+1},$$

  where the c_j's depend on the starting values,$y_0, y_1 \cdots y_{k-1}$ and the ξ_j's are the roots of the associated characteristic polynomial, $\alpha(z) - h\lambda\beta(z)$. In this case $|y_i|$ is bounded iff $|\xi_j| \leq 1, \; j = 1, 2 \cdots k$. (Note that $\alpha(z)$ has all roots inside or on the unit circle and this implies as $h\lambda \to 0$, the formula will be stable.)

- **Definition**
  The Region of absolute stability associated with a formula is

  $$\mathcal{R} \equiv \{h\lambda : \text{when applied to } y' = \lambda y \text{ with constant } h, |y_i| \to 0 \text{ for any } y_0\}.$$

  Therefore for an explicit Runge-Kutta formula we have,

  $$\mathcal{R} = \{h\lambda : |p_s(h\lambda)| \leq 1\},$$

  and for a multistep formula we have,

  $$\mathcal{R} = \{h\lambda : \text{ all roots of } \; \alpha(z) - (h\lambda)\beta(z) \; \text{ are inside the unit circle}\}.$$

3. **An Example:** Consider,

$$y' = \begin{bmatrix} -1 & 0 \\ 0 & -100 \end{bmatrix} y, \;\; y(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

The solution to this IVP is,

$$y(x) = \begin{bmatrix} e^{-x} \\ e^{-100x} \end{bmatrix}.$$

- The second component is a transient, and is only significant in an initial interval as it decays rapidly to zero. The accuracy of this component is only of concern in this initial transient region after which only the stability of this component and the accuracy of the first component is required.
- Conventional methods will require that $|h100|$ remain small throughout the integration ($|h\lambda| < \frac{1}{|\beta_0|}$ or $h\lambda \in \mathcal{R}$).

4. **Lemma:** In general, for a system of ODEs, $y' = Ay$, numerical stability requires that $\forall \lambda_j \in \rho(A),\ h\lambda_j \in \mathcal{R}$.

   <u>Proof</u> Consider the case $A = SDS^{-1}$ where, D is a diagonal matrix and the underlying numerical formula is a multistep formula,

   $$y_{i+1} = \sum_{j=1}^{k} \alpha_j y_{i+1-j} + h \sum_{j=0}^{k} \beta_j y'_{i+1-j}.$$

   Let $z_i = Sy_i$, then $y_i \to 0$ iff $z_i \to 0$. Multiplying the multistep formula by S we obtain,

   $$\begin{aligned} z_{i+1} &= \sum_{j=1}^{k} \alpha_j z_{i+1-j} + h \sum_{j=0}^{k} \beta_j SA(S^{-1}S) y_{i+1-j}, \\ &= \sum_{j=1}^{k} \alpha_j z_{i+1-j} + h \sum_{j=0}^{k} \beta_j D z_{i+1-j} \end{aligned}$$

   Hence the r^{th} component of z_{i+1} satisfies:

   $$z_{i+1}^{(r)} = \sum_{j=1}^{k} \alpha_j z_{i+1-j}^{(r)} + h \sum_{j=0}^{k} \beta_j \lambda_r z_{i+1-j}^{(r)},$$

   which is the approximation we would have obtained by applying the multistep formula to the scalar problem,

   $$z' = \lambda_r z, \quad z(0) = z_0^{(r)}.$$

   Now $z_i \to 0$ iff each $z_i^{(r)} \to 0$ and this is true iff $h\lambda_r \in \mathcal{R}$ for $r = 1, 2 \cdots n$.

   For more general A and when the underlying formula is a one-step formula the proof is similar.

5. For the model problem, $y' = \lambda y$ we know that $|y(x_i)|$ remains bounded iff $Re(\lambda) \leq 0$. Therefore an ideal stability situation would be to develop formulas whose regions of absolute stability include the LHP.

   - Definition (Dahlquist): A formula is <u>A-Stable</u> if its region of absolute stability, $\mathcal{R}$ contains the LHP, $\{h\lambda : Re(h\lambda) < 0\}$. An example of an A-Stable formula is the implicit Euler formula.

- Dahlquist's Key Results: (A characterization of A-stability and a negative result regarding the existence of such formulas.)

  (a) **Lemma:** A k-step multistep formula is A-stable iff $\frac{\alpha(z)}{\beta(z)}$ is analytic and has nonegative real part for $|z| > 1$.
  Proof:

  i. To show $\frac{\alpha(z)}{\beta(z)}$ analytic we need only show $\beta(z) \neq 0$ for $|z| > 1$.
     - since the formula is A-stable we have that, for real μ, as $\mu \to -\infty$, the roots of $\alpha(z) - \mu\beta(z)$ are bounded in magnitude by 1.
     - But for each root z_r of $\beta(z)$, there is a root, $z_r(\mu)$ of $\alpha(z) - \mu\beta(z)$, such that $z_r(\mu) \to z_r$ as $\mu \to -\infty$. [This follows since the roots of $\alpha(z) - \mu\beta(z)$ are the roots of $\beta(z) - \frac{1}{\mu}\alpha(z)$ and the roots of a polynomial depend continuously on the coefficients.]
     - Therefore, since $|z_r(\mu)| \leq 1$ for all $\mu < 0$ we must have

     $$\lim_{\mu\to-\infty} z_r(\mu) = z_r,$$

     must satisfy $|z_r| \leq 1$ and $\frac{\alpha(z)}{\beta(z)}$ is analytic in $|z| > 1$.

  ii. Now assume A-stability and consider $\bar{z}$ such that $|\bar{z}| > 1$. Let $\mu = \frac{\alpha(\bar{z})}{\beta(\bar{z})}$. Then $\bar{z}$ is a root of $\alpha(z) - \mu\beta(z)$, and if $\mu \in LHP$, then $|\bar{z}|$ must be inside the unit circle. This is a contradiction ($|\bar{z}| > 1$) and hence $\mu \notin LHP$, (ie. $Re(\mu) \geq 0$).

  iii. For the converse, assume $\frac{\alpha(z)}{\beta(z)}$ is analytic for $|z| > 1$ and $Re\left(\frac{\alpha(z)}{\beta(z)}\right) \geq 0$ for $|z| > 1$. We then have, if $\mu = \frac{\alpha(z)}{\beta(z)}$, and $Re(\mu) < 0$ then $|z| \leq 1$. This implies $LHP \in \mathcal{R}$ and the formula is A-stable.

  (b) **Theorem:** An explicit k-step multistep formula cannot be A-stable.
  Proof: Explicit $\Rightarrow \beta(z) \sim az^{k-m}$ as $|z| \to \infty$ for $a \neq 0,\ m \geq 1$.
  Now $\alpha(z) \sim z^k$ as $|z| \to \infty$. Therefore $\frac{\alpha(z)}{\beta(z)} \sim \frac{1}{a}z^m$ as $|z| \to \infty$.
  But A-stability implies $Re(\frac{\alpha(z)}{\beta(z)}) \geq 0$ for $|z| > 1$ and this contradicts the observation that $\frac{\alpha(z)}{\beta(z)} \sim \frac{1}{a}z^m$ as $|z| \to \infty$ since

  i. m odd and μ real, $\mu \to \infty \Rightarrow$

  $$Re(\frac{1}{a}\mu^m) = -Re(\frac{1}{a}(-\mu)^m) \neq 0.$$

  ii. For m even, consider $\pm\mu\, i$ as $\mu \to \infty$.

  (c) **Examples** of A-stable formulas: Consider the multistep formula defined by,

  $$\begin{aligned}\alpha(z) &= z^k - 1,\\ \beta(z) &= \frac{1}{2}\, k(z^k + 1).\end{aligned}$$

This formula is A-stable and convergent for any $k \geq 1$. This follows since $\alpha(1) = 0$, and $\alpha'(1) = \beta(1)$ implies convergence and $\frac{\alpha(z)}{\beta(z)}$ is clearly analytic in $|z| > 1$ with,

$$\begin{aligned} Re\left[\frac{z^k - 1}{\frac{1}{2}k(z^k+1)}\right] &= Re\left[\frac{(z^k-1)(\bar{z}^k+1)}{\frac{1}{2}k(z^k+1)(\bar{z}^k+1)}\right] \\ &= \frac{1}{\frac{1}{2}k|z^k+1|^2}\, Re\left[(z^k-1)(\bar{z}^k+1)\right] \\ &= \frac{2}{k|z^k+1|^2}\, Re\left[|z^k|^2 - 1 + z^k - \bar{z}^k\right] \\ &= \frac{2}{k|z^k+1|^2}\, Re\left[|z^k|^2 - 1\right] > 0. \end{aligned}$$

The special case $k = 1$ corresponds to the trapeziodal rule,

$$y_{i+1} = y_i + \frac{h}{2}(y_i' + y_{i+1}').$$

(d) **Main Theorem:** (Dahlquist – BIT 1963) The maximum order of an A-stable Multistep formula cannot exceed 2. The smallest error constant of all the A-stable formulas, is $c_{p+1} = 1/12$ and is obtained for the trapezoidal rule.
The proof of this result is a nice example of the application of theoretical results from Complex Variables.

- Is A-stability too strong a requirement of a formula? Consider the case where an eigenvalue of the system $y' = Ay$ is large in magnitude and 'near' the imaginary axis.
  (a) For $\lambda = \mu + i\omega$ where $\mu < 0$ and $|\omega| >> |\mu|$, we have $e^{\lambda(x-x_0)}$ is slowly damped and highly oscillatory.
  (b) Intuitively we expect the stepsize to be be kept small to follow the oscillations $h|\omega|$ small implies $h|\lambda|$ also small.
  (c) That is, when the eigenvalues are near the imaginary axis, the numerical accuracy corresponding to these components will likely determine the stepsize (and not the stability of these components).
- To derive multistep formulas of order higher than two that were suitable for Stiff equations, Widlund and Gear relaxed the requirement of A-stability:
  - **Definition:** (Widlund – 1967) A formula is A(α)-Stable if its stability region contains the wedge-shaped region, S(α),

$$S(\alpha) = \{z : \arg(-z) < \alpha \leq \pi/2\}.$$

Clearly a formula is A-Stable iff it is A($\pi/2$)-Stable.

Widlund derived 3^{rd} and 4^{th} order multistep formulas that are $A(\alpha)$-Stable for $\alpha < \pi/2$. The coefficients of these formulas must become unbounded as α approaches $\pi/2$. For $\alpha > 0$, a formula (multistep or one-step) can be A(α)-Stable only if it is implicit.

- **Definition:** (Gear 1967) A formula is Stiffly-Stable if its region of absolute stability contains a region of the form $R_1 \cup R_2$ where,

$$R_1 = \{z : Re(z) \leq D \ < 0\},$$

$$R_2 = \{z : D < Re(z) < 0, -\theta < Im(z) < \theta\}.$$

  Gear proposed the backward differentiation multistep formulas (BDF) for solving stiff problems as he showed they were stiffly stable for orders up to six.

- An additional stability requirement that is desirable when solving stiff problems. Consider λ real $\lambda << 0$ and x outside the transient region. The corresponding numerical solution, y_i satisfies $|y_i| < TOL$ and, with the A-Stable Trapezoidal rule, we should be able to take large stepsizes. Consider using $|h\lambda| \approx 100$, then we have,

$$y_{i+1} = \left(\frac{1 + h\lambda/2}{1 - h\lambda/2}\right) y_i \approx -\left(\frac{49}{50}\right) y_i.$$

  This means that the numerical solution will be slowly damped and oscillatory. This will affect the error estimate and artificially restrict the stepsize. The difficulty is that the trapezoidal rule exhibits little or no damping for large $|h\lambda|$.

  Definition: A formula will be L-Stable (or Stable at ∞) if there exists $W < 0$ such that.

$$\sup_{h\lambda < W} |\frac{y_{i+1}}{y_i}| = C < 1,$$

  where y_i is the sequence generated by solving $y' = \lambda y$ with a constant stepsize h.
  Note that the trapezoidal rule is not L-stable but the backward Euler formula is.

2.8 Suitable Methods for Stiff IVPs:

Multistep Methods – BDF formulas (Gear 1967)

Recall that for the k-step LMF,

$$y_{i+1} = \sum_{j=1}^{k} \alpha_j y_{i+1-j} + h \sum_{j=0}^{k} \beta_j y'_{i+1-j},$$

we have,

$$\mathcal{R} = \{\mu : \alpha(z) - \mu\beta(z) \text{ has all its roots inside UC }\}.$$

For a given LMF, let $p_\mu(z) \equiv \alpha(z) - \mu\beta(z)$.

Note that:

- To be suitable for stiff equations, we must have $\mu \in \mathcal{R}$ for $|\mu|$ large and μ negative real. This implies $\beta_0 \neq 0$ (otherwise $p_\mu(z)$ is monic for all μ and, as the coefficients of $p_\mu(z)$ are symmetric functions of the roots, at least one root must be unbounded as $\mu \to -\infty$).
- For $\bar{\mu}$ on the boundary of $\mathcal{R}$ one of the roots of $p_{\bar{\mu}}(z)$ is on the UC and therefore,
$$\bar{\mu} = \frac{\alpha(\bar{z})}{\beta(\bar{z})} \text{ for } \bar{z} = e^{i\theta}, \text{ for some } 0 \leq \theta \leq 2\pi.$$
Hence the mapping $\frac{\alpha(e^{i\theta})}{\beta(e^{i\theta})}$ for $\theta \in [0, 2\pi]$ must trace out the boundary of $\mathcal{R}$.
- The LMF will be L-Stable (or stable at ∞) iff all the roots of $\beta(z)$ are strictly inside the UC.

1. Gear's choice of coefficients for a class of multistep methods suitable for stiff equations was based on setting $\beta(z) = \beta_0 z^k$, and then choosing the parameters, $\alpha_1, \alpha_2 \cdots \alpha_k, \beta_0$ to obtain order k (and hoping that the resulting $\alpha(z)$ was stable).

   Note:

   - For each value of k this k-step <u>BDF</u> formula is determined by solving a linear sytem of equations.
   - Gear showed that these formulas were convergent for $k \leq 6$ and members of a family of multistep formulas which had already been investigated and analysed in earlier publications (although they had not been identified as being suitable for stiff equations).
   - For $k > 6$ these formulas are not stable (ie., $\alpha(z)$ has at least one root outside the UC).
   - The formulas are stiffly stable for $k \leq 6$ (but for $k = 6$ the stability region is very restrictive).

2. The BDF are then of the form,
$$y_{i+1} = \sum_{j=1}^{k} \alpha_j y_{i+1-j} + h\beta_0 y'_{i+1},$$
or equivalently,
$$y'_{i+1} = \frac{1}{h}\left(\frac{1}{\beta_0} y_{i+1} - \sum_{j=1}^{k}\left(\frac{\alpha_j}{\beta_0}\right) y_{i+1-j}\right).$$
That is, these multistep formulas can be interpreted as expressing an accurate approximation to y'_{i+1} in terms of 'past' values of y. It is this observation that allows us to derive a variable stepsize implementation of the BDF formulas.

To generalize BDF to variable stepsize, consider the Lagrange form of the polynomial interpolating $y(x)$ at $x_{i+1}, x_i \cdots x_{i+1-k}$,

$$P_{i,k}(x) = \sum_{j=0}^{k} \ell_j(x) y_{i+1-j},$$

where

$$\ell_j(x) = \frac{\prod_{r=0,r\neq j}^{k}(x - x_{i+1-r})}{\prod_{r=0,r\neq j}^{k}(x_{i+1-j} - x_{i+1-r})}.$$

Then the corresponding variable stepsize BDF is,

$$y'_{i+1} = P'_{i,k}(x_{i+1}) = \sum_{j=0}^{k} \ell'_j(x_{i+1}) y_{i+1-j},$$

or equivalently,

$$y_{i+1} = \sum_{j=1}^{k} \left(\frac{-\ell'_j(x_{i+1})}{\ell'_0(x_{i+1})} \right) y_{i+1-j} + \frac{1}{\ell'_0(x_{i+1})} y'_{i+1},$$

One can verify that, for fixed h,

$$\ell'_0(x_{i+1}) = \frac{1}{h\beta_0} \text{ and } \ell'_j(x_{i+1}) = \left(\frac{-\alpha_j}{h\beta_0} \right) \text{ for } j = 1, 2 \cdots k,$$

where $\beta_0, \alpha_1, \alpha_2 \cdots \alpha_k$ are the usual 'constant h' BDF coefficients derived by solving the linear equations associated with the order k expansion of the LTE.

3. The Corrector iteration for BDF:
   - Standard $P - C$ iteration (or functional iteration):
     P: predict $y_{i+1}^{(0)} = P_{i-1,k}(x_{i+1}), \;\; y_{i+1}^{\prime(0)} = P'_{i-1,k}(x_{i+1}),$
     E: evaluate $y_{i+1}^{\prime(l)} = f(x_{i+1}, y_{i+1}^{(l-1)}),$
     C: correct $y_{i+1}^{(l)} = y_{i+1}^{(l-1)} + \frac{1}{\ell'_0(x_{i+1})}[y_{i+1}^{\prime(l)} - y_{i+1}^{\prime(l-1)}]$
     for $l = 1, 2 \cdots M$.

     Clearly this iteration converges if

     $$\frac{1}{|\ell'_0(x_{i+1})|} \| \frac{\partial f}{\partial y} \| \leq 1,$$

     and this would be a severe stepsize restriction for problems with transients such as $y' = Ay$ where we would like to have $h\|A\| >> 1$ in the smooth region.
   - Newton corrector iteration:
     let the nonlinear equation to be solved on each time step be,

     $$F(y_{i+1}) \equiv y_{i+1} - \frac{1}{\ell'_0(x_{i+1})} f(x_{i+1}, y_{i+1}) - \sum_{j=1}^{k} \alpha_j y_{i+1-j}.$$

To solve $F(y_{i+1}) = 0$, we predict as before, $y_{i+1}^{(0)} = P_{i-1,k}(x_{i+1})$ and iterate using Newton's Method,

$$y_{i+1}^{(l)} = y_{i+1}^{(l-1)} - \left[F'(y_{i+1}^{(l-1)})\right]^{-1} F(y_{i+1}^{(l-1)}) \text{ for } l = 1, 2 \cdots M.$$

Or, if the 'Newton Correction' $\delta_{i+1}^{(l)} \equiv y_{i+1}^{(l)} - y_{i+1}^{(l-1)}$, then

$$W_i \delta_{i+1}^{(l)} = -F(y_{i+1}^{(l-1)}),$$

where

$$W_i = F'(y_{i+1}^{(l-1)}) = \left[I - \frac{1}{\ell_0'(x_{i+1})} \frac{\partial f}{\partial y}|_{y_{i+1}^{(l-1)}}\right].$$

In practice refactoring W_i on each iteration to solve this linear system is too expensive and a <u>modified</u> Newton iteration is used. That is, a linear system similar to $W_i \delta_{i+1} = -F(y_{i+1}^{(l-1)})$ is solved on each iteration except that W_i is approximated by $\tilde{W}_i$ where,

$$\tilde{W}_i \approx \left[I - \frac{1}{\ell_0'(x_{i+1})} \frac{\partial f}{\partial y}|_{y_{i+1}^{(l-1)}}\right].$$

This can be interpreted as a preconditioning strategy to solve the sequence of linear systems associated with Newtons method. For example one could use a W_i that is defined in terms of an outdated Jacobian,

$$\tilde{W}_i = \left[I - \frac{1}{\ell_0'(x_{i+1})} \frac{\partial f}{\partial y}|_{y_{i-r}}\right], \text{ for some } r > 0.$$

In such cases $\tilde{W}_i$ is only recalculated (and refactored) when:

- The order changes (β_0).
- The stepsize changes ($h\beta_0 = \frac{1}{\ell_0'(x_{i+1})}$).
- The iteration is not converging rapidly enough (for example if the magnitude of the Newton correction is not small after 3 iterations).

4. There are several numerical methods available based on the BDF formulas that are widely used and supported in various libraries and programming environments. They often differ in how they implement and handle the solution of the linear equations that arise in the modified newton iteration. This is often the most critical component of the implementation (in terms of computer time and storage requirement).

Runge-Kutta methods suitable for stiff ODEs :

Recall an s-stage Runge-Kutta formula is characterized by,

$$\begin{array}{c|cccc}
\alpha_1 & \beta_{11} & \beta_{12} & \ldots & \beta_{1s} \\
\alpha_2 & \beta_{21} & \beta_{22} & \ldots & \beta_{2s} \\
\vdots & \vdots & \vdots & & \vdots \\
\alpha_s & \beta_{s1} & \beta_{s2} & \ldots & \beta_{ss} \\
\hline
 & \omega_1 & \omega_2 & \ldots & \omega_s
\end{array}$$

In this section we will denote the $2s + s^2$ parameters that define this tableau by $\underline{\alpha}, \underline{\omega}$ and $A = (\beta_{i,j})$.

1. **Lemma 1:** All explicit Runge-Kutta formulas, when applied to $y' = \lambda y$ yield,

$$y_{i+1} = p_s(h\lambda)y_i,$$

where $p_s(z)$ is a polynomial of degree $\leq s$.
<u>Proof</u>
(outline) First, it is straightforward to show by induction that $k_r = \lambda q_{r-1}(h\lambda)y_i$, where $q_{r-1}(z)$ is a polynomial of degree $\leq r - 1$.
We then have,

$$\begin{aligned}
y_{i+1} &= y_i + h\sum_{j=1}^{s}\omega_j k_j \\
&= y_i + h\lambda\sum_{j=1}^{s}\omega_j q_{j-1}(h\lambda)y_i \\
&= \left[1 + \sum_{j=1}^{s}(h\lambda)\omega_j q_{j-1}(h\lambda)\right] y_i \\
&\equiv p_s(h\lambda)y_i
\end{aligned}$$

2. The stability region for an explicit Runge-Kutta formula is then seen to be,

$$\mathcal{R} = \{h\lambda : |p_s(h\lambda)| < 1\}.$$

This cannot be suitable for stiff problems since it contains only a small bounded subset of the LHP. Also note that if this formula is order p then $p_s(h\lambda) - e^{h\lambda} = O(h^{p+1})$ and this implies that the first p coefficients of $p_s(z)$ are fixed.

3. **Lemma 2:** All implicit Runge-Kutta formulas, when applied to $y' = \lambda y$, yield,

$$y_{i+1} = \left[\frac{p_s(h\lambda)}{q_s(h\lambda)}\right] y_i,$$

where $p_s(z)$ and $q_s(z)$ are polynomials of degree $\leq s$.

Proof: On each step, the equations defining the stages, $k_1, k_2 \cdots k_s$ are,

$$\begin{aligned} k_j &= f(x_i + \alpha_j h, y_i + h \sum_{r=1}^{s} \beta_{j,r} k_r) \\ &= \lambda y_i + (h\lambda)\beta_{j,1} k_1 + (h\lambda)\beta_{j,2} k_2 \cdots + (h\lambda)\beta_{j,s} k_s \end{aligned}$$

This is a linear equation in the unkowns. In matrix form this system of s equations in s unknowns becomes,

$$\left[\; I - (h\lambda)A \; \right] \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_s \end{bmatrix} = \lambda \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} y_i,$$

where I is the $s \times s$ identity matrix and $A = (\beta_{i,j})$. Now to solve this system by Cramers rule, we observe,

$$[I - (h\lambda)A]^{-1} = \frac{adj(I - (h\lambda)A)}{det(I - (h\lambda)A)},$$

where the $(i,j)^{th}$ element of $adj(B)$ is the $(j,i)^{th}$ cofactor of B ($= det$ of the $(s-1) \times (s-1)$ matrix obtained from B by deleting the i^{th} column and j^{th} row.

But the elements of the matrix $I - (h\lambda)A$ are linear in $(h\lambda)$. Hence $det(I - (h\lambda)A)$ is a polynomial (in $(h\lambda)$) of degree $\leq s$ and the elements of $adj(I - (h\lambda)A)$ are polynomials (in $(h\lambda)$), of degree $\leq s - 1$.
We then have,

$$\begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_s \end{bmatrix} = \lambda \, [I - (h\lambda)A]^{-1} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} y_i,$$

and therefore,

$$\begin{aligned} k_j &= \lambda \left[\sum_{r=1}^{s} ([I - (h\lambda)A]^{-1})_{j,r} \right] y_i \\ &= \frac{\lambda}{det(I - (h\lambda)A)} \left[\sum_{r=1}^{s} [adj(I - (h\lambda)A]_{j,r} \right] y_i \\ &= \frac{\lambda \hat{p}_{j,s-1}(h\lambda)}{q_s(h\lambda)} y_i, \end{aligned}$$

where $\hat{p}_{j,s-1}(z)$ is a polynomial of degree $\leq s - 1$ and $q_s(z) = det(I - zA)$ is a polynomial of degree $\leq s$.

We then have,

$$\begin{aligned}
y_{i+1} &= y_i + h\sum_{j=1}^{s}\omega_j k_j \\
&= y_i + \frac{(h\lambda)}{q_s(h\lambda)}\sum_{j=1}^{s}\omega_j \hat{p}_{j,s-1}(h\lambda) y_i \\
&= \frac{1}{q_s(h\lambda)}\left[q_s(h\lambda) + \sum_{j=1}^{s}(h\lambda)\omega_j \hat{p}_{j,s-1}(h\lambda)\right] y_i \\
&\equiv \frac{p_s(h\lambda)}{q_s(h\lambda)} y_i.
\end{aligned}$$

Note that, using the above expression for the $k_j's$, an alternative expression for y_{i+1} is,

$$\begin{aligned}
y_{i+1} &= y_i + h\underline{\omega}^T \underline{k}, \\
&= \left[1 + (h\lambda)\underline{\omega}^T [I - (h\lambda)A]^{-1}\underline{e}\right] y_i,
\end{aligned}$$

where $\underline{e} = [1, 1 \cdots 1]^T$. Hence we observe that the rational function that is associated with the stability of a Runge-Kutta formula satisfies,

$$\frac{p_s(z)}{q_s(z)} = [1 + z\underline{\omega}^T [I - zA]^{-1}\underline{e}].$$

4. What are the most suitable implicit Runge-Kutta formulas for stiff IVPs? There are two standard approaches to provide insight and guidance in addressing this question/issue. One is based on maximizing the order of the method (for a given value of s and associated stability constraint). The second is based on first restricting the class of formulas to be considered (to those that can be efficiently implemented) and then , within this restricted class, maximizing the obtainable order.

5. **Padé - approach** (maximize the order)

   (a) for $y' = \lambda y$ we know $z_i(x_{i+1}) = e^{h\lambda} y_i$. and therefore the local error satisfies,

   $$y_{i+1} - z_i(x_{i+1}) = \left[\frac{p_s(h\lambda)}{q_s(h\lambda)} - e^{h\lambda}\right] y_i$$

   and the formula will be of order p only if,

   $$\left(\frac{p_s(z)}{q_s(z)} - e^z\right) = O(z^{p+1}) \text{ as } z \to 0$$

   This observation leads to a topic in approximation theory and the subject of Padé approximations.

(b) Consider the power series expansion,

$$f(z) = \sum_{r=0}^{\infty} f^{(r)}(0)\frac{z^r}{r!}.$$

The degree k polynomial approximation to $f(z)$ of maximum order ($z \to 0$) is,

$$P_k(z) = \sum_{r=0}^{k} f^{(r)}(0)\frac{z^r}{r!},$$

and for sufficiently differentiable $f(z)$ we have

$$P_k(z) - f(z) = O(z^{k+1}) \text{ as } z \to 0.$$

Given a class of rational functions,

$$R_{n,m} \equiv \left\{ \frac{\sum_{r=0}^{n} a_r z^r}{1 + \sum_{r=1}^{m} b_r z^r} : (a_0, a_1 \cdots a_n, b_1, \cdots b_m) \in R^{n+m+1} \right\},$$

then the Padé approximation, $P_{n,m}(z)$ to $f(z)$ is the rational function in $R_{n,m}$ of maximum order. That is, it is the unique rational function in this class such that,

$$P_{n,m}(z) - f(z) = O(z^{n+m+1}).$$

The coefficients (the $a_r's$ and the $b_r's$, defining $P_{n,m}$) can be determined by replacing $f(z)$ by its power series in this expression, multiplying both sides by $(1 + \sum_{r=1}^{m} b_r z^r)$, and equating powers of h to as high a power as possible by solving the resulting nonsingular linear system of equations.

(c) Consider the following examples:

- A 1 stage, 2^{nd} order IRK (the midpoint formula),

$$\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}$$

$$y_{i+1} = y_i + hk_1, \quad k_1 = f(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1).$$

Applying this to the model problem $y' = \lambda y$,

$$y_{i+1} = \left(\frac{1 + (\frac{h\lambda}{2})}{1 - (\frac{h\lambda}{2})} \right) y_i = P_{1,1}(h\lambda) y_i.$$

where $P_{1,1}(z)$ is the first 'diagonal' ($m = n = 1$) Padè approximation to e^z. (It is straightforward to verify that this formula is A-stable.) Note that the rational function, $P_{1,1}(z)$, is the same as that associated with the stability of the Trapezoidal formula,

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}$$

- A 2 stage, 4^{th} order IRK formula.

$$\begin{array}{c|cc} \frac{1}{2}-\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\ \frac{1}{2}+\frac{\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

For $y' = \lambda y$ this yields,

$$y_{i+1} = \left[\frac{1 + \frac{h\lambda}{2} + \frac{(h\lambda)^2}{12}}{1 - \frac{h\lambda}{2} + \frac{(h\lambda)^2}{12}} \right] y_i = P_{2,2}(h\lambda) y_i.$$

(d) Butcher (1964) has shown that in general there exists an s stage IRK formula of maximum order $2s$ which, when applied to $y' = \lambda y$ yields,

$$y_{i+1} = P_{s,s}(h\lambda) y_i,$$

where $P_{s,s}(z)$ is the s diagonal Padé approximation to e^z. These formulas are based on Gaussian quadrature and weights . They are known to be A-stable (but not L-stable). This latter observation follows from the fact that

$$\lim_{z \to -\infty} P_{s,s}(z) = (-1)^s.$$

(e) To achieve high order and L-stability, consider an s stage, IRK formula with $k_s = f(x_{i+1}, y_{i+1})$, This implies $\alpha_s = 1$ and $\beta_{j,s} = \omega_j$ for $j = 1, 2 \cdots s$. Now recall that,

$$y_{i+1} = \left[\frac{p_s(h\lambda)}{q_s(h\lambda)} \right] y_i,$$

and

$$k_s = \lambda \left[\frac{\hat{p}_{s,s-1}(h\lambda)}{q_s(h\lambda)} \right] y_i \quad = \quad \lambda \left[\frac{p_s(h\lambda)}{q_s(h\lambda)} \right] y_i.$$

This implies the degree of $p_s(z) \leq s - 1$ and therefore, if the degree of $q_s(z) = s$, then we have L-stability. Note that the degree of $q_s(z)$ is equal to the degree of $[det(I - zA)] = rank(A)$.

Ehle and Axelson have shown that there exist s stage implicit Runge-Kutta formulas of this type, of order $2s - 1$, that are A-stable and correspond, when applied to $y' = \lambda y$, to

$$y_{i+1} = P_{s-1,s}(h\lambda) y_i,$$

where $P_{s-1,s}(z)$ is the s sub-diagonal Padè approximation to e^z. These formulas and weights are based on Radeau Quadrature formulas.

(f) A disadvantage of Padé IRK methods is that solving for the k_r's by a modified Newton's method (as is necessary for stiff problems) involves a fully coupled nonlinear system which requires $O(sn)^3$ operations on each step,

$$[I - h(A \otimes J)]\,(\delta\underline{k}) = RHS \;\; \text{(the residuals for the } k's\text{)}.$$

or more precisely,

$$\begin{bmatrix} (I - h\beta_{11}J) & -h\beta_{12}J & \cdots & -h\beta_{1s}J \\ -h\beta_{21}J & (I - h\beta_{22}J) & \cdots & -h\beta_{2s}J \\ \vdots & \vdots & \cdots & \vdots \\ -h\beta_{s1}J & -h\beta_{s2}J & \cdots & (I - h\beta_{ss}J) \end{bmatrix} \begin{bmatrix} \delta k_1 \\ \delta k_2 \\ \vdots \\ \delta k_s \end{bmatrix} = RHS,$$

It is also worth noting that with these formulas there is not a convenient error estimate available and one usually must resort to a 'step halving' strategy.

6. **Implementation-Oriented Approach:** (Reducing the costs associated with the nonlinear systems that must be solved)

(a) Consider a semi-implicit Runge-Kutta formula ($\beta_{j,r} = 0$ for $r > j$) where each

$$k_j = f(x_i + \alpha_j h, y_i + h\sum_{r=1}^{j} \beta_{j,r} k_r),$$

is solved sequentially ($j = 1, 2 \cdots s$). If a modified Newton iteration is used for each system, we must solve,

$$W_j \delta_j^{(l)} = k_j^{(l-1)} - f(x_i + \alpha_j h, y_i + h\sum_{r=1}^{j-1} \beta_{j,r} k_r + h\beta_{j,j} k_j^{(l-1)}),$$

where

$$W_j \approx [I - h\beta_{j,j}\frac{\partial f}{\partial y}].$$

Are there any semi-implicit formulas suitable for stiff equations and with equal values of $\beta_{j,j}$? Note that, for these formulas we would have $q_s(z) = det(I - zA) = \prod_{r=1}^{s}(1 - z\beta_{j,j})$.
Norsett and Wolfbrandt have investigated rational approximations to e^z, where the denominator has only real roots and shown that the maximum order is $s+1$ and with the further restriction that the $\beta_{j,j}$ be equal we can still obtain order $s+1$ (in which case $q_s(z) = (1 - z\beta)^s$).

These theoretical results are relevant to the approximation of e^z by a rational function and this allows us to infer an upper bound on the order of a semi-implicit Runge-Kutta formula. We still may not be able to find such formula which achieve this order.

(b) Alexander (SINUM 1978) has defined a diagonally implicit RK (DIRK) formula to be a semi-implicit Runge-Kutta formula with constant diagonal ($\beta_{j,j}$). Norsett, Alexander and Crouzeix have shown the following results which are relevant to this potentially effective class of methods:

- Norsett showed that the maximum order for a DIRK formula is $s+1$ for $s = 1, 2, 3, 5, 7, 9$ and s for $s = 4, 6, 8, 10$.
- Crouziex (1975) derived a 2 stage, 3^{rd} order, A-Stable DIRK and a 3 stage 4^{th} order, A-Stable DIRK.
- Alexander has shown that demanding L-Stability costs an order. He derived the following 2 stage, 2^{nd} order, A-Stable formula that is also L-stable for $\gamma = 1 \pm \frac{\sqrt{2}}{2}$,

$$\begin{array}{c|cc} \gamma & \gamma & \\ 1 & 1-\gamma & \gamma \\ \hline & 1-\gamma & \gamma \end{array}$$

  He also showed there are no 4^{th} order 4 stage DIRKs that are both A-stable and L-stable.
- DIRK formulas also suffer from the lack of a convenient local error estimate.

(c) Butcher and Burrage have considered a more general class of singly implicit Runge-Kutta (SIRK) formulas. These formulas are more expensive to implement (as there is more work per step) but higher order is possible for an s stage formula suitable for stiff equations.

- For a general implicit Runge-Kutta formula using a modified Newton iteration, we have to solve on each step,

$$[I - h(A \otimes J)]\,(\delta \underline{k}) = RHS \;\; \text{(the residuals for the } k's).$$

  If we use a similarity transformation, $T^{-1}AT = D$ then we have (using the properties of the Kronecker product),

$$\begin{aligned} [I - h(A \otimes J)] &= \left[I - h(T \otimes I)(D \otimes J)(T^{-1} \otimes I)\right] \\ &= (T \otimes I)[I - h(D \otimes J)](T^{-1} \otimes I). \end{aligned}$$

  Now if we use the transformed coordinate system $\bar{y} = (T^{-1} \otimes I)y$, for any vector $y \in R^n$, then the iteration in the tranformed space becomes,

$$[I - h(D \otimes J)]\,(\delta \bar{\underline{k}}) = \overline{RHS}.$$

- Butcher and Burrage called such formulas singly-implicit and suggested a particular one-parameter class of IRK formulas of this type based on Laguerre polynomials (from approximation theory) for which

T is determined explicitly and,

$$T^{-1}AT = D = \begin{bmatrix} \mu & 0 & \cdots & 0 \\ -\mu & \mu & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \mu \end{bmatrix} \quad (\mu \text{ is the parameter}).$$

Hence the linear system to be solved in the transformed coordinate system will be block bi-diagonal and lower triangular with constant diagonal blocks,

$$[I - h(D \otimes J)] = \begin{bmatrix} I - h\mu J & 0 & \cdots & 0 & 0 \\ h\mu J & I - h\mu J & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & h\mu J & I - h\mu J \end{bmatrix}.$$

- Note that, for this class of formulas,

$$q_s(z) = det(I - zA) = det(I - zD) = (1 - z\mu)^s.$$

By a careful choice of μ they derived :

i. An s stage implicit RK formula of order $s + 1$ for any s.

ii. An s stage, A-stable implicit RK formula of order $s + 1$ for $s = 1, 2, 3, 5$ (not 4 or 6).

iii. An s stage, A-stable, L-stable implicit RK formula of order s for $s = 1, 2, 3, 4, 5, 6, 8$ (not 7).

- They were able to develop computeable estimates of the local error for formulas in this class by showing how to embed the above one-parameter, s stage formula in an $(s + 1)$ stage formula pair of the form,

$$\begin{array}{c|cccc|c} \alpha_1 & \beta_{11} & \beta_{12} & \dots & \beta_{1s} & 0 \\ \alpha_2 & \beta_{21} & \beta_{22} & \dots & \beta_{2s} & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \alpha_s & \beta_{s1} & \beta_{s2} & \dots & \beta_{ss} & 0 \\ \alpha_{s+1} & \beta_{s+1,1} & \beta_{s+1,2} & \dots & \beta_{s+1,s} & \mu \\ \hline & \omega_1 & \omega_2 & \dots & \omega_s & 0 \\ & \hat{\omega}_1 & \hat{\omega}_2 & \dots & \hat{\omega}_s & \hat{\omega}_{s+1} \end{array}$$

In doing this they have identified choices for μ and the remaining parameters $(\beta_{s+1,1}, \beta_{s+1,2}, \cdots \beta_{s+1,1}, \hat{\omega}_1, \hat{\omega}_2, \cdots \hat{\omega}_{s+1})$ to yield an s stage order $(s, s+2)$ formula pair with the lower order formula A-stable for $s \leq 6$ and an s stage order $(s, s+1)$ formula pair with the lower order formula A-stable and L-stable for $s \leq 6$.
These formulas have been implemented by Butcher, Burrage and Chipman in the method, STRIDE.

Other Methods and Approaches for Stiff problems:

1. Composite Multistep or Block methods (STINT).
2. Extrapolation methods (METAN1).
3. Second Derivative Multistep methods (SECDER) – uses $y^{''} = f_x + f_y f$ which is inexpensive and available for autonomous ODEs.
4. Rosenbrock Runge-Kutta methods.
5. Blended Multistep methods – uses $(1+\omega)y_{i+1} = y_{i+1}^{A} + \omega y_{i+1}^{BDF}$.

Software for Stiff Problems:

From Netlib: EPSODE, VODE (VODEPK).

From Slatec: DEBDF.

From IMSL: IVPAG.

From Hairer and Wanner (1996) :
(http://www.unige.ch/math/folks/hairer/software.html), RADAU5, RADAU, SDIRK, ROSE4, SODEX.

See also: (http://pitagora.dm.uniba.it/ testset/).

From other Sources: STRUT (Krogh and Stewart, 1982). STRIDE, SECDER.

From MATLAB: ode15s.

2.9 Quantifying the Sensitivity of the Solution

In many applications, one is not only interested in the accurate and reliable approximation of the solution, but also in investigating the sensitivity of the solution to small changes in the problem specification. Consider an IVP that depends on a vector of fixed parameters, $\lambda \in R^m$,

$$y' = f(x, y, \lambda), \;\; y(a) = y_0.$$

One may be interested in the "sensitivity" of the solution, $y(x, \lambda)$ to perturbations in the initial conditions or to perturbations of the parameters. To investigate the sensitivity with respect to the initial conditions, let $Y(x, \lambda) \equiv \frac{\partial}{\partial y_0}(y(x, \lambda))$. We then have that $Y(x, \lambda)$ is an $n \times n$ matrix-valued function which satisfies the ODE,

$$\begin{aligned}
\frac{d}{dt}Y(x,\lambda) &= \frac{d}{dt}\frac{\partial}{\partial y_0}(y(x,\lambda)) \\
&= \frac{\partial}{\partial y_0}\frac{d}{dt}(y(x,\lambda)) \\
&= \frac{\partial}{\partial y_0} f(x, y(x,\lambda), \lambda) \\
&= \frac{\partial f}{\partial y}\frac{\partial y(x,\lambda)}{\partial y_0} \\
&= \frac{\partial f}{\partial y} Y(x,\lambda),
\end{aligned}$$

with associated initial conditions,

$$Y(a, \lambda) = \frac{\partial}{\partial y_0}(y(a, \lambda)) = \frac{\partial}{\partial y_0}(y_0) = I.$$

(Note that we use $\frac{\partial f}{\partial y}$, without arguments, to stand for $\frac{\partial f}{\partial y}(x, y(x, \lambda), \lambda)$.) Thus we have that $Y(x, \lambda)$ satisfies a linear homogeneous matrix IVP for $x \in [a, b]$ and its solution can be approximated by any initial value method.

Similarly to investigate the sensitivity of $y(x, \lambda)$ to perturbations in λ let,

$$Y^{\lambda}(x, \lambda) \equiv \frac{\partial}{\partial \lambda}(y(x, \lambda)).$$

Thus $Y^{\lambda}(x, \lambda)$ is an $n \times m$ matrix which satisfies the inhomogeneous linear ODE,

$$\begin{aligned}
\frac{d}{dt}(Y^{\lambda}(x, \lambda)) &= \frac{d}{dt}(\frac{\partial y(x, \lambda)}{\partial \lambda}) \\
&= \frac{\partial}{\partial \lambda}(\frac{dy(x, \lambda)}{dt}) \\
&= \frac{\partial}{\partial \lambda}(f(x, y(x, \lambda), \lambda)) \\
&= \frac{\partial f}{\partial y}\frac{\partial y(x, \lambda)}{\partial \lambda} + \frac{\partial f}{\partial \lambda} \\
&= \frac{\partial f}{\partial y}Y^{\lambda}(x, \lambda) + \frac{\partial f}{\partial \lambda},
\end{aligned}$$

with initial conditions,

$$Y^{\lambda}(a, \lambda) = \frac{\partial}{\partial \lambda}(y_0) = 0.$$

One can therefore determine an approximation to these sensitivities by approximating the solution of this matrix IVP by any reliable initial value method.

2.10 Differential/Algebraic Equations

In some applications ODEs arise where some of the defining equations have a small parameter multiplying the highest derivative of the system components. For example let $y(x) = (z(x)\ u(x))^T$, with $y'(x)$ defined by the ODE,

$$\begin{aligned}
z' &= A_{11}z + A_{12}u + f_1 \\
\epsilon u' &= A_{21}z + A_{22}u + f_2
\end{aligned}$$

where $\epsilon > 0$ is a small parameter. Note that, for any positive value of ϵ the resulting IVP is well-defined with a unique solution (under standard smoothness assumptions on

f_1 and f_2) for any initial condition, $y(a) = y_0$. On the other hand, in the limiting case of $\epsilon = 0$ this system reduces to,

$$\begin{aligned} z' &= A_{11}z + A_{12}u + f_1, \\ 0 &= A_{21}z + A_{22}u + f_2. \end{aligned}$$

For this problem, only the initial values of the components of $y(x)$ corresponding to $z(x)$, ($z(a) = z_0$), can be arbitrarily specified at $x = a$, while $u(a) = u_0$ must satisfy $A_{22}u_0 = -A_{21}z_0 - f_2$. It should not therefore be surprising that, for small $\epsilon > 0$, numerical difficulties can arise for some initial conditions in the initial region (x close to a). For ϵ nonzero this can lead to a special class of Stiff IVPs while the case $\epsilon = 0$ leads to a class of Singularly Perturbed ODEs. Note that, for ϵ nonzero we have,

$$u' = \frac{A_{22}}{\epsilon}u + \frac{A_{21}}{\epsilon}z + f_2/\epsilon,$$

which for stable problems will be stiff.

One can view the singularly perturbed ODE as an ODE for the variable $z(x)$ with an associated set of algebraic constraints that together will determine the unknown vector valued function, $y(x) = (z(x)\ u(x))^T$. This is a particular instance of the class of problems called Differential/Algebraic Equations (DAEs). We will consider a more general form of this type of equation but the insights and difficulty we have investigated for stiff IVPs will prove helpful and provide insight into understanding and handling the numerical difficulties arising in singularly perturbed DAEs as well as more general DAEs.

Consider the implicit IVP,

$$F(x, y, y') = 0, \quad y(x) \in \Re^n, \quad y(a) = y_0, \text{ for } x \in [a, b].$$

If $\frac{\partial F}{\partial y'}$ is nonsingular in some neighbourhood of $y(x)$ then, for a given value of x and y, one can 'solve' $F(x, y, y') = 0$ to determine $y'(x)$ (Implicit Function Theorem).

- This is called an 'index 0' DAE and one can apply any standard IVP method. Each evaluation of y' requires the solution of a nonlinear system of equations.
- For some methods only one nonlinear system of equations need be solved on each step, even when an implicit formula is involved. For this class of problems there may be little advantage in using an IVP method based on an explicit formula.

A more interesting and challenging case arises when $\frac{\partial F}{\partial y'}$ is of rank $r < n$. In this section we will assume that r is constant for all $x \in [a,\ b]$. In this situation some of the components of the system, $F(x, y, y') = 0$ are independent of y' (hence an algebraic constraint).

As an example, consider the case where the algebraic constraints are explicitly identified as in,

$$\begin{aligned} F(x, z, z', u) &= 0, \\ G(x, u) &= 0, \end{aligned}$$

where,

1. $z(x)$ is the vector of 'differential variables', $z(x) \in \Re^{n_1}$,
2. $u(x)$ is the vector of 'algebraic variables', $u(x) \in \Re^{n_2}$,
3. $F(x, z, z', u)$ has n_1 components.
4. The rank of $\frac{\partial F}{\partial z'}$ is n_1 (ie. it is of full rank).
5. $G(x, u)$ has n_2 linearly independent components.

Such a system is called an <u>Semi-Explicit</u> DAE.

Now consider a more general linear constant coefficient DAE,

$$Ay' + By = f(x), \quad y(x) \in \Re^n.$$

If $\text{rank}(A) = r < n$ then the problem is <u>not</u> index 0 and there exists a nonsingular matrix P such that,

$$PAy' + PBy = Pf(x) \equiv \tilde{f}(x),$$

where

$$PA = \begin{bmatrix} A_1 & A_2 \\ 0 & 0 \end{bmatrix}, \quad PB = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}, \quad [A_1 \; A_2] \text{ of rank } r.$$

Note that one such P can be determined as the product of Householder reflections such that PA is upper triangular. With this structure for (PA , PB) the DAE is equivalent to,

$$A_1 z' + A_2 u' + B_1 z + B_2 u = \tilde{f}_1(x), \quad B_3 z + B_4 u = \tilde{f}_2(x). \tag{8}$$

Differentiating the 'second equation' of (8) we obtain,

$$B_3 z' + B_4 u' = \tilde{f}_2'(x).$$

Replacing the second equation of (8) by this equation we obtain the system:

$$\underbrace{\begin{bmatrix} A_1 & A_2 \\ B_3 & B_4 \end{bmatrix}}_{\bar{A}} \begin{bmatrix} z' \\ u' \end{bmatrix} + \underbrace{\begin{bmatrix} B_1 & B_2 \\ 0 & 0 \end{bmatrix}}_{\bar{B}} \begin{bmatrix} z \\ u \end{bmatrix} = \begin{bmatrix} \tilde{f}_1(x) \\ \tilde{f}_2'(x) \end{bmatrix}.$$

If $\bar{A}$ is nonsingular we have that this transformed DAE is a standard IVP whose solution is also a solution of the original DAE.

- $\tilde{f}_2(x)$ must be differentiable otherwise there is no solution to our original DAE.
- The initial conditions must satisfy the algebraic constraints, $B_3 z_0 + B_4 u_0 = \tilde{f}_2(x)$.
- If $\bar{A}$ is singular (of rank s, $\;r \leq s < n$) then we can repeat this process until the resulting $\bar{A}$ is nonsingular (the number of times the constraints have to differentiated before the resulting $\bar{A}$ is nonsingular is defined to be, μ, the index of the DAE).

In general the higher the index of a DAE, the more difficult the problem – mathematically and numerically.

Index 0 Implicitly defined standard IVP. Standard theory and methods can be used.

Index 1 Usually well behaved and standard numerical methods can be applied with some care and modification.

Index ≥ 2 These problems usually require special structure before a numerical solution is possible. Several such problems arise in practical computation and we will consider some specific examples of index 2 and index 3 problems.

For index 1 problems it is possible (as we will see) to adapt existing IVP methods so they can be used to effectively solve DAEs. For higher index problems it is usually necessary to differentiate the constraints to obtain an equivalent index 0 or index 1 problem. The main difficulty with this approach is that the algebraic constraints will only be indirectly satisfied and the numerical solution will often 'drift off' the manifold of constraints as the integration proceeds. On such problems, the numerical method must monitor the 'constraint residual' and project the numerical solution back onto the constraint manifold when such a drift is detected. Another complication of this approach is the need to compute derivatives of the constraint functions.

- **Solving index 1 problems with standard stiff methods:**
  Consider applying the BDF formula,
  $$y'_{i+1} = \frac{1}{h\beta_0}(y_{i+1} - \sum_{j=1}^{k} \alpha_j y_{i+1-j}),$$
  to an index 1 problem,
  $$F(x, y, y') = 0,$$
  by determining y_{i+1} on each time step by solving (using Newton's method) the nonlinear system:
  $$F(x_{i+1}, y_{i+1}, \frac{1}{h\beta_0}(y_{i+1} - \sum_{j=1}^{k} \alpha_j y_{i+1-j})) = 0.$$

This system of nonlinear equations (in the unknown, y_{i+1}) is solved as $G(y_{i+1}) = 0$ where,

$$G(y_{i+1}) \equiv F(x_{i+1}, y_{i+1}, \frac{1}{h\beta_0}(y_{i+1} - \sum_{j=1}^{k} \alpha_j y_{i+1-j})).$$

We then have that

$$\frac{\partial G}{\partial y_{i+1}} = \frac{\partial F}{\partial y} + (\frac{1}{h\beta_0})\frac{\partial F}{\partial y'},$$

and therefore the associated Newton iteration is,

$$W\delta_{i+1}^{l} = -(h\beta_0)G(y_{i+1}^{l}),$$

where $\delta_{i+1}^{l} = y_{i+1}^{l+1} - y_{i+1}^{l}$ and

$$W \approx (h\beta_0)\frac{\partial F}{\partial y} + \frac{\partial F}{\partial y'}.$$

Note that when the index is greater than 0 (and therefore $\frac{\partial F}{\partial y'}$ singular); as $h \to 0$, W must eventually become ill-conditioned. For index 1 problems though, W is usually not ill-conditioned for 'reasonable' h.

To apply a Runge-Kutta method to this index 1 problem, recall that for standard ODEs, we have,

$$y_{i+1} = y_i + h\sum_{j=1}^{s} w_j k_j,$$

$$k_j = f(x_i + \alpha_j h, y_i + h\sum_{r=1}^{s} \beta_{jr} k_r), \quad j = 1, 2 \cdots s.$$

It is then natural to introduce, for an index 1 DAE, the nonlinear system associated with each step, $G(\underline{k}) = 0$, where $\underline{k} = (k_1, k_2 \cdots k_s)^T$, $G(\underline{k}) = (g_1(\underline{k}), g_2(\underline{k}) \cdots g_s(\underline{k}))^T$ and

$$g_j(\underline{k}) \equiv F(x_i + \alpha_j h, y_i + h\sum_{r=1}^{s} \beta_{jr} k_r,\ k_j) = 0.$$

If one can solve this system on each time step then the Runge-Kutta method can be applied directly to this DAE. Does this nonlinear system have a solution? To investigate this issue consider

$$\frac{\partial G}{\partial \underline{k}} = \begin{bmatrix} (\frac{\partial F}{\partial y'} + h\beta_{11}\frac{\partial F}{\partial y}) & h\beta_{12}\frac{\partial F}{\partial y} & \cdots & h\beta_{1s}\frac{\partial F}{\partial y} \\ h\beta_{21}\frac{\partial F}{\partial y} & (\frac{\partial F}{\partial y'} + h\beta_{22}\frac{\partial F}{\partial y}) & \cdots & h\beta_{2s}\frac{\partial F}{\partial y} \\ \vdots & \vdots & \cdots & \vdots \\ h\beta_{s1}\frac{\partial F}{\partial y} & h\beta_{s2}\frac{\partial F}{\partial y} & \cdots & (\frac{\partial F}{\partial y'} + h\beta_{ss}\frac{\partial F}{\partial y}) \end{bmatrix}.$$

By observing the rank of the diagonal blocks of this matrix we see that the underlying Runge-Kutta formula must be implicit with the rank of the coefficient matrix, (β_{ij}) being full (ie $= s$). This follows since $\frac{\partial F}{\partial y'}$ is singular and an explicit formula would have $\beta_{ij} = 0$ for $j \geq i$. Even with an implicit Runge-Kutta formula we will have ill-conditioning of this matrix as $h \to 0$ as with a BDF method

- **Index 2 problems – Hessenberg, semi-explicit**
  This is a class of Index 2 problems that arise in fluid flow and circuit simulations and have the very special structure,

  $$z' = f(x, z, u), \quad g(z) = 0,$$

  with $g_z f_u$ of full rank. In this case we can differentiate the constraints once to obtain the equivalent index 1 problem,

  $$z' = f(x, z, u), \quad g_z z' \equiv g_z f = 0,$$

  or we can differentiate twice to obtain the equivalent index 0 problem,

  $$z' = f(x, z, u), \quad (g_z f_u) u' = -g_{zz} f^2 - g_z f_x - g_z f_z f.$$

- **Index 3 problems – Constrained Mechanical systems**
  These systems arise in areas such as robotics and vehicle simulators where one models mechanical systems. Consider a mathematical model where $q(x)$ represents position, and $u(x)$ represents velocity. The DAE is then generally of the form,

  $$\begin{aligned} q' &= u, \\ M(q)u' &= f(q, u) - G^T(q)\lambda, \\ 0 &= g(q), \end{aligned}$$

  where $G(q) = \frac{\partial g}{\partial q}$, $GM^{-1}G^T$ has full rank and λ is a vector of 'Lagrange multipliers' – the algebraic variables. Note that the constraints are only on the 'position' variables and one can differentiate the constraints 3 times to obtain an equivalent standard IVP (left as an exercise) of index 0.

  More generally, differentiating the explictly identified constraints can be used to reduce the index 3 problem,

  $$\begin{aligned} y' &= f(x, y, z), \\ z' &= k(x, y, z, \lambda), \\ 0 &= g(y), \end{aligned}$$

  (with $g_y f_z k_\lambda$ of full rank) to a index 0 or index 1 DAE.

2.11 Delay Differential Equations (DDEs)

This topic is not addressed in the text. It is discussed at length in section II.15 of Hairer, Norsett and Wanner and in Chapter 4 of Shampine, Gladwell and Thomson.

The basic problem, in its simplest form can be seen as:

$$y' = f(x, y, y(x - \sigma)), \quad \text{on an interval } [x_0, x_F],$$

where the initial conditions must be specified on an initial interval, $y(x) = \phi(x)$, for $x \in [x_{-1}, x_0]$. A more typical example would involve multiple delays.

- Such equations arise in mathematical modelling in population dynamics, the spread of diseases, flight simulators etc.

- These problems can exhibit interesting dynamics with very few components.

- **An interesting example** (Hairer, Norsett and Wanner–p.295)
  Consider simulating the spread of an infectious disease. Let y_1 represent the susceptible portion of the population, y_2 represent the infected portion of the population and y_3 represent the immunized portion of the population. Then, assuming a 'rate constant' = 1, (quantifying how quickly an infection will spread), we have,

$$\begin{aligned} y_1' &= -y_1 y_2, \\ y_2' &= y_1 y_2 - y_2, \\ y_3' &= y_2. \end{aligned}$$

The solutions to this standard ODE, after an infection is introduced, typically will have the form shown in Fig.3.

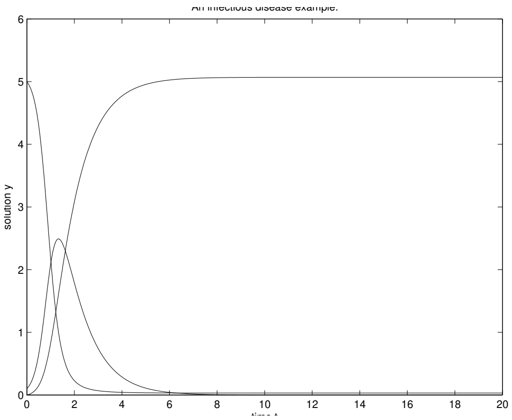


Figure 3: Typical solution of an ODE infectious disease model

Now assume that the immunized group becomes succeptible after 10 units of time and that there is an incubation period of 1 unit. Our ODE then becomes,

$$\begin{aligned} y_1' &= -y_1(x)y_2(x-1) + y_2(x-10), \\ y_2' &= y_1(x)y_2(x-1) - y_2(x), \\ y_3' &= y_2(x) - y_2(x-10). \end{aligned}$$

This model leads to very interesting periodic solutions and more accurately simulates the spread of real infections. For example if we assume $y_1(x) = 5.0, y_2(x) = .1, y_3(x) = 1.0$ for $x \leq 0$, then this DDE has the solution shown in Fig. 4.

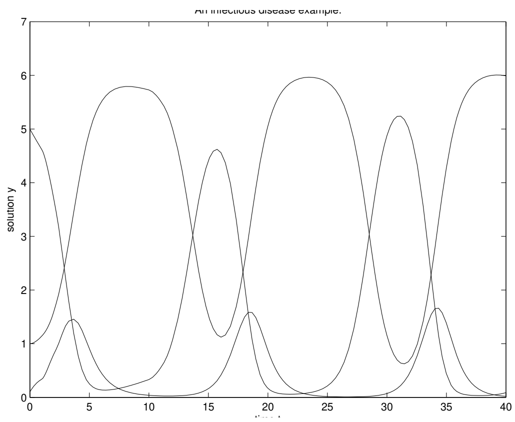


Figure 4: Typical solution of a DDE infectious disease model

There are several special difficulties that can arise when approximating the solution of DDEs. We will consider several such difficulties and how they can be resolved.

1. **Discontinuities** (Hairer, Norsett and Wanner– p. 286)
   Consider the exact solution of the simple scalar problem with a constant delay,

   $$y' = -y(x-1), \quad \text{for } \ x \in [0,3], \quad y(x) \equiv 1 \ \text{ on } \ [-1,0].$$

   (a) On $[0,1)$ $y' = -y(x-1) = -1 \ \Rightarrow y(x) = -x + c_1$, and $y(0) = 1$ implies $c_1 = 1$ and therefore $y(x) = 1 - x$.
   (b) On $[1,2)$ $y' = -y(x-1) = -[1-(x-1)] = x-2 \ \Rightarrow y(x) = \frac{x^2}{2} - 2x + c_2$ and $y(1) = 0$ implies $c_2 = \frac{3}{2}$ and therefore $y(x) = \frac{x^2}{2} - 2x + \frac{3}{2}$.
   (c) On $[2,3]$ $y' = -y(x-1) = -[\frac{(x-1)^2}{2} - 2(x-1) + \frac{3}{2}] = \frac{-x^2}{2} + 3x - 4 \Rightarrow$ $y(x) = -\frac{x^3}{6} + \frac{3x^2}{2} - 4x + c_3$ and $y(2) = -\frac{1}{2}$ implies $c_3 = \frac{17}{6}$ and therefore $y(x) = -\frac{x^3}{6} + \frac{3x^2}{2} - 4x + \frac{17}{6}$.

   The solution, $y(x)$ (see Fig. 5), has a discontinuous first derivative at $x = 0$; a discontinuous second derivative at $x = 1$; and a discontinuous third derivative at $x = 2$.

   In general a discontinuity in one of the low order derivatives of the exact solution will be introduced whenever $x - \sigma = x_0$ or $x - \sigma$ coincides with another point of discontinuity. The detection and location of these inherent discontinuities is important if a numerical method is to be effective. The situation is complicated if multiple delays are present (as in the first example) or if the delay is time-dependent and/or state-dependent (ie., the delay term is $x - \sigma(x,y)$). In most cases the solution will become smoother as it evolves since the order of these propogated discontinuities will increase each time they are propogated.

2. **Variable Delays**
   For any IVP method that determines a piecewise polynomial approximation, $S(x) \approx$

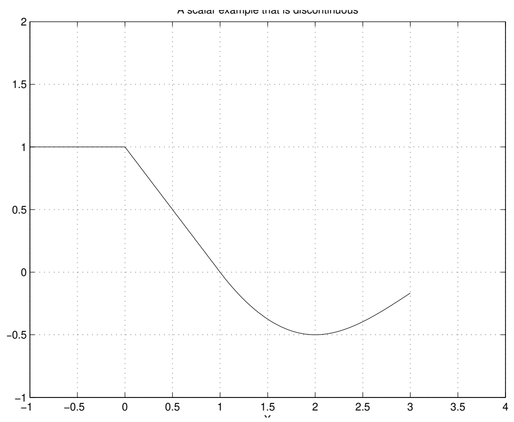


Figure 5: A Scalar Example with a Discontinuous Exact Solution

$y(x)$, one can consider adapting the method to handle a DDE with variable delay, $y' = f(x, y(x), y(x - \sigma(x, y)))$. We will consider, in particular, the application of an explicit CRK method where $S(x)$ is defined by $u_i(x),\ \ i = 0, 1 \cdots N - 1$,

$$u_i(x) = y_i + h_{i+1} \sum_{j=1}^{\bar{s}} b_j(\frac{x - x_i}{h_{i+1}})k_j,$$

$b_j(\tau)$ is a polynomial of degree at most p, $h_{i+1} = x_{i+1} - x_i$, and $\tau = \frac{x - x_i}{h_{i+1}}$. Let Y_j be defined by,

$$Y_j = y_i + h_{i+1} \sum_{r=1}^{\bar{s}} \beta_{rj} k_r \ \text{ for } \ j = 1, 2 \cdots s.$$

For a DDE it is natural to define the k_j's by the equations,

$$k_j = f(x_i + \alpha_j h_{i+1}, Y_j, S(x_i + \alpha_j h_{i+1} - \sigma(x_i + \alpha_j h_{i+1}, Y_j))).$$

Note that if $h_{i+1} \leq \sigma(x, y)$ then the 'delayed' solution value, $S(x_i + \alpha_j h_{i+1} - \sigma(x_i + \alpha_j h_{i+1}, Y_j))$ (required in the definition of k_j) are explicitly available using $S(x)$ defined for $x \leq x_i$ by the local polynomials,

$$\{u_r(x)\}_{r=0}^{i-1},$$

since $x_i + \alpha_j h_{i+1} - \sigma(x_i + \alpha_j h_{i+1}, Y_j) \leq x_i$. Note also that if $x_i + \alpha_j h_{i+1} - \sigma(x_i + \alpha_j h_{i+1}, Y_j) \leq x_0$ then the initial function, $\phi(x)$, must be used. Variable delay problems can then be solved using a CRK method (or a multistep method) provided $h \leq \sigma(x, y)$ is not a severe constraint and discontinuities are detected and carefully handled. One way to handle the detected discontinuities is to modify the stepsize selection strategy (of the CRK method) so that all potential points of discontinuity be forced to be meshpoints.

3. **Small or Vanishing Delays**
When $h \leq \sigma(x, y)$ is a severe constraint (as can happen when $\sigma(x, y) \to 0$), we

can avoid this stepsize constraint by introducing an alternative scheme to evaluate $S(x)$ when $x \in [x_i, x_{i+1}]$. The obvious choice is to use, $u_i(x)$, and this would make some or all of the equations defining the k_j's implicit,

$$k_j = f(x_i + \alpha_j h_{i+1}, Y_j, y_i + h_{i+1} \sum_{r=1}^{\bar{s}} b_r(\tau_j^*) k_r),$$

where $\tau_j^* = x_i + \alpha_j h_{i+1} - \sigma(x_i + \alpha_j h_{i+1}, Y_j)$ and a predictor/corrector or modified Newton iteration must be used.

4. **Derivative Delays – Neutral DDEs**
When a 'delayed' derivative appears in the DDE, the analysis as well as the numerics is more difficult. Consider the scalar problem,

$$y' = f(x, y, y'(x - \sigma(x, y))),$$

where $y'(x) = \phi'(x)$ is specified on $[x_{-1}, x_0]$ and we are interested in the solution on $[x_0, x_F]$. For these problems discontinuities propogate and persist at the same order (if there is a discontinuity of y' at the initial point, it is propogated as a discontinuity in y' at all future points as well). The accuracy of $S(x)$ is more critical for these problems as is the careful handling of discontinuities.

Software is available for DDEs that can cope with multiple delays, neutral delays and vanishing delays (see, for example, DDVERK from Netlib, or RADAR5 from, 'http://www.unige.ch/math/folks/hairer').

An 'improved Fortran-90 implementation of DDVERK, DDVERK90, developed as part of his MSc thesis by H. ZivariPiran as well as a DDE PSE, DDEM, also developed by H. ZivariPiran, in C as part of his PhD thesis, are both available from the link: `http://www.cs.utoronto.ca/~hzp/index_files/software.html`.

For a restricted class of constant delay problems the MATLAB routine, dde23 can be used.

2.12 An example of software and its use:

Any general-purpose numerical method for DDEs that can be used for multiple delays, and both retarded and neutral problems must have a complex calling sequence just to specify the 'mathematical' problem,

$$\begin{aligned} y' &= f(x, y(x), y(x - \sigma_1) \cdots y(x - \sigma_k), y'(x - \sigma_{k+1}), \\ &\quad \cdots y'(x - \sigma_{k+\ell}), \text{ for } x_0 \le x \le x_F, \end{aligned}$$

where

$$y(x) = \phi(x), \;\; y'(x) = \phi'(x), \;\; \text{for } x \le x_0,$$

and

$$\sigma_i \equiv \sigma_i(x, y(x)) \geq 0 \text{ for } i = 1, 2 \cdots k + \ell.$$

Note that when $\ell = 0$ the problem is consdered to be a 'Retarded' DE (RDE) and when $\ell > 0$ it is a Neutral DE (NDE).

As we have seen an effective numerical method for this class of problems will determine a piecewise approximation, $S(x)$, defined for $x \in [x_0, x_F]$ with and associated defect, $\delta(x)$,

$$\delta(x) = f(x, S(x), S(x - \sigma_1), \cdots S(x - \sigma_k), S'(x - \sigma_{k+1}), \cdots S'(x - \sigma_{k+\ell})).$$

We will investigate the use of the method ddverk, based on an underlying CRK formula, which when be applied to a DDE of the above form with a prescribed accuracy, TOL will attempt to produce a piecewise polynomial, $S(x)$ whose defect is bounded in norm by TOL in the interval of interest.

Consider the scalar NDE,

$$\begin{aligned} y' &= cos(x)(1 + y(xy^2(x))) + 0.1y(x)y'(xy^2(x)) + \\ &\quad .9sin(x)cos(xsin^2(x)) - sin(x + xsin^2(x)), \end{aligned}$$

with

$$x_0 = 0, \ x_F = \pi, \ \phi(x) = 0, \ \text{ and } \ \phi'(x) = 1 \quad \text{for } \ x \leq 0.$$

The true solution to this problem is $y(x) = sin(x)$ and it has vanishing delays at $x = 0, \pi/2, \pi$.

The next few pages illustrate the calling sequence of ddverk, a sample driver to solve this test problem, and the output from a typical run.

```
      SUBROUTINE DDVERK(N,NW,MU,OMEGA,IND,LOLD,T,TEND,Y,TOL,TOLD,KOLD
     +     ,W,C,FCN,SIGMA,YINIT,DYINIT)

C***********************************************************************
C                                                                      *
C            Description of DDVERK
C                                                                      *
C     This routine calls the delay and discontinuity routines.
C     Its purpose is to determine a numerical solution of a
C     delay differential equation (DDE). The numerical solution
C     that is generated is a piecewise polynomial, S(t), that
C     approximates the solution on the interval of interest.
C     This routine first initializes parameters and then calls the DVERK
C     IVP method, RDMETH. After each step of the integration,
C     RDMETH returns to this routine and this routine then checks if a
C     discontinuity exists in the step. If the existence of a          *
C     discontinuity is suspected, this routine calls DSLOC and         *
C     DSLOC detects the discontinuity point. Finally, if the step is   *
C     accepted, DDVERK calls STORE or STORED to store information      *
C     necessary to evaluate the piecewise polynomial, S(t).
C                                                                      *
C***********************************************************************
C                                                                      *
C                          Calling sequence                            *
C                                                                      *
C     CALL DDVERK(N,NW,MU,OMEGA,IND,LOLD,T,TEND,Y,TOL,TOLD,KOLD,W,C,   *
C    +            FCN,SIGMA,YINIT,DYINIT)                              *
C                                                                      *
C     INTEGER N,NW,IND,LOLD,MU,OMEGA                                   *
C     DOUBLE PRECISION T,TEND,Y(N),TOL,TOLD(LOLD),KOLD(NW,7,LOLD)      *
C     DOUBLE PRECISION W(NW,20+MU+OMEGA), C(*)                         *
C     EXTERNAL FCN,SIGMA,YINIT,DYINIT                                  *
C                                                                      *
C***********************************************************************
C                                                                      *
C                       Arguments for DDVERK                           *
C     These arguments are consistent (in most cases identical to) those
C     of the DVERK family of methods for standard IVPs.
C                                                                      *
C    N      Number of equations                                        *
C    NW     First dimension of workspace W and queue KOLD              *
C           NW must be at least MAX(N,MU+OMEGA)                        *
C           Note that KOLD is used to store the history of stage
C           vectors associated with each time step. It can then
C           be used to define the local polynomials introduced
```

```
C          on each time step to define the global interpolant, S(t).
C    MU    The number of solution delay terms                        *
C    OMEGA The number of derivative delay terms                      *
C    IND   Indicator --- same as for DVERK except for IND = 7:
C          ON INITIAL ENTRY IND MUST BE SET EQUAL TO  EITHER
C          1  OR  2. IF THE USER DOES NOT WISH TO USE ANY OPTIONS, HE *
C          SHOULD SET IND TO 1 - ALL THAT REMAINS FOR THE USER TO  DO *
C          THEN  IS  TO  DECLARE WORKSPACE AND SPECIFY THE PROBLEM
C          PARAMETERS. THE USER
C          MAY ALSO  SELECT  VARIOUS  OPTIONS  ON  INITIAL  ENTRY  BY *
C          SETTING IND = 2 AND INITIALIZING THE FIRST 9 COMPONENTS OF *
C          C AS DESCRIBED IN THE NEXT SECTION.  HE MAY ALSO  RE-ENTER *
C          THE  SUBROUTINE  WITH IND = 3 AS MENTIONED AGAIN BELOW. IN *
C          ANY EVENT, THE SUBROUTINE RETURNS WITH IND EQUAL TO       *
C             3 AFTER A NORMAL RETURN                                *
C             4,5,6 OR 7 AFTER AN INTERRUPT (SEE OPTIONS C(8), C(9))  *
C             -1, -2, OR -3 AFTER AN ERROR CONDITION (SEE BELOW)      *
C                                                                    *
C    LOLD  Length of queue (used for the efficient storage of S(t))
C    T     Independent variable
C    TEND  Value of T to which integration is to be carried out       *
C    Y     Solution vector at T                                       *
C    TOL   Error Tolerance
C    TOLD  Queue of discrete values t associated with
C          with the piecewise polynomial, S(t)
C    KOLD  Queue of solution approximations and stage vectors
C          associated with each time step (used to define S(t))
C          The first dimension of KOLD must be NW                     *
C          The second dimension of KOLD must be at least 7            *
C          The third dimension of KOLD must be LOLD                   *
C    W     Workspace array                                            *
C          The first dimension of W must be NW                        *
C          The second dimension of W must be at least 20 + ND         *
C          where ND is the total number of delay terms (= MU + OMEGA)
C    C     Communication vector - The dimension must be greater than  *
C          or equal to 33. C(1) to C(24) have the same interpretation
C          as in DVERK (see below).
C    FCN   Name of subroutine for computing the derivative
C          (must be supplied by the user)
C    SIGMA Name of subroutine for computing the delay argument(s)
C          (must be supplied by the user)
C    YINIT Name of subroutine for computing initial values on the     *
C          initial interval. (must be supplied by the user)
C    DYINIT Name of subroutine for computing derivative of initial    *
C          values on the  initial interval. (must be supplied by      *
```

```
C          the user)
C                                                                     *
C*********************************************************************
C                                                                     *
C                          Communication vector                       *
C                                                                     *
C           C(1) - C(24) are interpreted as they are for any IVP
C           method in the DVERK family:
C
C********************************************************************
C                                                                    *
C     OPTIONS - IF THE SUBROUTINE IS ENTERED WITH IND = 1,
C THE SUBROUTINE USES ONLY DEFAULT VALUES  FOR  EACH  OPTION.  IF  THE *
C SUBROUTINE  IS  ENTERED  WITH IND = 2, THE USER MUST SPECIFY EACH OF *
C THE FIRST 33 COMPONENTS - NORMALLY HE WOULD FIRST SET THEM ALL  TO  0,
C AND  THEN  MAKE  NON-ZERO  THOSE  THAT  CORRESPOND TO THE PARTICULAR *
C OPTIONS HE WISHES TO SELECT. IN ANY EVENT, OPTIONS MAY BE CHANGED ON *
C RE-ENTRY  TO  THE  SUBROUTINE  -  BUT IF THE USER CHANGES ANY OF THE *
C OPTIONS, OR TOL, IN THE COURSE OF A CALCULATION HE SHOULD BE CAREFUL *
C ABOUT  HOW  SUCH CHANGES AFFECT THE SUBROUTINE - IT MAY BE BETTER TO *
C RESTART WITH IND = 1 OR 2. (COMPONENTS 10 TO 24 OF C ARE USED BY THE *
C PROGRAM  -  THE INFORMATION IS AVAILABLE TO THE USER, BUT SHOULD NOT *
C NORMALLY BE CHANGED BY HIM.)                                         *
C                                                                      *
C  C(1)  ERROR CONTROL INDICATOR - THE NORM OF THE LOCAL ERROR IS  THE *
C          MAX  NORM  OF  THE  WEIGHTED  ERROR  ESTIMATE  VECTOR, THE *
C          WEIGHTS BEING DETERMINED ACCORDING TO THE VALUE OF C(1) -  *
C             IF C(1)=1 THE WEIGHTS ARE 1 (ABSOLUTE ERROR CONTROL)    *
C             IF C(1)=2 THE WEIGHTS ARE 1/ABS(Y(K))  (RELATIVE  ERROR *
C                CONTROL)                                             *
C             IF C(1)=3 THE  WEIGHTS  ARE  1/MAX(ABS(C(2)),ABS(Y(K))) *
C                (RELATIVE  ERROR  CONTROL,  UNLESS ABS(Y(K)) IS LESS *
C                THAN THE FLOOR VALUE, ABS(C(2)) )                    *
C             IF C(1)=4 THE WEIGHTS ARE 1/MAX(ABS(C(K+30)),ABS(Y(K))) *
C                (HERE INDIVIDUAL FLOOR VALUES ARE USED)              *
C             IF C(1)=5 THE WEIGHTS ARE 1/ABS(C(K+30))                *
C             FOR ALL OTHER VALUES OF C(1), INCLUDING  C(1) = 0,  THE *
C                DEFAULT  VALUES  OF  THE  WEIGHTS  ARE  TAKEN  TO BE *
C                1/MAX(1,ABS(Y(K))), AS MENTIONED EARLIER             *
C          (IN THE TWO CASES C(1) = 4 OR 5 THE USER MUST DECLARE  THE
C          DIMENSION OF C TO BE AT LEAST N+30 AND MUST INITIALIZE THE
C          COMPONENTS C(31), C(32), ..., C(N+30).)                    *
C                                                                     *
C  C(2)  FLOOR VALUE - USED WHEN THE INDICATOR C(1) HAS THE VALUE 3    *
C  C(3)  HMIN SPECIFICATION - IF NOT ZERO, THE SUBROUTINE CHOOSES HMIN *
```

```
C          TO BE ABS(C(3)) - OTHERWISE IT USES THE DEFAULT VALUE      *
C             10*MAX(DWARF,RREB*MAX(WEIGHTED NORM Y/TOL,ABS(X))),     *
C          WHERE DWARF IS A VERY SMALL POSITIVE  MACHINE  NUMBER  AND *
C          RREB IS THE RELATIVE ROUNDOFF ERROR BOUND                  *
C  C(4)  HSTART SPECIFICATION - IF NOT ZERO, THE SUBROUTINE  WILL  USE *
C          AN  INITIAL  HMAG EQUAL TO ABS(C(4)), EXCEPT OF COURSE FOR *
C          THE RESTRICTIONS IMPOSED BY HMIN AND HMAX  -  OTHERWISE IT *
C          USES THE DEFAULT VALUE OF HMAX*(TOL)**(1/6)                *
C  C(5)  SCALE SPECIFICATION - THIS IS INTENDED TO BE A MEASURE OF THE *
C          SCALE OF THE PROBLEM - LARGER VALUES OF SCALE TEND TO MAKE *
C          THE METHOD MORE RELIABLE, FIRST  BY  POSSIBLY  RESTRICTING *
C          HMAX  (AS  DESCRIBED  BELOW) AND SECOND, BY TIGHTENING THE *
C          ACCEPTANCE REQUIREMENT - IF C(5) IS ZERO, A DEFAULT  VALUE *
C          OF  1  IS  USED.  FOR  LINEAR  HOMOGENEOUS  PROBLEMS  WITH *
C          CONSTANT COEFFICIENTS, AN APPROPRIATE VALUE FOR SCALE IS A *
C          NORM  OF  THE  ASSOCIATED  MATRIX.  FOR OTHER PROBLEMS, AN *
C          APPROXIMATION TO  AN  AVERAGE  VALUE  OF  A  NORM  OF  THE *
C          JACOBIAN ALONG THE TRAJECTORY MAY BE APPROPRIATE           *
C                                                                     *
C  C(6)  HMAX SPECIFICATION - FOUR CASES ARE POSSIBLE                 *
C          IF C(6).NE.0 AND C(5).NE.0, HMAX IS TAKEN TO BE            *
C             MIN(ABS(C(6)),2/ABS(C(5)))                              *
C          IF C(6).NE.0 AND C(5).EQ.0, HMAX IS TAKEN TO BE  ABS(C(6)) *
C          IF C(6).EQ.0 AND C(5).NE.0, HMAX IS TAKEN TO BE            *
C             2/ABS(C(5))                                             *
C          IF C(6).EQ.0 AND C(5).EQ.0, HMAX IS GIVEN A DEFAULT  VALUE *
C             OF 2                                                    *
C                                                                     *
C  C(7)  MAXIMUM NUMBER OF FUNCTION EVALUATIONS  -  IF  NOT  ZERO,  AN *
C          ERROR  RETURN WITH IND = -1 WILL BE CAUSED WHEN THE NUMBER *
C          OF FUNCTION EVALUATIONS EXCEEDS ABS(C(7))                  *
C                                                                     *
C  C(8)  INTERRUPT NUMBER  1  -  IF  NOT  ZERO,  THE  SUBROUTINE  WILL *
C          INTERRUPT   THE  CALCULATIONS  AFTER  IT  HAS  CHOSEN  ITS *
C          PRELIMINARY VALUE OF HMAG, AND JUST BEFORE CHOOSING HTRIAL *
C          AND  XTRIAL  IN  PREPARATION FOR TAKING A STEP (HTRIAL MAY *
C          DIFFER FROM HMAG IN SIGN, AND MAY  REQUIRE  ADJUSTMENT  IF *
C          XEND  IS  NEAR) - THE SUBROUTINE RETURNS WITH IND = 4, AND *
C          WILL RESUME CALCULATION AT THE POINT  OF  INTERRUPTION  IF *
C          RE-ENTERED WITH IND = 4                                    *
C                                                                     *
C  C(9)  INTERRUPT NUMBER  2  -  IF  NOT  ZERO,  THE  SUBROUTINE  WILL *
C          INTERRUPT   THE  CALCULATIONS  IMMEDIATELY  AFTER  IT  HAS *
C          DECIDED WHETHER OR NOT TO ACCEPT THE RESULT  OF  THE  MOST *
C          RECENT  TRIAL STEP, WITH IND = 5 IF IT PLANS TO ACCEPT, OR *
```

```
C           IND = 6 IF IT PLANS TO REJECT -  Y(*)  IS  THE  PREVIOUSLY *
C           ACCEPTED  RESULT, WHILE W(*,9) IS THE NEWLY COMPUTED TRIAL *
C           VALUE, AND W(*,2) IS THE UNWEIGHTED ERROR ESTIMATE VECTOR. *
C           THE  SUBROUTINE  WILL  RESUME CALCULATIONS AT THE POINT OF *
C           INTERRUPTION ON RE-ENTRY WITH IND = 5 OR 6. (THE USER  MAY *
C           CHANGE IND IN THIS CASE IF HE WISHES, FOR EXAMPLE TO FORCE *
C           ACCEPTANCE OF A STEP THAT WOULD OTHERWISE BE REJECTED,  OR *
C           VICE VERSA. HE CAN ALSO RESTART WITH IND = 1 OR 2.)        *
C                                                                      *
C***********************************************************************
C                                                                      *
C  SUMMARY OF THE COMPONENTS OF THE COMMUNICATIONS VECTOR              *
C                                                                      *
C     PRESCRIBED AT THE OPTION       DETERMINED BY THE PROGRAM         *
C           OF THE USER                                                *
C                                                                      *
C                                    C(10) RREB(REL ROUNDOFF ERR BND)  *
C     C(1) ERROR CONTROL INDICATOR   C(11) DWARF (VERY SMALL MACH NO)  *
C     C(2) FLOOR VALUE               C(12) WEIGHTED NORM Y             *
C     C(3) HMIN SPECIFICATION        C(13) HMIN                        *
C     C(4) HSTART SPECIFICATION      C(14) HMAG                        *
C     C(5) SCALE SPECIFICATION       C(15) SCALE                       *
C     C(6) HMAX SPECIFICATION        C(16) HMAX                        *
C     C(7) MAX NO OF FCN EVALS       C(17) XTRIAL                      *
C     C(8) INTERRUPT NO 1            C(18) HTRIAL                      *
C     C(9) INTERRUPT NO 2            C(19) EST                         *
C                                    C(20) PREVIOUS XEND               *
C                                    C(21) FLAG FOR XEND               *
C                                    C(22) NO OF SUCCESSFUL STEPS      *
C                                    C(23) NO OF SUCCESSIVE FAILURES   *
C                                    C(24) NO OF FCN EVALS             *
C  IF C(1) = 4 OR 5, C(31), C(32), ... C(N+30) ARE FLOOR VALUES        *
C
C  RREB and DWARF are machine dependent constants currently set so     *
C  that they should be appropriate for most machines.  However, it may *
C  be appropriate to change them when this program is installed on a   *
C  new machine.                           K.R.J.   3 Oct 1991.         *
C                                                                      *
C***********************************************************************
C
C  Additional components of C are required for handling delay problems
C
C
C     C(25) Initial time                                                *
C     C(26) The first index of queue                                    *
```

```
C     C(27) The last index of queue                                     *
C     C(28) Flag indicating whether the first stage vector,
C           K1 needs to be computed on the next
C           step: if C(28) = 0.0 then we need to compute K1 on the next *
C           step                                                        *
C     C(29) Flag indicating which of the extrapolation or the special   *
C           interpolant is to be used: if C(29) = 0.0 then we use
C           extrapolation (used when iteration must be employed for
C           small delays and an initial starting guess is required)
C     C(30) TL values used in discontinuity locating routine DSLOC.     *
C     C(31) TH - TL used in DSLOC                                       *
C     C(32) Flag indicating which of the defect estimates is to be used:
C              1: 1pt, 2: 2pt, 3: asymptotically valid                  *
C              The default value is 1.                                  *
C     C(33) TS value (location of of a discontinuous point)
C           used in ddverk
C                                                                       *
C                                                                       *
C************************************************************************
C                                                                       *
C                             Work array                                *
C                                                                       *
C     W(*, 1:11)          K1 to K11 of CRK formula                      *
C     W(*, 12)            Solution at the right end point of the        *
C                         step                                         *
C     W(*, 13)            Work space                                    *
C     W(*, 14)            Estimate of Defect                            *
C     W(*, 15)            Delay argument                                *
C     W(*, 16)            Work space                                    *
C     W(*, 17)            Work space                                    *
C     W(*, 18)            Work space                                    *
C     W(*, 21:20+ND)      Delay values                                  *
C                                                                       *
C************************************************************************
```

```
C ... Sample program to solve Problem H2 by DDVERK

C ... Declaration of variables
C     .. Parameter integers ..
      INTEGER LOLD,N,NC,NW,MU,OMEGA
C        LOLD: the size of history queue
C        N: the number of equations
C        NC: the size of communication array (must be greater than 33)
C        NW: the leading dimension of work array (must be greater than
C            max(N,MU+OMEGA) )
C        MU: The number of solution delay terms
C        OMEGA: The number of derivative delay terms

C     .. Parameters ..
      PARAMETER(LOLD=100,N=1,NC=33,NW=2,MU=1,OMEGA=1,NSAMPLE=1001)

C     .. Scalar integers ..
      INTEGER I,IND
C        I: index for DO LOOP
C        IND: indicator

C     .. Scalar doubles ..
      DOUBLE PRECISION T,TEND,TOL,DM1,DELFM
C        T: independent variable
C        TEND: end time of integration
C        TOL: tolerance

C     .. Array doubles ..
      DOUBLE PRECISION Y(N),C(NC),W(NW,30)
      DOUBLE PRECISION TOLD(LOLD),KOLD(NW,7,LOLD)
C
C TT and YY correspond to the discrete solution computed by the
C underlying discrete RK formula and TTFM and YYFM correspond
C to the CRK formula, S(x), evaluated at the set of NSAMPLE
C sample points equally space in [T_0, TEND].
C
      DOUBLE PRECISION TT(100), YY(N, 100), YYP(N, 100)
      DOUBLE PRECISION TTFM(NSAMPLE), YYFM(N, NSAMPLE)
C        Y: solution
C        C: communication vector
C        W: work array (the second dimension of W should be at least
C           20 + MU + OMEGA)


C        TOLD: vector for storing T values
```

```
C        KOLD: array for storing solutions and derivatives
C              (the second dimension must be 7)

C     .. External subroutines ..
      EXTERNAL SIGMA,FCN,YINIT,DYINIT

C ... Solve the problem over different tolerances from 10^(-2) to 10^(-10)

      DO 10 I = -2, -10, -2
         IND = 2
         DO 3 k = 1,33
            C(k) = 0.e0
 3       CONTINUE
         C(32) =1.e0
         TOL = 10.D0**I
         PRINT '("***********************************")'
         PRINT '("TOL = ", 1PD9.2 )', TOL
         PRINT '("ERROPT = ", F10.0)', C(32)

C ... Initialization
         T = 0.D0
         Y(1) = 0.D0
         TEND = 4.D0 * DATAN(1.D0)
         TTFM(1) = T
         DELFM = (TEND - T)/ DFLOAT(NSAMPLE - 1)
         DO 5 k = 2, (NSAMPLE - 1)
            TTFM(k) = T + (k-1)*DELFM
 5       CONTINUE
         TTFM(NSAMPLE) = TEND

C ... Loop
 30      CALL DDVERK(N,NW,MU,OMEGA,IND,LOLD,T,TEND,Y,TOL,TOLD,KOLD,
     +     W,C,FCN,SIGMA,YINIT,DYINIT)

         IF (T .LT. TEND) GOTO 30
C ... End loop

C ... Print out the statistics
         PRINT '(/"IND = ", I3)', IND
         PRINT '("T = ", F12.5)', T
         PRINT '("Y = ", D26.15)', Y
         PRINT '("FCN = ", F10.0)', C(24)
         PRINT '("STEP = ", F10.0)', C(22)
         PRINT '("Global Error at T = ", D16.5)',
     +        DABS(Y(1) - DSIN(T))
```

```
C
C
C At this point the global derivative and interpolant routines,
C DERIVG and INTERPG, can be used to evaluate the piecewise polynomial,
C S(x) for any x in [T_0, TEND].
C
      Ksteps = C(22)
      DO 40 k = 1, Ksteps
C
C Determine the values of y_k and y_{k-1} for k = 1, 2 .. Ksteps.
C
         TT(k) = TOLD(k)
         YY(1, k) = KOLD(1, 3, k)
C        YY(2, k) = KOLD(2, 3, k)
         YYP(1, k) = KOLD(1, 4, k)
C        YYP(2, k) = KOLD(2, 4, k)
C        PRINT '("Tdiscrete = ", E12.5)', TT(k)
C        PRINT '("Ydiscrete = ", E12.5)', YY(1,k)
 40   CONTINUE
      TT(Ksteps + 1) = TOLD(Ksteps +1)
      YY(1, Ksteps+1)  = KOLD(1, 3, Ksteps+1)
C     YY(2, Ksteps+1)  = KOLD(2, 1, Ksteps+1)
      YYP(1, Ksteps+1) = KOLD(1, 4, Ksteps+1)
C     YYP(2, Ksteps+1) = KOLD(2, 2, Ksteps+1)
C     PRINT '("Tdiscrete = ", F12.5)', TT(Ksteps+1)
C     PRINT '("Ydiscrete = ", F12.5)', YY(1, Ksteps+1)
      DM1 = 0.e0
      DO 42 kk = 1,Ksteps
         DM1 = DMAX1(DM1, DABS((YY(1,kk) - DSIN(TT(kk)))))
 42   CONTINUE
      PRINT '("Maximum Discrete Global error = ", D16.5)', DM1
C
C Now determine the fine mesh solution using S(x) evaluated
C on the fine mesh of NSAMPLE equally spaced points, TFM(NSAMPLE).
C

      MM = 2
      DO 45 k = 1, (NSAMPLE-1)
         CALL INTRPG(N,NW,MM,TOLD(MM-1),TTFM(k),YYFM(1,k),
     +               TOLD(MM)-TOLD(MM-1),KOLD)
C        PRINT '("Tcont= ", E12.5)', TTFM(k)
C        PRINT '("Ycont= ", E12.5)', YYFM(1,k)
C        PRINT '("Global error = ", D16.5)',
C    +            DABS(YYFM(1,k)-DSIN(TTFM(k)))
```

```
         IF (TTFM(k+1) .GE. TOLD(MM)) THEN
C           LOOP
 41         CONTINUE
               MM = MM+1
               IF (TTFM(k+1) .LE. TOLD(MM)) GO TO 43
               GO TO 41
 43         CONTINUE
         END IF
 45   CONTINUE
      YYFM(1, NSAMPLE) = KOLD(1, 3, Ksteps + 1)
      DM1 = 0.e0
      DO 47 kk = 1, NSAMPLE
         DM1 = DMAX1(DM1, DABS((YYFM(1, kk) - DSIN(TTFM(kk)))))
 47   CONTINUE
      PRINT '("Maximum Continuous Global error = ", D16.5)', DM1
 10   CONTINUE

      END



      SUBROUTINE FCN(N,ND,NW,T,Y,YDEL,DY)

C ... This subroutine computes derivative

C     ... Scalar arguments
      INTEGER N,NW,ND
      DOUBLE PRECISION T
C       N: the number of equations (input)
C       NW: the leading dimension of work array W (input)
C       ND: the number of different delay arguments (input)
C       T: independent variable (input)

C     .. Array arguments
      DOUBLE PRECISION Y(N),YDEL(NW,ND),DY(N)
C       Y: solution at T (input)
C       YDEL: i, j element contain i-th component of delay
C             value at j-th delay argument (input)
C       DY: derivative value at T (output)

      dy(1) = dcos(t) * (1.d0 + ydel(1,1)) + 0.1d0 * y(1) * ydel(1,2)
     &     + 0.9d0 * dsin(t) * dcos(t*dsin(t)**2)
     &     - dsin(t + t * dsin(t)**2)

      RETURN
```

```
      END



      SUBROUTINE SIGMA(N,ND,T,Y,DELARG)

C ... This subroutine computes delay arguments

C     .. Scalar arguments ..
      INTEGER N,ND
      DOUBLE PRECISION T
C       N: the number of equations (input)
C       ND: the number of different delay argument (input)
C       T: independent variable (input)

C     .. Array arguments ..
      DOUBLE PRECISION Y(N),DELARG(ND)
C       Y: solution at T (input)
C       DELARG: delay argument (output)

      DELARG(1) = T * Y(1)**2
      DELARG(2) = T * Y(1)**2

      RETURN
      END



      SUBROUTINE YINIT(N,T,Y)

C ... This subroutine computes solutions at T on initial interval

C     .. Scalar arguments ..
      INTEGER N
      DOUBLE PRECISION T
C       N: the number of equations (input)
C       T: independent variable (input)

C     .. Array arguments ..
      DOUBLE PRECISION , Y(N)
C       Y: initial value at T (output)

         Y(1) = 0.D0

      RETURN
```

```
      END



      SUBROUTINE DYINIT(N,T,DY)

C ... This subroutine computes derivatives at T on initial interval

C     .. Scalar arguments ..
      INTEGER N
      DOUBLE PRECISION T
C       N: the number of equations (input)
C       T: independent variable (input)

C     .. Array arguments ..
      DOUBLE PRECISION DY(N)
C       Y: initial derivative value at T (output)

      DY(1) = 1.D0

      RETURN
      END
```

This is the outut from this sample program run on the above test problem.

```
***********************************
TOL =  1.00D-02
ERROPT =          3.

IND =   3
T =      3.14159
Y =     -0.248414800863683D-03
FCN =       132.
STEP =         7.
Global Error at T =      0.24841D-03
Maximum Discrete Global error =      0.62497D-03
Maximum Continuous Global error =      0.62644D-03
***********************************
TOL =  1.00D-04
ERROPT =          3.

IND =   3
T =      3.14159
Y =     -0.207498170218257D-05
FCN =       254.
STEP =        10.
Global Error at T =      0.20750D-05
Maximum Discrete Global error =      0.92995D-05
Maximum Continuous Global error =      0.11190D-04
***********************************
TOL =  1.00D-06
ERROPT =          3.

IND =   3
T =      3.14159
Y =     -0.244358529799538D-07
FCN =       333.
STEP =        15.
Global Error at T =      0.24436D-07
Maximum Discrete Global error =      0.95730D-07
Maximum Continuous Global error =      0.97124D-07
***********************************
TOL =  1.00D-08
ERROPT =          3.

IND =   3
T =      3.14159
Y =      0.751636530793576D-10
```

```
FCN =       581.
STEP =        29.
Global Error at T =      0.75164D-10
Maximum Discrete Global error =      0.85113D-09
Maximum Continuous Global error =      0.95214D-09
*************************************
TOL =  1.00D-10
ERROPT =         3.

IND =   3
T =      3.14159
Y =      0.627817242637718D-12
FCN =      1062.
STEP =        59.
Global Error at T =      0.62769D-12
Maximum Discrete Global error =      0.10295D-10
Maximum Continuous Global error =      0.10736D-10
```

3 Numerical methods for BVPs

3.1 Superposition (for Linear Problems)

This section is based on a sequence of Technical Reports and publications by M. Scott, H.A. Watts and colleagues who implemented a family of Two-point BV Codes at SANDIA Laboratories in the period 1978 - 1980. Extensions of the basic method, SUPORT, described in this section to handle nonlinear problems (SUPORQ) and eigenvalue problems (SUPORE) are also available.

1. Consider the Linear BVP,
$$y' = F(x)y + g(x),$$
with separated boundary conditions,
$$Ay(a) = y_a, \quad By(b) = y_b,$$
where $y,\ g\ \in \Re^n$, F is an $n \times n$ matrix, A is an $(n-k) \times n$ matrix of rank $n-k$, B is a $k \times n$ matrix of rank k, $y_a \in \Re^{n-k}$ and $y_b \in \Re^k$.

2. The Principle of Superposition (Mathematical Justification)

   - let $u_1(x), u_2(x) \cdots u_k(x)$ be <u>linearly independent</u> solutions of,
   $$u' = F(x)u,$$
   with initial conditions chosen so that
   $$Au_i(a) = 0, \quad \text{for} \quad i = 1, 2 \cdots k.$$
   (That is, the $u_i(a)$'s span the null space of A.) Furthermore, let $v(x)$ satisfy,
   $$v' = F(x)v + g(x), \quad Av(a) = y_a.$$
   - Now if,
   $$U(x) = \begin{bmatrix} u_1(x) & u_2(x) & \cdots & u_k(x) \end{bmatrix}_{n \times k},$$
   we have,
   $$y(x) = v(x) + c_1u_1(x) + c_2u_2(x) \cdots + c_ku_k(x) = v(x) + U(x)\underline{c},$$
   satisfies the differential equation ($y' = F(x)y + g(x)$) as well the 'left' boundary condition, $Ay(a) = y_a$ for any choice of $\underline{c}$.
   - If, in addition, $\underline{c}$ is chosen to satisfy,
   $$By(b) = Bv(b) + BU(b)\underline{c} = y_b,$$
   (or $[BU(b)]\underline{c} = y_b - Bv(b)$), a system of k linear equations in k unknowns, then the differential equations and all the boundary conditions will be satisfied.

3. The Superposition Algorithm is then:

   Choose orthonormal vectors, $u_1, u_2 \cdots u_k$, that span the null space of A, and v_0, such that $Av_0 = y_a$.

   Solve simultaneously the matrix IVP,

$$U' = F(x)U, \;\; U(a) = [u_1 u_2 \cdots u_k];$$

   and

$$v' = F(x)v + g(x), \;\; v(a) = v_0.$$

   Set $y(x) = v(x) + U(x)\underline{c}$, where $\underline{c}$ is the solution of the linear equation,

$$BU(b)\underline{c} = y_b - Bv(b).$$

4. **Difficulties:**

   - The columns of $U(x)$ often tend to become linearly dependent (as x increases).
   - It is often the case that $\|y(x)\|$ may be small while the $\|u_i(x)\|$ are large. This implies a 'loss of significance' in determining $y(x)$, even if the c_i's are accurate.

   To overcome these difficulties, Reorthonormalization is performed at a set of points: (P subintervals)

$$a = x_0 < x_1 \cdots < x_P = b.$$

   Note that orthogonality will ensure that linear independence is retained while the normalization will ensure that the size of the $\|u_i\|$ does not become large. Over each sub-interval, (x_m, x_{m+1}) we solve a matrix IVP,

$$U_m' = F(x)U_m, \;\; U_m(x(m)) = U_m^0;$$

   and the particular IVP,

$$v_m' = F(x)v_m + g(x), \;\; v_m(x_m) = v_m^0.$$

   The initial values are chosen as follows:

   (a) For $m = 0$, choose U_m^0 and v_m^0 as for standard superposition ($Av_0^0 = y_a, \;\; AU_0^0 = 0$).

   (b) For subsequent subintervals, $m > 0$, we first decompose the matrix $U_{m-1}(x_m)$ as,

$$U_{m-1}(x_m) = Q_m R_m,$$

   where Q_m is an $n \times k$ matrix with orthonormal columns and R_m is a $k \times k$ upper triangular matrix. This is a standard linear algebra decomposition and can be accomplished using modified Gram Schmidt or Householder Reflections. We then set the initial conditions for the IVPs on interval m by,

$$U_m^0 = Q_m, \;\; v_m^0 = v_{m-1}(x_m) - Q_m w_m,$$

where $w_m \in \Re^k$ is chosen to make v_m^0 orthogonal to the columns of U_m^0. That is,

$$Q_m^T v_m^0 = 0 \;\; \Rightarrow \;\; Q_m^T v_{m-1}(x_m) - w_m = 0,$$

and therefore

$$w_m = Q_m^T v_{m-1}(x_m)$$

and, from the definition of v_m^0,

$$v_m^0 = [I - Q_m Q_m^T] v_{m-1}(x_m).$$

Now we can introduce the unknown vectors, $\underline{c}_m \in \Re^k$ and let

$$y(x) = y_m(x) \;\text{ for }\; x \in (x_m, x_{m+1}),$$

where,

$$y_m(x) \equiv v_m(x) + U_m(x)\underline{c}_m.$$

(Note that with this definition, $y_m(x)$ satisfies the ODE for any $\underline{c}_m$.) Continuity at x_m will require $y_{m-1}(x_m) = y_m(x_m)$ or,

$$v_{m-1}(x_m) + U_{m-1}(x_m)\underline{c}_{m-1} = v_m(x_m) + U_m(x_m)\underline{c}_m.$$

From the definitions of the initial conditions, U_m^0, v_m^0 we then have,

$$v_{m-1}(x_m) + U_{m-1}(x_m)\underline{c}_{m-1} = v_m^0 + U_m^0\underline{c}_m = v_{m-1}(x_m) - Q_m w_m + Q_m\underline{c}_m.$$

Therefore we have,

$$U_{m-1}(x_m)\underline{c}_{m-1} = Q_m[\underline{c}_m - w_m],$$

or, since $U_{m-1}(x_m) = Q_m R_m$,

$$Q_m R_m c_{m-1} = Q_m[\underline{c}_m - w_m],$$

which, after multiplying on the left by Q_m^T, becomes,

$$R_m\underline{c}_{m-1} = [\underline{c}_m - w_m].$$

This triangular system of linear equations must be satisfied on each subinterval, $m = 1, 2 \cdots (p-1)$ and the boundary condition at $x = b$ requires,

$$BU_{p-1}(b)\underline{c}_{p-1} = y_b - Bv_{p-1}(b).$$

5. **Overview of Superposition Procedure:** Given $a = x_0 < x_1 \cdots < x_P = b$,

   - Determine the intial values for the first interval, $x_0 = a,\ U_0^0,\ v_0^0$.
   - For each interval, $m = 1, 2 \cdots (P-1)$,
     - Use an IV Solver to determine $U_{m-1}(x_m),\ v_{m-1}(x_m)$.
     - Decompose $U_{m-1}(x_m) = Q_m R_m$.
     - Set $U_m^0 = Q_m,\ w_m = Q_m^T v_{m-1}(x_m),\ v_m^0 = v_{m-1}(x_m) - Q_m w_m$.
   - End For
   - Use an IV Solver to determine $U_{P-1}(b),\ v_{P-1}(b)$.
   - Solve $BU_{P-1}(b)\underline{c}_{P-1} = y_b - Bv_{P-1}(b)$.
   - For $m = (P-1), (P-2) \cdots 1$,
     - Solve the triangular system, $R_m \underline{c}_{m-1} = (c_m - w_m)$.
   - End For

   With this procedure we can obtain output of the approximate solution, $y_m(x)$, at a set of output points, t_r, in two ways,

   (a) Locate the interval (x_m, x_{m+1}) containing t_r and then, using the stored values of $\underline{c}_m, v_m^0$, and U_m^0, determine the initial values $z_m^0 = v_m^0 + U_m^0 \underline{c}_m$ and use an IV solver to determine $z_m(x)$ by approximating the solution of the IVP,

   $$z' = f(x)z + g(x), \text{ for } x \in [x_m, x_{m+1}].$$

   (b) Save $U(t_r), v(t_r)$ at all output points during the solution step and then, after the back substitution step for interval m, determine $y_m(t_r)$ using $\underline{c}_m$.

6. **Some Implementation Considerations:**

   - Choosing the initial conditions, $U_0(a)$, so that $AU_0(a) = 0$. This is done using a QR factorization of A^T. With such a Q we set,

   $$U_0 \equiv Q \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \\ & & I_k & \end{bmatrix}.$$

   - Choosing the initial conditions, $v_0(a)$ so that $Av_0(a) = y_a$. This is done using standard LLSQ software to fine the min norm solution of the underdetermined LLSQ problem $Az = y_a$.
   - The current package allows a choice of IV methods to solve the required IVPs. The methods available are: RKF45, DE/STEP and LSODE.
   - The package chooses the orthonormalization points by factoring $U_m(x) = QR$ on each step and inserting a new point when a loss of independence is detected (when small magnitude diagonal elements are observed in R).

3.2 Initial-Value based BVP Methods (for Nonlinear Problems)

In this section we present an overview of Shooting methods for BVPs using, as an example code, the general purpose Multiple Shooting code developed at JPL by F. Krogh and colleagues. It is a state-of-the-art code that can handle efficiently many non-standard extensions of the basic TPBVP. Another code implementing Multiple Shooting with fewer extensions is discussed in detail in the text.

The basic idea behind a shooting method is to view the nonlinear TPBVP as a general nonlinear equation,

$$G(\underline{z}) = 0$$

(where the 'unknown' vector $\underline{z}$ includes, as a subset of its components, the initial value vector, $y(a)$) and solve this equation using Newton's Method – That is, apply the iteration: 'predict' $\underline{z}^0$, then 'iterate":

$$\frac{\partial G}{\partial \underline{z}}|_{\underline{z}^{l-1}}(\Delta \underline{z}^l) = -G(\underline{z}^{l-1}), \;\; \underline{z}^l = \underline{z}^{l-1} + (\Delta \underline{z}^l),$$

for $l = 1, 2 \cdots$.

Consider the TPBVP,

$$y' = f(x, y, \lambda), \;\; x \in [a, b],$$

with separated boundary conditions,

$$Ay(a) = y_a, \;\; By(b) = y_b,$$

where $y \in \Re^n, \; \lambda \in \Re^m \; m \geq 0, \;\; f : [a, b] \times \Re^n \times \Re^m \to \Re^n$, A is an $(n - k) \times n$ matrix of rank $n - k$, and B is a $(k + m) \times n$ matrix of rank $(k + m)$.

1. Assume we have a partition, $a = x_0 < x_1 \cdots < x_P = b$, (the breakpoints), and let $y_i(x, \lambda)$ be the solution of the local IVP,

   $$y' = f(x, y, \lambda), \;\; y_i(x_i, \lambda) = \underline{s}_i, \;\; x \in [x_i, x_{i+1}].$$

   (Note that in this equation, $\lambda, \underline{s}_i$ are assumed fixed.)

   Multiple Shooting involves replacing the original TPBVP with the equivalent (larger but better conditioned) multipoint BVP:

   $$\begin{aligned} g_0 &= A\underline{s}_0 - y_a = 0, \\ g_i &= y_{i-1}(x_i, \lambda) - \underline{s}_i \;\text{ for }\; i = 1, 2 \cdots (P - 1), \\ g_P &= By_{P-1}(b, \lambda) - y_b. \end{aligned}$$

   or $G(\underline{s}_0, \underline{s}_1 \cdots \underline{s}_{P-1}, \lambda) = 0$. This is a system of $nP + m$ nonlinear equations in the $nP + m$ unknowns,

   $$\underline{z} \equiv (\underline{s}_0, \underline{s}_1 \cdots \underline{s}_{P-1}, \lambda)^T.$$

2. A Newton-type iteration to solve this problem requires (for the computation of $\frac{\partial G}{\partial \underline{z}}$),

$$\frac{\partial g_i}{\partial \underline{s}_i} = -I \quad \text{and} \quad \frac{\partial g_i}{\partial \underline{s}_{i-1}} = \frac{\partial y_{i-1}(x_i, \lambda)}{\partial \underline{s}_{i-1}} \equiv Y_{i-1}.$$

Let $Y_{i-1}(x, \lambda) = \frac{\partial}{\partial \underline{s}_{i-1}}(y_{i-1}(x, \lambda))$. We then have that $Y_{i-1}(x, \lambda)$ satisfies the matrix ODE,

$$\begin{aligned}
\frac{d}{dt} Y_{i-1}(x, \lambda) &= \frac{d}{dt} \frac{\partial}{\partial \underline{s}_{i-1}}(y_{i-1}(x, \lambda)) \\
&= \frac{\partial}{\partial \underline{s}_{i-1}} \frac{d}{dt}(y_{i-1}(x, \lambda)) \\
&= \frac{\partial}{\partial \underline{s}_{i-1}} f(x, y_{i-1}(x, \lambda), \lambda) \\
&= \frac{\partial f}{\partial y} \frac{\partial y_{i-1}(x, \lambda)}{\partial \underline{s}_{i-1}} \\
&= \frac{\partial f}{\partial y} Y_{i-1}(x, \lambda).
\end{aligned}$$

with associated initial conditions,

$$Y_{i-1}(x_{i-1}, \lambda) = \frac{\partial}{\partial \underline{s}_{i-1}}(y_{i-1}(x_{i-1}, \lambda)) = \frac{\partial}{\partial \underline{s}_{i-1}}(\underline{s}_{i-1}) = I.$$

Thus we have that $Y_{i-1}(x, \lambda)$ satisfies a linear homogeneous matrix IVP for $x \in [x_{i-1}, x_i]$ and the required partial derivative, $\frac{\partial g_i}{\partial \underline{s}_{i-1}}$ is equal to $Y_{i-1}(x_i, \lambda)$.

3. Similarly, to compute $\frac{\partial g_i}{\partial \lambda}$, we have,

$$\frac{\partial g_i}{\partial \lambda} = \begin{cases} 0 & \text{for } i = 0, \\ \frac{\partial y_{i-1}(x_i, \lambda)}{\partial \lambda} & \text{for } i = 1, 2 \cdots (P-1), \\ B \frac{\partial y_{P-1}(x_P, \lambda)}{\partial \lambda} & \text{for } i = P, \end{cases}$$

and if we define,

$$Y^{\lambda}_{i-1}(x, \lambda) \equiv \frac{\partial}{\partial \lambda}(y_{i-1}(x, \lambda),$$

then $Y^{\lambda}_{i-1}(x, \lambda)$ satisfies the inhomogeneous linear ODE,

$$\begin{aligned}
\frac{d}{dt}(Y^{\lambda}_{i-1}) &= \frac{d}{dt}(\frac{\partial y_{i-1}(x, \lambda)}{\partial \lambda}, \\
&= \frac{\partial}{\partial \lambda}(\frac{d y_{i-1}(x, \lambda)}{dt}, \\
&= \frac{\partial}{\partial \lambda}(f(x, y_{i-1}(x, \lambda), \lambda)), \\
&= \frac{\partial f}{\partial y} \frac{\partial y_{i-1}}{\partial \lambda} + \frac{\partial f}{\partial \lambda}, \\
&= \frac{\partial f}{\partial y} Y^{\lambda}_{i-1} + \frac{\partial f}{\partial \lambda},
\end{aligned}$$

with initial conditions,

$$Y_{i-1}^{\lambda}(x_{i-1},\lambda) = \frac{\partial}{\partial\lambda}(y_{i-1}(x_{i-1},\lambda)) = \frac{\partial}{\partial\lambda}(\underline{s}_{i-1}) = 0.$$

4. Newton's iteration for solving our system of nonlinear equations becomes,
   - Predict $\underline{s}_i^0$ for $i = 0, 1 \cdots (P-1)$ and λ^0 (that is, determine $\underline{z}^0$).
   - For each iteration, $l = 1, 2 \cdots$; determine the correction vectors, $(\Delta\underline{s}_i)^l$ and $(\Delta\lambda)^l$ by solving the linear system:

5.

$$\begin{bmatrix} A & 0 & 0 & \cdots & 0 & 0 \\ Y_0(x_1,\lambda) & -I & 0 & \cdots & 0 & Y_0^{\lambda}(x_1,\lambda) \\ 0 & Y_1(x_2,\lambda) & -I & \cdots & 0 & Y_1^{\lambda}(x_2,\lambda) \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & BY_{P-1}(x_P,\lambda) & BY_{P-1}^{\lambda}(x_P,\lambda) \end{bmatrix} \begin{bmatrix} \Delta\underline{s}_0 \\ \Delta\underline{s}_1 \\ \vdots \\ \Delta\underline{s}_{P-1} \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} \underline{g}_0 \\ \underline{g}_1 \\ \underline{g}_2 \\ \vdots \\ \underline{g}_P \end{bmatrix}$$

Note that the matrix of coefficients and the RHS vector (the Newton Residual) can be computed in groups of n rows by solving three sets of IVPs between adjacent breakpoints, (x_{i-1}, x_i).

- Simple Shooting refers to the case $P = 1$ which results in a dense linear system of equations.
- For large values of P system has sparse block structure with the three sets of IVPs being of size n, n^2, and mn.
- It is straightforward to extend this derivation to acount for nonseparated boundary conditions or nonlinear boundary conditions.

6. **Nonlinear Interior Constraints**:
Consider a standard TPBVP with additional constraints specified as,

$$h_r(y(t_r,\lambda),\lambda) = 0, \quad \text{for } \; r = 1, 2 \cdots s,$$

where $t_r \in [x_{i_r}, x_{i_r+1}]$.

Note that with this type of constraint:

- One can specify multipoint problems or nonlinear endpoint conditions.

- Each constraint adds an extra row to the linear system but the block structure is preserved,

$$\frac{\partial h_r}{\partial \underline{s}_{i_r}} = \frac{\partial h_r}{\partial y}\frac{\partial y_{i_r}}{\partial \underline{s}_{i_r}} = \frac{\partial h_r}{\partial y}|_{t_r, y_{i_r}(t_r,\lambda)} Y_{i_r}(t_r, \lambda),$$

$$\frac{\partial h_r}{\partial \lambda} = \frac{\partial h_r}{\partial y}\frac{\partial y_{i_r}}{\partial \lambda} + \frac{\partial h_r}{\partial \lambda} = \frac{\partial h_r}{\partial y}|_{t_r, y_{i_r}(t_r,\lambda)} Y^{\lambda}_{i_r}(t_r, \lambda) + \frac{\partial h_r}{\partial \lambda}.$$

  Therefore the user must provide subroutines to compute h_r, $\frac{\partial h_r}{\partial y}$, $\frac{\partial h_r}{\partial \lambda}$ as well as f, $\frac{\partial f}{\partial y}$, $\frac{\partial f}{\partial \lambda}$.

- If the total number of constraints is greater than the number of unknowns, $n + k$, the system is treated as an overdetermined nonlinear least squares problem.

7. **Quadrature Constraints**:
   Eigenvalue problems often require orthogonality constraints to ensure the problem is well-posed. As an example consider the ODE, $y'' = -\lambda^2 y$. The solution can be normalized by requiring:

   (a) $\int_0^1 y_i^2 ds = 1$, where y_i is the i^{th} eigenfunction and

   (b) $\int_0^1 y_i y_j ds = 0, \;$ for $\; i \neq j$.

   These constraints can be handled by the multiple shooting code by specifying additional differential equations with appropriate boundary conditions,

   (a) $v_i' = y_i^2(t), \; v(0) = 0, \; v(1) = 1.$

   (b) $w_{i\,j}' = y_i(t) y_j(t), \; w_{i\,j}(0) = 0, \; w_{i\,j}(1) = 0.$

   These 'extra' differential equations are special since for all other components, f_r, of the system we have,

$$\frac{\partial f_r}{\partial v_i} = \frac{\partial f_r}{\partial w_{i\,j}} \equiv 0.$$

   That is, the corresponding columns of $\frac{\partial f}{\partial y}$ are 0.

8. Other Implementation Considerations:

   - The placement (or adjustment) of the 'breakpoints' is determined by monitoring the growth of the variational equations; Y_i, Y_i^{λ}. and the conditioning of the Y_i, $(\|Y_i\| \|Y_i^{-1}\|)$. The user can specify the initial locations.
   - Convergence of the modified Newton iteration can be very sensitive to the initial guess $(\underline{s}_0^0, \underline{s}_1^0 \cdots \underline{s}_{P-1}^0)$. A guess for $\underline{s}_0^0$ is required with the default guess for $\underline{s}_i^0$, $i > 0$ being $\underline{s}_i^0 = \underline{s}_0^0$, if no value is supplied by the user.
   - A block banded matrix technique is used to solve the linear system of equations associated with each Newton iteration.

- Marquardt stabilization is used to improve the chances for convergence (this is particularly useful when only a crude initial guess is available and the initial Newton residual quite large).
- The JPL code allows grouping of the variational equations and quadratures ($Y_i,\ Y_i^\lambda,\ v_i,\ w_{i\,j}$) and a different tolerance can be specified for each group. Higher order ODEs can be handled directly without conversion to first order systems and the initial value method used is a variable order Adams method.
- The NAG and IMSL (Visual Numerics) multiple shooting codes use a ERK code based on an efficient formula pair for the IVPs that must be solved.

9. **Obtaining accurate Initial Predictions**– Continuation.
In solving nonlinear BVPs with any method, codes inevitably use an iteration to solve a nonlinear system of equations and an initial 'guess' of the solution over the interval of interest is expected. While a crude approximation, such as $y(x) \equiv y_0$ may work, a more accurate initial approximation will generally result in more rapid convergence. Continuation is one technique that is often used to enable BVP methods to efficiently solve difficult nonlinear problems. We will first describe the generic approach (that can be used by any BV solver) and then consider how multiple shooting methods are able to adopt this technique in an automatic way.

    - Continuation – The Generic Approach: For many nonlinear problems it is difficult to obtain an approximate solution because insufficient information is known to enable one to choose an appropriate non-uniform mesh and/or a sufficiently accurate initial guess, As a result the modified Newton iteration (associated with the BV method) to solve a discrete problem will not converge. The motivation for continuation is that by solving sequentially a set of related 'nearby' problems we can, for each problem, hope to provide an appropriate initial mesh and initial guess for the solution. This approach can apply to any method that permits a user to specify an initial mesh (or selection of breakpoints for multiple shooting) together with an initial guess for the solution on this mesh. Although the procedure can be automated to some extent it is likely some user interaction or intervention will be required.

      To use this approach, the user must first introduce a scalar parameter, α, so that the differential equation becomes,

      $$y' = f(x, y, \alpha),$$

      with standard boundary conditions (which may also depend on α). This parameterization should be chosen so that $\alpha = 1$ corresonds to the original problem and $\alpha = 0$ corresponds to a problem which can be solved on a known mesh $x_0 < x_1 \cdots < x_N$. (For example, the value $\alpha = 0$ may correspond to a linear problem.)

An overview of this approach is presented below.

-set $\alpha_0 = 0$ and initial mesh $x_0^0 < x_1^0 \cdots x_{N_0}^0$;
-invoke the BV solver to determine the discrete solution on this mesh, $(y_0^0, y_1^0 \cdots y_{N_0}^0)$;
-set $j = 0$;
-<u>Repeat</u> <u>until</u> ($\alpha_j = 1$ or all attempts fail);
  -choose the next value, α_{j+1};
  -choose the initial mesh for the next problem,$x_0^{j+1}, x_1^{j+1} \cdots x_{N_{j+1}}^{j+1}$;
    (Usually this mesh will be equal to or a refinement of $x_0^j, x_1^j \cdots x_{N_j}^j$)
  -choose an initial guess for the solution at $x_0^{j+1}, x_1^{j+1} \cdots x_{N_{j+1}}^{j+1}$;
    This will involve referring to $y_0^j, y_1^j \cdots y_{N_j}^j$.
  -invoke BV solver on problem determined by α_{j+1} with initial mesh and corresponding initial guess $y_0^{j+1}, y_1^{j+1} \cdots y_{N_{j+1}}^{j+1}$;
  -<u>if</u> (BV solver was successfull) <u>then</u>
    -set $j = j + 1$;
  -<u>else</u>
    -consider reducing α_{j+1} for the next attempted step;
-<u>end</u> <u>Repeat</u>

- **Automatic parameter Continuation in Multiple Shooting**: Consider applying a multiple shooting method to the problem,

$$y' = f(x, y, \lambda, \alpha),$$

where we want to solve the problems corresponding to α_0 and $\alpha_j = \alpha_{j-1} + \Delta\alpha_j,\ j = 1, 2 \cdots k$ (where the $\Delta\alpha_j$ are known). After solving the j^{th} problem, $(\underline{s}_{0,j}, \underline{s}_{1,j} \cdots \underline{s}_{P-1,j}, \lambda_j)$, the method generates accurate starting values for the $(j+1)^{st}$ problem. Let $\underline{s}_{i,j+1}^0 = \underline{s}_{i,j} + \Delta\underline{s}_{i,j}$, and $\lambda_{j+1}^0 = \lambda_j + \Delta\lambda$, then we want to solve $G(\alpha_{j+1}, \underline{z}_{j+1}) = 0$ given that $G(\alpha_j, \underline{z}_j) = 0$ and $\alpha_{j+1} = \alpha_j + \Delta\alpha$. That is, find $z_{j+1} = z_j + \Delta z$ the solution of $G(\alpha_{j+1}, \underline{z}_{j+1}) = 0$. There are two alternatives: The first is to set $\underline{z}_{j+1}^0 = \underline{z}_j$ and iterate using Newtons method. The second alternative is based on an expansion of $G(\alpha, \underline{z})$ as a Taylor series in two variables.

$$\begin{aligned} G(\alpha_{j+1}, \underline{z}_{j+1}) &= G(\alpha_j + (\Delta\alpha), \underline{z}_j + \Delta\underline{z}) \\ &= G(\alpha_j, \underline{z}_j) + \frac{\partial G}{\partial \alpha}(\Delta\alpha) + \frac{\partial G}{\partial z}(\Delta z) \end{aligned}$$

Therefore (Δz) is determined from the solution of,

$$\frac{\partial G}{\partial z}(\Delta z) = -(\Delta\alpha)\frac{\partial G}{\partial \alpha},$$

and the initial guess is $\underline{z}_{j+1}^0 = \underline{z}_j + (\Delta z)$. To compute $\frac{\partial G}{\partial \alpha}|_{\alpha_j}$ we add yet another initial value sytem which need only be approximated on the last iteration of the j^{th} problem. This extra vector IVP has the form,

$$w' = \frac{\partial f}{\partial y} w + \frac{\partial f}{\partial \alpha}, \quad w(a) = 0.$$

This follows since,

$$\frac{\partial g_i}{\partial \alpha} = \begin{cases} 0 & \text{for } i = 0 \\ \frac{\partial y_{i-1}}{\partial \alpha}|_{x=x_i} & \text{for } i = 1, 2 \cdots (P-1) \\ B\frac{\partial y_{P-1}}{\partial \alpha}|_{x=x_P} & \text{for } i = P \end{cases}$$

Letting $u_i(x) \equiv \frac{\partial}{\partial \alpha} y_{i-1}(x, \lambda, \alpha)$ we note that u_i satifies the IVP:

$$u_i' = \frac{\partial f}{\partial y} u_i + \frac{\partial f}{\partial \alpha}, \quad u_i(x_{i-1}) = \frac{\partial}{\partial \alpha} \underline{s}_{i-1} = 0.$$

Therefore we have the desired (Δz) satisfies,

$$\frac{\partial G}{\partial z}|_{\underline{z}_j} (\Delta z) = -(\Delta \alpha_j)[0, u_1(x_1), u_2(x_2) \cdots u_{P-1}(x_{P-1}), B u_P(x_P)]^T.$$

The JPL shooting method implements this form of automatic parameter continuation. Note that the code has already solved a system of linear equations of this form (with a different RHS) in the last Newton iteration of the j^{th} problem.

3.3 Collocation based BVP Methods

Collocation methods are a particular instance of a broader class of Expansion Methods which have their roots in approximation theory and have been used extensively in developing numerical methods for PDEs. These methods are often considered inefficient and/or unstable for ODEs, but collocation for BVPs is an exception.

1. The key idea for any expansion method:–Choose a basis set of known, well-behaved, $h_i(x)$, and approximate the solution, $y(x)$, by a linear combination of the $h_i(x)$,

   $$y(x) \approx v(x) \equiv \sum_{k=1}^{K} a_k h_k(x),$$

   where the a_i's are chosen so that $v(x)$ will satisfy the boundary conditions and 'almost satisfy' the Differential Equation for all $x \in [a, b]$.

   Note if $y(x) \in \Re^n$ an expansion will be associated with each component and K may be different for different components. That is, for $r = 1, 2 \cdots n$,

   $$y_r(x) \approx v_r(x) \equiv \sum_{i=1}^{K_r} a_{r,i} h_i(x).$$

2. For <u>Collocation</u> the unknown coefficients, $\{a_{r,i}\}$, $r = 1, 2 \cdots n$: $i = 1, 2 \cdots K_r$, are defined by two sets of constraints (assume for simplicity that $K_r = K$ for $r = 1, 2 \cdots n$).

   - Boundary conditions – n constraints.
   - Collocation conditions – the defect of the ODE is zero at a prescribed set of $K - 1$ 'collocation' points, $\{x_j\}_{j=1}^{K-1}$. That is, each component of $v(x)$ satisfies the ODE exactly at $K - 1$ points,

$$v'(x_j) = \begin{bmatrix} v_1'(x_j) \\ v_2'(x_j) \\ \vdots \\ v_n'(x_j) \end{bmatrix} = \begin{bmatrix} f_1(x_j, v(x_j)) \\ f_2(x_j, v(x_j)) \\ \vdots \\ f_n(x_j, v(x_j)) \end{bmatrix}, \text{ for } j = 1, 2 \cdots K - 1.$$

     This results in $(K - 1)n$ equations in the Kn unknowns.
   - If we are not unlucky these equations will have a unique solution, defining $v(x)$, and as $K \to \infty$ the corresponding $v(x) \to y(x)$ for all $x \in [a, b]$. Such convergence results will rely on theorems from approximation theory and will make assumptions on the choice of $h_i(x)$ and the smoothness of $y(x)$. (Jackson Theorems)
   - Collocation can cope with multipoint boundary conditions without additional difficulties.
   - Collocation can handle higher order equations,

$$y^{(m)} = f(x, y, y' \cdots y^{(m-1)})$$

     directly provided:

     (a) $v(x)$ is constrained to be in $C^{m-1}[a, b]$.

     (b) The appropriate number of boundary conditions are specified to determine a unique solution.

3. **COLSYS/COLNEW – A collocation code:**
   We will now consider implementation details and the class of BVP problems that can be effectively handled by this approach using the COLSYS/COLNEW software package as an example. This package was introduced in a sequence of three papers by Ascher, Christiansen and Russell (ACM TOMS 7,2,(1981), pp. 209-229: Math. Comp, 33, (1979), pp. 659-679: and Springer Lecture Notes in Comp. Sc., 76, (1979)). The text also discusses the code at length and presents the method in detail in the appendix. We will adopt the notation of the text in this section (it is not the same as the notation used in the first few chapters of these notes although the transition should not be distracting).

   Consider a mixed-order system of d differential equations of orders $m_1 \leq m_2 \cdots \leq m_d$,

$$y_j^{(m_j)} = F_j(x, z(y)), \text{ for } j = 1, 2 \cdots d,$$

where $y(x) = (y_1(x), y_2(x) \cdots y_d(x))^T$ is the solution and

$$z(y) = (y_1, y_1' \cdots y_1^{(m_1-1)}, y_2, y_2' \cdots y_2^{(m_2-1)} \cdots y_d, y_d' \cdots y_d^{(m_d-1)})^T$$

is the vector of unknowns that would result after converting this mixed-order system into an equivalent first-order system. To be well-posed this problem requires $m^* = \sum_{j=1}^{d} m_j$ multipoint boundary conditions. We will assume these are separated and given in the form:

$$g_j(t_j, z(y(t_j))) = 0 \ \text{ for } \ j = 1, 2 \cdots m^*.$$

Note $z(y) \in \Re^{m^*}$ and therefore $g_j : \Re \times \Re^{m^*} \to \Re$.

Let $a = x_0 < x_1 \cdots < x_N = b$ define a mesh, Π, on $[a, b]$. In COLSYS the $h_i(x)$ define a basis for the space of piecewise polynomials. Let

$$P_{l,\Pi} = \{S(x) \mid S \text{ is a piecewise polynomial of degree less than } l \text{ on } \Pi\}.$$

The approximation determined by COLSYS is a vector valued function, $v = (v_1, v_2 \cdots v_d)^T$ such that

$$v_j \in P_{k+m_j,\Pi} \cap C^{(m_j-1)}[a, b] \ \text{ for } \ j = 1, 2 \cdots d,$$

with $k \geq m_d$ being the number of collocation points per subinterval, (x_{i-1}, x_i). The collocation points are $x_{i-1} + \alpha_r(x_i - x_{i-1}),\ r = 1, 2 \cdots k$, where the α_r's are the Gauss points defined as the zeros of the appropriate Gauss-Legendre polynomial.

Now the v_j's are determined by requiring,

Collocation: Nkd constraints,

$$v_j^{(m_j)}(x_{i-1}+\alpha_r(x_i-x_{i-1})) = F_j(x_{i-1}+\alpha_r(x_i-x_{i-1}), z(v(x_{i-1}+\alpha_r(x_i-x_{i-1})))),$$

for $i = 1, 2 \cdots N;\ r = 1, 2 \cdots k;\ j = 1, 2 \cdots d.$

Continuity: $(N-1)m^*$ constraints (linear),

$$v_j \in C^{(m_j-1)}[a, b], \ \text{ for } \ j = 1, 2 \cdots d.$$

Boundary Conditions: m^* constraints,

$$g_j(t_j, z(v(t_j))) = 0, \ \text{ for } \ j = 1, 2 \cdots m^*.$$

This gives a total of $Nm^* + Nkd$ constraints, while the number of unknowns $= N\sum_{j=1}^{d}(k + m_j) = N(kd + m^*)$. (Note if the original system had been reduced to the equivalent first order system, $z \in \Re^{m*}$, the number of unknowns would be $N(km^* + m*)$ as a separate polynomial for each derivative term would be required.)

Key Theoretical Result: One can show, using results from approximation theory, the following two properties which guarantee the convergence and accuracy of a Collocation BVP method that uses Gauss points as collocation points (and COLSYS in particular).

- If the underlying mathematical problem has a sufficiently smooth isolated solution, then for given Π and $k \geq m_d,\ v$, the solution to the above nonlinear system of equations, exists and Newtons Method converges quadratically provided the initial guess is 'close enough' to $y(x)$ and Π is fine enough.
- If $h = \max_{i=1,2\cdots N}(x_i - x_{i-1})$ then we have,

  $$|y_j^{(l)}(x) - v_j^{(l)}(x)| = O(h^{k+m_j-l}) \ \text{for} \ l = 1, 2\cdots m_j;\ j = 1, 2\cdots d,$$

  for any $x \in [a, b]$ and at the x_i's we observe superconvergence,

  $$|y_j^{(l)}(x_i) - v_j^{(l)}(x_i)| = O(h^{2k}) \ \text{for} \ l = 0, 1\cdots(m_j - 1).$$

4. **COLSYS/COLNEW – implementation considerations:**

   There are several strategies that are critical to the performance (efficiency and reliability) of a collocation method. We will consider five of these strategies and discuss how they have been implemented in COLSYS/COLNEW. These strategies are,

   - The choice of the Basis functions for the space, $P_{k+m_j,\Pi}$, (the $h_i(x)$). COLSYS uses a B-spline basis as robust packages existed and could be directly adopted. The more recent version, COLNEW, uses a monomial basis (for $x \in (x_{j-1}, x_j), h_i(x) = (x - x_{j-1})_+^{i-1}$).
   - The Linear System Solver to be used to solve the linear system associated with each iteration of the modified Newton method. It must exploit the special structure that arises because of the choice of basis set and the special form of the constraints.
   - The use of a reliable error estimate, $est_j^l(x) \approx e_j^l(x)$, where

     $$e_j^l(x) = y_j^l(x) - v_j^l(x) = O(h^{k+m_j-l}).$$

   - The Mesh Selection strategy whose goal is to equidistribute the error.
   - The Modified Newton Iteration to be used.

   Note that each strategy is critical but we will focus on the latter four which can be somewhat messy to analyse and justify.

   (a) **The Linear System Solver:**
   Each constraint involves at most the unknowns associated with the polynomials on adjacent sub-intervals, (x_{i-1}, x_i) and (x_i, x_{i+1}) (ie., $2(dk + m^*)$ unknowns). If these constraints are ordered carefully the resulting Newton Iteration Matrix of partial derivatives will be almost block-diagonal (for the monomial or B-spline basis) with,

      i. The number of unknowns $= N(dk + m^*)$.

ii. The size of the blocks $= dk + m^*$,

iii. The routine SOLVEBLOCK , a standard B-spline code, is used directly.

iv. If the problem were to be converted to a first order equivalent system, the vector z would have m^* components and the degree of each polynomial would be at most k. Therefore the number of unknowns would be $(k+1)m^*N$ and the size of the blocks would be $(k+1)m^*$.

(b) **Error Estimates:**

An error estimate is required for mesh selection and accuracy control. When $k > m_d$ one can show that for $x \in [x_{i-1}, x_i)$,

$$e_j^{(l)}(x) \equiv y_j^{(l)}(x) - v_j^{(l)}(x) = \frac{y^{(k+m_j)}(x_{i-1})}{2^{k+m_j-l}} \rho_k^{(k-m_j+l)}\left(\frac{2}{h_i}(x - x_{i-1/2})\right) h_i^{k+m_j-l} + O(h^{k+m_j+1-l}),$$

for $l = 0, 1 \cdots (m_j - 1)$; $j = 1, 2 \cdots d$, where $h_i = (x_i - x_{i-1})$; $h = \max_{i=1}^{N} h_i$; and

$$\rho_k(w) = \frac{(w^2-1)^k}{(2k)!}.$$

Note that $y^{(k+m_j)}(x_{i-1})$ can be approximated by $v^{(k+m_j)}(x_{i-1})$. This may not be very accurate but may suffice for mesh selection. Also note the local nature of the dominant error term as $h_i \to 0$.

Mesh halving is used to provide a more reliable estimate of $e_j^l(x)$. Let the two meshes be $\{x_i\}_{i=0}^{N}$, $\{\hat{x}_i\}_{i=0}^{2N}$ with $\hat{x}_{2i} = x_i$, and the corresponding two numerical approximations: $v(x) = [v_1(x), v_2(x) \cdots v_d(x)]^T$ and $\hat{v}(x) = [\hat{v}_1(x), \hat{v}_2(x) \cdots \hat{v}_d(x)]^T$. The more reliable error estimate is based on two 'representative' (or sampled) values of $\hat{v}(x) - v(x)$ per subinterval. If we let Δ_1 and Δ_2 represent the magnitude of the different components of these representative values, we have for fixed values of j, l and i, in the range $1 \le j \le d$; $0 \le l \le (m_j - 1)$; $1 \le i \le N$,

$$\Delta_1 \equiv |v_j^{(l)}(x_{i+1/6}) - \hat{v}_j^{(l)}(x_{i+1/6})|,$$

and

$$\Delta_2 \equiv |v_j^{(l)}(x_{i+1/3}) - \hat{v}_j^{(l)}(x_{i+1/3})|,$$

Then, from the above expression for the error term, one can precompute weights $w_{k,r}$ $k = 1, 2 \cdots 7$; $r = 0, 1 \cdots (k-1)$ such that

$$\max_{x \in [x_i, x_{i+1}]} |y_j^{(l)}(x) - v_j^{(l)}(x)| \approx w_{k,k+l-m_j}(\Delta_1 + \Delta_2).$$

In COLSYS the $w_{k,r}$ are defined by,

$$w_{k,r} = \frac{\|\rho_k^{(r)}\|}{|2^{2k-r}\rho_k^{(r)}(-2/3) - \rho_k^{(r)}(-1/3)| + |2^{2k-r}\rho_k^{(r)}(-1/3) - \rho_k^{(r)}(1/3)|},$$

where $\|\rho^{(k)}_{(r)}\|$ is a bound on $\rho^{(k)}_{(r)}(u)$ for $u \in [-1,1]$. These values are independent of the problem and the h_i . This estimate is quite reliable even when the mesh is non-uniform (in which case the $O(h^{k+m_j+1-l})$ term may be dominant).

With this reliable error estimate, a component-wise error control strategy is used. The user specifies error tolerances for some (or all) of the components of $z(v)$ and the method attempts to ensure that,

$$|z(y(x))_{\nu_r} - z(v(x))_{\nu_r}| \leq Tol_r + |z(v(x))_{\nu_r}|Tol_r.$$

(a mixed absolute/relative control). The user must specify $NTOL$, $\{Tol_r\}_{r=1}^{NTOL}$, and the vector $\{\nu_r\}_{r=1}^{NTOL}$. The solution $\hat{v}(x)$ is accepted if for each subinterval, $i = 1, 2 \cdots N$, we have,

$$est_{\nu_r} = w_{k,k+l-m_j}(\Delta_1+\Delta_2) < Tol_r+|z(\hat{v}(x_i))_{\nu_r}|Tol_r, \text{ for } r = 1, 2 \cdots NTOL.$$

(c) **Mesh Selection:**
The goal of mesh selection is twofold:

- Distribute the error equally – so that on each interval, $i = 1, 2 \cdots N$ we have comparable values for,

$$max_{r=1}^{NTOL} \frac{est_{\nu_r,i}}{(Tol_r(1 + |v^l_j(x_i)|))}.$$

- Minimize the work required to generate an acceptable solution. That is, find the smallest N^* such that for the mesh $a = x_0 < x_1 \cdots < x_{N^*} = b$, the error estimate is acceptable on each interval.

To accomplish this we recall,

$$|e^l_j| \approx c_{k,k-m_j+l}|y^{k+m_j}_j(x_{i-1})|h^{k+m_j-l}_i, \text{ for } i = 1, 2 \cdots N,$$

where the $c_{k,k-m_j+l}$ are computable constants stored in COLSYS, and the error criteria is equivalent to,

$$\frac{|est_{\nu_r,i}|}{(Tol_r(1 + |v^l_j(x_i)|))} \leq 1, \text{ for } i = 1, 2 \cdots N.$$

Furthermore, if the error is equally distributed and N is close to optimal then,

$$max_{r=1}^{NTOL} \left\{ \frac{|est_{\nu_r,i}|}{(Tol_r(1 + |v^l_j(x_i)|))} \right\} \approx 1 \text{ for } i = 1, 2 \cdots N.$$

Now let

$$s_r(x) = \frac{c_{k,k-m_j+l}}{(Tol_r(1 + |v^l_j(x_i)|))}|y^{k+m_j}_j(x)|,$$

and we then have:

$$max_{r=1}^{NTOL}\left\{\frac{|est_{\nu_r,i}|}{(Tol_r(1+|v_j^l(x_i)|))}\right\}\approx 1 \quad\Leftrightarrow\quad max_{r=1}^{NTOL}\left\{s_r(x_{i-1})h_i^{k+m_j-l}\right\}\approx 1$$

$$\Leftrightarrow\quad max_{r=1}^{NTOL}\left\{[s_r(x_{i-1})]^{1/(k+m_j-l)}h_i\right\}\approx 1.$$

Therefore if we define,

$$s(x)\equiv max_{r=1}^{NTOL}\left\{[s_r(x_{i-1})]^{1/(k+m_j-l)}\right\},$$

the mesh $\{x_i^*\}_{i=0}^{N^*}$ will be close to optimal if

$$s(x_i^*)h_i^*\approx 1 \ \text{ for } \ i=1,2\cdots N^*.$$

After solving for $z(v)$ on a given mesh, COLSYS will consider redistribition of the mesh on the next iteration by approximating $s(x)$ with a piecewise constant function, $\hat{s}(x)$, defined for $x\in[x_{i-1},x_i)$ by:

$$\hat{s}(x)=max_{r=1}^{NTOL}\left\{\left[\frac{c_{k.k-m_j+l}}{(Tol_r(1+|v_j^l(x_i)|))}\hat{v}_j^{(k+m_j)}\right]^{1/(k+m_j-l)}\right\},$$

where

$$\hat{v}_j^{(k+m_j)}=\frac{\hat{v}_j^{(k+m_j-1)}(x_{i+1+1/2})\ -\ \hat{v}_j^{(k+m_j-1)}(x_{i+1/2})}{\frac{1}{2}(x_{i+2}-x_i)}\approx\hat{v}_j^{k+m_j}(x_{i+1}).$$

The new mesh is chosen by first determining N^* and then determining the $\{x_i^*\}_{i=0}^{N^*}$ (in order) by satisfying,

$$\int_{x_{i-1}^*}^{x_i^*}\hat{s}(x)dx=\frac{1}{N^*}\sum_{m=1}^{N}\hat{s}(x_m)h_m, \ \text{ for } \ i=1,2\cdots(N^*-1).$$

Note that this last equation follows since $\hat{s}(x)$ is piecewise constant and

$$\int_a^b\hat{s}(x)dx=\sum_{m=1}^{N}\hat{s}(x_m)h_m.$$

An overview of the Mesh Selection Scheme for COLSYS/COLNEW is presented below:

-Repeat until (stopping criteria satisfied or $N \geq \bar{N}$);
  -attempt to solve the problem on the current mesh;
  -if (iteration scheme converged) then
    -if (mesh obtained by halving and both converged then
      -check stopping criteria and exit if OK;
    -endif
    -Consider Mesh refinement as Follows:
      -set $r_1 = max_{i=1}^{N}\{h_i \hat{s}(x_i)\}$;
      -set $r_2 = \sum_{i=1}^{N} h_i \hat{s}(x_i)$;
      -set $r_3 = \frac{r_2}{N}$;
      -if $(r_1 < 2r_2)$ then
        -halve the current mesh $(N^* = 2N)$ as the error distribution is OK;
      -else
        -set $N^* = min\{\bar{N}/2, N, max(N, r_2)/2\}$;
        -determine $\{x_i^*\}_{i=1}^{N^*}$ using
        $\int_{x_{i-1}^*}^{x_i^*} \hat{s}(x)dx = \frac{1}{N^*} \sum_{m=1}^{N} \hat{s}(x_m)h_m$,
        -set $N = N^*$ and current mesh $= \{x_i^*\}$;
      -endif;
  -else
    -halve the current mesh $(N = 2N)$;
  -endif;
-endRepeat;

(d) **The Modified Newton Iteration:**
The solution of the nonlinear equations is based on a modified Newton iteration with quasilinearization. Recall the original problem,

$$y_j^{(m_j)} = F_j(x, z(y)), \quad \text{for } j = 1, 2 \cdots d,$$

with boundary conditions,

$$g_s(t_s, z(y(t_s))) = 0, \quad s = 1, 2 \cdots m^*.$$

The corresponding Linearized BVP is given by:

$$w_j^{(m_j)} = \sum_{r=1}^{m^*} \frac{\partial F_j(x, z(v))}{\partial z_r} z(w) + [F_j(x, z(v)) - v^{m_j}],$$

with boundary conditions,

$$\beta_s w = \beta_s(v) w = \sum_{r=1}^{m^*} \frac{\partial g_s(t_s, z(v(t_s)))}{\partial z_r} z(w(t_s)),$$

where $v(x)$ is the current approximation to the solution of our nonlinear problem (ie., the current Newton iterate). On each Newton iteration the Newton correction, w^L is determined by solving this linearized problem with $v(x) = v^{L-1}(x)$, and the next iterate is defined by,

$$v^L(x) = v^{L-1}(x) + \lambda_L w^L,$$

where the 'damping factor', λ_L, satisfies $.01 \leq \lambda \leq 1$, and is chosen dynamically to improve convergence (a line search strategy). This is a 'Damped Newton' strategy combined with quasilinearization.

3.4 Finite Difference BVP Methods

Basic Idea: Replace any derivative in the ODE by a finite difference approximation of the appropriate order and then solve the resulting (usually large) nonlinear difference equation. For example, consider the second order system,

$$y'' = f(x, y, y'), \text{ for } x \in [a, b],$$

with non-separated boundary conditions,

$$g((y(a), y(b)) = 0, \quad g : \Re^n \times \Re^n \rightarrow \Re^{2n}.$$

If we introduce a uniform mesh $h = \frac{b-a}{N}$, $x_i = a + ih$, for $i = 0, 1 \cdots N$, then using central differences to replace (ie, approximate) derivatives, we obtain the following difference equation:

$$\frac{(y_{i+1} - 2y_i + y_{i-1})}{h^2} = f(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}), \quad 1 \leq i \leq N-1,$$

$$g_1(y_0, y_N) = g_2(y_0, y_N) = 0.$$

This discrete nonlinear system of equations to determine, $y_0, y_1 \cdots y_N \in \Re^{(N+1)n}$, can be solved by a modified Newton iteration to compute the approximations $y_i \approx y(x_i)$.

3.4.1 Background and Notation for Differences:

- Divided Differences (nonuniform mesh):

$$\begin{aligned} y[x_0] &\equiv y(x_0), \\ y[x_0 x_1 \cdots x_k] &\equiv \frac{y[x_1 x_2 \cdots x_k] - y[x_0 x_1 \cdots x_{k-1}]}{x_k - x_0}. \end{aligned}$$

  One can show:

$$y[x_0 x_1 \cdots x_k] = \frac{y^{(k)}(\eta)}{k!},$$

  for some η in the closed interval containing $x_0, x_1 \cdots x_k$.

- Differences (uniform mesh):

  Forward differences:

$$\begin{aligned}\Delta^1 f(x) &\equiv f(x+h) - f(x),\\ \Delta^{r+1} f(x) &\equiv \Delta^r f(x+h) - \Delta^r f(x).\end{aligned}$$

  for $r \geq 1$. Note that $\Delta^r f(x_k) = r!h^r f[x_k x_{k+1} \cdots x_{k+r}]$.

  Backward differences:

$$\begin{aligned}\nabla^1 f(x) &\equiv f(x) - f(x-h),\\ \nabla^{r+1} f(x) &\equiv \nabla^r f(x) - \nabla^r f(x-h),\end{aligned}$$

  for $r \geq 1$, and one can show,$\nabla^r f(x_k) = r!h^r f[x_k x_{k-1} \cdots x_{k-r}]$.

  Central differences:

$$\begin{aligned}\delta^1 f(x) &\equiv f(x+h/2) - f(x-h/2),\\ \delta^{r+1} f(x) &\equiv \delta^r f(x+h/2) - \delta^r f(x-h/2).\end{aligned}$$

  For $m \geq 1$ we have

$$\begin{aligned}\delta^{2m+1} f(x_k + h/2) &= h^{2m+1}(2m+1)! f[x_{k-m} x_{k-m+1} \cdots x_{k+m+1}],\\ \delta^{2m+1} f(x_k - h/2) &= h^{2m+1}(2m+1)! f[x_{k-m-1} x_{k-m} \cdots x_{k+m}],\\ \delta^{2m} f(x_k) &= h^{2m}(2m)! f[x_{k-m} x_{k-m+1} \cdots x_{k+m}].\end{aligned}$$

3.4.2 Convergence Results for Finite Difference Methods

Consider the first order system,

$$y' = f(x,y), \quad \text{on} \quad [a,b],$$

with boundary conditions,

$$g(y(a), y(b)) = 0,$$

where $y(x) \in \Re^n$; $g : \Re^n \times \Re^n \to \Re^n$.

- Keller has studied the following difference scheme applied to this BVP,

$$\frac{1}{h_i}\nabla y_i - f(x_{i-1/2}, \frac{y_i + y_{i-1}}{2}), \quad \text{for} \quad i = 1, 2 \cdots N,$$

$$g(y_0, y_N) = 0,$$

  where $\nabla y_i \equiv y_i - y_{i-1}$.

  As for IVPs, the Local Truncation Error (LTE) associated with this scheme for the i^{th} interval, d_i, is:

$$d_i = \frac{1}{h_i}\nabla y(x_i) - f(x_{i-1/2}, \frac{y(x_i) + y(x_{i-1})}{2}).$$

- In addition to LTE, other concepts from the analysis of methods for IVPs are used to investigate the convergence and efficiency of Finite Difference methods for BVPs:

  1. A difference scheme is of <u>order p</u> if the corresponding LTE $d_i = O(h_i^{p+1})$ for $i = 1, 2 \cdots N$.
  2. A difference scheme is <u>consistent</u> if it is of order p, where $p \geq 1$ ($p \geq m$ for an m^{th} order ODE).
  3. A difference scheme is <u>stable</u> if for all $\underline{y} = \{y_i\}_{i=0}^{N}$ and $\underline{z} = \{z_i\}_{i=0}^{N}$ and for some norm, $|||\underline{y}||| = max_{i=0}^{N}\|y_i\|$, there exists $K > 0$ such that for sufficiently small h,
  $$|||\underline{y} - \underline{z}||| \leq K|||d(\underline{y}) - d(\underline{z})|||,$$
  where,
  $$d(\underline{y}) = \begin{bmatrix} g(y_0, y_N) \\ d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix}.$$

- **Key Result:**
  A finite difference scheme will converge to the solution of this BVP (as $h \rightarrow 0$) if the ODE is Lipschitz continuous and the difference scheme is stable and consistent. Furthermore Newtons method will converge when applied to the nonlinear equations associated with the difference scheme provided h is small enough and the initial guess is sufficiently accurate.

- **Extrapolation:**
  Extrapolation can be used to increase the order of a finite difference method. If one can show that the LTE satisfies an expansion:
  $$d(y(x_0), y(x_1) \cdots y(x_N)) \equiv \begin{bmatrix} g(y(x_0), y(x_N)) \\ d_1(y(x_0), y(x_1) \cdots y(x_N)) \\ d_2(y(x_0), y(x_1) \cdots y(x_N)) \\ \vdots \\ d_N(y(x_0), y(x_1) \cdots y(x_N)) \end{bmatrix},$$
  where ,
  $$d_i(y(x_0), y(x_1) \cdots y(x_N)) = \sum_{r=1}^{k} h^{m_r} T_r[y(x_i)] + O(h^{m_{k+1}}),$$
  for $1 \leq i \leq N$, then on a uniform mesh the global error will satisfy,
  $$y_i - y(x_i) = \sum_{r=1}^{k} h^{m_r} e_r(x_i) + O(h^{m_{k+1}}),$$

where the $e_r(x)$ are solutions of linear BVPs whose coefficients depend on $y(x)$. Examples of schemes with such an error expansion are the 'Box Scheme' and the Trapezoidal rule. In these cases $m_r = 2r, r = 1, 2 \cdots (k+1)$.

Fom this expansion of the global error we see that extrapolation is justified. That is, after computing a sequence of approximations with successively refined meshes one can determine a suitable linear combination of these approximations that has higher order.

- **Deferred Correction:**
  An alternative (to extrapolation) which is based on the same expansion of the LTE, but can be implemented in a more effective way, is the approach of deferred correction. The basic idea of this approach is to first approximate the coefficient of h^{m_1} in the expansion of the LTE, $T_1[y(x_i)]$, by some 'difference operator', $T_{1,i}[y_0 \cdots y_N]$ satisfying,
  $$h^{m_1}\|T_1[y(x_i)] - T_{1,i}[y_0 \cdots y_N]\| \leq M_1 h^{m_2}.$$
  And then solve for the 'corrected' approximation, y_i^1, of order m_2 by solving the nearby difference equation,
  $$\begin{aligned} d_0(y_0^1, y_N^1) &= 0, \\ d_i(y_0^1, y_1^1 \cdots y_N^1) &= h^{m_1} T_{1,i}[y_0 \cdots y_N] \text{ for } i = 1, 2 \cdots N. \end{aligned}$$

  Note that we then have,
  $$\begin{aligned} d_i(y_0, y_1 \cdots y_N) &= 0, \\ d_i(y_0^1, y_1^1 \cdots y_N^1) &= h^{m_1} T_{1,i}[y_0, y_1 \cdots y_N], \end{aligned}$$
  and
  $$d_i(y(x_0), y(x_1) \cdots y(x_N)) = h^{m_1} T_1(x_i) + O(h^{m_2}),$$
  which implies,
  $$d_i(y(x_0), y(x_1) \cdots y(x_N)) = h^{m_1} T_{1,i}[y_0, y_1 \cdots y_N] + O(h^{m_2}),$$
  for $i = 1, 2 \cdots N$.

  From the definition of stability this implies,
  $$y(x_i) - y_i^1 = O(h^{m_2}).$$
  One can show that y_i^1 satisfy a similar error expansion to that for y_i and therefore the same process can be repeated to obtain y_i^2 (two steps of deferred correction) which is accurate to $O(h^{m_3})$.

  Note also,

1. This approach can be extended to nonuniform meshes.
2. Care must be taken when computing $T_{1,i}$ for i near 0 or i near N (otherwise 'centered' formulas can be used).
3. Often one can hold $\frac{\partial d}{\partial y}$ fixed (ie., W^l) at the value used to obtain convergence to the original discrete problem with the solution $\{y_i\}_{i=0}^N$, or at most do one additional Newton iterate with the residual redefined

3.4.3 Other Implementation Issues:

Consider the first order system,

$$y' = f(x, y), \quad \text{for } x \in [a, b],$$

with boundary conditions, $g(y(a), y(b)) = 0$. Recall, to approximate $y(x)$ on a given mesh, $\{x_i\}_{i=0}^N$, by a finite difference method we first introduce,

$$d(\underline{y}) = \begin{bmatrix} d_0(y_0, y_1 \cdots y_N; h) \\ d_1(y_0, y_1 \cdots y_N; h) \\ \vdots \\ d_N(y_0, y_1 \cdots y_N; h) \end{bmatrix},$$

and solve for $\underline{y} = (y_0, y_1 \cdots y_N)^T$ such that $d(\underline{y}) = 0$, where $d_0(\underline{y}) = g(y_0, y_N)$, and $d_i(\underline{y})$ for $i = 1, 2 \cdots N$ is a difference approximation to the ODE associated with $[x_{i-1}, x_i]$.

Examples:

1. The Box scheme (also called the Midpoint or centered Euler scheme) is a second order scheme defined by,

$$d_{i+1}(\underline{y}) = \frac{y_{i+1} - y_i}{h} - f(x_{i+1/2}, \frac{y_i + y_{i+1}}{2}) \approx [y' - f(x, y)]|_{x=x_{i+1/2}}.$$

2. The Trapezoidal rule is a second order scheme defined by

$$d_{i+1}(\underline{y}) = \frac{y_{i+1} - y_i}{h} - \frac{1}{2}[f(x_i, y_i) + f(x_{i+1}, y_{i+1})].$$

Note that, in solving for the discrete approximations using Newtons's method, the iteration matrix, $\frac{\partial d}{\partial y}$, will be block bi-diagonal for the above schemes since $\frac{\partial d_{i+1}}{\partial y_j} = 0$ except for $j = i + 1$, and $j = i$, for $i = 0, 1 \cdots (N - 1)$. That is, the iteration matrix, W, used in solving for the Newton step,

$$W\delta^{(l)} = -d(\underline{y}^{(l)}),$$

is,

$$W = \begin{bmatrix} A & 0 & 0 & \cdots & 0 & B \\ C_1 & D_1 & 0 & \cdots & 0 & 0 \\ 0 & C_2 & D_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & C_N & D_N \end{bmatrix},$$

where $A = \frac{\partial g}{\partial y_0}$, $B = \frac{\partial g}{\partial y_N}$ and,

$$C_i = \frac{\partial d_i}{\partial y_{i-1}} = \begin{cases} -\frac{1}{h}I - \frac{1}{2}\frac{\partial f}{\partial y}|_{(x_{i-1/2}, y_{i-1/2})}, & \text{for the Box Scheme} \\ -\frac{1}{h}I - \frac{1}{2}\frac{\partial f}{\partial y}|_{(x_{i-1}, y_{i-1})}, & \text{for the Trapezoidal rule} \end{cases}$$

and

$$D_i = \frac{\partial d_i}{\partial y_i} = \begin{cases} \frac{1}{h}I - \frac{1}{2}\frac{\partial f}{\partial y}|_{(x_{i-1/2}, y_{i-1/2})}, & \text{for the Box Scheme} \\ \frac{1}{h}I - \frac{1}{2}\frac{\partial f}{\partial y}|_{(x_i, y_i)}, & \text{for the Trapezoidal rule} \end{cases}$$

for $i = 1, 2 \cdots N$.

- In general we have, with a low order difference method, the bandwidth will be small for the block matrix $\frac{\partial d}{\partial y}$ but a small h will be necessary to obtain the specified accuracy. This leads to large linear systems which are sparse and banded. On the other hand, high order difference methods result in larger bandwidth systems (width $\approx p$, for p^{th} order) but larger h is possible. That is, we have smaller linear sytems with less sructure.
- For the solution of the nonlinear system, the iteration matrix, W is always block banded with $n \times n$ blocks and the total size of the system $(N+1)n \times (N+1)n$.
- The block banded structure may be disturbed because differences for intervals near the beginning or near the end may require special approximations. This is especially true if higher order differences are used or if nonuniform meshes used.
- An Overview of the Newton Iteration:

  -Given an initial approximation $\{y_i^{(0)}\}_{i=0}^N$;
  <u>for</u> $l = 0, 1 \cdots$ <u>until</u> convergence or <u>trouble</u> do:
  -Compute the residual,

$$d(y^{(l)}) = \begin{bmatrix} d_0(y_0^{(l)} y_1^{(l)} \cdots y_N^{(l)}) \\ d_1(y_0^{(l)} y_1^{(l)} \cdots y_N^{(l)}) \\ \vdots \\ d_N(y_0^{(l)} y_1^{(l)} \cdots y_N^{(l)}) \end{bmatrix};$$

  -Determine the Correction $\delta^{(l+1)}$ by solving:

$$W^l \delta^{(l+1)} = -d(y^{(l)},$$

  where $W^l \approx \frac{\partial d}{\partial y}|_{y^{(l)}}$;
  -Set $y^{(l+1)} = y^{(l)} + \delta^{(l+1)}$;
  <u>end</u>

- Note that with such an implementation:
  1. Quadratic convergence is observed if W^l is recomputed and factored on each iteration.
  2. In practice a difference method will often hold W^l constant as long as the residuals are decreasing in norm rapidly enough (linear convergence at best).
  3. The cost of evaluating W^l and its $L-U$ decomposition is $\approx [Npn^3 + Npn^2]$ flops plus N evaluations of the Jacobian matrix, $\frac{\partial f}{\partial y}$.
  4. After the basic finite difference solution is obtained, one can apply one or two steps of deferred correction to increase the accuracy on the same coarse mesh.
  5. There is no obvious piecewise polynomial to provide an approximation at off-mesh points.

3.5 Runge-Kutta Methods for BVPs

3.5.1 The Basic Approach

Consider the BVP,

$$y' = f(x, y), \text{ with separated boundary conditions,}$$

$$g_1(y(a)) = 0, \; g_2(y(b)) = 0.$$

For a given partitioning, $a = x_0 < x_1 \cdots < x_N = b$ consider, on each subinterval (x_{i-1}, x_i), introducing the residual $\Phi_i(y_{i-1}, y_i)$ defined in terms of how well y_i satisfies a particular Runge-Kutta formula assiciated with y_{i-1} and (x_{i-1}, x_i). That is, for a given RK formula applied to this ODE we have,

$$\hat{y}_i = y_{i-1} + h \sum_{j=1}^{s} w_j k_j = z_i(x_i) + O(h^{p+1}),$$

and the associated RK residual is defined by,

$$\Phi_i \equiv y_i - \hat{y}_i = y_i - y_{i-1} - (x_i - x_{i-1}) \sum_{j=1}^{s} w_j k_j.$$

- At convergence, $\Phi_i = 0$ for $i = 1, 2 \cdots N$, we will have $y_i = \hat{y}_i$ for all i. (That is, the discrete solution $\{x_i, y_i\}_{i=1}^{N}$ is a Runge-Kutta solution of the corresponding IVP with initial value, y_0.)
- The corresponding nonlinear set of equations is:

$$G(\underline{z}) = 0, \text{ where } \underline{z} = (y_0, y_1 \cdots y_N)^T,$$

and

$$G(\underline{z}) = \begin{bmatrix} g_1(y_0) \\ \Phi_1(y_0, y_1) \\ \Phi_2(y_1, y_2) \\ \vdots \\ \Phi_N(y_{N-1}, y_N) \\ g_2(y_N) \end{bmatrix}.$$

The corresponding Newton iteration matrix is:

$$\frac{\partial G}{\partial \underline{z}} = \begin{bmatrix} B_0 & 0 & 0 & \cdots & 0 & 0 \\ L_1 & R_1 & 0 & \cdots & 0 & 0 \\ 0 & L_2 & R_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & L_N & R_N \\ 0 & 0 & 0 & \cdots & 0 & B_1 \end{bmatrix},$$

where B_0, B_1 are associated with the boundary conditions and $L_i = \frac{\partial \Phi_i}{\partial y_{i-1}}$, $R_i = \frac{\partial \Phi_i}{\partial y_i} = I$ are each $n \times n$ matrices.

1. Note that,
$$L_i = \frac{\partial \Phi_i}{\partial y_{i-1}} = -I - h_i \sum_{j=1}^{s} w_j \frac{\partial k_j}{\partial y_{i-1}},$$
and since $k_j = f(x_{i-1}+\alpha_j h_i, y_{i-1}+h_i \sum_{r=1}^{s} \beta_{jr} k_r)$, the $\frac{\partial k_j}{\partial y_{i-1}}$ satisfy the 'matrix equation',
$$\frac{\partial k_j}{\partial y_{i-1}} = \frac{\partial f}{\partial y}\left[I + h_i \sum_{r=1}^{s} \beta_{jr} \frac{\partial k_r}{\partial y_{i-1}}\right], \quad \text{for } j = 1, 2 \cdots s.$$
2. In practice we can hold $\frac{\partial f}{\partial y}$, in this equation fixed; solve for the resulting matrices, $\frac{\partial k_j}{\partial y_{i-1}}$; and then determine L_i from (1). (A true Newton iteration would require that $\frac{\partial f}{\partial y}$ be evaluated at $(x_{i-1} + \alpha_j h_i, Y_j^l)$ for $j = 1, 2 \cdots s; l = 0, 1 \cdots$.)
3. Considerable simplification of this approach is possible if the RK formula is explicit (although such formulas are known to have poor stability properties).

- The structure and size of the Newton iteration matrix is independent of the order (unlike finite difference and collocation methods). As the order p increases the L_i become more expensive to compute (what about the conditioning?).
- The stability can be good (suitable) if the RK formula is implicit (particularly for problems with boundary layers – as we will see).
- Note that collocation formulas can be considered equivalent to a specific choice of an IRK formula and therefore this interpretation leads to a different implementation for collocation.

3.5.2 Mono-Implicit RK Formulas

- Consider the class of IRK formulas that are 'implicit' only in $\hat{y}_i$,

$$\hat{y}_i = y_{i-1} + h_i \sum_{j=1}^{s} w_j k_j,$$

where the corresponding RK tableau has $\alpha_s = 1, \beta_{sj} = w_j$ for $j = 1, 2 \cdots s$ and satisfies,

$$\begin{array}{c|cccccc}
0 & 0 & 0 & \dots & 0 & 0 \\
\alpha_2 & \beta_{21} & 0 & \dots & 0 & \beta_{2s} \\
\vdots & \vdots & \vdots & \dots & \vdots & \vdots \\
\alpha_{s-1} & \beta_{s-11} & \beta_{s-12} & \dots & 0 & \beta_{s-1s} \\
1 & w_1 & w_2 & \dots & w_{s-1} & w_s \\
\hline
 & w_1 & w_2 & \dots & w_{s-1} & w_s
\end{array}$$

(that is, the tableau is strictly lower triangular except for the last row and column). Note that with this structure, $k_s = f(x_i, \hat{y}_i)$ and the corresponding residual becomes (after replacing k_s by its 'converged' value, $f(x_i, y_i)$),

$$\Phi_i = y_i - y_{i-1} - h_i \sum_{j=1}^{s-1} w_j k_j - h_i w_s f(x_i, y_i),$$

$$k_j = f(x_{i-1} + \alpha_j h_i, y_{i-1} + h_i \sum_{r=1}^{j-1} \beta_{jr} k_r + h_i \beta_{js} f(x_i, y_i)).$$

- With these formulas the L_i and R_i can be determined as explicit calculations (low order polynomials in $\frac{\partial f}{\partial y}$) although R_i is no longer $= I$.

- With this degree of implicitness one can derive formulas with good stability properties. For example A-Stable formulas which are symmetric. That is, when applied to $y' = \lambda y$ yield,

$$y_i = R(h\lambda) y_{i-1} = \frac{P_s(h\lambda)}{P_s(-h\lambda)} y_{i-1},$$

and are of order $s + 1$.

- One can also consider the derivation of the extended class of formulas:

$$k_j = f(x_{i-1} + \alpha_j h_i, (1 - \nu_j) y_{i-1} + \nu_j y_i + h_i \sum_{r=1}^{j-1} \beta_{jr} k_r),$$

with the Continuous Extension,

$$u(x) = u(x_{i-1} + \tau h_i) = y_{i-1} + h_i \sum_{j=1}^{\bar{s}} b_j(\tau) k_j.$$

- One BV method based on these formulas of order $4-8$ is MIRKDC, developed at Toronto and available from NETLIB.

- **A Numerical Example**
  Consider the test problem, 'swirling flow III',

$$\epsilon f'''' + ff''' + gg' = 0,$$

$$\epsilon g'' + fg' - f'g = 0,$$

with $f(0) = f(1) = f'(0) = f'(1) = 0, \;\; g(0) = 1, g(1) = -1$ and parameter values $\epsilon = 1.0, .1, .05, .01, .005, .001, .0005$.

To report the performance of a method on this problem we tabulate (in the following tables), for $TOL = 10^{-6}$ and $TOL = 10^{-8}$, the statistics,

Time (in seconds)

Error (maximum observed, sampling $u(x)$ over 10 samples per step)

Defect (maximum magnitude of the defect, sampled over each step)

Number of Nonlinear Equations (reported as a sequence, $[(N_1q_1), (N_2q_2) \cdots (N_kq_k)]^T$, where k is the number of mesh refinements and (N_rq_r) indicates that on mesh refinement 'r' there were N_r meshpoints and q_r iterations).

$\epsilon \Rightarrow$	1.0	0.1	0.05	0.01	0.005	0.001	0.0005
Profile	$(5,2)$	$(5,2)$	$(5,3)$	$(5,*)$	$(5,*)$	$(5,*)(160,1)$	$(5,*)(110,1)$
	$(10,1)$	$(10,1)$	$(10,1)$	$(10,5)$	$(10,6)$	$(10,9)(320,1)$	$(10,*)(220,1)$
		$(20,1)$	$(20,1)$	$(20,1)$	$(10,1)$	$(10,2)$	$(20,*)(440,1)$
		$(40,1)$	$(40,1)$	$(40,1)$	$(20,1)$	$(20,2)$	$(40,10)$
			$(80,1)$	$(80,1)$	$(40,1)$	$(20,1)$	$(40,1)$
				$(58,1)$	$(80,1)$	$(40,1)$	$(40,1)$
				$(116,1)$	$(71,1)$	$(40,1)$	$(80,1)$
					$(142,1)$	$(80,1)$	$(160,1)$
Time(sec)	0.078	0.313	0.645	1.415	1.697	2.963	6.869
Error	$8.3 \cdot 10^{-7}$	$2.3 \cdot 10^{-7}$	$6.2 \cdot 10^{-8}$	$1.6 \cdot 10^{-7}$	$1.8 \cdot 10^{-7}$	$1.2 \cdot 10^{-7}$	$2.4 \cdot 10^{-8}$
Defect	$1.0 \cdot 10^{-4}$	$2.1 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$	$9.4 \cdot 10^{-4}$	$2.0 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	$2.1 \cdot 10^{-3}$

Table 1: COLNEW on Problem 1: 4^{th} order, tol $= 10^{-6}$

$\epsilon \Rightarrow$	1.0	0.1	0.05	0.01	0.005	0.001	0.0005
$tol \Rightarrow$	$6 \cdot 10^{-7}$	$8 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$2 \cdot 10^{-6}$	$3 \cdot 10^{-6}$	$4 \cdot 10^{-6}$	$3 \cdot 10^{-7}$
Profile	$(5,1)$	$(5,1)$	$(5,1)$	$(5,2)$	$(5,*)$	$(5,*)$	$(5,5)$
	$(16,1)$	$(20,1)$	$(20,1)$	$(20,2)$	$(10,2)$	$(10,4)$	$(20,*)$
	$(21,1)$	$(29,1)$	$(45,1)$	$(80,1)$	$(40,1)$	$(40,*)$	$(40,5)$
			$52,1)$	$(111,1)$	$(104,1)$	$(80,4)$	$(160,1)$
					$(125,1)$	$(153,1)$	$(323,1)$
						$(175,1)$	$(373,1)$
						$(192,1)$	
Time(sec)	0.059	0.078	0.169	0.358	0.468	1.468	1.667
Error	$3.7 \cdot 10^{-7}$	$9.6 \cdot 10^{-7}$	$7.6 \cdot 10^{-7}$	$7.2 \cdot 10^{-7}$	$8.9 \cdot 10^{-7}$	$8.4 \cdot 10^{-7}$	$8.0 \cdot 10^{-7}$
Defect	$3.5 \cdot 10^{-7}$	$4.3 \cdot 10^{-6}$	$1.5 \cdot 10^{-6}$	$3.0 \cdot 10^{-6}$	$4.5 \cdot 10^{-6}$	$9.7 \cdot 10^{-6}$	$2.0 \cdot 10^{-6}$

Table 2: MIRKDC on Problem 1: 4^{th} order, error $= 10^{-6}$

$\epsilon \Rightarrow$	1.0	0.1	0.05	0.01	0.005	0.001	0.0005
Profile	$(5,2)$	$(5,3)$	$(5,3)$	$(5,*)$	$(5,*)$	$(5,*)(320,1)$	$(5,*)(320,1)$
	$(10,1)$	$(10,1)$	$(10,1)$	$(10,6)$	$(10,7)$	$(10,9)(640,1)$	$(10,*)(640,1)$
	$(20,1)$	$(20,1)$	$(20,1)$	$(20,1)$	$(10,1)$	$(10,2)(325,1)$	$(20,*)$
	$(40,1)$	$(40,1)$	$(40,1)$	$(40,1)$	$(20,1)$	$(20,2)(650,1)$	$(40,10)$
		$(80,1)$	$(80,1)$	$(80,1)$	$(40,1)$	$(20,1)$	$(40,2)$
		$(160,1)$	$(160,1)$	$(80,1)$	$(80,1)$	$(40,1)$	$(40,2)$
				$(160,1)$	$(80,1)$	$(40,1)$	$(80,2)$
				$(320,1)$	$(160,1)$	$(80,1)$	$(160,1)$
				$(320,1)$	$(320,1)$	$(160,1)$	$(160,1)$
Time(sec)	0.28	1.13	1.15	3.00	3.13	9.73	9.26
Error	$2.5 \cdot 10^{-9}$	$8.6 \cdot 10^{-10}$	$4.1 \cdot 10^{-9}$	$2.2 \cdot 10^{-9}$	$6.3 \cdot 10^{-9}$	$1.0 \cdot 10^{-9}$	$5.4 \cdot 10^{-9}$
Defect	$1.8 \cdot 10^{-6}$	$3.4 \cdot 10^{-6}$	$1.9 \cdot 10^{-5}$	$4.6 \cdot 10^{-5}$	$1.8 \cdot 10^{-4}$	$2.3 \cdot 10^{-4}$	$6.8 \cdot 10^{-4}$

Table 3: COLNEW on Problem 1: 4^{th} order, tol $= 10^{-8}$

3.5.3 A Runge-Kutta BV method for a PSE

In the Problem Solving Environment (PSE) provided by MATLAB a built-in numerical BVP method, bvp4c, is provided. It is based on the same Runge-Kutta formula

$\epsilon \Rightarrow$	1.0	0.1	0.05	0.01	0.005	0.001	0.0005
$tol \Rightarrow$	$7 \cdot 10^{-9}$	$8 \cdot 10^{-8}$	$1 \cdot 10^{-8}$	$3 \cdot 10^{-8}$	$3 \cdot 10^{-8}$	$3 \cdot 10^{-8}$	$3 \cdot 10^{-9}$
Profile	$(5,1)$	$(5,1)$	$(5,1)$	$(5,3)$	$(5,5)$	$(5,*)$	$(5,5)$
	$(20,1)$	$(20,1)$	$(20,1)$	$(20,2)$	$(10,2)$	$(10,4)$	$(20,*)$
	$(52,1)$	$(71,1)$	$(80,1)$	$(80,1)$	$(40,1)$	$(40,*)$	$(40,5)$
		$(92,1)$	$(168,1)$	$(256,1)$	$(160,1)$	$(80,4)$	$(160,1)$
				$(323,1)$	$(341,1)$	$(320,1)$	$(640,1)$
					$(397,1)$	$(536,1)$	$(1071,1)$
						$(595,1)$	$(1186,1)$
Time(sec)	0.106	0.259	0.393	1.000	1.390	2.821	4.852
Error	$1.0 \cdot 10^{-8}$	$9.5 \cdot 10^{-9}$	$7.0 \cdot 10^{-9}$	$9.9 \cdot 10^{-9}$	$9.1 \cdot 10^{-9}$	$9.0 \cdot 10^{-9}$	$9.1 \cdot 10^{-9}$
Defect	$9.3 \cdot 10^{-9}$	$5.3 \cdot 10^{-8}$	$1.3 \cdot 10^{-8}$	$2.7 \cdot 10^{-8}$	$2.4 \cdot 10^{-8}$	$2.8 \cdot 10^{-8}$	$3.4 \cdot 10^{-9}$

Table 4: MIRKDC on Problem 1: 4^{th} order, error $= 10^{-8}$

$\epsilon \Rightarrow$	1.0	0.1	0.05	0.01	0.005	0.001	0.0005
Profile	$(5,2)$	$(5,3)$	$(5,3)$	$(5,*)$	$(5,*)$	$(5,*)$	$(5,*)$
	$(10,1)$	$(10,1)$	$(10,1)$	$(10,5)$	$(10,5)$	$(10,9)$	$(10,11)$
			$(20,1)$	$(20,1)$	$(20,1)$	$(10,1)$	$(10,1)$
				$(40,1)$	$(40,1)$	$(20,1)$	$(20,1)$
					$(80,1)$	$(20,1)$	$(20,1)$
						$(40,1)$	$(40,1)$
						$(80,1)$	$(80,1)$
Time(sec)	0.178	0.183	0.410	1.053	1.875	2.557	2.646
Error	$2.2 \cdot 10^{-10}$	$9.6 \cdot 10^{-8}$	$1.1 \cdot 10^{-8}$	$3.6 \cdot 10^{-8}$	$5.0 \cdot 10^{-9}$	$2.3 \cdot 10^{-8}$	$1.3 \cdot 10^{-7}$
Defect	$4.3 \cdot 10^{-8}$	$1.3 \cdot 10^{-5}$	$6.4 \cdot 10^{-6}$	$9.4 \cdot 10^{-5}$	$3.8 \cdot 10^{-5}$	$4.1 \cdot 10^{-4}$	$2.7 \cdot 10^{-3}$

Table 5: COLNEW on Problem 1: 6^{th} order, tol $= 10^{-6}$

as MIRKDC except that the error control and continuous extension are different. The issues arising in implementing a BVP solver in a PSE are different than those that are traditionally considered in a general purpose scientific computing environment (where efficiency and accuracy are the major concerns). We will present a discussion and justification of these issues and how they influenced the design and implementation of bvp4c.

$\epsilon \Rightarrow$	1.0	0.1	0.05	0.01	0.005	0.001	0.0005
$tol \Rightarrow$	$1 \cdot 10^{-4}$	$6 \cdot 10^{-5}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-5}$	$9 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$7 \cdot 10^{-6}$
Profile	$(5,1)$	$(5,1)$	$(5,1)$	$(5,2)$	$(5,3)$	$(5,14)$	$(5,11)$
		$(10,1)$	$(12,1)$	$(20,1)$	$(20,*)$	$(20,*)$	$(20,*)$
		$(12,1)$	$(15,1)$	$(38,1)$	$(40,3)$	$(40,4)$	$(40,5)$
				$(42,1)$	$(49,1)$	$(74,1)$	$(82,1)$
					$(53,1)$	$(82,1)$	$(90,1)$
						$(90,1)$	
Time(sec)	0.015	0.074	0.090	0.310	0.933	1.232	1.118
Error	$6.1 \cdot 10^{-7}$	$7.0 \cdot 10^{-7}$	$9.5 \cdot 10^{-7}$	$5.7 \cdot 10^{-7}$	$6.6 \cdot 10^{-7}$	$4.9 \cdot 10^{-7}$	$9.4 \cdot 10^{-7}$
Defect	$4.5 \cdot 10^{-6}$	$1.1 \cdot 10^{-5}$	$2.6 \cdot 10^{-5}$	$2.6 \cdot 10^{-6}$	$1.8 \cdot 10^{-5}$	$3.7 \cdot 10^{-6}$	$8.1 \cdot 10^{-6}$

Table 6: MIRKDC on Problem 1: 6^{th} order, error $= 10^{-6}$

$\epsilon \Rightarrow$	1.0	0.1	0.05	0.01	0.005	0.001	0.0005
Profile	$(5,2)$	$(5,3)$	$(5,3)$	$(5,*)$	$(5,*)$	$(5,*)$	$(5,*)$
	$(10,1)$	$(10,1)$	$(10,1)$	$(10,6)$	$(10,6)$	$(10,10)$	$(10,11)$
		$(20,1)$	$(20,1)$	$(20,1)$	$(20,1)$	$(10,1)$	$(10,*)$
			$(40,1)$	$(40,1)$	$(40,1)$	$(20,1)$	$(20,3)$
				$(80,1)$	$(80,1)$	$(20,1)$	$(20,1)$
				$(160,1)$	$(160,1)$	$(40,1)$	$(40,1)$
						$(80,1)$	$(80,1)$
						$(160,1)$	$(160,1)$
Time(sec)	0.169	0.348	0.778	1.955	3.589	4.212	4.702
Error	$2.2 \cdot 10^{-10}$	$1.4 \cdot 10^{-9}$	$1.6 \cdot 10^{-10}$	$5.0 \cdot 10^{-10}$	$8.7 \cdot 10^{-11}$	$5.2 \cdot 10^{-10}$	$2.5 \cdot 10^{-9}$
Defect	$4.3 \cdot 10^{-8}$	$4.8 \cdot 10^{-7}$	$2.3 \cdot 10^{-7}$	$3.5 \cdot 10^{-6}$	$1.3 \cdot 10^{-6}$	$1.5 \cdot 10^{-5}$	$1.0 \cdot 10^{-4}$

Table 7: COLNEW on Problem 1: 6^{th} order, tol $= 10^{-8}$

[For a detailed discussion and justification of the implementation issues see J. Kierzenka and L. Shampine, 'A BVP Solver based on residual control and the MATLAB PSE', ACM TOMS, 2001.]

- Special features/considerations relevant to a PSE:

$\epsilon \Rightarrow$	1.0	0.1	0.05	0.01	0.005	0.001	0.0005
$tol \Rightarrow$	$5 \cdot 10^{-7}$	$8 \cdot 10^{-7}$	$6 \cdot 10^{-7}$	$1 \cdot 10^{-7}$	$1 \cdot 10^{-7}$	$7 \cdot 10^{-8}$	$7 \cdot 10^{-8}$
Profile	$(5,1)$	$(5,1)$	$(5,1)$	$(5,3)$	$(5,3)$	$(5,14)$	$(5,11)$
	$(9,1)$	$(18,1)$	$(20,1)$	$(20,1)$	$(20,*)$	$(20,*)$	$(20,*)$
		$(23,1)$	$(32,1)$	$(72,1)$	$(40,3)$	$(40,4)$	$(40,5)$
				$(87,1)$	$(92,1)$	$(135,1)$	$(150,1)$
					$(104,1)$	$(162,1)$	$(183,1)$
						$(178,1)$	$(201,1)$
Time(sec)	0.039	0.125	0.161	0.541	1.178	1.886	2.160
Error	$6.9 \cdot 10^{-9}$	$7.2 \cdot 10^{-9}$	$8.6 \cdot 10^{-9}$	$8.5 \cdot 10^{-9}$	$8.6 \cdot 10^{-9}$	$8.1 \cdot 10^{-9}$	$8.8 \cdot 10^{-9}$
Defect	$8.4 \cdot 10^{-8}$	$2.1 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$	$3.6 \cdot 10^{-8}$	$3.6 \cdot 10^{-8}$	$2.0 \cdot 10^{-8}$	$3.6 \cdot 10^{-8}$

Table 8: MIRKDC on Problem 1: 6^{th} order, error $= 10^{-8}$

1. The focus is on visualizing the approximate solution and will usually require the graphical presentation of the results. For example standard x/y plots or phase plane plots are typically used to present or display functions that depend on only one variable (such as the solutions of BVPs).
2. Stringent accuracy is not likely required, although off-mesh accuracy may well be required.
3. The overhead of working in a PSE may be expensive.
4. The interface/documentaion must be simple if the software is ever to be used. (In particular the method should hide unecessary information.)
5. Users working in a PSE often have a class of problems to be solved, where the the 'class of problems' are characterized by a parameter (or vector of parameters) which define the mathematical model.
6. Defaults should be available for all the options (of the method) which require some knowledge of the problem or method.
7. The focus is not on efficiency but it is on 'ease of use' and 'robustness'.
8. The user is not expected to define the workspace but the code will fail when all available workspace is used and convergence to acceptable numerical solution has not been acheived.
9. The solution is returned as an abstract data structure and can be used directly in continuation. The numerical solution is a piecewise polynomial and the representaion is in terms of the Runge-Kutta stages associated with each step.

- The Class of Problems Considered:

$$y' = f(x, y, p), \quad x \in [a, b],$$

with boundary conditions,

$$g(y(a), y(b), p) = 0,$$

$y(x) \in \Re^n$, $p \in \Re^k$. If there are no parameters p can be ignored (hidden).

- The underlying RK formula is given by the tableau,

$$\begin{array}{r|ccc} 0 & 0 & 0 & 0 \\ 1/2 & 5/24 & 1/3 & \text{-}1/24 \\ 1 & 1/6 & 2/3 & 1/6 \\ \hline & 1/6 & 2/3 & 1/6 \end{array}$$

The continuous extension, $S(x)$, delivered by bvp4c, is defined for $x \in [x_{i-1}, x_i]$ to be the Hermite interpolant, $u_i(x)$, that satisfies the equations,

$$\begin{aligned} u_i(x_{i-1}) &= y_{i-1}, \\ u_i'(x_{i-1}) &= y_{i-1}' \ \ (= k_1), \\ u_i'(x_i) &= y_i' \ \ (= k_3 = f(x_i, y_i) \text{ at convergence}), \\ u_i'(x_{i-1} + h_i/2) &= k_2. \end{aligned}$$

This method is equivalent to a collocation method with the collocation points corresponding to $\tau = 0, 1/2, 1$.

- The error control and mesh selection strategies are based on direct control of the size of the defect associated with each subinterval. $S(x)$ satisfies,

$$\begin{aligned} \delta(x) &\equiv (S'(x) - f(x, S(x)) \\ &= S'(x) - y'(x) + O(h^4) \end{aligned}$$

(the error in the approximation of $y'(x)$ by $S'(x)$).

Shampine and Kiezenka show that, with this definition of $S(x)$ we have,

$$\|\delta(x)\|_i = C_1 h_i^3 \|y^{(4)}(x_{i-1/2})\| + O(h^4),$$

where $h = \max_{j=1}^N h_j$. Note that the size of this defect is dominated by the 'local' term and that it depends only on the solution (and not the differential equation). C_1 depends on $(x - x_{i-1})/h_i$ only and has its maximum value at $\theta = (3 \pm \sqrt{3})/6$. One sampled defect value is therefore used to give an asymptotically correct estimate of the maximumum magnitude defect on each subinterval. This value is used both for mesh redistribution and error control.

Note that in MIRKDC a different (more expensive) interpolant is used for the same 4^{th} order formula. The corresponding defect is $O(h_i^4)$ and is also used for mesh selection and error control. This is possible since one can easily show that if $\hat{u}_i(x) = z_i(x) + O(h_i^5)$ then the corresponding piecewise polynomial, $\hat{S}(x)$, satisfies,

$$\begin{aligned}\hat{\delta}(x) &\equiv (\hat{S}'(x) - f(x, \hat{S}(x)) \\ &= \hat{S}'(x) - z_i'(x) + O(h_i^4),\end{aligned}$$

and the $\hat{S}(x)$ implemented in MIRKDC satisfies this condition. With this more accurate interpolant one has the potential of using fewer mesh intervals (for the same accuracy) but it is more difficult to derive an inexpensive bound on the magnitude of the defect associated with each subinterval.

3.6 Special difficulties

3.6.1 Convergence for nonlinear problems

Convergence difficulties often arise on nonlinear problems, particularly when the initial mesh is coarse and/or not adapted to the behaviour of the solution. Different methods attempt to address this difficulty by adopting (or adapting) ideas from the literature on nonlinear equations,

Accurate initial guesses Multigrid ideas can be used to start the iteration on a 'refined' mesh and continuation can be used for the initial mesh.

Stabilization of the iteration Marquardt stabilization, Damped Newton or the use of 'trust regions' can improve the chances of convergence from an initial guess that has a large associated residual.

Efficiency of the iteration In solving the linear system that arises on each iteration, $W^l \delta^l = -G(y^{l-1})$, one can use an approximation to W^l to reduce the overall cost. W^l can be held constant or a rank one update used. If the latter approach is adopted, W^{l+1} is computed as a rank one correction to W^l such that,

$$W^{l+1}(y^l - y^{l-1}) = W^{l+1}\delta^l = G(y^l) - G(y^{l-1}).$$

3.6.2 Singular Perturbation Problems

This class of problems has been widely studied in the literature and arise in applications where there are different time scales associated with the underlying mathematical model. The numerical difficulty is related to stiff IVPs but it is more subtle and the 'remedy' is not as straightforward. We will first consider a simple linear example. Let $\epsilon > 0$ be a small parameter and the BVP be defined by,

$$\begin{aligned}\epsilon y' &= A_{11}y + A_{12}z + f_1, \\ z' &= A_{21}y + A_{22}z + f_2,\end{aligned}$$

for $0 \le x \le 1$, with boundary conditions,

$$B_0 \begin{bmatrix} y(0) \\ z(0) \end{bmatrix} + B_1 \begin{bmatrix} y(1) \\ z(1) \end{bmatrix} = \underline{\beta},$$

where $y(x) \in \Re^n, z(x) \in \Re^m$, and $A_{11}, A_{12}, A_{21}, A_{22}, B_0$ and B_1 are constant matrices of the appropriate dimensions.

Note that, when written as a standard linear first order system of ODEs, with $u = (y(x) \;\; z(x))^T$, we have,

$$u' = \begin{bmatrix} \frac{A_{11}}{\epsilon} & \frac{A_{12}}{\epsilon} \\ A_{21} & A_{22} \end{bmatrix} u + \bar{f} \equiv Au + \bar{f}, \text{ where } \bar{f} = \begin{bmatrix} \frac{f_1}{\epsilon} \\ f_2 \end{bmatrix},$$

and as $\epsilon \to 0$ some eigenvalues of A must approach the eigenvalues of $\frac{A_{11}}{\epsilon}$. (This follows from the 'block' extension of Gershgoran's therorem.)

Let these eigenvalues be $\lambda_1/\epsilon, \lambda_2/\epsilon \cdots \lambda_n/\epsilon$. We then have that some of the components of the homogeneous ODE, $u' = Au$, must grow (or decay) like $e^{(x\lambda_r)/\epsilon}$. In this case, for small values of ϵ, superposition will fail as it must represent a complete basis for the space of homogeneous solutions. Similarly multiple shooting will have difficulty since the variational equation, $Y' = AY; \;\; Y(0) = I$ will have rapidly increasing and/or decaying components, and the columns of $Y(x)$ (which are of the form $\sum_{r=1}^{n} \alpha_r e^{(x\lambda_r)/\epsilon}$) will either overflow (if $\mathrm{Re}(\lambda_r) > 0$) or become numerically linearly dependent and result in a singular matrix.

Note that, for this model problem to be mathematically well-posed, the boundary conditions must be such that,

1. For $Re(\lambda_r) < 0$, the corresponding decaying component, $e^{(x\lambda_r)/\epsilon}$, is determined by a boundary condition at $x = 0$.

2. For $Re(\lambda_r) > 0$, the corresponding increasing component, $e^{(x\lambda_r)/\epsilon}$, is determined by a boundary condition at $x = 1$.

One consequence of this observation is that if A_{11} has r eigenvalues in the left half plane and s eigenvalues in the right half plane, then $\mathrm{rank}(B_0) \ge r$, $\mathrm{rank}(B_1) \ge s$ for a well-posed problem.

To gain more insight and an understanding of this difficulty, consider the scalar IVP,

$$\epsilon y' = \lambda y + q(x), \text{ for } 0 \le x \le 1,$$

$$y(0) = y_0.$$

If $q(x) = q(x, \epsilon)$ has an expansion in ϵ,

$$q(x, \epsilon) = \sum_{\nu=0}^{\infty} \epsilon^{\nu} q_{\nu}(x) \quad (= q_0(x) + O(\epsilon)),$$

then the exact solution is,

$$\begin{aligned} y(x) &= y_0 e^{\frac{\lambda x}{\epsilon}} + \frac{1}{\epsilon}\int_0^x e^{\frac{\lambda(x-s)}{\epsilon}} q(s)ds \\ &= [y_0 + \frac{1}{\lambda} q_0(x)] e^{\frac{\lambda x}{\epsilon}} - \frac{1}{\lambda} q_0(x) + O(\epsilon). \end{aligned}$$

Therefore if $Re(\lambda) < 0$ the solution, for small ϵ, is composed of a <u>smooth component</u>,

$$y_R(x) = -\frac{1}{\lambda} q_0(x)$$

($y_R(x)$ is the solution to this scalar problem obtained by setting $\epsilon = 0$) and a <u>fast component</u>, $y_F(x)$ which 'connects' the initial value y_0 to $y_R(x)$ in a small boundary layer (or transition layer), near $x = 0$,

$$y_F(x) = [y_0 + \frac{1}{\lambda} q_0(x)] e^{\frac{\lambda x}{\epsilon}}.$$

Note that there will be no boundary layer if $y_0 = -\frac{1}{\lambda} q_0(0)$ ($= y_R(0)$). We know that for any $\epsilon > 0$ and any choice of y_0, this IVP has a unique stable solution but when $\epsilon = 0$ there is only one solution, $y_R(x)$ (with the only consistent initial value, $y_0 = y_R(0)$). Numerical difficulties that arise as ϵ approaches zero should not be surprising.

If $Re(\lambda) > 0$ then $y(x)$ is unstable (as a function of y_0) since, from the above expression for the exact solution we see that one component of $y(x)$ behaves like $y_0 e^{\frac{\lambda x}{\epsilon}}$. On the other hand, we can express $y(x)$ as a function of $y(1)$,

$$\begin{aligned} y(x) &= y(1) e^{\frac{-\lambda(1-x)}{\epsilon}} + \frac{1}{\epsilon}\int_x^1 e^{\frac{-\lambda(x-s)}{\epsilon}} q(s)ds \\ &= y_R(x) + [y(1) + \frac{1}{\lambda} q_0(x)] e^{\frac{-\lambda(1-x)}{\epsilon}} + O(\epsilon). \end{aligned}$$

This implies stability (as a function of $y(1)$) with a boundary layer near $x = 1$. Similarly $Re(\lambda) < 0$ and $y(1)$ specified will result in an ill-posed problem.

Now consider our system of equations $u' = Au + \bar{f}$ and recall that for small ϵ, some of the eigenvalues of A must converge to $\lambda_1/\epsilon, \lambda_2/\epsilon \cdots \lambda_n/\epsilon$ where $\lambda_1, \lambda_2 \cdots \lambda_n$ are the eigenvalues of A_{11}. For $Re(\lambda_i) < 0$ we have components of $u = (y \;\; z)^T$ behaving like,

$$\mu_i(x) = \underbrace{[\mu_{i0} + \frac{1}{\lambda_i} q_{i0}(x)] e^{\frac{\lambda_i x}{\epsilon}}}_{Fast} - \underbrace{\frac{1}{\lambda_i} q_{i0}(x)}_{Slow} + O(\epsilon).$$

Similarly for $Re(\lambda_i) > 0$, we have components of $u = (y \;\; z)^T$ behaving like,

$$\nu_i(x) = \underbrace{[\nu_i(1) + \frac{1}{\lambda_i} q_{i0}(x)] e^{\frac{-\lambda_i(1-x)}{\epsilon}}}_{Fast} - \underbrace{\frac{1}{\lambda_i} q_{i0}(x)}_{Slow} + O(\epsilon).$$

We then see that this problem will be well-posed only if,

1. The number of eigenvalues of A_{11} in the left half plane equals the number of 'Fast' components in the boundary layer near $x = 0$ and this must be $\leq \text{rank}(B_0)$.

2. The number of eigenvalues of A_{11} in the right half plane equals the number of 'Fast' components in the boundary layer near $x = 1$ and this must be $\leq \text{rank}(B_1)$.

This analysis gives some insight into the mesh selection strategy that must be used in a numerical method that is able to effectively solve this class of problems. In particular in the boundary layer(s), where $e^{\frac{\lambda x}{\epsilon}}$ (or $e^{\frac{-\lambda(1-x)}{\epsilon}}$) is significant, the method must accurately track the fast components and therefore $|\frac{h\lambda}{\epsilon}|$must be small. In addition outside the boundary layer(s), where both $e^{\frac{\lambda x}{\epsilon}}$ and $e^{\frac{-\lambda(1-x)}{\epsilon}}$ are insignificant, the mesh size (the h_j's) should be determined by the behaviour of the smooth components, $y_R(x)$.

Consider a collocation or Runge-Kutta method. Components of the form $e^{\frac{x\lambda}{\epsilon}}$ are represented by powers of $R(\frac{h\lambda}{\epsilon})$ where

$$R(z) = P(z)/Q(z),$$

is the stability function of the RK or collocation formula. Outside the boundary layer, for $Re(\lambda) \leq 0$, the method will remain stable for all $h > 0$ if $|R(\frac{h\lambda}{\epsilon})| \leq 1$. Also components of the form $e^{\frac{-\lambda(1-x)}{\epsilon}}$ are represented by powers of $R^{-1}(\frac{-h\lambda}{\epsilon})$ and therefore outside the boundary layer, for $Re(\lambda) > 0$, the method will remain stable for all $h > 0$ if $|R^{-1}(\frac{-h\lambda}{\epsilon})| \leq 1$.

There are two approaches that are used to accomplish this:

Kreiss/Ringhoffer Choose a strongly stable RK or collocation formula with degree $P <$ degree Q and a second complementary method with degree degree $P >$ degree Q. Uncouple the ODE on each step and apply the first method to those components with the corresponding eigenvalues in the left half plane and the second method to those components whose eigenvalues are in the right half plane.

Ascher/Weiss Choose a symmetric method (with $P(z) = Q(-z)$) with all the zeros of $P(z)$ in the left half plane (hence the zeros of $Q(-z)$ are all in the right half plane). In this case $|R(z)| = 1$ on the imaginary axis and this implies A-stability of the underlying formula with $|R(z)| \leq 1$ for $z \in LHP$ and $|R^{-1}(-z)| \leq 1$ for $z \in RHP$. This method can then be used for all λ_i. Of course h must be small in the boundary layer(s) to accurately resolve the fast components.