PHYS-4007/5007: Computational Physics
Course Lecture Notes
Section VIII

Dr. Donald G. Luttermoser
East Tennessee State University

Version 7.0

Abstract

These class notes are designed for use of the instructor and students of the course PHYS-4007/5007: Computational Physics taught by Dr. Donald Luttermoser at East Tennessee State University.

VIII. Numerical Solution of Ordinary Differential Equations (ODE)

A. Introduction

1. Many important problems in engineering and the physical sciences require the determination of a function satisfying an equation containing derivatives of the unknown function. Perhaps the most familiar example is Newton's Second Law of Motion:

$$m\frac{d^2r(t)}{dt^2} = F\left[t, r(t), \frac{dr(t)}{dt}\right] . \qquad \text{(VIII-1)}$$

a) The position $r(t)$ of a particle of mass m is acted upon by a force F, which may be a function of time t, position $r(t)$, and the velocity $dr(t)/dt$.

b) To determine the motion of the particle acted upon by a given force F it is necessary to find a function r satisfying Eq. (VIII-1).

c) It also is important to set up a coordinate system first with respect to the motion of the mass before setting up the equations to be solved.

d) Since the force due to gravity is pointing in the opposite direction of r, we get

$$m\frac{d^2r(t)}{dt^2} = -mg . \qquad \text{(VIII-2)}$$

Integrating Eq. (VIII-2) we have

$$\frac{dr(t)}{dt} = -gt + c_1 , \qquad \text{(VIII-3)}$$

$$r(t) = -\frac{1}{2}gt^2 + c_1 t + c_2 , \qquad \text{(VIII-4)}$$

where c_1 and c_2 are constants of integration.

e) To determine $r(t)$ completely, it is necessary to specify two additional conditions, such as the position and velocity of the particle of some instant of time $\Longrightarrow$ these **initial conditions** then give the value of c_1 and c_2. These are often referred to as time-dependent problems.

f) There are some problems where it is more convenient to give conditions at the boundaries of the integration path $\Longrightarrow$ **boundary conditions**. These are often referred to as time-independent problems.

g) In order to completely solve a differential equation, one needs either initial or boundary conditions.

2. If a differential equation (DE) depends on a *single* independent variable and only an *ordinary* derivative appears in the DE, then such a DE is called an **ordinary differential equation** (ODE).

a) For the following definitions, let's assume that the variable x represents the independent variable (similar to what is often done in a mathematics course), and $y = u(x)$ is the dependent variable (y is given by some function u of the independent variable x).

b) The **order** of an ODE is the order of the highest derivative that appears in the equation. Eqs. (VIII-1) and (VIII-2) are *second order* differential equations, and Eq. (VIII-3) is a *first order* DE.

c) It is convenient to follow the usual notation in the theory of DEs to represent $du(x)/dx, d^2u(x)/dx^2, ..., d^nu(x)/dx^n$ with the notation $y', y'', ..., y^{(n)}$. Thus Eq. (VIII-1) is written as

$$F\left(x, y, y', ..., y^{(n)}\right) = 0. \qquad \text{(VIII-5)}$$

d) Occasionally, other letters will be used instead of x for the independent variable and y for the dependent variable (*i.e.*, function) — the meaning will be clear from the context.

e) We shall assume that it is always possible to solve a given ODE for the highest derivative, obtaining

$$y^{(n)} = f\left(x, y, y', y'', ..., y^{(n-1)}\right). \qquad \text{(VIII-6)}$$

f) A second important classification of ODEs is according to whether they are **linear** or **nonlinear**. The DE in Eq. (VIII-5) is said to be *linear* if F is a linear function of the variables $y, y', ..., y^{(n)}$. Thus the general linear ODE of order n is

$$a_0(x)y^{(n)} + a_1(x)y^{(n-1)} + \cdots + a_n(x)y = g(x). \qquad \text{(VIII-7)}$$

g) An equation which is <u>not</u> of the form in Eq. (IV-7) is a *nonlinear* equation. For example, the equation for the motion of a pendulum is nonlinear:

$$\frac{d^2\theta}{dt^2} + \frac{g}{\ell}\sin\theta = 0. \qquad \text{(VIII-8)}$$

3. If a DE depends upon several independent variables, it is called a **partial differential equation** (PDE) $\Longrightarrow$ the DE contains *partial* derivatives (*e.g.*, $\partial/\partial t$) instead of ordinary derivatives (*e.g.*, d/dt). The *wave equation* is a good example of a PDE:

$$a^2\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial^2 u(x,t)}{\partial t^2}. \qquad \text{(VIII-9)}$$

B. Numerical Methods

1. Terminology:

a) **Shooting** or **marching methods**: The solution is calculated step by step by starting at one boundary and integrating toward the other. The **step size** is the change (*e.g.*, Δx) in independent variable used in a shooting or marching scheme.

b) **Iterative method**: A repetitive process by which successively more accurate approximations to the solution are obtained. An **iteration** is one cycle of the repetitive process.

c) **Difference equation**: An approximation to a DE where a derivative is replaced by a quotient.

d) **Relaxation method**: A solution is calculated everywhere at once by solving a set of difference equations in an iterative fashion.

e) **Computational mesh** or **grid**: The independent variable is represented by a set of discrete values (*e.g.*, a set layers of given thickness in a stellar atmosphere) called **grid points**, **zones**, or **cells**. The Δx_i (or Δr_i, ΔM_i, etc.) is called the **grid spacing** or **mesh size** or **interval**.

$$\Delta x_i \equiv x_{i+1} - x_i. \qquad \text{(VIII-10)}$$

f) A **model** then becomes the set of physical properties specified at all the grid points, *e.g.*,

$$\{P_i, T_i, F_i, \rho_i\} \text{ at zones } x_i. \qquad \text{(VIII-11)}$$

g) An **evolution** is a sequence of models at different times t_n. Each model is a **generation** or **cycle**. Each successive advance forward in time is called the **time step** and

$$\Delta t \equiv t_{n+1} - t_n \qquad \text{(VIII-12)}$$

is called the **time step size**.

h) **Truncation error** (TE) is the *per step* (for shooting schemes) or *per mesh* (for relaxation schemes) error in the calculation of the dependent variables. **Cumulative truncation error** (CE) is the total such error across the grid.

i) **Roundoff error** is error introduced by the finite number of digits carried by the computer. The higher precision you use, the smaller the roundoff error.

j) The **order** of a numerical scheme is the power of the mesh size or step size in the highest order terms which are accurately represented by a numerical scheme. Some books use the TE, some the CE, to define this, and so there is often some confusion. This definition of *order* should not be confused with the *order of the ODE*.

k) **Explicit schemes**: One where values at the next step are obtained by a direct algebraic computation involving only values from the previous step.

l) **Implicit schemes**: One where new values must be solved for iteratively.

2. Expansions:

a) Often there are times (especially at boundaries) where **singularities** appear in an equation (*i.e.*, divide by zero).

During these times, one must use expansions to avoid these formal singularities. For instance, in the hydrostatic equilibrium equation:

$$\frac{dP}{dr} = -\rho \frac{GM_r}{r^2}, \qquad \text{(VIII-13)}$$

as $r \to 0$, $M_r \to 0$, so the RHS $\to 0/0$!

b) One must use the boundary conditions to develop **Taylor expansions** away from the singularities.

i) The Taylor expansion takes the form

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \cdots, \qquad \text{(VIII-14)}$$

where the symbol $(\cdots)$ means higher-order terms that are usually dropped from the derivation.

ii) An alternative, equivalent form of the Taylor series used in numerical analysis is

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(\zeta), \quad \text{(VIII-15)}$$

where ζ is a value between x and $x+h$.

iii) We have not dropped any terms in Eq. (VIII-15); this expansion has a *finite* number of terms. Taylor's theorem guarantees that there exists *some* value ζ for which Eq. (VIII-15) is true, but it doesn't tell us what that value is.

c) With Taylor expansions in mind, we can transform the derivative formula from

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}, \qquad \text{(VIII-16)}$$

to

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{1}{2} h f''(\zeta), \qquad \text{(VIII-17)}$$

where $x \leq \zeta \leq x + h$ as covered in §VI.A.

i) This equation is known as the **right derivative** formula.

ii) The last term on the right is the **truncation error** $\Longrightarrow$ it is the error introduced by the truncation of the Taylor series.

iii) Since we don't know the value of ζ a priori, the $f''(x)$ term (truncate) is usually dropped and we say the error was we make by neglecting this term is the truncation error, and as such, Eq. (VIII-17) is often written as

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h), \qquad \text{(VIII-18)}$$

where the truncation error term is now just specified by its order in h.

iv) Note that the truncation error is linear in h, the smaller we make h, the smaller our TE, however, the more calculations we have to make, which results in larger roundoff errors.

v) Keep in mind that the *truncation error* depends on the approximation used in the algorithm, whereas the *roundoff error* depends on the hardware.

d) Note that we also can reduce the size of the TE by introducing a **centered formula** for the derivative equation:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}. \qquad \text{(VIII-19)}$$

i) This formula is said to be *centered* in x.

ii) While this formula looks very similar to Eq. (VIII-17), there is a big difference when h is finite, since a Taylor expansion of this form of the derivative takes on the following form:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{6}h^2 f^{(3)}(\zeta), \qquad \text{(VIII-20)}$$

where $f^{(3)}$ is the 3rd derivative of $f(x)$ and $x - h \leq \zeta \leq x + h$.

iii) The TE is now *quadratic* in h, a big improvement over Eq. (VIII-17).

iv) From this formalism, it can be shown that the Taylor expansion of the second derivative is

$$f''(x) = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} - \frac{1}{12}h^2 f^{(4)}(\zeta), \qquad \text{(VIII-21)}$$

where $x - h \leq \zeta \leq x + h$. Again, the TE is quadratic in h.

e) **Selecting values of** h: What do you pick for h?

i) First, define the absolute error as

$$\varepsilon = |(\text{true value}) - (\text{computed value})|. \qquad \text{(VIII-22)}$$

ii) Neglecting roundoff error, to make the TE term in Eq. (VIII-20) small with respect to the other term

in this equation, then choose

$$h < \sqrt{\frac{6\varepsilon}{|f^{(3)}(\zeta)|}}. \qquad \text{(VIII-23)}$$

iii) Generally, we don't know $f^{(3)}$, but often we can set a bound. For example, if $f(x) = \sin(x)$, then $|f^{(3)}(\zeta)| \leq 1$ so if we want an absolute error of $\varepsilon \approx 10^{-6}$ (a typical value), then we should take $h \approx 2 \times 10^{-3}$.

iv) If we cannot estimate an upper bound, then arbitrarily pick a value of h, use it, try a smaller value of h, compare the two answers, and if they are close enough, assume everything is fine and your answer is converged. If not, keep on choosing smaller values of h (*i.e.*, iterate) until the above is true. This is <u>not</u> universally true however!

3. Shooting Methods:

a) Assume we have a DE given by $dy/dx = f(x, y)$ as shown in Figure (VIII-1). Now, given (x_j, y_j) for all $j \leq i$, obtain (x_{i+1}, y_{i+1}). This is the standard technique in the **shooting** or **marching method**. There are a variety of ways to carry out such a calculation.

b) **Euler's Method**: Assume we wish to follow the motion of a mass m using Newton's 2nd Law of Motion. The equation of motion that we want to solve numerically is

$$\frac{d\vec{v}}{dt} = \vec{a}(\vec{r}, \vec{v}) = \frac{1}{m}\vec{F}, \qquad \text{(VIII-24)}$$

where

$$\frac{d\vec{r}}{dt} = \vec{v} \qquad \text{(VIII-25)}$$

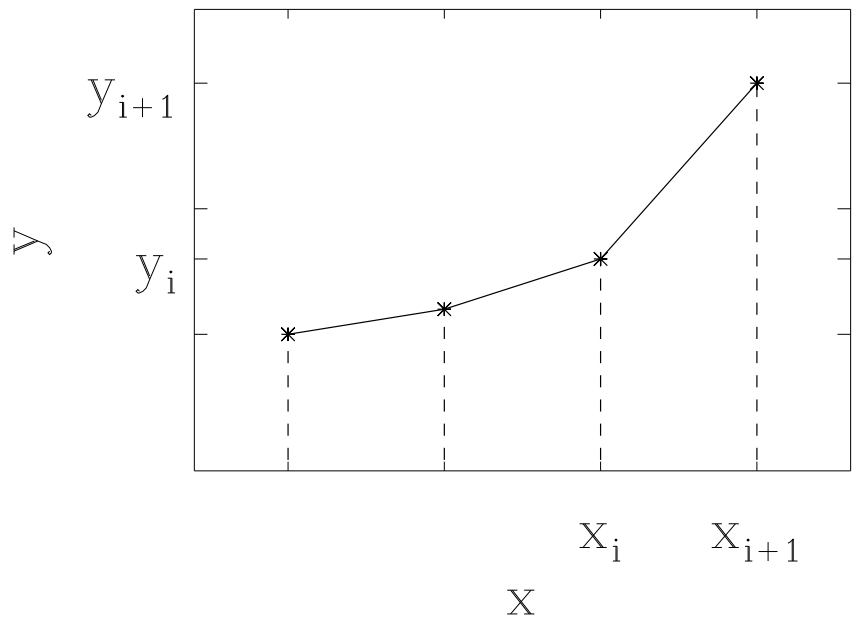


Figure VIII–1: Data points given by the DE $dy/dx = f(x, y)$.

and $\vec{a}$ is the acceleration. Euler's method uses the right derivative formula (see Eq. VIII-18), where we replace the grid step h with the time step τ. The equations of motion become

$$\frac{\vec{v}(t+\tau) - \vec{v}(t)}{\tau} + O(\tau) = \vec{a}[\vec{r}(t), \vec{v}(t)] \tag{VIII-26}$$

$$\frac{\vec{r}(t+\tau) - \vec{r}(t)}{\tau} + O(\tau) = \vec{v}(t) \tag{VIII-27}$$

or

$$\vec{v}(t+\tau) = \vec{v}(t) + \tau\vec{a}[\vec{r}(t), \vec{v}(t)] + O(\tau^2) \tag{VIII-28}$$

$$\vec{r}(t+\tau) = \vec{r}(t) + \tau\vec{v}(t) + O(\tau^2). \tag{VIII-29}$$

Notice that $\tau O(\tau) = O(\tau^2)$.

i) We introduce the notation:

$$f_n \equiv f[(n-1)\tau]; \quad n = 1, 2, ... \tag{VIII-30}$$

so $f_1 = f(t = 0)$. Our equations for the Euler method (dropping the error term) now take the

form

$$\vec{v}_{n+1} = \vec{v}_n + \tau \vec{a}_n \qquad \text{(VIII-31)}$$
$$\vec{r}_{n+1} = \vec{r}_n + \tau \vec{v}_n . \qquad \text{(VIII-32)}$$

ii) The calculation of trajectory would proceed as follows:

1. Specify the initial conditions $\vec{r}_i$ and $\vec{v}_i$.
2. Choose a time step τ.
3. Calculate the acceleration given the current $\vec{r}$ and $\vec{v}$.
4. Use the Euler method to compute the new $\vec{r}$ and $\vec{v}$.
5. Go to step 3 until enough trajectory points have been computed.

iii) The truncation error in such a scheme is

$$\text{TE} = \frac{1}{2}\tau_i^2 \left.\frac{d^2 r}{dt^2}\right|_i . \qquad \text{(VIII-33)}$$

Note that from Eqs. (VIII-26,27), Euler's method is 1st order accurate.

iv) The cumulative error is

$$\begin{aligned} \text{CE}_N &\approx \frac{1}{2}\sum_{i=1}^{N} \tau_i^2 \left.\frac{d^2 r}{dt^2}\right|_i , \\ &\approx \frac{1}{2} N \left(\frac{t_N - t_0}{N}\right)^2 \overline{\frac{d^2 r}{dt^2}} \\ &\approx \frac{1}{2}\left(\frac{t_N - t_0}{N}\right)(t_N - t_0)\overline{\frac{d^2 r}{dt^2}} \\ &\approx \frac{1}{2}\overline{\Delta t}(t_N - t_0)\overline{\frac{d^2 r}{dt^2}} \qquad \text{(VIII-34)} \end{aligned}$$

or CE $\sim O(\overline{\Delta t})$.

c) **Euler-Cromer and Midpoint Methods**: A modified version of Euler's method is

$$\vec{v}_{n+1} = \vec{v}_n + \tau \vec{a}_n \qquad \text{(VIII-35)}$$

$$\vec{r}_{n+1} = \vec{r}_n + \tau \vec{v}_{n+1}. \qquad \text{(VIII-36)}$$

i) The updated velocity is used in the second equation. This formula is called the *Euler-Cromer method*.

ii) Notice that the TE is still of order τ^2.

iii) We can modify this further to come up with the *midpoint method*,

$$\vec{v}_{n+1} = \vec{v}_n + \tau \vec{a}_n \qquad \text{(VIII-37)}$$

$$\vec{r}_{n+1} = \vec{r}_n + \tau \frac{\vec{v}_n + \vec{v}_{n+1}}{2}. \qquad \text{(VIII-38)}$$

iv) Plugging the velocity equation into the position equation, we see that

$$\vec{r}_{n+1} = \vec{r}_n + \tau \vec{v}_n + \frac{1}{2}\vec{a}_n \tau^2. \qquad \text{(VIII-39)}$$

v) The TE is still of order τ^2 in the velocity equation, but for position the TE is now τ^3. Unfortunately, this midpoint method gives good results for relatively few physical systems (projectile motion is one of them).

d) **Second-Order Runge-Kutta Method**: Runge-Kutta (RK) schemes establish higher derivatives by calculating intermediate (or provisional) values of y in the interval (x_i, x_{i+1}).

i) Second-order RK involves 2 substeps per step:

$$(1) \quad y^p_{i+1} = y_i + \Delta x_i f(x_i, y_i) \qquad \text{(VIII-40)}$$

$$(2) \quad y_{i+1} = y_i + \frac{\Delta x_i}{2} \left[f(x_i, y_i) + f(x_{i+1}, y^p_{i+1}) \right], \qquad \text{(VIII-41)}$$

where the superscript p represents *provisional*.

ii) Since RK schemes are widely used in computational physics, especially 4th-order RK, we will devote an entire subsection to it (see below).

e) **Predictor-Corrector Methods**: These use (x_j, y_j) for $j \leq i$ to establish higher derivatives $d^n y/dx^n$. We will not cover these methods in detail in this course.

4. **Relaxation (Henyey) Methods**: In these methods, one solves all equations at all grid points at once. In our example here, let's assume we have a spherical distribution of gas.

a) **Grid**: First, one needs to set up a grid of independent variable values from one boundary to the other. Let's assume that r_i corresponds to this grid, then the grid is represented as $r_i, i = 0, 1, ..., N$ or $N + 1$ grid points.

b) **Model**: The model is defined as the set of dependent variables at each grid point, *e.g.*, $\{P_i, T_i, M_{ri}, L_{ri}\}$ at r_i for $4N + 4$ unknowns.

c) Then the **Differential Equations** take on the form

$$\frac{dP}{dr} = f_1(P, T, M_r, L_r, r) \qquad \text{(VIII-42)}$$

$$\frac{dT}{dr} = f_2(P, T, M_r, L_r, r) \qquad \text{(VIII-43)}$$

$$\frac{dM_r}{dr} = f_3(P, T, M_r, L_r, r) \qquad \text{(VIII-44)}$$

$$\frac{dL_r}{dr} = f_4(P, T, M_r, L_r, r). \qquad \text{(VIII-45)}$$

d) Differencing Schemes:

i) Forward differencing:

$$\frac{P_{i+1} - P_i}{\Delta r_i} = f_1\left(P_i, T_i, \cdots\right). \qquad \text{(VIII-46)}$$

ii) Centered differencing:

$$\frac{P_{i+1} - P_i}{\Delta r_i} = f_1\left(\frac{P_{i+1} + P_i}{2}, \frac{T_{i+1} + T_i}{2}, \cdots\right). \qquad \text{(VIII-47)}$$

iii) Backward differencing:

$$\frac{P_{i+1} - P_i}{\Delta r_i} = f_1\left(P_{i+1}, T_{i+1}, \cdots\right). \qquad \text{(VIII-48)}$$

e) Boundary Conditions:

i) Center (*i.e.*, boundary 1):

$$\begin{aligned} C_1\left(P_0, T_0, M_{r0}, L_{r0}\right) &= 0 \\ C_2\left(P_0, T_0, M_{r0}, L_{r0}\right) &= 0. \end{aligned} \qquad \text{(VIII-49)}$$

ii) Surface (*i.e.*, boundary 2):

$$\begin{aligned} S_1\left(P_N, T_N, M_{rN}, L_{rN}\right) &= 0 \\ S_2\left(P_N, T_N, M_{rN}, L_{rN}\right) &= 0. \end{aligned} \qquad \text{(VIII-50)}$$

f) Difference Equations:

i) Choose one of the differencing schemes and apply to Eqs. (VIII-42) through (VIII-45).

ii) Plug in known r_i values which results in $4N$ **difference equations**, which <u>can</u> be <u>non</u>linear <u>algebraic</u> equations for $4N + 4$ unknowns.

iii) These equations can be written formally as

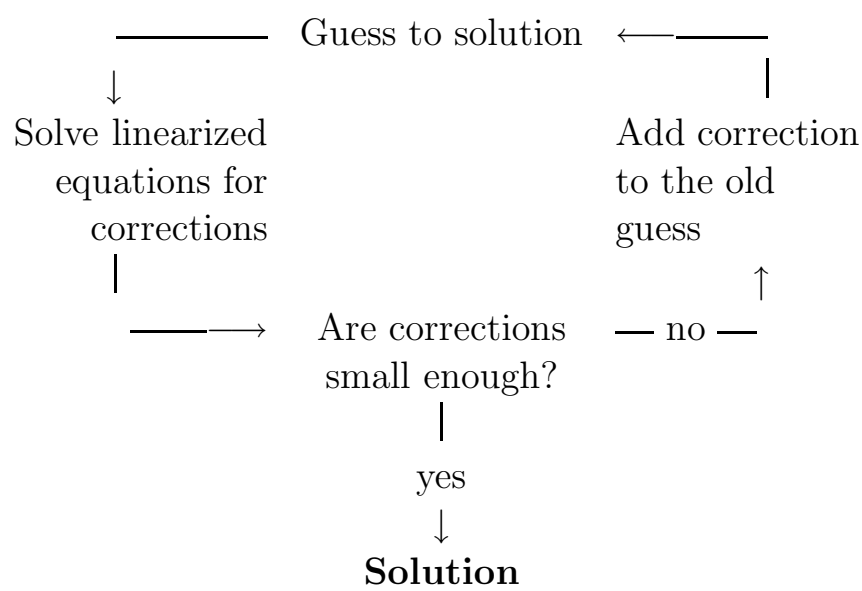
$$
\begin{aligned}
g_1\left(P_i, T_i, M_{ri}, L_{ri}, P_{i+1}, T_{i+1}, M_{ri+1}, L_{ri+1}\right) &= 0 \quad \text{(VIII-51)} \\
g_2\left(P_i, T_i, M_{ri}, L_{ri}, P_{i+1}, T_{i+1}, M_{ri+1}, L_{ri+1}\right) &= 0 \quad \text{(VIII-52)} \\
g_3\left(P_i, T_i, M_{ri}, L_{ri}, P_{i+1}, T_{i+1}, M_{ri+1}, L_{ri+1}\right) &= 0 \quad \text{(VIII-53)} \\
g_4\left(P_i, T_i, M_{ri}, L_{ri}, P_{i+1}, T_{i+1}, M_{ri+1}, L_{ri+1}\right) &= 0 \; . \quad \text{(VIII-54)}
\end{aligned}
$$

iv) We have such equations for $i = 0, 1, ..., N-1$.

$$
\begin{array}{ccc}
\text{(VIII-51) to (VIII-54)} & & 4N+4 \text{ equations} \\
\text{plus} & \Longrightarrow & \text{in} \\
\text{(VIII-49) and (VIII-50)} & & 4N+4 \text{ unknowns}
\end{array}
$$

v) Great! The only problem is that these equations are not linear. They can't be solved directly. A generalized Newton-Raphson iterative scheme is used to overcome this difficulty.

g) **Method**:



One *iterates* until the guesses *converge* to some prescribed accuracy determined by the size of the corrections.

$$\begin{array}{rl} j^{th} \text{ iteration} & \left\{P_i^{(j)}, T_i^{(j)}, M_{ri}^{(j)}, L_{ri}^{(j)}\right\} \\ j^{th} \text{ correction} & \left\{\delta P_i^{(j)}, \delta T_i^{(j)}, \delta M_{ri}^{(j)}, \delta L_{ri}^{(j)}\right\}. \end{array}$$

i) Consider one of the difference equations. In general,

$$g_m\left(P_i^{(j)}, T_i^{(j)}, ..., P_{i+1}^{(j)}, ...\right) \neq 0.$$

ii) Say we know $\left\{P_i^{(j)}, T_i^{(j)}, ...\right\}$ and want to calculate the jth correction. To generate linear equations we can solve for the jth correction, plug $\left\{P_i^{(j)} + \delta P_i^{(j)}, T_i^{(j)} + \delta T_i^{(j)}, ...\right\}$ into g_m, Taylor expand, keep terms up to first order only in the jth corrections, and then set it equal to zero.

$$\begin{aligned} g_m\left(P_i^{(j)} + \delta P_i^{(j)}, ..., P_{i+1}^{(j)} + \delta P_{i+1}^{(j)}, ...\right) &= 0 \\ g_m\left(P_i^{(j)}, ..., P_{i+1}^{(j)}, ...\right) + & \\ \delta P_i^{(j)} \frac{\partial g_m^{(j)}}{\partial P_i} + \delta T_i^{(j)} \frac{\partial g_m^{(j)}}{\partial T_i} + \cdots & \quad \text{(VIII-55)} \\ + \delta P_{i+1}^{(j)} \frac{\partial g_m^{(j)} \partial P_i}{+} \cdots &= 0 \end{aligned}$$

iii) We have now *linearized* the set of difference equations. The partial derivatives can be evaluated because they depend only on the known quantities $\left\{P_i^{(j)},\ T_i^{(j)}, ...\right\}$.

iv) By casting all of Eqs. (VIII-51) to (VIII-54) and (VIII-49) & (VIII-50), we get

$$\begin{array}{ccc} 4N+4 \text{ linear} & \text{for} & 4N+4 \text{ unknown} \\ \text{equations} & & \text{corrections} \end{array}$$

v) Formally,

$$\underset{\substack{\nearrow \\ \text{matrix of} \\ \text{partials}}}{\overleftrightarrow{A}} \cdot \underset{\substack{\nwarrow \\ \text{correction} \\ \text{vector}}}{\overrightarrow{\delta X}} = \underset{\substack{\uparrow \\ \text{error} \\ \text{vector}}}{-\vec{E}} \qquad \text{(VIII-56)}$$

$$\overrightarrow{\delta X} = \begin{bmatrix} \delta P_0^{(j)} \\ \delta T_0^{(j)} \\ \delta M_{r0}^{(j)} \\ \delta L_{r0}^{(j)} \\ \delta P_1^{(j)} \\ \delta T_1^{(j)} \\ \vdots \\ \delta M_{rN-1}^{(j)} \\ \delta L_{rN-1}^{(j)} \\ \delta P_N^{(j)} \\ \delta T_N^{(j)} \\ \delta M_{rN}^{(j)} \\ \delta L_{rN}^{(j)} \end{bmatrix} \qquad \vec{E} = \begin{bmatrix} C_1\left(P_0^{(j)}, \cdots\right) \\ C_2\left(P_0^{(j)}, \cdots\right) \\ g_1\left(P_0^{(j)}, \cdots\right) \\ g_2\left(P_0^{(j)}, \cdots\right) \\ g_3\left(P_0^{(j)}, \cdots\right) \\ g_4\left(P_0^{(j)}, \cdots\right) \\ g_1\left(P_1^{(j)}, \cdots\right) \\ \vdots \\ g_3\left(P_{N-1}^{(j)}, \cdots\right) \\ g_4\left(P_{N-1}^{(j)}, \cdots\right) \\ S_1\left(P_N^{(j)}, \cdots\right) \\ S_2\left(P_N^{(j)}, \cdots\right) \end{bmatrix}$$

$$\overleftrightarrow{A} = \begin{bmatrix}
\left.\frac{\partial C_1}{\partial P_0}\right|_0 & \left.\frac{\partial C_1}{\partial T_0}\right|_0 & \left.\frac{\partial C_1}{\partial M_{r0}}\right|_0 & \left.\frac{\partial C_1}{\partial L_{r0}}\right|_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\left.\frac{\partial C_2}{\partial P_0}\right|_0 & X & X & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
\left.\frac{\partial g_1}{\partial P_0}\right|_1 & \left.\frac{\partial g_1}{\partial T_0}\right|_1 & \left.\frac{\partial g_1}{\partial M_{r0}}\right|_1 & \left.\frac{\partial g_1}{\partial L_{r0}}\right|_1 & \left.\frac{\partial g_1}{\partial P_1}\right|_1 & \left.\frac{\partial g_1}{\partial T_1}\right|_1 & \left.\frac{\partial g_1}{\partial M_{r1}}\right|_1 & \left.\frac{\partial g_1}{\partial L_{r1}}\right|_1 & 0 & 0 & 0 & 0 & 0 & \\
\left.\frac{\partial g_2}{\partial P_0}\right|_1 & X & X & X & X & X & X & X & 0 & 0 & 0 & 0 & 0 & \\
X & X & X & X & X & X & X & X & 0 & 0 & 0 & 0 & 0 & \\
X & X & X & X & X & X & X & X & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & \left.\frac{\partial g_1}{\partial P_1}\right|_2 & \left.\frac{\partial g_1}{\partial T_1}\right|_2 & \left.\frac{\partial g_1}{\partial M_{r2}}\right|_2 & \left.\frac{\partial g_1}{\partial L_{r2}}\right|_2 & \left.\frac{\partial g_1}{\partial P_2}\right|_2 & \left.\frac{\partial g_1}{\partial T_2}\right|_2 & \left.\frac{\partial g_1}{\partial M_{r2}}\right|_2 & \left.\frac{\partial g_1}{\partial L_{r2}}\right|_2 & 0 & \cdots \\
0 & 0 & 0 & 0 & X & X & X & X & X & X & X & X & 0 & \\
0 & 0 & 0 & 0 & X & X & X & X & X & X & X & X & 0 & \\
0 & 0 & 0 & 0 & X & X & X & X & X & X & X & X & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \left.\frac{\partial g_1}{\partial P_2}\right|_3 & X & X & X & X & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X & X & X & X & X & \\
 & & & & & & \vdots & & & & & & &
\end{bmatrix}$$

vi) The $\overleftrightarrow{A}$ matrix has a *banded* structure and is mostly empty.

$$\begin{bmatrix} \begin{bmatrix} 2 \text{ x } 4 \end{bmatrix} & & & & \\ & \begin{bmatrix} 4 \text{ x } 8 \end{bmatrix} & & \bigcirc & \\ & & \begin{bmatrix} 4 \text{ x } 8 \end{bmatrix} & & \\ & & & \ddots & \\ & \bigcirc & & & \begin{bmatrix} 4 \text{ x } 8 \end{bmatrix} \\ & & & & & \begin{bmatrix} 2 \text{ x } 4 \end{bmatrix} \end{bmatrix}$$

vii) To solve for the jth correction, this matrix must be *inverted*. Numerically, the number of operations involved is

$$\begin{array}{ccc} \text{Full} \\ (4N+4) \times (4N+4) & \sim & (4N+4)^2 \\ \\ \text{Actual} \\ \text{Banded Matrix} & \sim & 8^2(4N+4) \end{array}$$

h) **Gaussian Elimination**: Many methods now exist for inverting banded matrices. In the last section of the notes, we were introduced to the Gaussian elimination scheme for solving sets of linear equations. To solve ODEs using this technique, do the following:

i) Start at one corner and solve one block of equations for some variables in terms of the others, *e.g.*, for upper left 2 x 4 block,

$$\begin{aligned} \delta P_0^{(j)} &= a_{P_0}\delta M_{r0}^{(j)} + b_{P_0}\delta L_{r0}^{(j)} + c_{P_0} \\ \delta T_0^{(j)} &= a_{T_0}\delta M_{r0}^{(j)} + b_{T_0}\delta L_{r0}^{(j)} + c_{T_0}. \end{aligned} \qquad \text{(VIII-57)}$$

ii) Store the a, b, c's.

iii) Substitute Eq. (VIII-57) in the next block and repeat (i) and (ii) above, *e.g.*,

$$\begin{aligned} \delta M_{r0}^{(j)} &= a_{M_{r0}}\delta M_{r1}^{(j)} + b_{M_{r0}}\delta L_{r1}^{(j)} + c_{M_{r0}} \\ \delta L_{r0}^{(j)} &= a_{L_{r0}}\delta M_{r1}^{(j)} + b_{L_{r0}}\delta L_{r1}^{(j)} + c_{L_{r0}}. \\ \delta P_1^{(j)} &= a_{P_1}\delta M_{r1}^{(j)} + b_{P_1}\delta L_{r1}^{(j)} + c_{P_1} \\ \delta T_1^{(j)} &= a_{T_1}\delta M_{r1}^{(j)} + b_{T_1}\delta L_{r0}^{(j)} + c_{T_1} \end{aligned} \qquad \text{(VIII-58)}$$

become the new set of Eqs. (VIII-57) to substitute into the next block.

iv) Solving the last set of equations at the other end of the matrix gives $\delta M_{rN}^{(j)} (= \delta M_\star^{(j)})$ and $\delta L_{rN}^{(j)} (= \delta L_\star^{(j)})$.

v) Now go backwards and *back substitute* into the Eqs. (VIII-58) using the stored a, b, c's to find all the jth corrections.

i) **Convergence Criterion**: The iterative process is said to *converge* when the corrections are as small as desired, *e.g.*,

$$\begin{aligned} \text{all } \frac{|\delta P_i|}{P_i}, \ \underline{\text{etc.}} \quad &< \ C = \ \text{const.} \\ C \ &\sim \ 10^{-2} \text{ to } 10^{-6}. \end{aligned}$$

Most codes (at least 1-D) converge after ∼3 to 5 iterations.

C. Fourth-Order Runge-Kutta Method

1. When numerically solving ODE, one typically rewrites 2nd and higher-order DEs into a set of 1st-order DEs:

a) A 2nd-order equation such as

$$a = \frac{d^2x}{dt^2}. \qquad \text{(VIII-59)}$$

b) This can be rewritten as 2 equations:

$$v = \frac{dx}{dt} \qquad \text{(VIII-60)}$$

$$a = \frac{dv}{dt}. \qquad \text{(VIII-61)}$$

2. As shown above, Euler's method is a first-order method which is graphically represented in Figure (VIII-2).

3. The Runge-Kutta method is essentially a modified Euler's method.

a) Use the derivative at one step to extrapolate the midpoint value — use the midpoint derivative to extrapolate the function at the next step (see Figure VIII-3).

b) Evaluates the derivative function twice at each step τ. Cumulative error is of order $O(\tau^2)$, a second-order method.

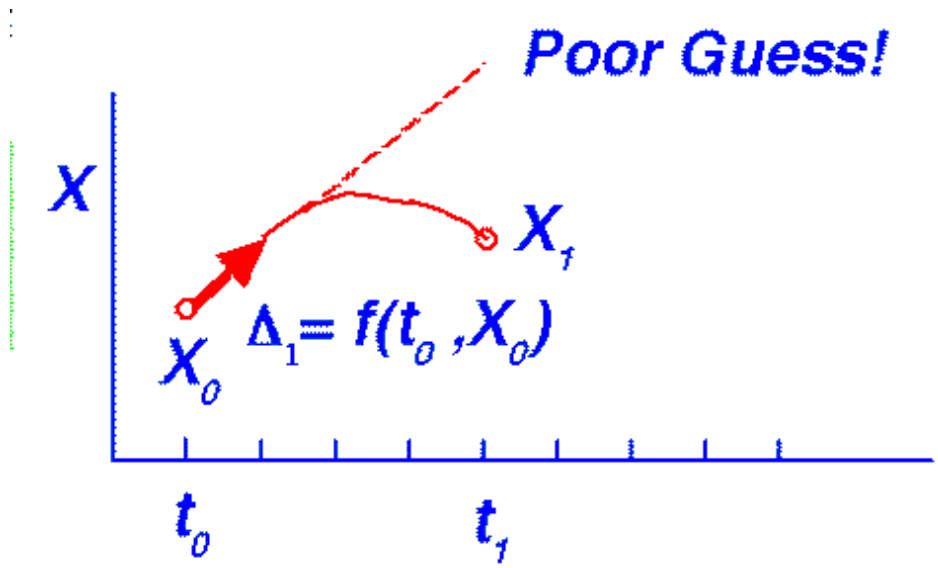


Figure VIII–2: Euler's method is a first-order method which is not necessarily accurate.

$$X_0 = \textit{initial value}$$
$$\Delta_1 = f(t_i, X_i)$$
$$\Delta_2 = f(t_i + \frac{1}{2}h, X_i + \frac{1}{2}h\Delta_1)$$
$$X_{i+1} = X_i + h * \Delta_2$$

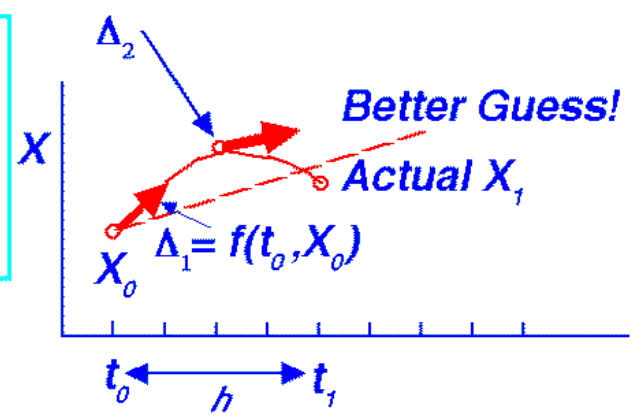


Figure VIII–3: The Runge-Kutta method is a second-order method which is more accurate than Euler's method.

4. Runge-Kutta methods achieve better results than Euler by using intermediate computations at intermediate grid steps.

5. The **fourth-order** rule is the favorite method as it achieves good accuracy with modest computational complexity.

a) Use derivative of first step to get trial midpoint.

b) Use its derivative at first step to get second trial midpoint.

c) Use its derivative to get a trail end point.

d) Integrate by **Simpson's Rule**, using average of two midpoint estimates.

e) The cumulative error is of fourth order and truncation error is $O(\tau^5)$.

6. The fourth-order RK scheme mathematically is:

$$\vec{x}(t+\tau) = \vec{x}(t) + \frac{1}{6}\tau\left(\vec{F}_1 + 2\vec{F}_2 + 2\vec{F}_3 + \vec{F}_4\right), \qquad \text{(VIII-62)}$$

where

$$\vec{F}_1 = \vec{f}(\vec{x}, t) \qquad \text{(VIII-63)}$$

$$\vec{F}_2 = \vec{f}\left(\vec{x} + \frac{1}{2}\tau\vec{F}_1, t + \frac{1}{2}\tau\right) \qquad \text{(VIII-64)}$$

$$\vec{F}_3 = \vec{f}\left(\vec{x} + \frac{1}{2}\tau\vec{F}_2, t + \frac{1}{2}\tau\right) \qquad \text{(VIII-65)}$$

$$\vec{F}_4 = \vec{f}\left(\vec{x} + \tau\vec{F}_3, t + \tau\right). \qquad \text{(VIII-66)}$$

7. Compared with Euler, 4th-order RK has 4 times more calculations per step, but uses the fourth root as many steps to achieve convergence (see Figure VIII-4).

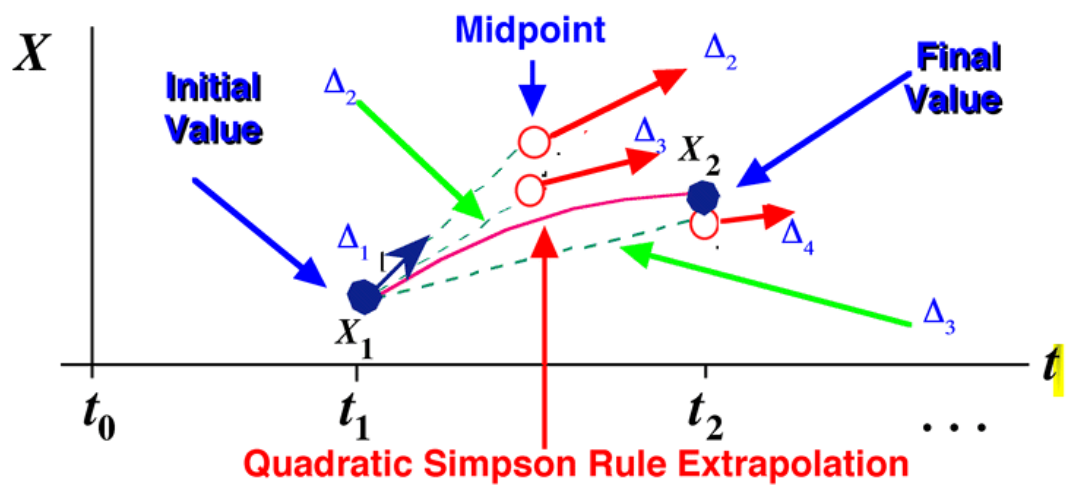


Figure VIII–4: Fourth-order RK with the $\vec{F}_i$'s in Eqs. (VIII-63) to (VIII-66) being represented with Δ_i's in this figure. Also $\vec{f}$ is given by X in this figure.

8. You may wonder, *Why 4th-order and not 8th- or 23rd-order Runge-Kutta?* Well, the higher order methods have better truncation error but also require more computation, hence, more roundoff error. The optimum, for RK schemes, is 4th order.

9. Sometimes accuracy can be improved through use of an **adaptive step size**. Adaptive methods are fairly easy to incorporate and I refer you to *Numerical Recipes* for a description on how to incorporate them.

10. The following program is taken from *Numerical Recipes* and is a Fortran 77 version of the 4th-order RK scheme:

```
*
* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<  RK4 - RK4 - RK4  >>>>>>>>>>>>>>>>>>>>>>>>>>
*
      SUBROUTINE RK4(Y, DYDX, N, X, H, YOUT, DERIVS)
*
* Given values for N variables Y and their derivatives DYDX known at X, use the
* 4th-order Runge-Kutta method to advance the solution over an interval H and
* return the incremented variables as YOUT, which need not be a distinct
* array from Y.  The user supplies the subroutine DERIVS(X, Y, DYDX) which
* returns the derivatives DYDX at X.
*
      IMPLICIT REAL*8 (A-H,O-Z)
      EXTERNAL DERIVS
      PARAMETER (NMAX = 10)
      DIMENSION Y(N), DYDX(N), YOUT(N), YT(NMAX), DYT(NMAX), DYM(NMAX)
*
      HH = H * 0.5D0
      H6 = H / 6.D0
      XH = X + HH
*
* First step.
*
      DO 11 I = 1, N
        YT(I) = Y(I) + HH*DYDX(I)
 11   CONTINUE
*
* Second step.
*
      CALL DERIVS(XH, YT, DYT)
      DO 12 I = 1, N
        YT(I) = Y(I) + HH*DYT(I)
 12   CONTINUE
*
* Third step.
*
      CALL DERIVS(XH, YT, DYM)
      DO 13 I = 1, N
        YT(I) = Y(I) + H*DYM(I)
        DYM(I) = DYT(I) + DYM(I)
 13   CONTINUE
*
* Fourth step.
*
      CALL DERIVS(X+H, YT, DYT)
      DO 14 I = 1, N
        YOUT(I) = Y(I) + H6*(DYDX(I)+DYT(I)+2.D0*DYM(I))
 14   CONTINUE
```

```
*
      RETURN
      END
```

D. The Adams Method: The Shampine-Gordon Routine.

1. In 1975, L.F. Shampine and M.K. Gordon wrote a book titled *Computer Solution of Ordinary Differential Equations: The Initial Value Problem.*

a) This textbook described numerical techniques in solving *non-stiff* initial value problems in ordinary differential equations.

b) The ODE code itself is comprised of a few subroutines and a driver program. The subroutines are

i) DE: Integrates a system of up to 20 first-order ODEs of the form

DY(I)/DT = F(T, Y(1), Y(2), ..., Y(NEQN))
Y(I) given at T.

This subroutine integrates from T to TOUT. On return, the parameters in the call list are initialized for continuing the integration. The user has only to define a new value of TOUT and call DE again.

ii) STEP: Integrates a system of first order ODEs over one step, normally from T to T + H, using a modified divided difference form of the **Adams Pece** formulas. Local extrapolation is used to improve absolute stability and accuracy. The code adjusts its order at step size to control the local error per unit step in a generalized sense. Special devices are included to control roundoff error and to detect when the user is requesting too much accuracy (see program ode.f on the course web pages for further details).

iii) INTRP: The methods used in subroutine STEP approximate the solution near x by a polynomial. Subroutine INTRP approximates the solution at x_{out} by evaluating the polynomial there. Information defining this polynomial is passed from STEP so INTRP cannot be used alone (see program ode.f on the Course Web Pages).

c) Note that the user of these routines also has to supply a subroutine called F which sets up the derivative equations YP to be solved by DE. An example of such a subroutine is provided in the ode.f file available on the Course Web Pages if you wish to use them. If you use any of these routines, please note that whenever you see a '***' mark in the far right of the program, these lines have to be modified to reflect the machine precision, ϵ_m, for single and double precision of the machine on which you are carrying out these calculations. You can determine this precision with machin.f program supplied to you on the Course Web Pages.