CHAPTER 9

ODEs: INITIAL VALUE PROBLEMS

SOLVED EXAMPLE PROBLEMS

for

NUMERICAL METHODS FOR SCIENTISTS AND ENGINEERS with Pseudocodes

By Zekeriya ALTAÇ

April 2025

EXAMPLE 9.1: Numerical Solution of First order IVPs

A spherical bead made of an aluminum alloy ($\rho = 2700$ kg/m^3, $c_p = 1000$ J/kg·K) with a radius of $R = 1$ cm and an initial temperature of $T_0 = 500$ K is quenched in a water tank at $T_\infty = 300$ K. As the bead sinks in the tank, the mean heat transfer coefficient resulting on its surface is given as a function of the surface temperature as $h(T) = h_0(1 - \beta T)$, where $h_0 = 450$ W/m^2·K and $\beta = 0.0005$ K^{-1}. The transient mean temperature of the bead is an initial value problem stated as follows:

$$\rho V c_p \frac{dT}{dt} = -h(T) A_s \left(T(t) - T_\infty\right), \quad T(0) = T_0$$

where A_s and V are the surface area and volume of the bead, respectively, and c_p and ρ are the specific heat and density of the aluminum alloy.

Solve the resulting IVP on [0,100] s with the explicit/implicit Euler and trapezoidal (linearized) methods using $\Delta t = 10$, 5 and 2 s. Compare your estimates with the true solution given by

$$T(t) = T_\infty + \frac{(T_0 - T_\infty)h(T_\infty)}{h_0 - h(T_0 - T_\infty) + h(T_0)\,\exp\left[3h(T_\infty)t/\rho c_p R\right]}$$

SOLUTION:

Having substituted the numerical values, the initial value problem becomes

$$\frac{dT}{dt} = f(t,T) = -0.05(1 - 0.0005T)(T - 300), \quad T(0) = 500\,\text{K}$$

Explicit Euler method for the current step estimate, T_{i+1}, yields

$$T_{i+1} = T_i + hf(t_i, T_i) + \mathcal{O}(h^2) \quad \text{for } i = 0, 1, 2, \ldots$$

where $t_i = ih$, $h = \Delta t$, and $T_i = T(t_i)$ denote the time, time step, and the temperature estimate at t_i (prior step), respectively.

Implicit Euler method for the current step estimate, T_{i+1}, yields

$$T_{i+1} = T_i + hf(t_{i+1}, T_{i+1}) + \mathcal{O}(h^2) \quad \text{for } i = 0, 1, 2, \ldots$$

which requires solving a nonlinear equation to find every current step if $f(t,T)$ is a nonlinear function.

Table 9.1: Absolute true errors and true solutions with various time steps using explicit and implicit Euler methods.

			Euler Explicit Method			Euler Implicit Method		
i	t_i	T_i	$h=10$	$h=5$	$h=2.5$	$h=10$	$h=5$	$h=2.5$
0	0	500						
1	10	436.306	11.306	5.037	2.391	7.682	4.164	2.174
2	20	391.657	15.876	7.161	3.419	11.245	6.044	3.142
3	30	361.063	16.053	7.391	3.559	12.088	6.424	3.318
4	40	340.424	14.036	6.628	3.225	11.381	5.969	3.06
5	50	326.647	11.300	5.487	2.700	9.942	5.139	2.613
6	60	317.516	8.633	4.314	2.149	8.277	4.212	2.123
7	70	311.493	6.365	3.275	1.651	6.666	3.337	1.666
8	80	307.531	4.576	2.423	1.237	5.240	2.579	1.276
9	90	304.931	3.230	1.759	0.909	4.045	1.957	0.958
10	100	303.227	2.248	1.258	0.658	3.079	1.463	0.709

In **Table 9.1**, the absolute true errors of the estimates obtained with the explicit and implicit methods for $h = 10$, 5 and 2.5, along with the true solution, are presented. The absolute true errors for all h depict an increase up to $t \approx 30$ s. Then the error decreases with increasing t; however, the magnitude of the errors is reduced with decreasing h. The mean values of the tabulated absolute errors for $h = 10$, 5 and 2.5 s are on the order of 8.6, 4.3, and 2.15, respectively. Note that the mean error decreased linearly by h, which is a direct result of a first-order numerical scheme.

The trapezoidal rule for the current step estimate, T_{i+1}, results in

$$T_{i+1} = T_i + \frac{h}{2}\Big(f(t_i, T_i) + f(t_{i+1}, T_{i+1})\Big) + \mathcal{O}(h^3) \quad \text{for } i = 0, 1, 2, \ldots$$

This scheme has a local truncation error of $\mathcal{O}(h^3)$ and a global error of $\mathcal{O}(h^2)$. The term $f(t_{i+1}+T_{i+1})$ makes this an implicit method, as T_{i+1} appears on both sides of the equation. This usually requires solving a nonlinear equation (or a linear one, if $f(t, T)$ is linear in T) at each time step. However, in this example, even though $f(t, T)$ is a nonlinear function, the resulting quadratic equation can be solved to find an expression for T_{i+1}:

$$T_{i+1} = \frac{1}{h}\left(40000 + 1150h - \sqrt{1.6 \times 10^9 + 9.2 \times 10^7 h + 122500h^2 - hT_i(80000 - 2300h + hT_i)}\right)$$

The absolute true errors of the estimates obtained with the linearized trapezoidal rule for $h = 10$, 5 and 2.5 are tabulated in **Table 9.2** along with the true solution. It should be noted that the absolute errors corresponding to the same value of h are much smaller than those for the explicit or implicit Euler methods. This is because the method has a second-order truncation error. The mean values of the absolute errors are 0.705, 0.175, and 0.043 for $h = 10$, 5 and 2.5, respectively. Also notice that by halving the time step, the average errors are reduced by a quarter; i.e., $\mathcal{O}(h^2/4)$.

Table 9.2: Estimates with the Linearized Trapezoidal rule.

			Absolute True Error		
i	t_i	T_i	$h = 10$	$h = 5$	$h = 2.5$
0	0	500	0	0	0
1	10	436.306	0.822	0.202	0.050
2	20	391.657	1.134	0.279	0.069
3	30	361.063	1.151	0.284	0.071
4	40	340.424	1.025	0.253	0.063
5	50	326.647	0.849	0.210	0.052
6	60	317.516	0.671	0.167	0.042
7	70	311.493	0.514	0.128	0.032
8	80	307.531	0.384	0.096	0.024
9	90	304.931	0.283	0.071	0.018
10	100	303.227	0.205	0.051	0.013

Next we investigate the behavior of the absolute true error over the solution interval. At the first time step, the error is exactly zero since the solution is the initial value. For subsequent time steps (t_1, t_2, ...), the error grows due to the inherent truncation error. At each step, the error for the trapezoidal rule grows approximately quadratically with the step size $\mathcal{O}(h^2)$. The local error at each step is of order $\mathcal{O}(h^3)$, but the global error after N steps grows like $\mathcal{O}(h^2)$, meaning that after N steps, the total error behaves as $\mathcal{O}(h^2)$.

Discussion: The main error associated with explicit and implicit Euler methods comes from two components: *truncation* and *round-off* errors.

Truncation error: It arises because the method uses a linear approximation for the solution over each time step. The explicit and implicit Euler methods are first-order methods, so the truncation error at each step is of order $\mathcal{O}(h)$, which means that the error at each time step is proportional to the step size h. On the other hand, the trapezoidal rule is a second-order method, and the truncation error at each step is of order $\mathcal{O}(h^2)$

Round-off error: It is the numerical error that arises due to the finite precision of computer arithmetic. Although the round-off error is typically smaller than the truncation error for large step sizes, it can accumulate over many steps.

Impact of Step Size on Error Distribution: For larger step sizes, the error grows more quickly because the method is less accurate. This leads to an increase in the solution error over time. For smaller step sizes, while the error per step decreases (as $\mathcal{O}(h)$ in explicit or implicit Euler methods or $\mathcal{O}(h^2)$ in the trapezoidal rule), the computational cost increases. A smaller h reduces the error per step, but the number of steps increases, and the overall error might still accumulate.

Stability and Error Growth: The explicit Euler method is conditionally stable, which means that the error can grow uncontrollably if the step size is too large. For stiff problems, the method can lead to very large errors even with moderate step sizes. In general, the absolute error tends to increase with time, especially in cases where the problem is stiff or if a large h is used. A smaller step size can reduce the error but increases the computational cost. For more accurate results, a balance must be struck between step size and computational efficiency. The scheme applying the trapezoidal rule is stable and good for *stiff problems*, where explicit methods like Euler can become unstable unless the step size is extremely small.

EXAMPLE 9.2: Taylor Polynomial Approximation

Consider **Example 9.1**. Obtain the third-degree Taylor polynomial approximation and use this approximation to obtain numerical solutions for $h = 10$, 5, and 2.5 and discuss the results.

SOLUTION:

The initial value problem is

$$\frac{dT}{dt} = f(t,T) = -0.05(1 - 0.0005T)(T - 300), \quad T(0) = 500\,\text{K}$$

Recall that any function $y(t)$ that is sufficiently differentiable can be expanded to a Taylor series. Likewise, the transient temperature, $T(t)$, which is unknown, can also be expanded to a third-degree Taylor polynomial, $p_3(T)$, which has the following form:

$$p_3(T_i) = T_{i+1} = T_i + h\frac{dT}{dt}(t_i) + \frac{h^2}{2!}\frac{d^2T}{dt^2}(t_i) + \frac{h^3}{3!}\frac{d^3T}{dt^3}(t_i) + \mathcal{O}(h^4)$$

Here, we need to analytically generate and compute the derivatives of $f(t,T)$ by using the chain rule repeatedly to compute $T'(t)$, $T''(t)$, and $T'''(t)$.

With the use of the chain rule, the required derivatives are obtained as follows:

$$\begin{aligned}
f(t,T) &= \frac{dT}{dt} = 15 - 0.0575\,T + 2.5\times10^{-5}\,T^2 \\
f'(t,T) &= \frac{d^2T}{dt^2} = \left(\frac{dT}{dt}\right)' = \left(-0.0575 + 0.00005\,T\right)T'(t) \\
&= -0.8625 + 0.00405625\,T - 4.3125\times10^{-6}\,T^2 + 1.25\times10^{-9}\,T^3 \\
f''(t,T) &= \frac{d^3T}{dt^3} = \left(\frac{d^2T}{dt^2}\right)' = 0.0608438 - 3.6261\times10^{-4}\,T + 6.53594\times10^{-7}\,T^2 \\
&\quad - 4.3125\times10^{-10}\,T^3 + 9.375\times10^{-14}\,T^4
\end{aligned}$$

where $T = T(t)$.

Substituting $T'(t_i)$, $T''(t_i)$, and $T'''(t_i)$ into the above polynomial approximation, simplifying and rearranging the terms, an expression for the current step value (for $i = 0,1,2,...$) is obtained as follows:

$$\begin{aligned}
T_{i+1} =& 15h - 0.43125h^2 + 0.0101406h^3 \\
&+ \left(1 - 0.0575h + 2.02813\times10^{-3}h^2 - 6.0435\times10^{-5}h^3\right)T_i \\
&+ \left(1 - 0.08625h + 0.0043573h^2\right)\,2.5\times10^{-5}\,h\,T_i^2 \\
&+ (1 - 0.115h)\,6.25\times10^{-10}\,h^2\,T_i^3 + 1.5625\times10^{-14}\,h^3\,T_i^4 + \mathcal{O}(h^4)
\end{aligned}$$

Note that the scheme is explicit with a fourth-order local error, and it does not require the solution of a nonlinear equation at each step. However, the number of arithmetic operations per time step can considerably increase, depending on the ODE (i.e., $T' = f(t,T)$), which can make it infeasible for many IVPs. That is why it is not typically used in large-scale or real-time simulations.

The absolute true errors of the estimates obtained with the third-order polynomial approximation for $h = 10$, 5 and 2.5 are tabulated in **Table 9.3** along with the true solution. The mean values of the tabulated absolute errors are 5.32×10^{-2}, 6.03×10^{-3}, and 6.56×10^{-4} for $h = 10$, 5, and 2.5, respectively. As the time step is halved, the mean errors are reduced by one-eighth due to the third-order

scheme, i.e., $\mathcal{O}\big((h/2)^3\big)$. It is noteworthy that the absolute true errors are much smaller compared to the first-order explicit or implicit Euler methods or the trapezoidal rule.

Table 9.3: Absolute true errors with the third-order Taylor polynomial approximation.

			Absolute True Error		
i	t_i	T_i	$h=10$	$h=5$	$h=2.5$
0	0	500			
1	10	436.306	2.994×10^{-2}	2.228×10^{-3}	2.340×10^{-4}
2	20	391.657	2.437×10^{-3}	1.581×10^{-3}	2.060×10^{-4}
3	30	361.063	4.455×10^{-2}	5.896×10^{-3}	6.890×10^{-4}
4	40	340.424	7.243×10^{-2}	8.583×10^{-3}	9.790×10^{-4}
5	50	326.647	8.287×10^{-2}	9.452×10^{-3}	9.792×10^{-4}
6	60	317.516	8.111×10^{-2}	9.024×10^{-3}	9.980×10^{-4}
7	70	311.493	7.280×10^{-2}	7.895×10^{-3}	8.590×10^{-4}
8	80	307.531	6.041×10^{-2}	6.513×10^{-3}	6.930×10^{-4}
9	90	304.931	4.830×10^{-2}	5.150×10^{-3}	5.320×10^{-4}
10	100	303.227	3.753×10^{-2}	3.942×10^{-3}	3.910×10^{-4}

Discussion: The procedure of constructing Taylor polynomial approximations for IVPs involves leveraging the differential equation to compute successive derivatives at the initial point, then building the polynomial from these derivatives. The steps to find an approximate solution to an IVP, $y' = f(t, y)$, using a Taylor polynomial of a specified degree can be described as follows:

1. Determine the order of the Taylor polynomial (n) desired (e.g., second-order $n = 2$, third-order $n = 3$, and so on.)
2. Compute the required derivatives of $y(t)$ at the initial point t_i by successively differentiating the differential equation and eliminating previous derivatives.
3. Use these derivatives to construct the Taylor polynomial centered at t_i.
4. Use this polynomial to approximate $y(t_{i+1}) = y_{i+1}$ near t_i:

$$p_n(y_i) = y_{i+1} = y_i + h\,y_i' + \frac{h^2}{2!}\,y_i'' + \frac{h^3}{3!}\,y_i''' + \ldots + \frac{h^n}{n!}y_i^{(n)} + \mathcal{O}(h^{n+1})$$

where $t_{i+1} - t_i = h$

For each term in the Taylor polynomial, we need to compute higher-order derivatives. In more complicated cases where $f(t, y)$ is a more complex function, computing these higher-order derivatives can become quite cumbersome and computationally expensive. This is why, in practice, Runge-Kutta methods are often preferred, as they achieve higher-order approximations without requiring the computation of these derivatives. Nonetheless, the Taylor polynomial approach is a powerful tool for understanding the local behavior of solutions to IVPs and can provide accurate approximations over short intervals, i.e., $h \approx 0$. It is also the foundation for many numerical methods, even if they do not compute the derivatives explicitly.

EXAMPLE 9.3: Runge-Kutta Method

Consider **Example 9.1**. Use the third- and fourth-order Runge-Kutta schemes to obtain numerical solutions for $h = 10$, 5, and 2.5, and discuss the results.

SOLUTION:

The initial value problem is

$$\frac{dT}{dt} = f(t,T) = -0.05(1 - 0.0005T)(T - 300), \quad T(0) = 500\,\text{K}$$

The third-order Runge-Kutta scheme

$$T_{i+1} = T_i + \frac{1}{6}\big(k_1 + 4k_2 + k_3\big) + \mathcal{O}(h^4)$$

where $t_{i+1/2} = t_i + h/2$, $t_{i+1} = t_i + h$, and T_{i+1} is the current step estimate. The k's are the intermediate slopes defined as follows:

$$k_1 = h\,f(t_i, T_i), \quad k_2 = h\,f(t_{i+1/2}, T_i + k_1/2), \quad \text{and} \quad k_3 = h\,f(x_{i+1}, T_i - k_1 + 2k_2)$$

The fourth-order Runge-Kutta scheme

$$T_{i+1} = T_i + \frac{1}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big) + \mathcal{O}(h^5)$$

where

$$\begin{aligned} k_1 &= h\,f(t_i, T_i), & k_2 &= h\,f(t_{i+1/2}, T_i + k_1/2), \\ k_3 &= h\,f(t_{i+1/2}, T_i + k_2/2), & \text{and} \quad k_4 &= h\,f(t_{i+1}, T_i + k_3) \end{aligned}$$

The true solution and the absolute true errors of the estimates obtained with the RK3 and RK4 schemes for $h = 10$ and 5 are presented in **Table 9.4**. For $h = 10$ and 5, the mean values of the tabulated absolute errors are respectively 0.149 and 0.0164 for RK3 and 0.0127 and 0.00675 for RK4. The absolute true errors are considerably smaller compared to the explicit or implicit Euler methods or the trapezoidal rule. By halving the step size, the absolute errors for the RK3 and RK4 schemes have been reduced by approximately $(1/2)^3$ and $(1/2)^4$, respectively.

Table 9.4: The true solution and absolute true errors using RK3 and RK4 schemes.

			RK3		RK4	
i	t_i	T_i	$h=10$	$h=5$	$h=10$	$h=5$
0	0	500				
1	10	436.306	0.13879	0.01573	0.01158	0.00063
2	20	391.657	0.21078	0.02361	0.01776	0.00095
3	30	361.063	0.23111	0.02564	0.01961	0.00105
4	40	340.424	0.21893	0.02411	0.01866	0.00099
5	50	326.647	0.19040	0.02084	0.01629	0.00086
6	60	317.516	0.15656	0.01706	0.01343	0.00071
7	70	311.493	0.12377	0.01345	0.01064	0.00056
8	80	307.531	0.09508	0.01030	0.00819	0.00043
9	90	304.931	0.07146	0.00773	0.00617	0.00033
10	100	303.227	0.05281	0.00570	0.00457	0.00024

Discussion: The Runge-Kutta methods are one-step numerical integration methods. That means, at each step, they estimate the current value y_{i+1} based only on the prior value y_i and evaluations of the function $f(t, y)$. The explicit RK methods improve upon Euler's method by taking multiple *slopes* (derivative estimates) at various points in the interval to compute a better average slope. This gives better accuracy without requiring higher derivatives, unlike Taylor polynomial approximations. Also note that any RK method requires fewer arithmetic operations than the approximation scheme obtained with the Taylor polynomial as depicted in **Example 9.2**.

Runge-Kutta methods give the analyst a lot of power and accuracy without much complexity. That's why RK4, for example, is often the go-to method for numerically solving IVPs. RK methods are generally stable for small step sizes and work well on a wide range of non-stiff ODEs. They can be naturally extended to solving systems of equations by simply applying the same RK logic to each component. Adaptive RK methods like Runge-Kutta-Fehlberg (RKF45) adjust the step size automatically to keep error under control, improving efficiency and reliability.

The explicit RK methods (presented here and in the textbook) are not suitable for stiff differential equations. For stiff problems, the step size must be extremely small to maintain stability, making RK methods inefficient. Using explicit RK methods on stiff equations can lead to instability or very slow computation. Implicit Euler and midpoint methods are implicit RK1 and RK2 schemes, respectively. Even for stiff problems, implicit multistep methods such as BDF methods are often preferred over implicit RK because they are more efficient for large systems and previous step estimates can be reused.

EXAMPLE 9.4: Runge-Kutta Method

Consider $y_1(t)$, $y_2(t)$, $y_3(t)$, and $y_4(t)$ representing concentrations of four interacting species in a biochemical system. The system equations is described by the following initial value problem:

$$\begin{aligned}
\frac{dy_1}{dt} &= k_2\, y_2(t)y_3(t) - k_1\, y_1(t), & y_1(0) &= 2\\
\frac{dy_2}{dt} &= k_1\, y_1(t) - k_2\, y_2(t)y_3(t) - k_3\, y_2^2(t), & y_2(0) &= 1\\
\frac{dy_3}{dt} &= k_5\, y_2(t)y_4(t) - k_4\, y_3(t), & y_3(0) &= 0.5\\
\frac{dy_4}{dt} &= k_6\, y_3^2(t) - k_5\, y_2(t)y_4(t) & y_4(0) &= 0.1
\end{aligned}$$

where k_1=1, k_2=0.5, k_3=0.1, k_4=0.7, k_5=0.3, and k_6=0.2. Obtain a numerical solution on [0,2] using the BDF2 scheme with h=0.1.

SOLUTION:

Employing the BDF2 scheme gives

$$\mathbf{y}_{i+1} = \frac{1}{3}\,(4\mathbf{y}_i - \mathbf{y}_{i-1}) + \frac{2h}{3}\,\mathbf{F}(t_{i+1}, \mathbf{y}_{i+1})$$

where $\mathbf{y}_i = \mathbf{y}(t_i)$, and

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t)\\ y_2(t)\\ y_3(t)\\ y_4(t)\end{bmatrix}, \qquad \mathbf{F}(t,\mathbf{y}) = \begin{bmatrix} k_2\, y_2y_3 - k_1\, y_1\\ k_1\, y_1 - k_2\, y_2y_3 - k_3\, y_2^2\\ k_5\, y_2y_4 - k_4\, y_3\\ k_6\, y_3^2 - k_5\, y_2y_4\end{bmatrix}$$

The residual vector is defined as follows:

$$\mathbf{R}(t_{i+1}, \mathbf{y}_{i+1}) = \mathbf{y}_{i+1} - \frac{1}{3}\,(4\mathbf{y}_i - \mathbf{y}_{i-1}) - \frac{2h}{3}\,\mathbf{F}(t_{i+1}, \mathbf{y}_{i+1})$$

which gives a system of four nonlinear algebraic equations with four unknowns (i.e., $y_1(t_{i+1})$, $y_2(t_{i+1})$, $y_3(t_{i+1})$, and $y_4(t_{i+1})$) due to the IVP (i.e., $\mathbf{F}(t,\mathbf{y})$) being nonlinear.

Next we expand the residual into a two-term Taylor series about $\mathbf{y}_{i+1}^{(p)}$, leading to

$$\mathbf{R}(t_{i+1}, \mathbf{y}_{i+1}^{(p+1)}) \cong \mathbf{R}(t_{i+1}, \mathbf{y}_{i+1}^{(p)}) + \left(\mathbf{I} - \frac{2h}{3}\frac{\partial \mathbf{F}}{\partial \mathbf{y}}(t_{i+1}, \mathbf{y}_{i+1}^{(p)})\right)(\mathbf{y}_{i+1}^{(p+1)} - \mathbf{y}_{i+1}^{(p)})$$

where the superscript "(p)" denotes the iteration step and $\partial\mathbf{F}/\partial\mathbf{y}$ is the Jacobian matrix found as

$$\mathbf{J}(t,\mathbf{y}) = \frac{\partial \mathbf{F}}{\partial \mathbf{y}} = \begin{bmatrix} -k_1 & k_2\, y_3 & k_2\, y_2 & 0\\ k_1 & -2k_3\, y_2 - k_2\, y_3 & -k_2\, y_2 & 0\\ 0 & k_5\, y_4 & -k_4 & k_5\, y_2\\ 0 & -k_5\, y_4 & 2k_6\, y_3 & -k_5\, y_2\end{bmatrix}$$

The system of nonlinear equations for the residuals will eventually converge to zero within a tolerance value. Thus, setting $\mathbf{R}(t_{i+1}, \mathbf{y}_{i+1}^{(p+1)}) = 0$ and solving it for $\mathbf{y}_{i+1}^{(p+1)}$, for $p \geqslant 0$, we find

$$\mathbf{y}_{i+1}^{(p+1)} = \mathbf{y}_{i+1}^{(p)} - \left(\mathbf{I} - \frac{2h}{3}\,\mathbf{J}(t_{i+1}, \mathbf{y}_{i+1}^{(p)})\right)^{-1}\left(\mathbf{y}_{i+1}^{(p)} - \frac{1}{3}\,(4\mathbf{y}_i - \mathbf{y}_{i-1}) - \frac{2h}{3}\,\mathbf{F}(t_{i+1}, \mathbf{y}_{i+1}^{(p)})\right)$$

where $\mathbf{y}_i$ and $\mathbf{y}_{i-1}$ are the converged (known) prior step estimates, and $\mathbf{y}_i^{(p)}$ is the current step estimate at the p'th iteration step. The most common and simple initial guess is chosen as the prior step estimate $\mathbf{y}_{i+1}^{(0)} = \mathbf{y}_i$, which should work well when the step size h is small, as the solution generally does not change rapidly.

Since BDF2 is a two-step method, we need both $\mathbf{y}_0$ and $\mathbf{y}_1$ to start the forward marching algorithm. In this regard, we should obtain an estimate for $\mathbf{y}_1$ using a second-order method, i.e., $\mathcal{O}(h^2)$. In this example, we will adopt the *trapezoidal rule*, resulting in

$$\mathbf{y}_1^{(p+1)} = \mathbf{y}_1^{(p)} - \left(\mathbf{I} - \frac{h}{2}\,\mathbf{J}(t_1, \mathbf{y}_1^{(p)})\right)^{-1} \left(\mathbf{y}_1^{(p)} - \mathbf{y}_0 - \frac{h}{2}\,\mathbf{F}(t_0, \mathbf{y}_0) - \frac{h}{2}\,\mathbf{F}(t_1, \mathbf{y}_1^{(p)})\right)$$

where $\mathbf{y}_0$ is the initial guess vector and $\mathbf{y}_1$ denotes the current step solution at $t = h$. In general, for the following time steps ($i \geqslant 1$), the BDF2 iteration process to solve a non-linear set of equations for the current step is continued until the convergence criterion is met, that is, $\|\mathbf{y}_{i+1}^{(p+1)} - \mathbf{y}_{i+1}^{(p)}\| < \varepsilon$. Then, once a converged estimate is found, the numerical solution is advanced to the next time step.

The numerical solutions up to $t = 2$ for y_1, y_2, y_3, and y_4 were obtained using the BDF2 scheme with $h = 0.1$ and $h = 0.05$ are presented in **Tables 9.5** and **9.6**, respectively. The absolute differences between the two solutions obtained with the time steps of $h = 0.1$ and $h = 0.05$ ($\mathbf{d} = |\mathbf{y}(h=0.1) - \mathbf{y}(h=0.05|$) are presented in **Tables 9.7**. We note that the average differences between the two solutions (over the solution interval) are of the order of 1.43×10^{-3}, 1.45×10^{-3}, 1.68×10^{-4}, and 1.10×10^{-4} for y_1, y_2, y_3, and y_4, respectively. In reality, the absolute error for $\mathbf{y}(h=0.05$ is much smaller than that of the average differences. We also note that the absolute differences show an increase up to $t \approx 0.9$, but decrease beyond this point.

Table 9.5: The numerical solution for $h = 0.1$.

i	t_i	$y_1(t_i)$	$y_2(t_i)$	$y_3(t_i)$	$y_4(t_i)$
0	0	2	1	0.5	0.1
1	0.1	1.834324	1.154017	0.469330	0.101447
2	0.2	1.685658	1.287742	0.441144	0.101899
3	0.3	1.552054	1.403238	0.415220	0.101500
4	0.4	1.431674	1.502532	0.391334	0.100392
5	0.5	1.322895	1.587481	0.369273	0.098707
6	0.6	1.224314	1.659754	0.348845	0.096563
7	0.7	1.134727	1.720829	0.329879	0.094062
8	0.8	1.053100	1.772019	0.312222	0.091292
9	0.9	0.978543	1.814482	0.295740	0.088330
10	1	0.910284	1.849245	0.280317	0.085237
11	1.1	0.847655	1.877214	0.265851	0.082068
12	1.2	0.790077	1.899194	0.252253	0.078864
13	1.3	0.737040	1.915894	0.239445	0.075662
14	1.4	0.688099	1.927942	0.227361	0.072488
15	1.5	0.642862	1.935897	0.215941	0.069366
16	1.6	0.600985	1.940250	0.205135	0.066314
17	1.7	0.562161	1.941438	0.194897	0.063343
18	1.8	0.526120	1.939847	0.185188	0.060464
19	1.9	0.492620	1.935820	0.175973	0.057684
20	2	0.461446	1.929661	0.167221	0.055007

Table 9.6: The numerical solution for $h = 0.05$.

i	t_i	$y_1(t_i)$	$y_2(t_i)$	$y_3(t_i)$	$y_4(t_i)$
0	0	2	1	0.5	0.1
2	0.1	1.834430	1.153912	0.469340	0.101440
4	0.2	1.686141	1.287256	0.441195	0.101866
6	0.3	1.552936	1.402340	0.415317	0.101437
8	0.4	1.432889	1.501284	0.391471	0.100302
10	0.5	1.324360	1.585967	0.369443	0.098596
12	0.6	1.225954	1.658050	0.349039	0.096436
14	0.7	1.136481	1.719003	0.330090	0.093924
16	0.8	1.054918	1.770124	0.312442	0.091148
18	0.9	0.980387	1.812562	0.295965	0.088183
20	1	0.912126	1.847333	0.280542	0.085091
22	1.1	0.849474	1.875338	0.266073	0.081924
24	1.2	0.791856	1.897373	0.252469	0.078724
26	1.3	0.738767	1.914142	0.239653	0.075527
28	1.4	0.689765	1.926272	0.227560	0.072360
30	1.5	0.644462	1.934315	0.216130	0.069245
32	1.6	0.602515	1.938761	0.205314	0.066200
34	1.7	0.563620	1.940044	0.195066	0.063236
36	1.8	0.527506	1.938549	0.185347	0.060365
38	1.9	0.493934	1.934618	0.176122	0.057592
40	2	0.462689	1.928553	0.167360	0.054923

Table 9.7: The absolute difference between two solutions found for $h = 0.1$ and $h = 0.05$.

i	t_i	$d_1(t_i)$	$d_2(t_i)$	$d_3(t_i)$	$d_4(t_i)$
0	0	0	0	0	0
1	0.1	1.060×10^{-4}	1.050×10^{-4}	1.000×10^{-5}	7.000×10^{-6}
2	0.2	4.830×10^{-4}	4.860×10^{-4}	5.100×10^{-5}	3.300×10^{-5}
3	0.3	8.820×10^{-4}	8.980×10^{-4}	9.700×10^{-5}	6.300×10^{-5}
4	0.4	1.215×10^{-3}	1.248×10^{-3}	1.370×10^{-4}	9.000×10^{-5}
5	0.5	1.465×10^{-3}	1.514×10^{-3}	1.700×10^{-4}	1.110×10^{-4}
6	0.6	1.640×10^{-3}	1.704×10^{-3}	1.940×10^{-4}	1.270×10^{-4}
7	0.7	1.754×10^{-3}	1.826×10^{-3}	2.110×10^{-4}	1.380×10^{-4}
8	0.8	1.818×10^{-3}	1.895×10^{-3}	2.200×10^{-4}	1.440×10^{-4}
9	0.9	1.844×10^{-3}	1.920×10^{-3}	2.250×10^{-4}	1.470×10^{-4} \$ 10
1	1.842×10^{-3}	1.912×10^{-3}	2.250×10^{-4}	1.460×10^{-4}	
11	1.1	1.819×10^{-3}	1.876×10^{-3}	2.220×10^{-4}	1.440×10^{-4}
12	1.2	1.779×10^{-3}	1.821×10^{-3}	2.160×10^{-4}	1.400×10^{-4}
13	1.3	1.727×10^{-3}	1.752×10^{-3}	2.080×10^{-4}	1.350×10^{-4}
14	1.4	1.666×10^{-3}	1.670×10^{-3}	1.990×10^{-4}	1.280×10^{-4}
15	1.5	1.600×10^{-3}	1.582×10^{-3}	1.890×10^{-4}	1.210×10^{-4}
16	1.6	1.530×10^{-3}	1.489×10^{-3}	1.790×10^{-4}	1.140×10^{-4}
17	1.7	1.459×10^{-3}	1.394×10^{-3}	1.690×10^{-4}	1.070×10^{-4}
18	1.8	1.386×10^{-3}	1.298×10^{-3}	1.590×10^{-4}	9.900×10^{-5}
19	1.9	1.314×10^{-3}	1.202×10^{-3}	1.490×10^{-4}	9.200×10^{-5}
20	2	1.243×10^{-3}	1.108×10^{-3}	1.390×10^{-4}	8.400×10^{-5}

The numerical solutions of y_1, y_2, y_3, and y_4 obtained using the BDF2 scheme with $h = 0.05$ are shown in **Figure 9.1**. We observe that while the species y_1, y_3, and y_4 decay to zero in time, y_2 asymptotically approaches 2.

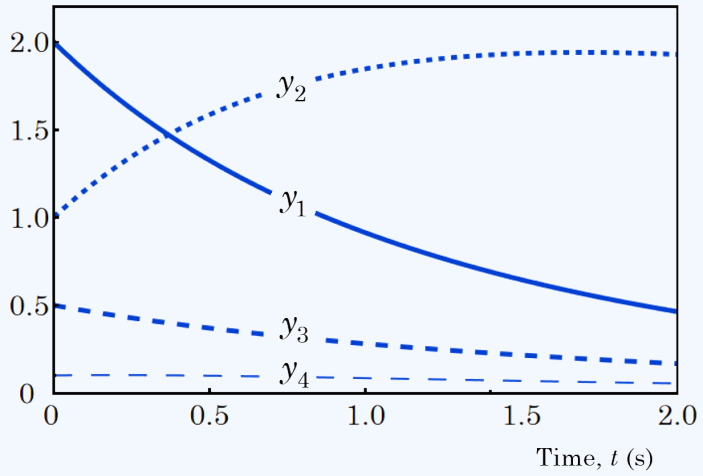

Figure 9.1

Discussion: The Backward Differentiation Formula of order 2 (BDF2) is a second-order accurate implicit method used to numerically solve ordinary differential equations (ODEs), particularly stiff equations. BDF2 is implicit, meaning that it requires solving a system of equations at each time step, which makes it more stable than explicit methods, especially for stiff IVPs.

BDF2 scheme has a local truncation error of $\mathcal{O}(h^3)$, giving global accuracy of $\mathcal{O}(h^2)$. We need a method to generate the second value ($\mathbf{y}_1$) after $\mathbf{y}_0$. It is A-stable (its region of absolute stability includes the entire left-half of the complex plane), making it well suited for stiff equations. However, it is not L-stable, which means it may not dampen fast transients as effectively as BDF1 (i.e., backward Euler scheme). In addition, each step requires the solution of a non-linear equation (often *via* Newton's method), making it computationally more involved than explicit methods.