1 Question 1

Suppose \( y(t) \) is the exact solution of the initial value problem
\[
y'(t) = f(t, y(t)),
\]
\[
y(0) = y_0,
\]
and \( u(t) \) is any approximation to \( y(t) \) with \( u(0) = y(0) \). Define the error \( e(t) = y(t) - u(t) \).

(a) Show that \( e(t) \) satisfies the initial value problem
\[
e'(t) = f(t, u(t) + e(t)) - u'(t)
\]
\[
e(0) = 0
\]

(b) Suppose \( f(t, y) = \lambda y \) for some constant \( \lambda \). Solve the initial value problem from (a) exactly to show that \( u(t) + e(t) = y(t) \).

Question (a)

Since the error is defined to be \( e(t) = y(t) - u(t) \), taking derivatives of both sides gives us:
\[
e'(t) = y'(t) - u'(t)
\]
\[
= f(t, y(t)) - u'(t)
\]
\[
= f(t, u(t) + e(t)) - u'(t)
\]
And clearly, \( e(0) = y(0) - u(0) = 0 \).

Question (b)

If \( f(t, y) = \lambda y \), the IVP becomes
\[
e'(t) = \lambda(u(t) + e(t)) - u'(t)
\]
To show that \( u(t) + e(t) = y(t) \), we first note that \( u(0) + e(0) = u(0) = y(0) \), and furthermore:
\[
u'(t) + e'(t) - y'(t) = y'(t) + \lambda(u(t) + e(t)) - y'(t) - \lambda y(t)
\]
\[
= \lambda(u(t) + e(t) - y(t))
\]
(1)

Therefore the function \( z(t) = u(t) + e(t) - y(t) \) satisfies the equation:
\[
z'(t) = \lambda z(t), \quad z(0) = 0
\]
Which we can solve as follows:
\[
z'(t)/z(t) = \lambda \implies \ln(z(t)) = \lambda t + c \implies z(t) = k e^{\lambda t}
\]
Enforcing the initial condition \( z(0) = 0 \), we get \( k = 0 \). Therefore \( z(t) = 0 = u(t) + e(t) - y(t) \) for all \( t \). So we can conclude \( u(t) + e(t) = y(t) \).
2 Question 2

Define a family of explicit Runge-Kutta methods parametrized by order \( p \), by applying \( p - 1 \) passes of deferred correction to \( p \) steps of Euler’s method. I.e. starting from \( u_n \), define the uncorrected solution by

\[
u_{n+j+1}^1 = u_{n+j}^1 + hf(t_{n+j}, u_{n+j}^1)
\]

for \( 0 \leq j \leq p - 1 \). Let \( u(t) = U_1(t) \) be the degree-\( p \) polynomial that interpolates the \( p + 1 \) values \( u_{n+j}^1 \) at the \( p + 1 \) points \( t = t_{n+j} \) for \( 0 \leq j \leq p \). Solve the error equation from question 1 by Euler’s method, yielding approximate errors \( e_{n+1}^1, e_{n+2}^1, \ldots, e_{n+p}^1 \). Produce a second-order accurate corrected solution

\[
u_{n+j}^2 = u_{n+j}^1 + e_{n+j}^1
\]

for \( 1 \leq j \leq p \). Repeat the procedure to produce \( u_{n+j}^2, \ldots, u_{n+p}^p \).

(a) Verify that \( p = 1 \) gives Euler’s method.

(b) For \( p = 2 \) express your method as a Runge-Kutta method in the form

\[
k_1 = f(t_n, u_n)
\]

\[
k_2 = f(t_n + c_22h, u_n + 2ha_{21}k_1)
\]

\[
k_3 = f(t_n + c_32h, u_n + 2h(a_{31}k_1 + a_{32}k_2))
\]

\[
u_{n+2} = u_n + 2h(b_1k_1 + b_2k_2 + b_3k_3).
\]

Find all the constants \( c_i, a_{ij} \) and \( b_j \) and arrange them in a Butcher array.

(c) For \( p = 2 \), ignore the \( t \) argument of \( f(t, u) \) and Taylor expand \( k_2(h) \) and \( k_3(h) \) to \( O(h^2) \). Show that your method has local truncation error \( \tau = O(h^2) \) and find the coefficient of the \( O(h^2) \) term.

(d) For arbitrary \( p \), verify that your method is equivalent to using fixed point iteration to solve an implicit Runge-Kutta method.

Question (a)

When \( p = 1 \), we do not apply any deferred correction steps. Therefore, we simply have

\[
u_{n+j+1} = u_{n+j} + hf(t_{n+j}, h_{n+j})
\]

And that is Euler’s Method.

Question (b)

When \( p = 2 \), we first apply Euler’s method twice to get:

\[
u_{n+1}^1 = u_n + hf(t_n, u_n)
\]

\[
u_{n+2}^1 = u_{n+1}^1 + hf(t_{n+1}, u_{n+1}^1)
\]

Next we apply deferred correction. As in problem 3, we know that

\[
e'(t) = f(t, u(t) + e(t)) - u'(t)
\]

\[
e(t_n) = 0
\]
so applying Euler’s method:

\[ e_{n+1}^1 = e(t_n) + h(f(t_n, e(t_n) + u(t_n)) - u'(t_n)) \]
\[ = hf(t_n, u_n) - hu'(t_n) \]

Next, we estimate \( u'(t_n) \) via Lagrange interpolation. So, let \( U_1(t) \) be the Lagrange polynomial going through \((t_n, u_n), (t_{n+1}, u_{n+1}^1), (t_{n+2}, u_{n+2}^1)\), then:

\[
U_1(t) = \frac{(t - t_n)(t - t_{n+1})}{(t_{n+2} - t_n)(t_{n+2} - t_{n+1})} u_{n+2}^1 + \frac{(t - t_n)(t - t_{n+2})}{(t_{n+1} - t_n)(t_{n+1} - t_{n+2})} u_{n+1}^1 + \frac{(t - t_{n+1})(t - t_{n+2})}{(t_n - t_{n+1})(t_n - t_{n+2})} u_n
\]

\[
U_1'(t) = \frac{(t - t_n) + (t - t_{n+1})}{(t_{n+2} - t_n)(t_{n+2} - t_{n+1})} u_{n+2}^1 + \frac{(t - t_n) + (t - t_{n+2})}{(t_{n+1} - t_n)(t_{n+1} - t_{n+2})} u_{n+1}^1 + \frac{(t - t_{n+1}) + (t - t_{n+2})}{(t_n - t_{n+1})(t_n - t_{n+2})} u_n
\]

So,

\[
e_{n+1}^1 = hf(t_n, u_n) + \frac{1}{2} \left( u_{n+2}^1 - 4u_{n+1}^1 + 3u_n \right)
\]

Similarly, applying the next Euler’s step:

\[
e_{n+2}^1 = e_{n+1}^1 + h(f(t_{n+1}, e_{n+1}^1 + u_{n+1}^1) - u'(t_{n+1}))
\]

\[
e_{n+2}^1 = e_{n+1}^1 + h(f(t_{n+1}, e_{n+1}^1 + u_{n+1}^1)) + \frac{u_n - u_{n+2}^1}{2}
\]

Now, we update our new estimates for \( u_{n+2} \):

\[ u_{n+2}^2 = u_{n+2}^1 + e_{n+2}^1 \]

So, let’s simplify what we have done. Let

\[
k_1 = f(t_n, u_n)
\]
\[
k_2 = f(t_{n+1}, u_{n+1}^1)
\]
\[
k_3 = f(t_{n+1}, e_{n+1}^1 + u_{n+1}^1)
\]

So

\[
u_{n+2}^2 = u_{n+2}^1 + e_{n+2}^1
\]
\[= u_{n+2}^1 + e_{n+1}^1 + hk_3 + \frac{u_n - u_{n+2}^1}{2} \]
\[= u_{n+2}^1 + hk_1 + \frac{1}{2}(u_{n+2}^1 - 4u_{n+1}^1 + 3u_n) + hk_3 + \frac{u_n - u_{n+2}^1}{2} \]
\[= u_{n+2}^1 - 2u_{n+1}^1 + 2u_n + hk_1 + hk_3 \]
\[= hk_2 - u_{n+1}^1 + 2u_n + hk_1 + hk_3 \]
\[= u_n - hk_1 + 2u_n + hk_1 + hk_3 \]
\[= u_n + hk_2 + hk_3 \]
Furthermore, we see that
\[
k_2 = f(t_{n+1}, u_{n+1}^1) \\
= f(t_n + h, u_n + h k_1) \\
k_3 = f(t_{n+1}, e_{n+1}^1 + u(t_{n+1})) \\
= f(t_n + h, hk_1 + \frac{1}{2}(u_{n+2}^1 - 4u_{n+1}^1 + 3u_n) + u_{n+1}^1) \\
= f(t_n + h, hk_1 + \frac{1}{2}(u_{n+1}^1 + h k_2 - 2u_{n+1}^1 + 3u_n)) \\
= f(t_n + h, u_n + \frac{1}{2}(hk_2 + h k_1))
\]

Therefore, we conclude that \(c_2 = a_{21} = c_3 = b_2 = b_3 = \frac{1}{2}, a_{31} = a_{32} = \frac{1}{4}\), and \(b_1 = 0\).

\[
\begin{array}{c|ccc}
0 & 0 & & \\
\frac{1}{2} & \frac{1}{2} & 0 & \\
\frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 \\
0 & \frac{1}{2} & \frac{1}{2} & \\
\end{array}
\]

Question (c)

Let us Taylor expand \(k_2(h)\) and \(k_3(h)\) to get (ignoring the \(t\) argument):
\[
k_2(h) = f(u_n + h k_1) \\
k_3(h) = f(u_n + \frac{1}{2}(hk_2 + h k_1)) \\
k_2(h) = k_2(0) + h k_2'(0) + \frac{h^2}{2} k_2''(0) + O(h^3) \\
k_3(h) = k_3(0) + h k_3'(0) + \frac{h^2}{2} k_3''(0) + O(h^3)
\]

Now systematically compute all these derivatives (similar to how we did RK order conditions):
\[
k_2(h) = f(u_n + h k_1) \\
k_2(0) = f(u_n) = k_1 \\
k_2'(h) = k_1 f'(u_n + h k_1) \\
k_2'(0) = k_1 f'(u_n) = f f' \\
k_2''(h) = k_1^2 f''(u_n + h k_1) \\
k_2''(0) = k_1^2 f''(u_n) = f^2 f''
\]

Put it all together:
\[
k_2(h) = f + h f' + \frac{h^2}{2} f^2 f'' + O(h^3)
\]
Similarly for $k_3$:

$$k_3(h) = f(u_n + \frac{1}{2}(hk_2 + hk_1))$$

$$k_3(0) = f(u_n) = k_1$$

$$k_3'(h) = \frac{1}{2}(k_2 + hk'_2 + k_1)f'(u_n + \frac{1}{2}(hk_2 + hk_1))$$

$$k_3'(0) = \frac{1}{2}(k_2(0) + k_1)f'(u_n)$$

$$= k_1 f'(u_n) = ff'$$

$$k_3''(h) = \frac{1}{2}(k_2 + hk''_2 + k_1)f'(u_n + \frac{1}{2}(hk_2 + hk_1)) + \frac{1}{4}(k_2 + hk'_2 + k_1)^2 f''(u_n + \frac{1}{2}(hk_2 + hk_1))$$

$$k_3''(0) = k_2(0)f'(u_n) + \frac{1}{4}(k_2(0) + k_1)^2 f''(u_n)$$

$$= k_1 f''(u_n)^2 + \frac{1}{4}(2k_1)^2 f''(u_n)$$

$$= ff'' + f^2 f''$$

Put it all together:

$$k_3(h) = f + hf f' + \frac{h^2}{2} (f(f')^2 + f^2 f'') + O(h^3)$$

Now the truncation error is:

$$\tau = \frac{y_{n+2} - y_n}{2h} - b_1 k_1 - b_2 k_2 - b_3 k_3$$

$$= \frac{y_n' + (2h)y_n'' + \frac{(2h)^2}{2} y_n'''}{2h} + O(h^4) - \frac{1}{2} k_2 - \frac{1}{2} k_3$$

$$= f + hf f' + \frac{2}{3} h^2 y''' + O(h^3) - \frac{1}{2} \left( f + hf f' + \frac{h^2}{2} f f'' + O(h^3) \right) + \ldots$$

$$\cdots - \frac{1}{2} \left( f + hf f' + \frac{h^2}{2} (f(f')^2 + f^2 f'') + O(h^3) \right)$$

$$\underbrace{\frac{1}{k_3}}_{h^2 f (f')^2 + \text{terms}}$$

$$\underbrace{\frac{1}{k_3}}_{h^2 f f''}$$

$$= f + hf f' + \frac{2}{3} h^2 (f(f')^2 + f^2 f'') - hf f' - \frac{1}{2} h^2 f^2 f'' - \frac{1}{4} h^2 f (f')^2 + O(h^3)$$

$$= \frac{5}{12} h^2 f (f')^2 + \frac{1}{6} h^2 f^2 f''$$

$$= O(h^2)$$

**Question (d)**

(See IDEC Handout - Fixed point equivalent)

Since $u^2$ is built from $u^1$ by:

$$e_{n+j+1} = e_{n+j} + h[f(t_{n+j}, u_{n+j}^1 + e_{n+j}) - U'(t_{n+j})]$$

$$u_{n+j}^2 = u_{n+j}^1 + e_{n+j}$$

Deferred correction is a fixed point iteration of the form
\[
\begin{bmatrix}
 u_{n+1}^2 \\
 u_{n+2}^2 \\
 \vdots \\
 u_{n+p}^2 
\end{bmatrix} = \begin{bmatrix}
 u_{n+1}^1 \\
 u_{n+2}^1 \\
 \vdots \\
 u_{n+p}^1 
\end{bmatrix} + \begin{bmatrix}
 e_{n+1} \\
 e_{n+2} \\
 \vdots \\
 e_{n+p} 
\end{bmatrix} = G \left( \begin{bmatrix}
 u_{n+1}^1 \\
 u_{n+2}^1 \\
 \vdots \\
 u_{n+p}^1 
\end{bmatrix} \right)
\]

or \( U^2 = G(U^1) \).

In the limit where \( U^k \to U \), \( U \) must satisfy \( U = G(U) \), or
\[
E = U^2 - U^1 = 0
\]

Equivalently:
\[
e_{n+j} \equiv 0
\]
so that
\[
0 = 0 + h[f(t_{n+j}, u_{n+j}) - U'(t_{n+j})]
\]
and
\[
U'(t_{n+j}) = f(t_{n+j}, u_{n+j}) \quad 1 \leq j \leq p
\]

Here \( U(t) \) is the interpolating polynomial satisfying
\[
U(t_{n+j}) = u_{n+j}
\]
so that
\[
U(t) = \sum_{j=0}^{p} L_j(t)u_{n+j}
\]
and
\[
U'(t_{n+j}) = \frac{1}{h} \sum_{k=0}^{p} d_{jk}u_{n+k}
\]
for some dimensionless differentiation constants \( d_{jk} \). Thus deferred correction is a fixed point iteration for solving
\[
\frac{1}{h} \sum_{k=0}^{p} d_{jk}u_{n+k} = f(t_{n+j}, u_{n+j}) \quad 1 \leq j \leq p \tag{2}
\]

It remains to show that (1) is an implicit Runge-Kutta method with \( p \) stages
\[
k_j = f(t_{n+j}, u_{n+j}) \quad 1 \leq j \leq p
\]

We have the below since differentiating a constant gives 0.
\[
\sum_{k=0}^{p} d_{jk} = 0
\]
Hence:
\[
\sum_{k=0}^{p} d_{jk}u_n = 0
\]
So combining with (1), \( u \) must satisfy:
\[
\sum_{k=0}^{p} d_{jk}(u_{n+k} - u_n) = hf(t_{n+j}, u_{n+j}) = hk_j
\]

since the \( k = 0 \) gives us \( d_{j0}(u_n - u_n) = 0 \), giving use a square system of equations to solve for \((u_{n+k} - u_n)\):
\[
\sum_{k=1}^{p} d_{jk}(u_{n+k} - u_n) = hk_j
\]

If we define \( c_{ij} \) to be the elements of the inverse matrix \( C = D^{-1} \) to the square \( p \times p \) matrix \( D \) with elements \( d_{ij} \) and apply to both sides, we can extract the updates:
\[
u_{n+k} - u_n = h \sum_{j=1}^{p} c_{kj}k_j
\]

and rewrite \( k_j \) as:
\[
k_j = f(t_{n+j}, u_{n+j}) = f(t_{n+j}, u_n + ph \sum_{r=1}^{p} (c_{jr}/p)k_r)
\]

So we identify this as a \( p \)-stage implicit Runge-Kutta method with stepsize \( ph \).

3 Question 3

Write, test and debug a matlab function

```matlab
function yb = idec(a, b, ya, f, r, p, n)
% a,b: interval endpoints with a < b
% ya: vector y(a) of initial conditions
% f: function handle f(t, y) to integrate (y is a vector)
% r: parameters to f
% p: number of euler substeps / correction passes
% n: number of time steps
% yb: output approximation to the final solution vector y(b)

which approximates the final solution vector \( y(b) \) of the vector initial value problem
\[
y' = f(t, y, r)
y(a) = y_a
\]

by the method you derived in problem 4, with \( u_0 = y_a \).

(a) Use idec.m with orders \( p = 1 \) through 7 and \( N = 10000, 20000, 40000 \) and 80000 steps to approximate the final solution vector \( u(T) \) of the initial value problem derived in problem 4 of problem set 8. Tabulate the errors
\[
E_{pN} = \max_{1 \leq j \leq 4} |u_j(T) - u_j(0)|.
\]

Estimate the constant \( C_p \) such that the error behaves like \( C_p h^p \).

(b) Measure the CPU time for each run and estimate the total CPU time necessary to obtain an orbit which is periodic to three-digit, six-digit and twelve-digit accuracy.

(c) Plot some inaccurate solutions and some accurate solutions and draw conclusions about values of the order \( p \) which give three, six or twelve digits of accuracy for minimal CPU time.
Question (a)

The solution idec.m is embedded. We store our system \( f \) in moonode.m.

We provide the function idecToTheMoon.m which solves the satellite problem in problem set 8. This code takes \( p \) as a single argument, and finds the error with that \( p \) and \( N = 10000, 20000, 40000, 80000 \). Then it approximates the slopes in our fit lines.

Below is the error table (\( E_{pN} = \max_{1 \leq j \leq 4} |u_j(T) - u_j(0)| \))

<table>
<thead>
<tr>
<th>( N )</th>
<th>( p )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>1</td>
<td>41.803</td>
<td>1.614</td>
<td>2.010</td>
<td>0.8409</td>
<td>8.455 \times 10^{-2}</td>
<td>4.196 \times 10^{-2}</td>
<td>6.268 \times 10^{-4}</td>
</tr>
<tr>
<td>20000</td>
<td>2.761</td>
<td>1.507</td>
<td>1.426</td>
<td>0.1327</td>
<td>3.210 \times 10^{-3}</td>
<td>5.600 \times 10^{-3}</td>
<td>2.100 \times 10^{-3}</td>
<td></td>
</tr>
<tr>
<td>40000</td>
<td>2.021</td>
<td>1.370</td>
<td>0.388</td>
<td>9.160 \times 10^{-3}</td>
<td>9.800 \times 10^{-5}</td>
<td>8.389 \times 10^{-6}</td>
<td>9.444 \times 10^{-8}</td>
<td></td>
</tr>
<tr>
<td>80000</td>
<td>1.740</td>
<td>1.026</td>
<td>4.319 \times 10^{-1}</td>
<td>5.872 \times 10^{-4}</td>
<td>2.911 \times 10^{-6}</td>
<td>4.110 \times 10^{-7}</td>
<td>1.981 \times 10^{-7}</td>
<td></td>
</tr>
</tbody>
</table>

Below are the estimates of \( C_p \) such that the error behaves like \( C_p h^p \).

<table>
<thead>
<tr>
<th>( p )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.772 \times 10^4</td>
<td>1.430 \times 10^5</td>
<td>3.180 \times 10^8</td>
<td>9.620 \times 10^{10}</td>
<td>5.830 \times 10^{12}</td>
<td>1.700 \times 10^{15}</td>
<td>1.487 \times 10^{15}</td>
</tr>
</tbody>
</table>

Some comments:

- As we double \( N \) (equivalently halve \( h \)), we see that the corresponding error is not decreasing in some systematic way (particularly for the lower \( p \)). This suggests we are not really seeing asymptotic behavior so it’s impossible to extrapolate from this data things like how long it takes to get three digit accuracy. This suggests that we should run more cases (increase \( N \)) until we start to see convergence.

- \( p = 7 \) seems to achieve the maximum error level (\( \approx 10^{-7} \)) almost immediately and doesn’t improve. So this suggests we couldn’t get something like 12 digit accuracy no matter what unless we use higher-precision arithmetic.

Question (b)

The CPU time for each run:

<table>
<thead>
<tr>
<th>( N )</th>
<th>( p )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>1</td>
<td>0.2092</td>
<td>0.8705</td>
<td>1.9132</td>
<td>3.3952</td>
<td>5.5645</td>
<td>8.2326</td>
<td>11.6935</td>
</tr>
<tr>
<td>20000</td>
<td>0.0483</td>
<td>0.2001</td>
<td>0.4711</td>
<td>0.8586</td>
<td>1.4402</td>
<td>2.0204</td>
<td>2.8860</td>
<td></td>
</tr>
<tr>
<td>40000</td>
<td>0.0968</td>
<td>0.4208</td>
<td>0.9438</td>
<td>1.7397</td>
<td>2.7893</td>
<td>4.1423</td>
<td>5.7881</td>
<td></td>
</tr>
<tr>
<td>80000</td>
<td>0.2092</td>
<td>0.8705</td>
<td>1.9132</td>
<td>3.3952</td>
<td>5.5645</td>
<td>8.2326</td>
<td>11.6935</td>
<td></td>
</tr>
</tbody>
</table>

Estimates on CPU time to get specified accuracy (as noted above, these are naive estimates which couldn’t actually be achieved):

<table>
<thead>
<tr>
<th>acc</th>
<th>( p )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-digit</td>
<td>2.621 \times 10^2</td>
<td>14.63</td>
<td>6.0234</td>
<td>2.921</td>
<td>1.744</td>
<td>2.054</td>
<td>2.527</td>
<td></td>
</tr>
<tr>
<td>6-digit</td>
<td>2.621 \times 10^2</td>
<td>4.626 \times 10^2</td>
<td>6.0234 \times 10^4</td>
<td>1.6431 \times 10^4</td>
<td>6.945</td>
<td>6.496</td>
<td>6.780</td>
<td></td>
</tr>
<tr>
<td>12-digit</td>
<td>2.621 \times 10^4</td>
<td>4.626 \times 10^6</td>
<td>6.0234 \times 10^8</td>
<td>5.1959 \times 10^2</td>
<td>1.100 \times 10^2</td>
<td>6.496 \times 10^1</td>
<td>4.879 \times 10^1</td>
<td></td>
</tr>
</tbody>
</table>
Question (c)

Some plots of the trajectories are provided below (for each $p$ and $N$, $p > 4$ omitted since they're all really good).
Looking at the data given in (b) - for three digit accuracy, \( p = 5 \) is fastest. For six digit accuracy, \( p = 6 \) is fastest. For twelve digit accuracy, \( p = 7 \) is fastest.

If you run the solution `idecToTheMoon(p)` for some given \( p \), the resulting output will look like
the following (for example $p = 2$):

```python
>>> idecToTheMoon(2)
p =
2
errors =
  1.614391472108695
  1.50721472488082
  1.370484330041670
  1.026408928083552
errorslope =
  1.430081749578059e+05
durations =
  0.109044391000000
  0.214791086000000
  0.433213913000000
  0.849915735000000
threeDigitDuration: 14.7601
sixDigitDuration: 466.756
twelveDigitDuration: 466756
```

4 Question 4

(BFB 5.4.31) Show that Heun’s method can be expressed in difference form as

\[
\begin{align*}
  w_0 &= \alpha, \\
  k_1 &= f(t_i, w_i), \\
  k_2 &= f(t_i + \frac{1}{3}h, w_i + \frac{1}{3}k_1), \\
  k_3 &= f(t_i + \frac{2}{3}h, w_i + \frac{2}{3}k_2), \\
  w_{i+1} &= w_i + \frac{h}{4}(k_1 + 3k_3)
\end{align*}
\]

for each $i = 0, \ldots, N - 1$.

**Solution**

The Heun’s method is given by

\[
\begin{align*}
  w_0 &= \alpha, \\
  w_{i+1} &= w_i + \frac{h}{4}\left(f(t_i, w_i) + 3f\left(t_i + \frac{2h}{3}, w_i + \frac{2h}{3}f\left(t_i + \frac{h}{3}, w_i + \frac{h}{3}f(t_i, w_i)\right)\right)\right)
\end{align*}
\]

For each step $i$, we define $k_1 = f(t_i, w_i)$, then the Heun’s method can be written as:

\[
\begin{align*}
  w_0 &= \alpha, \\
  k_1 &= f(t_i, w_i), \\
  w_{i+1} &= w_i + \frac{h}{4}\left(k_1 + 3f\left(t_i + \frac{2h}{3}, w_i + \frac{2h}{3}f\left(t_i + \frac{h}{3}, w_i + \frac{h}{3}f(t_i, w_i)\right)\right)\right)
\end{align*}
\]
Then we can define $k_2$,

\[
\begin{align*}
w_0 &= \alpha, \\
k_1 &= f(t_i, w_i), \\
k_2 &= f\left( t_i + \frac{h}{3}, w_i + \frac{h}{3} k_1 \right), \\
w_{i+1} &= w_i + \frac{h}{4} \left( k_1 + 3 f\left( t_i + \frac{2h}{3}, w_i + \frac{2h}{3} k_2 \right) \right).
\end{align*}
\]

And define $k_3$,

\[
\begin{align*}
\begin{align*}
w_0 &= \alpha, \\
k_1 &= f(t_i, w_i), \\
k_2 &= f\left( t_i + \frac{h}{3}, w_i + \frac{h}{3} k_1 \right), \\
k_3 &= f\left( t_i + \frac{2h}{3}, w_i + \frac{2h}{3} k_2 \right), \\
w_{i+1} &= w_i + \frac{h}{4} (k_1 + 3k_3).
\end{align*}
\end{align*}
\]

This is what we want to show.