**Question 1**

1. The following tables compares the exact solution and numerical solution at each timestep, as well as the number of iterations required to reach 10-digit accuracy. The left is with functional iteration and the right is with Newton’s method. Newton’s method speeds up the convergence by requiring fewer iterations (though note that a newton iteration is twice as expensive!)

<table>
<thead>
<tr>
<th>$t_i$</th>
<th>$w_i$</th>
<th>iter</th>
<th>$y(t_i)$</th>
<th>$t_i$</th>
<th>$w_i$</th>
<th>iter</th>
<th>$y(t_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1.02755910580</td>
<td>0</td>
<td>1.02755910580</td>
<td>0.01</td>
<td>1.02755910580</td>
<td>0</td>
<td>1.02755910580</td>
</tr>
<tr>
<td>0.02</td>
<td>1.05589929264</td>
<td>0</td>
<td>1.05589929264</td>
<td>0.02</td>
<td>1.05589929264</td>
<td>0</td>
<td>1.05589929264</td>
</tr>
<tr>
<td>0.03</td>
<td>1.08506614213</td>
<td>5</td>
<td>1.085066130050</td>
<td>0.03</td>
<td>1.08506614213</td>
<td>2</td>
<td>1.085066130050</td>
</tr>
<tr>
<td>0.04</td>
<td>1.1151093220</td>
<td>5</td>
<td>1.115109295567</td>
<td>0.04</td>
<td>1.1151093220</td>
<td>2</td>
<td>1.115109295567</td>
</tr>
<tr>
<td>0.05</td>
<td>1.14608312750</td>
<td>5</td>
<td>1.14608308389</td>
<td>0.05</td>
<td>1.14608312750</td>
<td>2</td>
<td>1.14608308389</td>
</tr>
<tr>
<td>0.06</td>
<td>1.17804706142</td>
<td>5</td>
<td>1.178046997403</td>
<td>0.06</td>
<td>1.17804706142</td>
<td>2</td>
<td>1.178046997403</td>
</tr>
<tr>
<td>0.07</td>
<td>1.21106652260</td>
<td>5</td>
<td>1.211066434160</td>
<td>0.07</td>
<td>1.21106652260</td>
<td>2</td>
<td>1.211066434160</td>
</tr>
<tr>
<td>0.08</td>
<td>1.24521361129</td>
<td>5</td>
<td>1.245213493530</td>
<td>0.08</td>
<td>1.24521361129</td>
<td>2</td>
<td>1.245213493530</td>
</tr>
<tr>
<td>0.09</td>
<td>1.2805680773</td>
<td>5</td>
<td>1.280567924202</td>
<td>0.09</td>
<td>1.2805680773</td>
<td>2</td>
<td>1.280567924202</td>
</tr>
<tr>
<td>0.1</td>
<td>1.31721844197</td>
<td>5</td>
<td>1.317218246057</td>
<td>0.1</td>
<td>1.31721844197</td>
<td>2</td>
<td>1.317218246057</td>
</tr>
<tr>
<td>0.11</td>
<td>1.35526333358</td>
<td>5</td>
<td>1.355263085552</td>
<td>0.11</td>
<td>1.35526333358</td>
<td>2</td>
<td>1.355263085552</td>
</tr>
<tr>
<td>0.12</td>
<td>1.39481308713</td>
<td>5</td>
<td>1.394812775322</td>
<td>0.12</td>
<td>1.39481308713</td>
<td>2</td>
<td>1.394812775322</td>
</tr>
<tr>
<td>0.13</td>
<td>1.43599167374</td>
<td>5</td>
<td>1.435991283350</td>
<td>0.13</td>
<td>1.43599167374</td>
<td>2</td>
<td>1.435991283350</td>
</tr>
<tr>
<td>0.14</td>
<td>1.47893904452</td>
<td>5</td>
<td>1.478938556666</td>
<td>0.14</td>
<td>1.47893904452</td>
<td>2</td>
<td>1.478938556666</td>
</tr>
<tr>
<td>0.15</td>
<td>1.52381400102</td>
<td>5</td>
<td>1.523813391315</td>
<td>0.15</td>
<td>1.52381400102</td>
<td>2</td>
<td>1.523813391315</td>
</tr>
<tr>
<td>0.16</td>
<td>1.57079774040</td>
<td>5</td>
<td>1.570796977072</td>
<td>0.16</td>
<td>1.57079774040</td>
<td>2</td>
<td>1.570796977072</td>
</tr>
<tr>
<td>0.17</td>
<td>1.62009827561</td>
<td>6</td>
<td>1.62009716704</td>
<td>0.17</td>
<td>1.62009827561</td>
<td>2</td>
<td>1.62009716704</td>
</tr>
<tr>
<td>0.18</td>
<td>1.6719560027</td>
<td>6</td>
<td>1.67195472218</td>
<td>0.18</td>
<td>1.6719560027</td>
<td>2</td>
<td>1.67195472218</td>
</tr>
<tr>
<td>0.19</td>
<td>1.7266507929</td>
<td>6</td>
<td>1.726649254948</td>
<td>0.19</td>
<td>1.7266507929</td>
<td>2</td>
<td>1.726649254948</td>
</tr>
<tr>
<td>0.2</td>
<td>1.78451113940</td>
<td>6</td>
<td>1.784509169260</td>
<td>0.2</td>
<td>1.78451113940</td>
<td>2</td>
<td>1.784509169260</td>
</tr>
</tbody>
</table>

The MATLAB code that generated this is attached as `numberone.m`. 
Question 2

2. (a) Consider the identity:

\[ y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} f(t, y(t)) \, dt \]

To derive two-step Adams-Bashforth, we replace \( f(t, y(t)) \) with the linear Lagrange polynomial \( P_1(t) \) interpolating \( f(t, y) \) at \( (t_{i-1}, y(t_{i-1})) \) and \( (t_i, y(t_i)) \).

Constructing \( P_1(t) \):

\[ P_1(t) = \frac{t - t_{i-1}}{t_i - t_{i-1}} f(t_i, y(t_i)) + \frac{t - t_i}{t_{i-1} - t_i} f(t_{i-1}, y(t_{i-1})) \]

Let \( h_0 = t_i - t_{i-1} \) and \( h_1 = t_{i+1} - t_i \):

Integrating \( P_1(t) \) from \( t_i \) to \( t_{i+1} \):

\[
\int_{t_i}^{t_{i+1}} P_1(t) \, dt = \frac{f(t_i, y(t_i))}{t_i - t_{i-1}} \int_{t_i}^{t_{i+1}} (t - t_{i-1}) \, dt + \frac{f(t_{i-1}, y(t_{i-1}))}{t_{i-1} - t_i} \int_{t_i}^{t_{i+1}} (t - t_i) \, dt
\]

\[
= \frac{f(t_i, y(t_i))}{h_0} (h_0 + h_1) \frac{h_0^2 - h_1^2}{2} + \frac{f(t_{i-1}, y(t_{i-1}))}{-h_0} h_1^2
\]

In the special case of equispaced mesh \( h \), this becomes:

\[
= \frac{f(t_i, y(t_i))}{h} \frac{3h^2}{2} + \frac{f(t_{i-1}, y(t_{i-1}))}{-h} \frac{h^2}{2}
\]

\[
= \frac{3h}{2} f(t_i, y(t_i)) - \frac{h}{2} f(t_{i-1}, y(t_{i-1}))
\]

So two-step Adams-Bashforth (with equispaced mesh) is:

\[ w_{i+1} = w_i + \frac{h}{2} [3f(t_i, w_i) - f(t_{i-1}, w_{i-1})] \]

(b) We do this using Newton Divided differences and then recognize the special case of equispaced mesh as the familiar four-step Adams-Bashforth method. This has the advantage (inherited from Newton interpolating polynomial) that we can increase order or go implicit by adding terms rather than recomputing.

Let’s consider the interpolating polynomial at the points \( \{t_i, t_{i-1}, t_{i-2}, t_{i-3}\} \) in Newton Form.

\[ P(t) = f[t_i] + f[t_i, t_{i-1}](t-t_i) + f[t_i, t_{i-1}, t_{i-2}](t-t_i)(t-t_{i-1}) + f[t_i, t_{i-1}, t_{i-2}, t_{i-3}](t-t_i)(t-t_{i-1})(t-t_{i-2}) \]

Since we’re not assuming an equispaced mesh, let’s define \( h_j := t_{i+j+1} - t_{i+j} \) (so for example \( h_0 = t_{i+1} - t_i \))
\[ w_{i+1} = w_i + \int_{t_i}^{t_{i+1}} f[t_i] + f[t_i, t_{i-1}](t - t_i) + f[t_i, t_{i-1}, t_{i-2}](t - t_i)(t - t_{i-1}) \ldots \]
\[ \cdots + f[t_i, t_{i-1}, t_{i-2}, t_{i-3}](t - t_i)(t - t_{i-1})(t - t_{i-2}) \, dt \]
\[ = w_i + \int_{t_i}^{t_{i+1}} f[t_i] + \left[ \frac{f[t_{i-1}] - f[t_i]}{t_{i-1} - t_i} \right] (t - t_i) + \left[ \frac{f[t_{i-2}] - f[t_{i-1}]}{(t_{i-2} - t_{i-1})(t_{i-2} - t_i)} \right] \cdots \]
\[ \cdots - \frac{f[t_{i-2}]}{(t_{i-2} - t_{i-1})(t_{i-3} - t_{i-2})} (t - t_i)(t - t_{i-1}) + \left[ \frac{f[t_{i-3}] - f[t_{i-2}]}{(t_{i-3} - t_{i-2})(t_{i-3} - t_{i-1})(t_{i-3} - t_i)} \right] \cdots \]
\[ \cdots - \frac{f[t_{i-2}]}{(t_{i-2} - t_{i-1})(t_{i-3} - t_{i-2})}(t_{i-3} - t_{i}) - \frac{f[t_{i-1} - t_i]}{(t_{i-1} - t_i)(t_{i-2} - t_i)(t_{i-3} - t_i)} (t - t_i)(t - t_{i-1})(t - t_{i-2}) \, dt \]
\[ = w_i + h_0 f(t_i, w_i) + \left[ \frac{f(t_i, w_i) - f(t_{i-1}, w_{i-1})}{h_{i-1}} \right] \left[ \frac{h_0(h_0 + h_{i-1})}{2} - \frac{(h_0 + h_{i-1})^3}{6} + \left( \frac{(-h_{i-1})^3}{6} \right) \right] \cdots \]
\[ \cdots + \left[ \frac{f(t_{i-1}, w_{i-1}) - f(t_{i-2}, w_{i-2})}{h_{i-2}(h_{i-2} + h_{i-1})} \right] \left[ \frac{h_0(h_0 + h_{i-1})^2}{2} \right. \left. - \frac{(h_0 + h_{i-1})^3}{6} + \left( \frac{(-h_{i-1})^3}{6} \right) \right] \cdots \]
\[ \cdots + \left[ \frac{f(t_{i-1}, w_{i-1}) - f(t_i, w_i)}{h_{i-1} + h_{i} + h_{i-2} + h_{i-3}} \right] \left[ \frac{h_0 + h_{i-1} + h_{i-2} + h_{i-3}}{3!} \frac{(h_0 + h_{i-1})^3}{2} + \frac{2(h_0)^4}{4!} \right] \cdots \]

In the special case of equispaced mesh \( h \), this becomes:
\[ = w_i + hf(t_i, w_i) + \left[ \frac{f(t_i, w_i) - f(t_{i-1}, w_{i-1})}{h} \right] \frac{h^2}{2} + \left[ \frac{f(t_{i-2}, w_{i-2})}{h} - 2f(t_{i-1}, w_{i-1}) + f(t_i, w_i) \right] \frac{5h^3}{6} \cdots \]
\[ \cdots + \left[ \frac{f(t_{i-3}, w_{i-3}) - 3f(t_{i-2}, w_{i-2}) + 3f(t_{i-1}, w_{i-1}) - f(t_i, w_i)}{h} \right] \frac{27h^4}{12} \cdots \]
\[ = w_i + \frac{h}{24} \left[ 55f(t_i, w_i) - 59f(t_{i-1}, w_{i-1}) + 37f(t_{i-2}, w_{i-2}) - 9f(t_{i-3}, w_{i-3}) \right] \]
Using the Newton backward-difference polynomial interpolant as specified in the book (3.13) with equi-
spaced mesh and the notation $\nabla f(t_i, w_i) = f(t_i, w_i) - f(t_{i-1}, w_{i-1})$ and $\nabla^k f(t_i, w_i) = \nabla(\nabla^{k-1} f(t_i, w_i))$:

\[
 w_{i+1} = w_i + \int_{t_i}^{t_{i+1}} \sum_{k=0}^{3} (-1)^k \binom{-s}{k} \nabla^k f(t_i, w_i) h \, ds
\]

\[
 = w_i + \sum_{k=0}^{3} \nabla^k f(t_i, w_i) h (-1)^k \int_0^1 \binom{-s}{k} \, ds
\]

\[
 = w_i + h \left[ f(t_i, w_i) + \frac{1}{2} \nabla f(t_i, w_i) + \frac{5}{12} \nabla^2 f(t_i, w_i) + \frac{3}{8} \nabla^3 f(t_i, w_i) \right]
\]

\[
 = w_i + h \left[ f(t_i, w_i) + \frac{1}{2} [f(t_i, w_i) - f(t_{i-1}, w_{i-1})] + \frac{5}{12} [f(t_i, w_i) - 2f(t_{i-1}, w_{i-1})]
\]

\[
 + f(t_{i-2}, w_{i-2})] + \frac{3}{8} [f(t_i, w_i) - 3f(t_{i-1}, w_{i-1}) + 3f(t_{i-2}, w_{i-2}) - f(t_{i-3}, w_{i-3})] \right]
\]

\[
 = w_i + \frac{h}{24} \left[ 55f(t_i, w_i) - 59f(t_{i-1}, w_{i-1}) + 37f(t_{i-2}, w_{i-2}) - 9f(t_{i-3}, w_{i-3}) \right]
\]
Question 3

3. (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, \( e(0) = y(0) - u(0) = 0 \).

(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:

\[
u'(t) + e'(t) - y'(t) = y'(t) + \lambda(u(t) + e(t)) - y'(t) = \lambda(u(t) + e(t) - y(t))
\]

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:

\[
z'(t)/z(t) = \lambda \implies \ln(z(t)) = \lambda t + c \implies z(t) = ke^{\lambda t}
\]

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 \( u(t) + e(t) = y(t) \).
Question 4

4. (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.

(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,

$$U_1'(t_n) = -\frac{1}{2h} u_{n+2}^{1} + \frac{2}{h} u_{n+1}^{1} - \frac{3}{2h} u_n$$

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

Similarly, applying the next Euler’s step:

$$e_{n+2}^{1} = e_{n+1}^{1} + hf(t_{n+1}, e_{n+1}^{1} + u_{n+1}^{1}) - u'(t_{n+1})$$

$$e_{n+2}^{1} = e_{n+1}^{1} + hf(t_{n+1}, e_{n+1}^{1} + u_{n+1}^{1})) + \frac{u_n - u_{n+2}^{1}}{2}$$

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
\begin{align*}
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 + hk_2 + hk_1 + hk_3 \\
&= u_n + hk_2 + hk_3
\end{align*}

Furthermore, we see that
\begin{align*}
k_2 &= f(t_{n+1}, u_{n+1}^1) \\
&= f(t_n + h, u_n + hk_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 + hk_2 - 2u_{n+1}^1 + 3u_n)) \\
&= f(t_n + h, u_n + \frac{1}{2}(hk_2 + hk_1))
\end{align*}

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 \(b_1 = 0.\)

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 \\
\frac{1}{2} & 0 & \frac{1}{4} & 0 \\
0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2}
\end{array}
\]

(c) Let us Taylor expand \(k_2(h)\) and \(k_3(h)\) to get (ignoring the \(t\) argument):
\begin{align*}
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)
\end{align*}

Now systematically compute all these derivatives (similar to how we did RK order conditions):
\begin{align*}
k_2(h) &= f(u_n + hk_1) \\
k_2(0) &= f(u_n) = k_1 \\
k_2'(h) &= k_1 f'(u_n + hk_1) \\
k_2'(0) &= k_1 f'(u_n) = ff' \\
k_2''(h) &= k_1^2 f''(u_n + hk_1) \\
k_2''(0) &= k_1^2 f''(u_n) = f^2 f''
\end{align*}
Put it all together:

\[ k_2(h) = f + hf' + \frac{h^2}{2} f'' + O(h^3) \]

Similarly for \( k_3 \):

\[
\begin{align*}
  k_3(h) &= f(u_n + \frac{1}{2}(hk_2 + hk_1)) \\
  k_3(0) &= f(u_n) = k_1 \\
  k'_3(h) &= \frac{1}{2}(k_2 + hk'_2 + k_1) f'(u_n + \frac{1}{2}(hk_2 + hk_1)) \\
  k'_3(0) &= \frac{1}{2}(k_2(0) + k_1) f'(u_n) \\
  &= k_1 f'(u_n) = f' \\
  k''_3(h) &= \frac{1}{2}(k'_2 + hk''_2 + k'_1) f'(u_n + \frac{1}{2}(hk_2 + hk_1)) + \frac{1}{4}(k_2 + hk'_2 + k_1)^2 f''(u_n + \frac{1}{2}(hk_2 + hk_1)) \\
  k''_3(0) &= k'_2(0) f'(u_n) + \frac{1}{4}(k_2(0) + k_1)^2 f''(u_n) \\
  &= k_1(f'(u_n))^2 + \frac{1}{4}(2k_1)^2 f''(u_n) \\
  &= ff'^2 + f^2f''
\end{align*}
\]

Put it all together:

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

Now the truncation error is:

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

\[
= \frac{y_h + \frac{1}{2} (2h)y' + \frac{1}{2} (2h)^2 y'' + \frac{1}{6} (2h)^3 y''' + O(h^4)}{2h} - \frac{1}{2} k_2 - \frac{1}{2} k_3
\]

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

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

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

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

\[
= O(h^2)
\]

d) (See IDEC Handout - Fixed point equivalent)

Since \( u^2 \) is built from \( u^1 \) by:

\[
\begin{align*}
  e_{n+j+1} &= e_{n+j} + h[f(t_{n+j}, u_{n+j}^1 + e_{n+j}) - U'(t_{n+j})] \\
  u_{n+j}^2 &= u_{n+j}^1 + e_{n+j}
\end{align*}
\]

deferred correction is a fixed point iteration of the form

\[
\begin{pmatrix}
  u_{n+1}^2 \\
  u_{n+2}^2 \\
  \vdots \\
  u_{n+p}^2
\end{pmatrix} = \begin{pmatrix}
  u_{n+1}^1 \\
  u_{n+2}^1 \\
  \vdots \\
  u_{n+p}^1
\end{pmatrix} + \begin{pmatrix}
  e_{n+1} \\
  e_{n+2} \\
  \vdots \\
  e_{n+p}
\end{pmatrix} = G \begin{pmatrix}
  u_{n+1}^1 \\
  u_{n+2}^1 \\
  \vdots \\
  u_{n+p}^1
\end{pmatrix}
\]
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
\[
\frac{1}{h} \sum_{k=0}^{p} d_{jk} u_{n+k} = f(t_{n+j}, u_{n+j}) \quad 1 \leq j \leq p
\]  

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:
\[
\sum_{k=0}^{p} d_{jk}(u_{n+k} - u_{n}) = h f(t_{n+j}, u_{n+j}) = h k_j
\]
since the \( k = 0 \) gives us \( d_{j0}(u_{n} - u_{n}) = 0 \), giving use a square system of equations to solve for \((u_{n+k} - u_{n})\):
\[
\sum_{k=1}^{p} d_{jk}(u_{n+k} - u_{n}) = h k_j
\]
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 $ph$. 
Question 5

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.

5a) 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>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>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>$5.600 \times 10^{-4}$</td>
<td>$2.100 \times 10^{-6}$</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>
</tr>
<tr>
<td>80000</td>
<td>1.740</td>
<td>1.026</td>
<td>$4.319 \times 10^{-4}$</td>
<td>$5.872 \times 10^{-4}$</td>
<td>$2.911 \times 10^{-6}$</td>
<td>$4.110 \times 10^{-7}$</td>
<td>$1.981 \times 10^{-9}$</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>$3.180 \times 10^5$</td>
<td>$9.620 \times 10^9$</td>
<td>$5.830 \times 10^{12}$</td>
<td>$1.700 \times 10^{15}$</td>
<td>$1.487 \times 10^{16}$</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.

b) The CPU time for each run:

<table>
<thead>
<tr>
<th>$N$</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>0.0271</td>
<td>0.1013</td>
<td>0.2261</td>
<td>0.4268</td>
<td>0.6902</td>
<td>1.0206</td>
<td>1.4188</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>
</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>
</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>
</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>1-digit</th>
<th>2-digit</th>
<th>3-digit</th>
<th>4-digit</th>
<th>5-digit</th>
<th>6-digit</th>
<th>7-digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-digit</td>
<td>$2.621\times 10^4$</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>
</tr>
<tr>
<td>6-digit</td>
<td>$2.621\times 10^9$</td>
<td>$4.626\times 10^4$</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>
</tr>
<tr>
<td>12-digit</td>
<td>$2.621\times 10^{14}$</td>
<td>$4.626\times 10^7$</td>
<td>6.0234 \times 10^3</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>
</tr>
</tbody>
</table>

c) Some plots of solutions:

The trajectories are provided below (for each $p$ and $N$, $p > 4$ omitted since they’re all really good).
Looking at the data - For three digit accuracy, $p = 5$ is best. For six digit accuracy, $p = 6$ is best. For twelve digit accuracy, $p = 7$ is best.

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

---

13
>> idecToTheMoon(2)

p =
    2

errors =
    1.614391472108695
    1.507214472488082
    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