We modify the MATLAB file to the following:

```matlab
x0 = 0;
y0 = 1;
x_end = 1/2;
N = round((x_end - x0)/h);
f = @(x,y) x-y;
y = @(x) 2*exp(-x) + x - 1;
[xE,yE] = Euler(f,x0,y0,h,N)
```

For h = .25, when we run the code, we find:

\[
xE =
\begin{bmatrix}
0 & 0.250000000000000 & 0.500000000000000 \\
\end{bmatrix}
\]

\[
yE =
\begin{bmatrix}
1.000000000000000 & 0.750000000000000 & 0.625000000000000 \\
\end{bmatrix}
\]

Now, we change the stepsize:

\[h = 0.1;\]

And run the code again:

\[
xE =
\begin{bmatrix}
0 & 0.100000000000000 & 0.200000000000000 & 0.300000000000000 \\
0.400000000000000 & 0.500000000000000 \\
\end{bmatrix}
\]
\[ y_E = \]

\[
1.000000000000000 \ 0.900000000000000 \ 0.820000000000000 \ 0.758000000000000 \\
0.712200000000000 \ 0.680980000000000 \ 0.680980000000000
\]

We can add the following line to get the exact value:

\[ y(1/2) \]

which prints:

\[ \text{ans} = \]

\[
0.713061319425267
\]

From this, we see that clearly the smaller stepsize gets closer to the appropriate endpoint.

[2.5.4] We don't need to change the code much from the previous problem except we now want the improved Euler:

\[
x_0 = 0; \\
y_0 = 1; \\
x_{\text{end}} = 1/2; \\
h = .1; \\
N = \text{round}((x_{\text{end}} - x_0)/h); \\
f = @(x,y) x-y; \\
y = @(x) 2*\exp(-x) + x - 1; \\
[xH, yH] = \text{Heun}(f, x_0, y_0, h, N)
\]

When we run this code, we get:

\[ xH = \]

\[
0 \ 0.100000000000000 \ 0.200000000000000 \ 0.300000000000000 \\
0.400000000000000 \ 0.500000000000000
\]
We can also add the line to evaluate our true solution at our x values:

\( y(x_H) \)

which, when we run the code, we get:

\[
\text{ans} = \\
1.000000000000000 \quad 0.909674836071919 \quad 0.837461506155964 \quad 0.781636441363436 \\
0.740640929271279 \quad 0.713061319425267
\]

We first note that we can solve this equation using separation of variables and get:

\[
y(x) = \frac{x-4}{x-2};
\]

Thus, we input the appropriate information to MATLAB:

\[
x_0 = 0; \\
y_0 = 2; \\
x_\text{end} = 1; \\
h = .01; \\
hprint = 0.2; \\
N = \text{round}((x_\text{end} - x_0)/h); \\
f = @(x,y) 0.5*(y-1).^2; \\
y = @(x) (x-4)./(x-2);
\]

And we only want to print out the values at multiples of 0.2, which we set through "hprint" and adding the lines:
\[ N\text{print} = \frac{h\text{print}}{h}; \]
\[ xH(1: N\text{print}: N+1) \]
\[ yH(1: N\text{print}: N+1) \]

We also want the error, so we add:
\[ \text{errH} = \text{error}(yH, y(xH)); \]
\[ \text{errH}(1: N\text{print}: N+1) \]

When we run this code, we get our x values, our estimates and our error:

\[ \text{ans} = \]

\[
\begin{array}{cccc}
0 & 0.200000000000000 & 0.400000000000000 & 0.600000000000000 \\
0.800000000000000 & 1.000000000000000 & 1.0e-04 & \\
\end{array}
\]

\[ \text{ans} = \]

\[
\begin{array}{cccc}
2.000000000000000 & 2.11109405511450 & 2.249995144764673 & 2.428560562358296 \\
2.666643673934022 & 2.999950378641234 & & \\
\end{array}
\]

\[ \text{ans} = \]

\[
1.0e-04 *
\]

\[
\begin{array}{cccc}
0 & 0.008079156291342 & 0.021578823676075 & 0.044743230548829 \\
0.086222747420206 & 0.165404529224311 & & \\
\end{array}
\]

Now, we just change \( h = .005 \); and rerun the code:

\[ \text{ans} = \]
\[ \begin{array}{cccc}
0 & 0.200000000000000 & 0.400000000000000 & 0.600000000000000 \\
0.800000000000001 & 1.000000000000001 & \\
\end{array} \]

\[ \text{ans =} \]

\[ \begin{array}{cccc}
2.000000000000000 & 2.1111063574705 & 2.249987237158 & 2.428568703654960 \\
2.666660898998120 & 2.99987547105968 & \\
\end{array} \]

\[ \text{ans =} \]

\[ \begin{array}{cccc}
1.0e-05 * \\
0 & 0.020251724486860 & 0.054100570759077 & 0.112202442855768 \\
0.216287908050283 & 0.415096467791069 & \\
\end{array} \]

It's hopefully clear the error is much smaller.

[2.6.4] We make the same adjustments to the code:

\[
\begin{align*}
x_0 &= 0; \\
y_0 &= 1; \\
h &= .1; \\
x_{\text{end}} &= .5; \\
f &= @(x,y) x-y; \\
y &= @(x) 2*exp(-x) + x -1; \\
\end{align*}
\]

\[ [x_{ RK}, y_{ RK}] = \text{RK}(f, x_0, y_0, h, N) \]

The result of this is:

\[ x_{ RK} = \]
\begin{verbatim}
     yRK =
    
          1.000000000000000   0.909675000000000   0.837461802812500   0.781636844002356
     0.740640577834981   0.713061868846760

Again, printing out the actual values:

y(.25)
y(.5)
yields:
ans =

     0.807601566142810

ans =

     0.713061319425267

[2.6.14] The analytical solution to this problem is \( y(x) = \frac{1}{1-\ln(x)} \).

We make the modifications to the code:

x0 = 1;
y0 = 1;
h = .2;
x_end = 2;
N = round((x_end - x0)/h);
\end{verbatim}
\begin{verbatim}
f = @(x,y) (y.^2)./x;
y = @(x) 1./(1-log(x));
[xRK,yRK] = RK(f,x0,y0,h,N)
errH = error(yRK, y(xRK))

This prints out:

xRK =

\begin{verbatim}
  1.00000000000000  1.20000000000000  1.40000000000000  1.60000000000000
  1.80000000000000  2.00000000000000
\end{verbatim}

yRK =

\begin{verbatim}
  1.00000000000000  1.22295663031221  1.50704021407243  1.88666657983663
  2.42558559879118  3.25794597514544
\end{verbatim}

errH =

\begin{verbatim}
  1.0e-03 *

  0   0.014726417798490  0.036977434634832  0.073559863222352
  0.141270756894174  0.290091943244311
\end{verbatim}

We can change the stepsize to be h=0.1; which prints out

xRK =

Columns 1 through 7
\end{verbatim}
\begin{verbatim}
1.000000000000000  1.100000000000000  1.200000000000000  1.300000000000000
1.400000000000000  1.500000000000000  1.600000000000001

Columns 8 through 11

1.700000000000001  1.800000000000001  1.900000000000001  2.000000000000001

yRK =

Columns 1 through 7

1.000000000000000  1.105350676686513  1.222973307490987  1.355680230036072
1.507091839230711  1.681980540462488  1.886795177176907

Columns 8 through 11

2.130491517621662  2.425903117720137  2.792115468058515  3.258821408636764

errH =

1.0e-04 *

Columns 1 through 7

0  0.004950553104641  0.010898482218918  0.018163704610930
0.027227088240970  0.038812119636889  0.054037395903431

Columns 8 through 11

0.074702767105681  0.103852202359298  0.146957568992027  0.214627081998646
\end{verbatim}
[3.1.10]

\[
\begin{align*}
&x + 3y + 2z = 2 \\
&2x + 7y + 7z = -1 \\
&2x + 5y + 2z = 7
\end{align*}
\]

We want to eliminate the \( x \) values:

\[
\begin{align*}
&x + 3y + 2z = 2 \\
&0x + 1y + 3z = -5 \\
&0x - 1y - 2z = 3
\end{align*}
\]

Now we want to eliminate the \( y \) values:

\[
\begin{align*}
&x + 3y + 2z = 2 \\
&0x + 1y + 3z = -5 \\
&0x + 0y + 1z = -2
\end{align*}
\]

From this, it’s immediately clear that \( z = -2 \), and we can substitute this back into the second equation to find that \( y = 1 \) and substitute that back into the first equation to find that \( x = 3 \).

[3.1.24]

It’s easy to verify that \( y(x) = A \cosh 3x + B \sinh 3x \) is a solution to the differential equation

\[
y'' - 9 = 0.
\]

To do so, just plug in for \( y \). I will omit the details of this here. The new idea in this problem is to find \( A, B \) such that the boundary conditions:

\[
y(0) = 5, \quad y'(0) = 12
\]

are indeed satisfied.

The first boundary condition says that, when we plug in \( x = 0 \), we get:

\[
y(0) = A \cosh 0 + B \sinh 0 = A = 5,
\]

since we know \( \cosh 0 = 1 \) and \( \sinh 0 = 0 \). The second condition, we have to take the derivative of our solution:

\[
y'(x) = 3A \sinh 3x + 3B \cosh 3x,
\]

and we can readily verify the second boundary condition:

\[
y'(0) = 3A \sinh 0 + 3B \cosh 0 = 3B = 12 \quad \implies \quad B = 4.
\]