CP 12.1.4 To set up the problem, we work with the related difference equation:
\[
\frac{x_j^2}{h^2} \frac{y_{j-1} - 2y_j + y_{j+1}}{h^2} - \frac{x_j}{h} \frac{y_{j+1} - y_{j-1}}{2h} + y_j = \ln x_j,
\]
where we consider \(x_j = 1 + jh\) for \(j = 0, 1, \ldots, n\), and \(y_j\) is meant to be an approximation to the true solution \(y(x_j)\) at the \(j\)th mesh point. We manipulate (using algebra) the above equation into the form
\[
x_j \left(x_j + \frac{h}{2}\right) y_{j-1} + (h^2 - 2x_j^2) y_j + x_j \left(x_j - \frac{h}{2}\right) y_{j+1} = h^2 \ln x_j,
\]
which holds for \(j = 1, \ldots, n - 1\) (i.e., holds at interior mesh points). Adding the two boundary conditions gives the matrix problem
\[
\begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & 0 \\
x_1^2 + \frac{h}{2} x_1 & h^2 - 2x_1^2 & x_1^2 - \frac{h}{2} x_1 & 0 & \cdots & 0 \\
0 & x_2^2 + \frac{h}{2} x_2 & h^2 - 2x_2^2 & x_2^2 - \frac{h}{2} x_2 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_n
\end{bmatrix}
= \begin{bmatrix}
0 \\
h^2 \ln x_1 \\
h^2 \ln x_2 \\
\vdots \\
-2
\end{bmatrix}.
\]

The following Octave commands construct these matrices and produce the resulting vector of unknown (approximate solution) values:

```octave
N = 8;
h = 1/N;
xjs = [1:h:2]';
intXs = xjs(2:end-1); % the interior mesh points
subdiag = [intXs .* (intXs + h/2); 0];
superdiag = [0; intXs .* (intXs - h/2)];
maindiag = [1; h^2-2*intXs.^2; 1];
A = diag(subdiag, -1) + diag(maindiag) + diag(superdiag, 1);

rhs = [0; log(intXs)*h^2; -2];
A \ rhs
```

**44** (a) I noticed that the `dirichletShooter()` routine with which I provided you was not working correctly when the DE was indicated to be nonlinear. (The
error arose from treating the returned \( t \)-values from \texttt{rk4sys()} routine as if they were the \( y \)-values, and vice versa.) That problem is now fixed (updated file is available at the above link.)

Using the updated routine, here is how I would go about solving this nonlinear BVP.

\[
q = [1 \ 2]; \quad \text{% two starting values for the routine to try out for } y'(0)
\]

```matlab
function yp = yprime(x, y)
    yp = zeros(size(y));
    yp(1) = y(2);
    yp(2) = y(2) - sin(x * y(1));
end
```

\[
[ys, xs] = \text{dirichletShooter(@(yprime, [0 \ 1], [1 \ 1.5], q, 10, 10^{-4}, 25, 0)}
\]

**Challenge:** The main issue is in dealing with the inhomogeneity \( r(x) \). In order to use a routine like mine to solve a problem of the form

\[
y'' + p(x)y' + q(x)y = r(x)
\]

we must convert to a system and have a function like this one

```matlab
function yp = yprime(x, y)
    yp = zeros(size(y));
    yp(1) = y(2);
    yp(2) = r(x) - p(x)*y(2) - q(x)*y(1);
end
```

which encodes the dynamics of the problem. If you solve the \( u \) and \( v \) problems I presented in class, this same \texttt{yprime()} routine gives the dynamics for both of these problems. However, if you solve the \( u, v \) problems presented in the text, the above \texttt{yprime()} function gives the dynamics only for the \( u \) problem. One must, effectively, supply another routine, one giving the dynamics of the \( v \)-problem, as an argument to your \textit{shooter} program. That is, in fact, what is required when using the \texttt{lshoot()} routine supplied with the text (and printed on pp. 476–477).