

ME 406

Assignment #1 Solutions

PROBLEM 1

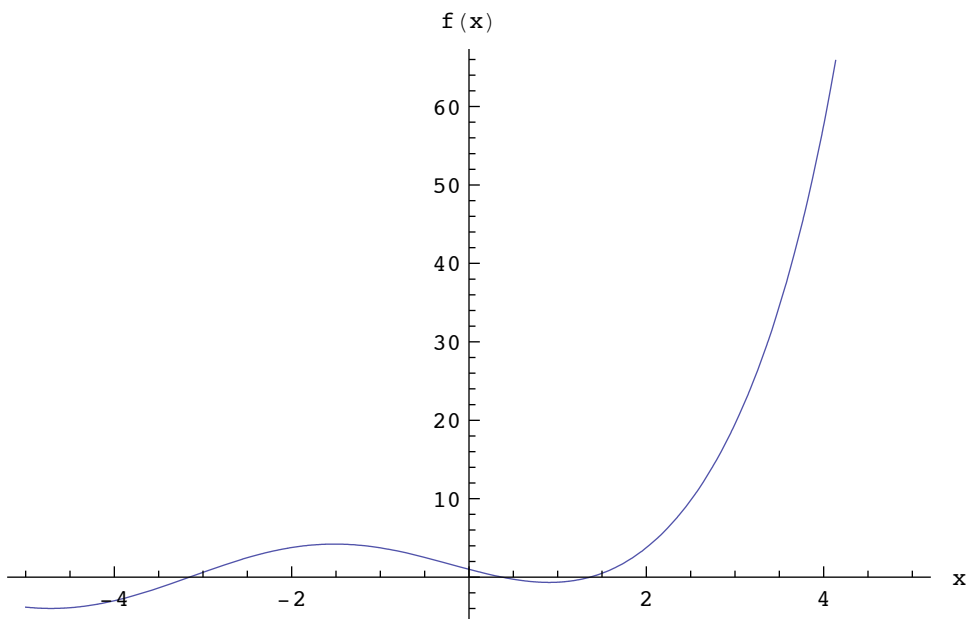
We define the function for *Mathematica*.

```
In[1]:= f[x_] := Exp[x] - 4 Sin[x]
```

(a) We use Plot to construct the plot.

```
In[2]:= Plot[f[x], {x, -5, 5}, AxesLabel -> {"x", "f(x)"}]
```

Out[2]=



(b) The graph shows that there are two positive roots, one near 0.4 and the other near 1.5. We use FindRoot to find them. We check each root after we find it.

```
In[3]:= FindRoot[f[x] == 0, {x, 0.4}]
```

```
Out[3]= {x -> 0.370558}
```

```
In[4]:= f[x] /. %
```

```
Out[4]= 0.
```

```
In[5]:= FindRoot[f[x] == 0, {x, 1.5}]
```

```
Out[5]= {x -> 1.36496}
```

```
In[6]:= f[x] /. %
```

```
Out[6]= 0.
```

The checks show that the roots are accurate.

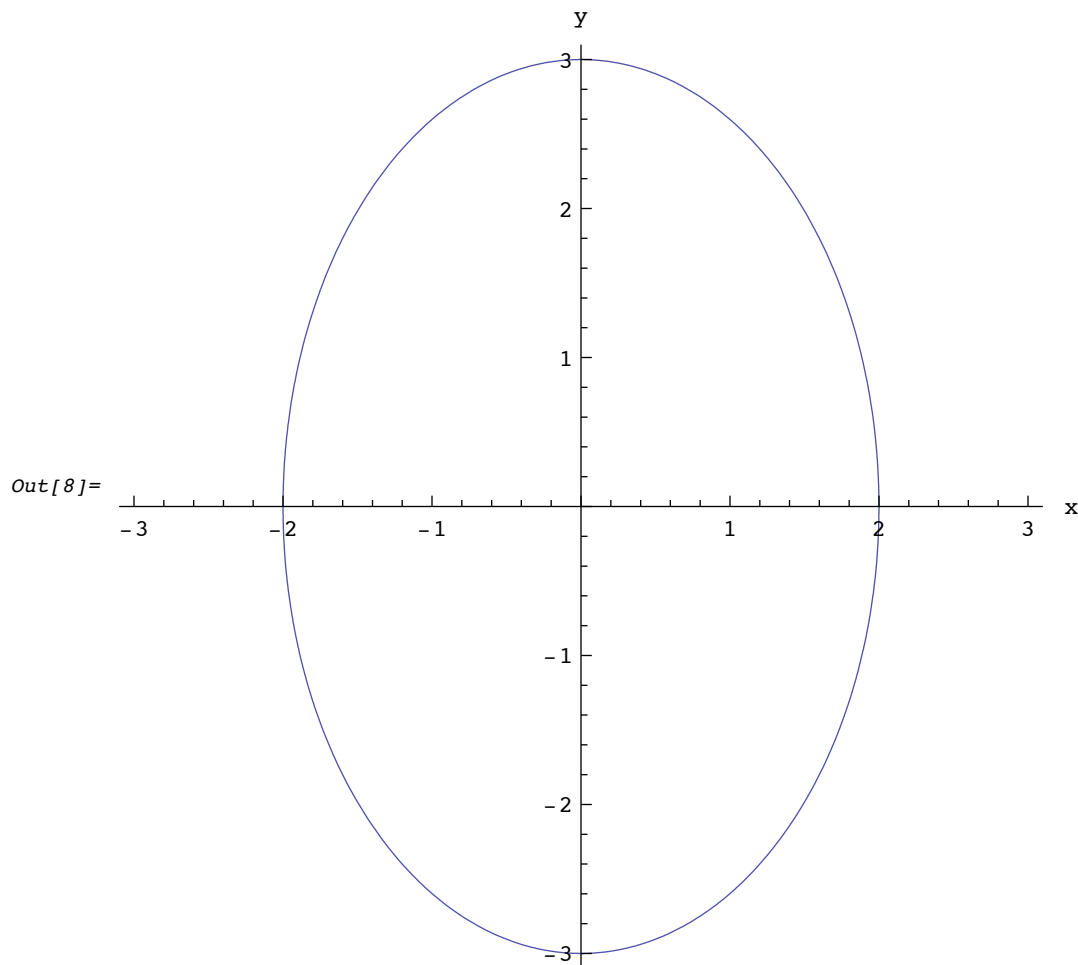
PROBLEM 2

A standard parametric representation is given by

```
In[7]:= x[t_] := 2 Cos[t]; y[t_] := 3 Sin[t]
```

We use ParametricPlot to plot this.

```
In[8]:= ParametricPlot[{x[t], y[t]}, {t, 0, 2 π}, AspectRatio → 1,  
PlotRange → {{-3.1, 3.1}, {-3.1, 3.1}}, AxesLabel → {"x", "y"}]
```



PROBLEM 3

We define the equation for *Mathematica*. We call it `equat3`.

```
In[9]:= Clear[x, m, c, k, a, b];
```

```
In[10]:= equat3 := {m x''[t] + c x'[t] + k x[t] == 0, x[0] == a, x'[0] == b};
```

We will use equat3 as an argument of DSolve and NDSolve. Now we specify the parameter values.

```
In[11]:= m = 2; c = 0.8; k = 8; a = 1; b = 2;
```

(a) We solve with DSolve.

```
In[12]:= ans3 = DSolve[equat3, x[t], t]
```

```
Out[12]= {{x[t] -> e^{-0.2 t} (1. Cos[1.98997 t] + 1.10554 Sin[1.98997 t])}}
```

We convert the replacement rule to a function named osc[t].

```
In[13]:= osc[t_] = x[t] /. Flatten[ans3]
```

```
Out[13]= e^{-0.2 t} (1. Cos[1.98997 t] + 1.10554 Sin[1.98997 t])
```

(b) We check first the initial conditions and then the equation.

```
In[14]:= osc[0]
```

```
Out[14]= 1.
```

```
In[15]:= osc'[0]
```

```
Out[15]= 2.
```

```
In[16]:= m osc''[t] + c osc'[t] + k osc[t]
```

```
Out[16]= 8 e^{-0.2 t} (1. Cos[1.98997 t] + 1.10554 Sin[1.98997 t]) +
0.8 (e^{-0.2 t} (2.2 Cos[1.98997 t] - 1.98997 Sin[1.98997 t]) -
0.2 e^{-0.2 t} (1. Cos[1.98997 t] + 1.10554 Sin[1.98997 t])) +
2 (e^{-0.2 t} (-3.96 Cos[1.98997 t] - 4.37794 Sin[1.98997 t]) -
0.4 e^{-0.2 t} (2.2 Cos[1.98997 t] - 1.98997 Sin[1.98997 t]) +
0.04 e^{-0.2 t} (1. Cos[1.98997 t] + 1.10554 Sin[1.98997 t]))
```

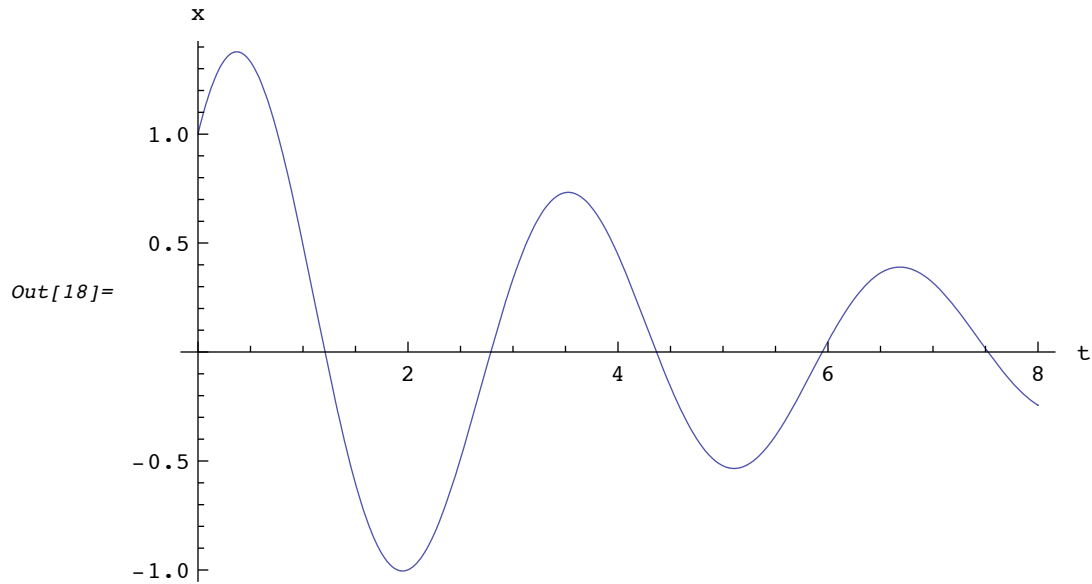
```
In[17]:= Simplify[%]
```

```
Out[17]= 0.
```

Initial conditions and equation both check.

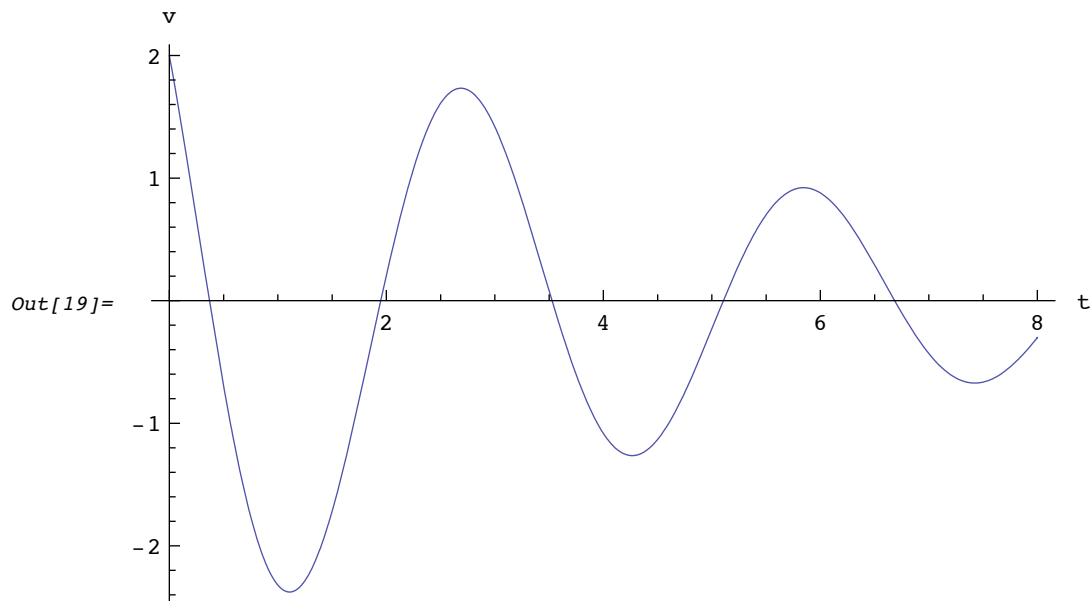
(c) We apply the plot command to osc[t]. We name the graph so that we can refer to it later when we combine it with another graph.

```
In[18]:= graphx = Plot[osc[t], {t, 0, 8}, AxesLabel -> {"t", "x"}]
```



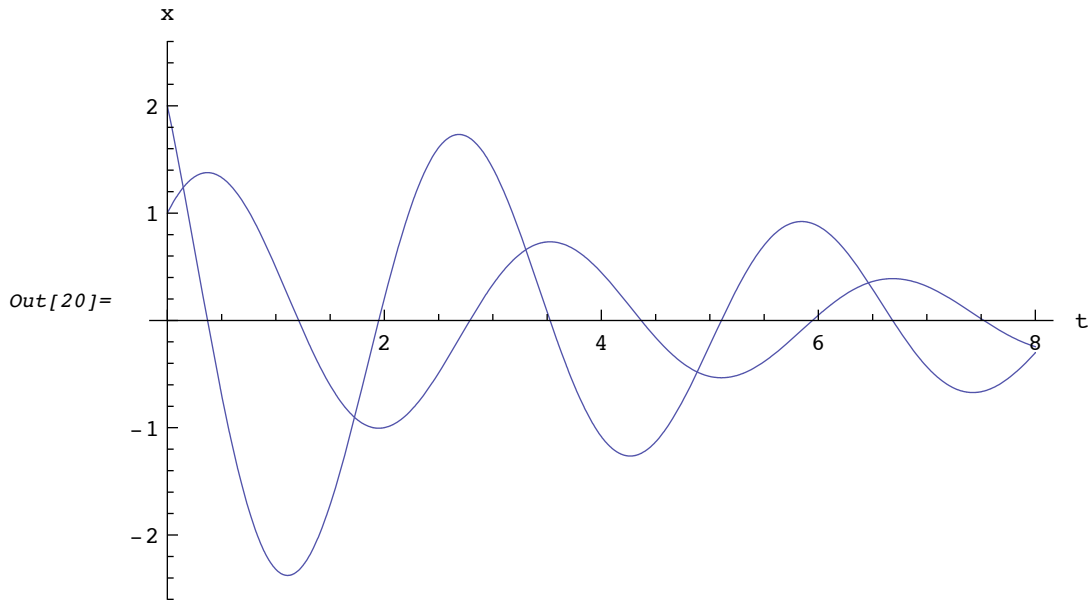
(d) We now apply Plot to $\text{osc}'[t]$, again naming the graph.

```
In[19]:= graphv = Plot[osc'[t], {t, 0, 8}, AxesLabel -> {"t", "v"}]
```



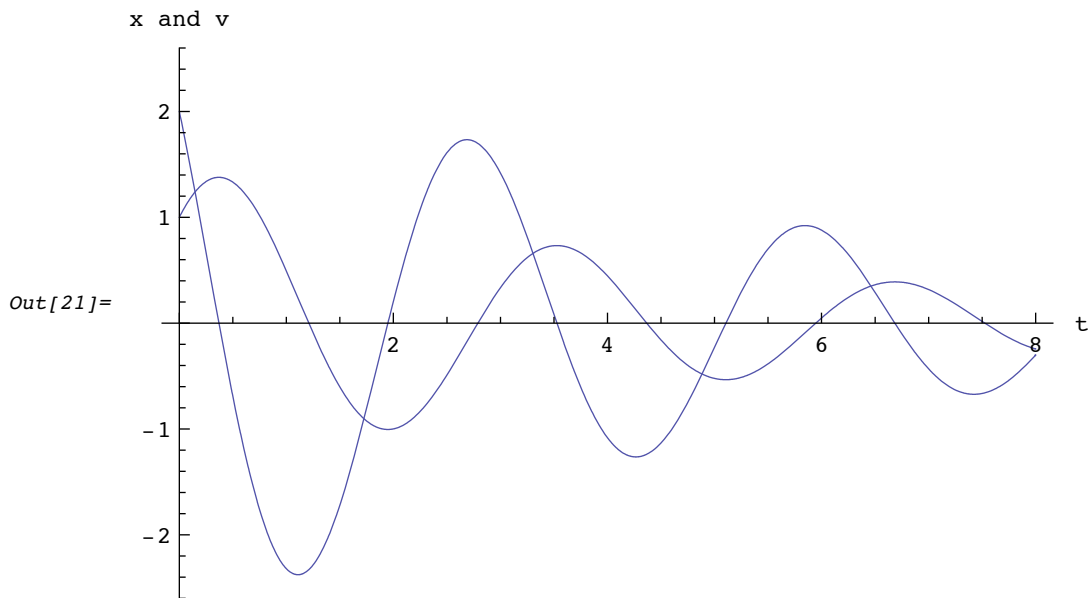
(e) We combine the graphs with a Show command.

```
In[20]:= Show[graphx, graphv, PlotRange -> {-2.5, 2.5}]
```



We can insert a more appropriate label on the vertical axis.

```
In[21]:= Show[graphx, graphv, PlotRange -> {-2.5, 2.5}, AxesLabel -> {"t", "x and v"}]
```



(f) Now we will use NDSolve to get the solution. We name the solution (i.e., the interpolating function output) oscnum.

```
In[22]:= oscnum = NDSolve[equat3, x[t], {t, 0, 8}]
```

```
Out[22]= {{x[t] -> InterpolatingFunction[{{0., 8.}}, <>][t]}}
```

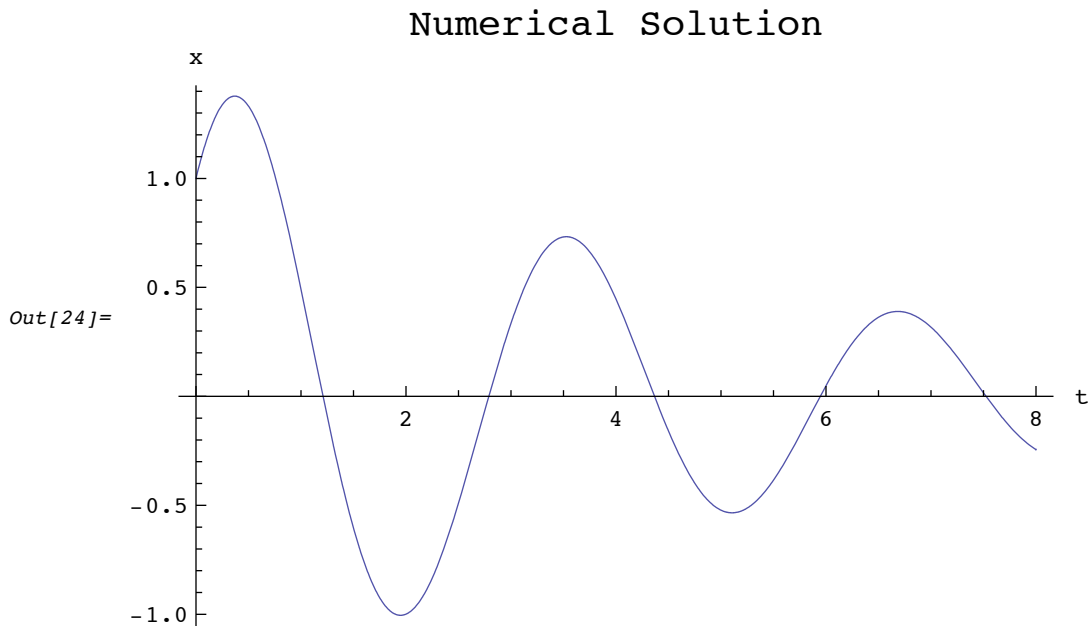
We convert this interpolating function output to an ordinary function, oscout[t].

```
In[23]:= osout[t_] = x[t] /. Flatten[oscnm]
```

```
Out[23]= InterpolatingFunction[{{0., 8.}}, <>][t]
```

Now we plot this function.

```
In[24]:= graphnumx =
  Plot[osout[t], {t, 0, 8}, AxesLabel -> {"t", "x"}, PlotLabel -> "Numerical Solution"]
```



This is the same as our earlier graph, except for the newly added plot label.

(g) We convert this equation to a system of two first order equations, by introducing $v = dx/dt$. The result is

$$\begin{aligned} \frac{dx}{dt} &= v, \\ \frac{dv}{dt} &= -\frac{c}{m} v - \frac{k}{m} x, \\ \text{with } x(0) &= a, \quad v(0) = b. \end{aligned}$$

We define this equation for *Mathematica*.

```
In[25]:= sysequat3 = {x'[t] == v[t], v'[t] == -(c/m) v[t] - (k/m) x[t], x[0] == a, v[0] == b};
```

Now we use this as one of the arguments of `NDSolve`.

```
In[26]:= sysnum = NDSolve[sysequat3, {x[t], v[t]}, {t, 0, 8}]
```

```
Out[26]= {{x[t] -> InterpolatingFunction[{{0., 8.}}, <>][t],
  v[t] -> InterpolatingFunction[{{0., 8.}}, <>][t]}}
```

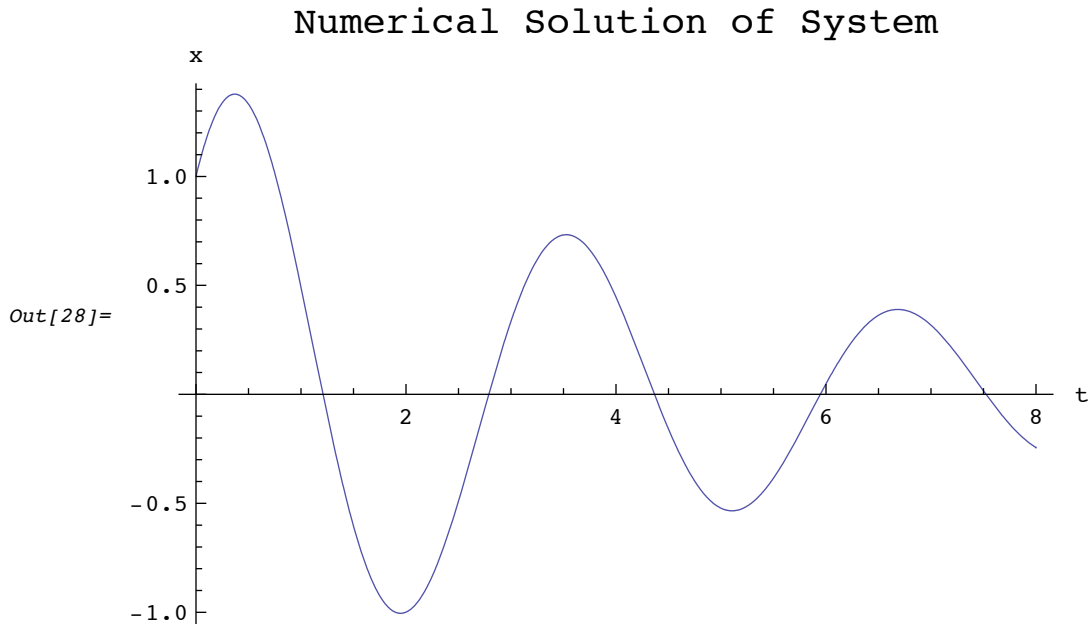
We convert the interpolating function output to an ordinary function. We call the two components of the ordinary function `xosc[t]` and `vosc[t]`.

```
In[27]:= {xosc[t_], vosc[t_]} = {x[t], v[t]} /. Flatten[sysnum]
```

```
Out[27]= {InterpolatingFunction[{{0., 8.}}, <>][t], InterpolatingFunction[{{0., 8.}}, <>][t]}
```

We plot xosc[t].

```
In[28]:= graphsysoscx = Plot[xosc[t], {t, 0, 8},
  AxesLabel -> {"t", "x"}, PlotLabel -> "Numerical Solution of System"]
```



PROBLEM 4

We define the matrix for *Mathematica*, and display it using MatrixForm.

```
In[29]:= A = {{1, -2, 3, 0}, {-2, 0, 3, 2}, {3, 3, 5, -6}, {0, 2, -6, 1}}
```

```
Out[29]= {{1, -2, 3, 0}, {-2, 0, 3, 2}, {3, 3, 5, -6}, {0, 2, -6, 1}}
```

```
In[30]:= MatrixForm[A]
```

```
Out[30]//MatrixForm=
```

$$\begin{pmatrix} 1 & -2 & 3 & 0 \\ -2 & 0 & 3 & 2 \\ 3 & 3 & 5 & -6 \\ 0 & 2 & -6 & 1 \end{pmatrix}$$

(a) We calculate the inverse, calling it Ainvs, and then check it by multiplying A and Ainvs.

```
In[31]:= Ainvs = Inverse[A]
```

```
Out[31]= {{101, 17, 18, 74}, {17, 40, 21, 46},
  {18, 21, 8, 6}, {74, 46, 6, 65}}
  {121, 121, 121, 121}, {121, 121, 121, 121},
  {121, 121, 121, 121}, {121, 121, 121, 121}}
```

```
In[32]:= MatrixForm[A.Ainv]
```

```
Out[32]//MatrixForm=
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To solve the linear equations $AX = b$, we first define b for *Mathematica*, and then calculate the solution as $Ainv.b$.

```
In[33]:= b = {1, 0, 1, 0}
```

```
Out[33]= {1, 0, 1, 0}
```

```
In[34]:= X = Ainv.b
```

```
Out[34]= { 119/121, 38/121, 26/121, 80/121 }
```

We can convert the exact rational numbers to numerical values with the `N` command.

```
In[35]:= N[X]
```

```
Out[35]= {0.983471, 0.31405, 0.214876, 0.661157}
```

Now we check the solution, still using the exact rational values.

```
In[36]:= A.X - b
```

```
Out[36]= {0, 0, 0, 0}
```

Because the entries of A were all integers, *Mathematica* did exact calculations. By using the `N` command to convert integer to real results, we can do the calculations in terms of reals rather than integers.

```
In[37]:= Ainv = Inverse[N[A]]
```

```
Out[37]= {{0.834711, 0.140496, 0.14876, 0.611157}, {0.140496, 0.330579, 0.173554, 0.380165},
           {0.14876, 0.173554, 0.0661157, 0.0495868}, {0.611157, 0.380165, 0.0495868, 0.53719}}
```

```
In[38]:= X = Ainv.b
```

```
Out[38]= {0.983471, 0.31405, 0.214876, 0.661157}
```

(b) Because all of the entries of A are exact integers, *Mathematica* will attempt an exact calculation of the eigenvalues and eigenvectors. Let's look at just the eigenvalues first.

```
In[39]:= Eigenvalues[A]
```

```
Out[39]= {Root[-121 + 214 #1 - 51 #1^2 - 7 #1^3 + #1^4 &, 4], Root[-121 + 214 #1 - 51 #1^2 - 7 #1^3 + #1^4 &, 1],
           Root[-121 + 214 #1 - 51 #1^2 - 7 #1^3 + #1^4 &, 3], Root[-121 + 214 #1 - 51 #1^2 - 7 #1^3 + #1^4 &, 2]}
```

A small lesson here: "exact" isn't always useful. In principle exact solutions of a quartic equation are possible. In practice it is not worth our time to unravel the above answer. We force a numerical approach by using the `N` function which converts exact values to numerical values.

```
In[40]:= Eigenvalues[N[A]]
```

```
Out[40]= {10.0724, -6.46288, 2.70274, 0.687735}
```

Now the eigenvectors.

```
In[41]:= eigvecs = Eigenvectors[N[A]]
```

```
Out[41]= {{0.24912, 0.0899059, 0.813309, -0.518059},  
          {-0.353199, -0.528408, 0.526355, 0.564788},  
          {-0.55554, 0.782235, 0.206176, 0.192287}, {0.710327, 0.317505, 0.137733, 0.612907}}
```

We check the pairwise orthogonality. First we extract the four vectors from the above list.

```
In[42]:= eig1 = eigvecs[[1]]
```

```
Out[42]= {0.24912, 0.0899059, 0.813309, -0.518059}
```

```
In[43]:= eig2 = eigvecs[[2]]
```

```
Out[43]= {-0.353199, -0.528408, 0.526355, 0.564788}
```

```
In[44]:= eig3 = eigvecs[[3]]
```

```
Out[44]= {-0.55554, 0.782235, 0.206176, 0.192287}
```

```
In[45]:= eig4 = eigvecs[[4]]
```

```
Out[45]= {0.710327, 0.317505, 0.137733, 0.612907}
```

Now we check the orthogonality.

```
In[46]:= eig1.eig2
```

```
Out[46]=  $-2.22045 \times 10^{-16}$ 
```

```
In[47]:= eig1.eig3
```

```
Out[47]=  $5.55112 \times 10^{-17}$ 
```

```
In[48]:= eig1.eig4
```

```
Out[48]=  $-1.66533 \times 10^{-16}$ 
```

```
In[49]:= eig2.eig3
```

```
Out[49]=  $-1.11022 \times 10^{-16}$ 
```

```
In[50]:= eig2.eig4
```

```
Out[50]=  $3.33067 \times 10^{-16}$ 
```

```
In[51]:= eig3.eig4
```

```
Out[51]=  $2.498 \times 10^{-16}$ 
```

Thus they are orthogonal to within the numerical accuracy of the calculation.