Chapter 6: Linear First Order Differential Equations

As was shown in Chapter 3, the form is $y' + p(x)y = q(x)$. Here the p and q can be as weird as desired, but the exponent on y and y' must be linear only. The category also includes Bernoulli equations, which by means of a substitution, can be reduced to first order if the q factor is above linear.

6.2 Solve the differential equation $y' - 3y = 6$.

Wolfram Alpha solves the problem, adusting internally for the necessary means to accomplish a solution. In the input field, it is necessary only to enter the equation, without category or directions. (Probably the y' clues W|A in on the desired treatment of the equation.) Python is used here to pretty print the answer, but takes no part in the calculations.

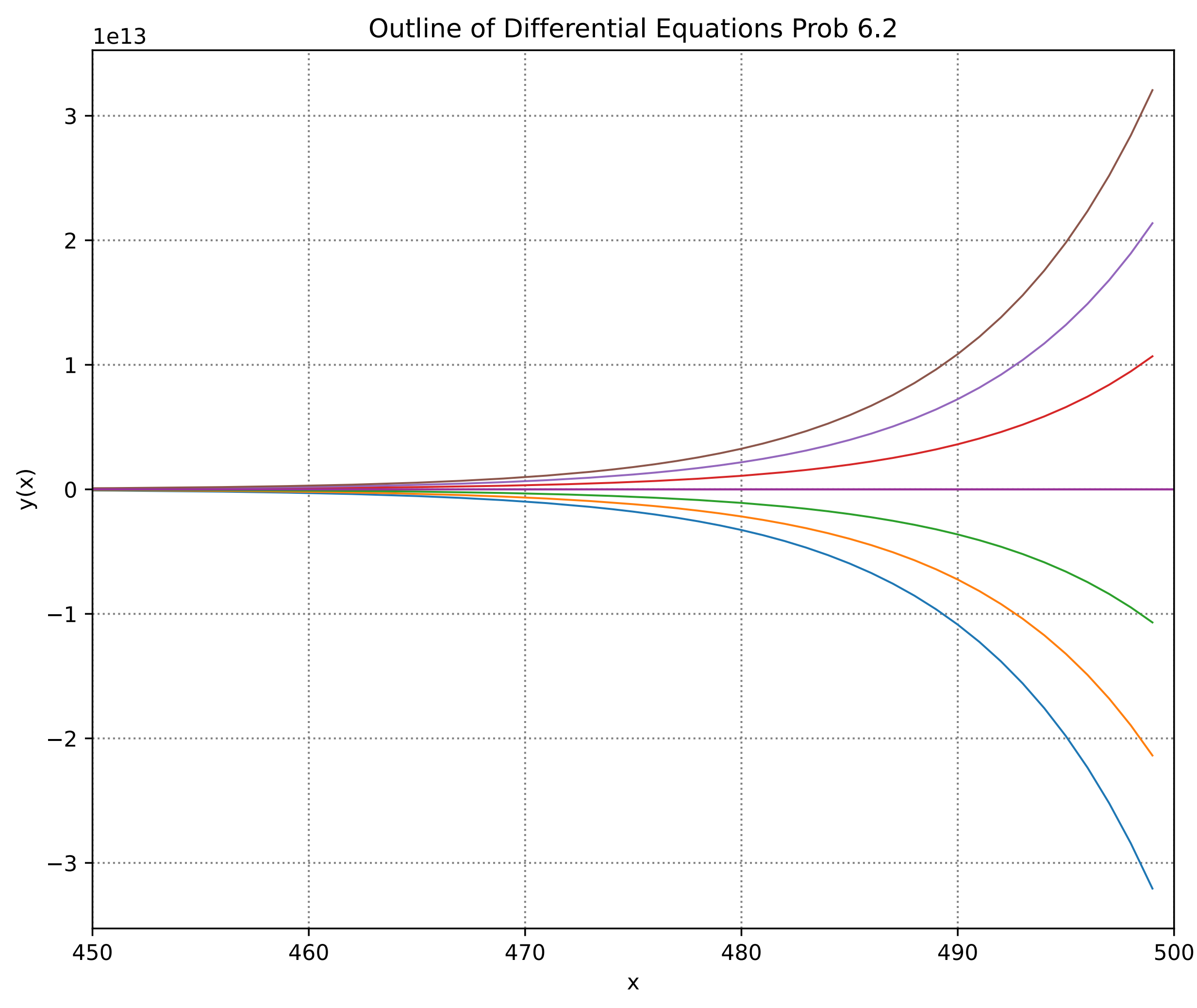
```
In [2]: from sympy import *

        (x, c1) = symbols('x c1')
        y = Function('y')(x)
        y = c1*exp(3*x) - 2
        y
```

Out[2]: $c_1 e^{3x} - 2$

Here matplotlib handles plotting the solution, in a group portrait which includes a few possible constant values.

```
In [18]: import numpy as np
         import matplotlib.pyplot as plt

         %config InlineBackend.figure_formats = ['svg']

         x=np.linspace(-10,10,500)

         two = np.array([-3,-2,-1, 1, 2, 3])

         def f(x):
             for k in two:
                 y = k*np.exp(3*x) - 2
                 plt.plot(y, linewidth = 0.9)
         y=f(x)

         plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
         plt.xlabel("x")
         plt.ylabel("y(x)")
         plt.title("Outline of Differential Equations Prob 6.2")
         plt.rcParams['figure.figsize'] = [9, 7.5]

         ax = plt.gca()
         ax.axhline(y=0, color='#993399', linewidth=1)
         ax.axvline(x=0, color='#993399', linewidth=1)

         plt.xlim(450, 500)
```

```
Out[18]: (450.0, 500.0)
```



6.4 Solve the differential equation $y' - 2xy = x$.

Wolfram Alpha describes the problem as "first-order linear ordinary differential equation." There is no time delay on the solution.

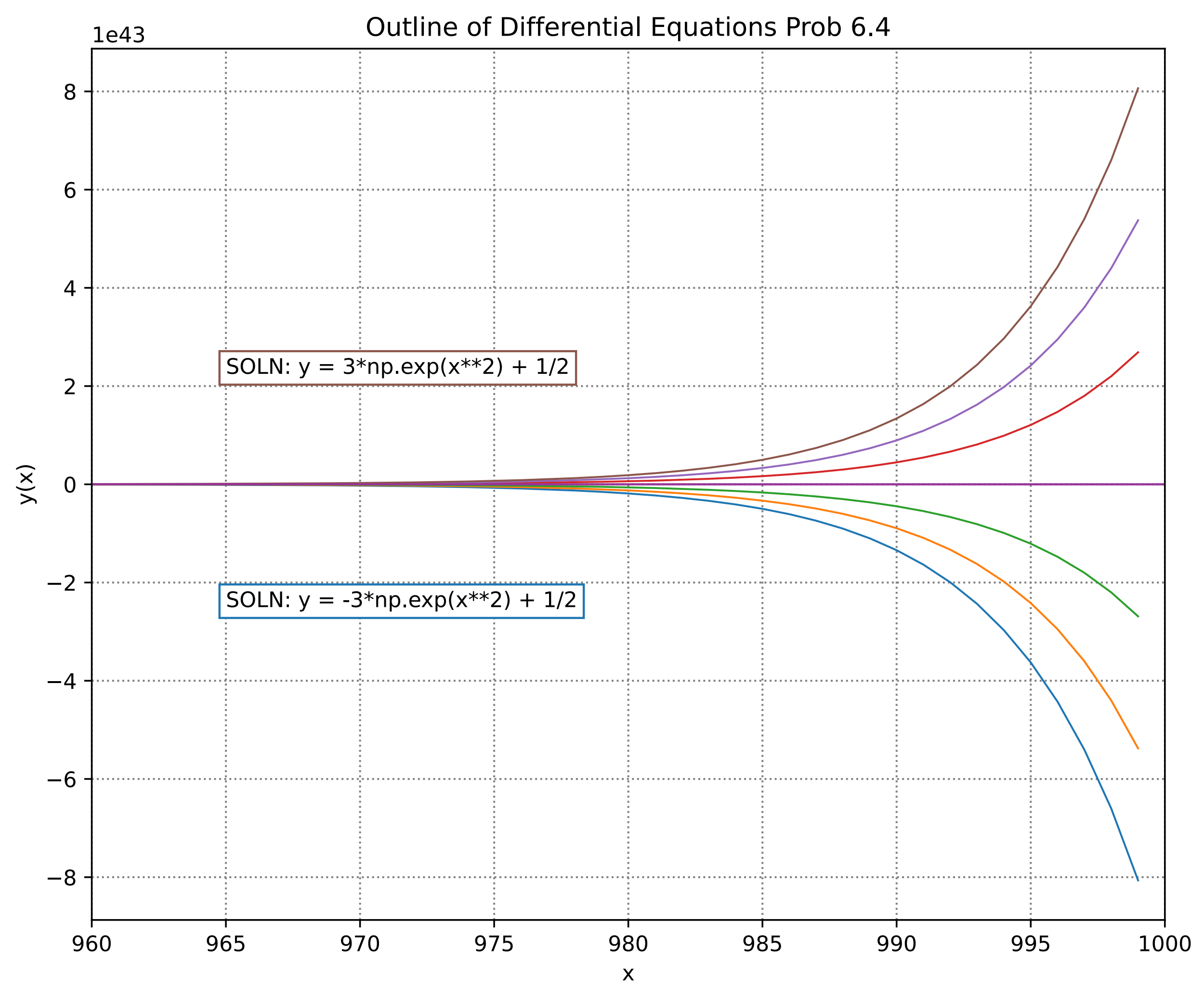
```
In [21]: from sympy import *
         from fractions import Fraction as frac

         (x, c1) = symbols('x c1')
         y = Function('y')(x)
         y = c1*exp(x**2) - frac(1,2)
         y
```

Out[21]: $c_1 e^{x^2} - \frac{1}{2}$

And matplotlib handles the plot, which is similar to the last. In the ethereal region which the plot occupies, some care is required in the details of the plot.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x=np.linspace(0,10,1000)

two = np.array([-3,-2,-1, 1, 2, 3])

def f(x):
    for k in two:
        y = k*np.exp(x**2) + 1/2
        plt.plot(y, linewidth = 0.9)
y=f(x)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 6.4")
plt.rcParams['figure.figsize'] = [9, 7.5]

ax = plt.gca()
ax.axhline(y=0, color='#993399', linewidth=1)
ax.axvline(x=0, color='#993399', linewidth=1)

plt.text(965, -2.5e43, "SOLN: y = -3*np.exp(x**2) + 1/2", size=10,\
         bbox=dict(boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(965, 2.25e43, "SOLN: y = 3*np.exp(x**2) + 1/2", size=10,\
         bbox=dict(boxstyle="square", ec=('#8C564B'),fc=(1., 1., 1),))
plt.xlim(960, 1000)
```

Out[2]:
```
(960.0, 1000.0)
```



6.6 Solve the differential equation $y' + (\frac{4}{x})y = x^4$.

Wolfram Alpha describes the problem as "first-order linear ordinary differential equation." There is no time delay on the solution.

In [56]:
```
from sympy import *
from fractions import Fraction

(x, c1) = symbols('x c1')
y = Function('y')(x)
y = c1/x**4 + x**5/9
y
```

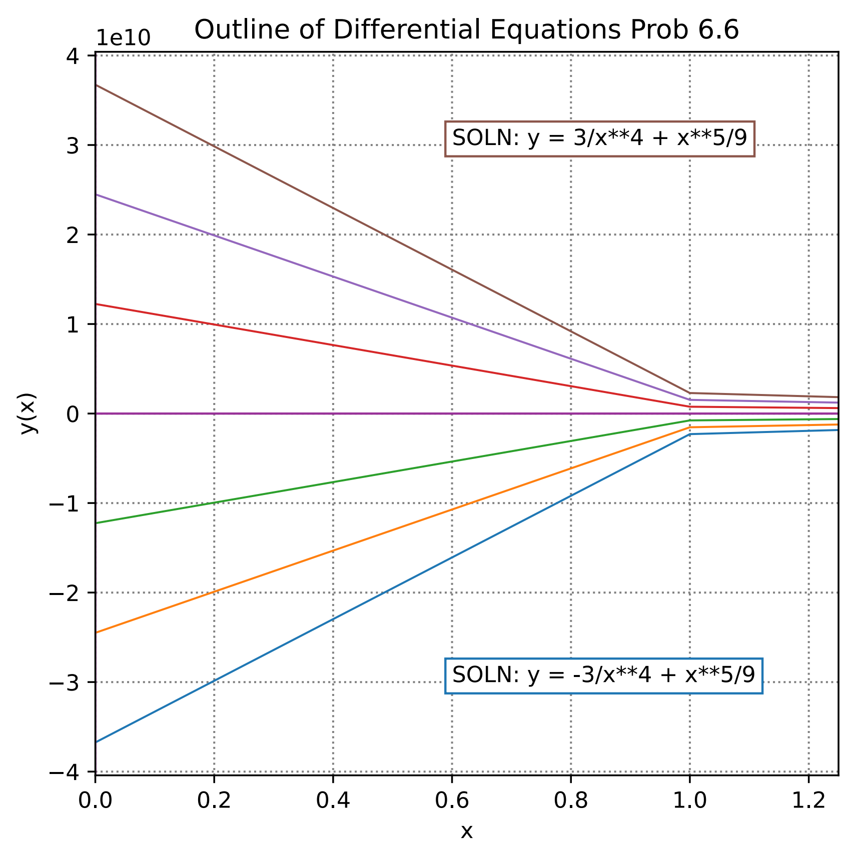
Out[56]:

$$\frac{c_1}{x^4} + \frac{x^5}{9}$$

The matplotlib plot is somewhat different from the last.

In [81]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x=np.linspace(0,10,500)
x = np.setdiff1d(np.linspace(0.,1.5,500),[0])

two = np.array([-3,-2,-1, 1, 2, 3])

def f(x):
    for k in two:
        y = k/x**4 + x**5/9
        plt.plot(y, linewidth = 0.9)
y=f(x)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 6.6")
plt.rcParams['figure.figsize'] = [6, 6]

ax = plt.gca()
ax.axhline(y=0, color='#993399', linewidth=1)
ax.axvline(x=0, color='#993399', linewidth=1)

plt.text(0.6, -3e10, "SOLN: y = -3/x**4 + x**5/9", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(0.6, 3e10, "SOLN: y = 3/x**4 + x**5/9", size=10,bbox=dict\
         (boxstyle="square", ec=('#8C564B'),fc=(1., 1., 1),))
plt.xlim(0, 1.25)
```

Out[81]:
```
(0.0, 1.25)
```



6.7 Solve the differential equation $y' + y = \sin x$.

Wolfram Alpha describes the problem as "first-order linear ordinary differential equation." There is no time delay on the solution.

In [92]:
```
from sympy import *

(x, c1) = symbols('x c1')
y = Function('y')(x)
y = c1*exp(-x) + sin(x)/2 - cos(x)/2
y
```

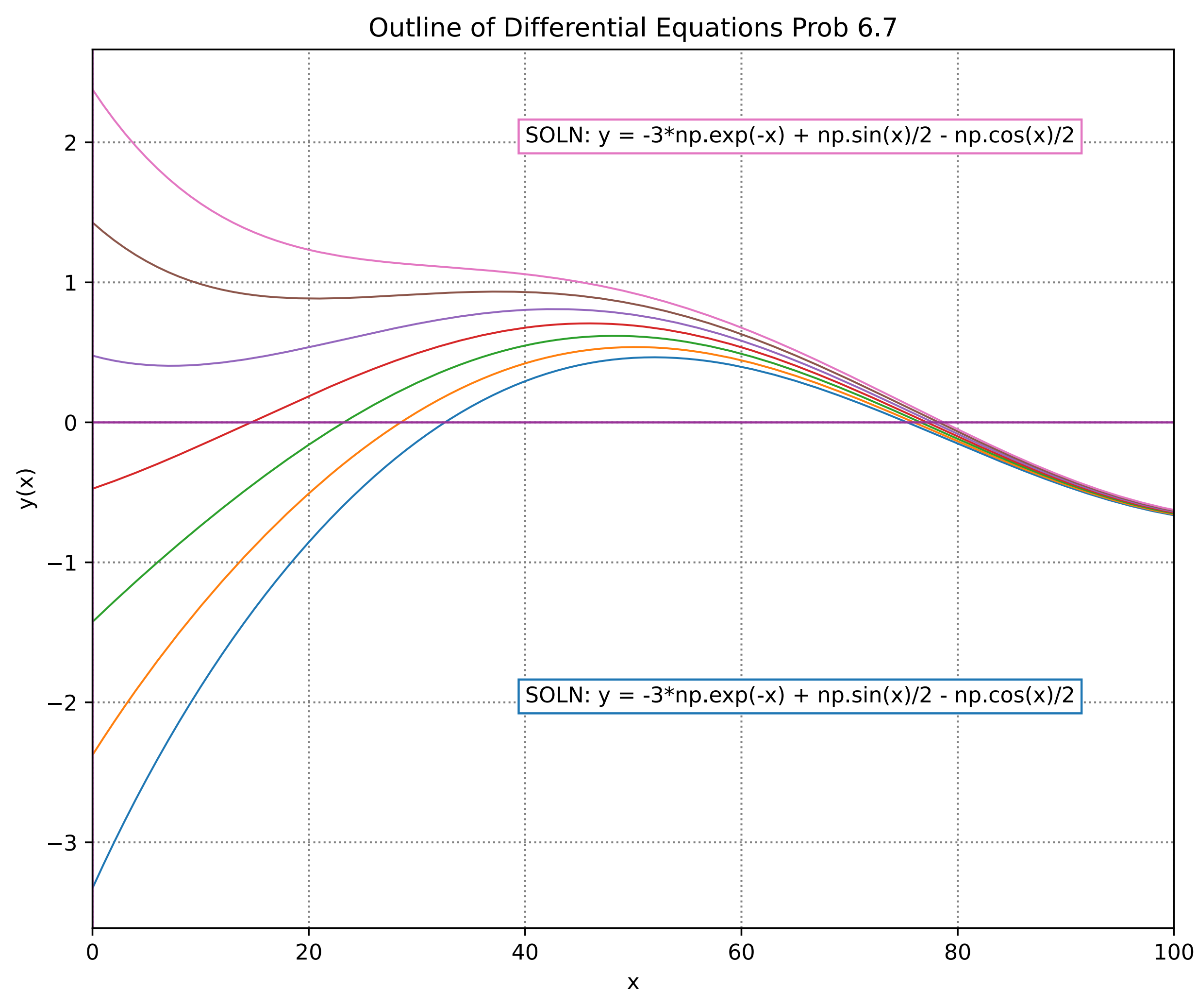
Out[92]:

$$c_1 e^{-x} + \frac{\sin(x)}{2} - \frac{\cos(x)}{2}$$

The matplotlib plot shows a bit of a swirl. With this array of constants, a zero was included experimentally.

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.setdiff1d(np.linspace(0.,10,200),[0]) #To remove zero

two = np.array([-3,-2,-1, 0, 1, 2, 3])

def f(x):
    for k in two:
        y = k*np.exp(-x) + np.sin(x)/2 - np.cos(x)/2
        plt.plot(y, linewidth = 0.9)
y=f(x)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 6.7")
plt.rcParams['figure.figsize'] = [9, 7.5]

ax = plt.gca()
ax.axhline(y=0, color='#993399', linewidth=1)
ax.axvline(x=0, color='#993399', linewidth=1)

plt.text(40, -2, "SOLN: y = -3*np.exp(-x) + np.sin(x)/2 - np.cos(x)/2",\
         size=10,bbox=dict(boxstyle="square", ec=('#1F77B4'),\
         fc=(1., 1., 1),))
plt.text(40, 2, "SOLN: y = -3*np.exp(-x) + np.sin(x)/2 - np.cos(x)/2",\
         size=10,bbox=dict(boxstyle="square", ec=('#E377C2'),\
         fc=(1., 1., 1),))
plt.xlim(0, 100)
```

Out[4]:
```
(0.0, 100.0)
```



6.8 Solve the initial value problem $y' + y = \sin x\,; \qquad y(\pi) = 1.$

Wolfram Alpha describes the problem as "first-order linear ordinary differential equation." To get a solution to an initial value problem, it is only necessary to separate the initial value(s) with a comma. To check the answer, enter exactly as an ivp, with comma separating the initial value.

In [22]:
```
from sympy import *
from fractions import Fraction as frac

(x, c1) = symbols('x c1')
y = Function('y')(x)
y = frac(1,2)*(exp(pi - x) + sin(x) - cos(x))
y
```

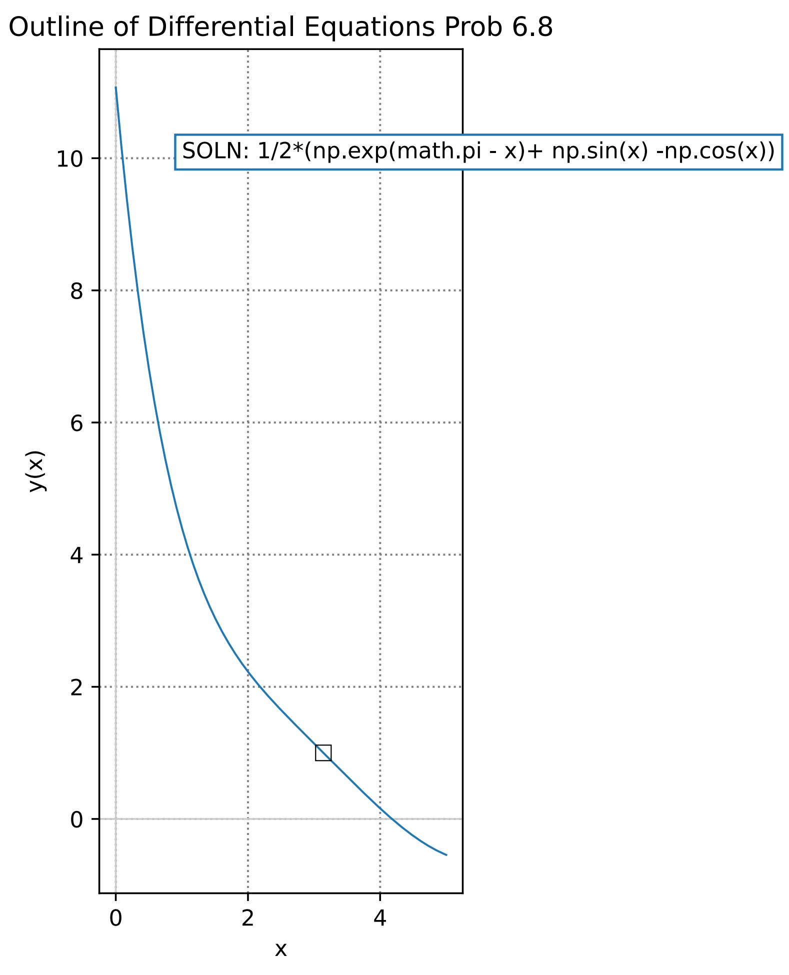
Out[22]: $\frac{e^{\pi - x}}{2} + \frac{\sin(x)}{2} - \frac{\cos(x)}{2}$

In [173]:
```
y = y.subs([(x, pi)])
y
```

Out[173]: 1

On to the plot. Turns out that matplotlib is a little picky when it comes to using pi directly as a parameter, and a reference to the math module seems required. The initial condition of the problem is depicted in the plot as a framed coordinate.

In [8]:
```
import numpy as np
import math
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(0.,5.,300)
y = 1/2*(np.exp(math.pi - x)+ np.sin(x) -np.cos(x))

ax = plt.gca()
ax.axhline(y = 0, color='0.8', linewidth=0.8)
ax.axvline(x = 0, color='0.8', linewidth=0.8)

plt.grid(True, linestyle=':', color='gray', linewidth=0.9)
plt.xlabel('x')
plt.ylabel('y(x)')
plt.title('Outline of Differential Equations Prob 6.8')
ratio = 1.0
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

plt.plot(x,y, linewidth = 0.9)
plt.rcParams['figure.figsize'] = [4, 7]
plt.text(1, 10, "SOLN: 1/2*(np.exp(math.pi - x)+ np.sin(x) -np.cos(x))",\
         size=10,bbox=dict(boxstyle="square", ec=('#1F77B4'),\
         fc=(1., 1., 1),))
apts = np.array([math.pi])
bpts = np.array(1)
plt.plot(apts, bpts, markersize=7, color='k', marker='s', mfc='none',\
         markeredgewidth=0.5)
#plt.ylim(-0.5, 2)
plt.show()
```



6.9 Solve the differential equation $y' - 5y = 0$.

Wolfram Alpha describes the problem as "first-order linear ordinary differential equation."

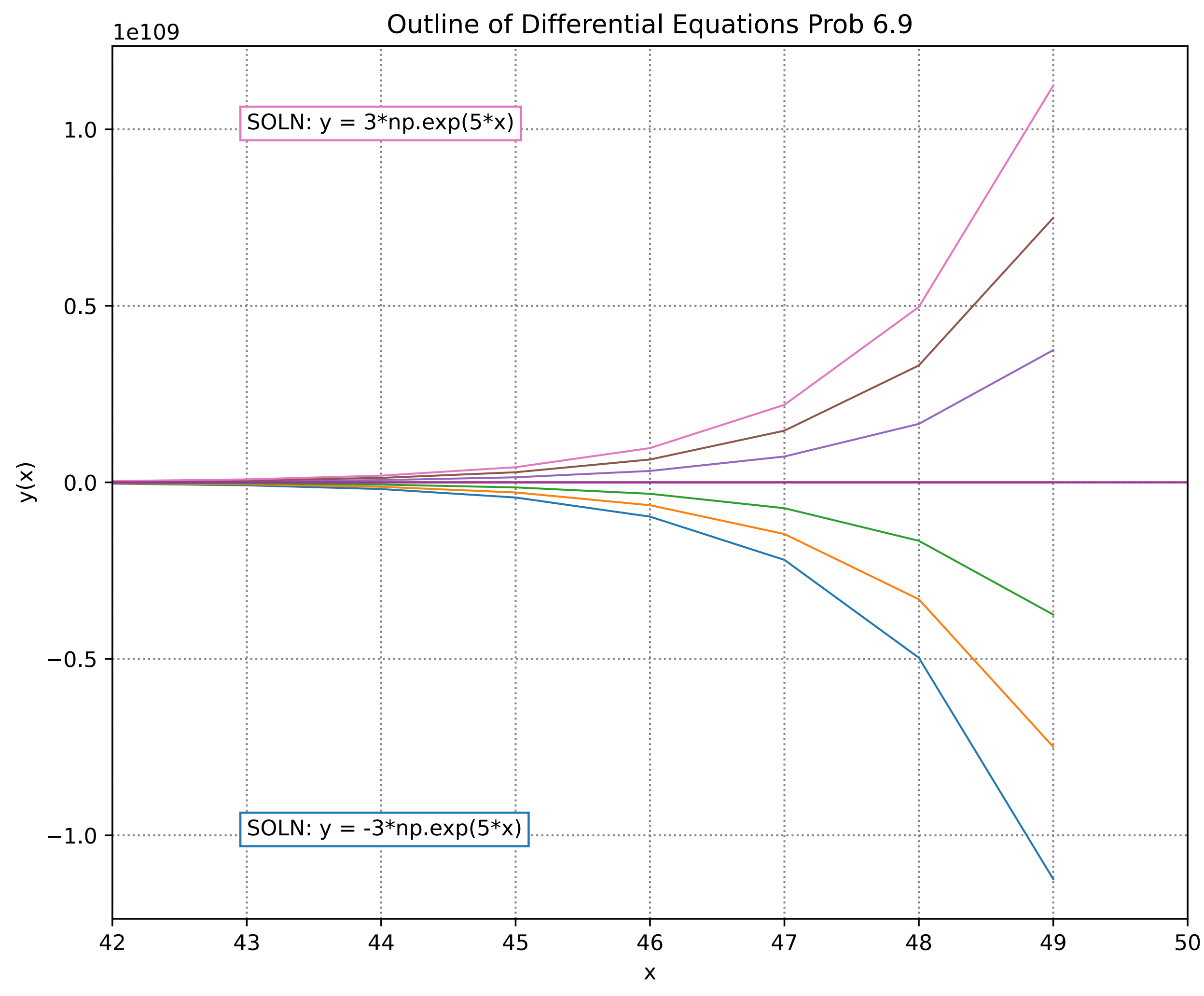
In [216]:
```
from sympy import *

(x, c1) = symbols('x c1')
y = Function('y')(x)
y = c1*exp(5*x)
y
```

Out[216]: $c_1 e^{5x}$

The plot suffers from an extreme of insensitivity. To get a normal view of the iterants, a hyper-extended y-axis scale is necessary, and little curve definition can be achieved within the maximum sustainable interval division, which is 50.

In [24]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

#x = np.linspace(-10, 10, 100, 50)
x = np.linspace(42., 50, 50)

two = np.array([-3,-2,-1, 0, 1, 2, 3])

def f(x):
    for k in two:
        y = k*np.exp(5*x)
        plt.plot(y, linewidth = 0.9)
y=f(x)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 6.9")
plt.rcParams['figure.figsize'] = [9, 7.5]

ax = plt.gca()
ax.axhline(y=0, color = '#993399', linewidth=1)
ax.axvline(x=0, color = '#993399', linewidth=1)

plt.text(43, -1e109, "SOLN: y = -3*np.exp(5*x)", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(43, 1e109, "SOLN: y = 3*np.exp(5*x)", size=10,bbox=dict\
         (boxstyle="square", ec=('#E377C2'),fc=(1., 1., 1),))
plt.xlim(42, 50)
```

Out[24]:
```
(42.0, 50.0)
```



6.10 Solve the differential equation $\frac{dz}{dx} - xz = -x$.

In Wolfram Alpha the equation is entered as $z' - xz = -x$.

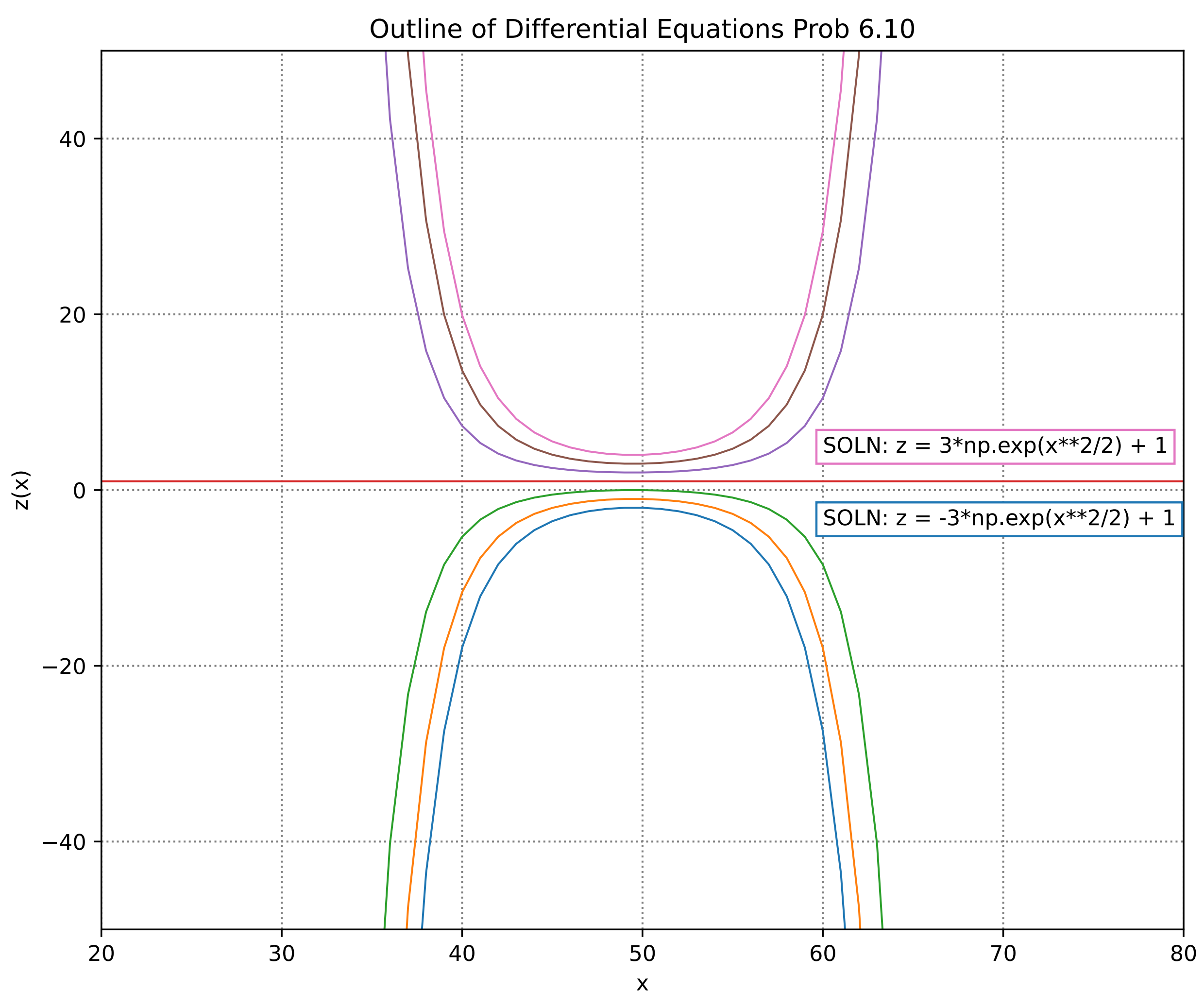
In [27]:
```
from sympy import *

(x, c1) = symbols('x c1')
z = Function('z')(x)
z = c1*exp(x**2/2) + 1
z
```

Out[27]: $c_1 e^{\frac{x^2}{2}} + 1$

And then the matplotlib plot. Strangely, inclusion of the customary cartesian axes crashes the plot. Also, though the final plot externally displays values of z, internally several matplotlib key terms need to refer to y.

In [72]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.setdiff1d(np.linspace(-10.,10,100),[0]) #To remove zero

two = np.array([-3,-2,-1, 0, 1, 2, 3])

def f(x):
    for k in two:
        y = k*np.exp(x**2/2) + 1
        plt.plot(y, linewidth = 0.9)
y=f(x)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("z(x)")
plt.title("Outline of Differential Equations Prob 6.10")
plt.rcParams['figure.figsize'] = [9, 7.5]

ax = plt.gca()

plt.text(60, -4, "SOLN: z = -3*np.exp(x**2/2) + 1", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(60, 4.25, "SOLN: z = 3*np.exp(x**2/2) + 1", size=10,bbox=dict\
         (boxstyle="square", ec=('#E377C2'),fc=(1., 1., 1),))
plt.ylim(-50, 50)
plt.xlim(20, 80)
```

Out[72]:
```
(20.0, 80.0)
```



6.11 Solve the initial value problem $z' - xz = -x$; $z(0) = -4$.

The Wolfram Alpha category label deems this to be "first-order linear ordinary differential equation." In the plot, again, references to z need to be disguised internally. The initial condition point is framed.

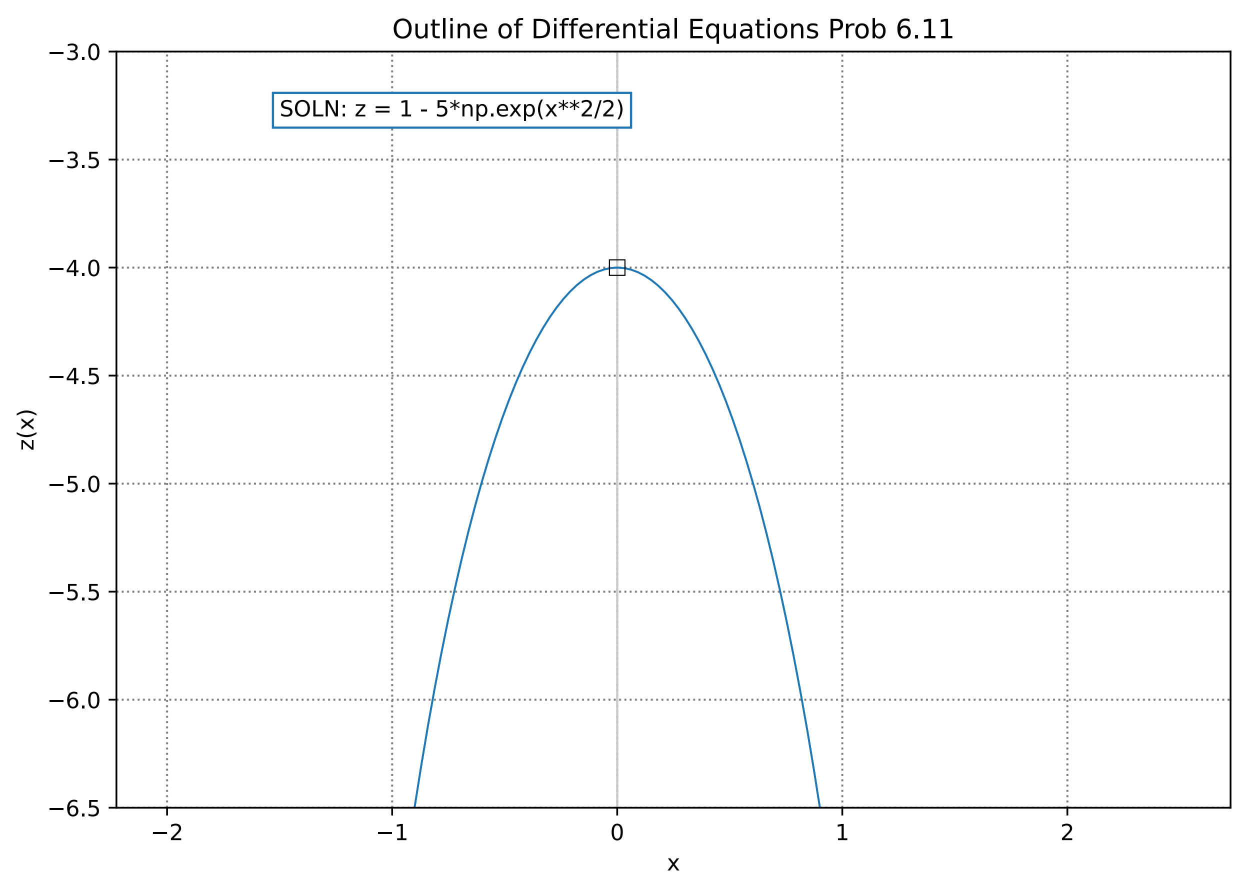
In [75]:
```
from sympy import *

(x, c1) = symbols('x c1')
z = Function('z')(x)
z = 1 - 5*exp(x**2/2)
z
```

Out[75]: $1 - 5e^{\frac{x^2}{2}}$

In [76]:
```
z = z.subs([(x, 0)])
z
```

Out[76]: -4

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(-2.,2.5,300)
y = 1 - 5*np.exp(x**2/2)

ax = plt.gca()
ax.axhline(y = 0, color='0.8', linewidth=1)
ax.axvline(x = 0, color='0.8', linewidth=1)

plt.grid(True, linestyle=':', color='gray', linewidth=0.9)
plt.xlabel('x')
plt.ylabel('z(x)')
plt.title('Outline of Differential Equations Prob 6.11')
ratio = 0.96
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

plt.plot(x,y, linewidth = 0.9)
plt.rcParams['figure.figsize'] = [7, 5]
plt.text(-1.5, -3.3, "SOLN: z = 1 - 5*np.exp(x**2/2)", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
apts = np.array([0])
bpts = np.array([-4])
plt.plot(apts, bpts, markersize=7, color='k', marker='s', mfc='none', \
         markeredgewidth=0.5)
plt.ylim(-6.5, -3)
plt.show()
```



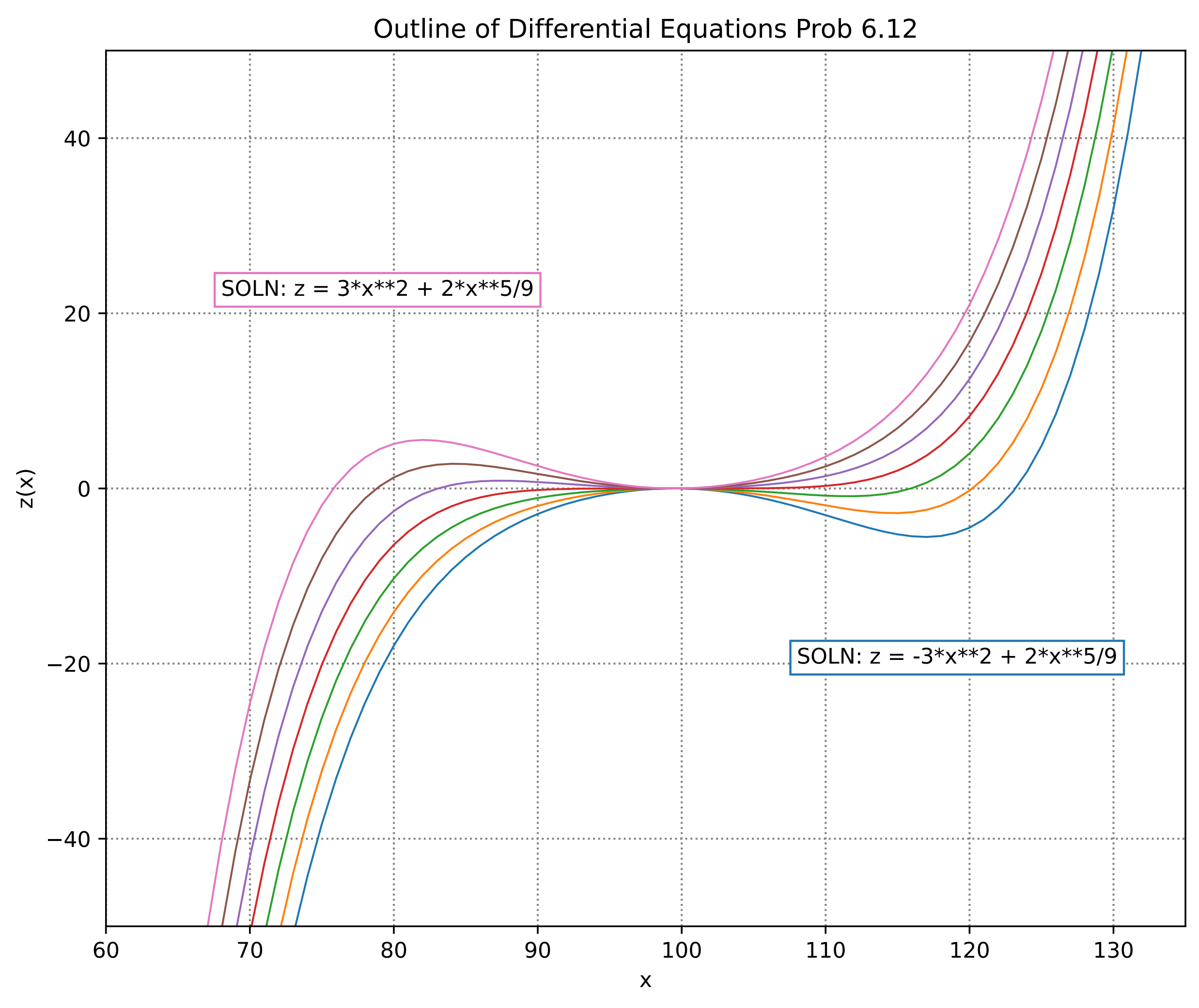
6.12 Solve $z' - \frac{2}{x}z = \frac{2}{3}x^4$.

The Wolfram Alpha category label deems this to be "first-order linear ordinary differential equation." In the plot, again, references to z need to be disguised internally. For some reason the cartesian axes are still prohibited.

In [137]:
```
from sympy import *

(x, c1) = symbols('x c1')
z = Function('z')(x)
z = c1*x**2 + 2*x**5/9
z
```

Out[137]: $c_1x^2 + \frac{2x^5}{9}$

In [170]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(-10.,10,200)

two = np.array([-3,-2,-1, 0, 1, 2, 3])

def f(x):
    for k in two:
        y = k*x**2 + 2*x**5/9
        plt.plot(y, linewidth = 0.9)
y=f(x)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("z(x)")
plt.title("Outline of Differential Equations Prob 6.12")
plt.rcParams['figure.figsize'] = [9, 7.5]

ax = plt.gca()

plt.text(108, -20, "SOLN: z = -3*x**2 + 2*x**5/9", size=10,bbox=dict\
        (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(68, 22, "SOLN: z = 3*x**2 + 2*x**5/9", size=10,bbox=dict\
        (boxstyle="square", ec=('#E377C2'),fc=(1., 1., 1),))
plt.ylim(-50, 50)
plt.xlim(60, 135)
```
Out[170]:
```
(60.0, 135.0)
```



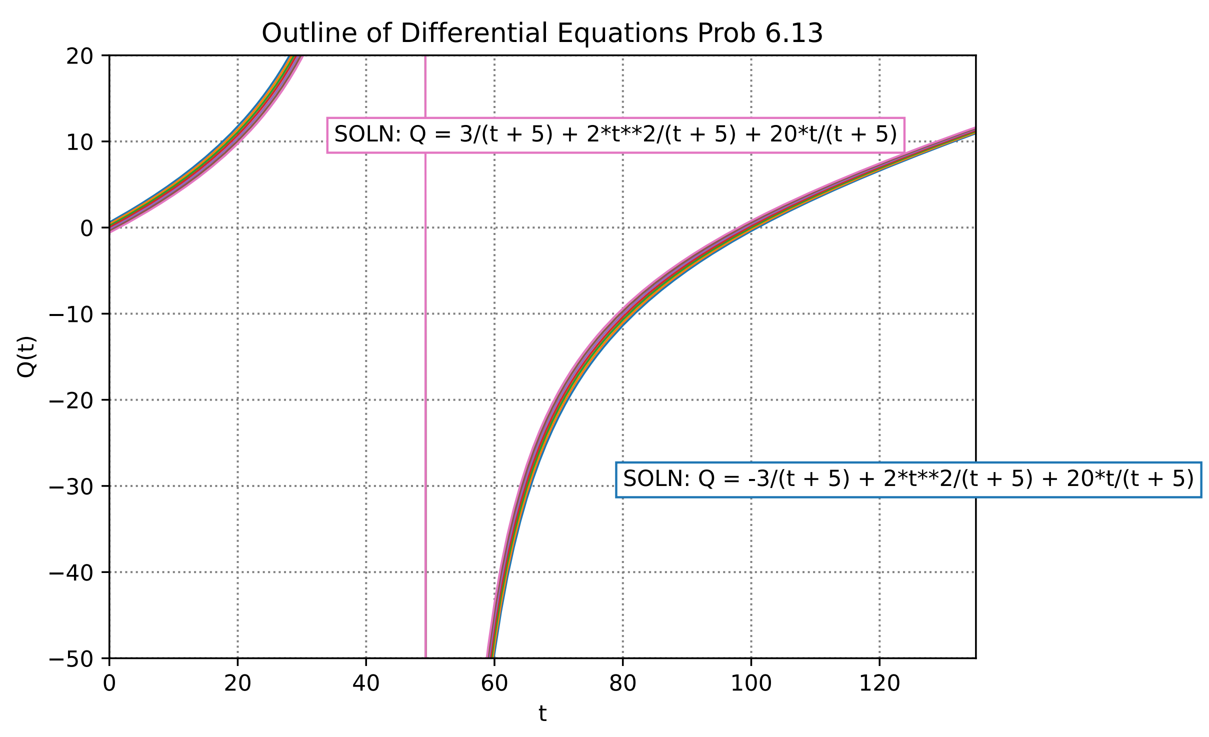
6.13 Solve $\frac{dQ}{dt} + \frac{2}{10+2t}Q = 4$.

In [163]:
```
from sympy import *

(t, c1) = symbols('t c1')
Q = Function('Q')(t)
Q = c1/(t + 5) + 2*t**2/(t + 5) + 20*t/(t + 5)
Q
```
Out[163]:

$$\frac{c_1}{t+5} + \frac{2t^2}{t+5} + \frac{20t}{t+5}$$

Below: Plotting a tightly grouped collection of functions which evidently does not depend much on the arbitrary constant.

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(-10.,10,200)

two = np.array([-3,-2,-1, 0, 1, 2, 3])

def f(x):
    for k in two:
        y = k/(x + 5) + 2*x**2/(x + 5) + 20*x/(x + 5)
        plt.plot(y, linewidth = 0.9)
y=f(x)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("t")
plt.ylabel("Q(t)")
plt.title("Outline of Differential Equations Prob 6.13")
plt.rcParams['figure.figsize'] = [9, 7.5]

ax = plt.gca()

plt.text(80, -30, "SOLN: Q=-3/(t + 5) + 2*t**2/(t + 5) + 20*t/(t + 5)",\
         size=10,bbox=dict\
        (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(35, 10, "SOLN: Q=3/(t + 5) + 2*t**2/(t + 5) + 20*t/(t + 5)",\
         size=10,bbox=dict\
        (boxstyle="square", ec=('#E377C2'),fc=(1., 1., 1),))
plt.ylim(-50, 20)
plt.xlim(0, 135)
```
Out[10]:
```
(0.0, 135.0)
```



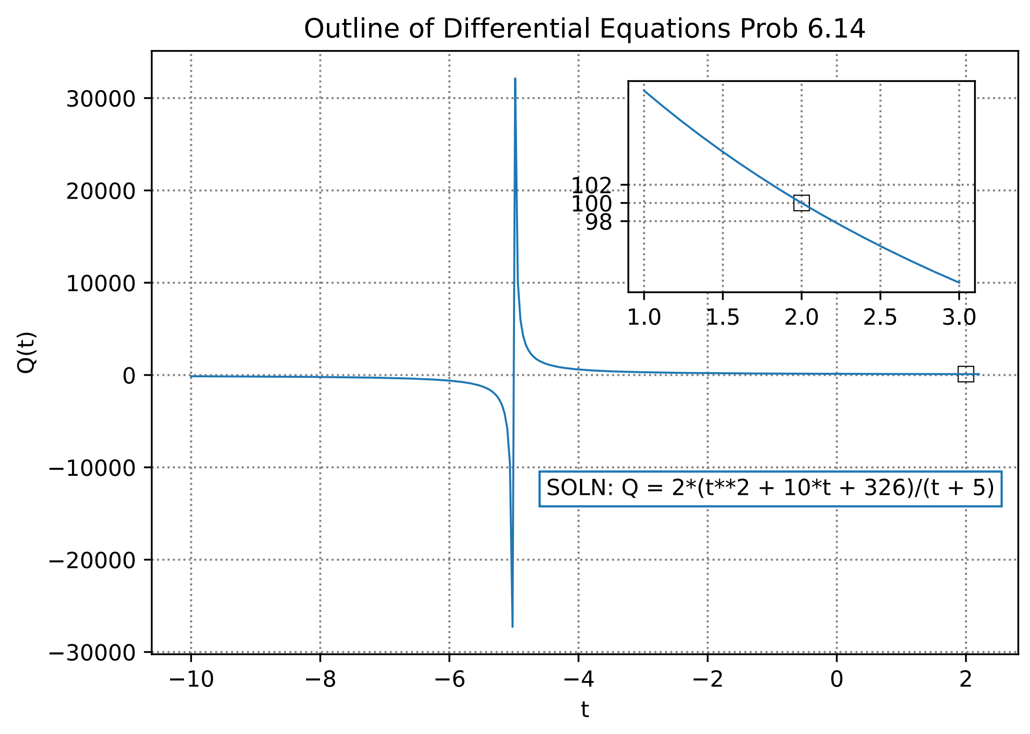
6.14 Solve the initial value problem $\frac{dQ}{dt} + \frac{2}{10+2t}Q = 4; \qquad Q(2) = 100.$

Another initial value problem. In the following plot, an inset shows a locator which would be indistinguishable against the huge vertical scale.

In [182]:
```
from sympy import *

(t) = symbols('t')
Q = Function('Q')(t)
Q = 2*(t**2 + 10*t + 326)/(t + 5)
Q
```
Out[182]:
$$\frac{2t^2 + 20t + 652}{t+5}$$

In [184]:
```
Q = Q.subs([(t, 2)])
Q
```
Out[184]: 100

In [259]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(-10.,2.2,300)
y = 2*(x**2 + 10*x + 326)/(x + 5)
x1 = np.linspace(1.0,3,200)
y1 = 2*(x1**2 + 10*x1 + 326)/(x1 + 5)

fig, ax = plt.subplots()
axin1 = ax.inset_axes([0.55, 0.6,
                        0.4, 0.35])
axin1.set_yticks([0, 98, 100, 102],
              labels=['0', '98',  '100', '102' ])
axin1.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
ax.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.title('Outline of Differential Equations Prob 6.14')
plt.xlabel('t')
plt.ylabel('Q(t)')

ax.text(-4.5, -13000, "SOLN: Q=2*(t**2 + 10*t + 326)/(t + 5)", size=10,\
        bbox=dict(boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
apts = np.array([2])
bpts = np.array([100])
ax.plot(apts, bpts, markersize=7, color='k', marker='s', mfc='none',\
        markeredgewidth=0.5)
axin1.plot(apts, bpts, markersize=7,color='k',marker='s',mfc='none',\
        markeredgewidth=0.5)
axin1.labelsize = 5

ax.plot(x, y, linewidth = 0.9)
axin1.plot(x1, y1, linewidth = 0.9)
```
Out[259]:
```
[<matplotlib.lines.Line2D at 0x7f4382af4850>]
```



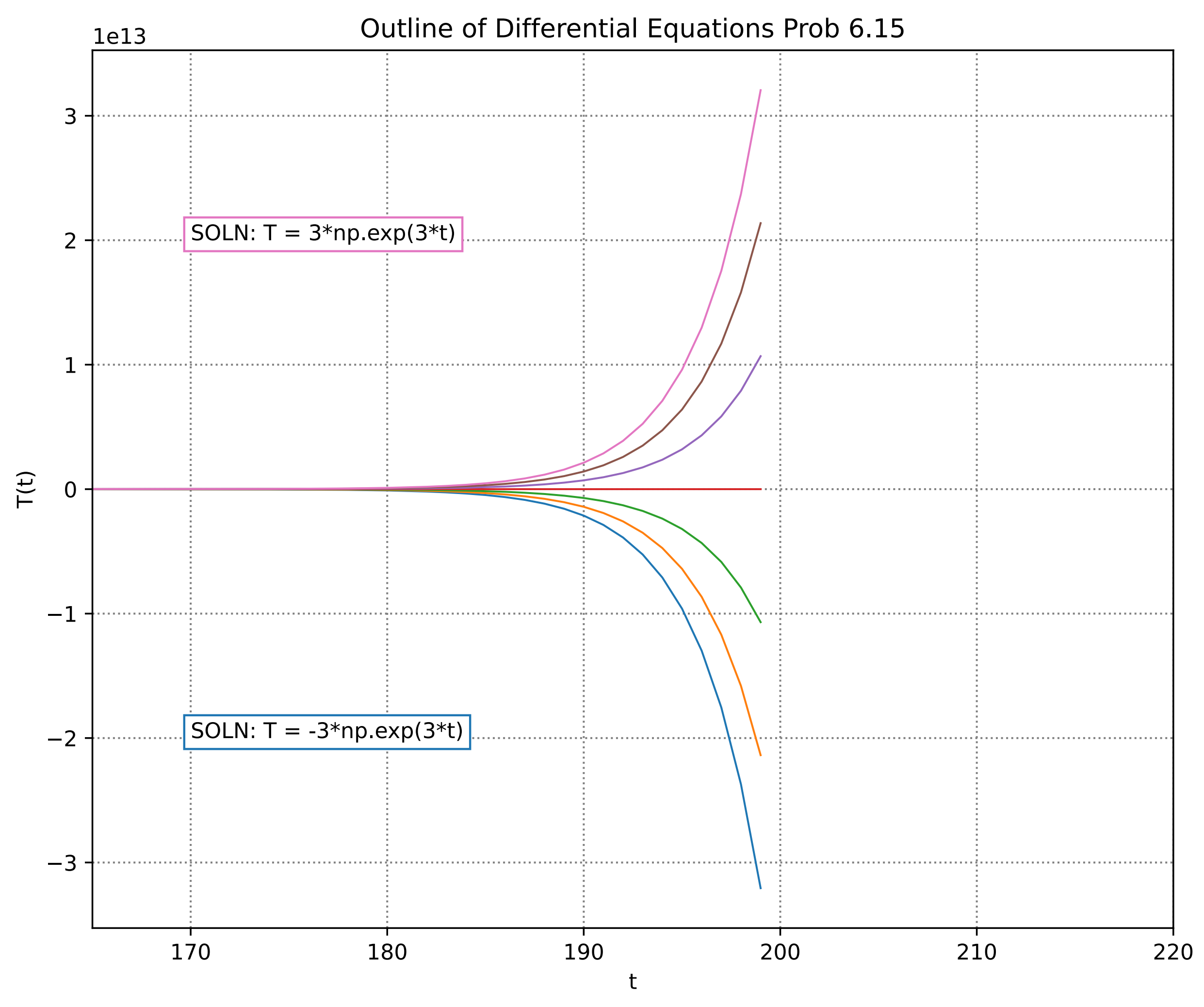
6.15 Solve $\frac{dT}{dt} + kT = 100k$.

Because the problem equation does not show a free t parameter, it is necessary to enter the equation into Wolfram Alpha just as it appears above, in order for W|A to catch the drift.

In [297]:
```
from sympy import *

(c1, k, t) = symbols('c1 k t')
T = Function('T')(t)
T = c1*exp(-k*t) + 100
T
```
Out[297]: $c_1 e^{-kt} + 100$

In the current problem solution there are two constants. Experimentation shows that the k constant has little effect, considering the large vertical scale. Therefore it is shown in the plot with a value of 1. The outer constant, c_1 , has an dispersion effect similar to those seen in several plots above. Note that the interval division factor for linspace_x is 200 here, giving a much smoother plot that the 50 factor of problem 6.9.

In [318]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(-10., 10, 200)

two = np.array([-3,-2,-1, 0, 1, 2, 3])

def f(x):
    for k in two:
        y = k*np.exp(3*x)
        plt.plot(y, linewidth = 0.9)
y=f(x)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("t")
plt.ylabel("T(t)")
plt.title("Outline of Differential Equations Prob 6.15")
plt.rcParams['figure.figsize'] = [9, 7.5]

ax = plt.gca()

plt.text(170, -2e13, "SOLN: T = -3*np.exp(3*t)", size=10,bbox=dict\
        (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(170, 2e13, "SOLN: T = 3*np.exp(3*t)", size=10,bbox=dict\
        (boxstyle="square", ec=('#E377C2'),fc=(1., 1., 1),))
#plt.ylim(-10, 30)
plt.xlim(165, 220)
plt.show()
```



6.16 Solve $y' + xy = xy^2$.

In [335]:
```
from sympy import *

(x, c1) = symbols('x c1')
y = Function('y')(x)
y = exp(c1 + x**2/2) + 1
y
```

Out[335]: $e^{c_1+\frac{x^2}{2}} + 1$

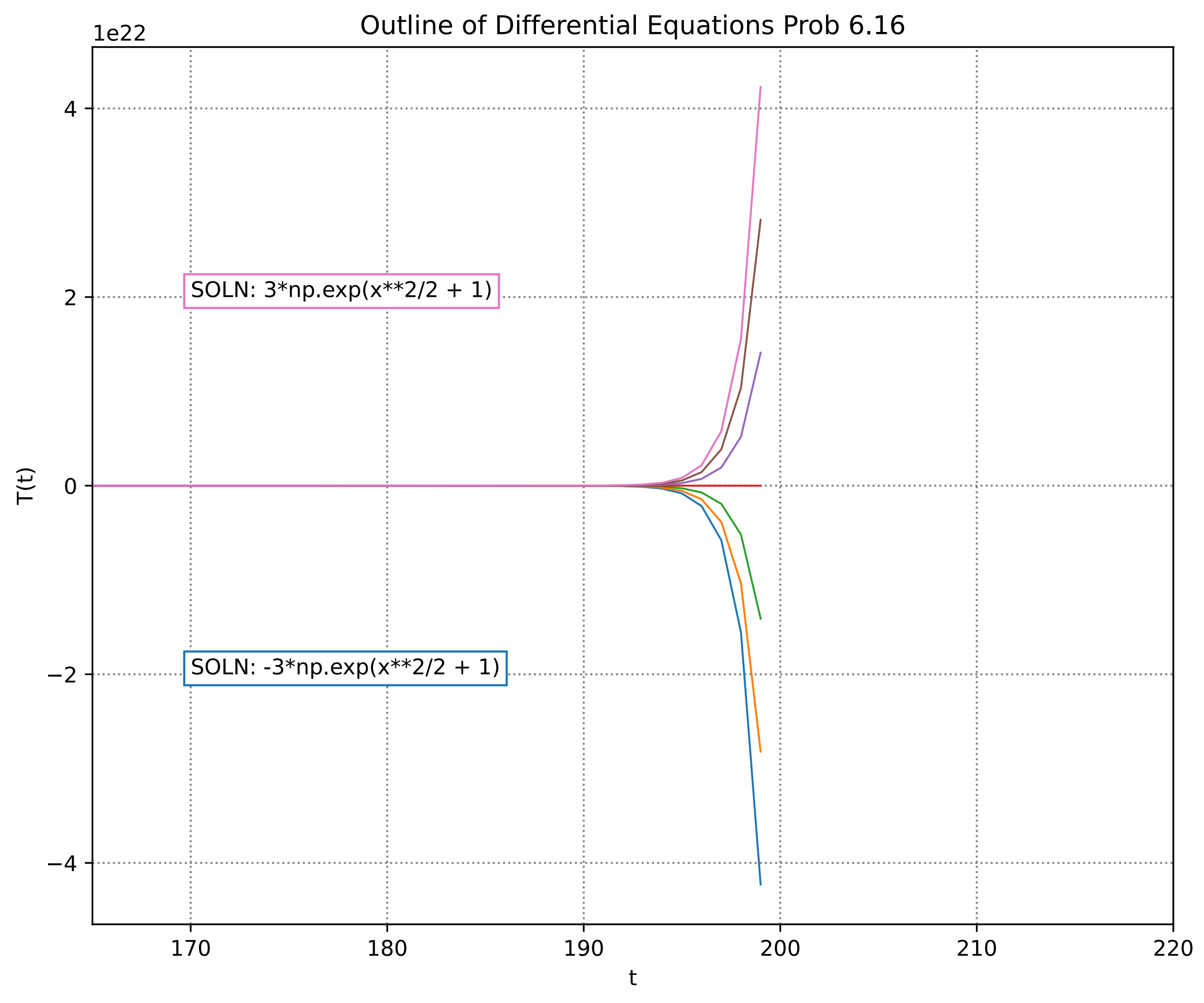
The expression shown in the cell above is the one found by Wolfram Alpha. The one shown in the cell below is the one in the text. The two forms can be reconciled, although with different values for the constant c_1 .

In [327]:
```
ta = c1*exp(x**2/2) + 1
ta
```

Out[327]: $c_1 e^{\frac{x^2}{2}} + 1$

The interval division factor of 200 in the following plot would normally be considered sufficient for a fairly smooth plot. But in the face of the huge vertical scale, the curves which are actually plotted are in fact very coarse.

In [6]:
```python
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(-10., 10, 200)

two = np.array([-3,-2,-1, 0, 1, 2, 3])

def f(x):
    for k in two:
        y = k*np.exp(x**2/2 + 1)
        plt.plot(y, linewidth = 0.9)
y=f(x)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("t")
plt.ylabel("T(t)")
plt.title("Outline of Differential Equations Prob 6.16")
plt.rcParams['figure.figsize'] = [9, 7.5]

ax = plt.gca()

plt.text(170, -2e22, "SOLN: -3*np.exp(x**2/2 + 1)", size=10,bbox=dict\
        (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(170, 2e22, "SOLN: 3*np.exp(x**2/2 + 1)", size=10,bbox=dict\
        (boxstyle="square", ec=('#E377C2'),fc=(1., 1., 1),))
plt.xlim(165, 220)
plt.show()
```



6.17 Solve $y' - \frac{3}{x}\, y = x^4\, y^{1/3}$.

In order to get Wolfram Alpha to offer a solution, it is necessary to request the 'principal root' instead of the 'real root'.

Below, sympy transcribes the text solution into a pretty printed version.

In [90]:
```python
from sympy import *
from fractions import Fraction as frac

(x, c1) = symbols('x c1')
y = Function('y')(x)
y = (frac(1,27))*(x**2*(1 + 2*x**3))**(3/2)
y
```

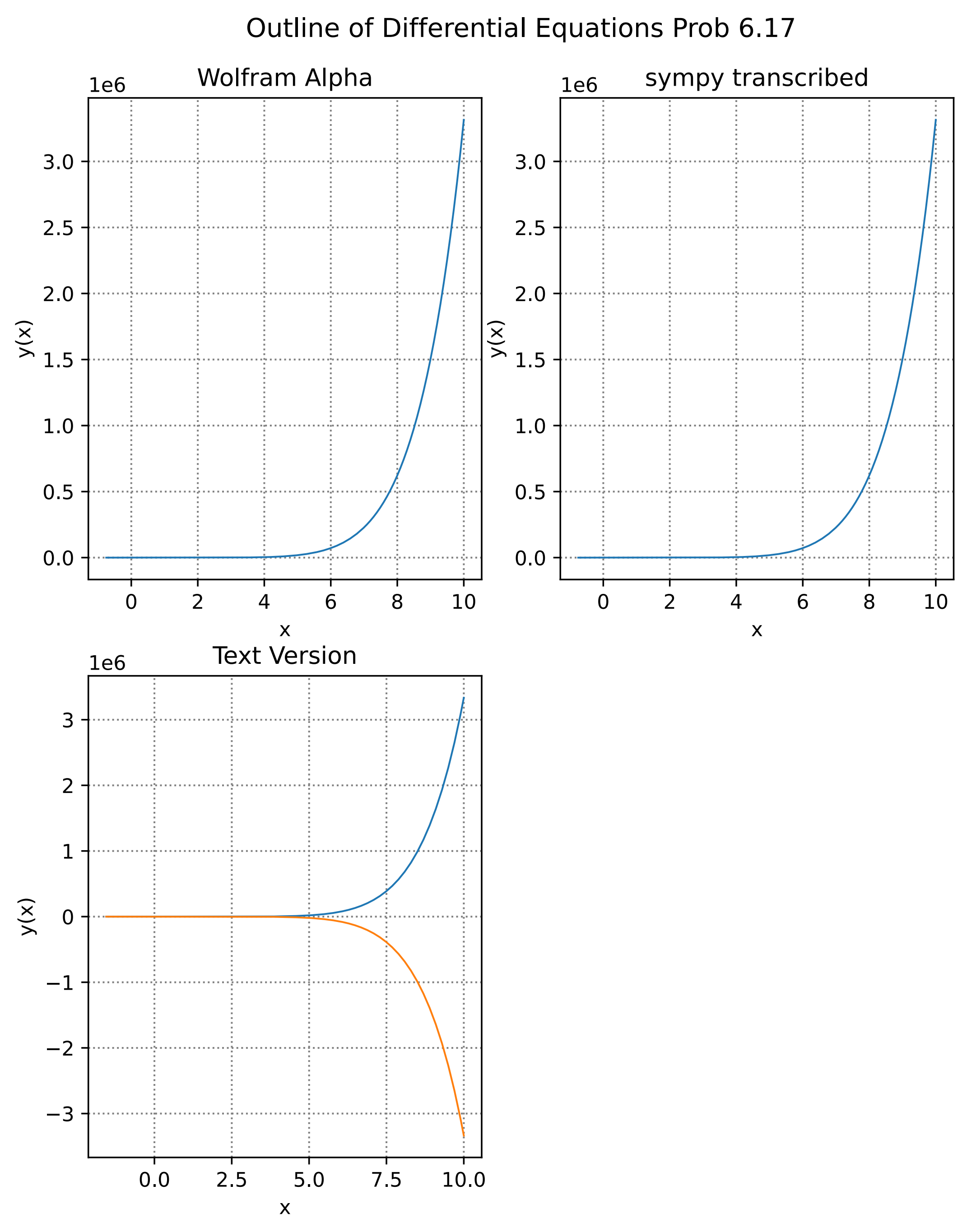
Out[90]:
$$\frac{\left(x^2 \cdot \left(2x^3+1\right)\right)^{1.5}}{27}$$

Just for the record, the form of the solution dispensed by Wolfram Alpha differs somewhat from the cell above. It is: $\frac{1}{27}\left(x^2\left(c_1 + 2x^3\right)\right)^{\frac{3}{2}}$. In the case of all plots constructed below, the constant c_1 is assigned a value of 1.

About runtime warnings. Python, or actually matplotlib, issues a runtime warning whenever a coordinate is served which has an imaginary component. This is logical, since in 2D there is no way to fit the imaginary components onto a cartesian plane. In the set of plots below, four runtime warnings are issued normally, one for each reference to cubic. This holds whenever the proscribed expression appears, whether destined to be incorporated into a curve or not. Any negative number raised to the 3/2 power will yield an imaginary component, thus the warnings. For purposes of displaying these solution alternatives, runtime warnings have been suppressed. The two lines which effect the suppression can be seen in the cell below as lines 7 and 8.

Of the three plot solutions examined below, the text answer is the only one that has the insight to provide for a negative function arm. Otherwise, in form and in scale, the three solutions look equal. (The two versions which do not possess a negative arm have been gifted with one, retained in commented-out form.)

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt
from fractions import Fraction as frac

%config InlineBackend.figure_formats = ['svg']

import warnings
warnings.filterwarnings("ignore")

plt.figure(figsize=[12, 7])
plt.suptitle('Outline of Differential Equations Prob 6.17', x=0.45,\
            y=1.23, fontsize=13)

xa = np.linspace(-10., 10, 200)
y1p = (1/27)*(xa**2*(1 + 2*xa**3))**(3/2)
#y1n = -(1/27)*(xa**2*(1 + 2*xa**3))**(3/2)

plt.subplot(2, 2, 1)
plt.plot(xa, y1p, linewidth = 0.9)
#plt.plot(xa, y1n, linewidth = 0.9)
plt.title('Wolfram Alpha')
plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel('x')
plt.ylabel('y(x)')

plt.subplot(2, 2, 2)
xs = np.linspace(-10., 10, 200)
ys = (1/27)*(xs**2*(2*xs**3 + 1))**(3/2) #text1
#ysn = -(1/27)*(xs**2*(2*xs**3 + 1))**(3/2) #text1
plt.plot(xs,ys, linewidth = 0.9)
#plt.plot(xs,ysn, linewidth = 0.9)
plt.title('sympy transcribed')
plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel('x')
plt.ylabel('y(x)')

plt.subplot(2, 2, 3)
xt = np.linspace(-10., 10, 200)
ytp = (xt**2 + 2/9*xt**5)**(3/2) #text1
ytn = -(xt**2 + 2/9*xt**5)**(3/2) #text2
plt.plot(xt,ytp, linewidth = 0.9)
plt.plot(xt,ytn, linewidth = 0.9)
plt.title('Text Version')
plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel('x')
plt.ylabel('y(x)')

plt.subplots_adjust(left= 0.2,
                    bottom=0.1,
                    right=0.7,
                    top=1.15,
                    wspace=0.2,
                    hspace=0.2)
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: