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.

In this section, both Wolfram Alpha and sympy have a chance to get a workout.

Note: When transcribing text for entry into Wolfram Alpha the "fenceposts" !| ... |! are excluded from the text.

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

Wolfram Alpha can solve the problem. In the input field, the entry is made:

!| y' -3 y=6 |!

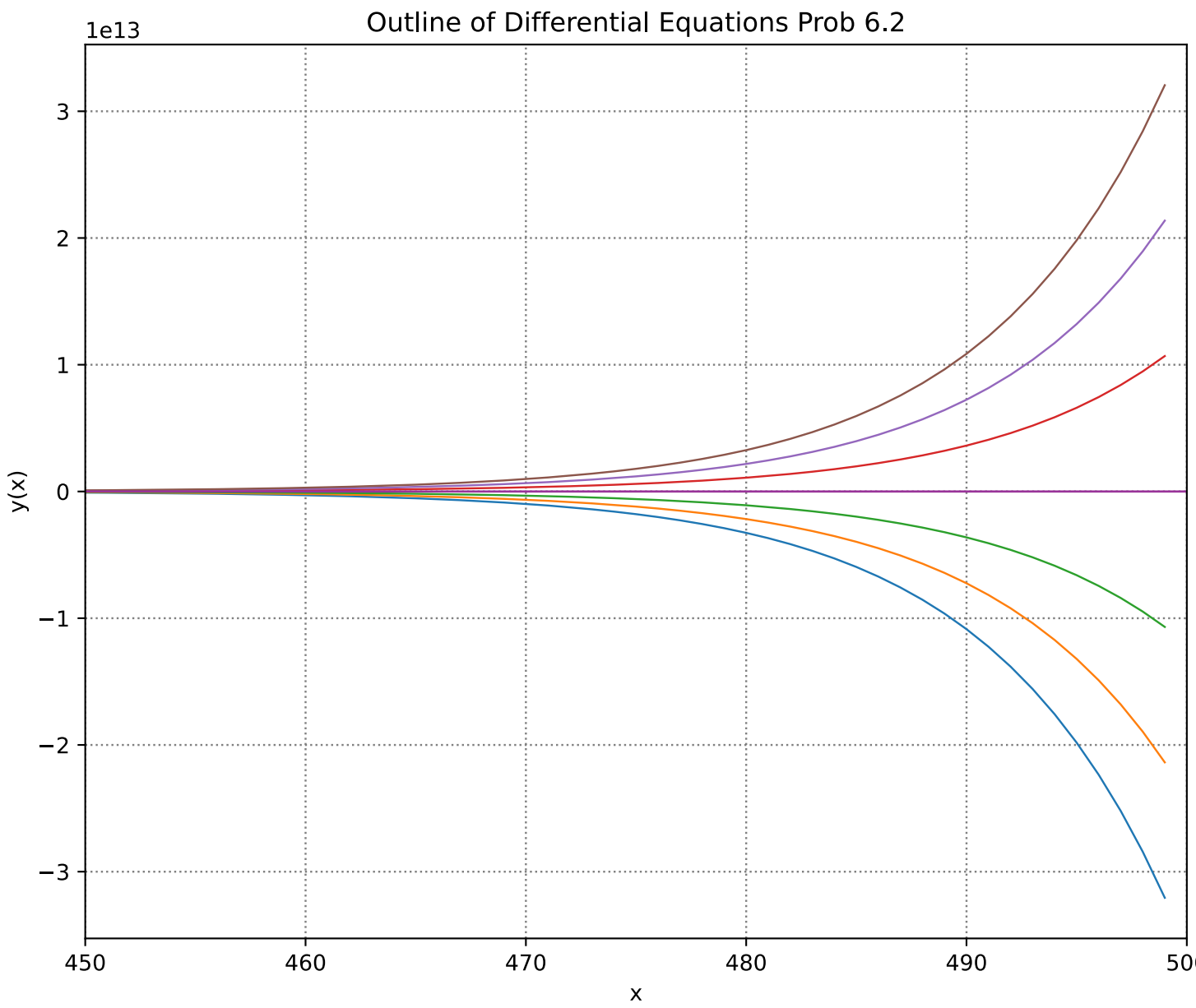
and the answer is returned:

$$y(x) = c_1 e^{3x} - 2$$

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." The entry line is:

!| y' -2 x y=x |!

and the returned answer is:

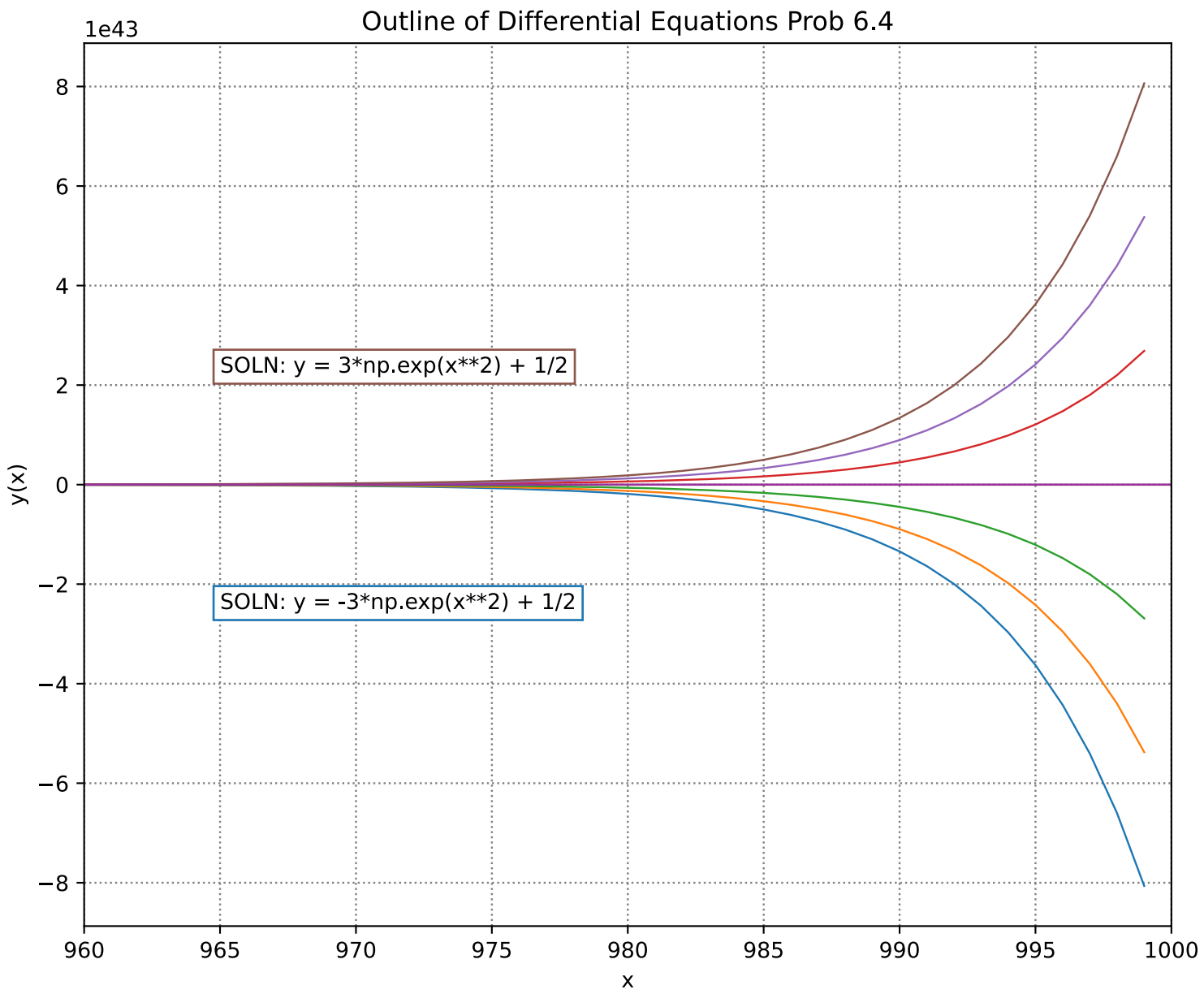
$$y(x) = c_1 e^{x^2} - \frac{1}{2}$$

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." The entry line is:

!| y' +4 y/x=x^4 |!

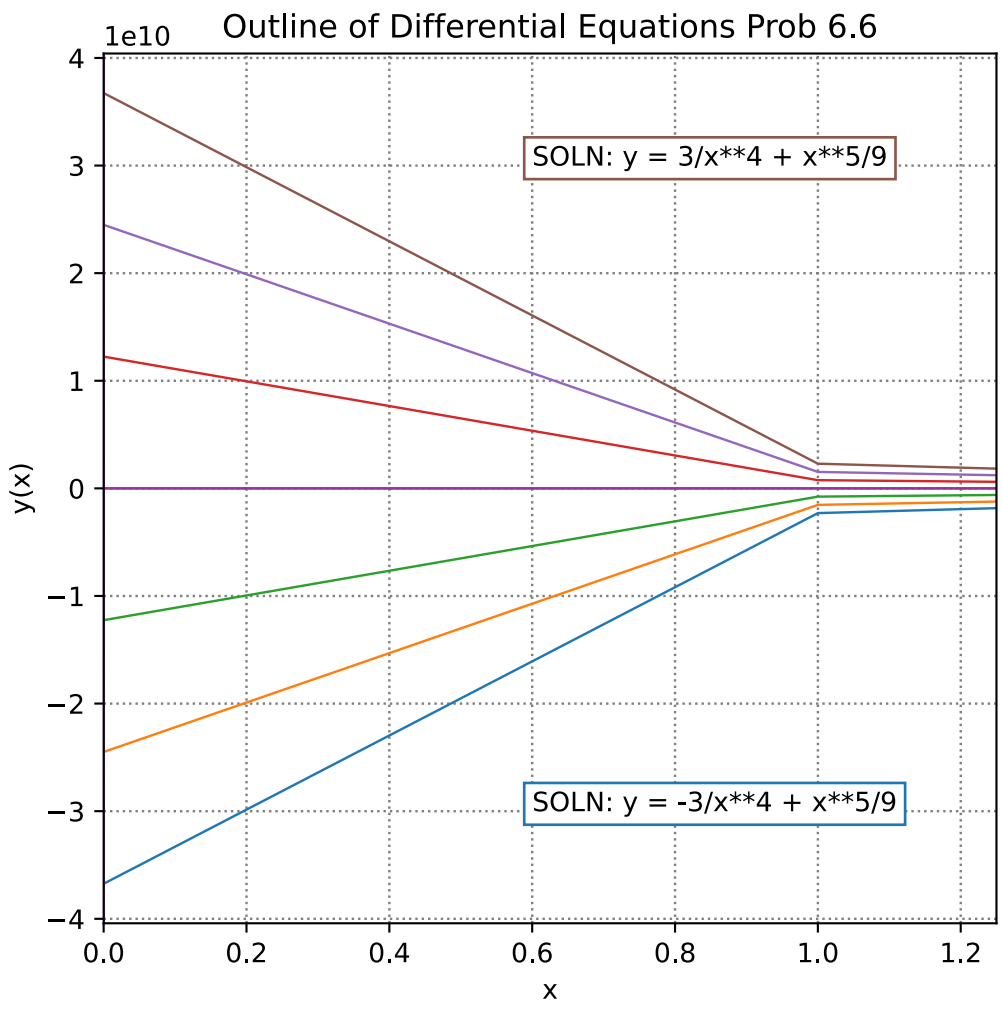
and the returned answer is:

$$y(x) = \frac{c_1}{x^4} + \frac{x^5}{9}$$

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." The entry line is:

!|y' +y=sin(x) |!

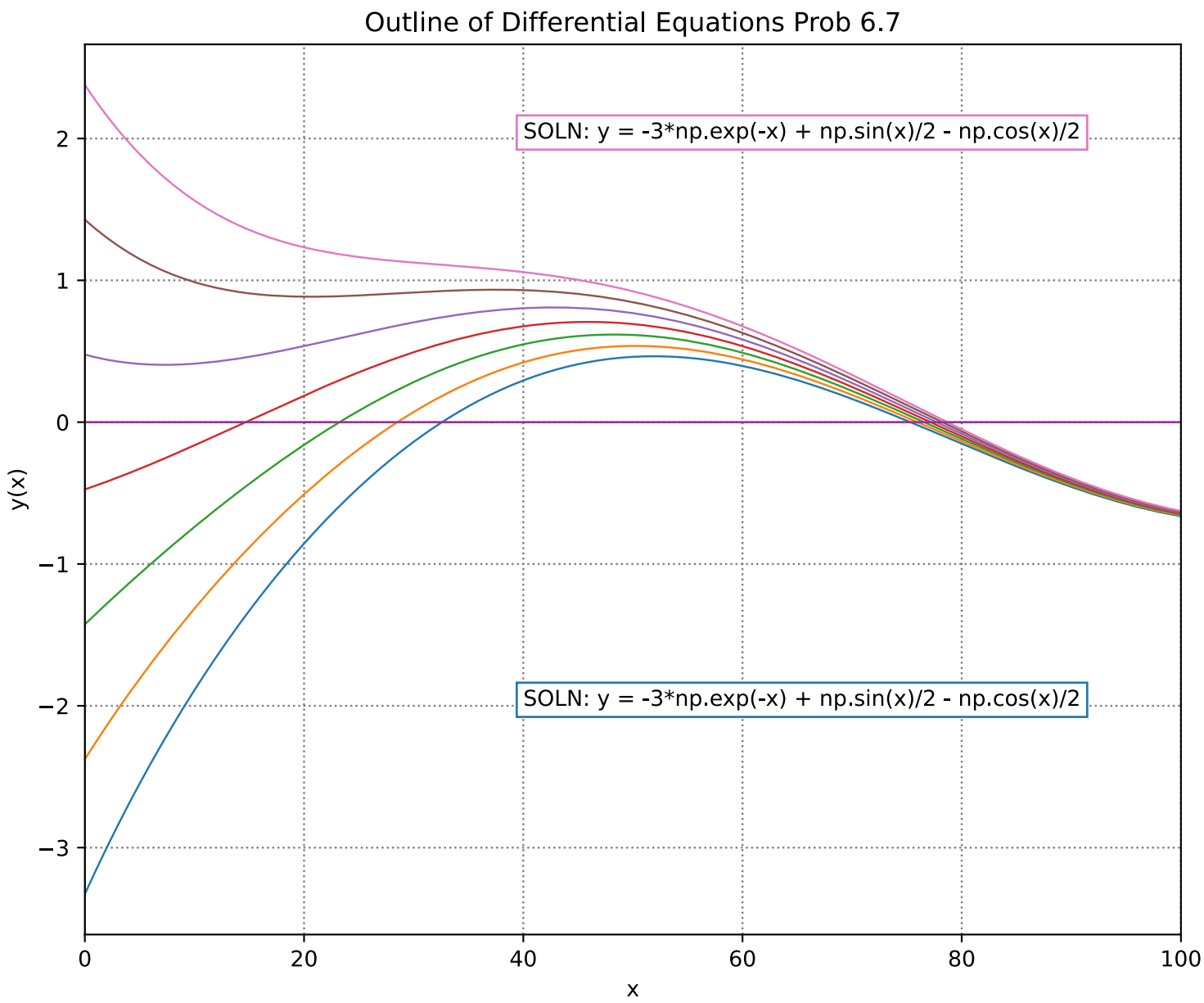
and the returned answer is:

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

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 $y' + y = \sin x$; $y(\pi) = 1$.

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.

Wolfram Alpha describes the problem as "first-order linear ordinary differential equation." The entry line is:

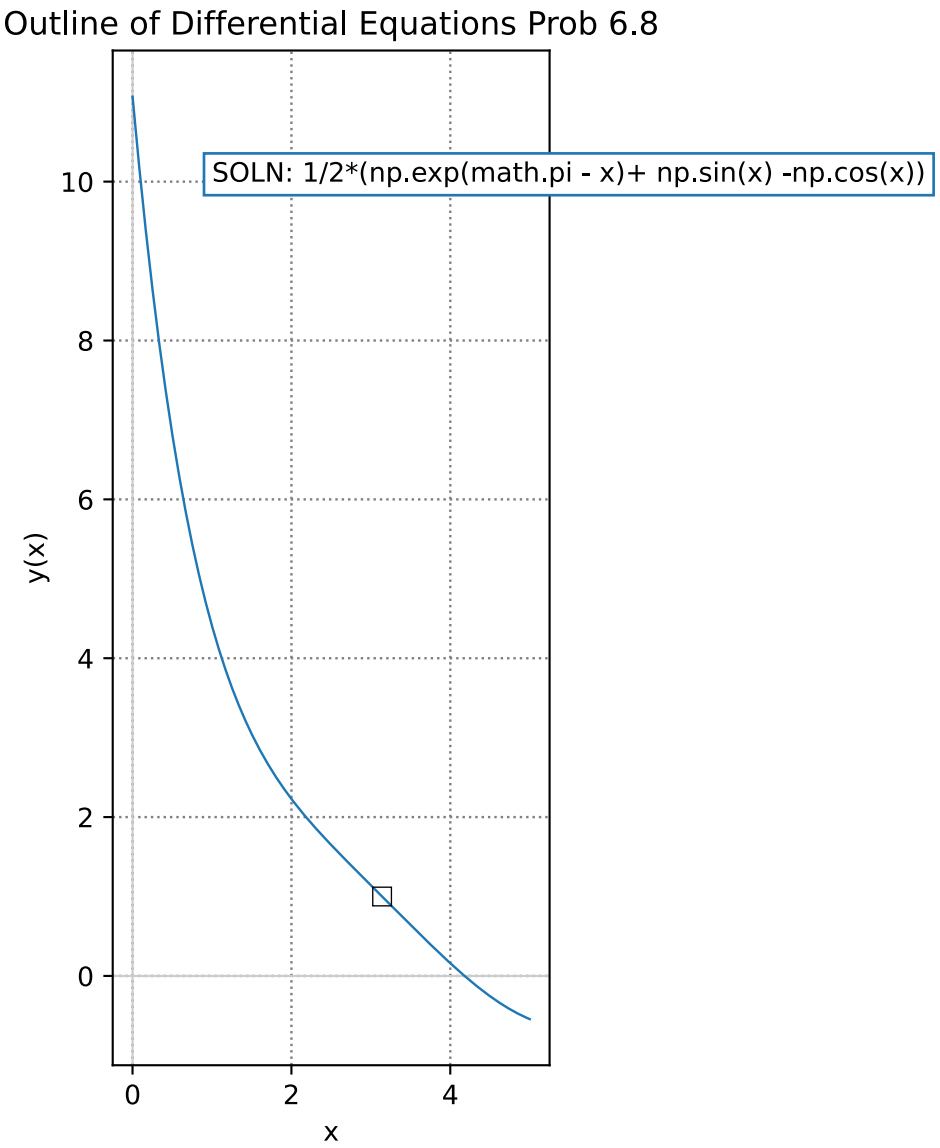
!|y' +y=sin(x),y(pi)=1 |!

and the returned answer is:

$$y(x) = \frac{1}{2}(e^{\pi - x} + \sin(x) - \cos(x))$$

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." But it can also be worked with Python. (Using two import lines to import sympy may look ridiculous, but "import everything" avoids having to tag the referrring commands with a prefix, while the alias form makes clear which module is intended if two modules are imported with dueling functions.)

In [98]:
```
from sympy import *
import sympy as sp
x = sp.Symbol('x')
y = sp.Function('y')(x)
```

```
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)
```

In [99]:
```
ode = Eq(yp - 5*y, 0)
ode
```

Out[99]:
$$-5y(x) + \frac{d}{dx}y(x) = 0$$

In [100]:
```
sol = dsolve(ode, y)
sol
```

Out[100]:
$$y(x) = C_1 e^{5x}$$

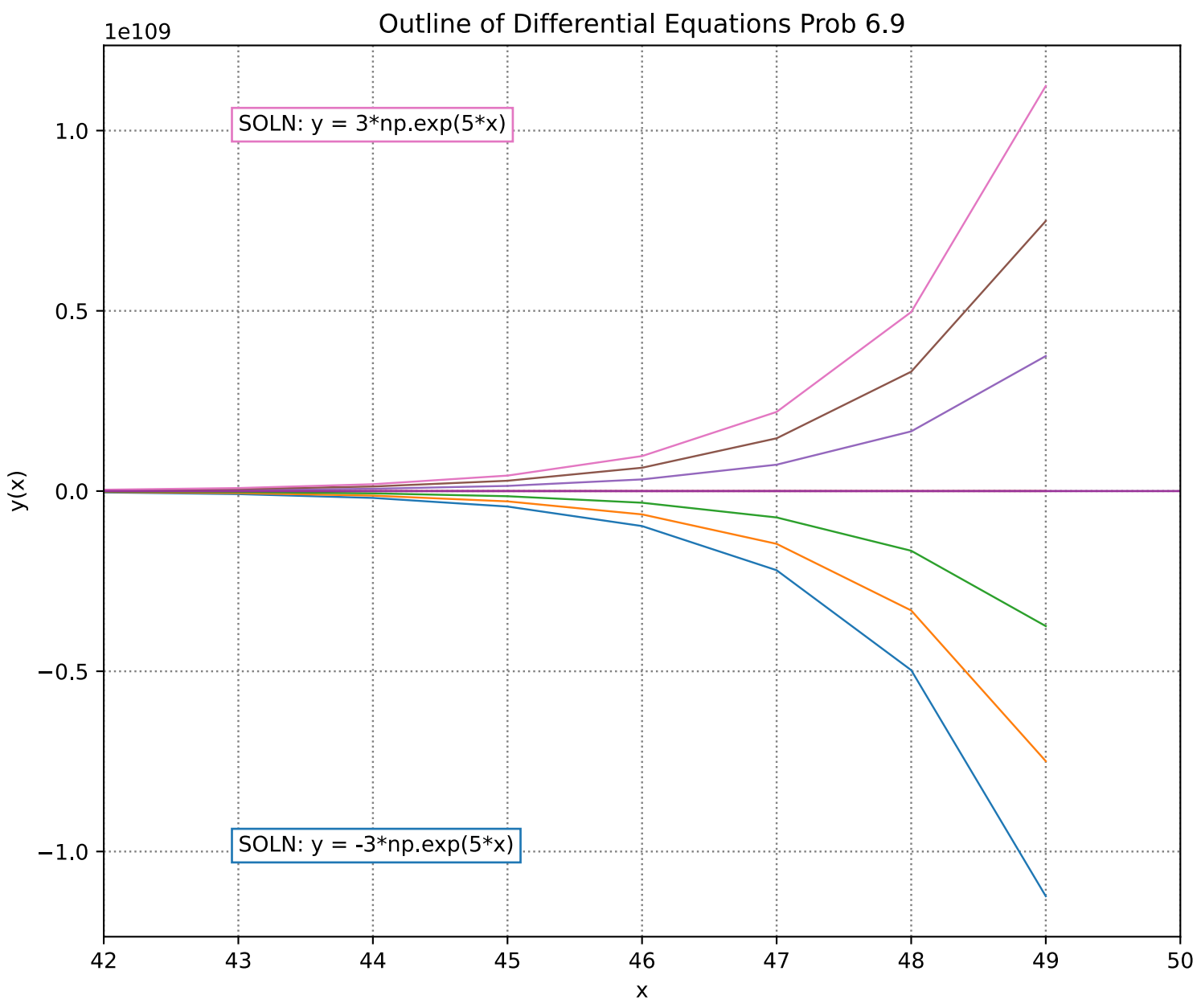
In [101]:
```
checkodesol(ode,sol)
```

Out[101]:
```
(True, 0)
```

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 [74]:
```
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[74]: (42.0, 50.0)



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

Python's sympy is also capable of solving ODEs. In this case the symbol 'y' was substituted for 'z', to make the ground friendly for matplotlib.

In [8]:
```
import sympy as sp
x = sp.Symbol('x')
y = sp.Function('y')(x)
```

```
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)
```

In [10]:
```
ode = sp.Eq(yp - x*y, -x)
ode
```

Out[10]:
$$-xy(x) + \frac{d}{dx}y(x) = -x$$

In [26]:
```
sol = dsolve(ode, y)
sol
```

Out[26]:
$$y(x) = C_1 e^{\frac{x^2}{2}} + 1$$

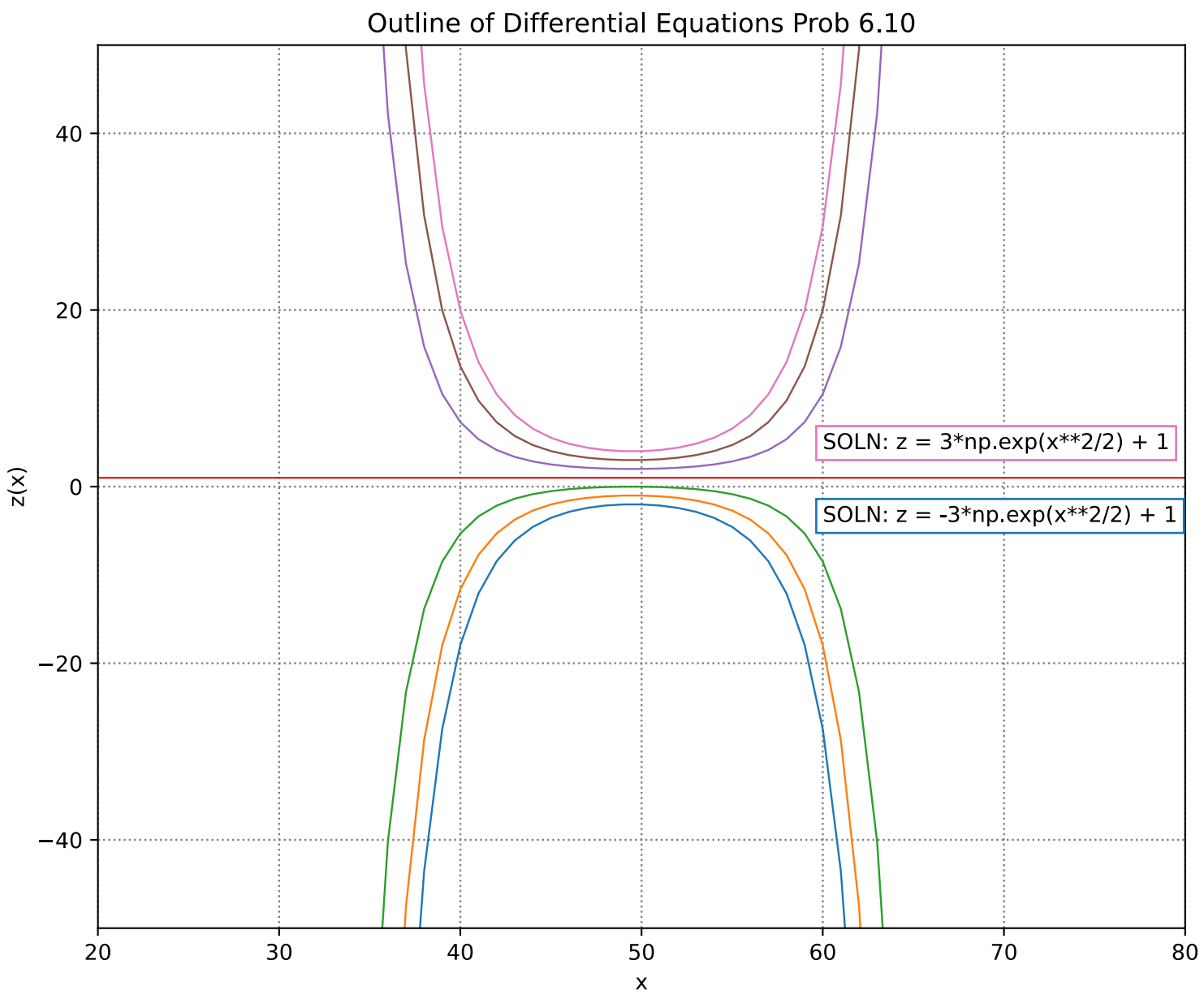
In [27]:
```
checkodesol(ode,sol)
```

Out[27]:
```
(True, 0)
```

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 $z' - xz = -x$; $z(0) = -4$.

Continuing with a sympy orientation.

In [23]:
```
import sympy as sp
x = sp.Symbol('x')
y = sp.Function('y')(x)
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)


```

In [25]:
```
ode = Eq(yp - x*y, -x)
ode
```

Out[25]:
$$-xy(x) + \frac{d}{dx}y(x) = -x$$

In [28]:
```
ics = {y.subs(x, 0): -4}
ics
```

Out[28]:
```
{y(0): -4}
```

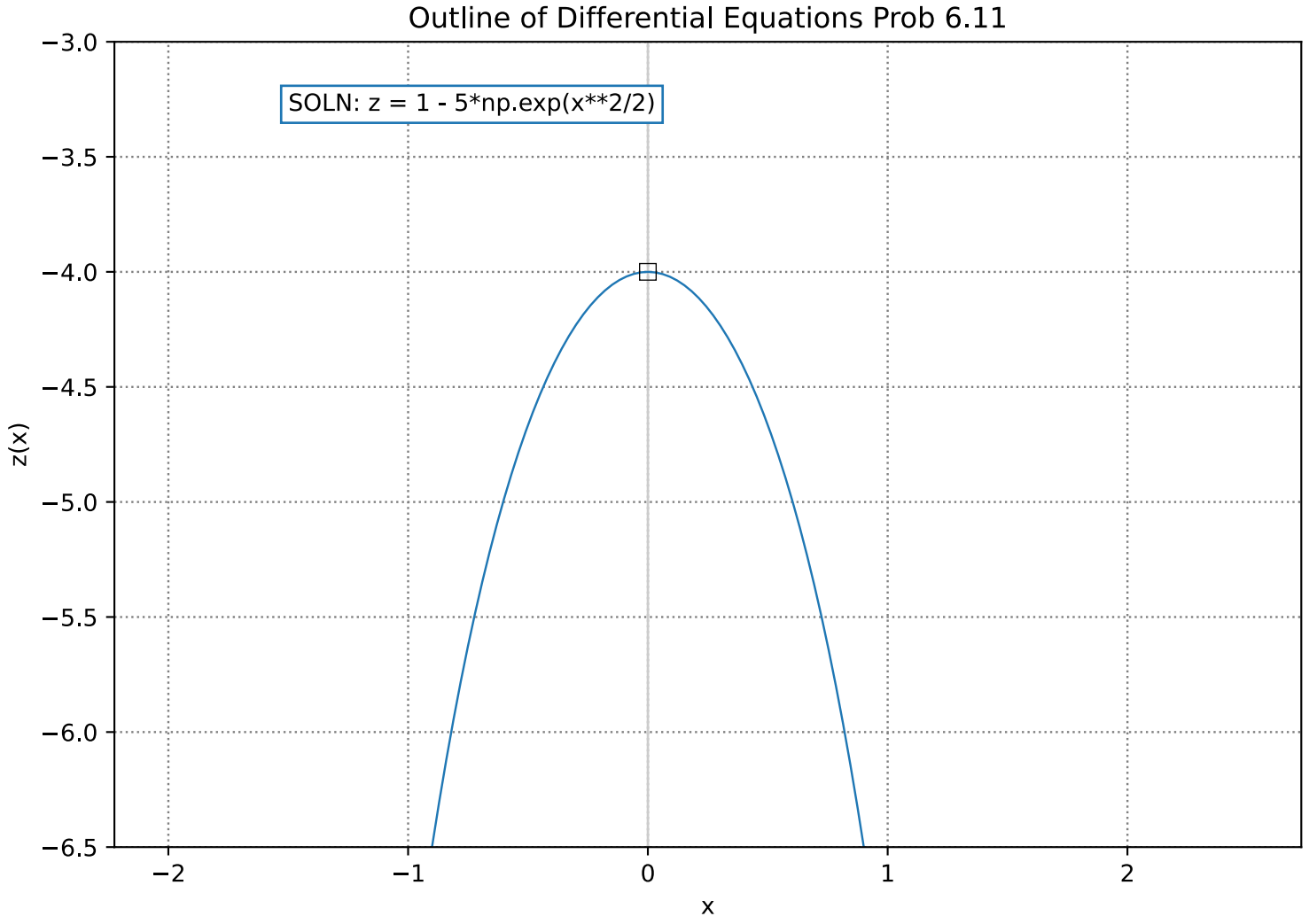
In [29]:
```
ivp = dsolve(ode, ics = ics)
ivp
```

Out[29]:
$$y(x) = 1 - 5e^{\frac{x^2}{2}}$$

In [30]:
```
checkodesol(ode, ivp)
```

Out[30]:
```
(True, 0)
```

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

No initial conditions apply to this problem.

In [44]:
```
import sympy as sp
x = sp.Symbol('x')
y = sp.Function('y')(x)
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)

```

In [45]:
```
ode = Eq(yp - 2*y/x, (2*x**4)/3)
ode

```

Out[45]:
$$\frac{d}{dx}y(x) - \frac{2y(x)}{x} = \frac{2x^4}{3}$$

In [46]:
```
sol = dsolve(ode,y)
sol

```

Out[46]:
$$y(x) = x^2 \left(C_1 + \frac{2x^3}{9}\right)$$

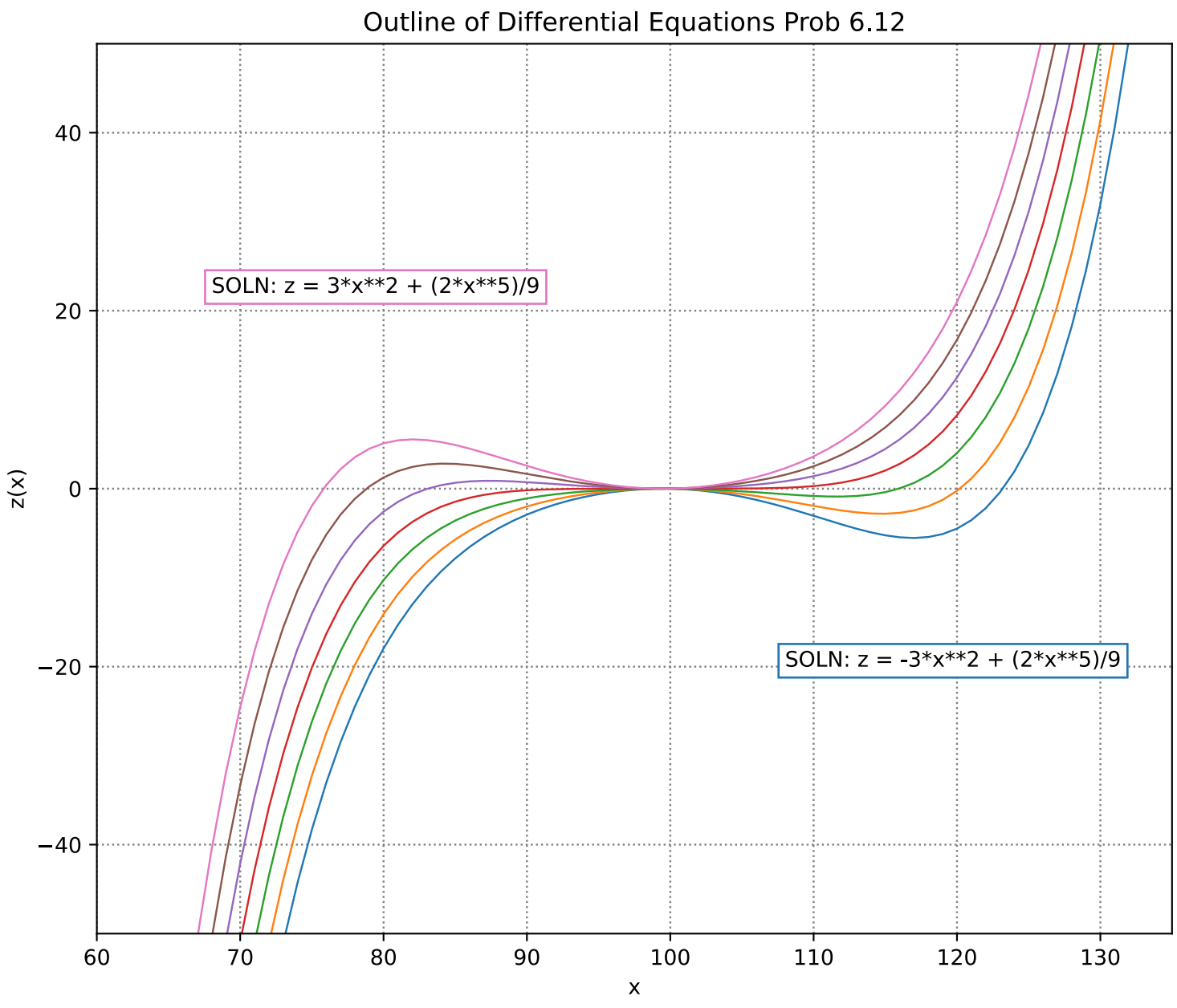
In [47]:
```
checkodesol(ode,sol)

```

Out[47]:
```
(True, 0)
```

In [49]:

```
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[49]: (60.0, 135.0)



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

In [52]:

```
import sympy as sp
x = sp.Symbol('x')
y = sp.Function('y')(x)
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)

```

In [54]:
```
ode = Eq(yp + 2/(10+2*x), 4)
ode
```

Out[54]:
$$\frac{d}{dx}y(x) + \frac{2}{2x+10} = 4$$

In [55]:
```
sol = dsolve(ode,y)
sol
```

Out[55]:
$$y(x) = C_1 + 4x - \log(x+5)$$

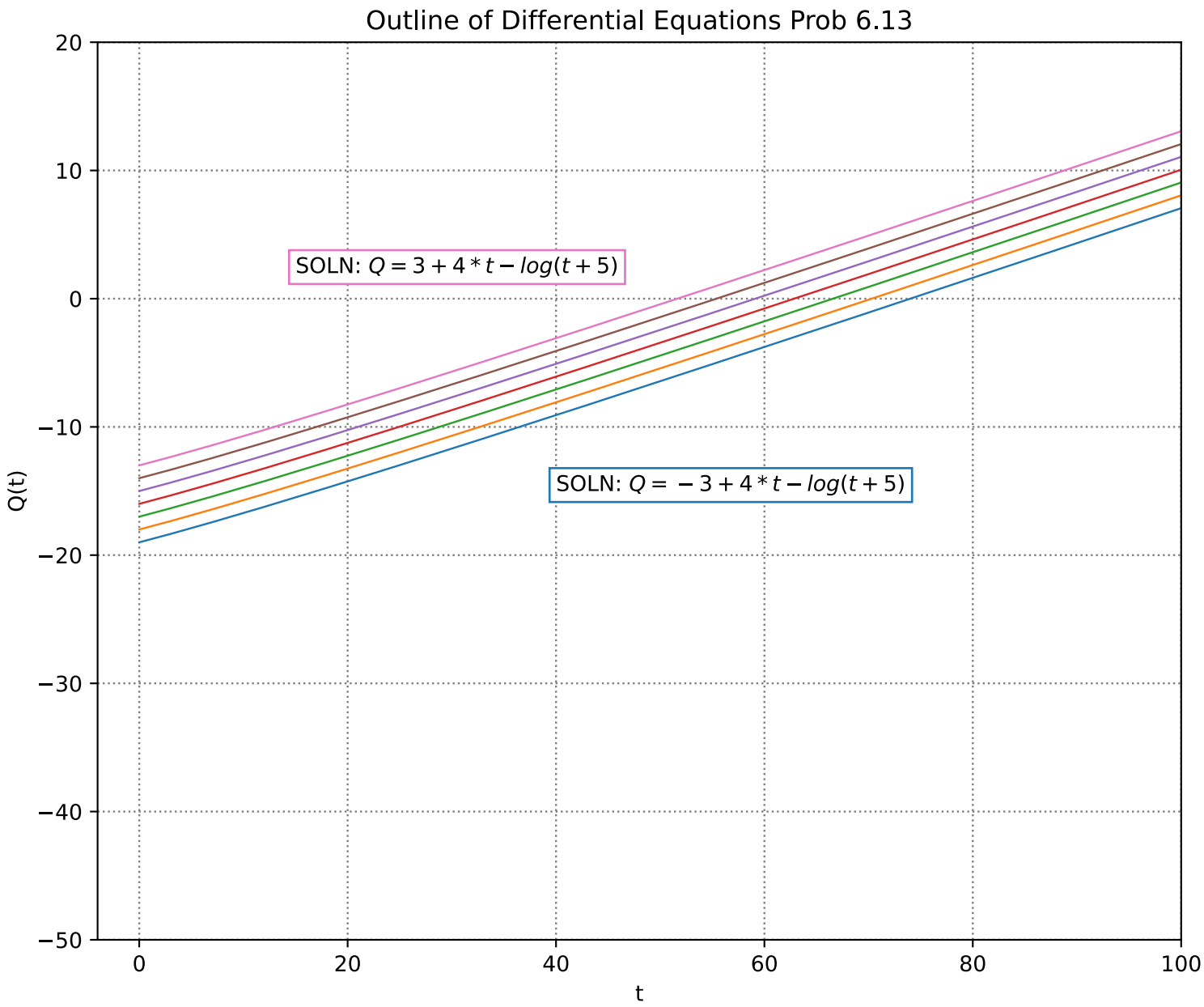
In [56]:
```
checkodesol(ode,sol)
```

Out[56]:
```
(True, 0)
```

Below: Plotting a group of functions which are apparently aligned linearly.

In [69]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(-4.,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)
        y = k + 4*x - np.log(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(40, -15, "SOLN: $Q=-3 + 4*t - log(t + 5)$",\
         size=10,bbox=dict\
        (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(15, 2, "SOLN: $Q=3 + 4*t - log(t + 5)$",\
         size=10,bbox=dict\
        (boxstyle="square", ec=('#E377C2'),fc=(1., 1., 1),))
plt.ylim(-50, 20)
plt.xlim(-4, 100)

```

Out[69]: (-4.0, 100.0)



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 the Python cells, temporary aliases keep things familair.

In [75]:
```
import sympy as sp
x = sp.Symbol('x')
y = sp.Function('y')(x)
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)

```

In [76]:
```
ode = Eq(yp + 2*y/(10 + 2*x), 4)
ode

```

Out[76]:
$$\frac{d}{dx}y(x) + \frac{2y(x)}{2x + 10} = 4$$

In [77]:
```
ics = {y.subs(x, 2): 100}
ics

```

Out[77]:
```
{y(2): 100}
```

In [78]:
```
ivp = dsolve(ode, ics = ics)
ivp

```

Out[78]:
$$y(x) = \frac{2x^2 + 20x + 652}{x + 5}$$

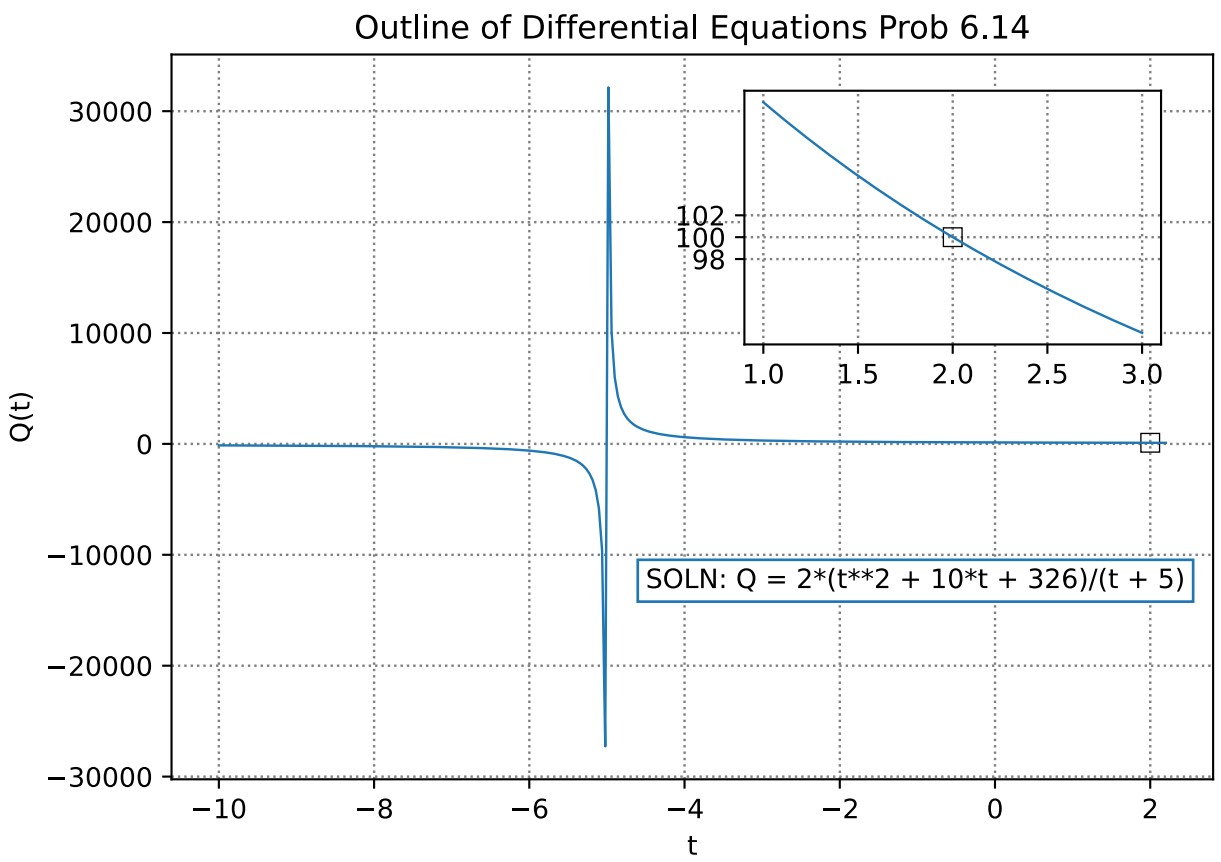
In [79]:
```
checkodesol(ode, ivp)

```

Out[79]:
```
(True, 0)
```

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

Wolfram Alpha describes the problem as a "separable equation". The entry line is:

!|dy/dx + k y=100 k |!

and the returned answer is:

$$y(x) = C_1 e^{-kx} + 100$$

At this point some substitutions could align the nomenclature with that of the given equation.

A check can be made to compare sympy's path regarding this problem.

In [9]:

```
from sympy import *
import sympy as sp
(x, k) = symbols('x k')
```

```
y = sp.Function('y')(x)
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)
```

In [11]:
```
ode = Eq(yp + k*y, 100*k)
ode
```

Out[11]:
$$ky(x) + \frac{d}{dx}y(x) = 100k$$

In [12]:
```
sol = dsolve(ode, y)
sol
```

Out[12]:
$$y(x) = C_1 e^{-kx} + 100$$

In [13]:
```
checkodesol(ode, sol)
```

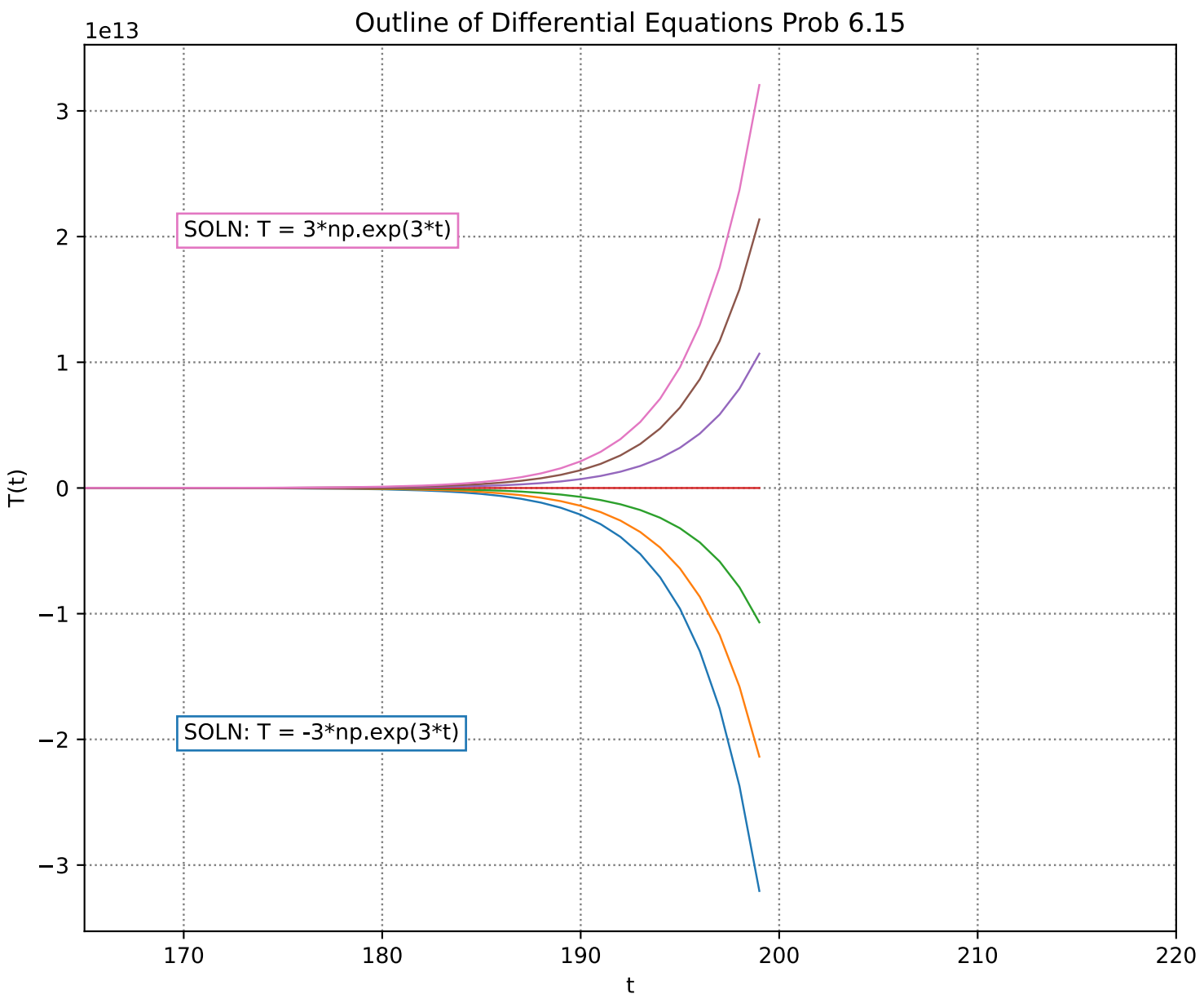
Out[13]:
```
(True, 0)
```

Wolfram Alpha and sympy agree on the solution.

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 $y' + xy = xy^2$.

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 [14]:

```
from sympy import *
import sympy as sp
(x) = symbols('x')
y = sp.Function('y')(x)
yp = sp.Derivative(y, x)
```

```
ydp= sp.Derivative(y, x, x)
```

In [15]:
```
ode = Eq(yp + x*y, x*y**2)
ode
```

Out[15]:
$$xy(x) + \frac{d}{dx}y(x) = xy^2(x)$$

In [16]:
```
sol = dsolve(ode, y)
sol
```

Out[16]:
```
[Eq(y(x), (sqrt(exp(2*C1 + x**2)) - 1)/(exp(2*C1 + x**2) - 1)),
 Eq(y(x), -(sqrt(exp(2*C1 + x**2)) + 1)/(exp(2*C1 + x**2) - 1))]
```

In [17]:
```
checkodesol(ode, sol)
```

Out[17]:
```
[(True, 0), (True, 0)]
```

There is a contrast between how sympy handles the problem and how Wolfram Alpha handles it. In W|A the input is:

!| dy/dx +x y=x y^2 |!

Wolfram Alpha categorizes this as a Bernoulli's equation.

The given answer is:

$$y(x) = \frac{1}{e^{c_1 + x^2/2}} + 1$$

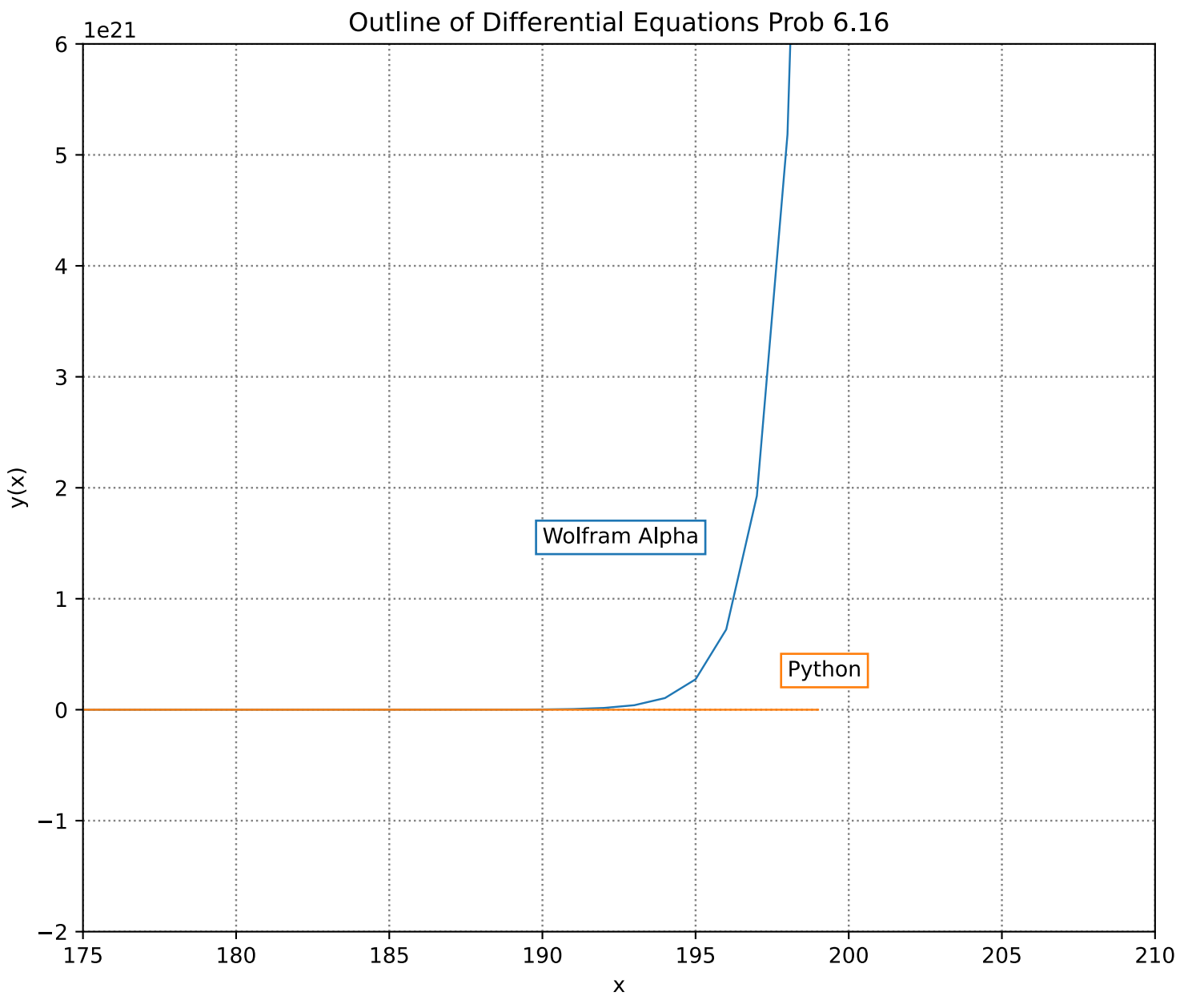
which differs from Python's answer.

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.

The plot coefficients which would show a curve in the Python solution are not presently known.

In [68]:

```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(-10., 10, 200)
x1 = np.linspace(-10., 10., 200)

two = np.array([1])

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

def g(x1):
    for k in two:
        g = (np.sqrt(np.exp(2*k + x1**2)) -1)/(np.exp(2*k + x1**2) - 1)
        plt.plot(g, linewidth = 0.9)

y1=g(x1)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 6.16")
plt.rcParams['figure.figsize'] = [9, 7.5]

ax = plt.gca()

plt.text(198, 3e20, "Python", size=10,bbox=dict\
        (boxstyle="square", ec=('#FF7F0E'),fc=(1., 1., 1),))
plt.text(190, 15e20, "Wolfram Alpha", size=10,bbox=dict\
        (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.xlim(175, 210)
plt.ylim(-.2e22, .6e22)
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'.

The input to Wolfram Alpha is:

!| dy/dx -(3/x) y=x^4 y^(1/3) |!

W|A classifies the equation as 'Bernoulli's equation'. The answer supplied is:

$$\frac{1}{27}(x^2(c_1 + 2x^3))^{3/2}$$

From the plot produced, it appears that by 'principal root' Wolfram Alpha signifies the positive real root.

The try by sympy is next. It adheres to the $\mathbb{R}$ line, but considers answers of both signs.

In [77]:
```
from sympy import *
import sympy as sp
from fractions import Fraction
(x, c1) = symbols('x c1')
y = sp.Function('y')(x)
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)

```

In [81]:
```
ode = Eq(yp - 3*(y/x), x**4*y**(Fraction(1,3)))
ode

```

Out[81]:
$$\frac{d}{dx}y(x) - \frac{3y(x)}{x} = x^4\sqrt[3]{y(x)}$$

In [82]:
```
sol = dsolve(ode, y)
sol

```

Out[82]:
```
[Eq(y(x), -x**3*(C1 + 2*x**3)**(3/2)/27),
 Eq(y(x), x**3*(C1 + 2*x**3)**(3/2)/27)]
```

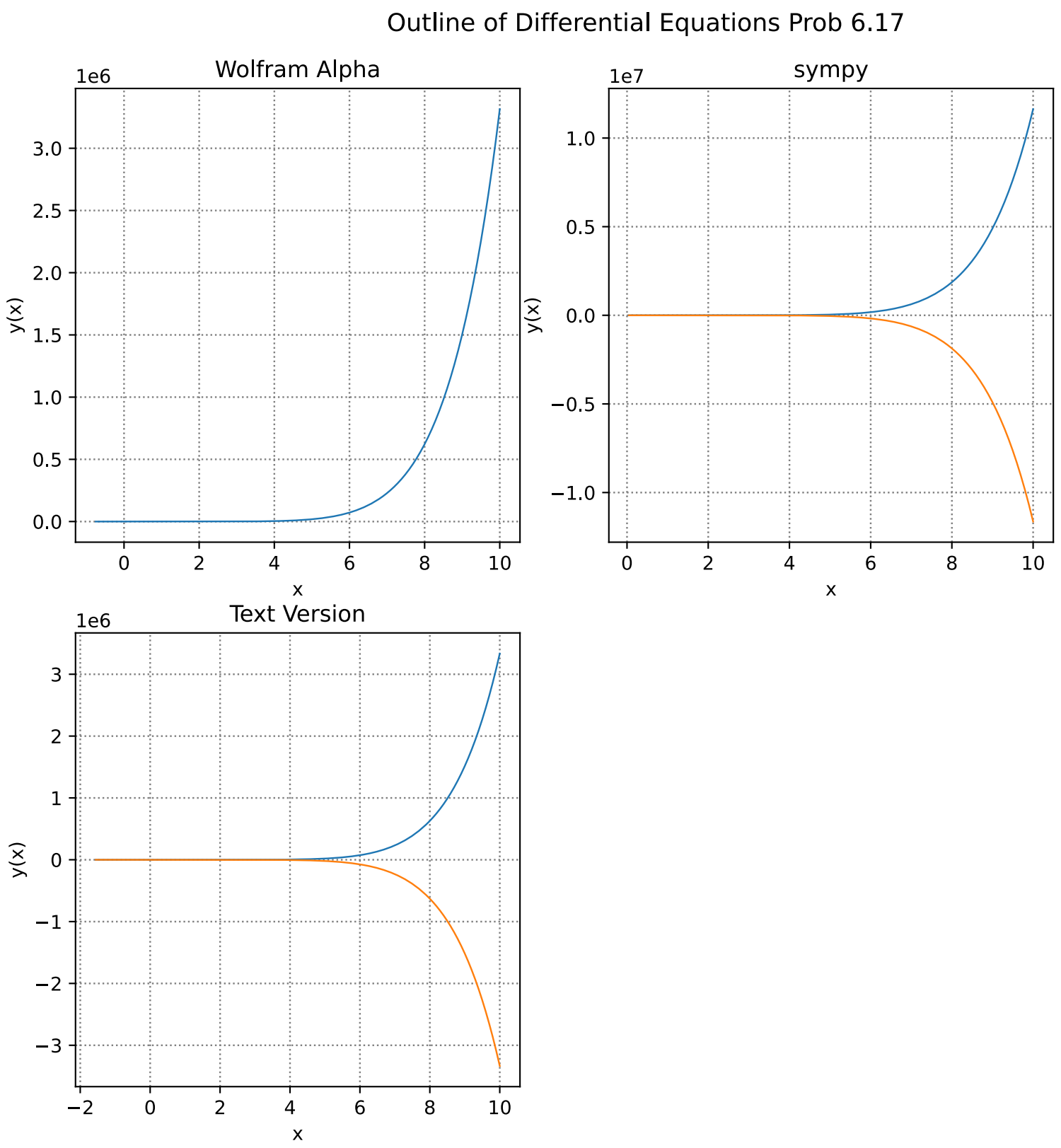
In the case of all plots constructed below, the constant c_1 is assigned a value of 1.

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 [93]:
```
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)
ysn = xs**3*(1 + 2*xs**3**(3/2))/27 #text1
ys = -xs**3*(1 + 2*xs**3**(3/2))/27 #text1

plt.plot(xs,ysn, linewidth = 0.9)
plt.plot(xs,ys, linewidth = 0.9)
plt.title('sympy')
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= 0.1,
                    bottom=0.1,
                    right=0.7,
                    top=1.15,
                    wspace=0.2,
                    hspace=0.2)
plt.show()

```



In [ ]:
```
1
```