Chapter 4: Separable First-Order Differential Equations Techniques for solving separable odes are considered. Covered are several detailed examples showing how conversion to differential form is achieved so that separability can be performed.

4.1. Solve the differential equation $x\,dx - y^2\,dy = 0$.

The direction the text takes is to solve the equations for y. However, it should be kept in mind that there is no specific linkage between y and x.

In [40]:
```python
from sympy import *
x, y, c1 = symbols('x y c1', real=True)
expr1=x
xup2=integrate(expr1,x)
xup2

```

Out[40]: $\frac{x^2}{2}$

In [41]:
```python
expr2=-y*y
yup3=integrate(expr2,y)
yup3

```

Out[41]: $-\frac{y^3}{3}$

In [42]:
```python
eq1 = Eq(xup2 + yup3, c1)
eq1

```

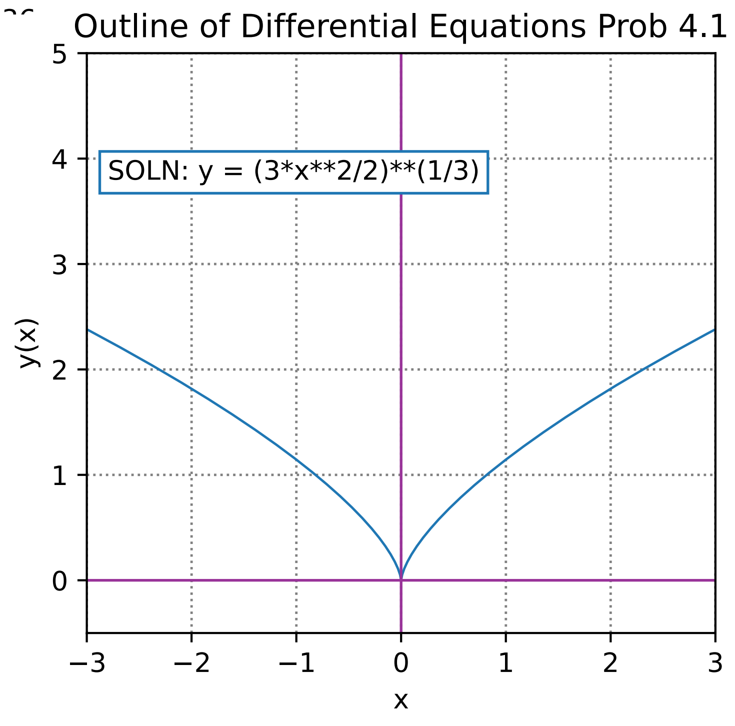
Out[42]: $\frac{x^2}{2} - \frac{y^3}{3} = c_1$

In [43]:
```python
result = solve([eq1],(y))
result

```

Out[43]:
```
[((-3*c1 + 3*x**2/2)**(1/3),),
 (-(-3*c1 + 3*x**2/2)**(1/3)/2 - sqrt(3)*I*(-3*c1 + 3*x**2/2)**(1/3)/2,),
 (-(-3*c1 + 3*x**2/2)**(1/3)/2 + sqrt(3)*I*(-3*c1 + 3*x**2/2)**(1/3)/2,)]
```

Above: The imaginaries are significant in size, and unnecessary in this context. The calculated value for (real) y agrees with that of the text. Below: the next cell plots the outcome of the calculations.

In [9]:
```python
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

%config InlineBackend.figure_formats = ['svg']

x = np.arange(-3., 3., 0.01)
y = (3*x**2/2)**(1/3)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 4.1")
plt.rcParams["figure.figsize"] = [7, 7]

plt.plot(x,y,linewidth = 0.9)

ax = plt.gca()
ratio = 0.4
#ratio is adjusted by eye to get squareness of x and y spacing
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)
ax.axis([-3, 3, -0.5, 5.])
ax.set_xticks([ -3, -2, -1, 0, 1, 2, 3])
ax.set_yticks([0,1,2,3,4,5])
ax.set_xlim(-3,3)
ax.axhline(y=0, color='#993399', linewidth=1)
ax.axvline(x=0, color='#993399', linewidth=1)

plt.text(-2.8, 3.8, "SOLN: y = (3*x**2/2)**(1/3)", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.ylim(-0.5, 5)
plt.show()
```



4.2. Solve the differential equation $y' = y^2 x^3$.

Necessary first to write the problem in differential form.

$$\frac{dy}{dx} = y^2 x^3$$

$$\implies \qquad dy\, y^{-2} = x^3\, dx$$

$$\implies \qquad dy\, y^{-2} - x^3\, dx = 0$$

In [2]:
```
from sympy import *
x, y, c1 = symbols('x y c1')
expr1=-x**3
xup3=integrate(expr1,x)
xup3
```

Out[2]:
$$-\frac{x^4}{4}$$

In [3]:
```
expr2= y**-2
yup2=integrate(expr2,y)
yup2
```

Out[3]:
$$-\frac{1}{y}$$

In [4]:
```
eq1 = Eq(yup2 + xup3, c1)
eq1
```

Out[4]:
$$-\frac{x^4}{4} - \frac{1}{y} = c_1$$

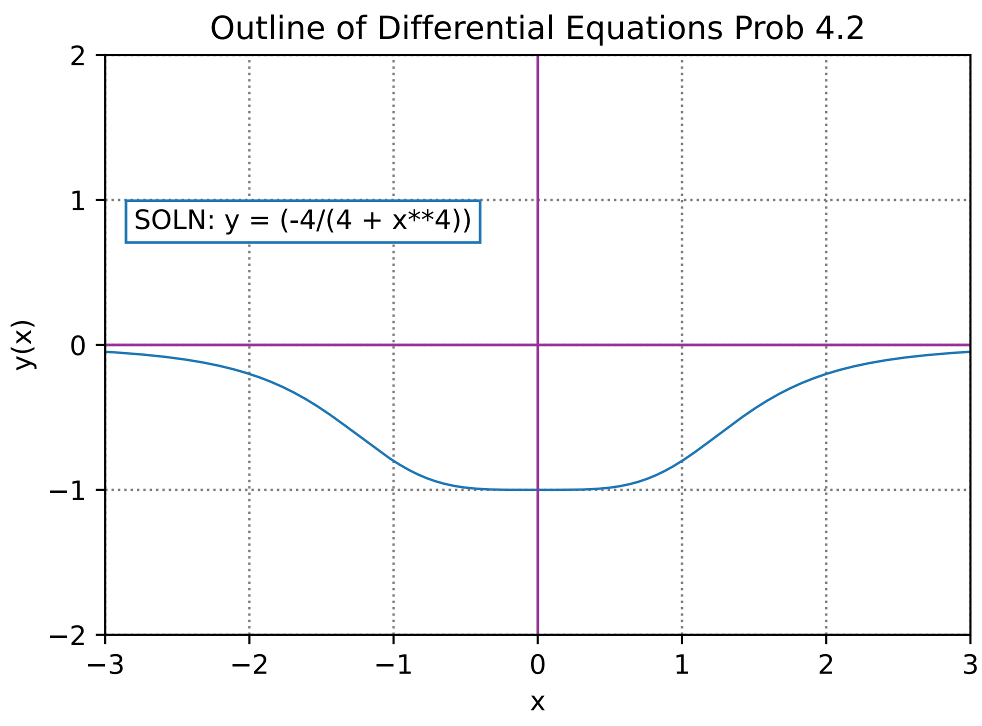
The cell above agrees numerically with the text for the factors of the final equation. However, in calculating y, below, a difference in sign develops. The sign difference disappears with a judicious choice of arbitrary constants.

A plot of the resulting equation is shown below.

In [6]:
```
result = solve([eq1],(y))
result

```

Out[6]:
```
{y: -4/(4*c1 + x**4)}
```

In [220]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

%config InlineBackend.figure_formats = ['svg']

x = np.arange(-5., 3., 0.01)
y = -4/(4 + x**4)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 4.2")
plt.rcParams["figure.figsize"] = [7, 7]

plt.plot(x,y,linewidth = 0.9)

ax = plt.gca()
ratio = 0.125
#ratio is adjusted by eye to get squareness of x and y spacing
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)
ax.axis([-3, 3, -0.5, 5.])
ax.set_xticks([ -3, -2, -1, 0, 1, 2, 3])
ax.set_yticks([ -3, -2, -1, 0,1,2])
ax.set_xlim(-3,3)
ax.axhline(y=0, color='#993399', linewidth=1)
ax.axvline(x=0, color='#993399', linewidth=1)

plt.text(-2.8, 0.8, "SOLN: y = (-4/(4 + x**4))", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.ylim(-2, 2)
plt.show()

```



4.3. Solve

$$\frac{dy}{dx} = \frac{x^2 + 2}{y}$$

Manipulating factors in the original equation, which is already in differential form, brings it to

$$(x^2 + 2)\,dx - y\,dy = 0$$

In [7]:
```
from sympy import *
x, y, c1 = symbols('x y c1')
expr1=x**2 + 2
xup3=integrate(expr1,x)
xup3
```

Out[7]: $\frac{x^3}{3} + 2x$

In [8]:
```
expr2= -y
yup2=integrate(expr2,y)
yup2
```

Out[8]: $-\frac{y^2}{2}$

In [11]:
```
eq1 = Eq( xup3 + yup2, c1)
```

Out[11]: $\frac{x^3}{3} + 2x - \frac{y^2}{2} = c_1$

In [12]:
```
result = solve([eq1],(y))
result
```

Out[12]:
```
[(-sqrt(6)*sqrt(-3*c1 + x**3 + 6*x)/3,), (sqrt(6)*sqrt(-3*c1 + x**3 + 6*x)/3,)]
```

Both of the solutions found by Sympy are real. To represent them in a more conventional notation:

$$-\sqrt{6}\,\frac{\sqrt{-3c_1 + x^3 + 6x}}{3}$$

$$\sqrt{6}\,\frac{\sqrt{-3c_1 + x^3 + 6x}}{3}$$

In order to make the solutions compatible with the text solution, the second can be re-expressed as:

$$\frac{\sqrt{-18c_1 + 6x^3 + 36x}}{\sqrt{9}}$$

$$\Longrightarrow \qquad \sqrt{-2c_1 + \frac{2}{3}x^3 + 4x}$$

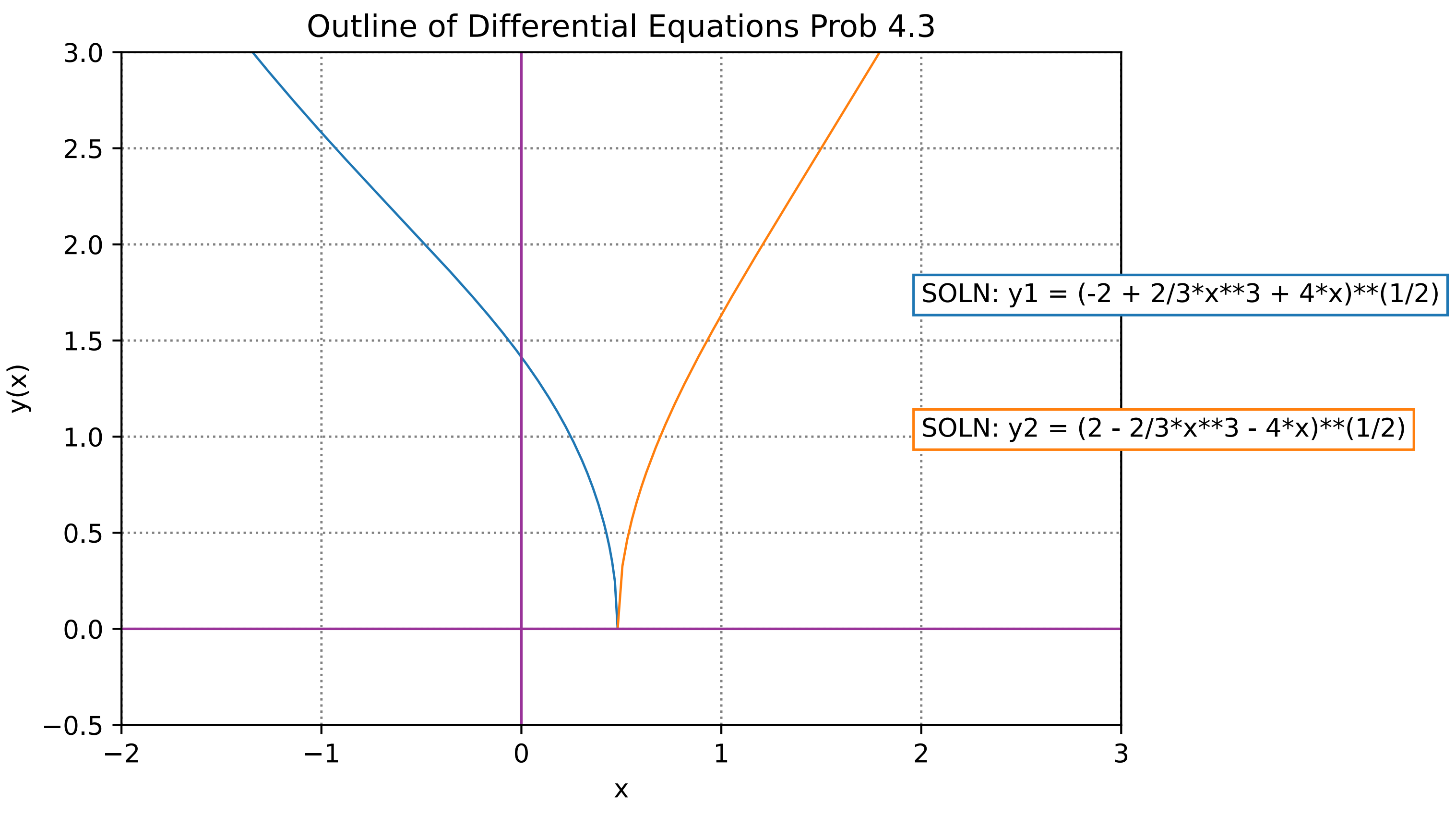
When allowing for the text answer's $k = 2c$, the answers agree.

As for the plotting phase, both solution equations are sensitive to one edge of their domain, and emit complex values when the boundaries are not observed exactly. Any complex value given by either solution causes a runtime error. The snippet below is used to find the boundary limits for the two solution equations.

In [185]:
```
import numpy as np
from sympy import *

x = np.arange(0, 1, 0.02)

def my_function(x):
   # return (2 - 2/3*x**3 - 4*x)**(1/2)
    return (-2 + 2/3*x**3 + 4*x)**(1/2)


print(my_function(.4814056002209))


#.4814056002208 highest acceptable teal
#.4814056002209 lowest acceptable orange

```

```
5.164064005364018e-07
```

In [219]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

%config InlineBackend.figure_formats = ['svg']

x1 = np.setdiff1d(np.linspace(-5, 0.4814056002208, 400),[0]) #to remove the zero
x2 = np.setdiff1d(np.linspace(0.4814056002209, 10, 400),[0]) # to remove the zero
y1 = np.sqrt(2 + -2/3*x1**3 - 4*x1)
y2 = np.sqrt(-2 + 2/3*x2**3 + 4*x2)


plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 4.3")
plt.rcParams["figure.figsize"] = [6, 4]

plt.plot(x1,y1,linewidth = 0.9)
plt.plot(x2,y2,linewidth = 0.9)

ax = plt.gca()
ratio = 1.7
#ratio is adjusted by eye to get squareness of x and y spacing
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)
ax.axis([-3, 3, -0.5, 5.])

ax.set_xticks([ -3, -2, -1, 0, 1, 2, 3])
#ax.set_yticks([ 0,0.125, 0.25,0.375, 0.5, 0.625, 0.75, 1,2])
ax.set_xlim(-2,3)
ax.axhline(y=0, color='#993399', linewidth=1)
ax.axvline(x=0, color='#993399', linewidth=1)

plt.text(2, 1.7, "SOLN: y1 = (-2 + 2/3*x**3 + 4*x)**(1/2)", \
        size=10, bbox=dict(boxstyle="square", ec=('#1F77B4'),\
                           fc=(1., 1., 1),))
plt.text(2, 1, "SOLN: y2 = (2 - 2/3*x**3 - 4*x)**(1/2)", \
        size=10, bbox=dict(boxstyle="square", ec=('#FF7F0E'),\
                           fc=(1., 1., 1),))
plt.ylim(-0.5, 3)
plt.show()

```



4.4. Solve
$$y' = 5y$$

By now it is second nature to re-express the problem in differential form, in this case:

$$\frac{dy}{y} - 5dx = 0$$

```
In [2]: from sympy import *
        x, y, c1 = symbols('x y c1')
        expr1 = 1/y
        yup=integrate(expr1 ,y)
        yup
```

Out[2]: $\log(y)$

```
In [6]: expr2 = -5
        xup = integrate(expr2 ,x)
        xup
```

Out[6]: $-5x$

```
In [7]: eq1 = Eq( yup + xup, c1)
        eq1
```

Out[7]: $-5x + \log(y) = c_1$

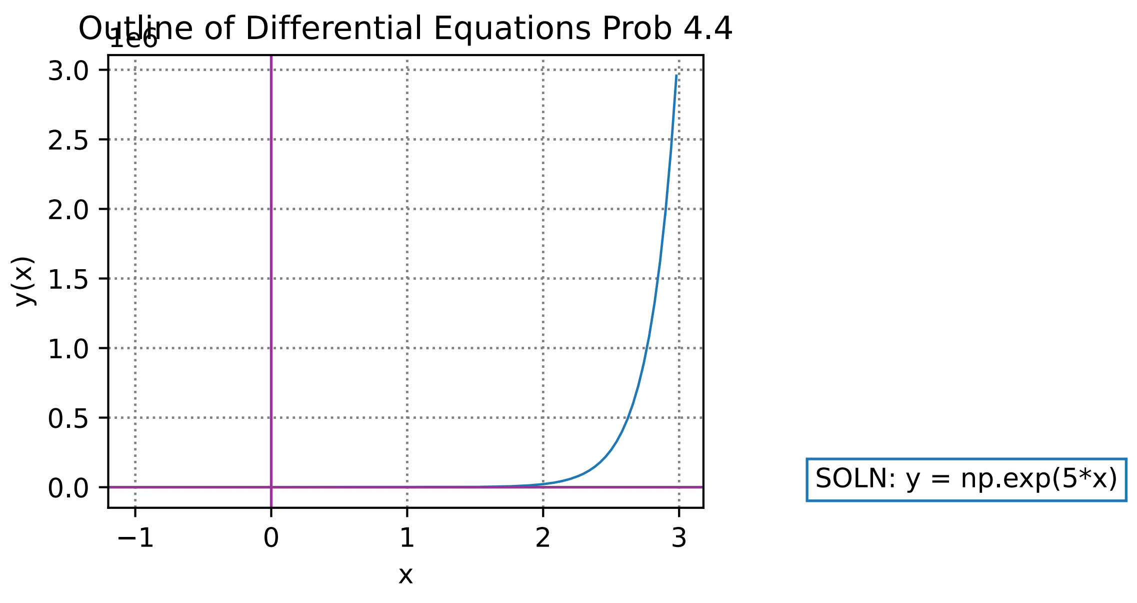
```
In [8]: result = solve([eq1],(y))
        result
```

```
Out[8]: {y: exp(c1)*exp(5*x)}
```

The answer in the above cell agrees with the text.

A plot of the solution is shown below.

In [125]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

%config InlineBackend.figure_formats = ['svg']

x = np.arange(-1., 3., 0.02)
#y = np.arange(-1., 1., 0.005)

y = np.exp(5*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 4.4")
plt.rcParams["figure.figsize"] = [5, 4]

plt.plot(x,y,linewidth = 0.9)

ax = plt.gca()
ratio = 0.76
#ratio is adjusted by eye to get squareness of x and y spacing
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)
#ax.axis([-3, 3, -0.5, 5.])

#ax.set_xticks([ -3, -2, -1, 0, 1, 2, 3])
#ax.set_yticks([ 0,1,2,3,4,5])
#ax.set_xlim(-3,3)
ax.axhline(y=0, color='#993399', linewidth=1)
ax.axvline(x=0, color='#993399', linewidth=1)

plt.text(4, -100, "SOLN: y = np.exp(5*x)", size=10,bbox=dict(boxstyle=\
                    "square", ec=('#1F77B4'),fc=(1., 1., 1),))
#plt.ylim(-0.5, 2)
plt.show()

```



4.5. Solve

$$y' = \frac{x+1}{y^4+1}$$

This time the quest for differential form takes the intrepid problem solver to the form

$$(y^4+1)\,dy - (x+1)\,dx = 0$$

In [9]:
```
from sympy import *
x, y, c1 = symbols('x y c1')
expr1 = y**4 + 1
yup=integrate(expr1, y)
yup

```

Out[9]: $\frac{y^5}{5} + y$

In [19]:
```
expr2 = -(x + 1)
xup = integrate(expr2, x)
xup
```

Out[19]:
$$-\frac{x^2}{2} - x$$

In [20]:
```
eq1 = Eq( yup + xup, c1)
eq1
```

Out[20]:
$$-\frac{x^2}{2} - x + \frac{y^5}{5} + y = c_1$$

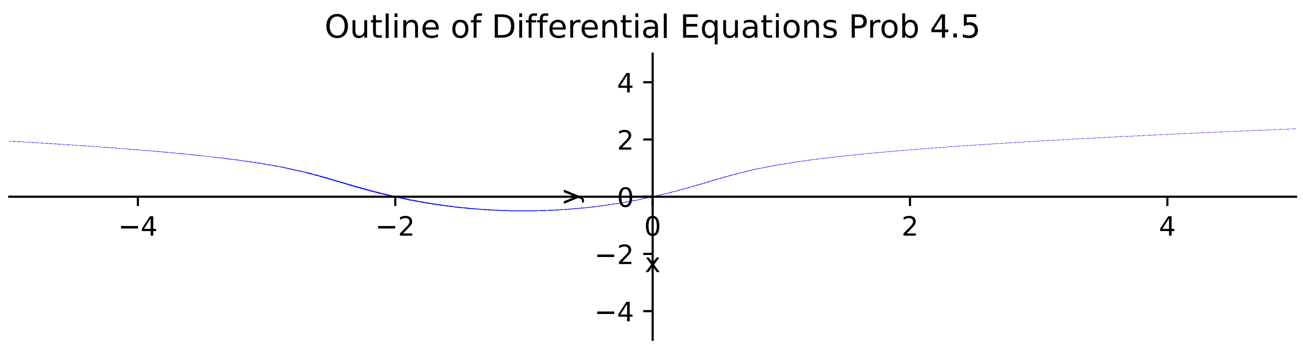
In [21]:
```
result = solve([eq1],(y))
result
```

Out[21]:
```
[]
```

An empty bracket is in some cases an indication of insufficiently adroit handling of Sympy. However, in this case it conveys an inability to reduce the solution to a simpler form, and the 5-factor implicit version of eq1 must suffice.

Below: Plotting from Sympy can't really compete with Matplotlib, but in this case, a plot procedure designed specifically for implicit plots, it can't be beat for simplicity.

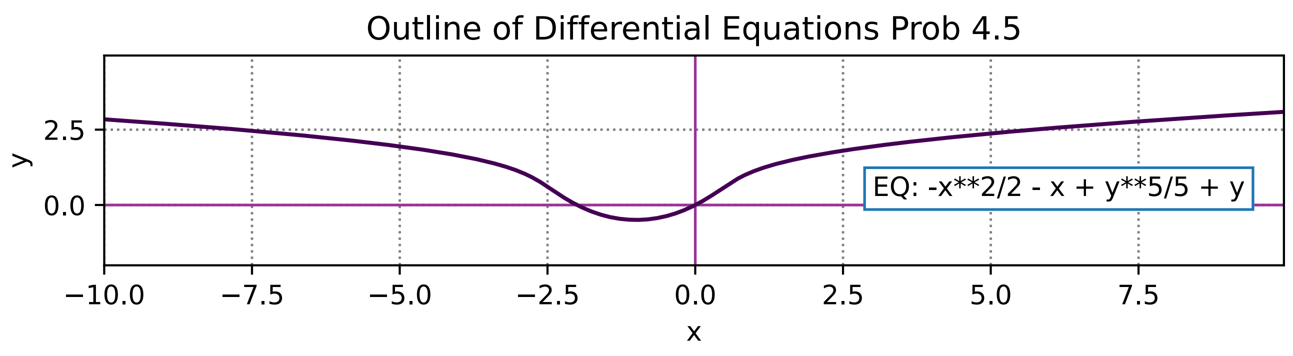
In [111]:
```
from sympy import var, plot_implicit

%config InlineBackend.figure_formats = ['svg']
var('x y')
plot_implicit((-x**2/2 - x + y**5/5 + y), title=\
              'Outline of Differential Equations Prob 4.5')
```



Out[111]:
```
<sympy.plotting.plot.Plot at 0x7f94410363a0>
```

Below: Below: Plotting certain implicit equations with Matplotlib is possible.

In [60]:
```
from matplotlib import pyplot as plt
import numpy as np

%config InlineBackend.figure_formats = ['svg']

delta = 0.04
xrange = np.arange(-10.0, 10.0, delta)
yrange = np.arange(-2.0, 5.0, delta)
x, y = np.meshgrid(xrange, yrange)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Outline of Differential Equations Prob 4.5")
plt.rcParams["figure.figsize"] = [7, 2]
ax = plt.gca()
ratio = 0.51
#ratio is adjusted by eye to get squareness of x and y spacing
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)
ax.axhline(y=0, color='#993399', linewidth=1)
ax.axvline(x=0, color='#993399', linewidth=1)
#ax.set_xticks([0, 100, 200, 300, 400, 500],
             # labels=['0', '2', '4', '6', '8', '10' ])
#ax.set_yticks([-3, -2, -1, 0, 1, 2, 3])
             # labels=['0', '1',  '2', '3',  4,  5,  6 ])
plt.text(3, 0.3, "EQ: -x**2/2 - x + y**5/5 + y", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))

equation = -x**2/2 - x + y**5/5 + y
plt.contour(x, y, equation, [0])
plt.show()
```



4.6. Solve

$$dy = 2t\,(y^2 + 9)\,dt$$

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

t, y, c1 = symbols('t y c1')
expr1 = 1/(y*y + 9)
yup=integrate(expr1, y)
yup

```

Out[27]: $\dfrac{\operatorname{atan}\left(\frac{y}{3}\right)}{3}$

In [28]:
```
expr2 = -2*t
xup = integrate(expr2, t)
xup

```

Out[28]: $-t^2$

In [29]:
```
eq1 = Eq( yup + xup, c1)
eq1

```

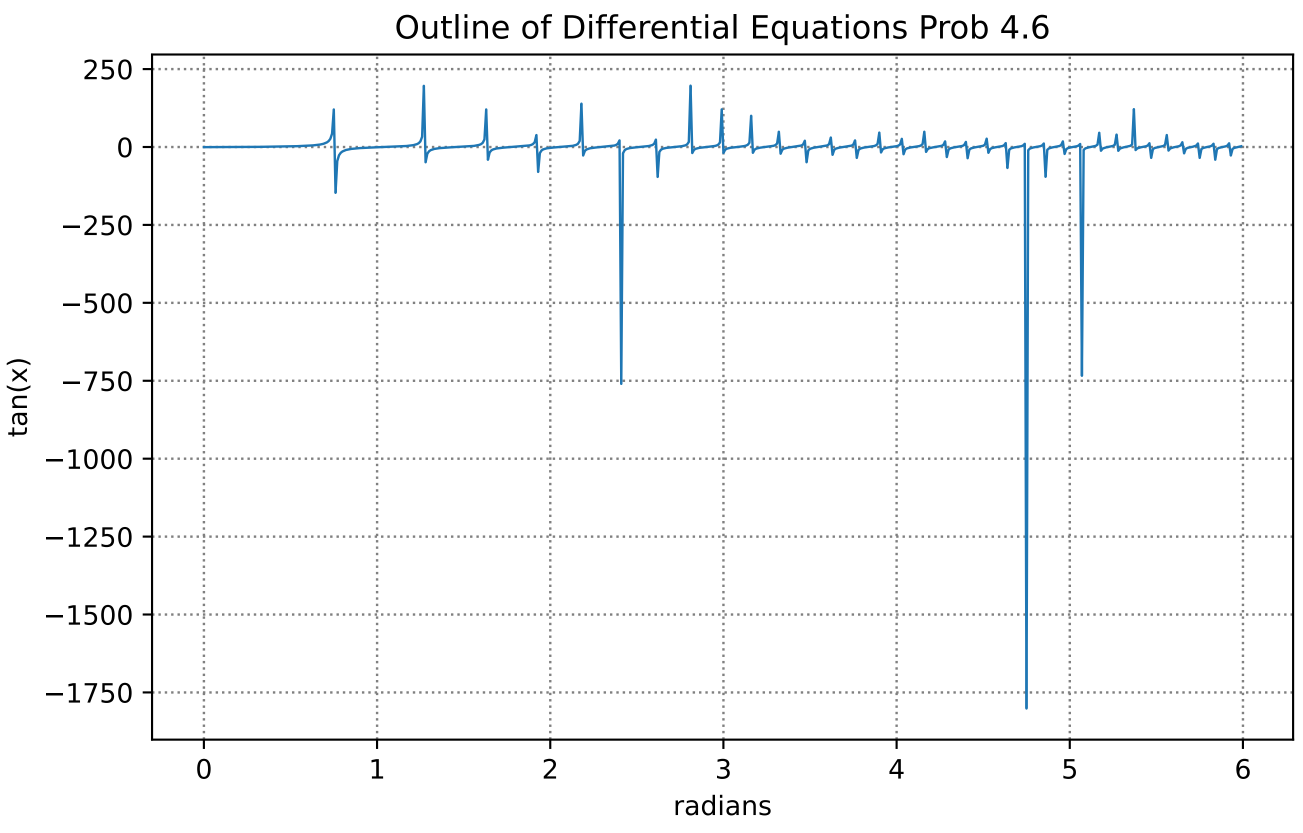
Out[29]: $-t^2 + \dfrac{\operatorname{atan}\left(\frac{y}{3}\right)}{3} = c_1$

In [30]:
```
result = solve([eq1],(y))
result

```

Out[30]:
```
{y: 3*tan(3*c1 + 3*t**2)}
```

The answer in the above cell agrees with the text. Below is a plot.

In [61]:
```python
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
%config InlineBackend.figure_formats = ['svg']

t = np.arange(0.0, 6, 0.01)
s = 3*np.tan(3*(t**2 + 1))
plt.rcParams["figure.figsize"] = (7,4.5)
fig, ax = plt.subplots()
ax.plot(t, s, linewidth = 0.9)
ax.set(xlabel='radians', ylabel='tan(x)',
       title='Outline of Differential Equations Prob 4.6')

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.show()

```



4.7. Solve

$$\frac{dx}{dt} = x^2 - 2x + 2$$

Already in differential form, but to make it usable it should look like:

$$\frac{1}{(x^2 - 2x + 2)}\,dx - dt = 0$$

In [33]:
```python
from sympy import *
t, x, c1 = symbols('t x c1')
expr1 = 1/(x**2 - 2*x + 2)
xup=integrate(expr1, x)
xup

```

Out[33]: $\operatorname{atan}(x - 1)$

In [34]:
```python
expr2 = -1
tup = integrate(expr2, t)
tup

```

Out[34]: $-t$

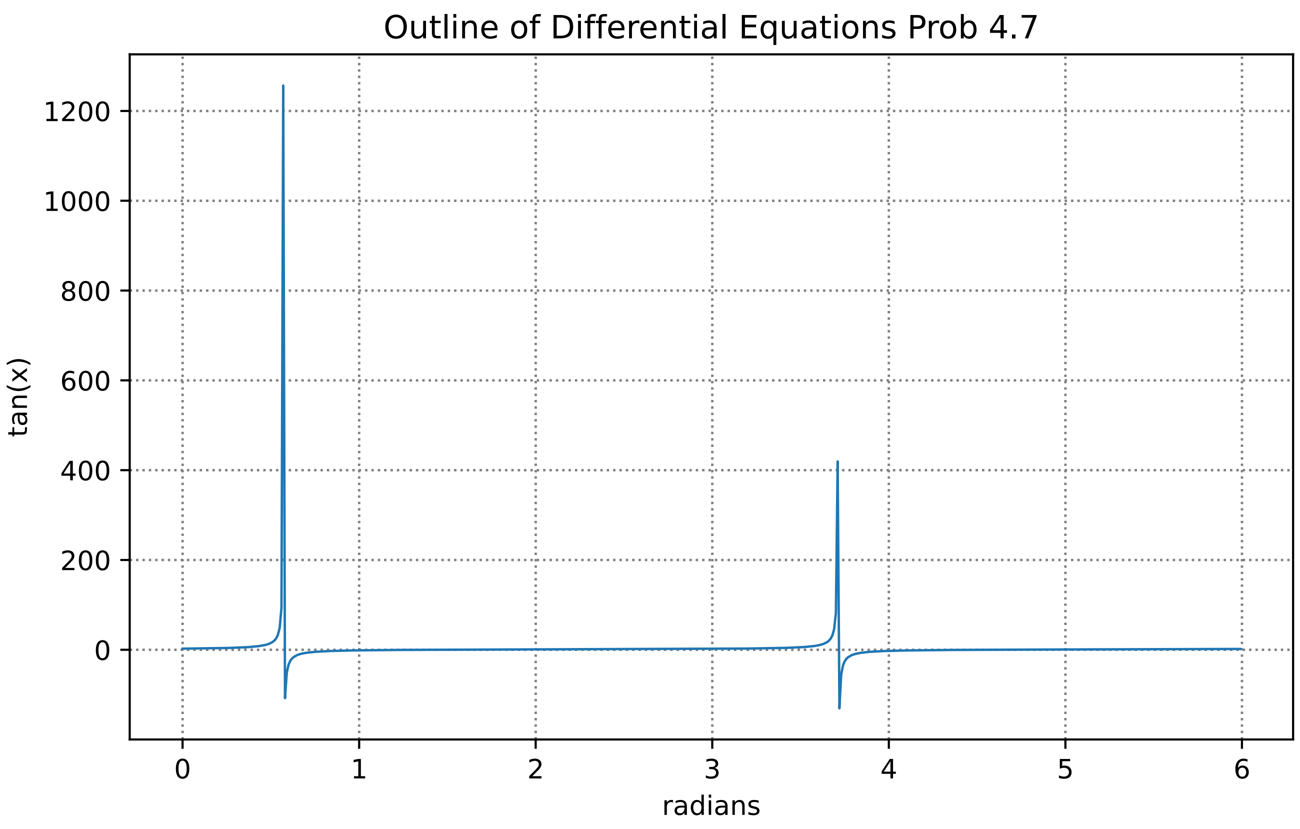
In [35]:
```python
eq1 = Eq( xup + tup, c1)
eq1

```

Out[35]: $-t + \operatorname{atan}(x - 1) = c_1$

In [40]:
```python
solve([eq1],(x))

```

Out[40]:
```
{x: tan(c1 + t) + 1}
```

In [62]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
%config InlineBackend.figure_formats = ['svg']

t = np.arange(0.0, 6, 0.01)
s = np.tan(1 + t) + 1
fig, ax = plt.subplots()
ax.plot(t, s, linewidth = 0.9)
ax.set(xlabel='radians', ylabel='tan(x)',
       title='Outline of Differential Equations Prob 4.7')
plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)

plt.show()

```



4.8. Solve

$$e^x\,dx - y\,dy = 0\,; \qquad y(0) = 1$$

Already in differential form. How to handle the initial condition should be interesting.

In [76]:
```
from sympy import *
from fractions import Fraction
y, x, c1 = symbols('y x c1')
expr1 = exp(x)
xup=integrate(expr1, x)
xup

```

Out[76]: e^x

In [77]:
```
expr2 = -y
yup = integrate(expr2, y)
yup

```

Out[77]: $-\frac{y^2}{2}$

In [78]:
```
eq1 = Eq( xup + yup, c1)
eq1

```

Out[78]: $-\frac{y^2}{2} + e^x = c_1$

In [79]:
```
s2 = eq1.subs([(x, 0), (y, 1)])
s2

```

Out[79]: $\frac{1}{2} = c_1$

In [80]:
```
eq2 = eq1.subs([(c1, Fraction(0.5))])
eq2
```

Out[80]:

$$-\frac{y^2}{2} + e^x = \frac{1}{2}$$

In [81]:

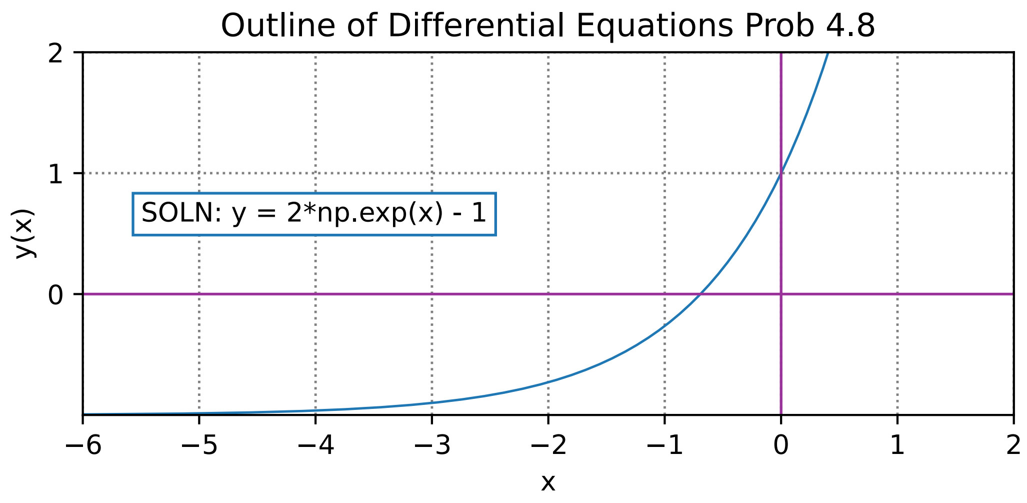
```
solve(eq2, (y**2))
```

Out[81]:

```
[2*exp(x) - 1]
```

In [113]:

```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

%config InlineBackend.figure_formats = ['svg']

x = np.arange(-6., 2., 0.01)
y = 2*np.exp(x) - 1

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 4.8")
plt.rcParams["figure.figsize"] = [7, 2.5]

plt.plot(x,y,linewidth = 0.9)

ax = plt.gca()
ratio = 1.9
#ratio is adjusted by eye to get squareness of x and y spacing
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)
ax.axis([-6, 2, -0.5, 5.])

ax.set_xticks([ -6, -5, -4,-3, -2, -1, 0, 1, 2, 3])
ax.set_yticks([ 0,1,2,3,4,5])

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

plt.text(-5.5, 0.6, "SOLN: y = 2*np.exp(x) - 1", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
ax.set_xlim(-6,2)
plt.ylim(-1, 2)
plt.show()
```



All self-explanatory, and matching the text answer. Note that, in the calculation section, an extra module needs to be imported to use fractional notation.

4.10. Solve

$$x \cos x \, dx + (1 - 6y^5)\, dy = 0; \qquad y(\pi) = 0$$

All is set up and ready in differential form.

In [88]:

```
from sympy import *
from fractions import Fraction
y, x, c1 = symbols('y x c1')
expr1 = x * cos(x)
xup=integrate(expr1, x)
xup
```

Out[88]: $x\hat{\sin}(x) + \cos(x)$

In [89]:
```
expr2 = (1 - 6*y**5)
yup = integrate(expr2, y)
yup
```

Out[89]: $-y^6 + y$

In [90]:
```
eq1 = Eq( xup + yup, c1)
eq1
```

Out[90]: $x\sin(x) - y^6 + y + \cos(x) = c_1$

In [91]:
```
s2 = eq1.subs([(x, 0), (y, 1)])
s2
```

Out[91]: $1 = c_1$

In [92]:
```
eq2 = eq1.subs([(c1, 1)])
eq2
```

Out[92]: $x\sin(x) - y^6 + y + \cos(x) = 1$

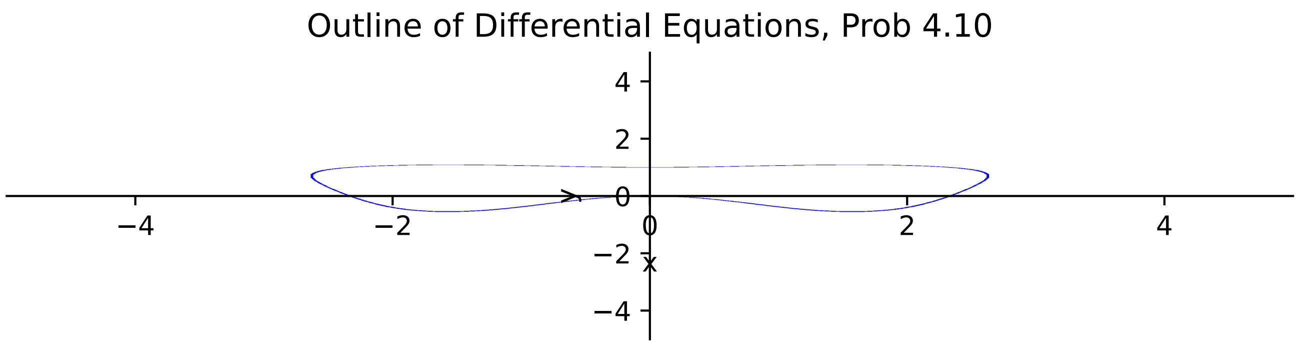
In [93]:
```
solve(eq2, (y))
```

Out[93]:
```
[]
```

Sympy says, and text confirms, that only implicit form -- eq2 -- is available.

Another opportunity to use Sympy's vastly convenient implicit plotter.

In [112]:
```

from sympy.plotting import plot_implicit
%config InlineBackend.figure_formats = ['svg']

var('x y')
plot_implicit((x*sin(x) - y**6 + y + cos(x) -1), points=1200, title\
              ='Outline of Differential Equations, Prob 4.10')

```

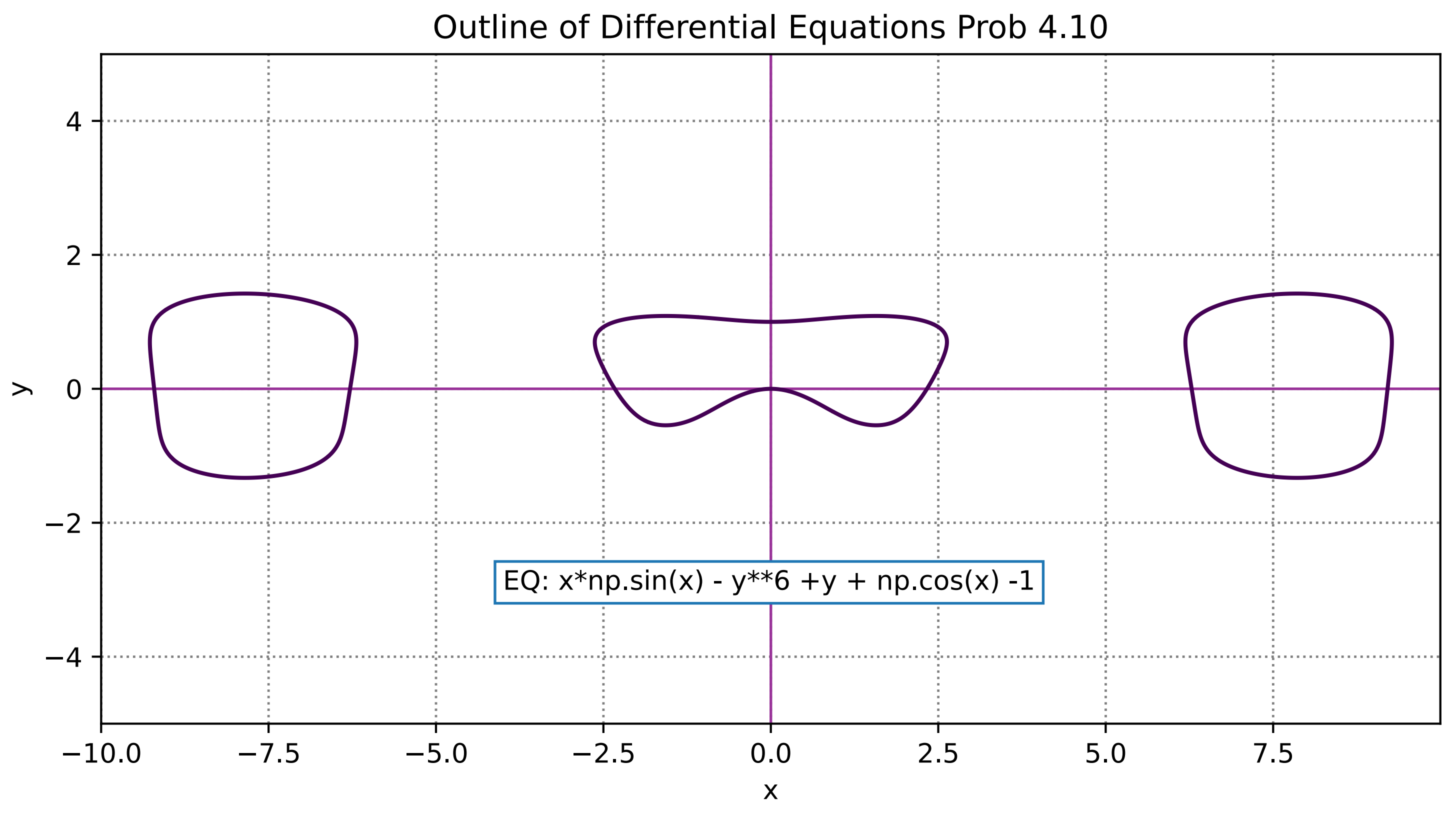


Out[112]:
```
<sympy.plotting.plot.Plot at 0x7f9435f1b940>
```

And another chance to look at Matplotlib's alternative implicit plotting capability.

In [7]:
```
from matplotlib import pyplot as plt
import numpy as np

%config InlineBackend.figure_formats = ['svg']

delta = 1/300
xrange = np.arange(-10.0, 10.0, delta)
yrange = np.arange(-5.0, 5.0, delta)
x, y = np.meshgrid(xrange, yrange)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Outline of Differential Equations Prob 4.10")
plt.rcParams["figure.figsize"] = [9, 9]
ax = plt.gca()
ratio = 1.
#ratio is adjusted by eye to get squareness of x and y spacing
xleft, xright = ax.get_xlim()
```

```
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)
ax.axhline(y=0, color='#993399', linewidth=1)
ax.axvline(x=0, color='#993399', linewidth=1)
#ax.set_xticks([0, 100, 200, 300, 400, 500],
           # labels=['0', '2', '4', '6', '8', '10' ])
#ax.set_yticks([-3, -2, -1, 0, 1, 2, 3])
           # labels=['0', '1',  '2', '3',  4,  5,  6 ])
plt.text(-4., -3, "EQ: x*np.sin(x) - y**6 +y + np.cos(x) -1", size=10,bbox=dict\
        (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))


equation = x*np.sin(x) - y**6 +y + np.cos(x) -1
plt.contour(x, y, equation, [0])
plt.show()
```



4.11. Solve

$$y' = \frac{y + x}{x}$$

Some changes are needed to get to the differential form. To solve this problem, some substitutions from homogeneous attributes need to be applied. That is,

$$y = xv$$

$$\frac{dy}{dx} = v + x\frac{dv}{dx}$$

Applying these substitutions to the given problem. The two equations above will be referred to as the h.equations. The rhs of the lower h.equation is substituted for the y' of the problem equation. Then the rhs of the upper h.equation is substituted for the rhs y in the problem equation, bringing it to:

$$v + x\frac{dv}{dx} = \frac{xv + x}{x} \qquad (1)$$

The left hand v of equation (1) can be brought to the rhs, giving (2).

$$x\frac{dv}{dx} = \frac{xv + x}{x} - v \qquad (2)$$

Multiplying this new v by $\frac{x}{x}$ and combining the rhs, converts the whole rhs to 1 as in (3).

$$x\frac{dv}{dx} = 1 \qquad (3)$$

Now moving the lhs x factor to rhs denominator, then multiplying remaining lhs denominator across, then subtracting remaining lhs factor, then mirroring, completely unpacks (3), giving (4).

$$\frac{1}{x}dx - dv = 0 \qquad (4)$$

The homogeneous-enabled substitutions have transformed the problem equation into one that is separable and ready to work on.

In [12]:
```
from sympy import *
from fractions import Fraction
v, x, c1, c2 = symbols('v x c1 c2')
expr1 = 1/x
xup=integrate(expr1, x)
xup

```

Out[12]: $\log(x)$

In [13]:
```
expr2 = -1
vup = integrate(expr2, v)
vup

```

Out[13]: $-v$

In [14]:
```
eq1 = Eq( xup + vup, c1)
eq1

```

Out[14]: $-v + \log(x) = c_1$

In [15]:
```
solve(eq1, (v))

```

Out[15]:
```
[-c1 + log(x)]
```

There are two differences between this answer and that of the text. First, the text has modified the constant so that $\log(x)$ and the constant can reside in the same log symbol. Second, in the text the argument of the log function is an absolute valued quantity, whereas that wrinkle is not shown in the current problem.

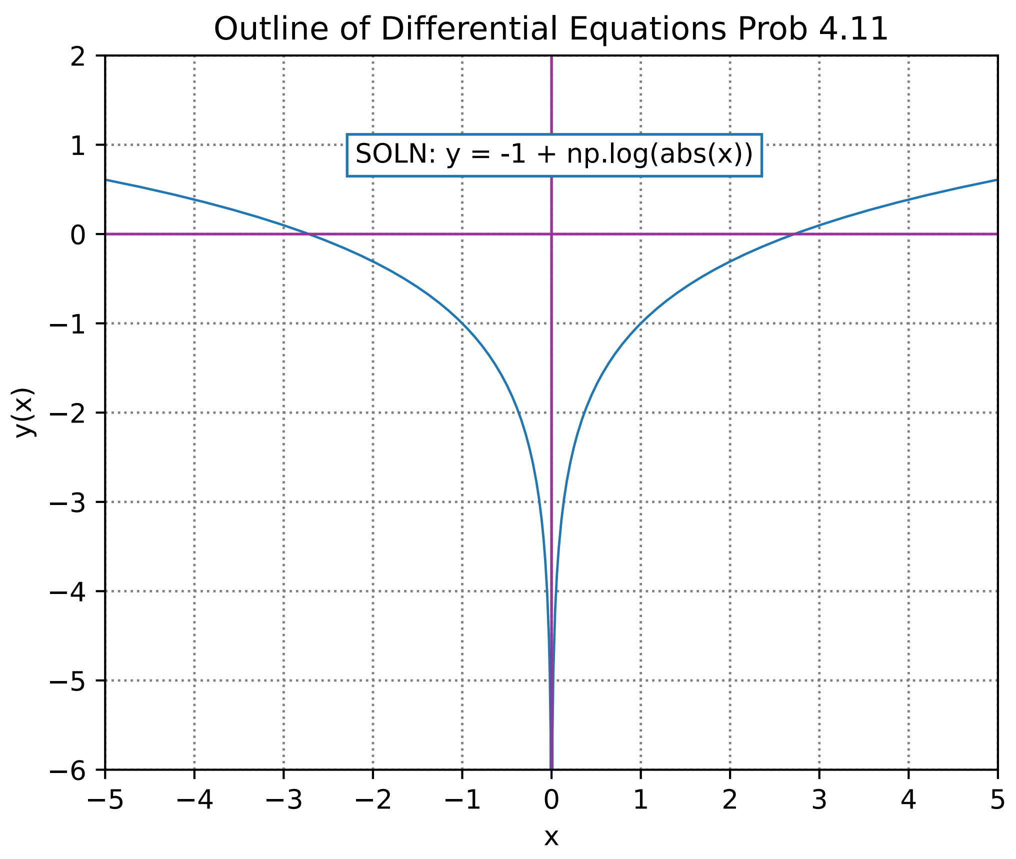
Some research on this suggests there is no current workaround which would make Sympy show the absolute value. So for now it looks like it is just one of those things that need to be remembered. (Looking at the plot shows that use of **abs** covers both wings of the graph.)

In [44]:
```
v = log(abs(x)) + -c1
v

```

Out[44]: $-c_1 + \log(|x|)$

In [129]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

%config InlineBackend.figure_formats = ['svg']

x = np.arange(-6., 5., 0.01)
y = -1 + np.log(abs(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 4.11")
plt.rcParams["figure.figsize"] = [6, 6]

ax = plt.gca()
ratio = 1.
#ratio is adjusted by eye to get squareness of x and y spacing
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)


ax.axis([-5, 5, -0.5, 5.])

ax.set_xticks([ -6, -5, -4,-3, -2, -1, 0, 1, 2, 3,4,5])
ax.set_yticks([ -6, -5, -4,-3, -2, -1,0,1,2])

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

plt.text(-2.2, 0.8, "SOLN: y = -1 + np.log(abs(x))", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
ax.set_xlim(-5,5)
```

```
plt.ylim(-6, 2)
plt.show()

```



4.12. Solve

$$y' = \frac{2y^4 + x^4}{xy^3}$$

Some changes are needed to get to the differential form. As in the last problem, to solve this problem, some substitutions from homogeneous attributes need to be applied. That is,

$$y = xv$$

$$\frac{dy}{dx} = v + x\frac{dv}{dx}$$

Applying these substitutions to the given problem. The two equations above will be referred to as the h.equations. The rhs of the lower h.equation is substituted for the y' of the problem equation. Then the rhs of the upper h.equation is substituted for the rhs y (two places) in the problem equation, bringing it to:

$$v + x\frac{dv}{dx} = \frac{2x^4v^4 + x^4}{x^4v^3} \tag{1}$$

The left hand v of equation (1) can be brought to the rhs, giving, after a rhs x-cancel, (2).

$$x\frac{dv}{dx} = \frac{2v^4 + 1}{v^3} - v \tag{2}$$

Multiplying this new v by $\frac{v^3}{v^3}$ and preparing to convert the rhs as in (3).

$$x\frac{dv}{dx} = \frac{2v^4 + 1}{v^3} - \frac{v^4}{v^3} \tag{3}$$

The subtraction operation finished, the expression is inverted on both sides of the equals sign, and the dv differential transferred to the rhs, resulting in the condition of (4).

$$\frac{1}{x}dx = \frac{v^3}{v^4 + 1}dv \tag{4}$$

Now moving the rhs expression to the lhs gives (5).

$$\frac{1}{x}dx - \frac{v^3}{v^4 + 1}dv = 0 \tag{5}$$

The homogeneous-enabled substitutions have transformed the problem equation into one that is separable and ready to work on.

In [33]:
```
from sympy import *
from fractions import Fraction
v, x, c1, y = symbols('v x c1 y', real=True)
expr1 = 1/x
xup=integrate(expr1, x)
xup
```

Out[33]: $\log(x)$

Trying to adhere to the main arguments without getting too far away from the text.

In [34]:
```
expr2 = -v**3/(v**4 + 1)
vup = integrate(expr2, v)
vup
```

Out[34]:
$$-\frac{\log\left(v^4+1\right)}{4}$$

In [35]:
```
eq1 = Eq( xup + vup, c1)
eq1
```

Out[35]:
$$\log(x) - \frac{\log\left(v^4+1\right)}{4} = c_1$$

In [36]:
```
eq1a = Eq(log(abs(x)) - log(v**4 + 1)/4 , c1)
eq1a
```

Out[36]:
$$-\frac{\log\left(v^4+1\right)}{4} + \log(|x|) = c_1$$

The cell above shows a log-corrected version of an equation similar to the text's.

In [37]:
```
solve(eq1a, (v**4))
```

Out[37]:
```
[x**4*exp(-4*c1) - 1]
```

Above: The expression for v^4 is $x^4 - 1$, modified by a factor descriptive of the constant value. The text's expression for v^4 is exactly the same, though the factor describing the arbitrary constant is different.

In [38]:
```
eq2 = eq1a.subs([(v, y/x)])
eq2
```

Out[38]:
$$-\frac{\log\left(1+\frac{y^4}{x^4}\right)}{4} + \log(|x|) = c_1$$

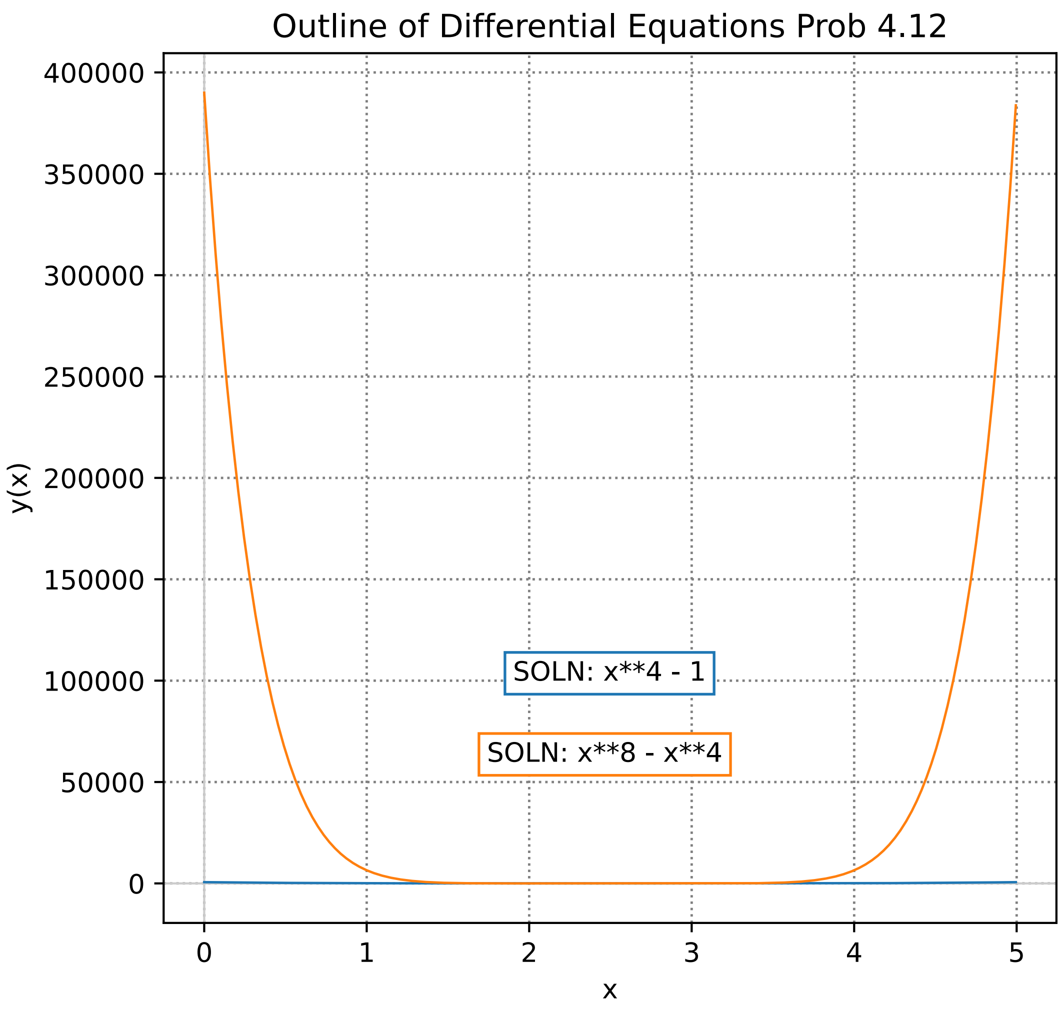
In [39]:
```
solve(eq2, y**4)
```

Out[39]:
```
[x**8*exp(-4*c1) - x**4]
```

Just as with the expressions for v^4 , the expressions for y^4 (representing the condition after the homogeneous substitutions have been reversed) are the same in the worked problem as in the text version. The value of the constant c_1 has been taken as -1.

In [173]:
```
import numpy as np
from sympy import *
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.arange(-5., 5., 0.01)


```

```
plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
ax = plt.gca()
#ratio = .0025
#xleft, xright = ax.get_xlim()
#ybottom, ytop = ax.get_ylim()
#ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)
ax.axhline(y=0, color='0.8', linewidth=0.8)
ax.axvline(x=0, color='0.8', linewidth=0.8)
ax.set_xticks([0,  200,  400,  600,  800, 1000],
             labels=['0','1', '2',  '3', '4', '5'])

plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 4.12")
plt.rcParams["figure.figsize"] = [6, 6]

plt.text(380, 100000, "SOLN: x**4 - 1", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
plt.text(348, 60000, "SOLN: x**8 - x**4", size=10,bbox=dict\
         (boxstyle="square", ec=('#FF7F0E'),fc=(1., 1., 1),))


plt.plot(x**4 - 1, linewidth = 0.9) #eq1a
plt.plot((x**8 - x**4), linewidth = 0.9)

plt.show()

```



4.16. Solve

$$y' = \frac{x^2 + y^2}{xy}; \qquad y(1) = -2$$

In [62]:
```
import sympy as sp
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

```

In [71]:
```
x = symbols("x", real=True)
y = Function("y")(x)
y

```

Out[71]: $y(x)$

In [72]:
```
y.diff()

```

Out[72]: $\frac{d}{dx} y(x)$

In [73]:
```
ode = Eq(y.diff(x), (x**2 + y**2)/(x*y))
ode
```

Out[73]:
$$\frac{d}{dx}y(x) = \frac{x^2 + y^2(x)}{xy(x)}$$

In [74]:
```
sol = dsolve(ode, y)
sol
```

Out[74]:
```
[Eq(y(x), -x*sqrt(C1 + 2*log(x))), Eq(y(x), x*sqrt(C1 + 2*log(x)))]
```

In [76]:
```
ics = {y.subs(x, 1): -2}
ics
```

Out[76]:
```
{y(1): -2}
```

In [77]:
```
ivp = dsolve(ode, ics=ics)
ivp
```

Out[77]:
$$y(x) = -x\sqrt{2\log(x) + 4}$$

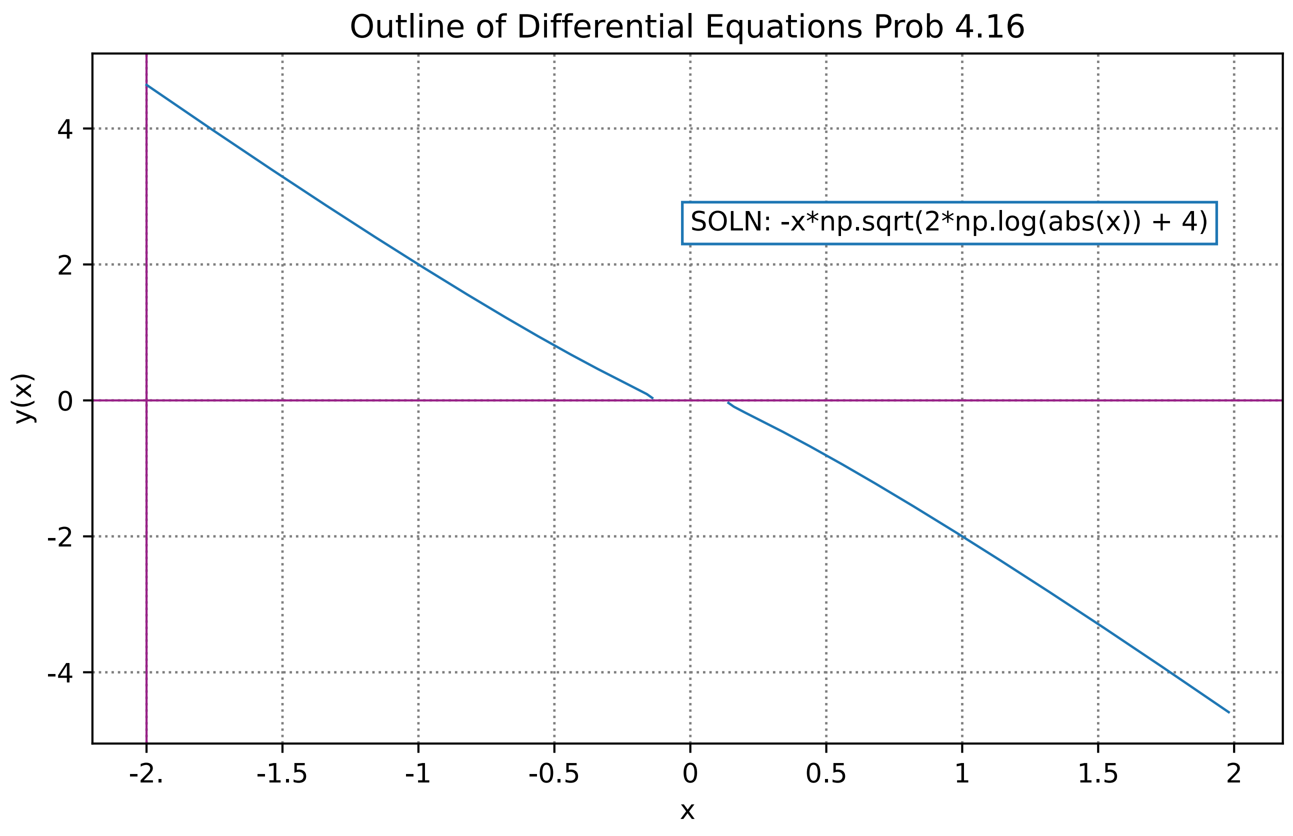
In [78]:
```
checkodesol(ode, ivp)
```

Out[78]:
```
(True, 0)
```

In [8]:

```python
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

%config InlineBackend.figure_formats = ['svg']

x = np.arange(-2., 2., 0.02)

plt.rcParams["figure.figsize"] = (8,5)
ax = plt.gca()
ax.axhline(y=0, color='#992288', linewidth=0.8)
ax.axvline(x=0, color='#992288', linewidth=0.8)
ax.set_xticks([0, 25, 50, 75, 100, 125, 150, 175, 200],
              labels=['-2.', '-1.5', '-1','-0.5','0','0.5','1','1.5','2'])
ax.set_yticks([ -4, -2, 0, 2, 4], \
              labels=[ -4, -2, 0, 2, 4])
ratio = 0.5
#ratio is adjusted by eye to get squareness of x and y spacing
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")

#ax.plot(x, y, label='SOLN: -x*np.sqrt(2*np.log(abs(x)) + 4)')
#ax.legend(loc='upper left', bbox_to_anchor=(0.55, 0.95), \
         # shadow=False, ncol=1)

ax.set_title('Outline of Differential Equations Prob 4.16')
plt.text(100, 2.5, "SOLN: -x*np.sqrt(2*np.log(abs(x)) + 4)", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))

plt.plot( -x*np.sqrt(2*np.log(abs(x)) + 4), linewidth = 0.9)
plt.show()

```

```
/tmp/ipykernel_2987/3789460332.py:36: RuntimeWarning: invalid value encountered in sqrt
  plt.plot( -x*np.sqrt(2*np.log(abs(x)) + 4), linewidth = 0.9)
```



The solution to the current problem runs into a runtime error condition. There are further considerations about this including an alternate plotting method, described below.

The function below shows the boundary values bracketing the x values which cause the runtime error above.

In [24]:

```python
import numpy as np
from sympy import *

x = np.arange(-2., 2., 0.02)

def my_function(x):
  return -x*np.sqrt(2*np.log(abs(x)) + 4)

#.-1353352832367
#.1353352832367
```

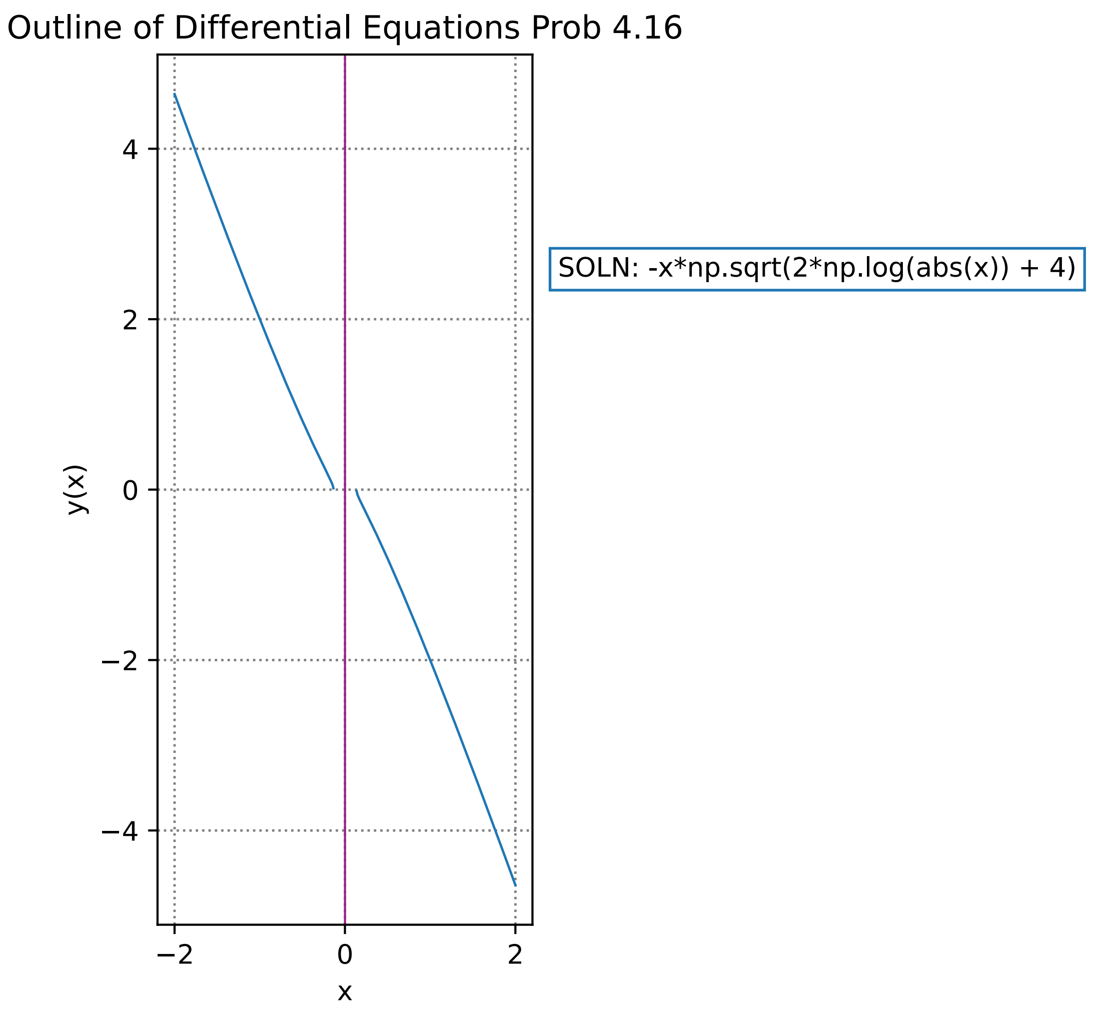
```
print(my_function(1))

```

```
-2.0
```

If it is desired to avoid the runtime error, one way is to use a piecewise function, as shown in the cell below.

In [3]:

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

%config InlineBackend.figure_formats = ['svg']

plt.rcParams["figure.figsize"] = (3, 6)
ax = plt.gca()
#ax.axhline(y=0, color='#992288', linewidth=0.8)
ax.axvline(x=0, color='#992288', linewidth=0.8)
#ax.set_xticks([0, 25, 50, 75, 100, 125, 150, 175, 200],
            #  labels=['-2.', '-1.5', '-1','-0.5','0','0.5','1','1.5','2'])
#ax.set_yticks([ -4, -2, 0, 2, 4], \
            #  labels=[ -4, -2, 0, 2, 4])
ratio = 1
#ratio is adjusted by eye to get squareness of x and y spacing
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")

def fun (x):
    if -2 <= x <= -.1353352832367:
        return -x*np.sqrt(2*np.log(abs(x)) + 4)
    elif .1353352832367 <= x <= 2:
        return -x*np.sqrt(2*np.log(abs(x)) + 4)
    #return 0.0

vfun = np.vectorize(fun)

x = np.linspace(-2, 2, 1600)
y = vfun(x)

ax.set_title('Outline of Differential Equations Prob 4.16')
plt.text(2.5, 2.5, "SOLN: -x*np.sqrt(2*np.log(abs(x)) + 4)", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))


plt.plot(x, y, '-', linewidth = 0.9)
plt.show()

```



Above: The x-axis has been commented out so that the open gap between the branches is visible.

In [11]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
import math
from fractions import Fraction
x = symbols('x', real=True)
y = Function('y')(x)

```

In [12]:
```
#y = -x*np.sqrt(2*np.log(x) + 4)

y = -x*sqrt(2*log(x) + 4)
y

```

Out[12]: $-x\sqrt{2\log(x)+4}$

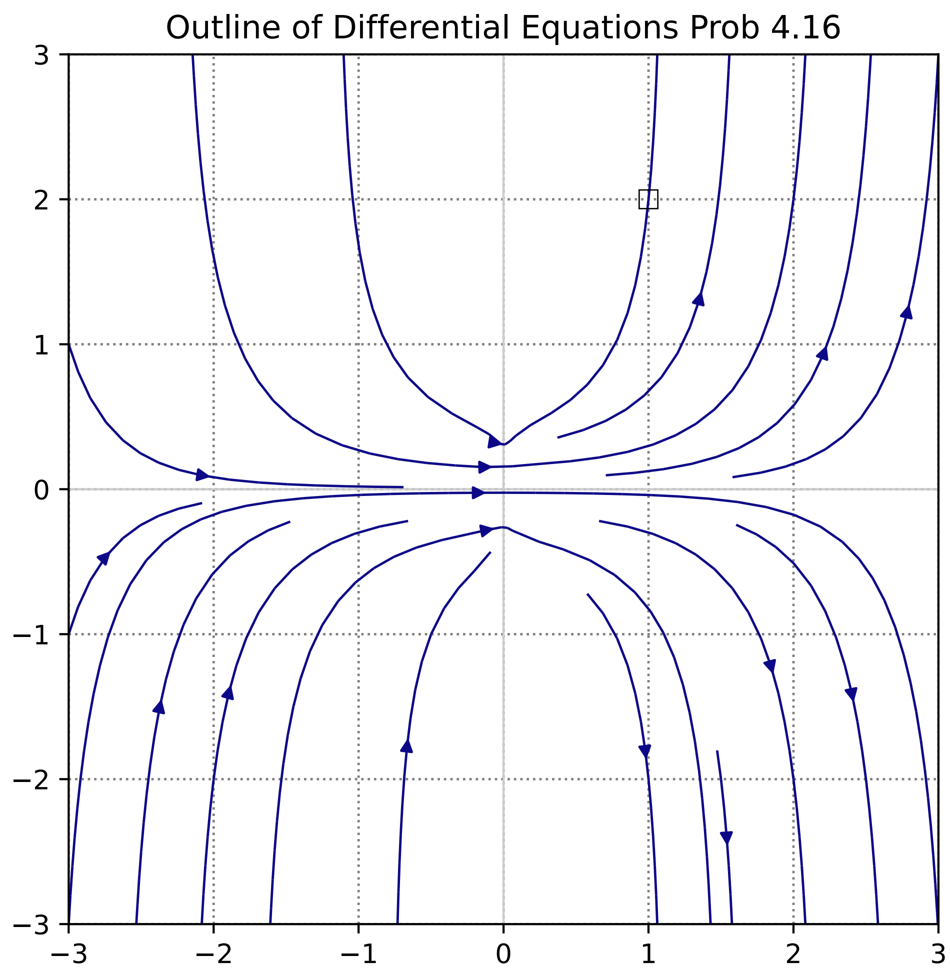
In [15]:
```
dif = diff(y, x)
dif

```

Out[15]: $-\sqrt{2\log(x)+4} - \frac{1}{\sqrt{2\log(x)+4}}$

Above. Trying to demonstrate the equality of the Sympy solution seems difficult. Shown is what should be the analogue of the original problem equation. However, with no y factors, it seems doubtful that it can really mount a match.

In [49]:

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

%config InlineBackend.figure_formats = ['svg']

# Creating dataset
w = 3
Y, X = np.mgrid[-w:w:100j, -w:w:100j]
U = np.ones_like(X) #dxdt = 1
V = (X**2 + Y**2) / X*Y
speed = np.sqrt(U**2 + V**2)
plt.rcParams["figure.figsize"] = (6,6)

# Controlling the starting points of the streamlines
seek_points = np.array([[-3,-2.5,-2,-1.45,1,1.4,2,2.5,3,1,-3,-3,\
                         -3,2.5,1.5,2,-3,-0.5],
                        [-3,-2.5,-2,-1.5,2,1.5,2,2.5,3,-2,-1,-.25\
                         ,1,-2,-2,-2,0.3,-1]])

fig, ax = plt.subplots()
ax.grid(True, which='both', linestyle='dotted')
ax.axhline(y=0, color='0.8', linewidth=0.8)
ax.axvline(x=0, color='0.8', linewidth=0.8)

ax.set_aspect('equal')
strm = ax.streamplot(X, Y, U, V, color = U,
                     linewidth = 0.9,
                     cmap ='plasma',
                     start_points = seek_points.T)

ax.set_title('Outline of Differential Equations Prob 4.16')

xpts = np.array([1])
ypts = np.array([2])
plt.plot(xpts, ypts, markersize=7, color='k', marker='s', mfc='none', \
         markeredgewidth=0.5)

plt.show()

```



Above. The streamplot can give a picture of a family of curves which may include the original differential equation. The curve with the framed point may indicate the original trace.

In [ ]: