In [1]:
```
%autosave 0
```

```
Autosave disabled
```

(Custom CSS files are not reliable for controlling Jupyter font style. To establish the same appearance as the original notebook, depend on the browser to control the font, by setting the desired font faces in the browser settings. For example, Chrome 135 or Firefox 134 can do this. In this notebook series, Bookerly font is for markdown and Monaco is for code.)

Chapter 28: Series Solutions Near a Regular Singular Point. Finding and expressing differential equation solutions in terms of power series.

Regular Singular Point

Consider a second-order ordinary differential equation

$$y'' + P(x)\,y' + Q(x)\,y = 0$$

If $P(x)$ and $Q(x)$ remain finite at $x = x_0$, then x_0 is called an *ordinary point*. If either $P(x)$ or $Q(x)$ diverges as $x \to x_0$ but $(x - x_0)P(x)$ and $(x - x_0)Q(x)$ remain finite as $x \to x_0$, then $x = x_0$ is called a *regular singular point*.

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

28.1 Determine whether $x = 0$ is a regular singular point of the differential equation $y'' - xy' + 2y = 0$.

This problem can be submitted to Wolfram Alpha. The entry line is:

!| y'' - x y' + 2 y = 0 |!

and the returned solutions is:

$$y(x) = c_2 \left(\sqrt{2\,\pi}\,(x^2 - 1)\,\text{erfi}\left(\frac{x}{\sqrt{2}} \right) - 2e^{x^2/2}\,x \right) + c_1\,(x^2 - 1)$$

Now, whether $x = 0$ is a regular singular point. The problem is second order, as stipulated. $P(x) = -x$ and $Q(x) = 2$. Neither $P(x)$ nor $Q(x)$ diverges as $x \to 0$, therefore $x = 0$ is not a regular singular point of the given equation.

28.2 Determine whether $x = 0$ is a regular singular point of the differential equation

$$2x^2y'' + 7x(x + 1)y' - 3y = 0.$$

This problem can be submitted to Wolfram Alpha. The entry line is:

!| 2 x^2 y'' + 7 x (x + 1) y' - 3 y = 0 |!

and the returned solutions is:

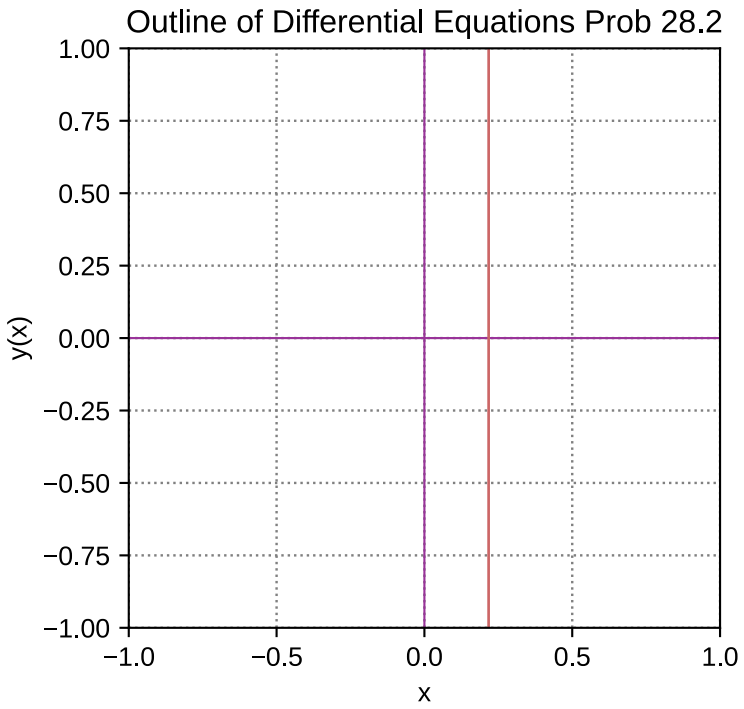
$$y(x) = \frac{1}{x^3} c_2 e^{-(7x)/2} \left(\sqrt{14\,\pi}\, e^{(7x)/2} \left(343 x^3 - 147 x^2 + 63 x - 15\right) \operatorname{erf}\left(\sqrt{\frac{7}{2}}\sqrt{x}\right) + 98 (7x - 4) x^{3/2} + 210 \sqrt{x} \right) + \frac{c_1 (7x (7x (7x - 3) + 9) - 15)}{x^3}$$

Now, whether $x = 0$ is a regular singular point. The problem is second order, as stipulated. After the equation has been modified to match the template, it looks like

$$y'' + \frac{7}{2x}(x + 1)y' - \frac{3}{2x^2}y = 0$$

Now $P(x) = \frac{7}{2x}(x + 1)$ and $Q(x) = -\frac{3}{2x^2}$. Both $P(x)$ and $Q(x)$ diverge as $x \to 0$. The limit of $(x - x_0)$ multiplied against $P(x)$ will keep the function finite, but it will not work on $Q(x)$. Therefore the point $x = 0$ is not a regular singular point of the given equation.

In [30]:
```
import numpy as np
from scipy import special
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.setdiff1d(np.linspace(0.,10.,300),[0]) #to remove the zero
y1 = (1/x**3)*np.exp(-(7*x)/2)*(np.sqrt(14*np.pi)*np.exp((7*x)/2)*\
      (343*x**3 - 147*x**2 + 63*x - 15)*special.erf\
      (np.sqrt(7/2)*np.sqrt(x)) + 98*(7*x - 4)*x**(3/2) +
      210*np.sqrt(x)) + ((7*x*(7*x)*(7*x - 3) + 9) - 15)/x**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 28.2")
plt.rcParams['figure.figsize'] = [4, 4]


ax = plt.gca()
ax.axhline(y=0, color='#993399', linewidth=0.8)
ax.axvline(x=0, color='#993399', linewidth=0.8)
ratio = 0.98
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)


#plt.text(965, -2.5e43, "-np.log(abs((np.cos(x/2))**2 - (np.sin(x/2))**2)))
#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.ylim(-1,1)
plt.xlim(-1, 1)
plt.plot(x, y1, linewidth = 1, color = '#CC6666')
plt.show()
```



Above is shown the exceedingly boring plot of the problem equation solution, as found by Wolfram Alpha, and it appears to have little if any reference to the $x = 0$ area.

28.3 Determine whether $x = 0$ is a regular singular point of the differential equation

$$x^3 y'' + 2x^2 y' + y = 0 .$$

With this equation Wolfram Alpha is able to generate a solution. The entry line is:

!| x^3 y'' + 2 x^2 y' + y = 0 |!

and the result is:

$$c_1 \sqrt{\frac{1}{x}}\, J_1\left(2\sqrt{\frac{1}{x}}\right) + 2\,i\,c_2 \sqrt{\frac{1}{x}}\, Y_1\left(2\sqrt{\frac{1}{x}}\right)$$

Wolfram Alpha explains that $J_n(z)$ is the Bessel function of the first kind, and $Y_n(x)$ is the Bessel function of the second kind. Also lurking in the expression above is the value i .

Now, whether $x = 0$ is a regular singular point. The problem is second order, as stipulated. After the equation has been modified to match the template, it looks like

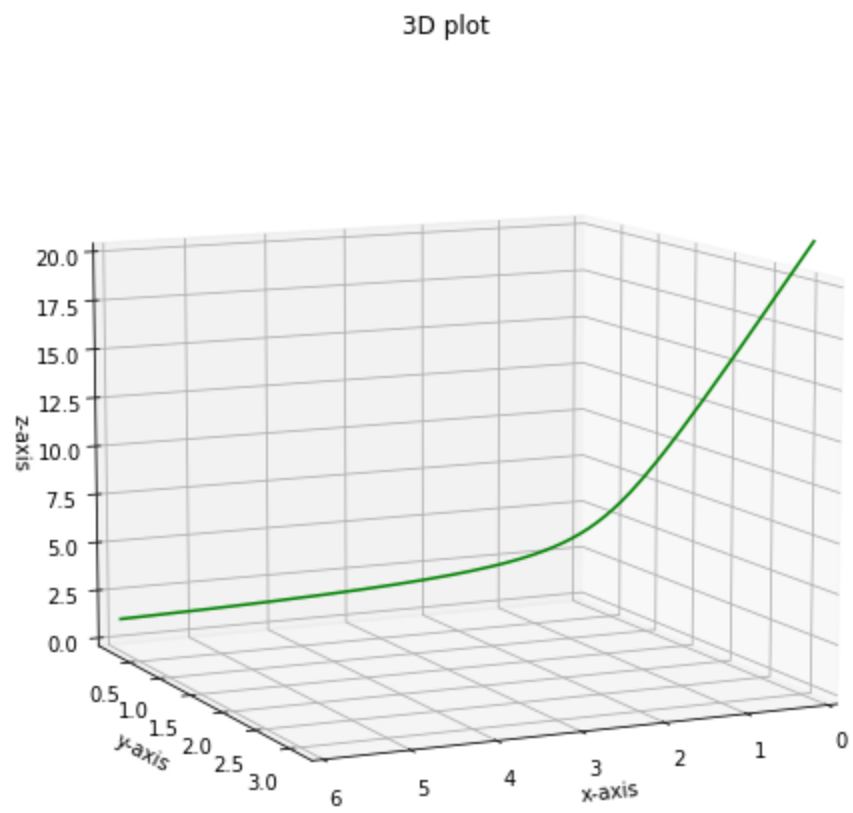
$$y'' + \frac{2}{x}y' + \frac{1}{x^3}y = 0$$

Now $P(x) = \frac{2}{x}$ and $Q(x) = \frac{1}{x^3}$. Both $P(x)$ and $Q(x)$ diverge as $x \to 0$. The limit of $(x - x_0)$ multiplied against $P(x)$ will keep the function finite, but it will not work on $Q(x)$. Therefore the point $x = 0$ is not a regular singular point of the given equation.

Looking at the equation for the solution, from Wolfram Alpha, it is seen that not only are both instances of the Bessels of first kind, they are also of first order, because there is nothing higher than first order in the expression which they inhabit. The next objective will be to create a 3D plot, so that the imaginary part of the equation can be displayed on a separate axis.

Note about 3D plotting in Python. The matplotlib module has a number of methods for making 3D plots, some of which include interactivity. However, they are supported by back ends, such as those based on QT, GTK, and Cairo; but choosing the right back end and then getting all parts to function smoothly without detrimental effects on other sections of the Jupyter file or on the platform ecosystem can be hit or miss. A relatively low level 3D experience is included in this chapter, in which view perspective can be controlled, but not through mouse gestures.

In [10]:
```
from mpl_toolkits import mplot3d
import sympy as sp
import numpy as np
from scipy import special
import matplotlib.pyplot as plt

x, y, z = sp.symbols('x y z')

fig = plt.figure(figsize = (8,8))
ax = plt.axes(projection = '3d')
ax.set_title("3D plot",pad=6)
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_xlim(0,6)
ax.set_zlim(0,20)

x = np.setdiff1d(np.linspace(0.,6.,300),[0]) #to remove the zero
y = np.sqrt(1/x)*(special.j1(1,2*np.sqrt(1/x)))
#y = x**2
z =abs( 2*2*np.sqrt(1/x)*special.y1(1,2*np.sqrt(1/x)))
ax.plot3D(x, y, z, 'green')
plt.xticks([0, 1, 2, 3, 4, 5, 6],
           ["0","1", "2", "3", "4","5", "6"])

#view x-z plane
#ax.view_init(0, 90)

#view y-x plane
#ax.view_init(90, 0)

#view y-z plane
#ax.view_init(0, 0)

#view from (50%-x, 100%-z) to (50%-x, 0%-z)
#ax.view_init(45, 90)

#view from (1, 1, 1) to (0, 0, 0)
#ax.view_init(45, 45)

#view x-y plane
#ax.view_init(45, 90)

ax.view_init(10, 65)

plt.show()
```



Above is shown the 3D plot of the Wolfram Alpha solution equation, with y-axis dedicated to the real half of the equation, and the z-axis dedicated to the complex half of the equation. Changing the view_init lines changes the perspective from which the model is viewed.

28.4 Determine whether $x = 0$ is a regular singular point of the differential equation

$$8x^2y'' + 10xy' + (x - 1)y = 0.$$

In [11]:
```
from sympy import *
import numpy as np

x = symbols("x")
y = Function("y")(x)
y
```

Out[11]: $y(x)$

In [12]:
```
y.diff()
```

Out[12]: $\frac{d}{dx}y(x)$

In [13]:
```
ode = Eq(8*x**2*y.diff(x,x) + 10*x*y.diff(x) + (x - 1)*y , 0)
ode
```

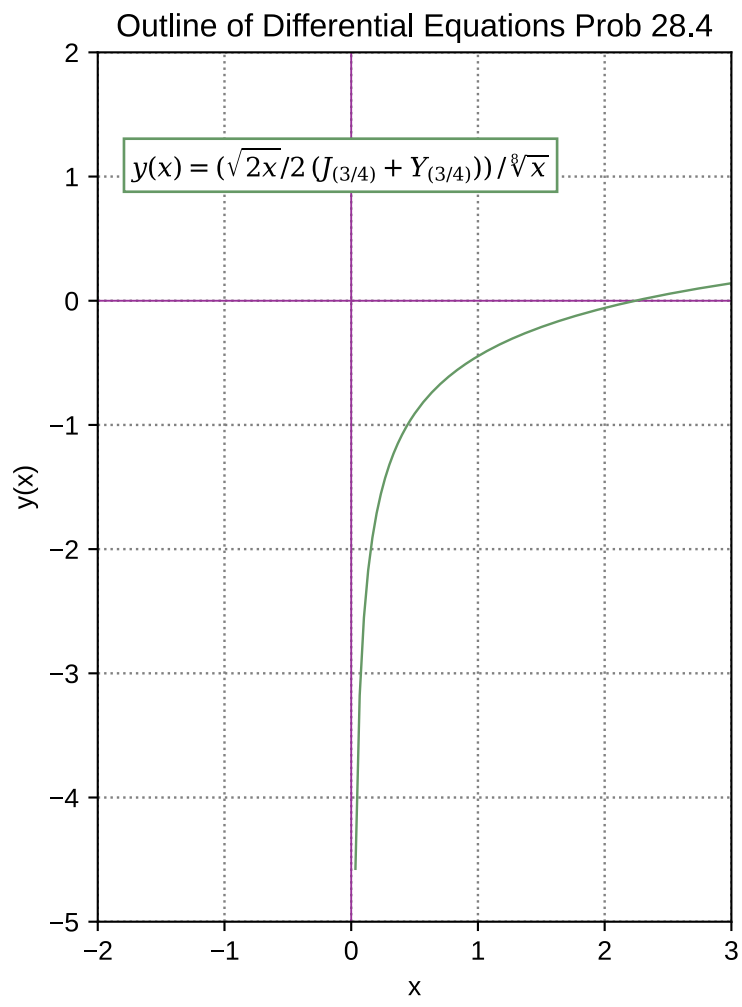
Out[13]: $8x^2\frac{d^2}{dx^2}y(x) + 10x\frac{d}{dx}y(x) + (x-1)\,y(x) = 0$

In [14]:
```
sol = dsolve(ode, y)
sol
```

Out[14]: $$y(x) = \frac{C_1 J_{\frac{3}{4}}\left(\frac{\sqrt{2}\sqrt{x}}{2}\right) + C_2 Y_{\frac{3}{4}}\left(\frac{\sqrt{2}\sqrt{x}}{2}\right)}{\sqrt[8]{x}}$$

Above is shown the sympy solution of the problem. It is somewhat less complicated than the Wolfram Alpha solution. Neither one shows an imaginary element, which makes the solution easier to plot. The order of the Bessel factors is seen to be fractional, not integer. Although some scipy kinds of bessel species are Integer, the most common of the J and Y types are Real, which is fortunate. Since no initial conditions are available, the value of both constants will be set to 1.

In [69]:
```
import numpy as np
import scipy.special as spe
import sympy as sp
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.linspace(0,10,300)
y = (spe.jv(0.75,(np.sqrt(2*x)/2)) + spe.yv(0.75, (np.sqrt(2*x)/2))) /x**(1/8)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 28.4")
plt.rcParams['figure.figsize'] = [5, 6]
plt.rcParams['font.sans-serif'] = ['Liberation Sans']
plt.rcParams['mathtext.fontset'] = 'dejavuserif'

ax = plt.gca()
ax.axhline(y=0, color='#993399', linewidth=0.7)
ax.axvline(x=0, color='#993399', linewidth=0.7)
ratio = 0.98
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

plt.text(-1.73, 1, "$y(x) = (\sqrt{2x}/2\,(J_{(3/4)}+Y_{(3/4)}))\,/\,\sqrt[8]{x}$",\
         size=10,bbox=dict(boxstyle="square",ec=('#669966'),fc=(1., 1., 1),))
plt.ylim(-5, 2)
plt.xlim(-2, 3)
plt.plot(x, y, linewidth = 0.9, color = '#669966')
plt.show()
```



Unlike the pure Bessel function, the solution equation does not show a propensity to recross the x-axis.

28.9 Find the general solution near $x = 0$ of $3x^2y'' - xy' + y = 0$.

This equation can be entered into Wolfram Alpha, and the entry line is:

!| 3 x^2 y'' - x y' + y = 0 |!

and the result returned is:

$$y(x) = c_1 \sqrt[3]{x} + c_2 x$$

28.10 Find the general solution near $x = 0$ of $x^2y'' + xy' + x^2y = 0$.

This equation can be entered into Wolfram Alpha, and the entry line is:

!| x^2 y'' + x y' + x^2 y = 0 |!

and the result returned is:

$$y(x) = c_1 J_0(x) + c_2 Y_0(x)$$

In addition to the answer, Wolfram Alpha gives sample plots complete with initial conditions. These could be useful to check the proper application of Bessel calculations to sample plots. The first set of initial conditions posits $y(1) = 1, \quad y'(1) = 0.$

The constants for the solution equation itself can be found using Maxima:

```
(%i1) bessel_j(0,1);
(%o1) bessel_j(0,1)
(%i2) float(%);
(%o2) 0.7651976865579666
(%i3) bessel_y(0,1);
(%o3) bessel_y(0,1)
(%i4) float(%);
(%o4) 0.08825696421567691
```

Which results in the equation:

$$c_1(0.765197686557) + c_2(0.088256964215) = 1$$

```
(%i17) bessel_j(1,1);
(%o17) bessel_j(1,1)
(%i18) float(%);
(%o18) 0.4400505857449335
(%i19) bessel_y(1,1);
(%o19) bessel_y(1,1)
(%i20) float(%);
(%o20) -0.7812128213002888
```

And results in another equation:

$$c_1(-0.440050585744) + c_2(0.781212821300) = 0$$

and brings circumstances to a favorable position regarding a sympy simultaneous solution.

It turns out that Wolfram Alpha can differentiate the Bessels. So

!| D[j0(x)] |!

gives the derivative for J, which is

$$\frac{1}{2}\,(J_{-1}(x) - \frac{J_0(x) + x\,J_1(x)}{x})$$

or, knowing that $J_0(x) = 1$ when $x = 1$

$$\frac{1}{2}\,(J_{-1}(1) - 1 - J_1(1))$$

Leaving this line of deduction alone for awhile, and looking at Y,

!| D[Y_0(x)] |!

gives the derivative for Y, which is

$$-Y_1(x)$$

or in this case, $Y_1(1)$. There would now be a need to go back to Maxima to get one more current value, except for the fact that, in Maxima, bessel_y(-1 , 1) = −bessel_y(1 , 1) , which has already been retrieved. Likewise, Maxima informs that bessel_j(-1 , 1) = −bessel_j(1 , 1) .

Assembling,

$$c1\,(\frac{1}{2}\,(-0.44005058) - (1) - (0.44005058)) + c2\,(-(-0.078121282)) = 0$$

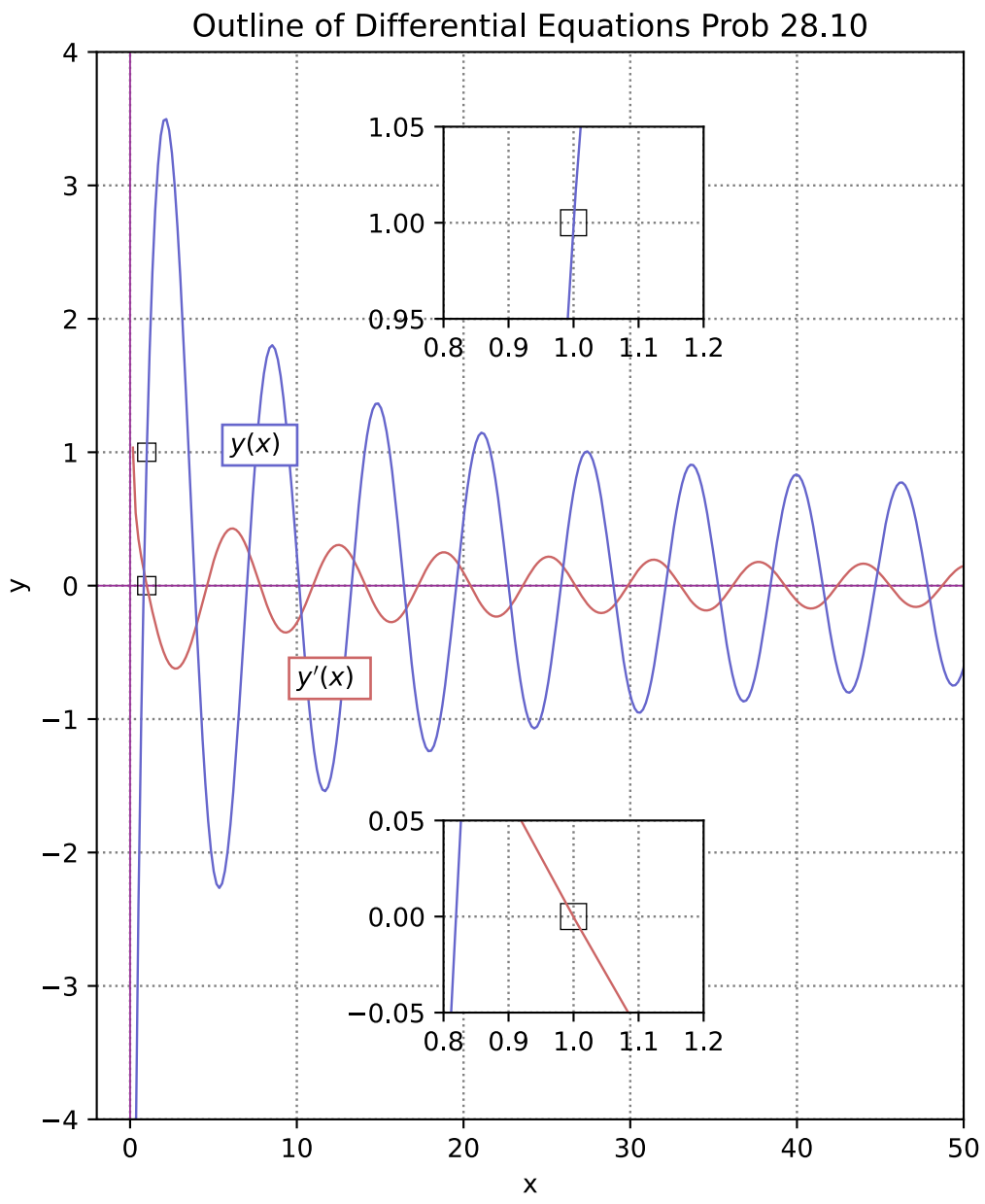
$$\implies c1\,(0.5)\,(-1.880101171489867) + c2\,(0.07812128213002888) = 0$$

In [1]:
```
import sympy as sp

c1, c2 = sp.symbols('c1 c2')

eq1 = sp.Eq( c1*(0.765197686557) + c2*(0.088256964215), 1)
eq2 = sp.Eq(c1*(0.5)*(-1.880101171489867) + c2*(0.0781212821300), 0)
res = sp.solve([eq1, eq2], (c1, c2))
res
```

Out[1]:
```
{c1: 0.547281520588188, c2: 6.58555901758219}
```

In [15]:

```
import numpy as np
import scipy.special as spe
import sympy as sp
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.setdiff1d(np.linspace(0.,50.,300),[0]) #to remove the zero
#y01 = (1.14757961723141*spe.jv(0,x) + 1.38090781660771*spe.yv(0, x))
y01 = (0.547281520588188*spe.jv(0,x) + 6.58555901758219*spe.yv(0, x))  # = y(x)
y4 = 0.475*(spe.jv(-1,x) - (spe.jv(0,x) + x*spe.jv(1,x))/x) +  spe.yv(-1, x)  # = y'(x)

fig, ax = plt.subplots()
axin1 = ax.inset_axes([0.4, 0.1,
                       0.3, 0.18])
axin1.set(xlim=(0.8, 1.2), ylim=(-.05, .05))
axin1.set_xticks([0.8, 0.9, 1.0, 1.1, 1.2], labels=\
    [ '0.8', '0.9', '1.0',  '1.1', '1.2'])
axin1.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
ax.grid(True, linestyle='dotted', color='gray', linewidth=0.9)

axin2 = ax.inset_axes([0.4, 0.75,   # H placement begin %total H, V placement of % total V
                       0.3, 0.18]) # W box width of % total W, box height of % total V
axin2.set(xlim=(0.8, 1.2), ylim=(.95, 1.05))
axin2.set_xticks([0.8, 0.9, 1.0, 1.1, 1.2], labels=\
    ['0.8', '0.9', '1.0',  '1.1', '1.2'])
axin2.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 28.10')
plt.rcParams['figure.figsize'] = [6, 9] #  WxH
plt.xlabel('x')
plt.ylabel('y')
ax.axhline(y=0, color='#993399', linewidth=0.7)
ax.axvline(x=0, color='#993399', linewidth=0.7)
ratio = 8
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

#plt.text(15, 1, "$y(x) = (\sqrt{2x}/2\,(J_{(3/4)}+Y_{(3/4)}))\,/\,\sqrt[8]{x}$",\
         #size=10,bbox=dict(boxstyle="square",ec=('#669966'),fc=(1., 1., 1),))
plt.text(6, 1, "$y(x)$ ",\
         size=10,bbox=dict(boxstyle="square",ec=('#6666CC'),fc=(1., 1., 1),))
plt.text(10, -0.75, "$y'(x)$ ",\
         size=10,bbox=dict(boxstyle="square",ec=('#CC6666'),fc=(1., 1., 1),))

xpts = np.array([1, 1])
ypts = np.array([1, 0])
plt.plot(xpts, ypts, markersize=7, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
axin1.plot(xpts, ypts, markersize=10, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
axin2.plot(xpts, ypts, markersize=10, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.ylim(-4, 4)
plt.xlim(-2, 50)
plt.plot(x, y4, linewidth = 0.9, color = '#CC6666')
plt.plot(x, y01, linewidth = 0.9, color = '#6666CC')

axin1.plot(x, y4, color = '#CC6666', linewidth = 0.9)
axin1.plot(x, y01, color = '#6666CC', linewidth = 0.9)
axin2.plot(x, y01, color = '#6666CC', linewidth = 0.9)
plt.show()
```



Above: It appears that both functions meet the prescribed initial conditions.

28.12 Use the method of Frobenius to find one solution near $x = 0$ of $x^2 y'' - x y' + y = 0$.

This equation can be solved by Wolfram Alpha. The entry line is:

!| x^2 y'' - x y' + y = 0 |!

and the returned answer is:

$$y(x) = c_1 x + c_2 x \log(x)$$

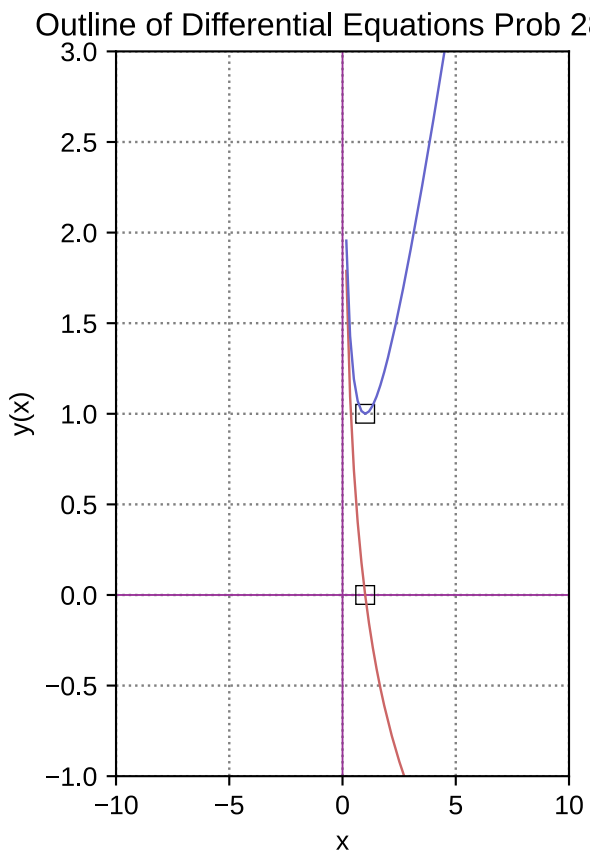
As a possible IVP, Wolfram Alpha suggests the initial conditions: $y(1) = 1, \quad y'(1) = 0$.

The derivative of the solution is:

$$y'(x) = c_1 + c_2 \log(x) + c_2$$

As for the constants, from the solution equation it can be seen that $c_1 = 1$ is the logical choice, whereas c_2 can be anything. From the derivative equation, $c_1 + c_2 = 0$, which implies that c_2 equals -1.

In [63]:
```
import numpy as np
import scipy.special as spe
import sympy as sp
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.setdiff1d(np.linspace(0.,50.,300),[0]) #to remove the zero
y01 = x - np.log(x)
y4 = 1 - np.log(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 28.12")
plt.rcParams['figure.figsize'] = [6, 5]
plt.rcParams['font.sans-serif'] = ['Liberation Sans']
plt.rcParams['mathtext.fontset'] = 'dejavuserif'

ax = plt.gca()
ax.axhline(y=0, color='#993399', linewidth=0.7)
ax.axvline(x=0, color='#993399', linewidth=0.7)
ratio = 8
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

#plt.text(15, 1, "$y(x) = (\sqrt{2x}/2\,(J_{(3/4)}+Y_{(3/4)}))\,/\,\sqrt[8]{x}$",\
        #size=10,bbox=dict(boxstyle="square",ec=('#669966'),fc=(1., 1., 1),))
xpts = np.array([1, 1])
ypts = np.array([1, 0])
plt.plot(xpts, ypts, markersize=7, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.ylim(-1, 3)
plt.xlim(-10, 10)
plt.plot(x, y4, linewidth = 0.9, color = '#CC6666')
plt.plot(x, y01, linewidth = 0.9, color = '#6666CC')
plt.show()
```



28.14 Use the method of Frobenius to find one solution near $x = 0$ of $x^2 y'' + (x^2 - 2x)y' + 2y = 0$.

This equation can be entered into Wolfram Alpha. The entry line is:

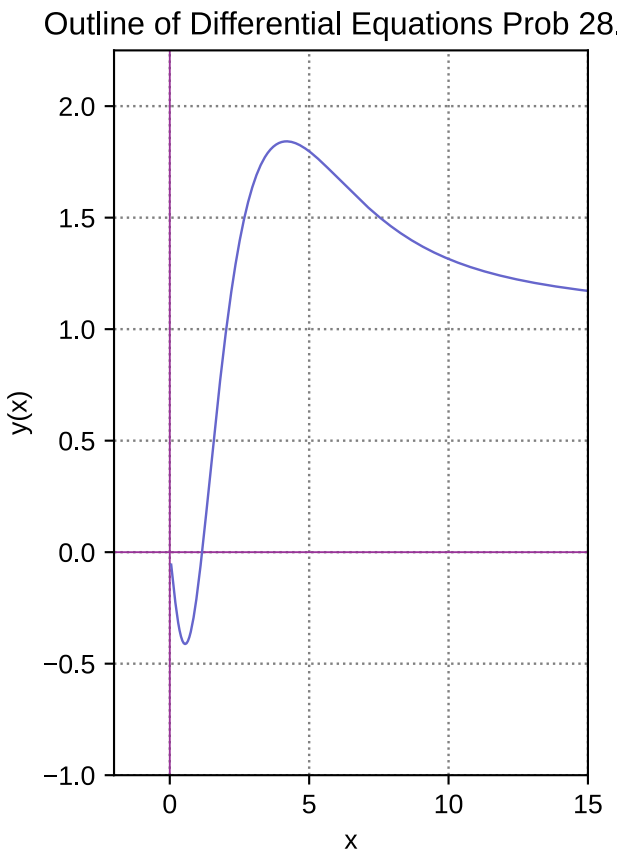
!| x^2 y'' + (x^2 - 2 x) y' + 2 y = 0 |!

and the returned answer is:

$$y(x) = c_1 e^{-x} x^2 \left(\text{Ei}(x) - \frac{e^x}{x}\right) + c_2 e^{-x} x^2$$

Ei, or the exponential function, is included in the scipy special functions section. Therefore the equation should be plottable.

In [79]:
```
import numpy as np
import scipy.special as spe
import sympy as sp
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.setdiff1d(np.linspace(0.,15.,300),[0]) #to remove the zero
y = np.exp(-x)*x**2*(spe.expi(x) - (np.exp(x)/x) + np.exp(-x)*x**2)

plt.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
plt.xlabel("x")
plt.ylabel("y(x)")
plt.title("Outline of Differential Equations Prob 28.14")
plt.rcParams['figure.figsize'] = [6, 5]
plt.rcParams['font.sans-serif'] = ['Liberation Sans']
plt.rcParams['mathtext.fontset'] = 'dejavuserif'

ax = plt.gca()
ax.axhline(y=0, color='#993399', linewidth=0.7)
ax.axvline(x=0, color='#993399', linewidth=0.7)
ratio = 8
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

#plt.text(15, 1, "$y(x) = (\sqrt{2x}/2\,(J_{(3/4)}+Y_{(3/4)}))\,/\,\sqrt[8]{x}$",\
         #size=10,bbox=dict(boxstyle="square",ec=('#669966'),fc=(1., 1., 1),))
#xpts = np.array([1, 1])
#ypts = np.array([1, 0])
#plt.plot(xpts, ypts, markersize=7, color='k', marker='s', \
         #mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.ylim(-1, 2.25)
plt.xlim(-2, 15)
#plt.plot(x, y4, linewidth = 0.9, color = '#CC6666')
plt.plot(x, y, linewidth = 0.9, color = '#6666CC')
plt.show()
```



28.16 Use the method of Frobenius to find one solution near $x = 0$ of $x^2 y'' + x y' + (x^2 - 1) y = 0$.

The assignment is to find one solution of the given equation close to $x = 0$. This is very simple in Wolfram Alpha. The value $x = 0.001$ is pretty close to zero, so try that one.

The entry line could be:

!| x^2 y'' + x y' + (x^2 - 1) y = 0, y(0.001)=1, y'(0.001)=0 |!

for which the returned answer is:

$$y(x) = 999.997\, J_1(x) - 0.000785398\, Y_1(x)$$

and to get the first derivative of y, enter

!| D[999.997 J_1(x) - 0.000785398 Y_1(x)] |!

and get back:

$$y'(x) = 499.999\, J_0(x) - 499.999\, J_2(x) - 0.000392699\, Y_0(x) + 0.000392699\, Y_2(x)$$

The function derivative is checked with Maxima, which delivers virtually the same answer, with, however, an extra decimal place.

After plotting the solution and the derivative of the solution, it becomes apparent that there is a lot of inaccuracy contained in the derivative. Python (sympy) can correct some of this.

In [49]:
```
from sympy import *
import sympy as sp
x = sp.Symbol('x', positive=True)
y = sp.Function('y')(x)
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)
equation = sp.Eq(x**2*ydp + x*yp + (x**2-1)*y, 0)
display(equation)
```

$$x^2 \frac{d^2}{dx^2} y(x) + x \frac{d}{dx} y(x) + \left(x^2 - 1\right) y(x) = 0$$

In [50]:
```
ics = {y.subs(x, 0.001): 1, yp.subs(x, 0.001): 0}
ics
```

Out[50]: {y(0.001): 1, Subs(Derivative(y(x), x), x, 0.001): 0}

In [51]:
```
ivp = sp.dsolve(equation, ics = ics)
ivp
```

Out[51]: $y(x) = 999.996738158091 J_1\left(x\right) - 0.00078539786887315 8 Y_1\left(x\right)$

In [52]:
```
sp.checkodesol(equation, ivp)
```

Out[52]: (True, 0)

In [53]:
```
y = 999.996738158091*sp.besselj(0,x) - 0.00078539786887315 8*sp.bessely(1,x)
y
```

Out[53]: $999.996738158091 J_0\left(x\right) - 0.00078539786887315 8 Y_1\left(x\right)$

In [54]:
```
dif1 = y.diff(x)
dif1
```

Out[54]: $-999.996738158091J_1\left(x\right) - 0.000392698934436579Y_0\left(x\right) + 0.000392698934436579Y_2\left(x\right)$

In [55]:
```
print(dif1.evalf(subs={x:0.001}))
```

```
-500.498179886250
```

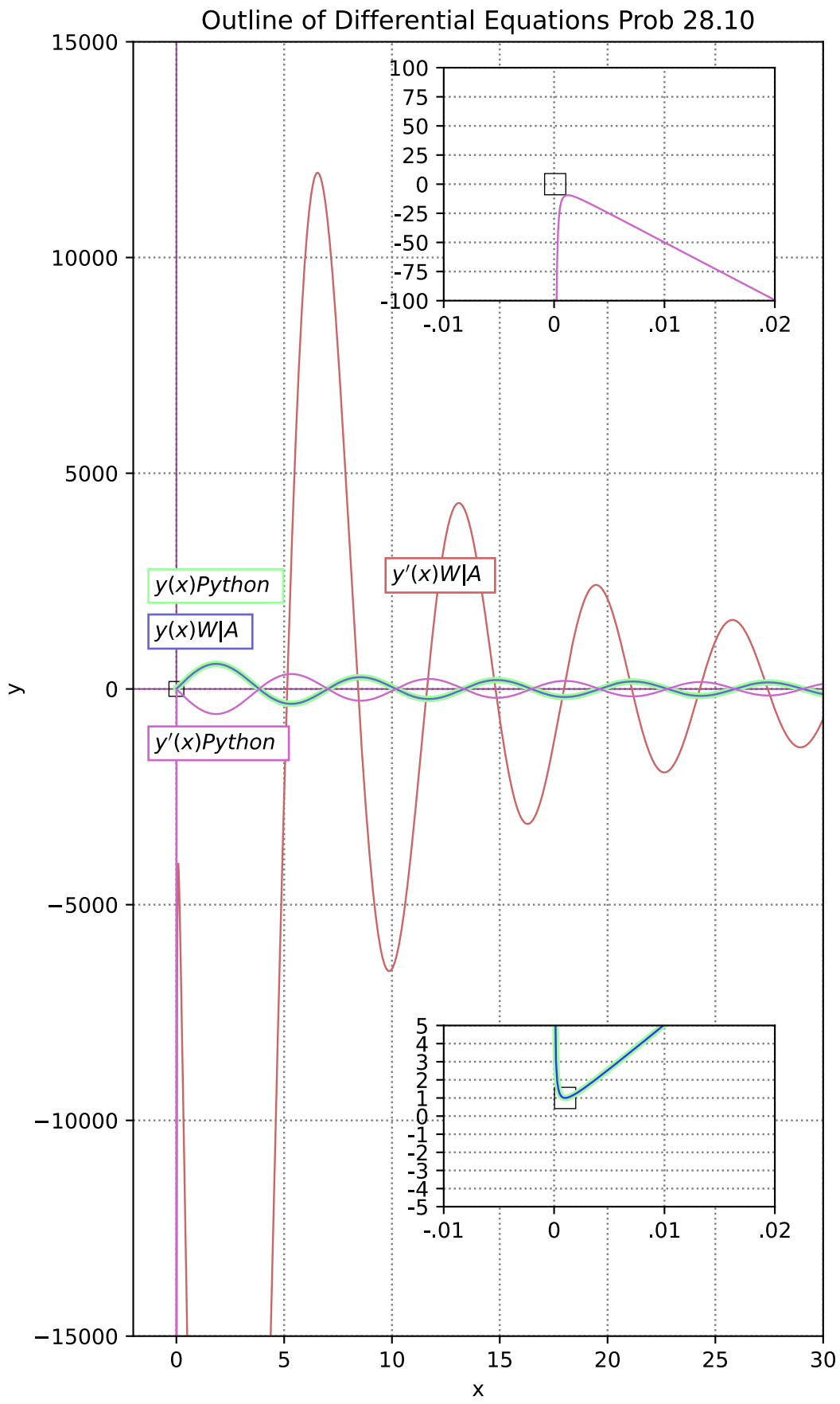
Look at all the decimal places that are suddenly available, thanks to Python!

However, even with the abundance of decimal places, the plot for the derivative is not very accurate. It only comes within about 500 units of the target. And this includes a linspace division of 500000, a huge amount. In the plot the red trace is the version due to Wolfram Alpha and Maxima, and the fuchsia trace is the one due to Python. It is uncertain what changes could be made to the procedure to get greater accruacy at the initial condition for the derivative.

Though the accuracy of the derivative fell short of desired, not too much emphasis should be placed thereon. When inventing arbitrary initial conditions using extreme values, expecting great accuracy with software may be unrealistic.

In [56]:

```
import numpy as np
import scipy.special as spe
import sympy as sp
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

#x = np.setdiff1d(np.linspace(0.,50.,300),[0]) #to remove the zero
x = np.linspace(-5.,50.,500000)
y01 = (999.997*spe.jv(1,x) - 0.000785398*spe.yv(1, x))  # = y(x) W|A
y02 = (999.996738158091*spe.jv(1,x) - 0.0007853978688873158*spe.yv(1, x))  # = y(x) Python
y4 = 499.9985*(spe.jv(0,x) - 499.9985*(spe.jv(2,x) - 0.000392699*spe.yv(0,x))/x) +  \
     0.000392699*spe.yv(2, x)  # = y'(x) W|A
y7 = -999.996738158091*spe.jv(1,x) -0.00039269893443657 9*spe.yv(0,x) +\
     0.00039269893443657 9*spe.yv(2,x) #=y'(x) Python

fig, ax = plt.subplots()
axin1 = ax.inset_axes([0.45, 0.1,
                       0.48, 0.14])
axin1.set(xlim=(-0.01, 0.02), ylim=(-5,5))
axin1.set_xticks([ -.01, 0, .01, .02], labels=\
    [ '-.01', '0', '.01', '.02'])
axin1.set_yticks([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5], labels=\
    ['-5', '-4', '-3', '-2', '-1', '0', '1', '2', '3', '4', '5'])
axin1.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
ax.grid(True, linestyle='dotted', color='gray', linewidth=0.9)

axin2 = ax.inset_axes([0.45, 0.8,   # H placement begin %total H, V placement of % total V
                       0.48, 0.18]) # W box width of % total W, box height of % total V
axin2.set(xlim=(-0.1, 0.2), ylim=(-100,100))
axin2.set_xticks([ -.1, 0, .1, .2], labels=\
    [ '-.01', '0', '.01', '.02'])
axin2.set_yticks([-100, -75, -50, -25, 0, 25, 50, 75, 100], labels=\
    ['-100', '-75', '-50', '-25', '0', '25', '50', '75', '100'])

axin2.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 28.10')
plt.rcParams['figure.figsize'] = [10,11] #  WxH
plt.xlabel('x')
plt.ylabel('y')
ax.axhline(y=0, color='#993399', linewidth=0.7)
ax.axvline(x=0, color='#993399', linewidth=0.7)
ratio = 0.002
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

#plt.text(15, 1, "$y(x) = (\sqrt{2x}/2\,(J_{(3/4)}+Y_{(3/4)}))\,/\,\sqrt[8]{x}$",\
         #size=10,bbox=dict(boxstyle="square",ec=('#669966'),fc=(1., 1., 1),))
plt.text(-1, 1200, "$y(x) W|A$ ",\
         size=10,bbox=dict(boxstyle="square",ec=('#6666CC'),fc=(1., 1., 1),))
plt.text(-1, -1400, "$y'(x) Python$ ",\
         size=10,bbox=dict(boxstyle="square",ec=('#CC66CC'),fc=(1., 1., 1),))
plt.text(-1, 2250, "$y(x) Python$ ",\
         size=10,bbox=dict(boxstyle="square",ec=('#99FF99'),fc=(1., 1., 1),))
plt.text(10, 2500, "$y'(x) W|A$ ",\
         size=10,bbox=dict(boxstyle="square",ec=('#CC6666'),fc=(1., 1., 1),))

xpts = np.array([.001])
ypts = np.array([1])
x1pts = np.array([.001])
y1pts = np.array([0])
plt.plot(xpts, ypts, markersize=7, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
axin1.plot(xpts, ypts, markersize=10, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
axin2.plot(x1pts, y1pts, markersize=10, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.ylim(-15000, 15000)
plt.xlim(-2, 30)
plt.plot(x, y4, linewidth = 0.9, color = '#CC6666')
plt.plot(x, y02, linewidth = 3, color = '#99FF99')
plt.plot(x, y01, linewidth = 0.9, color = '#6666CC')
plt.plot(x, y7, linewidth = 0.9, color = '#CC66CC')

axin1.plot(x, y02, color = '#99FF99', linewidth = 3)
axin1.plot(x, y01, color = '#3333FF', linewidth = 0.9)
axin2.plot(x, y4, color = '#CC6666', linewidth = 0.9)
axin2.plot(x, y7, color = '#CC66CC', linewidth = 0.9)
plt.show()
#print()
```



28.18 Use the method of Frobenius to find one solution near $x = 0$ of $x^2 y'' + (x^2 + 2x) y' - 2y = 0$.

Benefitting from experience in the last problem, a distance of fully 0.1 units away from $x = 0$ will be treated as the closest point. The entry for Wolfram Alpha is made as:

!| x^2 y''+(x^2+2 x) y'-2 y=0, y(0.1)=1, y'(0.1)=0 |!

and the answer is returned as:

$$y(x) = \frac{21\,x^2 - 42\,x - 41.9965\,e^{-x} + 42}{x^2}$$

Following this up with the entry:

!| D[(21 x^2-42 x-41.9964 e^(-x)+42)/x^2] |!

supplies the derivative function

$$y'(x) = \frac{e^{-x}(41.9964\,x + e^{x}(42x - 84) + 83.9928)}{x^3}$$

However, if W|A is asked to calculate this function's value at $x = 0.1$, like this

!| N[(e^(-0.1) (41.9964*0.1 + e^(0.1) (42*0.1 -84)+83.9928))/(0.1)^3] |!

it produces the value -0.1803001087175013, which is a long way away from 0.

Note: asterisks above are escaped.

Below: Note: if the x symbol is not listed with Sympy as being strictly positive, the solution to the ODE becomes intractably large.

In [1]:
```
import sympy as sp
x = sp.Symbol('x', positive=True)
y = sp.Function('y')(x)
yp = sp.Derivative(y, x)
ydp= sp.Derivative(y, x, x)
equation = sp.Eq(x**2*ydp + (x**2 + 2*x)*yp - 2*y, 0)
display(equation)
```

$$x^2 \frac{d^2}{dx^2} y(x) + \left(x^2 + 2x\right) \frac{d}{dx} y(x) - 2y(x) = 0$$

In [2]:
```
ics = {y.subs(x, 0.1): 1, yp.subs(x, 0.1): 0}
ics
```

Out[2]: {y(0.1): 1, Subs(Derivative(y(x), x), x, 0.1): 0}

In [3]:
```
ivp = sp.dsolve(equation, ics = ics)
ivp
```

Out[3]:
$$y(x) = 0.00301959379397967 x^{-\frac{x}{2} - \frac{\sqrt{x^2+2x+9}}{2} - \frac{1}{2}} + 6.0053208106784 x^{-\frac{x}{2} + \frac{\sqrt{x^2+2x+9}}{2} - \frac{1}{2}}$$

In [ ]:
```
sp.checkodesol(equation, ivp)
```

Above: *checkodesol* does not work properly with this problem.

Below: Although it is necessary to input the solution $y(x)$ by hand to access it for calculation, Python can compute its derivative and display the value of interest.

In [4]:
```
y = 0.00301959379397967*x**((-x/2)-(sp.sqrt(x**2+2*x+9))/2 -(1/2)) +\
6.0053208106784*x**((-x/2)+(sp.sqrt(x**2+2*x+9)/2)-(1/2))
y
```

Out[4]:
$$0.00301959379397967 x^{-\frac{x}{2} - \frac{\sqrt{x^2+2x+9}}{2} - 0.5} + 6.0053208106784 x^{-\frac{x}{2} + \frac{\sqrt{x^2+2x+9}}{2} - 0.5}$$

In [5]:
```
dif1 = y.diff(x)
dif1
```

Out[5]:

$$0.00301959379397967x^{-\frac{x}{2}-\frac{\sqrt{x^2+2x+9}}{2}-0.5}\left(\left(-\frac{x+1}{2\sqrt{x^2+2x+9}}-\frac{1}{2}\right)\log(x)+\frac{-\frac{x}{2}-\frac{\sqrt{x^2+2x+9}}{2}-0.5}{x}\right)$$

$$+\,6.0053208106784x^{-\frac{x}{2}+\frac{\sqrt{x^2+2x+9}}{2}-0.5}\left(\left(\frac{x+1}{2\sqrt{x^2+2x+9}}-\frac{1}{2}\right)\log(x)+\frac{-\frac{x}{2}+\frac{\sqrt{x^2+2x+9}}{2}-0.5}{x}\right)$$

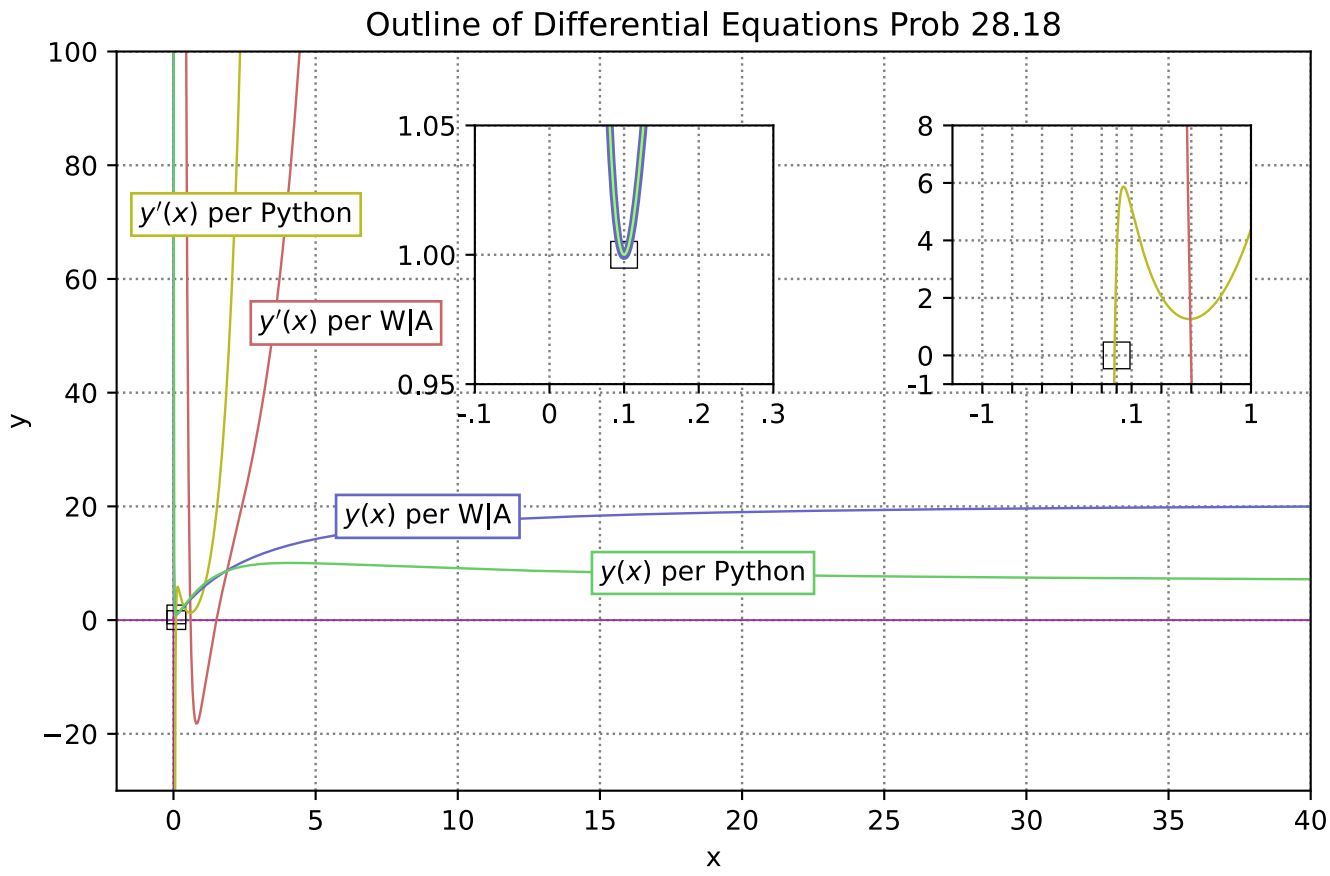
In [6]:
```
print(dif1.evalf(subs={x:0.1}))
```

```
4.51309599661136e-15
```

Python's answer for the value of $y'(0.1)$ is good; W|A's answer is not useful.

In [30]:

```
import numpy as np
import scipy.special as spe
import sympy as sp
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.setdiff1d(np.linspace(0.,50.,50000),[0]) #to remove the zero
#y01 = (1.14757961723141*spe.jv(0,x) + 1.38090781660771*spe.yv(0, x))
y01 = (21*x**2 - 42*x - 41.9965*np.exp(-x) + 42)/x**2 # = y(x)
y02 = 0.00301959379397967*x**((-x/2) -(np.sqrt(x**2+2*x+9)/2) -(1/2)) + \
      6.0053208106784*x**((-x/2) +(np.sqrt(x**2+2*x+9)/2) -(1/2))
y4 = (np.exp(-x)*41.9965 + np.exp(x)*(42*x - 84) + 83.9928)/x**3 # = y'(x)
y5 = (1/np.sqrt(x**2+2*x+9))*(x**((1/2)*(-np.sqrt(x**2+2*x+9))-x-3))*\
                (3.0026604053392*x**(np.sqrt(x**2+2*x+9))*(np.sqrt(x*2+2*x+9)-x-1)*\
                 (np.sqrt(x**2+2*x+9)-x*np.log(x))-0.00150979689698984*\
                (np.sqrt(x*2+2*x+9)+x+1)*(np.sqrt(x**2+2*x+9)+x*np.log(x)))
y6 = (42*x + 41.9964*np.exp(-x)-42)/x**2 - (2*(21*x**2 - 42*x - 41.9964*np.exp(-x)+42))/x**3
y7 = 0.00301959379397967*x**\
           ((-x/2)-(np.sqrt(x**2+2*x+9)/2)-0.5)*((-(x-1)/(2*np.sqrt(x**2+2*x+9)-(1/2)))*\
           np.log(x) + ((-(x/2)-(np.sqrt(x**2+2*x+9)-0.5))/x))\
           +6.0053208106784*x**(-(1/2)+(np.sqrt(x**2+2*x+9)/2)-(1/2))\
           *(((x+1)/2*np.sqrt(x**2+2*x+9)-(1/2))*np.log(x) + ((-x/2)+(np.sqrt(x**2+2*x+9)/2 -0.

fig, ax = plt.subplots()
axin1 = ax.inset_axes([0.7, 0.55,
                       0.25, 0.35])
axin1.set(xlim=(-1,1), ylim=(-1, 8))
axin1.set_yticks([-1, 0, 2, 4, 6, 8], labels=\
    [ '-1', '0', '2', '4', '6', '8'])
axin1.set_xticks([ -.8, -.6, -.4, -.2, 0, .1, .2,.4,.6, .8,1], labels=\
    [ '-1','','','','','','.1','','','','','1'])
axin1.grid(True, linestyle='dotted', color='gray', linewidth=0.9)
ax.grid(True, linestyle='dotted', color='gray', linewidth=0.9)

axin2 = ax.inset_axes([0.3, 0.55,   # H placement begin %total H, V placement of % total V
                       0.25, 0.35]) # W box width of % total W, box height of % total V
axin2.set(xlim=(-.1, .3), ylim=(.95, 1.05))
axin2.set_xticks([ -.1, 0, .1, .2, .3], labels=\
    [ '-.1', '0', '.1', '.2', '.3'])
axin2.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 28.18')
plt.rcParams['figure.figsize'] = [10, 5] #  WxH
plt.xlabel('x')
plt.ylabel('y')
ax.axhline(y=0, color='#993399', linewidth=0.7)
ax.axvline(x=0, color='#993399', linewidth=0.7)
ratio = 0.2
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

#plt.text(15, 1, "$y(x) = (\sqrt{2x}/2\,(J_{(3/4)}+Y_{(3/4)}))\,/\,\sqrt[8]{x}$",\
         #size=10,bbox=dict(boxstyle="square",ec=('#669966'),fc=(1., 1., 1),))
plt.text(6, 17, "$y(x)$ per W|A",\
         size=10,bbox=dict(boxstyle="square",ec=('#6666CC'),fc=(1., 1., 1),))
plt.text(15, 7, "$y(x)$ per Python",\
         size=10,bbox=dict(boxstyle="square",ec=('#66CC66'),fc=(1., 1., 1),))
plt.text(3, 51, "$y'(x)$ per W|A",\
         size=10,bbox=dict(boxstyle="square",ec=('#CC6666'),fc=(1., 1., 1),))
plt.text(-1.2, 70, "$y'(x)$ per Python",\
         size=10,bbox=dict(boxstyle="square",ec=('#BBBB22'),fc=(1., 1., 1),))


xpts = np.array([.1])
ypts = np.array([1])
zpts = np.array([0])
plt.plot(xpts, ypts, markersize=7, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.plot(xpts, zpts, markersize=7, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
axin1.plot(xpts, zpts, markersize=10, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
axin2.plot(xpts, ypts, markersize=10, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.ylim(-30, 100)
plt.xlim(-2, 40)
plt.plot(x, y4, linewidth = 0.9, color = '#CC6666')
#plt.plot(x, y5, linewidth = 0.9, color = '#DD66DD')
plt.plot(x, y7, linewidth = 0.9, color = '#BBBB22')
plt.plot(x, y01, linewidth = 0.9, color = '#6666CC')
plt.plot(x, y02, linewidth = 0.9, color = '#66CC66')
axin1.plot(x, y7, color = '#BBBB22', linewidth = 0.9)
axin1.plot(x, y4, color = '#CC6666', linewidth = 0.9)
axin2.plot(x, y01, color = '#6666CC', linewidth = 3)
axin2.plot(x, y02, color = '#99FF55', linewidth = 0.9)
plt.show()
```



28.21 Find the indicial equation of $x^2y'' + xe^xy' + (x^3 - 1)y = 0$ if the solution is required near $x = 0$.

Wolfram Alpha does **not** solve this equation. It draws a plot for it and offers several alternative forms, but no solution. However, sympy can solve it.

In [57]:
```
from sympy import *
import sympy as sp

x = symbols("x")
y = Function("y")(x)
y
```

Out[57]: $y(x)$

In [58]:
```
y.diff()
```

Out[58]: $\frac{d}{dx}y(x)$

In [59]:
```
ode = Eq(x**2*y.diff(x,x) + x*exp(x)*y.diff(x) + (x**3 - 1)*y , 0)
ode
```

Out[59]: $x^2\frac{d^2}{dx^2}y(x) + xe^x\frac{d}{dx}y(x) + \left(x^3 - 1\right)y(x) = 0$

In [60]:
```
sol = dsolve(ode, y)
sol
```

Out[60]:

$$y(x) = C_2 x \left(1 - \frac{x^3}{15}\right) + \frac{C_1 \left(\frac{x^6}{72} - \frac{x^3}{3} + 1\right)}{x} + O\left(x^6\right)$$

In [62]:
```
solapprox = x*(1-(x**3/15)) + ((x**6/72) - (x**3/3) + 1)/x
solapprox
```

Out[62]:

$$x \left(1 - \frac{x^3}{15}\right) + \frac{\frac{x^6}{72} - \frac{x^3}{3} + 1}{x}$$

In [65]:
```
print(solapprox.evalf(subs={x:0.1}))
```

```
10.0966601388889
```

In [66]:
```
print(solapprox.evalf(subs={x:0.01}))
```

```
100.009966666001
```

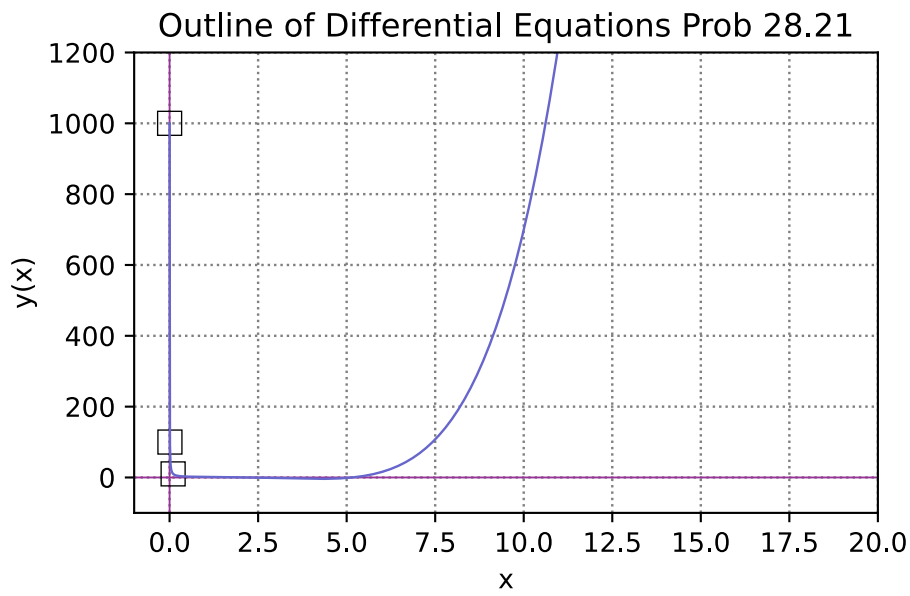
In [67]:
```
print(solapprox.evalf(subs={x:0.001}))
```

```
1000.00099966667
```

Notice that it is not necessary to import numpy to accomplish what has been done so far. A solution to the ODE has been provided, although the precision is not perfect, since the series remainder will have to be dropped at some point.

A plot of the solution function is made. The function values expand as the asymptote is approached. The three sample values calculated above are framed.

In [27]:
```
import numpy as np
import scipy.special as spe
import sympy as sp
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

x = np.setdiff1d(np.linspace(0.,30.,30000),[0]) #to remove the zero
y = x*(1 - (x**3/15)) + ((x**6/72) - (x**3/3) + 1)/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 28.21")
plt.rcParams['figure.figsize'] = [5, 6]
#plt.rcParams['font.sans-serif'] = ['Liberation Sans']
plt.rcParams['mathtext.fontset'] = 'dejavuserif'

ax = plt.gca()
ax.axhline(y=0, color='#993399', linewidth=0.7)
ax.axvline(x=0, color='#993399', linewidth=0.7)

ratio = 0.01
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

#plt.text(15, 1, "$y(x) = (\sqrt{2x}/2\,(J_{(3/4)}+Y_{(3/4)}))\,/\,\sqrt[8]{x}$",\
         #size=10,bbox=dict(boxstyle="square",ec=('#669966'),fc=(1., 1., 1),))
x1pts = np.array([.1])
x2pts = np.array([.01])
x3pts = np.array([.001])
y1pts = np.array([10.0966601388889])
y2pts = np.array([100.009966666001])
y3pts = np.array([1000.00099966667])
plt.plot(x1pts, y1pts, markersize=9, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.plot(x2pts, y2pts, markersize=9, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.plot(x3pts, y3pts, markersize=9, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.ylim(-100, 1200)
plt.xlim(-1, 20)
#plt.plot(x, y2, linewidth = 0.9, color = '#CC6666')
plt.plot(x, y, linewidth = 0.9, color = '#6666CC')
#plt.plot(x, y3, linewidth = 0.9, color = '#339933')
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: