(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 20 Numerical Methods for Solving 2nd Order Differential Equations via Systems. Manual calculations and reference to constructed tables forms the bulk of this chapter. However, because the philosophy of the project is only to get at an answer as quickly as possible (while using open source apps), the main thrust of the chapter is principally subverted.

20.1 Reduce the initial-value problem $3y'' - y = x; \quad y(0) = 0, y'(0) = 1$

This problem can be fed to Wolfram Alpha:

!| solve {3 * y'' - y = x}, y(0) = 0}, y(0) = 0, y'(0) = 1 |!

There is no need to recommend numerical methods to W|A here, because no mention is given about them. There is no "solution", but a "result", which does seem to be a solution.

Result: $y(x) = -x - \sqrt[3]{e^{-x/\sqrt{3}}} + \sqrt{3}\, e^{x/\sqrt{3}}$

20.2 Reduce the initial-value problem $3y'' - y = x; \quad y(0) = 0, y'(0) = 1$

This problem can be fed to Wolfram Alpha:

!| solve { y'' - 3 * y' + 2 * y = 0}, y(0) = -1, y'(0) = 0 |!

As before, there is no "solution", but a "result", which does seem to be a solution.

Result: $y(x) = e^x\,(e^x - 2)$

20.3 Reduce the initial-value problem $3x^2y'' - xy' + y = 0; \quad y(1) = 4, y'(1) = 2$

This problem can be fed to Wolfram Alpha:

!| solve {3 * x^2 * y'' - x * y' + y = 0}, y(1) = 4, y'(1) = 2 } |!

The role of numerical methods and approximation will be addressed starting in Problem 20.5.

Result: $y(x) = x + 3\sqrt[3]{x}$

20.4 Reduce the initial-value problem
$y''' - 2xy'' + 4y' - x^2 y = 1; \quad y(0) = 1, y'(0) = 2, y''(0) = 3$

This problem can be fed to Wolfram Alpha:

!| solve {y''' - 2 * x * y'' + 4 * y' - x^2 * y = 1}, y(0) = 1, y'(0) = 2, y''(0) = 3 |!

Somewhat surprisingly, considering the equation's complexity, Wolfram Alpha is able to get a solution for it.

Result:

$$y(x) = 1 + 2x + \frac{3x^2}{2} - \frac{7x^3}{6} - \frac{x^4}{4} + \frac{x^5}{60} + \frac{11x^7}{1260} -$$

$$\frac{x^8}{288} + \frac{43x^9}{90720} - \frac{47x^{10}}{129600} + \frac{43x^{11}}{831600} - \frac{1907x^{12}}{59875200} + O(x^{13})$$

The initial condition for $y(0)$ can easily be checked to give some degree of confidence in the solution, but to do very much with the whole equation would be very challenging.

20.5 Use Euler's method to solve $y'' - y = x; \quad y(0) = (0), y'(0) = 1$ on the interval [0, 1] with $h = 0.1$.

This problem can be fed to Wolfram Alpha:

!| y'' - y = x, y(0) = 0, y'(0) = 1, from 0 to 1 using Dormand-Prince method |!

Of the ten numerical methods available to W|F, the D-P method is the most accurate. For this problem Wolfram Alpha happens to have the exact solution available. The idea is that by entering the D-P method as the preferred option, the most accurate result will be obtained in case the exact solution is not available for some reason.If it had been necessary to use the numerical solve, D-P would have resulted in a global error of -2.60×10^{-9}.

Exact solution:

$y(x) = e^{-x}(-e^x x + e^{2x} - 1)$

20.6 Use Euler's method to solve $y'' - 3y' + 2y = 0; \quad y(0) = (-1), y'(0) = 0$ on the interval [0, 1] with $h = 0.1$.

This problem can be fed to Wolfram Alpha:

!| y'' - 3 * y' + 2 * y = 0, y(0) = -1, y'(0) = 0, from 0 to 1 using Dormand-Prince method |!

Wolfram Alpha had enough time to solve the problem, but not enough to summarize the potential performance of the various numerical methods.

Exact solution:

$y(x) = e^t (e^t - 2)$

20.7 Use the Runge-Kutta method to solve $y'' - y = x; \quad y(0) = (0), \ y'(0) = 1$ on the interval [0, 1] with $h = 0.1$.

This problem can be fed to Wolfram Alpha:

!| y'' - y = x, y(0) = 0, y'(0) = 1, from 0 to 1 using Dormand-Prince method |!

If it had been necessary to use the numerical solve, D-P would have resulted in a global error of -2.60×10^{-9}.

Exact solution:

$y(x) = e^{-x} + (-e^x x + e^{2x} - 1)$

20.8 Use the Runge-Kutta method to solve
$y'' - 3y' + 2y = 0; \quad y(0) = (-1), \ y'(0) = 0$ on the interval [0, 1] with $h = 0.1$.

This problem can be fed to Wolfram Alpha:

!| y'' - 3 *y' + 2 * y = 0, y(0) = -1, y'(0) = 0, from 0 to 1 using Dormand-Prince method |!

If it had been necessary to use the numerical solve, D-P would have resulted in a global error of -1.16×10^{-8}.

Exact solution:

$y(x) = e^x (e^x - 2)$

20.9 Use the Runge-Kutta method to solve
$3x^2 y'' - xy' + y = 0; \quad y(1) = (4), \ y'(1) = 2$ on the interval [1, 2] with $h = 0.1$.

This problem can be fed to Wolfram Alpha:

!| solve 3 * x^2 * y'' - x * y' + y = 0, y(1) = 4, y'(1) = 2 using Dormand-Prince method from 1 to 2 |!

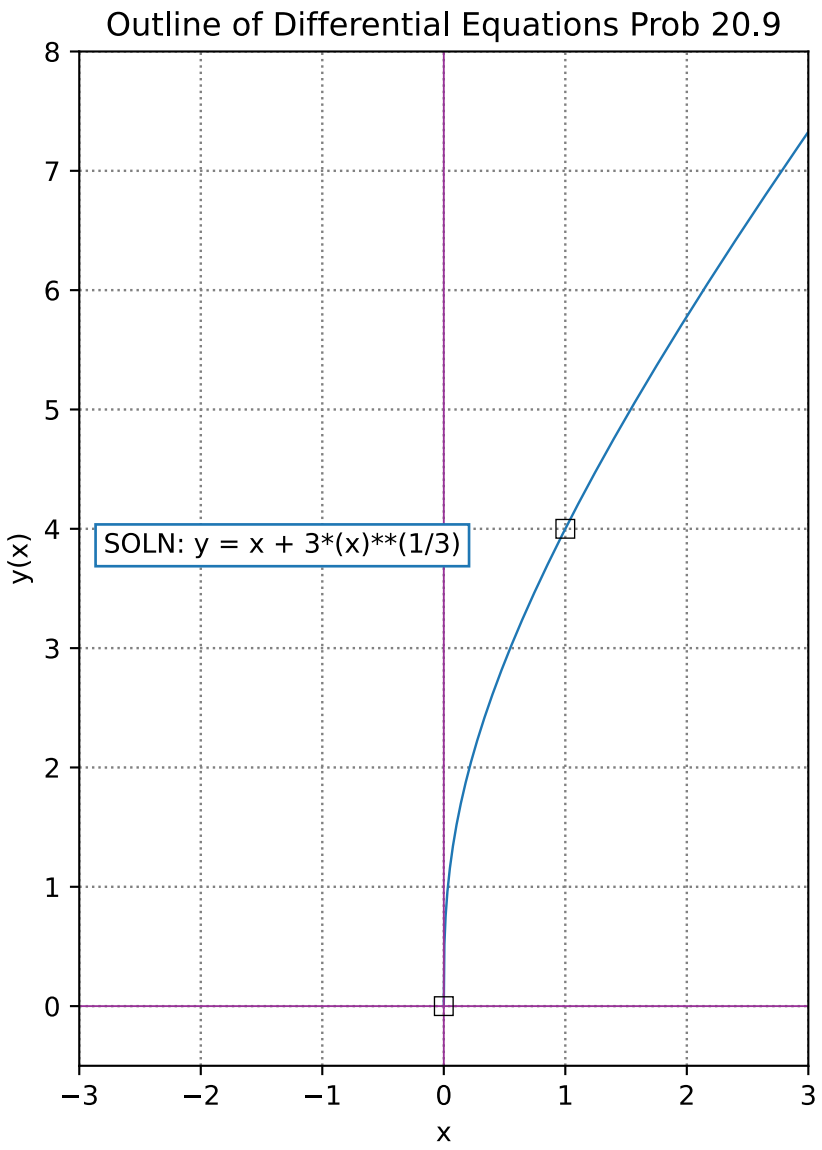
Here the behavior of W|A requires a little finesse. If an attempt to constrain the equation with the prescribed interval is attempted, W|A either refuses to do the problem or sticks a set of arbitrary constants in the solution. The trick is to include a call to a numerical method, as above. A plot may be able to show that the function itself is without blame.

Aside: (If the D-P method had actually been required, a global error in the solution of 2.71×10^{-9} would have been incurred. That the D-P method was **not** used is inferred from the inclusion of the term 'exact' in the output.)

Exact solution:

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

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

%config InlineBackend.figure_formats = ['svg']

x = np.arange(0, 3., 0.005)
y = x + 3*x**(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 20.9")
plt.rcParams["figure.figsize"] = [7, 7]

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

ax = plt.gca()
ratio = 2.4
#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, 15.])
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=0.7)
ax.axvline(x=0, color='#993399', linewidth=0.7)

plt.text(-2.8, 3.8, "SOLN: y = x + 3*(x)**(1/3)", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
xpts = np.array([0, 1])
ypts = np.array([0, 4])
plt.plot(xpts, ypts, markersize=7, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.ylim(-0.5, 8)
plt.show()
```



The function in the plot above looks continuous and well-behaved in the interval of interest.

Since the only difference between Problems 20.9 and 20.10 is the solution method, something which makes no difference to these pages, Problem 20.10 will be skipped.

20.11 Use the Adams-Bashforth-Moulton method to solve
$y'' - y = x; \quad y(0) = (0),\ y'(0) = 1$ on the interval [0, 1] with $h = 0.2$.

This problem can be fed to Wolfram Alpha:

!| solve y'' - y = x, y(0) = 0, y'(0) = 1 using Dormand-Prince method from 0 to 1 |!

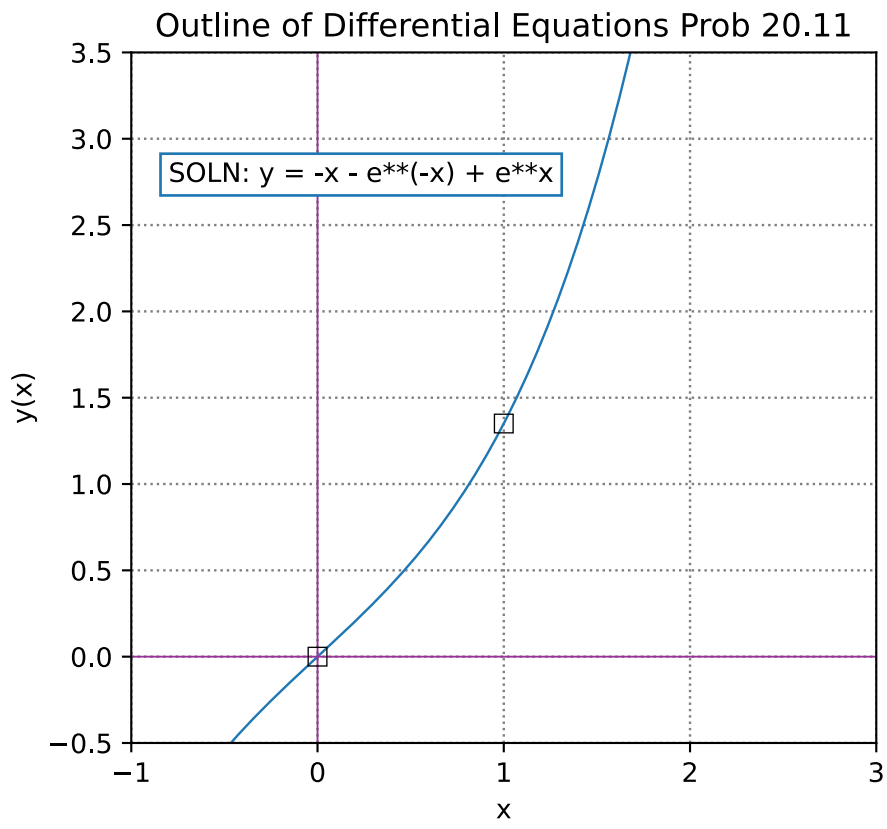
Wolfram Alpha appears to repent its confusing tricks. The way to get W|A to heed a specific interval appears to be to phrase the entry expression to include a numerical method, as above. (If the superfluous call to Dormand-Prince had been required, the global error in the result would have been -2.60×10^{-9}.) A representative plot is offered.

Exact solution of equation:

$y(x) = e^{-x}(-e^x x + e^{2x} - 1)$

In [6]:
```
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

%config InlineBackend.figure_formats = ['svg']

x = np.arange(-.5, 3., 0.005)
y = -x -np.exp(-x) + np.exp(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 20.11")
plt.rcParams["figure.figsize"] = [5, 5]

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

ax = plt.gca()
ratio = 4.65
#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, 15.])
ax.set_xticks([ -3, -2, -1, 0, 1, 2, 3])
#ax.set_yticks([0,1,2,3,4,5])
ax.set_xlim(-1,3)
ax.axhline(y=0, color='#993399', linewidth=0.7)
ax.axvline(x=0, color='#993399', linewidth=0.7)
```

```
endpoint = (-1-np.exp(-1)+np.exp(1))
plt.text(-0.8, 2.75, "SOLN: y = -x - e**(-x) + e**x", size=10,bbox=dict\
         (boxstyle="square", ec=('#1F77B4'),fc=(1., 1., 1),))
xpts = np.array([0, 1])
ypts = np.array([0, endpoint])
plt.plot(xpts, ypts, markersize=7, color='k', marker='s', \
         mfc='none', linestyle = 'none', markeredgewidth=0.5)
plt.ylim(-0.5,3.5)
plt.show()
```



As before. The function in the plot above looks continuous and well-behaved in the interval of interest.

20.14 Use Milne's method to solve $y'' - y = x; \quad y(0) = (0),\ y'(0) = 1$ on the interval [0, 1] with $h = 0.1$. Note: the only difference between this problem and the last is the numerical method employed. Since solving via Wolfram Alpha would be exactly the same, this problem will be skipped.

In [ ]:

In [ ]: