Solving ordinary differential equations

Using the **solve_ivp** function under scipy.integrate in the SciPy library.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

An ordinary differential equation of order n is:

$$F(x, y, y^0, y^1, \ldots, y^n) = 0,$$

where $y^k = \frac{d^k y}{dx^k}$. Alternatively, it can be written as: $y^n = g(x, y, y^0, y^1, \ldots, y^{n-1})$.

The solver basically solves the following: $\frac{dy}{dt} = f(t, y)$, given $y(t_0) = y_0$, where y is a function, either a scalar or a vector.

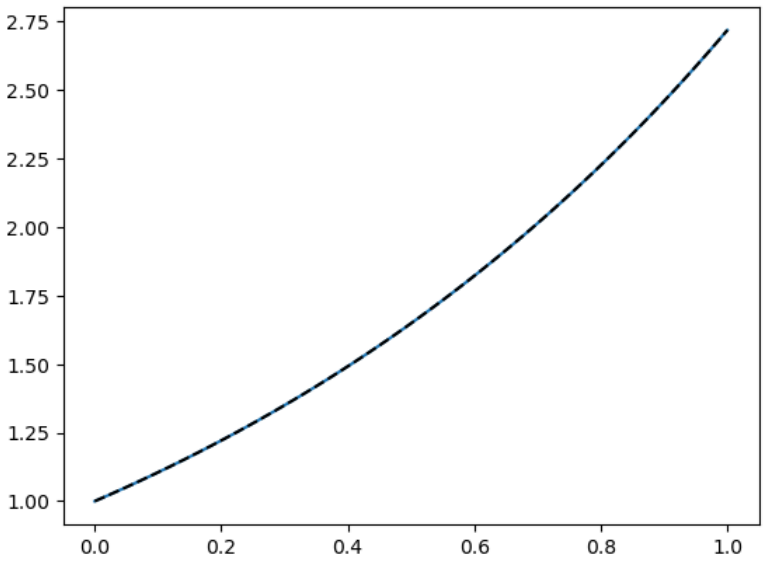
1) $\frac{dy}{dt} = y$

In [13]:
```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

def func(t,y):
    dydt = y
    return dydt

t = np.linspace(0,1,100)
y_out = solve_ivp(func, t_span = [0,1], y0 = np.array([1.0]), method='RK45', t_eval=t)

y_out # So y_out.t contains t, and y_out.y contains y(t). Careful of the containers.
```

Out[13]:
```
  message: The solver successfully reached the end of the integration interval.
  success: True
   status: 0
        t: [ 0.000e+00  1.010e-02 ...  9.899e-01  1.000e+00]
        y: [[ 1.000e+00  1.010e+00 ...  2.691e+00  2.718e+00]]
      sol: None
 t_events: None
 y_events: None
     nfev: 14
     njev: 0
      nlu: 0
```

In [11]:
```
plt.plot(y_out.t,y_out.y[0])
plt.plot(y_out.t,np.exp(y_out.t),'k--')
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x176c1fed0>]
```



2) $\frac{dy}{dt} = \sin(5t) + 0.1t$

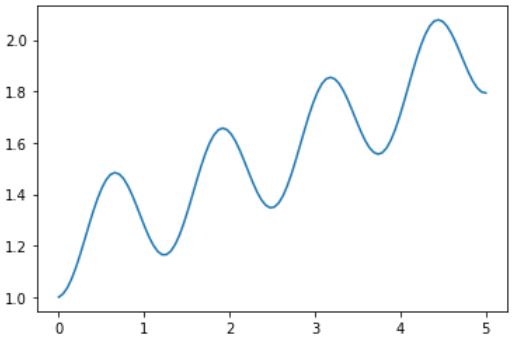
In [4]:
```
def func(t,y):
    dydt = np.sin(5*t)+0.1*y
    return dydt
```

```
t = np.linspace(0,5,100)
y_out = solve_ivp(func, t_span = [0,5], y0 = np.array([1.0]), method='LSODA', t_eval=t)
y_out
```

Out[4]:
```
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 115
     njev: 0
      nlu: 0
      sol: None
   status: 0
  success: True
        t: array([0.        , 0.05050505, 0.1010101 , 0.15151515, 0.2020202 ,
       0.25252525, 0.3030303 , 0.35353535, 0.4040404 , 0.45454545,
       0.50505051, 0.55555556, 0.60606061, 0.65656566, 0.70707071,
       0.75757576, 0.80808081, 0.85858586, 0.90909091, 0.95959596,
       1.01010101, 1.06060606, 1.11111111, 1.16161616, 1.21212121,
       1.26262626, 1.31313131, 1.36363636, 1.41414141, 1.46464646,
       1.51515152, 1.56565657, 1.61616162, 1.66666667, 1.71717172,
       1.76767677, 1.81818182, 1.86868687, 1.91919192, 1.96969697,
       2.02020202, 2.07070707, 2.12121212, 2.17171717, 2.22222222,
       2.27272727, 2.32323232, 2.37373737, 2.42424242, 2.47474747,
       2.52525253, 2.57575758, 2.62626263, 2.67676768, 2.72727273,
       2.77777778, 2.82828283, 2.87878788, 2.92929293, 2.97979798,
       3.03030303, 3.08080808, 3.13131313, 3.18181818, 3.23232323,
       3.28282828, 3.33333333, 3.38383838, 3.43434343, 3.48484848,
       3.53535354, 3.58585859, 3.63636364, 3.68686869, 3.73737374,
       3.78787879, 3.83838384, 3.88888889, 3.93939394, 3.98989899,
       4.04040404, 4.09090909, 4.14141414, 4.19191919, 4.24242424,
       4.29292929, 4.34343434, 4.39393939, 4.44444444, 4.49494949,
       4.54545455, 4.5959596 , 4.64646465, 4.6969697 , 4.74747475,
       4.7979798 , 4.84848485, 4.8989899 , 4.94949495, 5.        ])
t_events: None
        y: array([[1.        , 1.01290046, 1.03665266, 1.07165195, 1.11606229,
        1.16747389, 1.2229608 , 1.2795223 , 1.33389555, 1.3830944 ,
        1.42437222, 1.45620385, 1.47677143, 1.48407879, 1.4785399 ,
        1.46126544, 1.43363974, 1.39810658, 1.35692909, 1.31349487,
        1.27078566, 1.23194336, 1.2000104 , 1.17698142, 1.16491409,
        1.16573638, 1.17911078, 1.20496708, 1.24170659, 1.28778976,
        1.34052155, 1.39709974, 1.45440061, 1.50915162, 1.55856527,
        1.59998691, 1.63075183, 1.65011106, 1.65710599, 1.65093088,
        1.63334295, 1.60550425, 1.57000209, 1.52939707, 1.48669594,
        1.44523398, 1.40800027, 1.3778318 , 1.35729681, 1.34758734,
        1.35044365, 1.36640512, 1.3943792 , 1.43345832, 1.48124496,
        1.53564449, 1.5933784 , 1.65149   , 1.70677056, 1.7561582 ,
        1.79773777, 1.82932565, 1.84851821, 1.85433618, 1.84802884,
        1.8299276 , 1.80232877, 1.76712053, 1.72708209, 1.68552425,
        1.64538566, 1.60982131, 1.58183074, 1.56318311, 1.55575233,
        1.56146941, 1.57977854, 1.61028354, 1.65145933, 1.70120581,
        1.7572389 , 1.81622371, 1.8752636 , 1.93110863, 1.9807311 ,
        2.02231923, 2.05373785, 2.07263144, 2.07809154, 2.07154107,
        2.05334471, 2.02586809, 1.99118498, 1.95191296, 1.91153211,
        1.87299266, 1.83931713, 1.81350856, 1.79775357, 1.79365416]])
```

In [2]:
```
plt.plot(y_out.t,y_out.y[0])
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x7f1b019773d0>]
```



3) Solve the following ODE:

$$2\frac{dx}{dt} = -x(t) + u(t)$$

$$5\frac{dy}{dt} = -y(t) + x(t)$$

where, $u(t) = 2S(t-5)$, and $x(0) = y(0) = 0$.

$S(t-5)$ is a step function, $S = 0$ for $t < 5$, and $S = 1$ for $t \geq 5$.

In [18]:
```
def u(t):   # This bit is needed just for the plot
    if t < 5:
        return 0
    else:
        return 1

def func(t,x):
#     u = 1 if t >= 5 else 0
# #     y = np.zeros(len(x))
    x_1 = (1/2.)*(-x[0] + u(t))
    y_1 = (1/5.)*(-x[1] + x[0])
    return np.array([x_1,y_1])
```

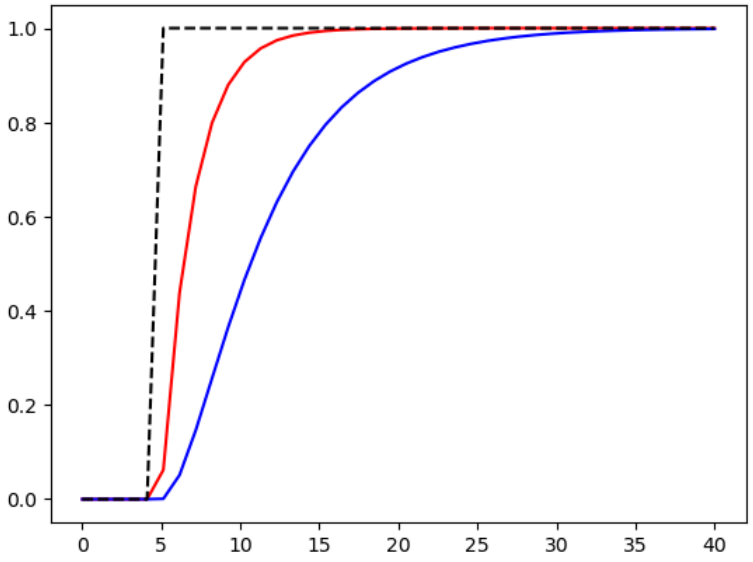
In [19]:
```
t = np.linspace(0,40,40)
x0 = np.array([0.0,0.0])
y_out = solve_ivp(func, t_span = [0,40], y0 = x0, method='BDF', t_eval=t)
```

In [20]:
```
print (y_out.y[1])
# print (y_out.y[1])
```

```
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 7.97028631e-04 5.12239939e-02 1.46344959e-01
 2.56256171e-01 3.65182681e-01 4.65510619e-01 5.54234783e-01
 6.30714893e-01 6.95559472e-01 7.49807016e-01 7.94814613e-01
 8.31990954e-01 8.62554974e-01 8.87661466e-01 9.08165877e-01
 9.25022039e-01 9.38751114e-01 9.50056525e-01 9.59259970e-01
 9.66829481e-01 9.73044619e-01 9.78089824e-01 9.82275972e-01
 9.85649575e-01 9.88371717e-01 9.90603401e-01 9.92354640e-01
 9.93797416e-01 9.94953205e-01 9.95868960e-01 9.96613742e-01
 9.97246635e-01 9.97745507e-01 9.98148545e-01 9.98501977e-01]
```

In [21]:
```
plt.plot(y_out.t,y_out.y[0],'r')
plt.plot(y_out.t,y_out.y[1],'b')
plt.plot(t,[u(i) for i in t],'k--')
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x176cc3390>]
```



4) Solve the following ODE:

$$\frac{dx_1}{dt} = x_2(t)$$

$$\frac{dx_2}{dt} = 2x_1(t) - x_2(t)$$

$$x_1(0) = 1;\ x_2(0) = 2.$$

which can be written as matrix multiplication:

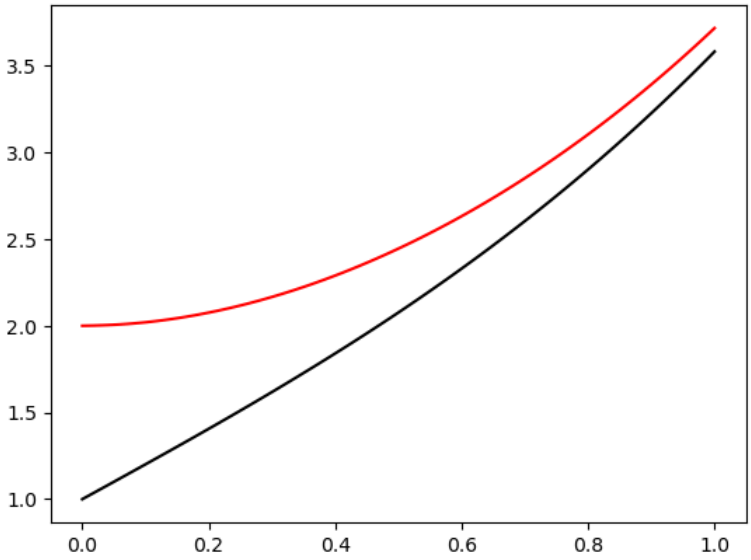
$$\frac{d\mathbf{x}}{dt} = A \times \mathbf{x},\ \text{where } A = \begin{pmatrix} 0 & 1 \\ 2 & -1 \end{pmatrix},\ \text{and } \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.\ \text{The initial condition is } \mathbf{x}(0) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

In [29]:
```
A = np.array([[0,1],[2,-1]])

def func(t,x):
    x1 = A@x
    return x1

t = np.linspace(0,1,100)
```

```
x0 = np.array([1.0,2.0])
y_out = solve_ivp(func, t_span = [0,5], y0 = x0, method='LSODA', t_eval=t)
```

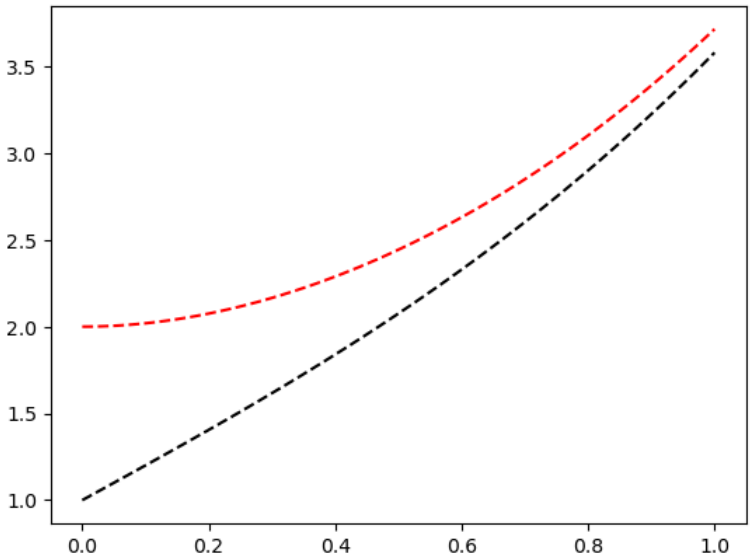
In [45]:
```
plt.plot(y_out.t,y_out.y[0],'k')
plt.plot(y_out.t,y_out.y[1],'r')
```

Out[45]: [<matplotlib.lines.Line2D at 0x177822850>]



In [46]:
```
from scipy import linalg as la

def mat_sol(x,t):
    return la.expm(A*t)@x

x_sol = [mat_sol(x0,t0) for t0 in t]

# plt.plot(t,x_sol[0])
plt.plot(t,np.transpose(x_sol)[0],'k--')
plt.plot(t,np.transpose(x_sol)[1],'r--')
```

Out[46]: [<matplotlib.lines.Line2D at 0x17787c7d0>]



Tasks for today

Solve the following problems.

- Consider a one-dimensional simple harmonic oscillator with m = 1 and $k = 1/2$. Use a ODE solver to find the displacement of the oscillator.

  Using matplotlib, plot the displacement x with time t, for $t \in [0, 50]$.

The SHO equation is given by:

$$m\frac{dx^2}{dt^2} = -kx.$$

Now, let $dx/dt = v$, such that we have,

$$\frac{dx}{dt} = v; \frac{dv}{dt} = -\frac{k}{m}x. \tag{1}$$

- Solve the differential equation:

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = x(\rho - z) - y$$

$$\frac{dz}{dt} = xy - \beta z$$

where $\sigma, \beta, \rho = 10, 4, 25$, respectively. You can take the time, $t \in [0, 40]$. Plot x vs y, x vs y, and z vs y.

- Convert the second order differential equation to a set of ODEs and solve: $\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + ty = \cos(t)$,

  where $\frac{dy}{dt} = y'(0) = -2$ and $y(0) = 2$. Plot the solutions of the ODEs for some time range.

- Consider the matrix equation:

$$\frac{d\mathbf{x}}{dt} = A\mathbf{x}, \text{ where } A = \begin{pmatrix} 0 & 1 & -2 \\ 1 & 3 & -1 \\ -2 & -1 & -2 \end{pmatrix}$$

Solve the above equation using **solve_ivp**, and compare the solutions with $\mathbf{x}(t) = e^{At}\mathbf{x}(0)$, where $\mathbf{x}(0) = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$.

- The trajectory of a cannon shell, fired from a point in ground, is given by a set of equations:

$$\frac{dx}{dt} = v_x$$
$$\frac{dy}{dt} = v_y$$
$$\frac{dv_x}{dt} = -bvv_x$$
$$\frac{dv_y}{dt} = -g - bvv_y$$

  where b is a drag constant, $g = 9.8\ \mathrm{m/s}$ and $v = \sqrt{v_x^2 + v_y^2}$ is the velocity of the projection. Choose $b = 0.0011\ \mathrm{m/s}$ and the initial launch velocity $v_0 = 100\ \mathrm{m/s}$. Plot the x V/s y trajectories for various equally spaced launch angles in the same frame. Note that the trajectory starts with the launch of cannon ball from origin and ends when cannon ball hits the ground. . Plot the range as a function of launch angles. Among these launch angles, find the angle that has the greatest range.

  Hint 1: While solving the ODE, take the time span to greater than $20s$ (Why?)

  Hint 2: The solution will give the parts where y<0. You don't need this data (Why?). Write a function to filter out y<0.