#1

(14.20)

In [10]:

```
import numpy as np
import pylab

def strlinregr(x,y):  #x and y are arrays of data
    '''Given x and y observations function returns best-fit
    straight line parameters
    , Rsq, and SE
    Inputs: paired x and y values
    Outputs: a0 = line intercept
        a1 = line slope
        Rsq = r squared
        SE = standard error'''
    if len(x) != len(y):
        return "x and y must be of the same length"
    n = len(x)          #Number of values
    sumx = np.sum(x)    #Sum of X
    xbar = sumx/n       #Average of x
    sumy = np.sum(y)
    ybar = sumy/n
    sumsqx = 0         #Placeholder for sum of x^2
    sumxy = 0         #Placeholder for sum of each x and y value
    for i in range(n):
        sumsqx = sumsqx + x[i]**2 #Adding each element in x list squared
        sumxy = sumxy + x[i]*y[i]
    a1 = (n*sumxy-sumx*sumy)/(n*sumsqx-sumx**2)  #Equation for slope
    a0 = ybar - a1*xbar    #Equation for intercept
    e = np.zeros((n))#Creating an empty matrix of length n to hold error
    SST = 0                    #Total sum of square error
    SSE = 0
    for i in range (n):
        e[i] = y[i] - (a0+a1*x[i])  #Difference between obs. & estimate
        SST = SST + (y[i] - ybar)**2
        SSE = SSE + e[i]**2
    SSR = SST - SSE
    Rsq = SSR/SST
    SE = np.sqrt(SSE/(n-2))
    return a0, a1, Rsq, SE
```

1a

In [11]:

```
#Create arrays of the data
time = np.array([10., 15., 20., 25., 40., 50., 55., 60., 75.])
TS = np.array([5., 20., 18., 40., 33., 54., 70., 60., 78.])

#Use strlinregr function and assign output to variables
a0, a1, R, SE = strlinregr(time, TS)

#Print the results
print("Intercept =", a0, "\nSlope =", a1, "\nR-squared =", R,"\nSE =", SE)

#Print the estimate at 32minutes
print("Predicted tensile strength at 32 min =", a0 + a1*32, "MPa")
```

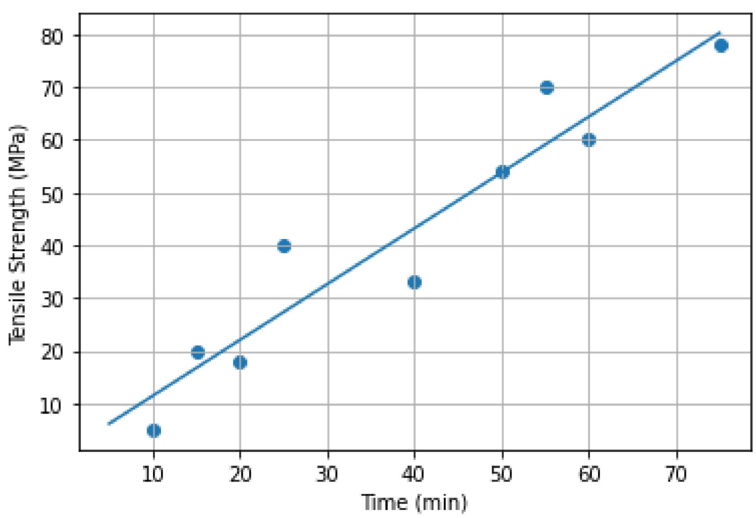
```
Intercept = 0.8179347826087024
Slope = 1.0589673913043478
R-squared = 0.9058334263824231
SE = 8.252023090925503
Predicted tensile strength at 32 min = 34.70489130434783 MPa
```

1b

In [12]:

```
#Plots of the data and best-fit regression line
x = np.linspace(5,75,2)  #You only need two values for a straight line
y = a0 + a1*x
pylab.plot(x,y)
pylab.scatter (time,TS)
pylab.grid()
pylab.xlabel("Time (min)")
pylab.ylabel("Tensile Strength (MPa)")
```

Out[12]: Text(0, 0.5, 'Tensile Strength (MPa)')

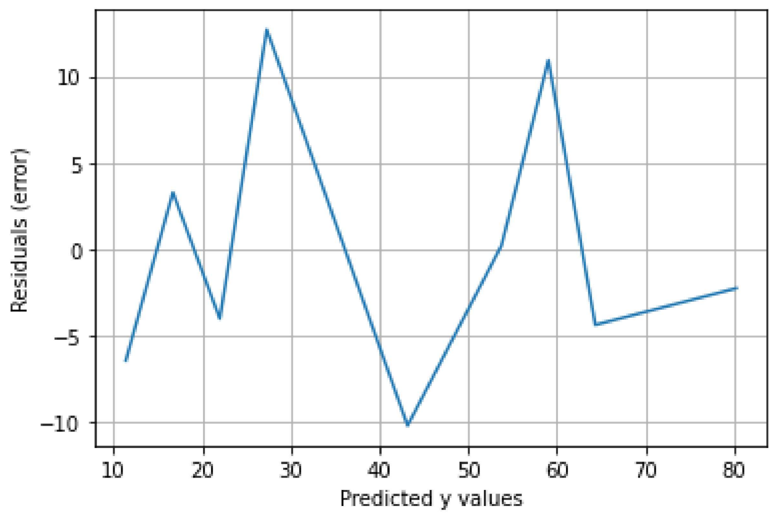


1c

In [13]:
```python
n = len(time)
ypred = np.zeros((n))
e = np.zeros((n))
for i in range (n):
    ypred[i] = a0+a1*time[i]  #Create an array of predicted values
    e[i] = TS[i] - (a0+a1*time[i]) #Calc difference in measured & predicted

pylab.plot(ypred, e)
pylab.grid()
pylab.xlabel("Predicted y values")
pylab.ylabel("Residuals (error)")
```

Out[13]: Text(0, 0.5, 'Residuals (error)')



1d. The residual plot shows no systematic behavior and the the model is likely an adequate model for this data.

#2

(14.9)

2a

In [14]:
```python
import pylab
import numpy as np

t = np.array([4., 8., 12., 16., 20., 24.])
c = np.array([1600, 1320, 1000, 890, 650, 560])
```

Proposed exponential model: c = a1*e^B1*t Linearized model: ln(c) = ln(a1)+ B1*t

Therefore, y = ln(c) x = t slope = B1 intercept = ln(a1)

Use straight-line regression to solve

In [15]:
```
#Assign outputs to my variables
intercept, slope, R, SE = strlinregr(t, np.log(c))
#Intercept, slope, Rsq, SE
```

Backtransform the model

In [16]:
```
import math

# intercept = ln(a1), so a1 = e^ln(a1)
a1 = math.exp(intercept)
print("The a1 parameter is:",a1)
B1 = slope
print("The B1 parameter is:", B1)
```

```
The a1 parameter is: 1985.4366459562054
The B1 parameter is: -0.05350634569158223
```

2b. Define backtransformed model as a function Estimate the concentration at t= 0 and Solve for t when c equal 200CFU/mL

In [17]:
```
#Define Function
import math

def EColiConc(hr):  #Exponential model with back transformed parameters
    conc = a1*np.exp(B1*hr)
    return conc

#Print Concentratin at t=0
print("At time=0, the E. Coli Concentration was",EColiConc(0))

#Print the time when concentration will be 200 CFU/mL
print("The concentration will reach 200CFU/mL at time ="\
      , (math.log(200/a1))/B1 )
```

```
At time=0, the E. Coli Concentration was 1985.4366459562054
The concentration will reach 200CFU/mL at time = 42.897281537165306
```

2c

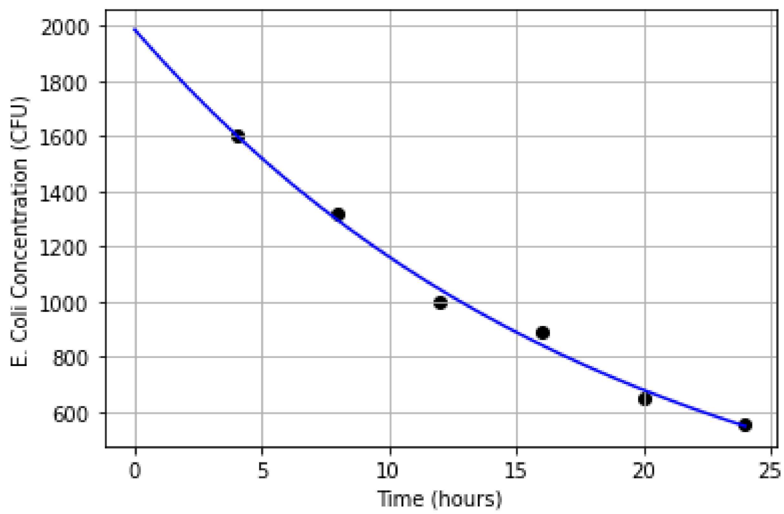
In [18]:

```
import numpy as np
import pylab

x = np.linspace(0,24,24)  #X-values (time)
pylab.plot(x,EColiConc(x), c = "b") #Plot x(time) vs. y(conc) model
pylab.scatter (t,c, c= "k")
pylab.grid()
pylab.xlabel("Time (hours)")
pylab.ylabel("E. Coli Concentration (CFU)")
```

Out[18]:

```
Text(0, 0.5, 'E. Coli Concentration (CFU)')
```



#3

(15.6)

In excel spreadsheet, calculate sums of T, C, DO, T^2, C^2, T*c, T*DO, c*DO Use these to populate the normal equations (found in lecture 11) for multivariate models

In [19]:
```python
#Normal equations  (the values will come from your spreadsheet)

A = np.array([[21,315,210],[315, 6825, 3150], [210, 3150,3500]])
b = np.array([[198.54],[2555.5],[1838.5]])

#Print them so I can see I entered correctly
print(A)
print(b)
```

```
[[  21  315  210]
 [ 315 6825 3150]
 [ 210 3150 3500]]
[[ 198.54]
 [2555.5 ]
 [1838.5 ]]
```

In [20]:
```python
x = np.linalg.solve(A,b)

#Print and assign to parameters
print("The model parameters are:\nb0 =",x[0],"\nb1 =", x[1], "\nb2 =", x[2])
```

```
The model parameters are:
b0 = [13.52214286]
b1 = [-0.2012381]
b2 = [-0.10492857]
```

3a.

Our resulting model is therefore: DO = 13.522 - 0.2012*T* - *0.1049*C

3b.

In [21]:
```python
#Predict DO at T = 12 and C = 15
#(There was an error in the given data should have all been in Celcius -
 #I'll accept either value, Celcius or Farenheit, both answers are provided)
t = 12 #C
tF = 53.6  #F
c = 15 #g/L
DO = x[0] + x[1]*t + x[2]*c  #Our regression
DOF = x[0] + x[1]*tF + x[2]*c  #Our regression using Farenheit

print("The DO at temp=", t,"(C)and c =",c,"(g/L) is", DO,"(mg/L)")
print("The DO at temp=", tF,"(F)and c =",c,"(g/L) is", DOF,"(mg/L)")
```

```
The DO at temp= 12 (C)and c = 15 (g/L) is [9.53335714] (mg/L)
The DO at temp= 53.6 (F)and c = 15 (g/L) is [1.16185238] (mg/L)
```

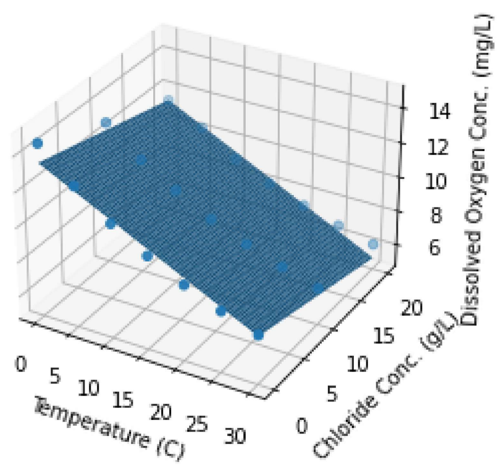
3c.

In [22]:

```python
#Plot 3D response surface
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
t = np.array([0,5,10,15,20,25,30,0,5,10,15,20,25,30,\
              0,5,10,15,20,25,30]) #Temp (x-values)
c = np.array([0,0,0,0,0,0,0,10,10,10,10,10,10,10,20,20,20,20,20,20,20])
DO = np.array([14.6,12.8,11.3,10.1,9.09,8.26,7.56,12.9,11.3,10.1,9.03,\
               8.17,7.46,6.85,11.4,10.3,8.96,8.08,7.35,6.73,6.2])
x  = np.linspace(0.,30,200)  #For the regression plot
y = np.linspace(0.,20, 200)# For the regression plot
x,y = np.meshgrid(x,y) #Creates a grid of values to populate regress. model
z = 13.522 - 0.2012*x - 0.1049*y #Model Equation
ax.scatter(t, c, DO)  #Experimental Data points
ax.plot_surface(x,y,z)  #Regression
ax.set_xlabel('Temperature (C)')
ax.set_ylabel('Chloride Conc. (g/L)')
ax.set_zlabel('Dissolved Oxygen Conc. (mg/L)')

plt.show()
```



#4

(15.15)

4a.

In [23]:

```
import numpy as np
import pylab

#Enter the data
x = np.array([5,10,15,20,25,30,35,40,45,50])
y = np.array([17,24,31,33,37,37,40,40,42,41])

#Straight-line model    (y = a0 +a1*x)
a = np.polyfit(x,y,1)  #part a parameters
print("Straight-line model parameters:\na1 =",a[0],"\na0 =",a[1],"\n")

#Quadratic Equation Model (2nd order polynomial) (y = b0 + b1*x +b2*x^2)
b = np.polyfit(x,y,2)
print("2nd order polynomial parameters:\nb2 =",b[0]\
      ,"\nb1=",b[1],"\nb0=",b[2])
```

```
Straight-line model parameters:
a1 = 0.4945454545454543
a0 = 20.600000000000005

2nd order polynomial parameters:
b2 = -0.016060606060602
b1= 1.3778787878787864
b0= 11.766666666666683
```

4b.

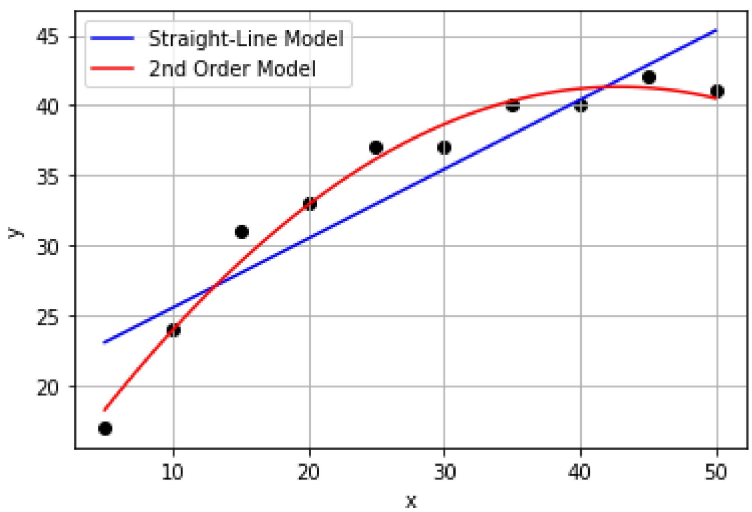
In [24]:

```
#Plot the data and models

import pylab

x1 = np.linspace(5,50,100)
pylab.plot(x1, np.polyval(a,x1), c = 'b', label = "Straight-Line Model")
pylab.plot(x1, np.polyval(b,x1), c = "r", label = "2nd Order Model")
pylab.scatter(x,y, c = 'k')  #our data points
pylab.grid()
pylab.xlabel("x")
pylab.ylabel("y")
pylab.legend()
```

Out[24]: <matplotlib.legend.Legend at 0x1acd7456d00>



#5

(17.6/17.6)

In [25]:

```python
def Lagrange (x,y,xx):
    """Lagrange Interpolating Polynomial
    Uses n-1 order lagrange interpolating polynomial based
    on n data pairs to return a value of the dependent variable, yint
    given an independent variable, xx.
    Input: x = array of indepdent variable values
        y = array of dependent variables
        xx = independent variable at which to interpolate
    Output: yint = interpolated dependent variable value"""
    n = len(x)
    if len(y) != n:
        return "x and y must be same length"
    s = 0
    for i in range (n):
        product = y[i]
        for j in range(n):
            if i != j:  #See general equation from slides
                product = product*(xx - x[j])/(x[i]-x[j])
        s = s + product
    yint = s
    return yint
```

In [26]:

```python
#Part a.)
import numpy as np
#Fourth Order
x4 = np.array([1,2,3,5,6])
y4 = np.array([4.75,4,5.25,19.75,36])

f4order4 = Lagrange(x4,y4,4)
print("F(4) for fourth order =",f4order4)

#Third Order
x = np.array([2,3,5,6])
y = np.array([4,5.25,19.75,36])

f4order3 = Lagrange(x,y,4)
print("F(4) for third order =",f4order3)


#Second Order
x = np.array([2,3,5])
y = np.array([4,5.25,19.75])

f4order2 = Lagrange(x,y,4)
print("F(4) for second order =",f4order2)

#First Order  (i.e. straight-line)
x = np.array([3,5])
y = np.array([5.25,19.75])

f4order1 = Lagrange(x,y,4)
print("F(4) for first order =",f4order1)
```

```
F(4) for fourth order = 10.0
F(4) for third order = 10.0
F(4) for second order = 10.5
F(4) for first order = 12.5
```

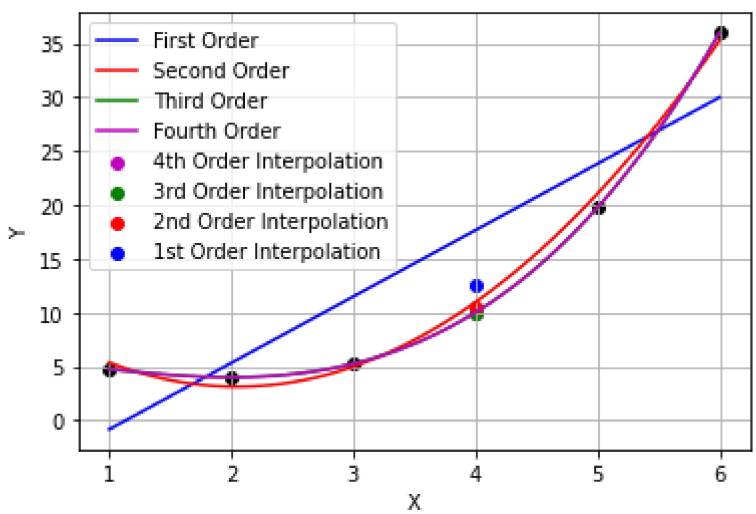
In [27]:
```python
#Part b.)

x = np.linspace(1,6,100)  #values for plotting regressions

#Generate 1-4 order y values
a = np.polyfit(x4,y4,1) #First order regression
b = np.polyfit(x4,y4,2) #Second order regression
c = np.polyfit(x4,y4,3) #Third order regression
d = np.polyfit(x4,y4,4) #Fourth order regression

import pylab
pylab.plot(x, np.polyval(a,x), c = 'b', label = "First Order")
pylab.plot(x, np.polyval(b,x), c = 'r', label = "Second Order")
pylab.plot(x, np.polyval(c,x), c = 'g', label = "Third Order")
pylab.plot(x, np.polyval(d,x), c = 'm', label = "Fourth Order")
pylab.scatter(4,f4order4, c= "m", label = "4th Order Interpolation")
pylab.scatter(4,f4order3, c= "g", label = "3rd Order Interpolation")
pylab.scatter(4,f4order2, c= "r", label = "2nd Order Interpolation")
pylab.scatter(4,f4order1, c= "b", label = "1st Order Interpolation")
pylab.scatter(x4,y4, c = 'k') #Original datapoints
pylab.grid()
pylab.xlabel("X")
pylab.ylabel("Y")
pylab.legend()
```

Out[27]: <matplotlib.legend.Legend at 0x1acd75358e0>



Part c.) The regression and interpolation values do not match up. Regressions minimize error, interpolations force the function to connect each point (i.e. have 'zero' error). For precise and accurate data points having zero error makes sense, but for data that has noise this can result in an 'overfit' model. Regressions specialize is smoothing out error so you can see the overall trend, interpolations assume there is no noise/error in the data and so if there actually is error it will add that into its predictions (which can throw it off). In this case we can see that the 3rd and 4th order interpolations and regressions are likely the best estimate of the value since they best model the given data. Since the interpolations look to be accurate the data provided is likely fairly precise.

In [ ]:

```

```

In [ ]:

```

```