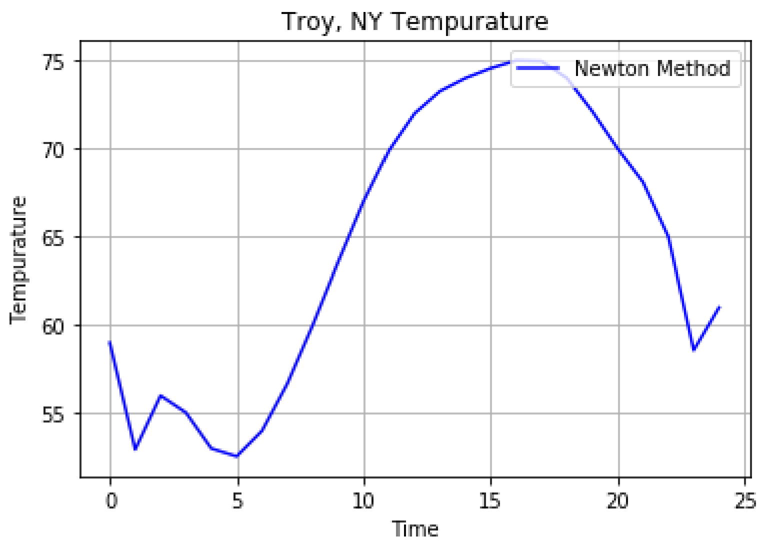
In [1]:
```
import numpy as np
from matplotlib import pyplot as plt

x = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]

y = [59, 56, 53, 54, 60, 67, 72, 74, 75, 74, 70, 65, 61]
```

In [2]:
```
def newton_val(x, y):

    points = len(x)

    counter  = 1
    coeffs = [y[0]]
    iter_yvals = y

    while counter < points:     #Coefficients

        iterdata = []

        for i in range(len(iter_yvals)-1):

            change_y = iter_yvals[i+1]-iter_yvals[i]
            change_x = x[i+counter]-x[i]
            iterval = (change_y/change_x)
            iterdata.append(iterval)

            if i==0:
                coeffs.append(iterval)

        iter_yvals = iterdata
        counter+=1

    def value(i):                #Values

        terms  = []
        retval = 0

        for j in range(len(coeffs)):

            iterval = coeffs[j]
            iterxvals = x[:j]
            for k in iterxvals:
                iterval = iterval * (i-k)
            terms.append(iterval)
            retval+=iterval
        return(retval)
    return(value)
```

In [3]:
```
newton_eval = newton_val(x, y)
```

In [4]:
```
newton_x = []
newton_y = []
for i in np.linspace(0, 24, 25):
    newton_x.append(i)
    newton_y.append(newton_eval(i))
```

In [5]:
```
plt.plot(newton_x, newton_y, 'b-', label='Newton Method')
plt.xlabel('Time')
plt.ylabel('Tempurature')
plt.grid()
plt.title('Troy, NY Tempurature')
plt.legend(loc='upper right')
plt.show()
```



In [ ]:

```
#https://medium.com/eatpredlove/natural-cubic-splines-implementation-with-python-edf68feb57aa

import pandas as pd
import numpy as np
def jacobi(A, b, x0, tol, n_iterations=300):
    """
    Performs Jacobi iterations to solve the line system of
    equations, Ax=b, starting from an initial guess, ``x0``.

    Returns:
    x, the estimated solution
    """

    n = A.shape[0]
    x = x0.copy()
    x_prev = x0.copy()
    counter = 0
    x_diff = tol+1

    while (x_diff > tol) and (counter < n_iterations): #iteration level
        for i in range(0, n): #element wise level for x
            s = 0
            for j in range(0,n): #summation for i !=j
                if i != j:
                    s += A[i,j] * x_prev[j]

            x[i] = (b[i] - s) / A[i,i]
        #update values
        counter += 1
        x_diff = (np.sum((x-x_prev)**2))**0.5
        x_prev = x.copy() #use new x for next iteration


    print("Number of Iterations: ", counter)
    print("Norm of Difference: ", x_diff)
    return x


def cubic_spline(x, y, tol = 1e-100):
    """
    Interpolate using natural cubic splines.
```

```
    Generates a strictly diagonal dominant matrix then applies Jacobi's method.

    Returns coefficients:
    b, coefficient of x of degree 1
    c, coefficient of x of degree 2
    d, coefficient of x of degree 3
    """
    x = np.array(x)
    y = np.array(y)
    ### check if sorted
    if np.any(np.diff(x) < 0):
        idx = np.argsort(x)
        x = x[idx]
        y = y[idx]

    size = len(x)
    delta_x = np.diff(x)
    delta_y = np.diff(y)

    ### Get matrix A
    A = np.zeros(shape = (size,size))
    b = np.zeros(shape=(size,1))
    A[0,0] = 1
    A[-1,-1] = 1

    for i in range(1,size-1):
        A[i, i-1] = delta_x[i-1]
        A[i, i+1] = delta_x[i]
        A[i,i] = 2*(delta_x[i-1]+delta_x[i])
    ### Get matrix b
        b[i,0] = 3*(delta_y[i]/delta_x[i] - delta_y[i-1]/delta_x[i-1])

    ### Solves for c in Ac = b
    print('Jacobi Method Output:')
    c = jacobi(A, b, np.zeros(len(A)), tol = tol, n_iterations=1000)

    ### Solves for d and b
    d = np.zeros(shape = (size-1,1))
    b = np.zeros(shape = (size-1,1))
    for i in range(0,len(d)):
        d[i] = (c[i+1] - c[i]) / (3*delta_x[i])
        b[i] = (delta_y[i]/delta_x[i]) - (delta_x[i]/3)*(2*c[i] + c[i+1])
```

```python
    return b.squeeze(), c.squeeze(), d.squeeze()
```

In [6]:

```python
#https://stackoverflow.com/questions/43458414/python-scipy-how-to-get-cubic-spline-equations-from-cubicspline
```

In [110]:

```
#Note: S(x) = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 for x_j <= x <= x_j+1
#INPUT n; x_0, x_1, ..., x_n; a_0 = f(x_0), a_1 = f(x_n)
def nat_cubic_spline(x, y):
    n= len(x)
    h = [None] * n
    a = y
    l = [None] * n
    u = [None] * n
    z = [None] * n
    c = [None] * n
    b = [None] * n
    d = [None] * n
#STEP1 For i = 0, 1, ..., n - 1 set h_i = x_i+1 - x_i
    for i in range(0, n - 1):
        h[i] = x[i + 1] - x[i]
#STEP2 For i = 1 2, ..., n - 1 set
    #a_i = (3/h_i)(a_i+1 - a_i) - (3/h_i-1)(a_i - a_i-1)
    for i in range (1, n - 1):
        a[i] = (3/h[i]) * (a[i]+1 - a[i]) - (3/h[i-1]) * (a[i] - a[i-1])
#STEP3 Set l_0 = 1;
    # u_0 = 0;
    # z_0 = 0;
    l[0] = 1
    u[0] = 0
    z[0] = 0
#STEP4 For i = 1, 2, ..., n - 1
    #Set l_i = 2(x_i+1 - x_i-1) - h_i-1 * u_i-1
    #u_i = h_i / l_i;
    #z_i = (a_i - h_i-1 * z_i-1)/l_i
    i = 1
    while i < (n - 1):
        l[i] = 2 * (x[i+1] - x[i-1]) - (h[i-1] * u[i-1])
        u[i] = h[i] / l[i]
        z[i] = (a[i] - h[i-1] * z[i-1])/l[i]
        i = i + 1
#STEP5 Set l_n = 1;
    #z_n = 0;
    #c_n = 0
    l.append(1)
    z.append(0)
    c.append(0)
#STEP6 For j = n-1, n-2, ..., 0
```

```
    # set c_j = z_j - u_j * c_j+1;
    # b_j = (a_j+1 - a_j)/h_j - h_j(c_j+1 + 2c_j)/3;
    # d_j = (c_j+1 - c_j)/(3h_j).
    j = n - 1
    while j > 0:
        c[j] = z[j] - u[j] * c[j+1]
        b[j] = (a[j+1] - a[j])/h[j] - h[j]*(c[j+1] + 2*c[j])/3
        d[j] = (c[j+1] - c[j])/(3*h[j])
        j = j - 1
#STEP7 OUTPUT (a_j, b_j, c_j, d_j for j = 0, 1, ..., n-1) STOP
    for j in range(0, n-1):
        return a[j]
        return b[j]
        return c[j]
        return d[j]
```

In [111]:
```
nat_cubic_spline(x, y)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-111-f877a3cd095d> in <module>()
----> 1 nat_cubic_spline(x, y)

<ipython-input-110-15316ca56a48> in nat_cubic_spline(x, y)
     47     j = n - 1
     48     while j > 0:
---> 49         c[j] = z[j] - u[j] * c[j+1]
     50         b[j] = (a[j+1] - a[j])/h[j] - h[j]*(c[j+1] + 2*c[j])/3
     51         d[j] = (c[j+1] - c[j])/(3*h[j])

TypeError: unsupported operand type(s) for *: 'NoneType' and 'int'
```

In [154]:
```
n= len(x)
h = [None] * n
a = y
l = [None] * n
u = [None] * n
z = [None] * n
c = [None] * n
b = [None] * n
d = [None] * n
for i in range(0, n - 1):
    h[i] = x[i + 1] - x[i]
for i in range (1, n):
    a[i] = (3/h[i]) * (a[i]+1 - a[i]) - (3/h[i-1]) * (a[i] - a[i-1])
l[0] = 1
u[0] = 0
z[0] = 0
i = 1
for i in range (1, n):
    l[i] = 2 * (x[i+1] - x[i-1]) - (h[i-1] * u[i-1])
    u[i] = h[i] / l[i]
    z[i] = (a[i] - h[i-1] * z[i-1])/l[i]
l[n] = 1
z.append(0)
c.append(0)
j = n - 1
while j > 0:
    c[j] = z[j] - u[j] * c[j+1]
    b[j] = (a[j+1] - a[j])/h[j] - h[j]*(c[j+1] + 2*c[j])/3
    d[j] = (c[j+1] - c[j])/(3*h[j])
    j = j - 1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-154-fed457aef6df> in <module>()
     11     h[i] = x[i + 1] - x[i]
     12 for i in range (1, n):
---> 13     a[i] = (3/h[i]) * (a[i]+1 - a[i]) - (3/h[i-1]) * (a[i] - a[i-1])
     14 l[0] = 1
     15 u[0] = 0

TypeError: unsupported operand type(s) for /: 'int' and 'NoneType'
```