In [28]:

```
#1. Taylor series sine

from math import *

def fprime(x,k):
    if k == 0:
        return sin(x)
    if k == 1:
        return cos(x)
    if k == 2:
        return -sin(x)
    if k == 3:
        return -cos(x)
#derivatives of sine



def main():
    #Looping until you reach the stopping criteria
    x0 = pi/12                      #initial x point value
    x = pi/2                        #where we want cos(x) evaluate
    h = x - x0                      #Stepsize of a function
    fapprx = 0.0                    #this is the function value we are calculating
    ftrue = sin(x)                  #set ftrue here -- it will not change
    err_stop = 1.0e-6               #this is what is called the stopping criterion
    true_err = 1.1*err_stop         #initially make sure rel_err is defined to be more than the err_stop
    max_iter = 100                  #set a max number of iterations
    for i in range(0,max_iter):     #for loop that will execute max_iter times unless there is a 'break'
        k = (i+4)%4                 #this gives us an index we can use to figure out fprime
        #Taylor series = sum f^i(x0)*h^i/i!
        fapprx+=fprime(x0,k)*h**i/factorial(i)
        true_err = abs((ftrue-fapprx)/ftrue)      #calc true_err
        if true_err <= err_stop:                  #is rel_err less than the err_stop
            print(f"i={i+1} \t f_{i+1}={fapprx} \t\t\t true_err={true_err}")
            break                                 #if it is less then stop iterating
        print(f"i={i+1} \t f_{i+1}={fapprx} \t\t\t true_err={true_err}")


if __name__ == '__main__':
    main()
```

```
i=1      f_1=0.25881904510252074                         true_err=0.7411809548974793
i=2      f_2=1.523212995011849                   true_err=0.5232129950118489
i=3      f_3=1.301473273950253                   true_err=0.30147327395025303
i=4      f_4=0.9403891277484051                          true_err=0.05961087225159489
i=5      f_5=0.9720512129174713                          true_err=0.02794878708252868
i=6      f_6=1.002986609432379                   true_err=0.0029866094323789394
i=7      f_7=1.0011782051781413                          true_err=0.0011782051781412672
i=8      f_8=0.9999161345531703                          true_err=8.386544682970065e-05
i=9      f_9=0.9999714676217051                          true_err=2.8532378294920946e-05
i=10     f_10=1.0000015026761868                         true_err=1.5026761868153216e-06
i=11     f_11=1.0000004492126515                         true_err=4.4921265152098044e-07
```

In [30]:

```python
#2. Taylor Series exponential

from math import *

def fprime(x,k):
    if k == 0:
        return exp(x)
    if k == 1:
        return exp(x)
    if k == 2:
        return exp(x)
    if k == 3:
        return exp(x)
#derivatives of sine



def main():
    #Looping until you reach the stopping criteria
    x0 = 0.0                        #initial x point value
    x = 4.0                         #where we want exp(x) evaluate
    h = x - x0                      #Stepsize of a function
    fapprx = 0.0                    #this is the function value we are calculating
    ftrue = exp(x)                  #set ftrue here -- it will not change
    err_stop = 1.0e-7               #this is what is called the stopping criterion
    true_err = 1.1*err_stop         #initially make sure rel_err is defined to be more than the err_stop
    max_iter = 100                  #set a max number of iterations
    for i in range(0,max_iter):     #for loop that will execute max_iter times unless there is a 'break'
        k = (i+4)%4                 #this gives us an index we can use to figure out fprime
        #Taylor series = sum f^i(x0)*h^i/i!
        fapprx+=fprime(x0,k)*h**i/factorial(i)
        true_err = abs((ftrue-fapprx)/ftrue)      #calc true_err
        if true_err <= err_stop:                  #is rel_err less than the err_stop
            print(f"i={i+1} \t f_{i+1}={fapprx} \t\t\t true_err={true_err}")
            break                                 #if it is less then stop iterating
        print(f"i={i+1} \t f_{i+1}={fapprx} \t\t\t true_err={true_err}")


if __name__ == '__main__':
    main()
```

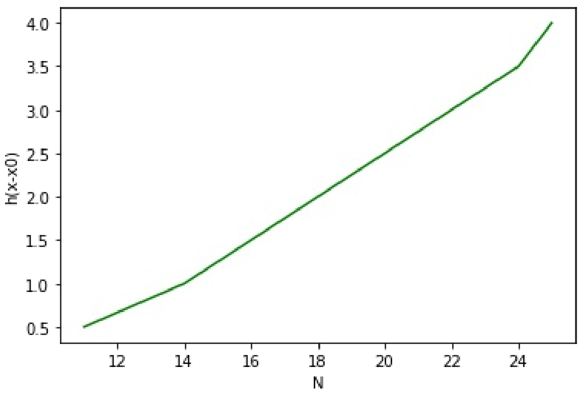
```
i=1      f_1=1.0                        true_err=0.9816843611112658
i=2      f_2=5.0                        true_err=0.9084218055563291
i=3      f_3=13.0                       true_err=0.7618966944464557
i=4      f_4=23.666666666666664                         true_err=0.566529879633291
i=5      f_5=34.33333333333333                  true_err=0.3711630648201265
i=6      f_6=42.86666666666666                  true_err=0.2148696129695949
i=7      f_7=48.55555555555555                  true_err=0.11067397840257374
i=8      f_8=51.8063492063492                   true_err=0.051133615792847344
i=9      f_9=53.43174603174603                  true_err=0.021363434487984l4
i=10     f_10=54.15414462081129                         true_err=0.008132242796933814
i=11     f_11=54.44310405643739                         true_err=0.0028397661205136573
i=12     f_12=54.54818021484688                         true_err=0.000915229147270035
i=13     f_13=54.583205600983376                        true_err=0.00027371682285549416
i=14     f_14=54.59398264287153                         true_err=7.632841534329772e-05
i=15     f_15=54.59706179769672                         true_err=1.993172748267018e-05
i=16     f_16=54.5978829056501                  true_err=4.892610719801462e-06
i=17     f_17=54.59808818263845                         true_err=1.1328315290842824e-06
i=18     f_18=54.598136483106295                        true_err=2.4817760186436607e-07
i=19     f_19=54.598147216543595                        true_err=5.158784024548016e-08
```

In [ ]:

In [12]:

```
#3. Taylor Series exponential
%matplotlib inline
from math import *
#Creating taylor series for the function e^x
def taylor_exp(x):
    fapprx = 0.0
    ftrue = exp(x)
    n = 0
    while ftrue - fapprx > 1.0e-10: # stopping criterion
        fapprx += (x**n)/factorial(n)
        n +=1
    return n

list_of_x0 = [0.0, 0.5, 1.5, 2.0, 2.5, 3.0, 3.5]
x = 4.0
print(f" There are {taylor_exp(x)} iterations.")

# calling function to know number of terms for each x-x0 pair
list_of_terms = [taylor_exp(x-i) for i in list_of_x0]
list_of_h = [x-i for i in list_of_x0]

plt.plot(list_of_terms, list_of_h ,'g')                 # making a plot of N vs h

# Labelling x and y axes and showing the graph
plt.xlabel("N")
plt.ylabel("h(x-x0)")
plt.show()
```

```
 There are 25 iterations.
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```