hw8_code

October 29, 2021

Michael Goforth CAAM 550 HW 8 Due 10/27/21

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd
from matplotlib import cm
```

Problem 1

part b

[2]:
```
# Making function for part b first in order to use it in part a

def horner(a, xgiven, xout):
    '''Function to evaluate polynomials using Horner's scheme.

    Parameters
    ----------
    a : np.array
        vector of interpolating polynomial coefficients
    xgiven : np.array
        points used to calculate interpolating polynomial
    xout : np.array
        points at which polynomial is to be evaluated

    Returns
    -------
    y : np.array
        vector of the value of the polynomial evaluated at the values in xout

    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''

    if np.isscalar(xout):
        n = 1
```

```
    else:
        n = xout.size
    if a.size == 1:
        return a
    else:
        val = a[-1] * np.ones(n)
        for i in range(a.size - 2, -1, -1):
            val = val * (xout - xgiven[i]) + a[i]
        return val
```

part c

[3]:
```
# Making function for part c first in order to use it in part a

def NewtonUpdate(a, x, xnew):
    '''Function to find additional coefficient of interpolating polynomial
       using Newton basis when given a new point.

    Parameters
    ----------
    a : np.array
        vector of original interpolating polynomial coefficients
    x : np.array
        vector of given points used to calculate original interpolating␣
 ↪polynomial
    xnew : tuple
           tuple of values (x, f(x)) used to update interpolation

    Returns
    -------
    a : np.array
        vector of the coefficients of the new Newton basis vectors in the␣
 ↪interpolation

    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''

    den = 1
    for xi in x:
        if xi == xnew[0]:
            raise Exception("This value of x has already been used in the␣
 ↪interpolation.")
        den = den * (xnew[0] - xi)
    anew = (xnew[1] - horner(a, x, xnew[0])) /  den
```

```
    return np.append(a, anew)
```

part a.

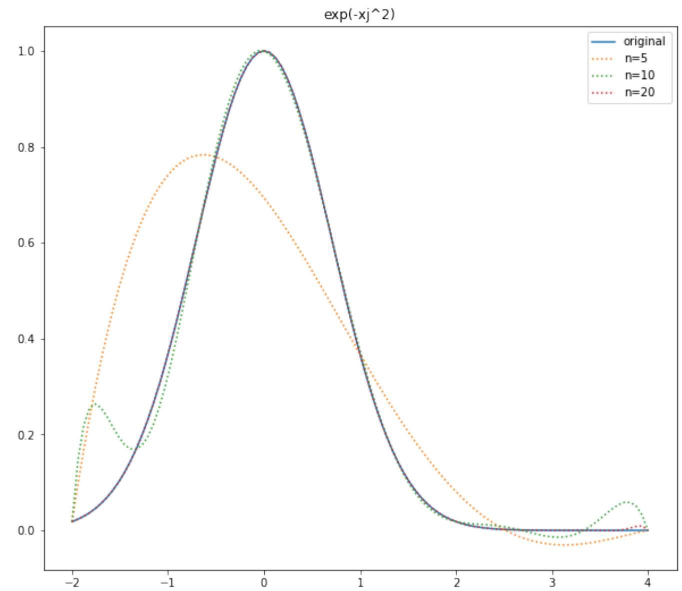
```
[4]: def NewtonInterpolate(x, f):
    '''Function to find coefficients of interpolating polynomial using Newton␣
 ↪basis.

    Parameters
    ----------
    x : np.array
        vector of given points
    f : np.array
        function value at given x points (f(x))

    Returns
    -------
    a : np.array
        vector of the coefficients of the Newton basis vectors in the␣
 ↪interpolation

    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''

    n = x.size
    a = np.array([f[0]])
    for i in range(1, n):
        a = NewtonUpdate(a, x[:i], (x[i], f[i]))
    return a
```

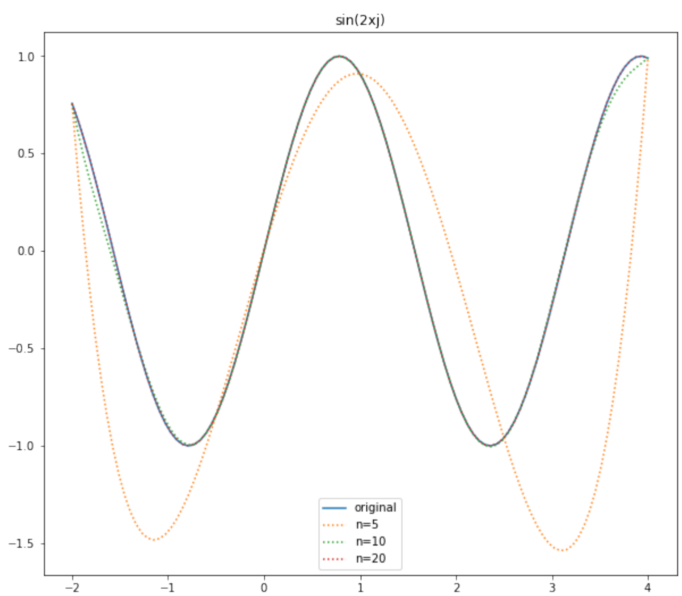
part d. f = tanh(x) is used as the function of my choice

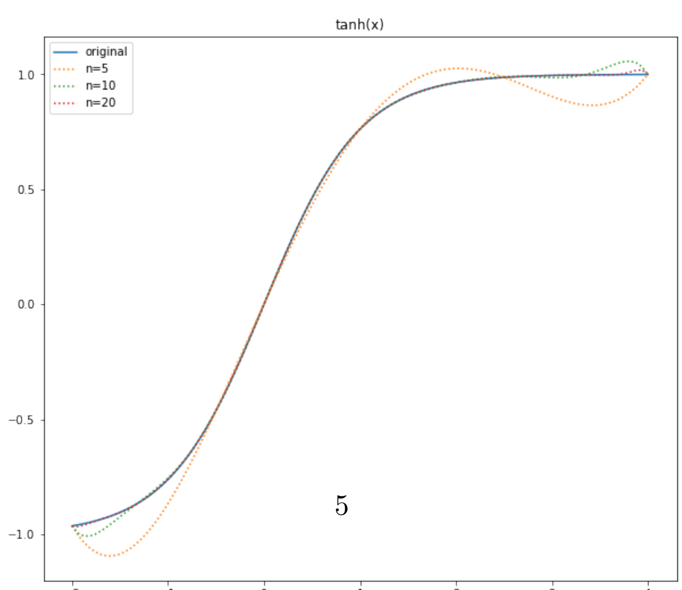
```
[5]: nvec = (5, 10, 20)
fig, axs = plt.subplots(3, figsize=(10, 30))

xplot = np.linspace(-2, 4, 100)
y01 = np.exp(-np.power(xplot, 2))
y02 = np.sin(2 * xplot)
y03 = np.tanh(xplot)

axs[0].plot(xplot, y01, label='original')
axs[1].plot(xplot, y02, label='original')
axs[2].plot(xplot, y03, label='original')

for n in nvec:
```

```python
    xj = np.linspace(-2, 4, n)
    f1 = np.exp(-np.power(xj, 2))
    f2 = np.sin(2 * xj)
    f3 = np.tanh(xj)

    a1 = NewtonInterpolate(xj, f1)
    a2 = NewtonInterpolate(xj, f2)
    a3 = NewtonInterpolate(xj, f3)

    y1 = horner(a1, xj, xplot)
    y2 = horner(a2, xj, xplot)
    y3 = horner(a3, xj, xplot)

    labelstr = 'n=' + str(n)
    axs[0].plot(xplot, y1, ':', label=labelstr)
    axs[1].plot(xplot, y2, ':', label=labelstr)
    axs[2].plot(xplot, y3, ':', label=labelstr)


axs[0].set_title('exp(-xj^2)')
axs[1].set_title('sin(2xj)')
axs[2].set_title('tanh(x)')

axs[0].legend()
axs[1].legend()
axs[2].legend()
```

```
[5]: <matplotlib.legend.Legend at 0x1afcda3abb0>
```


exp(-xj^2)
original
n=5
n=10
n=20


sin(2xj)
original
n=5
n=10
n=20

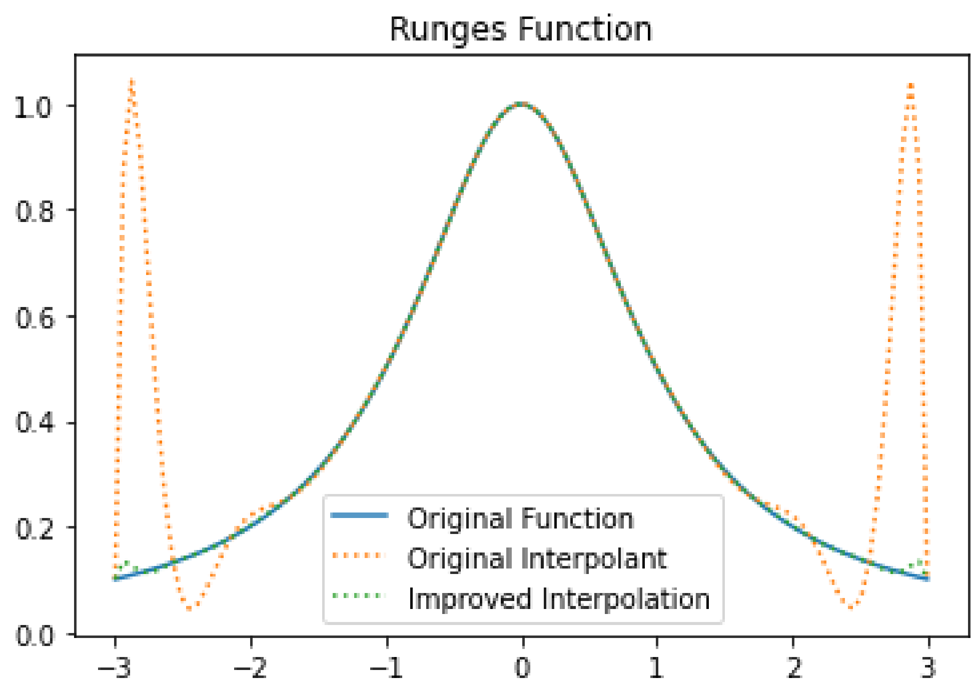

tanh(x)
original
n=5
n=10
n=20

part e.

[6]:
```
xplot = np.linspace(-3, 3, 100)
y01 = (1 + np.power(xplot, 2))**-1
plt.plot(xplot, y01, label='Original Function')

x1 = np.linspace(-3, 3, 15)
f1 = (1 + np.power(x1, 2))**-1
a = NewtonInterpolate(x1, f1)
y1 = horner(a, x1, xplot)
plt.plot(xplot, y1, ':', label='Original Interpolant')
plt.title('Runges Function')

print('Adding points at -2.8, -2.4, 2.4, 2.8 to offset Runge phenomena.')

newx = (-2.8, -2.4, 2.4, 2.8)
for pt in newx:
    f = (1 + pt**2)**-1
    a = NewtonUpdate(a, x1, (pt, f))
    x1 = np.append(x1, pt)
y2 = horner(a, x1, xplot)
plt.plot(xplot, y2, ':', label='Improved Interpolation')
plt.legend()
```

```
Adding points at -2.8, -2.4, 2.4, 2.8 to offset Runge phenomena.
```

[6]:
```
<matplotlib.legend.Legend at 0x1afcdaf6eb0>
```



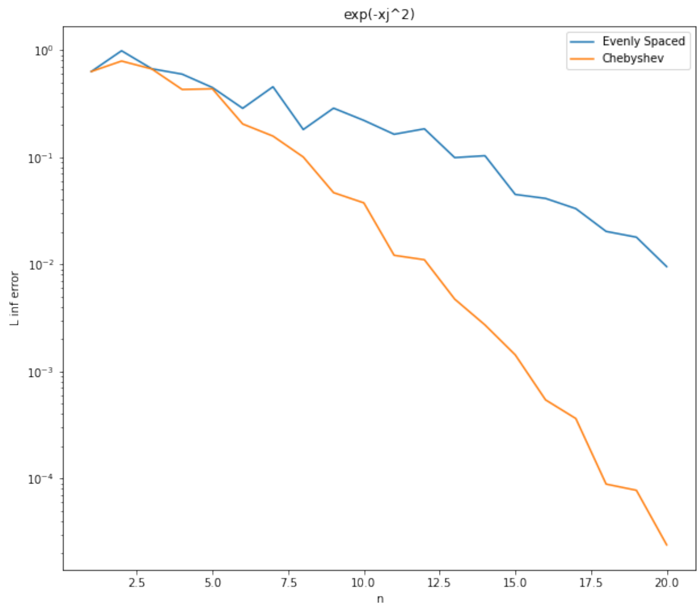
part f.

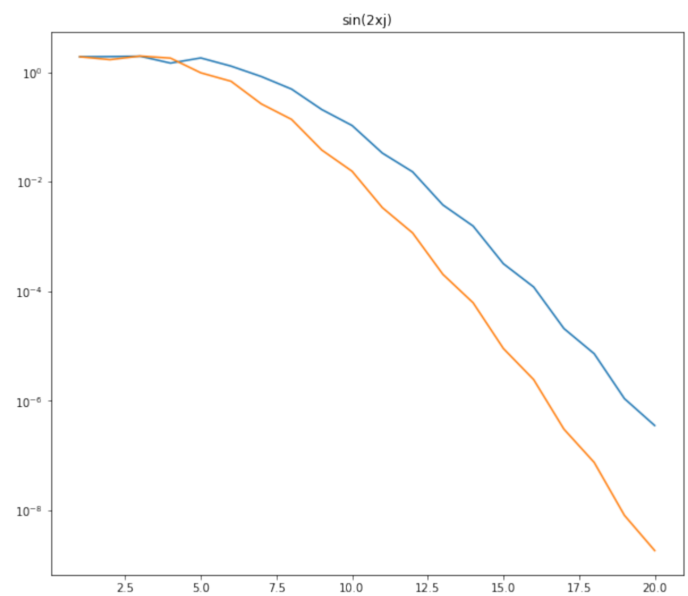
[7]:
```
n = range(1, 21)
err1even = np.zeros(20)
err1cheby = np.zeros(20)
err2even = np.zeros(20)
err2cheby = np.zeros(20)
err3even = np.zeros(20)
err3cheby = np.zeros(20)

a = -2
b = 4
xtest = np.linspace(-2, 4, num=3001)
f1test = np.exp(-np.power(xtest, 2))
f2test = np.sin(2 * xtest)
f3test = np.tanh(xtest)

for i in n:
    if i == 1:
        # for n = 1 chebyshev and even spaced are identical so just need to do␣
↪one
        # interpolation is y = f(x1)
        xeven = np.array([1])
        f1even = np.array(math.exp(-1))
        f2even = np.array(math.sin(2))
```

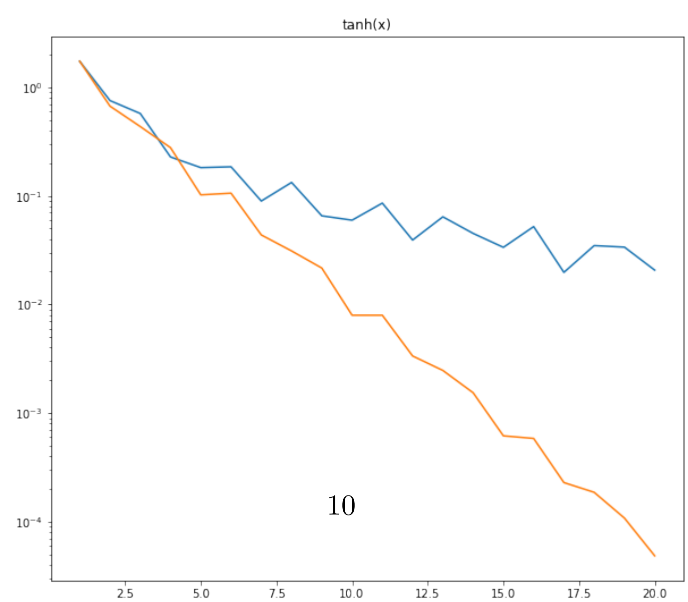
```
        f3even = np.array(math.tanh(1))
        err1even[0] = np.max(np.abs(f1test - f1even))
        err1cheby[0] = err1even[0]
        err2even[0] = np.max(np.abs(f2test - f2even))
        err2cheby[0] = err2even[0]
        err3even[0] = np.max(np.abs(f3test - f3even))
        err3cheby[0] = err3even[0]
    else:
        xeven = np.linspace(a, b, i)
        xcheby = np.zeros(i)
        for j in range(0, i):
            xcheby[j] = .5 * (a + b) + .5 * (b - a) * math.cos((2 * j + 1) * math.pi / (2 * i))

        f1even = np.exp(-np.power(xeven, 2))
        f2even = np.sin(2 * xeven)
        f3even = np.tanh(xeven)

        f1cheby = np.exp(-np.power(xcheby, 2))
        f2cheby = np.sin(2 * xcheby)
        f3cheby = np.tanh(xcheby)

        a1even = NewtonInterpolate(xeven, f1even)
        a2even = NewtonInterpolate(xeven, f2even)
        a3even = NewtonInterpolate(xeven, f3even)

        a1cheby = NewtonInterpolate(xcheby, f1cheby)
        a2cheby = NewtonInterpolate(xcheby, f2cheby)
        a3cheby = NewtonInterpolate(xcheby, f3cheby)

        y1even = horner(a1even, xeven, xtest)
        y2even = horner(a2even, xeven, xtest)
        y3even = horner(a3even, xeven, xtest)

        y1cheby = horner(a1cheby, xcheby, xtest)
        y2cheby = horner(a2cheby, xcheby, xtest)
        y3cheby = horner(a3cheby, xcheby, xtest)

        err1even[i - 1] = np.max(np.abs(f1test - y1even))
        err2even[i - 1] = np.max(np.abs(f2test - y2even))
        err3even[i - 1] = np.max(np.abs(f3test - y3even))

        err1cheby[i - 1] = np.max(np.abs(f1test - y1cheby))
        err2cheby[i - 1] = np.max(np.abs(f2test - y2cheby))
        err3cheby[i - 1] = np.max(np.abs(f3test - y3cheby))

fig, axs = plt.subplots(3, figsize=(10, 30))
```

```
axs[0].semilogy(n, err1even, n, err1cheby)
axs[0].legend(['Evenly Spaced', 'Chebyshev'])
axs[0].set_xlabel('n')
axs[0].set_ylabel('L inf error')

axs[1].semilogy(n, err2even, n, err2cheby)
axs[0].legend(['Evenly Spaced', 'Chebyshev'])
axs[0].set_xlabel('n')
axs[0].set_ylabel('L inf error')

axs[2].semilogy(n, err3even, n, err3cheby)
axs[0].legend(['Evenly Spaced', 'Chebyshev'])
axs[0].set_xlabel('n')
axs[0].set_ylabel('L inf error')

axs[0].set_title('exp(-xj^2)')
axs[1].set_title('sin(2xj)')
axs[2].set_title('tanh(x)')
```

[7]: Text(0.5, 1.0, 'tanh(x)')


exp(-xj^2)
Evenly Spaced
Chebyshev
L inf error
n


sin(2xj)


tanh(x)

Problem 2 part a.

```
[8]: def Problem2(x, f):
         '''Function to find coefficients of interpolating polynomial using Newton␣
     ↪basis
          of inverse of function f.

         Parameters
         ----------
         x : np.array
             vector of given points
         f : np.array
             function value at given x points (f(x))

         Returns
         -------
         a : np.array
             vector of the coefficients of the Newton basis vectors in the␣
     ↪interpolation of the inverse
             of function f

         Michael Goforth
         CAAM 550
         Rice University
         October 27, 2021
         '''

         n = x.size
         columnnames = ['k', 'pk(0)']
         data = pd.DataFrame(columns = columnnames)
         for k in range(1, n + 1):
             a = NewtonInterpolate(f[:k], x[:k])
             root = horner(a, f[:k], 0)
             data = data.append({'k': k, 'pk(0)': root}, ignore_index=True)
         return data
```

part b.

```
[9]: def func(x):
         '''Function given in problem 2b.

         Parameters
         ----------
         x : value
```

```
        value to evaluate function at

    Returns
    -------
    y : value
        f(x)

    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''

    return math.cos(x) * math.cosh(x) + 1

nvec = [2, 4, 6]
a = 1.6
b = 2.1
for n in nvec:
    x = np.array([a + (i) * (b - a) / (n - 1) for i in range(n)])
    y = np.array([func(xi) for xi in x])
    print(Problem2(x, y))
```

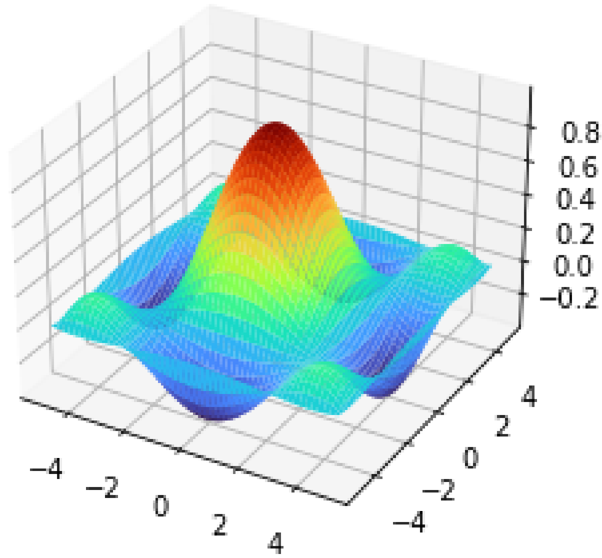
```
   k                pk(0)
0  1                [1.6]
1  2  [1.8292386237791756]
   k                pk(0)
0  1                [1.6]
1  2  [1.9017513821154133]
2  3  [1.8770705677750952]
3  4  [1.8756500401696137]
   k                pk(0)
0  1                [1.6]
1  2  [1.9198106219331343]
2  3  [1.8703235166601409]
3  4  [1.8749265605732877]
4  5   [1.875072022076762]
5  6  [1.8750942766664045]
```

Problem 3. part a.

[22]:
```
def func(x, y):
    '''Function given in problem 3.

    Parameters
    ----------
    x : value
        x value to evaluate function at
```

```
    y : value
        y value to evaluate function at

    Returns
    -------
    z : value
        f(x, y)

    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''
    return((1 + x**2)**-1 * (1 + y**2)**-1)

def LagEval(xin, yin, f, xout, yout):
    '''Function that evaluates a Lagrange Polynomial interpolation at points
    contained in grid of (xout, yout).

    Parameters
    ----------
    xin : np.array
          points used to create Lagrange Interpolation
    yin : np.array
          points used to create Lagrange Interpolation
    f : np.array
        matrix of coefficients of Lagrange Interpolation
    xout : np.array
           x values at which to evaluate the function
    yout : np.array
           y values at which to evaluate the function

    Returns
    -------
    fout : np.array
           matrix of values of Lagrange polynomial at grid defined by xout, yout


    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''
    result = np.zeros([xout.size, yout.size])
    n = xin.size
    m = yin.size
    for r in range(xout.size):
```

```python
        for s in range(yout.size):
            final = 0
            for k in range(n):
                pk = 1
                for j in range(n):
                    if k != j:
                        pk = pk * (xout[r] - xin[j]) / (xin[k] - xin[j])
                for l in range(m):
                    ql = 1
                    for j in range(m):
                        if l != j:
                            ql = ql * (yout[s] - y[j]) / (yin[l] - yin[j])
                    final = final + f[k, l] * pk * ql
            result[r, s] = final
    return result


n = 5
x = np.array([-5 + 2.5 * (j) for j in range(5)])
y = np.copy(x)

f = np.zeros([n, n])
for i in range(5):
    xi = x[i]
    for j in range(5):
        yj = y[j]
        f[i, j] = func(xi, yj)

xplot = np.linspace(-5, 5, 100)
yplot = np.linspace(-5, 5, 100)
fplot = LagEval(x, y, f, xplot, yplot)
xx, yy = np.meshgrid(xplot, yplot)

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

surf = ax.plot_surface(xx, yy, fplot, cmap=cm.turbo)
```



part b

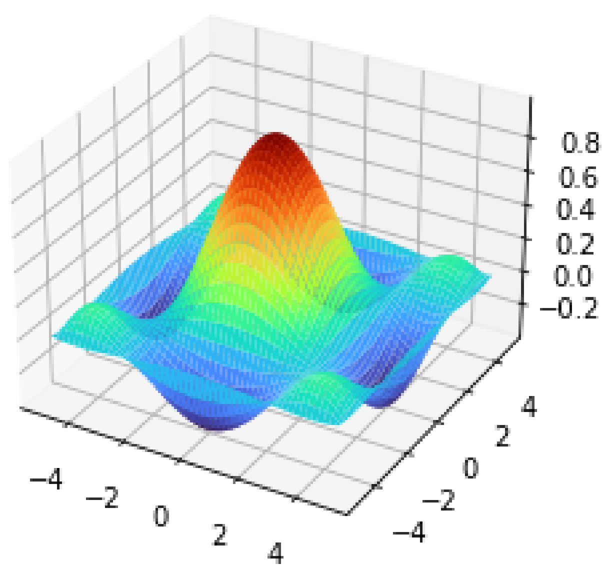
[47]:
```python
def NewtonEval(x, i, xdesired):
    '''Function to evaluate the i-th Newton polynomial at value xdesired.

    Parameters
    ----------
    x : np.array
        vector of given x values used to create Newton polynomial
    i : value
        i-th Newton polynomial will be returned
    xdesired : value
               value for which i-th polynomial is to be evaluated at


    Returns
    -------
    z : value
        value of the Newton polynomial at xdesired, N(xdesired)

    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''
    final = 1
    for k in range(i):
```

```
        final = final * (xdesired - x[k])
    return final


def NewtonPoly3D(x, y, f):
    '''Function to find coefficients of interpolating polynomial
       using Newton basis.

    Parameters
    ----------
    x : np.array
        vector of given x values
    y : np.array
        vector of given y values
    f : np.array
        matrix of values, where the value at [i, j] corresponds to f(xi, yj)


    Returns
    -------
    a : np.array
        matrix of the coefficients of the Newton basis vectors

    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''

    n = x.size
    m = y.size
    A = np.zeros([n, m])
    for k in range(n):
        xk = x[k]
        pk = NewtonEval(x, k, xk)
        for l in range(m):
            yl = y[l]
            ql = NewtonEval(y, l, yl)
            akl = f[k, l]
            for i in range(n):
                pi = NewtonEval(x, i, xk)
                for j in range(m):
                    qj = NewtonEval(y, j, yl)
                    akl = akl - A[i, j] * pi * qj
            A[k, l] = akl / (pk * ql)
    return A
```

```python
def NewtEval3D(xin, yin, f, xout, yout):
    '''Function that evaluates a Newton Polynomial interpolation at points␣
 ↪defined
    in grid created from xout, yout.

    Parameters
    ----------
    xin : np.array
          points used to create Newton Polynomials
    yin : np.array
          points used to create Newton Polynomial
    f : np.array
        matrix of coefficients of Newton Polynomial
    xout : np.array
           x values at which to evaluate the function
    yout : np.array
           y values at which to evaluate the function

    Returns
    -------
    fout : np.array
           matrix of values of Newton polynomial at grid defined by xout, yout


    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''
    result = np.zeros([xout.size, yout.size])
    n = xin.size
    m = yin.size
    for r in range(xout.size):
        xr = xout[r]
        for s in range(yout.size):
            ys = yout[s]
            #print('-------')
            #print(xr, ys)
            final = 0
            for k in range(n):
                pk = NewtonEval(xin, k, xr)
                for l in range(m):
                    ql = NewtonEval(yin, l, ys)
                    #print(pk, ql, f[k,l])
                    final = final + f[k, l] * pk * ql
            result[r, s] = final
```

```python
            #print(result)
    return result


A = NewtonPoly3D(x, y, f) # values defined in problem 3 part a
print(A)

f2plot = NewtEval3D(x, y, A, xplot, yplot)
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

surf = ax.plot_surface(xx, yy, f2plot, cmap=cm.turbo)
```

```
[[ 1.47928994e-03  1.53029994e-03  2.34645991e-03 -1.02019996e-03
   2.04039992e-04]
 [ 1.53029994e-03  1.58306890e-03  2.42737232e-03 -1.05537927e-03
   2.11075854e-04]
 [ 2.34645991e-03  2.42737232e-03  3.72197089e-03 -1.61824821e-03
   3.23649642e-04]
 [-1.02019996e-03 -1.05537927e-03 -1.61824821e-03  7.03586179e-04
  -1.40717236e-04]
 [ 2.04039992e-04  2.11075854e-04  3.23649642e-04 -1.40717236e-04
   2.81434472e-05]]
```



Problem 5 part a

```python
[14]: def PiecewiseLinear(f):
```

```
    '''Function that constructs a piecewise linear interpolant of a given␣
↪function.

    Parameters
    ----------
    f : np.array
        vector of values of f at 0, h, 2h, ..., 1

    Returns
    -------
    (c0, c1) : tuple of np.array
               vector of values of coefficients of the linear pieces of␣
↪interpolant (p_i(x) = c1(i)x + c0(i))

    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''

    n = f.size - 1
    h = 1 / n

    c0 = np.zeros([n])
    c1 = np.zeros([n])
    for i in range(n):
        c1[i] = (f[i+1] - f[i]) / h
        c0[i] = f[i] - c1[i] * i * h
    return (c0, c1)
```

part b

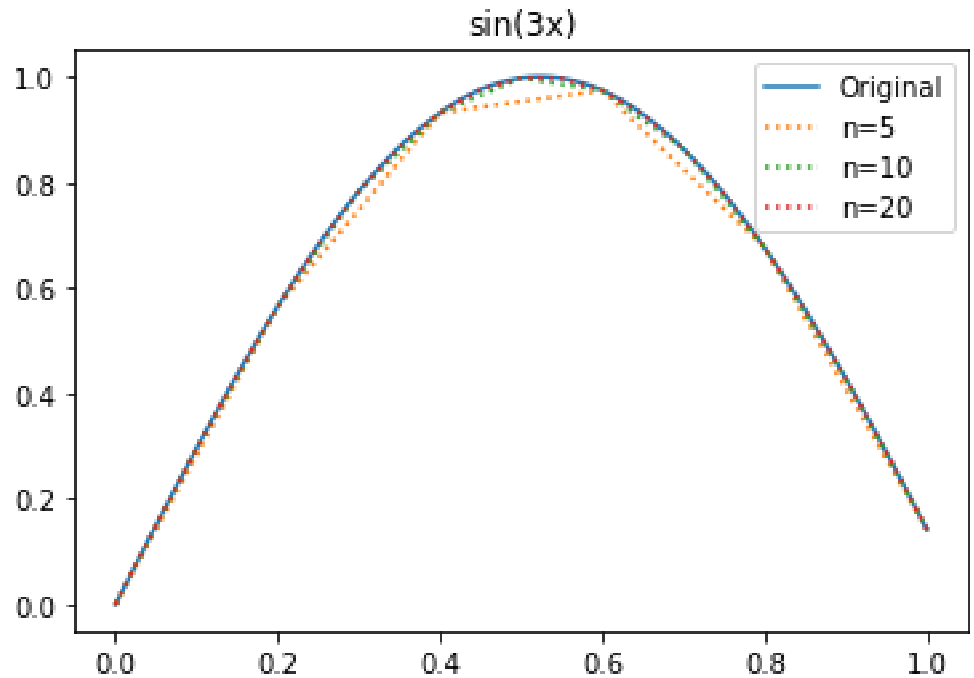
[15]:
```
def EvalPiecewiseLinear(c0, c1, x):
    '''Function that constructs a piecewise linear interpolant of a given␣
↪function.

    Parameters
    ----------
    c0 : np.array
         vector of values of the constant term coefficients of the linear␣
↪pieces of the interpolant
    c1 : np.array
         vector of values of the x term coefficients of the linear pieces of␣
↪the interpolant
    x : np.array
        vector of x values at which the piecewise linear interpolant is to be␣
↪evaluated at
```

```
    Returns
    -------
    y : np.array
        vector of y values at which the piecewise linear interpolant was␣
↪evaluated at, f(x) = y

    Michael Goforth
    CAAM 550
    Rice University
    October 27, 2021
    '''

    n = c0.size
    h = 1/n
    y = np.zeros([x.size])
    for i in range(n):
        cond1 = np.where(i * h <= x, 1, 0)
        cond2 = np.where(x < (i + 1) * h, 1, 0)
        y = y + cond1 * cond2 * (x * c1[i] + c0[i])
    condend = np.where(n * h == x, 1, 0)
    y = y + condend * (n * h * c1[-1] + c0[-1])
    return y
```

part c

[136]:
```
# sin(3x)
xplot = np.linspace(0, 1, 101)
f1plot = np.sin(3 * xplot)
plt.plot(xplot, f1plot, label='Original')
nvec = [5, 10, 20]
for n in nvec:
    x = np.linspace(0, 1, n + 1)
    f = np.sin(3 * x)
    c0, c1 = PiecewiseLinear(f)
    xint = EvalPiecewiseLinear(c0, c1, xplot)
    labelstr = 'n=' + str(n)
    plt.plot(xplot, xint, ':', label=labelstr)
plt.legend()
plt.title('sin(3x)')
```

[136]:
```
Text(0.5, 1.0, 'sin(3x)')
```

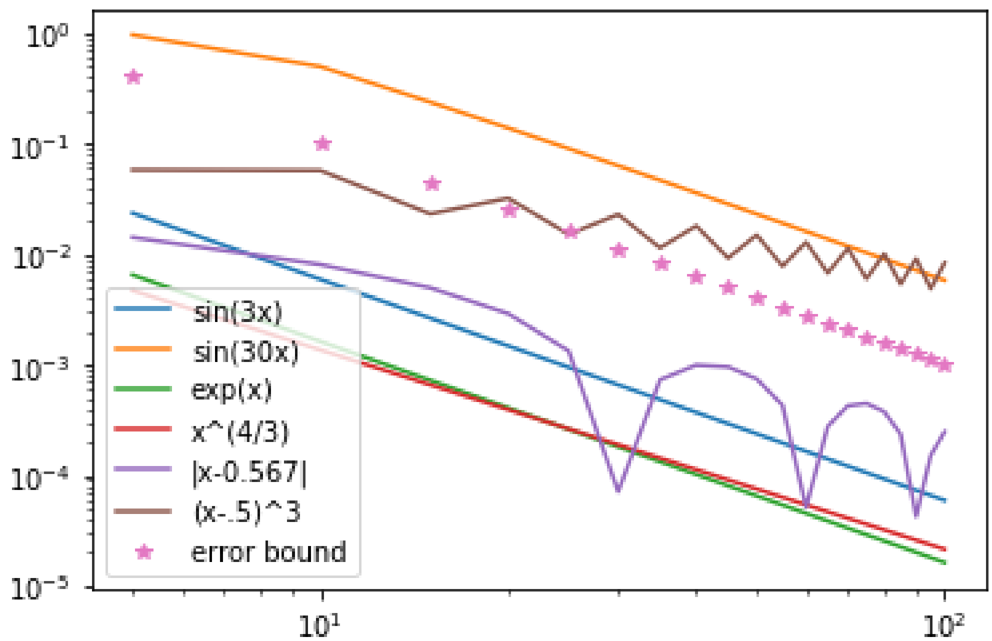


part d

[139]:
```
from scipy import integrate

nvec = np.arange(5, 105, 5)
l2err = np.zeros([nvec.size, 6]) #each column is error for 1 of the 6 given␣
↪functions
linferr = np.zeros([nvec.size, 6])
# Values to use in interval approximation
xtrue = np.linspace(0, 1, 1001)
ftrue = np.zeros([1001, 6])
ftrue[:, 0] = np.sin(3 * xtrue)
ftrue[:, 1] = np.sin(30 * xtrue)
ftrue[:, 2] = np.exp(xtrue)
ftrue[:, 3] = np.power(xtrue, 4/3)
ftrue[:, 4] = abs(xtrue - .567)
ftrue[:, 5] = np.cbrt(xtrue - .5)
count = 0
for n in nvec:
    x = np.linspace(0, 1, n + 1)
    f = np.zeros([n + 1, 6])
    f[:, 0] = np.sin(3 * x)
    f[:, 1] = np.sin(30 * x)
    f[:, 2] = np.exp(x)
    f[:, 3] = np.power(x, 4/3)
```

```
    f[:, 4] = abs(x - .567)
    f[:, 5] = np.cbrt(x - .5)
    for i in range(6):
        c0, c1 = PiecewiseLinear(f[:, i])
        ptrue = EvalPiecewiseLinear(c0, c1, xtrue)
        temp = np.power(abs(ptrue - ftrue[:, i]), 2)
        i_comp = integrate.simpson(temp, dx=1/1000)
        l2err[count, i] = i_comp**.5
        linferr[count, i] = np.max(temp)
    count = count + 1

plt.loglog(nvec, l2err[:, 0], label='sin(3x)')
plt.loglog(nvec, l2err[:, 1], label='sin(30x)')
plt.loglog(nvec, l2err[:, 2], label='exp(x)')
plt.loglog(nvec, l2err[:, 3], label='x^(4/3)')
plt.loglog(nvec, l2err[:, 4], label='|x-0.567|')
plt.loglog(nvec, l2err[:, 5], label='(x-.5)^3')
errbound = 10 * np.power(nvec.astype(float), -2)
plt.loglog(nvec, errbound, '*', label='error bound')
plt.legend()
```

[139]: <matplotlib.legend.Legend at 0x1afd8b23400>



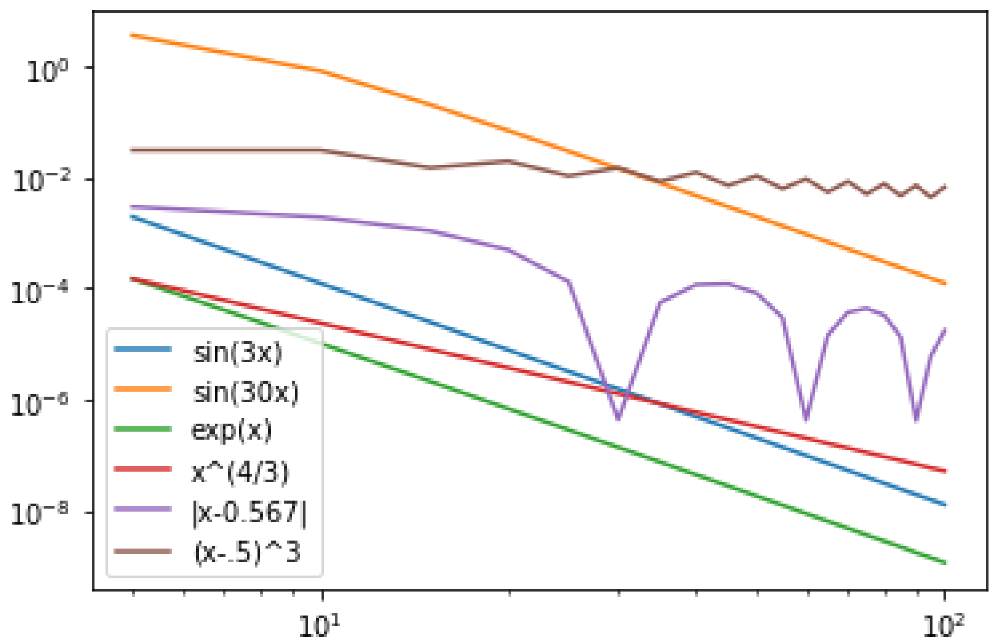
part e

[49]:

```
# L infinity error is calculated in previous problem
plt.loglog(nvec, linferr[:, 0], label='sin(3x)')
plt.loglog(nvec, linferr[:, 1], label='sin(30x)')
plt.loglog(nvec, linferr[:, 2], label='exp(x)')
plt.loglog(nvec, linferr[:, 3], label='x^(4/3)')
plt.loglog(nvec, linferr[:, 4], label='|x-0.567|')
plt.loglog(nvec, linferr[:, 5], label='(x-.5)^3')
plt.legend()
```

[49]:

```
<matplotlib.legend.Legend at 0x1afd1a9c640>
```

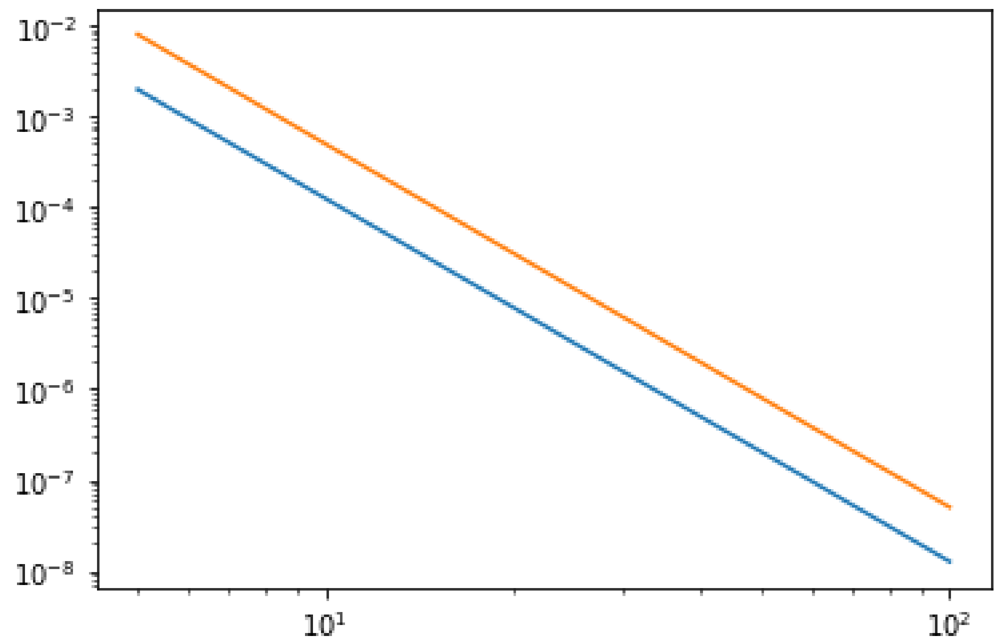


[178]:

```
#f(x) = sin(3x)
# Error is O(h^4)
hvec = 1 /nvec
plt.loglog(nvec, linferr[:, 0], nvec, 5*np.power(hvec, 4))
```

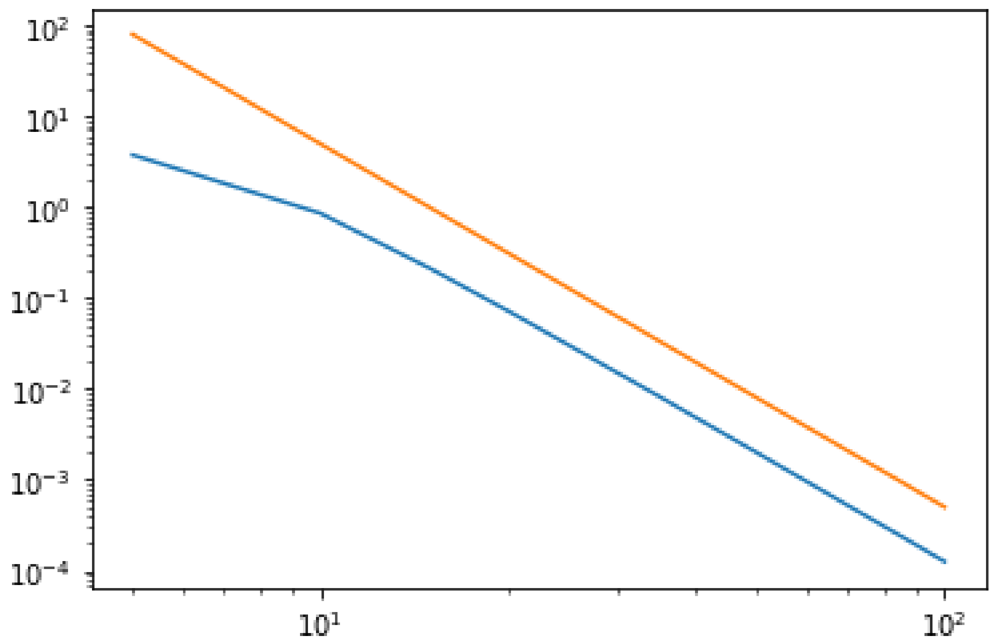
[178]:

```
[<matplotlib.lines.Line2D at 0x1afe02563a0>,
 <matplotlib.lines.Line2D at 0x1afe0256460>]
```



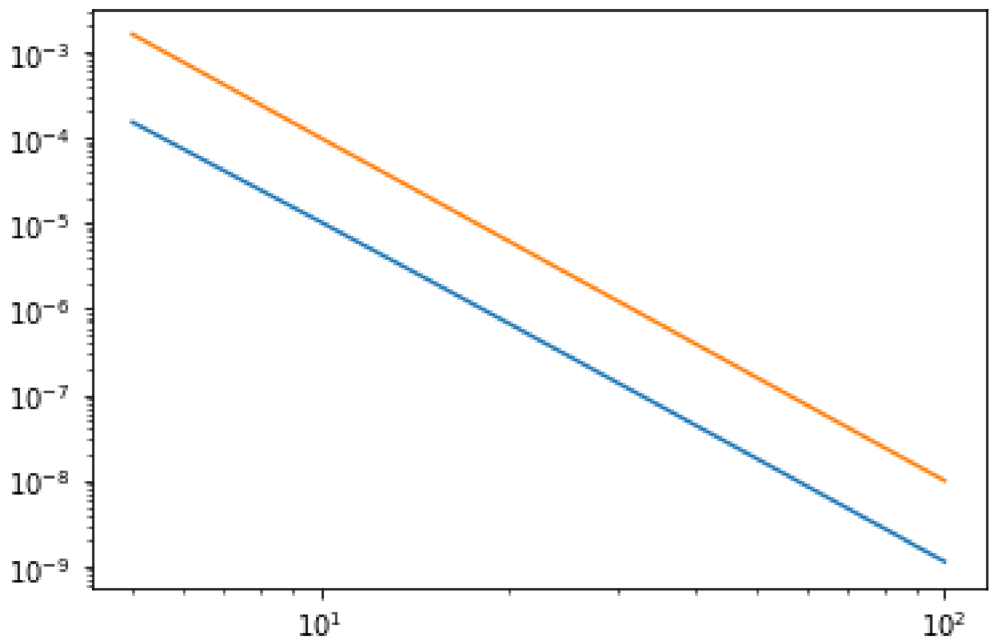
[194]:
```
#f(x) = sin(30x)
# Error is O(h^4)
hvec = 1 /nvec
plt.loglog(nvec, linferr[:, 1], nvec, 50000*np.power(hvec,4))
```

[194]:
```
[<matplotlib.lines.Line2D at 0x1afe3d19550>,
 <matplotlib.lines.Line2D at 0x1afe3d19610>]
```



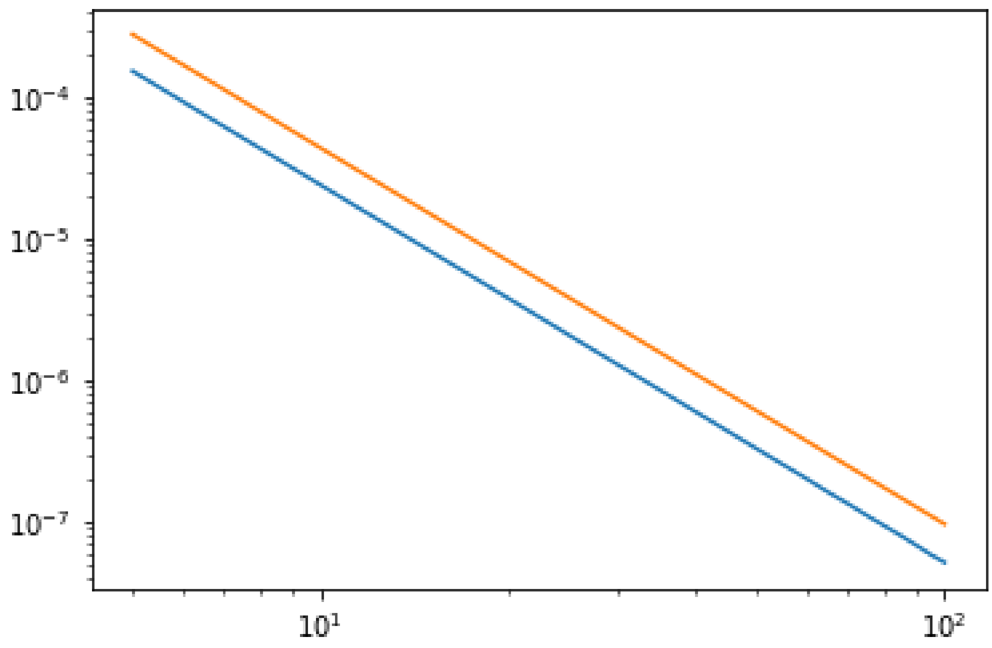
[213]:
```
#f(x) = e^x
# Error is O(h^4)
hvec = 1 /nvec
plt.loglog(nvec, linferr[:, 2], nvec, np.power(hvec,4))
```

[213]:
```
[<matplotlib.lines.Line2D at 0x1afe8b49df0>,
 <matplotlib.lines.Line2D at 0x1afe8b49eb0>]
```



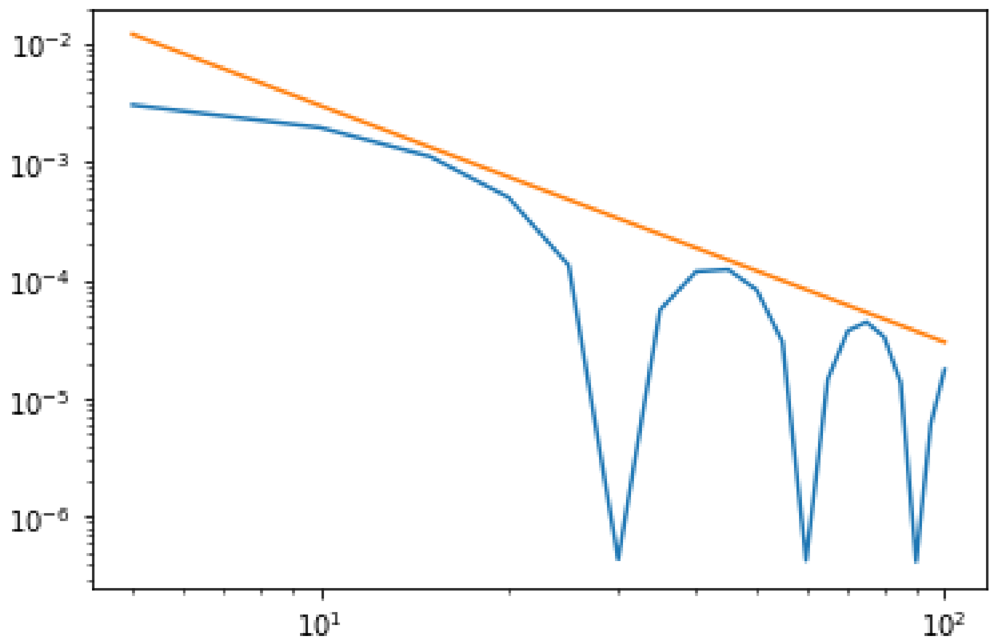

```
[223]: #f(x) = x^(4/3)
       # Error is O(h^2.66)
       hvec = 1 /nvec
       plt.loglog(nvec, linferr[:, 3], nvec, .02 * np.power(hvec,2.66))
```

```
[223]: [<matplotlib.lines.Line2D at 0x1afea6c8100>,
        <matplotlib.lines.Line2D at 0x1afea6c80d0>]
```



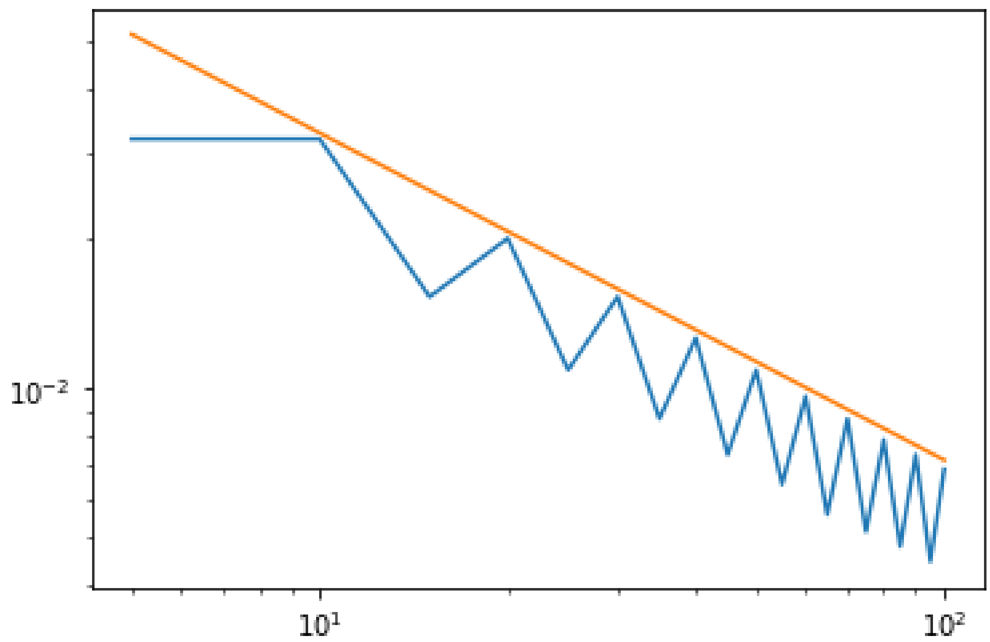
[250]:
```
#f(x) = |x-.0567|
# Error is O(h^2)
hvec = 1 /nvec
plt.loglog(nvec, linferr[:, 4], nvec, .3 * np.power(hvec, 2))
```

[250]:
```
[<matplotlib.lines.Line2D at 0x1aff10d1370>,
 <matplotlib.lines.Line2D at 0x1aff10d1430>]
```



[260]:
```
#f(x) = (x-.5)^3
# Error is O(h^(2/3))
hvec = 1 /nvec
plt.loglog(nvec, linferr[:, 5], nvec, .15 * np.power(hvec,.66))
```

[260]:
```
[<matplotlib.lines.Line2D at 0x1afedd29c70>,
 <matplotlib.lines.Line2D at 0x1afedd29b80>]
```


10^{-2}
10^{1}
10^{2}