This program generates the spin wave frequencies versus inverse film thickness, shown in Figure 5 of the article.

Results of the atomic layer method, as well as the corresponding continuum method, are displayed.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import eig
from matplotlib.ticker import MaxNLocator
from scipy.optimize import curve_fit
import sympy as sp
from sympy import expand, symbols
```

In [2]:

```
# Define Kostylev's 2Nx2N-4 matrix with boundary condition intergrated in to layer 2 and N-1:---------
def MatrixKdem(L,N,D,k):
    SW = np.zeros((2*N-4,2*N-4), dtype=np.complex128)                 # setup 2Nx2N-4 Matrix
    nn=np.linspace(0,N-1-2,N-2)                                       # counter for iteration
    dy=L/N                                                            # steps side \Delta
    dD=(-D)*k*b/A                                                     # d_D short hand (contain DMI const
    B_zee = mu_0*H                                                    # Zeeman effective field term
    B_ex = 2*A/(M_0*(dy**2))                                          # exchange effecive field term
    B_DM = dD*dy*B_ex/(1-(dD**2)*(dy**2))                             # DMI effective field term
    B_dem=mu_0*M_0
    for nn in range(0,1):                                             # i=2:
        SW[0][0]=-(1j)*B_DM                                           # x-direction x_2
        SW[0][1]=(B_zee+((dy**2)*(k**2)+2)*B_ex-B_DM/(dy*dD)+B_dem)      # x-direction y_2
        SW[0][3]=-B_ex                                                # x-direction y_3
        SW[1][0]=-(B_zee+((dy**2)*(k**2)+2)*B_ex-B_DM/(dy*dD))        # y-direction x_2
        SW[1][1]=-(1j)*B_DM                                           # y-direction y_2
        SW[1][2]=B_ex                                                 # y-direction x_3
        for nn in range(1,N-1-2):                                     # any i layers:
            SW[2*nn][2*(nn-1)+1]=-B_ex                                # x-direction y_i-1
            SW[2*nn][2*nn+1]=(B_zee+((dy**2)*(k**2)+2)*B_ex+B_dem)       # x-direction y_i
            SW[2*nn][2*(nn+1)+1]=-B_ex                                # x-direction y_i+1
            SW[2*nn+1][2*(nn-1)]=B_ex                                 # y-direction x_i-1
            SW[2*nn+1][2*nn]=-(B_zee+((dy**2)*(k**2)+2)*B_ex)         # y-direction x_i
            SW[2*nn+1][2*(nn+1)]=B_ex                                 # y-direction x_i+1
            for nn in range(N-1-2, N-2):                              # i=N-1:
                SW[2*nn][2*(nn-1)+1]=-B_ex                            # x-direction y_N-2
                SW[2*nn][2*nn+1]=(B_zee+((dy**2)*(k**2)+1)*B_ex+B_dem)   # X-direction y_N-1
                SW[2*nn+1][2*(nn-1)]=B_ex                             # y-direction x_N-2
                SW[2*nn+1][2*nn]=-(B_zee+((dy**2)*(k**2)+1)*B_ex)     # y-direction x_N-1
    return SW*gamma/(1j)
# 2Nx2N Matrix with demag. switch off-----------------------------------------------------------------
def Matr_0dem(N,D,kx,kz):                                            # N layernumber D DMI cost. k_x, k_z
    SW = np.zeros((2*N,2*N), dtype=np.complex128)                 # set up 2N x 2N matrix
    nn=np.linspace(0,N-1,N)                                       # counter for iteration
    K_ex = 2*(np.cos(a*kx)+np.cos(a*kz))                          # exchange term of the wavenumber
    K_DM = np.sin(a*kx)                                           # DMI term of the wavenumber
    B_DM = 2*D/(a*M_0)                                            # DMI effective field term
    B_ex = 2*A/( (a**2)*M_0 )                                     # exchange effective field term
    B_de = 0                                                      # demagnetizing effective field term sw
    for nn in range(0,1):                                         # i=1 (top)
```

```
        SW[0][0]=1j*B_DM*K_DM                                 # x-direction x_1
        SW[0][1]=B+B_ex*(5-K_ex)+B_de                         # x-direction y_1
        SW[0][3]=-B_ex                                        # x-direction y_2
        SW[1][0]=-(B+B_ex*(5-K_ex))                           # y-direction x_1
        SW[1][1]=1j*B_DM*K_DM                                 # y-direction y_1
        SW[1][2]=B_ex                                         # y-direction x_2
        for nn in range(1,N-1):                               # any i layers inbetween
            SW[2*nn][2*(nn-1)+1]=-B_ex                        # x-direction y_i-1
            SW[2*nn][2*nn+1]=B+B_ex*(6-K_ex)+B_de             # x-direction y_i
            SW[2*nn][2*(nn+1)+1]=-B_ex                        # x-direction y_i+1
            SW[2*nn+1][2*(nn-1)]=B_ex                         # y-direction x_i-1
            SW[2*nn+1][2*nn]=-(B+B_ex*(6-K_ex))               # y-direction x_i
            SW[2*nn+1][2*(nn+1)]=B_ex                         # y-direction x_i+1
            for nn in range(N-1, N):                          # i=N (bottom)
                SW[2*nn][2*(nn-1)+1]=-B_ex                    # x-direction y_N-1
                SW[2*nn][2*nn+1]=B+B_ex*(5-K_ex)+B_de         # x-direction y_N
                SW[2*nn+1][2*(nn-1)]=B_ex                     # y-direction x_N-1
                SW[2*nn+1][2*nn]=-(B+B_ex*(5-K_ex))           # y-direction x_N
    return SW*gamma/(1j)                                      # move iw/|r| to the other side
# define frequency funciont for Kostylev's matrix -----------------------------------------------
def FreqK(N,SW,n):                                          # N #ofDiscretePoints SW Matrix'MatrixK(L,
    omegaGHz=[]                                             # set up empty list
    w,v=eig(SW)                                             # obtain eigan value and eigenvalues and e
    omegaRaw=w                                              # extract raw arrays of eigenvalues
    idx1 = np.argsort(omegaRaw)                             # sort the eigenvalues in assending order
    omegaRaw = omegaRaw[idx1]                               # get the sortted index
    v = v[:,idx1]                                           # use the same index to sort the eigenvect
    omega=np.zeros(N-2, dtype = 'complex_')                 # set up empty array for frequencies
    for ii in range (0,N-2):                                # for loop to exact the right array
        omega[ii]=omegaRaw[ii+N-2]
    omegaGHz.append(omega.real[n]/1e9)                      # convert to GHz
    return omegaGHz
# define the mode amplitude function------------------------------------------------------------
def ModAmpXK(N,SW,n):                                         # N discrete#points
    w,v=eig(SW)                                               # obtain eigenvalues and vectors
    omegaRaw=w                                                # extract raw arrays of eigenvalues
    idx1 = np.argsort(omegaRaw)                               # sort the eigenvalues in assending ord
    omegaRaw = omegaRaw[idx1]                                 # get the sortted index
    v = v[:,idx1]                                             # sort the eigenvectors by the new inde
    EigV=np.zeros((2*(N-2),N-2),dtype = 'complex_')           # set up eigenvector arrays
    for ii in range (0,N-2):                                  # extract the amplitude on the X-direct
```

```
        for jj in range (0,2*(N-2)):
            EigV[jj][ii]=v[jj][ii+N-2]
    EigVx=np.zeros((N-2,N-2),dtype=np.complex128)
    for ii in range (0,N-2):
        for jj in range (0,N-2):
            EigVx[ii][jj]=EigV[2*ii][jj]
            if abs(EigVx[ii][jj].real)>abs(EigVx[ii][jj].imag):
                EigVx[ii][jj]=EigVx[ii][jj].real
            else:
                EigVx[ii][jj]=EigVx[ii][jj].imag
    return EigVx[:,n].real
# set up array for the layer number ----------------------------------------------------------------
    layers=np.linspace(2,N-1,N-2)
    return layers
# define dispersion relation iteration -------------------------------------------------------------
def SWfreqkxK(L,N,D,h,n):                                      # L layer thickness N layer# D DMI const.
    Max=30.1e6                                                     # maximum wavenumber
    omega1=[]                                                  # set up frequency list
    kx=-Max                                                    # starting k_x
    while kx<Max+1e6:                                          # while loop appends all the freqencies wi
        omega1.append(FreqK(N,MatrixK(L,N,D,kx),n))
        kx=kx+h*1e6                                            # increase steps
    return omega1
def SWfreqkx0(N,D,h,n):                                        # N layernumber D DMI const. h stepsize n
    Max=30.1e6                                                 # maximum wavenumber
    omega1=[]                                                  # set up frequency list
    kx=-Max                                                    # starting k_x
    kz=0                                                       # set k_z = 0
    while kx<Max+1e6:                                          # while loop appends all the freqencies wi
        omega1.append(Freq(N,Matr_0dem(N,D,kx,kz),n))
        kx=kx+h*1e6                                            # increase steps
    return omega1
def xSWfreqkxK(h):                                             # h stepsize
    Max=30.1e6
    x=[]
    kx=-Max
    kz=0
    while kx<Max+1e6:
        x.append(kx/1e6)
        kx=kx+h*1e6
    return x
```

In [3]:

```
# define 2Nx2N Matrix function--------------------------------------------------------------------
def Matrix(N,D,kx,kz):                                        # N layernumber D DMI cost. k_x, k_z wa
    SW = np.zeros((2*N,2*N), dtype=np.complex128)            # set up 2N x 2N matrix
    nn=np.linspace(0,N-1,N)                                   # counter for iteration
    K_ex = 2*(np.cos(a*kx)+np.cos(a*kz))                      # exchange term of the wavenumber
    K_DM = np.sin(a*kx)                                       # DMI term of the wavenumber
    B_DM = 2*D/(a*M_0)                                        # DMI effective field term
    B_ex = 2*A/( (a**2)*M_0 )                                 # exchange effective field term
    B_de = mu_0*M_0                                           # demagnetizing effective field term
    for nn in range(0,1):                                     # i=1 (top)
        SW[0][0]=1j*B_DM*K_DM                                 # x-direction x_1
        SW[0][1]=B+B_ex*(5-K_ex)+B_de                         # x-direction y_1
        SW[0][3]=-B_ex                                        # x-direction y_2
        SW[1][0]=-(B+B_ex*(5-K_ex))                           # y-direction x_1
        SW[1][1]=1j*B_DM*K_DM                                 # y-direction y_1
        SW[1][2]=B_ex                                         # y-direction x_2
        for nn in range(1,N-1):                               # any i layers inbetween
            SW[2*nn][2*(nn-1)+1]=-B_ex                        # x-direction y_i-1
            SW[2*nn][2*nn+1]=B+B_ex*(6-K_ex)+B_de             # x-direction y_i
            SW[2*nn][2*(nn+1)+1]=-B_ex                        # x-direction y_i+1
            SW[2*nn+1][2*(nn-1)]=B_ex                         # y-direction x_i-1
            SW[2*nn+1][2*nn]=-(B+B_ex*(6-K_ex))               # y-direction x_i
            SW[2*nn+1][2*(nn+1)]=B_ex                         # y-direction x_i+1
            for nn in range(N-1, N):                          # i=N (bottom)
                SW[2*nn][2*(nn-1)+1]=-B_ex                    # x-direction y_N-1
                SW[2*nn][2*nn+1]=B+B_ex*(5-K_ex)+B_de         # x-direction y_N
                SW[2*nn+1][2*(nn-1)]=B_ex                     # y-direction x_N-1
                SW[2*nn+1][2*nn]=-(B+B_ex*(5-K_ex))           # y-direction x_N
    return SW*gamma/(1j)                                      # move iw/|r| to the other side
# define frequncy function-----------------------------------------------------------------------
def Freq(N,SW,n):                                             # N layernumber SW matrix:'Matrix(N,D,k
    omegaGHz=[]                                               # set up list for the frequencies with
    w,v=eig(SW)                                               # obtain eigenvalues and vectors
    omegaRaw = w                                              # extract raw freqencies arrays
    idx1 = np.argsort(omegaRaw)                               # sort the freqencies
    omegaRaw = omegaRaw[idx1]                                 # extract the sorted freqencies index
    omega=np.zeros(N, dtype = 'complex_')                     # set up freqency array with unit Hz
    for ii in range (0,N):                                    # extract the correct freqencies arrays
        omega[ii]=omegaRaw[ii+N]
    omegaGHz.append(omega.real[n]/1e9)                        # convert to GHz
```

```
    return omegaGHz
# define dispersion relation freq. and wavenumber for long wavelength----------------------
# increasing only in kx, define freqency list function
def SWfreqkx(N,D,h,n):                                          # N layernumber D DMI const. h stepsize
    Max=30e6                                                    # maximum wavenumber
    omega1=[]                                                   # set up frequency list
    kx=-Max                                                     # starting k_x
    kz=0                                                        # set k_z = 0
    while kx<Max+1e6:
        omega1.append(Freq(N,Matrix(N,D,kx,kz),n))
        kx=kx+h*1e6                                             # increase steps
    return omega1
# define wavenumber list function
def xSWfreqkx(h):                                               # h stepsize
    Max=30e6
    x=[]
    kx=-Max
    kz=0
    while kx<Max+1e6:
        x.append(kx/1e6)
        kx=kx+h*1e6
    return x
# increasing in both kx and kz, define freqency list function
def SWfreqkxkz(N,D,h,n):
    Max=30e6
    omega1=[]
    kx=-Max
    kz=-Max
    while kx<Max+1e6:                                          # 2 while loops allow kx and kz increase
        while kz<Max+1e6:
            omega1.append(Freq(N,Matrix(N,D,kx,kz),n))
            kx=kx+h*1e6
            kz=kz+h*1e6
    return omega1
# Mode amplitudes in x-direction (y-direction is the same) ---------------------------------------
def ModAmpX(N,SW,n):                                               # N layer number
    w,v=eig(SW)                                                    # obtain eigenvalue and vectors
    omegaRaw=w                                                     # exact the raw eigenvalue array
    idx1 = np.argsort(omegaRaw)                                    # sort raw eigenvalue arrays
    omegaRaw = omegaRaw[idx1]                                      # obtain the sorted array by use new
    v = v[:,idx1]                                                  # sort the eigen vector aby use new
```

```
    EigV=np.zeros((2*N,N),dtype = 'complex_')
    for ii in range (0,N):                                          # obtain the amplitude in x-directio
        for jj in range (0,2*N):
            EigV[jj][ii]=v[jj][ii+N]
    EigVx=np.zeros((N,N),dtype=np.complex128)
    for ii in range (0,N):
        for jj in range (0,N):
            EigVx[ii][jj]=EigV[2*ii][jj]
            if abs(EigVx[ii][jj].real)>abs(EigVx[ii][jj].imag):
                EigVx[ii][jj]=EigVx[ii][jj].real
            else:
                EigVx[ii][jj]=EigVx[ii][jj].imag
    return EigVx[:,n].real
# layer number function --------------------------------------------------------
def xMobAmp(N):
    layers=np.linspace(1,N,N)
    return layers

def xMobthickness(N):
    layers=np.linspace(1,N,N)
    thickness=a*layers*1e9
    return thickness
```

In [4]:

```
# parameters of certain Permalloy form Kostylev's paper--------------------------------------
mu_0=4*np.pi*1e-7                     # permeability of free space
H=300*(1e3/(4*np.pi))                 # external H field
A=1.355e-11                           # exchange constant 1.355e-11 J/m
M_0=1.05/mu_0                         # magnetization in z direction M_s A/M
gamma=(2*np.pi)*2.8e6/((1e3/(4*np.pi))*mu_0)   # gyromagnetic ratio 2.8 MHz/Oe
b=0.248e-9                           # atomic layer thickness
a=0.248e-9                           # lattice constant
B=H*mu_0
```

In [5]:

```
#Matrix(N,D,kx,kz)
#Freq(N,SW,n)
def dw_nr(D,n,Nmax):
    kx1=20e6
    kx2=-20e6
    kz=0
    N=3
    dw_nr=[]
    while N<Nmax:
        sw1=Matrix(N,D,kx1,kz)
        sw2=Matrix(N,D,kx2,kz)
        dw_nr.append( (np.array(Freq(N,sw1,n))-np.array(Freq(N,sw2,n)))/(2*np.pi) )
        N=N+1
    return dw_nr

def xdw_nr(n,Nmax):
    N=3
    inv_n0=[]
    while N<Nmax:
        inv_n0.append( 1/(N*a*1e9) )
        N=N+1
    return inv_n0
```

In [6]:

```
def dw_nr2(D,n,Nmax):
    kx1=20e6
    kx2=-20e6
    N=3
    dw_nr2=[]
    while N<Nmax:
        L=N*b
        NN=N*10
        sw1=MatrixKdem(L,NN,D,kx1)
        sw2=MatrixKdem(L,NN,D,kx2)
        dw_nr2.append( (np.array(FreqK(NN,sw1,n))-np.array(FreqK(NN,sw2,n)))/(2*np.pi) )
        N=N+1
    return dw_nr2

def xdw_nr2(n,Nmax):
    N=3
    inv_n02=[]
    while N<Nmax:
        inv_n02.append( 1/(N*a*1e9) )
        N=N+1
    return inv_n02
```

In [7]:

```


xaxis=xdw_nr(0,101)
#n=0
ALMn0=dw_nr(4.2e-3,0,101)
Kostylevn0=dw_nr2(4.2e-3,0,101)
#n=1
ALMn1=dw_nr(4.2e-3,1,101)
Kostylevn1=dw_nr2(4.2e-3,1,101)
#n=2
ALMn2=dw_nr(4.2e-3,2,101)
Kostylevn2=dw_nr2(4.2e-3,2,101)

```

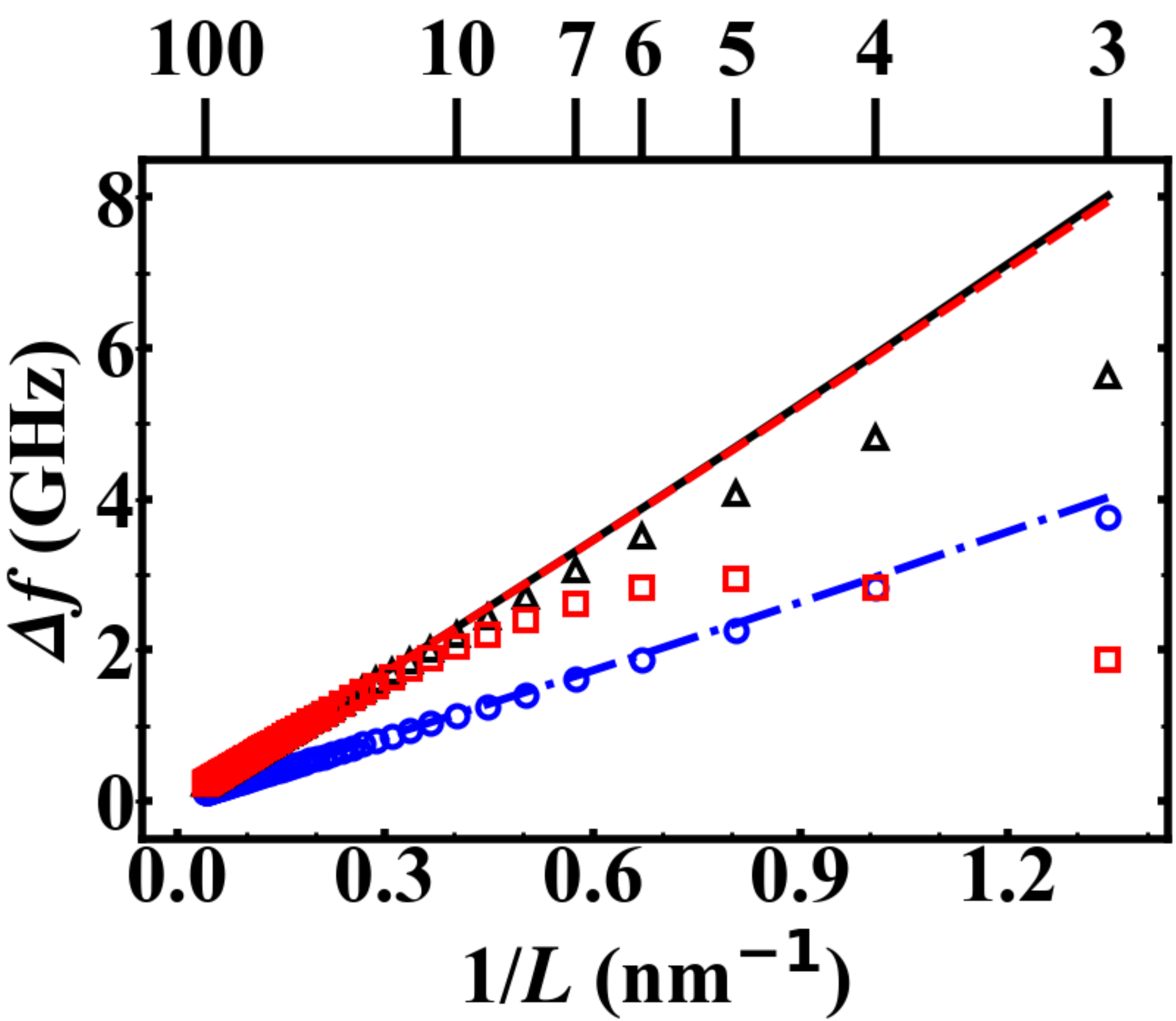
In [9]:

```
plt.rcParams["font.weight"] = "bold"
plt.rcParams["font.family"] = "Times New Roman"
plt.rcParams['mathtext.fontset'] = 'custom'
plt.rcParams['mathtext.it'] = 'STIXGeneral:italic'
plt.rcParams['mathtext.bf'] = 'STIXGeneral:italic:bold'
fig, (ax) = plt.subplots(1, 1, figsize=(9, 6))
ax.plot(xaxis,ALMn0,'o',color='blue',markerfacecolor='none',ms=10,markeredgewidth=3,label='Uniform',li
ax.plot(xaxis,Kostylevn0,'-.',color='blue',label='Kostylev',linewidth=4)
ax.plot(xaxis,ALMn1,'^',color='black',markerfacecolor='none',ms=10,markeredgewidth=3,label='1ex',linew
ax.plot(xaxis,Kostylevn1,'-',color='black',label='Kostylev',linewidth=4)
ax.plot(xaxis,ALMn2,'s',color='red',markerfacecolor='none',ms=10,markeredgewidth=3,label='2nd',linewid
ax.plot(xaxis,Kostylevn2,'--',color='red',label='Kostylev',linewidth=4)
ax.set_xlabel(r'1/$\mathbf{L}$ (nm$^{-1}$)',weight='bold',fontsize=40)
ax.set_ylabel(r'$\mathbf{\Delta f}$ (GHz)',weight='bold',fontsize=40)
ax.tick_params(axis='x',direction='in', labelsize=40)
ax.tick_params(axis='y',direction='in',labelsize=40)
ax.set_ylim([-0.5,8.5])
ax.set_xlim([-0.05,1.43])

ax.set_xticks([0,0.3,0.6,0.9,1.2])
ax.tick_params(axis='x',direction='in',width=4,length=5,labelsize=40)
sec3 = ax.secondary_xaxis(location='bottom')
sec3.set_xticks(np.linspace(0,1.4,15),labels=None)
sec3.tick_params(axis='x',direction='in',width=2,labelbottom=False)
thd3 = ax.secondary_xaxis(location='top')
thd3.set_xticks([1/(100*a*1e9),1/(10*a*1e9),1/(7*a*1e9),1/(6*a*1e9),1/(5*a*1e9),1/(4*a*1e9),1/(3*a*1e9
                ,labels=[100,10,7,6,5,4,3])
thd3.tick_params(axis='x',direction='out',width=4,length=30,labelsize=40)

ax.set_yticks([0,2,4,6,8])
ax.tick_params(axis='y', right=True,direction='in',length=5,width=4,labelsize=40)
sec2 = ax.secondary_yaxis(location=0)
sec2.set_yticks(np.linspace(0,8,9),labels=None)
sec2.tick_params(axis='y',right=True,direction='in',width=2,labelleft=False)
thd2 = ax.secondary_yaxis(location='right')
thd2.set_yticks(np.linspace(0,8,9),labels=None)
thd2.tick_params(axis='y',direction='in',width=2,labelright=False)

plt.setp(ax.spines.values(), lw=4)
plt.show()
```



100
10
7
6
5
4
3
8
6
4
2
0
Δf (GHz)
0.0
0.3
0.6
0.9
1.2
1/L (nm⁻¹)

The x-axis values for inverse thickness in units of inverse-nm:

In [24]:
```
print(xaxis)
```

```
[1.3440860215053763, 1.0080645161290323, 0.8064516129032259, 0.6720430107526881, 0.576036866359447, 0.504
0322580645161, 0.4480286738351254, 0.40322580645161293, 0.3665689149560117, 0.33602150537634407, 0.310173
6972704714, 0.2880184331797235, 0.26881720430107525, 0.25201612903225806, 0.23719165085388993, 0.22401433
69175627, 0.2122241086587436, 0.20161290322580647, 0.19201228878648233, 0.18328445747800584, 0.1753155680
224404, 0.16801075268817203, 0.16129032258064513, 0.1550868486352357, 0.14934289127837513, 0.144009216589
86174, 0.13904338153503892, 0.13440860215053763, 0.13007284079084286, 0.12600806451612903, 0.122189638318
67057, 0.11859582542694497, 0.11520737327188939, 0.11200716845878135, 0.1089799476896251, 0.1061120543293
718, 0.10339123242349048, 0.10080645161290323, 0.0983477576711251, 0.09600614439324116, 0.093773443360840
22, 0.09164222873900292, 0.08960573476702509, 0.0876577840112202, 0.08579272477693892, 0.0840053763440860
2, 0.08229098090849242, 0.08064516129032256, 0.07906388361796331, 0.07754342431761785, 0.0760803408399269
6, 0.07467144563918757, 0.07331378299120234, 0.07200460829493087, 0.07074136955291455, 0.0695216907675194
6, 0.06834335702569709, 0.06720430107526881, 0.06610259122157587, 0.06503642039542143, 0.064004096262160
78, 0.06300403225806452, 0.06203473945409429, 0.06109481915933528, 0.0601829561868079, 0.05929791271347248
4, 0.05843852267414679 4, 0.05760368663594469 4, 0.05679236710586097, 0.05600358422939067 5, 0.0552364118426
8669, 0.05448997384481255, 0.05376344086021505, 0.0530560271646859, 0.05236698785085882, 0.0516956162117
4524, 0.05104124132298897, 0.05040322580645162, 0.04978096375945838, 0.04917387883556255, 0.04858142246404
975, 0.04800307219662058, 0.04743833017077798, 0.04688672168042011, 0.04634779384501297 5, 0.0458211143695
0146, 0.04530627038782167, 0.04480286738351254 4, 0.04431052818149592, 0.0438288920056101, 0.0433576135969
47626, 0.04289636238846946, 0.04244482173174872 5, 0.04200268817204301, 0.04156967076820751, 0.04114549045
424621, 0.04072987943955685, 0.04032258064516128]
```

The quasi-uniform mode (n=0) has frequencies given by the atomic layer model and the continuum model:

In [29]:
```
counter = 0
ALMn0o=[]
Kostylevn0o=[]
while counter< len(ALMn0):
    ALMn0o.append(ALMn0[counter][0])
    Kostylevn0o.append(Kostylevn0[counter][0])
    counter = counter +1

print(ALMn0o)
print(Kostylevn0o)
```

```
[3.753140708661894, 2.814855654949532, 2.2518846792903617, 1.8765707433139556, 1.6084894016884106, 1.4074
284316365233, 1.2510477093456143, 1.1259431605069734, 1.023584919631649, 0.9382864097262502, 0.8661107697
794277, 0.8042459562096967, 0.7506298037142165, 0.7037156883092264, 0.6623208975323694, 0.62552543906618
, 0.5926034004673698, 0.5629734856414575, 0.5361654810041424, 0.5117945805626846, 0.489542901105506, 0.46
914553994782526, 0.4503799788291649, 0.4330579330625703, 0.4170190119835435, 0.40212573787592143, 0.38825
95954647024, 0.3753178715011214, 0.36321110606247975, 0.3518610217082714, 0.34119882917108385, 0.33116383
203552213, 0.3217022706155792, 0.3127663585234008, 0.30431347546286786, 0.29630548748500773, 0.2887081718
115743, 0.28149072794214713, 0.2746253603015266, 0.2680869205113817, 0.26185259957668905, 0.2559016620466
9257, 0.250215215621638, 0.2447760108102138, 0.23956826616273028, 0.23457751535036966, 0.2297904729693445
4, 0.22519491644833758, 0.22077958184962526, 0.2165340716923976, 0.21244877321211156, 0.20851478570128412
, 0.20472385577731225, 0.20106831958548424, 0.19754105108638678, 0.19413541569216838, 0.1908452286182451,
0.18766471739991228, 0.18458848809558956, 0.18161149476042024, 0.17872901182820253, 0.1759366090812669, 0
.17323012893139095, 0.17060566576642192, 0.16805954714569288, 0.16558831665542512, 0.16318871825422795, 0
.16085768196109804, 0.15859231075238106, 0.15638986855109366, 0.1542477692026833, 0.15216356634574027, 0.
1501349440918799, 0.14815970844246434, 0.14623577937329593, 0.14436118352873792, 0.142534047470544, 0.140
75259143293392, 0.1390151235403102, 0.13732003444831786, 0.13566579237163043, 0.13405093846785085, 0.1324
7408254631593, 0.13093389907693737, 0.12942912347397043, 0.1279585486333144, 0.1265210217031679, 0.125115
44107008318, 0.12374075354318524, 0.12239595172355286, 0.12108007154109181, 0.11979218995052078, 0.118531
42277193923, 0.117296922666719, 0.1160878772381506, 0.11490350724898568, 0.113743064946648, 0.11260583248
88142]
[4.021239060896241, 2.9630185224204757, 2.345723268957592, 1.9412884978901837, 1.655805172862192, 1.44352
27376968862, 1.279486343569643, 1.148926794231439, 1.0425449661144481, 0.9541939809241219, 0.879647859828
4327, 0.8159055453889896, 0.760777076769728, 0.7126269135910428, 0.6702089298447581, 0.6325570278348985,
0.5989106625581525, 0.5686629344617045, 0.5413236554397485, 0.5164925801520162, 0.4938396817327837, 0.473
0904000123553, 0.45401445871230817, 0.43641728329904006, 0.4201333402080065, 0.40502091365124154, 0.39095
797058190945, 0.377838858226003, 0.36557164494291244, 0.3540759628196372, 0.3432812449044267, 0.333125275
3577895, 0.32355298958917067, 0.31451547546771896, 0.3059691374003697, 0.29787499308202053, 0.29019807899
81347, 0.28290694555733475, 0.27597322645730826, 0.2693712698755261, 0.2630778213527577, 0.25707175012575
42, 0.2513338121095498, 0.24584644391518948, 0.24059358331207378, 0.2355605121756836, 0.23073371879335813
, 0.22610077672920395, 0.22165023800320702, 0.2173715386570516, 0.21325491502896593, 0.20929132937807654,
0.20547240365695704, 0.2017903603936614, 0.19823796982775002, 0.19480850252546328, 0.19149568684478924, 0
.18829367064884886, 0.1851969868065148, 0.1822005220532252, 0.1792994888091113, 0.17648939965482188, 0.17
376604417590658, 0.17112546790691652, 0.16856395318312706, 0.1660780016693695, 0.1636643184190456, 0.1613
1979731397922, 0.15904150771994874, 0.15682668227454902, 0.15467270566020652, 0.15257710432345514, 0.1505
375369784692, 0.14855178589405021, 0.14661774884502782, 0.1447334316888302, 0.14289694151551158, 0.141106
48029314846, 0.13936033900021016, 0.13765689219324911, 0.13599459294164537, 0.13437196817419977, 0.132787
61428216238, 0.1312401931062149, 0.12972842815442748, 0.12825110108405197, 0.1268070484392057, 0.12539515
85796363, 0.12401436884719408, 0.12266366288271045, 0.12134206812672318, 0.12004865351406213, 0.118782527
2502577, 0.11754283478169551, 0.11632875686335, 0.11513950776109079, 0.11397433355387157, 0.1128325105303
5502]
```

The first (n=1) PSSW mode with one node through the film thickness. Frequencies are:

In [31]:
```
counter = 0
ALMn1o=[]
Kostylevn1o=[]
while counter< len(ALMn1):
    ALMn1o.append(ALMn1[counter][0])
    Kostylevn1o.append(Kostylevn1[counter][0])
    counter = counter +1

print(ALMn1o)
print(Kostylevn1o)
```

```
[5.6297108495427866, 4.805258648906199, 4.073696778584784, 3.501727442870949, 3.0576866750133957, 2.70772
0783052909, 2.426645670928944, 2.1967760316654625, 2.005704379757824, 1.844598037761346, 1.70705011791393
52, 1.5883236015180775, 1.484851991065548, 1.393904754683359, 1.3133592332084447, 1.2415422921770416, 1.1
771184393405356, 1.1190094439729448, 1.066335681909395, 1.0183727111929881, 0.9745186906495495, 0.9342696
296323298, 0.8972003703255287, 0.8629498191005286, 0.8312093642448777, 0.8017137092125912, 0.774233555658
7651, 0.7485697164218632, 0.7245483438127888, 0.702017034895177, 0.6808416318766524, 0.6609035773820642,
0.642097715777165, 0.6243304553760511, 0.6075182243819213, 0.591586167353881, 0.5764670396351436, 0.56210
02656316646, 0.548431133090319, 0.5354101025176091, 0.522992208825593, 0.5111365478523373, 0.49980582770
66594, 0.48896597928350966, 0.4785858158805818, 0.4686367350180492, 0.4590924564813636, 0.449928791563503
57, 0.4411234392605671, 0.43265580581947727, 0.42450684457959265, 0.4166589134948947, 0.40909564810433663
, 0.4018018480332235, 0.3947633753701555, 0.3879670635032587, 0.3814006351747666, 0.37505262868966316, 0.
36891233134667484, 0.3629697192793875, 0.35721540300216487, 0.3516405780353557, 0.3462369800730182, 0.340
996844203543, 0.33591286777199264, 0.33097817650353617, 0.3261862935624428, 0.3215311125415845, 0.317006
8651072716, 0.3126081101114213, 0.3083296988985811, 0.304166761689382, 0.3001146878372737, 0.296169108826
4879, 0.29232588258978354, 0.28858107902915964, 0.28493096663282746, 0.2813720000943363, 0.27790080884441
96, 0.2745141864208288, 0.2712090806046812, 0.267982584259078, 0.2648319268131293, 0.26175446633831195, 0
.25874768217008054, 0.25580916803188, 0.25293662562123376, 0.2501278586223222, 0.24738076711253462, 0.244
69334233362205, 0.24206366179868416, 0.2394898847116278, 0.23697024767425381, 0.23450306066229518, 0.2320
867032494171, 0.22971962106182756, 0.22740032244749422, 0.22512737534478688]
[8.017192294745634, 5.915915345700472, 4.686422307269236, 3.879727703661352, 3.3098409664829886, 2.885871
814389045, 2.558154101909821, 2.2972595912865814, 2.084644818733459, 1.908045373593503, 1.759025986224364
2, 1.6315945101316336, 1.521377228660199, 1.4251070612953023, 1.340294413918394, 1.2650089041567976, 1.19
7730707600225, 1.1372469268424024, 1.0825778399340802, 1.032923438755623, 0.9876240307967178, 0.946130769
7566459, 0.907983313539335, 0.8727926765252537, 0.8402279198424418, 0.8100057135064074, 0.781882072616676
6, 0.7556457570347983, 0.7311129565893673, 0.7081229788643865, 0.6865347256883264, 0.6662237949820848, 0.
6470800822056755, 0.6290057837507996, 0.6119137258128892, 0.5957259584855799, 0.5803725672439184, 0.56579
06635680611, 0.551923524004872, 0.5387198527773053, 0.5261331477688607, 0.5141211533467684, 0.50264538645
44066, 0.4916707248037067, 0.48116504782265784, 0.4710989227131073, 0.4614453290638542, 0.452179416664278
97, 0.44327829192353574, 0.43472082899830766, 0.4264875024232958, 0.4185602383700129, 0.4109222821699517
, 0.403558080332856, 0.39645317437377825, 0.38959410653288884, 0.3829683344631254, 0.3765641547414538, 0.
37037063388145774, 0.3643775460388871, 0.35857531667429915, 0.35295497151641136, 0.3475080902404871, 0.34
22267643804227, 0.33710355901724076, 0.33213147785626784, 0.3273039313540792, 0.3226147075796919, 0.31805
79455490297, 0.31362811078431285, 0.30931997289855334, 0.3051285849753468, 0.3010492646454484, 0.29707757
66142793, 0.29320931659407384, 0.2894404964473589, 0.2857673304703125, 0.2821862227212075, 0.278693755271
2065, 0.27528667731725887, 0.2719618951052763, 0.2687164625382036, 0.2655475724993032, 0.2624525487249029
, 0.2594288382808989, 0.25647400454073316, 0.2535857206294002, 0.2507617633218845, 0.2480000732640387, 0
.24529841995731214, 0.2426550561456961, 0.24006805375257798, 0.2375356292241093, 0.23505607346727397, 0.2
326277480156084, 0.23024908141454442, 0.22791856582139, 0.2256347538413802]
```

The second (n=2) PSSW mode frequencies with 2 nodes:

In [32]:
```
counter = 0
ALMn2o=[]
Kostylevn2o=[]
while counter< len(ALMn2):
    ALMn2o.append(ALMn2[counter][0])
    Kostylevn2o.append(Kostylevn2[counter][0])
    counter = counter +1

print(ALMn2o)
print(Kostylevn2o)
```

```
[1.8765703455263536, 2.8148555015123593, 2.947754921514649, 2.8148554375673442, 2.6113651573829895, 2.402
6293219234383, 2.2094042374923966, 2.036848372271461, 1.884677086632516, 1.750863690589995, 1.63301086075
68354, 1.5288432932497877, 1.4363606974932757, 1.3538603341054924, 1.2799125451479385, 1.213322765036368,
1.1530930778894102, 1.0983879324837567, 1.0485051606525158, 1.0028520936754344, 0.9609261159226563, 0.922
2989098534524, 0.886603704011156, 0.8535249371145441, 0.8227898568518577, 0.7941616661040902, 0.767433908
3960646, 0.7424258480490161, 0.7189786515941343, 0.6969522170076891, 0.6762225290270759, 0.65667944339068
74, 0.6382248224679177, 0.6207709599458643, 0.6042392443167046, 0.5885590205524228, 0.5736666169057855, 0
.5595045098303639, 0.5460206049238622, 0.5331676156876607, 0.5209025250442458, 0.5091861171415397, 0.4979
8256903659877, 0.4872590935923928, 0.47698562626979835, 0.4671345497097342, 0.4576804509041883, 0.4485999
065771418, 0.43987129304531325, 0.43147461737156734, 0.4233913670987617, 0.4156043762295656, 0.4080977054
520157, 0.40085653487148554, 0.393867067775978, 0.38711644411461593, 0.3805926625918682, 0.37428451037572
81, 0.3681814995869866, 0.3622738098019142, 0.3565522359375478, 0.35100814092092975, 0.3456334126553945,
0.3404204248301275, 0.335362001178896, 0.33045138283964337, 0.3256821985126566, 0.3210484371301551, 0.316
54442280969997, 0.31216479185464874, 0.3079044716291287, 0.303758661114199, 0.2997228130021316, 0.2957926
171865041, 0.2919639855225336, 0.2882330377444191, 0.2845960884399976, 0.2810496349913922, 0.277590346398
5666, 0.2742150529120539, 0.27092073640593906, 0.26770452143354007, 0.2645636669054981, 0.261495558343685
6, 0.2584977006662402, 0.2555677114558131, 0.2527033146805594, 0.24990233482891808, 0.24716269142808797,
0.24448239391585871, 0.24185953684174927, 0.23929229537145283, 0.23677892107244206, 0.23431773796236868,
0.23190713879681726, 0.22954558158753832, 0.2272315863257022, 0.2249637319019416]
[7.941656283009398, 5.885624034683465, 4.671376932281783, 3.871195143866261, 3.3045447198896354, 2.882362
0059912662, 2.555709815707347, 2.2954897914855104, 2.083322590877353, 1.9070317418330347, 1.7582319921828
689, 1.6309610770655365, 1.5208638823955853, 1.424685321309475, 1.3399437692164242, 1.2647142802174793, 1
.1974808249716908, 1.1370332100148028, 1.0823936753172194, 1.0327636597794057, 0.9874845539991871, 0.9460
083333227876, 0.9078752866149185, 0.8726969182093948, 0.8401426736322619, 0.8099295273300107, 0.781813737
6286255, 0.7555842602980176, 0.7310574441504669, 0.7080727267085708, 0.6864891166903621, 0.66618230049078
82, 0.6470422472346262, 0.6289712149710266, 0.6118820817708619, 0.5956969415891651, 0.580345917147912, 0.
5657661517047511, 0.5519009490175925, 0.5386990367128265, 0.5261139328685221, 0.514103399340783, 0.502628
9682778048, 0.4916555306382416, 0.4811509774441966, 0.4710858860578315, 0.4614332450090712, 0.45216821197
698026, 0.4432679003359662, 0.43471119042925216, 0.4264785622771971, 0.4185519469522879, 0.41091459418928
41, 0.40355095427048393, 0.39644657234734415, 0.3895879937643023, 0.3829626790018539, 0.3765589271913962,
0.37036580715323103, 0.36437309512530014, 0.3585712184644566, 0.3529512046141061, 0.3475046348116875, 0.3
422236020186542, 0.3371006726104069, 0.33212885147920856, 0.32730155016893747, 0.32261255774265823, 0.318
05601412732765, 0.31362638568668233, 0.30931844280254084, 0.3051272392758736, 0.3010480933866965, 0.29707
65704512645, 0.29320846673798434, 0.2894397946273322, 0.28576676890124186, 0.2821857940623931, 0.27869345
25927324, 0.2752864940797555, 0.2719618251242435, 0.2687164999688212, 0.26554771180130615, 0.262452784653
5978, 0.2594291658652931, 0.25647441906103946, 0.2535862176087502, 0.2507623385054119, 0.2480006566687371
3, 0.24529913960627198, 0.24265584243717045, 0.24006890320423888, 0.23753653850909087, 0.2350570394185104
7, 0.23262876761073947, 0.23025015176304575, 0.2279196841742672, 0.2256359175575821]
```

In [ ]:
```
1
```