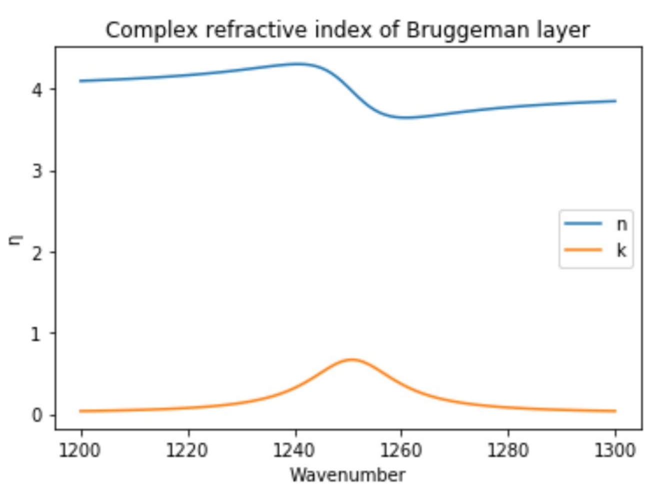
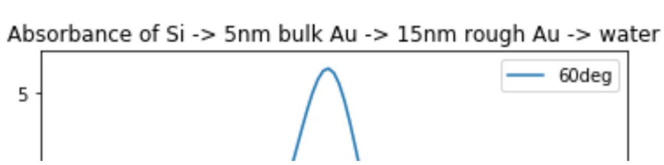
In [1]:

```
# I think it's probably a good idea to clear all variables before running the pro
%reset -f

# Import the classes and methods needed
from fresnel_equations import *

# Can use this function to see if the file has been imported correctly.
# It will return "Import successful." if the import worked.
testImport()

# Now we're off to the races. Let's define the basic parameters for a calculation
mat = materials("frequency", 60, 1200, 1301, 1)

# Define permittivity values for the different materials.
mat.setLorentz("LO", [170], [20], 1.8, [1250])
mat.importMat(["Au", "Water"])
mat.setFixed("Si", 3.4)
mat.setBruggeman("BR", "Au", "LO", "Water", 10, 1, 3, 0.7)

# Plot n and kappa for the Bruggeman layer.
plt.plot(mat.nu, mat.matDict["BR"]["eta"].real, mat.nu, mat.matDict["BR"]["eta"].
plt.legend(["n", "k"])
plt.xlabel("Wavenumber")
plt.ylabel("\u03B7")
plt.title("Complex refractive index of Bruggeman layer")
plt.show()

# Set up a stratified system materials and perform an absorbance calculation
phaseObj = phaseSys(4, mat)
phaseObj.setLayers(["Si", "Au", "BR", "Water"])
phaseObj.setThicknesses([5, 15])
absorbance = phaseObj.calcA('p', "ratioR")

plt.plot(mat.nu, absorbance)
plt.legend(['60deg'])
plt.xlabel("Wavenumber")
plt.ylabel("mAbs")
plt.title("Absorbance of Si -> 5nm bulk Au -> 15nm rough Au -> water")
```

```
Import successful.
```





In [2]:

```
%reset -f
from fresnel_equations import *

# Here I am fitting to reflectivity curves.
# Put the data that you want to fit into a folder titled "fitting_test_cases" wit
# and sub-subfolders "<materialName>". The data file should be named "<thickness>
# (e.g. 60_50.csv for a 60 nm thick sample at an angle of 50 degrees) and the ref
# named "au<aoi>.csv" (e.g. au50.csv for 50 degrees.)
# If you have already calculated a reflectance spectrum, the data file should be

pol = 'p'
aoi = '50'
thickness = '50'
material = 'ito'

# Also input a string; either
    # 'sb' for single beam (and single beam of reference mirror in the same folde
    # 'refl' for already calculated reflectance
spectrum = 'refl'

# The cutoff needs to be such that the arrays of values of eta (data from refract
# contain data (e.g. must be above 1000 cm^-1)
cutoff = 1081

fit = createFit(pol, aoi, thickness, material, spectrum, cutoff)

interact(fit.drude1, vD = (360, 1260, 1), vP = (5000, 10000, 10), e_inf = (3.6, 4
interact(fit.drude2, vD_surface = (360, 1260, 1), vP_surface = (5000, 10000, 10),
```

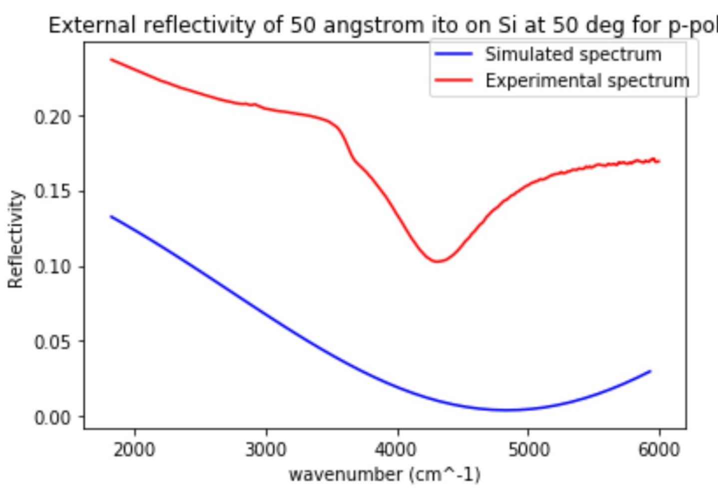
vD 810

vP 7500

e_inf 3.80

thick 285

ang 45.00



vD_surface 810

vP_surface 7500

vD_bulk 595

vP_bulk 7500

In [3]:

```
def drawLine(m,b):
    xvals = np.arange(10, 30, 0.1)
    yvals = np.zeros(200)
    for i in range(len(xvals)):
        yvals[i] = m*xvals[i] + b
    plt.plot(xvals, yvals)

# Note: this program assumes moderate refractive indices and moderate AOIs. It is
# the ray is transmitted through an interface, the next interface it will impinge
# will ignore inclusions which might result in the ray striking the interface thi
# transmitted.

def lineIntersect(m1, b1, m2, b2):
    xIntersect = ((b2 - b1)/(m1 - m2))
    yIntersect = (m1*b2 - m2*b1)/(m1 - m2)
    return [xIntersect, yIntersect]

class ray(object):
    def __init__(self, angle, xPosition):
        self.angleRad = angle*np.pi/180
        self.xPosition = xPosition
        self.slope = np.tan((np.pi/2)-self.angleRad)
        ### NOTE: there could be a problem with this if using a non-zero offset i
        self.offset = -self.slope*self.xPosition

class wafer(object):
    def __init__(self, angle=45, offset=0, slopeLength=5, terrace=1):
        self.angleRad = angle*np.pi/180
        self.slope = np.tan(self.angleRad)
        self.offset = offset
        self.slopeLength = slopeLength
        self.terrace = terrace
        self.period = 2*self.slopeLength + self.terrace


    # This function defines the interface of a grooved wafer
    def waferLine(self, x):
        self.positionWithinPeriod = x % self.period
        self.downSlopeOffset = (self.slope*self.slopeLength)+self.offset
        if self.positionWithinPeriod <= self.slopeLength:
            return self.slope*self.positionWithinPeriod + self.offset
        elif (self.positionWithinPeriod > self.slopeLength) and (self.positionWit
            return (-self.slope*(self.positionWithinPeriod-self.slopeLength)+self
        elif self.positionWithinPeriod > 2*self.slopeLength:
            return self.offset
        else:
            return None

    def intersect(self, ray):
        self.positionWithinPeriod = ray.xPosition % self.period
        self.downSlopeOffset = 2*self.slope*self.slopeLength + self.offset
        periodNumber = int(np.floor(ray.xPosition / self.period))
#         print("Period length: ", self.period, "\nPeriod # ", periodNumber, "\ni
        if (self.positionWithinPeriod > 2*self.slopeLength):
            return ray.xPosition
        else:
            offsetRay = self.offset-(ray.slope*ray.xPosition)
            # Calculate the intersection between the ray and the slopes of the wa
            # translate the line horizonatally to obtain the line equations for t
            relevantUpSlopeOffset = self.offset - (self.slope*(periodNumber*self.
            relevantDownSlopeOffset = self.downSlopeOffset - (-self.slope*(perioc
            upSlopeIntersect = lineIntersect(ray.slope, offsetRay, self.slope, re
            downSlopeIntersect = lineIntersect(ray.slope, offsetRay, -self.slope,

```

```
### TESTING

### testing drawLine()
# drawLine(1,-1)
# drawLine(0.6, 1)
# crossing = lineIntersect(1, -1, 0.6, 1)
# plt.scatter(crossing[0], crossing[1])
# plt.show()

## Testing intersecting rays on wafer interface

#Add this to the wafer.intersect() method:
# drawLine(self.slope, self.offset)
# drawLine(-self.slope, self.downSlopeOffset)
# drawLine(ray.slope, offsetRay)
# plt.legend(["up", "down", "ray"])
# return upSlopeIntersect, downSlopeIntersect

# irubis = wafer(45)
# ray60 = ray(60, 0)
# up, down = irubis.intersect(ray60)
# print("Up slope intercept: ", up, "\nDown slope intercept: ", down)
# plt.scatter(up[0], up[1])
```

In [ ]:

```
```