Odak: an open-source library for wave propagation and Fourier optics calculations

Kaan Akşit

***Abstract*—This paper introduces an implementation of an open-source library implemented under Python programming language [1] that provides built-in functions for wave propagation and Fourier optics calculation. Library provides callable functions for Fraunhofer diffraction and Fresnel diffraction under $odak.diffractions$ class. Library contains also a set of pre-defined aperture types under $odak.aperture$ class. In the near-future it is planned to implement a graphical user interface for end-user. Section IV provides a justification of the implemented methods.**

***Index Terms*—Beam propagation, Fourier optics, Fraunhofer diffraction, Fresnel diffraction, Python, Open-source.**

I. Introduction

THERE are few optics simulation software available on the market and in the open-source world. One of the most known optics simulation software on the market is ZEMAX [2]. In open-source world there are also quality software such as Opus [3]. The full list of available optics simulation programs can be found under [4]. The aim of this project is to provide an optics alternative simulation software. With such an alternative, it will be a lot more easier to implement wave propagation and Fourier optics inside a Python script, thus easiness is provided to Python users. Python is the most preferred programming language for academical purposes beside MATLAB and Octave. Syntaxes of Python is similar in a sense to ones in MATLAB and also provides far more better code flow control. It is also aimed to provide a licence free work bench for different platforms (currently supports only on Linux and Microsoft Windows).

The introduced system in this paper is able to provide calculations for Fourier calculations. But in near future a ray tracing extension with a graphical user interface and ray-tracing ability will be provided to the end-users.

II. How to make it work

A. Under Linux

Once $sample.py$ and $odak.py$ are downloaded from [5] and saved under the same folder; Linux users can download the dependencies of $matplotlib$, $numpy$ and $SciPy$ from their default package managers and they can execute the $sample.py$ using $python\ sample.py$ command under the terminal. Note that running this script will show you the results of the sample questions in Section IV.

B. Under Microsoft Windows

Windows packages of Python 2.7 from [1], $numpy$ from [6], $SciPy$ from [7] and $matplotlib$ from [8] should be downloaded and installed in the targeted computer. Once $sample.py$ and $odak.py$ are downloaded from [5] and saved under the same folder; double clicking on $sample.py$ will directly run or open an Idle window, if Idle window is opened simply press run (F5) from the above menus of the Idle. Note that running this script will show you the results of the sample questions in Section IV.

The author is with the Optical Microsystems Laboratory, Koç University, Istanbul, 34450 TURKEY (e-mail: kaksit@ku.edu.tr).

III. Manual

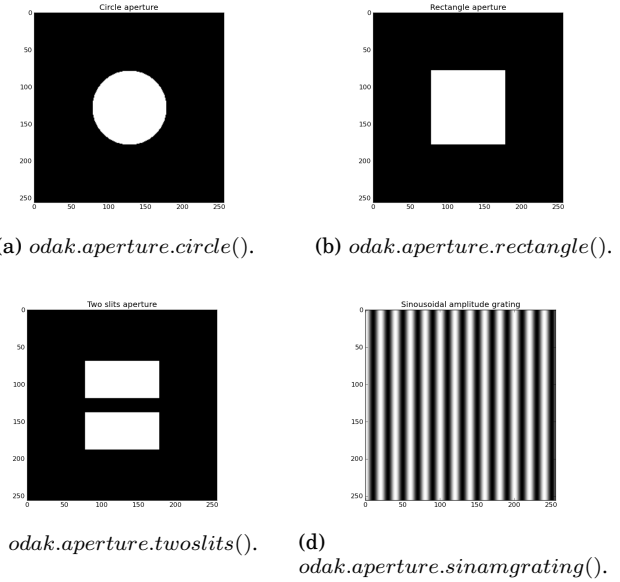
This parts describes available functions inside the library described in this paper.

A. Class:odak.aperture

This class contains pre-defined apertures. Currently available callable aperture function are as in Table III-A.

Available callable aperture functions
$odak.aperture.circle()$
$odak.aperture.lens()$
$odak.aperture.rectangle()$
$odak.aperture.sinamgrating()$
$odak.aperture.twoslits()$

Figure 1 shows different available apertures from $odak.aperture$ class.



(a) $odak.aperture.circle()$. (b) $odak.aperture.rectangle()$.

(c) $odak.aperture.twoslits()$. (d) $odak.aperture.sinamgrating()$.

Fig. 1: Sample apertures.

Explanation of each function is given in subsections provided in below. Note that each variable corresponds to a

single element or a pixel in the created aperture array.

1) odak.aperture.circle(nx,ny,radius): nx defines the width of the whole aperture. ny defines the height of the whole aperture. radius defines the radius of the white region.

2) odak.aperture.lens(self,nx,ny,focal,wavelength): nx defines the width of the whole aperture. ny defines the height of the whole aperture. focal is the focal length in meters of the lens. Wavelength is the wavelength in meters of the input light.

3) odak.aperture.rectangle(nx,ny,side): nx defines the width of the whole aperture. ny defines the height of the whole aperture. side defines one side of the white region.

4) odak.aperture.sinamgrating(self,nx,ny,grating): nx defines the width of the whole aperture. ny defines the height of the whole aperture. Grating is the spatial period in pixels.

5) odak.aperture.twoslits(nx,ny,X,Y,delta): nx defines the width of the whole aperture. ny defines the height of the whole aperture. X defines the width of the each rectangle shaped white region. Y defines the height of the each rectangle shaped white region. delta is the distance between the centroids of the two white rectangle regions.

B. Class:odak.diffractions

This class contains pre-defined callable functions for Fraunhofer and Fresnel diffractions. Currently available callable built-in functions inside $odak.diffractions$ are as in Table III-B.

Available callable aperture functions
$odak.diffractions.fresnelfraunhofer()$
$odak.diffractions.fraunhoferintegral()$
$odak.diffractions.count()$
$odak.diffractions.talbotcalculate()$

1) odak.diffractions.fresnelfraunhofer(obj,wavelength,distance): obj here is an numpy array. wavelength is the wavelength in nano-meters of the input light. Distance here is the distance in meters between the object and the desired plane. This function computes Fraunhofer diffraction of a given input

2) odak.diffractions.fraunhoferintegral(obj): obj here is an numpy array. This function computes 1D FFT of the given 2D array.

3) odak.diffractions.count(obj,x): obj here is an numpy array. This function computes how many x is available inside obj array.

4) odak.diffractions.talbotcalculate(obj,grating,wavelength): grating is the spatial period in meters of the diffraction grating. Wavelength is the wavelength in meters of the incoming light.

IV. Justification

A. Implementation of Fresnel/Fraunhofer diffraction

Fraunhofer diffraction occurs when the statement in Equation 1. Fresnel diffraction occurs when the statement in Equation 2.

$$F = \frac{a^2}{L\lambda} << 1 \tag{1}$$

$$F = \frac{a^2}{L\lambda} \geq 1 \tag{2}$$

Once this condition is fulfilled far-field approximation of Fraunhofer is presented as in Equation 3. Fresnel is represented by Equation 4, Equation 5 and Equation 6.

$$U(x,y) = \frac{e^{ikz}e^{\frac{ik(x^2+y^2)}{2z}}}{i\lambda z}\int\int_{-\infty}^{\infty} u(x',y') e^{-i\frac{2\pi}{\lambda z}(x'x+y'y)}dx'dy' \tag{3}$$

$$E(x,y,z) = \frac{z}{i\lambda}\iint_{-\infty}^{+\infty} E(x',y',0)\frac{e^{ikr}}{r^2}dx'dy' \tag{4}$$

$$r = \sqrt{(x-x')^2+(y-y')^2+z^2} \tag{5}$$

$$k = \frac{2\pi}{\lambda} \tag{6}$$

Similar to Fourier transform and Equation 3 and Equation 4 contains an integral. Analytical solution of this integral is impossible for all but the simplest diffraction geometries. Therefore, it is usually calculated numerically. The integral can be solved as in discrete Fourier transform algorithms. For this purpose $Cooley-Tukey$ algorithm is employed. An implementation of $Cooley-Tukey$ algorithm as in [9] is made and shown in below code, Equation 7 is employed for the implementation. Beside this implementation, built-in $fft2$ command is also applicable.

```
def fraunhoferintegral(self,obj):
    nx,ny  = obj.shape
    result = zeros((nx,ny),dtype=complex)
    for k in range(0,nx):
        for m in range(0,nx/2-1):
            result[k,:] += (obj[2*m,:]*exp
                (-4j*pi*m*k/nx)+exp(-2j*pi
                *k/nx)*obj[2*m+1,:]*exp(-4
                j*pi*m*k/nx))
    return result
```

$$\begin{aligned} X_k = & \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of even-indexed part of } x_m} \\ & + e^{-\frac{2\pi i}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of odd-indexed part of } x_m} \\ & = E_k + e^{-\frac{2\pi i}{N}k} O_k. \end{aligned} \tag{7}$$

After the implementation of the *fraunhoferintegral* function or in other saying 1D DFT algorithm with a 2D input; using the flow chart in Figure 2, Equation 3 is implemented as in below code. Implementation is made using fft approach.

```
def fresnelfraunhofer(self,obj,wavelength,
    distance,pixeltom):
    nx,ny               = obj.shape
    h                   = zeros((nx,ny),dtype
        =complex)
    k                   = 2*pi/wavelength
    distancecritical    = pow(float((nx*ny-
        self.count(abs(obj),0))*pixeltom)
        ,2)/distance/wavelength
    print ('Distance:%s'% distance)
    print ('Critical distance:%s'%
        distancecritical)
    if distance < distancecritical:
        for x in xrange(nx):
            for y in xrange(ny):
                h[x,y] =exp(1j*k*(x^2+y^2)
                    /(2*distance))
        result = fftshift((1/(1j*distance*
            wavelength))*h*self.
            fraunhoferintegral(transpose(
            nx*ny*self.fraunhoferintegral(
            obj*h))))
    else:
        for x in xrange(nx):
            for y in xrange(ny):
                h[x,y] =exp(1j*2*pow(pi,2)
                    *(x^2+y^2)*distance/k)
        result = fftshift(self.
            fraunhoferintegral(transpose(
            nx*ny*self.fraunhoferintegral(
            obj*h))))
    return result
```

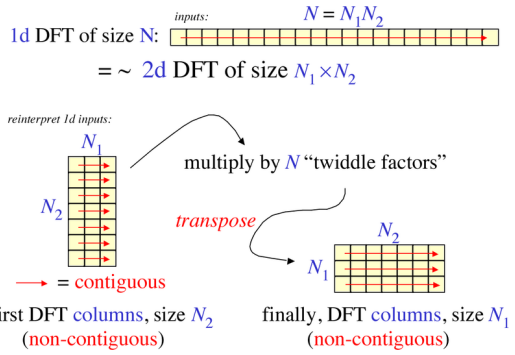


Fig. 2: Way to implement Fraunhofer integral, [9].

B. Sample question I

Sample question I is as follows: Starting with a rectangular aperture of size 1mm, observe the diffraction pattern at different critical propagation distances.

Rectangular aperture with size of 1mm can be examined under Figure 3. For the rest of the calculations wavelength is chosen as 532nm. Below code is used to plot the diffraction patterns at different critical propagation distances.

```
def question1():
    onepxtom    = pow(10,-4)
    aperture    = odak.aperture()
    rectangle   = aperture.rectangle
        (256,256,10)
    aperture.show(rectangle,'Rectangle
        aperture')
    diffrac     = odak.diffractions()
    distance    = 100
    wavelength  = 532*pow(10,-9)
    output      = diffrac.fresnelfraunhofer(1*
        rectangle,wavelength,distance,onepxtom
        )
    aperture.show(abs(output),'Fraunhofer
        diffraction at distance of %sm with %
        snm wavelength'% (distance,wavelength)
        )
    nx,ny       = output.shape
    intensity   = []
    for i in range(0,nx):
        intensity.append(sum(abs(output[i,:]))
            /ny)
    plt.figure()
    plt.plot(intensity)
    plt.show()
    return True
```

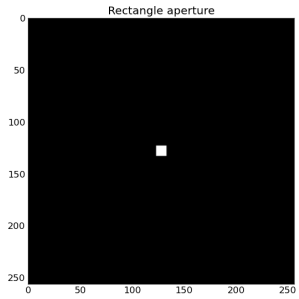


Fig. 3: Rectangular aperture with size of 1mm. Here 10 pixel corresponds to 1mm.

1) Fresnel/Fraunhofer diffraction patterns at different distances for sample question I: Distances of each Fraunhofer diffraction pattern is written on top of the figures. After each figure an intensity profile is provided as the next figure.

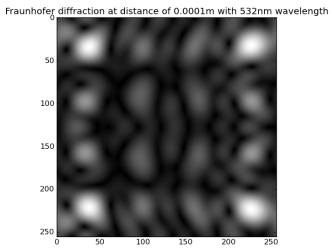


Fig. 4: Fraunhofer diffraction pattern

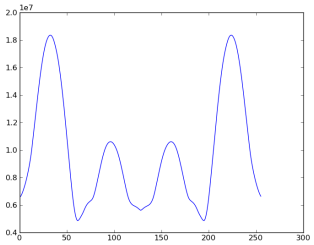


Fig. 5: Intensity profile of Fraunhofer diffraction pattern

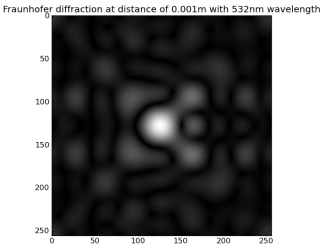


Fig. 6: Fraunhofer diffraction pattern

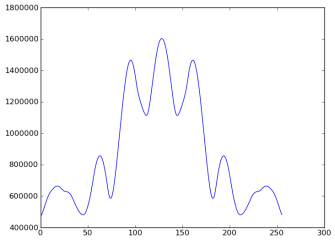


Fig. 7: Intensity profile of Fraunhofer diffraction pattern

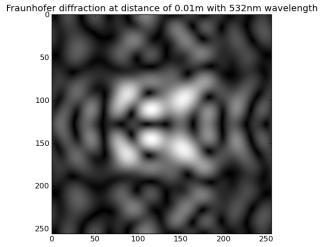


Fig. 8: Fraunhofer diffraction pattern

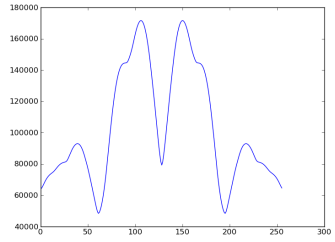


Fig. 9: Intensity profile of Fraunhofer diffraction pattern

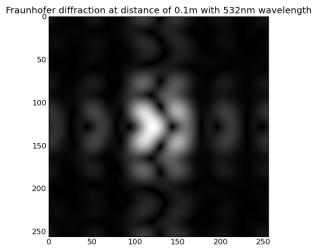


Fig. 10: Fraunhofer diffraction pattern

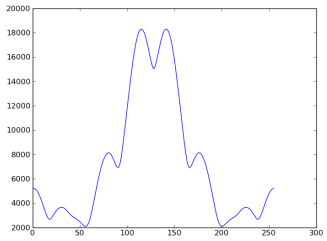


Fig. 11: Intensity profile of Fraunhofer diffraction pattern at 0.1m distance

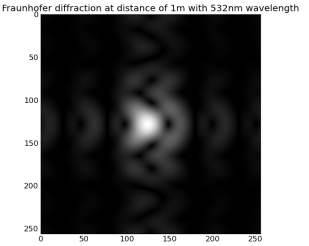


Fig. 12: Fraunhofer diffraction pattern at 1m distance

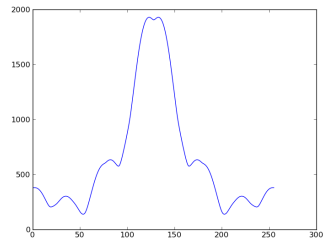


Fig. 13: Intensity profile of Fraunhofer diffraction pattern

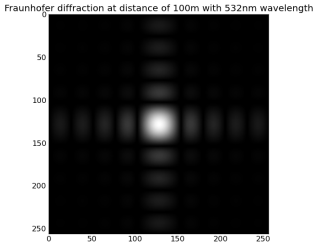


Fig. 14: Fraunhofer diffraction pattern at 100m distance

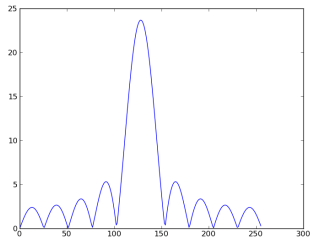


Fig. 15: Intensity profile of Fraunhofer diffraction pattern

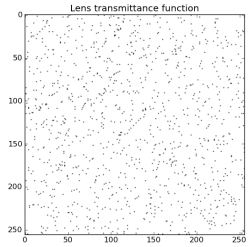


Fig. 16: Lens transmittance function.

Here are the outputs of the diffraction pattern under different distances:

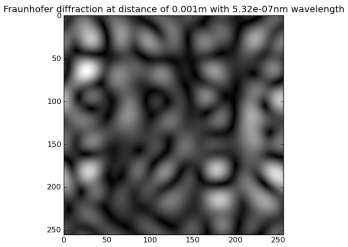


Fig. 17: Diffraction pattern.

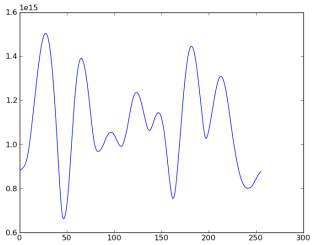


Fig. 18: Diffraction pattern cross-section.

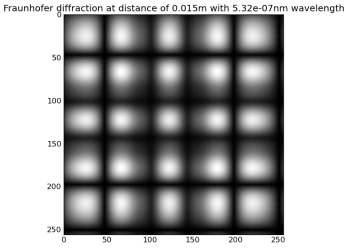


Fig. 19: Diffraction pattern.

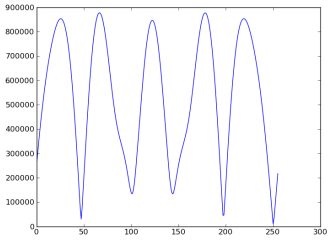


Fig. 20: Diffraction pattern cross-section.

C. Sample question II

Sample question II is as follows: Assume a converging spherical beam illuminates an object pattern of your choice (let's say a triangular or rectangular shaped binary object). Assume the object size is 1mm and the spherical wave is converging to a point 15mm behind the object. Find the irradiance profile at the focus of the spherical wave (15mm away from the object) and also at 30mm away from the object. Briefly explain what you observe.

Same aperture as in Figure 3 is used during the solution of this sample question. To focus the beam a lens is employed at the right position with the focal length of 15m. Lens transmittance function is calculated by implemented function called *odak.aperture.lens*(), see Figure 16 for lens transmittance function.

```
def question1():
    onepxtom    = pow(10,-4)
    aperture    = odak.aperture()
    rectangle   = aperture.rectangle
        (256,256,10)
    aperture.show(rectangle,'Rectangle␣
        aperture')
    diffrac     = odak.diffractions()
    distance    = 100
    wavelength  = 532*pow(10,-9)
    output      = diffrac.fresnelfraunhofer(1*
        rectangle,wavelength,distance,onepxtom
        )
    aperture.show(abs(output),'Fraunhofer␣
        diffraction␣at␣distance␣of␣%s␣m␣with␣%
        s␣nm␣wavelength'% (distance,wavelength)
        )
    nx,ny       = output.shape
    intensity   = []
    for i in range(0,nx):
        intensity.append(sum(abs(output[i,:]))
            /ny)
    plt.figure()
    plt.plot(intensity)
    plt.show()
    return True
```

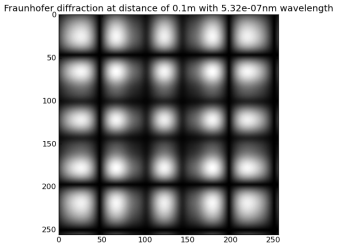


Fig. 21: Diffraction pattern.

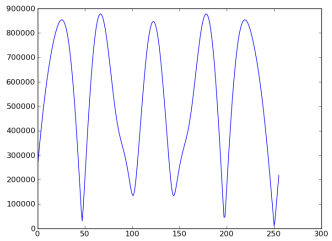


Fig. 22: Diffraction pattern cross-section.

D. Sample question III

Observe the diffraction patterns after sinusoidal amplitude grating at different critical propagation distances (e.g. at different Talbot images). Comment on the results.

Talbot distance is calculated using Equation 8. For $\lambda = 532$nm and $\Lambda = 2\ 10^{-7}$um. As discussed in [10], a rectangular aperture is multiplied with Equation 9 to form a sinusoidal amplitude grating.

$$T = \frac{2\Lambda^2}{\lambda} \tag{8}$$

$$t_A(x,y) = \frac{1}{2} + \frac{1}{2}\cos(2\pi x \frac{1}{\Lambda}) \tag{9}$$

After multiplying the rectangle form Figure 21 is derived.

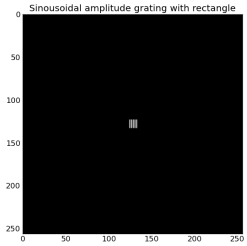


Fig. 23: Derived sinusoidal amplitude grating.

Below code is used to compute the sample question III.

```
def question3():
    onepxtom    = pow(10,-7)
    aperture    = odak.aperture()
    grating     = aperture.sinamgrating
        (256,256,2)
    rectangle   = aperture.rectangle
        (256,256,10)
    obj         = rectangle*grating
    aperture.show(abs(obj),'Sinousoidal
        amplitude grating with rectangle')
    diffrac     = odak.diffractions()
    wavelength  = 532*pow(10,-9)
    distance    = diffrac.talbotcalculate(pow
        (10,-6),wavelength)#/500
    output      = diffrac.fresnelfraunhofer(obj
        ,wavelength,distance,onepxtom)
    aperture.show(abs(output),'Fraunhofer
        diffraction at distance of %sm with %
        snm wavelength'% (distance,wavelength)
        )
    nx,ny       = output.shape
    intensity   = []
    for i in range(0,nx):
        intensity.append(sum(abs(output[i,:]))
            /ny)
    plt.figure()
    plt.plot(intensity)
    plt.show()
    return True
```

In different Talbot distances below Figures are derived using the above code.

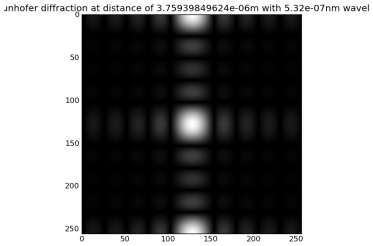


Fig. 24: Derived sinusoidal amplitude grating diffraction pattern.

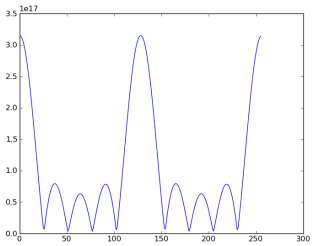


Fig. 25: Derived sinusoidal amplitude grating diffraction pattern cross-section.

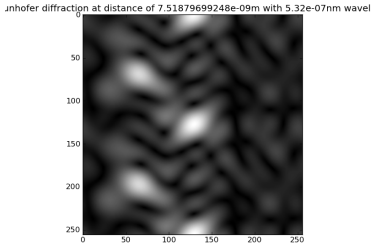


Fig. 26: Derived sinusoidal amplitude grating diffraction pattern.

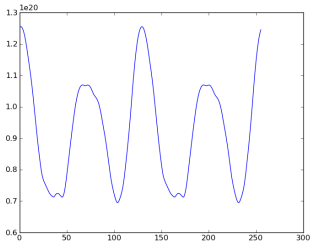


Fig. 27: Derived sinusoidal amplitude grating diffraction pattern cross-section.

Given patterns in above are similar to the ones in [10], therefore it is verified.

V. Conclusion

An open-source optics simulation software is presented, promoted and verified by presenting its solution to a set of known questions. The software has more items to work on it. Such as a graphical user interface and ray tracing support. In the near future it is expected to implement additional features. One can find most recent information about the discussed software under [5]. Enjoy the power of free and open-source software!

Appendix A
odak.py

```
import sys,matplotlib
import matplotlib.pyplot as plt
from scipy.signal import *
from numpy import *
from numpy.fft import *
from cmath import *

class aperture():
    def __init__(self):
        return
    def twoslits(self,nx,ny,X,Y,delta):
        obj=zeros((nx,ny),dtype=uint8)
        for i in range(int(nx/2-X/2),int(nx/2+
            X/2)):
            for j in range(int(ny/2+delta/2-Y
                /2),int(ny/2+delta/2+Y/2)):
                obj[ny/2-abs(ny/2-j),i] = 1
                obj[j,i] = 1
        return obj
    def rectangle(self,nx,ny,side):
        obj=zeros((nx,ny),dtype=uint8)
        for i in range(int(nx/2-side/2),int(nx
            /2+side/2)):
            for j in range(int(ny/2-side/2),
                int(ny/2+side/2)):
                obj[j,i] = 1
        return obj
    def circle(self,nx,ny,radius):
        obj=zeros((nx,ny),dtype=uint8)
        for i in range(int(nx/2-radius/2),int(
            nx/2+radius/2)):
            for j in range(int(ny/2-radius/2),
                int(ny/2+radius/2)):
                if (abs(i-nx/2)**2+abs(j-ny/2)
                    **2)**(0.5)< radius/2:
                    obj[j,i] = 1
        return obj
    def sinamgrating(self,nx,ny,grating):
        obj=zeros((nx,ny),dtype=complex)
        for i in xrange(nx):
            for j in xrange(ny):
                obj[i,j] = 0.5+0.5*cos(2*pi*j/
                    grating)
        return obj
    def lens(self,nx,ny,focal,wavelength):
        obj = zeros((nx,ny),dtype=complex)
        k   = 2*pi/wavelength
        for i in xrange(nx):
            for j in xrange(ny):
                obj[i,j] = exp(-1j*k*(pow(i,2)
                    +pow(j,2))/2/focal)
        return obj
    def show(self,obj,title='Aperture'):
        plt.figure(),plt.title(title),plt.
            imshow(obj,cmap=matplotlib.cm.gray
            ),plt.show()
        return True

class diffractions():
    def __init__(selfparams):
        return
    def fresnelfraunhofer(self,obj,wavelength,
        distance,pixeltom):
        nx,ny               = obj.shape
        h                   = zeros((nx,ny),dtype
            =complex)
        k                   = 2*pi/wavelength
        distancecritical = pow(float((nx*ny-
            self.count(abs(obj),0))*pixeltom)
            ,2)/distance/wavelength
        print ('Distance:%s'% distance)
        print ('Critical distance:%s'%
            distancecritical)
        if distance < distancecritical:
            for x in xrange(nx):
                for y in xrange(ny):
                    h[x,y] =exp(1j*k*(x^2+y^2)
                        /(2*distance))
            result = fftshift((1/(1j*distance*
                wavelength))*h*self.
                fraunhoferintegral(transpose(
                nx*ny*self.fraunhoferintegral(
                obj*h))))
```

```
        else:
            for x in xrange(nx):
                for y in xrange(ny):
                    h[x,y] =exp(1j*2*pow(pi,2)
                        *(x^2+y^2)*distance/k)
            result = fftshift(self.
                fraunhoferintegral(transpose(
                nx*ny*self.fraunhoferintegral(
                obj*h))))
        return result
    def fraunhoferintegral(self,obj):
        nx,ny  = obj.shape
        result = zeros((nx,ny),dtype=complex)
        for k in range(0,nx):
            for m in range(0,nx/2-1):
                result[k,:] += (obj[2*m,:]*exp
                    (-4j*pi*m*k/nx)+exp(-2j*pi
                    *k/nx)*obj[2*m+1,:]*exp(-4
                    j*pi*m*k/nx))
        return result
    def count(self,obj,number):
        obj    = abs(obj)
        result = 0
        for i in obj:
            for j in i:
                if j == number:
                    result += 1
        return result
    def talbotcalculate(self,grating,
        wavelength):
        result = float(2*pow(grating,2))/
            wavelength
        return result


class lens():
    def __init__(selfparams):
        return
    def calculatefocal(r1,r2,n,d):
        a=(n-1)*(1./r1+1./r2+(n-1)*d/(n*r1*r2)
            )
        f=pow(a,-1)
        return f

def main():
    print 'Odak'
    return True

if __name__ == '__main__':
    sys.exit(main())
```

APPENDIX B
SAMPLE.PY

```
import sys,odak
import matplotlib.pyplot as plt
from numpy import *

def sample():
    question1()
    question2()
    question3()
    return

def question1():
    onepxtom   = pow(10,-4)
    aperture   = odak.aperture()
    rectangle  = aperture.rectangle
        (256,256,10)
    aperture.show(rectangle,'Rectangle␣
        aperture')
    diffrac    = odak.diffractions()
    distance   = 100
    wavelength = 532*pow(10,-9)
    output     = diffrac.fresnelfraunhofer(1*
        rectangle,wavelength,distance,onepxtom
        )
    aperture.show(abs(output),'Fraunhofer␣
        diffraction␣at␣distance␣of␣%sm␣with␣%
        snm␣wavelength'% (distance,wavelength)
        )
    nx,ny      = output.shape
    intensity  = []
    for i in range(0,nx):
        intensity.append(sum(abs(output[i,:]))
            /ny)
    plt.figure()
    plt.plot(intensity)
    plt.show()
    return True

def question3():
    onepxtom   = pow(10,-7)
    aperture   = odak.aperture()
    grating    = aperture.sinamgrating
        (256,256,2)
    rectangle  = aperture.rectangle
        (256,256,10)
    obj        = rectangle*grating
    aperture.show(abs(obj),'Sinousoidal␣
        amplitude␣grating␣with␣rectangle')
    diffrac    = odak.diffractions()
    wavelength = 532*pow(10,-9)
    distance   = diffrac.talbotcalculate(pow
        (10,-6),wavelength)#/500
    output     = diffrac.fresnelfraunhofer(obj
        ,wavelength,distance,onepxtom)
    aperture.show(abs(output),'Fraunhofer␣
        diffraction␣at␣distance␣of␣%sm␣with␣%
        snm␣wavelength'% (distance,wavelength)
        )
    nx,ny      = output.shape
    intensity  = []
    for i in range(0,nx):
        intensity.append(sum(abs(output[i,:]))
            /ny)
    plt.figure()
    plt.plot(intensity)
    plt.show()
    return True

def question2():
    onepxtom   = pow(10,-7)
    aperture   = odak.aperture()
    rectangle  = aperture.rectangle
        (256,256,10)
```

```
    diffrac     = odak.diffractions()
    distance    = 1*pow(10,-3)
    focal       = 15*pow(10,-3)
    wavelength  = 532*pow(10,-9)
    lenstrans   = aperture.lens(256,256,focal,
        wavelength)
    aperture.show(abs(lenstrans),'Lens␣
        transmittance␣function')
    obj         = rectangle*lenstrans
    output      = diffrac.fresnelfraunhofer(obj
        ,wavelength,distance,onepxtom)
    aperture.show(abs(output),'Fraunhofer␣
        diffraction␣at␣distance␣of␣%sm␣with␣%
        snm␣wavelength'% (distance,wavelength)
        )
    nx,ny       = output.shape
    intensity   = []
    for i in range(0,nx):
        intensity.append(sum(abs(output[i,:]))
            /ny)
    plt.figure()
    plt.plot(intensity)
    plt.show()
    return True

if __name__ == '__main__':
    sys.exit(sample())
```

REFERENCES

[1] "Python programming language - Official Website," April 2011. [online] http://www.python.org/.

[2] "Home - ZEMAX: Software for Optical System Design," April 2011. [online] http://www.zemax.com/.

[3] "opus - optical design software," April 2011. [online] http://www.maxreason.com/software/optics/opus.html.

[4] "OpTaliX: Optical Design Software," April 2011. [online] http://www.optenso.com/links/links.html.

[5] "odak - Wave optics and ray tracing simulation software - Google Project Hosting," April 2011. [online] http://code.google.com/p/odak/.

[6] "scientific-computing-tools-for-python for Python," April 2011. [online] http://numpy.scipy.org/.

[7] "SciPy -," April 2011. [online] http://www.scipy.org/.

[8] "matplotlib: python plotting - Matplotlib v1.0.1 documentation," April 2011. [online] http://matplotlib.sourceforge.net/.

[9] "Cooley-Tukey FFT algorith- Wikipedia, the free encyclopaedia," April 2011. [online] http://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm.

[10] J. Goodman, *Introduction to Fourier optics*. Roberts & Company Publishers, 2005.