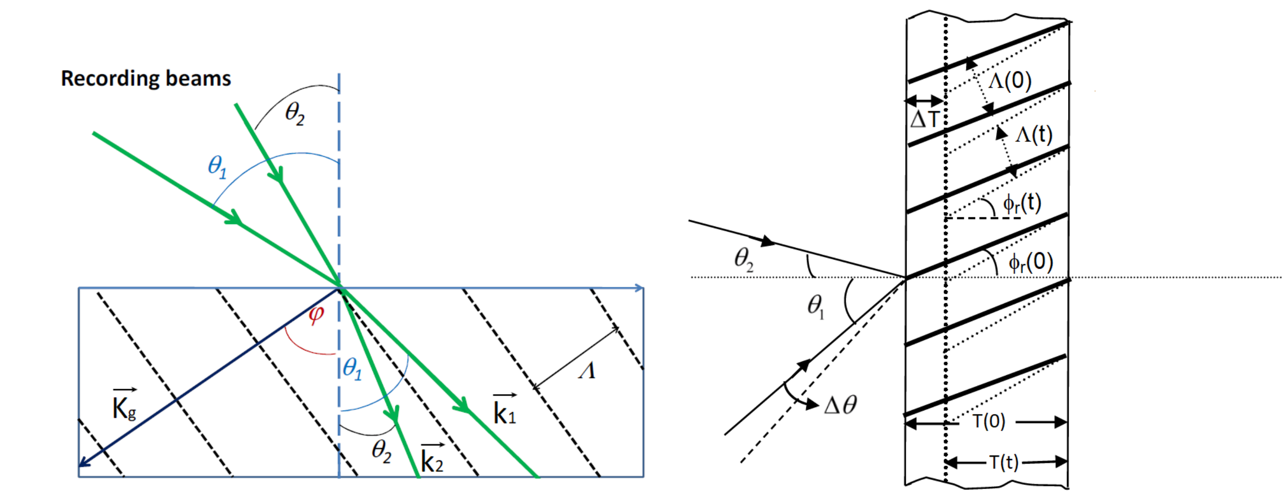
A Summary of my PhD Thesis: *Mathematical Modelling of Hybrid Photonic Structures for Holographic Sensors*

Jack Lyons, PhD

Motivation & Research Questions



(a) Set up

(b) Geometry

Figure 1: The optical set up and geometry for a holographic grating.

The motivation for this research project was as follows:

- Optical properties of holographic gratings are sensitive to external stimulus and can be exploited for the purpose of environmental sensing.
- Hybrid photopolymers are a strong candidate as recording media for holographic gratings because they offer a wide dynamic range and good selectivity.
- A mathematical framework modelling the formation and operation of a holographic sensor is needed to optimize the design.

The successful completion of this research was characterized by the ability to answer the following research questions.

1. How can we use the ...
   (a) host photopolymer material properties (monomer, binder matrix, dye, etc.)
   (b) recording conditions (recording intensity, spatial frequency, etc.)
   (c) nanoparticle properties (initial concentration, refractive index, etc.)

   in order to control the final grating in a hybrid polymer system and hence optimize the functionality of a holographic sensor?
2. Can the theoretical model predict IEO experimental results:
   (a) Significantly increased dynamic range over conventional photopolymer media.

(b) Non-linear response of refractive index modulation to increased doping.

(c) Photopolymerization-induced shrinkage is significantly reduced by the addition of zeolite nanoparticles.

(d) Increased shrinkage at high spatial frequencies.

Mathematical Model

In black, the previous model and in blue, the improvements I developed.

$$\frac{\partial b}{\partial t} = 0, \tag{1a}$$
$$\frac{\partial m}{\partial t} + \nabla \cdot \vec{J_m} = -\Phi(t)F(x,y,t)m, \tag{1b}$$
$$\frac{\partial p}{\partial t} + \nabla \cdot \vec{J_p} = \Phi(t)F(x,y,t)m - \Phi(t)\Gamma p^2, \tag{1c}$$
$$\frac{\partial q}{\partial t} = \Phi(t)\Gamma p^2, \tag{1d}$$
$$\frac{\partial z}{\partial t} + \nabla \cdot \vec{J_z} = 0. \tag{1e}$$

$$F(x,y,t) = k_p \left[I_0 e^{-\zeta(T-y)}\right]^a \left\{1 + e^{-\xi z} \cos\left[\frac{2\pi \cos\phi_r(t)}{\Lambda(t)}x - \frac{2\pi \sin\phi_r(t)}{\Lambda(t)}y\right]\right\}, \tag{2}$$

$$\vec{J_m} = -D_m \frac{\partial m}{\partial x}\vec{i} - D_m \frac{\partial m}{\partial y}\vec{j}, \tag{3a}$$
$$\vec{J_p} = -D_p \left\{\left[\frac{\partial p}{\partial x}\vec{i} + \frac{\partial p}{\partial y}\vec{j}\right] + \epsilon_z \left[\frac{\partial (pz)}{\partial x}\vec{i} + \frac{\partial (pz)}{\partial y}\vec{j}\right]\right\}, \tag{3b}$$
$$\vec{J_z} = -D_z \left\{\left[\frac{\partial z}{\partial x}\vec{i} + \frac{\partial z}{\partial y}\vec{j}\right] + \epsilon_z \left[\frac{\partial (pz)}{\partial x}\vec{i} + \frac{\partial (pz)}{\partial y}\vec{j}\right] + \epsilon_z \left[\frac{\partial (qz)}{\partial x}\vec{i} + \frac{\partial (qz)}{\partial y}\vec{j}\right]\right\}. \tag{3c}$$

$$0 \le x \le \hat{x}, \qquad 0 \le y \le T(t), \qquad t \ge 0. \tag{4}$$

Boundary Immobilization

$$\frac{\partial B}{\partial t} = \frac{Y}{u}\frac{du}{dt}\frac{\partial B}{\partial Y}, \tag{5a}$$
$$\frac{\partial M}{\partial t} = \frac{Y}{u}\frac{du}{dt}\frac{\partial M}{\partial Y} + \alpha_m^{(x)}\frac{\partial^2 m}{\partial x^2} + \alpha_m^{(y)}\frac{1}{u^2}\frac{\partial^2 M}{\partial Y^2} - \Phi(t)\beta F^*(x,Y,t)M, \tag{5b}$$
$$\frac{\partial P}{\partial t} = \frac{Y}{u}\frac{du}{dt}\frac{\partial P}{\partial Y} + \alpha_p^{(x)}\frac{\partial^2 P}{\partial x^2} + \alpha_p^{(y)}\frac{1}{u^2}\frac{\partial^2 P}{\partial Y^2} + \alpha_{pz}^{(x)}\frac{\partial^2 (PZ)}{\partial x^2} + \alpha_{pz}^{(y)}\frac{1}{u^2}\frac{\partial^2 (PZ)}{\partial Y^2} + \Phi\beta F^*(x,Y,t)M - \Phi(t)\gamma P^2, \tag{5c}$$
$$\frac{\partial Q}{\partial t} = \frac{Y}{u}\frac{du}{dt}\frac{\partial Q}{\partial Y} + \Phi(t)\gamma P^2, \tag{5d}$$
$$\frac{\partial Z}{\partial t} = \frac{Y}{u}\frac{du}{dt}\frac{\partial Z}{\partial Y} + \alpha_z^{(x)}\frac{\partial^2 Z}{\partial x^2} + \alpha_z^{(y)}\frac{1}{u^2}\frac{\partial^2 Z}{\partial Y^2} + \alpha_{pz}^{(x)}\frac{\partial^2 (PZ)}{\partial x^2} + \frac{1}{u^2}\alpha_{pz}^{(y)}\frac{\partial^2 (PZ)}{\partial Y^2} + \alpha_{qz}^{(x)}\frac{\partial^2 (QZ)}{\partial x^2} + \alpha_{qz}^{(y)}\frac{1}{u^2}\frac{\partial^2 (QZ)}{\partial Y^2}, \tag{5e}$$

$$F^*(x,Y,t) = e^{-a\zeta^* u(1-Y)}\left\{1 + e^{-\xi^* Z}\cos\left[2\pi\left(x - \frac{T_0}{\hat{x}}\tan\phi_r uY\right)\right]\right\} \tag{6}$$

Initial & Boundary Conditions

$$
\begin{aligned}
M(x,Y,0) &= 1, & P(x,Y,0) &= 0, & Q(x,Y,0) &= 0, & Z(x,Y,0) &= 1,\\
B(x,Y,0) &= 1, & u(0) &= 1, & u'(0) &= 0.
\end{aligned} \tag{7}
$$

$$\frac{\partial^n M}{\partial x^n}(0,Y,t) = \frac{\partial^n M}{\partial x^n}(1,Y,t) \qquad n = \{0,1,2,...\}, \tag{8a}$$

$$\frac{\partial^n P}{\partial x^n}(0,Y,t) = \frac{\partial^n P}{\partial x^n}(1,Y,t) \qquad n = \{0,1,2,...\}, \tag{8b}$$

$$\frac{\partial^n Z}{\partial x^n}(0,Y,t) = \frac{\partial^n Z}{\partial x^n}(1,Y,t) \qquad n = \{0,1,2,...\}. \tag{8c}$$

$$\frac{\partial M}{\partial Y}(x,0,t) = \frac{\partial P}{\partial Y}(x,0,t) = \frac{\partial Q}{\partial Y}(x,0,t) = \frac{\partial Z}{\partial Y}(x,0,t) = \frac{\partial B}{\partial Y}(x,0,t) = 0, \tag{8d}$$

$$\frac{\partial M}{\partial Y}(x,1,t) = \frac{\partial P}{\partial Y}(x,1,t) = \frac{\partial Q}{\partial Y}(x,1,t) = \frac{\partial Z}{\partial Y}(x,1,t) = \frac{\partial B}{\partial Y}(x,1,t) = 0. \tag{8e}$$

Numerical Scheme

Numerical simulation can be done using the Crank-Nicolson implicit finite-difference scheme. For example Eqn. 5c would be ...

$$
\begin{aligned}
\frac{M_{i,j}^{k+1} - M_{i,j}^{k}}{\Delta t} =& \frac{Y_j}{u_k}\frac{u_k - u_{k-1}}{\Delta t}\left(\frac{M_{i,j+1}^{k+1} - M_{i,j-1}^{k+1}}{4\Delta Y} + \frac{M_{i,j+1}^{k} - M_{i,j-1}^{k}}{4\Delta Y}\right) + \\
& \frac{\alpha_{mm}}{2}\left[\frac{M_{i-1,j}^{k+1} - 2M_{i,j}^{k+1} + M_{i+1,j}^{k+1}}{\Delta x^2} + \frac{M_{i-1,j}^{k} - 2M_{i,j}^{k} + M_{i+1,j}^{k}}{\Delta x^2}\right] + \\
& \frac{\alpha_{mm}}{2u_k^2}\left[\frac{M_{i,j-1}^{k+1} - 2M_{i,j}^{k+1} + M_{i,j+1}^{k+1}}{\Delta Y^2} + \frac{M_{i,j-1}^{k} - 2M_{i,j}^{k} + M_{i,j+1}^{k}}{\Delta Y^2}\right] + \\
& - \Phi^k \beta F_{i,j}^k \left(\frac{M_{i,j}^{k+1} + M_{i,j}^{k}}{2}\right),
\end{aligned} \tag{9a}
$$

$$
\begin{aligned}
\frac{P_{i,j}^{k+1} - P_{i,j}^{k}}{\Delta t} =& \frac{Y_j}{u_k}\frac{u_k - u_{k-1}}{\Delta t}\left(\frac{P_{i,j+1}^{k+1} - M_{i,j-1}^{k+1}}{4\Delta Y} + \frac{P_{i,j+1}^{k} - M_{i,j-1}^{k}}{4\Delta Y}\right) + \\
& \frac{\alpha_{pp}}{2}\left[\frac{P_{i-1,j}^{k+1} - 2P_{i,j}^{k+1} + P_{i+1,j}^{k+1}}{\Delta x^2} + \frac{P_{i-1,j}^{k} - 2P_{i,j}^{k} + P_{i+1,j}^{k}}{\Delta x^2}\right] + \\
& \frac{\alpha_{pp}}{2u_k^2}\left[\frac{P_{i,j-1}^{k+1} - 2P_{i,j}^{k+1} + P_{i,j+1}^{k+1}}{\Delta Y^2} + \frac{P_{i,j-1}^{k} - 2P_{i,j}^{k} + P_{i,j+1}^{k}}{\Delta Y^2}\right] + \\
& \frac{\alpha_{pz}}{2}\left[\frac{Z_{i-1,j}^{k}P_{i-1,j}^{k+1} - 2Z_{i,j}^{k}P_{i,j}^{k+1} + Z_{i+1,j}^{k}P_{i+1,j}^{k+1}}{\Delta x^2} + \right. \\
& \left. \frac{Z_{i-1,j}^{k}P_{i-1,j}^{k} - 2Z_{i,j}^{k}P_{i,j}^{k} + Z_{i+1,j}^{k}P_{i+1,j}^{k}}{\Delta x^2}\right] + \\
& \frac{\alpha_{pz}}{2u_k^2}\left[\frac{Z_{i,j-1}^{k}P_{i,j-1}^{k+1} - 2Z_{i,j}^{k}P_{i,j}^{k+1} + Z_{i,j+1}^{k}P_{i,j+1}^{k+1}}{\Delta Y^2} + \right. \\
& \left. \frac{Z_{i,j-1}^{k}P_{i,j-1}^{k} - 2Z_{i,j}^{k}P_{i,j}^{k} + Z_{i,j+1}^{k}P_{i,j+1}^{k}}{\Delta Y^2}\right] + \\
& \Phi^k \beta F_{i,j}^k \left(\frac{M_{i,j}^{k+1} + M_{i,j}^{k}}{2}\right) - \Phi^k \gamma P_{i,j}^k \left(\frac{P_{i,j}^{k} + P_{i,j}^{k+1}}{2}\right),
\end{aligned} \tag{9b}
$$

$$\frac{Q_{i,j}^{k+1}-Q_{i,j}^{k}}{\Delta t} = \frac{Y_j}{u_k}\frac{u_k-u_{k-1}}{\Delta t}\left(\frac{Q_{i,j+1}^{k+1}-Q_{i,j-1}^{k+1}}{4\Delta Y}+\frac{Q_{i,j+1}^{k}-Q_{i,j-1}^{k}}{4\Delta Y}\right)+\Phi^k\gamma P_{i,j}^k\left(\frac{P_{i,j}^k+P_{i,j}^{k+1}}{2}\right), \tag{9c}$$

$$\begin{aligned}\frac{Z_{i,j}^{k+1}-Z_{i,j}^{k}}{\Delta t} =& \frac{Y_j}{u_k}\frac{u_k-u_{k-1}}{\Delta t}\left(\frac{Z_{i,j+1}^{k+1}-Z_{i,j-1}^{k+1}}{4\Delta Y}+\frac{Z_{i,j+1}^{k}-Z_{i,j-1}^{k}}{4\Delta Y}\right)+\\
&\frac{\alpha_{zz}}{2}\left[\frac{Z_{i-1,j}^{k+1}-2Z_{i,j}^{k+1}+Z_{i+1,j}^{k+1}}{\Delta x^2}+\frac{Z_{i-1,j}^{k}-2Z_{i,j}^{k}+Z_{i+1,j}^{k}}{\Delta x^2}\right]+\\
&\frac{\alpha_{zz}}{2u_k^2}\left[\frac{Z_{i,j-1}^{k+1}-2Z_{i,j}^{k+1}+Z_{i,j+1}^{k+1}}{\Delta Y^2}+\frac{Z_{i,j-1}^{k}-2Z_{i,j}^{k}+Z_{i,j+1}^{k}}{\Delta Y^2}\right]+\\
&\frac{\alpha_{zp}}{2}\left[\frac{P_{i-1,j}^{k}Z_{i-1,j}^{k+1}-2P_{i,j}^{k}Z_{i,j}^{k+1}+P_{i+1,j}^{k}Z_{i+1,j}^{k+1}}{\Delta x^2}\right]+\\
&\frac{\alpha_{zp}}{2}\left[\frac{P_{i-1,j}^{k}Z_{i-1,j}^{k}-2P_{i,j}^{k}Z_{i,j}^{k}+P_{i+1,j}^{k}Z_{i+1,j}^{k}}{\Delta x^2}\right]+\\
&\frac{\alpha_{zp}}{2u_k^2}\left[\frac{P_{i,j-1}^{k}Z_{i,j-1}^{k+1}-2P_{i,j}^{k}Z_{i,j}^{k+1}+P_{i,j+1}^{k}Z_{i,j+1}^{k+1}}{\Delta Y^2}\right]+\\
&\frac{\alpha_{zp}}{2u_k^2}\left[\frac{P_{i,j-1}^{k}Z_{i,j-1}^{k}-2P_{i,j}^{k}Z_{i,j}^{k}+P_{i,j+1}^{k}Z_{i,j+1}^{k}}{\Delta Y^2}\right]+\\
&\frac{\alpha_{zq}}{2}\left[\frac{Q_{i-1,j}^{k}Z_{i-1,j}^{k+1}-2Q_{i,j}^{k}Z_{i,j}^{k+1}+Q_{i+1,j}^{k}Z_{i+1,j}^{k+1}}{\Delta x^2}\right]+\\
&\frac{\alpha_{zq}}{2}\left[\frac{Q_{i-1,j}^{k}Z_{i-1,j}^{k}-2Q_{i,j}^{k}Z_{i,j}^{k}+Q_{i+1,j}^{k}Z_{i+1,j}^{k}}{\Delta x^2}\right]+\\
&\frac{\alpha_{zq}}{2u_k^2}\left[\frac{Q_{i,j-1}^{k}Z_{i,j-1}^{k+1}-2Q_{i,j}^{k}Z_{i,j}^{k+1}+Q_{i,j+1}^{k}Z_{i,j+1}^{k+1}}{\Delta Y^2}\right]+\\
&\frac{\alpha_{zq}}{2u_k^2}\left[\frac{Q_{i,j-1}^{k}Z_{i,j-1}^{k}-2Q_{i,j}^{k}Z_{i,j}^{k}+Q_{i,j+1}^{k}Z_{i,j+1}^{k}}{\Delta Y^2}\right],\end{aligned} \tag{9d}$$

$$F_{i,j}^k = \exp\left[-a\zeta^* u_k(1-Y_j)\right]\left[1+\exp\left(-\xi^* Z_{i,j}^k\right)\cos\left(2\pi x - 2\pi\tan\phi_r^k u_k Y_j\right)\right]. \tag{9e}$$

$$\begin{aligned}\int_0^1\int_0^1 M\ dx\ dY \approx M_k^* =& \frac{\Delta x^2}{4}\left[M_{0,0}^k+M_{0,J}^k+M_{J,0}^k+M_{J,J}^k+2M_{0,1}^k+...+2M_{0,J-1}^k+\right.\\
&2M_{1,0}^k+...+2M_{J-1,0}^k+2M_{1,J}^k+...+2M_{J-1,J}^k+2M_{J,1}^k+...\\
&\left.+2M_{J,J-1}^k+4M_{2,2}^k+...+4M_{J-2,J-2}^k\right],\end{aligned} \tag{10a}$$

$$\begin{aligned}\int_0^1\int_0^1 P\ dx\ dY \approx P_k^* =& \frac{\Delta x^2}{4}\left[P_{0,0}^k+P_{0,J}^k+P_{J,0}^k+P_{J,J}^k+2P_{0,1}^k+...+2P_{0,J-1}^k+\right.\\
&2P_{1,0}^k+...+2P_{J-1,0}^k+2P_{1,J}^k+...+2P_{J-1,J}^k+2P_{J,1}^k+...\\
&\left.+2P_{J,J-1}^k+4P_{2,2}^k+...+4P_{J-2,J-2}^k\right],\end{aligned} \tag{10b}$$

$$\begin{aligned}\int_0^1\int_0^1 Q\ dx\ dY \approx Q_k^* =& \frac{\Delta x^2}{4}\left[Q_{0,0}^k+Q_{0,J}^k+Q_{J,0}^k+Q_{J,J}^k+2Q_{0,1}^k+...+2Q_{0,J-1}^k+\right.\\
&2Q_{1,0}^k+...+2Q_{J-1,0}^k+2Q_{1,J}^k+...+2Q_{J-1,J}^k+2Q_{J,1}^k+...\\
&\left.+2Q_{J,J-1}^k+4Q_{2,2}^k+...+4Q_{J-2,J-2}^k\right],\end{aligned} \tag{10c}$$

$$u_k = \left[\frac{b_0}{\rho_b} + \frac{1}{\rho_m} + \frac{z_0}{\rho_z}\right]^{-1} \left[\frac{b_0}{\rho_b} + \frac{M_k^*}{\rho_m} + \frac{P_k^*}{\rho_p} + \frac{Q_k^*}{\rho_p} + \frac{z_0}{\rho_z}\right], \tag{10d}$$

$$\phi_r^k = \tan^{-1}\left(\frac{\tan\phi_r^0}{u_k}\right), \tag{10e}$$

$$\Lambda_k = \Lambda_0 \frac{\cos\phi_r^k}{\cos\phi_r^0}. \tag{10f}$$

Refractive Index Modulation

$$\frac{n^2-1}{n^2+2} = \phi_m \frac{n_m^2-1}{n_m^2+2} + \phi_p \frac{n_p^2-1}{n_p^2+2} + \phi_q \frac{n_q^2-1}{n_q^2+2} + \phi_z \frac{n_z^2-1}{n_z^2+2} + \phi_b \frac{n_b^2-1}{n_b^2+2}. \tag{11}$$

Solving the Lorentz-Lorenz equation will give the RI of the nanocomposite as a function of x and t. The nanocomposite RI, $n(x,t)$, can be represented by a Fourier expansion series

$$n(x,y,t) \approx \sum_{i=0} A_i(y,t)\cos\left(\frac{2\pi}{\Lambda}ix\right) + B_i(y,t)\sin\left(\frac{2\pi}{\Lambda}ix\right), \tag{12}$$

$$A_0(y,t) = \frac{1}{\Lambda}\int_0^\Lambda n(x,y,t)\ dx, \tag{13a}$$

$$A_1(y,t) = \frac{2}{\Lambda}\int_0^\Lambda n(x,y,t)\cos\left(\frac{2\pi}{\Lambda}x\right)\ dx, \tag{13b}$$

$$B_1(y,t) = \frac{2}{\Lambda}\int_0^\Lambda n(x,y,t)\sin\left(\frac{2\pi}{\Lambda}x\right)\ dx. \tag{13c}$$

RI modulation can be modelled as

$$\Delta n(y,t) = 2\sqrt{A_1^2 + B_1^2}. \tag{14}$$

Shrinkage Modelling

We can calculate the volume at time t if we have expressions for the total volume of monomer, short polymer, cross-linked polymer and nanoparticles inside the grating

$$\begin{aligned} v(t) =& \frac{1}{\rho_m}\left[\frac{1}{\hat{x}T(t)}\int_0^{T(t)}\int_0^{\hat{x}} m\ dx\ dy\right] + \frac{1}{\rho_p}\left[\frac{1}{\hat{x}T(t)}\int_0^{T(t)}\int_0^{\hat{x}} p\ dx\ dy\right] + \\ & \frac{1}{\rho_p}\left[\frac{1}{\hat{x}T(t)}\int_0^{T(t)}\int_0^{\hat{x}} q\ dx\ dy\right] + \frac{1}{\rho_z}\left[\frac{1}{\hat{x}T(t)}\int_0^{T(t)}\int_0^{\hat{x}} z\ dx\ dy\right] + \\ & \frac{1}{\rho_b}\left[\frac{1}{\hat{x}T(t)}\int_0^{T(t)}\int_0^{\hat{x}} b\ dx\ dy\right]. \end{aligned} \tag{15}$$

An important assumptions of the fringe-plane rotation model is that all loss of volume due to polymerization takes place in the thickness of the recording medium

$$u(t) = \frac{T(t)}{T_0} = \frac{v(t)}{v(0)}. \tag{16}$$

$$u(t) = \left[\frac{1}{\rho_b}\frac{b_0}{m_0} + \frac{1}{\rho_m} + \frac{1}{\rho_z}\frac{z_0}{m_0}\right]^{-1} \left[\int_0^1\int_0^1 \frac{M}{\rho_m} + \frac{P}{\rho_p} + \frac{Q}{\rho_p} + \frac{z_0/m_0 Z}{\rho_z} + \frac{b_0/m_0}{\rho_b}\ dx\ dY\right], \tag{17}$$

$$\text{Actual Shrinkage} = \frac{u(0)-u(t)}{u(0)} = 1 - u(t), \tag{18}$$

$$\phi_r(t) = \tan^{-1}\left[\frac{\tan\phi_r(0)}{u(t)}\right], \tag{19a}$$

$$\Lambda(t) = \hat{x}\cos\phi_r(t), \tag{19b}$$

$$\overline{n}(t) = \frac{1}{\hat{x}T}\int_0^T\int_0^{\hat{x}} n(x,y,t)\ dx\ dy = \int_0^1\int_0^1 n(x,Y,t)\ dx\ dY, \tag{20}$$

$$\theta_B(t) = \sin^{-1}\left(\frac{\lambda_r}{2\overline{n}(t)\Lambda(t)}\right) - \phi_r(t), \tag{21}$$

$$\text{Apparent Shrinkage} = 1 - \frac{\tan\phi_r(0)}{\tan\left[\phi_r(0) + \Delta\theta_B\right]}. \tag{22}$$

Python Script

```python
from warnings import filterwarnings
filterwarnings("ignore")
import numpy as np
import pandas as pd
from math import pi, factorial
from numpy.linalg import inv
from time import time as gettime

class HolographicGrating:

    def __init__(
            self,
            start_exp=0,# Start of exposure
            end_exp=1e2,# End of exposure
            total_time=1e2,# Total simulation time
            lpmm=1e3,# Spatial frequency
            I0=5,# Intensity of recording beam
            slant_angle=1e-4,# Grating slant_angle
            xi=0.3,# Scattering coefficient
            n_m=1.55,# Monomer refractive index
            rhom=1.15,# Monomer density
            Dm=1.6e-7,# Monomer diffusion coefficient
            Dp=6.35e-10,# Polymer diffusion coefficient
            rhop=1.3,# Polymer density
            n_p=1.56,# Oligomer refractive index
            n_q=1.64,# Polymer refractive index
            Gamma=1,# Rate of immobilization
            wt_pc=5e-2,# Doping %
            Dz=1e-10,# Nanoparticle self-diffusion coefficient
            epsilon_pz=13,# Cross-diffusion ratio
            epsilon_qz=13,# Cross-diffusion ratio
            rhoz=1.74,# Nanoparticle mass density
            n_z=1.366,# Nanoparticle refractive index
            b0=5.05,# Ratio of binder to monomer mass
            n_b=1.5,# Binder refractive index
            rhob=1.19,# Binder mass density
            T0=50e-4,# Depth of photosensitive layer [cm]
            zeta=139,# absorption coefficient [cm**-1]
            lambda_probe=633e-7,# Wavelength of reconstruction beam
            Delta_t=1/100,# Numerical scheme time step
            Delta_x=1/20,# Numerical scheme spatial step
            output_time_step=1# Seconds
        ):

        self.total_time = total_time
        self.end_exp = end_exp
        self.lpmm = lpmm
        self.T0 = T0
        self.I0 = I0
        if slant_angle == 0:
            self.slant_angle = 1e-5
        else:
```

```
            self.slant_angle = slant_angle
        self.T0 = T0
        self.zeta = zeta
        self.xi = xi
        self.Dm = Dm
        self.n_m = n_m
        self.rhom = rhom
        self.Dp = Dp
        self.rhop = rhop
        self.n_p = n_p
        self.n_q = n_q
        self.Gamma = Gamma
        self.Dz = Dz
        self.epsilon_pz = epsilon_pz
        self.epsilon_qz = epsilon_qz
        self.wt_pc = wt_pc
        self.rhoz = rhoz
        self.n_z = n_z
        self.b0 = b0
        self.n_b = n_b
        self.rhob = rhob
        self.lambda_probe = lambda_probe
        self.Delta_x = Delta_x
        self.Nx = int(1/Delta_x) + 1
        self.Delta_Y = Delta_x
        self.Delta_t = Delta_t
        self.output_time_step = output_time_step


    def slanted_grating_simulation_v22(self):

        start_computation = gettime()
        # 1.2 --- Define parameters
        Nx=int(1/self.Delta_x) + 1# Number of spatial points
        Ny=int(1/self.Delta_Y) + 1# Number of spatial points
        if Nx%2==0:
          return "Number of x mesh points must be an odd number."
        if Ny%2==0:
           return "Number of y mesh points must be an odd number."

        x=np.linspace(0,1,Nx)# Non-dimensional grating distance
        n_iterations = int(self.total_time/self.Delta_t)+1# Total number of iterations
        r=self.Delta_t/self.Delta_x/self.Delta_x# Ratio of finite time step to squared finite spatial step
        m0=1# Initial mass of monomer
        t0=1 #  Reference time [s]
        Lambda0=1/10/self.lpmm # Grating period [cm]
        Lambda1=Lambda0
        j_end_exp = self.end_exp/self.Delta_t # Iteration of exposure end
        z0 = self.wt_pc/(1 - self.wt_pc)*(m0 + self.b0)# Initial nanoparticle to monomer

        # 1.3 --- Matrix initial conditions
        u1=1
        du_dt=0
        m1 = np.ones(Nx*Nx)# m at j=0
        p1 = np.zeros(Nx*Nx)# p at j=0
        q1 = np.zeros(Nx*Nx)# q at j=0
        z1 = np.ones(Nx*Nx)# z at j=0
        b1 = self.b0*np.ones(Nx*Nx)# b at j=0

        Volume0=m0/self.rhom + self.b0/self.rhob + z0/self.rhoz

        phi_m0=m0/self.rhom/Volume0
        phi_z0=z0/self.rhoz/Volume0
        phi_b0=self.b0/self.rhob/Volume0

        Lorentz_Lorenz_RHS = phi_m0*(self.n_m*self.n_m - 1)/(self.n_m*self.n_m + 2) + phi_b0*(self.n_b*self.n_b - 1)/(self.n_b*self.n_b + 2) + phi_z0*(self.n_z*self.n_z - 1)/(self.n_z*self.n_z + 2)

        Initial_RI = np.sqrt((2*Lorentz_Lorenz_RHS + 1)/(1 - Lorentz_Lorenz_RHS))

        inside_ri = Initial_RI

```

```
        n1=Initial_RI*np.ones(Nx*Nx)

        phi_0=np.arcsin(np.sin(self.slant_angle/180*pi)/Initial_RI)
        phi_1=phi_0
        theta_B0=np.arcsin(self.lambda_probe/2/Initial_RI/Lambda0) - phi_0
        y_hat0=Lambda0/np.sin(phi_0)
        y_hat1=y_hat0
        x_hat=Lambda0/np.cos(phi_0)

        # 1.5 --- Nondimensionalized parameters
        alpha_m_x=self.Dm*t0/x_hat/x_hat
        alpha_m_y=self.Dm*t0/self.T0/self.T0
        alpha_p_x=self.Dp*t0/x_hat/x_hat
        alpha_p_y=self.Dp*t0/self.T0/self.T0
        alpha_z_x=self.Dz*t0/x_hat/x_hat
        alpha_z_y=self.Dz*t0/self.T0/self.T0
        F0=0.1*self.I0**0.3
        beta=F0*t0
        gamma = self.Gamma*m0*t0
        zeta_star = self.zeta*self.T0
        xi_star=self.xi*z0
        interior_points = list(range(1,Nx-1))
        times_4 = [interior_points[i] for i in range(len(interior_points)) if interior_points[i]%2 != 0]
        times_2 = [interior_points[i] for i in range(len(interior_points)) if interior_points[i]%2 == 0]
        Y=np.arange(0,1+self.Delta_x, self.Delta_x)
        time=np.arange(1,self.total_time+self.output_time_step, self.output_time_step)
        Y1=[]
        x1=[]
        for i in range(Nx):
            for j in list(Y):
                Y1.append(j)
            for j in x:
                x1.append(j)
        Y1=np.sort(Y1)
        indexDF=pd.DataFrame({'x':x1, 'y':Y1})

        spatial_profile_DF=pd.DataFrame({"x": x1, 'Y': Y1, "monomer": m1, "short_polymer": p1, "immobile_polymer": q1, 'nanoparticles': z1, 'binder':b1, 'refractive_index': n1, "time": np.zeros(len(n1)), 'N0':np.zeros(len(n1))+Initial_RI, 'Delta_n':np.zeros(len(n1)), 'd2':np.zeros(len(n1))})

        optical_properties_DF=pd.DataFrame({'Y': Y, "time": np.zeros(len(Y)), 'N0':np.zeros(len(Y))+Initial_RI,'Delta_n':np.zeros(len(Y)),'nu':np.zeros(len(Y)),'d2':np.zeros(len(Y))})

        shrinkage_DF=pd.DataFrame({'time':[0], 'actual_shrinkage':[0],'phi_t':[phi_0],'theta_B':[theta_B0],'apparent_shrinkage':[0]})

        def trapezoidal_rule_integration(array):

            array=array.reshape(Nx,Nx).T

            return Delta_x*Delta_x/4*(array[0,0] + array[Nx-1,0] + array[0,Nx-1] + array[Nx-1,Nx-1] + np.sum(2*array[0,1:(Nx-1)]) + np.sum(2*array[1:(Nx-1),0]) + np.sum(2*array[Nx-1,1:(Nx-1)]) + np.sum(2*array[1:(Nx-1),Nx-1]) + np.sum(4*array[1:(Nx-1),1:(Nx-1)]))

        def simpsons_rule_1D(arr1D):

            intpts=list(range(1,len(arr1D)-1))
            times_4=[i for i in intpts if i%2!=0]
            times_2=[i for i in intpts if i%2==0]

            return Delta_x/3*(arr1D[0] + 4*sum(arr1D[times_4]) + 2*sum(arr1D[times_2]) + arr1D[len(arr1D)-1])



        # 1.6 --- Calculate each time step via implicit finite difference method
        for j in range(1,n_iterations):

            if j <= j_end_exp:
                Phi=1
            else:
```

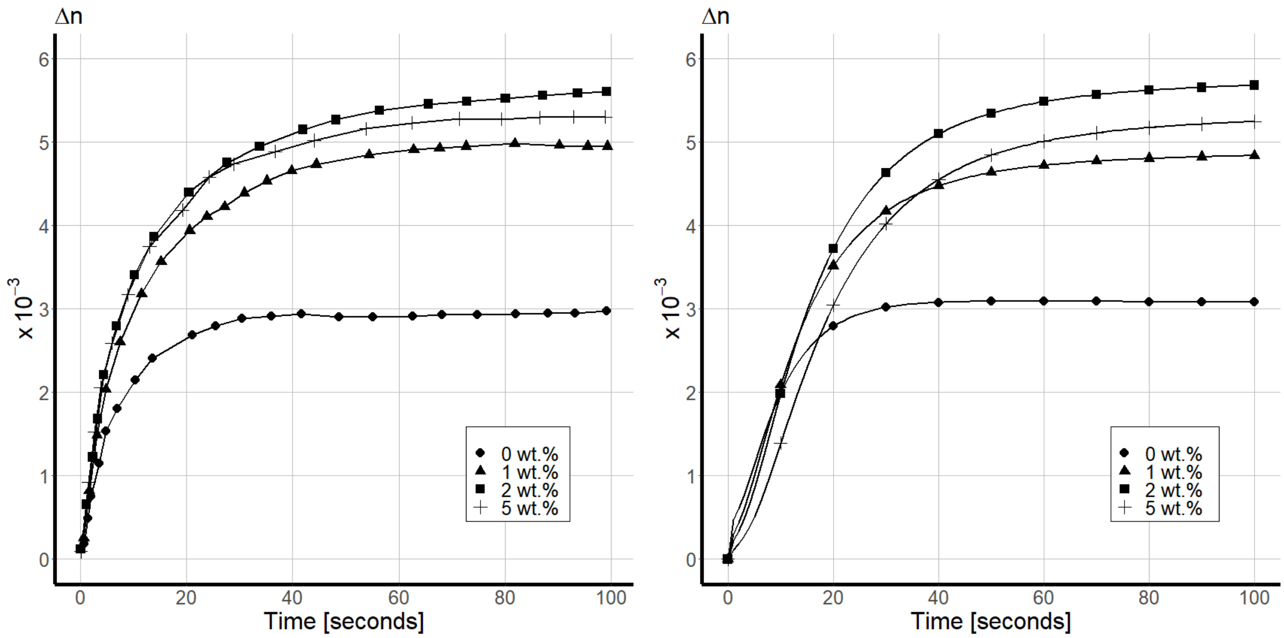
```
            Phi = 0# Phi=1 if illumination is on, 0 otherwise

        f = np.zeros(Nx*Nx).reshape(Nx,Nx)
        for i in range(Nx):
            matrix_z1=z1.reshape(Nx,Nx)
            z1_i=matrix_z1[:,i]
            f[:,i] = np.exp(-0.3*zeta_star*u1*(1-Y[i]))*(1 + np.exp(-xi_star*z1_i)*np.cos(2*pi*x - 2*pi*self.T0/y_hat1*u1*Y[i]))

        f = f.reshape(Nx*Nx,)

        MM2 = (2 + Phi*self.Delta_t*beta*f)*np.identity(Nx*Nx)
        MM1 = (2 - Phi*self.Delta_t*beta*f)*np.identity(Nx*Nx)
        PP2 = (2 + self.Delta_t*gamma*p1)*np.identity(Nx*Nx)
        PP1 = (2 - self.Delta_t*gamma*p1)*np.identity(Nx*Nx)
        PM2 = (+Phi*self.Delta_t*beta*f)*np.identity(Nx*Nx)
        PM1 = (+Phi*self.Delta_t*beta*f)*np.identity(Nx*Nx)
        QQ2 = 2*np.identity(Nx*Nx)
        QQ1 = 2*np.identity(Nx*Nx)
        QP2 = (+Phi*gamma*self.Delta_t*p1)*np.identity(Nx*Nx)
        QP1 = (+Phi*gamma*self.Delta_t*p1)*np.identity(Nx*Nx)
        ZZ2 = 2*np.identity(Nx*Nx)
        ZZ1 = 2*np.identity(Nx*Nx)
        BB2 = 2*np.identity(Nx*Nx)
        BB1 = 2*np.identity(Nx*Nx)

        for i in range(Nx*Nx):

            if i in list(indexDF.loc[indexDF.x==0,].index):
                i_minus_1 = i+Nx-2
            else:
                i_minus_1 = i-1

            if i in list(indexDF.loc[indexDF.x==1,].index):
                i_plus_1 = i-Nx+2
            else:
                i_plus_1 = i+1

            if i in list(indexDF.loc[indexDF.y==0,].index):
                j_minus_1 = i+Nx
            else:
                j_minus_1 = i-Nx

            if i in list(indexDF.loc[indexDF.y==1,].index):
                j_plus_1 = i-Nx
            else:
                j_plus_1 = i+Nx

            MM2[i, i_minus_1] = MM2[i, i_minus_1] - r*alpha_m_x
            MM2[i, j_minus_1] = MM2[i, j_minus_1] - r*alpha_m_y/u1/u1
            MM2[i, i] = MM2[i, i] + 2*r*alpha_m_x
            MM2[i, i] = MM2[i, i] + 2*r*alpha_m_y/u1/u1
            MM2[i, i_plus_1] = MM2[i, i_plus_1] - r*alpha_m_x
            MM2[i, j_plus_1] = MM2[i, j_plus_1] - r*alpha_m_y/u1/u1

            MM1[i, i_minus_1] = MM1[i, i_minus_1] + r*alpha_m_x
            MM1[i, j_minus_1] = MM1[i, j_minus_1] + r*alpha_m_y/u1/u1
            MM1[i, i] = MM1[i, i] - 2*r*alpha_m_x
            MM1[i, i] = MM1[i, i] - 2*r*alpha_m_y/u1/u1
            MM1[i, i_plus_1] = MM1[i, i_plus_1] + r*alpha_m_x
            MM1[i, j_plus_1] = MM1[i, j_plus_1] + r*alpha_m_y/u1/u1

            PP2[i, i_minus_1] = PP2[i, i_minus_1] - r*alpha_p_x*(1 + self.epsilon_pz*z0*z1[i_minus_1])
            PP2[i, j_minus_1] = PP2[i, j_minus_1] - r*alpha_p_y/u1/u1*(1 + self.epsilon_pz*z0*z1[j_minus_1])
            PP2[i, i] = PP2[i, i] + 2*r*alpha_p_x*(1 + self.epsilon_pz*z0*z1[i])
            PP2[i, i] = PP2[i, i] + 2*r*alpha_p_y/u1/u1*(1 + self.epsilon_pz*z0*z1[i])
            PP2[i, i_plus_1] = PP2[i, i_plus_1] - r*alpha_p_x*(1 + self.epsilon_pz*z0*z1[i_plus_1])
            PP2[i, j_plus_1] = PP2[i, j_plus_1] - r*alpha_p_y/u1/u1*(1 + self.epsilon_pz*z0*z1[j_plus_1])

            PP1[i, i_minus_1] = PP1[i, i_minus_1] + r*alpha_p_x*(1 + self.epsilon_pz*z0*z1[
```

```
i_minus_1])
            PP1[i, j_minus_1] = PP1[i, j_minus_1] + r*alpha_p_y/u1/u1*(1 + self.epsilon_pz*z0*z1[j_minus_1])
            PP1[i, i] = PP1[i, i] - 2*r*alpha_p_x*(1 + self.epsilon_pz*z0*z1[i])
            PP1[i, i] = PP1[i, i] - 2*r*alpha_p_y/u1/u1*(1 + self.epsilon_pz*z0*z1[i])
            PP1[i, i_plus_1] = PP1[i, i_plus_1] + r*alpha_p_x*(1 + self.epsilon_pz*z0*z1[i_plus_1])
            PP1[i, j_plus_1] = PP1[i, j_plus_1] + r*alpha_p_y/u1/u1*(1 + self.epsilon_pz*z0*z1[j_plus_1])

            ZZ2[i, i_minus_1] = ZZ2[i, i_minus_1] - r*alpha_z_x*(1 + self.epsilon_qz*q1[i_minus_1] + self.epsilon_pz*p1[i_minus_1])
            ZZ2[i, j_minus_1] = ZZ2[i, j_minus_1] - r*alpha_z_y/u1/u1*(1 + self.epsilon_qz*q1[j_minus_1] + self.epsilon_pz*p1[j_minus_1])
            ZZ2[i, i] = ZZ2[i, i] + 2*r*alpha_z_x*(1 + self.epsilon_qz*q1[i] + self.epsilon_pz*p1[i])
            ZZ2[i, i] = ZZ2[i, i] + 2*r*alpha_z_y/u1/u1*(1 + self.epsilon_qz*q1[i] + self.epsilon_pz*p1[i])
            ZZ2[i, i_plus_1] = ZZ2[i, i_plus_1] - r*alpha_z_x*(1 + self.epsilon_qz*q1[i_plus_1] + self.epsilon_pz*p1[i_plus_1])
            ZZ2[i, j_plus_1] = ZZ2[i, j_plus_1] - r*alpha_z_y/u1/u1*(1 + self.epsilon_qz*q1[j_plus_1] + self.epsilon_pz*p1[j_plus_1])

            ZZ1[i, i_minus_1] = ZZ1[i, i_minus_1] + r*alpha_z_x*(1 + self.epsilon_qz*q1[i_minus_1] + self.epsilon_pz*p1[i_minus_1])
            ZZ1[i, j_minus_1] = ZZ1[i, j_minus_1] + r*alpha_z_y/u1/u1*(1 + self.epsilon_qz*q1[j_minus_1] + self.epsilon_pz*p1[j_minus_1])
            ZZ1[i, i] = ZZ1[i, i] - 2*r*alpha_z_x*(1 + self.epsilon_qz*q1[i] + self.epsilon_pz*p1[i])
            ZZ1[i, i] = ZZ1[i, i] - 2*r*alpha_z_y/u1/u1*(1 + self.epsilon_qz*q1[i] + self.epsilon_pz*p1[i])
            ZZ1[i, i_plus_1] = ZZ1[i, i_plus_1] + r*alpha_z_x*(1 + self.epsilon_qz*q1[i_plus_1] + self.epsilon_pz*p1[i_plus_1])
            ZZ1[i, j_plus_1] = ZZ1[i, j_plus_1] + r*alpha_z_y/u1/u1*(1 + self.epsilon_qz*q1[j_plus_1] + self.epsilon_pz*p1[j_plus_1])

            MM2[i, j_minus_1] = MM2[i, j_minus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            MM1[i, j_minus_1] = MM1[i, j_minus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            MM2[i, j_plus_1] = MM2[i, j_plus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            MM1[i, j_plus_1] = MM1[i, j_plus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt

            PP2[i, j_minus_1] = PP2[i, j_minus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            PP1[i, j_minus_1] = PP1[i, j_minus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            PP2[i, j_plus_1] = PP2[i, j_plus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            PP1[i, j_plus_1] = PP1[i, j_plus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt

            QQ2[i, j_minus_1] = QQ2[i, j_minus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            QQ1[i, j_minus_1] = QQ1[i, j_minus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            QQ2[i, j_plus_1] = QQ2[i, j_plus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            QQ1[i, j_plus_1] = QQ1[i, j_plus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt

            ZZ2[i, j_minus_1] = ZZ2[i, j_minus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            ZZ1[i, j_minus_1] = ZZ1[i, j_minus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            ZZ2[i, j_plus_1] = ZZ2[i, j_plus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            ZZ1[i, j_plus_1] = ZZ1[i, j_plus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt

            BB2[i, j_minus_1] = BB2[i, j_minus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            BB1[i, j_minus_1] = BB1[i, j_minus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            BB2[i, j_plus_1] = BB2[i, j_plus_1] - Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt
            BB1[i, j_plus_1] = BB1[i, j_plus_1] + Y1[i]/u1*self.Delta_t/2/self.Delta_x*du_dt

        m2 = np.matmul(inv(MM2), np.matmul(MM1,m1))

        p2 = np.matmul(inv(PP2), np.matmul(PP1,p1) + np.matmul(PM2,m2) + np.matmul(PM1,m1))

        q2 = np.matmul(inv(QQ2), np.matmul(QQ1, q1) + np.matmul(QP2,p2) + np.matmul(QP1,p1))

        if z0==0:
            z2 = z1
        else:
            z2 = np.matmul(inv(ZZ2), np.matmul(ZZ1,z1))

        b2 = np.matmul(inv(BB2), np.matmul(BB1,b1))

```

```
            Vb = b2/self.rhob # cm**3
            Vm = m2*m0/self.rhom # cm**3
            Vp = p2*m0/self.rhop # cm**3
            Vq = q2*m0/self.rhop # cm**3
            Vz = z2*z0/self.rhoz # cm**3
            Vtotal=Vb+Vm+Vp+Vq+Vz # cm**3
            phi_m = Vm/Vtotal
            phi_b = Vb/Vtotal
            phi_p = Vp/Vtotal
            phi_q = Vq/Vtotal
            phi_z = Vz/Vtotal

            Lorentz_Lorenz_RHS = phi_m*(self.n_m*self.n_m - 1)/(self.n_m*self.n_m + 2) + phi_b*(self.n_b*self.n_b - 1)/(self.n_b*self.n_b + 2) + phi_p*(self.n_p*self.n_p - 1)/(self.n_p*self.n_p + 2) + phi_q*(self.n_q*self.n_q - 1)/(self.n_q*self.n_q + 2) + phi_z*(self.n_z*self.n_z - 1)/(self.n_z*self.n_z + 2)

            n2=np.sqrt((2*Lorentz_Lorenz_RHS + 1)/(1 - Lorentz_Lorenz_RHS))
            n2=n2.reshape(Nx,Nx).T

            inside_ri = simpsons_rule_1D(n2[int((Nx-1)/2)])

            N0=[(self.Delta_x/3*(n2[0,i] + np.sum(2*n2[times_2,i]) + np.sum(4*n2[times_4,i]) + n2[(Nx-1),i])) for i in range(Nx)]

            n2_cos=np.zeros(n2.shape)
            for i in range(Nx):
                n2_cos[:,i] = n2[:,i]*np.cos(2*pi*x)

            n2_sin=np.zeros(n2.shape)
            for i in range(Nx):
                n2_sin[:,i] = n2[:,i]*np.sin(2*pi*x)

            N1_a=np.array([2*self.Delta_x/3*(n2_cos[0,i] + np.sum(2*n2_cos[times_2,i]) + np.sum(4*n2_cos[times_4,i]) + n2_cos[(Nx-1),i]) for i in range(Nx)])

            N1_b=np.array([2*self.Delta_x/3*(n2_sin[0,i] + np.sum(2*n2_sin[times_2,i]) + np.sum(4*n2_sin[times_4,i]) + n2_sin[(Nx-1),i]) for i in range(Nx)])

            n_tilde=np.ones((Nx,Nx))
            for i in range(Nx):
                n_tilde[:,i]=N0[i]*n_tilde[:,i] + N1_a[i]*np.cos(2*pi*x) + N1_b[i]*np.sin(2*pi*x)

            sq_diff=(n2 - n_tilde)**2

            d2=[(self.Delta_x/3*(sq_diff[0,i] + np.sum(2*sq_diff[times_2,i]) + np.sum(4*sq_diff[times_4,i]) + sq_diff[(Nx-1),i])) for i in range(Nx)]

            n2=n2.T.reshape(Nx*Nx,)

            Volume1=(trapezoidal_rule_integration(m2)/self.rhom + trapezoidal_rule_integration(p2)/self.rhop + trapezoidal_rule_integration(q2)/self.rhop + trapezoidal_rule_integration(z2)*z0/self.rhoz + trapezoidal_rule_integration(b2)/self.rhob)

            u1 = Volume1/Volume0

            phi_1=np.arctan(np.tan(phi_0)/u1)

            Lambda1=np.cos(phi_1)/np.cos(phi_0)*Lambda0

            y_hat1=Lambda1/np.sin(phi_1)

            theta_B=np.arcsin(self.lambda_probe/2/inside_ri/Lambda1)-phi_1

            Delta_theta_B=theta_B0-theta_B,

            Delta_n=np.sqrt(N1_a*N1_a+N1_b*N1_b)

            nu=pi*Delta_n*self.T0*u1/self.lambda_probe/np.cos(theta_B)

            m1 = m2
            p1 = p2
            q1 = q2
            z1 = z2
```

```python
            b1 = b2
            n1 = n2

            if self.Delta_t*j in time:

                spatial_profile_DF=pd.concat([spatial_profile_DF, pd.DataFrame({"x": x1, 'Y': Y1,"monomer": m1,"short_polymer": p1,"immobile_polymer": q1,'nanoparticles': z1,'binder':b1,'refractive_index': n1,"time":j*self.Delta_t*np.ones(len(n1))})]).reset_index(drop=True)

                optical_properties_DF=pd.concat([optical_properties_DF,pd.DataFrame({'time':j*self.Delta_t,'Y':Y,'N0':N0,'Delta_n':Delta_n,'d2':d2, 'nu':nu})]).reset_index(drop=True)

                shrinkage_DF=pd.concat([shrinkage_DF,pd.DataFrame({'time':[self.Delta_t*j], 'actual_shrinkage':[1-u1],'phi_t':[phi_1], 'theta_B':[theta_B], 'apparent_shrinkage':[1 - np.tan(phi_0)/np.tan(phi_0 + Delta_theta_B)][0]})]).reset_index(drop=True)


        #Moharam_Young=lambda_probe*lambda_probe/Mean_RI/Delta_n/Lambda1/Lambda1/np.cos(phi_1)

        Klein_Cook=2*pi*self.lambda_probe*self.T0*u1/inside_ri/Lambda1/Lambda1/np.cos(phi_1)

        if Klein_Cook < 10:
            self.Geometry='Planar'
            J0=optical_properties_DF.nu/2
            J1=J0
            for l in range(1,101):
                J1 = J1 + ((-1)**l)/factorial(l)/factorial(l+1)*(J0**(2*l + 1))
            eta=J1*J1
        else:
            self.Geometry='Volume'
            eta=np.sin(np.sqrt(optical_properties_DF.nu*optical_properties_DF.nu))**2

        optical_properties_DF['eta']=eta

        end_computation = gettime()

        computation_duration = end_computation-start_computation

        return spatial_profile_DF, optical_properties_DF, shrinkage_DF, computation_duration

if __name__ == "__main__":

    a = HolographicGrating(total_time=0)
    df1, df2, df3, t0 = a.slanted_grating_simulation_v22()
    assert len(df3) == 1
    assert len(df2) == a.Nx
    assert len(df1) == a.Nx*a.Nx
    del a, df1, df2, df3, t0
    print("All tests passed.")
```

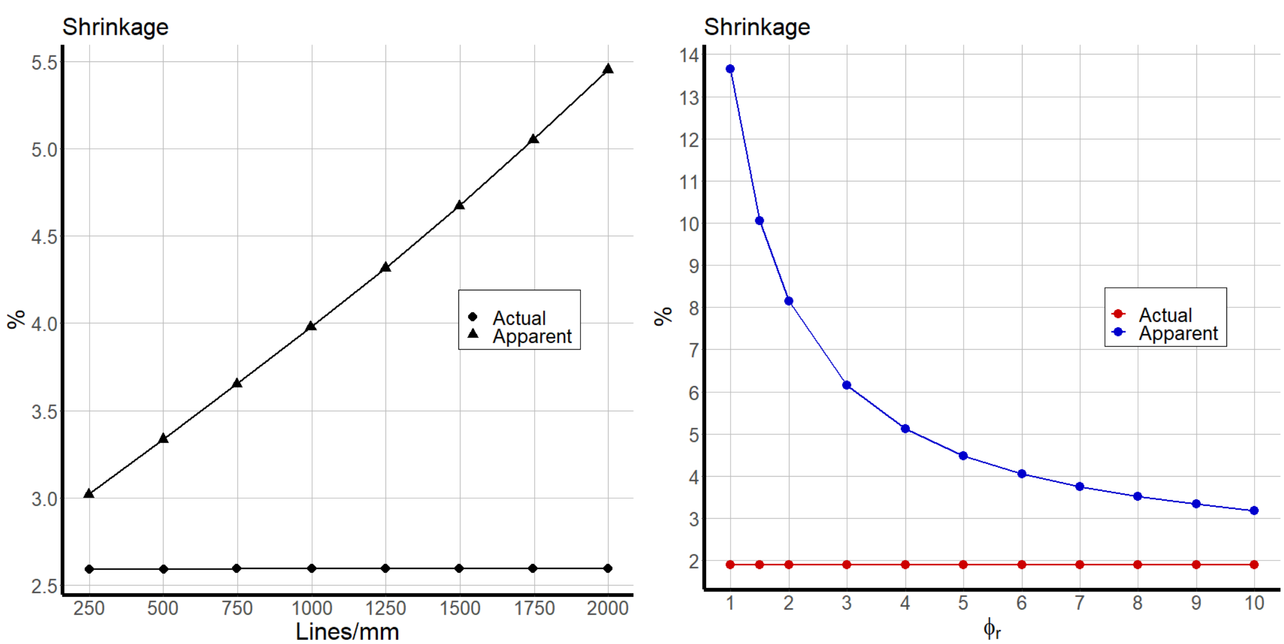
Results



(a) Observed[1]

(b) Predicted; $\epsilon_z = 13$; $\xi = 1.6$

Figure 2: Time evolution of Δn of an unslanted holographic grating recorded in AA/PVA photopolymer with increased doping of BEA nanozeolites.



(a) ... spatial frequencies ranging from 250 - 2000 lines/mm

(b) ... slant angles up to 10°

Figure 3: The predicted actual and apparent shrinkage for ...

[1]Cody, D et al. (2014), *Effect of zeolite nanoparticles on the optical properties of diacetone acrylamide-based photopolymer*, Optical Materials 37, 181-187