Molecularly Programmed Metasurfaces: Gaussian + Multiwfn + S4 Workflow

2025-11-09

1 Overview

Pairing **Gaussian** + **Multiwfn** (molecular optics) with **S4** (metasurface RCWA) is a practical route to design *molecularly programmed* metasurfaces. This note provides: (i) concrete project ideas, (ii) an end-to-end physics workflow that connects the tools, and (iii) ready-to-run code to convert TD-DFT outputs into dispersion $n(\lambda)$, $k(\lambda)$ for S4.

2 Feasible and Impactful Project Ideas

1) Molecularly tuned perfect absorber (critical coupling)

Design a guided-mode resonance or lamellar-grating metasurface whose linewidth and center frequency match a dye's excitonic line so that incident light is dissipated in the dye layer at the target wavelength.

Method: TD-DFT (Gaussian) → oscillator strengths → effective medium via Clausius–Mossotti/Lorentz–Lorenz → S4 for $R/T/A$ spectra and near fields (S4).

2) All-optical switching with photochromes

Compute two states (e.g., trans/cis) to obtain two dispersion models, and simulate a metasurface at a high phase-sensitivity point to get a large spectral/phase shift upon optical pumping.

3) Protein/DNA overlayer index engineering

A 5–20 nm biomolecular overlayer (protein monolayer, DNA brush) shifts a narrow resonance. Compute its complex refractive index from polarizability and predict limits of detection.

4) Excitonic anisotropy and dipole orientation

Aligned dyes yield anisotropic permittivity. Compute direction-dependent polarizability and simulate polarization-dependent metasurface phase.

Why this pairing works S4 captures the electromagnetic lattice response; Gaussian/Multiwfn provide molecular resonances and oscillator strengths; Clausius–Mossotti/Lorentz–Lorenz converts microscopic polarizability to macroscopic $\varepsilon(\omega)$ that S4 consumes wavelength-by-wavelength.

3 End-to-End Workflow

A. Quantum chemistry (Gaussian)

- Optimize ground-state geometry.

- TD-DFT for excited states (use range-separated hybrids when charge-transfer character is expected, e.g., CAM-B3LYP/ ωB97X-D).
- Optional: include solvent/host via PCM. Gaussian reports lines like:
  `Excited State 1: Singlet-A 2.35 eV 527.5 nm f=0.51` $<S^2>= 0.0000$
  Extract excitation energies E_j and oscillator strengths f_j.

B. Extract & sanity-check (Multiwfn)

Open the Gaussian log or fchk in Multiwfn (Spectrum/Excitations module) and export discrete lines (E_j, f_j) (Multiwfn manual).

C. Molecules $\rightarrow$ materials (physics bridge)

Use a Lorentz model for the molecular polarizability

$$\alpha_{\mathrm{mol}}(\omega) = \sum_j \frac{e^2}{m_e} \frac{f_j}{\omega_j^2 - \omega^2 - i\,\gamma_j\,\omega}\,. \tag{1}$$

Mix into a bulk film via Clausius–Mossotti / Lorentz–Lorenz

$$\frac{\varepsilon_r - 1}{\varepsilon_r + 2} = \frac{N\,\alpha_{\mathrm{mol}}}{3\,\varepsilon_0} \quad \Rightarrow \quad \varepsilon_r(\omega) = \frac{1 + 2X}{1 - X}, \quad X = \frac{N\,\alpha_{\mathrm{mol}}}{3\,\varepsilon_0}\,. \tag{2}$$

Here N is number density from wt% and film density. From ε_r obtain $n(\omega)$ and $k(\omega)$.

D. Electromagnetics (S4)

Define a periodic stack (e.g., air / patterned dielectric / dye film / substrate). For each wavelength λ: load $n(\lambda), k(\lambda)$ for the dye layer, set incidence and polarization, compute R, T, A and near fields, and sweep geometry to reach critical coupling or maximize phase slope (S4).

Practical tip For a host polymer (e.g., PMMA/PVA), convert its known n_{host} to a host term in Lorentz–Lorenz and add it before inverting to ε_r.

4 TD-DFT $\rightarrow$ $n(\lambda), k(\lambda)$: Ready-to-run Code

The script parses a Gaussian TD-DFT log, builds a Lorentz-oscillator polarizability from (E_j, f_j), mixes to a bulk film via Clausius–Mossotti, and writes a CSV of λ, n, k. One homogeneous linewidth γ (eV) is used; host index is optional.

Python script

```
#!/usr/bin/env python3
# gaussian_to_nk.py
# Convert Gaussian TD-DFT excited states (Ej, fj) to dispersive n(lambda), k(lambda)
# using Lorentz oscillators + Clausius-Mossotti / Lorentz-Lorenz mixing.

import re
import math
import csv
import argparse
from typing import List, Tuple
```

```
# Physical constants (SI)
EPS0 = 8.8541878128e-12      # F/m
QE   = 1.602176634e-19       # C
ME   = 9.1093837015e-31      # kg
HBAR = 1.054571817e-34       # J*s
C0   = 299792458.0           # m/s
NA   = 6.02214076e23         # 1/mol
EV_TO_J = QE                 # 1 eV = q_e Joules

EXCITED_STATE_RE = re.compile(
    r"Excited State\s+\d+:\s+.*?([\d\.]+)\s+eV\s+[\d\.]+\s+nm\s+f=([\d\.]+)",
    re.IGNORECASE
)

def parse_gaussian_td_log(path: str) -> List[Tuple[float, float]]:
    E_f = []
    with open(path, 'r', errors='ignore') as fh:
        for line in fh:
            m = EXCITED_STATE_RE.search(line)
            if m:
                E_eV = float(m.group(1))
                f    = float(m.group(2))
                E_f.append((E_eV, f))
    if not E_f:
        raise RuntimeError("No excited states found. Check your Gaussian log.")
    return E_f

def eV_to_omega(E_eV: float) -> float:
    return (E_eV * EV_TO_J) / HBAR

def eV_to_gamma_radps(gamma_eV: float) -> float:
    return (gamma_eV * EV_TO_J) / HBAR

def lorentz_alpha_mol(omega: float, lines: List[Tuple[float, float]], gamma_eV: float) -> co
    alpha = 0+0j
    gamma = eV_to_gamma_radps(gamma_eV)
    pref  = (QE**2) / ME
    for E_eV, f in lines:
        wj = eV_to_omega(E_eV)
        denom = (wj**2 - omega**2) - 1j*gamma*omega
        alpha += pref * f / denom
    return alpha

def lorentz_lorenz_X_from_n(n: float) -> float:
    return (n*n - 1.0) / (n*n + 2.0)

def epsilon_from_X(X: complex) -> complex:
    return (1 + 2*X) / (1 - X)

def nk_from_eps(eps_r: complex):
    nr = complex(eps_r)**0.5
```

```
    return (nr.real, nr.imag if nr.imag >= 0 else -nr.imag)

def number_density_from_wt_fraction(wt_frac, density_g_cm3, molar_mass_g_mol):
    rho = density_g_cm3 * 1000.0
    M   = molar_mass_g_mol / 1000.0
    c_mol_m3 = (wt_frac * rho) / M
    return c_mol_m3 * NA

def build_dispersion(lines, gamma_eV, N_m3, lambdas_nm, n_host=None):
    X_host = lorentz_lorenz_X_from_n(n_host) if n_host is not None else 0.0
    out = []
    for lam_nm in lambdas_nm:
        lam_m = lam_nm * 1e-9
        omega = 2*math.pi*C0/lam_m
        alpha = lorentz_alpha_mol(omega, lines, gamma_eV)
        X_dye = (N_m3 * alpha) / (3.0 * EPS0)
        X_tot = X_host + X_dye
        eps_r = epsilon_from_X(X_tot)
        n, k = nk_from_eps(eps_r)
        out.append((lam_nm, n, k))
    return out

def main():
    p = argparse.ArgumentParser(description="Convert Gaussian TD-DFT excitations to n(lambda
    p.add_argument("--log", required=True)
    p.add_argument("--gamma_eV", type=float, default=0.10)
    p.add_argument("--wt_percent", type=float, default=1.0)
    p.add_argument("--density_g_cm3", type=float, default=1.20)
    p.add_argument("--molar_mass_g_mol", type=float, required=True)
    p.add_argument("--lambda_min_nm", type=float, default=400.0)
    p.add_argument("--lambda_max_nm", type=float, default=800.0)
    p.add_argument("--lambda_step_nm", type=float, default=1.0)
    p.add_argument("--host_n", type=float, default=None)
    p.add_argument("--out_csv", default="nk.csv")
    args = p.parse_args()

    lines = parse_gaussian_td_log(args.log)
    wt_frac = args.wt_percent/100.0
    N = number_density_from_wt_fraction(wt_frac, args.density_g_cm3, args.molar_mass_g_mol)

    lambdas = []
    L = args.lambda_min_nm
    while L <= args.lambda_max_nm + 1e-9:
        lambdas.append(L)
        L += args.lambda_step_nm

    table = build_dispersion(lines, args.gamma_eV, N, lambdas, args.host_n)

    with open(args.out_csv, "w", newline="") as fh:
        w = csv.writer(fh)
        w.writerow(["lambda_nm", "n", "k"])
```

```
        for row in table:
            w.writerow(row)

    if max(x[2] for x in table) < 1e-4:
        print("Note: k is very small; increase wt% or gamma_eV or check fj.")
    if wt_frac > 0.2:
        print("Warning: wt% > 20% may be unphysical; Lorentz-Lorenz may break down.")

if __name__ == "__main__":
    main()
```

How to run

```
python3 gaussian_to_nk.py \
  --log your_td.log \
  --molar_mass_g_mol 534.6 \
  --wt_percent 1.0 \
  --density_g_cm3 1.20 \
  --host_n 1.49 \
  --gamma_eV 0.12 \
  --lambda_min_nm 450 --lambda_max_nm 750 --lambda_step_nm 1 \
  --out_csv dye_in_PMMA_nk.csv
```

5 Example Gaussian Inputs

Geometry optimization

```
%chk=dye_opt.chk
#p B3LYP/6-31G(d) Opt

Title: Optimize dye geometry
0 1
<XYZ coordinates>
```

TD-DFT (20 states) with PCM

```
%chk=dye_opt.chk
#p TD(NStates=20) CAM-B3LYP/6-31+G(d,p) Geom=AllCheck Guess=Read SCRF=(PCM,Solvent=Chlorofo

Title: TD-DFT excited states
```

6 Driving S4 with the Dispersion

Define materials and layers, then sweep wavelength. For each λ, update the dye-film material with the corresponding (n, k) from the CSV and evaluate $R/T/A$ and fields. Sweep period, fill factor, and thickness to hit critical coupling or maximize phase slope.

7 Practical Design Checklist

- **RCWA convergence**: increase Fourier orders until $R + T + A$ stabilizes; metal gratings often need more orders.

- **Linewidths**: start with $\gamma \sim 50$–$120\,\mathrm{meV}$ and refine.
- **Host mixing**: use Lorentz–Lorenz to combine host and dye; do not simply add ε.
- **Orientation**: anisotropic films require tensor ε; start isotropic, then extend.
- **Validation**: compare a uniform film's absorbance against the n, k model before adding patterning.

8 Why This is Near-Term Publishable

The toolchain is mature (Gaussian TD-DFT, Multiwfn spectra, S4 RCWA), the physics bridge (Lorentz–Lorenz / Clausius–Mossotti) is classical and robust, and the experiments are accessible (spin-coated dye-doped polymers on simple gratings, or photochrome SAMs on commercial gratings).

References & Pointers

1. S4 (RCWA/FMM) solver and Lua interface: Liu et al., CPC 183, 2233 (2012).
2. Gaussian TD-DFT overview: Gaussian TD.
3. Multiwfn manual (spectrum utilities): Multiwfn.
4. Clausius–Mossotti / Lorentz–Lorenz: Wikipedia.
5. On oscillator-strength accuracy in TD-DFT: Benchmarking TD-DFT.
6. S4 code repository: victorliu/S4.