

# Designing Molecularly Programmed Metasurfaces with Gaussian, Multiwfn, and S4

2025-11-09

## 1 Overview

Combining **Gaussian** (quantum chemistry), **Multiwfn** (molecular analysis), and **S4** (RCWA metasurface solver) enables *molecularly programmed* metasurfaces. We outline (i) concrete project ideas, (ii) a physics-grounded workflow linking the tools, and (iii) a pipeline (with code) to convert TD-DFT results into dispersive  $n(\lambda)$ ,  $k(\lambda)$  for S4.

## 2 Potential Projects

Using molecular resonances inside photonic metasurfaces yields useful functions:

- **Molecularly Tuned Perfect Absorber (Critical Coupling):** Match a metasurface resonance to a dye’s excitonic line so incident light is absorbed in the dye layer. Impact: ultrathin absorbers for sensing and pixels. TD-DFT → Clausius–Mossotti mixing → S4 (S4).
- **All-Optical Switch with Photochromes:** Compute two states (cis/trans) to obtain two dispersions; design a metasurface at a high phase-sensitivity point to achieve large all-optical modulation.
- **Biosensor via Overlayer Index Shift:** Thin protein/DNA films shift narrow resonances; compute complex index from polarizability and predict detection limits in S4.
- **Polarization Control with Aligned Dyes:** Aligned dipoles yield anisotropic  $\epsilon$ ; simulate polarization-dependent phase in S4 with tensor permittivity.

*Easy path:* The perfect absorber (first item) is experimentally straightforward: a periodic pattern plus a spin-coated dye film and passive R/T measurements.

## 3 End-to-End Workflow

### A. Quantum Chemistry (Gaussian TD-DFT)

Optimize the ground state; compute excited states and oscillator strengths with TD-DFT (range-separated hybrids such as CAM-B3LYP or  $\omega$ B97X-D recommended). Optional PCM captures environment shifts. Gaussian prints lines like: **Excited State 1: Singlet-A 2.35 eV 527.5 nm f=0.51** (use energies  $E_j$  and strengths  $f_j$ ).

### B. Extract & Verify (Multiwfn)

Load LOG/fchk in Multiwfn’s spectrum module to list  $(E_j, f_j)$ ; sanity-check dominant visible transitions. See manual here.

## C. Molecules $\rightarrow$ Bulk $n, k$

**Lorentz polarizability:**

$$\alpha_{\text{mol}}(\omega) = \sum_j \frac{e^2}{m_e} \frac{f_j}{\omega_j^2 - \omega^2 - i\gamma_j\omega}, \quad \omega_j = E_j/\hbar. \quad (1)$$

**Clausius–Mossotti / Lorentz–Lorenz:**

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

Include host by adding  $X_{\text{host}} = (n_{\text{host}}^2 - 1)/(n_{\text{host}}^2 + 2)$  to  $X$ . Then  $\varepsilon_r = (n + ik)^2$ .

## D. Electromagnetics (S4 RCWA)

Define air / patterned dielectric / dye film / substrate. For each wavelength, set dye  $n(\lambda), k(\lambda)$ , compute  $R, T, A$ , and sweep geometry to achieve critical coupling. See S4 paper (Liu & Fan, CPC 2012).

## 4 TD-DFT $\rightarrow n(\lambda), k(\lambda)$ : Python

**Script (verbatim)**

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

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

EPS0 = 8.8541878128e-12; QE = 1.602176634e-19; ME = 9.1093837015e-31
HBAR = 1.054571817e-34; CO = 299792458.0; NA = 6.02214076e23; EV_TO_J = QE

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

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

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

def lorentz_alpha_mol(omega: float, lines: List[Tuple[float, float]], gamma_eV: float) -> complex:
    alpha = 0+0j; gamma = eV_to_gamma(gamma_eV); pref = (QE**2)/ME
    for E_eV, f in lines:
```

```

        wj = eV_to_omega(E_eV)
        alpha += pref * f / ((wj**2 - omega**2) - 1j*gamma*omega)
    return alpha

def lorentz_lorenz_X(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 = eps_r**0.5
    return (nr.real, abs(nr.imag))

def number_density(wt_frac: float, density_g_cm3: float, molar_mass_g_mol: float) -> float:
    rho = density_g_cm3*1000.0; M = molar_mass_g_mol/1000.0
    return ((wt_frac*rho)/M)*NA

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

def main():
    ap = argparse.ArgumentParser(description='TD-DFT to n,k via Lorentz-Lorenz')
    ap.add_argument('--log', required=True)
    ap.add_argument('--molar_mass', type=float, required=True)
    ap.add_argument('--wt_percent', type=float, default=1.0)
    ap.add_argument('--density', type=float, default=1.2)
    ap.add_argument('--gamma', type=float, default=0.10)
    ap.add_argument('--host_n', type=float, default=None)
    ap.add_argument('--lambda_min', type=float, default=400.0)
    ap.add_argument('--lambda_max', type=float, default=800.0)
    ap.add_argument('--lambda_step', type=float, default=1.0)
    ap.add_argument('--out', default='nk.csv')
    args = ap.parse_args()

    lines = parse_gaussian_td_log(args.log)
    N = number_density(args.wt_percent/100.0, args.density, args.molar_mass)
    lambdas = [args.lambda_min + i*args.lambda_step
                for i in range(int((args.lambda_max-args.lambda_min)/args.lambda_step)+1)]
    table = build_dispersion(lines, args.gamma, N, lambdas, host_n=args.host_n)
    with open(args.out, 'w', newline='') as fh:
        w = csv.writer(fh); w.writerow(['lambda_nm', 'n', 'k']); w.writerows(table)
    if max(x[2] for x in table) < 1e-4:

```

```

        print('Note: k very small; increase wt% or gamma or check f_j')
    if args.wt_percent > 20.0:
        print('Warning: wt% > 20% may be unphysical; mixing may break down')

if __name__ == '__main__':
    main()

```

## 5 Using S4

1. Define materials/layers: air / grating (e.g., TiO<sub>2</sub> or Si<sub>3</sub>N<sub>4</sub>) / dye film / substrate (glass).
2. For each  $\lambda$ , set dye  $n, k$  from CSV; in Lua, update material, set frequency (e.g., `S:SetFrequency(c0/lambda)`), compute  $R, T$ .
3. Sweep period, height, and film thickness to reach *critical coupling* ( $R \approx 0, T \approx 0$  at  $\lambda_0$ ).
4. Inspect fields; ensure energy concentrates in the dye at resonance.

**Tips** Converge Fourier orders; choose realistic  $\gamma$  (50–120 meV typical); respect dye solubility; host index ( $\tilde{1.5}$ ) helps mode overlap; validate with a uniform film before patterning.

## 6 Example Next Steps

Pick a strong visible dye (peak  $\sim 600$  nm); generate  $n, k$  at 1 wt% in PMMA (1.2 g/cm<sup>3</sup>); design a 1D TiO<sub>2</sub> grating (period  $\sim \lambda/n_{\text{eff}}$ ) with a 50 nm dye film; optimize for  $A = 1 - R - T$  peak near 600 nm; then fabricate (nanoimprint or commercial grating) and measure R/T.

## References

S4 RCWA solver: Liu & Fan, CPC 183, 2233 (2012). Gaussian TD-DFT outputs: guide. Lorentz–Lorenz relation: overview. TD-DFT oscillator strength benchmarking: study.