```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
  graphs.py - Python code for the generation of graphs for manuscript
              "Photoinduced flipping of optical chirality during
              backward-wave parametric amplification in a chiral
              nonlinear medium" by Christos Flytzanis, Fredrik Jonsson
              and Govind Agrawal (April 2025).

Created on Wed Apr  2 14:53:08 2025
Copyright (C) 2025 under GPL 3.0, Fredrik Jonsson
"""
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.ticker import AutoLocator, AutoMinorLocator
from mpl_toolkits.axes_grid1 import make_axes_locatable
from stokes import saveStokesParameters

"""
As a global standard, use TeX-style labeling for everything graphics-related.
"""
plt.rcParams.update({
    "text.usetex" : True,
    "font.family" : "Computer Modern",
    "font.size"   : 12
})

def stokesparams(ap, am, normalize=True):
    """
    Compute Stokes parameters for the input field amplitudes $A_+$ and $A_-$,
    expressed in a circularly polarized basis.

    Parameters
    ----------
    ap : complex, np.array
        The LCP field envelope $A_+$.
    am : complex, np.array
        The RCP field envelope $A_-$.
    normalize : bool, optional
        If set to True, then normalize the Stokes parameters (S1,S2,S3) by S0
        before returning. This way, (S1,S2,S3) describe points on a sphere,
        enabling mapping onto the unitary PoincarÃ© sphere straight away.
        The default is True.

    Returns
    -------
    s0, s1, s2, s3 : float, np.array
        The Stokes parameters corresponding to the field envelopes $A_+$
        and $A_-$.
    """
    aap, aam = np.copy(ap), np.copy(am)
    s0 = np.square(np.absolute(aap))+np.square(np.absolute(aam))
    s1 = 2.0*np.real(np.multiply(np.conjugate(aap),aam))
    s2 = 2.0*np.imag(np.multiply(np.conjugate(aap),aam))
    s3 = np.square(np.absolute(aap))-np.square(np.absolute(aam))
    if normalize:
        s1 = np.divide(s1,s0)
        s2 = np.divide(s2,s0)
        s3 = np.divide(s3,s0)
    return s0, s1, s2, s3

def kappal(delta, beta):
    """
    Compute the $\kappa_{\pm}L/2$ coefficients, normalized by the length $L$
    and divided by 2. In terms of the normalized parameters, the definition
    of the returned variables yields $\kappa_{\pm}L/2=\delta(1\pm\beta)$.
```

```

    Parameters
    ----------
    delta : float
        The normalized electric dipolar quasi phase mismatch remainder
        $\delta={{\Delta kL}\over{2}}-{{2\pi L}\over{2\Lambda}}$ against
        the quasi phase matching period.
    beta : float
        The normalized nonlocal contribution (or "correction") to the
        phase mismatch $\Delta k$, defined as
        $\beta={{\Delta\alpha}\over{\Delta k-2\pi/\Lambda}}$

    Returns
    -------
    kappalp : float
        The coefficient $\kappa_+L/2$ for left circular polarization (LCP).
    kappalm : float
        The coefficient $\kappa_-L/2$ for right circular polarization (RCP).
    """
    kappalp = delta*(1.0+beta)
    kappalm = delta*(1.0+beta)
    return kappalp, kappalm

def bl(delta, beta, eta):
    """
    Compute the $b_{\pm}L$ coefficients, normalized by the length $L$. In
    terms of the normalized parameters, the definition of the returned
    variables yields $b_{\pm}L=(\delta^2(1\pm\beta)^2+\eta)^{1/2}$.

    Parameters
    ----------
    delta : float
        The normalized electric dipolar quasi phase mismatch remainder
        $\delta={{\Delta kL}\over{2}}-{{2\pi L}\over{2\Lambda}}$ against
        the quasi phase matching period.
    beta : float
        The normalized nonlocal contribution (or "correction") to the
        phase mismatch $\Delta k$, defined as
        $\beta={{\Delta\alpha}\over{\Delta k-2\pi/\Lambda}}$
    eta : float
        Pump intensity normalized against the threshold intensity,
        $\eta=({{\pi}/{2}})^2{{I_{\rm pump}}/{I_{\rm th}}}$

    Returns
    -------
    blp : float, np.array
        The coefficient $b_+L$ for left circular polarization (LCP).
    blm : float, np.array
        The coefficient $b_-L$ for right circular polarization (RCP).
    """
    kappalp, kappalm = kappal(delta, beta)

    blp = np.sqrt(np.multiply(np.square(delta),np.square(1.0+beta))+eta)
    blm = np.sqrt(np.multiply(np.square(delta),np.square(1.0-beta))+eta)
    return blp, blm

def gain(delta, beta, eta, verbose=True):
    """
    Compute the LCP (+) and RCP (-) signal gain $G_+$ and $G_-$.

    Parameters
    ----------
    delta : float
        The normalized electric dipolar quasi phase mismatch remainder
        $\delta={{\Delta kL}\over{2}}-{{2\pi L}\over{2\Lambda}}$ against
        the quasi phase matching period.
    beta : float
        The normalized nonlocal contribution (or "correction") to the
```

```python
        phase mismatch $\Delta k$, defined as
        $\beta={{\Delta\alpha}\over{\Delta k-2\pi/\Lambda}}$
    eta : float
        Pump intensity normalized against the threshold intensity,
        $\eta=({{\pi}/{2}})^2{{I_{\rm pump}}/{I_{\rm th}}}$

    Returns
    -------
    ggp : float
        The LCP (+) signal gain $G_+$.
    ggm : float
        The RCP (-) signal gain $G_-$.
    """
    d2 = np.square(delta)
    b2p = np.square(1.0+beta)
    b2m = np.square(1.0-beta)
    tp = np.sqrt(np.multiply(d2,b2p)+eta)
    tm = np.sqrt(np.multiply(d2,b2m)+eta)
    cos2p = np.square(np.cos(tp))
    cos2m = np.square(np.cos(tm))
    sin2p = np.square(np.sin(tp))
    sin2m = np.square(np.sin(tm))
    sp = np.multiply(d2,b2p)
    sm = np.multiply(d2,b2m)
    ggp = np.divide(tp, np.multiply(tp,cos2p)+np.multiply(sp,sin2p))
    ggm = np.divide(tm, np.multiply(tm,cos2m)+np.multiply(sm,sin2m))
    if verbose:
        print("LCP: max(G+)=%1.4f, min(G+)=%1.4f"%(np.max(ggp),np.min(ggp)))
        print("RCP: max(G-)=%1.4f, min(G-)=%1.4f"%(np.max(ggm),np.min(ggm)))
    return ggp, ggm

def afsignal(zn, delta, beta, eta, verbose=True):
    """
    Compute the forward traveling signal envelopes $A^{f+}_{\omega_2}(z)$
    (LCP) and  $A^{f-}_{\omega_2}(z)$ (RCP) as function of normalized
    parameters and normalized distance z/L.

    Parameters
    ----------
    zn : float, np.array
        The normalized spatial coordinate $z/L$.
    delta : float
        The normalized electric dipolar quasi phase mismatch remainder
        $\delta={{\Delta kL}\over{2}}-{{2\pi L}\over{2\Lambda}}$ against
        the quasi phase matching period.
    beta : float
        The normalized nonlocal contribution (or "correction") to the
        phase mismatch $\Delta k$, defined as
        $\beta={{\Delta\alpha}\over{\Delta k-2\pi/\Lambda}}$
    eta : float
        Pump intensity normalized against the threshold intensity,
        $\eta=({{\pi}/{2}})^2{{I_{\rm pump}}/{I_{\rm th}}}$

    Returns
    -------
    asp : float, np.array
        The envelope of the LCP (+) forward traveling signal
        $A^{f+}_{\omega_2}(z)$.
    asm : TYPE
        The envelope of the RCP (-) forward traveling signal
        $A^{f-}_{\omega_2}(z)$.
    """
    kappalp, kappalm = kappal(delta, beta)
    blp, blm = bl(delta, beta, eta)
    kbp, kbm = np.divide(kappalp,blp), np.divide(kappalm,blm)
    czp, szp = np.cos(blp*zn), np.sin(blp*zn)
    czm, szm = np.cos(blm*zn), np.sin(blm*zn)
    clp, slp = np.cos(blp), np.sin(blp)
```

```
    clm, slm = np.cos(blm), np.sin(blm)
    expkzp, expkzm = np.exp(1j*kappalp*zn), np.exp(1j*kappalm*zn)
    if verbose:
        print("b_+L = %f"%(blp))
        print("b_-L = %f"%(blm))
        print("kappa_+L/2 = %f"%(kappalp))
        print("kappa_-L/2 = %f"%(kappalm))

    """ Compute LCP component $A^{f+}_{\omega_2}(z)/A^{f+}_{\omega_2}(0)$ """
    asp = np.divide(slp-1j*kbp*clp, clp+1j*kbp*slp)
    asp = czp + np.multiply(asp, szp)
    asp = np.multiply(asp, expkzp)

    """ Compute RCP component $A^{f-}_{\omega_2}(z)/A^{f-}_{\omega_2}(0)$ """
    asm = np.divide(slm-1j*kbm*clm, clm+1j*kbm*slm)
    asm = czm + np.multiply(asm, szm)
    asm = np.multiply(asm, expkzm)

    return asp, asm

def makegraph_01(bw=True,printtitle=False,plots1s2=False):
    """
    Internally used parameters
    ----------
    zn : float, np.array
        Normalized spatial coordinate z/L, typcially in range 0 â\211¤ z/L â\211¤ 1.
    delta : float
        The normalized electric dipolar quasi phase mismatch remainder
        $\delta={{\Delta kL}\over{2}}-{{2\pi L}\over{2\Lambda}}$ against
        the quasi phase matching period.
    beta : float
        The normalized nonlocal contribution (or "correction") to the
        phase mismatch $\Delta k$, defined as
        $\beta={{\Delta\alpha}\over{\Delta k-2\pi/\Lambda}}$
    eta : float
        Pump intensity normalized against the threshold intensity,
        $\eta=({{\pi}/{2}})^2{{I_{\rm pump}}/{I_{\rm th}}}$
    """
    print("======= Generating image set 01 (graph-01-*) =======")
    zn = np.linspace(0.0, 1.0, 1024)
    deltamin, deltamax, numdelta = 0.5, 1.5, 3
    betamin, betamax, numbeta = -2.0, 2.0, 5
    deltarange = np.linspace(deltamin, deltamax, numdelta)
    betarange = np.linspace(betamax, betamin, numbeta)
    eta = 2.0

    """
    Define colors and linestyles for the five curves to be mapped for each
    value of the chiral coefficient beta.
    """
    dashdotdotted = (0, (5, 1, 1, 1, 1, 1))  # Custom 'â\200\224Â•Â•â\200\224Â•Â•â\200\224Â•Â•â\200\224'
    colors=['xkcd:red','xkcd:green','xkcd:azure','xkcd:tan','xkcd:teal']
    linestyles=['dashed','dashdot','dotted',dashdotdotted,'solid']

    for delta in deltarange:
        if plots1s2: # Plot all of S0, S1, S2 and S3
            fig, ax = plt.subplots(3,figsize=(5.4,5.0))
        else: # Plot only S0 and S3
            fig, ax = plt.subplots(2,figsize=(5.4,3.8))
        for k, beta in enumerate(betarange):
            """
            For each value for the electric dipolar phase mismatch against
            the nominal QPM period, generate a separate graph of the Stokes
            parameters S0, S1, S2 and S3.
            """
            asp, asm = afsignal(zn, delta, beta, eta)
            s0, s1, s2, s3 = stokesparams(asp/np.sqrt(2.0), asm/np.sqrt(2.0))
```

```
            labeltext='$\\beta=%1.1f$'%(beta)
            if bw:
                color = 'k'
                linestyle = linestyles[k]
            else:
                color = colors[k]
                linestyle = 'solid'
            ax[0].semilogy(zn, s0, color=color, linestyle=linestyle, label=labeltext)
            if plots1s2:
                ax[1].plot(zn, s1, color=color, linestyle=linestyle, label=labeltext)
                ax[1].plot(zn, s2, color=color, linestyle=linestyle)
                ax[2].plot(zn, s3, color=color, linestyle=linestyle, label=labeltext)
            else:
                ax[1].plot(zn, s3, color=color, linestyle=linestyle, label=labeltext)

            """
            Save each generated trajectory of (S0,S1,S2,S3) as a separate data
            set for postprocessing by, for example, the PoincarÃ© program.
            """
            filename = "data/data-01-delta-%1.2f-beta-%1.2f.dat"%(delta,beta)
            saveStokesParameters(zn, s0, s1, s2, s3, filename)

        ax[1].yaxis.set_minor_locator(AutoMinorLocator(5))
        if plots1s2:
            ax[2].yaxis.set_minor_locator(AutoMinorLocator(5))

        for j in range(3 if plots1s2 else 2):
            ax[j].autoscale(enable=True, axis='x', tight=True)
            ax[j].legend(loc='upper right', fontsize=9, handlelength=4)
            ax[j].grid(visible=True, which='major', axis='both')
            ax[j].tick_params(which="both", top=True, right=True,
                              labeltop=False, bottom=True, labelbottom=True,
                              direction="in")

        if printtitle:
            ax[0].set_title("Dipolar QPM mismatch $(\\Delta k-2\\pi/\\Lambda)L/2"
                            "=$%1.1f"%(delta))
        ax[0].set_ylabel("$S_0(z)/S_0(0)$")
        if plots1s2:
            ax[1].set_ylim(-1.05,1.05)
            ax[1].set_ylabel("$\\vbox{\\hbox{$S_1(z)/S_0(z),$}"
                             "\\hbox{$S_2(z)/S_0(z)$}}$")
            ax[2].set_ylim(-1.05,1.05)
            ax[2].set_ylabel("$S_3(z)/S_0(z)$")
            ax[2].set_xlabel("$z/L$")
        else:
            ax[1].set_ylim(-1.05,1.05)
            ax[1].set_ylabel("$S_3(z)/S_0(z)$")
            ax[1].set_xlabel("$z/L$")
        kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
        basename = "graphs/graph-01-delta-%1.2f"%delta
        if bw:
            basename += "-bw"
        if plots1s2:
            basename += "-all"
        for fmt in ['eps','svg','png']:
            fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
    return

def makegraph_02():
    """
    Graph of the single-pass gain of the signal as function of normalized
    chiral phase mismatch and at a set of pump intensity levels.
    Parameters:
```

```python
        beta: Normalized chiral mismatch, $\beta=\Delta\alpha L/2$
        eta: Normalized pump intensity, $\eta=I_{pump}/I_{th}$
    """
    print("======= Generating image set 02 (graph-02-*) =======")
    beta = np.linspace(-7.0,7.0,1000)
    fig, ax = plt.subplots(figsize=(5.4,4.0))
    for eta in np.linspace(2.0, 5.0, 4):
        delta = np.sqrt(np.square(beta)+eta)
        gg = np.divide(1.0,(np.square(np.cos(delta))
                          + ((beta/delta)**2)*np.square(np.sin(delta))))
        ax.semilogy(beta, gg, label='$\\eta=$%1.1f'%(eta))

    ax.autoscale(enable=True, axis='x', tight=True)
    ax.legend(loc='upper right', fontsize=11)
    ax.tick_params(axis="both",direction="in")
    ax.grid(visible=True, which='both', axis='both')
    ax.set_xlabel("$\\Delta\\alpha L/2$")
    ax.set_ylabel("$G_{\\pm}=|A^{\\pm}_{\\rm s}(L)/A^{\\pm}_{\\rm s}(0)|^2$")

    kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
    basename = "graphs/graph-02"
    for fmt in ['eps','svg','png']:
        fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

    return

def makegraph_03():
    """
    Graph of the single-pass gain of the signal as function of normalized
    electric dipolar and chiral phase mismatch.
    Parameters:
        beta  : Normalized chiral term \Delta\alpha/(\Delta k-2\pi/\Lambda)
        delta : Normalized dipolar mismatch (\Delta k-2\pi/\Lambda)L/2
        eta   : Normalized pump intensity, $\eta=I_{pump}/I_{th}$
    """
    print("======= Generating image set 03 (graph-03-*) =======")
    betamax = 2.0  # This is intended to be funny
    deltamax = betamax
    beta = np.linspace(-betamax, betamax, 2048)
    delta = np.linspace(-deltamax, deltamax, 2048)
    betag, deltag = np.meshgrid(beta, delta, indexing='xy')
    eta = 5.0
    ggp, ggm = gain(deltag, betag, eta)

    dx = (beta[1]-beta[0])/2.
    dy = (delta[1]-delta[0])/2.
    extent = [beta[0]-dx, beta[-1]+dx, delta[0]-dy, delta[-1]+dy]
    clevels = np.linspace(-4,4,9)

    ggp_db = 10*np.log10(ggp)
    ggm_db = 10*np.log10(ggm)

    """
    Map the gain G_+ and G_- as a surface, being a function of the normalized
    electric dipolar phase mismatch and its chiral contribution.
    """
    fig, ax = plt.subplots(figsize=(5.4,5.4),subplot_kw={"projection": "3d"})
    ax.plot_surface(betag,deltag,ggp_db,vmin=2.2*ggp_db.min(),cmap=cm.Oranges)
    ax.plot_surface(betag,deltag,ggm_db,vmin=2.2*ggp_db.min(),cmap=cm.Blues)
    ax.contour(betag, deltag, ggp_db, [-4,-3,-2,-1,0,1,2,3,4], zdir='z',
               offset=np.min(ggp_db)-1, cmap=cm.Oranges)
    ax.contour(betag, deltag, ggm_db, [-4,-3,-2,-1,0,1,2,3,4], zdir='z',
               offset=np.min(ggm_db)-1, cmap=cm.Blues)
    ax.autoscale(enable=True, axis='x', tight=True)
    ax.set_xlabel("$\\beta=\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
    ax.set_ylabel("$\\delta=(\\Delta k-2\\pi/\\Lambda)L/2$")
    ax.set_zlabel("$G_+$, $G_+$ (gain)")
    kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
```

```
    basename = "graphs/graph-03-surf"
    for fmt in ['eps','svg','png']:
        fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

    """
    Map the LCP gain G_+ as a surface, being a function of the normalized
    electric dipolar phase mismatch and its chiral contribution.
    """
    fig, ax = plt.subplots(figsize=(5.4,5.4))
    pos = ax.imshow(ggp_db, extent=extent, cmap=cm.Oranges)
    divider = make_axes_locatable(plt.gca())
    cax = divider.append_axes("right", "5%", pad="3%")
    plt.colorbar(pos, cax=cax)
    plt.tight_layout()
    cs = ax.contour(betag, deltag, ggp_db, clevels, colors='k')
    ax.clabel(cs, cs.levels, fontsize=12)
    ax.autoscale(enable=True, axis='xy', tight=True)
    ax.set_xlabel("$\\beta=\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
    ax.set_ylabel("$\\delta=(\\Delta k-2\\pi/\\Lambda)L/2$")
    ax.set_title("LCP gain $G_+$")
    kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
    basename = "graphs/graph-03-gplus-image"
    for fmt in ['eps','svg','png']:
        fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

    """
    Map the RCP gain G_- as a surface, being a function of the normalized
    electric dipolar phase mismatch and its chiral contribution.
    """
    fig, ax = plt.subplots(figsize=(5.4,5.4))
    pos = ax.imshow(ggm_db, extent=extent, cmap=cm.Blues)
    divider = make_axes_locatable(plt.gca())
    cax = divider.append_axes("right", "5%", pad="3%")
    plt.colorbar(pos, cax=cax)
    plt.tight_layout()
    cs = ax.contour(betag, deltag, ggm_db, clevels, colors='k')
    ax.clabel(cs, cs.levels, fontsize=12)
    ax.autoscale(enable=True, axis='xy', tight=True)
    ax.set_xlabel("$\\beta=\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
    ax.set_ylabel("$\\delta=(\\Delta k-2\\pi/\\Lambda)L/2$")
    ax.set_title("RCP gain $G_-$")
    kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
    basename = "graphs/graph-03-gminus-image"
    for fmt in ['eps','svg','png']:
        fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

    return

def makegraph_04():
    """
    Graph of the single-pass Stokes parameter gain S_0(L)/S_0(0) of the signal
    as function of normalized electric dipolar and chiral phase mismatch.
    Parameters:
        beta  : Normalized chiral term \Delta\alpha/(\Delta k-2\pi/\Lambda)
        delta : Normalized dipolar mismatch (\Delta k-2\pi/\Lambda)L/2
        eta   : Normalized pump intensity, $\eta=I_{pump}/I_{th}$
    """
    print("======= Generating image set 04 (graph-04-*) =======")
    betamax = 2.0  # This is intended to be funny
    deltamax = betamax
    beta = np.linspace(-betamax, betamax, 2048)
    delta = np.linspace(-deltamax, deltamax, 2048)
    betag, deltag = np.meshgrid(beta, delta, indexing='xy')
    dx = (beta[1]-beta[0])/2.
    dy = (delta[1]-delta[0])/2.
    extent = [beta[0]-dx, beta[-1]+dx, delta[0]-dy, delta[-1]+dy]

    eta = 5.0
```

```
    ggp, ggm = gain(deltag, betag, eta)
    s0 = (ggp+ggm)/2.0
    clevels = np.linspace(0.5,2.5,5)

    fig, ax = plt.subplots(figsize=(5.4,5.4),subplot_kw={"projection": "3d"})
    ax.plot_surface(betag, deltag, s0, cmap=cm.Blues)
    ax.contour(betag, deltag, s0, clevels, zdir='z', offset=np.min(s0),
               cmap=cm.Blues)
    ax.autoscale(enable=True, axis='xy', tight=True)
    ax.set_xlabel("$\\beta=\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
    ax.set_ylabel("$\\delta=(\\Delta k-2\\pi/\\Lambda)L/2$")
    ax.set_zlabel("$S_0(L)/S_0(0)$ (gain)")

    kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
    basename = "graphs/graph-04-s0-surface"
    for fmt in ['eps','svg','png']:
        fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

    """
    Map S_0(L)/S_0(0), being a function of the normalized electric dipolar
    phase mismatch and its chiral contribution, as an image with overlaid
    contours.
    """
    fig, ax = plt.subplots(figsize=(5.4,5.4))
    pos = ax.imshow(s0, extent=extent, cmap=cm.Blues)
    divider = make_axes_locatable(plt.gca())
    cax = divider.append_axes("right", "5%", pad="3%")
    plt.colorbar(pos, cax=cax)
    plt.tight_layout()
    cs = ax.contour(betag, deltag, s0, clevels, colors='k')
    ax.clabel(cs, cs.levels, fontsize=12)
    ax.autoscale(enable=True, axis='xy', tight=True)
    ax.set_xlabel("$\\beta=\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
    ax.set_ylabel("$\\delta=(\\Delta k-2\\pi/\\Lambda)L/2$")
    ax.set_title("Intensity gain $S_0(L)/S_0(0)$, "
                 "$I_{\\rm pump}/I_{\\rm th}=$%1.1f"%(eta))
    kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
    basename = "graphs/graph-04-s0-%1.2f-image"%eta
    for fmt in ['eps','svg','png']:
        fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

    """
    Identical to previous plot, but purely with contours in black, no other
    image data or colors.
    """
    fig, ax = plt.subplots(figsize=(5.4,5.4))
    cs = ax.contour(betag, deltag, s0, clevels, zdir='z', offset=np.min(s0),
                    colors='k')
    ax.clabel(cs, cs.levels, fontsize=12)
    ax.autoscale(enable=True, axis='xy', tight=True)
    ax.set_xlabel("$\\beta=\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
    ax.set_ylabel("$\\delta=(\\Delta k-2\\pi/\\Lambda)L/2$")
    kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
    basename = "graphs/graph-04-s0-contour-%1.2f-black"%eta
    for fmt in ['eps','svg','png']:
        fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

    return

def makegraph_05(printtitle=False):
    """
    Graph of the single-pass Stokes parameter S_3(L)/S_0(0) of the signal
    as function of normalized electric dipolar and chiral phase mismatch.
    Parameters:
        beta  : Normalized chiral term \Delta\alpha/(\Delta k-2\pi/\Lambda)
        delta : Normalized dipolar mismatch (\Delta k-2\pi/\Lambda)L/2
        eta   : Normalized pump intensity, $\eta=I_{pump}/I_{th}$
    """
```

```python
    print("======= Generating image set 05 (graph-05-*) =======")
    betamax = 2.0  # This is intended to be funny
    deltamax = betamax
    beta = np.linspace(-betamax, betamax, 2048)
    delta = np.linspace(-deltamax, deltamax, 2048)
    betag, deltag = np.meshgrid(beta, delta, indexing='xy')
    dx = (beta[1]-beta[0])/2.
    dy = (delta[1]-delta[0])/2.
    extent = [beta[0]-dx, beta[-1]+dx, delta[0]-dy, delta[-1]+dy]

    for eta in [2.0, 3.0, 4.0, 5.0]:
        ggp, ggm = gain(deltag, betag, eta)
        s3 = (ggp-ggm)/(ggp+ggm)
        clevels = np.linspace(-1.0,1.0,9)

        """
        Map S_3/S_0, being a function of the normalized electric dipolar phase
        mismatch and its chiral contribution, as a surface plot with contours
        underneath.
        """
        fig, ax = plt.subplots(figsize=(5.4,5.4),subplot_kw={"projection":"3d"})
        ax.plot_surface(betag, deltag, s3, cmap=cm.Blues)
        ax.contour(betag, deltag, s3, clevels, zdir='z', offset=np.min(s3),
                   cmap=cm.Blues)
        ax.autoscale(enable=True, axis='xy', tight=True)
        ax.set_xlabel("$\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
        ax.set_ylabel("$(\\Delta k-2\\pi/\\Lambda)L/2$")
        ax.set_zlabel("$S_3(L)/S_0(L)$")

        kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
        basename = "graphs/graph-05-s3-%1.2f-surface"%eta
        for fmt in ['eps','svg','png']:
            fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

        """
        Map S_3/S_0, being a function of the normalized electric dipolar phase
        mismatch and its chiral contribution, as plain black contours without
        any image of color. Useful for plain printing in B/W.
        """
        fig, ax = plt.subplots(figsize=(5.4,5.4))
        cs = ax.contour(betag, deltag, s3, clevels, zdir='z', offset=np.min(s3),
                        vmin=0.1*s3.min(), colors='k')
        ax.clabel(cs, cs.levels, fontsize=12)
        ax.autoscale(enable=True, axis='xy', tight=True)
        ax.set_xlabel("$\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
        ax.set_ylabel("$(\\Delta k-2\\pi/\\Lambda)L/2$")

        kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
        basename = "graphs/graph-05-s3-%1.2f-bw"%eta
        for fmt in ['eps','svg','png']:
            fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

        """
        Map S_3/S_0, being a function of the normalized electric dipolar phase
        mismatch and its chiral contribution, as an image with overlaid black
        contours.
        """
        cmp = LinearSegmentedColormap.from_list("",
                    ["xkcd:azure","xkcd:white","xkcd:orangered"])
        fig, ax = plt.subplots(figsize=(5.4,5.4))
        pos = ax.imshow(s3, extent=extent, cmap=cmp, vmin=-1.0, vmax=1.0)
        divider = make_axes_locatable(plt.gca())
        cax = divider.append_axes("right", "5%", pad="3%")
        plt.colorbar(pos, cax=cax)
        plt.tight_layout()
        cs = ax.contour(betag, deltag, s3, clevels, colors='k')
        ax.clabel(cs, cs.levels, fontsize=12)
        ax.autoscale(enable=True, axis='xy', tight=True)
```

```
        ax.set_xlabel("$\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
        ax.set_ylabel("$(\\Delta k-2\\pi/\\Lambda)L/2$")
        if printtitle:
            ax.set_title("Ellipticity $S_3(L)/S_0(L)$, "
                         "$I_{\\rm pump}/I_{\\rm th}=$%1.1f"%(eta))
        kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
        basename = "graphs/graph-05-s3-%1.2f-image"%eta
        for fmt in ['eps','svg','png']:
            fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)

    return

def main() -> None:
    for bw in [True,False]: # Generate image set 01 (graph-01-*.[eps|png|svg])
        for plots1s2 in [True,False]:
            makegraph_01(bw=bw, plots1s2=plots1s2)
    makegraph_02()     # Generate image set 02 (graph-02-*.[eps|png|svg])
    makegraph_03()     # Generate image set 03 (graph-03-*.[eps|png|svg])
    makegraph_04()     # Generate image set 04 (graph-04-*.[eps|png|svg])
    makegraph_05()     # Generate image set 05 (graph-05-*.[eps|png|svg])
    return

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