

```
1: #!/usr/bin/env python3
2: # -*- coding: utf-8 -*-
3: """
4:     graphs.py - Python code for the generation of graphs for manuscript
5:                 "Photoinduced flipping of optical chirality during
6:                 backward-wave parametric amplification in a chiral
7:                 nonlinear medium" by Christos Flytzanis, Fredrik Jonsson
8:                 and Govind Agrawal (April 2025).
9:
10: Created on Wed Apr  2 14:53:08 2025
11: Copyright (C) 2025 under GPL 3.0, Fredrik Jonsson
12: """
13: import numpy as np
14: import matplotlib.pyplot as plt
15: from matplotlib import cm
16: from matplotlib.colors import LinearSegmentedColormap
17: from matplotlib.ticker import AutoLocator, AutoMinorLocator
18: from mpl_toolkits.axes_grid1 import make_axes_locatable
19: from stokes import saveStokesParameters
20:
21: """
22: As a global standard, use TeX-style labeling for everything graphics-related.
23: """
24: plt.rcParams.update({
25:     "text.usetex" : True,
26:     "font.family" : "Computer Modern",
27:     "font.size"   : 12
28: })
29:
30: def stokesparams(ap, am, normalize=True):
31:     """
32:     Compute Stokes parameters for the input field amplitudes  $A_+$  and  $A_-$ ,
33:     expressed in a circularly polarized basis.
34:
35:     Parameters
36:     -----
37:     ap : complex, np.array
38:         The LCP field envelope  $A_+$ .
39:     am : complex, np.array
40:         The RCP field envelope  $A_-$ .
41:     normalize : bool, optional
42:         If set to True, then normalize the Stokes parameters  $(S_1, S_2, S_3)$  by  $S_0$ 
43:         before returning. This way,  $(S_1, S_2, S_3)$  describe points on a sphere,
44:         enabling mapping onto the unitary Poincaré sphere straight away.
45:         The default is True.
46:
47:     Returns
48:     -----
49:     s0, s1, s2, s3 : float, np.array
50:         The Stokes parameters corresponding to the field envelopes  $A_+$ 
51:         and  $A_-$ .
52:     """
53:     aap, aam = np.copy(ap), np.copy(am)
54:     s0 = np.square(np.absolute(aap)) + np.square(np.absolute(aam))
55:     s1 = 2.0*np.real(np.multiply(np.conjugate(aap), aam))
56:     s2 = 2.0*np.imag(np.multiply(np.conjugate(aap), aam))
57:     s3 = np.square(np.absolute(aap)) - np.square(np.absolute(aam))
58:     if normalize:
59:         s1 = np.divide(s1, s0)
60:         s2 = np.divide(s2, s0)
61:         s3 = np.divide(s3, s0)
62:     return s0, s1, s2, s3
63:
64: def kappal(delta, beta):
65:     """
66:     Compute the  $\kappa_{\pm} L/2$  coefficients, normalized by the length  $L$ 
67:     and divided by 2. In terms of the normalized parameters, the definition
68:     of the returned variables yields  $\kappa_{\pm} L/2 = \delta(1 \pm \beta)$ .
```

```

69:
70:     Parameters
71:     -----
72:     delta : float
73:         The normalized electric dipolar quasi phase mismatch remainder
74:          $\delta = \{\frac{\Delta kL}{2}\} - \{\frac{2\pi L}{2\Lambda}\}$  against
75:         the quasi phase matching period.
76:     beta : float
77:         The normalized nonlocal contribution (or "correction") to the
78:         phase mismatch  $\Delta k$ , defined as
79:          $\beta = \{\frac{\Delta\alpha}{\Delta k - 2\pi/\Lambda}\}$ 
80:
81:     Returns
82:     -----
83:     kappalp : float
84:         The coefficient  $\kappa_{+L/2}$  for left circular polarization (LCP).
85:     kappalm : float
86:         The coefficient  $\kappa_{-L/2}$  for right circular polarization (RCP).
87:     """
88:     kappalp = delta*(1.0+beta)
89:     kappalm = delta*(1.0+beta)
90:     return kappalp, kappalm
91:
92: def bl(delta, beta, eta):
93:     """
94:     Compute the  $b_{\pm L}$  coefficients, normalized by the length  $L$ . In
95:     terms of the normalized parameters, the definition of the returned
96:     variables yields  $b_{\pm L} = (\delta^2(1 \pm \beta)^2 + \eta)^{1/2}$ .
97:
98:     Parameters
99:     -----
100:    delta : float
101:        The normalized electric dipolar quasi phase mismatch remainder
102:         $\delta = \{\frac{\Delta kL}{2}\} - \{\frac{2\pi L}{2\Lambda}\}$  against
103:        the quasi phase matching period.
104:    beta : float
105:        The normalized nonlocal contribution (or "correction") to the
106:        phase mismatch  $\Delta k$ , defined as
107:         $\beta = \{\frac{\Delta\alpha}{\Delta k - 2\pi/\Lambda}\}$ 
108:    eta : float
109:        Pump intensity normalized against the threshold intensity,
110:         $\eta = (\{\pi/2\})^2 \{I_{\text{pump}}/I_{\text{th}}\}$ 
111:
112:    Returns
113:    -----
114:    blp : float, np.array
115:        The coefficient  $b_{+L}$  for left circular polarization (LCP).
116:    blm : float, np.array
117:        The coefficient  $b_{-L}$  for right circular polarization (RCP).
118:    """
119:    kappalp, kappalm = kappal(delta, beta)
120:
121:    blp = np.sqrt(np.multiply(np.square(delta), np.square(1.0+beta))+eta)
122:    blm = np.sqrt(np.multiply(np.square(delta), np.square(1.0-beta))+eta)
123:    return blp, blm
124:
125: def gain(delta, beta, eta, verbose=True):
126:     """
127:     Compute the LCP (+) and RCP (-) signal gain  $G_{+}$  and  $G_{-}$ .
128:
129:     Parameters
130:     -----
131:    delta : float
132:        The normalized electric dipolar quasi phase mismatch remainder
133:         $\delta = \{\frac{\Delta kL}{2}\} - \{\frac{2\pi L}{2\Lambda}\}$  against
134:        the quasi phase matching period.
135:    beta : float
136:        The normalized nonlocal contribution (or "correction") to the

```

```

137:     phase mismatch  $\Delta k$ , defined as
138:      $\beta = \frac{\Delta \alpha}{\Delta k - 2\pi/\Lambda}$ 
139:     eta : float
140:     Pump intensity normalized against the threshold intensity,
141:      $\eta = \left(\frac{\pi}{2}\right)^2 \frac{I_{\text{pump}}}{I_{\text{th}}}$ 
142:
143:     Returns
144:     -----
145:     ggp : float
146:         The LCP (+) signal gain  $G_+$ .
147:     ggm : float
148:         The RCP (-) signal gain  $G_-$ .
149:     """
150:     d2 = np.square(delta)
151:     b2p = np.square(1.0+beta)
152:     b2m = np.square(1.0-beta)
153:     tp = np.sqrt(np.multiply(d2,b2p)+eta)
154:     tm = np.sqrt(np.multiply(d2,b2m)+eta)
155:     cos2p = np.square(np.cos(tp))
156:     cos2m = np.square(np.cos(tm))
157:     sin2p = np.square(np.sin(tp))
158:     sin2m = np.square(np.sin(tm))
159:     sp = np.multiply(d2,b2p)
160:     sm = np.multiply(d2,b2m)
161:     ggp = np.divide(tp, np.multiply(tp, cos2p)+np.multiply(sp, sin2p))
162:     ggm = np.divide(tm, np.multiply(tm, cos2m)+np.multiply(sm, sin2m))
163:     if verbose:
164:         print("LCP: max(G+)=%1.4f, min(G+)=%1.4f"%(np.max(ggp), np.min(ggp)))
165:         print("RCP: max(G-)=%1.4f, min(G-)=%1.4f"%(np.max(ggm), np.min(ggm)))
166:     return ggp, ggm
167:
168: def afsignal(zn, delta, beta, eta, verbose=True):
169:     """
170:     Compute the forward traveling signal envelopes  $A^{f+}_{\omega_2}(z)$ 
171:     (LCP) and  $A^{f-}_{\omega_2}(z)$  (RCP) as function of normalized
172:     parameters and normalized distance  $z/L$ .
173:
174:     Parameters
175:     -----
176:     zn : float, np.array
177:         The normalized spatial coordinate  $z/L$ .
178:     delta : float
179:         The normalized electric dipolar quasi phase mismatch remainder
180:          $\Delta = \frac{\Delta k L}{2} - \frac{2\pi L}{2\Lambda}$  against
181:         the quasi phase matching period.
182:     beta : float
183:         The normalized nonlocal contribution (or "correction") to the
184:         phase mismatch  $\Delta k$ , defined as
185:          $\beta = \frac{\Delta \alpha}{\Delta k - 2\pi/\Lambda}$ 
186:     eta : float
187:         Pump intensity normalized against the threshold intensity,
188:          $\eta = \left(\frac{\pi}{2}\right)^2 \frac{I_{\text{pump}}}{I_{\text{th}}}$ 
189:
190:     Returns
191:     -----
192:     asp : float, np.array
193:         The envelope of the LCP (+) forward traveling signal
194:          $A^{f+}_{\omega_2}(z)$ .
195:     asm : TYPE
196:         The envelope of the RCP (-) forward traveling signal
197:          $A^{f-}_{\omega_2}(z)$ .
198:     """
199:     kappalp, kappalm = kappal(delta, beta)
200:     blp, blm = bl(delta, beta, eta)
201:     kbp, kbm = np.divide(kappalp,blp), np.divide(kappalm,blm)
202:     czp, szp = np.cos(blp*zn), np.sin(blp*zn)
203:     czm, szm = np.cos(blm*zn), np.sin(blm*zn)
204:     clp, slp = np.cos(blp), np.sin(blp)

```

```

205:     clm, slm = np.cos(blm), np.sin(blm)
206:     expkzp, expkzm = np.exp(1j*kappalp*zn), np.exp(1j*kappalm*zn)
207:     if verbose:
208:         print("b+L = %f"%(blp))
209:         print("b-L = %f"%(blm))
210:         print("kappa+L/2 = %f"%(kappalp))
211:         print("kappa-L/2 = %f"%(kappalm))
212:
213:     """ Compute LCP component  $A^{\{f+\}}_{\{\omega_2\}}(z)/A^{\{f+\}}_{\{\omega_2\}}(0)$  """
214:     asp = np.divide(slp-1j*kbp*clp, clp+1j*kbp*slp)
215:     asp = czp + np.multiply(asp, szp)
216:     asp = np.multiply(asp, expkzp)
217:
218:     """ Compute RCP component  $A^{\{f-\}}_{\{\omega_2\}}(z)/A^{\{f-\}}_{\{\omega_2\}}(0)$  """
219:     asm = np.divide(slm-1j*kbm*clm, clm+1j*kbm*slm)
220:     asm = czm + np.multiply(asm, szm)
221:     asm = np.multiply(asm, expkzm)
222:
223:     return asp, asm
224:
225: def makegraph_01(bw=True, printtitle=False, plots1s2=False):
226:     """
227:     Internally used parameters
228:     -----
229:     zn : float, np.array
230:         Normalized spatial coordinate z/L, typically in range 0 ≤ z/L ≤ 1.
231:     delta : float
232:         The normalized electric dipolar quasi phase mismatch remainder
233:          $\delta = \{\frac{\Delta k L}{2}\} - \{\frac{2\pi L}{2\Lambda}\}$  against
234:         the quasi phase matching period.
235:     beta : float
236:         The normalized nonlocal contribution (or "correction") to the
237:         phase mismatch  $\Delta k$ , defined as
238:          $\beta = \{\frac{\Delta \alpha}{\Delta k - 2\pi/\Lambda}\}$ 
239:     eta : float
240:         Pump intensity normalized against the threshold intensity,
241:          $\eta = (\{\pi/2\})^2 \{I_{\text{pump}}/I_{\text{th}}\}$ 
242:     """
243:     print("==== Generating image set 01 (graph-01-*) =====")
244:     zn = np.linspace(0.0, 1.0, 1024)
245:     deltamin, deltamax, numdelta = 0.5, 1.5, 3
246:     betamin, betamax, numbeta = -2.0, 2.0, 5
247:     deltarange = np.linspace(deltamin, deltamax, numdelta)
248:     betarange = np.linspace(betamax, betamin, numbeta)
249:     eta = 2.0
250:
251:     """
252:     Define colors and linestyles for the five curves to be mapped for each
253:     value of the chiral coefficient beta.
254:     """
255:     dashdotdotted = (0, (5, 1, 1, 1, 1, 1)) # Custom 'â\200\224Â•Â•â\200\224Â•Â•â\200\224Â•Â•â\200\224'
256:     colors=['xkcd:red', 'xkcd:green', 'xkcd:azure', 'xkcd:tan', 'xkcd:teal']
257:     linestyles=['dashed', 'dashdot', 'dotted', dashdotdotted, 'solid']
258:
259:     for delta in deltarange:
260:         if plots1s2: # Plot all of S0, S1, S2 and S3
261:             fig, ax = plt.subplots(3, figsize=(5.4, 5.0))
262:         else: # Plot only S0 and S3
263:             fig, ax = plt.subplots(2, figsize=(5.4, 3.8))
264:         for k, beta in enumerate(betarange):
265:             """
266:             For each value for the electric dipolar phase mismatch against
267:             the nominal QPM period, generate a separate graph of the Stokes
268:             parameters S0, S1, S2 and S3.
269:             """
270:             asp, asm = afsignal(zn, delta, beta, eta)
271:             s0, s1, s2, s3 = stokesparams(asp/np.sqrt(2.0), asm/np.sqrt(2.0))

```

```

272:     labeltext='$\\beta=1.1f$'%(beta)
273:     if bw:
274:         color = 'k'
275:         linestyle = linestyles[k]
276:     else:
277:         color = colors[k]
278:         linestyle = 'solid'
279:     ax[0].semilogy(zn, s0, color=color, linestyle=linestyle, label=labeltext
)
280:     if plots1s2:
281:         ax[1].plot(zn, s1, color=color, linestyle=linestyle, label=labeltext
)
282:         ax[1].plot(zn, s2, color=color, linestyle=linestyle)
283:         ax[2].plot(zn, s3, color=color, linestyle=linestyle, label=labeltext
)
284:     else:
285:         ax[1].plot(zn, s3, color=color, linestyle=linestyle, label=labeltext
)
286:
287:     """
288:     Save each generated trajectory of (S0,S1,S2,S3) as a separate data
289:     set for postprocessing by, for example, the Poincaré program.
290:     """
291:     filename = "data/data-01-delta-%1.2f-beta-%1.2f.dat"%(delta,beta)
292:     saveStokesParameters(zn, s0, s1, s2, s3, filename)
293:
294:     ax[1].yaxis.set_minor_locator(AutoMinorLocator(5))
295:     if plots1s2:
296:         ax[2].yaxis.set_minor_locator(AutoMinorLocator(5))
297:
298:     for j in range(3 if plots1s2 else 2):
299:         ax[j].autoscale(enable=True, axis='x', tight=True)
300:         ax[j].legend(loc='upper right', fontsize=9, handlelength=4)
301:         ax[j].grid(visible=True, which='major', axis='both')
302:         ax[j].tick_params(which="both", top=True, right=True,
303:                           labeltop=False, bottom=True, labelbottom=True,
304:                           direction="in")
305:
306:     if printtitle:
307:         ax[0].set_title("Dipolar QPM mismatch  $(\\Delta k - 2\\pi/\\Lambda)L/2$ "
308:                        "=%1.1f"%(delta))
309:     ax[0].set_ylabel("$S_0(z)/S_0(0)$")
310:     if plots1s2:
311:         ax[1].set_ylim(-1.05,1.05)
312:         ax[1].set_ylabel("$\\vbox{$\\hbox{$S_1(z)/S_0(z)$,$}$}"
313:                          "\\hbox{$S_2(z)/S_0(z)$}$)")
314:         ax[2].set_ylim(-1.05,1.05)
315:         ax[2].set_ylabel("$S_3(z)/S_0(z)$")
316:         ax[2].set_xlabel("$z/L$")
317:     else:
318:         ax[1].set_ylim(-1.05,1.05)
319:         ax[1].set_ylabel("$S_3(z)/S_0(z)$")
320:         ax[1].set_xlabel("$z/L$")
321:     kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
322:     basename = "graphs/graph-01-delta-%1.2f"%delta
323:     if bw:
324:         basename += "-bw"
325:     if plots1s2:
326:         basename += "-all"
327:     for fmt in ['eps', 'svg', 'png']:
328:         fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
329:     return
330:
331: def makegraph_02():
332:     """
333:     Graph of the single-pass gain of the signal as function of normalized
334:     chiral phase mismatch and at a set of pump intensity levels.
335:     Parameters:

```

```

336:         beta: Normalized chiral mismatch,  $\beta = \Delta \alpha L / 2$ 
337:         eta: Normalized pump intensity,  $\eta = I_{\text{pump}} / I_{\text{th}}$ 
338:     """
339:     print("==== Generating image set 02 (graph-02-*) =====")
340:     beta = np.linspace(-7.0, 7.0, 1000)
341:     fig, ax = plt.subplots(figsize=(5.4, 4.0))
342:     for eta in np.linspace(2.0, 5.0, 4):
343:         delta = np.sqrt(np.square(beta) + eta)
344:         gg = np.divide(1.0, (np.square(np.cos(delta))
345:                               + ((beta/delta)**2)*np.square(np.sin(delta))))
346:         ax.semilogy(beta, gg, label=f' $\eta = \eta_{1.1f} \% (\eta)$ ')
347:
348:     ax.autoscale(enable=True, axis='x', tight=True)
349:     ax.legend(loc='upper right', fontsize=11)
350:     ax.tick_params(axis="both", direction="in")
351:     ax.grid(visible=True, which='both', axis='both')
352:     ax.set_xlabel(" $\Delta \alpha L / 2$ ")
353:     ax.set_ylabel(" $G_{\text{pm}} = |A^{\text{pm}}_{\text{rm s}}(L) / A^{\text{pm}}_{\text{rm s}}(0)|^2$ ")
354:
355:     kwargs={'bbox_inches': 'tight', 'pad_inches': 0.0}
356:     basename = "graphs/graph-02"
357:     for fmt in ['eps', 'svg', 'png']:
358:         fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
359:
360:     return
361:
362: def makegraph_03():
363:     """
364:     Graph of the single-pass gain of the signal as function of normalized
365:     electric dipolar and chiral phase mismatch.
366:     Parameters:
367:         beta : Normalized chiral term  $\Delta \alpha / (\Delta k - 2\pi / \Lambda)$ 
368:         delta : Normalized dipolar mismatch  $(\Delta k - 2\pi / \Lambda) L / 2$ 
369:         eta : Normalized pump intensity,  $\eta = I_{\text{pump}} / I_{\text{th}}$ 
370:     """
371:     print("==== Generating image set 03 (graph-03-*) =====")
372:     betamax = 2.0 # This is intended to be funny
373:     deltamax = betamax
374:     beta = np.linspace(-betamax, betamax, 2048)
375:     delta = np.linspace(-deltamax, deltamax, 2048)
376:     betag, deltag = np.meshgrid(beta, delta, indexing='xy')
377:     eta = 5.0
378:     ggp, ggm = gain(deltag, betag, eta)
379:
380:     dx = (beta[1]-beta[0])/2.
381:     dy = (delta[1]-delta[0])/2.
382:     extent = [beta[0]-dx, beta[-1]+dx, delta[0]-dy, delta[-1]+dy]
383:     clevels = np.linspace(-4, 4, 9)
384:
385:     ggp_db = 10*np.log10(ggp)
386:     ggm_db = 10*np.log10(ggm)
387:
388:     """
389:     Map the gain  $G_+$  and  $G_-$  as a surface, being a function of the normalized
390:     electric dipolar phase mismatch and its chiral contribution.
391:     """
392:     fig, ax = plt.subplots(figsize=(5.4, 5.4), subplot_kw={"projection": "3d"})
393:     ax.plot_surface(betag, deltag, ggp_db, vmin=2.2*ggp_db.min(), cmap=cm.Oranges)
394:     ax.plot_surface(betag, deltag, ggm_db, vmin=2.2*ggp_db.min(), cmap=cm.Blues)
395:     ax.contour(betag, deltag, ggp_db, [-4, -3, -2, -1, 0, 1, 2, 3, 4], zdir='z',
396:                offset=np.min(ggp_db)-1, cmap=cm.Oranges)
397:     ax.contour(betag, deltag, ggm_db, [-4, -3, -2, -1, 0, 1, 2, 3, 4], zdir='z',
398:                offset=np.min(ggm_db)-1, cmap=cm.Blues)
399:     ax.autoscale(enable=True, axis='x', tight=True)
400:     ax.set_xlabel(" $\beta = \Delta \alpha / (\Delta k - 2\pi / \Lambda)$ ")
401:     ax.set_ylabel(" $\delta = (\Delta k - 2\pi / \Lambda) L / 2$ ")
402:     ax.set_zlabel(" $G_+$ ,  $G_-$  (gain)")
403:     kwargs={'bbox_inches': 'tight', 'pad_inches': 0.0}

```

```

404:     basename = "graphs/graph-03-surf"
405:     for fmt in ['eps', 'svg', 'png']:
406:         fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
407:
408:     """
409:     Map the LCP gain  $G_+$  as a surface, being a function of the normalized
410:     electric dipolar phase mismatch and its chiral contribution.
411:     """
412:     fig, ax = plt.subplots(figsize=(5.4,5.4))
413:     pos = ax.imshow(ggp_db, extent=extent, cmap=cm.Oranges)
414:     divider = make_axes_locatable(plt.gca())
415:     cax = divider.append_axes("right", "5%", pad="3%")
416:     plt.colorbar(pos, cax=cax)
417:     plt.tight_layout()
418:     cs = ax.contour(betag, deltag, ggp_db, clevels, colors='k')
419:     ax.clabel(cs, cs.levels, fontsize=12)
420:     ax.autoscale(enable=True, axis='xy', tight=True)
421:     ax.set_xlabel(" $\beta = \Delta\alpha / (\Delta k - 2\pi/\Lambda)$ ")
422:     ax.set_ylabel(" $\delta = (\Delta k - 2\pi/\Lambda)L/2$ ")
423:     ax.set_title("LCP gain  $G_+$ ")
424:     kwargs={'bbox_inches': 'tight', 'pad_inches': 0.0}
425:     basename = "graphs/graph-03-gplus-image"
426:     for fmt in ['eps', 'svg', 'png']:
427:         fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
428:
429:     """
430:     Map the RCP gain  $G_-$  as a surface, being a function of the normalized
431:     electric dipolar phase mismatch and its chiral contribution.
432:     """
433:     fig, ax = plt.subplots(figsize=(5.4,5.4))
434:     pos = ax.imshow(ggm_db, extent=extent, cmap=cm.Blues)
435:     divider = make_axes_locatable(plt.gca())
436:     cax = divider.append_axes("right", "5%", pad="3%")
437:     plt.colorbar(pos, cax=cax)
438:     plt.tight_layout()
439:     cs = ax.contour(betag, deltag, ggm_db, clevels, colors='k')
440:     ax.clabel(cs, cs.levels, fontsize=12)
441:     ax.autoscale(enable=True, axis='xy', tight=True)
442:     ax.set_xlabel(" $\beta = \Delta\alpha / (\Delta k - 2\pi/\Lambda)$ ")
443:     ax.set_ylabel(" $\delta = (\Delta k - 2\pi/\Lambda)L/2$ ")
444:     ax.set_title("RCP gain  $G_-$ ")
445:     kwargs={'bbox_inches': 'tight', 'pad_inches': 0.0}
446:     basename = "graphs/graph-03-gminus-image"
447:     for fmt in ['eps', 'svg', 'png']:
448:         fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
449:
450:     return
451:
452: def makegraph_04():
453:     """
454:     Graph of the single-pass Stokes parameter gain  $S_0(L)/S_0(0)$  of the signal
455:     as function of normalized electric dipolar and chiral phase mismatch.
456:     Parameters:
457:         beta : Normalized chiral term  $\Delta\alpha / (\Delta k - 2\pi/\Lambda)$ 
458:         delta : Normalized dipolar mismatch  $(\Delta k - 2\pi/\Lambda)L/2$ 
459:         eta : Normalized pump intensity,  $\eta = I_{\text{pump}}/I_{\text{th}}$ 
460:     """
461:     print("==== Generating image set 04 (graph-04-*) =====")
462:     betamax = 2.0 # This is intended to be funny
463:     deltamax = betamax
464:     beta = np.linspace(-betamax, betamax, 2048)
465:     delta = np.linspace(-deltamax, deltamax, 2048)
466:     betag, deltag = np.meshgrid(beta, delta, indexing='xy')
467:     dx = (beta[1]-beta[0])/2.
468:     dy = (delta[1]-delta[0])/2.
469:     extent = [beta[0]-dx, beta[-1]+dx, delta[0]-dy, delta[-1]+dy]
470:
471:     eta = 5.0

```

```

472:     ggp, ggm = gain(deltag, betag, eta)
473:     s0 = (ggp+ggm)/2.0
474:     clevels = np.linspace(0.5,2.5,5)
475:
476:     fig, ax = plt.subplots(figsize=(5.4,5.4),subplot_kw={"projection": "3d"})
477:     ax.plot_surface(betag, deltag, s0, cmap=cm.Blues)
478:     ax.contour(betag, deltag, s0, clevels, zdir='z', offset=np.min(s0),
479:               cmap=cm.Blues)
480:     ax.autoscale(enable=True, axis='xy', tight=True)
481:     ax.set_xlabel("$\\beta=\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
482:     ax.set_ylabel("$\\delta=(\\Delta k-2\\pi/\\Lambda)L/2$")
483:     ax.set_zlabel("$S_0(L)/S_0(0)$ (gain)")
484:
485:     kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
486:     basename = "graphs/graph-04-s0-surface"
487:     for fmt in ['eps','svg','png']:
488:         fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
489:
490:     """
491:     Map  $S_0(L)/S_0(0)$ , being a function of the normalized electric dipolar
492:     phase mismatch and its chiral contribution, as an image with overlaid
493:     contours.
494:     """
495:     fig, ax = plt.subplots(figsize=(5.4,5.4))
496:     pos = ax.imshow(s0, extent=extent, cmap=cm.Blues)
497:     divider = make_axes_locatable(plt.gca())
498:     cax = divider.append_axes("right", "5%", pad="3%")
499:     plt.colorbar(pos, cax=cax)
500:     plt.tight_layout()
501:     cs = ax.contour(betag, deltag, s0, clevels, colors='k')
502:     ax.clabel(cs, cs.levels, fontsize=12)
503:     ax.autoscale(enable=True, axis='xy', tight=True)
504:     ax.set_xlabel("$\\beta=\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
505:     ax.set_ylabel("$\\delta=(\\Delta k-2\\pi/\\Lambda)L/2$")
506:     ax.set_title("Intensity gain  $S_0(L)/S_0(0)$ , "
507:                 "$I_{\\rm pump}/I_{\\rm th}=%1.1f"%eta))
508:     kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
509:     basename = "graphs/graph-04-s0-%1.2f-image"%eta
510:     for fmt in ['eps','svg','png']:
511:         fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
512:
513:     """
514:     Identical to previous plot, but purely with contours in black, no other
515:     image data or colors.
516:     """
517:     fig, ax = plt.subplots(figsize=(5.4,5.4))
518:     cs = ax.contour(betag, deltag, s0, clevels, zdir='z', offset=np.min(s0),
519:                   colors='k')
520:     ax.clabel(cs, cs.levels, fontsize=12)
521:     ax.autoscale(enable=True, axis='xy', tight=True)
522:     ax.set_xlabel("$\\beta=\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
523:     ax.set_ylabel("$\\delta=(\\Delta k-2\\pi/\\Lambda)L/2$")
524:     kwargs={'bbox_inches':'tight', 'pad_inches':0.0}
525:     basename = "graphs/graph-04-s0-contour-%1.2f-black"%eta
526:     for fmt in ['eps','svg','png']:
527:         fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
528:
529:     return
530:
531: def makegraph_05(printtitle=False):
532:     """
533:     Graph of the single-pass Stokes parameter  $S_3(L)/S_0(0)$  of the signal
534:     as function of normalized electric dipolar and chiral phase mismatch.
535:     Parameters:
536:         beta : Normalized chiral term  $\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$ 
537:         delta : Normalized dipolar mismatch  $(\\Delta k-2\\pi/\\Lambda)L/2$ 
538:         eta : Normalized pump intensity,  $\\eta=I_{\\rm pump}/I_{\\rm th}$ 
539:     """

```



```

540:     print("=====Generating image set 05 (graph-05-*) =====")
541:     betamax = 2.0 # This is intended to be funny
542:     deltamax = betamax
543:     beta = np.linspace(-betamax, betamax, 2048)
544:     delta = np.linspace(-deltamax, deltamax, 2048)
545:     betag, deltag = np.meshgrid(beta, delta, indexing='xy')
546:     dx = (beta[1]-beta[0])/2.
547:     dy = (delta[1]-delta[0])/2.
548:     extent = [beta[0]-dx, beta[-1]+dx, delta[0]-dy, delta[-1]+dy]
549:
550:     for eta in [2.0, 3.0, 4.0, 5.0]:
551:         ggp, ggm = gain(deltag, betag, eta)
552:         s3 = (ggp-ggm)/(ggp+ggm)
553:         clevels = np.linspace(-1.0,1.0,9)
554:
555:         """
556:         Map S3/S0, being a function of the normalized electric dipolar phase
557:         mismatch and its chiral contribution, as a surface plot with contours
558:         underneath.
559:         """
560:         fig, ax = plt.subplots(figsize=(5.4,5.4), subplot_kw={"projection": "3d"})
561:         ax.plot_surface(betag, deltag, s3, cmap=cm.Blues)
562:         ax.contour(betag, deltag, s3, clevels, zdir='z', offset=np.min(s3),
563:                   cmap=cm.Blues)
564:         ax.autoscale(enable=True, axis='xy', tight=True)
565:         ax.set_xlabel(" $\Delta\alpha/(\Delta k-2\pi/\Lambda)$ ")
566:         ax.set_ylabel(" $(\Delta k-2\pi/\Lambda)L/2$ ")
567:         ax.set_zlabel("S3(L)/S0(L)")
568:
569:         kwargs={'bbox_inches': 'tight', 'pad_inches': 0.0}
570:         basename = "graphs/graph-05-s3-%1.2f-surface"%eta
571:         for fmt in ['eps', 'svg', 'png']:
572:             fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
573:
574:         """
575:         Map S3/S0, being a function of the normalized electric dipolar phase
576:         mismatch and its chiral contribution, as plain black contours without
577:         any image of color. Useful for plain printing in B/W.
578:         """
579:         fig, ax = plt.subplots(figsize=(5.4,5.4))
580:         cs = ax.contour(betag, deltag, s3, clevels, zdir='z', offset=np.min(s3),
581:                       vmin=0.1*s3.min(), colors='k')
582:         ax.clabel(cs, cs.levels, fontsize=12)
583:         ax.autoscale(enable=True, axis='xy', tight=True)
584:         ax.set_xlabel(" $\Delta\alpha/(\Delta k-2\pi/\Lambda)$ ")
585:         ax.set_ylabel(" $(\Delta k-2\pi/\Lambda)L/2$ ")
586:
587:         kwargs={'bbox_inches': 'tight', 'pad_inches': 0.0}
588:         basename = "graphs/graph-05-s3-%1.2f-bw"%eta
589:         for fmt in ['eps', 'svg', 'png']:
590:             fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
591:
592:         """
593:         Map S3/S0, being a function of the normalized electric dipolar phase
594:         mismatch and its chiral contribution, as an image with overlaid black
595:         contours.
596:         """
597:         cmp = LinearSegmentedColormap.from_list("",
598:           ["xkcd:azure", "xkcd:white", "xkcd:orangered"])
599:         fig, ax = plt.subplots(figsize=(5.4,5.4))
600:         pos = ax.imshow(s3, extent=extent, cmap=cmp, vmin=-1.0, vmax=1.0)
601:         divider = make_axes_locatable(plt.gca())
602:         cax = divider.append_axes("right", "5%", pad="3%")
603:         plt.colorbar(pos, cax=cax)
604:         plt.tight_layout()
605:         cs = ax.contour(betag, deltag, s3, clevels, colors='k')
606:         ax.clabel(cs, cs.levels, fontsize=12)
607:         ax.autoscale(enable=True, axis='xy', tight=True)

```

```
608:         ax.set_xlabel("$\\Delta\\alpha/(\\Delta k-2\\pi/\\Lambda)$")
609:         ax.set_ylabel("$\\Delta k-2\\pi/\\Lambda)L/2$")
610:         if printtitle:
611:             ax.set_title("Ellipticity $S_3(L)/S_0(L)$, "
612:                          "$I_{\\rm pump}/I_{\\rm th}=%1.1f"% (eta))
613:         kwargs={'bbox_inches': 'tight', 'pad_inches': 0.0}
614:         basename = "graphs/graph-05-s3-%1.2f-image"%eta
615:         for fmt in ['eps', 'svg', 'png']:
616:             fig.savefig(basename+'.'+fmt, format=fmt, **kwargs)
617:
618:     return
619:
620: def main() -> None:
621:     for bw in [True, False]: # Generate image set 01 (graph-01-*.eps|png|svg)
622:         for plots1s2 in [True, False]:
623:             makegraph_01(bw=bw, plots1s2=plots1s2)
624:             makegraph_02() # Generate image set 02 (graph-02-*.eps|png|svg)
625:             makegraph_03() # Generate image set 03 (graph-03-*.eps|png|svg)
626:             makegraph_04() # Generate image set 04 (graph-04-*.eps|png|svg)
627:             makegraph_05() # Generate image set 05 (graph-05-*.eps|png|svg)
628:     return
629:
630: if __name__ == "__main__":
631:     main()
```