Facultad de Ciencias

Simulación de peines de frecuencia óptica generados por láseres de semiconductor

Simulation of optical frequecy comb generate by semiconductor lasers

Trabajo de Fin de Grado para acceder al

Grado en Física

Autor
Jaime DÍEZ GONZÁLEZ-PARDO
Director
Dr. Á.A. VALLE GUTIERREZ

24 de agosto de 2019

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Láseres de Semiconductor

1.2. Procesos Estocásticos

1.3. Dinámica No Lineal

1.4. Peines de Frecuencia Óptica

Que son, caracteristicas principales y como se generan,...

1.4.1. *Gain-Switching*

hola que tal todos

1.4.2. Inyección Ópticar

adios a todos

1.4.3. Aplicaciones

1.5. Objetivo del Estudio

Capítulo 2

Modelo Computacional

2.1. RoF

2.2. Código de la Simulación

Capítulo 3

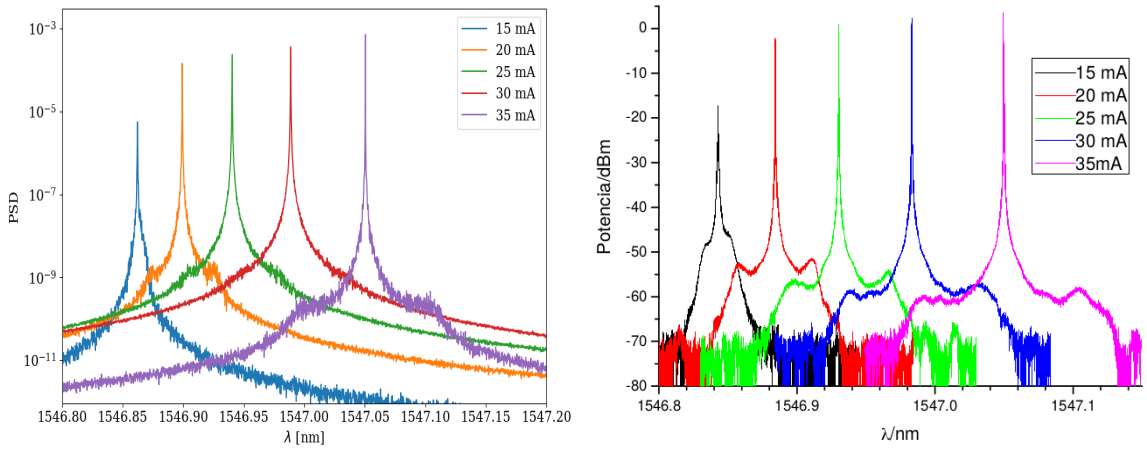
Láser en solitario

Antes de abordar el estudio de la dinámica no lineal del láser de semiconductor de modo discreto se ha realizado la simulación del láser en solitario, sin inyección de luz del láser esclavo ($P_{Iny} = 0$).Se han realizado simulaciones para el láser tanto en corriente continua (CW de sus siglas en inglés) con en Gain-Switching, comparando los resultados con los obtenidos experimentalmente en condiciones similares [1].

3.1. Láser en corriente continua

Para poder realizar las simulaciones en CW se ha trabajado con una corriente igual a la corriente de polarizaci'on ($I(t) = I_{Bias}$), tomando V_{RF}, y as'i la amplitud de modulaci'on.

3.1.1. Espectros de emisión



(a) *Espectros ópticos obtenidos mediante simulación.*

(b) *Espectros ópticos obtenidos experimentalmente.*

Figura 3.1: *Espectros ópticos del DML para diferentes corrientes de polarización I_{Bias} obtenidos mediante simulación (izquierda 3.1a) y esperimentalmente (derecha, 3.1b).*

I_{Bias}	λ_{sim}	λ_{exp}
15	1546.86	1546.84
20	1546.90	1546.88
25	1546.94	1546.93
30	1546.99	1546.98
35	1547.05	1547.05

Tabla 3.1: *Longitud de onda de las lineas de emisión del DML en función de la I_{Bias} obtenidas de la figura 3.1. Se muestran los valores experimentales λ_{exp} obtenidos de la gráfica 3.1b con un error de $\delta\lambda_{exp} = 0.02$, y los valores obtenidos de la simulación de la gráfica 3.1a.*

Las longitudes de onda de la tabla 3.1 muestran una gran concordancia entre los resultados esperimentales y los obtenidos a partir de la simulación del DML.

3.1.2. Transitorio

Pese a que la zona de estudio del DML comienza

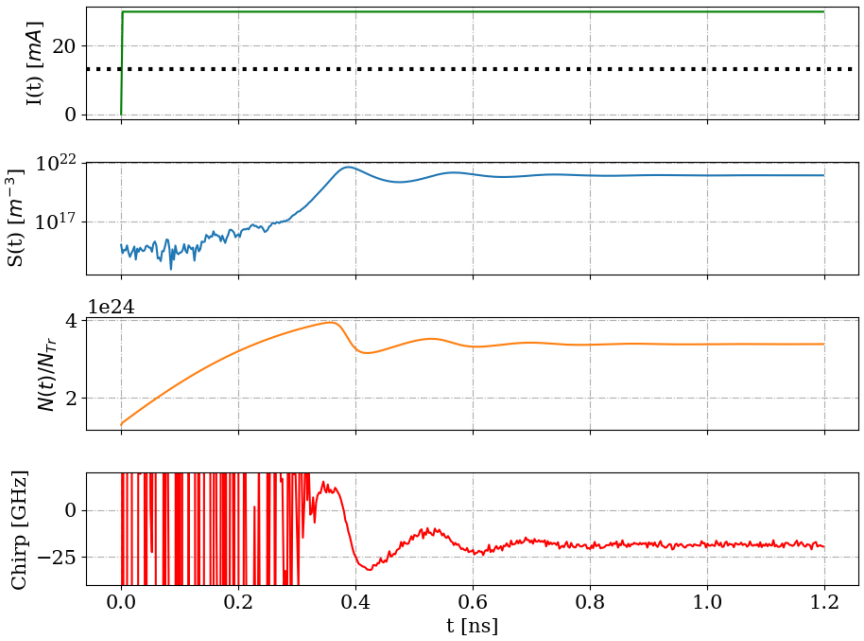


Figura 3.2: *Transitorio*

kkk

3.2. OFC (Gain-Switching)

3.2.1. Efecto de la amplitud de modulación a altas frecuencias

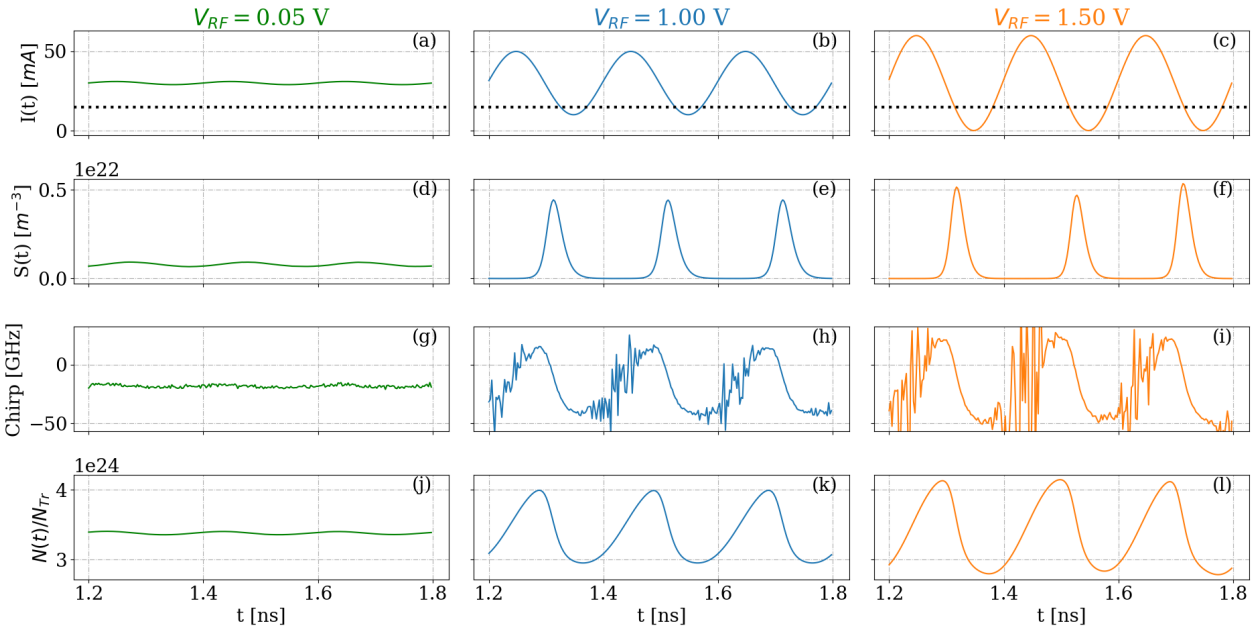


Figura 3.3: *RateEquations*

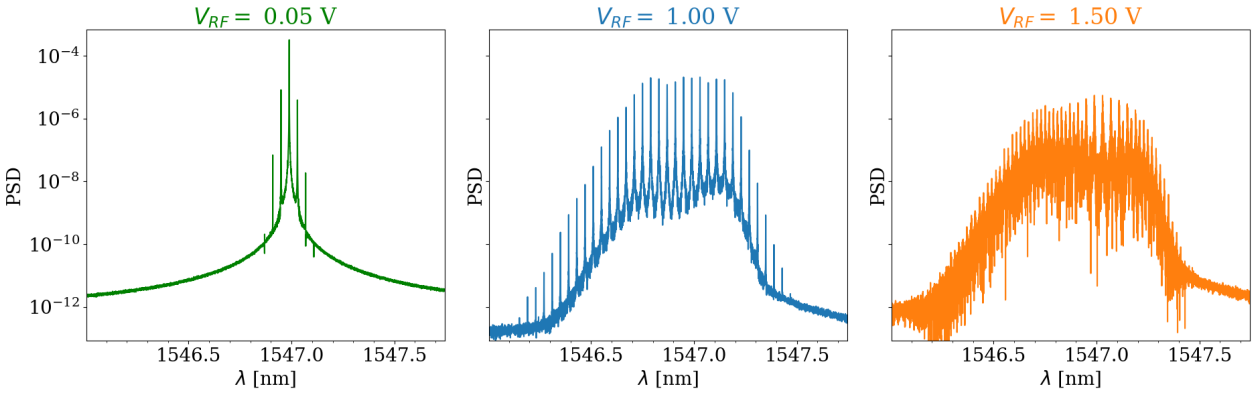


Figura 3.4: *PSD*

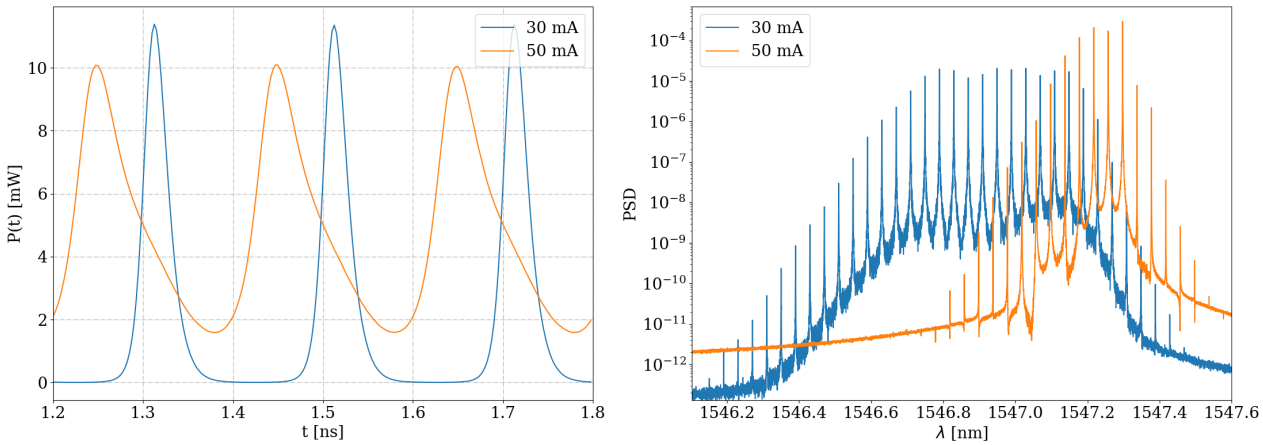


Figura 3.5: *Current*

3.2.2. Efecto de la amplitud de modulación a bajas frecuencias

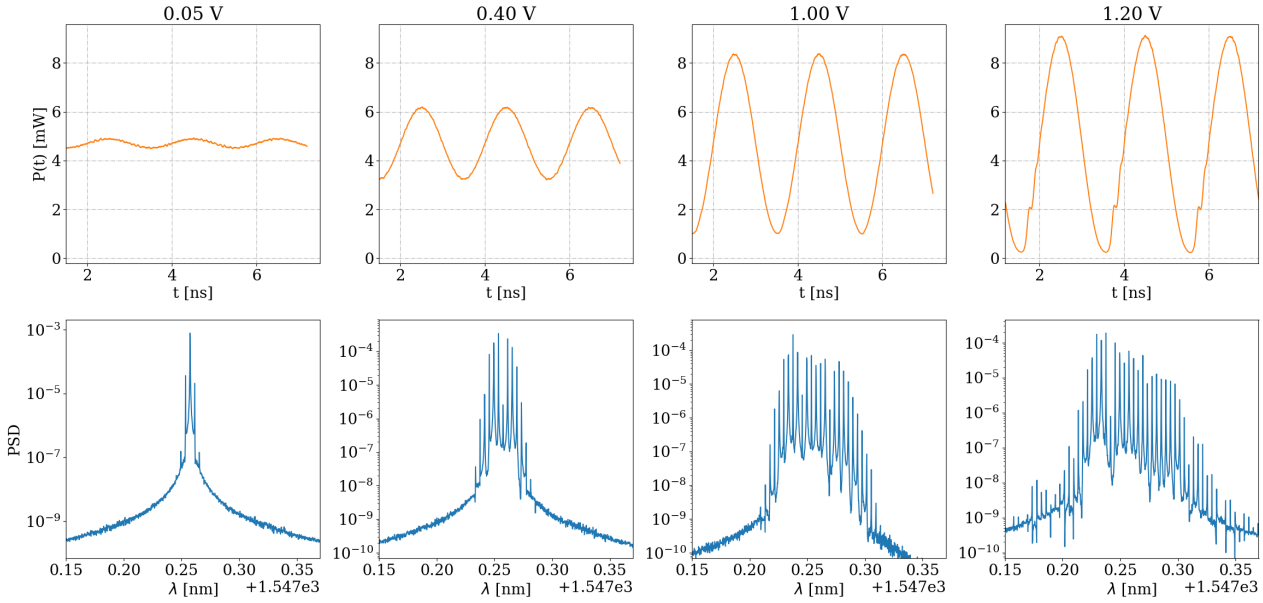


Figura 3.6: *500*

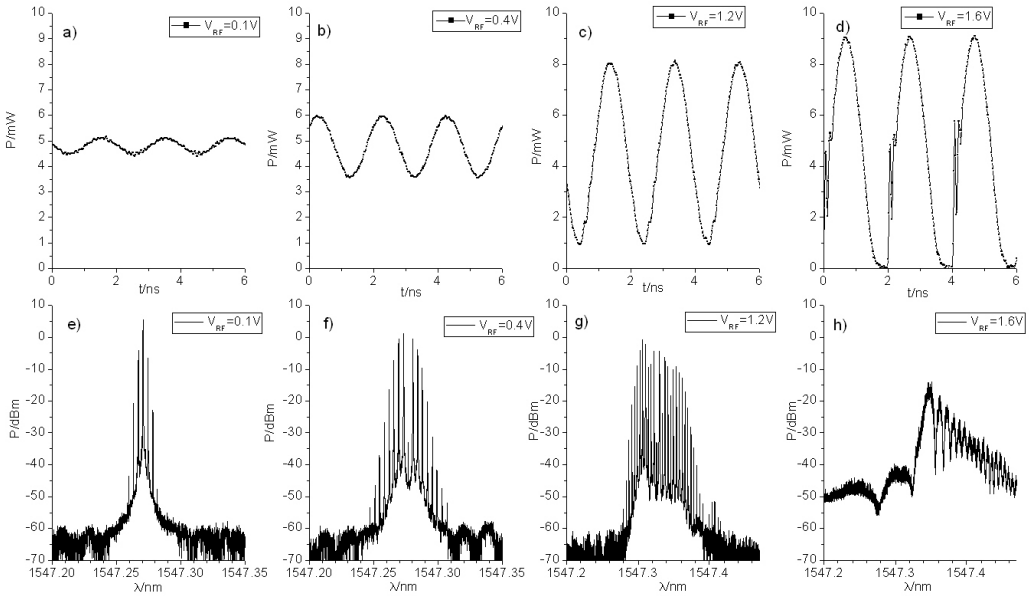


Figura 3.7: *500mhz*

Capítulo 4

Inyeccion de Luz

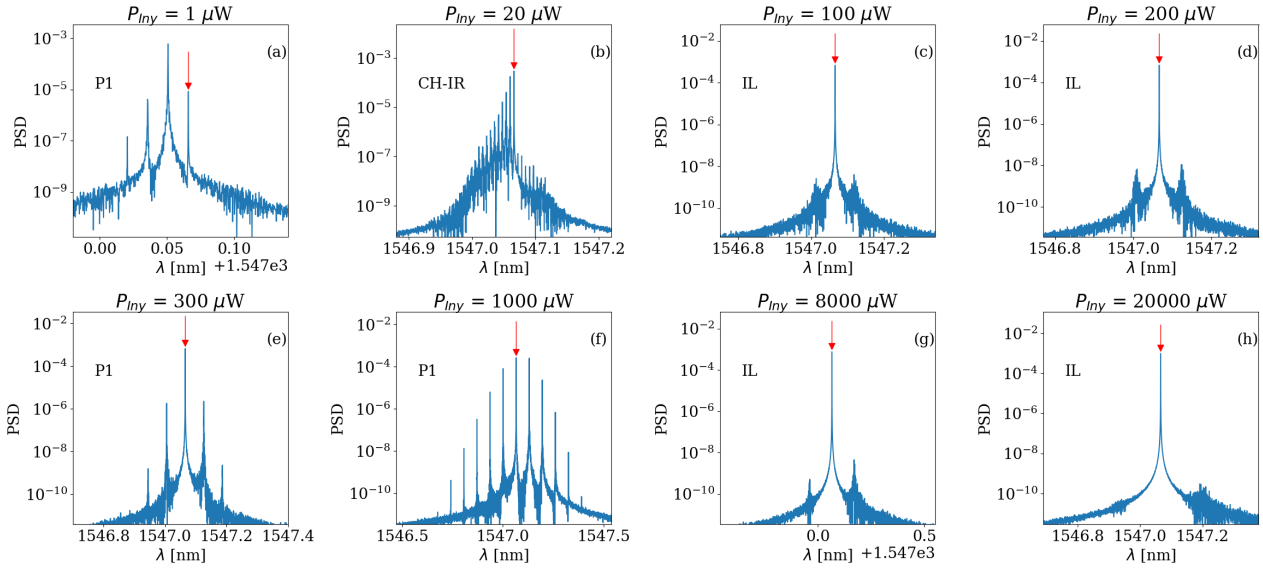


Figura 4.1: *el pie de pagina que le quieras poner a la imagen*

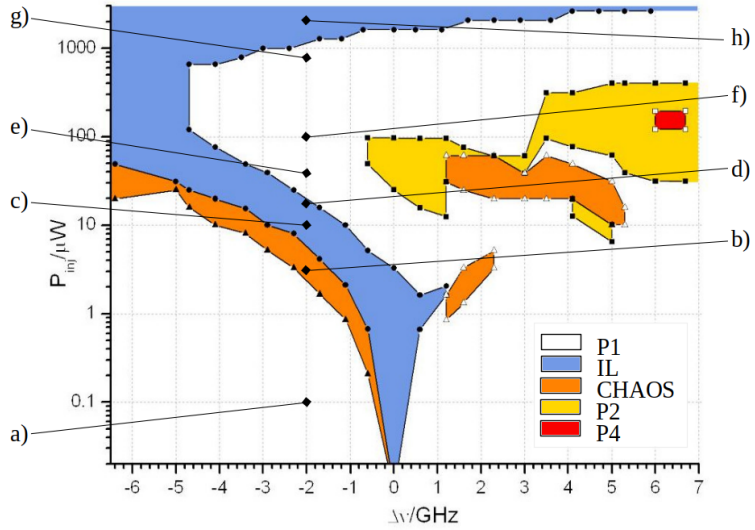


Figura 4.2: *Map*

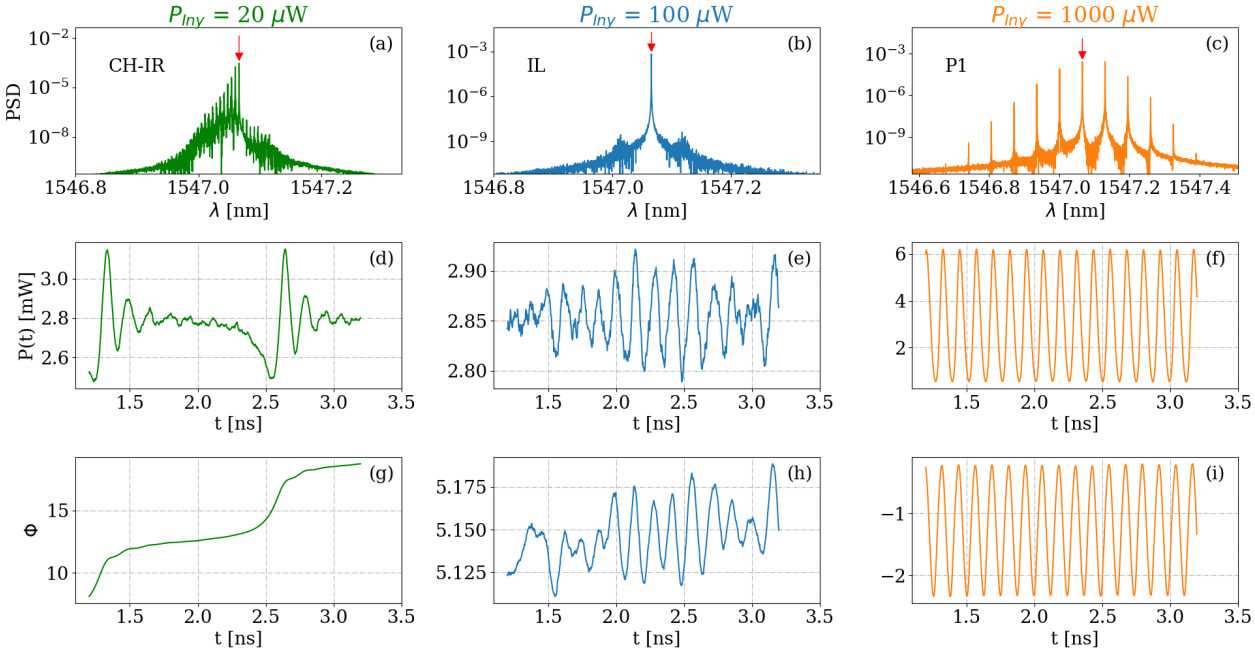


Figura 4.3: *ZoneRtEq*

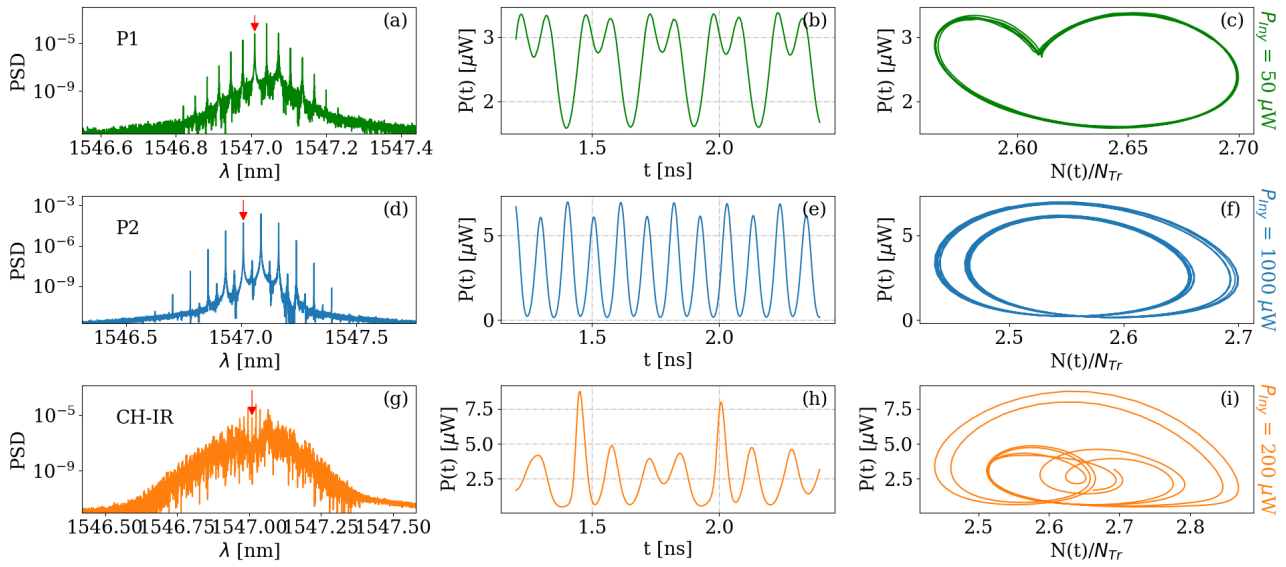


Figura 4.4: *P2zone*

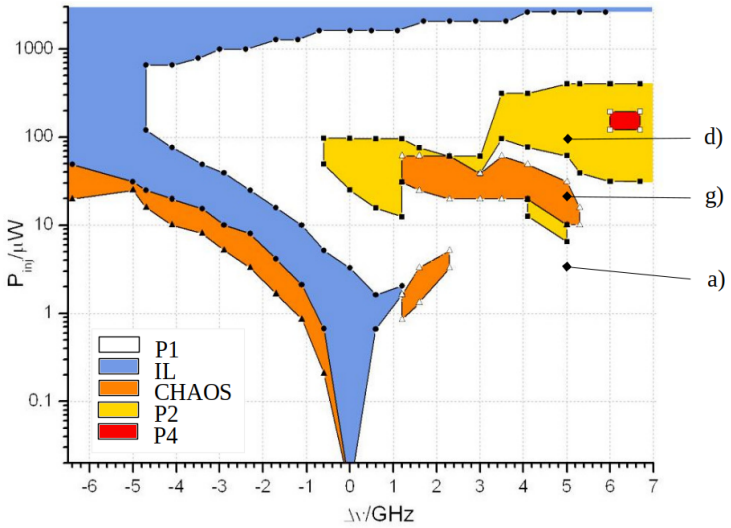


Figura 4.5: *Maps2*

Capítulo 5

Inyeccion de luz en OFC

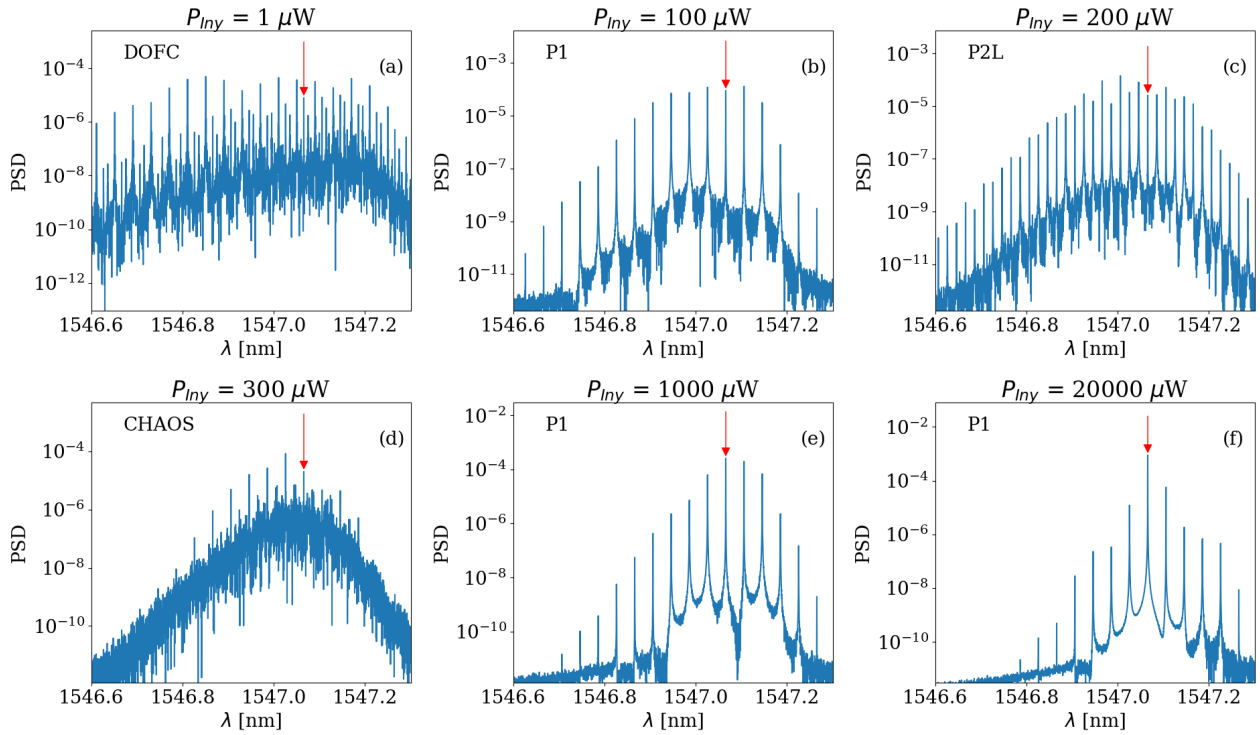


Figura 5.1: *el pie de pagina que le quieras poner a la imagen*

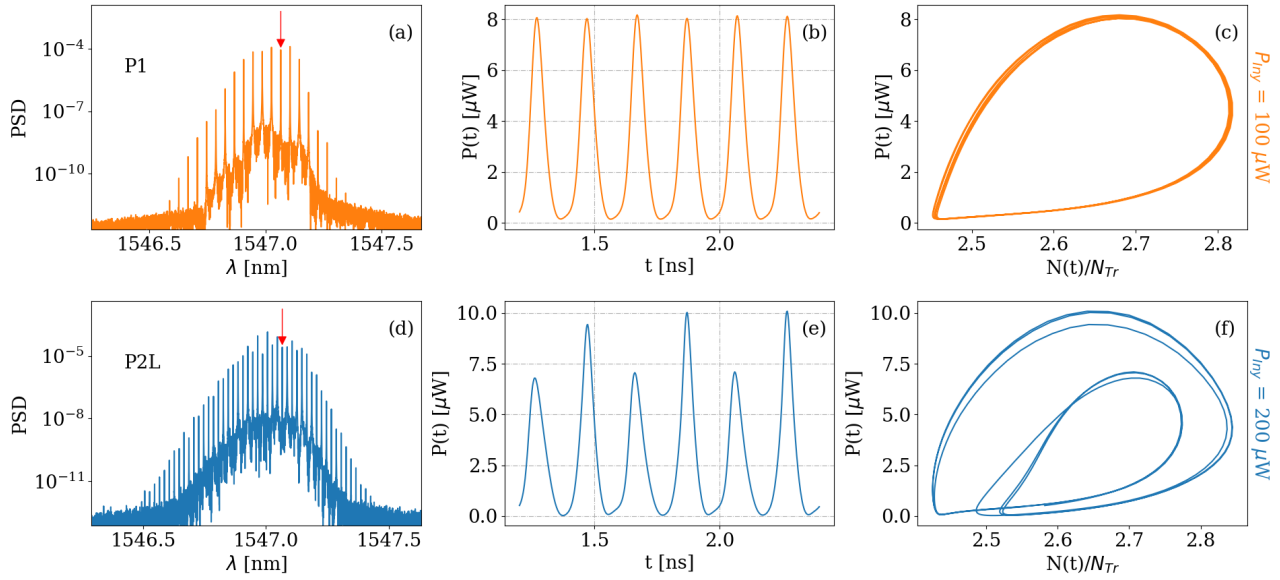


Figura 5.2: *p1-P2*

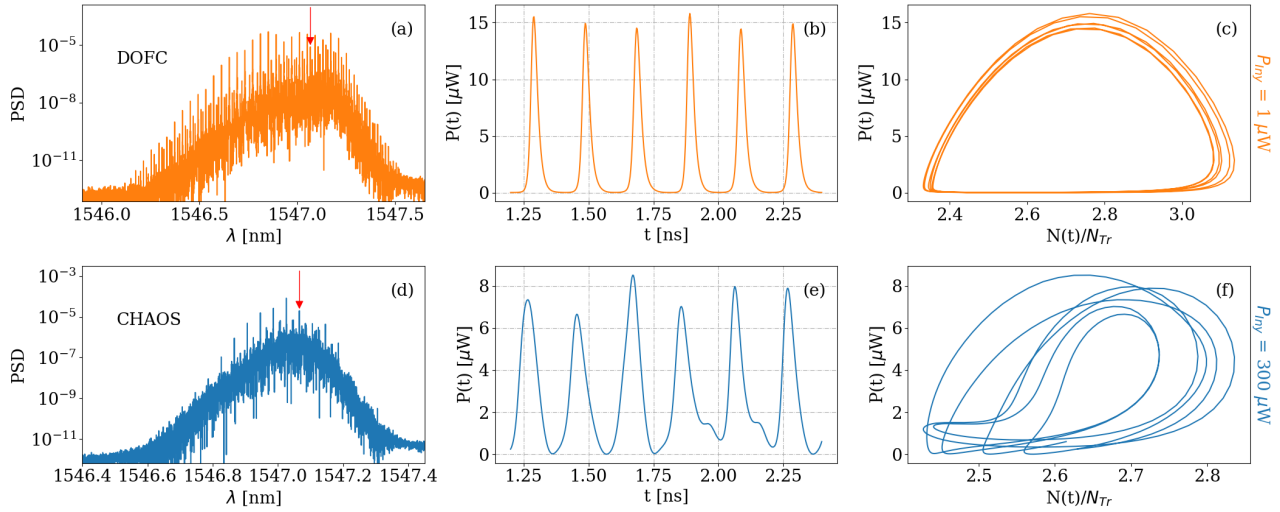


Figura 5.3: *Chaos*

Capítulo 6

Conclusiones

hola a todos

Bibliografía

[1] Diego Chaves Carriles and Ángel Alberto Valle Gutierrez. Peines de frecuencia óptica generados por láseres de semiconductor. *Trabajo Fin de Grado*, 2020.

Apéndice A

Código de la simulación

```
    def allSimulation(self):
        self.setArrays()

        ampInject = kc*np.sqrt(self.sInjct)*tIntev
        opFldInject = np.sqrt(rSL*constP*self.sInjct)*np.exp(1j*np.pi)

        derivPhaseTerm = self.phaseTerm / tIntev
        self.TFavg = 0
        self.TFang = 0

        frecuencyLimits = 1 / (2*delta)
        self.fftFreq = np.linspace(-frecuencyLimits,
                                   frecuencyLimits,
                                  len(self.opField)
                                 )

        deltaFreq = deltaFreqs[self.iBias]
        self.fftFreq += f0 - (deltaFreq/(2.0*np.pi))
        self.fftWL = (c0/self.fftFreq) *10**(9) # Wavelength [nm]

        for win in range(0, self.numWindw):

            #---------------------------------------------------------
            # Gaussian arrays N(0,1) for the Noise
            #---------------------------------------------------------
            X = np.random.normal(0, 1, nTotal)
            Y = np.random.normal(0, 1, nTotal)

            # Initial conditions are defined in order to resolved the SDE
            tempN = nTr
            tempS = float(10**(15))
            tempPhi = 0

            for q in range(0, mTrans):
                for k in range(0, ndelta):

                    index = q*ndelta + k

                    bTN = bTIntv * tempN * tempN
                    invS = 1 / ((1/tempS) + epsilon)
                    sqrtS = np.sqrt(abs(tempS))
                    cosPhi = np.cos(tempPhi)
                    senPhi = np.sin(tempPhi)

```

```
                    tempPhi = (tempPhi + aphvgTGmm*tempN - self.phaseTerm
                               + noisePhi*tempN*Y[index]/sqrtS
                               - (ampInject/sqrtS)*senPhi*cosInject[index]
                               + (ampInject/sqrtS)*cosPhi*senInject[index]
                              )

                    tempS = (tempS + vgTGmm*tempN*invS - vgTGmmN*invS
                             - intTtau*tempS + btGmm*bTN
                             + noiseS*tempN*sqrtS*X[index]
                             + 2*ampInject*sqrtS*cosPhi*cosInject[index]
                             + 2*ampInject*sqrtS*senPhi*senInject[index]
                            )

                    tempN = (tempN + self.currentTerm[index] - aTIntv*tempN
                             - bTN - cTIntv*tempN**3 - vgT*tempN*invS + vgtN*invS
                            )

                self.I[q] = self.currentTerm[index] *10**(12) / eVinv
                self.N[q] = tempN
                self.S[q] = tempS
                self.Phi[q] = tempPhi
                self.dPhi[q] = (1/(2*np.pi))*(derivAphvgTGmm*tempN
                                - derivPhaseTerm + derivNoisePhi*tempN*Y[q]/sqrtS
                               )

            for q in range(mTrans, mTotal):
                for k in range(0, ndelta):

                    index = q*ndelta + k

                    bTN = bTIntv * tempN * tempN
                    invS = 1 / ((1/tempS) + epsilon)
                    sqrtS = np.sqrt(tempS)
                    cosPhi = np.cos(tempPhi)
                    senPhi = np.sin(tempPhi)

                    tempPhi = (tempPhi + aphvgTGmm*tempN - self.phaseTerm
                               + noisePhi*tempN*Y[index]/sqrtS
                               - (ampInject/sqrtS)*senPhi*cosInject[index]
                               + (ampInject/sqrtS)*cosPhi*senInject[index]
                              )

                    tempS = (tempS + vgTGmm*tempN*invS - vgTGmmN*invS
                             - intTtau*tempS + btGmm*bTN
                             + noiseS*tempN*sqrtS*X[index]
                             + 2*ampInject*sqrtS*cosPhi*cosInject[index]
                             + 2*ampInject*sqrtS*senPhi*senInject[index]
                            )

                    tempN = (tempN + self.currentTerm[index]
                             - aTIntv*tempN - bTN - (cTIntv*tempN**3)
                             - vgT*tempN*invS + vgtN*invS
                            )

                self.I[q] = self.currentTerm[index]*10**(12) / eVinv
                self.N[q] = tempN
                self.S[q] = tempS
                self.Phi[q] = tempPhi
                self.dPhi[q] = ((derivAphvgTGmm*tempN - derivPhaseTerm
                                 + derivNoisePhi*tempN*Y[index]/np.sqrt(tempS))
                                 / (2*np.pi)
                               )
```

```
                    self.opField[q-mTrans] = (np.sqrt(constP*tempS)
                                              * np.exp(1j*tempPhi)
                                              + opFldInject
                                              * np.exp(1j*angInject[index])
                                             )

            self.I[0] = 0
            self.N[0] = nTr
            self.S[0] = float(10**(15))
            self.Phi[0] = 0
            transFourier = np.fft.fft(self.opField)
            self.TFavg += (abs(np.fft.fftshift(transFourier))
                           * abs(np.fft.fftshift(transFourier))
                           / float(self.numWindw)
                          )

            self.TFang += (np.angle(np.fft.fftshift(transFourier))
                           / float(self.numWindw)
                          )
```