

Solving The Rate-Equations

Niall Boohan

Tyndall National Institute of Ireland

January 23, 2020

Overview

- 1 Introduction
- 2 Coding overview
- 3 Code — section by section
- 4 Code extension
- 5 Bibliography
References

Introduction

- The aim of this presentation is to introduce a known functioning script to solve the laser-rate equations in Python.
- You may choose to make these calculations in a different programming language but it is still important to start from a well functioning piece of code.
- By the end of this presentation I will have covered what each section of my code does, this will be relatable to other programming languages.

The laser-rate equation overview.

- The laser-rate equation describes how photon (S) & carrier concentration (N) in a laser cavity change with time for a given input current.
- The most basic form has two coupled ODE's, one for S & one for N .
- Each factor in the S or N equation has an associated change in N/s or S/s units associated with it.
- A time base for the calculations e.g 1ps. So for every 1ps, the code calculates the change in N or S and adds or subtracts this to the initial or continuing S or N values.

Simple example

$$\frac{dN}{dt} = \frac{I}{eV_a} - v_g g_a \bar{S} - R(N)$$

$$\frac{d\bar{S}}{dt} = v_g g_c \bar{S} + R_{sp}$$

Figure: [Buus, Amann, and Blumenthal, 2010]

- This is a simple example to make equations' operation more clear.
- These are not the equations used in the coded example.
- N is carrier concentration, I is current, e is unitary charge, g is gain, \bar{S} is photon concentration, R is spontaneous emission rate, v_g is group velocity.
- Time steps are chosen specifically in code and should be ~ 100 times smaller than the smallest timer interaction.

Coding overview

Implementing a pre-packaged solver:

- The example given is in Python, if you do not have a strong preference for a programming language it is best to use Python or MATLAB for simple models to begin with.
- For solving laser-rate equations it is best to start with a template the you know works and is robust and expand it. There are plenty of examples for MATLAB online.
- Link to the codes I have made in Python:
<https://github.com/boohann/Laser-Rate-Equations-Python/>
- Model parameters are passed to the solver which will be written in a low level language (Fortran/C++), so code will solve relatively quickly.
- Implementation is quite similar between Python and MATLAB.

Code — section by section

Call libraries

```
1  ### Import necessary libraries ###  
2  from scipy.integrate import ode  
3  import numpy as np  
4  import matplotlib.pyplot as plt  
5
```

- Libraries are repositories of prewritten functions, used heavily in Python.
- Libraries is one way Python can differ from other programming languages, you may not have to call these in languages like MATLAB.
- Always declared at top of script.

Change code operation

```
19 ### Mode select ###  
20 Mode = 1 # 0 is dynamic, 1 is steady-state LI  
21
```

- The code has been written to switch between producing an LI for a range of input currents or producing a time dependant trace of photon and carrier concentrations.
- This will be clearer later in the code.

Declaring output storage location

```
### Simulation Outputs ###  
N   = []      # y[0] Carrier concentration  
S   = []      # y[1] Photon concentration  
T   = []      # Time array output  
N_end = []     # Take final N value for steady-state behaviour  
S_end = []     # Take final S value for steady-state behaviour
```

- Lists for storing output data from simulation.
- These can be passed to graphing libraries or programs or post-processed or saved and stored.

Declaring input parameters

```
### Simulation input parameters ###
IA  = 20                                # Pumping current (mA)
I   = IA/1e3                            # Pumping current (A)
iIA = np.linspace(0, 50, 20)           # Generate multiple I for LI curve (mA)
II  = [x/1e3 for x in iIA]             # Multiple I (A)
q   = 1.6e-19                           # Electron charge (C)
V   = 2e-11                             # Device volume (cm^3)
tn  = 1.0e-9                            # Carrier relaxation time in seconds (s)
g0  = 1.5e-5                            # Gain slope constant (cm^3s^-1)
Nth = 1e18                              # Threshold carrier density (cm^-3)
EPS = 1.5e-17                           # Gain compression factor (cm^3)
Gamma = 0.2                             # Confinement factor
tp  = 1.0e-12                           # Photon lifetime in cavity (s)
Beta = 1.0e-4                           # Spontaneous Emission Factor
h   = 6.62607004e-34                   # Plank's constant (Js)
c   = 2.99792458e8                     # SOL (ms^-1)
WL  = 1300                             # WL (nm)
f   = c/(WL/1e9)                       # Frequency (Hz)
```

- Different input parameters are best declared at the top to make changing these values easier.
- Note: different values for current for steady-state or dynamic operation.

Inputting equations into solver

```
def call_solver(x):  
  
    """ Ensures global values of S, N and T are updated from this function """  
    global S  
    global N  
    global T  
  
    """ Define equations to be solved """  
    def laser_rates(t, y, p):  
  
        dy = np.zeros([2])  
        dy[0] = (x/(q*V)) - (y[0]/tn) - g0*(y[0] - Nth)*(y[1]/(1 + EPS*y[1]))  
        dy[1] = Gamma*g0*(y[0] - Nth)*(y[1]/(1 + EPS*y[1])) - y[1]/tp + (Gamma*Beta*y[0]) / tn  
  
        return dy
```

$$\frac{dN(t)}{dt} = \frac{I(t)}{qV} - \frac{N(t)}{\tau_n} - g_0(N(t) - N_t) \frac{1}{(1 + \varepsilon S(t))} S(t) \quad (1)$$

$$\frac{dS(t)}{dt} = \Gamma g_0(N(t) - N_t) \frac{1}{(1 + \varepsilon S(t))} S(t) - \frac{S(t)}{\tau_p} + \frac{\Gamma \beta N(t)}{\tau_n} \quad (2)$$

[Cartledge and Srinivasan, 1997]

- The first function (def) allows the entire solver to be called repeatedly for varying input current to calculate an LI curve.
- The second function (def) defines the equations we will input into our solver.
- "global" ensures T, N and S values are updated.

Running the solver

```
### Time, initial conditions & add parameters ###
t0 = 0; tEnd = 1e-8; dt = 1e-13          # Time constraints
y0 = [1e16, 0]                          # Initial conditions [N, S]
Y=[]; T=[]                                # Create empty lists
p = [I, q, V, tn, g0, Nth, EPS, Gamma, tp, Beta] # Parameters for odes

### Setup integrator with desired parameters ###
r = ode(laser_rates).set_integrator('vode', method = 'bdf')
#r = ode(laser_rates).set_integrator('dopri5', nsteps = 1e4)
r.set_f_params(p).set_initial_value(y0, t0)

### Simulation check ###
while r.successful() and r.t+dt < tEnd:
    r.integrate(r.t + dt)
    Y.append(r.y)          # Makes a list of 1d arrays
    T.append(r.t)
```

- First block of code defines time extents, initial conditions and organises our parameters into the solver.
- The second block of code calls the integrator (vode) and method (bdf) with initial conditions and parameters. It is best not to change solver and method unless absolutely necessary.
- The third block of code runs the solver while it is within the time extents and not diverging.

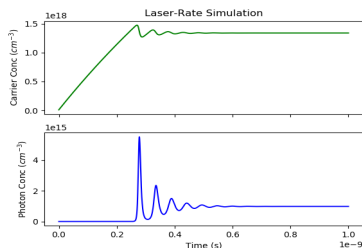
Saving outputs

```
### Format output ###  
Y = np.array(Y)          # Convert from list to 2d array  
N = Y[:, 0]  
S = Y[:, 1]  
  
### Take final value for steady-state LI ###  
S_end.append(S[-1:])  
N_end.append(N[-1:])
```

- This section of code outputs the data into the structures we defined at the beginning of the script.
- A save to data file routine could be added here if required.

Dynamic output

```
## Dynamic plotting ##  
def plot_dyan():  
  
    f, axarr = plt.subplots(2, sharex=True) # Two subplots, the axes array is 1-d  
    axarr[0].plot(T, N, 'G')  
    axarr[0].set_ylabel("Carrier Conc ($cm^{-3}$)")  
    axarr[0].set_title("Laser-Rate Simulation")  
    axarr[1].plot(T, S, 'B')  
    axarr[1].set_ylabel("Photon Conc ($cm^{-3}$)")  
    axarr[1].set_xlabel("Time (s)")  
    plt.show()  
  
    return;
```



- This routine plots the dynamic photon and carrier concentrations (Vs time).
- The solver is run for one input current.
- Generally this routine should be run up to $1e^{-9}s$, so the dynamic behaviour can be seen clearly.

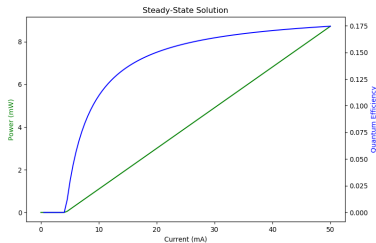
LI output

```
## Function for post solver steady-state LI calculations ##
def plot_SS():

    ## Post solver calculations
    P      = [h**4*((i*V)/tp)*1e3 for i in S_end]    # Power output (mW)
    QE      = [i/j for i,j in zip(P, IIA)]           # Convert for quantum efficiency

    ## Plotting two parameters on one plot ##
    fig, ax1 = plt.subplots()
    ax2 = ax1.twinx()
    ax1.plot(IIA, P, 'g-')
    ax2.plot(IIA, QE, 'b-')
    ax1.set_xlabel('Current (mA)')
    ax1.set_ylabel('Power (mW)', color='g')
    ax2.set_ylabel('Quantum Efficiency', color='b')
    plt.title('Steady-State Solution')
    plt.show()

    return;
```



- This routine plots the steady-state output power Vs input current.
- The solver is run for a range of input current ($0 \rightarrow 50\text{mA}$).
- Generally this routine should be run up to $1e^{-8}\text{s}$, so the laser is in a stable operating point when power value is extracted.
- Post-processing of data is taking place to convert photon concentration to output power.

"Main" section

```
### Dynamic mode ###  
if(Mode == 0):  
    call_solv(I)  
    plot_dynam()  
  
### Steady-state mode ###  
if(Mode == 1):  
    for i in iI:  
        call_solv(i)  
    plot_SS()
```

- This portion of the code is implementing the two different operational modes of the code.
- It can be noted for the LI operation the solver is called repeatedly for an increasing input current.

Code extension

Extending the code

```
## Define equations to be solved ##
def Laser_rates(t, y, p):
    dy = np.zeros([4])
    dy[0] = k/(q+1) - y[0]/t_0 - C*(y[0]**2)*2*N_d*(1-y[2]) # N
    dy[1] = -v_g*sigRes*(2*y[1]-1)*y[3] - y[1]/t_D0 + C*(y[0]**2)*(1-y[1]) - B_tran*(y[1]-y[2]) # roRes
    dy[2] = -v_g*sigRes*r_star*(2*y[1]-1)*y[3] - y[2]/t_D0 + C*(y[0]**2)*(1-y[2]) # ro
    dy[3] = -y[3]/t_ph + Gam*v_g*((2*N_d*sigRes*r_star*(2*y[1]-1))/d)*y[3] # S
    return dy
```

- The main aim of this presentation was to introduce a simple functioning solver for the laser-rate equations.
- A basic functioning template can then be extended in any number of ways to model various laser parameters.
- It can then be included with various other further models such as travelling-wave models or scattering matrix method models to capture higher levels of laser behaviour.

Bibliography

- Buus, Jens, Markus-Christian Amann, and Daniel J. Blumenthal (2010). *Fundamental Laser Diode Characteristics*. Pp. 7–41. ISBN: 0471208167. DOI: 10.1109/9780470546758.ch2.
- Cartledge, J. C. and R. C. Srinivasan (1997). “Extraction of DFB laser rate equation parameters for system simulation purposes”. In: *Journal of Lightwave Technology* 15.5, pp. 852–860. ISSN: 1558-2213. DOI: 10.1109/50.580827.

The End