Chapter 2

TL FROM DELOCALIZED TRANSITIONS: MODELS

1

Code 2.1: Solution of OTOR model using ODE solver odeint

```
# Solution of ODE's for the OTOR model using solver odeint
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
# Initial conditions.
n0, nc0 ,m0 = 1e10, 0, 1e10
# Initial conditions vector
y0 = n0, nc0, m0
# Numerical parameters for OTOR.
N,    An,    Am,    s,   E , hr = \
1e10, 1e-8, 1e-8,  1e12, 1 , 1
# A grid of time points in seconds
t = np.linspace(0, 180, 180)
kB=8.617e-5
# differential equations of the OTOR model.
def deriv(y, t):
    n, nc, m = y
    dndt = - n*s*np.exp(-E/(kB*(273+hr*t)))+ nc*An*(N-n)
    dncdt =  n*s*np.exp(-E/(kB*(273+hr*t)))- nc*An*(N-n)-m*Am*nc
    dmdt = -m*Am*nc
    return dndt, dncdt, dmdt
# Integrate the OTOR equations over the time grid, t.
# Call `odeint` to generate the solution.
ret = odeint(deriv, y0, t)
n, nc, m = ret.T
# Plot the data
plt.subplot(1,2, 1);
plt.plot(hr*t, n, 'b',  linewidth=2, label='n(t)');
plt.plot(hr*t,nc*1000,'r--',linewidth=2,label='nc(t) x10$^{3}$');
leg = plt.legend();
leg.get_frame().set_linewidth(0.0);
plt.ylabel('Filled traps [cm$^{-3}$]');
plt.ylim(0,1.2e10);
plt.xlim(0,180);
plt.title('(a)');
plt.text(20,.6e10,'OTOR model');
plt.text(20,.5e10,'numerical');
plt.text(20,.4e10,'solution');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.subplot(1,2, 2);
TL= n*s*np.exp(-E/(kB*(273+hr*t)))- nc*An*(N-n)
plt.plot(hr*t,TL,linewidth=2,label='TL=-dn/dt');
plt.ylim(0,2.2e8);
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.title('(b)');
plt.tight_layout()
plt.show()
```

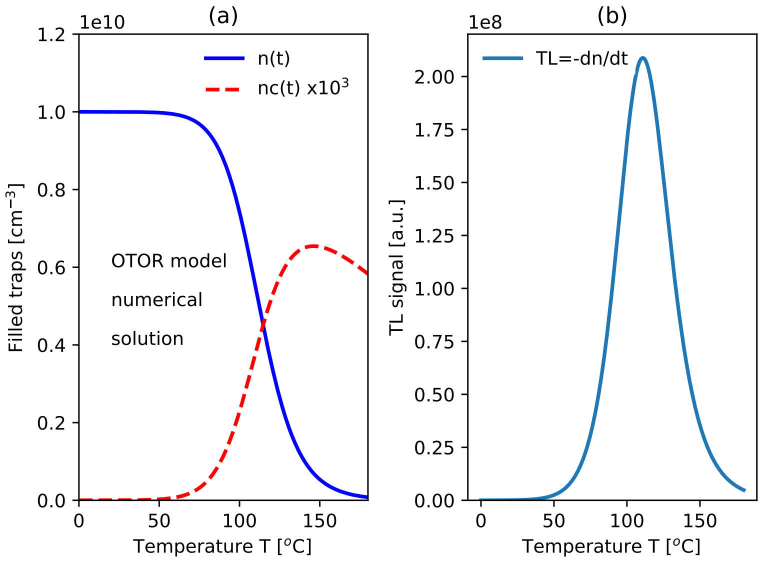
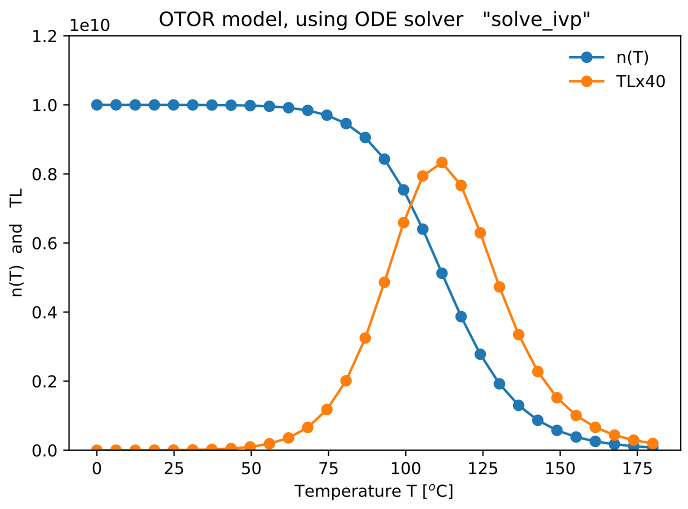


Fig. 2.1: Numerical solution of the system of differential equations for the OTOR model using the ODE solver function `odeint()`. (a) Plot of the concentrations of trapped electrons $n(T)$ (solid line) and the conduction band electrons $n_c(T)$ (dashed line), and (b) Plot of the luminescence intensity $I(T)$, during a typical TL experiment. Note that $n_c(T)$ has been multiplied in (a) by a factor of 10^3, for display purposes.

Code 2.2: Solution of OTOR model using the ODE solver solve_ivp

```
# Solution of the OTOR using the solver solve_ivp
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
# Numerical parameters for OTOR.
N,    An,   Am,     s,   E , hr ,  kB= \
1e10, 1e-8, 1e-8,  1e12, 1 , 1,   8.617e-5
# Define derivative function
def f(t, y):
    dydt=[
    -y[0]*s*np.exp(-E/(kB*(273+hr*t)))+y[1]*An*(N-y[0]),
    y[0]*s*np.exp(-E/(kB*(273+\
    hr*t)))- y[1]*An*(N-y[0])-y[2]*Am*y[1],
    -y[2]*Am*y[1]]
    return dydt
tspan = np.linspace(0, 180,30)
yinit = [1e10,0,1e10]
# Solve differential equation
sol = solve_ivp(lambda t, y: f(t, y), [tspan[0], tspan[-1]],\
yinit, t_eval=tspan)
temps=hr*sol.t.reshape(-1,1)
nvals,ncvals,mvals=np.squeeze(sol.y)
# Plots
plt.plot(temps,nvals,'o-',label='n(T)');
plt.ylabel('n(T)  and   TL');
plt.ylim(0,1.2e10);
plt.title('OTOR model, using ODE solver   "solve_ivp"');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.plot(sol.t, 40*Am*ncvals*mvals,'o-',label='TLx40');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.tight_layout()
plt.show()
```



Code 2.3: Varying the parameters E,s in the OTOR model

Fig. 2.2: The numerical solution obtained using the ODE solver function `solve_ivp()`. In this case the output of the solver is evaluated at equally spaced time intervals, determined by the parameter `tspan` in the Python code.

```
# Varying the parameters E,s in the OTOR model
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
# Numerical parameters for OTOR.
N,      Am,     hr ,  kB= \
1e10,  1e-8   , 1,  8.617e-5
yinit = [1e10,0,1e10]
def TL(An,E,s):
    def f(t, y):
        dydt =[-y[0]*s*np.exp(-E/(kB*(273+hr*t)))+y[1]*An*\
        (N-y[0]), y[0]*s*np.exp(-E/(kB*(273+hr*t)))- y[1]*\
        An*(N-y[0])-y[2]*Am*y[1], -y[2]*Am*y[1]]
        return dydt
    sol = solve_ivp(lambda t, y: f(t, y), [20,220],yinit)
    nvals,ncvals,mvals=np.squeeze(sol.y)
    temps=hr*np.squeeze(sol.t)
    return temps, nvals,ncvals,mvals
plt.subplot(1,2,1);
labels=['E=1.0 eV','E=1.05 eV','E=1.1 eV']
lins=['solid','dashed','dotted']
E=[1.,1.05,1.1]
for j in range (0,3,1):
     temps, nvals, ncvals, mvals=TL(1e-10,E[j],1e12)
     plt.plot(temps, Am*ncvals*mvals,
     linestyle=lins[j], linewidth=3,label=labels[j]);
plt.ylim(0,4.2e8);
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.title(r'(a)');
plt.subplot(1,2,2);
s=[1e11,1e12,5e12]
labels=[r's=10$^{11}$ s$^{-1}$',r's=10$^{12}$ s$^{-1}$',
r's=5x10$^{12}$ s$^{-1}$']
```

```
for j in range (0,3,1):
     temps, nvals, ncvals, mvals=TL(1e-10,1,s[j])
     plt.plot(temps, Am*ncvals*mvals,
     linestyle=lins[j],linewidth=3, label=labels[j]);
plt.ylim(0,5.2e8);
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.title(r'(b)');
plt.tight_layout()
plt.show()
```

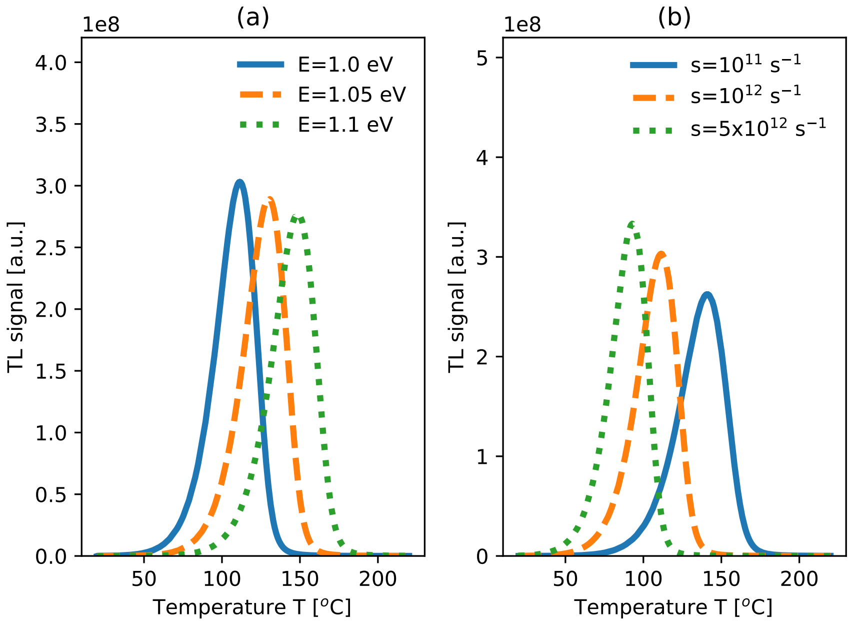


Fig. 2.3: The effect of changing the kinetic parameters E, s characterizing the electron trap, within the OTOR model. (a) As the energy E increases, the TL peak shifts towards higher temperatures, while the area under the curve remains the same. (b) Increasing the frequency factor s has the opposite effect, and the TL glow curve shifts towards lower temperatures, while the area stays again the same.

Code 2.4: Varying the retrapping ratio R in the OTOR model

```
# Varying the parameter R in the OTOR model
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
# Numerical parameters for OTOR.
N,      Am,     hr ,  kB= \
1e10,  1e-8   , 1,  8.617e-5
yinit = [1e10,0,1e10]
def TL(An,E,s):
    def f(t, y):
        dydt =[-y[0]*s*np.exp(-E/(kB*(273+hr*t)))+y[1]*An*\
        (N-y[0]), y[0]*s*np.exp(-E/(kB*(273+hr*t)))- y[1]*\
        An*(N-y[0])-y[2]*Am*y[1], -y[2]*Am*y[1]]
        return dydt
    sol = solve_ivp(lambda t, y: f(t, y), [50,200],yinit)
    nvals,ncvals,mvals=np.squeeze(sol.y)
    temps=hr*np.squeeze(sol.t)
    return temps, nvals,ncvals,mvals
#######
temps, nvals, ncvals, mvals=TL(1e-11,1,1e12)
plt.plot(temps, Am*ncvals*mvals,'-',linewidth=3,
label='R=0.001');
temps, nvals, ncvals, mvals=TL(1e-8,1,1e12)
plt.plot(temps, Am*ncvals*mvals,'--',linewidth=3,
label='R=1');
temps, nvals, ncvals, mvals=TL(3e-8,1,1e12)
plt.plot(temps, Am*ncvals*mvals,'-.',linewidth=3,
 label='R=3');
plt.ylim(0,4e8);
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.title(r'OTOR model, varying the retrapping ratio R');
plt.text(170,2.5e8,r'R=A$_{n}$/A$_{m}$');
plt.tight_layout()
plt.show()
```

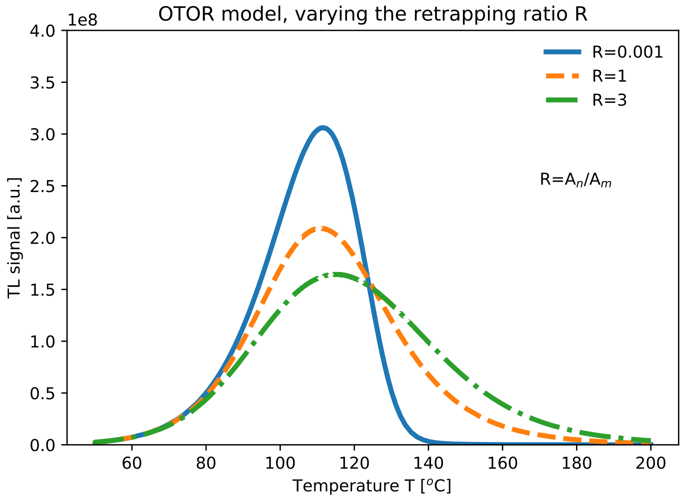


Fig. 2.4: The effect of changing the retrapping ratio $R = A_n/A_m$ on the TL glow curve, within the OTOR model. For low values of $R << 1$ the asymmetric shape corresponds to first order kinetics, while for values of $R \sim 1$ the almost symmetric shape is characteristic of second order kinetics.

Code 2.5: ODE for TL: First order kinetics

```
# Numerically solve the ODE for R-W: first order kinetics
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
# Function for analytical solution
def nfirst(n0,tmps):
    return n0*np.exp(-s*kB*((273+tmps)**2.0)/(hr*E)*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E))
def TLfirst(n0,tmps):
    return n0*s*np.exp(-E/(kB*(273+tmps)))*\
    np.exp(-s*kB*((273+tmps)**2.0)/(hr*E)*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E))
# Numerical parameters for R-W model.
```

```
kB,        N,     s,    E , n0 ,  hr=\
8.617e-5, 1e10,  1e12,  1 , 1e10, 1
# A grid of time points and temperatures
t = np.linspace(0, 180, 60)
temps=hr*t
# The R-W model differential equation.
def deriv(y, t):
    n = y
    dndt = - n*s*np.exp(-E/(kB*(273+hr*t)))
    return dndt
y0 = n0
ret = odeint(deriv, y0, t)
n= ret.flatten()
# Plot the data
plt.subplot(1,2, 1);
plt.plot(temps, n, 'o', label='Numerical');
plt.plot(temps,[nfirst(n0,x) for x in temps],'-',\
label='Analytical');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('Filled traps [cm$^{-3}$]');
plt.ylim(0,1.2e10);
plt.xlim(0,180);
plt.title('(a)');
plt.text(20,.6e10,'R-W model');
plt.text(20,.5e10,'numerical');
plt.text(20,.4e10,'solution n(t)');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.subplot(1,2, 2);
TL= n*s*np.exp(-E/(kB*(273+hr*t)))
plt.plot(temps,TL,'o',label='Numerical');
plt.plot(temps,[TLfirst(n0,x) for x in temps],'-',
label='Analytical');
plt.ylim(0,4e8);
plt.text(50,1.5,'TL');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.title('(b)');
plt.tight_layout()
plt.show()
```

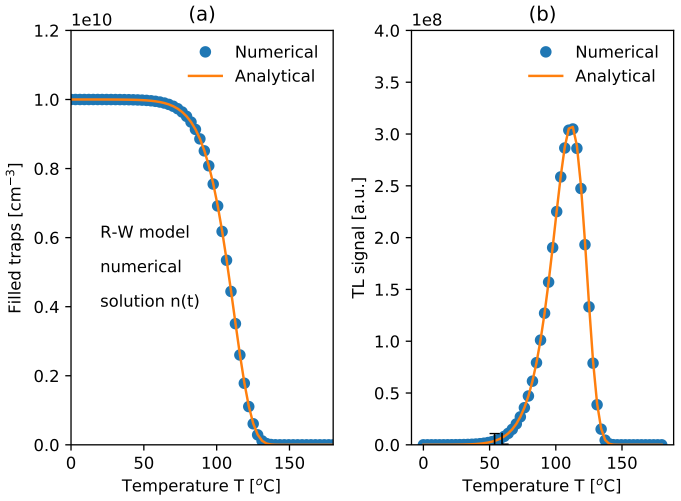


Fig. 2.5: Numerical solution of the Randall and Wilkins Eq.(**??**), for first order kinetics TL is shown as circles. (a) Remaining trapped electrons $n(T)$, and (b) The corresponding TL signal. The solid lines represent the analytical Eqs.(**??**) and (**??**), respectively.

Code 2.6: First-order TL by varying the initial trap concentrations

```
# Simulate first-order glow peaks with various
# initial electron trap concentrations (n0).
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
kB,       E,  s,    beta =\
8.617e-5, 1, 1e12, 1
def TLfirst(n0,tmps):
    return n0*s*np.exp(-E/(kB*(273+tmps)))*\
    np.exp(-s*kB*((273+tmps)**2.0)/(beta*E)*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E))
tims= range(50,int(170/beta)-1,1)
temps=[beta*tim for tim in tims]
plt.plot(temps,[TLfirst(1e10,x) for x in temps],'+-',c='r',
label=r'n$_{0}$=1e10 cm$^{-3}$');
plt.plot(temps,[TLfirst(2e10,x) for x in temps],'o-',c='g',
label=r'n$_{0}$=2e10 cm$^{-3}$');
plt.plot(temps,[TLfirst(3e10,x) for x in temps],'^-',c='b',
label=r'n$_{0}$=3e10 cm$^{-3}$');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.title('Variable Initial occupancy n0');
plt.text(60,8e8,'First order');
plt.text(60,7.2e8,'Kinetics');
plt.ylabel(r'TL [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.show()
```

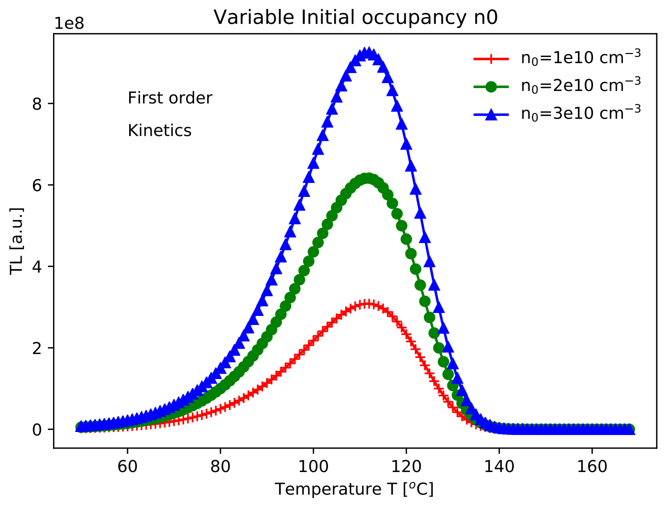


Fig. 2.6: Simulation of first-order TL glow peaks with three different initial electron trap concentrations $n_0 = 1\,,2,\ 3 \times 10^{10}$ cm^{-3}. The location of the maximum TL intensity does not shift significantly as n_0 changes, and the peak height and total area under the curve is proportional to n_0.

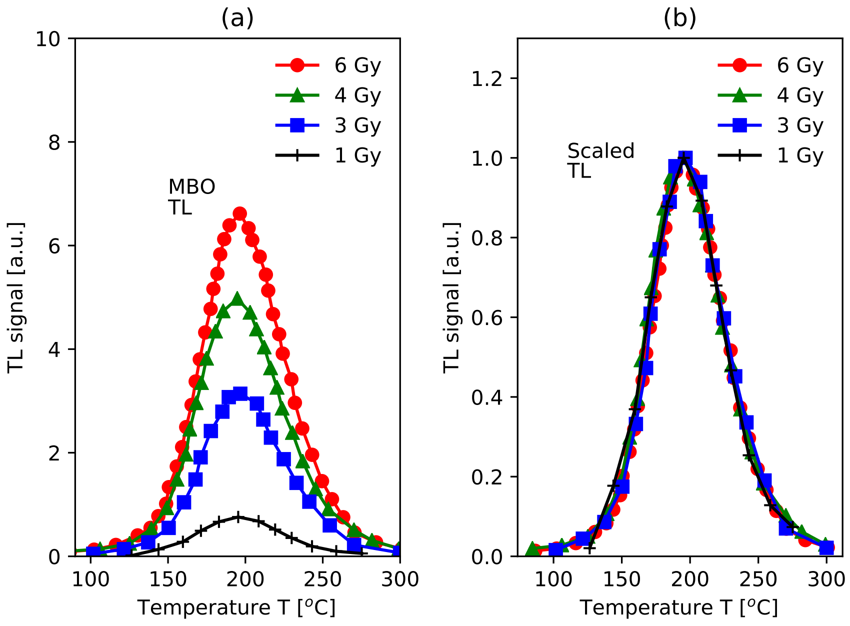


Fig. 2.7: (a) Example of a series of TL glow curves for sample MBO, at different beta doses. (b) The data in (a) is normalized to the maximum TL height. As the dose increases, the shape of the TL glow curves in (a) stays the same, while the height of the TL signal is proportional to the irradiation dose. For more details see Pagonis et al. [**?**].

Code 2.7: Second-order TL by varying the initial trap concentrations

```
# Simulate second-order glow peaks with various
# initial electron trap concentrations (n0).
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
kB,        E,  s,    beta , N=\
8.617e-5, 1, 1e12,   1 ,   1e10
```

```
def TLsec(n0,tmps):
    expint=kB*((273+tmps)**2.0)/E*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E)
    return (n0**2.0)*(s/N)*np.exp(-E/(kB*(273+tmps)))*\
    ((1+(n0*s/(beta*N))*expint)**(-2.00) )
tims= range(50,int(220/beta)-1,1)
temps=[beta*tim for tim in tims]
plt.plot(temps,[TLsec(2e9,x) for x in temps] ,'+-',
c='r',label=r'n$_{0}$/N=0.2');
plt.plot(temps,[TLsec(4e9,x) for x in temps] ,'o-',
c='g',label=r'n$_{0}$/N=0.4');
plt.plot(temps,[TLsec(6e9,x) for x in temps] ,'^-',
c='b',label=r'n$_{0}$/N=0.6');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.title('Variable Initial occupancy n0');
plt.text(50,1e8,'Second order');
plt.text(50,.9e8,'Kinetics');
plt.ylabel(r'TL [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.show()
```

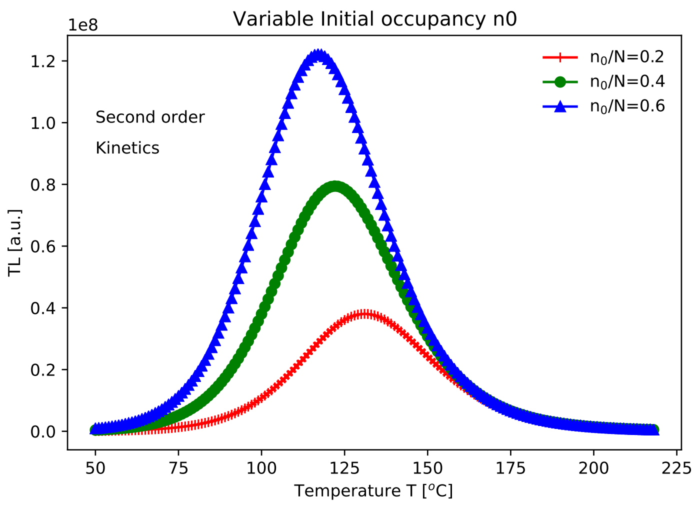


Fig. 2.8: Simulation of second order TL glow peaks with three different initial electron trap concentrations $n_0/N = 0.2, 0.4, 0.6$. As the dose increases, the shape of the TL glow curve does not stay the same, and the location of the maximum TL intensity shifts towards lower temperatures. However, the *area* under the peak stays proportional to n_0.

Code 2.8: 1st and 2nd order TL with the same parameters

```
# Compare 1st and 2nd order TL with the same parameters
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
kB,       E,  s,    beta , N=\
8.617e-5, 1, 1e12,   1 ,   1e10
def TLfirst(n0,tmps):
    return n0*s*np.exp(-E/(kB*(273+tmps)))*\
    np.exp(-s*kB*((273+tmps)**2.0)/(beta*E)*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E))
def TLsec(n0,tmps):
    expint=kB*((273+tmps)**2.0)/E*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E)
    return (n0**2.0)*(s/N)*np.exp(-E/(kB*(273+tmps)))*\
    ((1+(n0*s/(beta*N))*expint)**(-2.00) )
tims= range(25,int(220/beta)-1,1)
temps=[beta*tim for tim in tims]
plt.plot(temps,[TLfirst(1e10,x) for x in temps],'+-',
c='r',label=r'b=1');
plt.plot(temps,[TLsec(1e10,x) for x in temps] ,'o-',c='b',
label=r'b=2');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.title('Compare b=1 and b=2');
plt.text(40,2.7e8,'Compare Kinetics');
plt.text(40,2.5e8,'First order vs');
plt.text(40,2.3e8,'second order');
plt.ylabel(r'TL [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.show()
```

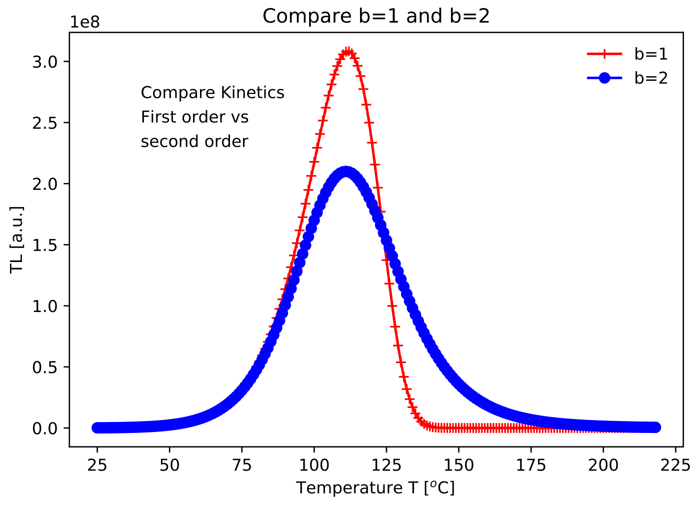


Fig. 2.9: Comparison of simulated first and second order TL peaks, evaluated with the same kinetic parameters. The second order process is slower in time, and the peak shape for second order is almost symmetric.

Code 2.9: The initial rise method: find energy E from TL data

```
# Apply the initial rise method to find the activation energy E
# Load the data from txt file, which  contains pairs of
# data  in the form: (Temperature_in_K,TL_Intensity (any units)
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from prettytable import PrettyTable
data = np.loadtxt('lbodata.txt')
x_data,y_data = data[:, 0], data[:, 1]
plt.subplot(1,2, 1);
plt.plot(x_data,y_data,'+-',c='green',label='TL');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [K]');
plt.ylim(0,140);
plt.text(370, 100,'Sample');
plt.text(370, 90,'LBO');
kB=8.617*1e-5 # Boltzmann constant in eV/K
initialPos=7  #analyze data points  #6 to #14
finalPos=13
plt.plot(x_data[initialPos:finalPos],y_data[initialPos:\
finalPos],"o",c='r',label='IR-data');
plt.title('(a)');
x= 1/(kB*x_data[initialPos:finalPos])
y= np.log(y_data[initialPos:finalPos])
slope, intercept,r_value,p_value, std_err=stats.linregress(x,y)
plt.subplot(1,2,2);
plt.plot(x,y,'ro',label='Data');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.plot(x,x*slope+intercept,label='Fit');
plt.xlabel(r'1/(kT) (eV$^{-1}$)');
plt.ylabel('ln(TL)');
plt.title('(b)  Initial rise method');
plt.tight_layout()
slope, intercept, rsqr, p_value,std_err=\
np.round(slope,2),np.round(intercept,2),np.round(\
r_value**2.0,3),format(p_value,"10.2E"), np.round(std_err,2)
myTable=PrettyTable(['E (eV)','dE (eV)','p-value','R^2'])
myTable.add_row([-slope, std_err, p_value,rsqr]);
print(myTable)
plt.show()
```

```
  +--------+---------+------------+-------+
  | E (eV) | dE (eV) |  p-value   |  R^2  |
  +--------+---------+------------+-------+
  |  1.26  |   0.05  |   1.56E-05 | 0.994 |
  +--------+---------+------------+-------+
```

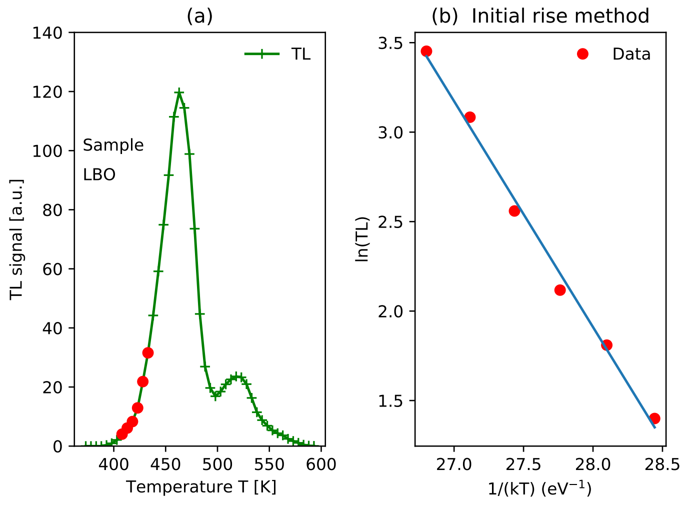


Fig. 2.10: Example of applying the initial rise method to find the activation energy E. (a) The red circles indicates the initial rise area being analyzed; (b) The slope of the best line fit gives the activation energy ($-E$).

Code 2.10: TL glow curve for three different heating rates

```
# Plot the same TL glow curve for three different heating rates
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
kB,        E,  s,      n0 ,     N=\
8.617e-5, 1, 1e12,   1e10 ,   1e10
def TLfirst(beta,tmps):
    return n0*(s/beta)*np.exp(-E/(kB*(273+tmps)))*\
    np.exp(-s*kB*((273+tmps)**2.0)/(beta*E)*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E))
temps=range(50,160,1)
symb=('o-','^-','+-')
for j in range(1,4,1):
    plt.plot(temps,[TLfirst(j,x) for x in temps],
    symb[j-1]);
plt.title('Variable heating rates');
plt.xlim(50,160);
plt.ylim(0,4e8);
plt.text(105,3.3e8,'1 K/s');
plt.text(117,3.1e8,'2 K/s');
plt.text(130,2.8e8,'3 K/s');
plt.ylabel(r'TL/$\beta$ [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.show()
```

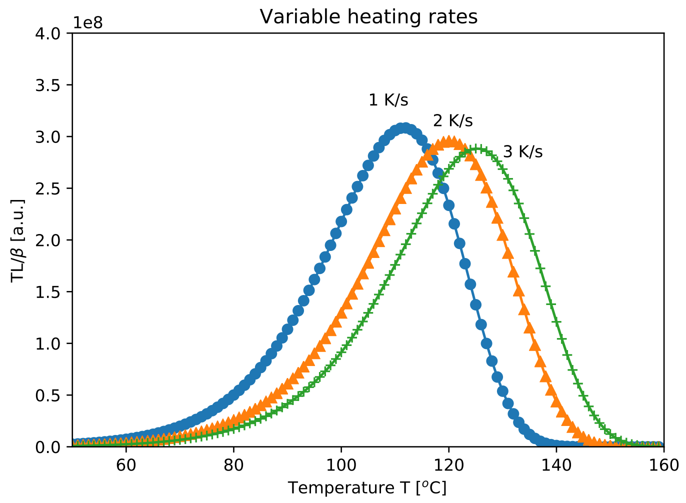


Fig. 2.11: Simulation of the same TL glow curve measured with three different heating rates $\beta = 1, 2, 3$ K/s. As the heating rate increases, the TL peak shifts to the right towards higher temperatures, and the intensity decreases. However, the area under the curves stays the same. Notice that the y-scale is TL/β, i.e. in counts/K, and not in counts/s.

Code 2.11: Apply heating rate method to TL data, to find E,s

```
# Apply the heating rate method to find E,s
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

```
from scipy import stats
from prettytable import PrettyTable
kB,       E, s,     n0 =\
8.617e-5, 1, 1e12, .2e10
def TLfirst(beta,temps):
    return n0*(s/beta)*np.exp(-E/(kB*(273+temps)))*\
    np.exp(-s*kB*((273+temps)**2.0)/(beta*E)*\
    np.exp(-E/(kB*(273+temps)))*(1-2*kB*(273+temps)/E))
temps=np.arange(50,160)
It1=[TLfirst(1,x) for x in temps]
It2=[TLfirst(2,x) for x in temps]
It3=[TLfirst(3,x) for x in temps]
It4=[TLfirst(4,x) for x in temps]
plt.subplot(1,2, 1);
symb=('o-','^-','+-','x-')
for j in range(1,5,1):
    plt.plot(temps,[TLfirst(j,x) for x in temps],symb[j-1]);
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.text(130,6e7,r'$\beta$=1-4 K/s');
plt.xlim(70,160);
plt.ylim(0,7e7);
plt.title('(a)');
Tmax=[273+temps[np.argmax(It1)],273+temps[np.argmax(It2)],
273+temps[np.argmax(It3)],273+temps[np.argmax(It4)]]
y=[np.log(Tmax[0]**2/1),np.log(Tmax[1]**2/2),np.log(Tmax[2]**2/\
3),np.log(Tmax[3]**2/4)]
x=[1.0/(kB*u) for u in Tmax]
plt.subplot(1,2, 2);
plt.plot(x,y,"o",c="r");
plt.ylabel(r'ln(T$_{m}^{2}$/$\beta$)');
plt.xlabel(r'1/(kT$_{m}$) [eV$^{-1}$]');
x=np.array(x)
y=np.array(y)
slope, intercept,r_value,p_value,std_err=stats.linregress(x,y)
plt.plot(x,x*slope+intercept,label='Fit');
slope, intercept, rsqr, std_err=np.round(slope,2),\
np.round(intercept,2),np.round(r_value**2.0,3),\
np.round(std_err,2)
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.title('(b)');
plt.text(29,11.6,'Heating rate');
plt.text(29,11.5,'method');
plt.tight_layout()
s=format(np.exp(-intercept)*slope/kB, "10.2E")
myTable=PrettyTable([ "E (eV)",  "dE (eV)",\
"intercept I","s (s^-1)","R^2"]);
myTable.add_row([slope, std_err, intercept, s, rsqr]);
print(myTable)
plt.show()
```

```
+--------+---------+-------------+------------+-------+
| E (eV) | dE (eV) | intercept I |  s (s^-1)  |  R^2  |
+--------+---------+-------------+------------+-------+
|  1.02  |   0.02  |    -19.0    |   2.11E+12 | 0.999 |
+--------+---------+-------------+------------+-------+
```

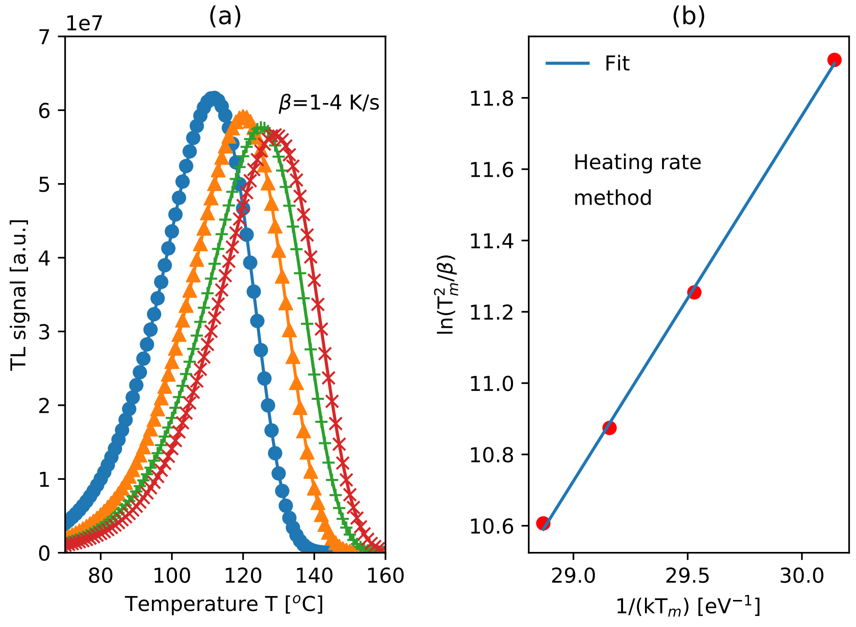


Fig. 2.12: Applying the heating rate method to obtain both the activation energy E and the frequency factor s. (a) The simulated TL glow curves for heating rates β =1-4 K/s and (b) The slope and intercept of the best fit line yield both parameters (E, s).

Code 2.12: Plot of the empirical analytical equation for GOK

```
# Plot of the analytical equation for GOK
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
kB,       E,  s,    beta , n0,    N=\
8.617e-5, 1, 1e12,   1 ,   1e10,  1e10
def TLgen(b,tmps):
    expint=kB*((273+tmps)**2.0)/E*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E)
    a=(n0**(b-1))*(s/(N**(b-1)))
    return n0*a*np.exp(-E/(kB*(273+tmps)))*\
    ((1+(b-1)*a/beta*expint)**(-b/(b-1)) )
tims= range(25,int(220/beta)-1,3)
temps=[beta*tim for tim in tims]
plt.plot(temps,[TLgen(1.001,x) for x in temps],'+-',
c='r',label=r'b=1.001');
plt.plot(temps,[TLgen(1.5,x) for x in temps] ,'o-',
 c='g',label=r'b=1.5');
plt.plot(temps,[TLgen(2.0,x) for x in temps] ,'^-',
 c='b',label=r'b=2');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.title('Compare GOK TL glow curves: b=1, 1.5 and 2');
plt.ylabel('TL [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.show()
```

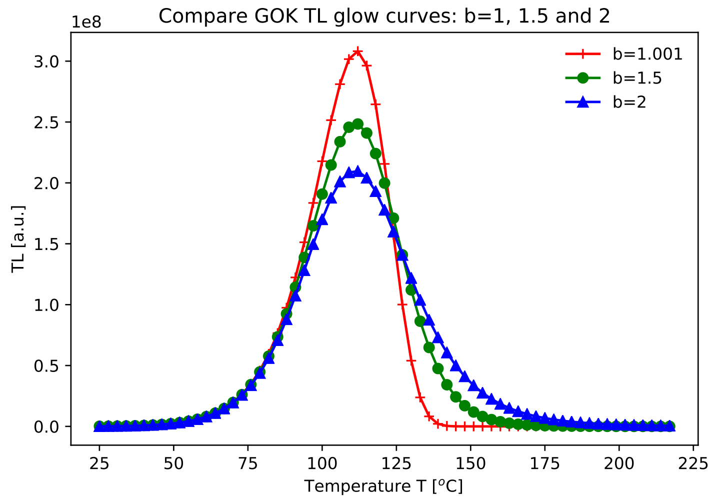


Fig. 2.13: Plot of GOK Eq.(**??**) for three values of the kinetic order parameter b. As b increases from $b = 1.001$ to $b = 2$, the TL glow curve changes from the asymmetric first order kinetics to the very nearly symmetric second order TL glow curve.

Code 2.13: The GOT equation for TL in the OTOR model

```
# Numerical solution of the GOT equation for TL
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
# Numerical integration of GOT model.
n0,    N,    R,    s,    E ,hr , kB=\
1e10, 1e10, 1,  1e12, 1 , 1,  8.617e-5
t = np.linspace(0, 180, 180)
def deriv(y, t):
    n = y
    dndt = - (n**2.0)*s*np.exp(-E/(kB*(273+hr*t)))/((R*(N-n)+n))
    return dndt
y0 = n0
ret = odeint(deriv, y0, t)
```

```
n= ret.flatten()
# Plot the data
plt.plot(hr*t, n, 'b+',  linewidth=3, label='n(t)');
plt.ylabel('n(t)  and  TL');
plt.ylim(0,1.2e10);
plt.xlim(0,180);
plt.title('Numerical integration of GOT model');
plt.xlabel(r'Temperature T [$^{o}$C]');
TL=  (n**2.0)*s*np.exp(-E/(kB*(273+hr*t)))/((R*(N-n)+n))
plt.plot(hr*t,30*TL,'r-', linewidth=3,label='TL x30');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.tight_layout()
plt.show()
```

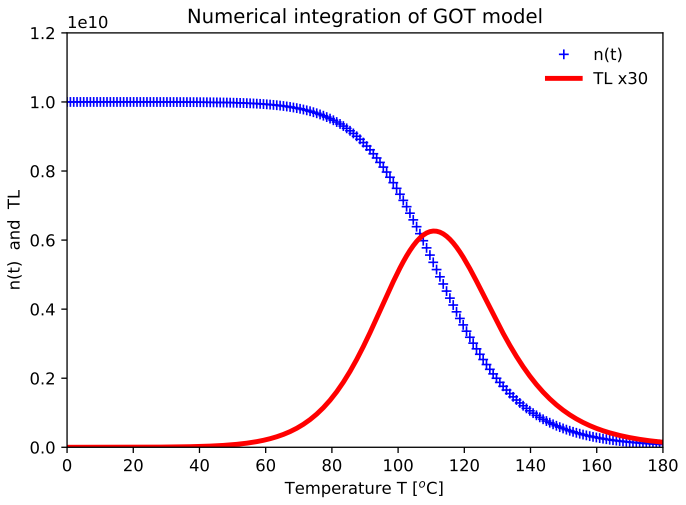


Fig. 2.14: Numerical solution of the GOT model Eq.(**??**) for TL, showing the concentration of filled traps $n(T)$, and the corresponding TL glow curve.

Code 2.14: Plot the Lambert function and solution to GOT model

```
#  Plot the analytical solution of GOT, using Lambert W-function
from scipy.special import lambertw
```

```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
t = np.arange(0.0, 150.0, 1.0)
ys = lambertw(t)
plt.subplot(1,2, 1);
plt.plot(t, ys,label='Lambert W(x)');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('W(x)');
plt.xlabel('x');
plt.title('(a)');
plt.subplot(1,2, 2);
t = np.arange(0.0, 150.0, 3)
no,      N,      R,    s,   E, kB,      beta= \
1.0E10, 1.0E10, 1e-4, 1e12, 1, 8.617e-5, 1
c=(no/N)*(1-R)/R
expint=kB*((273+beta*t)**2.0)/(beta*E)*\
    np.exp(-E/(kB*(273+t*beta)))*(1-2*kB*(273+beta*t)/E)
zTL=(1/c)-np.log(c)+(s*expint/(beta*(1-R)))
lam=np.real(lambertw(np.exp(zTL)))
plt.plot(t,(N*R/((1-R)))*(1/lam),'r-',label='n(t)');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('n(t)   and  TL');
plt.ylim(0,1.2e10);
plt.xlim(0,150);
plt.title('(b)');
plt.text(20,.6e10,'GOT model');
plt.text(20,.5e10,'KV-TL equation');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.plot(t,30*(N*R/((1-R)**2.0))*s*np.exp(-E/(kB*(273+t*beta)\
))/(lam+lam**2),'b--',label='TL x30');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.title('(b)');
plt.tight_layout()
plt.show()
```

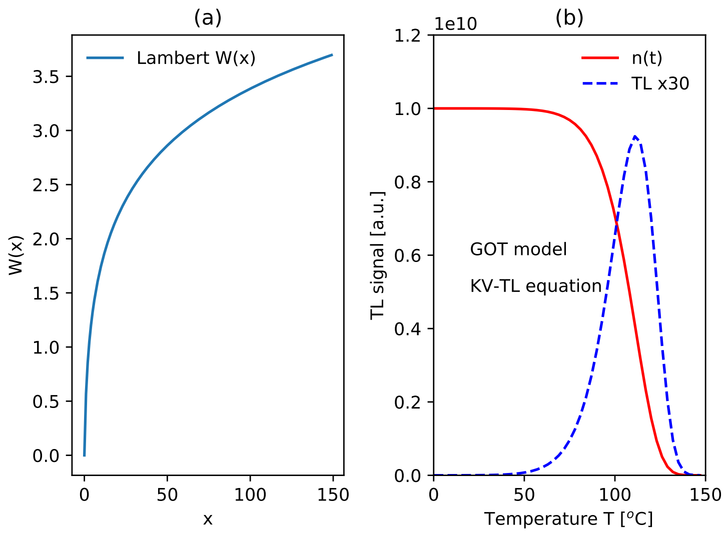


Fig. 2.15: (a) Plot of the real positive branch of the Lambert function $W(x)$ between $x = 0$ and $x = 150$. (b) Plot of the analytical Eq.(**??**) for $n(t)$ of the GOT model and the corresponding analytical KV-TL Eq.(**??**), based on the Lambert W function.

Code 2.15: Plot the solution of GOT model using omega function

```
#  Plot the solution of GOT, using the Wright omega function
from scipy.special import lambertw
from scipy.special import wrightomega
import numpy as np
import matplotlib.pyplot as plt
t = np.arange(0.0, 200.0, 1.0)
ys = lambertw(t)
no,      N,      R,    s,   E, kB,      beta= \
1.0E10, 1.0E10, .99, 1e12, 1, 8.617e-5, 1
c=(no/N)*(1-R)/R
expint=kB*((273+beta*t)**2.0)/(beta*E)*\
    np.exp(-E/(kB*(273+t*beta)))*(1-2*kB*(273+beta*t)/E)
zTL=(1/c)-np.log(c)+(s*expint/(beta*(1-R)))
```

```
lam=np.real(lambertw(np.exp(zTL)))
lam2=wrightomega(zTL)
plt.subplot(1,2, 1);
plt.plot(t,(N*R/((1-R)))*(1/lam),'r-',label='n(t)');
plt.plot(t,30*(N*R/((1-R)**2.0))*s*np.exp(-E/(kB*(273+t*beta)\
))/(lam+lam**2),'b--',label='TL x30');
plt.title('(a)');
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.text(0,.7e10,'KV-TL eqt.');
plt.text(0,.6e10,'lambertw');
plt.text(0,.5e10,'overflow');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.subplot(1,2, 2);
plt.ylabel('n(t)   and  TL');
plt.text(0,.7e10,'KV-TL eqt.');
plt.text(0,.6e10,'with');
plt.text(0,.5e10,'wrightomega');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.plot(t,(N*R/((1-R)))*(1/lam2),'r-',label='n(t)');
plt.plot(t,30*(N*R/((1-R)**2.0))*s*np.exp(-E/(kB*(273+t*beta)\
))/(lam2+lam2**2),'b--',label='TL x30');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('TL signal [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.title('(b)');
plt.tight_layout()
plt.show()
```

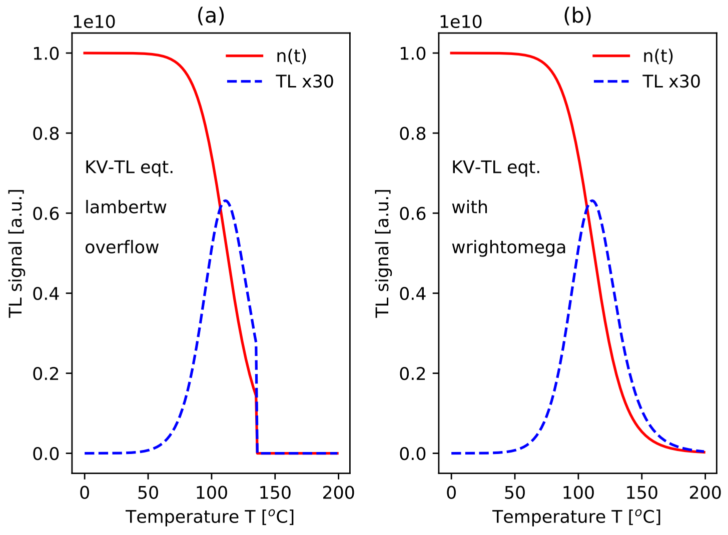


Fig. 2.16: (a) Example of numerical overflow while using the `lambertw()` function in Python to evaluate the GOT Eq.(**??**) and the corresponding analytical KV-TL Eq.(**??**). (b) The numerical overflow in (a) is overcome by using the `wrightomega()` function in Python.

Code 2.16: Plot the analytical solution of the MOK TL glow curves

```
# Plot of the analytical equations for MOK
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
kB,        E,  s,    beta =\
8.617e-5, 1, 1e12,   1
def nMOK(tmps):
    expint=kB*((273+tmps)**2.0)/E*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E)
    Ft=np.exp(g*Nd*s/beta*expint)
    return alpha*Nd/(Ft-alpha)
def TLMOK(tmps):
    expint=kB*((273+tmps)**2.0)/E*\
    np.exp(-E/(kB*(273+tmps)))*(1-2*kB*(273+tmps)/E)
    Ft=np.exp(g*Nd*s/beta*expint)
    return g*(Nd**2.0)*alpha*s*np.exp(-E/(kB*(273+tmps)))*\
    Ft/((Ft-alpha)**2.0)
tims= range(25,160)
temps=[beta*tim for tim in tims]
n10, N1, Nd= 1e10 ,1e10, 1e12
alpha, g =n10/(n10+Nd), 1/(N1+Nd)
plt.subplot(1,2, 1);
plt.plot(temps,[nMOK(x) for x in temps],'o',label='n(t)');
plt.plot(temps,[30*TLMOK(x) for x in temps],'+',
label='TL x30');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.title('(a)');
plt.ylabel('n(t) and TL (MOK)');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.text(25,.8e10,'MOK model');
plt.subplot(1,2, 2);
plt.plot(temps,[TLMOK(x) for x in temps],'+-',
label=r'$\alpha$=0.01');
n10, N1, Nd= 1e10 ,1e10, 1e10
alpha, g =n10/(n10+Nd), 1/(N1+Nd)
plt.plot(temps,[TLMOK(x) for x in temps],'^-',
label=r'$\alpha$=0.5');
n10, N1, Nd= 1e10 ,1e10, 1e8
alpha, g =n10/(n10+Nd), 1/(N1+Nd)
plt.plot(temps,[TLMOK(x) for x in temps],'o-',
label=r'$\alpha$=0.99');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.title('(b)');
plt.xlim(40,160);
plt.ylabel('TL [a.u.]');
plt.xlabel(r'Temperature T [$^{o}$C]');
plt.tight_layout()
plt.show()
```

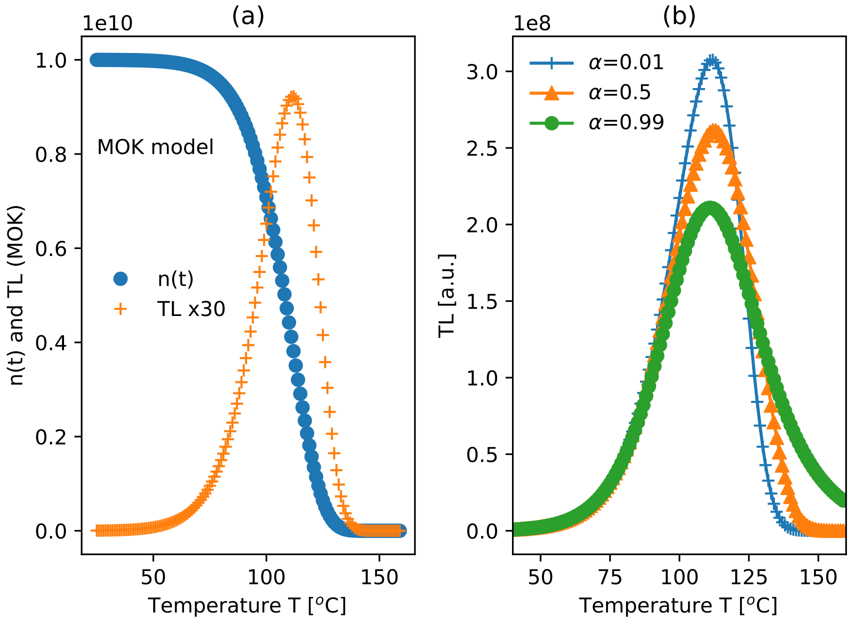


Fig. 2.17: (a) Plots of the analytical solution of the MOK model, Eqs.(**??**) and (**??**) for the parameters given in the text. The MOK parameter α = 0.0099 and the TL glow curve has first order characteristics. (b) Plot of the analytical MOK Eq.(**??**) for three values of the concentration of competitor trap $N_d = 10^{12}$, 10^{10}, 10^8 cm^{-3}. The values of α =0.01, 0.5, 0.99 correspond approximately to $b \simeq 1$, $b \simeq 1.5$ and $b \simeq 2$.