## Chapter 9

# OSL FROM DELOCALIZED TRANSITIONS: DATA ANALYSIS

**Code 9.1: Using the FOK-CW equation for CCDA of CW signals**

```
#  deconvolution with FOK-CW equation KST4 feldspar CW-IRSL Data
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import warnings
data = np.loadtxt('KST4ph300IR.txt')
x_data,y_data=data[:,0][1:800], data[:,1][1:800]
y_data=y_data/max(y_data)
def FOKCW(t, A,tau):
    CW=A*np.exp(-t/tau)
    return CW
def total_CW(t, *inis):
    u=np.array([0 for i in range(len(x_data))])
    As, taus=    inis[0:nPks], inis[nPks:2*nPks]
    bgd=inis[-1]
    for i in range(nPks):
        u=u+FOKCW(t,As[i],taus[i])
    u=u+bgd
    return u
P=int(max(x_data))
t = np.linspace(0, P, P)
nPks=2
```

```
inis=[.5,10,.5,50,.01]
params, cov = optimize.curve_fit(total_CW,\
x_data,y_data,p0=inis,maxfev=10000)
plt.scatter(x_data, y_data,c='r',label='KST4 feldspar CW-IRSL ');
plt.plot(x_data, total_CW(x_data,
 *params),c='black',label='FOK-CW equation',linewidth=1);
for i in range(0,nPks):
    CWi=FOKCW(t, params[i],params[nPks+i]);
    plt.plot(t,CWi);
plt.plot(t,[params[-1]]*len(t));
leg = plt.legend();
leg.get_frame().set_linewidth(0.0);
plt.ylabel('CW-IRSL [a.u.]');
plt.xlabel(r'Stimulation time [s]');
res=total_CW(x_data, *params)-y_data
FOM=100*np.sum(abs(res))/np.sum(y_data)
print('FOM=',round(FOM,1),' %')
As=[round(x,3) for x in params[0:nPks]]
taus=[round(x,1) for x in params[nPks:2*nPks]]
dAs=[round(np.sqrt(cov[x][x]),3) for x in range(0,nPks)]
dtaus=[round(np.sqrt(cov[x][x]),2) for x in\
range(nPks,2*nPks)]
bgd=round(params[-1],3)
myTable = PrettyTable([ "A (a.u.)","dA",\
'tau (s)','dtau (s)','bgd'])
myTable.add_row([As[0],dAs[0],taus[0],dtaus[0],bgd]);
myTable.add_row([As[1],dAs[1],taus[1],dtaus[1],' ']);
print(myTable)
plt.show();
```

```
FOM= 10.0  %
+----------+-------+---------+----------+-------+
| A (a.u.) |   dA  | tau (s) | dtau (s) |  bgd  |
+----------+-------+---------+----------+-------+
|  0.925   | 0.004 |   5.9   |   0.05   | 0.007 |
|  0.192   | 0.003 |   39.8  |   0.58   |       |
+----------+-------+---------+----------+-------+
```

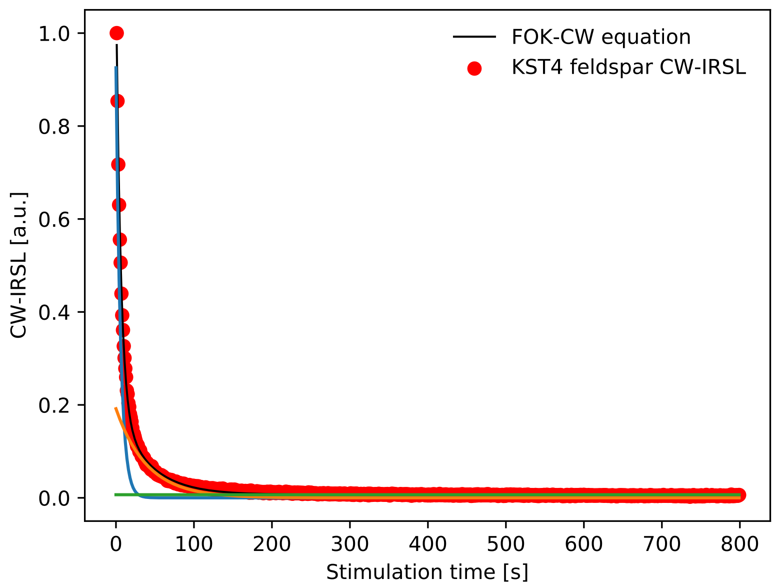



**Fig. 9.1:** Example of fitting a CW-IRSL signal with two exponential FOK-CW components plus a constant background. The CW-IRSL data are from a freshly irradiated aliquot of feldspar sample KST4 (Pagonis et al. [1]).

---

**Code 9.2: Using the KV-CW equation to fit a CW-IRSL signal**

```
#  deconvolution with KV-CW equation KST4 feldspar CW-IRSL Data
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
from scipy.special import wrightomega
import warnings
data = np.loadtxt('KST4ph300IR.txt')
x_data,y_data=data[:,0][1:800], data[:,1][1:800]
```

```
y_data=y_data/max(y_data)
def KVCW(t, A,c,sprime):
    zCW=(1/c)-np.log(c)+sprime*t
    lam=wrightomega(zCW)
    CW=A/(lam+lam**2)
    return CW
def total_CW(t, *inis):
    u=np.array([0 for i in range(len(x_data))])
    As, cs ,sprimes=    inis[0:nPks], inis[nPks:2*nPks],\
    inis[2*nPks:3*nPks]
    bgd=inis[-1]
    for i in range(nPks):
        u=u+KVCW(t,As[i],cs[i],sprimes[i])
    u=u+bgd
    return u
t = np.linspace(0, 200, 200)
nPks=1
A=[max(y_data)]*nPks
lowA, highA=[0.01*x for x in A], [200 for x in A]
c=[1]*nPks
lowc, highc= [0.001*x for x in c], [1e4*x for x in c]
sprime=[.1]*nPks
lowsprime, highsprime=  [0.001*x for x in sprime],\
[100*x for x in sprime]
bgd, lowbgd, highbgd=[.1,0,.15]
inis=A+c+sprime+[bgd]
lowbnds=lowA+lowc+lowsprime+[lowbgd]
highbnds=highA+highc+highsprime+[highbgd]
params, cov = optimize.curve_fit(total_CW,\
x_data,y_data,p0=inis,bounds=(lowbnds,highbnds),maxfev=10000)
plt.scatter(x_data, y_data,c='r',label='KST4 feldspar CW-IRSL ');
plt.plot(x_data, total_CW(x_data,
 *params),c='black',label='KV-CW equation',linewidth=1);
for i in range(0,nPks):
    CWi=KVCW(t, params[i],params[nPks+i], params[2*nPks+i]);
    plt.plot(t,CWi);
plt.plot(t,[params[-1]]*len(t));
leg = plt.legend();
leg.get_frame().set_linewidth(0.0);
plt.ylabel('CW-IRSL [a.u.]');
plt.xlabel(r'Stimulation time [s]');
res=total_CW(x_data, *params)-y_data
FOM=100*np.sum(abs(res))/np.sum(y_data)
print('FOM=',round(FOM,1),' %')
```

```
As=[round(x,2) for x in params[0:nPks]]
cs=[round(x,2) for x in params[nPks:2*nPks]]
sprimes=[round(x,2) for x in params[2*nPks:3*nPks]]
dAs=[round(np.sqrt(cov[x][x]),2) for x in range(nPks)]
dcs=[round(np.sqrt(cov[x][x]),2) for x in range(nPks,2*nPks)]
dsprimes=[round(np.sqrt(cov[x][x]),2) for x in\
range(2*nPks,3*nPks)]
bgd=round(params[-1],2)
myTable = PrettyTable([ "A (a.u.)","dA",\
'c','dc','s (s^-1)','ds (s^-1)'])
myTable.add_row([As[0],dAs[0],cs[0],dcs[0],sprimes[0],\
dsprimes[0]]);
print(myTable)
plt.show();
```

```
FOM= 11.5  %
+----------+--------+------+------+----------+-----------+
| A (a.u.) |   dA   |  c   |  dc  | s (s^-1) | ds (s^-1) |
+----------+--------+------+------+----------+-----------+
|  200.0   | 140.93 | 0.08 | 0.03 |   1.18   |    0.4    |
+----------+--------+------+------+----------+-----------+
```

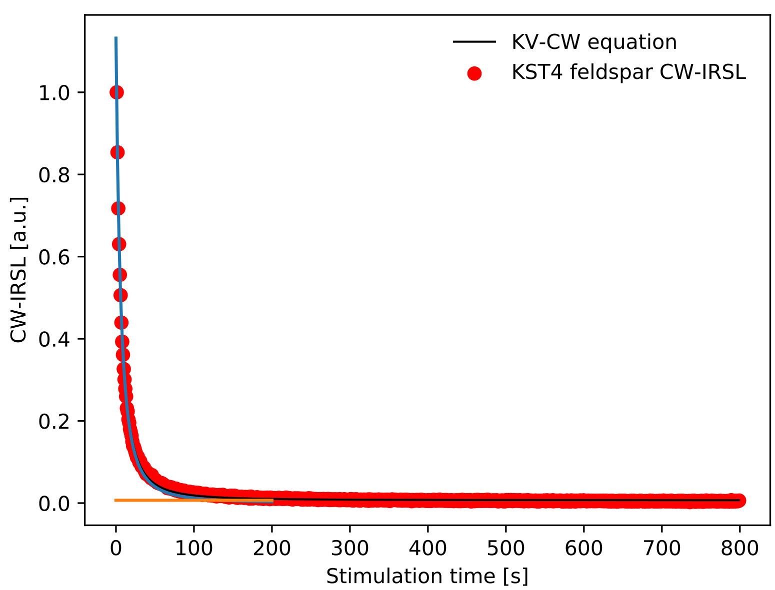


**Fig. 9.2:** Example of fitting the CW-OSL signal with one KV-CW component plus a constant background. The CW-IRSL data are from an aliquot of feldspar sample KST4 (Pagonis et al. [1])

---

**Code 9.3: Using the MOK-CW equation to fit a CW-IRSL signal**

```
# KST4 CW-IRSL deconvolution with MOK-CW equation
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('KST4ph300IR.txt')
x_data,y_data=data[:,0][1:800], data[:,1][1:800]
y_data=y_data/max(y_data)
def MOKCW(t, A,b,sprime, bgd):
    F=np.exp(sprime*t)
    ITL=A*F/((F-b)**2)
    return ITL
def total_CW(t, *inis):
    u=np.array([0 for i in range(len(x_data))])
    As, bs ,sprimes=inis[0:nPks], inis[nPks:2*nPks],\
    inis[2*nPks:3*nPks],
    bgd=inis[-1]
    for i in range(nPks):
        u=u+MOKCW(t,As[i],bs[i],sprimes[i],bgd)
    ubgd=bgd
    u=u+ubgd
    return u
P=int(max(x_data))
t = np.linspace(0, P, P)
nPks=1
A=[max(y_data)/2]*nPks
lowA, highA=[0.01*x for x in A], [2*x for x in A]
b=[0.1]*nPks
lowb, highb= [0.001 for x in b], [1 for x in b]
sprime=[.01]*nPks
```

```
lowsprime, highsprime=  [1e-4 for x in sprime], \
[1e4 for x in sprime]
bgd=.01
lowbgd,highbgd=0,.3
inis=A+b+sprime+[bgd]
lowbnds=lowA+lowb+lowsprime+[lowbgd]
highbnds=highA+highb+highsprime+[highbgd]
params, cov = optimize.curve_fit(total_CW,\
x_data,y_data,p0=inis,bounds=(lowbnds,highbnds),maxfev=10000)
res=total_CW(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
bgd=params[-1]
As=[round(x,3) for x in params[0:nPks]]
cs=[round(x,3) for x in params[nPks:2*nPks]]
sprime=[round(x,3) for x in params[2*nPks:3*nPks]]
dAs=[round(np.sqrt(cov[x][x]),3) for x in range(nPks)]
dcs=[round(np.sqrt(cov[x][x]),3) for x in range(nPks,2*nPks)]
dsprime=[round(np.sqrt(cov[x][x]),3) \
for x in range(2*nPks,3*nPks)]
myTable = PrettyTable([ "A (a.u.)","dA (a.u)",\
'alpha','dalpha',"s' (s^-1)","ds'"])
myTable.add_row([As[0],dAs[0],cs[0],dcs[0],sprime[0],\
dsprime[0]]);
print('FOM %=',FOM)
print(myTable)
```

```
FOM %= 10.86
+----------+----------+-------+--------+-----------+-------+
| A (a.u.) | dA (a.u) | alpha | dalpha | s' (s^-1) |  ds'  |
+----------+----------+-------+--------+-----------+-------+
|  0.005   |  0.005   | 0.934 | 0.035  |   0.006   | 0.003 |
+----------+----------+-------+--------+-----------+-------+
```

---

**Code 9.4: Fitting CW-OSL signal with GOK-CW equation for KST4 feldspar**

```
# KST4 CW-IRSL deconvolution with GOK-CW equation
from scipy import optimize
import numpy as np
```

```
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('KST4ph300IR.txt')
x_data,y_data=data[:,0][1:800], data[:,1][1:800]
y_data=y_data/max(y_data)
def GOKCW(t, A,b,sprime, bgd):
    ITL=A*(1+sprime*(b-1)*t)**(-b/(b-1))
    return ITL
def total_CW(t, *inis):
    u=np.array([0 for i in range(len(x_data))])
    As, bs ,sprimes=inis[0:nPks], inis[nPks:2*nPks],\
    inis[2*nPks:3*nPks]
    bgd=inis[-1]
    for i in range(nPks):
        u=u+GOKCW(t,As[i],bs[i],sprimes[i],bgd)
    ubgd=bgd
    u=u+ubgd
    return u
P=int(max(x_data))
t = np.linspace(0, P,P)
nPks=1
inis=[1,1.5,.1,.01]
params, cov = optimize.curve_fit(total_CW,\
x_data,y_data,p0=inis,maxfev=10000)
bgd=params[-1]
res=total_CW(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
print('FOM=',FOM)
As=[round(x,2) for x in params[0:nPks]]
bs=[round(x,2) for x in params[nPks:2*nPks]]
sprime=[round(x,3) for x in params[2*nPks:3*nPks]]
bgd=round(params[-1],3)
dAs=[round(np.sqrt(cov[x][x]),3) for x in range(nPks)]
dbs=[round(np.sqrt(cov[x][x]),3) for x in range(nPks,2*nPks)]
dsprime=[round(np.sqrt(cov[x][x]),4)\
for x in range(2*nPks,3*nPks)]
myTable = PrettyTable([ "A (a.u.)","dA (a.u)",\
'b','db',"s' (s^-1)","ds'"])
myTable.add_row([As[0],dAs[0],bs[0],dbs[0],sprime[0],\
dsprime[0]]);
print(myTable)

  FOM= 4.07
```

```
+----------+----------+------+------+-----------+--------+
| A (a.u.) | dA (a.u) |  b   |  db  | s' (s^-1) |  ds'   |
+----------+----------+------+------+-----------+--------+
|   1.22   |  0.002   | 2.83 | 0.02 |   0.074   | 0.0003 |
+----------+----------+------+------+-----------+--------+
```

---

**Code 9.5: Fitting CW-IRSL signal with KP-CW equation for KST4 feldspar**

```
# KST4 CW-IRSL deconvolution with GOK-CW equation
# CW-IRSL data fitted with KP-CW equation
from scipy import optimize
from sympy import *
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('KST4ph300IR.txt')
x_data,y_data=data[:,0][1:800], data[:,1][1:800]
y_data=y_data/max(y_data)
plt.subplot(2,1, 1);
def test_func(x, imax_fit,rho_fit, A_fit,bgd_fit):
    return imax_fit*np.exp (-rho_fit*(np.log(1 + A_fit*x))\
** 3.0)*(np.log(1+A_fit*x)**2.0)/(1+x*A_fit)+bgd_fit
params, cov = optimize.curve_fit(test_func,\
x_data, y_data)
drho= round(np.sqrt(cov[1][1]),5)
dA = round(np.sqrt(cov[2][2]),2)
dimax = round(np.sqrt(cov[0][2]),2)
plt.scatter(np.log(x_data), y_data, label='Experiment');
plt.plot(np.log(x_data), test_func(x_data, *params[0:4]),
         label='KP-CW equation');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('CW-IRSL signal [a.u.]');
plt.xlabel('ln(Time)');
# plt.text(2, 7000,'KST4 feldspar');
plt.subplot(2,1, 2);
```

```
plt.plot(np.log(x_data),test_func(x_data, *params[0:4])-\
y_data,"o",label='Residuals');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('Residuals');
plt.xlabel('ln(Time)');
plt.ylim(-.1,.1);
plt.tight_layout()
imax,rho, A, bgd=int(params[0]),round(params[1],5),\
round(params[2],2),round(params[-1],3)
res=test_func(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
myTable=PrettyTable(["A",'dA', "rho",  "d(rho)",\
"s'(s^-1)","ds'",'bgd',"FOM"]);
myTable.add_row([imax,dimax,rho,drho, A, dA,bgd,FOM]);
print(myTable)
plt.show()
```

```
+---+------+---------+--------+----------+------+-------+-----+
| A |  dA  |   rho   | d(rho) | s'(s^-1) | ds'  |  bgd  | FOM |
+---+------+---------+--------+----------+------+-------+-----+
| 1 | 0.01 | 0.00763 | 6e-05  |   5.94   | 0.04 | 0.005 | 5.1 |
+---+------+---------+--------+----------+------+-------+-----+
```

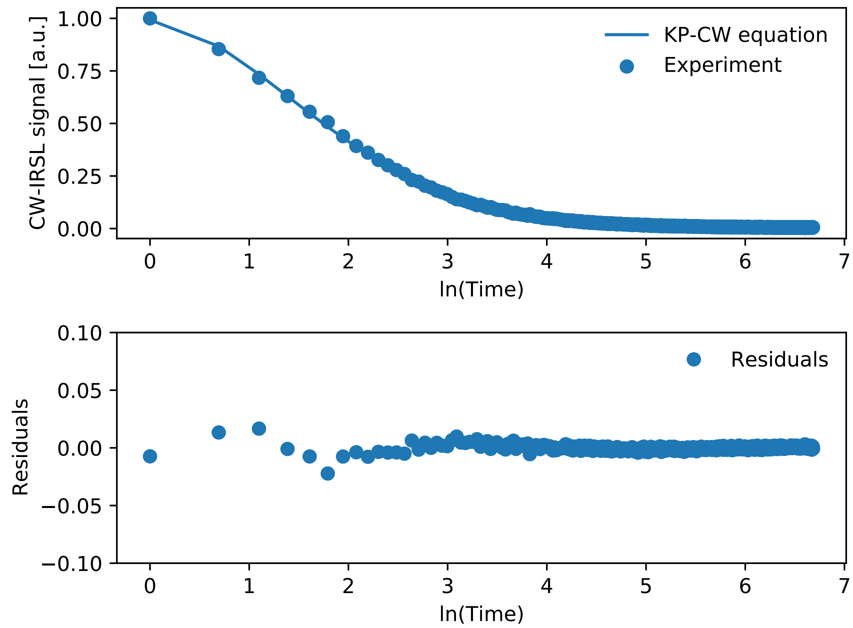



**Fig. 9.3:** Example of fitting a CW-IRSL signal with the KP-CW equation with one component and without a constant background. The CW-IRSL data are from a freshly irradiated aliquot of feldspar sample KST4 (Pagonis et al. [1])

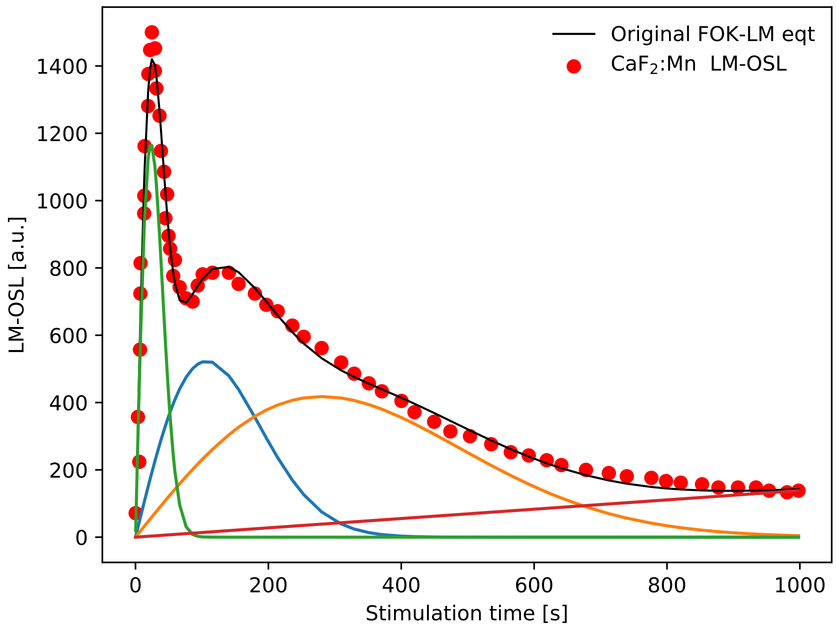
**Code 9.6: Analysis of 3-component LM-OSL signal**

```
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('CaF2LMOSL.txt')
x_data,y_data = data[:, 0], data[:, 1]
def LM(x,N,tau):
    u=np.abs(N)*(x/P)*(np.exp(-(x**2.0)\
```

```
    /(2*P*abs(tau))))
    return u
def total_FOKLM(x, *inis):
    u=np.array([0 for i in range(len(x_data))])
    Ns, taus =    inis[0:nPks], inis[nPks:2*nPks]
    for i in range(nPks):
        u=u+LM(x,Ns[i],taus[i])
    u=u+bgd*x/P
    return u
nPks= 3
P=int(max(x_data))
t=np.linspace(0,P,P)
inis=[1400,1,800,.1,500,.01]
bgd=y_data[-1]
params,cov =optimize.curve_fit(total_FOKLM,x_data,\
y_data,p0=inis)
params, cov = optimize.curve_fit(total_FOKLM,\
x_data,y_data,p0=inis,maxfev=10000)
plt.scatter(x_data, y_data,c='r',label=r'CaF$_2$:Mn  LM-OSL');
plt.plot(x_data, total_FOKLM(x_data,
 *params),c='black',label='Original FOK-LM eqt',linewidth=1);
totalArea=sum(total_FOKLM(x_data,  *params))
sums,pc=[0]*nPks, [0]*nPks
for i in range(0,nPks):
    FOKLMi=LM(x_data, params[i],params[nPks+i]);
    sums[i]=np.sum(FOKLMi)
    plt.plot(x_data,FOKLMi);
plt.plot(t,bgd*t/P);
for j in range(nPks):
    pc[j]=round(100*sums[j]/totalArea,1)
pcbgd=round(100*sum(bgd*x_data/P)/totalArea)
leg = plt.legend();
leg.get_frame().set_linewidth(0.0);
plt.ylabel('LM-OSL [a.u.]');
plt.xlabel(r'Stimulation time [s]');
res=total_FOKLM(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),1)
print('FOM=',FOM,' %')
plt.show();
Ns=[round(x,1) for x in params[0:nPks]]
taus=[round(x,2) for x in params[nPks:2*nPks]]
dN=[round(np.sqrt(cov[x][x]),1) for x in range(3)]
dtaus=[round(np.sqrt(cov[x][x]),2) for x in range(3,6)]
myTable = PrettyTable([ "N (a.u.)","dN (a.u)",\
```

```
'tau (s)',"dtau (s)","Area [%]"])
for j in range(nPks):
    myTable.add_row([Ns[j],dN[j],taus[j],dtaus[j],pc[j]]);
myTable.add_row(['','','','','bgd='+str(pcbgd)+'%']);
print(myTable)

  FOM= 5.2  %
```




```
  +----------+----------+---------+----------+----------+
  | N (a.u.) | dN (a.u) | tau (s) | dtau (s) | Area [%] |
  +----------+----------+---------+----------+----------+
  |  7999.8  |  497.8   |  11.58  |   1.29   |   24.9   |
  |  2446.1  |  313.4   |  79.06  |   7.63   |   24.8   |
  | 82081.2  |  1753.6  |   0.55  |   0.02   |   43.4   |
  |          |          |         |          | bgd=7.0% |
  +----------+----------+---------+----------+----------+
```

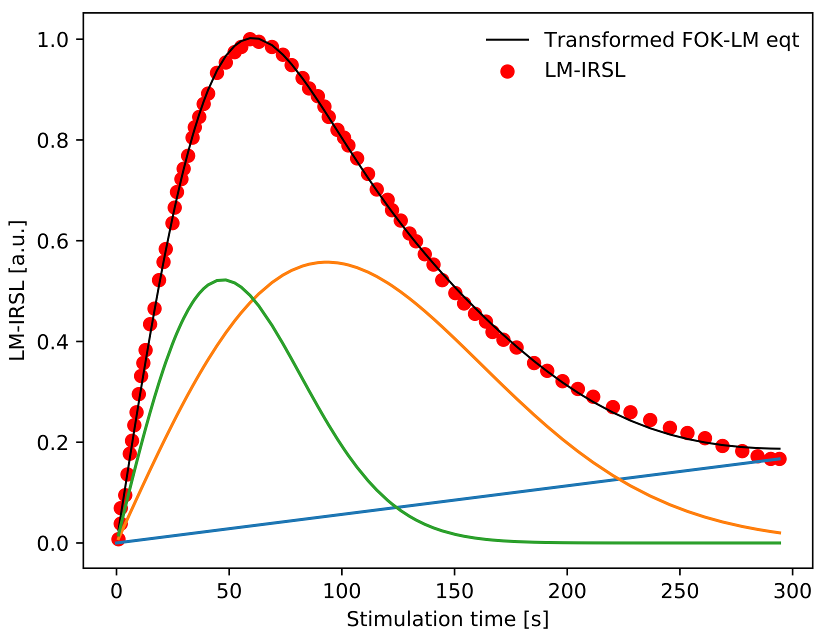
**Fig. 9.4:** Example of analyzing an LM-OSL signal from the dosimetric material $CaF_2$:N with three first order components, using the package *numOSL* (Kitis et al. [2]).

**Code 9.7: Analysis of 2-component LM-OSL signal usig FOK-LM eqt**

```
#Analysis of 2-component LM-OSL signal usig FOK-LM eqt
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('K120.txt')
x_data,y_data = data[:, 0], data[:, 1]
y_data =y_data/max(y_data)
def LM(x_data,N,xmax):
    u=1.6487*np.abs(N)*(x_data/abs(xmax))*(np.exp(-(x_data**\
    2.0)/(2*(abs(xmax)**2.0))))
    return u
def total_FOKLM(t, *inis):
    u=np.array([0 for i in range(len(x_data))])
    Ns, xmaxs =    inis[0:nPks], inis[nPks:2*nPks]
    for i in range(nPks):
        u=u+LM(t,Ns[i],xmaxs[i])
    u=u+bgd*t/P
    return u
nPks= 2
P=int(max(x_data))
t=np.linspace(0,P,P)
inis=[50,3,100,1]
bgd=y_data[-1]
params,cov =optimize.curve_fit(total_FOKLM,x_data,\
y_data,p0=inis)
params, cov = optimize.curve_fit(total_FOKLM,\
x_data,y_data,p0=inis,maxfev=10000)
plt.scatter(x_data, y_data,c='r',label=r'LM-IRSL');
plt.plot(x_data, total_FOKLM(x_data,
 *params),c='black',label='Transformed FOK-LM eqt',linewidth=1);
plt.plot(t,bgd*t/P);
leg = plt.legend();
leg.get_frame().set_linewidth(0.0);
plt.ylabel('LM-IRSL [a.u.]');
plt.xlabel(r'Stimulation time [s]');
for i in range(0,nPks):
    FOKLMi=LM(x_data, params[i],params[nPks+i]);
    plt.plot(x_data,FOKLMi);
```

```
res=total_FOKLM(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),1)
print('FOM=',FOM,' %')
plt.show();
Ns=[round(x,2) for x in params[0:nPks]]
xmaxs=[round(abs(x),2) for x in params[nPks:2*nPks]]
dN=[round(np.sqrt(cov[x][x]),2) for x in range(nPks)]
dxmaxs=[round(np.sqrt(cov[x][x]),2) for x in  range(nPks,2*nPks)]
myTable = PrettyTable([ "Im (a.u.)","dIm (a.u)",\
'tm (s)',"dtm (s)"])
for j in range(nPks):
    myTable.add_row([Ns[j],dN[j],xmaxs[j],dxmaxs[j]]);
print(myTable)
```

```
FOM= 1.5  %
```




```
+-----------+-----------+--------+---------+
| Im (a.u.) | dIm (a.u) | tm (s) | dtm (s) |
+-----------+-----------+--------+---------+
|    0.56   |    0.01   | 93.33  |   0.83  |
|    0.52   |    0.01   | 47.14  |   0.5   |
+-----------+-----------+--------+---------+
```

**Fig. 9.5:** Example of analyzing an LM-IRSL signal from a K feldspar with two first order components plus a linearly increasing background, using the transformed FOK-LM equation. For experimental details see Bulur and Göksu [3]

**Code 9.8: Fitting LM-OSL signal with KV-LM equation for feldspar**

```
# LM-OSL deconvolution with KV-LM equation plus background
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
from scipy.special import wrightomega
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('K120.txt')
x_data,y_data = data[:, 0], data[:, 1]
y_data=y_data/max(y_data)
def KVLMOSL(t, A,c,sprime):
    zLMOSL=(1/c)-np.log(c)+sprime*t**2/(2*P)
    lam=wrightomega(zLMOSL)
    LMOSL=A*t/(lam+lam**2)
    return LMOSL
def total_LMOSL(t, *inis):
    u=np.array([0 for i in range(len(x_data))])
    As, cs ,sprimes=inis[0:nPks], inis[nPks:2*nPks],\
    inis[2*nPks:3*nPks]
    for i in range(nPks):
        u=u+KVLMOSL(t,As[i],cs[i],sprimes[i])
    ubgd=bgd*t/300
    u=u+ubgd
    return u
P=int(max(x_data))
nPks=1
t = np.linspace(0, P, P)
bgd=y_data[-1]-.07       # adjusted parameter for better fit
inis=[1,.1,1e-4]
params, cov = optimize.curve_fit(total_LMOSL,\
x_data,y_data,p0=inis,maxfev=10000)
plt.scatter(x_data, y_data,c='r',label=r'LM-IRSL');
plt.plot(x_data, total_LMOSL(x_data, *params),c='black',\
```

```
label='KV-LM equation',linewidth=1);
plt.plot(t,bgd*t/P);
plt.plot(t,KVLMOSL(t,*params));
leg = plt.legend();
leg.get_frame().set_linewidth(0.0);
plt.ylabel('LM-IRSL [a.u.]');
plt.xlabel(r'Stimulation time [s]');
res=total_LMOSL(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
print('FOM=',FOM, '%')
As=[round(x,3) for x in params[0:nPks]]
cs=[round(x,2) for x in params[nPks:2*nPks]]
sprimes=[round(x,4) for x in params[2*nPks:3*nPks]]
dAs=[round(np.sqrt(cov[x][x]),2) for x in range(nPks)]
dcs=[round(np.sqrt(cov[x][x]),2) for x in range(nPks,2*nPks)]
dsprimes=[round(np.sqrt(cov[x][x]),5) for x in\
range(2*nPks,3*nPks)]
myTable = PrettyTable([ "A (a.u.)","dA",\
'c','dc','s (s^-1)','ds'])
myTable.add_row([As[0],dAs[0],cs[0],dcs[0],sprimes[0],\
 dsprimes[0]]);
print(myTable)
plt.show()
```

```
FOM= 1.28 %
+----------+------+------+------+----------+---------+
| A (a.u.) |  dA  |  c   |  dc  | s (s^-1) |    ds   |
+----------+------+------+------+----------+---------+
|  2.467   | 1.09 | 0.11 | 0.03 |  0.5401  | 0.11252 |
+----------+------+------+------+----------+---------+
```

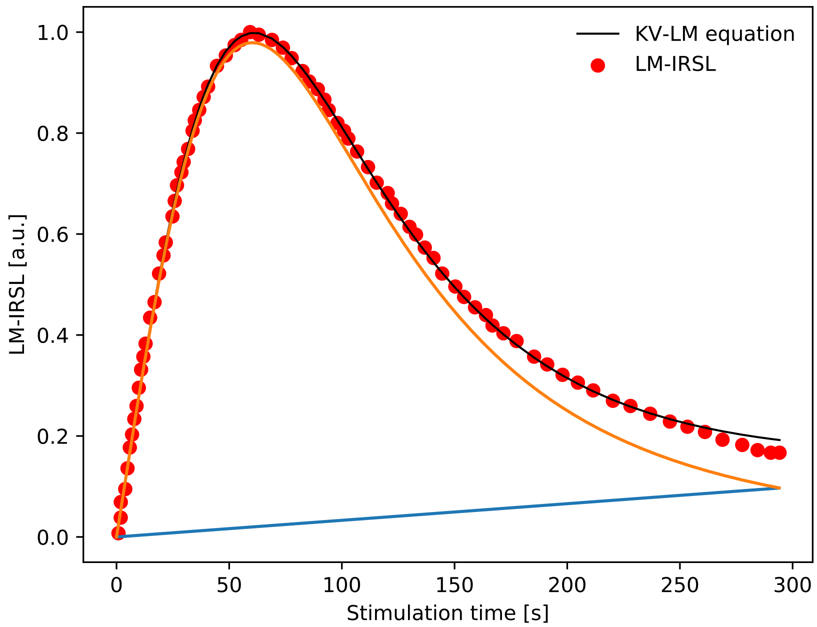



**Fig. 9.6:** Example of analyzing an LM-IRSL signal from a K feldspar with a single component using the KV-LM Eq.(**??**), plus a linearly increasing background. For experimental details see Bulur and Göksu [3]

---

**Code 9.9: CCDA with MOK-LM equation for feldspar**

```
# Deconvolution with MOK-LM equation, fixed bgd
from scipy import optimize
import numpy as np
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('K120.txt')
x_data,y_data = data[:, 0], data[:, 1]
```

```
y_data=y_data/max(y_data)
def MOKLM(t, A,b,sprime):
    F=np.exp(sprime*t**2/(2*P))
    LM=A*t/P*F/((F-b)**2)
    LM=LM+bgd*t/P
    return LM
P=int(max(x_data))
t = np.linspace(0, P, P)
inis=[A,b,sprime]=[.5, .1, 1e-4]
bgd=y_data[-1]
params, cov = optimize.curve_fit(MOKLM,\
x_data,y_data,p0=inis,maxfev=10000)
res=MOKLM(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
res=MOKLM(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
print('FOM=',FOM,' %')
[As,cs,sprimes]= [round(params[x],3) for x in range(3)]
[dAs,dcs,dsprimes]= [round(np.sqrt(cov[x][x]),3)\
for x in range(3)]
myTable = PrettyTable([ "A (a.u.)","dA",\
'alpha','dalpha','s (s^-1)','ds'])
myTable.add_row([As,dAs,cs,dcs,sprimes,dsprimes]);
print(myTable)
```

```
FOM= 1.44  %
+----------+-------+-------+--------+----------+-------+
| A (a.u.) |   dA  | alpha | dalpha | s (s^-1) |   ds  |
+----------+-------+-------+--------+----------+-------+
|  1.127   | 0.063 | 0.633 | 0.011  |  0.024   | 0.001 |
+----------+-------+-------+--------+----------+-------+
```

---

**Code 9.10: CCDA with transformed MOK-LM equation**

```
# Deconvolution with transformed MOK-LM equation, fixed bgd
# Deconvolution with MOK-LM equation, fixed bgd
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
```

```
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('K120.txt')
x_data,y_data = data[:, 0], data[:, 1]
y_data=y_data/max(y_data)
# b in this code represents the MOK parameter alpha
def MOKLM(t, Imax,b,tmax):
    Fm=1.6476-1.0012*b+0.357*b**2
    F=np.exp((t/tmax)**2*(Fm-b)/(2*(Fm+b)))
    LM=Imax*t/tmax*(((Fm-b)/(F-b))**2)*F/Fm
    LM=LM+bgd*t/P
    return LM
P=int(max(x_data))
t = np.linspace(0, P, P)
bgd=y_data[-1]
inis=[Imax,b,tmax]=[1, .8, 60]
params, cov = optimize.curve_fit(MOKLM,\
x_data,y_data,p0=inis,maxfev=10000)
res=MOKLM(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
print('FOM=',FOM,' %')
[Imaxs,cs,tmaxs]= [round(params[x],3) for x in range(3)]
[dImaxs,dcs,dtmaxs]= [round(np.sqrt(cov[x][x]),3) for x in \
range(3)]
myTable = PrettyTable([ "Im (a.u.)","dIm",\
'alpha','dalpha','tm (s)','dtm'])
myTable.add_row([Imaxs,dImaxs,cs,dcs,tmaxs,dtmaxs]);
print(myTable)
```

```
FOM= 1.44  %
+-----------+-------+-------+--------+--------+-------+
| Im (a.u.) |  dIm  | alpha | dalpha | tm (s) |  dtm  |
+-----------+-------+-------+--------+--------+-------+
|   0.965   | 0.002 | 0.633 | 0.011  | 59.86  | 0.192 |
+-----------+-------+-------+--------+--------+-------+
```

---

**Code 9.11: Fitting LM-OSL signal with GOK-LM equation for feldspar**

```
# LM-OSL deconvolution with original GOK-LM equation
# Deconvolution with GOK-LM equation, fixed bgd
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('K120.txt')
x_data,y_data = data[:, 0], data[:, 1]
y_data=y_data/max(y_data)
def GOKLM(t, A,b,sprime):
    LM=A*t*(1+(b-1)*sprime*t**2/(2*P))**(-b/(b-1))
    LM=LM+bgd*t/P
    return LM
P=int(max(x_data))
t = np.linspace(0, P, P)
inis=[A,b,sprime]=[.5, 1.5, 1e-4]
bgd=y_data[-1]
params, cov = optimize.curve_fit(GOKLM,\
x_data,y_data,p0=inis,maxfev=10000)
res=GOKLM(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
print('FOM=',FOM,' %')
[As,cs,sprimes]= [round(params[x],3) for x in range(3)]
[dAs,dcs,dsprimes]= [round(np.sqrt(cov[x][x]),4) \
for x in range(3)]
myTable = PrettyTable([ "A (a.u.)","dA",\
'b','db','s (s^-1)','ds'])
myTable.add_row([As,dAs,cs,dcs,sprimes,dsprimes]);
print(myTable)
```

```
FOM= 2.46  %
+----------+--------+-------+--------+----------+--------+
| A (a.u.) |   dA   |   b   |   db   | s (s^-1) |   ds   |
+----------+--------+-------+--------+----------+--------+
|  0.028   | 0.0002 | 1.603 | 0.0267 |  0.062   | 0.0005 |
+----------+--------+-------+--------+----------+--------+
```

9.6.

---

**Code 9.12: CCDA with transformed GOK-LM equation**

```
# Deconvolution with transformed GOK-LM equation plus bgd
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('K120.txt')
x_data,y_data = data[:, 0], data[:, 1]
y_data=y_data/max(y_data)
def GOKLM(t, Imax,b,tmax):
    LM=Imax*t/tmax*(((b-1)/b)*(t**2/(tmax**2))/2+(b+1)/\
(2*b))**(b/(1-b))
    LM=LM+bgd*t/P
    return LM
P=int(max(x_data))
t = np.linspace(0, P, P)
bgd=y_data[-1]
inis=[Imax,b,tmax]=[1, 1.5, 60]
params, cov = optimize.curve_fit(GOKLM,\
x_data,y_data,p0=inis,maxfev=10000)
res=GOKLM(x_data, *params)-y_data
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
print('FOM=',FOM,' %')
[Imaxs,cs,tmaxs]= [round(params[x],3) for x in range(3)]
[dImaxs,dcs,dtmaxs]= [round(np.sqrt(cov[x][x]),3) for x in \
range(3)]
myTable = PrettyTable([ "Im (a.u.)","dIm",\
'b','db','tm (s)','dtm'])
myTable.add_row([Imaxs,dImaxs,cs,dcs,tmaxs,dtmaxs]);
print(myTable)
```

```
FOM= 2.46  %
+-----------+-------+-------+-------+--------+-------+
| Im (a.u.) |  dIm  |   b   |   db  | tm (s) |  dtm  |
+-----------+-------+-------+-------+--------+-------+
|   0.972   | 0.004 | 1.603 | 0.027 | 60.522 | 0.327 |
+-----------+-------+-------+-------+--------+-------+
```

---

**Code 9.13: CCDA with KP-LM equation**

```
# Deconvolution with the KP-LM equation, fixed bgd
from scipy import optimize
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('K120.txt')
x_data,y_data=data[:,0], data[:,1]
y_data=y_data/max(y_data)
def KVLMOSL(x, A,sprime,rho):
    F=np.log(1+sprime*x**2/(2*P))
    LMOSL= abs(A)*np.exp (-rho*\
F** 3.0)*(F**2.0)/(1+sprime*x**2/(2*P))
    return LMOSL
def total_LMOSL(x, *inis):
    u=np.array([0 for i in range(len(x_data))])
    As, sprimes=inis[0:nPks], inis[nPks:2*nPks]
    rho=inis[-1]
    for i in range(nPks):
        u=u+KVLMOSL(x,As[i],sprimes[i],rho)
    return u
nPks=2
P=int(max(x_data))
inis=[.5,10,.5,100,.005]
params, cov = optimize.curve_fit(total_LMOSL,\
x_data, y_data,p0=inis)
plt.scatter(x_data, y_data, label='Experiment');
plt.plot(x_data, total_LMOSL(x_data, *params),
         label='KP-LM equation',linewidth=3);
for i in range(0,nPks):
    FOKLMi=KVLMOSL(x_data, params[i],params[nPks+i],params[-1]);
    plt.plot(x_data,FOKLMi);
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
plt.ylabel('LM-IRSL signal [a.u.]');
plt.xlabel('Time [s]');
leg = plt.legend()
leg.get_frame().set_linewidth(0.0)
A1,sprime1,A2,sprime2,rho=[round(params[x],3) for x in range(5)]
dA1,dsprime1,dA2,dsprime2,drho=[round(np.sqrt(cov[x][x]),4)\
for x in range(5)]
res=total_LMOSL(x_data, *params)-y_data
```

```
FOM=round(100*np.sum(abs(res))/np.sum(y_data),2)
myTable=PrettyTable(["A",'dA', "rho",  "d(rho)",\
"s'(s^-1)","ds","FOM"]);
myTable.add_row([A1,dA1,rho,drho, sprime1, dsprime1,FOM]);
myTable.add_row([A2,dA2,' ',' ', sprime2, dsprime2,' ']);
print(myTable)
plt.show()
```

```
+-------+--------+-------+--------+----------+--------+------+
|   A   |   dA   |  rho  | d(rho) | s'(s^-1) |   ds   | FOM  |
+-------+--------+-------+--------+----------+--------+------+
| 1.827 | 0.0151 | 0.008 | 0.0004 |  0.668   | 0.0176 | 2.32 |
| 0.785 | 0.0179 |       |        |  14.809  | 0.8542 |      |
+-------+--------+-------+--------+----------+--------+------+
```

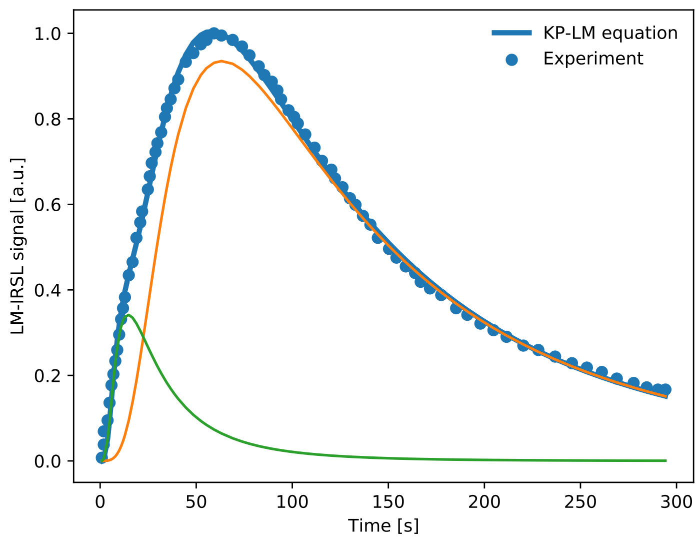



**Fig. 9.7:** Example of CCDA with the KP-LM equation and $N$ =2 components and no background component. The LM-IRSL data are from a K feldspar sample (Bulur and Göksu [3])

---

**Code 9.14: Transforming CW-IRSL signals into peak-shaped signals**

```
#Dependence of CW-OSL signal on power 10-90% in Quartz
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
data = np.loadtxt('Quartz_110OSL_polymeris.txt')
lps=np.arange(1,10,1)
plt.subplot(1,2,1);
for i in lps:
    x_data, y_data = np.array(data[:, 0]), np.array(data[:, i])
    plt.plot(x_data,y_data);
plt.xlabel('Time [s]');
plt.ylabel('CW-IRSL intenisty I(t) [a.u.]')
plt.text(100,900,"CW-IRSL")
plt.text(100,800,'Variable LED power')
plt.subplot(1,2,2);
for i in lps:
    x_data, y_data = np.array(data[:, 0]), np.array(data[:, i])
    y_data=np.array(x_data)*np.array(y_data)
    x_data =np.log(x_data)
    plt.plot(x_data,y_data);
plt.xlabel('ln(Time)');
plt.ylabel('t x I(t) [a.u.]')
plt.text(0,18000,"CW-IRSL")
plt.text(0,20000,'Trasformed')
plt.tight_layout();
plt.show()
```

```
Text(0, 0.5, 'CW-IRSL intenisty I(t) [a.u.]')
Text(100, 900, 'CW-IRSL')
Text(100, 800, 'Variable LED power')
Text(0, 0.5, 't x I(t) [a.u.]')
Text(0, 18000, 'CW-IRSL')
Text(0, 20000, 'Trasformed')
```

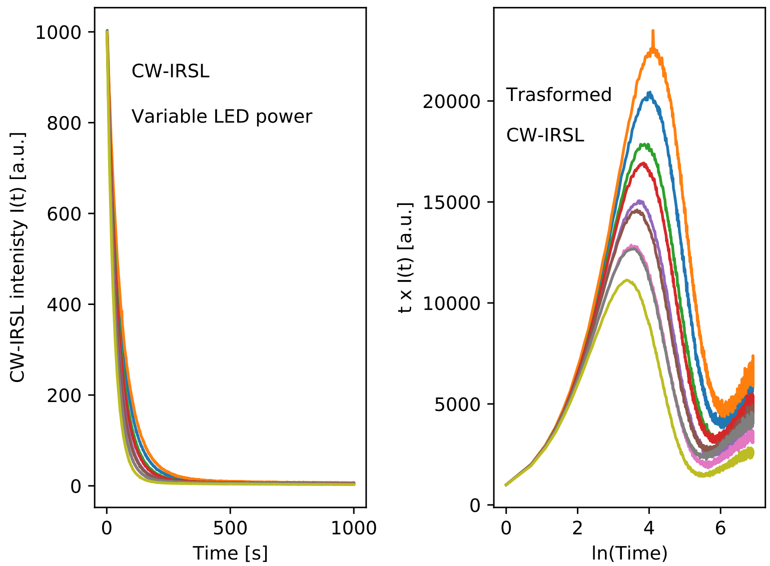



**Fig. 9.8:** (a) CW-OSL data in quartz, measured with different stimulating powers (b) The same data transformed into peak-shaped curves. For details of this data see Polymeris [4]

# References


1. V. Pagonis, G. S. Polymeris, G. Kitis, On the effect of optical and isothermal treatments on luminescence signals from feldspars, Radiation Measurements 82 (2015) 93–101.
2. G. Kitis, G. S. Polymeris, V. Pagonis, Stimulated luminescence emission: From phenomenological models to master analytical equations, Applied Radiation and Isotopes 153 (2019) 108797. `doi:https://doi.org/10.1016/j.apradiso.2019.05.041`.
   URL `http://www.sciencedirect.com/science/article/pii/S0969804319304142`
3. E. Bulur, H. Y. Göksu, Infrared (IR) stimulated luminescence from feldspars with linearly increasing excitation light intensity, Radiation Measurements 30 (1999) 505–512. `doi:10.1016/s1350-4487(99)00207-3`.
4. G. S. Polymeris, OSL at elevated temperatures: Towards the simultaneous thermal and optical stimulation, Radiation Physics and Chemistry 106 (2015) 184 – 192. `doi:https://doi.org/10.1016/j.radphyschem.2014.07.003`.
   URL `http://www.sciencedirect.com/science/article/pii/S0969806X14002916`