Boltzmann equation: scattering terms and beyond

O. Abdurazakov[1]

[1] ***NC State*** *University, Department of Physics, Raleigh, NC 27695*

In this project, I solve the semiclassical Boltzmann equation in the presence of electron-lattice scattering to study the decay of excited electrons in a model system. The decay rates are found to be dependent on the electronic density of states and lattice temperature.

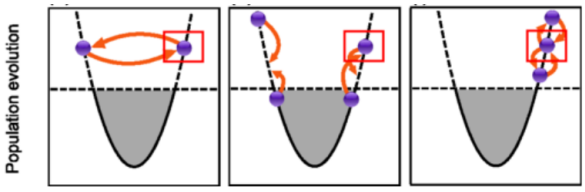

FIG. 1. Typical microscopic scattering processes electron experience in a material: electron-impurity, electron-electron, and electron-phonon[7]

INTRODUCTION

Most of the physical properties materials posses emerge out of the combination or/and competition between varius micrascopic scattering processes electrons experience. Out of these processes, the important ones are electron-impurity, Coulomb, and electron-lattice interactions. Since the advent of ultrafast optical techniques, probing these processes in their intrinsic timescales has been on the focus[1]. Whereas early ultrafast optical techniques such as time-resolved reflectivity only gave indirect access to the electron temperature through the changes caused in the dialectric function under optical excitation, a direct access to the dynamics of the electronic states through the time- and angle-resolved photoemission spectroscopy has been a big leap in this direction[1]. On the theory side, the description of excited electrons moved beyond multitemperature models. To account for the nonthermal behaviour of electrons observed in experiments, the Boltzamann transport equation has been solved numerically[2] and analytically for some limiting cases[3]. Also, numerous advanced computational techniques have been imployed such as time-dependent DMFT[4], nonequilibrium Keldysh method[5, 6]. In this project, we focus on the Boltzmann equation and its application to the dynamics of excited electrons. We compute the evolution of the excited electrons in the presence of electron-lattice scattering. Our main result is the dependence of the electron relaxation rates on the density of states and the lattice temperature.

METHOD AND MODEL

In the absence of the spacial diffusion of electrons after the optical excitation (typical timescales for the diffusion of electrons in a material are much longer compared to the Coulomb and electron-lattice scattering times), the evolution of the distribution function for the excited electrons is described by the Boltzmann Transport Equation[8]

$$\dot{f}(\vec{k},t) = I_{\rm coll}[f], \tag{1}$$

and $I_{\rm coll}[f] = I_{\rm e-imp}[f] + I_{\rm e-e}[f] + I_{\rm e-p}[f]$. Here, the collision term accounts for the different types of scattering processes electrons can encounter in a material: electron-impurity, electron-electron, and electron-phonon interactions. These processes are shown in Fig.1. In the case if impurity scattering, the energy of individual electron is conserved. Scattering off the almost immobile impurity sites can only change the direction of the electron momenta, not the magnitude. On the other hand, electrons can exchange energy and momenta amongst each other under the condition that the total electron energy is conserved. It can be shown that the electron-impurity contribution to the scattering rates is proportional to the density of states, and the Coulomb contribution scales as $1/\tau_{\rm e-e} \propto \omega^2 + (\pi T)^2$, where ω is the electron energy and T is the system temperature, respectively[9]. This is the typical behavior of a Fermi liquid, where the electrons behave as a free particle with renormalized parameters due to interactions. Here, we compute the decay rates only due to the electron-phonons scattering. As it was shown earlier that the impurity scattering does not contribute to the relaxation, and the Coulomb scattering only contribute in a subtle way, where the main features of the decay rate as a function of energy can be captured mainly by the electron-boson scattering channel as it is the only process which can draw energy out of electrons and let electrons relax toward the thermodynamic equilibrium[9]. Because of the filling and emptying processes from the continuum of energy states, the population density $f(x,t)$ evolves according to the Fermi's Golden rule

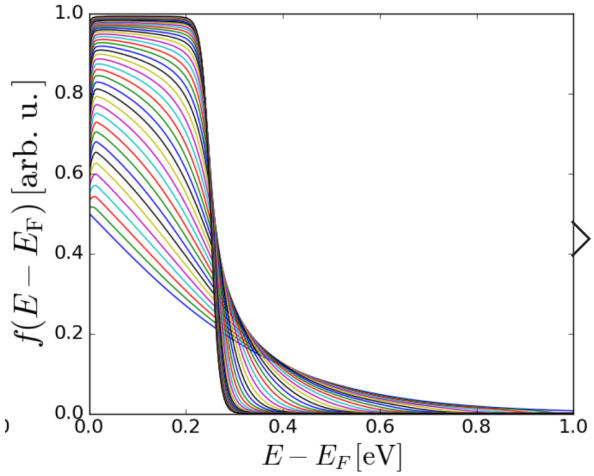

FIG. 2. Time evolution of the electron distribution function. It evolves toward the Fermi-Dirac distribution function.

$$\begin{aligned}\partial_t f(x) = \int d\omega F(\omega)\lambda\Big\{ & n(\omega)f(x-\omega)(1-f(x))N(x-\omega) \\ & -n(\omega)f(x)(1-f(x+\omega))N(x+\omega) \\ & +(n(\omega)+1)f(x+\omega)(1-f(x))N(x+\omega) \\ & -(n(\omega)+1)f(x)(1-f(x-\omega))N(x-\omega)\Big\} \\ & -\Gamma_{\rm esc}f(x), \end{aligned} \tag{2}$$

where F and N are the phonon and electron density of states, respectively. The phonon occupation number n is given by the Bose function $n(\omega) = 1/(e^{\omega/T_{\rm ph}} - 1)$. We also assume a constant electron-phonon coupling constant λ whose value does not change the results qualitatively. There are two types of scattering processes: intraband and interband. The number of electrons is conserved only in the former case. Electrons can escape from a band via the escape rate is given by $\Gamma_{\rm esc}$. We demonstrate our results for a linear electrons DOS and Gaussian shaped phonon DOS around 10 meV energy. The choice of the electronic and phononic input parameters are motivated by the 3D topological insulators of the likes Bismuth Selenide/Telluride[10]. The Boltzmann equation is a non-linear integro-differential equation, which can be solved numerically. During the time evolution, the total electron density $n \equiv \int dx N(x)f(x)$ is conserved. We use linearly discretized energy and time grids. The electrons are excited into the band perturbativaly, so the distribution function is given by $f(x,0) = e^{-x/\epsilon_0}$ where $\epsilon_0 = 0.2\,$eV. Initially, the lattice temperature is set to 100 K.

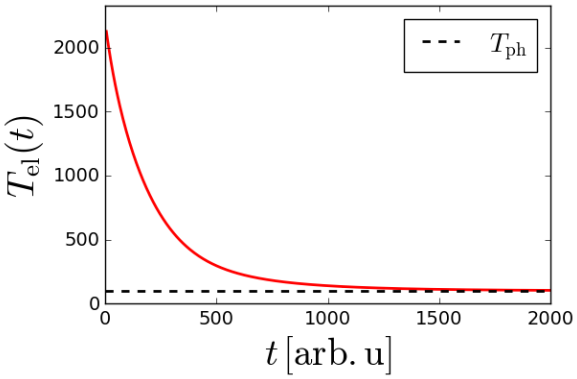

FIG. 3. Time evolution of the extracted electron temperature. It evolves toward that of the lattice.

RESULTS

In Fig.2, we present the time evolution of the electron distribution function. The initial Boltzmann type distribution function after some time settles into the Fermi-Dirac type distribution function at the lattice temperature. One can see this by fitting the distribution functions into the Fermi-Dirac distribution function and extract the effective electron temperatures. This is presented in the next figure. In our calculation, the lattice temperature is fixed. This is a good approximation in the low temperature limit, where the lattice heat capacity scales as $(T/T_{\rm D})^3$ and the electron heat capacity scales as $T/T_{\rm F}$, where $T_{\rm D}$ and $T_{\rm F}$ are the Debye and Fermi temperatures, respectively. In theory, one can write the same kind of integro-differential equation for the phonon occupation too.

Because the number of the electrons excited into the band is conserved through the interband scattering, the populations in lower energy do not decay as it is seen in Fig.4. Therefore, we set the electron escape rate from the band to be nonzero in order to extract the decay rates.

In Fig.5, the time traces of the electron populations at various energies are given. We can see that the those at higher energies decay faster, and vice versa. We can extract the decay rates by fitting these curves into exponential functions although the curves are not strictly exponential. Figure 6 shows the extracted rates at various lattice temperatures: 100 K, 150 K, 200 K, 250 K,. The rates are given in arbitrary units because we are only interested in their qualitative behavior with respect to energy and lattice temperature. The finite decay rates at the bottom of the band is equal to the band escape rate. Mostly, the rates scale linearly with energy as do the density of states. This is expected because the number of available states also scale linearly with energy. Also, the rates decrease with the increase of the lattice temperature. This is because the absorption rates go up with increase of the Bose function, which itself increases with

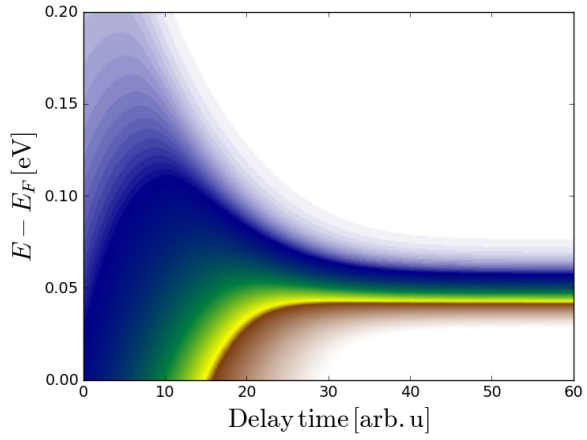

FIG. 4. Time evolution of the distribution function at lower energies.

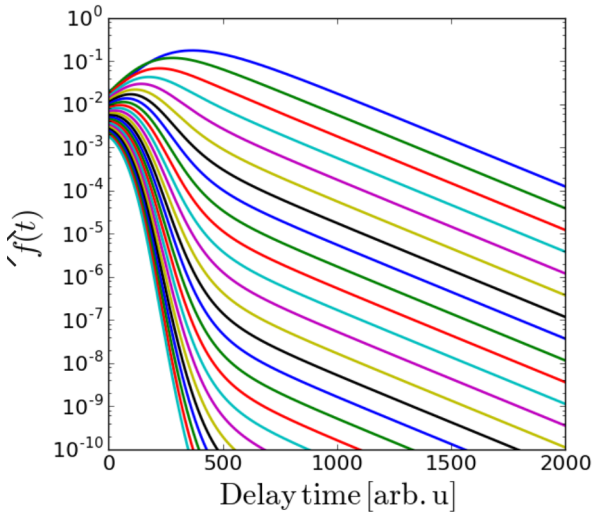

FIG. 5. Time traces of the distribution function at various energies.

increasing temperature.

CONCLUSIONS

In this project, we solve the Boltzmann integro-differential equation numerically in the presence of the electron-phonon collision term to study the dynamics of the excited electrons in a linear electronic band. We find that the relaxation via phonons is an efficient relaxation pathway which can readily draw the excess energy out of electrons even though the phonon dispersion is situated at the very low side of energy spectra. We also find that the decay rates depend on the electron density of states and the lattice temperature. These results may offer a valuable insight into the nonequilibrium phenomena in quantum materials. The calculations can be readily extended to include the phonon occupation number feedback mechanism and the other types of interactions such as impurity and Coulomb scattering. To treat the problem fully quantum mechanically, one needs to solve the Dyson equation self-consistently for the double time electron and phonon correlation functions, where the effect of initial correlations can be included, and the quasiparticle behavior of electrons can be accounted for[11].

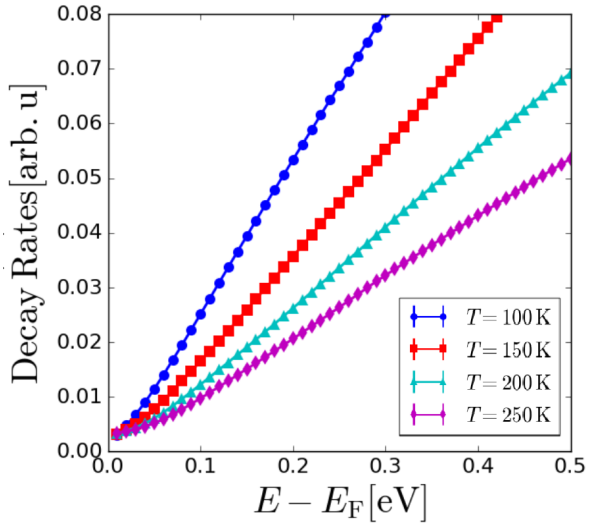

FIG. 6. The decay rates of excited electrons at various lattice temperatures.

[1] C. Giannetti, M. Capone, D. Fausti, M. Fabrizio, F. Parmigiani, and D. Mihailovic, Advances in Physics **65**, 58 (2016).

[2] R. H. M. Groeneveld, R. Sprik, and A. Lagendijk, Phys. Rev. B **51**, 11433 (1995).

[3] V. V. Kabanov and A. S. Alexandrov, Phys. Rev. B **78**, 174514 (2008).

[4] J. K. Freericks, V. M. Turkowski, and V. Zlatić, Phys. Rev. Lett. **97**, 266408 (2006).

[5] M. Sentef, A. F. Kemper, B. Moritz, J. K. Freericks, Z.-X. Shen, and T. P. Devereaux, Phys. Rev. X **3**, 041033 (2013).

[6] A. F. Kemper, M. A. Sentef, B. Moritz, J. K. Freericks, and T. P. Devereaux, Phys. Rev. B **90**, 075126 (2014).

[7] S.-L. Yang, J. A. Sobota, D. Leuenberger, Y. He, M. Hashimoto, D. H. Lu, H. Eisaki, P. S. Kirchmann, and Z.-X. Shen, Phys. Rev. Lett. **114**, 247001 (2015).

[8] J. M. Ziman, *Electrons and phonons: the theory of transport phenomena in solids*, International series of monographs on physics (Clarendon Press, Oxford, 1960).

[9] A. F. Kemper, O. Abdurazakov, and J. K. Freericks, Phys. Rev. X **8**, 041009 (2018).

[10] J. Sobota, S.-L. Yang, D. Leuenberger, A. Kemper, J. Analytis, I. Fisher, P. Kirchmann, T. Devereaux, and Z.-X. Shen, Journal of Electron Spectroscopy and Related Phenomena **195**, 249 (2014).

[11] A. Stan, N. E. Dahlen, and R. van Leeuwen, The Journal of chemical physics **130**, 224101 (2009).

APPENDIX: CODES

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


get_ipython().run_line_magic('pylab', 'inline')


# In[41]:


def fermi(x,T):
    return 1./(exp(x/T) + 1.0)


# In[3]:


def bose(x,T):
    return 1./(exp(x/T) - 1.0)


# In[4]:


def electron_dos_linear(x):
    r = zeros(len(x), dtype = float)
    cond = logical_and(x >= band_bottom, x <= band_top)
    r[cond] = x[cond]
#    r[cond] = x[cond]**2f
#    r[cond] = 1.0
    return r


# In[5]:


def phonon_dos(x):
    r = zeros(len(x), dtype = float)
    c1 = logical_and(x>=0.0, x <= 0.004)
    c2 = logical_and(x>0.004, x <= 0.018)
    r[c1] = 6.25*1e4*x[c1]**2
    r[c2] = 1.0
    return r


# In[6]:


def phonon_dos_gauss(x):
    # center
    mu = 0.010
    # width
    sigma = 0.002
    return exp(-power(x-mu,2.) / (2 * power(sigma,2)))


# In[7]:


band_bottom = 0.0
band_top = 2.0
phonon_band_width = 0.020
margin = 0.050

tmin = 0.0
tmax = 2000

dt = 0.2
dx = 0.001
dw = dx
Nt = int(round(tmax/dt)) + 1
Nx = int(round((band_top - band_bottom + 2 * margin)/dx))+1
Nw = int(round((phonon_band_width - band_bottom)/dx))+1

# Grids
t = linspace(tmin,tmax,Nt)
x = linspace(band_bottom - margin, band_top + margin,Nx)
w = linspace(band_bottom, phonon_band_width, Nw)

print "Electron Band: [",band_bottom,"-", band_top, "] eV"
print "Maximum Time: ", tmax


# In[8]:


# Electron DOS
N = electron_dos_linear(x)
# Phonon DOS
F = phonon_dos(w)


# In[9]:


fig,ax = subplots(1,1)
ax.plot(x,N,'b-',linewidth=3)
#ax.set_xlim(0,0.4)
#ax.set_ylim(0,1)
ax.set_xlabel(r'$x\mathrm{[eV]}$',fontsize=28)
ax.set_ylabel(r'$N(x)$',fontsize=28)
ax.tick_params(axis='both',labelsize=14)
savefig('edos.png',bbox_inches='tight')


# In[10]:


fig,ax = subplots(1,1)
ax.plot(w*1000,F,'r-o',linewidth=3)
#ax.set_ylim(0,1.1)
ax.set_xlim(0,0.02*1000)
ax.set_xlabel(r'$\omega\,\mathrm{[meV]}$', fontsize=28)
ax.set_ylabel(r'$F\,(\omega)$',fontsize=28)
ax.tick_params(axis='both',labelsize=14)
savefig('pdos.png',bbox_inches='tight')


# In[11]:


def Icoll(f,Tph):
```

```
    ### Collision integral due to  the eletron-phonon scattering

    coll = zeros(len(x), dtype = float)

    for ix, xx in enumerate(x):

        if xx >= band_bottom and xx <= band_top:
            fa,fe,ee,ea = 0.0,0.0,0.0,0.0

            for iw,ww in enumerate(w):
                if ww!=0.0:

                    fa +=       bose(ww,Tph)   * f[ix-iw] * (1 - occ*f[ix]    ) * N[ix-iw] * F[iw]
                    fe += (1 + bose(ww,Tph)) * f[ix+iw] * (1 - occ*f[ix]    ) * N[ix+iw] * F[iw]
                    ee += (1 + bose(ww,Tph)) * f[ix]    * (1 - occ*f[ix-iw]) * N[ix-iw] * F[iw]
                    ea +=       bose(ww,Tph)   * f[ix]    * (1 - occ*f[ix+iw]) * N[ix+iw] * F[iw]
            coll[ix] = lam * (fa + fe - ea - ee) * dw - G_escape * f[ix]
#             print coll[ix],fa,fe,ea,ee
    return coll


# In[12]:


def evolve(Tel,Tph):
        ### Time evolution of the electron density by the finite-difference method

        dens = zeros([len(t), len(x)], dtype = float)
        f = zeros(len(x), dtype = float)
        cond = logical_and(x >= band_bottom, x <= band_top)
        ## Initial electron distribution
        if fermi==True:
            f[cond] = fermi(x[cond],Tel)
        else:
            const = 0.1
            excit_height = 0.2
            f[cond] = const * exp(-x[cond]/excit_height)
        print sum(f*N)*dx
        for it,tt in enumerate(t):

                dens[it:,] = f
                f += Icoll(f,Tph)*dt
                if it%(int(round(Nt/10)))==0:
                        print sum(f*N)*dx, "Time: ", tt
        return dens


# In[14]:


occ = True
fermi =False
print "Occupied Band? ", occ
lam = 70.0
G_escape = 0.000
print "Escape rate from the band:", G_escape
Tel = 0.2
#Tph = 0.004309 # 50 K
Tph = 0.008617 # 100 K
#Tph = 0.01723 # 200 K
#Tph = 0.02585 # 300 K
print "Particle number in the band:"
import time
t0=time.clock()
pop = evolve(Tel,Tph)
t1=time.clock()
print "Time spent for evolution:",t1-t0," s"


# In[28]:


# fig,ax = subplots(1,1,figsize=(8,6))
# i = 90
# print x[i]
# ax.plot(t,pop[:,i],linewidth=1,label='python')
# xx,yy = loadtxt('../../lex_sobota/p04.dat', unpack=True)
# ax.plot(xx,yy,'k--',linewidth=3,label='c++')
# ax.legend(loc=0)
# savefig('trace.png',bbox_inches='tight')


# In[29]:


def check():

    fig,ax=subplots(1,2,figsize=(16,6))
    #ax[0].set_xlim(amin(t),amax(t))
    ax[0].set_xlabel(r'$\mathrm{Delay\,time\,[arb.u]}$',fontsize=24)
    #ax[0].set_ylim(0,0.55)
    ax[0].set_ylabel(r'$f(\mathrm{t})\mathrm{\,[arb.\,u.]}$',fontsize=28)
    #ax[1].set_xlim(amin(x),amax(x))
    ax[1].set_xlim(0,1)

    ax[1].set_ylabel(r'$f(E-E_\mathrm{F})\mathrm{\,[arb.\,u.]}$',fontsize=28)
    ax[1].set_xlabel(r'$E-E_F\,\mathrm{[eV]}$',fontsize=24)
    j=0
    for i in range(Nx/10,Nx/5,4):
        ax[0].plot(t,pop[:,i],linewidth=1)
        j += 1
    j=0
    for i in range(0,Nt,Nt/100):
#         ax[1].plot(x,pop[i,:],linewidth=1)
        ax[1].plot(x,pop[i,:],linewidth=1)

        j +=1
#     ax[1].set_ylim(0.001,2)
    #ax[1].legend(loc=1,fontsize=16)
    #ax[0].legend(loc=1,fontsize=16)
    ax[0].tick_params(axis='both',labelsize=16)
    ax[1].tick_params(axis='both',labelsize=16)
#     ax[1].plot(x,fermi(x-0.068571,T=Tph),'k--',linewidth=3,label=r'$n_\mathrm{F}(T_\mathrm{ph})$')
    ax[1].legend(loc=0,fontsize=24)

#     savefig('evolve_fermi_noescape.png',bbox_inches='tight')
```

```
# In[30]:


check()


# In[31]:


import matplotlib.colors as mcolors

def make_colormap(seq):
    """Return a LinearSegmentedColormap
    seq: a sequence of floats and RGB-tuples.
    The floats should be increasing
    and in the interval (0,1).
    """
    seq = [(None,) * 3, 0.0] + list(seq) + [1.0,
     (None,) * 3]
    cdict = {'red': [], 'green': [], 'blue': []}
    for i, item in enumerate(seq):
        if isinstance(item, float):
            r1, g1, b1 = seq[i - 1]
            r2, g2, b2 = seq[i + 1]
            cdict['red'].append([item, r1, r2])
            cdict['green'].append([item, g1, g2
    ])
            cdict['blue'].append([item, b1, b2])
    return mcolors.LinearSegmentedColormap('
    CustomMap', cdict)

darkblue = (0,0,139/256.)
darkgreen = (0,128/256.,64/256.)
blue = (128/256.,128/256.,256/256.)
green = (0/256.,128/256., 64/256.)
yellow = (1,1,0)
white = (1,1,1)
sienna4 = (128/256.,64/256.,20/256.)
rvb = make_colormap(         [ white, darkblue,
    0.075, darkblue, darkgreen, 0.3, darkgreen,
    yellow, 0.48, yellow, sienna4, 0.52, sienna4
    ,          white, 0.9, white, white, 1.0,
    white] )
rvb1 = make_colormap(         [ white, darkblue,
    0.2, darkblue, darkgreen, 0.2, darkgreen,
    yellow, 0.4, yellow, sienna4, 0.52, sienna4,
              white, 0.9, white, white, 1.0,
    white] )

rvb2 = make_colormap([white, blue,
    0.15/0.25-0.06, blue, green, 0.15/0.25+0.06,
     green, green, 0.75, green, white, 1., white
    ])


# In[32]:


fig,ax = subplots(1,1,figsize=(8,6))
im = ax.imshow(pop.T, extent=(0,tmax,band_bottom
    -margin,band_top+margin), origin='lower',
    aspect='auto', cmap=rvb)
ax.set_ylim(0,0.1)
ax.set_xlabel(r'$\mathrm{Delay\,time\,[arb.u]}$'
    ,fontsize=24)
ax.set_ylabel(r'$E-E_F\,\mathrm{[eV]}$',fontsize
    =24)
ax.tick_params(axis='both',labelsize=14)
#savefig('falsecolor_fermi_noescape.png',
    bbox_inches='tight')


# In[42]:


def get_temp(pop,ff=False):

    def boltzmann_fit(x,y,p0):

        from scipy.optimize import curve_fit
        def fitfunc(x,T,a):
            return  a*exp(-x/T)

        if p0==None:
            p0 = [0.05,1.0]
        itmin = argmin(abs(x-0))
        itmax = argmin(abs(x-1.0))
        xx = copy(x[itmin:itmax])
        yy = y[itmin:itmax]
        (popt, pcov) = curve_fit(fitfunc,xx,yy,
    p0=p0)
        p0 = popt

        return popt[0],sqrt(pcov[0,0]),popt[1]

    def fermi_fit(x,y,p0):

        from scipy.optimize import curve_fit
        def fitfunc(x,T,mu,a):
            return  a*ffermi(x-mu,T)
        if p0==None:
            p0 = [0.1,0.1,1.1]
        itmin = argmin(abs(x-0))
        itmax = argmin(abs(x-0.9))
        xx = copy(x[itmin:itmax])
        yy = y[itmin:itmax]
        (popt, pcov) = curve_fit(fitfunc,xx,yy,
    p0=p0)
        p0=popt
        return popt[0],sqrt(pcov[0,0]),popt[1]

    gamma = []
    err = []
    xl =[]
    p0=None
    for it,tt in enumerate(t):
        if it>0 and it%5==0:
            z = pop[it,:]
            #tt = copy(t[450:550])
            #zz = copy(z[450:550])
            #g,yerr=expo_fit(tt,zz)
            if ff:
                g,yerr,mu = fermi_fit(x,z,p0)
            else:
                g,yerr,mu = boltzmann_fit(x,z,p0
    )
            gamma.append(g)
            err.append(yerr)
            xl.append(tt)
            if it==2450:
                print mu

    gamma = array(gamma)
    err = array(err)
    xl = array(xl)
    return xl,gamma,err
```

```
# In[43]:


x1,y1,z1 = get_temp(pop,True)


# In[44]:


fig,ax = subplots(1,1)
con = 11604
ax.errorbar(x1,y1*con,yerr=z1,fmt='r-',color = 'r',markeredgewidth=0,linewidth=2)
ax.axhline(y=Tph*con,color='k',linestyle='--',linewidth=2,label=r'$T_\mathrm{ph}$')
ax.set_xlabel(r'$t\,\mathrm{[arb.u]}$',fontsize=28)
ax.set_ylabel(r'$T_\mathrm{el}(t)$',fontsize=28)
ax.tick_params(axis='both',labelsize=14)
ax.legend(loc=0,fontsize=20)
print 0.1*con
#ax.set_xlim(0,9)
ax.set_ylim(0,y1[0]*con+200)
#ax.annotate(r'$T_\mathrm{el}(0)=580\, \mathrm{K}$',xy=(0.6,0.6),xycoords='axes fraction',color='k',fontsize=20)
#savefig('temp_fermi_noescape.png',bbox_inches='tight')


# In[122]:


def get_rates(pop):

    def expo_fit(x,y,p0):

        from scipy.optimize import curve_fit
        def fitfunc(x,a,b,c):
            return a*exp(-x*b)+c
        def convolve_exp_norm(x,alpha, mu, sigma,a,b):
            co = alpha/2.0 * exp( alpha*mu+alpha*alpha*sigma*sigma/2.0)
            x_erf = (mu + alpha*sigma*sigma - x)/(sqrt(2.0)*sigma)
            r = a*co * exp(-alpha*x) * (1.0 - scipy.special.erf(x_erf))+b
            return r

        if p0==None:
            p0 = [y[0],0.002,y[-1]]
        (popt, pcov) = curve_fit(fitfunc,x,y,p0=p0)
        p0=popt
        #ax.plot(x,fitfunc(x,popt[0],popt[1],popt[2]),'r--',label='Fit',linewidth=2)
        return popt[1],sqrt(pcov[1,1]),p0
    gamma = []
    err = []
    xl =[]
    p0 = None
    for ixx,xx in enumerate(x):
        if xx>band_bottom and xx<=band_top and ixx%5==0:
            z = pop[:,ixx]
            z1 = 0.2*(amax(z)-z[-1])+z[-1]
            z2 = 0.002*(amax(z)-z[-1]) + z[-1]
            icen = argmin(abs(z-amax(z1)))
            imin = argmin(abs(z-z1))
            imax = argmin(abs(z-z2))
            tt = copy(t[imin:imax])
            zz = copy(z[imin:imax])
            g,yerr,p0=expo_fit(tt,zz,p0)
            print p0
#            g,yerr=expo_fit(t,z,p0)
            gamma.append(g)
            err.append(yerr)
            xl.append(xx)

    gamma = array(gamma)
    err = array(err)
    xl = array(xl)
    return xl,gamma,err



x1,y1,z1 = get_rates(pop)

fig,ax=subplots(1,1,figsize=(7,6))
#scaling factor to make the ylabel look better
sc = 200
ax.errorbar(x1,y1*sc,yerr=z1,fmt='d-',color = 'r',markersize=8,markeredgewidth=0,linewidth=2,label=r'$100\mathrm{K}$')
ax.set_xlabel(r'$E-E_F\,\mathrm{[eV]}$',fontsize=28)
ax.set_ylabel(r'$\mathrm{Decay\: Rate\,[arb.\,u.]}$',fontsize=28)
ax.tick_params('both', colors='k',labelsize=16)
ax.legend(loc=0,fontsize=24,frameon=False)
ax.set_ylim(0.0,0.03*sc)
ax.set_xlim(0,0.2)
ax.annotate(r'$\Gamma_\mathrm{escape}=%.1f$'%(G_escape*sc),xy=(0.6,0.1),xycoords='axes fraction',color='k',fontsize=28)
#savefig('rates_temp_occ.png',bbox_inches='tight')
```

. /home/omo/advanced_comp_physisc/final_project/boltzmann.py