Direct Solution of Boltzmann Transport Equation: Monte Carlo Method

Dragica Vasileska

Professor

Arizona State University

Tempe, AZ 85287-5706, USA

1. Introduction

The Ensemble Monte Carlo technique has been used now for over 30 years as a numerical method to simulate nonequilibrium transport in semiconductor materials and devices, and has been the subject of numerous books and reviews [i,ii,iii]. In application to transport problems, a random walk is generated to simulate the stochastic motion of particles subject to collision processes in some medium. This process of random walk generation may be used to evaluate integral equations and is connected to the general random sampling technique used in the evaluation of multi-dimensional integrals [iv].

The basic technique is to simulate the free particle motion (referred to as the free flight) terminated by instantaneous random scattering events. The Monte Carlo algorithm consists of generating random free flight times for each particle, choosing the type of scattering occurring at the end of the free flight, changing the final energy and momentum of the particle after scattering, and then repeating the procedure for the next free flight. Sampling the particle motion at various times throughout the simulation allows for the statistical estimation of physically interesting quantities such as the single particle distribution function, the average drift velocity in the presence of an applied electric field, the average energy of the particles, *etc*. By simulating an *ensemble* of particles, representative of the physical system of interest, the non-stationary time-dependent evolution of the electron and hole distributions under the influence of a time-dependent driving force may be simulated.

The particle-based picture, in which the particle motion is decomposed into free flights terminated by instantaneous collisions, is basically the same picture underlying the derivation of the semi-classical BTE. In fact, it may be shown that the one-particle distribution function obtained from the random walk Monte Carlo technique satisfies the BTE for a homogeneous system in the long-time limit [v].

2. Free-Flight Scatter

2.1 Free Flight Generation

In the Monte Carlo method, the dynamics of particle motion is assumed to consist of free flights terminated by instantaneous scattering events, which change the momentum and energy of the particle. To simulate this process, the probability density $P(t)$ is required, in which $P(t)dt$ is the joint probability that a particle will arrive at time t without scattering after the previous collision at $t = 0$, and then suffer a collision in a time interval dt around time t. The probability of scattering in the time interval dt around t may be written as $\Gamma[\mathbf{k}(t)]dt$, where $\Gamma[\mathbf{k}(t)]$ is the scattering rate of an electron or hole of wavevector **k**. The scattering rate, $\Gamma[\mathbf{k}(t)]$, represents the sum of the contributions from each individual scattering mechanism, which are usually calculated using perturbation theory, as described later. The implicit dependence of $\Gamma[\mathbf{k}(t)]$ on time reflects the change in **k** due to acceleration by internal and external fields. For electrons subject to time independent electric and magnetic fields, the equation of motion $\mathrm{F} = \hbar \frac{d\mathrm{k}}{dt} = -e\left[\mathrm{E} + \mathrm{v} \times \mathrm{B}\right]$ may be integrated to give the time evolution of **k** between collisions as

$$\mathbf{k}(t) = \mathbf{k}(0) - \frac{e(\mathbf{E} + \mathbf{v} \times \mathbf{B})t}{\hbar} , \tag{1}$$

where **E** is the electric field, **v** is the electron velocity and **B** is the magnetic flux density. In terms of the scattering rate, $\Gamma[\mathbf{k}(t)]$, the probability that a particle has not suffered a collision after a time t is given by $\exp\left(-\int_0^t \Gamma[\mathbf{k}(t')]dt'\right)$. Thus, the probability of scattering in the time interval dt after a free flight of time t may be written as the joint probability

$$P(t)dt = \Gamma[\mathbf{k}(t)]\exp\left[-\int_0^t \Gamma[\mathbf{k}(t')]dt'\right]dt . \tag{2}$$

Random flight times may be generated according to the probability density $P(t)$ above using, for example, the pseudo-random number generator implicit on most modern computers, which generate uniformly distributed random numbers in the range [0,1]. Using a direct method (see, for example [i]), random flight times sampled from $P(t)$ may be generated according to

$$r = \int_0^{t_r} P(t)dt\,, \tag{3}$$

where r is a uniformly distributed random number and t_r is the desired free flight time. Integrating (4) with $P(t)$ given by (3) above yields

$$r = 1 - \exp\left[-\int_0^{t_r} \Gamma[\mathbf{k}(t')]dt' \right]. \tag{4}$$

Since 1-r is statistically the same as r, Eq. (4) may be simplified to

$$-\ln r = \int_0^{t_r} \Gamma[\mathbf{k}(t')]dt'\,. \tag{5}$$

Equation (5) is the fundamental equation used to generate the random free flight time after each scattering event, resulting in a random walk process related to the underlying particle distribution function. If there is no external driving field leading to a change of **k** between scattering events (for example in ultrafast photoexcitation experiments with no applied bias), the time dependence vanishes, and the integral is trivially evaluated. In the general case where this simplification is not possible, it is expedient to introduce the so called self-scattering method [vi], in which we introduce a fictitious scattering mechanism whose scattering rate always adjusts itself in such a way that the total (self-scattering plus real scattering) rate is a constant in time

$$\Gamma = \Gamma[\mathbf{k}(t')] + \Gamma_{self}[\mathbf{k}(t')]\ , \tag{6}$$

where $\Gamma_{self}[\mathbf{k}(t')]$ is the self-scattering rate. The self-scattering mechanism itself is defined such that the final state before and after scattering is identical. Hence, it has no effect on the free flight trajectory of a particle when selected as the terminating scattering mechanism, yet results in the simplification of (6) such that the free flight is given by

$$t_r = -\frac{1}{\Gamma}\ln r\ . \tag{7}$$

The constant total rate (including self-scattering) Γ is chosen *a priori* so that it is larger than the maximum scattering encountered during the simulation interval. In the simplest case, a single value is chosen at the beginning of the entire simulation (constant gamma method), checking to ensure that the real rate never exceeds this value during the simulation. Other schemes may be chosen that are more computationally efficient, and which modify the choice of Γ at fixed time increments [vii].

2.2 Final State After Scattering

The algorithm described above determines the random free flight times during which the particle dynamics is treated semi-classically according to Eq. (1). For the scattering process itself, we need the type of scattering (i.e. impurity, acoustic phonon, photon emission, etc.) which terminates the free flight, and the final energy and momentum of the particle(s) after scattering. The type of scattering which terminates the free flight is chosen using a uniform random number between 0 and Γ, and using this pointer to select among the relative total scattering rates of all processes including self-scattering at the final energy and momentum of the particle

$$\Gamma = \Gamma_{self}\left[n,\mathbf{k}\right] + \Gamma_1\left[n,\mathbf{k}\right] + \Gamma_2\left[n,\mathbf{k}\right] + \ldots \Gamma_N\left[n,\mathbf{k}\right], \tag{8}$$

with n the band index of the particle (or subband in the case of reduced-dimensionality systems), and **k** the wavevector at the end of the free-flight.

Once the type of scattering terminating the free flight is selected, the final energy and momentum (as well as band or subband) of the particle due to this type of scattering must be selected. For this selection, the scattering rate, $\Gamma_j[n,\mathbf{k};m,\mathbf{k}']$, of the jth scattering mechanism is needed, where n and m are the initial and final band (subband) indices, and **k** and **k**' are the particle wavevectors before and after scattering. Defining a spherical coordinate system around the initial wavevector **k**, the final wavevector **k**' is specified by $|\mathbf{k}'|$ (which depends on conservation of energy) as well as the azimuthal and polar angels, φ and θ around **k**. Typically the scattering rate $\Gamma_j[n,\mathbf{k};m,\mathbf{k}']$ only depends on the angle θ between **k** and **k**'. Therefore, φ may be chosen using a uniform random number between 0 and 2π (i.e. $2\pi r$), while θ is chosen according to the cross-section for scattering arising from $\Gamma_j[n,\mathbf{k};m,\mathbf{k}']$. If the probability for scattering into a certain angle $P(\theta)d\theta$ is integrable, then random angles satisfying this probability density may be generated from a uniform distribution between 0 and 1 through inversion of Eq. (4). Otherwise, a rejection technique (see, for example, [i,ii]) may be used to select random angles according to $P(\theta)$.

3. Ensemble Monte Carlo Simulation

The algorithm above may be used to track a single particle over many scattering events in order to simulate the steady-state behavior of a system. Transient simulation requires the use of a *synchronous ensemble* of particles in which the algorithm above is repeated for each particle in the ensemble representing the system of interest until the simulation is completed. Figure 1 illustrates an ensemble Monte Carlo simulation which a fixed time step, Δt, is introduced to which the motion of all the carriers in the system is synchronized. The crosses **(×)** illustrate random, instantaneous, scattering events, which may or may not occur during one time-step. Basically, each carrier is simulated only up to the end of the time-step, and then the next particle in the ensemble is treated. Over each time step, the motion of each particle in the ensemble is simulated independent of the other particles. Nonlinear effects such as carrier-carrier interactions or the Pauli exclusion principle are then updated at each time step, as discussed in more detail below, in effect linearizing the solution.

The non-stationary one-particle distribution function and related quantities such as drift velocity, valley or subband population, etc., are then taken as averages over the ensemble at fixed time steps throughout the simulation. For example, the drift velocity in the presence of the field is given by the ensemble average of the component of the velocity at the *n*th time step as

$$\bar{v}_z(n\Delta t) \cong \frac{1}{N}\sum_{j=1}^{N} v_z^j(n\Delta t), \qquad (9)$$

where N is the number of simulated particles and j labels the particles in the ensemble. This equation represents an estimator of the true velocity, which has a standard error given by

$$s = \frac{\sigma}{\sqrt{N}}, \qquad (10)$$

where σ^2 is the variance which may be estimated from [iv]

$$\sigma^2 \cong \frac{N}{N-1}\left\{\frac{1}{N}\sum_{j=1}^{N}\left(v_z^j\right)^2 - \bar{v}_z^2\right\}. \qquad (11)$$

Similarly, the distribution functions for electrons and holes may be tabulated by counting the number of electrons in cells of k-space. From Eq. (11), we see that the error in estimated average quantities decreases as the square root of the number of particles in the ensemble, which necessitates the simulation of many particles. Typical ensemble sizes for good statistics are in the range of $10^4 - 10^5$ particles. Variance reduction techniques to decrease the standard error given by Eq. (11) may be applied to enhance statistically rare events such as impact ionization or electron-hole recombination [ii].

3.1 Scattering Processes

Free carriers (electrons and holes) interact with the crystal and with each other through a variety of scattering processes which relax the energy and momentum of the particle. Based on first order, time-dependent perturbation theory, the transition rate from an initial state **k** in band *n* to a final state **k**' in band *m* for the *j*th scattering mechanism is given by Fermi's Golden rule [viii]

$$\Gamma_j[n,\mathbf{k};m,\mathbf{k}'] = \frac{2\pi}{\hbar}\left|\left\langle m,\mathbf{k}'\middle|V_j(\mathbf{r})\middle|n,\mathbf{k}\right\rangle\right|^2 \delta\left(E_{\mathbf{k}'} - E_{\mathbf{k}} \mp \hbar\omega\right), \tag{12}$$

where $V_j(\mathbf{r})$ is the scattering potential of this process, $E_{\mathbf{k}}$ and $E_{\mathbf{k}'}$ are the initial and final state energies of the particle. The delta function results in conservation of energy for long times after the collision is over, with $\hbar\omega$ the energy absorbed (upper sign) or emitted (lower sign) during the process. Scattering rates calculated by Fermi's Golden rule above are typically used in Monte Carlo device simulation as well as simulation of ultrafast processes. The total rate used to generate the free flight in Eq. (9), discussed in the previous section, is then given by

$$\Gamma_j[n,\mathbf{k}] = \frac{2\pi}{\hbar}\sum_{m,\mathbf{k}'}\left|\left\langle m,\mathbf{k}'\middle|V_j(\mathbf{r})\middle|n,\mathbf{k}\right\rangle\right|^2 \delta\left(E_{\mathbf{k}'} - E_{\mathbf{k}} \mp \hbar\omega\right). \tag{13}$$

Figure 2 is given here for completeness. It illustrates the hierarchy of different scattering mechanisms that one typically includes in a standard EMC simulation, which is divided between lattice scattering mechanisms and various crystal defect types of scattering.

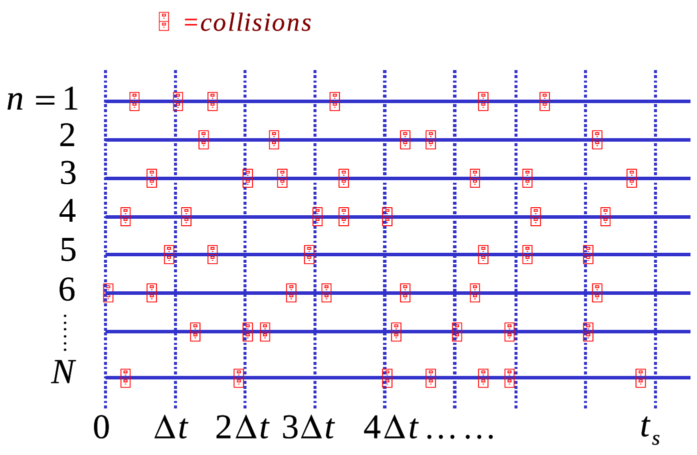


Figure 1. Ensemble Monte Carlo simulation in which a time step, Δ*t*, is introduced over which the motion of particles is synchronized. The squares represent random scattering events.

There are major limitations to the use of the Golden rule due to effects such as *collision broadening* and *finite collision duration time* [v]. The energy conserving delta function is only valid asymptotically for times long after the collision is complete. The broadening in the final state energy is given roughly by $\Delta E \approx \hbar/\tau$, where τ is the time after the collision, which implies that the normal $E(\mathbf{k})$ relation is only recovered at long times. Attempts to account for such *collision broadening* in Monte Carlo simulation have been reported in the literature [ix,x], although this is still an open subject of debate. Inclusion of the effects of *finite collision duration* in Monte Carlo simulation have also been proposed [xi,xii]. Beyond this, there is still the problem of dealing with the quantum mechanical phase coherence of carriers, which is neglected in the scatter free-flight algorithm of the Monte Carlo algorithm.

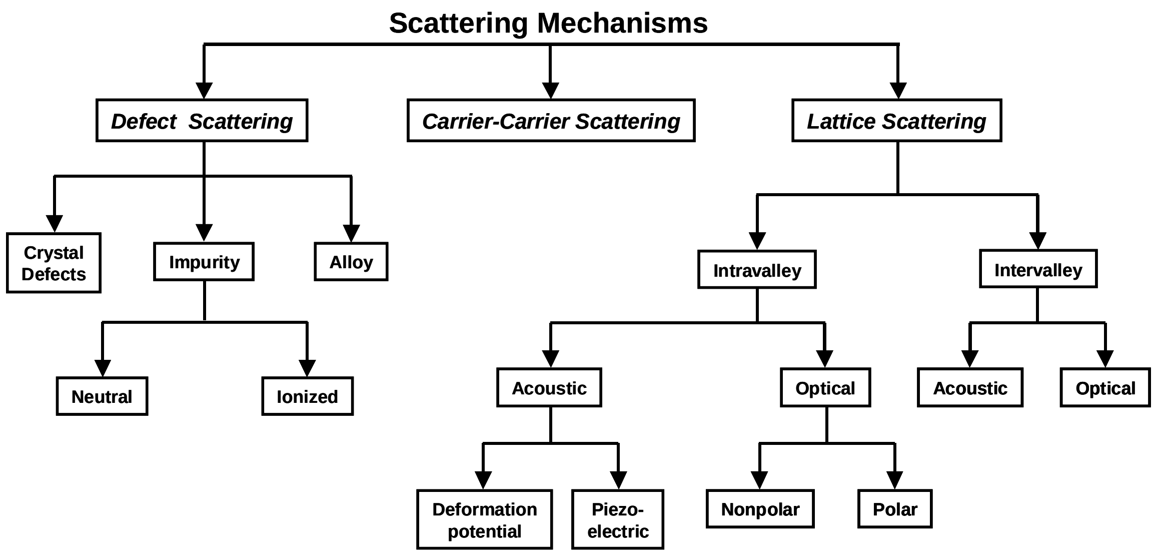


Figure 2. Scattering mechanisms in a typical semiconductor.

3.2 Bulk Monte Carlo Code for GaAs

To illustrate the practical implementation of the Ensemble Monte Carlo method discussed above, we give the details of a fortran code used for simulation of electrons in a typical three-valley semiconductor (such as GaAs in this case). Figure 6.3 shows the main flow-chart of the the Monte Carlo algorithim which is initiated through the execution of the **main()** routine that first initializes various parameters used in the code by calling the **readin()** subroutine and by the construction of the scattering table by calling the **sc_table()** subroutine. The next step in this procedure is to initialize the carrier energies and wavevectors by calling the **init()** routine. Having initialized the carriers, a check is performed by calling the **histograms()** subroutine that calculates the energy and wavevector histograms of the carriers, to ensure that carriers were initialized according to Maxwell-Boltzmann distribution. After the initialization procedure is completed, a free-flight-scatter loop is performed until the end of the desired simulation time specified in the subroutine **readin()**. Within each time step, the **free_flight_scatter()** subroutine is called first, then the program proceeds to the **histograms()** subroutine to check the change in the distribution function under the application of uniform electric field. Finally, the **write()** subroutine is called to record the time evolution of variables such as the carrier wavevectors, energy, valley index and other relevant parameters. A flow chart of the **main()** program is illustrated in Figure 3.

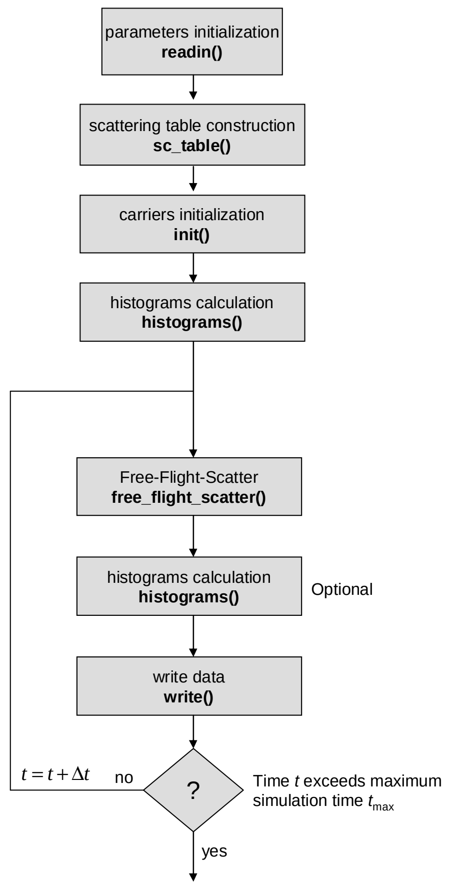


Figure 3. Flow-chart of the main program for a three valley EMC simulation.

The source code **main()** is listed below. Included here are also the **histograms()** and the **write()** subroutines.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      MAIN PROGRAM FOR MONTE CARLO SOLUTION OF THE BOLTZMANN
C      TRANSPORT EQUATION FOR A THREE-VALLEY SEMICONDUCTOR
C
       parameter(n_lev=1000, nele=1000)
C
C      n_lev  => # of energy levels in the scattering table
C      nele   => # total number of electrons that are simulated
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

       common
     &/time_1/dt,dtau,tot_time

       character*40 file_name
       integer nsim

C     Read parameters form the input file
       call readin(n_lev)

C      Start the Monte Carlo Simulation

C     Calculate the scattering table

       call sc_table(n_lev)

C     Initialize carriers

       nsim = nele
       call init(nsim)
       file_name = 'initial_distribution'
       call histograms(nsim,file_name)

       time = 0.
       j = 0
       flag_write = 0.

       do while(time.le.tot_time)

          j = j + 1
          time=dt*float(j)
          call free_flight_scatter(n_lev,nsim)
          file_name = 'current_distribution'
          call histograms(nsim,file_name)
          call write(nsim,j,time,flag_write)
          flag_write = 1.

       enddo   ! End of the time loop

       end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      WRITE AVERAGES IN FILES
C      (these averages correspond to one time step)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       subroutine write(nsim,iter,time,flag_write)
       parameter(n_val = 3)

       common
     &/ran_var/iso
     &/pi/pi,two_pi
     &/fund_const/q,h,kb,am0,eps_0
     &/dri/qh
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/variables/p(20000,7),ip(20000),energy(20000)

      integer nsim,iter
      real kb,time
      character*40 file_1,file_2
      character*40 file_3,file_4,file_5

      real nvaly(n_val)
      real velx_sum(n_val)
      real vely_sum(n_val)
      real velz_sum(n_val)
      real velocity_x(n_val)
      real velocity_y(n_val)
      real velocity_z(n_val)
      real sume(n_val)

C     Open output files

      file_1 = 'vx_time_averages'
      file_2 = 'vy_time_averages'
      file_3 = 'vz_time_averages'
      file_4 = 'energy_time_averages'
      file_5 = 'valley_occupation'
      open(unit=1,file=file_1,status='unknown')
      open(unit=2,file=file_2,status='unknown')
      open(unit=3,file=file_3,status='unknown')
      open(unit=4,file=file_4,status='unknown')
      open(unit=5,file=file_5,status='unknown')

      do i = 1,n_val
         velx_sum(i) = 0.
         vely_sum(i) = 0.
         velz_sum(i) = 0.
         sume(i) = 0.
         nvaly(i) = 0
      enddo

      do i = 1,nsim

         iv=ip(i)
         ee = energy(i)
         denom = 1./(1.+af2(iv)*ee)
         velx = h*p(i,1)*denom/am(iv)
         vely = h*p(i,2)*denom/am(iv)
         velz = h*p(i,3)*denom/am(iv)
         velx_sum(iv) = velx_sum(iv) + velx
         vely_sum(iv) = vely_sum(iv) + vely
         velz_sum(iv) = velz_sum(iv) + velz
         sume(iv) = sume(iv) + energy(i)
         nvaly(iv) = nvaly(iv) + 1

       enddo

       do i = 1,n_val
          if(nvaly(i).ne.0)then
             velocity_x(i)=velx_sum(i)/nvaly(i)
             velocity_y(i)=vely_sum(i)/nvaly(i)
             velocity_z(i)=velz_sum(i)/nvaly(i)
             sume(i)=sume(i)/nvaly(i)
          endif
       enddo

       if(flag_write.eq.0)then
          write(1,*)' time  vx_gamma  vx_L  vx_X'
          write(2,*)' time  vy_gamma  vy_L  vy_X'
          write(3,*)' time  vz_gamma  vz_L  vz_X'
          write(4,*)' time  Ek_gamma  Ek_L  Ek_X'
          write(5,*)' time  gamma  L_valley  X-valley'
       endif

       if(mod(iter,4).eq.0)then
      write(1,88)time,velocity_x(1),
     1                velocity_x(2),velocity_x(3)
      write(2,88)time,velocity_y(1),
     1                velocity_y(2),velocity_y(3)
      write(3,88)time,velocity_z(1),
     1                velocity_z(2),velocity_z(3)
      write(4,88)time,sume(1),sume(2),sume(3)
          write(5,88)time,nvaly(1),nvaly(2),nvaly(3)
88        format(2X,4(e12.6,4x))
       endif

       return
       end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      SAVE HISTOGRAMS
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       subroutine histograms(nsim,file_name)
       common
     &/variables/p(20000,7),ip(20000),energy(20000)

       integer nsim
       real kx,ky,kz,e
       character*30 file_name

       open(unit=6,file=file_name,status='unknown')
       write(6,*)'kx   ky   kz   energy'

       do i = 1, nsim
          kx = p(i,1)
      ky = p(i,2)
      kz = p(i,3)
      e = energy(i)
          write(6,*)kx,ky,kz,e
       enddo
       close(6)

       return
       end
```

The **readin()** subroutine, which contains all the relevant parameters for modeling electronic transport in GaAs bulk material system, is listed next. In our model, we have assumed three conduction band valleys: Γ, L and X as depicted in Figure 4. All variables defined in the **readin()** subroutine have descriptive names and can be easily identified.

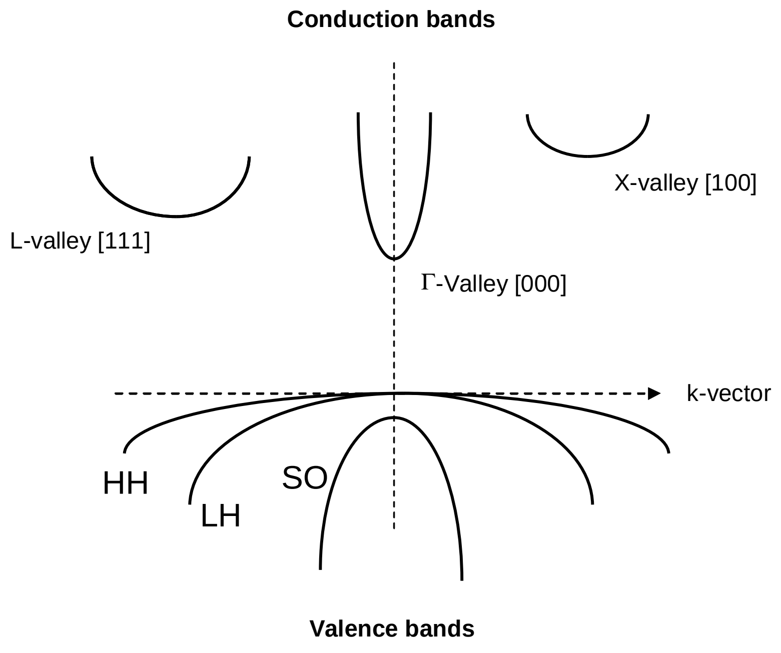


Figure 4. Energy band model for GaAs. The L-valley is at the (111) point and there are 8 equivalent [111] directions. Since these valleys are shared between Brillouin zones, there are a total of four equivalent L valleys. The X-valleys are at the [100] direction and since there are 6 equivalent [100] directions and the valleys are shared between Brillouin zones, there are 3 equivalent X valleys. HH stands for heavy hole band, LH for light-hole band and SO for split-off hole band.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C       READ INPUT PARAMETERS
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        subroutine readin(n_lev)

        common
     &/ran_var/iso
     &/pi/pi,two_pi
     &/fund_const/q,h,kb,am0,eps_0
     &/dri/qh
     &/temp/tem,Vt
     &/ek/am(3),smh(3),hhm(3)
     &/valley_splitting/split_L_gamma,split_X_gamma
     &/equiv_valleys/eq_valleys_gamma,eq_valleys_L,eq_valleys_X
     &/nonp/af(3),af2(3),af4(3)
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/dielec_func/eps_high,eps_low
     &/density/density,sound_velocity
     &/time_1/dt,dtau,tot_time
     &/force/fx,fy,fz
     &/select_acouctic/acoustic_gamma,acoustic_L,acoustic_X
     &/select_Coulomb/Coulomb_scattering
     &/select_polar/polar_gamma,polar_L,polar_X
     &/select_intervalley_1/intervalley_gamma_L
     &/select_intervalley_2/intervalley_gamma_X
     &/select_intervalley_3/intervalley_L_gamma
     &/select_intervalley_4/intervalley_L_L
     &/select_intervalley_5/intervalley_L_X
     &/select_intervalley_6/intervalley_X_gamma
     &/select_intervalley_7/intervalley_X_L
     &/select_intervalley_8/intervalley_X_X
     &/sigma_acoustic/sigma_gamma,sigma_L,sigma_X
     &/coulomb/doping_density,Energy_debye
     &/polar_en/polar_en_gamma,polar_en_L,polar_en_X
     &/Def_pot_1/DefPot_gamma_L,DefPot_gamma_X
     &/Def_pot_2/DefPot_L_gamma,DefPot_L_L,DefPot_L_X
     &/Def_pot_3/DefPot_X_gamma,DefPot_X_L,DefPot_X_X
     &/interval_phonons_1/phonon_gamma_L,phonon_gamma_X
     &/interval_phonons_2/phonon_L_gamma,phonon_L_L,phonon_L_X
     &/interval_phonons_3/phonon_X_gamma,phonon_X_L,phonon_X_X

        real kb
        real fx,fy,fz
        real nonparabolicity_gamma
        real nonparabolicity_L
        real nonparabolicity_X

C       Define fundamental constants and general parameters

        iso=1345
        am0=9.11e-31
        h=1.05459e-34
        q=1.60219e-19
        qh=q/h
        eps_0=8.85419e-12
        kb=1.38066e-23
        pi=4.*atan(1.0)
        two_pi = 2.*pi

C       Define time step and maximum simulation time

        dt=1.e-14
        tot_time = 100.e-12

C       Set temperature and doping density

        tem=300.
        Vt=kb*tem/q
        doping_density = 1.e21

C       Set the electric field
```

```
       fx = 0.
       fy = 7.e5
       fz = 0.

C      Define Masses for different valleys

       rel_mass_gamma = 0.063
       rel_mass_L = 0.170
       rel_mass_X = 0.58

C      Define non-parabolicity factors

       nonparabolicity_gamma = 0.62
       nonparabolicity_L = 0.50
       nonparabolicity_X = 0.30

c       nonparabolicity_gamma = 0
c       nonparabolicity_L = 0
c       nonparabolicity_X = 0

C      Define valley splitting and equivalent valleys

       split_L_gamma = 0.29
       split_X_gamma = 0.48
       eq_valleys_gamma = 1.
       eq_valleys_L = 4.
       eq_valleys_X = 3.

C      Define low-ferquency and high-frequency dielectric constants

       eps_high = 10.92
       eps_low = 12.9
       eps_high = eps_high*eps_0
       eps_low = eps_low*eps_0

C      Define crystal density and sound velocity

       density = 5370.
       sound_velocity = 5.22E3

C      Define parameters for the scattering table

       emax=1.0
       de=emax/float(n_lev)

C      Select scattering mechanisms

c      Coulomb_scattering = 1
       acoustic_gamma = 1
       acoustic_L = 1
       acoustic_X = 1
       polar_gamma = 1
       polar_L = 1
       polar_X = 1
       intervalley_gamma_L = 1
       intervalley_gamma_X = 1
       intervalley_L_gamma = 1
       intervalley_L_L = 1
       intervalley_L_X = 1
       intervalley_X_gamma = 1
       intervalley_X_L = 1
       intervalley_X_X = 1

C      Define coupling constants

       sigma_gamma = 7.01  !  [eV]
       sigma_L = 9.2  !  [eV]
       sigma_X = 9.0  !  [eV]

       polar_en_gamma = 0.03536  !  [eV]
       polar_en_L = 0.03536  !  [eV]
       polar_en_X = 0.03536  !  [eV]

       DefPot_gamma_L = 1.8E10   ! [eV/m]
       DefPot_gamma_X = 10.E10   ! [eV/m]
       DefPot_L_gamma = 1.8E10   ! [eV/m]
       DefPot_L_L = 5.E10   ! [eV/m]
       DefPot_L_X = 1.E10   ! [eV/m]
       DefPot_X_gamma = 10.E10   ! [eV/m]
       DefPot_X_L = 1.E10   ! [eV/m]
       DefPot_X_X = 10.E10   ! [eV/m]

       phonon_gamma_L = 0.0278  !  [eV]
       phonon_gamma_X = 0.0299  !  [eV]
       phonon_L_gamma = 0.0278  !  [eV]
       phonon_L_L = 0.029  !  [eV]
       phonon_L_X = 0.0293  !  [eV]
       phonon_X_gamma = 0.0299  !  [eV]
       phonon_X_L = 0.0293  !  [eV]
       phonon_X_X = 0.0299  !  [eV]

C      Map parameters into internal variables used in the code

       am(1) = rel_mass_gamma*am0
       am(2) = rel_mass_L*am0
       am(3) = rel_mass_X*am0
       af(1) = nonparabolicity_gamma
       af(2) = nonparabolicity_L
       af(3) = nonparabolicity_X

       do i = 1,3
          smh(i)=sqrt(2.*am(i))*sqrt(q)/h
          hhm(i)=h/am(i)/q*h/2.
          af2(i) = 2.*af(i)
          af4(i) = 4.*af(i)
c         print*,af(i),af2(i),af4(i)
       enddo

       return
       end
```

After the material and run parameters are read in, the next step is to construct scattering tables for the Γ, L and X valleys by calling the **sc_table()** subroutine that initializes a series of events that are summarized in Figure 5. At each energy, the cumulative scattering rates for each valley are stored in separate look-up tables, and renormalized according to the maximum scattering rate (including self-scattering) that occurs over the range of energies stored. The structure of these subroutines is such that adding additional scattering event is very trivial. In fact, one has to write only an additional subroutine that gives the energy dependence of the additional scattering mechanisms being considered and register that scattering process in the **sc_table()** subroutine itself.

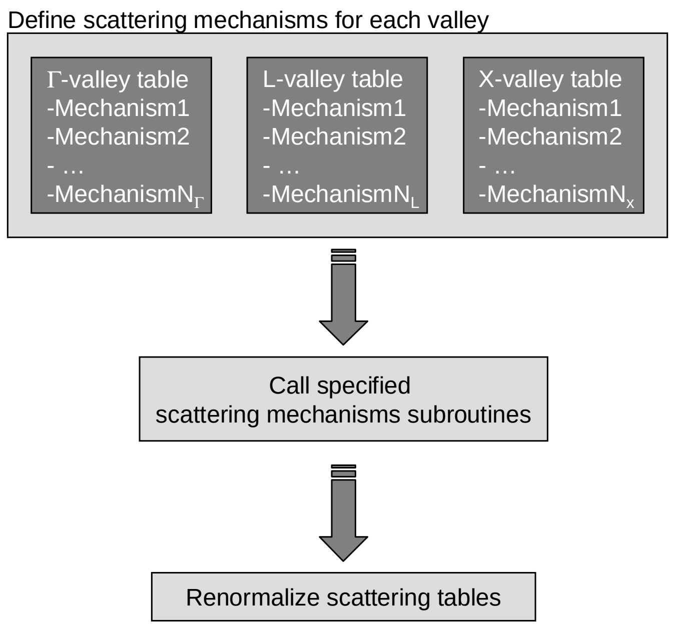


Figure 5. Procedure for the creation of the scattering tables.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       SUBROUTINE THAT CREATES THE SCATTERING TABLE
C       flag_mech = 1 ==> isotropic scattering process
C       flag_mech = 2 ==> polar optical phonons
C        flag_mech = 3 ==> Coulomb scattering
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       subroutine sc_table(n_lev)

       common
     &/pi/pi,two_pi
     &/temp/tem,Vt
     &/fund_const/q,h,kb,am0,eps_0
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/valley_splitting/split_L_gamma,split_X_gamma
     &/equiv_valleys/eq_valleys_gamma,eq_valleys_L,eq_valleys_X
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/scatt_par2/flag_mech(10,3),i_valley(10,3)
     &/table/scatt_table(4000,10,3)
     &/dielec_func/eps_high,eps_low
     &/density/density,sound_velocity
     &/select_acouctic/acoustic_gamma,acoustic_L,acoustic_X
     &/select_Coulomb/Coulomb_scattering
     &/select_polar/polar_gamma,polar_L,polar_X
     &/select_intervalley_1/intervalley_gamma_L
     &/select_intervalley_2/intervalley_gamma_X
     &/select_intervalley_3/intervalley_L_gamma
     &/select_intervalley_4/intervalley_L_L
     &/select_intervalley_5/intervalley_L_X
     &/select_intervalley_6/intervalley_X_gamma
     &/select_intervalley_7/intervalley_X_L
     &/select_intervalley_8/intervalley_X_X
     &/sigma_acoustic/sigma_gamma,sigma_L,sigma_X
     &/polar_en/polar_en_gamma,polar_en_L,polar_en_X
     &/Def_pot_1/DefPot_gamma_L,DefPot_gamma_X
     &/Def_pot_2/DefPot_L_gamma,DefPot_L_L,DefPot_L_X
     &/Def_pot_3/DefPot_X_gamma,DefPot_X_L,DefPot_X_X
     &/interval_phonons_1/phonon_gamma_L,phonon_gamma_X
     &/interval_phonons_2/phonon_L_gamma,phonon_L_L,phonon_L_X
     &/interval_phonons_3/phonon_X_gamma,phonon_X_L,phonon_X_X
     &/acoustic/sigma
     &/intervalley1/coupling_constant
     &/intervalley2/delta_fi,final_valleys,i_final

      integer n_lev
      character*30 out_file_1, out_file_2
      real kb

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     CREATE TABLE FOR THE GAMMA VALLEY
C     Scattering mechanism:   - acoustic phonons
C                             - Coulomb
C                             - polar optical phonons
C                             - gamma-to-L intervalley
C                             - gamma-to-X intervalley
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      iv = 1
      i_count = 0

C    Acoustic phonons scattering rate

      if(acoustic_gamma.eq.1)then
         sigma = sigma_gamma
         out_file_1 = 'acoustic_gamma'
         call acoustic_rate(i_count,iv,n_lev,out_file_1)
      endif

C    Coulomb scattering rate - Brooks-Herring approach

      if(Coulomb_scattering.eq.1)then
         out_file_1 = 'Coulomb_gamma'
         call Coulomb_BH(i_count,iv,n_lev,out_file_1)
      endif

C     Polar optical phonons scattering rate

      if(polar_gamma.eq.1)then
         w0 = polar_en_gamma
         out_file_1 = 'polar_gamma_ab'
         out_file_2 = 'polar_gamma_em'
         call polar_rate(i_count,iv,n_lev,w0,
     1                   out_file_1,out_file_2)
      endif

C    Intervalley scattering: gamma to L valley

      if(intervalley_gamma_L.eq.1)then
         w0 = phonon_gamma_L
         coupling_constant = DefPot_gamma_L
         delta_fi = split_L_gamma
         final_valleys = eq_valleys_L
         i_final = 2   ! final_valley number
         out_file_1 = 'intervalley_gamma_L_ab'
         out_file_2 = 'intervalley_gamma_L_em'
         call intervalley(i_count,iv,n_lev,w0,
     1                    out_file_1,out_file_2)
      endif

C    Intervalley scattering: gamma to X valley

      if(intervalley_gamma_X.eq.1)then
         w0 = phonon_gamma_X
         coupling_constant = DefPot_gamma_X
         delta_fi = split_X_gamma
         final_valleys = eq_valleys_X
         i_final = 3   ! final_valley number
         out_file_1 = 'intervalley_gamma_X_ab'
         out_file_2 = 'intervalley_gamma_X_em'
         call intervalley(i_count,iv,n_lev,w0,
     1                    out_file_1,out_file_2)
      endif

      call renormalize_table(iv,n_lev,i_count)
      print*,'Mechanisms in the gamma valley = ',i_count
      print*,'  '

      if(i_count.gt.0)then
         open(unit=31,file='gamma_table_renormalized',
     1                status='unknown')
         do i = 1,n_lev
            ee = float(i)*de
            write(31,32)ee,(scatt_table(i,k,1),k=1,10)
32          format(2X,11(F8.3,2X))
         enddo
      endif
      close(31)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     CREATE TABLE FOR THE L VALLEY
C     Scattering mechanism:   - acoustic phonons
C                             - Coulomb
C                             - polar optical phonons
C                             - L-to-gamma intervalley
C                             - L-to-L intervalley
C                             - L-to-X intervalley
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      iv = 2
      i_count = 0

C     Acoustic phonons scattering rate

      if(acoustic_L.eq.1)then
         sigma = sigma_L
         out_file_1 = 'acoustic_L'
         call acoustic_rate(i_count,iv,n_lev,out_file_1)
      endif

C    Coulomb scattering rate - Brooks-Herring approach

      if(Coulomb_scattering.eq.1)then
         out_file_1 = 'Coulomb_L'
         call Coulomb_BH(i_count,iv,n_lev,out_file_1)
      endif

C     Polar optical phonons scattering rate

      if(polar_L.eq.1)then
         w0 = polar_en_L
         out_file_1 = 'polar_L_ab'
         out_file_2 = 'polar_L_em'
         call polar_rate(i_count,iv,n_lev,w0,
     1                   out_file_1,out_file_2)
      endif

C    Intervalley scattering: L to gamma valley

      if(intervalley_L_gamma.eq.1)then
         w0 = phonon_L_gamma
         coupling_constant = DefPot_L_gamma
         delta_fi = - split_L_gamma
         final_valleys = eq_valleys_gamma
         i_final = 1   ! final_valley
         out_file_1 = 'intervalley_L_gamma_ab'
         out_file_2 = 'intervalley_L_gamma_em'
         call intervalley(i_count,iv,n_lev,w0,
     1                    out_file_1,out_file_2)
      endif

C    Intervalley scattering: L to L valley
```

```
      if(intervalley_L_L.eq.1)then
         w0 = phonon_L_L
         coupling_constant = DefPot_L_L
         delta_fi = 0.
         final_valleys = eq_valleys_L - 1.
         i_final = 2   ! final_valley
         out_file_1 = 'intervalley_L_L_ab'
         out_file_2 = 'intervalley_L_L_em'
         call intervalley(i_count,iv,n_lev,w0,
     1                    out_file_1,out_file_2)
      endif

C    Intervalley scattering: L to X valley

      if(intervalley_L_X.eq.1)then
         w0 = phonon_L_X
         coupling_constant = DefPot_L_X
         delta_fi = split_X_gamma - split_L_gamma
         final_valleys = eq_valleys_X
         i_final = 3   ! final_valley
         out_file_1 = 'intervalley_L_X_ab'
         out_file_2 = 'intervalley_L_X_em'
         call intervalley(i_count,iv,n_lev,w0,
     1                out_file_1,out_file_2)
      endif

      call renormalize_table(iv,n_lev,i_count)
      print*,'Mechanisms in the L valley = ',i_count
      print*,'  '

      if(i_count.gt.0)then
         open(unit=31,file='L_table_renormalized',
     1                status='unknown')
         do i = 1,n_lev
            ee = float(i)*de
            write(31,33)ee,(scatt_table(i,k,2),k=1,10)
33          format(2X,11(F8.3,2X))
         enddo
      endif
      close(31)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     CREATE TABLE FOR THE X VALLEY
C     Scattering mechanism:   - acoustic phonons
C                             - Coulomb
C                             - polar optical phonons
C                             - X-to-gamma intervalley
C                             - X-to-L intervalley
C                             - X-to-X intervalley
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      iv = 3
      i_count = 0

C     Acoustic phonons scattering rate

      if(acoustic_X.eq.1)then
         sigma = sigma_X
         out_file_1 = 'acoustic_X'
         call acoustic_rate(i_count,iv,n_lev,out_file_1)
      endif

C    Coulomb scattering rate - Brooks-Herring approach

      if(Coulomb_scattering.eq.1)then
         out_file_1 = 'Coulomb_X'
         call Coulomb_BH(i_count,iv,n_lev,out_file_1)
      endif

C     Polar optical phonons scattering rate

      if(polar_X.eq.1)then
```

```
            w0 = polar_en_X
            out_file_1 = 'polar_X_ab'
            out_file_2 = 'polar_X_em'
            call polar_rate(i_count,iv,n_lev,w0,
     1                    out_file_1,out_file_2)
         endif

C    Intervalley scattering: X to gamma valley

         if(intervalley_X_gamma.eq.1)then
            w0 = phonon_X_gamma
            coupling_constant = DefPot_X_gamma
            delta_fi = - split_X_gamma
            final_valleys = eq_valleys_gamma
            i_final = 1   ! final_valley
            out_file_1 = 'intervalley_X_gamma_ab'
            out_file_2 = 'intervalley_X_gamma_em'
            call intervalley(i_count,iv,n_lev,w0,
     1                     out_file_1,out_file_2)
         endif

C    Intervalley scattering: X to L valley

         if(intervalley_X_L.eq.1)then
            w0 = phonon_X_L
            coupling_constant = DefPot_X_L
            delta_fi = split_L_gamma - split_X_gamma
            final_valleys = eq_valleys_L
            i_final = 2   ! final_valley
            print*,w0,coupling_const
            out_file_1 = 'intervalley_X_L_ab'
            out_file_2 = 'intervalley_X_L_em'
            call intervalley(i_count,iv,n_lev,w0,
     1                     out_file_1,out_file_2)
         endif

C    Intervalley scattering: X to X valley

         if(intervalley_X_X.eq.1)then
            w0 = phonon_X_X
            coupling_constant = DefPot_X_X
            delta_fi = 0.
            final_valleys = eq_valleys_X - 1.
            i_final = 3   ! final_valley
            out_file_1 = 'intervalley_X_X_ab'
            out_file_2 = 'intervalley_X_X_em'
            call intervalley(i_count,iv,n_lev,w0,
     1                     out_file_1,out_file_2)
         endif

         call renormalize_table(iv,n_lev,i_count)
         print*,'Mechanisms in the X valley = ',i_count
         print*,'  '

         if(i_count.gt.0)then
            open(unit=31,file='X_table_renormalized',
     1                status='unknown')
            do i = 1,n_lev
               ee = float(i)*de
               write(31,34)ee,(scatt_table(i,k,3),k=1,10)
34             format(2X,11(F8.3,2X))
            enddo
         endif
         close(31)

         return
         end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     Generic subroutine that renormalizes the scattering table
C     for a given valley
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      subroutine renormalize_table(iv,n_lev,i_count)

       common
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/table/scatt_table(4000,10,3)

      integer n_lev,iv,i_count

      max_scatt_mech(iv) = i_count

      if(max_scatt_mech(iv).ge.1)then

      if(max_scatt_mech(iv).gt.1)then
         do i = 2, max_scatt_mech(iv)
         do k = 1, n_lev
            scatt_table(k,i,iv) = scatt_table(k,i-1,iv) +
     1                            scatt_table(k,i,iv)
         enddo
         enddo
      endif

      i_max = max_scatt_mech(iv)
      tau = 0.
      do i = 1,n_lev
         if(scatt_table(i,i_max,iv).gt.tau)
     1                  tau = scatt_table(i,i_max,iv)
      enddo

      do i = 1, max_scatt_mech(iv)
      do k = 1, n_lev
         scatt_table(k,i,iv) = scatt_table(k,i,iv)/tau
      enddo
      enddo

      tau_max(iv) = 1./tau
      print*,'valley index = ',iv,'  tau_max =',tau_max
      print*,'  '

      endif

      return
      end
```

The subroutines for acoustic phonons, polar optical phonons, Coulomb and intervalley scattering are listed below. In these subroutines we specify whether the scattering mechanism is elastic or inelastic, and whether it is isotropic or anisotropic. This information will be latter used in the **free_flight_scatter()** and the **scatter_carrier()** routines. The calculated scattering rates in GaAs for the Gamma valley associated with each of these mechanisms is shown in Fig. 6, along with the normalized cumulative rate (bottom panel). The corresponding rates in the L and X valleys are shown in Fig. 7.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C    Subroutine for the calculation of acoustic phonons
C    scattering rate
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      subroutine acoustic_rate(i_count,iv,n_lev,out_file)
```

```
       common
     &/pi/pi,two_pi
     &/temp/tem,Vt
     &/fund_const/q,h,kb,am0,eps_0
     &/dri/qh
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/scatt_par2/flag_mech(10,3),i_valley(10,3)
     &/table/scatt_table(4000,10,3)
     &/density/density,sound_velocity
     &/acoustic/sigma

      integer i_count,iv,n_lev
      real kb
      character*30 out_file

C    Calculate constant

      c_l = density*sound_velocity*sound_velocity
      const = sqrt(2.)*kb*tem/pi/h/c_l
     1      * (am(iv)/h)*(sqrt(am(iv))/h*sqrt(q))*(qh*q)
     2      * sigma*sigma

C    Create scattering table

      i_count = i_count + 1
      open(unit = 10, file=out_file, status='unknown')
      write(10,*)'energy  ',out_file
      do i = 1, n_lev
         ee = de*float(i)
         fe = ee*(1.+af(iv)*ee)
         acoustic = const*sqrt(fe)*(1.+af2(iv)*ee)
         scatt_table(i,i_count,iv) = acoustic
         write(10,*)ee,acoustic
      enddo
      close(10)

      flag_mech(i_count,iv) = 1
      w(i_count,iv) = 0.
      i_valley(i_count,iv) = iv

      return
      end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     Subroutine for the calculation of the
C     POLAR OPTICAL PHONONS scattering rate
C     (absorption + emission)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      subroutine polar_rate(i_count,iv,n_lev,w0,
     1                  out_file_1, out_file_2)

       common
     &/pi/pi,two_pi
     &/temp/tem,Vt
     &/fund_const/q,h,kb,am0,eps_0
     &/dri/qh
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/scatt_par2/flag_mech(10,3),i_valley(10,3)
     &/table/scatt_table(4000,10,3)
     &/dielec_func/eps_high,eps_low

      integer i_count,iv,n_lev
      real kb,w0
      character*30 out_file_1, out_file_2

C     Calculate constant

      rnq = 1./(exp(w0/Vt)-1.)
      const = qh*qh*q*w0*sqrt(am(iv)/2./q)/4./pi
     1      * (1./eps_high - 1./eps_low)

C      (a) Scattering rate - absorption

      i_count = i_count + 1
      open(unit=10, file=out_file_1, status='unknown')
      write(10,*)'energy  ',out_file_1
      polar_ab = rnq*const

      do i = 1, n_lev
         ee = de*float(i)
         ef = ee + w0
         ge = ee*(1.+af(iv)*ee)
         gf = ef*(1.+af(iv)*ef)
         rnum = sqrt(ge) + sqrt(gf)
         denom = sqrt(ge) - sqrt(gf)
         A = (2*(1.+af2(iv)*ee)*(1.+af(iv)*ef)
     1     + af(iv)*(ge+gf))**2.
         B = - af2(iv)*sqrt(ge*gf)
     1     * (4.*(1.+af(iv)*ee)*(1.+af(iv)*ef)
     1     + af(iv)*(ge+gf))
         C = 4.*(1.+af(iv)*ee)*(1.+af(iv)*ef)
     1     * (1.+af2(iv)*ee)*(1.+af2(iv)*ef)
         A = 4.
         C = 4.
         B = 0.
         factor = (1.+af2(iv)*ef)/sqrt(ge)
     1          * (A*log(abs(rnum/denom))+B)/C
         absorption = polar_ab*factor
         scatt_table(i,i_count,iv) = absorption
         write(10,*)ee,absorption
      enddo
      close(10)

      flag_mech(i_count,iv) = 2
      w(i_count,iv) = w0
      i_valley(i_count,iv) = iv
```

```
C     (b) Scattering rate - emission

      i_count = i_count + 1
      open(unit=11, file=out_file_2, status='unknown')
      write(11,*)'energy  ',out_file_2
      polar_em = (1.+rnq)*const

      do i = 1, n_lev
         ee = de*float(i)
         ef = ee - w0
         if(ef.le.0)then
            emission = 0
         else
            ge = ee*(1.+af(iv)*ee)
            gf = ef*(1.+af(iv)*ef)
            rnum = sqrt(ge) + sqrt(gf)
            denom = sqrt(ge) - sqrt(gf)
            A = (2*(1.+af2(iv)*ee)*(1.+af(iv)*ef)
     1         + af(iv)*(ge+gf))**2.
            B = - af2(iv)*sqrt(ge*gf)
     1         * (4.*(1.+af(iv)*ee)*(1.+af(iv)*ef)
     1         + af(iv)*(ge+gf))
            C = 4.*(1.+af(iv)*ee)*(1.+af(iv)*ef)
     1         * (1.+af2(iv)*ee)*(1.+af2(iv)*ef)
            A = 4.
            C = 4.
            B = 0.
            factor = (1.+af2(iv)*ef)/sqrt(ge)
     1               * (A*log(abs(rnum/denom))+B)/C
            emission = polar_em*factor
         endif
         scatt_table(i,i_count,iv) = emission
         write(11,*)ee,emission
      enddo
      close(11)

      flag_mech(i_count,iv) = 2
      w(i_count,iv) = - w0
      i_valley(i_count,iv) = iv

      return
      end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     Subroutine for the calculation of COULOMB SCATTERING rate
C     Assumption ==> elastic scattering process
c     (Brooks-Herring approach)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       subroutine Coulomb_BH(i_count,iv,n_lev,out_file_1)

       common
     &/pi/pi,two_pi
     &/temp/tem,Vt
     &/fund_const/q,h,kb,am0,eps_0
     &/dri/qh
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/scatt_par2/flag_mech(10,3),i_valley(10,3)
     &/table/scatt_table(4000,10,3)
     &/dielec_func/eps_high,eps_low
     &/coulomb/doping_density,Energy_debye

      integer i_count,iv,n_lev
      real kb
      character*30 out_file_1

C     Calculate constants

      Debye_length = sqrt(eps_high*Vt/q/abs(doping_density))
      Energy_debye = hhm(iv)/Debye_length/Debye_length
      final_mass = am(iv)
      factor = Debye_length*Debye_length/eps_high
      const = doping_density*final_mass*qh
     1      * sqrt(2.*final_mass)/pi*qh
     2      * factor*qh*factor*qh*sqrt(q)

C     Calculate scattering rate:

      i_count = i_count + 1
      open(unit=10, file=out_file_1, status='unknown')
      write(10,*)'energy  ',out_file_1

      do i = 1, n_lev
         ee = de*float(i)
         ge = ee*(1.+af(iv)*ee)
         factor = sqrt(ge)*(1.+af2(iv)*ee)
     1          / (1.+4.*ge/Energy_debye)
         scatt_rate = const*factor
         scatt_table(i,i_count,iv) = scatt_rate
         write(10,*)ee,scatt_rate
      enddo
      close(10)

      flag_mech(i_count,iv) = 3
      w(i_count,iv) = 0.
      i_valley(i_count,iv) = iv

      return
      end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     Generic subroutine for the calculation of
C     INTERVALLEY PHONONS scattering rate
c     (absorption + emission)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       subroutine intervalley(i_count,iv,n_lev,w0,
     1                     out_file_1,out_file_2)

       common
     &/pi/pi,two_pi
     &/temp/tem,Vt
     &/fund_const/q,h,kb,am0,eps_0
     &/dri/qh
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/scatt_par2/flag_mech(10,3),i_valley(10,3)
     &/table/scatt_table(4000,10,3)
     &/density/density
     &/intervalley1/coupling_constant
     &/intervalley2/delta_fi,final_valleys,i_final

      integer i_count,iv,n_lev
      real kb, w0
      character*30 out_file_1, out_file_2

C    Calculate constants

      rnq = 1./(exp(w0/Vt)-1.)
      final_mass = am(i_final)
      const = final_valleys*(coupling_constant**2.)
     1      * q*sqrt(q)/(sqrt(2.)*pi*density*w0)
     2      * (final_mass/h)*sqrt(final_mass)/h

C     (a) Scattering rate - absorption

      i_count = i_count + 1
      open(unit=10, file=out_file_1, status='unknown')
      write(10,*)'energy  ',out_file_1
      ab = rnq*const

      do i = 1, n_lev
         ee = de*float(i)
         ef = ee + w0 - delta_fi
         gf = ef*(1.+af(i_final)*ef)
         if(ef.le.0)then
            absorption = 0.
         else
            factor = sqrt(gf)*(1.+af2(i_final)*ef)
            absorption = ab*factor
         endif
         scatt_table(i,i_count,iv) = absorption
         write(10,*)ee,absorption
      enddo
      close(10)

      flag_mech(i_count,iv) = 1
      w(i_count,iv) = w0 - delta_fi
      i_valley(i_count,iv) = i_final

C     (b) Scattering rate - emission

      i_count = i_count + 1
      open(unit=11, file=out_file_2, status='unknown')
      write(11,*)'energy  ',out_file_2
```

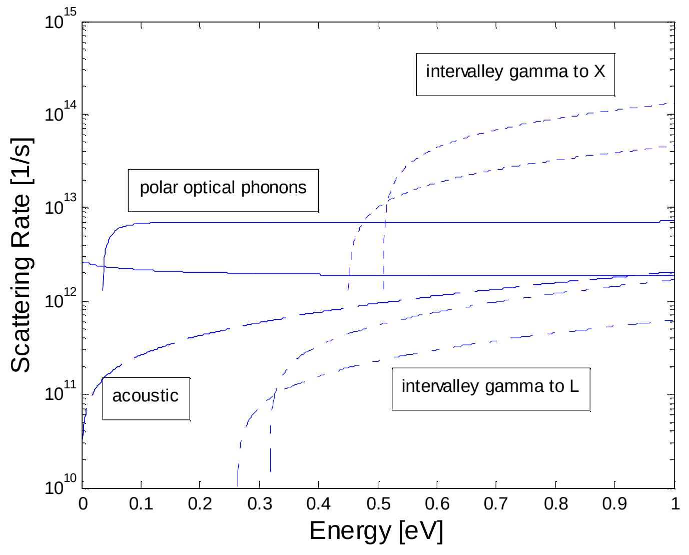
```
em = (1.+rnq)*const

do i = 1, n_lev
   ee = de*float(i)
   ef = ee - w0 - delta_fi
   gf = ef*(1.+af(i_final)*ef)
   if(ef.le.0)then
      emission = 0.
   else
      factor = sqrt(gf)*(1.+af2(i_final)*ef)
      emission = em*factor
   endif
   scatt_table(i,i_count,iv) = emission
   write(11,*)ee,emission
enddo
close(11)

flag_mech(i_count,iv) = 1
w(i_count,iv) = - w0 - delta_fi
i_valley(i_count,iv) = i_final

return
end
```

Having constructed the scattering table and after renormalizing the table, examples of which are given in Figures 6 and Figure 7 for the Γ, L, and X valley, the next step is to initialize carriers wavevector and energy and the initial free-flight time. This is accomplished by calling the subroutine **init()**. Energy and wavevector histograms of the initial carrier energy and the components of the wave-vector along the x-, y-, and z-axes are shown in Figure 8. Here the number of particles simulated is 10000, and one can see the statistical fluctuation of these average quanties associated with the finite number of particles. Notice that the initial y-component for the wavevector is symmetric around the y-axis which means that the average wavevector along the y-axis is zero, which should be expected since the electric field along the y-component is zero at t=0. Identical distributions have been obtained for the x- and for the z-components of the wavevector. Also note that the energy distribution has the Maxwell-Boltzmann form as it should be expected. One can also estimate from this graph that the average energy of the carriers is on the order of $(3/2)k_BT$.



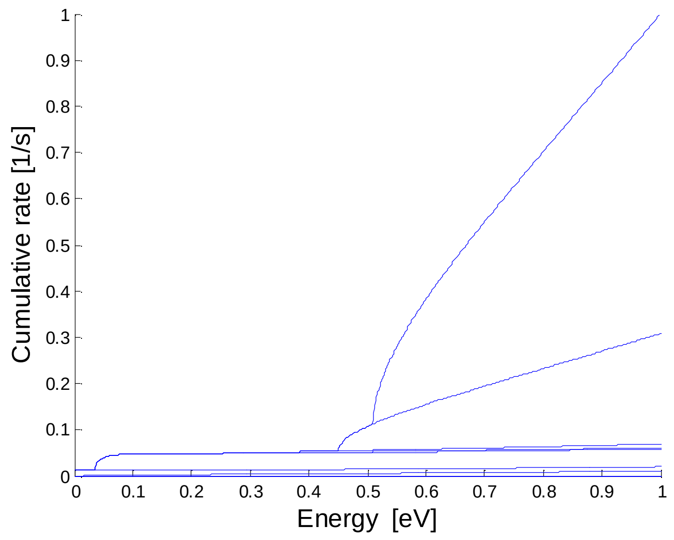
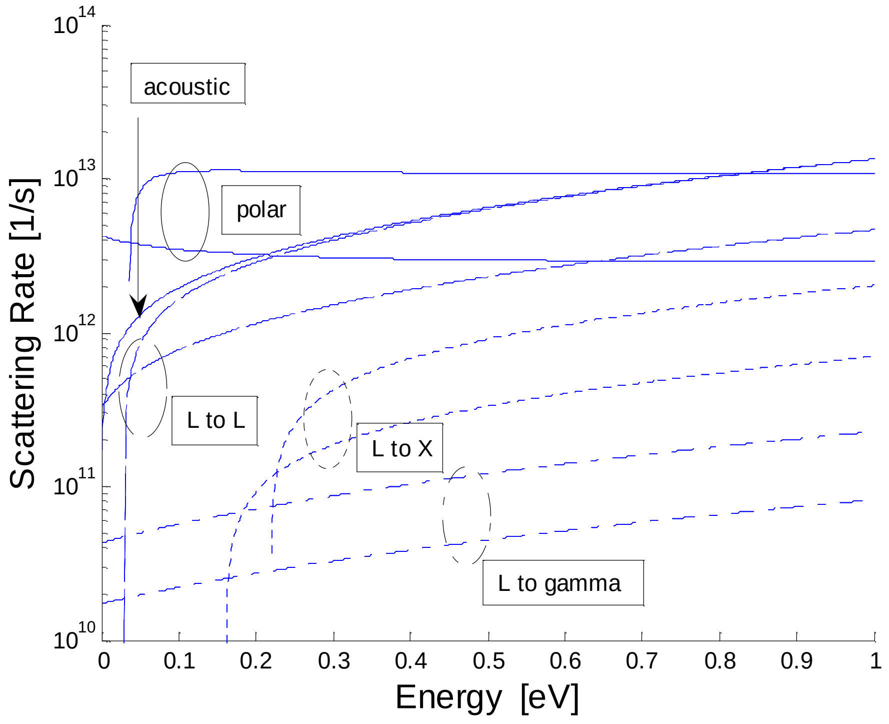


Figure 6. Top panel: scattering rates for the Γ-valley. For simplicity we have omitted Coulomb scattering in these calculations. In the top figure, the dashed line corresponds to the acoustic phonon scattering rate, solid lines correspond to polar optical phonon scattering (absorption and emission), and the dashed-dotted line corresponds to intervalley scattering from Γ-valley to L-valley. Since the L-valley is along the [111] direction, there are 8 equivalent directions and since these valleys are shared there are a total of 4 equivalent L valleys. The dotted line corresponds to scattering from the Γ-valley to X-valleys. The X-valleys are at the [100] direction and since there are 6 equivalent [100] directions and the valleys are shared between Brillouin zones, there are 3 equivalent X valleys. Bottom panel: normalized cumulative scattering table for the Γ-valley. Everything above the top line up to Γ=1 is self-scattering so it is advisable when checking the scattering mechanisms to first check whether the scattering mechanism chosen is self-scattering or not. This is in particular important for energies below 0.5 eV for this particular scattering table when the Γ to X intervalley scattering (absorption and emission) takes over.



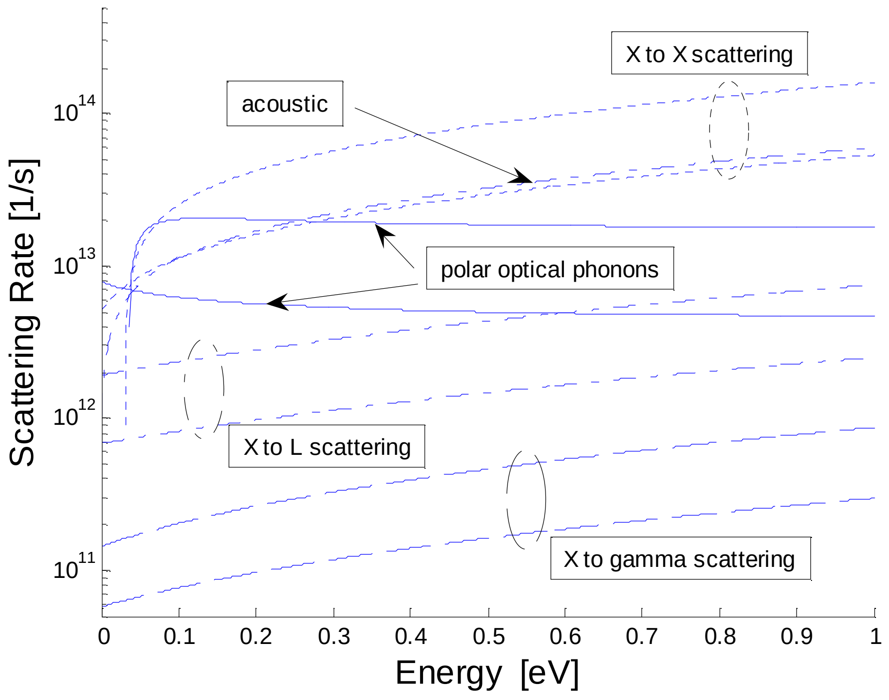
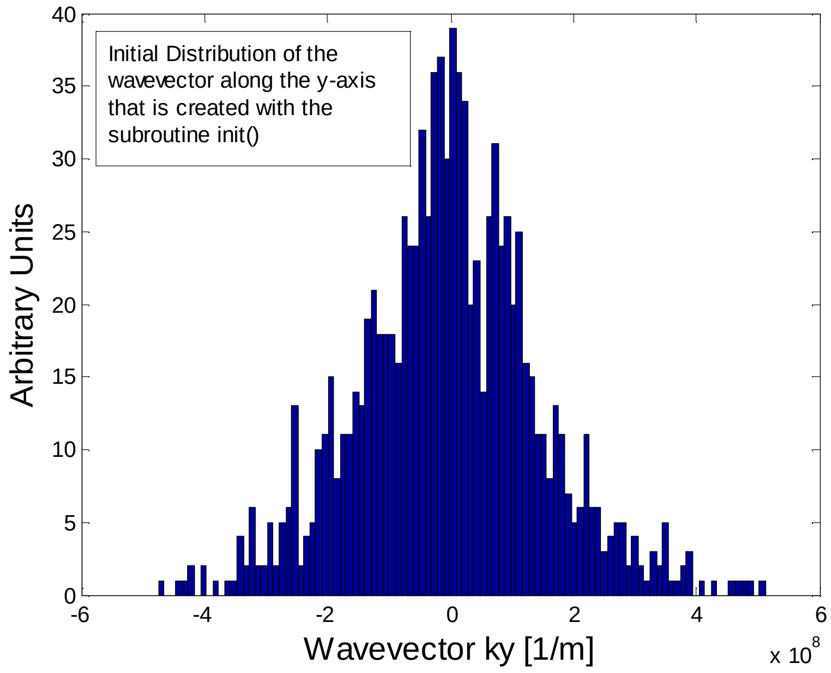


Figure 7. Scattering rates for the L (top panel) and X (bottom panel) valleys used to create the corresponding normalized scattering tables (not shown here).



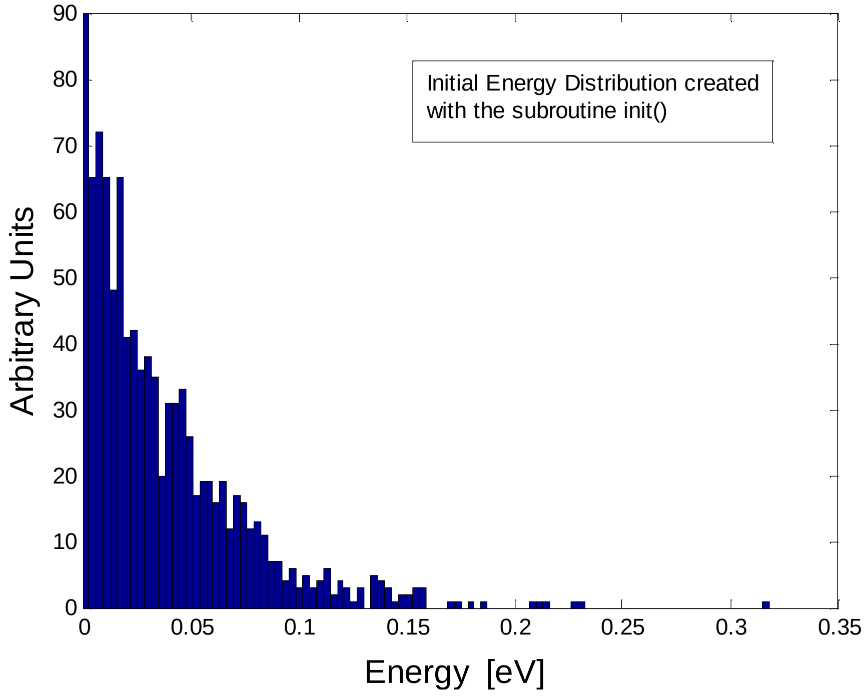


Figure 8. Initial carrier distribution for an ensemble of 10000 Particles. Top panel: distribution of wavevector k_y. Bottom panel: energy distribution.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C       INITIALIZE CARRIER ENERGY AND WAVEVECTOR ACCORDING TO THE
C       MAXWELL-BOLTZMANN STATISTICS
C
C       Assumption:  All carriers are initially in valley 1 for GaAs
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       subroutine init(nsim)

       common
     &/ran_var/iso
     &/pi/pi,two_pi
     &/temp/tem,Vt
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/variables/p(20000,7),ip(20000),energy(20000)

       real k,kx,ky,kz,e, sume
       integer iv, nsim

       sume = 0.

       do i = 1, nsim

       e = -(1.5*Vt)*log(ran(iso))
       sume = sume + e

c      Initial valley index

       rr = 3.*ran(iso)
       if(rr.le.1.)then
          iv=1
       elseif(rr.le.2.)then
      iv=2
       elseif(rr.le.3.)then
          iv=3
       endif
       iv=1            ! this is for GaAs materials

C      Initial wavevector

       k=smh(iv)*sqrt(e*(1.+af(iv)*e))
       fai=two_pi*ran(iso)
       ct=1.-2.*ran(iso)
       st=sqrt(1.-ct*ct)
       kx=k*st*cos(fai)
       ky=k*st*sin(fai)
       kz=k*ct

C      Initial free-flight

103    rr = ran(iso)
       if(rr.le.1.e-5)go to 103
       tc = -(log(rr))*tau_max(iv)

C      Map particle atributes

       p(i,1) = kx
       p(i,2) = ky
       p(i,3) = kz
       p(i,4) = tc
       ip(i) = iv
       energy(i) = e

       enddo

       sume = sume/float(nsim)
       print*,'Average carrier energy - initial distribution'
       print*,'Energy = ',sume
       print*,'   '

       return
       end
```

When the initialization process is finished, the main free-flight-scatter procedure that is executed within the **free_flight_scatter()** subroutine. There are two components in this routine; first the carriers accelerate freely due to the electric field, accomplished by calling the **drift()** subroutine, and then their free-flights are interrupted by random scattering events that are managed by the **scatter_carrier()** subroutine. The flow-chart for performing the free-flight-scatter process within one time step Δt is hown diagrammatically in Fig. 9 below. The actual **free_flight_scatter()**, **drift()** and **scatter_carrier()** subroutines are given afterwards.

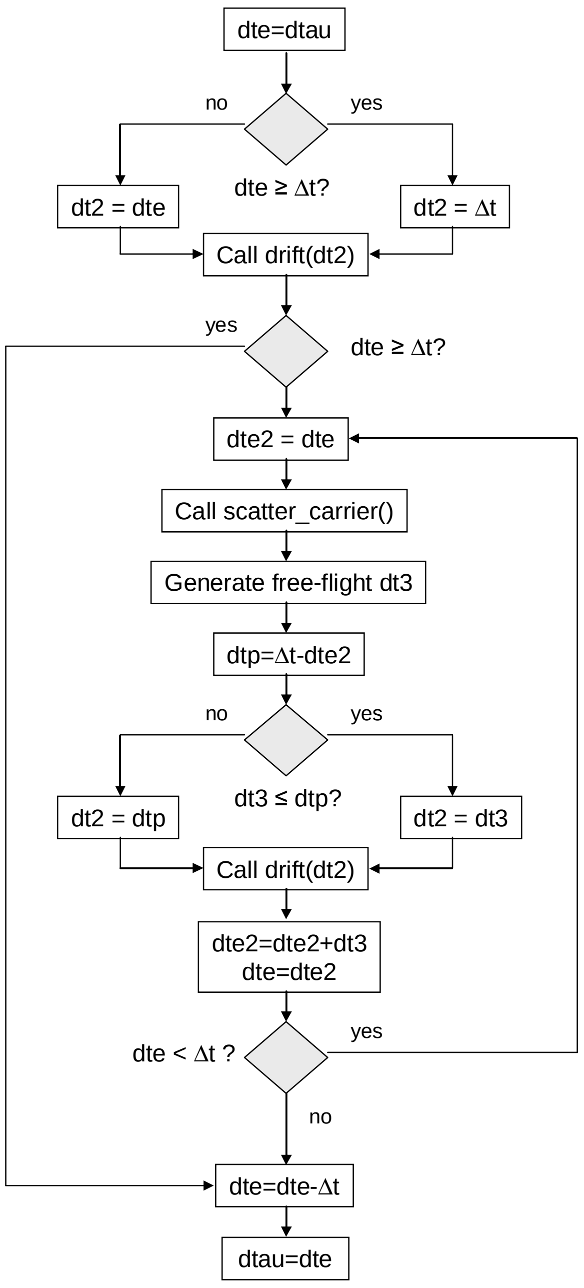


Figure 9 . Free-flight-scatter procedure within one time step.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     PERFORM THE FREE-FLIGHT AND SCATTER
C     PART WITHIN ONE TIME INTERVAL
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      subroutine free_flight_scatter(n_lev,nsim)
```

```
       common
      &/ran_var/iso
      &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
      &/time_1/dt,dtau,tot_time
      &/variables/p(20000,7),ip(20000),energy(20000)
      &/particle_atr/kx,ky,kz,iv,e
      &/frequency/freq(10,3)

       integer n_lev,nsim
       integer iv
       real kx,ky,kz,e

C     Reset counter for scattering frequency
       do i = 1,10
       do j = 1,3
          freq(i,j) = 0.
       enddo
       enddo

       do i = 1, nsim    ! loop for all carriers

C     Inverse mapping of particle atributes

       kx = p(i,1)
       ky = p(i,2)
       kz = p(i,3)
       dtau = p(i,4)
       iv = ip(i)
       e = energy(i)

C     Initial free-flight of the carriers

       dte = dtau
       if(dte.ge.dt)then
          dt2=dt
       else
          dt2=dte
       endif
       call drift(dt2)
       if(dte.gt.dt)goto 401

C     Free-flight and scatter part

402   dte2=dte
       call scatter_carrier(n_lev)
219   rr=ran(iso)
       if(rr.le.1e-6) go to 219
       dt3=-(log(rr))*tau_max(iv)
       dtp = dt - dte2    ! remaining time to scatter in dt-interval
       if(dt3.le.dtp)then
          dt2 = dt3
       else
          dt2 = dtp
       endif
       call drift(dt2)

c     Update times

       dte2 = dte2 + dt3
       dte = dte2
       if(dte.lt.dt)goto 402

401   dte = dte - dt
       dtau = dte

C     Map particle atributes

       p(i,1) = kx
       p(i,2) = ky
       p(i,3) = kz
       p(i,4) = dtau
       ip(i) = iv
       energy(i) = e

       enddo

       return
       end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     PERFORM THE K-SPACE AND REAL-SPACE MOTION OF THE CARRIERS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       subroutine drift(tau)

       common
      &/pi/pi,two_pi
      &/dri/qh
      &/temp/tem,Vt
      &/ek/am(3),smh(3),hhm(3)
      &/nonp/af(3),af2(3),af4(3)
      &/force/fx,fy,fz
      &/particle_atr/kx,ky,kz,iv,e

       real kx,ky,kz,k,e,tau
       real qh1,dkx,dky,dkz
       real fx,fy,fz
       integer iv

       qh1 = qh*tau
       dkx = -qh1*fx
       dky = -qh1*fy
       dkz = -qh1*fz

       kx = kx+dkx
       ky = ky+dky
       kz = kz+dkz

       skx = kx*kx
       sky = ky*ky
       skz = kz*kz
       sk = skx+sky+skz
       k = sqrt(sk)
       gk = hhm(iv)*sk
       e = 2*gk/(1.+sqrt(1.+af4(iv)*gk))

       return
       end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C     SELECT SCATTERING MECHANISM AND PERFORM
C     THE SCATTERING PART THAT MODIFIES PARTICLE ATRIBUTES
C     (kx, ky, kz, iv, energy)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       subroutine scatter_carrier(n_lev)

       common
      &/ran_var/iso
      &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
      &/scatt_par2/flag_mech(10,3),i_valley(10,3)
      &/table/scatt_table(4000,10,3)
      &/particle_atr/kx,ky,kz,iv,e
      &/frequency/freq(10,3)

       integer n_lev,iv,i_final
       real kx,ky,kz,e

C     Calculate index to the scattering table

       loc = e/de
       if(loc.eq.0)loc=1
       if(loc.gt.n_lev)loc=n_lev
```

```
C     Select scattering mechanism

      i_top = max_scatt_mech(iv)
      rr = ran(iso)
      if(rr.ge.scatt_table(loc,i_top,iv))then
         freq(i_top+1,iv)=freq(i_top+1,iv)+1
         goto 222  ! self-scattering
      endif
      if(rr.lt.scatt_table(loc,1,iv))then
         i_fix = 1
         freq(i_fix,iv)=freq(i_fix,iv)+1
         goto 111
      endif
      if(i_top.gt.1)then
        do i=1,i_top-1
           bound_lower = scatt_table(loc,i,iv)
           bound_upper = scatt_table(loc,i+1,iv)
           if(rr.ge.bound_lower.and.rr.lt.bound_upper)then
              i_fix = i + 1
              freq(i_fix,iv)=freq(i_fix,iv)+1
              goto 111
           endif
        enddo
      endif

111   continue

C     Perform scattering (change energy and randomize momentum)

      select_mech = flag_mech(i_fix,iv)
      i_final = i_valley(i_fix,iv)
      if(select_mech.eq.1)then
         call isotropic(i_fix,i_final)
      elseif(select_mech.eq.2)then
         call polar_optical_angle(i_fix)
      elseif(select_mech.eq.3)then
         call Coulomb_angle_BH()
      endif
      iv = i_final

222   continue

      return
      end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
```

```
C       SCATTERING SUBROUTINES
C       (CHANGE ENERGY AND WAVEVECTORS OF PARTICLES)
C
C       In the definition of the scattering rates, a variable called
C       'flag_mech' has been defined.  The values assigned to this
C       variable correspond to:
C
C       1  ==>  Isotropic scattering (acoustic, intervalley)
C       2  ==>  Polar_optical ==> anisotropic scattering (small angle)
C       3  ==>  Coulomb scattering ==> small-angle scattering
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C       ISOTROPIC SCATTERING PROCESS
C       uniform probability density for scattering in all directions
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        subroutine isotropic(i_fix,i_final)
        common
     &/ran_var/iso
     &/pi/pi,two_pi
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/particle_atr/kx,ky,kz,iv,e

      integer iv,i_final
      real kx,ky,kz,e

C     Update carrier energy
      e = e + w(i_fix,iv)

C     Update carrier wavevector
      rknew = smh(i_final)*sqrt(e*(1.+af(i_final)*e))
      fi = two_pi*ran(iso)
      ct = 1.-2.*ran(iso)
      st = sqrt(1.-ct*ct)
      kx = rknew*st*cos(fi)
      ky = rknew*st*sin(fi)
      kz = rknew*ct

      return
      end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      POLAR OPTICAL PHONONS SCATTERING ANGLE
C      Randomize the polar angle according to the notes
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      subroutine polar_optical_angle(i_fix)

       common
     &/ran_var/iso
     &/pi/pi,two_pi
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/particle_atr/kx,ky,kz,iv,e

      integer iv,i_fix
      real kx,ky,kz,e
      real k,kxy
      real kxp,kyp,kzp,kp

C     Update carrier energy
      enew = e + w(i_fix,iv)

C     Calculate the rotation angles
      kxy = sqrt(kx*kx+ky*ky)
      k = sqrt(kxy*kxy+kz*kz)
      cth0 = kz/k
      sth0 = kxy/k
      cfi0 = kx/kxy
      sfi0 = ky/kxy

C     Randomize momentum in the rotated coordinate system
      kp = smh(iv)*sqrt(enew*(1.+af(iv)*enew))
      ge = e*(1.+af(iv)*e)
      gnew = enew*(1.+af(iv)*enew)
      zeta = 2.*sqrt(ge*gnew)/(ge+gnew-2.*sqrt(ge*gnew))
      rr = ran(iso)
      cth = ((zeta+1.)-(2.*zeta+1)**rr)/zeta
      sth = sqrt(1.-cth*cth)
      fi = two_pi*ran(iso)
      cfi = cos(fi)
      sfi = sin(fi)
      kxp = kp*sth*cfi
      kyp = kp*sth*sfi
      kzp = kp*cth

C     Return back to the original coordinate system
      kx = kxp*cfi0*cth0-kyp*sfi0+kzp*cfi0*sth0
      ky = kxp*sfi0*cth0+kyp*cfi0+kzp*sfi0*sth0
      kz = -kxp*sth0+kzp*cth0

      e = enew

      return
      end


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     COULOMB SCATTERING ANGLE
C     Randomize the polar angle
C     The assumption is that the scattering process is elastic
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      subroutine  Coulomb_angle_BH()

       common
     &/ran_var/iso
     &/pi/pi,two_pi
     &/ek/am(3),smh(3),hhm(3)
     &/nonp/af(3),af2(3),af4(3)
     &/scatt_par/emax,de,w(10,3),tau_max(3),max_scatt_mech(3)
     &/coulomb/doping_density,Energy_debye
     &/particle_atr/kx,ky,kz,iv,e

      integer iv,i_fix
      real kx,ky,kz,e
      real k,kxy
      real kxp,kyp,kzp

C     Update carrier energy
c      enew = e + w(i_fix,iv)

C     Calculate the rotation angles
      kxy = sqrt(kx*kx+ky*ky)
      k = sqrt(kxy*kxy+kz*kz)
      cth0 = kz/k
      sth0 = kxy/k
      cfi0 = kx/kxy
      sfi0 = ky/kxy

C     Randomize momentum in the rotated coordinate system
      ge = e*(1.+af(iv)*e)
      rr = ran(iso)
      cth = 1. - 2*rr/(1.+4.*(1-rr)*ge/Energy_debye)
      sth = sqrt(1.-cth*cth)
      fi = two_pi*ran(iso)
      cfi = cos(fi)
      sfi = sin(fi)
      kxp = k*sth*cfi
      kyp = k*sth*sfi
      kzp = k*cth

C     Return back to the original coordinate system
      kx = kxp*cfi0*cth0-kyp*sfi0+kzp*cfi0*sth0
      ky = kxp*sfi0*cth0+kyp*cfi0+kzp*sfi0*sth0
      kz = -kxp*sth0+kzp*cth0

c     e = enew

      return
      end
```

In the **scatter_carrier()** subroutine, first the scattering mechanism terminating the free flight is chosen, to which certain attributes are associated such as the change in energy after scattering. For inelastic scattering processes, we have the change in energy due to emission or absorption of phonons, for example. Also, the nature of the scattering process is identified: isotropic or anisotropic. Note that when performing acoustic phonon and intervalley scattering for GaAs, both of which are isotropic scattering processes, no coordinate system transformation is needed to determine the final wavevector after scattering. Because polar optical phonon and Coulomb scattering mechanisms are anisotropic, it is necessary to do a rotation of the coordinate system, scatter the carrier in the rotated system and then perform inverse coordinate transformation. This procedure is needed because it is much easier to determine final carrier momentum in the rotated coordinate system in which the initial wavevector k is aligned with the z-axis. For this case, one can calculate that the final polar angle for scattering with polar optical phonons for parabolic bands in the rotated coordinate system is

$$\cos\theta = \frac{(1+\xi)-(1+2\xi)^r}{\xi}, \quad \xi = \frac{2\sqrt{E_k\left(E_k \pm \hbar\omega_0\right)}}{\left(\sqrt{E_k}-\sqrt{E_k \pm \hbar\omega_0}\right)^2} \tag{14}$$

where E_k is the carrier energy, $\hbar\omega_0$ is the polar optical phonon energy and r is a random number uniformly distributed between 0 and 1. The final angle for scattering with ionized impurities (Coulomb scattering) and for parabolic bands is

$$\cos\theta = 1-\frac{2r}{1+4k^2L_D^2(1-r)} \tag{15}$$

where **k** is the carrier wavevector, and L_D is the Debye screening length discussed in Chapter 4 in conjunction with drift-diffusion modeling. The azimuthal angle for both scattering processes is simply calculated using $\varphi = 2\pi r$. The importance of properly calculating the angle θ after scattering to describe small angle deflections in the case of Coulomb or polar optical phonon scattering is illustrated in Figure 10 (from 0 to π=3.141592654) where we plot the histogram of the polar angle after scattering for electron-polar optical phonon scattering, where we can clearly see the preference for small angle deflections that are characteristic for any Coulomb type interaction (polar optical phonon is in fact electron-dipole interaction). Graphical representation of the determination of the final angle after scattering for both isotropic and anisotropic scattering processes is given in Figure 11.

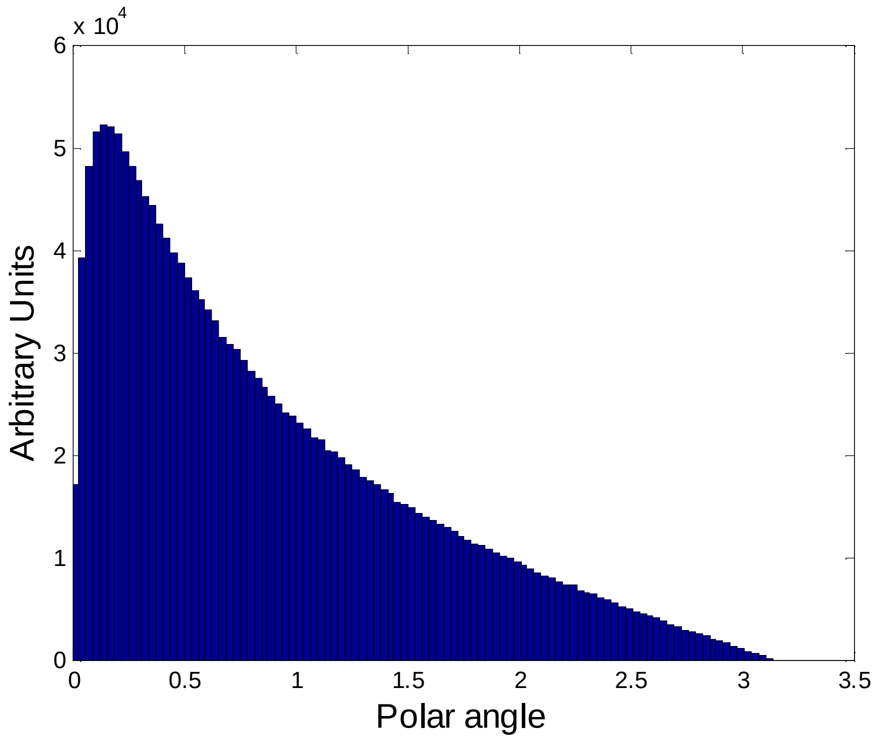
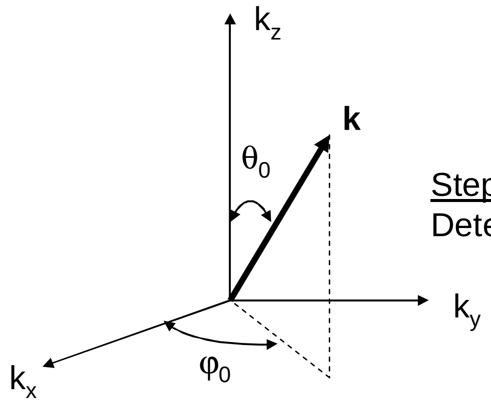


Figure 10. Histogram of the polar angle for electron – polar optical phonon scattering.

1. Isotropic scattering processes

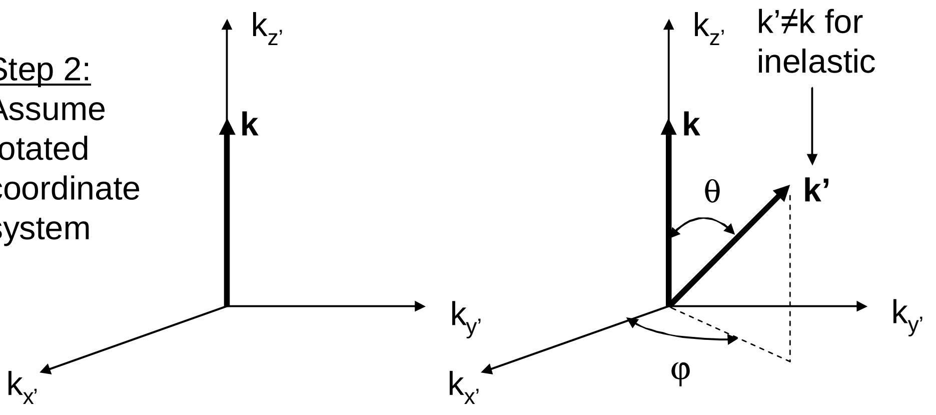
$$\cos\theta = 1 - 2r, \quad \varphi = 2\pi r$$

2. Anisotropic scattering processes (Coulomb, POP)



Step 1:
Determine θ_0 and φ_0

Step 2:
Assume rotated coordinate system



Step 3:
perform scattering

$$\cos\theta = \frac{(1+\xi)-(1+2\xi)^r}{\xi}, \quad \xi = \frac{2\sqrt{E_k\left(E_k \pm \hbar\omega_0\right)}}{\left(\sqrt{E_k}-\sqrt{E_k \pm \hbar\omega_0}\right)^2} \quad \text{POP}$$

$$\cos\theta = 1 - \frac{2r}{1+4k^2L_D^2(1-r)} \quad \text{Coulomb}$$

$\varphi = 2\pi r$ for both

Step 4:

$k_{xp} = k'\sin\theta\cos\varphi$, $k_{yp} = k'\sin\theta * \sin\varphi$, $k_{zp} = k'\cos\theta$

Return back to the original coordinate system:

$k_x = k_{xp}\cos\varphi_0\cos\theta_0 - k_{yp}\sin\varphi_0 + k_{zp}\cos\varphi_0\sin\theta_0$
$k_y = k_{xp}\sin\varphi_0\cos\theta_0 + k_{yp}\cos\varphi_0 + k_{zp}\sin\varphi_0\sin\theta_0$
$k_z = -k_{xp}\sin\theta_0 + k_{zp}\cos\theta_0$

Figure 11. Description of final angle selection for isotropic and anisotropic scattering processes using the direct technique.

The direct technique described above can be applied when the integrals describing cosθ can be analytically calculated. For most cases of interest, the integral cannot be easily inverted. In these cases a rejection technique may be employed. The procedure of the rejection technique goes as follows:

(1) Choose a maximum value *C*, such that $C > f(x)$ for all *x* in the interval (*a*,*b*).

(2) Choose pairs of random numbers are chosen, one between *a* and *b* ($x_1 = a + r_1(b-a)$) and another $f_1 = r_1'C$ between 0 and C, where r_1 and r_1' are random numbers uniformly distributed between zero and 1.

(3) If $f_1 \le f(x_1)$, then the number x_1 is accepted as a suitable value, otherwise it is rejected.

The three steps described above are schematically shown in the figure below (Figure 12). For $x = x_1$, r_1C is larger than $f(x_1)$ and in this case if this represents the final polar angle for scattering, this angle is rejected and a new sequence of two random numbers is generated to determine x_2 and r_2C. In this second case, $f(x_2)>r_2C$ and the polar angle $\theta=x_2$ is selected (for polar angle selection $a = 0$ and $b=\pi$).

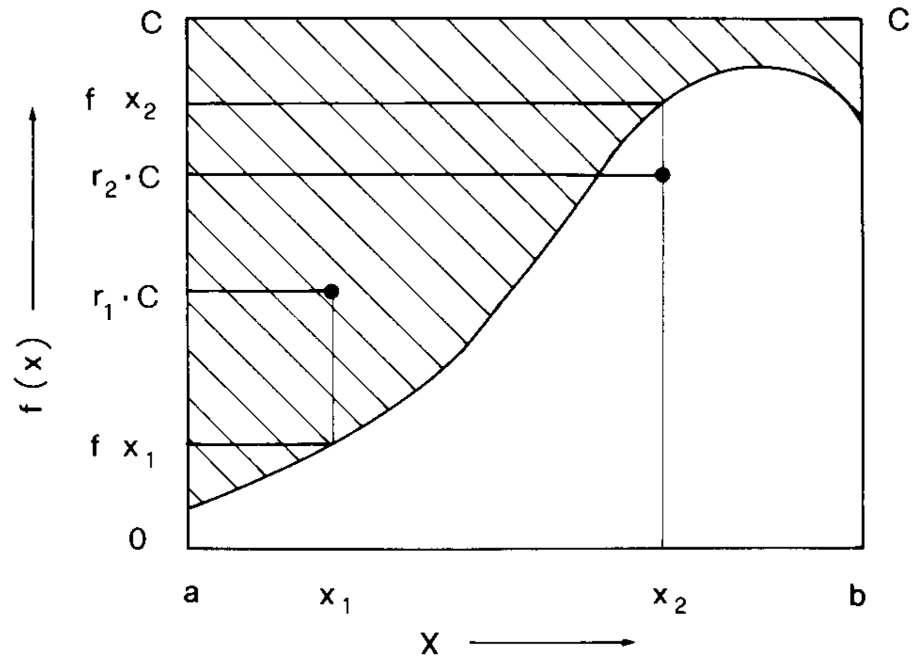


Figure 12. Schematic description of the rejection technique.

After the simulation is completed, typical results to check are the velocity-time, the energy-time and the valley occupation versus time characteristics, such as those shown in Figure 13, where the velocity time characteristics for applied electric fields ranging from 0.5 to 7 kV/cm, with an electric field increment of 0.5 kV/cm, are shown. These clearly demonstrate that after a transient phase, the system reaches a stationary steady state, after which time we can start taking averages for calculating steady-state quantities.

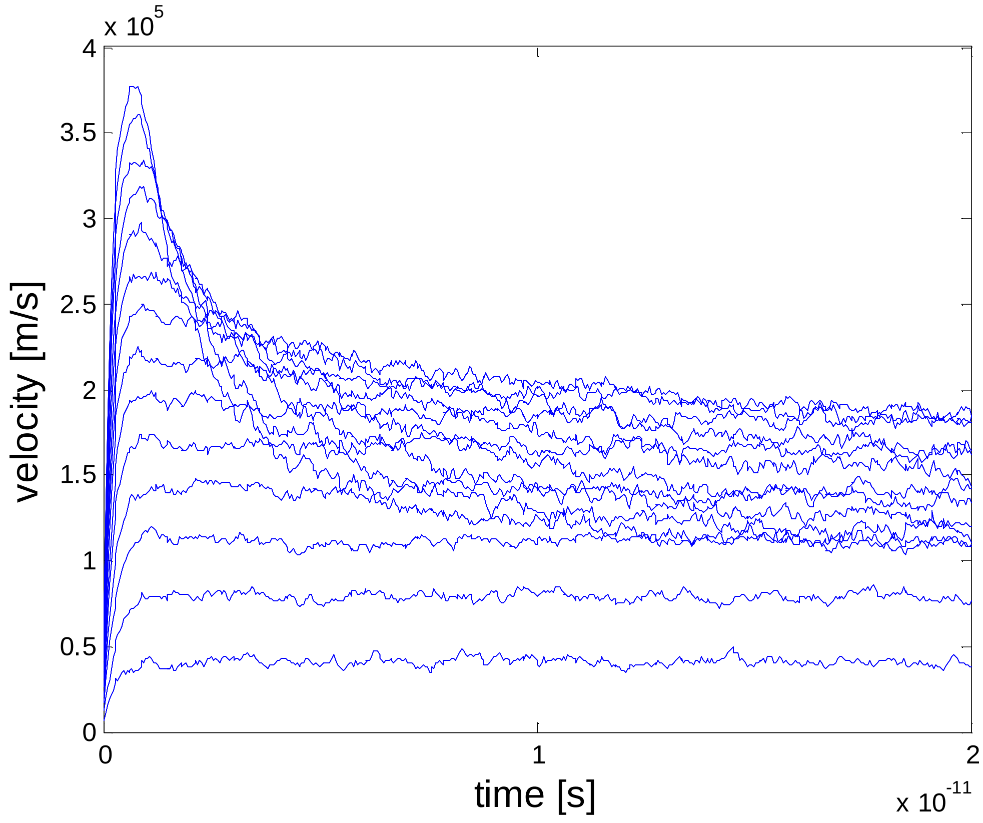


Figure 13. Time evolution of the drift velocity for electric field strengths ranging between 0.5 and 7 kV/cm, in 0.5 kV/cm increments.

From the results shown in Figure 13, one can see that steady-state is achieved for larger time intervals when the electric field value is increased and the carriers are still sitting in the Γ-valley. Afterwards the time needed to get to steady-state decreases. This trend is related to the valley repopulation and movement of the carriers from the Γ, into the X and finally into the L valley. The steady-state velocity-field and valley population versus electric field characteristics are shown in Figures 14 and 15, respectively. One can clearly see on the velocity-field characteristics that a low-field mobility of about 8000 cm2/V-s is correctly reproduced for GaAs without the use of any adjustable parameters.

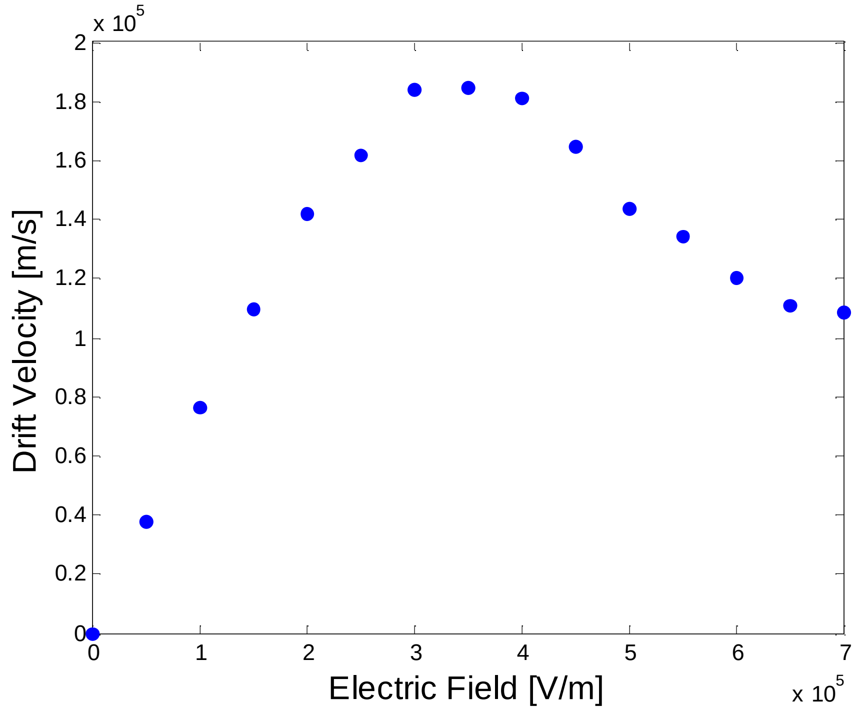


Figure 14. Steady state drift velocity vs. electric field.

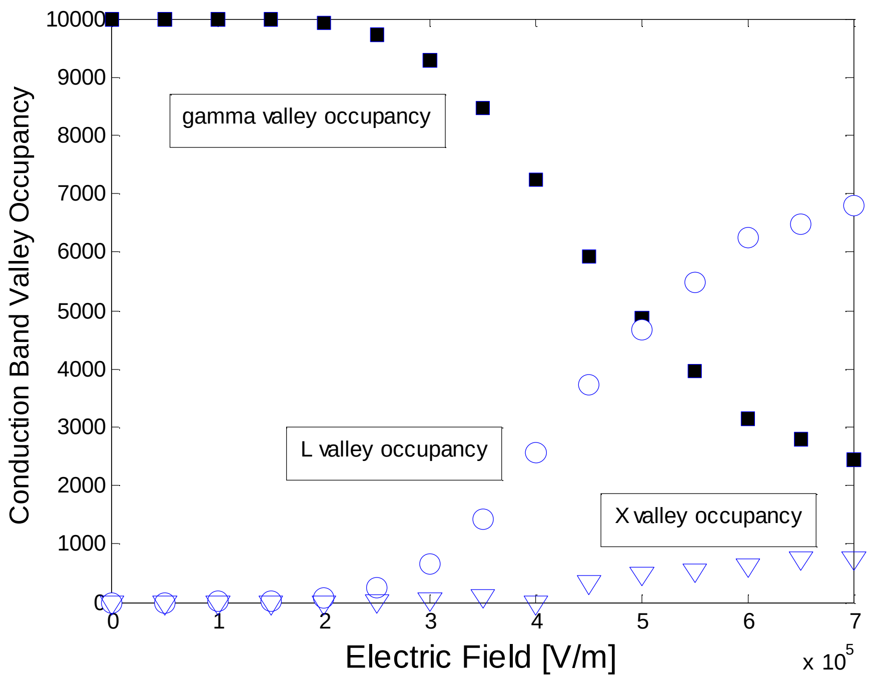
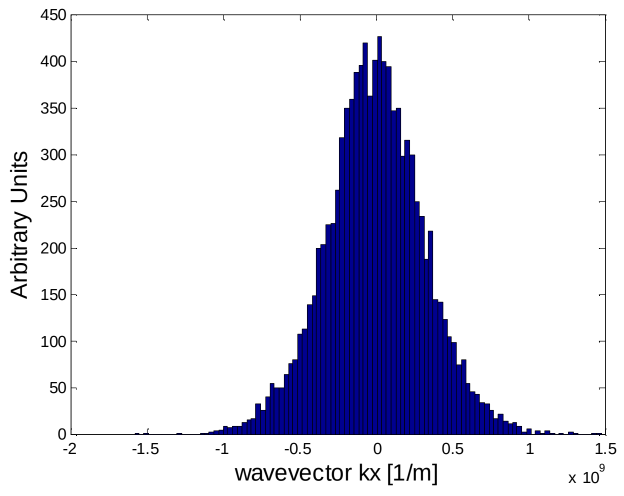
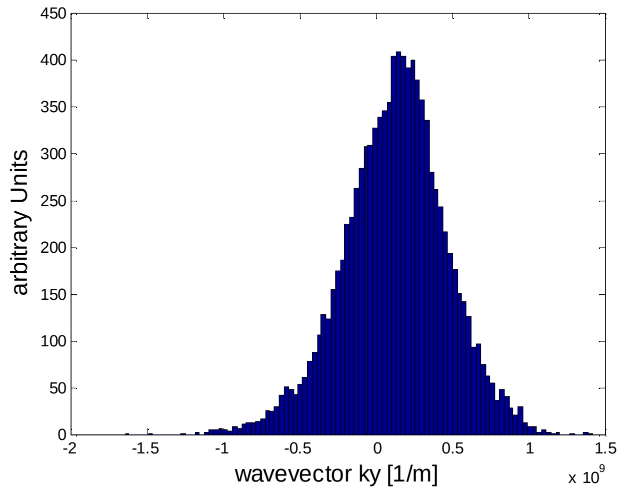


Figure 15. Different valley occupancy vs. electric field.

At this point, it is advisable to check the energy and wavevector histograms (Figure 16) to ensure that the energy range chosen in the scattering tables is correct or not for the particular maximum electric field strength being considered, which gives the worst case scenario. Since, as already noted, we apply the electric field in the y-direction, for comparative purposes we plot the histograms of the x-component of the wavevector, y-component of the wavevector, and the histogram of the final carrier energy distribution for which a drifted Maxwellian form is evident. Since there is no field applied in the x-direction, we see that the average wavevector in the x-direction is 0. Due to the application of the field in the y-direction, there is a finite positive shift in the y-component of the velocity, which is yet another signature for the displaced Maxwellian form of the energy distribution in the bottom histogram.





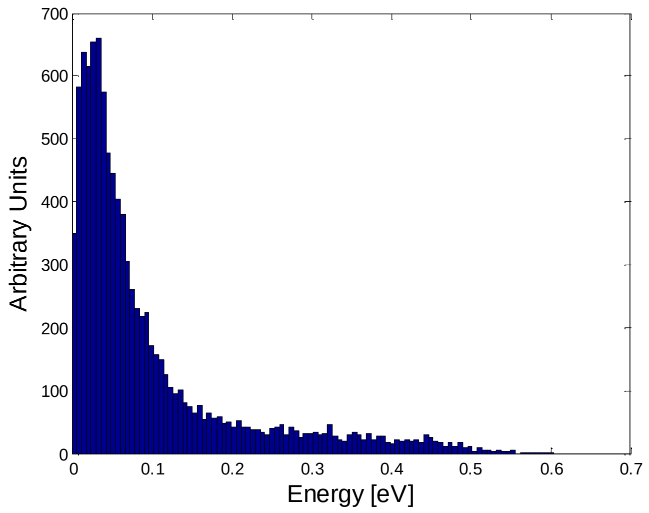


Figure 16. Top panel: histogram of the x-component of the wavevector. Middle panel: Histogram of the y-component of the wavevector. Bottom panel: histogram of the carrier energy. Applied electric field is 7 kV/cm.

References

[i] C. Jacoboni and L. Reggiani, *Rev. Mod. Phys.*, 55 (1983) 645.

[ii] C. Jacoboni and P. Lugli, *The Monte Carlo Method for Semiconductor Device Simulation*, Springer-Verlag, Vienna, 1989.

[iii] K. Hess, *Monte Carlo Device Simulation: Full Band and Beyond*, Kluwer Academic Publishing, Boston, 1991.

[iv] M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods*, Wiley, New York, 1986.

[v] D. K. Ferry, *Semiconductors*, Macmillan, New York, 1991.

[vi] H. D. Rees, *J. Phys. Chem. Solids*, 30 (1969) 643.

[vii] R. M. Yorston, *J. Comp. Phys.*, 64 (1986) 177.

[viii] L. I. Schiff, *Quantum Mechanics*, McGraw-Hill Inc., New York, 1955.

[ix] Y.-C. Chang, D. Z.-Y. Ting, J. Y. Tang and K. Hess, *Appl. Phys. Lett.*, 42 (1983) 76.

[x] L. Reggiani, P. Lugli and A. P. Jauho, *Phys. Rev. B*, 36 (1987) 6602.

[xi] D. K. Ferry, A. M. Kriman, H. Hida and S. Yamaguchi, *Phys. Rev. Lett.*, 67 (1991) 633.

[xii] P. Bordone, D. Vasileska and D. K. Ferry, *Phys. Rev. B*, 53 (1996) 3846.