aestimo 1D
User Guide and Tutorial

Zbigniew Koziol *†,
State University - Education-Science-Production Complex, Russia

September 12, 2013 – Version 0.4
Preliminary

Contents

*softquake@gmail.com
†With numerous contributions of Aestimo development team.

1 Aestimo? What is that?

Aestimo is a self-consistent Schrödinger-Poisson solver written for simulating basic physical phenomena of 1-dimensional (1-D) semiconductor heterostructures. Aestimo was started as a hobby by Professor Sefer Bora Lisesivdin from Gazi University, Ankara, Turkey, at the beginning of 2012, and become a usable tool which can be used as a co-tool in an educational and scientific work.

1.1 Who can use it?

Aestimo is aimed to be both educational and research tool. For educational purposes, it must be easy to understand. For research, it must have some point of computational correctness and sensitivity.

Written in Python scripting language, broadly used by scientific community, combines easy access to external quality libraries and source code with speedy and accurate simulation, extensibility, and community technical support. The very basic experience with computer programming is needed (if at all) and very basic physics understanding to start manipulating existing examples or even to design new models.

This document aims at explaining the use of Aestimo at a students level, and should be also useful for advanced researches.

1.2 Download and installation

Aestimo can be downloaded for free from Aestimo site, where additional documentation may be found.

You will need to have a recent version of Python installed on your computer (on Linux, almost certainly you have it already). For this, please refer to Python website, where binary packages for most platforms can be found. Additionally, you need libraries called numpy and pylab. Aestimo itself does not need compilation. Packages may need. If you are on Linux Ubuntu than installation of packages is straightforward from Ubuntu Software Center or by using apt-get from terminal console.

1.3 Copyright

Aestimo is copyrighted by Sefer Bora Lisesivdin under GNU GPL v.3. This document is copyrighted by Zbigniew Koziol under Creative Commons License (CC BY-NC).

2 What does it solve?

The following computational schemes are implemented (these are determined by the parameter *computation_scheme* in model definition file):

Table 1: The computational schemes implemented

computation_scheme	Model
0	Schrödinger
1	Schrödinger + nonparabolicity
2	Schrödinger-Poisson
3	Schrödinger-Poisson with nonparabolicity
4	Schrödinger-Exchange interaction
5	Schrödinger-Poisson + Exchange interaction
6	Schrödinger-Poisson + Exchange interaction with nonparabolicity

2.1 Schrödinger equation

Solving Schrödinger equation is the primary step in finding physical phenomena in heterostructures and quantum wells. In the first approximation, the potential in eq. 1 origins from mismatch of energy bands at heterointerfaces. One ought to pay attention only to proper differentiation there, due to mismatch of effective masses of particles at interface boundaries. Hence, the kinetic energy operator should not be used in the form $1/m(y) \cdot \frac{\partial^2}{\partial y^2}$ but instead in this way: $\frac{\partial}{\partial y}\left(\frac{1}{m(y)}\frac{\partial}{\partial y}\right)$.

Table 2: Aestimo files

File name	Description
aestimo.py	Main Aestimo file containing all procedures for some computational schemes (Table 1).
aestimo_numpy.py	A more advanced version of aestimo.py, recommended for use. It contains some procedures not available in aestimo.py and it uses numpy python module for more efficient computation. Likely will be renamed to aestimo.py in the future.
aestimo_numpy_h.py	A 3x3 k.p aestimo calculator for valence band calculations.
config.py	Configuration file available to user. There you put also the name of input file name with structure description you want to simulate.
database.py	Material parameters database. Keep an original copy of it, but entries there and parameters can be changed.
main.py	main file. Run it as: python main.py.
main_iterating.py	An example showing how to perform computation for a changing parameter.
VBHM.py	It is used as an include file by aestimo_numpy_h.py. It contains procedures for solving 3x3 Hamiltonian for a (001)-oriented zinc blende (ZB) crystal by using k.P theory + Finite Difference Method.
sample*.py	The name can be anything. These are sample input files with structure description. Put a file name name into config.py

$$\left(-\frac{\hbar^2}{2}\frac{\partial}{\partial y}\left(\frac{1}{m(y)}\frac{\partial}{\partial y}+V(y)-E+\frac{\hbar^2k^2}{2m(y)}\right)\right)\psi(y)=0, \tag{1}$$

Many example solutions of simple heterostructures and quantum wells are discussed in details by Harrison ([1]).

Aestimo solves 1-D Schrödinger equation by using shooting method. In numerical analysis, the shooting method is a method for solving a boundary value problem by reducing it to the solution of an initial value problem. For Schrödinger equation it is described in details by Harrison, [1].

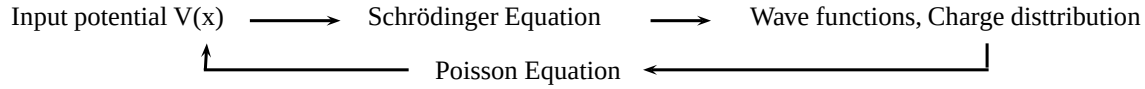
When Schrödinger equation alone is solved (also with nonparabolicity; scheme 0 and 1 in Table 1) than a one loop calculations are performed, only. When Poisson solving is included into computational scheme than self-consistent calculations in a loop are performed.

2.2 *Coupled Schrödinger-Poisson equations*

Solving Schrödinger equation (adding there nonparabolicity as well) goes in one iteration step. We have an input potential, V(x) and just find numericaly eigenvalues and corresponding wave functions.

The problem becomes more complex when we would like to take into account other effects.

Have a look on this schematic drawing:



Once we solve Schrödinger equation and get wavefunctions, we have at the same time charge distribution, since absolute value of wavefunctions is related to charge distribution. However, the resulting charge distribution will modify potential V(x) for which we solved the original Schrödinger equation. A new value of potential is found by solving Poisson equation. Therefore, we ought to solve Schrödinger equation again, in a new potential, and find again a new charge distribution. We should repeat these steps over and over again, until the results of the before the last solution are reasonably close to the results of outcome from the last solution. This method of solving equations is called a self-consistent method. Harrison draws the scheme of calculations as shown in Figure 1.

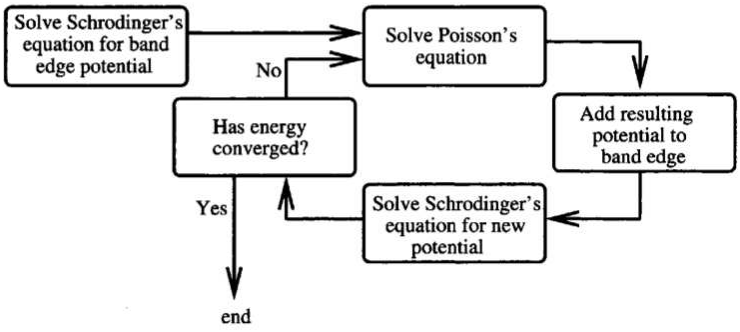


Figure 1: Block diagram illustrating the process of self-consistent iteration (Figure 3.35 in Harrison).

The self-consistent method is used whenever we solve Poisson equation or add any other corrections to potential. In these cases the calculations run in a loop, until either the allowed loop number is exceeded or calculations become stable, i.e. a convergence is reached.

There are a few main parameters that control loop calculations and convergence. At the moment of this writing, in case of aestimo version 0.8, these parameters are located in aestimo.py and aestimo_numpy.py files (Table 3).

Table 3: Parameters used to control convergence of self-consistent solving of coupled Schrodinger-Poisson equations by using shooting method

Parameter	Default	Description
max_iterations	80	convergence is reached when the ground state energy (meV) is stable to within this number between iterations
convergence_test	1e-6	Schrödinger + nonparabolicity
delta_E	1.0 meV	Energy step for initial search
E_start	0.0	Energy to start shooting method from
damping	0.5	Averaging factor between iterations to smooth convergence

2.3 *Nonparabolicity of E(k)*

Nonparabolicity parameter is introduced in order to take into account dependence of effective mass on particle energy. Without that correction, effective masses from the bottom of the conduction band is considered. The approximation in bulk semiconductors can be described by the dispersion relation (Harrison, [1], p. 105, Nelson [3]):

$$E = \frac{\hbar^2 k^2}{2m^*}\left(1 - \gamma k^2\right), \tag{2}$$

where E, k, and m^* are the energy, wave number, and effective mass of the charge carrier and γ is the nonparabolicity parameter.

In $database.py$, the nonparabolicity parameter for electrons is represented by m_e_alpha.

It is important to account for this effect in particular in case of narrow QWs or in higher lying subbands for wide QWs.

2.4 *Exchange interaction*

Electrons (holes) are Fermions. Two of them can not occupy the same state when their spins are the same. At the same time electrostatic interaction exists between them, regardless how far apart they are. A very introductory text on this matter is, for instance, in this PDF lecture. We may say that their quantum-mechanical state depends on an additional potential, which is called exchange-correlation potential, V_{xc} in Aestimo. There is no a theory that would allow to compute exactly how strong are these interactions. There are though approximations used that are in practice sufficiently accurate, even though V_{xc} correction to potential energy may be significant. That additional potential ought also be included into a self-consistent loop of iterations.

Exchange interaction plays more significant role in cases of strong doping (at low temperatures and very low doping levels it may be ignored). An effective field describing the exchange-correlation between the electrons is derived from Kohn-Sham density functional theory. This formula is given in many papers, for example see Gunnarsson et al. (1976) [12], Ando et al. (2003) [13], or Ch.1 in the book "Intersubband transitions in quantum wells" [14]. Exchange interaction is the main contribution to the effective narrowing of band gap (Band Gap Narrowing, BNG) when a large number of carriers is present there.

In Aestimo, that correction to potential is computed by procedure

```
calc_Vxc(sigma,eps,cb_meff)
```

which is a function of dielectric constant, effective mass and charge density. It is based on the equation:

$$V_{xc} = -A \cdot \frac{n^{1/3}}{\epsilon} \cdot (1 + 0.0545 \cdot r_s \cdot log(1 + 11.4/r_s)) \,, \tag{3}$$

where $A = q^2/(4\pi) \cdot (3/\pi)^{1/3}$, n is electron density per square meter, and r_s is average distance between charges in units of effective Bohr radius.

2.5 *k.p method for valance band splitting.*

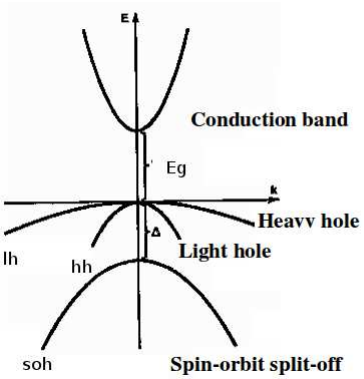


Figure 2: Kane model: Only a conduction band, a heavy and light-holes band and a spin-orbit split-off band with double degeneracy are considered.

In semiconductors, valence bands are well characterized by 3 Luttinger parameters. At the Γ-point in the band structure, $p_{3/2}$ and $p_{1/2}$ orbitals form valence bands. But we have there also a spin-orbit coupling: due to nonzero angular momentum state electrons (i.e., other than s-type wave functions) generate a magnetic field through which they interact with the spin of the electrons. This is particulary important for the valence band (p-like states). Because of this coupling neither spin nor orbital angular momentum but the total angular momentum becomes a good quantum number. The spin-orbit coupling splits sixfold degeneracy into high energy 4-fold and lower energy 2-fold bands. Again 4-fold degeneracy is lifted into heavy- and light hole bands by phenomenological Hamiltonian by J. M. Luttinger (see Wikipedia article on luttinger parameters).

Calculations for holes are performed by using *aestimo_numpy_h.py* file. That file includes *VBHM.py* which contains procedures for solving $3x3$ Hamiltonian for a (001)-oriented zinc blende (ZB) crystal by using k.p theory + Finite Difference Method. The acronym $3x3$ means in this case that we are dealing with 3 valance bands. This is a crude approximation, as it is known. All of three bands are double spin-degenerate. A better approximation would be to use $6x6$ method (but we have no explicite any magnetic interactions, so far, in our Aestimo). Even better one would be to use a $4x4$ method, when conduction band is included. And even better one would be to use $8x8$ method, especially in case of applied external magnetic field, when 3 valance bands interact and one conduction band, and all are spin-degenerate.

The method we use is explained by Harrison [1], page 110, and also Chuang [17], p. 183. For a same code in Fortran, and Dirichlet boundary conditions, see Ahn and Park [2]

3 How the material structure and grid are created?

3.1 How the device structure and grid are created?

The device structure and grid on which computation is performed are created in the simplest possible way. As an example, have a look to file $sample - qw - qwdope.py$. At the end, we find there the following definition:

```
material =[[ 20.0, 'AlGaAs', 0.3, 0, 'n'],
           [10.0, 'GaAs', 0, 2e18, 'n'],
           [20.0, 'AlGaAs', 0.3, 0, 'n']]
```

The meaning of it is as follows. Our device consists of three regions (layers). The first region has the width 20 nm, second 10 nm and the third one 20 nm. The first and third regions are of AlGaAs while the second is GaAs. In case of AlGaAs, we assume Al concentration $x = 0.3$. Only the second region is dopped, with concentration of $2 \cdot 10^{18} cm^{-3}$ and doping type is n.

The grid (a descrite set of points on which computation is performed) is controled by parameter $gridfactor$ in this file, which sets the maximum value (in nm) of distance between grid points. Additionaly, we have there a parameter $maxgridpoints$ that sets a limit on the number of allowed grid points. That last one can be safely set to a large number, a something like $10e5$, though for simple structures we use only the number of grid points of the order of $10^2 - 10^3$, when $gridfactor$ is set to to $1nm$ or $0.1nm$, which gives good accuracy of computation.

3.2 How the material database is implemented?

As of version 0.8 of aestimo, the following binary compounds and alloys have entries in material database: AlGaAs, InGaAs, InGaP, AlInP, GaSb, AlSb, InSb, AlGaAs, InGaAs, InGaP, AlInP. Have a look to $database.py$ file to find out how the database have been implemented in python. We use a very quick and dirty solution to code it. Its advantage is however that is is simple and could be understood, modified and new compounds and parameters can be added there by anyone. All parameters values listed there are for 300K. The main sources of the data are works of Adachi [15], Hamaguchi [16], Chuang [17], Zhang and Razeghi [18], etc.

Table 4: Database parameter names used in Aestimo, for binary and tertiary compounds

Parameter	Unit	Description
m_e	m_0	electron effective mass
m_hh	m_0	heavy hole effective mass
m_lh	m_0	light hole effective mass
epsilonStatic	ϵ_0	static electrical permittivity
Eg	eV	Energy gap
Bowing_param	-	Determines E_g change with alloy composition in ternary compounds
Band_offset	-	$\Delta E_c/\Delta E_g$, see Eq. 4
m_e_alpha	$1/eV$	non-parabolicity parameter for electrons (see Eq. 2).
G1, G2, G3	$\hbar^2/2m_0$	Luttinger Parameters γ_1, γ_2, , γ_3
C11,12	$10^{11} dyne/cm^2$	Elastic constants
a0		Lattice constant a_0
Ac	eV	deformation potential for conduction band (see Chuang, Table 10.1)
Av	eV	deformation potential for valence band (see Chuang, Table 10.1)
B	eV	shear deformation potenial (see Chuang, Table 10.1)
delta	eV	spin-orbit-split of valence bands
Material1	-	Binary compound No 1 in an tertiary alloy
Material2	-	Binary compound No 2 in an tertiary alloy

3.3 Bands alignement and initial potential

The very first step before any computation can be performed is creating the initial potential of the structure, the conduction band and valence band. There are two main approaches used for that by tools similar to Aestimo. Some prefer to use so called band offset parameters to align energy gaps at heterostructure junction (nextnano for instance) while others use the concept of affinity energy (Synopsys' Sentaurus TCAD).

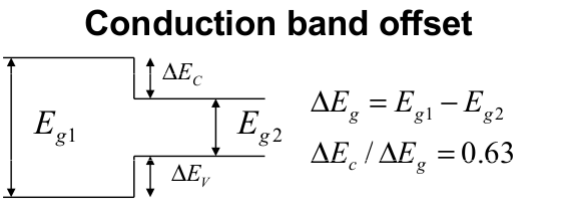


Figure 3: Illustration of band offset parameter at a heterostructure junction.

Both have pros and cons. Band offset parameters are somewhat more of empirical origin, but can not be used in case of arbitrary heterojunctions: band offset parameters refer to specific pairs of compounds, only. Otherwise with the concept of affinity energy, which is energy of an electron state at the bottom of conduction band needed to remove it from bulk material into vacuum. From the point of physics, the concept is nicely defined. However, it appears that in practical experiments using electron affinities does not always work very well for predicting measured heterostructure offsets (see Wikipedia article about the Anderson model of heterostructures).

For these reasons we allow users of Aestimo to decide themselves which method to use for aligning bandgaps. This is done with parameter $use\_bandgap\_offset$ in $config.py$. If it is set $True$ than the band offset method is used, with $Band\_offset$ parameter values from database. If $use\_bandgap\_offset$ is set $False$, initial heterostructure potential is computed by using $affinity\_energy$ parameter from database.

In case of conduction band, the following line of code is used in Aestimo (i enumerates grid points):

```
fi[i] = matprops['Band_offset']*matprops['Eg']*q #Joule
```

where

```
fi[i]
```

is potential energy,

```
matprops['Band_offset']
```

is value of $Band\_offset$ from database, and

```
matprops['Eg']
```

is energy gap of the material.

In other words, in Asetimo, when using band offset concept, we assume that conduction band energy is given by:

$$E_c = \Delta E_c / \Delta E_g \cdot E_g, \tag{4}$$

where $\alpha = \Delta E_c / \Delta E_g$ is the value of $Band\_offset$ parameter from database.

After we have E_c, valence band energy is obtained in a straightforward way, $E_v = E_c - E_g$.

Users may manipulate values of material parameters. Here is an example of how we were able to reproduce structure consisting of 3 compunds, with the following layers, 10 nm wide each of them:

GaSb, AlSb, InAs, AlSb, GaSb.

We did not had entries of $Band\_offset$ for all heterointerfaces in such a sandwich. Our aim was to reproduce diagram from Thesis of Sebastian Brosig from 2000 "Transport Measurements on InAs/AlSb Quantum Wells" (Fig. 3.2 in that work).

In order to get conduction band alignement, we used the following $Band\_offset$ parameters:

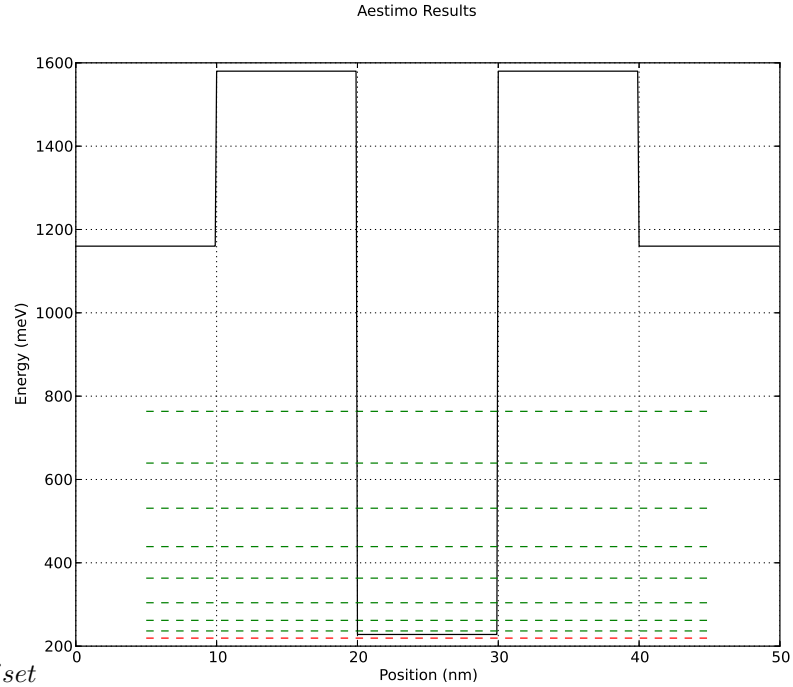


Figure 4: Conduction band alignement (black solid line), energy levels (broken green lines) and Fermi energy (red broken line) in $GaSb-AlSb-InAs$ heterostructure: materials are $GaSb$, $AlSb$, $InAs$, etc, starting from the left region.

```
GaSb: 1.6
AlSb: 1.0
InAs: 0.57
```

Actually, any values are good there, as long as the structure obtained corresponds well to that "real" one from measurements. Let us notice also that by selecting $Band\_offset = 1$ for a compund (in this case for $AlSb$) we implicitely select zero energy value at the top of valence band for that compund (see Figure 4).

If affinity energy (χ_e) scheme is used for band allignement than conduction band is given by:

$$E_c = -\chi_e, \tag{5}$$

while, obviously, $E_v = E_c - E_g$.

It is easy to check by using data in Table 5 that results displayed in Figure 4 are in a very good agreement with these that would be obtained by using affinity energy scheme of computation.

Table 5: Electron Affinity energy χ_e and energy gap E_g (in eV) for selected binary compounds, at T=300K

Compound	χ_e [eV]	Ref.	E_g	Ref.
AlAs	3.5	[8]	2.980	
AlP	3.98	[9]	2.48	
AlSb	3.64	[18]	1.58	[18]
GaAs	4.5	[18]	1.42	
GaN	3.4	[10]	3.39	[8]
GaP	3.8	[9]	2.261	
GaSb	4.03	[18]	0.725	[18]
InAs	4.45	[18]	0.4	
InP	4.5	[9]	1.35	
InSb	4.69	[18]	0.17	[18]

There is however an important advantage by using the last method. In case of heterostructures with alloys (ternary and tertiary compunds) it may become difficult to guess what $Band\_offset$ values should be used. While at the same time we usually know how E_g and χ_e change with alloy composition (in most cases, a linear interpolation between parameters of AB and BC compounds for $A_xB_{1-x}C$ offers a good approximation; exceptions are for instance $AlGaAs$ and $InAlAs$ where two linear interpolations must be used, depending on x range studied).

On another hand, once we know dependences of $\chi_e(x)$ and $E_g(x)$ for heterostructure alloys, we may find out $Band\_offset$ parameter α. Let us assume also that compound on right is the same as on left, but differes only in concentration of one component x, with $E_c^r = a + b \cdot x$ and $\chi_e^r = c + d \cdot x$. Hence, we search a value of $\Delta E_c^r / \Delta E_g^r$ on the right side of heterointerface:

$\Delta E_c^r / \Delta E_g^r = -\Delta \chi_e^r / \Delta E_g^r$,

or:

$\Delta E_c^r / \Delta E_g^r = -d \cdot \Delta x / b \cdot \Delta x = -d/b$. Simple and nice, right?

As an example, let us take $Al_{x_1}Ga_{1-x_1}As$ / $Al_{x_2}Ga_{1-x_2}As$ interface, where x_1 and x_2 are less than 0.45. For this compund, $d = (3.575 - 4.11826)/0.45eV$ and $b = (1.98515 - 1.42248)/0.45eV$ (the parameters were taken from Synopsus' Sentaurus database). Hence, we obtain $\Delta E_c^r / \Delta E_g^r = 0.54$, which is close to empirical value.

3.4 Alloys

3.4.1 Bowing parameter $Bowing\_param$

In metallurgy, Vegard's law is an approximate empirical rule which holds that a linear relation exists, at constant temperature, between the crystal lattice parameter of an alloy, a, and the concentrations of the constituent elements.

For example, consider the semiconductor compound InP_xAs_{1-x}. A relation exists then, such that:

$$a_{InPAs} = x \cdot a_{InP} + (1 - x) \cdot a_{InAs} \tag{6}$$

One can also extend this relation to determine semiconductor band gap energies. Using InP_xAs_{1-x} as before one can find an expression that relates the band gap energies, E_g, to the ratio of the constituents. In that case, however, and also in case of other physical quantities, a linear relation is usually better replaced by a quatratic one. A bowing parameter b, denoted in database by $Bowing\_param$ is added:

$$E_{g,InPAs} = x \cdot E_{g,InP} + (1 - x) \cdot E_{g,InAs} - b \cdot x \cdot (1 - x) \tag{7}$$

4 Overview of Examples

Table 6: Examples

No	File name	Run script	*computation_scheme*
1	sample-qw-qwdope	aestimo	2
2	sample-qw-barrierdope	aestimo-numpy	6
3	sample-moddop	aestimo	2
4	sample-qw-HarrisonCh3_3	aestimo	2
5	sample-qw-barrierdope-p	aestimo	2
6	sample-double-qw	aestimo	2
7	any	main_iterating.py	any

4.1 Quantum well doped AlGaAs/GaAs heterostructure.

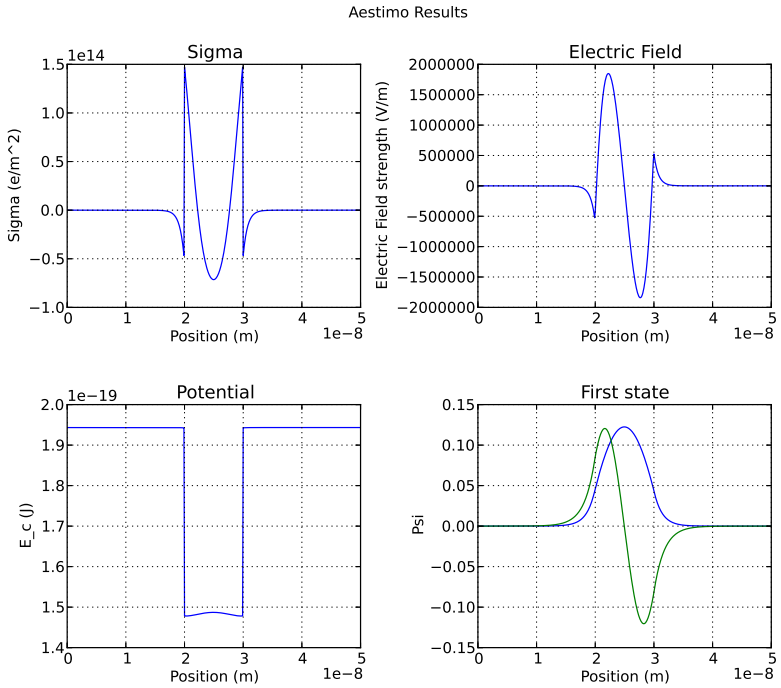


Figure 5: Output results obtained by running *sample-qw-qwdope.py* input file, for a single quantum well, doped, of AlGaAs. Figures show charge distribution, electric field, final potential, and two first wave functions.

4.2 Barrier doped AlGaAs/GaAs heterostructure.

VBHM.py It is used as an include file by aestimo_numpy_h.py. It contains procedures for solving 3x3 Hamiltonian for a (001)-oriented zinc blende (ZB) crystal by using k.P theory + Finite Difference Method.

Description: We consider the upper 3x3 Hamiltonian for a (001)-oriented zinc blende (ZB) crystal see ref [1], after block diagonalization the KP-FDM (k.P theory + Finite Difference Method) method is explained in ref [2] to ensure the Hermitian property of Hamiltonian you have to apply the we have to write all operators of the form presented in ref [3], to understand Hermiticity property see ref [2]. page 110,same code in fortran language presented in the index of ref [1]. Dirichlet boundary conditions were applied [1]. [1]:D.Ahn & S-H.Park 'ENGINEERING QUANTUM MECHANICS' P 238 [2]:P.Harrison 'QUANTUM WELLS, WIRES AND DOTS' P 357-362 [3]: S-L.CHUANG 'physics of Optoelectronic Devices' P 183

4.3 Modulation doped AlGaAs/GaAs heterostructure.

4.4 Shooting method.

4.5 Barrier doped AlGaAs/GaAs heterostructure.

4.6 Double Quantum well doped AlGaAs/GaAs heterostructure.

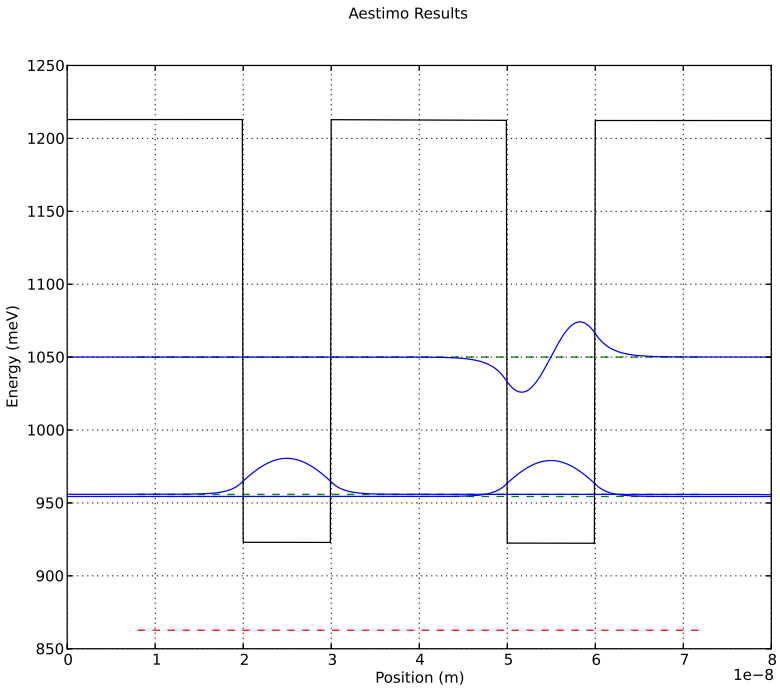


Figure 6: The first 2 localized wave functions (solid lines) and their corresponding energies (broken lines) for a simple 2-QWs structure obtained with *sample-double-qw.py* sample input file.

4.7 Calculations for many changing parameters: main_iterating.py.

In this example, calculations are performed for a set of values of thickness of the first region, within one run. Hence, we do not need to create a separate input file for every value of the parameter that we want to have results for. Have a look to the code of main_iterating.py for some explanations. Of course, if calculations for some another parameter need to be looped through than just create another version of main_iterating.py for your needs. Mostly, not many changes will be needed.

5 Working with Datafiles

5.1 Structure of Datafiles

Computation results are stored in subfolder *outputs* or *outputs-numpy* by default. These names can be changed by user in model configuration files, with parameter *output_directory*.

For instance, the following files may be found there:

```
efield.dat
potn.dat
sigma.dat
states.dat
wavefunctions.dat
```

Each of these files contains rows and columns of data. Columns are separated by empty space (0x20). This is a convenient ASCII format of storing data, since it allows for easy viewing the results by naked eye and also it is accessible by many software around (for instance, *gnuplot*), as well it can be easy parsed by *python* scripts.

Table 7 describes details of output files' structure.

The file *states.dat* contains information about energy states in multi-structure. Most likely it will be energy of localized Quantum-Well states. The fourh column in that file is carriers effective mass (in units of free electron mass). Where the zero of energy is localized depends on how the initial potential was constructed (refer to section *Bands alignement and initial potential*).

The file *wavefunctions.dat* contains results of computation of wavefunctions, solutions of Schrödinger equation (or self-consistent Schrödinger-Poisson equations, depending on computation scheme). The first column there is coordinate in sample, starting from it's left edge. It is given in meters. Next columns contain results for ψ's, while their number depends on the number of states computed, which is set in input file by parameter *subnumber_e* (in case of electrons).

Table 7: Structure of output datafiles

File name [.dat]	1st column	2nd column	3rd column	next columns
efield	x [m]	Electrical field [V/m]	-	-
potn	x [m]	Electrical potencial [J]	-	-
sigma	x [m]	Charge density $[e/m^2]$	-	-
states	state No	Energy [meV]	Density of states $[1/m^2]$	eff. mass $[m_e]$
wavefunctions	x [m]	1st Ψ amplitude	2nd Ψ amplitude	etc.

5.2.1 GNUPLOT

```
set term x11;

#set term postscript eps enhanced color "Times" 22
#set output "figures/gnuplot-example00.eps"

set xlabel "x [m]" font "Helvetica,18";
set ylabel "{/Symbol Y}" font "Helvetica,18"; #{/Symbol Y} produces Greek letter Psi in postscript

set xtics 1e-8 # to draw xtics every 1e-8; if not specified, the default is used which is too dense

plot \
"../outputs/wavefunctions.dat" u 1:2 with lines linewidth 3 title "{/Symbol Y}_1",\
"../outputs/wavefunctions.dat" u 1:3 with lines linewidth 3 title "{/Symbol Y}_2"
exit;
```

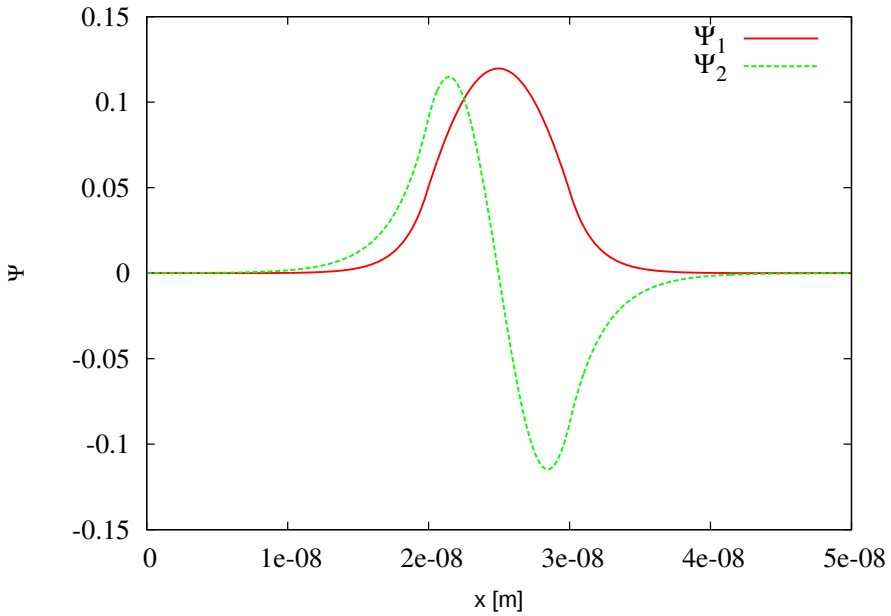


Figure 7: Example figure created in Gnuplot by using code as in this section. Results show wavefucions for the first and second bound states in a simple one QW heterostructure.

5.2.2 PYTHON

Some people have aversion to commercial software, others to software where intensive use of mouse and clicking is needed. Often these two sets of people overlap and I belong to that common part of them.

pylab provides a procedural interface to the matplotlib object-oriented plotting library. It is modeled closely after Matlab(TM). Therefore, the majority of plotting commands in pylab have Matlab(TM) analogs with similar arguments. Important commands are explained with interactive examples.

iPython tutorial for beginners: http://www.loria.fr/ rougier/teaching/matplotlib/

In interactive mode:

ipython –pylab

This opens a shall where commands can be typed in.

Or you may run examples by typing in terminal window:

python example_file_name.py

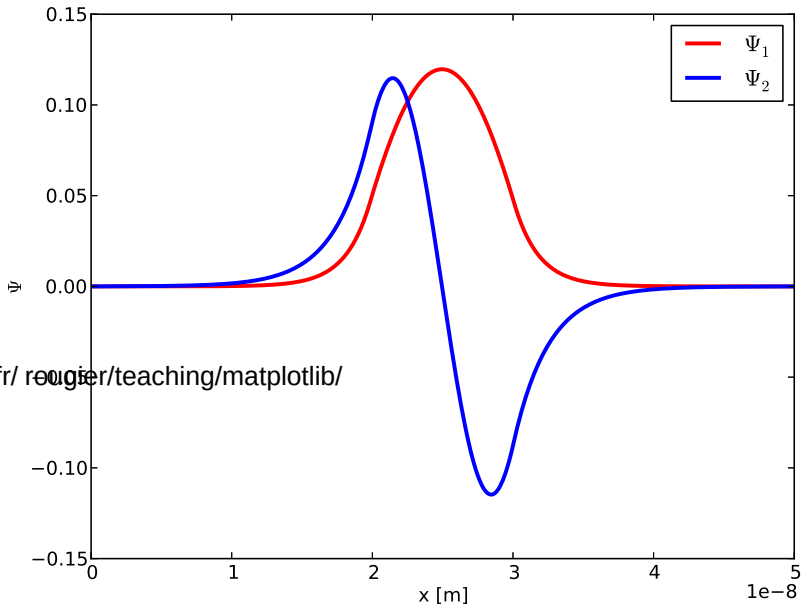


Figure 8: The same data as these in Figure 7, but this figure was created by using python example code from this section.

```
from pylab import *
import numpy
x, wave1, wave2 = numpy.loadtxt("../outputs/wavefunctions.dat", skiprows=0, unpack=True)

plot(x,wave1, color="red", linewidth=2.5, linestyle="-", label="$\Psi_1$")
plot(x,wave2, color="blue", linewidth=2.5, linestyle="-", label="$\Psi_2$")

legend()

ylabel('$\Psi$')
xlabel('x [m]')

savefig("figures/python-example00.eps",dpi=72)
show()
```

6 Appendix. Additional example how to control computation with Perl

Actually, I come through these problems when attempting to use perl for control of computational process(*). Yes, I know you are not enthusiasts of using perl for that. But, anyway, if python would be used then you probably will encounter similar problems with calculation flow as I did.

(*) perl was supposed to change, for instance, Al concentration in regions left and right of QW, in steps, controlled by an index. I wanted to have a separate log file for every index, and separate output file names. Right now that requires changes (during the run) of input file, config.py and even aestimo.py (or aestimo_numpy.py). If changes were done as in points 1) and 2) than only config.py needs to be changed on the run.

In file sample-config-perl-control-template.py we would like to have different file names for input. We put there:

```
inputfilename = "qw-perl-__MY_INDEX__"
```

and __MY_INDEX__ will determine the name of input filename.

We would like also to have a different log file name, also determined by loop index, hence we put there as well:

```
logfile = '__MY_INDEX__.log'
```

In file sample-qw-perl-control-template.py we need to replace __WAVEGUIDE_AL__ with Al concentration. This is the part of the file that interests us:

```
material =[[ 20.0, 'AlGaAs', 0.__WAVEGUIDE_AL__, 0, 'n'],
            [10.0, 'AlGaAs', 0.08, 1e16, 'n'],
            [20.0, 'AlGaAs', 0.__WAVEGUIDE_AL__, 0, 'n']]
```

Finaly, we would like to have calculation results saved to filenames that are also indexed by loop index. For instance, if we want to save data about potential and energy of states, than we need to have something like this in file aestimo-perl-template.py:

```
potn__MY_INDEX__.dat
states__MY_INDEX__.dat
```

instead of

```
potn.dat
states.dat
```

Here is a perl example:

```
#!/usr/bin/perl

my $CONFIG, $QWPERL, $AESTIMO;
```

```
open(CONFIG, "<", "sample-config-perl-control-template.py");
while(my $l=<CONFIG>) {
$CONFIG .= $l;
}
close CONFIG;

open(QWPERL, "<", "sample-qw-perl-control-template.py");
while(my $l=<QWPERL>) {
$QWPERL .= $l;
}
close QWPERL;

open(AESTIMO, "<", "aestimo-perl-template.py");
while(my $l=<AESTIMO>) {
$AESTIMO .= $l;
}
close AESTIMO;

# $waveguide is Al concentration in regions on left and right of QW.
# In this case, we change it from 0.081 to 0.400, in steps of 0.002
for (my $waveguide=79; $waveguide<=400; $waveguide++) {

$waveguide++;

my $idx_replacement = $waveguide;
if ($waveguide <100) {
$idx_replacement = "0" . $waveguide;
} else {
$idx_replacement = "" . $waveguide;
}

my $myOut = $CONFIG;
$myOut =~ s/__MY_INDEX__/$idx_replacement/g;

open (CONFIG, ">", "config.py");
print CONFIG $myOut;
close CONFIG;

my $myOut = $QWPERL;
$myOut =~ s/__MY_INDEX__/$idx_replacement/g;
$myOut =~ s/__WAVEGUIDE_AL__/$idx_replacement/g;

open (QWPERL, ">", "qw-perl-$idx_replacement.py");
print QWPERL $myOut;
close QWPERL;

my $myOut = $AESTIMO;
$myOut =~ s/__MY_INDEX__/$idx_replacement/g;
```

```
$myOut =~ s/__WAVEGUIDE_AL__/$idx_replacement/g;

open (AESTIMO, ">", "aestimo.py");
print AESTIMO $myOut;
close AESTIMO;

system('/usr/bin/python aestimo.py');
}
```

Now, once we have these results, we would like to draw dependence of bound QW energies as a function of Al content in regions surrounding QW.

We will use this Python script for that (see Figure 9 for drawing results):

```
from pylab import *
import numpy

CB0 = []
my_x = []
my_state_0 = []
my_state_1 = []
my_state_2 = []

for idx in range(80, 400, 2):

   if idx<100:
      myidx = "0" + `idx`
   else:
      myidx = `idx`

   my_potential_input_file_name = "output/potn" + myidx + ".dat"
   my_states_input_file_name = "output/states" + myidx + ".dat"

   x, potn = numpy.loadtxt(my_potential_input_file_name, skiprows=0, unpack=True)
   no, states, y1, y2 = numpy.loadtxt(my_states_input_file_name, skiprows=1, unpack=True)

   CB0.append(potn[0])
   my_x.append(idx*0.001)
   my_state_0.append(states[0])
   my_state_1.append(states[1])
   my_state_2.append(states[2])

xlim([0.08,0.4])

plot(my_x,CB0, color="red", linewidth=2.0, linestyle="-", label="$E_{CB0}$")
plot(my_x,my_state_0, color="green", linewidth=1.0, linestyle="-", label="$E_{C1}$")
plot(my_x,my_state_1, color="blue", linewidth=1.0, linestyle="-", label="$E_{C2}$")
plot(my_x,my_state_2, color="#F000F0", linewidth=1.0, linestyle="-", label="$E_{C3}$")

legend(loc="upper left")
```

```
ylabel('$E\ [meV]$')
xlabel('$x\ (Al\ content)$')

savefig("../tutorial/figures/python-example10.eps",dpi=72)

show()
```

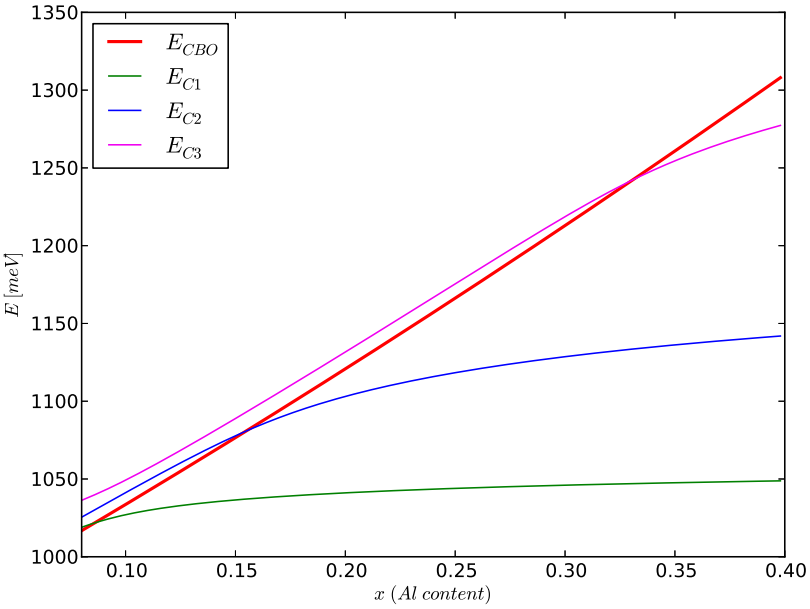


Figure 9: Dependence of bound QW energies for single QW heterostructure of AlGaAs/AlGaAs/AlGaAs, as a function of Al content in regions surrounding QW. Al in QW content is 0.08. Conduction Band Offset energy is drawn by thick red line. Parts of curves that lye below E_{CBO} represent electron states localized in QW. The figure is created with python code shown in this subsection.

References

[1] Paul Harrison, *Quantum Wells, Wires and Dots. Theoretical and Computational Physics of Nanostructures*. Wiley-Interscience, 2005.

[2] D.Ahn, S-H.Park 'ENGINEERING QUANTUM MECHANICS'

[3] Nelson, D. F., R. C. Miller, and D. A. Kleinman. "Band nonparabolicity effects in semiconductor quantum wells." Physical Review B 35, no. 14 (1987) 7770

[4] aestimo.ndct.org

[5] matplotlib.org

[6] matplotlib.org/contents.html

[7] Wei Zhang and M. Razeghi, in: Semiconductor Nanostructures for Optoelectronic Applications, Todd Steiner, Editor, Artech House, Inc.

[8] www.ioffe.rssi.ru

[9] Schubert

[10] SERGEY L. RUMYANTSEV AND MICHAEL. S. SHUR, MICHAEL E. LEVINSHTEIN, MATERIALS PROPERTIES OF NITRIDES. SUMMARY

[11] Wikipedia on Luttinger parameters

[12] Gunnarsson and Lundquist (1976) O. Gunnarsson, M. Jonson and B.I. Lundqvist, Exchange and Correlation in Atoms, Molecules and Solids. Phys. Lett. 59A, 177 (1976).

[13] Taro Ando, Hideaki Taniyama, Naoki Ohtani, Masaaki Nakayama, and Makoto Hosoda, Self-consistent calculation of subband occupation and electron–hole plasma effects: Variational approach to quantum well states with Hartree and exchange-correlation interactions, J. Appl. Phys. 94, 4489 (2003)

[14] M. Helm, in "Intersubband transitions in quantum wells", edited by Liu and Capasso.

[15] Sadao Adachi, Properties of Semiconductor Alloys: Group-IV, III-V and II-VI Semiconductors, 2009 John Wiley & Sons, Ltd.

[16] Chihiro Hamaguchi, Basic Semiconductor Physics, Second Edition, 2010, Springer

[17] S.-L. Chuang, Physics of Optoelectronic Devices , 1995, John Wiley & Sons. Inc

[18] Wei Zhang and M. Razeghi, in: Semiconductor Nanostructures for Optoelectronic Applications, Todd Steiner, Editor, Artech House, Inc.