Appendix A: Introduction to the computing

A large part of this book has been concerned with strengthening the connection between the theory of semiconductor heterostructures and their solution by computational methods. This latter aspect will be reinforced by the accompanying CD-ROM. The contents of the disk are a mountable file system which contain *all* of the worked examples illustrated and discussed in the book. In addition, the C source codes themselves are present, thus allowing interested readers, should they wish, to repeat the calculations in the book as a learning aid for their own work.

The computer codes themselves have been developed over a number of years; note however, that these are working research codes, which means that they require some degree of expertise in order to run them and to get the most out of them. The codes were developed on UNIX-like systems and the odd one does contain a system call, but as they conform to the ANSI standard, then they should execute under other operating systems, with a little effort.

There is a philosophy behind the structure of the computer codes. The calculations are, wherever possible, broken down into the smallest possible units, which allows for a great deal of flexibility. For example, the exciton-binding-energy calculation is written for any pair of electron and hole wave functions. Therefore, after these are generated by some solution of a single quantum well, the binding energy in a single quantum well can then be

calculated. If the binding energy of an exciton in a double quantum well is required, then the user only need calculate the one-particle solutions of the double quantum well, and supply these as input to the exciton binding energy calculation. When additional calculations for diffusion and electric field are added *before* the exciton calculation, it becomes apparent that the possible permutations are almost endless.

For several years now, execution of the codes has resembled typical UNIX-like command, with parameters being passed to the programs via command-line arguments. For example, the envelope-function solution to a single quantum well is invoked by using:

```
efsqw -a 50
```

The motivation behind this approach was to allow for automation of large amounts of simple calculations and to avoid the need to recompile the programs after the change of a system or material parameter. While each calculation on its own can be trivial and of little use, combining many calculations together to explore, say, the effect of the well width of the ground state energy, is of more interest. This could be achieved with a short shell script, i.e.

```
#!\bin\sh

for L in 20 40 60 80 100 120 140 160 180 200
do

 efsqw -a $L

 echo -n $L >> E-L.r
 nawk 'printf("%le\n",$2)' Ee.r >> E-L.r

done
```

The first line just specifies that this script should run under the standard shell. The loop defines the values which the well width `L` should take. Within the loop itself, which is specified between '{}', there are three commands. The first calculates the ground-state energy of a single quantum well of width `L`. The second writes this width to the output file which is called `E-L.r`, which stands for 'Energy versus L results'. The third and final line is a small awk script which filters the second argument (the energy itself) of the file output from '`efsqw`' and writes it after the well width entry in `E-L.r`. For more details on the specifics of the single-quantum-well calculation '`efsqw`', see later. The CD-ROM contains all of the data generated for the figures in the book, laid out as computationally worked examples. The latter make extensive use of such shell scripting (always contained within the file called '`history`'), and in the vast majority of cases it is only necessary to execute the appropriate script, simply by typing its name, for that particular set of calculations to be repeated and all of the data files regenerated.

It should be noted no knowledge of the C-programming language is needed, and the computer codes can be treated as 'black boxes' that need to be fed input data (most of which is generated automatically) and will then give output data. Some basic familarity with UNIX-like operating systems is required in order to *install* the software, while some knowledge of awk will allow the user to automate complex series of calculations—however, this is not essential.

Whatever the calculations the reader is interested in, it is very worthwhile reading the following two appendices, as these codes will often form the basis of the input to the more sophisticated calculations described later on. In addition, the philosophy behind the computational set-up is explained by way of selected examples.

It is difficult to guarantee that such an extensive range of source codes will be totally 'bug free', and the author therefore welcomes any 'bug reports' and 'fixes'. Software updates and further information are available on a web page, currently at:

`http://www.imp.leeds.ac.uk/qwwad/`

where the '`qwwad`' is derived from 'Quantum Wells, Wires And Dots'.

Appendix B: Computer codes used in Chapter 2

The computer codes described in this section are applicable to quantum well systems which have semi-analytical solutions. As these are within the realm of the effective-mass/envelope-function approximations, they are largely identified by the '`ef`' prefix.

Infinite quantum well (`efiw`)

This code generates the energy eigenvalues and wave functions of an infinitely deep quantum well. It is implemented simply by typing its executable filename

```
efiw
```

Variables are passed to the program by way of command-line arguments, which (as is the case with all of the codes) can be listed by specifying the '`-h`' for 'help' option, i.e.

```
efiw -h
```

gives

```
Usage:  efiw [-L well width (100A)][-m mass (0.067m0)]
```

```
[-N total number of points 100]
[-p particle (e, h, or l)]
[-s # states 1]
```

This help information is aimed at being as logical as possible, so the first option '`-L`' is a way of specifying the width of the infinitely deep quantum well. Within the square brackets the default value is represented in **bold** font together with the units. So, for example, in order to implement the calculation for a well of width 50 Å, the user would type:

```
efiw -L 50
```

The '`-m`' option specifies the effective mass of the particle within the well while, the '`-N`' option allows the user to change the number of points output in the wave-function file, the name of which, together with the name of the energy eigenvalues file, is controlled by the '`-p`' option. If an electron is the particle of interest, then '`-p e`' sends the energy levels to the file `Ee.r` and the wave functions to the files `wf_en.r`, where 'n' is the principal quantum number. This is the only function of the `-p` option, and does not in any way affect the output data themselves. The final option is used to control the number of energy levels calculated.

Therefore, an example calculation of the lowest three energy levels of a hole (`-p h`) of mass $0.6m_0$ in a 50 Å quantum well may look like the following (note the options can be supplied in any order):

```
efiw -L 50 -p h -m 0.6 -s 3
```

This would then give the energy eigenvalues in the file `Eh.r` in the form:

```
1  2.50688719003485510e+01
2  1.00275487601394200e+02
3  2.25619847103136950e+02
```

while the wave functions are in files `wf_h1.r`, `wf_h2.r`, and `wf_h3.r`. As with all of the envelope wave-function files, they are in a two-column format, the first of which is the position z along the growth axis of the quantum well system, while the second is the wave function, $\psi(z)$, itself. The first few lines of `wf_h1.r` look like:

```
0.00000000000000000e+00 0.00000000000000000e+00
5.05050505050505040e-11 6.34558669961352960e+02
1.01010101010101010e-10 1.26847839313129020e+03
1.51515151515151530e-10 1.90112086608365350e+03
2.02020202020202020e-10 2.53184907147498510e+03
2.52525252525252530e-10 3.16002791946699790e+03
3.03030303030303050e-10 3.78502488720820430e+03
3.53535353535353570e-10 4.40621065573081250e+03
4.04040404040404030e-10 5.02295974362158450e+03
```

```
4.54545454545454550e-10 5.63465113682859460e+03
```

The wave functions are normalised and, as is fairly universal for all of the data files, SI units are used.

Density of states (dos)

This code reads in a set of energy eigenvalues from the file `Ep.r`, where 'p' is the particle identifier 'e, h, or l' and calculates the density of states for bulk, and assuming that the eigenvalues are for quantum wells and wires, two-dimensional and one-dimensional systems, respectively. The options can be listed with '`dos -h`' which gives:

```
Usage:  dos [-m mass (0.067m0)][-p particle (e, h, or l)]
```

Again, the '`-p`' option merely specifies the name of the file from which to read the energy eigenvalues. The output from the program is written as four columns (energy, followed by the 3D, 2D and 1D density of states) in the file `rho.r`.

Subband populations (sbp)

This code module calculates the quasi-Fermi energies and Fermi–Dirac distribution functions, for given subband carrier densities, and at a given temperature. The input requirements are an energy-eigenvalues file, `Ep.r`, and the carrier densities in each level; these are specified in the file '`N.r`', an example of which is given below:

```
1 6
2 7
```

These data imply a carrier density of 6×10^{10} cm^{-2} in the ground state and 7×10^{10} cm^{-2} in the first excited state. Execution of '`sbp`' gives the quasi-Fermi energies in the file `Ef.r` and if required the Fermi–Dirac distribution functions as a function of energy, in the files `FDn.r`, where 'n' is the principal quantum number of the energy level. The options for '`sbp`' are:

```
Usage:  sbp [-f (output distributions false)][-m mass (0.067m0)]
            [-n (number of output energies 1000)][-p particle (e, h, or l)]
            [-T temperature (300K)]
```

'`sbp`' is also described in the context of scattering rate calculations in Appendix H.

Single quantum well (efsqw)

As the name suggests, this code implements the semi-analytical solution to the single quantum well. The system parameters such as well width, barrier height

and effective mass of the particle, are passed as command-line arguments, a full listing of which can be obtained with '`efsqw -h`' which gives:

```
Usage:  efsqw [-a well width (100A)][-b barrier width (200A)]
              [-k use alternative KE operator (1/m)PP P(1/m)P]
              [-m well mass (0.067m0)][-n barrier mass (0.067m0)
              [-p particle (e, h, or l)][-s # states 1]
              [-V barrier height (100meV)]
```

Again, the '`-p`' option just specifies the names of the output files, namely `Ee.r` for electrons, and `Eh.r` for holes, etc. The wave functions are again written in the same format as before to files with names of the form `wf_pn.r`, where 'p' is the particle identifier and 'n' is the principal quantum number.

This code will generate the solutions to the ground and excited states with the option `-s` as before. In addition, different masses in the well and barrier regions are supported. Even the two forms of the kinetic energy operator can be used, although the commonly accepted one, i.e. '`P(1/m)P`' is the default and should (nearly) always be taken.

The remaining option, '`-b`' does not affect the energy levels, and is only used to determine the distance into the barrier that the wave function should be plotted.

Kronig-Penney superlattice (`efkpsl`)

The operation of this code is very similar to '`efsqw`'. It implements the Kronig–Penney solution to an infinite superlattice for any particular particle wave vector k. This variable is passed as the command-line option '`-k`' in units of π/L, where L $(= a + b)$ is the superlattice period. Thus, the centre of the Brillouin zone is specified with '`-k 0`' and the edge with '`-k 1`'. The full list of options is given below

```
Usage:  efkpsl [-a well width (100A)][-b barrier width (100A)]
               [-k wave vector (0 pi/L)
               [-m well mass (0.067m0)][-n barrier mass (0.067m0)
               [-p particle (e, h, or l)][-s # states 1]
               [-V barrier height (100meV)]
```

Transmission coefficient for a single barrier (`tsb`)

This code calculates the transmission coefficient for a single barrier, the options are:

```
Usage:  tsb [-d energy step (0.1meV)][-L barrier width (100A)]
            [-m effective mass (0.067m0)][-V barrier height (100meV)]
```

which are all self-explanatory, except perhaps '`-d`'. This options specifies the energy increment between the output data points. The output from this

program is written to the file `T.r` in two columns, i.e. the energy E and the value of the transmission coefficient $T(E)$.

Transmission coefficient for a double barrier (`tdb`)

The transmission coefficient of a double barrier can be calculated with '`tdb`'; the mode of operation is very similar to '`tsb`', except that now two barrier widths and a well width need to be specified. The options are listed below:

```
Usage:  tdb [-a left hand barrier width (100A)][-b well width (100A)]
            [-c right hand barrier width (100A)][-d energy step (0.01meV)]
            [-m well effective mass (0.067m0)][-n barrier mass (0.067m0)]
            [-V barrier height (100meV)]
```

Current-voltage (I–V) characteristic for a double barrier (`ivdb`)

This code computes a very simple model for the I–V curve of a double barrier structure. The options follow in a similar way to the above example:

```
Usage:  ivdb [-a left hand barrier width (100A)][-b well width (100A)]
             [-c right hand barrier width (100A)]
             [-m effective mass (0.067m0)]
             [-T temperature (300K)]
```

Appendix C: Computer codes used in Chapter 3

The envelope-function/effective mass approximation approach to the solution of any general heterostructure, which can be represented by $A_{1-x}B_x$ or perhaps $A_{1-x-y}B_xC_yD$, is implemented by a set of program 'modules'. Note that in the first case, the most general description of a ternary heterostructure can be represented by the function $x(z)$, while a quaternary requires two functions, i.e. $x(z)$ and $y(z)$. Often the function of each module is simple, but when combined they form a versatile tool which allows the solution of any one-dimensional confining potential, be it in a binary, ternary, or quaternary compound.

The shooting technique approach to the solution of the Schrödinger equation, discretizes the second-order differential equation along a one-dimensional grid (or mesh). The material parameters such as potential $V(z)$ and effective mass $m^*(z)$ are required at each point on the grid. Working from the definition of the heterostructure, i.e. $x(z)$ for ternaries or $x(z)$ and $y(z)$ for quaternaries, the other parameters are created by individual modules and can then be stored in files.

The advantage of this modular form is that new types of structure can be addressed often by the addition of just one module. For example, diffused

quantum wells require merely a module to generate the $x(z)$ profile, and all of the other programs can be utilized as with simple quantum well systems.

Definition of heterostructure (`efsx`)

The name of this code, as with all other codes, does represent its function. All codes begin with '`ef`', implying 'envelope function', and in this case '`sx`' stands for 'structure to x', where x is the alloy concentration as a function of the displacement along the growth (z-) axis.

This code generates the heterostructure definitions, namely $x(z)$ and $y(z)$, for simple layered systems. It reads the user-defined input file `s.r` and generates the output file `x.r`. The input file contains the definition of the heterostructure and consists of three or four columns of data, with the first representing the thickness of a layer in Ångstroms, the second the alloy concentration x, the third the alloy concentration y, and the (optional) fourth the doping level in this layer (with p-type being positive and n-type negative) all taken for the same layer. For example, a 80 Å GaAs single quantum well within $Ga_{0.8}Al_{0.2}As$ barriers would be described in the `s.r` file as:

```
200 0.2 0.0
80  0.0 0.0
200 0.2 0.0
```

Note that in this case the material system is a ternary, and hence the second compulsory alloy concentration y is always zero. Similarly, a symmetric double quantum well may be represented with:

```
200 0.2 0.0
60  0.0 0.0
20  0.2 0.0
60  0.0 0.0
200 0.2 0.0
```

An example of a quaternary-based double quantum well is the $In_{1-x-y}Al_xGa_yAs$ system, where work has included calculations on symmetric double quantum wells of $In_{0.53}Ga_{0.47}As$ separated by AlAs barriers. A typical `s.r` was:

```
100 1.0 0.0
48  0.0 0.47
10  1.0 0.0
48  0.0 0.47
100 1.0 0.0
```

This code (like all modules) has options that can be invoked from the command line; a list of options can be obtained with '`-h`', for example, the command:

```
efsx -h
```

returns

```
Usage:  efsx [-N points/Angstrom 1]

Format of input file 's.r' for material A(1-x-y)B(x)C(y)D or
A(1-x)B(x)C(1-y)D(y)---check material definitions with command

efxm -M help

=========================begin s.r===================================
layer thickness (A)        x       y       doping level (10^18cm^-3)
```

This single option '`-N`' summarizes the function of the program, with the layer thicknesses and alloy concentrations being turned into a one-dimensional 'mesh'—the alloy concentrations are listed as a function of position z in the file `x.r`. The origin is always set as the first point in the list, i.e. the left-hand edge of the structure. For this module, requesting help gives a much longer listing than normal—which is a short reminder on how to format the `s.r` input file.

When the '`efsx`' command is run on the single-quantum-well description above, with the default '`N`' value of 1 point/Angstrom, the first 10 lines of the `x.r` have the form:

```
0.00000000000000000e+00 2.000000e-01 0.000000e+00
1.00000000000000000e-10 2.000000e-01 0.000000e+00
2.00000000000000010e-10 2.000000e-01 0.000000e+00
3.00000000000000000e-10 2.000000e-01 0.000000e+00
4.00000000000000010e-10 2.000000e-01 0.000000e+00
5.00000000000000030e-10 2.000000e-01 0.000000e+00
6.00000000000000000e-10 2.000000e-01 0.000000e+00
7.00000000000000060e-10 2.000000e-01 0.000000e+00
8.00000000000000030e-10 2.000000e-01 0.000000e+00
8.99999999999999990e-10 2.000000e-01 0.000000e+00
```

The section near the left hand well–barrier interface looks like:

```
1.94999999999999990e-08 2.000000e-01 0.000000e+00
1.96000000000000000e-08 2.000000e-01 0.000000e+00
1.97000000000000010e-08 2.000000e-01 0.000000e+00
1.98000000000000020e-08 2.000000e-01 0.000000e+00
1.99000000000000000e-08 2.000000e-01 0.000000e+00
2.00000000000000000e-08 0.000000e+00 0.000000e+00
2.01000000000000010e-08 0.000000e+00 0.000000e+00
2.01999999999999990e-08 0.000000e+00 0.000000e+00
```

```
2.03000000000000000e-08 0.000000e+00 0.000000e+00
2.04000000000000010e-08 0.000000e+00 0.000000e+00
```

The effect of the optional fourth column of data in the structure definition file `s.r` will be discussed later.

Generation of potential profile (`efxv`)

This program takes the alloy concentration data in `x.r` and converts it, at each z point, into the corresponding potential energy, writing the output to `v.r`. This module contains experimental data for the band offset and the bandgap as a function of the alloy concentration x (or concentrations x and y for quaternaries). Again, the options can be listed with '`efxv -h`', which gives:

```
Usage:  efxv [-M material Ga(1-x)Al(x)As][-p particle (e, h, or l)]
             [-g output bandgap false]
```

The '`-p`' option enables the user to choose the heterostructure potential for either the electron '`e`' (default), heavy hole '`h`' or light hole '`l`', providing of course that the data is defined within the program. The '`-M`' option allows the user to select the material system, currently there are three defined; which can be listed with '`efxv -M nonsense`', where '`nonsense`' is really any string. This yields the following:

```
The only materials defined in the database are
Ga(1-x)Al(x)As, Cd(1-x)Mn(x)Te, In(1-x-y)Al(x)Ga(y)As
```

The first of these is the default material and is selected with:

```
efxv -m gaalas
```

while the rest follow intuitively. Note that these definitions are important for defining the alloy concentrations x and y.

The final option, i.e. '`-g`', also generates the bandgap of the heterostructure, again as a function of z, and written in two columns to '`Eg.r`'; this is necessary to implement *non-parabolicity* in the solution to the Schrödinger equation.

Generation of effective mass profile (`efxm`)

This program is exactly analogous to '`efxv`' as described above, with the difference being it generates the effective mass m^* as a function of z and writes it to the file `m.r`. The options are the same as `efxv`, except that if the option '`-m`' is given followed by a number, then the material dependency of the effective mass is removed and is set to the constant value specified.

Numerical solution of Schrödinger's equation (`efshoot`)

This is perhaps the most important module. It applies the numerical shooting technique to solve the one-dimensional potential Schödinger equation. The energy levels are written to the file `Ee.r`, `Eh.r` or `El.r`, depending on the choice for the option '`-p`', as mentioned above for '`efxv`'. The options are listed as usual:

```
Usage:  efshoot [-a include non-parabolicity false]
                [-d energy step (1meV)][-e initial energy (0meV)]
                [-k use alternative KE operator (1/m)PP P(1/m)P]
                [-p particle (e, h, or l)]
                [-s number of states 1]
```

The first option has already been mentioned. The options '`-d`' and '`-e`' affect the numerics of the code. The program finds a solution to the equation $\psi_\infty(E) = 0$ by stepping up the energy E axis in regular steps (default is 1 meV) until the function changes sign. At this point, a Newton–Raphson iteration is implemented in order to accurately obtain the energy E of the state satisfying $\psi(z) \to 0$, as $z \to \infty$. Therefore, the energy step must be less than the separation between two adjacent levels. Generally the default of 1 meV is fine; however, for systems with near degeneracy, such as symmetric double quantum wells, this may have to be reduced.

As the above energy step can increase the computational time enormously, a quick route to the solutions can be taken by setting the *initial energy* with the '`-e`' option. For example, if the near-degenerate doublet is around 50 meV, then setting:

```
efshoot -d 0.01 -e 47
```

will enable the program to find the solution much more quickly than having to step up the energy axis all the way from the potential energy minimum (the default).

There has been controversy over the form of the kinetic energy operator, with the two opposing forms being:

$$-\frac{\hbar^2}{2m^*(z)}\frac{\partial^2}{\partial z^2} \qquad \text{or} \qquad -\frac{\hbar^2}{2}\frac{\partial}{\partial z}\frac{1}{m^*(z)}\frac{\partial}{\partial z}$$

The commonly accepted form is the latter and this is chosen as the default.

Again, the particle that is being solved for can be specified; the default is always the electron '`-p e`', and this only affects the energy-output filename. Finally, the number of confined-state solutions to be found can be set with '`-s`'.

Wave function calculation (`efwf`)

This program module calculates the wave function for each of the n energies stored in the file `Ee.r` (or `Eh.r`, etc.) and writes them to individual files

`wf_p`n`.r`, where the letter '`p`' actually represents the value of the `-p` option. The options of this code are a subset of '`efshoot`'.

Additional of electric field (`effv`)

After solving the zero-field case as above, it is often interesting to look at the effect of an electric field. The physics is contained fully by the addition of an extra term to the potential $V(z)$ contained in `v.r`. Remembering that the origin of the heterostructures is always taken as the left-hand edge of the structure, then the potential term added is in fact:

$$\text{electric field potential energy} = -eF(z - z_0)$$

where z_0 is the z-coordinate of the centre of the structure, and F is the electric field strength.

The effective-mass definitions remain the same, so after running through the sequence above for the zero-field case, execution of '`effv`' with the appropriate field option generates a new `v.r` file, with the field added. This can then be solved as above with '`efshoot`' and '`efwf`' as normal.

The options of '`effv`' can again be viewed with '`effv -h`':

```
Usage: effv [-f kV/cm] [-h]
```

Note that as '`effv`' uses the original zero field `v.r` file and replaces it with one containing an electric field potential, any subsequent execution of '`effv`' would *add* another field potential. This is guarded against automatically and the first run of `effv` looks for a file `v0.r`, which is acknowledged to contain the zero-field potential; if the file is not present, it copies `v.r` to `v0.r` and then proceeds. If the file *is* there, '`effv`' reads from `v0.r` instead of `v.r`.

Parabolic quantum wells (`efpqw`)

This code produces a parabolic quantum-well potential for solution with efshoot in the standard manner; '`efpqw -h`' gives:

```
Usage:  efpqw [-a width at top of well (100A)][-b barrier width (100A)]
              [-N number of points per Angstrom 1]
              [-o output data to screen false]
              [-x minimum alloy concentration x 0.0]
              [-y maximum alloy concentration x 0.1]
```

The actual form of the parabola is adjusted by way of the three parameters, '`-x`', '`-y`', and '`-a`', which in turn specify the minimum alloy concentration at the bottom of the potential, the maximum in the barrier and the width of the well at the top. By default, this generates a potential data file `v.r` with the alloy profile stored in the second column, i.e. it is assumed that the heterostructure is constructed from a ternary which can be written as $A_{1-x}B_xC$, such as $Ga_{1-x}Al_xAs$.

Pöschl–Teller potential hole (pth)

This code produces the potential profile of a Pöschl-Teller potential hole, suitable for solution with '`efshoot`', and gives the analytical solutions to the energy levels.

Heisenberg's uncertainty principle (hup)

The '`hup`' code evaluates the position and momentum expectation values necessary for a comparison with Heisenberg's uncertainty principle. The only input that it requires is an envelope function in the standard format of the `wf_p`n`.r` files. The particular wave function is assigned with command-line arguments, as specified with '`hup -h`':

```
Usage:  hup [-p particle (e, h, or l)][-s state 1]
```

As an example, when generating a wave function for the ground state of an infinitely deep quantum well, and taking all of the defaults with:

```
efiw
```

the subsequent execution of '`hup`' then gives:

```
<z>                                     5.00000000000000180e-09
<z^2>                                   2.82672738086797260e-17
Delta_z=sqrt(<z^2>-<z>^2)               1.80756018120551330e-09

<p>/hbar                                -1.49302650243043900e-08 i
<p^2>/sqr(hbar)                         9.86877620480528000e+16
Delta_p/hbar=sqrt(<p^2>-<p>^2)/hbar     3.14146083929201300e+08

Delta_z*Delta_p                         5.67837952392069470e-01 hbar
```

where these output lines, in order, represent the expectation value of position, position squared, and the resulting uncertainty in position (all in SI units). These are followed by the expectation value of momentum, momentum squared, and the resulting uncertainty in the momentum, this time in units of $\hbar$. The last line forms the product $\Delta z \Delta p$, it can be seen in this case that the latter is slightly greater than $\hbar/2$.

Self-consistent solution to Poisson's equation (scp)

Following the successful solutions of the energy levels and wave functions of a quantum-well system, it is possible to solve Poisson's equation; however some additional information does need to be supplied, e.g. the optional fourth column in the structure definition file `s.r` is required. This contains the doping density in each semiconductor layer in units of 10^{18} cm^{-3}. As an example, a doped single quantum well might look like:

```
200 0.2 0.0  0.0
100 0.0 0.0 -1
200 0.2 0.0  0.0
```

A negative value indicates n-type doping, while a positive value would represent p-type doping. When '`efsx`' is executed the presence of the fourth column leads to the generation of a new file `d.r`, which contains the dopant profile.

It is also necessary to specify the proportion of charge carriers in each of the quantum-well energy levels. This is done manually in the file `Nu.r`, where the '`N`' indicates the number of carriers in each level and the '`u`' just reminds the user that these are unnormalised populations. Therefore, percentages of the total could be input, e.g. for a population of electrons split 60:40 over the two lowest energy levels, `Nu.r` could be written:

```
1 60
2 40
```

or

```
1 6
2 4
```

One of the actions of the code '`scp`' is to normalise these populations according to the total dopant density specified in `d.r`; these are then written to the file `N.r` in units of 10^{10}cm^{-2}. This is a very important file and is used extensively later in the calculation of scattering rates.

The code name '`scp`' is something of a misnomer, as its actual function is to take this charge distribution and solve Poisson's equation, and then return the resulting potential in the file `V.r`, it also adds this potential to the original potential and puts this total potential in `v.r`. The previous data in this file are protected by storage in `v1.r`. Thus a second execution of '`efshoot`' solves the original band-edge potential plus the potential due to the charge distribution. Of course, this results in new energy levels and new wave functions, which in turn will affect the potential derived from Poisson's equation, and hence this loop of successive calculations with '`efshoot`' and '`scp`' must be continued until the energy levels of the system have converged, i.e. self-consistency is achieved between input and output.

The code '`scp`' produces some additional output files which are useful for diagnostics/understanding the physics of the problems, e.g. `F.r` is the electric field due to the charge distribution (both doping and carriers) and `sigma.r` is the net sheet charge density as a function of the position.

Appendix D: Computer codes used in Chapter 4

It has already been mentioned that any semiconductor quantum-well system can be represented most generally by the the spatial dependence of an alloy concentration, e.g. $x(z)$ for $A_{1-x}B_x$ or $A_{1-x}B_xC$. Chapter 4 was concerned with the modelling of the diffusion of the alloy components in quantum-well systems, all of which were performed with a single numerical solution of the diffusion equation.

General (numerical) solution to the diffusion equation (`gde`)

As the name suggests, this code implements the numerical solution to the diffusion equation. It requires two pieces of information, with the first of these being the initial diffusant profile $x(z)$. This is read in from the file `x.r` which has been created with `efsx` from the `s.r` file in the usual way. The second requirement is for the complete specification of the diffusion coefficient, which is written most generally as $\mathcal{D}(x, z, t)$. The standard options can be listed by using '`gde -h`', which gives:

```
Usage: gde [-a form of diffusion coefficient (0, 1, 2 or 3)
           [-D the constant coefficient, if a=0 selected (1A^2/s)
           [-d time interval dt (0.01s)
```

```
          [-t final time (1s)
```

The '`-t`' option is self-explanatory, i.e. it is the total time for diffusing, while '`-d`' is a computational option and gives the time steps by which the diffusant profile should be projected into the future. The default is a sensible one and should work for typical cases, but note, however, that a particularly strong dependence of the diffusion coefficient on another parameter might require it to be made smaller.

The form of the diffusion coefficient is specified by means of the argument to the '`-a`' option. The above listing indicates that choosing `-a 0` sets a constant diffusion coefficient, the value of which must be specified with the '`-D`' option. More details on the arguments for `-a` can be obtained by simply giving an invalid argument, e.g. '`gde -a ?`' gives:

```
Usage: gde [-a form of diffusion coefficient (0, 1, 2 or 3)
       0=>D=D0 a constant, specified by the -D option
       1=>D=D(z) only, read in from file D.r
       2=>D=D(D,x,z,t), general dependency, completely defined in dox.c
       3=>D=D(z,t), initial z-dependence from D.r, t-dependence defined in dox.c
```

The option '`0`' has been described, while '`1`' allows for a position-dependent diffusion coefficient to be read in from an external file called `D.r`. This is a simple two-column file containing the position z and diffusion coefficient $\mathcal{D}$, both of which are in SI units as usual.

Such z-dependencies, if they can be expressed as an analytical function, can be defined in the file `dox.c`, as can the time (t) and concentration (x) dependencies. This is selected with the option '`-a 2`'. Note that the file `dox.c` must be available at compilation time, regardless of whether it is to be used or not. In the latter case, the functional form contained within `dox.c` becomes irrelevant. It should also be noted that if the function is changed, then '`gde`' must be recompiled. Finally, the option '`-a 3`' allows for a positional- and time-dependent diffusion coefficient, as might occur in the annealing out of radiation damage.

The results of the numerical simulation of diffusion are written to the file `X.r`.

Appendix E: Computer codes used in Chapter 5

The codes used in this chapter refer specifically to 'donors'—as this was the motivation behind their creation. As discussed, they can also be applied to acceptors, but *with caution*. The three donor binding energy codes, i.e. '`d02D`', '`d03D`', and '`d0`', implement variational calculations, the first two with a single variational parameter (the Bohr radius λ) and the third with an additional symmetry parameter (ζ). The mapping of the parameter space is fairly crude, and merely increments the variational parameter until a turning point is found. At times this can be slow, and a welcome improvement would be to implement a more sophisticated method...

Two-dimensional donor wave function (`d02D`)

This code calculates the binding energy of a donor (or acceptor) by using a two-dimensional trial wave function. The options are:

```
Usage:  d02D [-d energy step (1meV)][-e relative permittivity 13.18]
             [-m mass (0.067m0)]
             [-s starting lambda (50A)][-t lambda increment (1A)]
             [-u final lambda (-1A)]
```

The options '-e' and '-m' specify, respectively, the required physical parameters of the permittivity of the material and the effective mass of the charge carrier, while all of the other parameters relate to computational aspects of the calculation. The first of these, i.e. '-d', has the same function as in efshoot, while '-s', '-t', and '-u' control the behaviour of the variational aspects of the calculation. The calculation increments λ in units specified by -t, from the starting value specified with -s. If the final value of λ is less than zero, then the calculation will proceed until a turning point, and hence the solution, is found. However, if the final value for λ is greater than the start value, then the calculation will continue to this upper limit. This is useful for mapping-out the energy–Bohr radius parameter space.

Three-dimensional donor wave function (d03D)

The remaining two versions of the donor (impurity) binding-energy calculation are similar in operation to the previous example, and thus it is sufficient here to describe the differences. The code, 'd03D' implements the calculation with the spherical trial wave function. The command-line options are:

```
Usage:  d03D [-d energy step (1meV)][-e relative permittivity 13.18]
             [-m mass (0.067m0)]
             [-s starting lambda (50A)][-t lambda increment (1A)]
             [-u final lambda (1A)]
```

which are identical to d02D.

Variable-symmetry donor wave function (d0)

This code implements the variable-symmetry, trial-wave-function donor-binding-energy calculation. Its operation is very similar to the above two functions, but note, however, that the second variational parameter leads to additional options, i.e.

```
Usage:  d0 [-d energy step (1meV)][-e relative permittivity 13.18]
           [-m mass (0.067m0)]
           [-s starting lambda (50A)][-t lambda increment (1A)]
           [-u final lambda (-1A)]
           [-w starting zeta 0.001][-x zeta increment 0.1][-y final zeta -1]
```

where '-w', '-x', and '-y' control the behaviour of the ζ (zeta) parameter in the same way as '-s', '-t', and '-u' do for λ.

Spin-flip Raman (sfr)

This program simply generates a 'fake' absorption line profile by using the donor-binding-energy calculations given above for a spatial distribution of

donors. Within a heterostructure, this spatial distribution also implies an energy distribution, and hence at times it is worthwhile to be able to combine these into some sort of spectrum. It is assumed that the 'spin-flip energies' i.e. the energy difference between the two donor spin states in a magnetic field, are stored in the file `e_sf.r`. This particular data file does break all of the rules, however, and contains these spin-flip energies in meV (rather than the usual SI units), as a function of position.

```
Usage:  sfr [-l linewidth (1cm^-1)]
            [-s lower limit of Raman spectra (0cm^-1)]
            [-t increment (1cm^-1)][-u upper limit (100cm^-1)]
```

The program assigns a Gaussian lineshape to each energy, where the 'full-width at half-maximum' values of which are specified with the '`-l`' option; summation is then carried out over all the donor positions for each energy point. The lower, upper and incremental values for the latter are controlled with the '`-s`', '`-u`' and '`-t`' options, respectively, and the spectrum is written to the file `I.r`.

Appendix F: Computer codes used in Chapter 6

Exciton-binding-energy calculation (ebe)

This calculation is performed by a code called '`ebe`', and requires the wave functions of the one-particle electron and hole states of interest. The '`ebe`' code represents a two-parameter (λ and β) variational calculation, with the control of this aspect being rather like that of the impurity calculations described in the previous section. This is illustrated by the following options:

```
Usage:  ebe [-a # electron eigenstate 1][-b # hole eigenstate 1]
            [-e relative permittivity 13.18]
            [-m electron mass (0.067m0)][-n hole mass (0.62m0)]
            [-N # points in w integration 100][-o output data to screen false]
            [-s starting lambda (70A)][-t lambda increment (1A)]
            [-u final lambda (-1A)]
            [-w starting beta 0.001][-x beta increment 0.05][-y final beta -1]
```

There are additional options when compared to the code '`d0`'. In particular, '`-a`' and '`-b`' allow the selection of a particular exciton from a whole collection of eigenstates. For example, in a quantum well with two conduction-band states and three valence-band states the following:

```
ebe -a 1 -b 1
```

would calculate the binding energy of the exciton formed between the lowest electron level and the lowest hole level, whereas:

```
ebe -a 2 -b 2
```

would calculate the 'e2hh2' exciton (say).

The option '`-N`' controls the accuracy of the integration with respect to w (see Chapter 6). The default value of 100 should suffice for most instances, while finally, '`-o`' allows the user to monitor the variational calculation as it progresses.

Appendix G: Computer codes used in Chapter 7

Infinitely deep wire (`efiwire`)

This program calculates the energy levels and cross-sectional charge density of a rectangular quantum wire surrounded by infinitely high barriers. The options are:

```
Usage:  efiwire [-m mass (0.067m0)][-N number of points 100]
                [-p particle (e, h, or l)][-s # states 1]
                [-y width (100A)][-z width 100A]
```

where the terms '`-m`' and '`-p`' have the usual function. '`-N`' controls the number of points for the charge density output along each axis, and hence the total number of points is $\mathtt{N}^2$. The option '`-s`' specifies the number of states to be output. As each state requires two quantum numbers to label it, i.e. the principal quantum numbers for both of the y- and z-axis states, then the ground state needs to be labelled as '11', with the excited states following as '12', '21', '22', etc. Hence, the charge densities are written to files of the form `cd11.r`, `cd12.r`, etc.

Circular wire (efcwire)

This code calculates the energy levels of a circular cross-sectional finite barrier (either sharp or graded) quantum wire by using a shooting method. The input data, such as effective mass and potential, are generated by using the corresponding codes for the quantum well systems, but now their interpretation is as functions of the radial coordinate r rather than along the z-axis. The options:

```
Usage:  efcwire [-a include non-parabolicity false]
                [-d energy step (1meV)][-e initial energy (0meV)]
                [-p particle (e, h, or l)]
                [-s number of states 1]
```

are a subset of those of the shooting-method solution of Schrodinger's equation for quantum wells (see the code '`efshoot`'). The energy levels are written to the files, `Ee.r`, `Eh.r`, as specified by the '`-p`' option.

Circular wire wave functions (efcwwf)

As with the quantum-well case ('`efshoot`' and '`efwf`'), the radial component of the wave functions for the circular wire are generated with a separate program, i.e. '`efcwwf`':

```
Usage:  efcwwf [-a include non-parabolicity false]
               [-p particle (e, h, or l)]
```

Spherical dot 'efsdot'

Again a shooting method is used to solve for the energy levels of a spherical quantum dot. The options are obtained in the usual way with '`efsdot -h`':

```
Usage:  efsdot [-a include non-parabolicity false]
               [-d energy step (1meV)][-e initial energy (0meV)]
               [-p particle (e, h, or l)]
               [-s number of states 1]
```

and execution follows as with '`efcwire`' and '`efshoot`'.

Appendix H: Computer codes used in Chapter 8

These computer codes use the data generated by the envelope-function programs (subband energies and wave functions) to calculate scattering rates. At present, three basic scattering mechanisms are available, i.e. electron–longitudinal optic phonon, electron–electron and electron–photon (spontaneous radiative scattering rate). Again, a systematic approach is taken with the executable filenames, all beginning with `'sr'`, for scattering rates.

Independently thermalised subband populations (`sbp`)

The scattering rate calculations require the subband populations to be defined and this is acheived by specifying both the populations and Fermi energies of each subband. The populations are defined manually in the file `N.r` (in units of 10^{10} cm^{-2}). For example, a two-level system with 1×10^{10} cm^{-2} carriers in both levels would be specified by:

```
1 1
2 1
```

Assuming the subband populations to be thermalised, i.e. able to be described by a Fermi–Dirac distribution function with a specified temperature, then,

the Fermi energies can be calculated by using the program module '`sbp`'. The options for '`sbp`' are

```
Usage:  sbp [-f (output distributions false)][-m mass (0.067m0)]
            [-n (number of output energies 1000)][-p particle (e, h, or l)]
            [-T temperature (300K)]
```

If the first flag is set, then the code outputs the subband populations as a function of their energies into files called `FDn.r`, where `n` is the principal quantum number. The option '`-n`' gives the number of energy points in these files, i.e. from the subband minima to the tops of the wells. The Fermi energies themselves are written to the file `Ef.r`, which is equivalent in format to the usual `Ee.r`, and `Eh.r`, etc.

The Fermi energies are a requirement of the electron–LO phonon and carrier–carrier scattering rate calculations, and hence '`sbp`' has to be executed before those modules for *every* change in population and temperature.

Electron–LO phonon scattering rate (`srelo`)

This code calculates the electron–longitudinal phonon (LO) scattering rates, due to *bulk* phonon modes. The code name is explicit in its use of 'electron', in fact '`srelo`' could be used to calculate hole-LO phonon scattering rates, but as warned in the main text, with caution! The code calculates the raw rate for all initial subband energies up to the top of the well, for both emission and absorption. The required rates are specified in the file `rrp.r` (derived from required rates for phonons), i.e. if the system of interest has three subbands and all of the rates that move electrons down a subband were required then `rrp.r` might look like:

```
3 2
3 1
2 1
```

At present many of the material parameters, such as phonon energies, etc. are defined within the code and can only be changed by editing the file manually and recompiling; these values are applicable to GaAs. The options that can be changed on the command-line can again be listed with '`srelo -h`':

```
Usage:  srelo [-A lattice constant (5.65A)][-a generate form factors false]
              [-e low frequency dielectric constant (13.18epsilon_0]
              [-f high frequency dielectric constant (10.89epsilon_0]
              [-m mass (0.067m0)][-p particle (e, h, or l)]
              [-T temperature (300K)]
```

Although the first option seems to be out of place, it does however allow the phonon form factor to be calculated to the edge of the second Brillouin

zone. The second option '`-a`', if set, prints the form factors versus the phonon wave vector to files of the form `Gif.r`, where 'i' and 'f' are, respectively, the principal quantum numbers of the initial and final states in the transition. The remainder of the options are self-explanatory; the dielectric constants are given in units of ϵ_0, while the mass is an in-plane mass in units of the free-electron mass m_0.

Defining the required rates in the file `rrp.r`, as above would give the following output files on execution of '`srelo`':

```
LOa32.r    LOe32.r
LOa31.r    LOe31.r
LOa21.r    LOe21.r
```

where '`LO`' indicates that these files contain electron–LO phonon scattering rates, the '`a`' or '`e`' indicate absorption or emission, and the two numbers give the initial and final states. Note that an electron *can* move from the initial state '3' to the final state '2' by *absorbing* a phonon. The files themselves contain the raw scattering rate as a function of the energy of the initial electron as averaged over all final electron states.

Mean phonon rate (`srmpr`)

The previous module gives 'raw' scattering rates, and doesn't take into account the number of electrons in each subband. In an experiment, these rates are not observed, but rather the weighted mean which accounts for carrier populations in both the initial and final states. For the electron–phonon mechanisms these weighted means are calculated from the output of '`srelo`' using the program '`srmpr`' (derived from mean phonon rate).

In essence, '`srmpr`' takes the scattering rates as a function of the initial subband energy and calculates the mean over Fermi–Dirac distributions in both the initial and final states. The subband populations themselves and the Fermi energies are supplied by the files `N.r` and `Ef.r`. The mean rates of all of the entries in `rrp.r` are written to `LOa-if.r` and `LOe-if.r`. The options:

```
Usage: srmpr [-E phonon energy (36meV)][-m mass (0.067m0)]
             [-M mode (AC, LO)][-T temperature (300K)]
```

The phonon energy is required in order to calculate the subband energy of the final state. The '`-M`' option is redundant at this point in that it will always be left as the default '`LO`', which means average the LO phonon scattering rates.

Thus, after specifying the subband populations in `N.r`, calculating the distribution functions with `sbp` and then the scattering rates with `srelo`, the weighted mean over an initial and final subband population can be obtained by using the following command:

srmpr

As the electron–phonon calculation, 'srelo' does not depend on the subband populations for the production of the raw rates, the effect of different subband populations can be investigated by manually editing N.r, and then re-executing 'sbp', followed by a new averaging with 'srmpr'; note that this is not the case for the carrier–carrier calculation outlined below.

Carrier-carrier scattering rate (srcc)

This module implements the carrier–carrier scattering rate calculation. As for the electron–phonon case, the required rates must be specified in an input file, but this time it is called simply rr.r. This file needs to have a different name as the format is different from rrp.r. Each carrier–carrier rate involves two initial and two final states, and hence each different scattering mechanism requires four numbers to identify it. An example rr.r for intersubband events which move electrons from the second state to the ground state would be:

```
2 2 2 1
2 2 1 1
2 1 1 1
```

while intrasubband events would be specified as:

```
2 2 2 2
1 1 1 1
```

Again, as with, the LO phonon scattering calculation, after deducing the subband populations with sbp the carrier–carrier scattering rate calculation can be simply implemented by using 'srcc'. The options are:

```
Usage:  srcc [-a generate form factors false][-e permittivity (13.18epsilon_0)]
             [-m mass (0.067m0)][-p particle (e, h, or l)]
             [-S screening true]
             [-T temperature (300K)][-w a well width (250A)]
```

The first option, if set, writes the form factors for each channel to files beginning with 'A', followed by the event description, e.g. the first line of the first rr.r above would produce a form factor file A2221.r. This option is rarely used and then only for diagnostic purposes; the last option, '-w', is used only to define the limit of electron wave vectors for these output files—*it does not represent the width of the quantum well!*

The raw scattering rates are written to the files ccijfg.r, where again the letters 'ijfg' specify the initial and final carrier states. As these rates are implicitly dependent upon the carrier densities, it is not possible to subsequently thermally average them over initial and final subband populations as a post-calculation process, and this has to be done within the main calculation. The resulting weighted means are written to the file ccABCD.r.

Screening is included by default, while the use of '`-S`' turns it off.

Radiative rates (`srrad`)

This code simply calculates the spontaneous radiative emission rate from one eigenstate to another. The 2D form requires a cavity length, the default for which is 3 μm, which is a reasonable value.

```
Usage:  srrad [-i initial state 2][-f final state 1][-p particle (e, h, or l)]
              [-m mass (0.067m0)][-3 3D value false]
              [-W cavity length (3microns)]
```

Selecting the option '`-3`' yields a value based on the standard approach for three-dimensional wave functions, and should be considered as being obsolete. This module writes data to standard output which can then be filtered or redirected as required.

Appendix I: Computer codes used in Chapter 9

As with the codes based on the envelope-function calculation which began with the stem '`ef`', the majority of the pseudopotential codes used in Chapter 9 and onwards are identified with '`pp`'. However, before the calculations themselves can begin, some ground work needs to be completed, in particular the construction of the unit cell.

Crystal structure of zinc-blende (`cszb`)

The simplest two-atom basis that occurs in pseudopotential calculations of bulk material has to be written to the file `atoms.xyz` by hand. The format of the file is as follows:

```
2

GE -0.7075 -0.7075 -0.7075
GE  0.7075  0.7075  0.7075
```

where the number of atoms defined in the file is given on the first line. The atom definitions themselves then follow after a further blank line. Each atom

is identified with a string; in this example of a bulk Ge calculation, this string is '`GE`', followed by the x-, y-, and z-coordinates of that atom in Angstroms.

However, more complex unit cells, such as the multiple-atom simple cube that occurs in the large-basis calculations, need to be constructed automatically. This is performed by using the code module '`cszb`', the options of which are as follows:

```
Usage:  cszb [-a anion AS][-c cation GA]
             [-x # cells along x-axis 1][-y # cells 1][-z # cells 1]
             [-A lattice constant (5.65A)]
```

The first two options allow the string identifier for the anion and cation species to be specified independently, while the last option '`-A`' gives the lattice constant A_0. The options '`-x`', '`-y`', and '`-z`' specify the number of eight-atom face-centred-cubic unit cells that stack together in that particular direction in order to create the large cuboid unit cell.

Reciprocal lattice vectors for a face-centred-cubic crystal (`rlv-fcc`)

In addition to the definition of the unit cell, an empirical pseudopotential calculation also requires some knowledge of the periodicity of the crystal. This is obtained from the set of reciprocal lattice vectors $\mathbf{G}$. The code '`rlv-fcc`' generates these reciprocal lattice vectors for a face-centred-cubic crystal as required for a bulk calculation. The options are:

```
Usage:  rlv-fcc [-g maximum value of |G| 4 (2*pi/A0)]
```

The single option controls the maximum magnitude of reciprocal lattice vector to be included, where the default of '`-g 4`' gives the familiar 65-plane-wave explansion set. The vectors themselves are written in units of $(2\pi/A_0)$ to the file `G.r`.

Reciprocal lattice vectors for a simple cubic crystal (`rlv-sc`)

The function of this program is identical to the above, except that it produces the `G`-basis set for the simple cubic crystal, as necessary for a large-basis calculation. The same cube extent, as specified with the options `-x`, `-y`, and `-z` when generating the unit cell with '`cszb`' have to be employed again here. The full list of options therefore becomes self-explanatory, i.e.

```
Usage:  rlv-sc [-g maximum value of |G| 4 (2*pi/A0)]
               [-x # fcc cells along x-axis 1][-y # 1][-z # 1]
```

Large-basis calculation (pplb)

This code module is the empirical pseudopotential calculation itself. It has been written to be as general as possible, and reads its required data from the files atoms.xyz and G.r. Hence, by reading in a two-atom unit cell from atoms.xyz and the reciprocal lattice vectors from the output of 'rlv-fcc', it can then perform a simple bulk calculation. On the other hand, a preceding execution of 'cszb' and 'rlv-sc' will allow a more complex large-basis cell to be solved. The only remaining input that 'pplb' requires are the electron wave vectors **k** at which the user would like the calculations to be carried out. These should be supplied in the file k.r in units of $(2\pi/A_0)$, and hence the Γ- and X-points for bulk could be written in k.r as:

```
0.0 0.0 0.0
0.0 0.0 1.0
```

The form-factor definitions are within the external file ppff.c, the name of which indicates 'pseudopotential form factors', and thus these codes need to be compiled together. The reasoning behind this separation into two units is to allow new form-factor definitions to be added *without* having to edit the main program, and it is hoped this will reduce the likelihood of the introduction of any unintentional alterations. In addition, it allows pplb.c to remain readable while keeping the cluttered multiple-form-factor definitions in a separate database. The options:

```
Usage:  pplb [-A lattice constant (5.64A)]
             [-n # lowest band 1][-m highest band 4], output states
             [-w output eigenvectors (wavefunctions) in range n->m]
```

The options '-n' and '-m' allow the user to specify which eigenvalues (energies) are output, with the bands being numbered 1, 2, 3, etc. Therefore, for bulk the top of the valence band is number 4, and the bottom of the conduction band number 5. For the first **k**-point in k.r, the eigenvalues are written to Ek0.r, and for the second point to Ek1.r, and so on. Thus, some postprocessing is required if the E–**k** curves are to be plotted. If the '-w' option is included, i.e.

```
pplb -n 1 -m 5 -w
```

then the eigenvectors (wave functions) will be written (in vertical columns) to the file ank.r.

Charge density (ppcd)

This program takes the eigenvectors output from pplb and turns them into a form suitable for plotting.

```
Usage: ppcd [-x # (0A0)][-X (0A0)]
            [-y # (0A0)][-Y (0A0)]
            [-z # (0A0)][-Z (1A0)]
            [-N # points per A0 20]
            [-n # lowest band 1][-m highest band 4, lowest band in ank file=1
            [-A Lattice constant (5.65Angstrom)]
```

This time, the options '`-n`' and '`-m`' imply a sum over these states, note that '`ppcd`' does not know what states where output from the previous '`pplb`' stage, and hence these states are numbered again, with the lowest state within the `ank.r` file being referred to as '1'. The option pairs '`-x`' and '`-X`', etc., specify the minimum and maximum extent for calculating the charge density; note that the units here are in lattice constants (A_0). With this setup '`ppcd`' can be used to plot the charge density along an axis; for example, with:

```
ppcd -z 0 -Z 2 -N 40
```

the charge density will be generated at 160 points spread across four lattice constants along the $x = y = 0$ axis, whereas:

```
ppcd -x 0 -X 1 -y 0 -Y 1 -z 0 -Z 0
```

will generate the charge density across the z=0 plane. With 40 points per A_0, this would give 40^2 points in total.

The pseudopotential form factors $V_f(q)$ (`ppfvq`)

This code allows a visual check of the pseudopotential form factors:

```
Usage: ppvfq [-A lattice constant (5.65A)][-a atom type SI]
             [-g maximum value of |G| (4*2*pi/A0)]
             [-N number of q points 100]
```

It simply generates the form factor versus q data in two-column format and then writes them to a file. The filename is dependent upon the choice of the atomic form factor; for example:

```
ppvfq -a GA
```

writes 100 points between $q = 0$ and $q = 8\pi/A_0$ to the file `VfqGA.r`.

Appendix J: Computer codes used in Chapter 10

Crystal Structure of single-spiral (`csss`)

For the pseudopotential calculation of a quantum well or superlattice the unit cell can be constructed by using '`csss`'. Its mode of operation is very similar to '`cszb`':

```
Usage:  csss [-a anion AS][-c cation GA]
             [-z # cells 1][-A lattice constant (5.65A)]
```

Superlattice wave (k) vectors (`slk`)

The perturbative approach to a superlattice requires a particular set of electron wave vectors in the file `k.r`. These can be generated with the code '`slk`', i.e.

```
Usage: slk [-k minizone k-point (0 2*pi/(n_z A0))][-z # cells 1]
```

The option '`-z`' enables the user to specify the period of the superlattice. The electron wave vectors themselves are given by using the option '`-k`'. Therefore,

the generation of the required wave vectors for a perturbative calculation of the centre of the superlattice Brillouin zone would require:

```
slk -k 0
```

whereas the edge of the superlattice Brillouin zone would require:

```
slk -k 1
```

(Perturbative) superlattice calculation (`ppsl`)

The perturbative approach to the superlattice requires the bulk band structure to be known at all of the appropriate electron wave vectors. On generation of these by using '`slk`' the bulk band structure of the host crystal must then be solved using '`pplb`' with a two-atom basis in the usual manner. The eigenvectors output from this calculation then form the expansion set for the superlattice calculation. A valence-band calculation may require all of the bulk bands as a basis set, and hence the bulk calculation may proceed by using the following:

```
pplb -n 1 -m 4 -w
```

With this perturbative approach, the superlattice unit cell requires two XYZ format files. The first of these is of the same size as the superlattice unit cell but contains just the host (bulk) atoms, this is, as ever, called `atoms.xyz`. The second contains the exact superlattice unit cell with the required heterostructure defined; this is called `atomsp.xyz`, with the addition `p` representing 'perturbed'. The superlattice calculation itself then deduces the perturbation as the *difference* between these two XYZ files.

This superlattice calculation has arguments which can be invoked on execution:

```
Usage: ppsl [-A lattice constant (5.65A)][-f (0kV/cm)]
            [-n # lowest band 1][-m highest band 4], output eigenvalues
            [-o output field FT][-p particle (e or h)]
```

'`ppsl`' automatically detects the number of basis functions in the expansion set and outputs the resulting eigenvalues of the superlattice to the file `Exi.r`. The only new options introduced by this module are those relating to the applied electric field. As before, '`-p`' indicates which type of particle is being investigated, this is important when an electric field is applied as the electrons move in difference directions to the holes. The electric field strength itself is specfied with '`-f`'. Finally '`-o`' writes the Fourier transform of the electric field potential to the external file `VFg.r`.

Appendix K: Computer codes used in Chapter 11

No new computational codes were used in Chapter 11, only utilities that generate input data (in particular, crystal structures) for '`pplb`'.

Crystal structure of a quantum dot (`csqd`)

This command simply substitutes a cube of atoms within a larger cube, thus creating a quantum dot. This utility is an awk script, and hence its execution is quite different to those described previously, i.e.

```
nawk -v BARRIER=1 -v SIDE=5 -v A0=5.65 -f csqd.awk \
  atoms.xyz > qd.atoms.xyz
```

This would place a cube of side three lattice constants within a larger cube of five lattice constants, defined in the file `atoms.xyz` into the new file `qd.atoms.xyz`. Note the line continuation character '\'; this just allows the command to be spread over two lines.

Crystal structure of a pyramidal dot (cspd)

This command is similar to the above, but in this case substitutes certain atoms within a larger cube to create a pyramidal dot. Execution is carried out by using:

```
nawk -v BARRIER=1 -v BASE=5 -v HEIGHT=5 -v A0=5.65 -f cspd.awk \
  atoms.xyz > qd.atoms.xyz
```

Appendix L: Computer code summary

Table L.1 summarises the executable filenames and Table L.2 the data filenames, produced or used by the executable codes above.

Table L.1 Executable filenames and summary of action

Command	Derived from
`cspd`	Crystal Structure Pyramidal Dot
`csqd`	Crystal Structure Quantum Dot
`csss`	Crystal Structure Single Spiral
`cszb`	Crystal Structure Zinc Blende
`d0`	D^0 (neutral donor binding energy)
`d02D`	D^0 (2D model)
`d03D`	D^0 (3D model)
`dos`	Density of States
`ebe`	Exciton Binding Energy

Command	Derived from
`efcwire`	Envelope Function Circular WIRE
`efcwwf`	Envelope Function Circular Wire Wave Functions
`effv`	Envelope Function Field potential (V)
`efiw`	Envelope Function Infinite Well
`efiwire`	Envelope Function Infinite WIRE
`efkpsl`	Envelope Function Kronig–Penney SuperLattice
`efmfv`	Envelope Function Magnetic Field Potential (V)
`efpqw`	Envelope Function Parabolic Quantum Well
`efsdot`	Envelope Function Spherical DOT
`efshoot`	Envelope Function SHOOTing solution
`efsqw`	Envelope Function Single Quantum Well
`efsx`	Envelope Function Structure to alloy profile (x)
`efwf`	Envelope Function Wave Functions
`efxm`	Envelope Function alloy profile (X) to Mass
`efxv`	Envelope Function alloy profile (X) to potential (V)
`gde`	General solution to Diffusion Equation
`hup`	Heisenberg's Uncertainty Principle
`ivdb`	current–voltage (I–V) for Double Barrier
`ovl`	OVerLap integral
`ppcd`	PseudoPotential Charge Density
`pplb`	PseudoPotential Large Basis
`ppsg`	PseudoPotential Sort **G** (reciprocal lattice vectors)
`ppsl`	PseudoPotential (perturbative) SuperLattice
`ppvfq`	Pseudopotential $V_f(q)$ (form factor versus q)
`pth`	Pöschl–Teller potential Hole
`rlv-fcc`	Reciprocal Lattice Vectors Face-Centred-Cubic lattice
`rlv-sc`	Reciprocal Lattice Vectors Simple Cubic
`rlv-ss`	Reciprocal Lattice Vectors Single Spiral
`sbp`	SubBand Populations
`scp`	Self-Consistent Poisson
`sfr`	Spin-Flip Raman
`slk`	SuperLattice **k**-vectors

Command	Derived from
`srcc`	Scattering Rate Carrier–Carrier
`srelo`	Scattering Rate Electron–LO phonon
`srmpr`	Scattering Rate Mean Phonon Rate
`srrad`	Scattering Rate spontaneous RADiative
`tdb`	Transmission coefficient Double Barrier
`tsb`	Transmission coefficient Single Barrier
`xyz2pdb`	XYZ to Protein DataBase format

Table L.2 Data filenames and summary of content

Filename	Content
`Aijfg.r`	carrier–carrier form factors $A_{\mathrm{ijfg}}(q_{xy})$ as a function of q_{xy}
`ank.r`	pseudopotential eigenvectors $a_{n,\mathbf{k}}(\mathbf{G})$
`atoms.xyz`	pseudopotential unit cell in XYZ format
`atomsp.xyz`	perturbed unit cell for superlattice calculation
`ccijfg.r`	Carrier–Carrier $\lvert\mathrm{i}\rangle\lvert\mathrm{j}\rangle\rightarrow\lvert\mathrm{f}\rangle\lvert\mathrm{g}\rangle$ scattering rate
`ccABCD.r`	thermally averaged Carrier–Carrier rates
`cd.r`	Charge Density from pseudopotential calculations
`cdnm.r`	Charge Density of quantum wire states
`d.r`	Dopant profile
`e.r`	donor binding Energy for each donor position
`Ekk.r`	pseudopotential Energy eigenvalues at each k-point
`Ef.r`	subband quasi-Fermi Energies
`Ep.r`	Energy levels for electron (p = e), hole (p = h), etc.
`EX0.r`	neutral-Exciton (X0) binding energy
`Exi.r`	superlattice Energy eigenvalues from '`ppsl`'
`e_sf.r`	Spin-Flip Energies
`F.r`	Poisson electric Field
`FDn.r`	Fermi–Dirac distribution function of subband n

Filename	Content
`G.r`	pseudopotential reciprocal lattice vectors $\mathbf{G}$
`I.r`	Raman signal Intensity versus energy (in wavenumbers)
`IV.r`	current (I)-Voltage curves
`k.r`	Electron wave vectors ($\mathbf{k}$)
`l.r`	donor Bohr radius (λ) for each r_d
`m.r`	effective Mass
`LOaif.r`	LO–phonon Absorption $\|i\rangle \rightarrow \|f\rangle$ rate
`LOa-if.r`	thermally averaged LO–phonon Absorption rates
`LOeif.r`	LO–phonon Emission $\|i\rangle \rightarrow \|f\rangle$ rate
`LOe-if.r`	thermally averaged LO–phonon Emission rates
`N.r`	Number of carriers per unit area in each subband
`Nu.r`	Unnormalised `N.r`
`r_d.r`	donor positions (r_d)
`rho.r`	Poisson charge density (ρ)
`s.r`	Structure definition file
`sigma.r`	Poisson sheet charge density (σ)
`T.r`	Transmission coefficient versus energy
`V.r`	Possion potential (V_ρ) profile
`VfqTYPE.r`	pseudopotential form factor of atom TYPE
`VFg.r`	Fourier transform of the electric field potential
`v.r`	potential (V) profile
`v0.r`	potential without field contribution
`v1.r`	potential without Poission contribution
`wf_pn.r`	wave function of particle ‘p’ in level ‘n’
`X.r`	diffusant profile
`x.r`	alloy concentration profile
`zeta.r`	the variational parameter ζ for each r_d