

Quantum confinement effects in semiconductor clusters

Computational Part III Project
Supervisor: Dr G Rajagopal
Author: Paul Wright
Churchill College, Cambridge

September 20, 2000

Abstract

Using the empirical pseudopotential method (EPM), the band structures and exciton energies of CdS, GaAs and GaN semiconductor clusters are calculated. The variation of the exciton energies with the size, shape and crystal structure of the cluster is examined. The calculated exciton energies for CdS are in good agreement with experimental data. The exciton energies of small CdS clusters are sensitive to whether the crystal structure is hexagonal or zinc-blende. In GaAs the exciton energies decrease with decreasing cluster size for small clusters, leading to a red-shift in their absorption spectra, whereas in CdS the exciton energy always increases with decreasing cluster size. The first predictions for exciton energies in GaN clusters are made. In GaN there is no decrease in exciton energy with decreasing cluster size. The optical spectroscopy of small semiconductor clusters is radically different from those of large clusters or bulk semiconductors. The use of the EPM allows prediction of effects which cannot be explained by the effective mass model.

Except where specific reference is made to the work of others, this work is original and has not already been submitted either wholly or in part to satisfy and degree requirement at this or any university.

Contents

1	Introduction	3
2	The Effective Mass Model	4
2.1	Review of the Effective Mass Model	4
2.2	Application to clusters of semiconductors	4
2.3	Limitations of the Effective Mass Model	5
3	The Empirical Pseudopotential Method	6
3.1	The diamond structure	7
3.2	The zinc-blende structure	7
3.3	The hexagonal structure	8
4	Method for calculations using the EPM	10
4.1	Use of the EPM to calculate electronic energies	10
4.2	Computation of bulk and cluster band gaps	10
5	Calculation of exciton energies	12
6	Results of EPM calculations	13
6.1	Results for CdS clusters	13
6.1.1	The effect of size	13
6.1.2	The effect of structure	16
6.1.3	The effect of shape	19
6.2	Results for GaAs clusters	19
6.3	Results for GaN clusters	20
7	Conclusion	23
8	Acknowledgements	23
A	Appendix: Computational matters	25
A.1	Overview of the workings of the programs	25
A.2	Source code	25
A.2.1	sethamil.F	25
A.2.2	lattice.inc	41
A.2.3	constants.inc	42
A.2.4	potential.inc	42
A.2.5	bandcalc.F	43
A.2.6	exciton.F	49
A.2.7	makefile	52
A.2.8	Example input file	52

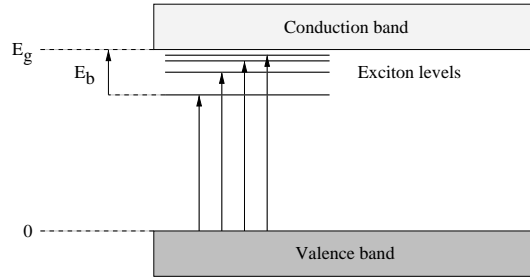


Figure 1: The optical transitions indicated by arrows show electrons from the top of the valence band forming excitons. E_b is the exciton binding energy. E_g is the band gap.

1 Introduction

The optical spectra of semiconductors may have structure for photon energies just below the band gap, where it might be expected that the crystal would be transparent. This structure is caused by the creation of a bound electron-hole pair, known as an exciton. The exciton is bound by the Coulombic attraction between the electron and the hole. The energy levels of an exciton created in a direct band gap semiconductor are shown in Figure 1.

In clusters, it is found that the band gap increases. This means that the first absorption peak in the cluster is normally blue shifted with respect to the bulk semiconductor.

Rama Krishna and Friesner[1] have made effective use of the empirical pseudopotential method (EPM) to predict band gaps and exciton energies in clusters of CdS, GaAs and GaP. In the case of CdS, the work of Wang and Herron[2] has enabled these predictions to be confirmed experimentally.

In this paper we will discuss the EMM in Section 2 and the EPM in Section 3, and examine the use of these methods to predict the electronic structure of clusters in Section 4. In Section 5 the calculation of exciton energies is discussed. In Section 6 we attempt to reproduce the results of Rama Krishna and Friesner[1] for CdS and GaAs and additionally make use of pseudopotentials for GaN to make new predictions concerning the properties of GaN clusters, before drawing some conclusions in Section 7. Appendix A contains details on and listings for the programs used to make the calculations in this paper.

2 The Effective Mass Model

2.1 Review of the Effective Mass Model

The effective mass model is a method for approximating the band-structure of a solid using a free electron-like formalism. The free electron energy-wave vector relation

$$E = \frac{\hbar^2 k^2}{2m} \quad (1)$$

can be thought of as defining a mass m which determines the curvature of the band.

We will assume a direct band gap semiconductor with the band gap at $k = 0$. Expanding about $k = 0$, for a given band we have

$$E \simeq E_0 + \frac{1}{2} \sum_{i,j} \left. \frac{\partial^2 E}{\partial k_i \partial k_j} \right|_{k=0} k_i k_j \quad (2)$$

By comparison with Equation 1 this leads to the definition of the effective mass tensor

$$\overline{\overline{M}}_{ij} = \frac{\hbar^2}{\frac{\partial^2 E}{\partial k_i \partial k_j}} \quad (3)$$

Near $k = 0$ at the bottom of the conduction band the tensor is nearly isotropic with an average diagonal element m_e , giving the energy of an electron in a given k -state as

$$E_e(k) = E_c + \frac{\hbar^2 k^2}{2m_e} \quad (4)$$

where E_c is the energy of the bottom of the conduction band. The quantity m_e is known as the effective mass of the electron.

The band structure near the top of the valence band can be complicated by spin-orbit splitting of the orbitals which form the bands near the top of the valence band, but under some circumstances we can define an effective mass for a “missing electron” or hole in the valence band by the curvature of that band, giving the energy of a hole in a given k -state as

$$E_h(k) = E_v - \frac{\hbar^2 k^2}{2m_h} \quad (5)$$

where E_v is the energy of the top of the valence band. The sign change is introduced to ensure that the hole effective mass m_h is a positive quantity.

Effective masses can be determined experimentally, for example by cyclotron resonance[8, page 196].

2.2 Application to clusters of semiconductors

The exciton absorption levels depend on both the binding energy of the exciton and the band gap in the cluster (see Figure 1). To predict exciton spectra for clusters we must determine both of these energies. We first concern ourselves with the band gap in the cluster.

In an infinite solid, the band structure describes a continuous E - k relation. In small clusters the quantisation of k becomes important, and we find there is some minimum, non-zero value of k determined by the size of the cluster. We find this value of k by treating an electron and a hole as “pseudo-particles”

confined by the cluster. Using the lowest allowed k value for these confined particles, we subtract the energies E_e and E_h given by Equations 4 and 5 to give the band gap in the cluster, since the difference between these two energies is the minimum energy separation between the valence and conduction bands for the allowed states in the cluster.

We model the electron or hole in the cluster as a particle in a spherical infinite potential well of radius R . The solutions for the particle wave function are proportional to spherical Bessel functions $j_l(kr)$. The particle in this well is taken to have a mass equal to the electron mass or hole mass, so the k in the argument to the spherical Bessel function corresponds to the k for an electron or hole[7].

Taking the lowest allowed state $j_0(kr)$ and using the boundary condition that $j_0(kR) = 0$, the lowest allowed value of k is given by

$$k_{min}^2 = \frac{\pi^2}{R^2} \quad (6)$$

So, using this value of k and Equations 4 and 5 we obtain

$$E_{g,cluster} = E_e(k_{min}) - E_h(k_{min}) = E_{g,bulk} + \frac{\hbar^2 \pi^2}{2R^2} \left(\frac{1}{m_e} + \frac{1}{m_h} \right) \quad (7)$$

2.3 Limitations of the Effective Mass Model

One limitation of the EMM is inherent in the derivation in Section 2.1: it is a quadratic approximation where we have assumed that the effective masses are independent of k . A deviation of the bands from the expected parabolic form will cause our calculation to be in error. Also we measure effective masses in bulk solids: if the crystal structure changes in the clusters the effective mass will also change. We shall shortly see that the EPM provides a more accurate method for calculating band gaps in clusters than the EMM.

3 The Empirical Pseudopotential Method

The wave functions and energies of electrons in a crystal are given by solving the Schrödinger equation

$$H\psi_{n,\mathbf{k}} = E_n(\mathbf{k})\psi_{n,\mathbf{k}}(\mathbf{r}) \quad (8)$$

where H is the Hamiltonian of the system, $\psi_{n,\mathbf{k}}$ are its wave functions, $E_n(\mathbf{k})$ the corresponding eigenvalues, \mathbf{k} the wave vectors of the wave functions, and n is the band index.

For the large number of electrons involved even in small clusters, exact solution of this equation is computationally unfeasible. Several approximations are used to convert this problem into one which can be solved in a reasonable amount of time. These are

1. The inner electrons are considered to be attached to the atom cores, only the valence electrons are free to move.
2. The cores are fixed in position, the Born-Oppenheimer approximation.
3. Each valence electron moves in the mean field of the fixed cores and the other valence electrons.

These assumptions give rise to the effective Hamiltonian

$$H = -\frac{\hbar^2}{2m}\nabla^2 + V_p(\mathbf{r}) \quad (9)$$

where $V_p(\mathbf{r})$ is a pseudopotential. In this paper, the pseudopotential is determined using the EPM.

In the EPM, we write the pseudopotential as

$$V_p(\mathbf{r}) = \sum_{\mathbf{R},j} v_j(\mathbf{r} - \mathbf{R} - \mathbf{d}_j) \quad (10)$$

where \mathbf{R} is the position of a lattice site, \mathbf{d}_j is the position of the j th basis atom with respect to the lattice site, and v_j is the atomic pseudopotential of the j th basis atom. The sum is taken over all lattice sites and basis atoms.

Fourier expansion of v_j gives

$$v_j(\mathbf{r} - \mathbf{R} - \mathbf{d}_j) = \frac{1}{Nn_a} \sum_{\mathbf{G}} v_j(\mathbf{G}) \exp[i\mathbf{G} \cdot (\mathbf{r} - \mathbf{R} - \mathbf{d}_j)] \quad (11)$$

where \mathbf{G} are the reciprocal lattice vectors, N is the number of lattice sites, n_a the number of basis atoms on each lattice site, and the form factor

$$v_j(\mathbf{G}) = \frac{n_a}{\Omega_0} \int_V d\mathbf{r} v_j(\mathbf{r}) \exp(-i\mathbf{G} \cdot \mathbf{r}) \quad (12)$$

where Ω_0 is the volume of the unit cell. There is a condition $v_j^*(\mathbf{G}) = v_j(-\mathbf{G})$ on $v_j(\mathbf{G})$ to ensure that $v_j(\mathbf{r})$ is real.

If the atomic potential $v_j(\mathbf{r}) = v_j(|\mathbf{r}|)$, as we might expect for a spherically symmetric atom, then from Equation 12 we obtain

$$v_j(\mathbf{G}) = \frac{4\pi n_a}{\Omega_0} \int_V dr r^2 j_0(Gr) v_j(r) \quad (13)$$

$$= v_j(|\mathbf{G}|) \quad (14)$$

where j_0 is the zeroth order spherical Bessel function. That is, for symmetric atomic potentials, the form factors $v_j(\mathbf{G})$ depend on the magnitude of \mathbf{G} only.

Substitution of Equation 11 into Equation 10 gives

$$V_p(\mathbf{r}) = \frac{1}{Nn_a} \sum_{\mathbf{G}} \sum_{\mathbf{R}, j} v_j(\mathbf{G}) \exp[i\mathbf{G} \cdot (\mathbf{r} - \mathbf{R} - \mathbf{d}_j)] \quad (15)$$

For the structures under consideration, the second summation over \mathbf{R} and j can be carried out separately, leaving $V_p(\mathbf{r})$ as a closed form function of $v_j(\mathbf{G})$ only. The factors $v_j(\mathbf{G})$ are determined empirically from optical data giving known energies in the band structure: Aourag[6] describes an iterative process for determining these factors.

3.1 The diamond structure

As a simple example of the use of Equation 15 we consider the diamond structure. The lattice is a fcc lattice with basis atoms at $\pm(1, 1, 1)a_0/8$ where a_0 is the lattice constant. Hence we take $n_a = 2$ and $\mathbf{d}_1 = -\mathbf{d}_2 = -\mathbf{t}_1$ where $\mathbf{t}_1 = (1, 1, 1)a_0/8$. Since the two atoms are of the same type $v_1 = v_2$. From Equation 15 we obtain

$$V_p(\mathbf{r}) = \frac{1}{2N} \sum_{\mathbf{G}} v_1(\mathbf{G}) \exp(i\mathbf{G} \cdot \mathbf{r}) \sum_{\mathbf{R}} \exp(-i\mathbf{G} \cdot \mathbf{R}) \sum_j \exp(-i\mathbf{G} \cdot \mathbf{d}_j) \quad (16)$$

Now $\mathbf{G} \cdot \mathbf{R} = 2n\pi$ by definition, so

$$\sum_{\mathbf{R}} \exp(-i\mathbf{G} \cdot \mathbf{R}) = \sum_{\mathbf{R}} 1 = N \quad (17)$$

and hence

$$V_p(\mathbf{r}) = \sum_{\mathbf{G}} v_1(\mathbf{G}) \exp(i\mathbf{G} \cdot \mathbf{r}) \frac{1}{2} [\exp(i\mathbf{G} \cdot \mathbf{t}_1) + \exp(-i\mathbf{G} \cdot \mathbf{t}_1)] \quad (18)$$

$$= \sum_{\mathbf{G}} v_1(\mathbf{G}) \exp(i\mathbf{G} \cdot \mathbf{r}) \cos(\mathbf{G} \cdot \mathbf{t}_1) \quad (19)$$

$$= \sum_{\mathbf{G}} v_1(\mathbf{G}) S(\mathbf{G}) \exp(i\mathbf{G} \cdot \mathbf{r}) \quad (20)$$

where

$$S(\mathbf{G}) = \cos(\mathbf{G} \cdot \mathbf{t}_1) \quad (21)$$

We find that the potential $V_p(\mathbf{r})$ is given by an empirically determined form factor $v_1(\mathbf{G})$ and a fixed structure factor $S(\mathbf{G})$.

3.2 The zinc-blende structure

The zinc-blende structure is similar to the diamond structure but with two different basis atoms, so $v_1 \neq v_2$. From Equation 15 using Equation 17 we have

$$V_p(\mathbf{r}) = \frac{1}{n_a} \sum_{\mathbf{G}} \exp(i\mathbf{G} \cdot \mathbf{r}) \sum_j v_j(G) \exp(-i\mathbf{G} \cdot \mathbf{d}_j) \quad (22)$$

$$= \sum_{\mathbf{G}} \exp(i\mathbf{G} \cdot \mathbf{r}) \frac{1}{2} [v_1(G) \exp(i\mathbf{G} \cdot \mathbf{t}_1) + v_2(G) \exp(-i\mathbf{G} \cdot \mathbf{t}_1)] \quad (23)$$

Using $\exp(i\theta) = \cos(\theta) + i\sin(\theta)$ and collecting terms in $\cos(\mathbf{G} \cdot \mathbf{t}_1)$ and $\sin(\mathbf{G} \cdot \mathbf{t}_1)$ we obtain

$$V_p(\mathbf{r}) = \sum_{\mathbf{G}} \exp(i\mathbf{G} \cdot \mathbf{r}) [V_S(G) \cos \mathbf{G} \cdot \mathbf{t}_1 + iV_A(G) \sin(\mathbf{G} \cdot \mathbf{t}_1)] \quad (24)$$

$$= \sum_{\mathbf{G}} \exp(i\mathbf{G} \cdot \mathbf{r}) [V_S(G)S_S(\mathbf{G}) + iV_A(G)S_A(\mathbf{G})] \quad (25)$$

where

$$V_S(G) = \frac{1}{2} [v_1(G) + v_2(G)]$$

$$V_A(G) = \frac{1}{2} [v_1(G) - v_2(G)] \quad (26)$$

$$(27)$$

and

$$S_S(\mathbf{G}) = \cos(\mathbf{G} \cdot \mathbf{t}_1)$$

$$S_A(\mathbf{G}) = \sin(\mathbf{G} \cdot \mathbf{t}_1) \quad (28)$$

For computational purposes, it is useful to define \mathbf{G} in units of $2\pi/a_0$

$$\mathbf{G} = \frac{2\pi}{a_0} (G_x, G_y, G_z) \quad (29)$$

from which, by Equation 28, we obtain

$$S_S(\mathbf{G}) = \cos \left[\frac{\pi}{4} (G_x + G_y + G_z) \right] \quad (30)$$

$$S_A(\mathbf{G}) = \sin \left[\frac{\pi}{4} (G_x + G_y + G_z) \right] \quad (31)$$

3.3 The hexagonal structure

In the hexagonal structure there are four basis atoms of two different types on each lattice site. The structure is characterised by three lengths a_0 , c_0 and u_0 . If we assume perfect packing then

$$c_0/a_0 = \sqrt{8/3}, \quad u_0 = 0.375 \quad (32)$$

The primitive lattice translation vectors are

$$\mathbf{a}_1 = \left(\frac{\sqrt{3}}{2}, -\frac{1}{2}, 0 \right) a_0$$

$$\mathbf{a}_2 = (0, 1, 0) a_0 \quad (33)$$

$$\mathbf{a}_3 = \left(0, 0, \frac{c_0}{a_0} \right) a_0$$

and the basis atom positions are

$$\mathbf{d}_1 = \left(-\frac{1}{4\sqrt{3}}, -\frac{1}{12}, -\frac{1-2u_0}{\sqrt{6}} \right) a_0 = -\mathbf{d}_4$$

$$\mathbf{d}_2 = \left(-\frac{1}{4\sqrt{3}}, -\frac{1}{12}, -\frac{1+2u_0}{\sqrt{6}} \right) a_0 = -\mathbf{d}_3 \quad (34)$$

where atoms 1 and 3, and 2 and 4 are identical. Note that there seems to be an error in Rama Krishna and Freisner's labelling of these vectors[1, equation 27], which leads to an incorrect formula for S_A if carried through: atoms of different types are related to each other by inversion through the origin, and not atoms of the same type[12].

So from Equations 22 and 34 we have

$$V_p(\mathbf{r}) = \sum_{\mathbf{G}} \exp(-\mathbf{G} \cdot \mathbf{r}) \frac{1}{4} \{v_1(\mathbf{G})[\exp(i\mathbf{G} \cdot \mathbf{d}_2) + \exp(-i\mathbf{G} \cdot \mathbf{d}_1)] + v_2(\mathbf{G})[\exp(i\mathbf{G} \cdot \mathbf{d}_1) + \exp(-i\mathbf{G} \cdot \mathbf{d}_2)]\} \quad (35)$$

If we write v_1 and v_2 in terms of V_S and V_A using Equation 26, after some trigonometric manipulation we obtain

$$V_p(\mathbf{r}) = \sum_{\mathbf{G}} \exp(-\mathbf{G} \cdot \mathbf{r}) [V_S(\mathbf{G}) \cos\left(\frac{\mathbf{d}_1 + \mathbf{d}_2}{2}\right) \cos\left(\frac{\mathbf{d}_1 - \mathbf{d}_2}{2}\right) + iV_A(\mathbf{G}) \cos\left(\frac{\mathbf{d}_1 + \mathbf{d}_2}{2}\right) \sin\left(\frac{\mathbf{d}_1 - \mathbf{d}_2}{2}\right)] \quad (36)$$

By assuming a hard sphere packing, we can relate the lattice constant in the zinc-blende and hexagonal forms by

$$a_0(\text{ZB}) = \sqrt{2}a_0(\text{hex}) \quad (37)$$

and by analogy with Equation 29 we define

$$\mathbf{G} = \frac{\sqrt{2}\pi}{a_0(\text{hex})} (G_x, G_y, G_z) \quad (38)$$

From Equations 36,38 and 34 we obtain Equation 25 again, but with

$$\begin{aligned} S_S(\mathbf{G}) &= \cos(\mathbf{G} \cdot \mathbf{t}_2) \cos(2\pi u_0 G_z / \sqrt{3}) \\ S_A(\mathbf{G}) &= \cos(\mathbf{G} \cdot \mathbf{t}_2) \sin(2\pi u_0 G_z / \sqrt{3}) \end{aligned} \quad (39)$$

where $\mathbf{t} = (1/\sqrt{48}, 1/12, 1/\sqrt{6})a_0$.

This differs slightly from the formulae given by Rama Krishna and Freisner[1, equation 32]. Their formulation is consistent if we define a vector $\mathbf{g} = (G_x, G_y, G_z)$ using the definition of G_x etc. from Equation 38, that is, \mathbf{g} is the numerical value of the \mathbf{G} vector in units of $\sqrt{2}\pi/a_0$. If we also define a vector $\mathbf{t}'_2 = (1/\sqrt{48}, 1/12, 1/\sqrt{6})$, we obtain the following expressions

$$\begin{aligned} S_S(\mathbf{G}) &= \cos(\sqrt{2}\pi \mathbf{g} \cdot \mathbf{t}'_2) \cos(2\pi u_0 G_z / \sqrt{3}) \\ S_A(\mathbf{G}) &= \cos(\sqrt{2}\pi \mathbf{g} \cdot \mathbf{t}'_2) \sin(2\pi u_0 G_z / \sqrt{3}) \end{aligned} \quad (40)$$

These are useful computationally as \mathbf{G} vectors are stored in units of $\sqrt{2}\pi/a_0$ in the programs used to perform the EPM calculations.

4 Method for calculations using the EPM

4.1 Use of the EPM to calculate electronic energies

To make use of the expressions for $V_p(\mathbf{r})$ we have derived, we expand the wave function using a plane wave basis set.

$$\psi_{n,\mathbf{k}}(\mathbf{r}) = \frac{1}{\sqrt{V}} \sum_{\mathbf{G}} a_{n,\mathbf{k}}(\mathbf{G}) \exp[i(\mathbf{G} + \mathbf{k}) \cdot \mathbf{r}] \quad (41)$$

We insert this expression for $\psi_{n,\mathbf{k}}(\mathbf{r})$ into Equation 8 to give

$$\sum_{\mathbf{G}} H \exp[i(\mathbf{G} + \mathbf{k}) \cdot \mathbf{r}] = \sum_{\mathbf{G}} E_{n,\mathbf{k}} a_{n,\mathbf{k}}(\mathbf{G}) \exp[i(\mathbf{G} + \mathbf{k}) \cdot \mathbf{r}] \quad (42)$$

Multiplying both sides by $\exp[-(\mathbf{G}' + \mathbf{k}) \cdot \mathbf{r}]$ and integrating over all volume we obtain the set of equations

$$\sum_{\mathbf{G}} [H_{\mathbf{G}',\mathbf{G}}(\mathbf{k}) - E_{n,\mathbf{k}} \delta_{\mathbf{G},\mathbf{G}'}] a_{n,\mathbf{k}}(\mathbf{G}) = 0 \quad (43)$$

where, from Equation 9 the matrix elements $H_{\mathbf{G}',\mathbf{G}}(\mathbf{k})$ are given by

$$H_{\mathbf{G}',\mathbf{G}}(\mathbf{k}) = \frac{1}{V} \int_V d\mathbf{r} \exp[-(\mathbf{G}' + \mathbf{k}) \cdot \mathbf{r}] H \exp[i(\mathbf{G} + \mathbf{k}) \cdot \mathbf{r}] \quad (44)$$

$$= \frac{\hbar^2}{2m} (\mathbf{G} + \mathbf{k})^2 \delta_{\mathbf{G},\mathbf{G}'} + V(\mathbf{G} - \mathbf{G}') \quad (45)$$

Hence for both the zinc-blende and hexagonal structures, the matrix elements are given by

$$\begin{aligned} H_{\mathbf{G}',\mathbf{G}}(\mathbf{k}) &= \frac{\hbar^2}{2m} (\mathbf{G} + \mathbf{k})^2 \delta_{\mathbf{G},\mathbf{G}'} + V_S(|\mathbf{G} - \mathbf{G}'|) S_S(\mathbf{G} - \mathbf{G}') \\ &\quad + iV_A(|\mathbf{G} - \mathbf{G}'|) S_A(\mathbf{G} - \mathbf{G}') \end{aligned} \quad (46)$$

The energy eigenvalues of the electronic wave function may be obtained for a given \mathbf{k} by diagonalization of a matrix composed of the elements given above.

4.2 Computation of bulk and cluster band gaps

Equation 46 provides a matrix element for calculation of the electronic energies. Two Fortran-77 programs were used to calculate these energies in bulk semiconductors and in clusters. Appendix A contains the source code for the programs and some further details on them.

The band structures in Section 6 are labelled with the high symmetry points of the lattice. For the zinc-blende structure these are

$$\begin{aligned} X &= (1, 0, 0) & W &= (1, 0.5, 0) & L &= (0.5, 0.5, 0.5) \\ \Gamma &= (0, 0, 0) & U &= K = (0, 0.75, 0.75) \end{aligned} \quad (47)$$

in units of $2\pi/a_0$. For the hexagonal structure these points are

$$A = (0, 0, a_0\sqrt{2}c_0) \quad \Gamma = (0, 0, 0) \quad M = (\sqrt{2}/3, 0, 0) \quad (48)$$

in units of $\sqrt{2}\pi/a_0$.

To compute band gaps in spherical clusters, we use the quantisation condition (Equation 6 of Section 2.2) to find the value of k_{min} . The matrix is diagonalized for $\mathbf{k} = (1, 1, 1)k_{min}/\sqrt{3}$ and the band gap at this value of \mathbf{k} is determined by subtracting the energy of the band at the bottom of the valence band from the energy of the band at the top of the conduction band.

It is also possible to find energies in cubic clusters of side L . Using the standard “waves in a box” argument,

$$\mathbf{k}_{min} = \frac{\pi}{L}(1, 1, 1) \quad (49)$$

To produce graphs of E_{ex} versus R or L , the matrix is diagonalized with $\mathbf{k} = \mathbf{k}_{min}$ for various values of R or L .

5 Calculation of exciton energies

As we mentioned in Section 1, an exciton is a bound electron-hole pair. We consider Mott-Wannier excitons where the electron and hole attract each other via the Coloumb potential

$$U(r) = \frac{-e^2}{\epsilon r} \quad (50)$$

where ϵ is the dielectric constant and r the distance between the electron and hole. This leads to bound states of the system with energies lower than the bottom of the conduction band. In the bulk semiconductor, the problem is that of the hydrogen atom and the exciton states have energies given by the Rydberg equation

$$E_n = E_g - \frac{\mu e^4}{2\hbar^2 \epsilon^2 n^2} \quad (51)$$

where μ is the reduced mass

$$\mu = \frac{1}{\frac{1}{m_e} + \frac{1}{m_h}} \quad (52)$$

By analogy with the hydrogen atom, we can define an exciton Bohr radius

$$R_{ex} = \frac{2\epsilon\hbar^2}{\mu e^2} \quad (53)$$

For an exciton confined to a cluster of radius R , we can write the Hamiltonian of the confined electron and hole as

$$H = \frac{\mathbf{p}_e^2}{2m_e} + \frac{\mathbf{p}_h^2}{2m_h} - \frac{e^2}{\epsilon|\mathbf{r}_e - \mathbf{r}_h|} + V(r_e, r_h) \quad (54)$$

where \mathbf{r}_e and \mathbf{r}_h are the positions of the electron and hole respectively, and

$$V(r_e, r_h) = \begin{cases} 0 & r_e, r_h < R \\ \infty & \text{otherwise} \end{cases} \quad (55)$$

Using this Hamiltonian, work by Brus[9] and Kayanuma[4, 5] has produced the following expression for the exciton energy

$$E_{ex} = E_{g,bulk} + \frac{\hbar^2 \pi^2}{2R^2} \left(\frac{1}{m_e} + \frac{1}{m_h} \right) - \frac{1.786}{\epsilon R} - 0.248 \frac{\mu e^4}{2\hbar^2 \epsilon^2} \quad (56)$$

To obtain this result Brus[9] took a simple product wave function of the two lowest order spherical Bessel function $j_0(k_{min}r)$ (see Section 2.2) for the confined electron and hole. By this method the first three terms in Equation 56 are obtained. Kayanuma[4, 5] multiplied this wave function by an exponential correlation term producing the full expression by a variational approach. The expression is valid for the limit $R/R_{ex} \rightarrow 0$. The fourth term is the residual Rydberg energy in this limit.

We can identify the second term in Equation 56 as a confinement term and the third term as a Coulombic term. Further, from Equation 7 in Section 2.2 we can identify the first two terms in Equation 56 as the cluster band gap obtained using the EMM. Since the EPM provides us with a method to evaluate this band gap directly from the computed band structure (see Section 4.2) we can use the EPM value for $E_{g,cluster}$ and write

$$E_{ex} = E_{g,cluster} - \frac{1.786}{\epsilon R} - 0.248 \frac{\mu e^4}{2\hbar^2 \epsilon^2} \quad (57)$$

We now proceed to calculate the exciton energies for various materials and, where possible, to compare them with experimental values.

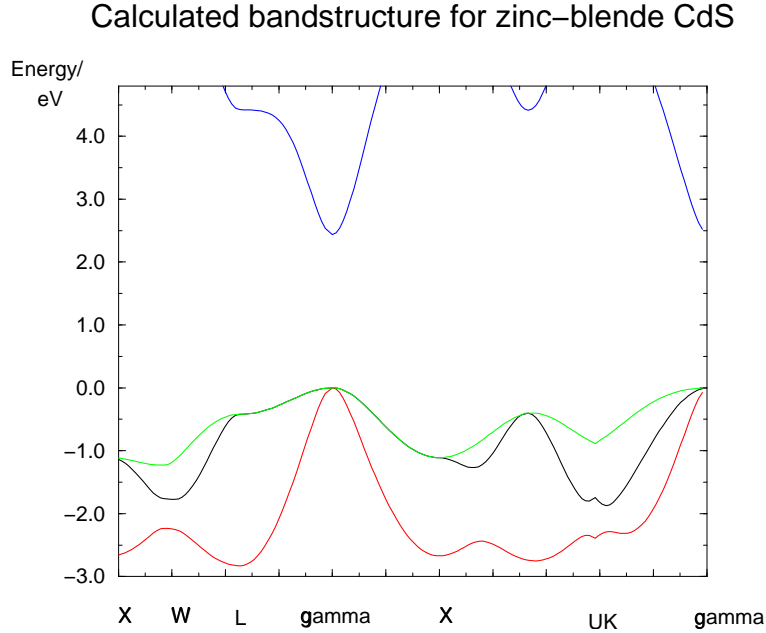


Figure 2: The band structure of CdS near the top of the valence band.

6 Results of EPM calculations

6.1 Results for CdS clusters

CdS exists in both zinc-blende and hexagonal forms. There is good experimental data on exciton energies in small clusters of CdS[2]. The EPM has been used to study the effect on the exciton energy of modifications to the size, shape and crystal structure of the clusters.

The form factors given in Table 1 were used to produce the band structure for bulk CdS shown in Figure 2. A 137×137 matrix was diagonalized for each value of \mathbf{k} sampled. The band gap predicted by the EPM is 2.44 eV, which is in good agreement with the experimental value of 2.5 eV.

6.1.1 The effect of size

Using Equation 57, with $E_{g,cluster}$ taken from the EPM calculations using the method described in Section 4.2, the exciton energies of clusters of CdS were computed for a range of cluster sizes. A 411×411 matrix was diagonalized for each cluster. The E_g value taken from the EPM was increased by 0.06 eV, the underestimate in the energy for the bulk solid band gap. Initially, the bulk solid value of the lattice parameter a_0 was used. All the material properties used in these calculations may be found in Table 2.

The results of this calculation are shown in Figure 3, along with the experimental data of Wang and Herron[2] and the prediction of the effective mass model.

Wang and Herron[2] have shown by X-ray diffraction that the lattice constant in the clusters decreases with cluster size. They provide peak positions for the (111) peak which may be used to compute the new lattice parameter via the

Table 1: Pseudopotential form factors used in the calculations.

Material	G^2	$V_S/\text{a.u.}$	$V_A/\text{a.u.}$
CdS (ZB)[1]	0	0.0	0.0
	3	-0.12	0.115
	4	0.0	0.065
	8	0.015	0.0
	11	0.020	0.025
	12	0.0	0.025
CdS (hex)[1]	0	0.0	0.0
	$\frac{3}{4}$	0.0	0.0
	$2\frac{2}{3}$	-0.145	0.0
	3	-0.10	0.115
	$3\frac{5}{12}$	-0.10	0.09
	$5\frac{2}{3}$	-0.012	0.04
	$6\frac{3}{4}$	0.0	0.0
	8	0.015	0.0
	$8\frac{3}{4}$	0.0	0.0
	$9\frac{5}{12}$	0.02	0.025
	$10\frac{2}{3}$	0.02	0.0
	11	0.02	0.025
	$11\frac{5}{12}$	0.02	0.025
	12	0.0	0.025
$13\frac{2}{3}$	0.01	0.015	
$14\frac{2}{3}$	0.0	0.01	
GaAs[1]	0	0.0	0.0
	3	-0.1225	0.031
	4	0.0	0.0175
	8	-0.0025	0.0
	11	0.0375	0.0015
GaN[6]	0	0.0	0.0
	3	-0.1109	0.168
	4	0.0	0.1215
	8	0.0090	0.0
	11	0.04945	0.020

Table 2: Material properties used in calculations.

Material	$a_0/\text{\AA}$	m_e	m_h	ϵ	$R_{ex}/\text{\AA}$	E_g/eV
CdS (ZB)[1]	5.818	0.19	0.80	5.5	38	2.50
CdS (hex)[1]	4.136	0.19	0.80	5.5	38	2.50
GaAs[1]	5.654	0.07	0.68	10.9	182	1.48
GaN (ZB)[6, 11]	4.4953	0.19	0.40	9.3	76	3.20

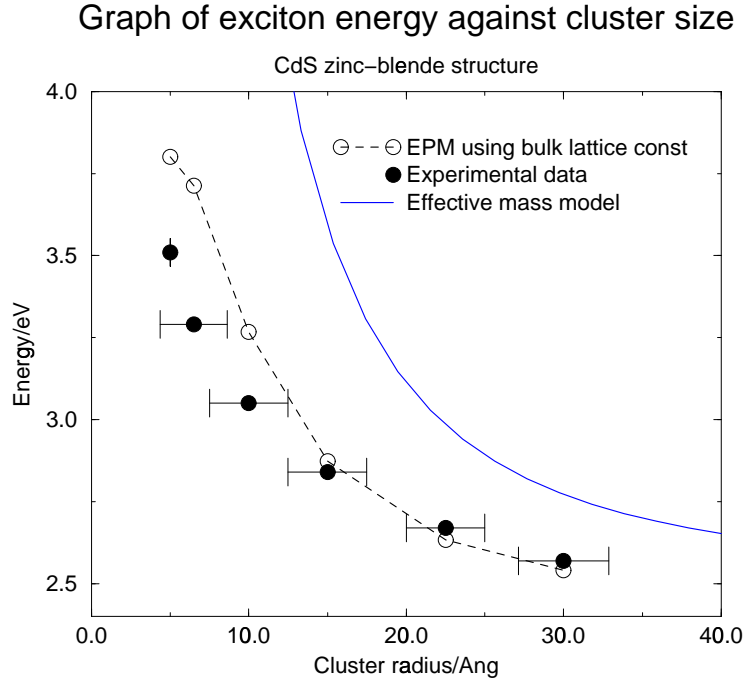


Figure 3: The exciton energies of zinc-blende CdS, calculated using the bulk lattice constant. The experimental data is that of Wang and Herron[2].

Table 3: Percentage lattice contractions computed from Wang and Herron’s data[2].

Cluster radius/Å	$2\theta/^\circ$	Percentage contraction
5.0	28.5	5.60
7.5	27.7	2.80
10.0	27.3	1.44
15.0	28.5	0.00

Bragg equation $n\lambda = 2d \sin \theta$, from which we obtain

$$\frac{a_{02}}{a_{01}} = \frac{\sin \theta_1}{\sin \theta_2} \tag{58}$$

where a_{01} and a_{02} are the lattice parameters in two clusters having (111) diffraction peaks at angles θ_1 and θ_2 respectively. Their data is presented in Table 3, along with the percentage contraction of the lattice constant with respect to the 15 Å cluster, computed using Equation 58.

The results from applying these contractions to a_0 when computing the band gap via EPM are shown in Figure 4. It is observed that the computed exciton energies match the experimental energies very well, with the exception of the 5 Å cluster. This cluster is pyramidal, because of the different synthesis process used in its creation (this is also the reason for the lack of error bars on the cluster size: these clusters are essentially monodisperse)[2]. In this case, the model of the system as a sphere clearly no longer applies. Also, the structure of the cluster is unknown, so the process of using the Bragg equation to obtain the lattice constant may not be valid, since it is not certain that the 5 Å cluster has the same structure as the 15 Å cluster. However, we can see that if we extrapolated the curve from the clusters which are known to have this structure, we would obtain a better prediction of the exciton energy, as Rama Krishna and Friesner[1, fig. 2(b)] showed. By simple volume arguments it may be shown that the 5 Å crystal contains around 13 CdS molecules, and hence is probably better modelled by some *ab initio* method.

6.1.2 The effect of structure

The effect of crystal structure on the exciton energies was investigated by repeating the process described above for the hexagonal form of CdS. The pseudopotentials from Table 1 were used to calculate the band structure shown in Figure 5. The calculated band gap in the bulk solid is 2.47 eV, compared to the experimental value of 2.50 eV, so 0.03 eV was added to the computed cluster band gaps. Because of the absence of any experimental data for hexagonal CdS clusters, the percentage lattice parameter contractions from Table 3 were used.

The results, shown in Figure 6, are a poorer match to the experimental data than those calculated for the zinc-blende CdS structure, indicating that these clusters have the zinc-blende structure. This is confirmed by experiment[2]. Also, though the band gap (and hence the exciton energy) in the bulk solid is not sensitive to structure, these results show that in the clusters this is not the case.

Graph of exciton energy against cluster size

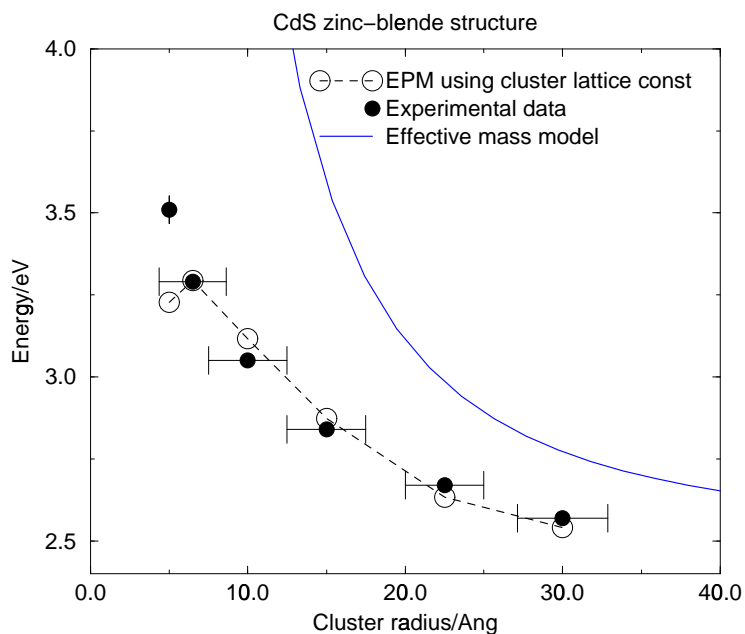


Figure 4: The exciton energies of zinc-blende CdS, calculated using the experimentally determined bulk lattice constant. The experimental data is that of Wang and Herron[2].

Calculated bandstructure for Hexagonal CdS

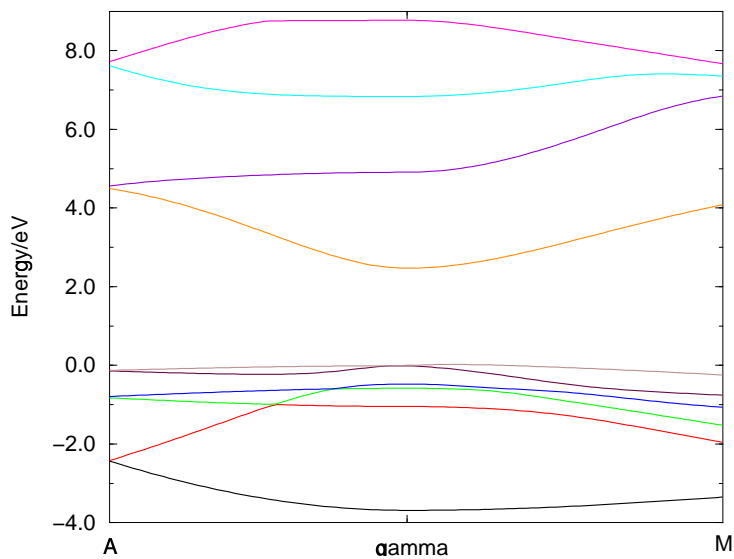


Figure 5: The band structure of hexagonal CdS near the top of the valence band.

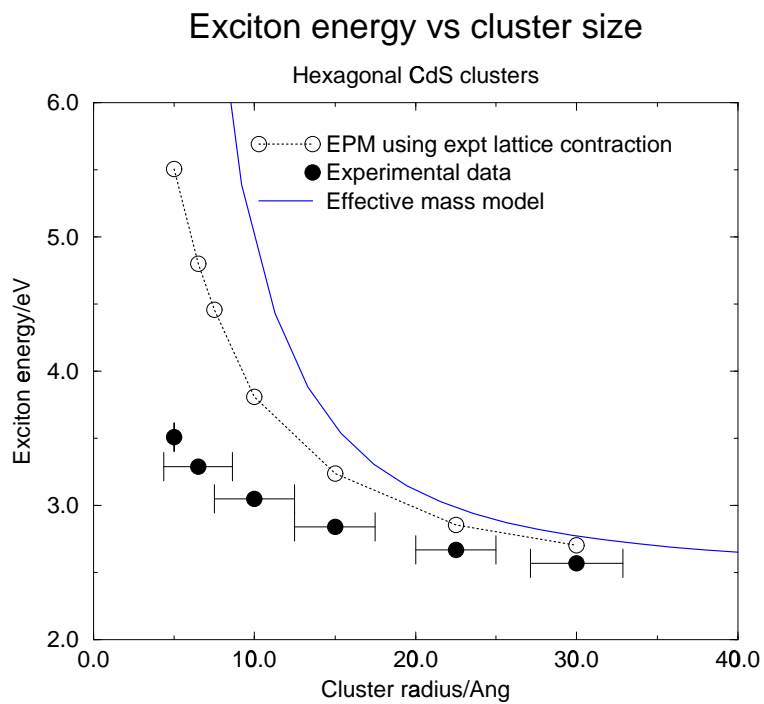


Figure 6: The exciton energies of hexagonal CdS, calculated using the lattice contractions obtained from experimental data. The experimental exciton energies are those of Wang and Herron[2].

Variation of exciton energies with cluster shape

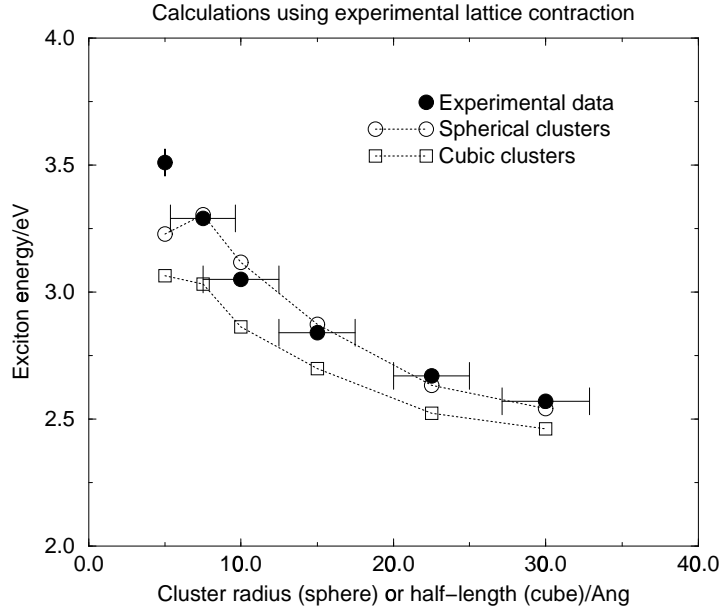


Figure 7: The exciton energies of zinc-blende CdS, calculated assuming square and spherical clusters. The experimental data exciton energies are those of Wang and Herron[2].

6.1.3 The effect of shape

The exciton energies in a cubic cluster of zinc-blende CdS of side $L = 2R$ were calculated, again using the lattice contraction factors given in Table 3. The results are shown in Figure 7, along with those for a spherical cluster. Clearly, there is a shape effect present in the clusters, even when lattice constant is that of the bulk lattice. The better match to experimental data obtained from assuming a spherical cluster confirms the observation that Wang and Herron’s clusters are roughly spherical.

6.2 Results for GaAs clusters

The EPM with the pseudopotentials for GaAs from Table 1 was used to calculate the band structure shown in Figure 8. The calculated band gap in the bulk solid was 1.50 eV, compared to the experimental value of 1.48 eV. The exciton energies for clusters of GaAs were calculated using a similar method to that of Section 6.1.1. The bulk lattice constant was used to calculate the exciton energies as there is no experimental data available on lattice constants in GaAs clusters. The calculated band gap in the cluster was corrected by 0.02 eV to allow for the overestimate of the bulk solid band gap.

The results of these calculations are shown in Figure 9. A red-shift in the exciton energy is observed for sufficiently small clusters. The origin of the red-shift can be seen in Figure 8: the difference in energy between the band at the top of the valence band and the band at the bottom of the conduction band first increases but then decreases as \mathbf{k}_{min} moves from Γ to L. Since this energy difference is the band gap in the cluster, we can see from Equation 6 that for

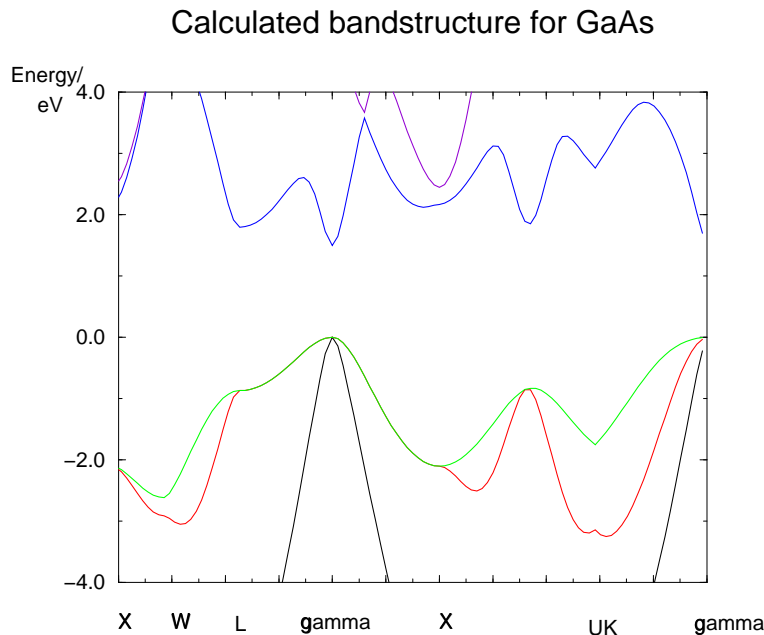


Figure 8: The calculated band structure for GaAs near the top of the valence band.

sufficiently small R , \mathbf{k}_{min} will pass through the position of local maximum at the bottom of the valence band, roughly mid-way between Γ and L. This result arises only from the EPM calculation: in the EMM the bands are assumed to be parabolic so the maximum does not occur.

There is a single experimental datum for GaAs clusters, due to Olshavsky *et. al.*[3], which gives the exciton energy in a 12 Å cluster as 2.52 eV. As Figure 9 shows, the calculated energy is in good agreement with this, however, this datum does not confirm the red-shift as the cluster is not small enough to be in the region which is expected to experience red-shift. This awaits further experimental work, although difficulties with the synthesis of GaAs clusters have yet to be overcome.

6.3 Results for GaN clusters

The EPM, with the pseudopotentials for GaN from Table 1, was used to calculate the band structure shown in Figure 10. The calculated band gap in the bulk solid was 3.25 eV, compared to the experimental value of 3.20 eV. The exciton energies for clusters of GaN were calculated using a similar method to that of Section 6.1.1. The bulk lattice constant was used to calculate the exciton energies as there is no experimental data available on lattice constants in GaN clusters. The calculated band gap in the cluster was corrected by 0.05 eV to allow for the overestimate of the bulk solid band gap.

The results of the exciton energy calculation are shown in Figure 11. I have been unable to find any experimental data with which to compare these predictions. They are also presented numerically in Table 4, since they may be the first such predictions for GaN clusters.

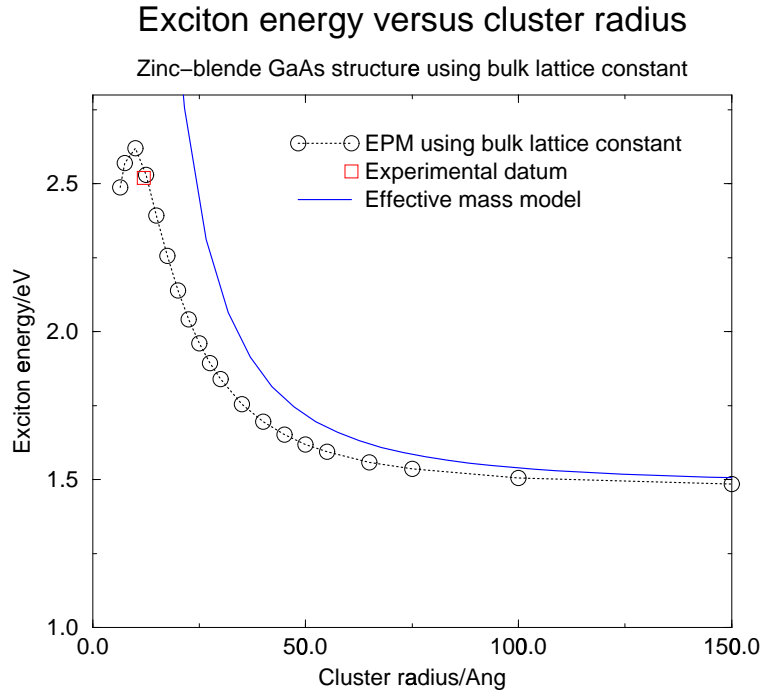


Figure 9: The calculated exciton energies for spherical GaAs clusters. The experimental datum is from Olshavsky[3].

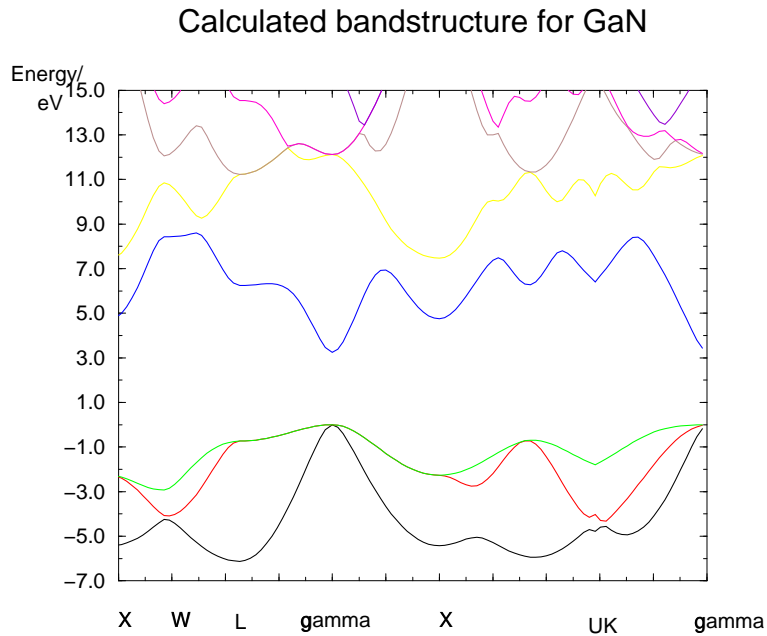


Figure 10: The calculated band structure for GaN near the top of the valence band.

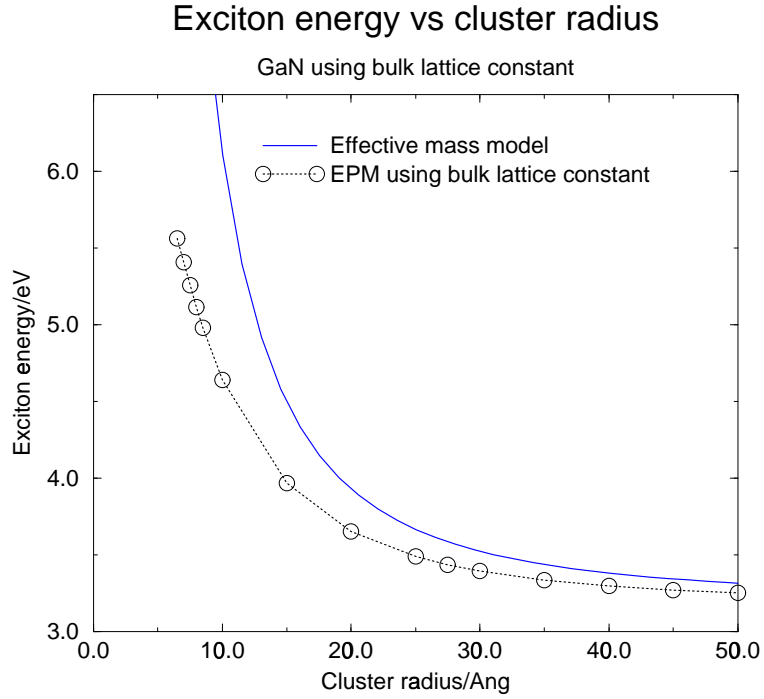


Figure 11: Calculated exciton energies for spherical GaN clusters.

Table 4: Calculated exciton energies for spherical GaN clusters.

$R/\text{\AA}$	E_g/eV	E_{ex}/eV
6.5	5.99	5.56
7.0	5.80	5.40
7.5	5.63	5.25
8.0	5.46	5.11
8.5	5.31	4.98
10.0	4.92	4.64
15.0	4.15	3.96
20.0	3.80	3.65
25.0	3.60	3.49
27.5	3.54	3.43
30.0	3.49	3.39
35.0	3.42	3.33
40.0	3.37	3.30
45.0	3.34	3.27
50.0	3.31	3.25
55.0	3.29	3.24
65.0	3.27	3.22
75.0	3.25	3.21
100.0	3.23	3.20
150.0	3.21	3.19

7 Conclusion

The EMM for exciton energies is not quantitatively accurate for small cluster sizes, as can be seen from comparison of the EMM predictions with experimental data and with the prediction of the EPM. As well as numerical inaccuracy, the EMM fails qualitatively to predict the red-shift for small cluster sizes in GaAs.

The results of using the EPM as a predictive method for exciton energies are:

- In CdS, the exciton energies are sensitive to the size, shape and crystal structure of the cluster.
- The predicted exciton energies for CdS are in good agreement with experiment.
- The first predictions of cluster exciton energies for GaN are made.
- In GaAs, there is a red-shift in the absorption spectra with decreasing cluster size, for small enough clusters. For CdS and GaN this shift does not occur as they lack the combination of a stationary point at the bottom of the conduction band and almost flat band at the top of the valence band which produces this effect in GaAs.

By way of application, it has been shown that by varying cluster size it is possible to produce optical properties in clusters of semiconductors which vary widely from the bulk properties, and which vary according to crystal structure and shape of cluster.

8 Acknowledgements

The author wishes to thank Dr G Rajagopal for his advice on this project, for the use of the TCM computing facilities, and for his provision of some Fortran code to compute reciprocal lattice vectors (in Section A.2.1 the routines to which this applies are marked with comments). Thanks are also due to Paul Bolchover for the lending of a book on L^AT_EX 2_ε, which enabled me to typeset this paper.¹

¹ This paper was typeset in L^AT_EX 2_ε, with graphs produced using `xmgr` and figures produced using `xfig`.

References

- [1] M. V. Rama Krishna and R. A. Friesner, *J. Chem. Phys.* **95**, 8309 (1991).
- [2] Y. Wang and N. Herron, *Phys. Rev. B* **42**, 7253 (1990).
- [3] M. A. Olshavsky, A. N. Goldstein, and A. P. Alivistos, *J. Am. Chem. Soc.* **112**, 9438 (1990).
- [4] Y. Kayanuma, *Solid State Comm.* **59**, 405 (1986).
- [5] Y. Kayanuma, *Phys. Rev. B* **38**, 9797 (1988).
- [6] H. Aourag, B. Bouhafs, and M. Certier, *Phys. Stat. Sol. (B)* **201**, 117 (1997).
- [7] S. Gasiorowicz, *Quantum Physics*, 2nd ed. (Wiley, New York, 1996).
- [8] C. Kittel, *Introduction to Solid State Physics*, 6th ed. (Wiley, New York, 1986).
- [9] L. E. Brus, *J. Chem. Phys.* **80**, 4403 (1984).
- [10] L. E. Brus, *J. Phys. Chem.* **90**, 2555 (1986).
- [11] D. C. Reynolds *et. al.*, *J. Appl. Phys.* **80**, 594 (1996).
- [12] *Concise Index of Crystal Lattice Structures*,
<http://cst-www.nrl.navy.mil/lattice/struk.vect/b4.gif>

A Appendix: Computational matters

A.1 Overview of the workings of the programs

Equation 46 provides a matrix element for calculation of the electronic energies. Two Fortran-77 programs were used to calculate these energies.

Both programs share a set of core routines for diagonalizing the Hamiltonian matrix. The programs generate a list of \mathbf{G} vectors from a user input file containing the primitive lattice translation vectors for the structure. The generated list contains all the vectors in reciprocal space up to a user specified value of $|\mathbf{G}|$. Using Equation 46, the appropriate structure factors for the lattice, and a user supplied list of the empirical potentials V_S and V_A , the program produces a matrix containing one element for each pair of vectors in this list, and diagonalizes the matrix using the NAG library routine F02HAF.

To compute a bulk crystal band structure, the first program (Section A.2.5) diagonalizes the matrix repeatedly for positions \mathbf{k} along lines joining these high symmetry points. A dummy index x is used to produce the correct placing of these positions on a two dimensional graph. The resulting bands are written out to files with the data in (x, E) pairs, with one file per band.

To compute band gaps in clusters, the second program (Section A.2.6) uses the quantisation conditions given Equation 6 or Equation 49 (according to whether the user specified spherical or square clusters in the input file) to find the value of \mathbf{k}_{min} . The matrix is diagonalized for \mathbf{k}_{min} and the band gap at this value of \mathbf{k} is determined by subtracting the energy of the band at the bottom of the valence band from the energy of the band at the top of the conduction band (these will be two successive eigenvalues in a sorted list of the eigenvalues of the matrix). From this band gap the program computes E_{ex} . The program loops over values of the cluster size R and writes data to a file in (R, E_g, E_{ex}) triplets.

A.2 Source code

The source code, makefile and a sample input file are presented below. They are also in `~pw201/project/` on the TCM machines.

A.2.1 sethamil.F

This program contains the routines which are common to both band structure and exciton energy calculations, for example those routines for reading information about the material from the user, for calculating reciprocal lattice vectors and for producing and diagonalizing the Hamiltonian matrix.

```
c      These are subroutines which are common to everything which uses
c      the EPM method.

c      *****
c      ** Latt:
c      ** Adapted from a routine provided by Dr Rajagopal          **
c      **                                                         **
c      ** given the real space lattice translation vectors, a1,a2,a3, **
c      ** this subroutine calculates the reciprocal space translation **
c      ** vectors, b1,b2,b3, the volume of the unit cell, and the   **
c      ** inner products of real and reciprocal space translation   **
c      ** vectors.                                                 **
c      **                                                         **
c      ** input :                                                 **
```

```

c      **                                     **
c      ** a1,a2,a3 ..... real space translation vectors of the **
c      ** supercell                                     **
c      **                                     **
c      ** b1,b2,b3 ..... reciprocal space translation vectors. **
c      ** defined so that a1.b1 = 1 etc                 **
c      **                                     **
c      ** volume ..... volume of the unit cell         **
c      **                                     **
c      ** b11,b22,b33 ... b1.b1  b2.b2  b3.b3         **
c      ** b12,b13,b23 ... b1.b2  b1.b3  b2.b3         **
c      **                                     **
c      ** a11,a22,a33 ... a1.a1  a2.a2  a3.a3         **
c      ** a12,a13,a23 ... a1.a2  a1.a3  a2.a3         **
c      **                                     **
c      ** >> important << the vectors b1, b2, b3 do not have the **
c      ** factor of 2*pi (or sqrt(2)*pi)               **
c      ** included in their                             **
c      ** definition! these will be included where     **
c      ** needed.                                       **
c      ****
c
c      subroutine latt()
c
c      implicit none
c
c      Parameters
c
c      Includes
c
c      #include "lattice.inc"
c
c      Variables
c
c      double precision volume
c      integer i
c
c      ****
c
c      calculate volume of the super cell :
c
c      volume = a1(1) * a2(2) * a3(3) + a1(2) * a2(3) * a3(1) +
&      a1(3) * a2(1) * a3(2) - a1(3) * a2(2) * a3(1) -
&      a1(1) * a2(3) * a3(2) - a1(2) * a2(1) * a3(3)
c
c      reciprocal space translation vectors :
c
c      b1(1) = ( a2(2) * a3(3) - a2(3) * a3(2) ) / volume
c      b1(2) = ( a2(3) * a3(1) - a2(1) * a3(3) ) / volume
c      b1(3) = ( a2(1) * a3(2) - a3(1) * a2(2) ) / volume
c      b2(1) = ( a3(2) * a1(3) - a1(2) * a3(3) ) / volume
c      b2(2) = ( a1(1) * a3(3) - a3(1) * a1(3) ) / volume
c      b2(3) = ( a3(1) * a1(2) - a1(1) * a3(2) ) / volume
c      b3(1) = ( a1(2) * a2(3) - a2(2) * a1(3) ) / volume
c      b3(2) = ( a2(1) * a1(3) - a1(1) * a2(3) ) / volume
c      b3(3) = ( a1(1) * a2(2) - a1(2) * a2(1) ) / volume
c
c      volume = abs(volume)
c
c      Note: the paper defines the reciprocal lattice vectors
c      in units of sqrt(2)*pi/a so we need to convert from the
c      conventional calculation of the vectors (in units of 2*pi which
c      is fine for zinc-blende as it's what we want) to the one the
c      paper uses for the hexagonal system.
c      if ( crystal_type .eq. 3 ) then
c          do i = 1, 3

```

```

        b1(i) = b1(i) * sqrt(2.0d0)
        b2(i) = b2(i) * sqrt(2.0d0)
        b3(i) = b3(i) * sqrt(2.0d0)
    enddo
endif

c    inner products of real space translation vectors :

a11 = a1(1) * a1(1) + a1(2) * a1(2) + a1(3) * a1(3)
a22 = a2(1) * a2(1) + a2(2) * a2(2) + a2(3) * a2(3)
a33 = a3(1) * a3(1) + a3(2) * a3(2) + a3(3) * a3(3)
a12 = a1(1) * a2(1) + a1(2) * a2(2) + a1(3) * a2(3)
a13 = a1(1) * a3(1) + a1(2) * a3(2) + a1(3) * a3(3)
a23 = a2(1) * a3(1) + a2(2) * a3(2) + a2(3) * a3(3)

c    inner products of reciprocal space translation vectors :

b11 = b1(1) * b1(1) + b1(2) * b1(2) + b1(3) * b1(3)
b22 = b2(1) * b2(1) + b2(2) * b2(2) + b2(3) * b2(3)
b33 = b3(1) * b3(1) + b3(2) * b3(2) + b3(3) * b3(3)
b12 = b1(1) * b2(1) + b1(2) * b2(2) + b1(3) * b2(3)
b13 = b1(1) * b3(1) + b1(2) * b3(2) + b1(3) * b3(3)
b23 = b2(1) * b3(1) + b2(2) * b3(2) + b2(3) * b3(3)

end

c    *****
c    **  ewlatt
c    **  Adapted from a routine provided by Dr Rajagopal          **
c    **
c    **  calculates the multiples ng1,ng2,ng3 of the reciprocal   **
c    **  lattice vectors b1,b2,b3 which define a parallelopiped  **
c    **  which completely encloses a sphere of radius sq gsqmax.  **
c    **
c    **  input
c    **
c    **  b1,2,3 ... the elementary reciprocal lattice vectors,    **
c    **           in units of reciprocal length determined by     **
c    **           a1,a2,a3
c    **
c    **  gsqmax ... squared length of the longest g-vector to be  **
c    **           considered (in the same units as b1,2,3 squared) **
c    **
c    **  output
c    **
c    **  ngtot ... total number of g-vectors within sphere
c    **
c    **  ng1,2,3 .. limits of reciprocal space in the directions  **
c    **           b1,2,3 (the summation will go over -ng1 to +ng1, **
c    **           etc.)
c    **  *****

subroutine ewlat()

implicit none

c    Includes:

#include "lattice.inc"

c    Variables:

double precision gsq , eps , two
parameter (eps=1.0d-8)
parameter ( two = 2.0d0 )

```

```

integer nng1 , nng2 , nng3
integer i1 , i2 , i3

c *****

c this choice always encloses a sphere of radius squared gsqmax

ng1 = int( sqrt( gsqmax * a11 ) + eps )
ng2 = int( sqrt( gsqmax * a22 ) + eps )
ng3 = int( sqrt( gsqmax * a33 ) + eps )

c try and reduce ng1,2,3

nng1 = 0
nng2 = 0
nng3 = 0
ngtot = 0

do i1 = -ng1 , ng1

  do i2 = -ng2 , ng2

    do i3 = -ng3 , ng3

      gsq = dble(i1 * i1) * b11 + dble(i2 * i2) * b22 +
&         dble(i3 * i3) * b33 + two *
&         ( dble(i1 * i2) * b12 + dble(i1 * i3) * b13 +
&         dble(i2 * i3) * b23 )

      if ( gsq .lt. ( gsqmax + eps ) ) then

        if ( abs(i1) .gt. nng1 ) nng1 = abs(i1)

        if ( abs(i2) .gt. nng2 ) nng2 = abs(i2)

        if ( abs(i3) .gt. nng3 ) nng3 = abs(i3)

        ngtot = ngtot + 1

      end if

    end do

  end do

end do

ng1 = nng1
ng2 = nng2
ng3 = nng3

end

c *****
c ** gshells **
c ** Adapted from a routine provided by Dr Rajagopal **
c ** this subroutine sets up the g-vectors that are needed in the **
c ** g-space part of the routines. these are the reciprocal **
c ** lattice vectors, ( i.e. integral multiples of the vectors **
c ** b1, b2, b3 ). **
c ** ** **
c ** ng1,2,3 => limits of the reciprocal space sum, in the **
c ** directions b1,2,3. ( the summation will go over **
c ** -ng1,2,3 to +ng1,2,3. **
c ** ** **

```

```

c  ** gsqmax => squared length of the longest g-vectors to be      **
c  **          considered. ( same units as b11,... ).              **
c  **          **                                                **
c  ** mvec   => maximum # of g-vectors allowed.                  **
c  **          **                                                **
c  ** ngvec0 => total # of g-vectors within the sphere of radius  **
c  **          squared gsqmax, in reciprocal space.              **
c  **          **                                                **
c  ** ngvec  => the # of g-vectors within the above-mentioned    **
c  **          sphere, such that only one of the g-vectors in    **
c  **          the pair (g1*b1+g2*b2+g3*b3 ; -g1*b1-g2*b2-g3*b3) **
c  **          is chosen. it will have a weight of 2 in          **
c  **          subsequent calculations provided that g1 is not   **
c  **          equal to 0. if it is, then its weight is 1.       **
c  **          **                                                **
c  *****

      subroutine gshells()

      implicit none

c    Includes:

#include "lattice.inc"

c    Variables:

      integer igvec, j, g1, g2, g3, igsq, ierr
      double precision gsq, two , zero , gsqvec,
&          gx , gy , gz

      parameter (two = 2.0d0 )
      parameter ( zero = 0.0d0 )

c    *****

      open(unit=41,file='list_of_g_vectors',status='unknown')

      ngvec0 = 0
c    Create the list of G vectors now we know the maximum multiple
c    of b1 b2 and b3 we're allowed (obtained from ewlat above)

      do g1 = -ng1 , ng1

         do g2 = -ng2 , ng2

            do g3 = -ng3 , ng3

               gsq = dble(g1 * g1) * b11 + dble(g2 * g2) * b22 +
&                  dble(g3 * g3) * b33 + two *
&                  (dble(g1 * g2) * b12 + dble(g1 * g3) * b13 +
&                  dble(g2 * g3) * b23)

               if (gsq .lt. gsqmax ) then

                  ngvec0 = ngvec0 + 1

                  if (ngvec0 .gt. mvec) then

                     go to 999

                  else

c    lisvec contains the length of the vector, and glist contains
c    its componets in the b1,b2,b3 basis
                     glist(1,ngvec0) = g1

```



```

c      Write them to a file for inspection
      write(41,1001) gx , gy , gz , gsqvec
1001  format(4(F18.8))
      lisvec(ngvec) = gsqvec
      end do
      lisvec(ngvec+1) = zero
      ngshell = 0
      do igvec = 1 , ngvec
        if( abs( lisvec(igvec+1) - lisvec(igvec) ) .gt. 1.0d-3 ) then
          ngshell = ngshell + 1
          gshell(ngshell) = igvec
        endif
      enddo
      close(unit=41)
      return
999  continue
      stop 'mvec is too small : change mvec '
      end

c      *****
c      ** Read in information about the lattice
c      ** Adapted from a routine provided by Dr Rajagopal          **
c      *****

      subroutine start()

      implicit none

c      Includes:

      #include "lattice.inc"
      #include "potential.inc"
      #include "constants.inc"

c      Variables:

      integer lat , i

c      *****

c      read in the values of gsqmax :

      read (5,fmt= * ) gsqmax

c      read in crystal type:
c      1=diamond
c      2=zinc blende
c      3=wurtzite

      read (5,fmt= *) crystal_type

```

```

c   read lattice constant ( in angstroms ) :
      read (5,fmt= * ) alat

c   convert lattice constant, alat to atomic units :
      alat = alat / agtoboehr

c   read dielectric constant for this material
      read (5,fmt= *) dielectric

c   read exciton bohr radius
      read (5,fmt =*) r_exciton

c   read electron and hole masses
      read (5,fmt =*) masse
      read (5,fmt =*) massh

c   read primitive lattice translation vectors :
      read (5,fmt= * ) a1
      read (5,fmt= * ) a2
      read (5,fmt= * ) a3

c   subroutine latt calculates the reciprocal lattice vectors,
c   (b1,b2,b3), squared lengths (a11,b11 etc.) and dot products,
c   (a12,b12 etc. ), for the primitive unit cell.
      call latt()

c   read cluster size and convert to atomic units
      read (5, fmt=* ) dimension
      dimension = dimension / agtoboehr

c   read what sort of quantisation we'd like
c   0=none
c   1=square cluster
c   2=spherical cluster
      read (5, fmt=*) quantisation

c   Where is the top of the valence band
      read (5, fmt=*) bandtop

c   Read in the pseudo-potential factors
      read (5,fmt=*) numfactors

      if ( numfactors .gt. maxnumfactors ) then
        stop 'Increase maxnumfactors and recompile'
      endif

      do i=1, numfactors
        read(5,fmt=*) modsq(i),v(1,i),v(2,i)
      enddo

      end

c   *****

```

```

c    ** Echo the information to the screen.
c    ** Adapted from a routine provided by Dr Rajagopal
c    ****
c
c    subroutine echo()
c
c    implicit none
c
c    Includes:
c
c    #include "lattice.inc"
c    #include "potential.inc"
c
c    Variables:
c
c    integer i
c
c    ****
c
c    write(6,*)
c    write(6,*)
c    write(6,'( 20x,' program to determine bandstructure  '))
c    write(6,'( 20x,' -----  '))
c    write(6,'( 20x,'  '))
c    write(6,'( 20x,' via emperical pseudopotential method  '))
c    write(6,'( 20x,' -----  '))
c    write(6,*)
c    write(6,*)
c
c    write(6,'( 2x,' parameters set in main program  '))
c    write(6,'( 2x,' -----  '))
c    write(6,*)
c    write(6,'( 2x,' alat    => lattice constant  '))
c    write(6,*)
c    write(6,'( 2x,' alat    =  ',f16.8  )') alat
c    write(6,*)
c    write(6,'( 2x,' a1,2,3 => lattice translation vectors  '))
c    write(6,'( 2x,'          of primitive cell  '))
c    write(6,*)
c    write(6,'( 2x,' a1    =  ',3f16.8  )') a1
c    write(6,'( 2x,' a2    =  ',3f16.8  )') a2
c    write(6,'( 2x,' a3    =  ',3f16.8  )') a3
c    write(6,*)
c    write(6,'( 2x,' paj    => dot products  '))
c    write(6,*)
c    write(6,'( 2x,' a11   =  ',f16.8  )') a11
c    write(6,'( 2x,' a22   =  ',f16.8  )') a22
c    write(6,'( 2x,' a33   =  ',f16.8  )') a33
c    write(6,'( 2x,' a12   =  ',f16.8  )') a12
c    write(6,'( 2x,' a13   =  ',f16.8  )') a13
c    write(6,'( 2x,' a23   =  ',f16.8  )') a23
c    write(6,*)
c    write(6,'( 2x,' pvolume => volume of supercell  '))
c    write(6,'( 2x,' pvolume =  ',f16.8  )') pvolume
c    write(6,*)
c    write(6,'( 2x,' b1,2,3 => reciprocal lattice vectors  '))
c    write(6,'( 2x,'          of primitive cell  '))
c    write(6,*)
c    write(6,'( 2x,' b1    =  ',3f16.8  )') b1
c    write(6,'( 2x,' b2    =  ',3f16.8  )') b2
c    write(6,'( 2x,' b3    =  ',3f16.8  )') b3
c    write(6,*)
c    write(6,'( 2x,' pbij   => dot products  '))
c    write(6,*)
c    write(6,'( 2x,' b11   =  ',f16.8  )') b11
c    write(6,'( 2x,' b22   =  ',f16.8  )') b22

```

```

write(6,'( 2x,' b33      =  ',f16.8      )') b33
write(6,'( 2x,' b12      =  ',f16.8      )') b12
write(6,'( 2x,' b13      =  ',f16.8      )') b13
write(6,'( 2x,' b23      =  ',f16.8      )') b23
write(6,*)
write(6,*)
write(6,*)

      write (6,fmt= * )
      write (6,fmt='( 2x,' ngtot =  ',i8 )') ngtot
      write (6,fmt='( 2x,' ng1  =  ',i8 )') ng1
      write (6,fmt='( 2x,' ng2  =  ',i8 )') ng2
      write (6,fmt='( 2x,' ng3  =  ',i8 )') ng3
      write (6,fmt= * )
      write (6,fmt='( 2x,' gsqmax =  ',f10.5 )') gsqmax

write(6,*)
write(6,'( 2x,' maxg      => max # of g-space vectors  ')')
write(6,*)
write(6,'( 2x,' maxg      =  ',i8      )') maxg
write(6,*)
write(6,*) ' total # of g-space vectors is      : ',ngvec0
write(6,*) ' actual # of g-vectors needed is    : ',ngvec
write(6,*) ' # of shells of g-space vectors is : ',ngshell
write(6,*) ' cluster size is ', dimension

if (quantisation .eq. 0 ) then
  write(6,*) ' no quantisation in k space '
elseif ( quantisation .eq. 1 ) then
  write(6,*) ' square cluster '
elseif ( quantisation .eq. 2 ) then
  write(6,*) ' sphere cluster '
endif

write(6,*) ' # of pseudopotential factors is : ',numfactors
write(6,*) ' modsq      symmetric antisymmetric'

do i=1, numfactors
  write(6,fmt='(3f10.5)') modsq(i), v(1,i), v(2,i)
enddo

write(6,*) 'dielectric is ', dielectric
write(6,*) 'electron mass ', masse
write(6,*) 'hole mass      ', massh

end

c *****
c ** Subtract two vectors
c *****

subroutine dvecsub( a, b, result )

implicit none

c Parameters:

doubleprecision a(3)      ! [i] First vector
doubleprecision b(3)      ! [i] Second vector
doubleprecision result(3) ! [o] First - Second

```

```

c   Includes:
c   Variables:
      integer i
c   *****

      do i=1,3
        result(i) = a(i) - b(i)
      enddo

999  continue

      return

      end

c   *****

c   *****
c   ** Compute the modulus of a vector
c   *****

      subroutine dvecmodsq( v, length )

      implicit none

c   Parameters:

      doubleprecision v(3)      ! [i] The vector
      doubleprecision length    ! [o] its length squared

c   Includes:
c   Variables:
c   *****

      length = (v(1) * v(1)) + (v(2) * v(2)) + (v(3) * v(3))

999  continue

      return

      end

c   *****
c   ** Scale a vector by a given amount
c   *****

      subroutine dvecscale( vec, scale, out )

      implicit none

c   Parameters:

      doubleprecision vec(3)    ! [i] Input vector
      doubleprecision scale     ! [i] Scale
      doubleprecision out(3)    ! [o] The scaled vector

c   Includes:
c   Variables:
c   *****

```

```

        out(1) = vec(1) * scale
        out(2) = vec(2) * scale
        out(3) = vec(3) * scale

999  continue

        return

        end

c *****

c *****
c ** Add two vectors
c *****

        subroutine dvecadd( vec1, vec2, out )

        implicit none

c Parameters:

        doubleprecision vec1(3)  ! [i] First vector
        doubleprecision vec2(3)  ! [i] Second vector
        doubleprecision out(3)   ! [o] Output

c Includes:

c Variables:

c *****

        out(1) = vec1(1) + vec2(1)
        out(2) = vec1(2) + vec2(2)
        out(3) = vec1(3) + vec2(3)

999  continue

        return

        end

c *****

c *****
c ** Copy one vector to another
c *****

        subroutine dveccopy( in, out)

        implicit none

c Parameters:

        doubleprecision in(3)  ! [i]
        doubleprecision out(3) ! [o]

c Includes:

c Variables:

c *****

```

```

out(1) = in(1)
out(2) = in(2)
out(3) = in(3)

999 continue

return

end

c *****
c *****
c ** Set up the hamiltonian matrix
c *****
c subroutine setup_hamiltonian(gvector,maxg,ngvec,
&                               crystal_type, complex_hamil)

implicit none

c Parameters:

doubleprecision gvector(3,*)      ! [i] array of allowed g-vectors
integer          maxg              ! [i] max poss no of g vectors
integer          ngvec            ! [i] actual no of g vectors
integer          crystal_type      ! [i]
complex*16       complex_hamil(maxg, maxg) ! [o] The Hamiltonian matrix

c Includes:

c Variables:

complex*16 ctemp1, ctemp2
integer ig, jg
doubleprecision re, im, gminus(3), gmod, temp, pot, ss, sa

integer sym, antisym
sym = 1
antisym = 2

c *****
c Zero out the Hamiltonian matrix

do jg = 1, ngvec
do ig = 1, ngvec
complex_hamil(ig,jg) = (0.0d0, 0.0d0)
end do
enddo

c Hamiltonian for zinc blende and wurtzite are same except
c their structure factors differ. Don't bother with diamond
c for now.

if(crystal_type .eq. 2 .or. crystal_type .eq. 3 ) then

c setup potential energy part

c Hamiltonian has only KE on leading diagonal as we can see from
c the formula that the potentials Va and Vs are both zero for
c  $G - G' = 0$ . So omit diagonal terms.
c Also it's Hermitian matrix so only need upper triangular
c region:

do ig=1, ngvec-1
do jg= ig+1, ngvec

```

```

c      gminus is G - G', the modulus squared of gminus is gmod
c      call dvecsub( gvector(1,ig), gvector(1,jg), gminus )
c      call dvecmodsq( gminus, gmod )

c      This is just the formula for the real and imaginary parts
c      of the matrix element. It calls the functions pot, ss, and sa
c      to get the form and structure factors
c      re= pot(gmod,sym) * ss(gminus, crystal_type)
c      im= pot(gmod,antisym) * sa(gminus, crystal_type)

c      complex_hamil(ig, jg) = dcplx(re, im )

c      enddo
c      enddo

c      end if

c      return

c      end

c      *****

c      *****
c      ** Pseudopotential co-efficients
c      ** Returns the form factors for a given |G|.
c      *****

c      doubleprecision function pot( length, type )

c      implicit none

c      Parameters:

c      doubleprecision length      ! [i] g squared
c      integer              type    ! [i] 1 for sym, 2 for antisym

c      Includes:

c      #include "potential.inc"

c      Variables:

c      integer i
c      logical found

c      *****

c      found = .false.

c      A match is considered to have been found when the value of gmod
c      in the table matches the one given to within 0.01. This is to
c      handle recurring fractions which occur in the hexagonal system:
c      you only need to enter them to that precision to be sure of which
c      one you have.
c      do i = 1, numfactors
c      if ( dabs( modsq(i) - length ) .lt. 0.01d0 ) then
c      pot = v(type,i)
c      found = .true.
c      goto 999
c      end if
c      end do

```

```

999  continue

c  sanity check: if you put the pseudopotential coefficients in
c  so you miss out a length which should have coefficients defined
c  you get told about it:
    if ( .not. found ) then
      if ( length .gt. modsq(1) .and. length .lt. modsq(numfactors) )
& then
        write (*,*) "***ERROR: vector length ", length, " has no match"
        stop
      else
        pot = 0d0
      endif
    endif
  endif

  return

end

c  *****
c  *****
c  ** Symmetric structure factor
c  *****

doubleprecision function ss( g, crystal_type )

implicit none

c  Parameters:

doubleprecision g(3)  ! [i] g vector
integer crystal_type ! [i]

c  Includes:

#include "constants.inc"

c  Variables:

doubleprecision gdot

c  *****

if ( crystal_type .eq. 2 ) then ! we have zinc blende
  ss = cos ( (pi/4.0d0) * ( g(1) + g(2) + g(3) ) )

else if ( crystal_type .eq. 3 ) then ! we have wurtzite

  gdot = g(1)*t1 + g(2)*t2 + g(3)*t3

  ss = cos ( sqrt(2.0d0) * pi * gdot ) * cos ( 2 * pi *
&      u * g(3) / sqrt(3.0d0) )

endif

return

end

c  *****

c  *****
c  ** Antisymmetric structure factor

```

```

c *****
doubleprecision function sa( g, crystal_type )
implicit none
c Parameters:
doubleprecision g(3) ! [i] g vector
integer crystal_type ! [i]
c Includes:
#include "constants.inc"
c Variables:
doubleprecision gdot
c *****
if ( crystal_type .eq. 2 ) then ! we have zinc blende
sa = sin ( (pi/4.0d0) * ( g(1) + g(2) + g(3) ) )
else if ( crystal_type .eq. 3 ) then ! we have wurtzite
gdot = g(1)*t1 + g(2)*t2 + g(3)*t3
sa = cos( sqrt(2.0d0) * pi * gdot ) * sin( 2 * pi *
& u * g(3) / sqrt(3.0d0) )
endif
999 continue
return
end
c *****
c *****
c ** Diagonalise the Hamiltonian and extract the band structure
c *****
subroutine eigenvalues( maxg, H, n, temp_hamil, energies, ierr )
implicit none
c Parameters:
integer maxg ! [i] size of most of the arrays
complex*16 H(maxg,maxg)! [i] Hamiltonian matrix
integer n ! [i] size of matrix
complex*16 temp_hamil(maxg, maxg) ! [i/o] Workspace
doubleprecision energies(maxg) ! [o] The energies ie the e-values
integer ierr ! [o] ierr .ne. 0 => error occurred
c For some unknown reason, I cannot define parameter ( lwork = maxg * 2)
c so use the pre-processor instead.
c Includes:
#define LWORK maxg*2

```

```

c   Variables:

doubleprecision rwork(maxg*3), w(maxg)

complex*16      work(LWORK)

integer i, j

c   *****

ierr = 0

c   The NAG routine overwrites the Hamiltonian it is passed
c   so we need to put it into a temporary array.

do i=1, n
do j=1, n

    temp_hamil(i,j) = H(i,j)

enddo
enddo

c   Use a NAG routine to perform the diagonalisation:

call F02HAF('N','U', n, temp_hamil, maxg, energies, rwork, work,
& LWORK, ierr )

c   and that's magic...

999 continue

return

end

c   *****

```

A.2.2 lattice.inc

```

c   *****
c   * Common blocks for lattice calculations etc
c   *****

integer mvec, mshell, nbasis, crystal_type, ngvec0, nitype,
&      ngtot, maxr, maxg

parameter ( mvec=1000, mshell=50, nbasis=2)
parameter ( maxg=mvec )

integer ng1 , ng2 , ng3,
&      ngvec ,
&      indx(mvec) , glist(3,mvec),
&      gshell(mshell) , icopy(mvec),
&      ngshell, quantisation, bandtop

double precision gvec(maxg) , gvector(3,maxg), kvec(3),
&      vs(mshell), va(mshell),
&      gsqmax ,
&      pvolume ,
&      lisvec(mvec) ,
&      basis(3,nbasis),
&      alat ,
&      a1(3) , a2(3) , a3(3) , b1(3) , b2(3) ,

```

```

&          b3(3), a11 , a22 , a33 , a12 , a13 ,a23,
&          b11 , b22 , b33 , b12 , b13 , b23 , t0 ,
&          gsqgvec(maxg) , gxgvec(maxg) , gygvec(maxg) ,
&          gzgvec(maxg)

doubleprecision dimension, dielectric, r_exciton, masse, massh

common /lattice_i/ ngvec0,
&          nitype, ngtot, maxr, ng1, ng2, ng3,
&          ngvec, indx, glist, gshell,
&          ngshell, quantisation, crystal_type, bandtop

common /lattice_d/ gvec, gvector, kvec,
&          vs, va,
&          gsqmax ,
&          pvolume ,
&          lisvec ,
&          basis,
&          alat ,
&          a1, a2, a3 , b1, b2 ,
&          b3, a11 , a22 , a33 , a12 , a13 ,a23,
&          b11 , b22 , b33 , b12 , b13 , b23 , t0 ,
&          gsqgvec, gxgvec, gygvec,
&          gzgvec, dimension, dielectric, r_exciton,
&          masse, massh
c *****

```

A.2.3 constants.inc

```

c *****
c ** Value for pi and a few other useful things like that
c *****

doubleprecision pi, u, t1, t2, t3, h_to_ev
double precision agtoboehr

parameter ( pi = 3.14159265358979 )
parameter ( h_to_ev = 27.2116 )
parameter ( u = 0.375 )

parameter ( t1 = 1.0d0/sqrt(48.0d0))
parameter ( t2 = 1.0d0/12.0d0)
parameter ( t3 = 1.0d0/sqrt(6.0d0) )

parameter(agtoboehr = 0.52917715d0)

```

A.2.4 potential.inc

```

c *****
c ** Common blocks for the pseudopotential factors for the lattice.
c *****

integer maxnumfactors
parameter ( maxnumfactors = 20 )
integer numfactors

doubleprecision modsq(maxnumfactors),v(2,maxnumfactors)

common /potential_d/ modsq, v
common /potential_i/ numfactors

```

A.2.5 bandcalc.F

This program was used to produce the band structures. It writes each band to a separate file called fort.*n* where *n* is the band index plus 50.

```
c *****
c * Empirical Pseudopotential Method Program
c *
c * Requires input to run it: the best way to provide this is to
c * feed it in from the shell using redirection.
c
c
c Revision history:
c 03/3/1997: Bandstructures working. Move on to exciton energy
c 15/2/1997: Fixed so matrix is not overwritten by diagonalization
c           routine.
c
c 4/2/1997: Fixed typo in potentials. Took out the ditching of
c degenerate eigenvalues. Fixed ss and sa to use double precision
c constants (4.0d0).
c
c *****

program epm

implicit none

#include "lattice.inc"
#include "constants.inc"

c *****

integer i , ierr, k, num_k, j, outunit
complex*16      complex_hamil(maxg,maxg), temp_hamil(maxg, maxg)
doubleprecision energies(maxg), temp, kunits, bandzero
doubleprecision emin, emax
parameter ( emin = -10.0d0 )
parameter ( emax = 10.0d0 )
integer maxk
parameter ( maxk = 500 )
doubleprecision kvecs(3,maxk), kaxis(maxk)

c subroutine start reads in run info etc.

      call start()

c subroutine ewlat gives the max # of unit cells in real space and
c max # of g-vectors in reciprocal space.

      call ewlat()

c subroutine gshells generates a sorted list of g vectors
      call gshells()

c echo information to stdout

      call echo()

c set up hamiltonian matrix
c Note: this does not set up leading diagonal. Off diagonal terms
c have the potential terms. Leading diagonal has kinetic terms only
c since we have  $G - G' = 0$  which gives  $V_a = V_s = 0$ 

      call setup_hamiltonian(gvector,maxg,ngvec,
&                          crystal_type,
&                          complex_hamil)
```

```

c   Set up energy conversion factor. See note below.
   if ( crystal_type .eq. 2 ) then
       kunits = (2*pi/alat)
   else if ( crystal_type .eq. 3 ) then
       kunits = sqrt(2.0d0)*(pi/alat)
   endif

c   We need the energy of the centre of the bandstructure (gamma
c   point) so we can make the zero of the output energy the same as
c   it. Set up the Hamiltonian for k=0

   do i = 1, ngvec
       temp = 0.5d0 * kunits * kunits *
&           ( gvector(1,i) **2
&           +   gvector(2,i) **2
&           +   gvector(3,i) **2 )
       complex_hamil(i,i) = dcmplx(temp,0.0d0)
   end do

c   and get the eigenvalues of it...
   call eigenvalues( maxg, complex_hamil,ngvec,temp_hamil,energies,
&                  ierr )

c   The user has hopefully told us which band is at the very top of
c   the valence band.
   bandzero = energies(bandtop)

c   We also need a list of k vectors to churn though to produce a
c   bandstructure.

   call make_klist( crystal_type, quantisation, dimension, maxk,
&                  kvecs, kaxis, num_k, ierr )

c   Open output file
c   open(unit=42,file='bandstructure',status='unknown')

   do k= 1, num_k

c   setup the diagonal kinetic energy cointribution
c   (NB this is the only part which changes with k, the off diagonal
c   elements are independent of k)

       do i= 1,ngvec

c   Note: internally we have stored k and G vectors in units of
c   2*pi/ a , hence we need to convert these for use in the KE
c   expression by scaling the modulus by 4pi**2 / a**2
c   alat is already in atomic units. So we should obtain energies
c   in atomic units.

           temp = 0.5d0 * kunits * kunits *
&           ( (gvector(1,i) + kvecs(1,k))**2
&           +   (gvector(2,i) + kvecs(2,k))**2
&           +   (gvector(3,i) + kvecs(3,k))**2 )
           complex_hamil(i,i) = dcmplx(temp,0.0d0)

       end do

c   Obtain the eigenvalues of the hamiltonian matrix

       call eigenvalues( maxg,complex_hamil,ngvec,temp_hamil,energies,
&                      ierr )

       if ( ierr .ne. 0 ) then

```

```

        stop 'Something went wrong with diagonalization'
    endif

c   Convert the energies to electron volts and zero them to the
c   centre of the band.

    do j=1, ngvec
        energies(j)= ( energies(j) - bandzero ) * h_to_ev
    enddo

c   Write out energies to file

    outunit=50
    do j=1, ngvec

c   This is a hack to put each band in a separate data file
c   The compiler responds to an attempt to write to a unit n by
c   creating a file called fort.n and writing to that. While the
c   program is running, additional writes to that unit append to it
c   So we can write each eigenvalue to a particular file by
c   incrementing the output unit for each eigenvalue.

        outunit=outunit+1

        if ( energies(j) .gt. emax ) then

c   Since energies are ordered in ascending order,
c   we can break out of the loop
c   once we have gone over the maximum wanted energy

            goto 111

        else if ( energies(j) .gt. emin ) then

c   write(42, 1002 ) kaxis(k), energies(j)
            write(outunit, 1002 ) kaxis(k), energies(j)

        end if
    end do
1002 format( 2(F18.12) )

111 continue

c   End loop over k vector

    enddo

c   close(unit=42)

    end

c   *****
c   ** Create list of k-vectors:
c   ** Note: for the plotting program we require an x-axis index which
c   ** will increase from left to right with the distances between
c   ** points on the graph being their separation in k-space. These
c   ** routines produce a list of k vectors to diagonalize for and
c   ** also the x-axis value at which to plot the energies for that
c   ** k-vector.
c   *****

    subroutine make_klist( crystal_type, quantisation, dimension,
&                          max_nk, vecs, kaxis,
&                          num_k, ierr )

    implicit none

```

```

c   Parameters:

integer crystal_type           ! [i] 1=diamond,2=ZB,3=hex
integer quantisation          ! [i] 0=none,1=square,2=sphere
doubleprecision dimension     ! [i] cluster size
integer max_nk                ! [i] size of the arrays
doubleprecision vecs(3,max_nk) ! [o] List of k-vectors
doubleprecision kaxis(*)      ! [o] The index on the graph
integer num_k                 ! [o] number of k-vecs
integer ierr                  ! [o] exit status

c   Includes:

#include "constants.inc"

c   Variables:

integer max_sym, num_pts
parameter (max_sym = 10)
doubleprecision s(3,max_sym), grid, offset

c   *****

c   This works by setting up a list s of the high symmetry points in
c   the zone. These vary between lattices but the code to generate
c   the list of k vectors doesn't have to.
c   if ( crystal_type .eq. 2 ) then
c   The symmetry points for zinc-blende

c   X
s(1,1) = 1.0d0

c   W
s(1,2) = 1.0d0
s(2,2) = 0.5d0

c   L
s(1,3) = 0.5d0
s(2,3) = 0.5d0
s(3,3) = 0.5d0

c   s(*,4) is the gamma point but array already has zero in it

c   X
s(1,5) = 1.d0

c   UK
s(1,6) = 0
s(2,6) = 0.75
s(3,6) = 0.75

c   s(*,7) is back to the gamma point again, so leave it zeroed.

num_pts = 7

c   This is a starting offset on the dummy x axis which puts
c   the gamma point at 0 on this axis.
offset = - ( 0.5d0 + 1.0d0/sqrt(2.0d0) + sqrt(3.0d0)/2.d0 )

elseif ( crystal_type .eq. 3 ) then
c   the symmetry points for the hexagonal system

c   A: a/c = sqrt(3/8) and so a/c * 1/sqrt(2) is...
s(3,1) = sqrt(3.0d0)/4.0d0

```

```

c   Gamma: just leave everything at zero

c   M
      s(1,3) = sqrt(2.0d0)/3

c   There are three points on our bandstructure graph
      num_pts = 3

c   The offset from the origin is just |A|
      offset = -s(3,1)

      endif

      if ( quantisation .eq. 0 ) then
        call list_noquant( num_pts, s, offset, max_nk, vecs, kaxis,
&          num_k, ierr )
      else

c   Set up appropriate distance between allowed k points
      if ( quantisation .eq. 1 ) then
        grid = pi/ dimension
      elseif ( quantisation .eq. 2 ) then
        grid = pi/ ( sqrt(3.0d0) * dimension )
      endif

c   and call the routine for it (which doesn't actually work but
c   should anyone ever write one...)
      call list_quant( grid, num_pts, s, offset, max_nk, vecs, kaxis,
&          num_k, ierr )
      endif

      return

      end

c   *****
c   ** The un-quantised one (just the solid bandstructure)
c   *****

      subroutine list_noquant( num_pts, s, offset, max_nk, vecs, kaxis,
&          num_k, ierr )

      implicit none

c   Parameters:

      integer num_pts           ! [i] Number of sym pts
      doubleprecision s(3,*)    ! [i] Their positions
      doubleprecision offset    ! [i] Offset to centre gamma pt
      integer max_nk            ! [i] size of the arrays
      doubleprecision vecs(3,max_nk) ! [o] List of k-vectors
      doubleprecision kaxis(*)  ! [o] The index on the graph
      integer num_k             ! [o] Actual number generated
      integer ierr              ! [o] exit status

c   Includes:

c   Variables:

      doubleprecision kspacing
      doubleprecision dir(3), dist, lambda, k(3)
      integer i, j, kpos, nk

c   *****

```

```

c      This is a rough way of getting a reasonable point spacing: saying
c      we want 50 points in the space between gamma and the first
c      symmetry point.
c      call dvecmodsq(s(1,1), kspacing)
c      kspacing = sqrt(kspacing)/50.0d0

c      Set up the list of k vectors

c      kpos = 1

c      Move from one point to the next
c      do i=1, num_pts-1

c      Compute direction vector from one point to next
c      call dvecsub(s(1,i+1), s(1,i), dir )
c      Compute distance from one point to next
c      call dvecmodsq(dir, dist)
c      dist = sqrt(dist)
c      Make direction vector a unit vector
c      call dvecscale( dir, 1/dist, dir)

c      nk = dist/kspacing

c      Vector equation of a line: scale direction by lambda where
c      that's the distance along the line in the direction required
c      Add on position of the point we're going from.

c      do j=0,nk-1

c      lambda = dble(j) * kspacing
c      call dvecscale( dir, lambda, k )
c      call dvecadd( k, s(1,i), vecs(1,kpos) )

c      kaxis(kpos) = offset + lambda

c      kpos = kpos + 1
c      if ( kpos .gt. max_nk ) then
c        stop 'Make max_nk bigger'
c      endif

c      enddo

c      offset = offset + dist

c      enddo

999 continue

c      num_k = kpos - 1

c      return

c      end

c      *****

c      *****
c      ** Generate quantised list of k-vectors
c      *****

c      subroutine list_quant( grid, num_pts, s, max_nk, vecs, kaxis,
c      & num_k, ierr )

c      implicit none

```

```

c   Parameters:

doubleprecision grid           ! [i] distance between k-pts
integer num_pts                ! [i] Number of sym pts
doubleprecision s(3,*)        ! [i] Their positions
doubleprecision offset        ! [i] Offset to centre gamma pt
integer max_nk                 ! [i] size of the arrays
doubleprecision vecs(3,max_nk) ! [o] List of k-vectors
doubleprecision kaxis(*)      ! [o] The index on the graph
integer num_k                  ! [o] Actual number generated
integer ierr                   ! [o] exit status

c   Includes:

c   Variables:

integer ns(3), ne(3), d(3), n(3)
doubleprecision dir(3), dist, lambda, k(3), work(3)
integer i, j, kpos, nk

doubleprecision small
parameter ( small = 1.0d-4 )

c   *****

stop 'Quantised band structure not supported yet.'

c   *****

```

A.2.6 exciton.F

This program was used to produce the graphs of exciton energy against cluster size.

```

c   *****
c   * Empirical Pseudopotential Method Program
c   * This one calculates exciton energies
c   * Requires input to run it: the best way to provide this is to
c   * feed it in from the shell using redirection.
c   *****

program exciton

implicit none

#include "lattice.inc"
#include "constants.inc"

c   *****

integer i , ierr, k, num_k, j, outunit
complex*16      complex_hamil(maxg,maxg), temp_hamil(maxg, maxg)
doubleprecision energies(maxg), temp, kunits, bandgap, kvecs(3)
doubleprecision gamma, rydberg, exciton_e
integer maxk, sizes
parameter ( maxk = 500 )
parameter ( sizes = 20 )
doubleprecision cluster_size(sizes), reduce(sizes)

c   subroutine start reads in run info etc.

      call start()

c   subroutine ewlat gives the max # of unit cells in real space and
c   max # of g-vectors in reciprocal space.

```

```

        call ewlat()

c   subroutine gshells generates a sorted list of G vectors

        call gshells()

c   echo information to stdout

        call echo()

c   set up hamiltonian matrix
c   Note: this does not set up leading diagonal. Off diagonal terms
c   have the potential terms. Leading diagonal has kinetic terms only
c   since we have  $G - G' = 0$  which gives  $V_a = V_s = 0$ 

        call setup_hamiltonian(gvector,maxg,ngvec,
&                               crystal_type,
&                               complex_hamil)

c   Work out effective rydberg energy in atomic units

        rydberg = (1/( 1/masse + 1/massh)) / (2 * dielectric**2 )

c   Open output file
        open(unit=42,file='exciton_energies',status='unknown')

c   What I did to vary the points and the
c   correction factor in the band structure was change the code
c   and recompile. Since it takes next to no time to recompile
c   this (since the main EPM routines are compiled separately and
c   linked to), its easier than having a huge datafile specifying
c   the material parameters and which points to plot etc. This
c   listing is the setup for GaN.

c   Set up sizes of clusters
        cluster_size(1) = 6.5d0
        cluster_size(2) = 7.0d0
        cluster_size(3) = 7.5d0
        cluster_size(4) = 8.0d0
        cluster_size(5) = 8.5d0
        cluster_size(6) = 10.0d0
        cluster_size(7) = 15.0d0
        cluster_size(8) = 20.0d0
        cluster_size(9) = 25.0d0
        cluster_size(10) = 27.5d0
        cluster_size(11) = 30.0d0
        cluster_size(12) = 35.0d0
        cluster_size(13) = 40.0d0
        cluster_size(14) = 45.0d0
        cluster_size(15) = 50.0d0
        cluster_size(16) = 55.0d0
        cluster_size(17) = 65.0d0
        cluster_size(18) = 75.0d0
        cluster_size(19) = 100.0d0
        cluster_size(20) = 150.0d0

c   Reduce lattice constant from bulk value
c   We use the values given in the paper by Wang and Herron,
c   which seem to differ slightly from those used by the paper
c   by Rama Krishna. Didn't apply to GaN but used for CdS
        reduce(1) = 0.0550d0
        reduce(2) = 0.0418d0
        reduce(3) = 0.0283d0

```

```

c   reduce(4) = 0.0144d0

c   Loop over values of cluster size

       do k=1,sizes

c   Work out cluster size in atomic units
       dimension = cluster_size(k) / agtoboehr

c   Set up conversion factor for converting reciprocal lattice 1/length
c   in internal units to atomic unit 1/lengths. See below
       if ( crystal_type .eq. 2 ) then
           kunits = (2*pi/(alat*(1-reduce(k))))
       else if ( crystal_type .eq. 3 ) then
           kunits = sqrt(2.0d0)*(pi/(alat*(1-reduce(k))))
       endif

c   Set up k-vector of gamma point since we're assuming that's where
c   the bandgap is for now.

       if ( quantisation .eq. 0 ) then
c   No cluster
           gamma = 0
       else if ( quantisation .eq. 1 ) then
c   Square cluster
           gamma = (pi/ (2*dimension)) / kunits
       else if ( quantisation .eq. 2 ) then
c   Spherical cluster
           gamma = ( pi / ( sqrt(3.0d0) * dimension ) ) /kunits
       endif

       do i=1,3
           kvecs(i)=gamma
       enddo

c   setup the diagonal kinetic energy cointribution

       do i= 1,ngvec

c   Note: internally we have stored k and G vectors in units of
c   2*pi/ a , hence we need to convert these for use in the KE
c   expression by scaling the modulus by 4pi**2 / a**2
c   alat is already in atomic units. So we should obtain energies
c   in atomic units.

           temp = 0.5d0 * kunits * kunits *
&               ( (gvector(1,i) + kvecs(1))**2
&               + (gvector(2,i) + kvecs(2))**2
&               + (gvector(3,i) + kvecs(3))**2 )
           complex_hamil(i,i) = dcmplx(temp,0.0d0)

       end do

c   Obtain the eigenvalues of the hamiltonian matrix

       call eigenvalues( maxg,complex_hamil,ngvec,temp_hamil,energies,
&                       ierr )

       if ( ierr .ne. 0 ) then
           stop 'Something went wrong with diagonalization'
       endif

c   EPM overestimates bulk sold band gap so we add a bit
c   to it for the clusters. This again got changed on the fly
c   for different materials and then recompiled.

```

```

        bandgap = energies(bandtop+1)-energies(bandtop) - 0.05d0/h_to_ev

c      Compute the exciton energy from the bandgap
        exciton_e = bandgap - 1.786d0/(dielectric*dimension)
&          - 0.248d0*rydberg

c      Write the size, bandgap and exciton energy to a file.
        write(42, 1002 ) cluster_size(k), bandgap* h_to_ev,
&          exciton_e * h_to_ev

1002 format( 3(F18.12) )

c      End loop over cluster sizes
        enddo

        close(unit=42)

        end

```

A.2.7 makefile

This is the makefile that was used to build the programs `bandcalc.exe` and `exciton.exe`.

```

FFLAGS = -g

exciton.exe : exciton.o sethamil.o
$(FC) $(FFLAGS) -lnag exciton.o sethamil.o -o exciton.exe

exciton.o : exciton.F lattice.inc constants.inc
$(FC) $(FFLAGS) -c exciton.F -o exciton.o

bandcalc.exe : bandcalc.o sethamil.o
$(FC) $(FFLAGS) -lnag bandcalc.o sethamil.o -o bandcalc.exe

bandcalc.o : bandcalc.F lattice.inc constants.inc
$(FC) $(FFLAGS) -c bandcalc.F -o bandcalc.o

sethamil.o : sethamil.F lattice.inc constants.inc potential.inc
$(FC) $(FFLAGS) -c sethamil.F -o sethamil.o

```

A.2.8 Example input file

This is an example input file, which was passed to the programs using `exciton.exe < inputfile`, for example.

```

27.00          ! gsqmax
2              ! xtal type 1=diamond,2=zinc-blende,3=wurtzite
5.818         ! lattice constant
5.5           ! dielectric constant
30            ! exciton Bohr radius
0.19         ! electron mass
0.80         ! hole mass
0.0,0.5,0.5  ! primitive lattice translation vectors
0.5,0.0,0.5
0.5,0.5,0.0
15 ! Cluster size
0 ! quantisation type 0=none,1=square,2=sphere
4 ! index of top of valence band
6 ! number of pseudopotential factors, followed by gsquared,vs,va
0.0,0.0,0.0
3.0,-0.12,0.115
4.0,0.0,0.065
8.0,0.015,0.0
11.0,0.020,0.025
12.0,0.0,0.025

```