A study in the computation time required for the inclusion of strain field effects in Bloch-wave simulations of TEM diffraction contrast images

B. J. Dulong[a], R. D. Haynes[b], M. D. Robertson[a]*

[a]*Dept. of Physics, Acadia University, Wolfville, NS, Canada B4P 2R6*
[b]*Dept. of Mathematics and Statistics, Acadia University, Wolfville, NS, Canada B4P 2R6*

May 4, 2007

Abstract

As transmission electron microscopy (TEM) imaging techniques continue to become more quantitative, interpretation of the experimental images demands that accurate image simulations be computed incorporating all important aspects of the image including: compositional, crystallographic and microscope effects, as well as contrast due to strain fields arising from stresses created by lattice misfit or defects. Incorporation of the effects of strain fields in the simulation of diffraction-contrast TEM images in the Bloch-wave formalism requires the integration of a system of first-order differential equations in order to modify the excitation amplitudes and produce contrast in the image. This integration is computationally demanding with the time for integration scaling as the cube of the number of beams included in the calculation. In order to investigate the computational requirements of the integration, a variety of numerical integration packages was evaluated with respect to timing and accuracy in the simulation of quantum dot, spherical inclusion and screw dislocation images. It was determined

*Corresponding author. Tel.: +1 902 585 1318; fax: +1 902 585-1816.
E–mail address: michael.robertson@acadiau.ca.

that a class of Adams-multistep methods can provide a decrease in computation time ranging from 2-4 as compared to the standard Runge–Kutta(4)5 approach depending on the simulation conditions.

PACS: 61.14.Dc; 61.14.Dc; 02.60.Jh; 81.07.Bc

Keywords: Transmission electron microscopy; Computer modeling and simulation; Strain field; Numerical integration; Diffraction contrast

1 Introduction

In order to extract quantitative information from diffraction contrast images in a transmission electron microscope (TEM) accurate simulation of the experimentally obtained images is often required. The two main approaches to TEM image simulation are the Bloch–wave [1–3] and multislice [4–6] formalisms, and in the case of images containing strain fields at low to medium magnifications, the Bloch–wave approach is usually the method of choice. Applications where the Bloch-wave approach have been applied to the quantitative analysis of strain fields in diffraction contrast images include: the classic work by Ashby and Brown on spherically symmetrical coherency strains and inclusions [7, 8], individual and interacting dislocations [9], coherent spheroidal particles [10], crack–shaped cavities in Si [11], antiphase boundaries in GaP [12] and InAs quantum dots in InP [13].

The intensity of a diffracted beam as calculated using the Bloch–wave approach is given by

$$I_g(z) = \left| \sum_j \alpha^{(j)}(z) C_g^{(j)} exp(2\pi i k_z^{(j)} z) \right|^2 \tag{1}$$

where z is the thickness of the crystal, k_z^j is the wavevector of the electron inside the crystal for the j^{th} Bloch wave, $C_g^{(j)}$ are the Bloch–wave coefficients, and $\alpha^{(j)}(z)$ is the excitation amplitude of the j^{th} Bloch wave. The incorporation of strain effects in the image intensities is accomplished by integrating the excitation amplitudes over the displacement field, $\mathbf{R}(z)$, in the direction of the electron beam and requires the solution of the following set of coupled first–order differential equations [1],

$$\begin{aligned}\frac{d\alpha^{(j)}(z)}{dz} &= 2\pi i \sum_i \alpha^{(i)}(z) \exp[2\pi i(k_z^{(i)} - k_z^{(j)})z] \sum_g C_g^{*(j)} C_g^{(i)} \frac{d}{dz} (\mathbf{g} \cdot \mathbf{R}(z)) \\ &\quad - 2\pi q^{(j)} \alpha^{(j)}(z).\end{aligned} \tag{2}$$

In these equations $q^{(j)}$ is the imaginary part of the wavevector for the j^{th} Bloch wave and $\mathbf{g}$ is the reciprocal lattice vector. For any non–trivial displacement field, the solution to the above set of equations cannot be determined analytically and numerical approaches must be used.

The time required to calculate image intensities in the Bloch-wave approach is largely governed by two factors: the matrix diagonalization required to determine $k_z^{(j)}$ and $C_g^{(i)}$, and the integration of equation (2). The amount of computation time for the matrix diagonalization step scales as n^2 where n is the number of beams included in the calculation and efficient linear algebra codes exist to perform this operation [15]. Similarly, the integration of the excitation amplitudes is an order n^3 process (cf. Section 4) and the time required for the calculation of TEM image intensities is dominated by this step, especially for large n.

The integration of the system of equations given in (2) has usually been accomplished by Runge–Kutta methods [10–13]. In particular, the fourth–order Runge–Kutta 4(5) method is a common implementation of the technique offering very good numerical stability as well as allowing the user the ability to specify a maximum error tolerance per step (EPS). However, Runge–Kutta 4(5) was developed as a reliable, general–purpose integration package and may or may not be the most time efficient technique for solving equation (2). The reason why computation time becomes important is the need for quantitative simulations that accurately portray the experimental system under investigation. Depending on the particular crystal orientation being modeled, a large number of reflections could be excited by the electron beam and hence the simulation of even a small 128×128 pixel image could take many hours, days or weeks. For example, Fig. 1 is a plot of the bright-field image intensity for 50 nm of unstrained InP crystal at the [001] zone–axis orientation including and excluding the effects of absorption. The intensity in both cases initially increases to a maximum at 49 beams and then monotonically decreases as the number of beams is increased beyond this point. This effect was caused by the effective extinction distance decreasing with an increase in the number of beams included in the calculation. When the calculation was performed with only 9 beams, the specimen thickness required to observe a maximum in the first bright thickness fringe (apart from the maximum at 0 nm) was about 60 nm which decreased to about 50 nm for 49 beams and 44 nm for 381 beams. Since the specimen thickness where a maximum in the thickness fringe intensity transitioned from above to below the target thickness of 50 nm due to reduced effective extinction distance values, the simulated intensity goes through a maximum as the number of beams is increased. It is observed that over 100 beams are required in order for the image intensity to converge to a stable value which can lead to very long calculation times. As a means of comparison, even if the $n = 100$ calculation took only one second to compute without strain for a single point, a 128×128 pixel image including the effects of strain might require over 450 hours to compute using a modern personal computer.

Since the time required for integration of the excitation amplitudes can dominate the total computation time for images containing strain for even small n, the objective of this research was to investigate a variety of integration schemes in order to determine the fastest methods for a given accuracy. Although computer processors continue to increase in speed and parallel computing methods can greatly reduce the time required to simulate images, efficiencies in numerical algorithms should continue to be sought to maximize the scale of simulation that can be performed in a reasonable amount of time with a given technology. In the following sections, we will (1) discuss the three strain fields analyzed, (2) provide a brief survey of the numerical approaches used,)3) examine the mathematical properties of equation (2), (4) give a discussion of the timing results and (4) provide a recommendation for the numerical integration approach most suited to the three cases investigated.

2 Simulation Method

The various integration routines were tested on three model strain fields. First, the 3.2% lattice misfit strain contrast arising from an InAs quantum dot in an InP matrix was studied. The dot was 25 nm in diameter, 2.0 nm in height and there was a 0.5 nm thick InAs wetting layer underneath the QD as shown in Fig. 2. The QD was located in the middle of a 50 nm $\times$ 50 nm $\times$ 50 nm cube of InP and the anisotropic elastic fields were solved by finite element methods. Different anisotropic elastic parameters were used for InAs and InP [16] and the displacement fields were output to a discrete $128 \times 128 \times 401$ point array. Each column of 401 points in the z direction was fit to a spline curve in order that the displacement fields, or their derivatives, could be calculated at any location within the specimen. Complete details of the QD system and finite-element method can be found in Robertson et al. [13].

The second model strain field evaluated was a 25 nm spherical InAs inclusion located within the centre of a 50 nm $\times$ 50 nm $\times$ 50 nm InP matrix. The displacement field was evaluated analytically in the isotropic approximation using [7, 17]

$$\begin{aligned} \mathbf{R} &= C\mathbf{r} \qquad \mathbf{r} \leq \mathbf{r_0}, \\ \mathbf{R} &= C\frac{r_0^3}{r^3}\mathbf{r} \quad \mathbf{r} \geq \mathbf{r_0}, \end{aligned} \tag{3}$$

where r_0 is the radius of the spherical inclusion and C is a parameter dependent upon the isotropic elastic constants. This model was chosen so that the numerical methods could be tested on an analytic displacement field in addition to the discretely calculated elastic field of the first example.

The last strain field considered was that due to a screw dislocation in InP using isotropic elasticity theory. Dislocations can have large displacement fields near their core and demand that any integration routine used to solve equation (2) be able to accommodate widely varying magnitudes of the displacement fields. The screw dislocation was located midway through the thickness of the crystal and the displacement field was expressed as [17, 18]

$$\mathbf{R} = \frac{\mathbf{b}}{2\pi} \tan^{-1}\left(\frac{z - z_d}{x}\right), \tag{4}$$

where $b = 1/2[110]$ is the Burgers vector of the screw dislocation, z is the depth in the crystal, z_d is the depth of the dislocation (i.e. 25 nm) in the 50 nm $\times$ 50 nm $\times$ 50 nm InP matrix, and x is the radial distance from the core of the dislocation.

All of the Bloch-wave simulations included the effects of absorption through the inclusion of a complex term in the atomic form factor as per Bird and King [19] and Bird [20]. The Debye-Waller factors were those given by Reid [21] for a temperature of 300 K. All of the simulations, except where otherwise indicated, were performed at the [001] zone-axis orientation and the thickness of the specimen was 50 nm.

The Bloch-wave image simulation code was written by the authors except for (1) subroutine ATOM used to calculate the atomic form factors [19], (2) the matrix diagonalization and inverse routines as supplied by the *LAPACK* project [15] and (3) the integration routines used were obtained from either *Numerical Recipes* [22] or the *Netlib* repository [23]. It should be noted that this implementation of the Bloch-wave theory has used the standard assumptions and approximations including (1) the column approximation, (2) the finite number-of-beams approximation, (3) the high-energy approximation, (4) the deformable-ion approximation, (5) elastic-scattering approximation and (6) the phenomenological treatment of absorption by means of a complex atomic potential. The Bloch-wave theory and the application of these approximations have been rigorously reported in the literature and will not be repeated here [1–3, 14]. The simulations were performed using a single node of a 50–node Beowulf cluster where each node contained a 64–bit AMD Operton processor operating at a clock speed of 1.4 GHz with 4 GB of RAM.

Two different timing evaluations were performed. The main timing trials involved the simulation of a 128×128 pixel image where the pixels were decoupled from one another using the column approximation. When it was desired to observe the behaviour of a particular integration routine as a function of distance in the specimen, z, a calculation was performed on a single column located at the edge of the QD where the elastic displacement fields in the x and y directions are equal and of the greatest magnitude as shown in Fig. 3.

3 Numerical Approaches

Presented in this section is a brief description of the numerical approaches studied for the integration of the excitation amplitudes given in equation (2). A complete overview of numerical methods for the solution of ordinary differential equations is beyond the scope of this study and the reader is directed to two general references for more detailed information [24, 25]. In the numerical integration literature, the independent integration parameter is usually referred to as time whereas in electron microscopy the integration is performed over distance, z.

3.1 Definitions

Equation (2) is a system of first–order coupled differential equations that can be written more generally as

$$y' = f(z, y), \tag{5}$$

and all of the numerical solvers investigated herein have been designed to solve equations of this form. As a point of nomenclature, numerical integration routines are often referred to as either explicit or implicit and the integrations defined as possessing either stiff or non-stiff properties. In explicit integration routines, the value of the integral at a position n, y_n, only depends on the previous values of y, i.e. y_{n-1}, y_{n-2}, etc. For example, the simple forward Euler method is given by

$$y_n = y_{n-1} + hf(z_{n-1}, y_{n-1}), \tag{6}$$

where h is the size of the step and $z_n = z_{n-1} + h$. Hence, the forward Euler method is an explicit solver since the current value of the parameter depends only on data from the previous points. Conversely, the backward Euler method is an example of an implicit solver since the current value depends on a function of the current value itself as shown below

$$y_n = y_{n-1} + hf(z_n, y_n). \tag{7}$$

Stiffness is a term more difficult to define. It may be thought of as a phenomenon which may emerge as one numerically solves systems of differential equations. Several factors combine to determine if a problem presents itself as being stiff including (1) features of the differential equation itself, (2) the user specified error tolerance and (3) the length of the integration interval. In physical models stiffness may occur when a system models interrelated phenomena which proceed on very different time scales [26]. A problem is considered stiff if the step size of an explicit method is restricted to small values in order to maintain stability of the calculation when accuracy requirements alone would allow for larger step sizes. Explicit methods are easily evaluated at each step, however, when solving a stiff problem the increased number of steps required by an explicit method can outweigh the total cost associated with an implicit method. In fact, a functional definition of stiffness is that an implicit method solves a stiff problem more efficiently than an explicit method. In addition, evidence for stiffness may also be provided by the eigenvalues of the Jacobian of the right hand side function $f(t, y)$ (cf. [27]). The problem is stiff if

$$h \cdot \min_j(Re(\lambda_j)) \ll -1, \tag{8}$$

where λ_j denote the eigenvalues of the Jacobian.

3.2 Runge Kutta

In general, a Runge–Kutta method to solve a system of differential equations (5) can be written as

$$\mathbf{y}_n = \mathbf{y}_{n-1} + h\sum_{i=1}^{s} b_i \mathbf{f}(z_{n-1} + c_i h, \mathbf{Y}_i), \tag{9}$$

where s is the number of intermediate stages, $\mathbf{y}_{n-1}$ and $\mathbf{y}_n$ denote approximate solutions at positions z_{n-1} and $z_n = z_{n-1} + h$, respectively. $\mathbf{Y}_i$ is a local intermediate solution evaluated at position $z_{n-1} + c_i h$ and is given by

$$\mathbf{Y}_i = \mathbf{y}_{n-1} + h\sum_{j=1}^{s} a_{ij} \mathbf{f}(z_{n-1} + c_j h, \mathbf{Y}_j), \qquad 1 \leq i \leq s. \tag{10}$$

Runge–Kutta methods are commonly referred to as one–step methods since the solution at position z_n is found only from the solution at the previous step z_{n-1}.

Runge–Kutta routines can be written in a convenient shorthand notation

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

known as a Butcher array or tableau. For example, the Butcher array

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

denotes the popular forward Euler method given by equation (6). Under relatively mild assumptions on the vector valued function f, it can be shown that the error in the forward Euler approximation satisfies

$$|y(z_n) - y_n| \leq Ch \quad \text{as} \quad h \to 0.$$

We say the forward Euler approximation is a first order method. Generally, a method is order p if the error is bounded by Ch^p for some constant C as $h \to 0$.

One can develop higher order methods by increasing the number of intermediate stages, s, and choosing appropriate coefficients in the Butcher array. For instance the Cash-Karp tableau

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & \\
\frac{3}{5} & \frac{3}{10} & \frac{-9}{10} & \frac{6}{5} & & & \\
1 & \frac{-11}{54} & \frac{5}{2} & \frac{-70}{27} & \frac{35}{27} & & \\
\frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} & \frac{575}{13824} & \frac{44275}{110592} & \frac{253}{4096} & \\
\hline
 & \frac{37}{378} & 0 & \frac{250}{621} & \frac{125}{594} & 0 & \frac{512}{1771} \\
\hline
 & \frac{2825}{27648} & 0 & \frac{18575}{48384} & \frac{13525}{55296} & \frac{277}{14336} & \frac{1}{4}
\end{array}$$

defines a particularly efficient embedded Runge–Kutta 4(5) pair. These parameters generate both a fourth and fifth order approximation using a total of only six function evaluations. The difference between the two approximations allows for an efficient estimation of the method's error per step and an adaptive step size selection procedure.

Runge Kutta schemes have successfully been used to solve equation (2) in [11–13, 28] and thus provides the benchmark for our simulations.

3.3 Bulirsch-Stöer

A popular approach to obtain high accuracy solutions to ordinary differential equations is the extrapolation technique due to Bulirsch and Stöer [29]. Extrapolation methods (cf. [25]) provide another way to obtain high order one–step methods and have been successfully applied to the solution of equation (2) in [10]. This class of methods apply a relatively low–order one–step Runge–Kutta method to a problem using a fixed sequence of sub–steps. Certain combinations of these approximations can be combined to obtain approximate solutions of increasing order. The Bulirsch-Stöer method makes repeated use of the Runge–Kutta midpoint method with tableau

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ \hline & 0 & 1 \end{array} .$$

Essentially, the solution is propagated from z_n to $z_{n+1} = z_n + h$ by using certain combinations of n_j modified midpoint steps of size $h_j = h/n_j$ where the n_js are chosen from some fixed list. Bulirsch and Stöer originally suggested $n_j \in \{2, 4, 6, 8, 12, 16, \ldots\}$ while later Deuflhard [30] proposed a more efficient sequence of approximations based on $n_j \in \{2, 4, 6, 8, 10, 14, \ldots\}$. The result is a sequence $y_{n+1}^{n_j}$ of approximations to y_{n+1}. These approximations are combined using either polynomial or rational extrapolation to obtain successively higher order approximations until the user supplied error tolerance is met.

3.4 Linear Multistep

The methods of the previous two sections are classified as one–step methods. That is, the solution at position z_n is determined only using the approximation obtained at position z_{n-1}. Intermediate solution approximations, obtained via additional function evaluations, are used to obtain higher order approximations. Linear multistep methods use an alternate approach where the solution and/or derivative approximations from previous step positions are used to generate a high–order approximation, often without any additional function evaluations.

The general class of k–step linear multistep methods may be written as

$$y_n = \sum_{j=1}^{k} \alpha_j y_{n-j} + h_n \sum_{j=0}^{k} \beta_j f(z_{n-j}, y_{n-j}). \tag{11}$$

Linear multistep methods are typically classified as either Adams methods, choosing $\alpha_1 = 1, \alpha_j = 0$ $(j = 2, \ldots, k)$, or Backward Differentiation Formulae (BDF methods) which satisfy $\beta_0 \neq 0, \beta_j = 0$ $(j = 1, \ldots, k)$. Since $\beta_0 \neq 0$ the BDF methods are necessarily implicit as y_n appears on both sides of (11). Implicit schemes require the additional cost of a nonlinear system solve at each step and the nonlinear equations are solved by either a fixed point iteration or some variant of Newton's method.

Adams methods are either explicit (if $\beta_0 = 0$) or implicit (if $\beta_0 \neq 0$). Explicit Adams methods are referred to as Adams–Bashforth methods while implicit Adams methods are known as Adams–Moulton methods. Although explicit schemes are less costly per step, implicit formulations have the advantage of allowing much larger steps for certain classes of problems [31].

Often explicit methods are used in combination with implicit Adams methods as a predictor–corrector pair (PECE). The explicit formula is used to provide an approximation to y_n for use on the right–hand side of the implicit scheme. The implicit scheme is used to correct the predictor provided by the explicit method.

3.5 Software

Table 1 summarizes the methods and particular codes compared in this work. Specifically, the table provides the subroutine name, the type of method as classified above, the library in which the subroutine may be found, and if the code is appropriate for stiff or non–stiff problems.

The Runge–Kutta routines from *Numerical Recipes* [22] and from *Netlib*, RKSUITE [23], provided the benchmark for our simulations. In order to test the Bulirsch–Stöer approach we used the routine BSSTEP from *Numerical Recipes* [22]. The remainder of the codes identified in Table 1 provided various multi–step methods and were obtained from the *ode*, *slatec* and *odepack* packages contained within the *Netlib* repository [23].

One of the inherent difficulties with multi-step methods is the handling of variable step–size implementations. The goal of modern ordinary differential equation solvers is to compute the solution over the distance interval of interest as efficiently as possible while ensuring the solution is accurate within the user supplied error tolerance. In order to achieve this, the step size, $h_n = z_n - z_{n-1}$, is allowed to vary. The step, h_n, is automatically chosen small in regions where the solution is particularly difficult and large when the solution is relatively easy to track.

The *slatec* [23] and *odepack* [23] libraries provided multistep methods via the codes sdriv [26] and SLSODE [31], respectively. Both allowed the choice of explicit or implicit solvers through the specification of user supplied parameters (MINT and MF). These libraries have different capabilities, for example, they offered different implementations of variable stepping strategies. Details of these implementations may be found in the references above and in [32].

The *slatec* libraries provided a version of sdriv, cdriv, which was particularly suited to handle problems involving complex data types. This was useful for the system of differential equations defined in (2). Most ordinary differential equation solvers required real valued data and hence the system needed to be recast in real form by writing each unknown function $\alpha^{(j)}(z)$ as $\alpha_R^{(j)}(z)+i\alpha_I^{(j)}(z)$ where the $\alpha_R^{(j)}$ and $\alpha_I^{(j)}$ were real valued functions. This effectively doubled the dimension of the system and may be an issue for implicit methods which require linear solves at each step involving the Jacobian matrix. Complex arithmetic, however, is often more costly (per operation) than real arithmetic[1]. It is difficult to say a priori if routines which allow complex data types will out–perform real valued solution methods.

4 Examination of the System of Ordinary Differential Equations

Computationally, the most expensive component in a single step of a numerical integration method for systems of ordinary differential equations are the evaluations of the right–hand side function of (5), $f(t, y)$. Each function evaluation for integration of the excitation amplitudes requires an evaluation of the right hand side of equation (2), and hence, the evaluation of three nested summations. Thus, the total cost of the function call scales as the cube of the number of diffracted beams included in the calculation. For example, for the integration

[1]The relative cost will depend highly on computer hardware and compiler optimizations.

located at the single point highlighted in Fig. 2 and only 11 beams, function calls to evaluate equation (2) accounted for approximately 90% of the total computation time for the Runge-Kutta 4(5) solver. Increasing the number of beams to 21 resulted in approximately 98% of the total computation time being spent performing function evaluations. Therefore, any numerical solver that minimizes the total number of function calls compared to the other solvers has a significant advantage in terms of the total calculation time required to form the image.

Another important aspect investigated with this system of ordinary differential equations was whether they displayed any degree of stiffness during the calculation. For example, if the presence of stiffness is dependent on the magnitude of the elastic displacements then a solver that can switch between stiff and non-stiff integration strategies might be more efficient than a single solver that has been configured to solve either stiff or non-stiff problems. As discussed in Section 3.1 and expressed mathematically by equation (8), a problem is considered stiff if the minimum eigenvalue of the Jacobian of equation (2) is much less than -1. Fig. 4 is a plot of the imaginary vs. real components of the eigenvalues of the Jacobian matrix for an integration at a single point for three locations along the integration. The calculation was performed along the [111] zone-axis orientation for 31 beams. The left and right plots correspond to the InP matrix above and below the quantum dot, respectively, and the centre plot was obtained at the centre of the quantum dot. In addition, a plot of the derivative of the elastic displacement field is provided to show more clearly the dependence of the eigenvalues on the elastic fields. Recall that it is the derivative of the elastic displacement field with respect to the direction of the electron propagation, z, that scales the changes in excitation amplitude in equation (2). It is observed that the minimum real eigenvalue is only about -0.002 which, considering the length of integration, does not satisfy the requirement for stiffness. However, one interesting phenomena that occurred was the expansion of the eigenvalues along the imaginary axis suggesting that the solutions become more oscillatory in regions where the gradient in the displacement field was large. As a further confirmation of the lack of stiffness present in equation (2), the exercise was repeated for larger numbers of beams and greater elastic strains. It was found that the distribution in eigenvalues did not change significantly when up to 91 beams were included in the calculation or when the lattice misfit between the quantum dot and the matrix was increased by 4 times up to 13.2%.

The presence of stiffness can also be observed if stiff solvers can perform the integration faster than non-stiff solvers and Table 2 compares the CPU time required to solve equation (2) using stiff and non–stiff solvers with various user supplied tolerances. The simulation conditions were the same as those discussed in the previous paragraph except for the choice of solvers. All of the timing values quoted were an average of 10 runs. The writing of the code and its compilation were optimized for speed and the reported results were obtained within the specified tolerance. The local error within each code was kept within the specified tolerance by adjusting the step size, h. Although there are standard approaches for estimating the local error, the implementation selected by the writer of the code can vary between codes which makes comparing the computation time required to achieve a set tolerance difficult for different codes. However, the local error control for two codes, cdriv and lsode, was calculated in a consistent manner for both the stiff and non-stiff settings of the code and the timing results for these two solvers are provided in Table 2. The simulations were performed for a single point at the location highlighted in Fig. 3 for 21 beams at the [001] zone-axis orientation. At every tolerance value, the non-stiff solver was able to perform the integration in about 75% of the time of the stiff solvers indicating that the non–stiff methods solve the integration problem more efficiently.

It should be noted that the problem did not display any stiffness for any of the conditions investigated, implying that non-stiff solvers, which are more efficient computationally, can be safely used.

5 Timing Results and Discussion

In this section the computation time of a 128×128 pixel image using one–step and multistep methods, as well as stiff and non–stiff solvers is compared. Full image simulations were obtained at the [001] zone–axis orientation for a truncated InAs quantum dot in InP, a spherical InAs inclusion in InP, and a screw dislocation in InP as discussed in Section 2. All simulations were performed with 21 beams over a range of error tolerances. It is recognized that many factors can influence the computation time required to generate a simulated image including (1) the error tolerance, (2) the number of diffracted beams included in the calculation and (3) the magnitude of the gradients of the elastic displacement field. Variations of these 3 parameters were considered. A short selection of results, representative of the large dataset generated by this parametric study, is presented here.

In order to ensure that the timing results could be compared at equivalent levels of accuracy, a reference image was computed. This was necessitated since different integration routines perform the local error control in different ways and relying on equivalent EPS values for each routine would not be possible. However, an absolute comparison can be made by comparing all of the simulations to a single reference image. The reference image was obtained by setting the user supplied error tolerance for the Runge–Kutta (4)5 integration routine close to the lowest value permitted for single-precision arithmetic, 2×10^{-8}. The computed images were then compared to this reference image by assigning a numerical value which represented the *Distance* between the images. *Distance* between images is calculated as the sum, over all pixels in the image, of the magnitude of the difference between a given image's pixel intensity and the corresponding pixel intensity of the reference image, i.e.

$$Distance = \sum_{j=1}^{\#pixels} \left| I_j - I_j^{ref} \right| , \tag{12}$$

where j is the index of pixels in the image,and I_j and I_j^{ref} denote the intensity of the j^{th} pixel in the simulated and reference images, respectively. Another way to define *Distance* is as the modulus of the difference between the simulated and reference image. A plot of the computation time as a function of *Distance* gives the required time for each numerical method to obtain an approximate image within a specified accuracy of the reference image.

The concept of *Distance* can be illustrated by comparing simulation quality as a function of the magnitude of the *Distance* value. Fig. 5 presents a series of quantum dot simulations calculated using the Runge–Kutta(4)5 method for nine different *Distance* values ranging from 1 to 225000. The images have been individually scaled so that black represents an intensity of 0 and white an intensity of 255. Visually, the image quality appears to have converged by Fig. 5e with a *Distance* value of 380. This indicates that qualitatively acceptable images can be obtained at larger *Distance* values, and hence less computation time, than the one derived from the image necessary for quantitative work.

Figures 6–8 show the computation time as a function of *Distance* for the quantum dot, spherical inclusion and screw dislocation, respectively. It is observed that the standard Runge–Kutta technique was not the fastest nor slowest technique, but performed about average when compared to all of the other techniques. If the accuracies are limited to *Distance* values of about 400 and less, the regime for acceptable quantitative and qualitative work, the fastest technique in all three cases was the Adams-multistep routine, SLSODE, from the odepack package of the *Netlib* repository. When compared to the Runge-Kutta(4)5 routine, the Adams-multistep method was consistently 2 to 4 times faster to simulate the image.

The Adams-multistep method was developed to integrate non-stiff systems as shown in Table 2 and its exceptional performance is consistent with the non-stiff properties of the system of differential equations given by (2) and discussed in Section 4. Fig. 9 shows the computation time required to simulate a 128 128 pixel image of a quantum dot as a function of the number of times equation (2) was evaluated for 5 of the non-stiff solvers at a *Distance* of 350. As mentioned previously, the computation time required to simulate the image is linearly proportional to the number of function calls, which is consistent with the results presented in Section 4 where about 98% of the total computation time for an image simulation incorporating 21 beams is due to function calls. Thus, the key to the success of the Adams-multistep method is that it requires fewer function calls than other methods in order to achieve a given accuracy.

6 Conclusions

TEM images of defects using diffraction contrast were simulated using the Bloch-wave method, whereby the excitation amplitudes were formulated as a system of first-order coupled differential equations that were scaled by the gradient of the elastic displacement field in the direction of the electron beam. There was one differential equation for each beam included in the simulation and the time required for the integration increased as the cube of the number of beams. Various integration methods were investigated. Three different strain systems were used as test cases, an InAs quantum dot in InP, an InAs spherical inclusion in InP and a screw dislocation in InP.

It appeared that the integration required about 90% of the total simulation time for 11 beams, a percentage that increased with the number of beams included in the calculation. The system of differential equations for the excitation parameters did not display any stiffness over the range of conditions investigated.

The Adams-multistep method as implemented in the SLSODE routines provided the shortest computation times for all three test cases. It was 2 to 4 times faster than the Runge-Kutta(4)5 method. The success of the Adams-multistep method was due to it being the only non-stiff solver out of all of the methods investigated, which required the fewest number of function evaluations in the integration.

7 Acknowledgments

The authors gratefully acknowledge the support provided by the Natural Sciences and Engineering Research Council of Canada and one of the authors (MR) wishes to acknowledge the support of the Canada Research Chairs Program.

References

[1] C.J. Humphreys, The scattering of fast electrons by crystals, Rep. Prog. Phys., Vol. 42, University of Oxford, 1979.

[2] L. Reimer, Transmission Electron Microscopy - Physics of Image Formation and Microanalysis, Springer–Verlag, New York, third edition, 1993.

[3] A. J. F. Metherell, Electron Microscopy in Materials Science, Third Course of the International School of Electron Microscopy, Commission of Euro Communities, Directorate General, Scientific and Technical Information, Luxenbourg, 1975.

[4] E. J. Kirkland, Advanced Computing in Electron Microscopy, Plenum Press, New York, 1998.

[5] J. M. Cowley and A. F. Moodie, Acta Cryst. 10 (1957) 609.

[6] P. Goodman and A. F. Moodie, Acta Cryst. A30 (1974) 280.

[7] M. F. Ashby and L. M. Brown, Phil. Mag. 8 (1963) 1083.

[8] M. F. Ashby and L. M. Brown, Phil. Mag. 8 (1963) 1649.

[9] A. K. Head, P. Humble, M. Clarebrough, A. J. Morton and C. T. Forwood, Defects in Crystalline Solids - Computed Micrographs and Defect Identification, Vol. 7, North Holland Publishing Company, Amsterdam, 1973.

[10] K. G. F. Janssens, O. Van der Biest, J. Vanhellemont and H. E. Maes, Ultramicroscopy 69 (1997) 151.

[11] K. Tillmann, N. Hüging, H. Trinkaus and M. Luysberg, Microscopy and Microanalysis 10 (2004) 199.

[12] D. Cohen and C. B. Carter, J. Micros. 208 (2002) 84.

[13] M. D. Robertson, J. C. Bennett, A. M. Webb, J. M. Corbett, S. Raymond and P. J. Poole, Ultramicroscopy 103 (2005) 205.

[14] P. Hirsch, A. Howie, R. Nicholson, D. W. Pashley and M. J. Whelan, Electron Microscopy of Thin Crystals, Krieger Publishing Company, Malabar, FL, 1977.

[15] The LAPACK project. LAPACK–Linear Algebra PACKage, http://www.netlib.org/lapack.

[16] I. Vurgaftman, J. R. Meyer and L. R. Ram-Mohan, Band parameters for III-V compound semiconductors and their alloys, J. Appl. Phys. 89 (2001) 5815.

[17] D. B. Williams and C. B. Carter, Transmission Electron Microscopy, Plenum, New York, 1996.

[18] J. P. Hirth and J. Lothe, Theory of Dislocations 2nd Edition, Krieger Publishing Company, Malabar, FL, 1992.

[19] D. M. Bird and Q. A. King, Acta Cryst. A46 (1990) 202.

[20] D. M. Bird, Acta Cryst. A46 (1990) 208.

[21] J. S. Reid, Acta Cryst. A39 (1983) 13.

[22] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, NUMERICAL RECIPES in Fortran 77, Second Edition, The Art of Scientific Computing, Cambridge University Press, Cambridge, 1986.

[23] The Netlib repository, http://www.netlib.org/.

[24] E. Hairer, S.P. Norsett and G. Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems, Springer–Verlag, New York, second edition, 1993.

[25] E. Hairer and G. Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems, Springer–Verlag, New York, second edition, 2004.

[26] D. Kahaner, C. Moler and S. Nash, Numerical Methods and Software, Prentice-Hall Inc., New Jersey, 1989.

[27] U. M. Ascher and L. R. Petzold, Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, Society for Industrial and Applied Mathematics, Philadelphia, 1998.

[28] Y. Androussi, Phil. Mag. Lett. 79 (1999) 201.

[29] R. Bulirsch and J. Stöer, Numerical treatment of ordinary differential equations by extrapolation methods, Numer. Math. 8 (1966) 1.

[30] P. Deuflhard, Order and Stepsize Control in Extrapolation Methods, Numer. Math. 41 (1983) 399.

[31] K. Radhakrishnan and A. C. Hindmarsh, Description and Use of LSODE the Livermore Solver for Ordinary Differential Equations, Lawrence Livermore National Laboratory Report UCRL-ID-113855, Livermore, CA, 1993.

[32] J. D. Lambert, Numerical methods for ordinary differential equations, Wiley, Chichester, 1991.

Code Name	Numerical Method	Stiff	Non-Stiff	Package
odeint	Runge Kutta 4(5)		✓	Num. Rec.
rksuite	Runge Kutta 2(3), 4(5), 7(8)		✓	ode
DEABM	Adams PECE		✓	slatec
BSSTEP	Bulirsch-Stoer		✓	Num. Rec.
sdaskr	BDF	✓		ode
sdriv2				
(MINT = 1)	Adams Multi-Step		✓	slatec
cdriv2				
(MINT = 1)	Adams Multi-Step		✓	slatec
(MINT = 2, MITER = 1)	BDF	✓		slatec
(MINT = 2, MITER = 2)	BDF	✓		slatec
(MINT = 3)	BDF / Adams Multi-Value	✓	✓	slatec
SLSODE				
(MF = 10)	Adams Multi-Step		✓	odepack
(MF = 20)	BDF	✓		odepack
(MF = 21)	BDF	✓		odepack
svode				
(MF = 10)	Adams Multi-Step		✓	ode

Table 1: Summary of software tested. MINT, MITER and MF are user supplied settings for configuring the integration routine to operate in a particular manner. MINT was a software switch to allow the user to control whether a stiff or non-stiff method was to be used in the integration procedure. MITER = 1 allowed a user defined Jacobian to be evaluated whereas MITER = 2 set the option for a numerically evaluated Jacobian. Similarly, MF = 20 and 21 refer to user defined and numerically calculated Jacobians, respectively.

	CDRIV		LSODE	
Tolerance	Adams (Non–Stiff)	BDF (Stiff)	Adams (Non-Stiff)	BDF (Stiff)
0.1	0.2035	0.2533	0.08594	0.11947
0.01	0.3109	0.4004	0.09375	0.15007
0.001	0.5533	0.6701	0.11068	0.14388
0.0001	0.8621	0.9766	0.17969	0.23991
0.00001	1.1258	1.3901	0.30990	0.41016

Table 2: Comparison of computation times (seconds) as a function of user supplied tolerance for two stiff and non–stiff solvers. The timing results were from an average of 10 runs at a single point as highlighted in Fig. 3. The calculations were performed for 21 beams at the [001] zone-axis orientation.

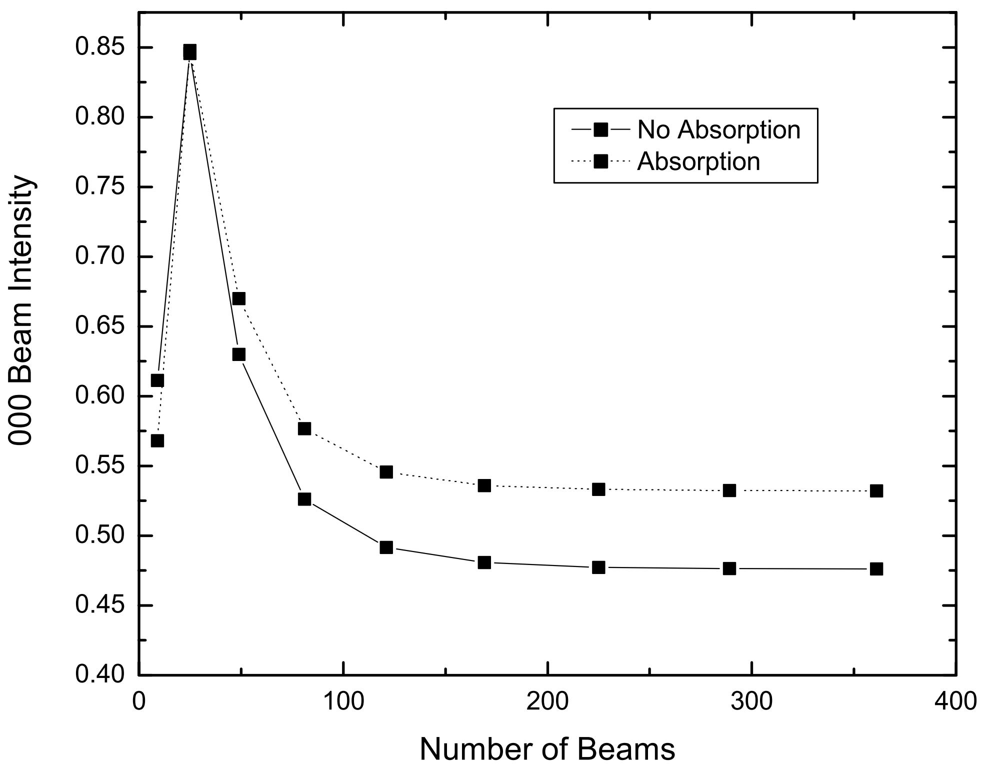

Figure 1: The bright–field intensity of 50 nm of unstrained InP in the [001] zone–axis orientation plotted as a function of the number of beams used in the calculation.

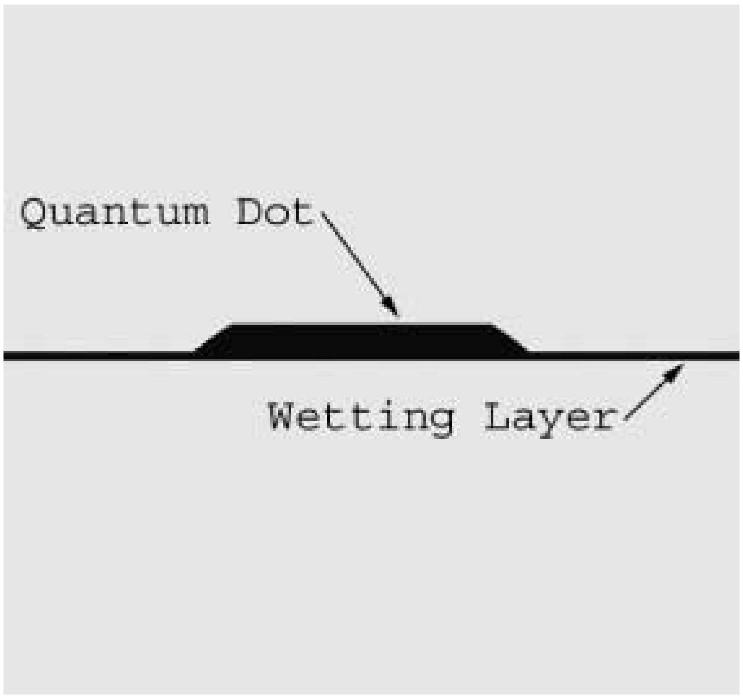

Figure 2: The cross sectional geometry of the InAs (black) quantum dot and wetting layer in the InP matrix (grey). The full width of the image is 50 nm.

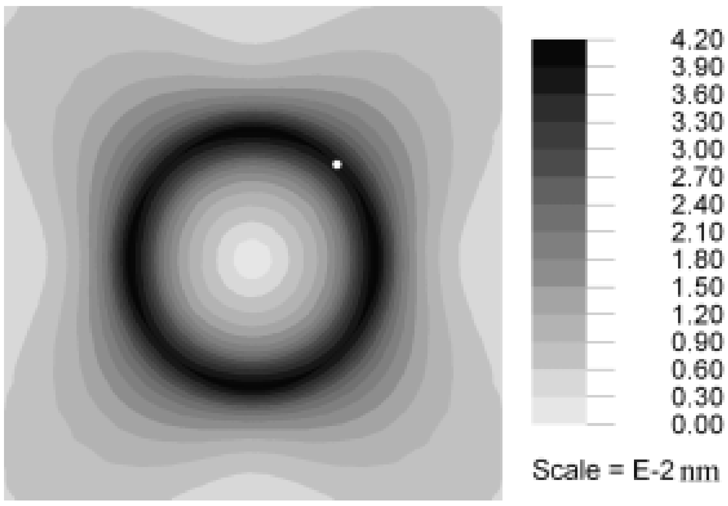

Figure 3: The magnitude of the total atomic displacements, $R_T = \sqrt{R_x^2 + R_y^2}$, due to the lattice misfit between the InAs quantum dot and the InP matrix. The location where the line integrations were performed is highlighted by the white circle. Full width of image = 50 nm.

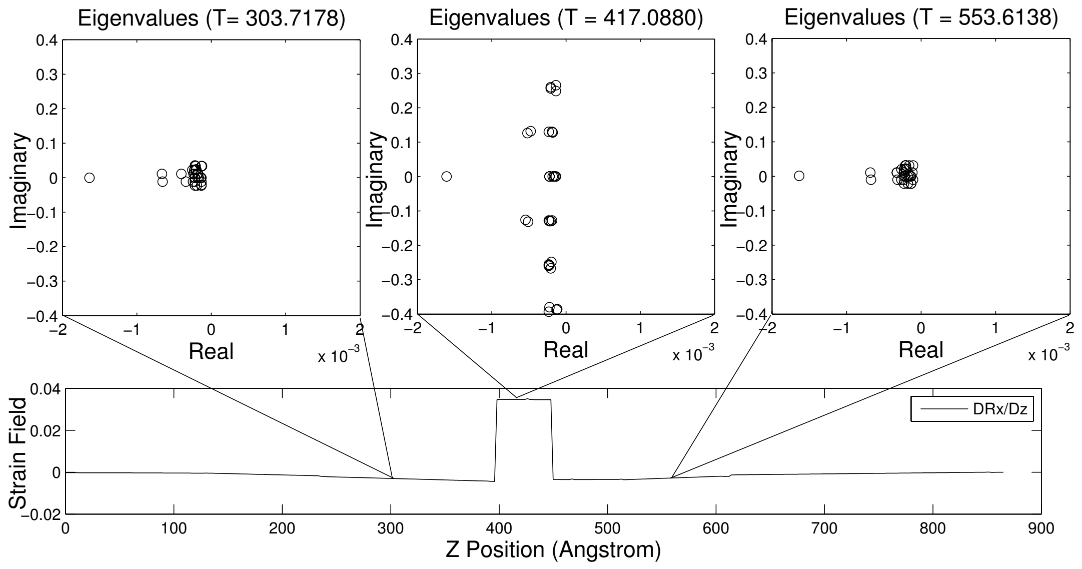

Figure 4: Eigenvalues of the Jacobian matrix shown in relation to the gradient of the displacement field (dR_x/dz, below) for 31 beams at the [111] zone-axis orientation. The calculations were performed at the single point highlighted in Fig. 3.

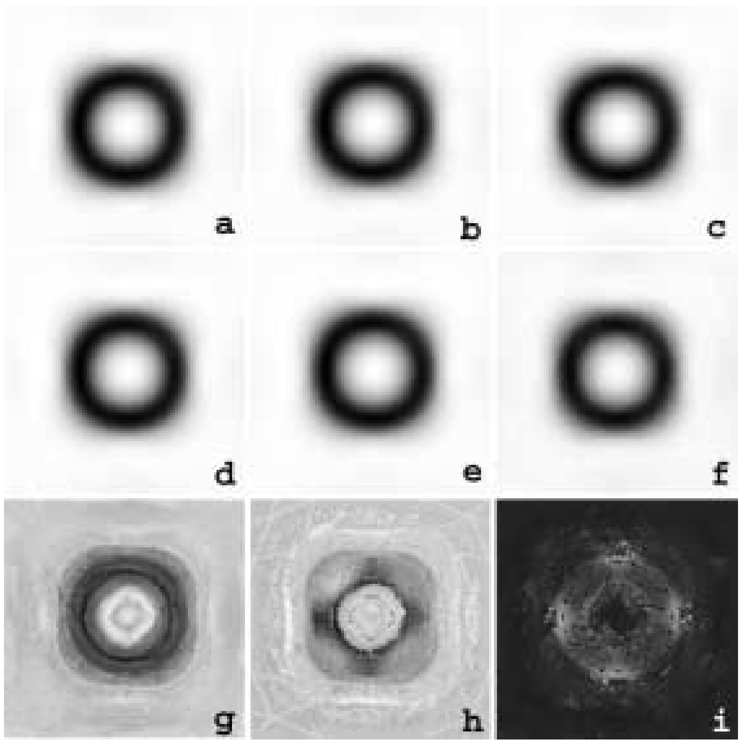

Figure 5: Simulated images of a truncated InAs QD in InP for varying values of the *Distance* parameter: a) 1.1, b) 2.0, c) 11, d) 29, e) 380, f) 2000, g) 21000, h) 53000 and i) 225000. The simulations were performed using 21 beams at the [001] zone-axis orientation and each image was scaled so that black/white represented the minimum/maximum intensity present within a particular image, respectively.

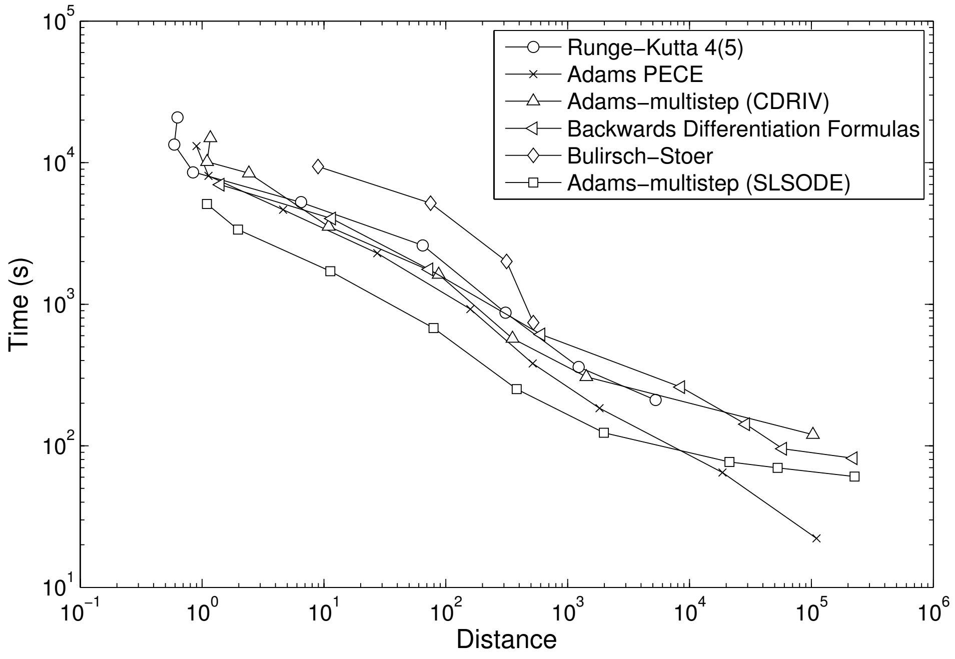

Figure 6: Calculation time as a function of the *Distance* (defined in the text) and calculation method for an InAs quantum dot in InP. The simulations were performed using 21 beams at the [001] zone-axis orientation.

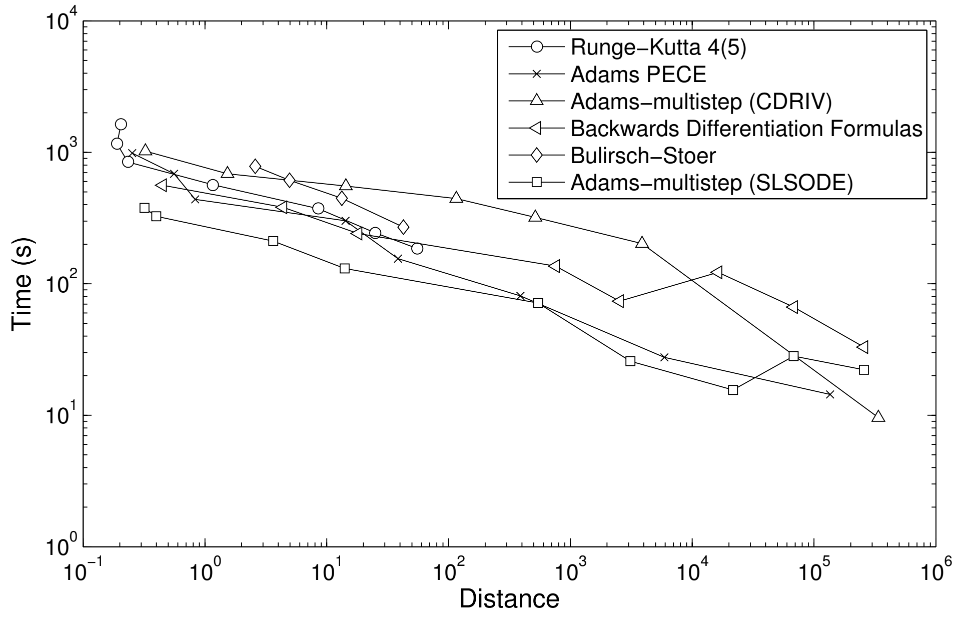

Figure 7: Calculation time as a function of the *Distance* (defined in the text) and calculation method for an InAs spherical inclusion in InP. The simulations were performed using 21 beams at the [001] zone-axis orientation.

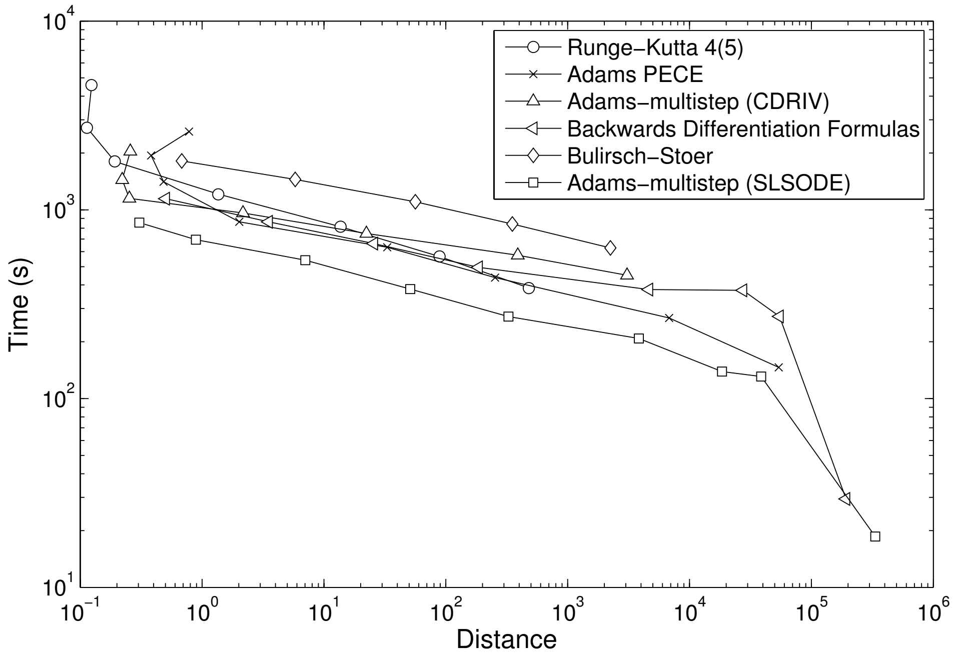

Figure 8: Calculation time as a function of the *Distance* (defined in the text) and calculation method for a screw dislocation in InP. The simulations were performed using 21 beams at the [001] zone-axis orientation.

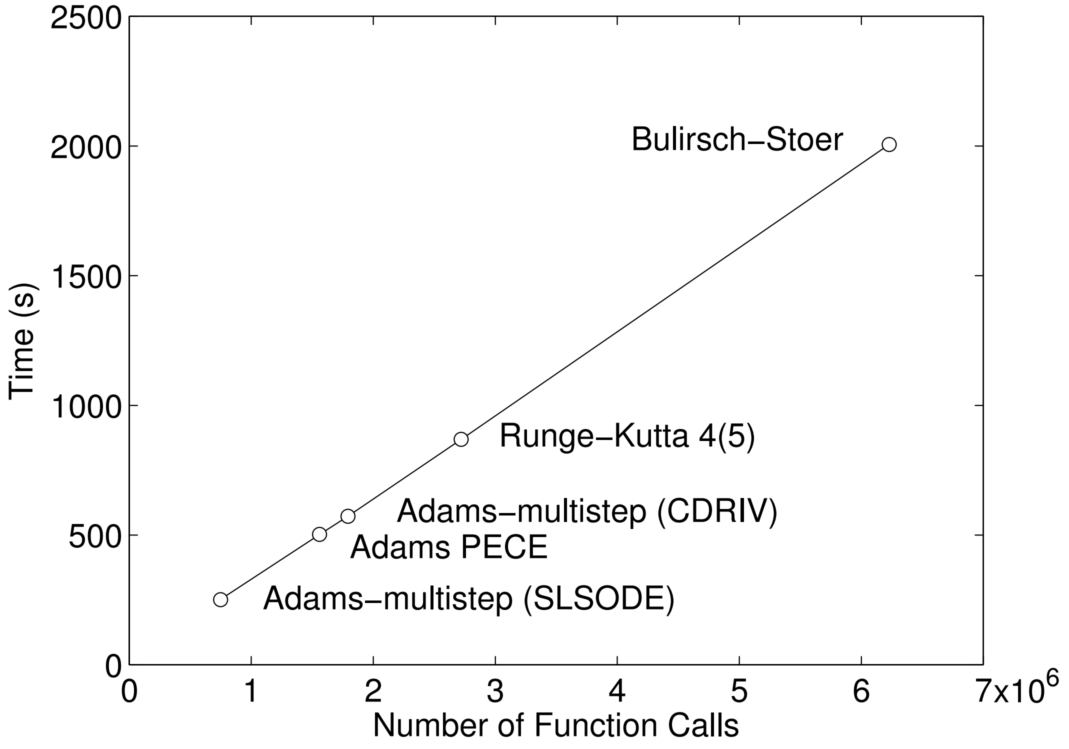

Figure 9: The time required to generate an image with a *Distance* value of 350 plotted against the number of function evaluations performed for 5 of the non-stiff integration routines.