Time-stepping methods for the simulation of the self-assembly of nano-crystals in MATLAB on a GPU

M. D. Korzec [*] T. Ahnert [†]

March 12, 2013

Abstract

Partial differential equations describing the patterning of thin crystalline films are typically of fourth or sixth order, they are quasi- or semilinear and they are mostly defined on simple geometries such as rectangular domains. For the numerical simulation of these kind of problems spectral methods are an efficient approach. We apply several implicit-explicit schemes to one recently derived PDE that we express in terms of coefficients of trigonometric interpolants. While the simplest IMEX scheme turns out to have the mildest step-size restriction, higher order SBDF schemes tend to be more unstable and exponential time integrators are fastest for the calculation of very accurate solutions. We implemented a reduced model in the EXPINT package syntax [3] and compared various exponential schemes. A convexity splitting approach was employed to stabilize the SBDF1 scheme. We show that accuracy control is crucial when using this idea, therefore we present a time-adaptive SBDF1/SBDF1-2-step method that yields convincing results reflecting the change in timescales during topological changes of the nanostructures. The implementation of all presented methods is carried out in MATLAB. We used the open source GPUmat package to gain up to 5-fold runtime benefits by carrying out calculations on a low-cost GPU without having to prescribe any knowledge in low-level programming or CUDA implementations and found comparable speedups as with MATLAB's PCT or with GPUmat run on Octave.

Keywords: Quantum dot self-assembly, time-stepping, GPU computing, IMEX, pseudospectral method, convexity splitting, exponential integrators, MATLAB

1 Introduction

There exists a large amount of literature that is concerned with the time-stepping of ODEs of the following type

$$u_t = \mathrm{L}u + \mathrm{N}(u) \in \mathbb{C}^N \,. \tag{1.1}$$

Many discretized PDEs can be written in this form with a linear operator L and a nonlinearity N, both are typically obtained from the application of suitable differentiation matrices. In this work we describe and compare several methods that have been used to solve an equation of type (1.1)

[*]Institute of Mathematics, Technical University Berlin, D-10623 Berlin, Straße des 17. Juni 136, Germany (korzec@math.tu-berlin.de).

[†]Institute of Mathematics, Technical University Berlin, D-10623 Berlin, Straße des 17. Juni 136, Germany (ahnert@math.tu-berlin.de).

numerically. The underlying PDE is a recently derived model for the self-arrangement of certain thin crystalline films. The physical set-up will be introduced later on, however, for a detailed treatment of the modeling and the analysis of the self-assembly of nanostructures such as quantum dots, the interested reader should study also other literature, e.g. [6, 12, 20, 21, 30, 33].

We treat the problem in discrete Fourier space, so that u is an N-vector of complex numbers, and we will show how the operators in (1.1) can be obtained from spectral differentiation in Fourier space. More precisely, the terms in the original evolution equation are interpolated trigonometrically. The coefficients for these globally defined functions form a system as above. Our pseudospectral method benefits from highly accurate derivatives. In the work by Korzec and Evans, where the underlying PDE has been derived [20], the linear part, which is a diagonal $N \times N$ matrix in the vectorial notation (1.1), has been treated implicitly while the nonlinearity has been approximated explicitely in a simple Euler type scheme that we will refer to as SBDF1 (semi-implicit backward differentiation formula). In this work we improve the method. We carry out a study by implementing several schemes known to serve well for problems of the form (1.1) and discuss them in terms of stability, accuracy and runtime. First of all we can stabilize the numerics by applying the convexity splitting idea in a very similar fashion as it has been used in connection with the Cahn-Hilliard equation. It has been introduced by Eyre [10], a more detailed discussion is given in [34], and it has been also applied successfully to related problems, such as binary inpainting [29]. Note that this approach yields stability, an unconditional one at its best, but does not necessarily bring along accuracy. To gain the latter, main parts of this work are devoted to elaborated time-discretizations.

Although ODE methods such as the Runge-Kutta formulae exist since more than 100 years [28], the development and improvement of schemes for discretized PDEs that can be written in the form (1.1) continues. Kassam and Trefethen found that an exponential time differencing (ETD) method with fourth order Runge-Kutta time-stepping (RK4) (see [18, 19]), a slight modification of the update by Cox and Matthews [8], lead to convincing results. We show that it is also an efficient method for the simulation of quantum dot self-assembly. We find that in terms of calculating accurate solutions the ETD4RK method is superior to semi-implicit backward-time differencing (SBDF) schemes (see [2]). The simple SBDF1 scheme has best time step restriction properties. If we are interested in qualitative behavior only, a simple, straightforward implementation suffices, while for quantitative agreement (e.g. for comparisons of time scales with experiments) we suggest fourth order exponential integrators, such as the ETD4RK method, as better alternative. We show that the application of the convexity splitting approach [10] is useful for our equation, but that it has to be applied with care, as the possibility to carry out larger steps can lead to very large errors. We extend the SBDF1 method by applying an SBDF1/SBDF1-2-step method of second order. In this way we gain error control and improve the order of the SBDF1 time-stepping procedure without much effort, and the convexity splitting approach can be applied in a more assured framework.

Because of the renewed interest in exponential integrators, the Matlab package EXPINT [3], containing a vast amount of schemes and examples, has been implemented a few years ago and the authors give an overview over the schemes. Grooms and Julien [13] analyzed a whole set of IMEX and ETD type methods on stiff systems of ODEs derived from PDEs. Also in this reference one can find a more complete discussion on high order integration methods. Overall we can conclude from these works that exponential time integration methods are competitive and accurate, but that results vary depending on the nonlinear problem one works on. We report on the method's applicability to the evolution equation describing quantum dot self-assembly, reduced to the two-dimensional setting. In particular, the ETD4RK method from before and a method by Strehmel and Weiner, see [32], turned out to be most efficient.

In a last step we transfer the calculations onto a GPU by exploiting the free *GPUmat* package, built on top of NVIDIA CUDA, and compare the runtime benefits with those obtained from the in-house MATLAB parallel computing toolbox (PCT). With a standard graphic card we obtained up to a five-fold acceleration in comparison to calculations carried out with standard workstation CPUs, in case of large-scale simulations. Therefore the user does not need to have any knowledge in CUDA or any deeper GPU architecture insights. We show a MATLAB implementation for the SBDF1 method and explain how to work with the GPUmat package for our needs. It is as efficient as the PCT, but one can save the costs for acquiring the package. Overall, for implementing a method that gives reasonable accuracy quickly, we suggest using an SBDF1/SBDF1-2-step method on a NVIDIA GPU. For the constant step method we we present a simple MATLAB code. It can be adjusted for a whole class of evolution equations.

We start the main body of this work with an introduction on the PDE describing the self-assembly of certain crystalline structures in Section 2, before we present the time-stepping methods applied to our problem, see Section 3. Next, in Section 4 it is explained how we use trigonometric interpolants to derive system (1.1) for the coefficients and we present how we split the operators for stability reasons. Section 5 is devoted to the actual results of the numerical simulations for the various time-stepping procedures. There we begin with few own implementations for the 2+1D setting and continue with many different methods from the EXPINT package in the reduced 1+1D case. In Section 6 we explain the benefit obtained by carrying out the computations on the GPU. Finally, in Section 7 we give a short summary and discuss our results and the future of GPU computing for PDEs in MATLAB. In the Appendix the reader may find corresponding MATLAB code snippets.

2 The quantum dot model

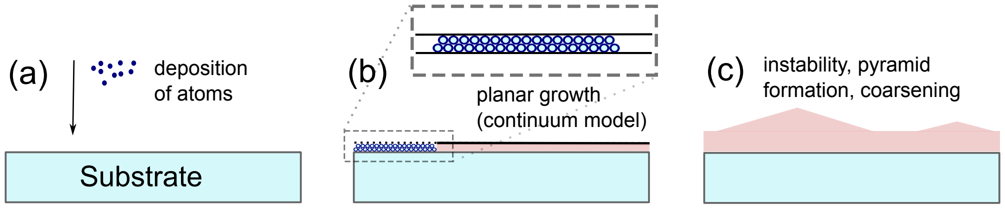
(a)
deposition of atoms
Substrate
(b)
planar growth (continuum model)
(c)
instability, pyramid formation, coarsening

Figure 1: 2D sketch of the Stranski-Krastanov growth + continuum approximation; (a) deposition of atoms onto a substrate; (b) planar growth and continuum approximation; (c) pyramids and coarsening during continued growth and/or annealing.

Recently a new model for the formation of quantum dots has been derived [20]. The corresponding experimental observations have been discussed in [9, 27]. Quantum dots are – generally speaking – three-dimensionally confined crystals on the nano-scale that find application in opto-electronics. They are in particular useful for blue lasers [26] or photovoltaics [24], as they have similar excitation and discrete energy level states as single atoms. There exist many different kinds of quantum dots, produced in complex processes that most often involve high temperatures, but do not rely on as high as melting temperatures. In the setting under consideration, atoms of one crystalline material such as germanium are slowly deposited onto a substrate, e.g. a silicon wafer that has perfectly arranged atoms in its natural crystalline cubic grid. The deposited adatoms (those absorbed by the substrate) adopt the positions of the substrate's ordering coherently. The process takes place in a hot chamber in near vacuum conditions. One observes a so-called Stranski-Krastanov growth-mode that is sketched in Figure 1. In the first stage of evolution a flat film forms that grows layer by layer in vertical direction. After a critical thickness is reached, a stress-driven instability occurs due to the lattice mismatch between the two materials. The Asaro-Tiller-Grinfeld (ATG) instability [1] releases the stresses and leads to ripple formation in the film. Eventually the surface decomposes into areas of a very thin film and islands, faceted pyramids that communicate through the thin wetting layer without introducing crystal failures at early stages of evolution. For a more detailed introduction to the topic, see [30]. In [20] a quasi-linear quantum dot self-assembly evolution equation has been obtained that was further extended to also capture strong surface energy anisotropies [21] with a randomly perturbed deposition term. For the purpose of numerical comparisons we treat a constant, nonperturbed positive flux F, constrain ourselves to weak anisotropies and analyze the model for the evolving surface $h = h(x_1, x_2, t)$ whose evolution is governed by the PDE

$$h_t = \nabla^2 \left(\mathcal{F}^{-1}[-\tilde{e}k\mathcal{F}[h]] - \nabla^2 h - \frac{\tilde{\gamma}}{h^2} - (\partial_{x_1}\partial_{h_{x_1}} + \partial_{x_2}\partial_{h_{x_2}})W(h_{x_1}, h_{x_2}) \right) + F. \tag{2.1}$$

Here, $\tilde{\gamma}$ is a wetting parameter, $\tilde{e}$ a material coefficient and $k = \sqrt{k_1^2 + k_2^2}$ is the length of the wave vector (k_1, k_2), corresponding to the Cartesian variables x_1 and x_2 in the spatial rectangular domain $[0, L_1] \times [0, L_2]$. $\mathcal{F}$ is the Fourier transform and $\mathcal{F}^{-1}$ is its inverse. The integral term stems from the elastic subproblem (linear elasticity in film and substrate with a mismatch condition at the interface) that has been treated by Tekalign and Spencer [33]. The fourth order derivatives of h always appear when incorporating a constant part in the surface energy density. The h^{-2} nonlinearity results from a boundary layer formula applied for the transition of surface energies between substrate and film. Finally, the W dependent term stems from the anisotropic part of the surface energy. In the discussed simulations we use a quadruple well representing the cubic anisotropy

$$W(h_{x_1}, h_{x_2}) = G((h_{x_1}^2 - 1)^2 + (h_{x_2}^2 - 1)^2) \geq 0, \tag{2.2}$$

i.e. four distinct slopes form a set of preferred orientations for the regularly growing surface. This formula makes the PDE quasilinear due to the quartic terms and larger anisotropy coefficients than $G = 0.25$ are forbidden. This has been indicated by a linear stability analysis as in [20], however, a rigorous well-posedness analysis has not yet been set up. In this work we concentrate on equation (2.1) with the anisotropy (2.2) and $G < 0.25$. Note that variations and extensions of the model are possible, where the same kind of numerical methods as those presented here would be applicable. In particular one can imagine different anisotropies and the additional effect of a sixth order term that stems from a corner regularization [21]. Finally we remark here that equation (2.1) is given in nondimensional form and all plots show nondimensional scales.

So far we have not talked about boundary conditions. As we are intrinsically forced to consider only a small part of a wafer due to the generic difference of its overall scale ($\sim$cm) and the dots characteristic lengths ($\sim$nm), one is content to work on sufficiently large parts of the wafer where the effects on the nano- or mesoscale are visible. Square domains with an extent of a few micrometers typically suffice. As the patterns of the dots are statistically of the same quality throughout the wafer, periodic boundary conditions are a realistic and suitable choice for this kind of patterning phenomena. When using Neumann boundary conditions instead, the evolution should not be affected too much either. As we apply a Fourier pseudospectral method, we stick to the periodic version. Figure 2 shows the typical evolution of a quantum dot array described by the above model. After a critical height is reached, h forms a ripple pattern that passes on to an array of pyramids that are connected by a thin layer. The images at later times, $t = 30, 150$ show how these islands evolve, i.e. how the average structure size grows. This happens in an Ostwald ripening fashion. The two images on the right of Figure 2 depict magnifications of one fourth of the domain at $t = 10$ and $t = 150$, clearly showing the effect of the surface energy anisotropy on the smoothly faceted geometry of larger islands.

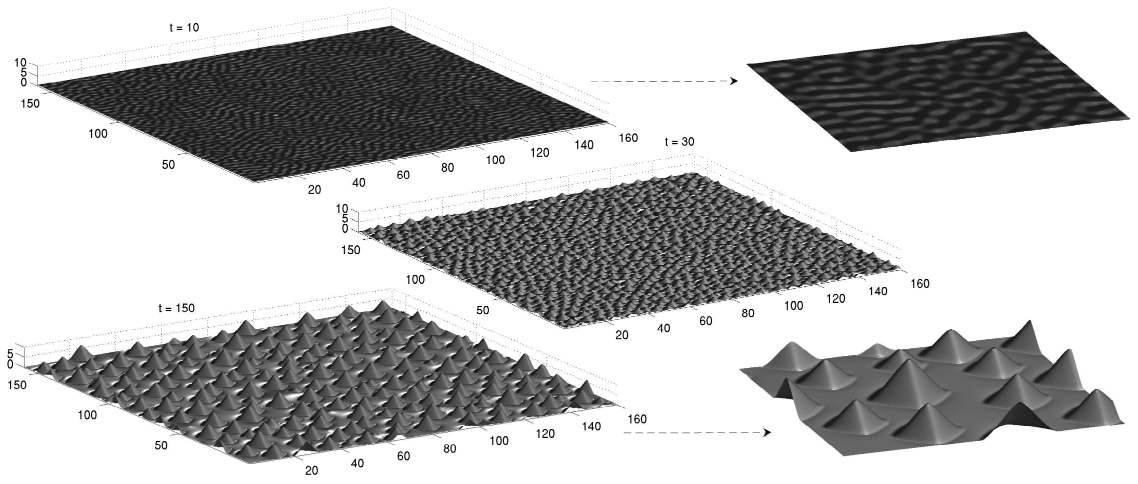

Figure 2: Evolution of quantum dot surface via equation (2.1).

3 Time-stepping methods

We work with one global interpolation method for the discretization of the spatial variables (presented in Section 4) and different time-stepping procedures. We first present the (multi-step) SBDF methods and the ETD4RK scheme based on fixed step sizes that are generally applicable for PDEs with linear and nonlinear contributions, and we shortly discuss the methods of the Matlab package EXPINT that we have used, too. Thereafter we show how we adapt the step sizes to the rates of changes of the evolving unknowns.

3.1 SBDF schemes and exponential integrators

We consider a problem that can be written as in (1.1) with a linear part L and a nonlinearity N. IMEX schemes extrapolate the nonlinearity from earlier time steps and are allowed to treat a portion of the linear part implicitly. The θ-method for the linear part and fixed explicit nonlinearity reads

$$u_{n+1} - u_n = dt\mathrm{N}(u_n) + dt[(1-\theta)\mathrm{L}(u_n) + \theta\mathrm{L}(u_{n+1})],$$

where the choice of $\theta \in [0, 1]$ defines different schemes. Here the subscript of the unknown vector u denotes the time level, $u_n = u(t = ndt)$, and dt is a fixed time step size. For $\theta = 0$ this is the well-known forward Euler scheme. For $\theta = 1$ the nonlinearity N is still extrapolated to time step t_{n+1}, but a first order backward differentiation scheme is used for the linear part L. Hence overall this is a first order semi-implicit backward differentiation formula (SBDF1). The update can be obtained by solving

$$(I - dt\mathrm{L})u_{n+1} = (u_n + dt\mathrm{N}(u_n))\,, \tag{3.1}$$

where I is the identity. It becomes particularly efficient for diagonal linearities, then the inverse becomes a scalar multiplication in each component, while in general one solves a system of linear equations.

The SBDF1 scheme can be generalized to higher order accuracy in terms of multi-step methods. These kind of methods have been presented by Ascher et al. [2] who derived it by stating a general equation for an s-step method with possibly implicit linear part and obtained a system of equations for an IMEX scheme of sth order. They are also given in Cox and Matthews work [8]. These are Adams-Bashforth backward differentiation type schemes, however, we stick to the SBDFk notation. By solving the system one obtains the following multistep SBDF2, SBDF3 and SBDF4 updates

$$(3I - 2dt\mathrm{L})u_{n+1} = 4u_n - u_{n-1} + 4dt\mathrm{N}(u_n) - 2dt\mathrm{N}(u_{n-1}), \tag{3.2}$$

$$(11I - 6dt\mathrm{L})u_{n+1} = 18u_n - 9u_{n-1} + 2u_{n-2} + dt[18\mathrm{N}(u_n) - 18\mathrm{N}(u_{n-1}) + 6\mathrm{N}(u_{n-2})], \tag{3.3}$$

$$\begin{aligned}(25I - 12dt\mathrm{L})u_{n+1} &= 48u_n - 36u_{n-1} + 16u_{n-2} - 3u_{n-3} \\ &\quad + dt[48\mathrm{N}(u_n) - 72\mathrm{N}(u_{n-1}) + 48\mathrm{N}(u_{n-2}) - 12\mathrm{N}(u_{n-3})]\,.\end{aligned} \tag{3.4}$$

The linear parts are treated as in the standard BDF updates and the nonlinearity does only depend on previous time steps. We can simply solve for u_{n+1} as before to obtain an explicit updating formulas.

Next, we introduce an exponential time differencing (ETD) method, the ETD4RK scheme. Multiplying the general equation (1.1) by $e^{-\mathrm{L}t}$ and integrating over one time step $[t_n, t_{n+1}] = [t_n, t_n + dt]$ one derives the update (for the more general case $\mathrm{N} = \mathrm{N}(u, t)$),

$$u_{n+1} = e^{\mathrm{L}dt}u_n + e^{\mathrm{L}dt}\int_0^{dt} e^{-\mathrm{L}\tau}\mathrm{N}(u(t_n + \tau), t_n + \tau)d\tau.$$

Dependent on the integral evaluation on the right hand side, many different methods are possible, similarly as for standard Runge-Kutta methods. Cox and Matthews [8] introduced an arbitrary order ETD scheme that was used to derive an ETD method based on a fourth order Runge-Kutta time-stepping. The update reads

$$\begin{aligned}u_{n+1} = e^{\mathrm{L}dt}u_n &+ \alpha_1\mathrm{N}(u_n, t_n) + \alpha_2(\mathrm{N}(a_n, t_n + dt/2) + \mathrm{N}(b_n, t_n + dt/2)) \\ &+ \alpha_3\mathrm{N}(c_n, t_n + dt)\end{aligned} \tag{3.5}$$

with the coefficients $\alpha_j, j = 1, 2, 3$ and a_n, b_n, c_n defined as in Appendix A. Kassam and Trefethen [19] made comparisons between different time-stepping methods and used also this ETD method. We adapted their code and the way they evaluated the coefficients α_j.

We show that the ETD method gives better results than the SBDF schemes, consequently we extend the analysis to more, related updates. Within the EXPINT package various exponential integrators have been implemented in Matlab [3]. We do not go into the details of the schemes, they can be found in the documentation file corresponding to the package. We will use the simple Crank-Nicholson (CN) scheme, the exponential time differencing methods ETD3RK, ETD4RK of third and fourth order, respectively, see [8], and the related fourth order schemes Lawson4 (L4, [23]), Adams-Bashforth Lawson4 (ABL4, [23]), Generalized Lawson41 (GL41, [22]), Friedli (F, [11]), Strehmel-Weiner (SW, [32]), Hochbruck-Ostermann4 (HO4, [15]). We will apply the methods to a dimension-reduced version of model (2.1).

3.2 Adaptive time-stepping

For increasing accuracy of a the low order SBDF1 formula, we make larger steps in regions with slow changes and small steps in the other case. The error control significantly improves the method, i.e. one captures the changes in the solutions that happen on varying time scales. We adjust the Euler/Euler-2-step method to a SBDF1/SBDF1-2-step variant, and calculate the two updates

$$(I - dt\mathrm{L})u_{n+1} = u_n + dt\mathrm{N}(u_n)\,,$$

and

$$(I - \frac{dt}{2}\mathrm{L})\bar{u}_{n+1/2} = u_n + \frac{dt}{2}\mathrm{N}(u_n)\,,$$
$$(I - \frac{dt}{2}\mathrm{L})\bar{u}_{n+1} = \bar{u}_{n+1/2} + \frac{dt}{2}\mathrm{N}(\bar{u}_{n+1/2})\,,$$

which are cheap to obtain when L is diagonal. This is true for our problem equation (2.1) equipped with periodic boundary conditions. Then the above systems are 'solved' by single multiplications in each component. This update yields a residual approximation by calculating their relative differences and assuming that the initial $u^n = u(t)$ is exact,

$$R = \|u^{n+1} - \bar{u}^{n+1}\|/dt \approx \frac{dt}{2}\|Lu'(t) - \frac{1}{2}u''(t)\|\,. \tag{3.6}$$

Once a tolerance ϵ is defined, one can proceed in the usual way: If $R \leq \epsilon$ the update is set to $u^+ = 2\bar{u}^{n+1} - u^{n+1}$, else, in the case when $R > \epsilon$, one repeats the time step starting from u^n with smaller dt. In both cases one can use the typical step size update

$$dt \leftarrow \nu\frac{\epsilon}{R}dt\,, \tag{3.7}$$

which incorporates the safety factor $\nu < 1$ to make rather too small than too large time steps. Typically $\nu = 0.9$ or $\nu = 0.95$ is used. Note that by updating in the above fashion, one reduces the truncation error by one order. While $u(t + dt) = \bar{u}^{n+1} + \mathcal{O}(dt^2)$, the new update corresponds to $u(t + dt) = 2\bar{u}^{n+1} - u^{n+1} + \mathcal{O}(dt^3)$. Depending on the definition on R, e.g. when taking the time-absolute error (dtR in (3.6)), the update (3.7) needs to be adjusted accordingly, e.g. by taking the square-root of ϵ/R.

4 A trigonometric interpolation method for PDE (2.1) and convexity splitting

In this section we show how the previously introduced PDE (2.1) with periodic boundary conditions can be written in the form (1.1). We write the bracket on the right hand side in (2.1) as

$$\mu = \mathrm{L}_\mu + \mathrm{N}_\mu\,,$$
$$\mathrm{L}_\mu = \mathcal{F}^{-1}[-\tilde{e}k\mathcal{F}[h]] - \nabla^2 h\,, \tag{4.1}$$
$$\mathrm{N}_\mu = -\frac{\tilde{\gamma}}{h^2} - 4G(\partial_{x_1}(h_{x_1}^3 - h_{x_1}) + \partial_{x_2}(h_{x_2}^3 - h_{x_2}))\,. \tag{4.2}$$

The nonlinear part (4.2) can be generalized for different anisotropies than (2.2). Using the quadruple-well we have a linear part in (4.2), however, we leave it in this form in scope of possible other anisotropy formulas. To discretize the right hand side of (2.1) we apply a Fourier collocation method (see for example [5]). Therefore we interpolate above defined expressions in terms of trigonometric global functions, using the complex exponentials as basis. As we prescribe periodic boundary conditions, this set seems a generic choice.

Let us consider the rectangular domain $\Omega = [0, L_1] \times [0, L_2]$ with N_1 equidistant grid points in x_1 and N_2 in x_2 directions. The corresponding two-dimensional discrete Fourier transform and its inverse are

$$\hat{u}_{\mathbf{k}} = N^{-1} \sum_{\mathbf{j}} u_{\mathbf{j}} e^{-i\mathbf{k}\cdot\mathbf{x_j}}, \quad u(\mathbf{x_j}) = \sum_{\mathbf{k}} \hat{u}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{x_j}}, \tag{4.3}$$

where $\mathbf{j} = (j_1, j_2)$, $\mathbf{k} = (k_1, k_2)$, $\mathbf{x_j} = \left(\frac{2\pi}{N_1} j_1, \frac{2\pi}{N_2} j_2\right)$ and $N = N_1 N_2$ is the overall number of grid points. The N-weight of both transforms can be found distributed in different ways elsewhere. The quantities $\hat{u}_{\mathbf{k}}$ and $u(\mathbf{x_j})$ can be stored in one complex and one real $N_1 \times N_2$ matrix. Replacing $\mathbf{x_j}$ in the inverse transform in (4.3) by an arbitrary $\mathbf{x} \in [0, L_1] \times [0, L_2]$, one sees that the inverse discrete Fourier transform directly brings along the trigonometric interpolant

$$\mathrm{I}_{\mathbf{n}} u(\mathbf{x}) = \sum_{\mathbf{k}} \hat{u}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{x}},$$

where $\mathbf{n} = (N_1, N_2)$. Now if u is sufficiently smooth, the derivatives of $\mathrm{I}_{\mathbf{n}} u$ are spectrally accurate, see e.g. Canuto et al. [5].

The wavenumber pairs $\mathbf{k}$ have to be chosen out of a suitable set $\mathcal{K}$. Its ordering is essential for correct calculations and it depends on the Fourier transform used by the programming language under consideration. The above interpolant has very simple derivatives, as e.g.

$$\partial_{x_l} \mathrm{I}_{\mathbf{n}} u(\mathbf{x}) = \sum_{\mathbf{k}} i k_l \hat{u}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{x}}, l \in \{1, 2\}, \ \nabla^2 \mathrm{I}_{\mathbf{n}} u(\mathbf{x}) = \sum_{\mathbf{k}} -|\mathbf{k}|^2 \hat{u}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{x}}, \quad |\mathbf{k}| = \sqrt{k_1^2 + k_2^2}.$$

We treat the global transforms in (4.1) as discrete ones, which we just defined in (4.3). Then we use the global interpolant $\mathrm{I}_{\mathbf{n}} h(\mathbf{x})$ instead of h itself, to gain the approximate linear part

$$\begin{aligned} \mathrm{L}_\mu &\approx \mathcal{F}^{-1}[-\tilde{e}|\mathbf{k}|\mathcal{F}[\mathrm{I}_{\mathbf{n}} h(\mathbf{x})]] - \nabla^2 \mathrm{I}_{\mathbf{n}} h(\mathbf{x}) \\ &= \sum_{\mathbf{k}} (-\tilde{e}|\mathbf{k}| + |\mathbf{k}|^2) \hat{h}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{x}}. \end{aligned}$$

For the other terms we interpolate the nonlinearities

$$\begin{aligned} \mathrm{N}_\mu &\approx -\mathrm{I}_{\mathbf{n}}[\frac{\tilde{\gamma}}{h^2}] - 4G \left(\partial_{x_1} \mathrm{I}_{\mathbf{n}}[h_{x_1}^3 - h_{x_1}] + \partial_{x_2} \mathrm{I}_{\mathbf{n}}[h_{x_2}^3 - h_{x_2}]\right) \\ &= \sum_{\mathbf{k}} \left(-\tilde{\gamma}\hat{v}_{\mathbf{k}} - 4Gi(k_1 \hat{\xi}_{\mathbf{k}} + k_2 \hat{\eta}_{\mathbf{k}})\right) e^{i\mathbf{k}\cdot\mathbf{x}}, \end{aligned}$$

with the two-dimensional transforms $\hat{v} = \mathcal{F}(1/h^2)$, $\hat{\xi} = \mathcal{F}(h_{x_1}^3 - h_{x_1})$ and $\hat{\eta} = \mathcal{F}(h_{x_2}^3 - h_{x_2})$. Inserting these expressions, using that the coefficients are time-dependent, into evolution equation (2.1), we gain

$$\begin{aligned}\sum_{\mathbf{k}} \partial_t \hat{h}_{\mathbf{k}} e^{i\mathbf{k}\cdot\mathbf{x}} &= \nabla^2 \sum_{\mathbf{k}} \Big((-\tilde{e}|\mathbf{k}| + |\mathbf{k}|^2)\hat{h}_{\mathbf{k}} - \tilde{\gamma}\hat{v}_{\mathbf{k}} - 4Gi(k_1\hat{\xi}_{\mathbf{k}} + k_2\hat{\eta}_{\mathbf{k}})\Big) e^{i\mathbf{k}\cdot\mathbf{x}} + F \\ &= \sum_{\mathbf{k}} \Big((\tilde{e}|\mathbf{k}|^3 - |\mathbf{k}|^4)\hat{h}_{\mathbf{k}} + |\mathbf{k}|^2\tilde{\gamma}\hat{v}_{\mathbf{k}} + 4Gi|\mathbf{k}|^2(k_1\hat{\xi}_{\mathbf{k}} + k_2\hat{\eta}_{\mathbf{k}})\Big) e^{i\mathbf{k}\cdot\mathbf{x}} + F\,.\end{aligned}$$

As F can be expressed as zeroth mode, $F = F\sum_{\mathbf{k}} \mathbb{1}_{\mathbf{k}=(0,0)} e^{i\mathbf{k}\cdot\mathbf{x}}$, where $\mathbb{1}_{\mathbf{k}=(0,0)}$ is the characteristic function being 1 at the wavenumber origin and 0 else, we derived the ODEs for the coefficients

$$\partial_t \hat{h}_{\mathbf{k}} = \Big((\tilde{e}|\mathbf{k}|^3 - |\mathbf{k}|^4)\hat{h}_{\mathbf{k}} + |\mathbf{k}|^2\tilde{\gamma}\hat{v}_{\mathbf{k}} + 4Gi|\mathbf{k}|^2(k_1\hat{\xi}_{\mathbf{k}} + k_2\hat{\eta}_{\mathbf{k}})\Big) + F\mathbb{1}_{\mathbf{k}=(0,0)}, \mathbf{k} \in \mathcal{K}\,.$$

One can guess from this ODE that it is hard to treat the nonlinear parts implicitly and this is why our time-stepping methods will always treat parts of the problem in an explicit fashion. This reasons that decomposing the equation in explicit and implicit parts is the way to proceed. As there are infinitely many possibilities to do so, we want to understand what makes an efficient approach.

We still need to define a suitable set of wavenumbers $\mathcal{K}$, therefore we set

$$\mathcal{K} = \{(k_1, k_2) : k_j \in \{0, 1, \ldots, \frac{N_j}{2}, -\frac{N_j}{2} + 1, \ldots, \ldots, -1\}\frac{2\pi}{L_j}, j = 1, 2\}$$

or in Matlab notation, with the ordering corresponding to its FFT,

```
k1=[0:n1/2 -n1/2+1:-1]'*2*pi/L1; k2=[0:n2/2 -n2/2+1:-1]'*2*pi/L2;
```

which gives the dense wavenumber matrices `[kx, ky] = meshgrid(k,k);` or for example the Laplacian `-kx.^2-ky.^2`.

The above dense matrix approach (the wavenumbers used for differentiation are stored in dense matrices) is convenient for the implementation and all updates presented in Section 3 are directly applicable in this way and this is how we have written the code in Appendix A. However, for theoretical discussions we stick to the vector notation introduced in (1.1), hence the derivatives become diagonal operators. Therefore the matrices of coefficients are rearranged from matrices to vectors and $\hat{h}(l, m) \leftrightarrow u(p)$, say for example $p = (l-1)N_2 + m$. The complete linear and nonlinear parts corresponding to the notation in (1.1) are

$$\mathrm{L} = \mathrm{diag}\left((\tilde{e}|\mathbf{k}|^3 - |\mathbf{k}|^4), \mathbf{k} \in \mathcal{K}\right) \tag{4.4}$$

and

$$\mathrm{N}(u)_{\mathbf{k}\in\mathcal{K}} = (|\mathbf{k}|^2\tilde{\gamma}\hat{v}_{\mathbf{k}} + 4Gi|\mathbf{k}|^2(k_1\hat{\xi}_{\mathbf{k}} + k_2\hat{\eta}_{\mathbf{k}}) + F\mathbb{1}_{\mathbf{k}=(0,0)}, \mathbf{k} \in \mathcal{K}). \tag{4.5}$$

4.1 Convexity splitting

For high order equations that are partly treated in an explicit fashion, step size restrictions due to stability considerations can be severe. The convexity splitting idea introduced by Eyre for the Cahn Hilliard equation [10] allows to make larger time steps without blow-up, and leads to unconditionally stable schemes at its best. It has been discussed and implemented in the last years by many authors for related problems [17, 29, 34]. The original idea is to decompose the energy into an convex and a concave part in a suitable way – there are infinitely many such possibilities –, treat the convex part implicitly and the concave part in an explicit fashion.

We refrain from presenting a detailed description and analysis of the convexity splitting approach applied for evolution equation (2.1). Instead we proceed in an intuitive fashion and show that the splitting we present in the following is indeed capable of stabilizing the time-stepping procedures, but that it should be applied with care. Therefore, similarly as in the Cahn-Hilliard equation, we add and subtract the convex function

$$c(h, \nabla h) = \frac{c_1}{2} h^2 + \frac{c_2}{2} |\nabla h|^2$$

to the energy of the model and treat one of these terms implicitly and the other ones in an explicit fashion, here $c_j \geq 0, j = 1, 2$. By going through the derivation of the PDE this gives linear second and fourth order terms. We replace expressions (4.4) and (4.5) by

$$\mathrm{L}_c = \mathrm{diag}((\tilde{e}|\mathbf{k}|^3 - (1 + c_2)|\mathbf{k}|^4 - c_1|\mathbf{k}|^2), \mathbf{k} \in \mathcal{K}), \tag{4.6}$$

$$\mathrm{N}_c(h)(\mathbf{k}) = N(h)(\mathbf{k}) + c_1|\mathbf{k}|^2 + c_2|\mathbf{k}|^4, \mathbf{k} \in \mathcal{K}. \tag{4.7}$$

To test the effect of the two parameters c_1, c_2, we applied the SBDF1 update (3.1) with L_c, N_c and various values for c_1 and c_2 between 0 and 0.5. We noted approximate values of the time step dt, for which an instability occurs in the time interval $[0, 1]$, with $N_1 = N_2 = 64$, $L_1 = L_2 = 5, G = 0.2$ and the other material parameters as before. We evolved from a randomly perturbed state around $H = 0.5$ for a time interval of dimensionless length one and used this shape, plotted in Figure 3 (a), repeatedly as initial condition for different values of (c_1, c_2). If dt, and hence the error, is sufficiently small, the final stage, after a time interval of length 50, is as in Figure 3 (b). To calculate the critical step sizes at which the schemes blow-up, we employed a bisection method. When a solution extends a certain threshold during evolution, i.e. $\max |h(x, y, t)| > tol_{\text{blowup}}$, the time step is marked as a blow-up step size. In this way we can work with intervals $[dt_1, dt_2]$, where the simulation is stable for the smaller dt_1 and unstable for dt_2. Such intervals are then halved in each iteration giving an intermediate dt_3. Either dt_1 is set to dt_3 (in case the calculation with dt_3 is stable) or dt_2 is updated to dt_3 (in the other case). This is repeated until a threshold for the final dt is reached. Due to the 'arbitrary' choice of tol_{blowup} the found critical step sizes are not exact, but they allow for qualitative statements. We used a cut-off time step size $dt_{max} = 5$, where we stopped increasing the values.

The numbers we obtained would differ on different space, time and grid domains. Also note that the underlying problem leads to more stability difficulties at larger times, which is possible due to the quasilinear character of the PDE. Our tests show that the presented approach for stabilizing the scheme works. Figure 3 (c) plots the critical step sizes in dependence of both parameters, c_1 and c_2. As expected the stability restrictions are less severe when either increasing one of the quantities. We see that at some trajectory in the (c_1, c_2)-plane the instability is suddenly suppressed and one reaches the maximal allowed value dt_{max}. This has the following reason: Using large values for the regularization induces errors that can be, if working without care, huge. In real space on the level of the PDE (not the energy) the implicit-explicit components sum up to

$$c_2(h_{n+1} - h_n)_{xxxx} + c_1(h_{n+1} - h_n)_{xx} \approx c_2 dt h_{xxxxt} + c_1 dt h_{xxt}\,.$$

By construction these terms are supposed to be approximately zero, which is the case, if once again dt is sufficiently small. The sudden change of blow-up step sizes in Figure 3 (c) can be explained, because for large values of (c_1, c_2) one does not even see the physical instability (ATG) happening to the surface h that one wants to see.The shape in Figure 3 (b) is adopted, when using too large

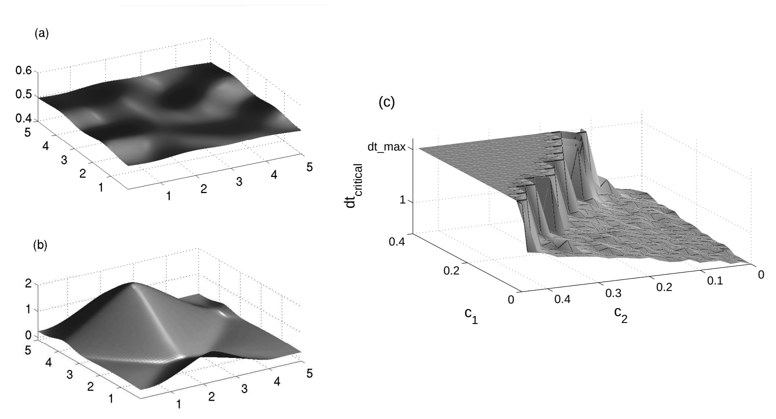

Figure 3: (a) Initial state; (b) correct end state; (c) approximated critical step size dependent on splitting parameters c_1, c_2. The vertical axis is in logarithmic scale, $dt_{max} = 5$, the minimal value, for the case without convexity splitting, is at $dt \approx 0.4$.

time steps. After evolving one dimensionless time one still is in a state similar to that given as initial condition. We conclude, convexity splitting is a possibility to stabilize schemes for the quantum dot model (2.1), but even if one is able to choose large time steps, it is not always a good idea. Note that it is possible to take larger values for the c_j, but in this case one should incorporate a control over the error.

5 Accuracy of the time-stepping methods

We report on the simulation runs carried out with the different methods introduced in Section 3. Therefore we work with a reduced 1+1-dimensional setting, as it is computationally hard to calculate a trustworthy reference solution in the full setting in a reasonable time frame. We believe that the results are similar if treating the full two-dimensional spatial domain, although quantities, such as blow-up step sizes may change. The results will show that the exponential time differencing method ETD4RK has stability benefits in comparison to higher order SBDF schemes, and that it is very accurate. Hence we investigated further, to more detail, with help of the EXPINT package in Section 5.2, to see if other exponential integrators can compete.

5.1 Simulations with SBDF and ETD4RK schemes

In Section 3 we introduced the ETD4RK method (3.5), and the SBDF schemes (3.1)-(3.4). We apply these methods the dimension-reduced version of PDE (2.1), namely

$$h_t = -\partial_{xxxx}h + \partial_{xx}\mathcal{F}^{-1}[-\tilde{e}|k|\mathcal{F}[h]] - \partial_{xx}\frac{\tilde{\gamma}}{h^2} - 4G\partial_{xxx}(h_x^3 - h_x), \tag{5.1}$$

here without the deposition term. The periodicity is now in the x-variable only. We used the parameters $G = 0.15, \tilde{e} = 1.28, \tilde{\gamma} = 0.05$ for all calculations. In terms of the coefficients of the Fourier interpolants we considered the problem

$$\begin{aligned} \hat{u}_t &= \mathrm{L}\hat{u} + \mathrm{N}(\hat{u})\,, \\ \mathrm{L}\hat{u} &= -k^4\hat{u} + \tilde{e}k^2|k|\hat{u}, \quad \mathrm{N}(\hat{u}) = k^2\mathcal{F}[\frac{\tilde{\gamma}}{\mathcal{F}^{-1}[\hat{u}]^2}] + 4Gik^3\mathcal{F}[\mathcal{F}^{-1}[ik\hat{u}]^3 - \mathcal{F}^{-1}[ik\hat{u}]]. \end{aligned}$$

Also here one could shift the linear part in N to the linearity. We leave it like that, but also work with the convexity splitting for the fourth order term, which gives with the parameter $c \geq 0$

$$\begin{aligned} \hat{u}_t &= \mathrm{L}_c\hat{u} + \mathrm{N}_c(\hat{u})\,, \\ \mathrm{L}_c\hat{u} &= -(1+c)k^4\hat{u} + \tilde{e}k^2|k|\hat{u}, \\ \mathrm{N}_c(\hat{u}) &= ck^4\hat{u} + k^2\mathcal{F}[\frac{\tilde{\gamma}}{\mathcal{F}^{-1}[\hat{u}]^2}] + 4Gik^3\mathcal{F}[\mathcal{F}^{-1}[ik\hat{u}]^3 - \mathcal{F}^{-1}[ik\hat{u}]]. \end{aligned}$$

Figure 4 shows a time space plot for the 1+1D problem. Starting with an initial condition above the

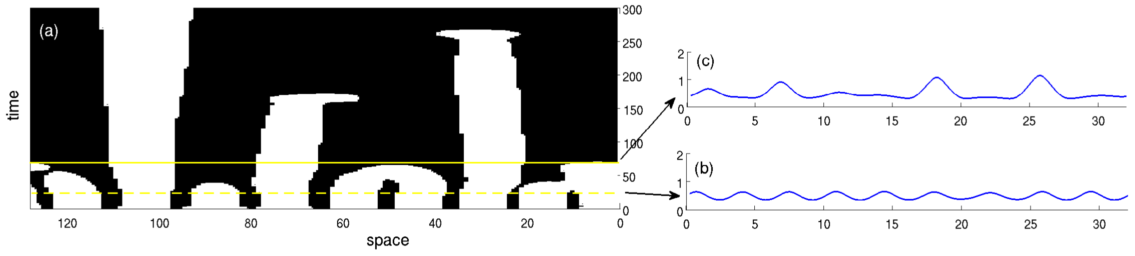

Figure 4: (a) Space-time plot for a calculation of approximate solutions to PDE (5.1). The white areas indicate islands; (b) early stage of evolution, wrinkle patterns, corresponding to the dashed line in (a); also: initial state for the EXPINT package tests; (c) later stage of evolution, 2D islands shapes after coarsening; end state for the EXPINT global error test.

critical thickness that is randomly perturbed, islands form and coarsen with time. In (a) the white areas depict the approximate islands (we used a simple height cut-off value that gives a rather coarse result), while the black areas represent the connecting thin layer. One clearly sees that after some time islands tend to collapse in an Ostwald ripening fashion. Figure 4 (b) shows the profile h at the level of the dashed line in (a), a rippled surface. We took this shape as initial condition for our calculations and also later for the tests with the EXPINT package; After an evolution of 50 dimensionless time units the profile $h(x,t)$ is similar as in Figure 4 (c), the final state, which has been used for global error measurements. To have an accurate solution we solved the problem with MATLAB's ode15s solver. With nearly as small as machine precision accuracy thresholds we obtained the final state depicted in Figure 4 (c). We denote it here as reference solution h_{ref}. To have proper initial conditions for the SBDF2, SBDF3 and SBDF4 methods, we again used the ode15s procedure to iterate accurately to capture the solution at the times $dt, 2dt$ and/or $3dt$. In this way we have accurate initial conditions. For the ETD4RK method we used a similar script as presented in [19].

We repeated several runs for a range of time step sizes and calculated the relative error $\|h - h_{ref}\|_\infty / \|h_{ref}\|_\infty$. Figure 5 depicts the global error plot. We see that both high order SBDF methods behave quite unstable. Especially in a range of dt that one typically likes to use, the solutions blew-up. This is indicated by missing markers for the corresponding step sizes and schemes. Our implementation of the SBDF4 method performs poorly, even when it converges, while the error with the SBDF3 scheme drops down satisfactory. It is questionable if these schemes should be applied for this kind of problem at all due to the severe step size restrictions. The convexity splitting approach did not stabilize the schemes in our tests. The ETD4RK method on the other side allowed for larger time steps and brought along higher accuracy than all of the SBDF schemes. In the range of larger time steps the error decreases like dt^2 and only for small values of dt the slope steepens. This reminds of the stiff order of the scheme, which is 2. The SBDF2 method is not much worse than the exponential integrator, while the SBDF1 method sticks to its first order. We conclude that if one chooses to work with a fixed time step size, the exponential time differencing method is preferable to the SBDF schemes. Only the SBDF2 method is a competitive alternative in accuracy and stability.

The question arises if the ETD4RK is the best exponential method there is and if the costs of the more expensive iterations are not slowing down the whole method. Furthermore it is interesting to see if one reaches the theoretically known decay rates for the local error. In the next section we will show consistent results for many different high order exponential schemes by working with the EXPINT package.

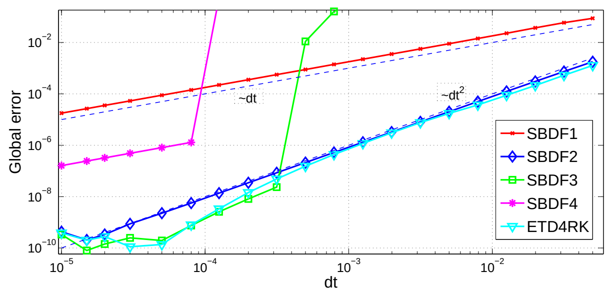

Figure 5: Doubly logarithmic global error plot for the SBDFj, j=1,2,3,4 schemes and for the ETD4RK method.

5.2 Comparisons using the EXPINT package

We found in the last Section 5.1 that the exponential scheme ETD4RK performed better than the SBDF schemes. This motivates to test also related exponential schemes. We were lucky to find the EXPINT package written in MATLAB [3]. It allowed us to test many other given exponential integrators, i.e. those introduced in the end of Section 3.1. Within the package PDE problems have been implemented that are similar to ours. The corresponding codes can be adjusted to our needs. We adapted the one-dimensional setting (5.1) to the syntax used in the package. This saved us a large amount of implementational work. In Appendix B we show and explain the MATLAB code snippets that were needed to incorporate this problem into the EXPINT package. How these can be used to calculate for example the global and local errors – presented in the following – is explained in the original reference [3].

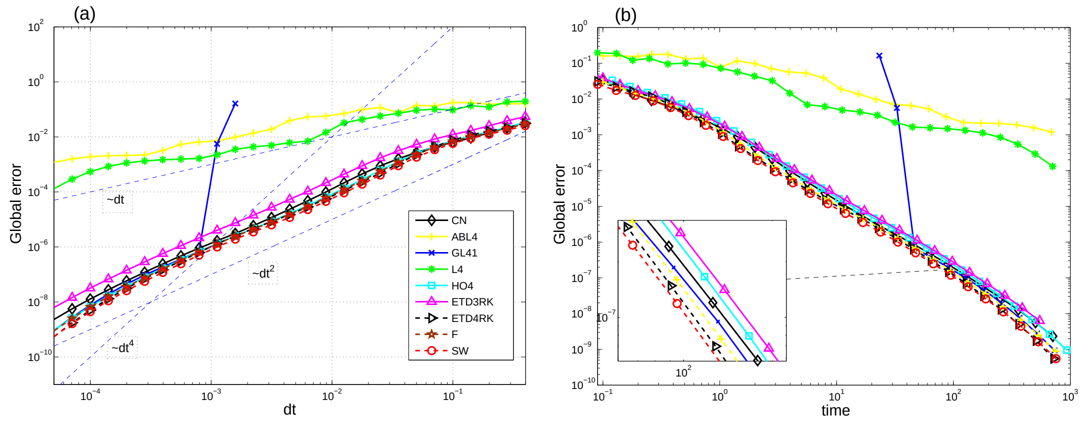

Figure 6: Global error results from application of different EXPINT schemes to PDE (5.1) in doubly logarithmic plots. (a) time step plotted against the error; (b) actual time needed for the calculations plotted against the error.

In Figure 6 we see the global error results obtained with different time-stepping procedures implemented in the EXPINT package. Therefore we used the parameters as befor and we applied convexity splitting with $(c_1, c_2) = (0, 1)$. The abbreviations in the figure have been introduced in the end of Section 3.1. Again we calculate the relative error to the reference solution. In Figure 6 (a) the time step size is plotted against the error. We see that apart from two methods, Lawson4 and its variant Adams-Bashforth Lawson 4, the error decreases initially (for larger values of dt) slower than dt^2, but that the slopes of the error curves steepen and eventually adopt a slope higher than dt^2 in the range of smaller time step sizes. They behave similarly as the ETD4RK method, which gives comparable numbers as our own implementation seen before. The GL41 method does not converge for larger time step sizes, but is very accurate for small ones, similarly as the SBDF3 and SBDF4 schemes before. Figure (b) gives us an idea which of the methods are indeed working most efficiently, as the time needed for one method to achieve a certain accuracy is plotted here. The box in the left bottom shows a magnification of the faster exponential integrators, here one sees that the Strehmel-Weiner method is fastest and that the ETD4RK method follows close by. It seems that all of the methods in the box are competitive. As the Lawson methods have either bad convergence or stability properties, they have to be marked as not very suitable for the application to our problem. Few other methods from the package not mentioned here, such as the ABNorsett4 scheme, failed at the whole time step range. Overall we can conclude that the ETD4RK method is one of the best exponential integrators one can use for solving PDE (5.1). Strehmel and Weiner's method, one of the earliest exponential Runge-Kutta methods, performs even more efficiently. We expect a similar result for the full equation

(2.1) as the stiffness due to the highest order term is influenced similarly by squares of the slopes h_x and h_y.

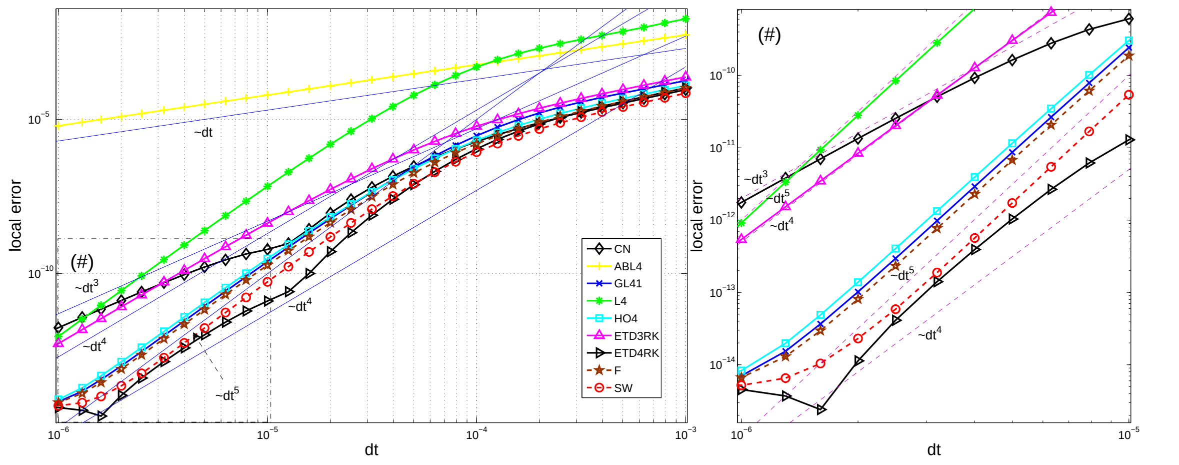

Figure 7: Doubly logarithmic local error plot for the different methods calculated with the EXPINT package applied PDE (5.1). The accuracy of one time step is tested with the initial condition depicted in Figure 4 (b). The dashed lines depict powers of the step sizes $Cdt^p, p = 1, 2, 3, 4$. The right plot is a magnification of the dashed marked square on the left, indicated by (#), showing the asymptotic orders expected by the methods.

To understand better how the results from the global error analysis can be explained and that they correspond well to the asymptotic theory as $dt \to 0$, we also calculated local errors. Again we started with the initial condition depicted in Figure 4 (b), but tested only one local step. Figure 7 shows the local errors made with the same methods as in Figure 6. As one has to calculate only one step, it is easier to analyze much smaller step sizes than in the global analysis. For a smaller range of dt values between 10^{-6} and 10^{-5} we magnified the error decays and plotted them again on the right of Figure 7. While the plots still do not explain the bad behavior of the ABL4 method, the original L4 scheme does better and indeed gives asymptotically the expected dt^5 decrease. However, the constant in front of the error seems so large that it leads to a very unfavorable delay until the theoretical slope is reached. Furthermore we observe the expected dt^3 and dt^4 slopes for the CN and the ETD3RK methods, respectively. All other schemes are of fourth order and give slopes proportional to dt^5. Only the ETD4RK scheme stands somewhat out as its slope bounces between different rates. The average however, seems to be good and in fact, for smaller values of dt we could even expect in the global analysis that the ETD4RK method becomes more accurate from a certain value of dt on.

5.3 The time-adaptive scheme

One can observe that the coarsening events, that is, the collapses of single dots, happens on a faster time scale than the other parts of the ripening process. Therefore it can be preferable to work with time adaptivity. We applied the SBDF1/SBDF1-2-step method presented in Section 3.2 to PDE (2.1). We used 256×256 wavenumbers for a 50×50 domain. For the convexity splitting we used $(c_1, c_2) = (0, 2)$, the safety parameter was chosen $\nu = 0.95$, the maximal step was 0.1 and the tolerance was relatively large, $\epsilon = 0.01$. We used the sum of squares as norm for the residual, which becomes easily large, so that we were content with this threshold. Figure 8 visualizes the evolving shapes that change with time as expected in the Stranski-Krastanov growth mode. The signal in the top left explains how the time step changes with time. As in the early stage of evolution many dots collapse, it remains small, and it grows at later times, always during intervals without coarsening events. The dashed ellipse in the signal and in the two plotted surfaces at $t = 102.41$ and $t = 103.792$ shall make visible that the collapse of a dot happens on a faster scale than the overall evolution and that because of this reason it is a very good property of the adaptive time stepping scheme to make steps smaller exactly at these events. Note that the upper plateau at $dt_{max} = 0.1$ is prescribed, as we not updated as in (3.7), but added the bound, so that the update is in fact $dt \leftarrow \min(\nu \frac{\epsilon}{R} dt, dt_{max})$. We found that the behavior of the method relies quite strongly on the size of the domain, the way the residual norm is defined and ϵ. So far we are not aware of optimal choices, but when working, the results are convincing. The global error decay of the discussed methods in Section 5 was of the order dt^2 for typical step sizes. We expect a similar decrease as with the SBDF2 method, $\mathcal{O}(dt^2)$, too, even with fixed step size, hence we think that this kind of adaptivity relieves the computational costs significantly.

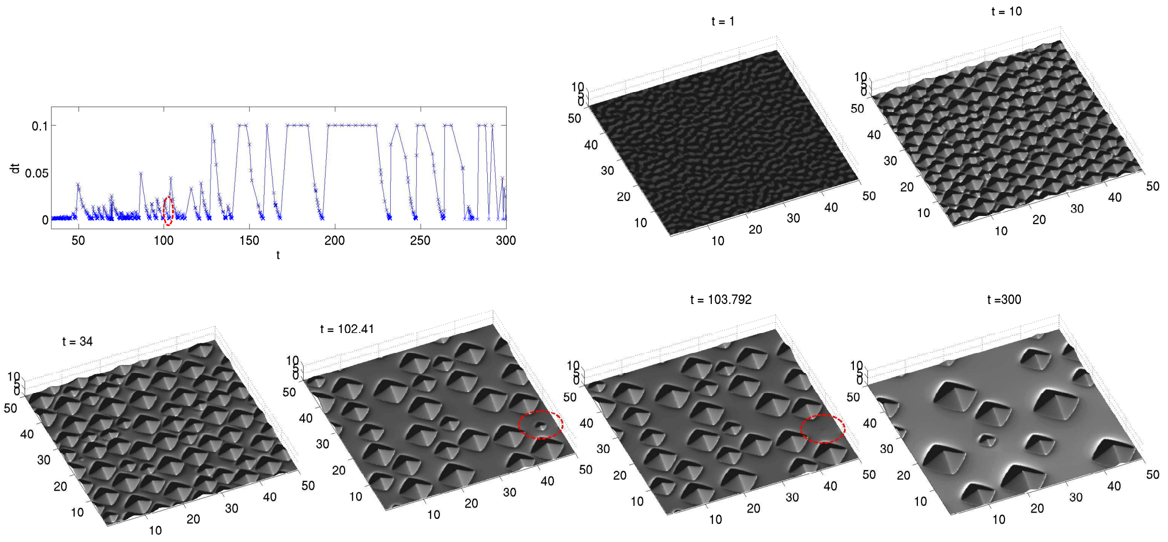

Figure 8: Temporal evolution of PDE (2.1) simulated with the SBDF1/SBDF1-2-step method from Section 3.2. The signal to the left depicts the changes in the time step size, the surrounding figures plot the state $h(x, y, t)$ at different time points. The dashed ellipses indicate one coarsening event.

6 Speeding up numerical simulations on a GPU

We begin with a section discussing GPU computing related to PDEs in general and the possibilities to work with Matlab. Thereafter, we report on our GPU approach for solving PDE (2.1) faster by exploiting the free GPUmat package. The runs are compared to Matlab parallel computing toolbox (PCT) based computations and an implementation of GPUmat in Octave.

6.1 PDEs and GPU computing

For a long time computations on GPUs have been restricted to the purpose they were invented for: accelerated graphic computations. On the verge of the new millennium people started to use so-called programmable shaders on GPUs, which provided just very basic programmable features and often needed to be programmed in low-level languages. This changed dramatically with the introduction of NVIDIA's CUDA architecture and non-graphical (general purpose GPU - GPGPU) applications gained momentum. GPUs evolve into high performance, highly parallel processing units, and the new generations support double arithmetic and allow for accurate scientific calculations. As also the philosophy of CPU innovation changed, work on parallel architectures will further gain significance in future.

For PDE applications one can find several publications that report on an order of magnitude gain in computational speed, e.g. for wave propagation [25], for groundwater flow simulation [16], for a Navier-Stokes solver [4] or the simulation of a Ginzburg-Landau equation [14]. These few works concerning simulations of PDEs on graphical units have one thing in common: the authors use CUDA and try to exploit the GPUs structure to gain the best benefits. Recently it has been reported that PDEs like the wave-equation can be solved 20-60 times faster when exploiting a GPU by programming in CUDA [25], which is a great improvement in runtime. On the down-side this means that the user needs to know quite something about graphics hardware, a new programming language that is bound to NVIDIA and time for carrying out the implementations. Another approach is the OpenCL language, which allows, in theory, to write a parallel program once and use it on different types of hardware, i.e. GPUs, CPUs, and others. However, OpenCL requires substantial programming efforts and we think many researchers that are more concerned with PDEs than with computer science would prefer if the implementation can be done in a simple language like Matlab instead of CUDA or OpenCL without having to cope with the details of specific hardware. Then one will not harvest the full fruits of parallel computing acceleration, but with little effort one has at least a significant speedup.

There are three main packages that allow GPU calculations with the MathWorks software; the free GPUmat package (http://gp-you.org), the in-house parallel computation toolbox (PCT) [7] that allows for GPU computing since 2010, or the well-performing Jacket package (http://www.accelereyes.com), see [35] for a comparison of the packages. Although they exist for some time now, there seems to be only one reported result for accelerating PDE computations on GPUs on the Matlab platform [31] and no results for pseudospectral methods at all. The lack of the possibility to carry out sparse calculations in PCT and GPUmat might be one of the major reasons and in GPUmat also the solving of dense linear systems is not implemented. However, with our approaches to approximate solutions to PDE (2.1) one does not rely on solutions to any equation. Our scheme works with basic arithmetic operations and the FFT. Because of this simplicity the adaptation of the scheme to a GPU becomes trivial. We are able to report on up to a five time increase in computational speed by incorporating the GPUmat package, which is great due to evanescent efforts and costs the user has to invest once the package is installed. One only needs a recent Matlab version and an up-to-date low-cost NVIDIA GPU that has sufficient memory to cope with the user's matrices. Finally, we ported the GPUmat package to GNU Octave. This allows to make all the parallel GPU computations, i.e. the implementation of our Fourier collocation schemes, on a completely free platform.

6.2 Solving PDE (2.1) on a GPU

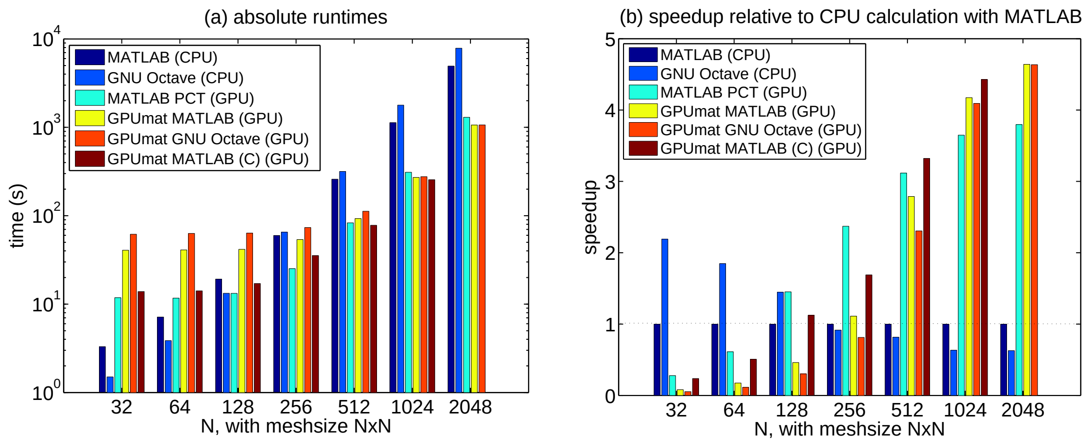

Figure 9: (a) Absolute runtimes (in seconds, doubly logarithmic scale) of quantum dot simulation computations on the GPU and CPU with different configurations for seven increasing values of N for an overall $N \times N$ spatial grid; (b) speedup relative to MATLAB code without GPU acceleration. The compiled program was unable to run for $N = 2048$ due to memory restrictions.

We solved PDE (2.1) as described earlier with the SBDF1 update that does not rely on the solution of systems of equations due to the explicit form the update is given in on CPU and GPU based configurations. For larger problems we measured an up to 5-fold speedup when exploiting the parallel structure of GPUs. Everyone who owns a good GPU and MATLAB can achieve similar improvement with the adjustments of the MATLAB code we describe here. Our results rely on the hard- and software described in Table 1.

Unit	GPU	CPU
Name	NVIDIA GeForce GTX 260	Intel(R) Core(TM) i7 CPU 860
Memory	896 MB	4096 MB

Table 1: The hard- and software under consideration. We use CUDA 4.2 with driver 310.32 on SUSE Linux 12.2 in AMD64 modus

An example source code for the SBDF1 method exploiting the GPUmat speedup is given in Appendix B. Figure 9 shows a study for the runtimes needed for the quantum dot simulations with this type of code. We averaged the elapsed times over three runs that turned out to have negligible runtime differences. We used the step size $dt = 0.01$, the time interval length $T = 50$ and different number of grid points. The x-axes denote N, for the overall matrix we have $N_{total} = N^2$ entries. In (a) we see absolute runtimes for three CPU configurations, MATLAB , GNU Octave and a multithreading developer version of GNU Octave, and four GPU runs with help of the PCT, GPUmat in MATLAB , GPUmat in GNU Octave and GPUmat in MATLAB with compilation of the main computation part. Figure 9 (b) shows the extent of this acceleration by plotting the quotient of the MATLAB CPU runtime against the other runtimes for the same N, resulting in a dimensionless speedup quantity. The compiled GPUmat and PCT versions start to be faster for problem sizes of 128×128 and the non-compiled GPUmat programs somewhat later at about 256×256. This difference can be explained by a dominant communication overhead in GPUmat for small problems, that is diminished by the use of a compiled version. The factor becomes negligible for larger grid sizes. In compiled mode memory is not freed until execution returns to MATLAB, therefore our graphics card had insufficient memory for a grid size of 2048×2048 in compiled mode. The PCT seems to use a similar technique, but hides it from the user and the memory management is more efficient as the biggest case performs without problems. Octave is superior for very small problem sizes, but is increasingly getting worse for larger N. The reason for Octave's speedup decrease is the lag of support for multithreading FFT calculations in its current stable branch as used in MATLAB . A developer version of Octave already implements this option and shows no such decrease, but performs worse for small N unless the multithreading of FFT calculations is turned off again. Especially for larger problems, i.e. 2048×2048, the observed runtime benefits are useful as one gains nearly a speedup of five. As runs on larger domains and time intervals can last for many hours, this is a significant speedup.

We compared the GPU/CPU quotients with those achieved for pure two-dimensional FFT calculations of random matrices of the same sizes as the discretizations and averaged these calculations over 10 FFT2 calls. Figure 10 shows these speedups and an increasing correlation between the FFT times and our simulation times for increasing values of N. We can conclude as expected, namely that the achievable runtime benefit is as good as the speedup of FFT evaluations, since this is the computation of dominant order in our program. By using a more advanced GPU such as the Tesla C2050 we hence expect a more dramatic picture. For the matrix size 8192×8192 MathWorks cites a 30fold speedup for the two-dimensional FFT in comparison to calculations on a quad-core Intel CPU. As the speedup of the FFT is directly linked to the speedup of the quantum dot simulation, we anticipate this kind of runtime improvement by using high end GPUs. However, we have shown that also on simple machines, easy to implement GPU based MATLAB and GNU Octave codes give a nice runtime improvement. Table 2 shows the requirements for working with the GPUmat package.

- **NVIDIA GPU with double support**
- **MATLAB R2008a or newer version**
- **GPUmat (from `http://gp-you.org`)**
- **CUDA (from `http://developer.nvidia.com/cuda/cuda-downloads`)**

Table 2: Requirements to run spectral IMEX schemes on a GPU with GPUmat on standard PC workstations.

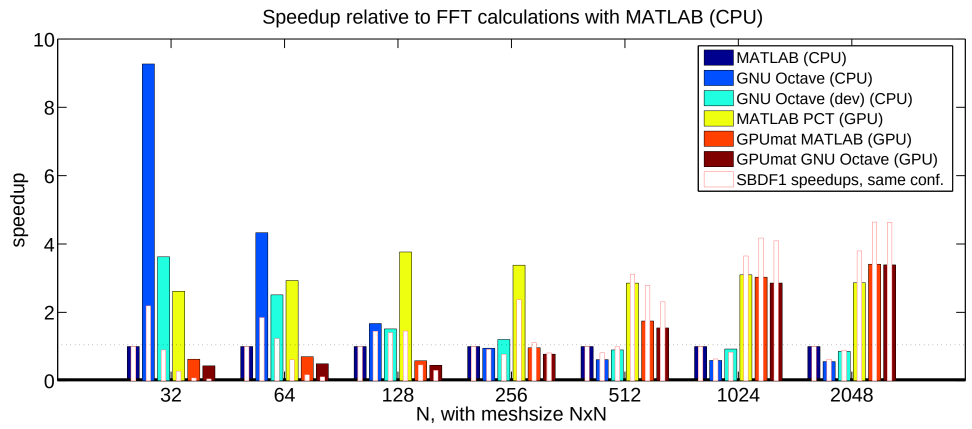

Figure 10: Speedups achieved for FFT and inverse FFT computations on the GPU for seven increasing values of N for an overall $N \times N$ grids, by using the different configurations. The SBDF1 quantum dot simulation speedups – as in Figure 9 (b) – shows that it correlates stronger with the FFT calculation times for larger values of N.

7 Conclusions and discussion

We have shown that fast and accurate simulations of the self-assembly of quantum dots, based on the small-slope approximation of a surface diffusion model, are possible in terms of Fourier collocation methods exploiting different time-stepping procedures. We want to stress that no supercomputers are needed to carry out calculations for domains of relevance, i.e. one succeeds with standard PC workstations. This is possible due to the reduction of the full problem from three to two dimensions, an efficient spatial approximation in terms of trigonometric interpolants – exploiting spectral accuracy –, efficient time-stepping and GPU acceleration. For the latter we used the free GPUmat MATLAB package for calculations on the parallel processors of the graphical unit and compared the runs with MATLAB's PCT and runs with Octave. As there appear new possibilities to work with MATLAB on GPUs, we believe that in near future more studies will be carried out on this platform. Up to now there is no publication we are aware of, where evolution equations are solved on a GPU in MATLAB or in GNU Octave. The new options of MATLAB's PCT allow for this kind of improvement. The cost for the package may be an obstacle that detains researchers on PDEs from working with it, more importantly for the FEM (and related) community is the fact that sparse computations are not yet implemented on this platform. The GPUmat package, however, is a competitive alternative to the PCT. For small mesh-sizes CPU simulations, both in Octave and MATLAB, have advantages compared to GPU based calculations. The picture changes for intermediate sizes, where the PCT and the compiled GPUmat configurations were fastest, though the code adjustments to work with the PCT are marginal. For large scale simulations GPUmat in GNU Octave or MATLAB showed similar superior performance. Overall we suggest to work on GPUs when working with FFTbased methods and sufficiently large grids. As GPUs and parallel packages evolve, we think that the benefits will be larger in future.

Trigonometric interpolation that generically imposes periodicity is a very good choice for the spatial differentiation in the presented problem setting. We have shown that many time-stepping schemes are possible, but not all of them are preferable. In general we found the high order SBDF schemes not very reliable as they tend to be unstable. Although showing a good asymptotic decay, an application for large scaled simulations seems unrealistic due to the severe step size restrictions. The extension of the SBDF1 scheme to an adaptive 2-step method leads to accuracy due to tracking of the topological changes of the evolving surface and corresponding adaption of the step size. The stability properties of this scheme are good and can be improved further by employing the convexity splitting approach, however, we have shown that the error control seems necessary when using this kind of stabilization. To answer quantitative questions, exact coarsening rates and time intervals where the ripening takes place, one of the more accurate exponential integrator schemes may be preferable. Our experience with the ETDRK4 scheme shows that for our problems it is a good alternative for calculation of highly accurate solutions while still maintaining reasonable computational times. The EXPINT package showed that it is not only stable and accurate, but that also the absolute computational time needed for a certain accuracy is small. Our tests on the dimension-reduced equation with the EXPINT package suggests that most of the exponential integrators suit well, in particular the Strehmel and Weiner method worked even better in the tests than the ETD4RK scheme.

Acknowledgment

MK's work was partly supported by the DFG Research Center Matheon, project C10. MK and TA gratefully acknowledge the support by the Federal Ministry of Education (BMBF) and the state government of Berlin (SENBWF) in the framework of the program "Spitzenforschung und Innovation in den Neuen Ländern" (Grant Number 03IS2151). Furthermore MK thanks L. N. Trefethen and A. Münch for motivating to carry out parts of this work.

A (The ETD4RK update)

Here we present the ETD4RK4 update derived by Cox and Matthews [8]. It reads

$$\begin{aligned} u_{n+1} = e^{\mathrm{L}dt}u_n + \alpha_1\mathrm{N}(u_n,t_n) + \alpha_2(\mathrm{N}(a_n,t_n+dt/2) + \mathrm{N}(b_n,t_n+dt/2)) \\ + \alpha_3\mathrm{N}(c_n,t_n+dt) \end{aligned} \tag{A.1}$$

with the variable coefficients

$$\begin{aligned} a_n &= e^{\mathrm{L}\frac{dt}{2}}u_n + \mathrm{L}^{-1}(e^{\mathrm{L}\frac{dt}{2}} - I)\mathrm{N}(u_n,t_n)\,, \\ b_n &= e^{\mathrm{L}\frac{dt}{2}}u_n + \mathrm{L}^{-1}(e^{\mathrm{L}\frac{dt}{2}} - I)\mathrm{N}(a_n,t_n+\frac{dt}{2})\,, \\ c_n &= e^{\mathrm{L}\frac{dt}{2}}a_n + \mathrm{L}^{-1}(e^{\mathrm{L}\frac{dt}{2}} - I)(2\mathrm{N}(b_n,t_n+\frac{dt}{2}) - \mathrm{N}(u_n,t_n))\,, \end{aligned}$$

and the constants

$$\begin{aligned} \alpha_1 &= dt^{-2}\mathrm{L}^{-3}[-4 - \mathrm{L}dt + e^{\mathrm{L}dt}(4 - 3\mathrm{L}dt + (\mathrm{L}dt)^2)]\,, \\ \alpha_2 &= 2dt^{-2}\mathrm{L}^{-3}[2 + \mathrm{L}dt + e^{\mathrm{L}dt}(-2 + \mathrm{L}dt)]\,, \\ \alpha_3 &= dt^{-2}\mathrm{L}^{-3}[-4 - 3\mathrm{L}dt - (\mathrm{L}dt)^2 + e^{\mathrm{L}dt}(4 - \mathrm{L}dt)]\,, \end{aligned}$$

that can be calculated before the actual time-integration is carried out, if the step-size *dt* is kept constant. This scheme is known to be cancellation error prone when L has eigenvalues close to zero. We use Kassam and Trefethen's [18, 19] approach to this problem and we adapted the MATLAB file given in the first reference for our problem equation.

B (MATLAB code for GPUmat SBDF1 simulation)

Here we post the MATLAB code which was used for the simulation with the constant time-stepping SBDF1 method on a GPU using the GPUmat package. The actual changes that have to be made to carry out the simulations on the GPU are in bold format. For using MATLAB's PCT an equivalent adjustment with slightly changed syntax is necessary. The following snippet includes graphical output.

```
function qd_gpu
% Simulation of quantum dot self-assembly, SBDF1 on GPU
G = 0.15; E = 1.2778;r = 0.05; flux = 0.002; % parameters
n = 1024;                    % n number of coefficients in one dimension
L = 250;                     % domain length in one dimension
dx = L/n;                          % grid spacing
xgrid = dx*(1:n)';                 % 1D spatial grid
[X, Y] = meshgrid(xgrid, xgrid);   % 2D spatial grid
k=[0:n/2 -n/2+1:-1]'*(2*pi/L);     % 1D wavenumbers
[kxT, kyT] = meshgrid(k,k);        % 2D wavenumbers

% wavenumber matrices (double precision) on GPU
kx = GPUdouble(kxT);
ky = GPUdouble(kyT);
k_2 = kx.^2 + ky.^2;
k31 = (k_2).*kx;
k32 = (k_2).*ky;
k4 = k_2.^2;
fluxmat_fft = fft2(-flux*ones(n,n,GPUdouble));

t=0; t_max = 100; dt = 0.001; % initial time, maximal time, step size
C = 2; % convexity splitting parameter
S = [1 - dt*(E*(k_2.^(3/2)) - (1+C)*k4)]; %linear part
H = GPUdouble(h_init_2D(xgrid));          %GPU array definition
u = fft2(H);                              %DFT

while t < t_max
    t = t+dt;
    H = real(ifft2(u));
    % visualization every few iterations
    if mod(round(t/dt),50) == 0
        figure(4);
        surfl(X,Y,double(H));
        colormap gray
        shading interp
        camlight right
        axis([xgrid(1) xgrid(end) xgrid(1) xgrid(end) -0.1 10]);
        view([-23.5 84]); drawnow;
    end
    % gradient of h, explicit nonlinear parts, update
    hx = real(ifft2(1i*kx.*u));
    hy = real(ifft2(1i*ky.*u));
    gx = 4*G*(hx.^3 - hx);
    gy = 4*G*(hy.^3 - hy);
    u_plus = (u - dt*(-C*k4.*u - r*k_2.*fft2(1./(H.^2)) - ...
        1i*(k31.*fft2(gx) + k32.*fft2(gy)) + fluxmat_fft))./S;
    u = u_plus;
end

function y = h_init_2D(x)
y = (rand(length(x), length(x))-rand(length(x),length(x)))*0.001 + 0.5;
```

Code 1: MATLAB code for the SBDF1 method with convexity splitting ($C = c_2$) with calculations on a GPU using GPUmat and with visualization. The differences to a CPU implementation are accentuated by the bold format of the GPU commands.

C (MATLAB snippets for EXPINT environment)

To simulate equation (5.1) in the EXPINT environment, we had to implement the following two MATLAB snippets for the linear, and nonlinear part, respectively. These are adaptions from the other problems given within the package.

```
problem.domlength = problem.ND/4;
problem.dx = problem.domlength/problem.ND;
problem.x = problem.dx*(1:problem.ND)';% N-1 inner points
% problem parameters
problem.gam = 0.05; problem.E = 1.28; problem.cvxty = .3;
% linear part L
problem.k = [0:problem.ND/2-1 0 -problem.ND/2+1:-1]'/(problem.domlength/(2*pi));
problem.k2 = problem.k.^2; problem.k3 = problem.k.^3; problem.k4 = problem.k.^4;
problem.L = problem.E*problem.k2.*abs(problem.k) ...
            - problem.k4 - problem.cvxty*problem.k4;

% Initial condition
IC = s.IC;
switch lower(IC);
    % add other options such as randomly perturbed initial state here
    case {'loaded'}
      H = load('init_forETD.dat');
      problem.y0 = H;
      problem.ICname = 'loaded';
      problem.ICnametex = 'perturbed flat state';
    otherwise
      error('koreva:invalidic', 'Unknown IC supplied');
end
problem.y0 = fft(problem.y0);

% Other problem parts
% nonlinear function, postprocessing, L+N function for ode15s, names
problem.N = 'koreva_N';
problem.postprocessing = 'koreva_post';
problem.LplusN = 'diagLplusN';
problem.problemname = ['Korzec-Evans, ND=', int2str(problem.ND), ...
                       ', IC: ', problem.ICname];
problem.problemnametex = ['Korzec-Evans, ', ...
                          '$\mathrm{ND}=', int2str(problem.ND), ...
                          '$, IC: $', problem.ICnametex, '$'];
```

Code 2: Main parts of the MATLAB code for setting up the problem structure for simulating equation (5.1) in the EXPINT environment. The linear part L is defined by the wavenumbers, corresponding to (4.6).

```
function Nr = koreva_N(U, t, problem)
h = real(ifft(U));
hx = real(ifft(1i*problem.k.*U));
Nr = problem.gam*problem.k2.*fft(1./(h.^2)) ...
    + 0.6*1i*problem.k3.*fft(hx.^3-hx) + problem.cvxty*problem.k4.*U;
```

Code 3: MATLAB code for the nonlinear function (4.7).

References

[1] R. J. Asaro and W. A. Tiller. Interface morphology development during stress corrosion cracking. II. Via volume diffusion. *Acta Metall.*, 23:341–344, 1975.

[2] U. M. Ascher, S. J. Ruuth, and B. Wetton. Implicit-explicit methods for time-dependent partial differential equations. *SIAM J. Numer. Anal.*, 32(2):797–823, 1995.

[3] H. Berland, B. Skaflestad, and W. M. Wright. EXPINT - A MATLAB Package for Exponential Integrators. *ACM Trans. Math. Soft.*, 33(1), 2007.

[4] T. Brandvik and G. Pullan. An accelerated 3D Navier-Stokes solver for flows in turbomachines. *Proc. GT2009*, 2009.

[5] C. G. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral Methods: Fundamentals in Single Domains.* Springer, 2010.

[6] C.-H. Chiu and Z. Huang. Common features of nanostructure formation induced by the surface undulation on the Stranski-Krastanow systems. *Appl. Phys. Lett.*, 89:171904, 2006.

[7] R. Choy and A. Edelman. Parallel Matlab: Doing it right. *Proc. IEEE*, 93(2):331–341, 2005.

[8] S. M. Cox and P. C. Matthews. Exponential Time Differencing for Stiff Systems. *J. Comp. Phys.*, 176:430–455, 2002.

[9] J. Drucker. Self-Assembling Ge(Si)/Si(001) Quantum Dots. *IEEE J. Quantum Electronics*, 38(8):975–987, 2002.

[10] D. J. Eyre. An unconditionally stable one-step scheme for gradient systems. *unpublished*, 1998.

[11] A. Friedli. *Verallgemeinerte Runge-Kutta Verfahren zur Lösung steifer Differentialgleichungssysteme.* Numerical Treatment of Differential Equations, R. Burlirsch, R. Grig- orie and J. Schröder, eds., vol. 631 of Lecture Notes in Mathematics, Springer, Berlin, 1978.

[12] A. A. Golovin, S. H. Davis, and P. W. Voorhees. Self-organization of quantum dots in epitaxially strained solid films. *Phys. Rev. E*, 68:056203, 2003.

[13] I. Grooms and K. Julien. Linearly implicit methods for nonlinear PDEs with linear dispersion and dissipation. *J. Comp. Phys.*, 230:3630–3650, 2011.

[14] K. A. Hawick and D. P. Playne. Numerical Simulation of the Complex Ginzburg-Landau Equation on GPUs with CUDA, 2010.

[15] M. Hochbruck and A. Ostermann. Explicit exponential Runge-Kutta methods for semilinear parabolic problems. *SIAM J. Num. Anal.*, 43(3):1069–1090, 2005.

[16] X. Ji, T. Cheng, and Q. Wang. CUDA-based solver for large-scale groundwater flow simulation. *Eng. Comp.*, 28:13–19, 2012.

[17] W. Jiang, W. Bao, C. V. Thompson, and D. J. Srolovitz. Phase field approach for simulating solid-state dewetting problems. *Acta Mat.*, 60:5578–5592, 2012.

[18] A.-K. Kassam. Solving reaction-diffusion equations 10 times faster. *Oxford Numerical Analysis Group Research Report: Technical report NA 03/16*, 2003.

[19] A.-K. Kassam and L. N. Trefethen. Fourth-order time-stepping for stiff PDEs. *SIAM J. Sci. Comput.*, 26(4):1214–1233, 2005.

[20] M. D. Korzec and P. L. Evans. From bell shapes to pyramids: A reduced continuum model for self-assembled quantum dot growth. *Physica D*, 239:465–474, 2010.

[21] M. D. Korzec, A. Münch, and B. Wagner. Anisotropic surface energy formulations and their effect on stability of a growing thin film. *IFB*, 14(4), 2012.

[22] S. Krogstad. Generalized integrating factor methods for stiff PDEs. *J. Comp. Phys.*, 203:72–88, 2005.

[23] J. D. Lawson. Generalized Runge-Kutta Processes for Stable Systems with Large Lipschitz Constants. *SIAM J. Num. Anal.*, 4(3):372–380, 1967.

[24] A. Martí, N. López, E. Antolín, E. Cánovas, C. Stanley, C. Farmer, L. Cuadra, and A. Luque. Novel semiconductor solar cell structures: The quantum dot intermediate band solar cell. *Thin Solid Films*, 511-512:638–644, 2006.

[25] D. Michéa and D. Komatitsch. Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards. *Geophys. J. Int.*, 182:389–402, 2010.

[26] S. Nakamura, G. Fasol, and S.J. Pearton. *The Blue Laser Diode: The Complete Story.* Springer, 2000.

[27] F. M. Ross, J. Tersoff, and R. M. Tromp. Coarsening of Self-Assembled Ge Quantum Dots on Si(001). *Phys. Rev. Lett.*, 80(5):984–987, 1998.

[28] C. Runge. Über die numerische Auflösung von Differentialgleichungen. *Math. Ann.*, 46(22):167–178, 1895.

[29] C.-B. Schönlieb and A. Bertozzi. Unconditionally stable schemes for higher order inpainting. *Comm. Math. Sci.*, 9(2):413–457, 2011.

[30] V. A. Shchukin and Dieter Bimberg. Spontaneous ordering of nanostructures on crystal surfaces. *Rev. Modern Phys.*, 71(4):1125–1171, 1999.

[31] C.-Y. Shei, P. Ratnalikar, and A. Chauhan. Automating GPU computing in Matlab . In *Proc. Int. Conf. Supercomp.*, ICS '11, pages 245–254. ACM, 2011.

[32] K. Strehmel and R. Weiner. *Linear-implizite Runge-Kutta Methoden und ihre Anwendungen.* Teubner, 1992.

[33] W. T. Tekalign and B. J. Spencer. Evolution equation for a thin epitaxial film on a deformable substrate. *J. Appl. Phys.*, 96(10):5505–5512, 2004.

[34] B. P. Vollmayr-Lee and A. D. Rutenberg. Fast and accurate coarsening simulation with an unconditionally stable time step. *Phys. Rev. E*, 68(066703), 2003.

[35] B. Zhang, S. Xu, F. Zhang, Y. Bi, and L. Huang. Accelerating MatLab code using GPU: A review of tools and strategies. In *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011*, pages 1875–1878, 2011.