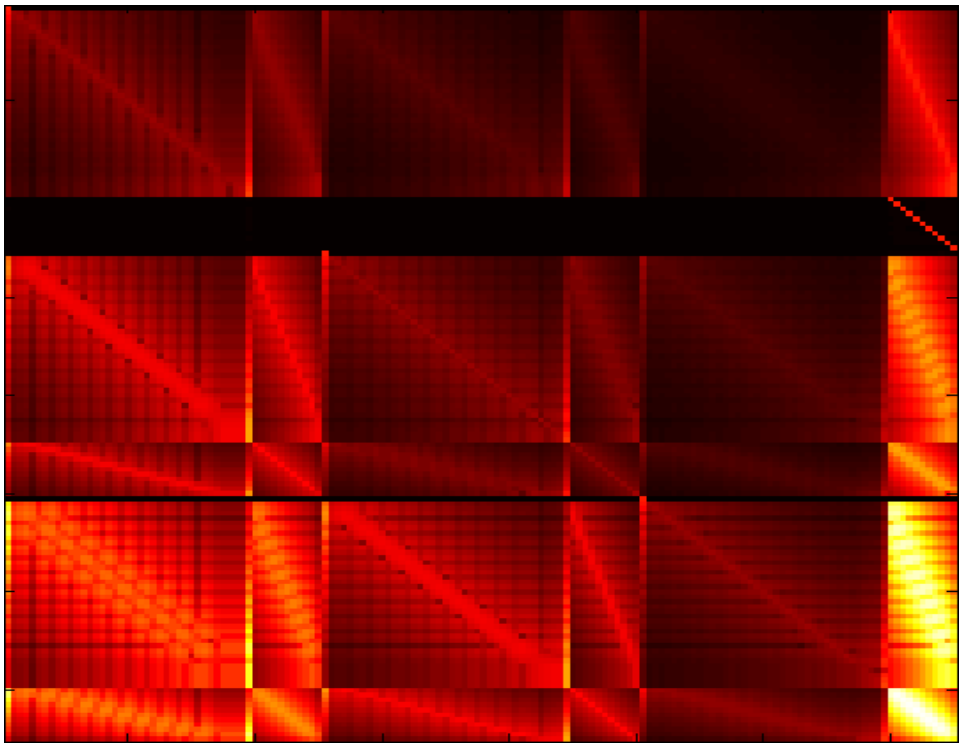
rijksuniversiteit groningen

faculteit Wiskunde en Natuurwetenschappen

Elimination based hybrid methods for solving time-dependent PDEs.



Bachelorondersoek Wiskunde

December 2011

Student: F.J. Koerts

Eerste Begeleider: dr. ir. F.W. Wubs

Tweede Begeleider: dr. K. Camlibel

Contents

1 Introduction

This thesis can be seen as a preliminary investigation on numerically solving time-dependent non-linear PDEs based upon elimination . With elimination we mean the process of decomposing a grid into two parts such that the state on one of these parts is substituted into the state of the other part. Elimination enables us to compute the state on the latter subgrid independent of the first one, where we might use different solving techniques. Here we will consider only the one-dimensional case, whereas in most practical situations, a higher dimensional model is required. This gives rise to a proposal for continued investigation on this topic, which can be found in the section 'conclusion and discussion'. In its turn, this topic can also been seen as a continued investigation on Jolanda Heijnen's thesis [1]. In that thesis, the (linear) heat equation is solved using an integration-(quasi) stationary hybrid method. In several ways we try to obtain results for more general cases.

We give an example of a case where elimination on higher dimensional models is useful. Consider a 2-dimensional grid that is decomposed into some non-overlapping subdomains and separator groups (1-dimensional boundaries between subdomains). See figure 1. The state on the subdomains can be eliminated exactly, hereby making use of independent processors. The required time to solve the whole system is highly dependent of interaction between the subdomains, which takes place via the separators. The unknowns on the separators are connected to all other separators in other domains (the matrix is irreducible). This requires that information from one processor has to go to every other processor. By transforming the problem to an explicit approach we can form a block diagonal matrix so that only communication with neigbouring nodes are needed. For more information we refer to [4].

This higher-dimensional case is beyond the scope of this thesis. Here I consider a 1-dimensional grid which is divided into two subgrids which we denote by a coarse and a fine subgrid. See figure 2. The fine subgrid consists of iso-

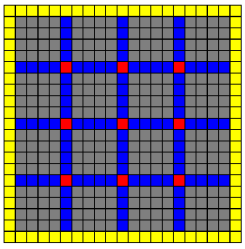


Figure 1: the areas in grey correspond to subdomains which we want to compute independent of each other. Separator nodes are marked in blue.

Figure 2: 1-dimensional grid where the coarse (red) subgrid divides the fine (blue) subgrid into subdomains which can be computed independent of each other

lated nodes, which divides the fine subgrid into a number of subdomains. The variables of each group of the fine subgrid can be eliminated and computed in a parallel way, using an implicit integration method or a (quasi) stationary method. After elimination we have a reduced system of ODEs where the variables belong to the coarse subgrid. If we employ an explicit integration method on that system, we only need communication with neighbouring nodes to perform one time step. Thus we obtain two hybrid methods: implicit-explicit (integration) methods and integration-(quasi) stationary methods. In this thesis, special attention is drawn for this integration-(quasi) stationary hybrid method.

A (quasi) stationary solution is the solution which is obtained by setting some order derivative equal to zero. Hence, if a (quasi) stationary method is applied to some subgrid, no time integration is needed. An advantage is that the state can be computed in one iteration process on the corresponding subgrid(s). However, the method can only be applied in situations which vary little in time (i.e. which is smooth in time) and it will lead to an approximation of the original system. A sufficient condition for the state to be computable, is that the state on the boundary nodes is known, including its derivatives. Hence, if an integration-(quasi) stationary method is used, in each time step the state on the subgrid to which explicit integration is applied, is computed first. Then the boundary state of each group is known and the state on the other subgrid can be computed.

An example of a non-linear partial differential equation (PDE) which I consider is the Burgers' equation, which appears mainly in modeling of fluid dynamics. In many applications, a (quasi) stationary solution is a good approximation of the real solution. For example, if we want to simulate the Earth's tide, which is a phenomenon caused by the moon, we can use such a stationary solution since the frequency of about 12 hours is relatively low. Another example can be found in modeling of air dynamics around the rotor of a helicopter with constant rotation speed. The Burgers' equation is enclosed in the Navier-Stokes

equations, which gives in this case a periodic solution too.

Research questions include stability and accuracy of some (combination of) methods which are described above. In doing this, we will investigate their relation with the grid partitioning, the time step, (in the case of (quasi) stationary (hybrid) methods:) the order of derivative which is set to zero, and the frequency of a periodic Dirichlet boundary condition. This will be investigated for full integration methods (where we use either one integration method, or a hybrid implicit-explicit method), the full (quasi) stationary method and hybrid integration-(quasi) stationary methods as described above.

2 Definition of the problem

We consider a second order non-linear PDE, discretized in space of the form:

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u}(t) + \mathbf{r}(\mathbf{u}(t)) + \mathbf{f}(t) \tag{1}$$

Here, $r(\mathbf{u})$ is some non-linear term, which we assume to be of a bilinear form $\mathbf{r}(\mathbf{u}) = \langle \mathbf{u}, \mathbf{u} \rangle$, where $\langle -, - \rangle$ denotes an inner product. In this thesis, we will specify it as a quadratic form: $\mathbf{r}(\mathbf{u}) = diag(\mathbf{u})\mathbf{B}\mathbf{u}$.

Integration methods for solving non-linear PDEs of the form (1), can be classified into explicit and implicit methods. Indeed, both types come with their own advantages and disadvantages. A disadvantage of explicit methods like the forward Euler method, is the small time step which is required to guarantee stability. On the other hand, implicit methods like the Backward Euler method require bigger computational costs per iteration. However, they generally allow a much bigger time step than explicit methods do. In this thesis I consider another method for solving (1), namely a combination of an implicit and an explicit method. The idea is to partition the grid into two sets of grid points, which I will call the subgrids. Now, the state on one of these sets of points will be used explicitly to solve (1), while the state on the other will be used to implicitly solve the PDE. Thus we try to combine the advantages of implicit methods (bigger stability) and explicit methods (less computational costs).

It is not only integration methods which we will investigate. By making the assumption that the k-th order derivative of the state $\mathbf{u}^{(k)} = 0$, for some k and state $\mathbf{u}$ on one or both of subgrids, integration in time is not needed. These assumptions, (quasi) stationary assumptions, can be applied to solutions which vary little in time and will lead to an approximation of the original system. In that case, the state $\mathbf{u}$ can be computed in one iteration process on the corresponding subgrid(s).

Unfortunately, it was not always possible to draw conclusions about the general

equation (1). In these cases, we tried to find statements for a special case, the Burgers' equation, which reads:

$$\frac{d\mathbf{u}}{dt} = \mu \mathbf{u}_{xx}(t) - \mathbf{u}(t)\mathbf{u}_x(t) \tag{2}$$

The PDE (1) can be described inside a system of equations which contains also the integration method. Once the state is known on time step j, this system is solvable for the state on a given time step $j+1$ (multi-step methods leaved out of consideration). For example, if we take the forward Euler method, the following system arises:

$$\begin{cases} \mathbf{v}^j = \mathbf{A}\mathbf{u}^j + \langle \mathbf{u}^j, \mathbf{u}^j \rangle + \mathbf{f}(t) \\ \mathbf{u}^{j+1} = \mathbf{u}^j + \Delta t \cdot \mathbf{v}^j \end{cases} \tag{3}$$

Here we have substituted $\mathbf{v} = \frac{d\mathbf{u}}{dt}$ and the exponential j refers to the approximation on the j'th time step. In this example it is clear that the system can be solved for $\mathbf{u}^{j+1}$ explicitly by first computing $\mathbf{v}^j$ from the 'PDE part' (first rows), and then substitute this in the 'method part' (bottom rows). If we would have used the backward Euler method, the system becomes:

$$\begin{cases} \mathbf{v}^{j+1} = \mathbf{A}\mathbf{u}^{j+1} + \langle \mathbf{u}^{j+1}, \mathbf{u}^{j+1} \rangle + \mathbf{f}(t + \Delta t) \\ \mathbf{u}^{j+1} = \mathbf{u}^j + \Delta t \cdot \mathbf{v}^{j+1} \end{cases}$$

Obviously, this system cannot be solved explicitly. In order to solve for $\mathbf{u}^{j+1}$, we can perform a number of Newton steps. This can be done after rewriting the systems such that the left-hand side becomes zero. For notational convenience, in the ongoing text we use $\mathbf{u} = \mathbf{u}^{j+1}$ and $\mathbf{v} = \mathbf{v}^{j+1}$. We now have:

$$\begin{cases} \mathbf{F}_1(\mathbf{y}) \equiv -\mathbf{v} + \mathbf{A}\mathbf{u} + \langle \mathbf{u}, \mathbf{u} \rangle + \mathbf{f}(t + \Delta t) \\ \mathbf{F}_2(\mathbf{y}) \equiv -\mathbf{u} + \mathbf{u}^j + \Delta t \cdot \mathbf{v} \end{cases}$$

Here, we have defined the system state $\mathbf{y}$ and, respectively, shall define the system $\mathbf{F}(\mathbf{y})$ as follows:

$$\mathbf{y} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}, \qquad \mathbf{F}(\mathbf{y}) = \begin{pmatrix} \mathbf{F}_1(\mathbf{y}) \\ \mathbf{F}_2(\mathbf{y}) \end{pmatrix}$$

and consider $\mathbf{u}^j$ and $\mathbf{v}^j$ to be constant. A Newton step then reads:

$$\mathbf{y}_1 = \mathbf{y}_0 + J_{\mathbf{F}}(\mathbf{y_0})^{-1}\mathbf{F}(\mathbf{y_0})$$

for some initial guess $\mathbf{y}_0$. Here, $J_{\mathbf{F}}(\mathbf{y_0})$ is the Jacobian matrix evaluated at $\mathbf{y}_0$. This matrix assumes the following form:

$$J_{\mathbf{F}}(\mathbf{y_0}) = \begin{pmatrix} \frac{d}{d\mathbf{u}}\mathbf{F}_1(\mathbf{y_0}) & \frac{d}{d\mathbf{v}}\mathbf{F}_1(\mathbf{y_0}) \\ \frac{d}{d\mathbf{u}}\mathbf{F}_2(\mathbf{y_0}) & \frac{d}{d\mathbf{v}}\mathbf{F}_2(\mathbf{y_0}) \end{pmatrix}$$

In general, with $\mathbf{y}_i$ we mean the approximation of the system state $\mathbf{y}$ yielded by the i'th Newton step. In the case we use a backward Euler method, and for initial guess $\mathbf{y}_0 = \mathbf{y}^j \equiv (\mathbf{u}^j, \mathbf{v}^j)^T$, we get:

$$J_{\mathbf{F}}(\mathbf{y^j}) = \begin{pmatrix} \mathbf{A} + g(\mathbf{B}, \mathbf{u}^j) & -\mathbf{I} \\ -\mathbf{I} & \Delta t \mathbf{I} \end{pmatrix}$$

where $g(\mathbf{B}, \mathbf{u}) = diag(\mathbf{Bu}) + diag(\mathbf{u})\mathbf{B}$.

If we use an explicit integration method instead, we obtain a Jacobian which gives rise to easier solving the system:

$$J_{\mathbf{F}}(\mathbf{y^j}) = \begin{pmatrix} \mathbf{A} + g(\mathbf{B}, \mathbf{u}^j) & -\mathbf{I} \\ -\mathbf{I} & 0 \end{pmatrix}$$

From this matrix it can also be seen that $\mathbf{v}$ can be easily computed from the integration method part (bottom part), once $\mathbf{u}$ is known.

Convergence of $\mathbf{y}_i$ to the system state $\mathbf{y}$ for $i \to \infty$, is dependent of the dominance of the non-linear term. For systems whose non-linear term is not dominant, i.e. a non-linear system whose linearization is locally a good approximation to the original one, convergence will be fast. Of more interest, is the question whether $\mathbf{u}$ is a good approximation of $\mathbf{u}(t + \Delta t)$, the real solution. Therefore, we ask ourselves which conditions have to be satisfied in order to guarantee stability.

So far, we varied the integration method inside system $\mathbf{F}$. Now we make an approximation on the original equation (1), so that the system is easier to solve. Instead of using $\mathbf{u}$ we will now use $\mathbf{u}^j$, making the PDE part explicit. In combination with the Backward Euler method, the following system appears:

$$\begin{cases} \mathbf{F}_1(\mathbf{y}) \equiv -\mathbf{v} + \mathbf{A}\mathbf{u}^j + \langle \mathbf{u}^j, \mathbf{u}^j \rangle + \mathbf{f}(t) \\ \mathbf{F}_2(\mathbf{y}) \equiv -\mathbf{u} + \mathbf{u}^j + \Delta t\ \mathbf{v} \end{cases}$$

Note that, with respect to $\mathbf{u}$, this system is equivalent to system (3). The Jacobian now assumes a much simpler form:

$$J_{\mathbf{F}}(\mathbf{y^j}) = \begin{pmatrix} 0 & -\mathbf{I} \\ -\mathbf{I} & \Delta t \mathbf{I} \end{pmatrix}$$

From this matrix it can be seen that $\mathbf{u}$ can be easily computed from the PDE part (upper part), once $\mathbf{v}$ is known. Of course, also a variant is possible where both the PDE part and the integration method are explicit. This combination is unfavorable, since it will generally give rise to less stable solutions, while the computational costs do not decrease.

3 (Quasi) stationary solutions

Solutions of the PDE (1) which change slowly in time, can also be approximated by a (quasi) stationary solution, i.e. a solution for which $\mathbf{u}^{(k)}(t) = 0$ for some k. In periodic solutions, this solution becomes generally a better approximation of the real solution if the period is larger, since then $|\mathbf{u}^j(t)|$ is less for all $j \in \mathbb{N}$.

In the previous section, we described a way to put equation (1) alongside the integration method in a system $\mathbf{F}$ from which the state on the $(j+1)$'th time step can be solved for. If we use a (quasi) stationary approximation instead, i.e. $\mathbf{u}^{(k)}(t) = 0$ for some k, we can also form a system which consists of the stationary assumption alone if $k = 1$. Note that this stationary assumption equals the right-hand side of the PDE:

$$\left\{ \ \mathbf{F}_1(\mathbf{y}) \equiv \mathbf{A}\mathbf{u} + \langle \mathbf{u}, \mathbf{u} \rangle + \mathbf{f}(t + \Delta t) \right.$$

If $k > 1$, then the system consists of the (derivatives of the) PDE together with the (quasi) stationary assumption, which is in fact the $(k-1)$st order derivative of the right-hand side of the PDE. The $k-1$ different derivatives of the PDE are needed to link $\mathbf{u}^{(k)}$ with the state $\mathbf{u}$. The following system then appears:

$$\left\{ \begin{array}{l} \mathbf{F}_1(\mathbf{y}) \equiv -\mathbf{U}_1 + \mathbf{A}\mathbf{U}_0 + \phi_0(\mathbf{B}, \mathbf{U}) + \mathbf{f}(t + \Delta t) \\ \vdots \\ \mathbf{F}_{j+1}(\mathbf{y}) \equiv -\mathbf{U}_{j+1} + \mathbf{A}\mathbf{U}_j + \phi_j(\mathbf{B}, \mathbf{U}) + \mathbf{f}^{(j)}(t + \Delta t) \\ \vdots \\ \mathbf{F}_{k-1}(\mathbf{y}) \equiv -\mathbf{U}_{k-1} + \mathbf{A}\mathbf{U}_{k-2} + \phi_{k-2}(\mathbf{B}, \mathbf{U}) + \mathbf{f}^{(k-2)}(t + \Delta t) \\ \mathbf{F}_k(\mathbf{y}) \equiv \mathbf{A}\mathbf{U}_{k-1} + \phi_{k-1}(\mathbf{B}, \mathbf{U}) + \mathbf{f}^{(k-1)}(t + \Delta t) \end{array} \right. \tag{4}$$

In this system, we have substituted $\mathbf{U}_j = \mathbf{u}^{(j)}$, which first must be treated as seperate variables for each j. All $\mathbf{U}_j$ are concatenated in the matrix $\mathbf{U} = (\mathbf{U}_0 \dots \mathbf{U}_k)$. We consider $\phi_j(\mathbf{B}, \mathbf{U})$ as the j-th derivative of $\langle \mathbf{u}, \mathbf{u} \rangle$ with respect to t. It is defined as follows:

$$\phi_j(\mathbf{B}, \mathbf{U}) = \sum_{i=0}^{j} \left(\begin{array}{c} j \\ i \end{array} \right) \langle \mathbf{U}_i, \mathbf{U}_{j-i} \rangle \tag{5}$$

The binomial coefficients can be explained as follows. For the derivative of $\langle \mathbf{u}^{(i)}, \mathbf{u}^{(j)} \rangle$ with respect to time, we have:

$$\frac{d}{dt} \langle \mathbf{u}^{(i)}, \mathbf{u}^{(j)} \rangle = \langle \mathbf{u}^{(i+1)}, \mathbf{u}^{(j)} \rangle + \langle \mathbf{u}^{(i)}, \mathbf{u}^{(j+1)} \rangle$$

It can be shown easily that the process of repeatedly making a derivative of $\langle \mathbf{u}, \mathbf{u} \rangle$, is similar to Pascal's triangle, each row representing the k'th order derivative and, except for the coefficients, each entry inside a row representing

one term of the right-hand side of (5). For each of these terms, a binomial coefficient therefore occurs.

In order to solve (4), we can again perform some Newton steps, where y is defined as:

$$\mathbf{y} = \begin{pmatrix} \mathbf{U}_0 \\ \mathbf{U}_1 \\ \vdots \\ \mathbf{U}_k \end{pmatrix}$$

Then, the Jacobian assumes the following form:

$$J_{\mathbf{F}}(\mathbf{y^j}) = \begin{pmatrix} \mathbf{A} + g_{0,0}(\mathbf{B}, \mathbf{U}_0) & -\mathbf{I} & 0 & \cdots & \\ g_{1,0}(\mathbf{B}, \mathbf{U}_1) & \mathbf{A} + g_{1,1}(\mathbf{B}, \mathbf{U}_0) & -\mathbf{I} & 0 & \cdots \\ \vdots & & \ddots & & \ddots \\ g_{k,0}(\mathbf{B}, \mathbf{U}_k) & g_{k,1}(\mathbf{B}, \mathbf{U}_{k-1}) & \cdots & & \mathbf{A} + g_{k,k}(\mathbf{B}, \mathbf{U}_0) \end{pmatrix}$$

where we have defined the function $g_{j,i}$ as:

$$g_{j,i}(\mathbf{B}, \mathbf{u}) = \begin{pmatrix} j \\ i \end{pmatrix} (diag(\mathbf{Bu}) + diag(\mathbf{u})\mathbf{B})$$

Note that, in notation of the previous section, $g = g_{0,0} = g_{1,0} = g_{1,1}$.

4 Use of subgrids

We will start with the given equation (1). The rows of this system can be rearranged in such a way that the rows corresponding to a certain subset of grid nodes appear on top, while the other rows appear on bottom. We refer to the first and latter subset of the grid nodes simply as the 'first subgrid' and the 'second subgrid', respectively. We thus get a system of the form:

$$\frac{d}{dt} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} + \begin{pmatrix} \mathbf{r}_1(\mathbf{u}) \\ \mathbf{r}_2(\mathbf{u}) \end{pmatrix} + \begin{pmatrix} \mathbf{f}_1(\mathbf{t}) \\ \mathbf{f}_2(\mathbf{t}) \end{pmatrix} \tag{6}$$

Here, $\mathbf{u}_1$ and $\mathbf{u}_2$ refers to the state on the first subgrid and the second subgrid, respectively. In the ongoing text, we assume that these vectors have length n_1 and n_2 respectively, such that the dimensions of $\mathbf{u}$ satisfy $n = n_1 + n_2$.

Now we have a system consisting of the PDE for the two subgrids, we can extend this system with an integration method or a (quasi) stationary approximation for both the first and second subgrid, such that the state can be computed. This gives rise to 4 possibilities. In order to separate different cases, we introduce the following notations: for $i \in \{1, 2\}$, if an integration method is used to compute

$\mathbf{u}_i$, then we define $\beta_i = 0$, otherwise $\beta_i = 1$, and k is the smallest integer for which we assume that $\mathbf{u}_i^{(k)} = 0$. If we restrict the system to be of size n ($k = 1$), we obtain the following system:

$$\begin{cases} \mathbf{F}_{1,1}(\mathbf{y}) \equiv \beta_1\big(\mathbf{A}_{11}\mathbf{u}_1 + \mathbf{A}_{12}\mathbf{u}_2 + \langle \mathbf{u}, \mathbf{u}\rangle_1 + \mathbf{f}_1(t+\Delta t)\big) + ... \\ \qquad\qquad (1-\beta_1)\varphi_1(\mathbf{u}_1, \mathbf{v}_1) \\ \mathbf{F}_{1,2}(\mathbf{y}) \equiv \beta_2\big(\mathbf{A}_{21}\mathbf{u}_1 + \mathbf{A}_{22}\mathbf{u}_2 + \langle \mathbf{u}, \mathbf{u}\rangle_2 + \mathbf{f}_2(t+\Delta t)\big) + ... \\ \qquad\qquad (1-\beta_2)\varphi_2(\mathbf{u}_2, \mathbf{v}_2) \\ \mathbf{F}_{2,1}(\mathbf{y}) \equiv -\mathbf{v}_1 + \mathbf{A}_{11}\mathbf{u}_1 + \mathbf{A}_{12}\mathbf{u}_2 + \langle \mathbf{u}, \mathbf{u}\rangle_1 + \mathbf{f}_1(t+\Delta t) \qquad (\beta_1 = 0) \\ \mathbf{F}_{2,2}(\mathbf{y}) \equiv -\mathbf{v}_2 + \mathbf{A}_{21}\mathbf{u}_1 + \mathbf{A}_{22}\mathbf{u}_2 + \langle \mathbf{u}, \mathbf{u}\rangle_2 + \mathbf{f}_2(t+\Delta t) \qquad (\beta_2 = 0) \end{cases}$$

Here, $\varphi_i(\mathbf{u}_i, \mathbf{v}_i), i \in \{1, 2\}$ describes a solving technique different from a (quasi) stationary assumption, e.g. an integration method. Since φ_i depends not only on the state $\mathbf{u}_i$, but also on the first order derivative $\mathbf{v}_i$, we have to add to the system a link between $\mathbf{u}_i$ and $\mathbf{v}_i$ in the case that $\beta_i = 0$. This link is given by the upper or bottom part of the rearranged PDE, which therefore appears in the bottom rows of the system if and only if this φ_i is used on the corresponding subgrid. Furthermore, $\langle \mathbf{u}, \mathbf{u}\rangle_i$ refers to the first ($i = 1$), or bottom ($i = 2$) n_i rows of $\langle \mathbf{u}, \mathbf{u}\rangle$, and more generally:

$$\langle \mathbf{p}, \mathbf{q}\rangle_i = (\mathbf{p}_i, \mathbf{B}_{i1}\mathbf{q}_1 + \mathbf{B}_{i2}\mathbf{q}_2)$$

We might use φ_i as an integration method such as the implicit theta method. This yields:

$$\varphi_i(\mathbf{u}_i, \mathbf{v}_i) = -\mathbf{u}_i + \mathbf{u}_i^j + \Delta t\,[\theta_i \mathbf{v}_i + (1-\theta_i)\mathbf{v}_i^j],$$

which we want to be zero for some predefined θ_i. φ_i can also appear as a multistage method, such as the family of Runge-Kutta methods. For example, if we use Heun's method, an explicit RK2 method, we have the following description of φ_i:

$$\tilde{\mathbf{u}} = \mathbf{u}^j + \Delta t\, rhs(\mathbf{u}^j(t))$$
$$\varphi(\mathbf{u}_i) \equiv -\mathbf{u}_i + \mathbf{u}_i^j + \frac{\Delta t}{2}\left(rhs(\mathbf{u}^j(t)) + rhs(\tilde{\mathbf{u}}(t))\right)_i$$

Here, $rhs(\mathbf{u}(t))$ is the right-hand side of the PDE, i.e. $rhs(\mathbf{u}(t)) = \mathbf{A}\mathbf{u}(t) + \mathbf{r}(\mathbf{u}(t)) + \mathbf{f}(t)$. In this definition of φ, the stages of the multistage method are supposed to be evaluated right behind each other. Note that $\tilde{\mathbf{u}}$ is evaluated on the whole subgrid in the first stage, which might give undesired results if we had specified a better method on the other subgrid. This is because φ_i might suffer from propagation of errors of $\tilde{\mathbf{u}}$ caused by the first-stage method on the whole grid, instead of only on a coarse subgrid. A natural way to avoid this problem is to solve a system in each stage of the multistage method. In the first stage of the RK2 example above, a system has to be solved where the forward Euler method is used on one subgrid (θ-method with $\theta_i = 0$), while another method can be used for the other subgrid. This defines $\tilde{\mathbf{u}}$, which is now evaluated by

means of the desired method on each subgrid. The second stage then defines φ_i which can be used to form another system, which we want to solve for the final state $\mathbf{u}$ on time step $j+1$.

In most cases, we will use $k=2$, in which case we have:

$$\begin{cases} \mathbf{F}_{1,1}(\mathbf{y}) \equiv -\mathbf{v}_1 + \mathbf{A}_{11}\mathbf{u}_1 + \mathbf{A}_{12}\mathbf{u}_2 + \langle \mathbf{u},\mathbf{u}\rangle_1 + \mathbf{f}_1(t+\Delta t) \\ \mathbf{F}_{1,2}(\mathbf{y}) \equiv -\mathbf{v}_2 + \mathbf{A}_{21}\mathbf{u}_1 + \mathbf{A}_{22}\mathbf{u}_2 + \langle \mathbf{u},\mathbf{u}\rangle_2 + \mathbf{f}_2(t+\Delta t) \\ \mathbf{F}_{2,1}(\mathbf{y}) \equiv \beta_1\big(\mathbf{A}_{11}\mathbf{v}_1 + \mathbf{A}_{12}\mathbf{v}_2 + \langle \mathbf{u},\mathbf{v}\rangle_1 + \langle \mathbf{v},\mathbf{u}\rangle_1 + \mathbf{f}_1'(t+\Delta t)\big) + \dots \\ \qquad\qquad (1-\beta_1)\varphi_1(\mathbf{u}_1,\mathbf{v}_1) \\ \mathbf{F}_{2,2}(\mathbf{y}) \equiv \beta_2\big(\mathbf{A}_{21}\mathbf{v}_1 + \mathbf{A}_{22}\mathbf{v}_2 + \langle \mathbf{u},\mathbf{v}\rangle_2 + \langle \mathbf{v},\mathbf{u}\rangle_2 + \mathbf{f}_2'(t+\Delta t)\big) + \dots \\ \qquad\qquad (1-\beta_2)\varphi_2(\mathbf{u}_2,\mathbf{v}_2) \end{cases}$$

For the linear case, see appendix A. In general, for all $k>1$, we can set up the system, for which we consider $\mathbf{U}_{j,i}$ as $\mathbf{u}_i^{(j)}$ $(i\in\{1,2\})$, and, as we did previously, $\mathbf{U}_j$ as $\mathbf{u}^{(j)}$. We use the definitions $\phi_{j,i}(\mathbf{B},\mathbf{U}) = \sum_{l=0}^{j} \begin{pmatrix} j \\ l \end{pmatrix} \langle \mathbf{U}_l, \mathbf{U}_{j-l}\rangle_i$,$\mathbf{U} = (\mathbf{U}_0, \mathbf{U}_1 \dots \mathbf{U}_k)$ and, similar to what we defined in the previous section, $\mathbf{U} = (\mathbf{U}_0 \dots \mathbf{U}_k)$. At least, we have $\mathbf{U}_{(1)} = (\mathbf{U}_{0,1}, \mathbf{U}_{1,1} \dots \mathbf{U}_{k,1})$ and $\mathbf{U}_{(2)} = (\mathbf{U}_{0,2}, \mathbf{U}_{1,2} \dots \mathbf{U}_{k,2})$ For these definitions, we assume $i\in\{1,2\}$, since it refers to variables corresponding to one of the two subgrids. We eventually get:

$$\begin{cases} \mathbf{F}_{1,1}(\mathbf{y}) \equiv -\mathbf{U}_{1,1} + \mathbf{A}_{11}\mathbf{U}_{0,1} + \mathbf{A}_{12}\mathbf{U}_{0,2} + \phi_{0,1}(\mathbf{B},\mathbf{U}) + \mathbf{f}_1(t+\Delta t) \\ \mathbf{F}_{1,2}(\mathbf{y}) \equiv -\mathbf{U}_{1,2} + \mathbf{A}_{21}\mathbf{U}_{0,1} + \mathbf{A}_{22}\mathbf{U}_{0,2} + \phi_{0,2}(\mathbf{B},\mathbf{U}) + \mathbf{f}_2(t+\Delta t) \\ \mathbf{F}_{2,1}(\mathbf{y}) \equiv -\mathbf{U}_{2,1} + \mathbf{A}_{11}\mathbf{U}_{1,1} + \mathbf{A}_{12}\mathbf{U}_{1,2} + \phi_{1,1}(\mathbf{B},\mathbf{U}) + \mathbf{f}_1^{(1)}(t+\Delta t) \\ \mathbf{F}_{2,2}(\mathbf{y}) \equiv -\mathbf{U}_{2,2} + \mathbf{A}_{21}\mathbf{U}_{1,1} + \mathbf{A}_{22}\mathbf{U}_{1,2} + \phi_{1,2}(\mathbf{B},\mathbf{U}) + \mathbf{f}_2^{(1)}(t+\Delta t) \\ \vdots \\ \mathbf{F}_{k-1,1}(\mathbf{y}) \equiv -\mathbf{U}_{k-1,1} + \mathbf{A}_{11}\mathbf{U}_{k-2,1} + \mathbf{A}_{12}\mathbf{U}_{k-2,2} + \phi_{k-2,1}(\mathbf{B},\mathbf{U}) + \dots \\ \qquad\qquad \mathbf{f}_1^{(k-2)}(t+\Delta t) \\ \mathbf{F}_{k-1,2}(\mathbf{y}) \equiv -\mathbf{U}_{k-1,2} + \mathbf{A}_{21}\mathbf{U}_{k-2,1} + \mathbf{A}_{22}\mathbf{U}_{k-2,2} + \phi_{k-2,2}(\mathbf{B},\mathbf{U}) + \dots \\ \qquad\qquad \mathbf{f}_2^{(k-2)}(t+\Delta t) \\ \mathbf{F}_{k,1}(\mathbf{y}) \equiv \beta_1\big(\mathbf{A}_{11}\mathbf{U}_{k,1} + \mathbf{A}_{12}\mathbf{U}_{k,2} + \phi_{k,1}(\mathbf{B},\mathbf{U}) + \mathbf{f}_1^{(k)}(t+\Delta t)\big) + \dots \\ \qquad\qquad (1-\beta_1)\varphi_1(\mathbf{u}_1,\mathbf{v}_1) \\ \mathbf{F}_{k,2}(\mathbf{y}) \equiv \beta_2\big(\mathbf{A}_{21}\mathbf{U}_{k,1} + \mathbf{A}_{22}\mathbf{U}_{k,2} + \phi_{k,2}(\mathbf{B},\mathbf{U}) + \mathbf{f}_2^{(k)}(t+\Delta t)\big) + \dots \\ \qquad\qquad (1-\beta_2)\varphi_2(\mathbf{u}_2,\mathbf{v}_2) \end{cases} \tag{7}$$

The corresponding state $\mathbf{y}$ is given by:

$$\mathbf{y} = \begin{pmatrix} \mathbf{U}_{0,1} \\ \mathbf{U}_{0,2} \\ \mathbf{U}_{1,1} \\ \mathbf{U}_{1,2} \\ \vdots \\ \mathbf{U}_{k,1} \\ \mathbf{U}_{k,2} \end{pmatrix}$$

This enables us to set up the Jacobian: $J_{\mathbf{F}}(\mathbf{y}^j) =$

$$\begin{pmatrix} \mathcal{A}_0 & -\mathbf{I} & 0 & & & \\ \mathcal{B}_{1,1} & \mathcal{A}_1 & -\mathbf{I} & 0 & & \\ \vdots & & & \ddots & & \\ \mathcal{B}_{k-1,k-1} & \mathcal{B}_{k-1,k-2} & \dots & \mathcal{B}_{k-1,1} & \mathcal{A}_{k-1} & -\mathbf{I} \\ \mathcal{B}^{\dagger} & \mathcal{B}^{\ddagger} & \mathcal{B}^{\#}_{k-2} & \dots & \mathcal{B}^{\#}_{1} & \mathcal{A}^{\#} \end{pmatrix} \tag{8}$$

where

$$\mathcal{A}_j = \begin{pmatrix} \mathbf{A}_{11} + \psi_{j,0,1} & \mathbf{A}_{12} + \eta_{j,0,1} \\ \mathbf{A}_{21} + \eta_{j,0,2} & \mathbf{A}_{22} + \psi_{j,0,2} \end{pmatrix}$$

$$\mathcal{A}^{\#} = \begin{pmatrix} \beta_1(\mathbf{A}_{11} + \psi_{k,0,1}) & \beta_1(\mathbf{A}_{12} + \eta_{k,0,1}) \\ \beta_2(\mathbf{A}_{21} + \eta_{k,0,2}) & \beta_2(\mathbf{A}_{22} + \psi_{k,0,2}) \end{pmatrix}$$

$$\mathcal{B}_{j,i} = \begin{pmatrix} \psi_{j,i,1} & \eta_{j,i,1} \\ \eta_{j,i,2} & \psi_{j,i,2} \end{pmatrix}$$

$$\mathcal{B}^{\dagger} = \begin{pmatrix} \beta_1\psi_{k,k,1} - (1-\beta_1)\mathbf{I} & \beta_1\eta_{k,k,1} \\ \beta_2\eta_{k,k,2} & \beta_2\psi_{k,k,2} - (1-\beta_2)\mathbf{I} \end{pmatrix}$$

$$\mathcal{B}^{\ddagger} = \begin{pmatrix} \beta_1\psi_{k,k-1,1} - (1-\beta_1)\theta_1\Delta t\ \mathbf{I} & \beta_1\eta_{k,k-1,1} \\ \beta_2\eta_{k,k-1,2} & \beta_2\psi_{k,k-1,2} - (1-\beta_2)\theta_2\Delta t\ \mathbf{I} \end{pmatrix}$$

$$\mathcal{B}^{\#}_i = \begin{pmatrix} \beta_1\psi_{k,i,1} & \beta_1\eta_{k,i,1} \\ \beta_2\eta_{k,i,2} & \beta_2\psi_{k,i,2} \end{pmatrix}$$

Furthermore,

$$\psi_{j,i,l} = \psi_{j,i,l}(\mathbf{B}, \mathbf{U}_{(1)}, \mathbf{U}_{(2)}) = \begin{pmatrix} j \\ i \end{pmatrix} (diag(\mathbf{B}_{l1}\mathbf{U}_{i,1} + \mathbf{B}_{l2}\mathbf{U}_{i,2}) + diag(\mathbf{U}_{i,l})\mathbf{B}_{ll})$$

$$\eta_{j,i,1} = \eta_{j,i,1}(\mathbf{B}, \mathbf{U}_{(1)}) = \begin{pmatrix} j \\ i \end{pmatrix} diag(\mathbf{U}_{i,1})\mathbf{B}_{12}$$

$$\eta_{j,i,2} = \eta_{j,i,2}(\mathbf{B}, \mathbf{U}_{(2)}) = \begin{pmatrix} j \\ i \end{pmatrix} diag(\mathbf{U}_{i,2})\mathbf{B}_{21}$$

Remark An extension of this system can be made by varying in treating $\mathbf{u}_1$ and $\mathbf{u}_2$ implicitly or explicitly in the PDE part, similar to what we did in section 2. Theoretically, this gives rise to 4 possibilities. However, if the state on one subgrid is treated explicitly while it is treated implicitly on the other one, than the matrix $\mathbf{A}$ affects strongly the difference between the resulting solution and the solution yielded from treating the subgrids equally, which we want to avoid. Therefore, only two possibilities of four are feasible. If $\mathbf{u}_1$ and $\mathbf{u}_2$ are treated implicitly in the PDE part, then we define $\alpha = 1$, otherwise (i.e. both $\mathbf{u}_1$ and $\mathbf{u}_2$ are treated explicitly in the PDE part), $\alpha = 0$. Although the α and β_i seems to be free parameters, not any combination yields a feasible solution. From the structure of the Jacobian and the matrix $\mathcal{B}^\dagger$, we conclude that a necessary condition for the system to have a feasible solution, is that if $\beta_1 = 1$ or $\beta_2 = 1$, then $\alpha = 1$. Consistency in time steps requires that if $\alpha = 0$ and an integration method is used on $\mathbf{u}_i$, then $\theta_i = 1$. So if $\alpha = 0$, then we require that $\beta_1, \beta_2 = 0$ and $\theta_1, \theta_2 = 1$. However, the resulting combination of parameters is equivalent to $\alpha = 1$, $\beta_1, \beta_2 = 0$, $\theta_1, \theta_2 = 0$ in that it gives rise to the same solution. In other words, if $\alpha = 0$, then we have to use explicit integration methods on both subgrids. Since in most cases it is unnecessary to use two different explicit integration methods, use of subgrids becomes unnecessary as well. For this reason we consider α as a redundant parameter and we refer to section 2 for solving a system with explicit use of the variables in the PDE part.

4.1 Special cases

In the special case that $k = 1$, the Jacobian assumes the form

$$J_{\mathbf{F}}(\mathbf{y}^j) = \begin{pmatrix} \beta_1(\mathbf{A}_{11} + \psi_{0,0,1}) - (1-\beta_1)\mathbf{I} & \beta_1(\mathbf{A}_{12} + \eta_{0,0,1}) \\ \beta_2(\mathbf{A}_{21} + \eta_{0,0,2}) & \beta_2(\mathbf{A}_{22} + \psi_{0,0,2}) - (1-\beta_2)\mathbf{I} \end{pmatrix}$$

In the case that $k = 2$, the Jacobian assumes the form

$$J_{\mathbf{F}}(\mathbf{y}^j) = \begin{pmatrix} \mathcal{A}_0 & -\mathbf{I} \\ \mathcal{B}^\dagger & \mathcal{A}^* \end{pmatrix}$$

where

$$A^* = \begin{pmatrix} \beta_1(\mathbf{A}_{11} + \psi_{1,0,1}) - (1-\beta_1)\theta_1\Delta t\ \mathbf{I} & \beta_1(\mathbf{A}_{12} + \eta_{1,0,1}) \\ \beta_2(\mathbf{A}_{21} + \eta_{1,0,2}) & \beta_2(\mathbf{A}_{22} + \psi_{1,0,2}) - (1-\beta_2)\theta_2\Delta t\ \mathbf{I} \end{pmatrix}$$

4.2 Parallel computing

Suppose we are given tridiagonal matrices $\mathbf{A}$ and $\mathbf{B}$ (which occurs very often, since the state on some node is most affected by nodes which are in its neighbourhood, and less affected by nodes further away). Define the second subgrid to consist of isolated nodes (with at least one node of the other subgrid between

each of them), eliminate the corresponding state ($\mathbf{u}_2$) and compute it with an explicit integration method. Then, if one time step is performed, the state on a node of the first subgrid (say $u_{1,j}$) does not depend on the state of another node $u_{1,k}$ of the first subgrid if between them there exists some node which belongs to the second subgrid. Therefore, the nodes of the second subgrid divides the first subgrid into groups that can be computed in parallel. In general, if $\mathbf{A}$ and $\mathbf{B}$ are both matrices of bandwidth $m \ll n$ and the nodes of the second subgrid are clustered in groups of at least m neighbouring nodes, then the first subgrid consists of groups of nodes that can be computed in parallel, which are divided by the clustered nodes of the second subgrid.

4.3 (Quasi) stationary approximations and elimination

If we use an explicit integration method for one subgrid, while we use a (quasi) stationary approximation for the other subgrid, what is the mathematical meaning of such a system? The integration method can be thought of as solution which depends on the first order derivative in time, whereas the (quasi) stationary is independent of this first order derivative. This implies that we do not need an initial condition to obtain a (quasi) stationary solution. In fact, we need only the boundary conditions, which might be time-dependent.

As usual, we assume that $\mathbf{u}_2$ is evaluated by an explicit integration method. In each time step, the state on the corresponding subgrid is computed first. Assume further that each group of neighbouring nodes in the first subgrid can be computed in parallel. Then the state on the two neighbouring nodes of each such a group belongs to the second subgrid and can be seen as 'local' (Dirichlet or Neumann or higher) boundary conditions, since they are already computed. Between these nodes, a (quasi) stationary solution is then obtained. Since in this solution only the boundary conditions are needed, the (quasi) stationary approximation can be seen as an interpolating method, which yields smooth solutions close to the real solution for sufficiently smooth real solutions.

4.4 Implementation of boundary conditions

Boundary conditions are required to solve the PDE (1). We assume that these are Dirichlet or Neumann conditions. The system can be adapted easily in implementing the boundary conditions. If we express the Dirichlet or Neumann condition in terms of the grid nodes, then it forms an equation which can be substituted in the system. If we assume that the grid is from L to R, then the Dirichlet condition $u(L) = a(t)$, or $u(R) = b(t)$, for $a, b \in \mathbb{R}$ can be written as:

$$u_1 = a(t), u_n = b(t)$$

A Neumann condition $u'(L) = a(t)$, or $u'(R) = b(t)$, can be approximated by:

$$\frac{u_2 - u_1}{h} = a(t), \frac{u_n - u_{n-1}}{h} = b(t)$$

5 Investigation

In all investigations that are done, the Burgers' equation (2) is considered, where $\mu = 0.1$ on a grid consisting of 50 nodes with equal distance from 0 to 1. We define a Dirichlet boundary condition on the left side

$$u(0,t) = 1 + 0.1\sin(\omega t)$$

for some $\omega \in \mathbb{R}$, while on the right boundary we set a Neumann condition $u_x(1,t) = 0$. In the limit of t, the solution will be periodic with period $T = \frac{2\pi}{\omega}$. For several methods described in the previous sections, our goal is to determine the asymptotic maximum error with respect to space (all nodes) and time (in one period). The reference solution is computed using a full integration method (namely, a Runge-Kutta 4 method), without making use of subgrids. For this solution we have used a small time step of $\Delta t = 5 \cdot 10^{-5}$. By comparing (a part) of this solution to solutions with even a smaller time step, it turned out that the error cannot be smaller. This was no problem, since this error was in the order of 10^{-15}, close to machine precision.

The methods are fully defined in terms of the parameters described in section 4. In the experiments, these are:

method	grid*	β_1	β_2	$\theta_1^\dagger$	$\theta_2^\dagger$
Full QS	1	1	-	-	-
Full RK4	1	0	-	RK4	-
Full CN	1	0	-	CN	-
I/QS: RK4	1,2	1	0	-	RK4
I/QS: RK2	1,2	1	0	-	RK2
I/QS: CN	1,2	1	0	-	CN
I/QS: BE	1,2	1	0	-	BE
I/QS: FE	1,2	1	0	-	FE
I/I: RK4/CN	1,2	0	0	CN	RK4
I/I: RK2/CN	1,2	0	0	CN	RK2
I/I: RK4/CN	1,2	0	0	BE	RK4
I/I: RK2/CN	1,2	0	0	BE	RK2

* the subgrids which are used
† the integration method used on the first subgrid
‡ the integration method used on the second subgrid

Here, RK4, RK2, CN, BE, FE refer to the Runge-Kutta 4, Runge-Kutta 2, Crank-Nicolson, Backward Euler and Forward Euler method, respectively. The method with FE will only be used in the stability investigation and for this

method we set $\mathbf{B} = 0$, i.e. we treat the linear case. Throughout this section, we assume that we perform the explicit integration method on the second subgrid.

If we refer to $\mathbf{u}_{[meth]}(t)$ as the solution of the equation on time t according to method *meth*, then for each method, we wish to determine the asymptotic maximum error in space and time:

$$\epsilon_{[meth]} \equiv \lim_{t \to +\infty} \left(\max_{x,\ s \in [t, t+T]} |\mathbf{u}_{[meth]}(s) - \mathbf{u}_{[ref]}(s)| \right)$$

Since the exact value will be very hard to determine for an arbitrary initial value, we approximate ϵ as follows: we choose ω such that T is a multiple of Δt. Then in each period, we measure the maximum error in space and time with regard to our reference solution. Since the error is computed at the same 'argument' in each period, these maximum errors can be compared to each other. When the maximum does not change anymore up to a relative difference of 10^{-3}, then ϵ is determined by the last computed maximum error.

Three types of investigation have been performed. The first one is a stability investigation, in which the stability of the 12 methods are determined for different values of ω, the value k and different grid partitions. The second one is a convergence investigation, in which we tried to find the order of convergence with respect to the time step Δt. This has been done for all described methods except the integration-QS hybrid method with FE, and various values of ω, k and different grid partitions. The third investigation is a broad investigation in which the convergence with respect to ω and k are determined and where the other variables are varied as much as possible.

5.1 Stability investigation

In this investigation, the stability of the 12 methods are determined for each combination of the following values of ω, the value k and grid partitions:

ω^*	k	second subgrid
8	0	1:2:n
4	1	1:4:n
2	2	1:8:n
1	3	[17 42]
.5		[33]
.25		
.125		

* in fact, these values are approximations. The used values are such that T is a multiple of $\Delta t = 2^*10^{-3}$, obtained by the following code (the used ω is such that ω_{used}=ratio*ω_{tabular}):

```
dt0=2e-3; r=10;
maxw=8;
```

```
m=r*round(2*pi/(r*maxw*dt0));
maxwcorr=2*pi/(m*dt0);
ratio=maxwcorr/maxw;
```

Empirically, for each combination of parameters, the maximum value of Δt is determined for which the solution is stable. This has been done with an accuracy of two digits. It turned out that ω did not affect this value. The results with respect to the other variables are shown below:

Full (quasi) stationary approximation

This method turns out to be always stable unless the solution crosses the u-axis. In that case, there exists no time step for which it becomes stable. This is exactly what we expect, since the (quasi) stationary approximation is independent of the first order derivative in time. It only depends on the boundary conditions.

Full time integration: RK4

For this method, distinction in various values of k and grid partitions is not relevant. We use the whole grid as one subgrid and we obtained the following maximum time step:

$\Delta t = 0.0029$

Full time integration: CN

This method turns out to be always stable too. This is indeed what we expect since Crank-Nicolson is an unconditionally stable implicit integration method.

Integration-(quasi) stationary hybrid: FE (linear case)

Note that for this case, $\mathbf{B}$ is set to zero. The results for this method are shown below:

index$_2$ ↓ k→	1	2	3	4
1:2:n	0.0041	0.0042	0.0042	0.0042
1:4:n	0.0084	0.0131	0.015	0.015
1:8:n	0.018	0.058	0.077	0.084
[17 42]	0.068	0.72	0.86	0.88
[33]	0.26	7.2	7.9	7.9

We see that the coarser the second subgrid, the bigger is the allowed time step, especially for bigger k. Also, we note that for increasing k, the results are better too. For $k = 2$, we will verify the results. From appendix A.1, it follows that $\mathbf{v}_2$, the first order derivative of the state on the coarse subgrid, can be given as: $\mathbf{v}_2 = D(\mathbf{u}_2 + \mathbf{v}_2^j)$, where $D = -(I + A_{21}A_{11}^{-2}A_{12})^{-1}(-A_{22} + A_{21}A_{11}^{-1}A_{12})$. The absolute stability region of the forward Euler method is $|z + 1| \leq 1$, where $z = \Delta t\, \lambda_i$ and λ_i is the i'th eigenvalue of the matrix D. This must hold for all

eigenvalues of this matrix, thus $|\sigma(D) + \Delta t| \leq \Delta t$. Hence, the eigenvalues of D must lie in a circle with radius Δt centered around $-\Delta t$. In the case of real eigenvalues, all eigenvalues have to be non-positive and if so, we expect that the maximum time step can be computed by: $\Delta t = -2/\min_{\lambda_i \in \sigma(D)} \lambda_i$. We have verified that in the cases for $k = 2$, D contains indeed real, negative eigenvalues. The minimum ones are -475.82, -152.02, -34.164, -2.7515, -0.27614 for the grid partitions in the same order as described above. All these values indeed yield the maximum time step equal to the results above.

Integration-(quasi) stationary hybrid: RK4

The results for this method are shown below:

$\text{index}_2 \downarrow$ k$\rightarrow$	1	2	3	4
1:2:n	0.0041	0.0058	0.0058	0.0058
1:4:n	0.0045	0.0067	0.010	0.015
1:8:n	0.0045	0.0089	0.020	0.044
[17 42]	0.0045	0.011	0.047	0.18
[33]	0.0045	0.012	0.054	0.24

A similar observation can be made: in general, bigger k and a finer subgrid 2 yield better results. If we compare these values to the full RK4 method, we notice overall improvement. However, the improvement of the maximum allowed time step for bigger k and a finer subgrid 2 is not as good as in the previous experiment with forward Euler. Although the linear case is treated there, it cannot explain the big difference, since the equation is not highly non-linear. Furthermore, a similar experiment where the linear case is treated yields very similar results. It remains a question why this RK4 hybrid method gives mostly worse results with regard to the FE hybrid method, however we think that a wrong implementation is responsible for this observation.

Integration-(quasi) stationary hybrid: RK2

The results for this method are shown below:

$\text{index}_2 \downarrow$ k$\rightarrow$	1	2	3	4
1:2:n	0.0041	0.0041	0.0042	0.0042
1:4:n	0.0041	0.0084	0.012	0.014
1:8:n	0.0041	0.016	0.043	0.052
[17 42]	0.0041	0.034	0.25	0.53
[33]	0.0041	0.039	0.34	0.65

Again, we obtain similar results: in general, bigger k and a finer subgrid 2 yield a bigger maximum time step and the results are in most cases worse with regard to the FE hybrid method. However, we notice that it is something better than the previous experiment. Also, this is strange. We would expect that RK4 gives results which are at least as good as results of RK2 which are in its turn at least as good as result of FE. We obtain opposite results.

Integration-(quasi) stationary hybrid: CN

This method turns out to be always stable. This is indeed what we expect since both the (quasi) stationary and the Crank- Nicolson method have no stability limits.

Integration-(quasi) stationary hybrid: BE

This method turns out to be always stable too. This is indeed what we expect, for the same reasons as above.

Implicit-explicit methods

$\text{index}_2 \downarrow$ method$\rightarrow$	CN&RK4	CN&RK2	BE&RK4	BE&RK2
1:2:n	0.0041	0.0041	0.0041	0.0041
1:4:n	0.0045	0.0041	0.0045	0.0041
1:8:n	0.0045	0.0041	0.0045	0.0041
[17 42]	0.0045	0.0041	0.0045	0.0041
[33]	0.0045	0.0041	0.0045	0.0041

We notice a marginal difference between the several methods. Furthermore, it seems that the grid partition does not affect the stability limit strongly.

5.2 Investigation of convergence w.r.t. Δt

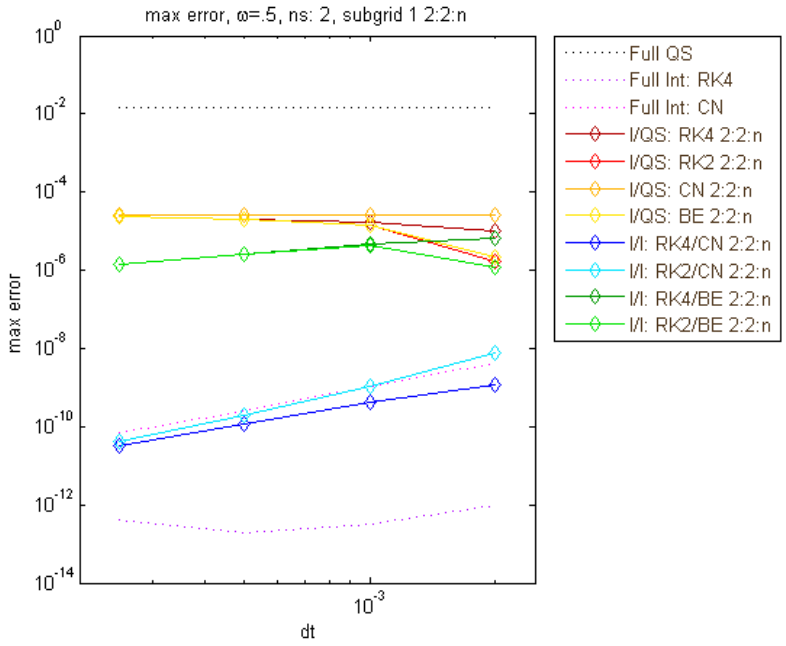
In this investigation we tried to find the order of convergence with respect to the time step Δt. This has been done for all described methods, $k = 2$, $\omega = 0.5^*$ and all combinations of various values of Δt and different grid partitions, given in the diagram below:

Δt	second subgrid
2^*10^{-3}	2:2:n
1^*10^{-3}	2:4:n
5^*10^{-4}	2:8:n
2.5^*10^{-4}	[17 42]
	[33]

* in fact, this value is an approximation. The used values are such that T is a multiple of $\Delta t = 2^*10^{-3}$, as in the previous investigation.

Note that for each used value of Δt, T is a multiple of Δt. The order of convergence is determined by the slope of the linear least squares solution through the points $(\log(\Delta t), \log(\epsilon))$. The results are shown below:

Subgrid 2: 2:2:n



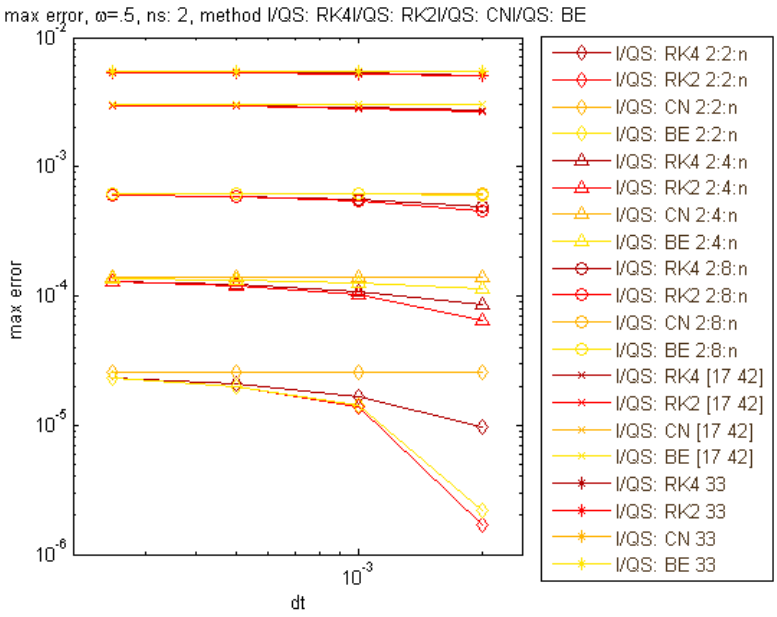
METHOD	A	B
'Full QS'	[-7.9191e-007]	[-6.1308]
'Full Int: RK4'	[-0.15638]	[-43.321]
'Full Int: CN'	[1.9722]	[-10.194]
'I/QS: RK4 2:2:n'	[-0.23267]	[-18.159]
'I/QS: RK2 2:2:n'	[-0.36375]	[-19.713]
'I/QS: CN 2:2:n'	[-1.2823e-006]	[-15.235]
'I/QS: BE 2:2:n'	[-0.35372]	[-19.591]
'I/I: RK4/CN 2:2:n'	[1.8379]	[-12.854]
'I/I: RK2/CN 2:2:n'	[2.3486]	[-6.4578]
'I/I: RK4/BE 2:2:n'	[0.86633]	[-9.0507]
'I/I: RK2/BE 2:2:n'	[0.78078]	[-10.062]

The above results are obtained from a linear least squares approach and have to be read as: $\epsilon = 2^{A^2 \log(\Delta t)+B} = \Delta t^A 2^B$.

The QS solution doesn't converge at all. This is because this method is independent of any state in the past: it assumes a solution which is fundamentally different. RK4 shows the best results, but its convergence seems to stagnate. This is probably due to the fact that the error approaches machine

precision. CN shows the expected convergence order of 2. All integration-QS hybrid methods don't converge as well. Obviously, this is caused by the QS part of the method. However, in absolute sense, these errors are decreased with regard to the error of the QS method by a factor of about 10^3, which must be caused by the integration part. Remarkable results can be found for $\Delta t = 0.002$, where the maximum error is smaller than for smaller values of Δt. It might be that the error due to the integration part cancels out partly the error due to the QS part (at least, for this specific set of parameters). Implicit-explicit methods converge with an order equal to the order of the used integration method with the biggest error. In the case CN is used, this number will be approximately 2 since RK4 and RK2 give better results than order 2 method CN. Similarly, for BE, this number is approximately 1.

Integration-(quasi) stationary hybrid methods

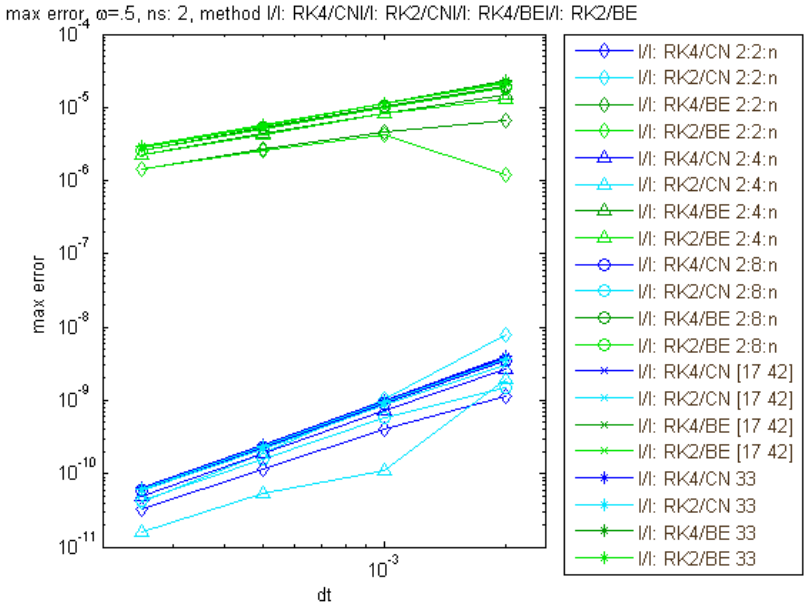


METHOD	A	B
'I/QS: RK4 2:2:n'	[-0.23267]	[-18.159]
'I/QS: RK2 2:2:n'	[-0.36375]	[-19.713]
'I/QS: CN 2:2:n'	[-1.2823e-006]	[-15.235]
'I/QS: BE 2:2:n'	[-0.35372]	[-19.591]
'I/QS: RK4 2:4:n'	[-0.13069]	[-14.459]
'I/QS: RK2 2:4:n'	[-0.17173]	[-14.947]
'I/QS: CN 2:4:n'	[-9.5994e-007]	[-12.816]

```
'I/QS: BE 2:4:n'        [   -0.049019]    [-13.428]
'I/QS: RK4 2:8:n'       [   -0.064292]    [-11.459]
'I/QS: RK2 2:8:n'       [   -0.079919]    [-11.645]
'I/QS: CN 2:8:n'        [ 1.1882e-008]    [-10.649]
'I/QS: BE 2:8:n'        [   -0.010512]    [ -10.78]
'I/QS: RK4 [17 42]'     [   -0.028583]    [-8.7298]
'I/QS: RK2 [17 42]'     [    -0.03474]    [-8.8032]
'I/QS: CN [17 42]'      [-1.0306e-006]    [-8.3694]
'I/QS: BE [17 42]'      [  -0.0018363]    [-8.3924]
'I/QS: RK4 33'          [   -0.016015]    [-7.7272]
'I/QS: RK2 33'          [   -0.019357]    [ -7.767]
'I/QS: CN 33'           [-1.3515e-006]    [-7.5253]
'I/QS: BE 33'           [ -0.00063627]    [-7.5332]
```

For all integration-QS hybrid methods used, we see that the order of convergence is approximately 0. This is what we expect from the QS part, which doesn't show convergence and its error dominates the error caused by the integration part. Note that the finer subgrid 2, the better the results are. Since the QS method yields bigger errors in our time step range, we would indeed expect that the finer subgrid 1, the bigger is the influence of the QS part and the bigger the maximum error. As in the previous plot, we notice a significant reduction of the error for bigger Δt in the order of 10^{-5} for all partitions. Also in this case, I assume that the error due to the integration part cancels out partly the error due to the QS part. We notice further that this cancellation error seems to be the biggest for the RK2 method and negligible for the RK4 method. This is probably because RK4 integration gives smaller errors than RK2. If we focus our attention to smaller Δt, then distinction of the error cannot be made between the several methods: for all integration methods used, the error is then negligible in comparison with the error caused by the QS part, even for a coarse subgrid 2.

Implicit-explicit methods



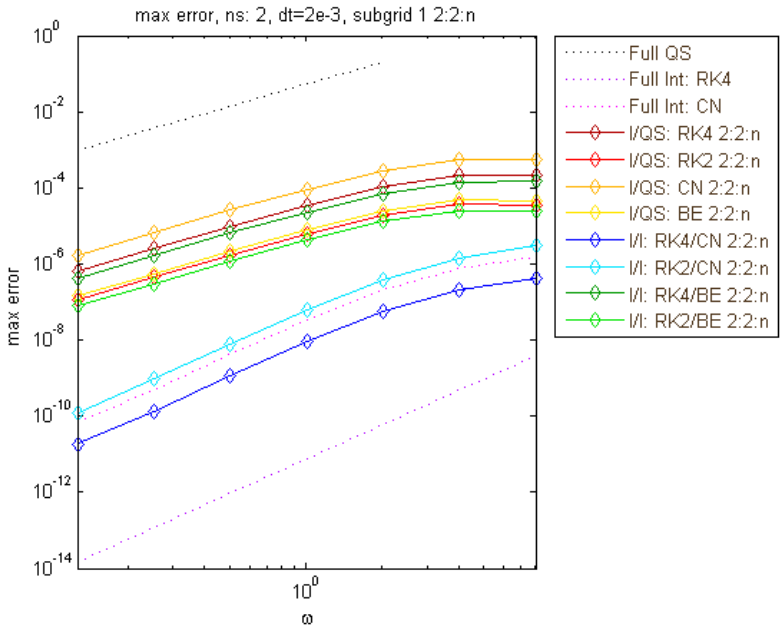
METHOD	A	B
'I/I: RK4/CN 2:2:n'	[1.8379]	[-12.854]
'I/I: RK2/CN 2:2:n'	[2.3486]	[-6.4578]
'I/I: RK4/BE 2:2:n'	[0.86633]	[-9.0507]
'I/I: RK2/BE 2:2:n'	[0.78078]	[-10.062]
'I/I: RK4/CN 2:4:n'	[1.9289]	[-11.133]
'I/I: RK2/CN 2:4:n'	[1.3999]	[-18.997]
'I/I: RK4/BE 2:4:n'	[0.95672]	[-7.3364]
'I/I: RK2/BE 2:4:n'	[0.93885]	[-7.5481]
'I/I: RK4/CN 2:8:n'	[1.9554]	[-10.58]
'I/I: RK2/CN 2:8:n'	[1.8696]	[-12.055]
'I/I: RK4/BE 2:8:n'	[0.98313]	[-6.7829]
'I/I: RK2/BE 2:8:n'	[0.97638]	[-6.863]
'I/I: RK4/CN [17 42]'	[1.9664]	[-10.334]
'I/I: RK2/CN [17 42]'	[1.9422]	[-10.774]
'I/I: RK4/BE [17 42]'	[0.99374]	[-6.5421]
'I/I: RK2/BE [17 42]'	[0.99127]	[-6.5714]
'I/I: RK4/CN 33'	[1.9698]	[-10.26]
'I/I: RK2/CN 33'	[1.9585]	[-10.469]
'I/I: RK4/BE 33'	[0.99686]	[-6.4699]
'I/I: RK2/BE 33'	[0.99564]	[-6.4844]

The first we notice is that the CN methods yield better results, both with respect to the absolute error (in our time step range) and the order of convergence, which is about 2 for the CN methods and 1 for BE methods. A better explicit method doesn't improve the results: RK4 hybrid methods don't perform better than RK2 hybrid methods. This is what we expect: the order of convergence is determined by the lowest order of the methods used, in each case this is the order of the implicit methods. However, the finer the subgrid to which the explicit method is performed, the bigger is the influence of the errors caused by this method. This explains probably the more irregular course for the RK2 method with fine subgrid 2.

5.3 Investigation of convergence w.r.t. ω

In this investigation, we will look how the maximum error evolves if the frequency of the solution is increased. We focus on the QS method, integration-QS methods with $k = 2$, implicit-explicit methods and two cases for several methods with $k = 2$ (subgrid 2: 2:2:n and [33]), where we begin with.

$k = 2$, subgrid 2: 2:2:n



METHOD	A	B
'Full QS'	[1.9709]	[-4.1977]
'Full Int: RK4'	[2.9957]	[-36.9933]
'Full Int: CN'	[2.9759]	[-24.9061]

'I/QS: RK4 2:2:n'	[1.9356]	[-14.7725]
'I/QS: RK2 2:2:n'	[1.9374]	[-17.2917]
'I/QS: CN 2:2:n'	[1.9355]	[-13.3620]
'I/QS: BE 2:2:n'	[1.9376]	[-16.9193]
'I/I: RK4/CN 2:2:n'	[2.9785]	[-26.7772]
'I/I: RK2/CN 2:2:n'	[3.0139]	[-23.9523]
'I/I: RK4/BE 2:2:n'	[1.9352]	[-15.3641]
'I/I: RK2/BE 2:2:n'	[1.9371]	[-17.8013]

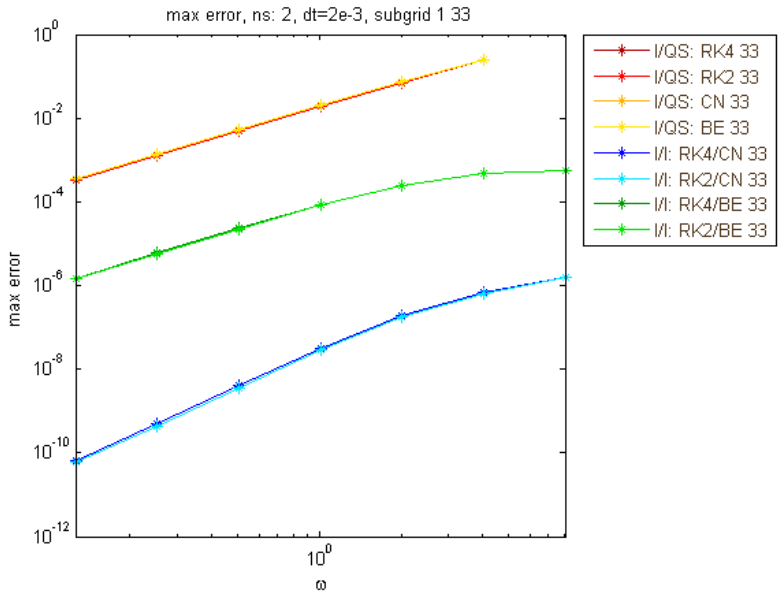
The order of convergence with respect to ω for different methods is shown above (under column A). A and B have similar meanings as above: $\epsilon = \omega^A 2^B$.

The results above are shown for $k = 2$. Subgrid 2 is 2:2:n when a hybrid method is used. The full QS method yields an order of 1. For higher k, the order is getting higher and satisfies being $O(\omega^k)$. This will be further outlined in a later subsection.

Although we don't have many results, it seems that for all hybrid methods, as we would expect, it assumes the order of the methods which yields the biggest error. Another, less solid presumption, is that for full integration methods (which aren't affected by k), the order of convergence with respect to ω exceeds the order of convergence with respect to Δt by one. However, for RK4 we then would expect an order of 5 instead of 3. I presume therefore that (at least) two processes determines this convergence order, however it is still a question which processes these are. Actually, we need more observations to explain these numbers. Therefore we investigate the behaviour of the hybrid methods with respect to ω later.

One notable thing is the flattening of the curve for higher values of ω. At first sight, we thought that this was due to similar magnitudes of the error and the solution itself. However, as we shall see later more obviously, the flattening is as good as visible for much lower errors. Another explanation is the following: if ω is big, then we have a very short period T in which errors have less time to propagate in integration methods. As we will see later, this phenomenon cannot be seen at (full) QS methods. Furthermore, note that the QS method is not stable anymore for higher ω. This is because the error becomes so big, that the solution crosses the u-axis, which gives troubles due to the non-linear term (the Jacobian then becomes singular).

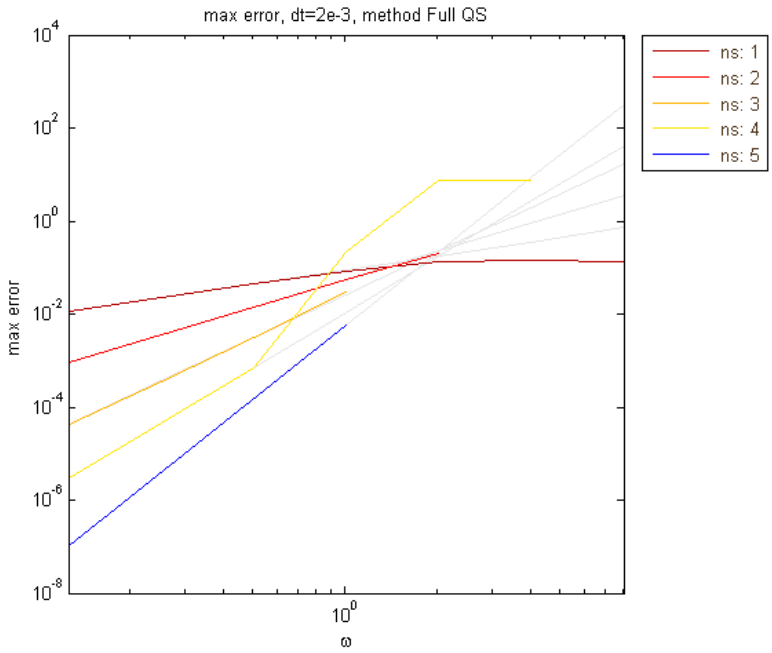
$k = 2$, **subgrid 2: [33]**



METHOD	A	B
'I/QS: RK4 33'	[1.9678]	[-5.6798]
'I/QS: RK2 33'	[1.9684]	[-5.7029]
'I/QS: CN 33'	[1.9656]	[-5.6012]
'I/QS: BE 33'	[1.9654]	[-5.6048]
'I/I: RK4/CN 33'	[2.9762]	[-24.9579]
'I/I: RK2/CN 33'	[2.9722]	[-25.1177]
'I/I: RK4/BE 33'	[1.9349]	[-13.5426]
'I/I: RK2/BE 33'	[1.9350]	[-13.5565]

The data reveals that the order of convergence has not been changed with respect to the previous result. In this data, subgrid 2 is taken to be very coarse, so we expect that the error is mainly determined by the method used on the other subgrid. To begin with, the errors of the integration-QS hybrid methods are indeed barely influenced by the integration method, which causes the red/yellow lines to be hardly distinctive. Furthermore, we see that they show a course of the error that is quite similar to that of the QS method which is shown in the previous plot. We obtain similar results for implicit-explicit methods. Since the error is hardly affected by the more accurate explicit method, each implicit-explicit method shows a curve similar to the curve of the used implicit methods alone (in the case of CN, this can be seen in the previous plot).

(Quasi) stationary method



METHOD	A	B
'k: 1'	[0.9968]	[-3.4854]
'k: 2'	[1.9864]	[-4.1616]
'k: 3'	[3.0967]	[-5.2531]
'k: 4'	[3.9647]	[-6.5619]
'k: 5'	[5.2536]	[-7.4736]

In this plot, the maximum errors of the full QS method are given for several k, together with their linear least squares approximation over the nodes with lowest ω in grey. We see that, except for $k = 2$, the error could not be determined for high ω. As earlier stated, this might be due to negative solutions, which causes the Jacobian to be singular. For $k = 2$ and $k = 5$, we notice a very different course of the plot for higher ω. I presume that the case for $k = 5$ is an inaccurate result due to a high condition number of the Jacobian. The flattening for $k = 2$ is explained in a previous result.

We notice that for small ω, the errors are smaller for higher k and that the order of convergence is approximately equal to k. Another interesting thing of this is plot is the intersection of all least squares maximum errors at $\omega \approx 2$. For all k, the maximum error is approximately 0.2 at $\omega = 2$. We will show that the error ϵ can be written as $\epsilon = c_1 \, (c_2\omega)^k$, where c_1 and c_2 are constants. Denote

by u and $\mathbf{u}$ the exact solution of the PDE and the solution obtained by the QS method, respectively. From system (4) where all ϕ_j are vanished, we can write $\mathbf{u}$ explicitly in terms of $\mathbf{f}(t)$ and its derivations:

$$\mathbf{u} = -\left(\mathbf{A}^{-k}\mathbf{f}^{(k-1)}(t) + \ldots + \mathbf{A}^{-1}\mathbf{f}^{(0)}(t)\right)$$

On the other hand, the exact solution u can be written as follows:

$$\begin{aligned} u(t) &= C\, e^{\mathbf{A}t} \\ \frac{dC}{dt} &= \frac{du}{dt}e^{-\mathbf{A}t} - u(t)Ae^{-\mathbf{A}t} = \mathbf{f}(t)e^{-\mathbf{A}t} \\ u(t) &= \int_0^t \mathbf{f}(s)e^{\mathbf{A}(t-s)}ds \end{aligned} \tag{9}$$

By performing partial integration k times on the right-hand side of (9), we obtain the following expression for $u(t)$:

$$u(t) = -\sum_{i=1}^{k} \mathbf{A}^i\mathbf{f}^{(i-1)}(t) + \sum_{i=1}^{k} \mathbf{A}^i e^{\mathbf{A}t}\mathbf{f}^{(i-1)}(0) + \int_0^t \mathbf{A}^{-k}e^{\mathbf{A}(t-s)}\frac{d^k\mathbf{f}}{ds^k}ds$$

Since $\lim_{t\to\infty} e^{\mathbf{A}t} = 0$, the second term vanishes and we can write the maximum error ϵ as:

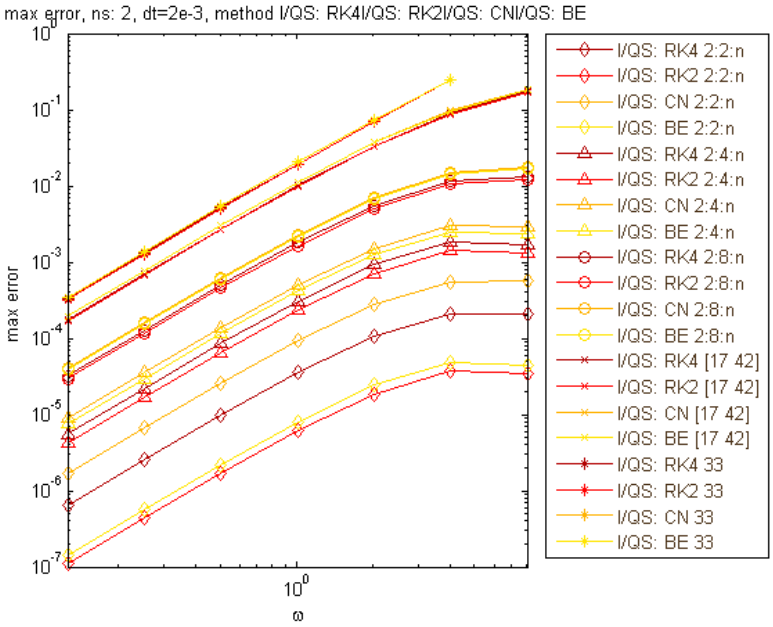
$$\begin{aligned} \epsilon &= \max_t \|u(t) - \mathbf{u}(t)\|_\infty \\ &= \left\| \int_0^t \mathbf{A}^{-k}e^{\mathbf{A}(t-s)}\frac{d^k\mathbf{f}}{ds^k}ds \right\|_\infty \\ &\leq \left\| \int_0^t \mathbf{A}^{-k}e^{\mathbf{A}(t-s)}ds \right\|_\infty \max\left|\frac{d^k\mathbf{f}}{ds^k}\right| \\ &= \left\|\mathbf{A}^{-k-1}(-I + e^{\mathbf{A}t})\right\|_\infty \max\left|\frac{d^k\mathbf{f}}{ds^k}\right| \\ &= \left\|\mathbf{A}^{-k-1}\right\|_\infty \max\left|\frac{d^k\mathbf{f}}{ds^k}\right| \\ &= \gamma\,|\lambda_{min}(\mathbf{A})|^{-k-1} \max\left|\frac{d^k\mathbf{f}}{ds^k}\right| \end{aligned}$$

for $t \to \infty$ and γ being approximately constant for $k \geq 0$ in the order of 1. In our investigation, we used $\mathbf{f}(t) = 1 + 0.1\sin(\omega t)$. This yields:

$$\epsilon \leq 0.1\gamma\,|\lambda_{min}(\mathbf{A})|^{-k-1}\ \omega^k$$

for $k \geq 1$. This is indeed of the form $\epsilon = c_1\,(c_2\omega)^k$, where $c_1 = 0.1\,\gamma|\lambda_{min}|^{-1}$ and $c_2 = |\lambda_{min}|^{-1}$. More general, c_2 is approximated by the inverse of the minimum eigenvalue of $\mathbf{A}$ in absolute sense, while c_1 is determined by the amplitude of the solution multiplied by γc_2.

$k = 2$, integration-QS hybrid methods



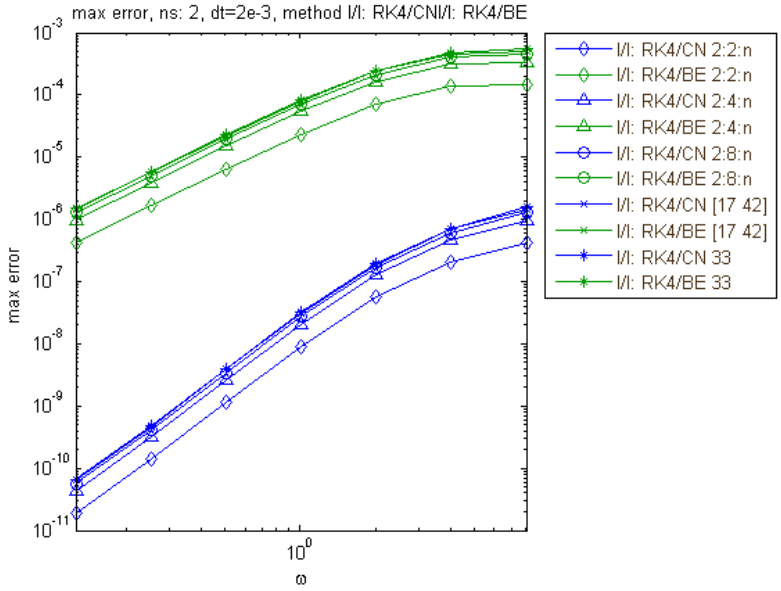
METHOD	A	B
'I/QS: RK4 2:2:n'	[1.9356]	[-14.7725]
'I/QS: RK2 2:2:n'	[1.9374]	[-17.2917]
'I/QS: CN 2:2:n'	[1.9355]	[-13.3620]
'I/QS: BE 2:2:n'	[1.9376]	[-16.9193]
'I/QS: RK4 2:4:n'	[1.9364]	[-11.6431]
'I/QS: RK2 2:4:n'	[1.9364]	[-12.0282]
'I/QS: CN 2:4:n'	[1.9366]	[-10.9415]
'I/QS: BE 2:4:n'	[1.9364]	[-11.2131]
'I/QS: RK4 2:8:n'	[1.9401]	[-9.1039]
'I/QS: RK2 2:8:n'	[1.9398]	[-9.2262]
'I/QS: CN 2:8:n'	[1.9409]	[-8.7671]
'I/QS: BE 2:8:n'	[1.9407]	[-8.8240]
'I/QS: RK4 [17 42]'	[1.9539]	[-6.6132]
'I/QS: RK2 [17 42]'	[1.9541]	[-6.6580]
'I/QS: CN [17 42]'	[1.9534]	[-6.4666]
'I/QS: BE [17 42]'	[1.9532]	[-6.4767]
'I/QS: RK4 33'	[1.9678]	[-5.6798]
'I/QS: RK2 33'	[1.9684]	[-5.7029]
'I/QS: CN 33'	[1.9656]	[-5.6012]
'I/QS: BE 33'	[1.9654]	[-5.6048]

Here, results are plotted for integration-QS hybrid methods with $k = 2$ for several grid partitions. The finer subgrid two is, the better are the results in absolute sense. Doubling the size of subgrid 2 results in smaller errors, especially when subgrid 2 is already fine. If subgrid 2 is coarse, then the effect of doubling of its size is marginal. I think a better criterion would be to look at the ratio of the sizes of subgrid 1 and subgrid 2: only for a fine subgrid 2, this ratio change heavily by doubling the size of subgrid 2.

If we look to the convergence order with respect to ω for small values of ω, then we see that this number is constant. It is neither affected by the type of the integration-QS hybrid method, nor from the grid partition. This allows us to state a general formula for ϵ of integration-QS hybrid methods, which becomes: $\epsilon = C(k)\,\omega^k$. Here, $C(k)$ depends on k, the grid partition and the integration method used, but does not depend on ω. We will specify this formula later.

We note further that for coarser subgrid 2, the difference between the solutions of the used hybrid methods becomes smaller: once more, this is caused by the small influence of the integration methods. Next we notice that there is no obvious consistency in which method is the best: probably, in some cases the error due to integration cancels out the error due to the QS assumption partly while in other case they amplify each other. For now, it can be seen as coincidence: it depends on an unknown set of parameters whether they amplify or cancel out each other. At least, we notice once more the flattening of the error for bigger ω, especially for methods with small absolute errors. Methods with fine subgrid 2 are more sensitive to this phenomenon. This is because only integration methods show this phenomenon: in periodic solutions, the error has in each iteration very less time to grow if ω is big. Therefore, the reduced number of time steps per period compensates for the increased error generated each time step. This reasoning doesn't hold for low ω, since the number of time steps per period is no bottleneck to form a maximum error. We conclude that the integration-QS hybrid methods give better results for low ω and fine subgrid 2.

$k = 2$, implicit-explicit methods



METHOD	A	B
'I/I: RK4/CN 2:2:n'	[2.9785]	[-26.7772]
'I/I: RK2/CN 2:2:n'	[3.0139]	[-23.9523]
'I/I: RK4/BE 2:2:n'	[1.9352]	[-15.3641]
'I/I: RK2/BE 2:2:n'	[1.9371]	[-17.8013]
'I/I: RK4/CN 2:4:n'	[2.9775]	[-25.5622]
'I/I: RK2/CN 2:4:n'	[2.9935]	[-26.0179]
'I/I: RK4/BE 2:4:n'	[1.9350]	[-14.1486]
'I/I: RK2/BE 2:4:n'	[1.9351]	[-14.3874]
'I/I: RK4/CN 2:8:n'	[2.9770]	[-25.1780]
'I/I: RK2/CN 2:8:n'	[2.9519]	[-26.4152]
'I/I: RK4/BE 2:8:n'	[1.9349]	[-13.7643]
'I/I: RK2/BE 2:8:n'	[1.9350]	[-13.8454]
'I/I: RK4/CN [17 42]'	[2.9759]	[-25.0091]
'I/I: RK2/CN [17 42]'	[2.9676]	[-25.3491]
'I/I: RK4/BE [17 42]'	[1.9348]	[-13.5941]
'I/I: RK2/BE [17 42]'	[1.9348]	[-13.6227]
'I/I: RK4/CN 33'	[2.9762]	[-24.9579]
'I/I: RK2/CN 33'	[2.9722]	[-25.1177]
'I/I: RK4/BE 33'	[1.9349]	[-13.5426]
'I/I: RK2/BE 33'	[1.9350]	[-13.5565]

In order to avoid a mess in the plot, we show only the RK4 hybrid methods. The results for the RK2 cases are similar. Roughly speaking, we notice the same

phenomena as in the plot above. Firstly, the finer subgrid 2 (which is responsible for the better explicit integration method), the better the results are. However, the order of convergence depends on the worst integration method, and doesn't depend on the grid partition. As we saw before, doubling of the size of subgrid 2 gives a marginal effect if subgrid 2 is rather small for the same reasons as stated above. The flattening of the plots can be explained similar as above. At first sight you might note that this flattening is stronger for BE methods. But as we correct for the order of convergence, this is not so obvious. Further, we still assume that the order of convergence with respect to ω exceeds the order of convergence with respect to Δt of the method which yields the biggest error by one. This can be explained as follows (we treat the linear case):

Consider $\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u}(t) + \mathbf{f}e^{i\omega t}$. Substituting $\mathbf{u}(t) = \mathbf{v}e^{i\omega t}$ and dividing by $e^{i\omega t}$, we obtain:

$$i\omega\mathbf{v} = \mathbf{A}\mathbf{v} + \mathbf{f}$$

$$\mathbf{v} = (i\omega I - \mathbf{A})^{-1}\mathbf{f}$$

Now consider the Euler method for

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t\mathbf{A}\mathbf{u}_n + \Delta t\mathbf{f}e^{i\omega n\Delta t}$$

Substituting $\mathbf{u}_n = \hat{\mathbf{v}}e^{i\omega n\Delta t}$ and dividing by $e^{i\omega t}$ yields:

$$\hat{\mathbf{v}} = \big(\frac{e^{i\omega\Delta t} - 1}{\Delta t}I - \mathbf{A}\big)^{-1}\mathbf{f}$$

Error:

$$\mathbf{u}(t_n) - \mathbf{u}_n = \big[(i\omega I - \mathbf{A})^{-1} - \big(\frac{e^{i\omega\Delta t} - 1}{\Delta t}I - \mathbf{A}\big)^{-1}\big]\mathbf{f}\, e^{i\omega t}$$

Bringing the part between brackets to the same denominator, we get in the numerator:

$$\big(\frac{e^{i\omega\Delta t} - 1}{\Delta t}\big)I - \mathbf{A} - (i\omega I - \mathbf{A}) =$$

$$-\frac{i\omega^2\Delta t}{2}I + O(\omega^3\Delta t^2)$$

So the convergence in ω is one order higher than that in Δt. For the more general case of forcing with more frequencies we have:

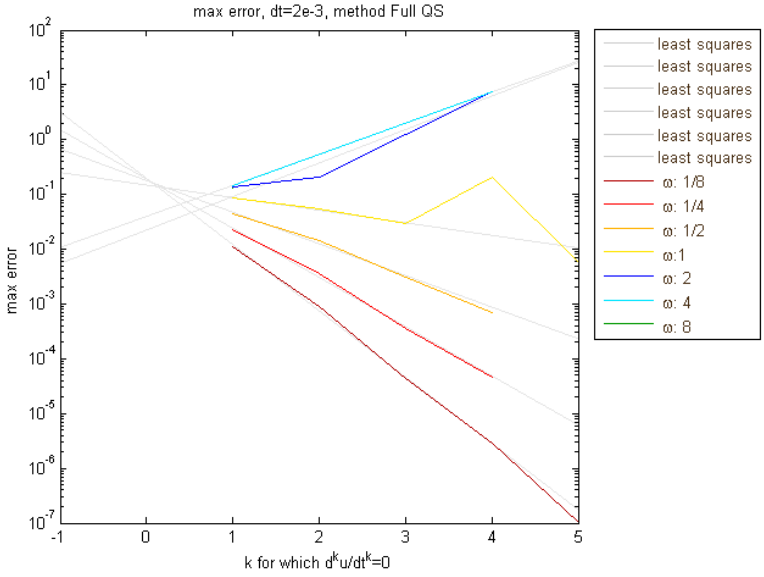
$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u} + \sum_{j=1}^{M}\mathbf{f}_j e^{i\omega_j t}$$

We pose $\mathbf{u}(t) = \sum_{j=1}^{M}\mathbf{v}_j e^{i\omega_j t} \Rightarrow \mathbf{v}_j = (i\omega_j I - \mathbf{A})^{-1}\mathbf{f}_j,\ j = 1, ..., M.$ etc A particular choice is $\mathbf{f}\sin\omega t = \mathbf{f}\left(\frac{e^{i\omega t} - e^{-i\omega t}}{2i}\right)$ where ω and $-\omega$ are involved.

5.4 Investigation of convergence w.r.t. k

Of special interest is the investigation of convergence with respect to k for integration-QS hybrid methods. Of course, we hope that we can improve these hybrid methods by choosing a higher value of k. We outline one case with low ω, a case with higher ω, and three methods in particular: the hybrid methods with RK4 and CN and the full QS method, where we begin with.

Full QS method



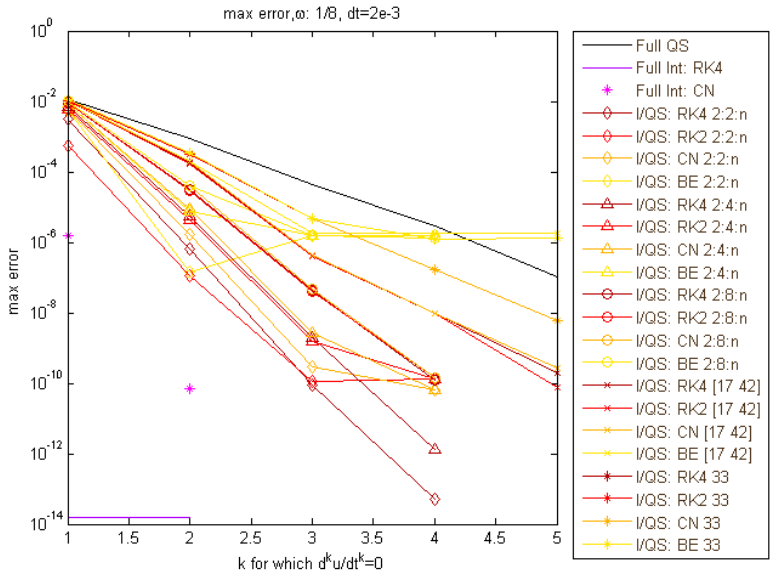
METHOD	A	B
'w: 1/8'	[-4.0177]	[-2.3230]
'w: 1/4'	[-2.9842]	[-2.3678]
'w: 1/2'	[-1.9178]	[-2.4694]
'w: 1'	[-0.7622]	[-2.7493]
'w: 2'	[2.0315]	[-5.4918]
'w: 4'	[1.8840]	[-4.6634]
'w: 8'	[NaN]	[NaN]

The data approximates $\epsilon = 2^{Ak+B}$ in a least squares sense.

The least squares solutions are added to the plot in grey. We observe that higher k not always yield better results. The order of convergence with respect to k is obviously better (more negative) for smaller ω. Moreover, in absolute sense, decreasing ω yields also smaller maximum errors. A remarkable result is

the coincidence of the least squares solutions for smaller ω at $k = 0$. It turns out that this value is approximately 0.2. In the investigation on QS methods in the previous paragraph, we already encountered this number as an upper bound for the error and the maximum error at $\omega = 2$ for all k. The fact that solutions intersect at $k = 0$, is consistent with the formula for the maximum error we derived there. It is given as $\epsilon = c_1 (c_2\omega)^k$, for some constants c_1, c_2 and ω sufficiently small. Filling in $k = 0$ or $\omega = c_2^{-1}$, we obtain $\epsilon = c_1$, which is a constant independent of ω and k. Results can be improved by choosing a higher k if ω satisfies $c_2\omega < 1$. For high values of ω, we noticed different behaviour in the previous paragraph. This might explain that the least squares solution of higher ω do not coincide in the same point in the plot above.

$\omega \approx 1/8$



METHOD	A	B
'Full QS'	[-4.0177]	[-2.3230]
'I/QS: RK4 2:2:n'	[-12.5555]	[4.3202]
'I/QS: RK2 2:2:n'	[-11.1386]	[-0.0378]
'I/QS: CN 2:2:n'	[-12.1295]	[4.8140]
'I/QS: BE 2:2:n'	[-5.9282]	[-4.6421]
'I/QS: RK4 2:4:n'	[-10.9123]	[3.9804]
'I/QS: RK2 2:4:n'	[-10.9569]	[3.7529]
'I/QS: CN 2:4:n'	[-10.7728]	[4.2000]

```
'I/QS: BE 2:4:n'        [     -6.2152]    [ -1.9672]
'I/QS: RK4 2:8:n'       [     -8.8851]    [  2.3761]
'I/QS: RK2 2:8:n'       [     -8.8591]    [  2.2471]
'I/QS: CN 2:8:n'        [     -8.8665]    [  2.5281]
'I/QS: BE 2:8:n'        [     -6.3059]    [ -0.9049]
'I/QS: RK4 [17 42]'     [     -7.3320]    [  1.2375]
'I/QS: RK2 [17 42]'     [     -7.3222]    [  1.1905]
'I/QS: CN [17 42]'      [     -7.3012]    [  1.2695]
'I/QS: BE [17 42]'      [     -6.2424]    [ -0.1456]
'I/QS: RK4 33'          [     -5.5657]    [ -0.7842]
'I/QS: RK2 33'          [     -5.5590]    [ -0.8106]
'I/QS: CN 33'           [     -5.5709]    [ -0.7376]
'I/QS: BE 33'           [     -5.5803]    [ -0.7261]
```

We first have to note that the data above should be seen as a very rough indication for CN and BE methods, since the error is far from linear with respect to k. Furthermore, if $k = 1$, the CN method is in fact a BE method. The pink star at $k = 1$ therefore represents the full BE integration method, while the other pink star represents the full CN integration method.

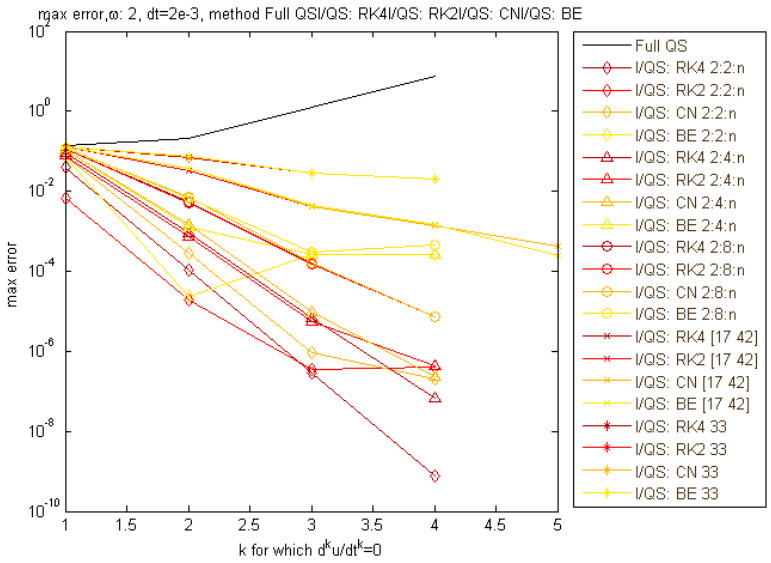
In general, the results are better for higher k. From the plot we conclude that a finer subgrid 2 gives not only rise to a smaller error in absolute sense, but also to a lower convergence order with respect to k. Thus, the reduction of the error for higher k cannot be fully caused by the QS part of the methods. For QS hybrid methods and small ω we obtain $\epsilon = C(k)\,\omega^k$, where $C(k)$ may depend on k, but not on ω. Since the order of convergence with respect to k is highly dependent of the grid partition and the integration method used, the same holds for $C(k)$. We can specify the formula as: $\epsilon = c_1\,(c_2\omega)^k$, just as in the case of the full QS method. c_1 might depend on the grid partition and the integration method, but is independent of k and ω. The same holds for c_2. We can verify this formula via $\epsilon = C(k)\,\omega^k$, from where we show that $C(k) = c_1\,c_2^k$. This latter equation follows from the following two observations. Firstly, for a hypothetic $k = 0$, the error will be independent of ω, since a previous result revealed that the order of convergence with respect to ω equals k. This causes c_1 to be independent of ω. Secondly, for all integration methods and grid partitions used, from least squares extrapolation over nodes with ω small, there seems to be a value of ω for which the error is constant with respect to k. This value does not depend on k nor on ω, which causes c_2 to be independent of k and ω. See also 5.4.4. From $\epsilon = c_1\,(c_2\omega)^k$, we expect the lines corresponding to a specific grid partition and integration method to intersect at $k = 0$. This is indeed visible in the plot above.

In absolute sense, the results are better for higher k. We notice however, that the error of the BE method stagnates after a while. This can also been seen for the CN and RK2 method, albeit that this stagnation occurs from higher

k and that this stagnation error is smaller. The value of this stagnation error seems to be independent of the grid partition: for BE methods this is approximately of the order 10^{-8}. If k is big, the corresponding integration methods seems to be the bottleneck to form the maximum error. In other words, the error of the integration part then dominates the error. The errors of the full integration methods support this. For BE, this number is 10^{-8}, while for CN it is approximately 10^{-10}, which are the same values as the stagnation errors for the corresponding hybrid methods. Therefore, we expect that for sufficient high k, the stagnation error for the RK4 hybrid method is 10^{-14}.

From these observations we conclude that if ω is small, then increasing k gives better results if ω satisfies $c_2\omega < 1$, where c_2 is from a general formula for the maximum error: $\epsilon = c_1\,(c_2\omega)^k$. $c_2\omega < 1$ is easier satisfied if the subgrid to which integration is performed is big, since then c_2 is rather small. However, the error of the integration-QS hybrid method is restricted to be at least as big as the maximum error of the corresponding full integration method (up to cancellation of errors). To obtain this optimum error, we either choose a large grid on which integration is performed and set k to a moderate value, or we choose a large k.

$\omega \approx 2$



METHOD	A	B
'Full QS'	[2.0315]	[-5.4918]
'I/QS: RK4 2:2:n'	[-8.5245]	[3.8507]
'I/QS: RK2 2:2:n'	[-7.1205]	[-0.5258]

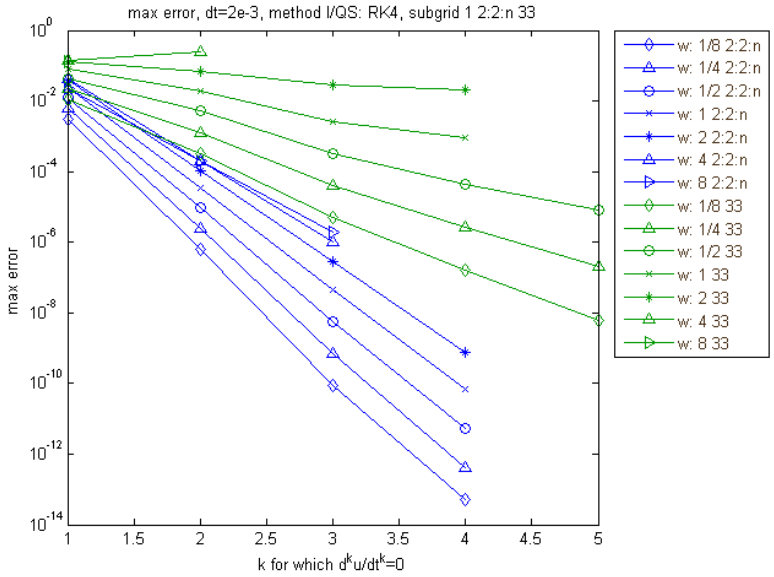
```
'I/QS: CN 2:2:n'        [-8.1352]    [ 4.4166]
'I/QS: BE 2:2:n'        [-4.0889]    [-2.1596]
'I/QS: RK4 2:4:n'       [-6.8732]    [ 3.5215]
'I/QS: RK2 2:4:n'       [-6.9111]    [ 3.2874]
'I/QS: CN 2:4:n'        [-6.7330]    [ 3.7274]
'I/QS: BE 2:4:n'        [-4.3618]    [ 0.4747]
'I/QS: RK4 2:8:n'       [-4.8066]    [ 1.8504]
'I/QS: RK2 2:8:n'       [-4.7857]    [ 1.7347]
'I/QS: CN 2:8:n'        [-4.7831]    [ 1.9867]
'I/QS: BE 2:8:n'        [-4.3399]    [ 1.3762]
'I/QS: RK4 [17 42]'     [-2.5009]    [-0.2663]
'I/QS: RK2 [17 42]'     [-2.4909]    [-0.3086]
'I/QS: CN [17 42]'      [-2.5014]    [-0.2012]
'I/QS: BE [17 42]'      [-2.4692]    [-0.2478]
'I/QS: RK4 33'          [-1.1201]    [-1.7271]
'I/QS: RK2 33'          [-1.1150]    [-1.7467]
'I/QS: CN 33'           [-1.1262]    [-1.6905]
'I/QS: BE 33'           [-1.1235]    [-1.6957]
```

Since the previous conclusion holds only for small ω, we investigate the case for the relatively high $\omega = 2$. Roughly, the same observations are visible, but some remarks have to be made.

With regard to the previous result, the order of convergence with respect to k becomes worse for all QS methods. In particular, we see that the full QS method gives even worse errors for higher k in absolute sense. In the formula for the maximum error $\epsilon = c_1 (c_2\omega)^k$, c_2 is much bigger such that ω must be small in order to satisfy $c_2\omega < 1$. Next we note that the stagnation errors for the hybrid methods are much higher. In the previous investigation about convergence with respect to ω can be read why this is holds for increasing ω.

Thus, if ω is quite big, then the QS hybrid methods are less attractive with regard to lower ω. However, for a sufficient big k and a sufficient large size of the subgrid to which integration is performed, these methods are still as good as the corresponding integration methods alone. In combination with an accurate integration method, hybrid methods can still be attractive.

Integration-QS hybrid method with RK4



METHOD	A	B
'w: 1/8 2:2:n'	[-12.5555]	[4.3202]
'w: 1/4 2:2:n'	[-11.5395]	[4.3010]
'w: 1/2 2:2:n'	[-10.5137]	[4.2543]
'w: 1 2:2:n'	[-9.4938]	[4.1436]
'w: 2 2:2:n'	[-8.5245]	[3.8507]
'w: 4 2:2:n'	[-7.6368]	[3.0481]
'w: 8 2:2:n'	[-6.7967]	[1.3630]
'w: 1/8 33'	[-5.5657]	[-0.7842]
'w: 1/4 33'	[-4.5508]	[-0.8042]
'w: 1/2 33'	[-3.5234]	[-0.8536]
'w: 1 33'	[-2.4778]	[-0.9869]
'w: 2 33'	[-1.1201]	[-1.7271]
'w: 4 33'	[0.7547]	[-3.5499]
'w: 8 33'	[NaN]	[NaN]

We focus on the RK4 hybrid method for a fine and coarse integration subgrid and different values of ω. The ω are such that $\omega \approx 2^{we}$, hence we consider $\omega = \frac{1}{8}$ till $\omega = 8$.

From the plot we see that for $\omega \approx 8$, the method gives no results for higher k. Probably, the threshold value for the Newton procedure was the problem.

The Newton procedure could not converge up to this threshold value.

Once more, we observe that increasing the size of subgrid 2, or decreasing ω, results in a better convergence order with respect to ω and in a smaller maximum error in absolute sense. Also, the formula for the maximum error $\epsilon = c_1 \, (c_2\omega)^k$ we derived earlier seems to hold here. All families of lines with the same grid partition seems roughly to coincide in one single point outside the plot at $k \approx 0$. Moreover, c_2 seems to be independent of k, as can be verified from figures 3-5. Focusing on the green lines (subgrid 2: [33]) in the plot above, we see that for $\omega \approx 3$, the maximum error becomes constant with respect to k. This means that $c_2 \approx 1/3$ for subgrid 2: [33] and RK4 as integration method. c_1 is of the order 0.1. In the following figures c_1 and c_2 are determined for several grid partitions using an alternative approach, namely we plot ϵ against ω instead of k and extrapolate the results from low values of ω to higher ones by means of a least squares approach. c_2 is independent of k (for small ω) if the least squares lines coincide in one point. For each grid partition, this is indeed the case (except possibly for higher values of k, in which case instability causes an irregular course of the graph). The coordinates of the intersection point equals $(\omega, \epsilon) = (c_2^{-1}, c_1)$. We conclude that a finer integration subgrid yields a smaller c_2 and a (slightly) bigger c_1.

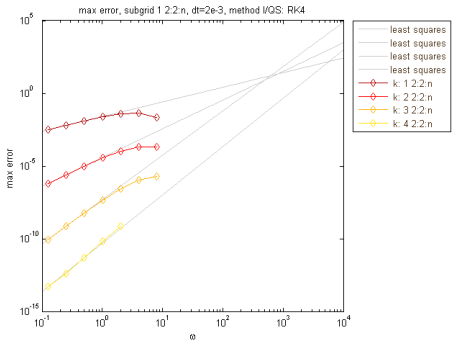


Figure 3: ϵ plotted against ω for several k and subgrid 2: 2:2:n. The lines for k small enough cross each other at $\omega \approx 700$. We have $c_2 \approx 1/700$ and $c_1 \approx 20$.

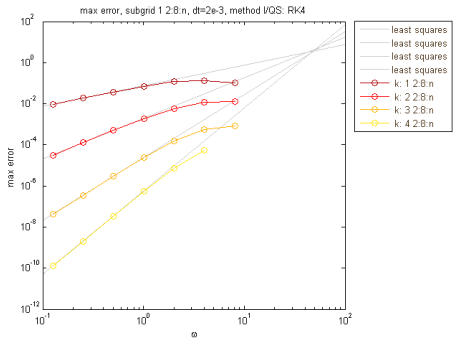


Figure 4: ϵ plotted against ω for several k and subgrid 2: 2:8:n. The lines for k small enough cross each other at $\omega \approx 45$. We have $c_2 \approx 1/45$ and $c_1 \approx 3$.

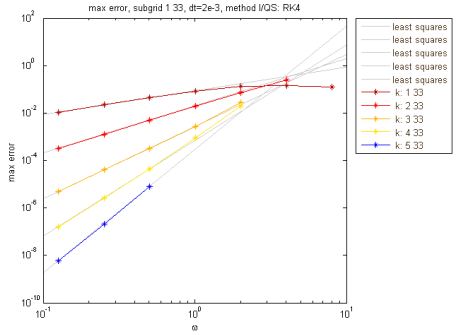
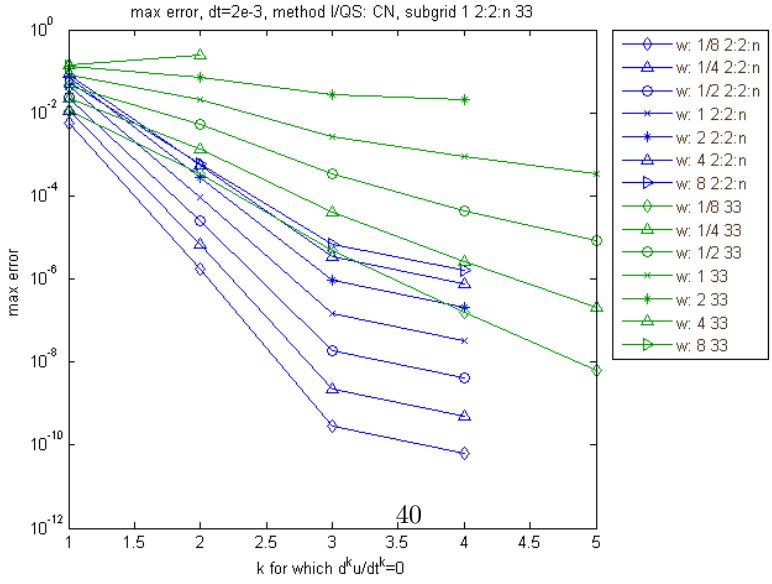


Figure 5: ϵ plotted against ω for several k and subgrid 2: [33]. The lines for k small enough cross each other at $\omega \approx 5$. We have $c_2 \approx 1/5$ and $c_1 \approx 0.5$.

Integration-QS hybrid method with CN



METHOD	A	B
'w: 1/8 2:2:n'	[-12.1295]	[4.8140]
'w: 1/4 2:2:n'	[-11.1238]	[4.8074]
'w: 1/2 2:2:n'	[-10.0990]	[4.7614]
'w: 1 2:2:n'	[-9.0845]	[4.6613]
'w: 2 2:2:n'	[-8.1352]	[4.4166]
'w: 4 2:2:n'	[-7.3062]	[3.7655]
'w: 8 2:2:n'	[-6.4897]	[2.2532]
'w: 1/8 33'	[-5.5709]	[-0.7376]
'w: 1/4 33'	[-4.5559]	[-0.7578]
'w: 1/2 33'	[-3.5283]	[-0.8082]
'w: 1 33'	[-2.4823]	[-0.9446]
'w: 2 33'	[-1.1262]	[-1.6905]
'w: 4 33'	[0.7733]	[-3.5604]
'w: 8 33'	[NaN]	[NaN]

The results for the hybrid method with CN are very similar to the previous results. For this method too, the lines seems to coincide at $k \approx 0$ and the formula used in the previous result, can be applied to this method as well. Note that for the the blue lines (subgrid 2: 2:2:n), the maximum error reaches the stagnation error for high k. Also in this case, for $\omega \approx 2$ (maybe a little higher), the error becomes independent of k, which determines c_2. Also other values of c_1 and c_2 are comparable to the previous result. We observe that $\epsilon = c_1 \, (c_2\omega)^k$ fits the integration-QS hybrid method very well until now. From the observations, we saw that a finer integration subgrid gives rise to a (much) smaller c_2, while c_1 might become (slightly) bigger. This is important, since for higher ω, we can still obtain a maximum error which is as good as the separate integration method by choosing a fine subgrid, which causes c_2 to be small, such that $c_2 \, \omega < 1$. In that case, by choosing k sufficiently big, the error becomes small enough.

6 Conclusion and discussion

In this thesis, we considered several methods to solve non-linear PDEs such as the Burgers' equation. These methods include integration methods, (quasi) stationary methods (QS methods), integration hybrid methods and integration-QS hybrid methods. For all these methods, we derived a manner to put equations into a system which can be solved by a Newton procedure, which is a sequence of Newton steps. In calculating the state, the Jacobian of the system plays an important role.

Stability From the stability investigation, it follows that the full QS method is always stable. The stability of integration-QS hybrid methods can be improved by choosing a bigger k or a finer subgrid to which integration is performed.

A remarkable observation is that the forward Euler hybrid method performs better than the RK2 and RK4 hybrid method. If we focus on the integration hybrid methods (implicit-explicit methods), we notice a roughly equal stability interval. This appeared to be due to the used implementation. Another variant is possible which is expected to have the same behaviour as the Forward Euler method.

Convergence with respect to Δt From the investigation on convergence with respect to the time step Δt, we observed no convergence of all (quasi) stationary (hybrid) methods. This is due to the independence of the first order derivatives with respect to time of these methods. However, for higher values of Δt, the error due to the integration part becomes more and more significant with regard to the error due to the QS method. Probably, in the cases we considered, these errors cancel out each other partly, for higher Δt. For all Δt where the error due to the QS part dominates, a finer subgrid to which integration is performed, results in smaller errors in absolute sense. For implicit-explicit methods, we saw that the order of convergence with respect to Δt is determined by the integration method which yields the biggest error in absolute sense.

Convergence with respect to ω From the investigation on convergence with respect to ω, we obtained evident results for the order of convergence in the case of small ω, but when ω is big, we observe smaller maximum errors than we might expect. Maybe the error has the same magnitude as the solution itself, and simply cannot become much bigger. However, this flattening was at least as good visible for methods with very small absolute errors at high ω. Another explanation is that for high ω, the number of time steps in each period is very small, and might become a bottleneck to form the maximum error. For small ω, the order of convergence with respect to ω turns out to be k for all QS methods. These methods satisfy $\epsilon = c_1 (c_2\omega)^k$, where c_1 and c_2 depend on the grid partition and the integration method, but do not depend on ω and k. In general, a finer subgrid to which integration is performed yields a smaller c_2. A similar result doesn't hold for integration methods: a better integration method does not yield always a smaller c_2. If there are only few nodes on which integration is performed, the maximum error is mainly determined by the QS part. In that case, the error is hardly affected by the integration part. Furthermore, c_1 seems to become slightly bigger for a finer integration subgrid, but overall ϵ becomes smaller for a finer integration subgrid. For integration (hybrid) methods, we see that the order of convergence is equal to $m+1$ where m is the order of convergence with respect to Δt of the method which yields the biggest error. However, it seems that there is some limit for this order, which I have not investigated so far.

Convergence with respect to k At least, the convergence with respect to k has been investigated for the QS methods. For the full QS method, the relation

between the log of the maximum error and k is linear. For sufficiently small ω, higher k yield a smaller maximum error, while for higher ω, the results become worse for increasing k. From $\epsilon = c_1 (c_2\omega)^k$, we know that ω must satisfy $c_2\omega < 1$ in order to have improvement for higher k. So in theory, for solutions without high frequency elements (in its Fourier form), the maximum error becomes small enough for k sufficiently large. Integration-QS methods show a similar result. However, the convergence order with respect to k depends not only on ω, but also on the grid partition and the integration method via c_1 and c_2. In general, better integration methods yield a better order of convergence. The same holds true for a lower ω and a finer subgrid to which integration is performed. In general, an instance with better order of convergence yields a smaller maximum error in absolute sense. In the plots we see that all lines coincide at some point at $k \approx 0$ and that most lines don't cross each other. Another important observation from these hybrid methods, is the existence of a lower limit of the maximum error, which is equal to the maximum error of the corresponding integration method.

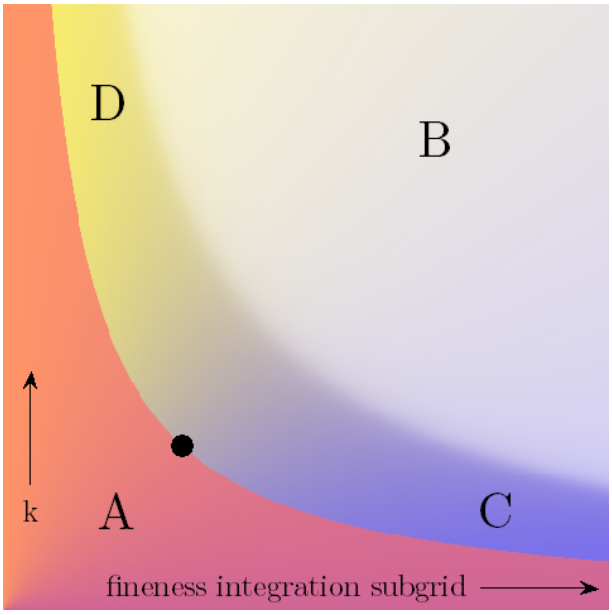


Figure 6: Finding the balance between k and the fineness of the subgrid to which integration is performed in integration-(quasi) stationary hybrid methods. In the region outside A the error is sufficiently small. B denotes the area where the integration part determines the (minimum value of the) maximum error. In region C the communication between processors is a bottleneck, whereas in region D the number of flops per Newton step per processor forms a bottleneck. The black dot indicates an optimal value. By increasing Δt, the region B will move towards A, while A remains roughly constant. Δt is optimal if the black dot is also on the boundary of B.

General conclusion In short, integration-(quasi) stationary hybrid methods can be a helpful tool in solving PDEs on a grid which is divided into subgrids that are linked to one processor each. The main advantage of these methods with regard to usual methods relies on the ability of parallel computing, while the accuracy can be as good as the integration method used on one subgrid. However, the coarser the subgrid to which integration is performed, the bigger the value of k must be in order to reach the desired accuracy. As is illustrated in figure 6, a balance between these factors must be made, since a finer subgrid to which integration is performed requires more communication between processors, which is easily a bottleneck in the process (region C), while increasing k requires a highly increased number of flops in each Newton step for each processor (region D). The PDE must have a solution which is sufficiently smooth in time in order to find values for the above-mentioned parameters that yield faster computing than usual methods, while the accuracy is the same. Thus, given some desired value for the accuracy and some integration method, we perform the following steps: firstly choose an appropriate Δt for the separate integration method which we use for the hybrid method too. In the optimal case we have then that the minimum error caused by the integration part is equal to maximum allowed error (in figure 6: the boundaries of regions A and B coincide). Secondly, set up a balance between k and the grid partition such that the errors made by the two parts are roughly equal and no severe bottleneck occurs in the two processes of computing time. Then, the set of freely chosen parameters of this integration-(quasi) stationary method is optimized. Another type of methods which is also suitable for parallel computing, is the family of implicit-explicit methods. A disadvantage of this type is that the maximum error is determined by the worst of the separate integration methods used.

Future research In this thesis, I considered a one-dimensional case. However, real life processes are more dimensional. A proposal for future investigation is therefore to study the usefulness of hybrid methods described in this thesis for parallel computing in 2D or 3D. Also, we made some remarkable observations which we could not fully explain. Among these is the observation that the order of convergence with respect to ω for integration hybrid methods equals $m + 1$ and attains some limit which became visible for the RK4 hybrid method, as is described above. Also, RK4 and RK2 hybrid methods yield stability values which are worse than those of the Forward Euler hybrid method, which probably suggests some error in the method. Can we improve the results by solving a system in each stage of multistage method, as is described in section 4? Another question is on the apparent flattening of the maximum error for higher ω. What happens if ω becomes even bigger? Can integration-(quasi) stationary hybrid methods still be a helpful tool in calculating a PDE which has a solution with high frequencies? At least we noted that integration-(quasi) stationary hybrid methods became unstable for higher k, probably because the tolerance value in the Newton process was too strict. If we increase this value, do we yield indeed stable solutions? How does that affect the accuracy of the solutions?

7 Appendices

A Linear case

According to another (preliminary) study [1] which makes also use of elimination, an explicit result is given in the case that $\beta_1 = 1, \beta_2 = 0$, with forward Euler as integration method. Furthermore, k is set to 2 and only a linear case is treated (i.e. $\mathbf{B} = \mathbf{0}$). I would like to show that this outcome is consistent with the theory described in this thesis. Firstly, a linear case makes life far more easy. After exactly one Newton step, $\mathbf{F}(\mathbf{y})$ becomes zero and the solution is computed. Since the Jacobian is in this case independent from the state $\mathbf{y}$ (and therefore constant), we can write $\mathbf{F}(\mathbf{y}) = J\mathbf{y} + \hat{f}$, for some $\hat{f}$ which depends only on time and the known state $\mathbf{y}^j$ at time j. In the notation of section 4, we can set up $\mathbf{y} = (u_1^T, u_2^T, v_1^T, v_2^T)^T$, as well as the Jacobian J and $\mathbf{F}$:

$$J = \begin{pmatrix} A_{11} & A_{12} & -I & 0 \\ A_{21} & A_{22} & 0 & -I \\ 0 & 0 & A_{11} & A_{12} \\ 0 & -I & 0 & 0 \end{pmatrix} \tag{10}$$

$$\mathbf{F} = \begin{pmatrix} -\mathbf{v}_1 + A_{11}\mathbf{u}_1 + A_{12}\mathbf{u}_2 + f_1(t + \Delta t) \\ -\mathbf{v}_2 + A_{21}\mathbf{u}_1 + A_{22}\mathbf{u}_2 + f_2(t + \Delta t) \\ A_{11}\mathbf{v}_1 + A_{12}\mathbf{v}_2 + f_1'(t + \Delta t) \\ -\mathbf{u}_2 + \mathbf{u}_2^j + \Delta t \mathbf{v}_2^j \end{pmatrix}$$

This yields the following form of $\hat{f}$:

$$\hat{f}(t + \Delta t) = \begin{pmatrix} f_1(t + \Delta t) \\ f_2(t + \Delta t) \\ f_1'(t + \Delta t) \\ \mathbf{u}_2^j + \Delta t \mathbf{v}_2^j \end{pmatrix}$$

Since $\mathbf{F}(\mathbf{y}) = J\mathbf{y} + \hat{f}(t + \Delta t)$ and the solution $\mathbf{y}$ must satisfy $\mathbf{F}(\mathbf{y}) = 0$, we simply compute $\mathbf{y} = -J^{-1}\hat{f}(t + \Delta t)$ to find it. This is equivalent to the result obtained from the Newton procedure after the first step: $\mathbf{y}_1 = \mathbf{y}^j - J^{-1}\mathbf{F}(\mathbf{y}^j) = \mathbf{y}^j - J^{-1}(J\mathbf{y}^j + \hat{f}(t + \Delta t)) = -J^{-1}\hat{f}(t + \Delta t)$. The inverse of the Jacobian assumes the following form:

$$J^{-1} = \begin{pmatrix} PA_{11} & PA_{12} & P & PQ \\ 0 & 0 & 0 & -I \\ A_{11}PA_{11} - I & A_{11}PA_{12} & A_{11}P & A_{11}PQ - A_{12} \\ A_{21}PA_{11} & A_{21}PA_{12} - I & A_{21}P & A_{21}PQ - A_{22} \end{pmatrix}$$

where $P = (A_{11}^2 + A_{12}A_{21})^{-1}$ and $Q = (A_{11}A_{12} + A_{12}A_{22})$. The second row of the inverse Jacobian reveals that in each time step, $\mathbf{u}_1, \mathbf{v}_1$ and $\mathbf{v}_2$ can be expressed in terms of $\mathbf{u}_2$. We get the following solution for $\mathbf{y}$:

$$\begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{v}_1 \\ \\ \mathbf{v}_2 \\ \\ \end{pmatrix} = \begin{pmatrix} -PA_{11}f_1 - PA_{12}f_2 - Pf_1' - PQ(\mathbf{u}_2^j + \Delta t\mathbf{v}_2^j) \\ \mathbf{u}_2^j + \Delta t\mathbf{v}_2^j \\ -(A_{11}PA_{11} - I)f_1 - A_{11}PA_{12}f_2 - A_{11}Pf_1' \ldots \\ -(A_{11}PQ - A_{12})(\mathbf{u}_2^j + \Delta t\mathbf{v}_2^j) \\ -A_{21}PA_{11}f_1 - (A_{21}PA_{12} - I)f_2 - A_{21}Pf_1' \ldots \\ -(A_{21}PQ - A_{22})(\mathbf{u}_2^j + \Delta t\mathbf{v}_2^j) \end{pmatrix}$$

where all (derivatives of) forced terms f_i are evaluated at $t+\Delta t$. Now define, in accordance with the notations used in [1] the following matrices:

$$B = -A_{22} + A_{21}A_{11}^{-1}A_{12}$$
$$C = I + A_{21}A_{11}^{-2}A_{12}$$
$$E = [\ A_{21}(A_{11}^{-1} + A_{11}^{-2}\tfrac{d}{dt}) \quad I\ ]$$

It can be shown that $\mathbf{v}_2$ in the solution for $\mathbf{y}$ can be equivalently written as:

$$\mathbf{v}_2 = -C^{-1}A_{21}A_{11}^{-1}f_1 + C^{-1}f_2 - C^{-1}A_{21}A_{11}^{-2}f_1' - C^{-1}B(\mathbf{u}_2^j + \Delta t\mathbf{v}_2^j)$$

which is in more compact form equal to:

$$\mathbf{v}_2 = C^{-1}(E\tilde{f} - B\mathbf{u}_2) \tag{11}$$

where $\tilde{f} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$. We will now show this, where we often make use of the rule $S(TS)^{-1} = (ST)^{-1}S$ for some matrix S and invertible matrices ST and TS. Unfortunately, this assumption is very restricting. In fact, if S is non-square, then either ST or TS has to be non-invertible.

- $A_{21}PA_{11} = C^{-1}A_{21}A_{11}^{-1}$:
  $A_{21}P = A_{21}(A_{11}^2 + A_{12}A_{21})^{-1} = A_{21}(I + A_{11}^{-2}A_{12}A_{21})^{-1}A_{11}^{-2} = (I + A_{21}A_{11}^{-2}A_{12})^{-1}A_{21}A_{11}^{-2} = C^{-1}A_{21}A_{11}^{-2}$
- $A_{21}PA_{12} - I = -C^{-1}$:
  $A_{21}PA_{12} + C^{-1} = A_{21}A_{11}^{-2}A_{12}C^{-1} + C^{-1} = (A_{21}A_{11}^{-2}A_{12} + I)C^{-1} = I$
- $A_{21}P = C^{-1}A_{21}A_{11}^{-2}$
- $A_{21}PQ - A_{22} = C^{-1}B$:
  $A_{21}PQ - A_{22} = A_{21}PA_{11}A_{12} + A_{21}PA_{12}A_{22} - A_{22} = C^{-1}A_{21}A_{11}^{-1}A_{12} + (A_{21}PA_{12} - I)A_{22} = C^{-1}A_{21}A_{11}^{-1}A_{12} - C^{-1}A_{22} = C^{-1}B$

Note that (10) is the result obtained from [1]. To verify the other parts of the solution of $\mathbf{y}$, note that $\mathbf{y}$ implies that:

$$\mathbf{u}_2 = \mathbf{u}_2^j + \Delta t\mathbf{v}_2^j$$

$$\mathbf{v}_1 = A_{11}\mathbf{u}_1 + A_{12}\mathbf{u}_2 + f_1$$
$$\mathbf{v}_2 = A_{21}\mathbf{u}_1 + A_{22}\mathbf{u}_2 + f_2$$

which completes the proof.

In our derivation, we didn't make use of the boundary conditions. If we include the boundary conditions, then the Jacobian (9) assumes another form since there is no -1 in the upper right block of the matrix in the rows corresponding to the boundaries. For Dirichlet boundary conditions, this won't influence the results, since it is previously defined and is not affected by the state on other nodes. For Neumann boundary conditions this reasoning doesn't hold, however we can eliminate the corresponding row to obtain a Jacobian of the form (9) with possibly an additional row corresponding to a Dirichlet boundary condition.

B The code

Several programs have been used, including a lot of controller programs. We will restrict us to the main programs, all of which have the goal to compute the solution of some system of ODEs, hereby making use of several methods described in this paper. The first program, EN, deals with systems of the form given in section 4 (θ-method). ENRK4 and ENRK2 do the same, but use RK4 and RK2 as standard explicit integration method, respectively.

B.1 EN

The main program is EN, which stands for 'elimination' and 'non-linear'. The input parameters are:

n: the number of nodes in the grid
uinit: the initial state.
dt: the time-step which is used for integration methods.
dur: duration of integration.
thres: threshold for the absolute value of the residual in the Newton steps.
ns: the number of subsystems, this is such that the Jacobian is of size ns*n $\times$ ns*n. Hence, $ns = k$.
bctype: a 2×1 vector with information about the type of boundary condition. Entries can assume the value 0 for a Dirichlet boundary condition, or a 1 for a Neumann boundary condition. Here the first entry in this vector represents the left boundary; the second entry the right one.
bc: a cell with the value of the boundary condition (given as a function handle), and its derivatives (at least ns derivatives of the boundary conditions are needed).
A: the matrix A of the PDE.
B: the matrix B of the PDE.

$falg$: a cell with the value of the forced term in the PDE (given as a function handle), and its derivatives (at least ns derivatives of the boundary conditions are needed).
$t0$: initial time.
$index1$: indices of the grid nodes which defines the first subgrid
$b1$: can assume the value 0 or 1, which stands for use of an integration method for the first subgrid, or use of a (quasi) stationary assumption for this grid, respectively. Hence, $b1 = \beta_1$.
$b2$: same as b1, for the second subgrid. $b2 = \beta_2$.
$a1$: can assume the value 0 or 1, which stands for explicitly or implicitly use of u_1 in the PDE part. Is always set to 1. $a1 = \alpha_1$.
$a2$: same as a1, for u_2. Is always set to 1. $a2 = \alpha_2$.
$t1$: theta value for the theta method, which is used if an integration method is used for u_1.
$t2$: same as t1, for u_2.

The output parameters are:

Y: the state at time $t0 + dur$.
$stab$: can assume the value 0 or 1, which stands for a stable or unstable solution, respectively.

The most important local variables are:

edl: variable which assumes the value 1 or 0 if for the left boundary, a Dirichlet condition is used or not, respectively.
enl: variable which assumes the value 1 or 0 if for the left boundary, a Neumann condition is used or not, respectively.
edr: variable which assumes the value 1 or 0 if for the right boundary, a Dirichlet condition is used or not, respectively.
enr: variable which assumes the value 1 or 0 if for the right boundary, a Neumann condition is used or not, respectively.
$index1r$: $index1$ without boundary points.
$index2r$: $index2$ without boundary points.
P: permutation matrix which permutes the grid in such a way that the nodes which belong to index1 are placed before the other nodes.
Axx, Bxx: block matrices of the decomposition of A and B.
$f1r, f2r$: cell which contains (the derivatives of) the forced term $falg$, as function handles for t, but already evaluated at the known grid points of the two subgrids.
$Ibc1, Ibc2$: matrices which can be used for selecting the rows of matrices corresponding to subgrids 1 and 2, respectively.
$ddt1, ddt2$: same as $Ibc1, Ibc2$, but multiplied by dt.
f: cell which contains (the derivatives of) the forced term $falg$, as function handles for t, but already evaluated at the known grid points of the reduced grid (without boundary points).

uinit: $n \times k$ matrix containing the initial state values and their k derivatives. If only the state is given in uinit as input, while the derivatives are not given (i.e. uinit is given as $n \times 1$ vector), the derivatives are computed in the program.
y: permuted state, adapted for the Newton steps.
$U1, U2$: current state evaluated on the first and second subgrid at the end of each Newton step.
$u1o, u2o$: old state from the last time step (before each Newton step iteration).
$v1o, v2o$: old first derivative of the state from the last time step (before each Newton step iteration).
$U1o, U2o$: equal to $U1$ and $U2$, except for the first column, which contains the old state. It is only used for $U1c$ and $U2c$, respectively.
$U1c, U2c$: for $i \in \{1, 2\}$, Uic is equal to Ui if $ai = 1$ and equal to Uio if $ai = 0$. As already stated, ai is always taken 1, therefore in all of our cases $Uic = Ui$.
$G1, G2$: equivalent of $\psi_{j,i,l}$, without its binomial coefficients.
$H12, H21$: equivalent of $\eta_{j,i,l}$, without its binomial coefficients.
$nvecl, nvecr$: auxiliary vector. If Neumann boundary condition is used, then a 1 or -1 are coupled to the corresponding boundary node (left and right, respectively) and their neighbouring nodes.
$nvecl, nvecr$: auxiliary vector. If Dirichlet boundary condition is used, then a 1 is coupled to the corresponding boundary node (left and right, respectively).
P: permutation matrix which permutes the grid in such a way that the boundary nodes are placed at appropriate places in the matrix.
Jpre: constant part of the Jacobian.
stab: test for stability: if the state doesn't converge after performing 100 Newton steps, the program considers the input data as an unstable combination.
bceval: the values of the boundary conditions evaluated at t.
F: the function in the Newton steps which has to become zero.
J: the Jacobian matrix as in (8), used in the Newton steps.
$v1c, v2c$: a combination of the first derivative on the old state (at time t) and the new state (at time $t + dt$), determined by θ_1 and θ_2.

The program consists of the following parts:

- *Adjusting to the bc's*: the first and last rows of A and B are eliminated since these rows correspond to the boundary nodes, for which an equation is already defined. Also, a permutation matrix P is defined in order to permute the nodes in such an order that the nodes which belong to $index1$ are placed before the other nodes.

- *Subgrids*: A and B are partitioned into 4 blocks as outlined in section 4. Note that the rows corresponding to boundary nodes are not included, while the corresponding columns are. Also, $falg$ is adapted for the two subgrids (without boundary nodes). At least, two diagonal matrices are defined with dt on the diagonals, except for entries which correspond to boundary nodes. These entries assumes a zero.

- *Computing initial derivatives*: The pde, included all derivatives of the

forced terms is defined first. Then we distinguish two cases. The initial condition can be given as input parameter either with or without its derivatives. In the first case, *uinit* is a $n \times k$ matrix, otherwise it is an n vector. In each case, the columns of uinit are premultiplied by permutation matrix P in order to let the entries of *uinit* correspond to the (nodes which correspond to the) rows of the reordered matrix A. The result is the initial state y. Via *pde*, the derivatives can be determined on the interior nodes. The boundary nodes are determined via the derivatives of the boundary conditions (which is given in *bc*). At least, the initial state (plus its derivatives) on the subgrids are determined.

- *Defining some functions and vectors*: Some functions which appear in the Jacobian are defined as function handles. In the Jacobian, the rows which correspond to a Dirichlet or Neumann boundary condition, contains a '1' or a '-1' together with a '1', respectively. These rows are stored in *nvecl*, *nvecr*, *dvecl* and *dvecr*, where d stands for Dirichlet, n for Neumann, l for left and r for right. The vector becomes zero if the boundary condition they stand for is not on the corresponding boundary. Hence, if we add the two vectors corresponding to one boundary, we get the right row in the Jacobian. Another important permutation matrix is $P2$. In the Jacobian, the boundary conditions are placed in the bottom rows. Via $P2$, all rows are reordered such that row i correspond to the same node as column i.

- *Constructing the constant part of the Jacobian* All blocks in the Jacobian which are independent of the state, are predefined. These blocks includes A_{11}, A_{12}, A_{21}, A_{22} and some identity block matrices (multiplied by dt).

- *Performing Newton steps*: First, the Jacobian is fully defined. Then the Newton step is performed in row 159. After that, the state on the two subgrids are updated. At least, F and its norm are computed in order to verify if the computed state is accurate enough. If the state fails to converge, then after some predefined maximum number of steps, the program qualifies the current problem to be unstable.

- *Evaluating F*: F is computed according to what is outlined in section 4. In an earlier version of this thesis, it was (wrongly) stated that if $k = 1$, then θ_1 and θ_2 must be zero. In the program, this is forced. In the other cases, $v1c$ and $v2c$ are a combination of the first derivative on the old state (at time t) and the new state (at time $t + dt$), determined by θ_1 and θ_2.

```
1 function [Y,stab,J,P2]=EN(n,uinit,dt,dur,thres,ns,bctype,...
2 bc,A,B,falg,t0,index1,b1,b2,a1,a2,t1,t2)
3
4 %Main parameters:
5 h=1/(n-1);
6 x=0:h:1; %grid points.
7 maxit=1e3;
```

```
8
9  %Adjusting to the bc's.
10 A([1 n],:)=[]; B([1 n],:)=[];
11 enl=bctype(1);enr=bctype(2);
12 edl=1-enl; edr=1-enr;
13 index2=setdiff(1:n,index1);
14 index1r=setdiff(index1,[1 n]);
15 index2r=setdiff(index2,[1 n]);
16 I=eye(n); P(1:length(index1),1:n)=I(index1,:);
17 P(length(index1)+1:n,1:n)=I(index2,:);
18 nr=n-2; xr=x(2:end-1);
19 Ibc=eye(n);Ibc([1 n],:)=[];
20
21 %Subgrids
22 n1=length(index1); n2=length(index2);
23 n1r=length(index1r);
24 x1r=x(index1r); x2r=x(index2r);
25 A11=A(index1r-1,index1); B11=B(index1r-1,index1);
26 A12=A(index1r-1,index2); B12=B(index1r-1,index2);
27 A21=A(index2r-1,index1); B21=B(index2r-1,index1);
28 A22=A(index2r-1,index2); B22=B(index2r-1,index2);
29 for k=1:ns
30     f1r{k}=@(t) falg{k}(x1r,t);
31     f2r{k}=@(t) falg{k}(x2r,t);
32 end
33 class1=[ismember(1,index1) ismember(n,index1)];
34 class2=[ismember(1,index2) ismember(n,index2)];
35 Ibc1=eye(n1); Ibc2=eye(n2);
36 if class1(1)==1; Ibc1(1,:)=[]; end;
37 if class1(2)==1; Ibc1(end,:)=[]; end;
38 if class2(1)==1; Ibc2(1,:)=[]; end;
39 if class2(2)==1; Ibc2(end,:)=[]; end;
40 if ns>1; ddt1=Ibc1*diag(dt*ones(n1,1)); else ddt1=dt; end
41 if ns>1; ddt2=Ibc2*diag(dt*ones(n2,1)); else ddt2=dt; end
42
43 %Computing initial derivatives:
44 for k=1:ns
45 f{k}=@(t) falg{k}(xr,t);
46 end
47 pde=@(u,t,k) A*u+diag(Ibc*u)*B*u+f{k}(t);
48 if numel(uinit)==n
49     y=P*uinit;
50     for k=1:ns-1
51     uinit(2:n-1,k+1)=pde(uinit(:,k),0,k);
52     uinit(1,k+1)=enl*(uinit(2,k+1)-h*bc{1,k+1}(0))+...
53         edl*bc{1,k+1}(0);
```

```
    uinit(n,k+1)=enr*(uinit(n-1,k+1)+...
        h*bc{2,k+1}(0))+edr*bc{1,k+1}(0);
    y(k*n+1:(k+1)*n)=P*uinit(:,k+1);
    end
else
    for k=0:ns-1
    y(k*n+1:(k+1)*n,1)=P*uinit(:,k+1);
    end
end
U1=uinit(index1,:); U2=uinit(index2,:);
u1o=U1(:,1); u2o=U2(:,1);
if ns>1
v1o=U1(:,2); v2o=U2(:,2);
else
v1o=A11*u1o+A12*u2o+f1r{1}(t0)+...
    diag(B11*u1o+B12*u2o)*Ibc1*u1o;
v2o=A21*u1o+A22*u2o+f2r{1}(t0)+...
    diag(B21*u1o+B22*u2o)*Ibc2*u2o;
end
U1o=U1; U1o(:,1)=u1o; U1c=a1*U1+(1-a1)*U1o;
U2o=U2; U2o(:,1)=u2o; U2c=a2*U2+(1-a2)*U2o;

%Defining some functions and vectors:
G1=@(u1,u2) diag(B11*u1+B12*u2)*Ibc1+diag(Ibc1*u1)*B11;
G2=@(u1,u2) diag(B21*u1+B22*u2)*Ibc2+diag(Ibc2*u2)*B22;
H12=@(u1) diag(Ibc1*u1)*B12;
H21=@(u2) diag(Ibc2*u2)*B21;
nvecl=enl*(P*[1; -1; zeros(n-2,1)])';
nvecr=enr*(P*[zeros(n-2,1); 1; -1])';
dvecl=edl*(P*[-1; zeros(n-1,1)])';
dvecr=edr*(P*[zeros(n-1,1); -1])';
P2pre=eye(n);
P2pre=P2pre([end-1 1:n1r end end-1 n1r+1:nr end],:);
if class1(1)==1; P2pre(n1+1,:)=[]; else P2pre(1,:)=[]; end
if class1(2)==1; P2pre(end,:)=[]; else P2pre(end-n2,:)=[]; end
for k=0:ns-1
P2(k*n+1:k*n+n,1:n*ns)=[zeros(n,k*n) P2pre zeros(n,(ns-k-1)*n)];
end

%Constructing the constant part of the Jacobian:
Jpre=zeros(n*ns);
for k=0:ns-1
    Jpre(k*n+1:k*n+n1r,k*n+1:k*n+n1)=...
        (k<ns-1||b1==1)*(k>0||a1==1)*A11;
    Jpre(k*n+1:k*n+n1r,k*n+n1+1:k*n+n)=...
        (k<ns-1||b1==1)*(k>0||a2==1)*A12;
```

```
100     Jpre(k*n+n1r+1:k*n+nr,k*n+1:k*n+n1)=...
101         (k<ns-1||b2==1)*(k>0||a1==1)*A21;
102     Jpre(k*n+n1r+1:k*n+nr,k*n+n1+1:k*n+n)=...
103         (k<ns-1||b2==1)*(k>0||a2==1)*A22;
104     if k<ns-1
105     Jpre(k*n+1:k*n+n1r,(k+1)*n+1:(k+1)*n+n1)=-Ibc1;
106     Jpre(k*n+n1r+1:k*n+nr,(k+1)*n+n1+1:(k+2)*n)=-Ibc2;
107     else
108     Jpre(k*n+1:k*n+n1r,1:n1)=...
109         -(b1==0)*Ibc1+b1*Jpre(k*n+1:k*n+n1r,1:n1);
110     if ns>1; Jpre(k*n+1:k*n+n1r,1+n:n1+n)=...
111         (b1==0)*t1*ddt1+b1*Jpre(k*n+1:k*n+n1r,1+n:n1+n); end
112     Jpre(k*n+n1r+1:k*n+nr,n1+1:n)=...
113         (b2==0)*-Ibc2+b2*Jpre(k*n+n1r+1:k*n+nr,n1+1:n);
114     if ns>1; Jpre(k*n+n1r+1:k*n+nr,n1+1+n:2*n)=...
115         (b2==0)*t2*ddt2+b2*Jpre(k*n+n1r+1:k*n+nr,n1+1+n:2*n); end
116     end
117 end
118
119 stab=1;
120 for t=t0+dt:dt:t0+dur
121
122     if stab==0
123         break
124     end
125     bcpre=@(t) cellfun(@(c) c(t),bc);
126     bceval=bcpre(t);
127
128     F=constrF(A11,A12,A21,A22,B11,B12,B21,B22,U1,U2,U1c,U2c,...
129     u1o,u2o,v1o,v2o,Ibc1,Ibc2,ddt1,ddt2,enl,edl,enr,edr,...
130     nvecl,dvecl,nvecr,dvecr,bceval,f1r,f2r,t,h,n,ns,t1,t2,b1,b2);
131
132     teller=0;
133     res=norm(F); %upd 13-10: ipv res=thres;
134
135     %Performing Newton steps:
136     while res>thres && teller<maxit && sum(isnan(res))==0
137     teller=teller+1;
138
139     J=Jpre;
140     for k=1:ns
141         l1=k<ns;
142         for j=1:k
143             l2=j>1;l3=j==k;
144             J((k-1)*n+1:k*n,(j-1)*n+1:j*n)=...
145             J((k-1)*n+1:k*n,(j-1)*n+1:j*n)+...
```

```
            [(l1||b1==1)*(l2||a1==1)*nchoosek(k-1,j-1)*...
                G1(U1c(:,k-j+1),U2c(:,k-j+1)),...
            (l1||b1==1)*(l2||a2==1)*nchoosek(k-1,j-1)*...
                H12(U1c(:,k-j+1));
            (l1||b2==1)*(l2||a1==1)*nchoosek(k-1,j-1)*...
                H21(U2c(:,k-j+1)),...
            (l1||b2==1)*(l2||a2==1)*nchoosek(k-1,j-1)*...
                G2(U1c(:,k-j+1),U2c(:,k-j+1));
            l3*(nvecl+dvecl);
            l3*(nvecr+dvecr)];
        end
    end

    y=y-sparse(P2*J)\(P2*F);

    for k=0:ns-1
        U1(1:n1,k+1)=y(k*n+1:k*n+n1);
        U2(1:n2,k+1)=y(k*n+n1+1:k*n+n);
    end
    U1o=U1; U1o(:,1)=u1o; U1c=a1*U1+(1-a1)*U1o;%upd 13-10
    U2o=U2; U2o(:,1)=u2o; U2c=a2*U2+(1-a2)*U2o;

    F=constrF(A11,A12,A21,A22,B11,B12,B21,B22,U1,U2,U1c,U2c,...
    u1o,u2o,v1o,v2o,Ibc1,Ibc2,ddt1,ddt2,enl,edl,enr,edr,...
    nvecl,dvecl,nvecr,dvecr,bceval,f1r,f2r,t,h,n,ns,t1,t2,b1,b2);

    res=norm(F);

    end

    u1o=U1(:,1); u2o=U2(:,1);
    if ns>1
    v1o=U1(:,2); v2o=U2(:,2);
    else
    v1o=A11*u1o+A12*u2o+f1r{1}(t)+...
        diag(B11*u1o+B12*u2o)*Ibc1*u1o;
    v2o=A21*u1o+A22*u2o+f2r{1}(t)+...
        diag(B21*u1o+B22*u2o)*Ibc2*u2o;
    end

if teller>=maxit; stab=0; else stab=1; end
end

Y(index1,1:ns)=U1;
Y(index2,1:ns)=U2;

```

```
end

function F=constrF(A11,A12,A21,A22,B11,B12,B21,B22,U1,U2,...
U1c,U2c,u1o,u2o,v1o,v2o,Ibc1,Ibc2,ddt1,ddt2,enl,edl,enr,edr,...
nvecl,dvecl,nvecr,dvecr,bceval,f1r,f2r,t,h,n,ns,t1,t2,b1,b2)

if ns>1
v1c=t1*U1(:,2)+(1-t1)*v1o;
v2c=t2*U2(:,2)+(1-t2)*v2o;
else v1c=v1o;v2c=v2o;
end

for k=1:ns
    if k<ns
       F((k-1)*n+1:k*n,1)=...
          [A11*U1c(:,k)+A12*U2c(:,k)+f1r{k}(t)-Ibc1*U1(:,k+1);
           A21*U1c(:,k)+A22*U2c(:,k)+f2r{k}(t)-Ibc2*U2(:,k+1);
           enl*(nvecl*[U1(:,k);U2(:,k)]+bceval(1,k)*h)+...
               edl*(dvecl*[U1(:,k);U2(:,k)]+bceval(1,k));
           enr*(nvecr*[U1(:,k);U2(:,k)]+bceval(2,k)*h)+...
               edr*(dvecr*[U1(:,k);U2(:,k)]+bceval(2,k))];
        else
            F((k-1)*n+1:k*n,1)=...
            [(b1==1)*(A11*U1c(:,k)+A12*U2c(:,k)+f1r{k}(t))+...
                (b1==0)*(Ibc1*u1o+ddt1*v1c-Ibc1*U1(:,1));
             (b2==1)*(A21*U1c(:,k)+A22*U2c(:,k)+f2r{k}(t))+...
                 (b2==0)*(Ibc2*u2o+ddt2*v2c-Ibc2*U2(:,1));
            enl*(nvecl*[U1(:,k);U2(:,k)]+bceval(1,k)*h)+...
                edl*(dvecl*[U1(:,k);U2(:,k)]+bceval(1,k));
            enr*(nvecr*[U1(:,k);U2(:,k)]+bceval(2,k)*h)+...
                edr*(dvecr*[U1(:,k);U2(:,k)]+bceval(2,k))];
        end
        for j=1:k
        F((k-1)*n+1:k*n,1)=F((k-1)*n+1:k*n,1)+...
             [(k<ns||b1==1)*nchoosek(k-1,j-1)*...
                 (diag(B11*U1c(:,j)+B12*U2c(:,j))*...
                 Ibc1*U1c(:,k-j+1));
              (k<ns||b2==1)*nchoosek(k-1,j-1)*...
                 (diag(B21*U1c(:,j)+B22*U2c(:,j))*...
                 Ibc2*U2c(:,k-j+1));
             0;0];
        end
end
end
```

B.2 ENRK4

Since the main part of this program is roughly the same as EN (except for the redundant θ_2 and β_2 input parameters, which have both been set to zero), we give only the function F:

```
function F=constrF(A11,A12,A21,A22,B11,B12,B21,B22,U1,U2,...
U1c,U2c,u1o,u2o,v1o,v2o,Ibc1,Ibc2,ddt1,ddt2,enl,edl,enr,edr,...
nvecl,dvecl,nvecr,dvecr,bceval,f1r,f2r,t,h,n,ns,t1,t2,b1,b2)

if ns>1
v1c=t1*U1(:,2)+(1-t1)*v1o;
v2c=t2*U2(:,2)+(1-t2)*v2o;
else v1c=v1o;v2c=v2o;
end

%rk4 part:
u(index1)=u1o; u(index2)=u2o; u=u';
k1(2:n-1,1)=pde(u,t-dt,1);
k1(1)=enl*(k1(2)-h*bc{1,2}(t-dt))+edl*bc{1,2}(t-dt);
k1(n)=enr*(k1(n-1)+h*bc{2,2}(t-dt))+edr*bc{2,2}(t-dt);
k2(2:n-1,1)=pde(u+.5*dt*k1,t-.5*dt,1);
k2(1)=enl*(k2(2)-h*bc{1,2}(t-.5*dt))+edl*bc{1,2}(t-.5*dt);
k2(n)=enr*(k2(n-1)+h*bc{2,2}(t-.5*dt))+edr*bc{2,2}(t-.5*dt);
k3(2:n-1,1)=pde(u+.5*dt*k2,t-.5*dt,1);
k3(1)=enl*(k3(2)-h*bc{1,2}(t-.5*dt))+edl*bc{1,2}(t-.5*dt);
k3(n)=enr*(k3(n-1)+h*bc{2,2}(t-.5*dt))+edr*bc{2,2}(t-.5*dt);
k4(2:n-1,1)=pde(u+dt*k3,t,1);
k4(1)=enl*(k4(2)-h*bc{1,2}(t))+edl*bc{1,2}(t);
k4(n)=enr*(k4(n-1)+h*bc{2,2}(t))+edr*bc{2,2}(t);
dunew=1/6*dt*(k1+2*k2+2*k3+k4);
dunew2=dunew(index2);

for k=1:ns
    if k<ns
       F((k-1)*n+1:k*n,1)=...
          [A11*U1c(:,k)+A12*U2c(:,k)+f1r{k}(t)-Ibc1*U1(:,k+1);
           A21*U1c(:,k)+A22*U2c(:,k)+f2r{k}(t)-Ibc2*U2(:,k+1);
           enl*(nvecl*[U1(:,k);U2(:,k)]+bceval(1,k)*h)+...
               edl*(dvecl*[U1(:,k);U2(:,k)]+bceval(1,k));
           enr*(nvecr*[U1(:,k);U2(:,k)]+bceval(2,k)*h)+...
               edr*(dvecr*[U1(:,k);U2(:,k)]+bceval(2,k))];
        else
            F((k-1)*n+1:k*n,1)=...
            [(b1==1)*(A11*U1c(:,k)+A12*U2c(:,k)+f1r{k}(t))+...
                (b1==0)*(Ibc1*u1o+ddt1*v1c-Ibc1*U1(:,1));
```

```
            Ibc2*(u2o+dunew2-U2(:,1));
            enl*(nvecl*[U1(:,k);U2(:,k)]+bceval(1,k)*h)+...
                edl*(dvecl*[U1(:,k);U2(:,k)]+bceval(1,k));
            enr*(nvecr*[U1(:,k);U2(:,k)]+bceval(2,k)*h)+...
                edr*(dvecr*[U1(:,k);U2(:,k)]+bceval(2,k))];
        end
        for j=1:k
        F((k-1)*n+1:k*n,1)=F((k-1)*n+1:k*n,1)+...
             [(k<ns||b1==1)*nchoosek(k-1,j-1)*...
                 (diag(B11*U1c(:,j)+B12*U2c(:,j))*...
                 Ibc1*U1c(:,k-j+1));
              (k<ns||b2==1)*nchoosek(k-1,j-1)*...
                 (diag(B21*U1c(:,j)+B22*U2c(:,j))*...
                 Ibc2*U2c(:,k-j+1));
              0;0];
        end
end
end
```

B.3 ENRK2

Since the main part of this program is roughly the same as EN (except for the redundant θ_2 and β_2 input parameters, which have both been set to zero), we give only the function F:

```
function F=constrF(A11,A12,A21,A22,B11,B12,B21,B22,U1,U2,...
U1c,U2c,u1o,u2o,v1o,v2o,Ibc1,Ibc2,ddt1,ddt2,enl,edl,enr,edr,...
nvecl,dvecl,nvecr,dvecr,bceval,f1r,f2r,t,h,n,ns,t1,t2,b1,b2)

if ns>1
v1c=t1*U1(:,2)+(1-t1)*v1o;
v2c=t2*U2(:,2)+(1-t2)*v2o;
else v1c=v1o;v2c=v2o;
end

%rk2 part:
u(index1)=u1o; u(index2)=u2o; u=u';
k1(2:n-1,1)=pde(u,t-dt,1);
k1(1)=enl*(k1(2)-h*bc{1,2}(t-dt))+edl*bc{1,2}(t-dt);
k1(n)=enr*(k1(n-1)+h*bc{2,2}(t-dt))+edr*bc{2,2}(t-dt);
k2(2:n-1,1)=pde(u+dt*k1,t,1);
k2(1)=enl*(k2(2)-h*bc{1,2}(t))+edl*bc{1,2}(t);
k2(n)=enr*(k2(n-1)+h*bc{2,2}(t))+edr*bc{2,2}(t);
dunew=1/2*dt*(k1+k2);
dunew2=dunew(index2);
```

```
21
22 for k=1:ns
23     if k<ns
24        F((k-1)*n+1:k*n,1)=...
25           [A11*U1c(:,k)+A12*U2c(:,k)+f1r{k}(t)-Ibc1*U1(:,k+1);
26            A21*U1c(:,k)+A22*U2c(:,k)+f2r{k}(t)-Ibc2*U2(:,k+1);
27            enl*(nvecl*[U1(:,k);U2(:,k)]+bceval(1,k)*h)+...
28                edl*(dvecl*[U1(:,k);U2(:,k)]+bceval(1,k));
29            enr*(nvecr*[U1(:,k);U2(:,k)]+bceval(2,k)*h)+...
30                edr*(dvecr*[U1(:,k);U2(:,k)]+bceval(2,k))];
31         else
32             F((k-1)*n+1:k*n,1)=...
33             [(b1==1)*(A11*U1c(:,k)+A12*U2c(:,k)+f1r{k}(t))+...
34                 (b1==0)*(Ibc1*u1o+ddt1*v1c-Ibc1*U1(:,1));
35             Ibc2*(u2o+dunew2-U2(:,1));
36             enl*(nvecl*[U1(:,k);U2(:,k)]+bceval(1,k)*h)+...
37                 edl*(dvecl*[U1(:,k);U2(:,k)]+bceval(1,k));
38             enr*(nvecr*[U1(:,k);U2(:,k)]+bceval(2,k)*h)+...
39                 edr*(dvecr*[U1(:,k);U2(:,k)]+bceval(2,k))];
40         end
41         for j=1:k
42         F((k-1)*n+1:k*n,1)=F((k-1)*n+1:k*n,1)+...
43              [(k<ns||b1==1)*nchoosek(k-1,j-1)*...
44                  (diag(B11*U1c(:,j)+B12*U2c(:,j))*...
45                  Ibc1*U1c(:,k-j+1));
46              (k<ns||b2==1)*nchoosek(k-1,j-1)*...
47                  (diag(B21*U1c(:,j)+B22*U2c(:,j))*...
48                  Ibc2*U2c(:,k-j+1));
49              0;0];
50         end
51 end
52 end
```

8 References

[1] J. Heijnen. *Modelordereductie van tijdsonafhankelijke PDV's*, Bacheloronderzoek, 2011.

[2] A. Quarteroni, R. Sacco, F. Saleri. *Numerical mathematics*, Springer, 2007.

[3] F.W. Wubs, E.D. De Goede. An explicit-implicit method for a class of time-dependent partial differential equations, *Applied Numerical Mathematics 9*, 1992.

[4] Fred W. Wubs and Jonas Thies. A Robust Two-Level Incomplete Factorization for (Navier)Stokes Saddle Point Matrices, *SIAM J. Matrix Anal. and Appl.*, 2011