M1 CSMI Internship
Adaptive Implicit Schemes for Hyperbolic Equations

Antoine REGARDIN
Supervisors: Andrea THOMANN, Emmanuel FRANCK
University of Strasbourg

Summer 2024

Contents

1 Introduction

In physics, especially gas or waveforms dynamics, many modelling problems require to solve hyperbolic equations for discontinuous functions. Well-known examples can be found in the study of Riemann problems, which are sets of hyperbolic conservation laws with piecewise-constant data presenting one discontinuous jump.

When studying hyperbolic partial differential equations with piecewise-smooth initial data presenting discontinuities, the usual methods for numerical solving loose a lot of precision, and thus a lot of interest. In the case of the family Euler methods, explicit schemes often present very bad, imprecise results, and solutions obtained with implicit schemes are too prone to dissipation. This comes mainly from the fact that "A shock in space is also a shock in time" Arbogast and Huang 2021, p. 1: the discontinuities are moving in space in function of the time, and thus we need to adapt the resolution in time of our equations. Other methods, for example Riemann solvers in the case of Riemann problems, have satisfying results but a high order of resolution making them heavy in terms of computations, whereas we aim to obtain lighter methods in order to solve problems for longer final times.

In this paper, we will study the Self-Adaptive Theta-scheme, proposed by Pr. Arbogast et al. in their 2021 paper [1], focusing on hyperbolic conservation laws only. The principle of this scheme is to take an usual Theta-scheme and implement a way to compute an adapted value of θ for each space and time cell, in order to handle the solving around the perturbations. Indeed, in a Theta-scheme, the value θ makes the balance between an explicit part and an implicit part of the resolution. The main idea is then to make sure we use a higher order of computation only around the singularities We will propose an implementation of this method in Python, and try to reproduce some results of Pr. Arbogast's paper by applying it on the linear advection equation, notably in the case of a Riemann problem. We will then expand this method to problems that have not been studied yet: Burgers equation and the RIPA model, with several initial data (multiple-shock piecewise functions and shock-creating sine function for Burgers, dam-breaking problems for the RIPA model). We will particularly study the numerical resolution with bad CFL coefficients, as this method is a good candidate to make physical simulations over long times without requiring very heavy computations.

2 Material and Methods

2.1 Hyperbolic Conservation Laws

Here the conservative form of a hyperbolic equation:

$$\partial_t u + \partial_x f(u) = 0 \tag{1}$$

With f a function that will first be linear, in the case of the linear advection equation, then nonlinear for the other problems. We will study these problems in rectangular closed domains $[a, b]^n$, with $a, b \in \mathbb{R}$ and $n = 1$ for the linear advection and Burgers equation, and $n = 3$ for the RIPA model. Two types of spatial boundary conditions will be considered: Dirichlet (constant flux on the first and last spatial cells of the numerical domain) and periodical. We will not be interested in the behaviour of the solutions next to the boundaries, we are only interested by their behaviour around the shocks.

2.2 Building the scheme

In this part, we will work in one dimension to give an outline of the definition of the discretization and the scheme. In this purpose, the following reasoning and formula of this part will be cited from Mr. Arbogast's work. When we will study the RIPA model, which depends on three variables and thus three dimensions, the formulas will remain the same. We will only have to apply them to vectors of size 3 instead of 1.

Let u be a general function defining a field in $[a, b]$. We split the field u with values $x_i, \quad i \in \{0, 1, \ldots, Nx\}$, and we will write $u(t, x_i) = u_i^t$. We note t_f the final time of our computations.
We will work with constant space and time steps for each cell, of respective values Δx and Δt.

As we said, a jump in space is also a jump in time, thus we need a way to adapt our scheme to the localization of the jump both in space and in time. We use a "Discontinuity-Aware Quadrature": we approximate the location of the discontinuity by: tau formula, cases for tau...

In part 3 of the paper, a Discontinuity-Aware Quadrature is built to take care of the spatio-temporal displacement of our discontinuity. We define a piecewise-smooth function v evolving in the time dimension $[0, \Delta t] \to \mathbb{R}$ that presents a jump at the time τ:

$$v(t) = \begin{cases} v^0 + v_L(t), & 0 \le t < \tau, \\ v^1 + v_R(t), & \tau < t \le \Delta t. \end{cases} \tag{2}$$

With v_L and v_R continuous functions, and v^0 and v^1 constants. In order to solve our problem, we want to approximate the integration of a smooth function g depending on $(t, v(t))$. We note $\tilde{v}$ the mean value of v over $[0, \Delta t]$: $\frac{1}{\Delta t}\int_0^{\Delta t} v(t)dt$ (3). We want to determine an approximation $\tau^* \approx \tau$.

$$\int_0^{\Delta t} g(t, v(t))dt \approx \int_0^{\tau^*} g(t, v^0)dt + \int_{\tau^*}^{\Delta t} g(t, v^1)dt \tag{3.1}$$

With $g(t, v(t)) = v(t)$, we have:

$$\Delta t\tilde{v} = \int_0^{\Delta t} v(t)dt \approx \tau^* v^0 + (\Delta t - \tau^*)v^1 \tag{3.2}$$

Which gives us, assuming $v^0 \neq v^1$:

$$\tau^* = \frac{v^1 - \tilde{v}}{v^1 - v^0}\Delta t \tag{3.3}$$

For the case $v^0 = v^1$, we assume $\tau^* = \frac{\Delta t}{2}$.

The idea is to use the estimated quantity τ^* can be used to balance the terms between implicit and explicit. We can use the following formula for θ:

$$\tau = 1 - \frac{\tau^*}{\Delta t} = \frac{\tilde{v} - v^0}{v^1 - v^0} \tag{3.4}$$

And using the discontinuity-aware Quadrature [1]:

$$\int_0^{\Delta t} f(v(t))dt \approx (f^0 + \theta(f^1 - f^0))\Delta t \tag{3.5}$$

Note that we only need to set all the thetas to the same constant value if we want to use a standard (constant) Theta scheme.

2.2.1 The Theta-Upwind scheme for Linear Advection

In the case of the linear advection equation, we can write our problem (1) with $f(u) = au$, with the parameter a characterizing the speed and direction of the modelized flux. We use the notation $f(\bar{u}_i) = \bar{f}_i$.

$$\begin{cases} \bar{u}_i^{n+1} = \bar{u}_i - \frac{1}{\Delta x} \int_{t^n}^{t_{n+1}} (\bar{f}_i - \bar{f}_{i-1}) dt, \\ \tilde{\bar{u}}_i^{n+1} = \bar{u}_i - \frac{1}{\Delta x} \int_{t^n}^{t_{n+1}} (\bar{f}_i - \bar{f}_{i-1}) \frac{t^{n+1} - t^n}{\Delta t^{n+1}} dt \end{cases} \tag{4}$$

From now on, we will note $v_i^{n+1} = \bar{u}_i^{n+1} - \bar{u}_i^n$ and $w_i^{n+1} = \tilde{\bar{u}}_i^{n+1} - \bar{u}_i^n$.

First, we will write our scheme using an upstream weighting, in order to have a modified version of a backwards Euler scheme.

$$\begin{cases} \bar{u}_i^{n+1} = \bar{u}_i - \frac{\Delta t^{n+1}}{\Delta x} [\bar{f}_i^n + \theta_i^{n+1} (\bar{f}_i^{n+1} - \bar{f}_i^n) - \bar{f}_{i-1}^n - \theta_{i-1}^{n+1} (\bar{f}_{i-1}^{n+1} - \bar{f}_{i-1}^n)], \\ \tilde{\bar{u}}_i^{n+1} = \bar{u}_i - \frac{\Delta t^{n+1}}{\Delta x} [\bar{f}_i^n + (\theta_i^{n+1})^2 (\bar{f}_i^{n+1} - \bar{f}_i^n) - \bar{f}_{i-1}^n - (\theta_{i-1}^{n+1})^2 (\bar{f}_{i-1}^{n+1} - \bar{f}_{i-1}^n)] \end{cases} \tag{5}$$

We introduce new parameters for our model: θ_{min}, θ^* and ϵ. And here is our formula to compute each θ_i^n , for $i \in [0, N]$ and $n \in [0, t_f]$.:

$$\theta_i^{n+1} = \begin{cases} \min(\max(\theta_{min}, \frac{w_i^{n+1}}{v_i^{n+1}}), 1) & \text{if } |w_i^{n+1}| > \epsilon. \\ \theta^* & \text{else} \end{cases} \tag{6}$$

When not precised, the value of *epsilon* will be $1e-6$, and we will work with $\theta_{min} \geq 0.5$, as suggested in Pr. Arbogast's paper.

2.2.2 The Lax-Friedrichs SATh

The upstream-weighted scheme can be used in the case of the linear advection equation, as it has a linear flux spreading in only one direction. However, the other equations we want to study present a nonlinear flux, representing waves that can spread in varying directions and speeds, making this first scheme unfit for the computation of numerical solutions. We thus want a new nonlinear scheme allowing to modelize this behaviour, and the one chosen by Pr. Arbogast is Lax-Friedrichs. We will now need to know the maximum wave speed of our field u, that we will write α. In the case of the linear advection we have $\alpha = a, (\text{with} au = f(u))$, and in the general case, we have:

$$\alpha = \max_u |f'(u)|.$$

We define the following numerical flux:

$$\hat{f}(u^-, u^+) = \frac{1}{2}[f(u^-) + f(u^+) - \alpha(u^+ - u^-)]. \tag{7.1}$$

With u^- and u^+ left and right limits of the solution at each cell. According to the paper, we use simple one point upstream weighting to determine these values: $u_{i+1/2}^- = \bar{u}_i$ and $u_{i+1/2}^+ = \bar{u}_{i+1}$. Now, we can write the Self-Adaptive Theta Lax-Friedrichs scheme by starting from () again:

$$\begin{cases} \bar{u}_i^{n+1} = \bar{u}_i^n - \frac{\Delta t^{n+1}}{2\Delta x_i}[\bar{f}_{i+1}^n + \theta_{i+1}^{n+1}(\bar{f}_{i+1}^{n+1} - \bar{f}_{i+1}^n) - \bar{f}_{i-1}^n - \theta_{i-1}^{n+1}(\bar{f}_{i-1}^{n+1} - \bar{f}_{i-1}^n) \\ \qquad - \alpha[\bar{u}_{i+1}^n + \theta_{i+1}^{n+1}(\bar{u}_{i+1}^{n+1} - \bar{u}_{i+1}^n) - 2\bar{u}_i^n - 2\theta_i^{n+1}(\bar{u}_i^{n+1} - \bar{u}_i^n) + \bar{u}_{i-1}^n + \theta_{i-1}^{n+1}(\bar{u}_{i-1}^{n+1} - \bar{u}_{i-1}^n)]], \\ \\ \tilde{\bar{u}}_i^{n+1} = \bar{u}_i^n - \frac{\Delta t^{n+1}}{4\Delta x_i}[\bar{f}_{i+1}^n + (\theta_{i+1}^{n+1})^2(\bar{f}_{i+1}^{n+1} - \bar{f}_{i+1}^n) - \bar{f}_{i-1}^n - (\theta_{i-1}^{n+1})^2(\bar{f}_{i-1}^{n+1} - \bar{f}_{i-1}^n) \\ \qquad - \alpha[\bar{u}_{i+1}^n + (\theta_{i+1}^{n+1})^2(\bar{u}_{i+1}^{n+1} - \bar{u}_{i+1}^n) - 2\bar{u}_i^n - 2(\theta_i^{n+1})^2(\bar{u}_i^{n+1} - \bar{u}_i^n) + \bar{u}_{i-1}^n + (\theta_{i-1}^{n+1})^2(\bar{u}_{i-1}^{n+1} - \bar{u}_{i-1}^n)]] \end{cases} \tag{7.2}$$

2.3 Implementation of the Scheme

We now have our formulas for the scheme, so the next step is to use them to obtain numerical solutions. At each time step, we do not only need to solve a system to compute the implicit part of our scheme, we also need to compute the $\theta_i$for each$i$, which depend on the v and w values of the next time step.

In order to do this, we propose to use the Newton method. The idea is that we want to write our schemes as a function F of a variable X, of which we want to find the 0. Instead of iterating the schemes over the simulation time, we will iterate them over an index k. We shift the index of the different θ that should (k) to $(k-1)$ in order to have all the values to iterate. We will use the following functions for F:
For the Upwind-SATh scheme, we start from (5), write $\lambda = \frac{\Delta t}{\Delta x}$ and move everything on the left of the equality:

$$\begin{cases} v_i^k + \frac{\Delta t}{\Delta x}[\bar{f}_i^{k-1} + \theta_i^{k-1}(\bar{f}_i^k - \bar{f}_i^{k-1}) - \bar{f}_{i-1}^{k-1} - \theta_{i-1}^{k-1}(\bar{f}_{i-1}^k - \bar{f}_{i-1}^{k-1})] = 0, \\ w_i^k + \frac{\Delta t}{\Delta x}[\bar{f}_i^{k-1} + (\theta_i^{k-1})^2(\bar{f}_i^k - \bar{f}_i^{k-1}) - \bar{f}_{i-1}^{k-1} - (\theta_{i-1}^{k-1})^2(\bar{f}_{i-1}^k - \bar{f}_{i-1}^{k-1})] = 0 \end{cases} \tag{8.1}$$

And for the Lax-Friedrichs-SATh scheme, we do the same thing from (987):

$$\begin{cases} v_i^k + \frac{\Delta t^k}{2\Delta x_i}[\bar{f}_{i+1}^{k-1} + \theta_{i+1}^{k-1}(\bar{f}_{i+1}^k - \bar{f}_{i+1}^{k-1}) - \bar{f}_{i-1}^{k-1} - \theta_{i-1}^{k-1}(\bar{f}_{i-1}^k - \bar{f}_{i-1}^{k-1}) \\ -\alpha[\bar{u}_{i+1}^{k-1} + \theta_{i+1}^{k-1}(\bar{u}_{i+1}^k - \bar{u}_{i+1}^{k-1}) - 2\bar{u}_i^{k-1} - 2\theta_i^{k-1}(\bar{u}_i^k - \bar{u}_i^{k-1}) + \bar{u}_{i-1}^{k-1} + \theta_{i-1}^{k-1}(\bar{u}_{i-1}^k - \bar{u}_{i-1}^{k-1})]] = 0, \\ w_i^k + \frac{\Delta t^k}{4\Delta x_i}[\bar{f}_{i+1}^{k-1} + (\theta_{i+1}^{k-1})^2(\bar{f}_{i+1}^k - \bar{f}_{i+1}^{k-1}) - \bar{f}_{i-1}^{k-1} - (\theta_{i-1}^{k-1})^2(\bar{f}_{i-1}^k - \bar{f}_{i-1}^{k-1}) \\ -\alpha[\bar{u}_{i+1}^{k-1} + (\theta_{i+1}^{k-1})^2(\bar{u}_{i+1}^k - \bar{u}_{i+1}^{k-1}) - 2\bar{u}_i^{k-1} - 2(\theta_i^{k-1})^2(\bar{u}_i^k - \bar{u}_i^{k-1}) + \bar{u}_{i-1}^{k-1} + (\theta_{i-1}^{k-1})^2(\bar{u}_{i-1}^k - \bar{u}_{i-1}^{k-1})]] = 0 \end{cases} \tag{8.2}$$

We thus have to solve $F(X^k) = 0 \quad$ with $\quad X^k = (v_0^k, w_0^k, v_1^k, w_2^k, \dots, v_{N+1}^k, w_{N+1}^k)$ iteratively over k, while updating the values of θ_i each time according to (6). We must not forget to take care of the boundary conditions: for Dirichlet boundary conditions, we will have $(v_0^k, w_0^k) = (0,0) = (v_{N+1}^k, w_{N+1}^k)$ and for periodical $(v_0^k, w_0^k) = (v_{N+2}^k, w_{N+2}^k)$, with a ghost cell of index $N+2$ right after the last real cell.
The iterations over k stop if the residue is inferior or equal to a tolerance threshold, or if the Newton method reached the maximum number of iterations.

When we call our Newton method at the time step n, we get the values v_i^{n+1} and w_i^{n+1} for each i. Then, to obtain the numerical solution at time step $n+1$, meaning the values $\bar{u}_i^{n+1}$ for each i, we only need to add the value v_i^{n+1} we just got to $\bar{u}_i^{n+1}$, as $v_i^{n+1} = \bar{u}_i^{n+1} - \bar{u}_i^n$.

We must note that if our parameters θ_{min} and θ^* are not equal, the function (6) can introduce a small discontinuous behaviour in F. This could cause trouble for the Newton method, in function of the chosen algorithm to implement it.

2.4 Settings for simulations

In the case of the linear advection, we will study a bell curve and a Riemann problem, defined by the initial condition:

$$u_0(x) = \begin{cases} 1 & \text{if} \quad x \leq x_0 \\ 0 & \text{if} \quad x > x_0. \end{cases}$$

With x_0 the location of the jump, that will often be 0.2. We will work with a speed $a = 1$.
For the burgers equation, defined by (1) with $f(u) = u^2$, we will test several initial settings: A shock propagation problem:

$$u_0(x) = \begin{cases} 0 & \text{if} \quad x \leq x_0 \\ 1 & \text{if} \quad x > x_0. \end{cases}$$

A shock and rarefaction problem:

$$u_0(x) = \begin{cases} 0 & \text{if} \quad x \leq x_1 \\ 1 & \text{if} \quad x > x_1 \quad \text{and} \quad x \leq x_2 \\ 0 & \text{if} \quad x > x_2 \end{cases}$$

With x_1 and x_2 locations of jumps.

And an shock creation problem. For this, we start with the sinusoidal function:

$$u_0(x) = 0.5 + \sin(\pi x)$$

and the shock is supposed to appear at the time $t = 1/\pi = 0.318$.

For the RIPA model, used to represent shallow water waves by taking care of the heat gradient, we rely on the paper al. 2016, p. 2. We study an equation over the 3 following dimensions: the water weight $h(x,t)$, the velocity $u(x,t)$ and the potential temperature field $\Theta(x,t)$. We have the following flux, for $W = (h, hu, h\Theta)$ and $g = 1$:

$$f(W) = \begin{pmatrix} hu \\ hu^2 + g\Theta h^2/2 \\ h\Theta u \end{pmatrix}$$

and we only had the time to start testing one configuration, a problem of dam break over a flat bottom:

$$(h, u, \Theta)(x, 0) = \begin{cases} (5, 0, 3) & \text{if} \quad x < 0 \\ (1, 0, 5) & \text{if} \quad x \geq 0 \end{cases}$$

2.5 Code

All of the code is written in Python, with the help of libraries `numpy`, `scipy` and `matplotlib`. To solve the iterations of the scheme and the Newton method, we use the function `scipy.optimize.newton_krylov`, corresponding to the Newton-Krylov algorithm. This algorithm is useful, because it spares us the computing of the Jacobian matrix of F and its inverse. Indeed, we only need to give it the function F as a linear operator, and an initial guess of the solving vector. For this we give, at the computation time n, the vector $(v_0^n, 0, v_1^n, 0, \dots v_{N+1}^n, 0)$.
If not precised, the tolerance used for the algorithm will be $6e-6$, as advised by the scipy documentation "scipy.optimize.newton_ krylov function" n.d., p. 3, and the maximum number of iterations will be 10, as advised for Newton methods because of significant digit precision.

3 Results

We will present some interesting results of computations implying the proposed schemes and problems, in order to give an outline of the actual implementation of the method. When needed, we will compute errors by taking the L2-norm and inifinity-norm of the difference between the numerical solution and the exact solution (if we have it) or a pseudo-exact one. These "pseudo-exact" solutions are computed by using a fully explicit Euler scheme with a very low CFL condition and a high mesh density, in order to obtain a very close (but very slow) result to the theoretical solution of the equation. We will extract the closest values in abscissa to the numerical solution we want to study in order to get their difference.

3.1 Linear Advection

First, we will compute solutions for the linear advection equation with various parameters, and explore some possibilities in a broader way than the next problems, in order to experiment with the SATh scheme.

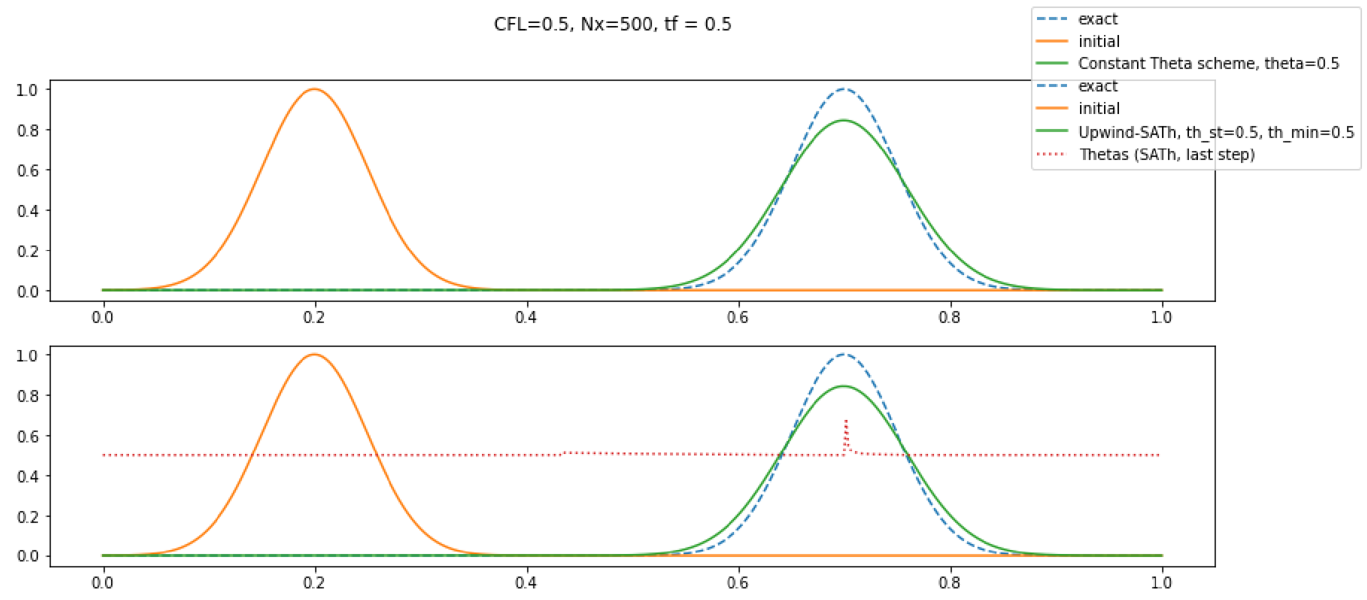

Figure 1: Smooth bell initial condition, with CFL=0.5 and theta paremeters =0.5, for both constant theta and SATh schemes. We can see that the numerical solutions are almost identical (L2 errors $\approx$ 0.93 for both), but the computation times are very different (0.2 seconds for constant theta, and 62 seconds for SATh). This is as expected, the SATh gives a similar result than the standard scheme for smooth problems, but is very slow and thus not interesting.

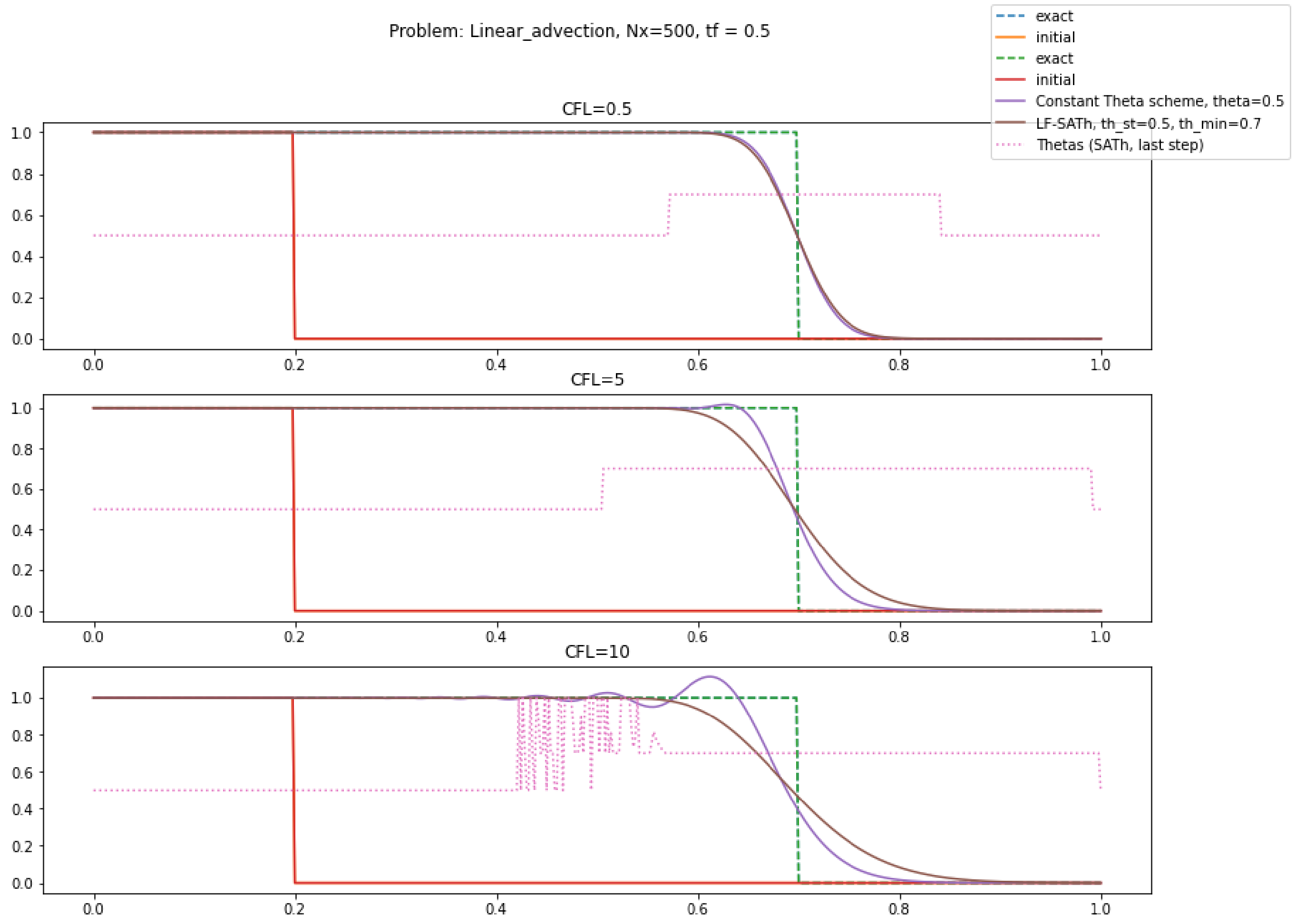

Figure 2: Let's move to discontinuous settings. Here are the plots for 3 different CFL conditions of the constant theta scheme with $\theta = 0.5$, and the SATh with $\theta_{min} = 0.6, \theta^* = 0.5$. We can see that the SATh is efficient to avoid the oscillations associated with the standard methods in high CFL conditions. We will see right after why we took this value for θ_{min}.

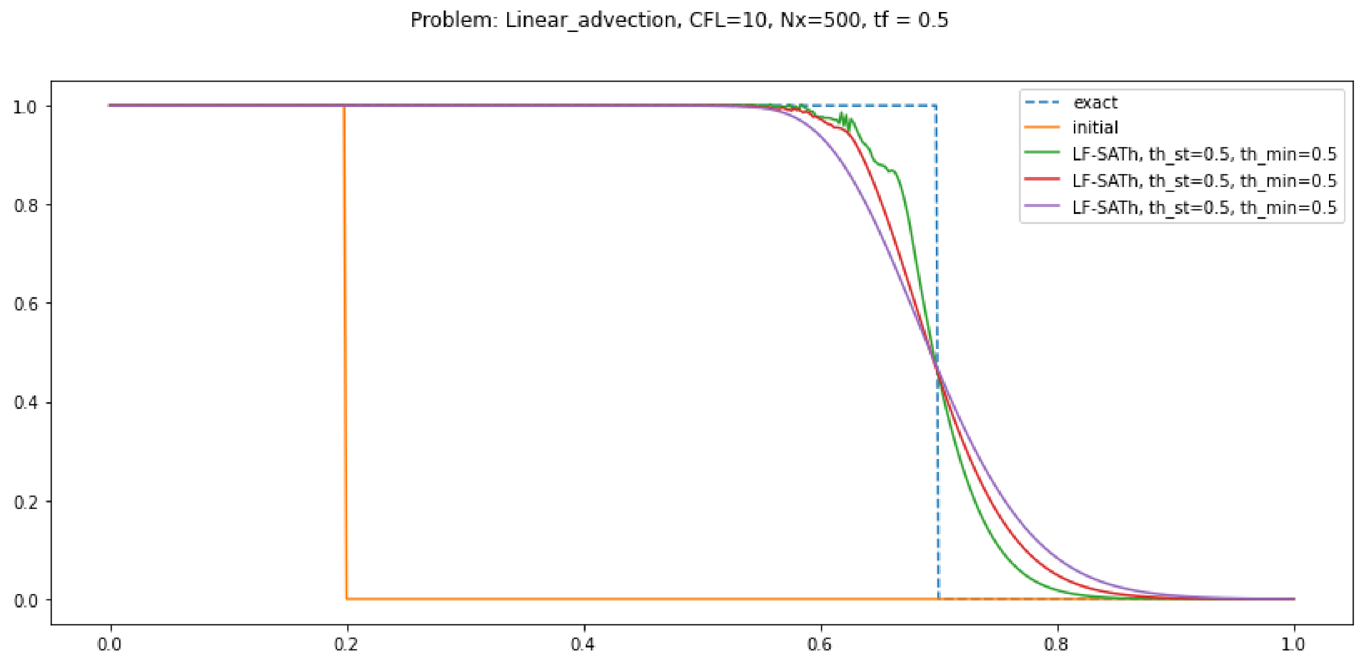

Figure 3: We notice that when $\theta_{min} = 0.5$ (with $\theta^* = 0.5$), a lot of small oscillations appear. When using higher values, they smear out, but the solutions are more dissipated. Changing the value of ϵ in a range from $1e-3$ to $1e-100$ in (6) does not suppress these oscillations. As this problem is not yet solved, we will plot with $\theta_{min} > 0.5$, but this should be adressed in the future.

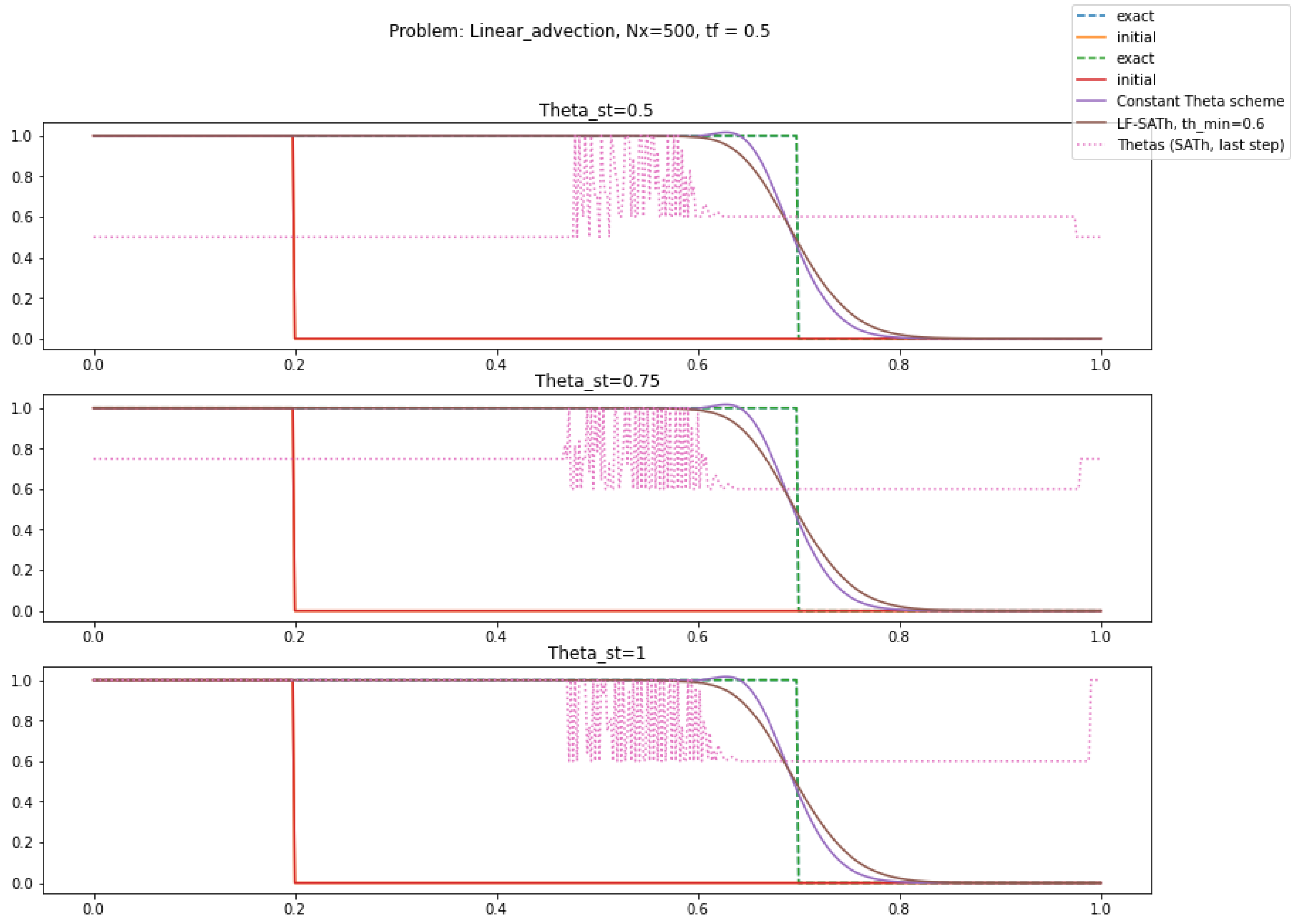

Figure 4: Plots for varying values of θ^*, compared to the Crank-Nicholson solution (constant theta = 0.5). We can see that the numerical solutions for the SATh are very similaar for different θ^*.

When varying the tolerance for residue convergence in the Newton-Krylov method between $1e-3$ and $1e-12$, we observe that the error of the numerical solution is not variating significatively. The computation times are also similar.

3.2 Burgers Equation

Now, we will present the most interesting results for the Burgers equation.

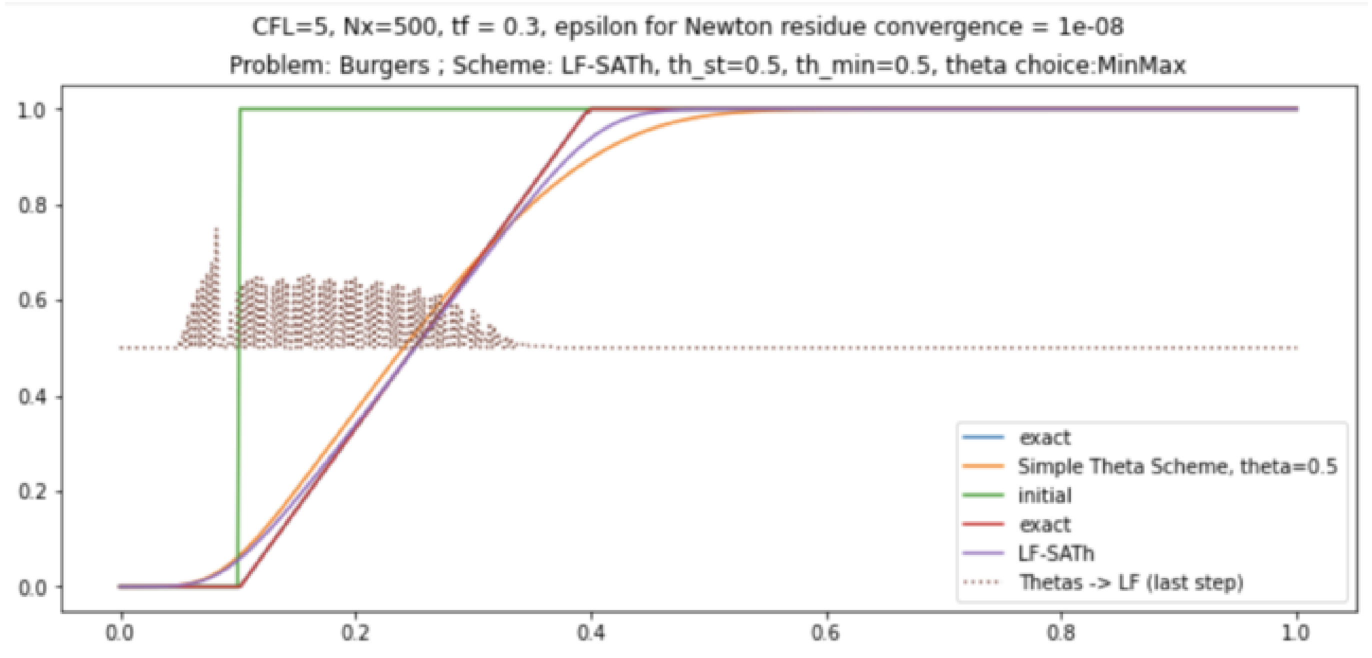

Figure 5: Simulation for Burgers with one jump initial condition.

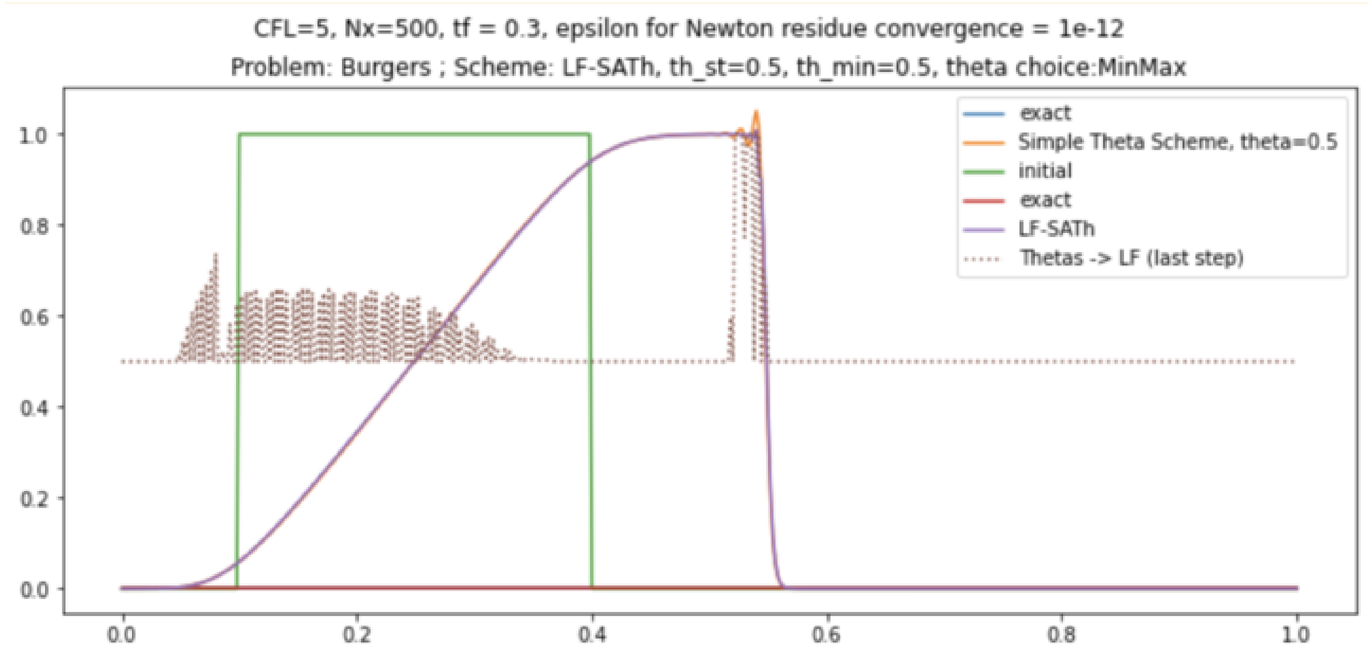

Figure 6: Simulation for Burgers with the shocks and rarefaction condition.

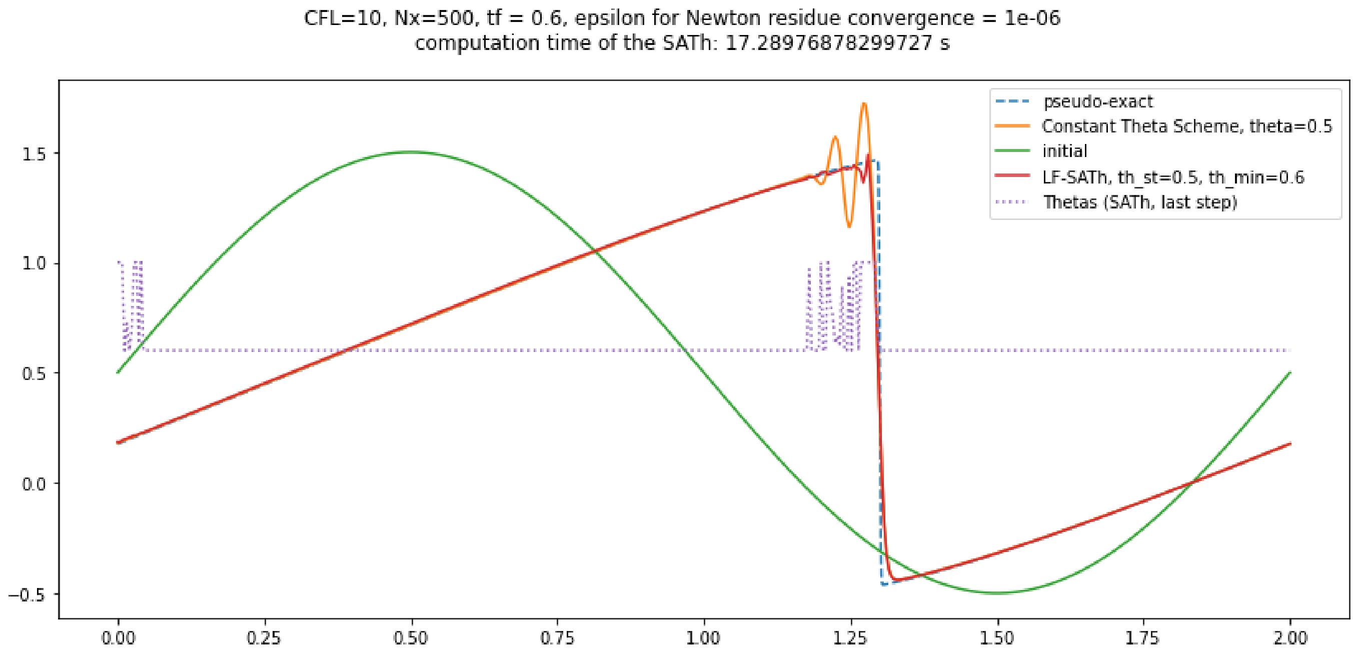

Figure 7: Simulation for Burgers with the shock creatin condition (with periodical boundary conditions)

We can see that the SATh presents promising results for the Burgers equation with high CFL conditions, compared to standard Theta method (Crank-Nicholson).

3.3 RIPA model

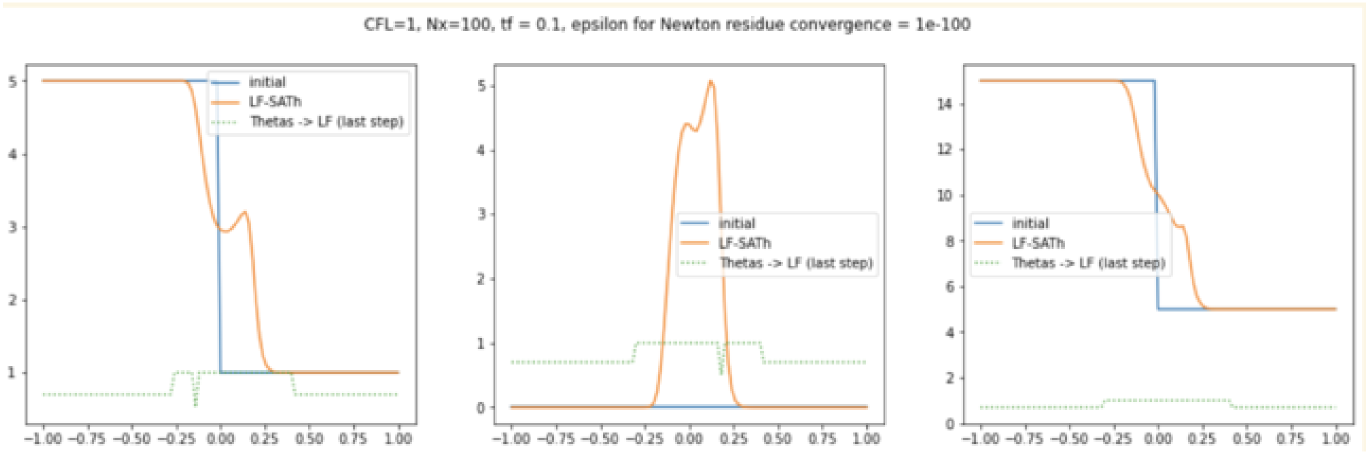

Figure 8: The first result we had for the RIPA model. This would need a longer computation time, but it is quite heavy in terms of computations. We did not have the time to pursue these simulations, but it is looking promising, when compared to results in [2].

4 Conclusion

In these experiments, we saw that our version of the Self-Adaptive Theta scheme needs improving, especially in the smoothness ans precision of numerical solutions. Nonetheless, this scheme is very interesting in the case of long-time simulations, e.g. with bad CFL conditions: we saw that the numerical solutions had quite a good behaviour with CFL conditions superior or equal to 5, and with accessible computation times that can still be improved by refining the method. Some behaviours need to be studied further, like the appearance of fast oscillations, or the qualification of boundary values for the parameters θ. Moreover, we noticed a strange behaviour of the solutions in function of the mesh precision that we chose to not show here because it could spark from an unidentified error: int the actual state, there seems to be no convergence in function of the mesh refinement.

5 Bibliography

Arbogast and Huang 2021 al. 2016 "scipy.optimize.newton_ krylov function" n.d.

References

1. al., Desveaux et (2016). "Well-balanced Schemes to capture non-explicit steady states: RIPA Model". In: *Mathematics of Computation* 85.300.

2. Arbogast and Huang (2021). "Self-Adaptive Theta Schemes for solving Hyperbolic Conservation Laws". In: *IMA Journal of Numerical Analysis* 42.4, pp. 3430–3463.

3. "scipy.optimize.newton_ krylov function" (n.d.). In: *Scipy Documentation* https://docs.scipy.org/doc/sci
().