SMAI-JCM
SMAI Journal of Computational Mathematics

Parallel reverse time integration and reduced order models

Bijan Mohammadi

Volume 1 (2015), p. 5-28.
<http://smai-jcm.cedram.org/item?id=SMAI-JCM_2015__1__5_0>

SMAI Journal of Computational Mathematics
Vol. 1, 5-28 (2015)

Parallel reverse time integration and reduced order models

Bijan Mohammadi [1]

[1] Montpellier University, Mathematics, CC51, 34095 Montpellier, France
E-mail address: bijan.mohammadi@umontpellier.fr.

Abstract. We discuss complexity issues in time dependent adjoint evaluation. We address the question of storage complexity and redundant calculation of intermediate states in adjoint calculations for time dependent flows. Parallel in time solutions are introduced in reverse time integration together with reduced order modelling for the recovery of intermediate forward states between checkpoints.

The approach is illustrated on an identification problem from partial macroscopic variables fields observations and also in the context of shape sensitivity evaluation in fluids for the pressure and viscous drag coefficients.

Math. classification. 65Y00, 65Y05, 68W10, 35Q93, 90C52.

Keywords. LBM, discrete adjoint, meta model, uncertainty, contour identification, shape optimization, parallel time reversal.

1. Introduction

Large dimensional sensitivity analysis is of utmost importance in many applications among which optimization and design, of shapes for instance, or identification of parameters in models or contours in an image or in a flow. The interest for sensitivity analysis also grows with the increasing needs for uncertainty quantification in simulations and the curse of dimensionality with the limitations of statistical tools when control spaces are of large dimension.

For instance, the first order Sobol indices [34, 35] for a functional $J(X), X = (X_1, .., X_n) \in \Pi \subset \mathbb{R}^n$ of random inputs $X_{i=1,..,n}$ can be estimated if the sensitivities of the functional with respect to the entries are known and are bounded. More precisely, one can prove that [35]:

$$s_1^i = \frac{Var(\mathbb{E}(J(X)|X_i))}{Var(J)} \leq \frac{C}{Var(J)} \int_\Pi \left(\frac{\partial J}{\partial X_i}\right)^2 dx_i,$$

where C gives an information on Π. In particular, in cases where X_i are Gaussian variables $X_i = N(\mu_i, \sigma_i^2)$ we have $C = \sigma_i^2/Var(J)$. Hence, when available, gradients can be useful beyond their traditional applications and also address quantitative robustness issues both in simulation and design. The interest for the gradients grows even more if these are accessible through an adjoint formulation with a cost of evaluation independent of n.

Adjoint techniques have been widely used with fluids, and in particular for aeronautics applications, with first applications back to the 70's [31, 32]. They are now the natural way to obtain the sensitivity of a functional in high dimensional control spaces. One difficulty is that developing an adjoint solver means extra development effort and also difficulty to maintain the adjoint code up to date with respect to the forward solver.

Automatic differentiation (AD) [12] tools have been of some help reducing the programming effort. But they appear often inefficient without some intervention and monitoring by the programmer. In previous works we showed how to use discrete adjoint constructions for large dimensional sensitivity analysis in robust shape design through adequate multi-point optimization [22, 24, 23] avoiding any sampling of a large dimensional control space.

But most of the previous works with the adjoint, in aeronautics for instance, is for steady flow configurations and the question of the complexity of backward sensitivity analysis for time dependent problems still remains an issue. This question concerns, for instance, the storage of forward states necessary during backward integration and the difficulty with redundant calculations in case of partial storage.

2. **Summary of the work**

The paper aims at discussing the complexity issues in time dependent adjoint calculations with emphasis on the identification of hidden parallelism. It proposes different possible strategies to break this complexity introducing a priori information on the physics of the problem and the use of meta-models when available.
The ingredients we propose are generic. We illustrate them on a model problem and for the flow solution by a Lattice Boltzmann Method (LBM). LBM is a good example as both the steady and unsteady situations follow a same calculation procedures and it is challenging as the volume of data in time is much higher than with classical incompressible Navier-Stokes fluid solvers (i.e. per time step, in 2D a minimum of 9 variables per node with the D2Q9 lattice instead of 3 with a Navier-Stokes solver and in 3D a minimum of 19 with the D3Q19 lattice instead of 4). Also, LBM consists of successive applications of two linear and nonlinear operators which need each specific treatment with the adjoint.
A major interest here is to see how to reduce the complexity of these sensitivity evaluations using reduced order modelling and functional reformulation. These are specially interesting in situations where the simulation chain cannot be fully differentiated, because of being partly available as a black-box solver for instance.
We discuss two situations. First we show that drastic simplification of the contributing terms to the sensitivity can be achieved for some specific functionals through the incomplete sensitivity concept [25]. Then, we see how the introduction of Bayesian-like a priori assumption on the behavior of a variable permits to improve the prediction of the sensitivities.
The paper illustrates its ingredients through two applications. First, the pertinence of the different strategies for the exact and approximate adjoints is analyzed on a problem of contours identification from partial macroscopic field observation where a level set parameterization is used to account for the presence of obstacles in the field. Then, incomplete sensitivity theory is used to reduce the complexity of adjoint evaluations for a shape sensitivity analysis problem for cost functions based on the pressure and viscous contributions to the aerodynamic drag coefficient.

3. **Flow solver**

As we said, the ingredients of the paper can be used with any time integration procedure. We choose to illustrate the approach with a Lattice Boltzmann Method for the solution of low speed flows and described in appendix A. We emphasis, in particular, on the Lattice Boltzmann Method chosen, the way solid walls are accounted for through a regularized level set function and the implementation of the boundary conditions for and finally the steps which are taken at each time iteration by the solver. This latter is important to our discussion on adjoint formulation.

3.1. **Backward linearization of a flow solver**

Let us consider the Navier-Stokes equations for incompressible fluids with initial $u(0,x) = u_0(x)$ and appropriate boundary conditions describing the velocity vector field u and pressure p for $(t,x) \in$

$[0,T]\times\Omega$:

$$u_t + (u\cdot\nabla)u - \nu\Delta u + \nabla p = 0, \quad \nabla\cdot u = 0. \tag{3.1}$$

And its adjoint equations (see [31, 32, 25] for instance for details) for the solution of Lagrange multipliers $\tilde{u}$ and $\tilde{p}$ having the same structure than u and p respectively, with final value $\tilde{u}(T,x)=0$ and appropriate boundary conditions:

$$-\tilde{u}_t - (\nabla\tilde{u})^T u + (\nabla u)\tilde{u} - \nu\Delta\tilde{u} - \nabla\tilde{p} = -J_u, \quad -\nabla\cdot\tilde{u} = -J_p. \tag{3.2}$$

The equations involve the partial derivatives of a functional J. In the sequel, we see several examples of J.

Hence, backward linearization of time dependent nonlinear state equations, such as this Navier-Stokes equations, requires the storage of the intermediate states (here $u(t,x), 0\le t\le T$).

In the Navier-Stokes equations, this is due to the presence of the nonlinear advection operator $(u\cdot\nabla)u$ and the right-hand sides, the other operators being linear (i.e. ∇p, $\nu\Delta u$, $\nabla\cdot u$).

It is fair to think that switching to a Lattice Boltzmann formulation should not modify this complexity. On the other hand, the LBM would have an enormous advantage over the classical PDE based formulation in sensitivity analysis. Considering the steps of our Lattice Boltzmann formulation in appendix A, one sees that unlike in the PDE based formulation where the nonlinearity is in the transport operator, the nonlinearity there is in the collision step which is at the origin of the diffusion mechanism in the Navier-Stokes equations. We develop this analysis in the next section.

4. **Sensitivity analysis**

The time dependent nonlinear equations solver we use to illustrate our ingredients (based on a Lattice Boltzmann Method presented in appendix A) appears as part of the following dependency chain in an optimization or identification problem:

$$\psi \to \{\mathbf{f}(\psi,t), t\in[0,T]\} \to J(\psi, \mathbf{U}(\mathbf{f}(\psi, t\in[0,T]))), \tag{4.1}$$

where ψ is the independent variable (here the shape parameterization). $\mathbf{U} = (\rho, u)^t$ gathers the macroscopic density and velocity distributions, $\mathbf{f} = (f_0, .., f_q)^t$ (for a DdQq stencil) and J is a scalar functional. To be accurate, one should have also considered the independent physical parameters $(\tau, \omega_i, c_i,$ etc.). But, this would have brought unnecessary complications to the notations.

As presented in sections 11.1 and 11.4, $\mathbf{f}$ can be formally seen as solution of a time dependent equation involving a first order time derivative:

$$\partial_t\mathbf{f} + F(\mathbf{f}, \mathbf{U}(\mathbf{f}), \psi) = 0, \ \mathbf{f}(0) = \mathbf{f}_0(\psi), \tag{4.2}$$

where $\mathbf{f}_0(\psi)$ is the initial density distribution on the lattice. Now consider a functional involving an integral over time and the macroscopic variables. Steady situations or when the functional is defined at a given time are particular cases of this:

$$J = \int_{(0,T)\times\Omega} j(\psi, \mathbf{U}(\mathbf{f}(\psi,t))), \tag{4.3}$$

where Ω denotes the lattice. There is usually no direct dependency between j and $\mathbf{f}$ as the functionals usually involve macroscopic quantities.

4.1. **Adjoint formulation**

Let us formally describe the adjoint method in the context of the LBM time dependent equations.

Linearizing J with respect to ψ one has:

$$\nabla_\psi J = \int_{(0,T)\times\Omega} (j_\psi + j_{\mathbf{U}}\mathbf{U}_{\mathbf{f}}\mathbf{f}_\psi).$$

In this expression $\mathbf{U}_\mathbf{f}$ is easy to get as the relation (11.2) between $\mathbf{f}$ and $\mathbf{U}$ is algebraic. Also, j_ψ and $j_\mathbf{U}$ are usually easy to analytically derive. $\mathbf{f}_\psi$, on the other hand, is costly to get as it requires the linearization of the LBM solver.

The linearized LBM solver for $\mathbf{f}_\psi$ can be formally seen as:

$$\partial_t(\mathbf{f}_\psi) + F_\psi + (F_\mathbf{U}\mathbf{U}_\mathbf{f} + F_\mathbf{f})\mathbf{f}_\psi = 0,\ \mathbf{f}_\psi(0) = \mathbf{f}_0'(\psi). \tag{4.4}$$

It permits to write for all function ϕ (where ϕ has the same structure than $\mathbf{f}$):

$$0 = \int_{(0,T)\times\Omega} \Big(\partial_t(\mathbf{f}_\psi) + F_\psi + (F_\mathbf{U}\mathbf{U}_\mathbf{f} + F_\mathbf{f})\mathbf{f}_\psi\Big)\ \phi.$$

We introduce the adjoint LBM operator $\mathbf{F}^*(\mathbf{f}, \mathbf{U}(\mathbf{f}), \psi) = (F_\mathbf{U}\mathbf{U}_\mathbf{f} + F_\mathbf{f})^*$ which corresponds to the adjoint of one step of the LBM described in section 11.4. This operator will be defined using automatic differentiation in section 5. Beyond what presented here, the LBM operator might be quite complex, including other ingredients to account for extra physics or numerical accuracy (turbulence modelling, wall functions, higher order scheme, etc). It is therefore interesting to handle the definition of $\mathbf{F}^*$ in an automatic manner and, in particular, separate it from the mentioned intermediate states storage complexity.

Integrating by parts and using $\mathbf{F}^*$ we have:

$$0 = \int_{(0,T)\times\Omega} (-\partial_t\phi + \mathbf{F}^*\ \phi)\mathbf{f}_\psi + \int_\Omega [\phi\mathbf{f}_\psi]_0^T + \int_{(0,T)\times\Omega} \phi F_\psi(\mathbf{f}, \mathbf{U}(\mathbf{f}), \psi).$$

Introducing the linear backward adjoint problem, we have:

$$\partial_t\phi + \mathbf{F}^*(\mathbf{f}, \mathbf{U}(\mathbf{f}), \psi)\phi = j_\mathbf{U}\mathbf{U}_\mathbf{f}(\mathbf{f}), \qquad \phi(T) = 0. \tag{4.5}$$

Which permits to eventually have:

$$\int_{(0,T)\times\Omega} j_\mathbf{U}\mathbf{U}_\mathbf{f}\mathbf{f}_\psi = \int_\Omega \phi(0)\mathbf{f}_0'(\psi) - \int_{(0,T)\times\Omega} \phi F_\psi(\mathbf{U}, \psi), \tag{4.6}$$

with ϕ solution of the backward adjoint equation (4.5) for the chosen final condition.

If there is no direct dependency between the initial condition $\mathbf{f}(0)$ and ψ the first term in the right-hand-side vanishes. In our implementation of the LBM, the initial density distribution is uniform and the presence of the obstacles are accounted for at step 2 of the algorithm given in section 11.4 and then at each of the time iterations. This is actually where the direct dependency in ψ is in F and linearizing (11.5) provides F_ψ.

As described in section 11.3 we use either periodic, slip or no-slip boundary conditions for the density $\mathbf{f}$. These give the corresponding boundary conditions for the adjoint variable ϕ.

Suppose N^d is the lattice size in d dimensions in space, then calculation and memory complexities can be estimated. Variable $\mathbf{f}_\psi$ in (4.5) is of size qN^{2d}. Calculating $\mathbf{f}_\psi$ means therefore solving N^d times equation (4.2). We recover the expected complexity of the direct mode. On the other hand, as $\mathbf{f}$, ϕ is of dimension qN^d but for its calculation we need states $\mathbf{f}$ in the reverse order because of the backward integration in time in (4.5).

One possibility to avoid this difficulty is to consider an approximation of the adjoint by considering only one time step in the direct LBM solution [30]. Previous tentative with automatic differentiation of LBM codes [16, 36] also show that brute force approach cannot be successful because of this storage complexity issue even with the introduction of a check-pointing technique to optimize the balance between intermediate storage and state recalculation [12, 14, 13, 11]. We discuss these options in the next section.

5. Discrete adjoint by Automatic Differentiation

The previous analysis has been implemented by automatic differentiation applied to our Lattice Boltzmann solver. In this section, we describe how the `Tapenade` AD tool [14] treats this problem. Appendix B gives a short description of automatic differentiation in direct and reverse modes.
Our LBM program can be seen taking the following steps. The program linking the independent variables to the functional J is called the forward code.

Forward code:
$\mathbf{f}$ = given, $J = 0$,
do iter = 1,.., iter_{max} (time iterations)
$\mathbf{f}_1 = l_1(\mathbf{f})$ (collision, n_1 relations, $l_1 : \mathbb{R}^{qN^d} \to \mathbb{R}^{qN^d}$)
$\mathbf{f}_2 = l_2(\mathbf{f}_1)$ (transport and boundary conditions, n_2 relations, $l_2 : \mathbb{R}^{qN^d} \to \mathbb{R}^{qN^d}$)
$\mathbf{f} = \mathbf{f}_2$ this is a linear operation and no storage is necessary.
$\mathbf{U} = l_3(\mathbf{f})$ (macroscopic variables, n_3 relations, $l_3 : \mathbb{R}^{qN^d} \to \mathbb{R}^{(d+1)N^d}$)
$J = J + l_4(\mathbf{U})$ (cost function, n_4 relations, $l_4 : \mathbb{R}^{(d+1)N^d} \to \mathbb{R}$)
done

The number of substitutions ('=' relationship) roughly describes the size of a program. What is inside the loop is the kernel of our LBM code and we will discuss how to separate its linearization from the rest of the program.

The adjoint code method [14, 25] (see appendix B for an example) considers a given computer program in the reverse order of execution and produces a new program where a given line $y = y + g(x)$ of the initial program gives $p_x = p_x + g' p_y$. A new complementary variable is introduced at each substitution and the intermediate forward states have been stored before each substitution for reverse integration (except if the program is identified to be linear with respect to its inputs).

A given complementary variable p_x has the same structure than x. Hence, below $\mathbf{p_f}$, $\mathbf{p_{f_1}}$ and $\mathbf{p_{f_2}}$ are in $\mathbb{R}^{qN^d}$, $\mathbf{p_U}$ in $\mathbb{R}^{(d+1)N^d}$ and $\mathbf{p_J}$ is scalar. All the complementary variables receive a zero initialization, except the last one which is set to one which constitutes the initialization of the first variable in the reverse integration.

After executing the direct LBM code above and storage of all intermediate variables, a sketch of the reverse mode code is as follows:

Reverse code:
$\mathbf{p_J} = 1, \mathbf{p_f} = \mathbf{p_{f_1}} = \mathbf{p_{f_2}} = 0, \mathbf{p_U} = 0$,
do iter = $\text{iter}_{max}, .., 1$ (reverse time iterations)
$\mathbf{p_U} = \mathbf{p_U} + l_4'^*(\mathbf{U}(\text{iter}))\mathbf{p_J}$, with $l_4'^* : R \to \mathbb{R}^{qN^d}$
$\mathbf{p_f} = l_3'^*(\mathbf{f}(\text{iter}))\mathbf{p_U}$, with $l_3'^* : R^{(d+1)N^d} \to \mathbb{R}^{qN^d}$
$\mathbf{p_{f_2}} = \mathbf{p_f}$
$\mathbf{p_{f_1}} = l_2'^*(\mathbf{f_1}(\text{iter}))\mathbf{p_{f_2}}$ with $l_2'^* : R^{qN^d} \to \mathbb{R}^{qN^d}$
$\mathbf{p_f} = l_1'^*(\mathbf{f}(\text{iter}))\mathbf{p_{f_1}}$ with $l_1'^* : R^{qN^d} \to \mathbb{R}^{qN^d}$
done

This is how the `tapenade` AD tool works and generates the discrete adjoint code.

6. Storage complexity of the AD reverse code

The total memory necessary for storage in our situation can be estimated as

$$\text{iter}_{max}(qN^d(n_1 + n_2) + (d+1)N^d n_3).$$

One can reduce the storage complexity in such a situation accepting redundant calculations. Checkpointing is available in `tapenade` [11, 13]. One defines optimal storage moments and recompute missing variables in between. This reduces the storage complexity to $\log_2(\text{iter}_{max})(qN^d(n_1+n_2)+(d+1)N^d n_3)$ which is still very demanding. This is the best complexity we can have if applying the AD tool directly to the whole code without any user intervention.

6.1. **User intervention**

We would like to go one step further requiring some user intervention. After the previous differentiation giving the reverse code with the mentioned storage complexity, we apply the automatic differentiation tool individually to each operator $l_i, i = 1, 2, 3, 4$ and generate $l_i'^*$. The user gathers the operators l_i and $l_i'^*$ and generates new operators we call $l_i \cup l_i'^*$. The user then assembles these pieces to build the reverse code with the aim of not storing more than one intermediate density by time iteration in the forward run. Applied together with the checkpointing strategy the total storage requirement can be reduced to $\log_2(\text{iter}_{max})qN^d$. This is the best one could expect in term of storage complexity and it is obviously still very challenging. In addition, this is achieved introducing a huge amount of redundant calculations. Figure 6.1 shows a sketch of this strategy. One of our aims here is to reduce or remove these redundant calculations and also, if possible, reduce even more the storage complexity. We will see that these can be achieved in specific physical situations and also introducing dynamic meta-models based on checkpoint states.

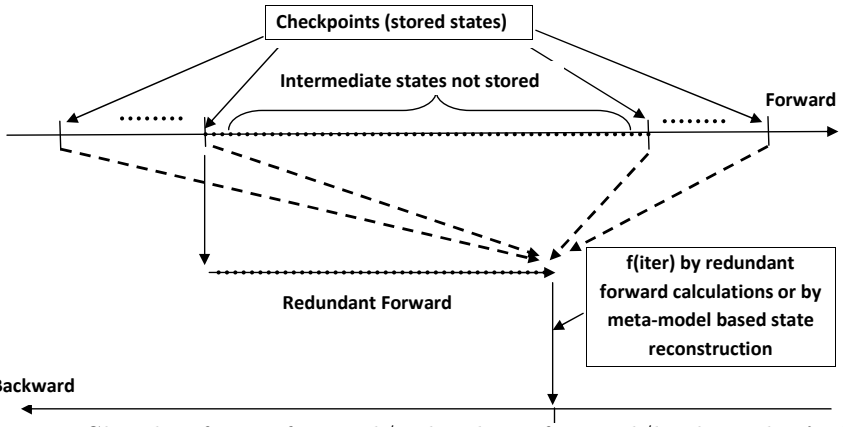

Figure 6.1. Sketch of our forward/redundant forward/backward combination, or when the redundant forward is replaced by a meta-model based reconstruction of non available intermediate states using checkpoint states.

6.2. **Steady flows**

A major complexity reduction can be achieved when looking for steady flows as limit solutions in a time marching procedure. Again, our LBM solver can be seen as discrete form of (4.2):

$$\mathbf{f}^{n+1} = \mathbf{f}^n - F(\mathbf{f}^n, \mathbf{U}(\mathbf{f}^n), \psi),\ \mathbf{f}^0 = \text{given}, \tag{6.1}$$

and steady solution is when $\|\mathbf{f}^{n+1} - \mathbf{f}^n\| \to 0$. We denote by $\mathbf{f}^\infty$ this solution. The reverse code can be seen as discrete form of equation (4.5) and taking advantage of the fact that we look for the sensitivity

for the stationary solution, we replace $\mathbf{f}(t)$ in this equation by $\mathbf{f}^\infty$:

$$\partial_t \phi + \mathbf{F}^*(\mathbf{f}^\infty, \mathbf{U}(\mathbf{f}^\infty), \psi)\phi = j_{\mathbf{U}} \mathbf{U}_{\mathbf{f}}(\mathbf{f}^\infty), \qquad \phi(T) = 0. \tag{6.2}$$

This can be achieved in our reverse code in section 5 by replacing $\mathbf{f}(\text{iter})$ by $\mathbf{f}(\text{iter}_{max})$ which corresponds to the steady solution by the forward code and iter_{max} is the number of iterations which were necessary to reach it. This, however, represents many manipulations of the AD generated codes by the user (which are not easy to read in general). A simpler way to proceed is to differentiate one iteration of the forward code and then change the number of iterations in the backward loop to iter_{max} and initiate the calculation with the stationary solution, previously computed. These practical issues aside and the final (steady) state being obviously available, there is no extra storage requirement [4, 33, 25].

6.3. **Meta-models to avoid redundant calculations**

Up to now we did not introduce any major approximation, even when addressing the steady state situation. At this point we propose to introduce dynamic meta-models for the density to avoid the redundant calculations to recover unavailable states between two checkpoints. Of course, many reduced order modelling approaches exist and the choice of a particular meta-model heavily depends on the domain of application and the nature of the equations. The checkpointing theory tells how to choose n_{cp} checkpoint locations in order to minimize the redundant calculations [11]. As with our meta-models we will not have any redundant calculations, we equidistribute our n_{cp} checkpoints every iter_{max}/n_{cp} iterations. Let us present the idea with data interpolation which is the most simple approach to build a reduced-order model based on a polynomial parametric approximation of a function known over a set of points. For instance, using a linear combination of $\mathbf{f}$ at checkpoints T_i we have for $1 \le t \le \text{iter}_{max}$:

$$\tilde{\mathbf{f}}(t, x) = \sum_{j=1}^{n_{cp}} \lambda_j(t) \mathbf{f}(T_j, x), \ 0 \le \lambda_j(t) \le 1, \tag{6.3}$$

where $\lambda_j(t)$ are, for instance, barycentric functions such that

$$\sum_{j=1}^{n_{cp}} \lambda_j(t) = 1, \quad \text{and } \lambda_j(T_i) = \delta_{ij}.$$

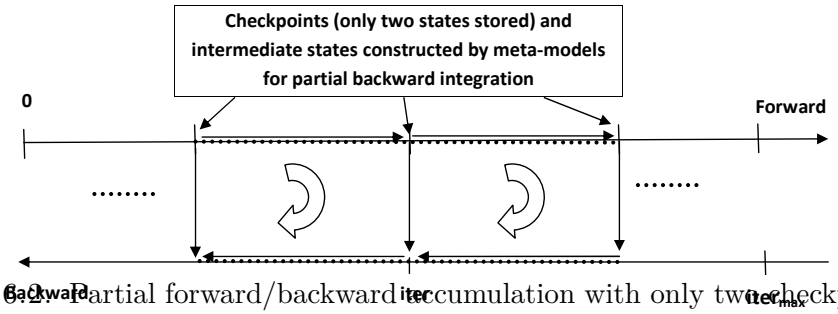

Figure 6.2. Partial forward/backward accumulation with only two checkpoint states stored and dynamically erased.

6.4. **Partial forward/backward accumulation**

Another major simplification one might be tempted with is to consider the backward integration only on a partial forward time window. This is illustrated in figure 6.2. To have a more detailed description

of this choice consider functional (4.3) and, for the sake of simplicity, $n+1$ uniformly distributed checkpoints at $t = T_i, i = 0, .., n$. The functional can be rewritten as:

$$J = \sum_{i=1}^{n} \int_{T_{i-1}}^{T_i} \int_\Omega j(\psi, \mathbf{U}) = -\sum_{i=n}^{1} \int_{T_i}^{T_{i-1}} \int_\Omega j(\psi, \mathbf{U}), \tag{6.4}$$

with $T_0 = 0$ and $T_n = T$. These representations are interesting as they permit to localize in time the derivation with respect to ψ. Reconsidering the analysis presented in section 4.1 for each interval $]T_{i-1}, T_i[$ we have:

$$\sum_{i=1}^{n} \int_{(T_{i-1},T_i)\times\Omega} j_{\mathbf{U}} \mathbf{U}_{\mathbf{f}} \mathbf{f}_\psi = \sum_{i=1}^{n} \left(\int_\Omega \phi(T_{i-1}) \mathbf{f}'_\psi(T_{i-1}) - \int_{(T_{i-1},T_i)\times\Omega} \phi F_\psi(\mathbf{U}, \psi) \right), \tag{6.5}$$

where $\mathbf{f}'_\psi(T_{i-1}) = 0$ as there is obviously no direct dependency between ψ and $\mathbf{f}(T_{i-1})$. The adjoint variable ϕ over this time interval is solution of:

$$\partial_t \phi + \mathbf{F}^*(\mathbf{f}, \mathbf{U}(\mathbf{f}), \psi)\phi = j_{\mathbf{U}} \mathbf{U}_{\mathbf{f}}(\mathbf{f}), \qquad \phi(T_i) = \phi^+(T_i), \tag{6.6}$$

where $\phi^+(T_i)$ indicates the solution at time $t = T_i$ of the adjoint equation over the interval (T_i, T_{i+1}) with $\phi^+(T_n) = 0$.

Partial forward/backward accumulation can be defined by simply setting $\phi^+(T_i) = 0$ for all i. And the introduction of a meta-model for $\mathbf{f}$ can be seen as solving:

$$\partial_t \phi + \mathbf{F}^*(\tilde{\mathbf{f}}, \mathbf{U}(\tilde{\mathbf{f}}), \psi)\phi = j_{\mathbf{U}} \mathbf{U}_{\mathbf{f}}(\tilde{\mathbf{f}}), \qquad \phi(T_i) = \phi^+(T_i), \tag{6.7}$$

where $\tilde{\mathbf{f}}(t \in]T_{i-1}, T_i[)$ is a linear interpolation between checkpoints $\mathbf{f}(T_{i-1})$ and $\mathbf{f}(T_i)$.

6.5. **Parallel fixed point partial backward with meta-forward**

We introduced two ingredients to address two difficulties in adjoint calculation for time dependent situations: forward states storage and redundant calculations in case of partial storage. And, we presented two alternatives to address these issues: meta-model construction for intermediate state recovery and partial forward/backward accumulation. Of course, each of them introduces some approximations and therefore errors. Here we would like to reduce some of these uncertainties taking advantage of the natural parallelism present in the two ingredients as illustrated in figure 6.3.

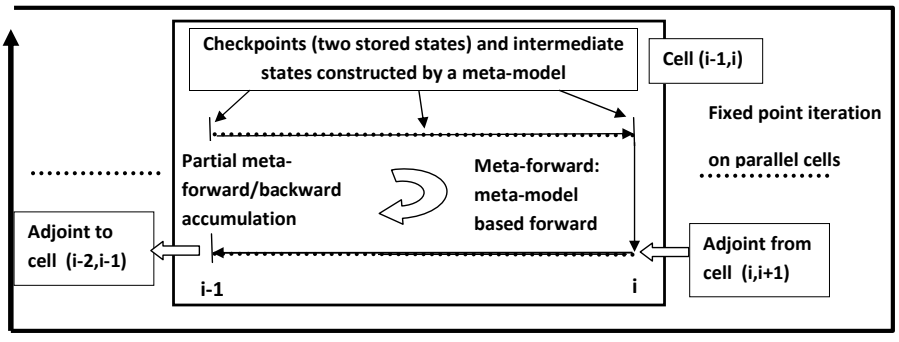

Figure 6.3. Parallel partial meta-forward/backward accumulations for independent cells $(i-1, i), i = 1, .., n$.

Figure 6.3 indicates how a fixed point iteration can be created over independent problems for cells $(T_{i-1}, T_i), i = 1, .., n$ where on each cell one solves a partial meta-forward/backward accumulation problem using the two ingredients presented above:

$$\partial_t \phi_k + \mathbf{F}^*(\tilde{\mathbf{f}}, \mathbf{U}(\tilde{\mathbf{f}}), \psi)\phi_k = j_{\mathbf{U}} \mathbf{U}_{\mathbf{f}}(\tilde{\mathbf{f}}), \qquad \phi_k(T_i) = \phi^+_{k-1}(T_i), \tag{6.8}$$

where $\phi^+_{k-1}(T_i)$ indicates the adjoint computed at the previous fixed point iteration by the sub-problem solved on cell (T_i, T_{i+1}). The fixed point iterations aim at removing the error in the partial accumulation which can be measured at its k^{th} iteration by:

$$\varepsilon_b = \sum_{i=1}^{n-1} \|\phi_k(T_i) - \phi_{k-1}(T_i)\|. \tag{6.9}$$

At each iteration of the fixed point, a cell problem on (T_{i-1}, T_i) is independent once it receives the adjoint $\phi^+_k(T_i)$ accumulated over cell (T_i, T_{i+1}). Eventually, it then has to communicate to the cell (T_{i-2}, T_{i-1}) his contribution at $t = T_{i-1}$ denoted by $\phi^+_k(T_{i-1})$. Of course, the approximation due to the introduction of the meta-model for intermediate states recovery will still be present.

6.5.1. *Links with the multiple shooting and the parareal algorithms*

This algorithm is similar in spirit to the multiple shooting algorithm [3] and to the parareal method [19, 8]. In these methods, introduced independently by the authors, the solution of a Cauchy problem is replaced by those of parallel coupled forward initial value problems on successive sub-intervals as in our case. The motivations for the introduction of the methods have been different and related to parallelization in time for the latter and error and stability control in solution of differential algebraic equations for the former.

With these algorithms, solution of problem (4.2) on $(0, T) = \cup_{i=1,n}(T_{i-1}, T_i)$ is replaced by iterations (denoted by k) of n parallel sub-problems:

$$\partial_t \mathbf{f}_k + F(\mathbf{f}_k, \mathbf{U}(\mathbf{f}_k), \psi) = 0, \ \mathbf{f}_k(T_{i-1}) = \mathbf{f}^-_{k-1}(T_{i-1}), \ \text{on cell } (T_{i-1}, T_i), \tag{6.10}$$

where $\mathbf{f}^-_{k-1}(T_{i-1})$ comes from the solution of the sub-problem for cell (T_{i-2}, T_{i-1}) if $i > 1$ as $\mathbf{f}_k(T_0) = \mathbf{f}_0$ is given for all k. An error indicator, similar to (6.9) measures the convergence of the forward parallel iterations:

$$\varepsilon_f = \sum_{i=1}^{n-1} \|\mathbf{f}_k(T_i) - \mathbf{f}_{k-1}(T_i)\|. \tag{6.11}$$

The complexity of these algorithms in term of number of iterations necessary to reduce sufficiently ε_f and ε_b is difficult to predict in distributed situations involving the solution of partial differential equations. With ordinary differential equations this complexity is bounded by n as we will see in section 7. This complexity can be reduced by using a reduced order model acting as either a preconditioner or in a sequential predictor step as in the parareal method [19, 8]. An interest of such splitting approaches is the possibility of using different numerical methods on the different cells, with different accuracy for instance.

7. **A model problem**

Let us illustrate the previous alternatives on a simple model:

$$v_t = \sin((v^2 + 1)t), \ v(0) = v_0 = 0, \ J(v_0) = \int_0^{100} v^2(t)dt.$$

Of course, in this case a direct linearization is possible because the control parameter is one dimensional. But, we would like to evaluate dJ/dv_0 with an adjoint formulation as in section 4.1. Denoting

the adjoint variable by v^* and here F being $\sin((v^2+1)t)$, a similar analysis than above gives for expression (4.6):

$$\frac{dJ}{dv_0} = \int_0^{100} 2vv_{_{v_0}} = v^*(0)v_0'(v_0) - \int_0^{100} v^* F_{_{v_0}} = v^*(0),$$

as $v_0'(v_0) = 1$ and $F_{_{v_0}} = 0$. To remain close to our LBM problem, the equation is solved with a backward Euler scheme with a time step of 0.1 s. Figure 7.1 shows dJ/dv_0 evaluated backward with a full adjoint (100 forward states stored) and using 5, 10 and 20 uniform checkpoints and between a linear interpolation as meta-model. We see that increasing the number of checkpoints improves the accuracy of the gradient as reducing the size of the time subdomain improves the accuracy of the meta-model. This approximation permits therefore to provide the adjoint with a prescribed memory requirement for the checkpoints. The necessary memory growths with the requested accuracy for the gradient. The necessary precision usually increases during an optimization with a descent method. Figure 7.2 shows the impact of partial reverse accumulation presented in figure 6.2. We saw that with this algorithm only two checkpoints are stored and dynamically reallocated and again a meta-model is used for intermediate states reconstructions. Here the best result is with fewer checkpoints which is reasonable as the partial reverse integration covers a larger portion of the whole integration time. Without a meta-model, fewer checkpoints would have implied more redundant calculations.

To go one step further and address this loss of accuracy, figure 7.3 shows the application of the parallel fixed point iterations applied to the partial backward accumulation with exact and meta-forward states described in 6.4. One sees that with 5 checkpoints and exact forward states the gradient is fully recovered after 3 iterations of the fixed point algorithm. The error indicator ε_b given by (6.9) is shown during fixed point iterations for 5, 10 and 20 checkpoints and it shows that the number of the fixed point iterations necessary to recover the gradient increases linearly with the number of the checkpoints and is bounded by the this number.

More generally, for the solution of backward ordinary differential equations, a worse case analysis gives a maximum of n iterations of the algorithm to fully remove the error introduced by the partial backward accumulation approximation. Indeed, n iterations will permit to an information at $T_n = T$ to reach $T_0 = 0$ as in a sequential calculation. One saw from figure 7.3 that the number of iterations is often much less than n. This analysis cannot be simply extended to distributed systems involving partial differential equations as with the LBM method as we will see in section 9.

The error introduced in the gradient when using a meta-model for the intermediate states cannot be removed with this fixed point iterations. It can, however, be reduced increasing the number of checkpoints. One sees that a compromise can be found with 20 checkpoints between accuracy, absence of redundant calculations and checkpoints storage. One also takes advantage of the fact that the calculations on the 20 cells between two checkpoints are fully parallel.

8. **Low complexity models and specific functionals**

Our aim in this section is to see how to reduce the complexity of these sensitivity evaluations using reduced order modelling and functional reformulation. Indeed, we have observed that some functionals are more suitable for sensitivity evaluation in term of calculation complexity [25]. One situation where this observation receives even more importance is when the functional enters in the domain of application of the concept of incomplete sensitivities. Beyond the computational complexity issue, the necessity for such alternatives also comes from the fact that it is not always possible to proceed with the linearization of the direct simulation chain used for the definition of J. Let us reconsider the simulation chain (4.1) where the variables have been split in two categories following their nature in term of computational complexity. This is a very common situation.

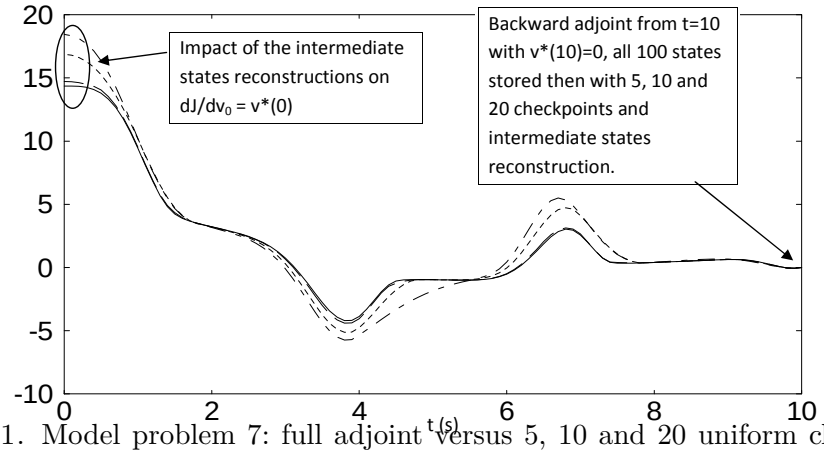

FIGURE 7.1. Model problem 7: full adjoint versus 5, 10 and 20 uniform checkpoints with the intermediate states reconstructed from these by a linear interpolation.

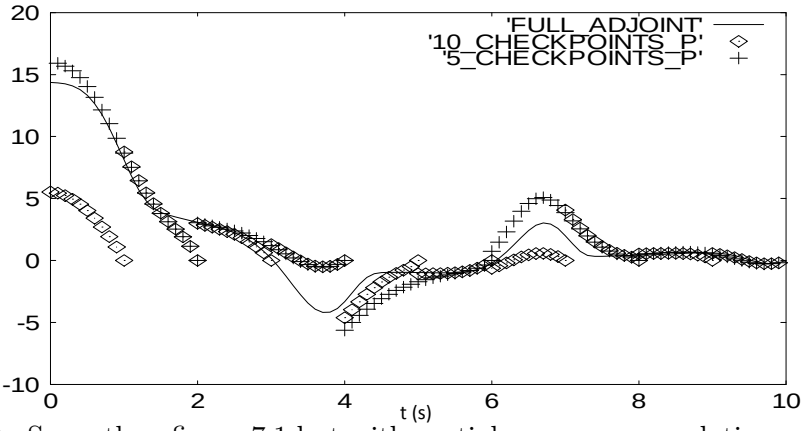

FIGURE 7.2. Same than figure 7.1 but with partial reverse accumulation and dynamic reallocation of the checkpoints. The same intermediate states reconstruction has been applied.

$$\psi \to q(\psi) \to \{\mathbf{f}(q(\psi),t), t \in [0,T]\} \to J(\psi, q(\psi), \mathbf{U}(\mathbf{f}(q(\psi), t \in [0,T]))), \tag{8.1}$$

where q is cheap to compute and represents here geometrical quantities. The other dependent variables $\mathbf{f}$ and $\mathbf{U}$ are expensive to compute as solution of the LBM solver. The gradient of J with respect to ψ is:

$$\nabla_{\psi} J = J_{\psi} + (J_q + J_U U_{\mathbf{f}} \mathbf{f}_q) q_{\psi}, \tag{8.2}$$

where J_{ψ}, J_q and $J_{\mathbf{U}}$ are easy to access and are usually provided by the user as external modules in an industrial simulation platform. $\mathbf{f}_q$ and q_{ψ} are, on the other hand, difficult to access. Also, the major part of the cost of this evaluation is due to $U_{\mathbf{f}} \mathbf{f}_q q_{\psi}$ and its evaluation with an adjoint method has been discussed throughout the paper. Today's industrial simulation platforms are more and more based on black-boxes and commercial packages not enabling the user for a direct access to the source of the codes. Linearizing the simulation codes by automatic differentiation is therefore off the table. In the same way, it is quite inconceivable to develop in house adjoint solvers when the cost function

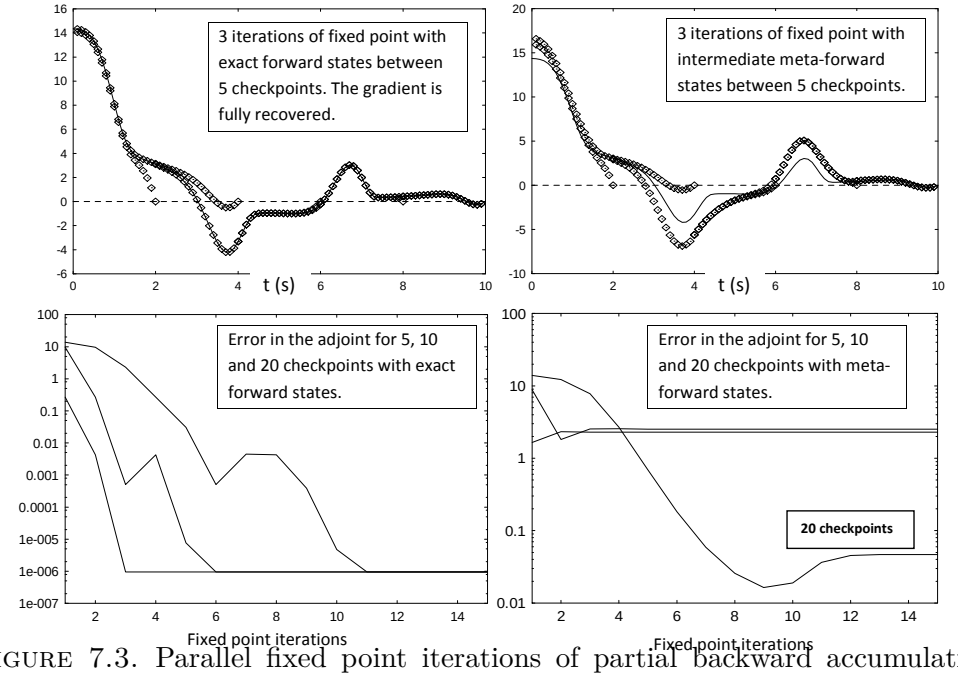

FIGURE 7.3. Parallel fixed point iterations of partial backward accumulation with meta-forward states described in 6.4. Upper left: with 5 checkpoints and exact forward states the gradient is fully recovered after 3 iterations of the fixed point algorithm. Upper right: the error in the gradient is due to the forward states between checkpoints based on a meta-model. Lower-left: error ε_b by (6.9) during fixed point iterations for 5, 10 and 20 checkpoints. Lower-right: same but with meta-forward states. One sees that a compromise can be found with 20 checkpoints between accuracy, absence of redundant calculations and checkpoints storage.

calculation relies on commercial packages. The only realistic gradient calculation approach with black-boxes is with finite difference approximation which has a cost proportional to the size of the control parameter space. This approach therefore is not an option either.

Suppose we have $\tilde{q}$ and $\tilde{\mathbf{U}}$ two reduced order models for q and $\mathbf{U}$ with relations $\Phi\tilde{q}(\psi) = q(\psi)$ and $\Psi\tilde{\mathbf{U}}(\tilde{q}(\psi)) = \mathbf{U}(q(\psi))$. Φ and Ψ are transfer functions which will be frozen during the linearization. One aims through $\tilde{\mathbf{U}}$ to remove the dependency with respect to $\mathbf{f}$, using macroscopic fluid models for instance. We consider the following approximate simulation chain where black-box and high complexity terms have been approximated:

$$\tilde{J} : \psi \to \tilde{q}(\psi) \to \tilde{\mathbf{U}}(\tilde{q}(\psi)) \to J\left(\psi, \Phi\tilde{q}(\psi), \Psi\tilde{\mathbf{U}}(\tilde{q}(\psi))\right). \tag{8.3}$$

We need both $\tilde{q}$ and $\tilde{\mathbf{U}}$ as q is now often part of a black-box package as when using Computer Aided Design (CAD) tools.

Linearizing (8.3) gives:

$$\nabla_{\psi}\tilde{J} = J_{\psi} + (J_q + J_{\mathbf{U}}(\Psi\tilde{\mathbf{U}}_{\tilde{q}}\ (\Psi^{-1}\mathbf{U})))\Phi\tilde{q}_{\psi}, \tag{8.4}$$

where $\tilde{\mathbf{U}}_{\tilde{q}}\ (\Psi^{-1}\mathbf{U})$ indicates the linearization of the reduced order model around the high-fidelity solution $\mathbf{U}$ restricted to the domain of definition of $\tilde{\mathbf{U}}$.

If this approach is effective, one will not need anymore to store or reconstruct $\mathbf{f}$ or even $\mathbf{U}$ during forward iterations.

One interesting example of approximate modelling concerns Hadamard incomplete sensitivity formulation. This is an example of simplification in the continuous level. However, it only concerns specific functionals [25, 27] where:

- the cost function J and control ψ have a same domain of definition (e.g. a shape and an aerodynamic coefficient defined over it),
- J is a product of geometry by state functions $J(\psi) = G(\psi, q(\psi))\ S(\mathbf{U}(q(\psi)))$.

If these requirements hold, we can use an incomplete evaluation of this gradient, neglecting the sensitivity with respect to the state, leading to the approximation $\nabla_\psi \tilde{J} \sim J_\psi + J_\mathbf{U} q_\psi = \nabla_\psi G\ S$. This is very interesting as $\nabla_\psi f$ can often be analytically calculated. And, if not, an approximate model $\tilde{q}$ can be used as described above leading to $\nabla_\psi \tilde{J} = J_\psi + j_\mathbf{U}(\Phi \tilde{q}_\psi)$. Finally, the approach is also interesting because the quantities involved are all locally defined on the domain of definition of ψ and do not involve the full domain of definition of the state variable $\mathbf{U}$. In the case of the shape optimization problem, for instance, there will be no field variable linearized as everything will be defined on the shape.

One can go one step further from incomplete sensitivities introducing low order models such as simple algebraic relations in $\tilde{U}$ [25]. We give examples of such algebraic relations in section 9.2 to address minimal drag sensitivity analysis where the incomplete sensitivity approximation is improved with the introduction of the cosine relation for the macroscopic pressure distribution over a shape and a priori assumptions for the behavior of the velocity in the normal direction to the shape.

9. **Numerical examples**

Now we illustrate our adjoint implementation for the LBM with the D2Q9 stencil presented in appendix A applied to a contour identification and a shape optimization problem. We show in particular how to use reduced order models and incomplete evaluations of sensitivities to break the complexity of adjoint evaluations.

9.1. **Contour recognition**

Non intrusive recognition of a body using partial observations of some macroscopic field is an important domain of application. It is obvious that reducing the computational complexity and improving the speed of the identification is interesting and helps making the necessary device more portable and cheap. We would like to test the pertinence of our adjoint formulations on this problem and see how the adjoint calculations can be accelerated. Also, because the aim is to detect the contours, the choice of the functional is free as far as it helps reducing the calculation complexity.

A typical functional is given by the measure of the deviation of a macroscopic velocity field u from observations u_{obs} and put under the form of (4.3), it reads:

$$J(\psi) = \int_0^T \int_{\Omega_{obs}} j(\psi, u(t,x)), \tag{9.1}$$

where $j(\psi, u) = (u(t,x) - u_{obs}(t,x))^t W^t C W (u(t,x) - u_{obs}(t,x))$, involving the covariance matrix C of the observation. C is diagonal if the observations are independent and with the diagonal element function of the variance of the observations. W is a diagonal weighting matrix which permits to give more importance to some of the data. These matrices can depend on time as well if the quality of the data acquisition is variable in time for instance. Ω_{obs} is the domain of observation which is usually only a subset of the whole calculation domain. This can also be a function of time if the spatial observation

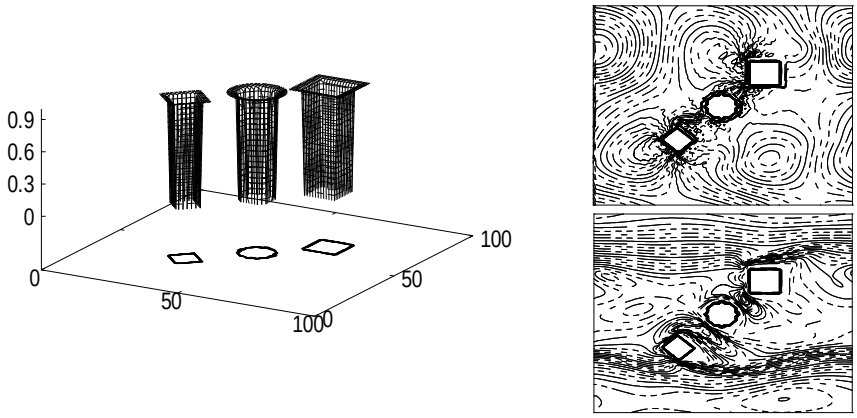

Figure 9.1. On a 100×100 lattice: L_1, L_2 and L_∞ unit balls and χ given by (11.4). On the right: contours for the macroscopic pressure and norm of the the macroscopic velocity after 10000 iterations of the LBM.

window changes with time. This is a very general formulation and we have used it, for instance, for the identification of the effective rigidity of the lithosphere using for u_{obs} the interseismic velocity field from surface global positioning system (GPS) data in time and space [26].

As this is a test problem to evaluate the pertinence of the different adjoint formulations, we would like to avoid any other sources of error and only concentrate on the complexity issues in terms of memory and redundant calculations. We consider $u_{obs}(t \in [0,T], x)$ where T denotes 100 iterations of the LBM solver. We consider a set of simple obstacles given in figure 9.2 on a 100×100 lattice. 100 iterations permits to a given information to cross the domain. The observation domain Ω_{obs} represents 1% of the calculation domain. An indication of this and the iso-contours of $dJ/d\psi$ by a full adjoint calculation with 7 checkpoints are given in figure 9.2. The number of stored state is 7, just above $\log_2(100)$ and is chosen by `tapenade` in order to minimize the redundant calculations to reconstruct missing intermediate states during backward adjoint integration. This is the reference solution.

7 checkpoints is affordable but 100 iterations is a very small compared to what is necessary with the LBM where the number of iterations is usually of several thousands. With 10000 iterations, the number of checkpoints will be 14 which is still acceptable. Therefore, the real difficulty is not a storage question but comes from very penalizing redundant forward calculations to recover intermediate states between the checkpoints. As we said, in `tapenade` the checkpoints are distributed in order to minimize these, but they can also be distributed uniformly or, in order for the reduced order model to fit best the state history, minimizing for instance the interpolation error [5] as in mesh adaptation by metric control [9]. One can measure the error one commits substituting a function of time $f(t)$ by its linear interpolation $\pi_{\delta(t)}f$ constructed from f given on n checkpoints distributed following $\delta(t)$ by $|f - \pi_{\delta(t)}f| \leq c\delta^2|f''|$. The distribution law $\delta(t)$ can be chosen as $\delta(t) = \min(\delta_{max}, \max(\delta_{min}, \epsilon/\sqrt{|f''|}))$ in order to equi-distribute this error. ϵ is an indication of the level of the error and δ_{max} and δ_{min} are cut-off bounds to avoid very large or small values for δ. Hence, n given checkpoints from $T_0 = 0$ can be distributed following $\delta(t)$ by $T_i = T_{i-1} + \delta(T_{i-1})$ such that $T_n = T$. Uniform distribution of the checkpoints can be obviously realized with $\delta = T/n$. As $\mathbf{f}$ is not a scalar function and has 9 components in D2Q9, one must adapt the distribution $\delta(t)$. The simplest way is to replace $|f''|$ by $\max(|f_i''|, i = 1,..,9)$. One

could also introduce different checkpoints for each of the 9 component (but then we need to handle conservation issues).

Hence, to avoid redundant calculations and provide a possible parallel solution we use the ingredients presented in sections 6.4 and 6.5 with in particular the parallel fixed point partial reverse accumulation algorithm with 5 uniformly distributed checkpoints. Figure 9.3 shows four snapshots of the evolution of $dJ/d\psi$ at iterations 5, 10, 15 and 20 of the parallel fixed point partial reverse accumulation to be compared to the full gradient presented in figure 9.2. One sees that unlike with the ordinary differential equation in section 7 here the number of iterations necessary has been much larger than the number of checkpoints. But still, no intermediate states have been stored and no redundant calculations performed. The reduced order model is again a linear interpolation between checkpoints. This could be improved using more sophisticated reduced order models to recover missing informations in the gradient in figure 9.3. One could also adapt the distribution of the checkpoints as suggested above. But these are not central to our discussion.

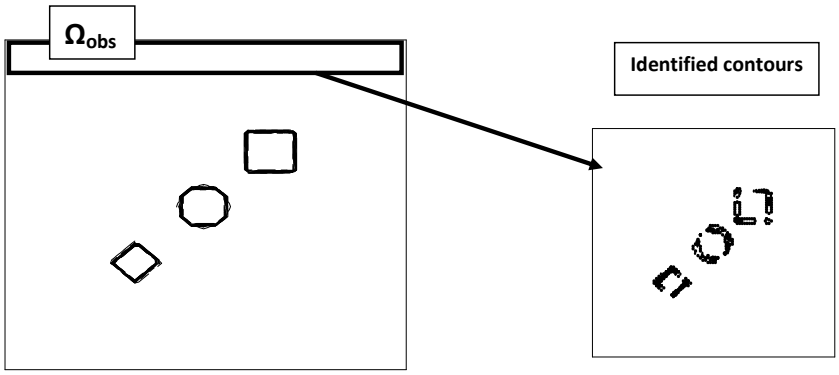

Figure 9.2. Contour identification from partial macroscopic field observation. Left: target shapes and two examples of the observation domain Ω_{obs} each making 1% of the total domain. Right: $dJ/d\psi$ by full adjoint of the LBM for J defined by (9.1) for the two observation domains.

9.2. **Minimal drag design**

We discuss Hadamard incomplete sensitivity and the use of reduced order models for the pressure and viscous drags evaluation through the comparison of these with full adjoint based gradients with respect to shape deformations.

Consider the mean pressure and viscous contributions to the drag coefficient over at time t where the macroscopic quantities come from our LBM solver: $C_d(t) = C_d^p(t) + C_d^v(t)$. Linearization of these functionals permits also to address situations where an integral over time of these quantities is involved such as in a functional of mean drag over time.

Let us first illustrate our purpose with the linearization of the pressure drag coefficient for a shape described by ψ (here scalar for simplicity):

$$C_d^p(t) = \frac{1}{2\rho_\infty \|u_\infty\|^2} \int_{shape(\psi)} p(t, q(\psi))(u_\infty \cdot n(q(\psi)), \tag{9.2}$$

where superscript ∞ indicates inflow conditions.

We are in the validity domain of Hadamard incomplete sensitivities described in section 8 with $S = p$ and $G = n$.

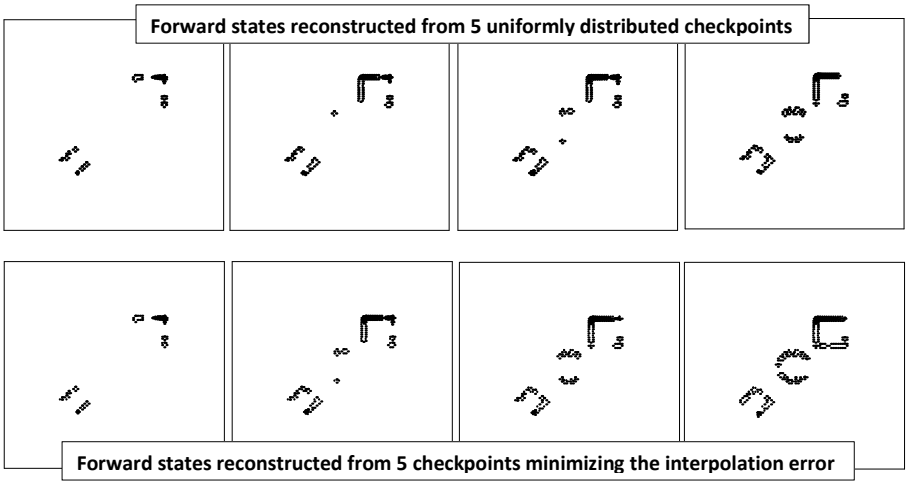

Figure 9.3. Contour identification from partial macroscopic field observation. Four snapshots of $dJ/d\psi$ at iterations 5, 10, 15 and 20 of the parallel fixed point partial reverse accumulation to be compared to the full gradient presented in figure 9.2. The forward states are either reconstructed from 5 uniformly distributed checkpoints (upper) and when the checkpoints are distributed in order to reduce the interpolation error between $\mathbf{f}$ and $\pi_\delta \mathbf{f}$.

Let us analyze the incomplete sensitivity approximation in a situation where the pressure is given analytically from the cosine-square law: $p(\psi) = p_{tot}(u_\infty \cdot n(\psi))^2$ where p_{tot} is the total pressure function of the inflow conditions.

We therefore have $p(\psi)u_\infty \cdot n(\psi) = p_{tot}(u_\infty \cdot n(\psi))^3$. Its derivative with respect to ψ is $(pu_\infty \cdot n)_\psi = (pu_\infty) \cdot n_\psi + p_\psi(u_\infty \cdot n) = 3p_{tot}u_\infty(u_\infty \cdot n)^2 n_\psi$. The first term in the sum is what we called above incomplete sensitivity and is $(pu_\infty)n_\psi = p_{tot}u_\infty(u_\infty \cdot n)^2 n_\psi$.

We see that if the pressure is defined by the cosine-square law, the exact and incomplete derivatives only differ by a factor of 3 and have the same sign.

Now what happens when the pressure comes from the LBM method and is given by $p = 1/3 \sum_i f_i$. The expression above can be rewritten as $pu_\infty \cdot n = p|u_\infty| \cos(\frac{u_\infty}{|u_\infty|} \cdot n)$. The incomplete sensitivity is therefore $p(u_\infty \cdot n)_\psi = -p|u_\infty| \sin(\frac{u_\infty}{|u_\infty|} \cdot n) = 0$ when n is aligned with u_∞. The incomplete sensitivity fails therefore for these area. On the other hand, the model tells us that the pressure sensitivity with respect to shape variations vanishes if those are along the normal to the shape such that $n_\psi = 0$. But this is compatible with the 'macroscopic' pressure boundary condition $p_n = 0$ even if not used in the LBM.

We therefore expect the incomplete sensitivity to be a good approximation of the gradient if the macroscopic pressure verifies the zero normal pressure condition and if the shape deformation parameterization is along the normal to the shape. As in our level set parameterization $n = \nabla\psi/|\nabla\psi|$, this latter means that the variation $\delta\psi$ must be along the normal such that: $|\delta\psi| = \delta\psi \cdot n$.

Now, it is interesting to notice that a descent method can also be interpreted as a Hamilton-Jacobi equation for the motion of the level set in the direction normal to the shape:

$$\psi_t = -\nabla_\psi J = -V\nabla\psi, \ \psi(0) = \text{given},$$

with $V = \frac{\nabla_\psi J}{|\nabla \psi|} \cdot n$. The incomplete sensitivity provides therefore a good approximation for the gradient of the pressure drag. We see that sensitivity evaluation can be dramatically simplified in some situations.

Figure 9.4 shows a comparison of $\nabla_\psi C_d^p(t)$ with the macroscopic pressure distribution obtained from $\mathbf{f}$ after 100 iterations of the LBM solver. It also shows the error one commits with the incomplete sensitivity approximation. The error between the i^{th} component of the gradient of a functional J and its approximation $\tilde{J}$ is estimated through:

$$\eta = \mathrm{sgn}(\nabla_\psi J_i \nabla_\psi \tilde{J}_i)|\nabla_\psi J_i - \nabla_\psi \tilde{J}_i|. \tag{9.3}$$

Two remarks can be made. First that the sign of the incomplete sensitivity is always correct as η is always positive. Second that the level of the error is about three orders of magnitude less than the gradient and is in particular concentrated where the condition $p_n = 0$ might not be well realized (e.g. at corners).

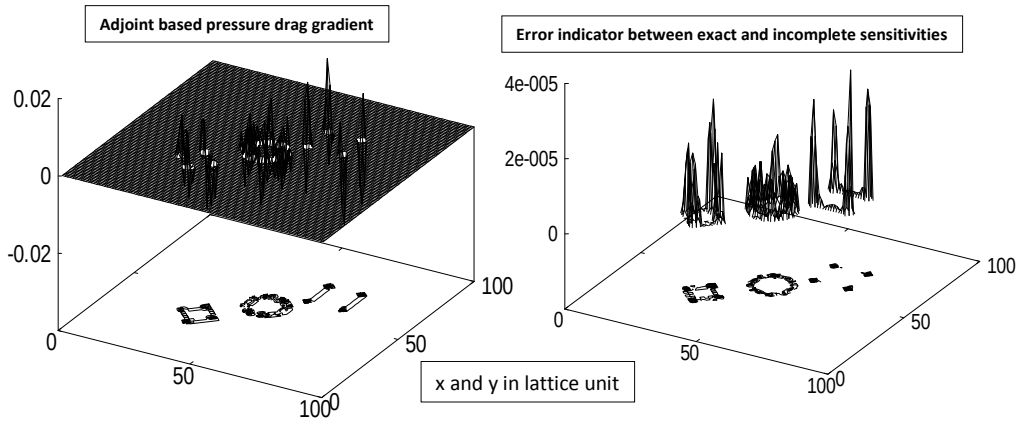

Figure 9.4. Left: pressure drag sensitivity $\nabla_\psi C_d^p(t)$ with full adjoint. Right: Local error between the full and incomplete evaluation of the gradient by indicator η from (9.3).

We discussed above on how to reduce the complexity of sensitivity evaluation for the pressure drag. The other contribution to the drag coefficient is the viscous drag which involves the boundary integral over the shape of the viscous contribution to the Newtonian stress tensor $D(t, q(\psi)) = -\nu(\nabla u(t, q(\psi)) + \nabla u^t(t, q(\psi)))$ involving the gradient of the macroscopic velocity $u = 1/\rho \sum_i c_i f_i$:

$$C_d^v(t) = \frac{1}{2\rho_\infty \|u_\infty\|^2} \int_{shape(\psi)} (D(t, q(\psi)) \cdot n(q(\psi))) \cdot u_\infty. \tag{9.4}$$

Here again we are in the validity domain of Hadamard incomplete sensitivities described in section 8 with $S = D$ and $G = n$. Figure 9.6 shows a comparison of $\nabla_\psi C_d^v(t)$ with the macroscopic velocity distribution obtained from $\mathbf{f}$ after 100 iterations of the LBM solver. It also shows the error one commits with the incomplete sensitivity approximation through indicator (9.3). We see that the incomplete sensitivity prediction does not perform so well in this situation with its sign often incorrect.

One interesting way to go further is to take advantage of the fact that the velocity must satisfy a no-slip boundary condition. This is unlike with the pressure where a Dirichlet boundary condition is

not explicitly prescribed. This a priori available information can be exploited as in Bayesian methods. Let us express the velocity in the domain by:

$$u(t,\psi) = w(\psi)v(t,w(\psi)), \tag{9.5}$$

where w tends to zero with the distance ψ to the shape in order for u to satisfy a homogeneous Dirichlet boundary condition on the shape and v is free and selected in order for wv to satisfy the state equations (for instance with $v = u/w$ where $w \neq 0$). The behavior of w is a priori selected (say linear in ψ for simplicity).

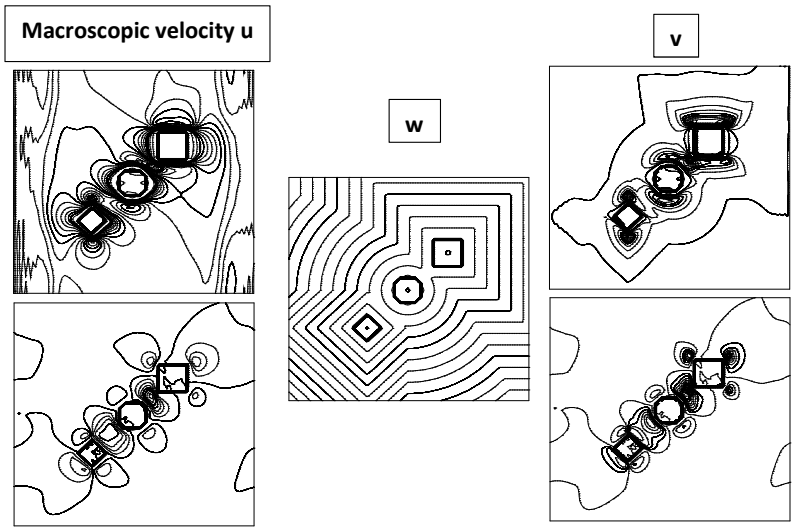

Figure 9.5. A snapshot of the macroscopic velocity u and its decomposition from (9.5).

Now, sensitivity analysis for a functional J (such as the viscous drag $J = C_d^v$) with respect to the shape gives:

$$\nabla_{\psi} J = J_{\psi} + J_q q_{\psi} + J_u(wv_{\psi} + vw_{\psi}),$$

which reduces at $\psi = 0$ to $J_{\psi} + J_q q_{\psi} + J_u v$, because w is chosen such that $w_{\psi} = 1$ and $w(\psi = 0) = 0$. Therefore, in cases where the near-wall dependency of the solution with respect to the distance to the shape can be reasonably guessed the sensitivity with respect to shape variations normal to the wall can be obtained without linearizing the state equation. Figure 9.6 shows an example of such decomposition with a linear dependency in ψ for w.

Figure 9.6 shows how this can improve the incomplete sensitivity prediction. Indeed, introducing this a priori information on the behavior of the velocity normal to the shape, information which is frozen during linearization, the sign of the gradient is now always correct and its amplitude is better predicted. This is therefore a very powerful approach and complete the Hadamard incomplete sensitivity approximation.

These examples are good indications of how reduced order approximations can fully remove the adjoint calculation and drastically reduce the complexity of sensitivity evaluations, especially penalizing in time dependent problems.

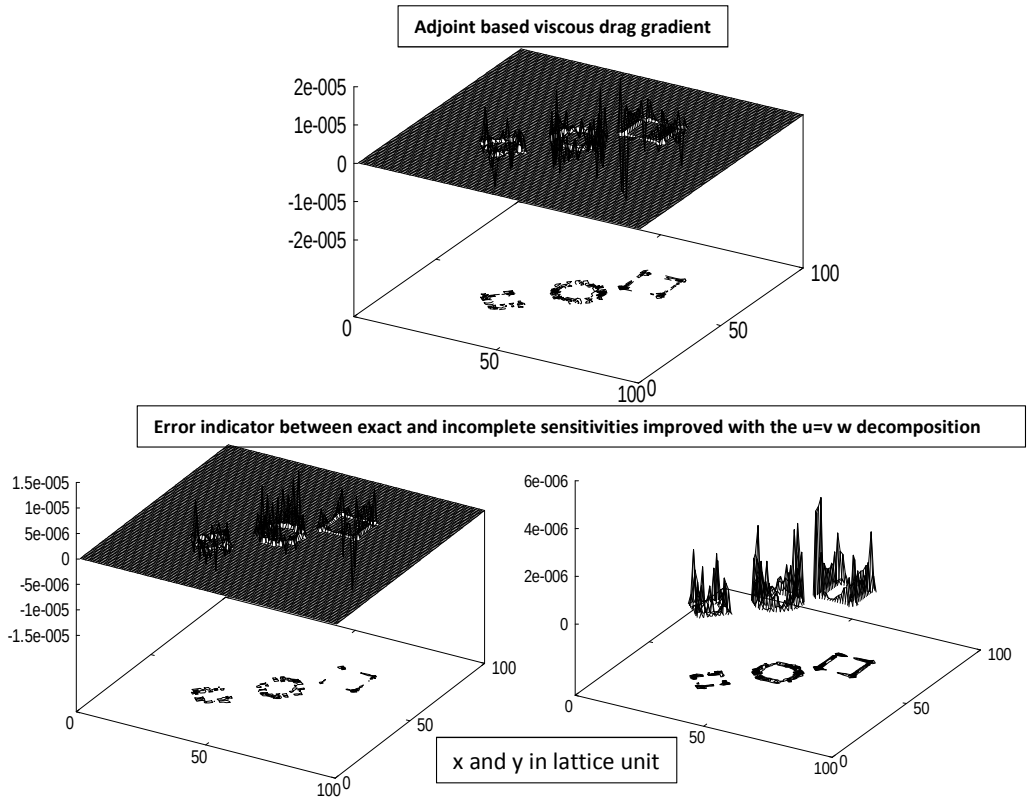

Figure 9.6. Top: viscous drag sensitivity $\nabla_\psi C_d^v(t)$ with full adjoint. Lower-left: Local error between the full and incomplete evaluation of the gradient by indicator η from (9.3). Lower-right: same but taking advantage of decomposition (9.5).

Acknowledgements The discrete adjoint Lattice Boltzmann solver has been obtained using `Tapenade` automatic differentiation tool developed at INRIA-Sophia Antipolis.

10. **Concluding remarks**

The complexity of sensitivity evaluations in adjoint mode has been discussed for time dependent problems. The different ingredients have been illustrated in the context of a lattice Boltzmann solver chosen because challenging in term of the number of variables per node and per time step. The techniques used here to reduce storage complexity in reverse mode of differentiation can be applied to any time marching solver. If an automatized differentiation is targeted, the paper shows the different steps a user should take before the application of automatic differentiation tools which are not necessarily efficient if directly applied to the code.

11. Appendix A: Lattice Boltzmann Method

This section shortly describes the ingredients of Lattice Boltzmann Methods.

11.1. Lattice Boltzmann solver

Lattice Boltzmann Method (LBM) [20] is a smoothed alternative to lattice gas automata. It is capable of solving low speed flow configurations. It mainly gained in popularity because of its algorithmic simplicity making it suitable for parallelization. Our discussion is general and addresses a generic DdQq stencil in d space dimension with q velocities $(c_0, .., c_{q-1})$ on a d-dimensional ($d = 2$ or 3) square or cubic lattice. The notations are classical with $c_0 = 0$ at the lattice center, etc. [17]. The evolution of the distribution function $f_i(x,t)$ is given by the Lattice Boltzmann equation with the BGK collision operator:

$$f_i(x + c_i\Delta t, t + \Delta t) - f_i(x,t) = -\frac{1}{\tau}(f_i - f_i^{eq}),\ i = 0, 1, .., q-1, \tag{11.1}$$

where x denotes the coordinate in the physical space. $\tau = 3\nu + \Delta t/2$ is the relaxation time related to the kinematic velocity ν. $c = h/\Delta t$ is the lattice constant. We unitize h and Δt such that $c = 1$. f_i^{eq} is the equilibrium distribution function given by:

$$f_i^{eq} = \omega_i \rho \left(1 + \frac{c_i \cdot u}{c_s^2} + \frac{1}{2}\frac{(c_i \cdot u)^2}{c_s^4} - \frac{1}{2}\frac{u \cdot u}{c_s^2}\right),\ i = 0, 1, 2, .., q-1,$$

where ω_i are positive weights depending on the stencil chosen. $c_s = \sqrt{3}/3$ is the speed of sound. The macroscopic variables ρ, the fluid density, u, its velocity, and p, the fluid pressure, are related to $f_i,\ i = 0, ..q-1$ through:

$$\rho(x,t) = \sum_i f_i(x,t),\ u(x,t) = \frac{1}{\rho}\sum_i c_i f_i(x,t),\ p = \rho c_s^2 = \rho/3. \tag{11.2}$$

Numerical examples in section 9 are with the D2Q9 stencil. This popular implementation [17] uses nine velocities $(c_0, .., c_8)$ on a two-dimensional square lattice. We use the classical notation where $c_0 = 0$ is at the lattice center and c_1 is toward east, c_2 north, c_3 west, c_4 south, c_5 north-east, etc. $c_i = c(\cos((i-1)\pi/2), \sin((i-1)\pi/2))^t$ for $i = 1, .., 4$ and $c_i = \sqrt{2}c(\cos((2i-9)\pi/4), \sin((2i-9)\pi/4))^t$ for $i = 5, .., 8$. The weights ω_i are $\omega_0 = 4/9$, $\omega_i = 1/9$ for $i = 1, .., 4$ and $\omega_i = 1/36$ for $i = 5, .., 8$.

11.2. Level set parameterization

The level set method, first introduced in [6] and [7] and popularized in [28] is an established technique to represent moving interfaces. Immersed boundary, fictitious domain methods as well as penalizing methods using forcing to account for the presence of curved and moving boundaries in cartesian meshes belong to the same class and have been widely used in different applications [29, 15, 18, 10, 1, 2, 21]. A review of these is presented in [30] together with an application with the LBM.
A level set parameterization is based on the signed Euclidean distance function ψ (other choices of distance are possible) to the boundary Γ which is described by the zero-level curve of ψ:

$$\Gamma = \{x \in \Omega\ :\ \psi(x) = 0\},\ \psi(x) = \pm \inf_{y \in \Gamma} |x - y|,$$

with the convention of a plus sign if $x \in \Omega$ and minus sign otherwise:

$$\psi|_\Gamma = 0,\ \psi|_{\mathbb{R}^d \setminus \Omega} < 0,\ \psi_\Omega > 0. \tag{11.3}$$

For a given shape given by (11.3), the normal to Γ is $n = \nabla\psi/|\nabla\psi|$ at $\psi = 0$, which is useful for Neumann and slip boundary conditions and also for sensitivity evaluation with respect to deformation

normal to the shape as presented in section 9.2.
More precisely, ψ known, we account for the boundary conditions in the state equation using a relaxed normalized distance function $\chi(\psi)$, $(0 \leq \chi(\psi) \leq 1)$. This is necessary when the iso-$(\psi = 0)$ and the lattices do not exactly coincide. More details are given in section 11.3 in the context of the LBM. The relaxation is defined through an explicit regularization expression:

$$\chi = \max(0, \frac{\psi}{|\psi| + \varepsilon h}), \tag{11.4}$$

where $\varepsilon \sim 0.01$ and $h = 1$ in the LBM.

11.3. **Boundary conditions**

We use three types of boundary conditions: no-slip, slip and periodic. The no-slip condition is for obstacles and the slip and periodic conditions for external boundaries.
The slip condition is somehow similar to the no-slip one but with a different condition on the velocities. The two conditions can be summarized as:

$$\text{no-slip condition:} f_i = f_j, \text{ if } c_i = -c_j \text{ and slip condition: } f_i = f_j, \text{ if } c_i \cdot c_j = 0,$$

where f_i and $f_j, i, j = 1, .., q-1$ are two densities in an element of the lattice.
To account for the presence of walls defined by the level set function one needs to identify the nodes where no-slip boundary condition must be enforced. To make this decision one applies the mask defined by χ on the lattice. An example of this is shown in figure 9.1 with periodic boundary conditions on the external boundaries.
When $\chi = 0$, this is clearly an obstacle node and the fluid is bounced back from those playing the role of the no-slip boundary condition. The velocity vector of all fluid densities is inverted, so all the fluid densities are sent back to the node where they were located before the last propagation step (see section 11.4), but with opposite velocity vector. For nodes in the buffer zone where $0 < \chi < 1$ we proceed with a linear interpolation following the value of χ between the propagated density $(\chi = 1)$ and the bounced back value $(\chi = 0)$. This representation is similar to the porous boundary representation in [30]. This formulation can be improved but this is not central to our discussion. Such constructions are necessary when the obstacle does not exactly match the nodes of the lattice.
The periodic condition is simply implemented linking the north and south nodes of two facing elements of the lattice in the top and bottom boundaries and the east and west nodes of two facing elements in the right and left boundaries.
The flow direction is enforced by density redistribution in the first lattice column: the west nodes densities are reduced by the suitable D2Q9 ratio and the east ones increased by the same. This is done as far as positivity can be ensured.

11.4. **Some implementation details**

At each time step the solver achieves the following tasks which can be taken in different orders that presented here:

- collision step with relaxation parameter $1/\tau$ applied from equation (11.1):

$$\tilde{f}_i(x,t) = f_i(x,t) - \frac{1}{\tau}(f_i(x,t) - f_i^{eq}(x,t)),\ i = 0, 1, .., q-1,$$

  where ρ and u in $f_i^{eq}(x,t)$ are evaluated from (11.2) using $f_i(x,t)$.

- density propagation where all fluid densities are propagated from free nodes (non body) along the lattice connection lines to their next neighbors made available by the periodic boundary conditions on the external boundaries:

$$K\ (\chi f_i(x+c_i,t+1)+(1-\chi)f_i(x-c_i,t+1))=\tilde{f}_i(x,t),\ i=0,1,..,q-1. \tag{11.5}$$

This step takes into account the no-slip boundary condition on obstacles as described in section 11.3 through a linear interpolation in the buffer area. K is a scaling factor to enforce local density conservation and is given by the ratio of the sum of the densities over $q-1$ nodes of an element of the lattice (excluding the central node) before and after the interpolation.

12. Appendix B: Principles of Automatic Differentiation by examples

We would like to give a brief description of automatic or algorithmic differentiation methods which permit to compute derivatives in discret level from a computer code linking the independent variables to the functional.
Consider the problem of finding $j'(u)$ when $j(u)$ is given by a computer program.

12.1. The direct mode of AD

Because the program is made of differentiable lines, j' can be computed by differentiating every line and adding them to the computer program immediately above each line. For instance,

Program for j .	**Lines to add**
$x=(1+u)*\log(u)$	$dx=(1+u)*du/u+\log(u)$
$z=x+\cos(u)$	$dz=dx-\sin(u)*du$
$j=x*z$	$dj=dx*z+x*dz.$

If this new program is run with `u=u0, du=1, dx=0, dz=0, dj=0`, then dj is the derivative of j with respect to u at $u0$. This is called the direct mode of AD.

12.2. The reverse mode of AD

The reverse mode of AD is similar to the continuous adjoint method presented in section 4.1 and aims to provide the gradient with a cost independent of the number of variables in the program. Let us interpret this mode introducing the Lagrangian of the code above by associating to each variable in the program a dual variable p, except for the last line for which $p=1$ (each line of a computer code is seen as an equality constraint and the final line as the cost function):

$$L=p_1[x-(1+u)\log(u)]+p_2[z-x-\cos(u)]+j-xz \tag{12.1}$$

Stationarity with respect to intermediate variables in reverse order (z,x) gives

$$\begin{aligned}\frac{\partial L}{\partial z} &= 0=x+p_2\\ \frac{\partial L}{\partial x} &= 0=z-p_2\cos(u)+p_1.\end{aligned}$$

This gives p_2 first, and then p_1, and then dj/du is

$$j'=\frac{\partial L}{\partial u}=p_2x\sin(u)-p_1(\log(u)+\frac{1+u}{u}).$$

This is different from the direct mode in term of complexity because whatever the number of independent variables, the adjoint variables p_i are evaluated only once. A powerful technique to avoid the Lagrange method is to use reverse accumulation and this is how `Tapenade` works as presented in section 5 for our LBM code. More precisely, for each assignment $y = y + f(x)$, the dual expression is $p_x = p_x + f'p_y$ with p_x and p_y the dual variables associated to x and y. Hence, for $(p_x = 0, p_y = 1)$ as initialization, this gives $p_x = f'$. The previous example becomes

$$p_x = p_z = p_u = 0,\ p_j = 1,$$
$$p_x = p_x + zp_j,\ p_z = p_z + xp_j,$$
$$p_x = p_x + p_z,\ p_u = p_u - \sin(u)p_z,$$
$$j' = p_u + (\log(u) + \frac{1+u}{u})p_x.$$

This approach can be used to directly write 'by hand' the adjoint code. This can also be seen as an alternative to deriving the continuous adjoint and programming it.

References

[1] D.M. Anderson, G.B. McFadden, and A. Wheeler. Diffuse-interface methods in fluid mechanics. *Annu. Rev. Fluid Mech.*, 30:139–165, 1998.

[2] P. Angot, C.-H. Bruneau, and P. Fabrie. A penalization method to take into account obstacles in viscous flows. *Numerische Mathematik*, 81:497–520, 1999.

[3] H.-G. Bock and K.J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the 9th IFAC World Congress*, pages 22–31. Budapest Univ., 1984.

[4] B. Christianson. Reverse accumulation and implicit functions. *Optimization Methods and Software*, 9:307–322, 1998.

[5] P.G. Ciarlet. *The finite element method for elliptic problems*. North-Holland, Amsterdam, 1978.

[6] A. Dervieux and F. Thomasset. A finite element method for the simulation of Rayleigh-Taylor instability. *Lecture Notes in Mathematics*, 771:145–159, 1979.

[7] A. Dervieux and F. Thomasset. Multifuid incompressible fows by a finite element method. *Lecture Notes in Physics*, 11:158–163, 1991.

[8] M. Gander and S. Vandewalle. Analysis of the parareal time-parallel time integration method. *SIAM J. Sci. Comput.*, 29:556–578, 2007.

[9] P.-L. George. *Automatic mesh generation. Applications to finite element method*. Wiley, London, 1991.

[10] R. Glowinski, T.W. Pan, and J. Periaux. A fictitious domain method for external incompressible viscous flows modeled by Navier-Stokes equations. *Comput. Meth. Appl. Mech. Eng.*, 112:133–148, 1994.

[11] A. Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1:35–54, 1992.

[12] A. Griewank. *Computational derivatives*. Springer, New York, 2001.

[13] L. Hascoet and M. Araya-Polo. Enabling user-driven checkpointing strategies in reverse mode AD. In Wesseling, editor, *ECCOMAS CFD conference*, pages 153–162. Springer, 2006.

[14] L. Hascoet and V. Pascual. Tapenade user's guide. In *INRIA Technical report*, pages 1–31. INRIA, 2004.

[15] J. Kim, D. Kim, and H. Choi. An immersed-boundary finite-volume method for simulations of flow in complex geometries. *J. Comput. Phys.*, 171:132–150, 2001.

[16] M.J. Krause. *Fluid flow simulation and optimisation with lattice Boltzmann methods on high performance computers*. PhD thesis, Karlsruhe Institute of Technology, 2010.

[17] P. Lallemand and L. Li-Shi. Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability. In *NASA/CR-2000-210103, ICASE Report No. 2000-17,*, pages 1–45. ICASE, 2000.

[18] R.J. Leveque and Z. Li. The immersed interface method for elliptic equations with discontinuous coefficients and singular sources. *SIAM J. Num. Anal.*, 31:1001–1025, 1994.

[19] J.-L. Lions, Y. Maday, and G. Turinici. A parareal in time discretization of PDE's. *C.R. Acad. Sci. Paris, Serie I.*, 332:1–8, 2001.

[20] G. McNamara and G. Zanetti. Use of the Boltzmann equation to simulate lattice-gas automata. *Phys. Rev. Letters*, 61:2332–2335, 1988.

[21] B. Mohammadi. Global optimization, level set dynamics, incomplete sensitivity and regularity control. *Int. J. Comp. Fluid. Dynamics*, 21:61–68, 2008.

[22] B. Mohammadi. Reduced sampling and incomplete sensitivity for low-complexity robust parametric optimization. *Int. J. Num. Meth. Fluids*, 73:307–323, 2013.

[23] B. Mohammadi. Uncertainty quantification by geometric characterization of sensitivity spaces. *Compt. Meth. Appl. Mech. Eng.*, 280:197–221, 2014.

[24] B. Mohammadi. Value at risk for confidence level quantifications in robust engineering optimization. *Optimal Control: Applications and Methods*, 35:179–190, 2014.

[25] B. Mohammadi and O.Pironneau. *Applied shape optimization for fluids (2nd Edition)*. Oxford Univ. Press, Oxford, 2009.

[26] B. Mohammadi, M. Peyret, J. Chery, and C. Joulain. Plate rigidity inversion in southern California using interseismic GPS velocity field. *Geophys. J. Int.*, 187:783–796, 2011.

[27] B. Mohammadi and O. Pironneau. Shape optimization in fluid mechanics. *Annu. Rev. of Fluid Mech.*, 36:255–279, 2004.

[28] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79:12–49, 1998.

[29] C.S. Peskin. The fluid dynamics of heart valves: experimental, theoretical and computational methods. *Annu. Rev. Fluid Mech.*, 14:235–259, 1981.

[30] G. Pingen, M. Waidmann, A. Evgrafov, and K. Maute. A parametric level-set approach for topology optimization of flow domains. *Struct. Multidisc. Optim.*, 41:117–131, 2010.

[31] O. Pironneau. On optimal shapes for Stokes flow. *J. Fluid Mech.*, 70:331–340, 1973.

[32] O. Pironneau. *Optimal shape design for elliptic systems*. Springer, Berlin, 1984.

[33] N. Rostaing. Direct and revers modes of AD for inverse problems. In *SIAM workshop on computational differentiation*, pages 253–282. SIAM, 1996.

[34] I.M. Sobol. Sensitivity estimates for nonlinear mathematical models. *Mathematical Modelling and Computational Experiments*, 1:407–414, 1993.

[35] I.M. Sobol and S. Kucherenko. Derivative based global sensitivity measures and their link with global sensitivity indices. *Mathematics and Computers in Simulation*, 79:3009–3017, 2009.

[36] M. Tekitek, M. Bouzidi, F. Dubois, and P. Lallemand. Adjoint lattice Boltzmann equation for parameter identification. *Computers and Fluids*, 35:805–813, 2006.