Stardis: Propagator evaluation for coupled heat transfer in large geometric models

Léa Penazzi, Stéphane Blanco, Cyril Caliot, C Coustet, Mouna El-Hafi, Richard A Fournier, Jacques Gautrais, Morgan Sans

Stardis: Propagator evaluation for coupled heat transfer in large geometric models

L. Penazzi[a,b,*], S. Blanco[c], C. Caliot[d], C. Coustet[e], M. El Hafi[a], R. Fournier[c], J. Gautrais[f], M. Sans[a]

[a]*RAPSODEE, UMR CNRS 5302, IMT Mines Albi, Campus Jarlard, Albi, France*
[b]*PROMES-CNRS, UPR 851, 7 rue du Four Solaire, 66120 Font Romeu Odeillo, France*
[c]*LAPLACE, UMR CNRS 5213, Université Paul Sabatier, 118 Route de Narbonne - 31062 Toulouse, France*
[d]*LMAP, UMR CNRS 5142, Université de Pau et des Pays de l'Adour, Avenue de l'Université, 64013 Pau, France*
[e]*Méso-Star - 8 rue des Pêchers, 31410 Longages, France*
[f]*Centre de Recherches sur la Cognition Animale (CRCA), Centre de Biologie Intégrative (CBI), Université de Toulouse, CNRS, Université Paul Sabatier - Toulouse, France*

Abstract

When heat transfer is linear or can be linearized around a given temperature field, a common engineering practice is to evaluate a propagator for each of the energy sources as well as of the initial temperature field (regarded itself as a source) so that the temperature at a given location and at a later time (a probe point) can be expressed as a simple sum of all sources at all previous times, multiplied by the value of the propagator towards the probe. These simple expressions are then immediately usable for all types of optimization, inversion, sensitivity analysis and command control objectives. However, when facing large CAD geometry models or large numbers of spatially distributed sources, evaluating the relevant propagator values with standard deterministic meshed methods can be a quite difficult or computationally expensive numerical task. On the contrary, Monte Carlo is well known for directly addressing these propagator values as well as for handling large geometrical models with ease. Until quite recently, very few Monte Carlo algorithms could be implemented for coupled transient heat transfer. Recent work on Green formulation and stochastic processes have led to the definition of conducto-convecto-radiative paths and they can now be sampled efficiently using advanced computer graphic tools in order to evaluate the temperature at a probe point, whatever the level of geometrical refinement is. The *Stardis* project holds an implementation of such recent advances, computing how sources propagate towards a probe point in space-time. A remarkable feature is that *Stardis* output propagation data themselves, which in turn can be repeatedly used for fast estimation over lots of sets of source values. We detail here this feature of the code and provide all the theoretical background required by a thermal scientist facing the need to use and adapt *Stardis*.

Keywords: Propagator, Monte Carlo method, Large geometric models, Coupled heat transfer

PROGRAM SUMMARY

Program Title: Stardis 0.7.2 (built on `stardis-solver` 0.12.3)
CPC Library link to program files: (to be added by Technical Editor)
Developer's repository link: `https://www.meso-star.com/projects/stardis/stardis.html`
Code Ocean capsule: (to be added by Technical Editor)
Licensing provisions: GPLv3
Programming language: ANSI C
Supplementary material: (Zip folder added to submission)
Nature of problem: Estimating a probe temperature at given time and location in a coupled heat transfer system involving large CAD and/or large numbers of spatially distributed sources.
Solution method: Stardis uses Monte Carlo Method in order to solve coupled thermal systems for cases where radiative transfer can be linearized. It evaluates the probe temperature as a propagator from each energy source in the system: the initial temperature, temperature boundary condition, volume power and surface heat flux.

*Corresponding author.
E-mail address: lea.penazzi@mines-albi.fr

Additional comments including restrictions and unusual features: Stardis estimate of the propagator is reliable when linearization of radiative transfer is relevant. For linearization, a temperature of reference has to be chosen in accord with the system under consideration.

1. Introduction

Stardis is a Monte Carlo code for estimating propagation data in the context of linear heat transfer.

Starting from G. Green's theory, the propagator concept was introduced by R. Feynmann as a way to picture, in integral terms, the solution $O(\vec{x},t)$ of a field physics problem at a location $\vec{x}$ and time t, when this physics is linear : $O(\vec{x},t)$ is viewed as an integral over all sources $S(\vec{x}_S,t_S)$ at all locations $\vec{x}$ inside the domain $\mathcal{D}$ and all times preceding t (down to initial time t_I), multiplied by a scalar $\zeta(\vec{x},t,\vec{x}_S,t_S)$:

$$O(\vec{x},t) = \int_{\mathcal{D}} d\vec{x}_S \int_{t_I}^{t} dt_S \; \zeta(\vec{x},t,\vec{x}_S,t_S)\, S(\vec{x}_S,t_S) \tag{1}$$

The propagator $\zeta(\vec{x},t,\vec{x}_S,t_S)$ indicates how each source impacts the solution, and invites for intuitions of the sources being propagated in space and time throughout the system, toward the considered location $\vec{x}$ and time t. Historically, this rewriting of G. Green's formalism is mainly significant with regards to the physical pictures it suggests. Here, we will concentrate on translating these pictures in pure computational terms : since $\zeta(\vec{x},t,\vec{x}_S,t_S)$ is independent of the source *values*, it can be numerically evaluated on its own. Then, any set of sources values can be plugged into Eq. (1) to compute the corresponding $O(\vec{x},t)$.

Let us illustrate this concept with a standard practice in radiative transfer, where the factors associated with the propagative point of view are named "shape factors" or "exchange surfaces" (depending on the context and the chosen formulations). Let us take a simple scene with stationary radiative transfer between a camera and two lamps of respective powers $\mathcal{P}_1$ and $\mathcal{P}_2$. Let O denote the radiative flux incident on a chosen pixel of the camera.

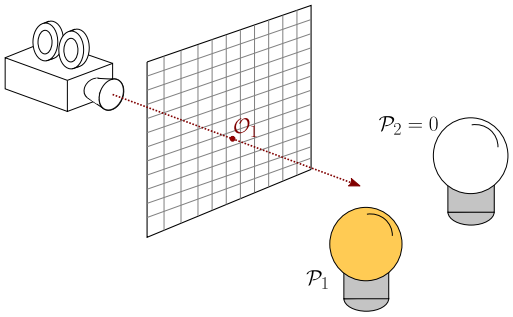

Figure 1: $\mathcal{P}_1$ light source on and $\mathcal{P}_2$ off

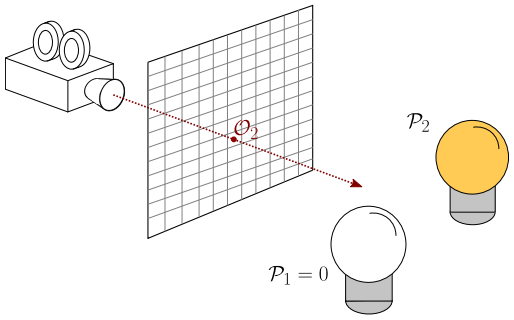

Figure 2: $\mathcal{P}_2$ light source on and $\mathcal{P}_1$ off

Evaluating the shape factor ξ_1 between the first lamp and the target pixel can be obtained by switching on the first lamp alone and solving the radiative transfer equation to get the corresponding pixel flux O_1 (see Fig. 1). The shape factor associated with this first lamp is then $\xi_1 = \frac{O_1}{\mathcal{P}_1}$. The same radiative transfer equation can also be solved to evaluate the pixel flux O_2 when only the second lamp is on, and the shape factor of the second lamp is $\xi_2 = \frac{O_2}{\mathcal{P}_2}$ (see Fig. 2). Once these factors are known, since they are independent of the powers of the lamps, they can be used to evaluate the pixel flux $\tilde{O}$ for any other set of lamp powers $\tilde{\mathcal{P}}_1$ and $\tilde{\mathcal{P}}_2$ when the two lamps are on simultaneously:

$$\tilde{O} = \xi_1\tilde{\mathcal{P}}_1 + \xi_2\tilde{\mathcal{P}}_2 \tag{2}$$

The required number of shape factors can be huge, for example in the domain of infrared radiation where all surface and volume parts of the system can emit radiation, so the question of the efficient numerical evaluation of large numbers of shape factors has raised numerous technical questions, in particular for multiple scattering and multiple

reflection configurations with semi-transparent materials. A classical approach is to use reverse Monte Carlo algorithms in which optical paths are tracked from the receptor (the target pixel in our example) backward to sources (the lamps). The procedure can be duplicated for each target of interest (e.g. the other pixels of the camera).

Such algorithms, initially designed to estimate O, can also be used to estimate each shape factor (from each lamp to the target pixel), with no additional computational cost : the paths sampled in one single computation can yield estimates of O as well as all the shape factors. In our example, the reverse Monte Carlo computation can then be made only once for a given set of the two lamps power, and does not need to be repeated for each lamp one by one. Once evaluated, the shape factors can be linearly combined with any sets of lamp powers. In this practice, there is one point of concern : we need to consider statistical correlations when evaluating the estimate uncertainty ("error bars") because the estimates are built using the same set of sampled optical paths over different sets of lamps power. Taking this point into account, the approach is then straightforward : only the shape factors required to estimate O are to be computed using a reverse Monte Carlo.

The Monte Carlo code of *Stardis* is based on such a reverse approach for estimating $\zeta(\vec{x}, t, \vec{x}_S, t_S)$ in the context of linear heat transfer. It samples thermal paths throughout the system, from the target $(\vec{x}, t)$, backward in time to its thermal sources. In the context of heat transfer, this implies a comprehensive definition of *sources*, in order to encompass all the information that may impact the solution : this will include the expected standard power sources, like imposed volume or surface power densities, but also all imposed boundary temperatures as well as the initial temperature field.

It is only quite recently that Monte Carlo can address the transient coupling of conduction, convection and radiation in geometric models with a complexity typical of industrial CAD, typically to address how the temperature of a junction inside an electronic device depends on the history of imposed fluctuating electrical currents or fluctuating radiator temperatures... We show here that such advances offer the same benefits in terms of $\zeta(\vec{x}, t, \vec{x}_S, t_S)$ computation as those experimented in radiative transfer. For the point of interest $(\vec{x}, t)$, the code can be run once for a given description of thermal sources (volume power, heat flux, imposed temperatures at the boundary, initial conditions) and all relevant propagation data can be stored.

One of the strongest significance of storing propagation data is that, then, basic algebraic functions can be coded to evaluate the temperature at $(\vec{x}, t)$ for any set of values for thermal sources, namely sums of sources values multiplied by the corresponding $\zeta(\vec{x}, t, \vec{x}_S, t_S)$. That would yield very fast estimations. Avoiding the cost of computing full heat transfer for each set, these fast functions can then be used as part of standard design optimisation or command algorithms, where estimations are needed over lots of varying sets of sources values.

The present article describes how the *Stardis* project implements these ideas:

- Sec. 2 describes the physical model for coupled heat transfer.
- Sec. 3 describes how a reverse Monte Carlo path sampling can be used to solve this model.
- Sec. 4 describes how *Stardis* stores the propagation data.
- Sec. 5 describes `stardis-solver`, an implementation of reference used by *Stardis*.
- Sec. 6 depicts simulation examples.
- Sec. 7 gives some hints towards a generalisation to non-uniform and time-dependent sources, before concluding remarks in Sec. 8.

2. Model for coupled heat transfer

2.1. System description

The system is delimited with a system-boundary surface $\mathcal{S}$ that is split into $N_{\mathcal{S}}$ sub-surfaces $\mathcal{S}_i$. The internal volume Ω is split into N_Ω sub-volumes Ω_i of boundaries $\partial\Omega_i$ (see Fig. 3).

Each sub-volume is either a uniform opaque solid or a perfectly mixed transparent fluid. The contact between adjacent solid sub-volumes is perfect (the thermal contact resistance is null) and the boundary layers at solid-fluid interfaces are not described explicitly (they are summarized by a convective exchange coefficients). The thermal

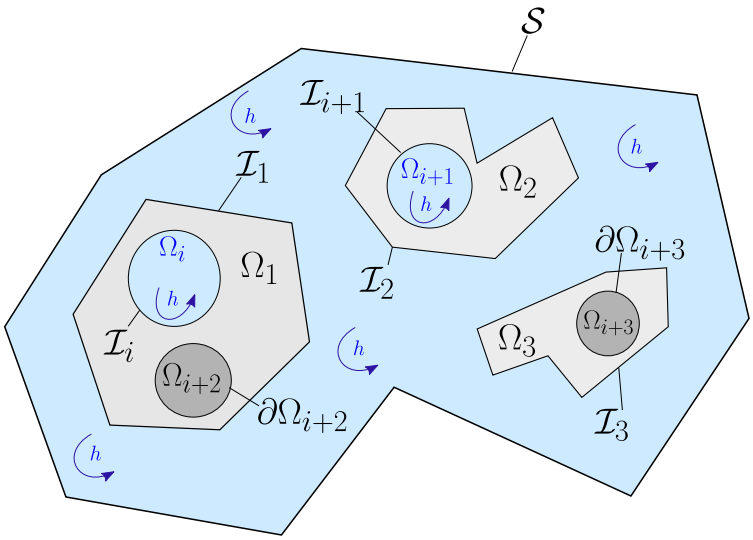

Figure 3: Sketch of the general configuration. The system is delimited with a $\mathcal{S}$ boundary, three internal volumes Ω_1, Ω_2 and Ω_3 are represented. In Ω_1 solid medium, there is a fluid medium Ω_i and a sub-solid volume corresponding to Ω_{i+2}. $\mathcal{I}_i$ is the solid/fluid boundary between Ω_1 and Ω_i. $\partial\Omega_{i+2}$ is the solid boundary between Ω_{i+2} and Ω_1. Similarly, Ω_2 contains a fluid sub-volume Ω_{i+1} and $\mathcal{I}_{i+1}$ is the solid/fluid boundary. The last volume Ω_3 contains a solid volume Ω_{i+3} and the solid boundary is $\partial\Omega_{i+3}$. Conduction occurs in the solid volumes, radiation and convection occur in the different fluid volumes. h represents the convective heat transfer coefficient.

properties of a solid sub-volume Ω_i are the thermal conductivity λ_i, the mass density ρ_i and the mass thermal capacity c_i. For a fluid sub-volume, the required thermal properties are only ρ_i, c_i and the fluid volume V_i. A power density ψ_i can also be prescribed inside each solid sub-volume. There cannot be two fluid sub-volumes adjacent to each other: a fluid sub-volume is always a fluid cell enclosed by solids.

The ensemble of all solid-fluid interfaces (between adjacent sub-volumes of different types) is noted $\mathcal{I}$. It is split into $N_{\mathcal{I}}$ sub-interfaces $\mathcal{I}_i$. The surface properties are uniform along each sub-interface: the convective exchange coefficient is noted h_i; the surface of the solid is grey of emissivity ϵ_i and reflection is modeled using a fraction α_i of specular reflection and a fraction $1-\alpha_i$ of diffuse reflection.

On each sub-surface $\mathcal{S}_i$, the boundary condition can be of the following types:

- *type-1* - $\mathcal{S}_i$ is along a solid sub-volume and the solid temperature is known at this boundary, noted $T_{\mathrm{B},i}$.
- *type-2* - $\mathcal{S}_i$ is along a solid sub-volume and the boundary flux density is known, noted $\varphi_{\mathrm{B},i}$.
- *type-3* - $\mathcal{S}_i$ is along a solid sub-volume, a transparent fluid is facing it, and the fluid temperature is known, noted $T_{\mathrm{BF},i}$. The boundary flux density is then the sum of the convective flux density and the radiative flux density, with uniform values of the convective exchange coefficient h_i, the emissivity ϵ_i and the specular/diffuse ratio α_i. At such a boundary, for incident directions that come from outside the system, the radiance temperature is known, noted θ_{BR}.
- *type-4* - $\mathcal{S}_i$ is at the limit of a fluid sub-volume and the limit temperature is known, noted $T_{\mathrm{B},i}$. This temperature is to be interpreted as that of a solid surface enclosing the fluid cell, with uniform values of the convective exchange coefficient h_i, the emissivity ϵ_i and the specular/diffuse ratio α_i.

2.2. Radiation

As the solids are opaque, the fluids are transparent and photon transport is instantaneous, radiative heat transfer can be summarized to instantaneous exchanges between solid surfaces. At a location $\vec{y}$ at the surface of a solid sub-volume $\mathcal{D}_i$ facing a fluid, the radiative flux density $\varphi_R(\vec{y},t)$ is the difference between absorption of radiation in all incident directions $\vec{\omega}$ and emission by the solid due to its local temperature $T_i(\vec{y},t)$:

$$\varphi_R(\vec{y},t) = -\epsilon_i\left(\sigma T_i(\vec{y},t)^4 - \int_{\mathcal{H}_i(\vec{y})} |\vec{\omega}.\vec{n}_i(\vec{y})|\, I(\vec{\omega},\vec{y},t)\, d\vec{\omega}\right) \tag{3}$$

where $I(\vec{\omega}, \vec{y}, t)$ is the spectrally integrated intensity at $\vec{y}$ in direction $\vec{\omega}$, σ is the Stefan-Boltzmann constant, $\vec{n}_i(\vec{y})$ is the unit normal to the solid at $\vec{y}$ and $\mathcal{H}_i(\vec{y})$ is the hemisphere of all incident directions at $\vec{y}$.

It is assumed that radiative transfer can be linearized with respect to the temperature around a given reference temperature T_{ref}, which means that $T_i^4 \approx T_{\text{ref}}^4 + 4T_{\text{ref}}^3(T_i - T_{\text{ref}})$ leading to the expression $h_R = 4\epsilon_i\sigma T_{\text{ref}}^3$. We then make the choice of translating the spectrally integrated intensity into a radiance temperature $\theta_R = \int_{\mathcal{D}_\Gamma} p_\gamma T(\vec{x}_\gamma) d\gamma$, i.e. a mean radiative temperature seen at the solid/fluid interface due to radiative exchanges through the fluid phase.

Observing that $\int_{\mathcal{H}_i(\vec{y})} \frac{|\vec{\omega}.\vec{n}_i(\vec{y})|}{\pi} d\vec{\omega} = 1$, Eq. (3) becomes

$$\varphi_R(\vec{y}, t) = -h_R \left(T_i(\vec{y}, t) - \int_{\mathcal{H}_i(\vec{y})} \frac{|\vec{\omega}.\vec{n}_i(\vec{y})|}{\pi} \theta_R(\vec{\omega}, \vec{y}, t)\, d\vec{\omega} \right) \tag{4}$$

We note $\vec{z} \equiv \vec{z}(\vec{y}, -\vec{\omega})$ the location of first intersection with a solid sub-volume Ω_j of a straight line starting from $\vec{y}$ in direction $-\vec{\omega}$. If there is no intersection ($\vec{z}$ at infinity), then $\theta_R(\vec{\omega}, \vec{y}, t)$ equals the incident radiance $\theta_{\text{BR}}(\vec{\omega}, \vec{y}, t)$ known at the system boundary. Otherwise, $\theta_R(\vec{\omega}, \vec{y}, t) = \theta_R(\vec{\omega}, \vec{z}, t)$ (pure transport) and $\theta_R(\vec{\omega}, \vec{z}, t)$ is modeled as the sum of the emission by the solid at temperature $T_j(\vec{z}, t)$, the specular reflection of incoming radiation in direction $-\vec{\omega}_S$ where $-\vec{\omega}_S$ is the symmetric of $\vec{\omega}$ around $\vec{n}_j(\vec{z})$, and the diffuse reflection of radiation incident in all the directions $\vec{\omega}'$ of the incident hemisphere $\mathcal{H}_j(\vec{z})$ at $\vec{z}$. Altogether,

$$\begin{cases} \text{If } \vec{z} \text{ at } \infty: & \theta_R(\vec{\omega}, \vec{y}, t) = \theta_{\text{BR}}(\vec{\omega}, \vec{y}, t) \\ \text{If } \vec{z} \in \partial\mathcal{D}_j: & \theta_R(\vec{\omega}, \vec{y}, t) = \epsilon_j T_j(\vec{z}, t) + (1 - \epsilon_j)\alpha_j \theta_R(-\vec{\omega}_S, \vec{z}, t) \\ & \qquad + (1 - \epsilon_j)(1 - \alpha_j) \displaystyle\int_{\mathcal{H}_j(\vec{z})} \frac{|\vec{\omega}'.\vec{n}_j(\vec{z})|}{\pi} \theta_R(\vec{\omega}', \vec{z}, t)\, d\vec{\omega}' \end{cases} \tag{5}$$

2.3. Conduction

At any location $\vec{x}$ inside a solid sub-volume $\mathcal{D}_i$, at any time t, the solid temperature $T_i \equiv T_i(\vec{x}, t)$ is solution of the following heat equation,

$$\rho_i c_i \frac{\partial T_i}{\partial t} = \lambda_i \Delta T_i + \psi_i \tag{6}$$

where $\psi_i \equiv \psi_i(\vec{x}, t)$ is the local value of the power density. The initial condition at time t_{I} is

$$T_i(\vec{x}, t_{\text{I}}) = T_{\text{I},i}(\vec{x}) \tag{7}$$

At any location $\vec{y}$ at the limit of $\mathcal{D}_i$ (i.e. $\vec{y} \in \partial\mathcal{D}_i$), at any time t, the modeling of the interface or the boundary condition is one of the following :

- If $\vec{y}$ is at an interface with another solid sub-volume Ω_j,

$$\lambda_i \vec{\nabla} T_i.\vec{n_i} = \lambda_j \vec{\nabla} T_j.\vec{n_i} \tag{8}$$

- If $\vec{y}$ is at an interface $\mathcal{I}_k$ with a fluid sub-volume $\mathcal{D}_j$ (with $h_R = 4\epsilon_k\sigma T_{\text{ref}}^3$),

$$-\lambda_i \vec{\nabla} T_i.\vec{n_i} = h_k(T_j - T_i) - h_R \left(T_i - \int_{\mathcal{H}_i(\vec{y})} \frac{|\vec{\omega}.\vec{n}_i(\vec{y})|}{\pi} \theta_R(\vec{\omega}, \vec{y}, t)\, d\vec{\omega} \right) \tag{9}$$

- If $\vec{y}$ is at the boundary of the system, in a sub-surface $\mathcal{S}_j$ with a *type-1* boundary condition,

$$T_i = T_{\text{B},j} \tag{10}$$

- If $\vec{y}$ is at the boundary of the system, in a sub-surface $\mathcal{S}_j$ with a *type-2* boundary condition,

$$-\lambda_i \vec{\nabla} T_i.\vec{n_i} = \varphi_{\text{B},j} \tag{11}$$

- If $\vec{y}$ is at the boundary of the system, in a sub-surface $\mathcal{S}_j$ with a *type-3* boundary condition (with $h_R = 4\epsilon_j\sigma T_{\text{ref}}^3$),

$$\begin{aligned} &-\lambda_i \vec{\nabla} T_i.\vec{n_i} = \\ &h_j(T_{\text{BF},j} - T_i) - h_R \left(T_i - \int_{\mathcal{H}_i(\vec{y})} \frac{|\vec{\omega}.\vec{n}_i(\vec{y})|}{\pi} \theta_R(\vec{\omega}, \vec{y}, t)\, d\vec{\omega} \right) \end{aligned} \tag{12}$$

2.4. Convection

Inside a fluid sub-volume $\mathcal{D}_i$, at any time t, the fluid temperature $T_i \equiv T_i(t)$ is uniform and its evolution equation is

$$\rho_i c_i V_i \frac{dT_i}{dt} = \int_{\partial\mathcal{D}_i} h(\vec{y})(T_S(\vec{y}) - T_i)d\vec{y} \tag{13}$$

where $h(\vec{y})$ and $T_S(\vec{y})$ are respectively the convective exchange coefficient and the surface temperature at $\vec{y}$ on one of the solid surfaces delimiting the fluid cell. If $\vec{y}$ is at an interface $\mathcal{I}_k$ with a solid sub-volume $\mathcal{D}_j$, then $h(\vec{y}) = h_k$ and $T_S(\vec{y}) = T_j(\vec{y}, t)$. If $\vec{y}$ is at the boundary of the system, in a sub-surface $\mathcal{S}_j$ with a *type-3* boundary condition, then $h(\vec{y}) = h_j$ and $T_S(\vec{y}) = T_{\mathrm{B},j}(\vec{y}, t)$.

3. Path sampling and propagation

In this section, we expose our reverse Monte Carlo algorithm, sampling paths driven by the model above, to estimate a local temperature at location $\vec{x}$ and time t, or a radiance temperature at location $\vec{x}$ and time t in direction $\vec{\omega}$.

When integrated quantities are required (an average temperature on a volume or a surface, a spatially and angularly integrated radiance for simulation of infrared camera pixels, ...), the only algorithmic change is that, prior to initiating a thermal path, $\vec{x}$ and/or $\vec{\omega}$ are sampled accordingly. We will not provide here description of such extensions.

3.1. A path sampling Monte Carlo algorithm

Depending on the choice of the quantity of interest, the estimate will yield either $\theta_R(\vec{\omega}, \vec{x}, t)$ for $\vec{x}$ inside a fluid sub-volume, either $T_i(\vec{x}, t)$ for $\vec{x}$ inside a solid sub-volume, or $T_i(t)$ for $\vec{x}$ inside a fluid sub-volume. In each case, N thermal paths γ_j are sampled and each path is used to produce a Monte Carlo weight w_{γ_j}. These weights are then averaged to produce an estimate m of the addressed quantity, together with a standard deviation s associated to this estimate, that can be interpreted in term of a numerical uncertainty.

$$T_i(\vec{x}, t) \text{ or } T_i(t) \text{ or } \theta_R(\vec{\omega}, \vec{x}, t) \approx m = \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j} \tag{14}$$

$$s = \frac{1}{\sqrt{N}}\left(\frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^2 - m^2\right) \tag{15}$$

We concentrate here on the calculation of w_γ for any path γ, highlighting its propagative nature and how the information about the sources is gathered along the path.

A thermal path is structured as a succession of conductive, convective and radiative sub-paths. From this point of view, the only difference between the paths used to evaluate $\theta_R(\vec{\omega}, \vec{x}, t)$ in a fluid, $T_i(\vec{x}, t)$ in a solid or $T_i(t)$ in a fluid is that they start with a radiative sub-path, a conductive sub-path or a convective sub-path respectively. We can therefore consider each sub-path independently, only keeping in mind that : a) at the beginning of the first sub-path the Monte Carlo weight is initiated to $w_\gamma = 0$, and b) that the end of each sub-path is either the start of a new sub-path, or the end of the whole path γ. Each path γ ends at a location $\vec{x}_{\gamma,\mathrm{end}}$, either inside the system at the initial time t_I or at the boundary at a time $t_{\gamma,\mathrm{end}}$. When it ends with a known incident radiant temperature at the boundary, the corresponding incident direction is $\vec{\omega}_{\gamma,\mathrm{end}}$.

3.1.1. Radiative sub-paths

Radiative sub-paths are constructed using a standard backward tracking multiple-reflection algorithm. Starting from $\vec{x}$ with the objective of evaluating $\theta_R(\vec{\omega}, \vec{x}, t)$, a ray is traced in the scene in direction $-\vec{\omega}$, looking for a first intersection $\vec{z}_1$ with a solid surface. If no intersection is found ($\vec{z}_1$ is at infinity), then our radiation model says that $\theta_R(\vec{\omega}, \vec{x}, t) = \theta_{\mathrm{BR}}(\vec{\omega}, \vec{x}, t)$ where θ_{BR} is a known incident radiance temperature. In this case, the path γ is ended at location $\vec{x}_{\gamma,\mathrm{end}} = \vec{x}$, the time $t_{\gamma,\mathrm{end}} = t$ and the direction $\vec{\omega}_{\gamma,\mathrm{end}} = \vec{\omega}$.

The Monte Carlo weight is increased by θ_{BR}:

$$w_\gamma \mathrel{+}= \theta_{\mathrm{BR}}(\vec{\omega}_{\gamma,\mathrm{end}}, \vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \tag{16}$$

Otherwise $\vec{z}_1$ belongs either to a sub-surface $\mathcal{S}_j$ or a sub-interface $\mathcal{S}_j$ where the emissivity ϵ_j and the specular/diffuse fraction α_j are known. A Bernoulli test of probability ϵ_j is made to decide whether absorption occurs. If the test is true, the radiative sub-path is ended at $\vec{z}_1$. If not, reflection occurs (with another Bernoulli test to decide between specular or diffuse reflection and a Lambertian sampling of the reflection direction in the diffuse case) and the path tracing process is continued from $\vec{z}_1$ in the direction of reflection $-\vec{\omega}_1$, etc, thus defining a succession of possible reflections at locations $\vec{z}_1$, $\vec{z}_2$, $\vec{z}_3$ ... until either no reflection is found or absorption occurs. If no reflection is found, the path is ended with the Monte Carlo weight increment of Eq. (16). When absorption occurs as a location $\vec{z}_k$, then there are two possible cases :

- If $\vec{z}_k$ belongs to a sub-surface at the system boundary and the temperature T_B is known at this surface (a radiative path travels necessarily in a fluid and only a *type-4* boundary condition can be encountered), the path γ is ended and the Monte Carlo weight is increased by T_B:

$$w_\gamma \mathrel{+}= T_\mathrm{B}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \tag{17}$$

  with $\vec{x}_{\gamma,\mathrm{end}} = \vec{z}_k$ and $t_{\gamma,\mathrm{end}} = t$.

- If $\vec{z}_k$ belongs to a sub-interface, then the encountered solid is inside the system and its temperature is unknown. The path γ must be continued with a new sub-path (with no change of w_γ) and a test is made to decide between the three heat transfer modes occurring at this interface: a radiative sub-path back into the fluid, a convective sub-path also in the fluid, or a conductive path inside the solid. This test is the object of Sec. 3.2.

Summary. : A radiative sub-path is an instantaneous backward traced ray in a transparent fluid with multiple reflections at solid surfaces. If the sub-path encounters a known incident radiance or a known solid temperature, then γ is ended and the Monte Carlo weight is increased by θ_BR or T_B. Otherwise, the Monte Carlo weight is unchanged and at the absorption location, γ is continued with the start of another sub-path at the corresponding solid-fluid interface.

3.1.2. Conductive sub-paths

Conductive sub-paths are approximate Brownian motions backward both in time and space inside a solid. They are constructed as successions of jumps of arbitrary length δ and in isotropically sampled directions. Convergence towards the exact solution is obtained for $\delta \to 0$ (Brownian motion is only exact at the limit $\delta = 0$). However, as the computational time increases considerably when the value of δ decreases, a compromise is required between computational cost and precision. Hence, δ needs to be set sufficiently low to ensure a satisfactory accuracy on the obtained solution and sufficiently high to provide an appropriate computational time.

Starting from $\vec{x}$ with the objective of evaluating $T(\vec{x}, t)$, the first algorithmic step is the sampling of a backward time shift δt according to an exponential law of parameter $\tau_i = \frac{\delta^2 \rho_i c_i}{6\lambda_i}$, i.e.

$$\delta t = -\tau_i \ln(r) \tag{18}$$

where r is sampled uniformly on $[0, 1]$. If $t - \delta t < t_\mathrm{I}$ (the backward shift has crossed the initial time), then γ is ended and the Monte Carlo weight is increased by the initial temperature:

$$w_\gamma \mathrel{+}= T_\mathrm{I}(\vec{x}_{\gamma,\mathrm{end}}) \tag{19}$$

with $\vec{x}_{\gamma,\mathrm{end}} = \vec{x}$. Otherwise a direction $\vec{u}$ is sampled isotropically in the unit sphere, a jump is made from $\vec{x}$ to $\vec{x} + \delta\vec{u}$ and the Monte Carlo weight is increased to account for the local power density ψ:

$$w_\gamma \mathrel{+}= \beta_\psi(\vec{x})\psi(\vec{x}, t - \delta t) \tag{20}$$

with $\beta_\psi = \frac{\delta^2}{6\lambda_i}$. δ is adjusted according to $\vec{u}$ in the vicinity of a solid surface so that $\vec{x} + \delta\vec{u}$ may either remain inside the solid or reach the solid surface exactly.

If $\vec{x} + \delta\vec{u}$ is still inside the same solid sub-volume, say $\mathcal{D}_i$, then the conductive sub-path is simply continued from this new location and this recursively until reaching the surface $\partial\mathcal{D}_i$ at a location $\vec{z}$ and time t_z where the conductive sub-path is stopped.

If the corresponding location belongs to an interface with another solid sub-volume $\mathcal{D}_j$, then the temperature of the interface is unknown and γ must be continued with another conductive sub-path, initiated either inside $\mathcal{D}_i$ or inside $\mathcal{D}_j$. If $\vec{z}$ belongs to an interface with a fluid sub-volume, then the temperature of the interface is also unknown and γ must be continued, but the next sub-path can be either a conductive one, back into $\mathcal{D}_i$, or a convective or a radiative one inside or through the fluid. The corresponding tests are described in Sec. 3.2.

If $\vec{z}$ is at the boundary of the system, then the algorithm depends on the boundary condition type:

- For a *type-1* boundary condition, the boundary temperature T_B is known and γ is stopped and the Monte Carlo weight is increased by T_B:
$$w_\gamma \mathrel{+}= T_\mathrm{B}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \tag{21}$$
with $\vec{x}_{\gamma,\mathrm{end}} = \vec{z}$ and $t_{\gamma,\mathrm{end}} = t_z$.
- For a *type-2* boundary condition, the location is shifted back into the solid sub-volume, of a distance δ along the normal, the conductive sub-path is continued from this new location, and the Monte Carlo weight is increased to account for the value of the local flux density:
$$w_\gamma \mathrel{+}= \beta_\varphi(\vec{z})\varphi(\vec{z}, t_z) \tag{22}$$
with $\beta_\varphi = \frac{\delta}{\lambda_i}$.
- For a *type-3* boundary condition, neither the boundary temperature nor the flux density is known and γ is continued exactly the same way as for a solid-fluid interface inside the system (see Sec. 3.2). The only difference is that when the following sub-path is a convective one, then the fluid temperature T_BF is known and γ is ended. In this case, the Monte Carlo weight is increased by T_BF:
$$w_\gamma \mathrel{+}= T_\mathrm{BF}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \tag{23}$$
with $\vec{x}_{\gamma,\mathrm{end}} = \vec{z}$ and $t_{\gamma,\mathrm{end}} = t_z$.

Summary. : A conductive sub-path is a Brownian motion backward in time inside a solid sub-volume until it reaches either the initial time or the sub-volume boundary. If the initial time is reached, then γ is ended and the Monte Carlo weight is increased by T_I. If the system boundary is reached at location where the temperature is known, then γ is ended and the Monte Carlo weight is increased by T_B. If the system boundary is reached at location where the flux density is known, then a new conductive sub-path is initiated, inside the same sub-volume, and the Monte Carlo weight is increased by $\beta_\varphi\varphi$. In all other cases, the conductive sub-path has reached a location where neither the temperature nor the flux density is known, and a new sub-path (conductive, convective or radiative) must be initiated from the corresponding interface. Along the path, the Monte Carlo weight is increased to account for the local value of the volume source density ψ. As Brownian motion is approximated with discrete jumps of length δ, the continuous effect of the source is replaced by a Monte Carlo weight increment of $\beta_\psi\psi$ at each jump.

3.1.3. Convective sub-paths

Convective sub-paths inside a fluid sub-volume $\mathcal{D}_i$ are independent of their initial location: the fluid cells are perfectly mixed so T_i is only a function of time, and the only required information is the time t at which the convective sub-path was initiated. From t, a backward time shift δt is sampled according to an exponential law of parameter $\frac{h_i V_i}{\rho_i c_i S_i}$, i.e.
$$\delta t = \frac{h_i V_i}{\rho_i c_i S_i} \ln(r) \tag{24}$$
where S_i and V_i are respectively the surface and volume of the fluid cavity, r is sampled uniformly on $[0, 1]$. If $t - \delta t < t_\mathrm{I}$ (the backward shift has crossed the initial time), then γ is ended and the Monte Carlo weight is increased by the initial temperature:
$$w_\gamma \mathrel{+}= T_{\mathrm{I},i} \tag{25}$$

Otherwise a location $\vec{z}$ is sampled on $\partial\mathcal{D}_i$ according to probability density $p_{\vec{Z}}$ proportional to the local value of the convective exchange coefficient:

$$p_{\vec{Z}}(\vec{z}) = \frac{h(\vec{z})}{\int_{\partial\mathcal{D}_i} h(\vec{z}')d\vec{z}'} \tag{26}$$

and the time is shifted to $t_z = t - \delta t$.

If $\vec{z}$ is at the system boundary, then this corresponds necessarily to a *type-4* boundary condition and the boundary temperature T_B is known, so γ is ended and the Monte Carlo weight is increased by T_B:

$$w_\gamma \mathrel{+}= T_\mathrm{B}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \tag{27}$$

with $\vec{x}_{\gamma,\mathrm{end}} = \vec{z}$ and $t_{\gamma,\mathrm{end}} = t_z$. Otherwise $\vec{z}$ is at a solid-fluid interface inside the system and the interface temperature is unknown. γ is then continued with a new conductive, convective or radiative sub-path as described in Sec. 3.2.

Summary. : A convective sub-path inside a fluid sub-volume is only a backward exponential shift in time. If the initial time is reached, then γ is ended and the Monte Carlo weight is increased by T_I. Otherwise a location is sampled on one of the solid surfaces surrounding the fluid. If this location is at the boundary of the system and the surface temperature is known, then γ is ended and the Monte Carlo weight is increased by T_B. Otherwise the convective path has reached an interface where the temperature is unknown and a new sub-path (conductive, convective or radiative) must be initiated from this interface.

3.2. Choosing the next sub-path at a solid-solid or a solid-fluid interface

When describing conductive sub-paths, we encountered an algorithmic step where a location $\vec{z}$ was reached, at the interface between two solid sub-volumes $\mathcal{D}_i$ and $\mathcal{D}_j$, at time t_z. At this interface, the temperature was unknown and γ had to be continued. The exact same question is raised when γ needs to be started at such a location in order to evaluate $T(\vec{x}, t)$ with $\vec{x} = \vec{z}$ and $t = t_z$. This is achieved by first shifting $\vec{z}$ along the normal, either by a distance δ_i inside $\mathcal{D}_i$ or by a distance δ_j inside $\mathcal{D}_j$, where δ_i and δ_j are the values of the numerical parameter δ used inside $\mathcal{D}_i$ and inside $\mathcal{D}_j$ respectively (depending of their characteristic dimensions). Then a conductive sub-path is started from this shifted location, still at the same date. Choosing the side is made by retaining $\mathcal{D}_i$ with probability $P_{\mathrm{cond},i}$ and $\mathcal{D}_j$ with probability $P_{\mathrm{cond},j}$:

$$\begin{aligned} P_{\mathrm{cond},i} &= \frac{\frac{\lambda_i}{\delta_i}}{\frac{\lambda_i}{\delta_i} + \frac{\lambda_j}{\delta_j}} \\ P_{\mathrm{cond},j} &= 1 - P_{\mathrm{cond},i} \end{aligned} \tag{28}$$

When describing each of the three sub-path types, we encountered the possibility that a location $\vec{z}$ is reached, at an interface $\mathcal{I}_k$ between a solid sub-volume $\mathcal{D}_i$ and a fluid sub-volume $\mathcal{D}_j$, at time t_z. At this interface the temperature is unknown and γ is to be continued. The exact same question is raised when γ needs to be started at such a location in order to evaluate $T(\vec{x}, t)$ with $\vec{x} = \vec{z}$ and $t = t_z$. The same question is also encountered when a conductive sub-path reaches a location at the system boundary with *type-4* boundary condition. We concentrate the description on the case of an internal interface between $\mathcal{D}_i$ and $\mathcal{D}_j$.

At such an interface, all three heat transfer modes are present: conduction inside $\mathcal{D}_i$, convection and radiation inside Ω_j. A test is therefore made to decide among a conductive, a convective or a radiative sub-path. Conductive, convective and radiative sub-paths are decided with probabilities P_cond, P_conv and P_ray respectively, with

$$\begin{aligned} P_\mathrm{cond} &= \frac{\frac{\lambda_i}{\delta_i}}{\frac{\lambda_i}{\delta_i} + h_k + h_R} \\ P_\mathrm{conv} &= \frac{h_k}{\frac{\lambda_i}{\delta_i} + h_k + h_R} \\ P_\mathrm{ray} &= 1 - P_\mathrm{cond} - P_\mathrm{conv} \end{aligned} \tag{29}$$

If conduction is retained, then $\vec{z}$ is shifted of along the normal of a distance δ_i inside $\mathcal{D}_i$ and the conductive sub-path is initiated at this shifted location. For convection or radiation, the corresponding sub-path is initiated at $\vec{z}$ at time t_z.

Summary. : Departing from a solid-solid or a solid-fluid interface is made by initiating a sub-path with a heat transfer mode that is sampled according to probabilities reflecting the flux continuity through the interface. When a conduction sub-path is chosen, the start location is shifted by a distance δ inside the solid. In all cases, there is no increment made to the Monte Carlo weight.

3.3. The Monte Carlo weight

As mentioned above, each path γ ends at a location $\vec{x}_{\gamma,\mathrm{end}}$, either inside the system at the initial time t_I or at the boundary at a time $t_{\gamma,\mathrm{end}}$. When it ends with a known incident radiant temperature at the boundary, the corresponding incident direction is $\vec{\omega}_{\gamma,\mathrm{end}}$. In each case, the Monte Carlo weight is increased by a temperature value that can be $T_\mathrm{I}(\vec{x}_{\gamma,\mathrm{end}})$, $T_\mathrm{B}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}})$, $T_\mathrm{BF}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}})$ or $\theta_\mathrm{BR}(\vec{\omega}_{\gamma,\mathrm{end}}, \vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}})$. Let $\mu_{\gamma,\mathrm{end}}$ denote the type of ending, from 0 to 3 in the order of this list.

We can then define $T_{\gamma,\mathrm{end}} \equiv T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}}, \mu_{\gamma,\mathrm{end}})$ as :

$$\begin{aligned}
T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}}, 0) &= T_\mathrm{I}(\vec{x}_{\gamma,\mathrm{end}}) \\
T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}}, 1) &= T_\mathrm{B}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \\
T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}}, 2) &= T_\mathrm{BF}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \\
T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}}, 3) &= \theta_\mathrm{BR}(\vec{\omega}_{\gamma,\mathrm{end}}, \vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}})
\end{aligned} \tag{30}$$

Along γ, conductive sub-paths may have crossed solid sub-volumes with a volume power source ψ, and we have seen that the Monte Carlo weight was increased by $\beta_\psi\psi$ at each discrete jump location. Let N_ψ denote the number of such locations, and $\vec{x}_{\gamma,\psi}(k)$ and $t_{\gamma,\psi}(k)$ the location and time of the k-th of these Monte Carlo weight increments. Similarly, conductive sub-paths may have visited boundary locations where the flux density φ is known, and we have seen that the Monte Carlo weight was increased by $\beta_\varphi\varphi$ at each such visit. Let N_φ denote the number of such visits, and $\vec{x}_{\gamma,\varphi}(k)$ and $t_{\gamma,\varphi}(k)$ the location and time of the k-th of these Monte Carlo weight increments.

With these notations, the complete expression of the Monte Carlo weight associated to γ is

$$\begin{aligned}
w_\gamma = &\sum_{k=1}^{N_\psi} \beta_\psi(\vec{x}_{\gamma,\psi}(k))\, \psi\left(\vec{x}_{\gamma,\psi}(k), t_{\gamma,\psi}(k)\right) \\
&+ \sum_{k=1}^{N_\varphi} \beta_\varphi(\vec{x}_{\gamma,\varphi}(k))\, \varphi\left(\vec{x}_{\gamma,\varphi}(k), t_{\gamma,\varphi}(k)\right) \\
&+ T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}}, \mu_{\gamma,\mathrm{end}})
\end{aligned} \tag{31}$$

The reading of this weight expression illustrates the intimate relation between such path sampling Monte Carlo algorithms and Green's theory. As mentioned above, Green's theory considers the temperature as a solution of the problem at $(\vec{x}*, t*)$, or the radiance temperature at $(\vec{\omega}*, \vec{x}*, t*)$, which results in an integral of all the sources, T_I, T_B, T_BF, θ_BR, ψ and φ, at $(\vec{x}, t, \vec{\omega})$ multiplied by a propagator density that is independent of the source values; Monte Carlo considers it as the average of a large number of weights that carry the same sources from $(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}})$ or from $(\vec{x}_{\gamma,\psi}(k), t_{\gamma,\psi}(k))$ and $(\vec{x}_{\gamma,\varphi}(k), t_{\gamma,\varphi}(k))$, multiplied by factors that are independent of the source values: the factor is 1 for T_I, T_B, T_BF and θ_BR, it is β_ψ for ψ and β_φ for φ. The paths sample $\zeta(\vec{x}, t, \vec{x}_S, t_S)S(\vec{x}_S, t_S)$ in a backward manner.

4. Stardis : storing the propagation data

4.1. Illustration of the principle in the case of two sources

Let us start by considering a simple configuration akin to the radiative transfer example with two lamps of power $\mathcal{P}_1$ and $\mathcal{P}_2$ viewed from a camera pixel.

Here, the heat transfer mode is conduction inside a cubic solid with two isothermal faces $\mathcal{S}_1$ and $\mathcal{S}_2$, facing each other, at temperatures $T_{\mathrm{B},1}$ and $T_{\mathrm{B},2}$, the other four faces being adiabatic (see Fig. 4). The addressed quantity is the stationary temperature $T(\vec{x})$ at a location $\vec{x}$ inside the solid. In terms of Green's theory, as the problem is stationary, no propagator is required for the initial condition (reported to $-\infty$). There are no volume sources and the only imposed

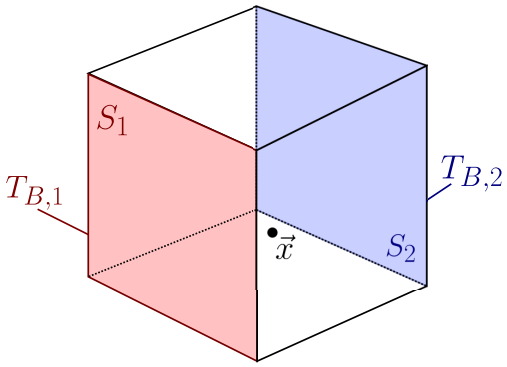

Figure 4: Schemes of the solid cube with two isothermal faces S_1 and S_2, respectively at temperature $T_{\mathrm{B},1}$ and $T_{\mathrm{B},2}$, with a location $\vec{x}$ inside the solid volume.

surface flux is null (at the adiabatic faces). The only sources are therefore $T_{\mathrm{B},1}$ and $T_{\mathrm{B},2}$ and we note $\zeta_{\mathrm{B},1}(\vec{x})$ and $\zeta_{\mathrm{B},2}(\vec{x})$ the corresponding propagators:

$$T(\vec{x}) = \zeta_{\mathrm{B},1}(\vec{x})T_{\mathrm{B},1} + \zeta_{\mathrm{B},2}(\vec{x})T_{\mathrm{B},2} \tag{32}$$

Considering the expression of the Monte Carlo weight in Eq. (31), how can we provide an estimate for $\zeta_{\mathrm{B},1}(\vec{x})$ and one for $\zeta_{\mathrm{B},2}(\vec{x})$ using the same thermal paths as those used to estimate $T(\vec{x})$?

In the expression of the Monte Carlo weight of the preceding section, for this simple case, the sums over N_ψ and N_φ vanish (no surface flux, no volume flux), $\vec{\omega}_{\gamma,\mathrm{end}}$ is not used (we estimate a local temperature and not a radiance temperature), $t_{\gamma,\mathrm{end}}$ is unused (the problem is stationary) and $\mu_{\gamma,\mathrm{end}} = 1$ (the path can only end at $\mathcal{S}_1$ or $\mathcal{S}_2$, i.e. at a boundary with a known solid temperature). Eq. (31) reduces to

$$w_\gamma = T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, 1) \tag{33}$$

with $T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, 1) = T_{\mathrm{B},1}$ if $\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_1$ and $T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, 1) = T_{\mathrm{B},2}$ if $\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_2$.

In the spirit of the example used in introduction, let us address $\zeta_{\mathrm{B},1}(\vec{x})$ by "turning off" the second source, i.e. $T_{\mathrm{B},2} = 0$, so that $\zeta_{\mathrm{B},1}(\vec{x}) = \frac{T(\vec{x})}{T_{\mathrm{B},1}}$.

This defines the Monte Carlo weight to be used for estimating $\zeta_{\mathrm{B},1}(\vec{x})$ as : $w_\gamma^{\mathrm{B},1} = \frac{w_\gamma}{T_{\mathrm{B},1}} = \frac{T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}},1)}{T_{\mathrm{B},1}}$
with $T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, 1) = T_{\mathrm{B},1}$ if $\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_1$ and $T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, 1) = 0$ if $\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_2$,
namely, $w_\gamma^{\mathrm{B},1} = 1$ if $\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_1$ and $w_\gamma^{\mathrm{B},1} = 0$ if $\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_2$.
This writes

$$w_\gamma^{\mathrm{B},1} = \mathcal{H}(\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_1) \tag{34}$$

where $\mathcal{H}$ is a test function, taking the value 1 if the condition is valid and 0 otherwise. Similarly,

$$w_\gamma^{\mathrm{B},2} = \mathcal{H}(\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_2) \tag{35}$$

In algorithmic terms,

- N paths γ_j are sampled;
- $w_{\gamma_j}^{\mathrm{B},1}$ is computed for each path (1 if $\mathcal{S}_1$ is reached, 0 otherwise);
- $w_{\gamma_j}^{\mathrm{B},2}$ is computed for each path (1 if $\mathcal{S}_2$ is reached, 0 otherwise);
- the Monte Carlo estimate of $\zeta_{\mathrm{B},1}(\vec{x})$ is $m_{\mathrm{B},1} = \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{B},1}$;
- the Monte Carlo estimate of $\zeta_{\mathrm{B},2}(\vec{x})$ is $m_{\mathrm{B},2} = \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{B},2}$.

Then, Eq. (32) can be used to estimate the results for any set of source values.

4.2. Implementation in Stardis

In *Stardis*, we consider scenes where boundary conditions can be split into a set of constant and uniform sources: $\theta_{i\mathrm{BR}}$ does not depend on location, time and direction, and for each geometrical element i, $T_{\mathrm{B},i}$, $\varphi_{\mathrm{B},i}$ and ψ_i are constant and uniform, $T_{\mathrm{I},i}$ does not depend on location. λ_i, ρ_i, c_i, h_i, ϵ_i and α_i are also considered uniform over element i.

Under these assumptions, $\zeta(\vec{x}, t, \vec{x}_S, t_S)$ can be aggregated by geometrical element as done above in Eqs. (34) and (35). For this, we use the test function to build one weight expression for each geometrical element i and each type of source, following:

$$\begin{aligned}
w_\gamma^{\mathrm{I},i} &= \mathcal{H}(\vec{x}_{\gamma,\mathrm{end}} \in \Omega_i)\mathcal{H}(\mu_{\gamma,\mathrm{end}} = 0)\\
w_\gamma^{\mathrm{B},i} &= \mathcal{H}(\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_i)\mathcal{H}(\mu_{\gamma,\mathrm{end}} = 1)\\
w_\gamma^{\mathrm{BF},i} &= \mathcal{H}(\vec{x}_{\gamma,\mathrm{end}} \in \mathcal{S}_i)\mathcal{H}(\mu_{\gamma,\mathrm{end}} = 2)\\
w_\gamma^{\mathrm{BR}} &= \mathcal{H}(\mu_{\gamma,\mathrm{end}} = 3)\\
w_\gamma^{\psi,i} &= \sum_{k=1}^{N_\psi} \mathcal{H}(\vec{x}_{\gamma,\psi}(k)) \in \Omega_i)\beta_\psi(\vec{x}_{\gamma,\psi}(k))\\
w_\gamma^{\varphi,i} &= \sum_{k=1}^{N_\varphi} \mathcal{H}(\vec{x}_{\gamma,\varphi}(k) \in \mathcal{S}_i)\beta_\varphi(\vec{x}_{\gamma,\varphi}(k))
\end{aligned} \tag{36}$$

These weights are then just a splitting of the weights given by Eq. 31 and only them need to be stored while solving the latter. The Monte Carlo algorithm is therefore modified to estimate:

$$\begin{aligned}
m_{\mathrm{I},i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{I},i}\\
m_{\mathrm{B},i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{B},i}\\
m_{\mathrm{BF},i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{BF},i}\\
m_{\mathrm{BR}} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{BR}}\\
m_{\psi,i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\psi,i}\\
m_{\varphi,i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\varphi,i}
\end{aligned} \tag{37}$$

Finally, once these estimates have been constructed, they can be used to estimate the addressed quantity for any set of

sources:

$$
\begin{aligned}
T(\vec{x},t) \text{ or } T(t) \text{ or } \theta_R(\vec{\omega},\vec{x},t) \approx m = & \sum_{i=1}^{N_\omega} m_{\mathrm{I},i}\, T_{\mathrm{I},i} \\
& + \sum_{i=1}^{N_S} m_{\mathrm{B},i}\, T_{\mathrm{B},i} \\
& + \sum_{i=1}^{N_S} m_{\mathrm{BF},i}\, T_{\mathrm{BF},i} \\
& + m_{\mathrm{BR}}\, \theta_{\mathrm{BR}} \\
& + \sum_{i=1}^{N_\omega} m_{\psi,i}\, \psi_i \\
& + \sum_{i=1}^{N_S} m_{\varphi,i}\, \varphi_i
\end{aligned}
\tag{38}
$$

4.3. Uncertainty estimation

The question of quantifying the uncertainty is less trivial than with an usual Monte Carlo (see Eq. (15)). In Eq. (37), it appears clearly that a Monte Carlo approach is used for estimating each propagator as an average of dedicated weights. We could therefore compute an uncertainty associated to each of these estimates by estimating the standard deviation of the Monte Carlo weights for each propagator. This would indeed provide a faithful information about the uncertainty with which each propagator is known for a given number of sampled thermal-paths, but this information would be insufficient to estimate the uncertainty of the finally addressed quantity ($T(\vec{x},t)$ for instance). The reason is that the same set of thermal-paths has been used to estimate all the propagators (in one single Monte Carlo run) and therefore the propagator estimates are correlated. When estimating $T(\vec{x},t)$ for a new set of sources in Eq. (38), i.e. summing the known sources multiplied by their propagators, these correlations seem to indicate that addressing the uncertainty of this estimate requires the knowledge of a correlation matrix.

In practice, this can be avoided because the paths that have been used to estimate each propagator are precisely those that would be used in a Monte Carlo run addressing $T(\vec{x},t)$. Although we have chosen to store the propagators for parts of the system (and not all the information carried by each path independently) we can still address the estimation of the uncertainty of the estimate m of $T(\vec{x},t)$ as if we had stored the paths. Somehow, the uncertainty is always to be estimated using Eq. (15). Practically, this means that for each sampled path, we increment a counter for the sum of the squares of the Monte Carlo weights for each propagator, leading to the final storing of the following sums,

$$
\begin{aligned}
ssq_{\mathrm{I},i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\mathrm{I},i}\right)^2 \\
ssq_{\mathrm{B},i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\mathrm{B},i}\right)^2 \\
ssq_{\mathrm{BF},i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\mathrm{BF},i}\right)^2 \\
ssq_{\mathrm{BR}} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\mathrm{BR}}\right)^2 \\
ssq_{\psi,i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\psi,i}\right)^2 \\
ssq_{\varphi,i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\varphi,i}\right)^2
\end{aligned}
\tag{39}
$$

that are then used to compute s as

$$
\begin{aligned}
s = \frac{1}{\sqrt{N}}\Bigg(\frac{1}{N}\Bigg(&\sum_{i=1}^{N_\omega} ssq_{\mathrm{L},i}\, T_{\mathrm{L},i}^2 \\
&+ \sum_{i=1}^{N_S} ssq_{\mathrm{B},i}\, T_{\mathrm{B},i}^2 \\
&+ \sum_{i=1}^{N_S} ssq_{\mathrm{BF},i}\, T_{\mathrm{BF},i}^2 \\
&+ ssq_{\mathrm{BR}}\, \theta_{\mathrm{BR}}^2 \\
&+ \sum_{i=1}^{N_\omega} ssq_{\psi,i}\, \psi_i^2 \\
&+ \sum_{i=1}^{N_S} ssq_{\varphi,i}\, \varphi_i^2\Bigg) - m^2\Bigg)^{1/2}
\end{aligned}
\tag{40}
$$

Of course, if for some reasons one propagator is to be addressed independently, its uncertainty can always be addressed using the sum of the square of the corresponding weights. The only point is that although this uncertainty would tell us about the accuracy with which the propagator is known, because of the correlations, it will not be a faithful information about how accurately $T(\vec{x}, t)$ is known. Eq. (38) is to be preferred.

5. Implementation

The implementation of `stardis-solver`, that is presented here, is a reference implementation suitable for execution with conventional computing resources (low-end personal computer).

The source code of the solver is designed to be easy to understand and suitable for training purposes. Users can then rely on this implementation and make it evolve according to their needs.

The current implementation is a compromise between the different possibilities described in Sec. 4. This compromise consists in:

- Grouping the terms related to volume power densities and heat flux densities, restraining heat flux and power to be uniform over time and space.
- Keeping all positions and times for other sources (initial temperature, ambient radiation temperature, fluid temperature, imposed temperature), allowing these sources to vary, either over time or space or both.

5.1. Code structure

Code structure is briefly presented to help the reader understand the topics related to the Green function. First, all the data structures and functions described thereafter in a literate programming-inspired way [1], are located in the file `sdis_green.c`. The file is structured as follows:

```
<sdis_green.c> =
  <license>
  <inclusions>
  <secondary types and functions>
  <green data structures> (1)
  <helper functions>
  <evaluation functions> (3)
  <build functions> (2)
```

The three main parts of interest and detailed below are:

(1) Data structures used to store the Green function (see Sec. 5.2).

(2) Functions used to fill up these data structures in the construction of the Green function (see Sec. 5.3).

(3) Functions using these data structures to evaluate the temperature for a given set of source values (see Sec. 5.4).

The source code for the data structures and functions described in these three sections are grouped in Appendix A at the end of the document.

5.2. Data structures

When building the Green function, the Monte-Carlo weights are not computed, but the data needed to compute them is stored, path by path, for later use. This storage requires two different types: one to store the data collected along individual Green paths, and another one to store the Green function itself, including the data of the sampled Green paths, as well as all the shared data referenced by the paths (materials, interfaces, ...):

```
<green data structures> =
  <green path data structure>
  <green function data structure>
```

Path storage. Green paths are constructed by `stardis-solver`, following the very same algorithm as when evaluating a temperature. The difference between temperature computation and the construction of the Green function is that when a path is sampled, some of the information is stored in the data structure corresponding to the Green path instead of being used on the fly to compute a temperature (see List. 2). Then, each path sampled by the solver results in a Green path data structure storing the information as follows:

```
<green path data structure> =
  struct green_path {
    <path elapsed time>
    <flux density terms collection>
    <power density terms collection>
    <end of path>
    <miscellaneous variables>
  };
```

Elapsed time is trivially a double:

```
<path elapsed time> =
  double elapsed_time;
```

Flux and power terms encountered along the path are partially merged and stored in dynamic arrays. Merging is done by material and interface: all contributions along the path are accumulated and stored as a single term associated with a given material or interface.

```
<flux density terms collection> =
  struct darray_flux_term flux_terms;
<power density terms collection> =
  struct darray_power_term power_terms;
```

As flux terms can only appear at interfaces, merged flux terms consist of an interface identifier, the involved side and the corresponding cumulated flux value. On the other hand, power terms appear in media, thus merged power terms consist of a medium identifier and the cumulated power value.

```
struct power_term {
  double term;
  unsigned id; /* Identifier of the medium */
};

struct flux_term {
```

```
  double term;
  unsigned id; /* Identifier of the interface */
  enum sdis_side side;
};
```

The end of the path can be of three types: at a boundary (`fragments`), in a volume (`vertex`), or a radiative exchange with the surrounding environment. Classically this end is represented by an union which is interpreted according to the value of the field `limit_type`, which also allows to interpret `limit_id` as being an identifier of medium (case in volume) or interface (case at boundary); note that the radiative case require neither an union member nor a `limit_id`:

```
<end of path> =
  union {
    struct sdis_rwalk_vertex vertex;
    struct sdis_interface_fragment fragment;
  } limit;
  unsigned limit_id;
  enum sdis_green_path_end_type end_type;
```

Green function storage. The main structure is used to store everything allowing the later evaluation of a temperature estimator. This includes the description of the sampled paths as well as all the shared data referenced by the sampled paths (see List. 1).

```
<green function data structure> =
  struct sdis_green_function {
    <media collection>
    <interfaces collection>
    <paths collection>
    <miscellaneous variables>
  };
```

Collections of media and interfaces accumulate the media and interfaces that have been visited when constructing the Green function. Individual paths can then reference this shared information. These collections are hash tables, i.e. associative containers that favor fast and constant time lookup, to ensure unique storage of only the media and interfaces visited by the paths:

```
<media collection> =
  struct htable_medium media;
<interfaces collection> =
  struct htable_interf interfaces;
```

The paths collection is the set of the paths sampled for the construction of the propagator. It is a dynamic array that is well suited for iterative storage and path iteration:

```
<paths collection> =
  struct darray_green_path paths;
```

5.3. Building propagation information

Various functions are needed to fill up the Green function's data structures. They can be divided in two groups:

```
<build functions> =
  <functions building the green function>
  <functions building green paths>
```

Functions building the Green function itself do not need further description, as they are limited to collections management.

```
<functions building green paths> =
  <functions that store path ending>
  <functions that accumulate data along a path>
```

5.3.1. Storing path ending

Since there are three different ways to end a path, there are three different functions that can be called to store information about the end of paths (see List. 3):

```
<functions that store path ending> =
  <store path's end at an interface>
  <store path's end in a medium>
  <store radiative path's end>
```

Let's start by describing the first function:

```
<store path's end at an interface> =
  res_T
  green_path_set_limit_interface_fragment
    (struct green_path_handle* handle,
     struct sdis_interface* interf,
     const struct sdis_interface_fragment* frag,
     const double elapsed_time)
  {
    res_T res = RES_OK;
    <check input arguments>
    <register interface 'interf' against the green function>
    <store path duration>
    <store the location at interface>
    <store identifier of interface 'interf'>
    <store the path ends up at an interface>
    return RES_OK;
  }
```

The current path ends at an interface that must be available at the time the Green function is evaluated. We start by making sure that is the case, returning an error if the process fails.

```
<register interface 'interf' against the green function> =
  res = ensure_interface_registration(handle->green, interf);
  if(res != RES_OK) return res;
```

Then the elapsed time is stored:

```
<store path duration> =
  handle->path->elapsed_time = elapsed_time;
```

Then the information related to the location at interface is stored (including position, normal, parametric coordinates and time):

```
<store the location at interface> =
  handle->path->limit.fragment = *frag;
```

Then the interface identifier is stored:

```
<store identifier of interface 'interf'> =
  handle->path->limit_id = interface_get_id(interf);
```

Finally, the type of the path ending is stored:

```
<store the path ends up at an interface> =
  handle->path->end_type = SDIS_GREEN_PATH_END_AT_INTERFACE;
```

The other two functions are built using the same pattern and are sketched thereafter:

```
  <store path's end in a medium> =
  res_T
  green_path_set_limit_interface_fragment
    (struct green_path_handle* handle,
     struct sdis_medium* mdm,
     const struct sdis_rwalk_vertex* vert,
     const double elapsed_time)
  {
    res_T res = RES_OK;
    <check input arguments>
    <register medium 'mdm' against the green function>
    <store path duration>
    <store the location in medium>
    <store identifier of medium 'mdm'>
    <store the path ends up in a medium>
    return RES_OK;
  }

  <store radiative path's end> =
  res_T
  green_path_set_limit_interface_fragment
    (struct green_path_handle* handle,
     const double elapsed_time)
  {
    res_T res = RES_OK;
    <check input arguments>
    <store path duration>
    <store the path ends up radiative>
    return RES_OK;
  }
```

5.3.2. Accumulating data along a path

The data accumulated along a path is volume power density terms and flux density terms. It is performed through the following two functions, that share the same pattern:

```
<functions that accumulate data along a path> =
  <register a power term>
  <register a flux term>

<register a power term> =
  res_T
  green_path_add_power_term
    (struct green_path_handle* handle,
     struct sdis_medium* mdm,
     const struct sdis_rwalk_vertex* vtx,
     const double val)
  {
    <local variables>
    <check input arguments>
    <register medium 'mdm' against the green function>
    <search for a power term associated to 'mdm'>
    <if a power term exist for 'mdm'> {
      <add 'val' to this power term>
    } else {
      <register 'val' as the power term of 'mdm'>
    }
```

```
    <finalize the add_power_term function>
  }

<register the accumulated flux term> =
  res_T
  green_path_add_flux_term
    (struct green_path_handle* handle,
     struct sdis_interface* interf,
     const struct sdis_interface_fragment* frag,
     const double val)
  {
    <local variables>
    <check input arguments>
    <register the interface 'interf' against the green function>
    <search for a flux term associated to 'interf'>
    <if a flux term exist for 'interf'> {
      <add 'val' to this flux term>
    } else {
      <register 'val' as the flux term of 'interf'>
    }
    <finalize the add_flux_term function>
  }
```

Examples of Green paths vs. Monte Carlo paths. Two examples of paths sampled by the *Stardis* probe temperature solver are shown in Figs. 5 and 6. For each example, the archived information for the construction of the Green path is shown in Tabs. 1 and 2.

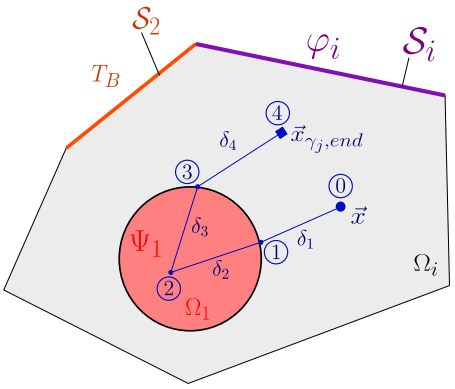

Figure 5: First path example

Step number	Path position	$\beta_\psi(\vec{x}_{\gamma,\psi}(1))$ value	$\mu_{\gamma_j,\text{end}}$ identifier
1	$\vec{x}_{\gamma_j} \in \Omega_i$	0	-
2	$\vec{x}_{\gamma_j} \in \Omega_1$	$\frac{\delta_2^2}{6\lambda_1}$	-
3	$\vec{x}_{\gamma_j} \in \Omega_1$	$\frac{\delta_2^2}{6\lambda_1} + \frac{\delta_3^2}{6\lambda_1}$	-
4	$\vec{x}_{\gamma_j,\text{end}} \in \Omega_i$	$\frac{\delta_2^2}{6\lambda_1} + \frac{\delta_3^2}{6\lambda_1}$	0

Table 1: Successive data stored (framed in green rectangles and referred as *green path*) during Monte Carlo calculation for the first Green path.

The first path starts at the probe point $\vec{x}$. Contributions $\beta_\psi(\vec{x}_{\gamma,\psi}(1))$ related to the power density term Ψ_1 in the medium Ω_1 will accumulate when the path crosses the medium, i.e. during the second and third jump (δ_2 and δ_3). Path ends with the initial temperature condition T_1 associated to the medium Ω_i (see Fig. 5).

Monte Carlo weight would be as follows:

$$w_{\gamma_j} = \psi_1 \beta_\psi(\vec{x}_{\gamma,\psi}(1)) + T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}, \vec{\omega}_{\gamma,\text{end}}, 0) \tag{41}$$

Data stored for the Green path (framed in green in Tab. 1) are: successive positions $\vec{x}_{\gamma_j}$, power density term contribution $\beta_\psi(\vec{x}_{\gamma,\psi}(1))$ along the path and the identifier of the boundary condition encountered $\mu_{\gamma,\text{end}} = 0$ ($T_{i,1}$: initial condition *type-0*).

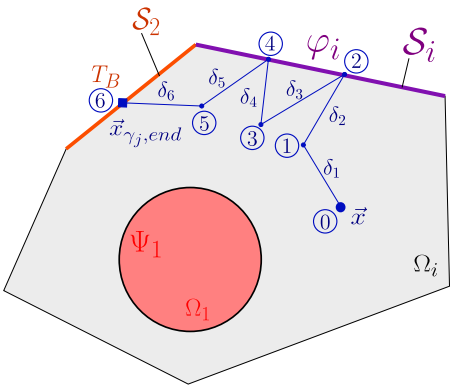

Figure 6: Second path example

Step number	Path position	$\beta_\varphi(\vec{x}_{\gamma,\varphi}(i))$ value	$\mu_{\gamma_j,\text{end}}$ identifier
1	$\vec{x}_{\gamma_j} \in \Omega_i$	0	-
2	$\vec{x}_{\gamma_j} \in \mathcal{S}_i$	0	-
3	$\vec{x}_{\gamma_j} \in \Omega_i$	$\frac{\delta_2^2}{\lambda_i}$	-
4	$\vec{x}_{\gamma_j} \in \mathcal{S}_i$	$\frac{\delta_2^2}{\lambda_i}$	-
5	$\vec{x}_{\gamma_j} \in \Omega_i$	$\frac{\delta_2^2}{\lambda_i} + \frac{\delta_4^2}{\lambda_i}$	-
6	$\vec{x}_{\gamma_j,\text{end}} \in \mathcal{S}_2$	$\frac{\delta_2^2}{\lambda_i} + \frac{\delta_4^2}{\lambda_i}$	1

Table 2: Successive data stored (framed in green rectangles and referred as *green path*) during Monte Carlo calculation for the second Green path.

The second path starts at the probe point $\vec{x}$. Contributions $\beta_\varphi(\vec{x}_{\gamma,\varphi}(i))$ related to flux density φ_i at $\mathcal{S}_i$ interface will accumulate when the path hits this interface, i.e. on the second and fourth jump (δ_2 et δ_4). The path ends at the interface $\mathcal{S}_2$ with the temperature imposed on this interface T_B (see Fig. 6).

Monte Carlo weight would be as follows:

$$w_{\gamma_j} = \varphi_i \beta_\varphi(\vec{x}_{\gamma,\varphi}(i)) + T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}, \vec{\omega}_{\gamma,\text{end}}, 1) \tag{42}$$

Data stored for the Green path (framed in green in Tab. 2) are: successive positions $\vec{x}_{\gamma_j}$, density flux contribution $\beta_\varphi(\vec{x}_{\gamma,\varphi}(i))$ along the path and the identifier of the boundary condition encountered $\mu_{\gamma,\text{end}} = 1$ (T_B : *type-1* boundary condition).

5.4. Using propagation information

Once a set of sampled paths is stored in the Green function, dedicated evaluation functions are needed to apply the Green function to a set of source values (see List. 5). These functions are presented thereafter. Note however that when using helper functions also used for standard Monte-Carlo computations (`solid_get_volumic_power`, `interface_side_get_flux`), they have to provide a vertex, even though the computation is time and space independent in the Green function context.

```
<evaluation functions> =
    res_T
    sdis_green_function_solve
      (struct sdis_green_function* green,
       struct sdis_estimator** out_estimator)
    {
      <local variables>
      <check input arguments>
      <create the estimator>

      <for each green path stored into 'green'> {
        <compute the weight of the current green path>
        <accumulate the resulting weight>
      }
      <setup the estimator>
      <finalize the solve function>
    }
```

The computation of the weight associated to a path is done with a dedicated function (see List. 4):

```
<compute the weight of the current green path> =
  double w; /* Monte Carlo weight to compute */
  res = green_function_solve_path(green, ipath, &w);
  <handle error code returned in 'res'>
```

The dedicated function simply takes into account the different contributions that have been stored in the current path (power, flux and end of path), and uses them to produce the Monte-Carlo weight of the path:

```
<evaluation functions> +=
  res_T
  green_function_solve_path
    (struct sdis_green_function* green,
     const size_t ipath,
     double* weight)
  {
    <local variables>
    <check input arguments>

    <compute the power collected along the path>
    <compute the flux collected along the path>
    <fetch the end of path>
    <compute the overall Monte-Carlo weight>

    <finalize the solve_path fuction>
  }
```

The volume power along the path is computed by considering each medium encountered along the path and accumulating the corresponding volume power contribution. Each volume power term `power_terms[i]` (see Eq. (36)), previously accumulated by the function `green_path_add_power_term` is multiplied by the new given volume power density value `solid_get_volumic_power(medium, &vtx)`. The total volume power contribution obtained for the path is `power`:

```
<Compute the volume power collected along the path> =
  power = 0;
  n = darray_power_term_size_get(&path->power_terms);
  power_terms = darray_power_term_cdata_get(&path->power_terms);

  FOR_EACH(i, 0, n) {
    vtx.time = INF;
    medium = green_function_fetch_medium(green, power_terms[i].id);
    power += power_terms[i].term * solid_get_volumic_power(medium, &vtx);
  }
```

The flux along the path is computed by considering each interface encountered along the path and accumulating the corresponding flux contribution.

Each flux terms `flux_terms[i]` (see Eq. (36)), previously accumulated by the function `green_path_add_flux_term` is multiplied by the new given flux density value `interface_side_get_flux(interf, &frag)`. The flux term obtained for the path is `flux`:

```
<Compute the flux collected along the path> =
  flux = 0;
  n = darray_flux_term_size_get(&path->flux_terms);
  flux_terms = darray_flux_term_cdata_get(&path->flux_terms);

  FOR_EACH(i, 0, n) {
    frag.time = INF;
```

```
    frag.side = flux_terms[i].side;
    interf = green_function_fetch_interf(green, flux_terms[i].id);
    flux += flux_terms[i].term * interface_side_get_flux(interf, &frag);
  }
```

The temperature at the end of the path, depending on the type of end, is the new given interface temperature value `interface_side_get_temperature(interf, &frag)`, the new given temperature value of the medium `medium_get_temperature(interf, &frag)`, or the new given ambient radiative temperature value `sdis_scene_get_ambient_radiative_temperature(scn, &end_temperature)`:

```
<fetch the end of path> =
  switch(path->end_type) {
    case SDIS_GREEN_PATH_END_AT_INTERFACE:
      interf = green_function_fetch_interf(green, path->limit_id);
      frag = path->limit.fragment;
      end_temperature = interface_side_get_temperature(interf, &frag);
      break;
    case SDIS_GREEN_PATH_END_IN_VOLUME:
      medium = green_function_fetch_medium(green, path->limit_id);
      vtx = path->limit.vertex;
      end_temperature = medium_get_temperature(medium, &vtx);
      break;
    case SDIS_GREEN_PATH_END_RADIATIVE:
      SDIS(green_function_get_scene(green, &scn));
      SDIS(scene_get_ambient_radiative_temperature(scn, &end_temperature));
      if(end_temperature <  0) { /* Cannot be negative if used */
        res = RES_BAD_ARG;
        goto error;
      }
      break;
    default: FATAL("Unreachable code.\n"); break;
  }
```

Path weight `weight`, computed from the new source values, is then simply the sum of the different contributions: `power`, `flux` and `end_temperature`.

```
<compute the overall Monte-Carlo weight> =
  *weight = power + flux + end_temperature;
```

6. Simulation examples

This section illustrates a typical use of the storage and use of the propagation information described in the previous sections. The geometrical and physical descriptions of the configurations used for this illustration, as well as the *Stardis* input files, are available in the enclosed zip file. The objective is essentially to show that a computation performed using the stored propagation information recovers the result that would be obtained with a complete MC run (with a particular attention to the associated statistical errorbars) and to illustrate the benefits in terms of computation times. As far as validation is concerned, we concentrate on the parts of the code constructing and using the propagators, not on the main code itself: *Stardis* is already validated elsewhere [2, 3]. However we still reproduce here parts of this validation by providing, together with each simulation example, a systematic comparison with the solution computed with a standard deterministic solver [4] (referred as *COMSOL Multiphysics*® hereafter and "Deterministic" in figures).

Two academic configurations are considered. They are designed as simplified versions of porous media, one with open porosities (see Fig. C.8 a)), the other with closed porosities (see Fig. C.8 c)). This benchmark was already used by Sans et. al [5] for the purpose of validating Monte Carlo simulations of coupled conduction-radiation heat transfer.

The open-porosity configuration corresponds to a heterogeneous 3D honeycomb that can be assimilated to a porous medium with open channels, like a heat exchanger configuration. The closed-porosity configuration has 22 enclosed cavities and may be assimilated to an insulation material.

The physical assumptions are those of Sec. 2.1 with same values for λ, ρ, c and ψ throughout the whole solid phase and same values for h and ϵ along all the solid-fluid interfaces. For open porosity, there is one single imposed fluid temperature T_{BF} (*type-3* boundary condition), the same inside the channels and outside the system. For closed porosity, T_{BF} is only imposed outside the system (fluid cells temperatures are free). In both cases, the incoming radiance temperature outside the system θ_{BR} is uniform and isotropic, the solid temperature T_{B} is imposed at the top surface (*type-1* boundary condition) and a flux density φ_B is imposed at the bottom surface (*type-2* boundary condition). The initial temperature T_{I} is uniform.

The estimated quantity is the temperature $T(\vec{x}_c, t)$ at the center $\vec{x}_c$ of the geometry for a given observation time t as a function depending on the six available sources:

- the initial temperature T_{I},
- the top boundary solid temperature T_{B},
- the ambient fluid temperature T_{BF},
- the ambient radiance temperature θ_{BR},
- the flux density at the bottom boundary φ,
- the power density throughout the solid phase ψ.

Tests are conducted first without radiation ($\epsilon = 0$, see Fig. C.9, Fig. C.10 and Fig. C.11) and then with radiation (black surfaces, $\epsilon = 1$, see Fig. C.12, Fig. C.13 and Fig. C.14). For each case, the propagation information are stored using a single Monte Carlo run. These propagation information are then used to predict $T(\vec{x}_c, t)$ (and to estimate its uncertainty) when varying the sources values with factors in the $[10^{-2}, 10^{2}]$ range around a fixed reference value for each source: $T_{\mathrm{I}}^{\mathrm{ref}}$, $T_{\mathrm{B}}^{\mathrm{ref}}$, $T_{\mathrm{BF}}^{\mathrm{ref}}$, $\theta_{\mathrm{BR}}^{\mathrm{ref}}$, φ^{ref} and ψ^{ref} (results labeled "Propagator" in the figures). Validation is achieved with the comparison of standard Monte Carlo results labeled "Monte-Carlo" in the figures. The perfect adequacy between the "Propagator" and "Monte-Carlo" results in Figs. C.9, C.10,C.11 C.12, C.13 and C.14 validates the implemented code and the quality of the stored propagation information. Fig. 7 gathers all the computation times, illustrating that the benefits of using the stored propagation information, instead of running the Monte Carlo, is a computation time reduction of the order 10^{-3} to 10^{-4} [6].

In closer details, the following comments can be made:

- "Propagator" (using the propagator) and "Monte-Carlo" results (re-running a Monte Carlo for control) are in perfect agreement, as expected, because "Propagator" is statistically rigorously equivalent to re-running the Monte Carlo (see Fig. C.9 a) and Fig. C.10 a)). However, for the validation runs, new random numbers are used to sample the paths, whereas all "Propagator" simulations are based on the same sampled paths. Therefore, although the errorbars associated to "Propagator" can be fully trusted, the simulations made for various values of the sources are all correlated: typically, there are no statistical fluctuations in the errorbars, and when a simulation result for a given source value happens to be below the reference (within the errorbar, otherwise the validation would have failed), it remains below the reference for all other values of the source.

- "Propagator" (or "Monte-Carlo") and "Deterministic" are in perfect agreement as long as the linearization of heat transfer remains relevant. It is well-known that, exchanged radiative heat flux being proportional to T^4, radiative heat transfer causes non-linear propagation. Moreover, the higher thermal gradients are, the higher such non-linear effects occur and the larger the bias induced by radiative transfer linearization. Here, without radiation and/or any thermal dependance of the thermal properties, "Propagator" predict the correct temperature

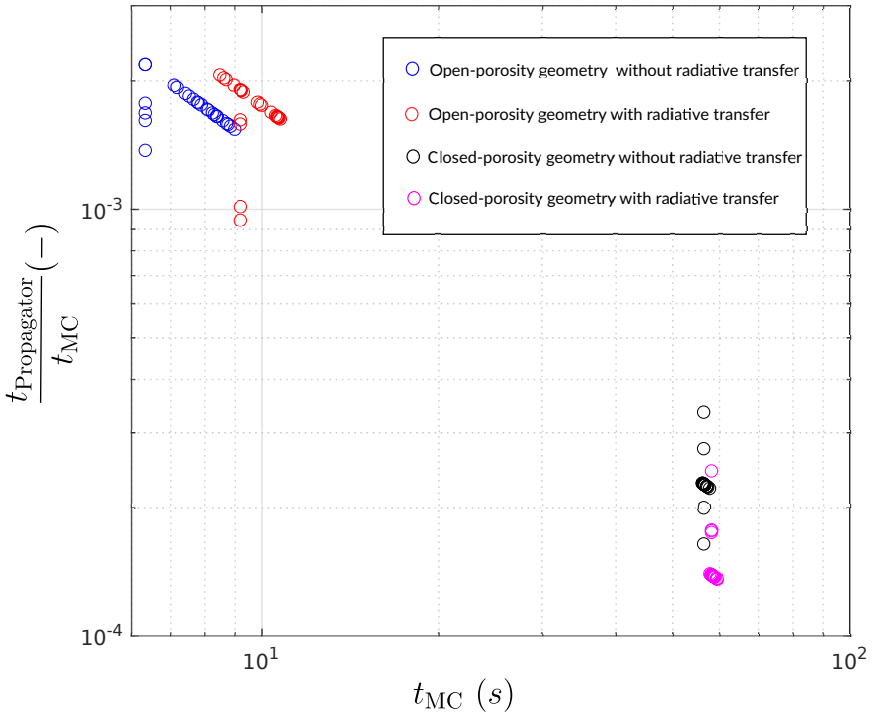

Figure 7: Evolution of the ratio of computation times $\frac{t_{\mathrm{Propagator}}}{t_{\mathrm{MC}}}$ for the corresponding t_{MC}. $t_{\mathrm{Propagator}}$ is the computation time for the propagator function and t_{MC} is the computation time for the corresponding Monte Carlo computation.

for a very wide range of sources values (see Fig. C.9, Fig. C.10 and Fig. C.11). Close to the set of reference values for sources, temperature gradients were purposely chosen as low as suitable for the frame of linear heat transfer. Thus, "Propagator" and "Deterministic" are in good agreement (see Fig. C.12 **b)**). Outside this range of source values, "Propagator" fails to predict the correct temperature field because the linearization of radiation, at the heart of *Stardis*, becomes meaningless (see Fig. C.12 **a)**). Here, two different physical models are solved : *Stardis* linearizes radiation where *COMSOL Multiphysics*® does not. Hence, the gaps observed between the "Propagator" and the "Deterministic" method come from the capacity of a given source to increase thermal gradients, and thus to strengthen non-linearity effects (see Fig. C.12, Fig. C.13 and Fig. C.14). We are presently working on this issue, starting from the non-linear Monte Carlo approach given by [7]).

- There is a notable distinction between open-porosity and closed-porosity as far as computation time is concerned. In the open-porosity case, a thermal path starting at the center of the system can encounter the flow quite rapidly and then the path is stopped because the fluid temperature is known. In the closed-porosity case, a thermal path starting at the center of the system can also encounter the flow inside a fluid cell quite rapidly, but then the flow temperature is unknown and the path is continued until a source is encountered (either the initial condition at any location, or a known temperature at the boundary). These paths are significantly longer and so is the computation time required for their construction. It can even be extremely long if the number of closed cells is increased: Monte Carlo may encounter difficulties when addressing insulating materials with large numbers of closed cells along all directions. However this difficulty is not reported in "Propagator": even if the computation cost associated to path-sampling in the reference Monte Carlo run is higher in the closed-porosity case, the propagation information stored are similar in the open-porosity and the closed-porosity cases and the computation times associated to the use, by "Propagator", of these propagation information are the same in both cases. The computation benefit of using "Propagator" instead of running a full Monte Carlo is therefore stronger for closed porosities.

7. Going further: Propagation information storage in case of non-uniform and time-dependent sources

When constructing estimates for the propagators, we have split the initial Monte Carlo weight and gathered all the propagation information corresponding to parts of the system where the sources were uniform and constant (see Sec. 4). As was already mentioned, *Stardis* makes uniformity and constancy assumptions for each geometrical part but ‘`stardis-solver`‘ does not. And indeed, in the Monte Carlo weight expression of Eq. (31), nothing prevents the initial temperatures, the imposed temperatures at the boundary, the incoming radiance temperature, the surface flux densities imposed at the boundary and the volume power densities imposed inside the solids to be non-uniform and non-constant fields: $\psi\left(\vec{x}_{\gamma,\psi}(k), t_{\gamma,\psi}(k)\right)$, $\varphi\left(\vec{x}_{\gamma,\varphi}(k), t_{\gamma,\varphi}(k)\right)$ and $T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}}, \mu_{\gamma,\mathrm{end}})$ can hold these informations as they are provided, whatever their geometric structure. This means that even running *Stardis* with uniform and constant sources in each part, if we store the locations, times and directions used when constructing Monte Carlo weights, then these informations can be later used to virtually re-run the Monte Carlo simulation with non-uniform time-dependent sources.

In algorithmic terms:

- N thermal paths are sampled exactly as in the Monte Carlo algorithm of Sec. 3.1;
- Along each path γ_j, we store the $N_{\varphi,j}$ locations $\vec{x}_{\gamma_j,\varphi}(k)$ and times $t_{\gamma_j,\varphi}(k)$ at which surface flux densities were accessed;
- Similarly we store the $N_{\psi,j}$ locations $\vec{x}_{\gamma_j,\psi}(k)$ and times $t_{\gamma_j,\psi}(k)$ at which volume power densities were accessed;
- The information concerning the end of the path are also stored: $\vec{x}_{\gamma_j,\mathrm{end}}$, $t_{\gamma_j,\mathrm{end}}$, $\vec{\omega}_{\gamma_j,\mathrm{end}}$ and $\mu_{\gamma_j,\mathrm{end}}$;
- When a Monte Carlo estimate is required for any set of non-uniform time-dependent source fields $\tilde{T}_{\mathrm{I}}$, $\tilde{T}_{\mathrm{B}}$, $\tilde{T}_{\mathrm{BF}}$, $\tilde{\theta}_{\mathrm{BR}}$, $\tilde{\psi}$ and $\tilde{\varphi}$, it is constructed the following way:

$$\tilde{T}(\vec{x},t) \text{ or } \tilde{T}(t) \text{ or } \tilde{\theta}_R(\vec{\omega},\vec{x},t) \approx \tilde{m} = \frac{1}{N}\sum_{j=1}^{N} \tilde{w}_j \tag{43}$$

with

$$\begin{aligned}\tilde{w}_j = \; & \mathcal{H}(\mu_{\gamma_j,\text{end}} = 0)\, \tilde{T}_\text{I}(\vec{x}_{\gamma_j,\text{end}}) \\ & + \mathcal{H}(\mu_{\gamma_j,\text{end}} = 1)\, \tilde{T}_\text{B}(\vec{x}_{\gamma_j,\text{end}}, t_{\gamma_j,\text{end}}) \\ & + \mathcal{H}(\mu_{\gamma_j,\text{end}} = 2)\, \tilde{T}_\text{BF}(\vec{x}_{\gamma_j,\text{end}}, t_{\gamma_j,\text{end}}) \\ & + \mathcal{H}(\mu_{\gamma_j,\text{end}} = 3)\, \tilde{\theta}_\text{BR}(\vec{x}_{\gamma_j,\text{end}}, t_{\gamma_j,\text{end}}, \vec{\omega}_{\gamma_j,\text{end}}) \\ & + \sum_{k=1}^{N_{\psi,j}} \beta_\psi(\vec{x}_{\gamma_j,\psi}(k))\, \tilde{\psi}\left(\vec{x}_{\gamma_j,\psi}(k), t_{\gamma_j,\psi}(k)\right) \\ & + \sum_{k=1}^{N_{\varphi,j}} \beta_\varphi(\vec{x}_{\gamma_j,\varphi}(k))\, \tilde{\varphi}\left(\vec{x}_{\gamma_j,\varphi}(k), t_{\gamma_j,\varphi}(k)\right)\end{aligned} \tag{44}$$

For this last strategy, dealing with uncertainties is straightforward. The estimate $\tilde{m}$ in Eq. (43) is finally built exactly as if the Monte Carlo was re-run using new sources. The uncertainty $\tilde{s}$ associated to $\tilde{m}$ is therefore the same as for any Monte Carlo simulation (using an estimate of the standard deviation of the Monte Carlo weights) as detailed in Eq. (15).

In terms of code changes, storing unmerged flux and power terms allows code simplification, as a significant part of the merged-storage version of the code is about retrieving the term to which the current contribution needs to be merged. On the other hand, allowing all types of non-uniform and time-dependent sources is still at the cost of storing more propagation since information terms data structures have to store the term's location (time and space). The data structures and functions that implement the unmerged storage of sources' contributions are available in Appendix B at the end of the document.

8. Conclusion

One of the strengths of Monte Carlo approaches is the ease with which information can be stored, during one run, and then used offline to learn about the physics, preserving all geometric features. The simplest example is the storing of the paths themselves (or of a large enough fraction of the paths). For coupled heat transfer, displaying a selection of the paths and analyzing how they visit the system, both in time and space, switching from one heat transfer mode to the other, is indeed a very practical way to learn how the sources are viewed from one location, how their impact is delayed by the inertial parts of the system, and therefore how a design can be adjusted to achieve a given objective. In the present paper, we have left aside these details about the paths themselves. We concentrated on the act of quantifying the propagation and not on the analysis of the physical phenomena and the coupling processes responsible for the propagation. But these two practices, computing the propagators and visualizing the propagation processes, are worth being considered sideways in all engineering contexts requiring a close understanding of heat transfer physics at the system scale. Therefore, in addition to the functionalities of *Stardis* described in Sec. 5, a set of post-treatment tools have also been designed to help visualizing and analyzing thermal paths throughout large scale geometries [3].

For automated engineering practices, e.g. inversion, optimization or command algorithms, analyzing the paths is useless; all is needed is the addressed quantity as a function of the sources: the tools described in the present paper are therefore self-sufficient. However there are numerous questions of direct interest to thermal engineers that cannot be addressed this way. These are all the dependencies on parameters that cannot be considered as sources (in the general sense provided by Green's theory). Typically the dependence on emissivities, convective exchange coefficients, conductivities or capacities rises more complex questions. If only sensitivities were required, i.e. the derivative with respect to each parameter, then the general theory of sensitivity evaluation in Monte Carlo algorithms could be used [8], but we would not build the complete dependence (the function) as we did here with the sources. Addressing the complete non-linear dependence on other parameters than sources is not theoretically unfeasible: it has notably been achieved in the field of radiative transfer under the literature name of "Symbolic Monte Carlo" [9, 10, 11, 12] and we have started to work on extending these symbolic techniques to coupled heat transfer, with the objective of implementing them inside `stardis-solver` [13, 14].

By far more difficult would be the question of addressing the dependence on geometrical parameters. Here also, some attempts have already been made in the field of radiative transfer, but to the best of our knowledge and although large impacts could be expected in terms of applications, there is no report available of any attempt to go beyond the computation of derivatives (geometrical sensitivities). Constructing a thermal heat transfer observable as a symbolic function of a geometric parameter is a question that has not yet been addressed.

Another difficult point associated to strong applicative concerns is the withdraw of the linearization of radiative transfer. This linearization is at the heart of present Monte Carlo approaches to coupled heat transfer. There are convincing perspectives as far as handling non-linearities in the Monte Carlo framework is concerned [7], and some of the corresponding propositions could be used to avoid the linearization of radiation, but then the overall coupled physical problem would be non-linear and the concept of propagation could not be used anymore. All our present proposition would then have to be revisited.

Acknowledgements

This project has received funding from the "Investissement d'avenir" program of the National Agency for Research of the French state under award number "ANR-10-LBX-22-01-SOLSTICE" and was supported by the ANR HIGH-TUNE, grant ANR-16-CE01-0010 and ANR MCG-RAD, grant ANR-18-CE46-0012 and from the Occitanie Region (Projet CLE-2016 ED-Star).

References

[1] D. Knuth, Literate Programming, The Computer Journal 27 (1984) 97–111.

[2] V. Eymet, F. Vincent, B. Piaud, C. Coustet, R. Fournier, S. Blanco, L. Ibarrart, J.-M. Tregan, P. Lavieille, C. Caliot, M. El-Hafi, J.-J. Bézian, R. Bouchie, M. Galtier, M. Roger, J. Dauchet, O. Farges, C. Péniguel, I. Rupp, G. Eymet, Synthèse d'images infrarouges sans calcul préalable du champ de température, in: SFT 2019 - 27ème Congrès Français de Thermique, Nantes, France, 2019, pp. 153–160.
URL https://hal.archives-ouvertes.fr/hal-02419604

[3] Méso-Star, Stardis (2021).
URL https://www.meso-star.com/projects/stardis/starter-pack.html

[4] Comsol multiphysics®, v. 5.6. (2020).
URL http://www.comsol.com

[5] M. Sans, O. Farges, V. Schick, C. Moyne, G. Parent, Modeling the Flash Method by using a Conducto-Radiative Monte-Carlo Method: Application to Porous Media, in: Proceeding of Proceedings of the 9th International Symposium on Radiative Transfer, RAD-19, Begellhouse, Athens, Greece, 2019, pp. 319–326. doi:10.1615/RAD-19.390.

[6] L. Penazzi, S. Blanco, C. Caliot, C. Coustet, M. El-Hafi, R. Fournier, J. Gautrais, M. Sans, Transfer function estimation with SMC method for combined heat transfer: insensitivity to detail refinement of complex geometries, in: CHT-21 ICHMT - International Symposium on Advances in Computational Heat Transfer, Rio de Janeiro (online), Brazil, 2021, pp. 383–386.
URL https://hal-mines-albi.archives-ouvertes.fr/hal-03374353

[7] J. Dauchet, J.-J. Bezian, S. Blanco, C. Caliot, J. Charon, C. Coustet, M. El Hafi, V. Eymet, O. Farges, V. Forest, R. Fournier, M. Galtier, J. Gautrais, A. Khuong, L. Pelissier, B. Piaud, M. Roger, G. Terrée, S. Weitz, Addressing nonlinearities in Monte Carlo, Scientific Reports 8 (1) (Dec. 2018). doi:10.1038/s41598-018-31574-4.

[8] P. Lapeyre, S. Blanco, C. Caliot, J. Dauchet, M. El Hafi, R. Fournier, O. Farges, M. Roger, Monte-Carlo and domain-deformation sensitivities, in: Proceeding of Proceedings of the 9th International Symposium on Radiative Transfer, RAD-19, Begellhouse, Athens, Greece, 2019, pp. 213–220. doi:10.1615/RAD-19.260.

[9] W. L. Dunn, Inverse Monte Carlo analysis, Journal of Computational Physics 41 (1) (1981) 154–166.

[10] W. L. Dunn, J. K. Shultis, Monte Carlo methods for design and analysis of radiation detectors, Radiation Physics and Chemistry 78 (10) (2009) 852–858. doi:10.1016/j.radphyschem.2009.04.030.

[11] M. Galtier, M. Roger, F. André, A. Delmas, A symbolic approach for the identification of radiative properties, Journal of Quantitative Spectroscopy and Radiative Transfer 196 (2017) 130–141. doi:10.1016/j.jqsrt.2017.03.026.

[12] Y. Maanane, M. Roger, A. Delmas, M. Galtier, F. André, Symbolic Monte Carlo method applied to the identification of radiative properties of a heterogeneous material, Journal of Quantitative Spectroscopy and Radiative Transfer 249 (2020) 107019. doi:10.1016/j.jqsrt.2020.107019.

[13] L. Penazzi, S. Blanco, C. Caliot, C. Coustet, M. El-Hafi, R. A. Fournier, M. Galtier, L. Ibarrart, M. Roger, Toward the use of Symbolic Monte Carlo for Conduction-Radiation Coupling in Complex Geometries, in: RAD-19 - 9th International Symposium on Radiative Transfer, Begellhouse, Athens, Greece, 2019, p. 8 p. doi:10.1615/RAD-19.380.
URL https://hal-mines-albi.archives-ouvertes.fr/hal-02265075

[14] M. Sans, S. Blanco, C. Caliot, M. El-Hafi, O. Farges, R. A. Fournier, L. Penazzi, Méthode de Monte-Carlo Symbolique pour la caractérisation des propriétés thermiques : application à la méthode flash, in: SFT 2021 - 29 ème congrès Franças de Thermique, Belfort (online), France, 2021, pp. 293–300. doi:10.25855/SFT2021-076.
URL https://hal-mines-albi.archives-ouvertes.fr/hal-03260534

Appendix A. Source code chunks implementing merged storage

Listing 1: struct sdis_green_function and related types

```
struct sdis_green_function {
  struct htable_medium media;
  struct htable_interf interfaces;
  struct darray_green_path paths; /* List of paths used to estimate the green */

  size_t npaths_valid;
  size_t npaths_invalid;

  struct accum realisation_time; /* Time per realisation */

  struct ssp_rng_type rng_type;
  FILE* rng_state;

  ref_T ref;
  struct sdis_scene* scn;
};
```

Listing 2: struct green_path

```
struct green_path {
  double elapsed_time;
  struct darray_flux_term flux_terms; /* List of flux terms */
  struct darray_power_term power_terms; /* List of volumic power terms */
  union {
    struct sdis_rwalk_vertex vertex;
    struct sdis_interface_fragment fragment;
  } limit;
  unsigned limit_id; /* Identifier of the limit medium/interface */
  enum sdis_green_path_end_type end_type;

  /* Indices of the last accessed medium/interface. Used to speed up the access
   * to the medium/interface. */
  uint16_t ilast_medium;
  uint16_t ilast_interf;
};

struct power_term {
double term;
unsigned id; /* Identifier of the medium */
};

struct flux_term {
double term;
unsigned id; /* Identifier of the interface */
enum sdis_side side;
};
```

Listing 3: green functions to store path data

```
res_T
green_path_set_limit_interface_fragment
  (struct green_path_handle* handle,
   struct sdis_interface* interf,
   const struct sdis_interface_fragment* frag,
   const double elapsed_time)
{
  res_T res = RES_OK;
  ASSERT(handle && interf && frag);
  ASSERT(handle->path->end_type == SDIS_GREEN_PATH_END_TYPES_COUNT__);
  res = ensure_interface_registration(handle->green, interf);
  if(res != RES_OK) return res;
```

```
  handle->path->elapsed_time = elapsed_time;
  handle->path->limit.fragment = *frag;
  handle->path->limit_id = interface_get_id(interf);
  handle->path->end_type = SDIS_GREEN_PATH_END_AT_INTERFACE;
  return RES_OK;
}

res_T
green_path_set_limit_vertex
  (struct green_path_handle* handle,
   struct sdis_medium* mdm,
   const struct sdis_rwalk_vertex* vert,
   const double elapsed_time)
{
  res_T res = RES_OK;
  ASSERT(handle && mdm && vert);
  ASSERT(handle->path->end_type == SDIS_GREEN_PATH_END_TYPES_COUNT__);
  res = ensure_medium_registration(handle->green, mdm);
  if(res != RES_OK) return res;
  handle->path->elapsed_time = elapsed_time;
  handle->path->limit.vertex = *vert;
  handle->path->limit_id = medium_get_id(mdm);
  handle->path->end_type = SDIS_GREEN_PATH_END_IN_VOLUME;
  return RES_OK;
}

res_T
green_path_set_limit_radiative
  (struct green_path_handle* handle,
   const double elapsed_time)
{
  ASSERT(handle);
  ASSERT(handle->path->end_type == SDIS_GREEN_PATH_END_TYPES_COUNT__);
  handle->path->elapsed_time = elapsed_time;
  handle->path->end_type = SDIS_GREEN_PATH_END_RADIATIVE;
  return RES_OK;
}

res_T
green_path_add_power_term
  (struct green_path_handle* handle,
   struct sdis_medium* mdm,
   const struct sdis_rwalk_vertex* vtx,
   const double val)
{
  struct green_path* path;
  struct power_term* terms;
  size_t nterms;
  size_t iterm;
  unsigned id;
  res_T res = RES_OK;
  ASSERT(handle && mdm && vtx);

  /* Unused position and time: the current implementation of the green function
   * assumes that the power is constant in space and time per medium. */
  (void)vtx;

  res = ensure_medium_registration(handle->green, mdm);
  if(res != RES_OK) goto error;

  path = handle->path;
  terms = darray_power_term_data_get(&path->power_terms);
  nterms = darray_power_term_size_get(&path->power_terms);
  id = medium_get_id(mdm);
  iterm = SIZE_MAX;
```

```
  /* Early find */
  if(path->ilast_medium < nterms && terms[path->ilast_medium].id == id) {
    iterm = path->ilast_medium;
  } else {
    /* Linear search of the submitted medium */
    FOR_EACH(iterm, 0, nterms) if(terms[iterm].id == id) break;
  }

  /* Add the power term to the path wrt the submitted medium */
  if(iterm < nterms) {
    terms[iterm].term += val;
  } else {
    struct power_term term = POWER_TERM_NULL__;
    term.term = val;
    term.id = id;
    res = darray_power_term_push_back(&handle->path->power_terms, &term);
    if(res != RES_OK) goto error;
  }

  /* Register the slot into which the last accessed medium lies */
  CHK(iterm < UINT16_MAX);
  path->ilast_medium = (uint16_t)iterm;

exit:
  return res;
error:
  goto exit;
}

res_T
green_path_add_flux_term
  (struct green_path_handle* handle,
   struct sdis_interface* interf,
   const struct sdis_interface_fragment* frag,
   const double val)
{
  struct green_path* path;
  struct flux_term* terms;
  size_t nterms;
  size_t iterm;
  unsigned id;
  res_T res = RES_OK;
  ASSERT(handle && interf && frag && val >= 0);

  res = ensure_interface_registration(handle->green, interf);
  if(res != RES_OK) goto error;

  path = handle->path;
  terms = darray_flux_term_data_get(&path->flux_terms);
  nterms = darray_flux_term_size_get(&path->flux_terms);
  id = interface_get_id(interf);
  iterm = SIZE_MAX;

  /* Early find */
  if(path->ilast_interf < nterms
  && terms[path->ilast_interf].id == id
  && terms[path->ilast_interf].side == frag->side) {
    iterm = path->ilast_interf;
  } else {
    /* Linear search of the submitted interface */
    FOR_EACH(iterm, 0, nterms) {
      if(terms[iterm].id == id && terms[iterm].side == frag->side) {
        break;
      }
```

```
    }
  }

  /* Add the flux term to the path wrt the submitted interface */
  if(iterm < nterms) {
    terms[iterm].term += val;
  } else {
    struct flux_term term = FLUX_TERM_NULL__;
    term.term = val;
    term.id = id;
    term.side = frag->side;
    res = darray_flux_term_push_back(&handle->path->flux_terms, &term);
    if(res != RES_OK) goto error;
  }

  /* Register the slot into which the last accessed interface lies */
  CHK(iterm < UINT16_MAX);
  path->ilast_interf = (uint16_t)iterm;

exit:
  return res;
error:
  goto exit;
}
```

Listing 4: green_function_solve_path

```
static res_T
green_function_solve_path
  (struct sdis_green_function* green,
   const size_t ipath,
   double* weight)
{
  const struct power_term* power_terms = NULL;
  const struct flux_term* flux_terms = NULL;
  const struct green_path* path = NULL;
  const struct sdis_medium* medium = NULL;
  const struct sdis_interface* interf = NULL;
  struct sdis_scene* scn = NULL;
  struct sdis_rwalk_vertex vtx = SDIS_RWALK_VERTEX_NULL;
  struct sdis_interface_fragment frag = SDIS_INTERFACE_FRAGMENT_NULL;
  double power;
  double flux;
  double end_temperature;
  size_t i, n;
  res_T res = RES_OK;
  ASSERT(green && ipath < darray_green_path_size_get(&green->paths) && weight);

  path = darray_green_path_cdata_get(&green->paths) + ipath;
  if(path->end_type == SDIS_GREEN_PATH_END_ERROR) { /* Rejected path */
    res = RES_BAD_OP;
    goto error;
  }

  /* Compute medium power terms */
  power = 0;
  n = darray_power_term_size_get(&path->power_terms);
  power_terms = darray_power_term_cdata_get(&path->power_terms);
  FOR_EACH(i, 0, n) {
    vtx.time = INF;
    medium = green_function_fetch_medium(green, power_terms[i].id);
    power += power_terms[i].term * solid_get_volumic_power(medium, &vtx);
  }

  /* Compute interface fluxes */
```

```
  flux = 0;
  n = darray_flux_term_size_get(&path->flux_terms);
  flux_terms = darray_flux_term_cdata_get(&path->flux_terms);
  FOR_EACH(i, 0, n) {
    frag.time = INF;
    frag.side = flux_terms[i].side;
    interf = green_function_fetch_interf(green, flux_terms[i].id);
    flux += flux_terms[i].term * interface_side_get_flux(interf, &frag);
  }

  /* Compute path's end temperature */
  switch(path->end_type) {
    case SDIS_GREEN_PATH_END_AT_INTERFACE:
      interf = green_function_fetch_interf(green, path->limit_id);
      frag = path->limit.fragment;
      end_temperature = interface_side_get_temperature(interf, &frag);
      break;
    case SDIS_GREEN_PATH_END_IN_VOLUME:
      medium = green_function_fetch_medium(green, path->limit_id);
      vtx = path->limit.vertex;
      end_temperature = medium_get_temperature(medium, &vtx);
      break;
    case SDIS_GREEN_PATH_END_RADIATIVE:
      SDIS(green_function_get_scene(green, &scn));
      SDIS(scene_get_ambient_radiative_temperature(scn, &end_temperature));
      if(end_temperature < 0) { /* Cannot have it negative if used */
        res = RES_BAD_ARG;
        goto error;
      }
      break;
    default: FATAL("Unreachable code.\n"); break;
  }

  /* Compute the path weight */
  *weight = power + flux + end_temperature;

exit:
  return res;
error:
  goto exit;
}
```

Listing 5: green_function_solve

```
res_T
sdis_green_function_solve
  (struct sdis_green_function* green,
   struct sdis_estimator** out_estimator)
{
  struct sdis_estimator* estimator = NULL;
  size_t npaths;
  size_t ipath;
  size_t N = 0; /* #realisations */
  double accum = 0;
  double accum2 = 0;
  res_T res = RES_OK;

  if(!green || !out_estimator) {
    res = RES_BAD_ARG;
    goto error;
  }

  npaths = darray_green_path_size_get(&green->paths);

  /* Create the estimator */
```

```
  res = estimator_create(green->scn->dev, SDIS_ESTIMATOR_TEMPERATURE, &estimator);
  if(res != RES_OK) goto error;

  /* Solve the green function */
  FOR_EACH(ipath, 0, npaths) {
    double w;

    res = green_function_solve_path(green, ipath, &w);
    if(res == RES_BAD_OP) continue;
    if(res != RES_OK) goto error;

    accum += w;
    accum2 += w*w;
    ++N;
  }

  /* Setup the estimated temperature */
  estimator_setup_realisations_count(estimator, npaths, N);
  estimator_setup_temperature(estimator, accum, accum2);
  estimator_setup_realisation_time
    (estimator, green->realisation_time.sum, green->realisation_time.sum2);

exit:
  if(out_estimator) *out_estimator = estimator;
  return res;
error:
  if(estimator) {
    SDIS(estimator_ref_put(estimator));
    estimator = NULL;
  }
  goto exit;
}
```

Appendix B. Source code chunks implementing unmerged storage

Listing 6: unmerged terms structs

```
struct unmerged_power_term {
  double term;
  unsigned id; /* Identifier of the medium */
  struct sdis_rwalk_vertex vertex; /* location of the term */
};

struct unmerged_flux_term {
  double term;
  unsigned id; /* Identifier of the interface */
  struct sdis_interface_fragment fragment; /* location of the term */
};
```

Listing 7: green functions to store unmerged path data

```
res_T
green_path_add_power_term
  (struct green_path_handle* handle,
   struct sdis_medium* mdm,
   const struct sdis_rwalk_vertex* vtx,
   const double val)
{
  struct green_path* path;
  struct unmerged_power_term* terms;
  struct power_term term = POWER_TERM_NULL__;
  size_t nterms;
  unsigned id;
  res_T res = RES_OK;
  ASSERT(handle && mdm && vtx);
```

```
  res = ensure_medium_registration(handle->green, mdm);
  if(res != RES_OK) goto error;

  path = handle->path;
  terms = darray_power_term_data_get(&path->power_terms);
  nterms = darray_power_term_size_get(&path->power_terms);
  id = medium_get_id(mdm);

  /* store term */
  term.term = val;
  term.id = id;
  term.vertex = *vtx;
  res = darray_power_term_push_back(&handle->path->power_terms, &term);
  if(res != RES_OK) goto error;

exit:
  return res;
error:
  goto exit;
}

res_T
green_path_add_flux_term
  (struct green_path_handle* handle,
   struct sdis_interface* interf,
   const struct sdis_interface_fragment* frag,
   const double val)
{
  struct green_path* path;
  struct flux_term* terms;
  struct unmerged_flux_term term = FLUX_TERM_NULL__;
  size_t nterms;
  unsigned id;
  res_T res = RES_OK;
  ASSERT(handle && interf && frag && val >= 0);

  res = ensure_interface_registration(handle->green, interf);
  if(res != RES_OK) goto error;

  path = handle->path;
  terms = darray_flux_term_data_get(&path->flux_terms);
  nterms = darray_flux_term_size_get(&path->flux_terms);
  id = interface_get_id(interf);

  /* store term */
  term.term = val;
  term.id = id;
  term.fragment = *frag;
  res = darray_flux_term_push_back(&handle->path->flux_terms, &term);
  if(res != RES_OK) goto error;

exit:
  return res;
error:
  goto exit;
}
```

Listing 8: green_function_solve_path for unmerged terms

```
static res_T
green_function_solve_path
  (struct sdis_green_function* green,
   const size_t ipath,
   double* weight)
```

```
{
  const struct power_term* power_terms = NULL;
  const struct flux_term* flux_terms = NULL;
  const struct green_path* path = NULL;
  const struct sdis_medium* medium = NULL;
  const struct sdis_interface* interf = NULL;
  struct sdis_scene* scn = NULL;
  double power;
  double flux;
  double end_temperature;
  size_t i, n;
  res_T res = RES_OK;
  ASSERT(green && ipath < darray_green_path_size_get(&green->paths) && weight);

  path = darray_green_path_cdata_get(&green->paths) + ipath;
  if(path->end_type == SDIS_GREEN_PATH_END_ERROR) { /* Rejected path */
    res = RES_BAD_OP;
    goto error;
  }

  /* Compute medium power terms */
  power = 0;
  n = darray_power_term_size_get(&path->power_terms);
  power_terms = darray_power_term_cdata_get(&path->power_terms);
  FOR_EACH(i, 0, n) {
    medium = green_function_fetch_medium(green, power_terms[i].id);
    power += power_terms[i].term
      * solid_get_volumic_power(medium, &power_terms[i].vertex);
  }

  /* Compute interface fluxes */
  flux = 0;
  n = darray_flux_term_size_get(&path->flux_terms);
  flux_terms = darray_flux_term_cdata_get(&path->flux_terms);
  FOR_EACH(i, 0, n) {
    interf = green_function_fetch_interf(green, flux_terms[i].id);
    flux += flux_terms[i].term
      * interface_side_get_flux(interf, &flux_terms[i].fragment);
  }

  /* Compute path's end temperature */
  switch(path->end_type) {
    case SDIS_GREEN_PATH_END_AT_INTERFACE:
      interf = green_function_fetch_interf(green, path->limit_id);
      end_temperature =
        interface_side_get_temperature(interf, &path->limit.fragment);
      break;
    case SDIS_GREEN_PATH_END_IN_VOLUME:
      medium = green_function_fetch_medium(green, path->limit_id);
      end_temperature = medium_get_temperature(medium, &path->limit.vertex);
      break;
    case SDIS_GREEN_PATH_END_RADIATIVE:
      SDIS(green_function_get_scene(green, &scn));
      SDIS(scene_get_ambient_radiative_temperature(scn, &end_temperature));
      if(end_temperature < 0) { /* Cannot have it negative if used */
        res = RES_BAD_ARG;
        goto error;
      }
      break;
    default: FATAL("Unreachable code.\n"); break;
  }

  /* Compute the path weight */
  *weight = power + flux + end_temperature;
```

```
exit:
  return res;
error:
  goto exit;
}
```

Appendix C. Figures

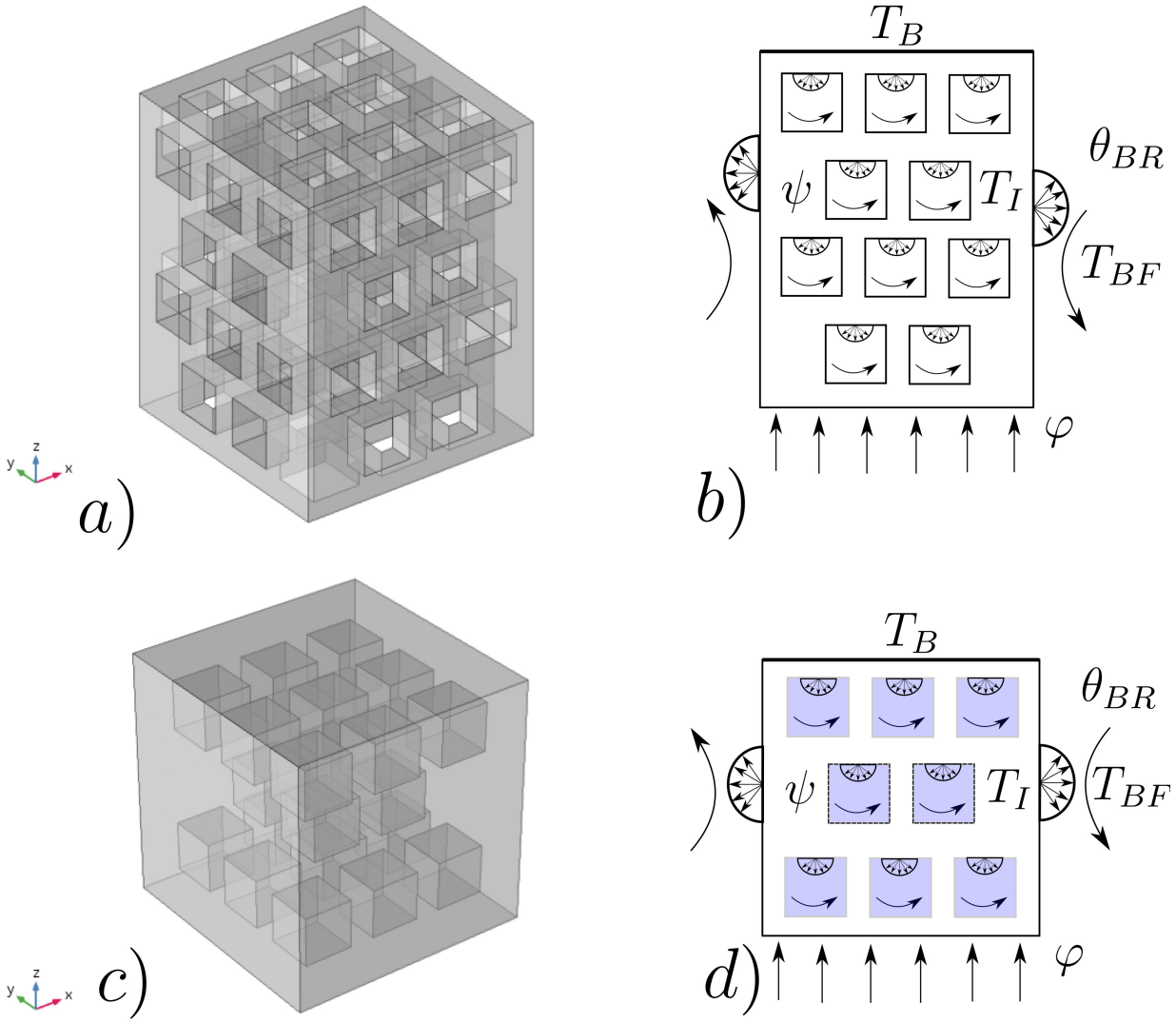

Figure C.8: Schemes of the two benchmark configurations. **a)** Geometry with open cavities; **b)** Different sources applied to this first benchmark for the physical problem. - **c)** Geometry with enclosed cavities, **d)** Sources applied the second benchmark for the physical problem.

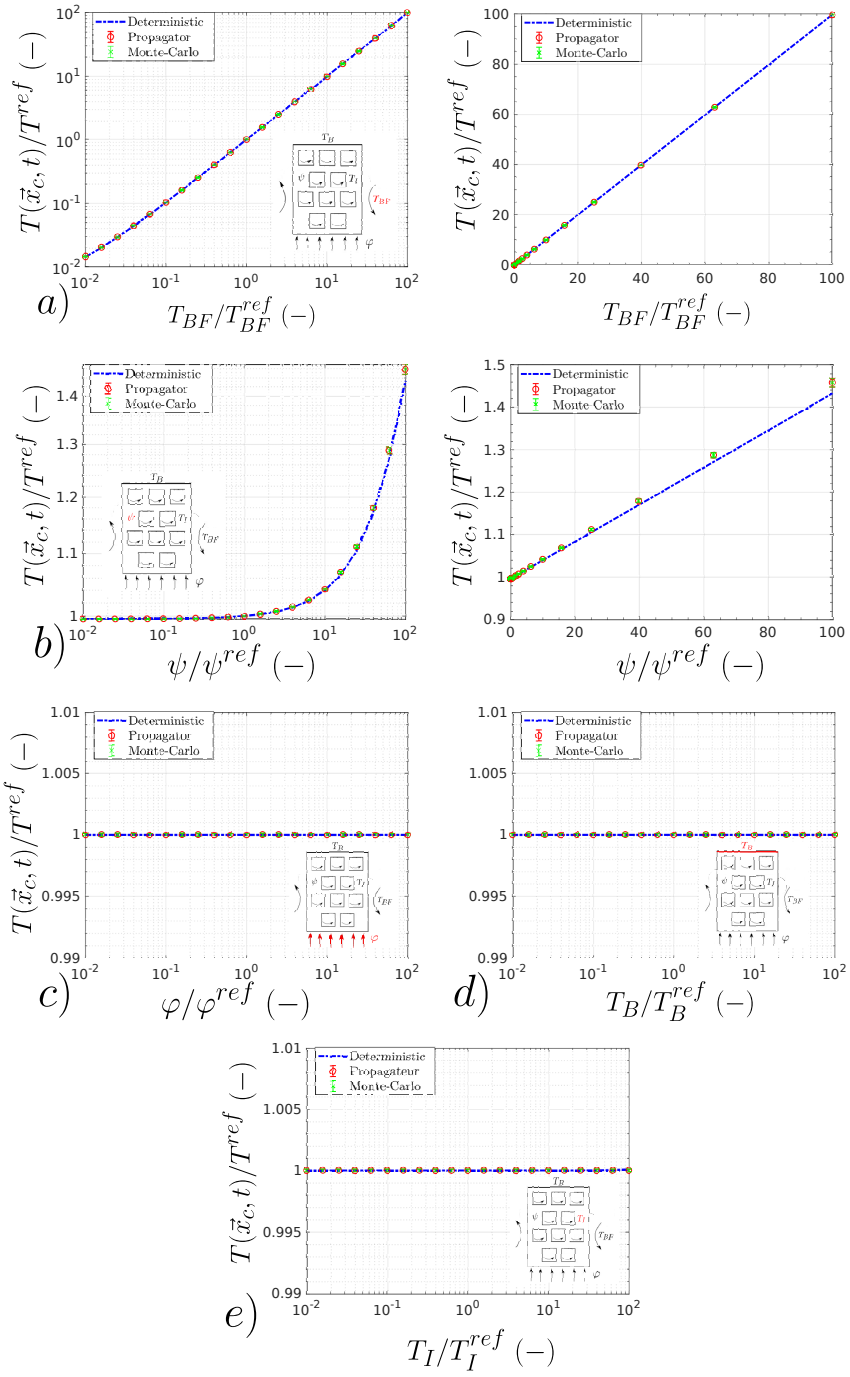

Figure C.9: Open-porosity geometry without radiative transfer : **a)** Ambient fluid temperature **b)** Power density **c)** Flux density **d)** Solid boundary temperature **e)** Initial temperature. Volume and surface of the geometry are noted V and S and $L = 4V/S$ ($L = 1m$) is retained as the characteristic size. The probe location $\vec{x}_c = (0.5, 0.5, 0.5)$ (at the center of the solid). The probe time estimation is $t* = \frac{\lambda t}{\rho c L^2 = 0.89}$ ($t = 1 \times 10^6$). The fluid reference temperature $T_{\mathrm{BF}}^{\mathrm{ref}} = 505\mathrm{K}$. The reference physical parameters are $\frac{T_{\mathrm{I}}^{\mathrm{ref}} - T_{\mathrm{BF}}^{\mathrm{ref}}}{T_{\mathrm{BF}}^{\mathrm{ref}}} = -0.01$ (reference initial temperature $T_{\mathrm{I}}^{\mathrm{ref}} = 500\mathrm{K}$), $\frac{T_{\mathrm{B}}^{\mathrm{ref}} - T_{\mathrm{BF}}^{\mathrm{ref}}}{T_{\mathrm{BF}}^{\mathrm{ref}}} = +0.01$ (known reference boundary temperature $T_{\mathrm{B}}^{\mathrm{ref}} = 510\mathrm{K}$). The convective heat transfer coefficient $h = 10\ W.m^{-2}.K^{-1}$ and the thermal conductivity $\lambda = 1\ W.m^{-1}.K^{-1}$ leading to $Bi = \frac{hL}{\lambda} = 10.68$ ($h = 10\ W.m^{-2}.K^{-1}$). For propagator function, initial calculation uses a dimensionless numerical step $\frac{\delta}{L} = 0.05$ and $N = 10^4$. The reference volume power value $\Psi^{\mathrm{ref}} = 20 W.m^{-3}$. The reference density flux $\varphi^{\mathrm{ref}} = 2000 W.m^{-2}$.

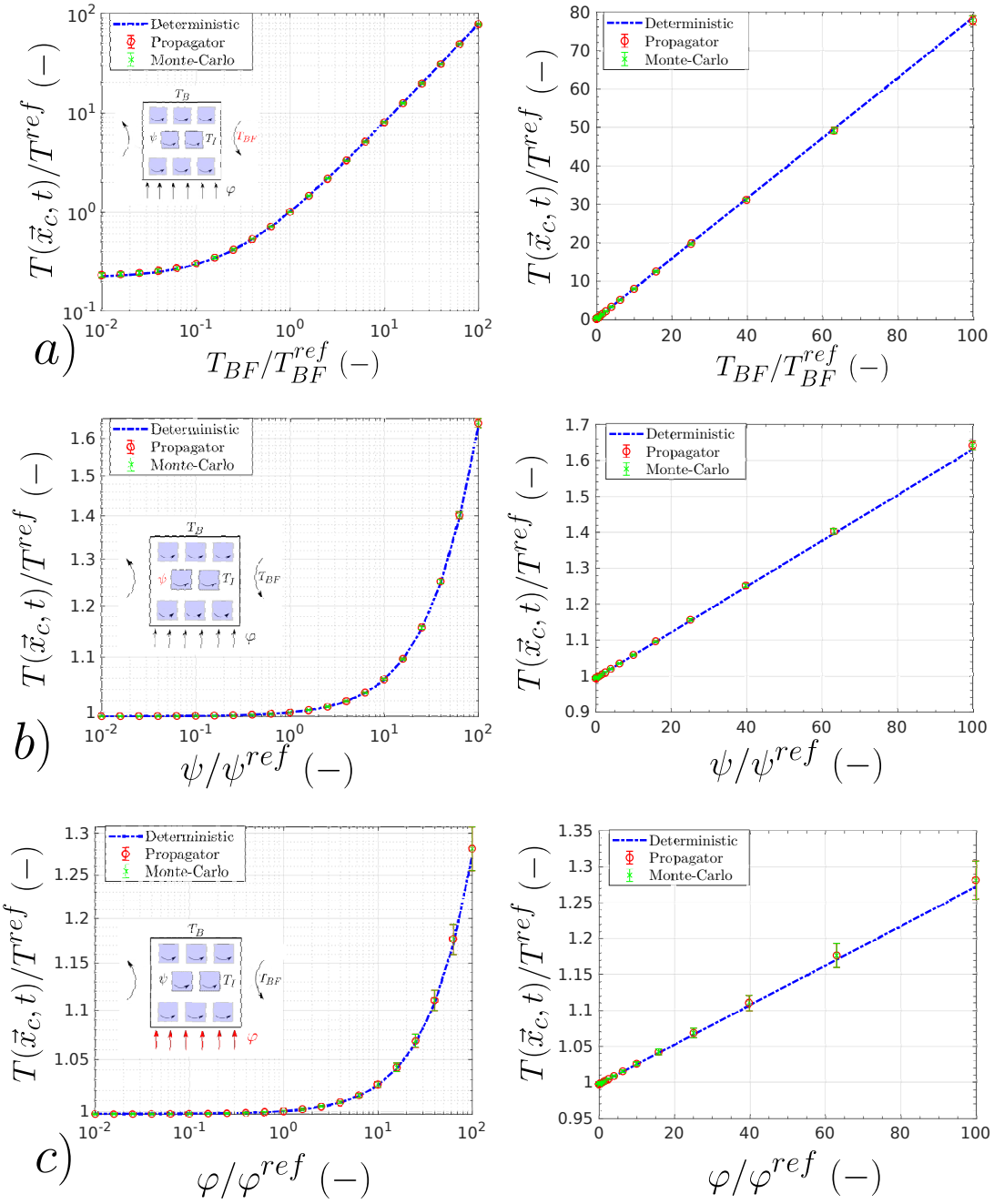

Figure C.10: Closed-porosity geometry without radiative transfer : **a)** Ambient fluid temperature **b)** Power density **c)** Flux density **d)** Solid boundary temperature **e)** Initial temperature. Volume and surface of the geometry are noted V and S and $L = 4V/S$ ($L = 2.4358m$) is retained as the characteristic size. The probe location $\vec{x}_c = (0.5, 0.5, 0.5)$ (at the center of the solid). The probe time estimation is $t* = \frac{\lambda t}{\rho c L^2}$=0.169 ($t = 1 \times 10^7$). The fluid reference temperature $T_{\mathrm{BF}}^{\mathrm{ref}} = 505\mathrm{K}$. The reference physical parameters are $\frac{T_{\mathrm{I}}^{\mathrm{ref}} - T_{\mathrm{BF}}^{\mathrm{ref}}}{T_{\mathrm{BF}}^{\mathrm{ref}}} = -0.01$ (reference initial temperature $T_{\mathrm{I}}^{\mathrm{ref}} = 500\mathrm{K}$), $\frac{T_{\mathrm{B}}^{ref} - T_{\mathrm{BF}}^{\mathrm{ref}}}{T_{\mathrm{BF}}^{\mathrm{ref}}} = +0.01$ (known reference boundary temperature $T_{\mathrm{B}}^{\mathrm{ref}} = 510\mathrm{K}$). The convective heat transfer coefficient is expressed as $h = 10\ W.m^{-2}.K^{-1}$, this is leading to $Bi = \frac{hL}{\lambda} = 24.358$ with $\lambda = 1\ W.m^{-1}.K^{-1}$. For propagator function, initial calculation uses a dimensionless numerical step $\frac{\delta}{L} = 0.05$ and $N = 10^4$. The reference volume power value $\Psi^{\mathrm{ref}} = 1W.m^{-3}$. The reference density flux $\varphi^{\mathrm{ref}} = 5W.m^{-2}$.

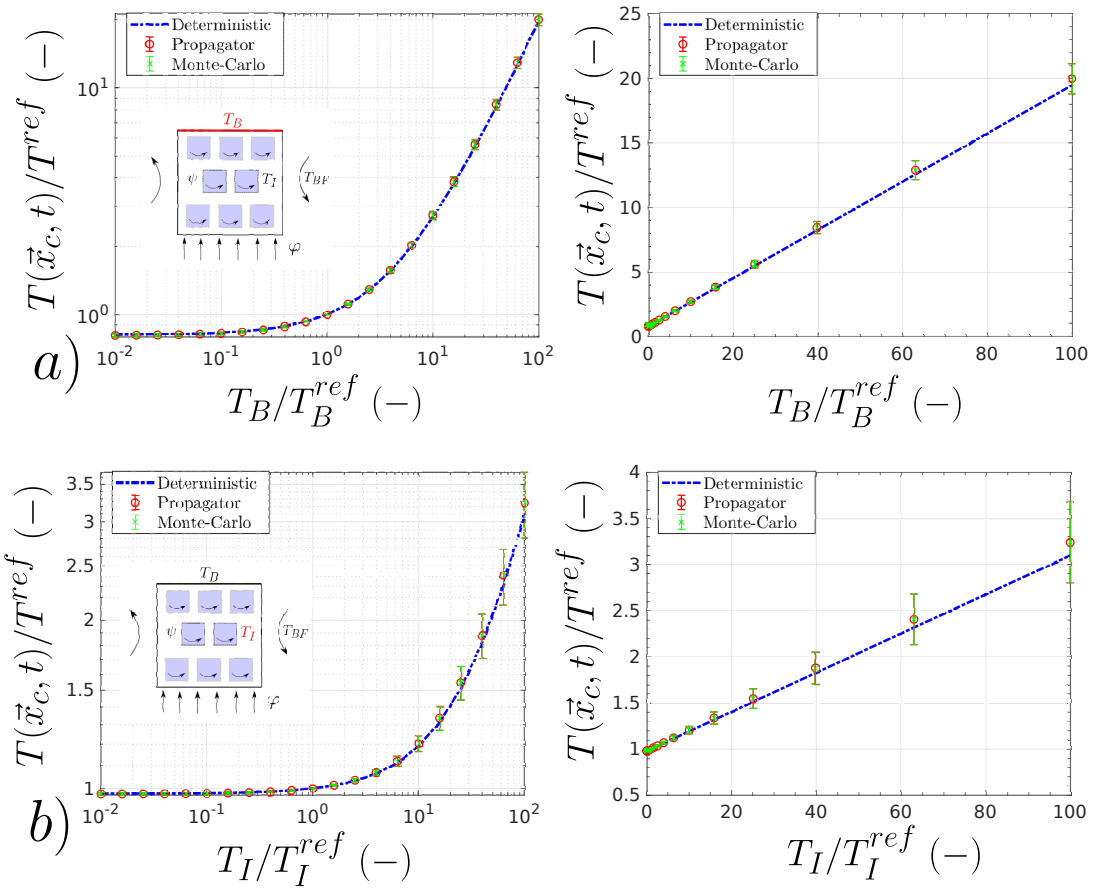

Figure C.11: Closed-porosity geometry without radiative transfer : **a)** ambient fluid temperature **b)** power density **c)** flux density **d)** solid boundary temperature **e)** initial temperature

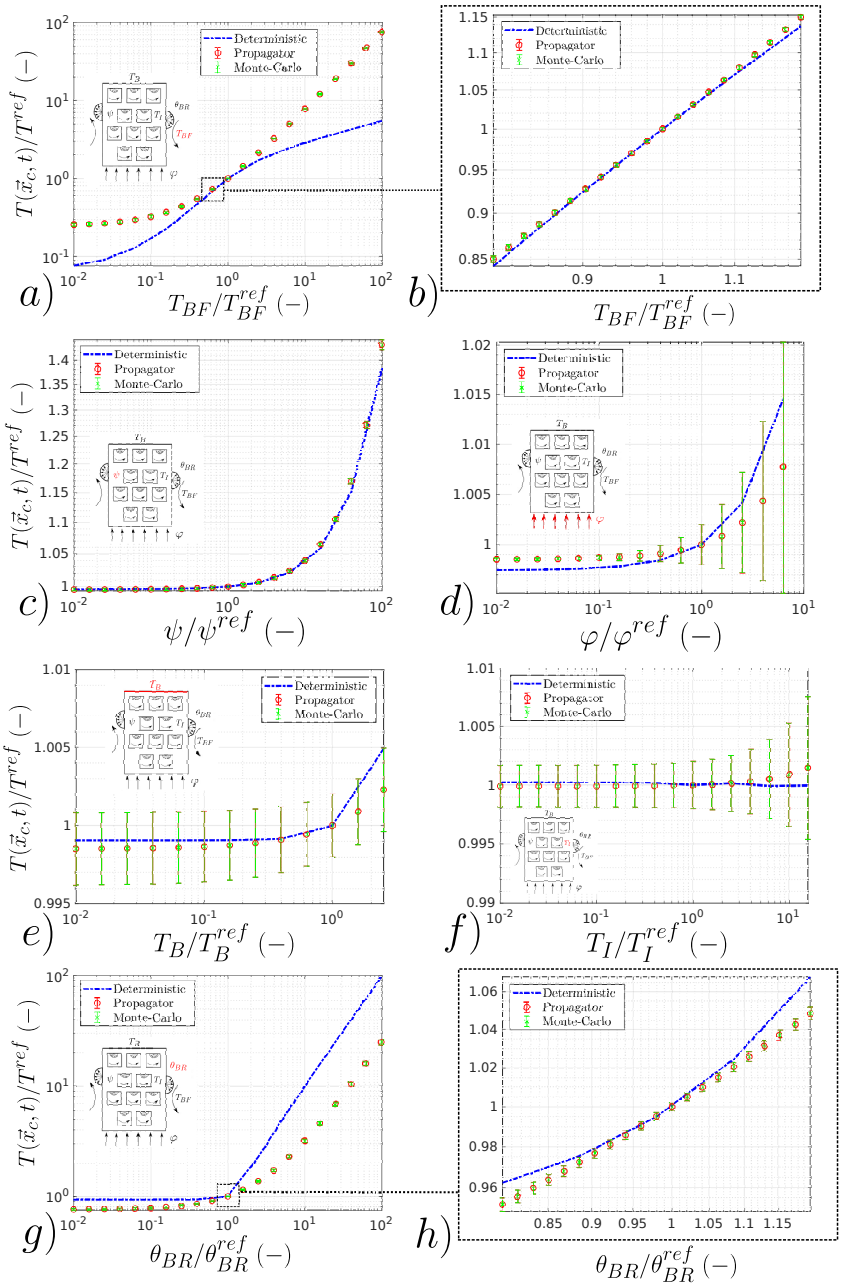

Figure C.12: open-porosity geometry with radiative transfer : **a-b)** ambient fluid temperature **c)** power density **d)** flux density **e)** solid boundary temperature **f)** initial temperature **g-h)** ambient radiant temperature. Volume and surface of the geometry are noted V and S and $L = 4V/S$ ($L = 1m$) is retained as the characteristic size. The probe location $\vec{x}_c = (0.5, 0.5, 0.5)$ (at the center of the solid). The probe time estimation is $t* = \frac{\lambda t}{\rho c L^2=0.89}$ ($t = 1 \times 10^6$). The fluid reference temperature $T_{\mathrm{BF}}^{\mathrm{ref}} = 505\mathrm{K}$. The reference physical parameters are $\frac{T_{\mathrm{I}}^{\mathrm{ref}} - \theta_{\mathrm{BR}}^{\mathrm{ref}}}{\theta_{\mathrm{BR}}^{\mathrm{ref}}} = -0.01$ (reference initial temperature $T_{\mathrm{I}}^{\mathrm{ref}} = 500\mathrm{K}$ and ambient radiative temperature $\theta_{\mathrm{BR}}^{\mathrm{ref}} = 505\mathrm{K}$), $\frac{T_{\mathrm{B}}^{\mathrm{ref}} - \theta_{\mathrm{BR}}^{\mathrm{ref}}}{\theta_{\mathrm{BR}}^{\mathrm{ref}}} = +0.01$ (known reference boundary temperature $T_{\mathrm{B}}^{\mathrm{ref}} = 510\mathrm{K}$). The radiative transfer coefficient is expressed as $h_R = 4\epsilon\sigma T_{\mathrm{ref}}^3 = 29.21$ with the emissivity epsilon $\epsilon = 1$, the Stefan-Boltzmann constant $\sigma = 5.6703\times10^{-8}\ J.s^{-1}.m^{-2}.K^{-4}$ and the reference temperature $T_{\mathrm{ref}} = 305\mathrm{K}$. This is leading to $Bi_R = \frac{h_R L}{\lambda} = 31.21$, with $\lambda = 1\ W.m^{-1}.K^{-1}$ and $Bi = \frac{hL}{\lambda} = 10.68$ ($h = 10\ W.m^{-2}.K^{-1}$). For propagator function, initial calculation uses a dimensionless numerical step $\frac{\delta}{L} = 0.05$ and $N = 10^4$. The reference volume power value $\Psi^{\mathrm{ref}} = 20W.m^{-3}$. The reference density flux $\varphi^{\mathrm{ref}} = 2000W.m^{-2}$.

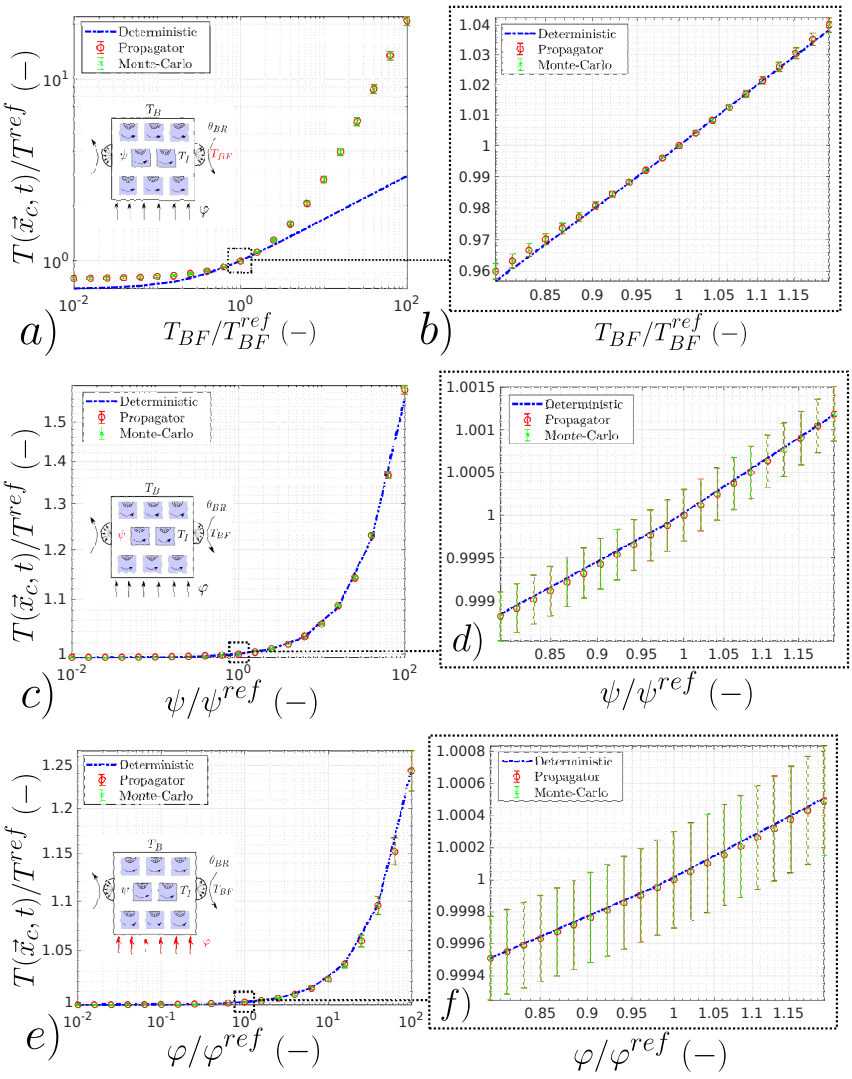

Figure C.13: closed-porosity geometry with radiative transfer : **a-b)** ambient fluid temperature **c-d)** power density **e-f)** flux density. Volume and surface of the geometry are noted V and S and $L = 4V/S$ ($L = 1m$) is retained as the characteristic size. The probe location $\vec{x}_c = (0.5, 0.5, 0.5)$ (at the center of the solid). The probe time estimation is $t* = \frac{\lambda t}{\rho c L^2 = 2.4358}$ ($t = 1 \times 10^6$). The fluid reference temperature $T_{\mathrm{BF}}^{\mathrm{ref}} = 505\mathrm{K}$. The reference physical parameters are $\frac{T_{\mathrm{I}}^{\mathrm{ref}} - \theta_{\mathrm{BR}}^{\mathrm{ref}}}{\theta_{\mathrm{BR}}^{\mathrm{ref}}} = -0.01$ (reference initial temperature $T_{\mathrm{I}}^{\mathrm{ref}} = 500\mathrm{K}$ and ambient radiative temperature $\theta_{\mathrm{BR}}^{\mathrm{ref}} = 505\mathrm{K}$), $\frac{T_{\mathrm{B}}^{\mathrm{ref}} - \theta_{\mathrm{BR}}^{\mathrm{ref}}}{\theta_{\mathrm{BR}}^{\mathrm{ref}}} = +0.01$ (known reference boundary temperature $T_{\mathrm{B}}^{\mathrm{ref}} = 510\mathrm{K}$). The radiative transfer coefficient is expressed as $h_R = 4\epsilon\sigma T_{\mathrm{ref}}^3 = 29.21$ with the emissivity epsilon $\epsilon = 1$, the Stefan-Boltzmann constant $\sigma = 5.6703 \times 10^{-8}\ J.s^{-1}.m^{-2}.K^{-4}$ and the reference temperature $T_{\mathrm{ref}} = 305\mathrm{K}$. This is leading to $Bi_R = \frac{h_R L}{\lambda} = 76.017$, with $\lambda = 1\ W.m^{-1}.K^{-1}$ and $Bi = \frac{hL}{\lambda} = 24.358$ ($h = 10\ W.m^{-2}.K^{-1}$). For propagator function, initial calculation uses a dimensionless numerical step $\frac{\delta}{L} = 0.05$ and $N = 10^4$. The reference volume power value $\Psi^{\mathrm{ref}} = 1W.m^{-3}$. The reference density flux $\varphi^{\mathrm{ref}} = 5W.m^{-2}$.

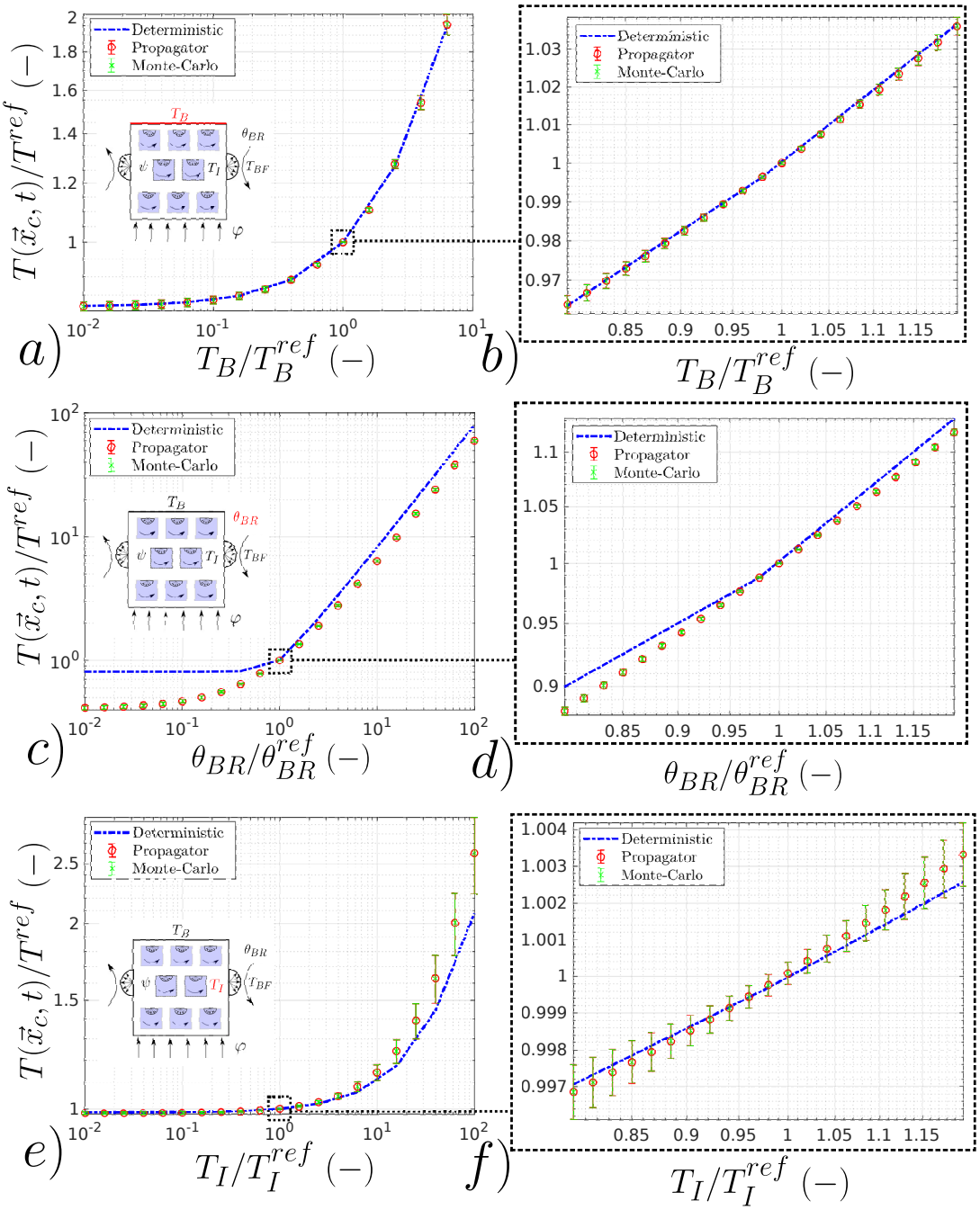

Figure C.14: Closed-porosity geometry with radiative transfer : **a-b)** solid boundary temperature **c-d)** ambient radiant temperature **e-f)** initial temperature