

Two point flux finite volume methods for mixture flows in porous media

Jürgen Fuhrmann

Weierstrass Institute for Applied Analysis and Stochastics (WIAS)

Joint work with **David Brust** (DLR Institute of Solar Research) and **Katharina Hopf** (WIAS)

Mixtures: Modeling, analysis and computing, Prague

February 05, 2025

Motivation

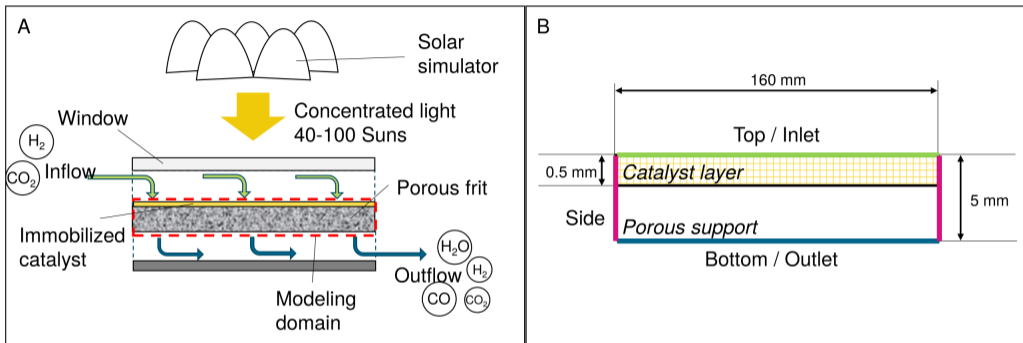
chemical processes driven by energy from solar light



- Use concentrated solar energy to drive chemical reactions
- Heat reactive gas mixture flowing through porous catalytic reactor
- E.g. conversion of CO_2 with H_2 to CO

Jülich solar tower
at DLR Institute for Solar Research
Image: DLR (CC BY-NC-ND 3.0)

The process



- Simulation domain delimited by red dashed lines

Multi-component reactive gas mixtures in porous media I

Darcy law + continuity for total mass density

$$\phi \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0$$

$$\vec{v} = -\frac{K}{\eta} \nabla p$$

ρ_i	Species mass density
T	Temperature
$\rho = \sum_{i=1}^n \rho_i$	Total mass density
$\varrho = (\rho_1 \dots \rho_n)$	vector of mass densities
\vec{v}	Mass averaged velocity
p	Total pressure
$\eta(\varrho, T)$	Viscosity
K, ϕ	Permeability, porosity

Multi-component reactive gas mixtures in porous media II

Convective-diffusive mass transport

$$\phi \frac{\partial \rho_i}{\partial t} + \nabla \cdot (\rho_i \vec{v} + \vec{J}_i) - \phi r_i = 0, \quad i = 1 \dots n$$

$$- \sum_{\substack{j=1 \\ j \neq i}}^n \frac{x_i x_j}{D_{ij}} \left(\frac{\vec{J}_j}{\rho_i} - \frac{\vec{J}_i}{\rho_j} \right) = \nabla x_i + (x_i - w_i) \frac{\nabla p}{p}$$

$$\sum_{i=1}^n x_i = 1 \quad \sum_{i=1}^n \vec{J}_i = 0$$

\vec{J}_i	Diffusive species mass flux
x_i	Species molar fractions
w_i	Species mass fractions
D_{ij}	Effective Maxwell-Stefan diffusivities
$r_i(\rho, T)$	Reaction terms

Multi-component reactive gas mixtures in porous media II

Thermal energy transport

$$\frac{\partial(1 - \phi)\rho_s h_s + \phi\rho h_{mix}}{\partial t} + \nabla \cdot (c\rho T\vec{v}) - \nabla \cdot (\lambda\nabla T) = \phi q$$

$\lambda(\varrho, T)$	Effective thermal conductivity
$c(\varrho, T)$	Mixture heat capacity
$q(\varrho, T)$	Heat sources from chemical reactions
$h_s(T)$	Porous matrix enthalpy
$h_{mix}(\varrho, T)$	Mixture enthalpy

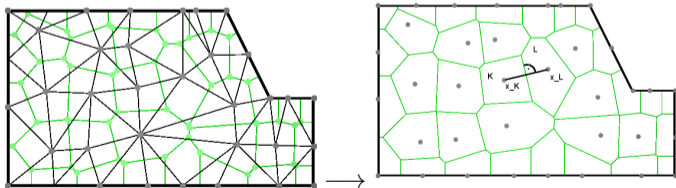
Further elements of the model

- Basic variables: $p, x_1 \dots x_n, T$
- Soret, Dufour effects
- Full nonlinear dependency of parameters on composition, temperature, pressure
- Boundary conditions: Defined by experimental conditions: one can control flow rate and composition at inlet, and pressure at outlet.
 - Inlet: given total mass fluxes, species mass fluxes
 - Outlet:
 - fixed pressure $p = p_0$
 - convective outflow (zero diffusion flux) for species: $\vec{J}_i \cdot \vec{n} = 0$
 - zero diffusion flux + heat source for heat : $\nabla T \cdot \vec{n} = \Psi(T)$
 - Robin boundary conditions depending on insolation for heat a surrogate for radiation transport outside of domain
- Free energy + entropy principle for isothermal case

Computational realization

Finite volume discretization method

- Subdivide computational domain into control volumes aka “representative elementary volumes” (REV)
- Use Gauss theorem to derive balance laws for REV's from PDE
- Tricky:
 - Numerically stable fluxes between REV's in the presence of convection
 ⇒ upwinding tools from semiconductor simulation
 - REV generation: start with triangulation, create REV's from joining triangle circumcenters



Two point flux finite volumes for systems of PDEs

- System of N coupled PDEs in $\Omega \subset \mathbb{R}^d$ and time interval $(0, T)$

$$\partial_t \mathbf{s}(\mathbf{u}) + \nabla \cdot \vec{\mathbf{j}}(\mathbf{u}, \vec{\nabla} \mathbf{u}) + \mathbf{r}(\mathbf{u}) = 0,$$

- $\mathbf{s} : \mathbb{R}^N \rightarrow \mathbb{R}^N$: local amount, $\mathbf{r} : \mathbb{R}^N \rightarrow \mathbb{R}^N$: reaction, $\vec{\mathbf{j}} : \mathbb{R}^N \times \mathbb{R}^{Nd} \rightarrow \mathbb{R}^{Nd}$: flux
- Integration over space-time control volume $\omega_k \subset \bar{\Omega}$, $[t^{m-1}, t^m] \in [0, T]$:

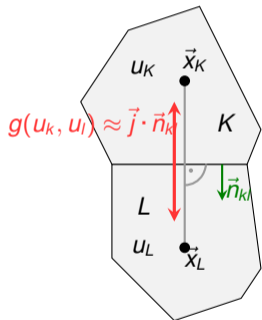
$$\int_{\omega_k} \mathbf{s}(\mathbf{u}(\vec{x}, t^m)) d\vec{x} - \int_{\omega_k} \mathbf{s}(\mathbf{u}(\vec{x}, t^{m-1})) d\vec{x} + \int_{t^{m-1}}^{t^m} \left(\int_{\partial\omega_k} \vec{\mathbf{j}} \cdot \vec{n} ds + \int_{\omega_k} \mathbf{r}(\mathbf{u}) d\vec{x} \right) dt = 0$$

- Discrete flux $\mathbf{g}(\mathbf{u}_k, \mathbf{u}_l) \approx h_{kl} \vec{\mathbf{j}} \cdot \vec{n}_{kl} \Rightarrow$ approximation:

$$|\omega_k| \left(\mathbf{s}(\mathbf{u}_k^m) - \mathbf{s}(\mathbf{u}_k^{m-1}) \right) + (t^m - t^{m-1}) \left(\sum_{\omega_l \text{ neighbour of } \omega_k} \frac{|\sigma_{kl}|}{h_{kl}} \mathbf{g}(\mathbf{u}_k^m, \mathbf{u}_l^m) + |\omega_k| \mathbf{r}(\mathbf{u}_k^m) \right) = 0$$

Deriving a software API

The structure of the discrete system allows to separate physics described by constitutive functions (\mathbf{r} , \mathbf{s} , \mathbf{g} + function describing boundary conditions) from geometry described by the discretization grid.



- Storage and reaction terms \mathbf{s} , \mathbf{r} are similar to those in the continuous formulation
- Similar case for boundary conditions
- Fluxes \mathbf{g} use stabilized finite difference expressions
- Some problem classes covered by this approach
 - Ion transport in electrolytes
 - Charge transport in semiconductors
 - Multiphase flow in porous media
 - Mixture flow in porous media
 - ...

With basis unknowns $\mathbf{u} = (x_1, \dots, x_n, p, T)$, the flux function \mathbf{g} can be written as

$$\mathbf{g}(\mathbf{u}_k, \mathbf{u}_l) = (g_{kl}^{\rho_1} \dots g_{kl}^{\rho_{n-1}}, g_{kl}^{\rho}, g_{kl}^h).$$

- (Arithmetic) edge averages of quantities: $\bar{\xi}_{kl} = \frac{1}{2} (\xi_k + \xi_l)$
- Darcy velocity: $v_{kl} = \left(\frac{K}{\bar{\eta}_{kl}} \right) (p_k - p_l)$
- Total mass flux: $g_{kl}^{\rho} = \bar{\rho}_{kl} v_{kl}$
- Species mass fluxes: sum of convective fluxes and diffusive fluxes J_{kl}^i

$$g_{kl}^{\rho_i} = J_{kl}^i + \bar{\rho}_{kl}^i v_{kl}, \quad i = 1, \dots, n-1$$

- Enthalpy flux: $g_{kl}^h = \bar{\lambda}_{kl} (T_k - T_l) + \sum_{i=1}^n \bar{h}_{kl}^i g_{kl}^i$

Discrete diffusive species fluxes

- J_{kl}^i are defined by the solution of an $(n-1) \times (n-1)$ linear system: $-\mathbf{H}\mathbf{J} = \mathbf{F}$

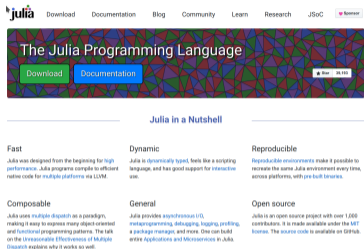
$$F_{kl}^i = (x_k^i - x_l^i) + \bar{D}_{p,kl}^i \frac{p_k - p_l}{\bar{p}_{kl}}, \quad i = 1, \dots, n-1,$$

$$H_{kl}^{ii} = \frac{\bar{w}_{kl}^i}{M^i M^n \mathcal{D}^{in}} + \sum_{\substack{o=1 \\ o \neq i}}^n \frac{\bar{w}_{kl}^o}{M^i M^o \mathcal{D}^{io}}, \quad i = 1, \dots, n-1$$

$$H_{kl}^{ij} = -\bar{w}_{kl}^i \left(\frac{1}{M^i M^j \mathcal{D}^{ij}} - \frac{1}{M^i M^n \mathcal{D}^{in}} \right), \quad (i, j) = 1, \dots, n-1, i \neq j$$

- D_p^i are the pressure-diffusion coefficients $x_i - w_i$
- Implementation requires solution of linear system with solution dependent matrix

Julia language for scientific computing & data science



The screenshot shows the Julia Programming Language website. At the top, there is a navigation bar with links for Download, Documentation, Blog, Community, Learn, Research, JSoC, and a GitHub repository link. Below the navigation bar is a large banner with the text "The Julia Programming Language" and two buttons: "Download" and "Documentation". A small badge on the right of the banner indicates "1.8.0" and "30,000". Below the banner is a section titled "Julia in a Nutshell" with three columns of text:

- Fast**: Julia was designed from the beginning for high performance. Julia programs compile to efficient native code for multiple platforms via LLVM.
- Dynamic**: Julia is dynamically typed, feels like a scripting language, and has good support for interactive use.
- Reproducible**: Reproducible environments make it possible to recreate the same Julia environment every time, across platforms, with one built binary.
- Composable**: Julia uses multiple dispatch as a paradigm, making it easy to express many object-oriented and functional programming patterns. The talk on the Unconquered Effectiveness of Multiple Dispatch explains why it works so well.
- General**: Julia provides asynchronous I/O, metaprogramming, debugging, logging, profiling, a package manager, and more. One can build entire applications and microservices in Julia.
- Open source**: Julia is an open source project with over 1,000 contributors. It is made available under the MIT license. The source code is available on GitHub.

- Just-in-time (JIT) compilation \Rightarrow C-like performance
- Syntax level like python, matlab
- Best-in-class package manager supporting reproducibility
- Open source (MIT License)

Growing package ecosystem

- SciML.ai
 - LinearSolve.jl: Common interface to dense, sparse, direct and iterative solvers
 - DifferentialEquations.jl: State of the art ODE solvers
 - Symbolic tools for code generation/transformation
- TetGen.jl, Triangulate.jl, Gmsh.jl for mesh generation
- Various visualization tools

Forward mode automatic differentiation

ForwardDiff.jl

- Let $\varepsilon^2 = 0$. Ring of dual numbers $\mathbb{D} = \{a + b\varepsilon \mid a, b \in \mathbb{R}\}$
- Evaluation of polynomial $p(x) = \sum_{i=0}^n p_i x^i$:

$$p(a + \varepsilon) = \sum_{i=0}^n p_i a^i + \sum_{i=1}^n i p_i a^{i-1} \varepsilon = p(a) + p'(a)\varepsilon.$$

- Library of differentiation rules for special function
- Generalization to multi-dual numbers
- Implement functions in a generic manner, JIT creates specialized code for “normal” or dual numbers, depending on what is passed to a function

- 1/2/3D
- Delaunay mesh generation using Triangle, TetGen mesh generators
- Implicit Euler time discretization
 - Optionally use ODE solvers from DifferentialEquations.jl
- Newton method with analytical Jacobians (+ damping, parameter embedding)
- Use forward mode automatic differentiation to assemble Jacobians
 - Write code just for functions
 - Calculate “local” jacobians from flux/reaction/storage functions using AD
 - Efficient assembly into global Jacobi matrix via intermediate linked list sparse matrix structure
- Various direct and iterative linear solvers via LinearSolve.jl
- Part of WIAS-PDELib.jl: Julia packages maintained by WIAS numerical math & scientific computing group

VoronoiFVM.jl

More features + usage cases

Some further features:

- Handling of surface species, different subdomain species sets
- Small signal analysis/impedance spectroscopy
- Coupling with pressure robust FEM for Navier-Stokes (WIAS-PDELib/ExtendableFEM.jl by Christian Merdon)
- Multithreading based parallelization

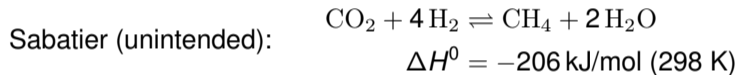
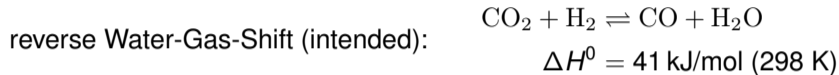
Some usage cases:

- Perovskites/semiconductors with additional ionic species ¹
- Solid oxide electrochemical cells ², redox flow cells
- Liquid electrolytes: Poisson-Nernst-Planck with ion size constraints ³
- Chemical reactors with heterogeneous catalysis

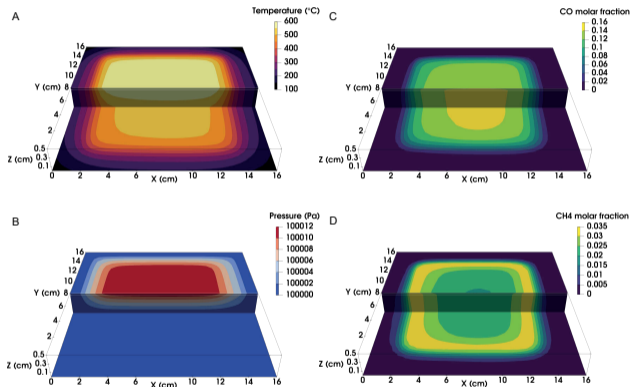
1)B. Spetzler, D. Abdel, F. Schwierz, M. Ziegler and Patricio. Farrell. Advanced Electronic Materials, 2300635 (2023)

2)V. Miloš, P. Vágner, D. Budáč, M. Carda, M. Paidar, J. Fuhrmann and K. Bouzek. Journal of the Electrochemical Society 044505 (2022)

3)Ch. Keller, J. Fuhrmann, and M. Landstorfer, WIAS Preprint 3072, 2023.

Creation of synthesis gas from CO₂ and concentrated solar energy

Stationary 3D simulation results



- A) temperature
- B) total pressure
- C) molar fraction of CO
- D) molar fraction of CH₄
 - Catalyst layer region outlined by a thin white line
 - Z axis scaled by factor of 4 to increase readability

- Solution via time embedding using implicit Euler method to obtain initial value for stationary solve.

Conclusions/outlook

Conclusions:

- Nonisothermal Darcy-Stefan-Maxwell model
- Outflow boundary conditions for gaseous species where outlet composition is unknown a-priori due to reactions in the interior of the domain
- 1D/2D/3D implementation via two-point flux finite volume method in Julia, taking advantage of automatic differentiation for the generation of Jacobi matrix

To be investigated:

- Entropy behaviour of the finite volume scheme à la Cancès/Cauvin-Vila/Chainais-Hillairet/Ehrlacher
- More realistic radiation transport
- ...