Department of Mathematics and Informatics
University of Cologne
Prof. Dr. Gregor Gassner

April 2, 2026

An Introduction to Climate Modeling with Julia

Milestone 5 - Constructing a System Matrix with FD Approach

1 Constructing a System Matrix with FD Approach

In this milestone, you will build the heart of the climate model: the system matrix for solving the energy balance model. The EBM model reads

$$C(x)\frac{\partial T}{\partial t} + A(CO_2) + BT - \underbrace{\vec{\nabla}\cdot(D(x)\nabla T)}_{L} = S_{\text{sol}}(x,t) \tag{1}$$

in spherical coordinates on the sphere's surface with the pair $(\tilde{\theta}, \varphi)$

$$L(\tilde{\theta},\varphi) = \csc^2(\tilde{\theta})\frac{\partial}{\partial\varphi}\left(\tilde{D}\frac{\partial T}{\partial\varphi}\right) + \frac{\partial}{\partial\tilde{\theta}}\left(\tilde{D}\frac{\partial T}{\partial\tilde{\theta}}\right) + \cot(\tilde{\theta})\tilde{D}\frac{\partial T}{\partial\tilde{\theta}} \tag{2}$$

When we apply the finite difference method, we have for each degree of freedom:

$$\frac{\partial T_{j,i}}{\partial t} \approx \underbrace{\frac{L_{j,i}}{C_{j,i}} - \frac{B}{C_{j,i}}T_{j,i}}_{R_{j,i}(T)} + \underbrace{\frac{1}{C_{j,i}}\left(S_{\text{sol},j,i}(t) - A\right)}_{F_{j,i}} \tag{3}$$

The discretization of the heat conduction term on the surface of a sphere with a Cartesian mesh of cell size h (uniform for latitude and longitude) reads:

- For the inner degrees of freedom:

$$\begin{aligned} L_{j,i} = \frac{\csc^2(\tilde{\theta}_{j,i})}{h^2}\Big(&- 2\tilde{D}_{j,i}T_{j,i} + \left[\tilde{D}_{j,i} - \frac{1}{4}(\tilde{D}_{j,i+1} - \tilde{D}_{j,i-1})\right]T_{j,i-1} \\ &+ \left[\tilde{D}_{j,i} + \frac{1}{4}(\tilde{D}_{j,i+1} - \tilde{D}_{j,i-1})\right]T_{j,i+1}\Big) \\ + \frac{1}{h^2}\Big(&- 2\tilde{D}_{j,i}T_{j,i} + \left[\tilde{D}_{j,i} - \frac{1}{4}(\tilde{D}_{j+1,i} - \tilde{D}_{j-1,i})\right]T_{j-1,i} \\ &+ \left[\tilde{D}_{j,i} + \frac{1}{4}(\tilde{D}_{j+1,i} - \tilde{D}_{j-1,i})\right]T_{j+1,i}\Big) \\ &+ \cot(\tilde{\theta}_{j,i})\frac{\tilde{D}_{j,i}}{2h}\left[T_{j+1,i} - T_{j-1,i}\right] \end{aligned} \tag{4}$$

- For the degrees of freedom at the poles:

$$L_{1,i} = \frac{\sin(h/2)}{4\pi\,\text{area[1]}}\sum_{k=1}^{\text{n\_longitude}}\frac{1}{2}\left(\tilde{D}_{1,k} + \tilde{D}_{2,k}\right)\left[T_{2,k} - T_{1,k}\right], \tag{5}$$

$$L_{\text{n\_latitude},i} = \frac{\sin(h/2)}{4\pi\,\text{area[n\_latitude]}}\sum_{k=1}^{\text{n\_longitude}}\frac{1}{2}\left(\tilde{D}_{\text{n\_latitude},k} + \tilde{D}_{\text{n\_latitude-1},k}\right)\left[T_{\text{n\_latitude-1},k} - T_{\text{n\_latitude},k}\right] \tag{6}$$

for all $i \in \{1,\ldots,\text{n\_longitude}\}$, where `area[1]` $:=$ `area[n_latitude]` $:= 0.5\,(1-\cos\,(h/2))$ contains the normalized area of the polar cap.

To implement the model, proceed as follows:

1. Implement the mesh as a struct in Julia or a class in Python with a constructor.

Parameter	*Description*
`n_latitude`	Number of DOFs in longitude
`n_longitude`	Number of DOFs in latitude
`ndof`	Total number of DOFs
`h`	Cell size in latitude direction in radians
`geom`	Geometrical parameter at poles with geom $= \frac{\sin(h/2)}{4\pi \texttt{area}[1]}$
`csc2`	Metric term for the transformation to spherical coordinates, the vector with entries $\csc^2(\tilde{\theta}_j)$
`cot`	Metric term for the transformation to spherical coordinates, the vector with entries $\cot(\tilde{\theta}_j)$
`area`	Area of each cell on the surface of the sphere which only depends on latitude

2. Calculate the earth surface type dependent heat conduction coefficients $\tilde{D}(\tilde{\theta}, \varphi) \in \mathbb{R}^{n_y \times n_x}$ by implementing a function `calc_diffusion_coefficients` that maps the following equation:

$$\tilde{D}(\tilde{\theta}, \varphi) \in \mathbb{R}^{n_y \times n_x} = \begin{cases} \tilde{D}_{\text{ocean,poles}} + (\tilde{D}_{\text{ocean,equ}} - \tilde{D}_{\text{ocean,poles}}) \sin(\tilde{\theta})^5 & \text{for the ocean} \\ \tilde{D}_{\text{NP}} + (\tilde{D}_{\text{equ}} - \tilde{D}_{\text{NP}}) \sin(\tilde{\theta})^5 & \text{for the Northern Hemisphere} \\ \tilde{D}_{\text{SP}} + (\tilde{D}_{\text{equ}} - \tilde{D}_{\text{SP}}) \sin(\tilde{\theta})^5 & \text{for the Southern Hemisphere} \end{cases} \tag{7}$$

The corresponding thermal conductivity coefficient[1] are as follows:

Types	*Thermal conductivity in* $W/m^2/K$
Ocean at Poles	0.4
Ocean at Equator	0.65
Equator	0.65
North Pole	0.28
South Pole	0.2

Table 1: Surface specific thermal conductivity for the determination of diffusion coefficients

Note: For the surface type dependency of the heat conduction, you will once again need to include the function `read_geography` from milestone 1.

3. Use the above formulas, the function `calc_diffusion_coefficients` and the functions for the other parameters from previous milestones to write a function `compute_equilibrium_EBM_2D` to compute the EBM for two spatial dimensions using a forward Euler time integration scheme. Try to calculate the EBM using this function. What happens?

4. Write a function `calc_jacobian_ebm_2d` that computes the Jacobian matrix of the FD operator numerically. You can calculate the j-th column of the Jacobian matrix with the equation:

$$\underline{\mathbf{A}}_{\text{EBM}} \hat{\mathbf{K}}^j = \mathbf{R}(\hat{\mathbf{K}}^j) - \mathbf{R}(\mathbf{0})$$

where $\hat{\mathbf{K}}^j = (\hat{k}_1^j, \hat{k}_2^j, \ldots, \hat{k}_{\text{NDOF}}^j)^T$ is a vector whose entries are defined as

$$\hat{k}_i^j = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

[1] K. Zhuang, G.R. North, M.J. Stevens, *A NetCDF version of the two-dimensional energy balance model based on the full multigrid algorithm*, SoftwareX, Vol. 6, pp. 198-202, July 7, 2017.

Note that the matrix F does not depend on T and cancels out.

Here, we treat $\mathbf{R}$ as a function $\mathbf{R}\colon \mathbb{R}^{\text{NDOFS}} \to \mathbb{R}^{\text{NDOFS}}$. More precisely, in differential geometry terms, $R^{m\times n}$ is an mn-dimensional manifold, and in order to compute a Jacobian of a function on that space (or do calculus in general), we need coordinate charts mapping this space to $\mathbb{R}^{\text{NDOFS}}$. Under these charts, we can calculate the Jacobian. A simple chart for this space maps a matrix to a vector containing all its entries. The inverse of this chart then reorganizes these entries into a matrix. In code, we can do the same with the functions `flatten` and `reshape`, which are available in all languages for numerical computing.

This function requires many calls to `calc_diffusion_coefficients`. It is therefore important that the diffusion coefficients can be computed in the microsecond range. Unfortunately, Python, being an interpreted language, is slow. To bring `calc_diffusion_coefficients` into the microsecond range, we recommend using Numba's `@njit` decorator. Note that this only works when the function parameters are of certain types, mostly Numpy types. Therefore, this function cannot take the mesh as a parameter and has to take the mesh's fields directly.

5. Examine the sparsity pattern of the Jacobian matrix. What does the matrix look like? Be aware that `spy` of Plots.jl can be very slow for large matrices when using the PythonPlot backend. If you want to examine the whole Jacobian, use the GR backend instead.

6. Compute the eigenvalues of the Jacobian matrix. Determine a practical approximation to the largest time step for which the scheme is stable using the forward Euler time integration scheme.

2 Control Solutions

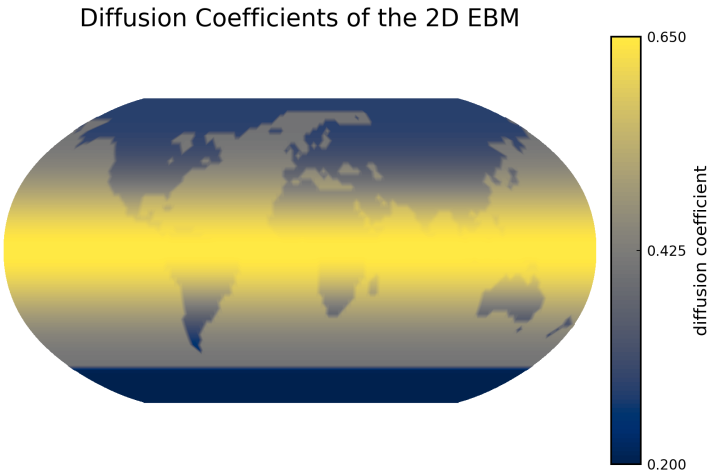

Figure 1: Diffusion coefficients

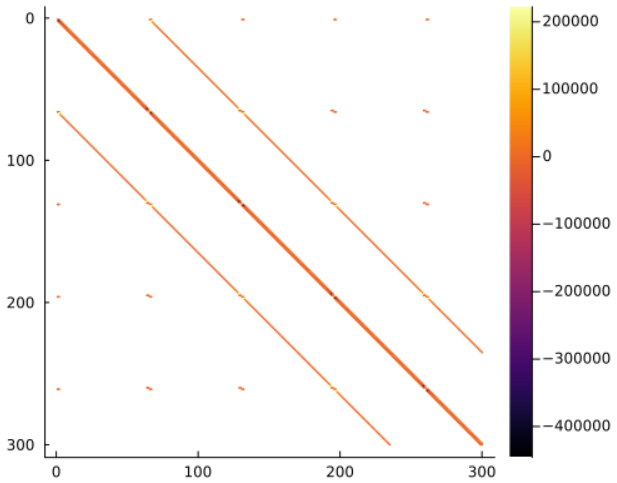

Figure 2: Sparsity pattern of the upper left part of the system matrix

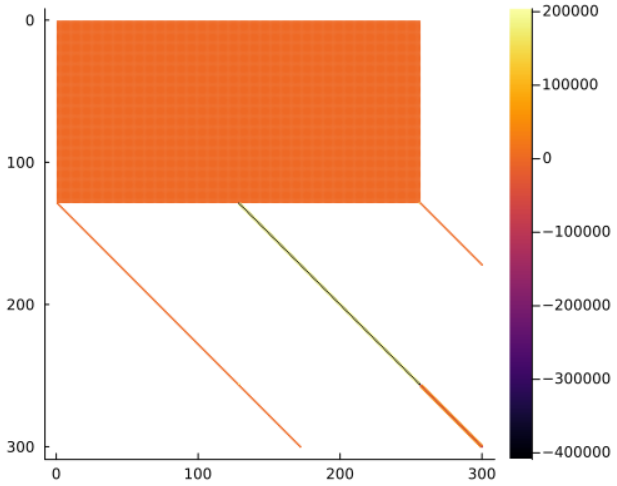

Figure 3: Sparsity pattern of the upper left part of the system matrix in a row-major language like Python