

Julia を用いた NetCDF の 解析・可視化

榎村 博基

はじめに

- 最近（ここ半年ほど）、プログラミング言語 Julia を用いてNetCDFデータの解析・可視化を試みているので、その感触をお伝えする
 - これまでの Ruby/GPhys/DCL 環境からの乗り換えを検討中
 - 手を動かす機会が少なく、まだ十分に乗り換えられていない
 - ちなみに、檜村の GPhys/DCL 歴は 20年（since 学部4年生）
 - ▶ DCLへのシステムフォントの導入
 - ▶ gpview を魔改造した独自のコマンドラインツール gpv の開発
 - ✓ そのときどきの（自分の）需要に応じて、建て増し改造
- <https://github.com/hiroki-mac/gpv>
- ✓ 紹介資料（DCPAM 解析・描画方法意見交換会 2020.3.30）
https://www.gfd-dennou.org/library/dcpam/related-visualization-info/2020-03-30/hiroki/kashimura_gpv.pdf

背景（動機）

- GPhys/DCCLの用途

- A. 自分で研究に使う
(大規模計算のデータ処理)

- B. 研究室の学生に使わせる

- C. 学部の少人数実習で、3年生に使わせる

- GPhys/DCCLの不満点

- ➡ 大規模データの処理が遅い
(2万×1万点の描画～十数分)
複数言語依存 (Fortran/C/Ruby)
で開発維持のハードルが高い

- ➡ メソッドへの依存
gpview 出力をそのまま論文に

- ➡ 環境構築の難易度が上がってきた
 - ARM版 Windows (WSL2)
 - gcc の厳格化
 - Ruby の進歩が速い生成 AI による誤情報
(Web情報が相対的に少ないので)

Python でいいのでは？

- xarray/matplotlib . . . みんな使ってる（よく知らんけど）
- Python を避けた理由
 - みんな使ってるから
 - わざわざ教えなくても、生成AIに聞けばいいじゃん
 - パッケージシステムが乱立
 - pip、conda、uv
 - バージョン問題に辟易
 - ライセンス問題？（anaconda）
 - 文法が好きじゃない（インデントの自由度、セミコロン）
 - NumPy 依存（結局、C が必要？）

Julia とは [検索は “julia lang”]

- 2009年にMITの研究者らによって開発が始まり、2012年に公開された科学技術計算 や 数値解析 を主な用途として設計されている汎用言語
- **特徴**
 - インタープリタ型（コンパイル不要）
 - 動的型付け
 - 高速（C / Fortran なみ）
 - 関数が主役（マルチパラダイム言語）
 - 実現したい機能を関数化し、関数毎にコンパイルすることで、抽象性と高速性を両立
 - 他言語（C / Fortran / Python など）の呼び出しが容易
 - 並列計算・分散処理
 - 組込の公式パッケージシステム
 - 配列が Fortran (Column-major) 並び

Julia の導入方法・使い方

- 導入方法

- Linux/WSL2

- ▶ `curl -fsSL https://install.julialang.org | sh`

- macOS

- ▶ `brew install julia` または `sudo port install julia`

- Windows11

- ▶ `winget install --name Julia --id 9NJNWW8PVKMN -e -s msstore`

- 使い方

- スクリプトファイルを作成して実行

- REPL で逐次実行

- ノートブック (JupyterLabなど) で実行

- VS Code から実行

REPL の便利機能（１）

- 起動方法

- ファイルを与えずに実行
 - `> julia`

- **juliaモード**（デフォルト）

- juliaプログラムを逐次実行
- プロンプトが `julia>` になる
- `exit()` or `Crtl-D` で終了

- **パッケージモード**

- juliaモードでカーソルが先頭の状態で、**]** **キー**を押して起動
- プロンプトが `(@v1.12) pkg>` などになる
- パッケージ管理（導入・更新・削除など）
- カーソルが先頭の状態で、バックスペースキーを押すと終了

REPL の便利機能（２）

- シェルモード

- juliaモードでカーソルが先頭の状態で、**;** **キー**を押して起動
- プロンプトが `shell>` になる
- 環境変数SHELLで指定したシェルが起動する
- カーソルが先頭の状態で、バックスペースキーを押すと終了

- ヘルプモード

- juliaモードでカーソルが先頭の状態で、**?** **キー**を押して起動
- プロンプトが `help>` になる
- Juliaおよびパッケージに関するヘルプドキュメントを表示
 - 例：`help> println`
 - ✓ 関数 `println()` のドキュメントが表示される
- カーソルが先頭の状態で、バックスペースキーを押すと終了

Julia の文法... は説明しません

気になる人は以下の参考文献を見てください

- 後藤俊介「実践 Julia 入門」
技術評論社、656頁、2023年、3960円
- 進藤裕之「1 から始める Julia プログラミング」
コロナ社、206頁、2020年、2970円
- 2026年度 ITPASS実習資料



<https://www.gfd-dennou.org/member/hiroki/file/itpass2026/>

NetCDF 関連パッケージ（１）

- NetCDF.jl

- <https://github.com/meggart/NetCDF.jl>
- 古参のNetCDF I/O ライブラリ
- MATLAB-like な上位API と C-like な下位API を提供

```
using NetCDF
air = ncread("air.2024.nc", "air") # ncread(ファイル名, 変数名)
typeof(air)                       # Array{Float32, 4}
```

- 読み込んだデータは、素の Julia 配列になる

NetCDF 関連パッケージ (2)

- **NCDatasets.jl**

- <https://juliageo.org/NCDatasets.jl/dev/>
- 新しめの高機能 NetCDF I/O ライブラリ

```
using NCDatasets
ds = NCDataset("air.2024.nc") # ファイルを開く
air = ds["air"]              # 変数を読み込む
typeof(air)
```

```
CommonDataModel.CFVariable{Union{Missing, Float32}, 4,
NCDatasets.Variable{Float32, 4, NCDataset{Nothing, Missing}},
CommonDataModel.Attributes{NCDatasets.Variable{Float32, 4,
NCDataset{Nothing, Missing}}}, @NamedTuple{fillvalue::Nothing,
missing_values::Tuple{Float32}, scale_factor::Nothing,
add_offset::Nothing, calendar::Nothing, DTsource::Nothing,
maskingvalue::Missing}}
```

ただし、軸データは別々に扱わなければならない

YAXArrays.jl

- Yet-Another Xarray-like Julia Package
- <https://juliadatacubes.github.io/YAXArrays.jl/stable/>
- Max-Planckのリモセン研究者が中心になって開発
- **GPhys のようなもの**
- 特徴
 - 複数のデータ形式に対応（NetCDF、Zarr、GeoTIFFなど）
 - NetCDF I/O は NetCDF.jl に依存
 - メタデータ+軸付きの多次元配列
 - 遅延読み込み（評価時にはじめてメモリに読み込む）
 - チャンク処理（大規模データを小分けにして処理）
 - 地球科学データを前提にした設計（lon、lat、time などの軸名）
 - ただし、GPhysのように気象解析メソッドが用意されては、ない

YAXArrays.jl (NetCDFを開く)

```
using YAXArrays
using NetCDF
ds = open_dataset("air.2024.nc") # ファイルを開く
```

YAXArray Dataset

Shared Axes:

(↓ lon Sampled{Float32} 0.0f0:2.5f0:357.5f0 ForwardOrdered Regular Points,
→ lat Sampled{Float32} 90.0f0:-2.5f0:-90.0f0 ReverseOrdered Regular Points,
↗ level Sampled{Float32} [1000.0f0, ..., 10.0f0] ReverseOrdered Irregular Points,
■ time Sampled{DateTime} [DateTime("2024-01-01T00:00:00"), ...,
DateTime("2024-12-31T00:00:00")] ForwardOrdered Irregular Points)

Variables:

air

Properties: Dict{String, Any}("References" => "http://www.psl.noaa.gov/data/gridded/data.ncep.reanalysis.html", "history" => "created 2013/12 by Hoop (netCDF2.3)", "dataset_title" => "NCEP-NCAR Reanalysis 1", "platform" => "Model", "title" => "mean daily NMC reanalysis (2014)", "description" => "Data is from NMC initialized reanalysis\n(4x/day). It consists of most variables interpolated to\npressure surfaces from model (sigma) surfaces.", "Conventions" => "COARDS")

YAXArrays.jl (変数を取り出す)

```
using YAXArrays
using NetCDF
ds = open_dataset("air.2024.nc")    # ファイルを開く
air = ds["air"]                    # ds.air でも可
```

```
| 144x73x17x366 YAXArray{Union{Missing, Float32}, 4} mean Daily Air temperature |  
|----- dims -----|  
↓ lon Sampled{Float32} 0.0f0:2.5f0:357.5f0 ForwardOrdered Regular Points,  
→ lat Sampled{Float32} 90.0f0:-2.5f0:-90.0f0 ReverseOrdered Regular Points,  
↗ level Sampled{Float32} [1000.0f0, ..., 10.0f0] ReverseOrdered Irregular Points,  
▣ time Sampled{DateTime} [DateTime("2024-01-01T00:00:00"), ..., DateTime("2024-12-31T00:00:00")] ForwardOrdered Irregular  
Points  
|----- metadata -----|  
Dict{String, Any} with 15 entries:  
"GRIB_id"           => 11  
"level_desc"        => "Pressure Levels"  
"least_significant_digit" => 1  
"long_name"          => "mean Daily Air temperature"  
"GRIB_name"          => "TMP"  
"precision"          => 2  
"var_desc"           => "Air temperature"  
"dataset"            => "NCEP Reanalysis Daily Averages"  
"valid_range"        => Float32[150.0, 350.0]  
"units"              => "degK"  
"name"               => "air"  
"missing_value"      => -9.96921f36  
"parent_stat"        => "Individual Obs"  
"actual_range"       => Float32[169.075, 320.15]  
"statistic"          => "Mean"  
|----- loaded lazily -----|  
  
data size: 249.5 MB
```

YAXArrays.jl (切り出し)

- インデックスによる切り出し

```
air[:, :, 1, 1]          # : はその軸の全範囲を表す  
air[1:end, 1:end, 1, 1]  # end はインデックスの最後を表す
```

- 軸名による切り出し

```
air[lon=1, level=1]      # 軸名とインデックスで指定  
air[lon=At(0), level=At(1000)]  # At()を用いると軸の値で指定できる  
air[lon=Near(1), level=Near(1100)] # Near()は最も近い軸の値が選択される
```

- 日付軸の場合

```
using CFTime              # 日付を扱うためのパッケージをロードする  
air[time=At(DateTime(2024, 1, 1))]  #  
air[time=Near(DateTime(2023, 1, 1))] #
```

- 範囲切り出し

```
air[lon=Near(90:180), lat=Near(90:-1:0), level=At(1000), time=1]  
air[90..180, 0..90, 1, 1]          # Intervalオブジェクトによる指定  
air[35:73, 1:37, 1, 1]              # インデックスの範囲指定  
air[lon=35:73, lat=1:37, level=1, time=1]  # 軸名とインデックス範囲指定  
air[lon=90..180, lat=0..90, level=1, time=1] # 軸名とIntervalによる指定
```

YAXArrays.jl (簡単な演算)

- 四則演算・数学関数 | 配列演算が可能
 - `air2 = air .* air`
 - `log10air = log10.(air)`
- 基礎的な統計量 | using Statistics してから
 - `air_bar = mean(air; dims=:lon)`
- 大規模データのチャンク処理 | map 関数
 - `newval = map(x -> x^2, air)`
 - `air_bar = mapslices(mean, air, dims="lon")`
 - `mapCube`関数

```
function abs_uv(out, u, v)          # 処理したい内容の関数を定義する
    out .= sqrt.(u.*u .+ v.*v)      # 第1引数を出力結果を、第2引数以降に入力を並べるようにする
end

hwnd = mapCube(abs_uv, (u, v);      # 第1引数が関数、第2引数に入力をタプルで与える
                indims=(InDims("lon", "lat"), InDims("lon", "lat")),
                outdims=OutDims("lon", "lat"))
```

Julia の可視化ライブラリ

- **Plots.jl**

- Juliaの標準的な可視化ライブラリ
 - <https://docs.juliaplots.org/stable/>
- クイックルックには十分
- 様々なバックエンド（GR、Plotly、PythonPlot、PGFPlotsXなど）
 - × 2次元図の重ねがきに難あり、地図投影なし、微調整が難しい

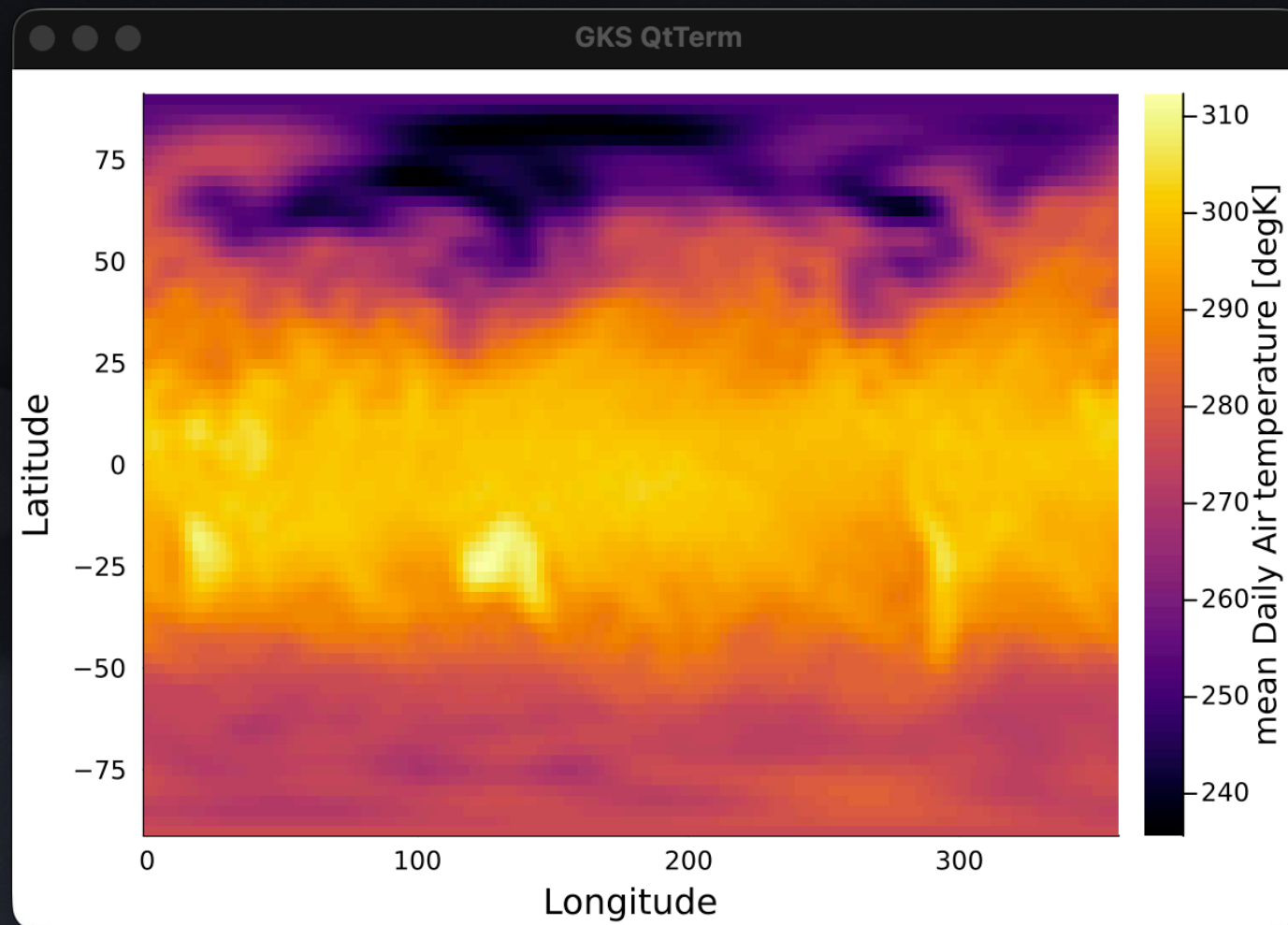
- **Makie.jl**

- より高機能な可視化ライブラリ（詳細は後述）
 - <https://makie.org/website/> [HP]
 - <https://docs.makie.org/stable/> [manual]
- 様々なバックエンド（GL、Cairo、WGL、RPR）
 - × Xwindow利用に難あり

Plots で YAXArray データを可視化

- 最少限の可視化コード

```
using YAXArrays
using NetCDF
using Plots
ds = open_dataset("air.2024.nc")
air = ds["air"]
heatmap(air[:, :, 1, 1])
```

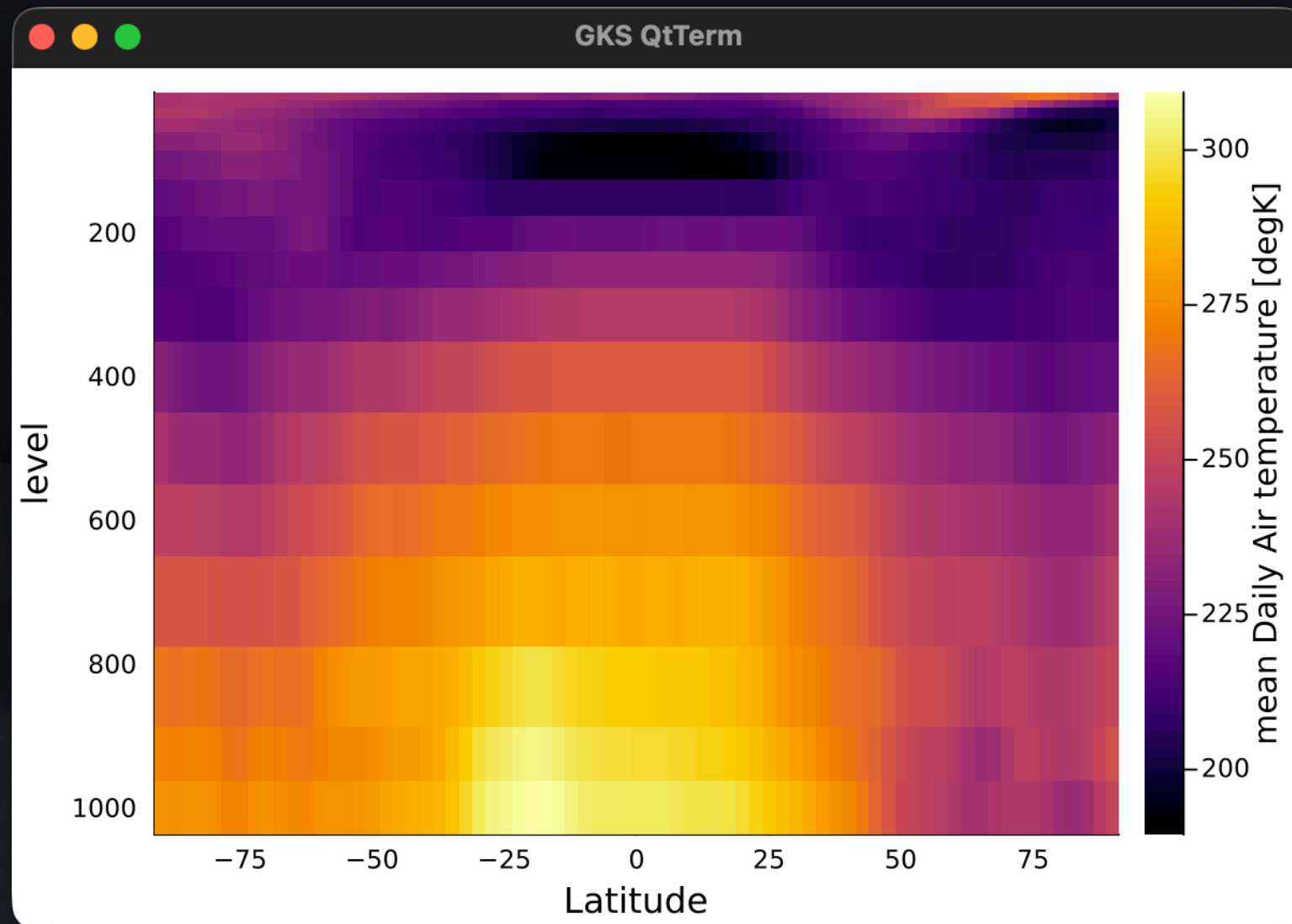


Plots で YAXArray データを可視化

- 鉛直断面

(つづき)

```
using CFTIME
air2d = air[lon=At(135),time=At(DateTime(2024,1,1))]
heatmap(air2d, yflip=true)
```

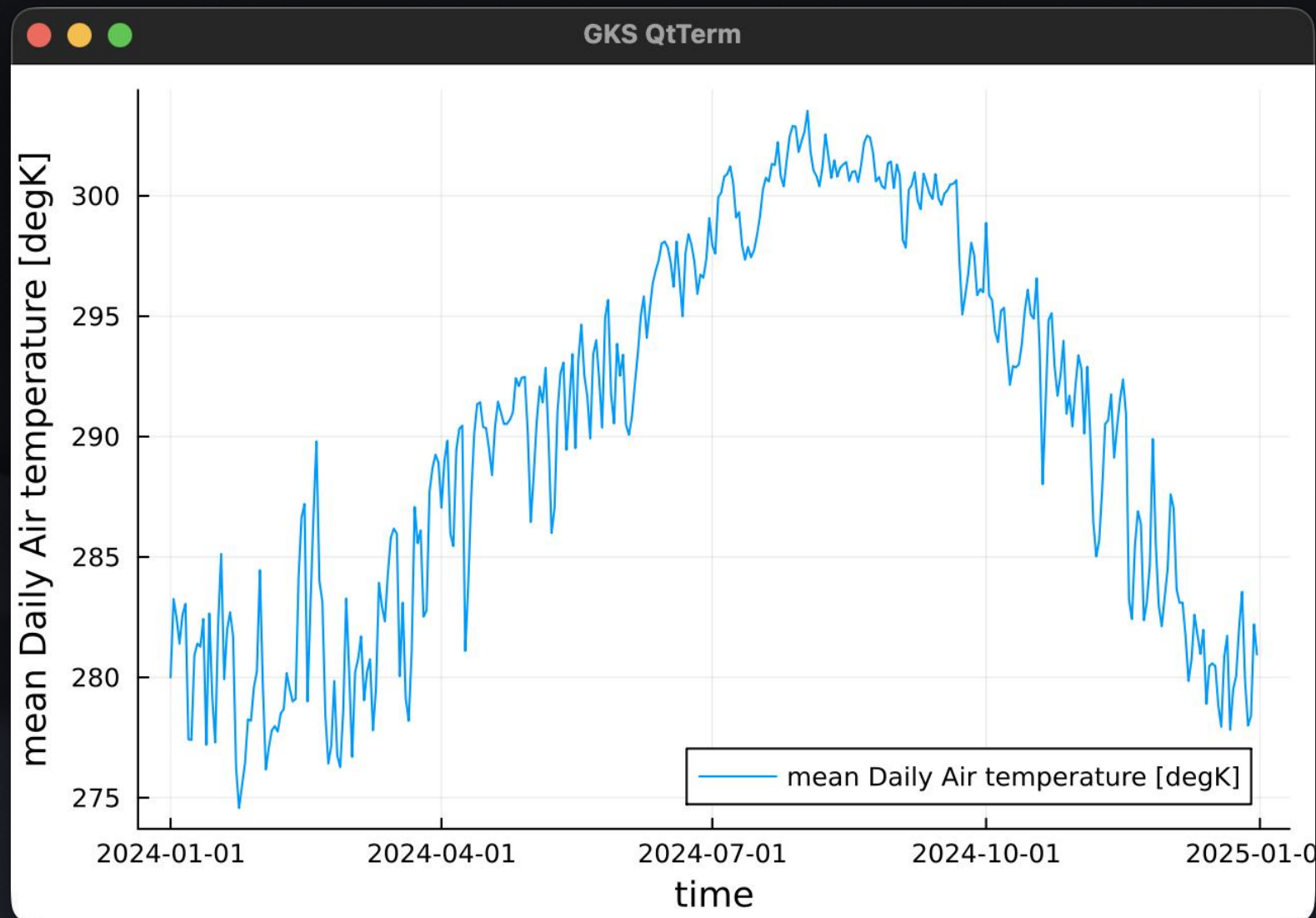


Plots で YAXArray データを可視化

- 1次元プロット

(つづき)

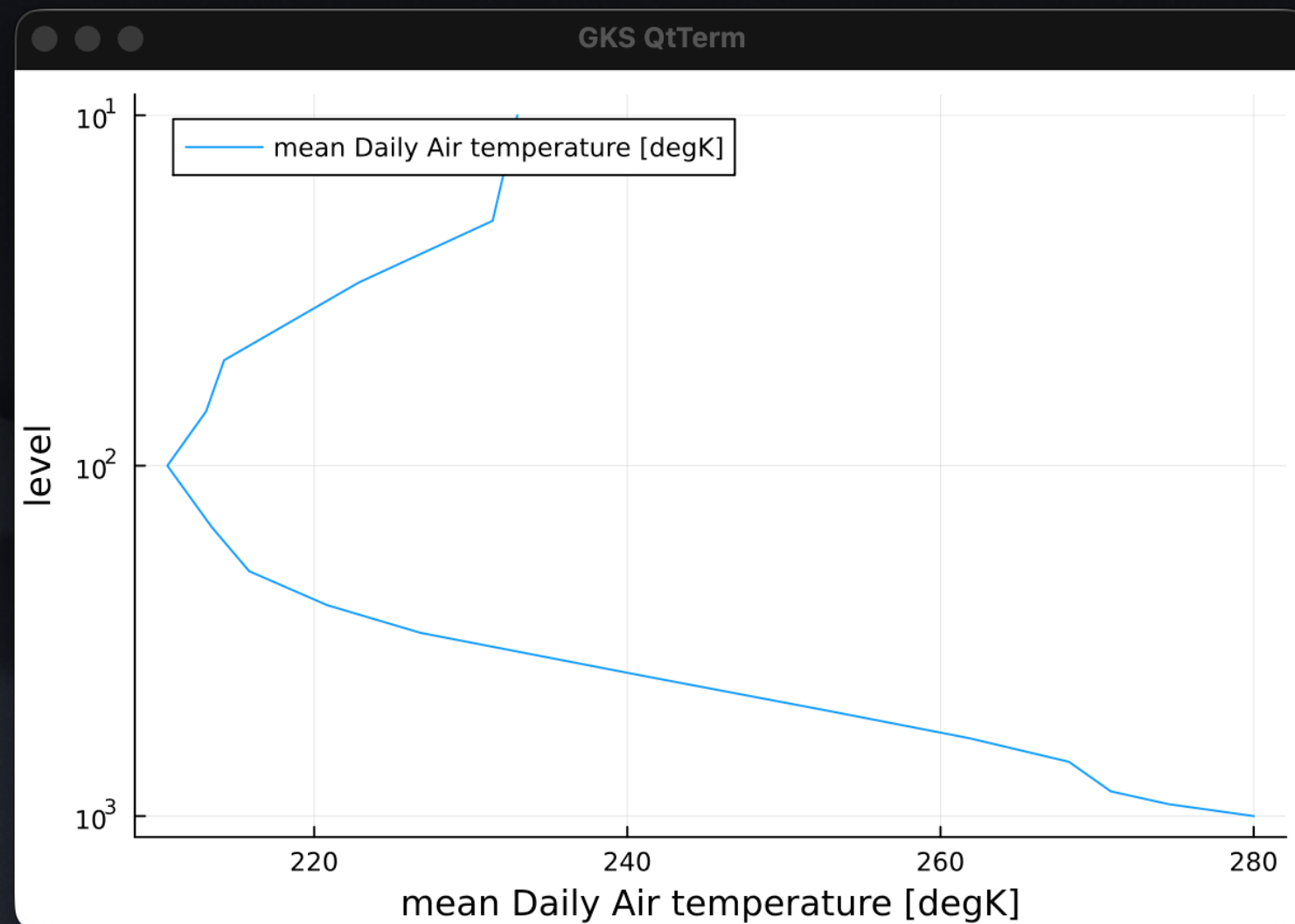
```
air1d = air[lon=At(135), lat=At(35), level=At(1000)]  
plot(air1d, legend=:bottomright)
```



Plots で YAXArray データを可視化

- 1次元プロット（その2）

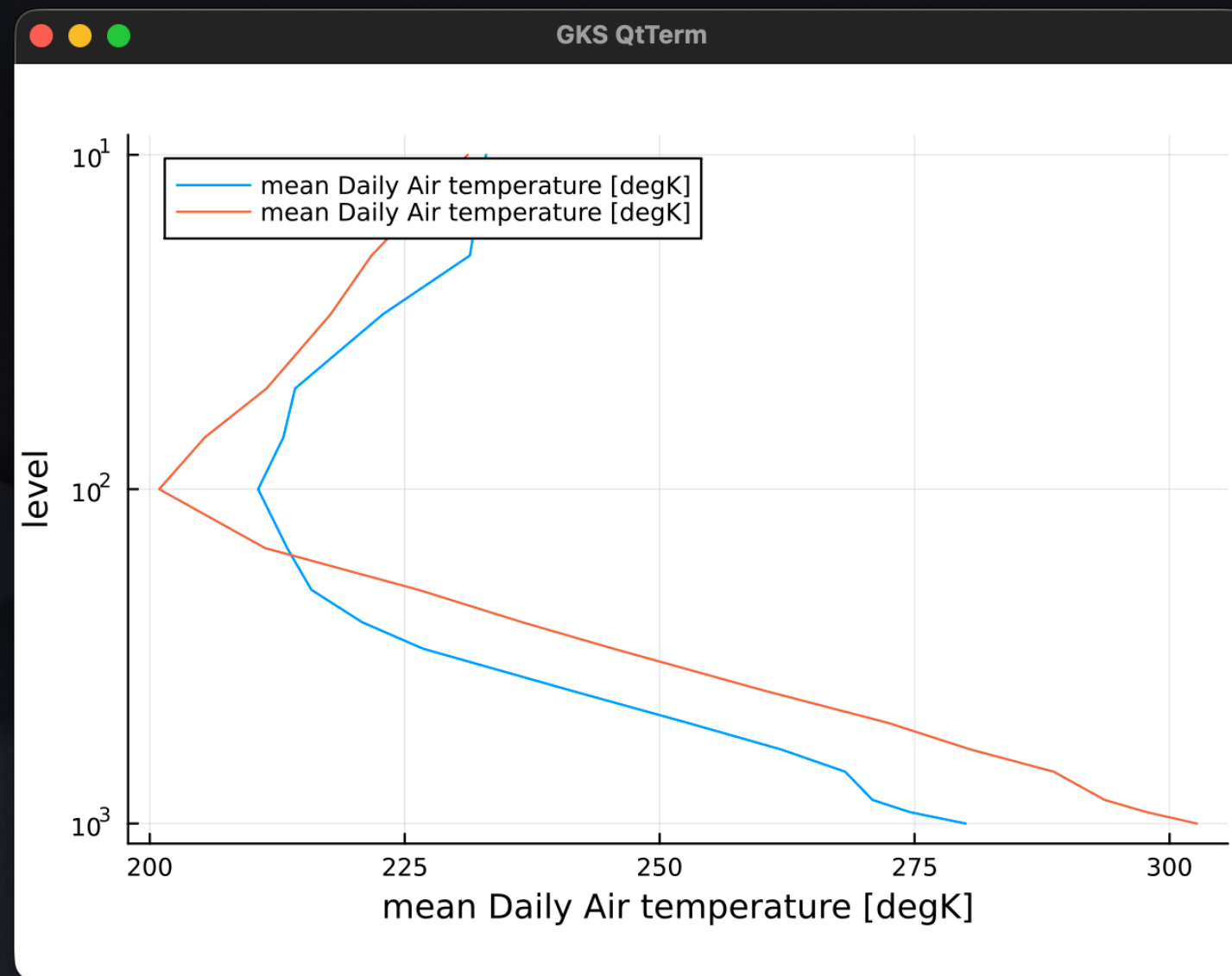
```
plot(air[lon=At(135), lat=At(35), time=At(DateTime(2024,1,1))], # 切り出し  
     permute=( :y, :x), # x/y軸の入れ替え  
     xflip=true,        # x軸（だった方）の反転  
     xscale=:log10)     # x軸（だった方）を対数軸に
```



Plots で YAXArray データを可視化

- 1次元プロットの重ねがき

```
plot(air[lon=At(135), lat=At(35), time=At(DateTime(2024,1,1))],  
      permute=( :y, :x), xflip=true, xscale=:log10) # 1つ目  
plot!(air[lon=At(135), lat=At(35), time=At(DateTime(2024,8,1))],  
       permute=( :y, :x), xflip=true, xscale=:log10) # 重ねるプロット
```



Makie (<https://makie.org/website/>)

- 主な特徴

- 調整の自由度が高い / GPUを使った高速描画 / 出版品質の描画 / 3D描画
- インタラクティブな描画 / 豊富な派生ライブラリ / 様々なバックエンド

- **GLMakie**

【手元のデータの可視化】

- GPUを活用して、大規模データの高速描画
- 2D描画、3D描画、インタラクティブな描画に対応

- **CairoMakie**

【論文用の図】

- 2D描画ライブラリCairoを利用して、高品位な2次元描画
- PDFやSVGなどのベクター形式での出力が可能

- **WGLMakie**

【リモートで可視化】

- Web技術を用いた描画で、Webブラウザで閲覧できる/動作する
- 2D描画、3D描画、インタラクティブな描画に対応

- **RPRMakie**

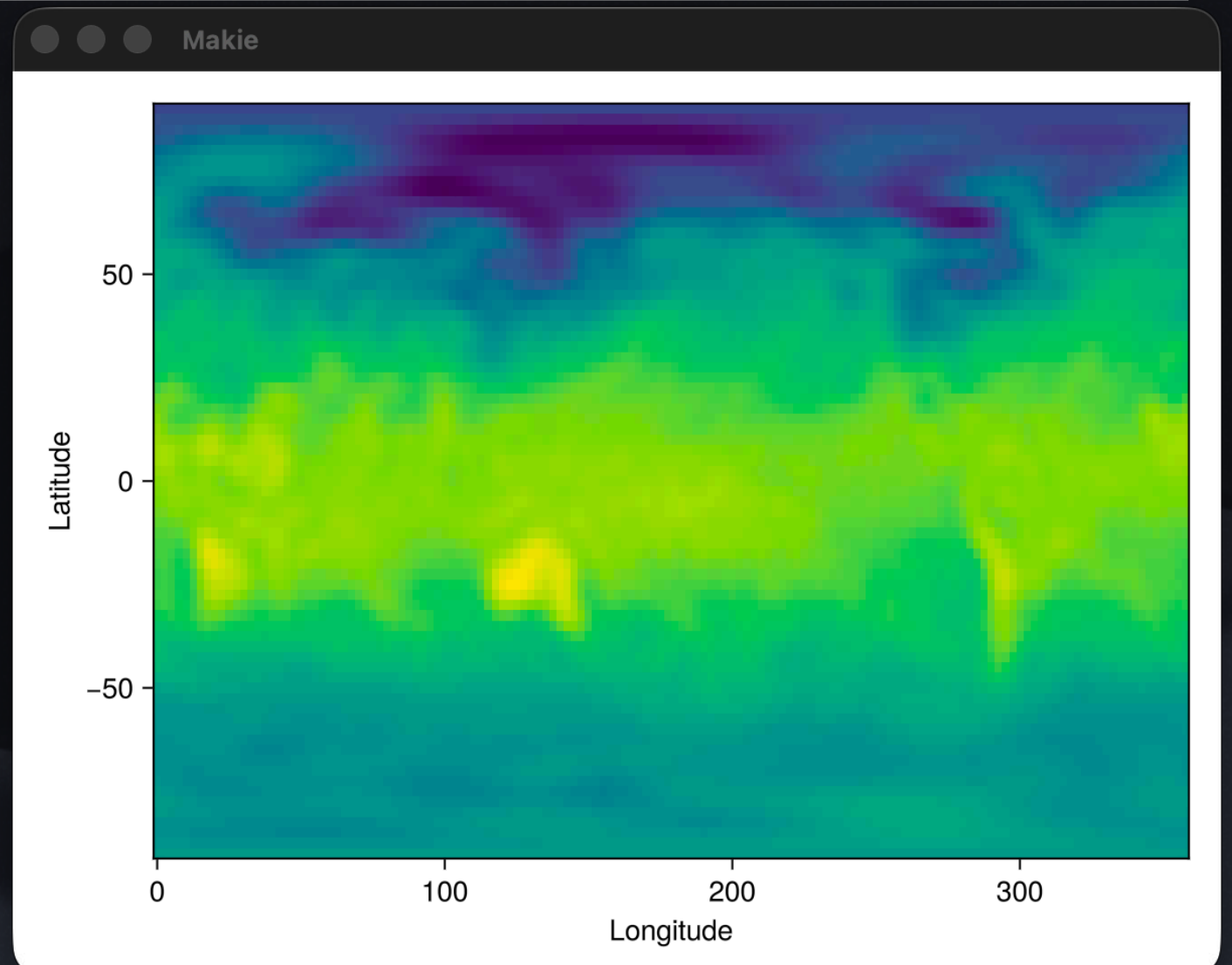
- レイトレーシングが可能な実験的バックエンド（らしい）

Makie で YAXArray データを可視化

- 最少限の可視化コード

```
using YAXArrays
using NetCDF
using GLMakie
air = open_dataset("air.2024.nc")["air"]
fig, ax, pl = heatmap(air[:, :, 1, 1])
```

- **fig** : **Figure**オブジェクト
 - 図を描く画面 (キャンバス)
- **ax** : **Axis**オブジェクト
 - 軸
 - 1つのFigureに対して複数OK
- **pl** : **Plot**オブジェクト
 - 種々のプロット
 - 1つの軸に対して複数OK



Makie で YAXArray データを可視化

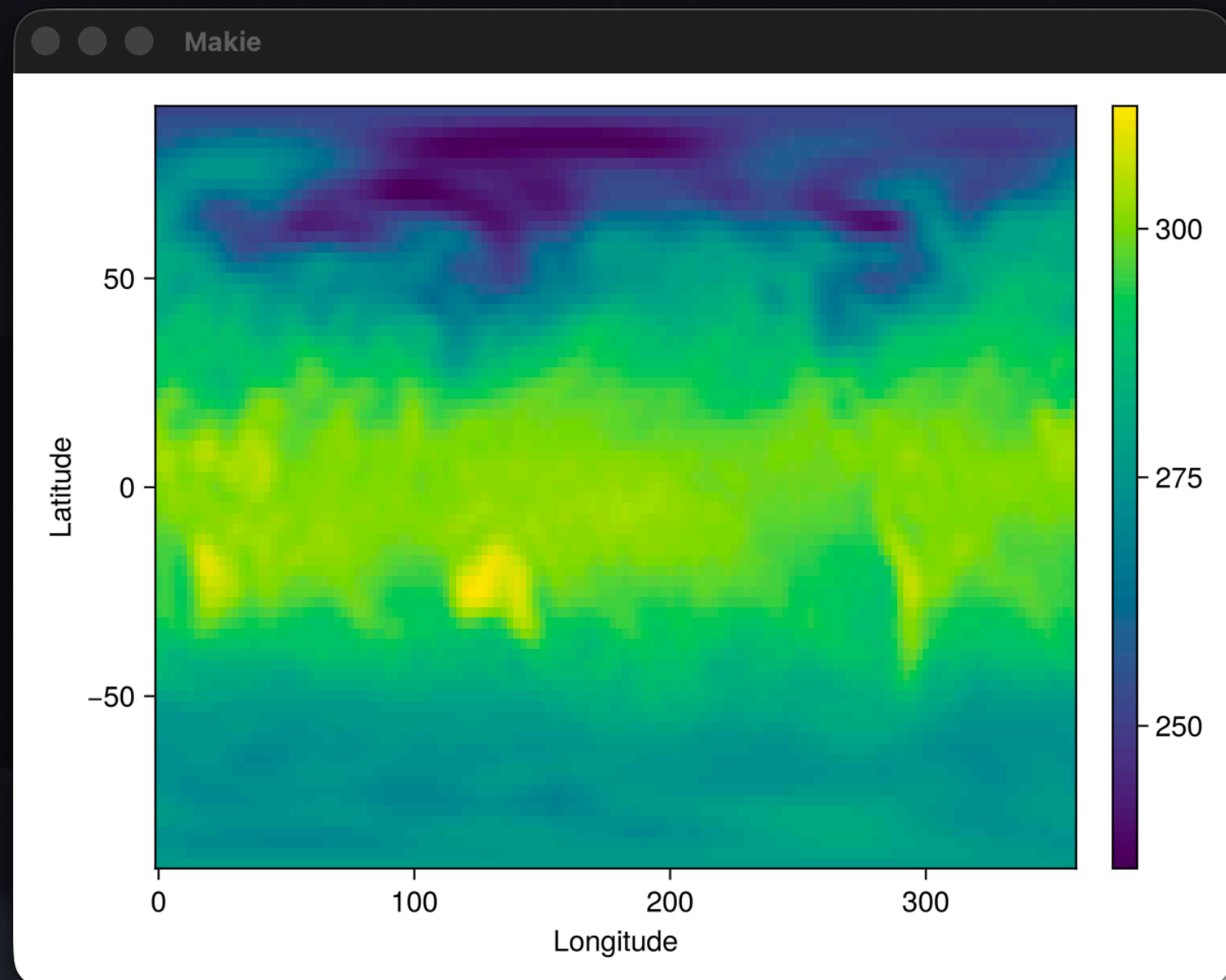
- カラーバーを追加する

(つづき)

```
cb = Colorbar(fig[1,2], pl)
```

- Figureオブジェクト fig の
1行2列の場所に、
Plotオブジェクト pl に対応
するカラーバーを描く
- 図の要素を後から追加/修正
できるのが Makie の強み

*NetCDFファイルによっては自動で
カラーバーが描画されることも



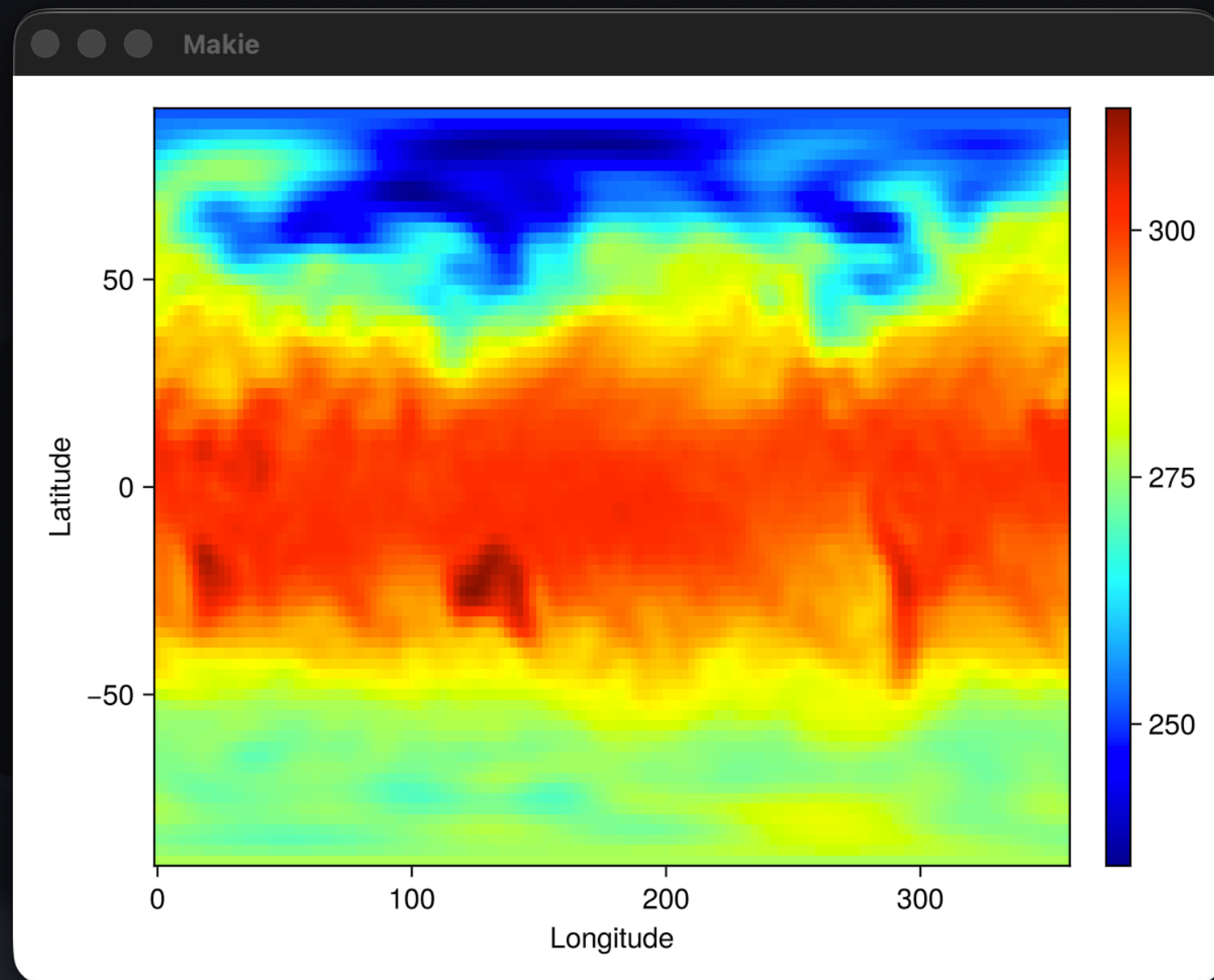
Makie で YAXArray データを可視化

- カラーマップを変更する

(つづき)

```
pl.colormap = :jet
```

- Plotオブジェクト pl が持つ colormap 属性を変更



Makie で YAXArray データを可視化

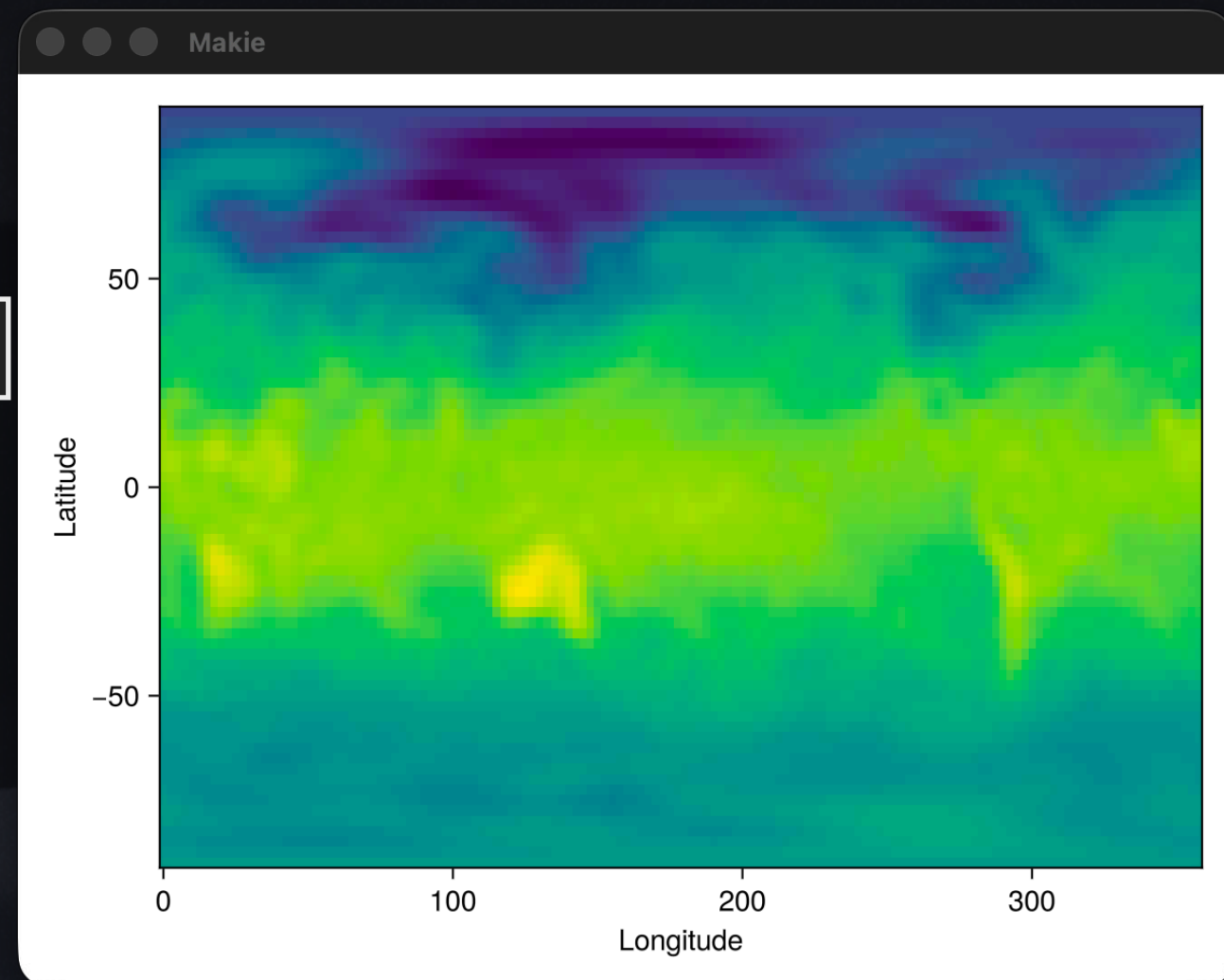
- オブジェクトを1つずつ生成する方法

```
fig = Figure()  
ax  = Axis(fig[1,1])  
pl  = heatmap!(ax, air[:, :, 1, 1])
```



同じ結果

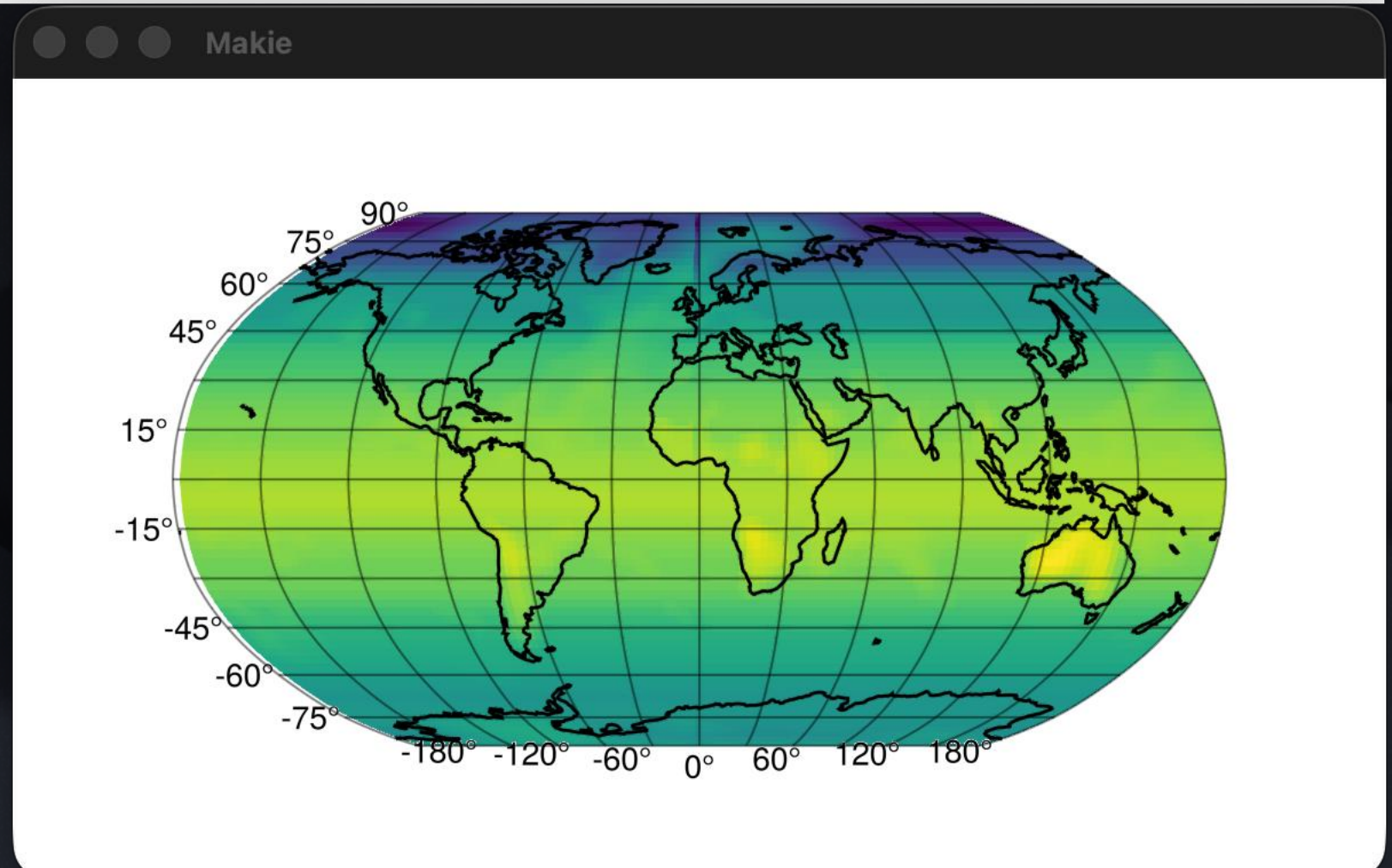
```
fig, ax, pl = heatmap(air[:, :, 1, 1])
```



GeoMakie で YAXArray データを可視化

```
using YAXArrays
using NetCDF
using GeoMakie
air = open_dataset("air.2024.nc")["air"]
fig = Figure()
ax = GeoAxis(fig[1,1], dest="proj=robin")
pl = surface!(ax, air[:, :, 1, 1], shading=NoShading, depth_shift=1,
interpolate = false)
cl = lines!(ax, GeoMakie.coastlines(), color=:black)
```

- 地図投影ライブラリ
PROJを利用している
が、すべてが使えるわけ
ではない
- 地図投影の便利さ・豊
富さは、DCLに劣る



こんなことができます集

① 高解像度データの滑らかな可視化

- 2万×1万点のデータの高速描画
- ズームイン/アウト、描画範囲の移動

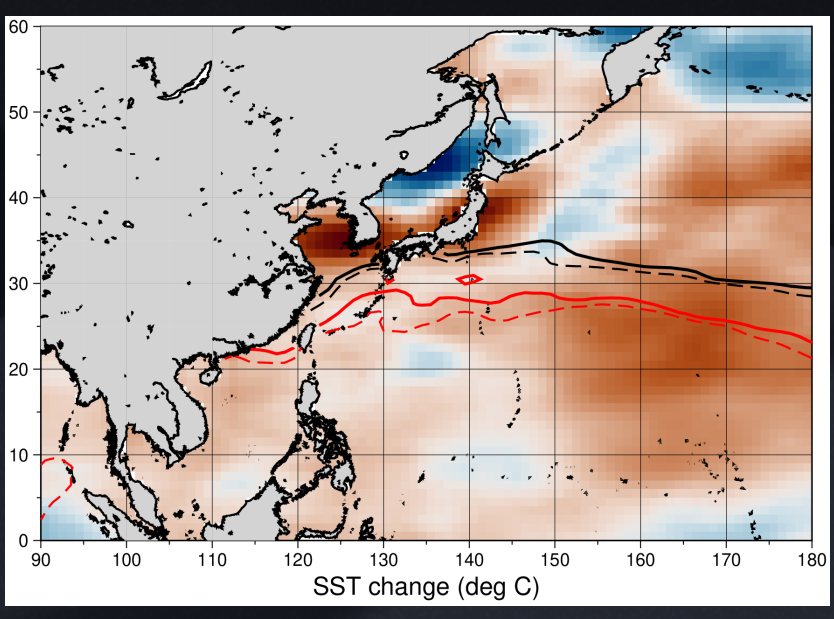
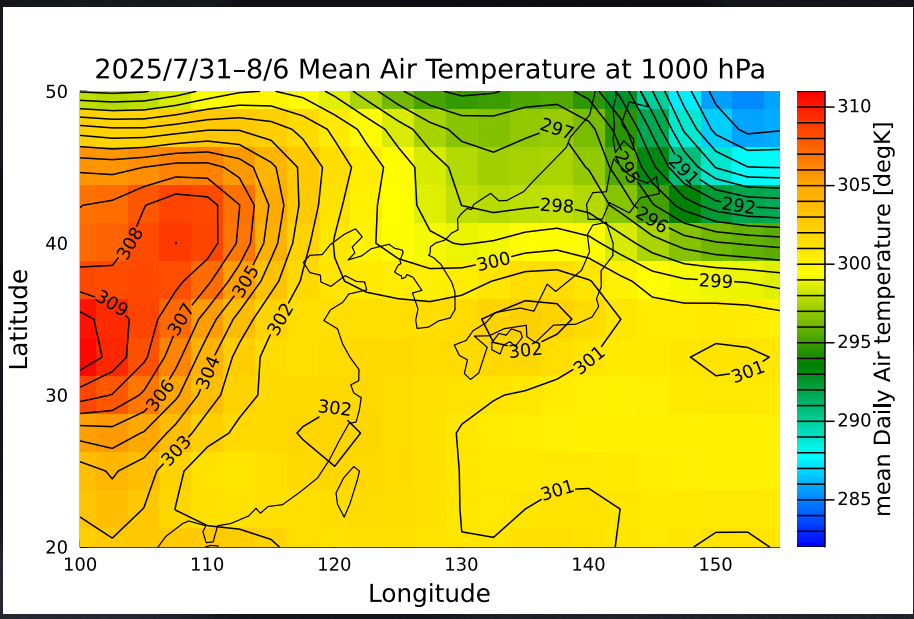
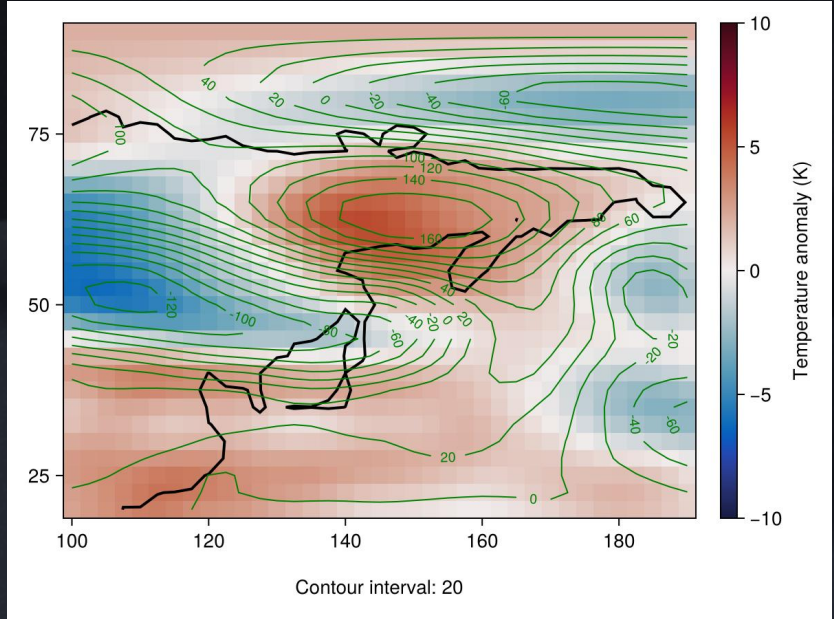
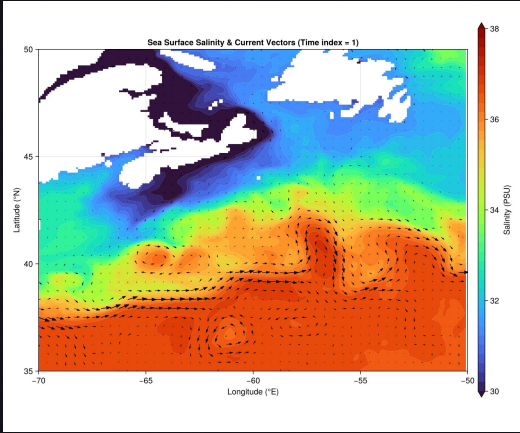
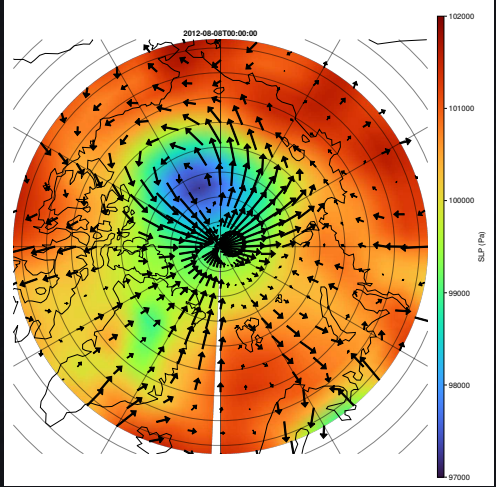
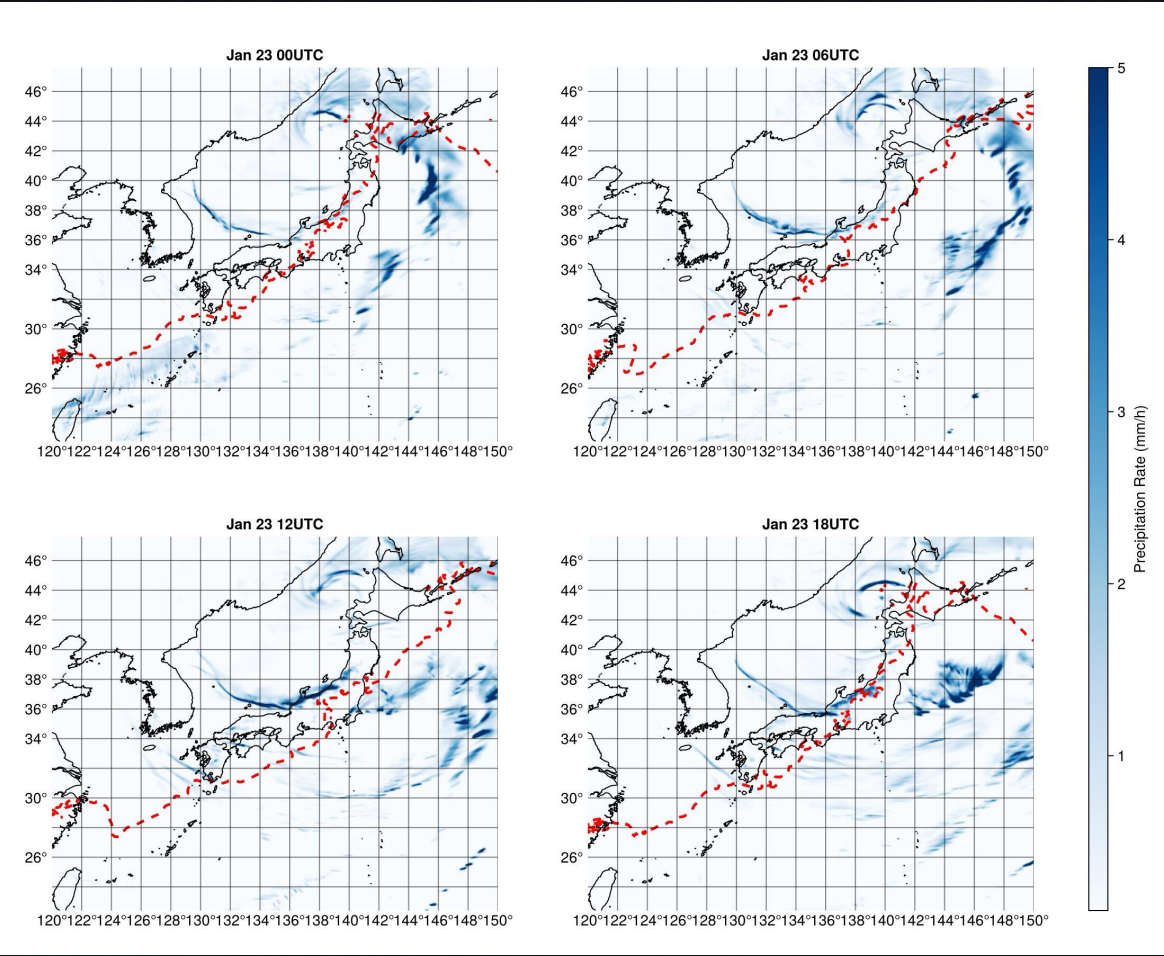
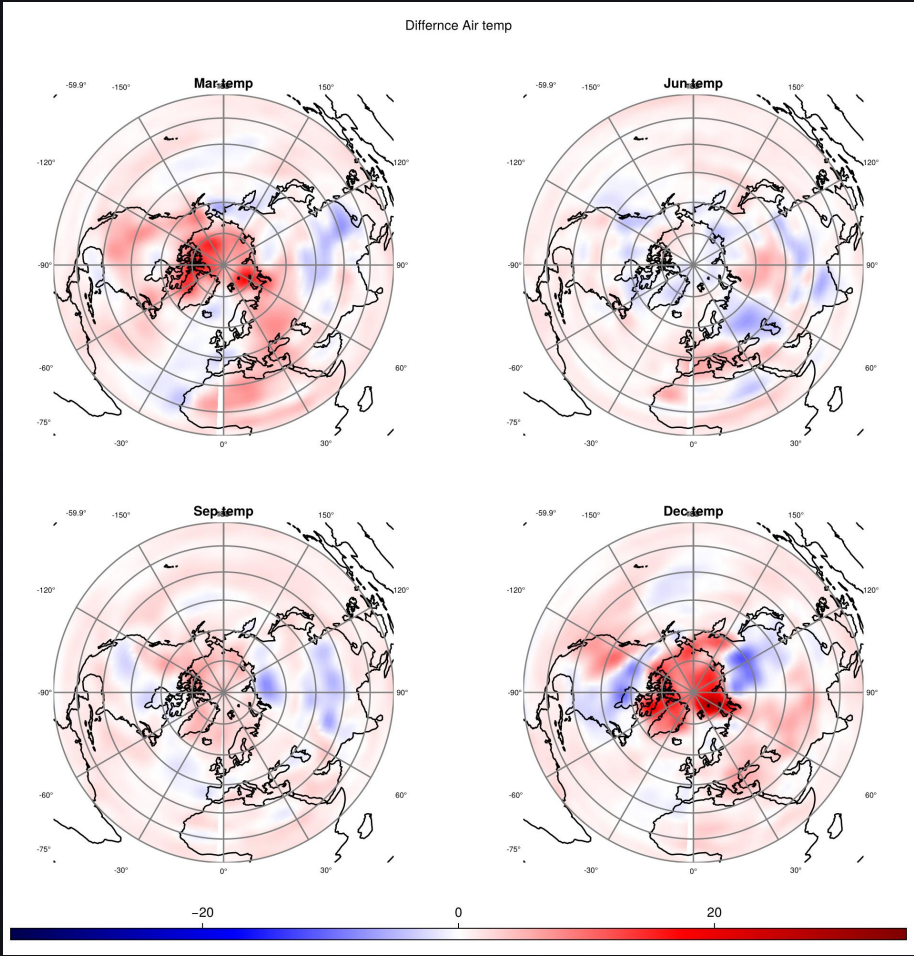
② アニメーション

- 再生 / 停止 / 逆再生

③ チェックボックスによるON/OFF

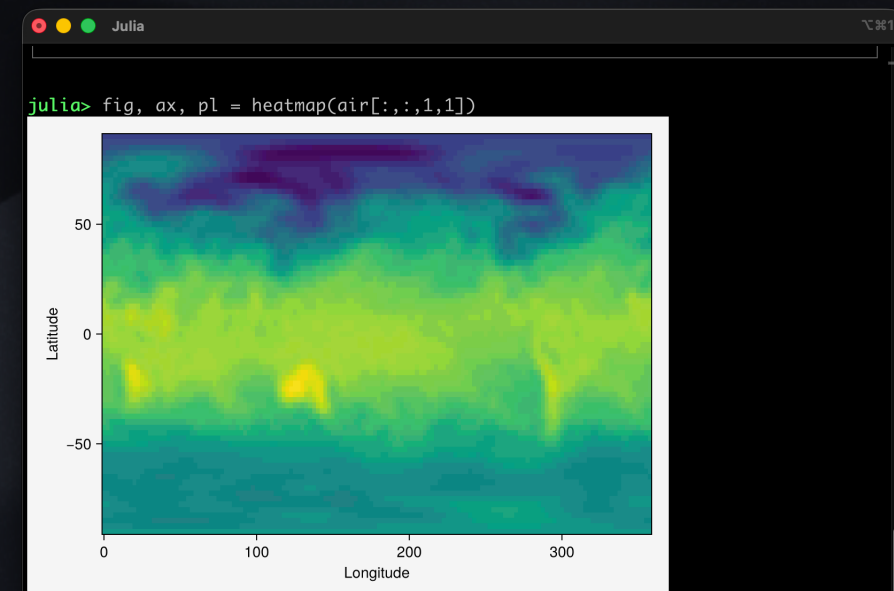
(公式サンプルをそのまま)

ギャラリー（研究室の学生作）



リモート描画

- GLMakie → ×?
 - VirtualGL というものを使えば可能らしい...
- WGLMakie → ◎
 - `ssh (-J SSHサーバ) -L 9384:localhost:9384` リモートサーバで接続して、ローカルのWebブラウザで、
<http://localhost:9384/browser-display> にアクセス
- CairoMakie → △
 - Xwindowでリモートの画像ビューアが表示される
- CairoMakie (Sixel) → ○
 - Sixel Graphics 対応ターミナルなら、ターミナル内に画像が表示される
- (Plots → ○)



まとめ

- Julia を用いた NetCDF データの解析・可視化方法を紹介
- Julia の特徴
 - インタープリタ型 / 動的型付け / 高速 / 関数が主役
 - 配列が Fortran 並び / 組込のパッケージシステム
 - 他言語呼び出し / 並列・分散処理
- 紹介したパッケージ
 - NetCDF入出力：**NetCDF.jl** Ruby対応物
← `ruby-netcdf`
 - 軸付多次元配列：**YAXArrays.jl** ← `GPhys` (気象解析メソッド除く)
 - 可視化ライブラリ
 - 簡単：**Plots.jl** ← `ruby-dcl`
 - 高機能：**Makie.jl** (GGraphほど自動化されてない)

まとめ（動機とメリット）

GPhys/DCCLの用途と不満点

→Julia/YAXArrays/Makie なら

A. 自分で研究に使う

- × 大規模データの処理が遅い
(2万×1万点の描画 ~ 十数分)
- × 複数言語依存 (Fortran/C/Ruby)
で開発維持のハードルが高い

○ 大規模データの処理が **高速**
(2万×1万点の描画 ~ **数十秒**)

○ **Julia で完結**
開発維持のハードルが低い？

B. 研究室の学生に使わせる

- × メソッドへの依存
- × gpview 出力をそのまま論文に

○ 関数自作 / 配列処理への回帰容易

○ 図の調整が容易 / 出版品質

C. 学部の数実習で、3年生に使わせる

- × 環境構築の難易度が上がってきた
- × 生成 AI による誤情報

○ **導入簡単&組込パッケージシステム**

○ 生成 AI による情報はまだマシ