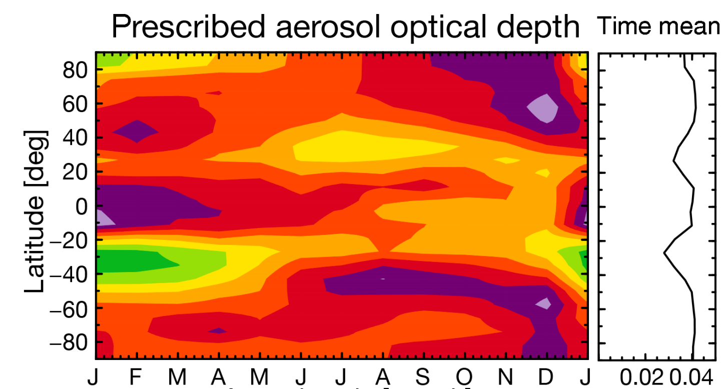
解析・可視化ツール gpv の紹介

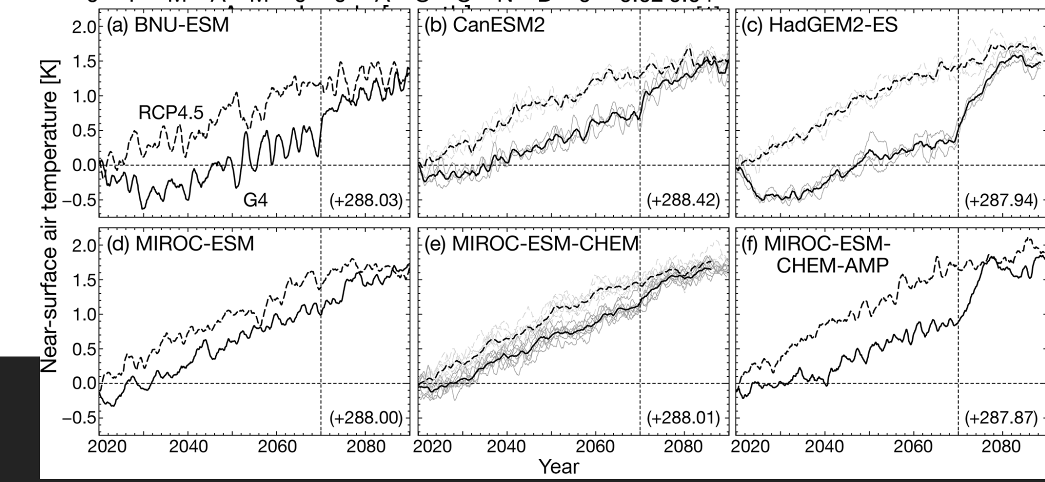
樫村 博基

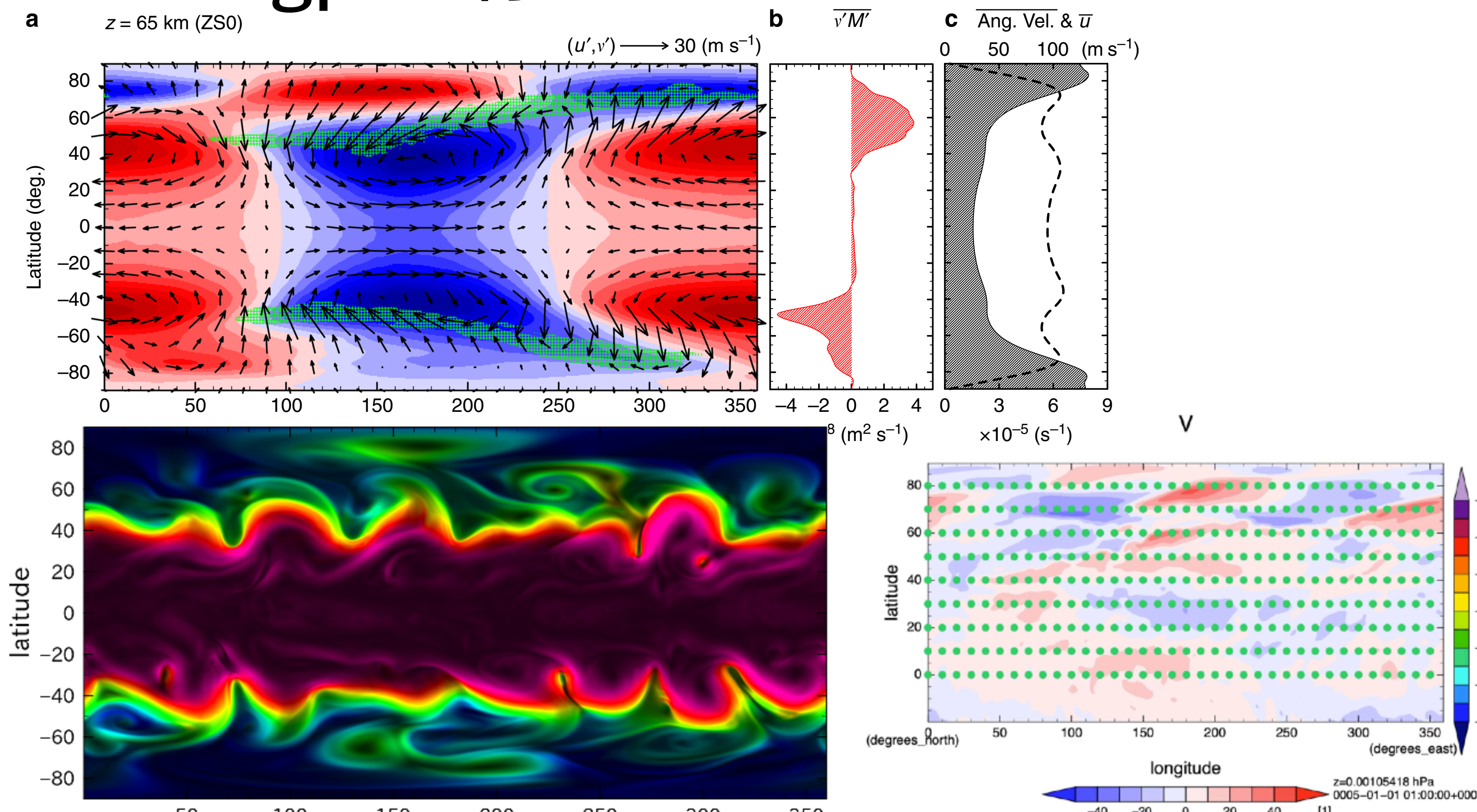
gpv とは？

- 樫村が2014年頃から、作成・使用している、解析・可視化コマンドラインツール
  - DCL、Ruby/GPhys ライブラリを使用
  - gpviewを建て増し改造し続けたもの
    - gpview: 974行　→　gpv: 9257行
  - 名前の由来
    - gpview と 6文字打つのが面倒だったので、はじめの3文字にした

- 2014年以降の解析・作図はほぼ全て、gpvで行っている







DCL～GPhys以外の依存ライブラリ

- RubyGems
  - pry｜対話的操作
  - parallel｜並列処理
  - ruby-progressbar｜並列処理の進行状況表示

- 外部コマンド（なくても依存する機能が使えないだけ）
  - ImageMagic｜画像メタデータ処理
  - Exiftool｜画像メタデータ処理
  - ssed｜画像メタデータ処理
  - pdfcrop (included in TexLive)｜PDFのクロップ
  - cpdf｜PDFの回転
  - FFmpeg｜動画作成

開発方針

- その場その場で必要になった機能を追加していく
  - 改造によって、これまでの機能が動作しなくなるかも知れない
    - 気づいたときに対処する

gpv_main.rb*	機能種類別にファイルを分割
gpv_analysis.rb	gpv_arrange.rb
gpv_sequence.rb	gpv_config.rb
gpv_gphysmod.rb	gpv_utils.rb
gpv_lagrange.rb	gpv_visualize.rb
gpv_spherical_harmonics_next.rb	

使い方

- gpview と同様
  - ただし利用可能なオプションが多い
    - gpview: 51個　→　gpv: 181個
  - オプション --mvo (multi-variable operation) が多くの解析で活躍
    - 複数変数を組み合わせた任意の処理を実行できる
    - シグマ座標のT, P から 温位θを計算する例（GAnalysisの関数を利用）

```
gpv T.ctl@T PonSig.nc@p --mvo “GAnalysis::Met::temp2theta(x,y)"
```

    - 与えたgturlの順に、x, y, z, u, v, w として使用できる
    - 座標変数も ax0, ax1, ax2 として利用可能

gpv開発の動機

- なぜコマンドライン?
  - スクリプトファイルだと
    - 異なるディレクトリに中身が微妙に違う同名のファイルができがち
    - 編集のためにいちいちエディタで開くのが面倒
  - シェルが記録してくれる
    - シェルが処理履歴を自動で記録してくれる
    - historyから検索可能
  - シェルにオプションの補完を教えられる
    - fishの場合 .config/fish/completions/gpv.fish
- なぜ同一コマンド?
  - 管理の手間が減る - 機能の足し算が可能
- 環境(OS)依存は?
  - gpv_config.rb に集約

運用方法

サーバ

手元のMac

自分で管理
(ubuntu)

他の人が管理
(debian)

自動同期型の
クラウドストレージ
(Dropbox, MEGA など)

git pull

git push

~~(プライベートレポジトリ)~~
公開した(2020.04.01)

大規模データへの対応策

- NArrayの2GB問題
  - numru-narray で回避可能だが、
    - あんまり速くない? (内部でOpenMP並列されるが…)
    - 油断するとマシンメモリを食い尽くす
  - gem: parallel による並列処理
    - gpv ではオブジェクトサイズを見て、自動処理
    - 処理の例

```
gparray = Parallel.map(da, :in_processes=>np){|i| #このブロックが並列処理される
  gp_subset = gp.cut_rank_conserving(dd=>i)  # サブセットを取り出す
  gp_subset.mean("t") # 処理を記述
}
gp_result = GPhys.join(gparray) # 分割されたものを１つにまとめる
```

    - 分割数が多すぎると、joinするのに時間がかかる

✓ 1処理が2GB未満でなるべく大きいほうがよい

大規模データへの対応策

- やっぱりメモリが足りない場合
  - 外部コマンド xargs と組み合わせて、並列処理→中間ファイル生成

①大規模データ T.ctl を1時刻ずつ処理して中間ファイルとして書き出す。
これを並列で処理する。

```
$ seq -w 0 23 | xargs -P 2 -I ii gpv T.ctl@T,t=^ii --eddy x --nc tmp_ii.nc --rename Teddy --nodraw --rank_conserving --silent --ntype sfloat
```

- 24時刻入っている想定（seq -w 0 23）
- 処理は2並列で進める（-P 2）

②時刻ごとに書き出された中間ファイル tmp_??.nc を1つに束ねる。

```
$ cdo -r mergetime tmp_??.nc tmp.nc
```

- gpvでjoinするよりcdoコマンドの方が速い

この図どうやって描いたんだっけ問題

- 図を残すときはPNGやPDFやEPSで出力する
  - gpvでは、ファイル出力した際に、その作図を行ったときの、カレントディレクトリとコマンドをメタデータとして図ファイルに自動で書き込む
- PNGの場合
  - 書き込み: system ("mogrify -comment '#{@comment}' dcl.png")
  - 読み出し: exiftool dcl.png | grep Comment
- PDFの場合
  - 書き込み: system ("exiftool -Title='#{@comment}' dcl.pdf -overwrite_original”)
  - 読み出し: PDFのプロパティ/タイトルを読む
- EPSの場合
  - 書き込み: system ("ssed -i -e '2i %%Comment: #{@comment}' dcl.eps”)
  - 読み込み: epsのヘッダ文字列を読む

参考

- ややで古いメモ
  - http://www.gfd-dennou.org/member/hiroki/homepage/main007.html
- 最新のオプション一覧
  - https://www.gfd-dennou.org/GFD_Dennou_Club/dc-arch/hiroki/gpv/gpv_options.txt
- githubの公開レポジトリ（2020.04.01に公開）
  - https://github.com/hiroki-mac/gpv