Fortran 90 数値解析ライブラリ (STPK) マニュアル (Ver.0.9.6.0)

辻野智紀

2013 年 2 月 24 日

目 次

第 1 章 概要 **1**

1.1 概要 1

1.1.1 本ライブラリの特徴 1

第 2 章 インストール・使用方法 **2**

2.1 インストール 2

2.1.1 アンインストール 3

2.2 使い方 3

2.3 本ライブラリの使用にあたって 4

第 3 章 サブルーチン一覧 **5**

3.1 alge_solv 5

3.1.1 Poisson_GauSei 5

3.1.2 Poisson_Jacobi 7

3.2 basis 9

3.2.1 c2i_convert 10

3.2.2 c2r_convert 10

3.2.3 i2c_convert 10

3.2.4 r2c_convert . . . 11
3.2.5 rand_make . . . 11
3.2.6 counter_day . . . 12
3.3 derivation . . . 13
3.3.1 curl . . . 13
3.3.2 curl_3d . . . 14
3.3.3 div . . . 16
3.3.4 div_3d . . . 17
3.3.5 grad_1d . . . 18
3.3.6 grad_2d . . . 19
3.3.7 grad_3d . . . 20
3.3.8 laplacian_1d . . . 21
3.3.9 zast_2_vel_2d . . . 21
3.3.10 zast_2_vel_3d . . . 23
3.4 ffts . . . 24
3.4.1 c2r_ffttp_1d . . . 24
3.4.2 ffttp_1d . . . 25
3.4.3 ffttp_2d . . . 27
3.4.4 prim_calc . . . 28
3.4.5 r2c_ffttp_1d . . . 28
3.4.6 rotate_calc . . . 30
3.5 file_operate . . . 30

3.5.1 auto_read_file . . . 31
3.5.2 line_number_counter . . . 32
3.5.3 read_file_grads . . . 32
3.5.4 request_axis_grads . . . 33
3.5.5 request_dim_grads . . . 34
3.5.6 request_vardim_grads . . . 34
3.5.7 read_file . . . 35
3.5.8 read_file_3d . . . 35
3.5.9 read_file_gtool3 . . . 36
3.5.10 read_file_gtool3_header_c . . . 37
3.5.11 read_file_gtool3_header_i . . . 37
3.5.12 read_file_gtool3_header_r . . . 38
3.5.13 read_file_text . . . 39
3.5.14 read_mgdsst . . . 39
3.5.15 write_file . . . 40
3.5.16 write_file_3d . . . 41
3.6 math_const . . . 41
3.6.1 rotate_array . . . 42
3.7 matrix_calc . . . 42
3.7.1 Gau_Sei . . . 43
3.7.2 Jacobi_algebra . . . 43
3.7.3 Jacobi_eigen . . . 44

3.7.4 LU_devs . . . 45
3.7.5 SOR_Gau_Sei . . . 45
3.7.6 SOR_Jacobi_algebra . . . 46
3.7.7 determ_2d . . . 47
3.7.8 eigenvalue_power . . . 48
3.7.9 gausss . . . 48
3.7.10 invert_mat . . . 49
3.7.11 schumit_norm . . . 49
3.7.12 trans_mat . . . 50
3.8 max_min . . . 50
3.8.1 max_val_1d . . . 51
3.8.2 max_val_2d . . . 51
3.8.3 max_val_3d . . . 52
3.8.4 min_val_1d . . . 52
3.8.5 min_val_2d . . . 53
3.8.6 min_val_3d . . . 53
3.9 phys_const . . . 54
3.10 poly_function . . . 54
3.10.1 chebyshev . . . 55
3.10.2 gegenbauer . . . 55
3.10.3 hermite . . . 56
3.10.4 jacobi_poly . . . 56

3.10.5 laguerre . . . 57
3.10.6 legendre . . . 58
3.10.7 sonine . . . 58
3.11 special_function . . . 59
3.11.1 Full_Ellip1_Func . . . 59
3.11.2 Full_Ellip2_Func . . . 59
3.11.3 bessj . . . 60
3.11.4 bessy . . . 60
3.11.5 beszero . . . 61
3.11.6 beta_func . . . 62
3.11.7 delta . . . 62
3.11.8 df_bessj . . . 63
3.11.9 df_bessy . . . 63
3.11.10 epsilon . . . 64
3.11.11 gamma_func . . . 64
3.11.12 kaijo . . . 65
3.11.13 sp_bessj . . . 65
3.11.14 sp_bessy . . . 66
3.12 statistics . . . 66
3.12.1 Anomaly_1d . . . 66
3.12.2 Anomaly_2d . . . 67
3.12.3 Anomaly_3d . . . 68

3.12.4 auto_interpolation_1d 68
3.12.5 auto_interpolation_2d 69
3.12.6 auto_interpolation_3d 70
3.12.7 Cor_Coe 70
3.12.8 Cor_Coe_2d 71
3.12.9 Cor_Coe_3d 72
3.12.10 LSM 72
3.12.11 LSM_2d 73
3.12.12 LSM_3d 74
3.12.13 LSM_poly 74
3.12.14 LSM_poly_2d 75
3.12.15 LSM_poly_3d 76
3.12.16 Mean_1d 76
3.12.17 Mean_2d 77
3.12.18 Mean_3d 77
3.12.19 Reg_Line 78
3.12.20 Reg_Line_2d 79
3.12.21 Reg_Line_3d 79
3.12.22 covariance 80
3.12.23 covariance_2d 81
3.12.24 covariance_3d 81
3.12.25 interpo_search_1d 82

3.12.26 interpo_search_2d 82
3.12.27 interpo_search_3d 83
3.12.28 interpolation_1d 84
3.12.29 interpolation_2d 85
3.12.30 interpolation_3d 86
3.12.31 nearest_search_1d 87
3.12.32 nearest_search_2d 88
3.12.33 nearest_search_3d 89
3.12.34 stand_vari 89
3.12.35 stand_vari_2d 90
3.12.36 stand_vari_3d 90
3.12.37 Move_ave 91
3.13 Thermo_Advanced_Function 92
3.13.1 CAPE 92
3.13.2 CIN 93
3.13.3 T_LFC 94
3.13.4 T_LNB 94
3.13.5 precip_water 95
3.13.6 qrsg_2_dbz 96
3.13.7 z_LCL 97
3.13.8 z_LFC 98
3.13.9 z_LNB 98

3.14 thermo_advanced_routine 99
3.14.1 Brunt_Freq 99
3.14.2 Ertel_PV 100
3.15 thermo_const 101
3.16 thermo_function 101
3.16.1 LH 102
3.16.2 RHTP_2_qv 102
3.16.3 RHT_2_e 103
3.16.4 TP_2_qvs 103
3.16.5 TP_2_rho 104
3.16.6 TqvP_2_TLCL 105
3.16.7 TqvP_2_thetae 105
3.16.8 TqvP_2_thetaes 106
3.16.9 eP_2_qv 107
3.16.10 eT_2_RH 107
3.16.11 es_Bolton 108
3.16.12 es_TD 108
3.16.13 exner_func_dry 109
3.16.14 get_gamma_d 109
3.16.15 goff_gratch 110
3.16.16 goff_gratch_i 110
3.16.17 hypsometric_form 111

3.16.18 TP_2_qv . . . 112
3.16.19 qvTP_2_RH . . . 112
3.16.20 rhoP_2_T . . . 113
3.16.21 rhoT_2_P . . . 113
3.16.22 tetens . . . 114
3.16.23 thetaP_2_T . . . 114
3.16.24 thetaT_2_P . . . 115
3.16.25 theta_dry . . . 116
3.16.26 theta_moist . . . 116
3.16.27 thetae_Bolton . . . 117
3.16.28 thetaes_Bolton . . . 117
3.17 Trajectory . . . 118
3.17.1 Backward_Traject_2d . . . 118
3.17.2 Backward_Traject_3d . . . 120
3.17.3 Forward_Traject_2d . . . 122
3.17.4 Forward_Traject_3d . . . 124
3.17.5 Stream_Line_2d . . . 126
3.17.6 Stream_Line_3d . . . 127
3.18 typhoon_analy . . . 129
3.18.1 grad_wind_pres . . . 129
3.18.2 hydro_grad_eqb . . . 130
3.18.3 tangent_mean_anom_scal . . . 131

3.18.4 tangent_mean_anom_scal_Cart . . . 133
3.18.5 tangent_mean_anom_vec . . . 134
3.18.6 tangent_mean_scal . . . 136
3.18.7 tangent_mean_vec . . . 138

第 4 章 サンプルプログラム 141

4.1 サンプルプログラムのコンパイル方法 . . . 141
4.2 各プログラムの説明 . . . 143
4.2.1 advection . . . 143
4.2.2 cov . . . 146
4.2.3 diffusion . . . 147
4.2.4 fft_test . . . 150
4.2.5 matrix_test . . . 154
4.2.6 normal_poly . . . 156
4.2.7 poison . . . 167
4.2.8 adjust . . . 169
4.2.9 read_mgdsst_nc . . . 173
4.2.10 SEQ . . . 174
4.2.11 thermo . . . 176
4.2.12 Thorpe . . . 178
4.2.13 sound_analysis . . . 178
4.2.14 wind . . . 183
4.2.15 NM01 . . . 187

第 5 章 付録 **195**

5.1 Statistics の付録 195

5.1.1 最小自乗法 195

5.1.2 線形内挿 197

5.2 付録 : サンプル 200

5.2.1 NM01 におけるモデルの詳細 200

5.2.2 adjust におけるモデルの詳細 201

5.2.3 adjust_2d におけるモデルの詳細 202

第1章 概要

1.1 概要

STPK ライブラリとは, Fortran 90 をベースとする数値解析ライブラリである。数値モデルの出力結果から様々な解析を行うために必要な処理をサブルーチンおよび関数という形で表現し、Fortran 90 のモジュールという概念を用いて用途別に関数を分類した関数集ライブラリである。Linux 系で動作するようにコーディングされているので、Linux 系のディストリビューションが多数を占める大学および研究機関の計算機、ワークステーション、果てはスーパーコンピュータにおいても問題なく動作することが期待される。また、近年計算機に使用されるプロセッサはデュアルコア、クアッドコアが当たり前となっている。そのような環境における高速な解析を実現するために、本ライブラリでは、OpenMP と呼ばれるスレッド並列命令を用いることで、一部処理の高速化を図っている。

1.1.1 本ライブラリの特徴

本ライブラリの特徴として以下のような事柄があげられる。

- lapack などの純粋な数学ライブラリと同様の処理を行う (例えば、連立一次方程式の求解や固有値問題) サブルーチンが充実している。これに関連して特に特殊関数を計算する関数、サブルーチンが非常に充実している。
- 流体力学において必須となる変数の空間微分等を計算するルーチンが充実している。
- 気象観測等の初期解析として必要となる熱力学関係の処理を行うルーチン (相当温位の計算等) が充実している。

以上のようなことから、本ライブラリを扱う上で必要な技能は、「プログラムのコンパイルができる」、「ルーチンと関数が適切に使える」のみである。これは、Fortran に限らず、プログラム言語を少し覚えて、簡単なプログラムなら書けるレベルとそれ以上のユーザーを対象としており、ライブラリとしての敷居が低いことが言えるであろう。

第2章　インストール・使用方法

2.1　インストール

本節では、STPK ライブラリのインストール方法を説明する。

1. tar.gz 形式のソースファイルを以下のコマンドで展開する。

```
$ tar zxvf ソースファイル.tar.gz
```

2. ソースファイルの中に configure ファイルが存在することを確認する。

3. コンパイル時に必要となる環境変数を設定する。ただし、用いるコンパイラによって設定する変数は様々であるので、ここでは以下のように例をあげておく。

Intel Fortran 版

```
FC = ifort, FCFLAGS='-assume byterecl (必 須) -convert big_endian
-openmp'
```

gfortran 版

```
FC = gfortran, FCFLAGS='-fconvert=big-endian -fopenmp'
```

Fujitsu Fortran 版

```
FC = frt, FCFLAGS='-KOMP -Am (必須)'
```

g95 版

```
FC = g95, FCFLAGS='-fendian=big'
```

ここで、"-convert big_endian", "-fconvert=big-endian", "-fendian=big" はともに、エンディアン設定である。また、"-openmp", "-fopenmp", "-KOMP"は OpenMP を有効にするオプションである。frt のみは、実行時オプションにてエンディアン変更を行う。さらに、gfortran にて、1 行の文字数制限によるエラーが出た際には、"-ffree-line-length-none" を追加する。また、ifort において、

OpenMP を有効にしたのち、OpenMP 由来の segmentation fault が発生する場合は、サブルーチンで stack に確保されるメモリ容量が大きすぎることが原因である可能性がある。その場合は、"-heap-arrays 数値" として、確保される配列の一部をヒープ領域に作成することで回避できる可能性がある。特に、3 次元配列を用いて大規模な解析を行う際には、用いるべきである。

4. 展開ディレクトリ内において、configure スクリプトによって Makefile を作成する。

```
$ ./configure --prefix=インストール先ディレクトリ --includedir=モジュールファイルインストール先
```

これらは、デフォルトでは、/usr/local にインストールされる。

5. make する。

```
$ make
```

6. make が正常に終了したら、make install でインストールを行う。

2.1.1 アンインストール

アンインストールは、インストール先のライブラリおよび、モジュールファイルを手動で削除すればよい。また、展開ディレクトリ内の config.log 等の中間生成ファイルは make distclean を実行することで削除可能である。

2.2 使い方

ここでは、適切に本ライブラリのモジュールを組み込み、適切にサブルーチンを用いたソースファイルがあるとして、そのソースをコンパイルすることを考える。このとき、プログラムのコンパイルには、以下の形式を自身の環境に合わせて適宜書き換えれば OK である。

```
コンパイラ名 -I(includedir のパス) ソースファイル名 -L(prefix/lib のパス) -lstpk
```

上を実行すると、a.out という実行形式のプログラムが作成されるはずである。ただし、コンパイル時のその他オプションはインストール時に FCFLAGS に設定したオプションをそのまま使用した方が無難である。

2.3 本ライブラリの使用にあたって

本ライブラリは自由に使用, ソースコードの改変を行って頂いて構いません. また, 本ライブラリを使用したことによって発生した損害等は開発者は一切責任を負いません. なお, 本ライブラリを用いた著作物を作成する際は「本研究の解析には数値解析ライブラリ STPK (Version.???) を用いた」と表記して頂ければ幸いである. また, 引用文献として記述する場合は, 「辻野智紀. 2012. Fortran 90 数値解析ライブラリ (STPK) マニュアル」と記述して頂ければ幸いである.

第3章 サブルーチン一覧

3.1 alge_solv

偏微分方程式を反復法を用いて計算するルーチン集.

3.1.1 Poisson_GauSei

機能

ガウスザイデル法（逐次反復法）を用いて、2 次元の非斉次 2 階線形偏微分方程式を計算する.

書式

```
call Poisson_GauSei( x, y, rho, eps, boundary, psi, [bound_opt], [a], [b], [c], [d], [e] )
```

引数

x	<R(:)>	in	右手系第一座標.
y	<R(:)>	in	右手系第二座標.
rho	<R(size(x),size(y))>	in	非斉次方程式の強制項（後述）.
eps	<R>	in	反復法の収束条件（後述）.
boundary	<C(4)>	in	4 辺の境界条件を与える（後述）.
psi	<R(size(x),size(y))>	inout	ポアソン方程式の解.
bound_opt	<R(size(x),size(y))>	in	境界での値（後述）.
			デフォルト：すべてゼロ.
a	<R(size(x),size(y))>	in	係数 1（後述）.
			デフォルト：すべて 1.
b	<R(size(x),size(y))>	in	係数 2（後述）.
			デフォルト：すべてゼロ.
c	<R(size(x),size(y))>	in	係数 3（後述）.
			デフォルト：すべて 1.
d	<R(size(x),size(y))>	in	係数 4（後述）.
			デフォルト：すべてゼロ.
e	<R(size(x),size(y))>	in	係数 5（後述）.
			デフォルト：すべてゼロ.

定義式

求める未知変数を ψ としたとき、以下の偏微分方程式を計算する。

$$a(x,y)\frac{\partial^2\psi}{\partial x^2} + b(x,y)\frac{\partial^2\psi}{\partial x\partial y} + c(x,y)\frac{\partial^2\psi}{\partial y^2} + d(x,y)\frac{\partial\psi}{\partial x} + e(x,y)\frac{\partial\psi}{\partial y} = \rho(x,y).$$

このとき、引数は以下の対応をする。

$$x : \texttt{x}, \quad y : \texttt{y}, \quad \rho : \texttt{rho}, \quad \psi : \texttt{psi}.$$

$$a : \texttt{a}, \quad b : \texttt{b}, \quad c : \texttt{c}, \quad d : \texttt{d}, \quad e : \texttt{e}.$$

備考

- 非斉次強制項 ρ は計算領域すべてについて、ゼロと設定すると、ラプラス方程式を計算することに相当する。
- 本ルーチンの収束条件は、反復の 1 ステップ前後での絶対誤差の最大値が閾値以下になったときに計算を終了するように設定しており、オプション引数 eps はその誤差の値を設定する。
- 本ルーチンでは、4 辺の境界条件を

  1 : 固定端境界 (ディリクレ型), 2 : 自由端境界 (ノイマン型), 3 : 周期境界条件

  の 3 種類で設定することができる。ここで、別のオプション引数 bound_opt を設定しておけば、その値で境界を強制することができる（固定端の場合は、引

数の値そのものが用いられ、自由端の場合は、境界を法線方向に流れるフラックスの値として引数の値が使用される。もちろん、周期境界ではこの値は設定しても反映されない)。この引数は 4 文字の文字型であり、それぞれの順番は図**??**のような対応をしている。ここで、引数は `bound\_opt='abcd'` の記号と対応している。

- 境界条件は自由に設定できるが、周期境界条件を設定した場合、その境界に隣接する反対側の境界も同一の境界条件の値を設定しておかなければならない。例えば、x 軸に平行な境界を周期境界とするのであれば、`boundary` の 1, 3 文字目は '3' でなければならない。そのようにされていない場合には、エラーを返す仕様になっている。
- 境界が固定端か自由端の場合、`bound\_opt` で設定される値のうち、境界上に存在する配列要素[*1]の値を参照して、固定端の場合はその値を直接固定端での値として、自由端の場合は、

$$\frac{\partial \psi}{\partial x} = f, \quad \frac{\partial \psi}{\partial y} = g$$

という境界条件である場合、f, g の値に相当する。つまり、このオプションにおいて、実際にルーチンではほとんどの配列が使用されないことになる。

- 偏微分方程式の各係数の値を表す `a, b, c, d, e` であるが、この値を省略すると、最高階の微分演算子に係る係数はすべて 1 に設定され、その他はすべてゼロに設定される(すなわち、純ポアソン方程式の計算となる)。これは、最高階の微分演算子までゼロに設定すると、方程式が特異系になり、この方法で処理すべき方程式の形ではなくなるための措置である。
- 偏微分方程式の係数は空間変化している場合でも計算することは可能であり、空間格子点が等間隔座標でない場合でも計算は可能である.

3.1.2 Poisson_Jacobi

機能

ヤコビ法を用いて、2 次元の非斉次 2 階線形偏微分方程式を計算する. OpenMP による並列処理が可能.

書式

```
call Poisson_Jacobi( x, y, rho, eps, boundary, psi, [bound_opt], [a],
[b], [c], [d], [e] )
```

引数

[*1] つまり、`bound_opt(1,:), bound_opt(size(x),:), bound_opt(:,1), bound_opt(:,size(y))` に設定されている値のこと。

x	<R(:)>	in	右手系第一座標.
y	<R(:)>	in	右手系第二座標.
rho	<R(size(x),size(y))>	in	非斉次方程式の強制項（後述）.
eps	<R>	in	反復法の収束条件（後述）.
boundary	<C(4)>	in	4 辺の境界条件を与える（後述）.
psi	<R(size(x),size(y))>	inout	ポアソン方程式の解.
bound_opt	<R(size(x),size(y))>	in	境界での値（後述）.
			デフォルト：すべてゼロ.
a	<R(size(x),size(y))>	in	係数 1（後述）.
			デフォルト：すべて 1.
b	<R(size(x),size(y))>	in	係数 2（後述）.
			デフォルト：すべてゼロ.
c	<R(size(x),size(y))>	in	係数 3（後述）.
			デフォルト：すべて 1.
d	<R(size(x),size(y))>	in	係数 4（後述）.
			デフォルト：すべてゼロ.
e	<R(size(x),size(y))>	in	係数 5（後述）.
			デフォルト：すべてゼロ.

定義式

求める未知変数を ψ としたとき、以下の偏微分方程式を計算する。

$$a(x,y)\frac{\partial^2\psi}{\partial x^2} + b(x,y)\frac{\partial^2\psi}{\partial x\partial y} + c(x,y)\frac{\partial^2\psi}{\partial y^2} + d(x,y)\frac{\partial\psi}{\partial x} + e(x,y)\frac{\partial\psi}{\partial y} = \rho(x,y).$$

このとき、引数は以下の対応をする。

$$x : \texttt{x}, \quad y : \texttt{y}, \quad \rho : \texttt{rho}, \quad \psi : \texttt{psi}.$$

$$a : \texttt{a}, \quad b : \texttt{b}, \quad c : \texttt{c}, \quad d : \texttt{d}, \quad e : \texttt{e}.$$

備考

- 非斉次強制項 ρ は計算領域すべてについて、ゼロと設定すると、ラプラス方程式を計算することに相当する。
- 本ルーチンの収束条件は、反復の 1 ステップ前後での絶対誤差の最大値が閾値以下になったときに計算を終了するように設定しており、オプション引数 eps はその誤差の値を設定する。
- 本ルーチンでは、4 辺の境界条件を

  1 : 固定端境界 (ディリクレ型), 2 : 自由端境界 (ノイマン型), 3 : 周期境界条件

  の 3 種類で設定することができる。ここで、別のオプション引数 bound_opt を設定しておけば、その値で境界を強制することができる（固定端の場合は、引

数の値そのものが用いられ、自由端の場合は、境界を法線方向に流れるフラックスの値として引数の値が使用される。もちろん、周期境界ではこの値は設定しても反映されない)。この引数は 4 文字の文字型であり、それぞれの順番は図**??**のような対応をしている。ここで、引数は `bound\_opt='abcd'` の記号と対応している。

- 境界条件は自由に設定できるが、周期境界条件を設定した場合、その境界に隣接する反対側の境界も同一の境界条件の値を設定しておかなければならない。例えば、x 軸に平行な境界を周期境界とするのであれば、`boundary` の 1, 3 文字目は '3' でなければならない。そのようにされていない場合には、エラーを返す仕様になっている。
- 境界が固定端か自由端の場合、`bound\_opt` で設定される値のうち、境界上に存在する配列要素[*2]の値を参照して、固定端の場合はその値を直接固定端での値として、自由端の場合は、

$$\frac{\partial \psi}{\partial x} = f, \quad \frac{\partial \psi}{\partial y} = g$$

という境界条件である場合、f, g の値に相当する。つまり、このオプションにおいて、実際にルーチンではほとんどの配列が使用されないことになる。
- 偏微分方程式の各係数の値を表す `a, b, c, d, e` であるが、この値を省略すると、最高階の微分演算子に係る係数はすべて 1 に設定され、その他はすべてゼロに設定される(すなわち、純ポアソン方程式の計算となる)。これは、最高階の微分演算子までゼロに設定すると、方程式が特異系になり、この方法で処理すべき方程式の形ではなくなるための措置である。
- 偏微分方程式の係数は空間変化している場合でも計算することは可能であり、空間格子点が等間隔座標でない場合でも計算は可能である.

3.2 basis

本ライブラリを使用する上で必要となるルーチンのうち、どのモジュールにも当てはまらなかった基本的な数学処理関数集である。

本モジュールには予約変数があり、現在は以下の表のように与えられている.

```
type dtime !  開始日の日付
integer ::  year_d !  西暦
integer ::  month_d !  月
integer ::  day_d !  日
end type dtime
```

[*2]つまり、`bound_opt(1,:), bound_opt(size(x),:), bound_opt(:,1), bound_opt(:,size(y))` に設定されている値のこと。

3.2.1 c2i_convert

機能

文字型を整数型に変換する。

書式

```
result = c2i_convert( cval )
```

引数

cval	<C(100)>	in	変換する文字.
戻り値	<I>	inout	変換された整数.

定義式

なし.

備考

なし.

3.2.2 c2r_convert

機能

文字型を実数型に変換する。

書式

```
result = c2r_convert( cval )
```

引数

cval	<C(100)>	in	変換する文字.
戻り値	<R>	inout	変換された実数.

定義式

なし.

備考

なし.

3.2.3 i2c_convert

機能

整数型を文字型に変換する。

書式

```
result = i2c_convert( ival, [forma] )
```

引数

ival	<I>	in	変換する整数.
forma	<C(*)>	in	変換時の形式.
戻り値	<C>	inout	変換された文字.

定義式

なし.

備考

なし.

3.2.4 r2c_convert

機能

実数型を文字型に変換する。

書式

```
result = r2c_convert( rval, [forma] )
```

引数

rval	<R>	in	変換する実数.
forma	<C(*)>	in	変換時の形式.
戻り値	<C>	inout	変換された文字.

定義式

なし.

備考

なし.

3.2.5 rand_make

機能

混合合同法を用いて擬似乱数を生成する。

書式

```
call rand_make( L, output )
```

引数

L	<I>	in	出力する最大桁数 +1 の数値.
output	<I>	inout	出力される乱数.

定義式

混合合同法は、数列 x_n について、整数 a, b を用いて、

$$x_n = \mathrm{mod}(ax_{n-1} + b, L)$$

という漸化式で表現される数列によって擬似乱数を生成するアルゴリズムである。ここで、L は整数である。a, b, L, x_0 については任意性が与えられているので、本ルーチンでは、

$$a = 11, b = 12$$

という値を採用した。また、L は使用者が必要とする任意の桁数 +1 の数値で求められ、x_0 は唯一性を重視するため、Fortran の組み込み関数である `time()` 関数を用いて得られる時刻を初期値として用いる。

備考

なし.

3.2.6 counter_day

機能

開始日から終了日までの日数をカウントする。

書式

```
result = counter_day( stime, etime )
```

引数

stime	<type(dtime)>	in	カウント開始の年月日.
etime	<type(dtime)>	in	カウント終了の年月日.
戻り値	<I>	inout	カウントされた日数.

定義式

なし.

備考

なし.

3.3 derivation

微分演算を有限差分で計算するルーチン集。本モジュールに組み込まれているルーチンでは、すべての微分計算において、2 次精度の中心差分近似を用いて微分を評価する。なお、評価点の不足する計算領域の端においては、1 次精度に差分の近似を落とすことで処理をする。

なお, すべてのルーチンで用いられているスケール因子の主な座標系での値と、任意の座標系におけるスケール因子の導出方法については、**??**参照。

3.3.1 curl

機能

2 次元ベクトルから渦度を計算する.

書式

```
call curl( x, y, u, v, val, [undef], [hx], [hy], [ord] )
```

引数

x	<R(:)>	in	右手系第一座標.
y	<R(:)>	in	右手系第二座標.
u	<R(size(x),size(y))>	in	x に対応するベクトル.
v	<R(size(x),size(y))>	in	y に対応するベクトル.
val	<R(size(x),size(y))>	inout	渦度.
undef	R	in	未定義値.
hx	<R(size(x),size(y))>	in	x 方向のスケール因子.
hy	<R(size(x),size(y))>	in	y 方向のスケール因子.
ord	L	in	微分計算の順番を入れ替える (備考参照).

定義式

互いに直交する基底ベクトル $(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$ をもつ曲線座標 (x_1, x_2, x_3) における, 独立した 2 ベクトルの回転 curl を計算する：

$$\mathrm{curl} \equiv \boldsymbol{e}_3 \cdot \boldsymbol{\nabla} \times \boldsymbol{u} = \frac{1}{h_1 h_2}\left[\frac{\partial(h_2 u_2)}{\partial x_1} - \frac{\partial(h_1 u_1)}{\partial x_2}\right]$$

ここで、$\boldsymbol{u} = u_1\boldsymbol{e}_1 + u_2\boldsymbol{e}_2 + u_3\boldsymbol{e}_3$ であり、h_1, h_2, h_3 はそれぞれ x_1, x_2, x_3 に対応するスケール因子である。このとき、引数は以下の対応をする。

$$x_1 : \mathtt{x}, \quad x_2 : \mathtt{y}, \quad u_1 : \mathtt{u}, \quad u_2 : \mathtt{v},$$

$$h_1 : \mathtt{hx}, \quad h_2 : \mathtt{hy}, \quad \mathrm{curl} : \mathtt{val}$$

デカルト座標系の場合、$h_1 = 1, h_2 = 1, h_3 = 1$ であるので、単純に

$$\mathrm{curl} = \frac{\partial u_2}{\partial x_1} - \frac{\partial u_1}{\partial x_2}$$

を計算することになる。

備考

- オプション `ord` は微分計算の後、正負を反転させるオプションであり、

  `.true.` : 通常計算, `.false.` : 正負反転

  となる。デフォルトは`.true.`。つまり、false を指定すれば、以下の計算を行うことと同義である。

$$\mathrm{curl} \equiv -\boldsymbol{e}_3 \cdot \boldsymbol{\nabla} \times \boldsymbol{u} = -\frac{1}{h_1 h_2}\left[\frac{\partial (h_2 u_2)}{\partial x_1} - \frac{\partial (h_1 u_1)}{\partial x_2}\right].$$

- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.3.2 curl_3d

機能

3 次元ベクトルから 3 次元渦度を計算する.

書式

```
call curl_3d( x, y, z, u, v, w, zeta, eta, xi, [undef], [hx], [hy], [hz]
)
```

引数

x	<R(:)>	in	右手系第一座標.
y	<R(:)>	in	右手系第二座標.
z	<R(:)>	in	右手系第三座標.
u	<R(size(x),size(y),size(z))>	in	x に対応するベクトル.
v	<R(size(x),size(y),size(z))>	in	y に対応するベクトル.
w	<R(size(x),size(y),size(z))>	in	z に対応するベクトル.
zeta	<R(size(x),size(y),size(z))>	inout	x に対応する回転.
eta	<R(size(x),size(y),size(z))>	inout	y に対応する回転.
xi	<R(size(x),size(y),size(z))>	inout	z に対応する回転.
undef	<R>	in	未定義値.
hx	<R(size(x),size(y),size(z))>	in	x 方向のスケール因子.
hy	<R(size(x),size(y),size(z))>	in	y 方向のスケール因子.
hz	<R(size(x),size(y),size(z))>	in	z 方向のスケール因子.

定義式

互いに直交する基底ベクトル $(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$ をもつ曲線座標 (x_1, x_2, x_3) における, 独立した 3 ベクトルの渦度 rot を計算する：

$$\begin{aligned}\mathrm{div} \equiv \boldsymbol{\nabla} \times \boldsymbol{u} = \frac{1}{h_1 h_2 h_3} \Bigg[& \boldsymbol{e}_1 \left\{ \frac{\partial (h_3 u_3)}{\partial x_2} - \frac{\partial (h_2 u_2)}{\partial x_3} \right\} \\ & + \boldsymbol{e}_2 \left\{ \frac{\partial (h_1 u_1)}{\partial x_3} - \frac{\partial (h_3 u_3)}{\partial x_1} \right\} \\ & + \boldsymbol{e}_3 \left\{ \frac{\partial (h_2 u_2)}{\partial x_1} - \frac{\partial (h_1 u_1)}{\partial x_2} \right\} \Bigg] = \qquad \zeta \boldsymbol{e}_1 + \eta \boldsymbol{e}_2 + \xi \boldsymbol{e}_3\end{aligned}$$

ここで、$\boldsymbol{u} = u_1 \boldsymbol{e}_1 + u_2 \boldsymbol{e}_2 + u_3 \boldsymbol{e}_3$ であり、h_1, h_2, h_3 はそれぞれ x_1, x_2, x_3 に対応するスケール因子である。このとき、引数は以下の対応をする。

$$x_1 \ : \ \texttt{x}, \quad x_2 \ : \ \texttt{y}, \quad x_3 \ : \ \texttt{z},$$

$$u_1 \ : \ \texttt{u}, \quad u_2 \ : \ \texttt{v}, \quad u_3 \ : \ \texttt{w},$$

$$h_1 \ : \ \texttt{hx}, \quad h_2 \ : \ \texttt{hy}, \quad h_3 \ : \ \texttt{hz}$$

$$\zeta \ : \ \texttt{zeta}, \quad \eta \ : \ \texttt{eta}, \quad \xi \ : \ \texttt{xi}.$$

デカルト座標系の場合、$h_1 = 1, h_2 = 1, h_3 = 1$ であるので、単純に

$$\mathrm{div} = \boldsymbol{e}_1 \left\{ \frac{\partial u_3}{\partial x_2} - \frac{\partial u_2}{\partial x_3} \right\} + \boldsymbol{e}_2 \left\{ \frac{\partial u_1}{\partial x_3} - \frac{\partial u_3}{\partial x_1} \right\} + \boldsymbol{e}_3 \left\{ \frac{\partial u_2}{\partial x_1} - \frac{\partial u_1}{\partial x_2} \right\}$$

を計算することになる。

備考

- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.3.3 div

機能

2 次元ベクトルから発散を計算する.

書式

```
call div( x, y, u, v, val, [undef], [hx], [hy] )
```

引数

x	<R(:)>	in	右手系第一座標.
y	<R(:)>	in	右手系第二座標.
u	<R(size(x),size(y))>	in	x に対応するベクトル.
v	<R(size(x),size(y))>	in	y に対応するベクトル.
val	<R(size(x),size(y))>	inout	発散.
undef	<R>	in	未定義値.
hx	<R(size(x),size(y))>	in	x 方向のスケール因子.
hy	<R(size(x),size(y))>	in	y 方向のスケール因子.

定義式

互いに直交する基底ベクトル $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ をもつ曲線座標 (x_1, x_2) における, 独立した 2 ベクトルの発散 div を計算する：

$$\mathrm{div} \equiv \boldsymbol{\nabla} \cdot \boldsymbol{u} = \frac{1}{h_1 h_2}\left[\frac{\partial(h_2 u_1)}{\partial x_1} + \frac{\partial(h_1 u_2)}{\partial x_2}\right]$$

ここで、$\boldsymbol{u} = u_1\boldsymbol{e}_1 + u_2\boldsymbol{e}_2$ であり、h_1, h_2 はそれぞれ x_1, x_2 に対応するスケール因子である。このとき、引数は以下の対応をする。

$$x_1 : \mathtt{x}, \quad x_2 : \mathtt{y}, \quad u_1 : \mathtt{u}, \quad u_2 : \mathtt{v},$$

$$h_1 : \mathtt{hx}, \quad h_2 : \mathtt{hy}, \quad \mathrm{div} : \mathtt{val}.$$

デカルト座標系の場合、$h_1 = 1, h_2 = 1$ であるので、単純に

$$\mathrm{div} = \frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2}$$

を計算することになる。

備考

- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.3.4 div_3d

機能

3 次元ベクトルから発散を計算する.

書式

```
call div_3d( x, y, z, u, v, w, val, [undef], [hx], [hy], [hz] )
```

引数

x	<R(:)>	in	右手系第一座標.
y	<R(:)>	in	右手系第二座標.
z	<R(:)>	in	右手系第三座標.
u	<R(size(x),size(y),size(z))>	in	x に対応するベクトル.
v	<R(size(x),size(y),size(z))>	in	y に対応するベクトル.
w	<R(size(x),size(y),size(z))>	in	z に対応するベクトル.
val	<R(size(x),size(y),size(z))>	inout	発散.
undef	<R>	in	未定義値.
hx	<R(size(x),size(y),size(z))>	in	x 方向のスケール因子.
hy	<R(size(x),size(y),size(z))>	in	y 方向のスケール因子.
hz	<R(size(x),size(y),size(z))>	in	z 方向のスケール因子.

定義式

互いに直交する基底ベクトル $(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$ をもつ曲線座標 (x_1, x_2, x_3) における, 独立した 3 ベクトルの発散 div を計算する：

$$\mathrm{div} \equiv \boldsymbol{\nabla} \cdot \boldsymbol{u} = \frac{1}{h_1 h_2 h_3} \left[\frac{\partial (h_2 h_3 u_1)}{\partial x_1} + \frac{\partial (h_3 h_1 u_2)}{\partial x_2} + \frac{\partial (h_1 h_2 u_3)}{\partial x_3} \right]$$

ここで、$\boldsymbol{u} = u_1 \boldsymbol{e}_1 + u_2 \boldsymbol{e}_2 + u_3 \boldsymbol{e}_3$ であり、h_1, h_2, h_3 はそれぞれ x_1, x_2, x_3 に対応するスケール因子である。このとき、引数は以下の対応をする。

$$x_1 \ : \ \mathtt{x}, \quad x_2 \ : \ \mathtt{y}, \quad x_3 \ : \ \mathtt{z},$$

$$u_1 \ : \ \mathtt{u}, \quad u_2 \ : \ \mathtt{v}, \quad u_3 \ : \ \mathtt{w},$$

$$h_1 \ : \ \mathtt{hx}, \quad h_2 \ : \ \mathtt{hy}, \quad h_3 \ : \ \mathtt{hz}$$

$$\mathrm{div} \ : \ \mathtt{val}.$$

デカルト座標系の場合、$h_1 = 1, h_2 = 1, h_3 = 1$ であるので、単純に

$$\mathrm{div} = \frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_3}{\partial x_3}$$

を計算することになる。

備考

- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.3.5 grad_1d

機能

任意変数の勾配を計算する.

書式

```
call grad_1d( x, u, val, [undef], [hx] )
```

引数

x	<R(:)>	in	空間座標.
u	<R(size(x))>	in	x の各点で定義される変数.
val	<R(size(x))>	inout	勾配値.
undef	<R>	in	未定義値.
hx	<R(size(x))>	in	x 方向のスケール因子.

定義式

座標 (x_1) における変数 u を計算する：

$$\mathrm{grad} = \frac{1}{h_1}\frac{\partial u}{\partial x_1}$$

このとき、引数は以下の対応をする：

$$x_1 : \mathtt{x}, \quad u_1 : \mathtt{u}, \quad h_1 : \mathtt{hx}, \quad \mathrm{grad} : \mathtt{val}.$$

デカルト座標系の場合、$h_1 = 1$ であるので、単純に

$$\mathrm{grad} = \frac{\partial u}{\partial x_1}$$

を計算することになる。

備考

- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.3.6 grad_2d

機能

任意変数の独立 2 次元勾配ベクトルを計算する.

書式

```
call grad_2d( x, y, u, valx, valy, [undeff], [hx], [hy] )
```

引数

x	<R(:)>	in	第一空間座標.
y	<R(:)>	in	第二空間座標.
u	<R(size(x),size(y))>	in	勾配を計算する変数.
valx	<R(size(x),size(y))>	inout	x 方向の勾配ベクトル.
valy	<R(size(x),size(y))>	inout	y 方向の勾配ベクトル.
undef	<R>	in	未定義値.
hx	<R(size(x),size(y))>	in	x 方向のスケール因子.
hy	<R(size(x),size(y))>	in	y 方向のスケール因子.

定義式

基底ベクトル $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ で定義される座標 (x_1, x_2) における変数 u を計算する：

$$\mathrm{grad} = \frac{\boldsymbol{e}_1}{h_1}\frac{\partial u}{\partial x_1} + \frac{\boldsymbol{e}_2}{h_2}\frac{\partial u}{\partial x_2}$$

このとき、引数は以下の対応をする。

$$x_1 \ : \ \mathtt{x}, \quad x_2 \ : \ \mathtt{y}, \quad u \ : \ \mathtt{u}, \quad h_1 \ : \ \mathtt{hx}, \quad h_2 \ : \ \mathtt{hy},$$

$$\frac{1}{h_1}\frac{\partial u}{\partial x_1} \ : \ \mathtt{valx}, \quad \frac{1}{h_2}\frac{\partial u}{\partial x_2} \ : \ \mathtt{valy}.$$

デカルト座標系の場合、$h_1 = 1, h_2 = 1$ であるので、単純に

$$\mathrm{grad} = \boldsymbol{e}_1\frac{\partial u}{\partial x_1} + \boldsymbol{e}_2\frac{\partial u}{\partial x_2}$$

を計算することになる。

備考

- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.3.7 grad_3d

機能

任意変数の独立 3 次元勾配ベクトルを計算する.

書式

```
call grad_3d( x, y, z, u, valx, valy, valz, [undeff], [hx], [hy], [hz]
)
```

引数

x	<R(:)>	in	第一空間座標.
y	<R(:)>	in	第二空間座標.
z	<R(:)>	in	第三空間座標.
u	<R(size(x),size(y),size(z))>	in	勾配を計算する変数.
valx	<R(size(x),size(y),size(z))>	inout	x 方向の勾配ベクトル.
valy	<R(size(x),size(y),size(z))>	inout	y 方向の勾配ベクトル.
valz	<R(size(x),size(y),size(z))>	inout	z 方向の勾配ベクトル.
undef	<R>	in	未定義値.
hx	<R(size(x),size(y),size(z))>	in	x 方向のスケール因子.
hy	<R(size(x),size(y),size(z))>	in	y 方向のスケール因子.
hz	<R(size(x),size(y),size(z))>	in	z 方向のスケール因子.

定義式

基底ベクトル $(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$ で定義される座標 (x_1, x_2, x_3) における変数 u を計算する：

$$\mathrm{grad} = \frac{\boldsymbol{e}_1}{h_1}\frac{\partial u}{\partial x_1} + \frac{\boldsymbol{e}_2}{h_2}\frac{\partial u}{\partial x_2} + \frac{\boldsymbol{e}_3}{h_3}\frac{\partial u}{\partial x_3}$$

このとき、引数は以下の対応をする。

$$x_1 : \mathtt{x}, \quad x_2 : \mathtt{y}, \quad x_3 : \mathtt{z}, \quad u : \mathtt{u}, \quad h_1 : \mathtt{hx}, \quad h_2 : \mathtt{hy}, \quad h_3 : \mathtt{hz}$$

$$\frac{1}{h_1}\frac{\partial u}{\partial x_1} : \mathtt{valx}, \quad \frac{1}{h_2}\frac{\partial u}{\partial x_2} : \mathtt{valy}, \quad \frac{1}{h_3}\frac{\partial u}{\partial x_3} : \mathtt{valz}$$

デカルト座標系の場合、$h_1 = 1$ であるので、単純に

$$\mathrm{grad} = \boldsymbol{e}_1\frac{\partial u}{\partial x_1} + \boldsymbol{e}_2\frac{\partial u}{\partial x_2} + \boldsymbol{e}_3\frac{\partial u}{\partial x_3}$$

を計算することになる。

備考

- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.3.8 laplacian_1d

機能

任意変数の勾配を計算する.

書式

```
call laplacian_1d( x, u, val, [undef] )
```

引数

x	<R(:)>	in	空間座標.
u	<R(size(x))>	in	x の各点で定義される変数.
val	<R(size(x))>	inout	2 階勾配値.
undef	<R>	in	未定義値.

定義式

座標 (x_1) における変数 u の 2 階微分を計算する：

$$\text{laplacian} = \frac{\partial^2 u}{\partial x_1^2}$$

このとき、引数は以下の対応をする：

$$x_1 \ : \ \texttt{x}, \quad u_1 \ : \ \texttt{u}, \quad \text{laplacian} \ : \ \texttt{val}.$$

備考

- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.3.9 zast_2_vel_2d

機能

terrain following 座標系（以下, terrain 系）で定義されるある 1 高度層において、3 次元風速成分をデカルト座標系成分に変換する。ただし、ここでは座標点の変換は行わない。あくまで terrain 系の格子点における各風速成分をデカルト系成分に変換するのみである。

書式

```
call zast_2_vel_2d( x, y, zast, u, v, w, uh, vh, wh, [undef] )
```

引数

x	<R(:)>	in	デカルト系第一座標.
y	<R(:)>	in	デカルト系第二座標.
zast	<R(size(x),size(y))>	in	terrain 系の任意 1 層 における鉛直高度 [m].
u	<R(size(x),size(y))>	in	zast に対応する東西風速 [m/s].
v	<R(size(x),size(y))>	in	zast に対応する南北風速 [m/s].
w	<R(size(x),size(y))>	in	zast に対応する鉛直風速 [m/s].
uh	<R(size(x),size(y))>	inout	変換後の東西風速 [m/s].
vh	<R(size(x),size(y))>	inout	変換後の南北風速 [m/s].
wh	<R(size(x),size(y))>	inout	変換後の鉛直風速 [m/s].
undef	R	in	未定義値.

定義式

本ルーチンは水平方向にはデカルト座標系のみを想定している。今後拡張予定。
terrain 系で定義されている東西、南北、鉛直風速を (U,V,W) とし、terrain 系における高度座標を z^* とすると、デカルト成分に変換したときの東西、南北、鉛直風速成分 (u,v,w) は以下のように変換される：

$$u = \frac{U}{\sqrt{1+\left(\frac{\partial z^*}{\partial x}\right)_z^2}}, \quad v = \frac{V}{\sqrt{1+\left(\frac{\partial z^*}{\partial y}\right)_z^2}},$$

$$w = W + \frac{U\left(\frac{\partial z^*}{\partial x}\right)_z}{\sqrt{1+\left(\frac{\partial z^*}{\partial x}\right)_z^2}} + \frac{V\left(\frac{\partial z^*}{\partial y}\right)_z}{\sqrt{1+\left(\frac{\partial z^*}{\partial y}\right)_z^2}}.$$

ここで、添字 z はデカルト座標系の z で評価された水平勾配であることを示している。また、このとき実際のプログラムにおける引数は以下の対応をする：

$$(u,v,w) \text{ : } \texttt{(uh,vh,wh)}, \quad (U,V,W) \text{ : } \texttt{(u,v,w)},$$

$$z^*(x,y) \text{ : } \texttt{zast(size(x),size(y))}, \quad (x,y) \text{ : } \texttt{(x,y)}.$$

備考

- 定義式の導出は付録**??**参照.
- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.3.10 zast_2_vel_3d

機能

terrain following 座標系（以下, terrain 系）で定義される 3 次元風速成分をデカルト座標系成分に変換する。ただし、ここでは座標点の変換は行わない。あくまで terrain 系の格子点における各風速成分をデカルト系成分に変換するのみである。

書式

```
call zast_2_vel_3d( x, y, zast, u, v, w, uh, vh, wh, [undef] )
```

引数

x	<R(:)>	in	デカルト系第一座標.
y	<R(:)>	in	デカルト系第二座標.
zast	<R(size(x),size(y),:)>	in	terrain 系の鉛直高度 [m].
u	<R(size(x),size(y),size(zast,3))>	in	zast に対応する東西風速 [m/s].
v	<R(size(x),size(y),size(zast,3))>	in	zast に対応する南北風速 [m/s].
w	<R(size(x),size(y),size(zast,3))>	in	zast に対応する鉛直風速 [m/s].
uh	<R(size(x),size(y),size(zast,3))>	inout	変換後の東西風速 [m/s].
vh	<R(size(x),size(y),size(zast,3))>	inout	変換後の南北風速 [m/s].
wh	<R(size(x),size(y),size(zast,3))>	inout	変換後の鉛直風速 [m/s].
undef	R	in	未定義値.

定義式

本ルーチンは水平方向にはデカルト座標系のみを想定している。今後拡張予定。
terrain 系で定義されている東西、南北、鉛直風速を (U,V,W) とし、terrain 系における高度座標を z^* とすると、デカルト成分に変換したときの東西、南北、鉛直風速成分 (u,v,w) は以下のように変換される：

$$u = \frac{U}{\sqrt{1+\left(\frac{\partial z^*}{\partial x}\right)_z^2}}, \quad v = \frac{V}{\sqrt{1+\left(\frac{\partial z^*}{\partial y}\right)_z^2}},$$

$$w = W + \frac{U\left(\frac{\partial z^*}{\partial x}\right)_z}{\sqrt{1+\left(\frac{\partial z^*}{\partial x}\right)_z^2}} + \frac{V\left(\frac{\partial z^*}{\partial y}\right)_z}{\sqrt{1+\left(\frac{\partial z^*}{\partial y}\right)_z^2}}.$$

ここで、添字 z はデカルト座標系の z で評価された水平勾配であることを示している。また、このとき実際のプログラムにおける引数は以下の対応をする：

$$(u,v,w) \ : \ \texttt{(uh,vh,wh)}, \quad (U,V,W) \ : \ \texttt{(u,v,w)},$$

$$z^*(x,y,z) \ : \ \texttt{zast(size(x),size(y),:)}, \quad (x,y) \ : \ \texttt{(x,y)}.$$

備考

- 定義式の導出は付録**??**参照.
- 未定義値が定義されている場合は、その値に対応する値が格子点に入っている場合、その格子を処理格子として使用している格子点での計算値がすべて未定義値として設定されて返される.

3.4 ffts

FFT (高速フーリエ変換) に関するルーチン集. FFT 計算の詳細については, **??** 参照.

3.4.1 c2r_ffttp_1d

機能

1 次元実数データの FFT 逆変換を行う。

書式

```
call c2r_ffttp_1d( nx, a, b, [prim], [prim_fact], [omega_fix], [omegan_fix]
)
```

引数

nx	<I>	in	データの個数.
a	<CP(nx/2)>	in	複素数変換前データ.
b	<R(nx)>	inout	実数変換後データ.
prim	C(1)	in	素因数分解判定（後述)。 'o' =分解する, 'x' =分解しない。
prim_fact	I(5)	in	素因数指数（後述)。
omega_fix	CP(nx/2,nx/2)	in	2,3,5,7 以外の素因数での 回転行列（後述)。
omegan_fix	CP(nx/2,nx/2)	in	全回転行列（後述)。

定義式

N 個のデータをもつ実数 $x(k),\ k = 0, \cdots, N-1$ を考える（N は偶数)。このデータの離散フーリエ正変換 $\hat{x}(l)$ は

$$\hat{x}(l) = \frac{1}{N} \sum_{k=0}^{N-1} x(k) e^{-2\pi i \frac{kl}{N}}, \quad l = 0, \cdots N-1$$

で定義される。逆変換は

$$x(k) = \sum_{l=0}^{N-1} \hat{x}(k) e^{2\pi i \frac{lk}{N}}, \quad k = 0, \cdots N-1$$

で定義される。これを Temperton のアルゴリズムを用いて FFT 処理する。実数データの FFT 計算の詳細については, **??** 参照。

備考

- 引数 prim はデータ数 N を素因数分解するものであり、このオプションが 'x' の場合はデータがそのままフーリエ変換処理に回されるので、単なる離散フーリエ変換を行うことになる。分解を行えば、データ数が $N = 2^a 3^b 5^c 7^d \times e$ まで分解を行い、FFT 処理を行う。引数 prim_fact はこのときの素因数指数の数を指定する. つまり,

  prim_fact(1:5)=(/a,b,c,d,e/)

  という対応となる.
- 引数 omegan_fix はデータ数 nx における FFT で使用する回転行列[*3]を指定する. 引数 omega_fix はデータ数が 2,3,5,7 以外の素因数 (1 つめの備考を例にすると, e がそれにあたる) に対応する回転行列. これらの行列は専用ルーチン rotate_calc で行うことができる.
- 本ルーチンの仕様から、波数成分 $N/2$ は配列 a(1) の複素成分に格納しなければならない。

3.4.2 fftp_1d

機能

1 次元複素数データから FFT を行う。

書式

```
call fftp_1d( nx, a, b, csign, [prim], [prim_fact], [omega_fix], [omegan_fix]
)
```

引数

[*3]回転行列の詳細な解説は付録**??**参照.

nx	<I>	in	データの個数.
a	<CP(nx)>	in	複素数変換前データ.
b	<CP(nx)>	inout	複素数変換後データ.
csign	C(1)	in	正逆変換判定。
			'r' =正変換, 'i' =逆変換。
prim	C(1)	in	素因数分解判定（後述）。
			'o' =分解する, 'x' =分解しない。
prim_fact	I(5)	in	素因数指数（後述）。
omega_fix	CP(nx,nx)	in	2,3,5,7 以外の素因数での
			回転行列（後述）。
omegan_fix	CP(nx,nx)	in	全回転行列（後述）。

定義式

N 個のデータをもつ複素数 $x(k),\ k = 0, \cdots, N-1$ を考える。このデータの離散フーリエ正変換 $\hat{x}(l)$ は

$$\hat{x}(l) = \frac{1}{N} \sum_{k=0}^{N-1} x(k) e^{-2\pi i \frac{kl}{N}}, \quad l = 0, \cdots N-1$$

で定義される。逆変換は

$$x(k) = \sum_{l=0}^{N-1} \hat{x}(k) e^{2\pi i \frac{lk}{N}}, \quad k = 0, \cdots N-1$$

で定義される。これを Temperton のアルゴリズムを用いて FFT 処理する。

備考

- 引数 prim はデータ数 N を素因数分解するものであり、このオプションが 'x' の場合はデータがそのままフーリエ変換処理に回されるので、単なる離散フーリエ変換を行うことになる。分解を行えば、データ数が $N = 2^a 3^b 5^c 7^d \times e$ まで分解を行い、FFT 処理を行う。引数 prim_fact はこのときの素因数指数の数を指定する. つまり,

  ```
  prim_fact(1:5)=(/a,b,c,d,e/)
  ```

  という対応となる.

- 引数 omegan_fix はデータ数 nx における FFT で使用する回転行列[*4]を指定する. 引数 omega_fix はデータ数が 2,3,5,7 以外の素因数 (1 つめの備考を例にすると, e がそれにあたる) に対応する回転行列. これらの行列は専用ルーチン rotate_calc で行うことができる.

[*4] 回転行列の詳細な解説は付録**??**参照.

3.4.3 fftp_2d

機能

2 次元複素数データから FFT を行う。

書式

```
call fftp_2d( nx, ny, a, b, csign, [prim], [prim_fact], [omega_fix],
[omegan_fix] )
```

引数

nx	<I>	in	1 次データの個数.
ny	<I>	in	2 次データの個数.
a	<CP(nx,ny)>	in	複素数変換前データ.
b	<CP(nx,ny)>	inout	複素数変換後データ.
csign	C(1)	in	正逆変換判定。
			'r' =正変換, 'i' =逆変換。
prim	C(1)	in	素因数分解判定（後述)。
			'o' =分解する, 'x' =分解しない。
prim_fact	I(5)	in	素因数指数（後述)。
omega_fix	CP(nx,nx)	in	2,3,5,7 以外の素因数での
			回転行列（後述)。
omegan_fix	CP(nx,nx)	in	全回転行列（後述)。

定義式

(N, M) 個のデータをもつ複素数 $x(k,l),\ k = 0, \cdots, N-1, l = 0, \cdots, M-1$ を考える。このデータの離散フーリエ正変換 $\hat{x}(m,n)$ は

$$\hat{x}(m,n) = \frac{1}{NM} \sum_{l=0}^{M-1} \left[\sum_{k=0}^{N-1} x(k,l) e^{-2\pi i \frac{km}{N}} \right] e^{-2\pi i \frac{ln}{M}}, \quad m = 0, \cdots, N-1, n = 0, \cdots, M-1$$

で定義される。逆変換は

$$\hat{x}(k,l) = \sum_{n=0}^{M-1} \left[\sum_{m=0}^{N-1} x(m,n) e^{2\pi i \frac{km}{N}} \right] e^{2\pi i \frac{ln}{M}}, \quad k = 0, \cdots, N-1, l = 0, \cdots, M-1$$

で定義される。これを Temperton のアルゴリズムを用いて FFT 処理する。

備考

- 引数 prim はデータ数 N を素因数分解するものであり、このオプションが 'x' の場合はデータがそのままフーリエ変換処理に回されるので、単なる離散フーリエ変換を行うことになる。分解を行えば、データ数が $N = 2^a 3^b 5^c 7^d \times e$ まで分解を行い、FFT 処理を行う。引数 prim_fact はこのときの素因数指数の数を指定する. つまり,

prim_fact(1:5)=(/a,b,c,d,e/)

という対応となる.

- 引数 omegan_fix はデータ数 nx における FFT で使用する回転行列[*5]を指定する. 引数 omega_fix はデータ数が 2,3,5,7 以外の素因数 (1 つめの備考を例にすると, e がそれにあたる) に対応する回転行列. これらの行列は専用ルーチン rotate_calc で行うことができる.

3.4.4 prim_calc

機能

任意の正数値について, 2,3,5,7 の素因数分解を行う.

書式

```
call prim_calc( n, factor, resid )
```

引数

n	<I>	in	分解する正整数値.
factor	<I(4)>	inout	2,3,5,7 の各素因数指数.
resid	<I>	inout	2,3,5,7 以外の素因数.

定義式

任意の正整数値 n が

$$n = 2^a 3^b 5^c 7^d \times e$$

であった場合, 上の各変数は以下のような対応となる.

$$n \to \mathtt{n},\ (a,b,c,d) \to \mathtt{factor}(1:4),\ e \to \mathtt{resid}$$

備考

特になし.

3.4.5 r2c_fftp_1d

機能

1 次元実数データから FFT を行う。

書式

```
call r2c_fftp_1d( nx, a, b, [prim], [prim_fact], [omega_fix], [omegan_fix]
)
```

[*5] 回転行列の詳細な解説は付録??参照.

引数

nx	<I>	in	データの個数.
a	<R(nx)>	in	実数変換前データ.
b	<CP(nx/2)>	inout	複素数変換後データ.
prim	C(1)	in	素因数分解判定（後述)。
			'o' =分解する, 'x' =分解しない。
prim_fact	I(5)	in	素因数指数（後述)。
omega_fix	CP(nx/2,nx/2)	in	2,3,5,7 以外の素因数での
			回転行列（後述)。
omegan_fix	CP(nx/2,nx/2)	in	全回転行列（後述)。

定義式

N 個のデータをもつ実数 $x(k),\ k = 0, \cdots, N-1$ を考える（N は偶数)。このデータの離散フーリエ正変換 $\hat{x}(l)$ は

$$\hat{x}(l) = \frac{1}{N} \sum_{k=0}^{N-1} x(k) e^{-2\pi i \frac{kl}{N}}, \quad l = 0, \cdots N-1$$

で定義される。逆変換は

$$x(k) = \sum_{l=0}^{N-1} \hat{x}(k) e^{2\pi i \frac{lk}{N}}, \quad k = 0, \cdots N-1$$

で定義される。これを Temperton のアルゴリズムを用いて FFT 処理する。実数データの FFT 計算の詳細については, **??** 参照。

備考

- 引数 prim はデータ数 N を素因数分解するものであり、このオプションが 'x' の場合はデータがそのままフーリエ変換処理に回されるので、単なる離散フーリエ変換を行うことになる。分解を行えば、データ数が $N = 2^a 3^b 5^c 7^d \times e$ まで分解を行い、FFT 処理を行う。引数 prim_fact はこのときの素因数指数の数を指定する. つまり,

  prim_fact(1:5)=(/a,b,c,d,e/)

  という対応となる.
- 引数 omegan_fix はデータ数 nx における FFT で使用する回転行列[*6]を指定する. 引数 omega_fix はデータ数が 2,3,5,7 以外の素因数 (1 つめの備考を例にすると, e がそれにあたる) に対応する回転行列. これらの行列は専用ルーチン rotate_calc で行うことができる.

[*6] 回転行列の詳細な解説は付録**??**参照.

- 実数データのフーリエ変換は変換後のデータ数が変換前のデータ数の半分に圧縮されていることに注意。また、本ルーチンの仕様から、複素波数成分 $N/2$ は b(1) の虚部に格納される。

3.4.6 rotate_calc

機能

データ数 nx の回転行列を計算する。

書式

```
call rotate_calc( nx, csign, prim_fact, omega, omegan )
```

引数

nx	<I>	in	データの個数.
csign	C(1)	in	正逆変換判定。 'r' =正変換, 'i' =逆変換。
prim_fact	I(5)	in	素因数指数（後述)。
omega	CP(prim_fact(5),prim_fact(5))	in	2,3,5,7 以外の素因数での回転行列。
omegan	CP(nx,nx)	in	全回転行列。

定義式

N 個のデータの離散フーリエ変換に用いる回転行列 $W_{j,k}$ は

$$W_{j,k} = e^{\pm 2\pi ijk/N}, \quad j = 0, 1, \cdots, N-1, \quad k = 0, 1, \cdots, N-1$$

で定義される。正変換は負号, 逆変換は正号で定義される. また, 2,3,5,7 以外の素因数 α での回転行列は上式において,

$$N \to \alpha$$

としただけである.

備考

特になし.

3.5 file_operate

IO ルーチン集. 現在対応しているファイルフォーマットは以下のとおりである.

- 4 バイトバイナリデータ.
- mgdsst データ.
- テキストカラムデータ.
- gtool3 フォーマットデータ.

3.5.1 auto_read_file

機能

2 次元, 3 次元変数混在の 4 バイトバイナリファイルからデータを読み出す `read_file` の自動化ルーチン (CReSS 仕様).

書式

```
call auto_read_file( file_name, d2n, d3n, d2val, d3val,
ad2val, ad3val, nx, ny, nz, d2var,
d3var, [specnz], [intvnz] )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
d2n	<I>	in	読み込む 2 次元変数の数.
d3n	<I>	in	読み込む 3 次元変数の数.
d2val	<C(*)>	in	読み込む 2 次元変数の順番 (後述).
d3val	<C(*)>	in	読み込む 3 次元変数の順番 (後述).
ad2val	<C(*)>	in	読み込む 2 次元変数の順番 (後述).
ad3val	<C(*)>	in	読み込む 3 次元変数の順番 (後述).
nx	<I>	in	x 方向の要素数.
ny	<I>	in	y 方向の要素数.
nz	<I>	in	z 方向の要素数.
d2var	<R(nx,ny,d2n)>	inout	2 次元データ.
d3var	<R(nx,ny,nz,d3n)>	inout	3 次元データ.
specnz	<I>	in	z 方向のある 1 層 (後述).
intvnz	<I(2)>	in	z 方向のある部分連続層 (後述).

定義式

`call read_file` 参照.

備考

- 4 バイトバイナリデータが 2, 3, 2, 3 次元という順番で 1 ファイルに格納されているようなファイルに用いられる.

- d2val, d3val, ad2val, ad3val はそれぞれ読み込む変数を示すフラグであり, 2, 3, 2, 3 の順番に 4, 3, 4, 5 個の変数が格納されている場合,

```
d2val = '0001', d3val = '000', ad2val = '0000', ad3val =
'10000'
```

  となり, この場合, 最初の 2 次元データの 4 番目, 最後の 3 次元データの 1 番目のみ読まれるので, d2n = 1, d3n = 1 で読み込めばよい.
- specnz が設定されている場合, 3 次元データは nz 個の中で, specnz 番目のデータのみが格納される.
- intvnz が設定されている場合, 3 次元データは nz 個の中で, intvnz(1) から intvnz(2) 番目までが格納される.

3.5.2 line_number_counter

機能

テキストカラムデータで格納されているファイルの行数を計算する.

書式

```
result = line_number_counter( file_name )
```

引数

file_name	<C(*)>	in	行数を計算するファイル名.
戻り値	<I>	inout	行数.

定義式

特になし.

備考

特になし.

3.5.3 read_file_grads

機能

grads のコントロールファイルから指定の変数を読み込むルーチン.

書式

```
call read_file_grads( fname, val_name, dim_len, intvz, intvt, val )
```

引数

fname	<C(*)>	in	読み出すコントロールファイル名.
val_name	<C(*)>	in	変数名.
dim_len	<I(4)>	in	各次元の要素数（後述）.
intvz	<I(2)>	in	高度方向の読み出す要素数（後述）.
intvt	<I(2)>	in	時間方向の読み出す要素区間（後述）.
val	<R(後述)>	inout	読み出した値が格納される変数.

定義式

特になし.

備考

- x, y, z, t の座標要素数がこの順番で dim_len(1:4) の順で格納される.
- intvz, intvt は読み出す高度, 時間範囲であり, 全て読み出す場合は intvz(1)=1, intvz(2)=nz, intvt(1)=1,intvt(2)=nt で設定しなければならない.
- 読み出す用の配列は

  R(dim_len(1),dim_len(2),intvz(1):intvz(2),intvt(1):intvt(2))

  で引数として与える.

3.5.4 request_axis_grads

機能

grads のコントロールファイルから各空間座標軸の座標値を取得するルーチン.

書式

```
call request_axis_grads( fname, axis_name, x, y )
```

引数

fname	<C(*)>	in	読み出すコントロールファイル名.
axis_name	<C(*)>	in	取得する軸の名前（後述）.
x	<R(:)>	inout	取得する軸の座標値（後述）.
y	<R(:)>	inout	pdef のときの軸の座標値（後述）.

定義式

特になし.

備考

- x, y, z のそれぞれの値を求める場合は xdef, ydef, zdef のそれぞれを axis_name に代入すればよい.

- x(:) として引数を与えるときの配列数は事前に request_dim_grads で要素数を取得しておかなければならない.
- GrADS のコントロールファイルにおけるパラメータの中に pdef というパラメータがあるため, そのパラメータを読み取るときのオプション. この場合, 東西方向の座標軸データが引数 x に入り, 南北座標軸データが引数 y に入る.

3.5.5 request_dim_grads

機能

grads のコントロールファイルから変数の空間次元の要素数を取得するルーチン.

書式

```
call request_dim_grads( fname, dim_len )
```

引数

fname	<C(*)>	in	読み出すコントロールファイル名.
dim_len	<I(4)>	inout	各次元の要素数（後述）.

定義式

特になし.

備考

- x, y, z, t の座標要素数がこの順番で dim_len(1:4) の順で格納される.

3.5.6 request_vardim_grads

機能

grads のコントロールファイルから任意の変数の高度方向の要素数を取得するルーチン.

書式

```
call request_vardim_grads( fname, var_name, nz )
```

引数

fname	<C(*)>	in	読み出すコントロールファイル名.
var_name	<C(*)>	in	高度要素を取得したい変数の名前.
nz	<I>	inout	取得する高度要素数.

定義式

特になし.

備考

特になし.

3.5.7 read_file

機能

ダイレクトアクセス形式のバイナリファイルから 2 次元データを読み出すルーチン.

書式

```
call read_file( file_name, nx, ny, rec_num, var )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
nx	<I>	in	第一要素配列数.
ny	<I>	in	第二要素配列数.
rec_num	<I>	in	nx×ny 分読み出すデータのレコード番号（後述）.
var	<R(nx,ny)>	inout	読み出すデータ.

定義式

```
1     open(unit=11, file=file_name, access='direct', recl=4*nx*ny, status='old')
2        read(11,rec=rec_num) ((var(i,j),i=1,nx),j=1,ny)
3     close(unit=11, status='keep')
```

備考

特になし.

3.5.8 read_file_3d

機能

ダイレクトアクセス形式のバイナリファイルから 3 次元データを読み出すルーチン.

書式

```
call read_file_3d( file_name, nx, ny, nz, rec_num, var )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
nx	<I>	in	第一要素配列数.
ny	<I>	in	第二要素配列数.
nz	<I>	in	第三要素配列数.
rec_num	<I>	in	読み出しを開始するデータのレコード番号（後述）.
var	<R(nx,ny,nz)>	inout	読み出すデータ.

定義式

```
1     open(unit=11, file=file_name, access='direct', recl=4*nx*ny, status='old')
2        read(11,rec=rec_num) ((var(i,j),i=1,nx),j=1,ny)
3     close(unit=11, status='keep')
```

備考

特になし.

3.5.9 read_file_gtool3

機能

gtool3 形式のデータを読み込むルーチン.

書式

```
call read_file_gtool3( file_name, nx, ny, nz, tstr, tstep, tnum, var )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
nx	<I>	in	第一要素配列数.
ny	<I>	in	第二要素配列数.
nz	<I>	in	第三要素配列数.
tstr	<I>	in	読み始める時刻 (後述).
tstep	<I>	in	読み飛ばす時間間隔 (後述).
tnum	<I>	in	読み込む時間数 (後述).
var	<R(nx,ny,nz,tnum)>	inout	読み出すデータ.

定義式

特になし.

備考

- tstr, tstep, tnum の関係は, 1 つのデータファイルに tmax 個分の時刻データが格納されている場合,

```
tstr + tstep * (tnum - 1) <= tmax
```

という関係が満たされればよい. ここで, tstr = 1 なら, 初期時刻が読み込まれることになる. 実際の計算時刻や出力時間ステップは関係ない.

3.5.10 read_file_gtool3_header_c

機能

gtool3 形式のヘッダーデータのうち, 文字形式のデータを読み出すルーチン.

書式

```
call read_file_gtool3_header_c( file_name, header, var )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
header	<C(*)(:)>	in	ヘッダーの名前.
var	<C(16)(size(header))>	inout	header に対応するデータ.

定義式

特になし.

備考

- ヘッダーの名称は,
  http://www.riam.kyushu-u.ac.jp/taikai/lab/others/Gtool/node111.html
  と同じ名称で与える. ただし, Fortran の予約語である unit, size については, uni, siz と入力すること.

3.5.11 read_file_gtool3_header_i

機能

gtool3 形式のヘッダーデータのうち, 整数形式のデータを読み出すルーチン.

書式

```
call read_file_gtool3_header_i( file_name, header, var )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
header	<C(*)(:)>	in	ヘッダーの名前.
var	<I(size(header))>	inout	header に対応するデータ.

定義式

特になし.

備考

- ヘッダーの名称は,
  http://www.riam.kyushu-u.ac.jp/taikai/lab/others/Gtool/node111.html
  と同じ名称で与える. ただし, Fortran の予約語である unit, size については, uni, siz と入力すること.

3.5.12 read_file_gtool3_header_r

機能

gtool3 形式のヘッダーデータのうち, 実数形式のデータを読み出すルーチン.

書式

```
call read_file_gtool3_header_r( file_name, header, var )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
header	<C(*)(:)>	in	ヘッダーの名前.
var	<R(size(header))>	inout	header に対応するデータ.

定義式

特になし.

備考

- ヘッダーの名称は,
  http://www.riam.kyushu-u.ac.jp/taikai/lab/others/Gtool/node111.html
  と同じ名称で与える. ただし, Fortran の予約語である unit, size については, uni, siz と入力すること.

3.5.13 read_file_text

機能

テキストカラム形式のデータを読み込むルーチン.

書式

```
call read_file_text( file_name, nx, ny, var, [skip], [forma] )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
nx	<I>	in	列数.
ny	<I>	in	行数.
var	<C(nx,ny)>	inout	読み出すデータ.
skip	<I>	in	先頭読み飛ばす行数.
forma	<C(*)>	in	読み込むフォーマット (後述).

定義式

コンマ, スペース区切りのテキストデータを読み込む. nx は 1 行に格納されているデータの列数, ny は先頭から読み取る行数. ただし, skip が設定されている場合は, skip 行だけ読み飛ばして読み始める.

備考

- デフォルトでは, 読み飛ばし変数 skip はゼロに設定されている.

3.5.14 read_mgdsst

機能

気象庁 MGDSST データを読み込むためのルーチン. 任意の領域ではなく, 全球のみ対応.

書式

```
call read_file( fname, var )
```

引数

fname	<C(*)>	in	読み出すファイル名.
var	<R(1440,720)>	inout	sst データ.

定義式

特になし. (MGDSST データのフォーマットについては,
`http://near-goos1.jodc.go.jp/rdmdb/format/JMA/mgdsst.txt`
参照.)

備考

- 要素番号の小さいものが地球の南西端 (経度 0 °, 緯度 −89.875 °) に入り, 配列第一要素は東向き, 第二要素は北向きに格子データが格納されて出力される.
- 未定義値は 999.0, 海氷データは 888.0 という値で格納される.

3.5.15 write_file

機能

ダイレクトアクセス形式のバイナリファイルに 2 次元データを書き出すルーチン.

書式

```
call write_file( file_name, nx, ny, rec_num, var, [mode] )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
nx	<I>	in	第一要素配列数.
ny	<I>	in	第二要素配列数.
rec_num	<I>	in	nx×ny 分読み出すデータのレコード番号.
var	<R(nx,ny)>	inout	読み出すデータ.
mode	<C(*)>	in	ファイルの書き出しオプション.

定義式

```
1     open(unit=11, file=file_name, access='direct', recl=4*nx*ny, status=mode)
2        write(11,rec=rec_num) ((var(i,j),i=1,nx),j=1,ny)
3     close(unit=11, status='keep')
```

備考

- mode は Fortran の書き出し status= に設定される値を代入.

3.5.16 write_file_3d

機能

ダイレクトアクセス形式のバイナリファイルに 3 次元データを書き出すルーチン.

書式

```
call write_file_3d( file_name, nx, ny, nz, rec_num, var, [mode] )
```

引数

file_name	<C(*)>	in	読み出すファイル名.
nx	<I>	in	第一要素配列数.
ny	<I>	in	第二要素配列数.
nz	<I>	in	第三要素配列数.
rec_num	<I>	in	読み出しを開始するデータのレコード番号.
var	<R(nx,ny,nz)>	inout	読み出すデータ.
mode	<C(*)>	in	ファイルの書き出しオプション.

定義式

```
1     open(unit=11, file=file_name, access='direct', recl=4*nx*ny, status=mode)
2        write(11,rec=rec_num) ((var(i,j),i=1,nx),j=1,ny)
3     close(unit=11, status='keep')
```

備考

- mode は Fortran の書き出し status= に設定される値を代入.

3.6 math_const

数学的普遍定数集. 現在は以下の表のように与えられている.

変数名	型	設定値	変数の意味
pi	<R>	3.14159265	円周率
img	<CP>	(0.0,1.0)	虚数単位

以下は parameter 属性のついていない準定数である.

変数名	型	属性	変数の意味
romega2	<CP(0:1,0:1)>	save	データ 2 の FFT 用回転行列
			(正変換用)
romega3	<CP(0:2,0:2)>	save	データ 3 の FFT 用回転行列
			(正変換用)
romega5	<CP(0:4,0:4)>	save	データ 5 の FFT 用回転行列
			(正変換用)
romega7	<CP(0:6,0:6)>	save	データ 7 の FFT 用回転行列
			(正変換用)
iomega2	<CP(0:1,0:1)>	save	データ 2 の FFT 用回転行列
			(逆変換用)
iomega3	<CP(0:2,0:2)>	save	データ 3 の FFT 用回転行列
			(逆変換用)
iomega5	<CP(0:4,0:4)>	save	データ 5 の FFT 用回転行列
			(逆変換用)
iomega7	<CP(0:6,0:6)>	save	データ 7 の FFT 用回転行列
			(逆変換用)

3.6.1 rotate_array

機能

データ数 2,3,5,7 の回転行列を計算する。

書式

```
call rotate_array( )
```

引数

なし. math_const において設定されている準定数 omega2, omega3, omega5, omega7 が更新される.

定義式

3.4.6 の定義式において, $N = 2, 3, 5, 7$ で計算している.

備考

特になし.

3.7 matrix_calc

単純な行列演算, 連立代数方程式の求解, 固有値・固有ベクトル計算用のルーチン集. 各ルーチンの詳細な計算方法については, **??** 参照.

3.7.1 Gau_Sei

機能

ガウスザイデル法による連立 1 次方程式の求解を行うルーチン.

書式

```
call Gau_Sei( a, b, eps, x )
```

引数

b	<R(:)>	inout	ベクトルの成分 (後述).
a	<R(size(b),size(b))>	inout	行列 (後述).
eps	<R>	in	収束条件値 (後述).
x	<R(size(b))>	inout	求めたいベクトル (後述).

定義式

行列 a_{ij}, ベクトル b_i としたとき,

$$a_{ij}x_j = b_i$$

という関係を満たす未知ベクトル x_j を求める. このとき, 引数との対応関係は

$$a_{ij} \ : \ \mathtt{a(i,j)}, \quad x_j \ : \ \mathtt{x(j)}, \quad b_i \ : \ \mathtt{b(i)}$$

である. ガウスザイデル法による計算方法の詳細は**??**参照.

備考

- 引数 eps は各配列の反復前後の相対誤差を表し, この値の最大値が eps 以下になったときに反復計算を終了する.

3.7.2 Jacobi_algebra

機能

ヤコビ法による連立 1 次方程式の求解を行うルーチン.

書式

```
call Jacobi_algebra( a, b, eps, x )
```

引数

b	<R(:)>	inout	ベクトルの成分 (後述).
a	<R(size(b),size(b))>	inout	行列 (後述).
eps	<R>	in	収束条件値 (後述).
x	<R(size(b))>	inout	求めたいベクトル (後述).

定義式

行列 a_{ij}, ベクトル b_i としたとき,

$$a_{ij}x_j = b_i$$

という関係を満たす未知ベクトル x_j を求める. このとき, 引数との対応関係は

$$a_{ij} \ : \ \mathtt{a(i,j)}, \quad x_j \ : \ \mathtt{x(j)}, \quad b_i \ : \ \mathtt{b(i)}$$

である. ヤコビ法による計算方法の詳細は**??**参照.

備考

- 引数 eps は各配列の反復前後の相対誤差を表し, この値の最大値が eps 以下になったときに反復計算を終了する.

3.7.3 Jacobi_eigen

機能

ヤコビ法を用いて任意の実対称行列の全固有値を求めるルーチン.

書式

```
call Jacobi_eigen( a, lambda, [eps] )
```

引数

a	<R(:,:)>	in	固有値を求める行列 (後述).
lambda	<R(size(a,1))>	inout	固有値 (後述).
eps	<R>	in	反復の収束判条件 (後述).

定義式

行列 a_{ij}, ベクトル x_i としたとき,

$$a_{ij}x_j = \lambda x_i$$

という関係を満たす未知数 λ を求める. ここで, λ の値は行列の次元数個ある. このとき, 引数との対応関係は

$$a_{ij} \ : \ \mathtt{a(i,j)}, \quad \lambda \ : \ \mathtt{lambda(k)}$$

である. k 番目の固有値をここでは lambda(k) とした. ヤコビ法による計算方法の詳細は**??**を参照.

備考

- eps は反復計算の収束判定条件であり, 1 回の反復計算前後での絶対誤差の最大値が eps 以下になったときに計算を終了する.

3.7.4 LU_devs

機能

部分ピボッティングつき LU 分解を行うルーチン.

書式

```
call LU_devs( a, b, x, itermax )
```

引数

b	<R(:)>	inout	ベクトルの成分 (後述).
a	<R(size(b),size(b))>	inout	行列 (後述).
x	<R(size(b))>	inout	求めたいベクトル (後述).
itermax	<I>	in	反復改良の回数.

定義式

行列 a_{ij}, ベクトル b_i としたとき,

$$a_{ij}x_j = b_i$$

という関係を満たす未知ベクトル x_j を求める. このとき, 引数との対応関係は

$$a_{ij} \ : \ \mathtt{a(i,j)}, \quad x_j \ : \ \mathtt{x(j)}, \quad b_i \ : \ \mathtt{b(i)}$$

である. 詳細な計算は**??**参照.

備考

特になし.

3.7.5 SOR_Gau_Sei

機能

SOR を用いてガウスザイデル法による連立 1 次方程式の求解を行うルーチン.

書式

```
call SOR_Gau_Sei( a, b, eps, accel, x )
```

引数

b	<R(:)>	inout	ベクトルの成分 (後述).
a	<R(size(b),size(b))>	inout	行列 (後述).
eps	<R>	in	収束条件値 (後述).
accel	<R>	in	SOR の加速係数 (後述).
x	<R(size(b))>	inout	求めたいベクトル (後述).

定義式

行列 a_{ij}, ベクトル b_i としたとき,

$$a_{ij}x_j = b_i$$

という関係を満たす未知ベクトル x_j を求める. このとき, 引数との対応関係は

$$a_{ij} \ : \ \mathtt{a(i,j)}, \quad x_j \ : \ \mathtt{x(j)}, \quad b_i \ : \ \mathtt{b(i)}$$

である. SOR つきガウスザイデル法による計算方法の詳細は**??**参照.

備考

- 引数 eps は各配列の反復前後の相対誤差を表し, この値の最大値が eps 以下になったときに反復計算を終了する.
- 数学上, 加速係数は 2 未満にしなければ安定計算ができない. 本ルーチンでは, 2 以上の値が設定された場合, エラーとなるようにしている.

3.7.6 SOR_Jacobi_algebra

機能

SOR を用いてヤコビ法による連立 1 次方程式の求解を行うルーチン.

書式

```
call SOR_Jacobi_algebra( a, b, eps, accel, x )
```

引数

b	<R(:)>	inout	ベクトルの成分 (後述).
a	<R(size(b),size(b))>	inout	行列 (後述).
eps	<R>	in	収束条件値 (後述).
accel	<R>	in	SOR の加速係数 (後述).
x	<R(size(b))>	inout	求めたいベクトル (後述).

定義式

行列 a_{ij}, ベクトル b_i としたとき,

$$a_{ij}x_j = b_i$$

という関係を満たす未知ベクトル x_j を求める. このとき, 引数との対応関係は

$$a_{ij} \ : \ \mathtt{a(i,j)}, \quad x_j \ : \ \mathtt{x(j)}, \quad b_i \ : \ \mathtt{b(i)}$$

である. SOR つきヤコビ法による計算方法の詳細は**??**参照.

備考

- 引数 eps は各配列の反復前後の相対誤差を表し, この値の最大値が eps 以下になったときに反復計算を終了する.
- 数学上, 加速係数は 2 未満にしなければ安定計算ができない. 本ルーチンでは, 2 以上の値が設定された場合, エラーとなるようにしている.

3.7.7 determ_2d

機能

2 次行列の行列式を返す.

書式

```
result = determ_2d( a )
```

引数

a　<I,R(2,2)>　in　計算する行列.

定義式

2 次元行列 A の行列式 $|A|$ は

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad \Rightarrow \quad |A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

である.

備考

特になし.

3.7.8 eigenvalue_power

機能

べき乗法を用いて行列の最大固有値とその固有値に対応する固有ベクトルを求めるルーチン.

書式

```
call eigenvalue_power( a, eps, lambda, lv )
```

引数

lv	<R(:)>	inout	固有ベクトル (後述).
a	<R(size(lv),size(lv))>	inout	固有値を求める行列 (後述).
eps	<R>	in	反復の収束判条件 (後述).
lambda	<R>	inout	固有値 (後述).

定義式

行列 a_{ij}, ベクトル x_i としたとき,

$$a_{ij}x_j = \lambda x_i$$

という関係を満たす未知数 λ を求める. ここで, λ の値は (縮退している場合を除いて,) 行列の次元数個あるので, そのうちの絶対値が最大となる固有値に対応する. このとき, 引数との対応関係は

$$a_{ij} \ : \ \mathtt{a(i,j)}, \quad \lambda \ : \ \mathtt{lambda}, \quad x_i \ : \ \mathtt{lv(i)}$$

である. べき乗法による計算方法の詳細は**??**を参照.

備考

特になし.

3.7.9 gausss

機能

部分ピボッティングつきガウスの消去法を行うルーチン.

書式

```
call gausss( a, b, x )
```

引数

b	<R(:)>	inout	ベクトルの成分 (後述).
a	<R(size(b),size(b))>	inout	行列 (後述).
x	<R(size(b))>	inout	求めたいベクトル (後述).

定義式

行列 a_{ij}, ベクトル b_i としたとき,

$$a_{ij}x_j = b_i$$

という関係を満たす未知ベクトル x_j を求める. このとき, 引数との対応関係は

$$a_{ij} \ : \ \mathtt{a(i,j)}, \quad x_j \ : \ \mathtt{x(j)}, \quad b_i \ : \ \mathtt{b(i)}$$

である. ガウスの消去法の詳細な計算方法は**??** 参照.

備考

特になし.

3.7.10 invert_mat

機能

任意正方行列の逆行列を返す.

書式

```
call invert_mat( a )
```

引数

a	<I,R(:,:)>	in	求める行列.
b	<I,R(size(a,1),size(a,1))>	inout	a の逆行列.

定義式

ガウスの消去法を用いて逆行列を計算する. 詳細な計算方法は**??**参照.

備考

特になし.

3.7.11 schumit_norm

機能

任意のベクトルの組をシュミットの直交化法を用いて正規直交ベクトル系に変換するルーチン.

書式

```
call schumit_norm( u, v )
```

引数

u	<R(:,:)>	in	計算するベクトル系.
v	<R(size(u,1),size(u,2))>	inout	正規直交化ベクトル.

定義式

任意のベクトル系 $\{\boldsymbol{u}\}_j$ を互いに線形独立な正規直交ベクトル系 $\{\boldsymbol{v}\}_j$ に変換するためのシュミットの直交化法は

$$\{\boldsymbol{v}\}_j = \frac{\{\boldsymbol{v}'\}_j}{\left|\{\boldsymbol{v}'\}_j\right|}, \quad \{\boldsymbol{v}'\}_j = \{\boldsymbol{u}\}_j - \sum_{k=0}^{j-1} (\{\boldsymbol{u}\}_j \cdot \{\boldsymbol{v}\}_k)\{\boldsymbol{v}\}_k$$

である.

備考

特になし.

3.7.12 trans_mat

機能

任意正方行列の転置を返す.

書式

```
call trans_mat( a )
```

引数

a	<I,R(:,:)>	inout	転置にしたい.

定義式

行列 A_{ij} の転置 ${}^TA$ は A_{ji} である.

備考

- 引数は inout 属性がついているので, 引数の結果は転置された値で返される.

3.8 max_min

配列の最大値, 最小値を検索するルーチン集.

3.8.1 max_val_1d

機能

1 次元配列の最大値とその値が格納されている配列要素番号を取得する。

書式

```
call max_val_1d( var, mamn, mamv, [undef] )
```

引数

var	<R(:)>	in	1 次元の探索配列.
mamn	<I>	inout	最大値が格納されている要素番号.
mamv	<R>	inout	var の最大値 .
undef	<R>	in	未定義値（後述）.

定義式

単純に 1 番目から配列の値を比較する。最大値が複数あった場合、最も要素番号の若いものが返される。

備考

- 未定義値が定義されている場合は、その配列を比較しない。

3.8.2 max_val_2d

機能

2 次元配列の最大値とその値が格納されている配列要素番号を取得する。

書式

```
call max_val_2d( var, mamn, mamv, [undef] )
```

引数

var	<R(:,:)>	in	2 次元の探索配列.
mamnx	<I>	inout	配列内の最大値に該当する第 1 配列要素番号.
mamny	<I>	inout	配列内の最大値に該当する第 2 配列要素番号.
mamv	<R>	inout	var の最大値 .
undef	<R>	in	未定義値（後述）.

定義式

単純に 1 番目から配列の値を比較する。最大値が複数あった場合、最も要素番号の若いものが返される。

備考

- 未定義値が定義されている場合は、その配列を比較しない。

3.8.3 max_val_3d

機能

3 次元配列の最大値とその値が格納されている配列要素番号を取得する。

書式

```
call max_val_3d( var, mamn, mamv, [undef] )
```

引数

var	<R(:)>	in	3 次元の探索配列.
mamnx	<I>	inout	配列内の最大値に該当する第 1 配列要素番号.
mamny	<I>	inout	配列内の最大値に該当する第 2 配列要素番号.
mamnz	<I>	inout	配列内の最大値に該当する第 3 配列要素番号.
mamv	<R>	inout	var の最大値 .
undef	<R>	in	未定義値（後述）.

定義式

単純に 1 番目から配列の値を比較する。最大値が複数あった場合、最も要素番号の若いものが返される。

備考

- 未定義値が定義されている場合は、その配列を比較しない。

3.8.4 min_val_1d

機能

1 次元配列の最小値とその値が格納されている配列要素番号を取得する。

書式

```
call min_val_1d( var, mamn, mamv, [undef] )
```

引数

var	<R(:)>	in	1 次元の探索配列.
mamn	<I>	inout	最小値が格納されている要素番号.
mamv	<R>	inout	var の最小値 .
undef	<R>	in	未定義値（後述）.

定義式

単純に 1 番目から配列の値を比較する。最小値が複数あった場合、最も要素番号の若いものが返される。

備考

- 未定義値が定義されている場合は、その配列を比較しない。

3.8.5 min_val_2d

機能

2 次元配列の最小値とその値が格納されている配列要素番号を取得する。

書式

```
call min_val_2d( var, mamn, mamv, [undef] )
```

引数

var	<R(:,:)>	in	2 次元の探索配列.
mamnx	<I>	inout	配列内の最小値に該当する第 1 配列要素番号.
mamny	<I>	inout	配列内の最小値に該当する第 2 配列要素番号.
mamv	<R>	inout	var の最小値 .
undef	<R>	in	未定義値（後述）.

定義式

単純に 1 番目から配列の値を比較する。最小値が複数あった場合、最も要素番号の若いものが返される。

備考

- 未定義値が定義されている場合は、その配列を比較しない。

3.8.6 min_val_3d

機能

3 次元配列の最小値とその値が格納されている配列要素番号を取得する。

書式

```
call min_val_3d( var, mamn, mamv, [undef] )
```

引数

var	<R(:)>	in	3 次元の探索配列.
mamnx	<I>	inout	配列内の最小値に該当する第 1 配列要素番号.
mamny	<I>	inout	配列内の最小値に該当する第 2 配列要素番号.
mamnz	<I>	inout	配列内の最小値に該当する第 3 配列要素番号.
mamv	<R>	inout	var の最小値 .
undef	<R>	in	未定義値（後述）.

定義式

単純に 1 番目から配列の値を比較する。最小値が複数あった場合、最も要素番号の若いものが返される。

備考

- 未定義値が定義されている場合は、その配列を比較しない。

3.9 phys_const

物理学的定数集. 物理学における普遍定数を以下の表のように与える. 出典は理科年表 (2008) より.

変数名	型	設定値	変数の意味	単位
Me	<R>	5.9736e24	地球の質量	[kg]
g	<R>	9.81	地球の標準重力加速度	$[m^2/s]$
omega	<R>	7.29e-5	地球の自転角速度	[1/s]
radius	<R>	6.357e6	地球の極半径	[m]

3.10 poly_function

直交多項式を計算するルーチン集. ここで定義される直交多項式ルーチンは, 引数の値に応じて interface 宣言されているので, 単精度の引数を与えれば単精度で, 倍精度の引数を与えれば倍精度で計算した結果を返す. また, ここで用いられている多項式の計算定義式の証明は**??**参照.

3.10.1 chebyshev

機能

$0-n$ 次のチェビシェフ多項式を計算する.

書式

```
call chebyshev( n, x, che )
```

引数

n	<I>	in	計算する最高次数.
x	<R,DP(:)>	in	求めたい引数.
che	<R,DP(0:n,size(x))>	inout	チェビシェフ多項式.

定義式

n 次チェビシェフ多項式を $T_n(x)$ とすると,

$$T_n(x) = \sum_{k=0}^{[n/2]} \frac{(-1)^k n\,!}{(2k)\,!\,(n-2k)\,!} x^{n-2}(1-x^2)^k.$$

備考

- 実際の評価は漸化式で評価されている. 評価方法の詳細については**??**参照.

3.10.2 gegenbauer

機能

$0-n$ 次のゲーゲンバウアー多項式を計算する.

書式

```
call gegenbauer( n, x, p, lambda )
```

引数

n	<I>	in	計算する最高次数.
x	<R,DP(:)>	in	求めたい引数.
p	<R,DP(0:n,size(x))>	inout	ゲーゲンバウアー多項式.
lambda	<R>	in	ゲーゲンバウアー係数.

定義式

n 次ゲーゲンバウアー多項式を $C_n^{\lambda}(x)$ とすると,

$$C_n^{\lambda}(x) = \frac{2^n \Gamma(n+\lambda)}{n\,!\,\Gamma(\lambda)} x^n F\left(-\frac{n}{2},\ \frac{1-n}{2},\ 1-n-\lambda;\ \frac{1}{x^2}\right).$$

ここで, F はガウスの超幾何級数である.

備考

- 実際の評価は漸化式で評価されている. 評価方法の詳細については**??**参照.

3.10.3 hermite

機能

$0-n$ 次のエルミート多項式を計算する.

書式

```
call hermite( n, x, p )
```

引数

n	<I>	in	計算する最高次数.
x	<R,DP(:)>	in	求めたい引数.
p	<R,DP(0:n,size(x))>	inout	エルミート多項式.

定義式

n 次エルミート多項式を $H_n(x)$ とすると,

$$H_n(x) = \sum_{k=0}^{[n/2]} \frac{(-1)^k (2k-1)\,!!\,n\,!}{(2k)\,!\,(n-2k)\,!} x^{n-2k}.$$

備考

- 実際の評価は漸化式で評価されている. 評価方法の詳細については**??**参照.

3.10.4 jacobi_poly

機能

$0-n$ 次のヤコビ多項式を計算する.

書式

```
call jacobi_poly( n, x, p, alpha, beta )
```

引数

n	<I>	in	計算する最高次数.
x	<R,DP(:)>	in	求めたい引数.
p	<R,DP(0:n,size(x))>	inout	ヤコビ多項式.
alpha	<R>	in	ヤコビ係数 1.
beta	<R>	in	ヤコビ係数 2.

定義式

n 次ヤコビ多項式を $G_n(\alpha,\ \beta;\ x)$ とすると,

$$G_n(\alpha,\ \beta;\ x) = 1 + \sum_{k=1}^{n} \frac{(-1)^k n\,!}{r\,!\,(n-r)\,!} \frac{\Gamma(\alpha+n+k)\Gamma(\beta)}{\Gamma(\alpha+n)\Gamma(\beta+k)} x^k.$$

備考

- 実際の評価は漸化式で評価されている. 評価方法の詳細については**??**参照.

3.10.5 laguerre

機能

$0-n$ 次のラゲール多項式を計算する.

書式

```
call laguerre( n, x, p )
```

引数

n	<I>	in	計算する最高次数.
x	<R,DP(:)>	in	求めたい引数.
p	<R,DP(0:n,size(x))>	inout	ラゲール多項式.

定義式

n 次ラゲール多項式を $L_n(x)$ とすると,

$$L_n(x) = \sum_{k=0}^{n} \frac{(-1)^k n\,!}{k\,!\,(n-k)\,!\,k\,!} x^k.$$

備考

- 実際の評価は漸化式で評価されている. 評価方法の詳細については**??**参照.

3.10.6 legendre

機能

$0-n$ 次のルジャンドル多項式を計算する.

書式

```
call legendre( n, x, p )
```

引数

n	<I>	in	計算する最高次数.
x	<R,DP(:)>	in	求めたい引数.
p	<R,DP(0:n,size(x))>	inout	ルジャンドル多項式.

定義式

n 次ルジャンドル多項式を $P_n(x)$ とすると,

$$P_n(x) = \sum_{k=0}^{\infty} i(-1)^k \frac{(n-k+1)(n-k+2)\cdots n(n+1)\cdots(n+k)}{k\,!\,k\,!} \left(\frac{1-x}{2}\right)^k.$$

備考

- 実際の評価は漸化式で評価されている. 評価方法の詳細については**??**参照.

3.10.7 sonine

機能

$0-n$ 次のソニン多項式を計算する.

書式

```
call sonine( n, x, p, lambda )
```

引数

n	<I>	in	計算する最高次数.
x	<R,DP(:)>	in	求めたい引数.
p	<R,DP(0:n,size(x))>	inout	ソニン多項式.
lambda	<R>	in	ソニン係数.

定義式

n 次ソニン多項式を $S_n^{\lambda}(x)$ とすると,

$$S_n^{\lambda}(x) = \sum_{k=0}^{n} \frac{(-1)^k(n+\lambda)\,!}{k\,!\,(n+\lambda-k)\,!\,k\,!} x^k.$$

備考

- 実際の評価は漸化式で評価されている. 評価方法の詳細については**??**参照.

3.11 special_function

特殊関数を計算する関数集. なお, 以下の関数マニュアルには, 関数の定義式のみ表記しており, 実際の評価手順は**??**を参考にされたい.

3.11.1 Full_Ellip1_Func

機能

第一種完全楕円関数を計算する.

書式

```
result = Full_Ellip1_Func( k )
```

引数

k	<R>	in	波数.
戻り値	<R,DP>	inout	計算結果.

定義式

$$K(k) = \int_0^{\pi/2} \frac{1}{\sqrt{1 - k^2 \sin^2 \theta}} d\theta.$$

備考

特になし.

3.11.2 Full_Ellip2_Func

機能

第一種完全楕円関数を計算する.

書式

```
result = Full_Ellip2_Func( k )
```

引数

k	<R>	in	波数.
戻り値	<R,DP>	inout	計算結果.

定義式

$$K(k) = \int_0^{\pi/2} \frac{1}{\sqrt{1 - k^2 \sin^2 \theta}} d\theta.$$

備考

特になし.

3.11.3 bessj

機能

第一種ベッセル関数を計算する.

書式

```
result = bessj( m, t )
```

引数

m	<I,R>	in	ベッセル関数の次数.
t	<R,DP>	in	変数.
戻り値	<R,DP>	inout	計算結果.

定義式

m 次ベッセル関数を $J_m(t)$ とすると,

$$J_m(t) = \sum_{k=0}^{\infty} \frac{(-1)^k}{k\,!\,\Gamma(k+m+1)} \left(\frac{t}{2}\right)^{2k+m}.$$

ここで, $\Gamma(m)$ は m 次ガンマ関数である.

備考

- 実際の評価式は積分形で評価されている. 評価方法の詳細については**??**参照.

3.11.4 bessy

機能

第二種ベッセル関数を計算する.

書式

```
result = bessy( m, t )
```

引数

m	<I,R>	in	ベッセル関数の次数.
t	<R,DP>	in	変数.
戻り値	<R,DP>	inout	計算結果.

定義式

m 次ベッセル関数を $Y_m(t)$ とすると,

$$Y_m(t) = J_m(t)\ln t - \frac{1}{2}\sum_{k=0}^{m-1}\frac{(m-k-1)\,!}{k\,!}\left(\frac{t}{2}\right)^{2k-m} - \frac{1}{2}\sum_{k=1}^{\infty}\frac{(-1)^k}{k\,!\,(n+k)\,!}\left[\left(1+\frac{1}{2}+\cdots+\frac{1}{k}\right)+\left(\frac{1}{n+1}+\cdots+\frac{1}{n+k}\right)\right]\left(\frac{t}{2}\right)^{2k+m}.$$

ここで, $\Gamma(m)$ は m 次ガンマ関数, $J_m(t)$ は m 次ベッセル関数である.

備考
- 実際の評価式は積分形で評価されている. 評価方法の詳細については**??**参照.

3.11.5 beszero

機能

第一種ベッセル関数のゼロ点を計算する.

書式

```
call beszero( nmax, mmax, k )
```

引数

nmax	<I>	in	ベッセル関数の最大次数.
mmax	<I>	in	求めるゼロ点の個数.
k	<R,DP(0:nmax,mmax)>	inout	ゼロ点での値 (備考参照).

定義式

第一種ベッセル関数の 0 次から n 次までの原点から m 個のゼロ点を計算する. このとき, 引数との対応関係は以下のようになる.

$$n \ : \ \texttt{nmax}, \quad m \ : \ \texttt{mmax}.$$

備考
- 例えば, n 次の m 番目のゼロ点での変数の値を求めたい場合, `call beszero( n, m, k(0:n,m) )` と指定すれば, `k(n,m)` にそのゼロ点の値が格納されている.

3.11.6 beta_func

機能

ベータ関数を計算する.

書式

```
result = beta_func( x, y )
```

引数

x	<R,DP>	in	第一要素.
y	<R,DP>	in	第二要素.
戻り値	<R,DP>	inout	計算結果.

定義式

ベータ関数を $B(x,y)$ とすると,

$$B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt, \quad x,y > 0$$

備考

- 実際の評価式は近似式で評価されている. 評価方法の詳細については**??**参照.
- ベータ関数には変数対称性をもつため,

$$B(x,y) = B(y,x)$$

であるので, 引数の順序は問わない.

3.11.7 delta

機能

クロネッカーのデルタを計算する.

書式

```
result = delta( i, j )
```

引数

i	<I>	in	第一要素.
j	<I>	in	第二要素.
戻り値	<R>	inout	計算結果.

定義式

$$\delta_{i,j} = \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases}$$

備考

特になし.

3.11.8 df_bessj

機能

第一種変形ベッセル関数を計算する.

書式

```
result = df_bessj( m, t )
```

引数

m	<I,R>	in	変形ベッセル関数の次数.
t	<R,DP>	in	変数.
戻り値	<R,DP>	inout	計算結果.

定義式

m 次変形ベッセル関数を $I_m(t)$ とすると,

$$I_m(t) = i^{-m} J_m(it).$$

ここで, $J_m(t)$ は m 次ベッセル関数, i は虚数単位である.

備考

- 実際の評価式は積分形で評価されている. 評価方法の詳細については**??**参照.

3.11.9 df_bessy

機能

第一種変形ノイマン関数を計算する.

書式

```
result = df_bessy( m, t )
```

引数

m	<I,R>	in	変形ノイマン関数の次数.
t	<R,DP>	in	変数.
戻り値	<R,DP>	inout	計算結果.

定義式

m 次変形ノイマン関数を $K_m(t)$ とすると,

$$K_m(t) = \frac{\pi}{2}\frac{I_{-m}(t) - I_m(t)}{\sin m\pi}.$$

ここで, $I_m(t)$ は m 次変形ベッセル関数である.

備考

- 実際の評価式は積分形で評価されている. 評価方法の詳細については**??**参照.

3.11.10 epsilon

機能

エディントンのイプシロンを計算する.

書式

```
result = epsilon( i, j, k )
```

引数

i	<I>	in	第一要素.
j	<I>	in	第二要素.
k	<I>	in	第三要素.
戻り値	<R>	inout	計算結果.

定義式

$$\varepsilon_{i,j,k} = \begin{cases} 1 & \text{偶置換} \\ -1 & \text{奇置換} \\ 0 & \text{その他} \end{cases}$$

備考

特になし.

3.11.11 gamma_func

機能

ガンマ関数を計算する.

書式

```
result = gamma_func( x )
```

引数

x	<I,R>	in	要素.
戻り値	<R,DP>	inout	計算結果.

定義式

ガンマ関数を $\Gamma(x)$ とすると,

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt, \quad x > 0$$

備考　• 実際の評価式は近似式で評価されている. 評価方法の詳細については**??**参照.

3.11.12 kaijo

機能

階乗を計算する.

書式

```
result = kaijo( i )
```

引数

i	<I>	in	要素.
戻り値	<I,R>	inout	計算結果.

定義式

$$i! = i \times (i-1) \times \cdots \times 2 \times 1$$

備考

特になし.

3.11.13 sp_bessj

機能

第一種球ベッセル関数を計算する.

書式

```
result = sp_bessj( m, t )
```

引数

m	<I,R>	in	球ベッセル関数の次数.
t	<R,DP>	in	変数.
戻り値	<R,DP>	inout	計算結果.

定義式

m 次球ベッセル関数を $j_m(t)$ とすると,

$$j_m(t) = \left(\frac{\pi}{2t}\right)^{1/2k} J_{m+1/2}(t).$$

ここで, $J_m(t)$ は m 次ベッセル関数である.

備考
- 実際の評価式は積分形で評価されている. 評価方法の詳細については**??**参照.

3.11.14 sp_bessy

機能

第一種球ノイマン関数を計算する.

書式

```
result = sp_bessy( m, t )
```

引数

m	<I,R>	in	球ノイマン関数の次数.
t	<R,DP>	in	変数.
戻り値	<R,DP>	inout	計算結果.

定義式

m 次球ノイマン関数を $j_m(t)$ とすると,

$$y_m(t) = \left(\frac{\pi}{2t}\right)^{1/2k} Y_{m+1/2}(t).$$

ここで, $Y_m(t)$ は m 次ノイマン関数である.

備考
- 実際の評価式は積分形で評価されている. 評価方法の詳細については**??**参照.

3.12 statistics

主要な統計処理に加え, 平均化, 内挿, フィッティング処理等を行うルーチン集.

3.12.1 Anomaly_1d

機能

1 次元配列の平均値からのアノマリーを計算する。

書式

```
call Anomaly_1d( x, anor, [error] )
```

引数

x	<R(:)>	in	1 次元データ.
anor	<R(size(x))>	inout	各 x(i) に対応する偏差.
error	<R>	in	欠損値（後述）.

定義式

任意の長さのデータ列 x_n が存在した場合、そのアノマリー a_n は以下のように計算される。

$$a_i = x_i - \frac{1}{n}\sum_{p=1}^{n} x_p$$

備考

- 欠損値が定義されている場合は、平均計算の際、その値が入っているデータ点を計算に入れない。また、アノマリーとしても error で定義されている値を返す。

3.12.2 Anomaly_2d

機能

2 次元配列の平均値からのアノマリーを計算する。

書式

```
call Anomaly_2d( x, anor, [error] )
```

引数

x	<R(:,:)>	in	1 次元データ.
anor	<R(size(x,1),size(x,2))>	inout	各 x(i,j) に対応する偏差.
error	<R>	in	欠損値（後述）.

定義式

任意の長さのデータ列 x_{mn} が存在した場合、そのアノマリー a_{mn} は以下のように計算される。

$$a_{ij} = x_{ij} - \frac{1}{mn}\sum_{q=1}^{n}\sum_{p=1}^{m} x_{pq}$$

備考

- 欠損値が定義されている場合は、平均計算の際、その値が入っているデータ点を計算に入れない。また、アノマリーとしても error で定義されている値を返す。

3.12.3 Anomaly_3d

機能

3 次元配列の平均値からのアノマリーを計算する。

書式

```
call Anomaly_3d( x, anor, [error] )
```

引数

x	<R(:)>	in	1 次元データ.
anor	<R(size(x,1),size(x,2),size(x,3))>	inout	各 x(i,j,k) に対応する偏差.
error	<R>	in	欠損値（後述）.

定義式

任意の長さのデータ列 x_{lmn} が存在した場合、そのアノマリー a_{lmn} は以下のように計算される。

$$a_{ijk} = x_{ijk} - \frac{1}{lmn}\sum_{r=1}^{n}\sum_{q=1}^{m}\sum_{p=1}^{l} x_{pqr}$$

備考

- 欠損値が定義されている場合は、平均計算の際、その値が入っているデータ点を計算に入れない。また、アノマリーとしても error で定義されている値を返す。

3.12.4 auto_interpolation_1d

機能

1 次元の異なる座標系における自動線形内挿ルーチン。通常、interpo_search ルーチンと interpolation の組み合わせで行われる内挿処理を完全自動化したもの.

書式

```
call auto_interpolation_1d( x, r, u, v, undef )
```

引数

x	<R(:)>	in	内挿の基準座標.
r	<R(:)>	in	実際に内挿を行う座標.
u	<R(size(x))>	in	座標 x で定義される配列データ.
v	<R(size(r))>	inout	座標 r で定義される配列データ.
[undef]	<R>	in	未定義値.

定義式

座標 x の各格子点 x(i) で定義されている値 u(i) の値を参照値として、x と同次元に存在する格子点配置の異なる別の座標系 r の各点 r(j) で定義されている値 v(j) に内挿を行うものである. イメージは図??のようなものである.

備考

特になし.

3.12.5 auto_interpolation_2d

機能

2 次元の異なる座標系における自動線形内挿ルーチン。通常、interpo_search ルーチンと interpolation の組み合わせで行われる内挿処理を完全自動化したもの.

書式

```
call auto_interpolation_2d( x, y, r, p, u, v, undef )
```

引数

x	<R(:)>	in	内挿の基準座標 1.
y	<R(:)>	in	内挿の基準座標 2.
r	<R(:)>	in	実際に内挿を行う座標 1.
p	<R(:)>	in	実際に内挿を行う座標 2.
u	<R(size(x),size(y))>	in	座標 x,y で定義される配列データ.
v	<R(size(r),size(p))>	inout	座標 r,p で定義される配列データ.
[undef]	<R>	in	未定義値.

定義式

座標 x,y の各格子点 x(i),y(j) で定義されている値 u(i,j) の値を参照値として、x,y と同次元に存在する格子点配置の異なる別の座標系 r,p の各点 r(l), p(m) で定義されている値 v(l,m) に内挿を行うものである. イメージは図??のようなものである.

備考

特になし.

3.12.6 auto_interpolation_3d

機能

3 次元の異なる座標系における自動線形内挿ルーチン。通常、interpo_search ルーチンと interpolation の組み合わせで行われる内挿処理を完全自動化したもの.

書式

```
call auto_interpolation_3d( x, y, z, r, p, q, u, v, undef )
```

引数

x	<R(:)>	in	内挿の基準座標 1.
y	<R(:)>	in	内挿の基準座標 2.
z	<R(:)>	in	内挿の基準座標 3.
r	<R(:)>	in	実際に内挿を行う座標 1.
p	<R(:)>	in	実際に内挿を行う座標 2.
q	<R(:)>	in	実際に内挿を行う座標 3.
u	<R(size(x),size(y),size(z))>	in	座標 x,y,z で定義される配列データ.
v	<R(size(r),size(y),size(z))>	inout	座標 r,p,q で定義される配列データ.
[undef]	<R>	in	未定義値.

定義式

座標 x,y,z の各格子点 x(i),y(j),z(k) で定義されている値 u(i,j,k) の値を参照値として、x,y,z と同次元に存在する格子点配置の異なる別の座標系 r,p,q の各点 r(l),p(m),q(n) で定義されている値 v(l,m,n) に内挿を行うものである. イメージは図??のようなものである.

備考

特になし.

3.12.7 Cor_Coe

機能

2 データの相関係数を計算するルーチン.

書式

```
call Cor_Coe( x, y, cc, [error] )
call Cor_Coe_1d( x, y, cc, [error] )
```

引数

x	<R(:)>	in	1 次元データ.
y	<R(size(x))>	in	1 次元データ.
cc	<R>	inout	相関係数.
error	<R>	in	欠損値.

定義式

任意の長さの 2 データ列 x_n, y_n が存在した場合、その相関係数 cc 以下のように計算される。

$$cc = \frac{cov}{stv(x) \times stv(y)}.$$

ただし, 以下のような記号を用いた.

$$cov = \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}), \quad stv(\phi) = \sum_{i=1}^{n} (\phi_i - \overline{\phi})^2$$

ここで、cov は共分散, $stv(\phi)$ は ϕ_n というデータ列の標準偏差となる. また, 上線のついた変数は平均値を表す.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.8 Cor_Coe_2d

機能

2 データの相関係数を計算するルーチン (2 次元配列版).

書式

```
call Cor_Coe_2d( x, y, cc, [error] )
```

引数

x	<R(:,:)>	in	2 次元データ.
y	<R(size(x,1),size(x,2))>	in	2 次元データ.
cc	<R>	inout	相関係数.
error	<R>	in	欠損値.

定義式

2 次元配列を 1 次元配列に並べ替え, Cor_Coe を用いている.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.9 Cor_Coe_3d

機能

2 データの相関係数を計算するルーチン (3 次元配列版).

書式

```
call Cor_Coe_3d( x, y, cc, [error] )
```

引数

x	<R(:,:,:)>	in	3 次元データ.
y	<R(size(x,1),size(x,2),size(x,3))>	in	3 次元データ.
cc	<R>	inout	相関係数.
error	<R>	in	欠損値.

定義式

3 次元配列を 1 次元配列に並べ替え, Cor_Coe を用いている.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.10 LSM

機能

あるデータ列に対して、線形関数にフィッティングする際に、最小二乗法によってその傾きと切片を計算するルーチン.

書式

```
call LSM( x, y, slope, intercept, [undef] )
call LSM_1d( x, y, slope, intercept, [undef] )
```

引数

x	<R(:)>	in	データ要素 1.
y	<R(size(x))>	in	データ要素 2.
slope	<R>	inout	求められる線形関数の傾き.
intercept	<R>	inout	求められる線形関数の切片.
undef	<R>	in	欠損値（後述）.

定義式

任意の長さのデータ列の組 x_n, y_n が存在した場合、そのデータを最小二乗法により以下のような線形関数 $F(x_n)$ にフィッティングする。

$$F(x_i) = a_1 x_i + a_0.$$

このとき、引数は以下の対応をする。

$$x_i \ : \ \texttt{x(i)}, \quad a_1 \ : \ \texttt{slope}, \quad a_0 \ : \ \texttt{intercept}.$$

なお、この最小二乗法による具体的な計算の証明は付録の 5.1.1 参照。

備考

- 欠損値が定義されている場合は、その値が入っているデータ点を計算に入れない。

3.12.11 LSM_2d

機能

ある 2 次元データ列に対して、線形関数にフィッティングする際に、最小二乗法によってその傾きと切片を計算するルーチン.

書式

```
call LSM_2d( x, y, slope, intercept, [undef] )
```

引数

x	<R(:,:)>	in	データ要素 1.
y	<R(size(x,1),size(x,2))>	in	データ要素 2.
slope	<R>	inout	求められる線形関数の傾き.
intercept	<R>	inout	求められる線形関数の切片.
undef	<R>	in	欠損値（後述）.

定義式

2 次元配列を 1 次元に置き換えて, LSM ルーチンを用いている.

備考

- 欠損値が定義されている場合は、その値が入っているデータ点を計算に入れない。

3.12.12 LSM_3d

機能

ある 3 次元データ列に対して、線形関数にフィッティングする際に、最小二乗法によってその傾きと切片を計算するルーチン.

書式

```
call LSM_3d( x, y, slope, intercept, [undef] )
```

引数

x	<R(:,:,:)>	in	データ要素 1.
y	<R(size(x,1),size(x,2),size(x,3))>	in	データ要素 2.
slope	<R>	inout	求められる線形関数の傾き.
intercept	<R>	inout	求められる線形関数の切片.
undef	<R>	in	欠損値（後述）.

定義式

3 次元配列を 1 次元に置き換えて, LSM ルーチンを用いている.

備考

- 欠損値が定義されている場合は、その値が入っているデータ点を計算に入れない。

3.12.13 LSM_poly

機能

あるデータ列に対して、任意次数の多項式にフィッティングする際に、最小二乗法によってその各係数を計算するルーチン.

書式

```
call LSM_poly( x, y, a, intercept, [undef] )
call LSM_poly_1d( x, y, a, intercept, [undef] )
```

引数

x	<R(:)>	in	データ要素 1.
y	<R(size(x))>	in	データ要素 2.
a	<R(:)>	inout	求められる多項式の各係数（後述）.
intercept	<R>	inout	求められる多項式の切片（後述）.
undef	<R>	in	欠損値（後述）.

定義式

任意の長さのデータ列の組 x_n, y_n が存在した場合、そのデータを最小二乗法により以下のような m 次多項式 $F(x_n)$ にフィッティングする。

$$F(x_i) = \sum_{j=1}^{m} a_j x_i^j + a_0.$$

このとき、引数は以下の対応をする。

$$x_i \ : \ \texttt{x(i)}, \quad a_j \ : \ \texttt{a(j)}, \quad a_0 \ : \ \texttt{intercept}.$$

引数の形から見て明らかなように、実引数として a を渡す際、その配列の要素数が多項式の最高次数となる。また、メインルーチン側で、intercept も配列 a に組み込みたい場合、実引数側で intercept の変数として、a(0) を代入すれば問題ない。なお、この最小二乗法による具体的な計算の証明は付録の**??**参照。

備考

- 欠損値が定義されている場合は、その値が入っているデータ点を計算に入れない。

3.12.14 LSM_poly_2d

機能

あるデータ列に対して、任意次数の多項式にフィッティングする際に、最小二乗法によってその各係数を計算するルーチン (2 次元版).

書式

```
call LSM_poly_2d( x, y, a, intercept, [undef] )
```

引数

x	<R(:,:)>	in	データ要素 1.
y	<R(size(x,1),size(x,2))>	in	データ要素 2.
a	<R(:)>	inout	求められる多項式の各係数（後述）.
intercept	<R>	inout	求められる多項式の切片（後述）.
undef	<R>	in	欠損値（後述）.

定義式

2 次元データを 1 次元データに置き換えて, LSM_poly を用いている.

備考

- 欠損値が定義されている場合は、その値が入っているデータ点を計算に入れない。

3.12.15 LSM_poly_3d

機能

あるデータ列に対して、任意次数の多項式にフィッティングする際に、最小二乗法によってその各係数を計算するルーチン (3 次元版).

書式

```
call LSM_poly_3d( x, y, a, intercept, [undef] )
```

引数

x	<R(:,:,:)>	in	データ要素 1.
y	<R(size(x,1),size(x,2),size(x,3))>	in	データ要素 2.
a	<R(:)>	inout	求められる多項式の各係数（後
intercept	<R>	inout	求められる多項式の切片（後述
undef	<R>	in	欠損値（後述）.

定義式

3 次元データを 1 次元データに置き換えて, LSM_poly を用いている.

備考

- 欠損値が定義されている場合は、その値が入っているデータ点を計算に入れない。

3.12.16 Mean_1d

機能

1 次元配列の平均値を計算する。

書式

```
call Mean_1d( x, ave, [error] )
```

引数

x	<R(:)>	in	1 次元データ.
ave	<R>	inout	計算する平均値.
error	<R>	in	欠損値（後述）.

定義式

任意の長さのデータ列 x_n が存在した場合、そのアノマリー ave は以下のように計算される。

$$ave = \frac{1}{n}\sum_{p=1}^{n} x_p$$

備考

- 欠損値が定義されている場合は、平均計算の際、その値が入っているデータ点を計算に入れない。

3.12.17 Mean_2d

機能

2 次元配列の平均値を計算する。

書式

```
call Mean_2d( x, ave, [error] )
```

引数

x	<R(:,:)>	in	2 次元データ.
ave	<R>	inout	計算する平均値.
error	<R>	in	欠損値（後述）.

定義式

任意の長さのデータ列 x_{mn} が存在した場合、そのアノマリー ave は以下のように計算される。

$$ave = \frac{1}{mn}\sum_{q=1}^{n}\sum_{p=1}^{m} x_{pq}$$

備考

- 欠損値が定義されている場合は、平均計算の際、その値が入っているデータ点を計算に入れない。

3.12.18 Mean_3d

機能

3 次元配列の平均値を計算する。

書式

```
call Mean_3d( x, ave, [error] )
```

引数

x	<R(:,:,:)>	in	3 次元データ.
ave	<R>	inout	計算する平均値.
error	<R>	in	欠損値（後述）.

定義式

任意の長さのデータ列 x_{lmn} が存在した場合、そのアノマリー ave は以下のように計算される。

$$ave = \frac{1}{lmn}\sum_{r=1}^{n}\sum_{q=1}^{m}\sum_{p=1}^{l} x_{pqr}$$

備考

- 欠損値が定義されている場合は、平均計算の際、その値が入っているデータ点を計算に入れない。

3.12.19 Reg_Line

機能

回帰直線の傾きと切片を計算するルーチン.

書式

```
call Reg_Line( x, y, slope, intercept, [error] )
call Reg_Line_1d( x, y, slope, intercept, [error] )
```

引数

x	<R(:)>	in	1 次元データ.
y	<R(size(x))>	in	1 次元データ.
slope	<R>	inout	回帰直線の傾き.
intercept	<R>	inout	回帰直線の切片.
error	<R>	in	欠損値.

定義式

任意の長さの 2 データ列 x_n, y_n が存在した場合、その回帰直線 $y_n = \alpha x_n + \beta$ における傾き α と切片 β を LSM ルーチンを用いて計算する。
正確な計算方法は付録**??**参照.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.20 Reg_Line_2d

機能

2 次元データについて回帰直線の傾きと切片を計算するルーチン.

書式

```
call Reg_Line_2d( x, y, slope, intercept, [error] )
```

引数

x	<R(:,:)>	in	2 次元データ.
y	<R(size(x,1),size(x,2))>	in	2 次元データ.
slope	<R>	inout	回帰直線の傾き.
intercept	<R>	inout	回帰直線の切片.
error	<R>	in	欠損値.

定義式

2 次元データを 1 次元データに置き換えて, Reg_Line ルーチンを用いる.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.21 Reg_Line_3d

機能

3 次元データについて回帰直線の傾きと切片を計算するルーチン.

書式

```
call Reg_Line_3d( x, y, slope, intercept, [error] )
```

引数

x	<R(:,:,:)>	in	3 次元データ.
y	<R(size(x,1),size(x,2),size(x,3))>	in	3 次元データ.
slope	<R>	inout	回帰直線の傾き.
intercept	<R>	inout	回帰直線の切片.
error	<R>	in	欠損値.

定義式

3 次元データを 1 次元データに置き換えて, Reg_Line ルーチンを用いる.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.22 covariance

機能

2 データの共分散を計算するルーチン.

書式

```
call covariance( x, y, cov, [error] )
call covariance_1d( x, y, cov, [error] )
```

引数

x	<R(:)>	in	1 次元データ.
y	<R(size(x))>	in	1 次元データ.
cov	<R>	inout	共分散.
error	<R>	in	欠損値.

定義式

任意の長さの 2 データ列 x_n, y_n が存在した場合、その共分散 cov 以下のように計算される。

$$cov = \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})$$

ここで、上線のついた変数は平均値を表す.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.23 covariance_2d

機能

2 データの共分散を計算するルーチン (2 次元配列版).

書式

```
call covariance_2d( x, y, cov, [error] )
```

引数

x	<R(:,:)>	in	2 次元データ.
y	<R(size(x,1),size(x,2))>	in	2 次元データ.
cov	<R>	inout	共分散.
error	<R>	in	欠損値.

定義式

2 次元データを 1 次元データに置き換えて, covariance ルーチンを用いる.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.24 covariance_3d

機能

2 データの共分散を計算するルーチン (3 次元配列版).

書式

```
call covariance_3d( x, y, cov, [error] )
```

引数

x	<R(:,:,:)>	in	3 次元データ.
y	<R(size(x,1),size(x,2),size(x,3))>	in	3 次元データ.
cov	<R>	inout	共分散.
error	<R>	in	欠損値.

定義式

3 次元データを 1 次元データに置き換えて, covariance ルーチンを用いる.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.25 interpo_search_1d

機能

1 次元の漸増配列（要素数が増えるごとに値が大きくなる配列）の中で、point を越えない最大の要素番号を出力する（いわゆるガウス記号のような処理を行う）。

書式

```
call interpo_search_1d( x, point, i, [undeff] )
```

引数

x	<R(:)>	in	1 次元漸増配列.
point	<R>	in	調べたい点.
i	<I>	inout	point を越えない最大の x(i) に相当する要素番号.
undeff	<I>	in	探索範囲外フラグ（後述）. デフォルトはゼロ.

定義式

模式図は図??.

備考

- 探索範囲の配列要素値より小さい値を探索しようとした際に undeff を返す。探索範囲より大きい場合はどのような値をとっても、x の最大値に相当する配列要素番号を返す。

3.12.26 interpo_search_2d

機能

2 次元の漸増配列（要素数が増えるごとに値が大きくなる配列）の中で、pointx, pointy を越えない最大の要素番号を出力する（いわゆるガウス記号のような処理を行う）。

書式

```
call interpo_search_2d( x, y, pointx, pointy, i, j, [undeff] )
```

引数

x	<R(:)>	in	1 次元漸増配列.
y	<R(:)>	in	1 次元漸増配列.
pointx	<R>	in	x について調べたい点.
pointy	<R>	in	y について調べたい点.
i	<I>	inout	pointx を越えない最大の x(i) に相当する要素番号.
j	<I>	inout	pointy を越えない最大の y(j) に相当する要素番号.
undeff	<I>	in	探索範囲外フラグ（後述）. デフォルトはゼロ.

定義式

模式図は図??.

備考

- 探索範囲の配列要素値より小さい値を探索しようとした際に undeff を返す。探索範囲より大きい場合はどのような値をとっても、x の最大値に相当する配列要素番号を返す。

3.12.27 interpo_search_3d

機能

3 次元の漸増配列（要素数が増えるごとに値が大きくなる配列）の中で、pointx, pointy, pointz を越えない最大の要素番号を出力する（いわゆるガウス記号のような処理を行う）。

書式

```
call interpo_search_3d( x, y, z, pointx, pointy, pointz, i, j, k, [undeff]
)
```

引数

x	<R(:)>	in	1 次元漸増配列.
y	<R(:)>	in	1 次元漸増配列.
z	<R(:)>	in	1 次元漸増配列.
pointx	<R>	in	x について調べたい点.
pointy	<R>	in	y について調べたい点.
pointz	<R>	in	z について調べたい点.
i	<I>	inout	pointx を越えない最大の x(i) に相当する要素番号.
j	<I>	inout	pointy を越えない最大の y(j) に相当する要素番号.
k	<I>	inout	pointz を越えない最大の z(k) に相当する要素番号.
undeff	<I>	in	探索範囲外フラグ（後述）. デフォルトはゼロ.

定義式

模式図は図??.

備考

- 探索範囲の配列要素値より小さい値を探索しようとした際に undeff を返す。探索範囲より大きい場合はどのような値をとっても、x の最大値に相当する配列要素番号を返す。

3.12.28 interpolation_1d

機能

1 次元の線形内挿ルーチン。内挿点と内挿の際に参照する点および、その点における値をもとに、内挿点での値を線形内挿によって計算する。

書式

```
call interpolation_1d( x, y, point, val )
```

引数

x	<R(2)>	in	内挿点の左右端（後述）.
y	<R(2)>	in	x で定義される値（後述）.
point	<R>	in	内挿点.
val	<R>	inout	内挿点での値.

定義式

模式図は図??. ある座標配列 x_n の各点に対して定義されている値 y_n をもとに、あ

る 2 点 x_i, x_{i+1} の間の点 x_{ip} における線形内挿値を計算したいとする。このとき、内挿値 y_{ip} は近傍 2 点の値を参照して、以下のように計算される。

$$y_{ip} = y_i + dx * (x_{ip} - x_i), \quad dx = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}.$$

このとき、引数は以下の対応をする。

$$x_i \ : \ \texttt{x(1)}, \quad x_{i+1} \ : \ \texttt{x(2)}, \quad y_i \ : \ \texttt{y(1)}, \quad y_{i+1} \ : \ \texttt{y(2)},$$

$$x_{ip} \ : \ \texttt{point}, \quad y_{ip} \ : \ \texttt{val}$$

備考

- 最初に、x(1:2) の点がわからない場合は、point の値をもとに、interpo_search_1d ルーチンを用い、これらの点をあらかじめ求めておけばよい。

3.12.29 interpolation_2d

機能

2 次元の線形内挿ルーチン。内挿点と内挿の際に参照する点および、その点における値をもとに、内挿点での値を線形内挿によって計算する。

書式

```
call interpolation_2d( x, y, z, point, val )
```

引数

x	<R(2)>	in	x 方向内挿点の左右端（後述）.
y	<R(2)>	in	y 方向内挿点の左右端（後述）.
z	<R(2,2)>	in	x, y で定義される値（後述）.
point	<R(2)>	in	x, y での内挿点. point(1) が x, point(2) が y に対応.
val	<R>	inout	内挿点での値.

定義式

模式図は図**??**. ある座標配列 x_n, y_m の各点に対して定義されている値 z_{nm} をもとに、ある 4 点 $(x_i, y_j), (x_{i+1}, y_j), (x_i, y_{j+1}, x_{i+1}, y_{j+1})$ で囲まれる点 (x_p, y_q) における双線形内挿値を計算したいとする。このとき、内挿値 z_{pq} は近傍 4 点の値を参照して、

$$z_{pq} = z_{ij} + \frac{z_{i+1j} - z_{ij}}{x_{i+1} - x_i}(x_p - x_i) + \frac{z_{ij+1} - z_{ij}}{y_{j+1} - y_j}(y_q - y_j)$$
$$[z_{i+1j+1} - z_{ij+1} - z_{i+1j} + z_{ij}] \frac{(x_p - x_i)(y_q - y_j)}{(x_{i+1} - x_i)(y_{j+1} - y_j)}$$

と計算される[*7]。このとき、引数は以下の対応をする。

$$x_i \ : \ \texttt{x(1)}, \quad x_{i+1} \ : \ \texttt{x(2)}, \quad y_j \ : \ \texttt{y(1)}, \quad y_{j+1} \ : \ \texttt{y(2)},$$

$$z_{ij} \ : \ \texttt{z(1,1)}, \quad z_{i+1j} \ : \ \texttt{z(2,1)}, \quad z_{ij+1} \ : \ \texttt{z(1,2)}, \quad z_{i+1j+1} \ : \ \texttt{z(2,2)},$$

$$x_p \ : \ \texttt{point(1)}, \quad y_q \ : \ \texttt{point(2)}, \quad z_{pq} \ : \ \texttt{val}$$

備考

- 最初に、x(1:2), y(1:2) の点がわからない場合は、point の値をもとに、interpo_search_2d ルーチンを用い、これらの点をあらかじめ求めておけばよい。

3.12.30 interpolation_3d

機能

3 次元の線形内挿ルーチン。内挿点と内挿の際に参照する点および、その点における値をもとに、内挿点での値を線形内挿によって計算する。

書式

```
call interpolation_3d( x, y, z, u, point, val )
```

引数

x	<R(2)>	in	x 方向内挿点の左右端（後述）.
y	<R(2)>	in	y 方向内挿点の左右端（後述）.
z	<R(2)>	in	z 方向内挿点の左右端（後述）.
u	<R(2,2,2)>	in	x, y, z で定義される値（後述）.
point	<R(3)>	in	x, y, z での内挿点. point(1) が x, point(2) が y に, point(3) が z に対応.
val	<R>	inout	内挿点での値.

定義式

模式図は図**??**. ある座標配列 x_n, y_m, z_l の各点に対して定義されている値 u_{nml} をもとに、ある 8 点で囲まれる点 (x_p, y_q, z_r) における双線形内挿値を計算したいとする。

[*7] この式の導出は**??**参照.

このとき、内挿値 z_{pqr} は近傍 8 点の値を参照して、

$$
\begin{aligned}
u_{pqr} = u_{ijk} &+ (u_{i+1jk} - u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} + (u_{ij+1k} - u_{ijk}) \frac{(y_q - y_j)}{y_{j+1} - y_j} \\
&+ (u_{ijk+1} - u_{ijk}) \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&+ (u_{i+1j+1k} - u_{ij+1k} - u_{i+1jk} + u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} \frac{(y_q - y_j)}{y_{j+1} - y_j} \\
&+ (u_{i+1jk+1} - u_{ik+1} - u_{i+1jk} + u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&+ (u_{ij+1k+1} - u_{ijk+1} - u_{ij+1k} + u_{ijk}) \frac{(y_q - y_j)}{y_{j+1} - y_j} \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&+ (u_{i+1j+1k+1} - u_{ij+1k+1} - u_{i+1jk+1} + u_{ijk+1} - u_{i+1j+1k} + u_{ij+1k} + u_{i+1jk} - u_{ijk}) \\
&\times \frac{(x_p - x_i)}{x_{i+1} - x_i} \frac{(y_q - y_j)}{y_{j+1} - y_j} \frac{(z_r - z_k)}{z_{k+1} - z_j}
\end{aligned}
$$

と計算される[*8]。このとき、引数は以下の対応をする。

x_i : `x(1)`,　x_{i+1} : `x(2)`,　y_j : `y(1)`,　y_{j+1} : `y(2)`,

z_k : `z(1)`,　z_{k+1} : `z(2)`,

z_{ijk} : `z(1,1,1)`,　z_{i+1jk} : `z(2,1,1)`,　z_{ij+1k} : `z(1,2,1)`,　z_{ijk+1} : `z(1,1,2)`,

$z_{i+1j+1k}$: `z(2,2,1)`,　$z_{i+1jk+1}$: `z(2,1,2)`,　$z_{ij+1k+1}$: `z(1,2,2)`,　$z_{i+1j+1k+1}$: `z(2,2,2)`,

x_p : `point(1)`,　y_q : `point(2)`,　z_r : `point(3)`,　u_{pqr} : `val`

備考

- 最初に、`x(1:2)`, `y(1:2)`, `z(1:2)` の点がわからない場合は、`point` の値をもとに、`interpo_search_3d` ルーチンを用い、これらの点をあらかじめ求めておけばよい。

3.12.31 nearest_search_1d

機能

1 次元の漸増配列（要素数が増えるごとに値が大きくなる配列）の中で、`point` に最も近い要素番号を出力する。

書式

```
call nearest_search_1d( x, point, i )
```

[*8] この式の導出は**??**参照.

引数

x	<R(:)>	in	1 次元漸増配列.
point	<R>	in	調べたい点.
i	<I>	inout	point に最も近い x(i) に相当する要素番号.

定義式

模式図は図??.

備考

なし。

3.12.32 nearest_search_2d

機能

2 次元の漸増配列（要素数が増えるごとに値が大きくなる配列）の中で、pointx, pointy に最も近い要素番号を出力する。

書式

```
call nearest_search_2d( x, y, pointx, pointy, i, j )
```

引数

x	<R(:)>	in	1 次元漸増配列.
y	<R(:)>	in	1 次元漸増配列.
pointx	<R>	in	x について調べたい点.
pointy	<R>	in	y について調べたい点.
i	<I>	inout	pointx に最も近い x(i) に相当する要素番号.
j	<I>	inout	pointy に最も近い y(j) に相当する要素番号.

定義式

模式図は図??.

備考

なし。

3.12.33 nearest_search_3d

機能

3 次元の漸増配列（要素数が増えるごとに値が大きくなる配列）の中で、pointx, pointy, pointz に最も近い要素番号を出力する。

書式

```
call nearest_search_3d( x, y, z, pointx, pointy, pointz, i, j, k )
```

引数

x	<R(:)>	in	1 次元漸増配列.
y	<R(:)>	in	1 次元漸増配列.
z	<R(:)>	in	1 次元漸増配列.
pointx	<R>	in	x について調べたい点.
pointy	<R>	in	y について調べたい点.
pointz	<R>	in	z について調べたい点.
i	<I>	inout	pointx に最も近い x(i) に相当する要素番号.
j	<I>	inout	pointy に最も近い y(j) に相当する要素番号.
k	<I>	inout	pointz に最も近い z(k) に相当する要素番号.

定義式

模式図は図??.

備考

なし。

3.12.34 stand_vari

機能

2 データの標準偏差を計算するルーチン.

書式

```
call stand_vari( x, anor, [error] )
call stand_vari_1d( x, anor, [error] )
```

引数

x	<R(:)>	in	1 次元データ.
anor	<R>	inout	標準偏差.
error	<R>	in	欠損値.

定義式

任意の長さのデータ列 x_n が存在した場合、その標準偏差 stv 以下のように計算される。

$$stv = \sum_{i=1}^{n} (x_i - \overline{x})^2$$

ここで、上線のついた変数は平均値を表す.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.35 stand_vari_2d

機能

2 データの標準偏差を計算するルーチン (2 次元配列版).

書式

```
call stand_vari_2d( x, anor, [error] )
```

引数

x	<R(:,:)>	in	2 次元データ.
anor	<R>	inout	標準偏差.
error	<R>	in	欠損値.

定義式

2 次元データを 1 次元データに置き換え, stand_vari ルーチンを用いる.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.36 stand_vari_3d

機能

2 データの標準偏差を計算するルーチン (3 次元配列版).

書式

```
call stand_vari_3d( x, anor, [error] )
```

引数

x	<R(:,:,:)>	in	3 次元データ.
anor	<R>	inout	標準偏差.
error	<R>	in	欠損値.

定義式

3 次元データを 1 次元データに置き換え, stand_vari ルーチンを用いる.

備考

- 欠損値が定義されている場合は, 計算の際, そのデータを計算しない.

3.12.37 Move_ave

機能

1 次元配列の移動平均値を計算する。

書式

```
call Move_ave( x, n, y, [error] )
```

引数

x	<R(:)>	in	1 次元データ.
n	<I>	in	平均をとる要素数.
y	<R(size(x))>	inout	平均した結果.
error	<R>	in	欠損値（後述）.

定義式

任意の長さのデータ列 x_i が存在した場合、その移動平均 y_i は以下のように計算される。ただし、n が移動平均する要素数である。

$$y_i = \frac{1}{n} \sum_{p=i}^{i+n-1} x_p$$

備考

- 欠損値は予約引数として用意してあるが、現在は使用していない。

- 移動平均の計算は定義式では上のように記述してあるが、実装は最初の n 個について平均を計算し、以降は最古値を引き、最新値を足すという作業を行っている。

3.13 Thermo_Advanced_Function

熱力学変数の処理を行う関数のうち, 他のモジュールに依存した計算を行う関数集.

3.13.1 CAPE

機能

対流有効位置エネルギー (Convective Available Potential Energy) を計算するルーチン.

書式

```
result = CAPE( p, z, qv, temp, z_ref, [undeff], [opt] )
```

引数

p	<R(:)>	in	気圧 [Pa].
z	<R(size(p))>	in	高度 [m].
qv	<R(size(p))>	in	水蒸気混合比 [kg/kg].
temp	<R(size(p))>	in	温度 [K].
z_ref	<R>	in	基準高度 [m].
戻り値	<R>	inout	CAPE [J/kg].
undeff	<R>	in	未定義値.
opt	<I>	in	LNB 計算の際のオプション (後述).

定義式

$$\int_{p_{\mathrm{LNB}}}^{p_{\mathrm{LFC}}} R(T_p - T_a) d\ln p.$$

ここで、p は気圧、T_a は環境場の温度 (つまり、ゾンデで観測される温度)、T_p は断熱減率に従って、基準高度 z_{ref} から上昇させたパーセルの温度、R は気体定数、LFC は自由対流高度 (Level of Free Convection)、LNB は中立浮力高度 (Level of Neutral Buoyancy).

備考

- 引数として与える各配列は、要素番号の小さい方がより地表に近い値となるように、つまり配列としてデータを与える際は、地表面から格納していく。
- 高度データは LFC, LNB の値を計算するためにのみ使用している。
- undeff が設定されている高度の値は積算されない。
- もし、基準高度を気圧基準で与える場合は、この関数を呼び出す前に基準気圧での高度を計算し、その値を引数として入れれば OK.
- opt 変数については, 関数 z_LNB でのオプション指定と同じ.

3.13.2 CIN

機能

対流抑制エネルギー (Convective INhibition) を計算するルーチン.

書式

```
result = CIN( p, z, qv, temp, z_ref, [undeff] )
```

引数

p	<R(:)>	in	気圧 [Pa].
z	<R(size(p))>	in	高度 [m].
qv	<R(size(p))>	in	水蒸気混合比 [kg/kg].
temp	<R(size(p))>	in	温度 [K].
z_ref	<R>	in	基準高度 [m].
戻り値	<R>	inout	CIN [J/kg].
undeff	<R>	in	未定義値.

定義式

$$\int_{p_{\mathrm{LFC}}}^{p_{ref}} R(T_p - T_a) d\ln p.$$

ここで、p は気圧、T_a は環境場の温度 (つまり、ゾンデで観測される温度)、T_p は断熱減率に従って、基準高度 z_{ref} から上昇させたパーセルの温度、R は気体定数、LFC は自由対流高度 (Level of Free Convection)、p_{ref} は z_{ref} での気圧.

備考

- 引数として与える各配列は、要素番号の小さい方がより地表に近い値となるように、つまり配列としてデータを与える際は、地表面から格納していく。
- 高度データは LFC, LNB の値を計算するためにのみ使用している。
- undeff が設定されている高度の値は積算されない。

- もし、基準高度を気圧基準で与える場合は、この関数を呼び出す前に基準気圧での高度を計算し、その値を引数として入れれば OK.

3.13.3 T_LFC

機能

LFC (Level of Free Convection) での温度を計算する関数.

書式

```
result = T_LFC( z_ref, z, temp, p, qv )
```

引数

z_ref	<R>	in	基準高度 [m].
z	<R(size(p))>	in	高度 [m].
temp	<R(size(p))>	in	温度 [K].
p	<R(:)>	in	気圧 [Pa].
qv	<R(size(p))>	in	水蒸気混合比 [kg/kg].
戻り値	<R>	inout	LFC での温度 [K].

定義式

断熱図におけるパラメータの詳細は**??**参照.

備考

- 引数として与える各配列は、要素番号の小さい方がより地表に近い値となるように、つまり配列としてデータを与える際は、地表面から格納していく。
- ある基準高度での相当温位を上昇させたとき, 周囲の飽和相当温位と一致する高度が LFC である. ここでは, 周囲の飽和相当温位と交差する 2 点を求め, そこから高度方向に内挿することで高度を決定する.

3.13.4 T_LNB

機能

LNB (Level of Neutral Buoyancy) での温度を計算する関数.

書式

```
result = T_LNB( z_ref, z, temp, p, qv, [option] )
```

引数

T_ref	<R>	in	基準高度 [m].
z	<R(size(p))>	in	高度 [m].
temp	<R(size(p))>	in	温度 [K].
p	<R(:)>	in	気圧 [Pa].
qv	<R(size(p))>	in	水蒸気混合比 [kg/kg].
戻り値	<R>	inout	LNB での温度 [K].
option	<R>	in	計算方法 (備考参照).

定義式

断熱図におけるパラメータの詳細は**??**参照.

備考

- 引数として与える各配列は、要素番号の小さい方がより地表に近い値となるように、つまり配列としてデータを与える際は、地表面から格納していく。
- option には数字が入る. 本関数では、LNB を計算するために、2 種類の方法をとることができる。1 つめは LFC 高度での飽和相当温位が LFC より上空で再び環境場の飽和相当温位と交わる点を計算し、その高度を LNB とする方法である。しかし、実際の高分解能な高層気象観測データでは、湿度のデータが大きく変動する場合があるため、LFC のすぐ上で LNB を迎えてしまい、正確に計算されない場合がある。そこで、2 つめは高層気象観測データの上端から下向きに飽和相当温位を計算し、最初に LFC での飽和相当温位と同じ値を迎えた高度を LNB とする方法で計算を行う。飽和相当温位は上層では漸増関数となるため、この手法をとっても問題はない。デフォルトでは option = 1 であり、これは 1 つめの手法による計算である。

3.13.5 precip_water

機能

可降水量を計算する関数.

書式

```
result = precip_water( p, qv, [undef] )
```

引数

p	<R(:)>	in	気圧 [Pa].
qv	<R(size(p))>	in	水蒸気混合比 [kg/kg].
戻り値	<R>	inout	可降水量 [kg/m^2].
undef	<R>	in	未定義値.

定義式

断熱図におけるパラメータの詳細は**??**参照.

備考

- 引数として与える各配列は、要素番号の小さい方がより地表に近い値となるように、つまり配列としてデータを与える際は、地表面から格納していく。

3.13.6 qrsg_2_dbz

機能

凝結物混合比から擬似的なレーダ反射強度を計算する関数.

書式

```
result = z_LFC( rho, qr, [qs], [qg] )
```

引数

rho	<R>	in	基準高度 [kg/m^3].
qr	<R>	in	雨水混合比 [kg/kg].
qs	<R>	in	雪混合比 [kg/kg].
qg	<R>	in	霰混合比 [kg/kg].
戻り値	<R>	inout	レーダ反射強度 [dBZ].

定義式

レーダ反射強度 dBZ は

$$\mathrm{dBZ} \equiv 10 \log_{10} Z$$

で定義されている. 本関数では, この Z の計算に Murakami (1990) の (54) 式：

$$\begin{aligned} Z = {} & \Gamma(7) N_{R0} \left(\frac{\bar{\rho} Q_R}{\pi \rho_W N_{R0}} \right)^{1.75} 10^{18} \\ & + \Gamma(7) \left[\left| \frac{\varepsilon_i - 1}{\varepsilon_i + 2} \right|^2 \Big/ \left| \frac{\varepsilon_w - 1}{\varepsilon_w + 2} \right|^2 \right] \left(\frac{\rho_s}{\rho_W} \right)^2 N_{s0} \left(\frac{\bar{\rho} Q_s}{\pi \rho_s N_{s0}} \right)^{1.75} 10^{18} \\ & + \Gamma(7) \left[\left| \frac{\varepsilon_i - 1}{\varepsilon_i + 2} \right|^2 \Big/ \left| \frac{\varepsilon_w - 1}{\varepsilon_w + 2} \right|^2 \right] \left(\frac{\rho_g}{\rho_W} \right)^2 N_{g0} \left(\frac{\bar{\rho} Q_g}{\pi \rho_g N_{g0}} \right)^{1.75} 10^{18} \end{aligned}$$

を用いている。ここで、各記号はそれぞれ以下のとおりである。

$\Gamma(x)$ ガンマ関数.
ε_i 氷の比誘電率.
ε_w 水の比誘電率.
N_{R0} 雨の数密度.
N_{s0} 雪の数密度.
N_{g0} 霰の数密度.
ρ_w 水の密度.
ρ_s 雪の密度.
ρ_q 霰の密度.
$\bar{\rho}$ 基本場の大気密度.

各物理パラメータは Murakami (1990) に記載されている値を用いるが, 記載されていないパラメータについては, 理科年表 (2008) の値を参考にしている. レーダ反射強度の詳細な計算は**??**参照.

備考

特になし.

3.13.7 z_LCL

機能

LCL (Lifted Condensation Level) を計算する関数.

書式

```
result = z_LCL( z_ref, z, temp, p, qv )
```

引数

z_ref	<R>	in	基準高度 [m].
z	<R(size(p))>	in	高度 [m].
temp	<R(size(p))>	in	温度 [K].
p	<R(:)>	in	気圧 [Pa].
qv	<R(size(p))>	in	水蒸気混合比 [kg/kg].
戻り値	<R>	inout	LCL 高度 [m].

定義式

断熱図におけるパラメータの詳細は**??**参照.

備考

- 引数として与える各配列は、要素番号の小さい方がより地表に近い値となるように、つまり配列としてデータを与える際は、地表面から格納していく。

3.13.8 z_LFC

機能

LFC (Level of Free Convection) を計算する関数.

書式

```
result = z_LFC( z_ref, z, temp, p, qv )
```

引数

z_ref	<R>	in	基準高度 [m].
z	<R(size(p))>	in	高度 [m].
temp	<R(size(p))>	in	温度 [K].
p	<R(:)>	in	気圧 [Pa].
qv	<R(size(p))>	in	水蒸気混合比 [kg/kg].
戻り値	<R>	inout	LFC 高度 [m].

定義式

断熱図におけるパラメータの詳細は**??**参照.

備考

- 引数として与える各配列は、要素番号の小さい方がより地表に近い値となるように、つまり配列としてデータを与える際は、地表面から格納していく。
- ある基準高度での相当温位を上昇させたとき, 周囲の飽和相当温位と一致する高度が LFC である. ここでは, 周囲の飽和相当温位と交差する 2 点を求め, そこから高度方向に内挿することで高度を決定する.

3.13.9 z_LNB

機能

LNB (Level of Neutral Buoyancy) を計算する関数.

書式

```
result = z_LNB( z_ref, z, temp, p, qv, [option] )
```

引数

z_ref	<R>	in	基準高度 [m].
z	<R(size(p))>	in	高度 [m].
temp	<R(size(p))>	in	温度 [K].
p	<R(:)>	in	気圧 [Pa].
qv	<R(size(p))>	in	水蒸気混合比 [kg/kg].
戻り値	<R>	inout	LNB 温度 [m].
option	<R>	in	計算方法 (備考参照).

定義式

断熱図におけるパラメータの詳細は**??**参照.

備考

- 引数として与える各配列は、要素番号の小さい方がより地表に近い値となるように、つまり配列としてデータを与える際は、地表面から格納していく。
- option には数字が入る. 本関数では、LNB を計算するために、2 種類の方法をとることができる。1 つめは LFC 高度での飽和相当温位が LFC より上空で再び環境場の飽和相当温位と交わる点を計算し、その高度を LNB とする方法である。しかし、実際の高分解能な高層気象観測データでは、湿度のデータが大きく変動する場合があるため、LFC のすぐ上で LNB を迎えてしまい、正確に計算されない場合がある。そこで、2 つめは高層気象観測データの上端から下向きに飽和相当温位を計算し、最初に LFC での飽和相当温位と同じ値を迎えた高度を LNB とする方法で計算を行う。飽和相当温位は上層では漸増関数となるため、この手法をとっても問題はない。デフォルトでは option = 1 であり、これは 1 つめの手法による計算である。

3.14 thermo_advanced_routine

熱力学関係の処理ルーチンの中で, スカラー値ではなく配列形式で引数を返す処理を行うルーチンのうち, 他のモジュールに依存する処理を行うものを集めたモジュール.

ここで用いている多くの熱力学変数の詳細は**??**参照。

3.14.1 Brunt_Freq

機能

ブラントバイサラ振動数の 2 乗を計算する.

書式

```
call Brunt_Freq( x, y, z, pt, BV, [undef] )
```

引数

x	<R(:)>	in	右手系第一座標.
y	<R(:)>	in	右手系第二座標.
z	<R(:)>	in	右手系第三座標.
pt	<R(size(x),size(y),size(z))>	in	温位 [K].
BV	<R(size(x),size(y),size(z))>	inout	ブラントバイサラ振動数 [/s].
undef	<R>	in	未定義値.

定義式

ブラントバイサラ振動数を N とすると、その定義は

$$N^2 = \frac{g}{\theta}\frac{\partial \theta}{\partial z}$$

となる。ここで、g, θ はそれぞれ重力加速度と温位である。

備考

特になし.

3.14.2 Ertel_PV

機能

エルテルのポテンシャル渦度を計算する.

書式

```
call Ertel_PV( x, y, z, u, v, w, rho, pt, cor, PV, [undef], [sx], [sy],
[sz] )
```

引数

x	<R(:)>	in	右手系第一座標.
y	<R(:)>	in	右手系第二座標.
z	<R(:)>	in	右手系第三座標.
u	<R(size(x),size(y),size(z))>	in	x 方向の速度.
v	<R(size(x),size(y),size(z))>	in	y 方向の速度.
w	<R(size(x),size(y),size(z))>	in	z 方向の速度.
rho	<R(size(x),size(y),size(z))>	in	密度 [kg/m^3].
pt	<R(size(x),size(y),size(z))>	in	温位 [K].
cor	<R(size(x),size(y))>	in	コリオリパラメータ [/s].
BV	<R(size(x),size(y),size(z))>	inout	エルテルの PV [K/kg s m^2].
undef	<R>	in	未定義値.

定義式

エルテルのポテンシャル渦度を P とすると、その定義は

$$P = \frac{\boldsymbol{\omega}_a \cdot \boldsymbol{\nabla}\theta}{\rho}$$

となる。ここで、$\boldsymbol{\omega}_a, \theta, \rho$ はそれぞれ絶対渦度、温位、密度である。

備考

特になし.

3.15 thermo_const

熱力学的定数集. 熱力学学における普遍定数を以下の表のように与える. 出典は理科年表 (2008) より.

変数名	型	設定値	変数の意味	単位
Cpd	<R>	1004.0	乾燥大気の定圧比熱	[J/K kg]
Cpv	<R>	1870.0	水蒸気の定圧比熱	[J/K kg]
LH0	<R>	2.5e6	水蒸気の潜熱	[J/kg]
LHS	<R>	2.8e6	水蒸気の昇華潜熱	[J/kg]
Md	<R>	28.96	乾燥大気の分子量	[kg/kmol]
Mv	<R>	18.0	水蒸気の分子量	[kg/kmol]
Rd	<R>	287.0	乾燥大気の気体定数	[J/K kg]
Rv	<R>	461.0	水蒸気の気体定数	[J/K kg]
e0	<R>	611.0	凝固点での水蒸気圧	[Pa]
ei0	<R>	611.73	凝固点での氷飽和での水蒸気圧	[Pa]
eps_rdrv	<R>	287.0/461.0	Rd/Rv	[1]
kalm	<R>	0.4	カルマン定数	[1]
p0	<R>	1.0e5	基準気圧	[Pa]
p00	<R>	1.01325e5	標準大気圧	[Pa]
rhow	<R>	1.0e3	水の密度	[kg/m^3]
t0	<R>	273.15	標準気圧での水の凝固点	[K]
ti0	<R>	273.16	水の三重点での凝固温度	[K]

3.16 thermo_function

熱力学に関する物理量間の変換関数集. 基本的に, スカラー変数を返すような処理はすべてこのモジュールに組み込まれる. それは引数が配列であっても, 返す量がスカラー変

数である場合についても適用される.

なお, すべてのルーチンで用いられている変換関数の定義は**??**参照。また, 各計算式の詳細な導出も同様の節参照.

3.16.1 LH

機能

キルヒホッフの式 (**??**) から潜熱の温度依存性を考慮した潜熱の値を計算する.

書式

```
result=LH( T )
```

引数

T	<R>	in	大気の温度 [K].
戻り値	<R>	inout	潜熱 [J/K kg].

定義式

$$LH(T) = LH_0 - 232(T - T_0), \quad LH_0 = 2.5 \times 10^5.$$

ここで, $T_0 = 273.15$ K である。

備考

特になし.

3.16.2 RHTP_2_qv

機能

相対湿度と温度と圧力から水蒸気混合比を計算する.

書式

```
result=RHTP_2_qv( RH, T, P )
```

引数

RH	<R>	in	大気の相対湿度 [%].
T	<R>	in	大気の温度 [K].
P	<R>	in	大気の全圧 [Pa].
戻り値	<R>	inout	大気の水蒸気混合比 [kg/kg].

定義式

計算過程は以下である.

RHT_2_e(RH,T) ⇒ 水蒸気圧 e⇒eP_2_qv(e,P)⇒ 水蒸気混合比

備考

特になし.

3.16.3 RHT_2_e

機能

相対湿度と温度から水蒸気圧を計算する.

書式

```
result=RHT_2_e( RH, T )
```

引数

RH	<R>	in	大気の相対湿度 [%].
T	<R>	in	大気の温度 [K].
戻り値	<R>	inout	大気の水蒸気分圧 [Pa].

定義式

相対湿度 RH と飽和水蒸気圧 e_s、水蒸気分圧 e の定義式：

$$RH \equiv \frac{e}{e_s} \times 100$$

を元に計算する。ここで、飽和水蒸気圧は温度から es_Bolton を用いて計算した値を用いる。

備考

特になし.

3.16.4 TP_2_qvs

機能

大気の温度と全圧から飽和混合比を計算する. ここで用いる飽和水蒸気圧の計算は es_Bolton による式を用いて行われるものとする.

書式

```
result=TP_2_qvs( T, P )
```

引数

T	<R>	in	大気の温度 [K].
P	<R>	in	大気の圧力 [Pa].
戻り値	<R>	inout	大気の飽和混合比 [kg/kg].

定義式

大気の飽和混合比を q_{vs} とすると、その定義から

$$q_{vs} = \frac{\rho_v}{\rho_d}$$

である。ここで、ρ_v, ρ_d はそれぞれ水蒸気の密度と乾燥大気の密度である。理想気体の状態方程式から、それぞれの密度は

$$\rho_v = \frac{e_s}{R_v T}, \quad \rho_d = \frac{p_d}{R_d T}$$

である。ここで、e_s, p_d, R_v, R_d, T はそれぞれ飽和水蒸気圧, 乾燥大気の分圧、水蒸気の気体定数、乾燥大気の気体定数、大気の温度である。よって、

$$q_{vs} = \frac{R_d}{R_v} \frac{e_s}{p - e_s}$$

となる。ここで、乾燥大気の分圧は大気の全圧 p から飽和水蒸気圧を差し引いたものであるので、上式は成り立つ。e_s は温度のみの関数であるため、大気の温度と全圧を与えれば、飽和混合比が得られることがわかる。

備考

特になし.

3.16.5 TP_2_rho

機能

乾燥大気の状態方程式から, 温度と気圧を与えて密度を得る.

書式

```
result=TP_2_rho( T, P )
```

引数

T	<R>	in	大気の温度 [K].
P	<R>	in	大気の圧力 [Pa].
戻り値	<R>	inout	大気の密度 [kg/m^3].

定義式

大気の状態方程式 :

$$p = \rho RT.$$

ここで、p, ρ, R, T はそれぞれ大気の圧力、密度、乾燥大気の気体定数、大気の温度である。すべての変数は MKS 単位系で処理される。

備考

特になし.

3.16.6 TqvP_2_TLCL

機能

Bolton (1980) の式から LCL 高度での温度を計算する.

書式

```
result=TqvP_2_TLCL( T, qv, pres )
```

引数

T	<R>	in	大気の温度 [K].
qv	<R>	in	大気の水蒸気混合比 [kg/kg].
pres	<R>	in	大気の気圧 [Pa].
戻り値	<R>	inout	LCL での温度 [K].

定義式

LCL での気温を T_{LCL} とすると、

$$T_{\mathrm{LCL}} = \frac{c}{a \ln T - \ln e - b} + 55, \quad a = 3.5,\ b = 4.805,\ c = 2840$$

ここで, e は水蒸気圧である。

備考

特になし.

3.16.7 TqvP_2_thetae

機能

相当温位を定義式から計算する.

書式

```
result=TqvP_2_thetae( T, qv, pres )
```

引数

T	<R>	in	大気の温度 [K].
qv	<R>	in	大気の水蒸気混合比 [kg/kg].
pres	<R>	in	大気の気圧 [Pa].
戻り値	<R>	inout	相当温位 [K].

定義式

相当温位を θ_e とすると、

$$\theta_e = \theta_d \exp\left[\frac{L(T_{\rm LCL})q_{vs}}{C_{pd}T_{\rm LCL}}\right]$$

ここで, $\theta_d, L, C_{pd}, T_{\rm LCL}, q_{vs}$ はそれぞれ乾燥大気の温位、水蒸気の潜熱、乾燥定圧比熱、LCL 温度、飽和混合比である。

備考

特になし.

3.16.8 TqvP_2_thetaes

機能

飽和相当温位を定義式から計算する.

書式

```
result=TqvP_2_thetaes( T, pres )
```

引数

T	<R>	in	大気の温度 [K].
pres	<R>	in	大気の気圧 [Pa].
戻り値	<R>	inout	相当温位 [K].

定義式

飽和相当温位を θ_{es} とすると、

$$\theta_{es} = \theta_d \exp\left[\frac{L(T)q_{vs}}{C_{pd}T}\right]$$

ここで, $\theta_d, L, C_{pd}, T, q_{vs}$ はそれぞれ乾燥大気の温位、水蒸気の潜熱、乾燥定圧比熱、温度、飽和混合比である。

備考

特になし.

3.16.9 eP_2_qv

機能

水蒸気圧と圧力から水蒸気混合比を計算する.

書式

```
result=eP_2_qv( e, P )
```

引数

e	<R>	in	大気の水蒸気圧 [Pa].
P	<R>	in	大気の全圧 [Pa].
戻り値	<R>	inout	大気の水蒸気混合比 [kg/kg].

定義式

$$q_v = \frac{\varepsilon e}{P - e}, \quad \varepsilon = \frac{R_d}{R_v}.$$

R_d, R_v はそれぞれ、乾燥大気、水蒸気の気体定数。

備考

特になし.

3.16.10 eT_2_RH

機能

水蒸気圧と温度から相対湿度を計算する.

書式

```
result=eT_2_RH( e, T )
```

引数

e	<R>	in	大気の水蒸気圧 [Pa].
T	<R>	in	大気の温度 [K].
戻り値	<R>	inout	大気の相対湿度 [%].

定義式

相対湿度 RH と飽和水蒸気圧 e_s、水蒸気分圧 e の定義式：

$$RH \equiv \frac{e}{e_s} \times 100$$

を元に計算する。ここで、飽和水蒸気圧は温度から es_Bolton を用いて計算した値を用いる。

備考

特になし.

3.16.11 es_Bolton

機能

Bolton (1980) の式から温度を元に飽和水蒸気圧を計算する.

書式

```
result=es_Bolton( T )
```

引数

T	<R>	in	大気の温度 [K].
戻り値	<R>	inout	大気の飽和水蒸気圧 [Pa].

定義式

$$e_s(T) = e_0 \exp a\frac{T-T_0}{T-c}, \quad a = 17.67,\ c = 29.65$$

ここで, $T_0 = 273.15$ K, $e_0 = 611.0$ Pa, $e_s(T)$ は飽和水蒸気圧である。

備考

特になし.

3.16.12 es_TD

機能

Bolton (1980) の式の逆算で、水蒸気圧を与えることでそのときの露点温度を計算する.

書式

```
result=es_TD( e )
```

引数

e	<R>	in	大気の水蒸気圧 [Pa].
戻り値	<R>	inout	大気の露点温度 [K].

定義式

$$T_D = c + a\frac{T_0 - c}{a - \log(e/e0)}, \quad a = 17.67,\ c = 29.65$$

ここで, $T_0 = 273.15$ K, $e_0 = 611.0$ Pa, T_D は露点温度である。

備考

特になし.

3.16.13 exner_func_dry

機能

乾燥大気におけるエクスナー関数を計算する.

書式

```
result=exner_func_dry( P )
```

引数

P	<R>	in	大気の圧力 [Pa].
戻り値	<R>	inout	乾燥大気のエクスナー関数 [無次元].

定義式

エクスナー関数 π_d の定義式 :

$$\pi_d = T \left(\frac{p}{p_0} \right)^{R_d/C_{pd}}$$

から求める. ここで、p_0 はエクスナー関数の基準気圧である。R_d, C_{pd} はそれぞれ、乾燥大気における気体定数と定圧比熱である。

備考

特になし.

3.16.14 get_gamma_d

機能

乾燥大気の断熱減率を取得する.

書式

```
result=get_gamma_d( )
```

引数

戻り値	<R>	inout	乾燥断熱減率 [K/m].

定義式

乾燥断熱減率 Γ_d は定圧比熱 C_{pd} と重力加速度 g を用いて、

$$\Gamma_d = -\frac{g}{C_{pd}}$$

備考

特になし.

3.16.15 goff_gratch

機能

Goff-Gratch の式から温度を元に飽和水蒸気圧を計算する.

書式

```
result=goff_gratch( T )
```

引数

T	<R>	in	大気の温度 [K].
戻り値	<R>	inout	大気の飽和水蒸気圧 [Pa].

定義式

$$pow = a\left(\frac{tst}{T} - 1.0\right) + b\log_{10}\left(\frac{tst}{T}\right) + c\left(10.0^{pa(1.0-T/tst)} - 1.0\right) + d\left(10.0^{pb(tst/T-1.0)} - 1.0\right),$$

$$e_s(T) = p_0 \times 10^{pow}.$$

ここで、

$$a = -7.90298,\ b = 5.02808,\ c = -1.3816 \times 10^{-7},\ d = 8.1328 \times 10^{-3},$$

$$pa = 11.344,\ pb = -3.49149,\ tst = 373.15,\ p_0 = 1.01325 \times 10^5$$

備考

特になし.

3.16.16 goff_gratch_i

機能

Goff-Gratch の式から温度を元に氷飽和で飽和水蒸気圧を計算する.

書式

```
result=goff_gratch_i( T )
```

引数

T	<R>	in	大気の温度 [K].
戻り値	<R>	inout	大気の飽和水蒸気圧 [Pa].

定義式

$$pow = a\left(\frac{t_{i0}}{T} - 1.0\right) + b\log_{10}\frac{t_{i0}}{T} + c\left(1.0 - \frac{T}{t_{i0}}\right)$$
$$e_s(T) = p_0 \times 10^{pow}.$$

ここで、

$$a = -9.09718,\ b = -3.56654,\ c = 0.876793,$$
$$p_0 = 611.73,\ t_{i0} = 273.16.$$

備考

特になし.

3.16.17 hypsometric_form

機能

ある高度における気圧の補正を行う. 高度に海面を与えれば海面更正気圧を計算することができる.

書式

```
result=hypsometric_form( p, z, T, [z_t] )
```

引数

p	<R>	in	基準高度での圧力 [Pa].
z	<R>	in	基準高度 [m].
T	<R>	in	基準高度での温度 [K].
z_t	<R>	in	更正を行う高度 [m]. デフォルト値は 0 m.
戻り値	<R>	inout	z_t での温度 [K].

定義式

対流圏における標準大気の気温減率を Γ =6.5 K/km として、高度 z_t での気圧 $p(z_t)$ は、既に気圧、温度の分かっている高度 z での値 $p(z), T(z)$ を用いて、

$$p(z_t) = p(z)\left(\frac{T(z) + \Gamma z}{T(z) + \Gamma z_t}\right)^{g/(\Gamma R_d)}$$

から求める. ここで、R_d は乾燥大気における気体定数である。

備考

特になし.

3.16.18 TP_2_qv

機能

温度と圧力から水蒸気混合比を計算する.

書式

```
result=TP_2_qv( T, P )
```

引数

T	<R>	in	大気の温度 [K].
P	<R>	in	大気の全圧 [Pa].
戻り値	<R>	inout	大気の水蒸気混合比 [kg/kg].

定義式

計算過程は, es_Bolton で温度から飽和水蒸気圧を計算し、その飽和水蒸気圧 E_s を用いて、

$$q_{vs} = \frac{\varepsilon e_s}{P - e_s}, \quad \varepsilon = \frac{R_d}{R_v}.$$

R_d, R_v はそれぞれ、乾燥大気、水蒸気の気体定数。

備考

特になし.

3.16.19 qvTP_2_RH

機能

水蒸気混合比と温度と圧力から相対湿度を計算する.

書式

```
result=qvTP_2_RH( qv, T, P )
```

引数

qv	<R>	in	大気の水蒸気混合比 [kg/kg].
T	<R>	in	大気の温度 [K].
P	<R>	in	大気の全圧 [Pa].
戻り値	<R>	inout	大気の相対湿度 [%].

定義式

計算過程は以下である.

qvP_2_e(qv,P) ⇒ 水蒸気圧 e⇒eT_2_RH(e,T)⇒ 相対湿度

備考

特になし.

3.16.20 rhoP_2_T

機能

乾燥大気の状態方程式から, 密度と気圧を与えて温度を得る.

書式

```
result=rhoP_2_T( rho, P )
```

引数

rho	<R>	in	大気の密度 [kg/m^3].
P	<R>	in	大気の圧力 [Pa].
戻り値	<R>	inout	大気の温度 [K].

定義式

大気の状態方程式 :

$$p = \rho RT.$$

ここで、p, ρ, R, T はそれぞれ大気の圧力、密度、乾燥大気の気体定数、大気の温度である。すべての変数は MKS 単位系で処理される。

備考

特になし.

3.16.21 rhoT_2_P

機能

乾燥大気の状態方程式から, 密度と温度を与えて気圧を得る.

書式

```
result=rhoT_2_P( rho, T )
```

引数

rho	<R>	in	大気の密度 [kg/m^3].
T	<R>	in	大気の温度 [K].
戻り値	<R>	inout	大気の気圧 [Pa].

定義式

大気の状態方程式 :

$$p = \rho RT.$$

ここで、p, ρ, R, T はそれぞれ大気の圧力、密度、乾燥大気の気体定数、大気の温度である。すべての変数は MKS 単位系で処理される。

備考

特になし.

3.16.22 tetens

機能

tetens の実験式から温度を元に飽和水蒸気圧を計算する.

書式

```
result=tetens( T )
```

引数

T	<R>	in	大気の温度 [K].
戻り値	<R>	inout	大気の飽和水蒸気圧 [Pa].

定義式

$$e_s(T) = e_0 \times 10^{a\frac{T-T_0}{T-T_0+b}}, \quad \begin{cases} a = 7.5,\ b = 237.7 & (T \geq T_0) \\ a = 9.5,\ b = 265.5 & (T < T_0) \end{cases}$$

ここで, $T_0 = 273.15$ K, $e_s(T)$ は飽和水蒸気圧である。

備考

特になし.

3.16.23 thetaP_2_T

機能

温位、圧力から温度を計算する.

書式

```
result=thetaP_2_T( theta, P )
```

引数

theta	<R>	in	大気の温位 [K].
P	<R>	in	大気の圧力 [Pa].
戻り値	<R>	inout	大気の温度 [K].

定義式

温位の定義式 :

$$T = \theta \left(\frac{P}{p_0} \right)^{R_d/C_{pd}}$$

から求める. ここで、p_0 はエクスナー関数の基準気圧である。R_d, C_{pd} はそれぞれ、乾燥大気における気体定数と定圧比熱である。

備考

特になし.

3.16.24 thetaT_2_P

機能

温位、温度から気圧を計算する.

書式

```
result=thetaT_2_P( theta, T )
```

引数

theta	<R>	in	大気の温位 [K].
T	<R>	in	大気の温度 [K].
戻り値	<R>	inout	大気の圧力 [Pa].

定義式

温位の定義式 :

$$P = p_0 \left(\frac{T}{\theta} \right)^{C_{pd}/R_d}$$

から求める. ここで、p_0 はエクスナー関数の基準気圧である。R_d, C_{pd} はそれぞれ、乾燥大気における気体定数と定圧比熱である。

備考

特になし.

3.16.25 theta_dry

機能

乾燥大気における温位を計算する.

書式

```
result=theta_dry( T, P )
```

引数

T	<R>	in	大気の温度 [K].
P	<R>	in	大気の圧力 [Pa].
戻り値	<R>	inout	乾燥大気の温位 [K].

定義式

温位の定義式 :

$$\theta_d = T\left(\frac{p_0}{P}\right)^{R_d/C_{pd}}$$

から求める. ここで、p_0 はエクスナー関数の基準気圧である。R_d, C_{pd} はそれぞれ、乾燥大気における気体定数と定圧比熱である。

備考

特になし.

3.16.26 theta_moist

機能

湿潤大気における温位を計算する.

書式

```
result=theta_moist( T, P, qv )
```

引数

T	<R>	in	大気の温度 [K].
P	<R>	in	大気の圧力 [Pa].
qv	<R>	in	大気の水蒸気混合比 [kg/kg].
戻り値	<R>	inout	湿潤大気の温位 [K].

定義式

湿潤大気における温位の定義式 :

$$\theta_m = T\left(\frac{p_0}{P}\right)^{pow}, \quad pow = \frac{R_d}{C_{pd}}\frac{1+q_v/\varepsilon}{1+q_v\times(C_{pv}/C_{pd})}$$

から求める. ここで、p_0 はエクスナー関数の基準気圧である。R_d, C_{pd}, R_v, C_{pv} はそれぞれ、乾燥大気における気体定数と定圧比熱, 水蒸気における気体定数と定圧比熱である。

備考

特になし.

3.16.27 thetae_Bolton

機能

Bolton (1980) の式から相当温位を計算する.

書式

```
result=thetae_Bolton( T, qv, pres )
```

引数

T	<R>	in	大気の温度 [K].
qv	<R>	in	大気の水蒸気混合比 [kg/kg].
pres	<R>	in	大気の気圧 [Pa].
戻り値	<R>	inout	相当温位 [K].

定義式

相当温位を θ_e とすると、

$$\theta_e = T\left(\frac{p_0}{p}\right)^{a(1-bq_{vs})} \exp\left\{q_{vs}\left(\frac{c}{T_{\mathrm{LCL}}}\right)(1+dq_{vs})\right\}, \quad a=0.2854,\ b=0.28,\ c=3376,\ d=0.81$$

ここで, $T, p_0, p, T_{\mathrm{LCL}}, q_{vs}$ はそれぞれ気温、1000 hPa、気圧、LCL 温度、飽和混合比である。

備考

特になし.

3.16.28 thetaes_Bolton

機能

Bolton (1980) の式から飽和相当温位を計算する.

書式

```
result=thetaes_Bolton( T, pres )
```

引数

T	<R>	in	大気の温度 [K].
pres	<R>	in	大気の気圧 [Pa].
戻り値	<R>	inout	相当温位 [K].

定義式

相当温位を θ_{es} とすると、

$$\theta_{es} = T\left(\frac{p_0}{p}\right)^{a(1-bq_{vs})} \exp\left\{q_{vs}\left(\frac{c}{T}\right)(1+dq_{vs})\right\}, \quad a = 0.2854,\ b = 0.28,\ c = 3376,\ d = 0.81$$

ここで, T, p_0, p, q_{vs} はそれぞれ気温、1000 hPa、気圧、飽和混合比である。

備考

特になし.

3.17 Trajectory

速度場のデータを与えて、流線、流跡線の計算を行うルーチン集。

3.17.1 Backward_Traject_2d

機能

2 次元ベクトル場のデータからオフラインで後方流跡線解析を行う。

書式

```
call Backward_Traject_2d( dt, stime, step, ini_x, ini_y, t, x, y, u, v,
traj_x, traj_y, FTF, [opt], [udef] )
```

引数

dt	<R>	in	計算する時間間隔 [s].
stime	<R>	in	初期位置での時刻 [s].
step	<I>	in	計算を行うステップ数.
ini_x	<R(:)>	in	初期位置 x [m].
ini_y	<R(size(ini_x))>	in	初期位置 y [m].
t	<R(:)>	in	速度場データのある時刻 (後述).
x	<R(:)>	in	右手系第一座標 [m].
y	<R(:)>	in	右手系第二座標 [m].
u	<R(size(x),size(y),size(t))>	in	x 方向のベクトル (後述).
v	<R(size(x),size(y),size(t))>	in	y 方向のベクトル (後述).
traj_x	<R(step,size(ini_x))>	inout	流跡線の x 位置座標 [m].
traj_y	<R(step,size(ini_x))>	inout	流跡線の y 位置座標 [m].
FTF	<L(size(ini_x))>	inout	計算領域外判定フラグ (後述).
opt	<C(*)>	in	時間積分のスキーム (後述). デフォルトは前方オイラースキーム.
undef	R	in	領域外処理用未定義値 (後述).

定義式

与えられる速度場のデータが時刻 t_1 から t_n までの n 個存在し、その時間間隔は非一様とする。このとき、時刻 t_s から dt ステップで nt 回だけ流跡線を計算するときの模式図が図**??**である。速度場データのない時刻では、その前後の速度場データから線形内挿を行って速度場を計算している。流跡線の定義式は、座標 (x, y) において、2 次元ベクトル $\boldsymbol{v}(x, y, t)$ が定義されており、この速度場に乗って移動するパーセルの軌跡 $\boldsymbol{x}(x, y, t)$ を

$$\delta\boldsymbol{x}(x, y, t) = \boldsymbol{v}(x, y, t)\delta t$$

という式の数値積分によって計算するものである。流線計算と異なり、速度場が時間の関数でもあるため、パーセルの位置 $\boldsymbol{x}(x, y, t)$ も時間の関数となっていることがわかる。もし、速度場が定常であれば、速度ベクトルは時間に依存しないので、パーセルの位置ベクトルも時間に依存することはなくなる。このことから、定常な速度場において、流線と流跡線は一致することがわかる。

図**??**における各変数と引数の対応関係は以下のとおりである。

$$t_s \ : \ \texttt{stime}, \quad dt \ : \ \texttt{dt}, \quad nt \ : \ \texttt{step}, \quad n \ : \ \texttt{size(t)},$$

$$t_i \ : \ \texttt{t(i)}$$

備考

- オプション opt は流跡線計算の際に行う時間積分の積分スキームを指定するオプションであり、以下の設定が可能である。

  EU1 前方 1 次精度のオイラースキーム.

HO1 ホインスキーム (改良オイラースキーム).

ME1 修正オイラースキーム.

RK3 3 次ルンゲ・クッタキーム.

RK4 4 次ルンゲ・クッタキーム.

これらのスキームの詳細については、付録**??**参照。

- 流跡線が x, y で指定される領域外に到達した場合、それ以降の時間ステップについては計算を放棄する。そして、放棄された時刻に相当する traj_x, traj_y の要素には undef で定義された値が返される。デフォルト設定では、0.0 が返されるようになっている。
- size(ini_x) を 2 以上にすればその各の初期値に対応した流跡線が計算される.
- FTF は流跡線解析時に, 計算領域外に流されたときの判定フラグ. .true. が領域外に出たことを, .false. は計算領域内に存在することを示す. .true. がついているパーセルについては, その出た時刻ステップ以降が 0.0 もしくは undef の値が入っている.

3.17.2 Backward_Traject_3d

機能

3 次元ベクトル場のデータからオフラインで後方流跡線解析を行う。

書式

```
call Backward_Traject_3d( dt, stime, step, ini_x, ini_y, ini_z,
t, x, y, z, u, v, w, traj_x, traj_y, traj_z, [opt], [udef] )
```

引数

dt	<R>	in	計算する時間間隔 [s].
stime	<R>	in	初期位置での時刻 [s].
step	<I>	in	計算を行うステップ数.
ini_x	<R(:)>	in	初期位置 x [m].
ini_y	<R(size(ini_x))>	in	初期位置 y [m].
ini_z	<R(size(ini_x))>	in	初期位置 z [m].
t	<R(:)>	in	速度場データのある時刻 (後述).
x	<R(:)>	in	右手系第一座標 [m].
y	<R(:)>	in	右手系第二座標 [m].
z	<R(:)>	in	右手系第三座標 [m].
u	<R(size(x),size(y), size(z), size(t))>	in	x 方向のベクトル (後述).
v	<R(size(x),size(y), size(z), size(t))>	in	y 方向のベクトル (後述).
w	<R(size(x),size(y), size(z), size(t))>	in	z 方向のベクトル (後述).
traj_x	<R(step,size(ini_x))>	inout	流跡線の x 位置座標 [m].
traj_y	<R(step,size(ini_x))>	inout	流跡線の y 位置座標 [m].
traj_z	<R(step,size(ini_x))>	inout	流跡線の z 位置座標 [m].
FTF	<L(size(ini_x))>	inout	計算領域外判定フラグ (後述).
opt	<C(*)>	in	時間積分のスキーム (後述). デフォルトは前方オイラースキーム.
undef	R	in	領域外処理用未定義値 (後述).

定義式

与えられる速度場のデータが時刻 t_1 から t_n までの n 個存在し、その時間間隔は非一様とする。このとき、時刻 t_s から dt ステップで nt 回だけ流跡線を計算するときの模式図が図**??**である。速度場データのない時刻では、その前後の速度場データから線形内挿を行って速度場を計算している。流跡線の定義式は、座標 (x, y, z) において、3 次元ベクトル $\boldsymbol{v}(x, y, z, t)$ が定義されており、この速度場に乗って移動するパーセルの軌跡 $\boldsymbol{x}(x, y, z, t)$ を

$$\delta\boldsymbol{x}(x, y, z, t) = \boldsymbol{v}(x, y, z, t)\delta t$$

という式の数値積分によって計算するものである。流線計算と異なり、速度場が時間の関数でもあるため、パーセルの位置 $\boldsymbol{x}(x, y, z, t)$ も時間の関数となっていることがわかる。もし、速度場が定常であれば、速度ベクトルは時間に依存しないので、パーセルの位置ベクトルも時間に依存することはなくなる。このことから、定常な速度場において、流線と流跡線は一致することがわかる。

図**??**における各変数と引数の対応関係は以下のとおりである。

$$t_s : \texttt{stime}, \quad dt : \texttt{dt}, \quad nt : \texttt{step}, \quad n : \texttt{size(t)},$$

t_i : t(i)

備考

- オプション opt は流跡線計算の際に行う時間積分の積分スキームを指定するオプションであり、以下の設定が可能である。

  EU1 前方 1 次精度のオイラースキーム.

  HO1 ホインスキーム (改良オイラースキーム).

  ME1 修正オイラースキーム.

  RK3 3 次ルンゲ・クッタキーム.

  RK4 4 次ルンゲ・クッタキーム.

  これらのスキームの詳細については、付録**??**参照。
- 流跡線が x，y，z で指定される領域外に到達した場合、それ以降の時間ステップについては計算を放棄する。そして、放棄された時刻に相当する traj_x，traj_y，traj_z の要素には undef で定義された値が返される。デフォルト設定では、0.0 が返されるようになっている。
- size(ini_x) を 2 以上にすればその各の初期値に対応した流跡線が計算される.
- FTF は流跡線解析時に, 計算領域外に流されたときの判定フラグ. .true. が領域外に出たことを, .false. は計算領域内に存在することを示す. .true. がついているパーセルについては, その出た時刻ステップ以降が 0.0 もしくは undef の値が入っている.

3.17.3 Forward_Traject_2d

機能

2 次元ベクトル場のデータからオフラインで前方流跡線解析を行う。

書式

```
call Forward_Traject_2d( dt, stime, step, ini_x, ini_y, t, x, y, u, v,
traj_x, traj_y, FTF, [opt], [udef] )
```

引数

dt	<R>	in	計算する時間間隔 [s].
stime	<R>	in	初期位置での時刻 [s].
step	<I>	in	計算を行うステップ数.
ini_x	<R(:)>	in	初期位置 x [m].
ini_y	<R(size(ini_x))>	in	初期位置 y [m].
t	<R(:)>	in	速度場データのある時刻 (後述).
x	<R(:)>	in	右手系第一座標 [m].
y	<R(:)>	in	右手系第二座標 [m].
u	<R(size(x),size(y),size(t))>	in	x 方向のベクトル (後述).
v	<R(size(x),size(y),size(t))>	in	y 方向のベクトル (後述).
traj_x	<R(step,size(ini_x))>	inout	流跡線の x 位置座標 [m].
traj_y	<R(step,size(ini_x))>	inout	流跡線の y 位置座標 [m].
FTF	<L(size(ini_x))>	inout	計算領域外判定フラグ (後述).
opt	<C(*)>	in	時間積分のスキーム (後述). デフォルトは前方オイラースキーム.
undef	R	in	領域外処理用未定義値 (後述).

定義式

与えられる速度場のデータが時刻 t_1 から t_n までの n 個存在し、その時間間隔は非一様とする。このとき、時刻 t_s から dt ステップで nt 回だけ流跡線を計算するときの模式図が図**??**である。速度場データのない時刻では、その前後の速度場データから線形内挿を行って速度場を計算している。流跡線の定義式は、座標 (x, y) において、2 次元ベクトル $\boldsymbol{v}(x, y, t)$ が定義されており、この速度場に乗って移動するパーセルの軌跡 $\boldsymbol{x}(x, y, t)$ を

$$\delta\boldsymbol{x}(x, y, t) = \boldsymbol{v}(x, y, t)\delta t$$

という式の数値積分によって計算するものである。流線計算と異なり、速度場が時間の関数でもあるため、パーセルの位置 $\boldsymbol{x}(x, y, t)$ も時間の関数となっていることがわかる。もし、速度場が定常であれば、速度ベクトルは時間に依存しないので、パーセルの位置ベクトルも時間に依存することはなくなる。このことから、定常な速度場において、流線と流跡線は一致することがわかる。

図**??**における各変数と引数の対応関係は以下のとおりである。

$$t_s \ : \ \texttt{stime}, \quad dt \ : \ \texttt{dt}, \quad nt \ : \ \texttt{step}, \quad n \ : \ \texttt{size(t)},$$

$$t_i \ : \ \texttt{t(i)}$$

備考

- オプション opt は流跡線計算の際に行う時間積分の積分スキームを指定するオプションであり、以下の設定が可能である。

  EU1 前方 1 次精度のオイラースキーム.

HO1 ホインスキーム (改良オイラースキーム).

ME1 修正オイラースキーム.

RK3 3 次ルンゲ・クッタキーム.

RK4 4 次ルンゲ・クッタキーム.

これらのスキームの詳細については、付録**??**参照。

- 流跡線が x, y で指定される領域外に到達した場合、それ以降の時間ステップについては計算を放棄する。そして、放棄された時刻に相当する traj_x, traj_y の要素には undef で定義された値が返される。デフォルト設定では、0.0 が返されるようになっている。
- size(ini_x) を 2 以上にすればその各の初期値に対応した流跡線が計算される.
- FTF は流跡線解析時に, 計算領域外に流されたときの判定フラグ. .true. が領域外に出たことを, .false. は計算領域内に存在することを示す. .true. がついているパーセルについては, その出た時刻ステップ以降が 0.0 もしくは undef の値が入っている.

3.17.4 Forward_Traject_3d

機能

3 次元ベクトル場のデータからオフラインで前方流跡線解析を行う。

書式

```
call Forward_Traject_3d( dt, stime, step, ini_x, ini_y, ini_z,
t, x, y, z, u, v, w, traj_x, traj_y, traj_z, [opt], [udef] )
```

引数

dt	<R>	in	計算する時間間隔 [s].
stime	<R>	in	初期位置での時刻 [s].
step	<I>	in	計算を行うステップ数.
ini_x	<R(:)>	in	初期位置 x [m].
ini_y	<R(size(ini_x))>	in	初期位置 y [m].
ini_z	<R(size(ini_x))>	in	初期位置 z [m].
t	<R(:)>	in	速度場データのある時刻 (後述).
x	<R(:)>	in	右手系第一座標 [m].
y	<R(:)>	in	右手系第二座標 [m].
z	<R(:)>	in	右手系第三座標 [m].
u	<R(size(x),size(y), size(z), size(t))>	in	x 方向のベクトル (後述).
v	<R(size(x),size(y), size(z), size(t))>	in	y 方向のベクトル (後述).
w	<R(size(x),size(y), size(z), size(t))>	in	z 方向のベクトル (後述).
traj_x	<R(step,size(ini_x))>	inout	流跡線の x 位置座標 [m].
traj_y	<R(step,size(ini_x))>	inout	流跡線の y 位置座標 [m].
traj_z	<R(step,size(ini_x))>	inout	流跡線の z 位置座標 [m].
FTF	<L(size(ini_x))>	inout	計算領域外判定フラグ (後述).
opt	<C(*)>	in	時間積分のスキーム (後述). デフォルトは前方オイラースキーム.
undef	R	in	領域外処理用未定義値 (後述).

定義式

与えられる速度場のデータが時刻 t_1 から t_n までの n 個存在し、その時間間隔は非一様とする。このとき、時刻 t_s から dt ステップで nt 回だけ流跡線を計算するときの模式図が図**??**である。速度場データのない時刻では、その前後の速度場データから線形内挿を行って速度場を計算している。流跡線の定義式は、座標 (x, y, z) において、3 次元ベクトル $\boldsymbol{v}(x, y, z, t)$ が定義されており、この速度場に乗って移動するパーセルの軌跡 $\boldsymbol{x}(x, y, z, t)$ を

$$\delta\boldsymbol{x}(x, y, z, t) = \boldsymbol{v}(x, y, z, t)\delta t$$

という式の数値積分によって計算するものである。流線計算と異なり、速度場が時間の関数でもあるため、パーセルの位置 $\boldsymbol{x}(x, y, z, t)$ も時間の関数となっていることがわかる。もし、速度場が定常であれば、速度ベクトルは時間に依存しないので、パーセルの位置ベクトルも時間に依存することはなくなる。このことから、定常な速度場において、流線と流跡線は一致することがわかる。

図**??**における各変数と引数の対応関係は以下のとおりである。

$$t_s \ : \ \texttt{stime}, \quad dt \ : \ \texttt{dt}, \quad nt \ : \ \texttt{step}, \quad n \ : \ \texttt{size(t)},$$

t_i : t(i)

備考

- オプション opt は流跡線計算の際に行う時間積分の積分スキームを指定するオプションであり、以下の設定が可能である。

  EU1 前方 1 次精度のオイラースキーム.

  HO1 ホインスキーム (改良オイラースキーム).

  ME1 修正オイラースキーム.

  RK3 3 次ルンゲ・クッタキーム.

  RK4 4 次ルンゲ・クッタキーム.

  これらのスキームの詳細については、付録**??**参照。
- 流跡線が x, y, z で指定される領域外に到達した場合、それ以降の時間ステップについては計算を放棄する。そして、放棄された時刻に相当する traj_x, traj_y, traj_z の要素には undef で定義された値が返される。デフォルト設定では、0.0 が返されるようになっている。
- size(ini_x) を 2 以上にすればその各の初期値に対応した流跡線が計算される.
- FTF は流跡線解析時に, 計算領域外に流されたときの判定フラグ. .true. が領域外に出たことを, .false. は計算領域内に存在することを示す. .true. がついているパーセルについては, その出た時刻ステップ以降が 0.0 もしくは undef の値が入っている.

3.17.5 Stream_Line_2d

機能

2 次元ベクトル場のデータから流線を計算する。

書式

```
call Stream_Line_2d( dt, step, ini_x, ini_y, x, y, u, v, traj_x, traj_y,
[opt], [udef] )
```

引数

dt	<R>	in	計算する時間間隔 [s].
step	<I>	in	計算を行うステップ数.
ini_x	<R>	in	初期位置 x [m].
ini_y	<R>	in	初期位置 y [m].
x	<R(:)>	in	右手系第一座標 [m].
y	<R(:)>	in	右手系第二座標 [m].
u	<R(size(x),size(y))>	in	x 方向のベクトル.
v	<R(size(x),size(y))>	in	y 方向のベクトル.
traj_x	<R(step)>	inout	流線の x 位置座標 [m].
traj_y	<R(step)>	inout	流線の y 位置座標 [m].
opt	<C(*)>	in	時間積分のスキーム (後述).
			デフォルトは前方オイラースキーム.
undef	R	in	領域外処理用未定義値 (後述).

定義式

流線の定義式は、座標 (x, y) において、2 次元ベクトル $\boldsymbol{v}(x, y)$ が定義されており、この速度場に乗って移動するパーセルの軌跡 $\boldsymbol{x}(x, y)$ を

$$\delta \boldsymbol{x}(x, y) = \boldsymbol{v}(x, y)\delta t$$

という式の数値積分によって計算するものである。

備考

- オプション opt は流線計算の際に行う時間積分の積分スキームを指定するオプションであり、以下の設定が可能である。

  EU1 前方 1 次精度のオイラースキーム.

  HO1 ホインスキーム (改良オイラースキーム).

  ME1 修正オイラースキーム.

  RK3 3 次ルンゲ・クッタキーム.

  RK4 4 次ルンゲ・クッタキーム.

  これらのスキームの詳細については、付録**??**参照。
- 流線が x, y で指定される領域外に到達した場合、それ以降の時間ステップについては計算を放棄する。そして、放棄された時刻に相当する traj_x, traj_y の要素には undef で定義された値が返される。デフォルト設定では、0.0 が返されるようになっている。

3.17.6 Stream_Line_3d

機能

3 次元ベクトル場のデータから流線を計算する。

書式

```
call Stream_Line_3d( dt, step, ini_x, ini_y, ini_z,
x, y, z, u, v, w, traj_x, traj_y, traj_z, [opt], [udef] )
```

引数

dt	<R>	in	計算する時間間隔 [s].
step	<I>	in	計算を行うステップ数.
ini_x	<R>	in	初期位置 x [m].
ini_y	<R>	in	初期位置 y [m].
ini_z	<R>	in	初期位置 z [m].
x	<R(:)>	in	右手系第一座標 [m].
y	<R(:)>	in	右手系第二座標 [m].
z	<R(:)>	in	右手系第三座標 [m].
u	<R(size(x),size(y), size(z))>	in	x 方向のベクトル (後述).
v	<R(size(x),size(y), size(z))>	in	y 方向のベクトル (後述).
w	<R(size(x),size(y), size(z))>	in	z 方向のベクトル (後述).
traj_x	<R(step)>	inout	流線の x 位置座標 [m].
traj_y	<R(step)>	inout	流線の y 位置座標 [m].
traj_z	<R(step)>	inout	流線の z 位置座標 [m].
opt	<C(*)>	in	時間積分のスキーム (後述). デフォルトは前方オイラースキーム.
undef	R	in	領域外処理用未定義値 (後述).

定義式

流線の定義式は、座標 (x, y, z) において、3 次元ベクトル $\boldsymbol{v}(x, y, z)$ が定義されており、この速度場に乗って移動するパーセルの軌跡 $\boldsymbol{x}(x, y, z)$ を

$$\delta\boldsymbol{x}(x, y, z) = \boldsymbol{v}(x, y, z)\delta t$$

という式の数値積分によって計算するものである。

備考

- オプション opt は流線計算の際に行う時間積分の積分スキームを指定するオプションであり、以下の設定が可能である。

  EU1 前方 1 次精度のオイラースキーム.

  HO1 ホインスキーム (改良オイラースキーム).

  ME1 修正オイラースキーム.

RK3 3 次ルンゲ・クッタキーム.

RK4 4 次ルンゲ・クッタキーム.

これらのスキームの詳細については、付録**??**参照。

- 流線が x, y, z で指定される領域外に到達した場合、それ以降の時間ステップについては計算を放棄する。そして、放棄された時刻に相当する traj_x, traj_y, traj_z の要素には undef で定義された値が返される。デフォルト設定では、0.0 が返されるようになっている。

3.18 typhoon_analy

台風解析用のスペシャルルーチン集. 主に、デカルト座標系におけるデータを円筒座標系に変換し、接線平均等の処理を行うものの集合体。

3.18.1 grad_wind_pres

機能

傾度風平衡場を満たす気圧場を計算する

書式

```
call grad_wind_pres( r, coril, v, rho, r_ref, p_ref, pres )
```

引数

r	<R(:)>	in	動径座標 [m].
coril	<R(size(r))>	in	コリオリパラメータ [1/s].
v	<R(size(r))>	in	軸対称流 [m/s].
rho	<R>	in	サウンディングの密度 [kg/m^3].
r_ref	<R>	in	積分定数の位置座標 [m].
p_ref	<R>	in	r_ref での気圧 [Pa].
pres	<R(size(r))>	inout	平衡場の気圧 [Pa].

定義式

傾度風平衡：

$$\frac{V_g^2}{r} + fV_g = \frac{1}{\rho}\frac{\partial p}{\partial r}$$

を満たすように、気圧場の水平分布を計算する。ここで、V_g は軸対称風, ρ は大気密度、p は気圧、r, z は動径、鉛直座標、f はコリオリパラメータである。傾度風平衡

の式から、軸対称風を元に r 方向に数値積分：

$$\int_{r_{ref}}^{r} \frac{\partial p}{\partial r} dr = \int_{r_0}^{r} \rho \left[\frac{V_g^2}{r} + f V_g \right] dr$$

を行う。この式を計算すると、

$$p(r) = p_{ref} + \int_{r_{ref}}^{r} \rho \left[\frac{V_g^2}{r} + f V_g \right] dr$$

となるので、与えられた軸対称風から各半径における気圧の値が求められる。

備考

なし.

3.18.2 hydro_grad_eqb

機能

サウンディングと軸対称流から静力学・傾度風平衡場の計算を行う.

書式

```
call hydro_grad_eqb( r, z, coril, v, pres_s, rho_s, pres, rho, [error]
)
```

引数

r	<R(:)>	in	動径座標 [m].
z	<R(:)>	in	鉛直座標 [m].
coril	<R(size(r),size(z))>	in	コリオリパラメータ [1/s].
v	<R(size(r),size(z))>	in	軸対称流 [m/s].
pres_s	<R(size(z))>	in	サウンディングの気圧 [Pa].
rho_s	<R(size(z))>	in	サウンディングの密度 [kg/m^3].
pres	<R(size(r),size(z))>	inout	平衡場の気圧 [Pa].
rho	<R(size(r),size(z))>	inout	平衡場の密度 [kg/m^3].
error	<R>	in	イタレーションの収束条件. デフォルト $= 10^{-5}$.

定義式

各高度において、静力学平衡：

$$\frac{\partial p}{\partial z} = -g\rho$$

および、傾度風平衡：

$$\frac{V_g^2}{r} + f V_g = \frac{1}{\rho} \frac{\partial p}{\partial r}$$

を満たすように、1 次元のサウンディングデータから気圧場、密度場の鉛直構造を修正する。ここで、V_g は軸対称風, ρ は大気密度、p は気圧、r, z は動径、鉛直座標、f はコリオリパラメータである。これらの構造が求められれば、理想気体の状態方程式から、温度場も自然に得られる。また、これらの平衡状態が実現しているということは、傾度風平衡版の温度風平衡も同時に成り立つことがわかる[*9]。修正の方法は、反復法を用いた収束による修正である。以下にその流れを示す。初期には密度、気圧場を各高度について水平一様に与えておく。また、修正を行う際の基準状態は熱帯低気圧の中心から十分離れた `r(nr), nr=size(r)` の点とする。

1. 傾度風平衡の式から、軸対称風を元に r 方向に数値積分：

$$\int_{r_0}^{r} \frac{\partial p}{\partial r} dr = \int_{r_0}^{r} \rho \left[\frac{V_g^2}{r} + f V_g \right] dr$$

   を行う。ここで、r_0 は上で定義した動径座標の外縁である。この式を計算すると、

$$p(r) = p(r_0) + \int_{r_0}^{r} \rho \left[\frac{V_g^2}{r} + f V_g \right] dr$$

   となるので、与えられた軸対称風から各高度の各半径における気圧の値が求められる。

2. 得られた気圧 $p(r, z)$ の値を用いて、各高度、各半径における密度を静力学平衡の関係から計算する。

以降は、この繰り返しで、1 ステップ前との変化量が `error` 以下なら反復を終了するように設定している。

備考

なし.

3.18.3 tangent_mean_anom_scal

機能

任意のスカラー量をデカルト座標系から円筒座標系に変換し, 接線方向に平均したアノマリーを計算する.

書式

```
call tangent_mean_anom_scal( x, y, xc, yc, u, r, theta, v,
[undef], [undefg], [undefgc] )
```

引数

[*9] 傾度風平衡版の温度風の関係式は Emanuel (1986) 等を参照。

x	<R(:)>	in	デカルト第一座標.
y	<R(:)>	in	デカルト第二座標.
xc	<R>	in	デカルト x 系での円筒座標原点 (後述).
yc	<R>	in	デカルト y 系での円筒座標原点 (後述).
u	<R(size(x),size(y))>	in	デカルト系でのアノマリを とるスカラー.
r	<R(:)>	in	円筒座標系の動径座標.
theta	<R(:)>	in	円筒座標系の接線座標 [rad].
v	<R(size(r),size(theta))>	inout	アノマリの値.
[undef]	<R>	in	接線平均可能半径外の値（後述）.
[undefg]	<R>	in	接線平均内での欠損値.
[undefgc]	<C(3)>	in	undefg が設定 されたときの処理（後述）.

定義式

2 次元デカルト座標系 (x, y) における点 (x_c, y_c) を原点とする円筒座標系 (r, θ) に、スカラーデータを変換し、θ 方向に平均化する。座標系の変換は双線形内挿を行う。

デカルト系 (x_i, y_j) で定義されたあるスカラー量 $u_{i,j}$ を円筒座標系 (r_m, θ_n) で定義されるスカラー量 $v_{m,n}$ に変換する式は

$$v_{m,n} = u_{i,j} + \frac{u_{i+1,j} - u_{i,j}}{x_{i+1} - x_i}(x_m - x_i) + \frac{u_{i,j+1} - u_{i,j}}{y_{j+1} - y_j}(y_n - y_j)$$
$$[u_{i+1,j+1} - u_{i,j+1} - u_{i+1,j} + u_{i,j}] \frac{(x_m - x_i)(y_n - y_j)}{(x_{i+1} - x_i)(y_{j+1} - y_j)}$$

である[*10]。このとき、引数は以下の対応をする。

$$x_i \ : \ \mathtt{x}, \quad y_j \ : \ \mathtt{y}, \quad u_{i,j} \ : \ \mathtt{u}, \quad v_{m,n} \ : \ \mathtt{v},$$

$$r_m \ : \ \mathtt{r}, \quad \theta_n \ : \ \mathtt{theta}$$

備考

- 座標間の概念図は図 3.1 参照.
- オプション引数 undef は接線平均可能半径外に存在する接線平均量の配列に代入される未定義値である. ここで, 接線平均可能半径とは, 円筒座標系で接線平均を行う半径の最大値 r(size(r)) のことである. もし, この半径が計算領域 x,y の外に出た場合, はみ出している部分は平均ができない. このはみ出し部分の, 平均が行えない領域については undef 値が返されるようになっている. もし, undef が設定されていない場合はゼロが代入されて返される.
- オプション引数 undefgc は接線平均領域内において, 欠損値 undefg が設定されている場合, どのように平均を行うかを判定する. このオプションは undefg

[*10] この式は??参照。

が設定されていなければ機能しない. undefg は接線平均領域内において, デカルト座標系で欠損値が存在する場合, デカルトから円筒への内挿の際にデカルト系の参照点が undefg であれば円筒系の内挿点は undefg の値が入れられる. この状況において, undefgc の値は

inc 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, その点での値は平均操作に入れずに, 有効値が入っている点のみで平均操作を行う. これに対応して, 内挿点が欠損の場合はアノマリも欠損となる.

err 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, 平均値は undefg となる. アノマリ計算においては, この平均値を参照するアノマリ計算点のすべてで欠損となる.

undefg が設定されており, undefgc が指定されていないデフォルトでは, inc として計算される.

3.18.4 tangent_mean_anom_scal_Cart

機能

任意のスカラー量を接線方向に平均したアノマリーを計算し, デカルト座標系に戻す.

書式

```
call tangent_mean_anom_scal( x, y, xc, yc, scal, r, theta, scal_anom,
[undef], [undefg], [undefgc] )
```

引数

x	<R(:)>	in	デカルト第一座標.
y	<R(:)>	in	デカルト第二座標.
xc	<R>	in	デカルト x 系での円筒座標原点 (後述).
yc	<R>	in	デカルト y 系での円筒座標原点 (後述).
scal	<R(size(x),size(y))>	in	デカルト系でのアノマリを とるスカラー.
r	<R(:)>	in	円筒座標系の動径座標.
theta	<R(:)>	in	円筒座標系の接線座標 [rad].
scal_anom	<R(size(x),size(y))>	inout	デカルト系でのアノマリの値.
[undef]	<R>	in	接線平均可能半径外の値（後述）.
[undefg]	<R>	in	接線平均内での欠損値.
[undefgc]	<C(3)>	in	undefg が設定 されたときの処理（後述）.

定義式

tangent_mean_scal で計算した接線平均値を用いて, 元のデカルト系での値のアノマリを計算する. デカルト座標系の各点での円筒座標系中心からの距離を元に, その

距離の近接両端での接線平均値を内挿処理し, その内挿値をそのデカルト座標系での接線平均値とする.

備考

- オプション引数 undef は接線平均可能半径外に存在する接線平均量の配列に代入される未定義値である. ここで, 接線平均可能半径とは, 円筒座標系で接線平均を行う半径の最大値 r(size(r)) のことである. もし, この半径が計算領域 x,y の外に出た場合, はみ出している部分は平均ができない. このはみ出し部分の, 平均が行えない領域については undef 値が返されるようになっている. もし, undef が設定されていない場合はゼロが代入されて返される.
- オプション引数 undefgc は接線平均領域内において, 欠損値 undefg が設定されている場合, どのように平均を行うかを判定する. このオプションは undefg が設定されていなければ機能しない. undefg は接線平均領域内において, デカルト座標系で欠損値が存在する場合, デカルトから円筒への内挿の際にデカルト系の参照点が undefg であれば円筒系の内挿点は undefg の値が入れられる. この状況において, undefgc の値は

  inc 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, その点での値は平均操作に入れずに, 有効値が入っている点のみで平均操作を行う. これに対応して, 内挿点が欠損の場合はアノマリも欠損となる.

  err 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, 平均値は undefg となる. アノマリ計算においては, この平均値を参照するアノマリ計算点のすべてで欠損となる.

  undefg が設定されており, undefgc が指定されていないデフォルトでは, inc として計算される.

3.18.5 tangent_mean_anom_vec

機能

デカルト座標系で定義された任意の 2 次元ベクトルをデカルト座標系から円筒座標系に変換し, 接線方向に平均し, そのアノマリをとる. このとき, 判別フラグを用いることで, 動径方向成分と接線方向成分に分割できる.

書式

```
call tangent_mean_anom_vec( charc, x, y, xc, yc, u1, u2, r, theta, v,

[undef], [undefg], [undefgc] )
```

引数

charc	<C(6)>	in	接線成分か動径成分の どちらに平均するか. "scalar" = 動径外向き成分. "vector" = 接線反時計成分.
x	<R(:)>	in	デカルト第一座標.
y	<R(:)>	in	デカルト第二座標.
xc	<R>	in	デカルト x 系での円筒座標原点 (後述).
yc	<R>	in	デカルト y 系での円筒座標原点 (後述).
u1	<R(size(x),size(y))>	in	デカルト系での平均するベクトル. x 成分.
u2	<R(size(x),size(y))>	in	デカルト系での平均するベクトル. y 成分.
r	<R(:)>	in	円筒座標系の動径座標.
theta	<R(:)>	in	円筒座標系の接線座標 [rad].
v	<R(size(r),size(theta))>	inout	アノマリの値.
[undef]	<R>	in	接線平均可能半径外の値（後述）.
[undefg]	<R>	in	接線平均内での欠損値.
[undefgc]	<C(3)>	in	undefg が設定 されたときの処理（後述）.

定義式

行う処理は tangent_mean_scal とほとんど同じ. 最初にフラグ charc でデカルト系ベクトルを円筒系ベクトルに変換し, そのうちフラグ成分のみをスカラーとして計算すれば, 上の処理が実現される。ここで、任意のデカルト系におけるベクトルを $\boldsymbol{u} = (u_1, u_2)$ とし、円筒座標系の位置ベクトルを $\boldsymbol{r} = r\boldsymbol{e}_r$ とする。ここで、$\boldsymbol{e}_r$ は動径方向外向きの基底ベクトルである。このとき、この位置ベクトルはデカルト座標系で表現すると、$\boldsymbol{r} = (x, y)$ で表現できるので、デカルト座標系同士の内積は

$$\boldsymbol{r} \cdot \boldsymbol{u} = |\boldsymbol{r}||\boldsymbol{u}_r|$$

という量を表現することになる。ここで、$\boldsymbol{u}_r$ はデカルト系でのベクトルの動径成分である。よって、これを距離で割れば動径方向外向きのベクトルが得られることになる。接線方向の成分もこれと同様の考え方で、位置ベクトルとデカルト系ベクトルの外積から反時計回りを正とする円筒座標系の接線成分を抽出することができる。2 次元デカルト座標系 (x, y) における点 (x_c, y_c) を原点とする円筒座標系 (r, θ) に、スカラーデータを変換し、θ 方向に平均化する。座標系の変換は双線形内挿を行う。

備考

- オプション引数 undef は接線平均可能半径外に存在する接線平均量の配列に代入される未定義値である. ここで, 接線平均可能半径とは, 円筒座標系で接線平均を行う半径の最大値 r(size(r)) のことである. もし, この半径が計算領

域 x,y の外に出た場合, はみ出している部分は平均ができない. このはみ出し部分の, 平均が行えない領域については undef 値が返されるようになっている. もし, undef が設定されていない場合はゼロが代入されて返される.

- オプション引数 undefgc は接線平均領域内において, 欠損値 undefg が設定されている場合, どのように平均を行うかを判定する. このオプションは undefg が設定されていなければ機能しない. undefg は接線平均領域内において, デカルト座標系で欠損値が存在する場合, デカルトから円筒への内挿の際にデカルト系の参照点が undefg であれば円筒系の内挿点は undefg の値が入れられる. この状況において, undefgc の値は

  inc 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, その点での値は平均操作に入れずに, 有効値が入っている点のみで平均操作を行う. これに対応して, 内挿点が欠損の場合はアノマリも欠損となる.

  err 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, 平均値は undefg となる. アノマリ計算においては, この平均値を参照するアノマリ計算点のすべてで欠損となる.

  undefg が設定されており, undefgc が指定されていないデフォルトでは, inc として計算される.

3.18.6 tangent_mean_scal

機能

任意のスカラー量をデカルト座標系から円筒座標系に変換し, 接線方向に平均する.

書式

```
call tangent_mean_scal( x, y, xc, yc, u, r, theta, v,
[undef], [undefg], [undefgc] )
```

引数

x	<R(:)>	in	デカルト第一座標.
y	<R(:)>	in	デカルト第二座標.
xc	<R>	in	デカルト x 系での円筒座標原点 (後述).
yc	<R>	in	デカルト y 系での円筒座標原点 (後述).
u	<R(size(x),size(y))>	in	デカルト系での平均するスカラー.
r	<R(:)>	in	円筒座標系の動径座標.
theta	<R(:)>	in	円筒座標系の接線座標 [rad].
v	<R(size(r))>	inout	接線平均したスカラー.
[undef]	<R>	in	接線平均可能半径外の値（後述）.
[undefg]	<R>	in	接線平均内での欠損値.
[undefgc]	<C(3)>	in	undefg が設定されたときの処理（後述）.

定義式

2 次元デカルト座標系 (x, y) における点 (x_c, y_c) を原点とする円筒座標系 (r, θ) に、スカラーデータを変換し、θ 方向に平均化する。座標系の変換は双線形内挿を行う。

デカルト系 (x_i, y_j) で定義されたあるスカラー量 $u_{i,j}$ を円筒座標系 (r_m, θ_n) で定義されるスカラー量 $v_{m,n}$ に変換する式は

$$v_{m,n} = u_{i,j} + \frac{u_{i+1,j} - u_{i,j}}{x_{i+1} - x_i}(x_m - x_i) + \frac{u_{i,j+1} - u_{i,j}}{y_{j+1} - y_j}(y_n - y_j)$$
$$[u_{i+1,j+1} - u_{i,j+1} - u_{i+1,j} + u_{i,j}] \frac{(x_m - x_i)(y_n - y_j)}{(x_{i+1} - x_i)(y_{j+1} - y_j)}$$

である[*11]。このとき、引数は以下の対応をする。

$$x_i \ : \ \mathtt{x}, \quad y_j \ : \ \mathtt{y}, \quad u_{i,j} \ : \ \mathtt{u}, \quad v_{m,n} \ : \ \mathtt{v},$$

$$r_m \ : \ \mathtt{r}, \quad \theta_n \ : \ \mathtt{theta}$$

備考

- 座標間の概念図は図 3.1 参照.
- オプション引数 undef は接線平均可能半径外に存在する接線平均量の配列に代入される未定義値である. ここで, 接線平均可能半径とは, 円筒座標系で接線平均を行う半径の最大値 r(size(r)) のことである. もし, この半径が計算領域 x,y の外に出た場合, はみ出している部分は平均ができない. このはみ出し部分の, 平均が行えない領域については undef 値が返されるようになっている. もし, undef が設定されていない場合はゼロが代入されて返される.
- オプション引数 undefgc は接線平均領域内において, 欠損値 undefg が設定されている場合, どのように平均を行うかを判定する. このオプションは undefg が設定されていなければ機能しない. undefg は接線平均領域内において, デカ

[*11] この式は??参照。

ルト座標系で欠損値が存在する場合, デカルトから円筒への内挿の際にデカルト系の参照点が undefg であれば円筒系の内挿点は undefg の値が入れられる. この状況において, undefgc の値は

inc 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, その点での値は平均操作に入れずに, 有効値が入っている点のみで平均操作を行う. これに対応して, 内挿点が欠損の場合はアノマリも欠損となる.

err 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, 平均値は undefg となる. アノマリ計算においては, この平均値を参照するアノマリ計算点のすべてで欠損となる.

undefg が設定されており, undefgc が指定されていないデフォルトでは, inc として計算される.

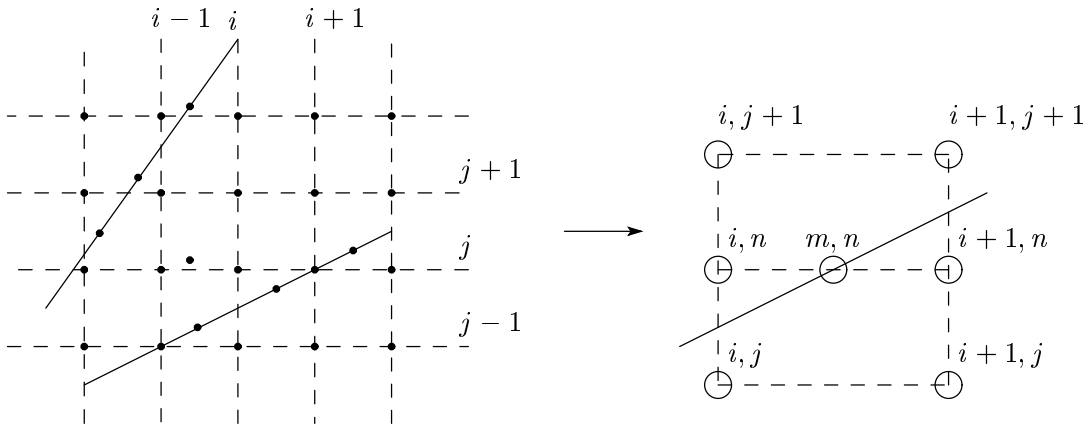


図 3.1: デカルト座標系から円筒座標系への双線形補間の概念. 円筒座標系で m, n 点を求めたい場合は, デカルト座標系で j 方向に内挿を計算し, その内挿点から i 方向に線形内挿する.

3.18.7 tangent_mean_vec

機能

デカルト座標系で定義された任意の 2 次元ベクトルをデカルト座標系から円筒座標系に変換し, 接線方向に平均する. このとき, 判別フラグを用いることで, 動径方向成分と接線方向成分に分割できる.

書式

```
call tangent_mean_vec( charc, x, y, xc, yc, u1, u2, r, theta, v,
[undef], [undefg], [undefgc] )
```

引数

charc	<C(6)>	in	接線成分か動径成分の どちらに平均するか. "scalar" = 動径外向き成分. "vector" = 接線反時計成分.
x	<R(:)>	in	デカルト第一座標.
y	<R(:)>	in	デカルト第二座標.
xc	<R>	in	デカルト x 系での円筒座標原点 (後述).
yc	<R>	in	デカルト y 系での円筒座標原点 (後述).
u1	<R(size(x),size(y))>	in	デカルト系での平均するベクトル. x 成分.
u2	<R(size(x),size(y))>	in	デカルト系での平均するベクトル. y 成分.
r	<R(:)>	in	円筒座標系の動径座標.
theta	<R(:)>	in	円筒座標系の接線座標 [rad].
v	<R(size(r))>	inout	接線平均したスカラー.
[undef]	<R>	in	接線平均可能半径外の値（後述）.
[undefg]	<R>	in	接線平均内での欠損値.
[undefgc]	<C(3)>	in	undefg が設定 されたときの処理（後述）.

定義式

行う処理は tangent_mean_scal とほとんど同じ. 最初にフラグ charc でデカルト系ベクトルを円筒系ベクトルに変換し, そのうちフラグ成分のみをスカラーとして計算すれば, 上の処理が実現される。ここで、任意のデカルト系におけるベクトルを $\boldsymbol{u} = (u_1, u_2)$ とし、円筒座標系の位置ベクトルを $\boldsymbol{r} = r\boldsymbol{e}_r$ とする。ここで、$\boldsymbol{e}_r$ は動径方向外向きの基底ベクトルである。このとき、この位置ベクトルはデカルト座標系で表現すると、$\boldsymbol{r} = (x, y)$ で表現できるので、デカルト座標系同士の内積は

$$\boldsymbol{r} \cdot \boldsymbol{u} = |\boldsymbol{r}||\boldsymbol{u}_r|$$

という量を表現することになる。ここで、$\boldsymbol{u}_r$ はデカルト系でのベクトルの動径成分である。よって、これを距離で割れば動径方向外向きのベクトルが得られることになる。接線方向の成分もこれと同様の考え方で、位置ベクトルとデカルト系ベクトルの外積から反時計回りを正とする円筒座標系の接線成分を抽出することができる。

備考

- オプション引数 undef は接線平均可能半径外に存在する接線平均量の配列に代入される未定義値である. ここで, 接線平均可能半径とは, 円筒座標系で接線平均を行う半径の最大値 r(size(r)) のことである. もし, この半径が計算領

域 x,y の外に出た場合, はみ出している部分は平均ができない. このはみ出し部分の, 平均が行えない領域については undef 値が返されるようになっている. もし, undef が設定されていない場合はゼロが代入されて返される.

- オプション引数 undefgc は接線平均領域内において, 欠損値 undefg が設定されている場合, どのように平均を行うかを判定する. このオプションは undefg が設定されていなければ機能しない. undefg は接線平均領域内において, デカルト座標系で欠損値が存在する場合, デカルトから円筒への内挿の際にデカルト系の参照点が undefg であれば円筒系の内挿点は undefg の値が入れられる. この状況において, undefgc の値は

  inc 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, その点での値は平均操作に入れずに, 有効値が入っている点のみで平均操作を行う. これに対応して, 内挿点が欠損の場合はアノマリも欠損となる.

  err 接線平均を行う際, 円筒系内挿点の一部に undefg が設定されていれば, 平均値は undefg となる. アノマリ計算においては, この平均値を参照するアノマリ計算点のすべてで欠損となる.

  undefg が設定されており, undefgc が指定されていないデフォルトでは, inc として計算される.

第4章　サンプルプログラム

ここでは, 各ルーチンの使用例としていくつかのサンプルプログラムをあげておく.

4.1　サンプルプログラムのコンパイル方法

demo ディレクトリ以下に, 本ライブラリを用いたサンプルプログラムが複数同梱されている. 本サンプルプログラムの実行には STPK ライブラリが正常にインストールされており, NetCDF ライブラリ, gtool5 ライブラリがインストールされている必要がある. サンプルプログラムの計算結果は NetCDF データで出力されるため, 可視化には別途可視化ツールが必要である.

サンプルプログラムのコンパイルには, demo/Mkinclude を各自の環境に合わせて編集しなければならない. 変更する可能性がある箇所は以下のとおりである.

FC	STPK をインストールしたコンパイラ.
FFLAGS	インストール時のコンパイルオプション.
ISTDIR	STPK のインストールディレクトリ.
INSTNC	netcdf のインストールディレクトリ.
INSTGT	gtool5 のインストールディレクトリ.

コンパイルは demo ディレクトリで make コマンドを実行すればよい.

もし, DCLF90 がインストールされているなら, make draw と実行することで, 2 次元のグラフを描画することのできる実行ファイル draw が作成される. また, Ruby-DCL がインストールされていれば, 同じディレクトリに同梱されている dclplot を用いて, テキストフォーマットのデータを gnuplot のようにグラフ化することができる. この dclplot は Ruby スクリプトであり, ruby dclplot で実行すると, どのような引数を与えなければならないかというような usage 情報が出力されるので, それを参考にされたい.

draw コマンドの使い方については, 各サンプルプログラムのファイル名の前に draw_ とヘッダのついた拡張子 .nml のネームリストファイルがあるので, 例えば, poison を用いた結果得られたデータ poison.nc を可視化したい場合は, ./draw < draw_poison.nml で可視化することができる. この draw に対応したネームリストファイルの各変数は以下

のとおりである.

```
        &drawinput
        fig_type = 1  ! 図の種類 1: 等値線, シェード
                      ! 2: 等値線, シェード, ベクトル
                      ! -1: 横軸折れ線, マーカー
                      ! -2: 縦軸折れ線, マーカー
                      ! 折れ線を引くなら, cont_val に変数名,
                      ! マーカーをうつなら, shade_val に変数名,
                      ! 両方とも描画可能.
                      ! 横軸折れ線の場合, 横軸の描画範囲は xmax, xmin で
                      ! そのときの縦軸の描画範囲は cmin, cmax, smin, smax で設
定でき,
                      ! cmin, cmax は折れ線, smin, smax はマーカーに対応.
                      ! shade_val, cont_val それぞれ設定されているときは絶対値
が
                      ! 大きい方で設定される.
                      ! ymin, ymax で設定された範囲内に存在するすべての y 格子の
                      ! x 軸データを複数折れ線 (マーカー) で描画する.
                      ! もし 1 本のみ欲しければ, ymin = ymax と設定すること.
                      ! 縦軸折れ線の場合, 横軸折れ線の全く逆の変数設定をしておく
こと.
                      ! ex. 横軸折れ線で xmin, xmax の場合, 縦軸折れ線で
は ymin, ymax
                      !     がこの変数に対応している.
        nx = 100  ! 横軸の格子点数
        ny = 100  ! 縦軸の格子点数
        xmin = 0.0  ! 横軸の左端
        xmax = 1.0  ! 横軸の右端
        ymin = 0.0  ! 縦軸の下端
        ymax = 1.0  ! 縦軸の上端
        fname = 'poison.nc'  ! 読み込む netcdf ファイル
        txtname = ''  ! テキストカラムデータ
        cont_val = 'rho'  ! 等値線で描く変数
        shade_val = 'psi'  ! カラーシェードで描く変数
        vx_val = ''  ! ベクトル x 成分で描く変数
        vy_val = ''  ! ベクトル y 成分で描く変数
        cmin = -4.0  ! 等値線の最小値
        cmax = 0.0  ! 等値線の最大値
        smin = 0.0  ! カラーシェードの最小値
        smax = 5.0  ! カラーシェードの最大値
        x_axis = 'x'  ! 横軸の変数名
        y_axis = 'y'  ! 縦軸の変数名
        sfact = 1.0  ! シェードのファクター
        cfact = 1.0  ! 等値線のファクター
        vxfact = 1.0  ! 単位ベクトルの長さ x
        vyfact = 1.0  ! 単位ベクトルの長さ y
        title = ''
        /
```

他の draw 用ネームリストファイルも全て同じ変数となっている. 各ネームリストファイルは各サンプルを可視化する際, 何も変更しなくても描画することができる.

4.2 各プログラムの説明

4.2.1 advection

モジュール fftp のテスト用プログラム. 線形 1 次元の移流方程式について, スペクトル法を用いてその時間発展を計算する.

計算を行う方程式系は空間方向に 1 次元の物理量 $u(x,t)$ についての純移流方程式：

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

である. これを空間方向にはスペクトル展開し, 時間方向にはクランクニコルソンスキームを用いている. 空間格子数等はネームリストファイル advection.nml で設定可能である.

サンプル

```
1  program advection
2    use ffts  ! FFT ルーチン
3    use file_operate  ! 結果ファイル出力
4    use Math_Const
5    use gtool_history
6    implicit none
7    integer :: nx  ! x 方向の切断波数
8    real :: dt  ! 計算時間間隔
9    integer :: nt  ! 計算ステップ数
10   real :: xmin  ! x 座標左端
11   real :: dx  ! x 座標格子間隔
12   integer :: j, k
13   real, allocatable, dimension(:) :: x  ! x 座標
14   real, allocatable, dimension(:) :: y  ! スカラー変数
15   real, allocatable, dimension(:) :: time  ! 時間
16   real, allocatable, dimension(:,:) :: amp  ! 時系列データ
17   complex, allocatable, dimension(:) :: fm1  ! スペクトル用
18   complex, allocatable, dimension(:) :: fm2  ! スペクトル用
19   complex, allocatable, dimension(:) :: fm_tmp  ! ダンプ出力用
20   complex, parameter :: i=(0.0,1.0)
21   namelist /input /nx,dt,nt,xmin,dx
22   read(5,input)
23   if(mod(nx,2)/=0)then
24      write(*,*) "*** ERROR ***"
25      write(*,*) "nx must be even number. Stop."
26      stop
27   end if
```

```
28     allocate(x(0:2*nx))
29     allocate(y(0:2*nx))
30     allocate(time(nt+1))
31     allocate(amp(0:2*nx,nt+1))
32     allocate(fm1(0:2*nx))
33     allocate(fm2(0:2*nx))
34     allocate(fm_tmp(0:2*nx))
35     x=(/((xmin+dx*real(j)),j=0,2*nx)/)
36     time=(/((dt*real(j-1)),j=1,nt+1)/)
37     do j=0,2*nx
38        y(j)=sin(x(j))
39        amp(j,1)=y(j)
40  !      y(j)=exp(-(x(j)-0.5*real(2*nx+1)*dx)**2)
41        fm_tmp(j)=y(j)
42     end do
43     call ffttp_1d( 2*nx+1, fm_tmp, fm1, 'r', 'o' )
44  !-- gtool history (netcdf dump)
45     call HistoryCreate( &                              ! ヒストリー作成
46       & file='advection.nc', title='spectral advecting model', &
47       & source='Sample program of gtool_history/gtool5',   &
48       & institution='GFD_Dennou Club davis project',       &
49       & dims=('x','t'), dimsizes=(/2*nx+1,nt+1/),                &
50       & longnames=('X-coordinate','time        '),       &
51       & units=('m','s'),                                   &
52       & origin=0.0, interval=real(nt*dt) )
53     call HistoryPut( 'x', x )                            ! 次元変数出力
54     call HistoryAddVariable( &                           ! 変数定義
55       & varname='amp', dims=('x','t'), &
56       & longname='amplitude', units='1', xtype='float')
57  !-- 時間積分開始
58     do j=1,nt
59  !-- CR 法で時間積分
60        fm2(0)=fm1(0)
61        do k=1,nx
62           fm2(k)=((1.0-0.25*(dt*real(k))**2-i*dt*real(k))/  &
63     &            (1.0+0.25*(dt*real(k))**2))*fm1(k)
64           fm1(k)=fm2(k)  ! 次ステップへ渡す
65        end do
66        do k=1,nx
67           fm2(nx+k)=conjg(fm2(nx-k+1))  ! 計算不要な配列に結果を渡す
68        end do
69        call ffttp_1d( 2*nx+1, fm2, fm_tmp, 'i', 'o' )
70        do k=0,2*nx  ! 実数への変換
71           y(k)=real(fm_tmp(k))
72           amp(k,j+1)=y(k)
73        end do
```

```
74      end do
75      call HistoryPut( 'amp', amp )                    ! 変数出力
76      call HistoryClose
77    end program
```

実行方法・ネームリスト

```
./advection < advection.nml
```

この結果, advection.nc が出力される. また, ネームリストの変数は以下のとおりである.

```
&input
 nx=100  ! 空間格子点数
 dt=0.05  ! 時間間隔
 nt=100  ! 計算ステップ数
 xmin=0.0  ! 領域左端
 dx=0.0628  ! 空間格子間隔
/
```

計算結果

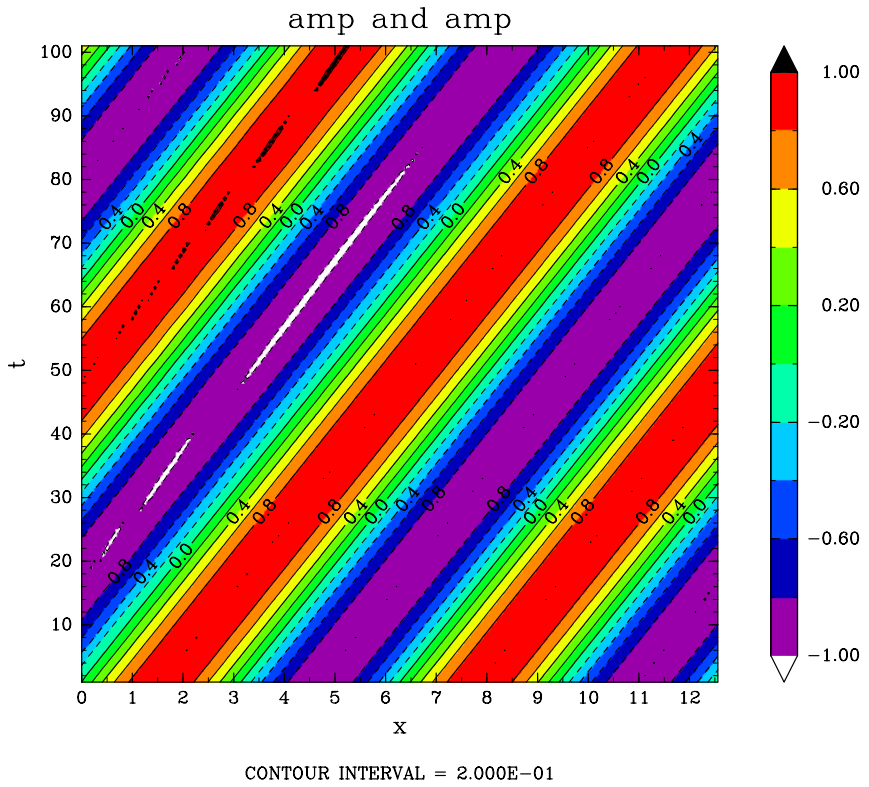


図 4.1: 波の時系列.

4.2.2 cov

モジュール Statistics のテスト用プログラム. ある与えられたデータファイル data.org を元に, 分布図を作成するプログラム. ただし, その分布図を元に回帰直線を引き, 相関係数を計算する.

サンプル

```
1  program cov
2    use Statistics
3    use file_operate
4    use max_min
5    use Basis
6    implicit none
7    real, allocatable, dimension(:) :: x, y, xa, ya
8    integer :: nx, i
9    real :: a, cc, slope, intercept
10   real :: xmax, xmin, ymax, ymin, xmean, ymean
11   integer :: xmim, ymim, xmam, ymam
12   character(10), allocatable, dimension(:,:) :: cval
13   nx=line_number_counter( 'data.org' )
14   allocate(cval(2,nx))
15   allocate(x(nx))
16   allocate(y(nx))
17   allocate(xa(nx))
18   allocate(ya(nx))
19   call read_file_text( 'data.org', 2, nx, cval )
20   do i=1,nx
21      x(i)=c2r_convert( cval(1,i) )
22      y(i)=c2r_convert( cval(2,i) )
23   end do
24   call Cor_Coe( x, y, cc )
25   call Reg_Line( x, y, slope, intercept )
26   call Mean_1d( x, xmean )
27   call Mean_1d( y, ymean )
28 !  call Anomaly_1d( nx, x, xa )
29 !  call Anomaly_1d( nx, y, ya )
30   do i=1,nx
31      xa(i)=x(i)/xmean
32      ya(i)=y(i)/ymean
33   end do
34   call max_val_1d(xa, xmam, xmax )
35   call max_val_1d(ya, ymam, ymax )
36   call min_val_1d(xa, xmim, xmin )
37   call min_val_1d(ya, ymim, ymin )
```

```
38     write(*,*) '************************************'
39     write(*,*) 'slope =', slope
40     write(*,*) 'intercept =', intercept
41     write(*,*) 'Cor_Coe =', cc
42     write(*,*) 'data number =', nx
43     write(*,*) '************************************'
44     deallocate(x)
45     deallocate(y)
46     deallocate(xa)
47     deallocate(ya)
48   end program
```

実行方法・実行結果

本プログラムは単体で実行するだけであり, その出力結果は以下のようになる.

```
************************************
slope = 4.4496446E-06
intercept = -3.6121133E-07
Cor_Coe = 3.6029361E-02
data number = 246
************************************
```

4.2.3 diffusion

モジュール fftttp のテスト用プログラム. 線形 1 次元の拡散方程式について, スペクトル法を用いてその時間発展を計算する.

計算を行う方程式系は空間方向に 1 次元の物理量 $u(x,t)$ についての純拡散方程式：

$$\frac{\partial u}{\partial t} = c\frac{\partial^2 u}{\partial x^2}$$

である. これを空間方向にはスペクトル展開し, 時間方向にはクランクニコルソンスキームを用いている. 空間格子数等はネームリストファイル diffusion.nml で設定可能である.

サンプル

```
1   program diffusion
2     use ffts  ! FFT ルーチン
3     use file_operate  ! 結果ファイル出力
4     use Math_Const
5     use gtool_history
6     implicit none
```

```
7     integer :: nx  ! x 方向の切断波数
8     real :: dt  ! 計算時間間隔
9     integer :: nt  ! 計算ステップ数
10    real :: xmin  ! x 座標左端
11    real :: dx  ! x 座標格子間隔
12    integer :: j, k
13    real, allocatable, dimension(:) :: x  ! x 座標
14    real, allocatable, dimension(:) :: y  ! スカラー変数
15    real, allocatable, dimension(:) :: time  ! 時間
16    real, allocatable, dimension(:,:) :: amp  ! 時系列
17    complex, allocatable, dimension(:) :: fm1  ! スペクトル用
18    complex, allocatable, dimension(:) :: fm2  ! スペクトル用
19    complex, allocatable, dimension(:) :: fm_tmp  ! ダンプ出力用
20    complex, parameter :: i=(0.0,1.0)
21    namelist /input /nx,dt,nt,xmin,dx
22    read(5,input)
23    if(mod(nx,2)/=0)then
24       write(*,*) "*** ERROR ***"
25       write(*,*) "nx must be even number. Stop."
26       stop
27    end if
28    allocate(x(0:2*nx))
29    allocate(y(0:2*nx))
30    allocate(time(nt+1))
31    allocate(amp(0:2*nx,nt+1))
32    allocate(fm1(0:2*nx))
33    allocate(fm2(0:2*nx))
34    allocate(fm_tmp(0:2*nx))
35    x=(/((xmin+dx*real(j)),j=0,2*nx)/)
36    time=(/((dt*real(j-1)),j=1,nt+1)/)
37    do j=0,2*nx
38 !     y(j)=sin(x(j))
39       y(j)=exp(-(x(j)-0.5*real(2*nx+1)*dx)**2)
40       amp(j,1)=y(j)
41       fm_tmp(j)=y(j)
42    end do
43    call ffttp_1d( 2*nx+1, fm_tmp, fm1, 'r', 'o' )
44 !-- gtool history (netcdf dump)
45    call HistoryCreate( &                              ! ヒストリー作成
46      & file='diffusion.nc', title='spectral advecting model', &
47      & source='Sample program of gtool_history/gtool5',   &
48      & institution='GFD_Dennou Club davis project',       &
49      & dims=('x','t'), dimsizes=(/2*nx+1,nt+1/),                &
50      & longnames=('X-coordinate','time        '),       &
51      & units=('m','s'),                                 &
```

```
52      & origin=0.0, interval=real(nt*dt) )
53    call HistoryPut( 'x', x )                              ! 次元変数出力
54    call HistoryAddVariable( &                             ! 変数定義
55      & varname='amp', dims=(/'x','t'/), &
56      & longname='amplitude', units='1', xtype='float')
57  !-- 時間積分開始
58    do j=1,nt
59  !-- CR 法で時間積分
60       fm2(0)=fm1(0)
61       do k=1,nx
62          fm2(k)=((1.0-0.5*dt*real(k)**2)/  &
63    &             (1.0+0.5*dt*real(k)**2))*fm1(k)
64          fm1(k)=fm2(k)  ! 次ステップへ渡す
65       end do
66       do k=1,nx
67          fm2(nx+k)=conjg(fm2(nx-k+1))  ! 計算不要な配列に結果を渡す
68       end do
69       call ffttp_1d( 2*nx+1, fm2, fm_tmp, 'i', 'o' )
70       do k=0,2*nx  ! 実数への変換
71          y(k)=real(fm_tmp(k))
72          amp(k,j+1)=y(k)
73       end do
74    end do
75    call HistoryPut( 'amp', amp )                           ! 変数出力
76    call HistoryClose
77  end program
```

実行方法・ネームリスト

```
./diffusion < diffusion.nml
```

この結果, diffusion.nc が出力される. また, ネームリストの変数は以下のとおりである.

```
&input
 nx=100  ! 空間格子点数
 dt=0.01  ! 時間間隔
 nt=100  ! 計算ステップ数
 xmin=0.0  ! 領域左端
 dx=0.0628  ! 空間格子間隔
/
```

計算結果

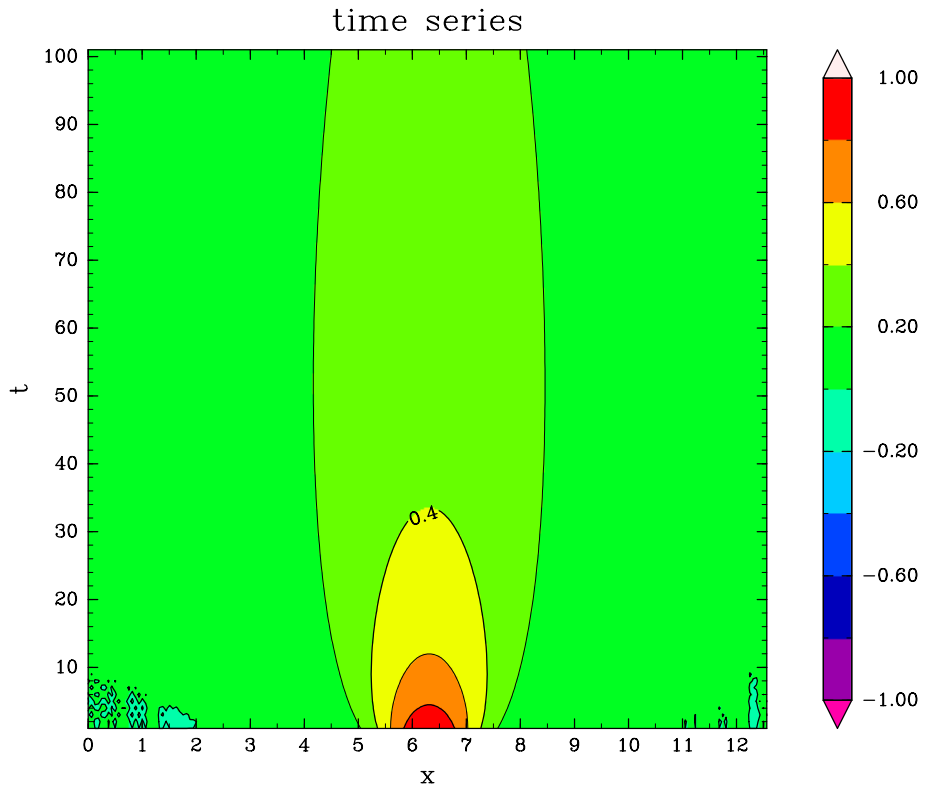


図 4.2: 波の時系列.

4.2.4 fft_test

FFT ルーチンのテストプログラム. 実数データを与えて実 FFT を計算し, 逆変換を行い元の実データに戻す. 戻したときに, 元データとどれだけ誤差があるかを表示する. また, 同じ動作を標準入力から指定する回数分繰り返す. FFT のパフォーマンステストも兼ねているので, 上とまったく同じ作業を離散フーリエ変換についても行い, 両者の実行時間を表示する. 用いるデータはデータ数の中央にピークをもつガウシアン分布である.

サンプル

```
1  program fft_test
2  ! fft ルーチンの実行速度を計算するルーチン
3  ! 実数データをスペクトル変換し，さらに逆変換して元に戻すことで
4  ! 変換前のデータとの誤差の最大値を返す.
5    use ffts
6    use Math_Const
```

```
7     use max_min
8     implicit none
9     integer :: n
10    real, allocatable :: re(:)
11    complex, allocatable :: co(:)
12    real, allocatable :: err(:)
13    real, allocatable :: ren(:)
14    real :: tf1, tf2, to1, to2, ta1, ta2, err_max
15    integer :: i, l, emax, loop
16    character(1) :: flag
17    integer, dimension(5) :: prim
18    complex, allocatable, dimension(:,:) :: omegar, omegai, omeganr, omegani
19    flag=""
20    write(*,*) "Input the data number (odd number)."
21    read(*,*) n
22    write(*,*) "Input the loop number."
23    read(*,*) loop
24    write(*,*) "Do calculate the prime factors? [y/n]"
25    read(*,*) flag
26    call prim_calc( n, prim(1:4), prim(5) )
27    write(*,*) "draw result prim", prim(1), prim(2), prim(3), prim(4), prim(5)
28    allocate(re(n))
29    allocate(co(n/2))
30    allocate(ren(n))
31    allocate(err(n))
32    do i=1,n
33  !     re(i)=sin(2.0*pi*i/real(n))
34        re(i)=exp(-4.0*(i-n/2)**2)
35    end do
36    call cpu_time(ta1)
37    if(flag=='n')then
38       allocate(omegar(0:n/2-1,0:n/2-1))
39       allocate(omegai(0:n/2-1,0:n/2-1))
40       allocate(omeganr(0:n/2-1,0:n/2-1))
41       allocate(omegani(0:n/2-1,0:n/2-1))
42       call rotate_array()
43       call rotate_calc( n/2, 'r', (/prim(1)-1,prim(2),prim(3),prim(4),prim(5)/),  &
44    &                    omegar(0:prim(5)-1,0:prim(5)-1),  &
45    &                    omeganr(0:n/2-1,0:n/2-1) )
46       call rotate_calc( n/2, 'i', (/prim(1)-1,prim(2),prim(3),prim(4),prim(5)/),  &
47    &                    omegai(0:prim(5)-1,0:prim(5)-1),  &
48    &                    omegani(0:n/2-1,0:n/2-1) )
49    end if
50    call cpu_time(ta2)
51    call cpu_time( tf1 )
52    do l=1,loop
```

```
53        if(flag=='y')then
54           call r2c_ffttp_1d( n, re, co, prim='o' )
55           call c2r_ffttp_1d( n, co, ren, prim='o' )
56        else
57           call r2c_ffttp_1d( n, re, co, prim='o', prim_fact=prim,  &
58     &                        omega_fix=omegar, omegan_fix=omeganr )
59           call c2r_ffttp_1d( n, co, ren, prim='o', prim_fact=prim,  &
60     &                        omega_fix=omegai, omegan_fix=omegani )
61        end if
62     end do
63     call cpu_time( tf2 )
64     do i=1,n
65        err(i)=abs(re(i)-ren(i))
66   !write(*,*) "fft check analysis", re(i), ren(i)
67     end do
68     call max_val_1d( err, emax, err_max )
69     write(*,*) "-------------------------------"
70     write(*,*) "fft error is ", err_max
71     call cpu_time( to1 )
72     do l=1,loop
73        if(flag=='y')then
74           call r2c_ffttp_1d( n, re, co, prim='x' )
75           call c2r_ffttp_1d( n, co, ren, prim='x' )
76        else
77           prim=(/1,0,0,0,n/2/)
78           call r2c_ffttp_1d( n, re, co, prim='x', prim_fact=prim,  &
79     &                       omega_fix=omeganr, omegan_fix=omeganr )
80           call c2r_ffttp_1d( n, co, ren, prim='x', prim_fact=prim,  &
81     &                       omega_fix=omegani, omegan_fix=omegani )
82        end if
83     end do
84     call cpu_time( to2 )
85     do i=1,n
86        err(i)=abs(re(i)-ren(i))
87   !write(*,*) "fft check analysis", re(i), ren(i)
88     end do
89     call max_val_1d( err, emax, err_max )
90     write(*,*) "fft error is ", err_max
91     write(*,*) "-------------------------------"
92     write(*,*) "-------------------------------"
93     write(*,*) "cpu time (rot) is ", ta2-ta1, "[s]."
94     write(*,*) "cpu time (fft) is ", tf2-tf1, "[s]."
95     write(*,*) "cpu time (dft) is ", to2-to1, "[s]."
96     write(*,*) "-------------------------------"
97   end program
```

実行方法

make すると, 実行ファイル fft_test が作成されているので, 実行すると,

```
Input the data number (odd number).
```

と聞かれる. これはデータの個数であり, 実 FFT 変換を行うため, 必ず偶数でなければならない. 次に

```
Input the loop number.
```

と聞かれる. これは同じ動作を何回繰り返すかを指定する.

```
Do calculate the prime factors?  [y/n]
```

と聞かれる. これはデータ数を素因数分解するかどうかを聞いている. これを 'y' とすると, FFT ルーチンが開始されるごとに毎回分解し, 回転行列を計算する. 本ルーチンでは, データ数が多くなるとともに最もボトルネックになるのが, 回転行列の計算であるため, loop number を多くとっている場合には, これを 'n' にして, 最初に計算した値を毎度参照するようにした方が, FFT 本体のパフォーマンスはよい.

計算結果

```
 Input the data number (odd number).
4096
 Input the loop number.
10
 Do calculate the prime factors? [y/n]
n
          12           0           0           0           1
 -------------------------------
 fft error is   7.2353985E-04
 fft error is   7.6539267E-04
 -------------------------------
 -------------------------------
 cpu time (rot) is   0.4000240     [s].
 cpu time (fft) is   0.2280140     [s].
 cpu time (dft) is    1.372087     [s].
 -------------------------------
```

備考

特になし.

4.2.5 matrix_test

3 次元行列を与え、そのデータをもとに様々な行列計算ルーチンを実行するプログラム.

サンプル

```
1  program matrix_test
2    use Matrix_Calc
3    implicit none
4    integer, parameter :: nx=3
5    real, dimension(nx) :: x, z
6    real, dimension(nx,nx) :: a, b
7    integer :: i, j, k
8    real :: eps, accel, val
9    write(*,*) "input the method."
10   write(*,*) "[1]=gauss, [2]=LU, [3]=Gau_sei, [4]=Jacobi,"
11   write(*,*) "[5]=SOR_Gau, [6]=SOR_Jacobi, [7]=invert, [8]=eigenvalue."
12   write(*,*) "[9]=Jacobi engen."
13   read(*,*) k
14   if(k==9.or.k==8)then  ! 9 番だけ, eigen.dat を開く.
15      open(unit=10,file='eigen.dat',status='old')
16   else
17      open(unit=10,file='data.dat',status='old')
18   end if
19      do i=1,nx
20         read(10,*) (a(i,j),j=1,nx), z(i)
21      end do
22   close(unit=10,status='keep')
23   write(*,*) "output reading array"
24   do i=1,nx
25      write(*,*) (a(i,j),j=1,nx)
26   end do
27   eps=1.e-6
28   accel=1.0
29   select case(k)
30   case(1)
31      call gausss( a, z, x )
32      do i=1,nx
33         write(*,*) x(i)
34      end do
35   case(2)
36      call LU_devs( a, z, x, 10 )
37      do i=1,nx
38         write(*,*) x(i)
39      end do
```

```
40     case(3)
41        call Gau_Sei( a, z, eps, x )
42        do i=1,nx
43           write(*,*) x(i)
44        end do
45     case(4)
46        call Jacobi_algebra( a, z, eps, x )
47        do i=1,nx
48           write(*,*) x(i)
49        end do
50     case(5)
51        call SOR_Gau_Sei( a, z, eps, accel, x )
52        do i=1,nx
53           write(*,*) x(i)
54        end do
55     case(6)
56        call SOR_Jacobi_algebra( a, z, eps, accel, x )
57        do i=1,nx
58           write(*,*) x(i)
59        end do
60     case(7)
61        b=0.0
62        call invert_mat( a, b )
63        write(*,*) "answer is,"
64        do i=1,nx
65           write(*,*) (b(i,j),j=1,nx)
66        end do
67     case(8)
68        b=0.0
69        x=0.0
70        call eigenvalue_power( a, eps, val, x )
71        write(*,*) "eigenvalue is ", val
72        write(*,*) "eigen vector is"
73        do i=1,nx
74           write(*,*) x(i)
75        end do
76     case(9)
77        b=0.0
78        x=0.0
79        call Jacobi_eigen( a, x, eps )
80        write(*,*) "eigenvalue is "
81        do i=1,nx
82           write(*,*) x(i)
83        end do
84     end select
85  end program
```

実行方法

本プログラムは単体で実行すればよい. 実行すると, どのような計算を行うかを聞かれるので, 標準入力で番号を入力する. 番号は以下のような対応になっている.

1. ガウスの消去法を用いて, 3 元の連立方程式の求解を行う.
2. LU 分解を用いて, 3 元連立方程式の求解を行う.
3. ガウスザイデル法を用いて, 3 元連立方程式の求解を行う.
4. ヤコビ法を用いて, 3 元連立方程式の求解を行う.
5. ガウスザイデル法を SOR で加速しながら, 3 元連立方程式の求解を行う.
6. ヤコビ法を SOR で加速しながら, 3 元連立方程式の求解を行う.
7. 3 次元正方行列の逆行列を計算する.
8. べき乗法を用いて 3 次元の正方行列の最大固有値とそれに対応する固有ベクトルを計算する.
9. ヤコビ法を用いて, 3 次元正方行列の全固有値を計算する.

4.2.6 normal_poly

モジュール poly_normal, special_function のテストプログラム. 標準入力する関数名の関数を一定の定義域において計算し, テキストカラムデータで出力するプログラム.

本プログラムの実行によって出力されたテキストデータファイルは同じディレクトリに同梱されている ruby スクリプト dclplot を実行することで容易に可視化できる.

サンプル

```
1  program normal_poly
2  ! 多項式, 特殊関数ルーチンのデモ
3    use poly_function
4    use special_function
5    implicit none
6    integer, parameter :: n=4, nmax=100
7    integer i, j, num, m
8    real :: xmin, xmax, dx, ymin, ymax
9    real :: x(nmax), p(0:n,nmax)
10   real :: y0(nmax), y1(nmax), y2(nmax), y3(nmax), y4(nmax)
```

```
11      real :: alpha, beta, lambda
12      character(10) :: title
13    ! 対応する整数を代入すると，その関数が計算される．
14      write(*,*) "input the calculate num."
15      write(*,*) "1->Legendre."
16      write(*,*) "2->Hermite."
17      write(*,*) "3->Laguerre."
18      write(*,*) "4->Jacobi."
19      write(*,*) "5->Gegenbauer."
20      write(*,*) "6->Chevishev"
21      write(*,*) "7->Sonine"
22      write(*,*) "8->Bessel"
23      write(*,*) "9->Gamma"
24      write(*,*) "10->Neumann"
25      write(*,*) "11->df_Bess"
26      write(*,*) "12->df_Neum"
27      read(*,*) num
28      select case (num)
29      case(1)
30         xmin=-1.0
31         xmax=1.0
32         dx=(xmax-xmin)/(nmax-1)
33         do i=1,nmax
34            x(i)=xmin+dx*(i-1)
35         end do
36         call legendre(n,x,p)
37         title="LEGENDRE"
38      case(2)
39         xmin=-5.0
40         xmax=5.0
41         dx=(xmax-xmin)/(nmax-1)
42         do i=1,nmax
43            x(i)=xmin+dx*(i-1)
44         end do
45         call hermite(n,x,p)
46         do j=0,n
47            if(j.eq.0)then
48               m=1
49            else
50               m=j*m
51            end if
52            do i=1,nmax
53               p(j,i)=(exp(-0.5*(x(i)**2)))*p(j,i)/(sqrt(2.0**j*m))
54            end do
55         end do
56         title="HERMITE"
57      case(3)
58         xmin=0.0
```

```
59          xmax=20.0
60          dx=(xmax-xmin)/(nmax-1)
61          do i=1,nmax
62             x(i)=xmin+dx*(i-1)
63          end do
64          call laguerre(n,x,p)
65          do j=0,n
66             if(j.eq.0)then
67                m=1
68             else
69                m=j*m
70             end if
71             do i=1,nmax
72                p(j,i)=(exp(-0.5*x(i))/m)*p(j,i)
73             end do
74          end do
75          title="LAGUERRE"
76       case(4)
77          xmin=-1.0
78          xmax=1.0
79          dx=(xmax-xmin)/(nmax-1)
80          do i=1,nmax
81             x(i)=xmin+dx*(i-1)
82          end do
83          write(*,*) "input the alpha, beta."
84          read(*,*) alpha, beta
85          call jacobi_poly(n,x,p,alpha,beta)
86          title="JACOBI"
87       case(5)
88          xmin=-1.0
89          xmax=1.0
90          dx=(xmax-xmin)/(nmax-1)
91          do i=1,nmax
92             x(i)=xmin+dx*(i-1)
93          end do
94          write(*,*) "input the lambda."
95          read(*,*) lambda
96          call gegenbauer(n,x,p,lambda)
97          title="GEGENBAUER"
98       case(6)
99          xmin=-1.0
100         xmax=1.0
101         dx=(xmax-xmin)/(nmax-1)
102         do i=1,nmax
103            x(i)=xmin+dx*(i-1)
104         end do
105         call chebyshev(n,x,p)
106         title="CHEBYSHEV"
107      case(7)
```

```
108         xmin=0.0
109         xmax=20.0
110         dx=(xmax-xmin)/(nmax-1)
111         do i=1,nmax
112            x(i)=xmin+dx*(i-1)
113         end do
114         write(*,*) "input the alpha."
115         read(*,*) alpha
116         call sonine(n,x,p,alpha)
117   !      do j=0,n
118   !     if(j.eq.0)then
119   !        m=1
120   !     else
121   !        m=j*m
122   !     end if
123         do j=0,n
124            do i=1,nmax
125               p(j,i)=(exp(-0.5*x(i)))*p(j,i)
126            end do
127         end do
128         title="SONINE"
129      case(8)
130         xmin=0.0
131         xmax=10.0
132         dx=(xmax-xmin)/(nmax-1)
133         do i=1,nmax
134            x(i)=xmin+dx*(i-1)
135            p(0,i)=bessj(0,x(i))
136            p(1,i)=bessj(1,x(i))
137            p(2,i)=bessj(2,x(i))
138            p(3,i)=bessj(3,x(i))
139            p(4,i)=bessj(4,x(i))
140         end do
141         title="BESSEL"
142      case(9)
143         xmin=-0.5
144         xmax=2.0
145         dx=(xmax-xmin)/(nmax-1)
146         do i=1,nmax
147            x(i)=xmin+dx*(i-1)
148            p(0,i)=gamma_func(x(i))
149            p(1,i)=0.0
150            p(2,i)=0.0
151            p(3,i)=0.0
152            p(4,i)=0.0
153         end do
154         title="GAMMA"
155      case(10)
```

```
156         xmin=1.0
157         xmax=10.5
158         dx=(xmax-xmin)/(nmax-1)
159         do i=1,nmax
160            x(i)=xmin+dx*(i-1)
161            p(0,i)=bessy(0,x(i))
162            p(1,i)=bessy(1,x(i))
163            p(2,i)=bessy(2,x(i))
164            p(3,i)=bessy(3,x(i))
165            p(4,i)=bessy(4,x(i))
166            do j=0,4
167               if(p(j,i)<-1.0)then
168                  p(j,i)=999.0
169               end if
170            end do
171         end do
172         title="NEUMANN"
173      case(11)
174         xmin=0.25
175         xmax=3.0
176         dx=(xmax-xmin)/(nmax-1)
177         do i=1,nmax
178            x(i)=xmin+dx*(i-1)
179            p(0,i)=df_bessj(0,x(i))
180            p(1,i)=df_bessj(1,x(i))
181            p(2,i)=df_bessj(2,x(i))
182            p(3,i)=df_bessj(3,x(i))
183            p(4,i)=df_bessj(4,x(i))
184         end do
185         title="DFBESS"
186      case(12)
187         xmin=0.5
188         xmax=3.0
189         dx=(xmax-xmin)/(nmax-1)
190         do i=1,nmax
191            x(i)=xmin+dx*(i-1)
192            p(0,i)=df_bessy(0,x(i))
193            p(1,i)=df_bessy(1,x(i))
194            p(2,i)=df_bessy(2,x(i))
195            p(3,i)=df_bessy(3,x(i))
196            p(4,i)=df_bessy(4,x(i))
197            do j=0,4
198               if(p(j,i)>10.0)then
199                  p(j,i)=999.0
200               end if
201            end do
202         end do
203         title="DFNEUM"
```

```
204     end select
205     do i=1,nmax
206        y0(i)=p(0,i)
207        y1(i)=p(1,i)
208        y2(i)=p(2,i)
209        y3(i)=p(3,i)
210        y4(i)=p(4,i)
211     end do
212     select case (num)
213     case(10)
214        ymin=-1.0
215        ymax=1.0
216     case(11)
217        ymin=0.0
218        ymax=10.0
219     case(12)
220        ymin=0.0
221        ymax=10.0
222     end select
223   !-- text writing
224     open(unit=10,file='normal_poly.dat',status='unknown')
225     write(10,'(1000a)') "'x-axis', ", "'", trim(title), "',", "'", trim(title), "',","'",
226     write(10,'(1000a)') "'', '', '', '', '', ''"
227     do i=1,nmax
228        write(10,'(f10.5,1a,f10.5,1a,f10.5,1a,f10.5,1a,f10.5,1a,f10.5)') x(i), ',', y0(i),
229     end do
230     close(unit=10,status='keep')
231   end program
```

実行方法

本プログラムは単体で実行することができ, 実行するとどの関数を計算するかということを標準入力で決定する. 各番号は以下のような対応をしている.

1. ルジャンドル多項式.
2. エルミート多項式.
3. ラゲール多項式.
4. ヤコビ多項式.
5. ゲーゲンバウアー多項式.
6. チェビシェフ多項式.
7. ソニン多項式.
8. ベッセル関数.
9. ガンマ関数.
10. ノイマン関数.
11. 変形ベッセル関数.
12. 変形ノイマン関数.

計算結果　ルジャンドル多項式の場合

図 4.3 にルジャンドル多項式の場合を示す.

エルミート多項式の場合

図 4.4 にエルミート多項式の場合を示す.

ラゲール多項式の場合

図 4.5 にラゲール多項式の場合を示す.

ベッセル関数の場合

図 4.6 にベッセル関数の場合を示す.

ノイマン関数の場合

図 4.7 にノイマン関数の場合を示す.

変形ベッセル関数の場合

図 4.8 に変形ベッセル関数の場合を示す.

変形ノイマン関数の場合

図 4.9 に変形ノイマン関数の場合を示す.

ガンマ関数の場合

図 4.10 にガンマ関数の場合を示す.

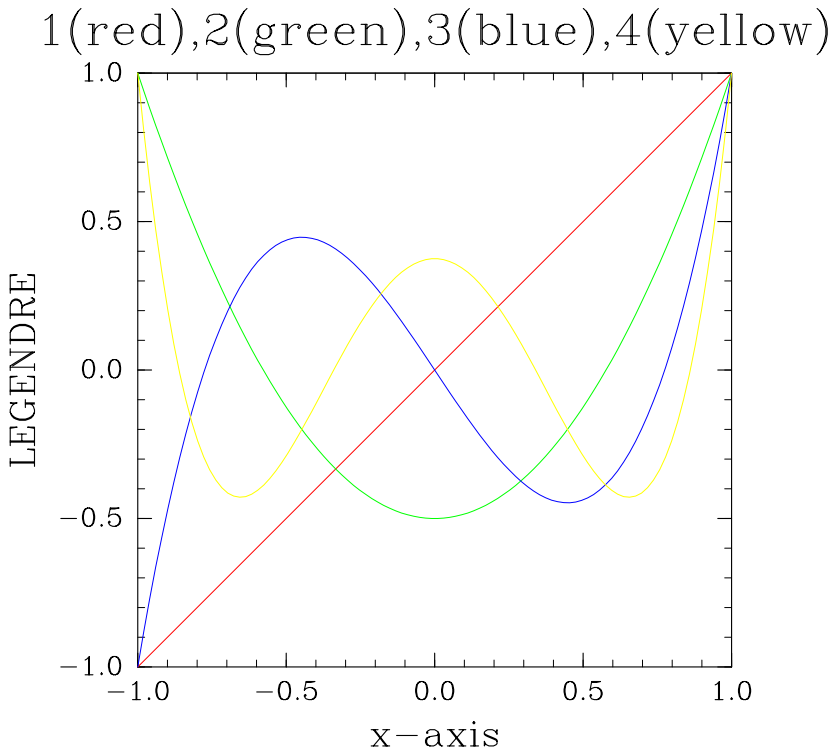


図 4.3: ルジャンドル多項式 (黒線が 0 次, 赤線が 1 次, 緑線が 2 次, 青線が 3 次, 黄線が 4 次).

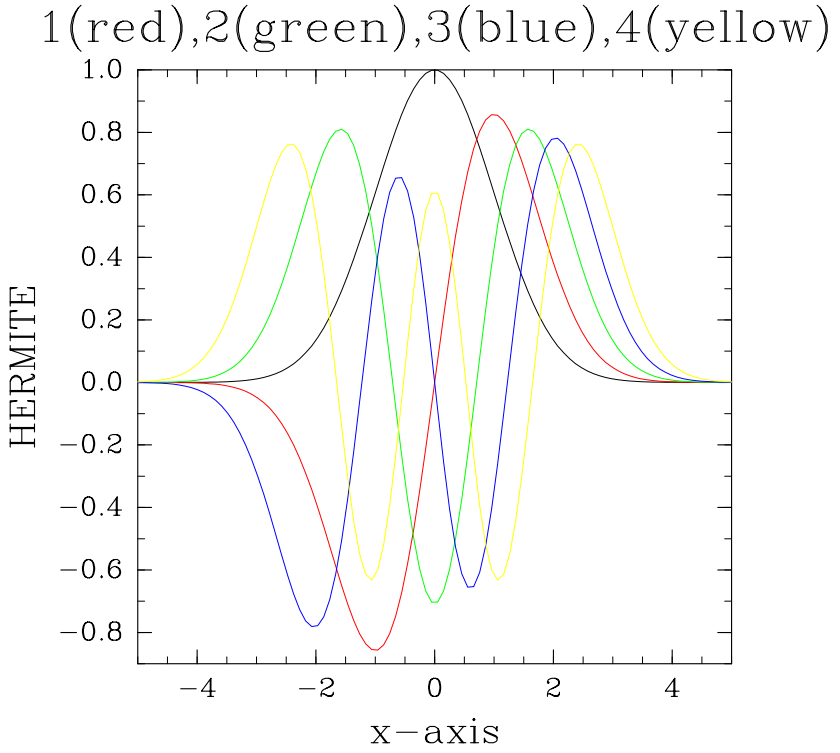


図 4.4: エルミート多項式 (黒線が 0 次, 赤線が 1 次, 緑線が 2 次, 青線が 3 次, 黄線が 4 次).

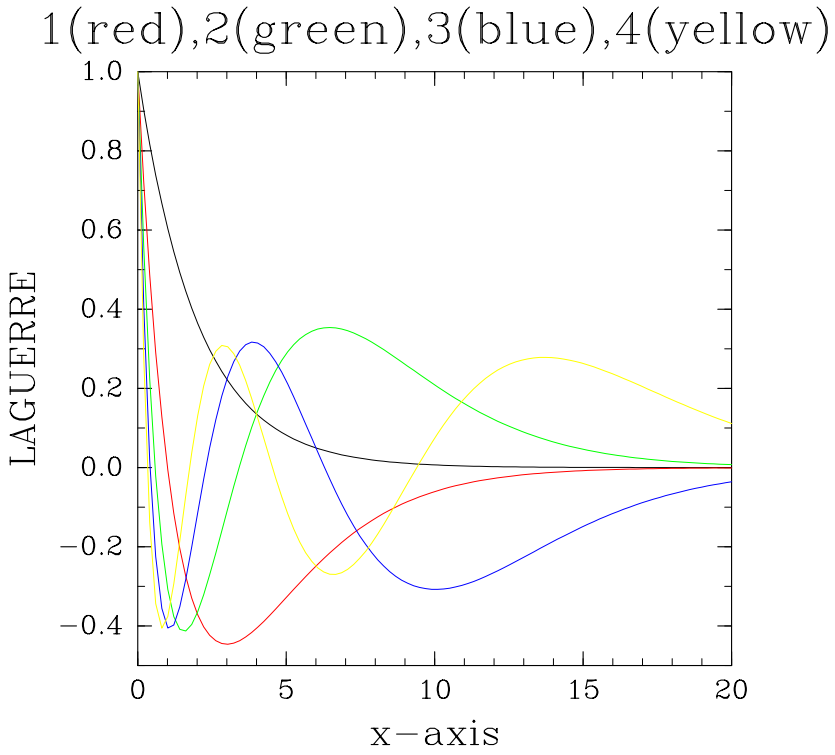


図 4.5: ラゲール多項式 (黒線が 0 次, 赤線が 1 次, 緑線が 2 次, 青線が 3 次, 黄線が 4 次).

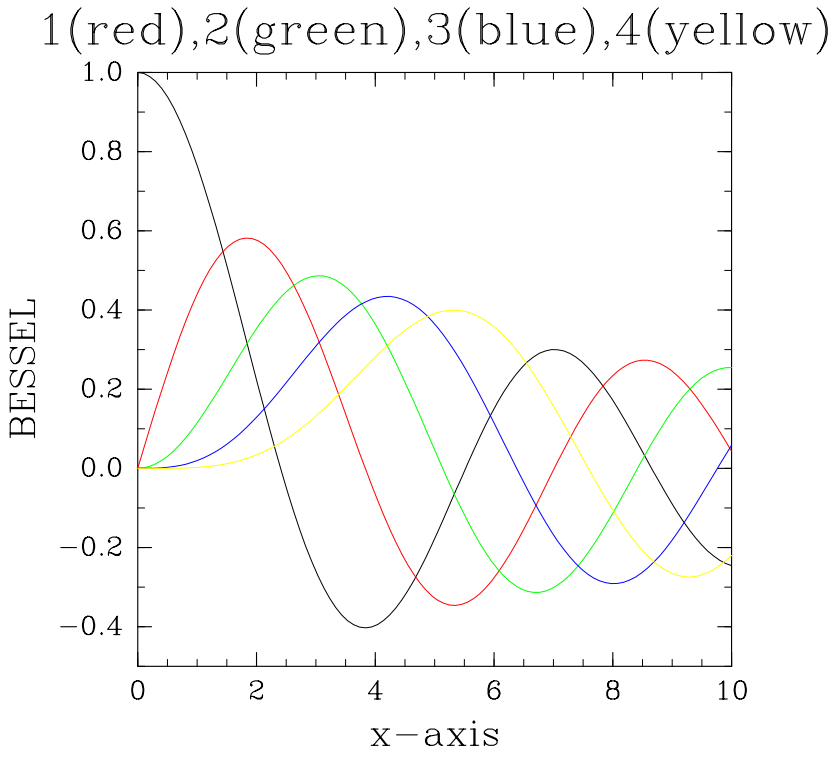


図 4.6: ベッセル関数 (黒線が 0 次, 赤線が 1 次, 緑線が 2 次, 青線が 3 次, 黄線が 4 次).

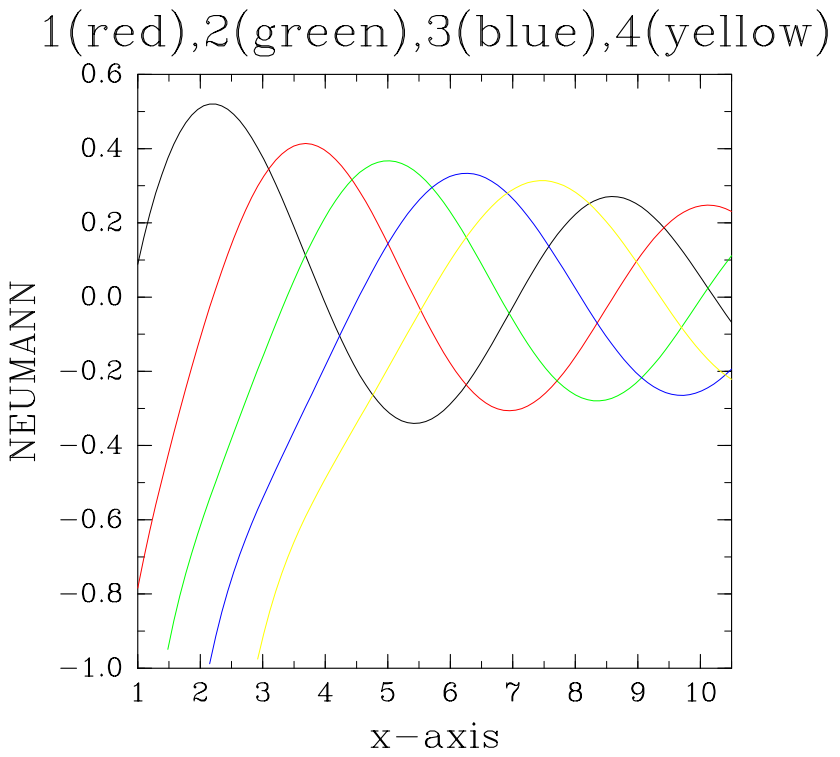


図 4.7: ノイマン関数 (黒線が 0 次, 赤線が 1 次, 緑線が 2 次, 青線が 3 次, 黄線が 4 次).

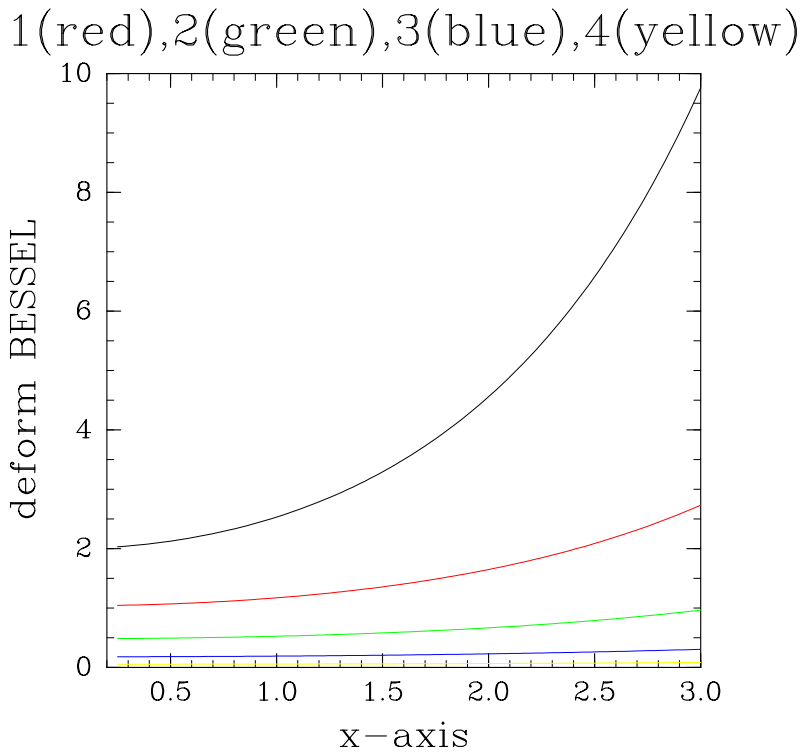


図 4.8: 変形ベッセル関数 (黒線が 0 次, 赤線が 1 次, 緑線が 2 次, 青線が 3 次, 黄線が 4 次).

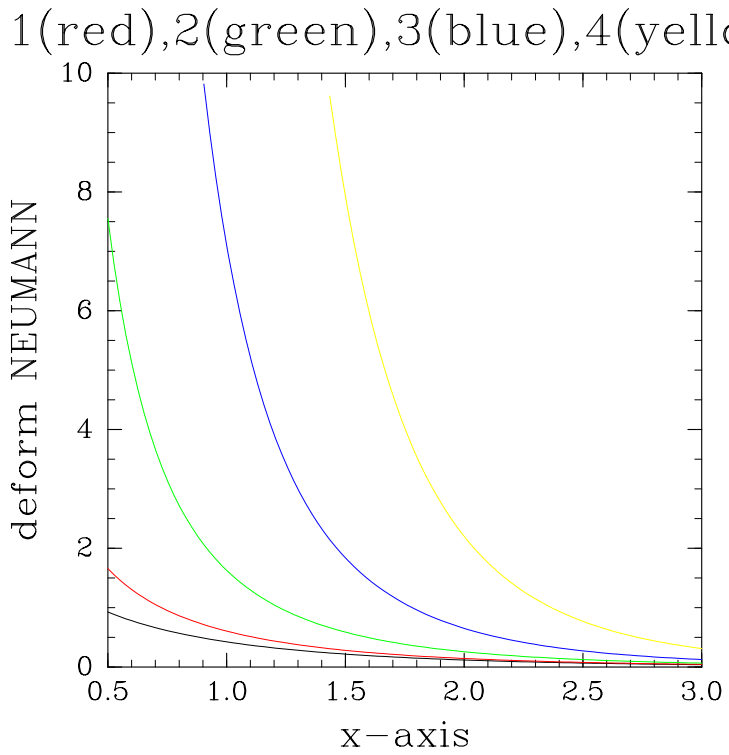


図 4.9: 変形ノイマン関数 (黒線が 0 次, 赤線が 1 次, 緑線が 2 次, 青線が 3 次, 黄線が 4 次).

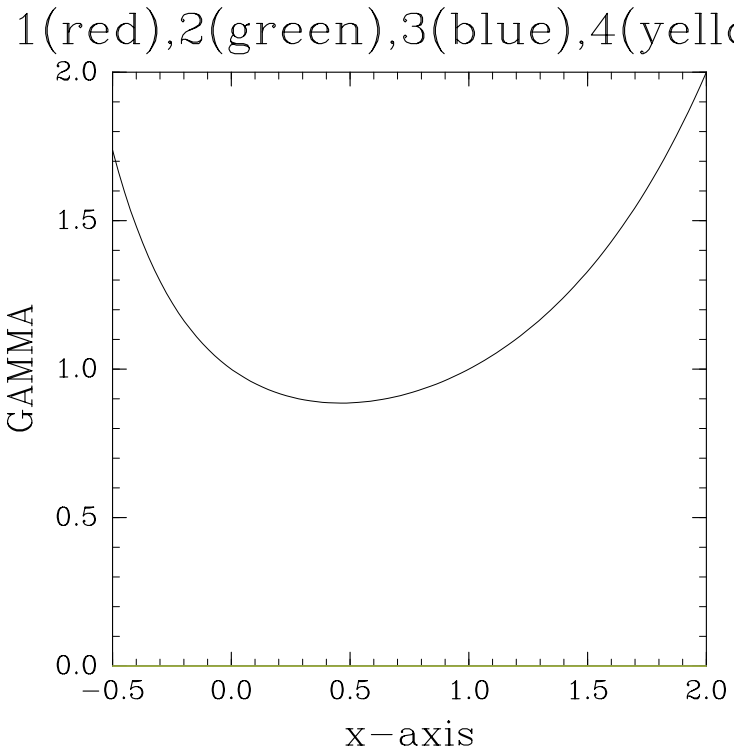


図 4.10: ガンマ関数.

4.2.7 poison

2 次元水平面におけるポアソン方程式を計算するプログラム. 強制項は領域の中心に有限領域をもつテーブル関数型の強制を設定している.

計算する方程式は

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = -\rho$$

である. ここで, ψ は 2 次元の未知関数, ρ は 2 次元の強制であり, 本計算では, 領域中心にテーブル型の関数を与えている.

ネームリスト

ネームリストファイルは poison.nml.

nx	<I>	右手系第一座標.
ny	<I>	右手系第二座標.
tp	<C(4)>	各境界での境界条件 (備考参照).
method	I	ポアソンソルバの選択 (備考参照).

サンプル

```
1  program poison
2    use gtool_history
3    use Alge_Solv
4    implicit none
5    integer :: nx, ny
6    integer :: i, j
7    real :: dx, dy
8    real, allocatable :: x(:), y(:)
9    real, allocatable :: rho(:,:)
10   real, allocatable :: psi(:,:)
11   integer :: method
12   character(4) :: tp
13  namelist /input /nx,ny,tp,method
14  read(5,input)
15   allocate(x(nx))
16   allocate(y(ny))
17   allocate(psi(nx,ny))
18   allocate(rho(nx,ny))
19   dx=1.0/real(nx-1)
20   dy=1.0/real(ny-1)
21   x=(/(dx*(i-1),i=1,nx)/)
22   y=(/(dy*(i-1),i=1,ny)/)
```

```
23      rho=0.0
24    ! do i=1,nx
25    ! do j=1,ny
26    ! rho(i,j)=exp(-10.0*((x(i)-0.5)**2+(y(j)-0.5)**2))
27    ! end do
28    ! end do
29      do i=1,nx
30         do j=1,ny
31            if((((i-50)**2+(j-50)**2)*dx*dy)<0.05*0.05)then
32               rho(i,j)=-1.0e4/8.85
33            end if
34         end do
35      end do
36      select case (method)
37      case(1)
38         call Poisson_GauSei(x,y,rho,1.0e-6,tp,psi)
39      case(2)
40         call Poisson_Jacobi(x,y,rho,1.0e-6,tp,psi)
41      end select
42    !-- gtool history (netcdf dump)
43      call HistoryCreate( &                              ! ヒストリー作成
44        & file='poison.nc', title='poison model', &
45        & source='Sample program of gtool_history/gtool5',   &
46        & institution='GFD_Dennou Club davis project',       &
47        & dims=('x','y'), dimsizes=(nx,ny),                   &
48        & longnames=('X-coordinate','Y-coordinate'),         &
49        & units=('m','m'),                                   &
50        & origin=0.0, interval=0.0 )
51      call HistoryPut( 'x', x )                            ! 次元変数出力
52      call HistoryPut( 'y', y )                            ! 次元変数出力
53      call HistoryAddVariable( &                           ! 変数定義
54        & varname='psi', dims=('x','y'), &
55        & longname='psi', units='1', xtype='float')
56      call HistoryPut('psi',psi)                          ! 変数出力
57      call HistoryAddVariable( &                           ! 変数定義
58        & varname='rho', dims=('x','y'), &
59        & longname='forcing', units='1', xtype='float')
60      call HistoryPut('rho',rho)                          ! 変数出力
61      call HistoryClose
62    end program
```

計算結果

領域中心に円形テーブル関数型の強制を与えたときのポアソン方程式の計算結果は図 4.11 である.

備考

- 境界条件の設定 tp はポアソンソルバの引数としてそのまま与えているので, 境界条件の数字の意味は 3.1.1, 3.1.2 参照.
- ポアソンソルバの設定 method は, 1 がガウスザイデル, 2 がヤコビ法.

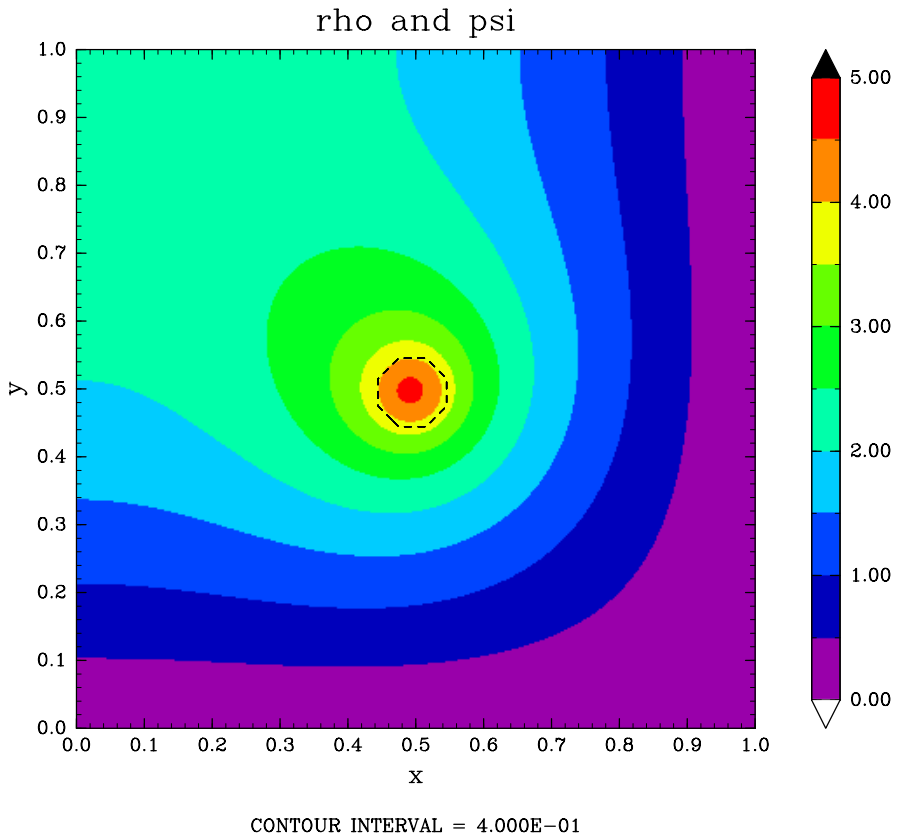


図 4.11: 領域中心にテーブル関数型の強制を与えた場合のポアソン応答. カラーが応答の分布であり, 等値線が強制の境界である.

4.2.8 adjust

教育用プログラム. 線形 1 次元の f 面浅水方程式について, 有限差分法を用いて地衡流調節と重力波放射をシミュレーションするプログラム.

方程式系などの詳細は 5.2.2 参照。

実行方法・ネームリスト

実行プログラムは `adjust` というプログラム単体である. 初期値は netcdf 形式の 1 次元格子座標データとその格子点で定義された高さ, 水平速度 2 成分のデータが格納されているファイルを読み込む。サンプル初期値は ruby-netcdf がインストールされていれば, `make.rb` というスクリプトを実行することで作成できる. この初期値データは水平速度がすべてゼロで高さのみ計算領域の中心で不連続となるプロファイルデータである（図 4.12 参照）。もしこのデータ以外で計算したい場合は各自でデータを用意すること。

実行方法は以下のコマンドである。

```
./adjust < adjust.nml
```

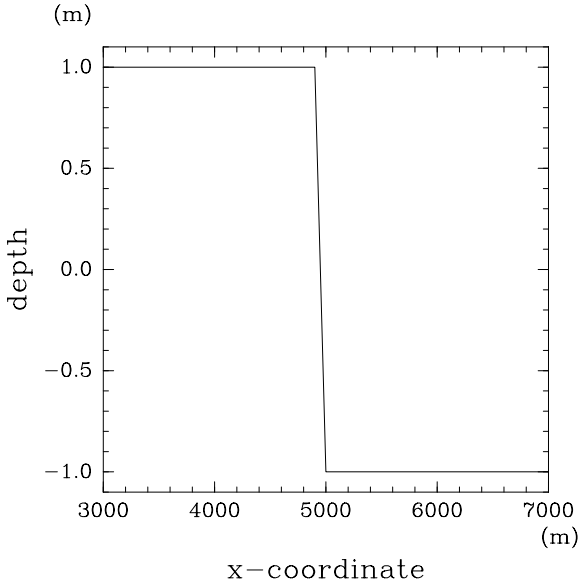


図 4.12: モデル初期の高さ偏差.

この結果, adjust.nml の oname で設定されているファイル名で計算結果が出力される.

ネームリストの変数は以下のとおりである.

```
&input
 corioli = 1.0e-1    ! コリオリパラメータ [1/s]
 beta = 1.0e-5          ! ベータ係数 [1/m s]
! beta = 0.0           ! ベータ係数 [1/m s]
 mean_height = 100.0 ! 流体層の平均厚さ [m]
 fname = 'init.dat'  ! 初期値ファイル
 oname = 'result.nc' ! 結果の出力ファイル
 nt = 20000           ! 計算時間ステップ
 dt = 0.1            ! 時間間隔 [s]
 dmpstep = 10         ! 結果の出力ステップ間隔
 nx = 100            ! x 方向の格子点数
 x_axis = 'x'        ! 初期値ファイルの x 軸の名前
 val_height = 'ht'   ! 流体層の深さ分布の名前
 val_ubar = 'ub'     ! x 方向の一般風の名前
 val_vbar = 'vb'     ! y 方向の一般風の名前
 val_h = 'h'         ! 初期値ファイルの深さ変動の名前
 val_u = 'u'         ! 初期値ファイルの x 方向速度の名前
 val_v = 'v'         ! 初期値ファイルの x 直交方向速度の名前
 bound = 2           ! 境界条件 "1" = no gradient, "2" = open bound
                     ! "3" = 周期境界条件
 steady_flag = 'ooo' ! 定常状態の基本場を結果データに入れるかどうか.
                     ! 1 文字目 = 山の高さ (ht)
```

```
                        ! 2 文字目 = x 方向の流速 (ubar)
                        ! 3 文字目 = y 方向の流速 (vbar)
    regist_flag = 'o'   ! 各物理量への付加力項フラグ
                        ! regist_flag(1) = 粘性による抵抗 (波数依存)
                        ! regist_flag(2) = エクマン摩擦による抵抗
    regist_coe = 1.0, 1.0, 1.0
                        ! regist_flag で指定した各付加力の係数
                        ! regist_coe(1) = 粘性係数 (u',v' への寄与)
                        ! regist_coe(2) = 拡散係数 (h' への寄与)
                        ! regist_coe(3) = エクマン摩擦のスピンアップ時間 [1/s]
   /
```

計算結果

デフォルトのネームリストに設定されている外部パラメータの各設定値から、重力波の位相速度はおよそ 31 m/s、変形半径はおよそ 300 m となっている。このとき、定常状態となったときの高さの偏差と地衡流調節によって駆動される地衡流を示した図がそれぞれ 4.13, 4.14 である。

図 4.15 は高さの偏差の時系列にロスビー変形半径を黒線で重ねたものである。同様に地衡流の分布を示した図が 4.16 である。

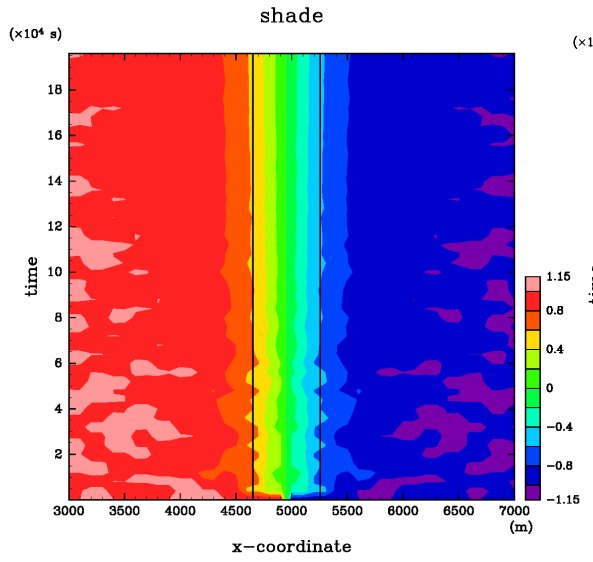


図 4.13: 高さの時系列変化。黒線がロスビー変形半径。

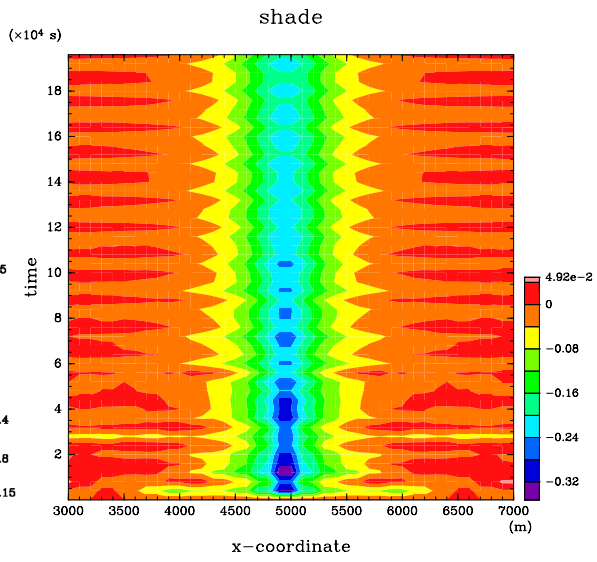


図 4.14: 地衡流の時系列変化。

さらに、図 4.17 は図 4.13 の積分開始初期部分だけを示したものであるが, 顕著な重力波が中心から左右外向きに伝播していることがわかる。また、この時系列から重力波の位相速度を計算するとおよそ 30 m/s 程度で先に示した理論値とよく合うこ

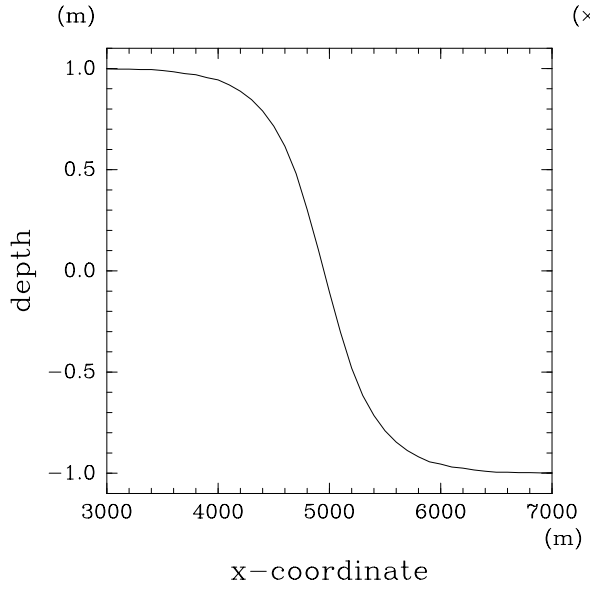


図 4.15: 定常状態での高さ。

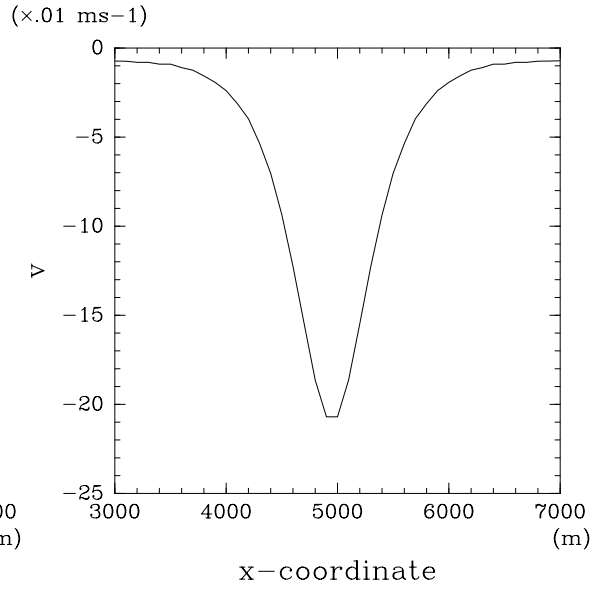


図 4.16: 定常状態での地衡流の分布。

とがわかる。波のエネルギーは振幅の 2 乗に比例するため、この図からエネルギーが重力波とともに外向きに伝播している様子も明らかであろう。

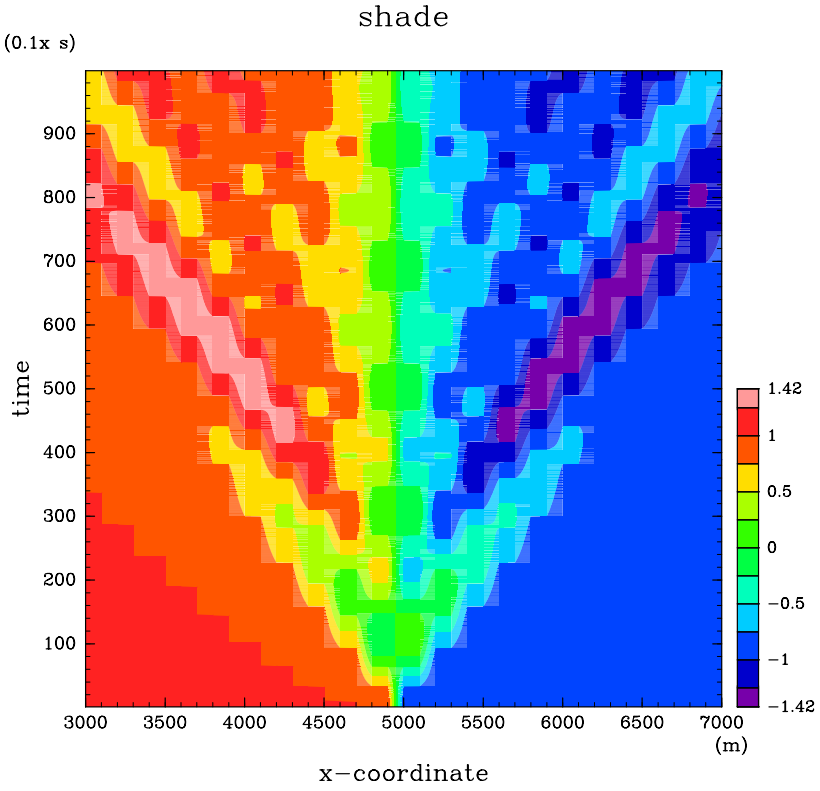


図 4.17: 積分開始初期における重力波の外向き伝播.

4.2.9 read_mgdsst_nc

mgdsst データを netcdf データに変換するためのプログラム.

実行方法・ネームリスト

実行前に, 変換する mgdsst ファイル名を全て `sst_list` という名称のファイルにテキストで保存しておく. その元ファイル名の拡張子を `.nc` とした名称で netcdf ファイルは生成される. 実行は, 単体で実行すれば, 先のリストに示された分だけ自動的に netcdf ファイルに変換される.

```
$ ./read_mgdsst_nc
```

ここで, オリジナルの mgdsst データは海氷を示す値が 888 であるが, この変換によって, その値は `263.0 K` に設定されることに注意.

計算結果

本プログラムの結果から得られた netcdf のデータをそのまま描画したものが図 4.18 である．本結果の描画ツールはこちらでは用意していないので, 各自使用しているツールを用いて変換されたい.

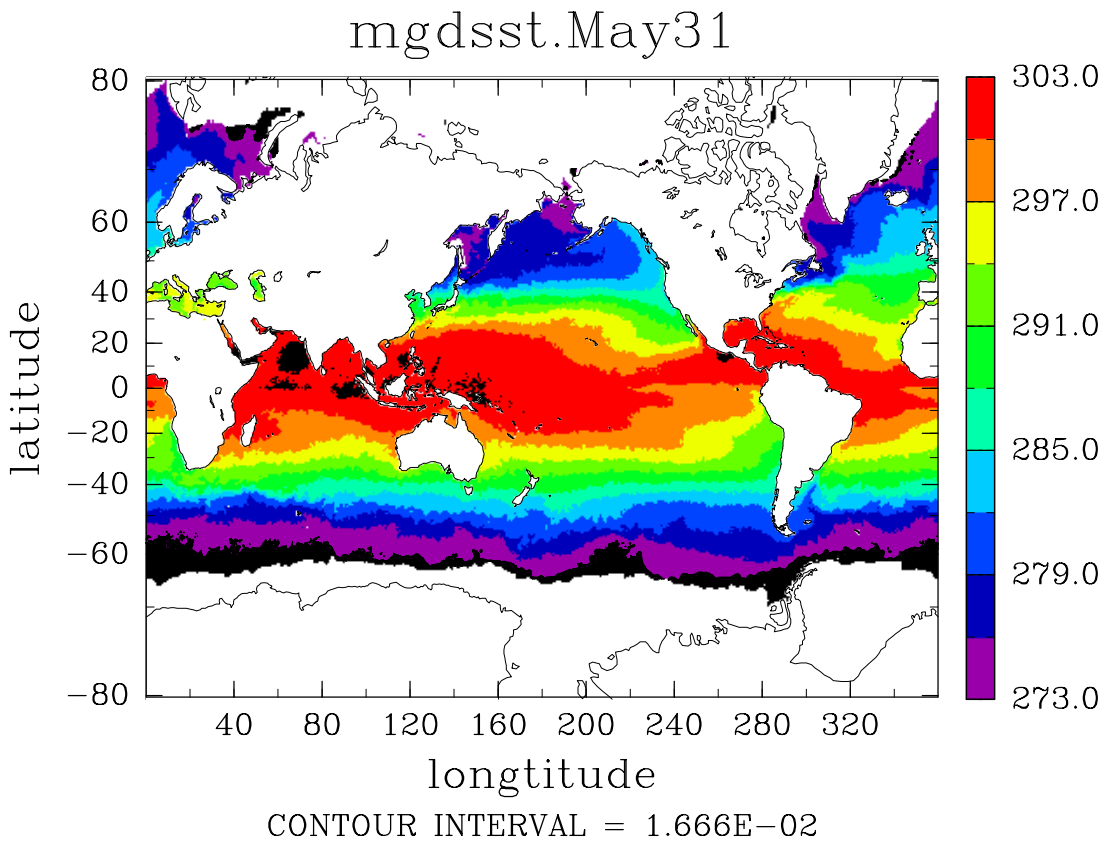


図 4.18: netcdf に変換されたデータから生成された全球海面水温の分布.

4.2.10 SEQ

Pendergrass and Willoughby 2009 (MWR) における 2 次元軸対称ソーヤーエリアッセンモデル (SEQ) である. 接線風の渦の分布と環境場の温度構造, 壁雲を想定した非断熱加熱の分布を与えると, 静力学, 傾度風バランスを満たした 2 次循環 (インフロー + 上昇流) の分布を計算する (いわゆる強制・応答問題).
demo/SEQ 以下にモデル一式が格納されている. `demo/SEQ` のディレクトリには, さらに詳細に各循環を示した図も同梱されているので, 確認されたい. 元の論文の結果とほぼ同様の結果となっていることが確認されるであろう.

なお, 詳細なモデル方程式等については付録参照.

実行方法・ネームリスト

本プログラムは 3 つのプログラムを順に実行することで 2 次循環を計算する.

1. sound_make は 1 次元の理想的なサウンディングファイルを作成する. 既にサウンディングがある場合は, このプロセスは省略することができる.
2. initial_make は先の 1 次元サウンディングファイルを 2 次元方向に拡張する. その際, このプログラム内で定義されている 1 次循環に基づいて, 傾度風バランス, 温度風バランスするように温度や気圧の水平分布を修正する.
3. 先の 2 次元初期値生成プログラムで作成された 1 次循環と非断熱加熱の分布, 温度分布をもとに, 2 次循環の分布を計算する.

```
$ ./sound_make < SEQ.nml
```
を実行すると, sounding.dat が生成される.
```
$ ./initial_make < SEQ.nml
```
を実行すると, 先の sounding.dat に基づいて, 2 次元の初期値データ initial.nc が生成される.
```
$ ./SEQ < SEQ.nml
```
を実行すると, initial.nc から 2 次循環の分布を診断的に計算する. その結果は result_initial.nc というファイルに出力される. ここで, 実行環境がマルチコア環境であるなら, 以下のコマンドを実行することにより, openMP 並列が設定される.
```
export OMP_NUM_THREADS=[number]
```
ここで, [number] には並列数を入力する. ただし, openMP 並列を行う場合は, ../Mkinclude の Fortran コンパイルフラグに openMP を実行するオプションをつけて make しておく必要があることに注意.

また, 実行に必要なネームリストファイルの内容は以下のとおりである.

```
&input
nr = 751  ! radial grid number
nz = 21  ! vertical grid number
dr = 2000.0  ! radial interval [m]
dz = 1000.0  ! vertical interval [m]
bc = '1112'  ! boundary conditions for poisson solver
             ! bc(1:1) = bottom
             ! bc(2:2) = center
             ! bc(3:3) = top
             ! bc(4:4) = outside
             ! '1' = rigid lid, '2' = non flux
fname = 'initial.nc'  ! forcing profile data (2d)
sound_name = 'sounding.dat'  ! sounding data (1d)
/
```

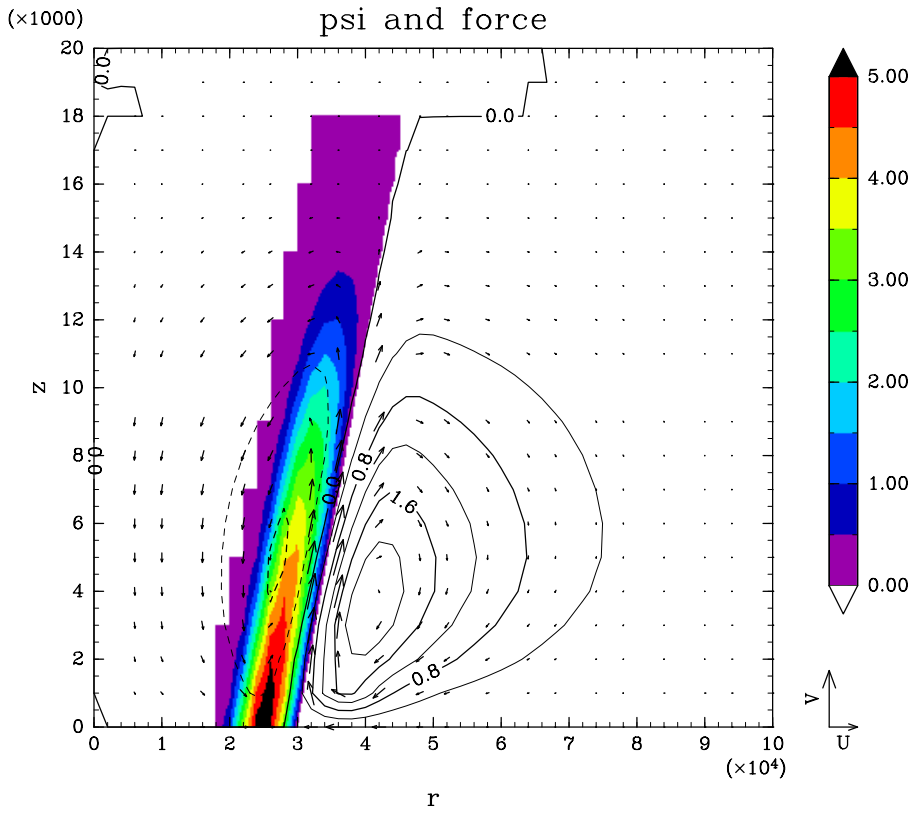


図 4.19: 診断的に求められた 2 次循環 (矢印) と非断熱加熱の分布 (カラー), さらに質量流線関数 (等値線) である.

計算結果

本プログラムの結果から得られた 2 次循環の分布は図 4.19 に示す.

描画

同ディレクトリに格納されている描画プログラムは親ディレクトリで

```
$ make draw
```

を実行すれば, SEQ ディレクトリにシンボリックリンクされている `draw` が使用できる. `draw` の使用方法は付録?? 参照.

4.2.11 thermo

`Thermo_Function` に登録されている関数について, テストを行うプログラム. 気圧, 温度, 湿度等の情報を与えて, 返された値が妥当なものであるかを検証するプログラム.

サンプル

```
1  program thermo  ! thermo 関係の関数確認用
2    use Thermo_Function
3    implicit none
4    real :: temp, pres, rh, rho, es, qv, tmp, pt
5    write(*,*) "pres [hPa]"
6    read(*,*) pres
7    write(*,*) "temp [K]"
8    read(*,*) temp
9    write(*,*) "rh [%]"
10   read(*,*) rh
11   pres=pres*100.0
12   es=RHT_2_e(rh,temp)
13   qv=eP_2_qv(es,pres)
14   rho=TP_2_rho(temp,pres)
15   pt=theta_dry(temp,pres)
16   tmp=tetens(temp)
17   write(*,*) "tetens", tetens(temp)*1.0e-2, "hPa"
18   write(*,*) "goff-gratch", goff_gratch(temp)*1.0e-2, "hPa"
19   write(*,*) "goff-gratch_i", goff_gratch_i(temp)*1.0e-2, "hPa"
20   write(*,*) "es_Bolton", es_Bolton(temp)*1.0e-2, "hPa"
21   write(*,*) "es_TD", es_TD(es)*1.0e-2, "hPa"
22   write(*,*) "LH", LH(temp), "J/kg"
23   write(*,*) "eP_2_qv", eP_2_qv(es,pres), "kg/kg"
24   write(*,*) "TP_2_qvs", TP_2_qvs(temp,pres), "kg/kg"
25   write(*,*) "qvP_2_e", qvP_2_e(qv,pres)*1.0e-2, "hPa"
26   write(*,*) "theta_dry", theta_dry(temp,pres), "K"
27   write(*,*) "theta_moist", theta_moist(temp,pres,qv), "K"
28   write(*,*) "thetaP_2_T", thetaP_2_T(temp,pres), "K"
29   write(*,*) "thetaT_2_P", thetaT_2_P(pt,temp)*1.0e-2, "hPa"
30   write(*,*) "TqvP_2_TLCL", TqvP_2_TLCL(temp,qv,pres), "K"
31   write(*,*) "thetae_Bolton", thetae_Bolton(temp,qv,pres), "K"
32   write(*,*) "thetaes_Bolton", thetaes_Bolton(temp,pres), "K"
33   write(*,*) "TqvP_2_thetae", TqvP_2_thetae(temp,qv,pres), "K"
34   write(*,*) "TqvP_2_thetaes", TqvP_2_thetaes(temp,pres), "K"
35   write(*,*) "RHT_2_e", RHT_2_e(rh,temp)*1.0e-2, "hPa"
36   write(*,*) "eT_2_RH", eT_2_RH(es,temp), "%"
37   write(*,*) "RHTP_2_qv", RHTP_2_qv(rh,temp,pres), "kg/kg"
38   write(*,*) "qvTP_2_RH", qvTP_2_RH(qv,temp,pres), "%"
39   tmp=exner_func_dry(pres)
40   write(*,*) "exner_func_dry", tmp
41   write(*,*) "hypsometric_form", hypsometric_form(pres, 0.0, temp, 100.0)
42   write(*,*) "rhoT_2_P", rhoT_2_P(rho,temp)*1.0e-2, "hPa"
43   write(*,*) "rhoP_2_T", rhoP_2_T(rho,pres), "K"
```

```
44      write(*,*) "TP_2_rho", TP_2_rho(temp,pres), "kg/m3"
45      write(*,*) "get_gamma_d", get_gamma_d()*1.0e3, "K/km"
46      write(*,*) "moist_laps_temp", moist_laps_temp(pres,temp,pres-10000.0)
47    end program
```

4.2.12 Thorpe

Thorpe and Bishop (1995) によって計算された PV inversion の 3 次元理想モデルを 2 次元版にして簡易計算させたモデル. 領域中央に円形の PV アノマリーを配置し, それに応答するジオポテンシャルの分布を計算する. ソースコードの強制項部分を変更するだけで, 様々なタイプのアノマリに対する応答を計算することができる. 出力結果は, ジオポテンシャルとそこから計算される地衡風の分布が格納されている.

実行方法・ネームリスト

本プログラムは以下のように実行する.

```
$ ./Thorpe < Thorpe.nml
これにより計算結果である Thorpe.nc が生成される.
```

また, 実行に必要なネームリストファイルの内容は以下のとおりである.

```
&input
nx=100
ny=100
tp='1111'
method=2
/
```

計算結果

本プログラムの結果から得られた 2 次循環の分布は図 4.20 に示す. 2 次元計算であるため, 地衡風は 1 成分しか計算することはできない. カラーで示した量は紙面に垂直な成分を示したものである.

4.2.13 sound_analysis

任意のテキストカラムデータを読み込み可視化するためのプログラム群. 可視化のためには, Fortran 90 版 DCL がインストールされている必要がある.

本プログラム群は高層気象観測における未処理のテキストカラムデータから初期解析のための可視化を自動で行うという目的で作成されたものである.

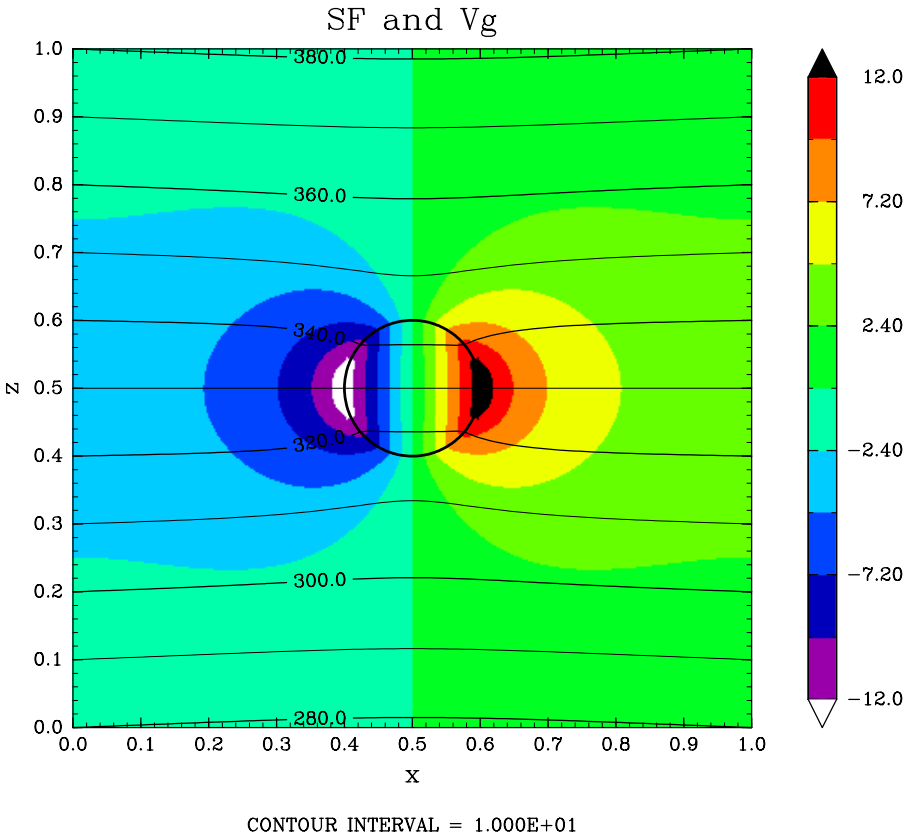


図 4.20: 中央の円形の黒線を境界にもつ PV アノマリが存在する場合の温位 (等値線) と地衡風 (カラー) の応答.

プログラム構成

本プログラムは以下のプログラム群で構成されている.

sound_conv

テキストカラム形式で格納された大気の鉛直プロファイルデータから, CReSS の 1 次元初期値を作成するための変換プログラム. 本プログラムは DCL がインストールされていなくても実行が可能である.

sound_1d

sound_conv で変換されたデータを元に, 鉛直プロファイル図を作成し, 可降水量等の鉛直パラメータを計算するプログラム. 実行には DCL がインストールされている必要がある.

sound_2d

sound_conv で変換されたデータを元に, 鉛直プロファイルの時系列図を作成し, 可降水量等の鉛直パラメータの時系列図も作成するプログラム. 実行には DCL がインストールされている必要がある.

コンパイル方法

DCL がインストールされていない場合, つまり CReSS 用の初期値作成のみ行いたい場合は,

```
$ make sound_conv
```

と実行すれば, 同ディレクトリに sound_conv というプログラムが作成されている.
　DCL がインストールされている場合, つまり可視化まで行いたい場合,

```
$ make sound_draw
```

と実行すれば, 同ディレクトリに sound_1d, sound_2d が作成されている.

実行方法

本プログラム群は全て以下のような形式で実行が可能である.

```
$ ./[program file] < [program file].nml
```

実行の流れは図 4.21 に示すとおりである. 可視化を行うには必ず sound_conv を実行して読み込むための専用形式に変換しなければならない.

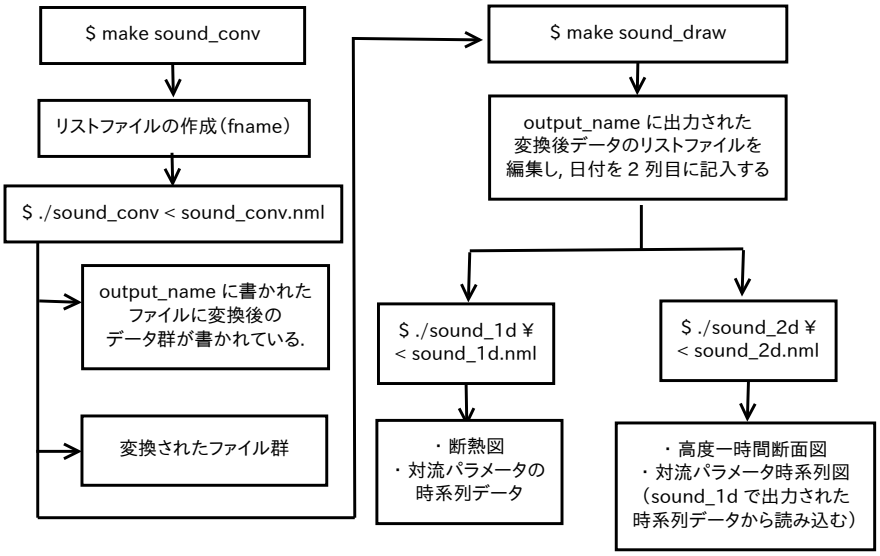


図 4.21: sound_analysis におけるプログラムの実行流れ.

ネームリスト

実行に必要なネームリストファイルの内容は以下のとおりである. これらのネームリストに必ず入力しなければならない「変換するファイル列が書かれたリストファイル」というものは, 変換したいファイルが複数存在する場合, そのファイルたちの名前を 1 行 1 ファイルで並べた 1 つのリストファイルを作成するという意味であ

る. 例として, 201103.dat から 201203.dat という 13 個のファイルを順に変換したいとすると, test.dat という空ファイルに

```
201103.dat
201104.dat
⋮
201202.dat
201203.dat
```

という内容を書けばよい. ネームリストファイルの「変換するファイル列が書かれたリストファイル」には, test.dat が該当する.

さらに, sound_1d, sound_2d については, 「読み込むファイルリストの書かれたファイル」は, 1 列目にファイル名を記述し, 2 列目には, そのファイルに対応する日付を yyyymmddhh というフォーマットで格納する. 可視化した際のタイトルはこの 2 列目の文字が代入される. 上の例を用いると,

```
201103.dat 2011030100
201104.dat 2011040100
⋮
201202.dat 2012020100
201203.dat 2012030100
```

という形式でリストファイルを作成すればよい. 重要なことは, これらのプログラムに必要なファイルは**実際に気象データの入ったテキストカラムデータ群とそのデータ群をリストアップした 1 つのリストファイル**である, ということである.

ちなみに, 可視化用プログラムのリストファイルは読み込むデータファイル名に ”unknown” という単語を記述し, 後ろに日付を入れると, その日は欠損値扱いとし, 時系列作成プログラム内で白抜き表示される. sound_2d を実行する際, リストファイルの日時の最初は 00 時, 最後も 00 時で統一すること. 該当するデータがない場合は先述の ”unknown” で代用すること.

sound_conv

変数

```
&input
fname = 'list.dat'  ! 変換するファイル列が書かれたリストファイル
sign_flag = '0a00000006510000000032400000000000000000'  ! 各カラムの表す

! '1' = height
! '2' = temperature
! '3' = pressure
! '4' = vapor
! '5' = west wind  ! positive value is from west to east
! '6' = south wind  ! positive value is from south to north
! '7' = tmp1
! '8' = tmp2
! '9' = tmp3
! '0' = no read
```

```
! 'a' = starting observation time [s] (option)
undef = '--'  ! 欠損値として定義されている文字
conv_undef = -999.0  ! 変換後のデータに対する欠損値
skip_num = 8  ! 先頭の読み飛ばし行数 [ヘッダー等を読み飛ばす]
unity = 'm', 'degC', 'hPa', '%', 'degree', 'false'  ! 単位
! unity(1:1) = 'm' .or. 'km'  ! 高度の単位
! unity(2:2) = 'K' .or. 'degC'  ! 温度の単位
! unity(3:3) = 'Pa' .or. 'hPa'  ! 気圧の単位
! unity(4:4) = '%' .or. 'g/kg' .or. 'kg/kg'  ! 水蒸気量の単位
! unity(5:5) = 'm/s' .or. 'rad' . or. 'degree'  ! 風速の単位
! unity(6:6) = 'true' or 'false'  ! 風向
! もし，風が風速と風向で記録されている場合は rad が角度を表し，原点方位は北，
! かつ，時計まわりに正方向とする．
! このような単位で記録されている場合，"west wind" に風速，"south wind" に
! 風向が読み込まれるように sign_flag を設定する．
! unity(6:6) :
! 風が流れる方向を正として格納している場合 'true'
! 風が流れてくる方向を正として格納している場合 'false'
! [exam.] : 北風が正なら，'true'，北よりの風が正なら 'false'．
limit_height = 500.0  ! reading start height [m]
snd_height = 100.0  ! 観測点の標高 [m]
output_name = 'tested.dat'  ! 変換したファイル列が書き込まれるリストファイ
ル名
conv_inter = .true.  ! 変換データを間引くかどうか
dz_conv = 100.0  ! conv_inter = .true. のときの間引き間隔 [m]
/
```

sound_1d

```
&input
z_ref = 1500.0     ! 対流パラメータ計算の際の基準高度 [m]
p_ref = -999.0  ! 対流パラメータ計算の際の基準高度 [Pa]
                  ! もし，z_ref を基準とするなら，p_ref = -999.0,
                  ! p_ref を基準とするなら，z_ref=-999.0 とすること．
list_name = 'tested.dat'  ! sound_conv で変換したデータリストファイル
dmp_flag = .true.  ! 対流パラメータをテキストデータとして保存する [true].
draw_flag = 'oooo'  ! 描画に関するフラグ．
            ! 'o' = デフォルト設定で描画する．'x' = 描画しない．
            ! その他の文字による設定は以下参照．
            ! draw_flag(1:1) = 温位，相当温位，飽和相当温位図を作成する．
            ! draw_flag(2:2) = Skew-T 作成する．
            ! draw_flag(3:3) = Skew-T に水平風速の分布を追記する．
            ! draw_flag(4:4) = Skew-T に各対流パラメータの計算結果を表示す
る．
draw_region_z = 100.0, 17000.0  ! 描画する高度領域 [m]
draw_region_p = 10000.0, 100000.0  ! 描画する圧力領域 [Pa]
            ! draw_region はどちらも要素数 2 の 1 次元配列であり，
            ! draw_region(1) はグラフの下端，draw_region(2) はグラフの上端
領域．
IWS = 2
undef = -999.0  ! 欠損値
/
```

sound_2d

```
&input
 flist = 'tested.dat'  ! 描画用時系列データファイル
                       ! 時系列データは 1 列目にファイル名,
                       ! 2 列目に yyyymmddhh 形式でその時刻を入れる
 conv_dat = ''   ! sound_1d で出力される対流パラメータを読み込み,
                       ! 時系列図を作成する.
 conv_list = 'tzpxxoxx'  ! conv_dat のデータのうち, 描くデータが入っている
                       ! カラムの順番に 'o' をつける. 'x' なら書かない.
                       ! 't' は時刻が格納されている列
                       ! 複数 'o' が入っている場合は, 別々の時系列図として
                       ! 出力する.
                       ! 'z', 'p' は対流パラメータを計算した基準高度が格
納
                       ! されている列を表す.
                       ! 時系列のタイトルはそれぞれ, データ行の先頭を参照.
 conv_undef = -999.0     ! conv_dat の未定義値.
 dz = 100.0  ! 描画間隔 [m]
 z_bot = 500.0  ! 描画下端 [m]
 z_top = 17000.0  ! 描画上端 [m]
 IWS = 2  ! DCL 描画デバイス
 title_txt = 'equivalent PT'  ! タイトル
 cmin = 300.0  ! 等値線の最小値
 cmax = 400.0  ! 等値線の最大値
 smin = 300.0  ! カラーの最小値
 smax = 400.0  ! カラーの最大値
 cont_val = 'ept'  ! 等値線として表示する変数名*
 shade_val = 'ept'  ! カラーとして表示する変数名*
 vec_val = .true.  ! 水平風速ベクトルを描くか [true で描く]
 cnum = 10  ! 等値線の本数
 snum = 10  ! カラーの数
 undef = -999.0  ! 未定義値
/
! * 描画変数の種類は以下のとおり
! 'temp' = 温度
! 'rh' = 湿度
! 'pt' = 温位
! 'ept' = 相当温位
! 'sept' = 飽和相当温位
! 'east' = 東西風
! 'north' = 南北風
```

計算結果

4.2.14 wind

モジュール Derivation, Trajectory のルーチンをテストするプログラム.
解析的な気圧場を与えて, そこから気圧勾配を計算し地衡風場を, その得られた地衡風に

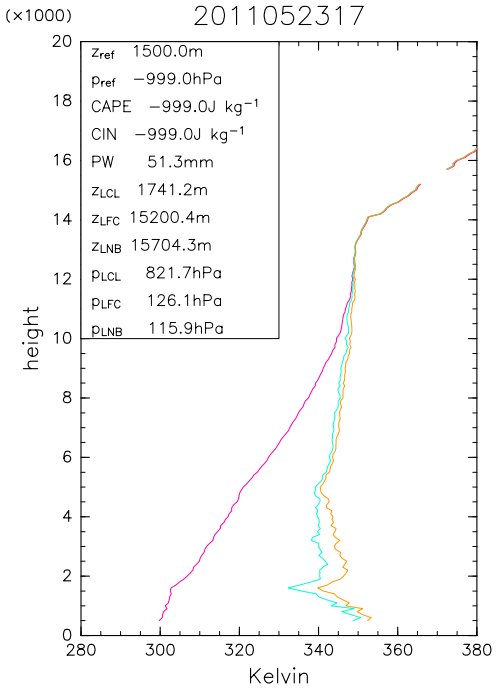


図 4.22: sound_1d で可視化されるサウンディングプロファイル (Skew-T 版).

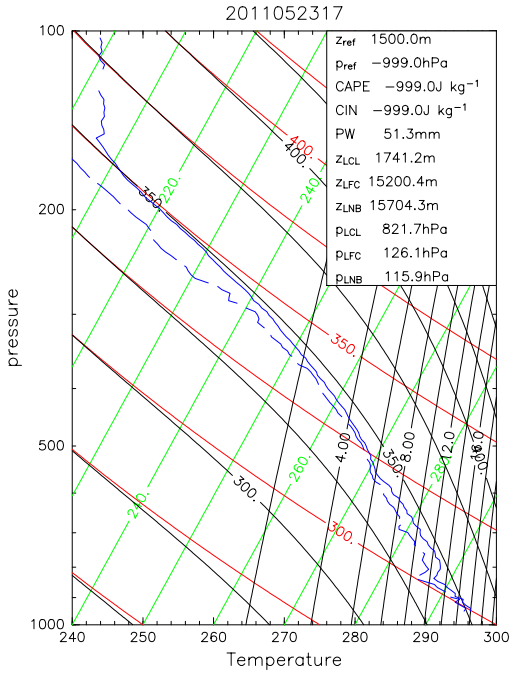


図 4.23: sound_1d で可視化されるサウンディングプロファイル (エマグラム版).

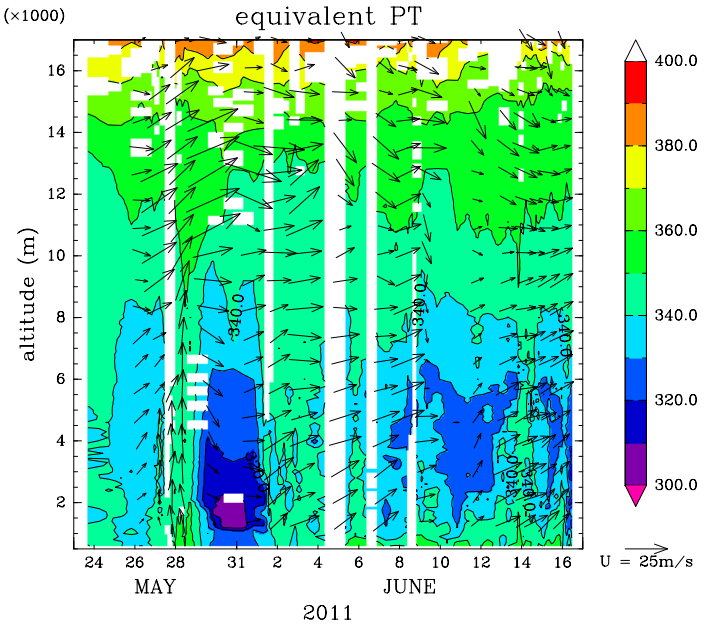


図 4.24: sound_2d で可視化される高度ー時間断面図.

ついて渦度を計算し, さらに速度場から流線計算を行う. 気圧場の分布と最後に計算された流線の軌跡が一致することを確かめることができる. ここで与える気圧場 $p(x, y)$ は

$$p(x, y) = \cos x + \cos y$$

である.

サンプル

```
1  program wind
2  ! 解析的な気圧場を与え，空間微分ルーチンから地衡風や渦度を計算するルーチン．
3  ! また，流線も計算する．
4    use Math_Const
5    use trajectory
6    use Derivation
7    use gtool_history
8    implicit none
9    integer :: i, j
10   integer, parameter :: nx=300, ny=300    ! 水平格子点数
11   integer, parameter :: tstep=20000       ! 流線の計算ステップ
12   real, dimension(nx,ny) :: u, v          ! 地衡風
13   real, dimension(nx,ny) :: rot_2d, pres  ! 鉛直渦度と気圧
14   real, dimension(tstep,1) :: tx, ty      ! 流線の x, y 座標
15   real :: x(nx), y(ny)                    ! 空間座標
16   real, parameter :: xmin=-pi, xmax=pi    ! x 軸の領域
17   real, parameter :: ymin=-pi, ymax=pi    ! y 軸の領域
18   real, parameter :: dt=0.1               ! 流線の計算時間間隔
19   real :: dx, dy                          ! 格子解像度
20   character(3) :: sche
21   dx=(xmax-xmin)/(nx-1)
22   dy=(ymax-ymin)/(ny-1)
23   x=(/((xmin+dx*(i-1)),i=1,nx)/)
24   y=(/((ymin+dy*(i-1)),i=1,ny)/)
25   write(*,*) "input scheme of time integration."
26   write(*,*) "EU1 or RK4."
27   read(*,*) sche
28   do j=1,ny
29      do i=1,nx
30         pres(i,j)=cos(x(i))+cos(y(j))  ! 解析的な気圧場を与える．
31      end do
32   end do
33   do j=1,ny
34      call grad_1d( x, pres(:,j), v(:,j) )  ! dp/dx を計算
35   end do
36   do i=1,nx
37      call grad_1d( y, pres(i,:), u(i,:) )  ! dp/dy を計算
38   end do
39   v=-v
40   call curl( x, y, u, v, rot_2d )  ! u, v から地衡風渦度を計算
41   call Stream_Line_2d( dt,  &              ! 時間間隔
42   &                    tstep,  &           ! 時間ステップ
```

```
43      &                       0.1, 0.1,  &          ! パーセルの初期位置 (x,y) 座
標
44      &                       x, y,  &              ! x, y 座標値
45      &                       u, v,  &              ! x, y の各点で定義されてい
る速度場
46      &                       tx(:,1), ty(:,1),  &  ! 実際の流線の軌跡
47      &                       sche )                ! 時間スキーム
48    !-- gtool writing (netcdf)
49      call HistoryCreate( &                                ! ヒストリー作成
50        & file='wind.nc', title='derivation test program', &
51        & source='Sample program of gtool_history/gtool5',   &
52        & institution='GFD_Dennou Club davis project',       &
53        & dims=('x','y'), dimsizes=(nx,ny),                   &
54        & longnames=('X-coordinate','Y-coordinate'),       &
55        & units=('m','m'),                                 &
56        & origin=0.0, interval=0.0 )
57      call HistoryPut('x',x)                               ! 次元変数出力
58      call HistoryPut('y',y)                               ! 次元変数出力
59      call HistoryAddVariable( &                           ! 変数定義
60        & varname='pres', dims=('x','y'), &
61        & longname='pressure', units='1', xtype='float')
62      call HistoryPut('pres',pres)                         ! 変数出力
63      call HistoryAddVariable( &                           ! 変数定義
64        & varname='rot', dims=('x','y'), &
65        & longname='rotation', units='1/s', xtype='float')
66      call HistoryPut('rot',rot_2d)                         ! 変数出力
67      call HistoryAddVariable( &                           ! 変数定義
68        & varname='u', dims=('x','y'), &
69        & longname='X-wind', units='1', xtype='float')
70      call HistoryPut('u',u)                          ! 変数出力
71      call HistoryAddVariable( &                           ! 変数定義
72        & varname='v', dims=('x','y'), &
73        & longname='Y-wind', units='1', xtype='float')
74      call HistoryPut('v',v)                          ! 変数出力
75      call HistoryClose
76      open(unit=10,file='wind.dat',status='unknown')
77         do i=1,tstep
78            write(10,*) tx(i,1), ty(i,1)
79         end do
80      close(unit=10)
81    end program
```

計算結果

解析的に与えた気圧場は図 4.25, この気圧場から計算される水平速度場は図 4.26, 得られた速度場から計算される鉛直渦度場は図 4.27, 水平速度場をもとに計算される流線は図 4.28 である。ここで、流線は領域中心から最も近い円形であり、気圧場の等値線と同心円を形成していることに注意。

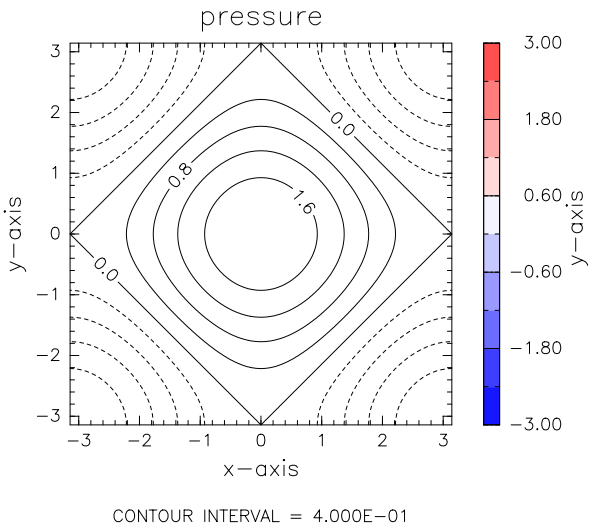


図 4.25: 解析的に与えられた気圧の場.

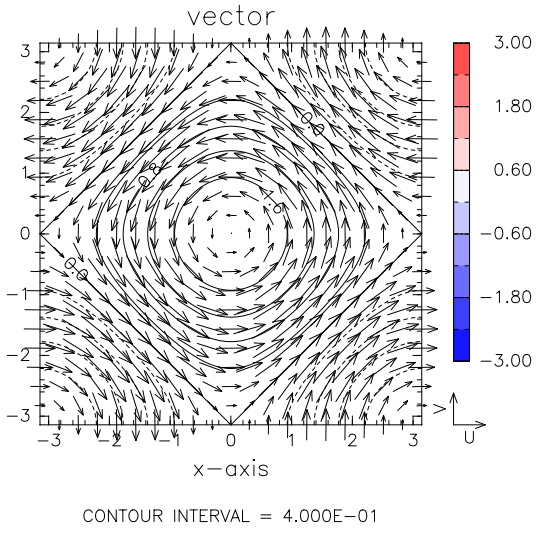


図 4.26: 気圧場から診断的に求められた水平風速場.

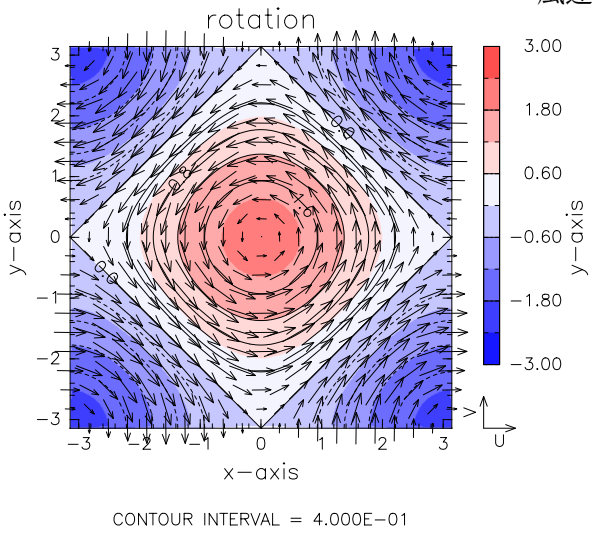


図 4.27: 速度場から求められた鉛直渦度場.

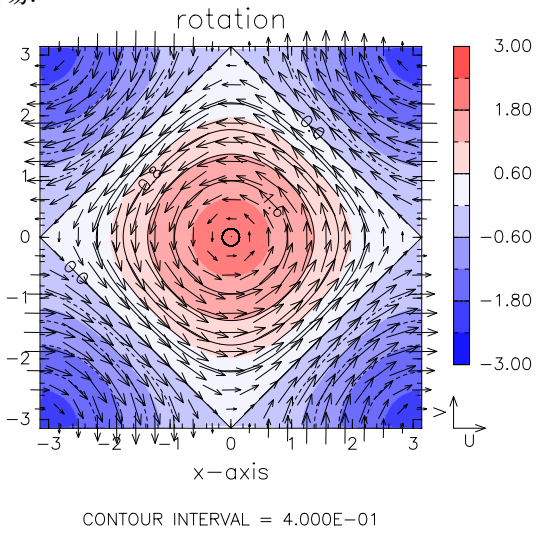


図 4.28: 速度場から計算される流線の軌跡.

4.2.15　NM01

浅水系における軸対称流な基本場に埋め込まれた非軸対称成分の時間発展を計算する非線形ハイブリッドスペクトルモデル（その発展の様子から, **洗濯機モデル**と作成者は命名）. 以下はモデルの詳細である.

座標系は円筒座標系 (r, θ)、微分の評価は r 方向には 2 次精度の中心差分、θ 方向にはフーリエスペクトル法を採用した。スペクトル法における非線形項計算は変換法を用いている。定常な軸対称流を基本場とし、初期にその流れに埋め込まれた有限振幅の擾乱の時間発展を計算する。接線方向の軸対称流の動径分布を与えると傾度風バランスするように

基本場の深さが自動的に与えられる。また、動径方向の基本場についても接線方向の軸対称流と同様に動径方向の分布を与えれば時間発展で考慮することができる[*1]。

本モデルは単一プロセッサのみで計算させるシングル版 (NM01_s) と複数プロセッサで計算させるマルチ版 (NM01_m) が存在する。シングル版は OpenMP 並列のみ行っているが, マルチ版は OpenMP 並列に加え, ノード間については MPI 並列も行っている.

シングル版

コンパイル

本プログラムは 2 つのプログラムを順に実行することで非軸対称成分の時間発展を計算する.

1. 以下のコマンドにより, 実行ファイル（make_init, VRWS）を生成する.

```
$ make
```

2. make_init は動径方向の軸対称流の分布ファイルを作成する. 既に分布ファイルがある場合は, このプロセスは省略することができる.
3. VRWS は先の軸対称流の分布をもとに, ネームリスト（後述）で設定された波数の単色波を初期値としてその非軸対称成分の時間発展を計算する.

実行方法

以下のコマンドを順次実行する.

```
$ ./make_init
input the initial file name. (作成する初期データファイル名)
You have velocity data [y/n]. (軸対称のデータがあるか; 後述)
$ export OMP_NUM_THREADS=[number] (CPU 並列数; 任意)
$ ./VRWS < namelist.nml
```

- make_init では, 2 つ目に聞かれる質問で, 軸対称風のデータも全くないときは ”y” を選択する. すると, make_init に設定されている理想的な動径方向の速度分布が与えられる. 高度場は計算した理想的なプロファイルから地衡流バランスするような分布が自動的に計算される.
- ”n” を選ぶと, 実行者が用意した風の動径プロファイルのデータを使って計算させることができる. このとき, 読み込む形式は NetCDF 形式のみ対応している. ”n” と選択すると,

[*1] 方程式系などのモデルの詳細は付録 5.2.1 参照.

```
input file name and radial grid number.
radial flow is forced to zero ?  [y/n].
```

という 2 つの質問を聞かれるので, 1 つ目は用意した NetCDF ファイルの名前と格子点数を, 2 つ目は動径風がデータとして入っている場合, それを計算で使用するか, それともゼロとして計算しないかを入力する. "Y" ならば, 動径風も計算に組み込む.

- OMP_NUM_THREADS は OpenMP 並列における CPU の並列数に対応する環境変数であり, もし, コンパイル時に OpenMP の実行が可能なオプションをつけておけば, この環境変数の設定によって並列計算が可能である. [number] には並列数を入力する.
- VRWS を実行すると, 計算が開始する. その結果は namelist.nml の foname で設定しているファイルに出力される。

ネームリスト

実行に必要なネームリストファイルの内容は以下のとおりである.

```
&input
nr = 196  ! radial grid number
hnt = 90  ! tangential truncation wavenumber
          ! tangential grid number is equal to "2 * hnt + 1"
nt = 720  ! calculating time step
rmin = 0.0  ! radial center [default]
dr = 4000.0  ! radial grid interval [m]
dt = 0.5  ! time interval [s]
dmpstp = 50  ! dumping time step
              ! dumping time interval is equal to "dmpstp * dt"
finame = "initial.nc"  ! data for axisymmetric profiles
foname = "result.nc"  ! result data
ni = 10000  ! radial grid number in "finame"
nl_flag = .false.  ! flag of non-linear term
                   ! "false" is not calculating a non linear terms
r_dmp = 600000.0  ! effective radius for Rayleigh damping
                  ! the damping is forced outside this radius
time_flag = '1'  ! time scheme :
                 ! [1] = 4th order's Runge-Kutta
                 ! [2] = Leap Frog
                 ! [3] = 1th Explicit Euler
force_flag = 'oxoxxxxxxx'  ! forcing flag
                           ! 'o' = calculating, 'x' = neglecting
                           ! force_flag(1:1) = linear advection term
                           ! force_flag(2:2) = corioli term
                           ! force_flag(3:3) = diffusion term
                           ! force_flag(4:4) = Reiley dumping term
```

```
                                ! force_flag(5:5) = non-linear advection term
                                ! force_flag(6:6) = centifugal force term
                                ! force_flag(7:7) = divergence term [only "depth"]
                                ! force_flag(8:8) = gravity wave term
                                ! force_flag(9:9) = temporary value [not using]
                                ! force_flag(10:10) = temporary value [not using]
  diff_r = 1.0  ! radial diffusion coefficient [m^2/s]
  diff_t = 1.0  ! tangential diffusion coefficient [m^2/s]
  init_n = -1   ! initial non-axisymmtric wave number
                ! -1 = random wave
  /
```

計算結果

本プログラムの結果から得られた非軸対称成分の分布は図 4.29 に示す.

描画

同ディレクトリに格納されている描画プログラム `draw_polar` を実行することで描画が可能である. このプログラムの使用方法は付録 **??** 参照.

マルチ版

[**注意**] 本プログラムを make コマンドによってコンパイルする場合, Makefile の初期設定は `../../Mkinclude_MPI` に記述されているので, ここを自身の環境に合った設定にすること.

コンパイル

本プログラムは STPK をコンパイルした Fortran コンパイラと同一のコンパイラでビルドされた MPI ライブラリが必要である. 現在, OpenMPI による並列化は正常に機能している.

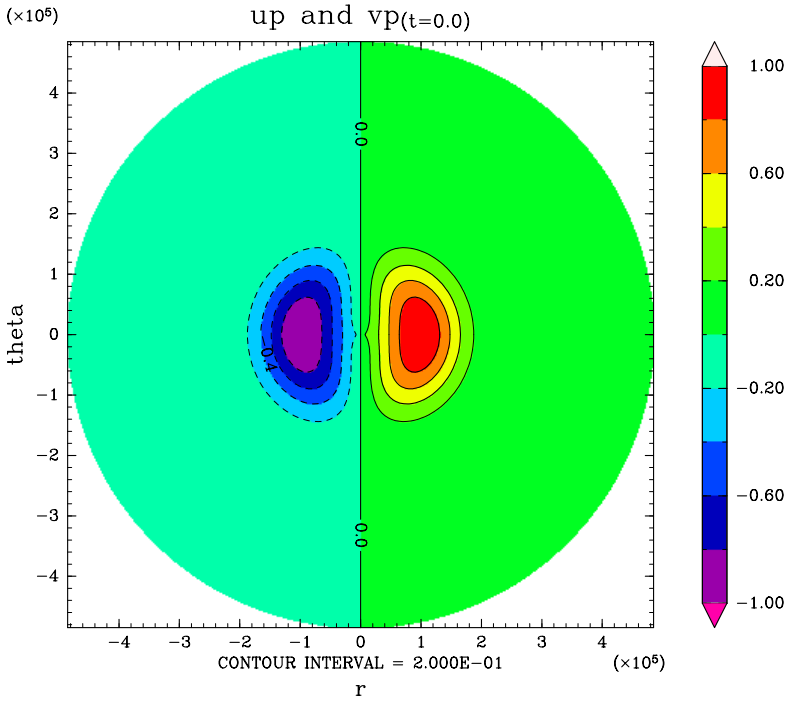
本プログラムは 4 つのプログラムを順に実行することで非軸対称成分の時間発展を計算する.

1. 以下のコマンドにより, 実行ファイル（`make_init`, `splitter`, `cpmbinator`, `VRWS`）を生成する.

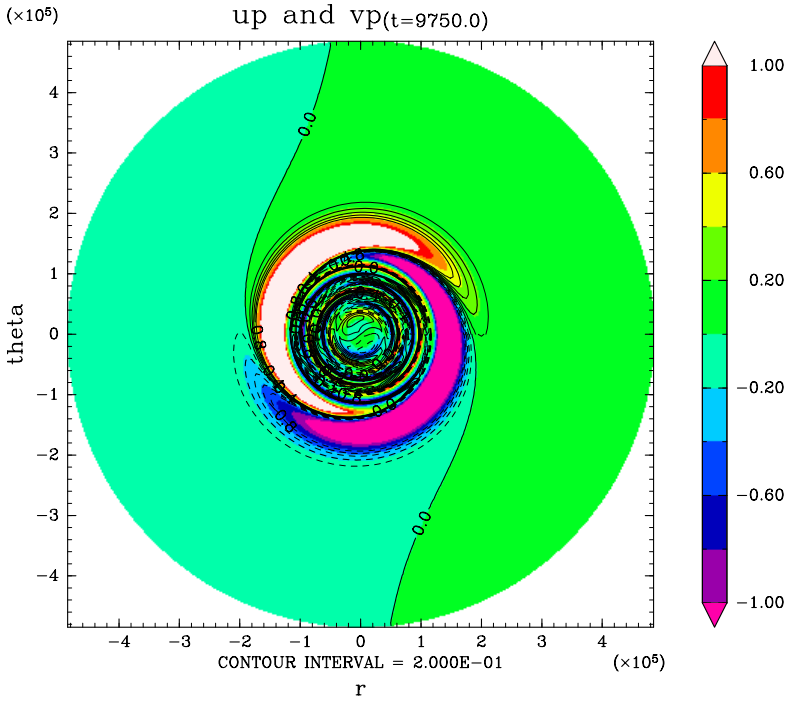
   ```
   $ make
   ```

2. `make_init` は動径方向の軸対称流の分布ファイルを作成する. 既に分布ファイルがある場合は, このプロセスは省略することができる.

3. `splitter` は `make_init` によって作成, あるいはすでに用意されている分布ファイルを namelist ファイルの情報をもとにして適切な初期値データに分割する (各プロセッサが各自読み込むファイルに分割する).



(a)



(b)

図 4.29: 計算結果から得られた非軸対称成分のうち, 等値線で動径方向の速度, カラーで接線方向の速度を示す. 上は初期値, 下は初期からある程度時間が経過した時刻の非軸対称成分の分布.

4. VRWS は先の軸対称流の分布をもとに, ネームリスト（後述）で設定された波数の単色波を初期値としてその非軸対称成分の時間発展を計算する.
5. combinator は VRWS の計算結果 (分割データ) を namelist ファイルの設定をもとに 1 つに合わせる.

実行方法

以下のコマンドを順次実行する.

```
$ ./make_init
input the initial file name. (作成する初期データファイル名)
You have velocity data [y/n]. (軸対称のデータがあるか; 後述)
./splitter < namelist.nml
$ export OMP_NUM_THREADS=[number] (CPU 並列数; 任意)
$ mpirun -np [並列ノード数 (proc)] ./VRWS < namelist.nml (後述)
./combinator < namelist.nml
```

- make_init では, 2 つ目に聞かれる質問で, 軸対称風のデータも全くないときは "y" を選択する. すると, make_init に設定されている理想的な動径方向の速度分布が与えられる. 高度場は計算した理想的なプロファイルから地衡流バランスするような分布が自動的に計算される.
- "n" を選ぶと, 実行者が用意した風の動径プロファイルのデータを使って計算させることができる. このとき, 読み込む形式は NetCDF 形式のみ対応している. "n" と選択すると,

  ```
  input file name and radial grid number.
  radial flow is forced to zero ?  [y/n].
  ```

  という 2 つの質問を聞かれるので, 1 つ目は用意した NetCDF ファイルの名前と格子点数を, 2 つ目は動径風がデータとして入っている場合, それを計算で使用するか, それともゼロとして計算しないかを入力する. "Y" ならば, 動径風も計算に組み込む.
- splitter は namelist ファイルの finame で設定されたファイル名のファイルを読み込み, 各プロセッサが読み込むための最適な初期値データを作成する.
- OMP_NUM_THREADS は OpenMP 並列における CPU の並列数に対応する環境変数であり, もし, コンパイル時に OpenMP の実行が可能なオプションをつけておけば, この環境変数の設定によって並列計算が可能である. [number] には並列数を入力する.
- ここでは, OpenMPI の並列実行コマンド mpirun を用いる. 並列ノード数は namelist ファイルに設定される proc で指定している数と同じものを用いること. VRWS を実行すると, 計算が開始する. その結果は namelist.nml の foname で設定しているファイルに出力される。

- combinator は VRWS で出力された分割ファイルを 1 つに結合する．このとき，結合されたファイル名は namelist ファイルの foname で設定されたものとなる．

ネームリスト

実行に必要なネームリストファイルの内容は以下のとおりである．

```
&input
nr = 198  ! radial grid number
hnt = 90  ! tangential truncation wavenumber
          ! tangential grid number is equal to "2 * hnt + 1"
nt = 720  ! calculating time step
rmin = 0.0  ! radial center [default]
dr = 4000.0  ! radial grid interval [m]
dt = 0.5  ! time interval [s]
dmpstp = 50  ! dumping time step
              ! dumping time interval is equal to "dmpstp * dt"
finame = "initial.nc"  ! data for axisymmetric profiles
foname = "result.nc"  ! result data
ni = 10000  ! radial grid number in "finame"
nl_flag = .false.  ! flag of non-linear term
                   ! "false" is not calculating a non linear terms
r_dmp = 600000.0  ! effective radius for Rayleigh damping
                  ! the damping is forced outside this radius
time_flag = '1'  ! time scheme :
                 ! [1] = 4th order's Runge-Kutta
                 ! [2] = Leap Frog
                 ! [3] = 1th Explicit Euler
force_flag = 'ooooooxxxx'  ! forcing flag
                           ! 'o' = calculating, 'x' = neglecting
                           ! force_flag(1:1) = linear advection term
                           ! force_flag(2:2) = corioli term
                           ! force_flag(3:3) = diffusion term
                           ! force_flag(4:4) = Reiley dumping term
                           ! force_flag(5:5) = non-linear advection term
                           ! force_flag(6:6) = centifugal force term
                           ! force_flag(7:7) = divergence term [only "depth"]
                           ! force_flag(8:8) = gravity wave term
                           ! force_flag(9:9) = temporary value [not using]
                           ! force_flag(10:10) = temporary value [not using]
diff_r = 1.0  ! radial diffusion coefficient [m^2/s]
diff_t = 1.0  ! tangential diffusion coefficient [m^2/s]
init_n = -1  ! initial non-axisymmtric wave number
             ! -1 = random wave
/
&para
proc = 2     ! MPI parallel number (nodes number)
/
```

計算結果

結果はシングル版と同じ (図 4.29 参照).

描画

同ディレクトリに格納されている描画プログラム `draw_polar` を実行することで描画が可能である. このプログラムの使用方法は付録 **??** 参照.

第5章　付録

Ver.1.0.0.0 で完成予定. 現在必要不可欠なもののみ掲載.

5.1 Statistics の付録

5.1.1 最小自乗法

あるデータの組、x, y が存在する。このデータの分布を x を変数とすると、

$$y = y(x)$$

という関係をもつ。このようなデータの組を x についての有限の多項式による線型結合で表すことを考える。ただし、その関数は x での y の値とそのときの関数の値の 2 乗差が最小となるように選ばれるとする。

まず、簡単のために x について 1 次関数で表現することを考える。このときの関数を

$$F(x) = a_0 + a_1 x \tag{5.1.1}$$

とする。ここで、a_0, a_1 は x, y に依存しない定数である。実際のデータは離散データであるので、x_i, y_i というデータの組み合わせとなる。すなわち、i 番目のデータ対は

$$y_i = y(x_i)$$

である。同様に 1 次関数も

$$F(x) = F(x_i)$$

となる。この直線 $F(x_i)$ と y_i の 2 乗差を最小とするようにとることを考える。つまり、これらの残差を R とすると、

$$R = \sum_{i=1}^{N} \{y(x_i) - F(x_i)\}^2 = \sum_{i=1}^{N} \{y(x_i) - a_0 - a_1 x_i\}^2 \tag{5.1.2}$$

が最小となるようにとられなければならない。この条件を満たす関数 F を求めるには、F の形状を決める x, y に独立な 2 定数 a_0, a_1 を (5.1.2) が最小となるようにとればよいということになる。このような問題を最小自乗法問題といい、次のような手順を用いて解かれる。

1. 残差 R を a_0, a_1 について微分する。
2. それらがともにゼロとなるような a_0, a_1 の値が R の極値である。
3. R はその定義から、a_0, a_1 のパラメータ空間において、正値をとる下に凸な関数であるため、極値が存在すれば、そこが最小値であるということになる。

この手順にしたがい、

$$\frac{\partial R}{\partial a_0} = 0, \quad \text{and,} \frac{\partial R}{\partial a_1} = 0 \tag{5.1.3}$$

を満たす a_0, a_1 を求める。(5.1.2) 式から、

$$\begin{aligned} \frac{\partial R}{\partial a_0} &= \sum_{i=1}^{N} \{y(x_i) - a_0 - a_1 x_i\} = 0 \\ \frac{\partial R}{\partial a_1} &= \sum_{i=1}^{N} x_i \{y(x_i) - a_0 - a_1 x_i\} = 0 \end{aligned} \tag{5.1.4}$$

である。これを整理すると、

$$\begin{aligned} a_0 N + a_1 \sum_{i=1}^{N} x_i &= \sum_{i=1}^{N} y(x_i) \\ a_0 \sum_{i=1}^{N} x_i + a_1 \sum_{i=1}^{N} x_i^2 &= \sum_{i=1}^{N} x_i y(x_i) \end{aligned} \tag{5.1.5}$$

という a_0, a_1 についての 1 次連立方程式が得られるので、これを解けば、a_0, a_1 が得られ、最小自乗直線が得られる。(5.1.5) 式を解くと、

$$\begin{aligned} a_0 &= \frac{\sum_{i=1}^{N} x_i^2 \sum_{i=1}^{N} y(x_i) - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i y(x_i)}{N \sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2} \\ a_1 &= \frac{N \sum_{i=1}^{N} x_i y(x_i) - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y(x_i)}{N \sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2} \end{aligned} \tag{5.1.6}$$

が得られる。同様の考えを発展させると、任意の有限次数の多項式曲線によるフィッティングが可能であることがわかる。

そこで、ここでは、$F(x) = \sum_{k=0}^{M} a_k x^k$ という多項式でフィッティングすることを考える。ここでも最小自乗法で用いた手法が適用できる[*1]。残差 R は

$$R = \sum_{i=1}^{N} \{y_i - F(x_i)\}^2 = \sum_{i=1}^{N} \left\{ y_i - \sum_{k=0}^{M} a_k x_i^k \right\}^2 \tag{5.1.7}$$

[*1] 以降では、表記の簡便さから、$y(x_i) = y_i$ と表記する。

となり、この値を最小にするには、$a_k, (k = 0, 1, \cdots, M)$ の各係数について、それによる残差の微分がゼロとなればよい。つまり、

$$\frac{\partial R}{\partial a_j} = 0, \quad (j = 0, 1, 2, \cdots M) \tag{5.1.8}$$

という $M+1$ 個の 1 次連立方程式を立てることができる。未知係数 $M+1$ 個について、独立な $M+1$ 個の連立方程式が立てられるので、完全に閉じた系であることは明らかである。これは非常に一般化されているので、任意の $M+1$ 次多項式で近似しても、同様の計算でフィッティングができることを示唆している。(5.1.8) 式を具体的に計算すると、

$$\sum_{i=1}^{N} x_i^j \left\{ y_i - \sum_{k=0}^{M} a_k x_i^k \right\} = 0, \quad (j = 0, 1, 2, \cdots M)$$

となり、これを整理すると、

$$\sum_{i=1}^{N} x_i^j y_i - \sum_{i=1}^{N} \sum_{k=0}^{M} a_k x_i^{k+j} = 0, \quad (j = 0, 1, 2, \cdots M) \tag{5.1.9}$$

となる。左辺第 2 項については、総和演算の順序が可換であるため、

$$\sum_{i=1}^{N} x_i^j y_i - \sum_{k=0}^{M} a_k \sum_{i=1}^{N} x_i^{k+j} = 0, \quad (j = 0, 1, 2, \cdots M) \tag{5.1.10}$$

となる。これを具体的に展開すると、以下のような式が得られる。

$$\begin{aligned}
j = 0 \ &: a_0 N + a_1 \sum_{i=1}^{N} x_i + \cdots + a_M \sum_{i=1}^{N} x_i^M = \sum_{i=1}^{N} y_i \\
j = 1 \ &: a_0 \sum_{i=1}^{N} x_i + a_1 \sum_{i=1}^{N} x_i^2 + \cdots + a_M \sum_{i=1}^{N} x_i^{M+1} = \sum_{i=1}^{N} x_i y_i \\
&\vdots \\
j = M \ &: a_0 \sum_{i=1}^{N} x_i^M + a_1 \sum_{i=1}^{N} x_i^{M+1} + \cdots + a_M \sum_{i=1}^{N} x_i^{2M} = \sum_{i=1}^{N} x_i^M y_i
\end{aligned} \tag{5.1.11}$$

これを見ると、$k = M+1$ 行 $j = M+1$ 列の正方行列と $M+1$ 個の係数 a_j についての列ベクトルの積とすると、正方行列の逆行列を求めることで、完全に a_j が求められることになる。このとき、正方行列は対称行列となっていることに注意。

5.1.2 線形内挿

ここでは、本ルーチンで行っている内挿方法について証明し、将来導入予定の内挿方法についても証明を行う。

まず、本ルーチンでは、すべての内挿ルーチンにおいて、線形内挿を行っている。この

内挿方法は、参照点 2 点からその間の値を 2 点の直線で結び、その直線上に値が存在すると仮定した内挿法である。1 次元の模式図は図**??**に示されている。ある値 y_{i+1}, y_i が定義された異なる 2 点 x_{i+1}, x_i をもとに、その間に存在する内挿点 x_p での線形内挿の値 y_p は参照する 2 点から計算された直線の傾きを dx とすると、

$$y_p = y_i + dx * (x_p - x_i)$$

で計算することができる。このとき、dx は $\dfrac{y_{i+1} - y_i}{x_{i+1} - x_i}$ となる。

この考えを 2 次元空間に拡張したのが、双線形内挿と呼ばれる方法であり、その模式図は図**??**に示されている。2 次元空間 x_i, y_j で定義された値 z_{ij} の、ある空間点 x_p, y_q での内挿値 z_{pq} は、以下のように計算される。

$$z_{pq} = z_{pj} + \frac{z_{pj+1} - z_{pj}}{y_{j+1} - y_j}(y_q - y_j)$$

これは、y_j, y_{j+1} の 2 点について、それぞれ x_p における内挿値 $z_{p,j+1}, z_{p,j}$ を計算し、それらの値から y 方向に線形内挿して、z_{pq} を求めるという手続きになる。上式を整理すると以下のようになる。

$$\begin{aligned}
z_{pq} &= z_{pj} + \frac{z_{pj+1} - z_{pj}}{y_{j+1} - y_j}(y_q - y_j) \\
&= z_{ij} + \frac{z_{i+1j} - z_{ij}}{x_{i+1} - x_i}(x_p - x_i) \\
&\quad + \left[z_{ij+1} + \frac{z_{i+1j+1} - z_{ij+1}}{x_{i+1} - x_i}(x_p - x_i) \right. \\
&\quad \left. - z_{ij} - \frac{z_{i+1j} - z_{ij}}{x_{i+1} - x_i}(x_p - x_i) \right] \frac{(y_q - y_j)}{y_{j+1} - y_j} \\
&= z_{ij} + \frac{z_{i+1j} - z_{ij}}{x_{i+1} - x_i}(x_p - x_i) + \frac{z_{ij+1} - z_{ij}}{y_{j+1} - y_j}(y_q - y_j) \\
&\quad [z_{i+1j+1} - z_{ij+1} - z_{i+1j} + z_{ij}] \frac{(x_p - x_i)(y_q - y_j)}{(x_{i+1} - x_i)(y_{j+1} - y_j)}
\end{aligned} \tag{5.1.12}$$

これより、双線形内挿は、$F(x, y) = a + bx + cy + dxy$ という平面上に内挿値が存在するということを仮定した内挿であるということが示される。

同様の議論で 3 次元の線形内挿を計算すると[*2]、3 次元空間 x_i, y_j, z_k で定義された値 u_{ijk} の、ある空間点 x_p, y_q, z_r での内挿値 u_{pqr} は、以下のように計算される。

$$u_{pqr} = u_{pqk} + \frac{u_{pqk+1} - u_{pqk}}{z_{k+1} - z_j}(z_r - z_k)$$

[*2]模式図は図**??**に示されている。

この計算手続きは 2 次元の内挿をさらに 3 次元に単純拡張しただけである。よって、上式をさらに整理すると、

$$
\begin{aligned}
u_{pqr} &= u_{pqk} + \frac{u_{pqk+1} - u_{pqk}}{z_{k+1} - z_j}(z_r - z_k) \\
&= u_{pjk} + \frac{u_{pj+1k} - u_{pjk}}{y_{j+1} - y_j}(y_q - y_j) \\
&\quad + \left[u_{pjk+1} + \frac{u_{pj+1k+1} - u_{pjk+1}}{y_{j+1} - y_j}(y_q - y_j) - u_{pjk} - \frac{u_{pj+1k} - u_{pjk}}{y_{j+1} - y_j}(y_q - y_j) \right] \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&= u_{pjk} + (u_{pj+1k} - u_{pjk}) \frac{(y_q - y_j)}{y_{j+1} - y_j} + (u_{pjk+1} - u_{pjk}) \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&\quad + (u_{pj+1k+1} - u_{pjk+1} - u_{pj+1k} + u_{pjk}) \frac{(y_q - y_j)}{y_{j+1} - y_j} \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&= u_{ijk} + (u_{i+1jk} - u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} \\
&\quad + \left[u_{ij+1k} + (u_{i+1j+1k} - u_{ij+1k}) \frac{(x_p - x_i)}{x_{i+1} - x_i} - u_{ijk} - (u_{i+1jk} - u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} \right] \frac{(y_q - y_j)}{y_{j+1} - y_j} \\
&\quad + \left[u_{ijk+1} + (u_{i+1jk+1} - u_{ik+1}) \frac{(x_p - x_i)}{x_{i+1} - x_i} - u_{ijk} - (u_{i+1jk} - u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} \right] \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&\quad + \left[u_{ij+1k+1} + (u_{i+1j+1k+1} - u_{ij+1k+1}) \frac{(x_p - x_i)}{x_{i+1} - x_i} - u_{ijk+1} - (u_{i+1jk+1} - u_{ijk+1}) \frac{(x_p - x_i)}{x_{i+1} - x_i} \right. \\
&\quad \left. - u_{ij+1k} - (u_{i+1j+1k} - u_{ij+1k}) \frac{(x_p - x_i)}{x_{i+1} - x_i} + u_{ijk} + (u_{i+1jk} - u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} \right] \frac{(y_q - y_j)}{y_{j+1} - y_j} \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&= u_{ijk} + (u_{i+1jk} - u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} + (u_{ij+1k} - u_{ijk}) \frac{(y_q - y_j)}{y_{j+1} - y_j} \\
&\quad + (u_{ijk+1} - u_{ijk}) \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&\quad + (u_{i+1j+1k} - u_{ij+1k} - u_{i+1jk} + u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} \frac{(y_q - y_j)}{y_{j+1} - y_j} \\
&\quad + (u_{i+1jk+1} - u_{ik+1} - u_{i+1jk} + u_{ijk}) \frac{(x_p - x_i)}{x_{i+1} - x_i} \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&\quad + (u_{ij+1k+1} - u_{ijk+1} - u_{ij+1k} + u_{ijk}) \frac{(y_q - y_j)}{y_{j+1} - y_j} \frac{(z_r - z_k)}{z_{k+1} - z_j} \\
&\quad + (u_{i+1j+1k+1} - u_{ij+1k+1} - u_{i+1jk+1} + u_{ijk+1} - u_{i+1j+1k} + u_{ij+1k} + u_{i+1jk} - u_{ijk}) \\
&\quad \times \frac{(x_p - x_i)}{x_{i+1} - x_i} \frac{(y_q - y_j)}{y_{j+1} - y_j} \frac{(z_r - z_k)}{z_{k+1} - z_j}
\end{aligned}
\tag{5.1.13}
$$

となる。

5.2 付録 : サンプル

5.2.1 NM01 におけるモデルの詳細

浅水流体における大気の支配方程式系は

$$\frac{d\boldsymbol{v}}{dt} = -f\boldsymbol{k}\times\boldsymbol{v} - g\boldsymbol{\nabla}h + \boldsymbol{F},$$

$$\frac{\partial h}{\partial t} + \boldsymbol{\nabla}\cdot(h\boldsymbol{v}) = \mathrm{Src}.$$

である。ここで、$\boldsymbol{v}, h, f, g$ は水平速度ベクトル、流体の深さ、コリオリパラメータ、重力加速度である。また、ベクトル微分演算子は水平 2 成分のみ有している。$\boldsymbol{F}$, Src はそれぞれ流れに対する外力とソースである。これを円筒座標系で表記することを考える。ラグランジュ微分は速度が水平 2 成分であるため、

$$\frac{d}{dt} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial r} + \frac{v}{r}\frac{\partial}{\partial\theta}$$

である。ここで、r, θ はそれぞれ動径、同位角方向であり、動径方向については原点から外向きを正、同位角方向については反時計回りを正とする。また、速度ベクトルは $\boldsymbol{v} = u\boldsymbol{e}_r + v\boldsymbol{e}_\theta$ で表記することができる。このとき、$\boldsymbol{e}_r, \boldsymbol{e}_\theta$ はそれぞれ動径、同位角方向の単位ベクトルであり、u, v はそれぞれの方向の速度である。すると、先の支配方程式は

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial r} + \frac{v}{r}\frac{\partial u}{\partial\theta} - \frac{v^2}{r} = fv - g\frac{\partial h}{\partial r} + F_r, \tag{5.2.14}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial r} + \frac{v}{r}\frac{\partial v}{\partial\theta} + \frac{uv}{r} = -fu - \frac{g}{r}\frac{\partial h}{\partial\theta} + F_\theta, \tag{5.2.15}$$

$$\frac{\partial h}{\partial t} + u\frac{\partial h}{\partial r} + \frac{v}{r}\frac{\partial h}{\partial\theta} + h\left[\frac{\partial u}{\partial r} + \frac{u}{r} + \frac{1}{r}\frac{\partial v}{\partial\theta}\right] = \mathrm{Src}. \tag{5.2.16}$$

となる。

本モデルは動径方向には傾度風平衡状態にあり、定常状態の軸対称流の中に埋め込まれた非軸対称成分の時間発展を計算するモデルであるため、各従属変数は同位角方向に一様でかつ、時間発展しない軸対称流と同位角方向に非一様で時間発展が許される非軸対称成分に分けて表現するのが便利である。そこでこの方程式を軸対称流と非軸対称流に分ける。つまり、各従属変数 φ を

$$\varphi(r,\theta,t) = \overline{\varphi}(r) + \varphi'(r,\theta,t)$$

と分けると (5.2.14) - (5.2.16) 式は

$$\begin{aligned}
&\frac{\partial u'}{\partial t}+\overline{u}\frac{\partial \overline{u}}{\partial r}+\overline{u}\frac{\partial u'}{\partial r}+u'\frac{\partial \overline{u}}{\partial r}+u'\frac{\partial u'}{\partial r}+\frac{\overline{v}}{r}\frac{\partial u'}{\partial \theta}+\frac{v'}{r}\frac{\partial u'}{\partial \theta}-2\frac{\overline{v}v'}{r}-\frac{v'^2}{r}\\
&=fv'-g\frac{\partial h'}{\partial r}+F_r, \qquad (5.2.17)\\
&\frac{\partial v'}{\partial t}+\overline{u}\frac{\partial \overline{v}}{\partial r}+\overline{u}\frac{\partial v'}{\partial r}+u'\frac{\partial \overline{v}}{\partial r}+u'\frac{\partial v'}{\partial r}+\frac{\overline{v}}{r}\frac{\partial v'}{\partial \theta}+\frac{v'}{r}\frac{\partial v'}{\partial \theta}+\frac{\overline{v}\overline{u}}{r}+\frac{\overline{u}v'}{r}+\frac{u'\overline{v}}{r}+\frac{u'v'}{r}\\
&=-f\left(\overline{u}+u'\right)-\frac{g}{r}\frac{\partial h'}{\partial \theta}+F_\theta, \qquad (5.2.18)\\
&\frac{\partial h'}{\partial t}+\overline{u}\frac{\partial \overline{h}}{\partial r}+\overline{u}\frac{\partial h'}{\partial r}+u'\frac{\partial \overline{h}}{\partial r}+u'\frac{\partial h'}{\partial r}+\frac{\overline{v}}{r}\frac{\partial h'}{\partial \theta}+\frac{v'}{r}\frac{\partial h'}{\partial \theta}\\
&=-\left(\overline{h}+h'\right)\left[\frac{\partial}{\partial r}\left(\overline{u}+u'\right)+\frac{\left(\overline{u}+u'\right)}{r}+\frac{1}{r}\frac{\partial v'}{\partial \theta}\right]+\mathrm{Src.} \qquad (5.2.19)
\end{aligned}$$

となる。ここで、接線方向には、傾度風平衡の関係：

$$f\overline{v}+\frac{\overline{v}^2}{r}=g\frac{\partial \overline{h}}{\partial r}$$

を用いた。また、本モデルでは、各外力、ソースとして、動径、同位角方向の 2 次の水平拡散を `diff_r`, `diff_t` という拡散係数で施す。また、領域の外側ではレイリーダンピングを施す。

5.2.2 adjust におけるモデルの詳細

f 面 1 次元線形浅水流体方程式系は

$$\frac{\partial u'}{\partial t}-f_0v'=-g\frac{\partial h'}{\partial x},$$

$$\frac{\partial v'}{\partial t}+f_0u'=0.$$

$$\frac{\partial h'}{\partial t}+H\frac{\partial u'}{\partial x}=0.$$

である。この式は基本場は静止状態を仮定し、そのときの流体の高さが H として基本場の周りで線形化したものである。このとき、初期値に不連続な h' のプロファイルを与えると, 初期に u', v' がゼロであっても地衡流調節という作用によって h' のバランスを解消しようと運動エネルギーを獲得することになる。サンプルでのプログラムはその過渡過程をシミュレートするものである。この方程式系で地衡流調節が作用する場合、方程式系がポテンシャル渦度保存を満たすので、調節後の定常状態の分布は実は初期値から見積もることができる。

5.2.3 adjust_2d におけるモデルの詳細

β 面 2 次元線形浅水流体方程式系は

$$\frac{\partial u'}{\partial t} - fv' = -g\frac{\partial h'}{\partial x},$$

$$\frac{\partial v'}{\partial t} + fu' = -g\frac{\partial h'}{\partial y},$$

$$\frac{\partial h'}{\partial t} + H\left(\frac{\partial u'}{\partial x} + \frac{\partial v'}{\partial y}\right) = 0.$$

である。この式は基本場は静止状態を仮定し、そのときの流体の高さが H として基本場の周りで線形化したものである。このとき、初期値に不連続な h' のプロファイルを与えると, 初期に u', v' がゼロであっても地衡流調節という作用によって h' のバランスを解消しようと運動エネルギーを獲得することになる。サンプルでのプログラムはその過渡過程をシミュレートするものである。この方程式系で地衡流調節が作用する場合、方程式系がポテンシャル渦度保存を満たすので、調節後の定常状態の分布は実は初期値から見積もることができる。