

predict-timeseries-20130925.tar.bz2 の取扱説明書

株式会社あいはら 研究開発チーム 山田 泰司・高橋 純

内容物

vector_io.hpp	… ベクトル入出力用のテンプレート関数
vector_numeric.hpp	… ベクトル演算用のテンプレート関数、テンプレート関数オブジェクト（valarray の上位互換）
multi_array_io.hpp	… 多次元配列入出力用のテンプレート関数（但し、2次元配列のみ）
dynamical_system.hpp	… 力学系テンプレートクラス
vector_dynamical_system.hpp	… ベクトル型力学系テンプレートクラス（力学系テンプレートクラスを継承）
difference_equations.hpp	… 差分方程式テンプレートクラス（ベクトル型力学系テンプレートクラスを継承）
ordinary_differential_equations.hpp	… 常微分方程式テンプレートクラス（ベクトル型力学系テンプレートクラスを継承）
generate-timeseries.cc	… 検証用サンプルデータ生成プログラム
embed-timeseries.cc	… 時系列データ埋め込みプログラム
embed_timeseries.hpp	… 時系列データ埋め込みテンプレート関数
prediction_tools.hpp	… 予測データベース構築用のテンプレート関数
predictor_lorenz_analogues.hpp	… ローレンツの類推法による予測モデル（差分方程式テンプレートクラスを継承）
variance_covariance_matrix.hpp	… 分散共分散行列テンプレート関数
predictor_lu_factorize.hpp	… 分散共分散行列のLU分解法による予測モデル（差分方程式テンプレートクラスを継承）
predictor_jacobian_matrix.hpp	… ヤコビアン行列推定法による予測モデル（差分方程式テンプレートクラスを継承）
predictor_rbf_networks.hpp	… 動径基底関数(RBF)ネットワークによる予測モデル（差分方程式テンプレートクラスを継承）
predict-timeseries.cc	… 時系列データ予測プログラム（ローレンツの類推法4種、LU分解法、ヤコビ行列推定法、RBF4種）
timeseries_tools.hpp	… 時系列データ／埋め込み時系列データ用正規化テンプレートクラス
stochastic_system.hpp	… 確率系テンプレートクラス
vector_stochastic_system.hpp	… ベクトル型確率系テンプレートクラス
stochastic_tools.hpp	… 統計処理関連テンプレート関数（ヒストグラム、任意ユニットサイズ版ヒストグラム、等）

<code>makefile</code>	… Unix/Mac 用メイクファイル
<code>msvc/*</code>	… Microsoft Visual C++ Express 用ソリューションファイル及びプロジェクトファイル
<code>examples/00timeseries.cc</code>	… 時系列データ入力サンプルプログラム
<code>examples/01embed.cc</code>	… 時系列データ埋め込みサンプルプログラム
<code>examples/02database.cc</code>	… 予測データベース構築サンプルプログラム
<code>examples/03predict.cc</code>	… 時系列データ予測サンプルプログラム
<code>examples/04predict.cc</code>	… 時系列データ予測（時系列の成分毎の正規化）サンプルプログラム
<code>examples/05predict.cc</code>	… 時系列データ予測（埋め込み空間上の正規化）サンプルプログラム

なお、ここに挙げていないフォルダ及びファイルは作業用のものなので、無視して構いません。

実行形式の使い方

さらに、上記「*.cc」からビルドした実行形式ファイルとして以下が付属します。

```
msvc/Release/generate-timeseries.exe    … 検証用サンプルデータ生成プログラム
msvc/Release/embed-timeseries.exe       … 時系列データ埋め込みプログラム
msvc/Release/predict-timeseries.exe     … 時系列データ予測プログラム
```

これらはコマンドラインで使います。コマンドラインプロンプトを「>」で表します。

まず、サンプルデータとして、刻み幅0.02、データ点数10000のローレンツアトラクタの時系列データファイル「lorenz.dat」を生成するには、以下のように実行します。

```
> generate-timeseries --lorenz --delta_t 0.02 -i 10000 > lorenz.dat
```

この時系列データの第1変数について埋め込み次元3、ラグ6で、埋め込みデータファイル「lorenz.dat.emb」を作成するには、以下のように実行します。

```
> embed-timeseries -d 3 -l 6 lorenz.dat > lorenz.dat.emb
```

ちなみに、第1変数が埋め込み次元2、ラグ6、第2変数は埋め込み次元0（つまり観測対象としない）、第3変数が埋め込み次元3、ラグ8のようにするには「-d 2,0,3 -l 6,0,8」のようなオプション引数を指定します。

さて、埋め込み時系列データから予測を行うにはファイル「lorenz.dat.emb」を使っても構いませんが、予測プログラムにも埋め込み機能が備わっていますので、元の時系列データファイル「lorenz.dat」を使います。例えば、この時系列データの第1変数について埋め込み次元3、ラグ6で、ローレンツの類推法の平均場版（文献1の式5.5.3）を、直接予測ステップ数2、近傍点数5、再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --lorenz_uniform -p 2 -m 5 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

また、この時系列データの第1変数について埋め込み次元3、ラグ6で、ローレンツの類推法の一般化平均場版（文献1の式5.5.3にて一般化平均にしたもの）（但し、 $l=2$, $m=-1$ ）を、直接予測ステップ数2、近傍点数5、再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --lorenz_generalized_uniform --norm_power 2 --mean_power -1 -p 2 -m 5 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

また、同様に、ローレンツの類推法の距離の逆数版（文献2の式8）を、直接予測先ステップ数2、近傍点数5、再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --lorenz_reciprocal -p 2 -m 5 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

また、同様に、ローレンツの類推法の距離の負の指数版（文献1の式5.5.5）を、直接予測先ステップ数2、近傍点数5、再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --lorenz_exponential -p 2 -m 5 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

また、同様に、分散共分散行列のLU分解法（未発表）を、直接予測先ステップ数2、近傍点数20、再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --lu_factorize -p 2 -m 20 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

また、同様に、ヤコビアン行列推定法（文献1の式5.5.9）を、直接予測先ステップ数2、近傍点数20、再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --jacobian_matrix -p 2 -m 20 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

また、同様に、動径基底関数ネットワークによる予測法（文献2の式13にて $m_a=0$ の場合）を、直接予測先ステップ数2、重みの数100、センターステップ5、自動基底関数係数($20\sigma_1^2$, $20\sigma_2^2$, $20\sigma_3^2$), 再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --rbf_network -p 2 -m 100 -c 5 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

また、同様に、スムージング動径基底関数ネットワークによる予測法（文献3の式6.17）を、直接予測先ステップ数2、重みの数100、センターステップ5、半自動基底関数係数($100, 100\sigma_2^2/\sigma_1^2, 100\sigma_3^2/\sigma_1^2$), 半自動スムージング・パラメータ($1, 1\sigma_2^2/\sigma_1^2, 1\sigma_3^2/\sigma_1^2$), 再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --smoothing_rbf_network -p 2 -m 100 -c 5 -b 1e2 --smooth 1e+0 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

また、同様に、アファイン写像プラス動径基底関数ネットワークによる予測法（文献2の式13）を、直接予測先ステップ数2、重みの数100、センターステップ5、自動基底関数係数($20\sigma_1^2$, $20\sigma_2^2$, $20\sigma_3^2$), 再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --affine_plus_rbf_network -p 2 -m 100 -c 5 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

また、同様に、アファイン写像プラス・スミージング動径基底関数ネットワークによる予測法（文献3の式6.17を拡張したもの）を、直接予測先ステップ数2、重みの数100、センターステップ5、半自動基底関数係数($100, 100\sigma_2^2/\sigma_1^2, 100\sigma_3^2/\sigma_1^2$), 半自動スミージング・パラメータ($1, 1\sigma_2^2/\sigma_1^2, 1\sigma_3^2/\sigma_1^2$), 再帰的予測ステップ数3で1000イタレーションのフリーラン予測した時系列データ「lorenz.dat.prd」を作成するには、以下のように実行します。

```
> predict-timeseries -d 3 -l 6 --affine_plus_smoothing_rbf_network -p 2 -m 100 -c 5 -b 1e2 --smooth 1e0 -r 3 -i 1000 lorenz.dat > lorenz.dat.prd
```

予測プログラム「predict-timeseries」の出力は予測対象となっている埋め込み対象成分のみとなりますので、予測結果時系列データファイル「lorenz.dat.prd」の埋め込み空間上の振る舞いを記録するには、以下のように改めて埋め込みデータファイル「lorenz.dat.prd.emb」を作成する必要があります。

```
> embed-timeseries -d 3 -l 1 lorenz.dat.prd > lorenz.dat.prd.emb
```

次節の図は、以上の出力ファイルをそれぞれグラフ化したものとなります。

実行形式の使用例の結果のグラフ

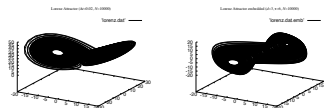


図 1：元の時系列データ、埋め込みデータ

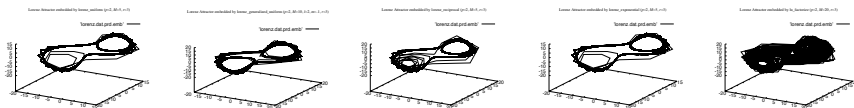


図 2：ローレンツ類推法の平均場版、一般化平均場版、距離の逆数版、距離の負の指数版、分散共分散行列のLU分解法

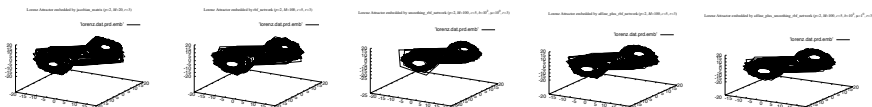


図 3：ヤコビアン行列推定法、RBFネットワーク法、SRBFネットワーク法、APRBFネットワーク法、APSRBFネットワーク法

ソースコードの利用方法

ソースコードを利用するには、以下の外部ライブラリが必要です。

- ・ Boost C++ ライブラリ Version 1.39.0 以上

特に、以下のテンプレートライブラリは必須です。

- ・ boost/multi_array.hpp … generate-timeseries.cc 以外で一般的に使用
- ・ boost/numeric/ublas/lu.hpp, matrix.hpp … predictor_lu_factrix.hpp で使用
- ・ boost/numeric/ublas/lu.hpp, matrix.hpp … predictor_jacobian_matrix.hpp で使用
- ・ boost/numeric/bindings/lapack/gesdd.hpp, gelsd.hppなどの Boost Numeric Bindings ライブラリ及び CLAPACK
 … predictor_rbf_networks.hpp で使用

また、以下のテンプレートライブラリは必須ではありませんが、上述の実行形式のビルドには必要です。

- ・ boost/lexical_cast.hpp … 実行形式プログラムのソースコードで使用

Microsoft Visual C++ 2010 Express で Debug 構成でビルドしたときにコンパイルエラーが発生する場合、以下の付属ファイルが先に読み込まれるようにプロジェクトのインクルードディレクトリを設定してください。

- ・ msvc/boost_1_47_0/boost/multi_array/iterator.hpp … boost::multi_array の修正済みヘッダファイル

同梱されているソリューションファイル及びプロジェクトファイルでは既にそのようになっています。

なお、同梱されているソリューションファイル及びプロジェクトファイルは、Boost ルートディレクトリが C:\Program Files\boost\boost_1_47_0 となるように Boost C++ ライブラリ Version 1.47.0 がインストールされていることを前提に、プロジェクトのインクルードディレクトリが設定されています。

時系列データの入出力

時系列データを標準入力から読み込み、標準出力へ書き出すには、以下の「examples/00timeseries.cc」のように行います。

```
#include <iostream>
#include <boost/multi_array.hpp>
#include "multi_array_io.hpp"
using namespace std;
int main()
{
    boost::multi_array<double, 2> timeseries;
    cin >> timeseries;
    cout << timeseries << endl;
}
```

「timeseries」に時系列データが2次元配列で入力されます。次元数は行頭のベクトルの要素数により決定され、データ数は行数分、記憶域の許す限り読み込まれます。そして、「multi_array_io.hpp」が数値のみのTab-Separated Values(TSV)形式、Comma-Separated Values(CSV)形式の2次元配列データの入出力機能を提供しています。

時系列データの埋め込み

時系列データを標準入力から読み込み、その時系列データを各成分について次元2,0,3、ラグ6,0,8で埋め込んで、標準出力へ書き出すには、以下の「examples/01embed.cc」ように行います。但し、以下ではコロン「:」のみの行は、上述の繰り返しになりますので省略しています。よって、完全な動作についてはサンプルプログラムをご覧ください。

```
#include "embed_timeseries.hpp"
using namespace std;
int main()
{
    :
    vector<size_t> dimensions, lags;
    boost::multi_array<double, 2> embedded_timeseries;
    size_t n_dimension, n_data;
    vector<size_t> target_indice, source_indice;
    dimensions.resize(3); lags.resize(3);
    dimensions[0] = 2; lags[0] = 6;
    dimensions[1] = 0; lags[1] = 0;
    dimensions[2] = 3; lags[2] = 8;
    embed_timeseries(timeseries, dimensions, lags,
                     embedded_timeseries, n_data, n_dimension, target_indice, source_indice);
    cout << embedded_timeseries << endl;
}
```

「timeseries」には上述の時系列データ、「dimensions」には時系列データの各成分毎の埋め込み次元、「lags」には時系列データの各成分毎の埋め込みラグを指定します。埋め込み次元が1以下の成分については、ラグは意味を成しません。そして、

「embedded_timeseries」に埋め込み時系列データ、「n_data」には埋め込み時系列データの数、「n_dimension」には次元の総数(上例では5)、「target_indice」には予測対象となる添字の配列(上例では1,4)が返されます。「source_indice」には埋め込み結果の添字に対応する元の時系列データの添字(上例では0,0,2,2,2)が返されます。

予測データベースの構築

埋め込み時系列データから予測データベースを構築するには、以下の「examples/02database.cc」ように行います。但し、以下ではコロン「:」のみの行は、上述の繰り返しになりますので省略しています。よって、完全な動作についてはサンプルプログラムをご覧ください。

```

:
#include "prediction_tools.hpp"
using namespace std;
int main()
{
    :
    size_t prediction_step = 1;
    vector<pair<vector<double>, vector<double> > > database;
    append_timeseries_database(embedded_timeseries, prediction_step, database);
}

```

「embedded_timeseries」には上述の埋め込み時系列データ、「prediction_step」には直接予測先ステップ数、「database」にベクトル型のペアのベクトル型として予測データベースが構築されます。ちなみに、新たな埋め込み時系列データとこの「database」で手続き「append_timeseries_database」を呼び出すと、新たな予測データベースが追加されます。

予測モデルから時系列データの予測

構築した予測データベースと予測モデル（以下の例ではヤコビアン行列推定法）から時系列データの予測をするには、以下の「examples/03predict.cc」ように行います。但し、以下ではコロン「:」のみの行は、上述の繰り返しになりますので省略しています。よって、完全な動作についてはサンプルプログラムをご覧ください。

```

:
#include "predictor_lu_factorize.hpp"
using namespace std;
int main()
{
    :
    size_t n_neighbor = 30, recursive_step = 1, iteration = 5000;
    predictor_lu_factorize<double>::parameters_t parameters = { database, n_neighbor };
    predictor_lu_factorize<double> predictor(parameters);
    predictor.v0 = database[database.size()-1].second;
    for (size_t i=0; i<iteration; ++i) {
        size_t j;
        for (j=0; j<recursive_step; ++j) {
            predictor.map();
            predictor.update();
        }
        for (j=0; j+1<target_indice.size(); ++j)
            cout << predictor.v1[target_indice[j]] << ',';
        cout << predictor.v1[target_indice[j]] << endl;
    }
}
```

「n_neighbor」には近傍点数、「recursive_step」には再帰的予測ステップ数、「iteration」にはフリーラン予測のイタレーション回数を指定します。上例では、「predictor.map()」でデータベースの最後の時刻の値を初期値(predictor.v0)として予測値(predictor.v1)を推定し、「predictor.update()」で予測値を新たな初期値に更新し、それを繰り返すことで再帰的フリーラン予測の値を次々と出力しています。他の予測手法についても同様に行うことができます。詳しくは、「predict-timeseries.cc」をご覧ください。

元の時系列データの段階で正規化すべき予測モデルから時系列データの予測

先の時系列データの例にて、成分毎に正規化した予測をするには、以下の「examples/04predict.cc」ように行います。但し、以下ではコロンの「:」のみの行は、上述の繰り返しになりますので省略しています。よって、完全な動作についてはサンプルプログラムをご覧ください。

```

:
#include "timeseries_tools.hpp"
:
using namespace std;
int main()
{
    boost::multi_array<double, 2> timeseries;
    cin >> timeseries;

    timeseries_normalizer_statistic<double> tn;
    tn.normalize(timeseries);

    :
    for (size_t i=0; i<iteration; ++i) {
        size_t j;
        for (j=0; j<recursive_step; ++j) {
            predictor.map();
            predictor.update();
        }
        for (j=0; j+1<target_indice.size(); ++j)
            cout << tn.unnormalize(source_indice[target_indice[j]], predictor.v1[target_indice[j]]) << ',';
        cout << tn.unnormalize(source_indice[target_indice[j]], predictor.v1[target_indice[j]]) << endl;
    }
}
```

「timeseries_normalizer_*<double>」のように正規化のためのインスタンス、上例では「tn」を作成します。成分毎の正規化ですので、埋め込む前に tn.normalize を行います。そして、予測結果に対して tn.unnormalize を行うことで正規化前の値に戻すことが出来ますので、上例のように target_indice, source_indice を活用して下さい。他の正規化のためのインスタンスについても同様に行うことができます。詳しくは、補遺Aと「predict-timeseries.cc」をご覧ください。

埋め込み時系列データの段階で正規化すべき予測モデルから時系列データの予測

先の時系列データの例にて、埋め込み空間上で正規化した予測をするには、以下の「examples/05predict.cc」ように行います。但し、以下ではコロン「:」のみの行は、上述の繰り返しになりますので省略しています。よって、完全な動作についてはサンプルプログラムをご覧ください。

```

:
#include "timeseries_tools.hpp"
:
using namespace std;
int main()
{
    :
    embed_timeseries(timeseries, dimensions, lags,
                     embedded_timeseries, n_data, n_dimension, target_indice, source_indice);

    timeseries_normalizer_domain<double> etn;
    etn.normalize(embedded_timeseries);
    :

    for (size_t i=0; i<iteration; ++i) {
        :
        for (j=0; j+1<target_indice.size(); ++j)
            cout << tn.unnormalize(source_indice[target_indice[j]], etn.unnormalize(predictor.v1)[target_indice[j]]) << ',';
        cout << tn.unnormalize(source_indice[target_indice[j]], etn.unnormalize(predictor.v1)[target_indice[j]]) << endl;
    }
}
```

「timeseries_normalizer_*<double>」のように正規化のためのインスタンス、上例では「etn」を作成します。埋め込み空間上での正規化ですので、埋め込んだ後に etn.normalize を行います。そして、予測結果に対して etn.unnormalize を行うことで正規化前の値に戻すことが出来ますので、上例のようにして下さい。詳しくは、補遺Aと「predict-timeseries.cc」をご覧ください。

これらのサンプルプログラムの実行形式ファイルを以下のように実行させます。

```
> 01embed < example.dat > example.dat.emb  
> 03predict < example.dat > example.dat.emb.prd
```

そして、その入出力データをグラフ化したものが以下の図となります。

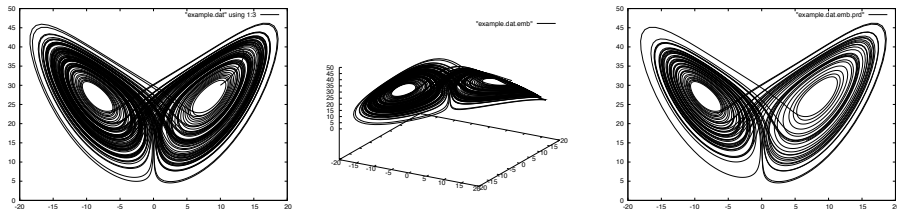


図3：元の時系列データの第1，3成分、埋め込みデータ、分散共分散行列のLU分解法

まとめ

サンプルプログラムやプログラムのコードをご覧になれば分かるように、C++テンプレートプログラミングにより、汎用的で柔軟なカオス時系列解析および予測技術の実装が、極めてシンプルに実現されています。よって、実用的なカオス時系列予測の研究開発を目指すには、これは極めて理想的な出発点になるものと考えます。

補遺A

離散時間力学系

$$x(t+1) = F(x(t)), \quad x \in \mathbb{R}^m$$

再構成状態ベクトル

$$\chi(t) = (x_{i_1}(t - \tau_1), x_{i_2}(t - \tau_2), \dots, x_{i_d}(t - \tau_d))$$

$$\text{where } x \in \mathbb{R}^m, \chi \in \mathbb{R}^d,$$

$$\{i_j | i_j, j \in \mathbb{Z}, 0 < i_j \leq m, 0 < j \leq d\}$$

予測モデル

$$\hat{x}_i(t+p) = \hat{F}_i(\chi(t)) \quad \text{もしくは} \quad \hat{\chi}(t+p) = \hat{F}(\chi(t)) \quad \text{で表す。}$$

学習データセット

$$\mathcal{L} = \{\mathbf{X}_n, \mathbf{Y}_n\}_{n=1}^N,$$

$$\mathbf{X}_n = \chi(t_n), \quad \mathbf{Y}_n = \chi(t_n + p)$$

M近傍点群の添字リスト

k_i : the indices of M -neighbor points of $\chi(t)$ in χ .

ローレンツの類推法の平均場版

$$\hat{\chi}(t+p) = \frac{1}{M} \sum_{i=1}^M \chi(k_i + p) \quad (\text{文献1の式5.5.3})$$

ローレンツの類推法の一般化平均場版

$$\hat{\chi}(t+p) = \sqrt[m]{\frac{1}{M} \sum_{i=1}^M \chi(k_i + p)^m} \quad (\text{文献1の式5.5.3にて一般化平均にしたもの})$$

geometric mean for $m \rightarrow 0$ as follows:

$$\hat{\chi}(t+p) = \sqrt[M]{\prod_{i=1}^M \chi(k_i + p)}$$

ここでの近傍点は特に一般化されたノルムから求める。

$$\|v\|_l = \sqrt[l]{\sum_{i=1}^d |v_i|^l}$$

maximum norm for $l \rightarrow \infty$ as follows:

$$\|v\|_\infty = \max(|v_i|)$$

ローレンツの類推法の距離の逆数版

$$\hat{\chi}(t+p) = \begin{cases} \frac{\sum_{i=1}^M |\chi(k_i) - \chi(t)|^{-1} \chi(k_i + p)}{\sum_{i=1}^M |\chi(k_i) - \chi(t)|^{-1}} \\ \chi(k_1 + p) & \text{if } \chi(k_i) \neq \chi(t) \text{ for all } i \\ \chi(k_1 + p) & \text{otherwise} \end{cases} \quad (\text{文献2の式8})$$

ローレンツの類推法の距離の負の指数版

$$\hat{\chi}(t+p) = \frac{\sum_{i=1}^M \exp(-|\chi(k_i) - \chi(t)|) \chi(k_i + p)}{\sum_{i=1}^M \exp(-|\chi(k_i) - \chi(t)|)} \quad (\text{文献1の式5.5.5})$$

分散共分散行列のLU分解法

$$\hat{\chi}(t+p) = \mathbf{L}(k_1) (\chi(t) - \chi(k_1)) + \chi(k_1 + p),$$

$$\mathbf{y}' = \chi(k_i) - \chi(k_1),$$

$$W_{kl} = \frac{1}{M} \sum_{i=1}^M y'_{ik} y'_{il},$$

$$\mathbf{L} \cdot \mathbf{U} = \mathbf{W} \quad (\text{未発表})$$

ヤコビアン行列推定法

$$\hat{\chi}(t+p) = \mathbf{G}(k_1) (\chi(t) - \chi(k_1)) + \chi(k_1 + p),$$

$$\mathbf{y}' = \chi(k_i) - \chi(k_1),$$

$$\mathbf{z}' = \chi(k_i + p) - \chi(k_1 + p),$$

$$\mathbf{G}(k_1) \mathbf{W} = \mathbf{C},$$

$$W_{kl} = \frac{1}{M} \sum_{i=1}^M y'_{ik} y'_{il},$$

$$C_{kl} = \frac{1}{M} \sum_{i=1}^M z'_{ik} y'_{il},$$

$$\mathbf{W}^T \mathbf{G}(k_1)^T = \mathbf{C}^T. \quad (\text{文献1の式5.5.9})$$

アフファイン写像プラス動径基底関数ネットワークによる予測法

$$P = \begin{bmatrix} 1 & \chi_1(t_1) & \dots & \chi_{m_a}(t_1) & \varphi(|\chi(t_1) - \mathbf{c}_1|) & \dots & \varphi(|\chi(t_1) - \mathbf{c}_{m_b}|) \\ & & & & \vdots & & \\ 1 & \chi_1(t_N) & \dots & \chi_{m_a}(t_N) & \varphi(|\chi(t_N) - \mathbf{c}_1|) & \dots & \varphi(|\chi(t_N) - \mathbf{c}_{m_b}|) \end{bmatrix}$$

$$\mathbf{w}_k = (a_0, a_1, \dots, a_{m_a}, b_1, \dots, b_{m_b})^T,$$

$$\mathbf{c}_i = \chi(m_a + ci \bmod N), \quad c \in \mathbb{Z},$$

$$\mathbf{y}_k = (\chi_k(t_1 + p), \dots, \chi_k(t_N + p))^T$$

$$P \mathbf{w}_k = \mathbf{y}_k \quad (\text{文献2の式13})$$

アフライン写像プラス・スムージング動径基底関数ネットワークによる予測法

$$P = \begin{bmatrix} 1 & \chi_1(t_1) & \dots & \chi_{m_a}(t_1) & \varphi(|\chi(t_1) - \mathbf{c}_1|) & \dots & \varphi(|\chi(t_1) - \mathbf{c}_{m_b}|) \\ & & & & \vdots & & \\ 1 & \chi_1(t_N) & \dots & \chi_{m_a}(t_N) & \varphi(|\chi(t_N) - \mathbf{c}_1|) & \dots & \varphi(|\chi(t_N) - \mathbf{c}_{m_b}|) \end{bmatrix}$$

$$Q = \begin{bmatrix} 0 & \underbrace{0 \dots 0}_{m_a} & \ddot{\varphi}(|\chi(t_1) - \mathbf{c}_1|) + (d-1) \frac{\dot{\varphi}(|\chi(t_1) - \mathbf{c}_1|)}{|\chi(t_1) - \mathbf{c}_1|} \dots \ddot{\varphi}(|\chi(t_1) - \mathbf{c}_{m_b}|) + (d-1) \frac{\dot{\varphi}(|\chi(t_1) - \mathbf{c}_{m_b}|)}{|\chi(t_1) - \mathbf{c}_{m_b}|} \\ & & \vdots \\ 0 & \underbrace{0 \dots 0}_{m_a} & \ddot{\varphi}(|\chi(t_N) - \mathbf{c}_1|) + (d-1) \frac{\dot{\varphi}(|\chi(t_N) - \mathbf{c}_1|)}{|\chi(t_N) - \mathbf{c}_1|} \dots \ddot{\varphi}(|\chi(t_N) - \mathbf{c}_{m_b}|) + (d-1) \frac{\dot{\varphi}(|\chi(t_N) - \mathbf{c}_{m_b}|)}{|\chi(t_N) - \mathbf{c}_{m_b}|} \end{bmatrix}$$

$$w_k = (a_0, a_1, \dots, a_{m_a}, b_1, \dots, b_{m_b})^T,$$

$$\mathbf{c}_i = \chi(m_a + ci \bmod N), \quad c \in \mathbb{Z},$$

$$y_k = (\chi_k(t_1 + p), \dots, \chi_k(t_N + p))^T$$

$$\left(P^T P + \frac{\mu}{N} Q^T Q \right) w_k = P^T y_k$$

(文献3の式6.17を拡張したもの)

成分毎の0-1正規化と復元化

The 0-1 normalizer:

$$\bar{x}_i(t) = \frac{x_i(t) - \min\{x_i(t)\}}{\max\{x_i(t)\} - \min\{x_i(t)\}},$$

where $0 < i \leq d$, $0 \leq t < T$.

The 0-1 unnormalizer:

$$x_i(t) = \bar{x}_i(t) \{\max\{x_i(t)\} - \min\{x_i(t)\}\} + \min\{x_i(t)\}.$$

成分毎の統計的正規化と復元化

The statistic normalizer:

$$\bar{x}_i(t) = \frac{x_i(t) - \mu_i}{\sigma_i},$$

where $0 < i \leq d$, $0 \leq t < T$ and

$$\mu_i = \frac{1}{T} \sum_{t=0}^{T-1} x_i(t),$$

$$\begin{aligned} \sigma_i^2 &= \frac{1}{T} \sum_{t=0}^{T-1} \{x_i(t) - \mu_i\}^2 \\ &= \frac{1}{T} \left\{ \sum_{t=0}^{T-1} \{x_i(t) - \mu_i\}^2 - \frac{1}{T} \left\{ \sum_{t=0}^{T-1} \{x_i(t) - \mu_i\} \right\}^2 \right\}. \end{aligned}$$

The statistic unnormalizer:

$$x_i(t) = \bar{x}_i(t)\sigma_i + \mu_i.$$

多次元領域の最大二点間距離による正規化と復元化

The domain normalizer:

$$\bar{\chi}(t) = \frac{\chi(t)}{\|\chi_{\max} - \chi_{\min}\|_2},$$

where $0 \leq t < T$ and

$$\chi_{\min} = (\min\{x_1(t)\}, \dots, \min\{x_d(t)\})^T,$$

$$\chi_{\max} = (\max\{x_1(t)\}, \dots, \max\{x_d(t)\})^T.$$

The domain unnormalizer:

$$\chi(t) = \bar{\chi}(t) \|\chi_{\max} - \chi_{\min}\|_2$$

The inter-point normalizer:

$$\bar{\chi}(t) = \frac{\chi(t) - \chi(t_i)}{\|\chi(t_i) - \chi(t_j)\|_2},$$

where $0 \leq t < T$ and

$$t_i = i, t_j = j \text{ when } \max \|\chi(i) - \chi(j)\|$$

The inter-point unnormalizer:

$$\chi(t) = \bar{\chi}(t)\|\chi(t_i) - \chi(t_j)\|_2 + \chi(t_i)$$

補遺B

使用しているコンテナ

`std::vector<T> ...` 標準テンプレートライブラリの配列コンテナ。力学系の状態などで使用。四則演算などの演算子は、`vector_numerics.hpp` で自前で用意（なぜなら、小さいサイズの`ublas::vector<T>`や`ublas::matrix<T>`などは、むしろ効率が悪いからである）。入出力は `vector_io.hpp` で自前で用意。型Tには`double`や`long double`等が指定される。

`boost::multi_array<T, 2> ...` Boost C++ ライブラリの多次元配列コンテナ。時系列データや埋め込みデータなどで使用。入出力は `multi_array_io.hpp` で自前で用意。型Tには`double`や`long double`等が指定される。

`std::pair<std::vector<T>, std::vector<T>> ...` 標準テンプレートライブラリのペアコンテナ。学習データセットの要素で使用。型Tには`double`や`long double`等が指定される。

`ublas::vector<T> ...` Boost C++ ライブラリの uBLAS(`usual basic linear algebra subroutines`) のベクトルコンテナ。基本線形代数がサポートされている。効率が良くなる大きいサイズのときに使用。

`ublas::matrix<T> ...` Boost C++ ライブラリの uBLAS(`usual basic linear algebra subroutines`) の行列コンテナ。基本線形代数がサポートされている。効率が良くなる大きいサイズのときに使用。

`ublas::matrix<T, ublas::column_major> ...` 同上だが、Boost Numeric Bindings ライブラリによる LAPACK の利用に必要な、メモリレイアウトに関する特殊化を指定している。

使用しているアルゴリズム

`std::partial_sort ...` 標準部分ソートアルゴリズム。2点間距離の配列から、その添字の配列をソートするために使用。上位Mまでしか必要ないので、部分ソートの方が標準ソート`std::sort`よりも高速。

`ublas::lu_factorize, ublas::lu_substitute ...` uBLAS同梱のLU分解と後方代入。

lapack::gesdd ... LAPACKの特異値分解(SVD: singular value decomposition)の uBLAS へのバインディング。最小二乗法による関数近似に利用。

lapack::gelsd ... LAPACKの特異値分解を用いた最小二乗法の uBLAS へのバインディング。 $Ax=B$ の最小二乗解を求めるのに利用。

参考文献

1. 合原 編, 池口, 山田, 小室 著「カオス時系列解析の基礎と応用」産業図書, 2002.
2. 山田, 高橋, 合原「地域風況の予測技術と風力発電」日本信頼性学会誌, Vol.28, No.6, 2006年11月.
3. 合原 編「カオスセミナー」海文堂, 1994.