6 Emulating Principal Components

This section is incomplete, and the PCA software is not yet fully tested.

6.1 Overview

Rather than emulating all observables, it can be more efficient to emulate a handful of principal components. After generating the training-point data, one can run the `pca_calctransformation` program included with the distribution. This will create files that shadow those used to emulate the observables. This will create a file `PCA_Info/observable_info.txt` which shadows the `Info/observable_info.txt`. The difference is that the observables will be named `z1,z2···`. In each run directory, alongside the `obs.txt` files, there will be a `obs_pca.txt` file. Finally, there will be a file `PCA_Info/tranformation_info.txt` file that contains all the information and matrices required to perform the basis transformation. If the parameter `Use_PCA` is set to `true`, the emulator will use the PCA files above instead of the observable files. The emulator will then store the Taylor coefficients in the directory `coefficients_pca/` rather than in `coefficients/`.

6.2 Compiling and Running the PCA Programs

To get an idea of the capabilities and functionality of the PCA elements of the *Smooth Emulator* Distribution, one can view the sample main program,
`${MY_LOCAL}/main_programs/pca_calctransformation_main.cc`:

```
int main() {
CparameterMap parmap;
parmap.ReadParsFromFile("parameters/emulator_parameters.txt");
  NBandSmooth::PCA *pca = new NBandSmooth::PCA(&parmap);
  pca->CalcTransformationInfo();
}
```

To compile the program,

```
${MY_LOCAL}/main_programs% make pca_calctransformation
```

Before running one needs to have set up the usual files defining the training points, e.g. running *Simplex Sampler*, and running the full model at the training points. One can now run the program that creates files containing all the observable information, but translated into PCA components:

```
${MY_PROJECT}% pca_calctransformation
```

One can check the directory `${MY_PROJECT}/PCA_Info/` to make sure that one sees the files `experimental_info.txt`, `observable_info.txt` and `transformation_info.txt`. The latter file includes the transformation information so that one can readily translate from the nominal observables to the PCA components.

Before performing the tuning, one needs to edit the `parameters/emulator_parameters.txt` file and change the parameter `SmoothEmulator_UsePCA` to `true`. The output when tuning the emulator should look something like:

```
${MY_PROJECT}% smoothy_tune
 ---- Tuning for z0 ----
 ---- Tuning for z1 ----
 ---- Tuning for z2 ----
 ---- Tuning for z3 ----
 ---- Tuning for z4 ----
```

⋮ The Taylor-expansion coefficients for the PCA observables are in the directory `${MY_PROJECT}/coefficients_pca/`.

Another example program for PCA analysis is `${MY_LOCAL}/main_programs/pca_readinfo_calcy_main.cc`:

```
 int main() {
CparameterMap parmap;
parmap.ReadParsFromFile("parameters/emulator_parameters.txt");
  NBandSmooth::PCA *pca = new NBandSmooth::PCA(&parmap);
  pca->ReadTransformationInfo();
vector<double> Z,Y,SigmaZ_emulator,SigmaY_emulator;
Z.resize(pca->Nobs);
Y.resize(pca->Nobs);
SigmaZ_emulator.resize(pca->Nobs);
SigmaY_emulator.resize(pca->Nobs);
printf("---- Start with these values of Z ----\n");
for(unsigned int iobs=0;iobs<pca->Nobs;iobs++){
Z[iobs]=iobs;
printf("Z[%u]=%g\n",iobs,Z[iobs]);
}
printf("---- Translated values of Y ----\n");
SigmaZ_emulator[0]=SigmaZ_emulator[1]=SigmaZ_emulator[2]=SigmaZ_emulator[3]=SigmaZ_emulator

pca->TransformZtoY(Z,SigmaZ_emulator,Y,SigmaY_emulator);
for(unsigned int iobs=0;iobs<pca->Nobs;iobs++){
printf("%10.5f %10.5f\n",Y[iobs],SigmaY_emulator[iobs]);
}
printf("---- (Re)Translated values of Z ----\n");
pca->TransformYtoZ(Y,SigmaY_emulator,Z,SigmaZ_emulator);
for(unsigned int iobs=0;iobs<pca->Nobs;iobs++){
printf("%10.5f %10.5f\n",Z[iobs],SigmaZ_emulator[iobs]);
}
printf("---- Retranslated values of Z should match original and SigmaZ should all be unity
 }
```

This program illustrates how one can insert the PCA transformations into another program. After defining the `pca` object, the program reads in the transformation information with the `pca->ReadTransformati` command. The remainder of the program simply test the transformation by making a fake vector of the $\boldsymbol{Z}$ (PCA) components, then translating them to $\boldsymbol{Y}$ (observable) values and then back. The original values for $\boldsymbol{Z[i]}$ are set to $\boldsymbol{0, 1, 2, \cdots}$, which should then match the values after translating back and forth as illustrated.

To compile the program:

```
${MY_LOCAL}/main_programs% make pca_readinfo_calcy
```

Running the program:

```
${MY_PROJECT}% pca_readinfo_calcy
 ---- Start with these values of Z ----
Z[0]=0
Z[1]=1
Z[2]=2
Z[3]=3
Z[4]=4
Z[5]=5
---- Translated values of Y and SigmaY ----
  -2.09833   24.99999
  -1.03694   33.33329
  -0.26087   41.66668
  -5.36371    0.16667
   3.49439    0.08333
   2.91093    0.08333
---- (Re)Translated values of Z and SigmaZ ----
   0.00000    1.00000
   1.00000    1.00000
   2.00000    1.00000
   3.00000    1.00000
   4.00000    1.00000
   5.00000    1.00000
---- Retranslated values of Z should match original and SigmaZ should all be unity ----
```

Note that for the PCA observables the scaling ensures they all have the same uncertainty, 1.0.

6.3 PCA Parameters (not model parameters!)

The PCA programs uses parameter that are prefixed with **SmoothEmulator**. One would typically use the same parameter file as used for running *Smooth Emulator*. The relevant parameters are:

1. **SmoothEmultor_UsePCA**
   If one wishes to emulate the PCA observables, i.e. those that are linear combinations of the real observables, this should be set to true. One must then be sure to have run the PCA decomposition programs first.

2. **SmoothEmulator_ModelRunDirName** and **SmoothEmulator_TrainingPts** should be set the same as used by *Smooth Emulator*.