Elementary Operations with Polynomial Chaos Expansions

UQTk Example

Example description

- Command-line usage:

  ```
  ./Ops.x
  ```

  followed by

  ```
  ./plot_pdf.py samples.a.dat
  ./plot_pdf.py samples.loga.dat
  ```

  to plot select probability distributions based on samples from Polynomial Chaos Expansions (PCE) utilized in this example.

- For a description of PCEs see Lecture #1 and the paper:

  [BD05]: “Numerical challenges in the use of polynomial chaos representations for stochastic processes”, by B.J Debusschere, H.N Najm, P.P Pébay, O.M Knio, R.G Ghanem, O.P.L Maître. *Siam J Sci Comput,* 2005, vol. 26(2) pp.698-719

Ops.x step-by-step

- Wherever relevant the PCSet class implements functions that take either "double *" arguments or array container arguments. The array containers, named "Array1D", "Array2D", and"Array3D", respectively, are provided with the UQTk library to streamline the management of data structures.

1. Instantiate a PCSet class for a 2nd order 1D PCE using Hermite-Gauss chaos.

```
int ord = 2;
int dim = 1;
PCSet myPCSet("ISP",ord,dim,"HG");
```

2. Initialize coefficients for HG PCE expansion $\hat{a}$ given its mean and standard deviation:

```
double ma = 2.0; // Mean
double sa = 0.1; // Std Dev
myPCSet.InitMeanStDv(ma,sa,a);
```

$$\hat{a} = \sum_{k=0}^{P} a_k \Psi_k(\xi), \quad a_0 = \mu, \quad a_1 = \frac{\sigma}{\sqrt{\langle \psi_1^2 \rangle}}, \quad a_2 = a_3 = \ldots = 0$$

Ops.x step-by-step

3. Initialize $\hat{b} = 2.0\psi_0(\xi) + 0.2\psi_1(\xi) + 0.01\psi_2(\xi)$ and subtract $\hat{b}$ from $\hat{a}$:

```
b[0] = 2.0;
b[1] = 0.2;
b[2] = 0.01;
myPCSet.Subtract(a,b,c);
```

   The subtraction is a term by term operation: $c_k = a_k - b_k$

4. Product of PCE's, $\hat{c} = \hat{a} \cdot \hat{b}$:

```
myPCSet.Prod(a,b,c);
```

$$\hat{c} = \sum_{k=0}^{P} c_k \Psi_k(\xi) = \left(\sum_{k=0}^{P} a_k \Psi_k(\xi)\right) \left(\sum_{k=0}^{P} b_k \Psi_k(\xi)\right)$$

$$c_k = \sum_{i=0}^{P} \sum_{j=0}^{P} C_{ijk} a_i b_j, \quad C_{ijk} = \frac{\langle \psi_i \psi_j \psi_k \rangle}{\langle \psi_k^2 \rangle}$$

   The triple product C_{ijk} is computed and stored when the PCSet class is instantiated.

Ops.x step-by-step

5. Exponential of a PCE, $\hat{c} = \exp(\hat{a})$ is computed using a Taylor series approach

```
myPCSet.Exp(a,c);
```

$$\hat{c} = \exp(\hat{a}) = \exp(a_0)\left(1 + \sum_{n=0}^{N_T} \frac{\hat{d}^n}{n!}\right) \tag{1}$$

where

$$\hat{d} = \hat{a} - a_0 = \sum_{k=1}^{P} a_k \tag{2}$$

The number of terms N_T in the Taylor series expansion are incremented adaptively until an error criterion is met (relative magnitude of coefficients compared to the mean) or the maximum number of terms is reached. Currently, the default relative tolerance and maximum number of Taylor terms are 10^{-6} and 500. This values can be changed by the user using public PCSet methods `SetTaylorTolerance` and `SetTaylorTermsMax`, respectively.

Ops.x step-by-step

6. Division, $\hat{c} = \hat{a}/\hat{b}$:

   ```
   myPCSet.Div(a,b,c);
   ```

   Internally the division operation is cast as a linear system, see item 4, $\hat{a} = \hat{b} \cdot \hat{c}$, with unknown coefficients c_k and known coefficients a_k and b_k. The linear system is sparse and it is solved with a GMRES iterative solver provided by NETLIB

7. Natural logarithm, $\hat{c} = \log(\hat{a})$:

   ```
   myPCSet.Log(a,c);
   ```

   Currently, two methodologies are implemented to compute the logarithm of a PCE: Taylor series expansion and an integration approach. For more details see **[BD05]**

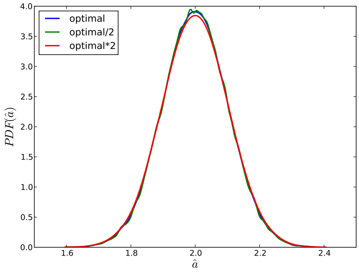
8. Draw samples from the random variable $\hat{a}$ represented as a PCE:

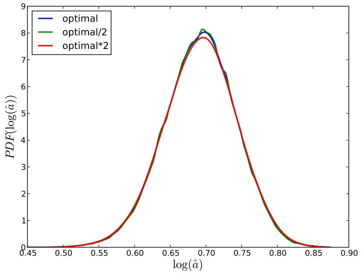
   ```
   myPCSet.DrawSampleSet(aa,aa_samp);
   ```

   Currently "Ops.x" draws sample from both $\hat{a}$ and $\log(\hat{a})$ and saves the results to files "samples.a.dat" and "samples.loga.dat", respectively.

Ops.x step-by-step

9. The directory contains a python script that computes probability distributions from samples via Kernel Density Estimate (KDE, also see Lecture #1) and generates two plots, “samples.a.dat.pdf” and “samples.loga.dat.pdf”, also shown below:





(results in these figures are generated using several KDE bandwidths. This feature is available in the python’s scipy package starting with version 0.11)