Polynomial chaos expansions part I: Method Introduction

Jonathan Feinberg and Simen Tennøe

Kalkulo AS

January 15, 2015

Lecture will include many examples using the Chaospy software

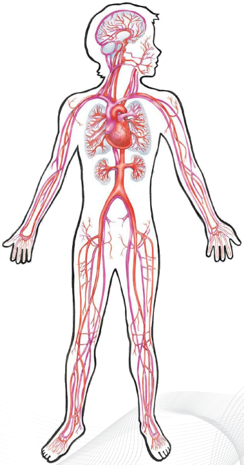
Installation instructions:

`http://github.com/hplgit/chaospy/`

Interactive sessions:

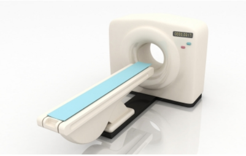
`path/to/ipython/notebook/sessions`

Practicle application involving bloodflow simulations



In colaboration with V. Eck and L. Hellevik

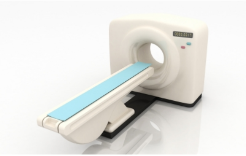
Modelling require uncertainty quantification



STARFiSh
STochastic ARterial Flow Simulations

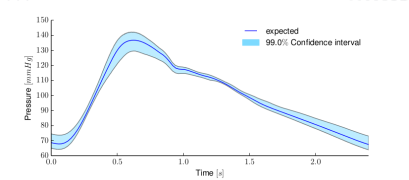
Chaos Py

Modelling require uncertainty quantification



STARFiSh
STochastic ARterial Flow Simulations

Chaos Py



ntnu/results/sensitivity-point

Introducing a naive testcase as a working example

Introducing a naive testcase as a working example

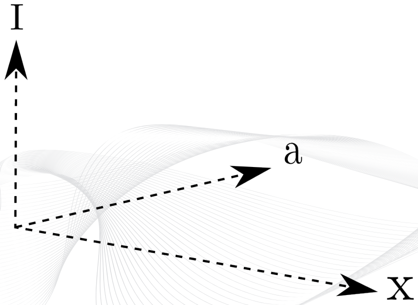
$$\frac{du(x)}{dx} = -au(x) \qquad\qquad u(0) = I$$

- u The quantity of interest
- x Spatio-temperal locations
- a, I Parameters containting uncertainties

Introducing a naive testcase as a working example

$$\frac{du(x)}{dx} = -au(x) \qquad\qquad u(0) = I$$

u The quantity of interest
x Spatio-temperal locations
a, I Parameters containting uncertainties



Trivial models can be analysed analytically

$$u(x; a, I) = Ie^{-ax}$$

Trivial models can be analysed analytically

$$u(x; a, I) = Ie^{-ax}$$

Initially assume model parameters:

$$a \sim \text{Uniform}(0, 0.1) \qquad\qquad I = 1$$

Trivial models can be analysed analytically

$$u(x; a, I) = Ie^{-ax}$$

Initially assume model parameters:

$$a \sim \text{Uniform}(0,\ 0.1) \qquad\qquad I = 1$$

$$E(u) = \int_0^{0.1} e^{-ax} \frac{1}{10} da = -\frac{1}{10x}\left(e^{-0.1x} - 1\right) = \frac{1 - e^{-0.1x}}{10x}$$

Trivial models can be analysed analytically

$$u(x; a, I) = Ie^{-ax}$$

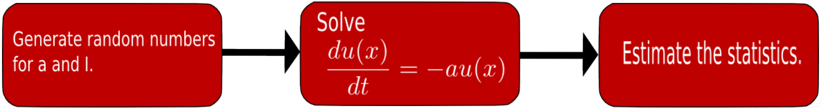
Initially assume model parameters:

$$a \sim \text{Uniform}(0,\ 0.1) \qquad\qquad I = 1$$

$$E(u) = \int_0^{0.1} e^{-ax}\frac{1}{10}da = -\frac{1}{10x}\left(e^{-0.1x} - 1\right) = \frac{1 - e^{-0.1x}}{10x}$$

$$Var(u) = \int_0^{0.1} \left(e^{-ax}\right)^2\frac{1}{10}da - E(u)^2 = \frac{1 - e^{-0.2ax}}{20x} - \left(\frac{1 - e^{-0.1x}}{10x}\right)^2$$

Non-trivial models can be analysed using Monte Carlo integration



Monte Carlo with chaospy

```
import chaospy as cp
import numpy as np

def u(x, a):
  return np.exp(-a*x)
```

Monte Carlo with chaospy

```
import chaospy as cp
import numpy as np

def u(x, a):
  return np.exp(-a*x)


a = cp.Uniform(0.0.1)

samples_a = a.sample(1000)
```

Monte Carlo with chaospy

```
import chaospy as cp
import numpy as np

def u(x, a):
  return np.exp(-a*x)


a = cp.Uniform(0.0.1)

samples_a = a.sample(1000)

x = np.linspace(0, 10, 100)
```

Monte Carlo with chaospy

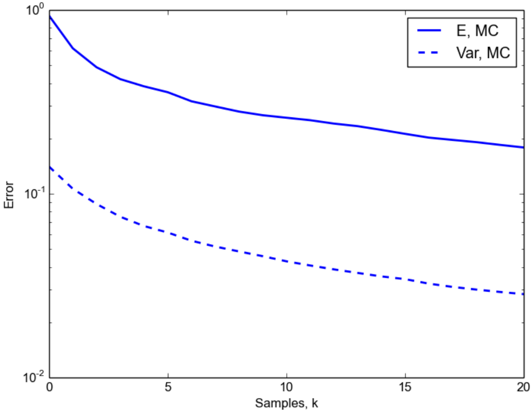
```
import chaospy as cp
import numpy as np

def u(x, a):
  return np.exp(-a*x)


a = cp.Uniform(0.0.1)

samples_a = a.sample(1000)

x = np.linspace(0, 10, 100)

U = [u(x,q) for q in samples_a]
```

Monte Carlo with chaospy

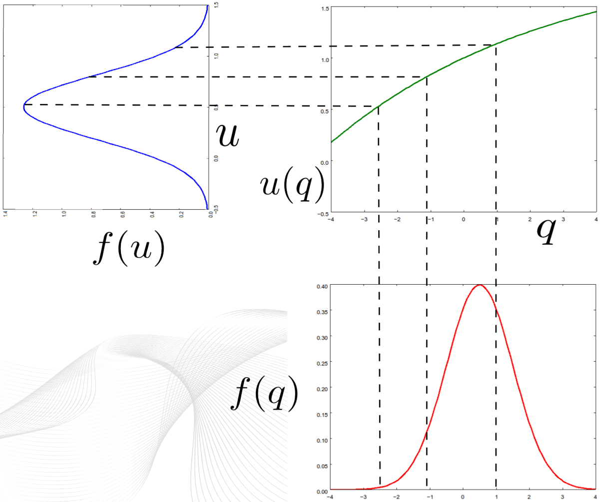
```
import chaospy as cp
import numpy as np

def u(x, a):
    return np.exp(-a*x)


a = cp.Uniform(0.0.1)

samples_a = a.sample(1000)

x = np.linspace(0, 10, 100)

U = [u(x,q) for q in samples_a]

E = np.mean(U)
Var = np.var(U)
```

Convergence of Monte Carlo is slow

$$\varepsilon_E = \int_0^{10} |E(u) - E(\hat{u})| \, dx \quad \varepsilon_{Var} = \int_0^{10} |Var(u) - Var(\hat{u})| \, dx$$



Monte Carlo is based on the idea of indirect sampling



Using Lagrange polynomials to approximate the model

$$u(x;a) \approx \hat{u}_M(x;a) = \sum_{n=0}^{N} c_n(x) P_n(a) \qquad N = M + 1,$$

where

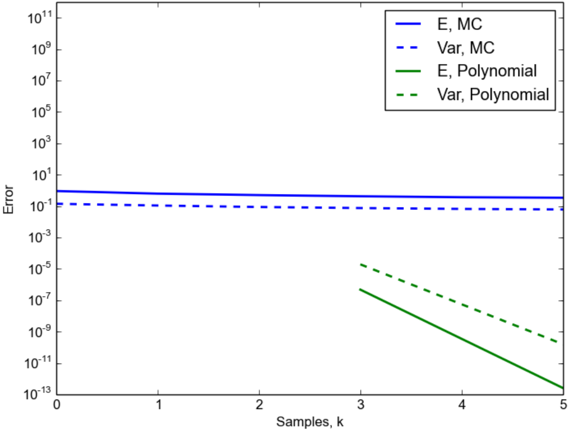
c_n are model evaluations $u(x, a_n)$

P_n are Lagrange polynomials:

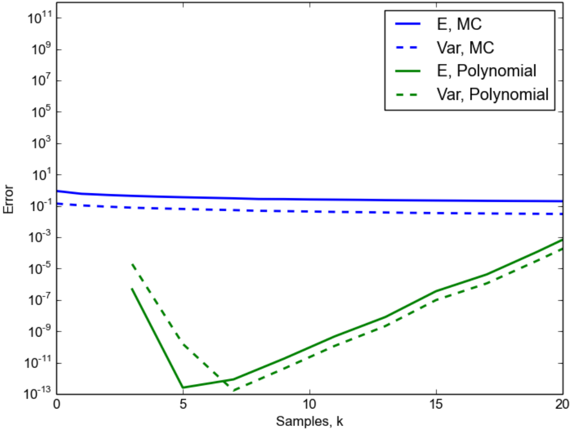
$$P_n(a) = \prod_{\substack{m=0 \\ m \neq n}}^{N} \frac{a - a_n}{a_m - a_n}$$

a_n are collocation nodes

Much better convergence properties than Monte Carlo



Stochastic analysis of model approximations can be dangerous



Combining polynomial approximation with uncertainty quantification well requires a better underlying theory

$$\langle u, v \rangle_Q = E(u \cdot v) \qquad \|u\|_Q = \sqrt{\langle u, u \rangle_Q}$$

where Q is a random vector, i.e. (a, I).

Combining polynomial approximation with uncertainty quantification well requires a better underlying theory

$$\langle u, v\rangle_Q = E(u \cdot v) \qquad \|u\|_Q = \sqrt{\langle u, u\rangle_Q}$$
$$= \int f_Q(q)u(x,q)v(x,q)dq$$

where Q is a random vector, i.e. (a, I).

Combining polynomial approximation with uncertainty quantification well requires a better underlying theory

$$\langle u, v \rangle_Q = E(u \cdot v) \qquad \|u\|_Q = \sqrt{\langle u, u \rangle_Q}$$
$$= \int f_Q(q) u(x, q) v(x, q) dq$$

where Q is a random vector, i.e. (a, I).

Orthogonality:

$$\langle P_n, P_m \rangle = \begin{cases} \|P_n\|_Q^2 & n = m \\ 0 & n \neq m \end{cases}$$

Coefficients are Fourier when polynomials are orthogonal

$$u = \sum_{n=0}^{N} c_n P_n$$

Coefficients are Fourier when polynomials are orthogonal

$$u = \sum_{n=0}^{N} c_n P_n$$

$$\langle u, P_k \rangle_Q = \left\langle \sum_{n=0}^{N} c_n P_n, P_k \right\rangle_Q \qquad k = 0, \ldots, N$$

Coefficients are Fourier when polynomials are orthogonal

$$
\begin{aligned}
u &= \sum_{n=0}^{N} c_n P_n \\
\langle u, P_k \rangle_Q &= \left\langle \sum_{n=0}^{N} c_n P_n, P_k \right\rangle_Q \\
&= \sum_{n=0}^{N} c_n \langle P_n, P_k \rangle_Q \qquad\qquad k = 0, \dots, N
\end{aligned}
$$

Coefficients are Fourier when polynomials are orthogonal

$$
\begin{aligned}
u &= \sum_{n=0}^{N} c_n P_n \\
\langle u, P_k \rangle_Q &= \left\langle \sum_{n=0}^{N} c_n P_n, P_k \right\rangle_Q \\
&= \sum_{n=0}^{N} c_n \langle P_n, P_k \rangle_Q \\
&= c_k \langle P_k, P_k \rangle_Q \qquad\qquad k = 0, \ldots, N
\end{aligned}
$$

Coefficients are Fourier when polynomials are orthogonal

$$u = \sum_{n=0}^{N} c_n P_n$$

$$\begin{aligned}
\langle u, P_k \rangle_Q &= \left\langle \sum_{n=0}^{N} c_n P_n, P_k \right\rangle_Q \\
&= \sum_{n=0}^{N} c_n \langle P_n, P_k \rangle_Q \\
&= c_k \langle P_k, P_k \rangle_Q
\end{aligned}$$

$$c_k = \frac{\langle u, P_k \rangle_Q}{\| P_k \|_Q^2} \qquad k = 0, \ldots, N$$

Optimality linked to statistical property

$$(c_0, \ldots, c_N) = \underset{c_0,\ldots,c_N}{\operatorname{argmin}} \|u - \hat{u}_M\|_Q$$

Optimality linked to statistical property

$$(c_0, \dots, c_N) = \underset{c_0,\dots,c_N}{\operatorname{argmin}} \|u - \hat{u}_M\|_Q$$

$$= \underset{c_0,\dots,c_N}{\operatorname{argmin}} \|u - \hat{u}_M\|_Q^2$$

Optimality linked to statistical property

$$(c_0, \ldots, c_N) = \operatorname*{argmin}_{c_0,\ldots,c_N} \|u - \hat{u}_M\|_Q$$

$$= \operatorname*{argmin}_{c_0,\ldots,c_N} \|u - \hat{u}_M\|_Q^2$$

$$= \operatorname*{argmin}_{c_0,\ldots,c_N} \mathrm{E}\big((u - \hat{u}_M)^2\big)$$

Optimality linked to statistical property

$$
\begin{aligned}
(c_0, \ldots, c_N) &= \operatorname*{argmin}_{c_0,\ldots,c_N} \|u - \hat{u}_M\|_Q \\
&= \operatorname*{argmin}_{c_0,\ldots,c_N} \|u - \hat{u}_M\|_Q^2 \\
&= \operatorname*{argmin}_{c_0,\ldots,c_N} \mathrm{E}\big((u - \hat{u}_M)^2\big) \\
&= \operatorname*{argmin}_{c_0,\ldots,c_N} \mathrm{Var}(u - \hat{u}_M)
\end{aligned}
$$

The mean and variance has a simpler form

The mean and variance has a simpler form

Assumption: $P_0 = 1$

The mean and variance has a simpler form

Assumption: $P_0 = 1$

$$\mathrm{E}(\hat{u}_M) = \mathrm{E}\left(\sum_{n=0}^{N} c_n P_n\right)$$

The mean and variance has a simpler form

Assumption: $P_0 = 1$

$$\mathrm{E}(\hat{u}_M) = \mathrm{E}\left(\sum_{n=0}^{N} c_n P_n\right)$$
$$= \sum_{n=0}^{N} c_n \mathrm{E}(P_n)$$

The mean and variance has a simpler form

Assumption: $P_0 = 1$

$$\mathrm{E}(\hat{u}_M) = \mathrm{E}\left(\sum_{n=0}^{N} c_n P_n\right)$$

$$= \sum_{n=0}^{N} c_n \mathrm{E}(P_n)$$

$$= \sum_{n=0}^{N} c_n \left\langle P_n, P_0 \right\rangle_Q$$

The mean and variance has a simpler form

Assumption: $P_0 = 1$

$$\mathrm{E}(\hat{u}_M) = \mathrm{E}\left(\sum_{n=0}^{N} c_n P_n\right)$$
$$= \sum_{n=0}^{N} c_n \mathrm{E}(P_n)$$
$$= \sum_{n=0}^{N} c_n \left\langle P_n, P_0 \right\rangle_Q$$

$$\mathrm{E}(\hat{u}_M) = c_0$$

The mean and variance has a simpler form

Assumption: $P_0 = 1$

$$\mathrm{E}(\hat{u}_M) = \mathrm{E}\left(\sum_{n=0}^{N} c_n P_n\right) \qquad \mathrm{Var}(\hat{u}_M) = \mathrm{Var}\left(\sum_{n=0}^{N} c_n P_n\right)$$

$$= \sum_{n=0}^{N} c_n \mathrm{E}(P_n)$$

$$= \sum_{n=0}^{N} c_n \left\langle P_n, P_0 \right\rangle_Q$$

$$\mathrm{E}(\hat{u}_M) = c_0$$

The mean and variance has a simpler form

Assumption: $P_0 = 1$

$$\mathrm{E}(\hat{u}_M) = \mathrm{E}\left(\sum_{n=0}^{N} c_n P_n\right) = \sum_{n=0}^{N} c_n \mathrm{E}(P_n) = \sum_{n=0}^{N} c_n \langle P_n, P_0 \rangle_Q$$

$$\mathrm{Var}(\hat{u}_M) = \mathrm{Var}\left(\sum_{n=0}^{N} c_n P_n\right) = \sum_{\substack{n=0\\m=0}}^{N} c_n c_m \left(\mathrm{E}(P_n P_m) - \mathrm{E}(P_n)\mathrm{E}(P_m)\right)$$

$$\mathrm{E}(\hat{u}_M) = c_0$$

The mean and variance has a simpler form

Assumption: $P_0 = 1$

$$\mathrm{E}(\hat{u}_M) = \mathrm{E}\left(\sum_{n=0}^{N} c_n P_n\right) = \sum_{n=0}^{N} c_n \mathrm{E}(P_n) = \sum_{n=0}^{N} c_n \langle P_n, P_0\rangle_Q$$

$$\mathrm{Var}(\hat{u}_M) = \mathrm{Var}\left(\sum_{n=0}^{N} c_n P_n\right) = \sum_{\substack{n=0\\m=0}}^{N} c_n c_m \left(\mathrm{E}(P_n P_m) - \mathrm{E}(P_n)\mathrm{E}(P_m)\right) = \sum_{\substack{n=0\\m=0}}^{N} c_n c_m \langle P_n, P_m\rangle_Q - c_0^2$$

$$\mathrm{E}(\hat{u}_M) = c_0$$

The mean and variance has a simpler form

Assumption: $P_0 = 1$

$$\mathrm{E}(\hat{u}_M) = \mathrm{E}\left(\sum_{n=0}^{N} c_n P_n\right) \qquad \mathrm{Var}(\hat{u}_M) = \mathrm{Var}\left(\sum_{n=0}^{N} c_n P_n\right)$$

$$= \sum_{n=0}^{N} c_n \mathrm{E}(P_n) \qquad = \sum_{\substack{n=0\\m=0}}^{N} c_n c_m \left(\mathrm{E}(P_n P_m) - \mathrm{E}(P_n)\mathrm{E}(P_m)\right)$$

$$= \sum_{n=0}^{N} c_n \langle P_n, P_0 \rangle_Q \qquad = \sum_{\substack{n=0\\m=0}}^{N} c_n c_m \langle P_n, P_m \rangle_Q - c_0^2$$

$$\mathrm{E}(\hat{u}_M) = c_0 \qquad \mathrm{Var}(\hat{u}_M) = \sum_{n=1}^{N} c_n^2 \|P_n\|_Q$$

Construct an orthogonal polynomial expansion using Gram-Schmidt orthogonalization

$$v_0, v_1, \ldots, v_N = 1, q, \ldots, q^N$$

Construct an orthogonal polynomial expansion using Gram-Schmidt orthogonalization

$$v_0, v_1, ..., v_N = 1, q, ..., q^N$$

The Gram Schmidt method is

$$P_0 = v_0$$

Construct an orthogonal polynomial expansion using Gram-Schmidt orthogonalization

$$v_0, v_1, ..., v_N = 1, q, ..., q^N$$

The Gram Schmidt method is

$$P_0 = v_0$$

$$P_n = v_n - \sum_{m=0}^{n-1} \frac{\langle v_n, P_m \rangle_Q}{\|P_m\|_Q^2}$$

Construct an orthogonal polynomial expansion using Gram-Schmidt orthogonalization

$$v_0, v_1, ..., v_N = 1, q, ..., q^N$$

The Gram Schmidt method is

$$\begin{aligned} P_0 &= v_0 \\ P_n &= v_n - \sum_{m=0}^{n-1} \frac{\langle v_n, P_m \rangle_Q}{\|P_m\|_Q^2} \\ &= v_n - \sum_{m=0}^{n-1} \frac{\mathrm{E}(v_n P_m)}{\mathrm{E}(P_m^2)} \end{aligned}$$

Gram-Schmidt with chaospy

```
M = 5; D = 1; N = M - 1

a = cp.Uniform(0,0.1)
```

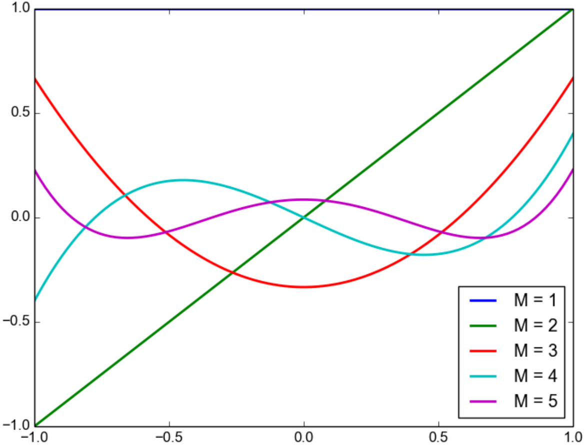
Gram-Schmidt with chaospy

```
M = 5; D = 1; N = M - 1

a = cp.Uniform(0,0.1)

v = cp.basis(0,M,1)
P = [v[0]]
```

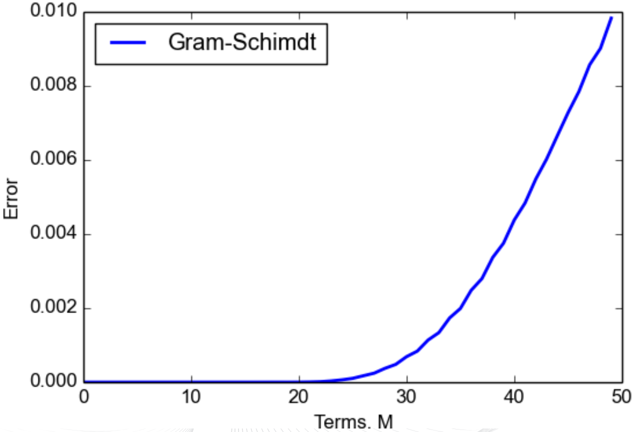
Gram-Schmidt with chaospy

```
M = 5; D = 1; N = M - 1

a = cp.Uniform(0,0.1)

v = cp.basis(0,M,1)
P = [v[0]]

for n in xrange(1,N):
    p = v[n]
    for m in xrange(0,n):
        p -= P[m]*cp.E(v[n]*P[m],a)
                            /cp.E(P[m]**2,a)
    P.append(p)
P = cp.Poly(P)
```

Plot of all generated polynomials



Most constructors of orthogonal polynomials are illposed



The only numerically stable method for calculating orthogonal polynomials is through the discretized Stiltjes method

The only numerically stable method for calculating orthogonal polynomials is through the discretized Stiltjes method

Three terms recursion relation:

$$P_{n+1} = (x - A_n)P_n - B_n P_{n-1} \qquad P_{-1} = 0 \qquad P_0 = 1,$$

The only numerically stable method for calculating orthogonal polynomials is through the discretized Stiltjes method

Three terms recursion relation:

$$P_{n+1} = (x - A_n)P_n - B_n P_{n-1} \qquad P_{-1} = 0 \qquad P_0 = 1,$$

where

$$A_n = \frac{\langle qP_n, P_n \rangle_Q}{\|P_n\|_Q^2} \qquad\qquad B_n = \begin{cases} \frac{\|P_n\|_Q^2}{\|P_{n-1}\|_Q^2} & n > 0 \\ \|P_n\|_Q^2 & n = 0 \end{cases}$$

Askey scheme, organization of orthogonal polynomials

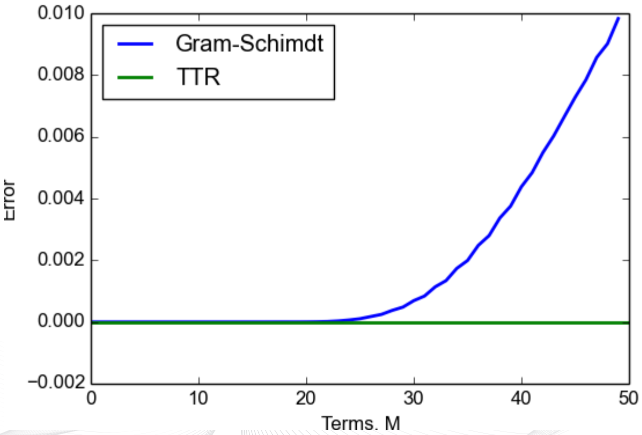
The askey scheme gives a relation between a probability distribution and the corresponding orhogonal polynomial.

Distribution	Polynomial	Support
Gaussian	Hermite	$(-\infty, \infty)$
Gamma	Laguerre	$[0, \infty]$
Beta	Jacobi	$[a, b]$
Uniform	Legendre	$[a, b]$

Three terms recursion in Chaospy

```
import chaospy as cp

M = 3
a = cp.Normal()
P = cp.orth_ttr(M, a)

print P

[1.0, q0, q0^2-1.0, q0^3-3.0q0]
```

Discretized Stiltjes method is numerically stable



Repetition of the problem

$$u(x; a, I) = Ie^{-ax}$$

Repetition of the problem

$$u(x; a, I) = Ie^{-ax}$$

Initially assume model parameters:

$$a \sim \text{Uniform}(0, 0.1) \qquad\qquad I = 1$$

Repetition of the problem

$$u(x; a, I) = Ie^{-ax}$$

Initially assume model parameters:

$$a \sim \text{Uniform}(0, 0.1) \qquad\qquad I = 1$$

$$E(u) = \frac{1 - e^{-0.1x}}{10x} \qquad Var(u) = \frac{1 - e^{-0.2ax}}{20x} - \left(\frac{1 - e^{-0.1x}}{10x}\right)^2$$

Repetition of the problem

$$u(x; a, I) = Ie^{-ax}$$

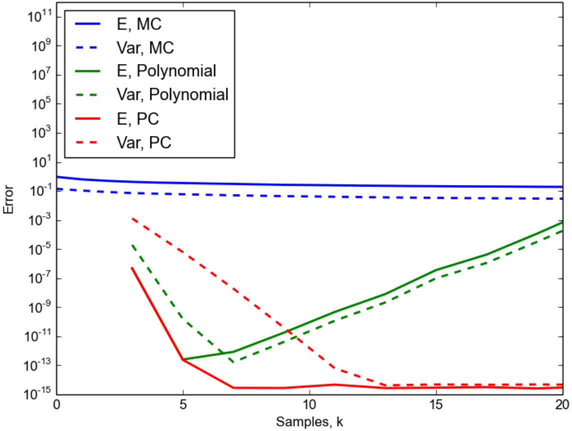
Initially assume model parameters:

$$a \sim \text{Uniform}(0,\ 0.1) \qquad\qquad I = 1$$

$$E(u) = \frac{1 - e^{-0.1x}}{10x} \qquad Var(u) = \frac{1 - e^{-0.2ax}}{20x} - \left(\frac{1 - e^{-0.1x}}{10x}\right)^2$$

$$\varepsilon_E = \int_0^{10} |E(u) - E(\hat{u})|\, dx \quad \varepsilon_{Var} = \int_0^{10} |Var(u) - Var(\hat{u})|\, dx$$

Convergence of orthogonal polynomial approximation



Extending the theory to multiple dimensions

Extending the theory to multiple dimensions

$$P_n = P^{(1)}_n, ..., P^{(D)}_{n_D} \qquad\qquad n \longleftrightarrow (n_1, ..., n_D)$$

Extending the theory to multiple dimensions

$$P_n = P_n^{(1)}, ..., P_{n_D}^{(D)} \qquad n \longleftrightarrow (n_1, ..., n_D)$$

$\cdots$						
$\mathbf{P}_4$	$\cdots$					
$\mathbf{P}_3$	$\mathbf{P}_3\mathbf{P}_0$	$\cdots$				
$\mathbf{P}_2$	$\mathbf{P}_2\mathbf{P}_0$	$\mathbf{P}_2\mathbf{P}_1$	$\cdots$			
$\mathbf{P}_1$	$\mathbf{P}_1\mathbf{P}_0$	$\mathbf{P}_1\mathbf{P}_1$	$\mathbf{P}_1\mathbf{P}_2$	$\cdots$		
$\mathbf{P}_0$	$\mathbf{P}_0\mathbf{P}_0$	$\mathbf{P}_0\mathbf{P}_1$	$\mathbf{P}_0\mathbf{P}_2$	$\mathbf{P}_0\mathbf{P}_3$	$\cdots$	
	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\cdots$

Construct polynomial approximation

Multi-index

$$\mathbf{P}_{00}$$
$$\mathbf{P}_{10} \quad \mathbf{P}_{01}$$
$$\mathbf{P}_{20} \quad \mathbf{P}_{11} \quad \mathbf{P}_{02}$$
$$\mathbf{P}_{30} \quad \mathbf{P}_{21} \quad \mathbf{P}_{12} \quad \dots$$

Single-index

$$\mathbf{P}_{0}$$
$$\mathbf{P}_{1} \quad \mathbf{P}_{2}$$
$$\mathbf{P}_{3} \quad \mathbf{P}_{4} \quad \mathbf{P}_{5}$$
$$\mathbf{P}_{6} \quad \mathbf{P}_{7} \quad \mathbf{P}_{8} \quad \dots$$

Orthogonality for multivariate polynomials

$$\langle \mathbf{P}_n, \mathbf{P}_m \rangle_Q = \mathrm{E}\Big(P_{n_1}^{(1)} \cdots P_{n_D}^{(D)} \cdot P_{m_1}^{(1)} \cdots P_{m_D}^{(D)} \Big)$$

Orthogonality for multivariate polynomials

$$\langle \mathbf{P}_n, \mathbf{P}_m \rangle_Q = \mathrm{E}\left(P^{(1)}_{n_1} \cdots P^{(D)}_{n_D} \cdot P^{(1)}_{m_1} \cdots P^{(D)}_{m_D} \right)$$

$$= \mathrm{E}\left(P^{(1)}_{n_1} \cdot P^{(1)}_{m_1} \right) \cdots \mathrm{E}\left(P^{(D)}_{n_D} \cdot P^{(D)}_{m_D} \right)$$

Orthogonality for multivariate polynomials

$$\langle \mathbf{P}_n, \mathbf{P}_m \rangle_Q = \mathrm{E}\left(P_{n_1}^{(1)} \cdots P_{n_D}^{(D)} \cdot P_{m_1}^{(1)} \cdots P_{m_D}^{(D)}\right)$$

$$= \mathrm{E}\left(P_{n_1}^{(1)} \cdot P_{m_1}^{(1)}\right) \cdots \mathrm{E}\left(P_{n_D}^{(D)} \cdot P_{m_D}^{(D)}\right)$$

$$= \left\langle P_{n_1}^{(1)} \cdot P_{m_1}^{(1)} \right\rangle_Q \cdots \left\langle P_{n_D}^{(D)} \cdot P_{m_D}^{(D)} \right\rangle_Q$$

Orthogonality for multivariate polynomials

$$\begin{aligned}
\langle \mathbf{P}_n, \mathbf{P}_m \rangle_Q &= \mathrm{E}\left(P^{(1)}_{n_1} \cdots P^{(D)}_{n_D} \cdot P^{(1)}_{m_1} \cdots P^{(D)}_{m_D} \right) \\
&= \mathrm{E}\left(P^{(1)}_{n_1} \cdot P^{(1)}_{m_1} \right) \cdots \mathrm{E}\left(P^{(D)}_{n_D} \cdot P^{(D)}_{m_D} \right) \\
&= \left\langle P^{(1)}_{n_1} \cdot P^{(1)}_{m_1} \right\rangle_Q \cdots \left\langle P^{(D)}_{n_D} \cdot P^{(D)}_{m_D} \right\rangle_Q \\
&= \left\| P^{(1)}_{n_1} \right\|_Q \delta_{n_1 m_1} \cdots \left\| P^{(D)}_{n_D} \right\|_Q \delta_{n_D m_D}
\end{aligned}$$

Orthogonality for multivariate polynomials

$$
\begin{aligned}
\langle \mathbf{P}_n, \mathbf{P}_m \rangle_Q &= \mathrm{E}\left(P_{n_1}^{(1)} \cdots P_{n_D}^{(D)} \cdot P_{m_1}^{(1)} \cdots P_{m_D}^{(D)} \right) \\
&= \mathrm{E}\left(P_{n_1}^{(1)} \cdot P_{m_1}^{(1)} \right) \cdots \mathrm{E}\left(P_{n_D}^{(D)} \cdot P_{m_D}^{(D)} \right) \\
&= \left\langle P_{n_1}^{(1)} \cdot P_{m_1}^{(1)} \right\rangle_Q \cdots \left\langle P_{n_D}^{(D)} \cdot P_{m_D}^{(D)} \right\rangle_Q \\
&= \left\| P_{n_1}^{(1)} \right\|_Q \delta_{n_1 m_1} \cdots \left\| P_{n_D}^{(D)} \right\|_Q \delta_{n_D m_D}
\end{aligned}
$$

$$
\langle \mathbf{P}_n, \mathbf{P}_m \rangle_Q = \| \mathbf{P}_n \|_Q \, \delta_{nm}
$$

Creating multivariate orthogonal polynomials in Chaospy

```
a = cp.Uniform(0, 0.1)
I = cp.Uniform(8, 10)
dist = cp.J(a,I)
```

Creating multivariate orthogonal polynomials in Chaospy

```python
a = cp.Uniform(0, 0.1)
I = cp.Uniform(8, 10)
dist = cp.J(a,I)

P = cp.orth_ttr(1, dist)
print P
[1.0, q1-9.0, q0]
```

Creating multivariate orthogonal polynomials in Chaospy

```
a = cp.Uniform(0, 0.1)
I = cp.Uniform(8, 10)
dist = cp.J(a,I)

P = cp.orth_ttr(1, dist)
print P
[1.0, q1-9.0, q0]

P = cp.orth_ttr(3, dist)
print cp.E(P[1]*P[2],dist)
0.0
```

Creating multivariate orthogonal polynomials in Chaospy

```
a = cp.Uniform(0, 0.1)
l = cp.Uniform(8, 10)
dist = cp.J(a,l)

P = cp.orth_ttr(1, dist)
print P
[1.0, q1-9.0, q0]

P = cp.orth_ttr(3, dist)
print cp.E(P[1]*P[2],dist)
0.0
print cp.E(P[3]*P[3],dist)
0.0888888888903
```

The multidimensional problrm

$$u(x; a, I) = Ie^{-ax}$$

The multidimensional problrm

$$u(x; a, I) = Ie^{-ax}$$

Initially assume model parameters:

$$a \sim \text{Uniform}(0,\ 0.1) \qquad\qquad I = \text{Uniform}(8,\ 10)$$

The multidimensional problrm

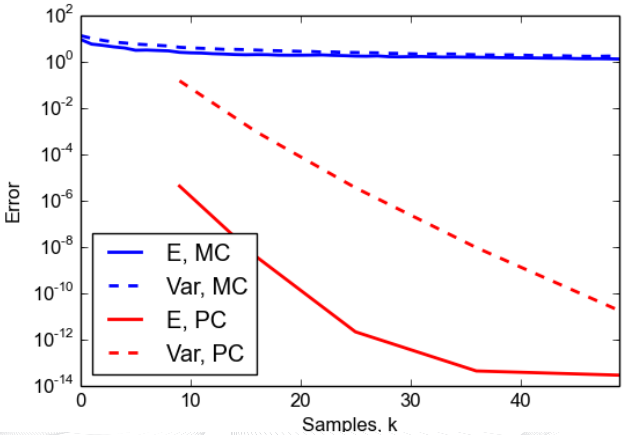
$$u(x; a, I) = Ie^{-ax}$$

Initially assume model parameters:

$$a \sim \text{Uniform}(0,\ 0.1) \qquad I = \text{Uniform}(8,\ 10)$$

$$\varepsilon_E = \int_0^{10} |E(u) - E(\hat{u})|\, dx \quad \varepsilon_{Var} = \int_0^{10} |Var(u) - Var(\hat{u})|\, dx$$

Convergence of a multidimensional problem



Teaser of the full implementation

```
def u(x,a, I):
    return I*np.exp(-a*x)
```

Teaser of the full implementation

```python
def u(x,a, I):
    return I*np.exp(-a*x)

a = cp.Uniform(0, 0.1)
I = cp.Uniform(8, 10)
dist = cp.J(a,I)
```

Teaser of the full implementation

```
def u(x,a, I):
    return I*np.exp(-a*x)

a = cp.Uniform(0, 0.1)
I = cp.Uniform(8, 10)
dist = cp.J(a,I)
```

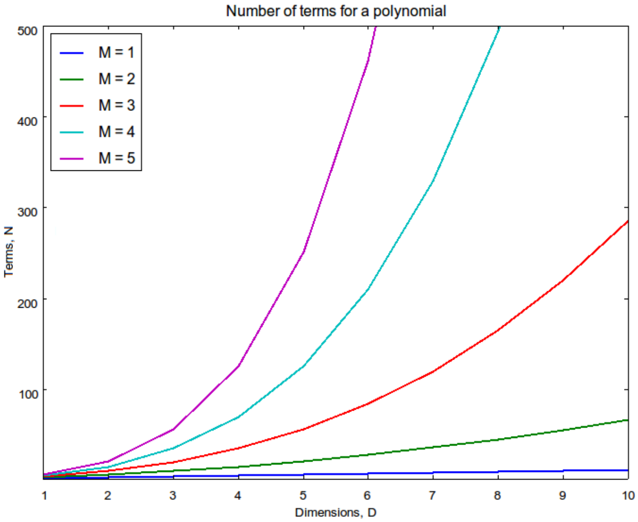
Teaser of the full implementation

```python
def u(x,a, I):
    return I*np.exp(-a*x)

a = cp.Uniform(0, 0.1)
I = cp.Uniform(8, 10)
dist = cp.J(a,I)

P = cp.orth_ttr(2, dist)
```

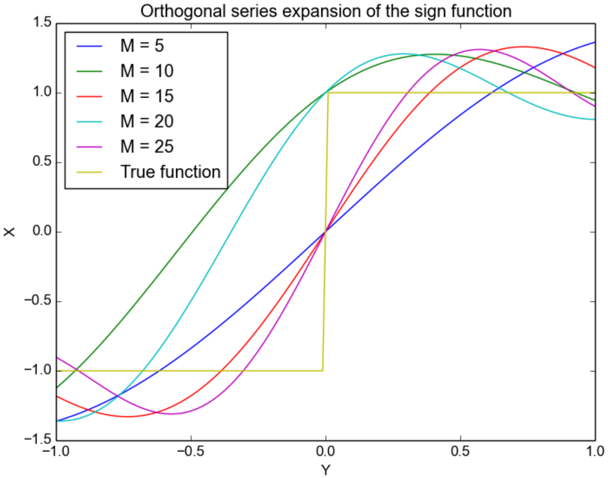
Teaser of the full implementation

```python
def u(x,a, I):
  return I*np.exp(-a*x)

a = cp.Uniform(0, 0.1)
I = cp.Uniform(8, 10)
dist = cp.J(a,I)

P = cp.orth_ttr(2, dist)

nodes, weights = \
    cp.generate_quadrature(3, dist, rule="G")

x = np.linspace(0, 10, 100)
solves = [u(x, *node) for node in nodes.T]

u_hat = cp.fit_quadrature(P, nodes, weights, solves)

mean, var = cp.E(u_hat, dist), cp.Var(u_hat, dist)
```

The curse of dimensionality



Gibb's Phenomena, discontinues methods are troublesome



Higher number of samples justifies higher number of collocation nodes