Polynomial chaos expansions part I: Method Introduction

by Jonathan Feinberg and Simen Tennøe

Kalkulo AS

March 4, 2015

Slides will include many examples using the Chaospy software

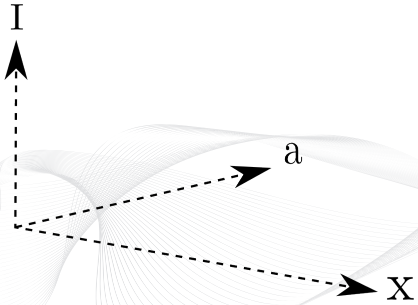
A very basic introduction to scientific Python programming:
http://hplgit.github.io/bumpy/doc/pub/sphinx-basics/index.html

Installation instructions:
https://github.com/hplgit/chaospy

Introducing a testcase as a working example

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

u The quantity of interest
x Spatial location
a, I Parameters containting uncertainties



This model can be analysed analytically

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

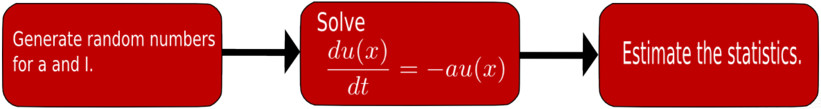
Initially assume model parameters:

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

$$\mathrm{E}(u) = \int_{-\infty}^{\infty} u(x; a) f_a(a) da = \int_0^{0.1} e^{-ax} \frac{1}{10} da = \frac{1 - e^{-0.1x}}{10x}$$

$$\mathrm{Var}(u) = \int_{-\infty}^{\infty} (u(x; a) - \mathrm{E}(u))^2 f_a(a) da = \frac{1 - e^{-0.2ax}}{20x} - \left(\frac{1 - e^{-0.1x}}{10x}\right)^2$$

In general, models can be analysed using Monte Carlo integration

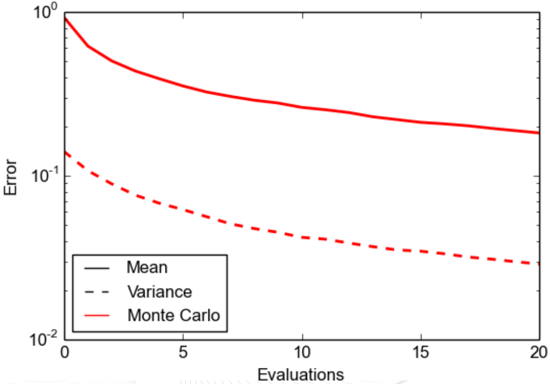


Monte Carlo with Chaospy

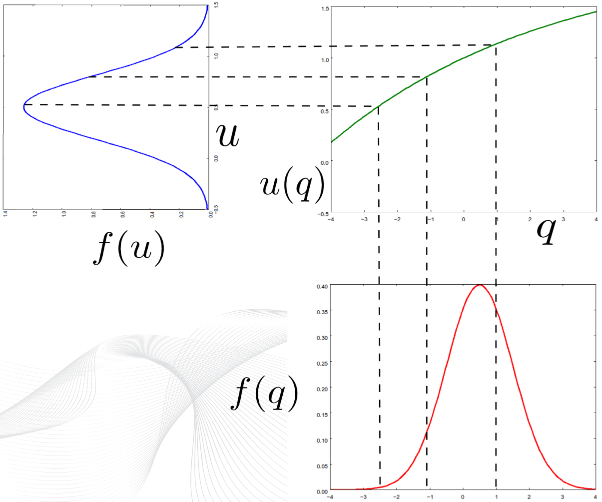
```
import chaospy as cp
import numpy as np

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


dist_a = cp.Uniform(0,0.1)

samples_a = dist_a.sample(size=1000)

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

samples_u = [u(x, a) for a in samples_a]

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

Convergence of Monte Carlo is slow

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



Assumption: mapping from input q to output u is smooth



Using Lagrange polynomials to approximate $u(q)$ (N-th degree polynomial interpolation)

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

where

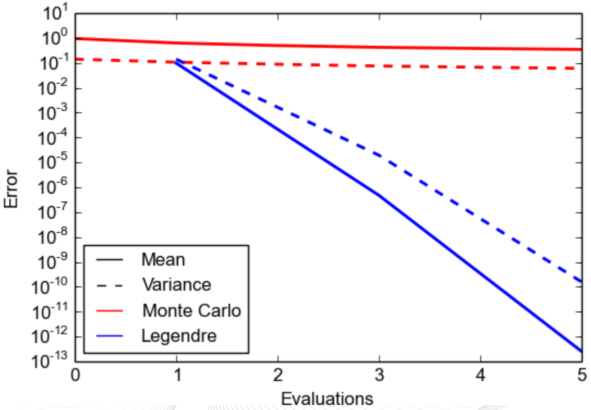
c_n are model evaluations $u(x, a_n)$

P_n are Lagrange polynomials:

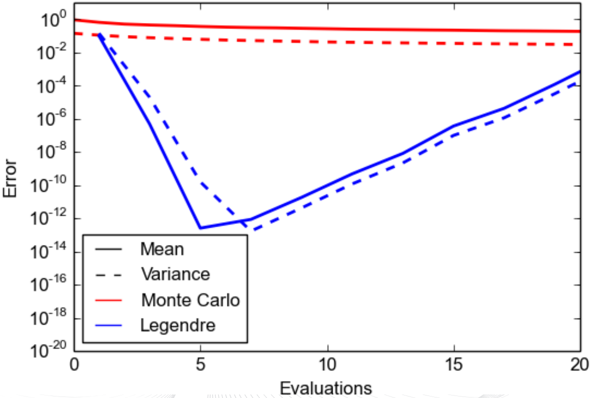
$$P_n(a) = \prod_{\substack{m=0 \\ m \neq n}}^{N} \frac{a - a_n}{a_m - a_n}$$

a_n are collocation nodes

The interpolation has much better convergence properties than Monte Carlo!



Oscillations in Lagrange polynomials (for large N) destroy the convergence



Let us introduce a better polynomial approximation: Polynomial Chaos (PC) theory

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

PC employs inner product spaces weighted with the probability distribution

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

$$= \int f_Q(q) u(x, q) v(x, q) dq$$

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

Orthogonality:

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

Coefficients are determined by least squares minimization

$$\min_{c_0,\dots,c_N} ||u - \hat{u}_M||_Q^2$$

$$\vdots$$

$$\left\langle \sum_{n=0}^{N} c_n P_n, P_k \right\rangle_Q = \sum_{n=0}^{N} c_n \langle P_n, P_k \rangle_Q = c_k \langle P_k, P_k \rangle_Q \quad k = 0, \dots, N$$

$$c_k = \frac{\langle u, P_k \rangle_Q}{\|P_k\|_Q^2}$$

Fourier coefficients

Least squares minimization implies minimization of variance

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

The mean and variance have a simpler form

Assumption: $P_0 = 1$

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

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

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

Construct an orthogonal polynomial expansion using Gram-Schmidt orthogonalization

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

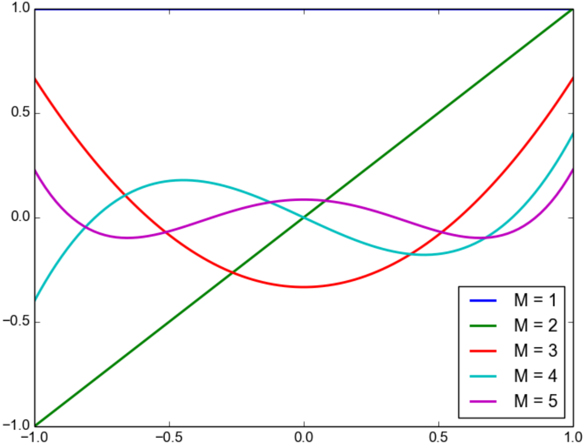
The Gram Schmidt method is

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

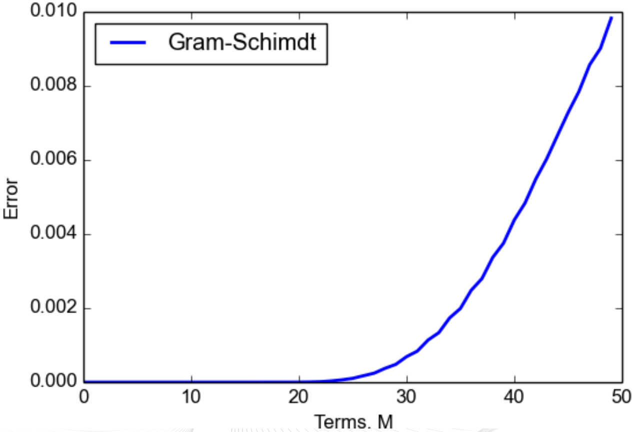
Gram-Schmidt with chaospy

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

dist_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], dist_a)
                                /cp.E(P[m]**2, dist_a)
    P.append(p)
P = cp.Poly(P)
```

Plot of all generated polynomials



Most constructors of orthogonal polynomials are ill-conditioned



The only numerically stable method for calculating orthogonal polynomials is through the three-term discretized Stiltjes recursion

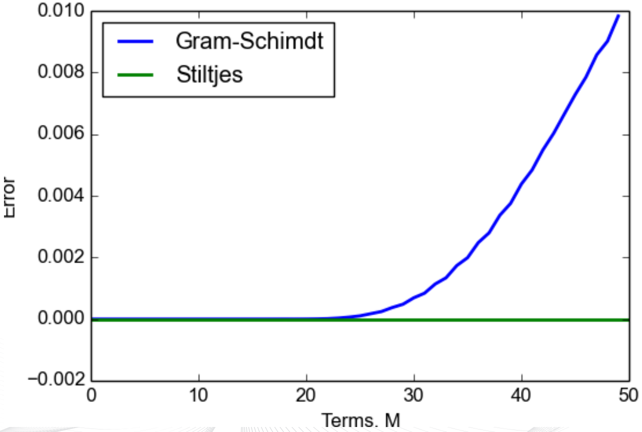
Three terms recursion relation:

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

where

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

Discretized Stiltjes method is numerically stable



People have found analytical orthogonal polynomials for many common probability distributions

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

Three terms recursion in Chaospy

```
dist_a = cp.Normal()
P = cp.orth_ttr(3, dist_a)

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

Repetition of the problem

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

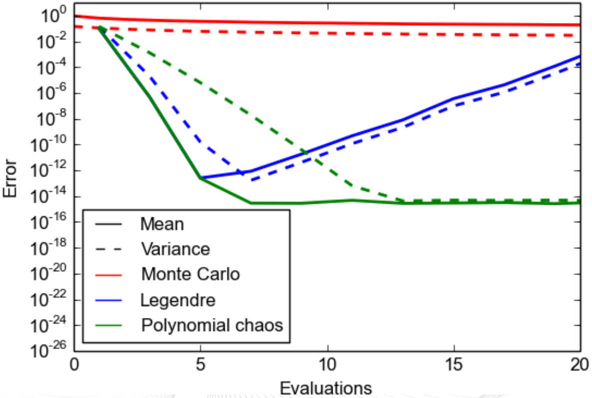
Initially assume model parameters:

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

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

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

Convergence of orthogonal polynomial approximation



Next step: Extend the theory to multiple dimensions

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

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

We need a mapping from multiple indices to single index

Multi-index

$$
\begin{array}{cccc}
 & & \mathbf{P}_{00} & \\
 & \mathbf{P}_{10} & \mathbf{P}_{01} & \\
\mathbf{P}_{20} & \mathbf{P}_{11} & \mathbf{P}_{02} & \\
\mathbf{P}_{30} & \mathbf{P}_{21} & \mathbf{P}_{12} & \dots
\end{array}
$$

Single-index

$$
\begin{array}{cccc}
 & & \mathbf{P}_{0} & \\
 & \mathbf{P}_{1} & \mathbf{P}_{2} & \\
\mathbf{P}_{3} & \mathbf{P}_{4} & \mathbf{P}_{5} & \\
\mathbf{P}_{6} & \mathbf{P}_{7} & \mathbf{P}_{8} & \dots
\end{array}
$$

$$N = \binom{M+D}{M}$$

Orthogonality for multivariate polynomials

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

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

Creating multivariate orthogonal polynomials in Chaospy

```
dist_a = cp.Uniform(0, 0.1)
dist_I = cp.Uniform(8, 10)
dist = cp.J(dist_a, dist_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
print cp.E(P[3]*P[3],dist)
0.0888888888903
```

A two-dimensional problem

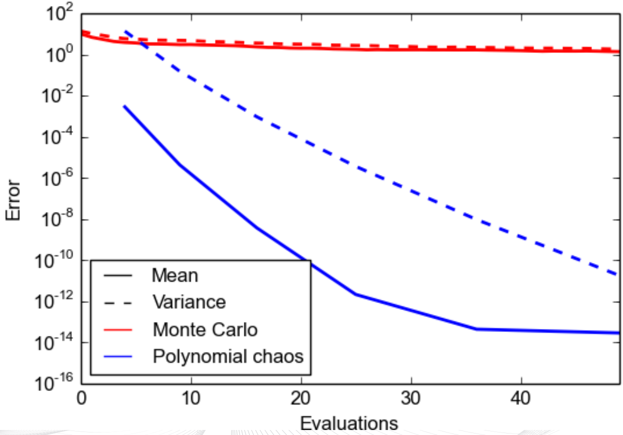
$$u(x; a, I) = Ie^{-ax}$$

Uncertain model parameters:

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

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

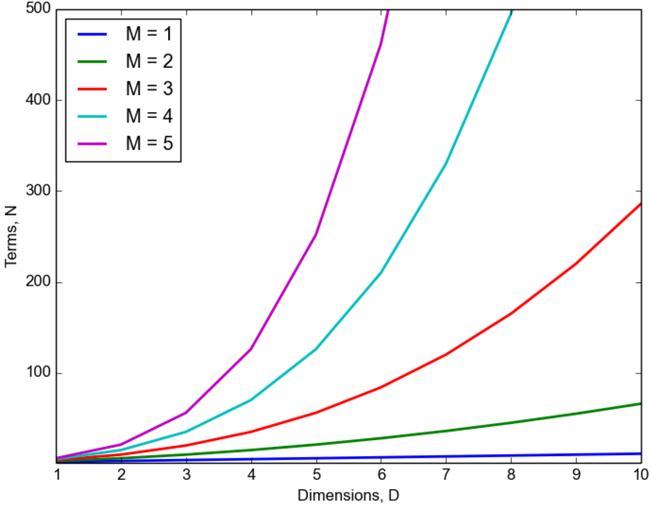
Convergence of the two-dimensional (a, l) problem



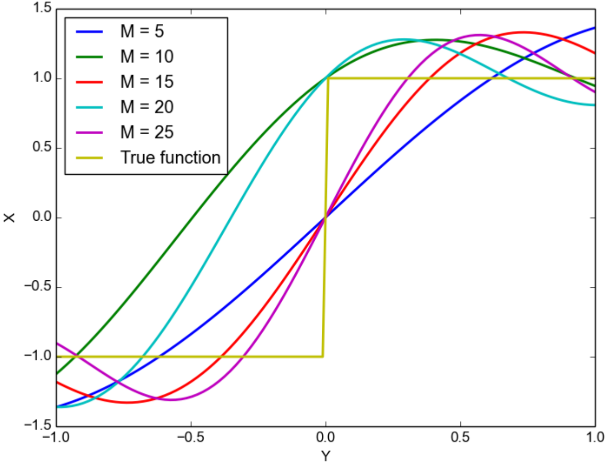
Teaser of the full implementation

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

dist_a = cp.Uniform(0, 0.1)
dist_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)
samples_u = [u(x, *node) for node in nodes.T]

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

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

The curse of dimensionality



Gibb's Phenomena: discontinuities give oscillations



Higher number of samples justifies higher number of collocation nodes