Uncertainty quantification using polynomial chaos expansion

Jonathan Feinberg

Kalkulo AS

March 4, 2015

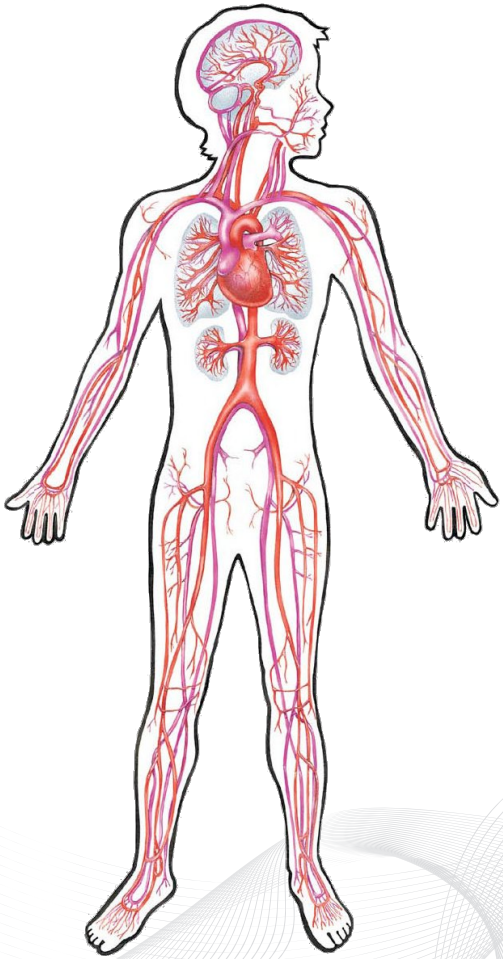
About me

UNIVERSITY
OF OSLO

[simula]

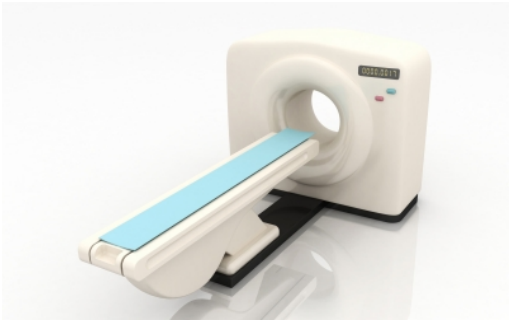
kalkulo

Example: bloodflow simulations



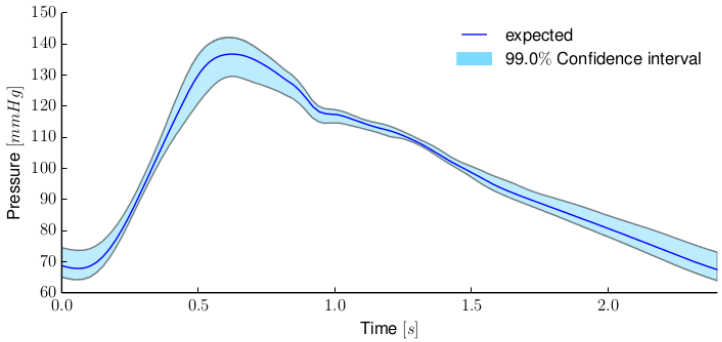
In colaboration with V. Eck and L. Hellevik

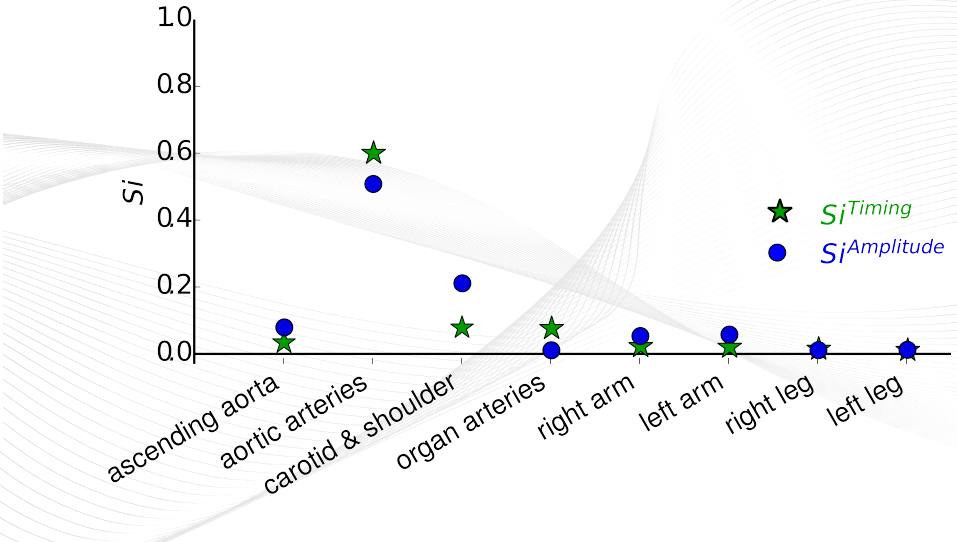
Modelling require uncertainty quantification



STARFiSh
STochastic ARterial Flow Simulations

ChaosPy





Introducing a testcase as a working example

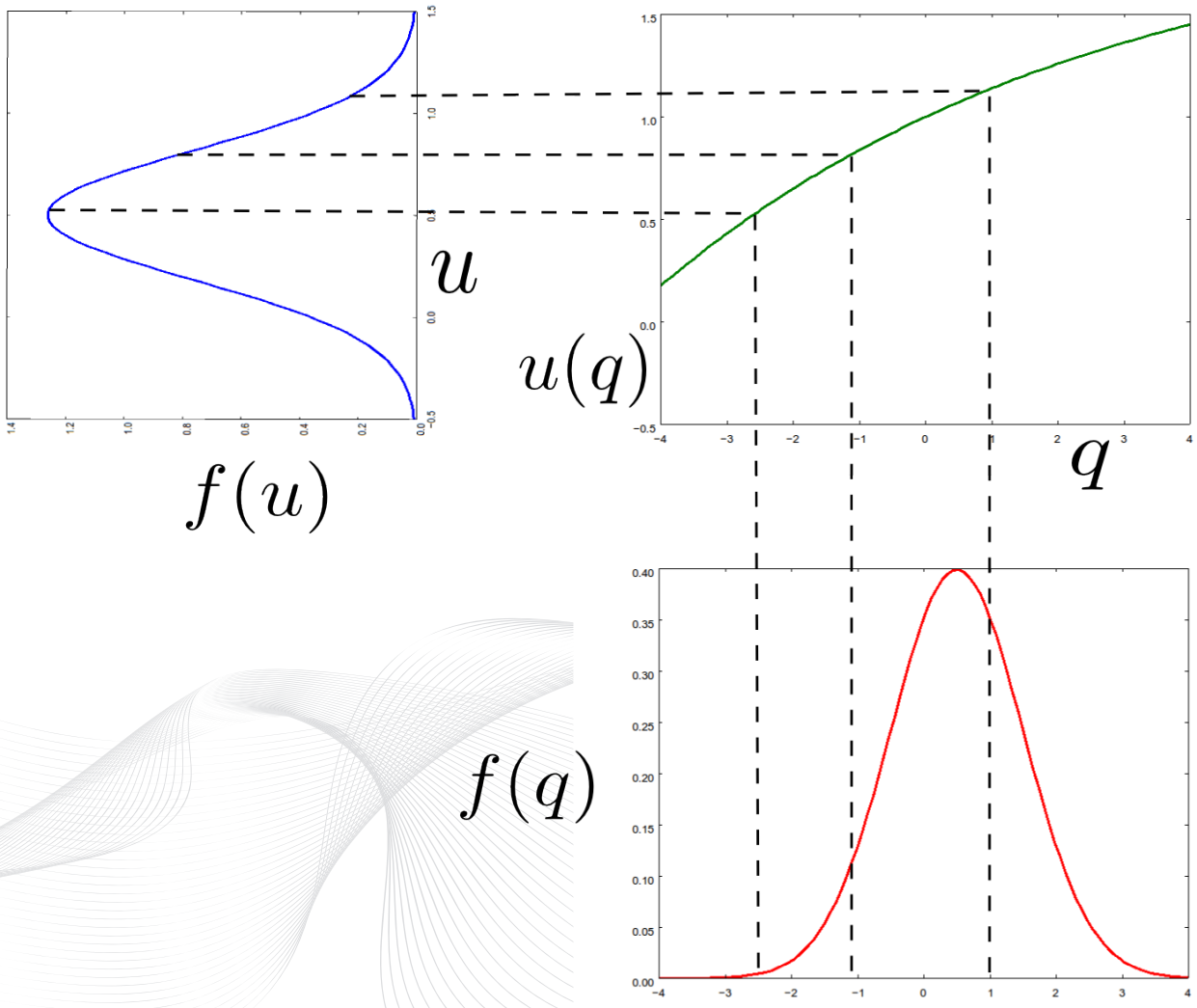
$$\frac{du(x)}{dx} = -q_0 u(x) \qquad\qquad u(0) = q_1$$

- u The quantity of interest
- x Spatio-temporal locations
- q Parameters containing uncertainties with probability density function $f(q)$

Closed form solution:

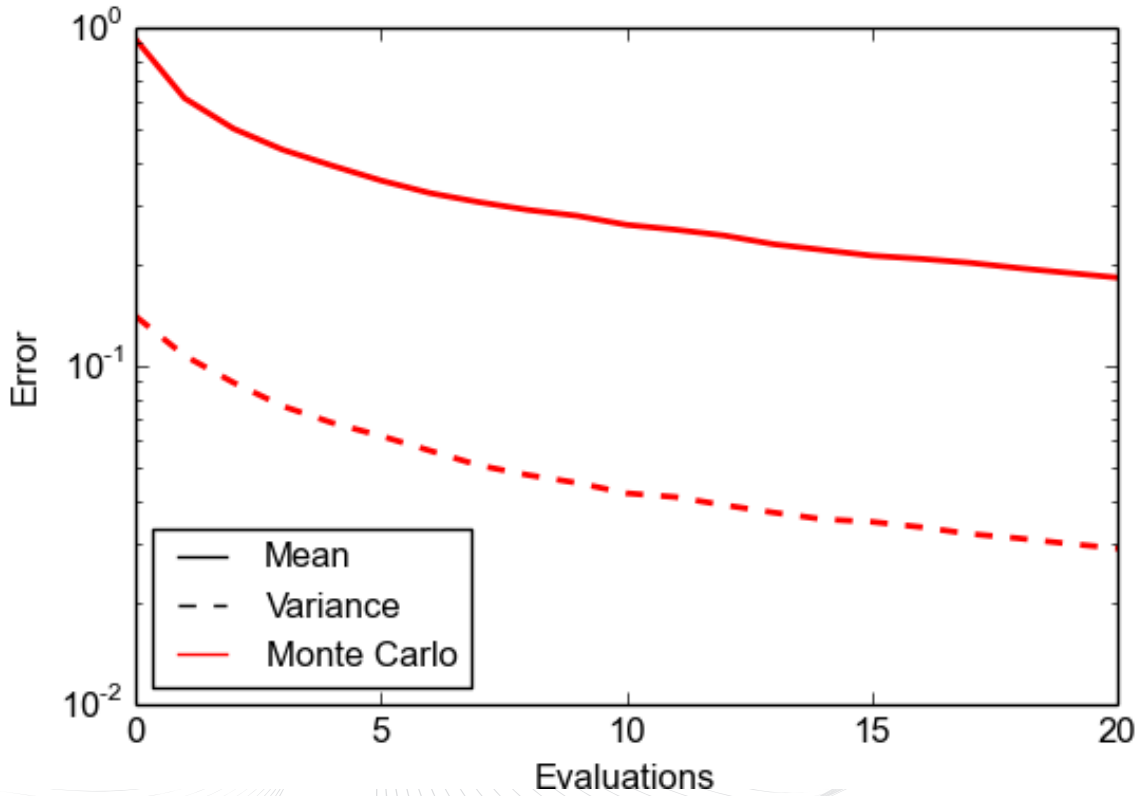
$$u(x;q) = q_1 e^{-q_0 x}$$

Monte Carlo integration is an indirect sampling scheme

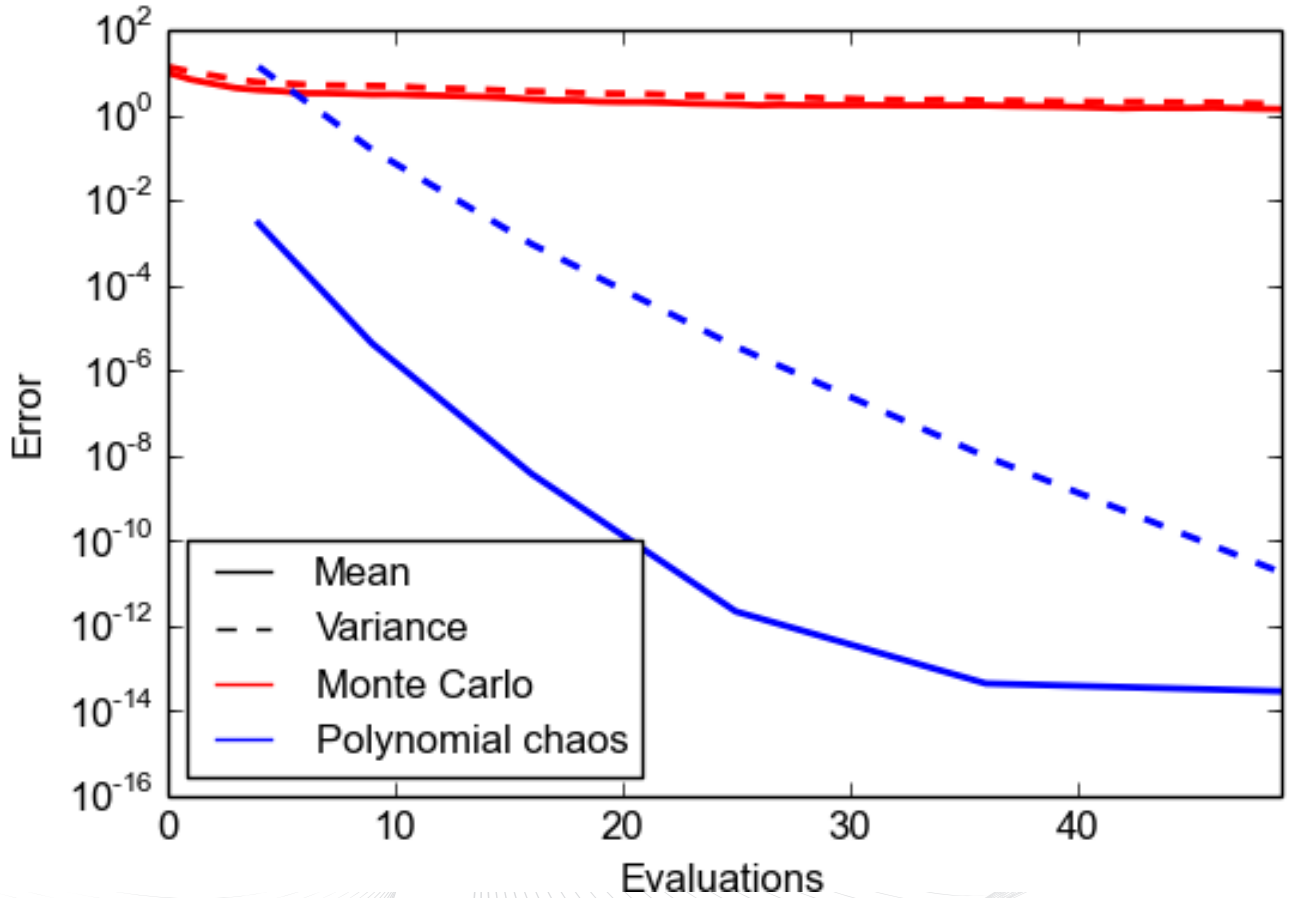


Convergence of Monte Carlo is very slow!

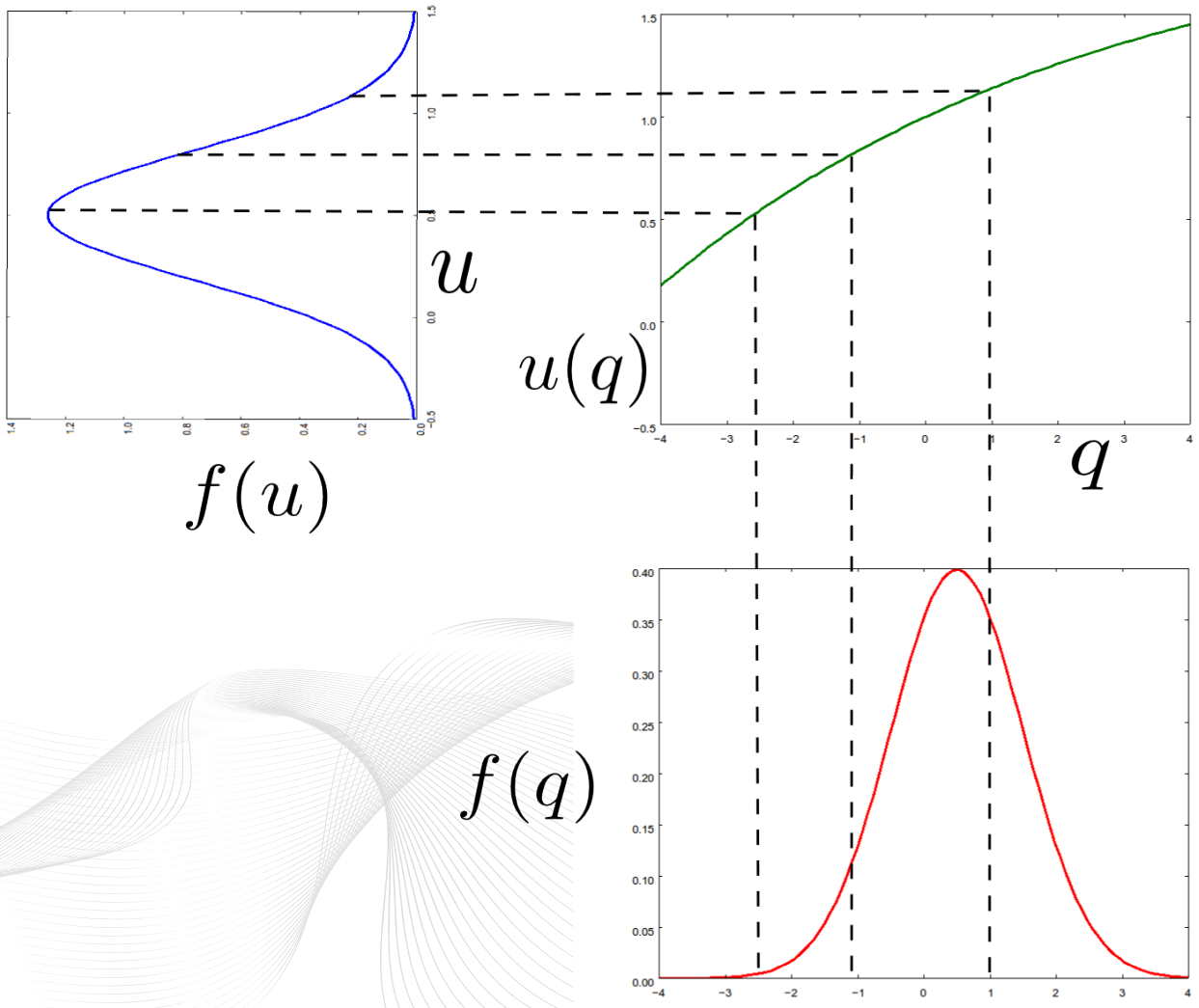
$$\varepsilon_E = \int_0^{10} |\mathrm{E}(u) - \mathrm{E}(\hat{u})|\, dx \quad \varepsilon_{Var} = \int_0^{10} |\mathrm{Var}(u) - \mathrm{Var}(\hat{u})|\, dx$$



Teaser: Convergence using polynomial chaos expansion is much faster



Assumption: mapping from input q to output u is smooth



Using polynomial approximation to create a surrogate model

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

M	Polynomial order
$N+1$	Number of polynomial terms
c_n	Coefficients
P_n	Polynomials

Polynomial chaos expansions are polynomial approximation that fitted using orthogonal polynomial basis

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

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

Orthogonality:

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

Chaos expansions have optimality criterion linked to minimal variance

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

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

Three terms recursion relation:

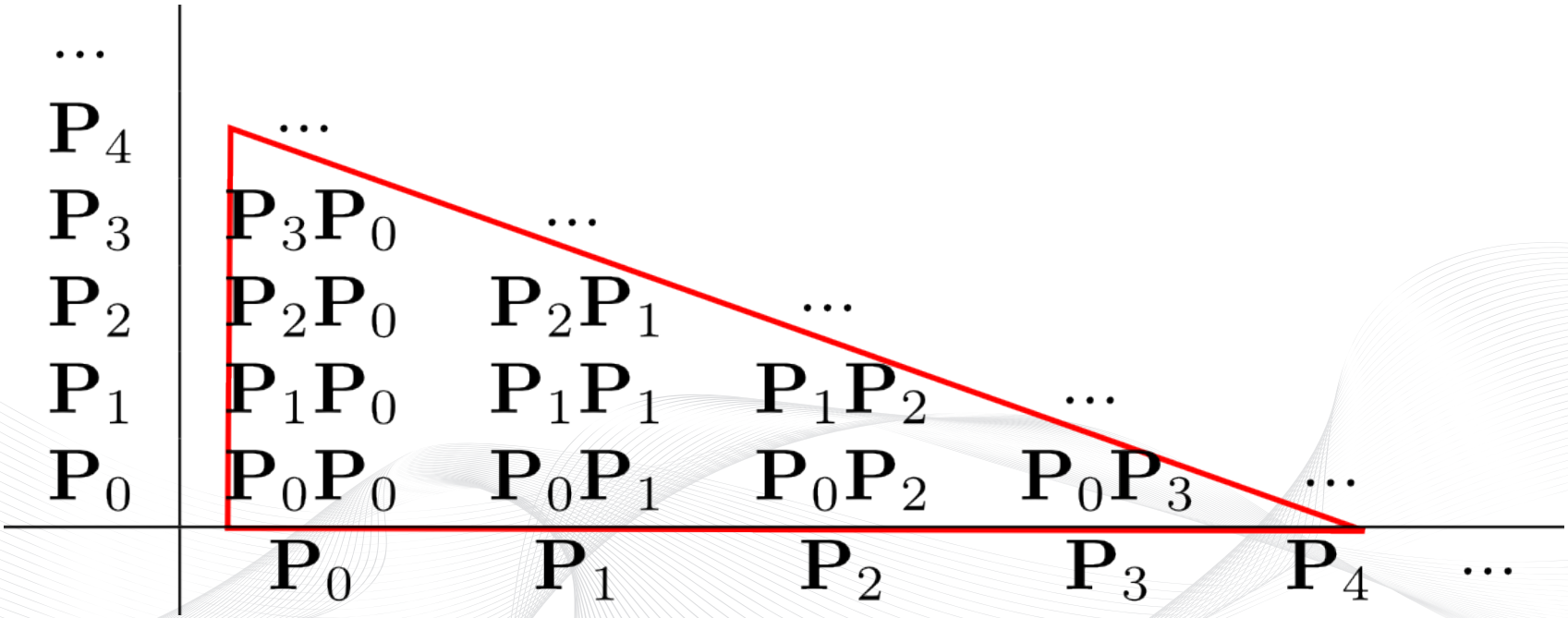
$$P_{n+1} = (x - A_n)P_n - B_n P_{n-1} \qquad P_{-1} = 0 \qquad P_0 = 1,$$

where

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

Using the theory on multiple dimensions

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



Mapping from multiple indices to single index to simplify notation

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

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

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

Orthogonality for multivariate polynomials

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

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

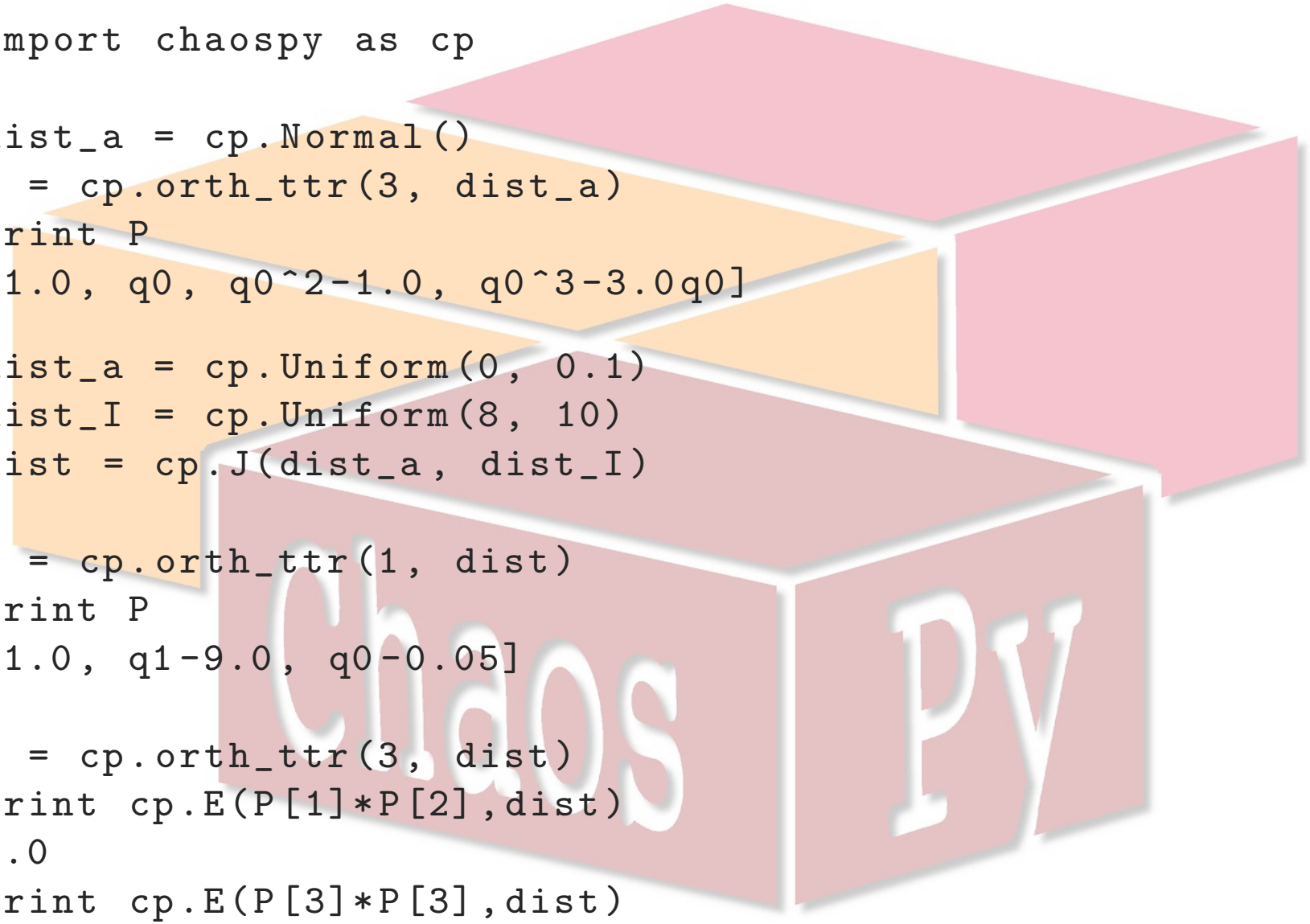
$$= \left\langle P_{n_1}^{(1)}, P_{m_1}^{(1)} \right\rangle_Q \cdots \left\langle P_{n_D}^{(D)}, P_{m_D}^{(D)} \right\rangle_Q$$

$$= \left\| P_{n_1}^{(1)} \right\|_Q \delta_{n_1 m_1} \cdots \left\| P_{n_D}^{(D)} \right\|_Q \delta_{n_D m_D}$$

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

Three terms recursion in Chaospy software

```
import chaospy as cp

dist_a = cp.Normal()
P = cp.orth_ttr(3, dist_a)
print P
[1.0, q0, q0^2-1.0, q0^3-3.0q0]

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, q0-0.05]

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
```



Coefficients are determined by least squares minimization

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

$$\vdots$$

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

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

Fourier coefficients

The computational essence of polynomial chaos

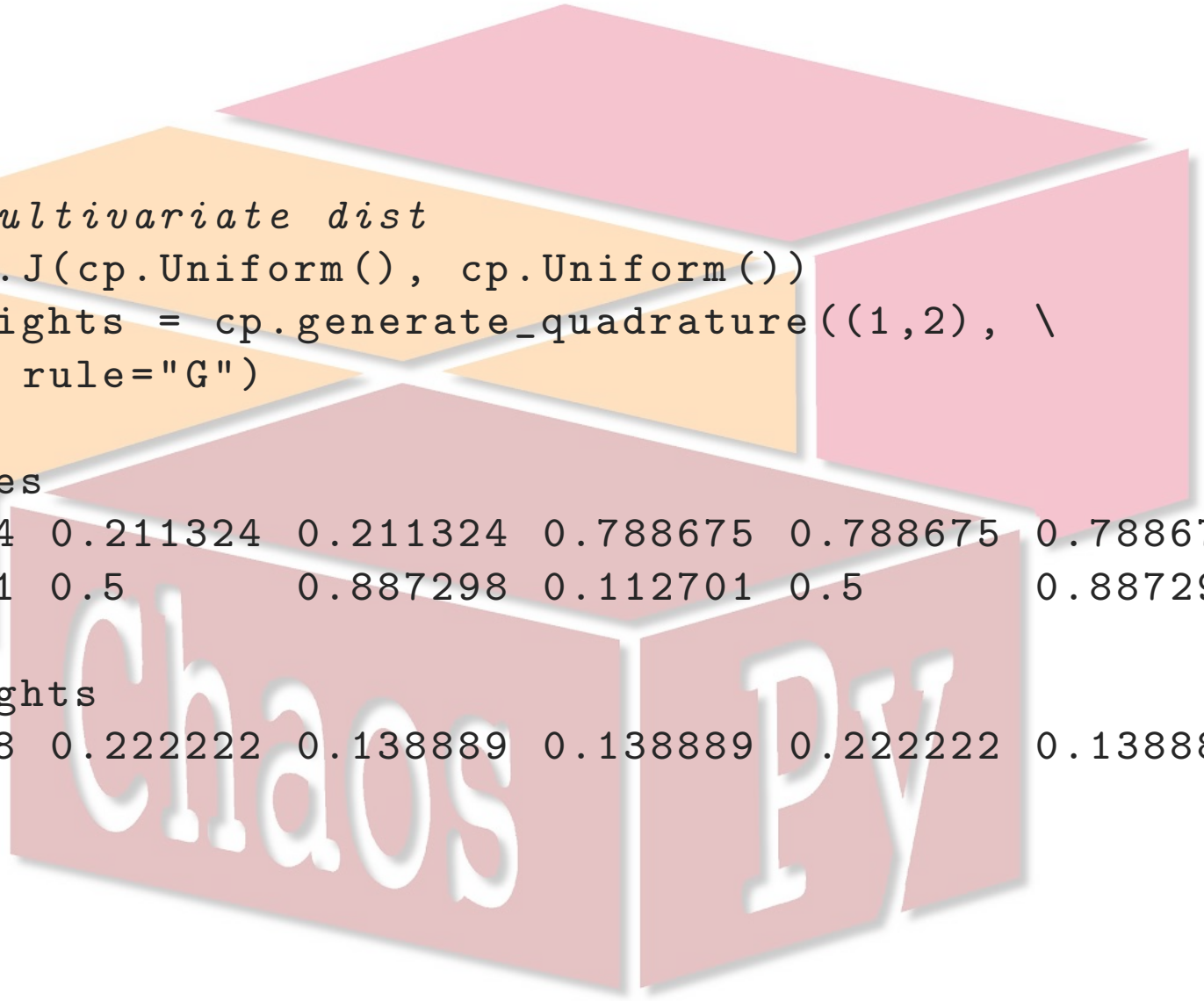
$$
\begin{aligned}
c_n(x) &= \frac{\langle u, P_n \rangle_Q}{\|P_n\|_Q^2} = \frac{\mathrm{E}(uP_n)}{\mathrm{E}(P_n^2)} \\
&= \frac{1}{\mathrm{E}(P_n^2)} \int u(x;q) P_n(q) f(q) dq \approx \\
\hat{c}_n(x) &= \frac{1}{\mathrm{E}(P_n^2)} \sum_{k=0}^{K} P_n(q_k) u(x;q_k) f(q_k) \omega_k
\end{aligned}
$$

The numerical integral approximation is named *pseudo-spectral method*.

q_k quadrature nodes, ω_k quadrature weights

Generating multivariate integration rules in Chaospy

```
# joint multivariate dist
dist = cp.J(cp.Uniform(), cp.Uniform())
nodes, weights = cp.generate_quadrature((1,2), \
    dist, rule="G")

print nodes
[[0.211324 0.211324 0.211324 0.788675 0.788675 0.788675]
 [0.112701 0.5      0.887298 0.112701 0.5      0.887298]]

print weights
 [0.138888 0.222222 0.138889 0.138889 0.222222 0.138889]
```



Returning to the simplified model

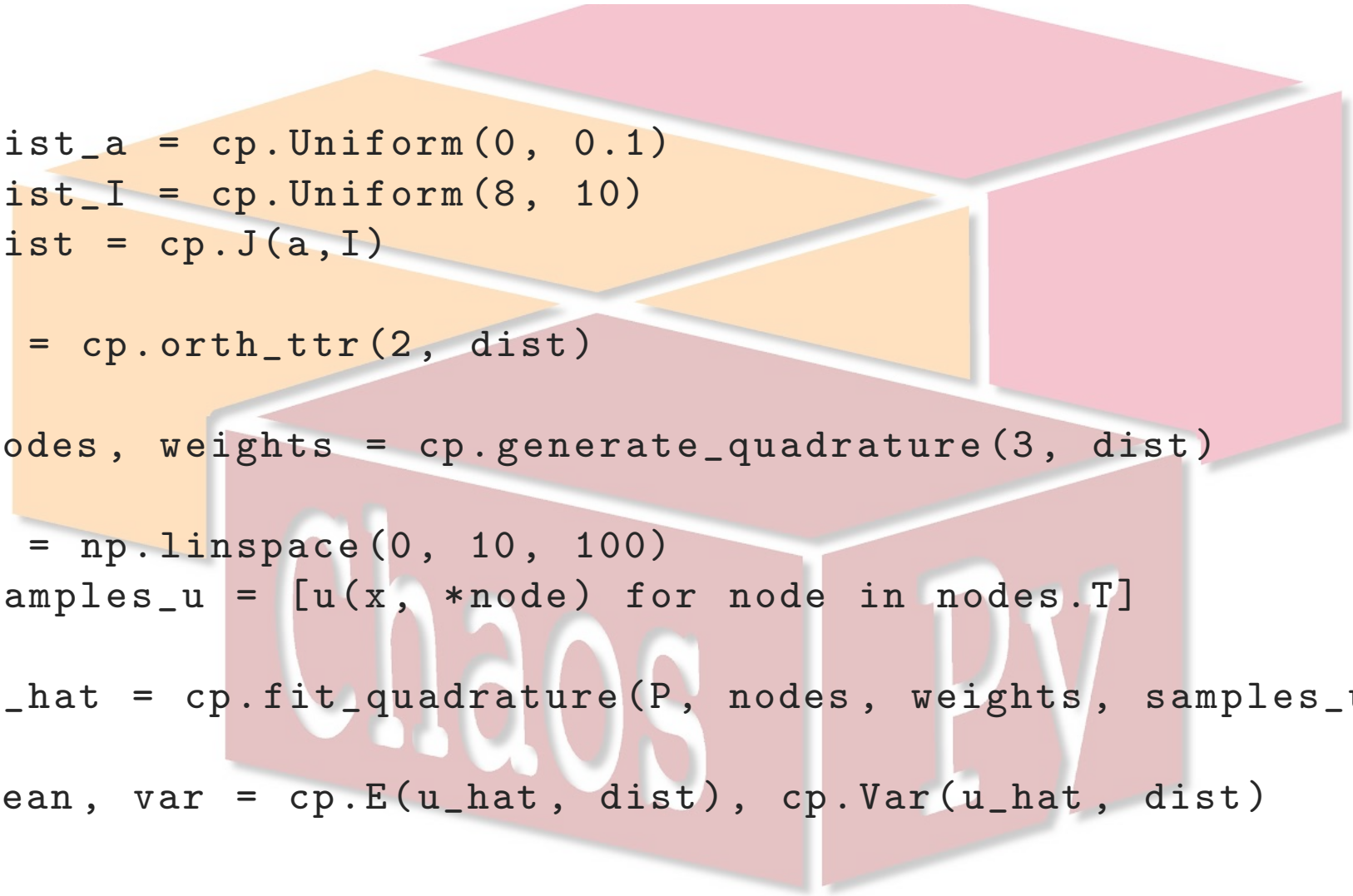
$$u(x; q) = q_1 e^{-q_0 x}$$

Uncertain model parameters:

$$q_0 \sim \text{Uniform}(0,\, 0.1) \qquad q_1 = \text{Uniform}(8,\, 10)$$

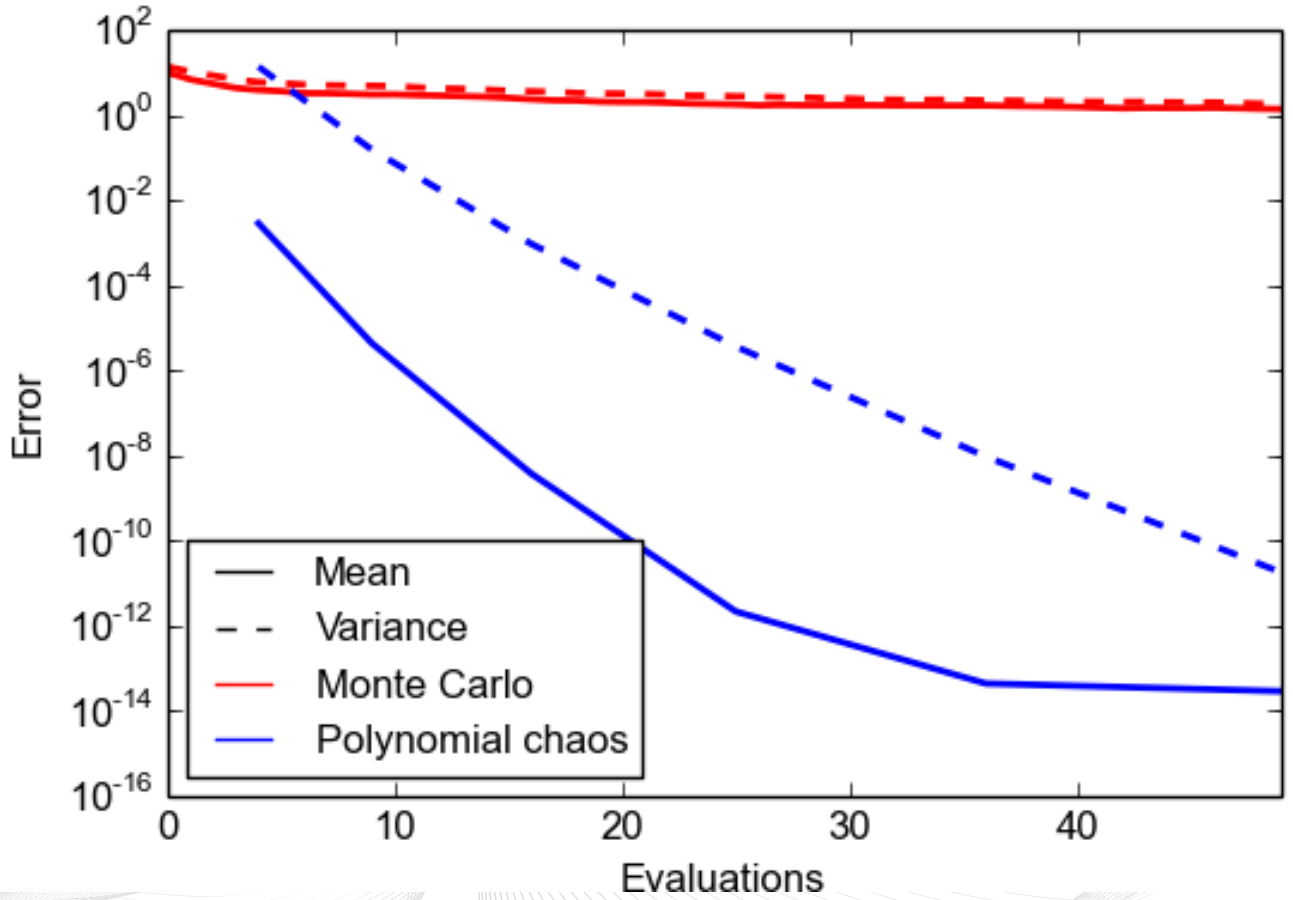
$$\varepsilon_E = \int_0^{10} |E(u) - E(\hat{u})|\, dx \quad \varepsilon_{Var} = \int_0^{10} |Var(u) - Var(\hat{u})|\, dx$$

A full implementation of pseudo-spectral projection in Chaospy




```
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)

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)
```

Convergence of the two-dimensional problem



The point collocation method is alternative to the pseudo-spectral method

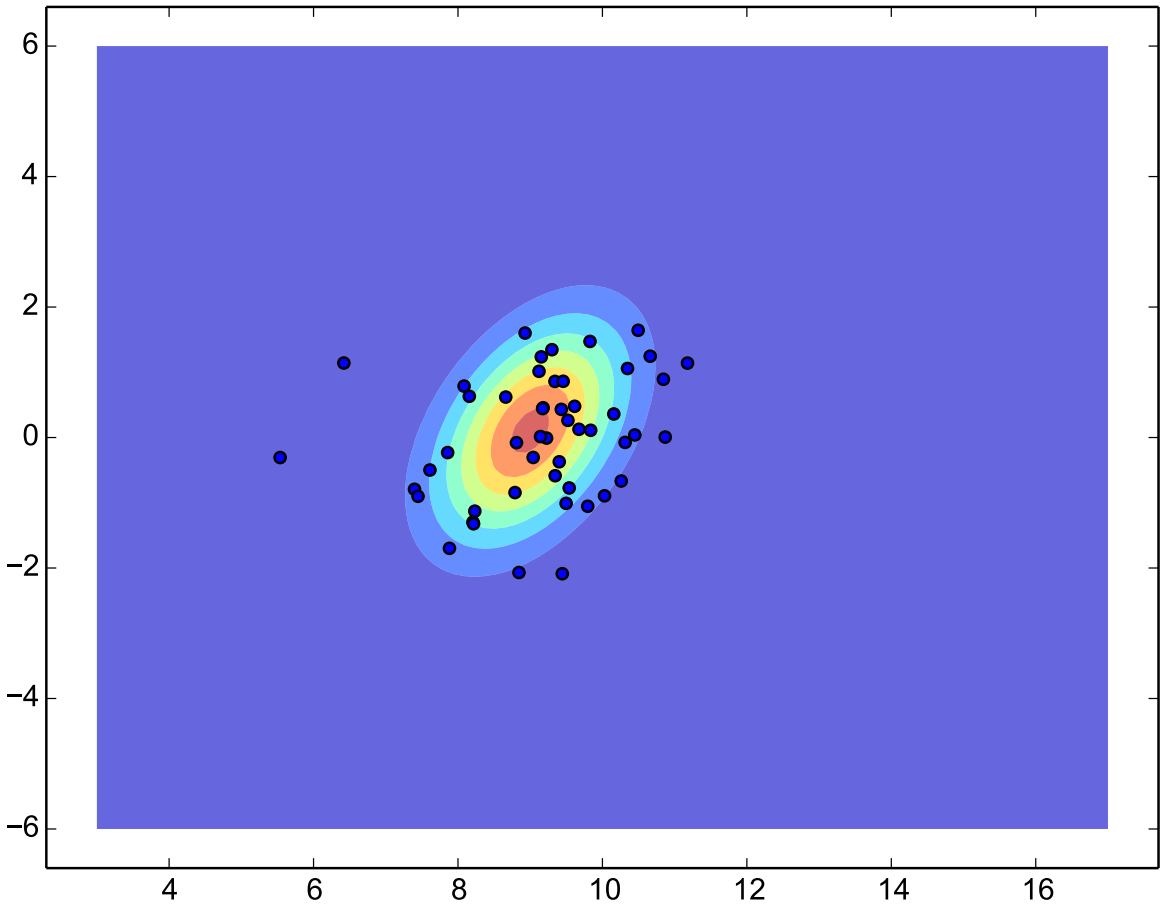
1. Psuedo-spectral method:
   - 1.1 Determine polynomial approximation of model by least squares minimization in a space weighted with the probability distribution
   - 1.2 Approximate integrals in c_n by quadrature rules
2. Point collocation method:
   - 2.1 Determine polynomial approximation of model by least squares minimization in a vector space as in regression (or overdetermined matrix systems)
   - 2.2 Need to choose a set of nodes (regression points)

The point collocation method: estimate c_n using linear regression

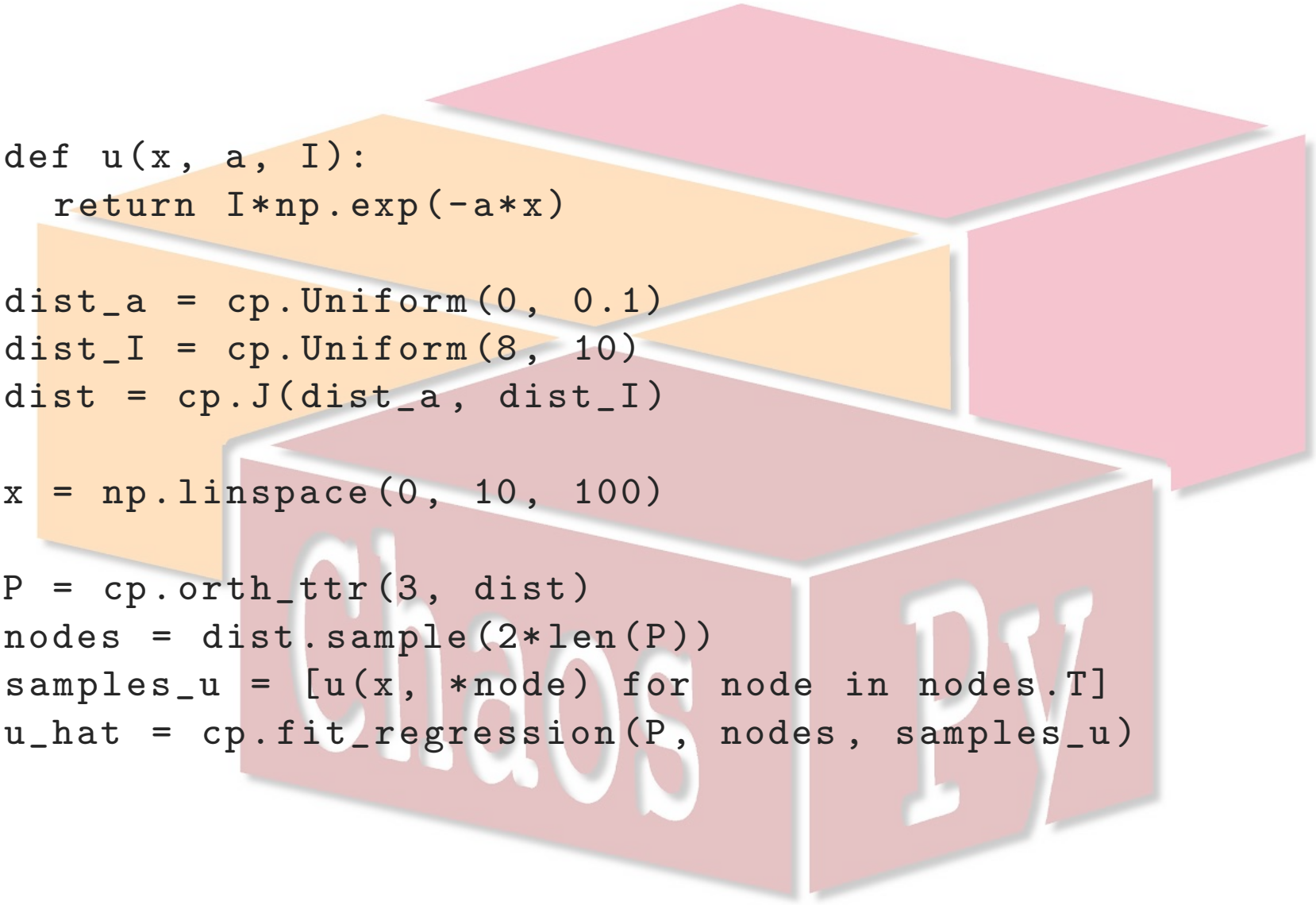
$$\mathbf{c} = \begin{bmatrix} c_0(x) \\ \vdots \\ c_N(x) \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} P_0(q_0) & \cdots & P_N(q_0) \\ \vdots & & \vdots \\ P_0(q_K) & \cdots & P_N(q_K) \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} u(x;q_0) \\ \vdots \\ u(x,q_K) \end{bmatrix}$$

$$\begin{aligned} \hat{\mathbf{c}} &= \underset{\mathbf{c}}{\operatorname{argmin}} \|\mathbf{P}\mathbf{c} - \mathbf{u}\|_2^2 \\ &= (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T\mathbf{u} \end{aligned}$$

Collocation nodes should be placed where probability is high

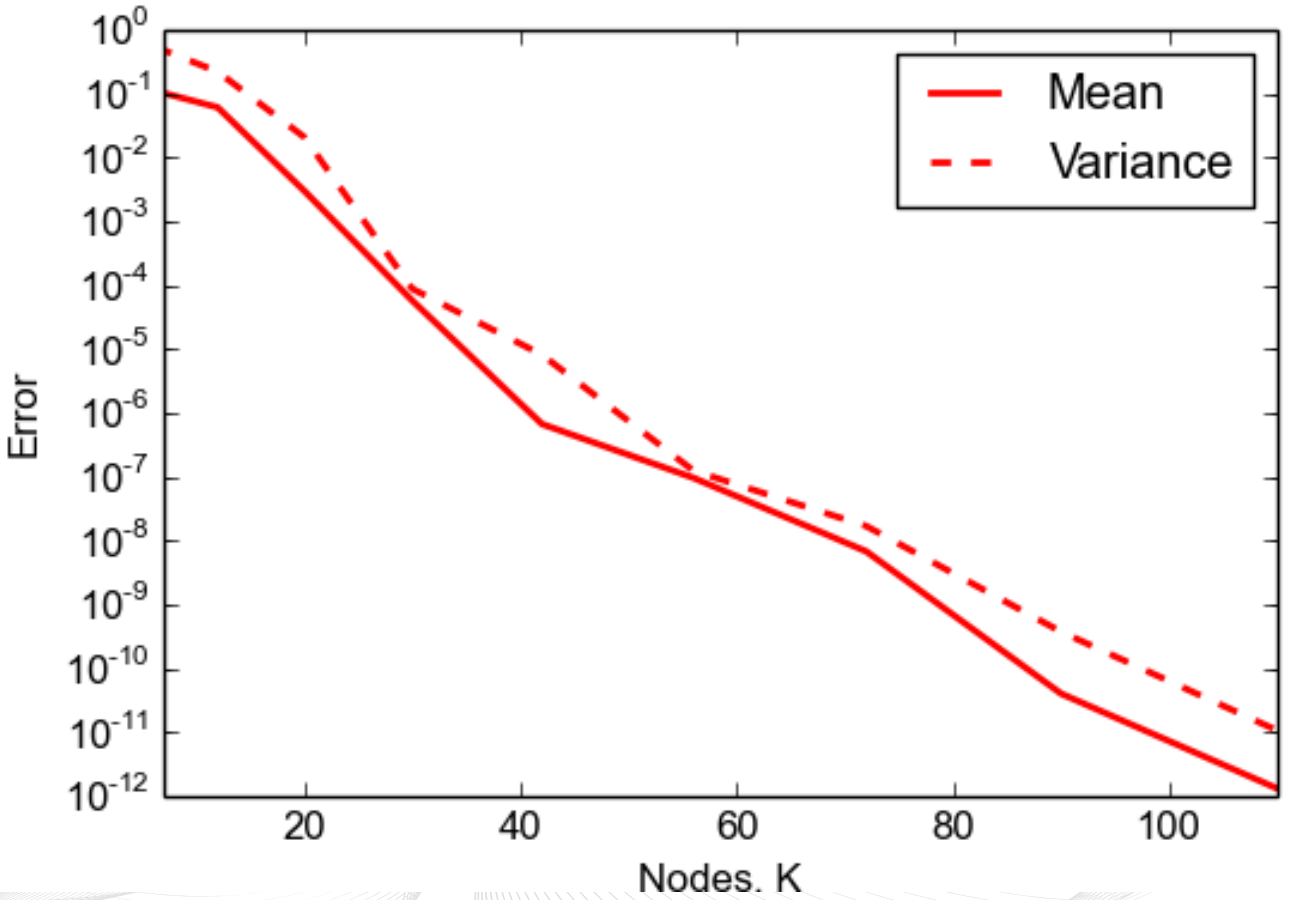


Code for least square minimization

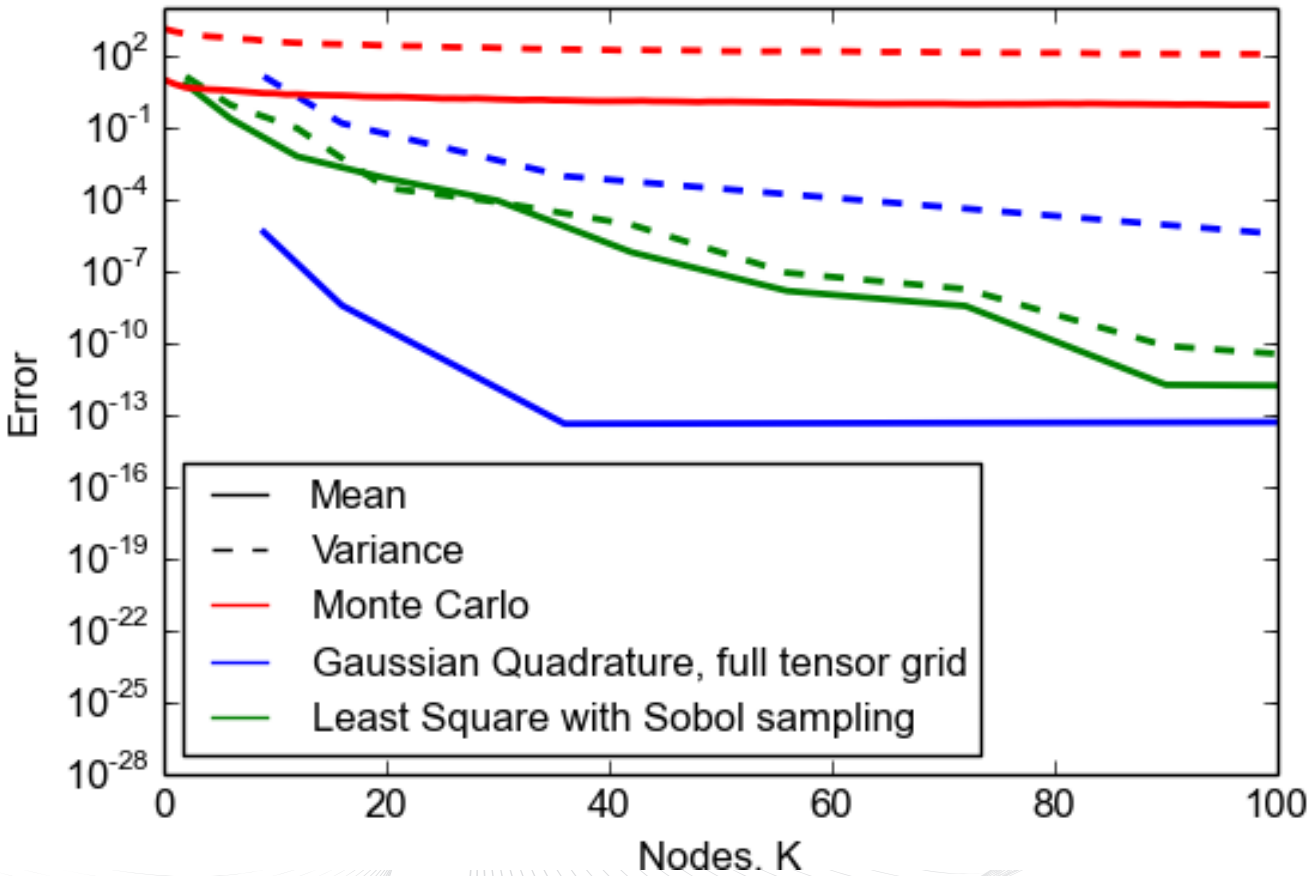



```
def u(x, a, I):
  return I*np.exp(-a*x)

dist_a = cp.Uniform(0, 0.1)
dist_I = cp.Uniform(8, 10)
dist = cp.J(dist_a, dist_I)

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

P = cp.orth_ttr(3, dist)
nodes = dist.sample(2*len(P))
samples_u = [u(x, *node) for node in nodes.T]
u_hat = cp.fit_regression(P, nodes, samples_u)
```

Convergence using point collocation method



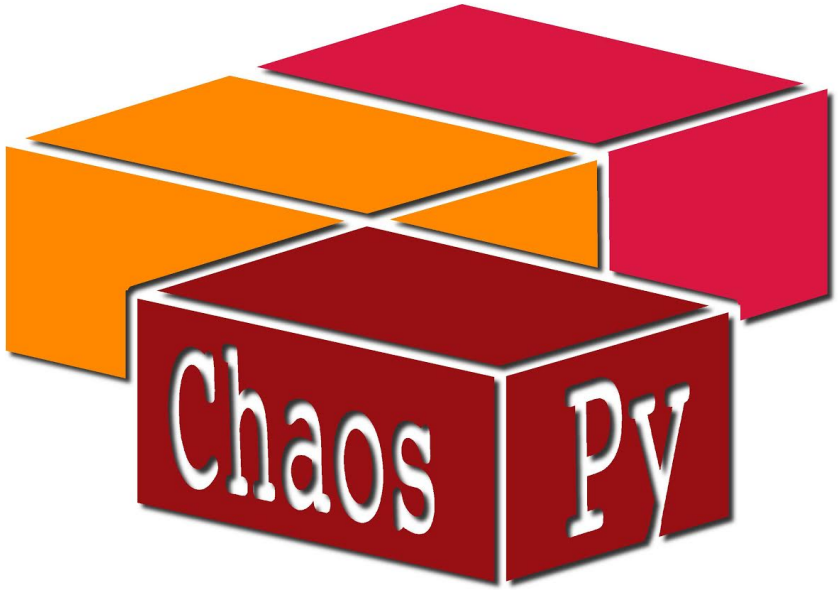
What is best of pseudo-spectral and point collocation method? It's problem dependent!



Which method to choose for your problem

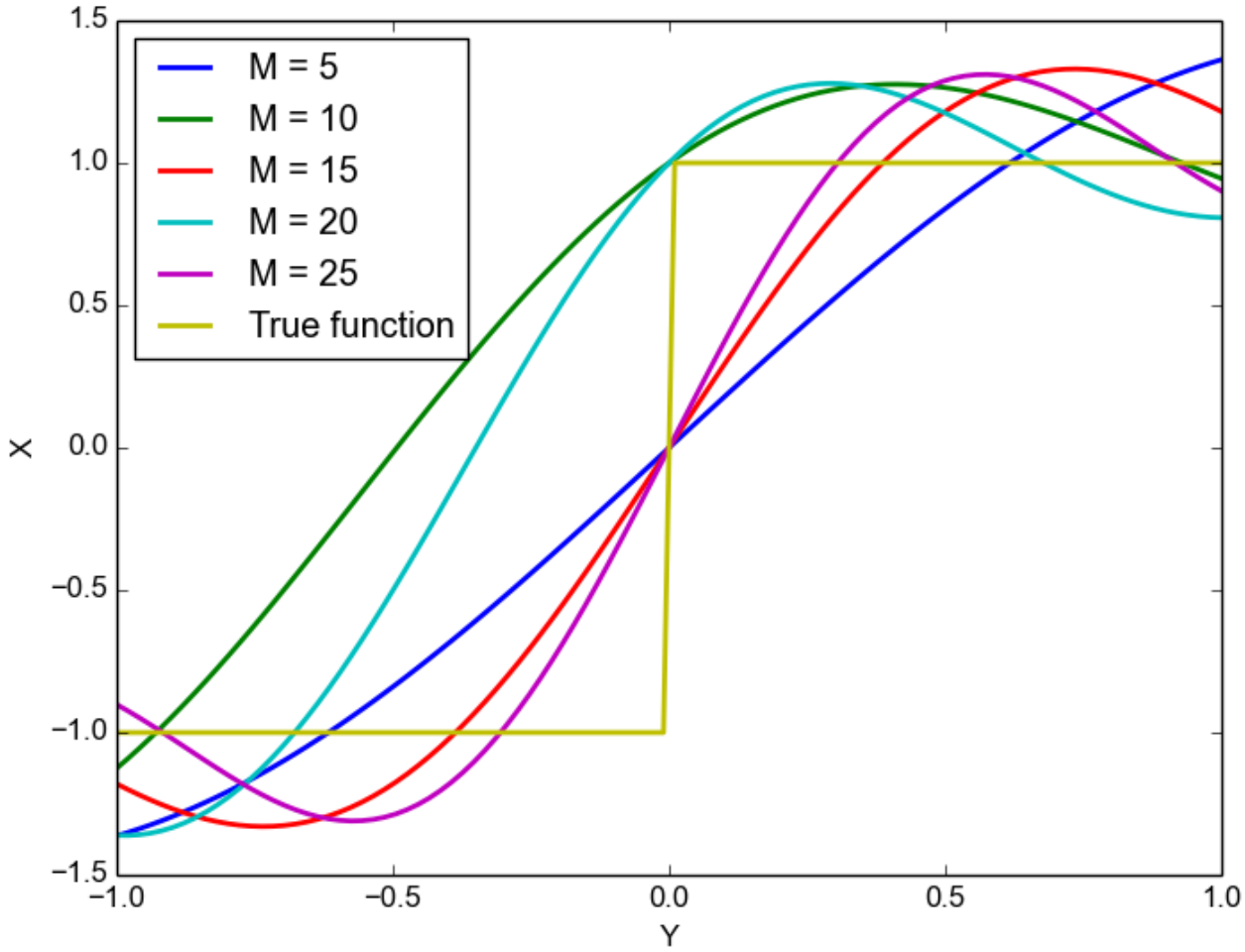
	Pseudo-spectral	Point collocation	Monte Carlo
Efficiency	Highest	Very high	Very low
Stability	Low	Medium	Very high
Dimension-independence	Lowest	Low	Highest

Thank you for your attention

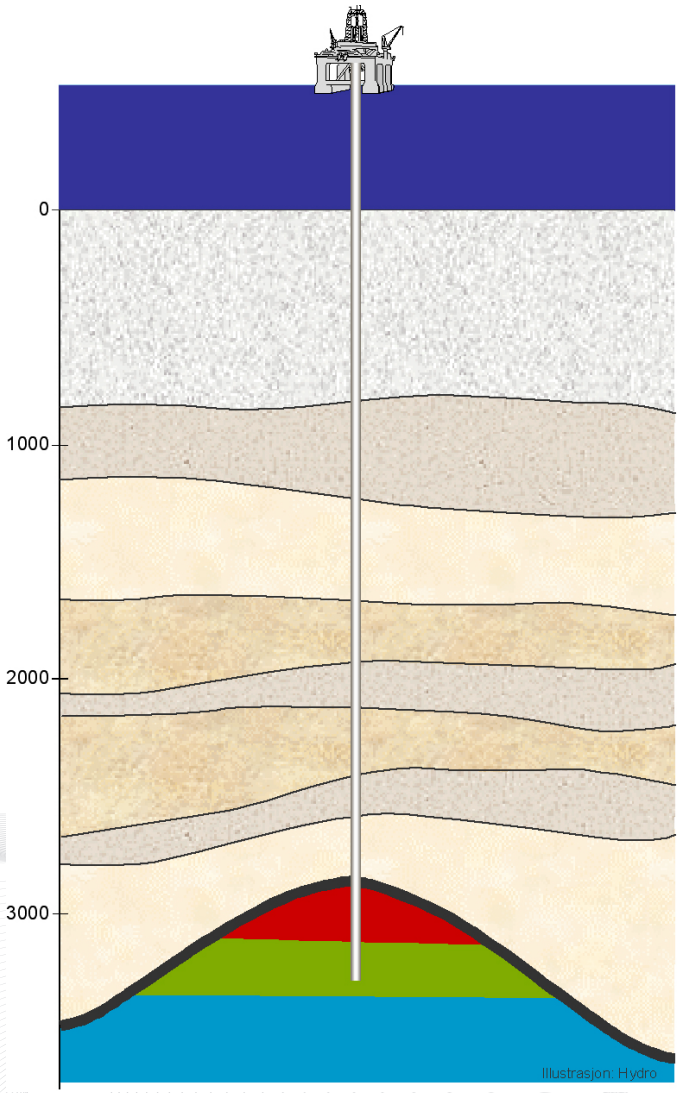


Software page:
https://github.com/hplgit/chaospy

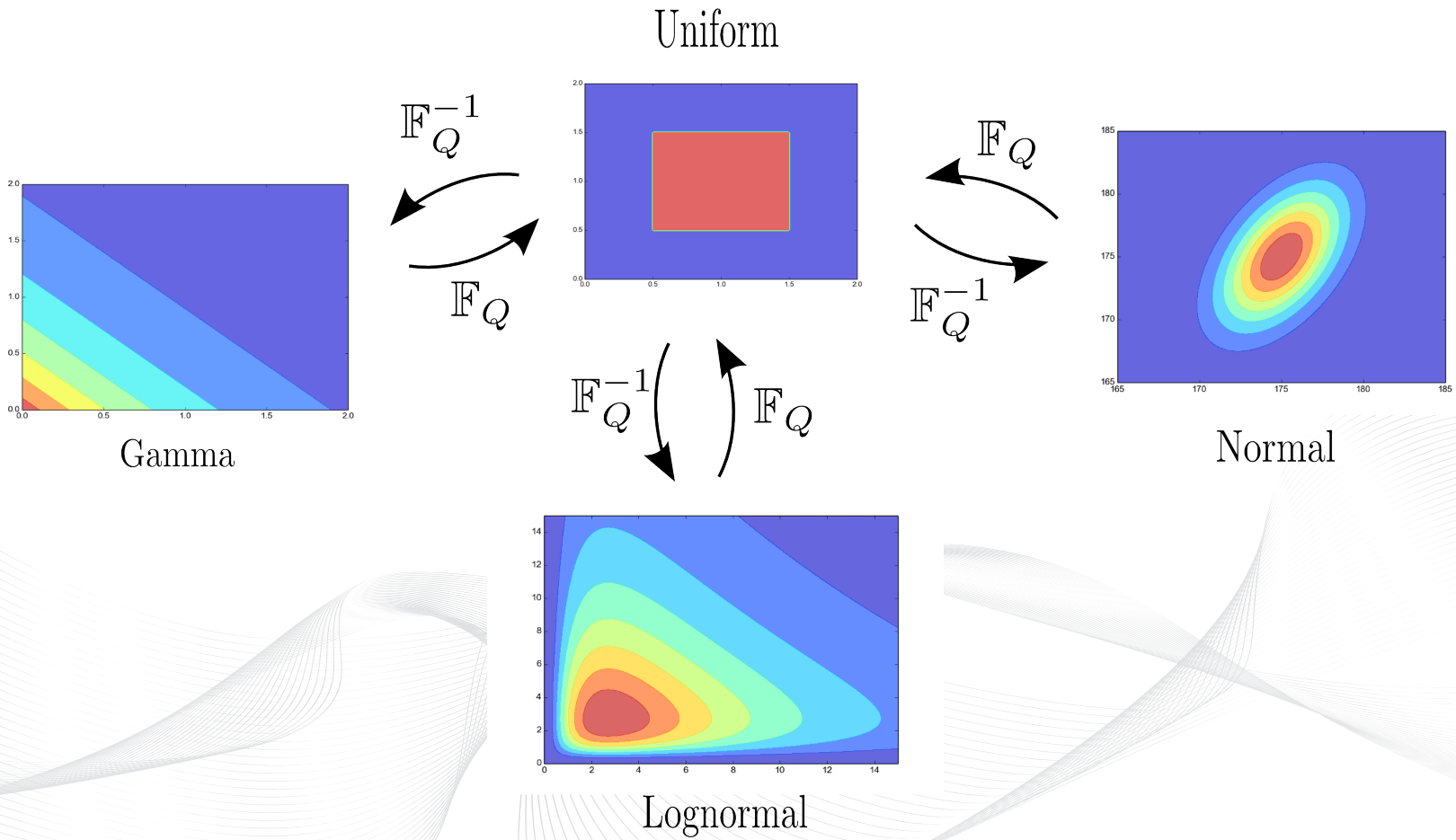
Gibb's Phenomena: discontinuities give oscillations



Example: 1-dimensional diffusion problem



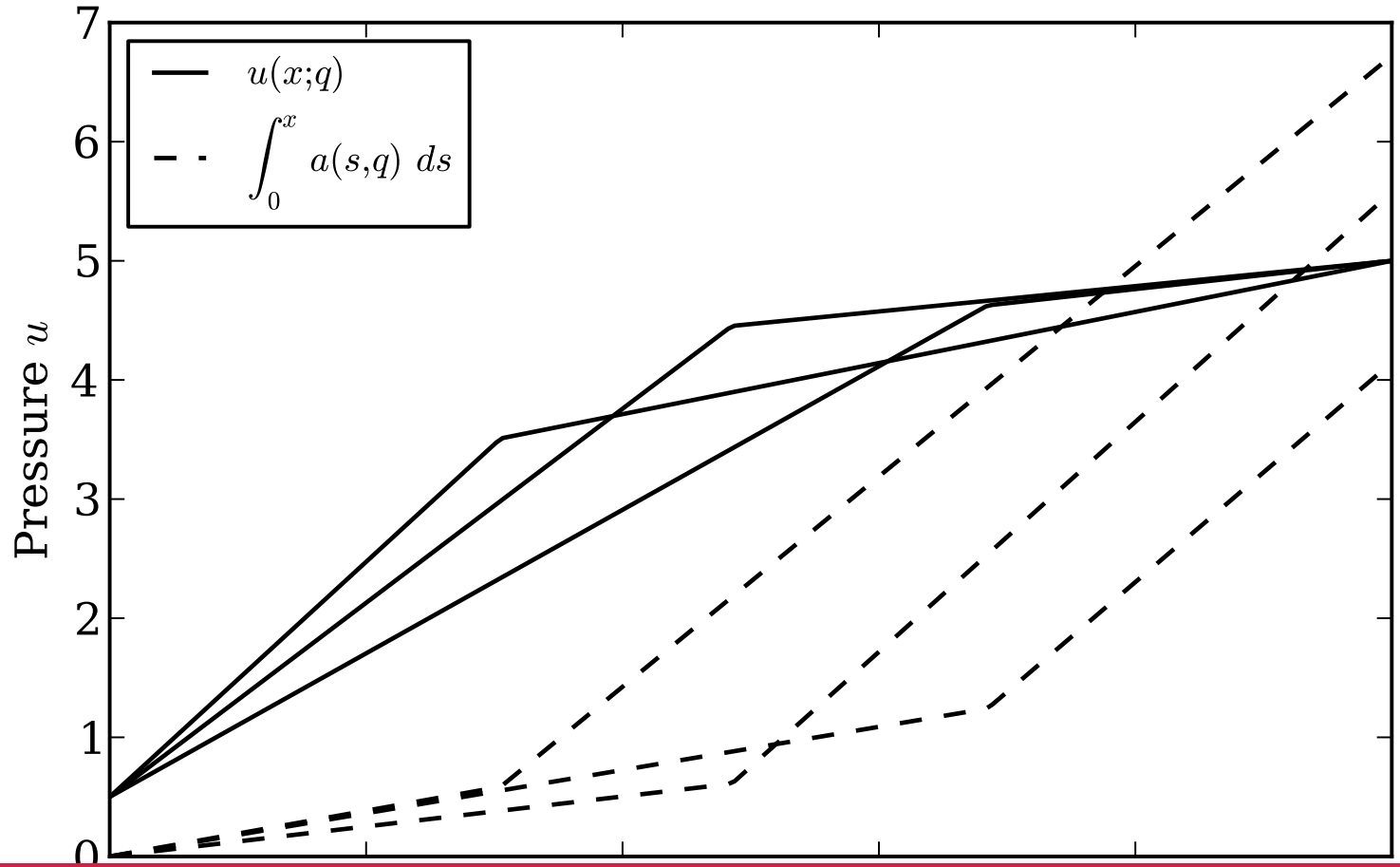
All random variables can with aid of the Rosenblatt transformations be transformed to/from the uniform distribution



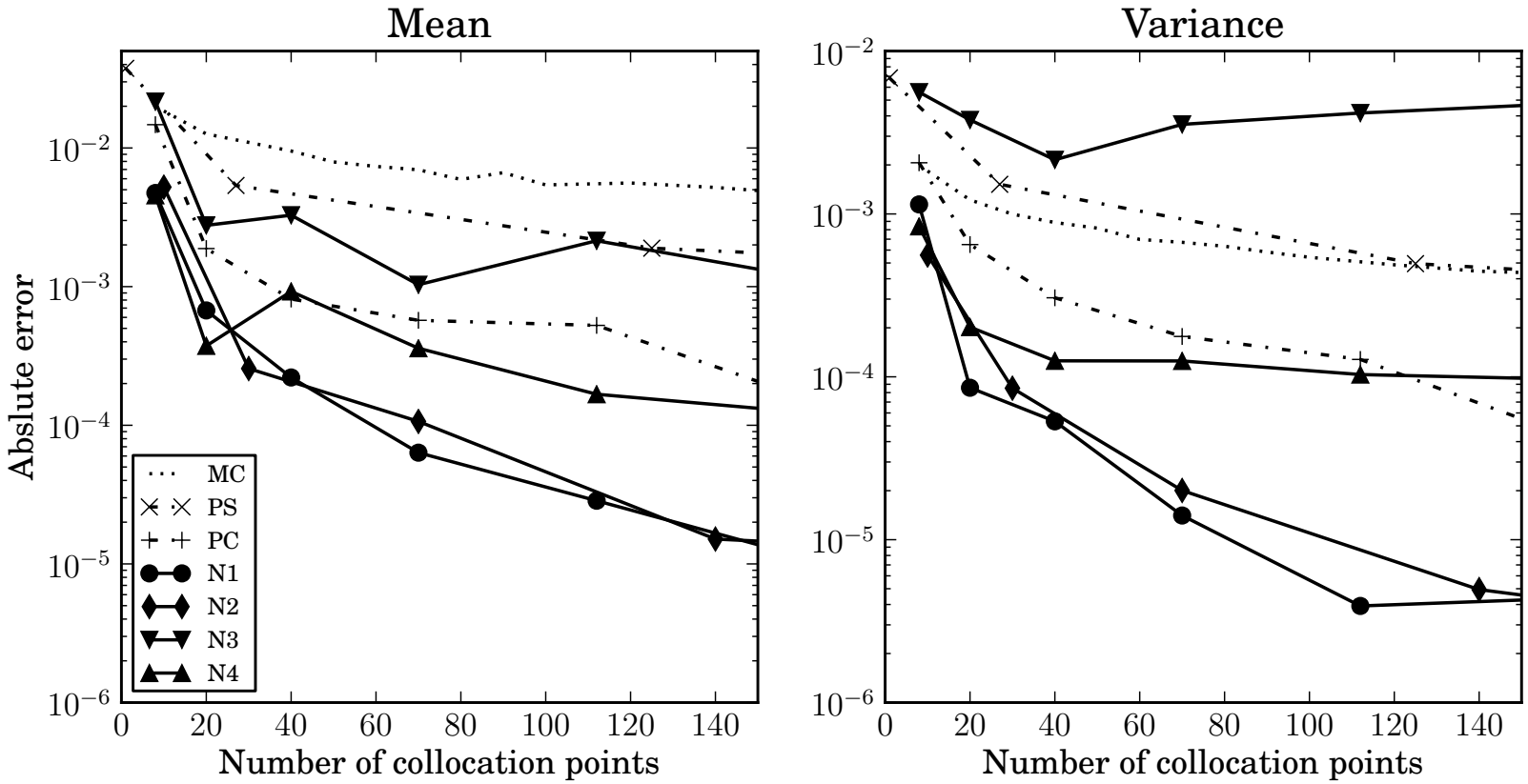
Transformations can be used to reparameterize the model solver

$$\hat{u}_M(x;q) = \hat{u}_M(x;T(r)) = \sum_{n=0}^{N} c_n(x) P_n(r)$$

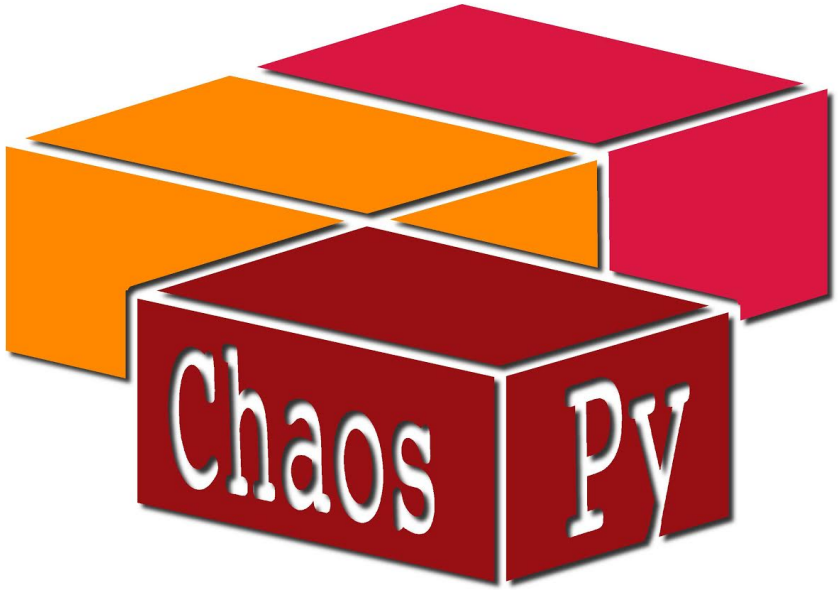
If discontinuity can be tracked, a transformations can aften be created to mimic the discontinuety in probability space



Convergence can often be restored!



Thank you for your attention (again)



Software page:
https://github.com/hplgit/chaospy