

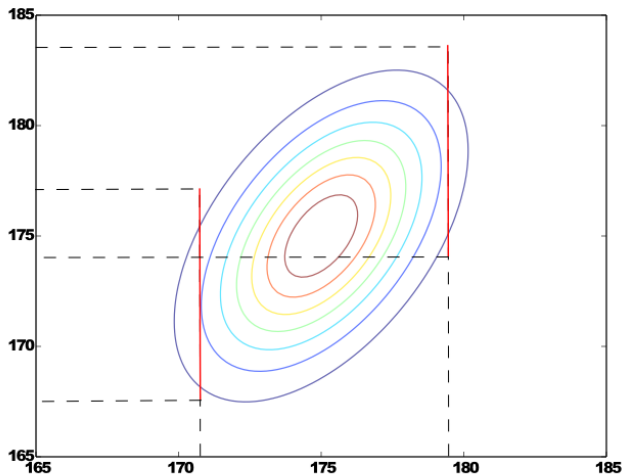
Polynomial chaos expansions part 4: Generalized polynomial chaos

Jonathan Feinberg and Simen Tennøe

Kalkulo AS

December 16, 2015

Some random variables are dependent



Repetition of our model problem with two independent variables I and a

```
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)  
  
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)
```

Dependent variables break the orthogonality property of polynomials!

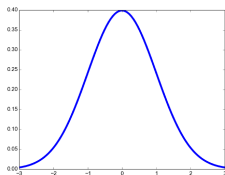
$$P_{\mathbf{i}} = P_{i_1}^{(1)} P_{i_2}^{(2)} \dots P_{i_D}^{(D)} \quad \mathbf{i} = (i_1, i_2, \dots, i_D)$$

$$\begin{aligned} \langle P_{\mathbf{i}}, P_{\mathbf{j}} \rangle_Q &= E(P_{\mathbf{i}}, P_{\mathbf{j}}) \\ &= E\left(P_{i_1}^{(1)} \dots P_{i_D}^{(D)} P_{j_1}^{(1)} \dots P_{j_D}^{(D)}\right) \\ &= E\left(P_{i_1}^{(1)} P_{j_1}^{(1)}\right) \dots E\left(P_{i_D}^{(D)} P_{j_D}^{(D)}\right) \\ &= \dots \\ &= \left\| P_{\mathbf{i}}^{(1)} \right\|_Q \delta_{\mathbf{ij}} \end{aligned}$$

But the problem is:

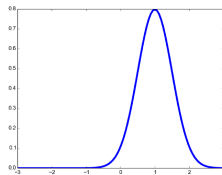
$$E(uv) \neq E(u) E(v) \quad \text{when } u \text{ and } v \text{ are stochastically dependent}$$

Transformations manipulates probability distributions



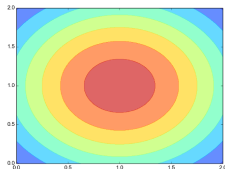
$$R \sim N(0, 1)$$

$$R = \frac{Q - \mu}{\sigma}$$
$$Q = R\sigma + \mu$$



$$Q \sim N(\mu, \sigma^2)$$

Transformation for multivariate distributions

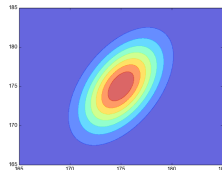


$$R \sim MvN(\mathbf{0}, I)$$

$$R = L^{-1}(Q - \mu)$$



$$Q = LR + \mu$$



$$Q \sim MvN(\mu, \Sigma_Q)$$

$$\Sigma_Q = L^T L$$

$$\mu_Q = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\Sigma_Q = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$$

$$\mu_R = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\Sigma_R = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$Q_1(R_1, R_2) = R_1$$

$$Q_2(R_1, R_2) = \rho R_2 + \sqrt{1 - \rho^2} R_1$$

Transformations can be used to model dependent variables effectively as a parameterization of independent variables

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

Code for dependent normal variables

```
x = np.linspace(0, 1, 100)
def u(x, a, I):
    return I*np.exp(-a*x)

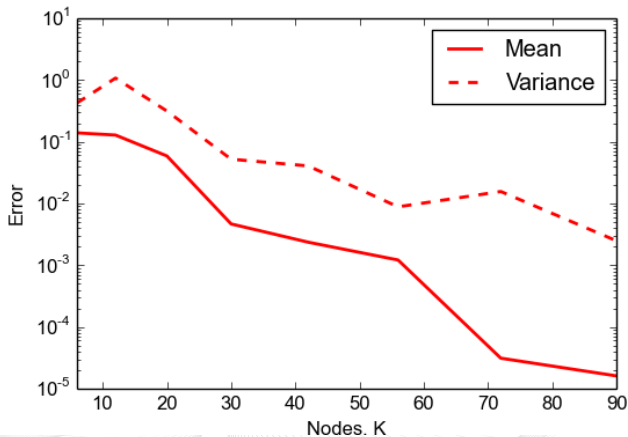
C = [[2,1],[1,2]]
mu = [0.5,0.5]

dist_R = cp.J(cp.Normal(), cp.Normal())
P = cp.orth_ttr(M, dist_R)
L = np.linalg.cholesky(C)          # C = np.dot(L.T, L)
def T(r):
    return np.dot(L,r) + mu

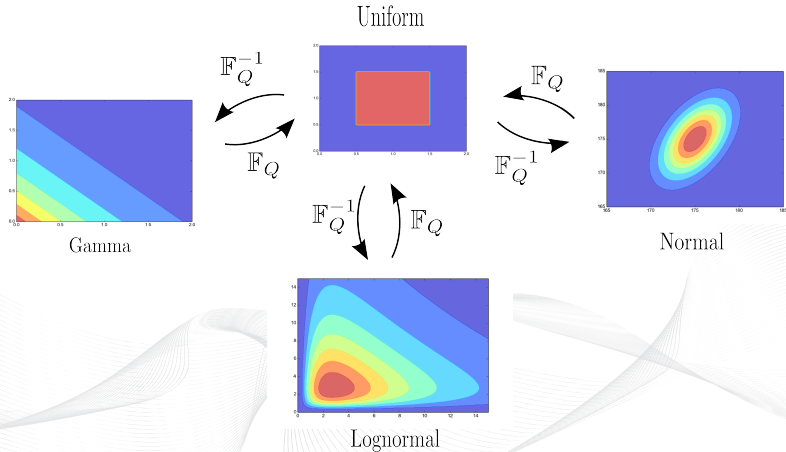
nodes_R = dist_R.sample(2*len(P), "M")
nodes_Q = T(nodes_R)

samples_u = [u(x, *node) for node in nodes_Q.T]
u_hat = cp.fit_quadrature(\
    P, nodes_R, weights, samples_u)
```


Convergence plot



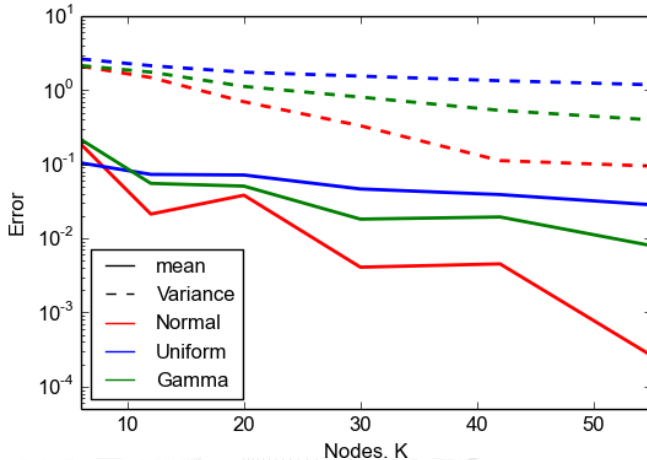
All random variables can with aid of the Rosenblatt transformations be transformed to/from the uniform distribution



Point collocation with Rosenblatt transformation

```
def u(x,a, I):  
    return I*np.exp(-a*x)  
  
dist_R = cp.J(cp.Normal(), cp.Normal())  
C = [[1, 0.5], [0.5, 1]]  
mu = [0, 0]  
dist_Q = cp.MvNormal(mu, C)  
  
P = cp.orth_ttr(M, dist_R)  
nodes_R = dist_R.sample(2*len(P), "M")  
nodes_Q = dist_Q.inv(dist_R.fwd(nodes_R))  
  
x = np.linspace(0, 1, 100)  
samples_u = [u(x, *node) for node in nodes_Q.T]  
u_hat = cp.fit_regression(P, nodes_R, samples_u)
```

Convergence of point collocation with Rosenblatt transformations



Rosenblatt transformations is essentially variable substitution

$$E(u) = \int u(x; q) f_Q(q) dq = \int u(x; q) f_Q(q) \frac{f_R(r)}{f_Q(q)} dr$$

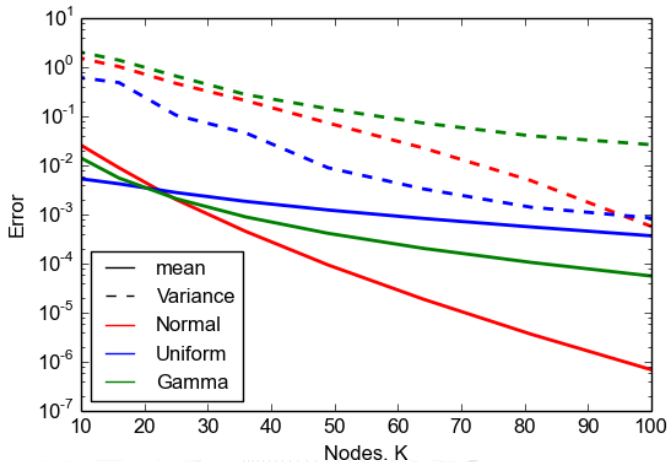
$$F_Q(q) = F_R(r) \quad q = F_Q^{-1}(F_R(r))$$

$$f_Q(q) dq = f_R(r) dr \quad dq = \frac{f_R(r)}{f_Q(q)} dr$$

Pseudo-spectral with Rosenblatt transformation

```
def u(x,a, I):  
    return I*np.exp(-a*x)  
  
C = [[1,0.5],[0.5,1]]  
mu = np.array([0, 0])  
dist_R = cp.J(cp.Normal(), cp.Normal())  
dist_Q = cp.MvNormal(mu, C)  
  
P = cp.orth_ttr(M, dist_R)  
nodes_R, weights_R = cp.generate_quadrature(M+1, dist_R)  
nodes_Q = dist_Q.inv(dist_R.fwd(nodes_R))  
weights_Q = weights_R*\  
    dist_Q.pdf(nodes_Q)/dist_R.pdf(nodes_R)  
  
x = np.linspace(0, 1, 100)  
samples_u = [u(x, *node) for node in nodes_Q.T]  
u_hat = cp.fit_quadrature(P, nodes_R, weights_Q, samples_u)
```

Convergence of pseudo-spectral projection with Rosenblatt transformations



Thank you



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>