Polynomial chaos expansions: Solutions

Jonathan Feinberg and Simen Tennøe

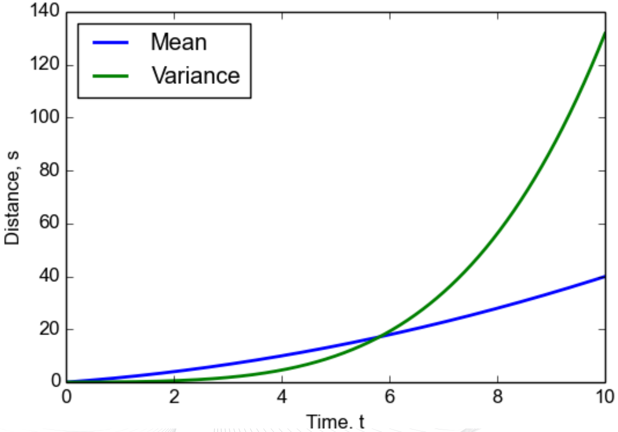
Kalkulo AS

December 16, 2015

Traveling with constant acceleration, classical Monte Carlo integration.

```
def s(t, v0, a):
    return v0*t + 0.5*a*t**2

N = 1000
v0 = cp.Uniform(1,2)
a = cp.Beta(2,2)
t = np.linspace(0,10,1000)

samples_v0 = v0.sample(N)
samples_a = a.sample(N)

distance = np.array([s(t,v0_,a_) for v0_,a_ \
                in zip(samples_v0.T, samples_a.T)])
E = np.sum(distance,0)/N
Var = np.sum(distance**2,0)/N - E**2
plot(t, E)
plot(t, Var)
```

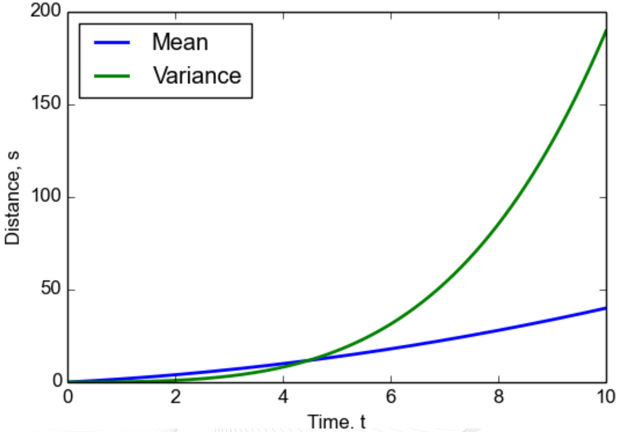
Traveling with constant acceleration, classical Monte Carlo integration.



Traveling with constant acceleration, Quasi-Monte Carlo using Sobol sequence.

```
def s(t, v0, a):
    return v0*t + 0.5*a*t**2

N = 1000
v0 = cp.Uniform(1,2)
a = cp.Beta(2,2)
t = np.linspace(0,10,1000)

samples_v0 = v0.sample(N, "S")
samples_a = a.sample(N, "S")

distance = np.array([s(t,v0_,a_) for v0_,a_ \
                in zip(samples_v0.T, samples_a.T)])
E = np.sum(distance,0)/N
Var = np.sum(distance**2,0)/N - E**2
plot(t, E)
plot(t, Var)
```

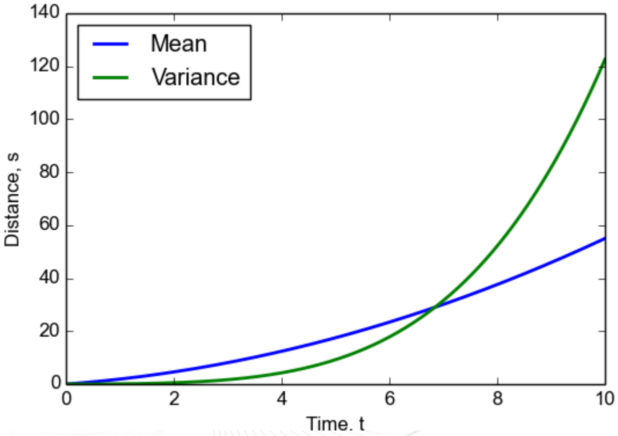
Traveling with constant acceleration, Quasi-Monte Carlo using Sobol sequence.



Traveling with constant acceleration, pseudo-spectral projection with full tensor grid Gaussian quadrature.

```
def s(t, v0, a):
    return v0*t + 0.5*a*t**2

v0 = cp.Uniform(1,2)
a = cp.Beta(2,2)
dist = cp.J(v0,a)

t = np.linspace(0,10,1000)

P = cp.orth_ttr(5, dist)
nodes, weights = cp.generate_quadrature(6, dist, rule="G")
samples_u = [s(t, *n) for n in nodes.T]
u_hat = cp.fit_quadrature(P, nodes, weights, samples_u)

plot(t, cp.E(u_hat, dist))
plot(t, cp.Var(u_hat, dist))
```

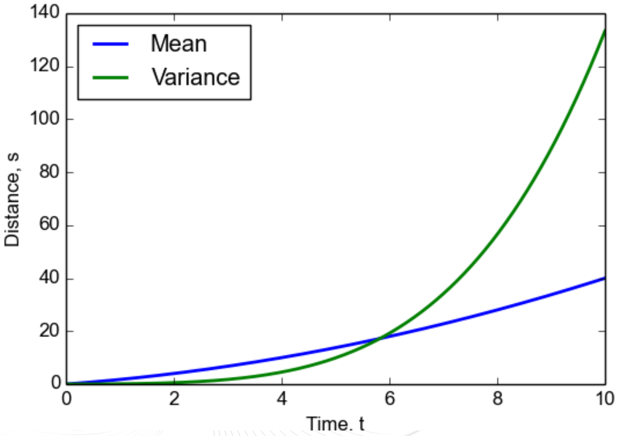
Traveling with constant acceleration, pseudo-spectral projection with full tensor grid Gaussian quadrature.



Traveling with constant acceleration, pseudo-spectral projection with Clenshaw-Curtis and Smolyak sparse grid.

```
def s(t, v0, a):
    return v0*t + 0.5*a*t**2

v0 = cp.Uniform(1,2)
a = cp.Beta(2,2)
dist = cp.J(v0,a)

t = np.linspace(0,10,1000)

P = cp.orth_ttr(M, dist)
nodes, weights = cp.generate_quadrature(6, dist, rule="C" \
                                         , sparse=True)
samples_u = [s(t, *n) for n in nodes.T]
u_hat = cp.fit_quadrature(P, nodes, weights, samples_u)

plot(t, cp.E(u_hat, dist))
plot(t, cp.Var(u_hat, dist))
```

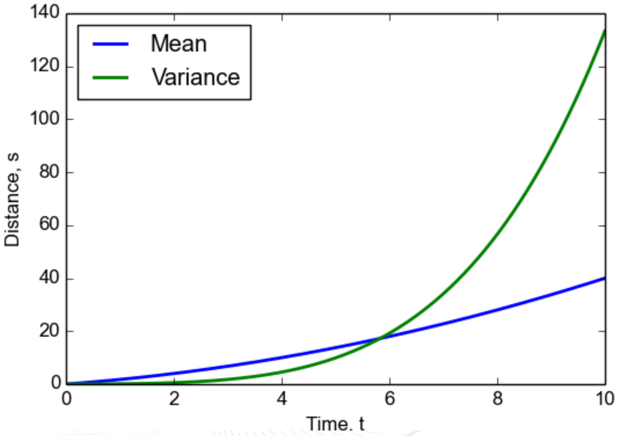
Traveling with constant acceleration, pseudo-spectral projection with Clenshaw-Curtis and Smolyak sparse grid.



Traveling with constant acceleration, point collocation with random samples and least squares minimization.

```
def s(t, v0, a):
    return v0*t + 0.5*a*t**2

v0 = cp.Uniform(1,2)
a = cp.Beta(2,2)
dist = cp.J(v0,a)

t = np.linspace(0,10,1000)

P = cp.orth_ttr(5, dist)
nodes = dist.sample(2*len(P))
samples_u = [s(t, *n) for n in nodes.T]
u_hat = cp.fit_regression(P, nodes, samples_u, rule="LS")

plot(t, cp.E(u_hat, dist))
plot(t, cp.Var(u_hat, dist))
```

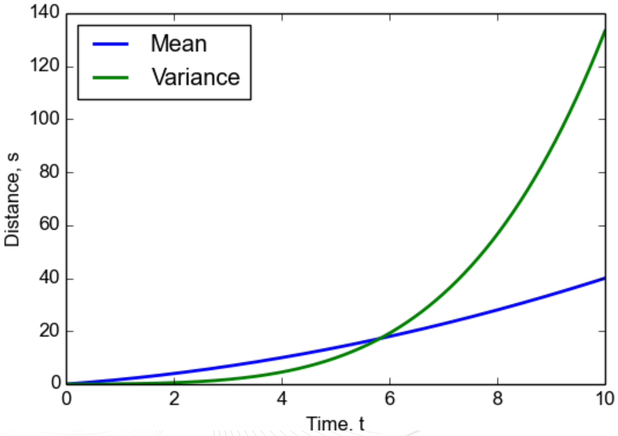
Traveling with constant acceleration, point collocation with random samples and least squares minimization.



Traveling with constant acceleration, point collocation with Hammersley samples and Tikhonov regularization.

```
def s(t, v0, a):
    return v0*t + 0.5*a*t**2

v0 = cp.Uniform(1,2)
a = cp.Beta(2,2)
dist = cp.J(v0,a)

t = np.linspace(0,10,1000)

P = cp.orth_ttr(5, dist)
nodes = dist.sample(2*len(P), "M")
samples_u = [s(t, *n) for n in nodes.T]
u_hat = cp.fit_regression(P, nodes, samples_u, rule="T")

plot(t, cp.E(u_hat, dist))
plot(t, cp.Var(u_hat, dist))
```

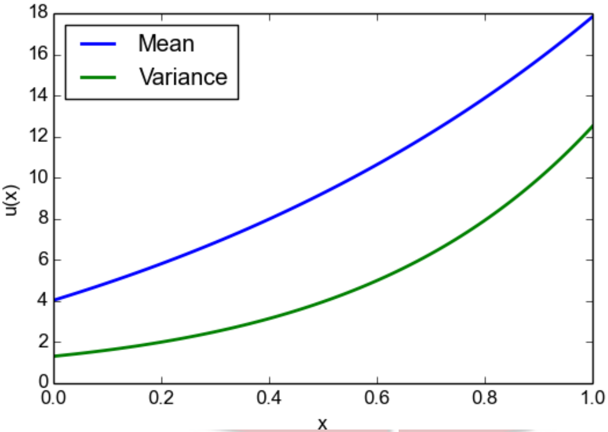
Traveling with constant acceleration, point collocation with Hammersley samples and Tikhonov regularization.



A different differential equation

```
a = cp.Normal(4,1)
I = cp.Uniform(2, 6)
dist_Q = cp.J(a, I)
dist_R = cp.J(cp.Normal(), cp.Uniform())

x = np.linspace(0,1,100)

P = cp.orth_ttr(2, dist_R)
nodes_R, weights_R = cp.generate_quadrature(3, 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)

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

A different differential equation



A different differential equation

```
a = cp.Normal(4,1)
I = cp.Uniform(2, 6)
dist = cp.J(a, I)

x = np.linspace(0,1,100)

P, norm = cp.orth_ttr(5, dist, retall=True)

q0, q1 = cp.variable(2)

P_nk = cp.outer(P, P)
E_ak = cp.E(q0*P, dist)
E_ik = cp.E(q1*P, dist)
E_nk = cp.E(P_nk, dist)
```

A different differential equation

```
def f(c_k,x):
    return (c_k + E_ak)*cp.sum(E_nk, -1)/norm

solver = odespy.RK4(f)
c_0 = E_ik/norm
solver.set_initial_condition(c_0)
c_n, x_ = solver.solve(x)
u_hat = cp.sum(P*c_n,-1)

E = cp.E(u_hat, dist)
Var = cp.Var(u_hat, dist)

plot(x, cp.E(u_hat, dist))
plot(x, cp.Var(u_hat, dist))
```

A different differential equation