A primer on uncertainty quantifications

Kjetil Olsen Lye

Uncertainty quantification

- Models of physical systems always have uncertainties

Uncertainty quantification

- Models of physical systems always have uncertainties
- Uncertainties in parameters, initial conditions, boundary conditions, model form, …

SINTEF

Uncertainty quantification

- Models of physical systems always have uncertainties
- Uncertainties in parameters, initial conditions, boundary conditions, model form, …
- How do these uncertainties affect the model predictions?

SINTEF

Uncertainty quantification

- Models of physical systems always have uncertainties
- Uncertainties in parameters, initial conditions, boundary conditions, model form, …
- How do these uncertainties affect the model predictions?
- **Goal:** How can we effectively compute statistics of quantities of interest given uncertainties in input data?

Problem formulation

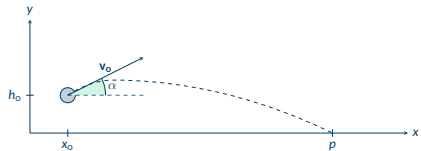
$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

Problem formulation

$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

For example

- Projectile motion:

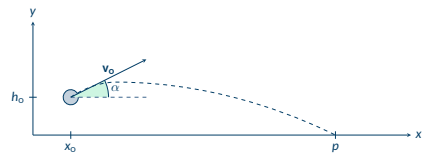


Problem formulation

$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

For example

- Projectile motion:
  - Q: Landing position of a ball

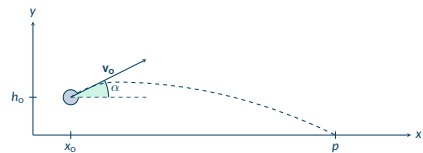


Problem formulation

$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

For example

- Projectile motion:
  - Q: Landing position of a ball
  - $\mathbf{y}$: Initial height, angle, velocity, air resistance, ...



Problem formulation

$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

For example

- Projectile motion:
  - Q: Landing position of a ball
  - $\mathbf{y}$: Initial height, angle, velocity, air resistance, ...
- Subsurface oil reservoir:

Problem formulation

$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

For example

- Projectile motion:
  - Q: Landing position of a ball
  - $\mathbf{y}$: Initial height, angle, velocity, air resistance, ...
- Subsurface oil reservoir:
  - Q: Amount of oil extracted in 10 years

Problem formulation

$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

For example

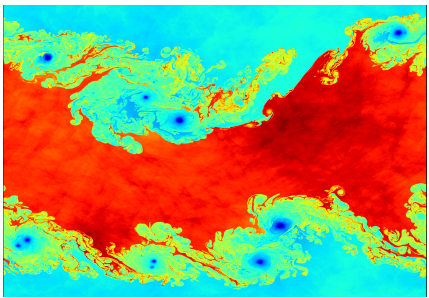
- Projectile motion:
  - Q: Landing position of a ball
  - $\mathbf{y}$: Initial height, angle, velocity, air resistance, ...
- Subsurface oil reservoir:
  - Q: Amount of oil extracted in 10 years
  - $\mathbf{y}$: Permeability field, porosity field, ...

Problem formulation

$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

For example

- Projectile motion:
  - Q: Landing position of a ball
  - $\mathbf{y}$: Initial height, angle, velocity, air resistance, ...
- Subsurface oil reservoir:
  - Q: Amount of oil extracted in 10 years
  - $\mathbf{y}$: Permeability field, porosity field, ...
- Turbulence:

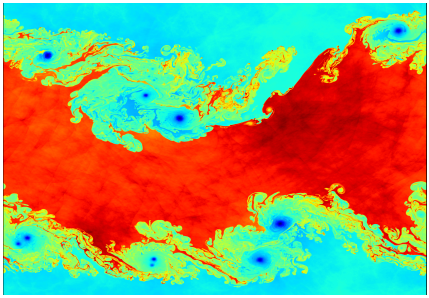


Problem formulation

$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

For example

- Projectile motion:
  - Q: Landing position of a ball
  - $\mathbf{y}$: Initial height, angle, velocity, air resistance, ...
- Subsurface oil reservoir:
  - Q: Amount of oil extracted in 10 years
  - $\mathbf{y}$: Permeability field, porosity field, ...
- Turbulence:
  - Q: Drag on airplane wing

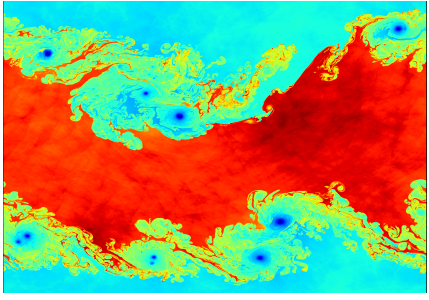


Problem formulation

$$\text{Quantity of interest } Q = Q(\underbrace{\text{Model input}}_{\text{uncertain}}) = Q(\mathbf{y})$$

For example

- Projectile motion:
  - Q: Landing position of a ball
  - $\mathbf{y}$: Initial height, angle, velocity, air resistance, ...
- Subsurface oil reservoir:
  - Q: Amount of oil extracted in 10 years
  - $\mathbf{y}$: Permeability field, porosity field, ...
- Turbulence:
  - Q: Drag on airplane wing
  - $\mathbf{y}$: Initial conditions, perturbations, ...



Next three lectures summarized in one slide

- $\mathbf{y} \in D \subset \mathbb{R}^d$ uncertain parameters

SINTEF

Next three lectures summarized in one slide

- $\mathbf{y} \in D \subset \mathbb{R}^d$ uncertain parameters
- $Q(\mathbf{y})$ quantity of interest

SINTEF

Next three lectures summarized in one slide

- $\mathbf{y} \in D \subset \mathbb{R}^d$ uncertain parameters
- $Q(\mathbf{y})$ quantity of interest

How do we efficiently compute:

- Expectation of Q $(= \mathbb{E}(Q))$?

SINTEF

Next three lectures summarized in one slide

- $\mathbf{y} \in D \subset \mathbb{R}^d$ uncertain parameters
- $Q(\mathbf{y})$ quantity of interest

How do we efficiently compute:

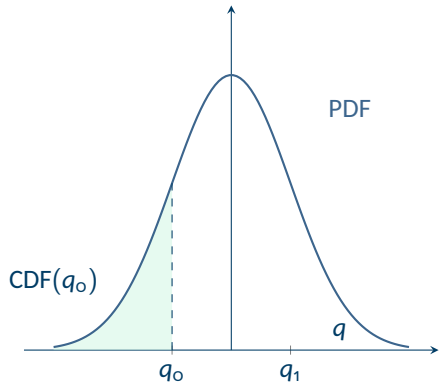
- Expectation of Q ($= \mathbb{E}(Q)$)?
- Variance of Q ($= \mathrm{Var}(Q)$)?

Next three lectures summarized in one slide

- $\mathbf{y} \in D \subset \mathbb{R}^d$ uncertain parameters
- $Q(\mathbf{y})$ quantity of interest

How do we efficiently compute:

- Expectation of Q $(= \mathbb{E}(Q))$?
- Variance of Q $(= \mathrm{Var}(Q))$?
- PDF, CDF, ...



Focus of talk

Compute

$$\int_{[0,1]^d} Q(\mathbf{y})\, d\mathbf{y}$$

Focus of talk

Compute

$$\int_{[0,1]^d} Q(\mathbf{y})\, d\mathbf{y}$$

where

- Q is expensive to compute

Focus of talk

Compute

$$\int_{[0,1]^d} Q(\mathbf{y})\, d\mathbf{y}$$

where

- Q is expensive to compute
- d is large

Focus of talk

Compute

$$\int_{[0,1]^d} Q(\mathbf{y})\, d\mathbf{y}$$

where

- Q is expensive to compute
- d is large
- Q could be ill-behaved (eg. discontinuous)

Focus of talk

Compute

$$\int_{[0,1]^d} Q(\mathbf{y})\, d\mathbf{y}$$

where

- Q is expensive to compute
- d is large
- Q could be ill-behaved (eg. discontinuous)
- Q could be vector-valued (or even function-valued)

SINTEF

Random variables

- A random variable X is a measurable function $X : \Omega \to \mathbb{R}^d$

SINTEF

Random variables

- A random variable X is a measurable function $X : \Omega \to \mathbb{R}^d$
- $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space

SINTEF

Random variables

- A random variable X is a measurable function $X : \Omega \to \mathbb{R}^d$
- $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space
- The **law** (or distribution) of X is the pushforward measure

$$\mu_X(A) = \mathbb{P}(X^{-1}(A)) \text{ for } A \subset \mathbb{R}^d \text{ Borel}$$

SINTEF

Random variables

- A random variable X is a measurable function $X : \Omega \to \mathbb{R}^d$
- $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space
- The **law** (or distribution) of X is the pushforward measure

$$\mu_X(A) = \mathbb{P}(X^{-1}(A)) \text{ for } A \subset \mathbb{R}^d \text{ Borel}$$

- Change of variables formula:

$$\mathbb{E}(f(X)) = \int_\Omega f(X(\omega))\, d\mathbb{P}(\omega) = \int_{\mathbb{R}^d} f(x)\, d\mu_X(x)$$

SINTEF

Random variables

- A random variable X is a measurable function $X : \Omega \to \mathbb{R}^d$
- $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space
- The **law** (or distribution) of X is the pushforward measure

$$\mu_X(A) = \mathbb{P}(X^{-1}(A)) \text{ for } A \subset \mathbb{R}^d \text{ Borel}$$

- Change of variables formula:

$$\mathbb{E}(f(X)) = \int_\Omega f(X(\omega))\, d\mathbb{P}(\omega) = \int_{\mathbb{R}^d} f(x)\, d\mu_X(x)$$

- If μ_X is absolutely continuous w.r.t. Lebesgue measure: $\mu_X = \rho \cdot dx$

SINTEF

Random variables

- A random variable X is a measurable function $X : \Omega \to \mathbb{R}^d$
- $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space
- The **law** (or distribution) of X is the pushforward measure

$$\mu_X(A) = \mathbb{P}(X^{-1}(A)) \text{ for } A \subset \mathbb{R}^d \text{ Borel}$$

- Change of variables formula:

$$\mathbb{E}(f(X)) = \int_\Omega f(X(\omega))\, d\mathbb{P}(\omega) = \int_{\mathbb{R}^d} f(x)\, d\mu_X(x)$$

- If μ_X is absolutely continuous w.r.t. Lebesgue measure: $\mu_X = \rho \cdot dx$
- Then ρ is the probability density function (PDF)

Common distributions

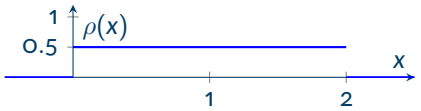
Uniform distribution $X \sim \mathcal{U}[a, b]$

- PDF: $\rho(x) = \frac{1}{b-a}\mathbf{1}_{[a,b]}(x)$
- $\mu_X = \frac{1}{b-a}dx$ on $[a, b]$

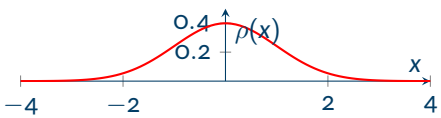
Normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$

- PDF: $\rho(x) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
- $\mu_X = \rho(x)dx$ on $\mathbb{R}$

Uniform $\mathcal{U}[0, 2]$

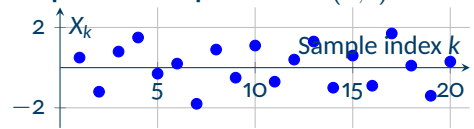


Normal $\mathcal{N}(0, 1)$



Visual interpretation: i.i.d. samples

Independent samples from $\mathcal{N}(0, 1)$



- Each X_k is drawn from same distribution
- Knowing X_1 tells us nothing about X_2

How computers generate random numbers

- Computers are deterministic machines

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

How computers generate random numbers

- Computers are deterministic machines
- Cannot generate truly random numbers

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

How computers generate random numbers

- Computers are deterministic machines
- Cannot generate truly random numbers
- Instead: **Pseudo-random number generators** (PRNGs)

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

How computers generate random numbers

- Computers are deterministic machines
- Cannot generate truly random numbers
- Instead: **Pseudo-random number generators** (PRNGs)
- Start with a **seed** value s_0

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

How computers generate random numbers

- Computers are deterministic machines
- Cannot generate truly random numbers
- Instead: **Pseudo-random number generators** (PRNGs)
- Start with a **seed** value s_0
- Generate sequence using deterministic formula:

$$s_{n+1} = f(s_n) \mod m$$

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

How computers generate random numbers

- Computers are deterministic machines
- Cannot generate truly random numbers
- Instead: **Pseudo-random number generators** (PRNGs)
- Start with a **seed** value s_0
- Generate sequence using deterministic formula:

$$s_{n+1} = f(s_n) \mod m$$

- Common example: Linear Congruential Generator (LCG)

$$s_{n+1} = (a \cdot s_n + c) \mod m$$

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

How computers generate random numbers

- Computers are deterministic machines
- Cannot generate truly random numbers
- Instead: **Pseudo-random number generators** (PRNGs)
- Start with a **seed** value s_0
- Generate sequence using deterministic formula:

$$s_{n+1} = f(s_n) \mod m$$

- Common example: Linear Congruential Generator (LCG)

$$s_{n+1} = (a \cdot s_n + c) \mod m$$

- Map to $[0, 1)$: $U_n = s_n/m$

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

How computers generate random numbers

- Computers are deterministic machines
- Cannot generate truly random numbers
- Instead: **Pseudo-random number generators** (PRNGs)
- Start with a **seed** value s_0
- Generate sequence using deterministic formula:

$$s_{n+1} = f(s_n) \mod m$$

- Common example: Linear Congruential Generator (LCG)

$$s_{n+1} = (a \cdot s_n + c) \mod m$$

- Map to $[0, 1)$: $U_n = s_n/m$
- U_n are **uniformly distributed** in $[0, 1)$

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

How computers generate random numbers

- Computers are deterministic machines
- Cannot generate truly random numbers
- Instead: **Pseudo-random number generators** (PRNGs)
- Start with a **seed** value s_0
- Generate sequence using deterministic formula:

$$s_{n+1} = f(s_n) \mod m$$

- Common example: Linear Congruential Generator (LCG)

$$s_{n+1} = (a \cdot s_n + c) \mod m$$

- Map to $[0, 1)$: $U_n = s_n/m$
- U_n are **uniformly distributed** in $[0, 1)$
- Good PRNGs produce sequences that "look random"

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

How computers generate random numbers

- Computers are deterministic machines
- Cannot generate truly random numbers
- Instead: **Pseudo-random number generators** (PRNGs)
- Start with a **seed** value s_0
- Generate sequence using deterministic formula:

$$s_{n+1} = f(s_n) \mod m$$

- Common example: Linear Congruential Generator (LCG)

$$s_{n+1} = (a \cdot s_n + c) \mod m$$

- Map to $[0, 1)$: $U_n = s_n/m$
- U_n are **uniformly distributed** in $[0, 1)$
- Good PRNGs produce sequences that "look random"
- Same seed $\Rightarrow$ same sequence (reproducibility!)

In Julia:

```
using Random
Random.seed!(1234)
```

SINTEF

Computers: from uniform to other distributions

- Start with $U \sim \mathcal{U}[0, 1]$

In Julia:

```
using Random, Distributions
X = rand(Exponential(lambda))
Y = rand(Normal(mu, sigma))
Z = rand(Uniform(a, b))
```

SINTEF

Computers: from uniform to other distributions

- Start with $U \sim \mathcal{U}[0, 1]$
- To get X with CDF F_X, use **inverse transform sampling**:

$$X = F_X^{-1}(U)$$

In Julia:

```
using Random, Distributions
X = rand(Exponential(lambda))
Y = rand(Normal(mu, sigma))
Z = rand(Uniform(a, b))
```

SINTEF

Computers: from uniform to other distributions

- Start with $U \sim \mathcal{U}[0, 1]$
- To get X with CDF F_X, use **inverse transform sampling**:

$$X = F_X^{-1}(U)$$

- Example: Exponential distribution with rate λ

In Julia:

```
using Random, Distributions
X = rand(Exponential(lambda))
Y = rand(Normal(mu, sigma))
Z = rand(Uniform(a, b))
```

SINTEF

Computers: from uniform to other distributions

- Start with $U \sim \mathcal{U}[0, 1]$
- To get X with CDF F_X, use **inverse transform sampling**:

$$X = F_X^{-1}(U)$$

- Example: Exponential distribution with rate λ
  - CDF: $F_X(x) = 1 - e^{-\lambda x}$ for $x \geqslant 0$

In Julia:

```
using Random, Distributions
X = rand(Exponential(lambda))
Y = rand(Normal(mu, sigma))
Z = rand(Uniform(a, b))
```

SINTEF

Computers: from uniform to other distributions

- Start with $U \sim \mathcal{U}[0, 1]$
- To get X with CDF F_X, use **inverse transform sampling**:

$$X = F_X^{-1}(U)$$

- Example: Exponential distribution with rate λ
  - CDF: $F_X(x) = 1 - e^{-\lambda x}$ for $x \geqslant 0$
  - Inverse CDF: $F_X^{-1}(u) = -\frac{1}{\lambda} \ln(1 - u)$

In Julia:

```
using Random, Distributions
X = rand(Exponential(lambda))
Y = rand(Normal(mu, sigma))
Z = rand(Uniform(a, b))
```

SINTEF

Computers: from uniform to other distributions

- Start with $U \sim \mathcal{U}[0, 1]$
- To get X with CDF F_X, use **inverse transform sampling**:

$$X = F_X^{-1}(U)$$

- Example: Exponential distribution with rate λ
  - CDF: $F_X(x) = 1 - e^{-\lambda x}$ for $x \geqslant 0$
  - Inverse CDF: $F_X^{-1}(u) = -\frac{1}{\lambda} \ln(1 - u)$
  - Sample: $X = -\frac{1}{\lambda} \ln(1 - U)$

In Julia:

```
using Random, Distributions
X = rand(Exponential(lambda))
Y = rand(Normal(mu, sigma))
Z = rand(Uniform(a, b))
```

SINTEF

Computers: from uniform to other distributions

- Start with $U \sim \mathcal{U}[0, 1]$
- To get X with CDF F_X, use **inverse transform sampling**:

$$X = F_X^{-1}(U)$$

- Example: Exponential distribution with rate λ
  - CDF: $F_X(x) = 1 - e^{-\lambda x}$ for $x \geqslant 0$
  - Inverse CDF: $F_X^{-1}(u) = -\frac{1}{\lambda} \ln(1 - u)$
  - Sample: $X = -\frac{1}{\lambda} \ln(1 - U)$
- Other methods: Box-Muller transform for normal distribution, rejection sampling, etc.

In Julia:

```
using Random, Distributions
X = rand(Exponential(lambda))
Y = rand(Normal(mu, sigma))
Z = rand(Uniform(a, b))
```

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$
- Each component $Y_i \sim \mathcal{U}[0,1]$ independently

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$
- Each component $Y_i \sim \mathcal{U}[0,1]$ independently
- **Method:** Sample each component independently

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$
- Each component $Y_i \sim \mathcal{U}[0,1]$ independently
- **Method:** Sample each component independently
  1. Generate $U_1, U_2, \ldots, U_d$ from PRNG

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$
- Each component $Y_i \sim \mathcal{U}[0,1]$ independently
- **Method:** Sample each component independently
  1. Generate $U_1, U_2, \ldots, U_d$ from PRNG
  2. Set $\mathbf{Y} = (U_1, U_2, \ldots, U_d)$

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$
- Each component $Y_i \sim \mathcal{U}[0,1]$ independently
- **Method:** Sample each component independently
  1. Generate $U_1, U_2, \ldots, U_d$ from PRNG
  2. Set $\mathbf{Y} = (U_1, U_2, \ldots, U_d)$
- For M independent samples $\mathbf{Y}_1, \ldots, \mathbf{Y}_M$:

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$
- Each component $Y_i \sim \mathcal{U}[0,1]$ independently
- **Method:** Sample each component independently
  1. Generate $U_1, U_2, \ldots, U_d$ from PRNG
  2. Set $\mathbf{Y} = (U_1, U_2, \ldots, U_d)$
- For M independent samples $\mathbf{Y}_1, \ldots, \mathbf{Y}_M$:
  - Generate $M \times d$ independent uniform random numbers

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$
- Each component $Y_i \sim \mathcal{U}[0,1]$ independently
- **Method:** Sample each component independently
  1. Generate $U_1, U_2, \ldots, U_d$ from PRNG
  2. Set $\mathbf{Y} = (U_1, U_2, \ldots, U_d)$
- For M independent samples $\mathbf{Y}_1, \ldots, \mathbf{Y}_M$:
  - Generate $M \times d$ independent uniform random numbers
  - Organize into M vectors of dimension d

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$
- Each component $Y_i \sim \mathcal{U}[0,1]$ independently
- **Method:** Sample each component independently
  1. Generate $U_1, U_2, \ldots, U_d$ from PRNG
  2. Set $\mathbf{Y} = (U_1, U_2, \ldots, U_d)$
- For M independent samples $\mathbf{Y}_1, \ldots, \mathbf{Y}_M$:
  - Generate $M \times d$ independent uniform random numbers
  - Organize into M vectors of dimension d
- **Example in Julia:**

SINTEF

Sampling random vectors in $[0,1]^d$

- Need to generate $\mathbf{Y} = (Y_1, \ldots, Y_d) \in [0,1]^d$
- Each component $Y_i \sim \mathcal{U}[0,1]$ independently
- **Method:** Sample each component independently
  1. Generate $U_1, U_2, \ldots, U_d$ from PRNG
  2. Set $\mathbf{Y} = (U_1, U_2, \ldots, U_d)$
- For M independent samples $\mathbf{Y}_1, \ldots, \mathbf{Y}_M$:
  - Generate $M \times d$ independent uniform random numbers
  - Organize into M vectors of dimension d
- **Example in Julia:**
  - `Y = rand(M, d)  # M samples in R^d`

Monte Carlo algorithm

Goal: Approximate $\mathbb{E}(Q(\mathbf{Y}))$ where $\mathbf{Y} \sim \mathcal{U}([0,1]^d)$

Algorithm:

1. Draw M independent samples $\mathbf{Y}_1, \ldots, \mathbf{Y}_M \sim \mathcal{U}([0,1]^d)$
2. Compute $Q(\mathbf{Y}_k)$ for each $k = 1, \ldots, M$
3. Approximate:

$$\mathbb{E}(Q(\mathbf{Y})) \approx \frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k)$$

Error bound:

$$\mathbb{E}\left(\left| \frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k) - \mathbb{E}(Q(\mathbf{Y})) \right|^2 \right)^{1/2} = \frac{\mathrm{Var}(Q(\mathbf{Y}))^{1/2}}{\sqrt{M}}$$

Key property: Error decreases as $\mathcal{O}(M^{-1/2})$, *independent of dimension d*.

Monte Carlo algorithm

Goal: Approximate $\mathbb{E}(Q(\mathbf{Y}))$ where $\mathbf{Y} \sim \mathcal{U}([0,1]^d)$

Algorithm:

1. Draw M independent samples $\mathbf{Y}_1, \ldots, \mathbf{Y}_M \sim \mathcal{U}([0,1]^d)$
2. Compute $Q(\mathbf{Y}_k)$ for each $k = 1, \ldots, M$
3. Approximate:

$$\mathbb{E}(Q(\mathbf{Y})) \approx \frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k)$$

Error bound:

$$\mathbb{E}\left(\left| \frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k) - \mathbb{E}(Q(\mathbf{Y})) \right|^2 \right)^{1/2} = \frac{\mathrm{Var}(Q(\mathbf{Y}))^{1/2}}{\sqrt{M}}$$

Key property: Error decreases as $\mathcal{O}(M^{-1/2})$, *independent of dimension d.* (Provided the variance stays bounded!)

SINTEF

Monte Carlo for general integrals

- Can approximate *any* integral of the form:

$$\int_D f(\mathbf{y})\, d\mathbf{y}$$

Monte Carlo for general integrals

- Can approximate *any* integral of the form:

$$\int_D f(\mathbf{y})\, d\mathbf{y}$$

- Key idea: Transform to an expectation

SINTEF

Monte Carlo for general integrals

- Can approximate *any* integral of the form:

$$\int_D f(\mathbf{y})\, d\mathbf{y}$$

- Key idea: Transform to an expectation
- If $\mathbf{Y} \sim \mathcal{U}(D)$ (uniform on domain D), then:

$$\int_D f(\mathbf{y})\, d\mathbf{y} = |D| \cdot \mathbb{E}(f(\mathbf{Y}))$$

where $|D|$ is the volume of D

SINTEF

Monte Carlo for general integrals

- Can approximate *any* integral of the form:

$$\int_D f(\mathbf{y})\, d\mathbf{y}$$

- Key idea: Transform to an expectation
- If $\mathbf{Y} \sim \mathcal{U}(D)$ (uniform on domain D), then:

$$\int_D f(\mathbf{y})\, d\mathbf{y} = |D| \cdot \mathbb{E}(f(\mathbf{Y}))$$

  where $|D|$ is the volume of D
- Approximate by sampling:

$$\int_D f(\mathbf{y})\, d\mathbf{y} \approx |D| \cdot \frac{1}{M} \sum_{k=1}^{M} f(\mathbf{Y}_k)$$

  where $\mathbf{Y}_1, \ldots, \mathbf{Y}_M \sim \mathcal{U}(D)$ are i.i.d.

SINTEF

Monte Carlo for general integrals

- Can approximate *any* integral of the form:

$$\int_D f(\mathbf{y})\, d\mathbf{y}$$

- Key idea: Transform to an expectation
- If $\mathbf{Y} \sim \mathcal{U}(D)$ (uniform on domain D), then:

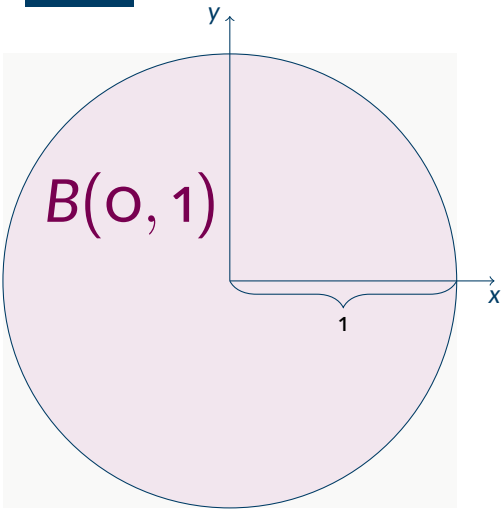
$$\int_D f(\mathbf{y})\, d\mathbf{y} = |D| \cdot \mathbb{E}(f(\mathbf{Y}))$$

  where $|D|$ is the volume of D
- Approximate by sampling:

$$\int_D f(\mathbf{y})\, d\mathbf{y} \approx |D| \cdot \frac{1}{M} \sum_{k=1}^{M} f(\mathbf{Y}_k)$$

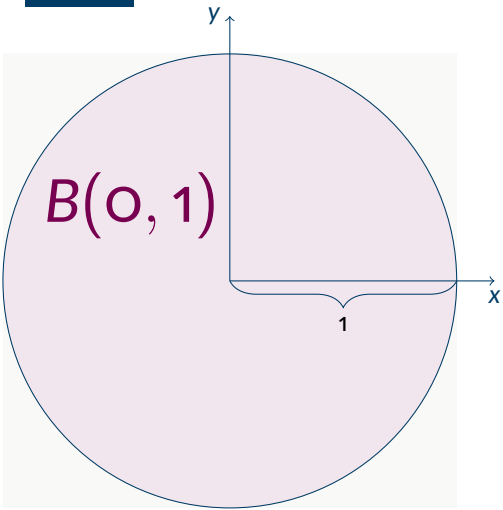
  where $\mathbf{Y}_1, \ldots, \mathbf{Y}_M \sim \mathcal{U}(D)$ are i.i.d.
- For $D = [0,1]^d$: simply $\frac{1}{M} \sum_{k=1}^{M} f(\mathbf{Y}_k)$

Obligatory example: Computing π



$$|B(o,1)| = \pi$$

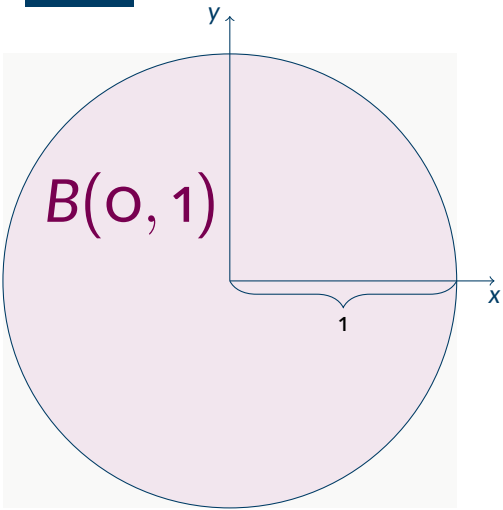
Obligatory example: Computing π



$$|B(o,1)| = \pi$$

$$|B(o,1)| = \int_{-1}^{1}\int_{-1}^{1} \mathbf{1}_{B(o,1)}(x,y)\, dx\, dy$$

Obligatory example: Computing π

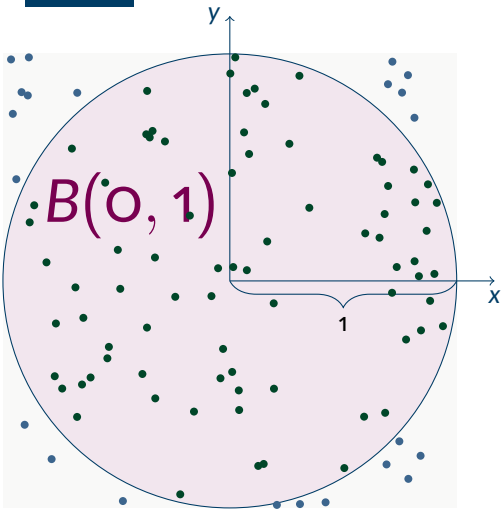


$$|B(o,1)| = \pi$$

$$|B(o,1)| = \int_{-1}^{1}\int_{-1}^{1} \mathbf{1}_{B(o,1)}(x,y)\, dx\, dy$$

$$\Rightarrow \pi = \int_{-1}^{1}\int_{-1}^{1} \mathbf{1}_{B(o,1)}(x,y)\, dx\, dy$$

Obligatory example: Computing π



$$|B(o,1)| = \pi$$

$$|B(o,1)| = \int_{-1}^{1}\int_{-1}^{1} \mathbf{1}_{B(o,1)}(x,y)\, dx\, dy$$

$$\Rightarrow \pi = \int_{-1}^{1}\int_{-1}^{1} \mathbf{1}_{B(o,1)}(x,y)\, dx\, dy$$

$$\approx 4 \cdot \frac{1}{M}\sum_{k=1}^{M} \mathbf{1}_{B(o,1)}(X_k, Y_k)$$

$X_k \sim \mathcal{U}[-1,1], Y_k \sim \mathcal{U}[-1,1]$
independent random variables

SINTEF

Summary in Julia

```
## Drawing random samples:
using Random
using Distributions
X = rand(Uniform(a, b), M)  # M samples from U[a,b]
Y = rand(Normal(mu, sigma), M)   # M samples from N(mu,sigma^2)
```

SINTEF

Summary in Julia

```
## Drawing random samples:
using Random
using Distributions
X = rand(Uniform(a, b), M)  # M samples from U[a,b]
Y = rand(Normal(mu, sigma), M)   # M samples from N(mu,sigma^2)


## Monte Carlo integration:
function monte_carlo_integral(f, M, d)
  sum = 0.0
  for k in 1:M
    y = rand(d)  # Sample in [0,1]^d
    sum += f(y)
  end
  return sum / M  # Approximate integral
end
```

Spatial discretization error

- So far: assumed we can compute $Q(\mathbf{y})$ exactly

SINTEF

Spatial discretization error

- So far: assumed we can compute $Q(\mathbf{y})$ exactly
- In practice: Q often requires spatial discretization

SINTEF

Spatial discretization error

- So far: assumed we can compute $Q(\mathbf{y})$ exactly
- In practice: Q often requires spatial discretization
- Discretized quantity: $Q^{\Delta}(\mathbf{y})$ with mesh size Δ

SINTEF

Spatial discretization error

- So far: assumed we can compute $Q(\mathbf{y})$ exactly
- In practice: Q often requires spatial discretization
- Discretized quantity: $Q^{\Delta}(\mathbf{y})$ with mesh size Δ
- Discretization error bound:

$$\|Q^{\Delta}(\mathbf{y}) - Q(\mathbf{y})\| \leqslant C_{\mathbf{y}} \Delta^{\gamma}$$

for some $\gamma > 0$ (e.g., $\gamma = 2$ for second-order methods)

SINTEF

Spatial discretization error

- So far: assumed we can compute $Q(\mathbf{y})$ exactly
- In practice: Q often requires spatial discretization
- Discretized quantity: $Q^{\Delta}(\mathbf{y})$ with mesh size Δ
- Discretization error bound:

$$\|Q^{\Delta}(\mathbf{y}) - Q(\mathbf{y})\| \leqslant C_{\mathbf{y}} \Delta^{\gamma}$$

  for some $\gamma > 0$ (e.g., $\gamma = 2$ for second-order methods)
- Now have **two sources of error**:

Spatial discretization error

- So far: assumed we can compute $Q(\mathbf{y})$ exactly
- In practice: Q often requires spatial discretization
- Discretized quantity: $Q^{\Delta}(\mathbf{y})$ with mesh size Δ
- Discretization error bound:

$$\|Q^{\Delta}(\mathbf{y}) - Q(\mathbf{y})\| \leqslant C_{\mathbf{y}} \Delta^{\gamma}$$

  for some $\gamma > 0$ (e.g., $\gamma = 2$ for second-order methods)
- Now have **two sources of error**:
  1. Statistical (Monte Carlo): $\mathcal{O}(M^{-1/2})$

Spatial discretization error

- So far: assumed we can compute $Q(\mathbf{y})$ exactly
- In practice: Q often requires spatial discretization
- Discretized quantity: $Q^{\Delta}(\mathbf{y})$ with mesh size Δ
- Discretization error bound:

$$\|Q^{\Delta}(\mathbf{y}) - Q(\mathbf{y})\| \leqslant C_{\mathbf{y}} \Delta^{\gamma}$$

  for some $\gamma > 0$ (e.g., $\gamma = 2$ for second-order methods)
- Now have **two sources of error**:
  1. Statistical (Monte Carlo): $\mathcal{O}(M^{-1/2})$
  2. Discretization: $\mathcal{O}(\Delta^{\gamma})$

Total error with discretization

- Monte Carlo estimator with discretization:

$$\hat{Q}_M^{\Delta} = \frac{1}{M} \sum_{k=1}^{M} Q^{\Delta}(\mathbf{Y}_k)$$

Total error with discretization

- Monte Carlo estimator with discretization:

$$\hat{Q}_M^{\Delta} = \frac{1}{M} \sum_{k=1}^{M} Q^{\Delta}(\mathbf{Y}_k)$$

- Target: $\mathbb{E}(Q(\mathbf{Y}))$

Total error with discretization

- Monte Carlo estimator with discretization:

$$\hat{Q}_M^\Delta = \frac{1}{M}\sum_{k=1}^{M} Q^\Delta(\mathbf{Y}_k)$$

- Target: $\mathbb{E}(Q(\mathbf{Y}))$
- Total error decomposes as:

$$\begin{aligned}\left|\hat{Q}_M^\Delta - \mathbb{E}(Q(\mathbf{Y}))\right| &= \left|\hat{Q}_M^\Delta - \mathbb{E}(Q^\Delta(\mathbf{Y})) + \mathbb{E}(Q^\Delta(\mathbf{Y})) - \mathbb{E}(Q(\mathbf{Y}))\right| \\ &\leqslant \underbrace{\left|\hat{Q}_M^\Delta - \mathbb{E}(Q^\Delta(\mathbf{Y}))\right|}_{\text{Statistical error}} + \underbrace{\left|\mathbb{E}(Q^\Delta(\mathbf{Y})) - \mathbb{E}(Q(\mathbf{Y}))\right|}_{\text{Discretization bias}}\end{aligned}$$

SINTEF

Total error with discretization

- Monte Carlo estimator with discretization:

$$\hat{Q}_M^\Delta = \frac{1}{M} \sum_{k=1}^{M} Q^\Delta(\mathbf{Y}_k)$$

- Target: $\mathbb{E}(Q(\mathbf{Y}))$
- Total error decomposes as:

$$\begin{aligned} \left|\hat{Q}_M^\Delta - \mathbb{E}(Q(\mathbf{Y}))\right| &= \left|\hat{Q}_M^\Delta - \mathbb{E}(Q^\Delta(\mathbf{Y})) + \mathbb{E}(Q^\Delta(\mathbf{Y})) - \mathbb{E}(Q(\mathbf{Y}))\right| \\ &\leqslant \underbrace{\left|\hat{Q}_M^\Delta - \mathbb{E}(Q^\Delta(\mathbf{Y}))\right|}_{\text{Statistical error}} + \underbrace{\left|\mathbb{E}(Q^\Delta(\mathbf{Y})) - \mathbb{E}(Q(\mathbf{Y}))\right|}_{\text{Discretization bias}} \end{aligned}$$

- Statistical error (in expectation): $\mathcal{O}(M^{-1/2})$

Total error with discretization

- Monte Carlo estimator with discretization:

$$\hat{Q}_M^\Delta = \frac{1}{M} \sum_{k=1}^{M} Q^\Delta(\mathbf{Y}_k)$$

- Target: $\mathbb{E}(Q(\mathbf{Y}))$
- Total error decomposes as:

$$\begin{aligned} \left|\hat{Q}_M^\Delta - \mathbb{E}(Q(\mathbf{Y}))\right| &= \left|\hat{Q}_M^\Delta - \mathbb{E}(Q^\Delta(\mathbf{Y})) + \mathbb{E}(Q^\Delta(\mathbf{Y})) - \mathbb{E}(Q(\mathbf{Y}))\right| \\ &\leqslant \underbrace{\left|\hat{Q}_M^\Delta - \mathbb{E}(Q^\Delta(\mathbf{Y}))\right|}_{\text{Statistical error}} + \underbrace{\left|\mathbb{E}(Q^\Delta(\mathbf{Y})) - \mathbb{E}(Q(\mathbf{Y}))\right|}_{\text{Discretization bias}} \end{aligned}$$

- Statistical error (in expectation): $\mathcal{O}(M^{-1/2})$
- Discretization bias: $\leqslant \mathbb{E}(C_\mathbf{Y})\Delta^\gamma = \mathcal{O}(\Delta^\gamma)$

Total error with discretization

- Monte Carlo estimator with discretization:

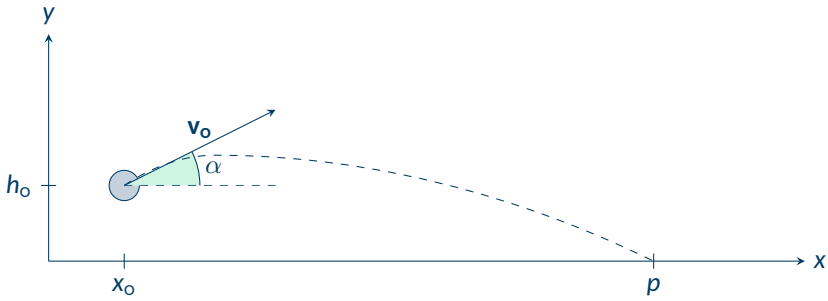
$$\hat{Q}_M^\Delta = \frac{1}{M}\sum_{k=1}^{M} Q^\Delta(\mathbf{Y}_k)$$

- Target: $\mathbb{E}(Q(\mathbf{Y}))$
- Total error decomposes as:

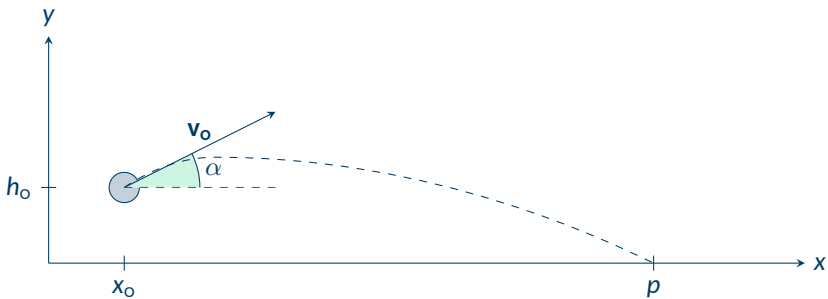
$$\begin{aligned}\left|\hat{Q}_M^\Delta - \mathbb{E}(Q(\mathbf{Y}))\right| &= \left|\hat{Q}_M^\Delta - \mathbb{E}(Q^\Delta(\mathbf{Y})) + \mathbb{E}(Q^\Delta(\mathbf{Y})) - \mathbb{E}(Q(\mathbf{Y}))\right| \\ &\leqslant \underbrace{\left|\hat{Q}_M^\Delta - \mathbb{E}(Q^\Delta(\mathbf{Y}))\right|}_{\text{Statistical error}} + \underbrace{\left|\mathbb{E}(Q^\Delta(\mathbf{Y})) - \mathbb{E}(Q(\mathbf{Y}))\right|}_{\text{Discretization bias}}\end{aligned}$$

- Statistical error (in expectation): $\mathcal{O}(M^{-1/2})$
- Discretization bias: $\leqslant \mathbb{E}(C_\mathbf{Y})\Delta^\gamma = \mathcal{O}(\Delta^\gamma)$
- **Balance errors:** Choose $M \sim \Delta^{-2\gamma}$ to get total error $\mathcal{O}(\Delta^\gamma)$

Example



Example



$$p(h_o) = v_o \cos(\alpha) \frac{v_o \sin(\alpha) + \sqrt{(v_o \sin(\alpha))^2 + 2gh_o}}{g} + x_o$$

SINTEF

Interesting quantities

Assume $h_0 \sim \mathcal{U}[0, 1]$

SINTEF

Interesting quantities

Assume $h_0 \sim \mathcal{U}[0, 1]$

- Expectation $\mathbb{E}(p) = \int_0^1 p(h_0)\, dh_0$

SINTEF

Interesting quantities

Assume $h_o \sim \mathcal{U}[0, 1]$

- Expectation $\mathbb{E}(p) = \int_0^1 p(h_o)\, dh_o$
- Variance $\mathrm{Var}(p) = \mathbb{E}((p - \mathbb{E}(p))^2) = \int_0^1 (p(h_o) - \mathbb{E}(p(h_o)))^2\, dh_o$

SINTEF

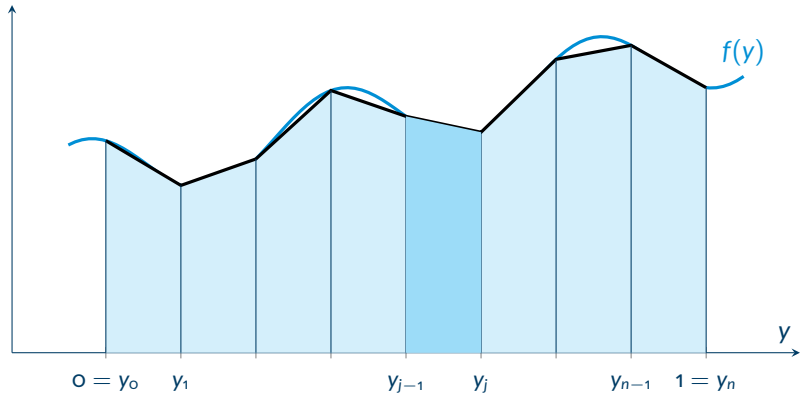
Interesting quantities

Assume $h_o \sim \mathcal{U}[0, 1]$

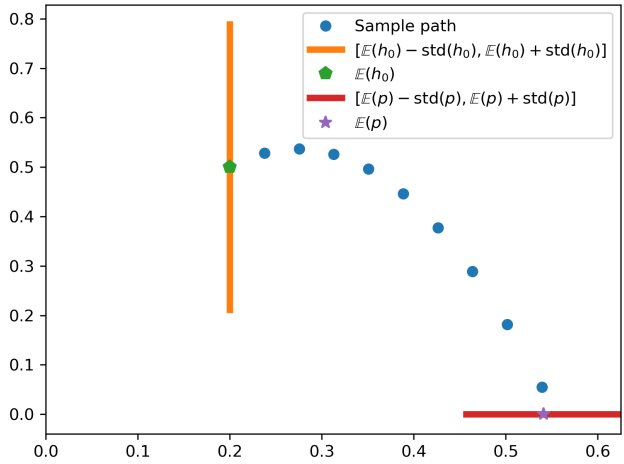
- Expectation $\mathbb{E}(p) = \int_o^1 p(h_o)\, dh_o$
- Variance $\mathrm{Var}(p) = \mathbb{E}((p - \mathbb{E}(p))^2) = \int_o^1 (p(h_o) - \mathbb{E}(p(h_o)))^2\, dh_o$

All quantities involving the integral!

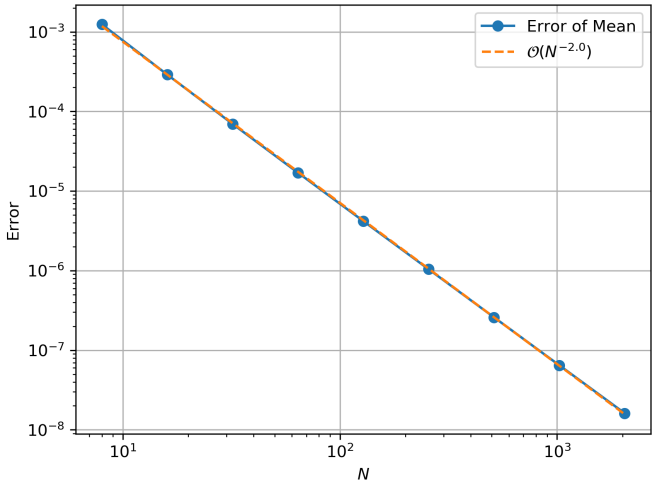
Approximate by Trapezoidal rule



Results varying initial height



Error versus number of evaluations



Making the problem more complicated

- Keep $h_0 \sim \mathcal{U}[0, 1]$

SINTEF

Making the problem more complicated

- Keep $h_0 \sim \mathcal{U}[0, 1]$
- Let the angle $\alpha \sim \mathcal{U}[0, 3\pi/6]$

SINTEF

Making the problem more complicated

- Keep $h_0 \sim \mathcal{U}[0, 1]$
- Let the angle $\alpha \sim \mathcal{U}[0, 3\pi/6]$
- Now two parameters: h_0 and α

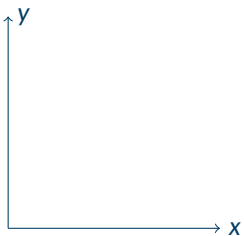
SINTEF

Making the problem more complicated

- Keep $h_o \sim \mathcal{U}[0, 1]$
- Let the angle $\alpha \sim \mathcal{U}[0, 3\pi/6]$
- Now two parameters: h_o and α
- Integration in 2D?

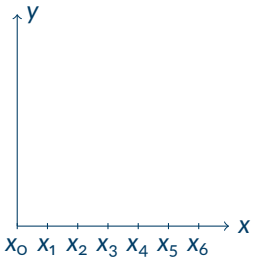
Trapezoidal rule for two dimensions

$$\int_0^1 \int_0^1 f(x, y)\, dx\, dy$$



Trapezoidal rule for two dimensions

$$\int_0^1 \int_0^1 f(x,y)\,dx\,dy \approx \int_0^1 \left(f(x_0,y) + 2\sum_{i=1}^{N-1} f(x_i,y) + f(x_N,y) \right) \Delta x\,dy$$

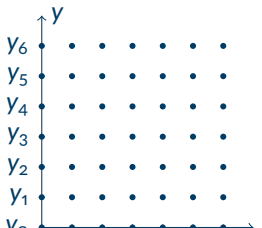


Trapezoidal rule for two dimensions

$$\int_0^1 \int_0^1 f(x,y)\,dx\,dy \approx \int_0^1 \left(f(x_0,y) + 2\sum_{i=1}^{N-1} f(x_i,y) + f(x_N,y) \right) \Delta x\,dy$$

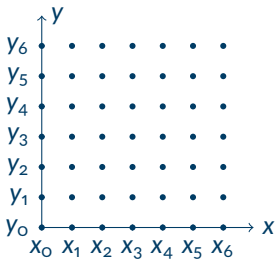
$$\approx \Big(f(0,0) + f(1,0) + f(0,1) + f(1,1) + 4\sum_{i=1}^{N-1}\sum_{i=1}^{N-1} f(x_i,y_i)$$

$$+ 2\sum_{i=1}^{N-1} f(x_i,0) + 2\sum_{i=1}^{N-1} f(x_i,1) + 2\sum_{i=1}^{N-1} f(1,y_i) + 2\sum_{i=1}^{N-1} f(0,y_i) \Big)$$



Trapezoidal rule for two dimensions

$$\int_0^1 \int_0^1 f(x,y)\,dx\,dy \approx \sum_{i=0}^{N}\sum_{j=0}^{N} w_{i,j} f(x_i, y_j)$$



SINTEF

Even more complications

- Keep $h_0 \sim \mathcal{U}[0, 1]$, $\alpha \sim \mathcal{U}[0, 5\pi/12]$

SINTEF

Even more complications

- Keep $h_0 \sim \mathcal{U}[0, 1]$, $\alpha \sim \mathcal{U}[0, 5\pi/12]$
- Initial velocity, $v_0 \sim \mathcal{U}[20, 40]$

SINTEF

Even more complications

- Keep $h_0 \sim \mathcal{U}[0, 1]$, $\alpha \sim \mathcal{U}[0, 5\pi/12]$
- Initial velocity, $v_0 \sim \mathcal{U}[20, 40]$
- Air resistance $F_D = \frac{1}{2}\rho v^2 C_D \pi r^2$, where

SINTEF

Even more complications

- Keep $h_0 \sim \mathcal{U}[0, 1]$, $\alpha \sim \mathcal{U}[0, 5\pi/12]$
- Initial velocity, $v_0 \sim \mathcal{U}[20, 40]$
- Air resistance $F_D = \frac{1}{2}\rho v^2 C_D \pi r^2$, where
  - $C_D \sim \mathcal{U}[0.09, 0.11]$ (Drag coefficient)

SINTEF

Even more complications

- Keep $h_0 \sim \mathcal{U}[0, 1]$, $\alpha \sim \mathcal{U}[0, 5\pi/12]$
- Initial velocity, $v_0 \sim \mathcal{U}[20, 40]$
- Air resistance $F_D = \frac{1}{2}\rho v^2 C_D \pi r^2$, where
  - $C_D \sim \mathcal{U}[0.09, 0.11]$ (Drag coefficient)
  - $r \sim \mathcal{U}[0.22, 0.235]$ (Radius of ball)

SINTEF

Even more complications

- Keep $h_0 \sim \mathcal{U}[0, 1]$, $\alpha \sim \mathcal{U}[0, 5\pi/12]$
- Initial velocity, $v_0 \sim \mathcal{U}[20, 40]$
- Air resistance $F_D = \frac{1}{2}\rho v^2 C_D \pi r^2$, where
  - $C_D \sim \mathcal{U}[0.09, 0.11]$ (Drag coefficient)
  - $r \sim \mathcal{U}[0.22, 0.235]$ (Radius of ball)
  - $\rho \sim \mathcal{U}[1.1, 1.4]$

SINTEF

Even more complications

- Keep $h_0 \sim \mathcal{U}[0, 1]$, $\alpha \sim \mathcal{U}[0, 5\pi/12]$
- Initial velocity, $v_0 \sim \mathcal{U}[20, 40]$
- Air resistance $F_D = \frac{1}{2}\rho v^2 C_D \pi r^2$, where
  - $C_D \sim \mathcal{U}[0.09, 0.11]$ (Drag coefficient)
  - $r \sim \mathcal{U}[0.22, 0.235]$ (Radius of ball)
  - $\rho \sim \mathcal{U}[1.1, 1.4]$
- Mass $m \sim \mathcal{U}[0.142, 0.149]$

SINTEF

Even more complications

- Keep $h_0 \sim \mathcal{U}[0, 1]$, $\alpha \sim \mathcal{U}[0, 5\pi/12]$
- Initial velocity, $v_0 \sim \mathcal{U}[20, 40]$
- Air resistance $F_D = \frac{1}{2}\rho v^2 C_D \pi r^2$, where
  - $C_D \sim \mathcal{U}[0.09, 0.11]$ (Drag coefficient)
  - $r \sim \mathcal{U}[0.22, 0.235]$ (Radius of ball)
  - $\rho \sim \mathcal{U}[1.1, 1.4]$
- Mass $m \sim \mathcal{U}[0.142, 0.149]$
- **Seven** parameters

SINTEF

Landing position depends on 7 parameters

$$p(h_o, \alpha, v_o, C_D, r, \rho, m) = \ldots$$

SINTEF

Landing position depends on 7 parameters

$$p(h_0, \alpha, v_0, C_D, r, \rho, m) = \ldots$$

$$\mathbb{E}(p) = \int_0^1 \int_0^{5\pi/12} \int_{20}^{40} \int_{0.09}^{0.11} \int_{0.22}^{0.235} \int_{1.1}^{1.4} \int_{0.142}^{0.149} \ldots$$
$$p(h_0, \alpha, v_0, C_D, r, \rho, m) \frac{dm\ dr\ dC_D\ dv_0\ d\alpha\ dh_0}{20 \cdot 5\pi/12 \cdot 0.2 \cdot 0.015 \cdot 0.3 \cdot 0.07}$$

SINTEF

How many evaluations of p do we need?

- We have 7 parameters ($d = 7$)
- Trapezoidal rule error

$$\text{Error} \leqslant Ch^2$$

Require Error $< 1/25 \approx 4\%$

- $\Rightarrow h \leqslant \sqrt{1/25} = 1/5$

SINTEF

How many evaluations of p do we need?

- We have 7 parameters ($d = 7$)
- Trapezoidal rule error

$$\text{Error} \leqslant Ch^2$$

Require Error $< 1/25 \approx 4\%$

- $\Rightarrow h \leqslant \sqrt{1/25} = 1/5$
- $\Rightarrow N \geqslant 5$

How many evaluations of p do we need?

- We have 7 parameters ($d = 7$)
- Trapezoidal rule error

$$\text{Error} \leqslant Ch^2$$

Require Error $< 1/25 \approx 4\%$

- $\Rightarrow h \leqslant \sqrt{1/25} = 1/5$
- $\Rightarrow N \geqslant 5$
- $\Rightarrow N^d = N^7 = 5^7 = 78125$ evaluations

Finding p

Governing ODE

$$\begin{cases} \dfrac{d^2}{dt^2}\mathbf{x}(t) = -\mathbf{F}_D(C_D, r, \rho, \dfrac{d}{dt}\mathbf{x}(t))/m - g\mathbf{e}_2 \\ \mathbf{x}(0) = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \\ \mathbf{x}'(0) = \begin{pmatrix} v_0 \cos(\alpha) \\ v_0 \sin(\alpha) \end{pmatrix} \end{cases}$$

SINTEF

Finding p

$\mathbf{x}(t)$ depends on the parameters

$$\begin{cases} \dfrac{d^2}{dt^2}\mathbf{x}(h_o, \alpha, v_o, C_D, r, \rho, m; t) = -\mathbf{F}_D(C_D, r, \rho, \dfrac{d}{dt}\mathbf{x}(\ldots; t))/m - g\mathbf{e}_2 \\ \mathbf{x}(h_o, \alpha, v_o, C_D, r, \rho, m; 0) = \begin{pmatrix} 0 \\ y_o \end{pmatrix} \\ \dfrac{d}{dt}\mathbf{x}(h_o, \alpha, v_o, C_D, r, \rho, m; 0) = \begin{pmatrix} v_o \cos(\alpha) \\ v_o \sin(\alpha) \end{pmatrix} \end{cases}$$

Finding p

Assuming underlying probability space Ω

$$\begin{cases} \dfrac{d^2}{dt^2}\mathbf{x}(\omega; t) = -\mathbf{F}_D(C_D(\omega), r(\omega), \rho(\omega), \dfrac{d}{dt}\mathbf{x}(\omega; t))/m(\omega) - g\mathbf{e}_2 \\ \mathbf{x}(\omega; 0) = \begin{pmatrix} 0 \\ y_0(\omega) \end{pmatrix} \\ \dfrac{d}{dt}\mathbf{x}(\omega; 0) = \begin{pmatrix} v_0(\omega)\cos(\alpha(\omega)) \\ v_0(\omega)\sin(\alpha(\omega)) \end{pmatrix} \end{cases}$$

for $\omega \in \Omega$.

Finding p

Assuming underlying probability space Ω

$$\begin{cases} \dfrac{d^2}{dt^2}\mathbf{x}(\omega;t) = -\mathbf{F}_D(C_D(\omega), r(\omega), \rho(\omega), \dfrac{d}{dt}\mathbf{x}(\omega;t))/m(\omega) - g\mathbf{e}_2 \\ \mathbf{x}(\omega;0) = \begin{pmatrix} 0 \\ y_0(\omega) \end{pmatrix} \\ \dfrac{d}{dt}\mathbf{x}(\omega;0) = \begin{pmatrix} v_0(\omega)\cos(\alpha(\omega)) \\ v_0(\omega)\sin(\alpha(\omega)) \end{pmatrix} \end{cases}$$

for $\omega \in \Omega$. Solve using eg. Forward-Euler:

$$\begin{aligned} \mathbf{x}^{n,\Delta t}(\omega) &= \mathbf{x}^{n-1,\Delta t}(\omega) + \Delta t \mathbf{v}^{n-1,\Delta t}(\omega) \\ \mathbf{v}^{n,\Delta t}(\omega) &= \mathbf{v}^{n-1,\Delta t}(\omega) + \Delta t \left(-\mathbf{F}_D(C_D(\omega), r(\omega), \rho(\omega), \mathbf{v}^{n-1,\Delta t})/m(\omega) - g\mathbf{e}_2\right) \end{aligned}$$

Finding p

Assuming underlying probability space Ω

$$\begin{cases} \dfrac{d^2}{dt^2}\mathbf{x}(\omega;t) = -\mathbf{F}_D(C_D(\omega), r(\omega), \rho(\omega), \dfrac{d}{dt}\mathbf{x}(\omega;t))/m(\omega) - g\mathbf{e}_2 \\ \mathbf{x}(\omega;0) = \begin{pmatrix} 0 \\ y_0(\omega) \end{pmatrix} \\ \dfrac{d}{dt}\mathbf{x}(\omega;0) = \begin{pmatrix} v_0(\omega)\cos(\alpha(\omega)) \\ v_0(\omega)\sin(\alpha(\omega)) \end{pmatrix} \end{cases}$$

for $\omega \in \Omega$. Solve using eg. Forward-Euler:
Until $y^{n,\Delta t}(\omega) < 0$

$$\mathbf{x}^{n,\Delta t}(\omega) = \mathbf{x}^{n-1,\Delta t}(\omega) + \Delta t \mathbf{v}^{n-1,\Delta t}(\omega)$$
$$\mathbf{v}^{n,\Delta t}(\omega) = \mathbf{v}^{n-1,\Delta t}(\omega) + \Delta t \left(-\mathbf{F}_D(C_D(\omega), r(\omega), \rho(\omega), \mathbf{v}^{n-1,\Delta t})/m(\omega) - g\mathbf{e}_2\right)$$

Finding p

Assuming underlying probability space Ω

$$\begin{cases} \dfrac{d^2}{dt^2}\mathbf{x}(\omega;t) = -\mathbf{F}_D(C_D(\omega), r(\omega), \rho(\omega), \dfrac{d}{dt}\mathbf{x}(\omega;t))/m(\omega) - g\mathbf{e}_2 \\ \mathbf{x}(\omega;0) = \begin{pmatrix} 0 \\ y_0(\omega) \end{pmatrix} \\ \dfrac{d}{dt}\mathbf{x}(\omega;0) = \begin{pmatrix} v_0(\omega)\cos(\alpha(\omega)) \\ v_0(\omega)\sin(\alpha(\omega)) \end{pmatrix} \end{cases}$$

for $\omega \in \Omega$. Solve using eg. Forward-Euler:
Until $y^{n,\Delta t}(\omega) < 0$

$$\mathbf{x}^{n,\Delta t}(\omega) = \mathbf{x}^{n-1,\Delta t}(\omega) + \Delta t \mathbf{v}^{n-1,\Delta t}(\omega)$$
$$\mathbf{v}^{n,\Delta t}(\omega) = \mathbf{v}^{n-1,\Delta t}(\omega) + \Delta t \left(-\mathbf{F}_D(C_D(\omega), r(\omega), \rho(\omega), \mathbf{v}^{n-1,\Delta t})/m(\omega) - g\mathbf{e}_2\right)$$

Set $p^{\Delta t}(\omega) = y^{N,\Delta t}(\omega)$.

Algorithm: Compute expected position using Trapezoidal rule

Input: $\Delta t, \Delta h_o, \Delta\alpha, \Delta v_o \ \ldots$

1. Initialize mean $E = 0$
2. For each integration point $h_o^i,\ \alpha^j,\ v_o^k, \ldots$:
   - 2.1 Use forward Euler to compute
   $$p^{\Delta t}(h_o^i, \alpha^j, v_o^k, \ldots)$$
   - 2.2 Update mean
   $$E = E + w_{i,j,k,\ldots} \cdot p^{\Delta t}(h_o^i, \alpha^j, v_o^k, \ldots)$$

Each evaluation is expensive when done 100 000 times!

Runtime to obtain an error ϵ:

SINTEF

Each evaluation is expensive when done 100 000 times!

Runtime to obtain an error ϵ:

- Forward Euler: $\mathcal{O}(\epsilon^{-1})$

SINTEF

Each evaluation is expensive when done 100 000 times!

Runtime to obtain an error ϵ:

- Forward Euler: $\mathcal{O}(\epsilon^{-1})$
- Trapezoidal in 7D: $\mathcal{O}(\epsilon^{-7/2})$

SINTEF

Each evaluation is expensive when done 100 000 times!

Runtime to obtain an error ϵ:

- Forward Euler: $\mathcal{O}(\epsilon^{-1})$
- Trapezoidal in 7D: $\mathcal{O}(\epsilon^{-7/2})$
- Total cost:

$$\overbrace{\mathcal{O}(\epsilon^{-1})}^{\text{Forward Euler}} \cdot \underbrace{\mathcal{O}(\epsilon^{-7/2})}_{\text{Trapezoidal}} = \mathcal{O}(\epsilon^{-4.5}).$$

SINTEF

Higher order quadratures?

- For every dimension k, we could find integration rules such that

$$\text{Error} \leqslant Ch^k$$

SINTEF

Higher order quadratures?

- For every dimension k, we could find integration rules such that

$$\text{Error} \leqslant Ch^k$$

- Then to obtain an error $\mathcal{O}(\epsilon)$, we choose $h = \mathcal{O}(\sqrt[k]{\epsilon})$

SINTEF

Higher order quadratures?

- For every dimension k, we could find integration rules such that

$$\text{Error} \leqslant Ch^k$$

- Then to obtain an error $\mathcal{O}(\epsilon)$, we choose $h = \mathcal{O}(\sqrt[k]{\epsilon})$
- $N = \mathcal{O}(\epsilon^{-1/k}) \Rightarrow M = \mathcal{O}(\epsilon^{-1})$

SINTEF

Higher order quadratures?

- For every dimension k, we could find integration rules such that

$$\text{Error} \leqslant Ch^k$$

- Then to obtain an error $\mathcal{O}(\epsilon)$, we choose $h = \mathcal{O}(\sqrt[k]{\epsilon})$
- $N = \mathcal{O}(\epsilon^{-1/k}) \Rightarrow M = \mathcal{O}(\epsilon^{-1})$
- Good idea?

SINTEF

Higher order quadratures?

- For every dimension k, we could find integration rules such that

$$\text{Error} \leqslant Ch^k$$

- Then to obtain an error $\mathcal{O}(\epsilon)$, we choose $h = \mathcal{O}(\sqrt[k]{\epsilon})$
- $N = \mathcal{O}(\epsilon^{-1/k}) \Rightarrow M = \mathcal{O}(\epsilon^{-1})$
- Good idea?
  - Hard to implement

SINTEF

Higher order quadratures?

- For every dimension k, we could find integration rules such that

$$\text{Error} \leqslant Ch^k$$

- Then to obtain an error $\mathcal{O}(\epsilon)$, we choose $h = \mathcal{O}(\sqrt[k]{\epsilon})$
- $N = \mathcal{O}(\epsilon^{-1/k}) \Rightarrow M = \mathcal{O}(\epsilon^{-1})$
- Good idea?
  - Hard to implement
  - Large constant

SINTEF

Higher order quadratures?

- For every dimension k, we could find integration rules such that

$$\text{Error} \leqslant Ch^k$$

- Then to obtain an error $\mathcal{O}(\epsilon)$, we choose $h = \mathcal{O}(\sqrt[k]{\epsilon})$
- $N = \mathcal{O}(\epsilon^{-1/k}) \Rightarrow M = \mathcal{O}(\epsilon^{-1})$
- Good idea?
  - Hard to implement
  - Large constant
  - Need a stencil of size at least k

Higher order quadratures?

- For every dimension k, we could find integration rules such that

$$\text{Error} \leqslant Ch^k$$

- Then to obtain an error $\mathcal{O}(\epsilon)$, we choose $h = \mathcal{O}(\sqrt[k]{\epsilon})$
- $N = \mathcal{O}(\epsilon^{-1/k}) \Rightarrow M = \mathcal{O}(\epsilon^{-1})$
- Good idea?
  - Hard to implement
  - Large constant
  - Need a stencil of size at least k
  - Oscillation issuess

SINTEF

Algorithm: Using Monte Carlo for ODE

$$\begin{cases} u'(\omega; t) = F(u(\omega, t)) \\ u(\omega, 0) = u_0(\omega) \end{cases}$$

Approximate: $\mathbb{E}(u(\cdot; T))$

Algorithm: Using Monte Carlo for ODE

$$\begin{cases} u'(\omega; t) = F(u(\omega, t)) \\ u(\omega, 0) = u_0(\omega) \end{cases}$$

Approximate: $\mathbb{E}(u(\cdot; T))$ Input: Δt, M

1. Initialize mean $E = 0$

Algorithm: Using Monte Carlo for ODE

$$\begin{cases} u'(\omega; t) = F(u(\omega, t)) \\ u(\omega, 0) = u_0(\omega) \end{cases}$$

Approximate: $\mathbb{E}(u(\cdot; T))$ Input: Δt, M

1. Initialize mean $E = 0$
2. Draw M i.i.d samples $u_0^1, \ldots, u_0^k, \ldots, u_0^M$.

Algorithm: Using Monte Carlo for ODE

$$\begin{cases} u'(\omega; t) = F(u(\omega, t)) \\ u(\omega, 0) = u_0(\omega) \end{cases}$$

Approximate: $\mathbb{E}(u(\cdot; T))$ Input: Δt, M

1. Initialize mean $E = 0$
2. Draw M i.i.d samples $u_0^1, \ldots, u_0^k, \ldots, u_0^M$.
3. For each $k = 1, \ldots, M$:
   3.1 Compute (with eg Forward Euler):

$$u^k_{\Delta T, N} = \mathcal{F}(u_0^k)$$

Algorithm: Using Monte Carlo for ODE

$$\begin{cases} u'(\omega; t) = F(u(\omega, t)) \\ u(\omega, 0) = u_0(\omega) \end{cases}$$

Approximate: $\mathbb{E}(u(\cdot; T))$ Input: Δt, M

1. Initialize mean $E = 0$
2. Draw M i.i.d samples $u_0^1, \ldots, u_0^k, \ldots, u_0^M$.
3. For each $k = 1, \ldots, M$:
   3.1 Compute (with eg Forward Euler):
   $$u^k_{\Delta T, N} = \mathcal{F}(u_0^k)$$
   3.2 Update mean
   $$E = E + u^k_{\Delta T, N}$$

Algorithm: Using Monte Carlo for ODE

$$\begin{cases} u'(\omega; t) = F(u(\omega, t)) \\ u(\omega, 0) = u_0(\omega) \end{cases}$$

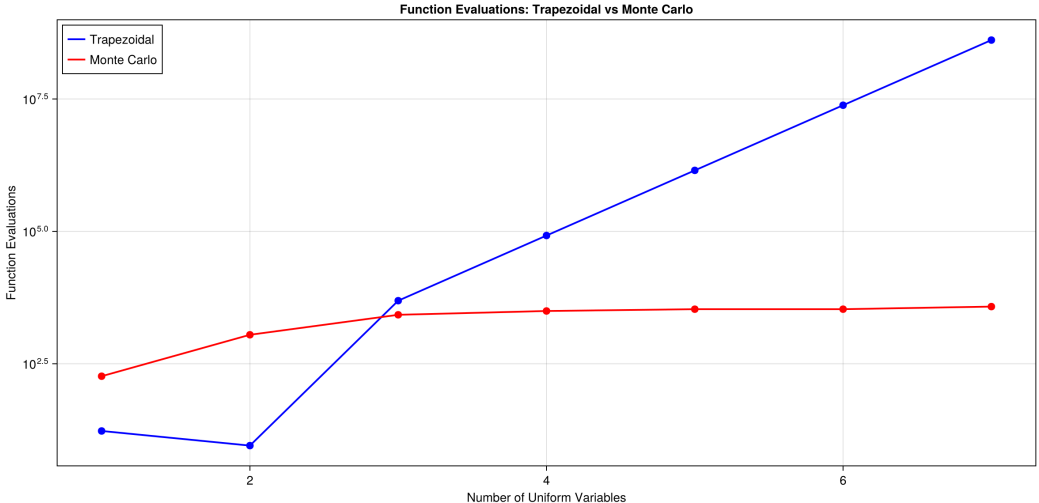
Approximate: $\mathbb{E}(u(\cdot; T))$ Input: Δt, M

1. Initialize mean $E = 0$
2. Draw M i.i.d samples $u_0^1, \ldots, u_0^k, \ldots, u_0^M$.
3. For each $k = 1, \ldots, M$:
   3.1 Compute (with eg Forward Euler):
   $$u_{\Delta T,N}^k = \mathcal{F}(u_0^k)$$
   3.2 Update mean
   $$E = E + u_{\Delta T,N}^k$$

Then

$$E \approx \mathbb{E}(u(\cdot, T))$$

Function evaluations comparison: Trapezoidal vs Monte Carlo



Various smaller remarks

SINTEF

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

SINTEF

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

- Naive approach: Store all samples $X_1, \ldots, X_M$, then compute

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

- Naive approach: Store all samples $X_1, \ldots, X_M$, then compute
- Issues with storing samples:

SINTEF

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

- Naive approach: Store all samples $X_1, \ldots, X_M$, then compute
- Issues with storing samples:
  - Memory: M samples $\times$ size of each sample

SINTEF

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

- Naive approach: Store all samples $X_1, \ldots, X_M$, then compute
- Issues with storing samples:
  - Memory: M samples $\times$ size of each sample
  - Storage admins are *not happy* when you store many/large files!

SINTEF

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

- Naive approach: Store all samples $X_1, \ldots, X_M$, then compute
- Issues with storing samples:
  - Memory: M samples $\times$ size of each sample
  - Storage admins are *not happy* when you store many/large files!
  - Example: $M = 10^6$ samples, each 1GB $\Rightarrow$ 1PB storage

SINTEF

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

- Naive approach: Store all samples $X_1, \ldots, X_M$, then compute
- Issues with storing samples:
  - Memory: M samples $\times$ size of each sample
  - Storage admins are *not happy* when you store many/large files!
  - Example: $M = 10^6$ samples, each 1GB $\Rightarrow$ 1PB storage
- **Solution:** Online (streaming) algorithms

SINTEF

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

- Naive approach: Store all samples $X_1, \ldots, X_M$, then compute
- Issues with storing samples:
  - Memory: M samples $\times$ size of each sample
  - Storage admins are *not happy* when you store many/large files!
  - Example: $M = 10^6$ samples, each 1GB $\Rightarrow$ 1PB storage
- **Solution:** Online (streaming) algorithms
- Update mean and variance incrementally without storing samples

SINTEF

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

- Naive approach: Store all samples $X_1, \ldots, X_M$, then compute
- Issues with storing samples:
  - Memory: M samples $\times$ size of each sample
  - Storage admins are *not happy* when you store many/large files!
  - Example: $M = 10^6$ samples, each 1GB $\Rightarrow$ 1PB storage
- **Solution:** Online (streaming) algorithms
- Update mean and variance incrementally without storing samples

Computing variance and means online

- Problem: Computing variance requires two passes over data. Alternatively:

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$ Sensitive to cancellation

- Naive approach: Store all samples $X_1, \ldots, X_M$, then compute
- Issues with storing samples:
  - Memory: M samples $\times$ size of each sample
  - Storage admins are *not happy* when you store many/large files!
  - Example: $M = 10^6$ samples, each 1GB $\Rightarrow$ 1PB storage
- **Solution:** Online (streaming) algorithms
- Update mean and variance incrementally without storing samples

Welford's online algorithm:

$$M_k = M_{k-1} + \frac{X_k - M_{k-1}}{k} \qquad \text{(running mean)}$$

$$S_k = S_{k-1} + (X_k - M_{k-1})(X_k - M_k) \qquad \text{(running sum of squares)}$$

Then: $\mathbb{E}(X) \approx M_k$, $\mathrm{Var}(X) \approx S_k/(k-1)$

Storage: Only need to keep M_k and S_k (constant memory!)

SINTEF

Programming tips: Use an explicit random generator

- `seed!`, —randn()—, `rand()`, etc modify global state

Prefer to use an explicit random number generator (available in Julia, Python, C++, ...):

```
using Random
using Distributions
rng = MersenneTwister(1234)
U = rand(rng)
X = rand(rng, Exponential(lambda))
Y = rand(rng, Normal(mu, sigma))
Z = rand(rng, Uniform(a, b), (10, 10))
```

SINTEF

Programming tips: Use an explicit random generator

- `seed!`, —randn()—, `rand()`, etc modify global state
- Not thread safe

Prefer to use an explicit random number generator (available in Julia, Python, C++, ...):

```
using Random
using Distributions
rng = MersenneTwister(1234)
U = rand(rng)
X = rand(rng, Exponential(lambda))
Y = rand(rng, Normal(mu, sigma))
Z = rand(rng, Uniform(a, b), (10, 10))
```

SINTEF

Programming tips: Use an explicit random generator

- `seed!`, —randn()—, `rand()`, etc modify global state
- Not thread safe
- What happens if other parts of the code also use random numbers?

Prefer to use an explicit random number generator (available in Julia, Python, C++, ...):

```
using Random
using Distributions
rng = MersenneTwister(1234)
U = rand(rng)
X = rand(rng, Exponential(lambda))
Y = rand(rng, Normal(mu, sigma))
Z = rand(rng, Uniform(a, b), (10, 10))
```

Programming tips: Use an explicit random generator

- `seed!`, —randn()—, `rand()`, etc modify global state
- Not thread safe
- What happens if other parts of the code also use random numbers?
- Difficult to reproduce results

Prefer to use an explicit random number generator (available in Julia, Python, C++, ...):

```
using Random
using Distributions
rng = MersenneTwister(1234)
U = rand(rng)
X = rand(rng, Exponential(lambda))
Y = rand(rng, Normal(mu, sigma))
Z = rand(rng, Uniform(a, b), (10, 10))
```

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points
- Example: N grid points $x_1, \ldots, x_N$

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points
- Example: N grid points $x_1, \ldots, x_N$
- Draw independent random variables $a_1, \ldots, a_N$

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points
- Example: N grid points $x_1, \ldots, x_N$
- Draw independent random variables $a_1, \ldots, a_N$
- Set $a(x_i, \omega) = a_i(\omega)$

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points
- Example: N grid points $x_1, \ldots, x_N$
- Draw independent random variables $a_1, \ldots, a_N$
- Set $a(x_i, \omega) = a_i(\omega)$
- **Problems:**

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points
- Example: N grid points $x_1, \ldots, x_N$
- Draw independent random variables $a_1, \ldots, a_N$
- Set $a(x_i, \omega) = a_i(\omega)$
- **Problems:**
  - Not well-defined in continuous limit ($N \to \infty$)

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points
- Example: N grid points $x_1, \ldots, x_N$
- Draw independent random variables $a_1, \ldots, a_N$
- Set $a(x_i, \omega) = a_i(\omega)$
- **Problems:**
  - Not well-defined in continuous limit ($N \to \infty$)
  - No spatial correlation structure

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points
- Example: N grid points $x_1, \ldots, x_N$
- Draw independent random variables $a_1, \ldots, a_N$
- Set $a(x_i, \omega) = a_i(\omega)$
- **Problems:**
  - Not well-defined in continuous limit ($N \to \infty$)
  - No spatial correlation structure
  - Extremely rough/discontinuous realizations

SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points
- Example: N grid points $x_1, \ldots, x_N$
- Draw independent random variables $a_1, \ldots, a_N$
- Set $a(x_i, \omega) = a_i(\omega)$
- **Problems:**
  - Not well-defined in continuous limit ($N \to \infty$)
  - No spatial correlation structure
  - Extremely rough/discontinuous realizations
  - Number of parameters grows with mesh refinement

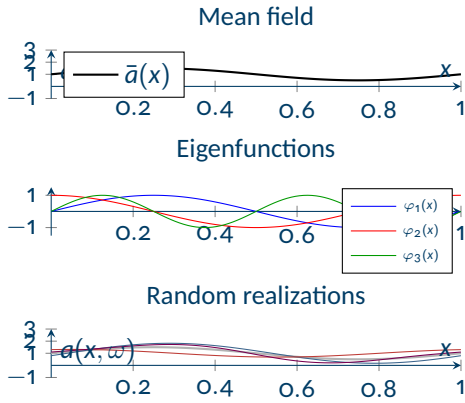
SINTEF

Random functions: naive approach

- Suppose we want a random coefficient field $a(x, \omega)$ for $x \in [0, 1]$
- **Naive idea: (don't do this!)** Generate random values at grid points
- Example: N grid points $x_1, \ldots, x_N$
- Draw independent random variables $a_1, \ldots, a_N$
- Set $a(x_i, \omega) = a_i(\omega)$
- **Problems:**
  - Not well-defined in continuous limit ($N \to \infty$)
  - No spatial correlation structure
  - Extremely rough/discontinuous realizations
  - Number of parameters grows with mesh refinement
  - Cannot compare solutions on different meshes

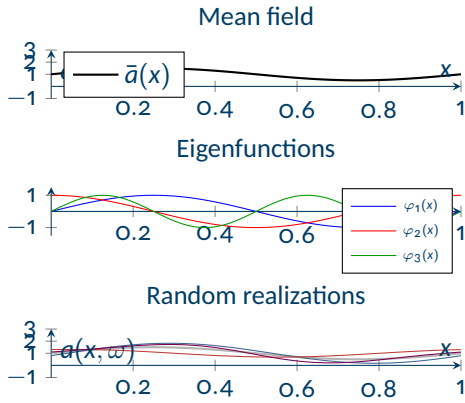
Uncertain functions: Karhunen-Loève expansion

- Sometimes uncertainty is in a *function*, not just parameters



Uncertain functions: Karhunen-Loève expansion

- Sometimes uncertainty is in a *function*, not just parameters
- Example: Random field $a(x, \omega)$ for $x \in D$, $\omega \in \Omega$

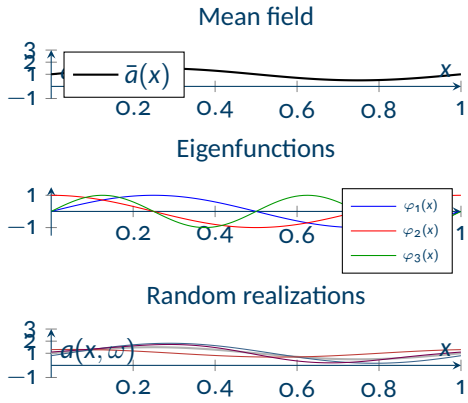


Uncertain functions: Karhunen-Loève expansion

- Sometimes uncertainty is in a *function*, not just parameters
- Example: Random field $a(x, \omega)$ for $x \in D$, $\omega \in \Omega$
- **Karhunen-Loève (KL) expansion:**

$$a(x, \omega) = \bar{a}(x) + \sum_{i=1}^{\infty} \sqrt{\lambda_i} \varphi_i(x) Y_i(\omega)$$

where



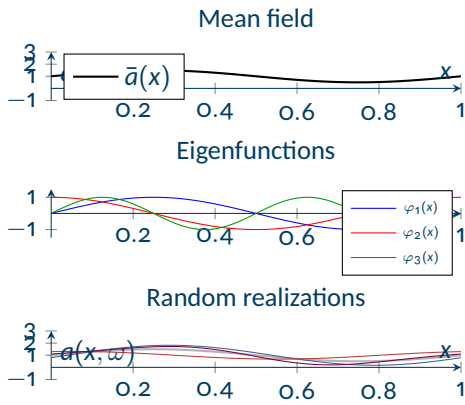
Uncertain functions: Karhunen-Loève expansion

- Sometimes uncertainty is in a *function*, not just parameters
- Example: Random field $a(x, \omega)$ for $x \in D$, $\omega \in \Omega$
- **Karhunen-Loève (KL) expansion:**

$$a(x, \omega) = \bar{a}(x) + \sum_{i=1}^{\infty} \sqrt{\lambda_i} \varphi_i(x) Y_i(\omega)$$

where

- $\bar{a}(x) = \mathbb{E}(a(x, \cdot))$ is the mean



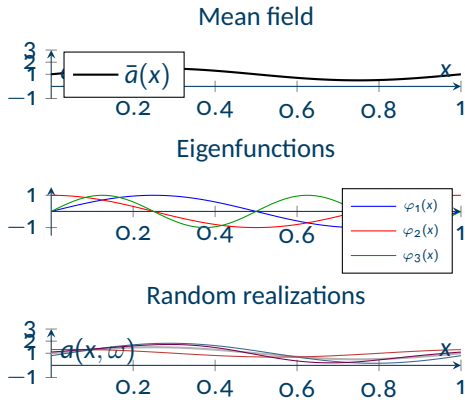
Uncertain functions: Karhunen-Loève expansion

- Sometimes uncertainty is in a *function*, not just parameters
- Example: Random field $a(x, \omega)$ for $x \in D$, $\omega \in \Omega$
- **Karhunen-Loève (KL) expansion:**

$$a(x, \omega) = \bar{a}(x) + \sum_{i=1}^{\infty} \sqrt{\lambda_i} \varphi_i(x) Y_i(\omega)$$

where

- $\bar{a}(x) = \mathbb{E}(a(x, \cdot))$ is the mean
- λ_i, φ_i are eigenvalues/eigenfunctions of covariance



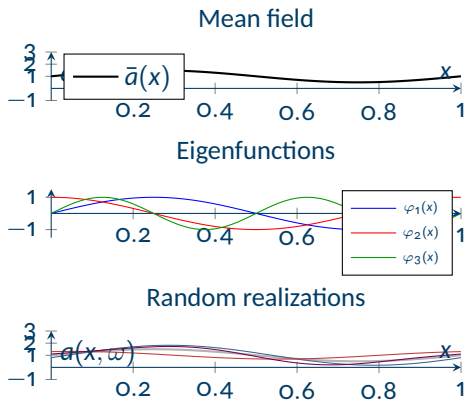
Uncertain functions: Karhunen-Loève expansion

- Sometimes uncertainty is in a *function*, not just parameters
- Example: Random field $a(x, \omega)$ for $x \in D$, $\omega \in \Omega$
- **Karhunen-Loève (KL) expansion:**

$$a(x, \omega) = \bar{a}(x) + \sum_{i=1}^{\infty} \sqrt{\lambda_i} \varphi_i(x) Y_i(\omega)$$

where

- $\bar{a}(x) = \mathbb{E}(a(x, \cdot))$ is the mean
- λ_i, φ_i are eigenvalues/eigenfunctions of covariance
- $Y_i(\omega)$ are uncorrelated random variables



Uncertain functions: Karhunen-Loève expansion

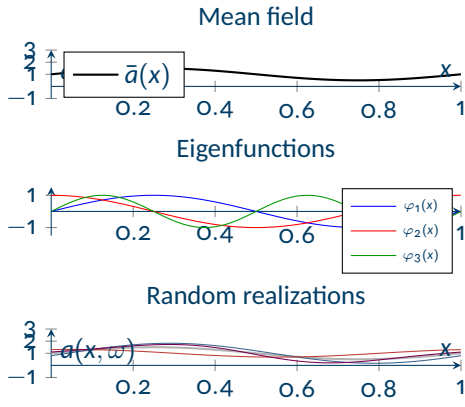
- Sometimes uncertainty is in a *function*, not just parameters
- Example: Random field $a(x, \omega)$ for $x \in D$, $\omega \in \Omega$
- **Karhunen-Loève (KL) expansion:**

$$a(x, \omega) = \bar{a}(x) + \sum_{i=1}^{\infty} \sqrt{\lambda_i} \varphi_i(x) Y_i(\omega)$$

where

  - $\bar{a}(x) = \mathbb{E}(a(x, \cdot))$ is the mean
  - λ_i, φ_i are eigenvalues/eigenfunctions of covariance
  - $Y_i(\omega)$ are uncorrelated random variables

- Truncate to finite sum:
  $a(x, \omega) \approx \bar{a}(x) + \sum_{i=1}^{d} \sqrt{\lambda_i} \varphi_i(x) Y_i(\omega)$



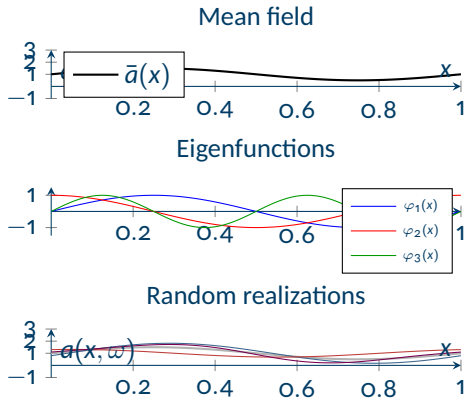
Uncertain functions: Karhunen-Loève expansion

- Sometimes uncertainty is in a *function*, not just parameters
- Example: Random field $a(x, \omega)$ for $x \in D$, $\omega \in \Omega$
- **Karhunen-Loève (KL) expansion:**

$$a(x, \omega) = \bar{a}(x) + \sum_{i=1}^{\infty} \sqrt{\lambda_i} \varphi_i(x) Y_i(\omega)$$

where
  - $\bar{a}(x) = \mathbb{E}(a(x, \cdot))$ is the mean
  - λ_i, φ_i are eigenvalues/eigenfunctions of covariance
  - $Y_i(\omega)$ are uncorrelated random variables

- Truncate to finite sum: $a(x, \omega) \approx \bar{a}(x) + \sum_{i=1}^{d} \sqrt{\lambda_i} \varphi_i(x) Y_i(\omega)$



SINTEF

Programming tips: Make sure you use a suitable random number generator

- When using KL expansion with d terms, need d random numbers

SINTEF

Programming tips: Make sure you use a suitable random number generator

- When using KL expansion with d terms, need d random numbers
- If using Monte Carlo with M samples, need $M \times d$ random numbers

SINTEF

Programming tips: Make sure you use a suitable random number generator

- When using KL expansion with d terms, need d random numbers
- If using Monte Carlo with M samples, need $M \times d$ random numbers
- If using multilevel methods with L levels, need $\sum_{\ell=0}^{L} M_\ell \times d$ random numbers

SINTEF

Programming tips: Make sure you use a suitable random number generator

- When using KL expansion with d terms, need d random numbers
- If using Monte Carlo with M samples, need $M \times d$ random numbers
- If using multilevel methods with L levels, need $\sum_{\ell=0}^{L} M_\ell \times d$ random numbers
- Make sure your random number generator can provide enough random numbers

SINTEF

Programming tips: Make sure you use a suitable random number generator

- When using KL expansion with d terms, need d random numbers
- If using Monte Carlo with M samples, need $M \times d$ random numbers
- If using multilevel methods with L levels, need $\sum_{\ell=0}^{L} M_\ell \times d$ random numbers
- Make sure your random number generator can provide enough random numbers
- Some generators have limits on how many random numbers can be drawn

SINTEF

Programming tips: Make sure you use a suitable random number generator

- When using KL expansion with d terms, need d random numbers
- If using Monte Carlo with M samples, need $M \times d$ random numbers
- If using multilevel methods with L levels, need $\sum_{\ell=0}^{L} M_\ell \times d$ random numbers
- Make sure your random number generator can provide enough random numbers
- Some generators have limits on how many random numbers can be drawn
- However: Mersenne Twister has period of $2^{19937} - 1 \approx 4.3 \times 10^{6000}$

SINTEF

Conclusion

- Monte Carlo

$$\mathbb{E}(Q(\mathbf{Y})) = \int_{[0,1]^d} Q(\mathbf{y})\, d\mathbf{y} \approx \frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k)$$

Conclusion

- Monte Carlo

$$\mathbb{E}(Q(\mathbf{Y})) = \int_{[0,1]^d} Q(\mathbf{y})\, d\mathbf{y} \approx \frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k)$$

- Error of Monte Carlo

$$\mathbb{E}\left(\left(\frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k) - \mathbb{E}(Q(\mathbf{Y}))\right)^2\right)^{1/2} = \frac{\mathrm{Var}(Q(\mathbf{Y}))^{1/2}}{M^{1/2}}$$

Conclusion

- Monte Carlo

$$\mathbb{E}(Q(\mathbf{Y})) = \int_{[0,1]^d} Q(\mathbf{y})\, d\mathbf{y} \approx \frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k)$$

- Error of Monte Carlo

$$\mathbb{E}\left(\left(\frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k) - \mathbb{E}(Q(\mathbf{Y})) \right)^2 \right)^{1/2} = \frac{\mathrm{Var}(Q(\mathbf{Y}))^{1/2}}{M^{1/2}}$$

- Motivation for the coming part: Monte Carlo is *slow*

Conclusion

- Monte Carlo

$$\mathbb{E}(Q(\mathbf{Y})) = \int_{[0,1]^d} Q(\mathbf{y})\, d\mathbf{y} \approx \frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k)$$

- Error of Monte Carlo

$$\mathbb{E}\left(\left(\frac{1}{M} \sum_{k=1}^{M} Q(\mathbf{Y}_k) - \mathbb{E}(Q(\mathbf{Y}))\right)^2\right)^{1/2} = \frac{\mathrm{Var}(Q(\mathbf{Y}))^{1/2}}{M^{1/2}}$$

- Motivation for the coming part: Monte Carlo is *slow*
- …but sometimes the best we have

Recap

Convergence of Monte Carlo

$$\left(\mathbb{E}\left[\mathbb{E}(X) - \frac{1}{M} \sum_{k=1}^{M} X_k \right]^2 \right)^{1/2} = \frac{\mathrm{Var}(X)^{1/2}}{M^{1/2}}$$

New notation:

$$\|f\|_{L^2(\Omega)} := \left(\mathbb{E}(f^2)\right)^{1/2}, \quad f \in L^2(\Omega)$$

Thus,

$$\left\| \mathbb{E}(X) - \frac{1}{M} \sum_{k=1}^{M} X_k \right\|_{L^2(\Omega)} = \frac{\mathrm{Var}(X)^{1/2}}{M^{1/2}}$$

Model Problem

$$\begin{cases} \dfrac{d}{dt}u(\omega;t) = F(u(\omega;t)), \\ u(\omega;0) = u_0(\omega), \end{cases}$$

Let $u_0^1, \ldots, u_0^M$ be i.i.d. samples of u_0.
Let

$$u^k_{\Delta t,T}(\omega) = F_{\Delta t,T}(\omega; u_0^k),$$

where $F_{\Delta t,T}(\omega;\cdot)$ is a numerical scheme.

Approximate:

$$\mathbb{E}(u(\cdot;T)) \approx \frac{1}{M}\sum_{k=1}^{M} u^k_{\Delta t,T}(\omega)$$

Error Estimation

Triangle inequality:

$$\left\| \mathbb{E}(u(\cdot;T)) - \frac{1}{M}\sum_{k=1}^{M} u^k_{\Delta t,T} \right\|_{L^2(\Omega)} \leq \varepsilon_1 + \varepsilon_2$$

where

$$\varepsilon_1 = \left\| \mathbb{E}(u(\cdot;T)) - \mathbb{E}(u^k_{\Delta t,T}) \right\|_{L^2(\Omega)}$$

$$\varepsilon_2 = \left\| \mathbb{E}(u^k_{\Delta t,T}) - \frac{1}{M}\sum_{k=1}^{M} u^k_{\Delta t,T} \right\|_{L^2(\Omega)}$$

Bounding the Errors

Assume

$$|u(\omega; T) - u^k_{\Delta t, T}(\omega)| \leq C\Delta t^s \quad \forall \omega \in \Omega$$

Then

$$\varepsilon_1 \leq C\Delta t^s$$

And by Monte Carlo convergence,

$$\varepsilon_2 \leq \frac{\mathrm{Var}(u_{\Delta t, T})^{1/2}}{M^{1/2}}$$

Hence,

$$\left\| \mathbb{E}(u(\cdot; T)) - \frac{1}{M} \sum_{k=1}^{M} u^k_{\Delta t, T} \right\|_{L^2(\Omega)} \leq C\Delta t^s + \frac{\mathrm{Var}(u_{\Delta t, T})^{1/2}}{M^{1/2}}$$

Equilibrating the Error

Choose

$$C\Delta t^s \approx \frac{\mathrm{Var}(u_{\Delta t,T})^{1/2}}{M^{1/2}}$$

This gives

$$M = \mathcal{O}(\Delta t^{-2s})$$

Total cost:

$$\text{Cost} = M \cdot \text{Cost per sample} = \mathcal{O}(\Delta t^{-2s}) \cdot \mathcal{O}(\Delta t^{-1}) = \mathcal{O}(\Delta t^{-(2s+1)})$$

So if we want to go from an error tolerance of ϵ to $\epsilon/2$, we need to increase the computational cost by a factor of

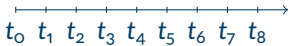
$$\left(\frac{\epsilon}{\epsilon/2}\right)^{2+\frac{1}{s}} = 2^{2+\frac{1}{s}}$$

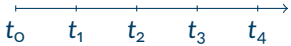
Multilevel Monte Carlo

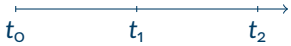
Multilevel telescoping sum

Numerical approximation u^{Δ}

Multilevel telescoping sum

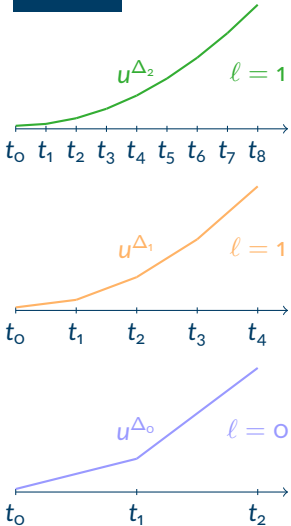






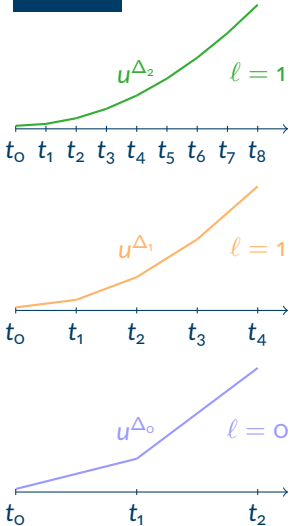
Numerical approximation u^{Δ}

Multilevel telescoping sum



Numerical approximation u^{Δ}

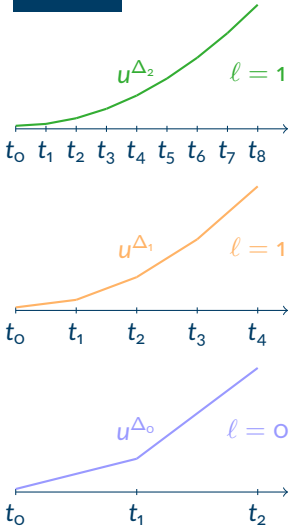
Multilevel telescoping sum



Numerical approximation u^{Δ}

$$u^{\Delta_1} = (u^{\Delta_1} - u^{\Delta_0}) + u^{\Delta_0}$$

Multilevel telescoping sum



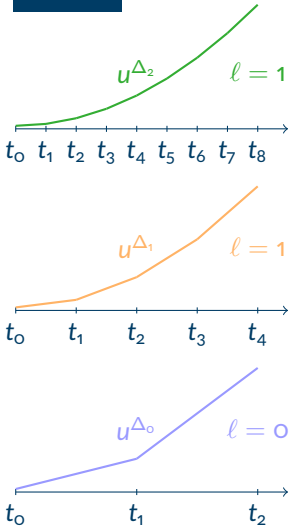
Numerical approximation u^{Δ}

$$u^{\Delta_1} = (u^{\Delta_1} - u^{\Delta_0}) + u^{\Delta_0}$$

Similarly,

$$u^{\Delta_2} = (u^{\Delta_2} - u^{\Delta_1}) + (u^{\Delta_1} - u^{\Delta_0}) + u^{\Delta_0}$$

Multilevel telescoping sum



Numerical approximation u^{Δ}

$$u^{\Delta_1} = (u^{\Delta_1} - u^{\Delta_0}) + u^{\Delta_0}$$

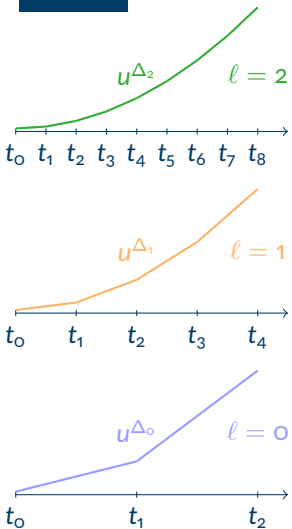
Similarly,

$$u^{\Delta_2} = (u^{\Delta_2} - u^{\Delta_1}) + (u^{\Delta_1} - u^{\Delta_0}) + u^{\Delta_0}$$

In general,

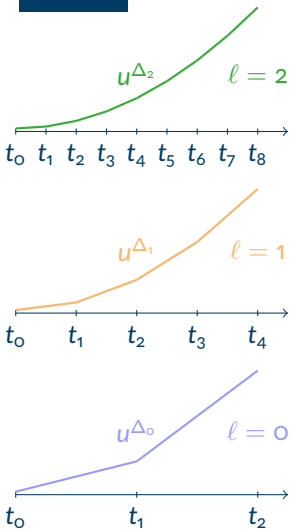
$$u^{\Delta_L} = \sum_{l=1}^{L} (u^{\Delta_\ell} - u^{\Delta_{\ell-1}}) + u^{\Delta_0}$$

Multilevel Monte Carlo [Giles, 2008; Heinrich 2001]



Random variable $u^{\Delta}(\omega)$

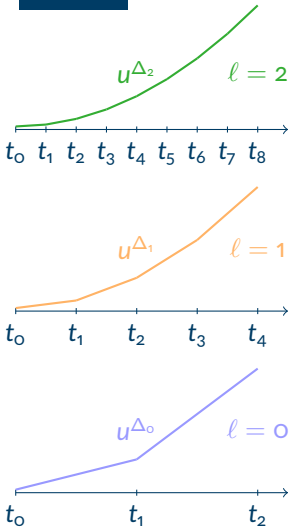
Multilevel Monte Carlo [Giles, 2008; Heinrich 2001]



Random variable $u^{\Delta}(\omega)$

$$\mathbb{E}(u^{\Delta_1}) = \mathbb{E}(u^{\Delta_1} - u^{\Delta_0}) + \mathbb{E}(u^{\Delta_0})$$

Multilevel Monte Carlo [Giles, 2008; Heinrich 2001]



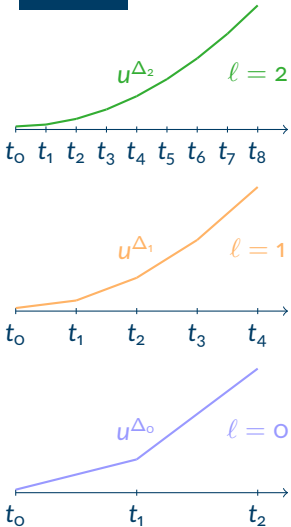
Random variable $u^{\Delta}(\omega)$

$$\mathbb{E}(u^{\Delta_1}) = \mathbb{E}(u^{\Delta_1} - u^{\Delta_0}) + \mathbb{E}(u^{\Delta_0})$$

Similarly,

$$\mathbb{E}(u^{\Delta_2}) = \mathbb{E}(u^{\Delta_2} - u^{\Delta_1}) + \mathbb{E}(u^{\Delta_1} - u^{\Delta_0}) + \mathbb{E}(u^{\Delta_0})$$

Multilevel Monte Carlo [Giles, 2008; Heinrich 2001]



Random variable $u^{\Delta}(\omega)$

$$\mathbb{E}(u^{\Delta_1}) = \mathbb{E}(u^{\Delta_1} - u^{\Delta_0}) + \mathbb{E}(u^{\Delta_0})$$

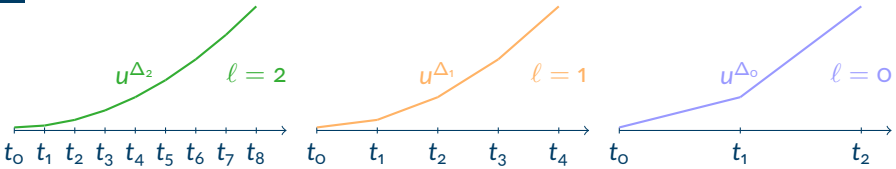
Similarly,

$$\mathbb{E}(u^{\Delta_2}) = \mathbb{E}(u^{\Delta_2} - u^{\Delta_1}) + \mathbb{E}(u^{\Delta_1} - u^{\Delta_0}) + \mathbb{E}(u^{\Delta_0})$$

In general,

$$\mathbb{E}(u^{\Delta_L}) = \sum_{l=1}^{L} \mathbb{E}(u^{\Delta_\ell} - u^{\Delta_{\ell-1}}) + \mathbb{E}(u^{\Delta_0})$$

Multilevel Monte Carlo Estimator



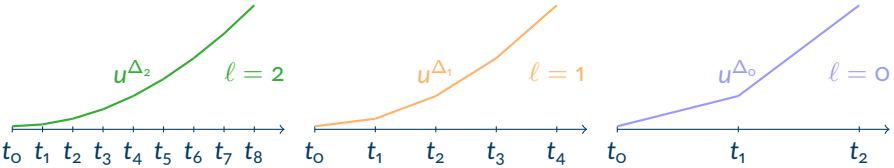
$$E_L(u) = \sum_{\ell=1}^{L} \overbrace{\frac{1}{M_\ell} \sum_{k=1}^{M_\ell} \left(u^{k,\Delta_\ell,+} - u^{k,\Delta_{\ell-1},-}\right)}^{\text{Monte Carlo estimate of } \mathbb{E}(u^{\Delta_\ell} - u^{\Delta_{\ell-1}})} + \underbrace{\frac{1}{M_0} \sum_{k=1}^{M_0} u^{k,\Delta_0}}_{\text{Monte Carlo estimate of } \mathbb{E}(u^{\Delta_0})}$$

Key idea: Use different sample sizes M_ℓ on different levels

Estimator properties:

- Unbiased: $\mathbb{E}(E_L(u)) = \mathbb{E}(u^{\Delta_L})$
- Key idea: $\mathrm{Var}(u^{\Delta_\ell} - u^{\Delta_{\ell-1}})$ decreases with ℓ
- $\Rightarrow$ can use smaller M_ℓ on finer levels

Multilevel Monte Carlo: Drawing samples



Monte Carlo estimate of $\mathbb{E}(u^{\Delta_\ell} - u^{\Delta_{\ell-1}})$

$$E_L(u) = \sum_{\ell=1}^{L} \frac{1}{M_\ell} \sum_{k=1}^{M_\ell} \left(u^{k,\Delta_\ell,+} - u^{k,\Delta_{\ell-1},-}\right) + \frac{1}{M_0} \sum_{k=1}^{M_0} u^{k,\Delta_0}$$

Monte Carlo estimate of $\mathbb{E}(u^{\Delta_0})$

$u^{k,\Delta_\ell,+}$ and $u^{k,\Delta_{\ell-1},-}$ must be computed using the **same** random input.

```
u_delta_l_plus = solve_model(random_input_k, delta_l);
u_delta_lminus = solve_model(random_input_k, delta_lminus);
```

MLMC:General observable

$$E_L(g(u)) = \sum_{\ell=1}^{L} \overbrace{\frac{1}{M_\ell} \sum_{k=1}^{M_\ell} \left(g(u^{k,\Delta_\ell,+}) - g(u^{k,\Delta_{\ell-1},-}) \right)}^{\text{Monte Carlo estimate of } \mathbb{E}(g(u^{\Delta_\ell}) - g(u^{\Delta_{\ell-1}}))} + \underbrace{\frac{1}{M_o} \sum_{k=1}^{M_o} g(u^{k,\Delta_o})}_{\text{Monte Carlo estimate of } \mathbb{E}(g(u^{\Delta_o}))}$$

Picking the numbers of samples

Error Decomposition

Error satisfies

$$\|\mathbb{E}(u) - E_L(u)\|_{L^2(\Omega)} \leq \underbrace{\|\mathbb{E}(u) - \mathbb{E}(u_{\Delta_L})\|_{L^2(\Omega)}}_{\text{Discretization error}} + \underbrace{\|E_L(u) - \mathbb{E}(u_{\Delta_L})\|_{L^2(\Omega)}}_{\text{Stochastic error}}$$

Stochastic error satisfies

$$\|E_L(u) - \mathbb{E}(u_{\Delta_L})\|_{L^2(\Omega)} \leq \sum_{\ell=1}^{L} \frac{\mathrm{Var}(u_{\Delta_\ell} - u_{\Delta_{\ell-1}})^{1/2}}{M_\ell^{1/2}} + \frac{\mathrm{Var}(u_{\Delta_0})^{1/2}}{M_0^{1/2}}$$

Picking the numbers of samples

Assume we have a target tolerance $\tau > 0$ for the stochastic error:

$$\|\mathbb{E}(u_L) - E_L(u)\|_{L^2(\Omega)} \leq \tau$$

We let V_l denote the variance at level ℓ:

$$V_l = \mathrm{Var}(u_{\Delta_\ell} - u_{\Delta_{\ell-1}}), \quad V_0 = \mathrm{Var}(u_{\Delta_0})$$

and the C_l denote the cost per sample at level ℓ:

$$C_l = \text{Cost per sample at level } \ell$$

Picking the numbers of samples

Assume we have a target tolerance $\tau > 0$ for the stochastic error:

$$\|\mathbb{E}(u_L) - E_L(u)\|_{L^2(\Omega)} \leq \tau$$

We let V_l denote the variance at level ℓ:

$$V_l = \mathrm{Var}(u_{\Delta_\ell} - u_{\Delta_{\ell-1}}), \quad V_0 = \mathrm{Var}(u_{\Delta_0})$$

and the C_l denote the cost per sample at level ℓ:

$$C_l = \text{Cost per sample at level } \ell$$

Then we want to solve the **optimization problem**

$$\begin{cases} \min\limits_{M_0,\ldots,M_L} \sum\limits_{\ell=0}^{L} M_\ell \cdot C_\ell \\ \text{subject to} \quad \sum\limits_{\ell=0}^{L} \dfrac{V_\ell}{M_\ell} \leq \tau^2 \end{cases}$$

Solution via Lagrange multipliers

$$\begin{cases} \min\limits_{M_0,\ldots,M_L} \sum\limits_{\ell=0}^{L} M_\ell \cdot C_\ell \\ \text{subject to} \quad \sum\limits_{\ell=0}^{L} \dfrac{V_\ell}{M_\ell} \leq \tau^2 \end{cases}$$

One can show that the solution to the optimization problem is

$$M_\ell = \left\lceil \frac{1}{\tau^2} \sqrt{\frac{V_\ell}{C_\ell}} \sum_{k=0}^{L} \sqrt{V_k C_k} \right\rceil, \quad \ell = 0, \ldots, L \tag{3.1}$$

SINTEF

Convergence rates and MLMC complexity [Giles, 2015]

Theorem

Assume

$$V_\ell \leq c_2 2^{-\beta \ell},$$

$$C_\ell \leq c_3 2^{\gamma \ell},$$

the computational cost of MLMC to achieve a root mean square error (RMSE) of τ is bounded by

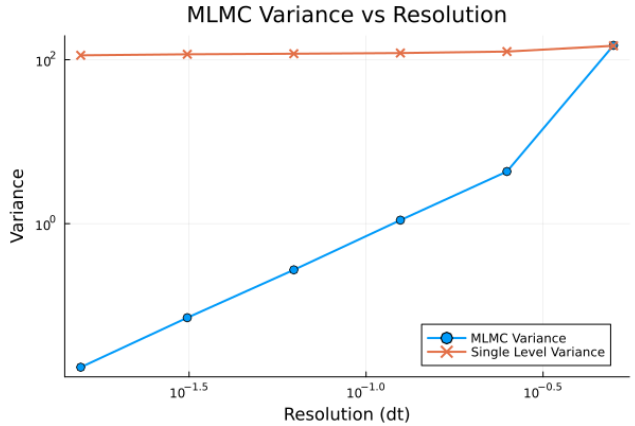
$$\begin{cases} c_4 \tau^{-2}, & \beta > \gamma, \\ c_4 \tau^{-2} (\log \tau)^2, & \beta = \gamma, \\ c_4 \tau^{-2-(\gamma-\beta)/\alpha}, & \beta < \gamma. \end{cases}$$

MLMC for the projectile motion

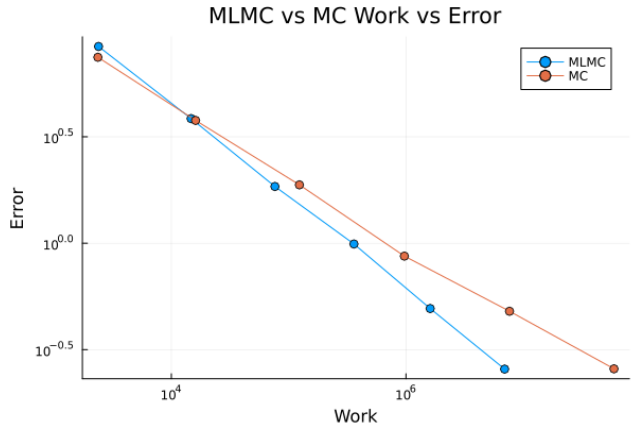
$$C_\ell = C\Delta t_\ell^{-1} = \mathcal{O}(\text{Number of timesteps}) \qquad V_\ell = \text{Var}(u_{\Delta_\ell} - u_{\Delta_{\ell-1}}) = \mathcal{O}(\Delta t_\ell^2)$$

MLMC for the projectile motion

$C_\ell = C\Delta t_\ell^{-1} = \mathcal{O}(\text{Number of timesteps})$ $\quad V_\ell = \text{Var}(u_{\Delta_\ell} - u_{\Delta_{\ell-1}}) = \mathcal{O}(\Delta t_\ell^2)$



MLMC for the projectile motion: Work comparison



Pesudo-sketch for MLMC

```
function simulate_mlmc_samples(resolutions, required_samples)
    samples = []
    for (level, M) in enumerate(required_samples)
        push!(samples, [])
        for _ in 1:M
            parameters = rand(base_parameters)
            fine_solution = model(parameters; dt = timestepsizes[level])
            if level == 1
                push!(samples[level], fine_solution)
            else
                coarse_solution = model(parameters;
                    dt = timestepsizes[level-1])
                push!(samples[level], (fine_solution, coarse_solution))
            end
        end
    end
    return samples
end
```

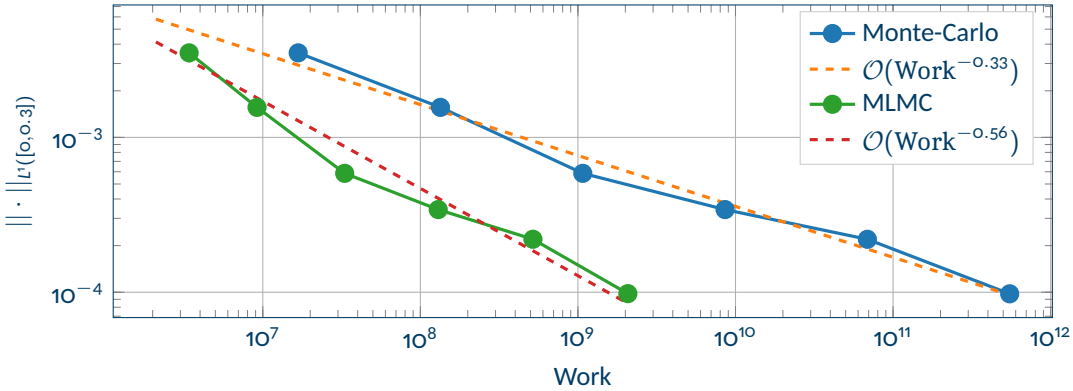
SINTEF

Pesudo-sketch for MLMC: Computing mean of a quantity of interest

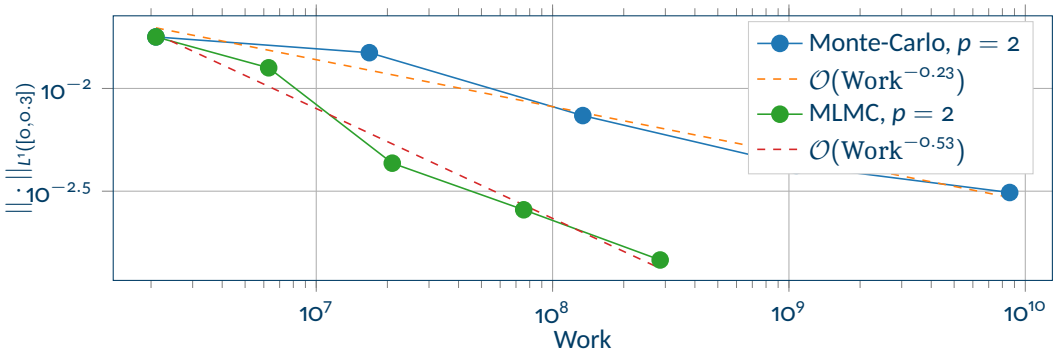
```
function mlmc_estimate(mlmc_samples, g = x -> x)
    mean_estimate = zero(typeof(g(mlmc_samples[1][1])))

    for level in 1:length(mlmc_samples)
        if level == 1
            mean_estimate += mean(g.(mlmc_samples[level]))
        else
            # last element is coarse, first is fine (only two elements per sample)
            mean_estimate += mean(g.(first.(mlmc_samples[level])) \
                                  .- g.(last.(mlmc_samples[level])))
        end
    end
    return mean_estimate
end
```

Academic examples

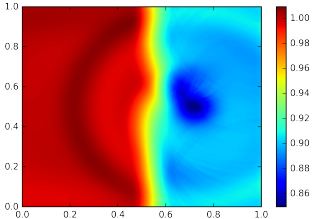
Burgers': Uncertain shock location: MLMC



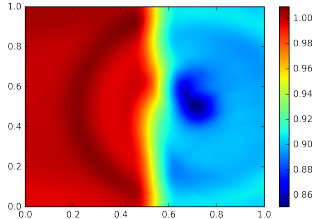
Burgers': Brownian initial data: MLMC



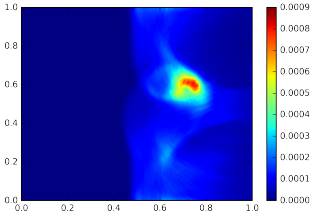
MLMC test: Shock-Vortex



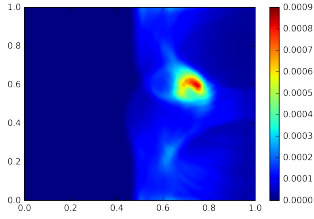
Mean, MLMC



Mean, MC



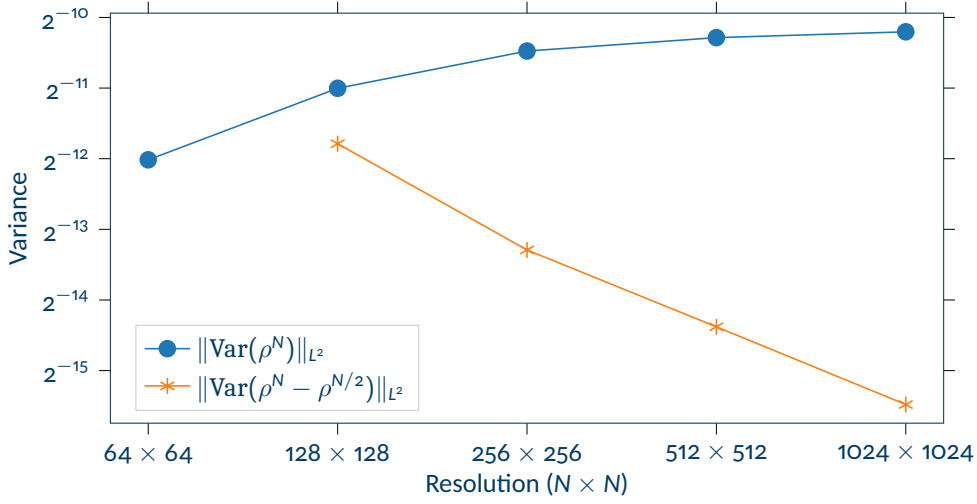
Variance, MLMC



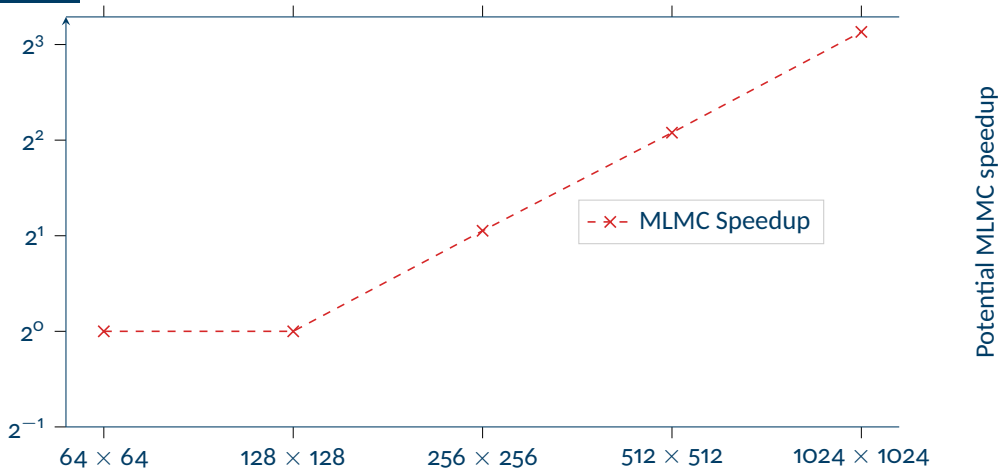
Variance, MC

SINTEF

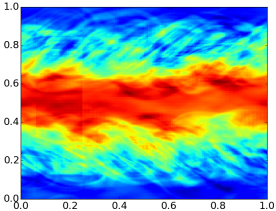
Shock-Vortex: Variance decay



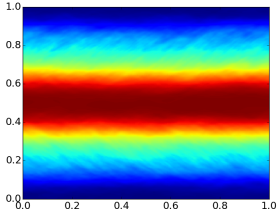
Shock-Vortex: Speedup



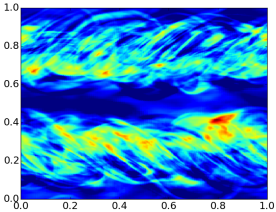
Kelvin–Helmholtz + MLMC

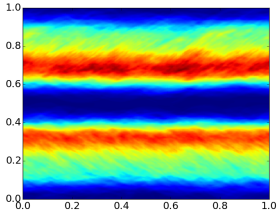


Mean, MLMC



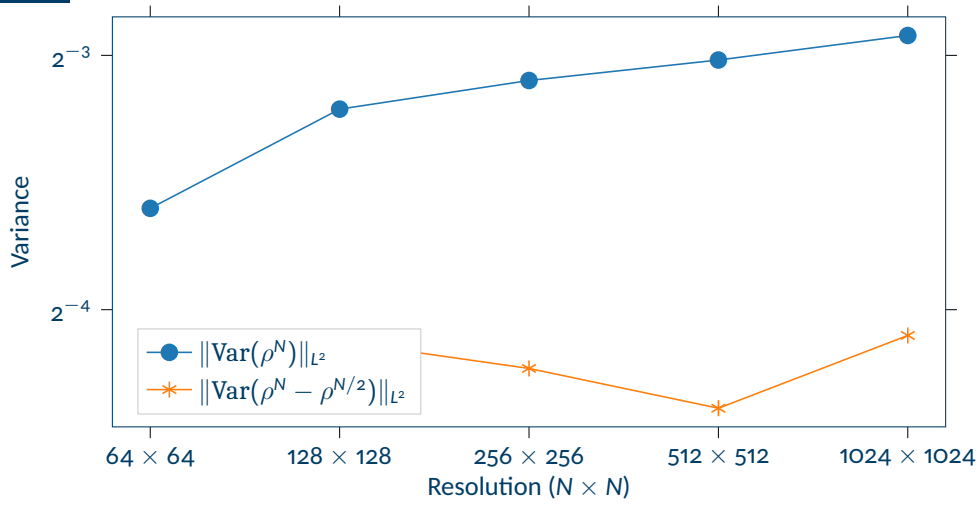
Mean, MC



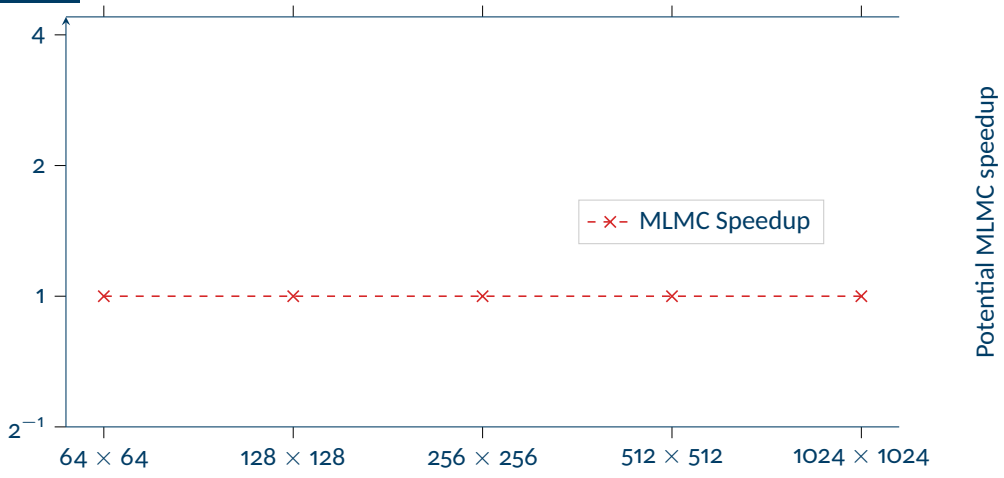


SINTEF

Kelvin–Helmholtz: Variance decay



Kelvin–Helmholtz: Speedup



Industrial examples

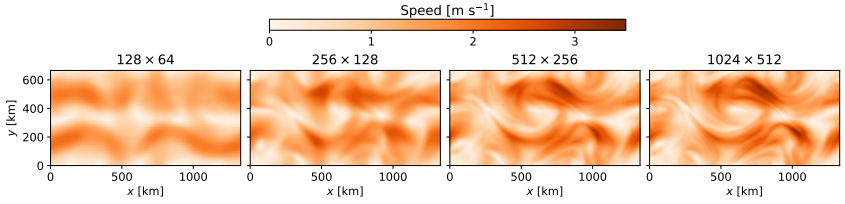
Multilevel EnKF for Shallow Water Equations (Beiser, Holm, Lye, Eidsvik, 2024)

Assimilate (synthetic) momentum observations in a 2D shallow water model:

$$\frac{\partial \eta}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = 0$$

$$\frac{\partial (hu)}{\partial t} + \frac{\partial}{\partial x}\left(\frac{(hu)^2}{H+\eta} + \frac{1}{2}g(H+\eta)^2\right) + \frac{\partial}{\partial y}\left(\frac{(hu)(hv)}{H+\eta}\right) = -g(H+\eta)\frac{\partial B}{\partial x}$$

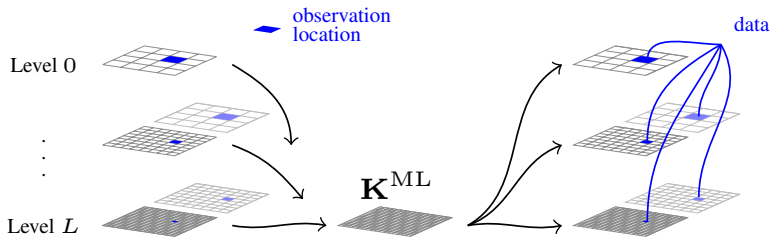
$$\frac{\partial (hv)}{\partial t} + \frac{\partial}{\partial x}\left(\frac{(hu)(hv)}{H+\eta}\right) + \frac{\partial}{\partial y}\left(\frac{(hv)^2}{H+\eta} + \frac{1}{2}g(H+\eta)^2\right) = -g(H+\eta)\frac{\partial B}{\partial y}$$



Multilevel EnKF for Shallow Water Equations (Beiser, Holm, Lye, Eidsvik, 2024)

Multi-Level Monte Carlo (MLMC) for Covariance

$$\mathbf{\Sigma}^{\mathrm{ML}}[\mathbf{x}^L, \mathbf{x}^L] = \frac{1}{N^{\mathrm{o}} - 1} \sum_{e=1}^{N^{\mathrm{o}}} \tilde{\mathbf{x}}_e^{\mathrm{o}} (\tilde{\mathbf{x}}_e^{\mathrm{o}})^\top + \sum_{l=1}^{L} \frac{1}{N^l - 1} \sum_{e=1}^{N^l} \left(\tilde{\mathbf{x}}_e^{l+} (\tilde{\mathbf{x}}_e^{l+})^\top - \tilde{\mathbf{x}}_e^{l-} (\tilde{\mathbf{x}}_e^{l-})^\top \right),$$



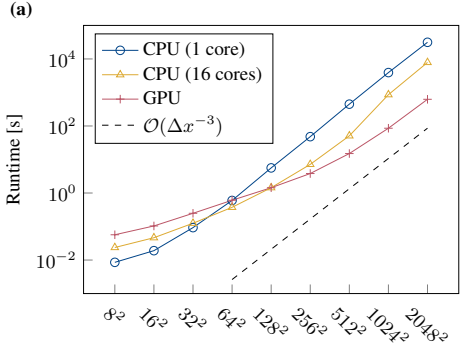
Speedup from MLMC for EnKF (SWE Model)

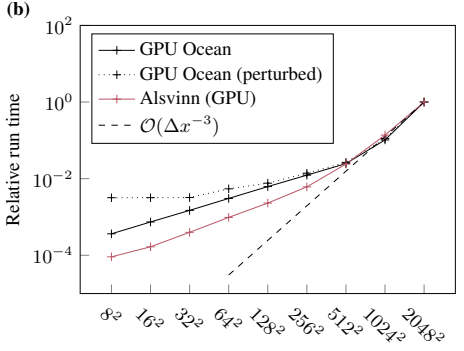
Theoretical speedup: 2.3, **practical speedup**: 1.7.

Speedup from MLMC for EnKF (SWE Model)

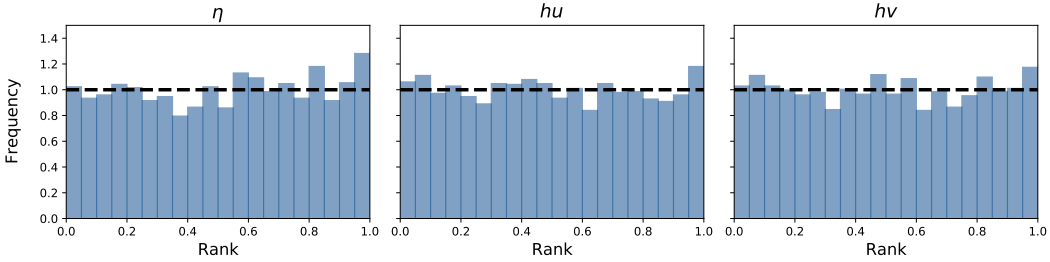
Theoretical speedup: 2.3, **practical speedup**: 1.7.

Numerical code does not always scale perfectly!





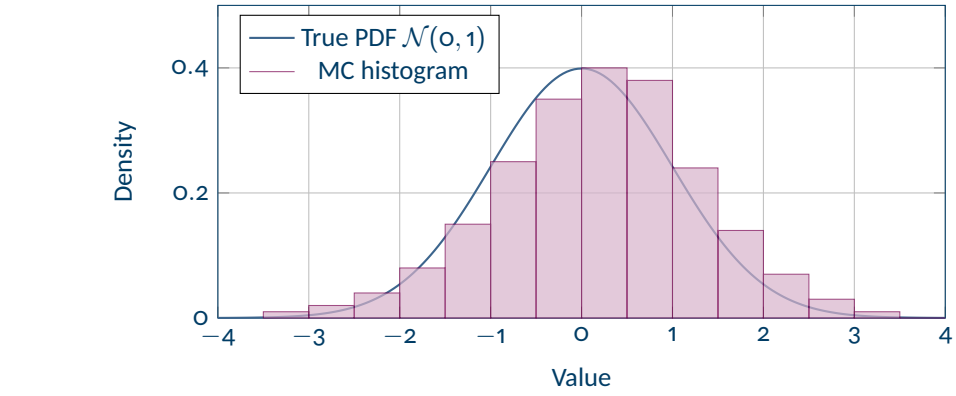
Rank Histograms (SWE Model)



Estimating probability density functions with MLMC

Probability density function and MLMC

With singlelevel Monte Carlo, the histogram approximates the PDF well:



However, with MLMC, the histogram is not easily available.

SINTEF

Orthogonal series density estimation: Estimating the PDF with MLMC

Recall:

$$E_L(g(u)) \approx \mathbb{E}(g(u)) = \int g(u) f_U(u) du$$

where $f_U(u)$ is the PDF of u.

Orthogonal series density estimation: Estimating the PDF with MLMC

Recall:

$$E_L(g(u)) \approx \mathbb{E}(g(u)) = \int g(u) f_U(u) du$$

where $f_U(u)$ is the PDF of u. Let $(\varphi_i)_{i=1}^N$ be a set of orthonormal basis functions (e.g., polynomials, wavelets, etc.). We can write

$$f_U(u) \approx \sum_{i=1}^{N} \alpha_i \varphi_i(u)$$

with coefficients

$$\alpha_i = \int \varphi_i(u) f_U(u) du = \mathbb{E}(\varphi_i(u)) \approx E_L(\varphi_i(u))$$

We can thus estimate the PDF by estimating the coefficients α_i using MLMC.

Estimating the PDF with MLMC using maximum entropy*

Let $(\varphi_i)_{i=0}^{N}$ be a set of orthonormal basis functions. Let

$$\tilde{\alpha}_i = E_L(\varphi_i(u)), \quad i = 0, \ldots, N.$$

The maximum entropy method finds the PDF $\tilde{f}_U(u)$ that maximizes the entropy by finding λ_i such that

$$\begin{cases} \tilde{f}_U(u) := \exp\left(\sum_{k=0}^{N} \tilde{\lambda}_k \varphi_k(u)\right), & \tilde{\lambda}_k \in \mathbb{R}, \\ \tilde{\alpha}_k = \int_I \varphi_k(u)\, \tilde{f}_U(u)\, du, & 0 \leq k \leq N. \end{cases}$$

then

$$\tilde{f}_U(u) = \exp\left(\sum_{k=0}^{N} \tilde{\lambda}_k \varphi_k(u)\right) \approx f_U(u)$$

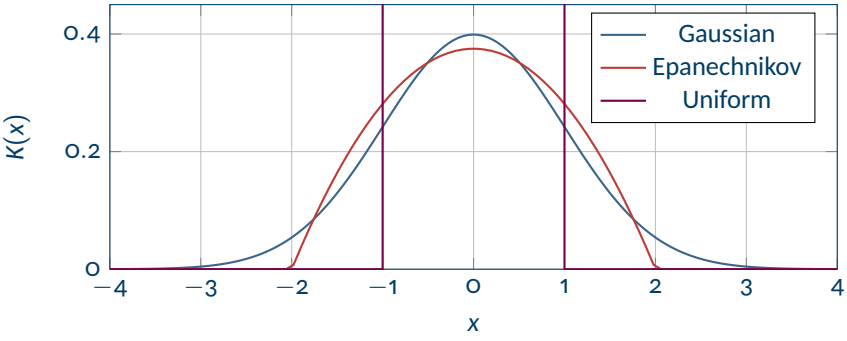
* Bierig and Chernov, 2016

Estimating the PDF with Kernel Density Estimation (KDE)

Idea: Let K_ϵ be a kernel with bandwidth $\epsilon > 0$.

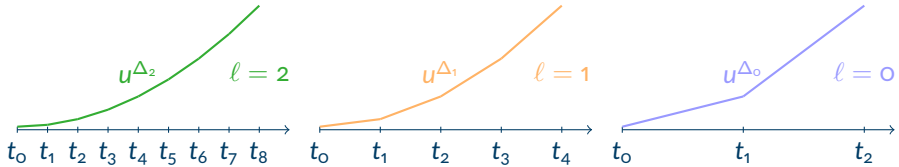
$$f_U(u) \approx \int K_\epsilon(u - v) f_U(v)\, dv = \mathbb{E}\left(K_\epsilon(u - U)\right) \approx E_L\left(K_\epsilon(u - U)\right)$$

Common kernel functions



Conclusions

- Monte Carlo is a simple and robust method for uncertainty quantification
- Monte Carlo converges slowly: error $\sim M^{-1/2}$
- Multilevel Monte Carlo can significantly reduce the computational cost
- Main motivation for Multilevel Monte Carlo: Get the same spatial accuracy at lower cost



Monte Carlo estimate of $\mathbb{E}(u^{\Delta_\ell} - u^{\Delta_{\ell-1}})$

$$E_L(u) = \sum_{\ell=1}^{L} \frac{1}{M_\ell} \sum_{k=1}^{M_\ell} \left(u^{k,\Delta_\ell,+} - u^{k,\Delta_{\ell-1},-} \right) + \underbrace{\frac{1}{M_0} \sum_{k=1}^{M_0} u^{k,\Delta_0}}_{\text{Monte Carlo estimate of } \mathbb{E}(u^{\Delta_0})}$$