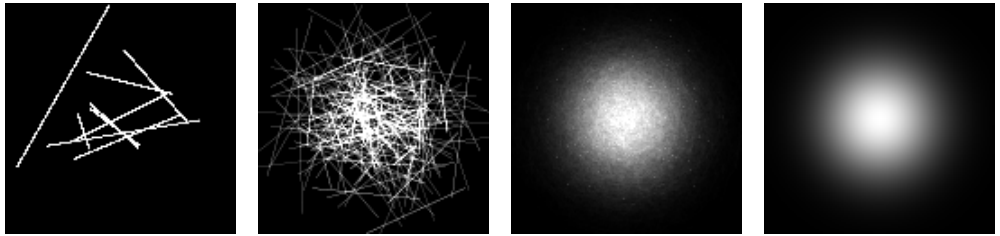
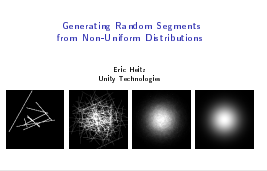
Generating Random Segments from Non-Uniform Distributions

Eric Heitz
Unity Technologies

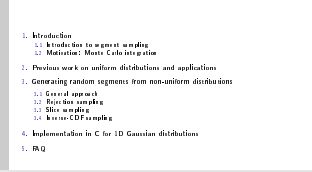




This presentation contains many animated slides that do not display well with all PDF viewers.

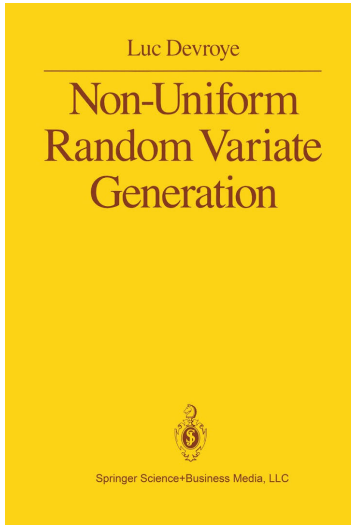
We recommend using **Adobe Acrobat Reader**.

Organization

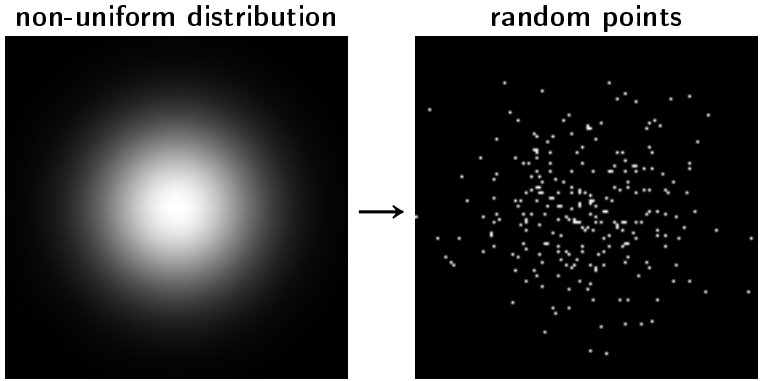


1.1 Introduction to segment sampling

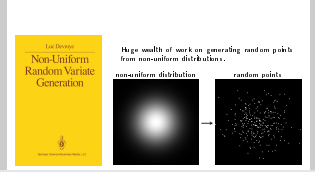
1.1 Introduction to segment sampling



Huge wealth of work on generating random points from non-uniform distributions.



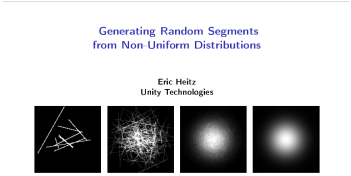
Generating Random Segments from Non-Uniform Distributions



Generating random variates is the fundamental operation of Monte Carlo algorithms. Many approaches have been designed for generating random variate from non-uniform distribution.

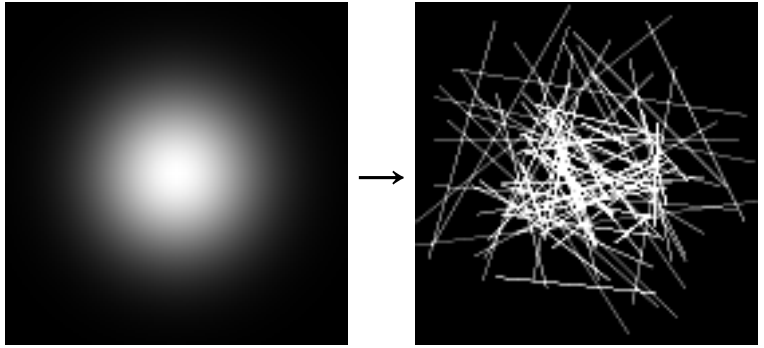
In general, "random variates" actually means "random points".

1.1 Introduction to segment sampling

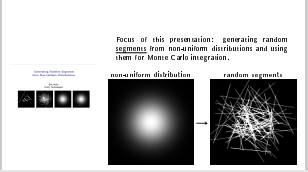


Focus of this presentation: generating random segments from non-uniform distributions and using them for Monte Carlo integration.

non-uniform distribution → random segments



Generating Random Segments from Non-Uniform Distributions



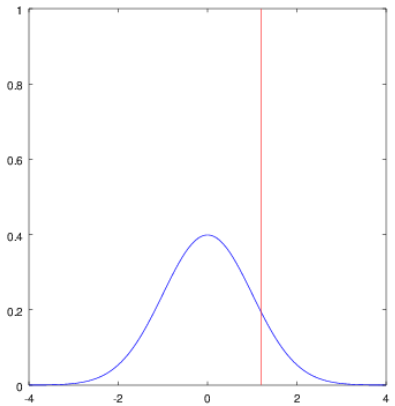
The goal of this presentation is to show how to generate random segments instead of random points from non-uniform distributions.

A segment sample represents not just one but multiple points at the same time. Doing calculations with a segment sample is equivalent to average the calculations that would be done with all the points from this segment.

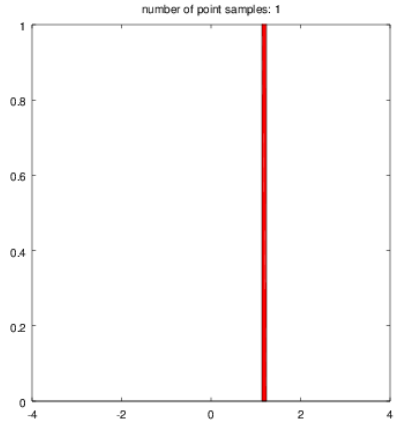
The main motivation for generating random segment is Monte Carlo integration. Random point generation is usually used to compute and average punctual evaluations of a function. With random segment generation, the contributions of all the points contained in a segment can be averaged at the same time. This yields significant variance reduction in the Monte Carlo estimator for many applications.

1.1 Introduction to segment sampling

distribution $f(x)$



histogram of random points



Generating Random Segments from Non-Uniform Distributions

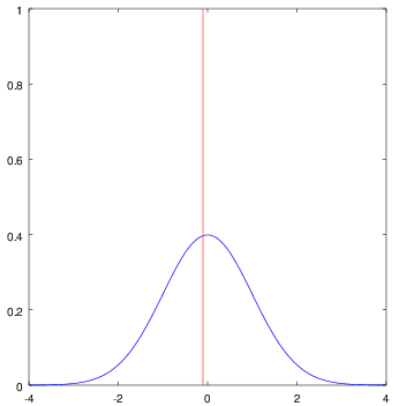
We consider a non-uniform Probability Density Function (PDF) f.

Intuitively, generating random points from f means generating them such that their histogram converges towards f as the number of points increases.

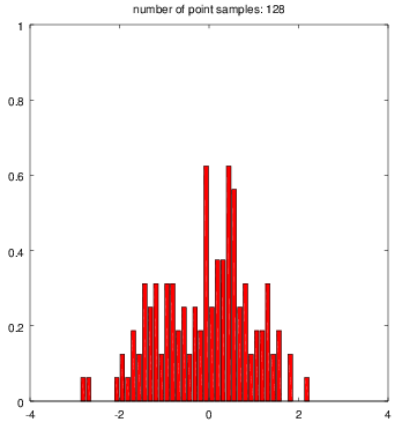
⚠This slide is animated (works with Acrobat Reader).

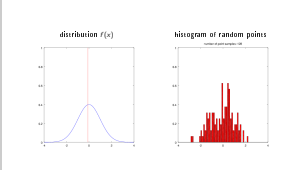
1.1 Introduction to segment sampling

distribution $f(x)$



histogram of random points

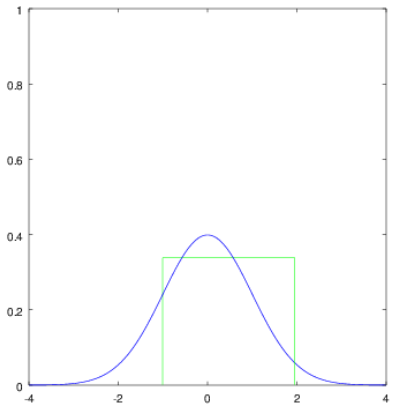




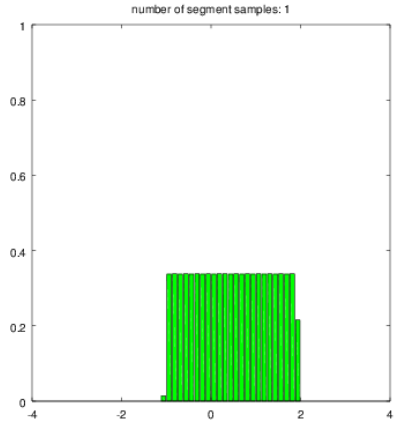
⚠This is a back-up slide if animations don't work.

1.1 Introduction to segment sampling

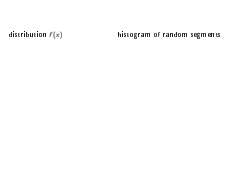
distribution $f(x)$



histogram of random segments



Generating Random Segments from Non-Uniform Distributions



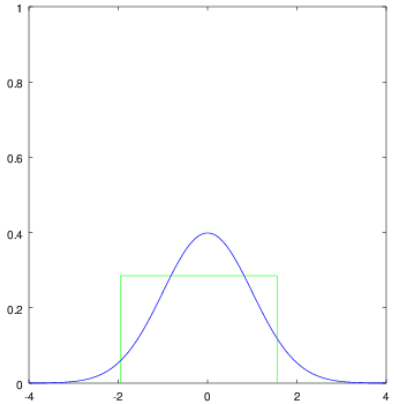
As for points, generating random segments from f means that their histogram converges towards f.

A segment is associated with a normalized uniform distribution and all the segment samples have the same contribution to the histogram. Adding a segment to the histogram means distributing its contribution over the bins covered by this segment. The larger a segment is, the more its contribution is distributed over the bins, and the smaller its per-bin contribution is.

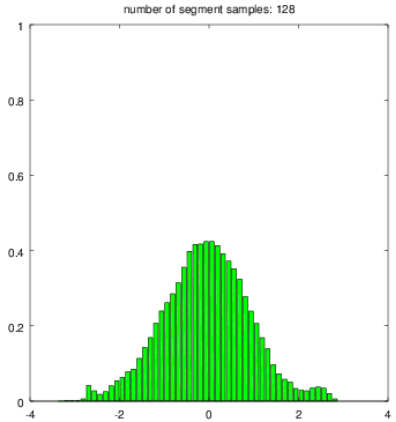
⚠This slide is animated (works with Acrobat Reader).

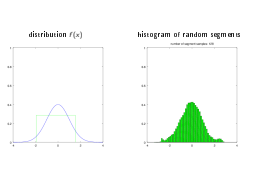
1.1 Introduction to segment sampling

distribution $f(x)$



histogram of random segments

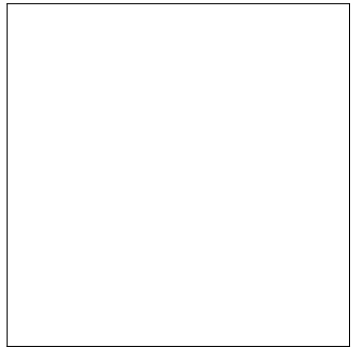




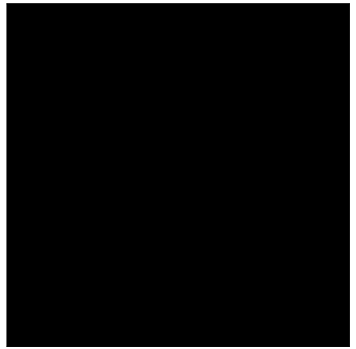
⚠This is a back-up slide if animations don't work.

1.1 Introduction to segment sampling

distribution $f(x)$

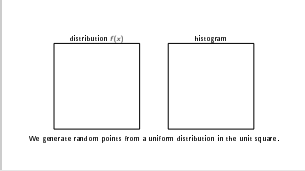


histogram



We generate random points from a uniform distribution in the unit square.

Generating Random Segments from Non-Uniform Distributions



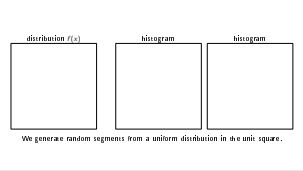
In this example, we generate random points uniformly in the unit square.
Their histogram converges towards a uniform distribution.

⚠This slide is animated (works with Acrobat Reader).

1.1 Introduction to segment sampling

distribution $f(x)$ | histogram | histogram

We generate random segments from a uniform distribution in the unit square.



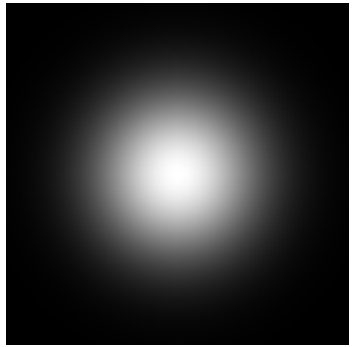
We generate random segments such that their histogram converges towards a uniform distribution.

The direction of the segments can be chosen arbitrarily. It can be constant or distributed randomly following an arbitrary directional distribution.

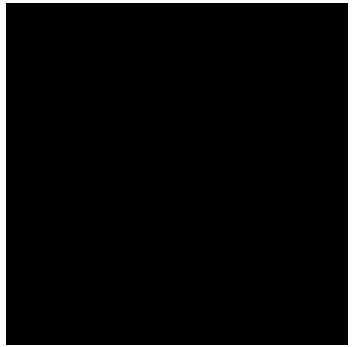
⚠This slide is animated (works with Acrobat Reader).

1.1 Introduction to segment sampling

distribution $f(x)$

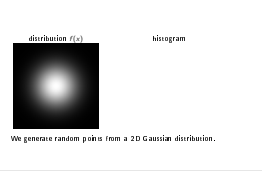


histogram



We generate random points from a 2D Gaussian distribution.

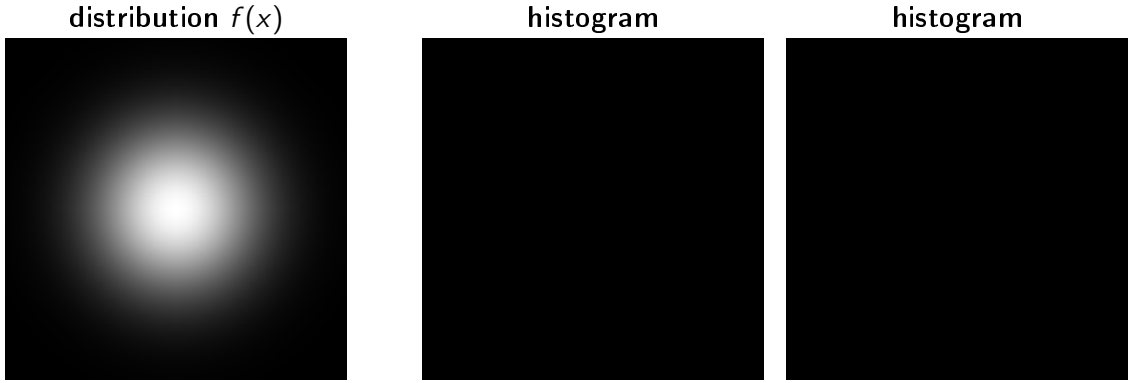
Generating Random Segments from Non-Uniform Distributions



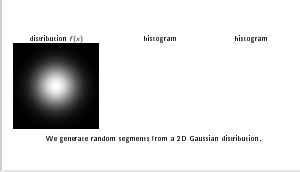
In this example, we generate random points from a bivariate Gaussian distribution (for instance with a Box-Muller transform).

⚠This slide is animated (works with Acrobat Reader).

1.1 Introduction to segment sampling



We generate random segments from a 2D Gaussian distribution.



We generate random segments such that their histogram converges towards a bivariate Gaussian distribution.

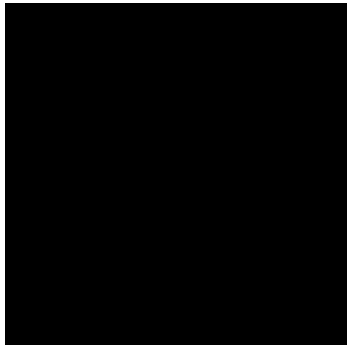
The direction of the segments can be chosen arbitrarily. It can be constant or distributed randomly following an arbitrary directional distribution.

⚠This slide is animated (works with Acrobat Reader).

1.1 Introduction to segment sampling

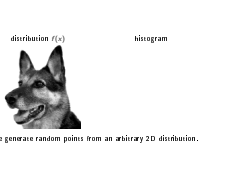
distribution $f(x)$

histogram



We generate random points from an arbitrary 2D distribution.

Generating Random Segments from Non-Uniform Distributions

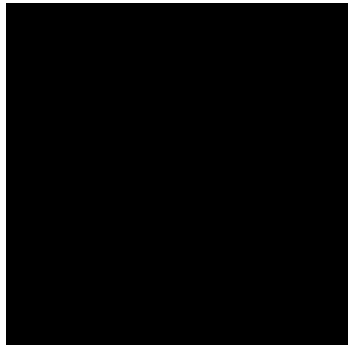


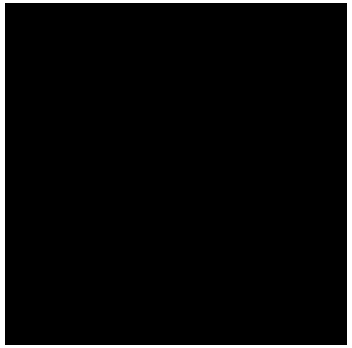
In this example, we generate random points from an arbitrary distribution (for instance with rejection sampling).

⚠This slide is animated (works with Acrobat Reader).

1.1 Introduction to segment sampling

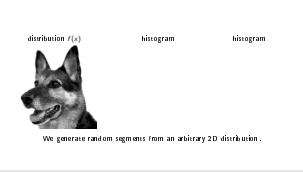
distribution $f(x)$ | histogram | histogram





We generate random segments from an arbitrary 2D distribution.

Generating Random Segments from Non-Uniform Distributions

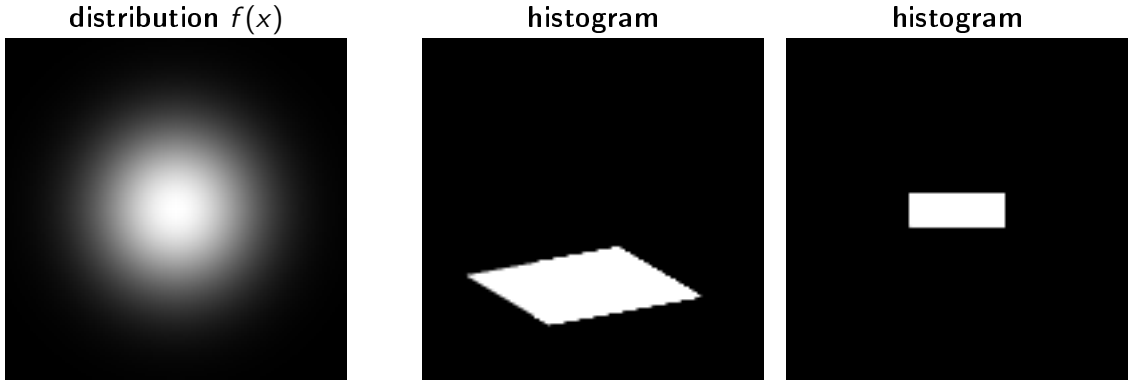


We generate random segments such that their histogram converges towards the target arbitrary distribution.

The direction of the segments can be chosen arbitrarily. It can be constant or distributed randomly following an arbitrary directional distribution.

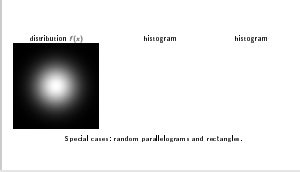
⚠This slide is animated (works with Acrobat Reader).

1.1 Introduction to segment sampling



Special cases: random parallelograms and rectangles.

Generating Random Segments from Non-Uniform Distributions



In some special cases, it is possible to sample higher-dimensional primitives.

Convolved distributions: the summation of independent random variates is distributed in the convolution of their distributions. Similarly, if the distribution is the result of a convolution of two distributions (e.g. a Gaussian can be obtained by convolving two Gaussians), we obtain random parallelograms by generating random segments and convolving them.

Separable distributions: if the distribution is separable: $f(x,y) = f_X(x) f_Y(y)$, we obtain random rectangles by computing the Cartesian product of random segments generated from f_X and f_Y.

⚠This slide is animated (works with Acrobat Reader).

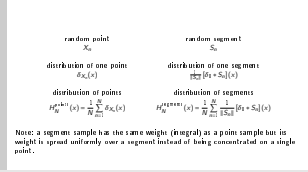
1.2 Motivation: Monte Carlo integration

1.2 Motivation: Monte Carlo integration

random point	random segment
X_n	S_n
distribution of one point	**distribution of one segment**
$\delta_{X_n}(x)$	$\frac{1}{\|S_n\|}[\delta_0 * S_n](x)$
distribution of points	**distribution of segments**
$H_N^{\text{points}}(x) = \frac{1}{N}\sum_{n=1}^{N}\delta_{X_n}(x)$	$H_N^{\text{segments}}(x) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}[\delta_0 * S_n](x)$

Note: a segment sample has the same weight (integral) as a point sample but its weight is spread uniformly over a segment instead of being concentrated on a single point.

Generating Random Segments from Non-Uniform Distributions



The Monte Carlo estimator based on point sampling associates normalized Dirac delta distributions with the point samples:
$\int_{\mathscr{R}^n} \delta_{X_n}(x)\,\mathrm{d}x = 1.$

Similarly, the Monte Carlo estimator based on segment sampling associates 1D Dirac delta distributions with the segments (i.e. the convolution of the Dirac delta distribution δ_0 centered in 0 and uniform distributions over the segments). The distribution associated with each segment is normalized and uniform over the segment:
$\int_{\mathscr{R}^n} \frac{1}{\|S_n\|}[\delta_0 * S_n](x)\,\mathrm{d}x = 1.$

1.2 Motivation: Monte Carlo integration

Objective: estimate $I = \int_{\mathscr{R}^n} f(x)g(x)\,\mathrm{d}x$ **using** N **random samples from** f.

Monte Carlo estimator (points)

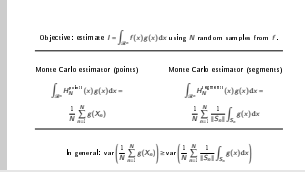
$$\int_{\mathscr{R}^n} H_N^{\text{points}}(x)g(x)\,\mathrm{d}x = \frac{1}{N}\sum_{n=1}^{N} g(X_n)$$

Monte Carlo estimator (segments)

$$\int_{\mathscr{R}^n} H_N^{\text{segments}}(x)g(x)\,\mathrm{d}x = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x$$

In general: $\mathbf{var}\left(\frac{1}{N}\sum_{n=1}^{N} g(X_n)\right) \geq \mathbf{var}\left(\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x\right)$

Generating Random Segments from Non-Uniform Distributions



Generating random samples from f can be used for Monte Carlo integration against another function g. One can imagine that we are simulating the reception of a signal g using a sensor f. The measure is given by the integral of their product:

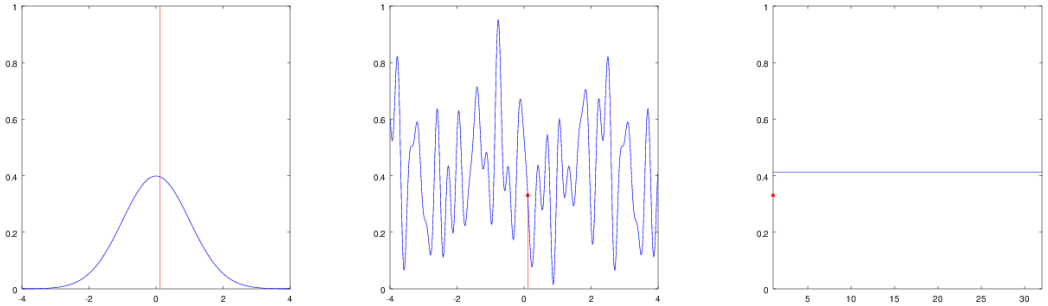
$$I = \int_{\mathbb{R}} f(x)g(x)\,\mathrm{d}x$$

The Monte Carlo estimator with point samples averages punctual evaluations of g.

The Monte Carlo estimator with segment samples averages the average values of g over the segments. Since this estimator averages values that are already averaged of values of g it has generally lower variance.

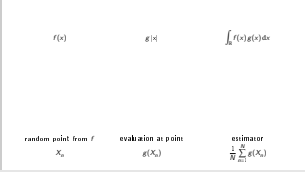
1.2 Motivation: Monte Carlo integration

$f(x)$ | $g(x)$ | $\int_{\mathbb{R}} f(x)g(x)\,\mathrm{d}x$



random point from f | **evaluation at point** | **estimator**

X_n | $g(X_n)$ | $\frac{1}{N}\sum_{n=1}^{N} g(X_n)$



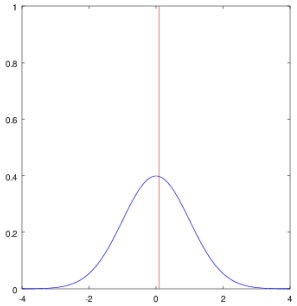
Convergence of the point-sample estimator on a 1D example.

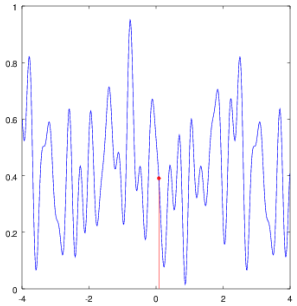
$$\lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} g(X_n) = \int_{\mathbb{R}} f(x)g(x)\,\mathrm{d}x$$

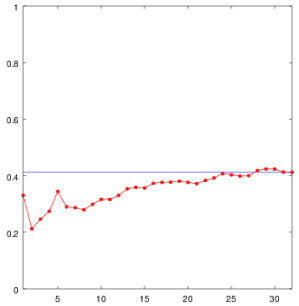
⚠This slide is animated (works with Acrobat Reader).

1.2 Motivation: Monte Carlo integration

$f(x)$ $g(x)$ $\int_{\mathbb{R}} f(x)g(x)\,\mathrm{d}x$







random point from f

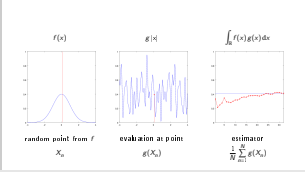
X_n

evaluation at point

$g(X_n)$

estimator

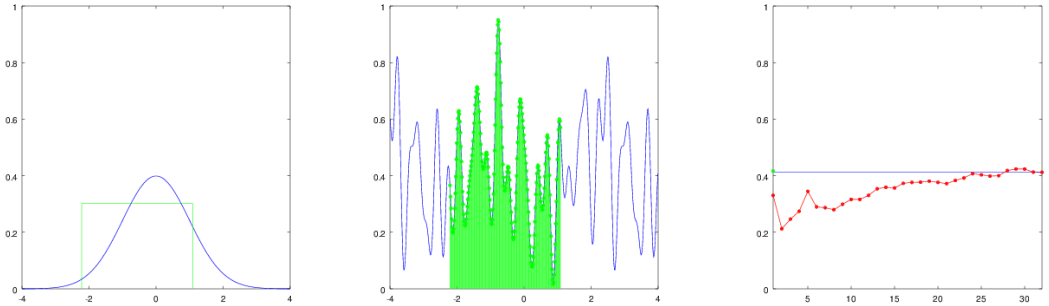
$$\frac{1}{N}\sum_{n=1}^{N} g(X_n)$$



⚠This is a back-up slide if animations don't work.

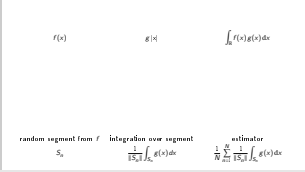
1.2 Motivation: Monte Carlo integration

$f(x)$ | $g(x)$ | $\int_{\mathbb{R}} f(x)g(x)\,\mathrm{d}x$



random segment from f | **integration over segment** | **estimator**

S_n | $\frac{1}{\|S_n\|}\int_{S_n} g(x)\,dx$ | $\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x$



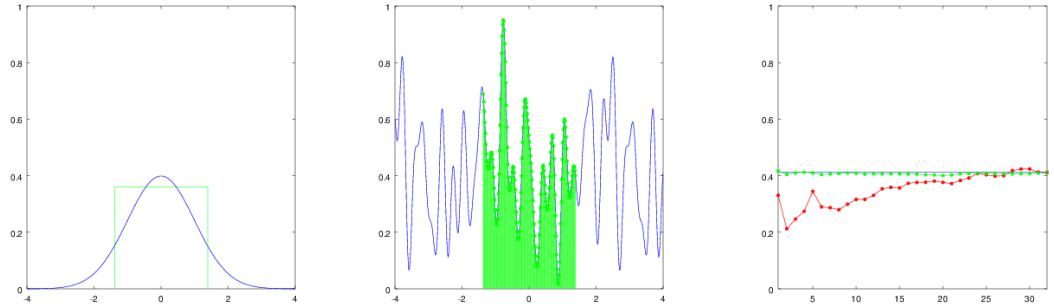
Convergence of the segment-sample estimator on a 1D example.

$$\lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x = \int_{\mathbb{R}} f(x)g(x)\,\mathrm{d}x$$

⚠This slide is animated (works with Acrobat Reader).

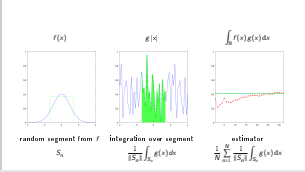
1.2 Motivation: Monte Carlo integration

$f(x)$ | $g(x)$ | $\int_{\mathbb{R}} f(x)g(x)\,\mathrm{d}x$



random segment from f | **integration over segment** | **estimator**

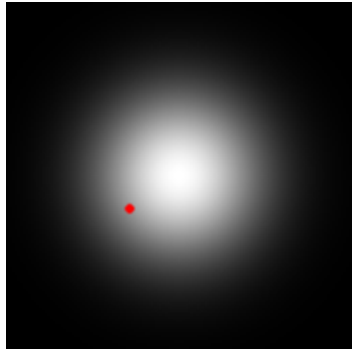
S_n | $\frac{1}{\|S_n\|}\int_{S_n} g(x)\,dx$ | $\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x$

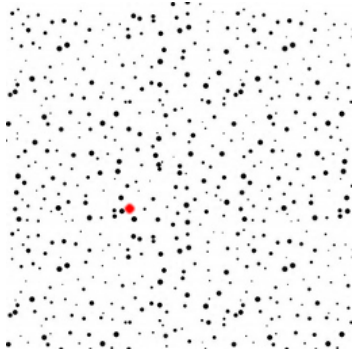


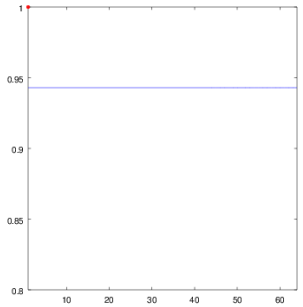
⚠This is a back-up slide if animations don't work.

1.2 Motivation: Monte Carlo integration

$f(x,y)$ | $g(x,y)$ | $\int_{\mathbb{R}^2} f(x,y)g(x,y)\,\mathrm{d}x\mathrm{d}y$







random point from f

X_n

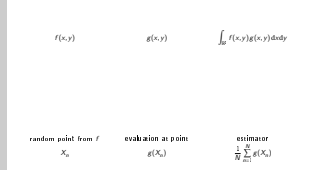
evaluation at point

$g(X_n)$

estimator

$$\frac{1}{N}\sum_{n=1}^{N} g(X_n)$$

Generating Random Segments from Non-Uniform Distributions

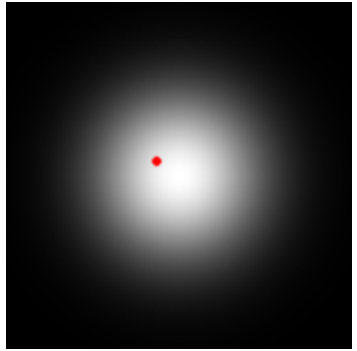


Convergence of the point-sample estimator on a 2D example.

⚠This slide is animated (works with Acrobat Reader).

1.2 Motivation: Monte Carlo integration

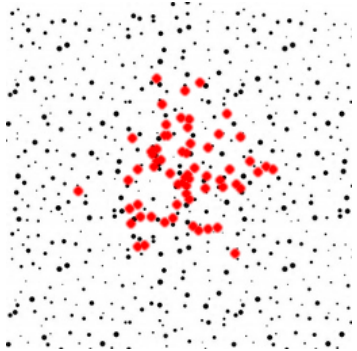
$f(x,y)$



random point from f

X_n

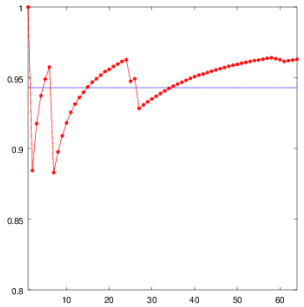
$g(x,y)$



evaluation at point

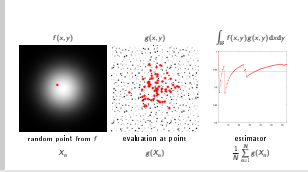
$g(X_n)$

$\int_{\mathbb{R}^2} f(x,y)g(x,y)\,\mathrm{d}x\mathrm{d}y$



estimator

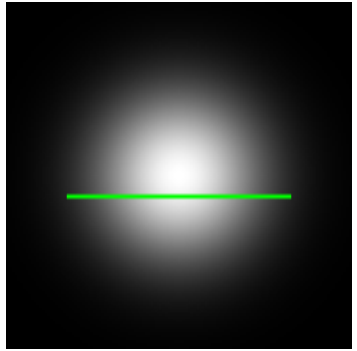
$$\frac{1}{N}\sum_{n=1}^{N} g(X_n)$$



⚠This is a back-up slide if animations don't work.

1.2 Motivation: Monte Carlo integration

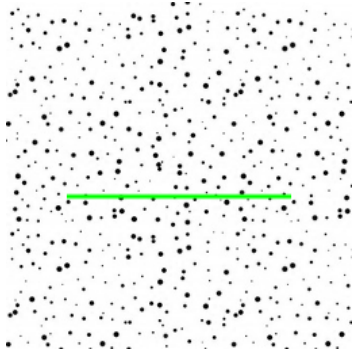
$f(x,y)$



random segment from f

S_n

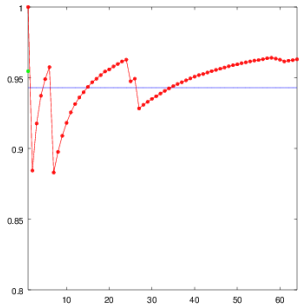
$g(x,y)$



integration over segment

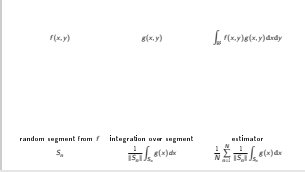
$$\frac{1}{\|S_n\|}\int_{S_n} g(x)\,dx$$

$$\int_{\mathbb{R}^2} f(x,y)g(x,y)\,\mathrm{d}x\mathrm{d}y$$



estimator

$$\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x$$



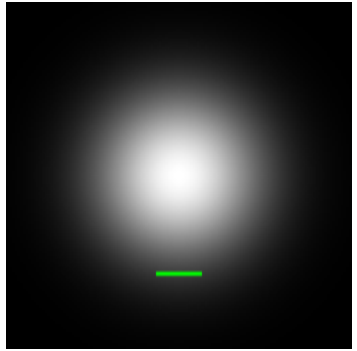
Convergence of the segment-sample estimator on a 2D example.

With 2D images, the integral of g over segments can be computed efficiently with a Summed Area Table (SAT).

⚠This slide is animated (works with Acrobat Reader).

1.2 Motivation: Monte Carlo integration

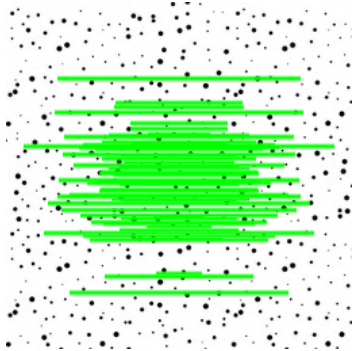
$f(x,y)$



random segment from f

S_n

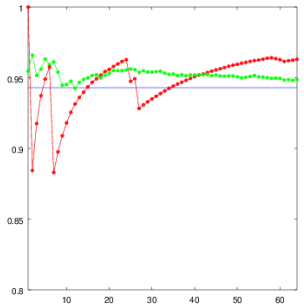
$g(x,y)$



integration over segment

$$\frac{1}{\|S_n\|}\int_{S_n} g(x)\,dx$$

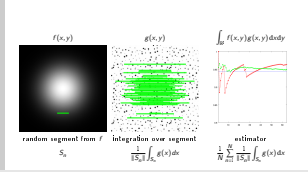
$$\int_{\mathbb{R}^2} f(x,y)g(x,y)\,\mathrm{d}x\mathrm{d}y$$



estimator

$$\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x$$

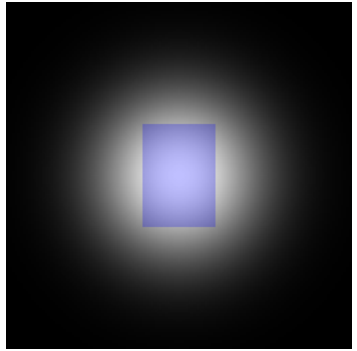
Generating Random Segments from Non-Uniform Distributions

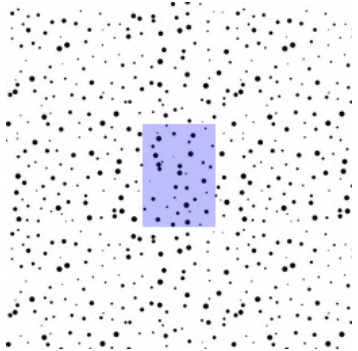


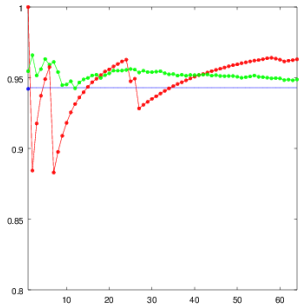
⚠This is a back-up slide if animations don't work.

1.2 Motivation: Monte Carlo integration

$f(x,y)$ | $g(x,y)$ | $\int_{\mathbb{R}^2} f(x,y)g(x,y)\,\mathrm{d}x\mathrm{d}y$







random rectangle from f

R_n

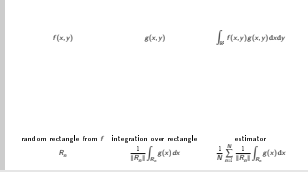
integration over rectangle

$$\frac{1}{\|R_n\|}\int_{R_n} g(x)\,dx$$

estimator

$$\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|R_n\|}\int_{R_n} g(x)\,\mathrm{d}x$$

Generating Random Segments from Non-Uniform Distributions



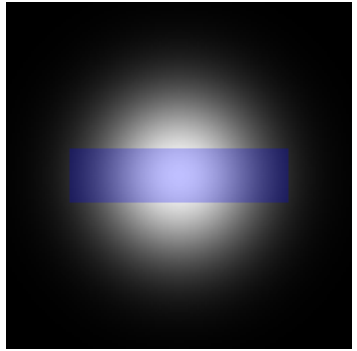
Convergence of the rectangle-sample estimator on a 2D example.

With 2D images, the integral of g over rectangles can be computed efficiently with a Summed Area Table (SAT).

⚠This slide is animated (works with Acrobat Reader).

1.2 Motivation: Monte Carlo integration

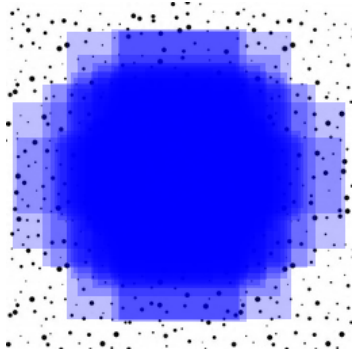
$f(x,y)$



random rectangle from f

R_n

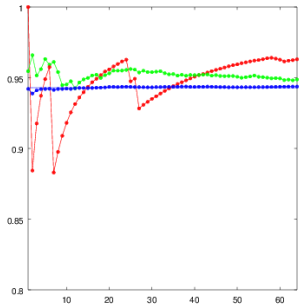
$g(x,y)$



integration over rectangle

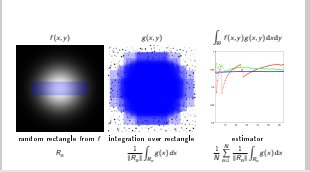
$$\frac{1}{\|R_n\|}\int_{R_n} g(x)\,dx$$

$$\int_{\mathbb{R}^2} f(x,y)\,g(x,y)\,\mathrm{d}x\mathrm{d}y$$



estimator

$$\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|R_n\|}\int_{R_n} g(x)\,\mathrm{d}x$$



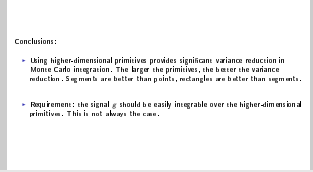
⚠This is a back-up slide if animations don't work.

1.2 Motivation: Monte Carlo integration

Conclusions:

- Using higher-dimensional primitives provides significant variance reduction in Monte Carlo integration. The larger the primitives, the better the variance reduction. Segments are better than points, rectangles are better than segments.

- Requirement: the signal g should be easily integrable over the higher-dimensional primitives. This is not always the case.

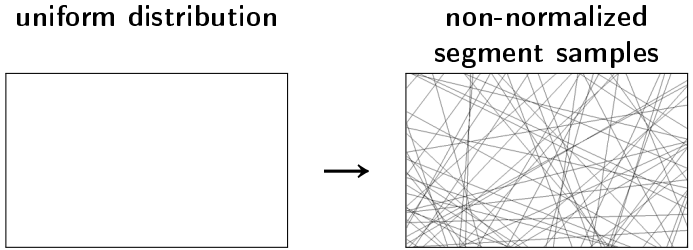

Conclusions:
- Using higher-dimensional primitives provides significant variance reduction in Monte Carlo integration. The larger the primitives, the better the variance reduction. Segments are better than points, rectangles are better than segments.
- Requirement: the signal g should be easily integrable over the higher-dimensional primitives. This is not always the case.

Monte Carlo integration with segment samples requires the ability to integrate efficiently g over the segments, i.e. compute $\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x$. This integral is simple to obtain in some problems but this is not always the case. For some problems, segment sampling is not usable in practice.

2. Previous work on uniform distributions and applications

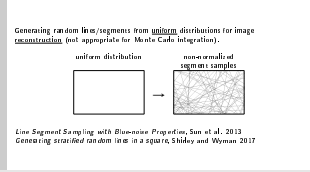
2. Previous work on uniform distributions and applications

Generating random lines/segments from uniform distributions for image reconstruction (not appropriate for Monte Carlo integration).



Line Segment Sampling with Blue-noise Properties, Sun et al. 2013
Generating stratified random lines in a square, Shirley and Wyman 2017

Generating Random Segments from Non-Uniform Distributions



Some previous works have been dedicated to generate random segments.
These approaches have two differences with our work:

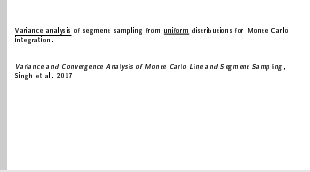
1. They focus on generating random segments from uniform distributions. We target the more general case of non-uniform distribution.
2. These approaches cannot be used for Monte Carlo integration because the segment samples they generate are non-normalized distributions because all the segments have the same amplitude. In our case, the segments are normalized distributions and their amplitude is the inverse of their length. This property is important for Monte Caro integration (see our FAQ section).

2. Previous work on uniform distributions and applications

Variance analysis of segment sampling from uniform distributions for Monte Carlo integration.

Variance and Convergence Analysis of Monte Carlo Line and Segment Sampling, Singh et al. 2017

Generating Random Segments from Non-Uniform Distributions



Singh et al. use a Fourier analysis to investigate how segment sampling reduces variance for Monte Carlo integration in comparison to classic point sampling. They study the case where random segments are generated from a uniform distribution.

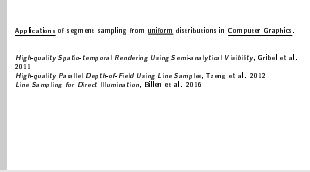
Their investigation shows that using segments instead of points yields significant variance reduction for many applications.

2. Previous work on uniform distributions and applications

Applications of segment sampling from uniform distributions in Computer Graphics.

High-quality Spatio-temporal Rendering Using Semi-analytical Visibility, Gribel et al. 2011
High-quality Parallel Depth-of-Field Using Line Samples, Tzeng et al. 2012
Line Sampling for Direct Illumination, Billen et al. 2016

Generating Random Segments from Non-Uniform Distributions



Segment sampling from uniform distributions has been used in several Computer Graphics applications.

2. Previous work on uniform distributions and applications

Conclusions from previous work:

- The benefits of segment sampling are supported by theoretical analyses and empirical evidences in Computer Graphics applications.
- Segment sampling is currently limited to uniform distributions.

Focus of this presentation:

- We generalize segment sampling to non-uniform distributions.

Conclusions from previous work:
- The benefits of segment sampling are supported by theoretical analyses and empirical evidences in Computer Graphics applications.
- Segment sampling is currently limited to uniform distributions.

Focus of this presentation:
- We generalize segment sampling to non-uniform distributions.

3. Generating random segments from non-uniform distributions

Generating Random Segments from Non-Uniform Distributions

3. Generating random segments from non-uniform distributions

So far, we have motivated segment sampling by showing how to use it for Monte Carlo rendering but we did not explain how they can actually be generated.

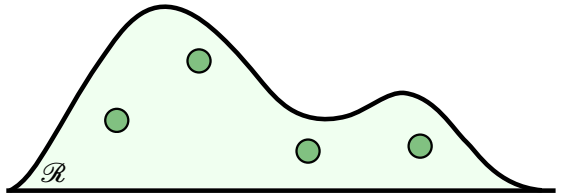
We propose 3 algorithms that are inspired by classic point sampling methods:

- **rejection sampling**,
- **slice sampling**,
- **inverse-CDF sampling**.

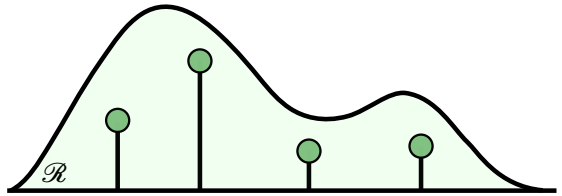
3.1 General approach

3.1 General approach

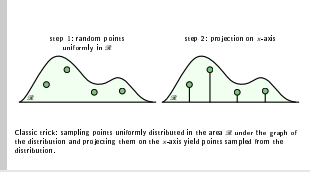
step 1: random points uniformly in $\mathscr{R}$



step 2: projection on x-axis



Classic trick: sampling points uniformly distributed in the area $\mathscr{R}$ under the graph of the distribution and projecting them on the x-axis yield points sampled from the distribution.

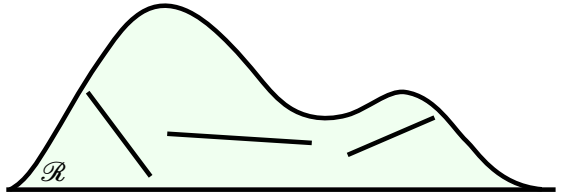


The trick of our segment sampling algorithms is to transform the problem of sampling from a non-uniform distribution into the problem of sampling from a uniform distribution.

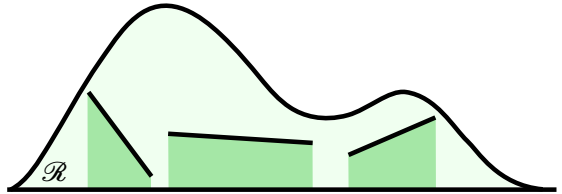
To do that, we leverage a well-known property of point sampling algorithms: one can obtain random points from a 1D non-uniform distribution by generating 2D random points uniformly in the area under the curve of the distribution and projecting them on the x-axis. The idea generalizes trivially to higher-dimensional distributions.

3.1 General approach

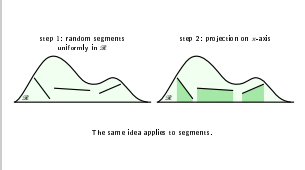
step 1: random segments uniformly in $\mathscr{R}$



step 2: projection on x-axis



The same idea applies to segments.

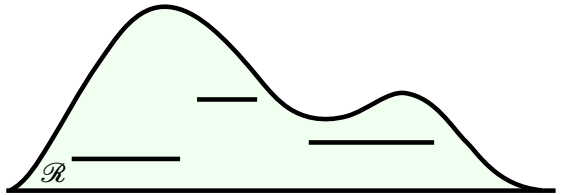


The same idea applies to segments: if we can random segments uniformly distributed in $\mathscr{R}$, we obtain segments from the distribution by projecting them on the x-axis.

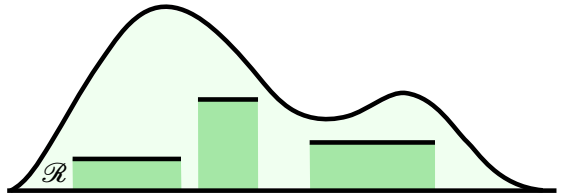
This is because a segment sample is a uniform distribution and the projection operator does not change its uniformity (because the Jacobian of an orthogonal projection is constant).

3.1 General approach

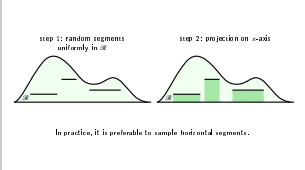
step 1: random segments uniformly in $\mathscr{R}$



step 2: projection on x-axis

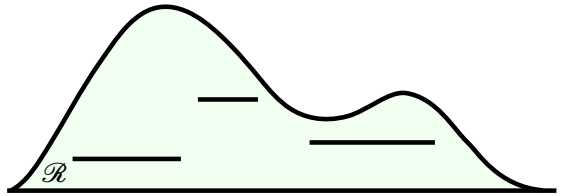


In practice, it is preferable to sample horizontal segments.



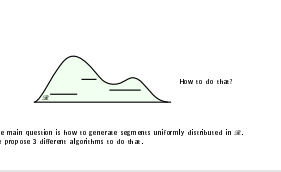
Note that the segments can be generated with arbitrary directions. However, in practice, it is preferable to sample horizontal segments. Indeed, horizontal segments produces longer segments after the projection on the x-axis, which is a desirable property: the longer the segments, the more they reduce the variance of the Monte Carlo estimator. In contrast, vertical segments project on a single point on the x-axis: sampling vertical segments is equivalent to sampling points.

3.1 General approach



How to do that?

The main question is how to generate segments uniformly distributed in $\mathscr{R}$.
We propose 3 different algorithms to do that.

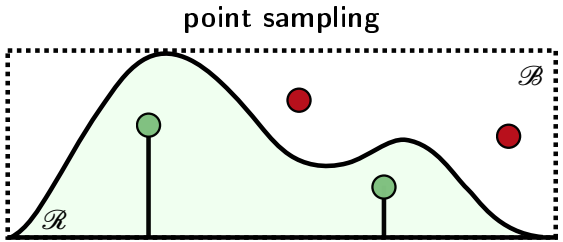


The question is now: "Is it possible to generate segments uniformly in the area (or volume) under the graph of the distribution?".

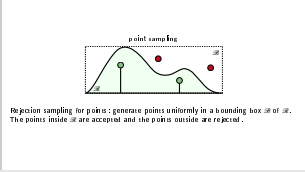
Doing this is turns out to be pretty simple and it is the focus of our algorithms.

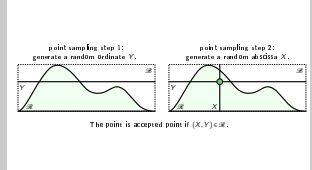
3.2 Rejection sampling

3.2 Rejection sampling



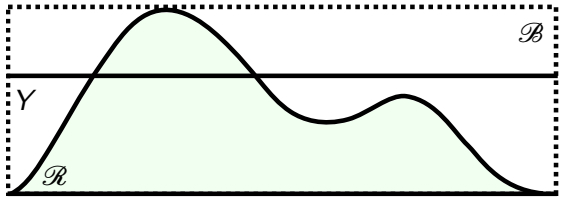
Rejection sampling for points: generate points uniformly in a bounding box $\mathscr{B}$ of $\mathscr{R}$. The points inside $\mathscr{R}$ are accepted and the points outside are rejected.



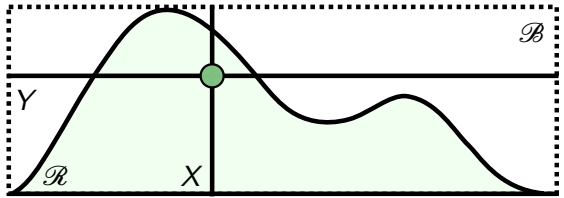


3.2 Rejection sampling

point sampling step 1:
generate a random ordinate Y.



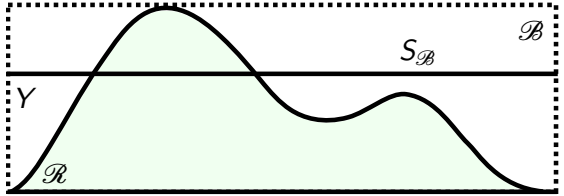
point sampling step 2:
generate a random abscissa X.



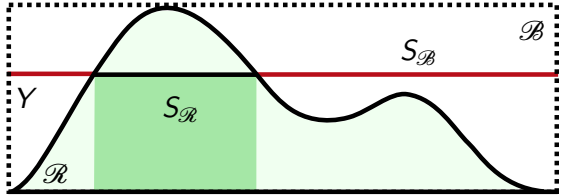
The point is accepted point if $(X, Y) \in \mathscr{R}$.

3.2 Rejection sampling

segment sampling step 1:
generate a random ordinate Y

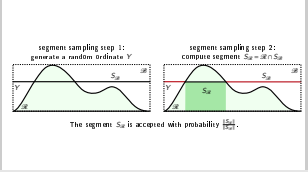


segment sampling step 2:
compute segment $S_{\mathscr{R}} = \mathscr{R} \cap S_{\mathscr{B}}$



The segment $S_{\mathscr{R}}$ is accepted with probability $\frac{\|S_{\mathscr{R}}\|}{\|S_{\mathscr{B}}\|}$.

Generating Random Segments from Non-Uniform Distributions



The first step for segments is the same as for the points.

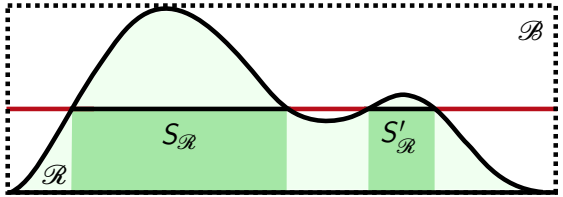
Note that the random Y yields a random segment from the uniform distribution in the bounding box $S_{\mathscr{B}}$.

In the second step, instead of choosing a single abscissa X, we want to consider all the valid abscissae at the same time. They yield a segment $S_{\mathscr{R}} \subseteq S_{\mathscr{B}}$.

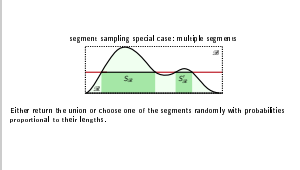
We accept $S_{\mathscr{R}}$ with probability $\frac{\|S_{\mathscr{R}}\|}{\|S_{\mathscr{B}}\|}$. This is the probability of generating a valid abscissa X given ordinate Y, or, equivalently, it is the probability that a point sampled $S_{\mathscr{B}}$ is inside $S_{\mathscr{R}}$.

3.2 Rejection sampling

segment sampling special case: multiple segments



Either return the union or choose one of the segments randomly with probabilities proportional to their lengths.



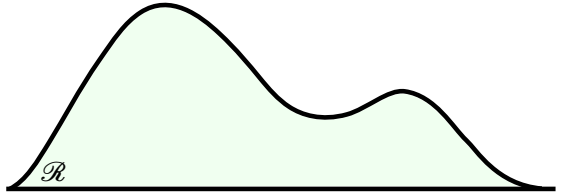
The part of the segment under the graph of the distribution can be the union of multiple disconnected segments $S_{\mathscr{R}} \cap S'_{\mathscr{R}}$. In this case, the probability of acceptance is $\frac{\|S_{\mathscr{R}} \cap S'_{\mathscr{R}}\|}{\|S_{\mathscr{B}}\|}$.

In case of acceptance, either the union can be returned (if the following calculation can deal with a union of segments), or one of the multiple segments can be chosen randomly with probabilities proportional to their respective lengths.

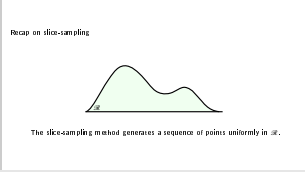
3.3 Slice sampling

3.3 Slice sampling

Recap on slice-sampling

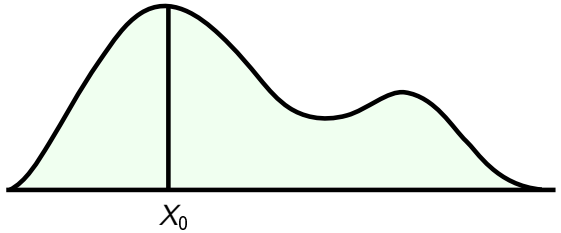


The slice-sampling method generates a sequence of points uniformly in $\mathscr{R}$.

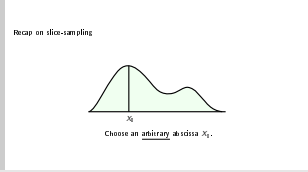


3.3 Slice sampling

Recap on slice-sampling



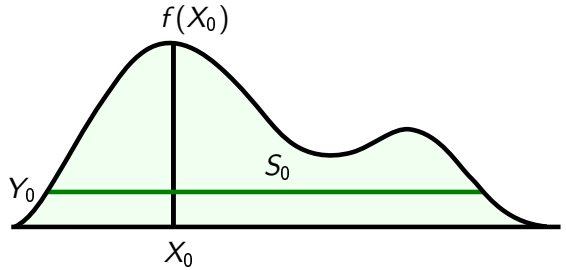
Choose an arbitrary abscissa X_0.



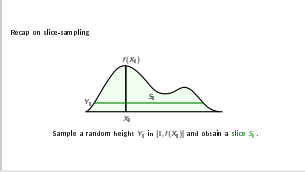
The first point of the sequence is chosen arbitrarily.

3.3 Slice sampling

Recap on slice-sampling

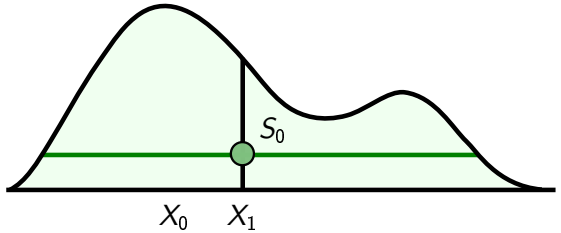


Sample a random height Y_0 in $[0, f(X_0)]$ and obtain a slice S_0.

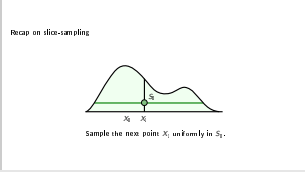


3.3 Slice sampling

Recap on slice-sampling

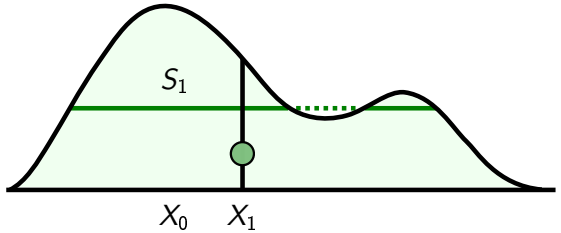


Sample the next point X_1 uniformly in S_0.

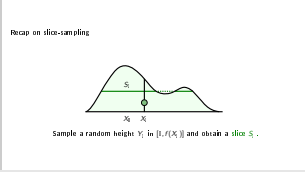


3.3 Slice sampling

Recap on slice-sampling

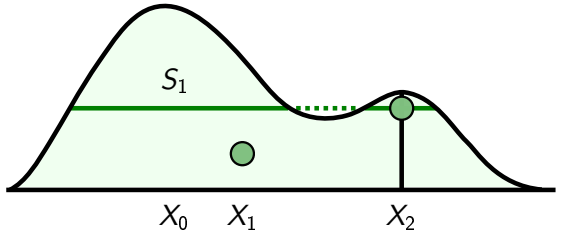


Sample a random height Y_1 in $[0, f(X_1)]$ and obtain a slice S_1.

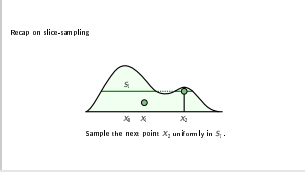


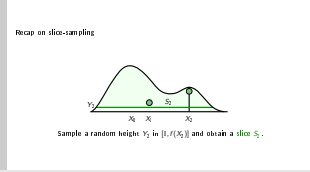
3.3 Slice sampling

Recap on slice-sampling



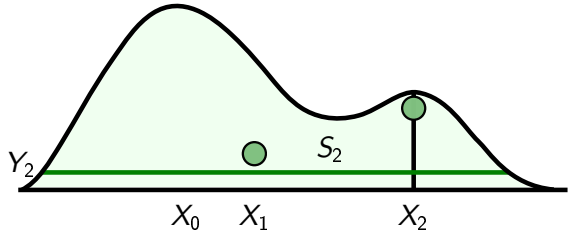
Sample the next point X_2 uniformly in S_1.





3.3 Slice sampling

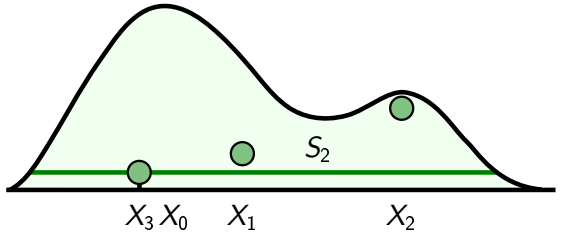
Recap on slice-sampling



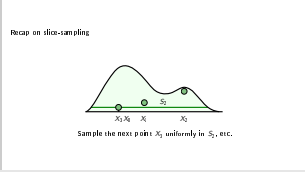
Sample a random height Y_2 in $[0, f(X_2)]$ and obtain a slice S_2.

3.3 Slice sampling

Recap on slice-sampling

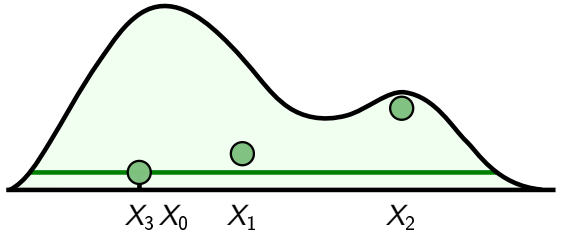


Sample the next point X_3 uniformly in S_2, etc.

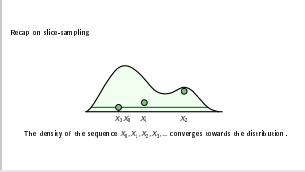


3.3 Slice sampling

Recap on slice-sampling

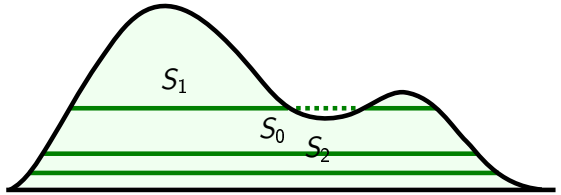


The density of the sequence $X_0, X_1, X_2, X_3, \ldots$ converges towards the distribution.

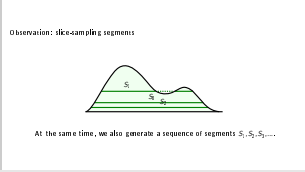


3.3 Slice sampling

Observation: slice-sampling segments



At the same time, we also generate a sequence of segments $S_1, S_2, S_3, \ldots$.

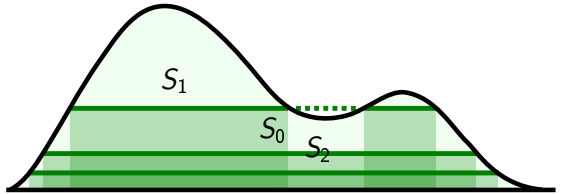


Each step of the slice-sampling method generates does not only generate a new point X_i, it also generates a slice X_i.

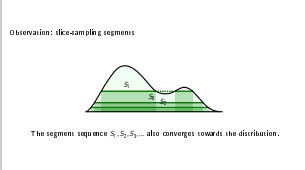
The slices S_i are actually segment samples (they are uniform distributions).

3.3 Slice sampling

Observation: slice-sampling segments



The segment sequence $S_1, S_2, S_3, \ldots$ also converges towards the distribution.



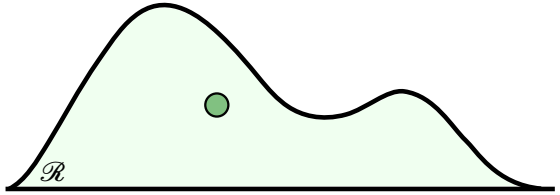
Furthermore, since each X_i is generated uniformly in S_i, we deduce that the sequence of segments converges towards the same distribution as the sequence of points.

Hence, the slice-sampling method is a MCMC method that generates at the same time a sequence of points and a sequence of segments from the distribution.

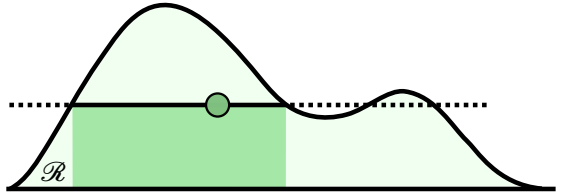
3.3 Slice sampling

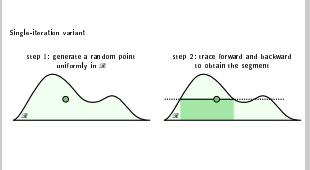
Single-iteration variant

step 1: generate a random point uniformly in $\mathscr{R}$



step 2: trace forward and backward to obtain the segment





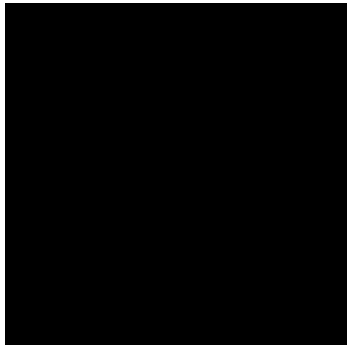
MCMC methods usually need a burn-in period before the sequence becomes effectively distributed in the target distribution. This is because of the bias due to the arbitrary choice of the starting point X_0.

However, if the starting point X_0 is sampled without bias using a point sampling algorithm, then the next point X_1 is also sampled without bias and so is the first segment S_1. Hence, if we use a point sampling algorithm to generate a starting point X_0, then the first segment is a valid segment sample from the distribution.

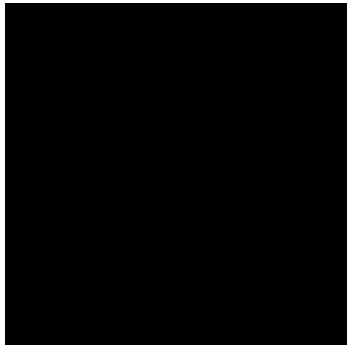
Furthermore, with this approach there is no need to consider multiple disconnected intersections with $\mathscr{R}$. Computing the connex segment inside $\mathscr{R}$ that contains the starting point is enough. This is because the unbiased choice of the starting point makes sure that all the regions of $\mathscr{R}$ can be sampled, even if $\mathscr{R}$ is not connex.

3.3 Slice sampling

same direction

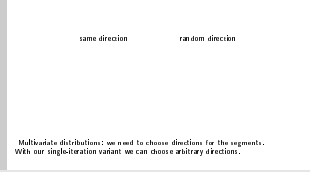


random direction



Multivariate distributions: we need to choose directions for the segments.
With our single-iteration variant we can choose arbitrary directions.

Generating Random Segments from Non-Uniform Distributions



If the distribution is multivariate, we need to make a choice for the directions of the horizontal lines in the slice-sampling method. For instance, if we sample segments from a 2D image, we need to choose a 2D direction for each segment.

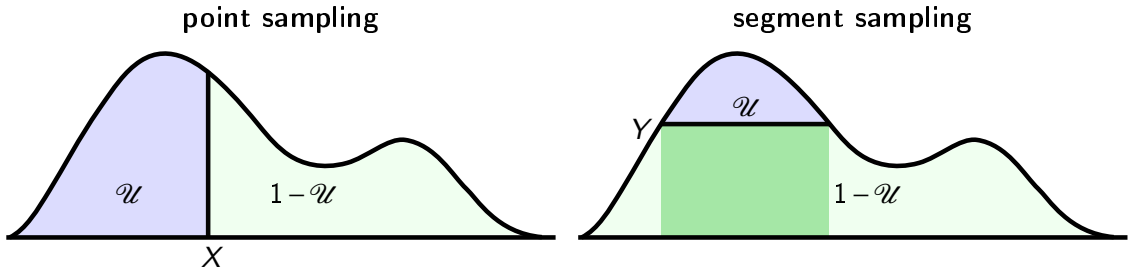
With the MCMC method, it is important that all the dimensions of the distribution are explored. If the same direction is used for each iteration, the sequence can only explore a single line instead of the full 2D space.

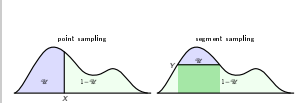
However, with our single-iteration variant, the choice of the direction is arbitrary: it can be always the same direction or randomly generated from a directional distribution. This is because the unbiased choice of the starting point X_0 ensures that all the dimensions are explored.

⚠This slide is animated (works with Acrobat Reader).

3.4 Inverse-CDF sampling

3.4 Inverse-CDF sampling

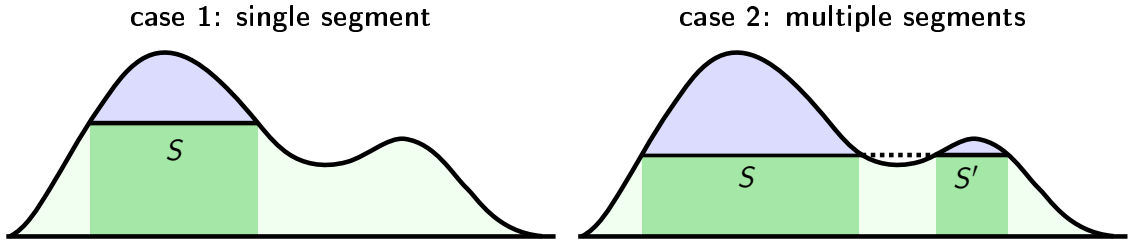


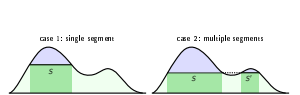


Inverse-CDF sampling for points: generate a random number $\mathscr{U}$ and compute the abscissa X that divides the domain in two parts: one of area $\mathscr{U}$ and one of area $1-\mathscr{U}$. The point X is a valid point sample from the distribution.
Remark: generating a uniform $Y \in [0, f(X)]$ yields a 2D point (X, Y) uniformly generated from $\mathscr{R}$.

Inverse-CDF sampling for segments: generate a random number $\mathscr{U}$ and compute the ordinate Y that divides the domain in two parts: one of area $\mathscr{U}$ and one of area $1-\mathscr{U}$. Any point generated uniformly in the segment of height Y inside $\mathscr{R}$ is a uniform point from $\mathscr{R}$. Hence, this segment is a valid segment sample from the distribution.

3.4 Inverse-CDF sampling





As for rejection sampling, it is possible that the set of points of height Y is the union of multiple segments $S \cap S'$. In this case, one can either return the union or choose one of the multiple segments randomly with probabilities proportionally to their respective lengths.

4. Implementation in C for 1D Gaussian distributions

4. Implementation in C for 1D Gaussian distributions

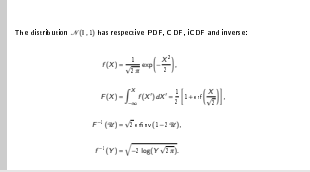
The distribution $\mathcal{N}(0,1)$ has respective PDF, CDF, iCDF and inverse:

$$f(X) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{X^2}{2}\right),$$

$$F(X) = \int_{-\infty}^{X} f(X')\,dX' = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{X}{\sqrt{2}}\right)\right],$$

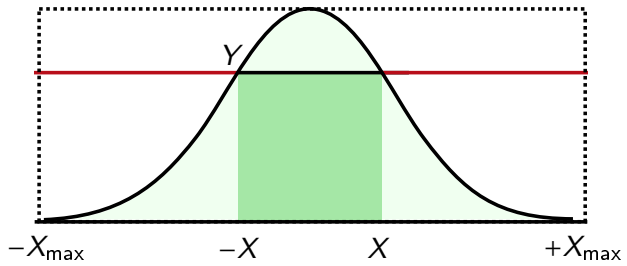
$$F^{-1}(\mathscr{U}) = \sqrt{2}\,\operatorname{erfinv}(1-2\mathscr{U}),$$

$$f^{-1}(Y) = \sqrt{-2\log(Y\sqrt{2\pi})}.$$

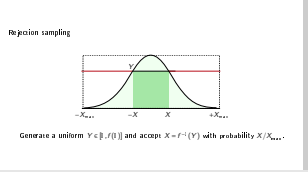


4. Implementation in C for 1D Gaussian distributions

Rejection sampling



Generate a uniform $Y \in [0, f(0)]$ and accept $X = f^{-1}(Y)$ with probability X/X_{max}.



For the rejection-sampling method we need a bounding box, which is problematic since the Gaussian distribution is unbounded. We make the approximation of clamping the distribution to the interval $[-X_{max}, X_{max}]$ with $X_{max} = 5$. The error on the integral of the distribution is lower than 10^{-6} outside this interval.

We generate uniform random heights $Y \in [0, f(0)]$ and we obtain a segment $[-X, X]$ with $X = f^{-1}(Y)$.

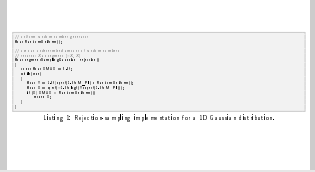
The acceptance probability of the segment $[-X, X]$ is X/X_{max}.

4. Implementation in C for 1D Gaussian distributions

```
// uniform random number generator
float RandomUniform();

// uses an undetermined amount of random numbers
// returns: X as segment [-X, X]
float segmentSamplingGaussian_rejection()
{
    const float XMAX = 5.0f;
    while(true)
    {
        float Y = 1.0f/sqrtf(2.0f*M_PI) * RandomUniform();
        float X = sqrtf(-2.0f*logf(Y*sqrtf(2.0f*M_PI)));
        if (X/XMAX > RandomUniform())
            return X;
    }
}
```

Listing 1: Rejection-sampling implementation for a 1D Gaussian distribution.

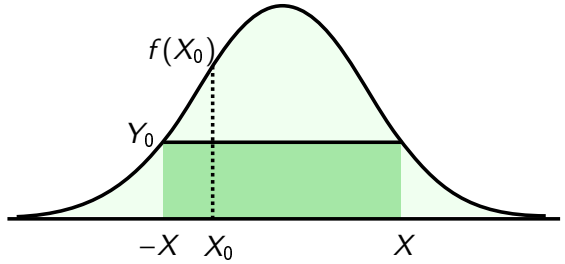


Listing 2: Rejection-sampling implementation for a 1D Gaussian distribution.

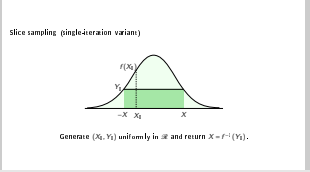
Implementation in C.

4. Implementation in C for 1D Gaussian distributions

Slice sampling (single-iteration variant)



Generate (X_0, Y_0) uniformly in $\mathscr{R}$ and return $X = f^{-1}(Y_0)$.



For the slice-sampling method, we use two uniform random numbers $\mathscr{U}_1$ and $\mathscr{U}_2$.

We start by sampling a random point with the classic inverse-CDF method $X_0 = F^{-1}(\mathscr{U}_1)$

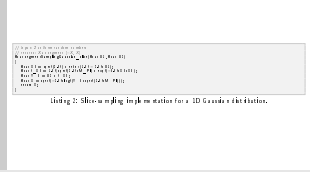
Then, we generate a random height $Y_0 = \mathscr{U}_2 f(X_0)$ and we compute the segment that contains all the points under the curve at height Y_0.

This segment is $[-X, X]$ with $X = f^{-1}(Y_0)$.

4. Implementation in C for 1D Gaussian distributions

```
// input: 2 uniform random numbers
// returns: X as segment [-X, X]
float segmentSamplingGaussian_slice(float U1, float U2)
{
    float X0 = sqrtf(2.0f) * erfinv(1.0f - 2.0f*U1);
    float f_X0 = 1.0f/sqrtf(2.0f*M_PI) * expf(-0.5f*X0*X0);
    float Y_0 = U2 * f_X0;
    float X = sqrtf(-2.0f*logf(Y_0*sqrtf(2.0f*M_PI)));
    return X;
}
```

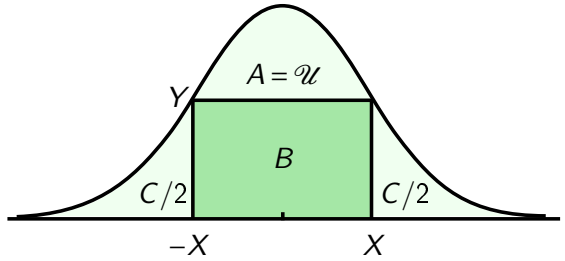
Listing 2: Slice-sampling implementation for a 1D Gaussian distribution.



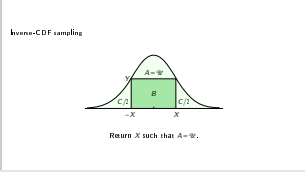
Implementation in C.

4. Implementation in C for 1D Gaussian distributions

Inverse-CDF sampling



Return X such that $A = \mathscr{U}$.



With the vertical inverse-CDF method, we use one uniform random number $\mathscr{U}$. The objective is to find X such that $A = \mathscr{U}$. We obtain:

$$A = \mathscr{U}, \;\; B = 2XY, \;\; C = 2(1 - F(X)), \;\; A + B + C = 1,$$

and the equation to solve is

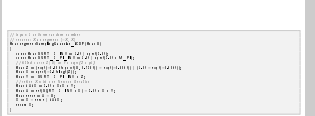
$$\mathscr{U} = 2F(X) - 2Xf(X) - 1.$$

Since this equation cannot be solved analytically, we implemented a numerical inversion algorithm. We made a fit of function $Z(U) = \sqrt{2\pi}\,Y \approx \frac{\exp(-0.806\,U^{0.728}) - \exp(-0.806)}{1 - \exp(-0.806)}$, from which we obtain $Y = Z/\sqrt{2\pi}$ and $X = f^{-1}(Y)$. Then, we refine the value of X with one Newton iteration. The maximal error of this implementation is $\max_{U \in [0,1]} |A - \mathscr{U}| < 10^{-3}$.

4. Implementation in C for 1D Gaussian distributions

```
// input: 1 uniform random number
// returns: X as segment [-X, X]
float segmentSamplingGaussian_iCDF(float U)
{
    const float SQRT_2_INV = 1.0f / sqrtf(2.0f);
    const float SQRT_2_PI_INV = 1.0f / sqrtf(2.0f * M_PI);
    // fitted curve Z(U) = Y * sqrt{2 * pi}
    float Z = (expf(-0.806f*powf(U, 0.728f)) - expf(-0.806f)) / (1.0f - expf(-0.806f));
    float X = sqrtf(-2.0f*logf(Z));
    float Y = SQRT_2_PI_INV * Z;
    // refine X with one Newton iteration
    float dAdX = 2.0f * X*X * Y;
    float A = erf(SQRT_2_INV * X) - 2.0f * X * Y;
    float error = A - U;
    X = X - error / dAdX;
    return X;
}
```

Listing 3: Inverse-CDF sampling implementation for a 1D Gaussian distribution.



Listing 3: Inverse-CDF sampling implementation for a 1D Gaussian distribution.

Implementation in C.

5. FAQ

5. FAQ

Why is it important that segment samples are normalized distributions?
In the Monte Carlo estimator:

$$\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x$$

normalizing by the factor $\frac{1}{\|S_n\|}$ makes sure that the varying lengths of the segments does not introduce variance in the estimator.

Consider the case were $g(x)=1$. In this case, the estimator has null variance:

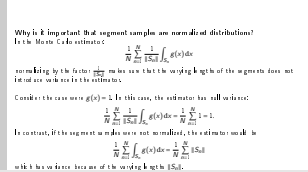
$$\frac{1}{N}\sum_{n=1}^{N}\frac{1}{\|S_n\|}\int_{S_n} g(x)\,\mathrm{d}x = \frac{1}{N}\sum_{n=1}^{N} 1 = 1.$$

In contrast, if the segment samples were not normalized, the estimator would be

$$\frac{1}{N}\sum_{n=1}^{N}\int_{S_n} g(x)\,\mathrm{d}x = \frac{1}{N}\sum_{n=1}^{N}\|S_n\|$$

which has variance because of the varying lengths $\|S_n\|$.

Generating Random Segments from Non-Uniform Distributions



In other words, using normalized segment samples makes sure that each segment weights the same in the estimator. Different weighting for the segments would introduce variance in the estimator even if the signal g to measure is constant. This would be undesirable.

Some previous work such as:
Line Segment Sampling with Blue-noise Properties, Sun et al. 2013
Generating stratified random lines in a square, Wyman et al. 2017
focus on generating non-normalized segments. Hence, they are not appropriate for Monte Carlo integration.

5. FAQ

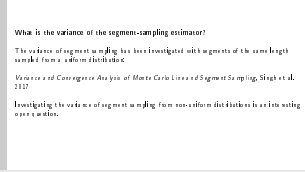
What is the variance of the segment-sampling estimator?

The variance of segment sampling has been investigated with segments of the same length sampled from a uniform distribution:

Variance and Convergence Analysis of Monte Carlo Line and Segment Sampling, Singh et al. 2017

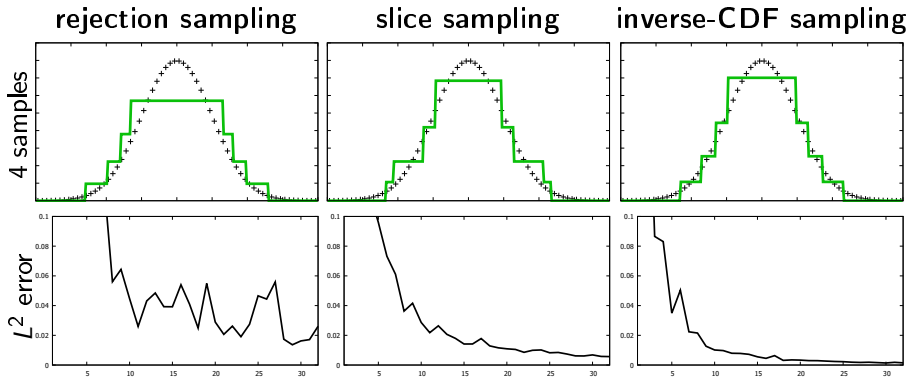
Investigating the variance of segment sampling from non-uniform distributions is an interesting open question.

Generating Random Segments from Non-Uniform Distributions

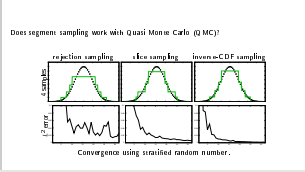


5. FAQ

Does segment sampling work with Quasi Monte Carlo (QMC)?



Convergence using stratified random number.



We compare the convergence of our methods applied to a 1D Gaussian distribution when they are used with stratified random numbers.

In our experiment, we obtained what one would expect: the less random numbers a method uses the better it parameterizes the sampling space and the better it converges with QMC.

In the case of the 1D Gaussian distribution:
Rejection sampling: undetermined amount of random numbers
Slice sampling: 2 random numbers
Inverse-CDF sampling: 1 random number