Lecture 4

Sampling:

Inverse Transform, Rejection Sampling, and Stratified Sampling

Announcements

- You will have upto 10am tomorrow on homework and subsequent homework
- You can take upto 5 late days (bumped up from 3). No more than 1 late day per homework

Last Time:

- Expectations and some notation
- The Law of large numbers
- Simulation and Monte Carlo for Integration
- Sampling and the CLT
- Errors in Monte Carlo

Expectation $E_f[X]$

$$E_f X = \int x dF(x) = \begin{cases} \sum_x x f(x) & \text{if X is discrete} \\ \int x f(x) dx & \text{if X is continuous} \end{cases}$$

LOTUS, if $Y = r(X)$:

$$E[Y] = \int r(x) dF(x)$$

If $r(X) = I_A(X)$, Indicator for event A, $p(X \in A) = E_F[I_A(X)] =$ frequentist probability

Law of Large numbers (LLN)

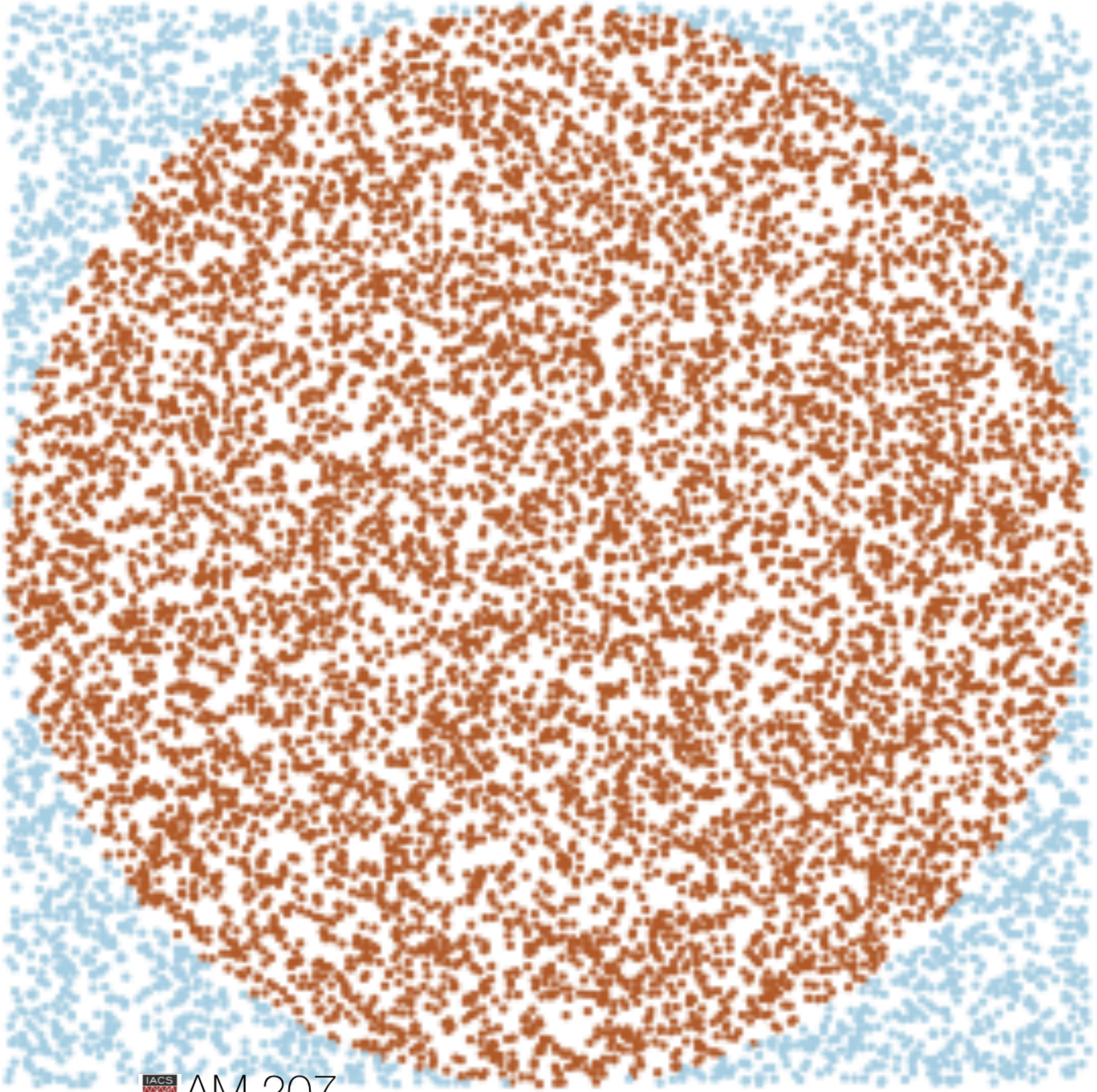
- Expectations become sample averages. Convergence for large N.

$$E_f[g] = \int g(x) dF = \int g(x) f(x) dx$$
$$= \lim_{n \to \infty} \frac{1}{N} \sum_{x_i \sim f} g(x_i)$$

- foundation of Monte Carlo techniques for expectations and integrals, which allow us to replace integration with summation

Central Limit Theorem

- note that we compute integrals from samples in one replication
- the sample averages are distributes around the true (distribution) expectation in a gaussian distribution with standard error $s = \frac{\sigma}{\sqrt{n}}$
- which mean to use depends on the accuracy you desire



Monte Carlo π

- LLN says throw rocks to compute expectation below
- $E_f[I_{\in C}(X,Y)] = \int\int_{\in C} f_{X,Y}(x,y)dxdy$
- which is probability of being in C
- If $f_{X,Y}(x,y) \sim Uniform(V)$:

$$= \frac{1}{V}\int\int_{\in C} dxdy = \frac{A}{V}$$

Formalize Monte Carlo Integration idea

For Uniform pdf: $U_{ab}(x) = 1/V = 1/(b-a)$

$$J = \int_a^b f(x) U_{ab}(x)\, dx = \int_a^b f(x)\, dx / V = I/V$$

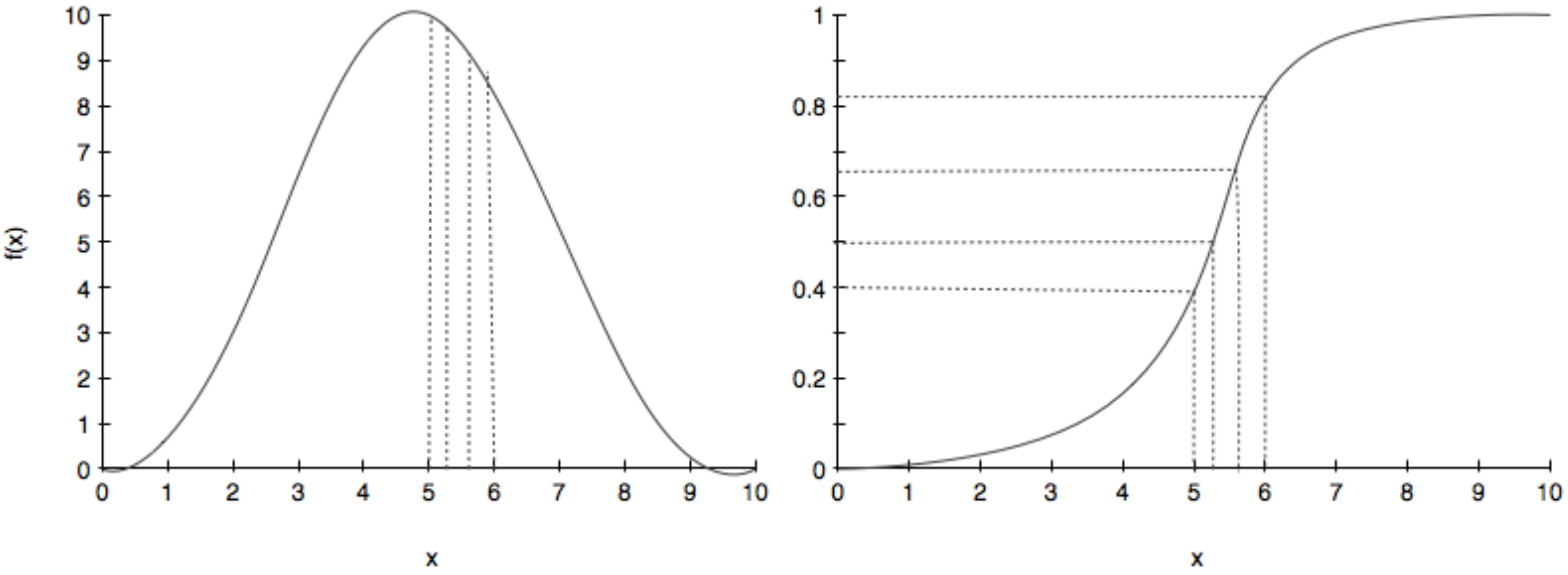
From LOTUS and the law of large numbers:

$$I = V \times J = V \times E_U[f] = V \times \lim_{n \to \infty} \frac{1}{N} \sum_{x_i \sim U} f(x_i)$$

Today: We need Samples

- to compute expectations, integrals and do statistics, we need samples
- we start that journey today
- inverse transform
- rejection sampling
- importance sampling: a direct, low-variance way to do integrals and expectations

Inverse transform



algorithm

The CDF F must be invertible!

1. get a uniform sample u from $Unif(0, 1)$
2. solve for x yielding a new equation $x = F^{-1}(u)$ where F is the CDF of the distribution we desire.
3. repeat.

Why does it work?

$$F^{-1}(u) = \text{smallest x such that } F(x) >= u$$

What distribution does random variable $y = F^{-1}(u)$ follow?

The CDF of y is $p(y <= x)$. Since F is monotonic:

$$p(y <= x) = p(F(y) <= F(x)) = p(u <= F(x)) = F(x)$$

F is the CDF of y, thus f is the pdf.

Example: exponential

pdf: $f(x) = \frac{1}{\lambda} e^{-x/\lambda}$ for $x \geq 0$ and $f(x) = 0$ otherwise.

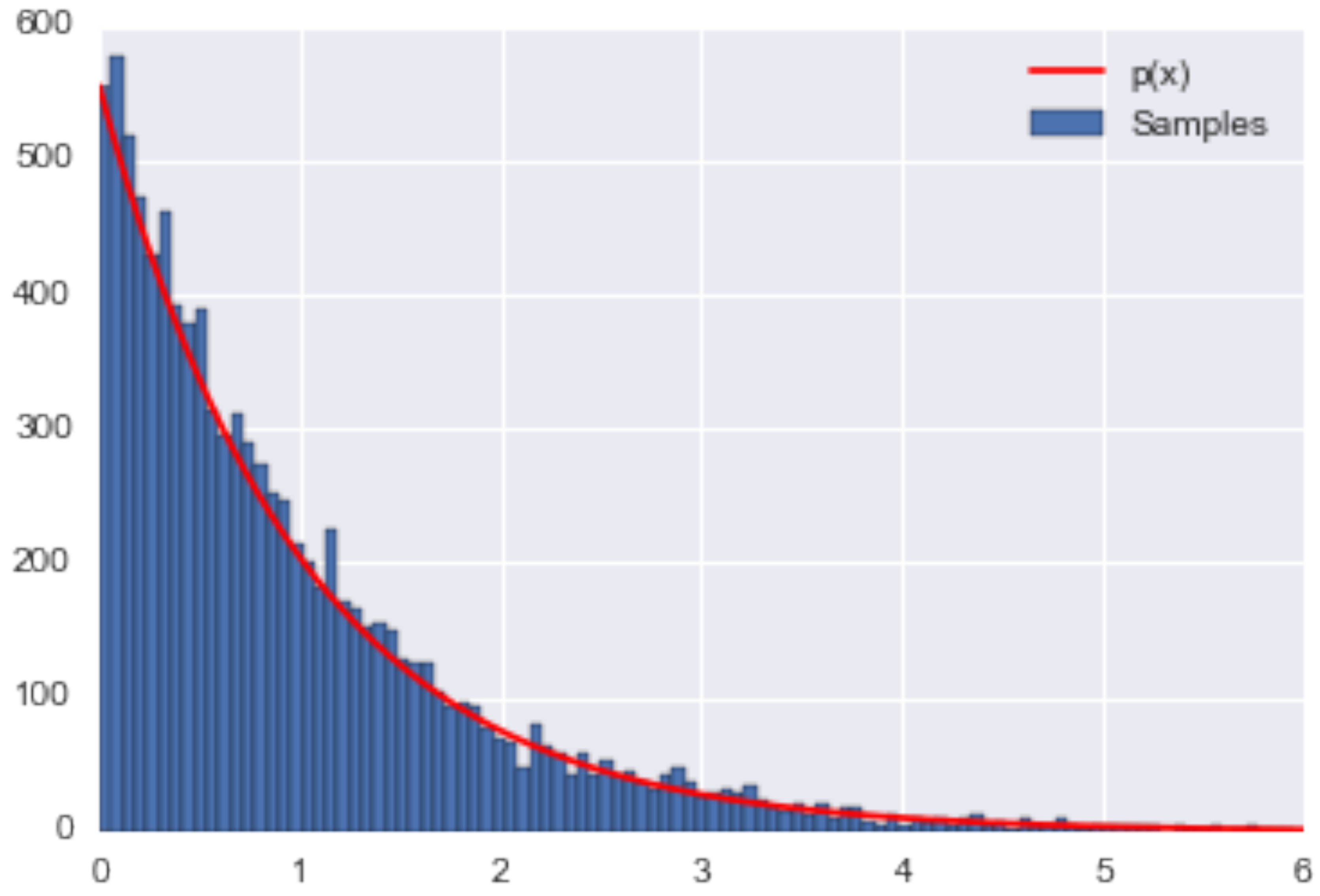
$$u = \int_0^x \frac{1}{\lambda} e^{-x'/\lambda} dx' = 1 - e^{-x/\lambda}$$

Solving for x

$$x = -\lambda \ln(1-u)$$

code

```
p = lambda x: np.exp(-x)
CDF = lambda x: 1-np.exp(-x)
invCDF = lambda r: -np.log(1-r) # invert the CDF
xmin = 0 # the lower limit of our domain
xmax = 6 # the upper limit of our domain
rmin = CDF(xmin)
rmax = CDF(xmax)
N = 10000
# generate uniform samples in our range then invert the CDF
# to get samples of our target distribution
R = np.random.uniform(rmin, rmax, N)
X = invCDF(R)
hinfo = np.histogram(X,100)
plt.hist(X,bins=100, label=u'Samples');
# plot our (normalized) function
xvals=np.linspace(xmin, xmax, 1000)
plt.plot(xvals, hinfo[0][0]*p(xvals), 'r', label=u'p(x)')
plt.legend()
```



Box-Muller

- how to draw from a normal?
- the CDF integral is not analytically solvable.

$$I = \frac{1}{2\pi} \int_{-\infty}^{x} e^{-x'^2/2} dx'$$

- can do numerical inversion (out of scope) or use box-muller trick.
-trick involves starting with two Normals $N(0, 1)$

$$X \sim N(0,1), Y \sim N(0,1) \implies X, Y \sim N(0,1)N(0,1)$$

pdf:

$$f_{XY}(x,y) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \times \frac{1}{\sqrt{2\pi}} e^{-y^2/2} = \frac{1}{2\pi} \times e^{-r^2/2}$$

where $r^2 = x^2 + y^2$.

Using polar co-ordinates r and θ, we have...

$$\Theta \sim Unif(0, 2\pi), S = R^2 \sim Exp(1/2)$$

$$s = r^2 = -2ln(1-u)$$

$$r = \sqrt{-2\, ln(u_1)}, \theta = 2\pi\, u_2$$

where u_1 and $u_2 \sim Unif(0, 1)$.

Now, use $x = r\, cos\theta, y = r\, sin\theta$ to obtain Normal samples.

What is $f_{R,\Theta}(r, \theta)$?

General transforms of a pdf

Let $z = g(x)$ so that $x = g^{-1}(z)$

Define the Jacobian $J(z)$ of the transformation $x = g^{-1}(z)$ as the partial derivatives matrix of the transformation.

Then:

$$f_Z(z) = f_X(g^{-1}(z)) \times det(J(z))$$

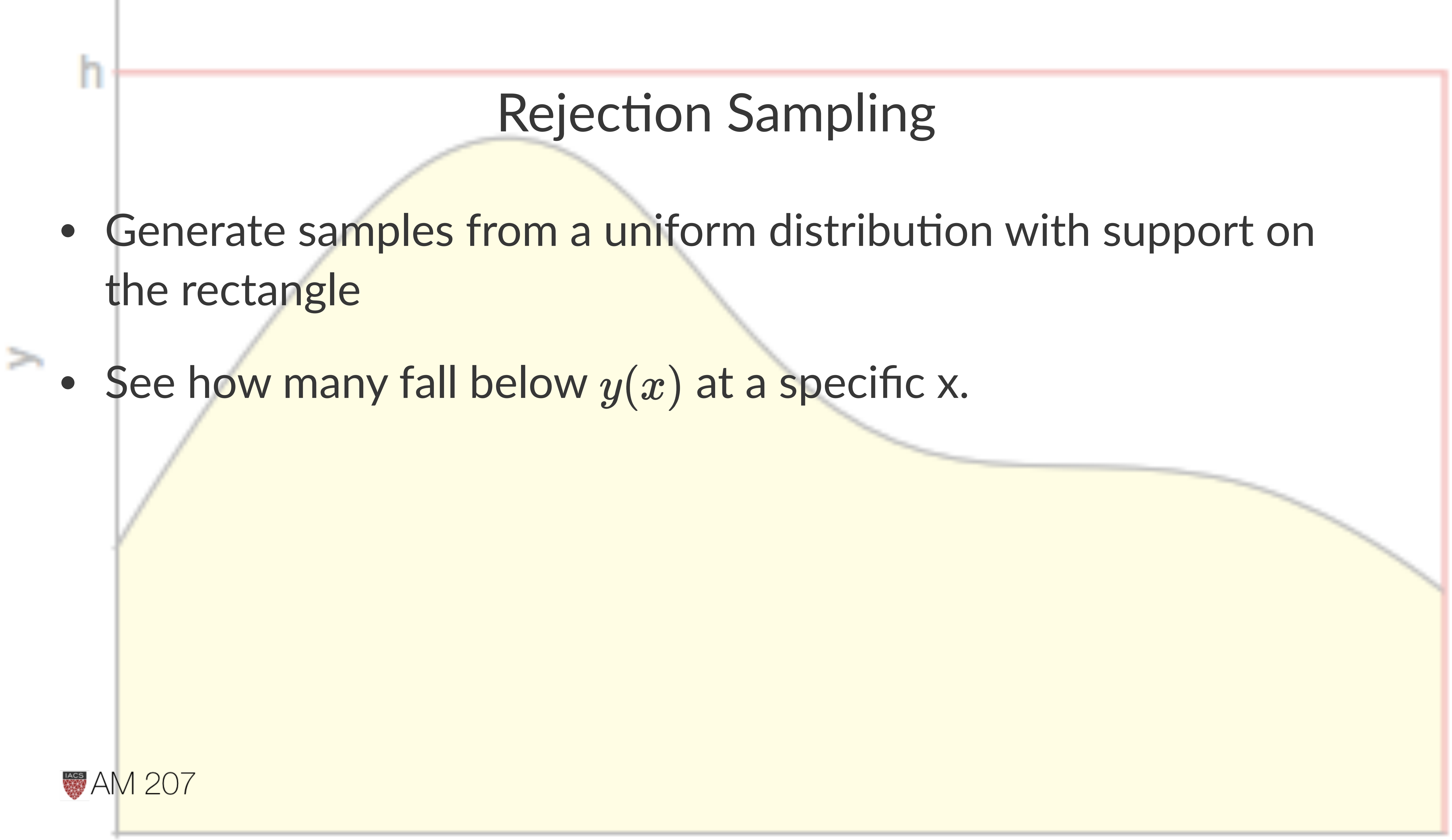
Let $g : r = \sqrt{x^2 + y^2}, tan(\theta) = y/x$. Then $g^{-1} : x = r\, cos(\theta)$, $y = r\, sin(\theta)$

$$J = \begin{pmatrix} cos(\theta) \; sin(\theta) \\ -rsin(\theta) \; rcos(\theta) \end{pmatrix}, det(J) = r$$

$$f_{R,\Theta}(r, \theta) = f_{X,Y}(rcos(\theta), rsin(\theta)) \times r$$

$$= \frac{1}{\sqrt{2\pi}} e^{-(rcos(\theta))^2/2} \times \frac{1}{\sqrt{2\pi}} e^{-(rsin(\theta))^2/2} = \frac{1}{2\pi} \times e^{-r^2/2} \times r.$$

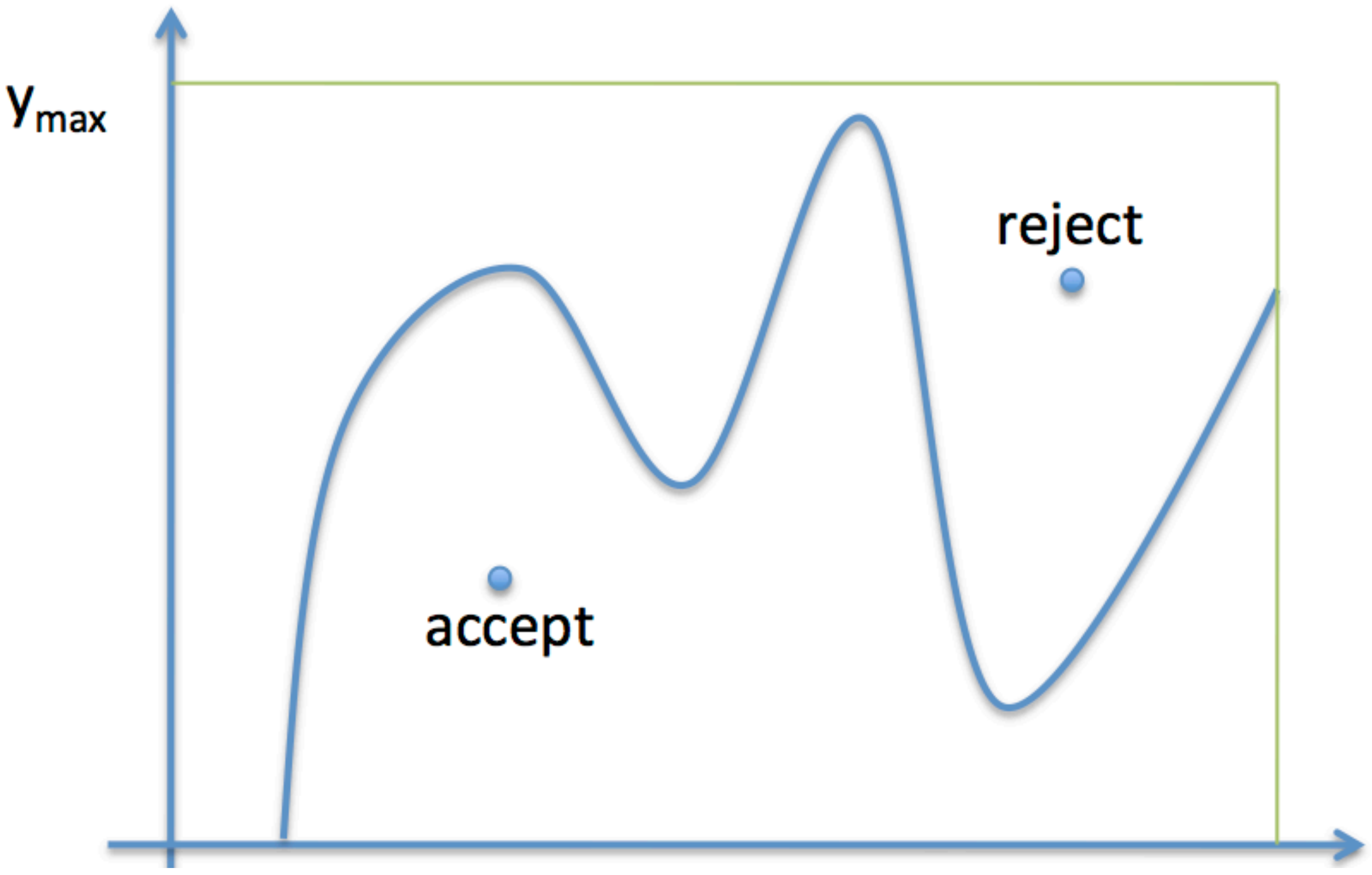
Rejection Sampling



- Generate samples from a uniform distribution with support on the rectangle
- See how many fall below $y(x)$ at a specific x.

Algorithm

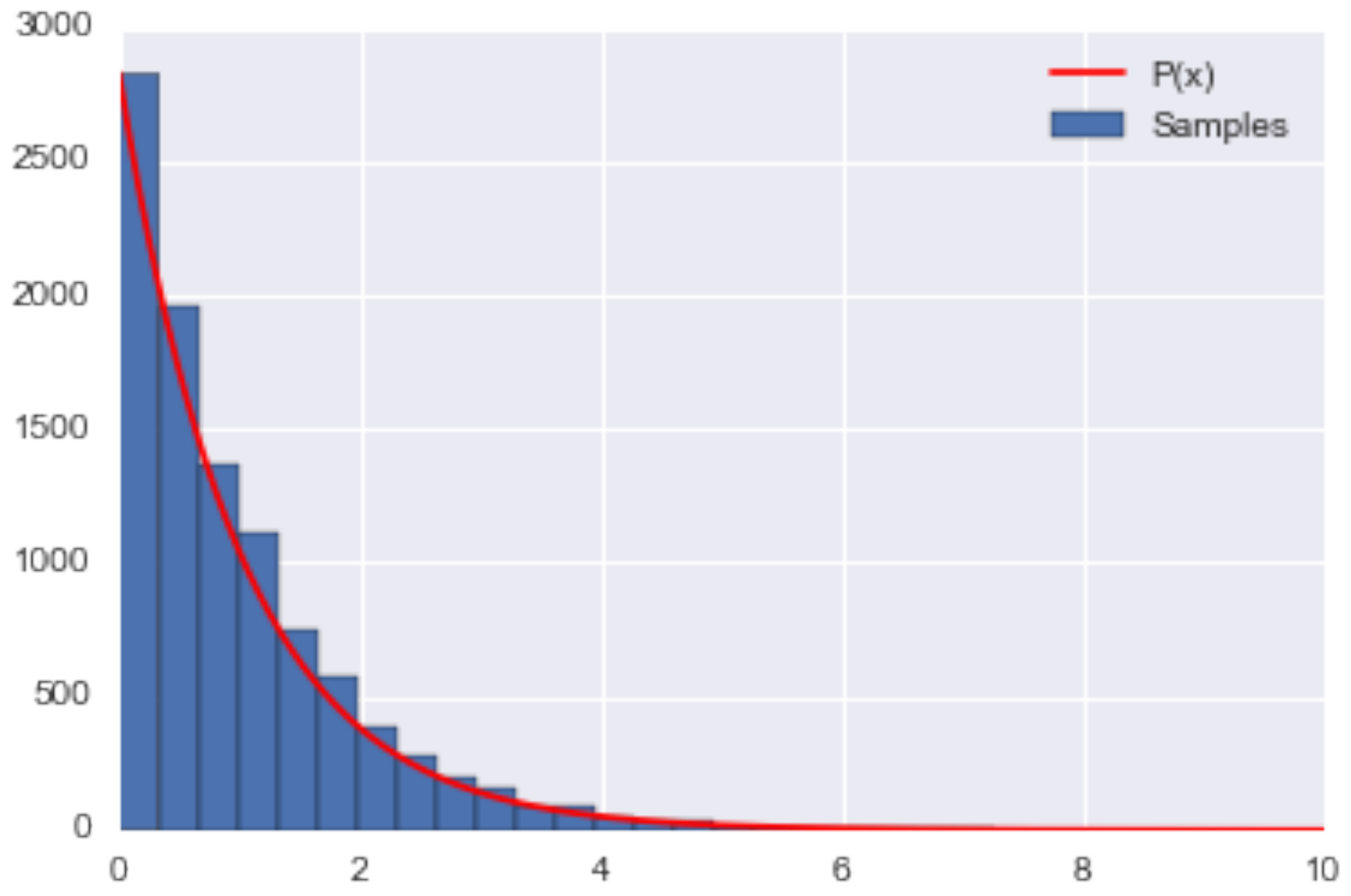
1. Draw x uniformly from $[x_{min},\, x_{max}]$
2. Draw y uniformly from $[0,\, y_{max}]$
3. if $y < f(x)$, accept the sample
4. otherwise reject it
5. repeat



example

```
P = lambda x: np.exp(-x)
xmin = 0 # the lower limit of our domain
xmax = 10 # the upper limit of our domain
ymax = 1
#you might have to do an optimization to find this.
N = 10000 # the total of samples we wish to generate
accepted = 0 # the number of accepted samples
samples = np.zeros(N)
count = 0 # the total count of proposals

while (accepted < N):
    # pick a uniform number on [xmin, xmax) (e.g. 0...10)
    x = np.random.uniform(xmin, xmax)
    # pick a uniform number on [0, ymax)
    y = np.random.uniform(0,ymax)
    # Do the accept/reject comparison
    if y < P(x):
        samples[accepted] = x
        accepted += 1

    count +=1

print("Count",count, "Accepted", accepted)
hinfo = np.histogram(samples,30)
plt.hist(samples,bins=30, label=u'Samples');
xvals=np.linspace(xmin, xmax, 1000)
plt.plot(xvals, hinfo[0][0]*P(xvals), 'r', label=u'P(x)')
plt.legend()
```

```
Count 100294 Accepted 10000
```



problems

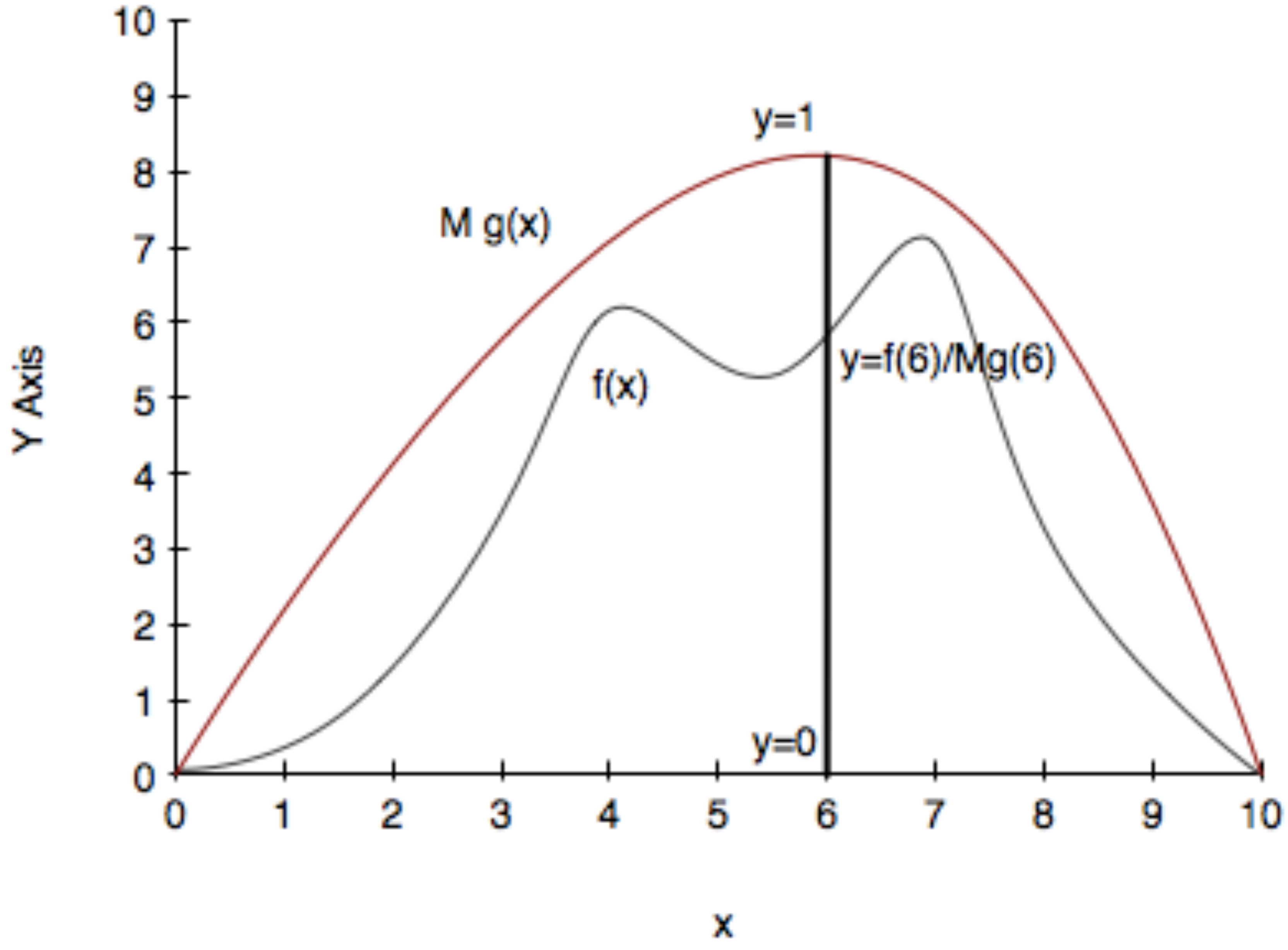
- determining the supremum may be costly
- the functional form may be complex for comparison
- even if you find a tight bound for the supremum, basic rejection sampling is very inefficient: **low acceptance probability**
- infinite support

Variance Reduction

Rejection on steroids

Introduce a **proposal density** $g(x)$.

- $g(x)$ is easy to sample from and (calculate the pdf)
- Some M exists so that $M\,g(x) > f(x)$ in your entire domain of interest
- ideally $g(x)$ will be somewhat close to f
- optimal value for M is the supremum over your domain of interest of f/g.
- probability of acceptance is $1/M$



Algorithm

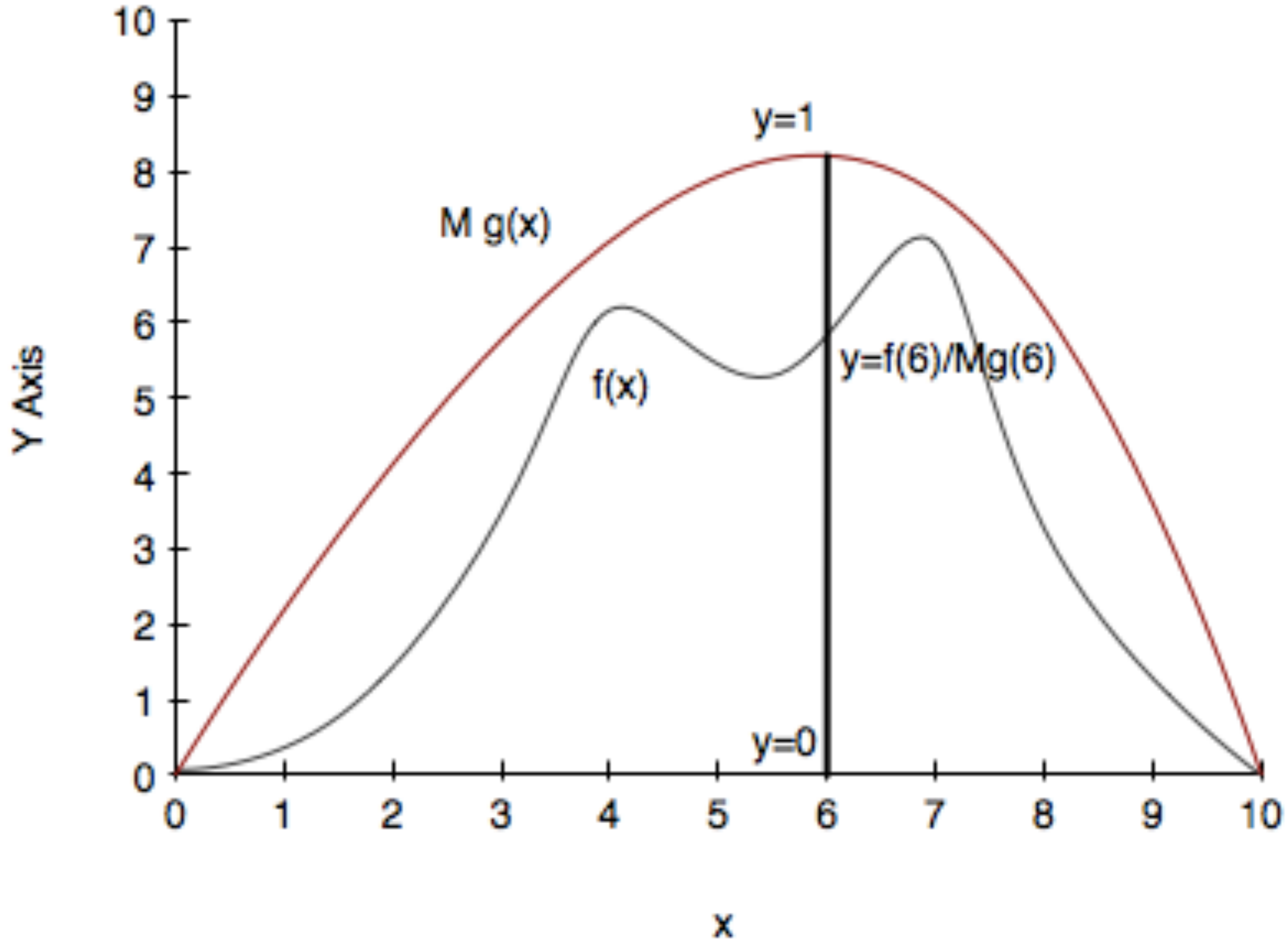
1. Draw x from your proposal distribution $g(x)$

2. Draw y uniformly from [0,1]

3. if $y < f(x)/M\, g(x)$, accept the sample

4. otherwise reject it

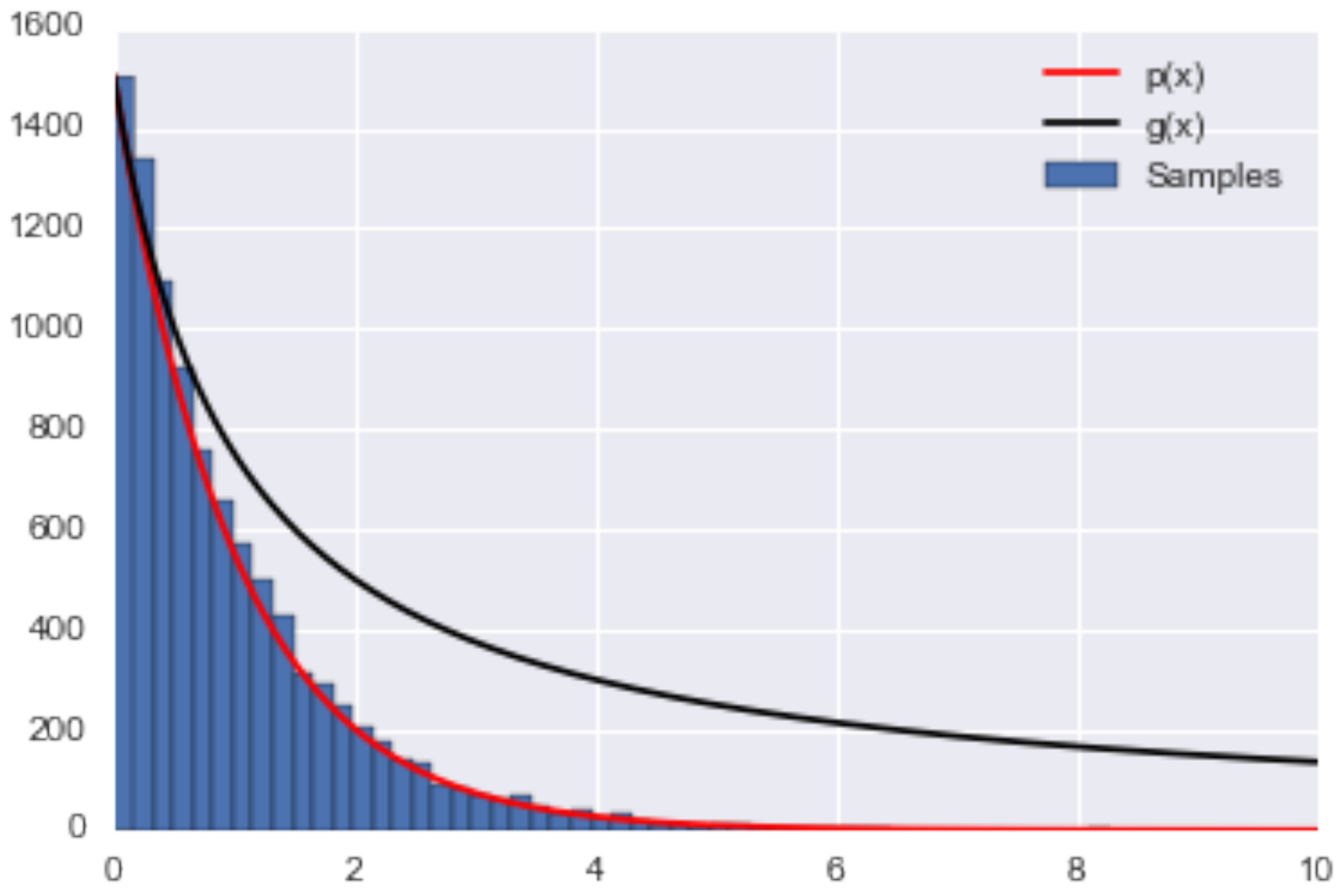
5. repeat



Example

```python
p = lambda x: np.exp(-x)  # our distribution
g = lambda x: 1/(x+1)  # our proposal pdf (we're thus choosing M to be 1)
invCDFg = lambda x: np.log(x +1) # generates our proposal using inverse sampling
xmin = 0 # the lower limit of our domain
xmax = 10 # the upper limit of our domain
# range limits for inverse sampling
umin = invCDFg(xmin)
umax = invCDFg(xmax)
N = 10000 # the total of samples we wish to generate
accepted = 0 # the number of accepted samples
samples = np.zeros(N)
count = 0 # the total count of proposals

while (accepted < N):

    # Sample from g using inverse sampling
    u = np.random.uniform(umin, umax)
    xproposal = np.exp(u) - 1

    # pick a uniform number on [0, 1)
    y = np.random.uniform(0,1)

    # Do the accept/reject comparison
    if y < p(xproposal)/g(xproposal):
        samples[accepted] = xproposal
        accepted += 1

    count +=1

print("Count", count, "Accepted", accepted)
# get the histogram info
hinfo = np.histogram(samples,50)
plt.hist(samples,bins=50, label=u'Samples');
xvals=np.linspace(xmin, xmax, 1000)
plt.plot(xvals, hinfo[0][0]*p(xvals), 'r', label=u'p(x)')
plt.plot(xvals, hinfo[0][0]*g(xvals), 'k', label=u'g(x)')
plt.legend()
```

```
Count 23809 Accepted 10000
```



Importance sampling

The basic idea behind importance sampling is that we want to draw more samples where $h(x)$, a function whose integral or expectation we desire, is large. In the case we are doing an expectation, it would indeed be even better to draw more samples where $h(x)f(x)$ is large, where $f(x)$ is the pdf we are calculating the integral with respect to.

Unlike rejection sampling we use all samples!!

$$E_f[h] = \int_V f(x)h(x)dx.$$

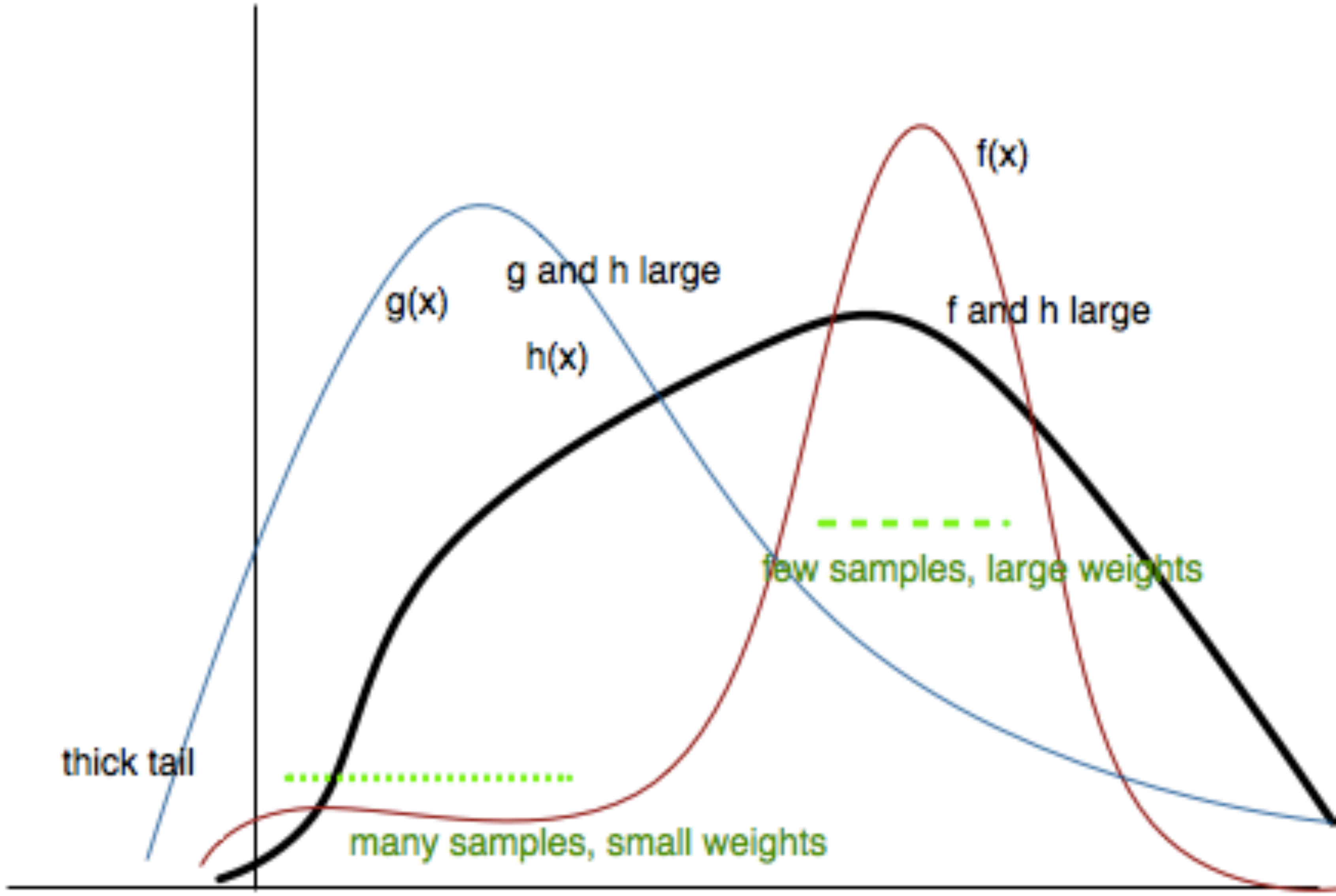
Choosing a proposal distribution $g(x)$:

$$E_f[h] = \int h(x)g(x)\frac{f(x)}{g(x)}dV$$

$$E_f[h] = \lim_{N\to\infty}\frac{1}{N}\sum_{x_i\sim g(.)} h(x_i)\frac{f(x_i)}{g(x_i)}$$

If $w(x_i) = f(x_i)/g(x_i)$:

$$E_f[h] = \lim_{N\to\infty}\frac{1}{N}\sum_{x_i\sim g(.)} w(x_i)h(x_i)$$



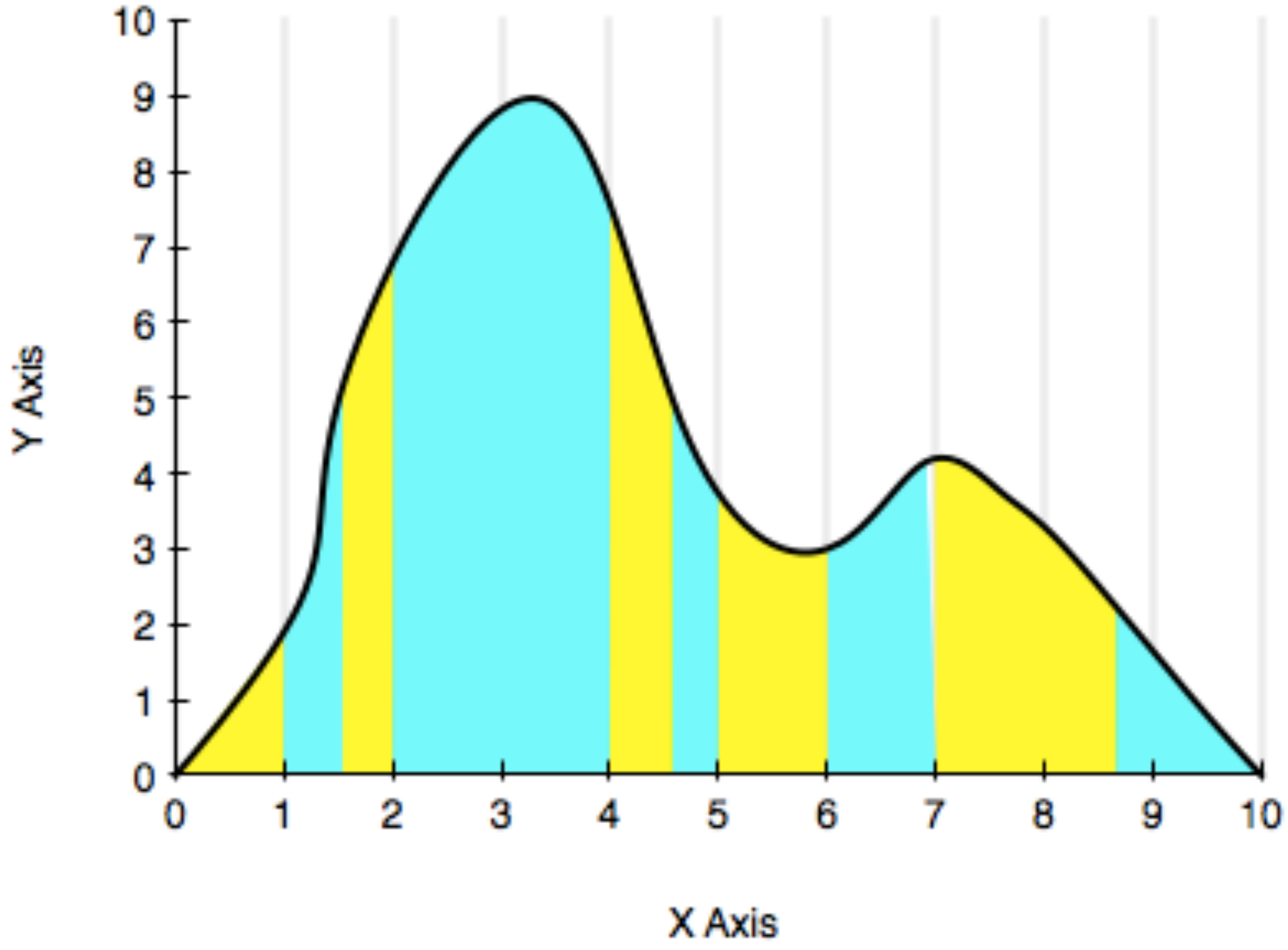
Stratified Sampling

Split the domain on which we wish to calculate an expectation or integral into strata, to minimize variance.

Intuitively, smaller samples have less variance.

Want $\mu = E_f[h] = \int_D h(x) f(x)\, dx$

$$\hat{\mu} = (1/N) \sum_{x_k \sim f} h(x_k); E_R[\hat{\mu}] = \mu.$$



Break the interval into M *strata* and take n_j samples for each strata j, such that $N = \sum_j n_j$.

$$\mu = \int_D h(x)f(x)dx = \sum_j \int_{D_j} h(x)f(x)dx$$

Say probability of being in region D_j is p_j. Then:

$$p_j = \int_{D_j} f(x)dx.$$ Thus pdf in the jth strata is: $f_j(x) = \frac{f(x)}{p_j}$.

Then

$$\mu = \sum_j p_j \int_{D_j} h(x) \frac{f(x)}{p_j} dx = \sum_j p_j \mu_j,$$

where

$$\mu_j = E_{f_j}[h] \text{ and thus MC gives } \hat{\mu_j} = \frac{1}{n_j} \sum_{x_i j \sim f_j} h(x_i j).$$

Define $\hat{\mu_s} = \sum_j p_j \hat{\mu_j}$.

Then:

$$E_R[\hat{\mu_s}] = \sum_j p_j E_R[\hat{\mu_j}] = \sum_j p_j \mu_j = \mu$$

Thus $\hat{\mu_s}$ is an unbiased estimator of μ. Yay!

What about the variance?

$$Var_R[\hat{\mu_s}] = Var_R[\sum_j p_j \hat{\mu_j}] = \sum_j p_j^2 Var_R[\hat{\mu_j}] = \sum_j p_j^2 \frac{\sigma_j^2}{n_j}$$

where $\sigma_j^2 = \int_{D_j} (h(x) - \mu_j)^2 f_j(x) dx$

is the "population variance" of $h(x)$ with respect to pdf $f_j(x)$ in region of support D_j.

$$Var_R[\hat{\mu}] = \frac{\sigma^2}{N} = \frac{1}{N}\int_D (h(x)-\mu)^2 f(x)dx$$

$$= \frac{1}{N}\sum_j p_j \int_{D_j} (h(x)-\mu)^2 f_j(x)dx = \frac{1}{N}\sum_j p_j \left(\int_{D_j} h(x)^2 f_j(x)dx + \mu^2 \int_{D_j} f_j(x)dx - 2\mu \int_{D_j} h(x) f_j(x)dx\right)$$

$$= \frac{1}{N}\left(\sum_j p_j \int_{D_j} h(x)^2 f_j(x)dx - \mu^2\right)$$

$$= \frac{1}{N}\left(\sum_j p_j [\sigma_j^2 + \mu_j^2] - \mu^2\right)$$

Remember $Var_R[\hat{\mu_s}] = \sum_j p_j^2 \frac{\sigma_j^2}{n_j}$ and assume that $n_j = p_j N$

we get:

$Var_R[\hat{\mu}] = \frac{1}{N} \sum_j p_j \sigma_j^2 + \frac{1}{N} \left(\sum_j p_j \mu_j^2 - \mu^2 \right)$ which is the stratified variance plus a quantity that can be be shown to be positive by the cauchy schwartz equality.

MLE for Logistic Regression

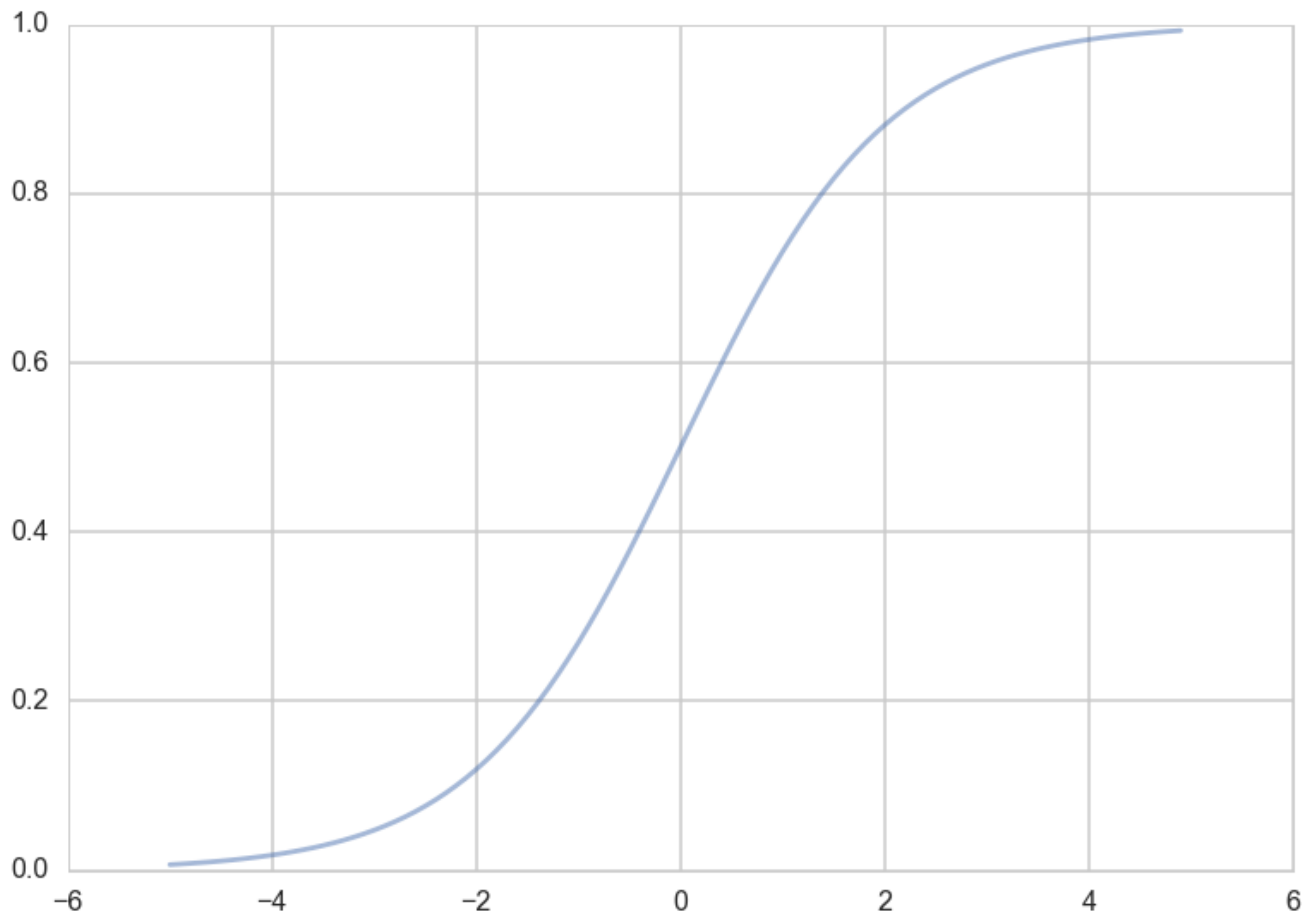
- example of a Generalized Linear Model (GLM)
- "Squeeze" linear regression through a **Sigmoid** function
- this bounds the output to be a probability
- What is the sampling Distribution?

Sigmoid function

This function is plotted below:

```
h = lambda z: 1./(1+np.exp(-z))
zs=np.arange(-5,5,0.1)
plt.plot(zs, h(zs), alpha=0.5);
```

Identify: $z = \mathbf{w} \cdot \mathbf{x}$. and $h(\mathbf{w} \cdot \mathbf{x})$ with the probability that the sample is a '1' ($y = 1$).



Then, the conditional probabilities of $y = 1$ or $y = 0$ given a particular sample's features $\mathbf{x}$ are:

$$
\begin{aligned}
P(y = 1|\mathbf{x}) &= h(\mathbf{w}\cdot\mathbf{x}) \\
P(y = 0|\mathbf{x}) &= 1 - h(\mathbf{w}\cdot\mathbf{x}).
\end{aligned}
$$

These two can be written together as

$$
P(y|\mathbf{x}, \mathbf{w}) = h(\mathbf{w}\cdot\mathbf{x})^{y}(1 - h(\mathbf{w}\cdot\mathbf{x}))^{(1-y)}
$$

BERNOULLI!!

Multiplying over the samples we get:

$$P(y|\mathbf{x},\mathbf{w}) = P(\{y_i\}|\{\mathbf{x}_i\},\mathbf{w}) = \prod_{y_i \in \mathcal{D}} P(y_i|\mathbf{x}_i,\mathbf{w}) = \prod_{y_i \in \mathcal{D}} h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i}\left(1 - h(\mathbf{w}\cdot\mathbf{x}_i)\right)^{(1-y_i)}$$

A noisy y is to imagine that our data $\mathcal{D}$ was generated from a joint probability distribution $P(x, y)$. Thus we need to model y at a given x, written as $P(y \mid x)$, and since $P(x)$ is also a probability distribution, we have:

$$P(x, y) = P(y \mid x)P(x),$$

Indeed its important to realize that a particular sample can be thought of as a draw from some "true" probability distribution.

maximum likelihood estimation maximises the **likelihood of the sample y**,

$$\mathcal{L} = P(y \mid \mathbf{x}, \mathbf{w}).$$

Again, we can equivalently maximize

$$\ell = log(P(y \mid \mathbf{x}, \mathbf{w}))$$

Thus

$$
\begin{aligned}
\ell &= log\left(\prod_{y_i \in \mathcal{D}} h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i}\,(1-h(\mathbf{w}\cdot\mathbf{x}_i))^{(1-y_i)}\right) \\
&= \sum_{y_i \in \mathcal{D}} log\left(h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i}\,(1-h(\mathbf{w}\cdot\mathbf{x}_i))^{(1-y_i)}\right) \\
&= \sum_{y_i \in \mathcal{D}} log\,h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i} + log\,(1-h(\mathbf{w}\cdot\mathbf{x}_i))^{(1-y_i)} \\
&= \sum_{y_i \in \mathcal{D}} (y_i log(h(\mathbf{w}\cdot\mathbf{x})) + (1-y_i)log(1-h(\mathbf{w}\cdot\mathbf{x})))
\end{aligned}
$$