IMPRS - BBL

Numerical methods

Lecture 4

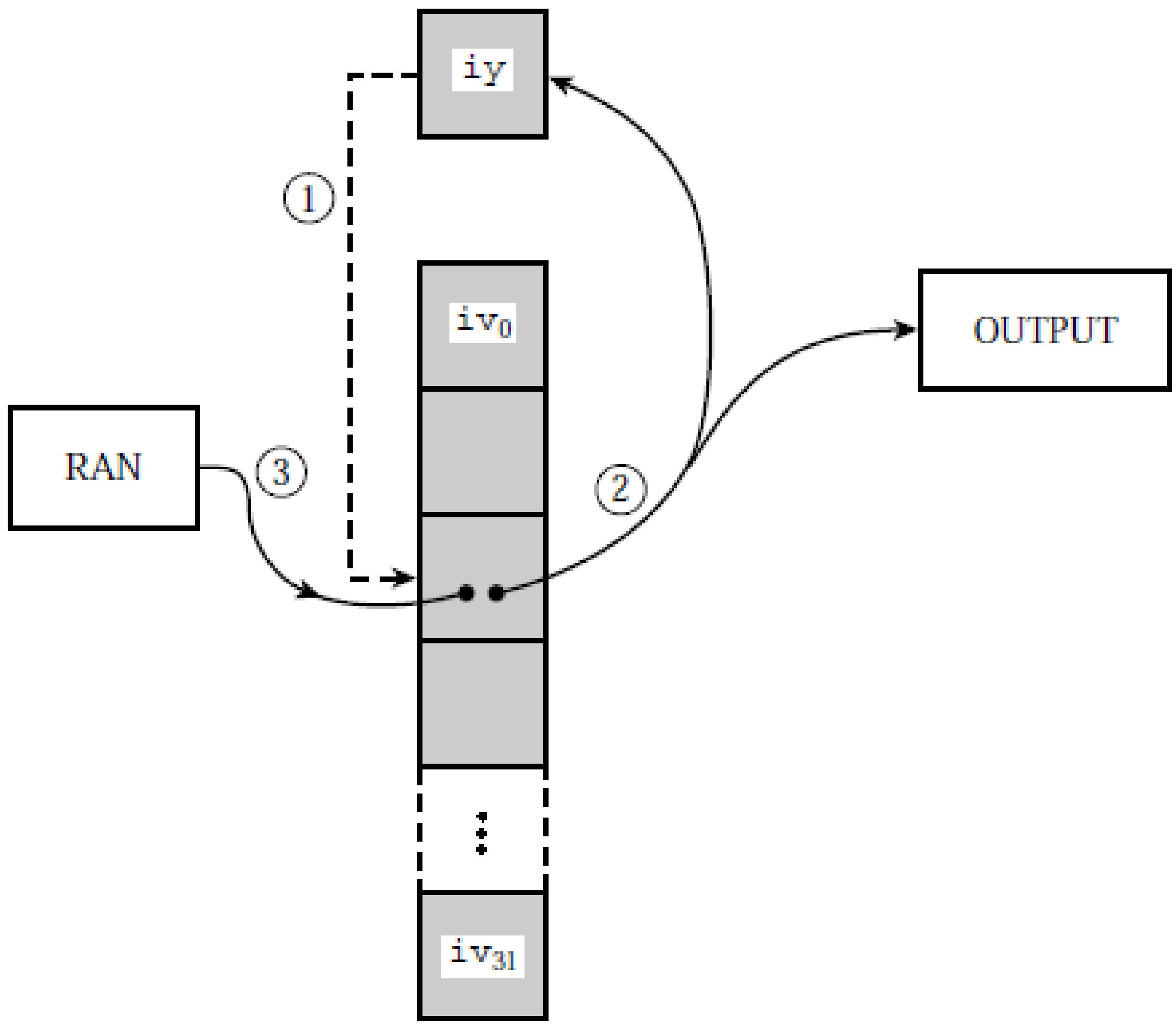
Michael Marks

Topics:

Day 1: Linear algebraic equations
Day 2: Inter- and Extrapolation
Day 3: Integration
Day 4: Random numbers and distribution functions
Day 5: Root finding, Minimization and Maximization
Day 6: Differentiation

Shuffling procedure

Break-up of sequential correlation



Numerical Recipes

Random deviates

Exponential

```
#include <math.h>

float expdev(long *idum)
```

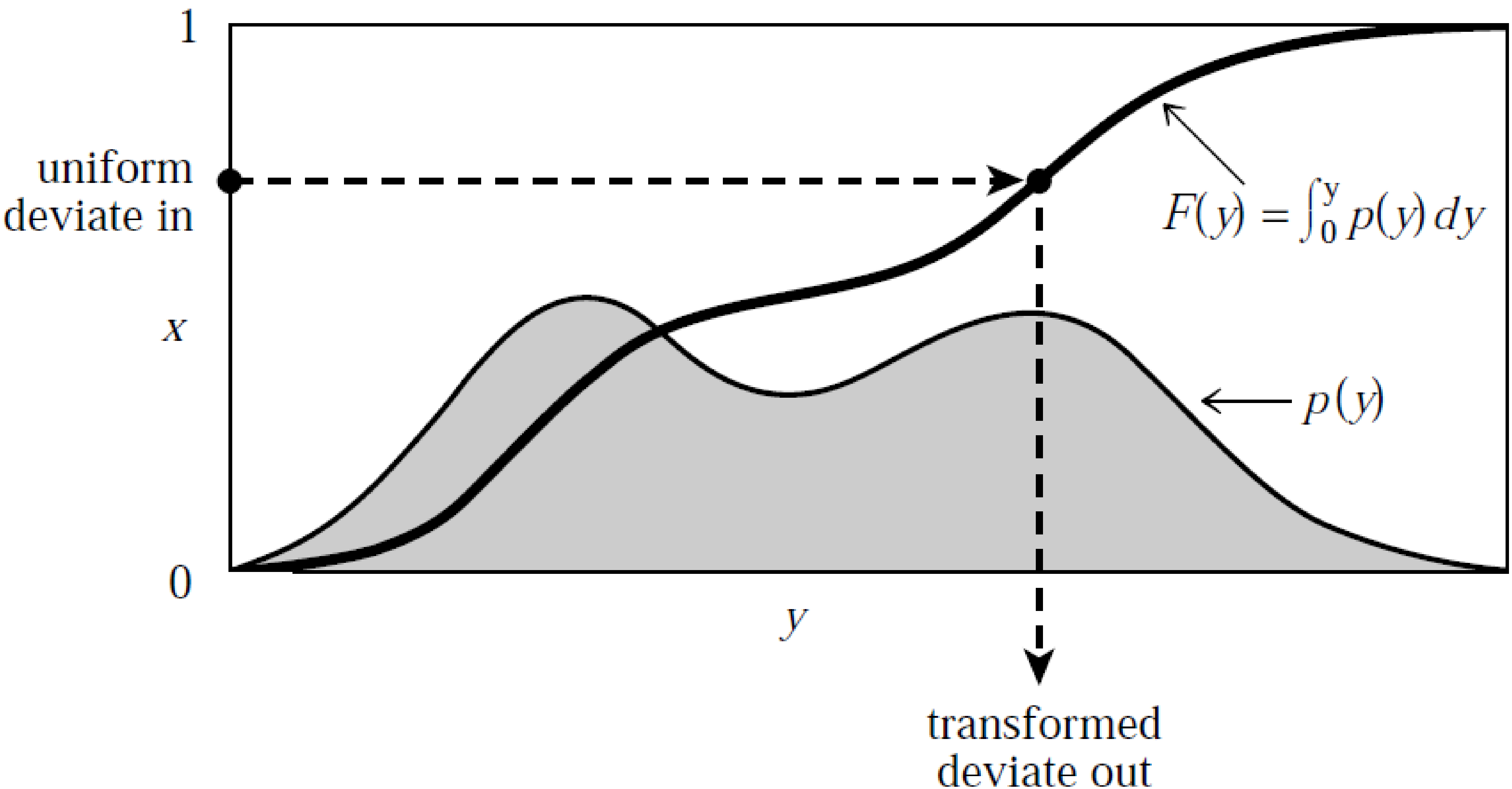
Returns an exponentially distributed, positive, random deviate of unit mean, using `ran1(idum)` as the source of uniform deviates.

```
{
    float ran1(long *idum);
    float dum;

    do
        dum=ran1(idum);
    while (dum == 0.0);
    return -log(dum);
}
```

Random number generator in Numerical Recipes

Numerical Recipes

Transformation method



Numerical Recipes

Random deviates

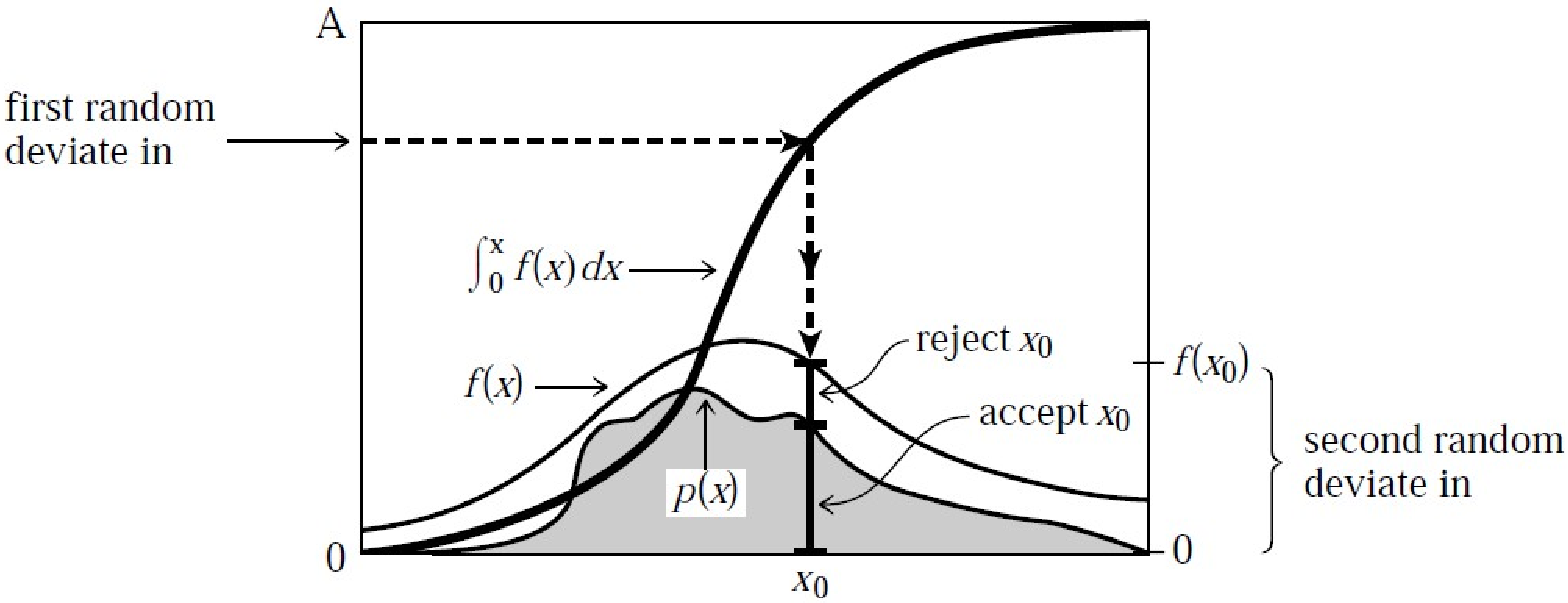
Gauss

```
#include <math.h>

float gasdev(long *idum)
Returns a normally distributed deviate with zero mean and unit variance, using ran1(idum)
as the source of uniform deviates.
{
    float ran1(long *idum);
    static int iset=0;
    static float gset;
    float fac,rsq,v1,v2;

    if (*idum < 0) iset=0;                    Reinitialize.
    if (iset == 0) {                          We don't have an extra deviate handy, so
        do {
            v1=2.0*ran1(idum)-1.0;            pick two uniform numbers in the square ex-
            v2=2.0*ran1(idum)-1.0;                tending from -1 to +1 in each direction,
            rsq=v1*v1+v2*v2;                  see if they are in the unit circle,
        } while (rsq >= 1.0 || rsq == 0.0);       and if they are not, try again.
        fac=sqrt(-2.0*log(rsq)/rsq);
        Now make the Box-Muller transformation to get two normal deviates. Return one and
        save the other for next time.
        gset=v1*fac;
        iset=1;                               Set flag.
        return v2*fac;
    } else {                                  We have an extra deviate handy,
        iset=0;                               so unset the flag,
        return gset;                          and return it.
    }
}
```

Numerical Recipes

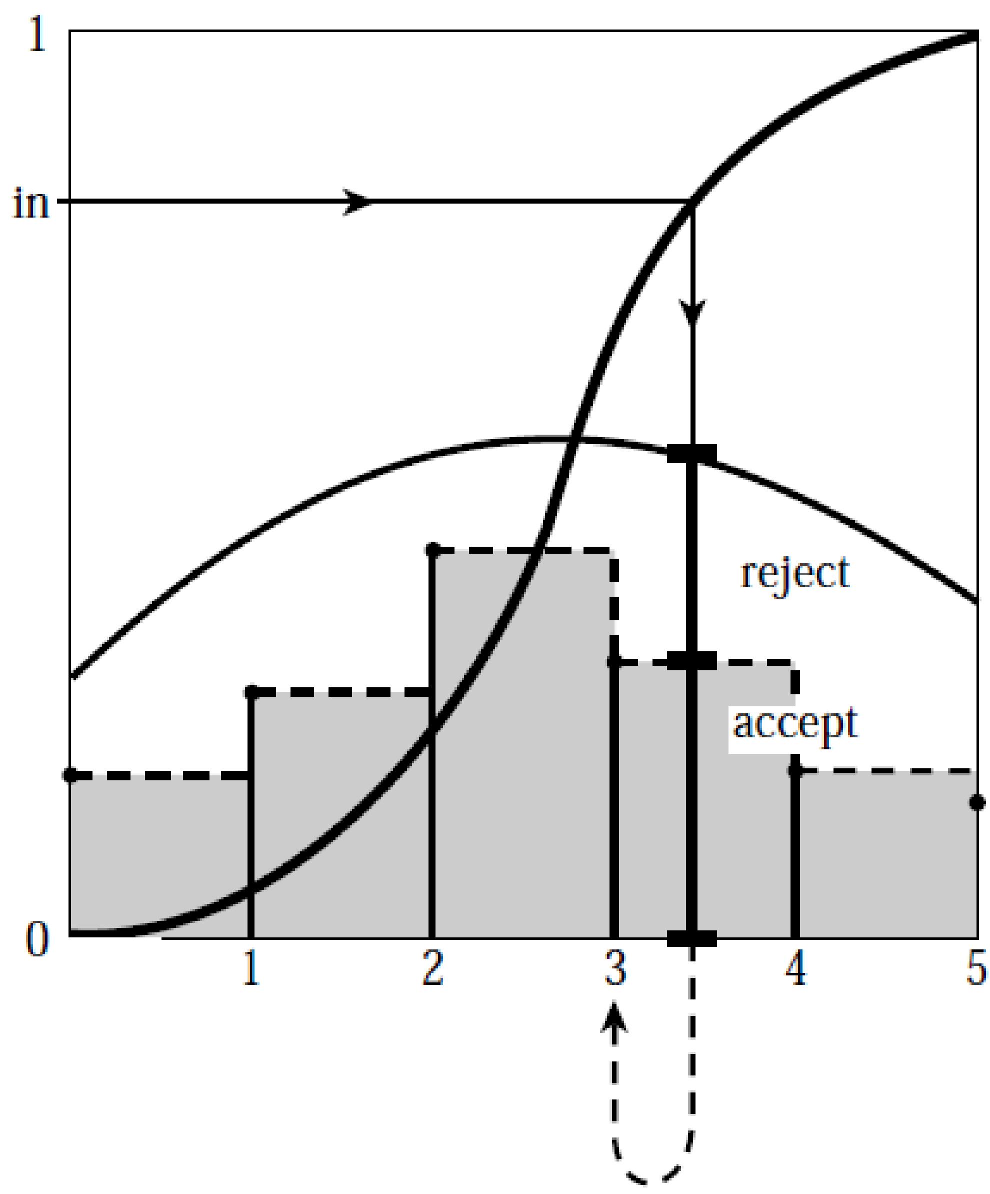
Rejection method

Continous distribution



Rejection method

binned distribution



Numerical Recipes

Exercise

Randomly draw 1000 masses m in the range [0.5:150] M_{sun} from a single power-law distributed stellar mass function of the form:

$$k \times m^{-2.35}$$

where k is a normalization constant and -2.35 is the „Salpeter value“. Check if your so drawn random numbers are indeed distributed as expected, by binning the masses into equal-size bins in a log(number) vs. m diagram.

<u>Steps:</u>

1) Calculate the normalization constant k from the requirement that its probability distribution shall be normalized to unity within the given mass-range (by hand!).
2) Find the inverse of the primitive integral to draw masses from this distribution (by hand!). Using a uniform deviate provided by the standard random number generator of your preferred programming language.

Solution

1) calculate k:

$\text{Integral}_{0.5}^{150}[k\,(m')^{-2.35}]\,dm' = 1$

$\Rightarrow k = 1.35 / (0.5^{-1.35} - 150^{-1.35})$

2) First determine, then invert the primitive integral:

$X = \text{Integral}_{0.5}^{m}[k\,(m')^{-2.35}]\,dm' = k\,[m^{-1.35} - 0.5^{-1.35}] / -1.35$

(X is uniform random deviate)

$\Rightarrow m = [\,(-1.35\,X / k) + 0.5^{-1.35}\,]^{-1.0/1.35}$

Solution

3) C code:

```
#define N 1000

int main(void) {

    float k,X;
    float m[N];

    srand(time_t(NULL)); // initialize random number generator with time as seed

    k = 1.35 / (pow(0.5,-1.35) – pow(150,-1.35)); // normalization constant

    for(i=0;i<N;i++) {
        do {
            X = rand();
            m[i] = pow((-1.35 X / k) + 0.5^-1.35),(-1.0/1.35));
        } while(m[i]<0.5 || m[i]>150);
    }

    // then do something with m

}
```