Lattice Reduction Algorithms:

EUCLID, GAUSS, LLL

Description and Probabilistic Analysis

Brigitte VALLÉE
(CNRS and Université de Caen, France)

Ecole du CIMPA en Mauritanie
Nouakchott, Février 2016

The general problem of lattice reduction

A lattice of $\mathbb{R}^p$ = a discrete additive subgroup of $\mathbb{R}^p$.

A lattice $\mathcal{L}$ possesses a basis $B := (b_1, b_2, \ldots, b_n)$ with $n \le p$,

$$\mathcal{L} := \{x \in \mathbb{R}^p; \quad x = \sum_{i=1}^{n} x_i b_i, \qquad x_i \in \mathbb{Z}\}$$

... and in fact, an infinite number of bases....

If now $\mathbb{R}^p$ is endowed with its (canonical) Euclidean structure, there exist bases (called reduced) with good Euclidean properties: their vectors are short enough and almost orthogonal.

Lattice reduction Problem : From a lattice $\mathcal{L}$ given by a basis B, construct from B a reduced basis $\hat{B}$ of $\mathcal{L}$.

Many applications of this problem in various domains:
number theory, arithmetics, discrete geometry..... and cryptology.

The general problem of lattice reduction

A lattice of $\mathbb{R}^p$ = a discrete additive subgroup of $\mathbb{R}^p$.

A lattice $\mathcal{L}$ possesses a basis $B := (b_1, b_2, \ldots, b_n)$ with $n \le p$,

$$\mathcal{L} := \{x \in \mathbb{R}^p; \quad x = \sum_{i=1}^{n} x_i b_i, \qquad x_i \in \mathbb{Z}\}$$

... and in fact, an infinite number of bases....

If now $\mathbb{R}^p$ is endowed with its (canonical) Euclidean structure,
there exist bases (called reduced) with good Euclidean properties:
their vectors are short enough and almost orthogonal.

Lattice reduction Problem : From a lattice $\mathcal{L}$ given by a basis B,
construct from B a reduced basis $\hat{B}$ of $\mathcal{L}$.

Many applications of this problem in various domains:
number theory, arithmetics, discrete geometry..... and cryptology.

The general problem of lattice reduction

A lattice of $\mathbb{R}^p$ = a discrete additive subgroup of $\mathbb{R}^p$.

A lattice $\mathcal{L}$ possesses a basis $B := (b_1, b_2, \ldots, b_n)$ with $n \le p$,

$$\mathcal{L} := \{x \in \mathbb{R}^p; \quad x = \sum_{i=1}^{n} x_i b_i, \qquad x_i \in \mathbb{Z}\}$$

... and in fact, an infinite number of bases....

If now $\mathbb{R}^p$ is endowed with its (canonical) Euclidean structure,
there exist bases (called reduced) with good Euclidean properties:
their vectors are short enough and almost orthogonal.

Lattice reduction Problem : From a lattice $\mathcal{L}$ given by a basis B,
construct from B a reduced basis $\hat{B}$ of $\mathcal{L}$.

Many applications of this problem in various domains:
number theory, arithmetics, discrete geometry..... and cryptology.

The general problem of lattice reduction

A lattice of $\mathbb{R}^p$ = a discrete additive subgroup of $\mathbb{R}^p$.

A lattice $\mathcal{L}$ possesses a basis $B := (b_1, b_2, \ldots, b_n)$ with $n \le p$,

$$\mathcal{L} := \{x \in \mathbb{R}^p; \quad x = \sum_{i=1}^{n} x_i b_i, \qquad x_i \in \mathbb{Z}\}$$

... and in fact, an infinite number of bases....

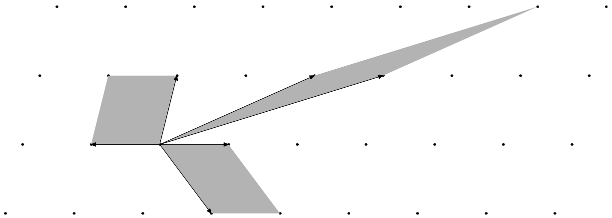
If now $\mathbb{R}^p$ is endowed with its (canonical) Euclidean structure,
there exist bases (called reduced) with good Euclidean properties:
their vectors are short enough and almost orthogonal.

Lattice reduction Problem : From a lattice $\mathcal{L}$ given by a basis B,
construct from B a reduced basis $\hat{B}$ of $\mathcal{L}$.

Many applications of this problem in various domains:

number theory, arithmetics, discrete geometry..... and cryptology.

Lattice reduction algorithms in the two dimensional case.



Three main cases,

according to the increasing dimension n of the lattice.

$n = 1$: the Euclid algorithm

computes the greatest common divisor $\gcd(u, v)$

$n = 2$: the Gauss algorithm

computes a minimal basis of a lattice of two dimensions

$n \geq 3$: the LLL algorithm

computes a reduced basis of a lattice of any dimensions.

Each algorithm can be viewed

as an extension of the previous one

Three main cases,
according to the increasing dimension n of the lattice.

$n = 1$: the Euclid algorithm
computes the greatest common divisor $\gcd(u, v)$

$n = 2$: the Gauss algorithm
computes a minimal basis of a lattice of two dimensions

$n \geq 3$: the LLL algorithm
computes a reduced basis of a lattice of any dimensions.

Each algorithm can be viewed
as an extension of the previous one

Three main cases,
according to the increasing dimension n of the lattice.

$n = 1$: the Euclid algorithm
computes the greatest common divisor $\gcd(u, v)$

$n = 2$: the Gauss algorithm
computes a minimal basis of a lattice of two dimensions

$n \geq 3$: the LLL algorithm
computes a reduced basis of a lattice of any dimensions.

Each algorithm can be viewed
as an extension of the previous one

Part I – The Euclidean Algorithms

I-1. Two main Euclid algorithms

I-2. Many variants

I-3.- Algorithmic study

I-4. Some extensions

The (classical) Euclid Algorithm: the grand father of all the algorithms.

On the input (u, v), it computes the gcd of u and v, together with the Continued Fraction Expansion of u/v. $u_0 := v;\ u_1 := u; u_0 \geq u_1 > 0$

$$\left\{\begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & u_2 & & 0 < u_2 < u_1 \\ u_1 & = & m_2 u_2 & + & u_3 & & 0 < u_3 < u_2 \\ \ldots & = & \ldots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & u_p & & 0 < u_p < u_{p-1} \\ u_{p-1} & = & m_p u_p & + & 0 & & u_{p+1} = 0 \end{array}\right\}$$

u_p is the gcd of u and v, the m_i's are the digits. p is the depth.

CFE of $\frac{u}{v}$:
$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{1}{m_2 + \cfrac{1}{\ddots + \cfrac{1}{m_p}}}},$$

The (classical) Euclid Algorithm: the grand father of all the algorithms.

On the input (u, v), it computes the gcd of u and v, together with the Continued Fraction Expansion of u/v. $u_0 := v;\ u_1 := u; u_0 \geq u_1 > 0$

$$\left\{\begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & u_2 & & 0 < u_2 < u_1 \\ u_1 & = & m_2 u_2 & + & u_3 & & 0 < u_3 < u_2 \\ \dots & = & \dots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & u_p & & 0 < u_p < u_{p-1} \\ u_{p-1} & = & m_p u_p & + & 0 & & u_{p+1} = 0 \end{array}\right\}$$

u_p is the gcd of u and v, the m_i's are the digits. p is the depth.

CFE of $\frac{u}{v}$: $$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{1}{m_2 + \cfrac{1}{\ddots + \cfrac{1}{m_p}}}},$$

The (classical) Euclid Algorithm: the grand father of all the algorithms.

On the input (u, v), it computes the gcd of u and v, together with the Continued Fraction Expansion of u/v. $u_0 := v;\ u_1 := u; u_0 \geq u_1 > 0$

$$\left\{\begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & u_2 & & 0 < u_2 < u_1 \\ u_1 & = & m_2 u_2 & + & u_3 & & 0 < u_3 < u_2 \\ \ldots & = & \ldots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & u_p & & 0 < u_p < u_{p-1} \\ u_{p-1} & = & m_p u_p & + & 0 & & u_{p+1} = 0 \end{array}\right\}$$

u_p is the gcd of u and v, the m_i's are the digits. p is the depth.

CFE of $\frac{u}{v}$:

$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{1}{m_2 + \cfrac{1}{\ddots + \cfrac{1}{m_p}}}},$$

Three main outputs for any Euclidean Algorithm

– the $\gcd(u, v)$ itself

Essential in exact rational computations,

for keeping rational numbers under their irreducible forms

60% of the computation time in some symbolic computations

– the Continued Fraction Expansion CFE (u/v)

often used directly in computation over rationals.

– For its extended version (with computation of Bezout coefficients)

– the modular inverse $u^{-1} \mod v$, when $\gcd(u, v) = 1$.

or more generally

– the algorithmic version of the Chinese Remainder Theorem

A basic algorithm ... Perhaps the fifth main operation?

With many variants....

Extensively used in cryptography

Three main outputs for any Euclidean Algorithm

– the $\gcd(u, v)$ itself

Essential in exact rational computations,

for keeping rational numbers under their irreducible forms

60% of the computation time in some symbolic computations

– the Continued Fraction Expansion CFE (u/v)

often used directly in computation over rationals.

– For its extended version (with computation of Bezout coefficients)

– the modular inverse $u^{-1} \mod v$, when $\gcd(u, v) = 1$.

or more generally

– the algorithmic version of the Chinese Remainder Theorem

A basic algorithm ... Perhaps the fifth main operation?

With many variants....

Extensively used in cryptography

Three main outputs for any Euclidean Algorithm

– the $\gcd(u, v)$ itself

Essential in exact rational computations,

for keeping rational numbers under their irreducible forms

60% of the computation time in some symbolic computations

– the Continued Fraction Expansion CFE (u/v)

often used directly in computation over rationals.

– For its extended version (with computation of Bezout coefficients)

– the modular inverse $u^{-1} \mod v$, when $\gcd(u, v) = 1$.

or more generally

– the algorithmic version of the Chinese Remainder Theorem

A basic algorithm ... Perhaps the fifth main operation?

With many variants....

Extensively used in cryptography

An important variant : The centered Euclid Algorithm.

On the input (u, v), with the Centered division,

$$v = mu + \epsilon r, \quad \epsilon = \pm 1, \quad 0 \le r \le u/2$$

it computes gcd (u, v),

together with the Centered Continued Fraction Expansion of u/v.

if $v \ge 2u$, then $u_0 := v;\ u_1 := u$

$$\left\{\begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & \epsilon_1\, u_2 & 0 < u_2 \le u_1/2, & \epsilon_1 = \pm 1 \\ u_1 & = & m_2 u_2 & + & \epsilon_2\, u_3 & 0 < u_3 \le u_2/2, & \epsilon_2 = \pm 1 \\ \ldots & = & \ldots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & \epsilon_{p-1}\, u_p & 0 < u_p \le u_{p-1}/2, & \epsilon_{p-1} = \pm 1 \\ u_{p-1} & = & m_p u_p & + & 0 & u_{p+1} = 0 & \end{array}\right\}$$

u_p is the gcd of u and v, the (m_i, ϵ_i) are the digits. p is the depth.

C-CFE of $\frac{u}{v}$: $$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{\epsilon_1}{m_2 + \cfrac{\epsilon_2}{\ddots + \cfrac{\epsilon_{p-1}}{m_p}}}},$$

An important variant : The centered Euclid Algorithm.

On the input (u, v), with the Centered division,

$$v = mu + \epsilon r, \quad \epsilon = \pm 1, \quad 0 \le r \le u/2$$

it computes gcd (u, v),

together with the Centered Continued Fraction Expansion of u/v.

if $v \ge 2u$, then $u_0 := v;\ u_1 := u$

$$\left\{ \begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & \epsilon_1\, u_2 & 0 < u_2 \le u_1/2, & \epsilon_1 = \pm 1 \\ u_1 & = & m_2 u_2 & + & \epsilon_2\, u_3 & 0 < u_3 \le u_2/2, & \epsilon_2 = \pm 1 \\ \ldots & = & \ldots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & \epsilon_{p-1}\, u_p & 0 < u_p \le u_{p-1}/2, & \epsilon_{p-1} = \pm 1 \\ u_{p-1} & = & m_p u_p & + & 0 & u_{p+1} = 0 & \end{array} \right\}$$

u_p is the gcd of u and v, the (m_i, ϵ_i) are the digits. p is the depth.

C-CFE of $\frac{u}{v}$:
$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{\epsilon_1}{m_2 + \cfrac{\epsilon_2}{\ddots + \cfrac{\epsilon_{p-1}}{m_p}}}},$$

An important variant : The centered Euclid Algorithm.

On the input (u, v), with the Centered division,

$$v = mu + \epsilon r, \quad \epsilon = \pm 1, \quad 0 \le r \le u/2$$

it computes gcd (u, v),

together with the Centered Continued Fraction Expansion of u/v.

if $v \ge 2u$, then $u_0 := v;\ u_1 := u$

$$\left\{\begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & \epsilon_1\, u_2 & 0 < u_2 \le u_1/2, & \epsilon_1 = \pm 1 \\ u_1 & = & m_2 u_2 & + & \epsilon_2\, u_3 & 0 < u_3 \le u_2/2, & \epsilon_2 = \pm 1 \\ \ldots & = & \ldots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & \epsilon_{p-1}\, u_p & 0 < u_p \le u_{p-1}/2, & \epsilon_{p-1} = \pm 1 \\ u_{p-1} & = & m_p u_p & + & 0 & u_{p+1} = 0 & \end{array}\right\}$$

u_p is the gcd of u and v, the (m_i, ϵ_i) are the digits. p is the depth.

C-CFE of $\frac{u}{v}$: $$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{\epsilon_1}{m_2 + \cfrac{\epsilon_2}{\ddots + \cfrac{\epsilon_{p-1}}{m_p}}}},$$

Underlying dynamical systems for the Euclid algorithm

pause

To better understand the algorithms,

a good idea to study the map which "extends" the division

– the properties of the iterations of the map

– and thus the underlying dynamical system

Input.- A discrete algorithm.

Step 1.- Extend the discrete algorithm into a continuous process,

i.e. a dynamical system. (X, V) X compact, $V : X \to X$,

where the discrete alg. gives rise to particular trajectories.

Step 2.- Study this dynamical system, via its generic trajectories.

Step 3.- Coming back to the algorithm: we need proving that

"the discrete trajectories behaves like the generic trajectories".

Output.- Analysis of the Algorithm.

Underlying dynamical systems for the Euclid algorithm

pause

To better understand the algorithms,

a good idea to study the map which "extends" the division

– the properties of the iterations of the map

– and thus the underlying dynamical system

Input.- A discrete algorithm.

Step 1.- Extend the discrete algorithm into a continuous process,

i.e. a dynamical system. (X, V) X compact, $V : X \to X$,

where the discrete alg. gives rise to particular trajectories.

Step 2.- Study this dynamical system, via its generic trajectories.

Step 3.- Coming back to the algorithm: we need proving that

"the discrete trajectories behaves like the generic trajectories".

Output.- Analysis of the Algorithm.

Underlying dynamical systems for the Euclid algorithm

pause

To better understand the algorithms,

a good idea to study the map which "extends" the division

– the properties of the iterations of the map

– and thus the underlying dynamical system

Input.- A discrete algorithm.

Step 1.- Extend the discrete algorithm into a continuous process,

i.e. a dynamical system. (X, V) X compact, $V : X \to X$,

where the discrete alg. gives rise to particular trajectories.

Step 2.- Study this dynamical system, via its generic trajectories.

Step 3.- Coming back to the algorithm: we need proving that

"the discrete trajectories behaves like the generic trajectories".

Output.- Analysis of the Algorithm.

Underlying dynamical systems for the Euclid algorithm

pause

To better understand the algorithms,

a good idea to study the map which "extends" the division

– the properties of the iterations of the map

– and thus the underlying dynamical system

Input.- A discrete algorithm.

Step 1.- Extend the discrete algorithm into a continuous process,

i.e. a dynamical system. (X, V) X compact, $V : X \to X$,

where the discrete alg. gives rise to particular trajectories.

Step 2.- Study this dynamical system, via its generic trajectories.

Step 3.- Coming back to the algorithm: we need proving that

"the discrete trajectories behaves like the generic trajectories".

Output.- Analysis of the Algorithm.

For instance : The `C-Euclid` dynamical system (I).

The trace of the execution of the Euclid Algorithm on (u_1, u_0) is:

$$(u_1, u_0) \to (u_2, u_1) \to (u_3, u_2) \to \ldots \to (u_{p-1}, u_p) \to (u_{p+1}, u_p) = (0, u_p)$$

Replace the integer pair (u_i, u_{i-1}) by the rational $x_i := \dfrac{u_i}{u_{i-1}}$.

The division $u_{i-1} = m_i u_i + \epsilon_i\, u_{i+1}$ is then written as

$$x_{i+1} = \epsilon\left(\frac{1}{x_i}\right)\left(\frac{1}{x_i} - \left\lfloor \frac{1}{x_i} \right\rceil\right) \qquad \text{with} \quad \epsilon(x) := \mathrm{sign}(x - \lfloor x \rceil),$$

or $\quad x_{i+1} = U(x_i), \qquad$ with

$$U(x) = \epsilon\left(\frac{1}{x}\right)\left(\frac{1}{x}\right)\left(\frac{1}{x} - \left\lfloor \frac{1}{x} \right\rceil\right) \quad \text{for} \quad x \neq 0, \quad U(0) = 0$$

An execution of the Euclidean Algorithm $(x, U(x), U^2(x), \ldots, 0)$
$=$ A rational trajectory of the Dynamical System $([0,1], U)$
$=$ a trajectory that reaches 0.

For instance : The `C-Euclid` dynamical system (I).

The trace of the execution of the Euclid Algorithm on (u_1, u_0) is:

$$(u_1, u_0) \to (u_2, u_1) \to (u_3, u_2) \to \ldots \to (u_{p-1}, u_p) \to (u_{p+1}, u_p) = (0, u_p)$$

Replace the integer pair (u_i, u_{i-1}) by the rational $x_i := \dfrac{u_i}{u_{i-1}}$.

The division $u_{i-1} = m_i u_i + \epsilon_i\, u_{i+1}$ is then written as

$$x_{i+1} = \epsilon\left(\frac{1}{x_i}\right)\left(\frac{1}{x_i} - \left\lfloor \frac{1}{x_i} \right\rceil\right) \qquad \text{with} \quad \epsilon(x) := \operatorname{sign}(x - \lfloor x \rceil),$$

$$\text{or} \quad x_{i+1} = U(x_i), \qquad \text{with}$$

$$U(x) = \epsilon\left(\frac{1}{x}\right)\left(\frac{1}{x}\right)\left(\frac{1}{x} - \left\lfloor \frac{1}{x} \right\rceil\right) \quad \text{for} \ \ x \neq 0, \quad U(0) = 0$$

An execution of the Euclidean Algorithm $(x, U(x), U^2(x), \ldots, 0)$

$=$ A rational trajectory of the Dynamical System $([0,1], U)$

$=$ a trajectory that reaches 0.

For instance : The `C-Euclid` dynamical system (I).

The trace of the execution of the Euclid Algorithm on (u_1, u_0) is:

$$(u_1, u_0) \to (u_2, u_1) \to (u_3, u_2) \to \ldots \to (u_{p-1}, u_p) \to (u_{p+1}, u_p) = (0, u_p)$$

Replace the integer pair (u_i, u_{i-1}) by the rational $x_i := \dfrac{u_i}{u_{i-1}}$.

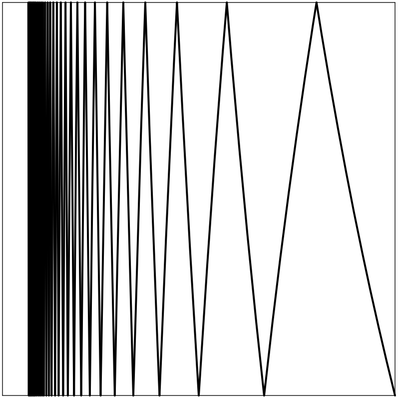
The division $u_{i-1} = m_i u_i + \epsilon_i \, u_{i+1}$ is then written as

$$x_{i+1} = \epsilon\left(\frac{1}{x_i}\right)\left(\frac{1}{x_i} - \left\lfloor \frac{1}{x_i} \right\rceil\right) \qquad \text{with} \quad \epsilon(x) := \mathrm{sign}(x - \lfloor x \rceil),$$

$$\text{or} \quad x_{i+1} = U(x_i), \qquad \text{with}$$

$$U(x) = \epsilon\left(\frac{1}{x}\right)\left(\frac{1}{x}\right)\left(\frac{1}{x} - \left\lfloor \frac{1}{x} \right\rceil\right) \quad \text{for} \;\; x \neq 0, \quad U(0) = 0$$

An execution of the Euclidean Algorithm $(x, U(x), U^2(x), \ldots, 0)$
$=$ A rational trajectory of the Dynamical System $([0,1], U)$
$=$ a trajectory that reaches 0.



The Euclidean dynamical system (II).

A dynamical system with a denumerable system of branches $(U_{[m,\epsilon]})_{(m,\epsilon)\geq(2,1)}$,

$$U_{[m,\epsilon]} : \left]\frac{1}{m}, \frac{2}{2m+\epsilon}\right[\longrightarrow]0,1[, \qquad U_{[(m,\epsilon)]}(x) := \epsilon\left(\frac{1}{x} - m\right)$$

The set $\mathcal{H}$ of the inverse branches of U is

$$\mathcal{H} := \left\{ h_{[m,\epsilon]} : \left]0, \frac{1}{2}\right[\longrightarrow \left]\frac{1}{m}, \frac{2}{2m+\epsilon}\right[; \qquad h_{[m,\epsilon]}(x) := \frac{1}{m+\epsilon x}\right\}$$

The set $\mathcal{H}$ builds one step of the CF's.

The set $\mathcal{H}^n$ of the inverse branches of U^n builds CF's of depth n.

The set $\mathcal{H}^\star := \bigcup \mathcal{H}^n$ builds all the (finite) CF's.

$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{\epsilon_1}{m_2 + \cfrac{\epsilon_2}{\ddots + \cfrac{\epsilon_{p-1}}{m_p}}}} = h_{[m_1,\epsilon_1]} \circ h_{[m_2,\epsilon_2]} \circ \ldots \circ h_{[m_p]}(0)$$

The Euclidean dynamical system (II).

A dynamical system with a denumerable system of branches $(U_{[m,\epsilon]})_{(m,\epsilon)\geq(2,1)}$,

$$U_{[m,\epsilon]} : \left]\frac{1}{m}, \frac{2}{2m+\epsilon}\right[\longrightarrow]0,1[, \qquad U_{[(m,\epsilon)]}(x) := \epsilon\left(\frac{1}{x} - m\right)$$

The set $\mathcal{H}$ of the inverse branches of U is

$$\mathcal{H} := \left\{ h_{[m,\epsilon]} : \left]0, \frac{1}{2}\right[\longrightarrow \left]\frac{1}{m}, \frac{2}{2m+\epsilon}\right[; \qquad h_{[m,\epsilon]}(x) := \frac{1}{m+\epsilon x}\right\}$$

The set $\mathcal{H}$ builds one step of the CF's.

The set $\mathcal{H}^n$ of the inverse branches of U^n builds CF's of depth n.

The set $\mathcal{H}^\star := \bigcup \mathcal{H}^n$ builds all the (finite) CF's.

$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{\epsilon_1}{m_2 + \cfrac{\epsilon_2}{\ddots + \cfrac{\epsilon_{p-1}}{m_p}}}} = h_{[m_1,\epsilon_1]} \circ h_{[m_2,\epsilon_2]} \circ \ldots \circ h_{[m_p]}(0)$$

The Euclidean dynamical system (II).

A dynamical system with a denumerable system of branches $(U_{[m,\epsilon]})_{(m,\epsilon)\geq(2,1)}$,

$$U_{[m,\epsilon]} : \left]\frac{1}{m}, \frac{2}{2m+\epsilon}\right[\longrightarrow]0,1[, \qquad U_{[(m,\epsilon)]}(x) := \epsilon\left(\frac{1}{x} - m\right)$$

The set $\mathcal{H}$ of the inverse branches of U is

$$\mathcal{H} := \left\{ h_{[m,\epsilon]} : \left]0, \frac{1}{2}\right[\longrightarrow \left]\frac{1}{m}, \frac{2}{2m+\epsilon}\right[; \qquad h_{[m,\epsilon]}(x) := \frac{1}{m+\epsilon x} \right\}$$

The set $\mathcal{H}$ builds one step of the CF's.

The set $\mathcal{H}^n$ of the inverse branches of U^n builds CF's of depth n.

The set $\mathcal{H}^\star := \bigcup \mathcal{H}^n$ builds all the (finite) CF's.

$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{\epsilon_1}{m_2 + \cfrac{\epsilon_2}{\ddots + \cfrac{\epsilon_{p-1}}{m_p}}}} = h_{[m_1,\epsilon_1]} \circ h_{[m_2,\epsilon_2]} \circ \ldots \circ h_{[m_p]}(0)$$

Part I – The Euclidean Algorithms

I-1. Two main Euclid algorithms

I-2. Many variants

I-3.- Algorithmic study

I-4. Some extensions

Many variants of the Euclid Algorithm.

A Euclidean algorithm:=

any algorithm which performs a sequence of divisions $v = mu + \epsilon 2^k r$.

There are various possible types of Euclidean divisions

– MSB divisions [directed by the Most Significant Bits]

shorten integers on the left,

and provide a remainder r smaller than u,

(w.r.t the usual absolute value), i.e. with more zeroes on the left.

– LSB divisions [directed by the Least Significant Bits]

shorten integers on the right,

and provide a remainder r smaller than u

(w.r.t the dyadic absolute value), i.e. with more zeroes on the right.

– Mixed divisions

shorten integers both on the right and on the left,

with new zeroes both on the right and on the left.

Many variants of the Euclid Algorithm.

A Euclidean algorithm:=

any algorithm which performs a sequence of divisions $v = mu + \epsilon 2^k r$.

There are various possible types of Euclidean divisions

– MSB divisions [directed by the Most Significant Bits]
shorten integers on the left,
and provide a remainder r smaller than u,
(w.r.t the usual absolute value), i.e. with more zeroes on the left.

– LSB divisions [directed by the Least Significant Bits]
shorten integers on the right,
and provide a remainder r smaller than u
(w.r.t the dyadic absolute value), i.e. with more zeroes on the right.

– Mixed divisions
shorten integers both on the right and on the left,
with new zeroes both on the right and on the left.

Many variants of the Euclid Algorithm.

A Euclidean algorithm:=

any algorithm which performs a sequence of divisions $v = mu + \epsilon 2^k r$.

There are various possible types of Euclidean divisions

– MSB divisions [directed by the Most Significant Bits]

shorten integers on the left,

and provide a remainder r smaller than u,

(w.r.t the usual absolute value), i.e. with more zeroes on the left.

– LSB divisions [directed by the Least Significant Bits]

shorten integers on the right,

and provide a remainder r smaller than u

(w.r.t the dyadic absolute value), i.e. with more zeroes on the right.

– Mixed divisions

shorten integers both on the right and on the left,

with new zeroes both on the right and on the left.

Many variants of the Euclid Algorithm.

A Euclidean algorithm:=

any algorithm which performs a sequence of divisions $v = mu + \epsilon 2^k r$.

There are various possible types of Euclidean divisions

– MSB divisions [directed by the Most Significant Bits]

shorten integers on the left,

and provide a remainder r smaller than u,

(w.r.t the usual absolute value), i.e. with more zeroes on the left.

– LSB divisions [directed by the Least Significant Bits]

shorten integers on the right,

and provide a remainder r smaller than u

(w.r.t the dyadic absolute value), i.e. with more zeroes on the right.

– Mixed divisions

shorten integers both on the right and on the left,

with new zeroes both on the right and on the left.

Instances of MSB Algorithms.

– Variants according to the position of remainder r,

By Default:	$v = mu + r$	with	$0 \le r < u$
By Excess:	$v = mu - r$	with	$0 \le r < u$
Centered:	$v = mu + \epsilon r$	with	$\epsilon = \pm 1, \;\; 0 \le r \le u/2$

– Subtractive Algorithm :

A division with quotient m can be replaced by m subtractions

`While` $v \ge u$ do $v := v - u$

Instances of MSB Algorithms.

– Variants according to the position of remainder r,

By Default: $v = mu + r$ with $0 \le r < u$

By Excess: $v = mu - r$ with $0 \le r < u$

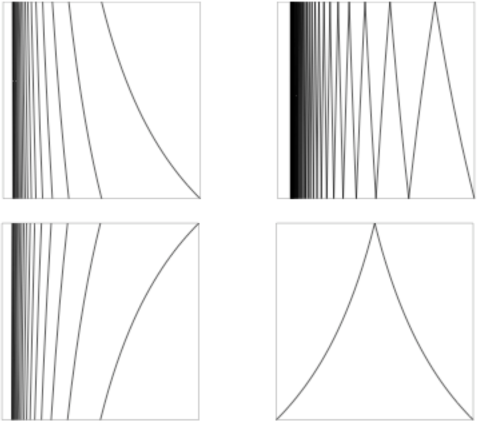
Centered: $v = mu + \epsilon r$ with $\epsilon = \pm 1, \ \ 0 \le r \le u/2$

– Subtractive Algorithm :

A division with quotient m can be replaced by m subtractions

`While` $v \ge u$ `do` $v := v - u$

Four Euclidean dynamical systems (related to MSB divisions)

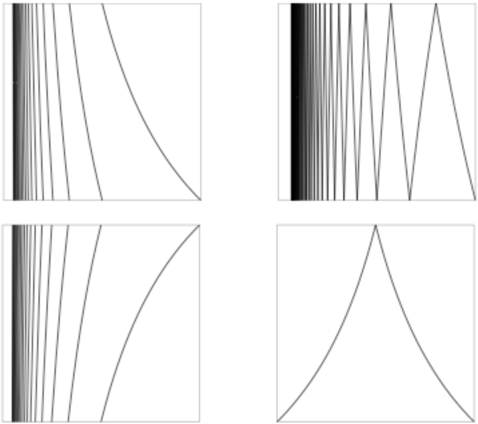


Two different classes

Fast Class

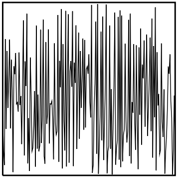
Slow Class

Four Euclidean dynamical systems (related to MSB divisions)



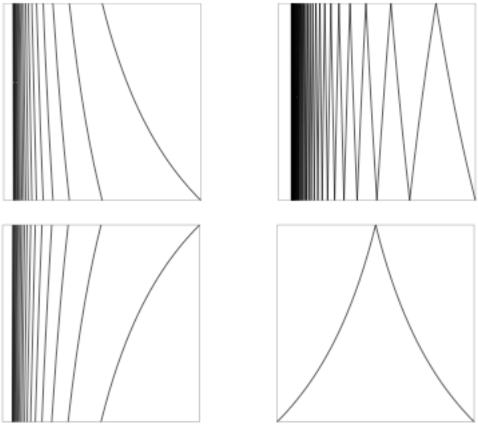
Two different classes

Fast Class



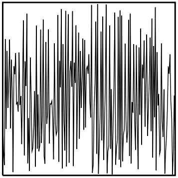
Slow Class

Four Euclidean dynamical systems (related to MSB divisions)

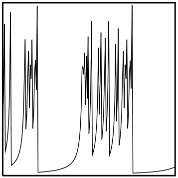


Two different classes

Fast Class



Slow Class



Dynamical Systems relative to MSB Algorithms.

Key Property : Expansiveness of branches of the shift U

$$|U'(x)| \geq A > 1 \text{ for all } x \text{ in } \mathcal{I}$$

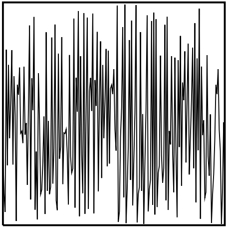
When true, this implies a chaotic behaviour for trajectories.

The associated algos are Fast and belong to the Good Class

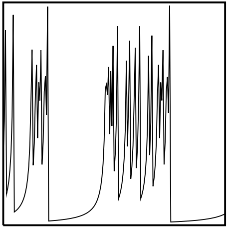
When this condition is violated at only one indifferent point,

this leads to intermittency phenomena.

The associated algos are Slow.



Chaotic Orbit [Fast Class],



Intermittent Orbit [SlowClass].

An instance of a Mixed Algorithm.

The Subtractive Algorithm,

where the zeroes on the right are removed from the remainder

defines the Binary Algorithm.

Subtractive Gcd Algorithm.	Binary Gcd Algorithm.
Input. u, v; $v \geq u$	**Input.** u, v odd; $v \geq u$
`While` $(u \neq v)$ `do`	`While` $(u \neq v)$ `do`
`While` $v > u$ `do`	`While` $v > u$ `do`
	$k := \nu_2(v-u)$;
$v := v - u$	$v := \dfrac{v-u}{2^k}$;
Exchange u and v.	Exchange u and v.
Output. u (or v).	**Output.** u (or v).

The 2-adic valuation ν_2 counts the number of zeroes on the right

An instance of a LSB Algorithm.

On a pair (u, v) with v odd and u even,

with $\nu_2(u) = k$, of the form $u := 2^k\, u'$

the LSB division writes $\qquad v = a \cdot u' + 2^k \cdot r'$,

with $\nu_2(r') > \nu_2(u') = 0$ and $\gcd(u, v) = \gcd(r', u')$.

The pair (u', r') will be the new pair for the next step.

An execution of the LSB Algorithm:

the Tortoise and the Hare

0	10001100101000001
1	11110101100000101000
2	1100100110110100000
3	11000011000101000000
4	1001100011110000000
5	11101001010100000000
6	11000001001000000000
7	10001000110000000000
8	100000101100000000000
9	11000000000000
10	100000100000000000000
11	10001000000000000000
12	11000000000000000000
13	100000000000000000000000

Two Euclidean dynamical systems, related to mixed or LSB divisions: the Binary Algorithm and the LSB Algorithm.

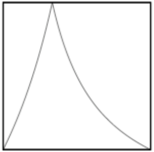
These algorithms use the 2–adic valuation ν .... only defined on rationals.

The 2–adic valuation ν is extended to a real random variable ν with

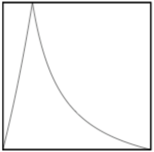
$$\Pr[\nu = k] = 1/2^k \qquad \text{for} \quad k \geq 1.$$

This gives rise to probabilistic dynamical systems.

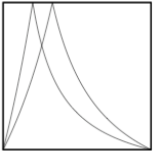
(I) The DS relative to the Binary Algorithm



$k = 1$



$k = 2$



$k = 1$ and $k = 2$

Two other Euclidean dynamical systems, related to mixed or LSB divisions: the Binary Algorithm and the LSB Algorithm.

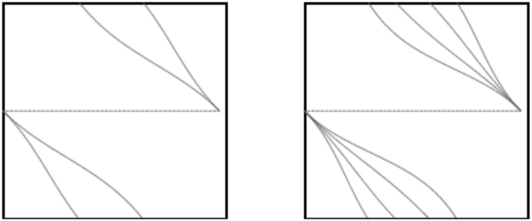
These algorithms use the 2–adic valuation ν .... only defined on rationals.

The 2–adic valuation ν is extended to a real random variable ν with

$$\Pr[\nu = k] = 1/2^k \qquad \text{for} \quad k \geq 1.$$

This gives rise to probabilistic dynamic systems.

(II) The DS relative to the LSB Algorithm



Two other Euclidean dynamical systems, related to mixed or LSB divisions:
the Binary Algorithm and the LSB Algorithm.

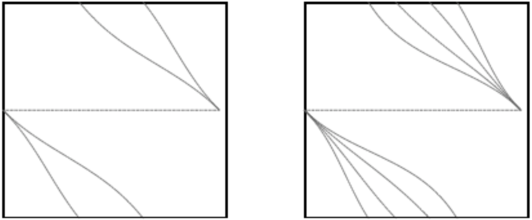
These algorithms use the 2–adic valuation ν .... only defined on rationals.

The 2–adic valuation ν is extended to a real random variable ν with

$$\Pr[\nu = k] = 1/2^k \qquad \text{for} \quad k \geq 1.$$

This gives rise to probabilistic dynamic systems.

(II) The DS relative to the LSB Algorithm



Part I – The Euclidean Algorithms

I-1. Two main Euclid algorithms

I-2. Many variants

I-3.- Algorithmic study

I-4. Some extensions

Why using the Euclidean algorithm? For which computations?

– the $\gcd(u, v)$ itself

Essential in exact rational computations,

for keeping rational numbers under their irreducible forms

60% of the computation time in some symbolic computations

– the Continued Fraction Expansion CFE (u/v)

Often used directly in computation over rationals.

– the modular inverse $u^{-1} \mod v$, when $\gcd(u, v) = 1$.

or more generally

– the algorithmic version of the Chinese Remainder Theorem

Why using the Euclidean algorithm? For which computations?

– the $\gcd(u, v)$ itself

Essential in exact rational computations,

for keeping rational numbers under their irreducible forms

60% of the computation time in some symbolic computations

– the Continued Fraction Expansion CFE (u/v)

Often used directly in computation over rationals.

– the modular inverse $u^{-1} \mod v$, when $\gcd(u, v) = 1$.

or more generally

– the algorithmic version of the Chinese Remainder Theorem

Why using the Euclidean algorithm? For which computations?

– the $\gcd(u, v)$ itself

Essential in exact rational computations,

for keeping rational numbers under their irreducible forms

60% of the computation time in some symbolic computations

– the Continued Fraction Expansion CFE (u/v)

Often used directly in computation over rationals.

– the modular inverse $u^{-1} \mod v$, when $\gcd(u, v) = 1$.

or more generally

– the algorithmic version of the Chinese Remainder Theorem

The extended Euclid Algoritthm.

Also computes Bezout coefficients a and b for which $au_0+bu_1 = \gcd(u_0, u_1)$

Execution of the plain algorithm:

$$\left\{\begin{array}{ccccc} u_0 & = & m_1 u_1 & + & u_2 & & 0 < u_2 < u_1 \\ u_1 & = & m_2 u_2 & + & u_3 & & 0 < u_3 < u_2 \\ \dots & = & \dots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & u_p & & 0 < u_p < u_{p-1} \\ u_{p-1} & = & m_p u_p & + & 0 & & u_{p+1} = 0 \end{array}\right\}$$

u_p is the gcd of u and v, the m_i's are the digits. p is the depth.

Compute two sequences a_i and b_i for which $a_i u_0 + b_i u_1 = u_i$. (1)

– The pair (a_p, b_p) is convenient.

– The following sequences satisfy (1) for all i (with an easy recurrence)

$$a_0 = 1,\ b_0 = 0; \quad a_1 = 0,\ b_1 = 1;$$

$$a_{i+1} = a_{i-1} - m_i a_i; \quad b_{i+1} = b_{i-1} - m_i b_i \qquad (i \geq 1)$$

–They may be computed during the execution of the plain Algorithm

The extended Euclid Algoritthm.

Also computes Bezout coefficients a and b for which $au_0+bu_1 = \gcd(u_0, u_1)$

Execution of the plain algorithm:

$$\left\{\begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & u_2 & & 0 < u_2 < u_1 \\ u_1 & = & m_2 u_2 & + & u_3 & & 0 < u_3 < u_2 \\ \dots & = & \dots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & u_p & & 0 < u_p < u_{p-1} \\ u_{p-1} & = & m_p u_p & + & 0 & & u_{p+1} = 0 \end{array}\right\}$$

u_p is the gcd of u and v, the m_i's are the digits. p is the depth.

Compute two sequences a_i and b_i for which $a_i u_0 + b_i u_1 = u_i$. (1)

– The pair (a_p, b_p) is convenient.

– The following sequences satisfy (1) for all i (with an easy recurrence)

$$a_0 = 1,\ b_0 = 0; \quad a_1 = 0,\ b_1 = 1;$$

$$a_{i+1} = a_{i-1} - m_i a_i; \quad b_{i+1} = b_{i-1} - m_i b_i \qquad (i \geq 1)$$

–They may be computed during the execution of the plain Algorithm

The extended Euclid Algoritthm.

Also computes Bezout coefficients a and b for which $au_0 + bu_1 = \gcd(u_0, u_1)$

Execution of the plain algorithm:

$$\left\{\begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & u_2 & & 0 < u_2 < u_1 \\ u_1 & = & m_2 u_2 & + & u_3 & & 0 < u_3 < u_2 \\ \dots & = & \dots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & u_p & & 0 < u_p < u_{p-1} \\ u_{p-1} & = & m_p u_p & + & 0 & & u_{p+1} = 0 \end{array}\right\}$$

u_p is the gcd of u and v, the m_i's are the digits. p is the depth.

Compute two sequences a_i and b_i for which $a_i u_0 + b_i u_1 = u_i$. (1)

– The pair (a_p, b_p) is convenient.

– The following sequences satisfy (1) for all i (with an easy recurrence)

$$a_0 = 1,\ b_0 = 0; \quad a_1 = 0,\ b_1 = 1;$$

$$a_{i+1} = a_{i-1} - m_i a_i; \quad b_{i+1} = b_{i-1} - m_i b_i \qquad (i \geq 1)$$

–They may be computed during the execution of the plain Algorithm

The extended Euclid Algoritthm.

Also computes Bezout coefficients a and b for which $au_0+bu_1 = \gcd(u_0, u_1)$

Execution of the plain algorithm:

$$\left\{\begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & u_2 & & 0 < u_2 < u_1 \\ u_1 & = & m_2 u_2 & + & u_3 & & 0 < u_3 < u_2 \\ \dots & = & \dots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & u_p & & 0 < u_p < u_{p-1} \\ u_{p-1} & = & m_p u_p & + & 0 & & u_{p+1} = 0 \end{array}\right\}$$

u_p is the gcd of u and v, the m_i's are the digits. p is the depth.

Compute two sequences a_i and b_i for which $a_i u_0 + b_i u_1 = u_i$. (1)

– The pair (a_p, b_p) is convenient.

– The following sequences satisfy (1) for all i (with an easy recurrence)

$$a_0 = 1,\ b_0 = 0; \quad a_1 = 0,\ b_1 = 1;$$

$$a_{i+1} = a_{i-1} - m_i a_i; \quad b_{i+1} = b_{i-1} - m_i b_i \qquad (i \geq 1)$$

–They may be computed during the execution of the plain Algorithm

The extended Euclid Algoritthm.

Also computes Bezout coefficients a and b for which $au_0+bu_1 = \gcd(u_0, u_1)$

Execution of the plain algorithm:

$$\left\{\begin{array}{ccccccc} u_0 & = & m_1 u_1 & + & u_2 & & 0 < u_2 < u_1 \\ u_1 & = & m_2 u_2 & + & u_3 & & 0 < u_3 < u_2 \\ \ldots & = & \ldots & + & & & \\ u_{p-2} & = & m_{p-1} u_{p-1} & + & u_p & & 0 < u_p < u_{p-1} \\ u_{p-1} & = & m_p u_p & + & 0 & & u_{p+1} = 0 \end{array}\right\}$$

u_p is the gcd of u and v, the m_i's are the digits. p is the depth.

Compute two sequences a_i and b_i for which $a_i u_0 + b_i u_1 = u_i$. (1)

– The pair (a_p, b_p) is convenient.

– The following sequences satisfy (1) for all i (with an easy recurrence)

$$a_0 = 1,\ b_0 = 0; \quad a_1 = 0,\ b_1 = 1;$$

$$a_{i+1} = a_{i-1} - m_i a_i; \quad b_{i+1} = b_{i-1} - m_i b_i \qquad (i \geq 1)$$

–They may be computed during the execution of the plain Algorithm

Chinese Remainder Theorem

Consider k integers $n_1, n_2, \ldots, n_k$
for which all the pairs (n_i, n_j) are coprime for all $i \neq j$.
Consider $n := \prod_{i=1}^{k} n_i$.
Then, for any k-uple (y_i), there exists a unique $y \in [1, n]$
for which $y = y_i \mod n_i$.

Let $q_i = \prod_{j \neq i} n_j$.
For each $j \in [1..k]$, there exists a pair (u_j, v_j) for which $u_j n_j + v_j q_j = 1$.
The integer $w_j := v_j q_j$ satisfies

$$w_j = 0 \mod n_i \quad (j \neq i), \qquad w_i = 1 \mod n_i.$$

Then $\quad y = \sum_{j=1}^{k} w_j y_j \quad$ satisfies $\quad y \mod n_i = y_i \mod n_i$

Chinese Remainder Theorem

Consider k integers $n_1, n_2, \ldots, n_k$
for which all the pairs (n_i, n_j) are coprime for all $i \neq j$.
Consider $n := \prod_{i=1}^{k} n_i$.
Then, for any k-uple (y_i), there exists a unique $y \in [1, n]$
for which $y = y_i \mod n_i$.

Let $q_i = \prod_{j \neq i} n_j$.
For each $j \in [1..k]$, there exists a pair (u_j, v_j) for which $u_j n_j + v_j q_j = 1$.
The integer $w_j := v_j q_j$ satisfies

$$w_j = 0 \mod n_i \quad (j \neq i), \qquad w_i = 1 \mod n_i.$$

Then $\quad y = \sum_{j=1}^{k} w_j y_j \quad$ satisfies $\quad y \mod n_i = y_i \mod n_i$

Chinese Remainder Theorem

Consider k integers $n_1, n_2, \ldots, n_k$
for which all the pairs (n_i, n_j) are coprime for all $i \neq j$.
Consider $n := \prod_{i=1}^{k} n_i$.
Then, for any k-uple (y_i), there exists a unique $y \in [1, n]$
for which $y = y_i \mod n_i$.

Let $q_i = \prod_{j \neq i} n_j$.
For each $j \in [1..k]$, there exists a pair (u_j, v_j) for which $u_j n_j + v_j q_j = 1$.
The integer $w_j := v_j q_j$ satisfies

$$w_j = 0 \mod n_i \quad (j \neq i), \qquad w_i = 1 \mod n_i.$$

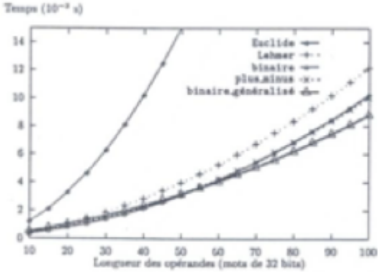
Then $\quad y = \sum_{j=1}^{k} w_j y_j \quad$ satisfies $\quad y \mod n_i = y_i \mod n_i$

Main algorithmic questions.

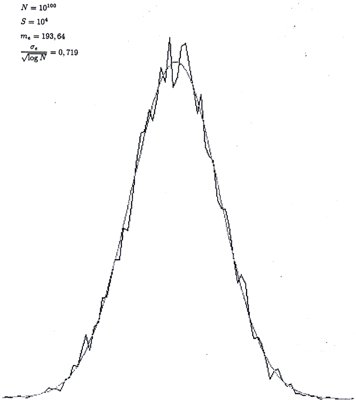
– Analyse the behaviour and the efficiency of these various versions

– Compare them with respect to various costs

and particularly the bit–complexity.



Experimental comparison
of bit–complexities.



A gaussian law
for the number of steps?

Comparison for five algorithms on the input $(2011176, 72001)$

Evolution of the remainders

Standard	Centered	By-Excess	Binary	LSB
67149	4852	4852	44849	51637
4852	779	779	1697	12485
4073	178	601	1697	2447
779	67	423	125	3733
178	23	245	125	1545
67	2	67	9	547
44	1	23	9	523
23	–	2	5	3
19	–	1	1	65
4	–	–	–	17
3	–	–	–	3
1	–	–	–	1

Comparison for five algorithms on the input $(2011176, 72001)$

Evolution of the remainders

Standard	Centered	By-Excess	Binary	LSB
67149	4852	4852	44849	51637
4852	779	779	1697	12485
4073	178	601	1697	2447
779	67	423	125	3733
178	23	245	125	1545
67	2	67	9	547
44	1	23	9	523
23	–	2	5	3
19	–	1	1	65
4	–	–	–	17
3	–	–	–	3
1	–	–	–	1

Explain the behaviour of algorithms

For instance, an execution of the LSB Algorithm : the Tortoise and the Hare

0	10001100101000001
1	111101011000000101000
2	11001001101101010000
3	110000110001010000000
4	10011000111100000000
5	111010010101000000000
6	110000010010000000000
7	100010001100000000000
8	1000001011000000000000
9	1100000000000000
10	1000001000000000000000
11	100010000000000000000
12	110000000000000000000
13	10000000000000000000000

Analysis of Algorithms

Analysis of the worst-case or of the generic case?

(Probabilistic) Analysis of Algorithms

– An algorithm with a set of inputs Ω:

here $\Omega := \{(u, v) \in \mathbb{N}^2, \mid 0 \le u \le v\}$

– A cost C defined on Ω which describes

  – the execution of the algorithm (number of iterations, bit–complexity)

  – or the geometry of the output (here: the continued fraction)

– Gather the inputs wrt to their sizes:

here: $\Omega_M := \{(u, v) \in \Omega, \quad 0 \le u \le v \le M\}$

– Study the cost C on Ω_M, in an asymptotic way for $M \to \infty$

Two possibilities:

  – Worst-case study : $W_M := \max\{C(u, v) \mid (u, v) \in \Omega_M\}$,

  – Probabilistic study :

    – with a distribution on Ω_M, study the random variable C on Ω_M

    – Estimate the mean value of $C_M := C_{|\Omega_M}$,

its variance, its distribution...

(Probabilistic) Analysis of Algorithms

– An algorithm with a set of inputs Ω:

$$\text{here } \Omega := \{(u,v) \in \mathbb{N}^2, \mid 0 \le u \le v\}$$

– A cost C defined on Ω which describes

  – the execution of the algorithm (number of iterations, bit–complexity)

  – or the geometry of the output (here: the continued fraction)

– Gather the inputs wrt to their sizes:

$$\text{here: } \Omega_M := \{(u,v) \in \Omega, \quad 0 \le u \le v \le M\}$$

– Study the cost C on Ω_M, in an asymptotic way for $M \to \infty$

Two possibilities:

  – Worst-case study : $W_M := \max\{C(u,v) \mid (u,v) \in \Omega_M\}$,

  – Probabilistic study :

    – with a distribution on Ω_M, study the random variable C on Ω_M

    – Estimate the mean value of $C_M := C_{|\Omega_M}$,

its variance, its distribution...

(Probabilistic) Analysis of Algorithms

– An algorithm with a set of inputs Ω:

$$\text{here } \Omega := \{(u,v) \in \mathbb{N}^2, \mid 0 \le u \le v\}$$

– A cost C defined on Ω which describes

  – the execution of the algorithm (number of iterations, bit–complexity)

  – or the geometry of the output (here: the continued fraction)

– Gather the inputs wrt to their sizes:

$$\text{here: } \Omega_M := \{(u,v) \in \Omega, \quad 0 \le u \le v \le M\}$$

– Study the cost C on Ω_M, in an asymptotic way for $M \to \infty$

Two possibilities:

  – Worst-case study : $W_M := \max\{C(u,v) \mid (u,v) \in \Omega_M\}$,

  – Probabilistic study :

    – with a distribution on Ω_M, study the random variable C on Ω_M

    – Estimate the mean value of $C_M := C_{|\Omega_M}$,

its variance, its distribution...

(Probabilistic) Analysis of Algorithms

– An algorithm with a set of inputs Ω:

here $\Omega := \{(u, v) \in \mathbb{N}^2, \mid 0 \leq u \leq v\}$

– A cost C defined on Ω which describes

  – the execution of the algorithm (number of iterations, bit–complexity)

  – or the geometry of the output (here: the continued fraction)

– Gather the inputs wrt to their sizes:

here: $\Omega_M := \{(u, v) \in \Omega, \quad 0 \leq u \leq v \leq M\}$

– Study the cost C on Ω_M, in an asymptotic way for $M \to \infty$

Two possibilities:

  – Worst-case study : $W_M := \max\{C(u, v) \mid (u, v) \in \Omega_M\}$,

  – Probabilistic study :

    – with a distribution on Ω_M, study the random variable C on Ω_M

    – Estimate the mean value of $C_M := C_{|\Omega_M}$,

its variance, its distribution...

Some results on the analysis of Euclidean Algorithms.

Worst-case analysis.

For the classical Euclid algorithm, the sequence

$$F_0 = 1, \qquad F_1 = 1, \quad F_{n+1} = F_n + F_{n-1}$$

represents the sequence of the smallest possible numbers which possibly appear in the execution. Then the pair (F_{n+1}, F_n) is the smallest pair on which the Euclid Algo performs $n + 1$ iterations.
Then (with ϕ the Golden ratio,)

$$R(u, v) \geq n + 1 \Longrightarrow v \geq F_{n+1} \geq \phi^{n+1}/\sqrt{5}$$

The maximum number R_M of iterations of the Euclid Algo on Ω_M

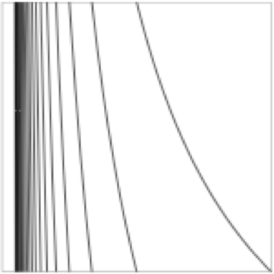
$$R_M \leq n + 1 \leq \log_\phi(\sqrt{5}M)$$

For the `C-Euclid` algorithm, the minimal sequence is

$$A_0 = 0, \quad A_1 = 1, \qquad A_{n+1} = 2A_n + A_{n-1}, \quad A_n \geq (1 + \sqrt{2})^{n-2}$$

Return to the underlying dynamical systems.

$$V(x) := \frac{1}{x} - \left\lfloor \frac{1}{x} \right\rfloor$$



The branches :

$$V_{[m]} : \left] \frac{1}{m+1}, \frac{1}{m} \right[\longrightarrow]0,1[,$$

$$V_{[m]}(x) := \frac{1}{x} - m$$

The inverse branches

$$h_{[m]} :]0,1[\longrightarrow \left] \frac{1}{m+1}, \frac{1}{m} \right[$$

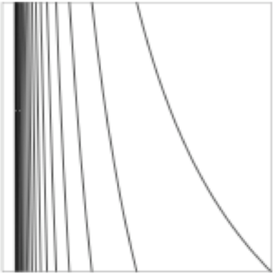
$$h_{[m]}(x) := \frac{1}{m+x}$$

The set $\mathcal{H}$ of the inverse branches of V builds CF's

$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{1}{m_2 + \cfrac{1}{\ddots + \cfrac{1}{m_p}}}} = h_{[m_1]} \circ h_{[m_2]} \circ \ldots \circ h_{[m_p]}(0)$$

Return to the underlying dynamical systems.

$$V(x) := \frac{1}{x} - \left\lfloor \frac{1}{x} \right\rfloor$$



The branches :

$$V_{[m]} : \left] \frac{1}{m+1}, \frac{1}{m} \right[\longrightarrow]0,1[,$$

$$V_{[m]}(x) := \frac{1}{x} - m$$

The inverse branches

$$h_{[m]} :]0,1[\longrightarrow \left] \frac{1}{m+1}, \frac{1}{m} \right[$$

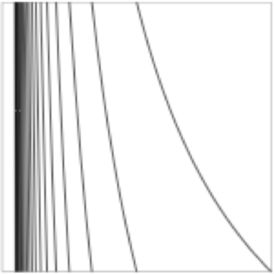
$$h_{[m]}(x) := \frac{1}{m+x}$$

The set $\mathcal{H}$ of the inverse branches of V builds CF's

$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{1}{m_2 + \cfrac{1}{\ddots + \cfrac{1}{m_p}}}} = h_{[m_1]} \circ h_{[m_2]} \circ \ldots \circ h_{[m_p]}(0)$$

Return to the underlying dynamical systems.

$$V(x) := \frac{1}{x} - \left\lfloor \frac{1}{x} \right\rfloor$$



The branches :

$$V_{[m]} : \left] \frac{1}{m+1}, \frac{1}{m} \right[\longrightarrow]0,1[,$$

$$V_{[m]}(x) := \frac{1}{x} - m$$

The inverse branches

$$h_{[m]} :]0,1[\longrightarrow \left] \frac{1}{m+1}, \frac{1}{m} \right[$$

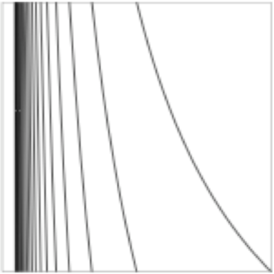
$$h_{[m]}(x) := \frac{1}{m+x}$$

The set $\mathcal{H}$ of the inverse branches of V builds CF's

$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{1}{m_2 + \cfrac{1}{\ddots + \cfrac{1}{m_p}}}} = h_{[m_1]} \circ h_{[m_2]} \circ \ldots \circ h_{[m_p]}(0)$$

Return to the underlying dynamical systems.

$$V(x) := \frac{1}{x} - \left\lfloor \frac{1}{x} \right\rfloor$$



The branches :

$$V_{[m]} : \left]\frac{1}{m+1}, \frac{1}{m}\right[\longrightarrow]0,1[,$$

$$V_{[m]}(x) := \frac{1}{x} - m$$

The inverse branches

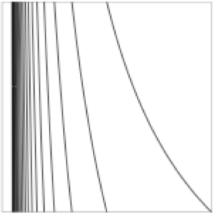
$$h_{[m]} :]0,1[\longrightarrow \left]\frac{1}{m+1}, \frac{1}{m}\right[$$

$$h_{[m]}(x) := \frac{1}{m+x}$$

The set $\mathcal{H}$ of the inverse branches of V builds CF's

$$\frac{u}{v} = \cfrac{1}{m_1 + \cfrac{1}{m_2 + \cfrac{1}{\ddots + \cfrac{1}{m_p}}}} = h_{[m_1]} \circ h_{[m_2]} \circ \ldots \circ h_{[m_p]}(0)$$

The Euclidean dynamical system.



Density Transformer:

For a density f on $[0,1]$, $\mathbf{H}[f]$ is the density on $[0,1]$ after one iteration of the shift

$$\mathbf{H}[f](x) = \sum_{h \in \mathcal{H}} |h'(x)|\, f \circ h(x) = \sum_{m \in \mathbb{N}} \frac{1}{(m+x)^2} f(\frac{1}{m+x}).$$

Transfer operator (Ruelle):

$$\mathbf{H}_s[f](x) = \sum_{h \in \mathcal{H}} |h'(x)|^s\, f \circ h(x).$$

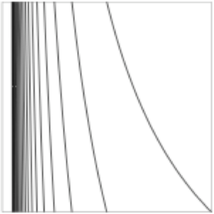
The operator $\mathbf{H}_s$ is a central tool for the analysis.

For $s = 1$, it coincides with the density transformer. One has

$$\mathbf{H}\left[\frac{1}{1+x}\right] = \frac{1}{1+x}$$

The function $f(x) = 1/(1+x)$ is an eigenfunction for $\lambda = 1$.

The Euclidean dynamical system.



Density Transformer:

For a density f on $[0,1]$, $\mathbf{H}[f]$ is the density on $[0,1]$ after one iteration of the shift

$$\mathbf{H}[f](x) = \sum_{h \in \mathcal{H}} |h'(x)| \, f \circ h(x) = \sum_{m \in \mathbb{N}} \frac{1}{(m+x)^2} f(\frac{1}{m+x}).$$

Transfer operator (Ruelle):

$$\mathbf{H}_s[f](x) = \sum_{h \in \mathcal{H}} |h'(x)|^s \, f \circ h(x).$$

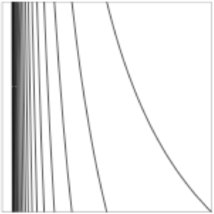
The operator $\mathbf{H}_s$ is a central tool for the analysis.

For $s = 1$, it coincides with the density transformer. One has

$$\mathbf{H}\left[\frac{1}{1+x}\right] = \frac{1}{1+x}$$

The function $f(x) = 1/(1+x)$ is an eigenfunction for $\lambda = 1$.

The Euclidean dynamical system.



Density Transformer:

For a density f on $[0,1]$, $\mathbf{H}[f]$ is the density on $[0,1]$ after one iteration of the shift

$$\mathbf{H}[f](x) = \sum_{h\in\mathcal{H}} |h'(x)|\, f\circ h(x) = \sum_{m\in\mathbb{N}} \frac{1}{(m+x)^2} f(\frac{1}{m+x}).$$

Transfer operator (Ruelle):

$$\mathbf{H}_s[f](x) = \sum_{h\in\mathcal{H}} |h'(x)|^s\, f\circ h(x).$$

The operator $\mathbf{H}_s$ is a central tool for the analysis.

For $s=1$, it coincides with the density transformer. One has

$$\mathbf{H}\left[\frac{1}{1+x}\right] = \frac{1}{1+x}$$

The function $f(x) = 1/(1+x)$ is an eigenfunction for $\lambda = 1$.

Here, focus on average-case results on Ω_M [input size $:= \log M$]

– For the Standard, Centered Euclidean Algorithms,

– the mean number of iterations is

$$\mathbb{E}_M[P] \sim \frac{2}{h(\mathcal{S})} \log M,$$

– the mean bit-complexity is quadratic.

$$\mathbb{E}_M[B] \sim \frac{1}{h(\mathcal{S})} \mu[\ell] \log^2 M$$

– The main constant $h(\mathcal{S})$ is the entropy of the Dynamical System.

The entropy is a well-defined mathematical object, computable.

$$h(\mathcal{S}) = \frac{\pi^2}{6 \log 2} \sim 2.37 \quad \text{[Standard]}, \qquad h(\mathcal{S}) = \frac{\pi^2}{6 \log \phi} \sim 3.41 \quad \text{[Centered]}.$$

– The constant $\mu[\ell]$ is the mean value of cost c, with respect to the invariant density. For Centered Euclidean Alg.

$$\mu(\ell) = 3 + \frac{\log 2}{\log \phi} + \frac{1}{\log \phi} \prod_{k \geq 3} \frac{(2^k - 1)\phi^2 + 2\phi}{(2^k - 1)\phi^2 - 2}$$

Here, focus on average-case results on Ω_M [input size $:= \log M$]

– For the Standard, Centered Euclidean Algorithms,

– the mean number of iterations is

$$\mathbb{E}_M[P] \sim \frac{2}{h(\mathcal{S})} \log M,$$

– the mean bit-complexity is quadratic.

$$\mathbb{E}_M[B] \sim \frac{1}{h(\mathcal{S})} \mu[\ell] \log^2 M$$

– The main constant $h(\mathcal{S})$ is the entropy of the Dynamical System.

The entropy is a well-defined mathematical object, computable.

$$h(\mathcal{S}) = \frac{\pi^2}{6 \log 2} \sim 2.37 \ \text{[Standard]}, \quad h(\mathcal{S}) = \frac{\pi^2}{6 \log \phi} \sim 3.41 \ \text{[Centered]}.$$

– The constant $\mu[\ell]$ is the mean value of cost c, with respect to the invariant density. For Centered Euclidean Alg.

$$\mu(\ell) = 3 + \frac{\log 2}{\log \phi} + \frac{1}{\log \phi} \prod_{k \geq 3} \frac{(2^k - 1)\phi^2 + 2\phi}{(2^k - 1)\phi^2 - 2}$$

Here, focus on average-case results on Ω_M [input size $:= \log M$]

– For the Standard, Centered Euclidean Algorithms,

– the mean number of iterations is

$$\mathbb{E}_M[P] \sim \frac{2}{h(\mathcal{S})} \log M,$$

– the mean bit-complexity is quadratic.

$$\mathbb{E}_M[B] \sim \frac{1}{h(\mathcal{S})} \mu[\ell] \log^2 M$$

– The main constant $h(\mathcal{S})$ is the entropy of the Dynamical System.

The entropy is a well-defined mathematical object, computable.

$$h(\mathcal{S}) = \frac{\pi^2}{6 \log 2} \sim 2.37 \text{ [Standard]}, \quad h(\mathcal{S}) = \frac{\pi^2}{6 \log \phi} \sim 3.41 \text{ [Centered]}.$$

– The constant $\mu[\ell]$ is the mean value of cost c, with respect to the invariant density. For Centered Euclidean Alg.

$$\mu(\ell) = 3 + \frac{\log 2}{\log \phi} + \frac{1}{\log \phi} \prod_{k \geq 3} \frac{(2^k - 1)\phi^2 + 2\phi}{(2^k - 1)\phi^2 - 2}$$

Part I – The Euclidean Algorithms

I-1. Two main Euclid algorithms

I-2. Many variants

I-3.- Algorithmic study

I-4. Some extensions

Extension I

Mean bit–complexity of fast variants of the Euclid Algorithm

Main principles of Fast Euclid Algorithms:

– Based on a Divide and Conquer principle with two recursive calls.

– Study "slices" of the original Euclid Algorithm

  – begin when the data has already lost δn bits.

  – end when the data has lost γn additional bits.

– Replace large divisions by small divisions and large multiplications.

– Use fast multiplication algorithms (based on the FFT)

of complexity $n \log n \, a(n)$

with $a(n) = \log\log n$ [Schönhage Strassen]

now $a(n) = 2^{O(\log^\star n)}$ [Fürer, 2007]

with $\log^\star n =$ the smallest integer k for which $\log^{(k)} n < 1$

We obtain the mean bit-complexity of (variants of) these algorithms

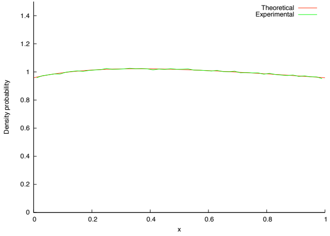
$$\Theta(n(\log n)^2 a(n))$$

with a precise estimate of the hidden constants

Analysis based on the answer to the question:

What is the distribution of the data

when they have already lost a fraction δ of its bits?



Unexpected occurrence

of a particular density ψ

$$\psi(x) = \frac{12}{\pi^2} \sum_{m \geq 1} \frac{\log(m+x)}{(m+x)(m+x+1)}$$

distinct of the Gauss density

$$\varphi(x) = \frac{1}{\log 2} \frac{1}{1+x}$$

Extension II
Computing the gcd of ℓ inputs

For $\ell = 2$: the "classical" Euclid algorithm :

a sequence of Euclidean divisions

For $\ell \geq 3$, there are various strategies.

The plain algorithm performs a sequence of computations on two entries;
On the input $(x_1, x_2, \ldots, x_\ell)$, it computes

– first: $y_2 := \gcd(x_1, x_2)$

– then, for $k \in [3..\ell]$: $y_k := \gcd(x_k, y_{k-1}) = \gcd(x_1, x_2, \ldots, x_k)$.

The "total" gcd $y_\ell := \gcd(x_1, x_2, \ldots, x_\ell)$ is obtained after $\ell - 1$ phases.
Each phase performs a call to the classical Euclid algorithm.

A very natural scheme, proposed in Knuth's book.....

but not yet analyzed for $\ell > 2$ (Problem HM 48)

Extension II

Computing the gcd of ℓ inputs

For $\ell = 2$: the "classical" Euclid algorithm :

a sequence of Euclidean divisions

For $\ell \geq 3$, there are various strategies.

The plain algorithm performs a sequence of computations on two entries;
On the input $(x_1, x_2, \ldots, x_\ell)$, it computes

– first: $y_2 := \gcd(x_1, x_2)$

– then, for $k \in [3..\ell]$: $y_k := \gcd(x_k, y_{k-1}) = \gcd(x_1, x_2, \ldots, x_k)$.

The "total" gcd $y_\ell := \gcd(x_1, x_2, \ldots, x_\ell)$ is obtained after $\ell - 1$ phases.
Each phase performs a call to the classical Euclid algorithm.

A very natural scheme, proposed in Knuth's book.....

but not yet analyzed for $\ell > 2$ (Problem HM 48)

Extension II

Computing the gcd of ℓ inputs

For $\ell = 2$: the "classical" Euclid algorithm :

a sequence of Euclidean divisions

For $\ell \geq 3$, there are various strategies.

The plain algorithm performs a sequence of computations on two entries;
On the input $(x_1, x_2, \ldots, x_\ell)$, it computes

– first: $y_2 := \gcd(x_1, x_2)$

– then, for $k \in [3..\ell]$: $y_k := \gcd(x_k, y_{k-1}) = \gcd(x_1, x_2, \ldots, x_k)$.

The "total" gcd $y_\ell := \gcd(x_1, x_2, \ldots, x_\ell)$ is obtained after $\ell - 1$ phases.
Each phase performs a call to the classical Euclid algorithm.

A very natural scheme, proposed in Knuth's book.....

but not yet analyzed for $\ell > 2$ (Problem HM 48)

Extension II

Computing the gcd of ℓ inputs

For $\ell = 2$: the "classical" Euclid algorithm :

a sequence of Euclidean divisions

For $\ell \geq 3$, there are various strategies.

The plain algorithm performs a sequence of computations on two entries;
On the input $(x_1, x_2, \ldots, x_\ell)$, it computes

– first: $y_2 := \gcd(x_1, x_2)$

– then, for $k \in [3..\ell]$: $y_k := \gcd(x_k, y_{k-1}) = \gcd(x_1, x_2, \ldots, x_k)$.

The "total" gcd $y_\ell := \gcd(x_1, x_2, \ldots, x_\ell)$ is obtained after $\ell - 1$ phases.
Each phase performs a call to the classical Euclid algorithm.

A very natural scheme, proposed in Knuth's book.....

but not yet analyzed for $\ell > 2$ (Problem HM 48)

Extension II

Computing the gcd of ℓ inputs

For $\ell = 2$: the "classical" Euclid algorithm :

a sequence of Euclidean divisions

For $\ell \geq 3$, there are various strategies.

The plain algorithm performs a sequence of computations on two entries;
On the input $(x_1, x_2, \ldots, x_\ell)$, it computes

– first: $y_2 := \gcd(x_1, x_2)$

– then, for $k \in [3..\ell]$: $y_k := \gcd(x_k, y_{k-1}) = \gcd(x_1, x_2, \ldots, x_k)$.

The "total" gcd $y_\ell := \gcd(x_1, x_2, \ldots, x_\ell)$ is obtained after $\ell - 1$ phases.
Each phase performs a call to the classical Euclid algorithm.

A very natural scheme, proposed in Knuth's book.....

but not yet analyzed for $\ell > 2$ (Problem HM 48)

Which behavior can be expected?

Knuth wrote: "In most cases, the size of the partial gcd decreases rapidly during the first few phases of the calculation.
This will make the remainder of the computation quite fast".

Our analysis exhibits a more precise phenomenon:
A strong difference between the first phase and the subsequent phases.
In most cases, "almost all the calculation" is done during the first phase.

We prove the following facts about the number of divisions performed, measured with respect to the size of the input:

- during the first phase:
  - it is linear on average,
  - it asymptotically follows a beta law;
- during subsequent phases:
  - it is constant on average
  - it asymptotically follows a geometric law

The same phenomena occur for the size of the partial gcd.

Which behavior can be expected?

Knuth wrote: "In most cases, the size of the partial gcd decreases rapidly during the first few phases of the calculation.
This will make the remainder of the computation quite fast".

Our analysis exhibits a more precise phenomenon:
A strong difference between the first phase and the subsequent phases.
In most cases, "almost all the calculation" is done during the first phase.

We prove the following facts about the number of divisions performed, measured with respect to the size of the input:

- during the first phase:
  - it is linear on average,
  - it asymptotically follows a beta law;
- during subsequent phases:
  - it is constant on average
  - it asymptotically follows a geometric law

The same phenomena occur for the size of the partial gcd.

Which behavior can be expected?

Knuth wrote: "In most cases, the size of the partial gcd decreases rapidly during the first few phases of the calculation.
This will make the remainder of the computation quite fast".

Our analysis exhibits a more precise phenomenon:
A strong difference between the first phase and the subsequent phases.
In most cases, "almost all the calculation" is done during the first phase.

We prove the following facts about the number of divisions performed, measured with respect to the size of the input:

- during the first phase:
  - it is linear on average,
  - it asymptotically follows a beta law;
- during subsequent phases:
  - it is constant on average
  - it asymptotically follows a geometric law

The same phenomena occur for the size of the partial gcd.

Which behavior can be expected?

Knuth wrote: "In most cases, the size of the partial gcd decreases rapidly during the first few phases of the calculation.
This will make the remainder of the computation quite fast".

Our analysis exhibits a more precise phenomenon:
A strong difference between the first phase and the subsequent phases.
In most cases, "almost all the calculation" is done during the first phase.

We prove the following facts about the number of divisions performed, measured with respect to the size of the input:

- during the first phase:
  - it is linear on average,
  - it asymptotically follows a beta law;
- during subsequent phases:
  - it is constant on average
  - it asymptotically follows a geometric law

The same phenomena occur for the size of the partial gcd.

Which behavior can be expected?

Knuth wrote: "In most cases, the size of the partial gcd decreases rapidly during the first few phases of the calculation.
This will make the remainder of the computation quite fast".

Our analysis exhibits a more precise phenomenon:
A strong difference between the first phase and the subsequent phases.
In most cases, "almost all the calculation" is done during the first phase.

We prove the following facts about the number of divisions performed, measured with respect to the size of the input:

- during the first phase:
  - it is linear on average,
  - it asymptotically follows a beta law;
- during subsequent phases:
  - it is constant on average
  - it asymptotically follows a geometric law

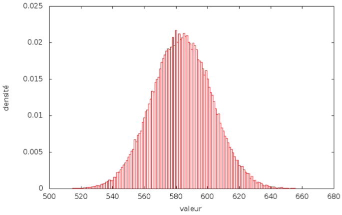
The same phenomena occur for the size of the partial gcd.

Examples of limit laws (discrete or continuous)

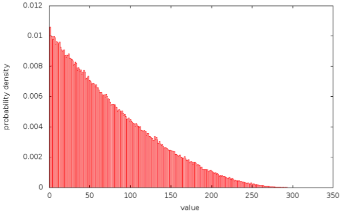
x-axis: possible values of the cost $L(\omega)$

y-axis: probability density $x \mapsto f(x)$

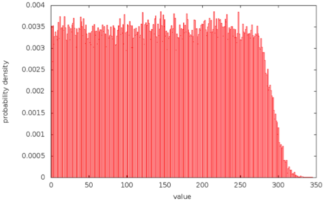
$$f(x)dx := \Pr[\omega;\quad L(\omega) \in [x, x+dx]]$$



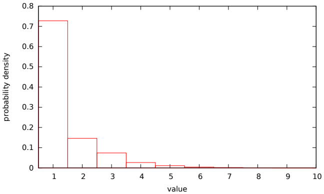
Gaussian law $f(t) \asymp e^{-t^2/2}$



Beta law (a,b) $f(t) \asymp t^{a-1}(1-t)^{b-1}$



Uniform law $f(t) \asymp 1$



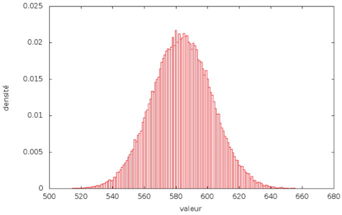
A discrete law: Geometric law $f(n) \asymp a^n$

Examples of limit laws (discrete or continuous)

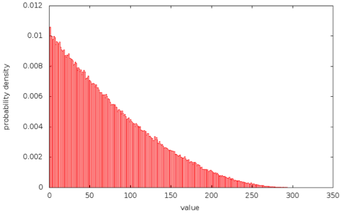
x-axis: possible values of the cost $L(\omega)$

y-axis: probability density $x \mapsto f(x)$

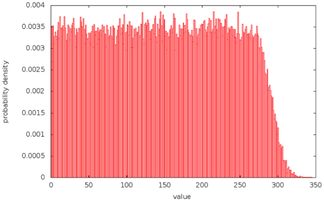
$$f(x)dx := \Pr[\omega; \quad L(\omega) \in [x, x+dx]]$$



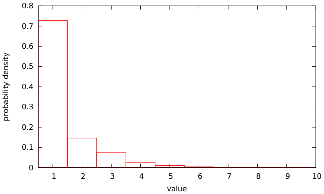
Gaussian law $f(t) \asymp e^{-t^2/2}$



Beta law (a, b) $f(t) \asymp t^{a-1}(1-t)^{b-1}$



Uniform law $f(t) \asymp 1$



A discrete law: Geometric law $f(n) \asymp a^n$