1 Complexity of real functions
   - Introduction
   - Computable Analysis
   - GPAC
   - Analog Church Thesis

2 Toward a Complexity Theory for the GPAC
   - What is the problem?
   - A complexity class

3 Conclusion
Motivating example

Example (Sine function)

Given $x \in \mathbb{R}$, compute $\sin(x)$.

⇒ "clearly $\sin$ is computable:"

But... how do you represent a real number? (infinite object)

Amaury Pouly

Computational Complexity of real functions

April 28, 2015 1 / 19
Motivating example

Example (Sine function)

Given $x \in \mathbb{R}$, compute $\sin(x)$. 

⇒ "clearly $\sin$ is computable."

But... how do you represent a real number? (infinite object)
Motivating example

Example (Sine function)

Given $x \in \mathbb{R}$, compute $\sin(x)$.

⇒ “clearly sin is computable:”
Motivating example

Example (Sine function)

Given $x \in \mathbb{R}$, compute $\sin(x)$.

⇒ “clearly $\sin$ is computable:”

But...

- how do you represent a real number? (infinite object)
Motivating example

Example (Sine function)

Given \( x \in \mathbb{R} \), compute \( \sin(x) \).

⇒ “clearly \( \sin \) is computable:”

But...

- how do you represent a real number? (infinite object)
- what is a program working on them?
A classical approach

Computable analysis
A classical approach

Computable analysis

- a real number is a program:
Complexity of real functions

A classical approach

Computable analysis

- a real number is a program: it computes arbitrary approximations
A classical approach

Computable analysis

- a real number is a program: it computes arbitrary approximations
- a function is a program transformation:
A classical approach

Computable analysis

- a real number is a program: it computes arbitrary approximations
- a function is a program transformation: it transforms one approximation into another
A classical approach

Computable analysis

- a real number is a program: it computes arbitrary approximations
- a function is a program transformation: it transforms one approximation into another

- Intuition: can draw the graph of a function with arbitrary zoom
- Very analytic, approximation theory
- Can lift Turing complexity to real functions
- Has a nice theory of open sets
Computable real

A real \( r \in \mathbb{R} \) is computable if one can compute an arbitrary close approximation for a given precision: Given \( p \in \mathbb{N} \), compute \( r_p \) such that

\[ |r - r_p| \leq 2^{-p} \]

Example: Rational numbers, \( \pi \), \( e \), . . .

Example (Non-computable real):

\[ r = \sum_{n=0}^{\infty} d_n 2^{-n} \]

where \( d_n = 1 \iff \) the \( n \)th Turing Machine halts on input \( n \).
A real \( r \in \mathbb{R} \) is computable is one can compute an arbitrary close approximation for a given precision:
Computable real

**Definition (Computable Real)**

A real \( r \in \mathbb{R} \) is computable is one can compute an arbitrary close approximation for a given precision:

Given \( p \in \mathbb{N} \), compute \( r_p \) s.t. \( |r - r_p| \leq 2^{-p} \)
Computable real

Definition (Computable Real)
A real $r \in \mathbb{R}$ is computable is one can compute an arbitrary close approximation for a given precision:

Given $p \in \mathbb{N}$, compute $r_p$ s.t. $|r - r_p| \leq 2^{-p}$

Example
Rational numbers, $\pi$, $e$, ...
Computable real

Definition (Computable Real)

A real \( r \in \mathbb{R} \) is computable is one can compute an arbitrary close approximation for a given precision:

Given \( p \in \mathbb{N} \), compute \( r_p \) s.t. \( |r - r_p| \leq 2^{-p} \)

Example

Rational numbers, \( \pi \), \( e \), \ldots

Example (Non-computable real)

\[
r = \sum_{n=0}^{\infty} d_n 2^{-n}
\]

where

\( d_n = 1 \iff \) the \( n^{th} \) Turing Machine halts on input \( n \)
Definition (Computable function)

\( f : [a, b] \rightarrow \mathbb{R} \) is computable iff \( \exists m, \psi \) computable functions s.t. \( \forall n \in \mathbb{N} \):

- \( \forall x, y, |x - y| \leq 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leq 2^{-n} \) ▶ effective continuity
- \( \forall r \in \mathbb{Q}, |\psi(r, n) - f(r)| \leq 2^{-n} \) ▶ approximability


**Computable function**

**Definition (Computable function)**

\( f : [a, b] \to \mathbb{R} \) is computable iff \( \exists m, \psi \) computable functions s.t. \( \forall n \in \mathbb{N} : \)

- \( \forall x, y, |x - y| \leq 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leq 2^{-n} \) ▶ effective continuity
- \( \forall r \in \mathbb{Q}, |\psi(r, n) - f(r)| \leq 2^{-n} \) ▶ approximability

**Definition (Equivalent)**

\( f : [a, b] \to \mathbb{R} \) is computable iff \( \exists M \) a Turing Machine s.t. \( \forall x \in [a, b] \) and oracle \( \mathcal{O} \) computing \( x \), \( M^\mathcal{O} \) computes \( f(x) \).
Computable function

**Definition (Computable function)**

$f : [a, b] \rightarrow \mathbb{R}$ is computable iff $\exists m, \psi$ computable functions s.t. $\forall n \in \mathbb{N}$:

- $\forall x, y, |x - y| \leq 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leq 2^{-n}$  ► effective continuity
- $\forall r \in \mathbb{Q}, |\psi(r, n) - f(r)| \leq 2^{-n}$  ► approximability

**Definition (Equivalent)**

$f : [a, b] \rightarrow \mathbb{R}$ is computable iff $\exists M$ a Turing Machine s.t. $\forall x \in [a, b]$ and oracle $O$ computing $x$, $M^O$ computes $f(x)$.

**Example**

Polynomials, trigonometric functions, $e^\cdot$, $\sqrt{\cdot}$, ...
Computable function

**Definition (Computable function)**

\( f : [a, b] \rightarrow \mathbb{R} \) is computable iff \( \exists m, \psi \) computable functions s.t. \( \forall n \in \mathbb{N} : \)

- \( \forall x, y, |x - y| \leq 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leq 2^{-n} \) \( \Rightarrow \) effective continuity
- \( \forall r \in \mathbb{Q}, |\psi(r, n) - f(r)| \leq 2^{-n} \) \( \Rightarrow \) approximability

**Definition (Equivalent)**

\( f : [a, b] \rightarrow \mathbb{R} \) is computable iff \( \exists M \) a Turing Machine s.t. \( \forall x \in [a, b] \) and oracle \( \mathcal{O} \) computing \( x \), \( M^{\mathcal{O}} \) computes \( f(x) \).

**Example**

Polynomials, trigonometric functions, \( e^x \), \( \sqrt{x} \), ...

**Example (Counter-Example)**

\( f(x) = \lceil x \rceil \) \( \Rightarrow \) not continuous
Thoughts on the model

- reuses existing theory on Turing machines
Thoughts on the model

- reuses existing theory on Turing machines
- gives “natural” complexity classes related to the classical ones
Thoughts on the model

- reuses existing theory on Turing machines
- gives “natural” complexity classes related to the classical ones
- but feels very discrete machine oriented
Thoughts on the model

- reuses existing theory on Turing machines
- gives “natural” complexity classes related to the classical ones
- but feels very discrete machine oriented

Question

Can we give a purely analog model of computation?
General Purpose Analog Computer

by Claude Shannon (1941)
General Purpose Analog Computer
- by Claude Shanon (1941)
- idealization of an analog computer: Differential Analyzer
General Purpose Analog Computer

- by Claude Shannon (1941)
- idealization of an analog computer: Differential Analyzer
- circuit built from:

  - A constant unit: $k$
  - An adder unit: $u + v$
  - An multiplier unit: $uv$
  - An integrator unit: $\int u \, dv$
Theorem

$y$ is generated by a GPAC iff it is a component of the solution $y = (y_1, \ldots, y_d)$ of the ordinary differential equation (ODE):

\[
\begin{aligned}
    y'(t) &= p(y(t)) \\
    y(t_0) &= y_0
\end{aligned}
\]

where $p$ is a vector of polynomials.
Example (One variable, linear system)

\[ \int e^t \]

\[
\begin{align*}
    y' &= y \\
    y(0) &= 1
\end{align*}
\]
Example (One variable, linear system)

\[ \int e^t \{ y' = y \quad y(0) = 1 \} \]

Example (One variable, nonlinear system)

\[ t \rightarrow -2 \rightarrow \times \rightarrow \times \rightarrow \int \frac{1}{1+t^2} \{ y' = -2ty^2 \quad y(0) = 1 \} \]
Example (One variable, linear system)
\[ t \quad \int \quad e^t \quad \left\{ \begin{array}{l} y' = y \\ y(0) = 1 \end{array} \right. \]

Example (Two variable, nonlinear system)
\[ t \quad \times \quad \times \quad \int \quad \frac{1}{1+t^2} \quad \left\{ \begin{array}{l} y' = -2ty^2 \\ y(0) = 1 \\ t' = 1 \\ t(0) = 0 \end{array} \right. \]
Example (Two variables, linear system)

\[ \begin{align*}
  t & \quad -1 \quad \times \quad f \quad f \quad \sin(t) \\
  y' &= z \\
  z' &= -y \\
  y(0) &= 0 \\
  z(0) &= 1
\end{align*} \]
Example (Two variables, linear system)

\[
\begin{align*}
\int & \int \\ y' & = z \\ z' & = -y \\ y(0) & = 0 \\ z(0) & = 1
\end{align*}
\]

Exercice (Tear your mind apart)

\[
\begin{align*}
y_1' & = y_1 \\
y_2' & = y_2y_1 \\
& \vdots \\
y_n' & = y_ny_{n-1}\cdots y_2y_1
\end{align*}
\]
**Example (Two variables, linear system)**

\[
\begin{array}{c}
t \\
\downarrow \quad \times \quad \int \quad \int \\
\downarrow \\
y(t) = \sin(t)
\end{array}
\]

\[
\begin{align*}
y' &= z \\
z' &= -y \\
y(0) &= 0 \\
z(0) &= 1
\end{align*}
\]

**Exercice (Tear your mind apart)**

\[
\begin{array}{c}
t \\
\downarrow \quad \int \quad \int \quad \cdots \\
\downarrow \quad \int \\
y_n(t)
\end{array}
\]

\[
\begin{align*}
y_1(t) &= e^t \\
y_2(t) &= e^{e^t} \\
&\quad \cdots \\
y_n(t) &= e^{e^{e^{\cdots^t}}}
\end{align*}
\]
Slight issue is...

- the GPAC generated functions are analytical
Slight issue is...

- the GPAC generated functions are analytical
- the computable functions from Computable Analysis are “only” continuous

Question
Can we bridge the gap? Why should we?
The case of discrete computations

Many models:

- Recursive functions
- Turing machines
- $\lambda$-calculus
- circuits
- ...
The case of discrete computations

Many models:
- Recursive functions
- Turing machines
- λ-calculus
- circuits
- ...

Church Thesis
All reasonable discrete models of computation are equivalent.
The case of discrete computations

Many models:
- Recursive functions
- Turing machines
- $\lambda$-calculus
- circuits
- ...

Church Thesis
All reasonable discrete models of computation are equivalent.

Can be extended to complexity when relevant.
GPAC: back to the basics

Definition

$f$ is **generated** by a GPAC iff it is a component of the solution $y$ of:

\[
\begin{align*}
    y' &= p(y) \\
    y(t_0) &= y_0
\end{align*}
\]
GPAC: back to the basics

**Definition**

$f$ is **generated** by a GPAC iff it is a component of the solution $y$ of:

\[
\left\{ \begin{array}{c}
y' = p(y) \\
y(t_0) = y_0
\end{array} \right.
\]

**Definition**

$f$ is **computable** by a GPAC iff $\exists p, q$ polynomials s.t. $\forall x \in \mathbb{R}$, the solution $y = (y_1, \ldots, y_d)$ of:

\[
\left\{ \begin{array}{c}
y' = p(y) \\
y(t_0) = q(x)
\end{array} \right.
\]

satisfies $f(x) = \lim_{t \to \infty} y_1(t)$. 

Amaury Pouly
Computational Complexity of real functions
April 28, 2015
**Definition**

$f$ is **computable** by a GPAC iff $\exists p, q$ polynomials s.t. $\forall x \in \mathbb{R}$, the solution $y = (y_1, \ldots, y_d)$ of:

$$
\begin{align*}
  y' &= p(y) \\
  y(t_0) &= q(x)
\end{align*}
$$

satisfies $f(x) = \lim_{t \to \infty} y_1(t)$.

**Example**

![Graph showing the function q(x) and its derivative y(t), along with the limit f(x) as t approaches infinity.](image-url)
Computable Analysis = GPAC ? (again)

Theorem (Bournez, Campagnolo, Graça, Hainry)

$f$ is GPAC-computable functions iff it is computable (in the sense of Computable Analysis).
Time Scaling

<table>
<thead>
<tr>
<th>System</th>
<th>#1</th>
<th>#2</th>
</tr>
</thead>
</table>
| PIVP   | \[
|       | \begin{align*}
|       | y'(t) &= p(y(t)) \\
|       | y(1) &= q(x) \\
|       | \end{align*}
|       |                                        | \begin{align*}
|       | z'(t) &= u(t)p(z(t)) \\
|       | u'(t) &= u(t) \\
|       | z(t_0) &= q(x) \\
|       | u(1) &= 1 \end{align*} |
Time Scaling

<table>
<thead>
<tr>
<th>System</th>
<th>#1</th>
<th>#2</th>
</tr>
</thead>
</table>
| PIVP   | \[
\begin{align*}
  y'(t) &= p(y(t)) \\
  y(1) &= q(x)
\end{align*}
\] | \[
\begin{align*}
  z'(t) &= u(t)p(z(t)) \\
  u'(t) &= u(t) \\
  z(t_0) &= q(x) \\
  u(1) &= 1
\end{align*}
\] |

Remark

Same curve, different speed: \( u(t) = e^t \) and \( z(t) = y(e^t) \)

Example
Time Scaling

<table>
<thead>
<tr>
<th>System</th>
<th>#1</th>
<th>#2</th>
</tr>
</thead>
</table>
| PIVP   | \[
\begin{align*}
    y'(t) &= p(y(t)) \\
    y(1) &= q(x)
\end{align*}
\] | \[
\begin{align*}
    z'(t) &= u(t)p(z(t)) \\
    u'(t) &= u(t) \\
    z(t_0) &= q(x) \\
    u(1) &= 1
\end{align*}
\] |

Computed Function

| Same |

Remark

Same curve, different speed: \( u(t) = e^t \) and \( z(t) = y(e^t) \)

Example

\[ y_0(x) \quad y(t) \quad f(x) \]
### Time Scaling

| PIVP         | $y' = p(y)$ | $z(t) = y(e^t)$ → \[
\begin{align*}
    z' &= up(z) \\
    u' &= u
\end{align*}
\]
| Computed Function | Same | Exponentially faster |
| Convergence       |       |                       |

**Example**

- $y_0(x)$
- $y(t)$
- $f(x)$
- $t$
**Time Scaling**

| PIVP | $y' = p(y)$ | $z(t) = y(e^t)$ → \[
\begin{cases}
  z' = \text{up}(z) \\
  u' = u
\end{cases}
\]
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Computed Function</td>
<td>Same</td>
<td></td>
</tr>
<tr>
<td>Time for precision $\mu$</td>
<td>$\text{tm}(\mu)$</td>
<td>$\text{tm}'(\mu) = \log(\text{tm}(\mu))$</td>
</tr>
</tbody>
</table>

**Example**

$\mu \uparrow$

$z(t)$

$y(t)$

$\|
\|y_1(\text{tm}(\mu)) - f(x)\| \leq \mu$

**Remark**

$\text{tm}$ is not a good measure of complexity.
**Time Scaling**

<table>
<thead>
<tr>
<th>PIVP</th>
<th>$y' = p(y)$</th>
<th>$z(t) = y(e^t)$ → $\begin{cases} z' = up(z) \ u' = u \end{cases}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computed Function</td>
<td>Same</td>
<td></td>
</tr>
<tr>
<td>Time for precision $\mu$</td>
<td>$tm(\mu)$</td>
<td>$tm'(\mu) = \log(tm(\mu))$</td>
</tr>
<tr>
<td>Bounding box for PIVP at time $t$</td>
<td>$sp(t)$</td>
<td>$sp'(t) = \max(sp(e^t), e^t)$</td>
</tr>
</tbody>
</table>

**Example**

\[ sp'(t) = \sup_{\xi \in [1,t]} \|y(\xi)\| \]

\[ sp'(t) = \sup_{\xi \in [1,t]} \|z(\xi), u(\xi)\| \]
### Time Scaling

<table>
<thead>
<tr>
<th>PIVP</th>
<th>$y' = p(y)$</th>
<th>$z(t) = y(e^t)$ → [ \begin{cases} z' = up(z) \ u' = u \end{cases} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computed Function</strong></td>
<td></td>
<td>Same</td>
</tr>
<tr>
<td><strong>Time for precision $\mu$</strong></td>
<td>$tm(\mu)$</td>
<td>$tm'(\mu) = \log(tm(\mu))$</td>
</tr>
<tr>
<td><strong>Bounding box for PIVP at time $t$</strong></td>
<td>$sp(t)$</td>
<td>$sp'(t) = \max(sp(e^t), e^t)$</td>
</tr>
</tbody>
</table>

**Remark**

- $tm(\mu)$ and $sp(t)$ depend on the convergence rate
## Time Scaling

<table>
<thead>
<tr>
<th>PIVP</th>
<th>$y' = p(y)$</th>
<th>$z(t) = y(e^t) \rightarrow \begin{cases} z' = up(z) \ u' = u \end{cases}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computed Function</td>
<td>Same</td>
<td></td>
</tr>
<tr>
<td>Time for precision $\mu$</td>
<td>$tm(\mu)$</td>
<td>$tm'(\mu) = \log(tm(\mu))$</td>
</tr>
<tr>
<td>Bounding box for PIVP at time $t$</td>
<td>$sp(t)$</td>
<td>$sp'(t) = \max(sp(e^t), e^t)$</td>
</tr>
<tr>
<td>Bounding box for PIVP at precision $\mu$</td>
<td>$sp(tm(\mu))$</td>
<td>$\max(sp(tm(\mu)), tm(\mu))$</td>
</tr>
</tbody>
</table>

### Remark
- $tm(\mu)$ and $sp(t)$ depend on the convergence rate
- $sp(tm(\mu))$ seems not
Proper Measures

Proper measures of “complexity”:
- time scaling invariant
- property of the curve
Proper Measures

Proper measures of “complexity”:
- time scaling invariant
- property of the curve

Possible choices:
- Bounding Box at precision $\mu \Rightarrow$ Ok but geometric interpretation?
Proper Measures

Proper measures of “complexity”:
- time scaling invariant
- property of the curve

Possible choices:
- Bounding Box at precision $\mu \Rightarrow$ Ok but geometric interpretation ?
- Length of the curve until precision $\mu \Rightarrow$ Much more intuitive
Complexity based on the length of the curve

Definition

$f$ is **poly-computable** by a GPAC iff $\exists p, q$ polynomials s.t. $\forall x \in \mathbb{R}$, the solution $y = (y_1, \ldots, y_d)$ of:

\[
\begin{align*}
    y'(t) &= p(y(t)) \\
    y(t_0) &= q(x)
\end{align*}
\]

satisfies that $\|f(x) - y_1(t)\| \leq e^{-\mu}$ when $\ell(t) \geq \text{len}(\|x\|, \mu)$ where:

- $\text{len}$ is a polynomial
- $\ell(t)$ is the length of the curve $y$ from 0 to $t$
An equivalent classes

**Definition**

A function $f$ is **poly-computable** by a GPAC iff there exist polynomials $p, q$ such that for all $x \in \mathbb{R}$, the solution $y = (y_1, \ldots, y_d)$ of:

$$\begin{cases}
  y'(t) = p(y(t)) \\
  y(t_0) = q(x)
\end{cases}$$

satisfies that:

- $\|f(x) - y_1(t)\| \leq e^{-\mu}$ when $t \geq \text{poly}(\|x\|, \mu)$
- $\|y(t)\| \leq \text{poly}(\|x\|, t)$
Equivalence theorem

**Theorem**

$f$ is poly-computable if and only if it is computable in polytime in the sense of Computable Analysis.
Conclusion

- Complexity theory for a continuous model of computation
- Natural, machine-independent definition of real computable functions
Complexity theory for a continuous model of computation
Natural, machine-independent definition of real computable functions

Future work:
Other complexity classes (time or space)
Better understand how the restrictions constraint the complexity
Questions?

Do you have any questions?