Semi-infinite programming and applications

A. Ismael F. Vaz

Production and Systems Department
Engineering School
Minho University - Braga - Portugal
aivaz@dps.uminho.pt

with special thanks to Eugénio Ferreira and Edite Fernandes.

Universidade Federal do Rio de Janeiro

12 November 2007
1 Semi-Infinite Programming (SIP) Notation

2 Numerical methods for SIP

3 Some practical applications

4 The particle swarm algorithm

5 Modification of PSOA for multi-local optimization
Outline

1. Semi-Infinite Programming (SIP) Notation
2. Numerical methods for SIP
3. Some practical applications
4. The particle swarm algorithm
5. Modification of PSOA for multi-local optimization
Outline

1. Semi-Infinite Programming (SIP) Notation
2. Numerical methods for SIP
3. Some practical applications
4. The particle swarm algorithm
5. Modification of PSOA for multi-local optimization
Outline

1. Semi-Infinite Programming (SIP) Notation
2. Numerical methods for SIP
3. Some practical applications
4. The particle swarm algorithm
5. Modification of PSOA for multi-local optimization
1. Semi-Infinite Programming (SIP) Notation
2. Numerical methods for SIP
3. Some practical applications
4. The particle swarm algorithm
5. Modification of PSOA for multi-local optimization
Outline

1. Semi-Infinite Programming (SIP) Notation
2. Numerical methods for SIP
3. Some practical applications
4. The particle swarm algorithm
5. Modification of PSOA for multi-local optimization
Semi-Infinite Programming (SIP) Notation

General formulation - Nonlinear semi-infinite programming

Problem

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad f(x) \\
\text{s.t.} & \quad g(x, t) \leq 0 \\
& \quad \forall t \in T
\end{align*}
\]

\[(NLSIP)\]

- \(f(x)\) is the objective function
- \(g(x, t)\) is the infinite constraint function
- \(T \subset \mathbb{R}^p\) is, usually, a cartesian product of intervals
  
  \([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \ldots \times [\alpha_p, \beta_p]\)

Note

A more general problem could be defined, but the extension is straightforward.
General formulation - Nonlinear semi-infinite programming

Problem

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad f(x) \\
\text{s.t.} & \quad g(x, t) \leq 0 \\
& \quad \forall t \in T
\end{align*}
\]

\text{(NLSIP)}

- \( f(x) \) is the objective function
- \( g(x, t) \) is the infinite constraint function
- \( T \subset \mathbb{R}^p \) is, usually, a cartesian product of intervals 
  \( ([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \ldots \times [\alpha_p, \beta_p]) \)

Note

A more general problem could be defined, but the extension is straightforward.
General formulation - Nonlinear semi-infinite programming

Problem

\[ \min_{x \in \mathbb{R}^n} f(x) \]
\[ \text{s.t. } g(x, t) \leq 0 \quad \forall t \in T \]  

(NLSIP)

- \( f(x) \) is the objective function
- \( g(x, t) \) is the infinite constraint function
- \( T \subset \mathbb{R}^p \) is, usually, a cartesian product of intervals \([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \ldots \times [\alpha_p, \beta_p]\)

Note

A more general problem could be defined, but the extension is straightforward.
General formulation - Nonlinear semi-infinite programming

Problem

\[
\min_{x \in \mathbb{R}^n} \ f(x) \\
\text{s.t.} \quad g(x, t) \leq 0 \\
\forall t \in T
\]  

(NLSIP)

- \(f(x)\) is the objective function
- \(g(x, t)\) is the infinite constraint function
- \(T \subset \mathbb{R}^p\) is, usually, a cartesian product of intervals 
  \([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \ldots \times [\alpha_p, \beta_p]\)

Note

A more general problem could be defined, but the extension is straightforward.
General formulation - Nonlinear semi-infinite programming

**Problem**

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad f(x) \\
\text{s.t.} & \quad g(x, t) \leq 0 \\
& \quad \forall t \in T
\end{align*}
\]

(NLSIP)

- \( f(x) \) is the objective function
- \( g(x, t) \) is the infinite constraint function
- \( T \subset \mathbb{R}^p \) is, usually, a cartesian product of intervals
  \[ ([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \ldots \times [\alpha_p, \beta_p]) \]

**Note**

A more general problem could be defined, but the extension is straightforward.
Why semi-infinite programming?

The infinite set $T$ may be viewed as an infinite index set, i.e.,

Index set

$$\begin{align*}
\min_{x \in \mathbb{R}^n} f(x) \\
\text{s.t. } g_t(x) \leq 0 \quad \forall t \in T
\end{align*}$$

(NLSIP)

Semi-infinite

The problem has a finite number of variables subject to an infinite number of constraints.

Practical applications

In practical applications the index set $T$ is related with time or space.
Why semi-infinite programming?

The infinite set $T$ may be viewed as an infinite index set, i.e.,

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad f(x) \\
\text{s.t.} & \quad g_t(x) \leq 0 \quad \forall t \in T
\end{align*}
\]  

(NLSIP)

Semi-infinite

The problem has a finite number of variables subject to an infinite number of constraints.

Practical applications

In practical applications the index set $T$ is related with time or space.
Why semi-infinite programming?

The infinite set $T$ may be viewed as an infinite index set, i.e.,

\[
\min_{x \in \mathbb{R}^n} f(x) \\
\text{s.t. } g_t(x) \leq 0 \quad \forall t \in T
\]  
(NLSIP)

Semi-infinite

The problem has a finite number of variables subject to an infinite number of constraints.

Practical applications

In practical applications the index set $T$ is related with time or space.
An very simple academic example \((n = 1 \text{ and } p = 1)\)

**Example**

\[
\min_{x \in \mathbb{R}} x^2, \quad \text{s.t.} \quad \frac{x}{t} \sin(t) - \frac{x}{10} \leq 0, \quad \forall t \in [2\pi, 10\pi]
\]

\[
g(3, t) = \frac{3}{t} \sin(t) - \frac{3}{10}
\]

Feasibility

Is \(\bar{x} = 3\) feasible?
An very simple academic example \((n = 1 \text{ and } p = 1)\)

**Example**

\[
\min_{x \in \mathbb{R}} x^2, \quad \text{s.t.} \quad \frac{x}{t} \sin(t) - \frac{x}{10} \leq 0, \quad \forall t \in [2\pi, 10\pi]
\]

\[
g(3, t) = \frac{3}{t} \sin(t) - \frac{3}{10}
\]

**Feasibility**

Is \(\bar{x} = 3\) feasible?
Another example – Chebyshev approximation problem

To approximate the function $t^2$ by a combination of $t$ and $e^t$ function in a given set. $d$ is the minimum distance.

$$\min_{p,d \in \mathbb{R}^{2+1}} d$$

s.t.

$$|t^2 - (p_1 t + p_2 e^t)| \leq d$$

$$\forall t \in [0, 2]$$
Definition of stationary point

Let \( x^* \in \mathbb{R}^n \) be a point such that

\[
g(x^*, t) \leq 0, \quad \forall t \in T,
\]

and there exists \( t^1, t^2, \ldots, t^{m^*} (\in T) \) and non negative numbers \( \lambda_0^*, \lambda_1^*, \lambda_2^*, \ldots, \lambda_{m^*}^* \) such that

\[
\lambda_0^* \nabla_x f(x^*) + \sum_{i=1}^{m^*} \lambda_i^* \nabla_x g(x^*, t^i) = 0.
\]

with

\[
g(x^*, t^i) = 0, \quad i = 1, \ldots, m^*.
\]

Then \( x^* \) is a stationary point for the (NLSIP).
Where global (multi-local) optimization plays a role?

The \( t^i, \, i = 1, \ldots, m^*, \) points are global solutions of the problem

\[
\max_{t \in T} g(x^*, t)
\]

- The simple check for feasibility requests the computation of the global solutions for the lower level problem (not completely true).
- In order to obtain global convergence for some methods the computation of all the global and local solutions for the lower level problem is necessary.
Where global (multi-local) optimization plays a role?

The $t^i, i = 1, \ldots, m^*$, points are global solutions of the problem

Multi-local problem (also called lower level problem)

$$\max_{t \in T} g(x^*, t)$$

- The simple check for feasibility requests the computation of the global solutions for the lower level problem (not completely true).
- In order to obtain global convergence for some methods the computation of all the global and local solutions for the lower level problem is necessary.
Where global (multi-local) optimization plays a role?

The $t^i$, $i = 1, \ldots, m^*$, points are global solutions of the problem

**Multi-local problem (also called lower level problem)**

$$\max_{t \in T} g(x^*, t)$$

- The simple check for feasibility requests the computation of the global solutions for the lower level problem (not completely true).
- In order to obtain global convergence for some methods the computation of all the global and local solutions for the lower level problem is necessary.
Outline

1. Semi-Infinite Programming (SIP) Notation
2. Numerical methods for SIP
3. Some practical applications
4. The particle swarm algorithm
5. Modification of PSOA for multi-local optimization
Available numerical methods

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>![ ] Discretization</td>
</tr>
<tr>
<td>![ ] Exchange</td>
</tr>
<tr>
<td>![ ] Reduction</td>
</tr>
<tr>
<td>![ ] Constraints transcription</td>
</tr>
<tr>
<td>![ ] Dual</td>
</tr>
</tbody>
</table>
Available numerical methods

Methods

- Discretization
- Exchange
- Reduction
- Constraints transcription
- Dual
Available numerical methods

Methods

- Discretization
- Exchange
- Reduction
- Constraints transcription
- Dual
Available numerical methods

Methods

- Discretization
- Exchange
- Reduction
- Constraints transcription
- Dual
Available numerical methods

- Discretization
- Exchange
- Reduction
- Constraints transcription
- Dual
Available numerical methods

Methods

- Discretization
- Exchange
- Reduction
- Constraints transcription
- Dual
Discretization methods

First approach

A natural way to address problem (NLSIP) is to consider a discretization of the set $T$ (in an equally spaced grid of points). Usually discretization methods solve a sequence of finite (discretized) problems for successive grid refinements.

Bad properties

- These methods are *Outer approximation* methods and an infeasible solution is usually obtained.
- The problems solution is only obtained when the grid is close to the set $T$.
- High number of constraints to be considered if an accurate solution is requested (ill conditioning can occur).
First approach

A natural way to address problem (NLSIP) is to consider a discretization of the set $T$ (in an equally spaced grid of points). Usually discretization methods solve a sequence of finite (discretized) problems for successive grid refinements.

Bad properties

- These methods are *Outer approximation* methods and an infeasible solution is usually obtained.
- The problems solution is only obtained when the grid is close to the set $T$.
- High number of constraints to be considered if an accurate solution is requested (ill conditioning can occur).
Discretization methods

First approach

A natural way to address problem (NLSIP) is to consider a discretization of the set $T$ (in an equally spaced grid of points).

Usually discretization methods solve a sequence of finite (discretized) problems for successive grid refinements.

Bad properties

- These methods are *Outer approximation* methods and an infeasible solution is usually obtained.
- The problems solution is only obtained when the grid is close to the set $T$.
- High number of constraints to be considered if an accurate solution is requested (ill conditioning can occur).
Discretization methods

First approach
A natural way to address problem (NLSIP) is to consider a discretization of the set $T$ (in an equally spaced grid of points).
Usually discretization methods solve a sequence of finite (discretized) problems for successive grid refinements.

Bad properties
- These methods are *Outer approximation* methods and an infeasible solution is usually obtained.
- The problems solution is only obtained when the grid is close to the set $T$.
- High number of constraints to be considered if an accurate solution is requested (ill conditioning can occur).
Discretization methods

First approach

A natural way to address problem (NLSIP) is to consider a discretization of the set $T$ (in an equally spaced grid of points). Usually discretization methods solve a sequence of finite (discretized) problems for successive grid refinements.

Bad properties

- These methods are *Outer approximation* methods and an infeasible solution is usually obtained.
- The problems solution is only obtained when the grid is *close* to the set $T$.
- High number of constraints to be considered if an accurate solution is requested (ill conditioning can occur).
Discretization methods

Good properties

It is easy to implement and a solver for finite optimization can be used.

Algorithm ($h$ is a grid space parameter)

$S_0$: Define $T[h^0]$. Let $\tilde{T}[h^0] = T[h^0]$. Solve NLP($\tilde{T}[h^0]$) and let $x_0$ be the found solution.

$S_k$: If $x_{k-1}$ is not feasible in $T[h^{k-1}]$

$\quad$ Include all infeasible points into $\tilde{T}[h^{k-1}]$. Solve NLP($\tilde{T}[h^{k-1}]$) and let $x_{k-1}$ be the found solution.

$\quad$ Go to Step $k$.

$\quad$ else:

$\quad$ If the maximum number of refinements is attained then stop.

$\quad$ Otherwise build another set $\tilde{T}[h^k]$ from $T[h^k]$ and $\tilde{T}[h^{k-1}]$. Solve NLP($\tilde{T}[h^k]$) and let $x_k$ be the found solution. Go to Step $k+1$. 
Discretization methods

Good properties
It is easy to implement and a solver for finite optimization can be used.

Algorithm (h is a grid space parameter)

S0: Define \( T[h^0] \). Let \( \tilde{T}[h^0] = T[h^0] \). Solve NLP(\( \tilde{T}[h^0] \)) and let \( x_0 \) be the found solution.

Sk: If \( x_{k-1} \) is not feasible \( \forall t \in T[h^{k-1}] \)

- Include all infeasible points into \( \tilde{T}[h^{k-1}] \). Solve NLP(\( \tilde{T}[h^{k-1}] \)) and let \( x_{k-1} \) be the found solution. Go to Step k.
- Else: If the maximum number of refinements is attained then stop.
- Otherwise build another set \( T[h^k] \) from \( T[h^k] \) and \( \tilde{T}[h^{k-1}] \). Solve NLP(\( \tilde{T}[h^k] \)) and let \( x_k \) be the found solution. Go to Step k + 1.
Discretization methods

Good properties

It is easy to implement and a solver for finite optimization can be used.

Algorithm ($h$ is a grid space parameter)

**S0:** Define $T[h^0]$. Let $\tilde{T}[h^0] = T[h^0]$. Solve NLP($\tilde{T}[h^0]$) and let $x_0$ be the found solution.

**Sk:** If $x_{k-1}$ is not feasible $\forall t \in T[h^{k-1}]$ then: Include all infeasible points into $T[h^{k-1}]$. Solve NLP($\tilde{T}[h^{k-1}]$) and let $x_{k-1}$ be the found solution. Go to Step $k$.

deck{If the maximum number of refinements is attained then stop.
Otherwise build another set $T[h^k]$ from $T[h^{k-1}]$ and $\tilde{T}[h^{k-1}]$. Solve NLP($\tilde{T}[h^k]$) and let $x_k$ be the found solution. Go to Step $k+1$.}
Discretization methods

Good properties

It is easy to implement and a solver for finite optimization can be used.

Algorithm ($h$ is a grid space parameter)

**S0:** Define $T[h^0]$. Let $\tilde{T}[h^0] = T[h^0]$. Solve NLP($\tilde{T}[h^0]$) and let $x_0$ be the found solution.

**Sk:** If $x_{k-1}$ is not feasible $\forall t \in T[h^{k-1}]$

- then: Include all infeasible points into $\tilde{T}[h^{k-1}]$. Solve NLP($\tilde{T}[h^{k-1}]$) and let $x_{k-1}$ be the found solution. Go to Step $k$.
- else: If the maximum number of refinements is attained then stop. Otherwise build another set $\tilde{T}[h^k]$ from $T[h^k]$ and $\tilde{T}[h^{k-1}]$. Solve NLP($\tilde{T}[h^k]$) and let $x_k$ be the found solution. Go to Step $k + 1$. 
Discretization methods

Good properties

It is easy to implement and a solver for finite optimization can be used.

Algorithm ($h$ is a grid space parameter)

\begin{itemize}
  \item [S0:] Define $T[h^0]$. Let $\tilde{T}[h^0] = T[h^0]$. Solve NLP($\tilde{T}[h^0]$) and let $x_0$ be the found solution.
  \item [S_k:] If $x_{k-1}$ is not feasible $\forall t \in T[h^{k-1}]$
    \begin{itemize}
      \item [then:] Include all infeasible points into $\tilde{T}[h^{k-1}]$. Solve NLP($\tilde{T}[h^{k-1}]$) and let $x_{k-1}$ be the found solution. Go to Step $k$.
      \item [else:] If the maximum number of refinements is attained then stop. Otherwise build another set $\tilde{T}[h^k]$ from $T[h^k]$ and $\tilde{T}[h^{k-1}]$. Solve NLP($\tilde{T}[h^k]$) and let $x_k$ be the found solution. Go to Step $k + 1$.
    \end{itemize}
\end{itemize}
Numerical methods for SIP  Discretization methods

Discretization methods

Good properties
It is easy to implement and a solver for finite optimization can be used.

Algorithm (\(h\) is a grid space parameter)

\textbf{S0:} Define \(T[h^0]\). Let \(\tilde{T}[h^0] = T[h^0]\). Solve NLP(\(\tilde{T}[h^0]\)) and let \(x_0\) be the found solution.

\textbf{Sk:} If \(x_{k-1}\) is not feasible \(\forall t \in T[h^{k-1}]\)
then: Include all infeasible points into \(\tilde{T}[h^{k-1}]\). Solve NLP(\(\tilde{T}[h^{k-1}]\)) and let \(x_{k-1}\) be the found solution. \textbf{Go to Step} \(k\).
else: If the maximum number of refinements is attained then \textbf{stop}.
Otherwise build another set \(\tilde{T}[h^k]\) from \(T[h^k]\) and \(\tilde{T}[h^{k-1}]\). Solve NLP(\(\tilde{T}[h^k]\)) and let \(x_k\) be the found solution. \textbf{Go to} \textbf{Step} \(k + 1\).
Exchange methods

In exchange methods approximate solution(s) to the problem (we have as many subproblems as infinite constraints in the (NLSIP)).

Lower level subproblem (multi-local)

$$\max_{t \in T} g(\bar{x}, t)$$

Key idea

The solution(s) (points) of the lower level subproblem are added to a set $\tilde{T}$ while previous added points may be dropped (exchange of points).
A sequence of finite problems is solved considering the set $T$ replaced by the finite set $\tilde{T}$. 
Exchange methods

In exchange methods approximate solution(s) to the problem (we have as many subproblems as infinite constraints in the (NLSIP)).

Lower level subproblem (multi-local)

$$\max_{t \in T} g(\bar{x}, t)$$

Key idea

The solution(s) (points) of the lower level subproblem are added to a set $\tilde{T}$ while previous added points may be dropped (exchange of points). A sequence of finite problems is solved considering the set $T$ replaced by the finite set $\tilde{T}$. 
Properties and algorithm

Bad properties

Slow rate of convergence.

Exchange algorithm

1. Let $T^0 = \emptyset$, $x^0$ be an initial guess and $k = 0$.
2. Approximately solve the lower level subproblem $S_k = \text{arg max}_{t \in T} g(x^k, t)$.
3. If $g(x^k, t) \leq 0$, $\forall t \in S_k$ then stop.
4. Add the new constraints and eventually drop others $(\tilde{T}^{k+1} \subseteq T^k \cup S_k)$.
5. Solve NLP$(\tilde{T}^{k+1})$ and let $x^{k+1}$ be the solution found.
6. Set $k = k + 1$ and go to step 2.
Properties and algorithm

Bad properties
Slow rate of convergence.

Exchange algorithm

1. Let $\tilde{T}^0 = \emptyset$, $x^0$ be an initial guess and $k = 0$.
2. Approximately solve the lower level subproblem $S^k = \arg \max_{t \in T} g(x^k, t)$.
3. If $g(x^k, t) \leq 0$, $\forall t \in S^k$ then stop.
4. Add the new constraints and eventually drop others ($\tilde{T}^{k+1} \subseteq \tilde{T}^k \cup S^k$).
5. Solve NLP($\tilde{T}^{k+1}$) and let $x^{k+1}$ be the solution found.
6. Set $k = k + 1$ and go to step 2.
Properties and algorithm

Bad properties

- Slow rate of convergence.

Exchange algorithm

1. Let $\tilde{T}^0 = \emptyset$, $x^0$ be an initial guess and $k = 0$.

2. Approximately solve the lower level subproblem $S^k = \arg \max_{t \in T} g(x^k, t)$.

3. If $g(x^k, t) \leq 0$, $\forall t \in S^k$ then stop.

4. Add the new constraints and eventually drop others ($\tilde{T}^{k+1} \subseteq \tilde{T}^k \cup S^k$).

5. Solve NLP($\tilde{T}^{k+1}$) and let $x^{k+1}$ be the solution found.

6. Set $k = k + 1$ and go to step 2.
Properties and algorithm

Bad properties

Slow rate of convergence.

Exchange algorithm

1. Let \( \tilde{T}^0 = \emptyset \), \( x^0 \) be an initial guess and \( k = 0 \).
2. Approximately solve the lower level subproblem
   \[ S^k = \arg \max_{t \in T} g(x^k, t). \]
3. If \( g(x^k, t) \leq 0 \), \( \forall t \in S^k \) then stop.
4. Add the new constraints and eventually drop others
   \( (\tilde{T}^{k+1} \subseteq \tilde{T}^k \cup S^k) \).
5. Solve NLP(\( \tilde{T}^{k+1} \)) and let \( x^{k+1} \) be the solution found.
6. Set \( k = k + 1 \) and go to step 2.
Properties and algorithm

Bad properties

Slow rate of convergence.

Exchange algorithm

1. Let $\tilde{T}^0 = \emptyset$, $x^0$ be an initial guess and $k = 0$.
2. Approximately solve the lower level subproblem $S^k = \arg\max_{t \in T} g(x^k, t)$.
3. if $g(x^k, t) \leq 0$, $\forall t \in S^k$ then stop.
4. Add the new constraints and eventually drop others $(\tilde{T}^{k+1} \subseteq \tilde{T}^k \cup S^k)$.
5. Solve NLP(\tilde{T}^{k+1}) and let $x^{k+1}$ be the solution found.
6. Set $k = k + 1$ and go to step 2.
Properties and algorithm

Bad properties

Slow rate of convergence.

Exchange algorithm

1. Let $\tilde{T}^0 = \emptyset$, $x^0$ be an initial guess and $k = 0$.

2. Approximately solve the lower level subproblem $S^k = \arg\max_{t \in T} g(x^k, t)$.

3. if $g(x^k, t) \leq 0$, $\forall t \in S^k$ then stop.

4. Add the new constraints and eventually drop others $(\tilde{T}^{k+1} \subseteq \tilde{T}^k \cup S^k)$.

5. Solve $\text{NLP}(\tilde{T}^{k+1})$ and let $x^{k+1}$ be the solution found.

6. Set $k = k + 1$ and go to step 2.
Properties and algorithm

Bad properties

Slow rate of convergence.

Exchange algorithm

1. Let $\tilde{T}^0 = \emptyset$, $x^0$ be an initial guess and $k = 0$.

2. Approximately solve the lower level subproblem
   \[ S^k = \arg \max_{t \in T} g(x^k, t). \]

3. if $g(x^k, t) \leq 0$, $\forall t \in S^k$ then stop.

4. Add the new constraints and eventually drop others
   \[ (\tilde{T}^{k+1} \subseteq \tilde{T}^k \cup S^k). \]

5. Solve NLP($\tilde{T}^{k+1}$) and let $x^{k+1}$ be the solution found.

6. Set $k = k + 1$ and go to step 2.
Properties and algorithm

Bad properties

Slow rate of convergence.

Exchange algorithm

1. Let $\tilde{T}^0 = \emptyset$, $x^0$ be an initial guess and $k = 0$.
2. Approximately solve the lower level subproblem $S^k = \arg \max_{t \in T} g(x^k, t)$.
3. if $g(x^k, t) \leq 0, \forall t \in S^k$ then stop.
4. Add the new constraints and eventually drop others ($\tilde{T}^{k+1} \subseteq \tilde{T}^k \cup S^k$).
5. Solve NLP($\tilde{T}^{k+1}$) and let $x^{k+1}$ be the solution found.
6. Set $k = k + 1$ and go to step 2.
Reduction type methods

Reduction methods use the more accurate solutions to the subproblem \( \max_{t \in T} g(\bar{x}, t) \), computing all the global solutions and as much as possible the local ones (multi-local optimization).

Bad properties

Obtaining all the global and local maximizer is not an easy task (even for problems with only bound constraints).

Good properties

Good theoretical properties with fast convergence.
Reduction type methods

Reduction methods

Reduction type methods use the more accurate solutions to the subproblem $\max_{t \in T} g(\bar{x}, t)$, computing all the global solutions and as much as possible the local ones (multi-local optimization).

Bad properties

Obtaining all the global and local maximizer is not an easy task (even for problems with only bound constraints).

Good properties

Good theoretical properties with fast convergence.
Reduction type methods

Reduction methods

Reduction type methods use the more accurate solutions to the subproblem \( \max_{t \in T} g(\bar{x}, t) \), computing all the global solutions and as much as possible the local ones (multi-local optimization).

Bad properties

Obtaining all the global and local maximizer is not an easy task (even for problems with only bound constraints).

Good properties

Good theoretical properties with fast convergence.
Numerical methods for SIP

Reduction type methods

Conceptual algorithm

Reduction type algorithm

1. Let $x^0$ be an initial guess and $k = 0$.
2. Obtain all the global and local solutions to the lower level subproblem.
   Let $A^k = \arg \max_{t \in T} g(x^k, t)$.
3. if $g(x^k, t) \leq 0, \forall t \in A^k$ then stop.
4. Solve NLP($A^k$) and let $x^{k+1}$ be the solution found.
5. Set $k = k + 1$ and go to step 2.
Conceptual algorithm

Reduction type algorithm

1. Let $x^0$ be an initial guess and $k = 0$.
2. Obtain all the global and local solutions to the lower level subproblem.
   Let $A_k = \arg \max_{t \in T} g(x^k, t)$.
3. if $g(x^k, t) \leq 0$, $\forall t \in A_k$ then stop.
4. Solve NLP($A_k$) and let $x^{k+1}$ be the solution found.
5. Set $k = k + 1$ and go to step 2.
Conceptual algorithm

Reduction type algorithm

1. Let $x^0$ be an initial guess and $k = 0$.

2. Obtain all the global and local solutions to the lower level subproblem. Let $A^k = \arg \max_{t \in T} g(x^k, t)$.

3. If $g(x^k, t) \leq 0$, $\forall t \in A^k$ then stop.

4. Solve NLP($A^k$) and let $x^{k+1}$ be the solution found.

5. Set $k = k + 1$ and go to step 2.
Conceptual algorithm

Reduction type algorithm

1. Let $x^0$ be an initial guess and $k = 0$.
2. Obtain all the global and local solutions to the lower level subproblem. Let $A^k = \arg \max_{t \in T} g(x^k, t)$.
3. If $g(x^k, t) \leq 0$, $\forall t \in A^k$ then stop.
4. Solve NLP($A^k$) and let $x^{k+1}$ be the solution found.
5. Set $k = k + 1$ and go to step 2.
Conceptual algorithm

Reduction type algorithm

1. Let $x^0$ be an initial guess and $k = 0$.
2. Obtain all the global and local solutions to the lower level subproblem. Let $A^k = \arg \max_{t \in T} g(x^k, t)$.
3. if $g(x^k, t) \leq 0, \forall t \in A^k$ then stop.
4. Solve $\text{NLP}(A^k)$ and let $x^{k+1}$ be the solution found.
5. Set $k = k + 1$ and go to step 2.
Conceptual algorithm

Reduction type algorithm

1. Let $x^0$ be an initial guess and $k = 0$.
2. Obtain all the global and local solutions to the lower level subproblem.
   Let $A^k = \arg \max_{t \in T} g(x^k, t)$.
3. if $g(x^k, t) \leq 0$, $\forall t \in A^k$ then stop.
4. Solve NLP($A^k$) and let $x^{k+1}$ be the solution found.
5. Set $k = k + 1$ and go to step 2.
Constraint transcription

The technique

The constraint transcription is based on the idea that

$$g(\bar{x}, t) \leq 0 \equiv \int_T [g(\bar{x}, t)]_+ dt = 0$$

Used methods

- Regular finite optimization methods can be applied.
  - * Penalty methods
  - * Interior point methods

Bad properties

LICQ is violated.
Constraint transcription

The technique

The constraint transcription is based on the idea that

\[ g(\bar{x}, t) \leq 0 \equiv \int_{T} [g(\bar{x}, t)]_+ dt = 0 \]

Used methods

Regular finite optimization methods can be applied.
- Penalty methods
- Interior point methods

Bad properties

LICQ is violated.
Constraint transcription

The technique

The constraint transcription is based on the idea that

\[ g(\bar{x}, t) \leq 0 \equiv \int_T [g(\bar{x}, t)]_+ dt = 0 \]

Used methods

Regular finite optimization methods can be applied.

- Penalty methods
- Interior point methods

Bad properties

LICQ is violated.
Constraint transcription

The technique

The constraint transcription is based on the idea that

\[ g(\bar{x}, t) \leq 0 \equiv \int_T [g(\bar{x}, t)]_+ dt = 0 \]

Used methods

Regular finite optimization methods can be applied.

- Penalty methods
- Interior point methods

Bad properties

LICQ is violated.
The technique

The constraint transcription is based the idea that

\[ g(\bar{x}, t) \leq 0 \equiv \int_T [g(\bar{x}, t)]_+ dt = 0 \]

Used methods

Regular finite optimization methods can be applied.
- Penalty methods
- Interior point methods

Bad properties

LICQ is violated.
A local quadratic approximation to the (NLSIP) problem is:

\[
\min_{d \in \mathbb{R}^n} f_Q(d) \equiv \frac{1}{2} d^T H^k d + d^T \nabla f(x^k)
\]

s.t. \( d^T \nabla_x g(x^k, t) + g(x^k, t) \leq 0, \quad \forall t \in [a, b] \),

where \( H_k \) is a symmetric positive definite approximation to the Lagrangian Hessian matrix.

The Lagrangian function

\[
\mathcal{L}(d, v) = \frac{1}{2} d^T H^k d + d^T \nabla f(x^k) + \int_a^b \left( d^T \nabla_x g(x^k, t) + g(x^k, t) \right) dv(t)
\]
Dual type methods

A local quadratic approximation to the (NLSIP) problem is:

\[ \min_{d \in \mathbb{R}^n} f_Q(d) \equiv \frac{1}{2} d^T H^k d + d^T \nabla f(x^k) \]

s.t. \( d^T \nabla_x g(x^k, t) + g(x^k, t) \leq 0, \quad \forall t \in [a, b] \),

where \( H_k \) is a symmetric positive definite approximation to the Lagrangian Hessian matrix.

The Lagrangian function

\[ \mathcal{L}(d, v) = \frac{1}{2} d^T H^k d + d^T \nabla f(x^k) + \int_a^b \left( d^T \nabla_x g(x^k, t) + g(x^k, t) \right) dv(t) \]
Solution method

The dual problem $\min L(d, v)$ is solved by approximate the Lagrange multipliers function $v(t)$ by linear segments.

Conceptual algorithm

The local quadratic approximation is used in a sequential quadratic programming (SQP) algorithm.
Dual type methods

Solution method

The dual problem $\min L(d, v)$ is solved by approximate the Lagrange multipliers function $v(t)$ by linear segments.

Conceptual algorithm

The local quadratic approximation is used in a sequential quadratic programming (SQP) algorithm.
Available solvers

Commercial software

MATLAB implements a discretization method in the optimization toolbox (fseminf function).

Public domain software

The Nonlinear Semi-Infinite Programming Solver (NSIPS) is publicly available. It implements:

- A discretization method (two versions).
- A penalty function method (with 5 penalty functions), based on the constraint transcription technique.
- An interior point method, based on the constraint transcription technique.
- A SQP method, based on a dual local quadratic approximation.
Available solvers

Commercial software
MATLAB implements a discretization method in the optimization toolbox (fseminf function).

Public domain software
The Nonlinear Semi-Infinite Programming Solver (NSIPS) is publicly available. It implements:

- A discretization method (two version).
- A penalty function method (with 5 penalty functions), based on the constraint transcription technique.
- An interior point method, based on the constraint transcription technique.
- A SQP method, based on a dual local quadratic approximation.
Available solvers

Commercial software

MATLAB implements a discretization method in the optimization toolbox (fseminf function).

Public domain software

The Nonlinear Semi-Infinite Programming Solver (NSIPPS) is publicly available. It implements:

- A discretization method (two version).
- A penalty function method (with 5 penalty functions), based on the constraint transcription technique.
- An interior point method, based on the constraint transcription technique.
- A SQP method, based on a dual local quadratic approximation.
Available solvers

Commercial software
MATLAB implements a discretization method in the optimization toolbox (fseminf function).

Public domain software
The Nonlinear Semi-Infinite Programming Solver (NSIPS) is publicly available. It implements:

- A discretization method (two version).
- A penalty function method (with 5 penalty functions), based on the constraint transcription technique.
- An interior point method, based on the constraint transcription technique.
- A SQP method, based on a dual local quadratic approximation.
Available solvers

## Commercial software
MATLAB implements a discretization method in the optimization toolbox (`fseminf` function).

## Public domain software
The Nonlinear Semi-Infinite Programming Solver (NSIPS) is publicly available. It implements:
- A discretization method (two version).
- A penalty function method (with 5 penalty functions), based on the constraint transcription technique.
- An interior point method, based on the constraint transcription technique.
- A SQP method, based on a dual local quadratic approximation.
Available solvers

Commercial software
MATLAB implements a discretization method in the optimization toolbox (fseminf function).

Public domain software
The Nonlinear Semi-Infinite Programming Solver (NSIPS) is publicly available. It implements:

- A discretization method (two version).
- A penalty function method (with 5 penalty functions), based on the constraint transcription technique.
- An interior point method, based on the constraint transcription technique.
- A SQP method, based on a dual local quadratic approximation.
Available tools

The SIPAMPL

To provide a database with SIP problems an extension to AMPL was developed. SIPAMPL currently provides:

- A database with over than 160 SIP problems
- An interface between AMPL and MATLAB
- A select tool to allow queries to the database
Available tools

The SIPAMPL

To provide a database with SIP problems an extension to AMPL was developed.

SIPAMPL currently provides:

- A database with over than 160 SIP problems
- An interface between AMPL and MATLAB
- A select tool to allow queries to the database
Available tools

The SIPAMPL

To provide a database with SIP problems an extension to AMPL was developed. SIPAMPL currently provides:

- A database with over than 160 SIP problems
- An interface between AMPL and MATLAB
- A select tool to allow queries to the database
Available tools

The SIPAMPL

To provide a database with SIP problems an extension to AMPL was developed. SIPAMPL currently provides:

- A database with over than 160 SIP problems
- An interface between AMPL and MATLAB
- A select tool to allow queries to the database
Outline

1. Semi-Infinite Programming (SIP) Notation
2. Numerical methods for SIP
3. Some practical applications
4. The particle swarm algorithm
5. Modification of PSOA for multi-local optimization
Practical application I

Fed-batch fermentation process
Fed-batch fermentation process

A great number of valuable products are produced using fermentation processes and thus optimizing such processes is of great economic importance.

Fermentation modeling process involves, in general, highly nonlinear and complex differential equations.

Often optimizing these processes results in control optimization problems for which an analytical solution is not possible.
Fed-batch fermentation process

- A great number of valuable products are produced using fermentation processes and thus optimizing such processes is of great economic importance.
- Fermentation modeling process involves, in general, highly nonlinear and complex differential equations.
- Often optimizing these processes results in control optimization problems for which an analytical solution is not possible.
Fed-batch fermentation process

- A great number of valuable products are produced using fermentation processes and thus optimizing such processes is of great economic importance.
- Fermentation modeling process involves, in general, highly nonlinear and complex differential equations.
- Often optimizing these processes results in control optimization problems for which an analytical solution is not possible.
The control problem

The optimal control problem is described by a set of differential equations
\[ \dot{\chi} = h(\chi, u, t), \quad \chi(t^0) = \chi^0, \quad t^0 \leq t \leq t^f, \]
where \( \chi \) represent the state variables and \( u \) the control variables.

The performance index \( J \) can be generally stated as

\[ J(t^f) = \varphi(\chi(t^f), t^f) + \int_{t^0}^{t^f} \phi(\chi, u, t) dt, \]

where \( \varphi \) is the performance index of the state variables at final time \( t^f \) and \( \phi \) is the integrated performance index during the operation.

Additional constraints that often reflect some physical limitation of the system can be imposed.
The control problem

The optimal control problem is described by a set of differential equations
\[ \dot{\chi} = h(\chi, u, t), \quad \chi(t_0) = \chi^0, \quad t_0 \leq t \leq t_f, \]
where \( \chi \) represent the state variables and \( u \) the control variables.

The performance index \( J \) can be generally stated as

\[ J(t_f) = \varphi(\chi(t_f), t_f) + \int_{t_0}^{t_f} \phi(\chi, u, t) dt, \]

where \( \varphi \) is the performance index of the state variables at final time \( t_f \)
and \( \phi \) is the integrated performance index during the operation.

Additional constraints that often reflect some physical limitation of the system can be imposed.
The control problem

- The optimal control problem is described by a set of differential equations \( \dot{x} = h(x, u, t) \), \( x(t^0) = x^0 \), \( t^0 \leq t \leq t^f \), where \( x \) represent the state variables and \( u \) the control variables.

- The performance index \( J \) can be generally stated as

\[
J(t^f) = \varphi(x(t^f), t^f) + \int_{t^0}^{t^f} \phi(x, u, t) \, dt,
\]

where \( \varphi \) is the performance index of the state variables at final time \( t^f \) and \( \phi \) is the integrated performance index during the operation.

- Additional constraints that often reflect some physical limitation of the system can be imposed.
The control problem

The general maximization problem \((P)\) can be posed as

\[
\begin{align*}
\text{max} & \quad J(t_f) \\
\text{s.t.} & \quad \dot{x} = h(x, u, t) \\
& \quad x \leq x(t) \leq \bar{x}, \\
& \quad u \leq u(t) \leq \bar{u}, \\
& \quad \forall t \in [t_0, t_f]
\end{align*}
\]

Where the state constraints (3) and control constraints (4) are to be understood as componentwise inequalities.

How we addressed problem \((P)\)?
The control problem

The general maximization problem \((P)\) can be posed as

\[
\begin{align*}
\text{max} & \quad J(t^f) \\
\text{s.t.} & \quad \dot{\chi} = h(\chi, u, t) \\
& \quad \underline{\chi} \leq \chi(t) \leq \bar{\chi}, \\
& \quad \underline{u} \leq u(t) \leq \bar{u}, \\
& \quad \forall t \in [t^0, t^f]
\end{align*}
\]

Where the state constraints (3) and control constraints (4) are to be understood as componentwise inequalities.

How we addressed problem \((P)\)?
**Approaches** - Fed trajectory $u(t)$ approximated by a Linear spline $w(t)$.

- Penalty function for state constraints
- The multi-local (getting all local optima) problem is easy to solve

**Objective function**

\[
\hat{J}(t_f) = \begin{cases} 
J(t_f) & \text{if } \chi \leq \chi(t) \leq \bar{\chi}, \\
\forall t \in [t^0, t_f] 
-\infty & \text{otherwise}
\end{cases}
\]

**State constraints**

\[ u \leq w(t^i) \leq \bar{u}, \quad i = 1, \ldots, n \]

Where $t^i$ are the spline knots.

The maximization NLP problem is

\[
\max_{w(t^i)} \hat{J}(t_f), \quad \text{s.t. } u \leq w(t^i) \leq \bar{u}, \quad i = 1, \ldots, n
\]
Some practical applications
Fed-batch fermentation process

Approaches - Fed trajectory $u(t)$ approximated by a Linear spline $w(t)$.

- Penalty function for state constraints
- The multi-local (getting all local optima) problem is easy to solve

### Objective function

$$
\hat{J}(t^f) = \begin{cases} 
J(t^f) & \text{if } \chi \leq \chi(t) \leq \bar{\chi}, \\
-\infty & \text{otherwise} 
\end{cases}
$$

### State constraints

$$u \leq w(t^i) \leq \bar{u}, \quad i = 1, \ldots, n$$

Where $t^i$ are the spline knots.

The maximization NLP problem is

$$\max \hat{J}(t^f), \quad \text{s.t. } u \leq w(t^i) \leq \bar{u}, \quad i = 1, \ldots, n$$
Some practical applications

Fed-batch fermentation process

**Approaches** - Fed trajectory $u(t)$ approximated by a Linear spline $w(t)$.

- Penalty function for state constraints
- The multi-local (getting all local optima) problem is easy to solve

<table>
<thead>
<tr>
<th>Objective function</th>
<th>State constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{J}(t_f) = \begin{cases} J(t_f) &amp; \text{if } \chi \leq \chi(t) \leq \bar{\chi}, \ -\infty &amp; \text{otherwise} \end{cases}$</td>
<td>$u \leq w(t^i) \leq \bar{u}, \ i = 1, \ldots, n$</td>
</tr>
</tbody>
</table>

Where $t^i$ are the spline knots.

The maximization NLP problem is

$$\max_{w(t^i)} \hat{J}(t_f), \ s.t. \ u \leq w(t^i) \leq \bar{u}, \ i = 1, \ldots, n$$
Approaches - Fed trajectory $u(t)$ approximated by a Cubic spline $s(t)$.

- Penalty function for state constraints
- The multi-local (getting all local optima) problem is hard to solve
- No off-the-shelf software to address this problem
- A new penalty function defined for control constraints

**Objective function**

\[
\hat{J}(t^f) = \begin{cases} 
  J(t^f) & \text{if } \chi \leq \chi(t) \leq \chi, \\
  -\infty & \text{otherwise}
\end{cases} \quad \forall t \in [t^0, t^f]
\]

**New objective function**

\[
\bar{J}(t^f) = \begin{cases} 
  \hat{J}(t^f) & \text{if } u \leq w(t) \leq \bar{u}, \\
  -\infty & \text{otherwise}
\end{cases} \quad \forall t \in [t^0, t^f]
\]
Some practical applications

Fed-batch fermentation process

Approaches - Fed trajectory \( u(t) \) approximated by a Cubic spline \( s(t) \).

- Penalty function for state constraints
- The multi-local (getting all local optima) problem is hard to solve
- No off-the-shelf software to address this problem
- A new penalty function defined for control constraints

Objective function

\[
\hat{J}(t^f) = \begin{cases} 
J(t^f) & \text{if } \chi \leq \chi(t) \leq \overline{\chi}, \\
\forall t \in [t^0, t^f] \\
-\infty & \text{otherwise}
\end{cases}
\]

New objective function

\[
\bar{J}(t^f) = \begin{cases} 
\hat{J}(t^f) & \text{if } u \leq w(t) \leq \overline{u}, \\
\forall t \in [t^0, t^f] \\
-\infty & \text{otherwise}
\end{cases}
\]
Approaches - Fed trajectory $u(t)$ approximated by a Cubic spline $s(t)$.

- Penalty function for state constraints
- The multi-local (getting all local optima) problem is hard to solve
- No off-the-shelf software to address this problem
- A new penalty function defined for control constraints

### Objective function

$$\hat{J}(t^f) = \begin{cases} 
J(t^f) & \text{if } \chi \leq \chi(t) \leq \chi, \\
& \forall t \in [t^0, t^f] \\
-\infty & \text{otherwise}
\end{cases}$$

### New objective function

$$\bar{J}(t^f) = \begin{cases} 
\hat{J}(t^f) & \text{if } u \leq w(t) \leq \bar{u}, \\
& \forall t \in [t^0, t^f] \\
-\infty & \text{otherwise}
\end{cases}$$
Some practical applications

Fed-batch fermentation process

Approaches - Fed trajectory $u(t)$ approximated by a Cubic spline $s(t)$.

- Penalty function for state constraints
- The multi-local (getting all local optima) problem is hard to solve
- No off-the-shelf software to address this problem
- A new penalty function defined for control constraints

**Objective function**

$$\hat{J}(t_f) = \begin{cases} 
J(t_f) & \text{if } \chi \leq \chi(t) \leq \bar{\chi}, \\
-\infty & \text{otherwise} 
\end{cases} \quad \forall t \in [t_0, t_f]$$

**New objective function**

$$\bar{J}(t_f) = \begin{cases} 
\hat{J}(t_f) & \text{if } u \leq w(t) \leq \bar{u}, \\
-\infty & \text{otherwise} 
\end{cases} \quad \forall t \in [t_0, t_f]$$
Implementation details

- The AMPL modeling language:
  - was used to model five optimal control problems
  - dynamic external library facility was used to solve the ordinary differentiable equations

  AMPL - A Modeling Programming Language
  www.ampl.com

- The ordinary differentiable equations were solved using the CVODE software package.

  http://www.llnl.gov/casc/sundials/

- A stochastic algorithm based on particle swarm was used to solve the non-differentiable optimization problem. We address this algorithm later on.
Implementation details

- The AMPL modeling language:
  - was used to model five optimal control problems
  - dynamic external library facility was used to solve the ordinary differentiable equations

  AMPL - A Modeling Programming Language
  www.ampl.com

- The ordinary differentiable equations were solved using the CVODE software package.
  http://www.llnl.gov/casc/sundials/

- A stochastic algorithm based on particle swarm was used to solve the non-differentiable optimization problem. We address this algorithm later on.
Implementation details

- The AMPL modeling language:
  - was used to model five optimal control problems
  - dynamic external library facility was used to solve the ordinary differentiable equations

  AMPL - A Modeling Programming Language
  www.ampl.com

- The ordinary differentiable equations were solved using the CVODE software package.

  http://www.llnl.gov/casc/sundials/

- A stochastic algorithm based on particle swarm was used to solve the non-differentiable optimization problem. We address this algorithm later on.
Implementation details

- The AMPL modeling language:
  - was used to model five optimal control problems
  - dynamic external library facility was used to solve the ordinary differentiable equations

AMPL - A Modeling Programming Language
www.ampl.com

- The ordinary differentiable equations were solved using the CVODE software package.

http://www.llnl.gov/casc/sundials/

- A stochastic algorithm based on particle swarm was used to solve the non-differentiable optimization problem. We address this algorithm later on.
Implementation details

- The AMPL modeling language:
  - was used to model five optimal control problems
  - dynamic external library facility was used to solve the ordinary differentiable equations

  AMPL - A Modeling Programming Language  
  www.ampl.com

- The ordinary differentiable equations were solved using the CVODE software package.

  http://www.llnl.gov/casc/sundials/

- A stochastic algorithm based on particle swarm was used to solve the non-differentiable optimization problem. We address this algorithm later on.
The problems set

We obtained numerical results for five case studies.

Problem

- Penicillin refers to a problem of fed-batch fermentation process where the optimal feed trajectory is to be computed while the penicillin production is to be maximized.

- Ethanol refers to a similar optimal control problem where the ethanol production is to be maximized.

- Chemotherapy is the only optimal control problem that does not refer to a fed-batch fermentation process. It is a problem of drug administration in chemotherapy. The optimal trajectory to be computed is the quantity of drug that must be present in order to achieve a specified tumor reduction.

- Protein optimal control problem is to compute a unique trajectory (substrate to be fed) problem. This process includes also a trajectory for an inducer. Both problems refer to a maximization for protein production.
The problems set

- We obtained numerical results for five case studies.

**Problem**

- **penicillin** refers to a problem of fed-batch fermentation process where the optimal feed trajectory is to be computed while the penicillin production is to be maximized.

- **ethanol** refers to a similar optimal control problem where the ethanol production is to be maximized.

- **chemotherapy** is the only optimal control problem that does not refer to a fed-batch fermentation process. It is a problem of drug administration in chemotherapy. The optimal trajectory to be computed is the quantity of drug that must be present in order to achieve a specified tumor reduction.

- **hprotein** optimal control problem is to compute a unique trajectory (substrate to be fed) problem. **rprotein** includes also a trajectory for an inducer. Both problems refer to a maximization for protein production.
The problems set

- We obtained numerical results for five case studies.

- Problem
  - penicillin refers to a problem of fed-batch fermentation process where the optimal feed trajectory is to be computed while the penicillin production is to be maximized.
  - ethanol refers to a similar optimal control problem where the ethanol production is to be maximized.
  - chemotherapy is the only optimal control problem that does not refer to a fed-batch fermentation process. It is a problem of drug administration in chemotherapy. The optimal trajectory to be computed is the quantity of drug that must be present in order to achieve a specified tumor reduction.
  - hprotein optimal control problem is to compute a unique trajectory (substrate to be fed) problem hprotein includes also a trajectory for an inducer. Both problems refer to a maximization for protein production.
The problems set

- We obtained numerical results for five case studies.

**Problem**

- *penicillin* refers to a problem of fed-batch fermentation process where the optimal feed trajectory is to be computed while the penicillin production is to be maximized.

- *ethanol* refers to a similar optimal control problem where the ethanol production is to be maximized.

- *chemotherapy* is the only optimal control problem that does not refer to a fed-batch fermentation process. It is a problem of drug administration in chemotherapy. The optimal trajectory to be computed is the quantity of drug that must be present in order to achieve a specified tumor reduction.

- *hprotein optimal control problem is to compute a unique trajectory (substrate to be fed) problem rprotein includes also a trajectory for an inducer. Both problems refer to a maximization for protein production.*
The problems set

- We obtained numerical results for five case studies.
- **Problem**
  - Penicillin refers to a problem of fed-batch fermentation process where the optimal feed trajectory is to be computed while the penicillin production is to be maximized.
  - Ethanol refers to a similar optimal control problem where the ethanol production is to be maximized.
  - Chemotherapy is the only optimal control problem that does not refer to a fed-batch fermentation process. It is a problem of drug administration in chemotherapy. The optimal trajectory to be computed is the quantity of drug that must be present in order to achieve a specified tumor reduction.
  - Hprotein optimal control problem is to compute a unique trajectory (substrate to be fed) problem. Hprotein includes also a trajectory for an inducer. Both problems refer to a maximization for protein production.
Some practical applications
Fed-batch fermentation process

The problems set

- We obtained numerical results for five case studies.
- Problem
  - **penicillin** refers to a problem of fed-batch fermentation process where the optimal feed trajectory is to be computed while the penicillin production is to be maximized.
  - **ethanol** refers to a similar optimal control problem where the ethanol production is to be maximized.
  - **chemotherapy** is the only optimal control problem that does not refer to a fed-batch fermentation process. It is a problem of drug administration in chemotherapy. The optimal trajectory to be computed is the quantity of drug that must be present in order to achieve a specified tumor reduction.
  - **hprotein** optimal control problem is to compute a unique trajectory (substrate to be fed) problem. **rprotein** includes also a trajectory for an inducer. Both problems refer to a maximization for protein production.
Characteristics and parameters

- The time displacement ($h_i$) are fixed while the optimal trajectory values are to be approximated.
- Particle swarm is a population based optimization algorithm and a population size of 60 was used with a maximum of 1000 iterations.
- Since a stochastic algorithm was used we performed 10 runs of the solver and the best solution is reported.
Characteristics and parameters

- The time displacement ($h_i$) are fixed while the optimal trajectory values are to be approximated.
- Particle swarm is a population based optimization algorithm and a population size of 60 was used with a maximum of 1000 iterations.
- Since a stochastic algorithm was used we performed 10 runs of the solver and the best solution is reported.
Characteristics and parameters

- The time displacement \( h_i \) are fixed while the optimal trajectory values are to be approximated.
- Particle swarm is a population based optimization algorithm and a population size of 60 was used with a maximum of 1000 iterations.
- Since a stochastic algorithm was used we performed 10 runs of the solver and the best solution is reported.
Numerical results

<table>
<thead>
<tr>
<th>Problema</th>
<th>NT</th>
<th>n</th>
<th>$t_f$</th>
<th>Cubic</th>
<th>Linear</th>
<th>Literature</th>
</tr>
</thead>
<tbody>
<tr>
<td>penicillin</td>
<td>1</td>
<td>5</td>
<td>132.00</td>
<td>87.70</td>
<td>88.29</td>
<td>87.99</td>
</tr>
<tr>
<td>ethanol</td>
<td>1</td>
<td>5</td>
<td>61.20</td>
<td>20550.70</td>
<td>20379.50</td>
<td>20839.00</td>
</tr>
<tr>
<td>chemotherapy</td>
<td>1</td>
<td>4</td>
<td>84.00</td>
<td>15.75</td>
<td>16.83</td>
<td>14.48</td>
</tr>
<tr>
<td>hprotein</td>
<td>1</td>
<td>5</td>
<td>15.00</td>
<td>38.86</td>
<td>32.73</td>
<td>32.40</td>
</tr>
<tr>
<td>rprotein</td>
<td>2</td>
<td>5</td>
<td>10.00</td>
<td>0.13</td>
<td>0.12</td>
<td>0.16</td>
</tr>
</tbody>
</table>

$$J(t_f) = \hat{J}(t_f) = \bar{J}(t_f), \quad \text{for all feasible points - splines}$$

Similar results between approaches. A new solution for the ethanol case.
Some practical applications
Fed-batch fermentation process

Plots - Linear spline approximation - ethanol case

Control profile

State profile

- $X_1$ - Cell mass
- $X_2$ - Substrate
- $X_3$ - Product
- $X_4$ - Volume

Ismael Vaz (UMinho - PT)
Some practical applications

Fed-batch fermentation process

Plots - Cubic spline approximation - Similar result

Control profile

State profile

X1 − Cell mass
X2 − Substrate
X3 − Product
X4 − Volume
Plots - Cubic spline approximation - Best result

Control profile

State profile

- $X_1$ – Cell mass
- $X_2$ – Substrate
- $X_3$ – Product
- $X_4$ – Volume
Practical application II

Robot trajectory planning
Robot trajectory definition

Notions:
- Links
- Joints
- Degree of freedom (d.o.f.)

Example

Two links ($d_i$ - length, $m_i$ - mass), three d.o.f. ($\theta_1$ - rotation about $Y$, $\theta_2$ - aperture of link 1, $\theta_3$ - aperture of link 2)
Robot trajectory definition

Notions:
- Links
- Joints
- Degree of freedom (d.o.f.)

Example:
Two links ($d_i$ - length, $m_i$ - mass), three d.o.f. ($\theta_1$ - rotation about $Y$, $\theta_2$ - aperture of link 1, $\theta_3$ - aperture of link 2)
Robot trajectory definition

Notions:
- Links
- Joints
- Degree of freedom (d.o.f.)

Example

Two links ($d_i$ - length, $m_i$ - mass), three d.o.f. ($\theta_1$ - rotation about $Y$, $\theta_2$ - aperture of link 1, $\theta_3$ - aperture of link 2)
Robot trajectory definition

Notions:
- Links
- Joints
- Degree of freedom (d.o.f.)

Example

Two links ($d_i$ - length, $m_i$ - mass), three d.o.f. ($\theta_1$ - rotation about $Y$, $\theta_2$ - aperture of link 1, $\theta_3$ - aperture of link 2)
Robot trajectory definition

Notions:
- Links
- Joints
- Degree of freedom (d.o.f.)

Example
Two links ($d_i$ - length, $m_i$ - mass), three d.o.f. ($\theta_1$ - rotation about $Y$, $\theta_2$ - aperture of link 1, $\theta_3$ - aperture of link 2)
Ways to defining a trajectory

In Cartesian space

\[
(x(\tau), y(\tau), z(\tau), p(\tau))^T
\]
Ways to defining a trajectory

In Joint space

$$\theta(\tau) = (\theta_1(\tau), \theta_2(\tau), \theta_3(\tau))^T$$
Optimize trajectory

We can optimize the trajectory for:

- **Minimum trajectory time**
- Minimum energy consumption
- Maximum load capacity
Optimize trajectory

We can optimize the trajectory for:

- **Minimum trajectory time**
- **Minimum energy consumption**
- **Maximum load capacity**
Optimize trajectory

We can optimize the trajectory for:

- **Minimum trajectory time**

- **Minimum energy consumption**

- **Maximum load capacity**
Robot limitations

- Maximum velocity in each joint \( \left| \frac{d\theta_i(\tau)}{d\tau} \right| \leq C_{i,1}, \ i = 1, \ldots, l \)
- Maximum acceleration in each joint \( \left| \frac{d^2\theta_i(\tau)}{d\tau^2} \right| \leq C_{i,2}, \ i = 1, \ldots, l \)
- Maximum jerk in each joint \( \left| \frac{d^3\theta_i(\tau)}{d\tau^3} \right| \leq C_{i,3}, \ i = 1, \ldots, l \)

or

- Maximum joint torque in each joint

\[ |F_i(\tau)| \leq C_i, \ \tau \in [0, \tau_f], \ i = 1, \ldots, l \]
Robot limitations

- Maximum velocity in each joint \( \left| \frac{d\theta_i(\tau)}{d\tau} \right| \leq C_i,1, \ i = 1, \ldots, l \)

- Maximum acceleration in each joint \( \left| \frac{d^2\theta_i(\tau)}{d\tau^2} \right| \leq C_i,2, \ i = 1, \ldots, l \)

- Maximum jerk in each joint \( \left| \frac{d^3\theta_i(\tau)}{d\tau^3} \right| \leq C_i,3, \ i = 1, \ldots, l \)

or

- Maximum joint torque in each joint \( |F_i(\tau)| \leq C_i, \ \tau \in [0, \tau_f], \ i = 1, \ldots, l \)
Robot limitations

- Maximum velocity in each joint \( \left| \frac{d\theta_i(\tau)}{d\tau} \right| \leq C_{i,1}, \ i = 1, \ldots, l \)

- Maximum acceleration in each joint \( \left| \frac{d^2\theta_i(\tau)}{d\tau^2} \right| \leq C_{i,2}, \ i = 1, \ldots, l \)

- Maximum jerk in each joint \( \left| \frac{d^3\theta_i(\tau)}{d\tau^3} \right| \leq C_{i,3}, \ i = 1, \ldots, l \)

or

- Maximum joint torque in each joint

\[
|F_i(\tau)| \leq C_{i}, \ \tau \in [0, \tau_f], \ i = 1, \ldots, l
\]
Some practical applications

Robot trajectory planning

Robot limitations

- Maximum velocity in each joint: $\left| \frac{d\theta_i(\tau)}{d\tau} \right| \leq C_{i,1}, \; i = 1, \ldots, l$
- Maximum acceleration in each joint: $\left| \frac{d^2\theta_i(\tau)}{d\tau^2} \right| \leq C_{i,2}, \; i = 1, \ldots, l$
- Maximum jerk in each joint: $\left| \frac{d^3\theta_i(\tau)}{d\tau^3} \right| \leq C_{i,3}, \; i = 1, \ldots, l$

or

- Maximum joint torque in each joint: $|F_i(\tau)| \leq C_i, \; \tau \in [0, \tau_f], \; i = 1, \ldots, l$
Trajectory limitations

- Robot is in movement

\[
\sum_{i=1}^{l} \left( \frac{d\theta_i}{d\tau} \right)^2 > 0, \quad \tau \in (0, \tau_f)
\]

- except in initial and end positions

\[
\frac{d\theta}{d\tau}(0) = \frac{d\theta}{d\tau}(\tau_f) = 0
\]

- Acceleration in initial and end positions should not be zero

\[
\frac{d^2\theta}{d\tau^2}(0), \quad \frac{d^2\theta}{d\tau^2}(\tau_f) \neq 0
\]
Some practical applications

Robot trajectory planning

Trajectory limitations

- Robot is in movement

\[ \sum_{i=1}^{l} \left( \frac{d\theta_i}{d\tau} \right)^2 > 0, \quad \tau \in (0, \tau_f) \]

- except in initial and end positions

\[ \frac{d\theta}{d\tau}(0) = \frac{d\theta}{d\tau}(\tau_f) = 0 \]

- Acceleration in initial and end positions should not be zero

\[ \frac{d^2\theta}{d\tau^2}(0), \quad \frac{d^2\theta}{d\tau^2}(\tau_f) \neq 0 \]
Trajectory limitations

- Robot is in movement

\[ \sum_{i=1}^{l} \left( \frac{d\theta_i}{d\tau} \right)^2 > 0, \quad \tau \in (0, \tau_f) \]

- except in initial and end positions

\[ \frac{d\theta}{d\tau}(0) = \frac{d\theta}{d\tau}(\tau_f) = 0 \]

- Acceleration in initial and end positions should not be zero

\[ \frac{d^2\theta}{d\tau^2}(0), \quad \frac{d^2\theta}{d\tau^2}(\tau_f) \neq 0 \]
Optimal cubic polynomial joint trajectories

Given a set of via points defining a trajectory

Assume that \([\theta_1(\tau_0), \ldots, \theta_1(\tau_n)], [\theta_2(\tau_0), \ldots, \theta_2(\tau_n)], \ldots, [\theta_l(\tau_0), \ldots, \theta_l(\tau_n)]\) are the vectors of points (knots) where the joint trajectory passes through.

Find the best trajectory

The optimization consists of finding the optimum total displacements time that fits the joint trajectory by using cubic splines constrained to velocity, acceleration, jerk and torque bounds.
Optimal cubic polynomial joint trajectories

Given a set of via points defining a trajectory

Assume that $[\theta_1(\tau_0), \ldots, \theta_1(\tau_n)], [\theta_2(\tau_0), \ldots, \theta_2(\tau_n)], \ldots, [\theta_l(\tau_0), \ldots, \theta_l(\tau_n)]$ are the vectors of points (knots) where the joint trajectory passes through.

Find the best trajectory

The optimization consists of finding the optimum total displacements time that fits the joint trajectory by using cubic splines constrained to velocity, acceleration, jerk and torque bounds.
Additional notation

Let

* $t_0 < t_1 < \cdots < t_n$ be a time sequence where $t_i$ is the time where the robot is in the joint position $[\theta_1(\tau_i), \ldots, \theta_l(\tau_i)]$

* $h_1 = t_1 - t_0$, $h_2 = t_2 - t_1$, \ldots, $h_n = t_n - t_{n-1}$ be the time displacements

* $Q_{ij}(t)$ be the cubic spline for joint $i$ in $[t_{j-1}, t_j]$ and $Q_i(t)$ be the cubic spline for joint $i$.

We will use the notation $Q'(t) = \frac{dQ(t)}{dt}$ for the derivative.
Additional notation

Let

- $t_0 < t_1 < \cdots < t_n$ be a time sequence where $t_i$ is the time where the robot is in the joint position $[\theta_1(\tau_i), \ldots, \theta_l(\tau_i)]$
- $h_1 = t_1 - t_0, h_2 = t_2 - t_1, \ldots, h_n = t_n - t_{n-1}$ be the time displacements
- $Q_{ij}(t)$ be the cubic spline for joint $i$ in $[t_{j-1}, t_j]$ and $Q_i(t)$ be the cubic spline for joint $i$.

We will use the notation $Q'(t) = \frac{dQ(t)}{dt}$ for the derivative.
Additional notation

Let

1. \( t_0 < t_1 < \cdots < t_n \) be a time sequence where \( t_i \) is the time where the robot is in the joint position \([\theta_1(\tau_i), \ldots, \theta_l(\tau_i)]\)
2. \( h_1 = t_1 - t_0, h_2 = t_2 - t_1, \ldots, h_n = t_n - t_{n-1} \) be the time displacements
3. \( Q_{ij}(t) \) be the cubic spline for joint \( i \) in \([t_{j-1}, t_j]\) and \( Q_i(t) \) be the cubic spline for joint \( i \).

We will use the notation \( Q'(t) = \frac{dQ(t)}{dt} \) for the derivative.
Generalized SIP

The SIP problem can be formulated in the following mathematical form:

$$\min \sum_{j=1}^{n} h_j \equiv t_n - t_0$$

s.t.  
$$|Q_i'(t)| \leq C_{i,1}$$  
$$|Q_i''(t)| \leq C_{i,2}$$  
$$|Q_i'''(t)| \leq C_{i,3}$$  
$$|F_i(t)| \leq C_i, \quad i = 1, \ldots, l$$
$$h_j > 0 \quad j = 1, \ldots, n;$$
$$\forall t \in [t_0, t_n]$$

where $C_{i,1}$, $C_{i,2}$, $C_{i,3}$ and $C_i$ are the bounds for the velocity, acceleration, jerk and torque, respectively, on joint $i$. 
Torque expression

The expression for the manipulator’s torque is

\[
F_i(t) = J_i n_i Q''_i(t) + B_i n_i Q'_i(t) + \frac{1}{n_i} \left( \sum_{j=1}^{l} I_{ij}(Q(t)) Q''_j(t) \\
+ \sum_{j=1}^{l} \sum_{k=1}^{l} C_{ijk}(Q(t)) Q'_j(t) Q'_k(t) + d_i(Q(t)) \right)
\]

where for the \( i \)th robot joint
Torque expression (cont.)

\[ J_i = \text{motor inertia} \ (J_i > 0, \ i = 1, \ldots, l); \]
\[ n_i = \text{gear ratio}; \]
\[ B_i = \text{viscous damping coefficient} \ (B_i > 0, \ i = 1, \ldots, l); \]
\[ (I_{ij}(Q(t)))_{i,j=1,...,l} = \text{inertia matrix (positive definite)}; \]
\[ (C_{ijk}(Q(t)))_{i,j,k=1,...,l} = \text{Coriolis tensor}; \]
\[ d_i(Q(t)) = \text{gravitational torque}. \]
Some practical applications

Robot trajectory planning

Reformulation as standard SIP

\[
\begin{align*}
\min \sum_{j=1}^{n} h_j \\
\text{s.t.} \quad & Q'_i \left( \tau \sum_{k=1}^{n} h_k + t_0 \right) \leq C_{i,1} \\
& Q''_i \left( \tau \sum_{k=1}^{n} h_k + t_0 \right) \leq C_{i,2} \\
& Q'''_i \left( \tau \sum_{k=1}^{n} h_k + t_0 \right) \leq C_{i,3} \\
& F_i \left( \tau \sum_{k=1}^{n} h_k + t_0 \right) \leq C_{i}, \quad i = 1, \ldots, l
\end{align*}
\]

\[ h_j > 0, \quad j = 1, \ldots, n, \quad \forall \tau \in [0, 1]. \]

using the linear transformation
\[
t = \tau \sum_{k=1}^{n} h_k + t_0.
\]
Some practical applications
Robot trajectory planning

Some results with problems available at SIPAMPL

Plot - Joint 5 for problem lin2
Some more results

Plot - Joint 1 for problem deluca1
Some practical applications

Robot trajectory planning

Optimal parametrization of curves for robot joint trajectories

Another trajectory optimization problem

The robot trajectory is known \( (\theta_i(\tau)) \) and a parametrization \( (t = h(\tau)) \) is to be computed.

Find a parametrization

\[
t = h(\tau), \quad \tau \in [0, 1] \quad \tau_f = 1
\]

where \( \theta^*_i(t) = \theta_i(h^{-1}(t)) \), such that

<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h(0) = 0 )</td>
<td>( h(0) = 0 )</td>
</tr>
<tr>
<td>( h(1) ) is minimum</td>
<td>( h(1) ) is minimum</td>
</tr>
<tr>
<td>( h'(\tau) &gt; 0, \quad \tau \in [0, 1] )</td>
<td>( h'(\tau) &gt; 0, \quad \tau \in [0, 1] )</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\left| \frac{d\theta^*_i(t)}{dt} \right| & \leq C_{i,1} \\
\left| \frac{d^2\theta^*_i(t)}{dt^2} \right| & \leq C_{i,2} \\
\left| \frac{d^3\theta^*_i(t)}{dt^3} \right| & \leq C_{i,3}
\end{align*}
\]

\( |F_i(t)| \leq C_i \)

\( i = 1, \ldots, l \)
Optimal parametrization of curves for robot joint trajectories

Another trajectory optimization problem

The robot trajectory is known \((\theta_i(\tau))\) and a parametrization \((t = h(\tau))\) is to be computed.

Find a parametrization

\[
t = h(\tau), \quad \tau \in [0, 1], \quad \tau_f = 1
\]

where \(\theta^*_i(t) = \theta_i(h^{-1}(t))\), such that

<table>
<thead>
<tr>
<th>(Model 1)</th>
<th>(Model 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(h(0) = 0)</td>
<td>(h(1)) is minimum</td>
</tr>
<tr>
<td>(h'(\tau) &gt; 0, \quad \tau \in [0, 1])</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>\frac{d\theta^*_i(t)}{dt}</td>
</tr>
<tr>
<td>(</td>
<td>\frac{d^2\theta^*_i(t)}{dt^2}</td>
</tr>
<tr>
<td>(</td>
<td>\frac{d^3\theta^*_i(t)}{dt^3}</td>
</tr>
</tbody>
</table>

\(i = 1, \ldots, l\)
Some practical applications

Robot trajectory planning

Using B-splines

The objective is to find a parametrization \( h(\tau) \) that minimizes the total time travel.

Let

\[
g(\tau) = h'(\tau)
\]

be approximated by a B-Spline \( B_{k,\xi}(\tau) \)

The total time travel is simply the integral of the parametric curve:

\[
\int_0^1 g(\tau) d\tau = \int_0^1 B_{k,\xi}(\tau) d\tau = \frac{1}{k} \sum_{i=1}^{n} x_i \left( \xi_{i+k} - \xi_i \right)
\]
The complete problem formulation

**Model 1**

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad \int_0^1 g(\tau) d\tau \\
\text{s.t.} & \quad g(\tau) > 0 \\
& \quad \left| \frac{d\theta^*_i}{dt} \right| \leq C_{i,1} \\
& \quad \left| \frac{d^2\theta^*_i}{dt^2} \right| \leq C_{i,2} \\
& \quad \left| \frac{d^3\theta^*_i}{dt^3} \right| \leq C_{i,3} \quad i = 1, \ldots, l \\
& \quad \forall \tau \in [0, 1]
\end{align*}
\]

**Model 2**

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad \int_0^1 g(\tau) d\tau \\
\text{s.t.} & \quad g(\tau) > 0 \\
& \quad |F_i| \leq C_{i} \quad i = 1, \ldots, l \\
& \quad \forall \tau \in [0, 1].
\end{align*}
\]
The complete problem formulation

**Model 1**

\[
\min_{x \in \mathbb{R}^n} \int_0^1 g(\tau) d\tau \\
\text{s.t.} \quad g(\tau) > 0 \\
\left| \frac{d\theta_i^*}{dt} \right| \leq C_{i,1} \\
\left| \frac{d^2\theta_i^*}{dt^2} \right| \leq C_{i,2} \\
\left| \frac{d^3\theta_i^*}{dt^3} \right| \leq C_{i,3} \quad i = 1, \ldots, l \\
\forall \tau \in [0, 1] ,
\]

**Model 2**

\[
\min_{x \in \mathbb{R}^n} \int_0^1 g(\tau) d\tau \\
\text{s.t.} \quad g(\tau) > 0 \\
\left| F_i \right| \leq C_i \quad i = 1, \ldots, l \\
\forall \tau \in [0, 1] .
\]
Some practical applications

Some results with problems available at SIPAMPL

Plot - $g(\tau)$ and $g'(\tau)$ for problem elke1std (Model 1)
Practical application III

Air pollution control
Coordinate system

(a, b)  stack position
\( d \)  stack internal diameter
\( h \)  stack height
\( \Delta H \)  plume rise
\( H = h + \Delta H \)  effective stack height
\( \theta \)  mean wind direction

\( \Delta H \)  rise of the plume
\( H \)  effective stack height
\( b \)  stack base diameter
\( d \)  stack internal diameter
\( h \)  stack height
\( \theta \)  mean wind direction

Ismael Vaz  (UMinho - PT)
Assuming that the plume has a Gaussian distribution, the concentration, of gas or aerosol (particles with diameter less than 20 microns) at position $x$, $y$ and $z$ of a continuous source with effective stack height $H$, is given by

$$C(x, y, z, H) = \frac{Q}{2\pi \sigma_y \sigma_z U} e^{-\frac{1}{2} \left( \frac{y}{\sigma_y} \right)^2} \left( e^{-\frac{1}{2} \left( \frac{z-H}{\sigma_z} \right)^2} + e^{-\frac{1}{2} \left( \frac{z+H}{\sigma_z} \right)^2} \right)$$

where $Q\ (gs^{-1})$ is the pollution uniform emission rate, $U\ (ms^{-1})$ is the mean wind speed affecting the plume, $\sigma_y\ (m)$ and $\sigma_z\ (m)$ are the standard deviations in the horizontal and vertical planes, respectively.
Change of coordinates

The source change of coordinates to position \((a, b)\), in the wind direction. \(Y\) is given by

\[ Y = (x - a) \sin(\theta) + (y - b) \cos(\theta), \]

where \(\theta\) (rad) is the wind direction \((0 \leq \theta \leq 2\pi)\).

\(\sigma_y\) and \(\sigma_z\) depend on \(X\) given by

\[ X = (x - a) \cos(\theta) - (y - b) \sin(\theta). \]
Plume rise

The effective emission height is the sum of the stack height, $h \ (m)$, with the plume rise, $\Delta H \ (m)$. The considered elevation is given by the Holland equation

$$\Delta H = \frac{V_0 d}{U} \left(1.5 + 2.68 \frac{T_o - T_e}{T_o} d\right),$$

where $d \ (m)$ is the internal stack diameter, $V_0 \ (ms^{-1})$ is the gas out velocity, $T_o \ (K)$ is the gas temperature and $T_e \ (K)$ is the environment temperature.
Formulations

- Assuming $n$ pollution sources distributed in a region;
- $C_i$ is the source $i$ contribution for the total concentration;
- Gas chemical inert.

We can derive three formulations:

- Minimize the stack height;
- Maximum pollution computation and sampling stations planning;
- Air pollution abatement.
Formulations

- Assuming \( n \) pollution sources distributed in a region;
- \( C_i \) is the source \( i \) contribution for the total concentration;
- Gas chemical inert.

We can derive three formulations:

- Minimize the stack height;
- Maximum pollution computation and sampling stations planning;
- Air pollution abatement.
Formulations

- Assuming $n$ pollution sources distributed in a region;
- $C_i$ is the source $i$ contribution for the total concentration;
- Gas chemical inert.

We can derive three formulations:

- Minimize the stack height;
- Maximum pollution computation and sampling stations planning;
- Air pollution abatement.
Formulations

- Assuming \( n \) pollution sources distributed in a region;
- \( C_i \) is the source \( i \) contribution for the total concentration;
- Gas chemical inert.

We can derive three formulations:
- Minimize the stack height;
- Maximum pollution computation and sampling stations planning;
- Air pollution abatement.
Assuming $n$ pollution sources distributed in a region;

- $C_i$ is the source $i$ contribution for the total concentration;

- Gas chemical inert.

We can derive three formulations:

- Minimize the stack height;

- Maximum pollution computation and sampling stations planning;

- Air pollution abatement.
Formulations

- Assuming $n$ pollution sources distributed in a region;
- $C_i$ is the source $i$ contribution for the total concentration;
- Gas chemical inert.

We can derive three formulations:

- Minimize the stack height;
- Maximum pollution computation and sampling stations planning;
- Air pollution abatement.
Minimum stack height

Minimizing the stack height $u = (h_1, \ldots, h_n)$, while the pollution ground pollution level is kept below a given threshold $C_0$, in a given region $\mathcal{R}$, can be formulated as a SIP problem

$$\min_{u \in \mathbb{R}^n} \sum_{i=1}^{n} c_i h_i$$

$$s.t. \ g(u, v \equiv (x, y)) \equiv \sum_{i=1}^{n} C_i(x, y, 0, H_i) \leq C_0$$

$$\forall v \in \mathcal{R} \subset \mathbb{R}^2,$$

where $c_i$, $i = 1, \ldots, n$, are the construction costs.

Note: more complex objective function can be considered.
Maximum pollution and sampling stations planning

The maximum pollution concentration \( (l^*) \) in a given region can be obtained by solving the following SIP problem

\[
\min_{l \in \mathbb{R}} l \\
\text{s.t. } g(z, v) = \left( x, y \right) \equiv \sum_{i=1}^{n} C_i(x, y, 0, \mathcal{H}_i) \leq l \\
\forall v \in \mathcal{R} \subset \mathbb{R}^2.
\]

The active points \( v^* \in \mathcal{R} \) where \( g(z^*, v^*) = l^* \) are the global optima and indicate where the sampling (control) stations should be placed.
Minimizing the pollution abatement (minimizing clean costs, maximizing the revenue, minimizing the economical impact) while the air pollution concentration is kept below a given threshold can be posed as a SIP problem

\[
\min_{u \in \mathbb{R}^n} \sum_{i=1}^{n} p_i r_i \\
\text{s.t. } g(u, v) \equiv (x, y) \equiv \sum_{i=1}^{n} (1 - r_i) C_i(x, y, 0, H_i) \leq C_0 \\
\forall v \in \mathcal{R} \subset \mathbb{R}^2,
\]

where \( u = (r_1, \ldots, r_n) \) is the pollution reduction and \( p_i, i = 1, \ldots, n, \) is the source \( i \) cost (cleaning or not producing).
### Numerical results – Minimum stack height (vaz1)

<table>
<thead>
<tr>
<th></th>
<th>Instance 1</th>
<th>Instance 2</th>
<th>Instance 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>0.00</td>
<td>10.00</td>
<td>196.93</td>
</tr>
<tr>
<td>$h_2$</td>
<td>78.26</td>
<td>69.09</td>
<td>380.06</td>
</tr>
<tr>
<td>$h_3$</td>
<td>0.00</td>
<td>10.00</td>
<td>403.12</td>
</tr>
<tr>
<td>$h_4$</td>
<td>153.17</td>
<td>152.64</td>
<td>428.38</td>
</tr>
<tr>
<td>$h_5$</td>
<td>80.90</td>
<td>71.27</td>
<td>344.81</td>
</tr>
<tr>
<td>$h_6$</td>
<td>0.00</td>
<td>10.00</td>
<td>274.58</td>
</tr>
<tr>
<td>$h_7$</td>
<td>13.52</td>
<td>13.52</td>
<td>402.83</td>
</tr>
<tr>
<td>$h_8$</td>
<td>161.78</td>
<td>161.87</td>
<td>396.82</td>
</tr>
<tr>
<td>$h_9$</td>
<td>141.73</td>
<td>141.63</td>
<td>415.58</td>
</tr>
<tr>
<td>$h_{10}$</td>
<td>15.05</td>
<td>15.05</td>
<td>423.99</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>644.40</strong></td>
<td><strong>655.06</strong></td>
<td><strong>3667.10</strong></td>
</tr>
</tbody>
</table>

Instance 1 – no limit on stack, Instance 2 – limit of 10m, Instance 3 – Portuguese legislation.
Outline

1. Semi-Infinite Programming (SIP) Notation
2. Numerical methods for SIP
3. Some practical applications
4. The particle swarm algorithm
5. Modification of PSOA for multi-local optimization
We intended to solve the following global optimization problem with a particle swarm algorithm.

**Global optimization problem**

\[
\max_{t \in T} \bar{g}(t) \equiv g(\bar{x}, t)
\]

with \( T \in \mathbb{R}^p \).
The Particle Swarm Paradigm (PSP)

The PSP is a population (swarm) based algorithm that mimics the social behavior of a set of individuals (particles).

An individual behavior is a combination of its past experience (cognition influence) and the society experience (social influence).

In the optimization context a particle $\varphi$, at time instant $k$, is represented by its current position ($t^\varphi(k)$), its best ever position ($y^\varphi(k)$) and its traveling velocity ($v^\varphi(k)$).
The Particle Swarm Paradigm (PSP)

The PSP is a population (swarm) based algorithm that mimics the social behavior of a set of individuals (particles).

An individual behavior is a combination of its past experience (cognition influence) and the society experience (social influence).

In the optimization context a particle \( \varphi \), at time instant \( k \), is represented by its current position \( t_\varphi(k) \), its best ever position \( y_\varphi(k) \) and its traveling velocity \( v_\varphi(k) \).
The Particle Swarm Paradigm (PSP)

The PSP is a population (swarm) based algorithm that mimics the social behavior of a set of individuals (particles).

An individual behavior is a combination of its past experience (cognition influence) and the society experience (social influence).

In the optimization context a particle $\varphi$, at time instant $k$, is represented by its current position ($t^\varphi(k)$), its best ever position ($y^\varphi(k)$) and its traveling velocity ($v^\varphi(k)$).
The new travel position and velocity

The new particle position is updated by

**Update position**

\[ t^\varphi(k + 1) = t^\varphi(k) + v^\varphi(k + 1), \]

where \( v^\varphi(k + 1) \) is the new velocity given by

**Update velocity**

\[ v^\varphi_j(k + 1) = \iota(k) v^\varphi_j(k) + \mu \omega_1 \psi_j(k) \left( y^\varphi_j(k) - t^\varphi_j(k) \right) + \nu \omega_2 \psi_j(k) \left( \hat{y}_j(k) - t^\varphi_j(k) \right), \]

for \( j = 1, \ldots, p \).

- \( \iota(k) \) is a weighting factor (inertial)
- \( \mu \) is the cognition parameter and \( \nu \) is the social parameter
- \( \omega_1(k) \) and \( \omega_2(k) \) are random numbers drawn from the uniform \((0, 1)\) distribution.
The new travel position and velocity

The new particle position is updated by

Update position

\[ t^\phi(k + 1) = t^\phi(k) + v^\phi(k + 1), \]

where \( v^\phi(k + 1) \) is the new velocity given by

Update velocity

\[ v^\phi_j(k + 1) = \nu(k)v^\phi_j(k) + \mu \omega_1 j(k) \left( y^\phi_j(k) - t^\phi_j(k) \right) + \nu \omega_2 j(k) \left( \hat{y}_j(k) - t^\phi_j(k) \right), \]

for \( j = 1, \ldots, p. \)

- \( \nu(k) \) is a weighting factor (inertial)
- \( \mu \) is the cognition parameter and \( \nu \) is the social parameter
- \( \omega_1 j(k) \) and \( \omega_2 j(k) \) are random numbers drawn from the uniform \((0, 1)\) distribution.
The new travel position and velocity

The new particle position is updated by

**Update position**

\[
t^{\mathcal{P}}(k+1) = t^{\mathcal{P}}(k) + v^{\mathcal{P}}(k+1),
\]

where \( v^{\mathcal{P}}(k+1) \) is the new velocity given by

**Update velocity**

\[
v^{\mathcal{P}}_{j}(k+1) = \iota(k)v^{\mathcal{P}}_{j}(k) + \mu \omega_{1_{j}}(k) \left( y^{\mathcal{P}}_{j}(k) - t^{\mathcal{P}}_{j}(k) \right) + \nu \omega_{2_{j}}(k) \left( \hat{y}_{j}(k) - t^{\mathcal{P}}_{j}(k) \right),
\]

for \( j = 1, \ldots, p \).

- \( \iota(k) \) is a weighting factor (inertial)
- \( \mu \) is the cognition parameter and \( \nu \) is the social parameter
- \( \omega_{1_{j}}(k) \) and \( \omega_{2_{j}}(k) \) are random numbers drawn from the uniform \((0, 1)\) distribution.
The new travel position and velocity

The new particle position is updated by

\[ t^\phi(k + 1) = t^\phi(k) + v^\phi(k + 1), \]

where \( v^\phi(k + 1) \) is the new velocity given by

\[ v^\phi_j(k + 1) = \iota(k)v^\phi_j(k) + \mu \omega_1 j(k) \left( y^\phi_j(k) - t^\phi_j(k) \right) + \nu \omega_2 j(k) \left( \hat{y}_j(k) - t^\phi_j(k) \right), \]

for \( j = 1, \ldots, p. \)

- \( \iota(k) \) is a weighting factor (inertial)
- \( \mu \) is the cognition parameter and \( \nu \) is the social parameter
- \( \omega_1 j(k) \) and \( \omega_2 j(k) \) are random numbers drawn from the uniform \((0,1)\) distribution.
The new travel position and velocity

The new particle position is updated by

**Update position**

\[
t^\varphi(k + 1) = t^\varphi(k) + v^\varphi(k + 1),
\]

where \(v^\varphi(k + 1)\) is the new velocity given by

**Update velocity**

\[
v^\varphi_j(k + 1) = \nu(k)v^\varphi_j(k) + \mu_1j(k) \left( y^\varphi_j(k) - t^\varphi_j(k) \right) + \nu_2j(k) \left( \hat{y}_j(k) - t^\varphi_j(k) \right),
\]

for \(j = 1, \ldots, p\).

- \(\nu(k)\) is a weighting factor (inertial)
- \(\mu\) is the *cognition* parameter and \(\nu\) is the *social* parameter
- \(\omega_1j(k)\) and \(\omega_2j(k)\) are random numbers drawn from the uniform \((0, 1)\) distribution.
The best ever particle

\( \hat{y}(k) \) is a particle position with global best function value so far, i.e.,

\[
\hat{y}(k) \in \arg \min_{a \in \mathcal{A}} \bar{g}(a)
\]

\[
\mathcal{A} = \{ y^1(k), \ldots, y^s(k) \}.
\]

where \( s \) is the number of particles in the swarm.

Note

In an algorithmic point of view we just have to keep track of the particle with the best ever function value.
**The best ever particle**

\( \hat{y}(k) \) is a particle position with global best function value so far, *i.e.*, 

\[
\hat{y}(k) \in \arg \min_{a \in \mathcal{A}} \bar{g}(a)
\]

\[
\mathcal{A} = \{ y^1(k), \ldots, y^s(k) \}.
\]

where \( s \) is the number of particles in the swarm.

**Note**

In an algorithmic point of view we just have to keep track of the particle with the best ever function value.
The particle swarm algorithm

Some features

Features

Population based algorithm.

🌟 Good
- Easy to implement.
- Easy to parallelize.
- Easy to handle discrete variables.
- Only uses objective function evaluations.

🌟 Not so good
- Slow rate of convergence near an optimum.
- Quite large number of function evaluations.
- In the presence of several global optima the algorithm may not converge.
Features

Population based algorithm.

**Good**
- Easy to implement.
- Easy to parallelize.
- Easy to handle discrete variables.
- Only uses objective function evaluations.

**Not so good**
- Slow rate of convergence near an optimum.
- Quite large number of function evaluations.
- In the presence of several global optima the algorithm may not converge.
The particle swarm algorithm

Some features

Features

Population based algorithm.

- **Good**
  - Easy to implement.
  - Easy to parallelize.
  - Easy to handle discrete variables.
  - Only uses objective function evaluations.

- **Not so good**
  - Slow rate of convergence near an optimum.
  - Quite large number of function evaluations.
  - In the presence of several global optima the algorithm may not converge.
Features

Population based algorithm.

- **Good**
  - Easy to implement.
  - Easy to parallelize.
  - Easy to handle discrete variables.
  - Only uses objective function evaluations.

- **Not so good**
  - Slow rate of convergence near an optimum.
  - Quite large number of function evaluations.
  - In the presence of several global optima the algorithm may not converge.
The particle swarm algorithm

Features

Population based algorithm.

![Good]

- Easy to implement.
- Easy to parallelize.
- Easy to handle discrete variables.
- Only uses objective function evaluations.

![Not so good]

- Slow rate of convergence near an optimum.
- Quite large number of function evaluations.
- In the presence of several global optima the algorithm may not converge.
Features

Population based algorithm.

- **Good**
  - Easy to implement.
  - Easy to parallelize.
  - Easy to handle discrete variables.
  - Only uses objective function evaluations.

- **Not so good**
  - Slow rate of convergence near an optimum.
  - Quite large number of function evaluations.
  - In the presence of several global optima the algorithm may not converge.
Features

Population based algorithm.

- **Good**
  - Easy to implement.
  - Easy to parallelize.
  - Easy to handle discrete variables.
  - Only uses objective function evaluations.

- **Not so good**
  - Slow rate of convergence near an optimum.
  - Quite large number of function evaluations.
  - In the presence of several global optima the algorithm may not converge.
Features

Population based algorithm.

- **Good**
  - Easy to implement.
  - Easy to parallelize.
  - Easy to handle discrete variables.
  - Only uses objective function evaluations.

- **Not so good**
  - Slow rate of convergence near an optimum.
  - Quite large number of function evaluations.
  - In the presence of several global optima the algorithm may not converge.
Features

Population based algorithm.

- **Good**
  - Easy to implement.
  - Easy to parallelize.
  - Easy to handle discrete variables.
  - Only uses objective function evaluations.

- **Not so good**
  - Slow rate of convergence near an optimum.
  - Quite large number of function evaluations.
  - In the presence of several global optima the algorithm may not converge.
Properties

- With a proper selection of the algorithm parameters finite termination of the algorithm can be established, in a probabilistic sense.
- Convergence for a global optimum is not guaranteed by this simple version of the particle swarm algorithm, but some adaption can be introduce to guarantee it.
Properties

- With a proper selection of the algorithm parameters finite termination of the algorithm can be established, in a probabilistic sense.

- Convergence for a global optimum is not guaranteed by this simple version of the particle swarm algorithm, but some adaption can be introduce to guarantee it.
Outline

1. Semi-Infinite Programming (SIP) Notation
2. Numerical methods for SIP
3. Some practical applications
4. The particle swarm algorithm
5. Modification of PSOA for multi-local optimization
Multi-local revisited

Given $\bar{x}$ the multi-local optimization problem is defined as

$$\max_{t \in T} g(\bar{x}, t) \equiv \bar{g}(t)$$

with $T \in \mathbb{R}^n$.

The multi-local concept

All the global and local optima are to be computed.

Some characteristics

These problems are mostly differentiable and the objective function computation is costless.
Given $\bar{x}$ the multi-local optimization problem is defined as

$$\max_{t \in T} g(\bar{x}, t) \equiv \bar{g}(t)$$

with $T \in \mathbb{R}^n$.

The multi-local concept

All the global and local optima are to be computed.

Some characteristics

These problems are mostly differentiable and the objective function computation is costless.
Multi-local revisited

Given $\bar{x}$ the multi-local optimization problem is defined as

**Multi-local optimization problem**

$$\max_{t \in T} g(\bar{x}, t) \equiv \bar{g}(t)$$

with $T \in \mathbb{R}^n$.

**The multi-local concept**

All the global and local optima are to be computed.

**Some characteristics**

These problems are mostly differentiable and the objective function computation is costless.
PSP with the steepest ascent (quasi-Newton) direction

The new particle position update equation is kept while the new velocity equation is given by

\[ v_j^o(k+1) = \iota(k)v_j^o(k) + \mu \omega_1 j(k) (y_j^o(k) - t_j^o(k)) + \nu \omega_2 t(t) \left( \nabla_j g(y_j^o(k)) \right), \]

for \( j = 1, \ldots, p \), where \( \nabla \bar{g}(t) \) is the gradient of the objective function.

Each particle uses the steepest ascent direction computed at each particle best position \( (y_j^o(k)) \).

The inclusion of the steepest ascent direction in the velocity equation aims to drive each particle to a neighbor local maximum and since we have a population of particles, each one will be driven to a local maximum.
PSP with an ascent direction

Other approach is to use

Ascent velocity formula

\[
   w^\varphi = \frac{1}{\sum_{j=1}^{m} |\bar{g}(z_j^\varphi) - \bar{g}(y^\varphi)|} \sum_{j=1}^{m} \frac{(\bar{g}(z_j^\varphi) - \bar{g}(y^\varphi))}{\|z_j^\varphi - y^\varphi\|} (z_j^\varphi - y^\varphi)
\]

as an ascent direction at \( y^\varphi \), in the velocity equation, to overcome the need to compute the gradient.

Where

- \( y^\varphi \) is the best position of particle \( \varphi \)
- \( \{z_j^\varphi\}_{j=1}^{m} \) is a set of \( m \) (random) points close to \( y^p \),

Under certain conditions \( w^\varphi \) simulates the steepest ascent direction.
Stopping criterion

We propose the stopping criterion

**Minimum velocity attained**

$$\max_{\phi} [v^{\phi}(k)]_{opt} \leq \epsilon_{\phi}$$

where

**Constrained velocity**

$$[v^{\phi}(k)]_{opt} = \left( \sum_{j=1}^{p} \begin{cases} 0 & \text{if } t^{\phi}_j(k) = \beta_j \text{ and } v^{\phi}_j(k) \geq 0 \\ 0 & \text{if } t^{\phi}_j(k) = \alpha_j \text{ and } v^{\phi}_j(k) \leq 0 \\ \left(v^{\phi}_j(k)\right)^2 & \text{otherwise} \end{cases} \right)^{1/2}$$

The stopping criterion is based on the optimality conditions for the multi-local optimization problem.
## Numerical results

### Gradient version

<table>
<thead>
<tr>
<th>F.O.</th>
<th>Nafe</th>
<th>Nage</th>
<th>$g_a^*$</th>
<th>$g_{best}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>0</td>
<td>177</td>
<td>4,652E+03</td>
<td>2,393E+03</td>
</tr>
<tr>
<td>18</td>
<td>100</td>
<td>1850</td>
<td>-9,160E+00</td>
<td>-1,026E+01</td>
</tr>
<tr>
<td>19</td>
<td>100</td>
<td>2126</td>
<td>-7,801E+00</td>
<td>-8,760E+00</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>1909</td>
<td>-9,401E+00</td>
<td>-9,997E+00</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>335</td>
<td>-1,024E+02</td>
<td>-1,648E+02</td>
</tr>
<tr>
<td>22</td>
<td>100</td>
<td>973</td>
<td>-4,075E-01</td>
<td>-4,075E-01</td>
</tr>
<tr>
<td>23</td>
<td>100</td>
<td>570</td>
<td>-1,806E+01</td>
<td>-1,806E+01</td>
</tr>
<tr>
<td>24</td>
<td>100</td>
<td>194</td>
<td>-2,278E+02</td>
<td>-2,278E+02</td>
</tr>
<tr>
<td>25</td>
<td>100</td>
<td>167</td>
<td>-2,429E+03</td>
<td>-2,429E+03</td>
</tr>
<tr>
<td>26</td>
<td>90</td>
<td>81</td>
<td>-2,477E+04</td>
<td>-2,478E+04</td>
</tr>
<tr>
<td>27</td>
<td>10</td>
<td>58</td>
<td>1,607E+05</td>
<td>-2,436E+05</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>141</td>
<td>4,470E+02</td>
<td>3,102E+01</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
<td>135</td>
<td>1,289E+05</td>
<td>7,935E+02</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>16314</td>
<td>8,325E-112</td>
<td>0,000E+00</td>
</tr>
<tr>
<td>31</td>
<td>100</td>
<td>313</td>
<td>1,997E-13</td>
<td>2,780E-21</td>
</tr>
<tr>
<td>32</td>
<td>40</td>
<td>160</td>
<td>8,338E+00</td>
<td>3,031E-04</td>
</tr>
</tbody>
</table>

### Approximate descent direction version

<table>
<thead>
<tr>
<th>F.O.</th>
<th>Nafe</th>
<th>$g_a^*$</th>
<th>$g_{best}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>100 5589</td>
<td>2,203E-01</td>
<td>1,327E-01</td>
</tr>
<tr>
<td>18</td>
<td>100 4066</td>
<td>-1,052E+01</td>
<td>-1,052E+01</td>
</tr>
<tr>
<td>19</td>
<td>100 3906</td>
<td>-1,012E+01</td>
<td>-1,014E+01</td>
</tr>
<tr>
<td>20</td>
<td>100 4069</td>
<td>-1,052E+01</td>
<td>-1,052E+01</td>
</tr>
<tr>
<td>21</td>
<td>60   9999</td>
<td>-1,867E+02</td>
<td>-1,867E+02</td>
</tr>
<tr>
<td>22</td>
<td>100 804</td>
<td>-4,075E-01</td>
<td>-4,075E-01</td>
</tr>
<tr>
<td>23</td>
<td>100 902</td>
<td>-1,806E+01</td>
<td>-1,806E+01</td>
</tr>
<tr>
<td>24</td>
<td>100 1003</td>
<td>-2,278E+02</td>
<td>-2,278E+02</td>
</tr>
<tr>
<td>25</td>
<td>100 1160</td>
<td>-2,429E+03</td>
<td>-2,429E+03</td>
</tr>
<tr>
<td>26</td>
<td>100 1278</td>
<td>-2,478E+04</td>
<td>-2,478E+04</td>
</tr>
<tr>
<td>27</td>
<td>100 1418</td>
<td>-2,493E+05</td>
<td>-2,493E+05</td>
</tr>
<tr>
<td>28</td>
<td>60   9759</td>
<td>3,977E-02</td>
<td>2,506E-02</td>
</tr>
<tr>
<td>29</td>
<td>0    16905</td>
<td>3,633E-01</td>
<td>2,404E-01</td>
</tr>
<tr>
<td>30</td>
<td>100 1264</td>
<td>4,987E-07</td>
<td>4,464E-08</td>
</tr>
<tr>
<td>31</td>
<td>100 5221</td>
<td>2,231E-04</td>
<td>6,612E-05</td>
</tr>
<tr>
<td>32</td>
<td>100 6065</td>
<td>2,005E-03</td>
<td>1,186E-03</td>
</tr>
</tbody>
</table>
The test set for SIP

The test problems were obtained from SIP where $\bar{x}$ was replaced by $x^*$, where $x^*$ is the SIP solution included in the SIPAMPL database. SIPAMPL stands for SIP with AMPL and is a software package that provides, among other features, a database of SIP coded problems.

All SIP problems considered have only one infinite constraint.

<table>
<thead>
<tr>
<th>SIP problem</th>
<th>Test problem</th>
<th>$p$</th>
<th>Obs</th>
</tr>
</thead>
<tbody>
<tr>
<td>watson2</td>
<td>sip_wat2</td>
<td>1</td>
<td>Unidimensional</td>
</tr>
<tr>
<td>vaz3</td>
<td>sip_vaz3</td>
<td>2</td>
<td>Air pollution abatement</td>
</tr>
<tr>
<td>priceS6</td>
<td>sip_S6</td>
<td>6</td>
<td>Higher dimension in SIPAMPL</td>
</tr>
<tr>
<td>priceU</td>
<td>sip_U</td>
<td>6</td>
<td>Higher dimension in SIPAMPL</td>
</tr>
<tr>
<td>random</td>
<td>sip_rand</td>
<td>6</td>
<td>Random generated with known solution</td>
</tr>
</tbody>
</table>
The test set for SIP

The test problems were obtained from SIP where $\bar{x}$ was replaced by $x^*$, where $x^*$ is the SIP solution included in the SIPAMPL database. SIPAMPL stands for SIP with AMPL and is a software package that provides, among other features, a database of SIP coded problems.

All SIP problems considered have only one infinite constraint.

<table>
<thead>
<tr>
<th>SIP problem</th>
<th>Test problem</th>
<th>$p$</th>
<th>Obs</th>
</tr>
</thead>
<tbody>
<tr>
<td>watson2</td>
<td>sip_wat2</td>
<td>1</td>
<td>Unidimensional</td>
</tr>
<tr>
<td>vaz3</td>
<td>sip_vaz3</td>
<td>2</td>
<td>Air pollution abatement</td>
</tr>
<tr>
<td>priceS6</td>
<td>sip_S6</td>
<td>6</td>
<td>Higher dimension in SIPAMPL</td>
</tr>
<tr>
<td>priceU</td>
<td>sip_U</td>
<td>6</td>
<td>Higher dimension in SIPAMPL</td>
</tr>
<tr>
<td>random</td>
<td>sip_rand</td>
<td>6</td>
<td>Random generated with known solution</td>
</tr>
</tbody>
</table>
Numerical results

- A population of 40 particles and a maximum of 2000 iterations was used, with the steepest ascent direction version.
- A global and a local maxima were found. 10 particles converged to the local maxima \( t = 1 \) with \( \bar{g}(1) = -0.058594 \) and the remaining 30 to the global one \( (t = 0) \) with \( \bar{g}(0) = -2.5156e - 08 \).
- In sip_vaz3 the objective function is flat (equal to zero) in a subregion.

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \bar{g}(t) )</th>
<th>( npar )</th>
</tr>
</thead>
<tbody>
<tr>
<td>((-0.783012, 2.172526))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((-0.112199, -0.686259))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((-0.278460, 0.095245))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((-0.446057, 1.157275))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((0.443709, 3.811052))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((3.684002, -0.629689))</td>
<td>0.500007</td>
<td>22</td>
</tr>
<tr>
<td>((1.099826, 0.112477))</td>
<td>0.500055</td>
<td>13</td>
</tr>
</tbody>
</table>
Numerical results

- A population of 40 particles and a maximum of 2000 iterations was used, with the steepest ascent direction version.
- sip_wat2 a global and a local maxima were found. 10 particles converged to the local maxima \( t = 1 \) with \( \bar{g}(1) = -0.058594 \) and the remaining 30 to the global one \( (t = 0) \) with \( \bar{g}(0) = -2.5156e - 08 \)
- In sip_vaz3 the objective function is flat (equal to zero) in a subregion.

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \bar{g}(t) )</th>
<th>( npar )</th>
</tr>
</thead>
<tbody>
<tr>
<td>((-0.783012, 2.172526))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((-0.112199, -0.686259))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((-0.278460, 0.095245))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((-0.446057, 1.157275))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((0.443709, 3.811052))</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>((3.684002, -0.629689))</td>
<td>0.500000</td>
<td>22</td>
</tr>
<tr>
<td>((1.099826, 0.112477))</td>
<td>0.500055</td>
<td>13</td>
</tr>
</tbody>
</table>
Numerical results

- A population of 40 particles and a maximum of 2000 iterations was used, with the steepest ascent direction version.
- `sip_wat2` a global and a local maxima were found. 10 particles converged to the local maxima \( t = 1 \) with \( \bar{g}(1) = -0.058594 \) and the remaining 30 to the global one \( (t = 0) \) with \( \bar{g}(0) = -2.5156e - 08 \)
- In `sip_vaz3` the objective function is flat (equal to zero) in a subregion.

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \bar{g}(t) )</th>
<th>( npar )</th>
</tr>
</thead>
<tbody>
<tr>
<td>((-0.783012, 2.172526))</td>
<td>0.0000000</td>
<td>1</td>
</tr>
<tr>
<td>((-0.112199, -0.686259))</td>
<td>0.0000000</td>
<td>1</td>
</tr>
<tr>
<td>((-0.278460, 0.095245))</td>
<td>0.0000000</td>
<td>1</td>
</tr>
<tr>
<td>((-0.446057, 1.157275))</td>
<td>0.0000000</td>
<td>1</td>
</tr>
<tr>
<td>((0.443709, 3.811052))</td>
<td>0.0000000</td>
<td>1</td>
</tr>
<tr>
<td>((3.684002, -0.629689))</td>
<td>0.500007</td>
<td>22</td>
</tr>
<tr>
<td>((1.099826, 0.112477))</td>
<td>0.500055</td>
<td>13</td>
</tr>
</tbody>
</table>
# Numerical results

**sip_S6** a reported global maximizer and two local with objective function values of 0.027092, -3.69008 and -1.95425 respectively.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\bar{g}(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$(1.622134, 1.687810, 2.000000, 0.085439, 2.000000, 0.350174)$</td>
<td>0.024811</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$(1.634326, 1.671065, 2.000000, 0.054348, 2.000000, 2.000000)$</td>
<td>-1.954538</td>
</tr>
</tbody>
</table>

**sip_U** reported two global maximizers and eleven local maximizers.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\bar{g}(t)$</th>
<th>npar</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(-0.665555,-1.000000,1.00,1.00,1.00,1.00)$</td>
<td>-0.002587</td>
<td>1</td>
</tr>
<tr>
<td>$(-0.689138,-0.933410,1.00,1.00,1.00,1.00)$</td>
<td>-0.003319</td>
<td>1</td>
</tr>
<tr>
<td>$(-0.890160,-1.000000,1.00,1.00,1.00,1.00)$</td>
<td>-0.000225</td>
<td>1</td>
</tr>
<tr>
<td>$(-0.894640,-1.000000,1.00,1.00,1.00,1.00)$</td>
<td>-0.000103</td>
<td>1</td>
</tr>
<tr>
<td>$(-0.897369,-1.000000,1.00,1.00,1.00,1.00)$</td>
<td>-0.000648</td>
<td>1</td>
</tr>
<tr>
<td>$(1.000000,1.000000,1.00,1.00,1.00,1.00)$</td>
<td>$0.239638e-07$</td>
<td>35</td>
</tr>
</tbody>
</table>
### Numerical results

**sip_S6** reported a global maximizer and two local with objective function values of 0.027092, -3.69008 and -1.95425 respectively.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\bar{g}(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(1.622134, 1.687810, 2.000000, 0.085439, 2.000000, 0.350174)</td>
<td>0.024811</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(1.634326, 1.671065, 2.000000, 0.054348, 2.000000, 2.000000)</td>
<td>-1.954538</td>
</tr>
</tbody>
</table>

**sip_U** reported two global maximizers and eleven local maximizers.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\bar{g}(t)$</th>
<th>$npar$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-0.665555, -1.000000, 1.00, 1.00, 1.00, 1.00)</td>
<td>-0.002587</td>
<td>1</td>
</tr>
<tr>
<td>(-0.689138, -0.933410, 1.00, 1.00, 1.00, 1.00)</td>
<td>-0.003319</td>
<td>1</td>
</tr>
<tr>
<td>(-0.890160, -1.000000, 1.00, 1.00, 1.00, 1.00)</td>
<td>-0.000225</td>
<td>1</td>
</tr>
<tr>
<td>(-0.894640, -1.000000, 1.00, 1.00, 1.00, 1.00)</td>
<td>-0.000103</td>
<td>1</td>
</tr>
<tr>
<td>(-0.897369, -1.000000, 1.00, 1.00, 1.000, 1.00)</td>
<td>-0.000648</td>
<td>1</td>
</tr>
<tr>
<td>(1.000000, 1.000000, 1.00, 1.00, 1.00, 1.00)</td>
<td>0.239638e-07</td>
<td>35</td>
</tr>
</tbody>
</table>
A quasi-Newton approach is incorporated with a particle swarm strategy in order to reduce the number of function evaluations.

A line-search is being used in order to guarantee the converge to at least a local solutions with high accuracy.

A MATLAB version is already implemented.

To implement a reduction type method for SIP using the developed strategy for multi-local optimization.
A quasi-Newton approach is incorporated with a particle swarm strategy in order to reduce the number of function evaluations.

A line-search is being used in order to guarantee the convergence to at least a local solutions with high accuracy.

A MATLAB version is already implemented.

To implement a reduction type method for SIP using the developed strategy for multi-local optimization.
Ongoing work

- A quasi-Newton approach is incorporated with a particle swarm strategy in order to reduce the number of function evaluations.
- A line-search is being used in order to guarantee the converge to at least a local solutions with high accuracy.
- A MATLAB version is already implemented.
- To implement a reduction type method for SIP using the developed strategy for multi-local optimization.
A quasi-Newton approach is incorporated with a particle swarm strategy in order to reduce the number of function evaluations.

A line-search is being used in order to guarantee the converge to at least a local solutions with high accuracy.

A MATLAB version is already implemented.

To implement a reduction type method for SIP using the developed strategy for multi-local optimization.
THE END

email: aivaz@dps.uminho.pt
Web http://www.norg.uminho.pt/aivaz