A Fast Interior Point Method for FreeFem++

Quick Tutorial

Sylvain Auliac - Laboratoire Jacques-Louis Lions
Interior-Point Methods

- A short overview
- The IPOPT software
- Using IPOPT through FreeFem++
Interior-Point Methods

- A short overview
- The IPOPT software
- Using IPOPT through FreeFem++

Some examples

- Quadratic problem
- A non-quadratic example
Interior Point Methods: a short overview
Classical minimization under constraints problem:

Objective function:

\[ f : \mathbb{R}^n \rightarrow \mathbb{R} \]

Find:

\[ x_0 = \arg\min_{x} f(x) \quad \forall i, \quad c_i(x) \geq 0 \]

Constraints:

\[ c = (c_1, \ldots, c_m) : \mathbb{R}^n \rightarrow \mathbb{R}^m \]
Classical minimization under constraints problem:

Objective function:
\[ f : \mathbb{R}^n \rightarrow \mathbb{R} \]

Find: \[ x_0 = \arg\min_{x \in \mathbb{R}^n} f(x) \quad \forall i, c_i(x) \geq 0 \]

Constraints:
\[ c = (c_1, \ldots, c_m) : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

Barrier Function:
For a given \( \mu > 0 \), solve the unconstrained problem of finding
\[ x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{m} \ln c_i(x) \]
**Classical minimization under constraints problem:**

**Objective function:**
\[
f : \mathbb{R}^n \rightarrow \mathbb{R}
\]

**Constraints:**
\[
c = (c_1, \ldots, c_m) : \mathbb{R}^n \rightarrow \mathbb{R}^m
\]

Find:
\[
x_0 = \arg\min_{x \in \mathbb{R}^n} f(x) \quad \forall i, c_i(x) \geq 0
\]

**Barrier Function:**
For a given \(\mu > 0\), solve the unconstrained problem of finding
\[
x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{m} \ln c_i(x)
\]

**IP Methods:**
given a decreasing sequence \((\mu_k)\) with \(\mu_k \rightarrow +\infty\)
find a sequence \((x_{\mu_k})\) of associated minimizers.
Barrier Function:
For a given $\mu > 0$, solve the unconstrained problem of finding

$$x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{m} \ln c_i(x)$$

IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \to +\infty$ find a sequence $(x_{\mu_k})$ of associated minimizers.
**Barrier Function**: For a given $\mu > 0$, solve the unconstrained problem of finding

$$x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{m} \ln c_i(x)$$

**IP Methods**: given a decreasing sequence $(\mu_k)$ with $\mu_k \to 0$ as $k \to +\infty$ find a sequence $(x_{\mu_k})$ of associated minimizers.

**Illustration**: minimize $f(x, y) = (x + 0.1)^2 + (y - 0.5)^2$ on a triangle
Barrier Function:
For a given $\mu > 0$, solve the unconstrained problem of finding

$$x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{m} \ln c_i(x)$$

IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \to 0$ find a sequence $(x_{\mu_k})$ of associated minimizers.

Illustration: minimize $f(x, y) = (x + 0.1)^2 + (y - 0.5)^2$ on a triangle

Isovalues of $f$
Barrier Function:
For a given $\mu > 0$, solve the unconstrained problem of finding

$$x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{m} \ln c_i(x)$$

IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \to 0$
find a sequence $(x_{\mu_k})$ of associated minimizers.

Illustration: minimize $f(x, y) = (x + 0.1)^2 + (y - 0.5)^2$ on a triangle

Isovalues of $f$

$log(\mu) = 2$
Barrier Function:
For a given $\mu > 0$, solve the unconstrained problem of finding

$$x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^m \ln c_i(x)$$

IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \to 0$ as $k \to +\infty$, find a sequence $(x_{\mu_k})$ of associated minimizers.

Illustration: minimize $f(x, y) = (x + 0.1)^2 + (y - 0.5)^2$ on a triangle
Barrier Function:
For a given $\mu > 0$, solve the unconstrained problem of finding

$$x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{m} \ln c_i(x)$$

IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \to 0$ as $k \to +\infty$ find a sequence $(x_{\mu_k})$ of associated minimizers.

Illustration: minimize $f(x, y) = (x + 0.1)^2 + (y - 0.5)^2$ on a triangle
Barrier Function:
For a given $\mu > 0$, solve the unconstrained problem of finding

$$x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^m \ln c_i(x)$$

IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \to +\infty$ find a sequence $(x_{\mu_k})$ of associated minimizers.

**Illustration**: minimize $f(x, y) = (x + 0.1)^2 + (y - 0.5)^2$ on a triangle
Barrier Function:
For a given $\mu > 0$, solve the unconstrained problem of finding

$$x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{m} \ln c_i(x)$$

IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \rightarrow 0$
find a sequence $(x_{\mu_k})$ of associated minimizers.

Illustration: minimize $f(x, y) = (x + 0.1)^2 + (y - 0.5)^2$ on a triangle
Barrier Function:
For a given $\mu > 0$, solve the unconstrained problem of finding

$$x_\mu = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{i=1}^{m} \ln c_i(x)$$

IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \to 0$
find a sequence $(x_{\mu_k})$ of associated minimizers.

**Illustration**: minimize $f(x, y) = (x + 0.1)^2 + (y - 0.5)^2$ on a triangle

Isovalues of $f$

$\log(\mu) = -5$
IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \to 0$ as $k \to +\infty$, find a sequence $(x_k)$ such that:

$$\forall k, \quad x_k = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu_k \sum_{i=1}^{m} \ln c_i(x)$$

Links with the original problem:
IP Methods: given a decreasing sequence \((\mu_k)\) with \(\mu_k \xrightarrow[k \to +\infty]{} 0\)
find a sequence \((x_k)\) such that:

\[
\forall k, \quad x_k = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu_k \sum_{i=1}^{m} \ln c_i(x)
\]

Links with the original problem:

Extremum point:

\[
\nabla f(x_k) - \sum_{i=1}^{m} \frac{\mu_k}{c_i(x_k)} \nabla c_i(x_k) = 0
\]
IP Methods: given a decreasing sequence \( (\mu_k) \) with  \( \mu_k \to +\infty \)
find a sequence \( (x_k) \) such that:

\[
\forall k, \quad x_k = \arg\min_{x \in \mathbb{R}^n} f(x) - \mu_k \sum_{i=1}^{m} \ln c_i(x)
\]

Links with the original problem:

Extremum point:

\[
\nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \text{ where } \lambda_k = \mu_k / c(x_k)
\]
IP Methods: given a decreasing sequence $(\mu_k)$ with $\mu_k \to 0$ as $k \to +\infty$, find a sequence $(x_k)$ such that:

$$\forall k, \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m$$

Links with the original problem:

Extremum point: $$\nabla f(x_k) - J_c(x_k)^T \lambda_k = 0 \quad c_i(x_k) \lambda_{k,i} = \mu_k$$
IP Methods: given a decreasing sequence \( (\mu_k) \) with \( \mu_k \to 0 \) as \( k \to +\infty \), find a sequence \( (x_k) \) such that:

\[
\forall k, \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \frac{\mu_k}{c(x_k)} \in \mathbb{R}^m
\]

Links with the original problem:

**Extremum point:**

\[
\nabla f(x_k) - J_c(x_k)^T \lambda_k = 0 \quad c_i(x_k) \lambda_{k,i} = \mu_k
\]

**KKT Conditions:**

\[
\nabla f(x^*) - J_c(x^*)^T \lambda^* = 0 \quad c(x^*) \geq 0
\]

\[
\forall i, \lambda^*_i \geq 0 \quad \forall i, c_i(x^*) \lambda^*_i = 0
\]
**IP Methods** : given a decreasing sequence \((\mu_k)\) with \(\mu_k \to 0\) as \(k \to +\infty\), find a sequence \((x_k)\) such that:

\[
\forall k, \quad \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \frac{\mu_k}{c(x_k)} \in \mathbb{R}^m
\]

**Links with the original problem**:

**Extremum point**:

\[
\nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad c_i(x_k)\lambda_{k,i} = \mu_k
\]

**KKT Conditions**:

\[
\nabla f(x^*) - J_c(x^*)^T \lambda^* = 0, \quad \mu_k \to 0
\]

\[
c(x^*) \geq 0
\]

\[
\forall i, \lambda^*_i \geq 0
\]

\[
\forall i, \quad c_i(x^*)\lambda^*_i = 0
\]
IP Methods: given a decreasing sequence \((\mu_k)\) with \(\mu_k \to 0\) as \(k \to +\infty\), find a sequence \((x_k)\) such that:

\[ \forall k, \quad \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m \]

Links with the original problem:
Extremum point: \( \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0 \quad c_i(x_k) \lambda_{k,i} = \mu_k \)

\( \exists (x^*, \lambda^*) \) KKT point and multipliers such that, Under some favorable assumptions:

\[ x_k \quad \to \quad x^* \quad k \to +\infty \]

With even more assumptions:

\[ \lambda_k \quad \to \quad \lambda^* \quad k \to +\infty \]
**IP main step:** given a decreasing sequence \((\mu_k)\) with \(\mu_k \to 0\) 
find a sequence \((x_k)\) such that:

\[
\forall k, \quad \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m
\]

**Links with the original problem:**

**Extremum point:**  
\[
\nabla f(x_k) - J_c(x_k)^T \lambda_k = 0 \quad \lambda_k = \mu_k / c(x_k)
\]

\(\exists (x^*, \lambda^*)\) KKT point and multipliers such that,

**Under some favorable assumptions:**  
\[
x_k \to x^* \quad k \to +\infty
\]

**With even more assumptions:**  
\[
\lambda_k \to \lambda^* \quad k \to +\infty
\]
**Interior Point Method**

**IP main step:** given a decreasing sequence $\{\mu_k\}$ with $\mu_k \to 0$ as $k \to +\infty$, find a sequence $\{x_k\}$ such that:

$$
\forall k, \quad \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m
$$

**Links with the original problem:**

- **Extremum point:**
  $$
  \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0 \quad \lambda_k = \mu_k / c(x_k)
  $$

- **KKT point and multipliers** such that,
  $$
  \exists (x^*, \lambda^*) \quad \text{KKT point and multipliers such that,}
  $$

  - **Under some favorable assumptions:**
    $$
    x_k \quad \to \quad x^* \quad k \to +\infty
    $$

  - **With even more assumptions:**
    $$
    \lambda_k \quad \to \quad \lambda^* \quad k \to +\infty
    $$

lundi 14 mai 12
IP main step: given a decreasing sequence \( (\mu_k) \) with \( \mu_k \rightarrow 0 \) as \( k \rightarrow +\infty \), find a sequence \( (x_k) \) such that:

\[
\forall k, \quad \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m
\]

Links with the original problem:

\( \exists (x^*, \lambda^*) \) KKT point and corresponding multipliers of the original problem, such that, under some assumptions:

\[
x_k \rightarrow x^*, \quad k \rightarrow +\infty
\]

\[
\lambda_k \rightarrow \lambda^*, \quad k \rightarrow +\infty
\]
**IP main step:** given a decreasing sequence \((\mu_k)\) with \(\mu_k \to 0\) as \(k \to +\infty\), find a sequence \((x_k)\) such that:

\[
\forall k, \quad \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m
\]

**Links with the original problem:**

\(\exists (x^*, \lambda^*)\) KKT point and corresponding multipliers of the original problem, such that, under some assumptions:

\[
x_k \quad \overset{k \to +\infty}{\longrightarrow} \quad x^*
\]

\[
\lambda_k \quad \overset{k \to +\infty}{\longrightarrow} \quad \lambda^*
\]
**IP main step:** given a decreasing sequence \((\mu_k)\) with \(\mu_k \to 0\) as \(k \to +\infty\), find a sequence \((x_k)\) such that:
\[
\forall k, \quad \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m
\]

**Links with the original problem:** \(\exists (x^*, \lambda^*)\) KKT point and corresponding multipliers of the original problem, such that, under some assumptions:
\[
x_k \to x^* \quad \text{as} \quad k \to +\infty \quad \lambda_k \to \lambda^* \quad \text{as} \quad k \to +\infty
\]
IP main step: given a decreasing sequence $\mu_k$ with $\mu_k \rightarrow 0$
find a sequence $(x_k)$ such that:

$$\forall k, \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m$$

Links with the original problem: $\exists (x^*, \lambda^*)$ KKT point
and corresponding multipliers of the original problem, such that, under some assumptions:

$$x_k \rightarrow x^* \quad \lambda_k \rightarrow \lambda^*$$
**IP main step:** given a decreasing sequence \((\mu_k)\) with \(\mu_k \to 0\) as \(k \to +\infty\), find a sequence \((x_k)\) such that:

\[
\forall k, \quad \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \frac{\mu_k}{c(x_k)} \in \mathbb{R}^m
\]

**Links with the original problem:** \(\exists (x^*, \lambda^*)\) KKT point and corresponding multipliers of the original problem, such that, under some assumptions:

\[
x_k \to x^* \quad \lambda_k \to \lambda^*
\]

**Primal-dual IP method** - Newton’s method is applied to the non-linear system:

\[
\begin{align*}
\nabla f(x) - J_c(x)^T \lambda &= 0 \\
c_i(x) \lambda_i &= \mu, \quad \forall i \leq m
\end{align*}
\]
Interior Point Method

Primal-dual IP method - Newton’s method is applied to the non-linear system:

\[
\begin{align*}
\nabla f(x) - J_c(x)^T \lambda &= 0 \\
c_i(x)\lambda_i &= \mu, \quad \forall i \leq m
\end{align*}
\]

Newton update \((\Delta x, \Delta \lambda)\):

\[
\begin{pmatrix}
H & -J_c^T \\
J_c & C
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta \lambda
\end{pmatrix}
= 
\begin{pmatrix}
\nabla f - J_c^T \lambda \\
C\lambda - \mu 1
\end{pmatrix}
\]

Where \(H = \nabla^2 f - \sum_{i=1}^{m} \lambda_i \nabla^2 c_i\) and \(C = (\delta_{ij}c_i)_{1 \leq i, j \leq m}\) (as for \(\Lambda\)).
**Interior Point Method**

**IP main step:** given a decreasing sequence \((\mu_k)\) with \(\mu_k \stackrel{k \to +\infty}{\to} 0\), find a sequence \((x_k)\) such that:

\[
\forall k, \ nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \; \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m
\]

**Links with the original problem:** \(\exists (x^*, \lambda^*)\) KKT point and corresponding multipliers of the original problem, such that, under some assumptions:

\[
x_k \stackrel{k \to +\infty}{\to} x^* \quad \lambda_k \stackrel{k \to +\infty}{\to} \lambda^*
\]

**Primal-dual IP method** - Newton’s method is applied to the non-linear system:

\[
\begin{align*}
\nabla f(x) - J_c(x)^T \lambda &= 0 \\
J_c(x) \lambda_i &= \mu, \; \forall i \leq m
\end{align*}
\]

**Newton update** \((\Delta x, \Delta \lambda)\):

\[
\begin{pmatrix}
H & -J_c^T \\
\Lambda J_c & C
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta \lambda
\end{pmatrix}
= 
\begin{pmatrix}
\nabla f - J_c^T \lambda \\
C \lambda - \mu 1
\end{pmatrix}
\]

Where \(H = \nabla^2 f - \sum_{i=1}^{m} \lambda_i \nabla^2 c_i\) and \(C = (\delta_{ij} c_i)_{1 \leq i, j \leq m}\) (as for \(\Lambda\)).
**IP main step:** given a decreasing sequence \((\mu_k)\) with \(\lim_{k \to +\infty} \mu_k = 0\), find a sequence \((x_k)\) such that:

\[
\forall k, \quad \nabla f(x_k) - J_c(x_k)^T \lambda_k = 0, \quad \lambda_k = \mu_k / c(x_k) \in \mathbb{R}^m
\]

**Links with the original problem:** \(\exists (x^*, \lambda^*)\) KKT point and corresponding multipliers of the original problem, such that, under some assumptions:

\[
x_k \xrightarrow[k \to +\infty]{} x^* \quad \lambda_k \xrightarrow[k \to +\infty]{} \lambda^*
\]

---

**Primal-dual IP method** - Newton’s method is applied to the non-linear system:

\[
\begin{aligned}
\nabla f(x) - J_c(x)^T \lambda &= 0 \\
c_i(x) \lambda_i &= \mu, \quad \forall i \leq m
\end{aligned}
\]

**Newton update** \((\Delta x, \Delta \lambda)\):

\[
\begin{pmatrix}
H & -J_c^T \\
\Lambda J_c & C
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta \lambda
\end{pmatrix}
= \begin{pmatrix}
\nabla f - J_c^T \lambda \\
C \lambda - \mu 1
\end{pmatrix}
\]

Where \(H = \nabla^2 f - \sum_{i=1}^{m} \lambda_i \nabla^2 c_i\) and \(C = (\delta_{ij} c_i)_{1 \leq i, j \leq m}\) (as for \(\Lambda\)).
The IPOPT software:

- C++ open source implementation of a primal-dual interior point method for non-linear smooth optimization, from the COIN-OR project
  - [https://projects.coin-or.org/Ipopt](https://projects.coin-or.org/Ipopt)
  - [the mailing list](https://projects.coin-or.org/Ipopt/wiki/IpoptPapers)

- Uses sparse matrices and sparse symetric linear solvers (mumps, PARDISO, etc...) for the Newton’s steps.
- Handles both inequality and/or equality constraints
- A fortran variant for variationnal inequalities...?

Bibliography:

- Some IPOPT related papers: [https://projects.coin-or.org/Ipopt/wiki/IpoptPapers](https://projects.coin-or.org/Ipopt/wiki/IpoptPapers)

lundi 14 mai 12
The FreeFem++ Interface:

\[
x^* = \arg\min_{x \in \mathbb{R}^n} f(x)
\]

\[
\forall i \leq n, \quad x_i^{lb} \leq x_i^* \leq x_i^{ub}
\]

\[
\forall i \leq m, \quad c_i^{lb} \leq c_i(x^*) \leq c_i^{ub}
\]
The FreeFem++ Interface:

\[
\begin{align*}
    x^* &= \arg\min \ f(x) \\
    x &\in \mathbb{R}^n \\
    \forall i \leq n, \ x_i^{\text{lb}} &\leq x_i^* \leq x_i^{\text{ub}} \\
    \forall i \leq m, \ c_i^{\text{lb}} &\leq c_i(x^*) \leq c_i^{\text{ub}}
\end{align*}
\]

⇒ Simple bounds constraints
The FreeFem++ Interface:

\[
\left\{
\begin{align*}
    x^* & = \arg\min_{x \in \mathbb{R}^n} f(x) \\
    \forall i \leq n, \quad & x^*_i \leq x_i^* \leq x_i^{ub} \\
    \forall i \leq m, \quad & c_i^{lb} \leq c_i(x^*) \leq c_i^{ub}
\end{align*}
\right.
\]

«True» constraints
The FreeFem++ Interface:

IPOPT needs \( f, \nabla f, c, J_c, x^{lb}, x^{ub}, c^{lb}, c^{ub} \) and...

\[
H : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathcal{M}_n(\mathbb{R})
\]

\[
(x, \lambda) \mapsto \nabla^2 f(x) - \sum_{i=1}^{m} \lambda_i \nabla^2 c_i(x)
\]

\[
\begin{cases}
  x^* = \arg\min_{x \in \mathbb{R}^n} f(x) \\
  \forall i \leq n, x_i^{lb} \leq x_i^* \leq x_i^{ub} \\
  \forall i \leq m, c_i^{lb} \leq c_i(x^*) \leq c_i^{ub}
\end{cases}
\]

lundi 14 mai 12
The FreeFem++ Interface:

IPOPT needs $f$, $\nabla f$, $c$, $J_c$, $x^{lb}$, $x^{ub}$, $c^{lb}$, $c^{ub}$ and...

$H : \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m \longrightarrow \mathcal{M}_n(\mathbb{R})$

$$(x, \sigma, \lambda) \longmapsto \sigma \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 c_i(x)$$
The FreeFem++ Interface:

IPOPT needs

\( f \), \( \nabla f, c \)

\( J_c \)

\( H \)
The FreeFem++ Interface:

IPOPT needs

\[
\begin{align*}
\nabla f, \ c & \\
J_c & \\
H & \\
\end{align*}
\]

\[
\begin{align*}
\begin{cases}
\quad x^* = \arg\min_{x \in \mathbb{R}^n} f(x) \\
\quad \forall i \leq n, \ x_i^{lb} \leq x_i^* \leq x_i^{ub} \\
\quad \forall i \leq m, \ c_i^{lb} \leq c_i(x^*) \leq c_i^{ub}
\end{cases}
\end{align*}
\]
The FreeFem++ Interface:

IPOPT needs

\[
\begin{align*}
\vec{x}^* &= \arg\min_{x \in \mathbb{R}^n} f(x) \\
\forall i \leq n, \ x_i^{\text{lb}} &\leq x_i^* \leq x_i^{\text{ub}} \\
\forall i \leq m, \ c_i^{\text{lb}} &\leq c_i(x^*) \leq c_i^{\text{ub}}
\end{align*}
\]

\[
\begin{align*}
J_c(x^*)
\end{align*}
\]

\[
H
\]
**The FreeFem++ Interface:**

**IPOPT needs**

\[
\begin{align*}
  f & : \text{func real } f(\text{real}[\text{int}] & \ X) \{\ldots\} \\
  \nabla f, \ c & : \text{func real}[\text{int}] \ df(\text{real}[\text{int}] & \ X) \{\ldots\} \\
  & \text{func real}[\text{int}] \ c(\text{real}[\text{int}] & \ X) \{\ldots\} \text{ (optional)} \\
  J_c & : \text{func matrix } Jc(\text{real}[\text{int}] & \ X) \{\ldots\}
\end{align*}
\]

Needed only if there are constraint functions

\[
\begin{align*}
  x^* = \arg\min_{x \in \mathbb{R}^n} f(x) \\
  \forall i \leq n, \ x_i^{\text{lb}} \leq x_i^* \leq x_i^{\text{ub}} \\
  \forall i \leq m, \ c_i^{\text{lb}} \leq c_i(x^*) \leq c_i^{\text{ub}} \\

  x^{\text{lb}}, \ x^{\text{ub}}, \ c^{\text{lb}}, \ c^{\text{ub}}
\end{align*}
\]

\[H\]
The FreeFem++ Interface:

**IPOPT needs**

\[ f : \text{func } \text{real } f(\text{real \[ int \]} &X) \{\ldots\} \]

\[ \nabla f, \ c : \text{func } \text{real \[ int \]} \ df(\text{real \[ int \]} &X) \{\ldots\} \]

\[ \text{func } \text{real \[ int \]} \ c(\text{real \[ int \]} &X) \{\ldots\} \text{ (optional)} \]

\[ J_c : \text{func } \text{matrix } Jc(\text{real \[ int \]} &X) \{\ldots\} \]

Needed only if there are constraint functions

\[ H : \text{func } \text{matrix } H(\text{real \[ int \]} &X, \text{real } s, \text{real \[ int \]} &L) \]

In case of affine constraints, the prototype may be:

\[ \text{func } \text{matrix } H(\text{real \[ int \]} &X) \{\ldots\} \]

If \( H \) is omitted Newton is replaced by a BFGS algorithm.

\[ x^* = \arg\min_{x \in \mathbb{R}^n} f(x) \]

\[ \forall i \leq n, \ x_i^{\text{lb}} \leq x_i^* \leq x_i^{\text{ub}} \]

\[ \forall i \leq m, \ c_i^{\text{lb}} \leq c_i(x^*) \leq c_i^{\text{ub}} \]

\[ x^{\text{lb}}, x^{\text{ub}}, c^{\text{lb}}, c^{\text{ub}} \]
The FreeFem++ Interface:

**IPOPT needs**

\[
\begin{align*}
    f & : \text{func real } f(\text{real[|int|]} & X) \{\ldots\} \\
    \nabla f, c & : \text{func real[|int|]} \ df(\text{real[|int|]} & X) \{\ldots\} \\
    & \quad \text{func real[|int|]} \ c(\text{real[|int|]} & X) \{\ldots\} \text{ (optional)} \\
    J_c & : \text{func matrix } Jc(\text{real[|int|]} & X) \{\ldots\} \\
    & \quad \text{Needed only if there are constraint functions} \\
    H & : \text{func matrix } H(\text{real[|int|]} & X, \text{ real } s, \text{ real[|int|]} & L) \\
    & \quad \text{In case of affine constraints, the prototype may be:} \\
    & \quad \text{func matrix } H(\text{real[|int|]} & X) \{\ldots\} \\
    & \quad \text{If } H \text{ is omitted Newton is replaced by a BFGS algorithm.}
\end{align*}
\]

The objective function \( x^* = \arg \min_{x \in \mathbb{R}^n} f(x) \)

\[
\forall i \leq n, \quad x_i^{\text{lb}} \leq x_i^* \leq x_i^{\text{ub}} \\
\forall i \leq m, \quad c_i^{\text{lb}} \leq c_i(x^*) \leq c_i^{\text{ub}}
\]

\[
\begin{align*}
    x^{\text{lb}}, x^{\text{ub}}, c^{\text{lb}}, c^{\text{ub}} : \text{real[|int|]} \text{ arrays} \\
    \bullet \text{size } n \text{ for } x \text{ bounds} \\
    \bullet \text{size } m \text{ for } c \text{ bounds} \\
    \bullet x \text{ bounds are optionnal} \\
    \bullet \text{Set components to } \pm 10^19 \\
    \text{for unboundedness in particular directions.}
\end{align*}
\]
The FreeFem++ Interface:

**IPOPT needs**

\[ f : \text{func real } f(\text{real[int]} &X) \{\ldots\} \]

\[ \nabla f, \ c : \text{func real[int]} df(\text{real[int]} &X) \{\ldots\} \]

\[ \text{func real[int]} \ c(\text{real[int]} &X) \{\ldots\} \text{ (optional)} \]

\[ J_c : \text{func matrix } Jc(\text{real[int]} &X) \{\ldots\} \]

Needed only if there are constraint functions

\[ H : \text{func matrix } H(\text{real[int]} &X, \text{real } s, \text{real[int]} &L) \]

In case of affine constraints, the prototype may be:

\[ \text{func matrix } H(\text{real[int]} &X) \{\ldots\} \]

If H is omitted Newton is replaced by a BFGS algorithm.

Call IPOPT: full featured

\[ \text{int status = IPOPT}(f, \text{dF}, \text{c, Jc, H, xstart, lb=xlb,ub=xub,clb=clb,cub=cub, ... ... }); \]

Remark: requires load "ff-Ipopt"; earlier in the script
The FreeFem++ Interface:

**IPOPT needs**

\[ f : \text{func real } f(\text{real}\,[\text{int}] \ & X) \{...\} \]

\[ \nabla f, \ c : \text{func real}\,[\text{int}] \ df(\text{real}\,[\text{int}] \ & X) \{...\} \]

\[ \text{func real}\,[\text{int}] \ c(\text{real}\,[\text{int}] \ & X) \{...\} \text{(optional)} \]

\[ J_c : \text{func matrix } Jc(\text{real}\,[\text{int}] \ & X) \{...\} \]

Needed only if there are constraint functions

\[ H : \text{func matrix } H(\text{real}\,[\text{int}] \ & X, \ \text{real } s, \ \text{real}\,[\text{int}] \ & L) \]

In case of affine constraints, the prototype may be:

\[ \text{func matrix } H(\text{real}\,[\text{int}] \ & X) \{...\} \]

If \( H \) is omitted Newton is replaced by a BFGS algorithm.

\[ x^* = \arg\min_{x \in \mathbb{R}^n} f(x) \]

\[ \forall i \leq n, \ x_i^{lb} \leq x_i^* \leq x_i^{ub} \]

\[ \forall i \leq m, \ c_i^{lb} \leq c_i(x^*) \leq c_i^{ub} \]

\[ x^{lb}, \ x^{ub}, \ c^{lb}, \ c^{ub} : \text{real}\,[\text{int}] \text{ arrays} \]

- size \( n \) for \( x \) bounds
- size \( m \) for \( c \) bounds
- \( x \) bounds are optional
- Set components to \( \pm 1e19 \) for unboundedness in particular directions.

Call IPOPT: unconstrained

\[ \text{int status} = \text{IPOPT}(f, \ df, \ H, \ xstart, \ lb=xlb, \ ub=xub, \ ... \ ... \ ); \]

Remark: requires \text{load} "ff-Ipopt"; earlier in the script
The FreeFem++ Interface:

**IPOPT needs**

\[ f : \text{func real } f(\text{real[int]} &X) \{\ldots}\] 
\[ \nabla f, \ c : \text{func real[int]} \ df(\text{real[int]} &X) \{\ldots\} \] 
\[ \text{func real[int]} \ c(\text{real[int]} &X) \{\ldots\} \text{ (optional)} \] 
\[ J_c : \text{func matrix } Jc(\text{real[int]} &X) \{\ldots\} \]

Needed only if there are constraint functions

\[ H : \text{func matrix } H(\text{real[int]} &X, \text{ real } s, \text{ real[int]} &L) \]

In case of affine constraints, the prototype may be:

\[ \text{func matrix } H(\text{real[int]} &X) \{\ldots\} \]
If H is omitted Newton is replaced by a BFGS algorithm.

**Call IPOPT : with BFGS**

\[ \text{int status = IPOPT}(f, \text{dF}, \text{c}, \text{xstart}, \text{lb}=\text{xl}, \text{ub}=\text{xb}, \text{clb}=\text{cl}, \text{ cub}=\text{cb} \ldots \ldots ); \]
Remark: requires load "ff-Ipopt"; earlier in the script
The FreeFem++ Interface:

IPOPT needs

\[ f : \text{func } \text{real } f(\text{real[\text{int}]} &X) \{\ldots\} \]

\[ \nabla f, \ c : \text{func } \text{real[\text{int}]} \ df(\text{real[\text{int}]} &X) \{\ldots\} \]
\[ \text{func } \text{real[\text{int}]} \ c(\text{real[\text{int}]} &X) \{\ldots\} \text{ (optional)} \]

\[ J_c : \text{func } \text{matrix } Jc(\text{real[\text{int}]} &X) \{\ldots\} \]

Needed only if there are constraint functions

\[ H : \text{func } \text{matrix } H(\text{real[\text{int}]} &X, \ \text{real } s, \ \text{real[\text{int}]} &L) \]

In case of affine constraints, the prototype may be:

\[ \text{func } \text{matrix } H(\text{real[\text{int}]} &X) \{\ldots\} \]

If \( H \) is omitted Newton is replaced by a BFGS algorithm.

\[ x^* = \arg \min_{x \in \mathbb{R}^n} f(x) \]
\[ \forall i \leq n, \ x_i^{lb} \leq x_i^* \leq x_i^{ub} \]
\[ \forall i \leq m, \ c_i^{lb} \leq c_i(x^*) \leq c_i^{ub} \]

\[ x^{lb}, x^{ub}, c^{lb}, c^{ub} : \text{real[\text{int}]} \text{ arrays} \]

- size \( n \) for \( x \) bounds
- size \( m \) for \( c \) bounds
- \( x \) bounds are optionnal
- Set components to \( \pm 1e19 \) for unboundedness in particular directions.

Call IPOPT: unconstrained with BFGS

\[ \text{int } \text{status } = \text{IPOPT}(f, dF, \text{xstart}, \text{lb} = x^{lb}, \text{ub} = x^{ub}, \ldots \ldots ) ; \]

Remark: requires \text{load } "ff-Ipopt"; earlier in the script
The FreeFem++ Interface: case of quadratic objective and affine constraints

\[
\begin{align*}
    x^* &= \arg\min_{x \in \mathbb{R}^n} f(x) \\
    \forall i \leq n, \quad &x_i^{lb} \leq x_i^* \leq x_i^{ub} \\
    \forall i \leq m, \quad &c_i^{lb} \leq c_i(x^*) \leq c_i^{ub}
\end{align*}
\]
The FreeFem++ Interface: case of quadratic objective and affine constraints

\[ \forall x \in \mathbb{R}^n, \quad f(x) = \frac{1}{2} \langle Ax, x \rangle + \langle b, x \rangle \]

\[ c(x) = Cx + d \]

\[ (A, b) \in \mathcal{M}_{n,n}(\mathbb{R}) \times \mathbb{R}^n, \quad (C, d) \in \mathcal{M}_{n,m}(\mathbb{R}) \times \mathbb{R}^m \]

\[ x^* = \arg\min_{x \in \mathbb{R}^n} f(x) \]

\[ \forall i \leq n, \quad x_i^{\text{lb}} \leq x_i^* \leq x_i^{\text{ub}} \]

\[ \forall i \leq m, \quad c_i^{\text{lb}} \leq c_i(x^*) \leq c_i^{\text{ub}} \]
The FreeFem++ Interface: case of quadratic objective and affine constraints

\[
\forall x \in \mathbb{R}^n, \quad f(x) = \frac{1}{2} \langle Ax, x \rangle + \langle b, x \rangle \\
c(x) = Cx + d
\]

\((A, b) \in \mathcal{M}_{n,n}(\mathbb{R}) \times \mathbb{R}^n, \quad (C, d) \in \mathcal{M}_{n,m}(\mathbb{R}) \times \mathbb{R}^m\)

\(\mapsto\) Possibility to directly pass the matrices and vectors
The FreeFem++ Interface: case of quadratic objective and affine constraints

\[ \forall x \in \mathbb{R}^n, \quad f(x) = \frac{1}{2} \langle Ax, x \rangle + \langle b, x \rangle \]
\[ c(x) = Cx + d \]

\[ (A, b) \in \mathcal{M}_{n,n}(\mathbb{R}) \times \mathbb{R}^n, \quad (C, d) \in \mathcal{M}_{n,m}(\mathbb{R}) \times \mathbb{R}^m \]

⇒ Possibility to directly pass the matrices and vectors

Quadratic objective and affine constraints:

```plaintext
... //begining of the script
matrix A = ... ;
matrix C = ... ;
real[int] b = ... , d = ... ;
IPOPT([A,b] , [C,d] , xstart , /*named parameters*/ ) ;
```
The FreeFem++ Interface : case of quadratic objective and affine constraints

\[
\forall x \in \mathbb{R}^n, \quad f(x) = \frac{1}{2} \langle Ax, x \rangle + \langle b, x \rangle \\
c(x) = Cx + d
\]

\((A, b) \in \mathcal{M}_{n,n}(\mathbb{R}) \times \mathbb{R}^n, \quad (C, d) \in \mathcal{M}_{n,m}(\mathbb{R}) \times \mathbb{R}^m\)

\[x^* = \arg\min_{x \in \mathbb{R}^n} f(x)\]
\[\forall i \leq n, \quad x^*_i^{lb} \leq x^*_i \leq x^*_i^{ub}\]
\[\forall i \leq m, \quad c_i^{lb} \leq c_i(x^*) \leq c_i^{ub}\]

⇒ Possibility to directly pass the matrices and vectors

Homogeneous quadratic objective and affine constraints :

... //begining of the script

matrix A = ... ;
matrix C = ... ;
real[int] d = ... ;
IPOPT(A, [C,d], xstart, /*named parameters*/ );
The FreeFem++ Interface: case of quadratic objective and affine constraints

$$\forall x \in \mathbb{R}^n, \quad f(x) = \frac{1}{2} \langle Ax, x \rangle + \langle b, x \rangle$$

$$c(x) = Cx + d$$

$$(A, b) \in \mathcal{M}_{n,n}(\mathbb{R}) \times \mathbb{R}^n, \quad (C, d) \in \mathcal{M}_{n,m}(\mathbb{R}) \times \mathbb{R}^m$$

⇒ Possibility to directly pass the matrices and vectors

Standard objective and affine constraints:

```c++
... //beginning of the script
func real f(real[int] &X) {...}
func real[int] df(real[int] &X) {...}
func matrix H(real[int] &X) {...} //no need for the full prototype
matrix C = ... ;
real[int] d = ... ;
IPOPT(f, df, H, [C,d] , xstart , /*named parameters*/  );
```
The FreeFem++ Interface: case of quadratic objective and affine constraints

\[ \forall x \in \mathbb{R}^n, \quad f(x) = \frac{1}{2} \langle Ax, x \rangle + \langle b, x \rangle \]

\[ c(x) = Cx + d \]

\[(A, b) \in \mathcal{M}_{n,n}(\mathbb{R}) \times \mathbb{R}^n, \quad (C, d) \in \mathcal{M}_{n,m}(\mathbb{R}) \times \mathbb{R}^m\]

⇒ Possibility to directly pass the matrices and vectors

Quadratic objective and standard constraints: hazardous case

```cpp
... //begining of the script
func real[int] c(real[int] &X) {...}
func matrix Jc(real[int] &X) {...}
matrix A = ... ;
real[int] b = ... ;
IPOPT([A,b] , c, Jc, xstart , /*named parameters*/  );
```
The FreeFem++ Interface: case of quadratic objective and affine constraints

\[ \forall x \in \mathbb{R}^n, \quad f(x) = \frac{1}{2} \langle Ax, x \rangle + \langle b, x \rangle \]
\[ c(x) = Cx + d \]

\((A, b) \in \mathcal{M}_{n,n}(\mathbb{R}) \times \mathbb{R}^n, \quad (C, d) \in \mathcal{M}_{n,m}(\mathbb{R}) \times \mathbb{R}^m\)

\[ x^* = \arg \min_{x \in \mathbb{R}^n} f(x) \]
\[ \forall i \leq n, \quad x^*_{lb} \leq x^*_i \leq x^*_{ub} \]
\[ \forall i \leq m, \quad c_{lb}^i \leq c_i(x^*) \leq c_{ub}^i \]

⇒ Possibility to directly pass the matrices and vectors

Quadratic objective and standard constraints: hazardous case

... //begining of the script
func real[int] c(real[int] &X) {...}
func matrix Jc(real[int] &X) {...}
matrix A = ... ;
real[int] b = ... ;

IPOPT([A,b] , c, Jc, xstart , /*named parameters*/  );

Here, \( H(x, \sigma, \lambda) \) is always equal to \( A \)
⇒ erroneous hessian in some cases
The FreeFem++ Interface: some hints and tips
The FreeFem++ Interface: some hints and tips

Matrix returning functions: must return a global object

```cpp
func matrix H(real[int] &X)
{
    matrix M;
    ...
    //compute M
    return M;
}
```
The FreeFem++ Interface: some hints and tips

Matrix returning functions: must return a global object

```cpp
func matrix H(real[int] &X) {
    matrix M;
    ...
    //compute M
    return M;
}
```

M is cleaned after closing the function block, so the returned object does not exist anymore...
The FreeFem++ Interface: some hints and tips

**Sparse matrix returning functions:** must return a global object

```cpp
matrix M; // just declare a global matrix
func matrix H(real[int] &X)
{
    ...
    // do something to fill M
    return M; // M will not be deleted before H goes out of scope
}
```
The FreeFem++ Interface: some hints and tips

**Sparse matrix returning functions**: must return a global object

```cpp
matrix M; //just declare a global matrix
func matrix H(real[int] &X)
{
    //do something to fill M
    return M;  //M will not be deleted before H goes out of scope
}
```

**The structure of the matrices**: has to be constant through the optimization process...

- Use `varf` as much as possible to build matrices, matrices built with the same `varf` always have the same structure even if some coefficients becomes null.
- Remember that zeros are discarded when building sparse matrices from arrays:
  ```cpp
  real[int] c=...;
  int[int] I=..., J=...;
  matrix M = [I, J ,c];
  //two dangerous methods
  ```
  ```cpp
  real[int,int] tmp=...;
  matrix M = tmp;
  ```
The FreeFem++ Interface: some hints and tips (2)
The FreeFem++ Interface: some hints and tips (2)

**Named parameters:** subset of the IPOPT parameters, plus a few FreeFem++ specific ones, which can be used for:

- Stopping criteria (tol, maxiter, maxcputime, etc...)
- Getting final values (objvalue, lm, uz, lz, etc...)
- Perform a warm start
- Call a derivatives checker
- Force the BFGS mode (add bfgs=1)
- Specify an extern option file (see the [IPOPT documentation](#))
- etc...

Check the documentation for more informations.
The FreeFem++ Interface : some hints and tips (2)

**Named parameters** : subset of the IPOPT parameters, plus a few FreeFem++ specific ones, which can be use for :
- Stopping criteria (tol, maxiter, maxcputime, etc...)
- Getting final values (objvalue, lm, uz, lz, etc...)
- Perform a warm start
- Call a derivatives checker
- Force the BFGS mode (add bfgs=1)
- Specify an extern option file (see the IPOPT documentation)
- etc...

Check the documentation for more informations.

**Returned value** : the IPOPT function returns an int revealing what happens during the optimization. Positive values often means IPOPT performed well and negative values are relevant of troubles.
A Quadratic Example:
A Quadratic Example :

**Example 0.3 (IpopoVI2.edp)** Let $\Omega$ be a domain of $\mathbb{R}^2$, $f_1, f_2 \in L^2(\Omega)$ and $g_1, g_2 \in L^2(\partial\Omega)$ four given functions with $g_1 \leq g_2$ almost everywhere. We define the space:

$$V = \{ (v_1, v_2) \in H^1(\Omega)^2; v_1|_{\partial\Omega} = g_1, v_2|_{\partial\Omega} = g_2, v_1 \leq v_2 \text{ a.e. } \}$$

as well as the functional $J : H^1(\Omega)^2 \rightarrow \mathbb{R}$:

$$J(v_1, v_2) = \frac{1}{2} \int_{\Omega} |\nabla v_1|^2 - \int_{\Omega} f_1 v_1 + \frac{1}{2} \int_{\Omega} |\nabla v_2|^2 - \int_{\Omega} f_2 v_2$$

The problem consists in finding (numerically) two functions $(u_1, u_2) = \arg\min_{(v_1, v_2) \in V} J(v_1, v_2).$
A Quadratic Example:

**Example 0.3 (IpoptVI2.edp)** Let $\Omega$ be a domain of $\mathbb{R}^2$, $f_1, f_2 \in L^2(\Omega)$ and $g_1, g_2 \in L^2(\partial \Omega)$ four given functions with $g_1 \leq g_2$ almost everywhere. We define the space:

$$V = \{(v_1, v_2) \in H^1(\Omega)^2; v_1|_{\partial \Omega} = g_1, v_2|_{\partial \Omega} = g_2, v_1 \leq v_2 \text{ a.e.}\}$$

as well as the functional $J : H^1(\Omega)^2 \rightarrow \mathbb{R}$:

$$J(v_1, v_2) = \frac{1}{2} \int_\Omega |\nabla v_1|^2 - \int_\Omega f_1 v_1 + \frac{1}{2} \int_\Omega |\nabla v_2|^2 - \int_\Omega f_2 v_2$$

The problem consists in finding (numerically) two functions $(u_1, u_2) = \arg \min_{(v_1, v_2) \in V} J(v_1, v_2)$. 

![Graphical representation of a quadratic problem](image)
A Constrained Minimal-Surface Example:
A Constrained Minimal-Surface Example:

Given $g$, let $E = \{ v \in L^2(\Omega) \mid v - g \in H^1_0(\Omega) \}$, and:

$$\forall v \in E, \quad S(v) = \int_{\Omega} \sqrt{1 + |\nabla v|^2} , \quad V(v) = \int_{\Omega} v$$

And, given $V_0 > 0$, try to find:

$$u = \arg\min_{v \in E, \quad V(v) \geq V_0} S(v)$$
A Constrained Minimal-Surface Example:

Given $g$, let $E = \{ v \in L^2(\Omega) \mid v - g \in H^1_0(\Omega) \}$, and:

$$
\forall v \in E, \quad S(v) = \int_{\Omega} \sqrt{1 + \vert \nabla v \vert^2}, \quad V(v) = \int_{\Omega} v
$$

And, given $V_0 > 0$, try to find:

$$
u = \arg\min_{v \in E} S(v)
$$

$V(v) \geq V_0$

Computations with $g = x^2y^2$, $\Omega = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq 1\}$

$V_0 = 0$, $V_0 = \frac{1}{2}$, $V_0 = 1$, $V_0 = 2$