Constrained optimisation

Numerical optimisation

Francesco Corona

Constrained optimisation

Numerical optimisation

Constrained optimisation

Two strategies for solving constrained minimisation problems

- **The penalty method**: Problems with both equality and inequality constraints
- **The augmented Lagrangian method**: Problems with equality constraints only

The two methods allow the solution of simple problems and provide basic tools for more robust and complex algorithms

Constrained optimisation (cont.)

Definition

Let $f : \mathbb{R}^n \to \mathbb{R}$ with $n \geq 1$ be a **cost** or **objective function**

The **constrained optimisation** problem is

$$\min_{\mathbf{x} \in \Omega \subset \mathbb{R}^n} f(\mathbf{x}) \tag{1}$$

The closed subset Ω is determined by either equality and inequality constraints that are dictated by the nature of the problem to solve

1. Given functions $h_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, \dots, p$

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : h_i(\mathbf{x}) = 0, \text{ for } i = 1, \dots, p\} \tag{2}$$

2. Given functions $g_j : \mathbb{R}^n \to \mathbb{R}$ for $j = 1, \dots, g$

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : g_j(\mathbf{x}) \geq 0, \text{ for } j = 1, \dots, q\} \tag{3}$$

p and q are natural numbers

Constrained optimisation (cont.)

Definition

$$\min_{\mathbf{x} \in \Omega \subset \mathbb{R}^n} f(\mathbf{x})$$

In general, Ω is defined by both equality and inequality constraints

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : h_i(\mathbf{x}) = 0 \text{ for } i \in \mathcal{I}_h, g_j(\mathbf{x}) \geq 0 \text{ for } j \in \mathcal{I}_g\}$$

The two sets $\mathcal{I}_h$ and $\mathcal{I}_g$ are st $\mathcal{I}_h = \emptyset$ in Eq. 3 and $\mathcal{I}_g = \emptyset$ in Eq. 2

Constrained optimisation (cont.)

The constrained optimisation problem can thus be rewritten

Definition

$$\begin{aligned}
\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) & \text{ subjected to} \\
h_i(\mathbf{x}) = 0, \forall i \in \mathcal{I}_h \\
g_j(\mathbf{x}) \geq 0, \forall j \in \mathcal{I}_g
\end{aligned} \tag{4}$$

Constrained optimisation (cont.)

We assume that $f \in \mathbb{C}^1(\mathbb{R}^n)$, and also h_i and g_j are $\mathbb{C}^1(\mathbb{R}^n)$, $\forall i, j$

- Points $\mathbf{x} \in \Omega$ are said to be **admissible** as they fulfil all the constraints
- Ω is the set of all admissible points

Constrained optimisation (cont.)

A point $\mathbf{x}^* \in \Omega \subset \mathbb{R}^n$ is a **global minimiser** for the problem if

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \tag{5}$$

A point $\mathbf{x}^* \in \Omega \subset \mathbb{R}^n$ is a **local minimiser** for the problem if there is a ball $B_r(\mathbf{x}) \in \mathbb{R}^n$ with radius $r > 0$ and centred in $\mathbf{x}^*$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in B_r(\mathbf{x}^*) \cap \Omega \tag{6}$$

Constrained optimisation (cont.)

A constraint is **active** at $\mathbf{x} \in \Omega$ if it is satisfied with equality at $\mathbf{x}$

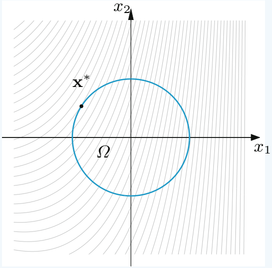
- According to this definition, active constraints at $\mathbf{x}$ are all the h_i as well as those g_j such that $g_j(\mathbf{x}) = 0$

Constrained optimisation (cont.)

Consider the following constrained optimisation problems

Example

Minimise $f(\mathbf{x})$ with $f(\mathbf{x}) = \dfrac{3}{5}x_1^2 + \dfrac{1}{2}x_1 x_2 - x_2 + 3x_1$, under the equality constraint $h_1(\mathbf{x}) = x_1^2 + x_2^2 - 1 = 0$

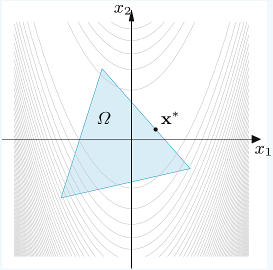


- Contour lines of the cost $f(\mathbf{x})$
- Admissibility set $\Omega \in \mathbb{R}^2$
- The global minimiser $\mathbf{x}^*$ constrained to Ω

Constrained optimisation (cont.)

Example

Minimise $f(\mathbf{x})$ with $f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$, under the following inequality constraints



$$g_1(\mathbf{x}) = -34x_1 - 30x_2 + 19 \geq 0$$
$$g_2(\mathbf{x}) = +10x_1 - 05x_2 + 11 \geq 0$$
$$g_3(\mathbf{x}) = +03x_1 + 22x_2 + 08 \geq 0$$

- Contour lines of the cost $f(\mathbf{x})$
- Admissibility set $\Omega \in \mathbb{R}^2$
- The global minimiser $\mathbf{x}^*$ constrained to Ω

Constrained optimisation (cont.)

If Ω is a non-empty, bounded and closed set, Weierstrass theorem guarantees the existence of a maximum and a minimum for f in Ω

- Consequently, problem in Definition 4 admits a solution

Constrained optimisation (cont.)

Definition

We recall that a function $f : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$ is **strongly convex** in Ω if there exists a $\rho > 0$ such that $\forall \mathbf{x}, \mathbf{y} \in \Omega$ and $\forall \alpha \in [0, 1]$, we have

$$f(\alpha \mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}) - \alpha(1-\alpha)\rho ||\mathbf{x} - \mathbf{y}||^2 \quad (7)$$

This reduces to the usual definition of convexity when $\rho = 0$

Constrained optimisation (cont.)

Proposition

Optimality conditions

Let $\Omega \subset \mathbb{R}^n$ be a convex set, $\mathbf{x}^* \in \Omega$ be such that $f \in \mathbb{C}^1(B_r(\mathbf{x}^*))$

If $\mathbf{x}^*$ is a local minimiser for the constrained minimisation problem,

$$\text{then, } \nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) \geq 0, \quad \forall \mathbf{x} \in \Omega \tag{8}$$

If f is convex in Ω and (8) is satisfied, then $\mathbf{x}^*$ is a global minimiser

Under the additional requirement for Ω to be closed and for f to be strongly convex, it can be shown that the minimiser is unique

Constrained optimisation (cont.)

Many algos for solving constrained minimisation problems can be related to the search of the stationary points of the **Lagrangian function** (the so-called **KKT** or **Karush-Kuhn-Tucker points**)

Definition

The **Lagrangian function** associated with problem $\min_{\mathbf{x} \in \Omega} f(\mathbf{x})$ is

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) - \sum_{j \in \mathcal{I}_g} \mu_j g_j(\mathbf{x}) \qquad (9)$$

where $\boldsymbol{\lambda} = (\lambda_i)$ for $i \in \mathcal{I}_h$ and $\boldsymbol{\mu} = (\mu_i)$ for $j \in \mathcal{I}_g$ are **Lagrangian multipliers** associated with the equality and inequality constraints

Constrained optimisation (cont.)

Definition

Karush-Kuhn-Tucker conditions

Point $\mathbf{x}^*$ is a KKT point for $\mathcal{L}$ if there exist $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that the triplet $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ satisfies the following conditions

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) - \sum_{j \in \mathcal{I}_g} \mu_j^* \nabla g_j(\mathbf{x}^*) = \mathbf{0}$$

$$\begin{aligned} h_i(\mathbf{x}^*) &= 0, \quad \forall i \in \mathcal{I}_h \\ g_i(\mathbf{x}^*) &= 0, \quad \forall j \in \mathcal{I}_g \\ \mu_j^* &\geq 0, \quad \forall j \in \mathcal{I}_g \\ \mu^*_J g_j(\mathbf{x}^*) &= 0, \quad \forall j \in \mathcal{I}_g \end{aligned}$$

Constrained optimisation (cont.)

Definition

For a $\mathbf{x}$, constraints satisfy a **linear independence** (**constraint**) **qualification** (**LI**(**C**)**Q**) in $\mathbf{x}^*$, if the gradients $\nabla h_i(\mathbf{x})$ and $\nabla g_j(\mathbf{x})$ associated with the active constraints in $\mathbf{x}$ are linearly independent

Constrained optimisation (cont.)

Theorem

First order KKT conditions

If $\mathbf{x}^*$ is a local minimum for the constrained problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \text{ subjected to}$$

$$h_i(\mathbf{x}) = 0, \forall i \in \mathcal{I}_h$$
$$g_j(\mathbf{x}) \geq 0, \forall j \in \mathcal{I}_g$$

if f, h_i and g_j are $\mathbb{C}^1(\Omega)$, if the constraints are LIQ in $\mathbf{x}^*$, then there exist $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ is a KKT point

As a consequence, local minima must be searched among KKT points and among points that do not satisfy LICQ conditions

Constrained optimisation (cont.)

Note that in the absence of inequality constraints, the Lagrangian function takes the form $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*)$

- The KKT conditions are Lagrange (necessary) conditions

$$\begin{aligned} \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) &= \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0} \\ h_i(\mathbf{x}^*) &= 0, \forall i \in \mathcal{I}_h \end{aligned} \tag{10}$$

Remark

Sufficient conditions for a KKT point to be a minimiser of f in Ω require knowledge about the Hessian of the Lagrangian or, alternatively, strict convexity hypothesis on f and the constraints

Constrained optimisation (cont.)

In general, it is possible to reformulate a constrained optimisation problem in the form of an unconstrained optimisation problem

- **Penalty function**
- **Augmented Lagrangian**

The penalty method

Constrained optimisation

The penalty method

A strategy for solving a general constrained optimisation problem

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) &\text{ subjected to} \\ h_i(\mathbf{x}) &= 0, \forall i \in \mathcal{I}_h \\ g_j(\mathbf{x}) &\geq 0, \forall j \in \mathcal{I}_g \end{aligned}$$

is to reformulate it as a new unconstrained optimisation problem

Definition

$$\mathcal{P}_\alpha(\mathbf{x}) = f(\mathbf{x}) + \frac{\alpha}{2} \sum_{i \in \mathcal{I}_h} h_i^2(\mathbf{x}) + \frac{\alpha}{2} \sum_{j \in \mathcal{I}_g} \Big(\max\{-g_j(\mathbf{x}), 0\} \Big)^2 \quad (11)$$

a modified **penalty function**, for a **penalty parameter** $\alpha > 0$

- When the constraints are not satisfied at $\mathbf{x}$, the sums quantify how far point $\mathbf{x}$ is from the admissibility set Ω
- A large α heavily penalises such a violation

The penalty method (cont.)

If $\mathbf{x}^*$ is a solution, clearly $\mathbf{x}^*$ must also be a minimiser of $\mathcal{P}$

Conversely, under some regularity hypothesis for f, h_i and g_i,

$$\lim_{\alpha \to \infty} \mathbf{x}^*(\alpha) = \mathbf{x}^*,$$

in which $\mathbf{x}^*(\alpha)$ denotes a minimiser of $\mathcal{P}_\alpha(\mathbf{x})$

The penalty method (cont.)

Due to numerical instability, it is not advised to minimise $\mathcal{P}_\alpha(\mathbf{x})$ directly for a large value of α

- Rather, consider an increasing and unbounded series of parameters $\{\alpha_k\}$
- For each α_k, calculate an approximation $\mathbf{x}^{(k)}$ of the solution $\mathbf{x}^*(\alpha_k)$ of $\min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$

$$\mathbf{x}^{(k)} = \arg\min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$$

  with an unconstrained optimisation method
- At each step k, α_{k+1} is a chosen as a function of α_k (e.g., $\alpha_{k+1} = \delta\alpha_k$, for $\delta \in [1.5, 2]$) and $\mathbf{x}^{(k)}$ is used as initial point for solving the minimisation at step $k+1$

The penalty method (cont.)

In the first iterations there is no reason to believe that the solution to $\min_{\mathbf{x}\in\mathbb{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$ should resemble the solution to the original problem

- This supports the idea of searching for an inexact solution to $\min_{\mathbf{x}\in\mathbb{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$ that differs from the exact one, $\mathbf{x}^{(k)}$, a small ε_k

The penalty method (cont.)

- Given α_0, (typically, $\alpha_0 = 1$), ε_0 (typically $\varepsilon_0 = 1/10$), $\overline{\varepsilon} > 0$, $\mathbf{x}_0^{(0)} \in \mathbb{R}^n$ and $\boldsymbol{\lambda}_0^{(0)} \in \mathbb{R}^p$ for $k = 0, 1, \dots$ until convergence

Pseudocode

Compute an approx. solution $\mathbf{x}^{(k)} = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$ to

$\min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$, by using initial point $\mathbf{x}_0^{(0)}$ and tolerance ε_k

If $||\nabla_{\mathbf{x}} \mathcal{L}_A(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \alpha_k)|| \leq \overline{\varepsilon}$
    Set $\mathbf{x}^* = \mathbf{x}^{(k)}$ (convergence)
else
    Choose $\alpha_{k+1} > \alpha_k$
    Choose $\varepsilon_{k+1} < \varepsilon_k$
    Set $\mathbf{x}_0^{(k+1)} = \mathbf{x}^{(k)}$
Endif

Note the extra tolerance $\overline{\varepsilon}$ to assess the gradient of $\mathcal{P}_{\alpha_k}$ at $\mathbf{x}^{(k)}$

The penalty method (cont.)

```
% PENALTY Constrained optimisation with penalty function
% [X,ERR,K]=PFUNCTION(F,GRAD_F,H,GRAD_H,G,GRAD_G,X_0,TOL,...
%                     KMAX,KMAXD,TYP)
%  Approximate a minimiser of the cost function F
%  under constraints H=0 and G>=0
%
% X0 is initial point, TOL is tolerance for stop check
% KMAX is the maximum number of iterations
% GRAD_F, GRAD_H, and GRAD_G are the gradients of F, H, and G
% H and G, GRAD_H and GRAD_G can be initialised to []
%
% For TYP=0 solution by FMINSEARCH M-function
%
% For TYP>0 solution by a DESCENT METHOD
%  KMAXD is maximum number of iterations
%  TYP is the choice of descent directions
%  TYP=1 and TYP=2 need the Hessian (or an approx. at k=0)
%  [X,ERR,K]=PFUNCTION(F,GRAD_F,H,GRAD_H,G,GRAD_G,X_0,TOL,...
%                     KMAX,KMAXD,TYP,HESS_FUN)
%  For TYP=1 HESS_FUN is the function handle associated
%  For TYP=2 HESS_FUN is a suitable approx. of Hessian at k=0
```

The penalty method (cont.)

```
function [x,err,k]=pFunction(f,grad_f,h,grad_h,g,grad_g,...
                              x_0,tol,kmax,kmaxd,typ,varargin)

xk=x_0(:); mu_0=1.0;

if typ==1; hess=varargin{1};
 elseif typ==2; hess=varargin{1};
 else; hess=[]; end
if ~isempty(h), [nh,mh]=size(h(xk)); end
if ~isempty(g), [ng,mg]=size(g(xk)); end

err=1+tol; k=0; muk=mu_0; muk2=muk/2; told=0.1;

while err>tol && k<kmax
 if typ==0
  options=optimset('TolX',told);
  [x,err,kd]=fminsearch(@P,xk,options); err=norm(x-xk);
 else
  [x,err,kd]=dScent(@P,@grad_P,xk,told,kmaxd,typ,hess);
  err=norm(grad_P(x));
 end

 if kd<kmaxd; muk=10*muk; muk2=0.5*muk;
 else muk=1.5*muk; muk2=0.5*muk; end

 k=1+k; xk=x; told=max([tol,0.10*told]);
end
```

The penalty method (cont.)

```
function y=P(x) % This function is nested inside pFunction

y=fun(x);
if ~isempty(h); y=y+muk2*sum((h(x)).^2); end
if ~isempty(g); G=g(x);
 for j=1:ng
  y=y+muk2*max([-G(j),0])^2;
 end
end
```

```
function y=grad_P(x) % This function is nested in pFunction

y=grad_fun(x);
if ~isempty(h), y=y+muk*grad_h(x)*h(x); end
if ~isempty(g), G=g(x); Gg=grad_g(x);
 for j=1:ng
  if G(j)<0
   y=y+muk*Gg(:,j)*G(j);
  end
 end
end
```

The augmented Lagrangian

Constrained optimisation

The augmented Lagrangian

Consider minimisation problems with equality constraints ($\mathcal{I}_g = \emptyset$)

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) & \text{ subjected to} \\ h_i(\mathbf{x}) &= 0, \forall i \in \mathcal{I}_h \\ g_j(\mathbf{x}) &\geq 0, \forall j \in \mathcal{I}_g \end{aligned}$$

Definition

For a suitable coefficient $\alpha > 0$, define the **augmented Laplacian**

$$\mathcal{L}_A(\mathbf{x}, \boldsymbol{\lambda}, \alpha) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) + \frac{\alpha}{2} \sum_{i \in \mathcal{I}_h} h_i^2(\mathbf{x}) \qquad (12)$$

The augmented Lagrangian (cont.)

The augmented Laplacian method is an iterative method that, at the k-th iteration and for a given α_k and a given $\boldsymbol{\lambda}^{(k)}$ computes

$$\mathbf{x}^{(k)} = \underset{\mathbf{x} \in \mathbb{R}^n}{\arg\min}\, \mathcal{L}_A(\mathbf{x}, \boldsymbol{\lambda}^{(k)}, \alpha_k) \tag{13}$$

in such a way that the sequence $\mathbf{x}^{(k)}$ converges to the KKT point for the Lagrangian $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x})$

The augmented Lagrangian (cont.)

Initial α_0 and $\boldsymbol{\lambda}^{(0)}$ are set arbitrarily and new values are given by

- Coefficient α_{k+1} is obtained from α_k, such that $\alpha_{k+1} > \alpha_k$
- To set $\boldsymbol{\lambda}^{(k+1)}$, compute the gradient of the augmented Lagrangian wrt $\mathbf{x}$ $\nabla_{\mathbf{x}} \mathcal{L}_A(\mathbf{x}, \boldsymbol{\lambda}^{(k)}, \alpha_k)$ and set it to zero

The augmented Lagrangian (cont.)

$$\nabla_{\mathbf{x}} \mathcal{L}_A(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \alpha_k) = \nabla f(\mathbf{x}^{(k)}) - \sum_{i \in \mathcal{I}_h} \Big(\lambda_i^{(k)} - \alpha_k h_i(\mathbf{x}^{(k)}) \Big) \nabla h_i(\mathbf{x}^{(k)})$$

We identify $\lambda_i^{(k)}$, by comparison with optimality condition

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0}$$

$$h_i(\mathbf{x}^*) = 0, \quad \forall i \in \mathcal{I}_h$$

The augmented Lagrangian (cont.)

The comparison yields $\lambda_i^{(k)} - \alpha_k h_i(\mathbf{x}^{(k)}) \simeq \lambda_i^*$ and we define

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} - \mu_k h_i(\mathbf{x}^{(k)}) \tag{14}$$

We identify $\mathbf{x}^{(k+1)}$ by solving with k replaced by $k+1$

$$\mathbf{x}^k = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \mathcal{L}_A(\mathbf{x}, \boldsymbol{\lambda}^k, \alpha_k)$$

The augmented Lagrangian (cont.)

- Given α_0, (typically, $\alpha_0 = 1$), ε_0 (typically $\varepsilon_0 = 1/10$), $\overline{\varepsilon} > 0$, $\mathbf{x}_0^{(0)} \in \mathbb{R}^n$ and $\boldsymbol{\lambda}_0^{(0)} \in \mathbb{R}^p$ for $k = 0, 1, \dots$ until convergence

Pseudocode

Compute an approx. solution $\mathbf{x}^{(k)} = \underset{\mathbf{x} \in \mathbb{R}^n}{\arg\min}\, \mathcal{L}_A(\mathbf{x}, \boldsymbol{\lambda}^{(k)}, \alpha_k)$,

by using initial point $\mathbf{x}_0^{(0)}$ and a tolerance ε_k

If $||\nabla_{\mathbf{x}} \mathcal{L}_A(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \alpha_k)|| \leq \overline{\varepsilon}$
    Set $\mathbf{x}^* = \mathbf{x}^{(k)}$ (convergence)
else
    Compute $\lambda_i^{(k+1)} = \lambda_i^{(k)} - \mu_k h_i(\mathbf{x}^{(k)})$
    Choose $\alpha_{k+1} > \alpha_k$
    Choose $\varepsilon_{k+1} < \varepsilon_k$
    Set $\mathbf{x}_0^{(k+1)} = \mathbf{x}^{(k)}$
Endif

The augmented Lagrangian (cont.)

The implementation of the algorithm is given in the following

- Except for `lambda_0` that contains the initial vector $\boldsymbol{\lambda}^{(0)}$ of Lagrange multipliers, all other inputs and outputs have been already explained for `pFunction`, `dScent` and others

The augmented Lagrangian (cont.)

```
% ALGRNG Constrained optimisation with augmented Lagrangian
% [X,ERR,K]=ALGRNG(F,GRAD_F,H,GRAD_H,X_0,LAMBDA_0,...
%                  TOL,KMAX,KMAXD,TYP)
%  Approximate a minimiser of the cost function F
%  under equality constraints H=0
%
% X_0 is initial point, TOL is tolerance for stop check
% KMAX is the maximum number of iterations
% GRAD_F and GRAD_H are the gradients of F and H
%
% For TYP=0 solution by FMINSEARCH M-function
% FOR TYP>0 solution by a DESCENT METHOD
%  KMAXD is maximum number of iterations
%  TYP is the choice of descent directions
%  TYP=1 and TYP=2 need the Hessian (or an approx. at k=0)
```

The augmented Lagrangian (cont.)

```
function [x,err,k]=aLgrng(f,grad_f,h,grad_h,x_0,lambda_0,...
                          tol,kmax,kmaxd,typ,varargin)

mu_0=1.0;

if typ==1; hess=varargin{1};
 elseif typ==2; hess=varargin{1};
 else; hess=[]; end

err=1+tol+1; k=0; xk=x_0(:); lambdak=lambda_0(:);

if ~isempty(h); [nh,mh]=size(h(xk)); end

muk=mu_0; muk2=muk/2; told=0.1;

while err>tol && k<kmax
 if typ==0
  options=optimset ('TolX',told);
  [x,err,kd]=fminsearch(@L,xk,options); err=norm(x-xk);
 else
  [x,err,kd]=descent(@L,@grad_L,xk,told,kmaxd,typ,hess);
  err=norm(grad_L(x));
 end

 lambdak=lambdak-muk*h(x);
 if kd<kmaxd; muk=10*muk; muk2=0.5*muk;
 else muk=1.5*muk; muk2=0.5*muk; end

 k=1+k; xk=x; told=max([tol,0.10*told]);
```

The augmented Lagrangian (cont.)

```
function y=L(x) % This function is nested inside aLgrng

y=fun(x);
if ~isempty(h)
 y=y-sum(lambdak'*h(x))+muk2*sum((h(x)).^2);
end
```

```
function y=grad_L(x) % This function is nested inside aLgrng

y=grad_fun(x);
if ~isempty(h)
 y=y+grad_h(x)*(muk*h(x)-lambdak);
end
```

The augmented Lagrangian (cont.)

Example

```
fun = @(x) 0.6*x(1).^2 + 0.5*x(2).*x(1) - x(2) + 3*x(1);
grad_fun = @(x) [1.2*x(1) + 0.5*x(2) + 3; 0.5*x(1) - 1];

h = @(x) x(1).^2 + x(2).^2 - 1;
grad_h = @(x) [2*x(1); 2*x(2)];

x_0 = [1.2,0.2]; tol = 1e-5; kmax = 500; kmaxd = 100;
p=1; % The number of equality constraints
lambda_0 = rand(p,1); typ=2; hess=eye(2);

[xmin,err,k] = aLagrange(fun,grad_fun,h,grad_h,x_0,...
                         lambda_0,tol,kmax,kmax,typ,hess)
```

As stopping criterion, we have set the tolerance to be 10^{-5} and we opted for an associated unconstrained minimisation problem by quasi-Newton descent directions (with `typ=2` and `hess=eye(2)`)