Constrained optimisation
(CK0031/CK0248)

Francesco Corona

Department of Computer Science
Federal University of Ceará, Fortaleza

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 relatively simple problems

- 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*

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

Ω is a closed subset determined by equality or inequality constraints

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

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

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

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

p and q are natural numbers

Constrained optimisation (cont.)

More generally,

$$\min_{\mathbf{x} \in \Omega \subset \mathbb{R}^n} f(\mathbf{x}) \tag{4}$$

Ω a closed subset determined both equality and inequality constraints

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

The two sets $\mathcal{I}_h$ and $\mathcal{I}_g$

- ⇝ In Equation (3), $\mathcal{I}_h = \emptyset$
- ⇝ In Equation (2), $\mathcal{I}_g = \emptyset$

Constrained optimisation (cont.)

Definition

The general constrained optimisation problem

$$\begin{gathered} \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \\ \textit{subjected to} \\ h_i(\mathbf{x}) = 0, \quad \textit{for all } i \in \mathcal{I}_h \\ g_j(\mathbf{x}) \geq 0, \quad \textit{for all } j \in \mathcal{I}_g \end{gathered} \tag{5}$$

Constrained optimisation (cont.)

Suppose that $f \in \mathbb{C}^1(\mathbb{R}^n)$ and that h_i and g_j are class $\mathbb{C}^1(\mathbb{R}^n)$, for all i, j

Points $\mathbf{x} \in \Omega$ that satisfy all the constraints are **feasible points**

$\rightsquigarrow$ The closed subset Ω is the set of all feasible points

Constrained optimisation (cont.)

Consider a point $\mathbf{x}^* \in \Omega \subset \mathbb{R}^n$ such that

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

Point $\mathbf{x}$ is said to be a **global minimiser** for the problem

Consider a point $\mathbf{x}^* \in \Omega \subset \mathbb{R}^n$ such that

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in B_r(\mathbf{x}^*) \cap \Omega \tag{7}$$

- $B_r(\mathbf{x}) \in \mathbb{R}^n$ is a ball centred in $\mathbf{x}^*$ and radius $r > 0$

Point $\mathbf{x}$ is said to be a **local minimiser** for the problem

Constrained optimisation (cont.)

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

- Active constraints at $\mathbf{x}$ are all the h_i and the g_j such that $g_j(\mathbf{x}) = 0$

Constrained optimisation (cont.)

Example

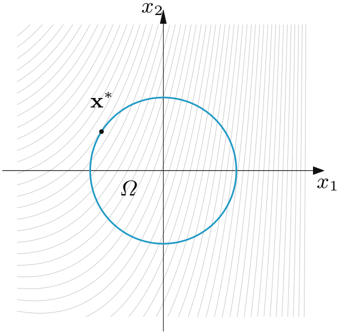
Consider the minimisation of function $f(\mathbf{x})$ under equality constraint $h_1(\mathbf{x})$

Let

$$f(\mathbf{x}) = 3/5x_1^2 + 1/2x_1x_2 - x_2 + 3x_1$$

Let

$$h_1(\mathbf{x}) = x_1^2 + x_2^2 - 1 = 0$$



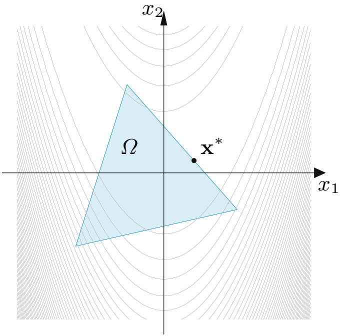
- Contour lines of the cost $f(\mathbf{x})$
- Admissibility set $\Omega \in \mathbb{R}^2$
- The global minimiser $\mathbf{x}^*$ constrained to Ω

Constrained optimisation (cont.)

Example

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

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



- 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.)

Let Ω be a non-empty, bounded and closed set

Weierstrass guarantees existence of a maximum and a minimum for f in Ω

$\rightsquigarrow$ The general constrained optimisation problem admits a solution

Constrained optimisation (cont.)

Definition

*Recall the conditions for $f : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ to be **strongly convex** in Ω*

f is strongly convex if $\exists \rho > 0$ such that $\forall \mathbf{x}, \mathbf{y} \in \Omega$ and $\forall \alpha \in [0, 1]$

$$\underbrace{f\big[\alpha \mathbf{x} + (1-\alpha)\mathbf{y}\big] \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y})}_{Convexity} - \alpha(1-\alpha)\rho||\mathbf{x}-\mathbf{y}||^2 \qquad (8)$$

Strong convexity reduces to the usual convexity when $\rho = 0$

Constrained optimisation (cont.)

Proposition

Optimality conditions

Let $\Omega \subset \mathbb{R}^n$ *be a convex set*

Let $\mathbf{x}^* \in \Omega$ *be such that* $f \in \mathbb{C}^1\left[B_r(\mathbf{x}^*)\right]$

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

$$\nabla f(\mathbf{x}^*)^T(\mathbf{x} - \mathbf{x}^*) \geq 0, \quad \forall \mathbf{x} \in \Omega \tag{9}$$

If f *is convex in* Ω *and (9) is satisfied, then* $\mathbf{x}^*$ *is a global minimiser*

Suppose that we require Ω *to be closed and* f *to be strongly convex*

$\leadsto$ *It can be shown that the minimiser is unique*

Constrained optimisation (cont.)

There are many algorithms for solving constrained minimisation problems

Many search for the stationary points of the **Lagrangian function**

- The **KKT** or **Karush-Kuhn-Tucker points**

Definition

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

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) - \sum_{j \in \mathcal{I}_g} \mu_j g_j(\mathbf{x}) \tag{10}$$

$\boldsymbol{\lambda}$ *and* $\boldsymbol{\mu}$ *are* ***Lagrangian multipliers***

- $\boldsymbol{\lambda} = (\lambda_i)$, *for* $i \in \mathcal{I}_h$
- $\boldsymbol{\mu} = (\mu_i)$, *for* $j \in \mathcal{I}_g$

They are (weights) associated with the equality and inequality constraints

Constrained optimisation (cont.)

Definition

Karush-Kuhn-Tucker conditions

A point $\mathbf{x}^*$ is said to be 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 Karush-Kuhn-Tucker 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.)

Let $\mathbf{x}$ be given point

Suppose that the gradients $\nabla h_i(\mathbf{x})$ and $\nabla g_j(\mathbf{x})$ associated with the active constraints in $\mathbf{x}$ are linearly independent

The constraints satisfy the **linear independence** (**constraint**) **qualification** (**LI**(**C**)**Q**) in $\mathbf{x}$

Constrained optimisation (cont.)

Theorem

First order KKT conditions

Let $\mathbf{x}^*$ be a local minimum for the constrained problem

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

Let f, h_i and g_j be $\mathbb{C}^1(\Omega)$

Let the constraints be LIQ in $\mathbf{x}^*$

Then, there exist $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ is a KKT point

Constrained optimisation (cont.)

In the absence of inequality constraints, the Lagrangian takes the form

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

These KKT conditions are known as Lagrange (necessary) conditions

$$\begin{aligned} \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) &= \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0} \\ h_i(\mathbf{x}^*) &= 0, \forall i \in \mathcal{I}_h \end{aligned} \qquad (11)$$

Constrained optimisation (cont.)

Remark

Sufficient conditions for a KKT point to be a minimiser of f in Ω

- $\rightsquigarrow$ Knowledge about the Hessian of the Lagrangian is required

Alternatively, we need strict convexity hypothesis on f and the constraints

Constrained optimisation (cont.)

In general, it is possible to reformulate a constrained optimisation problem

- As an unconstrained optimisation problem

The idea is to replace the original problem by a sequence of subproblems in which the constraints are represented by terms added to the objective

$\rightsquigarrow$ **(Quadratic) Penalty function**

$\rightsquigarrow$ **Augmented Lagrangian**

The penalty method

Constrained optimisation

The penalty method

Consider solving the general constrained optimisation problem

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

$$\text{subjected to}$$

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

$$g_j(\mathbf{x}) \geq 0, \quad \forall j \in \mathcal{I}_g$$

We reformulate it as an unconstrained optimisation problem

The penalty method

Definition

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

$$\mathcal{P}_{\alpha}(\mathbf{x}) = f(\mathbf{x}) + \frac{\alpha}{2} \sum_{i \in \mathcal{I}_h} h_i^2(\mathbf{x}) + \frac{\alpha}{2} \sum_{j \in \mathcal{I}_g} \Big[\max\{-g_j(\mathbf{x}), 0\} \Big]^2 \tag{12}$$

The method adds a multiple of the square of the violation of each constraint

- Terms are zero when $\mathbf{x}$ does not violate the constrain

By making the coefficients larger, we penalise violations more severely

- This forces the minimiser closer to the feasible reagion

The penalty method (cont.)

Consider the situation in which the constraints are not satisfied at $\mathbf{x}$

- The sums quantify how far point $\mathbf{x}$ is from the feasibility set Ω
- A large α heavily penalises such a violation

The penalty method (cont.)

Example

Consider the minimisation of function $f(\mathbf{x})$ under equality constraint $h_1(\mathbf{x})$

Let

$$f(\mathbf{x}) = x_1 + x_2$$

Let

$$h_1(\mathbf{x}) = x_1^2 + x_2^2 - 2 = 0$$

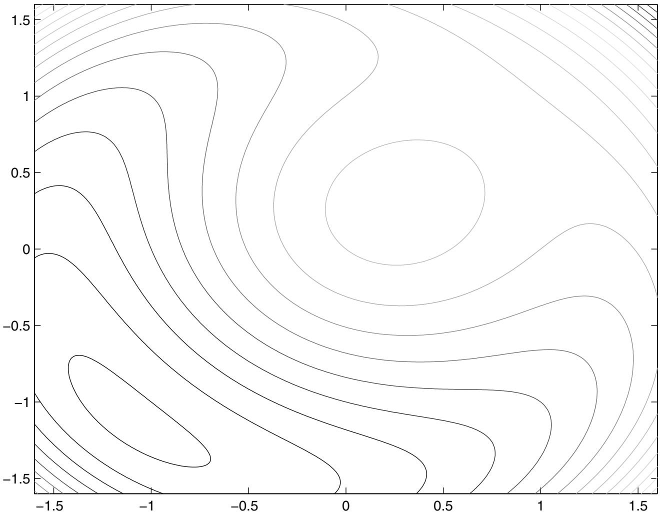
Consider the quadratic penalty function

$$\mathcal{P}_\alpha = (x_1 + x_2) + \frac{\alpha}{2}(x_1^2 + x_2^2 - 2)^2$$

The minimiser is $(-1, -1)'$

The penalty method (cont.)

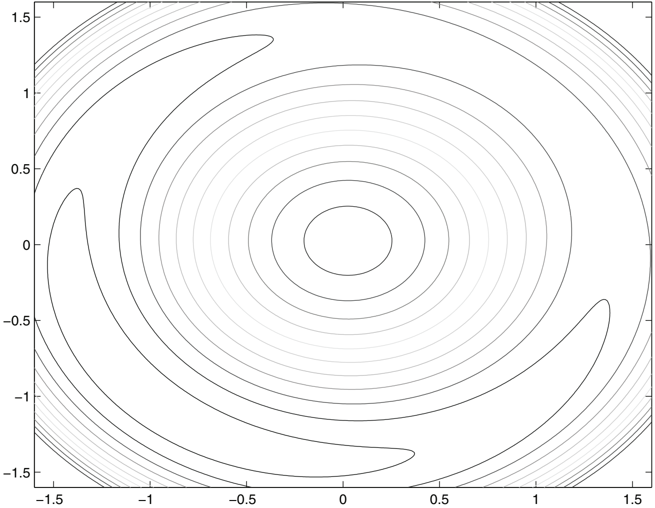
The plot of the contour of the penalty function for $\alpha = 1$



There is a local minimiser near $(0.3, 0.3)'$

The penalty method (cont.)

The plot of the contour of the penalty function for $\alpha = 10$



Points outside the feasible region suffer a much greater penalty

The penalty method (cont.)

If $\mathbf{x}^*$ is a solution to the constrained problem, $\mathbf{x}^*$ is a minimiser of $\mathcal{P}$

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

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

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

As $\alpha >> 1$, $\mathbf{x}^*(\alpha)$ is a good approximation of $\mathbf{x}^*$

The penalty method (cont.)

Not advised (instability) to minimise $\mathcal{P}_\alpha(\mathbf{x})$ directly for large values of α

Rather, consider an increasing and unbounded sequence of parameters $\{\alpha_k\}$

- For each α_k, calculate an approximation $\mathbf{x}^{(k)}$ of the solution $\mathbf{x}^*(\alpha_k)$ to the unconstrained optimisation problem $\min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$

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

- At step k, α_{k+1} is a chosen as a function of α_k (say, $\alpha_{k+1} = \delta \alpha_k$, for $\delta \in [1.5, 2]$) and $\mathbf{x}^{(k)}$ is used to initialise 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 correct 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, $\alpha_0 = 1$), given ε_0 (typically $\varepsilon_0 = 1/10$), given $\overline{\varepsilon} > 0$, given $\mathbf{x}_0^{(0)} \in \mathbb{R}^n$ and given $\boldsymbol{\lambda}_0^{(0)} \in \mathbb{R}^p$, for $k = 0, 1, \dots$ until convergence

Pseudo-code

Compute an approximated solution to $\min\limits_{\mathbf{x} \in \mathbb{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$

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

(Using the initial point $\mathbf{x}_0^{(0)}$ *and tolerance* ε_k*)*

If $\big|\big|\nabla_{\mathbf{x}} \mathcal{L}_A\big[\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \alpha_k\big]\big|\big| \leq \overline{\varepsilon}$
    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

The extra tolerance $\overline{\varepsilon}$ is used 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 ($\mathcal{I}_g = \emptyset$)

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

$$\text{subjected to}$$

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

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

Definition

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

$$\mathcal{L}_A(\mathbf{x}, \boldsymbol{\lambda}, \alpha) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) + \alpha/2 \sum_{i \in \mathcal{I}_h} h_i^2(\mathbf{x}) \tag{13}$$

The augmented Laplacian method is an iterative method

The augmented Lagrangian (cont.)

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

We build a sequence of parameters $\mu_k \rightarrow \infty$

$\alpha_k \rightarrow \infty$ is st $\{(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)})\}$ converges to a KKT point for the Lagrangian

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

At the k-th iteration, for a given α_k and for a given $\boldsymbol{\lambda}^{(k)}$, we compute

$$\mathbf{x}^{(k)} = \underset{\mathbf{x} \in \mathbb{R}^n}{\arg\min}\, \mathcal{L}_A\big[\mathbf{x}, \boldsymbol{\lambda}^{(k)}, \alpha_k\big] \tag{14}$$

The augmented Lagrangian (cont.)

We obtain the multipliers $\boldsymbol{\lambda}^{(k+1)}$ from the gradient of the augmented Lagrangian with respect $\mathbf{x}$ and we set it to be equal to zero

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

By comparison with optimality condition

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

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

we identify $\lambda_i^{(k)}$ as

$$\lambda_i^{(k)} - \alpha_k h_i \big[\mathbf{x}^{(k)}\big] \simeq \lambda_i^*$$

The augmented Lagrangian (cont.)

We thus define,

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} - \alpha_k h_i \big[\mathbf{x}^{(k)}\big] \tag{15}$$

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

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

The augmented Lagrangian (cont.)

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

Pseudo-code

Compute an approximated solution

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

(Using the initial point $\mathbf{x}_0^{(0)}$ *and tolerance* ε_k*)*

If $\big|\big|\nabla_{\mathbf{x}} \mathcal{L}_A\big[\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \alpha_k\big]\big|\big| \leq \overline{\varepsilon}$
    Set $\mathbf{x}^* = \mathbf{x}^{(k)}$ *(convergence)*
else
    Compute $\lambda_i^{(k+1)} = \lambda_i^{(k)} - \mu_k h_i\big[\mathbf{x}^{(k)}\big]$
    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

```
% 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.)

- `lambda_0` contains the initial vector $\boldsymbol{\lambda}^{(0)}$ of Lagrange multipliers

Other inputs/outputs have been explained for `pFunction`, `dScent`, ...

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)
```

Stopping criterion: A tolerance set 10^{-5}

The unconstrained minimisation by quasi-Newton descent directions

- (with `typ=2` and `hess=eye(2)`)