Solving Optimization Problems using the Matlab Optimization Toolbox – a Tutorial

Article

2 authors, including:

Abebe Geletu
Technische Universität Ilmenau
20 PUBLICATIONS **88** CITATIONS

SEE PROFILE

Solving Optimization Problems using the Matlab Optimization Toolbox - a Tutorial

TU-Ilmenau, Fakultät für Mathematik und Naturwissenschaften
Dr. Abebe Geletu

December 13, 2007

Contents

1 Introduction to Mathematical Programming **2**
1.1 A general Mathematical Programming Problem 2
1.1.1 Some Classes of Optimization Problems 2
1.1.2 Functions of the Matlab Optimization Toolbox 5

2 Linear Programming Problems **6**
2.1 Linear programming with MATLAB 6
2.2 The Interior Point Method for LP 8
2.3 Using linprog to solve LP's 11
2.3.1 Formal problems 11
2.3.2 Approximation of discrete Data by a Curve 13

3 Quadratic programming Problems **15**
3.1 Algorithms Implemented under quadprog.m 16
3.1.1 Active Set-Method 17
3.1.2 The Interior Reflective Method 19
3.2 Using quadprog to Solve QP Problems 24
3.2.1 Theoretical Problems 24
3.2.2 Production model - profit maximization 26

4 Unconstrained nonlinear programming **30**
4.1 Theory, optimality conditions 30
4.1.1 Problems, assumptions, definitions 30
4.2 Optimality conditions for smooth unconstrained problems 31
4.3 Matlab Function for Unconstrained Optimization 32
4.4 General descent methods - for differentiable Optimization Problems 32
4.5 The Quasi-Newton Algorithm -idea 33
4.5.1 Determination of Search Directions 33
4.5.2 Line Search Strategies- determination of the step-length α_k 34
4.6 Trust Region Methods - idea 35
4.6.1 Solution of the Trust-Region Sub-problem 36
4.6.2 The Trust Sub-Problem under Considered in the Matlab Optimization toolbox . 38
4.6.3 Calling and Using fminunc.m to Solve Unconstrained Problems 39
4.7 Derivative free Optimization - direct (simplex) search methods 45

Chapter 1

Introduction to Mathematical Programming

1.1 A general Mathematical Programming Problem

$$\boxed{\begin{array}{lcll} & f(x) & \longrightarrow & \min \quad (\max) \\ \textit{subject to} & & & \\ & x & \in & M. \end{array}} \tag{O}$$

The function $f : \mathbb{R}^n \to \mathbb{R}$ is called the *objective function* and the set $M \subset \mathbb{R}^n$ is the f*easible set* of (O). Based on the description of the function f and the feasible set M, the problem (O) can be classified as linear, quadratic, non-linear, semi-infinite, semi-definite, multiple-objective, discrete optimization problem etc[1].

1.1.1 Some Classes of Optimization Problems

Linear Programming

If the objective function f and the defining functions of M are linear, then (O) will be a **linear optimization** problem.
General form of a linear programming problem:

$$\boxed{\begin{array}{lcl} c^\top x & \longrightarrow & min\ (max) \\ s.t. & & \\ Ax & = & a \\ Bx & \leq & b \\ lb \leq & x & \leq ub; \end{array}} \tag{LO}$$

i.e. $f(x) = c^\top x$ and $M = \{x \in \mathbb{R}^n \mid Ax = a, Bx \leq b, lb \leq x \leq ub\}$.

Under linear programming problems are such practical problems like: linear discrete Chebychev approximation problems, transportation problems, network flow problems,etc.

[1]The terminology mathematical **programming** is being currently contested and many demand that problems of the form (O) be always called mathematical **optimization** problems. Here, we use both terms alternatively.

Quadratic Programming

$$\begin{array}{rcl} \frac{1}{2}x^T Qx + q^\top x & \rightarrow & \min \\ \text{s.t.} & & \\ Ax & = & a \\ Bx & \leq & b \\ x & \geq & u \\ x & \leq & v \end{array} \qquad \text{(QP)}$$

Here the objective function $f(x) = 12x^\top Qx + q^\top x$ is a **quadratic function**, while the feasible set $M = \{x \in \mathbb{R}^n \mid Ax = a, Bx \leq b, u \leq x \leq v\}$ is defined using linear functions.

One of the well known practical models of quadratic optimization problems is the least **squares approximation** problem; which has applications in almost all fields of science.

Non-linear Programming Problem

The general form of a non-linear optimization problem is

$$\begin{array}{lrcll} & f(x) & \longrightarrow & \min & (\max) \\ \textit{subject to} & & & & \\ \text{equality constraints:} & g_i(x) & = & 0, & i \in \{1, 2, \ldots, m\} \\ \text{inequality constraints:} & g_j(x) & \leq & 0, & j \in \{m+1, m+2, \ldots, m+p\} \\ \text{box constraints:} & u_k \leq & x_k & \leq v_k, & k = 1, 2, \ldots, n; \end{array} \qquad \text{(NLP)}$$

where, we assume that all the function are **smooth**, i.e. the functions

$$f,\ g_l \ : \ U \longrightarrow \mathbb{R}\ l = 1, 2, \ldots, m+p$$

are sufficiently many times differentiable on the open subset U of $\mathbb{R}^n$. The feasible set of (NLP) is given by

$$M = \{x \in \mathbb{R}^n \mid g_i(x) = 0, i = 1, 2, \ldots, m; g_j(x) \leq 0, j = m+1, m+2, \ldots, m+p\}.$$

We also write the (NLP) in vectorial notation as

$$\begin{array}{rcl} f(x) & \rightarrow & \min\ (\max) \\ h(x) & = & 0 \\ g(x) & \leq & 0 \\ u & \leq & x \leq v. \end{array}$$

Problems of the form (NLP) arise frequently in the numerical solution of control problems, non-linear approximation, engineering design, finance and economics, signal processing, etc.

Semi-infinite Programming

$$\boxed{\begin{array}{rcl} f(x) & \rightarrow & \min \\ \text{s.t.} & & \\ G(x,y) & \leq & 0, \forall y \in Y; \\ h_i(x) & = & 0, i = 1, \ldots, p; \\ g_j(x) & \leq & 0, j = 1, \ldots, q; \\ x & \in & \mathbb{R}^n; \\ Y & \subset & \mathbb{R}^m. \end{array}} \qquad \text{(SIP)}$$

Here, $f, h_i, g_j : \mathbb{R}^n \rightarrow \mathbb{R}, i \in \{1, \ldots, p\}, j \in \{1, \ldots, q\}$ are smooth functions; $G : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ is such that, for each fixed $y \in Y$, $G(\cdot, y) : \mathbb{R}^n \rightarrow \mathbb{R}$ is smooth and, for each fixed $x \in \mathbb{R}^n, G(x, \cdot) : R^m \rightarrow \mathbb{R}$ is smooth; furthermore, Y is a compact subset of $\mathbb{R}^m$. Sometimes, the set Y can also be given as

$$Y = \{y \in \mathbb{R}^m \mid u_k(y) = 0, k = 1, \ldots, s_1; \;\; v_l(y) \leq 0, l = 1, \ldots, s_2\}$$

with smooth functions $u_k, v_l : \mathbb{R}^m \rightarrow \mathbb{R}, k \in \{1, \ldots, s_1\}, l \in \{1, \ldots, s_2\}$.

The problem (SIP) is called semi-infinite, since its an optimization problem with finite number of variables (i.e. $x \in \mathbb{R}^n$) and infinite number of constraints (i.e. $G(x, y) \leq 0, \forall y \in Y$).

One of the well known practical models of (SIP) is the continuous Chebychev approximation problem. This approximation problem can be used for the approximation of functions by polynomials, in filter design for digital signal processing, spline approximation of robot trajectory

Multiple-Objective Optimization

A multiple objective Optimization problem has a general form

$$\boxed{\begin{array}{c} min(f_1(x), f_1(x), \ldots, f_m(x)) \\ \text{s.t.} \\ x \in M; \end{array}} \qquad \text{(MO)}$$

where the functions $f_k : \mathbb{R}^n \rightarrow \mathbb{R}, k = 1, \ldots, m$ are smooth and the feasible set M is defined in terms of linear or non-linear functions. Sometimes, this problem is also alternatively called **multiple-criteria**, **vector optimization**, **goal attainment** or **multi-decision analysis** problem. It is an optimization problem with more than one objective function (each such objective is a criteria). In this sense, (LO),(QP)(NLO) and (SIP) are single objective (criteria) optimization problems. If there are only two objective functions in (MO), then (MO) is commonly called to be a **bi-criteria** optimization problem. Furthermore, if each of the functions $f_1, \ldots, f_m$ are linear and M is defined using linear functions, then (MO) will be a linear multiple-criteria optimization problem; otherwise, it is non-linear.

For instance, in a financial application we may need, to maximize revenue and minimize risk at the same time, constrained upon the amount of our investment. Several engineering design problems can also be modeled into (MO). Practical problems like autonomous vehicle control, optimal truss design, antenna array design, etc are very few examples of (MO).

In real life we may have several objectives to arrive at. But, unfortunately, we cannot satisfy all our objectives optimally at the same time. So we have to find a compromising solution among all our objectives. Such is the nature of multiple objective optimization. Thus, the minimization (or

maximization) of several objective functions can not be done in the usual sense. Hence, one speaks of so-called **efficient points** as solutions of the problem. Using special constructions involving the objectives, the problem (MO) can be reduced to a problem with a single objective function.

1.1.2 Functions of the Matlab Optimization Toolbox

Linear and Quadratic Minimization problems.

linprog - Linear programming.

quadprog - Quadratic programming.

Nonlinear zero finding (equation solving).

fzero - Scalar nonlinear zero finding.

fsolve - Nonlinear system of equations solve (function solve).

Linear least squares (of matrix problems).

lsqlin - Linear least squares with linear constraints.

lsqnonneg - Linear least squares with nonnegativity constraints.

Nonlinear minimization of functions.

fminbnd - Scalar bounded nonlinear function minimization.

fmincon - Multidimensional constrained nonlinear minimization.

fminsearch - Multidimensional unconstrained nonlinear minimization, by Nelder-Mead direct search method.

fminunc - Multidimensional unconstrained nonlinear minimization.

fseminf - Multidimensional constrained minimization, semi-infinite constraints.

Nonlinear least squares (of functions).

lsqcurvefit - Nonlinear curvefitting via least squares (with bounds).

lsqnonlin - Nonlinear least squares with upper and lower bounds.

Nonlinear minimization of multi-objective functions.

fgoalattain - Multidimensional goal attainment optimization

fminimax - Multidimensional minimax optimization.

Chapter 2

Linear Programming Problems

2.1 Linear programming with MATLAB

For the linear programming problem

$$\begin{array}{lcl} c^\top x & \longrightarrow & min \\ s.t. & & \\ Ax & \leq & a \\ Bx & = & b \\ lb \leq & x & \leq ub; \end{array} \qquad \text{(LP)}$$

MATLAB: The program `linprog.m` is used for the minimization of problems of the form (LP).

Once you have defined the matrices A, B, and the vectors c,a,b,lb and ub, then you can call `linprog.m` to solve the problem. The general form of calling `linprog.m` is:

```
[x,fval,exitflag,output,lambda]=linprog(f,A,a,B,b,lb,ub,x0,options)
```

Input arguments:

c	coefficient vector of the objective
A	Matrix of inequality constraints
a	right hand side of the inequality constraints
B	Matrix of equality constraints
b	right hand side of the equality constraints
lb,[ ]	$lb \leq x$: lower bounds for x, no lower bounds
ub,[ ]	$x \leq ub$: upper bounds for x, no upper bounds
$x0$	Startvector for the algorithm, if known, else [ ]
options	options are set using the optimset funciton, they determine what algorism to use,etc.

Output arguments:

x	optimal solution
fval	optimal value of the objective function
exitflag	tells whether the algorithm converged or not, exitflag > 0 means convergence
output	a struct for number of iterations, algorithm used and PCG iterations(when LargeScale=on)
lambda	a struct containing lagrange multipliers corresponding to the constraints.

Setting Options

The input argument `options` is a structure, which contains several parameters that you can use with a given Matlab optimization routine.

For instance, to see the type of parameters you can use with the `linprog.m` routine use

```
>>optimset('linprog')
```

Then Matlab displays the fileds of the structure `options`. Accordingly, before calling `linprog.m` you can set your preferred parameters in the `options` for `linprog.m` using the `optimset` command as:

```
>>options=optimset('ParameterName1',value1,'ParameterName2',value2,...)
```

where `'ParameterName1','ParameterName2',...` are those you get displayed when you use `optimset('linprog')`. And `value1, value2,...` are their corresponding values.

The following are parameters and their corresponding values which are frequently used with `linprog.m`:

Parameter	Possible Values
'LargeScale'	'on','off'
'Simplex'	'on','off'
'Display'	'iter','final','off'
'Maxiter'	Maximum number of iteration
'TolFun'	Termination tolerance for the objective function
'TolX'	Termination tolerance for the iterates
'Diagnostics'	'on' or 'off' (when 'on' prints diagnostic information about the objective function)

Algorithms under linprog

There are three type of algorithms that are being implemented in the `linprog.m`:

- a simplex algorithm;
- an active-set algorithm;
- a primal-dual interior point method.

The simplex and active-set algorithms are usually used to solve **medium-scale** linear programming problems. If any one of these algorithms fail to solve a linear programming problem, then the problem at hand is a **large scale** problem. Moreover, a linear programming problem with several thousands of variables along with sparse matrices is considered to be a large-scale problem. However, if coefficient matrices of your problem have a dense matrix structure, then `linprog.m` assumes that your problem is of medium-scale.

By default, the parameter `'LargeScale'` is always `'on'`. When `'LargeScale'` is 'on', then `linprog.m` uses the primal-dual interior point algorithm. However, if you want to set if off so that you can solve a medium scale problem , then use

```
>>options=optimset('LargeScale','off')
```

In this case `linprog.m` uses either the simplex algorithm or the active-set algorithm. (Nevertheless, recall that the simplex algorithm is itself an active-set strategy).

If you are specifically interested to use the active set algorithm, then you need to set both the parameters `'LargeScale'` and `'Simplex'`, respectively, to 'off':

```
>>options=optimset('LargeScale','off','Simplex','off')
```

Note: Sometimes, even if we specified 'LargeScale' to be 'off', when a linear programming problem cannot be solved with a medium scale algorithm, then linprog.m automatically switches to the large scale algorithm (interior point method).

2.2 The Interior Point Method for LP

Assuming that the simplex method already known, we find this section a brief discussion on the primal-dual interior point method for (LP).

Let $A \in \mathbb{R}^{m\times n}, a \in \mathbb{R}^m, B \in \mathbb{R}^{p\times n}, b \in \mathbb{R}^p$. Then, for the linear programming problem

$$\boxed{\begin{array}{lcl} c^\top x & \longrightarrow & min \\ s.t. & & \\ Ax & \leq & a \\ Bx & = & b \\ lb \leq & x & \leq ub; \end{array}} \qquad \text{(LP)}$$

if we set $\widetilde{x} = x - lb$ we get

$$\boxed{\begin{array}{lcl} c^\top \widetilde{x} - c^\top lb & \longrightarrow & min \\ s.t. & & \\ A\widetilde{x} & \leq & a - A(lb) \\ B\widetilde{x} & = & b - B(lb) \\ 0 \leq & \widetilde{x} & \leq ub - lb; \end{array}} \qquad \text{(LP)}$$

Now, by adding slack variables $y \in \mathbb{R}^m$ and $s \in \mathbb{R}^n$ (see below), we can write (LP) as

$$\boxed{\begin{array}{lcccl} c^\top \widetilde{x} - c^\top lb & \longrightarrow & min & & \\ s.t. & & & & \\ A\widetilde{x} & +y & & = & a - A(lb) \\ B\widetilde{x} & & & = & b - B(lb) \\ \widetilde{x} & & +s & = & ub - lb \\ & \widetilde{x} \geq 0, & y \geq 0, & s \geq 0. & \end{array}} \qquad \text{(LP)}$$

Thus, using a single matrix for the constraints, we have

$$\begin{array}{l} c^\top \widetilde{x} - c^\top lb \longrightarrow min \\ s.t. \\ \quad \begin{pmatrix} A & I_m & O_{m\times n} \\ B & O_{p\times m} & O_{p\times n} \\ I_n & O_{n\times n} & I_n \end{pmatrix} \begin{pmatrix} \widetilde{x} \\ y \\ s \end{pmatrix} = \begin{pmatrix} a - A(lb) \\ b - B(lb) \\ ub - lb \end{pmatrix} \\ \qquad \widetilde{x} \geq 0, y \geq 0, s \geq 0. \end{array}$$

Since, a constant in the objective does not create a difficulty, we assume w.l.o.g that we have a problem of the form

$$\begin{array}{lcl} c^\top x & \longrightarrow & min \\ s.t. & & \\ Ax & = & a \\ x & \geq & 0. \end{array} \tag{LP'}$$

In fact, when you call `linprog.m` with the original problem (LP), this transformation will be done by Matlab internally. The aim here is to briefly explain the algorithm used, when you set the `LargeScale` parameter to `'on'` in the `options` of `linprog`.
Now the dual of (LP') is the problem

$$\begin{array}{lcl} b^\top w & \longrightarrow & max \\ s.t. & & \\ A^\top w & \leq & c \\ w & \in & \mathbb{R}^m. \end{array} \tag{LP$_D$}$$

Using a slack variable $s \in \mathbb{R}^n$ we have

$$\begin{array}{lcll} b^\top w & & \longrightarrow & max \\ s.t. & & & \\ \quad A^\top w + s & = & c & \\ \quad w \in \mathbb{R}^m, & s \geq 0 & . & \end{array} \tag{LP$_D$}$$

The problem (LP') and (LP$_D$) are called primal-dual pairs.

Optimality Condition

It is well known that a vector (x^*, w^*, s^*) is a solution of the primal-dual if and only if it satisfies the Karush-Kuhn-Tucker (KKT) optimlaity condition. The KKT conditions here can be written as

$$\begin{aligned} &A^\top w + s = c \\ &Ax = a \\ &x_i s_i = 0, i = 1, \ldots, n \text{(Complemetarity conditions)} \\ &(x, y) \geq 0. \end{aligned}$$

This system can be written as

$$F(x, w, s) = \begin{bmatrix} A^\top w + s - c \\ Ax - a \\ XSe \end{bmatrix} = 0 \tag{2.1}$$

$$(x, s) \geq 0, \tag{2.2}$$

where $X = diag(x_1, x_2, \ldots, x_n), S = diag(s_1, s_2, \ldots, s_n) \in \mathbb{R}^{n \times n}, e = (1, 1, \ldots, 1)^\top \in \mathbb{R}^n$.

Primal-dual interior point methods generate iterates (x^k, w^k, s^k) that satisfy the system (2.1) & (2.2) so that (2.2) is satisfied strictly; i.e. $x^k > 0, s^k > 0$. That is, for each k, (x^k, s^k) lies in the interior of the nonnegative-orthant. Thus the naming of the method as **interior point method**. Interior point methods use a variant of the Newton method for the system (2.1) & (2.2).

Central Path

Let $\tau > 0$ be a parameter. The **central path** is a curve $\mathcal{C}$ which is the set of all points $(x(\tau), w(\tau), s(\tau)) \in \mathcal{C}$ that satisfy the parametric system :

$$\begin{aligned} &A^\top w + s = c, \\ &Ax = b, \\ &xs_i = \tau, i = 1, \ldots, n \\ &(x, s) > 0. \end{aligned}$$

This implies $\mathcal{C}$ is the set of all points $(x(\tau), w(\tau), s(\tau))$ that satisfy

$$F(x(\tau), w(\tau), s(\tau)) = \begin{bmatrix} 0 \\ 0 \\ \tau e \end{bmatrix}, (x(\tau), s(\tau)) > 0. \tag{2.3}$$

Obviously, if we let $\tau \downarrow 0$, the the system (2.3) goes close to the system (2.1) & (2.2).

Hence, theoretically, primal-dual algorithms solve the system

$$J(x(\tau), w(\tau), s(\tau)) \begin{bmatrix} \triangle x(\tau) \\ \triangle w(\tau) \\ \triangle s(\tau) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -XSe + \tau e \end{bmatrix}$$

to determine a search direction $(\triangle x(\tau), \triangle w(\tau), \triangle s(\tau))$, where $J(x(\tau), w(\tau), s(\tau))$ is the Jacobian of $F(x(\tau), w(\tau), s(\tau))$. And the new iterate will be

$$(x^+(\tau), w^+(\tau), s^+(\tau)) = (x(\tau), w(\tau), s(\tau)) + \alpha(\triangle x(\tau), \triangle w(\tau), \triangle s(\tau)),$$

where α is a step length, usually $\alpha \in (0, 1]$, chosen in such a way that $(x^+(\tau), w^+(\tau), s^+(\tau) \in \mathcal{C}$.

However, practical primal-dual interior point methods use $\tau = \sigma\mu$, where $\sigma \in [0, 1]$ is a constant and

$$\mu = \frac{x^\top s}{n}$$

The term $x^\top s$ is the **duality gap** between the primal and dual problems. Thus, μ is the measure of the (average) duality gap. Note that, in general, $\mu \geq 0$ and $\mu = 0$ when x and s are primal and dual optimal, respectively.

Thus the Newton step $(\triangle x(\mu), \triangle w(\mu), \triangle s(\mu))$ is determined by solving:

$$\begin{bmatrix} O_n & A^\top & I_n \\ A & O_{n\times m} & O_{m\times n} \\ S & O_{n\times m} & X \end{bmatrix} \begin{bmatrix} \triangle x(\tau) \\ \triangle w(\tau) \\ \triangle s(\tau) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -XSe + \sigma\mu e \end{bmatrix}. \tag{2.4}$$

The Newton step $(\triangle x(\mu), \triangle w(\mu), \triangle s(\mu))$ is also called **centering direction** that pushes the iterates $(x^+(\mu), w^+(\mu), s^+(\mu)$ towards the central path $\mathcal{C}$ along which the algorithm converges more rapidly. The parameter σ is called **the centering parameter**. If $\sigma = 0$, then the search direction is known to be an **affine scaling** direction.

Primal-Dual Interior Point Algorithm

Step 0: Start with (x^0, w^0, s^0) with $(x^0, s^0) > 0$, $k = 0$

Step k: choose $\sigma_k \in [0, 1]$, set $\mu_k = (x^k)^\top s^k/n$ and solve

$$\begin{bmatrix} O_n & A^\top & I_n \\ A & O_{n\times m} & O_{m\times n} \\ S & O_{n\times m} & X \end{bmatrix} \begin{bmatrix} \triangle x^k \\ \triangle w^k \\ \triangle s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma_k \mu_k e \end{bmatrix}.$$

set

$$(x^{k+1}, w^{k+1}, s^{k+1}) \leftarrow (x^k, w^k, s^k) + \alpha_k(\triangle x^k, \triangle w^k, \triangle s^k)$$

choosing $\alpha_k \in [0, 1]$ so that $(x^{k+1}, s^{k+1}) > 0$.

If (convergence) STOP else set $k \leftarrow k + 1$ GO To Step k.

The Matlab `LargeSclae` option of `linprog.m` uses a predicator-corrector like method of Mehrotra to guarantee that $(x^k, s^k) > 0$ for each $k, k = 1, 2, \ldots$

Predicator Step: A search direction (a predicator step) $d_p^k = (\triangle x^k, \triangle w^k, \triangle s^k)$ is obtained by solving the non-parameterized system (2.1) & (2.2).

Corrector Step: For a centering parameter σ is obtained from

$$d_c^k = [F^\top(x^k, w^k, s^k)]^{-1} F(x^k + \triangle x^k, w^k \triangle w^k, s^k + \triangle s^k) - \sigma \widehat{e}$$

where $\widehat{e} \in \mathbb{R}^{n+m+n}$, whose last n components are equal to 1.

Iteration: for a step length $\alpha \in (0, 1]$

$$(x^{k+1}, w^{k+1}, s^{k+1}) = (x^k, w^k, s^k) + \alpha(d_p^k + d_c^k).$$

2.3 Using linprog to solve LP's

2.3.1 Formal problems

1. Solve the following linear optimization problem using `linprog.m`.

$2x_1+$	$3x_2$	$\rightarrow$	max
s.t.			
x_1+	$2x_2$	$\leq$	8
$2x_1+$	x_2	$\leq$	10
	x_2	$\leq$	3
$x_1,$	x_2	$\geq$	0

For this problem there are no equality constraints and box constraints, i.e. `B=[],b=[],lb=[]` and `ub=[]`. Moreover,

```
>>c=[-2,-3]'; % linprog solves minimization problems
>>A=[1,2;2,1;0,1];
>>a=[8,10,3]';
>>options=optimset('LargeScale','off');
```

i) If you are interested only on the solution, then use
```
>>xsol=linprog(c,A,b,[],[],[],[],[],options)
```
ii) To see if the algorithm really converged or not you need to access the exit flag through:
```
>>[xsol,fval,exitflag]=linprog(c,A,a,[],[],[],[],[],options)
```
iii) If you need the Lagrange multipliers you can access them using:
```
>>[xsol,fval,flag,output,LagMult]=linprog(c,A,a,[],[],[],[],[],options)
```
iv) To display the iterates you can use:
```
>>xsol=linprog(c,A,a,[],[],[],[],[],optimset('Display','iter'))
```

2. Solve the following LP using `linprog.m`

$$\boxed{\begin{array}{c} c^\top x \longrightarrow \max \\ Ax = a \\ Bx \geq b \\ Dx \leq d \\ lb \leq x \leq lu \end{array}}$$

where

$$(A|\,a) = \left(\begin{array}{cccccc|c} 1 & 1 & 1 & 1 & 1 & 1 & 10 \\ 5 & 0 & -3 & 0 & 1 & 0 & 15 \end{array}\right), \qquad (B|\,b) = \left(\begin{array}{cccccc|c} 1 & 2 & 3 & 0 & 0 & 0 & 5 \\ 0 & 1 & 2 & 3 & 0 & 0 & 7 \\ 0 & 0 & 1 & 2 & 3 & 0 & 8 \\ 0 & 0 & 0 & 1 & 2 & 3 & 8 \end{array}\right),$$

$$(D|\,d) = \left(\begin{array}{cccccc|c} 3 & 0 & 0 & 0 & -2 & 1 & 5 \\ 0 & 4 & 0 & -2 & 0 & 3 & 7 \end{array}\right), \quad lb = \begin{pmatrix} -2 \\ 0 \\ -1 \\ -1 \\ -5 \\ 1 \end{pmatrix}, lu = \begin{pmatrix} 7 \\ 2 \\ 2 \\ 3 \\ 4 \\ 10 \end{pmatrix}, c = \begin{pmatrix} 1 \\ -2 \\ 3 \\ -4 \\ 5 \\ -6 \end{pmatrix}.$$

When there are large matrices, it is convenient to write m files. Thus one possible solution will be the following:

```
function LpExa2
A=[1,1,1,1,1,1;5,0,-3,0,1,0]; a=[10,15]';
B1=[1,2,3,0,0,0; 0,1,2,3,0,0;... 0,0,1,2,3,0;0,0,0,1,2,3]; b1=[5,7,8,8];b1=b1(:);
D=[3,0,0,0,-2,1;0,4,0,-2,0,3]; d=[5,7]; d=d(:);
lb=[-2,0,-1,-1,-5,1]'; ub=[7,2,2,3,4,10]';
c=[1,-2,3,-4,5,-6];c=c(:);
B=[-B1;D]; b=[-b1;d];
[xsol,fval,exitflag,output]=linprog(c,A,a,B,b,lb,ub)
fprintf('%s %s \n', 'Algorithm Used: ',output.algorithm);
disp('================================');
disp('Press Enter to continue'); pause
options=optimset('linprog');
```

```
options = optimset(options,'LargeScale','off','Simplex','on','Display','iter');
[xsol,fval,exitflag]=linprog(c,A,a,B,b,lb,ub,[],options)
fprintf('%s %s \n', 'Algorithm Used: ',output.algorithm);
fprintf('%s','Reason for termination:')
if (exitflag)
        fprintf('%s \n',' Convergence.');
else
        fprintf('%s \n',' No convergence.');
end
```

Observe that for the problem above the simplex algorithm does not work properly. Hence, `linprog.m` uses automatically the interior point method.

2.3.2 Approximation of discrete Data by a Curve

Solve the following discrete approximation problem and plot the approximating curve.

Suppose the measurement of a real process over a 24 hours period be given by the following table with 14 data values:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
t_i	0	3	7	8	9	10	12	14	16	18	19	20	21	23
u_i	3	5	5	4	3	6	7	6	6	11	11	10	8	6

The values t_i represent time and u_i's are measurements. Assuming there is a mathematical connection between the variables t and u, we would like to determine the coefficients $a, b, c, d, e \in \mathbb{R}$ of the function

$$u(t) = a\,t^4 + b\,t^3 + c\,t^2 + d\,t + e$$

so that the value of the function $u(t_i)$ could best approximate the discrete value u_i at $t_i, i = 1, \ldots, 14$ in the Chebychev sense. Hence, we need to solve the **Chebyshev approximation** problem

$$(CA) \qquad \boxed{\begin{array}{c} \max_{i=1,\ldots,14} \left| u_i - \left(a\,t_i^4 + b\,t_i^3 + c\,t_i^2 + d\,t_i + e\right)\right| \to \min \\ \text{s.t. } a, b, c, d, e \in \mathbb{R} \end{array}}$$

Solution: Define the additional variable:

$$f := \max_{i=1,\ldots,14} \left| u_i - \left(a\,t_i^4 + b\,t_i^3 + c\,t_i^2 + d\,t_i + e\right)\right|.$$

Then it follows that the problem (CA) can be equivalently written as:

$$\begin{aligned} (LP) \qquad & f \to min \\ & s.t. \\ & -f \le u_i - \left(a\,t_i^4 + b\,t_i^3 + c\,t_i^2 + d\,t_i + e\right) \le f, \forall i \in \{1, \ldots, 14\}. \end{aligned}$$

Consequently, we have

$$
\begin{aligned}
(LP) \qquad & f \to min \\
& s.t. \\
& -\left(a\,t_i^4 + b\,t_i^3 + c\,t_i^2 + d\,t_i + e\right) - f \leq -u_i, \forall i \in \{1,\ldots,14\} \\
& \left(a\,t_i^4 + b\,t_i^3 + c\,t_i^2 + d\,t_i + e\right) - f \leq u_i, \forall i \in \{1,\ldots,14\}.
\end{aligned}
$$

The solution is provide in the following m-file:

```
function ChebApprox1
%ChebApprox1:
%   Solves a discrete Chebychev polynomial
%   approximation Problem

t=[0,3,7,8,9,10,12,14,16,18,19,20,21,23]';
u=[3,5,5,4,3,6,7,6,6,11,11,10,8,6]';

A1=[-t.^4,-t.^3,-t.^2,-t,-ones(14,1),-ones(14,1)];
A2=[t.^4,t.^3,t.^2,t,ones(14,1),-ones(14,1)];

c=zeros(6,1); c(6)=1; %objective function coefficient
A=[A1;A2];%inequality constraint matrix
a=[-u;u];%right hand side vectro of ineq constraints

[xsol,fval,exitflag]=linprog(c,A,a);

%%%%next plot the Data points and the function %%%%%%%%%

plot(t,u,'r*'); hold on tt=[0:0.5:25];

ut=xsol(1)*(tt.^4)+xsol(2)*(tt.^3)+xsol(3)*(tt.^2)+xsol(4)*tt+...
 xsol(5);
plot(tt,ut,'-k','LineWidth',2)
```

Chapter 3

Quadratic programming Problems

Problem

$$\begin{array}{rcl} \frac{1}{2}x^TQx + q^Tx & \rightarrow & \min \\ \text{s.t.} & & \\ Ax & \leq & a \\ Bx & = & b \\ lb \leq & x & \leq lu \\ x & \in & \mathbb{R}^n. \end{array} \qquad \text{(QP)}$$

where $Q \in \mathbb{R}^{n\times n}, A \in \mathbb{R}^{m\times n}, B \in l \times n, a \in \mathbb{R}^m$ and $b \in \mathbb{R}^l$.

Matlab: To solve quadratic optimization problem with Matlab you use the `quadprog.m` function.

The general form for calling quadprog.m of the problem (QP) is

```
[xsol,fval,exitflag,output,lambda] = quadprog(Q,q,A,a,B,b,lb,ub,x0,options)
```

Input arguments:

Q	Hessian of the objective function
q	Coefficient vector of the linear part of the objective function
A,[ ]	Matrix of inequality constraints, no inequality constraints
a,[ ]	right hand side of the inequality constraints, no inequality constraints
B,[ ]	Matrix of equality constraints, no equality constraints
b,[ ]	right hand side of the equality constraints,no equality constraints
lb,[ ]	$lb \leq x$: lower bounds for x, no lower bounds
ub,[ ]	$x \leq ub$: upper bounds for x, no upper bounds
$x0$	Startvector for the algorithm, if known, else [ ]
options	options are set using the optimset funciton, they determine what algorism to use,etc.

Output arguments:

x	optimal solution
fval	optimal value of the objective function
exitflag	tells whether the algorithm converged or not, exitflag > 0 means convergence
output	a struct for number of iterations, algorithm used and PCG iterations(when LargeScale=on)
lambda	a struct containing lagrange multipliers corresponding to the constraints.

There are specific parameter settings that that you can use with the `quadprog.m` funciton. To see the options parameter for `quadprog.m` along with their default values you can use

```
>>optimset('quadprog')
```

Then Matlab displays a structure containing the options related with `quadprog.m` function. Obsever that, in contrast to `linprog.m`, the fields

```
options.MaxIter, options.TolFun options.TolX, options.TolPCG
```

posses default values in the `quadprog.m`.

With `quadrprog.m` you can solve large-scale and a medium-scale quadratic optimization problems. For instance, to specify that you want to use the medium-scale algorithm:

```
>>oldOptions=optimset('quadprog');
>>options=optimset(oldOptions,'LargeScale','off');
```

The problem (QP) is assumed to be **large-scale**:

- either if there are no equality and inequality constraints; i.e. if there are only lower and upper bound constraints;

$$\begin{array}{rcl} \frac{1}{2}x^TQx + q^Tx & \rightarrow & \min \\ \text{s.t.} & & \\ lb \leq & x & \leq lu \end{array} \qquad \text{(QP)}$$

- or if there are only linear equality constraints.

$$\begin{array}{rcl} \frac{1}{2}x^TQx + q^Tx & \rightarrow & \min \\ \text{s.t.} & & \\ Ax & \leq & a \\ x & \in & \mathbb{R}^n. \end{array} \qquad \text{(QP)}$$

In all other cases (QP) is assumed to be **medium-scale**.

For a detailed description of the quadprog.m use

```
>>doc quadprog
```

3.1 Algorithms Implemented under quadprog.m

(a) Medium-Scale algorithm: Active-set strategy.

(b) Large-Scale algorithm: an interior reflective Newton method coupled with a trust region method.

3.1.1 Active Set-Method

Given the problem

$$\begin{array}{rcl} \frac{1}{2}x^TQx + q^Tx & \rightarrow & \min \\ \text{s.t.} & & \\ Ax & \leq & a \\ Bx & = & b \\ lb \leq \quad x & & \leq lu \end{array} \tag{QP}$$

Matlab **internally** writes bound constraints as linear constraints and extends the matrix A and the vector a to $\tilde{A}$ and $\tilde{a}$, so that the inequality constraint becomes:

$$\tilde{A}x := \begin{pmatrix} A \\ -I_n \\ I_n \end{pmatrix} x \leq \begin{pmatrix} a \\ -u \\ v \end{pmatrix} =: \tilde{a}$$

Consequently, the problem becomes

$$\begin{array}{rcl} \frac{1}{2}x^TQx + q^Tx & \rightarrow & \min \\ \text{s.t.} & & \\ \widetilde{A}x & \leq & \widetilde{a} \\ Bx & = & b \\ x & \in & \mathbb{R}^n. \end{array} \tag{QP}$$

Without loss of generality, we assume now that we have a problem of the form

$$\begin{array}{rcl} f(x) = \frac{1}{2}x^\top Qx + q^\top x & \rightarrow & \min \\ \text{s.t.} & & \\ Ax & \leq & a \\ Bx & = & b \\ x & \in & \mathbb{R}^n. \end{array} \tag{QP}$$

with $Q \in \mathbb{R}^{n\times n}, A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{l\times n}, a \in \mathbb{R}^m$ and $b \in \mathbb{R}^l$.

Optimality conditions:
Necessary condition: $\bar{x}$ is a local (global) solution of (QP) $\Longrightarrow$ There are multipliers $\mu \in \mathbb{R}^m_+, \lambda \in \mathbb{R}^l$ such that

$$\begin{array}{rcl} x^\top Q + q^\top + \mu^\top A + \lambda^\top B & = & 0^\top \\ \mu^\top (a - A\overline{x}) & = & 0 \\ \mu & \geq & 0. \end{array} \tag{KKT2}$$

(Karush-Kuhn-Tucker Conditions for QP).

Sufficient condition: If the matrix Q is **positively semi definite** then (KKT2) is sufficient for global optimality.

Let $I := \{1, \ldots, m\}$ be the index set corresponding to the inequality constraints and for a given feasible point x^k of (QP), define the active index set of inequality constraints by

$$I(k) = \{i \in \{1, \ldots, m\} \mid A(i,:)x^k = a\},$$

where the Matlab notation $A(i,:)$ represents the i-th row of the matrix A and

$$A(k) := \text{a matrix whose rows are the rows of matrix A corresponding to } I(k).$$

Hence, $A(k)x^k = a$.

The Basic Active-set Algorithm

Phase -I: Determine an initial feasible point x^1 for (QP). (Solve an associated LP).

If such x^1 does not exist, then the feasible set of (QP) is empty (i.e. (QP) is infeasible). STOP.
Else Set $k = 1$.

Phase -II:

While 1

Step 1: Determine the active index set $I\left(x^k\right)$ and the matrix $A(k)$.

Step 2: Solve the system of equations

$$\begin{bmatrix} A(k) \\ B \end{bmatrix} d^k = 0 \tag{3.1}$$

$$\left(x^k\right)^\top Q + q^\top + \mu_{active}^\top A(k) + \lambda^\top B = 0. \tag{3.2}$$

to determine μ_{active}^k, λ^k and (the projected gradient) d^k.

Step 3:

If $d^k = 0$

If $\mu_{active}^k \geq 0$, **then**

x^k is a (local) optimal solution , **RETURN**

Else

(a) Determine the most negative Lagrange multiplier and remove the corresponding active index (constraint) from $I(k)$; i.e.

$$\begin{aligned} &\mu_{active}^{i_r} := \min\{\mu_{active}^{k_i} \mid \mu_{active}^{k_i} < 0, i \in I(k)\} \\ &I(k) = I(k) \setminus \{i_r\}. \end{aligned}$$

(b) Determine $\overline{\alpha}$ such that

$$\begin{aligned} \overline{\alpha}_k &:= \max\{\alpha \mid x^k + \alpha\, d^k \text{ is feasible for (QP)}\} \\ &= \min\left\{ \frac{(A(k,:)x^k - a_0)}{(-A(k,:)d^k)} \mid A(k,:)d^k < 0, k \in I \setminus I(k) \right\} \end{aligned}$$

(c) Compute a step length α with $\alpha \leq \overline{\alpha}$ satisfying

$$f(x^k + \alpha d^k) < f(x^k).$$

(d) Set $x^{k+1} = x^k + \alpha\, d^k$

(e) Set $k = k + 1$.

end

Finding Initial Solution for the Active Set method

The active-set algorithm requires an initial feasible point x^1. Hence, in **Phase-I** the following linear programming problem with artificial variables $x_{n+1}, \ldots, x_{n+m+l}$ attached to the constraints of (QP) is solved:

$$(LP_{(QP)}) \qquad \phi(x) := \sum_{i=1}^{m+l} x_{n+i} \to \min \tag{3.3}$$

$$s.t. \tag{3.4}$$

$$Ax + \begin{pmatrix} x_{n+1} \\ \vdots \\ x_{n+m} \end{pmatrix} \leq a; \tag{3.5}$$

$$Bx + \begin{pmatrix} x_{n+m+1} \\ \vdots \\ x_{n+m+l} \end{pmatrix} = b. \tag{3.6}$$

If (LP$_{(QP)}$) has no solution, then (QP) is infeasible. If (LP$_{(QP)}$) has a solution x^1, then we have $\phi(x^1) = 0$. This problem can be solved using `linprog.m`.

3.1.2 The Interior Reflective Method

The 'LargeScale', 'on' option of quadparog.m can be used only when the problem (QP) has simple bound constraints. When this is the case, quadparog.m uses the *interior reflective method* of Coleman and Li to solve (QP). In fact, the interior reflective method also works for non-linear optimization problems of the form

$$(NLP) \qquad \begin{aligned} & f(x) \to \min \\ & s.t. \\ & \quad lb \leq x \leq ub, \end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function, $lu \in \{\mathbb{R} \cup \{-\infty\}\}^n$ and $lb \in \{\mathbb{R} \cup \{\infty\}\}^n$. Thus, for $f(x) = \frac{1}{2}x^\top Qx + q^\top x$, we have a simple bound (QP).

General assumption:

GA1: There is a feasible point that satisfies the bound constraints strictly, i.e.

$$lb < ub.$$

Hence, if

$$\mathcal{F} := \{x \in \mathbb{R}^n \mid lb \leq x \leq ub\}$$

is the feasible set of (NLP), then $int(\mathcal{F}) = \{x \in \mathbb{R}^n \mid lb < x < ub\}$ is non-empty;

GA2: f is at lest twice continuously differentiable;

GA3: for $x^1 \in \mathcal{F}$, the level set

$$\mathcal{L} := \{x \in \mathcal{F} \mid f(x) \leq f(x^1)\}$$

is compact.

The Interior Reflective Method (idea):

- generates iterates x^k such that
$$x^k \in int(\mathcal{F})$$
using a certain affine (scaling) transformation;
- uses a reflective line search method;
- guarantees global super-linear and local quadratic convergence of the iterates.

The Affine Transformation

We write the problem (NLP) as

$$\begin{aligned} (NLP) \qquad & f(x) \to \min \\ & s.t. \\ & \quad x - ub \le 0, \\ & \quad -x + lb \le 0. \end{aligned}$$

Thus, the first order optimality (KKT) conditions at a point $x^* \in \mathcal{F}$ will be

$$\nabla f(x) + \mu_1 - \mu_2 = 0 \tag{3.7}$$
$$x - ub \le 0 \tag{3.8}$$
$$-x + lb \le 0 \tag{3.9}$$
$$\mu_1(x - u) = 0 \tag{3.10}$$
$$\mu_2(-x + l) = 0 \tag{3.11}$$
$$\mu_1 \ge 0, \mu_2 \ge 0. \tag{3.12}$$

That is $\mu_1, \mu_2 \in \mathbb{R}^n_+$. For the sake of convenience we also use l and u instead of lb and ub, respectively. Consequently, the KKT condition can be restated according to the position of the point x in $\mathcal{F}$ as follows:

- if, for some i, $l_i < x_i < u_i$, then the complementarity conditions imply that $(\nabla f(x))_i = 0$;
- if, for some i, $x_i = u_i$, then using (GA1) we have $l_i < x_i$, thus $(\mu_2)_i = 0$. Consequently, $(\nabla f(x))_i = (\mu_1)_i \Rightarrow (\nabla f(x))_i \le 0$.
- if, for some i, $x_i = l_i$, then using (GA1) we have $x_i < u_i$, thus $(\mu_1)_i = 0$. Consequently, $(\nabla f(x))_i = (\mu_2)_i \Rightarrow (\nabla f(x))_i \ge 0$.

Putting all into one equation we find the equivalent (KKT) condition

$$(\nabla f(x))_i = 0, \text{ if } l < x_i < u_i; \tag{3.13}$$
$$(\nabla f(x))_i \le 0, \text{ if } x_i = u_i; \tag{3.14}$$
$$(\nabla f(x))_i \ge 0, \text{ if } x_i = l_i. \tag{3.15}$$

Next define the matrix

$$D(x) := \begin{pmatrix} |v_1(x)|^{1/2} & & & \\ & |v_2(x)|^{1/2} & & \\ & & \ddots & \\ & & & |v_n(x)|^{1/2} \end{pmatrix} =: diag(|v(x)|^{1/2}),$$

where the vector $v(x) = (v_1(x), \ldots, v_n(x))^\top$ is defined as:

$$\begin{array}{ll} v_i := x_i - u_i, & \text{if } \nabla(f(x))_i < 0 \text{ and } u_i < \infty; \\ v_i := x_i - l_i, & \text{if } \nabla(f(x))_i \geq 0 \text{ and } l_i > -\infty; \\ v_i := -1, & \text{if } \nabla(f(x))_i < 0 \text{ and } u_i = \infty; \\ v_i := 1, & \text{if } \nabla(f(x))_i \geq 0 \text{ and } l_i = -\infty. \end{array} \tag{3.16}$$

Lemma 3.1.1. *A feasible point x satisfies the first order optimality conditions (3.13)-(3.15) iff*

$$D^2(x)\nabla f(x) = 0. \tag{3.17}$$

Therefore, solving the equation $D^2(x)\nabla f(x) = 0$ we obtain a point that satisfies the (KKT) condition for (NLP). Let $F(x) := D^2(x)\nabla f(x)$. Given the iterate x^k, to find $x^{k+1} = x^k + \alpha_k d^k$, a search direction d^k to solve the problem $F(x) = 0$ can be obtained by using the Newton method

$$J_F(x^k)d^k = -F(x^k), \tag{3.18}$$

where $J_F(\cdot)$ represents the Jacobian matrix of $F(\cdot)$. It is easy to see that

$$\begin{aligned} J_F(x^k) &= D_k^2 H_k + diag(\nabla f(x^k))J_v(x^k) \\ F(x^k) &= D^2(x^k)\nabla f(x^k), \end{aligned}$$

where

$$\begin{aligned} H_k &:= \nabla^2 f(x^k) \\ D_k &:= diag(|v(x^k)|^{1/2}) \\ J_v(x^k) &:= \begin{pmatrix} \nabla|v_1|^\top \\ \vdots \\ \nabla|v_n|^\top \end{pmatrix} \in \mathbb{R}^{n\times n} \end{aligned}$$

Remark 3.1.2. *Observe that,*

(a) since it is assumed that $x^k \in int(\mathcal{F})$, it follows that $|v(x^k)| > 0$. This implies, the matrix $D_k = D(x^k)$ is invertible.

(b) due to the definition of the vector $v(x)$, the matrix $J_v(x)$ is a diagonal matrix.

Hence, the system (3.18) can be written in the form

$$\begin{aligned} &\left(D_k^2 H_k + diag(\nabla f(x^k))J_v(x^k)\right) d^k = -D_k^2 \nabla f(x^k). \\ &\Rightarrow \left(D_k H_k D_k + \underbrace{D_k^{-1} diag(\nabla f(x^k))J_v(x^k) D_k}_{=diag(\nabla f(x^k))J_v(x^k)} \right) D_k^{-1} d^k = -D_k \nabla f(x^k) \\ &\Rightarrow \left(D_k H_k D_k + diag(\nabla f(x^k))J_v(x^k)\right) D_k^{-1} d^k = -D_k \nabla f(x^k). \end{aligned}$$

Now define

$$\begin{aligned} \hat{x}^k &:= D_k^{-1} x^k \\ \hat{d}^k &:= D_k^{-1} d^k \\ \hat{B}_k &:= D_k H_k D_k + diag(\nabla f(x^k))J_v(x^k) \\ \hat{g}^k &:= D_k \nabla f(x^k), \end{aligned}$$

where $\hat{x}^k = D_k^{-1}x^k$ defines an affine transformation of the variables x into $\hat{x}$ using $D(x)$. In general, we can also write $\hat{B}(x) := D(x)H(x)D(x) + diag(\nabla f(x))J_v(x)$ and $g(x) = D(x)\nabla f(x)$. Hence, the system (3.18) will be the same as

$$\hat{B}_k\hat{d}^k = -\hat{g}_k.$$

Lemma 3.1.3. *Let $x^* \in \mathcal{F}$.*

(i) If x^* is a local minimizer of (NLP), then $\hat{g}(x^*) = 0$;

(ii) If x^* is a local minimizer of (NLP), then $B(x^*)$ is positive definite and $\hat{g}(x^*) = 0$;

(iii) If $B(x^*)$ is positive definite and $\hat{g}(x^*) = 0$, then x^* is a local minimizer of (NLP).

Remark 3.1.4. *Thus statements (i) and (ii) of Lem. 3.1.3 imply that*

$$x^* \textbf{ is a solution of (NLP) } \Leftrightarrow \hat{g}(x^*) = 0 \textbf{ and } \hat{B}(x^*) \textbf{ is positive definite}.$$

It follows that, through the affine transformation $\hat{x} = D^{-1}x$, the problem (NLP) has been transformed into an unconstrained minimization problem with gradient vector $\hat{g}$ and Hessian matrix $\widehat{B}$. Consequently, a local quadratic model for the transformed problem can be given by the trust region problem:

$$(QP)_{TR} \qquad \begin{array}{l} \psi(\hat{s}) = \frac{1}{2}\hat{s}\hat{B}_k\hat{s} + \hat{g}_k^\top\hat{s} \to \min, \\ s.t. \\ \qquad \|\hat{s}\| \le \triangle_k. \end{array}$$

Furthermore, the system $\hat{B}_k\hat{d}^k = -\hat{g}_k$ can be considered as the first order optimality condition (considering $\hat{d}^k$ as a local optimal point) for the unconstrained problem (QP)$_{TR}$ with $\|\hat{d}^k\| < \triangle_k$.

Retransforming variables the problem $(QP)_{TR}$

$$\begin{array}{l} \frac{1}{2}\hat{s}[D_kH_kD_k + diag(\nabla f(x^k))J_v(x^k)]\hat{s} + (D_k\nabla f(x^k))^\top\hat{s} \to \min, \\ s.t. \\ \qquad \|s\| \le \triangle_k. \end{array}$$

$\Rightarrow$

$$\begin{array}{l} \frac{1}{2}\hat{s}D_k[H_k + D_k^{-1}diag(\nabla f(x^k))J_v(x^k)D_k^{-1}]D_k\hat{s} + f(x^k)^\top D_k\hat{s} \to \min, \\ s.t. \\ \qquad \|s\| \le \triangle_k. \end{array}$$

with $s = D_k\hat{s}$ and

$$B_k := H_k + D_k^{-1}diag(\nabla f(x^k))J_v(x^k)D_k^{-1}$$

we obtain the following quadratic problem in terms of the original variables

$$(QP)_{TRO} \qquad \begin{array}{l} \frac{1}{2}sB_ks + \nabla f(x^k)^\top s \to \min, \\ s.t. \\ \qquad \|D_k^{-1}s\| \le \triangle_k. \end{array}$$

Consequently, given x^k the problem (QP)$_{TRO}$ is solved to determine s^k so that

$$x^{k+1} = x^k + \alpha_k s^k,$$

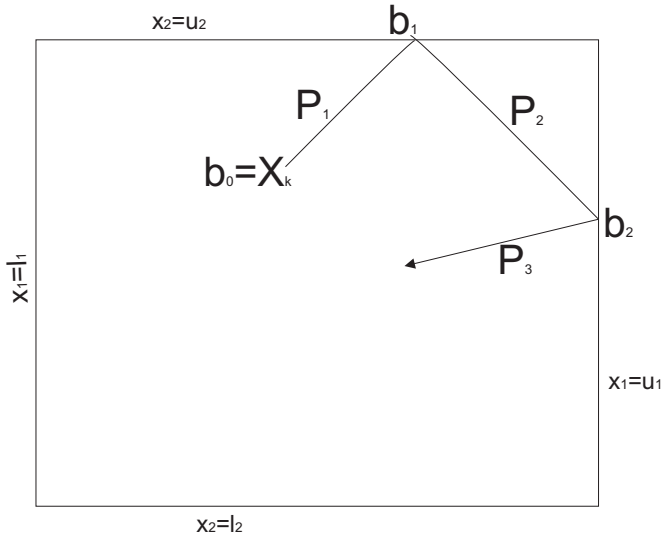
for some step length α_k.

Reflective Line Search

The choice of the step length α_k depends on how far $x^k + s^k$ lies from the boundary of the bounding box $\mathcal{F} = \{x \mid l \leq x \leq u\}$. Since iterates are assumed to lie in the interior of the box, i.e. $int\mathcal{F}$, when an iterate lies on the boundary it will be reflected into the interior of the box.

For given x^k and s^k, then define

$$r_k = \begin{bmatrix} max\{(x_1^k - l_1)/s_1^k, (u_1 - x_1^k)/s_1^k\} \\ \vdots \\ max\{(x_n^k - l_n)/s_n^k, (u_n - x_1^k)/s_n^k\} \end{bmatrix}$$



The reflective path (RP)

Step 0: Set $\beta_k^0 = 0, p_1^k = s^k, b_0^k = x^k$.

Step 1: Let β_k^i be the distance from x^k to the nearest boundary point along s^k:

$$\beta_i^k = \min\{r_i^k \mid r_i^k > 0\}.$$

Step 2: Define the i-th boundary point as: $b_i^k = b_{i-1}^k + (\beta_i^k - \beta_{i-1}^k)p_i^k$.

Step 3: Reflect to get a new direction and update the vector r^k:

(a) $p_{i+1}^k = p_i^k$

(b) For each j such that $(b_i^k)_j = u_j (or (b_i^k)_j = l_j)$

- $r_j^k = r_j^k + |\frac{u_j - l_j}{s_j}|$
- $(p_{i+1}^k) = -(p_i^k)_j$ (reflection)

Observe that in the above $\beta_{i-1}^k < \beta_i^k, i = 1, 2, \ldots$ Thus, the **reflective search path** (direction) is defined as

$$p^k(\alpha) = b_{i-1}^k + (\alpha - \beta_{i-1}^k)p_i^k, \text{ for } \beta_{i-1}^k \leq \alpha < \beta_i^k.$$

The interior-reflective method (TIR)

Step 1: Choose $x_1 \in int(\mathcal{F})$.

Step 2: Determine a descent direction s_k for f at $x_k \in int(\mathcal{F})$. Then determine the reflective search path $p^k(\alpha)$ using the algorithm (RP).

Step 2: Solve $f(x^k + p^k(\alpha)) \to \min$ to determine α_k in such a way that $x^k + p^k(\alpha_k)$ is not a boundary point of $\mathcal{F}$.

Step 3: $x^{k+1} = x^k + p^k(\alpha_k)$.

Under additional assumptions, the algorithm (TIR) has a global super-linear and local quadratic convergence properties. For details see the papers of Coleman & Li ([1] - [3]).

Multiple image restoration and enhancement

3.2 Using quadprog to Solve QP Problems

3.2.1 Theoretical Problems

1. Solve the following quadratic optimization by using `quadprog.m`

$$\begin{array}{llcl} x_1^2 + 2x_2^2 + & 2x_1 + & 3x_2 & \to \min \\ & \text{s.t.} & & \\ & x_1 + & 2x_2 & \leq 8 \\ & 2x_1 + & x_2 & \leq 10 \\ & & x_2 & \leq 3 \\ & x_1, & x_2 & \geq 0 \end{array}$$

Soution:

$$Q = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}, q = \begin{pmatrix} 2 \\ 3 \end{pmatrix}, A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 0 & 1 \end{pmatrix}, a = \begin{pmatrix} 8 \\ 10 \\ 3 \end{pmatrix}, B = [\,], b = [\,], lb = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, ub = \begin{pmatrix} \infty \\ \infty \end{pmatrix}$$

Matlab Solution

```
function QpEx1

Q=[2,0;0,4]; q=[2,3]'; A=[1,2;2,1;0,1]; a=[8,10,3];"

lb=[0,0]'; ub=[inf;inf];

options=optimset('quadprog');

options=optimset('LargeScale','off');
[xsol,fsolve,exitflag,output]=QUADPROG(Q,q,A,a,[],[],lb,ub,[],options);
```

```
fprintf('%s ','Convergence: ')
if exitflag > 0
  fprintf('%s \n','Ja!');
  disp('Solution obtained:')
xsol
else
fprintf('%s \n','Non convergence!');
end

fprintf('%s %s \n','Algorithm used: ',output.algorithm)
x=[-3:0.1:3]; y=[-4:0.1:4]; [X,Y]=meshgrid(x,y);
Z=X.^2+2*Y.^2+2*X+3*Y; meshc(X,Y,Z); hold on
plot(xsol(1),xsol(2),'r*')
```

2. Solve the following (QP) using `quadprog.m`

$x_1^2 + x_1x_2 + 2{x_2}^2$	$+2x_3^2$	$+2x_2x_3$	$+4x_1+$	$6x_2$	$+12x_3$	$\rightarrow$	min
	s.t.						
	x_1	$+x_2$	$+x_3$	$\geq$	6		
	$-x_1$	$-x_2$	$+2x_3$	$\geq$	2		
	$x_1,$	x_2	x_3	$\geq$	0		

Soution:

$$Q = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 4 & 2 \\ 0 & 2 & 4 \end{pmatrix}, q = \begin{pmatrix} 4 \\ 6 \\ 12 \end{pmatrix}, A = \begin{pmatrix} -1 & -1 & -1 \\ 1 & 1 & -2 \end{pmatrix}, a = \begin{pmatrix} -6 \\ -12 \end{pmatrix},$$

$$B = [\,], b = [\,], lb = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, ub = \begin{pmatrix} \infty \\ \infty \\ \infty \end{pmatrix}$$

Matlab Solution

```
function QpEx2

Q=[2,1,0;1,4,2;0,2,4]; q=[4,6,12]; A=[-1,-1,-1;1,2,-2]; a=[-6,-2];

lb=[0;0;0]'; ub=[inf;inf;inf];

options=optimset('quadprog');

options=optimset('LargeScale','off');
[xsol,fsolve,exitflag,output]=QUADPROG(Q,q,A,a,[],[],lb,ub,[],options);
fprintf('%s ','Convergence: ')
if exitflag > 0
```

```
  fprintf('%s \n','Ja!');
  disp('Solution obtained:')
xsol
else
fprintf('%s \n','Non convergence!');
end

fprintf('%s %s \n','Algorithm used: ',output.algorithm)
```

3.2.2 Production model - profit maximization

Let us consider the following production model. A factory produces $n-$ articles A_i ,$i = 1, 2, ..., n$, The cost function $c(x)$ for the production process of the articles can be desribed by

$$c(x) = k_p^T x + k_f + k_m(x)$$

where k_p, k_f and k_m denote the variable production cost rates, the fix costs and the variable costs for the material. Further assume that the price p of a product depends on the number of products x in some linear relation

$$p_i = a_i - b_i x_i,\ a_i, b_i > 0,\ \ i = 1, 2, ..., n$$

The profit $\Phi(x)$ is given as the difference of the turnover

$$T(x) = p(x)^T x^T = \sum_{i=1}^{n} \left(a_i x_i - b_i x_i^2\right)$$

and the costs $c(x)$. Hence

$$\begin{aligned}
\Phi(x) &= \sum_{i=1}^{n} \left(a_i x_i - b_i x_i^2\right) - \left(k_p^T x + k_f + k_m(x)\right) \\
&= \frac{1}{2} x^T \underbrace{\begin{pmatrix} -2b_1 & 0 & \cdots & 0 \\ 0 & -2b_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -2b_n \end{pmatrix}}_{:=\widetilde{B}} x + (a - k_p)^T x - k_f - k_m(x)
\end{aligned}$$

Further we have the natural constraints

$$a_i - b_i x_i \geq 0,\ \ i = 1, 2, ..., n \Rightarrow \underbrace{\begin{pmatrix} b_1 & 0 & \dots & 0 \\ 0 & \dots & & 0 \\ \vdots & \ddots & & \vdots \\ 0 & \dots & 0 & b_n \end{pmatrix}}_{B'} x \leq a$$

$$k_p^T x + k_f \leq k_0$$
$$0 < x_{\min} \leq x \leq x_{\max}$$

and constraints additional constraints on resources. There are resources $B_j, j = 1, \dots, m$ with consumption amount $y_j, j = 1, 2, ..., m$. Then the $y_j's$ and the amount of final products x_i have the linear connection

$$y = Rx$$

Further we have some bounds

$$0 < y_{\min} \leq y \leq y_{\max}$$

based on the minimum resource consumption and maximum available resources. When we buy resources B_j we have to pay the following prices for y_j units of B_j

$$(c_j - d_j y_j)\, y_j, j = 1, 2, ..., m$$

which yields the cost function

$$\begin{aligned} k_m(x) &= \sum_{j=1}^{m} (c_j - d_j y_j)\, y_j \\ &= c^T y + \frac{1}{2} y^T \begin{pmatrix} -2d_1 & 0 & \cdots & 0 \\ 0 & -2d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -2d_m \end{pmatrix} y \\ &= c^T Rx + \frac{1}{2} x^T R^T \underbrace{\begin{pmatrix} -2d_1 & 0 & \cdots & 0 \\ 0 & -2d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -2d_m \end{pmatrix}}_{:=D} Rx \end{aligned}$$

We assume some production constraints given by

$$\begin{aligned} Ax &\leq r \\ Bx &\geq s. \end{aligned}$$

If we want to maximize the profit (or minimize loss) we get the following quadratic problem

$$\begin{aligned} &\frac{1}{2} x^\top \left(\widetilde{B} + R^\top D R \right) x + (a - k_p + R^\top c)^\top x - k_f \to \max \\ &s.t. \\ &\quad B'x \leq a \\ &\quad k_p^\top x \leq k_0 - k_f \\ &\quad Rx \geq y_{min} \\ &\quad Rx \leq y_{\max} \\ &\quad Ax \leq r \\ &\quad Bx \geq s \\ &\quad 0 < x_{\min} \leq x \leq x_{\max}. \end{aligned}$$

Use hypothetical, but properly chosen, data for A,B,R,a,b,c,d,r,s,xmin,xmax,kp,k0,kf to run the following program to solve the above profit maximization problem.

```
function [xsol,fsolve,exitflag,output]=profit(A,B,R,a,b,c,d,r,s,xmin,xmax,kp,k0,kf)

%[xsol,fsolve,exitflag,output]=profit(A,B,R,a,b,c,d,r,s,xmin,xmax,kp,k0,kf)
%solves a profit maximization prblem
```

```
Bt=-2*diag(b); D=-2*diag(d);

%%1/2x'Qx + q'x
Q=Bt + R'*D*R; q=a-kp+R'*c;

%coefficeint matrix of inequality constraints

A=[diag(b);kp';-R;R;A;-B];

%right hand-side vector of inequality constraints
a=[a;k0-kf;-ymin;ymax;r;-s];
%Bound constraints
lb=xmin(:); ub=xmax(:);

%Call quadprog.m to solve the problem

options=optimset('quadprog');
options=optimset('LargeScale','off');
[xsol,fsolve,exitflag,output]=quadprog(-Q,-q,A,a,[],[],lb,ub,[],options);
```

Bibliography

[1] T. F. Coleman, Y. Li: On the convergence of interior-reflective Newton methods for non-linear optimization subjecto to bounds. Math. Prog., V. 67, pp. 189-224, 1994.

[2] T. F. Coleman, Y. Li: A reflective Newton method for minimizing a quadratic function subject to bounds on some of the variables. SIAM Journal on Optimization, Volume 6 , pp. 1040 - 1058, 1996.

[3] T. F. Coleman, Y. Li: An interior trust region approach for nonlinear optimization subject to bounds. SIAM Journal on Optimization, SIAM Journal on Optimization, V. 6, pp. 418-445, 1996.

Chapter 4

Unconstrained nonlinear programming

4.1 Theory, optimality conditions

4.1.1 Problems, assumptions, definitions

Uncostrained Optimization problem:

$$(NLP_U) \qquad \begin{array}{l} f(x_1, x_2, \ldots, x_n) \longrightarrow \min \\ x \in \mathbb{R}^n. \end{array}$$

Assumptions:

- Smooth problem:

$$\boxed{\begin{array}{c} f \in C^k,\ k \geq 1 \\ M \text{ open set} \end{array}}$$

- Convex problem:

$$\boxed{\begin{array}{ll} f : M \subset \mathbb{R}^n \longrightarrow \mathbb{R} & \\ M & \text{convex set} \\ f & \text{convex function} \end{array}}$$

- nonsmooth problem (d.c. programming):

$$\boxed{\begin{array}{ll} f & \text{locally Lipschitzian} \\ f & \text{difference of convex functions} \end{array}}$$

M convex:

$$\boxed{\begin{array}{l} \text{For all } x, y \in M, \lambda \in [0,1]: \\ \lambda x + (1-\lambda)\, y \in M \end{array}}$$

f convex:

$$\boxed{\begin{array}{l} M \text{ convex and for all } x, y \in M, \lambda \in [0,1]: \\ f(\lambda x + (1-\lambda)\, y) \leq \lambda f(x) + (1-\lambda)\, f(y) \end{array}}$$

f Lipschitzian:

$$\boxed{\begin{array}{l}\text{There is a } K \geq 0 \text{ such that for all } x, y \in M: \\ |f(x) - f(y)| \leq K \|x - y\| \\ \|x - y\| = \sqrt{(x-y)^T (x-y)}\end{array}}$$

Locally Lipschitzian: If, for any $x \in M$, there is a ball $B(x)$ around x and a constant K such that f is Lipschitzian on $M \cap B$.

4.2 Optimality conditions for smooth unconstrained problems

Necessary condition: Assumption $f \in C^1$

$$\boxed{\begin{array}{ll} & f \text{ has a local minimum at } \bar{x} \in M \\ \iff & \text{There is a ball } B \text{ around } \bar{x} \text{ such that} \\ & f(x) \leq f(\bar{x}) \quad \text{for all } x \in M \cap B(x) \\ \implies & \dfrac{\partial}{\partial x_k} f(\bar{x}) = 0, \quad k = 1, 2, ..., n \end{array}}$$

Sufficient condition: Assumption $f \in C^2$

$$\boxed{\begin{array}{ll} (1) & \dfrac{\partial}{\partial x_k} f(\bar{x}) = 0, \quad k = 1, 2, ..., n \\ (2) & H_f(\bar{x}) = \nabla^2 f(\bar{x}) = \left(\dfrac{\partial^2}{\partial x_j \partial x_k} f(\bar{x}) \right)_{nn} \\ & \text{is positively definite} \\ \implies & f \text{ has a strict local minimum at } \bar{x} \in M \end{array}}$$

Remark: A symmetric Matrix H is positively definite $\iff$ all eigenvalues of H are greater than zero.

MATLAB call:

$$\boxed{\lambda = \text{eig}(H)}$$

computes all eigenvalues of the Matrix H and stores them in the vector λ.

Saddle point: Assumption $f \in C^2$

$$\boxed{\begin{array}{ll} (1) & \dfrac{\partial}{\partial x_k} f(\bar{x}) = 0, \quad k = 1, 2, ..., n \\ (2) & H_f(\bar{x}) = \nabla^2 f(\bar{x}) = \left(\dfrac{\partial^2}{\partial x_j \partial x_k} f(\bar{x}) \right)_{nn} \\ & \text{has only nonzero eigenvalues but at least } \textbf{two} \\ & \text{with } \textbf{different sign} \\ \implies & f \text{ has a saddle point at } \bar{x} \in M \end{array}}$$

4.3 Matlab Function for Unconstrained Optimization

There are two Matlab functions to solve unconstrained optimization problems

- `fminunc.m` - when the function f, to be minimized, has at least a continuous gradient vector. The `fminunc.m` uses descent direction methods which are known as quasi-Newton methods along with line search strategies based on quadratic or cubic interpolations.
- `fminsearch.m` - when the function f has no derivative information, or has discontinuities or if f is highly non-linear.

4.4 General descent methods - for differentiable Optimization Problems

A level set of f:

$$\boxed{N_f(\alpha) = \{x \in M \mid f(x) \leq \alpha\}}$$

Gradient of f: (in orthonormal cartesian coordinates)

$$\boxed{\nabla f(x) = \left(\frac{\partial}{\partial x_k} f(x)\right)_n = \begin{pmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{pmatrix}}$$

- The gradient vector $\nabla f(x)$ at x is a vector orthogonal to the tangent hyperplane of f at x.
- The gradient at x is the direction in which the function f has locally the largest descent at x; i.e. the vector $\nabla f(x)$ points into the direction in which f decreases most rapidly.

Descent direction: A vector d is a descent for f at x if the angle between $\nabla f(x)$ and d is larger than 180^o:

$$\boxed{\nabla f(x)^T d < 0}$$

Direction of steepest descent:

$$\boxed{d = -\nabla f(x)}$$

A general descent algorithm:

(1)	choose start vector $x^1 \in M,\ k = 1,$
	tolerance ε
While	$\left\|\nabla f\left(x^k\right)\right\| > \varepsilon$
$(k1)$	choose descent direction d^k
$(k2)$	choose step length $\alpha_k > 0$ such that
	$f\left(x^k\right) > f\left(x^k + \alpha_k d^k\right)$
$(k3)$	$x^{k+1} = x^k + \alpha_k d^k$
	$k = k + 1$
end	
	x^k is approximate solution of (UP)

Methods for unconstrained optimization differ in:

(k1) the choice of the descent direction d^k

(k2) the choice of the step length α_k.

4.5 The Quasi-Newton Algorithm -idea

4.5.1 Determination of Search Directions

At each iteration step $k \in \{1, 2, \ldots\}$, the quasi-Newton method determines the search direction d^k using

$$d^k = -H_k^{-1} \nabla f(x^k),$$

where H_k is a symmetric positive definite matrix that approximates the hessian matrix $\nabla^2 f(x^k)$ of f at x^k in such a way that the condition

$$H_k \left(x^k - x^{k-1} \right) = \nabla f(x^k) - \nabla f(x^{k-1}) \qquad \text{(quasi-Newton condition (QNC1))}$$

is satisfied. It follows that

$$H_{k+1} \left(x^{k+1} - x^k \right) = \nabla f(x^{k+1}) - \nabla f(x^k).$$

If we set $s^k := x^{k+1} - x^k$ and $q^k := \nabla f(x^{k+1}) - \nabla f(x^k)$, then we obtain

$$H_{k+1} s^k = q^k.$$

To determine a symmetric positive matrix H_{k+1}, given that H_k is symmetric positive definite matrix H_k and satisfies the (QNC1), we must have

$$0 < \left(s^k \right)^\top H_{k+1} s^k = \left(s^k \right)^\top q^k.$$

$\Rightarrow$

$$\left(s^k \right)^\top q^k > 0 \qquad \text{(quasi-Newton condition (QNC2))}$$

Thus, given a symmetric positive definite matrix H_k, at each successive iteration the matrix H_{k+1} is determined using *a matrix updating Formula*:

1. **The BFGS - update (Broyde-Fletcher-Goldfarb-Shanon)**

$$H_{k+1} = H_k - \frac{(H_k s^k)(H_k s^k)^\top}{(s^k)^\top H_k s^k} + \frac{q^k (q^k)^\top}{(q^k)^\top s^k},$$

where $s^k := x^{k+1} - x^k$ and $q^k := \nabla f(x^{k+1}) - \nabla f(x^k)$. Thus, given x^k and α_k, to determine

$$x^{k+1} = x^k + \alpha_k d^{k+1}$$

the search direction d^{k+1} is found using

$$d^{k+1} = -H_{k+1}^{-1} \nabla f(x^k).$$

However, in the above formula, the determination of H_{k+1}^{-1} at each step might be computationally expensive. Hence, d^{k+1} can also be determined by using

2. **the inverse DFP - update (Davidon-Fletcher-Powell)**

$$B_{k+1} = B_k - \frac{(B_k q^k)^\top}{(q^k)^\top B_k q^k} + \frac{d^k (d^k)^\top}{(d^k)^\top q^k}$$

where $s^k := x^{k+1} - x^k$ and $q^k := \nabla f(x^{k+1}) - \nabla f(x^k)$. In this case, given x^k and α_k, to determine

$$x^{k+1} = x^k + \alpha_k d^{k+1}$$

the search direction d^{k+1} is found using

$$d^{k+1} = -B_{k+1} \nabla f(x^k).$$

The following hold true:

- for, each $k \in \{1, 2, \dots, \}$, the search direction d^k, that is determined using either BFGS or DFP is a descent direction for f at x^k, i.e. $\nabla f(x^k)^\top d^k < 0$;
- if H_k and B_k are symmetric and positive definite and $(s^k)^\top q^k > 0$, then the matrices H_{k+1} and B_{k+1} are symmetric and positive definite;
- if $x^k \to x^*$, then sequence of matrices $\{H_k\}$ converges to $\nabla^k f(x^*)$ and $\{B_k\}$ converges to $(\nabla^2 f(x^*))^{-1}$.

But note that the determination of H_k and B_k does not require the hessian matrix $\nabla^2 f(x^k)$.

4.5.2 Line Search Strategies- determination of the step-length α_k

In general, for the step length α_k in $x^{k+1} = x^k + \alpha_k x^k$ to be acceptable the relation

$$f(x^k + \alpha_k x^k) = f(x^{k+1}) < f(x^k) \tag{4.1}$$

should be satisfied. That is the value of the function must reduce at x^{k+1} sufficiently as compared to its value at x^k. Hence, at each step $k \in \{1, 2, \dots\}$, α_k should be chosen so that this condition is satisfied. Moreover, to guarantee that the matrices H_k and B_k are symmetric and positive definite we need to have

$$0 < (q^k)^\top s^k = \alpha_k \left(\nabla f(x^{k+1}) - \nabla f(x^k) \right)^\top d^k = \alpha_k (\nabla f(x^{k+1})^\top d^k - \nabla f(x^k)^\top d^k).$$

$\Rightarrow$

$$0 < (q^k)^\top s^k = \alpha_k \nabla f(x^{k+1})^\top d^k - \alpha_k \nabla f(x^k)^\top d^k.$$

Note that, since d^k is a descent direction we have, for $\alpha_k > 0$, that $-\alpha_k \nabla f(x^k)^\top d^k > 0$. Consequently, $(q^k)^\top s^k$ will be positive if α_k is chosen so that $\alpha_k \nabla f(x^{k+1})^\top d^k$ is smaller as compared to $-\alpha_k \nabla f(x^k)^\top d^k$. Nevertheless, it is not clear whether

$$\nabla f(x^{k+1})^\top d^k \tag{4.2}$$

is positive or not. Hence, the determination of the step length α_k is based up on the satisfaction of (4.1) and on the signs of $\nabla f(x^{k+1})^\top d^k$. Thus, given an α_k if one of the conditions

$$f(x^k + \alpha_k d^k) = f(x^{k+1}) < f(x^k) \quad \text{or} \quad 0 < (q^k)^\top s^k$$

fails to hold this α_k is not acceptable. Hence, α_k should be adjust to meet these conditions. Considering the function $\varphi(\alpha) := f(x^k + \alpha d^k)$, if we could determine a minimum of φ (in some interval), then the condition $f(x^{k+1}) < f(x^k)$ can be satisfied. However, instead of directly minimizing the function φ, Matlab uses quadratic and cubic interpolating functions of φ to approximate the minima. (See for details the Matlab Optimization Toolbox users guide).

4.6 Trust Region Methods - idea

Suppose the unconstrained non-linear optimization problem

$$f(x) \to \min$$
$$s.t.\ \ x \in \mathbb{R}^n$$

with $f \in \mathcal{C}^2(\mathbb{R}^n)$ be given. For a known iterate x^k the trust region method determines subsequent iterates using

$$x^{k+1} = x^k + d^k$$

where d^k is determined by minimizing a local quadratic (approximating) model of f at x^k given by

$$q_k(x) := f(x^k) + \nabla f(x^k)^\top d + \frac{1}{2} d^\top H_k d$$

constrained to a domain Ω, where we expect the resulting direction vector d^k could yield a "good" reduction of f at $x^{k+1} = x^k + d^k$. The constraint set Ω is usually given by

$$\Omega_k = \{d \in \mathbb{R}^n \mid \|d\| \leq \triangle_k\}$$

and is known as the *trust-region*[1], where we hope (trust) that the **global** solution of the problem

$$\begin{aligned} (QPT)_k \qquad & q_k(d) := f(x^k) + \nabla f(x^k)^\top d + \frac{1}{2} d^\top \nabla^2 f(x^k) d \to \min \\ & s.t. \\ & d \in \Omega = \{d \in \mathbb{R}^n \mid \|d\| \leq \triangle_k\} \end{aligned}$$

yields a direction vector d^k that brings a "good" reduction of f at $x^{k+1} = x^k + d^k$. Thus, the problem $(QPT)_k$ is widely known as the *trust region problem* associated with (NLP) at the point x^k and $\triangle_k > 0$ is the radius of the *trust-region*.

A general trust-region algorithm

$(k0)$	choose start vector x^0 parameters $\overline{\triangle}, \triangle_0 \in (0, \overline{\triangle}), 0 < \eta_1 \leq \eta_2 < 1$ $0 < \gamma_1 < 1 < \gamma_2$, tolerance $\varepsilon > 0$, and $k = 0$.
WHILE	$\left\|\nabla f\left(x^k\right)\right\| > \varepsilon$
$(k1)$	Approximately solve (QPT)$_k$ to determine d^k.
$(k2)$	Compute • $r_k = \frac{f(x^k) - f(x^k + d^k)}{q_k(0) - q_k(d^k)}$ • Set $x^{k+1} = \begin{cases} x^k + d^k, & \text{if } r_k \geq \eta_1, \\ x^k, & otherwise. \end{cases}$
$(k3)$	Adjust the trust-region radius $\triangle_{k+1}$ • If $r_k < \eta_1$, then $\triangle_{k+1} :\in (0, \gamma_1 \triangle_k)$ (Shrink trust-region). • If $\eta_1 \leq r_k < \eta_2$, then $\triangle_{k+1} \in [\gamma_1 \triangle_k, \triangle_k]$ (Shrink or accept the old trust-region). • If $r_k \geq \eta_2$ and $\|d^k\| = \triangle_k$, then $\triangle_{k+1} \in [\triangle_k, \min\{\gamma_2 \triangle_k, \overline{\triangle_k}\}]$ (Enlarge trust-region).
$(k4)$	Set $k = k + 1$ and update the quadratic model (QPT)$_k$; i.e., update $q_k(\cdot)$ and the trust-region.
END	

[1] The norm $\|\cdot\|$ is usually assumed to be the Euclidean norm $\|\cdot\|_2$.

In the above algorithm the parameter $\overline{\triangle}$ is the overall bound for the trust region radius $\triangle_k$ and the expression:

$$r_k = \frac{f(x^k) \ - \ f(x^k + d^k)}{q_k(0) \ - \ q_k(d^k)}$$

measures how best the quadratic model approximates the unconstrained problem (NLP). Thus,

- $ared_k := f(x^k) - f(x^k + d^k)$ is known as the *actual reduction* of f at step $k+1$; and
- $pred_k := q_k(0) - q(d^k)$ is the *predicted reduction* of f achievable through the approximating model (QPT)$_k$ at step $k+1$.

Consequently, r_k measures the ratio of the actual reduction to the predicted reduction. Hence, at step $k+1$,

- if $r_k \geq \eta_2$, then this step is called a **very successful step**. Accordingly, the trust-region will be enlarged; i.e. $\triangle_{k+1} \geq \triangle_k$. In particular, a sufficient reduction of f will be achieved through the model, since
$$f(x^k) \ - \ f(x^k + d^k) \geq \eta_2(q_k(0) \ - \ q_k(d^k)) > 0.$$
- if $r_k \geq \eta_1$, then this step is called a **successful step** since
$$f(x^k) \ - \ f(x^k + d^k) \geq \eta_1(q_k(0) \ - \ q_k(d^k)) > 0.$$
Accordingly, the search direction is accepted, for it brings a sufficient decrease in the value of f; i.e. $x^{k+1} = x^k + d^k$ and the trust-region can remain as it is for the next step.
- However, if $r_k < \eta_1$, then the step is called **unsuccessful**. That is the direction d^k does not provide a sufficient reduction of f. This is perhaps due to a too big trust-region (i.e. at this step the model (QPT)$_k$ is *not trustworthy*). Consequently, d^k will be rejected and so $x^{k+1} = x^k$ and for the next step the trust-region radius will be reduced.

Note, that in general, if $r_k \geq \eta_1$ or $r_k \geq \eta_2$, then $x^{k+1} = x^k + d^k$ and the corresponding step is successful. In any case, the very important issue in a trust-region algorithm is how to solve the trust-region sub-problem (QPT)$_k$.

4.6.1 Solution of the Trust-Region Sub-problem

Remark 4.6.1. *Considering the trust-region sub-problem (QPT)$_k$:*

- *since, for each k, $q_k(\cdot)$ is a continuous function and $\Omega = \{d \in \mathbb{R}^n \mid \|d\| \leq \triangle_k\}$ is a compact set, the problem (QPT)$_k$ has always a solution.*
- *Furthermore, if f is not a convex function, then the hessian matrix $\nabla^2 f(x^k)$ of f at x^k may not be positive definite. Thus, the global solution of (QPT)$_k$ may not exist.*

The existence of a global solution for (QPT)$_k$ is guaranteed by the following statement:

Theorem 4.6.2 (Thm. 2.1. [3], Lem. 2.8. [10], also [4])**.**
Given a quadratic optimization problem

$$(QP) \qquad \begin{aligned} & q(d) := f + g^\top d + \frac{1}{2} d^\top H d \to \min \\ & s.t. \\ & \quad d \in \Omega = \{d \in \mathbb{R}^n \mid \|d\| \leq \triangle\} \end{aligned}$$

with $f \in \mathbb{R}, g \in \mathbb{R}^n$ and H a symmetric matrix and $\triangle > 0$. Then $d^* \in \mathbb{R}^n$ is a global solution of (QP) if and only if, there is a (unique) λ^* such that

(a) $\lambda^* \geq 0$, $\|d^*\| \leq \triangle$ *and* $\lambda^*(\|d^*\| - \triangle) = 0$*;*

(b) $(H + \lambda^* I)d^* = -g$ *and* $(H + \lambda^* I)$ *is positive definite;*

where I *is the identity matrix.*

The statements of Thm. 4.6.2 describe the optimality conditions of d^k as a solution of the trust-region sub-problem $(\text{QPTR})_k$. Hence, the choice of d^k is based on whether the matrix $H_k = \nabla^2 f(x^k)$ is positive definite or not.

Remark 4.6.3. *Note that,*

(i) if H *is a positive definite matrix and* $\|H_k^{-1} g_k\| < \triangle$*, then for the solution of* d^* *of (QP),* $\|d^k\| = \triangle$ *does note hold; i.e., in this case the solution of (QP) does not lie on the boundary of the feasible set. Assume that* H_k *is a positive definite,* $\|H_k^{-1} g\| < \triangle$ *and* $\|d^k\| = \triangle$ *hold at the same time. Hence, from Thm. 4.6.2(b), we have that*

$$(I + \lambda_* H_k^{-1})d^* = -H_k^{-1} g_k \Rightarrow {d^k}^\top d^* + \lambda_k {d^k}^\top H_k^{-1} d^k = -{d^k}^\top H_k^{-1} g_k,$$

where $g_k := \nabla f(x^k)$*. By the positive definiteness of* H^{-1} *we have*

$$ {d^k}^\top d^k + \frac{\lambda_k}{maxEig(H_k)} {d^k}^\top d^k \leq -{d^k}^\top H_k^{-1} g_k$$

$\Rightarrow$

$$\left(1 + \frac{\lambda_k}{maxEig(H_k)}\right) \|d^k\|^2 \leq \| - {d^k}^\top \| \|H_k^{-1} g_k\| < \|d^k\| \triangle = \|d^k\|^2,$$

where $maxEig(H_k)$ *is the largest eigenvalue of* H*. Hence,* $\left(1 + \frac{\lambda_k}{maxEig(H_k)}\right) \|d^k\|^2 < \|d^k\|^2$*. But this is a contradiction, since* $\lambda_k \geq 0$*.*

(ii) Conversely, if (QP) has a solution d^* *such that* $\|d^k\| < \triangle$*, it follows that* $\lambda_k = 0$ *and* H_k *is positive definite. Obviously, from Thm. 4.6.2(a), we have* $\lambda_k = 0$*. Consequently,* $H_k + \lambda_k I$ *is positive definite, implies that* H_k *is positive definite.*

In general, if H_k *is positive definite, then* $H + \lambda_* I$ *is also positive definite.*

The Matlab trust-region algorithm tries to find a search direction d^k in a two-dimensional subspace of $\mathbb{R}^n$. Thus, the trust-region sub-problem has the form:

$$\begin{aligned} (QPT)_k \qquad & q_k(d) := f(x^k) + g_k^\top d + \frac{1}{2} d^\top H_k d \to \min \\ & s.t. \\ & \quad d \in \Omega = \{d \in \mathbb{R}^n \mid \|d\| \leq \triangle_k, d \in \mathcal{S}_k\}, \end{aligned}$$

where $\mathcal{S}$ is a two-dimensional subspace of $\mathbb{R}^n$; i.e. $\mathcal{S}_k =< u_k, v_k >$; with $u, v \in \mathbb{R}^n$. This approach reduces computational costs of when solving large scale problems [**?**]. Thus, the solution algorithm for $(\text{QPT})_k$ chooses an appropriate two-dimensional subspace at each iteration step.

Case -1: If H_k is positive definite, then

$$\mathcal{S}_k = \left\langle g_k, -H_k^{-1} g_k \right\rangle$$

Case -2 : If H_k is not positive definite, then

$$\mathcal{S}_k = \langle g_k, u_k \rangle$$

such that

Case - 2a: either u_k satisfies

$$H_k u_k = -g_k;$$

Case - 2b: or u_k is chosen as a direction of negative-curvature of f at x^k; i.e.

$$u_k^\top H_k u_k < 0.$$

Remark 4.6.4. *Considering the case when H_k is not positive definite, we make the following observations.*

(i) If the H_k has negative eigenvalues, then let λ_1^k be the smallest negative eigenvalue and w_k be the corresponding eigenvector . Hence, we have

$$H_k w_k = \lambda_1 w_k \Rightarrow w_k^\top H_k w_k = \lambda_1^k \|w_k\|^2 < 0.$$

satisfying Case 2a. Furthermore,

$$(H_k + (-\lambda_1^k)I)w_k = 0.$$

Since $w_k \neq 0$, it follows that $(H_k + (-\lambda_1^k)I)$ is not positive definite. Hence, in the Case 2a, the vector u_k can be chosen to be equal to w_k. Furthermore, the matrix $(H_k + (-\lambda_1^k)I)$ is also not positive definite.

(ii) If all eigenvalues of H_k are equal to zero, then

(iia) if g_k is orthogonal to the null-space of H_k, then there is a vector u_k such that

$$H_k u_k = -g_k.$$

This follows from the fact that H_k is a symmetric matrix and $\mathbb{R}^n = \mathcal{N}(H_k) \bigoplus \mathcal{R}(H_k)$[2].

(iia) Otherwise, there is always a non-zero vector $u_k \neq 0$ such that $u_k^\top H_k u_k \leq 0$.

(iii) The advantage of using a direction of negative curvature is to avoid the convergence of the algorithm to a saddle point or a maximizer of (QPTR)$_k$

A detailed discussion of the algorithms for the solution of the trust-region quadratic sub-problem (QPT)$_k$ are found in the papers [1, 9] in the recent book of Conn et al. [2].

4.6.2 The Trust Sub-Problem under Considered in the Matlab Optimization toolbox

The Matlab routine `fminunc.m` attempts to determine the search direction d^k by solving a trust-region quadratic sub-problem on a two dimensional sub-space $\mathcal{S}_k$ of the following form :

$$(QPT2)_k \qquad \begin{aligned} & q_k(d) := \nabla g_k^\top d + \frac{1}{2} d^\top H_k d \to \min \\ & s.t. \\ & \quad d \in \Omega = \{d \in \mathbb{R}^n \mid \|Dd\| \leq \triangle_k, d \in \mathcal{S}_k\}, \end{aligned}$$

where $dim(\mathcal{S}_k) = 2$ and D is a non-singular diagonal scaling matrix.

[2] $\mathcal{N}(H_k) \bigoplus \mathcal{R}(H_k)$ represents the direct-sum of the null- and range -spaces of H_k.

The scaling matrix D is usually chosen to guarantee the **well-posedness** of the problem. For this problem the optimality condition given in Thm. 4.6.2 can be stated as

$$(H_k + \lambda_k D^\top D)d^k = -g_k.$$

At the same time, the parameter λ_k can be thought of as having a **regularizing** effect in case of **ill-posedness**. However, if we use the variable transformation $s = Dd$ we obtain that

$$\begin{aligned}(QPT2)_k \qquad & q_k(s) := \nabla g_k^\top (D^{-1}s) + \frac{1}{2}(D^{-1}s)^\top H_k (D^{-1}s) \to \min \\ & s.t. \\ & \quad d \in \Omega = \{d \in \mathbb{R}^n \mid \|s\| \le \triangle_k, D^{-1}s \in \mathcal{S}_k\},\end{aligned}$$

But this is the same as

$$\begin{aligned}(QPT2)_k \qquad & q_k(s) := \nabla \widetilde{g}_k^\top s + \frac{1}{2} s^\top \widetilde{H}_k s \to \min \\ & s.t. \\ & \quad d \in \Omega = \{d \in \mathbb{R}^n \mid \|s\| \le \triangle_k, D^{-1}s \in \mathcal{S}_k\},\end{aligned}$$

where $\widetilde{g}_k := D^{-1}g_k$ and $\widetilde{H}_k := D^{-1}H_k D_k$. The approach described in Sec. 4.6.1 can be used to solve (QPT2)$_k$ without posing any theoretical or computational difficulty. For a discussion of problem (QPT2)$_k$ with a general non-singular matrix D see Gay [3].

4.6.3 Calling and Using fminunc.m to Solve Unconstrained Problems

[xsol,fopt,exitflag,output,grad,hessian] = fminunc(fun,x0,options)

Input arguments:

fun	a Matlab function m-file that contains the function to be minimzed
$x0$	Startvector for the algorithm, if known, else [ ]
options	options are set using the optimset funciton, they determine what algorism to use,etc.

Output arguments:

$xsol$	optimal solution
fopt	optimal value of the objective function; i.e. $f(xopt)$
exitflag	tells whether the algorithm converged or not, exitflag > 0 means convergence
output	a struct for number of iterations, algorithm used and PCG iterations(when LargeScale=on)
grad	gradient vector at the optimal point $xsol$.
hessian	hessian matrix at the optimal point $xsol$.

To display the type of options that are available and can be used with the `fminunc.m` use

```
>>optimset('fminunc')
```

Hence, from the list of option parameters displayed, you can easily see that some of them have default values. However, you can adjust these values depending on the type of problem you want to solve. However, when you change the default values of some of the parameters, Matlab might adjust other parameters automatically.

As for quadprog.m there are two types of algorithms that you can use with fminunc.m

(i) **Medium-scale algorithms:** The medium-scale algorithms under fminunc.m are based on the **Quasi-Newton method**. This options used to solve problems of smaller dimensions. As usual this is set using

```
>>OldOptions=optimset('fminunc');
>>Options=optimset(OldOptions,'LargeScale','off');
```

With medium Scale algorithm you can also decide how the search direction d^k be determined by adjusting the parameter `HessUpdate` by using one of:

```
>>Options=optimset(OldOptions,'LargeScale','off','HessUpdate','bfgs');
>>Options=optimset(OldOptions,'LargeScale','off','HessUpdate','dfp');
>>Options=optimset(OldOptions,'LargeScale','off','HessUpdate','steepdesc');
```

(ii) **Large-scale algorithms:** By default the LargeScale option parameter of Matlab is always on. However, you can set it using

```
>>OldOptions=optimset('fminunc');
>>Options=optimset(OldOptions,'LargeScale','on');
```

When the 'LargeScale' is set 'on', then `fminunc.m` solves the given unconstrained problem using the **trust-region method**. Usually, the large-scale option of `fminunc` is used to solve problems with very large number of variables or with sparse hessian matrices. Such problem, for instance, might arise from discretized optimal control problems, some inverse-problems in signal processing etc.

However, to use the large-scale algorithm under `fminunc.m`, *the gradient of the objective function must be provided by the user* and the parameter 'GradObj' must be set 'on' using:

```
>>Options=optimset(OldOptions,'LargeScale','on','GradObj','on');
```

Hence, for the large-scale option, you can define your objective and gradient functions in a single function m-file :

```
function [fun,grad]=myFun(x)
fun = ...;
if nargout > 1
grad = ...;
end
```

However, if you fail to provide the gradient of the objective function, then fminunc uses the medium-scale algorithm to solve the problem.

Experiment: Write programs to solve the following problem with fminunc.m using both the medium- and large-scale options and compare the results.

$$f(x) = x_1^2 + 3x_2^2 + 5x_3^2 \to min$$
$$x \in \mathbb{R}^n.$$

Solution

Define the problem in an m-file, including the derivative in case if you want to use the LargeSclae option.

```
function [f,g]=fun1(x)
%Objective function for example (a)
%Defines an unconstrained optimization problem to be solved with fminunc

f=x(1)^2+3*x(2)^2+5*x(3)^2;

if nargout > 1
   g(1)=2*x(1);
   g(2)=6*x(2);
   g(3)=10*x(3);
end
```

Next you can write a Matlab m-file to call fminunc to solve the problem.

```
function [xopt,fopt,exitflag]=unConstEx1

options=optimset('fminunc');
options.LargeScale='off'; options.HessUpdate='bfgs';

%assuming the function is defined in the
%in the m file fun1.m we call fminunc
%with a starting point x0
x0=[1,1,1];

[xopt,fopt,exitflag]=fminunc(@fun1,x0,options);
```

If you decide to use the Large-Scale algorithm on the problem, then you need to simply change the option parameter LargeScale to on.

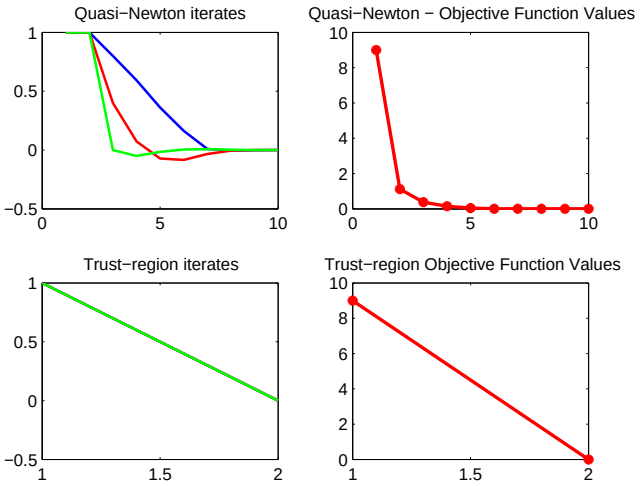
```
function [xopt,fopt,exitflag]=unConstEx1

options=optimset('fminunc');
options.LargeScale='on';
options.Gradobj='on';

%assuming the function is defined as in fun1.m
%we call fminunc with a starting point x0
x0=[1,1,1];
```

```
[xopt,fopt,exitflag]=fminunc(@fun1,x0,options);
```

To compare the medium- and large-Scale algorithms on the problem given above you can use the following m-function

```
function TestFminunc
% Set up shared variables with OUTFUN
history.x = []; history.fval = [];
%searchdir = [];

% call optimization
disp('****************************************')

disp('Iteration steps of the Matlab quasi-Newton algorithm')

i=1;x0= [1,1,1]; options =
optimset('outputfcn',@outfun,'display','iter',...
'largescale','off'); xopt = fminunc(@fun1,x0,options);

function stop = outfun(x,optimValues,state) stop = false;

switch state case 'iter'
    % Concatenate current point and objective function
    % value with history. x must be a row vector.
    history.fval = [history.fval; optimValues.fval];
    history.x = [history.x; x];
case 'done'
    m=length(history.x(:,1));
    n=length(history.fval);

    if i==1
        A=history.x;
        subplot(2,2,1);
        plot([1:m],A(:,1),'b',[1:m],A(:,2),'r',[1:m],A(:,3),'g','LineWidth',1.5);
        title('Quasi-Newton iterates');
        subplot(2,2,2);
        plot([1:n],history.fval,'*-r','LineWidth',2);
        title('Quasi-Newton - Objective Function Values');
     end
     if i==2
        A=history.x;
        subplot(2,2,3);
        plot([1:m],A(:,1),'b',[1:m],A(:,2),'r',[1:m],A(:,3),'g','LineWidth',1.5);
        title('Trust-region iterates');
        subplot(2,2,4);
        plot([1:n],history.fval,'*-r','LineWidth',2);
        title('Trust-region Objective Function Values');
    end
  end
end disp('****************************************') disp('Iteration
steps of the Matlab trust-region algorithm') i=2; history.x = [];
history.fval = []; options =
optimset('outputfcn',@outfun,'display','iter',...
'largescale','on','Gradobj','on'); xopt = fminunc(@fun1,x0,options);
end
```

If you run `TestFminunc.m` under Matlab you will get the following graphic output



which indicates that the LargeScale option of fminunc applied to the problem in `func1.m` converges after a single iterative step. (Compare also the tables that will be displayed when running `TestFminunc.m`).

Exercises Solve the following unconstrained optimization problems using both the LargeScale and MediumScale options of the Matlab function `fminunc` and compare your results. (Hint: define your functions appropriately and use `TestFminunc.m` correspondingly.)

(a) $f(x) = x_1 x_2^2 x_3^3 x_4^4 \exp(-(x_1 + x_2 + x_3 + x_4)) \rightarrow min$
s.t. $x \in \mathbb{R}^n$.
$x0 = [3, 4, 0.5, 1]$

(b) $f(x) = e^{x_1+x_2+1} - e^{-x_1-x_2-1} + e^{-x_1-1} \rightarrow \min$
s.t. $x \in \mathbb{R}^n$.
$x0 = [1, 1]$

(c) (Sine-Valley-Function)
$f(x) = 100(x_2 - sin(x_1))^2 + 0.25x_1^2 \rightarrow \min$
s.t. $x \in \mathbb{R}^n$.
$x0 = [1, 1]$

(d) (Powell-Funciton with 4 variables)
$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \rightarrow \min$
s.t. $x \in \mathbb{R}^n$.
$x0 = [3, -1, 0, 1]$

Hint: For problem (d) the trust-region method performs better. The quasi-Newton method terminates without convergence, since maximum number of function evaluations (i.e. `options.MaxFunEvals`) exceeded 400 (100*numberofvariables). Hence, increase the value of the parameter `options.MaxFunEvals` sufficiently and see the results.

4.7 Derivative free Optimization - direct (simplex) search methods

The Matlab `fminsearch.m` function uses the Nelder-Mead direct search (also called simplex search) algorithm. This method requires only function evaluations, but not derivatives. As such the method is useful when

- the derivative of the objective function is expensive to compute;
- exact first derivatives of f are difficult to compute or f has discontinuities;
- the values of f are 'noisy'.

There are many practical optimization problems which exhibit some or all of the the above difficult properties. In particular, if the objective function is a result of some experimental (sampled) data, this might usually be the case.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function. A simplex $\mathcal{S}$ in $\mathbb{R}^n$ is a polyhedral set with $n+1$ vertices $x_1, x_2, \ldots, x_{n+1} \in \mathbb{R}^n$ such that the set $\{x_k - x_i \mid k \in \{1, \ldots, n+1\} \setminus \{i\}\}$ is linearly independent in $\mathbb{R}^n$. A simplex $\mathcal{S}$ is *non-degenerate* if none of its three vertices lie on a line or if none of its four points lie on a hyperplane, etc.

Thus the Nelder-Mead simplex algorithms search the approximate minimum of a function by comparing the values of the function on the vertices of a simplex. Accordingly, $\mathcal{S}^k$ is a simplex at k-th step of the algorithm with ordered vertices $\{x_1^k, x_2^k, \ldots, x_{n+1}^k\}$ in such a way that

$$f(x_1^k) \leq f(x_2^k) \leq \ldots \leq f(x_{n+1}^k).$$

Hence, x_{n+1}^k is termed the 'worst' vertex, x_n^k the next 'worst' vertex, etc., for the minimization, while x_1^k is the 'best' vertex. Thus a new simplex $\mathcal{S}^{k+1}$ will be determined by dropping vertices which yield larger function values and including new vertices which yield reduced function values, but all the time keeping the number of vertices $n+1$ (1 plus problem dimension). This is achieved through reflection, expansion, contraction or shrinking of the simplex $\mathcal{S}^k$. In each step it is expected that $\mathcal{S}^k \neq \mathcal{S}^{k+1}$ and the the resulting simplices remain non-degenerate.

Nelder-Mead Algorithm(see Wright et. al. [6, 13] for details)

Step 0: Start with a non-degenerate simplex $\mathcal{S}^0$ with vertices $\{x_1^0, x_2^0, \ldots, x_{n+1}^0\}$. Choose the constants

$$\rho > 0,\ \chi > 1\ (\chi > \rho),\ 0 < \gamma < 1,\ \text{ and }\ 0 < \sigma < 1$$

known as reflection, expansion, contraction and shrinkage parameters, respectively.

While (1)

Step 1: Set $k \leftarrow k+1$ and label the vertices of the simplex $\mathcal{S}^k$ so that $x_1^k, x_2^k, \ldots, x_{n+1}^k$ according to

$$f(x_1^k) \leq f(x_2^k) \leq \ldots \leq f(x_{n+1}^k).$$

Step 2: Reflect

- Compute the reflection point x_r^k.

$$x_r^k = \overline{x}^k + \rho(\overline{x}^k - x_{n+1}^k) = (1+\rho)\overline{x}^k - \rho x_{n+1}^k;$$

where $\overline{x}^k$ is the centroid of the of the simplex $\mathcal{S}^k$

$$\overline{x}^k = \sum_{i=1}^{n} x_i^k, \quad \text{(here the 'worst' point } x_{n+1}^k \text{ will not be used).}$$

- Compute $f(x_r^k)$.
- **If** $f(x_1^k) \leq f(x_r^k) < f(x_n^k)$, **then** accept x_r^k and reject x_{n+1}^k and GOTO Step 1. **Otherwise** Goto **Step 3**.

Step 3: Expand

(a) **If** $f(x_r^k) < f(x_1^k)$, **then** calculate the expansion point x_e^k:

$$x_e^k = \overline{x}^k + \chi(x_r^k - \overline{x}^k) = \overline{x}^k + \rho\chi(\overline{x}^k - x_{n+1}^k) = (1+\rho\chi)\overline{x}^k - \rho\chi x_{n+1}^k.$$

- Compute $f(x_e^k)$.
  - ♢ **If** $f(x_e^k) < f(x_r^k)$, **then** accept x_e^k and reject x_{n+1}^k and GOTO Step 1.
  - ♢ **Else** accept x_r^k and reject x_{n+1}^k and GOTO Step 1.

(b) **Otherwise** Go to Step 4.

Step 4: Contract

If $f(x_r^k) \geq f(x_n^k)$, **then** perform a contraction between $\overline{x}^k$ and the better of x_{n+1}^k and x_r^k:

(a) **Outside Contraction: If** $f(x_n^k) \leq f(x_r^k) < f(x_{n+1}^k)$ (i.e. x_r^k is better than x_{n+1}^k), **then** perform outside contraction, i.e. calculate

$$x_c^k = \overline{x}^k + \gamma(x_r^k - \overline{x}_k)$$

and evaluate $f(x_c^k)$.

♢ **If** $f(x_c^k) \leq f(x_r^k)$, **then** accept x_c^k and reject x_{n+1}^k and GOTO Step 1.

♢ Otherwise GOTO Step 5. (perform shrink)

(b) **Inside Contraction:** If $f(x_r^k) \geq f(x_{n+1}^k)$ (i.e. x_{n+1}^k is better than x_r^k), then perform an inside contraction; i.e. calculate

$$x_{cc}^k = \overline{x}^k - \gamma(\overline{x}^k - x_{n+1}^k).$$

♢ **If** $f(x_{cc}^k) < f(x_{n+1}^k)$, **then** accept x_{cc}^k and reject x_{n+1}^k and GOTO Step 1.

♢ **Otherwise** GOTO Step 5. (perform shrink)

Step 5: Shrink

Define n new points

$$v_i = x_1 + \sigma(x_i^k - x_1), i = 2, \ldots, n+1$$

so that the $n+1$ points

$$x_1, v_2, \ldots, v_{n+1}$$

form the vertices of a simplex. GOTO Step 1.

END

Unfortunately, to date, there is no concrete convergence property that has been proved of the original Nelder-Mead algorithm. The algorithm might even converge to a non-stationary point of the objective function (see Mickinnon[7] for an example). However, in general, it has been tested to provide rapid reduction in function values and successful implementations of the algorithm usually terminate with bounded level sets that contain possible minimum points. Recently, there are several attempts to modify the Nelder-Mead algorithm to come up with convergent variants. Among these: the fortified-descent simplical search method (Tseng [12]) and a multidimensional search algorithm (Torczon [11]) are two of the most successful ones. See Kelley [5] for a Matlab implementation of the multidimensional search algorithm of Torczon.

Bibliography

[1] R. H. Byrd and R. B. Schnabel, Approximate solution of the trust-region problem by minimization over two-dimensional subspaces. Math. Prog. V. 40, pp. 247-263, 1988.

[2] A. Conn, A. I . M. Gould, P. L. Toint, Trust-region methods. SIAM 2000.

[3] D. M. Gay, Computing optimal locally constrained steps. SIAM J. Sci. Stat. Comput. V. 2., pp. 186 - 197, 1981.

[4] C. Geiger and C. Kanzow, Numerische Verfahren zur Lösung unrestringierte Optimierungsaufgaben. Springer-Verlag, 1999.

[5] C. T. Kelley, Iterative Methods for Optimization. SIAM, 1999.

[6] J. C. Larigas, J. A. Reeds, M. H. Wright and P. E. Wright, Convergence properties of the Nelder-Mead simplex method in low dimensions. SIAM J. Optim. V. 9, pp. 112-147.

[7] K. I. M. McKinnon, Convergence of the Nelder-Mead simplex method to a nonstationary point. SIAM Journal on Optimization. V. 9, pp. 148 - 158, 1998.

[8] J. A. Nelder and R. Mead, A simplex method for function minimization, Computer Journal, V. 7, pp. 308-313, 1965.

[9] G. A. Schultz, R. B. Schnabel and R. H. Byrd, A family of trust-region-based algorithms for uncostrained minimization with strong global convergence properties. SIAM J. Numer. Anal., V. 22, pp. 47- 67, 1985.

[10] D. C. Sorensen, Newton's method with a model trust region modification. SIAM J. Optim., V. 19, pp. 409-426, 1982.

[11] V. Torczon, On the convergence of multidimensional search algorithm. SIAM J. Optim., V. 1, pp. 123-145.

[12] P. Tseng, Fortified-descent simplical search metho. SIAM J. Optim., Vol. 10, pp. 269-288, 1999.

[13] M. H. Wright, Direct search methods: Once scorned, now respectable, in Numerical Analysis 1995 (ed. D. F. Griffiths and G. A. Watson). Longman, Harlow, pp. 191-208, 1996.

[14] W. H. Press, Numerical recipes in C : the art of scientific computing. Cambridge Univ. Press, 2002.