Spring 2015

# Parameters Estimation of Material Constitutive Models using Optimization Algorithms

Kiswendsida Jules Kere
kjk68@zips.uakron.edu

Honors Research Project

# Parameters Estimation of Material Constitutive Models using Optimization Algorithms

By

Kiswendsida Jules Kere

Date:

04/25/2015

# Table of contents

## List of figures

## List of tables

## Abstract

Optimization Algorithms are very useful for solving engineering problems. Indeed, optimization algorithms can be used to optimize engineering designs in terms of safety and economy. Understanding the proprieties of materials in engineering designs is very important in order to make designs safe. Materials are not really perfectly homogeneous and there are heterogeneous distributions in most materials. In this paper, Self-OPTIM which is an inverse constitutive parameter identification framework will be used to identify parameters of a linear elastic material constitutive model. Data for Self-OPTIM will be obtained using ABAQUS simulation of a dog-bone uniaxial test. Optimization Algorithms will be used to find parameters such as Young's modulus (E) and poison ratio ($\upsilon$).

## I. Introduction

This Honors Research Project is a project every Honors College student at the University of Akron must complete to demonstrate his skills that he has developed during his curriculum. I'm working on this project under the supervision of Dr. Gunjin Yun, Associate Professor in the department of Civil Engineering at the University of Akron. This project is about parameters estimation of material constitutive models using Self-Optimization Inverse Analysis Method (Self-OPTIM) and optimization algorithms. The concept of optimization will be explained using Newton's method and secant method.

## II. Concept of Engineering Optimization

Optimization is a set of methods (algorithms, mathematics, and modeling) used to make optimal decisions or close to the optimum. For instance, in complex problem optimization methods can be used to minimize a cost, maximize a profit, or optimize a design. Numerical optimization is used in engineering design. In fact, an optimization problem defines the objective function that is the procedure used to minimize or maximize some parameters. In addition to the objective function, some requirements called constraints may be needed to be specified. When the optimization required some constraints it is called a constraint optimization problem otherwise it is called an unconstrained optimization problem. For example, in structural design the dimensions of a beam can be found so that it minimizes the weight of the beam. Here, the objective function is to minimize the weight of the beam. The constraints can be defined in the way that the dimensions of the beam must be within a certain interval or the dimensions are linked through the design requirements.

## III. Numerical Optimization Algorithms

Actually, there are many algorithms used in numerical optimization methods. Some of these methods such as linear, quadratic, dynamic, and geometric programming algorithms have been developed to treat with particular classes of optimization problems [1]. Algorithm such as nonlinear programing has evolved for the solution of general optimization problems [1]. There are some methods of finding roots that can be used in the numerical optimization. Newton's method and secant method will be used in this project to understand the basics of numerical optimization. Newton's method and secant method are used to find the roots of a function or a system of functions. In the optimization, applying Newton's method and secant method to the derivative $f'$ of a function $f$, the roots of the derivative $f'$can be found. The roots are the stationary points or extrema of the function $f$.

## 1. Algorithms of Newton's method and secant method

### a. Newton's method

Indeed, the Newton's method is used to find the roots of a differentiable function by iteration. The following algorithm is used:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n \geq 0$$

For higher dimension (matrix) the algorithm can be written as:

$$X_{n+1} = X_n - [J_F(X_n)]^{-1} F(X_n), n \geq 0 \text{ ,} J_F \textit{ is the jacobian of } F$$

Let's solve Example 1 using the Newton's method.

**Example 1**

$$\begin{cases} x^2 + y^2 = 2 \\ x^2 - y^2 = 1 \end{cases} \quad (1)$$

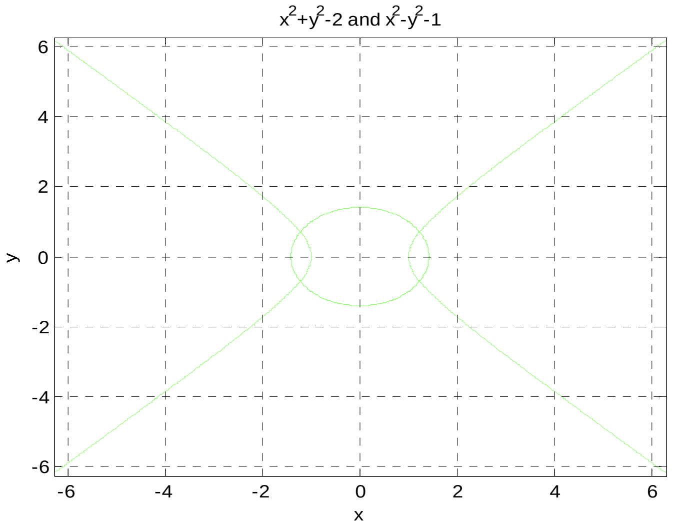

**Figure 1.** Plot of system of equations (1)

Analytically, the solutions of the system of equations (1) can be found. The following points are the solutions:

$$\left(\sqrt{\frac{3}{2}}, \quad \frac{\sqrt{2}}{2}\right), \left(\sqrt{\frac{3}{2}}, -\frac{\sqrt{2}}{2}\right), \left(-\sqrt{\frac{3}{2}}, -\frac{\sqrt{2}}{2}\right), \left(-\sqrt{\frac{3}{2}}, \quad \frac{\sqrt{2}}{2}\right)$$

The system of equations can be rewrite as follow:

$$F(X) = \begin{cases} f_1(x,y) = x^2 + y^2 - 2 = 0 \\ f_2(x,y) = x^2 - y^2 - 1 = 0 \end{cases}$$

$with\ X = (x,y)$

Using Newton's method, the solution can be determined by finding the roots of the system of functions *F(X).*

Let $X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

Using MATLAB code, the solution can be found. (See Appendix A)

$X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, the root found is $X^* = \begin{pmatrix} 1.2247 \\ 0.7071 \end{pmatrix}$ $and\ the\ number\ of\ iterations = 5$

The Newton's method has some limitations for finding roots. The method finds one root. The function may have many roots. In Example 1, there are four roots. When taking

$X_0 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$, the root found is $X^* = \begin{pmatrix} -1.2247 \\ -0.7071 \end{pmatrix}$ $and\ the\ numberof\ iterations = 5$

Depending on the initial value the algorithm will find different root. Newton's method required one initial guess and the initial guess must be close to the root to work efficiently. It takes less time and iteration to find the root. However, Newton's method required the derivative or the Jacobian and in many practical cases it is difficult to find the derivative or the Jacobian. Therefore, the secant method is more appropriate because it can approximate the derivative or the Jacobian.

### b. Secant method (Broyden's method)

The following algorithm for one dimension is used for the secant method:

$$x_{n+2} = x_n - \frac{x_n - x_{n+1}}{f(x_n) - f(x_{n+1})} f(x_n), n \geq 0$$

For higher dimension (matrix) the secant updating method also known as Broyden's method can be used. The algorithm can be written as [2]:

$$Given\ F: \mathbb{R}^n \rightarrow \mathbb{R}^n, x_0 \in \mathbb{R}^n, A_0 \in \mathbb{R}^{nxn}$$

$$Do\ for\ k = 0, 1, \ldots:$$

$$Solve\ A_k\, s_k = -F(x_k)\ \ for\ \ s_k\ , \qquad A_k = J_F(x_k)$$

$$x_{k+1} = x_k + s_k$$

$$y_k = F(x_{k+1}) - F(x_k)$$

$$A_{k+1} = A_k + \frac{(y_k - A_k\, s_k){s_k}^T}{{s_k}^T s_k}$$

Let's solve example 1 using the Broyden's method:

**Example 1**

$$\begin{cases} x^2 + y^2 = 2 \\ x^2 - y^2 = 1 \end{cases} (1)$$

$$F(X) = \begin{cases} f_1(x,y) = x^2 + y^2 - 2 = 0 \\ f_2(x,y) = x^2 - y^2 - 1 = 0 \end{cases}$$

$with\ X = (x,y)$

The initial guesses are $X_0\ \ and\ \ A_0$

$A_0 = J_F(X_0) = jacobian\ of\ F(X_0)$

$$X_0 = \begin{pmatrix} X_1 \\ X_2 \\ \dots \\ X_n \end{pmatrix},\ A_0 = J_F(X_0) = \begin{bmatrix} \frac{\partial F_1}{\partial X_1} & \cdots & \frac{\partial F_1}{\partial X_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial X_1} & \cdots & \frac{\partial F_n}{\partial X_n} \end{bmatrix}$$

$A_0 = J_F(X_0)$ is difficult to calculate in most cases, so the finite difference method can be used to approximate $J_F(X_0)$. The forward difference is used.

$$\begin{pmatrix} \frac{\partial F_1}{\partial X_1} \\ \vdots \\ \frac{\partial F_n}{\partial X_1} \end{pmatrix} = \begin{pmatrix} \frac{F_1(X_1+h)-F_1(X_1)}{h} \\ \vdots \\ \frac{F_n(X_1+h)-F_n(X_1)}{h} \end{pmatrix}, \dots \dots \begin{pmatrix} \frac{\partial F_1}{\partial X_n} \\ \vdots \\ \frac{\partial F_n}{\partial X_n} \end{pmatrix} = \begin{pmatrix} \frac{F_1(X_n+h)-F_1(X_n)}{h} \\ \vdots \\ \frac{F_n(X_n+h)-F_n(X_n)}{h} \end{pmatrix}$$

Let $X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

Using MATLAB code, an approximation of $A_0$ can be found. (See Appendix B)

$$A_0 = \begin{bmatrix} 2.000 & 2.000 \\ 2.000 & -2.000 \end{bmatrix}$$

In this example, it is easy to found $A_0$ using the formula of the Jacobian.

$$J_F(X) = \begin{bmatrix} 2\text{x} & 2\text{y} \\ 2\text{x} & -2\text{y} \end{bmatrix}$$

$A_0 = J_F(X_0) = \begin{bmatrix} 2 & 2 \\ 2 & -2 \end{bmatrix}$, which is equal to the $A_0$ approximated by the finite difference method. Therefore, using the finite difference method is a good start to approximate $A_0$.

When an approximation of $A_0$ is obtained, the solution can be found using MATLAB code. (See Appendix C)

$X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, A_0 = \begin{bmatrix} 2 & 2 \\ 2 & -2 \end{bmatrix}, the\ root\ found\ is\ X^* = \begin{pmatrix} 1.2247 \\ 0.7071 \end{pmatrix}\ and\ the\ number\ of\ iterations = 19$

Like the Newton's method, depending on the initial value the Broyden's method algorithm will find one root. Let's take another initial guess $X_0$.

$X_0 = \begin{pmatrix}-1\\-1\end{pmatrix}, A_0 = \begin{bmatrix}-2 & -2\\-2 & 2\end{bmatrix}$, the root found is $X^* = \begin{pmatrix}-1.2247\\-0.7071\end{pmatrix}$ *and the number of iteration* $= 19$

The same solution is found using both methods of Newton and Broyden. The Newton's method takes less iteration than the Broyden method. In example 1, the number of iterations for the Newton's method is 5 and that of the Broyden method is 19.

## 2. Optimization algorithms using Newton's method and Broyden-Fletcher-Goldfarb-Shanno (BFGS) secant update method.

### a. Newton's method

Newton's method in optimization is a second order method. The following algorithm is used.

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, n \geq 0$$

For n variables the algorithm can be written as [1]:

$$X^{q+1} = X^q - [H(X^q)]^{-1}\nabla F(X^q), \qquad q \geq 0, \qquad H(X^q) \text{ is the Hessian matrix of } X^q$$

$$X^q = \begin{Bmatrix} X_1 \\ X_2 \\ \cdots \\ X_n \end{Bmatrix}$$

$$\nabla F(X^q) = \begin{Bmatrix} \dfrac{\partial F(X^q)}{\partial X_1} \\ \dfrac{\partial F(X^q)}{\partial X_2} \\ \cdots\cdots \\ \dfrac{\partial F(X^q)}{\partial X_n} \end{Bmatrix}, \quad H(X^q) = \begin{bmatrix} \dfrac{\partial^2 F(X^q)}{\partial {X_1}^2} & \dfrac{\partial^2 F(X^q)}{\partial X_1 X_2} & \cdots & \dfrac{\partial^2 F(X^q)}{\partial X_1 X_n} \\ \dfrac{\partial^2 F(X^q)}{\partial X_2 X_1} & \dfrac{\partial^2 F(X^q)}{\partial {X_2}^2} & \ddots & \dfrac{\partial^2 F(X^q)}{\partial X_2 X_n} \\ \vdots & & & \vdots \\ \dfrac{\partial^2 F(X^q)}{\partial X_n X_1} & \dfrac{\partial^2 F(X^q)}{\partial X_n X_2} & \cdots & \dfrac{\partial^2 F(X^q)}{\partial {X_n}^2} \end{bmatrix}$$

Let's use the algorithm in the following example of optimization [1]:

**Example 2**

Minimize $F = X_1^2 - 3X_1X_2 + 4X_2^2 + X_1 - X_2$

$$X^0 = \begin{Bmatrix} 2 \\ 2 \end{Bmatrix}$$

Using MATLAB code, the minimum can be found. (See Appendix D)

$X^*$= (-5/7, -1/7) minimizes F with F($X^*$) = -2/7 and the number of iterations is 2.

## b. BFGS secant update method

BFGS secant update method is most successful used in unconstrained nonlinear optimization problems. The following algorithm is used [3]:

$x_{k+1} = x_k + \alpha_k \mathrm{d}_k$ where $\alpha_k$ is the search parameter

$\mathrm{d}_k = -H_k g_k$

$p_k = \alpha_k \mathrm{d}_k = x_{k+1} - x_k$

$g_k = \nabla f(x_k), \quad g_{k+1} = \nabla f(x_{k+1})$

$q_k = g_{k+1} - g_k$

$$H_{k+1} = \left( H_k - \frac{H_k q_k q_k^T H_k}{q_k^T H_k q_k} + \phi \mathrm{v}_k \mathrm{v}_k{}^T \right) * \frac{p_k^T q_k}{q_k^T H_k q_k} + \frac{p_k p_k^T}{p_k^T q_k}$$

ϕ is a parameter that generally permits to vary from one iteration to another.

$$\mathrm{v}_k = (q_k{}^T H_k q_k)^{\frac{1}{2}} \left( \frac{p_k}{p_k{}^T q_k} - \frac{H_k q_k}{q_k{}^T H_k q_k} \right)$$

Let's use the algorithm to solve example 3.

**Example 3**

Minimize $f(x_1, x_2) = (x_1 - 2)^4 + (x_1 - 2)^2 x_2^2 + (x_2 + 1)^2$ [2]

$f \text{ has the minimizer } X_* = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$

Let's start with $X_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. In practice, it is common to start with $H_0 = I$

$H_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Using MATLAB code, the minimizer can be determined. (See Appendix E)

$X_* = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$ is the minimizer of f and the number of iterations is 19. It takes more time and iterations to get the solution. Sometimes it is difficult to get the gradient of the objective function. So, the finite difference method can be used to approximate the gradient of the objective function.

## IV. Practical Optimization Problem: Self-OPTIM

In this project, the Self-Optimizing Inverse Method (Self-OPTIM) will be used for practical optimization problem. In fact, Self-OPTIM is an inverse constitutive parameter identification framework developed by Yun et al. Self-OPTIM is capable of identifying parameters of nonlinear material constitutive models and was demonstrated only by numerically simulated testing with full- field displacement fields and prescribed boundary loadings [4]. The Self-OPTIM is capable of identifying parameters of the chosen class of material constitutive models through minimization of an implicit objective function defined as a function of full-field stress and strain fields in the optimization process [4]. Using optimization algorithms, parameters of material constitutive models can be identified. In this project, the objective function is defined as a function of full-field stress and strain fields of a material constitutive model. Self-OPTIM data are obtained through ABAQUS simulation of an isotropic dog-bone uniaxial test.  DIC data are used to improve the data of the test. Digital Image Correlation (DIC) is a 3D, full-field, non-contact optical technique to measure contour, deformation vibration and strain on almost any material [5]. It can be used for many tests including tensile, torsion, bending and combined loading for both static and dynamics applications [5]. The test simulation will be done without and with DIC data. Here the objective function is an unconstrained nonlinear several variable function and has 2 variables; Young's modulus (E) and poison ratio (υ). E and υ are the parameters of the material constitutive model to be identified. Finding the minimum of the objective function will give the parameters of the material constitutive model or parameters close to the true values depending on the efficiency of the optimization algorithm used. Figure 2 and Figure 3 display the surface of the objective function without DIC data and with DIC data respectively. The surface of the objective function is obtained using MALAB code. (See Appendix F)

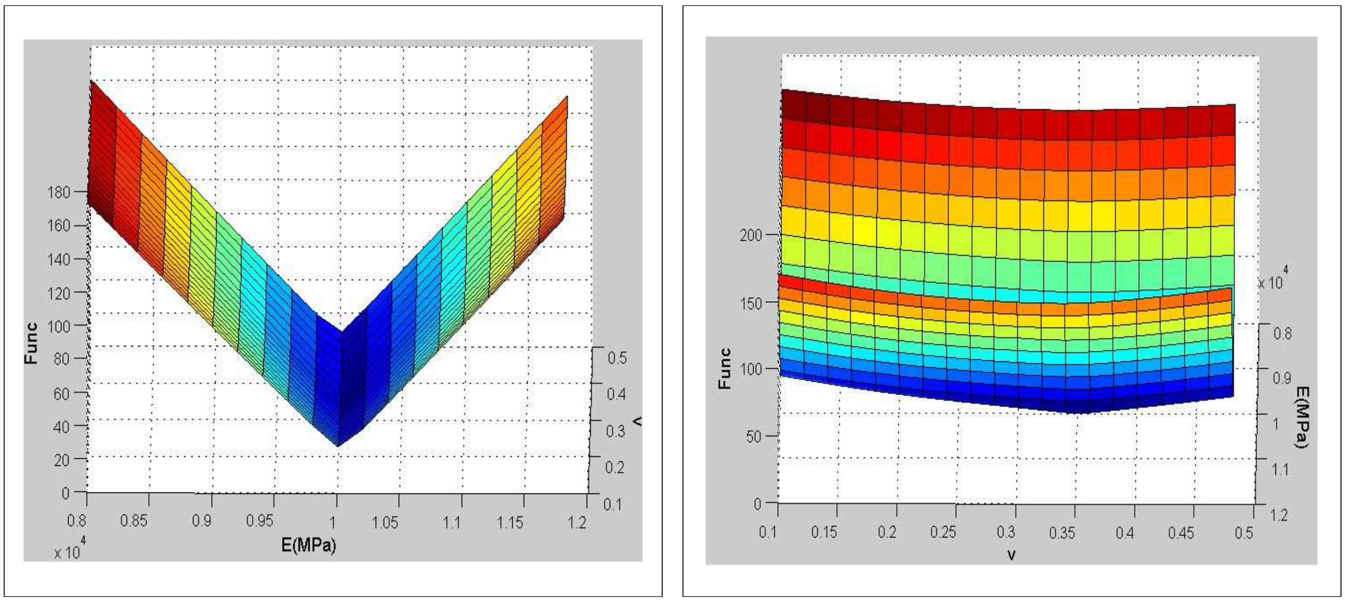

**Figure 2.** Surface of the objective function without DIC data

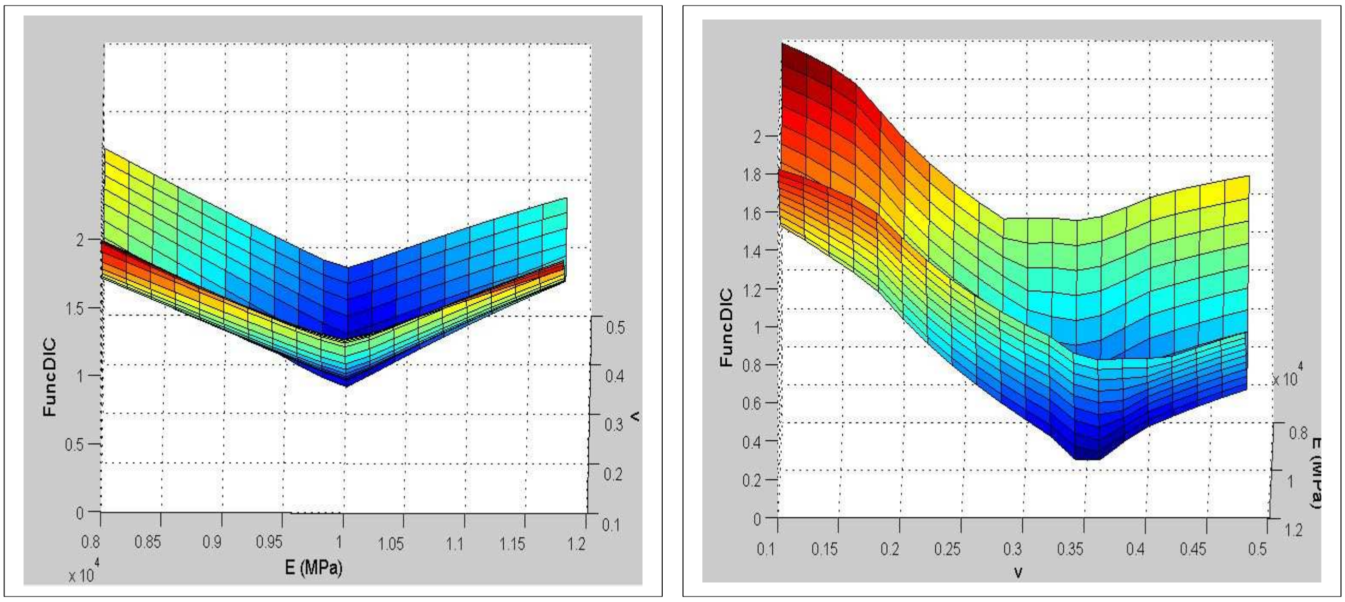

**Figure 3.** Surface of the objective function with DIC data

Actually, the true values of the parameters are E =10000 MPa and υ=0.35. The minimum of the objective function is 0. The optimization algorithms used above to solve simple mathematical optimization problems were not capable of finding the parameters that minimize the objective function. For this reason, MATLAB optimization toolbox with solvers "fmincon" and "fminsearch" were used. The first solver 'fmincon' attempts to find a constrained minimum of a scalar function of several variables starting at an initial estimate [6]. Also, Fmincon provides 4 options for optimization algorithm; 'interior-point', sqp', 'active-set', and 'trust-region-reflective'. In this paper, 'interior-point' with options bfgs, central difference, TolX =1e-20, TolCon=1e-20, TolFun=1e-20, and TypicalX= [8000, 0.2] were used. The second solver 'fminsearch' finds the minimum of a scalar function of several variables, starting at an initial estimate by using Nelder-Mead simplex algorithm [7]. The method does not call for any information about the gradients or Hessian, which makes it suitable for problems with either non-smooth function or rapidly changed Hessian [8].

**Table 1.** Results obtained without DIC data and DIC data.

| | MatLab built-in function | Algorithms | Xoptim | Fvalue | Iteration |
|---|---|---|---|---|---|
| Without DIC | fmincon | "Interior-point":bfgs | E = 10000.367 MPa | 0.00335 | 63 |
| | | | v = 0.348 | | |
| | fminsearch | Simplex method | E =10000.23 MPa | 0.000102 | 35 |
| | | | v =0.35001 | | |
| With DIC | fmincon | "Interior-point":bfgs | E = 10001 MPa | 0.0472 | 44 |
| | | | v = 0.341 | | |
| | fminsearch | Simplex method | E =10000 MPa | 0.00000123 | 61 |
| | | | v =0.35 | | |

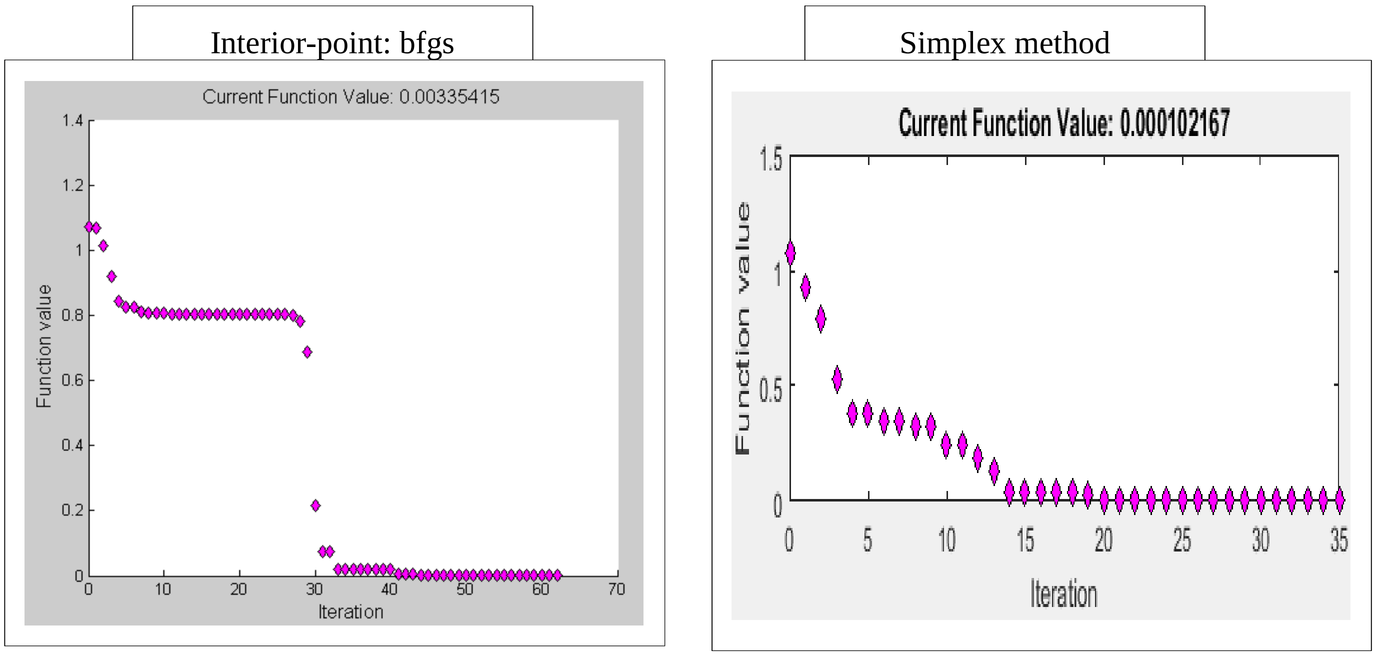

**Figure 4.** Convergence of objective function without DIC data

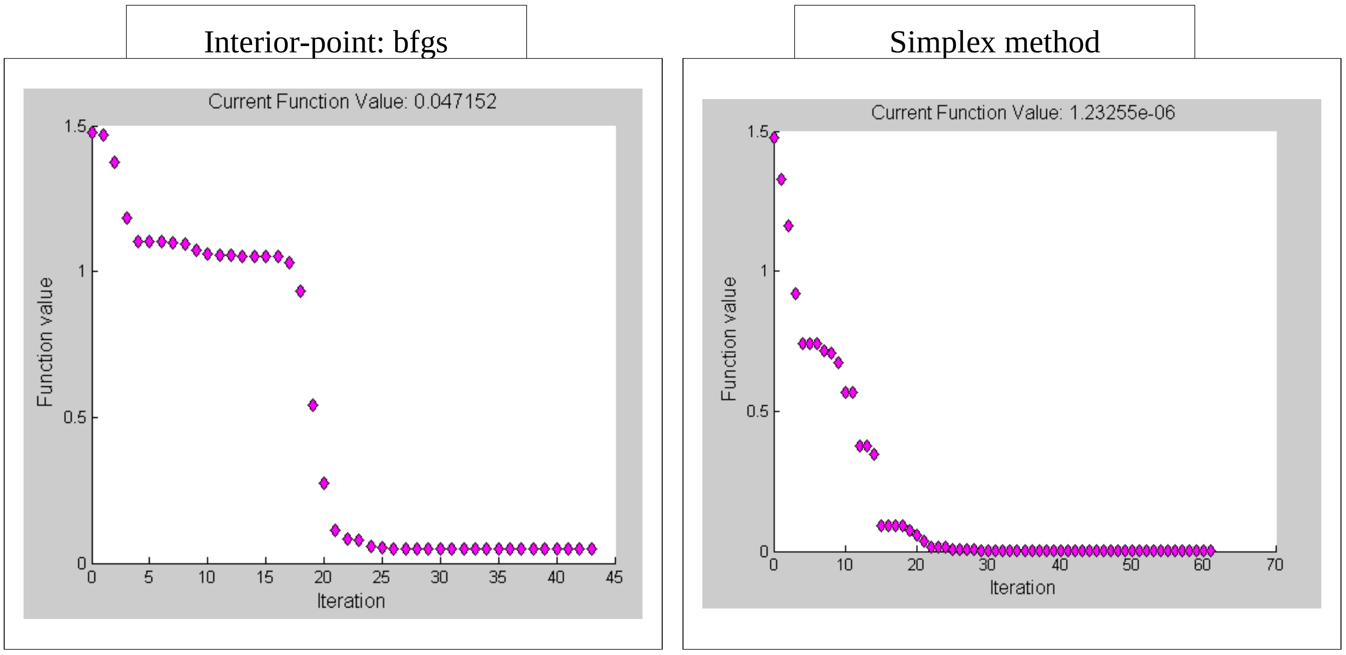

**Figure 5.** Convergence of objective function with DIC data

Table 1 shows the results obtained using interior-point algorithm (bfgs) and simplex method. The optimization algorithms were run without DIC data and with DIC data. The results show that without DIC data the interior-point algorithm is more accurate than the one that runs with DIC data; but it requires less iteration with DIC data. For the Simplex method, the results are more accurate with DIC data; but it requires more iterations. The simplex method is better than the interior-point in both cases (without DIC data and with DIC data).
Figure 4 and Figure 5 show the convergence of the objective function without DIC data and with DIC data respectively. The simplex method converges faster to the solution than the interior-point algorithm.

## V. Conclusion

In this paper, basic knowledge of numerical optimization algorithms was discussed using simple mathematical problems. Practical optimization problem such as Self-OPTIM was also used to emphasize the importance of optimization in engineering. The results from Self-OPTIM showed that optimization is very useful in engineering. Being able to identify parameters of material constitutive models can improve engineering designs. Although in this paper the data used are from test simulation, other articles have showed that Self-OPTIM works with more realistic experimental data.

## Appendix A. MATLAB code for example 1 using Newton’s method

```
% Newton's method to find a solution of a system of equations
% f is a system of symbolic functions
% X0 is the initial value
% tol is the tolerance
% maxiter is the max of iteration allowed
% c is a root of f
% iter is the number of iteration to get the root c

syms x y
f = [x^2+y^2-2;x^2-y^2-1];
X0 = [1;1];
maxiter = 200;
tol = [1e-10;1e-10];
Fxy = matlabFunction(f);
J = matlabFunction(jacobian(f, [x, y]));

Xn=X0;
for n = 1:maxiter
    Xnplus1 = Xn - inv(J(Xn(1), Xn(2)))*Fxy(Xn(1), Xn(2));
    abserr = abs (Xnplus1 -Xn);
    relerr = abserr./(abs(Xn)+ eps);

    a1 = all(abserr<tol);
    a2 = all(relerr<tol);

    if a1==1 && a2==1
        c = Xnplus1;
        iter = n;
        return;
    end
   Xn = Xnplus1;
end
```

## Appendix B. MATLAB code for approximation of $A_0$

```
% Finite method to approximate jacobian
% h=step for finite method
% A0= jacobian of f at x0
syms x y
f = [x^2+y^2-2;x^2-y^2-1];
x0 = [1;1];
h =  1e-10;
A = (subs(f, [x],[x0(1)+h])-subs(f, [x],[x0(1)]))/h;
B = (subs(f, [y],[x0(2)+h])-subs(f, [y],[x0(2)]))/h;
A0 = double([A,B]);
```

## Appendix C. MATLAB code for example 1 using secant method

```
% Boyden's method (secant method)to find a solution of a system of
equations
% f is a system of symbolic functions
% x0 is the initial value
% A0 is the initial guess for the jacobian of f at x0
% tol is the tolerance
% maxiter is the max of iteration allowed

% c is a root of f
% iter is the number of iteration to get the root c
syms x y
f = [x^2+y^2-2;x^2-y^2-1];
x0 = [1;1];
A0 = [2,2;2,-2];
tol = [1e-10;1e-10];
maxiter = 200;
Fxy = matlabFunction(f);
Ak= A0;
xk=x0;

for k = 1:maxiter
sk = -1 * inv(Ak)*Fxy(xk(1), xk(2));
xkplus1 = xk + sk;
yk = Fxy(xkplus1(1),xkplus1(2)) - Fxy(xk(1),xk(2));
Akplus1 = Ak + sk'*(yk-Ak*sk)/(sk'*sk);

abserr = abs (xkplus1 -xk);
relerr = abserr./(abs(xk)+ eps);

a1 = all(abserr<tol);
a2 = all(relerr<tol);

if a1==1 && a2==1
    c = xkplus1;
    iter= k;
    return;
end
xk = xkplus1;
Ak = Akplus1;
end
```

## Appendix D. MATLAB code for example 2 using Newton’s method

```
% optimization of a objective function using Newton’s method
syms X1 X2;
F=X1^2-3*X1*X2+4*X2^2+X1-X2;
X0=sym([2;2]);    % X0 is the initial value
tol=[1e-10;1e-10];   % tol is the tolerance
maxiter=200;   % maxiter is the max of iteration allowed
H=hessian(F);
G=gradient(F);
Xn=X0;
for n = 1:maxiter
    Xnplus1 = Xn-
inv(subs(H,[X1,X2],[Xn(1),Xn(2)]))*subs(G,[X1,X2],[Xn(1),Xn(2)]);
    abserr = abs (Xnplus1 -Xn);
    relerr = abserr./(abs(Xn)+ eps);
    if abserr<tol & relerr<tol;
       c = Xnplus1;   % c is the minimizer of F
       iter = n;    % iter is the number of iterations

        return;
    end
   Xn = Xnplus1;
end
```

## Appendix E. MATLAB code for example 3 using BFGS secant update method

```
clear all
fclose('all');
syms X1 X2;
f=(X1-2)^4+(X1-2)^2*X2^2+(X2+1)^2;
X0=[1;1];% X0 is the initial value
tol=1e-6; % tol is the tolerance
h=[1e-10;1e-10]; % step used  for the finite differnce method
maxiter=200; % maxiter is the max of iteration allowed
H0=eye(2); % H0 is the initial value of the hessian of f at x0
Xk=X0;
Hk=H0;
F=matlabFunction(f);
phi=1.0;
alpha=1.0;  % search parameter

for k = 1:maxiter

    %GXK is an approximation of gradient of F(Xk)using finite difference
     GXk=[(F(Xk(1)+h(1),Xk(2))-F(Xk(1),Xk(2)))/h(1);
        (F(Xk(1),Xk(2)+h(2))-F(Xk(1),Xk(2)))/h(2)];
    Xkp = Xk-Hk*GXk;

    %GXKplus1 is an approximation of gradient of F(Xkp) using finite
difference
     GXkplus1=[(F(Xkp(1)+h(1),Xkp(2))-F(Xkp(1),Xkp(2)))/h(1);
       (F(Xkp(1),Xkp(2)+h(2))-F(Xkp(1),Xkp(2)))/h(2)];
    qk = GXkplus1-GXk;
    pk = Xkp-Xk;

    if mod(k,size(Hk,1))==0
        Hkplus1=H0;
    else
        vk = sqrt(qk'*Hk*qk).*(pk./(pk'*qk)-(Hk*qk)./(qk'*Hk*qk));
        Hkplus1 = (Hk-
(Hk*(qk)*qk'*Hk)./(qk'*Hk*qk)+phi.*vk*vk').*(pk'*qk/(qk'*Hk*qk))+(pk*pk')./
(pk'*qk);
    end
    abserr = abs (Xkp-Xk);
    relerr = abserr./(abs(Xk)+ eps);
    if max(abserr)<tol && max(relerr)<tol;
        c = Xkp; % c is the minimizer of F
        iter = k; % iter is the number of iterations
        return;
    end
    Xk = Xkp;
    Hk=Hkplus1;
end;
```

## Appendix F. MATLAB code for surface of objective function

```
% visualization of self-optim objective function surface
% isotropic materials

% Func: objective function without DIC data
% FuncDIC: objective function with DIC data

clear all;
fclose('all');
E=8000:200:11900;
v=0.1:0.02:0.49;
for i=1:size(E,2)
    for j=1:size(E,2)
       g(i,j) = Func(E(1,i),v(1,j));
      %g(i,j) = FuncDIC(E(1,i),v(1,j));

    end
end
[E,v]=meshgrid(E,v);
surf(E,v,g');
```

## References

[1] Vanderplaats, Garret N. (1984). Numerical optimization techniques for engineering design: with application. McGraw-Hill, Inc.

[2] Dennis, JR.J.E. (1983). Numerical methods for unconstrained optimization and nonlinear equations. New Jersey: Prentice-Hall, Inc., Englewood Cliffs.

[3] Luenberger, David G. (1984). Linear and nonlinear programming. Addison-Wesley Publishing Company, Inc.

[4] Yun, G.J.; Shang, S., (2012): Identification of Elasto-Plastic Constitutive Parameters by Self-Optimizing Inverse Method: Experimental Verifications. *Computers, Materials & Continua*, Vol. 27, No. 1, pp. 55-72.

[5] Dantecdynamics. (2013). Digital Image Correlation-DIC. Retrieve April 14, 2015, from http://www.dantecdynamics.com/digital-image-correlation

[6] TheMathWorks, Inc. (2015). Fmincon. Retrieve April 14, 2015, from http://www.mathworks.com/help/optim/ug/fmincon.html

[7] TheMathWorks, Inc. (2015). Fminsearch. Retrieve April 14, 2015, from http://www.mathworks.com/help/matlab/ref/fminsearch.html

[8] Yun, G.J.; Shang, S., (2011). A Self-Optimizing Inverse Analysis Method for Estimation of Cyclic Elasto-Plasticity Model Parameters. *International Journal of Plasticity* 27, 576-595.