Fit a stress strain curve with 8 parameters

An arbitrary stress strain curve is fitted. The number of required objective function evaluations is compared for the NNO algorithm and for the conventional least squares algorithm (lsqnonlin).

Contents

Define input data for the NNO algorithm

Name of residual function

```
objFun='func';
```

Number of design variables

```
nVar=8;
```

Lower and upper bound vectors

```
lb=0*ones(nVar,1);
ub=1*ones(nVar,1);
```

Number of Abaqus analyses for initial training of the neural network

```
initSim=5;
```

Number and size of hidden layers

```
hiddenSizes = 15; % row vector
```

Population size

```
Psize=10;
```

Termination tolerance of error between target and simulated curve

```
funTol=0.0005;
```

Maximum number of iterations

```
maxSim=60;
```

Stall tolerance for X

```
XTol=0.001;
```

Stall tolerance for Y

```
YTol=0.001;
```

Set rng for repeatability

```
rng(0)
```

Solution with the Neural Network Optimization algorithm

Apply the NNO function

```
[xSim,ySim,errSim,errANN,ind,nEval1,exitFlag] = ...
    NNO(objFun,nVar,lb,ub,... % optimization properties
    initSim,hiddenSizes,Psize,... % ANN/GA properties
    funTol,maxSim,XTol,YTol); % termination properties
```

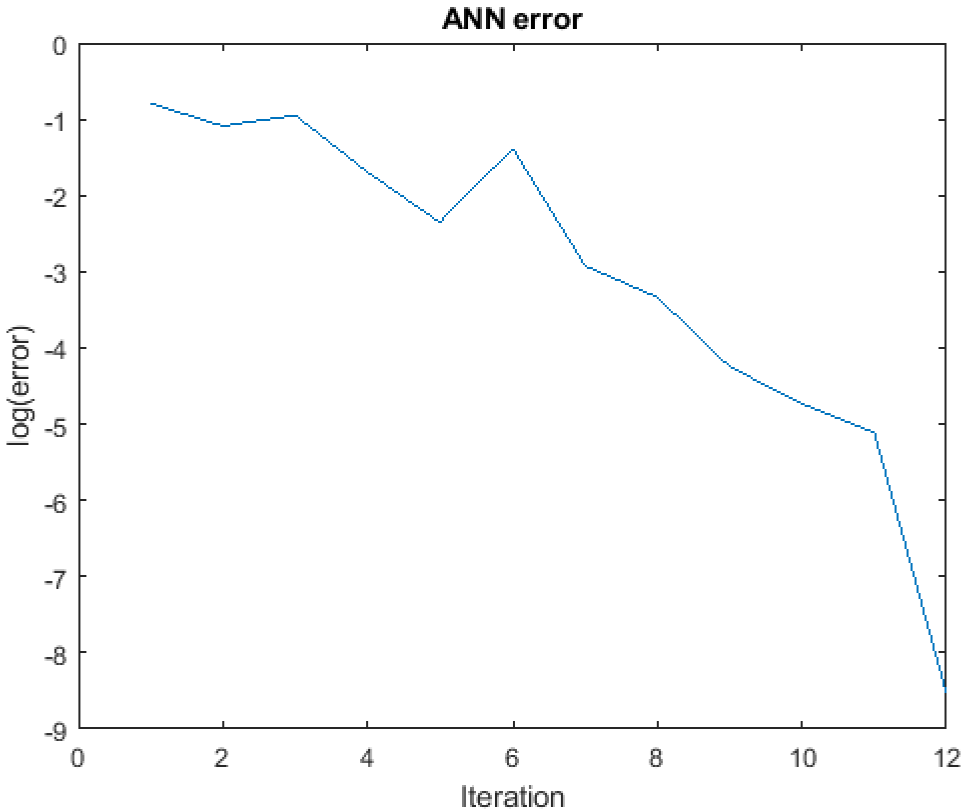
Output of the Neural Network Optimization algorithm

Check the evolution of the OBJFUN error

```
figure(1)
plot(log(errSim))
xlabel('Iteration')
ylabel('log(error)')
title('OBJFUN error')
```

Check the evolution of the optimum point of the dummy ANN function

```
figure(2)
plot(log(errANN(initSim+1:end)))
xlabel('Iteration')
ylabel('log(error)')
title('ANN error')
```



Print the optimum values of the design variables

```
xSim(:,ind(1))
```

```
ans =

    0.7868
    0.4763
    0.1964
    0.4985
    0.9068
    0.4416
    0.6664
    0.4669
```

Compare the target curve and the optimum curve

x coordinates of target curve

```
xI=(0.01:0.01:0.15)';
```

y coordinates of target curve

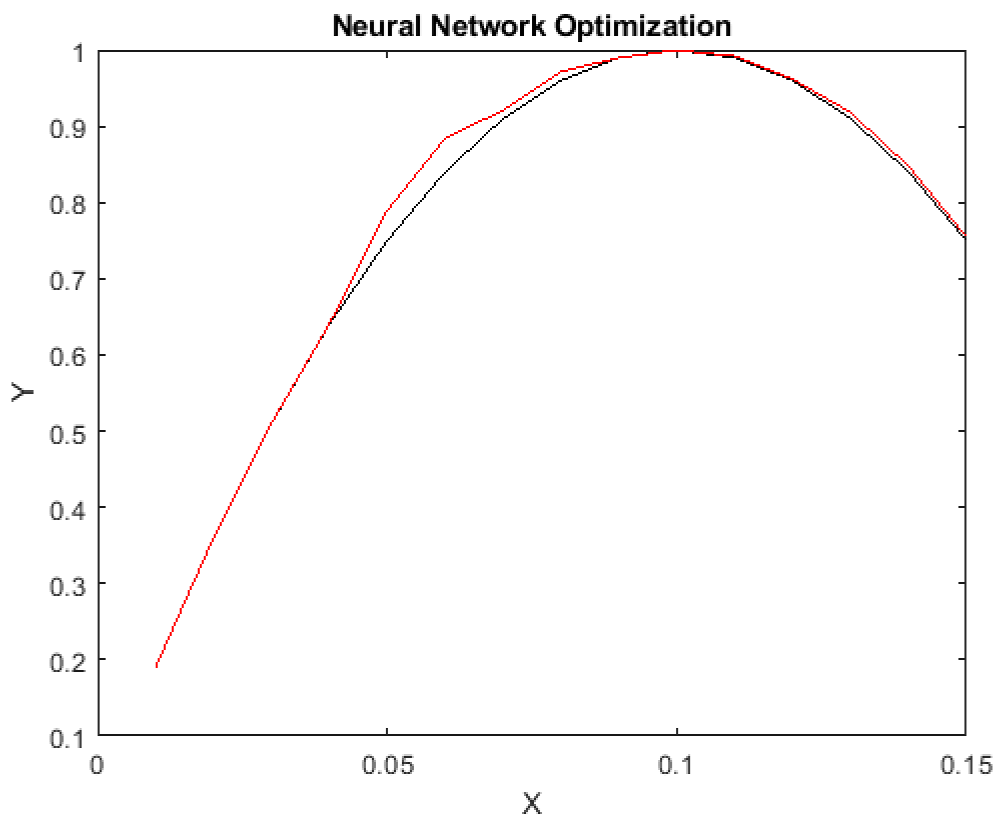
```
yI=1-100*(xI-0.1).^2;
```

Optimum curve based on the optimum values of the design variables

```
yOpt1 = func(xSim(:,ind(1)));
yOpt1=yOpt1.*yI+yI;
```

Plot

```
figure(3)
plot(xI,yI,'Color','black')
hold on
plot(xI,yOpt1,'Color','red')
hold off
title('Neural Network Optimization')
xlabel('X')
ylabel('Y')
```



Solution with the lsqnonlin function

Apply the lsqnonlin function

```
x0=lb+rand(8,1).*(ub-lb);
options=optimset('lsqnonlin');
options.TolFun=0.02;
[x,resnorm,residual,exitflag,output] = lsqnonlin(objFun,x0,lb,ub,options);
```

```
Local minimum found.
```

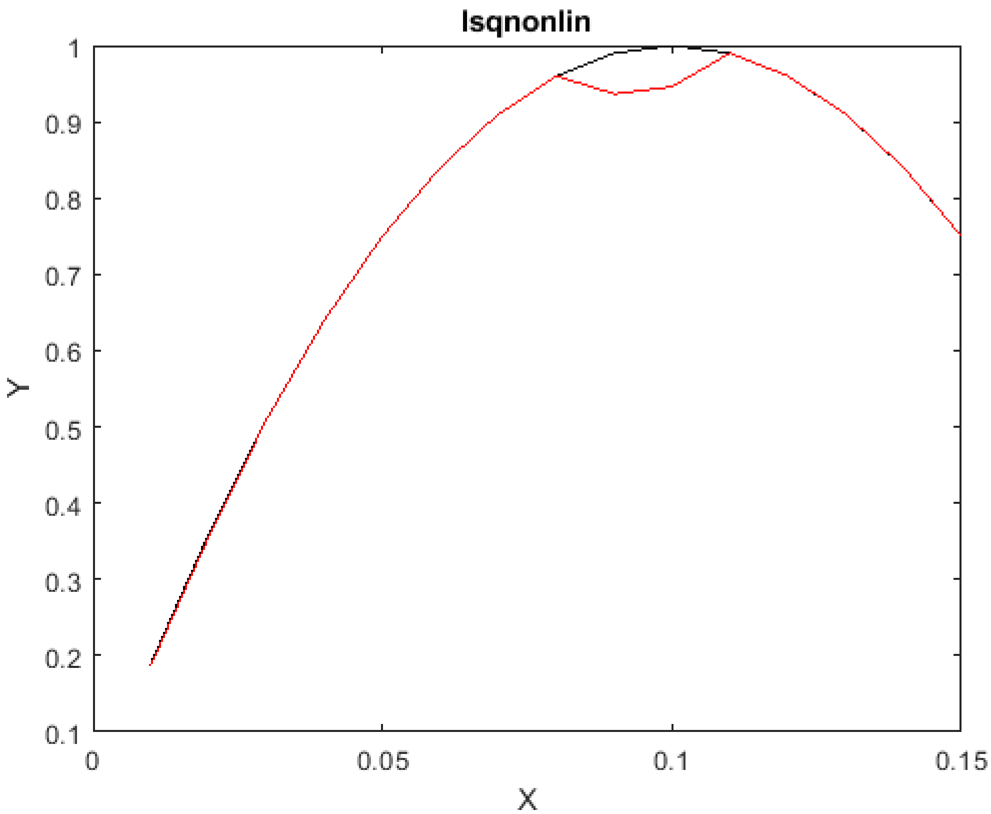
```
Optimization completed because the size of the gradient is less than
the selected value of the optimality tolerance.
```

Optimum curve based on the optimum values of the design variables

```
yOpt2 = func(x);
yOpt2=yOpt2.*yI+yI;
```

Plot

```
figure(4)
plot(xI,yI,'Color','black')
hold on
plot(xI,yOpt2,'Color','red')
hold off
title('lsqnonlin')
xlabel('X')
ylabel('Y')
```



Compare number of objective function evaluations

For the proposed Neural Network Optimization algorithm:

```
nEval1
```

```
nEval1 =

    17
```

For the conventional lsqnonlin optimization algorithm:

```
nEval2=output.funcCount
```

```
nEval2 =

    36
```

Published with MATLAB® R2017b

NNO

Documentation of the NNO function.

```
helpFun('NNO')
```

```
Neural Network Optimization (NNO)

Syntax
    [XSIM,YSIM,ERRSIM,ERRANN,IND,NEVAL,EXITFLAG] = ...
        NNO(OBJFUN,NVAR,LB,UB,...
        INITSIM,hiddenSizes,PSIZE,...
        FUNTOL,MAXSIM,XTOL,YTOL)

Description
    Apply the Neural Network Optimization (NNO) algorithm to solve
    nonlinear least-squares (nonlinear data-fitting) problems. The NNO
    algorithm uses an Artificial Neural Network (ANN) coupled with a
    Genetic Algorithm (GA) towards minimizing the sum of squares of a
    vector-valued objective function. The ANN is used as a dummy internal
    objective function equivalent to OBJFUN. The GA algorithm is used for
    minimizing the ANN. The optimum solution of the ANN given by the GA
    will be the optimum solution of OBJFUN, since the ANN and the OBJFUN
    are equivalent.
    The optimization procedure goes as follows:
    (1) An initial set of training data is produced based on OBJFUN
    (2) The ANN is trained based on the above data set.
    (3) The ANN is used as an objective function in GA and is minimized.
    (4) OBJFUN is evaluated at the optimum solution that is found by GA.
    (5) This extra data is added at the initial set of training data,
        thus extending the data by one additional OBJFUN function
        evaluation.
    (6) Replace the initial training data with the extended training data
    (7) Continue with step (2) above

Input arguments
    OBJFUN [char(1 x :inf)] is the name of the objective function whose
        sum of squares is minimized. See the file func.m for details
        about its syntax and a coding example.
    NVAR [double(1 x 1)] is the number of design variables.
    LB [double(:inf x 1)] is a vector containing the lower bounds of the
        design variables
    UB [double(:inf x 1)] is a vector containing the upper bounds of the
        design variables
    INITSIM [double(1 x 1)] is the number of the initial evaluations of
        OBJFUN before the first training of the ANN.
    HIDDENSIZES [double(1 x :inf)] is the size of the hidden layers
        in the ANN, specified as a row vector. The length of the vector
        determines the number of hidden layers in the ANN.
    PSIZE [double(1 x 1)] is the size of the population used by the GA.
    FUNTOL [double(1 x 1)] is the termination tolerance of the objective
        function. If the sum of squares of OBJFUN becomes lower than
        FUNTOL, optimum solution is considered to have been reached and
        the optimization algorithm is terminated.
    MAXSIM [double(1 x 1)] is the number of maximum OBJFUN function
        evaluations (NEVAL). If NEVAL>MAXSIM, the optimization algorithm
```

```
        is terminated.
    XTOL [double(1 x 1)] is the tolerance for the change in the design
        variables (X). If norm((X(N+1)-X(N))./(abs(X(N+1))+abs(X(N)))) <
        XTOL, the optimization algorithm is terminated.
    YTOL [double(1 x 1)] is the tolerance for the change in the output of
        OBJFUN. If norm((Y(N+1)-Y(N))./(abs(Y(N+1))+abs(Y(N)))) < YTOL,
        the optimization algorithm is terminated.

Output arguments
    XSIM [double(NVAR x MAXSIM+1)] contains the values of the design
        variables that are used throughout the whole optimization
        history. The optimum values of the NNO are equal to
        XSIM(:,IND(1)).
    YSIM [double(:inf x MAXSIM+1)] contains the values of OBJFUN that are
        used throughout the whole optimization history. The optimized
        output of OBJFUN is equal to YSIM(:,IND(1)).
    ERRSIM [double(1 x MAXSIM+1)] contains the error which is equal to
        the sum of squares of OBJFUN throughout the whole optimization
        history. The optimized error of OBJFUN is equal to
        ERRSIM(IND(1)).
    ERRANN [double(1 x MAXSIM+1)] contains the error which is equal to
        the sum of squares of the output of the ANN, used as a dummy
        objective function equivalent to OBJFUN, throughout the whole
        optimization history. The optimized error of ANN is equal to
        ERRANN(IND(1)).
    IND [double(1 x 1)] is the position of the optimum in the
        optimization history.
    NEVAL [double(1 x 1)] is the number of OBJFUN function evaluations.
    EXITFLAG [double(1 x 1)] is an integer, showing the reason the solver
        stopped. It can take the following values (compatible with the
        exitflag output of the Matlab function LSQNONLIN):
        EXITFLAG=1: Function converged to a solution
        EXITFLAG=2: Change in X is less than the specified tolerance TOLX
        EXITFLAG=3: Change in Y is less than the specified tolerance TOLY
        EXITFLAG=0: Number of function evaluations exceeded MAXSIM
        EXITFLAG=-1: An error in OBJFUN stopped the solver.

Example
    objFun='func';
    nVar=8;
    lb=0*ones(nVar,1);
    ub=1*ones(nVar,1);
    initSim=5;
    hiddenSizes = 15; % row vector
    Psize=10;
    funTol=0.001;
    maxSim=60;
    XTol=0.001;
    YTol=0.001;
    [xSim,ySim,errSim,errANN,ind,nEval,exitFlag] = ...
        NNO(objFun,nVar,lb,ub,... % optimization properties
        initSim,hiddenSizes,Psize,... % ANN/GA properties
        funTol,maxSim,XTol,YTol); % termination properties
    % Output of the Neural Network Optimization algorithm
    % Evolution of the OBJFUN error
    figure(1)
    plot(log(errSim))
    xlabel('Iteration')
    ylabel('log(error)')
    title('OBJFUN error')
    % Evolution of the optimum point of the dummy ANN function
```

```
figure(2)
plot(log(errANN(initSim+1:end)))
xlabel('Iteration')
ylabel('log(error)')
title('ANN error')
% Print the optimum values of the design variables
xSim(:,ind(1))
% Compare the target curve and the optimum curve
% x coordinates of target curve
xI=(0.01:0.01:0.15)';
% y coordinates of target curve
yI=1-100*(xI-0.1).^2;
% Optimum curve based on the optimum values of the design variables
yOpt = func(xSim(:,ind(1)));
yOpt=sqrt(yOpt).*yI+yI;
% Plot
figure(3)
plot(xI,yI,'Color','black')
hold on
plot(xI,yOpt,'Color','red')
hold off
```

func

Documentation of the func function.

```
helpFun('func')
```

```
Objective function whose sum of squares is minimized

Syntax
    Y = FUNC(X)

Input arguments
    X [double(:inf x 1)] is a vector containing the values of the
        design variables

Output arguments
    Y [double(:inf x 1)] is a vector containing the values of the
        function at X.

Example
    % Evaluate the relative error with respect to the (xI-yI) stress
    % strain curve for a random combination of design variables
    x = rand(8,1);
    y = func(x);

Copyright (c) 2021 by George Papazafeiropoulos
```

crossoverFun

Documentation of the crossoverFun function.

```
helpFun('crossoverFun')
```

```
 Weighted average crossover

 Syntax
     XOVERKIDS = crossoverFun(PARENTS,OPTIONS,GENOMELENGTH, ...
         FITNESSFCN,UNUSED,THISPOPULATION,RATIO)

 Description
     Create the crossover children XOVERKIDS of the given population
     THISPOPULATION using the available PARENTS. Depending on the value of
     the variable RATIO, children are generated on the line between the
     parents (if RATIO is scalar) or children are generated within the
     hypercube with the parents at opposite corners (if RATIO is vector
     with size [1 x GENOMELENGTH]).

 Input arguments
     PARENTS [double(1 x :inf)] is the vector of parents chosen by the
         selection function.
     OPTIONS [struct(1 x 1)] is a structure containing the ga options,
         given by the command >>OPTIONS = optimoptions('ga').
     GENOMELENGTH [double(1 x 1)] is the number of the design variables
     THISPOPULATION [double(PSIZE x GENOMELENGTH)] contains the
         individuals in the current population.
     RATIO [double(1 x 1)] is the weight applied for the weighted average
         of the parents.

 Output arguments
     XOVERKIDS [double(length(PARENTS)/2 x GENOMELENGTH)] is the
         offspring that results from the crossover operation.

 Example
     % Create an options structure using crossoverFun as the crossover
     % function with default ratio = ones(1,GenomeLength)
     options = optimoptions('ga', 'CrossoverFcn', @crossoverFun);
     % Create an options structure using crossoverFun as the crossover
     % function with RATIO of 0.5
     ratio = 0.5
     options = optimoptions('ga', 'CrossoverFcn', {@crossoverFun, ratio});

 Copyright (c) 2021 by George Papazafeiropoulos
 __________________________________________________________________________
```

mutationFun

Documentation of the mutationFun function.

```
helpFun('mutationFun')
```

```
Uniform multi-point mutation

Syntax
    MUTATIONCHILDREN = MUTATIONFUN(PARENTS,OPTIONS,GENOMELENGTH,...
        FITNESSFCN,STATE,THISSCORE,THISPOPULATION, ...
        MUTATIONRATE,MUTATIONSCALE)

Description
    Create mutated children using uniform mutations at multiple points.
    Mutated genes are uniformly distributed over the range of the gene.
    The new value is NOT a function of the parents value for the gene.

Input arguments
    PARENTS [double(1 x :inf)] is the vector of parents chosen by the
        selection function.
    OPTIONS [struct(1 x 1)] is a structure containing the ga options,
        given by the command >>OPTIONS = optimoptions('ga').
    GENOMELENGTH [double(1 x 1)] is the number of the design variables
    FITNESSFCN [function handle] is the fitness function. It can be
        called by the command >>Y=FitnessFcn(X), where X [double(N x
        GENOMELENGTH)] is an array containing N individuals, each
        containing GENOMELENGTH values of the design variables. Y
        [double(1 x N)] contains the fitness values of the population X.
    STATE [struct(1 x 1)] is a structure Structure containing information
        about the current generation.
    THISSCORE [double(PSIZE x 1)] contains the scores of the current
        population. PSIZE [double(1 x 1)] is the population size.
    THISPOPULATION [double(PSIZE x GENOMELENGTH)] contains the
        individuals in the current population.
    MUTATIONRATE [double(1 x 1)] is the mutation rate. Each entry of an
        individual has a probability rate of being mutated equal to
        MUTATIONRATE.
    MUTATIONSCALE [double(1 x 1)] is the mutation scale. If an entry of
        an individual is being mutated, the new value is given by
        >>m=lb+MUTATIONSCALE*rand*(ub-lb), where lb, ub are the lower and
        upper bounds of this entry. If m<lb, then it is set m=lb. If
        m>ub, then it is set m=ub.

Output arguments
    MUTATIONCHILDREN [double(length(PARENTS) x GENOMELENGTH)] is the
        mutated offspring.

Example
    % Create an options structure specifying that the mutation function
    % to be used is MUTATIONFUN, with MUTATIONRATE equal to 0.05 and
    % MUTATIONSCALE equal to 1.2.
    mutationRate = 0.05;
    mutationScale = 1.2;
    options=optimoptions('ga','MutationFcn',...
          {@mutationFun,mutationRate,mutationScale});
```