```
function wskaznik = dobierz_wskaznik_czestotliwosci(zrodlo,
 numer_pomiaru, liczba_pomiarow, min_czestotliwosc, max_czestotliwosc)

wskaznik = [];

for licznik = 1:liczba_pomiarow

    if zrodlo(numer_pomiaru).impedancja.czestotliwosc(licznik) <
 min_czestotliwosc
        wskaznik = [wskaznik; 0];
    elseif zrodlo(numer_pomiaru).impedancja.czestotliwosc(licznik) >
 max_czestotliwosc
        wskaznik = [wskaznik; 0];
    else
        wskaznik = [wskaznik; 1];
    end

end

end
--------------------------------------------------------------------------

function wynik = main_ga(sciezka_pliku, moduly, dolna_granica,
 gorna_granica)

%% Wczytywanie wynikow z pomiarow

[Z_exp_calosc, liczba_pomiarow] = wczytaj_LRC2(sciezka_pliku);
czestotliwosci = wczytaj_czestotliwosci2(sciezka_pliku);

%% Dane do symulacji

% przyjmowane od u#ytkownika
liczba_parametrow = policz_parametry(moduly);

%% Optymalizacja

for numer_pomiaru = 1:liczba_pomiarow

    Z_exp = Z_exp_calosc(numer_pomiaru).imp;

    suma_bledow = @(parametry) oblicz_sume_bledow(Z_exp, moduly,
 parametry, czestotliwosci);

    ga_wektor_parametrow = ga(suma_bledow, liczba_parametrow, [], [],
[] ,[], dolna_granica, gorna_granica);

    Z_sym = wykonaj_symulacje(moduly, ga_wektor_parametrow,
 czestotliwosci);


    %% Podsumowanie
```

```
    temperatura = Z_exp_calosc(numer_pomiaru).temperature;
    blad_bezwzgledny = oblicz_sume_bledow(Z_exp, moduly,
 ga_wektor_parametrow, czestotliwosci);

    wynik(numer_pomiaru) = podsumuj(ga_wektor_parametrow,
 blad_bezwzgledny, temperatura, Z_exp, Z_sym, czestotliwosci);

end

%% Wy#wietlenie wynikow - rysowanie wykresu, dopasowany wektor
 parametrow

    % wyswietlanie w aplikacji

end
--------------------------------------------------------------------------

function wynik = main_granice(sciezka_pliku, moduly, parametry,
 dolna_granica, gorna_granica)

%% Wczytywanie wynikow z pomiarow

[Z_exp_calosc, liczba_pomiarow] = wczytaj_LRC(sciezka_pliku);
czestotliwosci = wczytaj_czestotliwosci(sciezka_pliku);

%% Dane do symulacji

% wprowadzane z pliku

%% Optymalizacja

for numer_pomiaru = 1:liczba_pomiarow

Z_exp = Z_exp_calosc(numer_pomiaru).imp;

suma_bledow = @(parametry) oblicz_sume_bledow(Z_exp, moduly,
 parametry, czestotliwosci);
% [dolna_granica, gorna_granica] = wyznacz_granice(moduly);
optionsN = optimoptions('fmincon');

wektor_parametrow = fmincon(suma_bledow, parametry, [], [], [], [],
 dolna_granica, gorna_granica, [], optionsN);

Z_sym = wykonaj_symulacje(moduly, wektor_parametrow, czestotliwosci);


%% Wy#wietlenie wynikow - rysowanie wykresu, dopasowany wektor
 parametrow

% realizowane w appdesigner

%% Podsumowanie
```

```
temperatura = Z_exp_calosc(numer_pomiaru).temperature;

blad_bezwzgledny = oblicz_sume_bledow(Z_exp, moduly,
 wektor_parametrow, czestotliwosci);

wynik(numer_pomiaru) = podsumuj(wektor_parametrow, blad_bezwzgledny,
 temperatura, Z_exp, Z_sym, czestotliwosci);

end
end
--------------------------------------------------------------------------

% Obliczanie impedancji C
function ZC = oblicz_impedancje_C(C,w)

ZC = -1j./(w.*C);

end

% Obliczanie impedancji RC
function ZP = oblicz_impedancje_P(R,C,w)

ZR = oblicz_impedancje_R(R);
ZC = oblicz_impedancje_C(C,w);

S =  1/ZR + 1./ZC;

ZP = 1./S;

end

% Obliczanie impedancji R
function ZR = oblicz_impedancje_R(R)

ZR = R;

end


% Obliczanie impedancji CPE
function ZCPE = oblicz_impedancje_S(A,Y0,w)

ZCPE = 1./((1j*w).^A *Y0 );

end


% Obliczenie impedancji Warburga
function ZW = oblicz_impedancje_W(Z0,w)

ZW = Z0*(1-1j)./sqrt(w);

end
```

```
------------------------------------------------------------------------

function err = oblicz_sume_bledow(Z_exp, moduly, parametry,
 czestotliwosci)

Z_sym = wykonaj_symulacje(moduly, parametry, czestotliwosci);
error_diff = Z_sym - Z_exp;
error_sum = sum(abs(error_diff).^2);

liczba_uwzglednianych_pomiarow = length(Z_exp);

err = error_sum/liczba_uwzglednianych_pomiarow;

end

------------------------------------------------------------------------

function err = oblicz_sume_bledow_ponownie(Z_exp, moduly, parametry,
 czestotliwosci, wskaznik)

Z_sym = wykonaj_symulacje(moduly, parametry, czestotliwosci);
error_diff = Z_sym - Z_exp;
error_sum = sum(abs(error_diff.*wskaznik).^2);

liczba_uwzglednianych_pomiarow = sum(wskaznik);

err = error_sum/liczba_uwzglednianych_pomiarow;

end

------------------------------------------------------------------------

function wynik_dla_temp = optymalizuj_ponownie(wynik, numer_pomiaru,
 moduly, wskaznik)

Z_exp = wynik(numer_pomiaru).impedancja.Z_exp;
parametry = wynik(numer_pomiaru).wektor;
czestotliwosci = wynik(numer_pomiaru).impedancja.czestotliwosc;

suma_bledow = @(parametry) oblicz_sume_bledow_ponownie(Z_exp, moduly,
 parametry, czestotliwosci, wskaznik);
[dolna_granica, gorna_granica] = wyznacz_granice(moduly);
optionsN = optimoptions('fmincon');

wektor_parametrow = fmincon(suma_bledow, parametry, [], [], [], [],
 dolna_granica, gorna_granica, [], optionsN);
Z_sym = wykonaj_symulacje(moduly, wektor_parametrow, czestotliwosci);
blad_bezwzgledny = oblicz_sume_bledow_ponownie(Z_exp, moduly,
 wektor_parametrow, czestotliwosci, wskaznik);

wynik_dla_temp.wektor = wektor_parametrow;
wynik_dla_temp.blad = blad_bezwzgledny;
wynik_dla_temp.temperatura = wynik(numer_pomiaru).temperatura;
```

```
wynik_dla_temp.impedancja.czestotliwosc =
 wynik(numer_pomiaru).impedancja.czestotliwosc;
wynik_dla_temp.impedancja.Z_exp = Z_exp.*wskaznik;
wynik_dla_temp.impedancja.Z_sym = Z_sym.*wskaznik;
wynik_dla_temp.min_czestotliwosc = czestotliwosci(1);
wynik_dla_temp.max_czestotliwosc = czestotliwosci(end);

end

------------------------------------------------------------------------

function wynik = podsumuj(wektor_parametrow, blad, temperatura, Z_exp,
 Z_sym, czestotliwosci)

wynik.wektor = wektor_parametrow;
wynik.blad = blad;
wynik.temperatura = temperatura;
wynik.impedancja.czestotliwosc = czestotliwosci;
wynik.impedancja.Z_exp = Z_exp;
wynik.impedancja.Z_sym = Z_sym;
wynik.min_czestotliwosc = czestotliwosci(1);
wynik.max_czestotliwosc = czestotliwosci(end);

end

------------------------------------------------------------------------

function liczba_parametrow = policz_parametry(moduly)

liczba_parametrow = 0;
znak = 1;

for znak = 1:length(moduly)
    switch moduly(znak)
        case 'R'
            liczba_parametrow = liczba_parametrow + 1;

        case 'C'
            liczba_parametrow = liczba_parametrow + 1;

        case 'L'
            liczba_parametrow = liczba_parametrow + 1;

        case 'P'
            liczba_parametrow = liczba_parametrow + 2;

        case 'W'
            liczba_parametrow = liczba_parametrow + 1;

        case 'S'
            liczba_parametrow = liczba_parametrow + 2;

    end
end
```

```
end

-------------------------------------------------------------------------

function nowy_wynik_temp = uwzglednij_wskaznik_czestotliwosci(wynik,
 numer_pomiaru, moduly, min_czestotliwosc, max_czestotliwosc,
 liczba_uwzglednianych_pomiarow)

liczba_pomiarow = length(wynik(numer_pomiaru).impedancja.Z_exp);

wskaznik = dobierz_wskaznik_czestotliwosci(wynik, numer_pomiaru,
 liczba_pomiarow, min_czestotliwosc, max_czestotliwosc);

nowy_wynik_temp = optymalizuj_ponownie(wynik, numer_pomiaru, moduly,
 wskaznik);

end

-------------------------------------------------------------------------

function czestotliwosci = wczytaj_czestotliwosci(filename)

%% Import data from text file.
% Script for importing data from the text file

%% Initialize variables.
% filename = '/home/aak/Documents/inzynierka/test 2.LCR';
delimiter = ' ';
startRow = 5;
endRow = 17;

%% Format for each line of text:
formatSpec = '%f%f%f%f%f%f%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to the format.
dataArray = textscan(fileID, formatSpec, endRow-startRow
+1, 'Delimiter', delimiter, 'MultipleDelimsAsOne',
 true, 'TextType', 'string', 'EmptyValue', NaN, 'HeaderLines',
 startRow-1, 'ReturnOnError', false, 'EndOfLine', '\r\n');

%% Close the text file.
fclose(fileID);

%% Create output variable
macierz_czestotliwosci = [dataArray{1:end-1}];

czestotliwosci = [];
[liczba_wierszy, liczba_kolumn] = size(macierz_czestotliwosci);

for i = 1:liczba_wierszy
    for j = 1:liczba_kolumn
```

```
        if isnan(macierz_czestotliwosci(i,j)) == false
            czestotliwosci = [czestotliwosci
 macierz_czestotliwosci(i,j)];
        end
    end
end

czestotliwosci = czestotliwosci';

%% Clear temporary variables
clearvars i j liczba_kolumn liczba_wierszy macierz_czestotliwosci filename delimit

end

--------------------------------------------------------------------------------

function [wynik_pomiaru, liczba_pomiarow] = wczytaj_LRC(filename)


%% Initialize variables.
% filename = '/home/aak/Documents/inzynierka/test 2.LCR';
startRow = 18;

%% Read columns of data as text:
formatSpec = '%16s%17s%17s%22s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to the format.
dataArray = textscan(fileID,
 formatSpec, 'Delimiter', '', 'WhiteSpace', '', 'TextType', 'string', 'HeaderLines
 startRow-1, 'ReturnOnError', false, 'EndOfLine', '\r\n');

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric text to numbers.
% Replace non-numeric text with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = mat2cell(dataArray{col},
 ones(length(dataArray{col}), 1));
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[1,2,3,4]
    % Converts text in the input cell array to numbers. Replaced non-
numeric
    % text with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1)
        % Create a regular expression to detect and remove non-numeric
 prefixes and
```

```
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]
{0,1}\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d
+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData(row), regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if numbers.contains(',')
                thousandsRegExp = '^[-/+]*\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(numbers, thousandsRegExp, 'once'))
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric text to numbers.
            if ~invalidThousandsSeparator
                numbers =
 textscan(char(strrep(numbers, ',', '')), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch
            raw{row, col} = rawData{row};
        end
    end
end


%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw); % Find non-
numeric cells
raw(R) = {NaN}; % Replace non-numeric cells

%% Create output variable
all = cell2mat(raw);
wynik_pomiaru = [];

for i = 1:1:size(all)

    if any(isnan(all(i,:)))
        start_row = all(i,:);
        liczba_pomiarow = start_row(1);
        wynik_pomiaru(liczba_pomiarow).number = start_row(1);
        wynik_pomiaru(liczba_pomiarow).temperature = start_row(2);
        j = 1;
    else
        wynik_pomiaru(liczba_pomiarow).results.Z(j) = all(i,1);
        wynik_pomiaru(liczba_pomiarow).results.teta(j) = all(i,2);
        wynik_pomiaru(liczba_pomiarow).results.E_1(j) = all(i,3);
        wynik_pomiaru(liczba_pomiarow).results.E_2(j) = all(i,4);
```

```
        j = j + 1;

    end

end

    for i = 1:liczba_pomiarow
        wynik_pomiaru(i).results.Z = (wynik_pomiaru(i).results.Z)';
        wynik_pomiaru(i).results.teta =
 (wynik_pomiaru(i).results.teta)';
        wynik_pomiaru(i).results.E_1 =
 (wynik_pomiaru(i).results.E_1)';
        wynik_pomiaru(i).results.E_2 =
 (wynik_pomiaru(i).results.E_2)';
    end

%% Calc impedation

    for i = 1:liczba_pomiarow

%         test_result(i).imp = test_result(i).results.Z .*
 exp(1j.*(test_result(i).results.teta)*360/(2*pi));
        wynik_pomiaru(i).imp = wynik_pomiaru(i).results.Z .*
 exp(1j.*(wynik_pomiaru(i).results.teta));

    end

%% Clear temporary variables
clearvars all i j start_row filename startRow endRow formatSpec fileID dataArray a

end

--------------------------------------------------------------------------------

function Z_sym = wykonaj_symulacje(moduly, parametry, czestotliwosci)

liczba_modulow = length(moduly);
aktualny_indeks = 1;

w = czestotliwosci;

wynik_symulacji = zeros([length(w) 1]);

for sign = 1:liczba_modulow

    switch moduly(sign)
        case 'R'
            r = 10^(parametry(aktualny_indeks));
            impedancja_pomiaru = oblicz_impedancje_R(r);
            aktualny_indeks = aktualny_indeks + 1;

        case 'C'
            c = 10^(parametry(aktualny_indeks));
            impedancja_pomiaru = oblicz_impedancje_C(c, w);
```

```
                aktualny_indeks = aktualny_indeks + 1;

            case 'W'
                Z0 = 10^(parametry(aktualny_indeks));
                impedancja_pomiaru = oblicz_impedancje_W(Z0, w);
                aktualny_indeks = aktualny_indeks + 1;


            case 'S'
                A = parametry(aktualny_indeks);
                Y0 = 10^(parametry(aktualny_indeks + 1));
                impedancja_pomiaru = oblicz_impedancje_S(A,Y0, w);
                aktualny_indeks = aktualny_indeks + 2;

            case 'P'
                r = 10^(parametry(aktualny_indeks));
                c = 10^(parametry(aktualny_indeks + 1));
                impedancja_pomiaru = oblicz_impedancje_P(r, c, w);
                aktualny_indeks = aktualny_indeks + 2;

        end

        wynik_symulacji = wynik_symulacji + impedancja_pomiaru;

    end


    Z_sym = wynik_symulacji;

    end

    ------------------------------------------------------------------------

    function [lb, ub] = wyznacz_granice(moduly)

    liczba_modulow = length(moduly);

    lb = [];
    ub = [];

    for litera = 1:liczba_modulow

        switch moduly(litera)
            case 'R'
                lb = [lb, -20];
                ub = [ub, 7];
            case 'C'
                lb = [lb, -20];ra)
            case 'R'
                lb = [lb, -20];
                ub = [ub, 7];
            case 'C'
                lb = [lb, -20];
                ub = [ub, -5];
```

```
        case 'W'
            lb = [lb, -11];
            ub = [ub, -3];

        case 'S'
            lb = [lb, 0, -11];
            ub = [ub, 1, 0];

        case 'P'
            lb = [lb 0, -20];
            ub = [ub 7, -3];
        end
end
end

------------------------------------------------------------------------------

function startRow = znajdzStartRow(filename)

%% Initialize variables.
delimiter = ' ';
startRow = 4;
endRow = 4;

%% Format for each line of text:
formatSpec = '%f%*s%*s%*s%*s%*s%*s%*s%*s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to the format.
dataArray = textscan(fileID, formatSpec, endRow-startRow
+1, 'Delimiter', delimiter, 'MultipleDelimsAsOne',
 true, 'TextType', 'string', 'HeaderLines',
 startRow-1, 'ReturnOnError', false, 'EndOfLine', '\r\n');

%% Close the text file.
fclose(fileID);

%% Create output variable
liczbaCzestotliwosci = [dataArray{1:end-1}];
%% Clear temporary variables
clearvars filename delimiter startRow endRow formatSpec fileID dataArray ans;

liczbaWierszyCzest = idivide(liczbaCzestotliwosci, int32(8)) + 1;

startRow = 4 + liczbaWierszyCzest + 1

end
```

Published with MATLAB® R2018b