```
>> type gradient
function varargout = gradient(f,varargin)
%GRADIENT Przybliżony gradient.
% [FX,FY] = GRADIENT(F) zwraca liczbowy gradient macierzy F.
% FX odpowiada dF/dx, czyli różnicom w kierunku x (poziomym).
% FY odpowiada dF/dy, czyli różnicom w kierunku y (pionowym).
% Przyjmuje się, że odstępy między punktami w każdym kierunku wynoszą jeden.
% Gdy F jest wektorem, DF = GRADIENT(F) jest gradientem 1-D.
%
% [FX,FY] = GRADIENT(F,H), gdzie H jest skalarem, wykorzystuje H jako stałą
% odstępu między punktami w każdym kierunku.
%
% [FX,FY] = GRADIENT(F,HX,HY), gdy F jest dwuwymiarowe, używa odstępów określonych
% przez HX i HY. HX i HY mogą być albo skalarami, aby określić odstęp
% między współrzędnymi lub wektorami określającymi współrzędne
% punktów. Jeśli HX i HY są wektorami, to ich długości muszą odpowiadać
% odpowiadającym im wymiarom F.
%
% [FX,FY,FZ] = GRADIENT(F), gdy F jest tablicą trójwymiarową, zwraca liczbowy % gradient F. FZ odpowiada
gradientowi FZ.
% FZ odpowiada dF/dz, czyli różnicom w kierunku z
% kierunku. GRADIENT(F,H), gdzie H jest skalarem, używa H jako stałej
% odstępu między punktami w każdym kierunku.
%
% [FX,FY,FZ] = GRADIENT(F,HX,HY,HZ) wykorzystuje odstępy określone przez HX, HY, HZ.
%
% [FX,FY,FZ,...] = GRADIENT(F,...) rozszerza się podobnie, gdy F jest N-D i
% musi być wywołana z N wyjściami i 2 lub N+1 wejściami.
%
% Uwaga: Pierwsze wyjście FX jest zawsze gradientem wzdłuż pierwszego wymiaru
% wymiaru F, przechodzący przez kolumny.  Drugie wyjście FY jest zawsze
% gradientem wzdłuż drugiego wymiaru F, przechodzącym przez wiersze.  Dla strony
% trzeciego wyjścia FZ i kolejnych wyjść, N-te wyjście jest
% gradientem wzdłuż N-tego wymiaru F.
%
% Przykłady:
```

```
% x = -2:.2:2; y = (-1:.15:1).";
% z = x .* exp(-x.^2 - y.^2);
% [px,py] = gradient(z,.2,.2);
% contour(x,y,z), hold on
% quiver(x,y,px,py), hold of
%    for k = 1:ndimsf    for k = 1:ndimsf
% Obsługa klas dla danych wejściowych F:
% float: double, single
%
% Zobacz także DIFF, DEL2.
% Copyright 1984-2019 The MathWorks, Inc.
[f,ndim,loc,rflag] = parse_inputs(f,varargin);
nargoutchk(0,ndim);
% Pętla nad każdym wymiarem.
varargout = cell(1,ndim);
siz = size(f);
% pierwszy wymiar
prototyp = real(full(f([])));
g = zeros(siz, 'like', prototyp); % przypadek wymiaru pojedynczego
h = loc{1}(:);
n = siz(1);
% Bierzemy różnice w przód na lewej i prawej krawędzi
if n > 1
  g(1,:) = (f(2,:) - f(1,:))/(h(2)-h(1));
  g(n,:) = (f(n,:) - f(n-1,:))/(h(end)-h(end-1));
end
% Wyznacz różnice wyśrodkowane na punktach wewnętrznych
if n > 2
  g(2:n-1,:) = (f(3:n,:)-f(1:n-2,:)) ./ (h(3:n) - h(1:n-2));
end
varargout{1} = g;
% drugi wymiar i więcej
if ndim == 2
   % szczególny przypadek macierzy 2-D do obsługi macierzy nieliczbowych,
   % które nie obsługują operacji N-D, w tym przekształcania
   % i indeksowanie
```

```
    n = siz(2);
    h = reshape(loc{2},1,[]);
    g = zeros(siz, 'like', prototyp);

    % Uwzględnij różnice w przód na lewej i prawej krawędzi
    if n > 1
        g(:,1) = (f(:,2) - f(:,1))/(h(2)-h(1));
        g(:,n) = (f(:,n) - f(:,n-1))/(h(end)-h(end-1));
    end

    % Wyznacz różnice wyśrodkowane na punktach wewnętrznych
    if n > 2
        h = h(3:n) - h(1:n-2);
        g(:,2:n-1) = (f(:,3:n) - f(:,1:n-2)) ./ h;
    end
    varargout{2} = g;

elseif ndim > 2
    % Przypadek N-D
    for k = 2:ndim
        n = siz(k);
        newsiz = [prod(siz(1:k-1)) siz(k) prod(siz(k+1:end))];
        nf = reshape(f,newsiz);
        h = reshape(loc{k},1,[]);
        g = zeros(size(nf), 'like', prototype);

        % Weź do przodu różnice na lewej i prawej krawędzi
        if n > 1
            g(:,1,:) = (nf(:,2,:) - nf(:,1,:))/(h(2)-h(1));
            g(:,n,:) = (nf(:,n,:) - nf(:,n-1,:))/(h(end)-h(end-1));
        end

        % Wyznacz różnice wyśrodkowane na punktach wewnętrznych
        if n > 2
            h = h(3:n) - h(1:n-2);
            g(:,2:n-1,:) = (nf(:,3:n,:) - nf(:,1:n-2,:)) ./ h;
```

```
        end

        varargout{k} = reshape(g,siz);
    end
end
% Zamień 1 i 2, ponieważ x jest drugim wymiarem, a y pierwszym.
if ndim > 1
    varargout([2 1]) = varargout([1 2]);
elseif rflag
    varargout{1} = varargout{1}.';
end
%---------------------------------------------------
function [f,ndim,loc,rowflag] = parse_inputs(f,h)
loc = {}; % odstępy wzdłuż kierunków x,y,z,...
ndimsf = ndims(f);
ndim = ndimsf;
rowflag = false;
if isvector(f)
    ndim = 1;
    if isrow(f) % Traktuj wektor wierszy jako wektor kolumn
        rowflag = true;
        f = f.';
    end
end
% Domyślne rozmiary kroków: hx = hy = hz = 1
indx = size(f);
if isempty(h)
    % gradient(f)
    loc = cell(1,ndimsf);
    for k = 1:ndimsf
        loc(k) = {1:indx(k)};
    end
elseif isscalar(h) % gradient(f,h)
    if isscalar(h{1})
        % Rozszerz wielkość kroku skalarnego
        loc = cell(1,ndimsf);
```

```
        for k = 1:ndimsf
            loc(k) = {h{1}*(1:indx(k))};
        end
    elseif ndim == 1
        % Sprawdzenie, czy istnieje przypadek wektora
        if numel(h{1}) ~= numel(f)
            error(message('MATLAB:gradient:InvalidGridSpacing'));
        end
        loc(1) = h(1);
    else
        error(message('MATLAB:gradient:InvalidInputs'));
    end
elseif ndimsf == numel(h) % gradient(f,hx,hy,hz,...)
    % Zamień 1 i 2, ponieważ x jest drugim wymiarem, a y pierwszym.
    loc = h;
    if  ndim > 1
        loc([2 1]) = loc([1 2]);
    end
    % zastąp dowolny skalarny rozmiar kroku odpowiednim wektorem pozycji, oraz
    % sprawdzić, czy wartości podane w każdym wektorze położenia mają właściwy
    % kształt i rozmiar.
    for k = 1:ndimsf
        if isscalar(loc{k})
            loc{k} = loc{k}*(1:indx(k));
        elseif ~isvector(squeeze(loc{k})) || numel(loc{k}) ~= size(f, k)
            error(message('MATLAB:gradient:InvalidGridSpacing'));
        end
    end
else
    error(message('MATLAB:gradient:InvalidInputs'));
end
```