Symulacja obwodu RLC

Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Teleinformatyka

Metody numeryczne

Listopad 2017

Piotr Gregor

Wstęp

Dokument ten jest raportem z wykonania symulacji stanu nieustalonego obwodu szeregowego RLC w obecności wymuszjącej siły elektromotorycznej o zadanej wartości częstotliwości i wartości maksymalnej (amplitudy). Symulacji dokonano w środowisku Matlab.

W części pierwszej przedstawiamy rezultaty symulacji obwodu dla napięcia i prądu z wykorzystaniem ulepszonej metody Eulera. W części drugiej wyznaczamy pojemność kondensatora w zadanym obwodzie, niezbędną aby obwód oscylował w częstotliwości rezonansowej. W części trzeciej szacujemy moc obwodu całkując chwilowe wartości iloczynu napięcia wymuszającego I prądu w obwodzie, korzystając z metody złożonej parabol (Simpsona).

Jako parametry obwodu przyjęto: R = 30 Ohm, L = 20 mH, C = µF.

Spis treści

1. Prąd i napięcie w stanie nieustalonym

1.1 Program

Metodą ulepszoną Eulera symulowano obwód z wykorzystaniem następującego kodu.

```
% Simulate RLC circuit using improved Euler method.
%
% Params:
% t0 - start time [ms]
% dt - time resolution [ms]
% n - number of steps (total simulation time = n * dt)
% R - resistance
% L - inductance
% C - capacitance
% EMFm - max value of electromotive force EMF (amplitude)
% f - frequency of the EMF (driving frequency in Hz)
% square – if 0 ignorred, otherwise makes square wave
%
% Details
% Function is using improved Euler method for function approximation.
%
% Piotr Gregor <piotr@dataanadsignal.com>


function [] = rlc_simulate_euler(t0, dt, n, R, L, C, EMFm, f, square)
    clearvars -global
    global t0_
    global dt_
    global h_
    global n_
    global R_
    global L_
    global C_

    global uC_
    global iL_
    global uCe_
    global iLe_
    global diL_

    global EMF_
    global EMFm_
    global f_d_
    global omega_d_

    global omega_
    global f_
    global XC_
    global XL_
    global XC_res_
    global XL_res_


    % Vectors
```

```
    t = t0 : dt : t0 + n * dt;
    uC_ = t;
    iL_ = t;
    duC = t;
    diL_ = t;
    uCe_ = t;
    dUce = t;
    iLe_ = t;

    % Init state of RLC circuit
    t0_ = t0;
    dt_ = dt;
    h_ = dt_ / 2;
    n_ = n;
    R_ = R;
    L_ = L;
    C_ = C;


    % Driving angular frequency
    f_d_ = f;
    omega_d_ = 2 * pi * f;
    XL_ = omega_d_ * L;
    XC_ = 1 / (omega_d_ * C);

    % Circuit's natural frequency
    omega_ = 1 / sqrt(L * C);
    f_ = omega_ / (2 * pi);
    XL_res_ = omega_ * L;
    XC_res_ = 1 / (omega_ * C);

    % Initial inputs of EMF, voltage across capacitor and current through inductor
    EMFm_ = EMFm;
    if (f_d_ == 0.0)
        EMF_ = EMFm_ * ones(1,n + 1);
        if (square)
            EMF_(n / 4 : n / 2) = 0;
            EMF_(3 * n / 4 : end) = 0;
        end
    else
        EMF_ = EMFm_ * sin(omega_d_ * t);
    end

    uC_(1) = 0;     % initial voltage across capacitor must be zero
    iL_(1) = 0;     % initial current through the circuit is zero
    uCe_(1) = 0;
    iLe_(1) = 0;

    % Print info
    fprintf("Circuit's parameters:\n");
    fprintf("EMF:\n     max %f\n", EMFm);
    fprintf("Frequency:\n");
    fprintf("    freq %f [Hz]\n    angular driving freq %f [rad/s] (omega), T %f\n", f_d_, omega_d_,
1/f_d_);
    fprintf("Natural freq (resonanant)\n");
    fprintf("    freq %fHz\n    angular natural freq %f [rad/s] (omega), T %f\n", f_, omega_, 1/f_);
    fprintf("Reactance:\n");
    fprintf("    XL %f\n    XC %f", XL_, XC_);
    if (XL_ > XC_)
        fprintf("    circuit is INDUCTIVE\n");
    else
        if (XL_ < XC_)
```

```
            fprintf("    circuit is CAPACITIVE\n");
        else
            fprintf("    circuit is RESISTIVE\n");
        end
    end
    fprintf("Reactance (resonant):\n");
    fprintf("    XL(at resonance) %f\n    XC(at resonance) %f\n", XL_res_, XC_res_);

    % Error checking
    if (n < 2)
        fprintf("\nErr, cannot run simulation. Number of steps too small...\n");
        return
    end

    fprintf("Simulating %d steps...\n", n);
    fprintf("Time step (time differential) dt %f\n", dt_);

    % Simulation
    for i = 1 : n + 1
        if (i > 1)
            % uC
            duC(i) = duC_dt(i) * dt;
            uC_(i) = uC_(i - 1) + duC(i - 1);

            % iL
            diL_(i) = diL_dt(i) * dt;
            iL_(i) = iL_(i - 1) + diL_(i - 1);

            % uCe_
            uCe_(i) = uCe_(i - 1) + h_ * duC_dt_e(i - 1);
            %uCe_(i) = uC_(i);

            % iLe_
            iLe_(i) = iLe_(i - 1) + h_ * diL_dt_e(i - 1);
        end

    end

    % plot
    plot_as_one(t, EMF_, 'EMF', uC_, 'uC', uCe_, 'uC euler', iL_, 'iL', iLe_, 'iL euler');
end


% Derivation

function [duC_dt] = duC_dt(i)
    global C_
    global iL_

    if (i == 1)
        duC_dt = 0;
    else
        duC_dt = (iL_(i - 1) ./ C_);
    end
end

function [diL_dt] = diL_dt(i)
    global R_
    global L_
    global iL_
    global uC_
    global EMF_
```

```
    if (i == 1)
        diL_dt = 0;
    else
        diL_dt = (EMF_(i - 1) - R_ .* iL_(i - 1) - uC_(i - 1)) ./ L_;
    end
end

function [duC_dt_e] = duC_dt_e(i)
    global C_
    global iL_
    global h_

    if (i == 1)
        duC_dt_e = 0;
    else
        h05 = h_ / 2;
        didt = diL_dt(i);
        duC_dt_e = ((iL_(i - 1) + h05 .* didt) / C_);   % estimate uC_dt in half of dt_
    end
end

function [diL_dt_e] = diL_dt_e(i)
    global R_
    global L_
    global iLe_
    global uC_
    global EMF_
    global h_

    if (i == 1)
        diL_dt_e = 0;
    else
        h05 = h_ / 2;
        didt = (EMF_(i - 1) - R_ * iLe_(i - 1) - uC_(i - 1)) / L_;
        diL_dt_e = (EMF_(i - 1) - R_ * (iLe_(i - 1) + h05*didt) - uC_(i - 1)) / L_;
    end
end


% Plotting

function [] = plot_as_one(t, y1, s1, y2, s2, y3, s3, y4, s4, y5, s5)
    figure();
    xlabel('Time [s]');
    yyaxis left
    plot(t, y1, 'r', t, y2, 'g', t, y3, 'blue');
    ylabel('Potential difference [V]');

    yyaxis right
    plot(t, y4, 'black', t, y5, 'blue');
    ylabel('Current [A]');
    legend(s1, s2, s3, s4, s5);
end

function [] = plot_as_sub(t, y1, s1, y2, s2, y3, s3)
    figure();
    subplot(3,1,1), plot(t, y1,'r');
    legend(s1);
    subplot(3,1,2), plot(t, y2,'g');
    legend(s2);
    subplot(3,1,3), plot(t, y3,'b');
```

```
    legend(s3);
end

function [] = plot_all(t, y1, s1, y2, s2)
    figure();
    p1 = plot(t, y1, 'r');
    legend(p1, s1);
    figure();
    p2 = plot(t, y2, 'b');
    legend(p2, s2);
end
```

Uzyskano następujące wyniki, dla różnej funkcji napięcia wymuszającego.

1.2 E = 1V

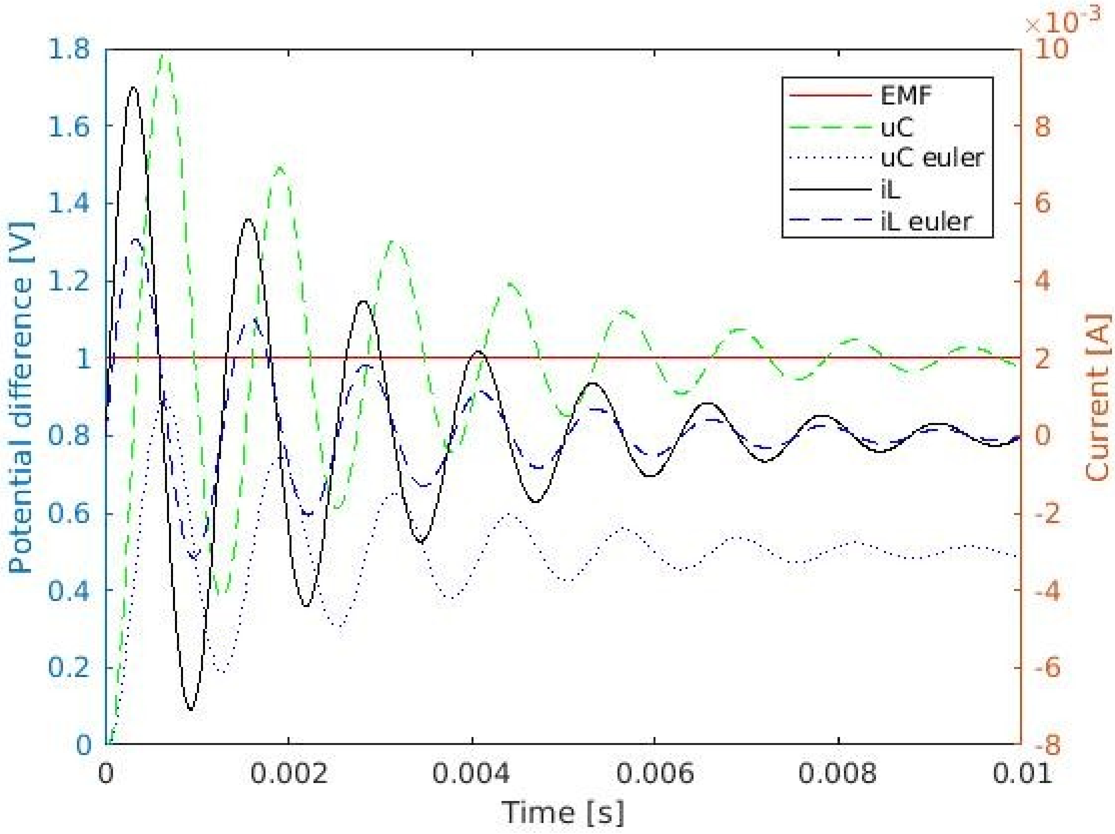

Illustration 1: E = 1V

1.3 E = 1V * sin(2 * PI * 50 * t)

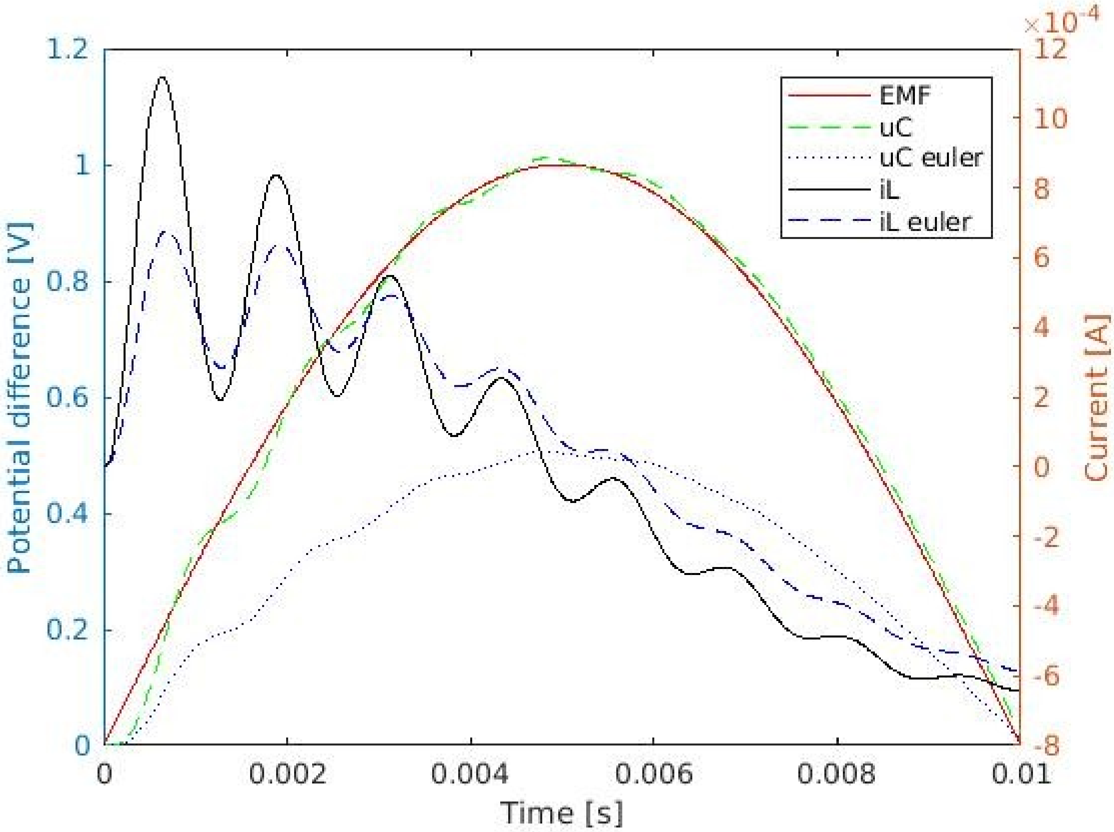

*Illustration 2: E = sin(2 * PI * 50 * t)*

1.4 E = 1V * sin(2 * PI * 796 * t)

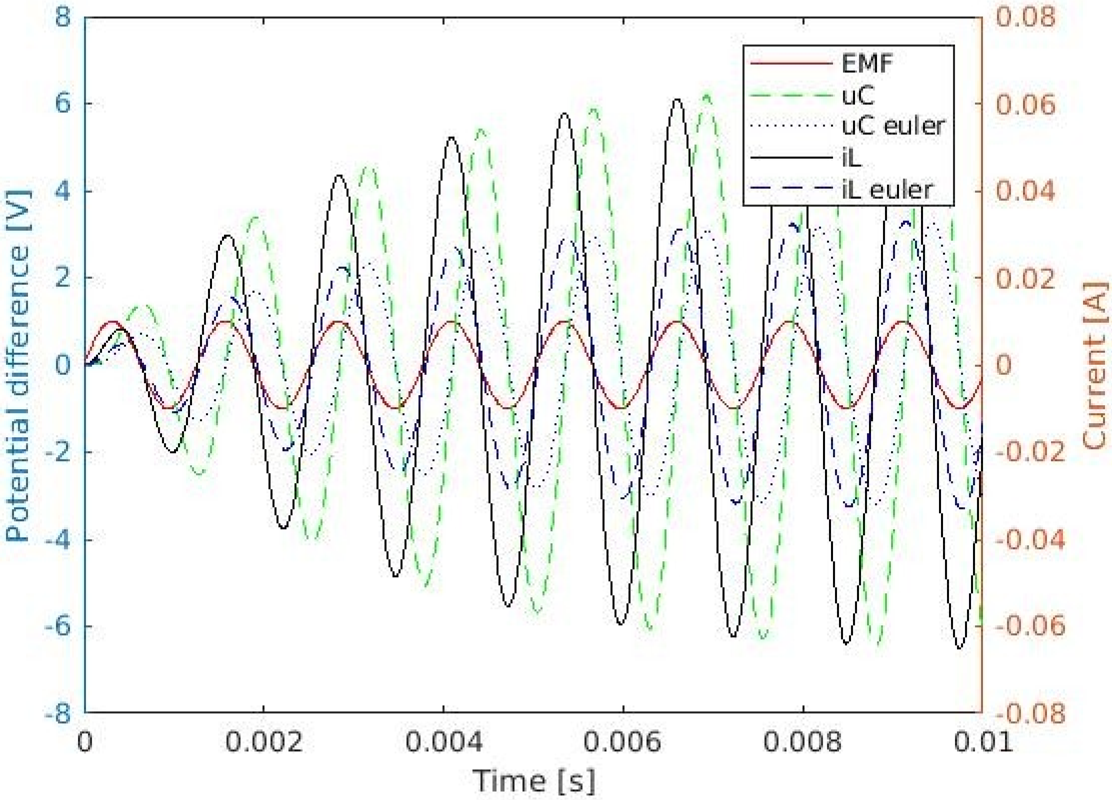

*Illustration 3: E = sin(2 * PI * 796 * t)*

1.5 E = 1V * sin(2 * PI * 900 * t)

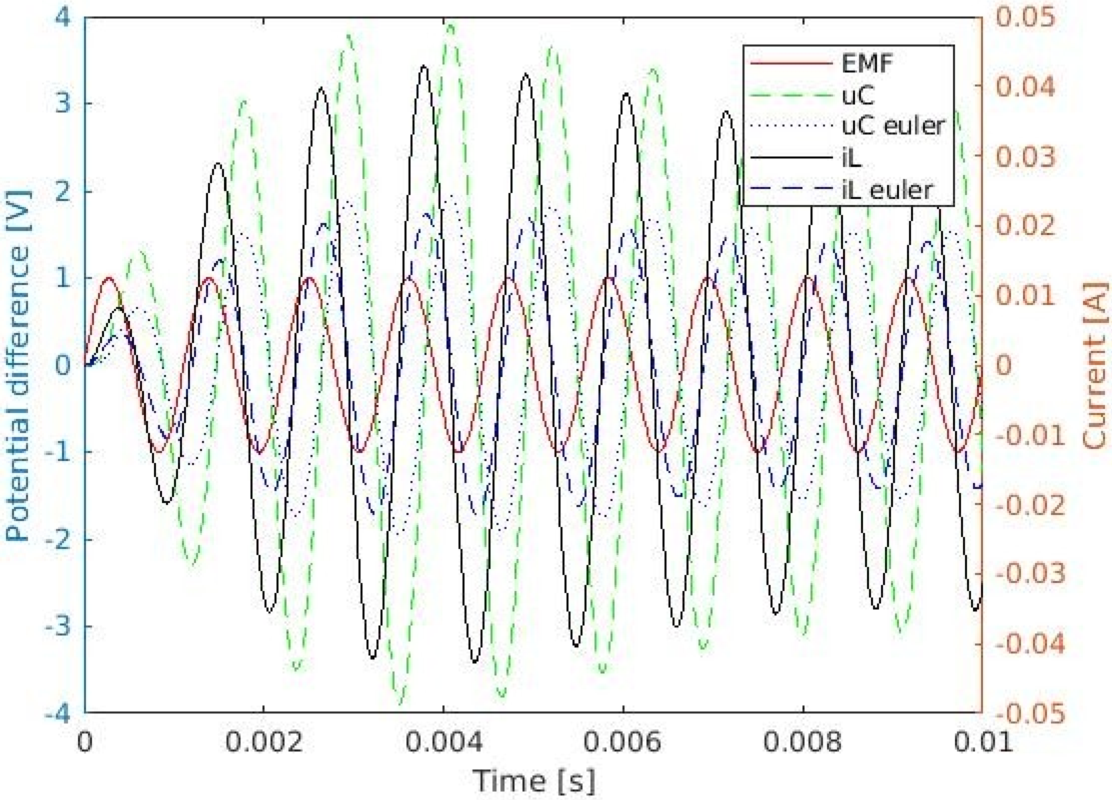

*Illustration 4: E = sin(2 * PI * 900 * t)*

1.6 E = 1V, squared

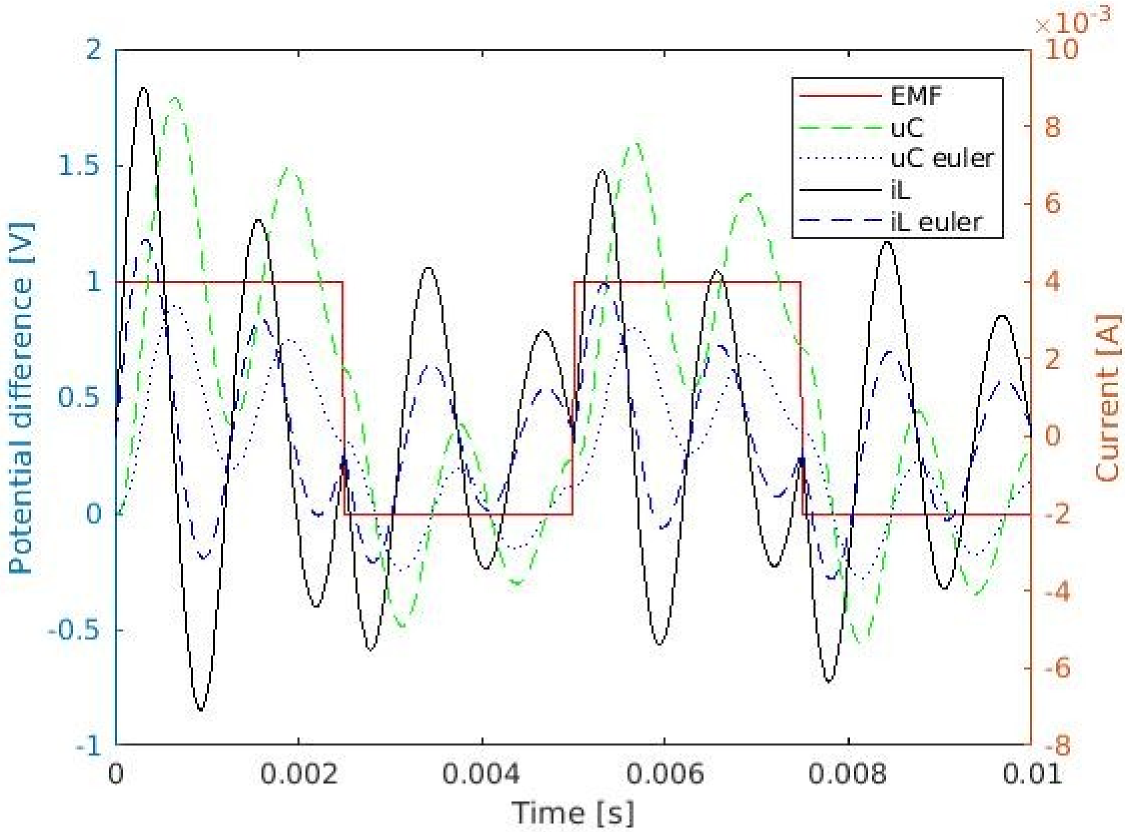

Illustration 5: E= 1V w przedziale [0,250]ms oraz [500, 750]ms, 0 w [250, 500]ms oraz [750, 1000]ms

2. Szacowanie pojemności rezonansowej

Metodą Newton-Raphson oszacowano pojemność kondensatora niezbędną do tego aby obwód RLC, przy zadanej wartości rezystancji, indukcyjności oraz napięcia wymuszającego znajdował się w stanie rezonansu, to jest częstotliwość naturalna obwodu zrównała się częstotliwości napięcia wymuszjącego.

```
>> f = rlc_resonant(500, 0.000002, 0.000008)

f =

   5.0661e-06
```

2.1 Program

```
% Find the capacitance value resulting in the circuit natural frequency
% being equal to driving (forced) frequency.
%
% Params:
% C_min - capacitance left bounday
% C_max - capacitance right boundary
% f_d - forced frequency of the EMF (driving frequency in Hz)
%
% Return
% Capacitance needed to make circuit oscillating at resonance (f == f_d).
%
% Details:
% Function is using Newton-Raphson method for root searching.
%
% Piotr Gregor <piotr@dataanadsignal.com>

function [ret] = rlc_resonant(f_d, C_min, C_max)
clearvars -global
global L_

global N_  % max number of iterations performed in Newton-Raphson method

L_ = 0.02;
N_ = 1000;
epsilon = 0.000001;   % accuracy of root searching in Newton-Raphson

if ((C_min == 0) || (C_max == 0))
    fprintf("Err, capacitance cannot be 0 in 'proper' RLC circuit.");
    fprintf(" Please change the range\n");
    ret = NaN;
    return;
end

if (C_min >= C_max)
    fprintf("Err, capacitance is not a range: C_min == C_max == %f\n", C_min);
    ret = NaN;
    return;
end

% Init
dx = (C_max - C_min) / N_;
x = C_min : dx : C_max;

% Circuit's is in resonance if forced frequency f_d equals natural circuit's
% frequency f = 1/(2*pi*sqrt(LC))
y = @(x) 1.0 ./ (2.0 .* pi * sqrt(L_ .* x)) - f_d;
x_min = C_min;
x_max = C_max;
y_prim = @(x) -1.0 ./ (4.0 * pi * sqrt(L_ .* x .* x .* x));
y_prim_prim = @(x) 3.0 ./ (8.0 .* pi .* x .* x .* sqrt(L_ .* x));

ret = newton_raphson(x, y, x_min , x_max, y_prim, y_prim_prim, N_, epsilon);

end

function [ret] = newton_raphson(x, y, x_min , x_max, y_prim, y_prim_prim, N, epsilon)
global N_
```

```
if (y(x_min) == 0)
    ret = x_min;
    return;
end

if (y(x_max) == 0)
    ret = x_max;
    return;
end

if (x_min > x_max)
    fprintf("Err, x argument is not a range: x_min == x_max == %f\n", x_min);
    ret = NaN;
    return;
end

% Check mandatory conditions
if (y(x_min) * y(x_max) > 0)
    fprintf("Err, bad init condition: y boundaries are same in sign\n");
    fprintf("Please consider increasing range or shifting it.\n");
    ret = NaN;
    return;
end

% Choose starting point
x = x_min;
if (y(x_max) * y_prim_prim(x_max) > 0)
    x = x_max;
end

% Iterative computation
i = 1;
x_vec = 1 : 1 : 1000;
y_vec = 1 : 1 : 1000;
x_vec(1) = x;
y_vec(1) = y(x);
while ((i < N) && (abs(y(x)) > epsilon))
    x_vec(i + 1) = x;
    y_vec(i + 1) = y(x);
    x = x - y(x) / y_prim(x);
    i = i + 1;
end

if ((i > 1) && (i < N))
    x_vec(i + 1) = x;
    y_vec(i + 1) = y(x);
    i = i + 1;
end

if (i < N)
    x_vec = x_vec(1 : i);
    y_vec = y_vec(1 : i);
end

sz = linspace(100, 1, i);
color = linspace(1, 10, i);
scatter(x_vec, y_vec, sz, color, 'filled');
xlabel("Capacitnace [F]");
ylabel("Frequency [Hz]");
legend({['f - f_d : diff between' char(10) 'circuits frequency' char(10) 'at the given capacitance'
char(10) 'and the driving EMF''s freq']});
```

```
ret = x;
end
```

2.2 Wykres

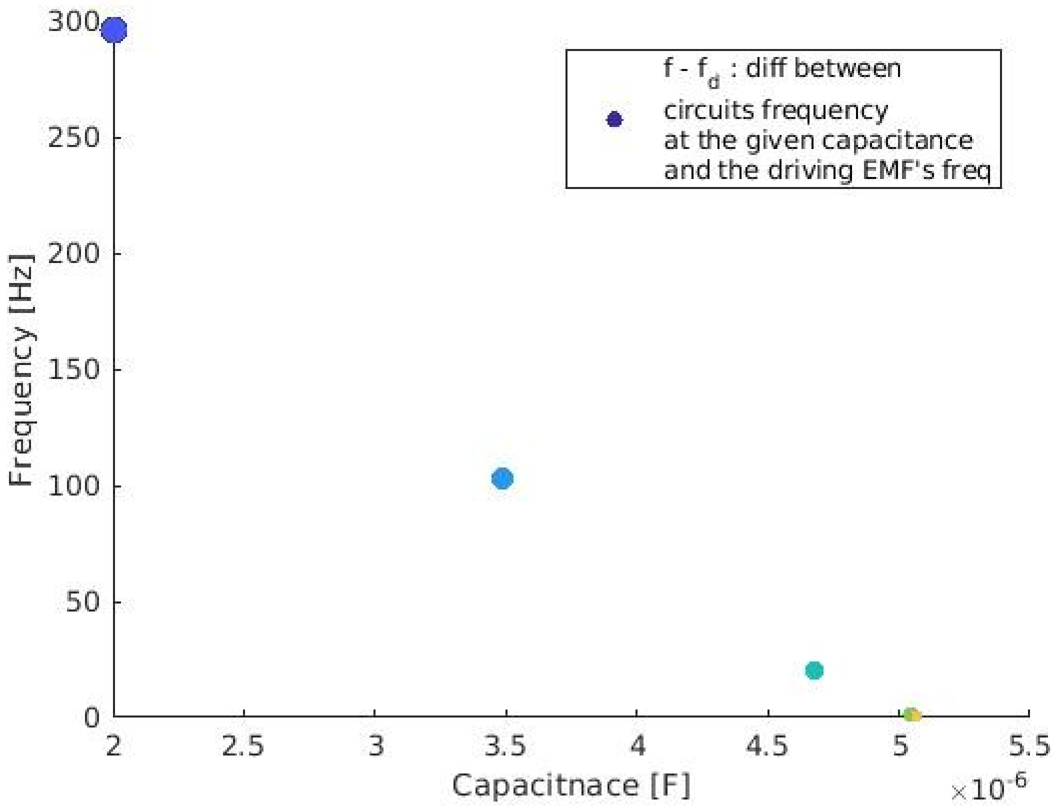

Illustration 6: Zbieganie pojemności do wartości rezonansowej w metodzie Newton-Raphson

3 Moc

Metodą złożonych parabol (Simpsona) oszacowano całkę z wartości chwilowych iloczynu napięcia wymuszającego oraz prądu w obwodzie.

```
>> p = rlc_simulate_power_euler(0, 0.000010, 1000, 30, 0.02, 0.000002, 1, 0);
Circuit's parameters:
EMF:
    max 1.000000
```

Frequency:

freq 0.000000 [Hz]

angular driving freq 0.000000 [rad/s] (omega), T Inf

Natural freq (resonanant)

freq 795.774715Hz

angular natural freq 5000.000000 [rad/s] (omega), T 0.001257

Reactance:

XL 0.000000

XC Inf   circuit is CAPACITIVE

Reactance (resonant):

XL(at resonance) 100.000000

XC(at resonance) 100.000000

Simulating 1000 steps...

Time step (time differential) dt 0.000010

>> p

p =

1.9780e-06

3.1 Program

```
% Simulate power produced by RLC circuit.
%
% Details
% Using improved Euler method for U and I. Using Simpson method for
% integration of instantaneous power.
%
% Params:
% t0 - start time [ms]
% dt - time resolution [ms]
% n - number of steps (total simulation time = n * dt)
% R - resistance
% L - inductance
% C - capacitance
% EMFm - max value of electromotive force EMF (amplitude)
% f - frequency of the EMF (driving frequency in Hz)
%
% Return
```

```
% Power produced by the circuit in time [t0, t0 + n*dt].
%
% Piotr Gregor <piotr@dataanadsignal.com>


function [power] = rlc_simulate_power_euler(t0, dt, n, R, L, C, EMFm, f)
    clearvars -global
    global t0_
    global dt_
    global h_
    global n_
    global R_
    global L_
    global C_

    global uC_
    global iL_
    global uCe_
    global iLe_
    global diL_

    global EMF_
    global EMFm_
    global f_d_
    global omega_d_

    global omega_
    global f_
    global XC_
    global XL_
    global XC_res_
    global XL_res_

    power = NaN;

    % Vectors
    t = t0 : dt : t0 + n * dt;
    uC_ = t;
    iL_ = t;
    duC = t;
    diL_ = t;
    uCe_ = t;
    dUce = t;
    iLe_ = t;

    % Init state of RLC circuit
    t0_ = t0;
    dt_ = dt;
    h_ = dt_ / 2;
    n_ = n;
    R_ = R;
    L_ = L;
    C_ = C;

    if (mod(n_, 2))
        fprintf("Err, number of steps must be even\n");
        return;
    end
    p_ = t(1 : n_ / 2);
    p_time_ = 2 * t(1 : n_ / 2);

    % Driving angular frequency
```

```
    f_d_ = f;
    omega_d_ = 2 * pi * f;
    XL_ = omega_d_ * L;
    XC_ = 1 / (omega_d_ * C);

    % Circuit's natural frequency
    omega_ = 1 / sqrt(L * C);
    f_ = omega_ / (2 * pi);
    XL_res_ = omega_ * L;
    XC_res_ = 1 / (omega_ * C);

    % Initial inputs of EMF, voltage across capacitor and current through inductor
    EMFm_ = EMFm;
    if (f_d_ == 0.0)
        EMF_ = EMFm_ * ones(1, n + 1);
    else
        EMF_ = EMFm_ * sin(omega_d_ * t);
    end

    uC_(1) = 0;     % initial voltage across capacitor must be zero
    iL_(1) = 0;     % initial current through the circuit is zero
    uCe_(1) = 0;
    iLe_(1) = 0;

    % Print info
    fprintf("Circuit's parameters:\n");
    fprintf("EMF:\n     max %f\n", EMFm);
    fprintf("Frequency:\n");
    fprintf("    freq %f [Hz]\n    angular driving freq %f [rad/s] (omega), T %f\n", f_d_, omega_d_,
1/f_d_);
    fprintf("Natural freq (resonanant)\n");
    fprintf("    freq %fHz\n    angular natural freq %f [rad/s] (omega), T %f\n", f_, omega_, 1/f_);
    fprintf("Reactance:\n");
    fprintf("    XL %f\n    XC %f", XL_, XC_);
    if (XL_ > XC_)
        fprintf("    circuit is INDUCTIVE\n");
    else
        if (XL_ < XC_)
            fprintf("    circuit is CAPACITIVE\n");
        else
            fprintf("    circuit is RESISTIVE\n");
        end
    end
    fprintf("Reactance (resonant):\n");
    fprintf("    XL(at resonance) %f\n    XC(at resonance) %f\n", XL_res_, XC_res_);

    % Error checking
    if (n < 2)
        fprintf("\nErr, cannot run simulation. Number of steps too small...\n");
        return;
    end

    fprintf("Simulating %d steps...\n", n);
    fprintf("Time step (time differential) dt %f\n", dt_);

    % Simulation
    for i = 1 : n + 1
        if (i > 1)
            % uC
            duC(i) = duC_dt(i) .* dt;
            uC_(i) = uC_(i - 1) + duC(i - 1);
```

```
            % iL
            diL_(i) = diL_dt(i) .* dt;
            iL_(i) = iL_(i - 1) + diL_(i - 1);

            % uCe_
            uCe_(i) = uCe_(i - 1) + h_ .* duC_dt_e(i - 1);
            %uCe_(i) = uC_(i);

            % iLe_
            iLe_(i) = iLe_(i - 1) + h_ .* diL_dt_e(i - 1);

            % Power simulation
            if (mod(i, 2) == 1)
                y_0 = EMF_(i - 2) .* iLe_(i - 2);
                y_mid = EMF_(i - 1) .* iLe_(i - 1);
                y_1 = EMF_(i) .* iLe_(i);
                p = simpson_integral(y_0, y_mid, y_1, dt);
                if (i == 3)
                    p_((i-1) ./ 2) = p;
                else
                    p_((i-1) ./ 2) = p_((i-1) ./ 2 - 1) + p;
                end
            end
        end
    end

    power = p_(n_ ./ 2);

    % plot
    plot_as_one(t, EMF_, 'EMF', uC_, 'uC', uCe_, 'uC euler', iL_, 'iL', iLe_, 'iL euler');
    plot_one(p_time_, p_, 'Power', 'Time [s]', 'Watt [W]');
end


% Derivation

function [duC_dt] = duC_dt(i)
    global C_
    global iL_

    if (i == 1)
        duC_dt = 0;
    else
        duC_dt = (iL_(i - 1) ./ C_);
    end
end

function [diL_dt] = diL_dt(i)
    global R_
    global L_
    global iL_
    global uC_
    global EMF_

    if (i == 1)
        diL_dt = 0;
    else
        diL_dt = (EMF_(i - 1) - R_ .* iL_(i - 1) - uC_(i - 1)) ./ L_;
    end
end

function [duC_dt_e] = duC_dt_e(i)
```

```matlab
    global C_
    global iL_
    global h_

    if (i == 1)
        duC_dt_e = 0;
    else
        h05 = h_ ./ 2;
        didt = diL_dt(i);
        duC_dt_e = ((iL_(i - 1) + h05 .* didt) ./ C_);   % estimate uC_dt in half of dt_
    end
end

function [diL_dt_e] = diL_dt_e(i)
    global R_
    global L_
    global iLe_
    global uC_
    global EMF_
    global h_

    if (i == 1)
        diL_dt_e = 0;
    else
        h05 = h_ ./ 2;
        didt = (EMF_(i - 1) - R_ .* iLe_(i - 1) - uC_(i - 1)) ./ L_;
        diL_dt_e = (EMF_(i - 1) - R_ .* (iLe_(i - 1) + h05 .* didt) - uC_(i - 1)) ./ L_;
    end
end


% Integration

% Compute integral of y in the range h = x(y_1) - x(y_0) using Simpson's
% method
function [ret] = simpson_integral(y_0, y_mid, y_1, h)
    ret = (h ./ 3) * (y_0 + 4 .* y_mid + y_1);
end


% Plotting

function [] = plot_as_one(t, y1, s1, y2, s2, y3, s3, y4, s4, y5, s5)
    figure();
    xlabel('Time [s]');
    yyaxis left
    plot(t, y1, 'r', t, y2, 'g', t, y3, 'blue');
    ylabel('Potential difference [V]');

    yyaxis right
    plot(t, y4, 'black', t, y5, 'blue');
    ylabel('Current [A]');
    legend(s1, s2, s3, s4, s5);
end

function [] = plot_as_sub(t, y1, s1, y2, s2, y3, s3)
    figure();
    subplot(3,1,1), plot(t, y1,'r');
    legend(s1);
    subplot(3,1,2), plot(t, y2,'g');
    legend(s2);
    subplot(3,1,3), plot(t, y3,'b');
```

```
    legend(s3);
end

function [] = plot_one(t, y1, s1, x_desc, y_desc)
    figure();
    p1 = plot(t, y1, 'r');
    legend(p1, s1);
    xlabel(x_desc);
    ylabel(y_desc);
end
```

3.2 Wykres

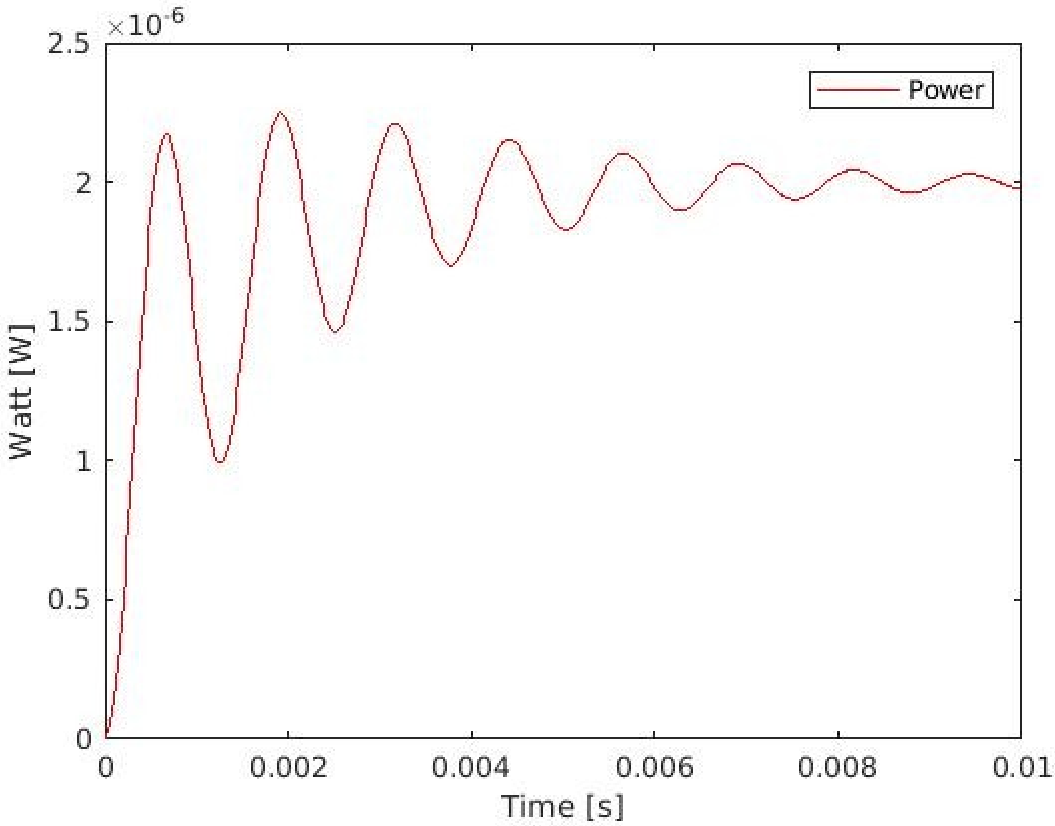

Illustration 7: Obliczanie mocy obwodu metodą złożonych parabol (Simpsona)