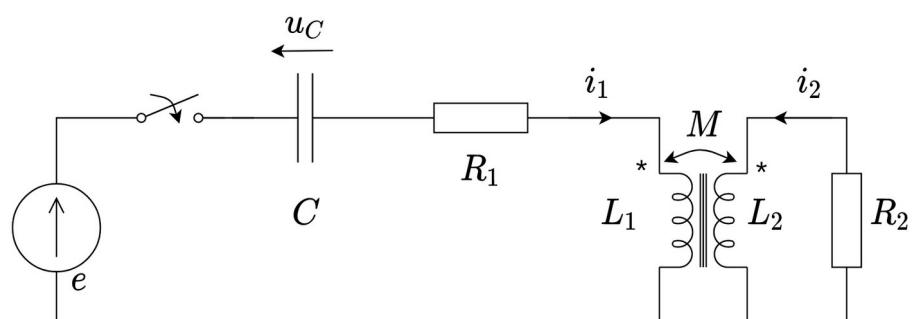


Projekt zaliczeniowy

Piotr Heinzelman, alb. 146703

Symulator parametrów obwodu ze sprzężeniem indukcyjnym



1. Symulator - omówienie kodu

Kod symulatora jest napisany tak aby był reużywalny. Za wartości ustawień parametrów symulatora odpowiada osobna funkcja - config, oraz dla zwiększenia czytelności kilka funkcji typu *setter*, umieszczone w pakiecie config.

```
% // config
function config( type , line )
global CFG
  if ( type>0 )
    CFG(:,type)=line';
  else
    CFG = [ 0, 0, 0 ]'; % start params ( y1=i1, y2=i2 y3=uc )
config( 2, [ 0, 0.0001, 30 ]); % h params
config( 3, [ 0.1, 10, 0.5 ]); % system params: 3 - R1 R2 C
config( 4, [ 3, 5, 0 ]); % system params: 4 - L1 L2 M(not used!)
config( 5, [ 1, 0, 0 ]); % Euler mode: 0 - normal, 1 - extended
config( 6, [ 0, 100, 1 ]); % UGeneratorType: "const." , "sinus" , "rectangle"
config( 7, [ 0, 0.8, 0 ]); % MjMode: "const" [0,value], [1..]-Vander [2]-Sklejane 1st
config( 8, [ 0, 0, 0 ]); % images number, section
  UGen();
  set_Mj( (CFG(1,7)) );
end
end
```

przykładowy setter:

```
function setSection( sec )
global CFG
  config( 8, [ 0, sec, 0 ]);
end
```

Pakiet zamyka funkcja usługowa plt() której zadaniem jest przechwytywanie, opisywanie wykresów, nadawanie kolejnych nazw i zapisywanie w odpowiednim folderze, do dalszego wykorzystania. Funkcja ta nie jest optymalizowana, jest rozwlekła, ale czytelna. Jej wydajność nie ma znaczenia.

```
% // plt.m
function plt()
global CFG
  imageNumber = CFG(1,8);
  imageNumber = imageNumber+1;
  CFG(1,8) = imageNumber;
  imageSection= CFG(2,8);

  t = CFG(1,2):CFG(2,2):CFG(3,2);
  Y = Euler( t );

  hold off;
  tl = tiledlayout("flow");

  nexttile;
  plot(t,Y,"-");

  info="";
  if ((CFG(1,5))<2)
```

```

legend("i1","i2","Uc");
if (CFG(1,6))==0
    info = "u(t)="+CFG(2,6)+"*sin(2πft); f="+ceil((CFG(3,6))/(2*3.13));
elseif (CFG(1,6))==-1
    info = "u(t)="+CFG(2,6)+ "[V]";
else
    info = "u(t)=prostokat";
end
end
if CFG(1,5)==0
    eul="normalna";
else
    eul="ulepszona";
end
if CFG(1,7)==0
    mjname="stała 0.8H";
elseif CFG(1,7)==1
    mjname="interpolacja wielom.";
elseif CFG(1,7)==2
    mjname="f-sklejana 3st.";
elseif CFG(1,7)==3
    mjname="aproks. wielomian 3st";
elseif CFG(1,7)==4
    mjname="aproks. wielomian 5st";
else
    mjname="modyfikowana funkcja";
end

title(info + ", Eul: " + eul + ", Mj: " + mjname);
exportgraphics(tl, "INDD/links/" + imageNumber + "_" + imageSection + "_img_.png", 'Resolution',600);
clear plt
end

```

1.1 Symulator - przygotowanie do pracy i uruchomienie.

Zmienne środowiska, ścieżka przeszukiwania, importy i załadowanie do pamięci domyślnych wartości dzieją się po uruchomieniu funkcji prepare:

```
function prepare()
```

```
addpath 'config'  
addpath 'mjs'  
addpath 'ugen'  
addpath 'Euler'  
addpath 'power'
```

```
import config.*
```

```
import mjs.*
```

```
import ugen.*
```

```
import Euler.*
```

```
import power.*
```

```
global CFG uGen
```

```
config(-1,[]);
```

```
info=" ** Start **"
```

Właściwie niewiele się w niej dzieje, wykonanie funkcji `config(-1,[])`; ładuje domyślne parametry.

Użycie symulatora, generowanie wykresów i wypisanie wyników obliczenia mocy przebiega następująco:

```
% Euler = "normal","extend"  
% u(t) = setUSin( 250 , 2*3.14*5 ) , setUProst(Vmax), setUConst(Vmax)  
% Mj = "const",...  
prepare(); %załadowanie ustawień  
  
setSection(0);  
% Część 1  
if (false)  
setSection(1);  
setEuler("normal");  
setMj( "const" );  
  
setUConst(110);  
plt();  
c = bisekcja(0.0, 0.1 ,1); info="bisekcja: 0.0 - 0.1"+ c
```

setSection - ustawia fragment nazwy oddzielający wykresy z kolejnych sekcji, setEuler("normal") - ustawia tryb całkowania numerycznego metodą Eulera zwykły, lub poprawiony "extend".

Kolejny setMj("const") - ustawia implementację funkcji Mj na funkcję zwracającą wartość stałą 0.8. W pakiecie mjs zdefiniowanych jest kilka funkcji obliczających wartość współczynnika sprężenia w zależności od przekazanego parametru U1 - czyli napięcia na cewce. Mamy do wyboru kilka implementacji poza trywialną $y=0.8$ i są to:

- interpolacja wielomianowa (Newtona)
- interpolacja funkcjami sklejonymi
- aproksymacja wielomianowa stopnia 3
- aproksymacja wielomianowa stopnia 5

wywołanie setUConst(110); ustawia wymuszenie na 110[V] napięcia stałego, a funkcja plt(); wykresła wynik dla danych: Euler zwykły, wymuszenie 110V, Mj=0.8[H].

Uruchomienie kodu **bisekcja(0.0, 0.1, 1)** spowoduje obliczenie miejsca zerowego funkcji Pow(F) zwracającej moc pomniejszoną o 406[W]. Moc wyzwalaną na rezystancjach oblicza funkcja Power() dla identycznie ustawionych parametrów układu. Częstotliwość można ustawić setterem setUSin(*napięcie*, 2*3.14*50), wpisując napięcie Um oraz pulsację - może być w formacie 2*3.14*5.

Wymuszenia prądowe również są konfigurowalne i możemy ustawić 3 rodzaje:

- napięcie stałe - setUConst(Vmax)
- napięcie sinusoidalne o zadanej pulsacji - setUSin(Vmax, Omega)
- napięcie prostokątne - setUProst(Vmax)

Zmiana rodzaju dzieje się automatycznie po użyciu odpowiedniego settora.

Podsumowując należy ustawić parametry układu, a następnie uruchomić rysowanie wykresów lub obliczanie mocy bądź miejsc zerowych zależnie od konfiguracji.

Obliczenia zależą od ustawionej implementacji funkcji liczących.

Poniżej wyszczególnię implementacje funkcji wymuszeń u(t), umieściłem je w jednym pliku. Jednak warto zauważyć implementację wymuszenia prostokątnego bez użycia konstrukcji IF.

```
function UGen()
    global CFG uGen
    row=CFG(:,6);
    h=CFG(2,2);
    if      ( row(1)== -1 ) uGen = @(t) row(2);
    elseif ( row(1)==  0 ) uGen = @(t) (row(2))*sin(t*row(3));
    else    uGen = @(t) (bitand(round(1023*sin(((t-.714)*3.1415)/3)+0)^2),512))*(row(2)/512);
    end
end
```

1.2 Symulator

Funkcja obliczająca kolejną wartość funkcji przy wykorzystaniu poprzedniej wartości funkcji metodą Eulera służy nam do całkowania metodą numeryczną funkcji. Mówiąc dokładniej, obliczamy wektor stanu układu, otrzymując jako parametr poprzedni wektor stanu układu. Pierwszy wektor to wektor wartości początkowych, kolejne wyliczamy i przechowujemy w tablicy wektorów (jak powie programista) lub w wektorze wektorów. Mając zbiór takich wektorów, dodatkowo łącząc je z wartościami czasu, możemy policzyć całkę (pole pod krzywą) lub narysować przebieg krzywej (bez liczenia całki - wprost kreśląc kolejne wartości wektorów).

poniżej kod symulatora:

```
function Y = Euler ( T )
    import mjs.*;
    global CFG uGen Mj MjUL D1 D2 e1
    Y=CFG(:,1);

R1=CFG(1,3);           % R1=0.1;
R2=CFG(2,3);           % R2=10;
C=CFG(3,3);            % C=0.5;
L1=CFG(1,4);           % L1=3;
L2=CFG(2,4);           % L2=5;

Emode=CFG(1,5);
h=CFG(2,2);

for i=1:length(T)-1
    Y(:, i+1) = stepEuler( T(i) , Y(:, i) );
end

function dY = stepEuler( t , Y ) % y1=i1   y2=i2   y3=uc
    i1=Y(1);
    i2=Y(2);
    Uc=Y(3);
    e1=uGen(t);
    UL=e1-Uc-(i1*R1); %UL=e-Uc-i1*R z prawa napięciowego Kirhoffa
    MjUL=Mj(UL);

    D1=(L1/MjUL)-(MjUL/L2);
    D2=(MjUL/L1)-(L2/MjUL);

% całkowanie Eulera zwykłe i ulepszone, zależnie od parametru
    if (Emode==1) % Yn+1=Y+hf( X+h/2 , Y+h/2*(f(X,Y)) ) % ulepszone
        dY = [ ( i1 + h*fdy1( t+(h/2) , Y+(h/2)*fdy1(t,Y) ) )
                ( i2 + h*fdy2( t+(h/2) , Y+(h/2)*fdy2(t,Y) ) )
                ( Uc + h*fdy3( t+(h/2) , Y+(h/2)*fdy3(t,Y) ) ) ];
    else % (Emode==0) % Yn+1=Y+hf(X) % Zwykłe
        dY = [ ( i1 + h*fdy1(t,Y) )
                ( i2 + h*fdy2(t,Y) )
                ( Uc + h*fdy3(t,Y) ) ];
    end
end
```

```
% odseparowane poszczególne obliczenia
function dy1 = fdy1(t,Y)
    i1=Y(1);
    i2=Y(2);
    Uc=Y(3);

    dy1 = -i1*(R1/(MjUL*D1)) + i2*(R2/(L2*D1)) - Uc/(MjUL*D1) + e1/(MjUL*D1);
end

function dy2 = fdy2(t,Y)
    i1=Y(1);
    i2=Y(2);
    Uc=Y(3);

    dy2 = -i1*(R1/(L1*D2)) +i2*(R2/(MjUL*D2)) - Uc/(L1*D2) + e1/(L1*D2);
end

function dy3 = fdy3(t,Y)
    i1=Y(1);
    i2=Y(2);
    Uc=Y(3);

    dy3 = i1*(1/C);
end
end
```

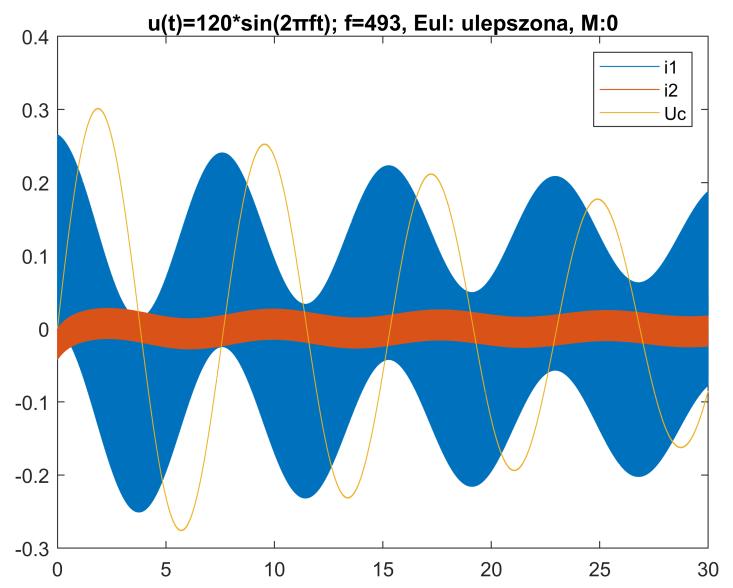
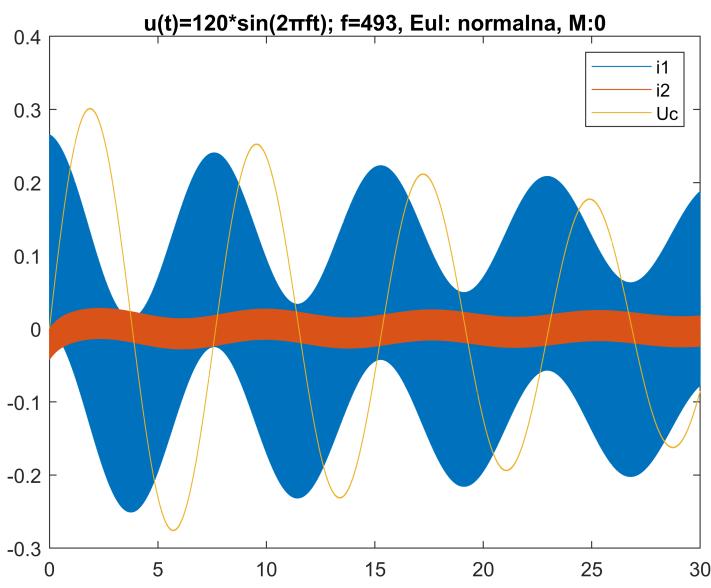
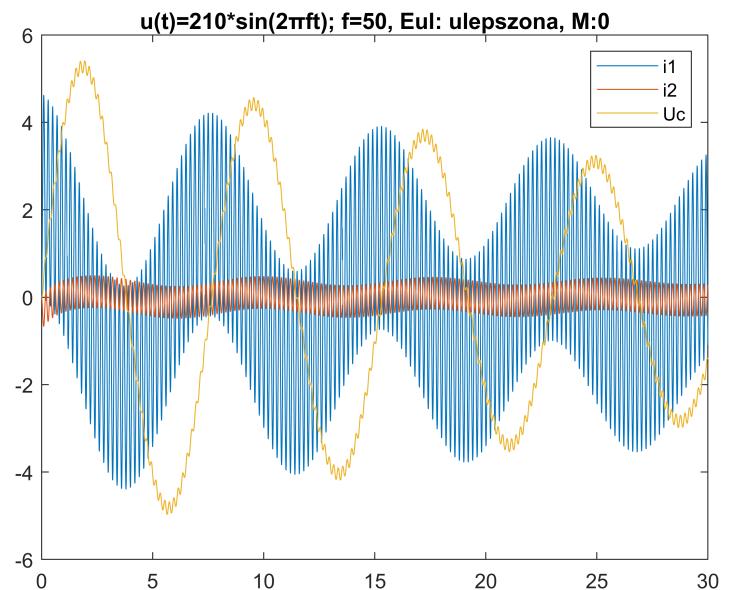
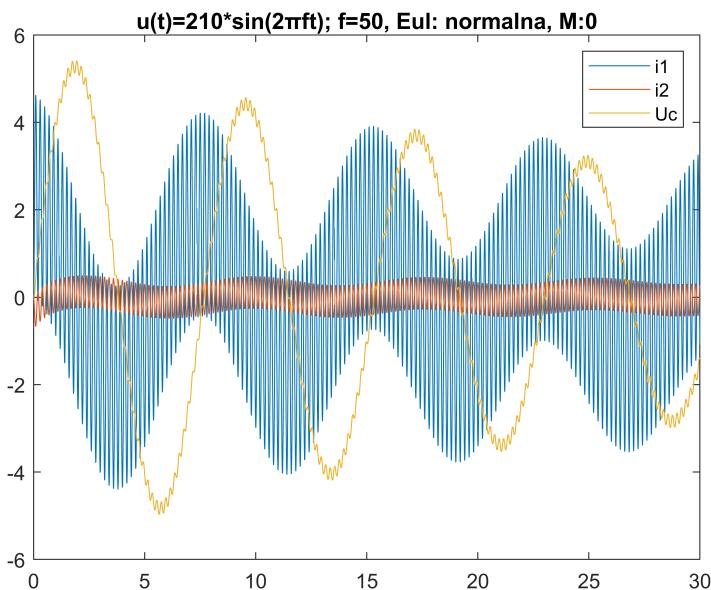
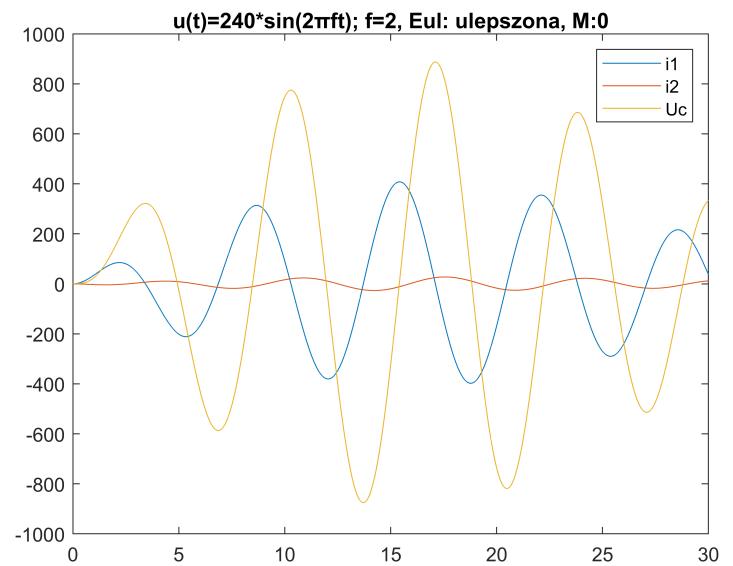
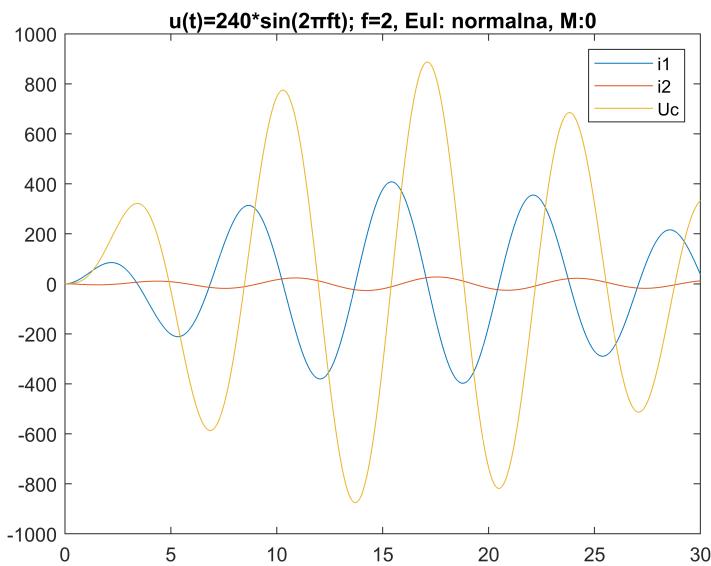
|Funkcję starałem się zoptymalizować choć trochę, stąd takie wywołanie -

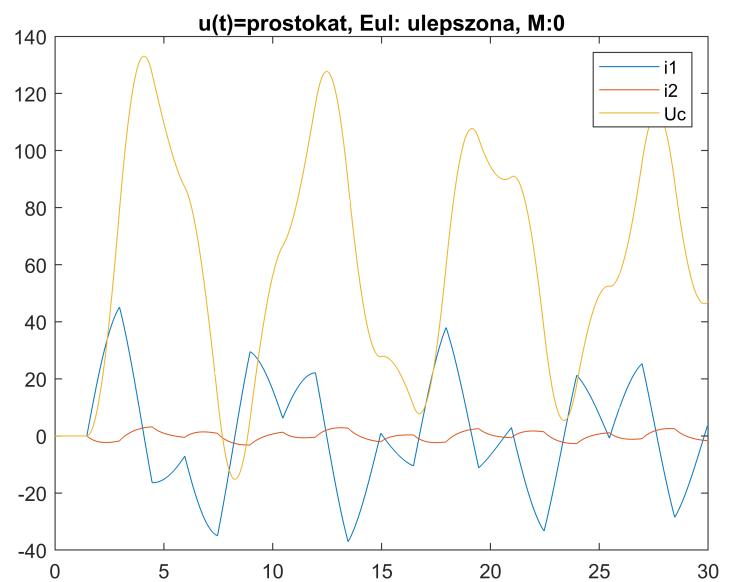
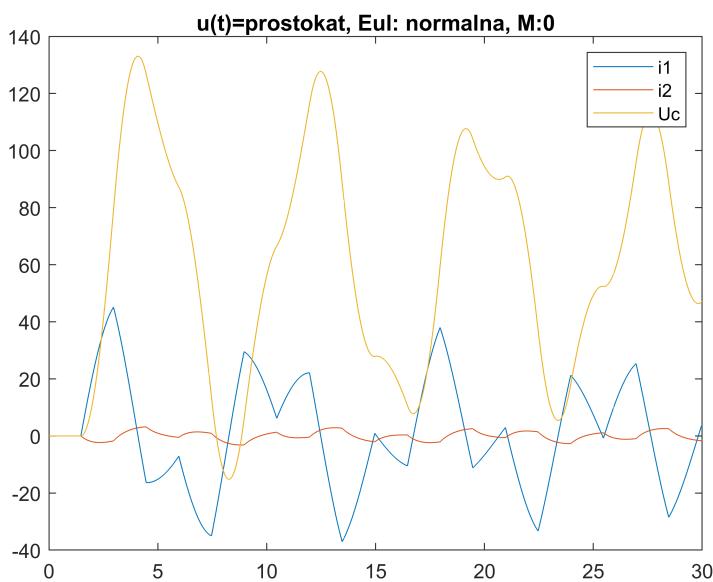
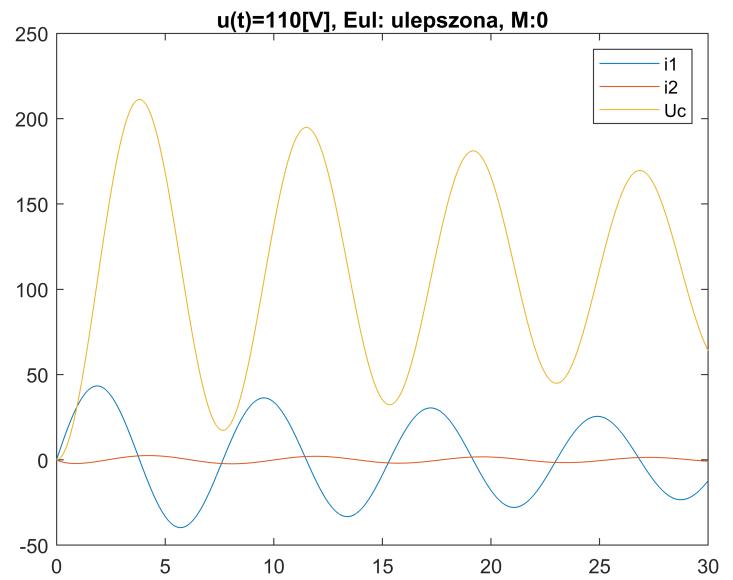
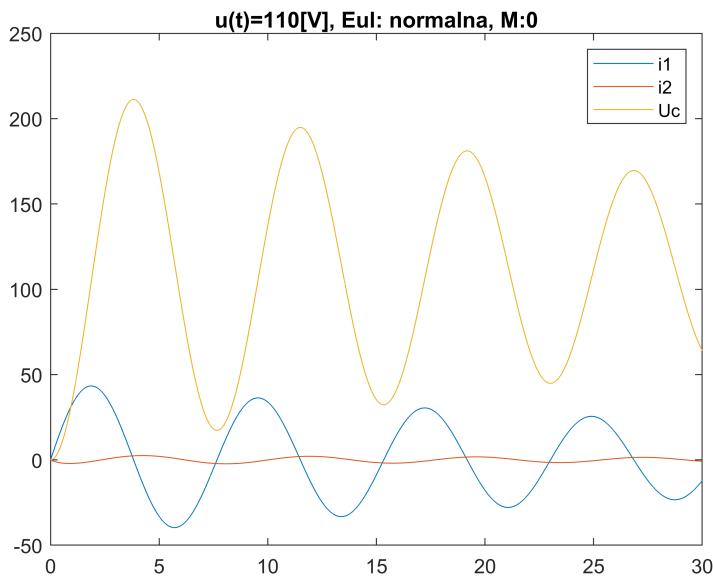
```
for i=1:length(T)-1
    Y(:, i+1) = stepEuler( T(i) , Y(:, i) );
end
```

- by wielokrotnie nie inicjalizować wartości, które można zinicjalizować raz.

Kod jest tak podzielony i tak są nazwane zmienne, że zgodnie z zasadami czystego kodu - intencje i formuły są czytelne. Na początku wprowadziłem funkcję Mj(UL) dodając implementację 0.8[H], żeby w drugiej części projektu nie zmieniać kodu, a ewentualnie coś dodać.

poniżej wykresy wygenerowane za pomocą symulatora:





Na pierwszy rzut oka trudno znaleźć różnice między metodą zwykłą a ulepszoną. Ciężko też byłoby powiedzieć która zwraca wyniki lepsze, jeśli nie bardzo się wie, jaką dokładnie powinna być funkcja wynikowa. Tu akurat można podejrzewać, że wyniki są przynajmniej zbliżone do oczekiwanych. Przy wymuszeniu prostokątnym można zauważać na wykresie cykle ładowania i rozładowywania. Przy wymuszeniu częstotliwością 1Hz napięcia sięgają 800[V], przy wyższych częstotliwościach spadają do 0.3[V]. Odpowiedź układu stabilizuje się po około 50 sekundach. Kształty przebiegów widoczne na powyższych wykresach przypominają te z podręcznika "Teoria obwodów" Ossowskiego, Siwka i Śmiałka. Można więc założyć, że symulator wykonuje swoje zadanie jak dotąd w miarę poprawnie.

2. Interpolacje i aproksymacje Mj(UL)

Aby wyliczyć wartość indukcyjności wzajemnej, używamy wybranej implementacji funkcji, która zawiera wyliczone wartości wielomianów lub wektorów, by nie liczyć tego za każdym razem, niemniej jednak funkcje służące tym obliczeniom zostały także umieszczone w kodzie.

Poniżej podam implementację oraz wykresy charakterystyk funkcji Mj w wersji skróconej.

```
% "const" - funkcja stała
function y = Mj0(u)
    y=.8;
end

% "inter"- interpolacja wielomianowa
function y = Mj1(u)
y=0.337534516765287+u*(0.00545792899408279+u*(5.21966907736125e-05+u*(-1.05465592811740e-06+u*(5.75208196361954e-09+u*(-1.78522463291645e-11+u*(3.61385053692621e-14+u*(-3.46263423186378e-17))))));
end

% "sklejana" funkcja sklejana wielomianem 3 stopnia
% aby w implementacji nie liczyć tego przy każdym użyciu, wstawiam wartości
% HARD CODE

function y = Mj2(u)
%C = obliczC()
C=[0.0030      0.0511      0.1129      0.1374      0.1175      0.0726      0.0321      0.0290];

x=u;
a=0;
h=50;
h_=1/(h*h*h);
sum=C(1);

for i=1:8 % i=czesc
    sum=sum+(fun(i,x)*C(i));
end
y=sum;

function value=fun(i,x)
X=[-50,0,50,100,150,200,250,300,350];
xi=X(i);
h=50;
xi_2=xi-h-h;
xi_1=xi-h;
xi=xi;
xi1=xi+h;
xi2=xi+h+h;

    if ( x>=xi2 ) value = 0;
    elseif ( x>=xi1 ) value = (xi2-x)^3;
    elseif ( x>=xi ) value = (xi2-x)^3-4*(xi1-x)^3;
    elseif ( x>=xi_1 ) value = (-xi_2+x)^3-4*(-xi_1+x)^3;
    elseif ( x>=xi_2 ) value = (-xi_2+x)^3;
    else value=0;
end
value=value*(1/(h*h*h));

end
end
```

```

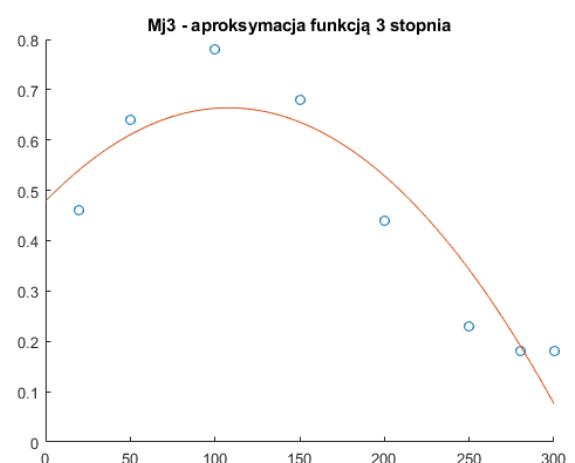
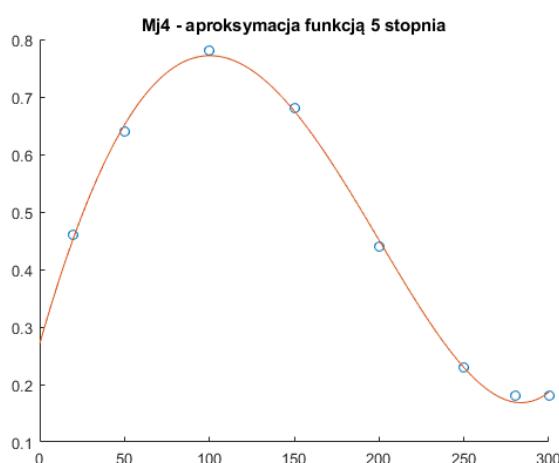
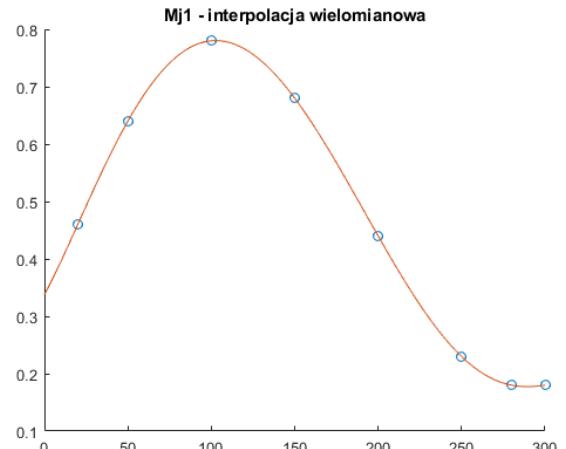
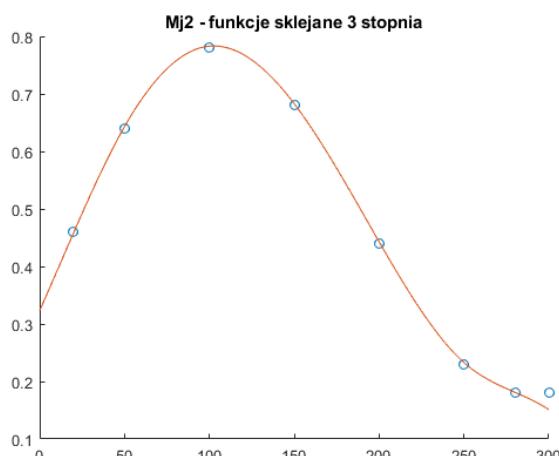
% "ap3st" aproksymacja wielomianem 3 st.
function y = Mj3(u)
    % ObliczA();
    y=0.478524285255746+u*(0.00343707585345168+u*(-1.59263685413181e-05));
end

% "ap3st" aproksymacja wielomianem 5 st.
function y = Mj4(u)
    %ObliczA();
    y=0.270529360498919+u*(0.0102965513204268+u*(-5.28805197847902e-05+u*(-2.89019332648018e-08+u*(2.92192106012333e-10)));
end

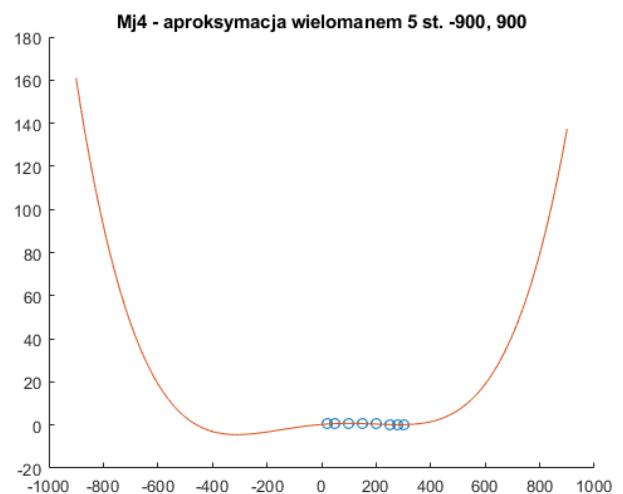
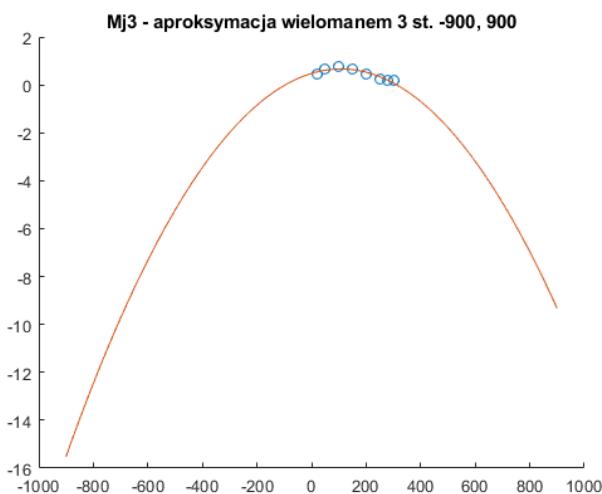
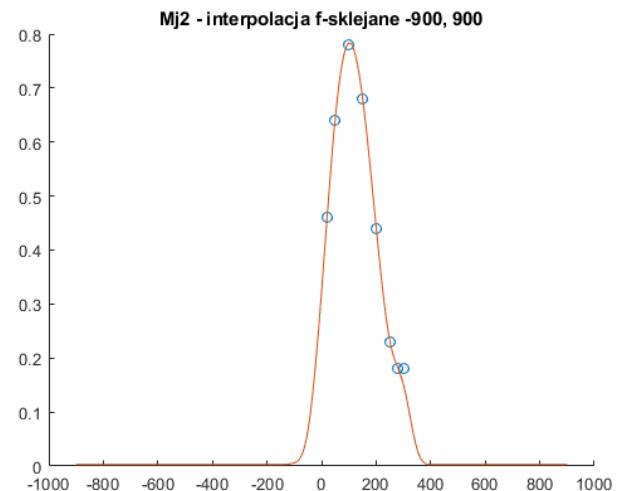
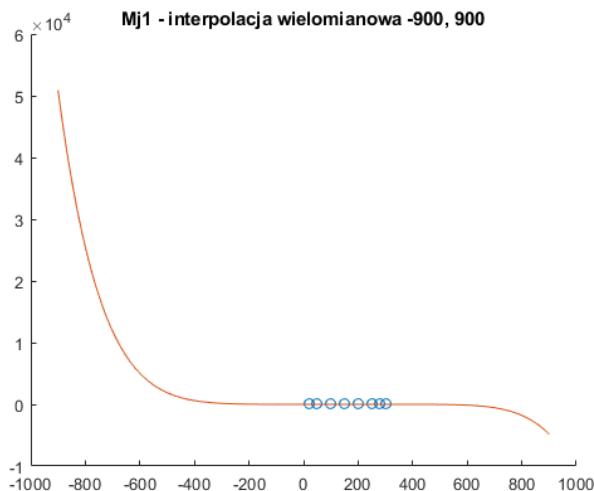
% funkcja eksperimentalna - sklejana / aproksymowana wielomianami 3 stopnia
function y = Mj5(u)
    multi=1; if (u<0) u=-u; multi=-1; end
    if (u>400) y=0.18;
    elseif (u>280)
        y0=0.171013;y1=0.18;
        y=y0+(y1-y0)*((u-280)/(400-280));
    else
        y=0.270529360498919+u*(0.0102965513204268+u*(-5.28805197847902e-05+u*(-2.8901933264...8018e-08+u*(2.92192106012333e-10)));
    end
    y=y*multi;
end

```

oraz wykresy tych metod:

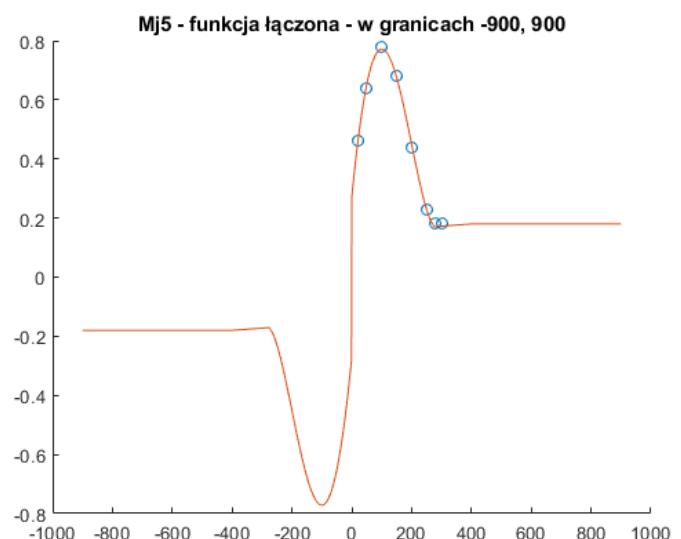


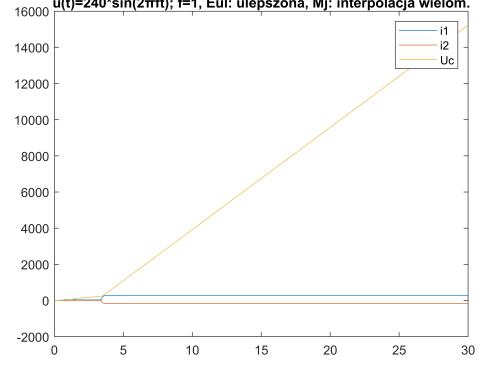
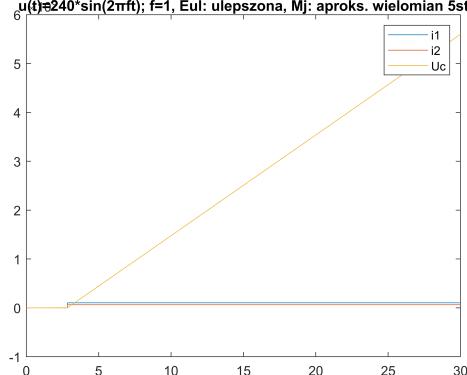
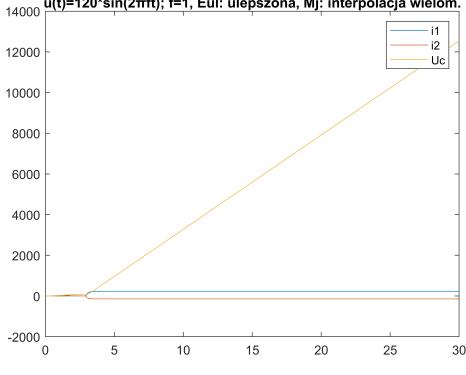
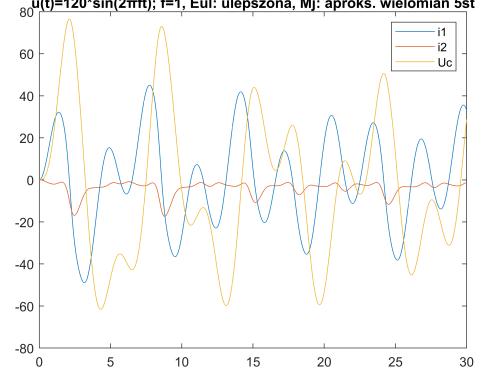
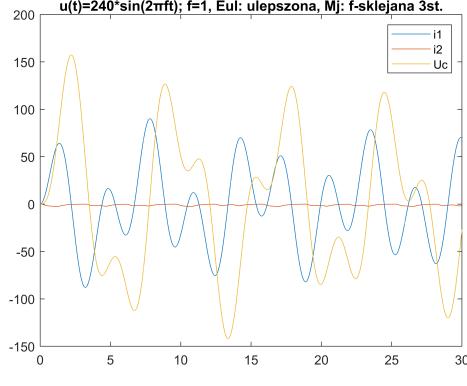
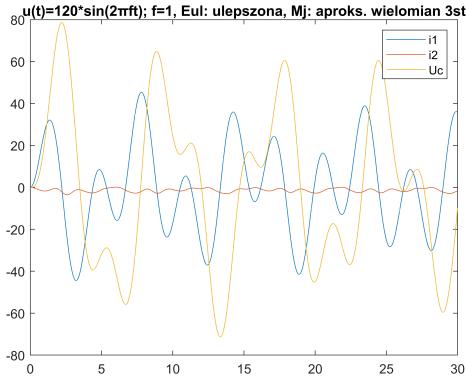
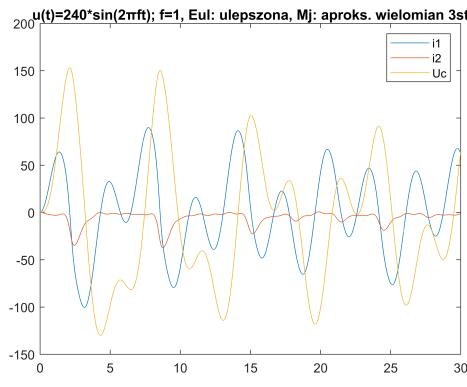
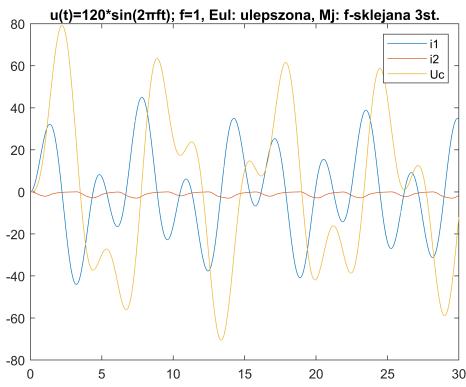
Wszystko wygląda bardzo ładnie ale przy większych zakresach wartości UL dzieją się rzeczy zaskakujące:



Funkcje sklejane mają dziwne wartości już przy napięciach -100 V i 300 V, interpolacje wielomianowe poniżej -400 V i 600 V. Aproksymacje wielomianowe odbiegają poniżej -200 V i ponad 400 V, a w naszej symulacji pojawiają się napięcia chwilowe do 900 V.

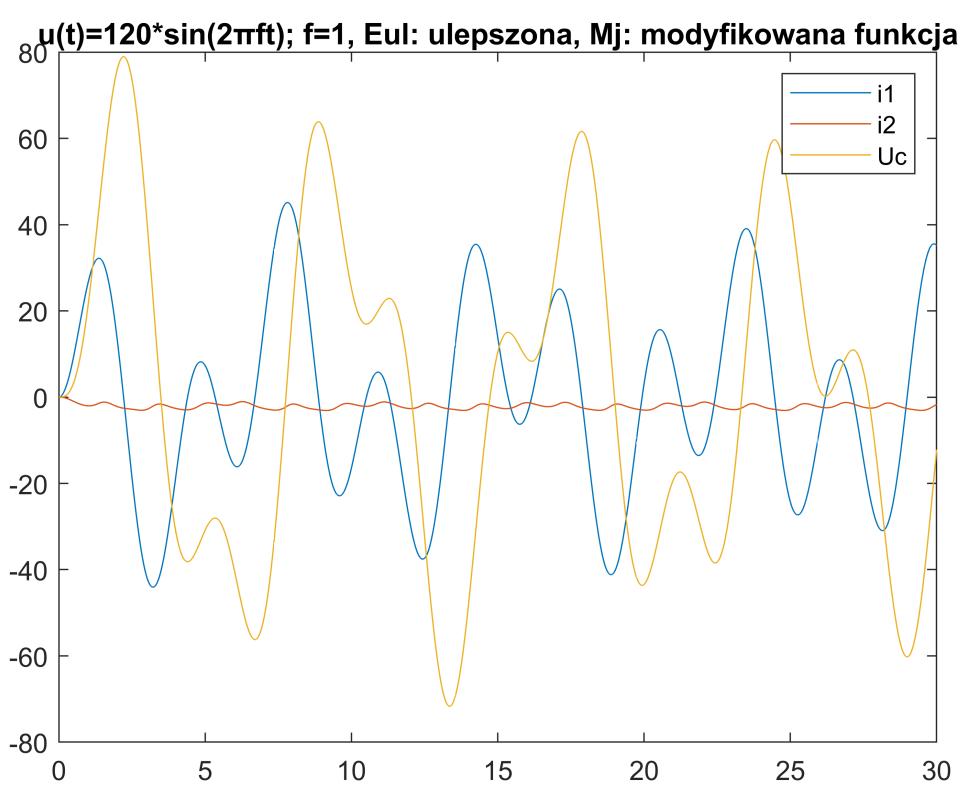
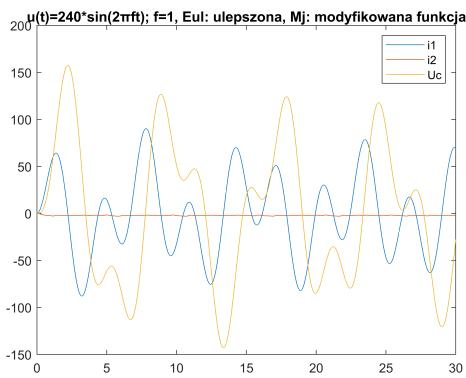
Poniżej - nieco poprawiona funkcja Mj(Ul), która nie ucieka do nieskończoności, ponadto jest nieparzysta, czyli $-f(x) = -f(-x)$, co zdaje się mieć sens: w momencie zmiany kierunku prądu cewki działają na siebie "z przeciwnym znakiem".





Wyniki pomiarów dla różnych implementacji funkcji Mj(Ul) - odpowiedzi układu i_2 widoczne na wykresach nie przypominają kształtu sinusoidy.

Użycie poprawionej wersji funkcji pozwala osiągnąć bardziej sinusoidalny wykres prądu i_2 .



3. Moc

Implementacje funkcji liczących moc, gdzie parametrem wysłanym do funkcji jest zbiór wektorów stanu układu.

funkcja licząca:

```
function pow = Power(Y, type)
global CFG

R1=CFG(1,3);
R2=CFG(2,3);
h=CFG(2,2);
s1=0;
s2=0;

for i=1:length(Y)
    Y(3,i)=Y(1,i)^2;
    Y(4,i)=Y(2,i)^2;
end
t = CFG(1,2):CFG(2,2):CFG(3,2);

if ( type==1 )
    s1 = calkaProstokat ( Y(3,:) );
    s2 = calkaProstokat ( Y(4,:) );
    pow=(s1*R1+s2*R2)*h;
else
    s3 = calkaParabol ( Y(3,:) );
    s4 = calkaParabol ( Y(4,:) );
    pow=(s3*R1+s4*R2)*h;
end

function value=calkaProstokat ( Y )
S=0;
for k=1:length(Y)-1
    S=S+Y(k);
end
value=S;
end

function value=calkaParabol ( Y )
S2=0;
len=length(Y);
le=len/2;
for j=1:le-1
    S2=S2+(( Y(2*j) + 4*(Y(1+2*j)) + Y(2+2*j))/3);
end
value=S2;
end

end
```

funkcja licząca moc w zależności od częstotliwości.

```
function P = Pow(Fi)
    global CFG
    t = CFG(1,2):CFG(2,2):CFG(3,2);
    config( 6, [ 0, 100, 2*3.15*Fi ]);
    UGen();
    Y = Euler( t );
    po=Power(Y,0);
    info="100*sin(2πwt) f="+Fi+" Power = "+po+": po-406="+(po-406)+" .";
    P=po-406;
end
```

4. Funkcje obliczające miejsca zerowe funkcji Pow() bisekcja, sieczne i Norton

Numeryczne poszukiwanie miejsc zerowych sprowadza się, podobnie jak całkowanie, do wykonywania kolejnych cykli obliczeń, w których parametrami wejściowymi są wyniki poprzedniego cyklu. Krocąc w ten sposób, zbliżamy się coraz bardziej do rozwiązania (o ile funkcja jest zbieżna, a w większości metod tak jest). Po kilku cyklach otrzymamy wystarczająco dokładne przybliżenie wyniku.

```
% bisekcja
function c = bisekcja(a,b,level)
    level=level+1;
    Pa=Pow(a);
    c=(a+b)/2;
    Pc=Pow(c);
    if (Pa*Pc>0)
        a=c;
    else
        b=c;
    end

    if (level<15)
        c = bisekcja(a,b,level);
    end
end
```

```

function c = Sieczne(x0,x1)

rot=1;
%deltaX=1e-4;
%poniewaz 2 pochodna i funkcja mają znak ten sam (ujemny)
po prawej stronie pierwiastka obieram: x0=.7 x1=1
%FPx0=Fprim(x0)
%FPx1=Fprim(x1);
% FPPx0=Fprimprim(x0)
% FPPx1=Fprimprim(x1)
% info="x0: " + x0 + ", F(x0): " + Fx0 + ", x1: " + x1 +
", F(x1): " + Fx1 + ", F'(x0): "+FPx0+", F''(x0): "+FPPx0 +",
F'(x1): "+FPx1+", F''(x1): "+FPPx1

Fx0=Fx(x0);
for j=1:7
    Fx1=Fx(x1);
    dx=Fx1*(x1-x0)/(Fx1-Fx0);
    x2=x1-dx;
    info = "x0: " + x0 + ":" + Fx0
+ ", x1: " + x1 + ":" + Fx1 + ", x2: " + x2 + ",
Fx(x2) " + Fx(x2) + "obliczen: " + rot
    Fx0=Fx(x1);
    x0=x1;
    x1=x2;
end
c=x2;

function y=Fx(x)
rot=rot+1;
t=0:1e-4:30;
config( 6, [ 0, 100, 2*3.15*x ]);
UGen();
Y = Euler( t );
y=Power(Y,0);
y=y;
end

function y=Fprimprim(x)
x_0=Fprim(x);
x_1=Fprim(x+deltaX);
y=(x_1-x_0)/deltaX;
end

function y=Fprim(x)
x__0=Fx(x);
x__1=Fx(x+deltaX);
y=(x__1-x__0)/deltaX;
end

end

function c = Newton(F0)

deltaX=1e-4;
t=0:deltaX:30;

% roznica:0.0051098 : deltaX6.1035e-05 % obliczam deltaX dla
ktorego pochodna deltaX/2 rozni sie maksymalnie o 1%
if (false)
x=1/4; deltaX=1; t=0:1e-5:30;
yPx(x,t); yPrim_minus1=0;
for i=1:40
    deltaX=deltaX/2;
    yNext=Px(x+deltaX,t );
    yPrim=(yNext-y)/deltaX;
    roznica=( yPrim_minus1-yPrim )/yPrim;
    y=yNext;
    a=roznica
if (roznica*roznica<0.01*0.01)
info = "roznica:" + roznica + " : deltaX" + deltaX
    end
    yPrim_minus1=yPrim;
end
i=0;
for i=0:15
    x=F0;
    Pi=Px(x ,t );
    Fi=Pi-406;
    Pprimi=Px(x+deltaX , t );
    Fprimi=Pprimi-406;
    dFSdf =(Fprimi-Fi)/deltaX;

    F0=F0-Fi/dFSdf;
    i=i+1;
    info="i: " + i + "F0: " + F0 + ", Fi: " +Fi + "
Pi: " + Pi
    c=F0
end

function y=Px(x ,t )
config( 6, [ 0, 100, 2*3.15*x ]);
UGen();
Y = Euler( t );
y=Power(Y,0);
end

% F(x), F'(x), F''(x)
%function yPrimPrim = PIx(y , x)
%    yPrimPrim = PIx(y, x)
%end

function yPrim = PIx(Px0Dx, x)
%y=Pow(x);
yPrim=(Px(x+deltaX)-Px0Dx)/deltaX;
end

```

Wyniki obliczeń całkowania mocy:

Wymuszenie	metoda prostokątów		metoda parabol	
	0.5[s]	00001[s]	0.5[s]	00001[s]
e(t) = 1[V]	4586	2.9527e-12	4864	3.0282e-12
prost. 120V	26672255	9.8564e-09	27864834	1.0093e-08
240sin(t)	1311465145	8.9143e-10	1355003286	9.0219e-10
210sin(2*3*5t)	352	1.9454e-12	431	1.9759e-12
210sin(2*3*50t)	2170615	1.1078e-08	2496659	1.1131e-08

Metoda	Wartość rozwiązania f	wartość funkcji f	Liczba iteracji	Liczba obliczeń
bisekcji	0.035284, 0.67313	-3.6402e-08	35	70
siecznych	0.15089, 6.6157	3.4330	2	7
Newtona	0.035284, 0.67312	5.2895e+04	15	30