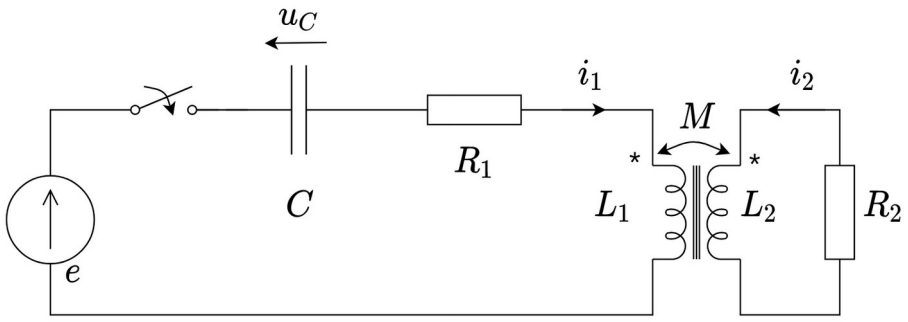
Projekt zaliczeniowy

Piotr Heinzelman, alb. 146703

Symulator parametrów obwodowych obwodu ze sprzężeniem indukcyjnym



1. Struktura programu

1.1 moduł config.m

```
%config.m
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,   1e-3,  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, [   3,     0,    0   ]); %     Euler mode: 0: normal, 1:extended, 2:power test, 3:Mu(Uc) test
    config( 6, [   0,    240,   1   ]); % UGeneratorType: "const."[-1,V,-], "sinus"[0,V,*2πf],"rect":[ 2,Vmax,-]
    config( 7, [   3,    0.8,   0   ]); % MjMode: "const" [0,value,-], [1..]-Vander [2]-Sklejane
                                          %   [3]-apr. wielomian 3st [4]-apr. wielomian 5 stopnia
        UGen ();
      MuBuild();
    end
end
```

funkcja umożliwia proste i usystematyzowane ustawianie parametrów programu,
takich jak:
- parametrów początkowych
- zakresu i wielkości stałej h
- parametrów układu
- rodzaj źródła napięcia (wymuszenia) oraz parametry tego źródła np. U_{max}
- rodzaj (i ewentualnie parametry) funkcji wartości indukcyjności M(U)
- wyboru pomiędzy metodą Eulera zwykła i ulepszoną

(a także dwoma dodatkowymi: - obrazującymi a) przebieg napięcia w czasie, b) charakterystykę funkcji M(u))

wywołana bez parametrów ustawi domyślne wpisane w kodzie wartości.

Ustawione wartości są dostępne przez globalną zmienną CFG która jest tablicą wartości.

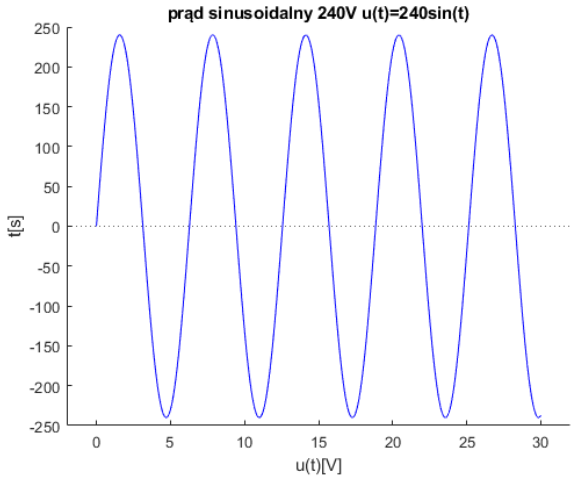
moduł UGen.m

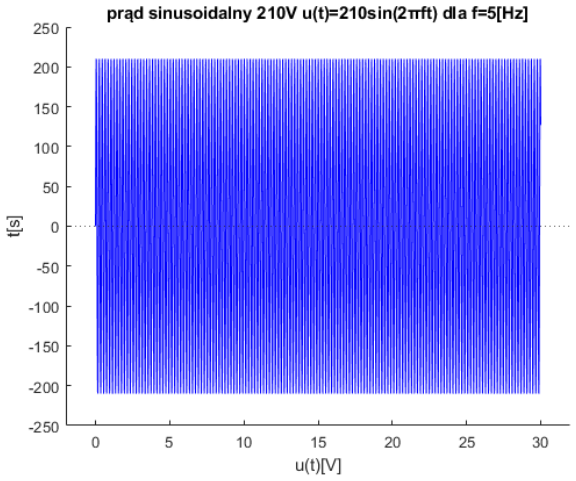
```
% UGen.m
function UGen()
    global CFG uGen
    row=CFG(:,6);
    h=CFG(2,2);
    if     ( row(1)== -1 ) uGen = @(t)  row(2);
    elseif ( row(1)==  0 ) uGen = @(t) (row(2))*sin(t*row(3));
    else   uGen = @(t) (bitand(round(1023*sin((((t-.714)*3.1415)/3)+0)^2),512))*(row(2)/512);
    end
end
```

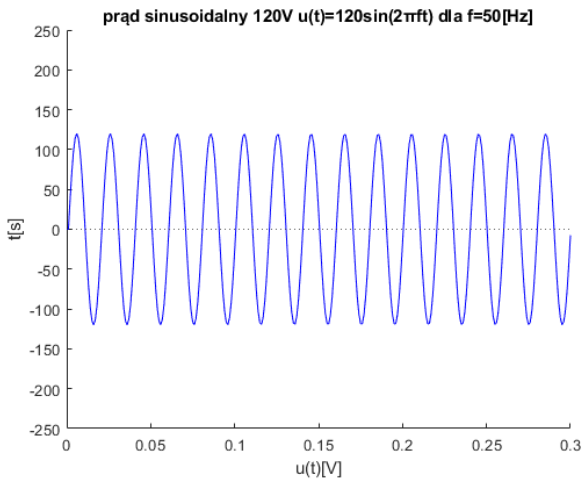
funkcja UGen realizuje INTERFEJS generatora wymuszania. "Interfejs" w rozumieniu języka Java, umożliwia przypisanie różnych implementacji tej samej zmiennej.

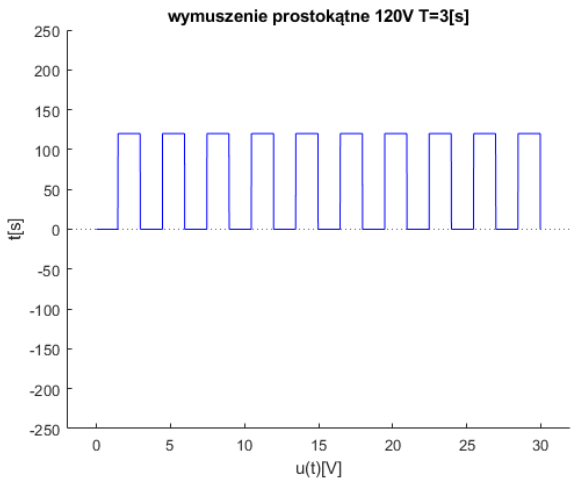
funkcja pobiera konfigurację globalną i na jej podstawie wybiera rodzaj wymuszenia i ewentualnie przypisuje potrzebne wartości takie jak wielkość napięcia,wartość pulsacji lub długość okresu przy wymuszeniu prostokątnym.

modyfikując konfigurację programu można łatwo uzyskać charakterystyki napięć w czasie.





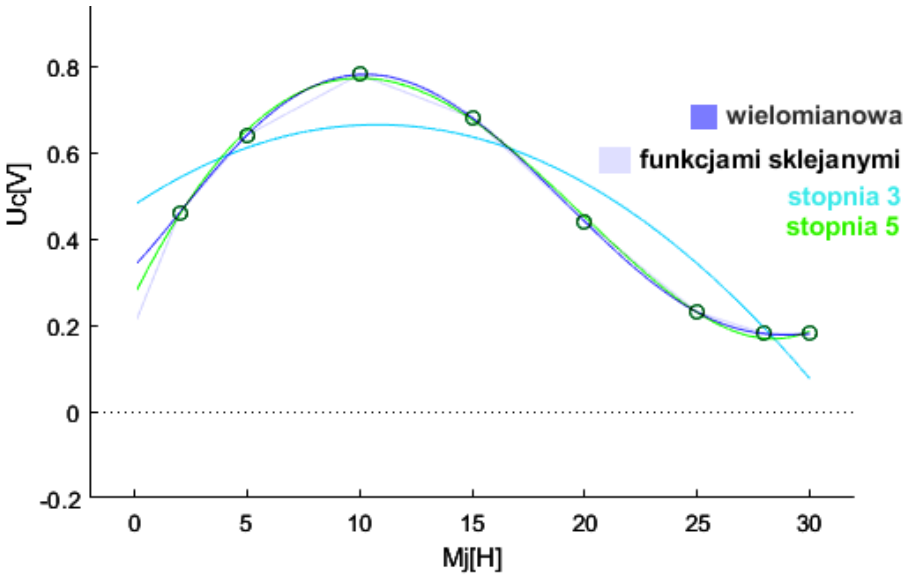




moduł MuBuild.m

```
%MuBuild.m
function MuBuild()
    global CFG Mu
    row=CFG(:,7);
    switch(row(1))
        case 0
        str = horzcat("@(u) 0+", row(2));
        case 1
        str = "@(u) 0.337534516765287+u*(0.00545792899408279+u*(5.21966907736125e-05+u*(-
              1.05465592811740e-06+u*(5.75208196361954e-09+u*(-1.78522463291645e-11+u*(3.6-
              1385053692621e-14+u*(-3.46263423186378e-17)))))));";
        case 2
        str = func2str ( @b_sklejana );
        case 3
        str = "@(u) 0.47852+u*0.003437+u*u*-0.000015926368;";
        case 4
        str = "@(u) 0.270529360498919+u*(0.0102965513204268+u*(-5.28805197847902e-05+u*(-
              2.89019332648018e-08+u*(2.92192106012333e-10))))";
    end
    Mu = str2func(join(str));
end
```

```
function y = b_sklejana(u)
    X = [ 20, 50, 100, 150, 200, 250, 280, 300   ];
    Y = [ 0.46 , 0.64 , 0.78 , 0.68 , 0.44 , 0.23 , 0.18 , 0.18  ];
    y=0;
    if (u>=X(1))
        if (u<=X(end))
        for i=1:length(X)-1
            if  ( u<=X( i+1 ))
                xx=u-X(i);
                H=(Y(i+1))-(Y(i));
                W=(X(i+1))-(X(i));
                y=Y(i)+(xx*H)/W;
                return
            end
        end
        end
    end
end
```



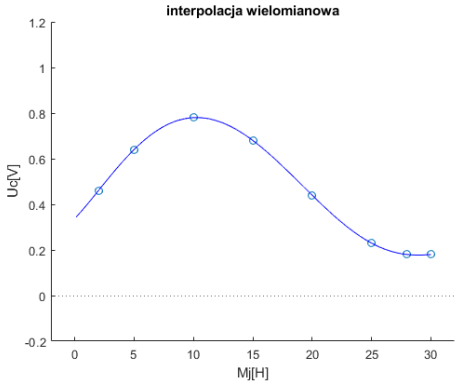
Funkcja MuBuild() realizuje interfejs funkcji Mj(Uc), i przypisuje różne implementacje funkcji Mj(Uc). Raz jest to funkcja stała, niezależna od Uc o wartości 0.8H (przypadek 1) lub inne funkcje interpolujące lub aproksymujące wartoście przekazane w tabeli :

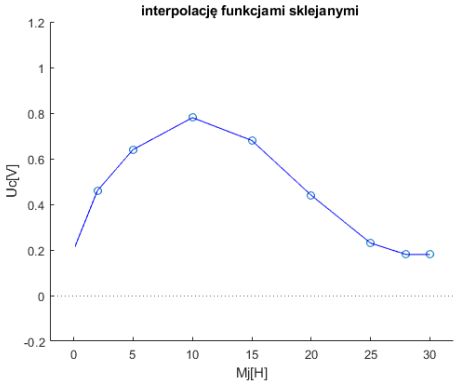
Implementacje to gotowe funkcje bez załaczonych wzorów, schematów wyprowadzeń oraz obliczeń.
Nie chciałem zaciemniać obrazu programu przez umieszczenie tych informacji. Zarówno na egzaminie jak i w pracach domowych wyprowadzaliśmy podobne funkcje dlatego pozwoliłem sobie nie umieszczać listingu w tym dokumencie. Pełna informacja o wyprowadzeniu i kody liczące zamieściłem w repozytorium na github pod adresem:
https://github.com/piotrHeinzelman/MetodyNumeryczne/blob/main/Projekt_1/aproxy/main.m

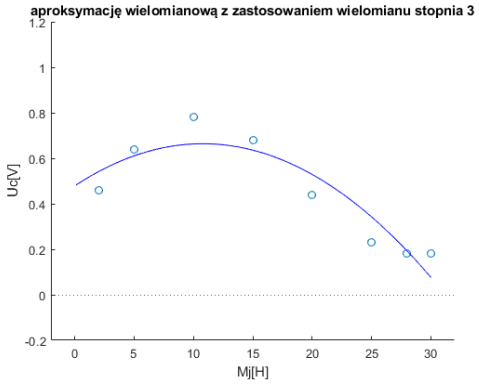
funkcja pobiera konfigurację globalną i na jej podstawie przypisuje do zmiennej globalnej Mu funkcję Mu(Uc) która zwraca wartość indukcyjności odpowiednią do globalnej konfiguracji.

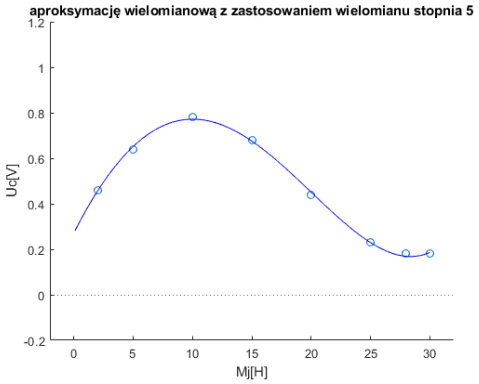
$u_{L1,j}$ [V]	20	50	100	150	200	250	280	300
Mj [H]	0.46	0.64	0.78	0.68	0.44	0.23	0.18	-.18

modyfikując konfigurację programu można łatwo uzyskać charakterystyki Mu(Uc).
wprowadziliśmy 5 rodzajów funkcji:
x) wartośc stała 0.8H
a) interpolacja wielomianowa (użyłem f. Newtona)
b) interpolacja funkcjami sklejanymi
c) aproksymacja wielomianowa wielomianami 3 stopnia
d) aproksymacja wielomianowa wielomianami 5 stopnia









Różnice w kształtach, i dopasowaniu wynikają z:
przy interpolacji krzywa musi przechodzić dokładnie przez wszystkie punkty, jeśli punktów jest wiele krzywa zachowuje się bardzo nieprzewidywalnie zwłaszcza w sytuacji gdy zastosujemy węzły równoodległe lub podobne. Krzywą interpolacji nieco stabilizuje zastosowanie węzłów Czybyszewa czyli badamy wartość funkcji w określonych punktach wyliczanych ze wzorów.

Interpolacja funkcją sklejaną pierwszego stopnia to dopasowanie łamanej przechodzącej przez podane punkty, wartości pośrednie wylicza się z prostej proporcji właściwej dla każdego zakresu.

Aproksymacja nawet jeśli mamy dane z wielu punktów tak dobiera parametry krzywej aby błąd (właściwie kwadrat błędu) dopasowania krzywej był najmniejszy. Można zastosować niewielki stopień krzywej. Dobrze jeśli stopień krzywej odpowiada mniej więcej charakterystyce krzywej którą aproksymujemy.

Jeśli aproksymujemy krzywą stopnia 1 - czyli odcinkiem (prostą) a charakter aproksymowanej funkcji także jest liniowy to wyliczony model dobrze dopasowuje oczekiwane wartości. Jeśli natomiast wybierzemy zbyt niski stopień krzywej tak jak u nas stopień 3 - wówczas krzywa bardzo odbiega od zadanej, a błąd jest duży. W naszym przypadku aproksymacja krzywą stopnia 5 daje bardzo gładki i dobrze dopasowany przebieg.

Ograniczenie wynika wprost z charakterystyki krzywych, i tak krzywa stopnia 1 będzie miała pochodną stałą, i nie będzie miała żadnych ekstremów. Krzywa stopnia 2 będzie miała tylko 1 ekstremum. bez punktóe przegięcia, 1 pochodna będzie liniowa. Krzywa stopnia 3 będzie miała 2 extrema, i może mieć 1 punkt przegięcia. i tak dalej.

moduł Euler.m

```
%Euler.m
function Y = Euler ( T )
    global CFG uGen Mu
    Y = CFG(:,1);
    Uc=Y(3);

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);
D1=(L1/(Mu(Uc)))-((Mu(Uc))/L2);
D2=((Mu(Uc))/L1)-(L2/(Mu(Uc)));

    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

% caÅ‚kowanie Eulera zwykÅ‚e i ulepszone, zaleznie od parametru
     if (Emode==1) % Yn+1=Y+hf( X+h/2 , Y+h/2*(f(X,Y)) ) % ulepszone
        dY = [ ( Y(1) + h*fdy1( t+(h/2), Y+(h/2)*fdy1(t,Y)) )  % nieefektywne - jednak czytelniejsze TU
dla celÃ³w dydaktycznych OK
               ( Y(2) + h*fdy2( t+(h/2), Y+(h/2)*fdy2(t,Y)) )
               ( Y(3) + h*fdy3( t+(h/2), Y+(h/2)*fdy3(t,Y)) )];

     elseif (Emode==2) % Power check
        dY = [ ( 0 )
               ( 0 )
               ( uGen(t) )];

     elseif (Emode==3) % Mu(Uc) check
        dY = [ ( 0 )
               ( 0 )
               ( Mu(t*10)) ];

     else %(Emode==0) % Yn+1=Y+hf(X) % ZwykÅ‚e
        dY = [ ( Y(1) + h*fdy1(t,Y) )
               ( Y(2) + h*fdy2(t,Y) )
               ( Y(3) + h*fdy3(t,Y) )];
    end

%Dy1/dt = (1/C)*y2
%Dy2/dt= (1/L)*(E-Ry2-y1)
end
```

```
% odseparowane poszczegÃ³lne obliczenia
    function dy1 = fdy1(t,Y)
       dy1 =  -Y(1)*(R1/((Mu(Uc))*D1)) + Y(2)*(R2/(L2*D1)) - Y(3)/((Mu(Uc))*D1) + uGen(t)/
((Mu(Uc))*D1);
    end

    function dy2 = fdy2(t,Y)
       dy2 = -Y(1)*(R1/(L1*D2)) +Y(2)*(R2/((Mu(Uc))*D2)) - Y(3)/(L1*D2) + uGen(t)/(L1*D2);
    end

    function dy3 = fdy3(t,Y)
       dy3 =  Y(1)*(1/C);
    end

end
```

Właściwa funkcja stepEuler(t , Y) - oblicza wartość następnego kroku całkowania, i zwraca 3-elementowy wektor. Do pracy potrzebuje parametrów h (z globalnej zmiennej CFG) oraz poprzednią wartość wektora Y. Obliczenia dla uproszczenia (zapisu i programu) wykonywane są przez pomocnicze funkcje dy1 dy2 idy3. Funkcja w zależności od konfiguracji oblicza wartość całki metodą Eulera lub ulepszoną Eulera.

Dla zwiększenia wydajności i poprawienia czytelności oraz zgodości z wymaganiem DRY funkcja jest obudowana dekoratorem, który RAZ przygotowuje parametry pracy i wywołuje obliczenia w pętli tak by zwrócić gotowy zbiór wektorów wyniku.

Przy obliczaniu każdego kroku trzeba wyliczać wartość indukcyjności, używając odpowiedniej funkcji aproksymującej, funkcja ta jest wybierana w zależności od konfiguracji - czyli wartości tablicy globalnej CFG.

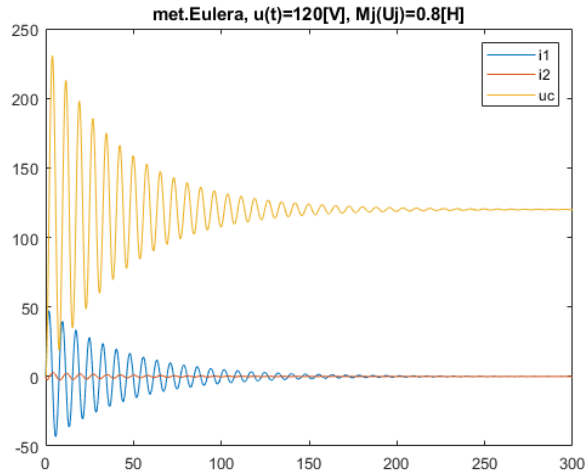
```
% main.m //  fragment

clear all
global CFG uGen
config(-1,[]);
t = CFG(1,2):CFG(2,2):CFG(3,2);
Y = Euler( t );
plot( t, Y(1,:), "-" , t, Y(2,:) , "-",t, Y(3,:) , "-" );


legend ("i1","i2","uc");
% y1=i1
% y2=i2
% y3=uc
```

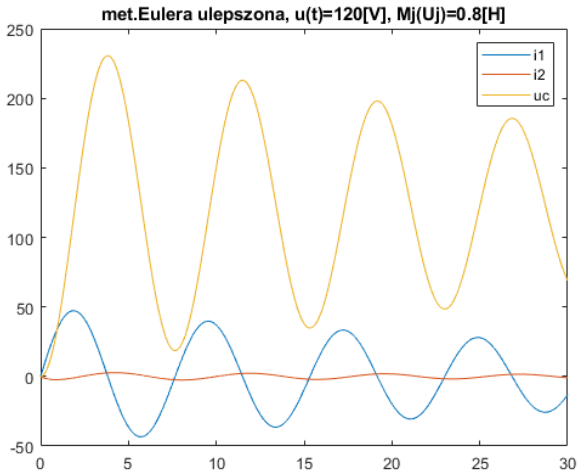
Funkcja głowna Main ma bardzo prostą budowę, po pierwsze deklaruje korzystanie z globalnej zmiennej CFG, ustawia domyślne wartości aplikacji, i wywołuje funkcję roboczą Euler(t). Wynik działania całkowania numerycznego w postaci zbioru wektorów obrazuje za pomocą funkcji systemowej plot oraz dodaje opis.

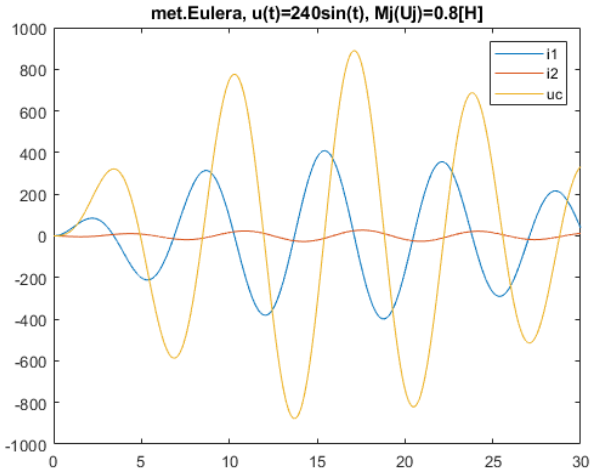
Faktycznie jest to tylko studium użycia funkcji roboczej Euler.

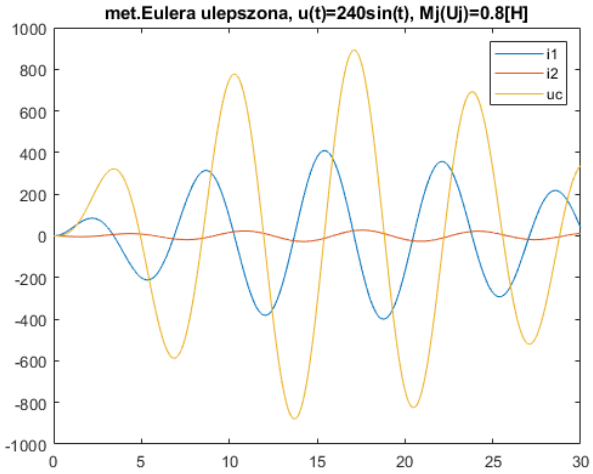
2. Wyniki (wykresy) obliczeń dla Mj=0.8[H]

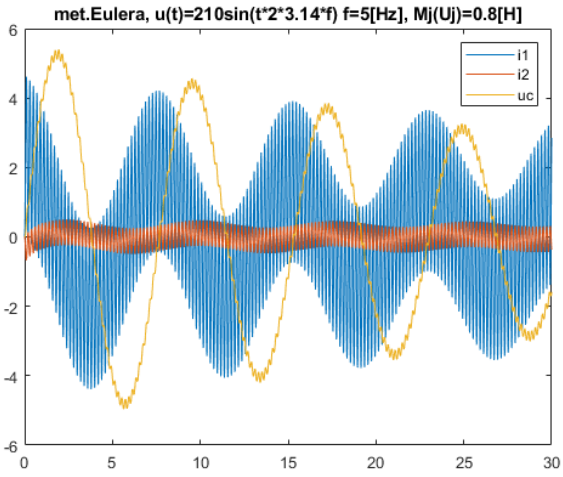
przy zadanych domyślnych parametrach układu
wyliczone metodą Eulera (lewe) i Eulera poprawioną (prawe) wykresy.

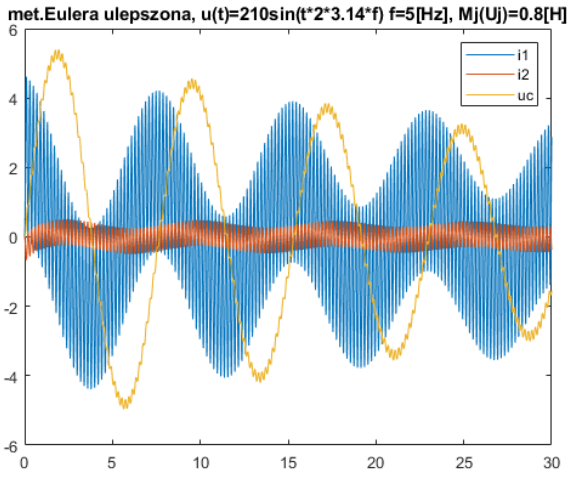


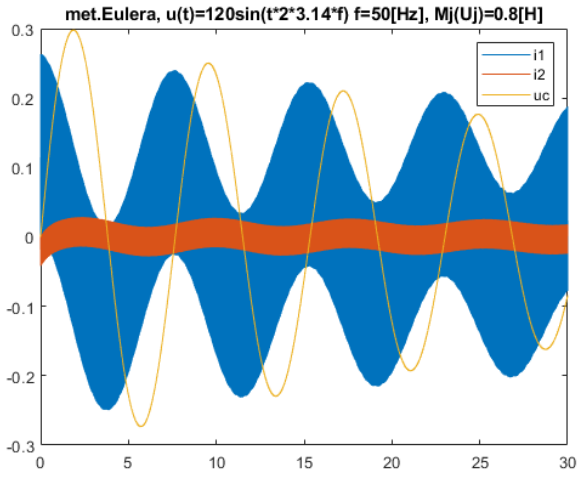


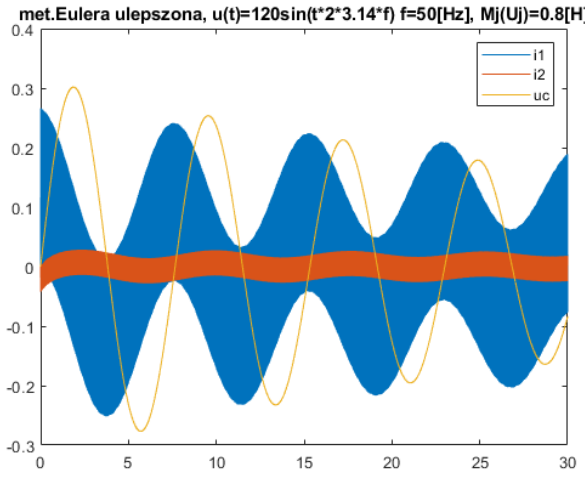


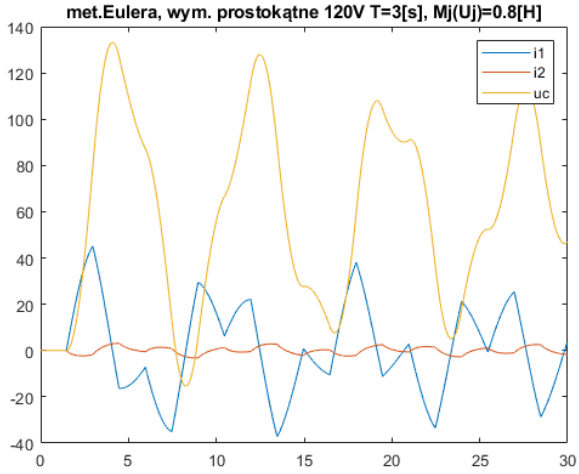


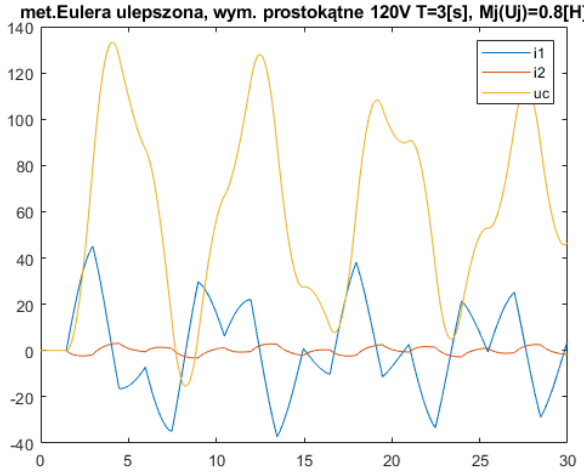




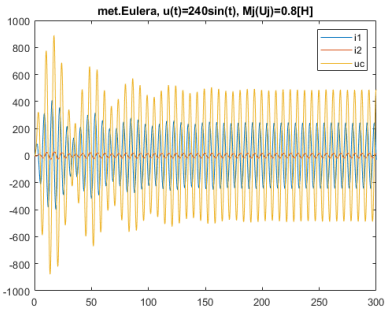


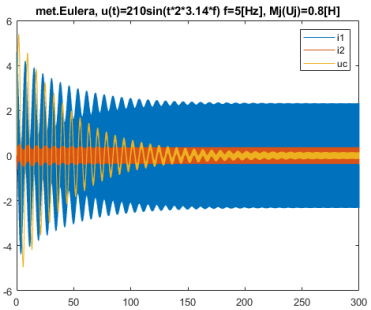


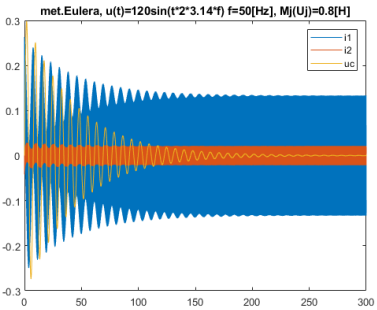




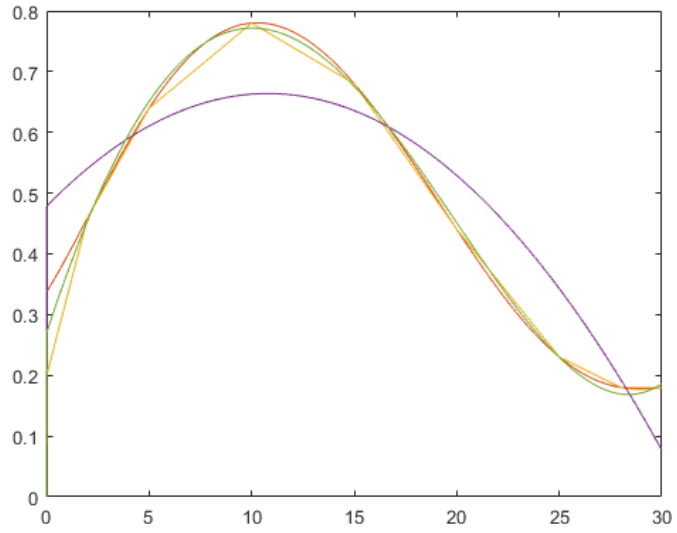
Pierwszy wykres został narysowany w większym zakresie czasu - a podglądałem też inne wykresy przy wiekszym przedziale czasowym - pokazuje że stan układu stabilizuje się na poziomach oczekiwanych czyli napięcie na pojemności nieco przesunięte w fazie, przy napięciu stałym osiąga wartość źródła. Przy wymuszeniu sinusoidalnym wartość nieco mniejsza od wartości wymuszenia. Prąd I2 w okolicy zera, niewielki prad w stanie nieustalonym.
Przy niższych częstotliwościach maksymalne chwilowe napięcia sięgają 900V przy wymuszeniu 240V. Dla takich wartości od -1000 do 1000 [V] funkcja Mj(Uc) powinna zwracać sensowne wartości, podobnie z wartością ujemną prądu funkcja Mj(Uc) powinna być symetryczna.
Przy częstotliwościach wyższych napięcia osiągają 6[V] i 0.3[V] - drgania układu są bardzo mocno tłumione.
Można zatem założyć że symulator nie odbiega w stopniu od zachowania oczekiwanego.
Różnica między metodą Eulera a Eulera ulepszoną w tym symulatorze nie jest jakaś drastyczna.
Generalnie można stwierdzić że całkowanie numeryczne zmniejsza błędy odwzorowania, natomiast różniczkowanie numerczne zdjększa błąd odwzorowania.

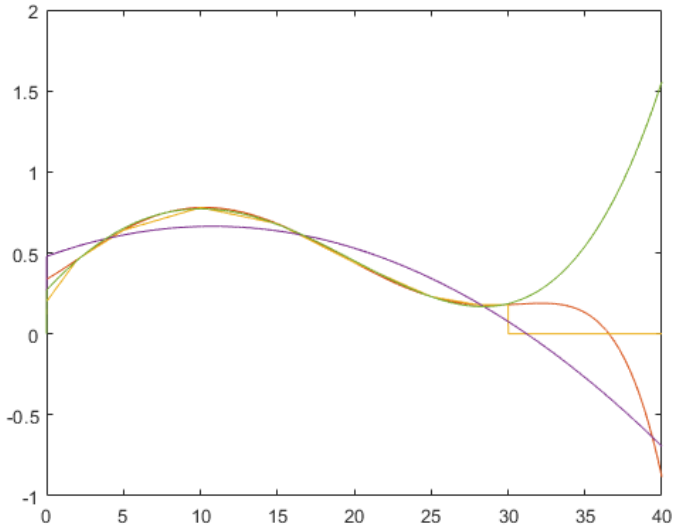


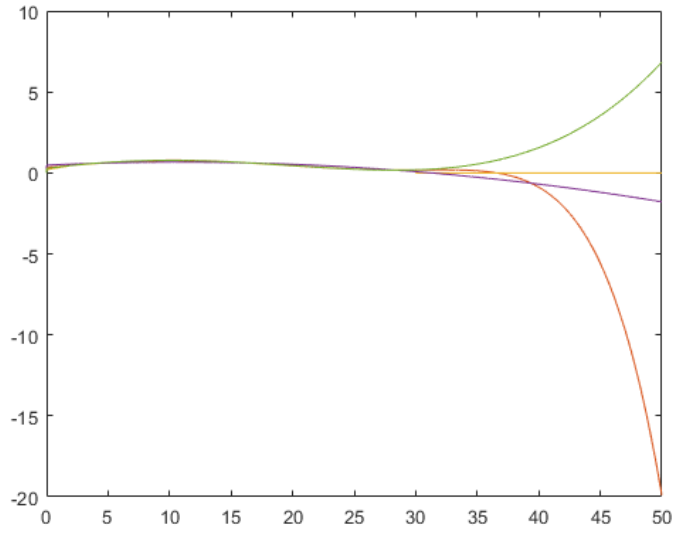




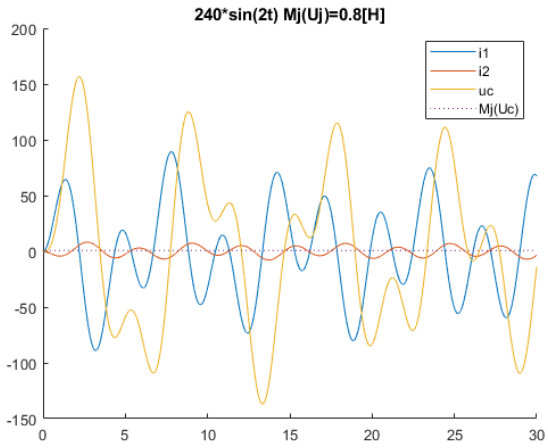
Jednakże funkcja Mj(Uc) zwraca wartości sensowne w granicach 0-30 to przy wyższych napięciach dzieje się coś dziwnego. Prąd płynące przez cewkę dla wartości ujemnych powinien dać ujemną wartość. ale zobaczmy co się stanie....

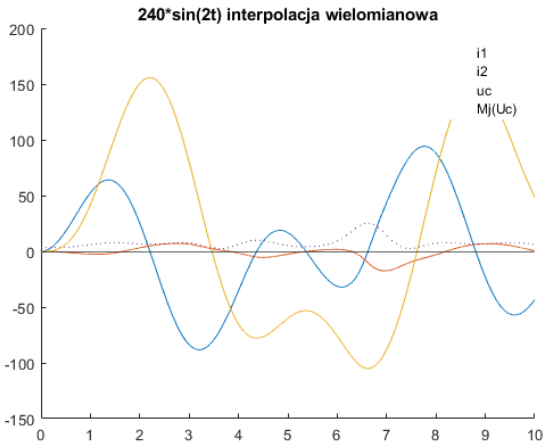


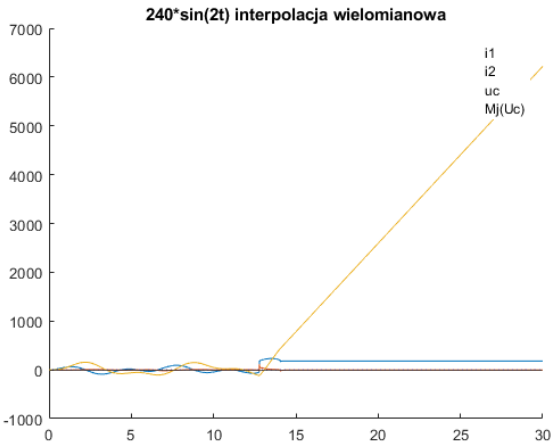


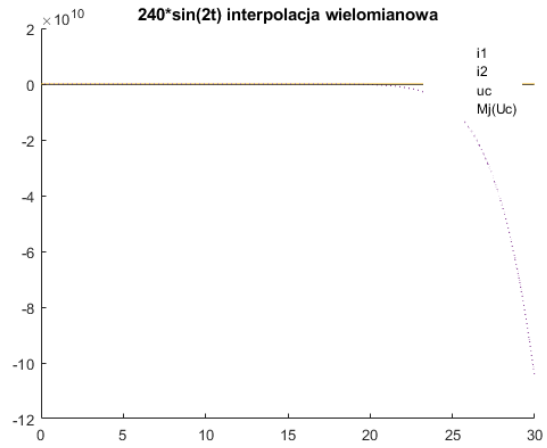


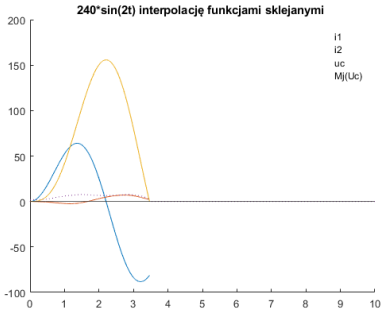
Poniżej 8 wymaganych wykresów 4 dla 240sin(t) i dla 120(sint2)
pierwszy Mj=0.8[H], kolejne - interpolacja wielomianowa, różne zakresy czasu.
na ostatnim wykresie pokazałem także wartość Mj.
następne - kolejne interpolacje. Interpolacja sklejana - błąd przy wartościach ujemnych ?
aproksymacja wielomianowa stopnia 3 i stopnia 5 wygląda w miarę dobrze.

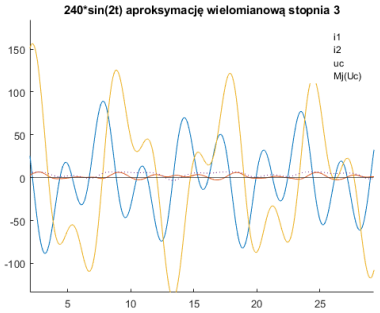


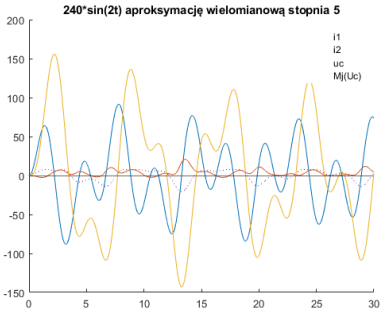




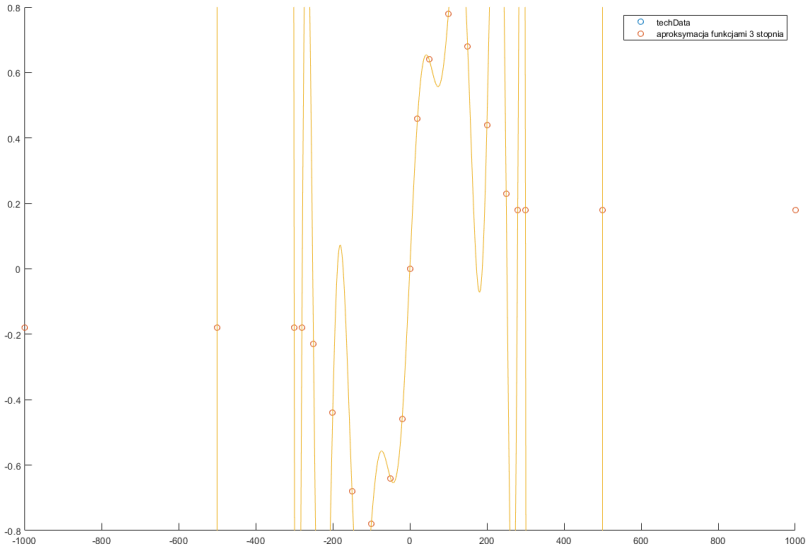




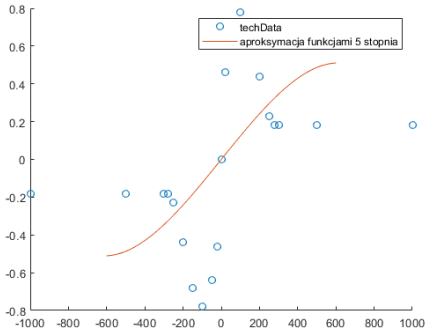


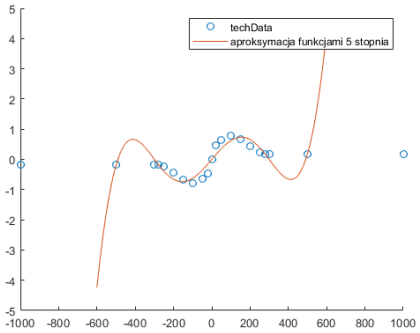


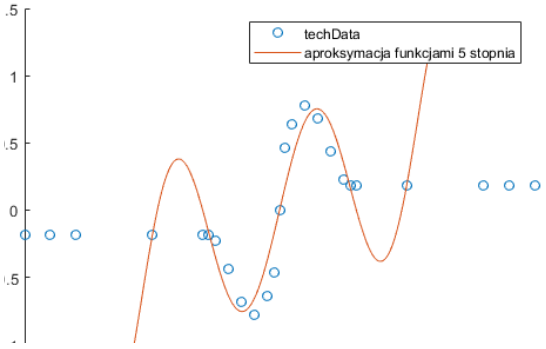
interpolacja Vandermorta dla danych:
X = [-1000, -500, -300, -280, -250, -200, -150, -100, -50, -20, 0, 20, 50, 100, 150, 200, 250, 280, 300, 500, 1000];
Y = [-0.18 , -0.18 , -0.18, -0.18, -0.23, -0.44, -0.68, -0.78, -0.64, -0.46, 0, 0.46 , 0.64 , 0.78 , 0.68 , 0.44 , 0.23 , 0.18 , 0.18 , 0.18, 0.18];



ta funkcja nie nadaje się do tego rozwizania.

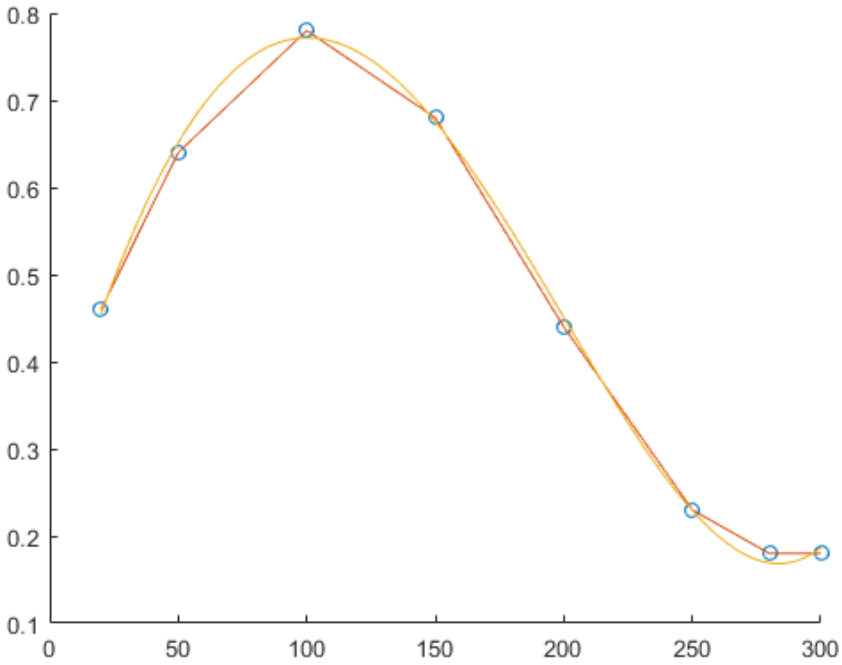






aproksymacja krzywą 5 stopnia... też jeszcze niedoskonała

konejne podejście do aproksymacji funkcjami 5 stopnia
kod do automatycznego wyliczania
a) wartości wektora współczynników
b) obliczania wartości y

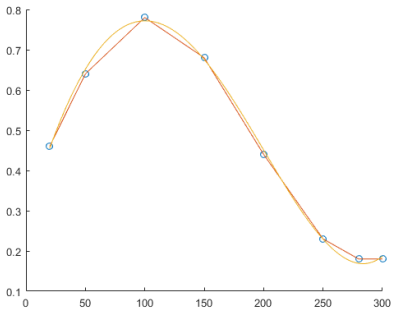


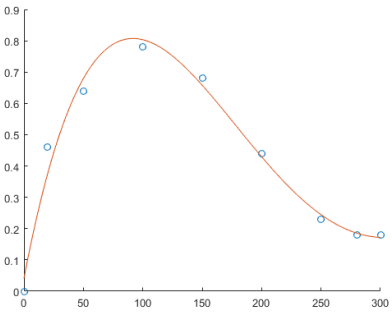

```
 clear all;

X = [ 20, 50, 100, 150, 200, 250, 280, 300 ];
XX= 20:.1:300;
Y = [ 0.46 , 0.64 , 0.78 , 0.68 , 0.44 , 0.23 , 0.18 , 0.18 ];


global vectorA
vectorA = aproxFun5stVector();

for i=1:length(XX)
    bSK(i)=FunkcjaSklejana(XX(i));
    dAP5(i)=aproxFun5st(XX(i));
end

hold on;
plot( X,Y,"o" );
plot( XX,bSK,"-" ); % b - f.sklejana
plot( XX,dAP5,"-" ); % d - aproksymacja 5 st.
```

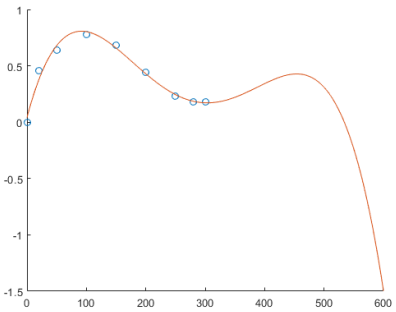
```
% funkcja sklejana
function y = FunkcjaSklejana(x)
    X = [ 20, 50, 100, 150, 200, 250, 280, 300   ];    %% local data + aprox data
    Y = [ 0.46 , 0.64 , 0.78 , 0.68 , 0.44 , 0.23 , 0.18 , 0.18  ];
    if (x>=X(1))
        if (x<=X(end))
        for i=1:length(X)-1
            if  ( x<=X( i+1 ))
                xx=x-X(i);
                H=(Y(i+1))-(Y(i));
                W=(X(i+1))-(X(i));
                y=Y(i)+(xx*H)/W;
                return
            end % H/z proporcji H/y=W/x
        end
        end
    end
end
```

```
function vectorA = aproxFun5stVector()

    X = [ 20, 50, 100, 150, 200, 250, 280, 300   ];    %% local data + aprox data
    Y = [ 0.46 , 0.64 , 0.78 , 0.68 , 0.44 , 0.23 , 0.18 , 0.18  ];
    W = [ones(length(X),1) X' X'.^2 X'.^3 X'.^4];
    A = W'*W;
    b = W'*Y';
    vectorA = A\b;
end       % y=0.270529360498919+x*(0.0102965513204268+x*(-5.28805197847902e-05+x*(-2.89019332648018e-08+x-
*(2.92192106012333e-10))));
```

```
function y = aproxFun5st(x)
    global vectorA

   y=0;
   len=length(vectorA);
   for i=1:len-1
    bck = (len-i+1);
    ai=vectorA(bck);
      y = x*(ai+y);
   end
   y=y+vectorA(1);
end
```

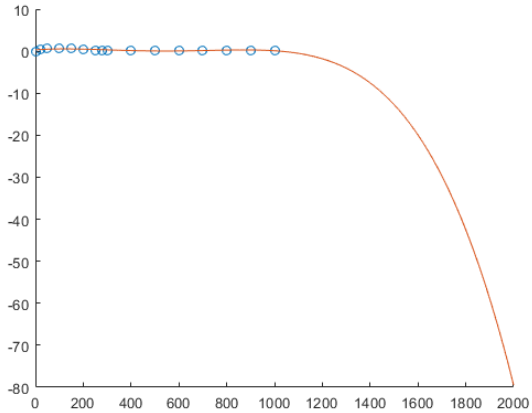






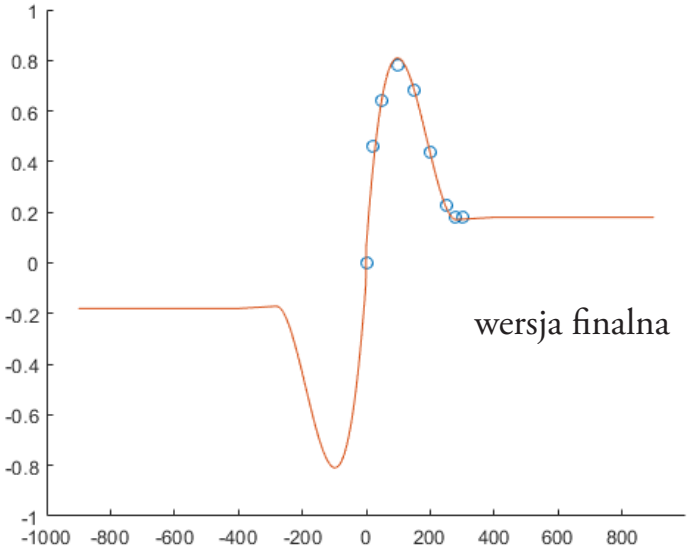
Funkcja Mj(Uc) powinna być nieparzysta. Przy zmianie kiedunku prądu powinna zwracać Mj z odwrotnym znakiem. zatem powinna przechodzić przez punkt 0,0.
wydaje się że powinna zbiegać do 0.18 w nieskończoności i do -0.18 w -nieskończoności.
Prawdopodobnie ciężko będzie dobrać takie wartości wektora by dopasować sensownie funkcję zarówno przy wartościach około 600-800 a nawet 1000, oraz tak by jej kształt był dopasowany lub bardzo zbliżony do danych odczytanych w labolatorium.

Jeśli nie uda się dopasować krzywej niewielkiego rzędu w zakresach -1000 1000 możemy posłużyć się wybiegiem z funkcji sklejanych i skleić sobie zakresy
< -∞ , -300 > , <-300,+300> , <300,+ ∞>
lub <0,300>, <300, ∞> i przy x<0 odwracać znaki wejściowego x i wyjściowego y.



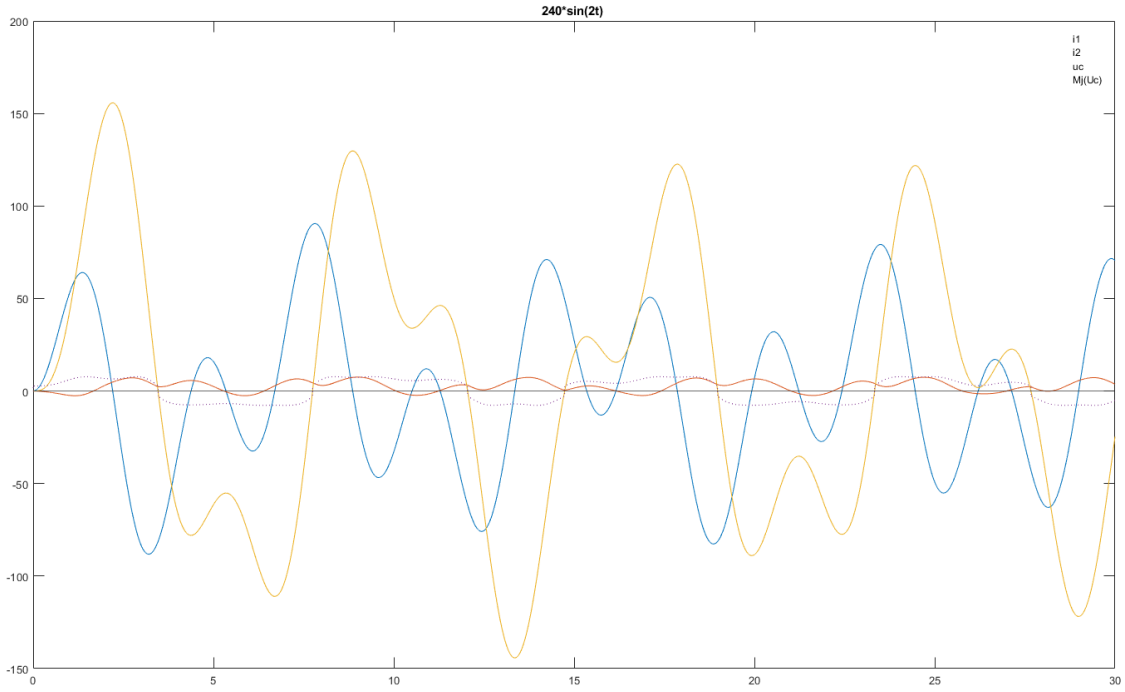
aproksymacja krzywymi 5. stopnia przy większych zakresach zdaje się nie radzić sobie z tak odległymi puntkami.
Jest to dość oczywiste, posługujemy się skwantowanymi liczbami, przy ograniczeniu ich długości do 64 bitów. więc wyniki na dłuższych dystansach błedy będą potęgowały.

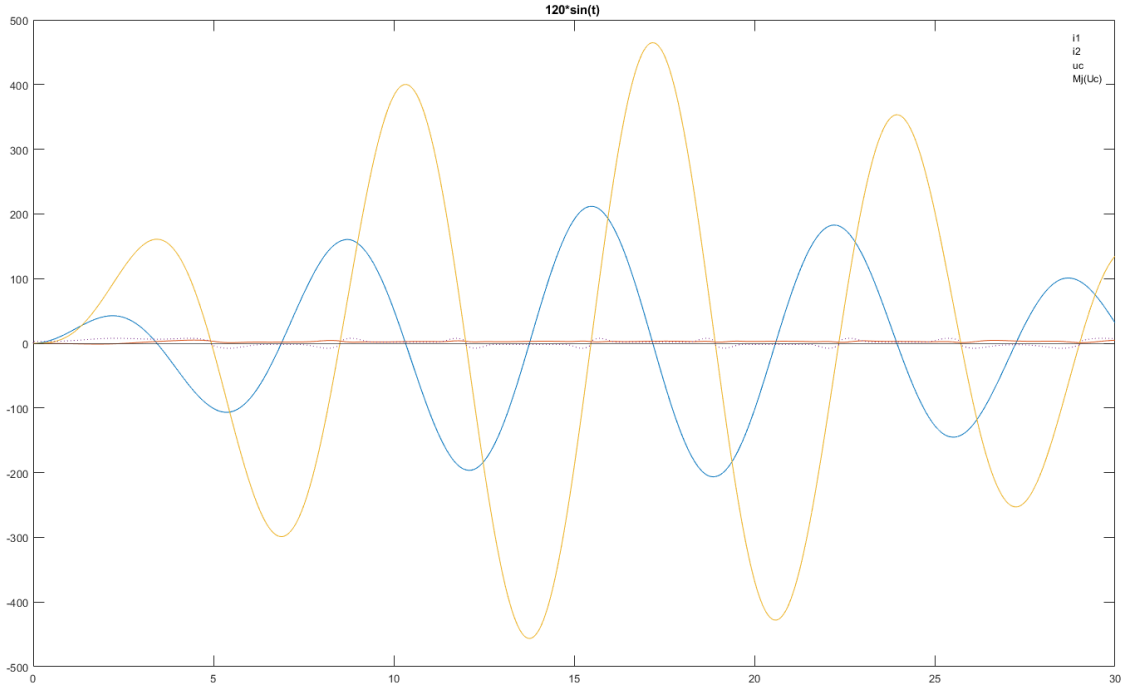
cóż potniemy zakresy i zobaczymy




```
function MuAprox()
XXX = [ 0, 20, 50, 100, 150, 200, 250, 280, 300];
YYY = [ 0, 0.46 , 0.64 , 0.78 , 0.68 , 0.44 , 0.23 , 0.18 ,0.18 ];
vectorA = aproxFun5stVector();

XX=-XXX(end)*3:1:XXX(end)*3;
for j=1:length(XX)
    dAP5(j)=aproxFun5st(XX(j));
end

hold on;
plot( XXX,YYY,"o" );
plot( XX,dAP5,"-" ); % d - aproksymacja 5 st.

function vectorA = aproxFun5stVector()
    X=XXX;
    Y=YYY;
    W = [ones(length(X),1) X' X'.^2 X'.^3  ];
    A = W'*W;
    b = W'*Y';
    vectorA = A\b;
end


     function y = aproxFun5st(x)
        multi=1; if (x<0) x=-x; multi=-1; end % nieparzystosc

        if (x>400) y=0.18;
        elseif (x>280)
            y0=0.171013;y1=0.18;
            y=y0+(y1-y0)*((x-280)/(400-280));
        else
            y=0;
            len=length(vectorA);
            for i=1:len-1
             bck = (len-i+1);
             ai=vectorA(bck);
               y = x*(ai+y);
            end
            y=y+vectorA(1);
        end
        y=y*multi;
     end

end
```

ostateczne wersje dla napięcia 240sin(2t) oraz 120sin(t); dla Mj(Uc) poprawionego





Część 3 Moc

funkcja całkująca:

```
%Power.m
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 całkująca - użycie:

```
%main.m
clear all
global CFG uGen
config(-1,[]);
t = CFG(1,2):CFG(2,2):CFG(3,2);


if (true)
config( 6, [   -1,     1,    1   ]); info="1V h=0.01[s]";  UGen(); % "sinus"[0,V,*2πf], "rectangle": [ 2,Vmax,0 ][2,120,0]
Y = Euler( t ); v=[Power(Y,1),Power(Y,0),info]
config( 6, [   2,     120,    1          ]); info="prostokat 120V"; UGen();
Y = Euler( t ); v=[Power(Y,1),Power(Y,0),info]
config( 6, [   0,     240,    1          ]); info="240*sin(t)"; UGen();
Y = Euler( t ); v=[Power(Y,1),Power(Y,0),info]
config( 6, [   0,     210,    2*3.14*5   ]); info="210*sin(2πωt)f=5[Hz]"; UGen();
Y = Euler( t ); v=[Power(Y,1),Power(Y,0),info]
config( 6, [   0,     120,    2*3.14*50  ]); info="120*sin(2πωt)f=50[Hz]"; UGen();
Y = Euler( t ); v=[Power(Y,1),Power(Y,0),info]
return
end
```

wyniki dla t=0.01[s]

Wymuszenie	prostokąt t=0.01 (Mj=0.8)	prostokąt t=0.000001 (Mj=0.8)	parabola t=0.01 (Mj=0.8)	parabola t=0.000001 (Mj=0.8)	prostokąt t=0.01 Mj(Uc)	parabola t=0.01 Mj(Uc)
sin(t)						2.8096 (t=10e-5)
e(t)=1V h=0.01[s]	0.20094	0.18729	0.20093	0.18729	0.16492	0.16492
prostokat 120V	1853.1679	1765.7214	1853.0935	1765.7214	1544.5026	1544.4958
240*sin(t)	234847.7905	212410.0256	234838.3941	212410.0248	158482.4858	158482.1632
210*sin(2πωt)f=5[Hz]	35.5903	34.7711	35.5872	34.7711	16.2771	16.2771
120*sin(2πωt)f=50[Hz]	0.28521	0.11369	0.28445	0.11369	0.052846	0.052846

Część 4 Moc

F(f)=P(f)-406:

```
%Pow.m
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(2πωt) f="+Fi+" Power = "+po+" : po-406=" + (po-406) + "."
 P=po-406;
end
```

```
%bisekcja.m
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
```

wynik = 0.6736

Część 4.a Newton obliczenia deltaX

```
obliczenia wartości deltaX dla obliczeń pochodnej
     x=1/4;
        deltaX=1;
        t=0:1e-5:30;
        y=Px(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
        end

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

wyniki:
    "roznica:-0.0056791 : deltaX0.00012207"
```

Dla wartości deltaX równego 0.00012207 wartośc pochodnej różni się od wartości pochodnej o wartość 0.0056791%. Należy zwrócić uwagę że różnica pomiędzy następującymi krokami może być dodatnia lub ujemna. Zatem aby znaleźć rozwiązanie można użyć warunku:

```
if (roznica*roznica<0.01*0.01)
```

co jest tożsame z |roznica|<0.01 lub mniej eleganckim ((roznica>-0.01) && (roznica<0.01)).

Odnaleziona wartośc jest większa 10-krotnie niż "krok" całkowania który wynosi 10^-5.

w obliczeniach przyjmuję zatem wartość deltaX =0.0001;

Część 4.b Newton

Obliczenia, przyjąłem podobnie jak w metodzie bisekcji 15 cykli obliczeń wartości Fi.

poniżej kod

```
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="F0: " + F0 + ", Fi: " +Fi
end
```

natomiast już przy 8 iteracji dostajemy wyniki:
"i: 8 F0: 0.67312, Fi: 1.1836e-06 Pi: 406"
dostajemy błąd rzędu 10e-6 przy 16 obliczeniach wartości fukcji, i uzyskujemy o jeden rzęd wielkości dokładniejszą wartość szukaną od metody bisekcji, z 15 cyklami i 30 obliczeniami.
Choć można to oczywiście trochę optymalizować, ponieważ pewne obliczenia robimy dwa razy.

Część 4.c Metoda siecznych

Obliczenia, przyjąłem podobnie jak w metodzie bisekcji 15 cykli obliczeń wartości Fi.

poniżej kod

```
        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;
```

Przyjmuję x0=0.8, oraz x1=1; wynik działania metody po 2 iteracjach=0.67133

```
Wyniki obliczania częstotliwości dla mocy = 406 [W]
```

metoda	Wartość rozwiązania - częstotliwość f	Wartość funkcji F	Liczba iteracji metody	Liczba obliczeń mocy P
Bisekcji	0.673552640102571	-3.6402e-08	35	70
Siecznych	0.67133	3.4330	2	7
Quasi-Newtona	0.67312	5.2895e+04	15	30

Najbardziej wydajna jest metoda siecznych. metoda Newtona jest nieco mniej wydajna. Metoda bisekcji jest najprostrza do wdrożenia, nie jest bardzo wydajna, jednak zawsze można dobrać ilośc iteracji do oczekiwanego poziomu błędu.