Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Metody Numeryczne

Projekt zaliczeniowy

Symulator parametrów obwodowych obwodu ze sprzężeniem indukcyjnym

Dorota Sadowska

315570

WARSZAWA, 2021

Spis treści

Część 1

Wstęp teoretyczny

W części pierwszej należało napisać symulator stanu nieustalonego dla wskazanych wymuszeń za pomocą źródła napięcia e(t) przy założonych parametrach:

- $\mathrm{R}_1 = 0.1[\Omega]$,
- $\mathrm{R}_2 = 10[\Omega]$,
- $\mathrm{C} = 0.5[\mathrm{F}]$,
- $\mathrm{L}_1 = 3[\mathrm{H}]$,
- $\mathrm{L}_2 = 5[\mathrm{H}]$,
- $\mathrm{M} = 0.8[\mathrm{H}]$.

Przedział czasowy $t \in < 0; 30 > [s]$ podzielono z krokiem $h = 0.001$. Stała $y = [0; 0; 0]$ jest to stan początkowy i oznacza kolejno $I_1 = 0[A]$, $I_2 = 0[A]$, $U_C = 0[V]$.

W celu obliczenia wartości I_1, I_2, U_C dla danego wymuszenia $e(t)$ w danej chwili zastosowano metodę Eulera oraz zmodyfikowaną (ulepszoną) metodę Eulera.

W prostej metodzie Eulera w każdej iteracji od 1 do ilości przedziałów wektora czasu przypisywano wartość macierzy y na kolejnym miejscu względem iteratora, biorąc obecne miejsce macierzy y względem iteratora i dodając do niego pomnożoną przez krok h pochodną $dy = f(t, y)$.

W zmodyfikowanej metodzie Eulera pochodna obliczana jest dwukrotnie: najpierw pochodna wewnętrzna z obecnego miejsca macierzy y względem iteratora pomnożona przez połowę kroku, a następnie powyższa wartość jest dodana do wartości obecnego miejsca macierzy y względem iteratora, a do funkcji czasu dodana jest połowa kroku h. Z powyższych wartości t, y jest liczona pochodna $dy = f(t, y)$. Dalej, analogicznie jak w metodzie prostej Eulera, jest ona mnożona przez krok h, a następnie dodana do obecnego miejsca macierzy y względem iteratora.

```
function y = simply_euler(t, y, h)
    for i=1:length(t)-1
        y(:, i+1)=y(:, i)+h*f(t(i),y(:, i));
    end
end

function y1 = modified_euler(t, y1, h)
    for i=1:length(t)-1
        y1(:, i+1)=y1(:, i)+h*f(t(i)+h/2, y1(:, i)+f(t(i),y1(:, i))*h/2);
    end
end
```

Rys. 1. Implementacja funkcji obliczającej kolejne wartości macierzy y metodą Eulera i zmodyfikowaną metodą Eulera/.

Zmodyfikowana metoda Eulera jest dokładniejsza, lecz w przypadku zadanego symulatora różnice pomiędzy powyższymi metodami są niezauważalne. Jednak ze względu na mniejszy błąd, w dalszych częściach (2., 3. 4.) projektu zastosowano zmodyfikowaną metodę Eulera.

W celu policzenia pochodnej $dy = f(t, y)$ zastosowano macierzową metodę rozwiązywania układów równań:

```
function dy = f(t,y)
    e=E(t);
    R1 = 0.1;       %[Ohm]
    R2 = 10;        %[Ohm]
    C = 0.5;        %[F]
    L1 = 3;         %[H]
    L2 = 5;         %[H]
    M = 0.8;        %[H]
    i1=y(1);
    i2=y(2);
    uC=y(3);
    D1=L1/M-M/L2;
    D2=M/L1-L2/M;

    B = [-R1/(M*D1)   R2/(L2*D1)    -1/(M*D1);
         -R1/(L1*D2)  R2/(M*D2)     -1/(L1*D2);
            1/C            0           0   ];
    G = [1/(M*D1); 1/(L1*D2); 0];
    H = [i1; i2; uC];
    dy = B*H+G.*e;
end
```

Rys. 2. Implementacja funkcji obliczającej różniczkę w metodzie Eulera.

W przypadku ostatniego członu $G * e(t)$ jest to mnożenie macierzy przez skalar, czyli każdy element macierzy G pomnożony jest przez wartość wymuszenia dla danej chwili.

Przebiegi prądów i_1, i_2 oraz napięć u_c i e

Kalibracja symulatora

W celu ustalenia poprawności zastosowanych metod numerycznych oraz ich implementacji zastosowano kalibrację dla $e = 1[V]$ (rys. 1) oraz $e = sin(t)[V]$ (rys. 2).

W przypadku $e = 1[V]$ wartości prądów i_1, i_2 oraz napięcia u_c powinny w czasie $t \to \infty$ dążyć do stałej wartości ($0A$ dla prądów oraz $1V$ dla napięcia u_c), co widoczne jest na poniższych wykresach. W celu uwidocznienia przebiegu dla $t \to \infty$, zastosowano zwiększony przedział czasowy $t \in < 0; 200 >$.

W przypadku $e = \sin(t)[V]$ wartości prądów i_1, i_2 oraz napięcia u_c powinny w czasie $t \to \infty$ dążyć do ustabilizowania oscylacji wokół wartości $0[A]$ dla prądów oraz $0[V]$ dla napięcia u_c. Można zauważyć tę stabilizację po 140 sekundzie. W celu uwidocznienia przebiegu dla $t \to \infty$, zastosowano zwiększony przedział czasowy
$t \in < 0; 200 >$.

Przebieg zamieszczonych wykresów jest zgodny z przewidywaniami; można zatem wnioskować, iż użyte metody zostały zaimplementowane w poprawny sposób.

$$E = \begin{cases} 120\ dla\ t < \frac{T}{2} \\ 0\ dla\ t \geq \frac{T}{2} \end{cases} [V], \quad T = 3[s];$$

Dla przebiegu prostokątnego napięcie na kondensatorze w początkowej chwili oscyluje z dużą amplitudą. Wraz z czasem (dla t>30s), oczekiwano, iż amplituda wychyleń zmniejszy się, aż do ustabilizowania oscylacji wokół 60V, co zostało osiągnięte. Przedstawione poniżej wykresy (rys. 5) zgodne są z wykresami wzorowymi zamieszczonymi w skrypcie.

$$e(t) = 240\sin(t)$$

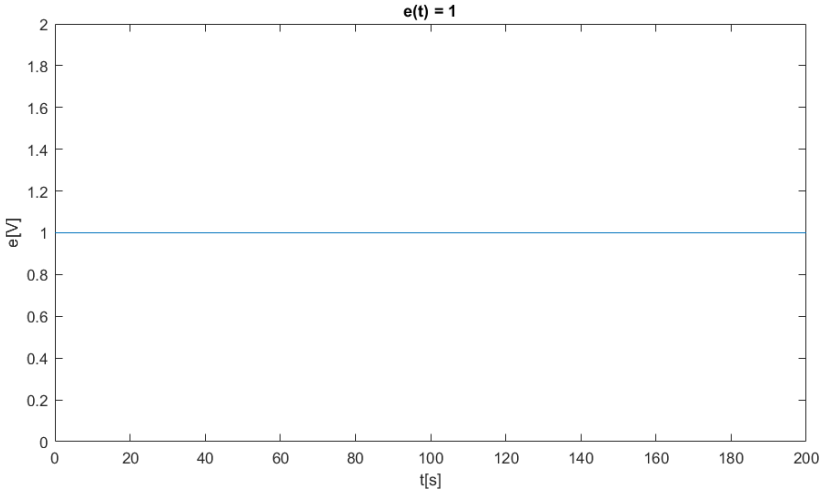
Wymuszenie $e(t) = 240\sin(t)$ od przykładu z kalibracji $e(t) = \sin(t)$ różni się jedynie wielkością amplitudy sinusoidy napięcia na kondensatorze U_C oraz prądów I_1, I_2. Na poniższym wykresie widać pierwszą paczkę falową, dla której dalej, analogicznie jak w przypadku przykładu, spodziewano się ustabilizowania wokół zera.

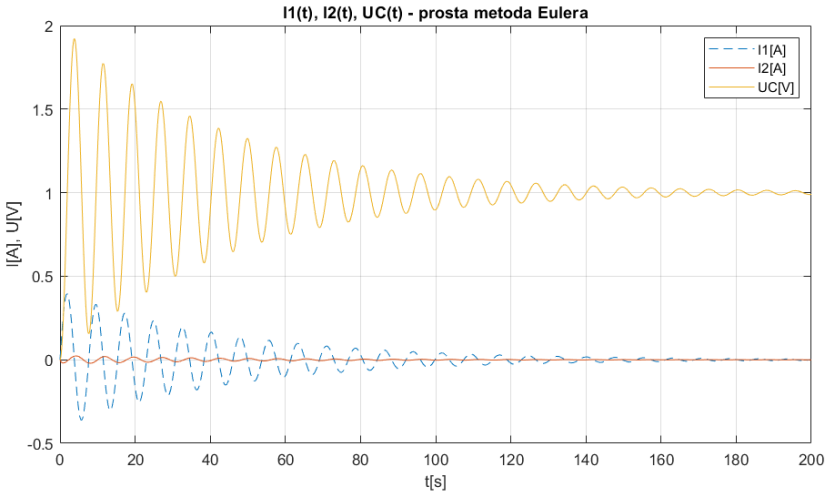
$$e(t) = 210\sin(2\pi f t)\,, dla\ f = 5Hz$$

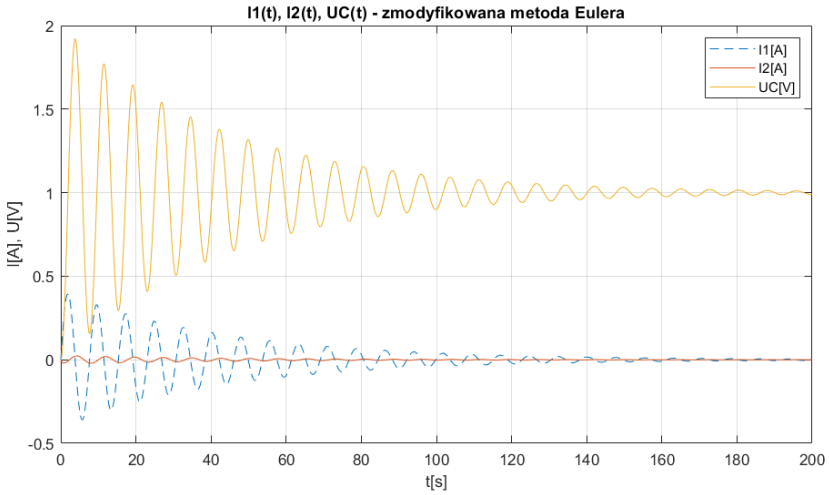
Otrzymane wykresy są zgodne z wykresami wzorcowymi ze skryptu. Widoczny jest wpływ częstotliwości na przebiegi – są zagęszczone.

$$e(t) = 120\sin(2\pi f t)\,, dla\ f = 50Hz$$

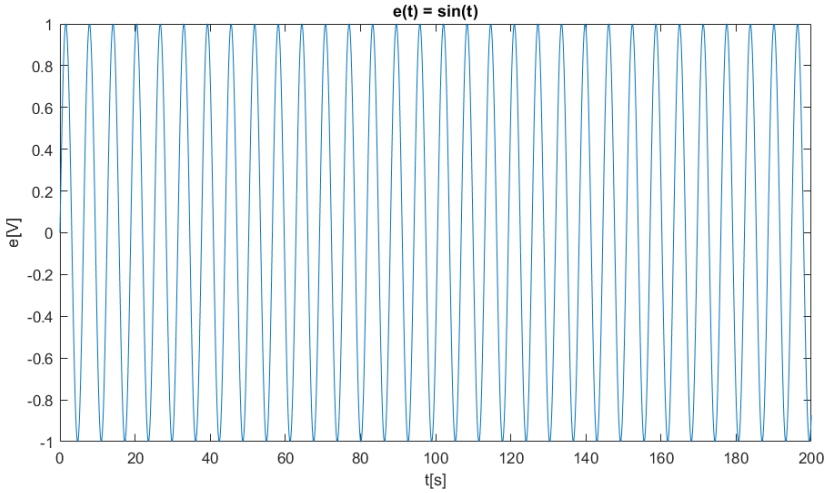
Duża częstotliwość w wymuszeniu powoduje, że przebiegi są gęste, a pojedyncze sinusoidy niewidoczne.

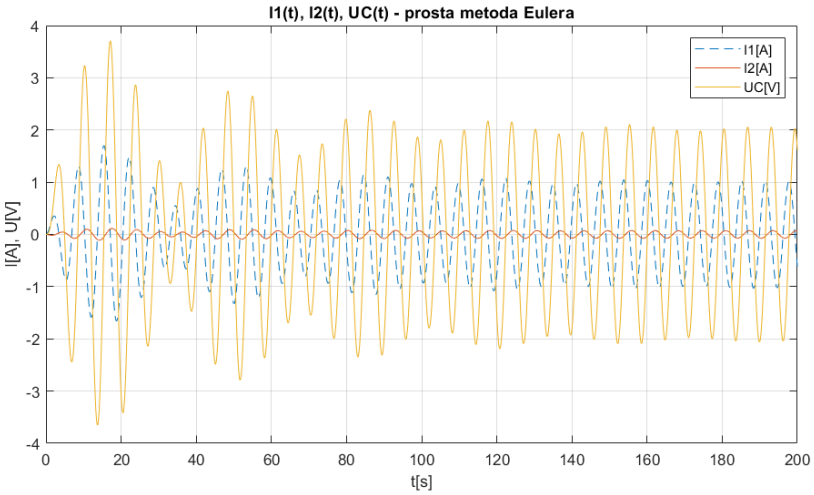


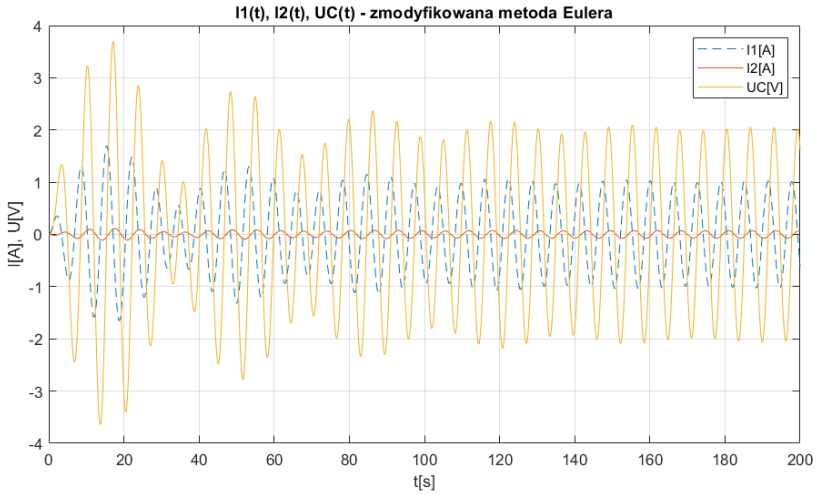




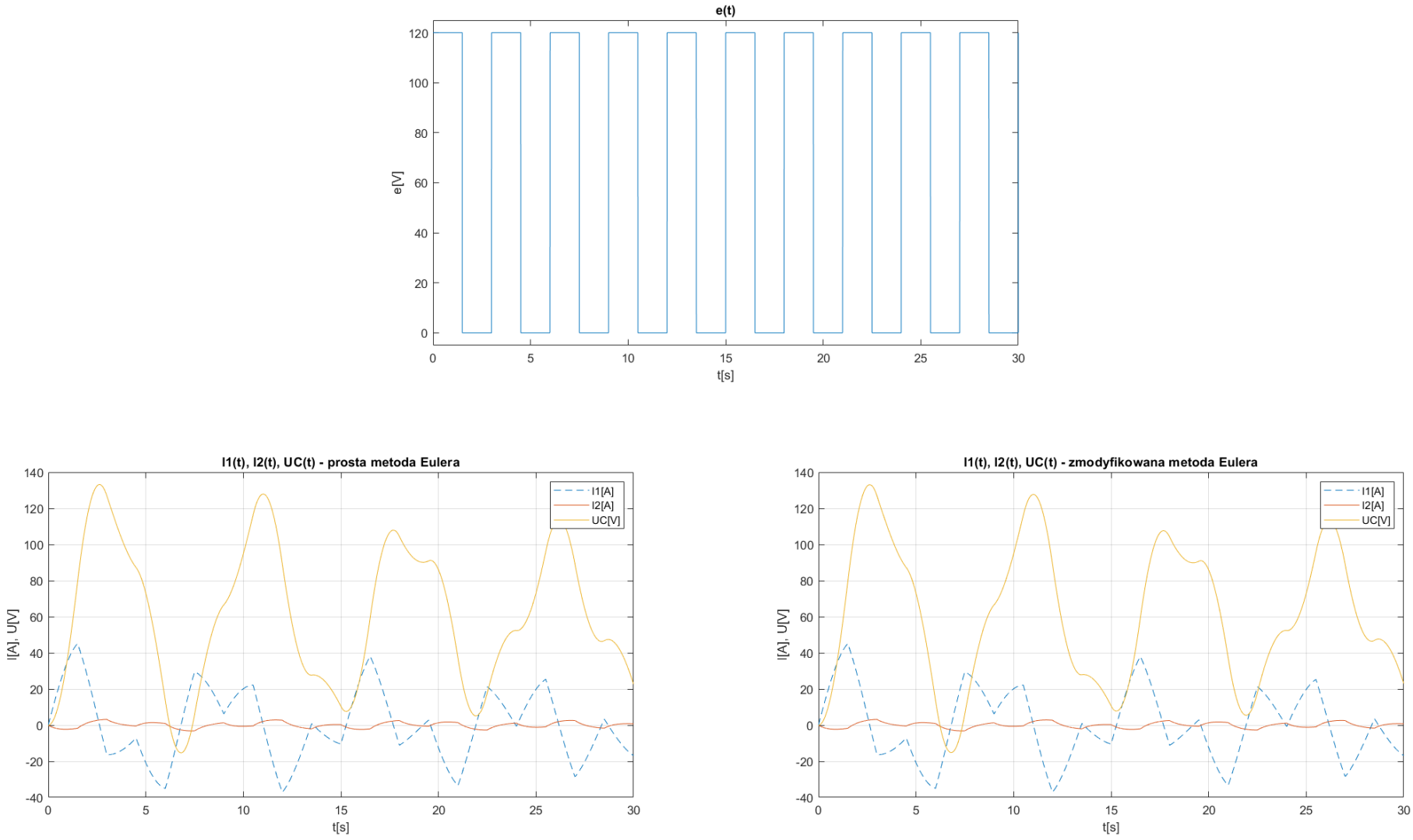
Rys. 3. Przebiegi (kolejno): wymuszenia e(t)=1V, prądów I1, I2, napięcia Uc prostą metodą Eulera oraz zmodyfikowaną metodą Eulera.



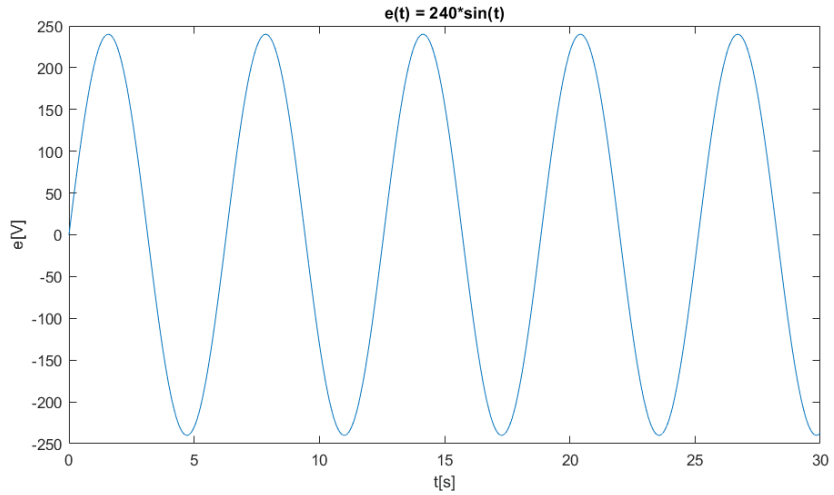


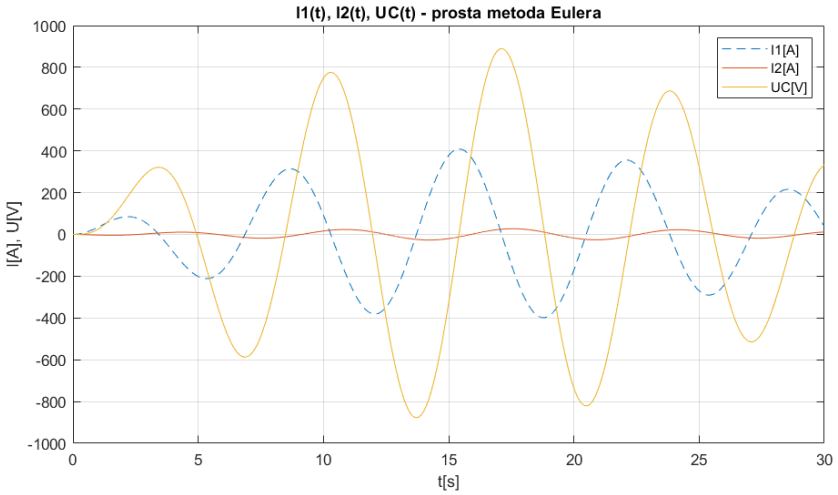


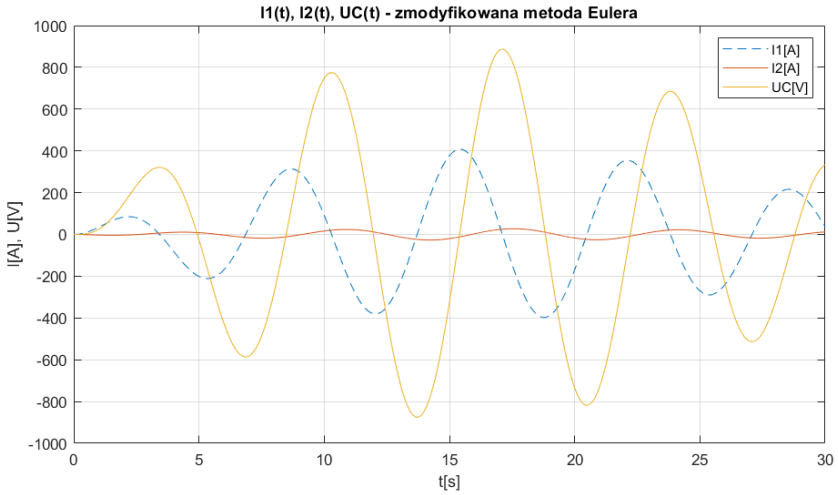
Rys. 4. Przebiegi (kolejno): wymuszenia e(t)=sin(t), prądów I1, I2, napięcia Uc prostą metodą Eulera oraz zmodyfikowaną metodą Eulera.



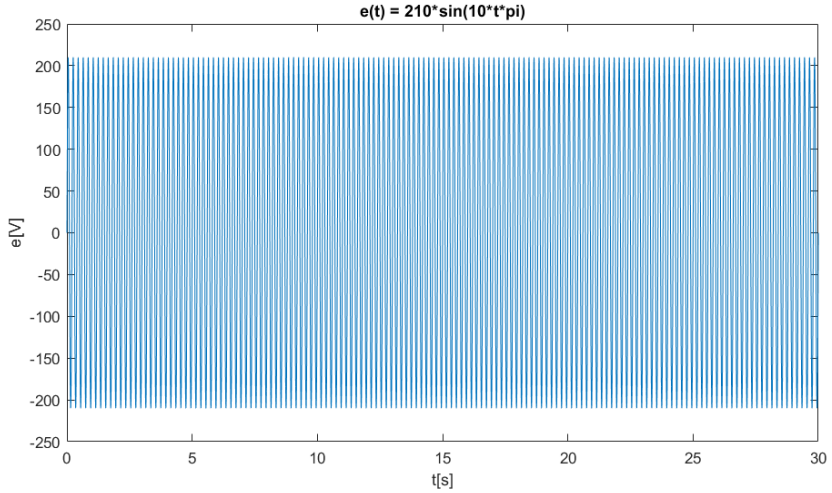
Rys. 5. Przebiegi (kolejno): wymuszenia prostokątnego, prądów I1, I2, napięcia Uc prostą metodą Eulera oraz zmodyfikowaną metodą Eulera.

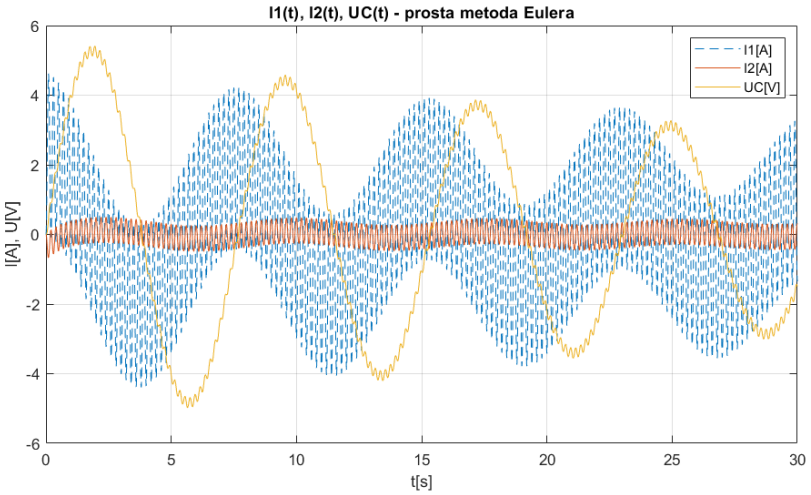


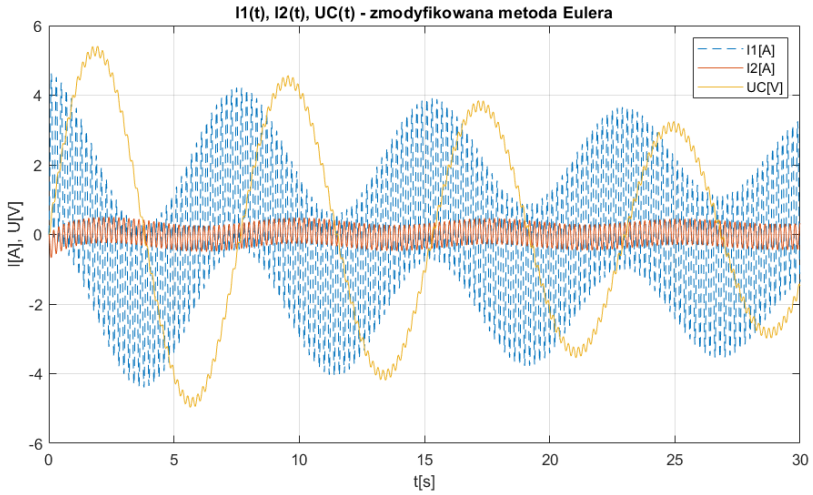




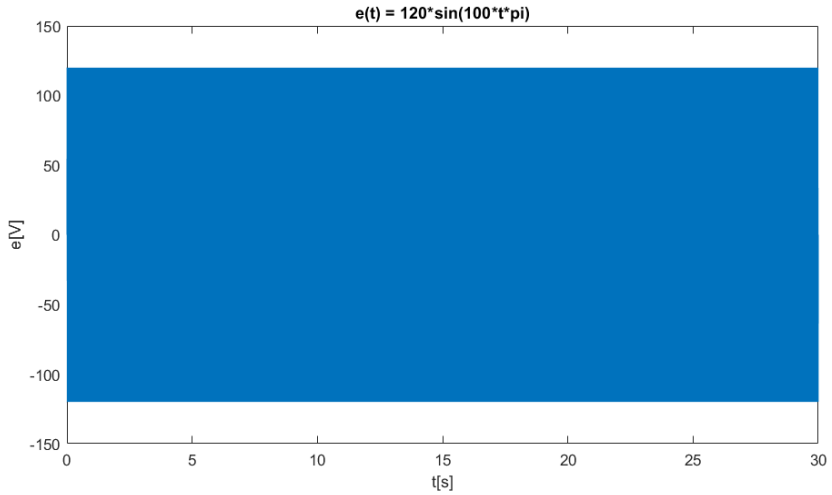
Rys. 6. Przebiegi (kolejno): wymuszenia e(t)=240sin(t), prądów I1, I2, napięcia Uc prostą metodą Eulera oraz zmodyfikowaną metodą Eulera.

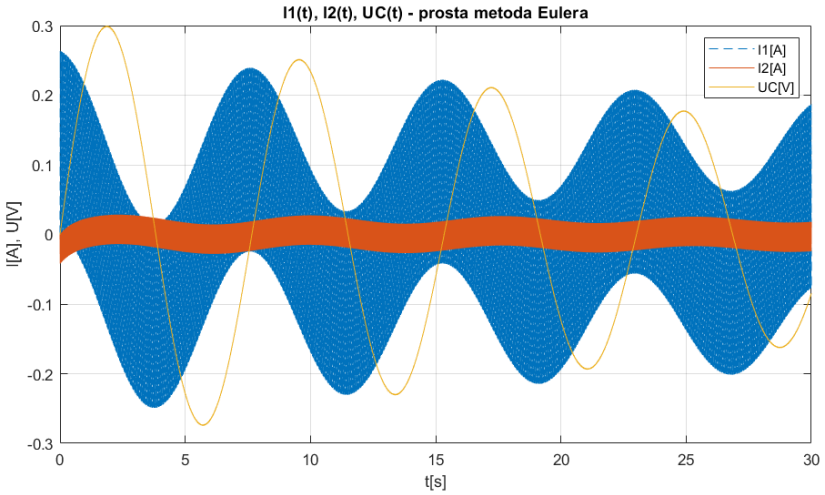


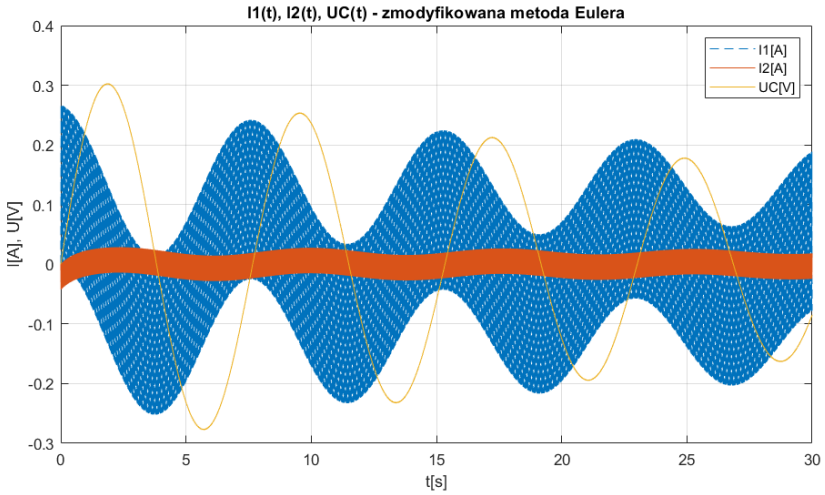




Rys. 7. Przebiegi (kolejno): wymuszenia e(t)=210 sin(2πft) dla f=5Hz, prądów I1, I2, napięcia Uc prostą metodą Eulera oraz zmodyfikowaną metodą Eulera.







Rys. 8. Przebiegi (kolejno): wymuszenia e(t)=120sin(2πft) dla f=50Hz prądów I1, I2, napięcia Uc prostą metodą Eulera oraz zmodyfikowaną metodą Eulera.

Część 2

Wstęp teoretyczny

Założono, że sprzężenie między obwodami pierwotnym i wtórnym jest nieliniowe zgodnie z tabelą:

$u_{L_1,j}$ [V]	20	50	100	150	200	250	280	300
M_j [H]	0.46	0.64	0.78	0.68	0.44	0.23	0.18	0.18

W ramach ćwiczenia należy porównać przebiegi prądów i napięć stosując do przybliżenia charakterystyki indukcyjności wzajemnej:

- interpolację wielomianową,
- interpolację funkcjami sklejanymi.
- aproksymację wielomianową z zastosowaniem wielomianu stopnia 3,
- aproksymację wielomianową z zastosowaniem wielomianu stopnia 5.

W celu otrzymania ciągłej charakterystyki indukcyjności wzajemnej, dla $U_L = 0[V]$, eksperymentalnie dobrano wartość indukcji dla tego punktu $M = 0.3[H]$.

W części 2., jak już wspomniano wyżej, ze względu na większą dokładność, zastosowano zmodyfikowaną metodę Eulera (opisaną w cz.1.).

Interpolacja wielomianowa pobiera zmienne x, y, xp – w danym przypadku to U_L. M_j oraz napięcie na oporniku nr 2 $U_{R2} = I_2 * R_2$. Aby uniknąć dzielenia przez zero w funkcji bazowej - zgodnie z metodą Lagrange'a - zakładamy warunek dla iteratorów „jeśli k nie jest równe j". Na początku założona jest liniowość poza granicami napięcia na cewce U_L. Następnie obliczana jest funkcja bazowa p. Ostatecznie funkcja interpolująca zwraca sumę iloczynów funkcji bazowej i odpowiadającej wartości szukanej napięcia U_{R2} wartości indukcyjności wzajemnej. Jest to wartość wielomianu interpolującego LaPoly dla danego punktu.

```
function LaPoly = Lagrange_Polynomial(x, y, xp)
    LaPoly=0;
    kn = length(x);
    for k=1:kn
        p=1;
        for j=1:kn
            if k~= j
                    if xp > x(end)
                        xp=x(end);
                    end
                    if xp < x(1)
                        xp=x(1);
                    end
                p = p.*(xp-x(j))./(x(k)-x(j));
            end
        end
        LaPoly =  LaPoly + y(k)*p;
    end
end
```

Rys. 9. Implementacja interpolacji metodą wielomianów Lagrange'a.

Interpolacja funkcjami sklejanymi została przeprowadzona za pomocą splajnów 1. stopnia zgodnie z przykładem ze skryptu. Dobór stopnia podyktowany był nierównoodległymi punktami oraz brakiem danych odnośnie pochodnych na krańcach przedziału. Funkcje sklejane 1 stopnia pozwalają na połączenie punktów łamaną, a następnie odczytanie wartości interpolowanej za pomocą relacji liniowej. Zaimplementowane to zostało za pomocą macierzowej metody rozwiązywania układów równań. Warunek początkowy if-else pozwala na określenie liniowej zależności indukcyjności wzajemnej poza granicami przedziałów.

```
function S = Spline_interp(x, y, xp)        %1st order
    k=1;
    if xp > x(end-1)
        k=length(x)-1;
    else
        while xp >= x(k+1)
            k=k+1;
        end
    end
    A=[1 x(k);
       1 x(k+1)];
    B=[y(k);
       y(k+1)];
    X=A\B;

    S = X(1)+X(2)*xp;
end
```

Rys. 10. Implementacja interpolacji funkcjami sklejanymi 1. stopnia

Aproksymacja wielomianowa została zaimplementowana dla dowolnego stopnia - o. Poza stopniem, funkcja pobiera parametry takie, jak interpolacja wielomianowa oraz interpolacja splajnami. Tworzony jest wyznacznik Vandermonde'a, a następnie macierz współczynników wielomianu A. Następnie w zmiennej p zapisywana jest postać wielomianu aproksymującego dla przekazanego parametru xp (w danym przypadku U_{R2}).

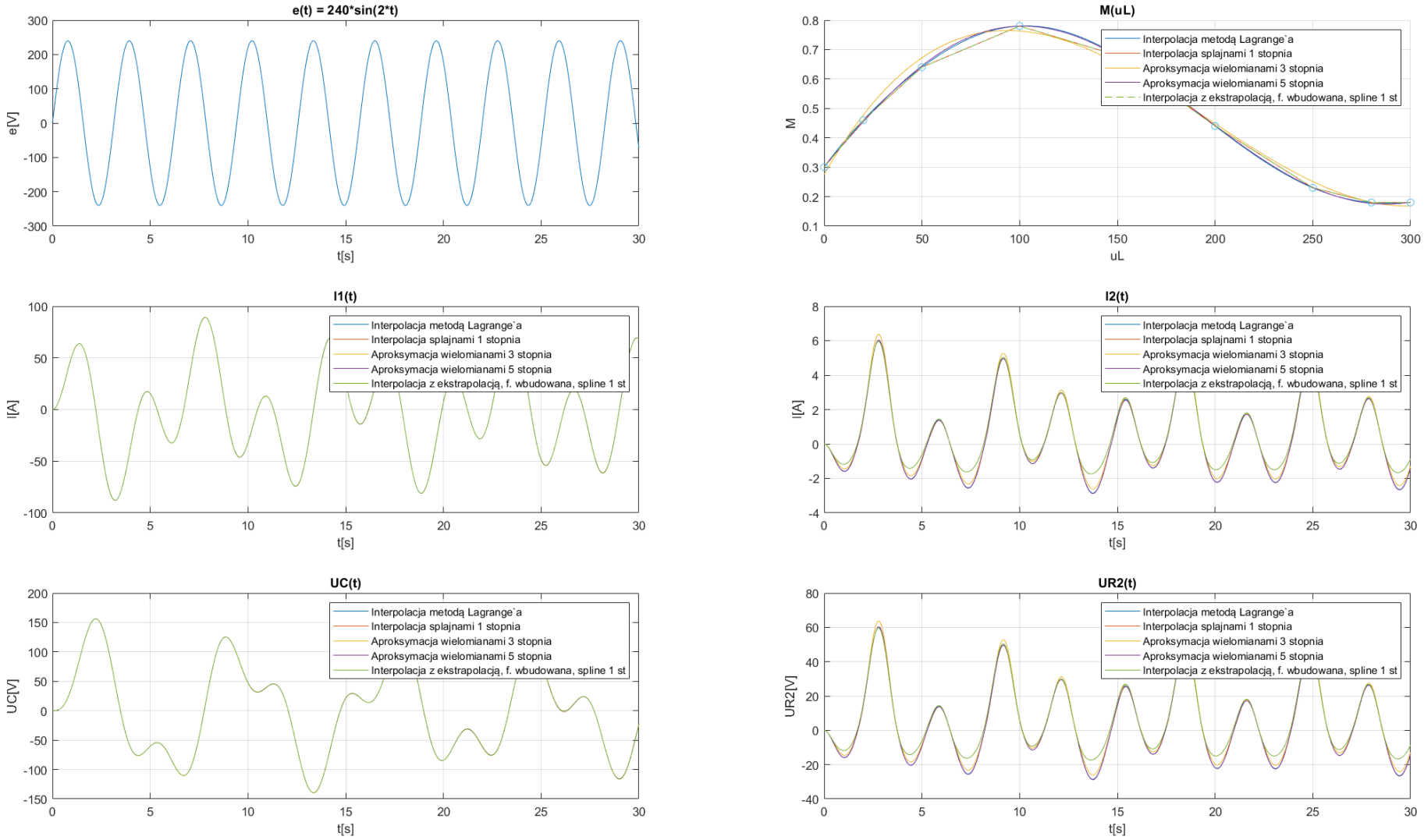
```
function p = Poly_approximation(x, y, xp, o)
   M=ones(length(x), o+1);
    for k=1:length(x)
        for j=2:o+1
            M(k,j)=M(k,j-1)*x(k);
        end
    end
    %x = A\B solves the system of linear equations A*x = B.
    Y = y';
    MTM = M \ M;
    MTY = M \ Y;
    A = MTM \ MTY;


    p=A(1);
    for k=2:o+1
        if xp > x(end)
            xp=x(end);
        end
        if xp < x(1)
            xp=x(1);
        end
        p=p+A(k).*xp.^(k-1);
    end
end
```

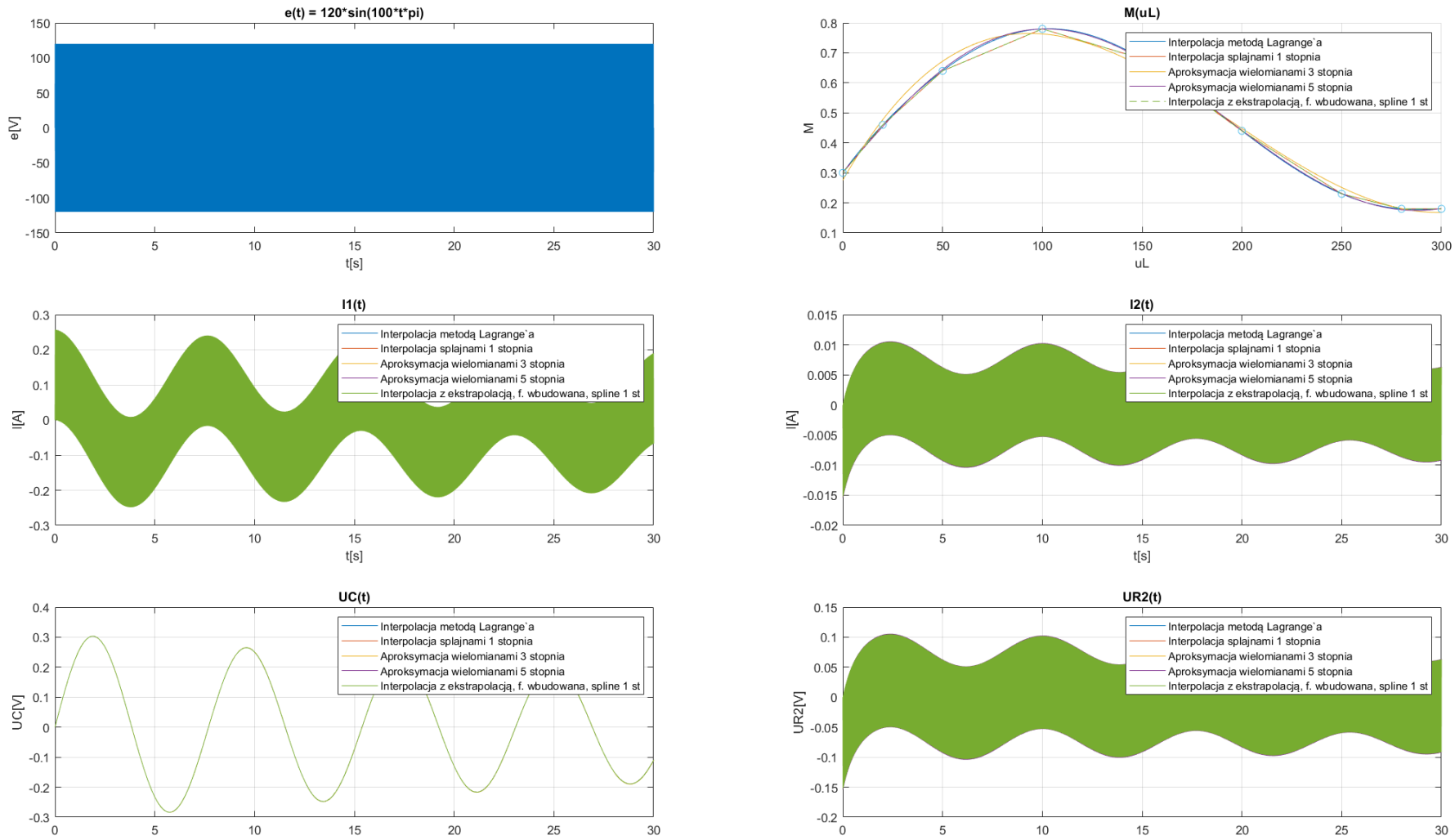
Rys. 11. Implementacja aproksymacji wielomianowej dla dowolnego stopnia

Analiza wyników

W celu sprawdzenia poprawności zastosowanych metod, użyto wbudowanej funkcji interpolującej funkcjami sklejanymi 1 stopnia wraz z ekstrapolacją. Jak widać na wykresach (rys. 12., rys.13.), przebieg funkcji jest poprawny. Jedyne różnice, jakie można zauważyć, są dla niewielkich wartości – widoczne na wykresach $I_2(t)$. Możliwe różnice wartości mogą wynikać z dokładności metod oraz z różnicy między aproksymacją a interpolacją. Różnica ta polega na tym, że interpolując funkcję, dopasowuje się ją tak, aby przechodziła przez wszystkie punkty pomiarowe – możliwe jest wtedy zaobserwowanie efektu nadmiernego dopasowania. Aproksymacją dobiera się tak funkcję, aby przechodziła możliwie jak najbliżej punktów pomiarowych, lecz niekoniecznie przez nie. Dla ujemnych skrajnych wartości $I_2(t)$ widać największą różnicę między wbudowaną metodą interpolującą, a zaimplementowanymi metodami. Prawdopodobnie jest to spowodowane założeniem liniowości w funkcjach zaimplementowanych na krańcach przedziałów. Dla dodatnich skrajnych wartości największe odchylenie jest dla aproksymacji wielomianowej 3.stopnia - wynika to z mniejszego dopasowania funkcji aproksymującej, co widać na wykresie $M(U_L)$. Zaimplementowana interpolacja funkcjami sklejanymi 1.stopnia całkowicie pokrywa się z interpolacją wbudowaną, co widać na wykresie $M(U_L)$.



Rys. 12 Przebiegi (kolejno): wymuszenia e(t)=240sin(2t), interpolacji/aproksymacji indukcyjności wzajemnej, prądów I1, I2, napięcia Uc oraz UR2.



Rys. 13. Przebiegi (kolejno): wymuszenia e(t)=120sin(2πft) dla f=50Hz, interpolacji/aproksymacji indukcyjności wzajemnej, prądów I1, I2, napięcia Uc oraz UR2.

Część 3

Wstęp teoretyczny

W części 3 należy wyznaczyć ciepło (moc czynną wydzielaną na obu rezystorach w układzie), które absorbuje nasz obwód pasywny z wykorzystaniem elementu liniowego oraz nieliniowego dla tych samych funkcji wymuszeń w stanie nieustalonym w czasie: 0 < t < 30 [s].

Aby policzyć ciepło na obu rezystorach, należy obliczyć całkę oznaczoną po czasie z sumy iloczynów napięć i natężeń dla każdego rezystora.

$$P = \int_{t=0}^{30\ [\mathrm{s}]} \left(R_1 i_1^2(t) + R_2 i_2^2(t)\right) \mathrm{d}t$$

W tym celu zaimplementowano metodę złożoną prostokątów oraz metodę złożoną kwadratów dla dwóch kroków całkowania ($\Delta t_1 = 0.005$ - duży krok; $\Delta t_2 = 0.0005$ – mały krok) oraz dwóch charakterystyk indukcyjności wzajemnej (liniowej i nieliniowej). Dobór wielkości kroku był podyktowany stabilnością przebiegu – dla kroku $\Delta t_1 = 0.01$ przebieg $e(t) = 120\sin(2\pi f t)\,, dla\ f = 50Hz$ przy nieliniowej charakterystyce indukcyjności wzajemnej był niestabilny. Z tego powodu zdecydowano się na zmniejszenie kroku o połowę; natomiast różnica o rząd między małym a dużym krokiem pozwala na wyciągnięcie wniosków odnośnie dokładności metody. Zdecydowano się na obliczenie nieliniowej indukcyjności wzajemnej metodą aproksymacji wielomianowej 5. stopnia.

Metodami złożonymi prostokątów i parabol obliczono moc chwilową, a następnie, aby uzyskać moc całkowitą, zsumowano ją

W metodzie prostokątów pobierana jest wartość dla danego punktu, a następnie mnożona przez krok całkowania.

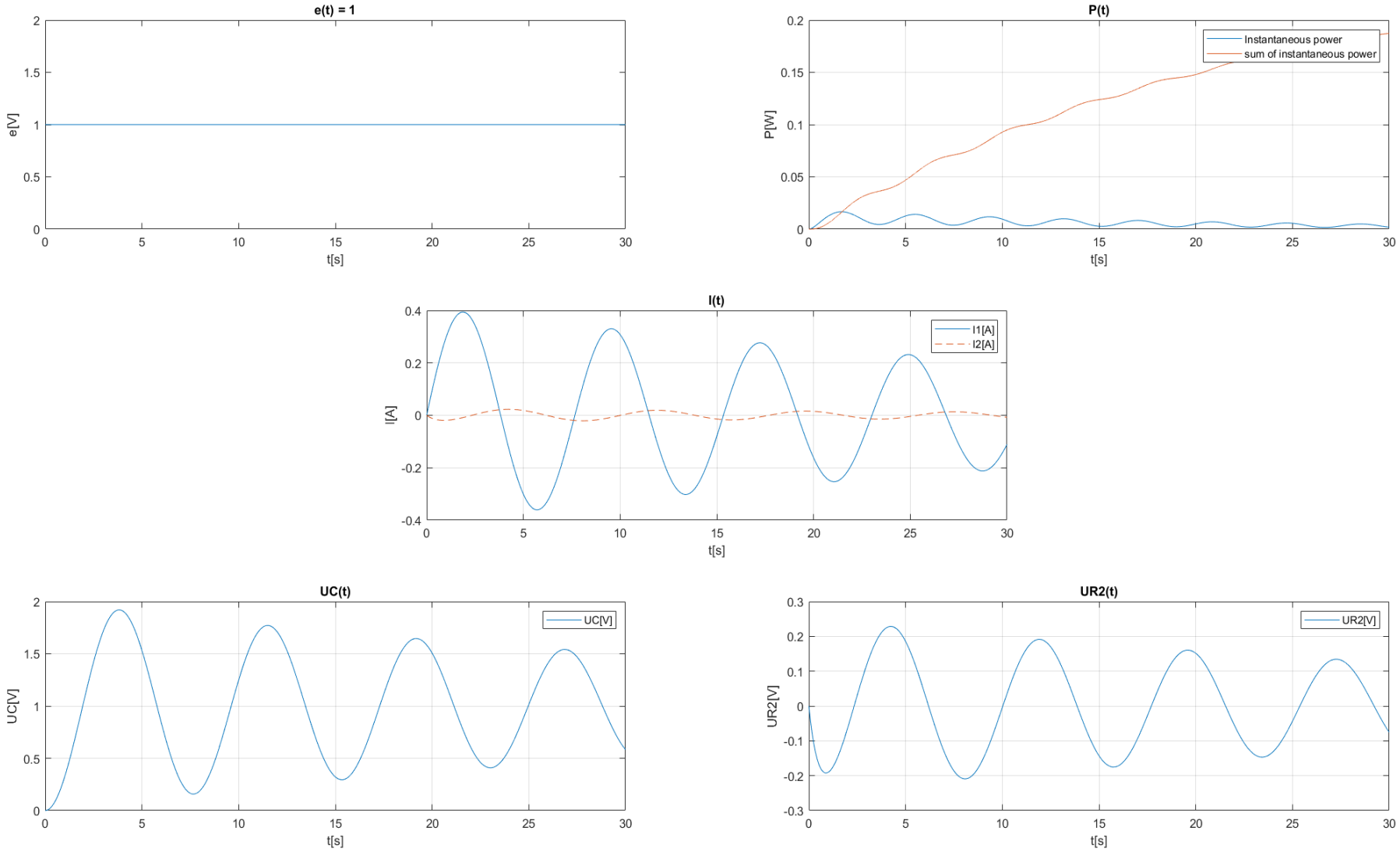
```
function I = int_r(f, h)
    l=length(f);
    I=zeros(1,l);
    for o=1:l
        I(1,o)=f(o)*h;
    end
end
```

Rys. 14. Implementacja metody złożonej prostokątów.

```
function I = int_p(f,h)
    l=length(f);
    I=zeros(1,l);
    for o=2:2:length(f)-1
        I(o)=2*(f(o))*h/3;
    end
    for o=3:2:length(f)-1
        I(o)=4*(f(o))*h/3;
    end
    I(1)=f(1)*h/3;
    I(end)=f(end)*h/3;
end
```

Rys. 15 Implementacja metody złożonej parabol.

W celu sprawdzenia poprawności metody, oszacowano (poprzez policzenie iloczynu z połowy przedziału osi x oraz wartości maksymalnej wartości pierwszego piku), że wartość całki dla przebiegu $e(t) = 1[V]$ nie powinna przekroczyć $0.015 * 15 = 0.225$, co zgadza się z wynikiem obliczonym numerycznie. Naniesiono na wykresie $P(t)$ wartość całki dla danej chwili (rys. 16). Pozwoliło to ocenić, iż metody zostały zaimplementowane w poprawny sposób.



Rys. 16. Przebiegi (kolejno): wymuszenia e(t)=1[V], mocy chwilowej oraz przebiegu całki z mocy chwilowej, prądów I1, I2, napięcia Uc oraz UR2 dla liniowej indukcyjności wzajemnej.

Analiza wyników

Poniżej w tabelach przedstawiono wyniki otrzymane powyższymi metodami. Można zauważyć, że nieliniowa charakterystyka indukcyjności wzajemnej ma wpływ na zmniejszenie wydzielanego ciepła. Największe różnice dla zastosowanych metod szacowania całki widoczne są dla wymuszenia $e(t) = 120\sin(2\pi ft)\,, dla\ f = 50Hz$, zarówno dla liniowego, jak i nieliniowego przebiegu indukcyjności wzajemnej.

Tabela 1. Wartości całek policzonych metodami złożonymi prostokątów i parabol dla liniowej wartości indukcyjności.

Wymuszenie	Metoda złożona prostokątów		Metoda złożona parabol	
	Δt_1	Δt_2	Δt_1	Δt_2
$e(t) = 1[V]$	0.1873	0.1873	0.1873	0.1873
$E = \begin{cases} 120\ dla\ t < \frac{T}{2} \\ 0\ dla\ t \geq \frac{T}{2} \end{cases} [V], \quad T = 3[s];$	1.7878e+03	1.7877e+03	1.7876e+03	1.7877e+03
$e(t) = 240\sin(t)$	2.1242e+05	2.1241e+05	2.1241e+05	2.1241e+05
$e(t) = 210\sin(2\pi ft)\,, dla\ f = 5Hz$	34.8111	34.7373	34.8061	34.7368
$e(t) = 120\sin(2\pi ft)\,, dla\ f = 50Hz$	0.1404	0.1138	0.1793	0.1138

Tabela 2. Wartości całek policzonych metodami złożonymi prostokątów i parabol dla nieliniowej wartości indukcyjności.

Wymuszenie	Metoda złożona prostokątów		Metoda złożona parabol	
	Δt_1	Δt_2	Δt_1	Δt_2
$e(t) = 1[V]$	0.1653	0.1653	0.1653	0.1653
$E = \begin{cases} 120\ dla\ t < \frac{T}{2} \\ 0\ dla\ t \geq \frac{T}{2} \end{cases} [V], \quad T = 3[s];$	1.3160e+03	1.3159e+03	1.3159e+03	1.3159e+03
$e(t) = 240\sin(t)$	1.8787e+05	1.8786e+05	1.8786e+05	1.8786e+05
$e(t) = 210\sin(2\pi ft)\,, dla\ f = 5Hz$	15.9521	15.9185	15.9509	15.9184
$e(t) = 120\sin(2\pi ft)\,, dla\ f = 50Hz$	0.0638	0.0517	0.0785	0.0517

Część 4

Wstęp teoretyczny

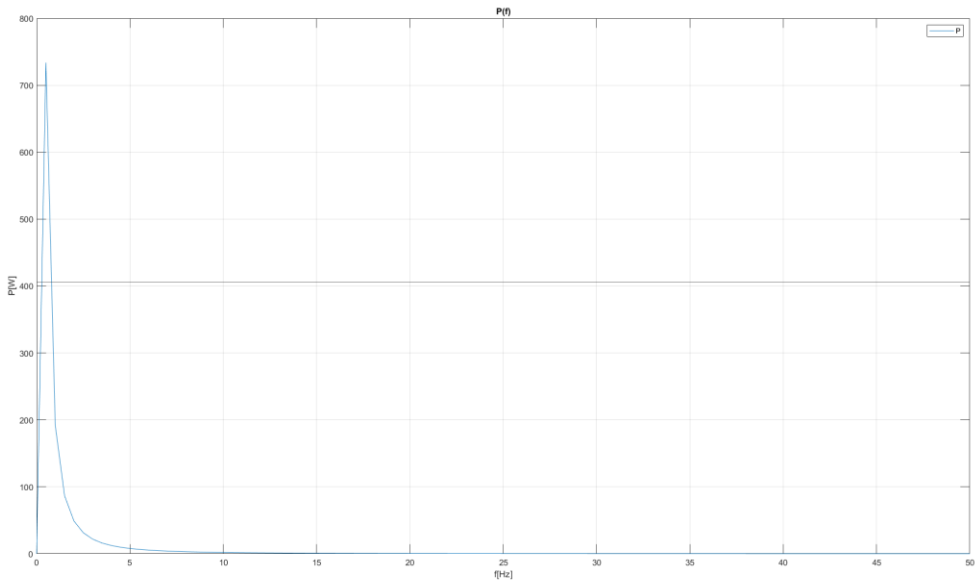
Cel części 4 projektu to napisanie symulatora w programie MATLAB, który będzie wyznaczać częstotliwość f sygnału wymuszającego. Całkowita moc czynna wydzielająca się na rezystancjach R1 i R2 ma być równa wartości P = 406 [W] przy założeniu wymuszenia:

$$e(t) = 100sin(2\pi ft)$$

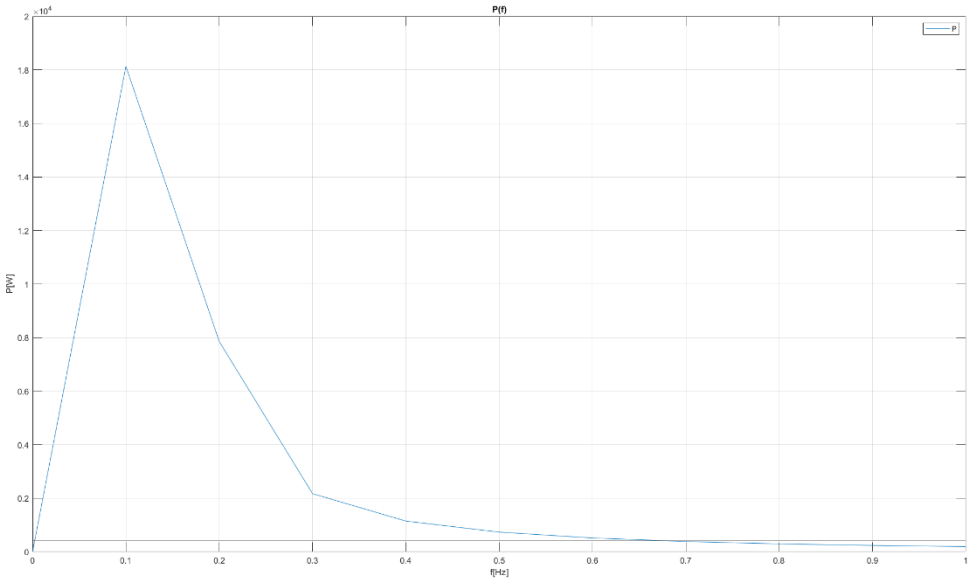
Aby rozwiązać problem, należy sformułować nieliniową funkcję celu w postaci:

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

Aby uzyskać spodziewany przebieg funkcji, w celu izolacji pierwiastków, określono wektor F o krańcach $f_1 = 0[Hz]$, $f_2 = 200[Hz]$ oraz obliczono metodą zmodyfikowaną Eulera wartości mocy w tym przedziale z krokiem $h_f = 0.3$. Otrzymany wykres przedstawiono na rys. 17. Ze względu na istniejące pierwiastki jedynie w przedziale $f \in < 0; 1 >$ zdecydowano się zawęzić dziedzinę funkcji. Wygenerowano wektor wartości f z krokiem 0.1 w celu zwiększenia szybkości działania funkcji znajdujących pierwiastki (zmniejszenie przedziałów powoduje mniejszą liczbę iteracji oraz wywołań funkcji obliczającej wartość mocy P). Otrzymany wykres przedstawiono na rys. 18.



Rys. 17. Spodziewany przebieg funkcji P(f) dla zakresu $f \in < 0; 50 >$



Rys. 18. Spodziewany przebieg funkcji P(f) po zawężeniu przedziału $f \in < 0; 1 >$

Następnie, aby znaleźć miejsca zerowe funkcji, zaimplementowano metody:

- bisekcji,
- siecznych,
- quasi-Newtona.

W celu określenia efektywności metod do funkcji dodano licznik iteracji potrzebny do znalezienia rozwiązania poszczególnymi metodami oraz licznik wywołań funkcji P.

Implementację funkcji obliczającej wartość mocy przedstawiono na rys. 19. Najpierw wywoływana jest metoda Eulera w celu obliczenia wartości prądów I_1, I_2. Następnie moc chwilowa obliczana jest za pomocą wzoru analogicznie jak w części 3:

$$P = \int_{t=0}^{30\ [\mathrm{s}]} \left(R_1 i_1^2(t) + R_2 i_2^2(t)\right) \mathrm{d}t$$

Całka obliczana jest zaimplementowaną tak, jak w części 3 metodą parabol dla kroku całkowania h=0.001. Od sumy z mocy chwilowej odejmuje się wartość p_0 = 406W, by uzyskać rozwiązanie równania:

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

```
function ipsum = P0(x_i, h, y, p_0)
    syms t;
    E=@(t) 100*sin(2*pi*x_i*t);
    a = 0;
    b = 30;
    t = a:h:b;      %[s]
    E(t);
    y1=modified_euler(t, y, h, E);
    Pinst(1,:)=(y1(1,:).^2).*R1+(y1(2,:).^2).*R2;
    Ip = int_p(Pinst, h);
    ipsum=sum(Ip)-p_0;
end
```

Rys. 19. Funkcja obliczająca wartość mocy.

W metodzie bisekcji (rys. 20) importowane są: przedziały, wartości mocy dla przedziałów (P-406), dokładność, z jaką ma być obliczany pierwiastek oraz parametry niezbędne do wywołania funkcji obliczającej moc P – wartość początkowa y, krok metody Eulera oraz wartość p_0 = 406W.

Najpierw przyjmuje się wartości przedziałów oraz ich wartości na osi y. Jeśli iloczyn wartości na osi y jest dodatni, oznacza to, że na krańcach przedziałów funkcja ma taki sam znak. Przy małych zakresach przedziałów oraz wspomagając się wykresem pomocniczym (rys. 18) można wnioskować, że funkcja w całym przedziale nie zmienia wartości. Jeśli warunek ten nie jest spełniony, funkcja przechodzi do pętli nieskończonej pętli while. Każda iteracja powoduje zwiększenie licznika *counter* o 1. Przedział dzielony jest na pół, następnie jeśli wartość połowy przedziału jest mniejsza od określonej wielkości błędu, pętla przerywa się. Jeśli nie, to dalej obliczana jest wartość mocy dla połowy przedziału, licznik wywołań funkcji P zwiększa się o 1. Sprawdzane są warunki odnośnie wartości P dla tego miejsca – jeśli są mniejsze od określonej wartości błędu, to przyjmuje się to miejsce jako miejsce zerowe i przerywa funkcję. Jeśli nie, to dalej obiera się nowe krańce tak, aby znaki na obu krańcach były różne. Nieskończona pętla while kończy się tylko wtedy, gdy przerwie ją komenda break.

Po zakończeniu pętli while, znalezione miejsce zerowe dodawane jest do wektora miejsc zerowych x0 na ostatnie miejsce. Drukowana jest wartość znalezionego miejsca zerowego oraz liczba powtórzeń pętli while potrzebna do znalezienia danego pierwiastka.

Po zakończeniu pętli for dla każdego przedziału drukowana jest wartość liczników wywołań funkcji P potrzebnych do znalezienia wszystkich miejsc zerowych oraz całkowita wartość powtórzeń pętli while.

```
function x_0 = bisection(interv, p, epsx, epsy, y, h, p_0)
    x_0=[];
    Pcounter=0;
    bcounter=0;
        for i=1:length(interv)-1
            ai=interv(i);
            bi=interv(i+1);
            fa=p(i);
            fb=p(i+1);
            fprintf('Interval:')
            fprintf('\t %.2f', ai, bi)
            fprintf('\n')
            if (fa * fb > 0)
                disp('The same sign at endpoints of the interval')
            else
                counter=0;
                while(true)
                    counter=counter+1;
                    x_i=(ai+bi)/2;
                    if abs(ai-x_i) < epsx
                        break;
                    end

                    f_div=P0(x_i, h, y, p_0);
                    Pcounter=Pcounter+1;

                    if abs(f_div) < epsy
                        break;
                    end
                    if (fa*f_div < 0)
                        bi = x_i;
                    else
                        ai = x_i;
                        fa=f_div;
                    end
                end
                bcounter=bcounter+counter;
                x_0(end+1)=x_i;
                fprintf('Root:\t\t')
                fprintf('%.5f \n',x_i)
                fprintf('Iteration counter per root: %.f \n',counter)
            end
            fprintf('_____________________________________________\n\n')
        end
    fprintf('P-function counter per bisection function: %.f \n',Pcounter)
    fprintf('Iteration counter per bisection function: %.f \n',bcounter)
end
```

Rys. 20. Implementacja metody bisekcji

W metodzie siecznych (rys. 22) pobierane są takie same parametry, jak w metodzie bisekcji. Najpierw zerowane są liczniki wywołań funkcji P oraz licznik iteracji dla całego rozwiązania. Następnie przypisywane są krańce przedziałów i ich wartości do zmiennych an, fa oraz ab, fb. Pierwszy warunek sprawdza znak na krańcach przedziału – jeśli jest różny, funkcja przechodzi do sprawdzenia warunków iloczynu wartości na krańcach i drugiej pochodnej tej wartości. Powinien być on dodatni. Obliczana pochodna została zaimplementowana numerycznie tak, jak na rys. 21.

```
function f_diff = fdiff(f, df,h,y,p_0)
    f_diff = (P0(f+df,h,y,p_0)-P0(f,h,y,p_0))/df;
end
```

Rys. 21. Implementacja numerycznie liczonej pochodnej funkcji.

W każdej pochodnej funkcja P wywoływana jest dwukrotnie, stąd licznik wywołań funkcji P jest zwiększony o 4. Jeśli iloczyn drugiej pochodnej i wartości funkcji dla któregoś z końców jest dodatni, jest ustalany on jako punkt startowy do liczenia siecznych, jeśli nie, pętla zostaje przerwana. Dalej zerowany jest licznik iteracji dla każdego pierwiastka. Po rozpoczęciu funkcji while jest on inkrementowany, a następnie dodano warunek: jeśli kraniec przedziału wynosi zero, to zwiększamy go o 0.01, aby uniknąć wejścia w ujemne wartości dziedziny w następnym kroku - pobierana jest wartość krańca przedziału i wartość poprzedzająca ten kraniec o df – krok różniczkowania. Przez te dwa początkowe punkty prowadzona jest sieczna, która przecina oś X. Punkt przecięcia wyznacza kolejną iterację, dopóki wartość dla tego punktu nie wynosi prawie 0 (dokładność epsy). Dla każdego wywołania funkcji P licznik jest inkrementowany. Tak jak wyżej, po zakończeniu pętli while, znalezione miejsce zerowe dodawane jest do wektora miejsc zerowych x0 na ostatnie miejsce. Drukowana jest wartość znalezionego miejsca zerowego oraz liczba powtórzeń pętli while potrzebna do znalezienia danego pierwiastka.

Po zakończeniu pętli for dla każdego przedziału drukowana jest wartość liczników wywołań funkcji P potrzebnych do znalezienia wszystkich miejsc zerowych oraz całkowita wartość powtórzeń pętli while.

Metoda quasi-Newtona (rys. 23) jest metodą Newtona, dla której pochodną oblicza się numerycznie zgodnie z funkcją przedstawioną na rys. 21. Tak jak w metodzie siecznych prowadzona jest sieczna, tak w metodzie Newtona dla wartości jednego punktu prowadzona jest styczna, która przecina oś X, wyznaczając tym samym kolejną iterację. Kolejność kroków jest taka, jak w metodzie siecznych.

```
function x_0 = secant(interv, p, epsx, epsy, y, h, p_0)
    x_0=[];
    Pcounter=0;
    bcounter=0;
    for i=1:length(interv)-1
        an=interv(i);
        bn=interv(i+1);
        fa=p(i);
        fb=p(i+1);
        fprintf('Interval:')
        fprintf('\t %.2f', an, bn)
        fprintf('\n')
        if (fa * fb > 0)
            disp('The same sign at endpoints of the interval')
        else
            if fa*(fdiff(fdiff(an, df,h,y,p_0),df,h,y,p_0)) > 0
                Pcounter=Pcounter+4;
                x_i = an;
                fi=fa;
            else
                x_i = bn;
                fi=fb;
            end
            counter=0;
            while(true)
                counter=counter+1;
                if x_i == 0
                    x_i = 0.01;
                end
                x_h=x_i-df;
                x_j=x_i-(fi*(x_i-x_h)/(P0(x_i, h, y, p_0)-P0(x_h, h, y, p_0)));
                fj=P0(x_j, h, y, p_0);
                Pcounter=Pcounter+3;
                if abs(x_j-x_i) < epsx
                    break;
                end
                if abs(fj) < epsy
                    break;
                else
                    x_i=x_j;
                    fi=P0(x_i, h, y, p_0);
                    Pcounter=Pcounter+1;
                end
            end
            x_0(end+1)=x_i;
            fprintf('Root:\t\t')
            fprintf('%.5f \n',x_i)
            fprintf('Iteration counter per root: %.f \n',counter)
            bcounter=bcounter+counter;
        end
        fprintf('_____________________________________________________\n\n')
    end
    fprintf('P-function counter per secant function: %.f \n',Pcounter)
    fprintf('Iteration counter per secant function: %.f \n',bcounter)
end
```

Rys. 22. Implementacja metody siecznych

```
function x_0 = q_newton(interv, p, epsx, epsy,  y, h, p_0)
x_0=[];
Pcounter=0;
bcounter=0;
counter = 0;
    for i=1:length(interv)-1
        an=interv(i);
        bn=interv(i+1);
        fa=p(i);
        fb=p(i+1);
        fprintf('Interval:')
        fprintf('\t %.2f', an, bn)
        fprintf('\n')
        if (fa * fb > 0)
            disp('The same sign at endpoints of the interval')
        else
            if fa*(fdiff(fdiff(an, df,h,y,p_0),df,h,y,p_0)) > 0
                Pcounter=Pcounter+4;
                x_i = an;
                fi=fa;
            else
                x_i = bn;
                fi=fb;
            end
            counter=0;
            while(true)
                counter=counter+1;
                if x_i == 0
                    x_i = 0.01;
                end
                x_j=x_i-(fi/fdiff(x_i, df,h,y,p_0));
                Pcounter=Pcounter+2;
                fj=P0(x_j, h, y, p_0);
                Pcounter=Pcounter+1;
                if abs(x_j-x_i) < epsx
                    break;
                end
                if abs(fj) < epsy
                    break;
                else
                    x_i=x_j;
                    fi=P0(x_i, h, y, p_0);
                    Pcounter=Pcounter+1;
                end
            end
            bcounter=bcounter+counter;
            x_0(end+1)=x_i;
            fprintf('Root:\t\t')
            fprintf('%.5f \n',x_i)
            fprintf('Iteration counter per root: %.f \n',counter)
        end
        fprintf('_____________________________________________________\n\n')
    end
    fprintf('P-function counter per quasi-Newton function: %.f \n',Pcounter)
    fprintf('Iteration counter per quasi-Newton function: %.f \n',bcounter)
end
```

Rys. 23. Implementacja metody quasi-Newtona.

Analiza wyników

W każdej metodzie znaleziono dwa pierwiastki o takich samych wartościach dla każdej metody: 0.03534 oraz 0.67563. W celu porównania poszczególnych metod, dokonano zliczenia koniecznych wywołań funkcji P, iteracji oraz czasu trwania obliczeń dla każdego rozwiązania. Wyniki zamieszczono w tabeli 3.

Tabela 3. Porównanie liczby iteracji, wywołań funkcji P oraz czasu potrzebnego do wykonania obliczeń przy znajdywaniu pierwiastków dla metody bisekcji, siecznych i quasi-Newtona dla przedziałów dziedziny (krok 0.1)

	Metoda bisekcji		**Metoda siecznych**		**Metoda quasi-Newtona**	
	Iteracje	**P**	**Iteracje**	**P**	**Iteracje**	**P**
	34	32	15	62	18	74
Czas [s]	11.885313		25.551841		29.392678	

Pomimo zmniejszenia liczby iteracji, liczba wywołań funkcji P zwiększa się – powodem jest liczenie pochodnych – każdorazowe wywołanie funkcji liczącej pochodną zwiększa licznik wywołań funkcji P o 2. Do zmierzenia czasu potrzebnego na znalezienie rozwiązania wykorzystano funkcję wbudowaną tic toc. Jak widać w tabeli 3, liczba wywołań funkcji P ma krytyczny wpływ na szybkość wykonywania metody. Pomimo tego, że metoda Newtona jest metodą szybko zbieżną, to wykonanie obliczeń pochodnych oraz funkcji P dla podanej długości przedziałów spowodowało, że znalezienie rozwiązania tą metodą trwało najdłużej. Najszybsza okazała się metoda bisekcji, która jest metodą wolno zbieżną.

Porównanie czasu trwania obliczeń w zależności od długości przedziałów dziedziny przedstawiono w tabeli 4.

Tabela 4. Zależność czasu trwania obliczeń od długości przedziałów dziedziny

Długość przedziału	**Czas trwania obliczeń**		
	Metoda bisekcji	**Metoda siecznych**	**Metoda quasi-Newtona**
0.1	11.885313	25.551841	29.392678
0.25	13.026943	26.715562	31.400778
0.5	13.539215	26.777169	30.993414

Wydłużenie długości przedziałów dziedziny spowodowało, że czas potrzebny na znalezienie pierwiastków wydłużył się – z wyjątkiem największego kroku dla metody quasi-Newtona, gzie znalezienie pierwiastków trwało krócej niż dla kroku 0.25. Niestety nie ma możliwości dalszego zwiększania przedziałów w celu przeprowadzenia analizy, jak długość kroku wpływa na szybkość obliczeń ze względu na bliskie położenie względem siebie obu pierwiastków.