

mypice

1.0

Wygenerowano przez Doxygen 1.8.20

1 myspice - symulator obwodów liniowych	1
1.0.1 Wersja podstawowa	1
1.0.2 Wersja rozszerzona	2
1.0.3 Zasada działania programu	2
2 Wersja rozszerzona	3
2.0.1 Przykłady	4
2.0.1.1 Dzielnik rezystorowy	4
2.0.1.2 Dolnoprzepustowy filtr Sallen-Key II-rzędu	4
2.0.1.3 Układ rezonansowy RLC	6
3 Sprawdzenie poprawności	9
3.0.1 myspice	9
3.0.2 LTSpice	10
3.0.3 ngspice	10
4 Indeks przestrzeni nazw	13
4.1 Lista przestrzeni nazw	13
5 Indeks hierarchiczny	15
5.1 Hierarchia klas	15
6 Indeks klas	17
6.1 Lista klas	17
7 Indeks plików	19
7.1 Lista plików	19
8 Dokumentacja przestrzeni nazw	21
8.1 Dokumentacja przestrzeni nazw mna	21
8.1.1 Opis szczegółowy	21
9 Dokumentacja klas	23
9.1 Dokumentacja struktury ac_analysis_params	23
9.1.1 Opis szczegółowy	24
9.1.2 Dokumentacja atrybutów składowych	24
9.1.2.1 exponent	24
9.1.2.2 start	24
9.1.2.3 steps	24
9.1.2.4 stop	24
9.2 Dokumentacja struktury mna::admittance	25
9.2.1 Opis szczegółowy	25
9.2.2 Dokumentacja atrybutów składowych	25
9.2.2.1 nodes	25
9.2.2.2 Y	25

9.3 Dokumentacja struktury <code>bipole_component</code>	26
9.3.1 Opis szczegółowy	27
9.3.2 Dokumentacja konstruktora i destruktora	27
9.3.2.1 <code>bipole_component()</code>	27
9.3.3 Dokumentacja atrybutów składowych	27
9.3.3.1 <code>nodes</code>	27
9.4 Dokumentacja struktury <code>capacitor</code>	27
9.4.1 Opis szczegółowy	29
9.4.2 Dokumentacja konstruktora i destruktora	29
9.4.2.1 <code>capacitor()</code>	29
9.4.3 Dokumentacja funkcji składowych	29
9.4.3.1 <code>admittance()</code>	29
9.4.4 Dokumentacja atrybutów składowych	29
9.4.4.1 <code>C</code>	30
9.5 Dokumentacja struktury <code>circuit_component</code>	30
9.5.1 Opis szczegółowy	30
9.5.2 Dokumentacja konstruktora i destruktora	30
9.5.2.1 <code>~circuit_component()</code>	31
9.6 Dokumentacja struktury <code>circuit_simulation</code>	31
9.6.1 Opis szczegółowy	31
9.6.2 Dokumentacja atrybutów składowych	31
9.6.2.1 <code>ac</code>	32
9.6.2.2 <code>circ</code>	32
9.6.2.3 <code>probes</code>	32
9.6.2.4 <code>title</code>	32
9.7 Dokumentacja klasy <code>circuit_solver</code>	32
9.7.1 Opis szczegółowy	33
9.7.2 Dokumentacja konstruktora i destruktora	33
9.7.2.1 <code>circuit_solver()</code>	33
9.7.3 Dokumentacja funkcji składowych	34
9.7.3.1 <code>current()</code> [1/2]	34
9.7.3.2 <code>current()</code> [2/2]	34
9.7.3.3 <code>get_node_map()</code>	35
9.7.3.4 <code>get_solution()</code>	35
9.7.3.5 <code>get_solution_omega()</code>	35
9.7.3.6 <code>power()</code> [1/2]	36
9.7.3.7 <code>power()</code> [2/2]	36
9.7.3.8 <code>solve()</code>	36
9.7.3.9 <code>update()</code>	37
9.7.3.10 <code>voltage()</code> [1/3]	38
9.7.3.11 <code>voltage()</code> [2/3]	38
9.7.3.12 <code>voltage()</code> [3/3]	38

9.8 Dokumentacja klasy <code>current_probe</code>	39
9.8.1 Opis szczegółowy	40
9.8.2 Dokumentacja konstruktora i destruktor	40
9.8.2.1 <code>current_probe()</code>	41
9.8.3 Dokumentacja funkcji składowych	41
9.8.3.1 <code>get_value()</code>	41
9.8.4 Dokumentacja atrybutów składowych	41
9.8.4.1 <code>m_probing_method</code>	42
9.8.4.2 <code>m_ref</code>	42
9.9 Dokumentacja struktury <code>mna::current_source</code>	42
9.9.1 Opis szczegółowy	42
9.9.2 Dokumentacja atrybutów składowych	43
9.9.2.1 <code>I</code>	43
9.9.2.2 <code>nodes</code>	43
9.10 Dokumentacja struktury <code>current_source</code>	43
9.10.1 Opis szczegółowy	44
9.10.2 Dokumentacja konstruktora i destruktor	44
9.10.2.1 <code>current_source()</code>	44
9.10.3 Dokumentacja atrybutów składowych	45
9.10.3.1 <code>acI</code>	45
9.10.3.2 <code>dcl</code>	45
9.11 Dokumentacja struktury <code>inductor</code>	45
9.11.1 Opis szczegółowy	46
9.11.2 Dokumentacja konstruktora i destruktor	46
9.11.2.1 <code>inductor()</code>	47
9.11.3 Dokumentacja funkcji składowych	47
9.11.3.1 <code>admittance()</code>	47
9.11.4 Dokumentacja atrybutów składowych	47
9.11.4.1 <code>L</code>	47
9.12 Dokumentacja szablonu klasy <code>matrix< T ></code>	48
9.12.1 Opis szczegółowy	49
9.12.2 Dokumentacja konstruktora i destruktor	49
9.12.2.1 <code>matrix()</code> [1/4]	49
9.12.2.2 <code>matrix()</code> [2/4]	49
9.12.2.3 <code>~matrix()</code>	49
9.12.2.4 <code>matrix()</code> [3/4]	49
9.12.2.5 <code>matrix()</code> [4/4]	50
9.12.3 Dokumentacja funkcji składowych	50
9.12.3.1 <code>at()</code> [1/2]	50
9.12.3.2 <code>at()</code> [2/2]	50
9.12.3.3 <code>data()</code>	51
9.12.3.4 <code>get_height()</code>	52

9.12.3.5 <code>get_width()</code>	52
9.12.3.6 <code>operator>()</code> [1/2]	53
9.12.3.7 <code>operator>()</code> [2/2]	53
9.12.3.8 <code>operator*()</code>	53
9.12.3.9 <code>operator+()</code>	53
9.12.3.10 <code>operator=()</code> [1/2]	54
9.12.3.11 <code>operator=()</code> [2/2]	54
9.12.3.12 <code>replace()</code>	54
9.12.3.13 <code>transpose()</code>	55
9.13 Dokumentacja struktury <code>mna::mna_problem</code>	55
9.13.1 Opis szczegółowy	56
9.13.2 Dokumentacja funkcji składowych	56
9.13.2.1 <code>solve()</code>	56
9.13.3 Dokumentacja atrybutów składowych	57
9.13.3.1 <code>admittances</code>	57
9.13.3.2 <code>current_sources</code>	57
9.13.3.3 <code>opamps</code>	57
9.13.3.4 <code>voltage_sources</code>	57
9.14 Dokumentacja klasy <code>mna::mna_solution</code>	57
9.14.1 Opis szczegółowy	58
9.14.2 Dokumentacja konstruktora i destruktora	58
9.14.2.1 <code>mna_solution()</code>	58
9.14.3 Dokumentacja funkcji składowych	58
9.14.3.1 <code>get_matrix()</code>	58
9.14.3.2 <code>opamp_current()</code>	58
9.14.3.3 <code>voltage()</code>	59
9.14.3.4 <code>voltage_source_current()</code>	59
9.15 Dokumentacja struktury <code>opamp</code>	60
9.15.1 Opis szczegółowy	61
9.15.2 Dokumentacja konstruktora i destruktora	61
9.15.2.1 <code>opamp()</code>	61
9.15.3 Dokumentacja atrybutów składowych	62
9.15.3.1 <code>neg_input_node</code>	62
9.15.3.2 <code>output_node</code>	62
9.15.3.3 <code>pos_input_node</code>	62
9.16 Dokumentacja struktury <code>mna::opamp</code>	62
9.16.1 Opis szczegółowy	63
9.16.2 Dokumentacja atrybutów składowych	63
9.16.2.1 <code>neg_input_node</code>	63
9.16.2.2 <code>output_node</code>	63
9.16.2.3 <code>pos_input_node</code>	63
9.17 Dokumentacja struktury <code>passive_component</code>	64

9.17.1 Opis szczegółowy	65
9.17.2 Dokumentacja funkcji składowych	65
9.17.2.1 admittance()	65
9.17.2.2 bipole_component()	65
9.18 Dokumentacja klasy power_probe	66
9.18.1 Opis szczegółowy	67
9.18.2 Dokumentacja konstruktora i destruktora	67
9.18.2.1 power_probe()	67
9.18.3 Dokumentacja funkcji składowych	67
9.18.3.1 get_value()	67
9.19 Dokumentacja klasy probe	68
9.19.1 Opis szczegółowy	69
9.19.2 Dokumentacja funkcji składowych	69
9.19.2.1 get_name()	69
9.19.2.2 get_value()	69
9.19.3 Dokumentacja atrybutów składowych	69
9.19.3.1 m_name	69
9.20 Dokumentacja struktury resistor	70
9.20.1 Opis szczegółowy	71
9.20.2 Dokumentacja konstruktora i destruktora	71
9.20.2.1 resistor()	71
9.20.3 Dokumentacja funkcji składowych	71
9.20.3.1 admittance()	71
9.20.4 Dokumentacja atrybutów składowych	71
9.20.4.1 R	72
9.21 Dokumentacja klasy voltage_probe	72
9.21.1 Opis szczegółowy	73
9.21.2 Dokumentacja konstruktora i destruktora	73
9.21.2.1 voltage_probe() [1/2]	73
9.21.2.2 voltage_probe() [2/2]	73
9.21.3 Dokumentacja funkcji składowych	74
9.21.3.1 get_value()	74
9.22 Dokumentacja struktury voltage_source	74
9.22.1 Opis szczegółowy	76
9.22.2 Dokumentacja konstruktora i destruktora	76
9.22.2.1 voltage_source()	76
9.22.3 Dokumentacja atrybutów składowych	76
9.22.3.1 acV	76
9.22.3.2 dcV	76
9.23 Dokumentacja struktury mna::voltage_source	77
9.23.1 Opis szczegółowy	77
9.23.2 Dokumentacja atrybutów składowych	77

9.23.2.1 nodes	77
9.23.2.2 V	77
10 Dokumentacja plików	79
10.1 Dokumentacja pliku mspice.dox	79
10.2 Dokumentacja pliku src/circuit.cpp	79
10.2.1 Opis szczegółowy	79
10.3 Dokumentacja pliku src/circuit.hpp	80
10.3.1 Opis szczegółowy	81
10.3.2 Dokumentacja definicji typów	81
10.3.2.1 circuit	81
10.4 Dokumentacja pliku src/extended.cpp	82
10.4.1 Opis szczegółowy	83
10.4.2 Dokumentacja typów wyliczanych	83
10.4.2.1 complex_probing_method	83
10.4.3 Dokumentacja funkcji	84
10.4.3.1 create_component()	84
10.4.3.2 main_extended()	85
10.4.3.3 probe_complex()	85
10.4.3.4 probing_method_suffix()	86
10.4.3.5 read_spice_file()	87
10.4.3.6 si_string_to_double()	87
10.4.3.7 tokenize_string()	88
10.4.3.8 tolower()	88
10.5 Dokumentacja pliku src/extended.hpp	88
10.5.1 Opis szczegółowy	89
10.5.2 Dokumentacja funkcji	89
10.5.2.1 main_extended()	89
10.6 Dokumentacja pliku src/legacy.cpp	90
10.6.1 Opis szczegółowy	90
10.6.2 Dokumentacja funkcji	90
10.6.2.1 help()	91
10.6.2.2 main_legacy()	91
10.6.2.3 print_solution()	92
10.6.2.4 read_netlist()	92
10.6.2.5 solve_legacy()	93
10.7 Dokumentacja pliku src/legacy.hpp	94
10.7.1 Opis szczegółowy	95
10.7.2 Dokumentacja funkcji	95
10.7.2.1 main_legacy()	95
10.8 Dokumentacja pliku src/matrix.hpp	96
10.8.1 Opis szczegółowy	97

10.8.2 Dokumentacja funkcji	97
10.8.2.1 join_matrices_horizontal()	97
10.8.2.2 join_matrices_vertical()	98
10.8.2.3 operator*()	98
10.8.2.4 operator<<()	99
10.9 Dokumentacja pliku src/mna.cpp	99
10.9.1 Opis szczegółowy	100
10.9.2 Dokumentacja funkcji	100
10.9.2.1 gaussian_elimination()	100
10.10 Dokumentacja pliku src/mna.hpp	101
10.10.1 Opis szczegółowy	102
10.11 Dokumentacja pliku src/my spice.cpp	103
10.11.1 Opis szczegółowy	103
10.11.2 Dokumentacja funkcji	103
10.11.2.1 main()	103
Indeks	105

Rozdział 1

myspice - symulator obwodów liniowych

Myspice jest programem pozwalającym na analizę elektronicznych obwodów liniowych. Wykorzystuje on algorytm **MNA** (ang. Modified Node Analysis), który pozwala zarówno na analizę obwodów prądu stałego jak i zmiennego, a nawet umożliwia symulację układów ze wzmacniaczami operacyjnymi.

Z tego powodu program może zostać skompilowany w jednej z dwóch wersji:

- `legacy` - wersja podstawowa, zgodna z założeniami polecenia
- `extended` - wersja rozszerzona, częściowo zgodna z symulatorami z rodziny SPICE, rozszerzona o dodatkowe funkcje

Ustawienie parametru CMake `EXTENDED` na `ON` przy kompilacji skutkuje kompilacją wersji rozszerzonej.

1.0.1 Wersja podstawowa

Podstawowa wersja programu pozwala na symulację obwodów prądu stałego złożonych wyłącznie z rezystancji, SEM i SPM.

By przeprowadzić analizę obwodu należy wywołać program w następujący sposób:

```
myspice NETLISTA [WYNIK]
```

gdzie `NETLISTA` to nazwa pliku zawierającego opis obwodu zgodny ze specyfikacją zadania 20, a `[WYNIK]` to opcjonalna nazwa pliku wynikowego. W przypadku niedostarczenia nazwy pliku wynikowego, wyniki zostaną wypisane na standardowe wyjście. Niedostarczenie żadnych argumentów powoduje wypisanie krótkiej informacji o sposobie użycia programu.

Wynikiem działania programu jest lista potencjałów węzłowych oraz informacje o spadku napięcia, prądzie i mocy traconej na każdym z elementów układu. Przykładowo, dla prostego dzielnika rezystorowego 1/2, opisanego `netlistą`:

```
E 1 2 5
R 2 3 1000
R 3 1 1000
```

Wynikiem działania programu jest:

```
Potencjały węzłowe:
  V(1) = 0 V
  V(2) = 5 V
  V(3) = 2.5 V
E1 - [1, 2]:
  V(E1) = 5 V
  I(E1) = -0.0025 A
  P(E1) = -0.0125 W
R1 - [2, 3]:
```

```

V(R1) = -2.5 V
I(R1) = -0.0025 A
P(R1) = 0.00625 W
R2 - [3, 1]:
V(R2) = -2.5 V
I(R2) = -0.0025 A
P(R2) = 0.00625 W
Moc całkowita: 0.0125 W.

```

Niestety treść zadania nie przewiduje sposobu określania węzła odniesienia (masy). Zważywszy na to, że wybór takiego węzła jest konieczny do przeprowadzenia analizy, program zawsze przyjmuje węzeł nr 1 jako punkt odniesienia. **Oznacza to, że węzeł nr 1 jest wymagany, by przeprowadzenie poprawnej symulacji było możliwe.**

W celu sprawdzenia poprawności działania programu, przykładowy obwód podany w treści zadania został zasymulowany w programach myspice, ngspice i LTSpice, a wyniki zostały porównane tutaj: [Sprawdzenie poprawności](#)

1.0.2 Wersja rozszerzona

Działanie rozszerzonej wersji programu zostało dokładniej opisane tutaj - [Wersja rozszerzona](#).

1.0.3 Zasada działania programu

Zasada działania programu jest bardzo podobna w przypadku obu wersji - podstawowej i rozszerzonej. Pierwszym krokiem jest wczytanie i interpretacja netlisty. W wersji podstawowej jest to zadanie na tyle proste, że może być realizowane przez jedną funkcję ([read_netlist\(\)](#)). Tekstowa netlista jest przekształcana na wewnętrzną reprezentację obwodu - [circuit](#) - zbiór nazwanych elementów.

Analiza obwodów przeprowadzana jest przez klasę [circuit_solver](#). Po dostarczeniu jej obwodu [circuit](#), przekształca go na bardziej elementarną reprezentację - [mna::mna_problem](#) - listę admitancji międzywęzłowych, SEM, SPM i wzmacniaczy operacyjnych. Admitancje międzywęzłowe obliczane są na podstawie typu elementów w układzie i częstotliwości, dla której przeprowadzana jest analiza ([circuit_solver::solve\(\)](#)).

Na podstawie struktury [mna::mna_problem](#) formułowany jest układ równań liniowych w postaci macierzowej ([matrix](#)) zgodnie z algorytmem [MNA](#). Do rozwiązania układu wykorzystywany jest algorytm eliminacji Gaussa ([mna::gaussian_elimination\(\)](#)). Na podstawie wektora będącego rozwiązaniem układu tworzona jest klasa [mna::mna_solution](#), która pozwala na łatwiejszą interpretację wyników - odczyt wybranych potencjałów węzłowych i prądów pobieranych z SEM (i wyjść wzmacniaczy operacyjnych).

Po wywołaniu [circuit_solver::solve\(\)](#), klasa [circuit_solver](#) pozwala na odczyt napięć między węzłami układu, prądów płynących przez komponenty i mocy traconej na komponentach. Analiza punktu pracy DC kończy się wypisaniem właśnie tych informacji. W przypadku analizy AC (sweep), układ rozwiązywany jest dla różnych pulsacji z wybranego przedziału.

Nic nie stoi na przeszkodzie, by w przyszłości dodać analizę stanów przejściowych (transient) i elementy nieliniowe. Prawdopodobnie dobrym punktem wyjścia jest implementacja pamięci stanu kondensatorów i indukcyjności, tak jak to opisano [tutaj](#).

Rozdział 2

Wersja rozszerzona

Rozszerzona wersja programu pozwala na wykorzystywanie w układach dodatkowych elementów: kondensatorów, cewek i wzmacniaczy operacyjnych. Pozwala zatem na prowadzenie analizy punktu pracy DC oraz analizę AC. Istotną różnicą jest też obsługiwany format pliku wejściowego opisującego układ - jest on częściowo kompatybilny z formatem symulatorów z rodziny SPICE. Dane wejściowe są zawsze wczytywane przez standardowe wejście, a wyjściowe wypisywane na standardowe wyjście. Dodatkowo, rozluźnione zostały wymagania dotyczące numeracji węzłów w układzie. Wymagane jest tylko istnienie węzła zerowego, który stanowi punkt odniesienia (masę).

Pełna interpretacja plików SPICE wymagałaby stworzenia zaawansowanego analizatora składni, co zdecydowanie wykracza poza tematykę zadania. Dlatego też obsługiwane są tylko następujące polecenia:

- `.ac lin/oct/dec N fs fe - analiza AC` dla zadanego przedziału częstotliwości $[f_s, f_e]$
- `.print dc/ac [mierzone wielkości]` - wypisanie mierzonych wartości

Tabela wielkości możliwych do pomiaru/wyświetlenia:

Składnia	Znaczenie
$V(x)$	Pomiar potencjału węzła x . Odpowiednik $V(x, 0)$
$V(x, y)$	Pomiar napięcia między węzłami x i y
$I(c)$	Pomiar prądu płynącego przez komponent c
$P(c)$	Pomiar mocy traconej na komponencie c

Przy analizie AC mamy do czynienia z zespolonymi wartościami napięć i prądów. W tym celu wprowadzone zostały dodatkowe oznaczenia umożliwiające pomiar wybranych parametrów wielkości zespolonej:

Składnia	Znaczenie
$V/P/Imag(x)$	Pomiar modułu wielkości zespolonej
$V/P/Iph(x)$	Pomiar fazy (argumentu) wielkości zespolonej
$V/P/Ire(x)$	Pomiar części rzeczywistej wielkości zespolonej
$V/P/Iim(x)$	Pomiar części urojonej wielkości zespolonej

Tabela obsługiwanych elementów:

Składnia	Znaczenie
$Rx \ A \ B \ VAL$	Rezystor R_x o wartości VAL łączący węzły A i B

Składnia	Znaczenie
Cx A B VAL	Kondensator Cx o wartości VAL łączący węzły A i B
Lx A B VAL	Indukcyjność Lx o wartości VAL łącząca węzły A i B
Vx A B DCV [AC ACV]	SEM o składowej stałej DCV i składowej zmiennej ACV podłączona dodatnim wyprowadzeniem do węzła A i ujemnym do węzła B
Ix A B DCI [AC ACI]	SPM o składowej stałej DCI i składowej zmiennej ACI podłączona dodatnim wyprowadzeniem do węzła A i ujemnym do węzła B
OPAx P N O	Idealny wzmacniacz operacyjny - wejście nieodwracające podłączone do węzła P, wej. odw. do węzła N, a wyjście do węzła O

Pierwsza linia pliku stanowi jest traktowana jako nazwa układu. Jeżeli w pliku nie znajduje się polecenie `.ac` przeprowadzana jest analiza punktu pracy DC (odpowiednik `.op` w SPICE).

Nota

Symulacja wzmacniaczy operacyjnych opiera się na założeniu, że napięcie między węzłami wejściowymi jest równe 0, a wzmacniacz pracuje z ujemnym sprzężeniem zwrotnym.

2.0.1 Przykłady

2.0.1.1 Dzielnik rezystorowy

```
Dzielnik 1/3 2/3
V1 1 0 9
R1 1 2 10k
R2 2 3 10k
R3 3 0 10k
.print dc I(V1) V(1) V(2) V(3)
```

Wynik działania programu:

```
I(V1) = -0.0003
V(1) = 9
V(2) = 6
V(3) = 3
```

2.0.1.2 Dolnoprzepustowy filtr Sallen-Key II-rzędu

Analiza odpowiedzi częstotliwościowej filtru aktywnego Sallen-Key II rzędu o częstotliwości odcięcia $f_c=1\text{kHz}$.

```
Sallen-Key lowpass filter
OPA1 4 3 4
R1 1 2 16k
R2 2 3 16k
C1 2 4 0.01u
C2 3 0 0.01u
V1 1 0 0 ac 1
.ac dec 5 10 100k
.print ac V(4)
```

Wynik działania programu:

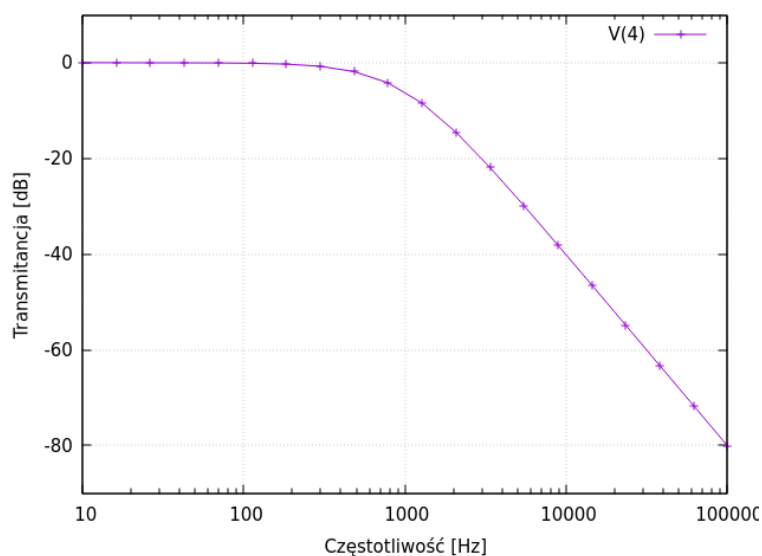
```
step    frequency    V(4)
0        10        0.999899
1       16.2378    0.999734
2       26.3665    0.999298
3       42.8133    0.998151
4       69.5193    0.995139
5      112.884    0.987285
6      183.298    0.967159
7      297.635    0.917827
8      483.293    0.809023
9      784.76     0.616369
10     1274.27    0.378635
11     2069.14    0.187726
12     3359.82    0.0805895
13     5455.59    0.0321746
14     8858.67    0.0124515
```

```

15      14384.5  0.00475925
16      23357.2  0.00181039
17      37926.9  0.000687396
18      61584.8  0.000260819
19      100000   9.89367e-05

```

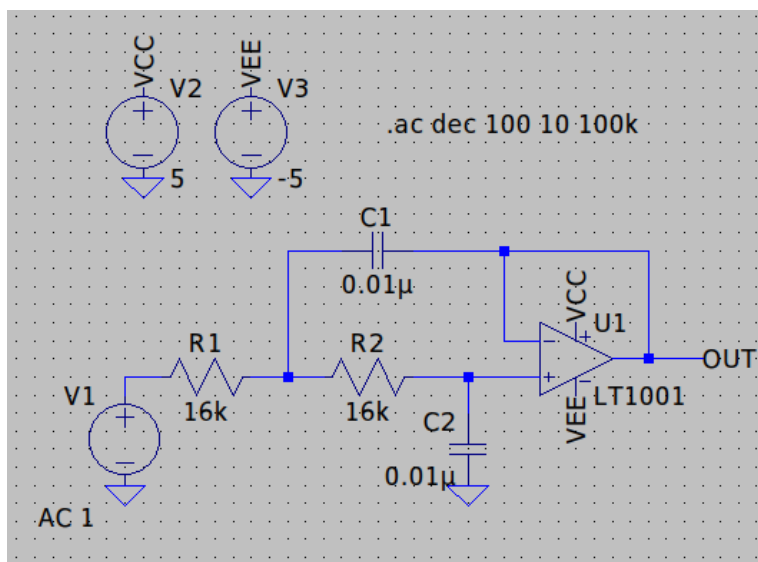
Porównanie wyników symulacji w programie myspace i LTSpice. Na wykresie z programu LTSpice można zaobserwować "odbicie się" odpowiedzi częstotliwościowej układu w okolicach 100kHz. Jest to spowodowane zastosowaniem niedoskonałego wzmacniacza LT1001:



Rysunek 2.1 Wynik symulacji w myspace



Rysunek 2.2 Wynik symulacji w LTSpice

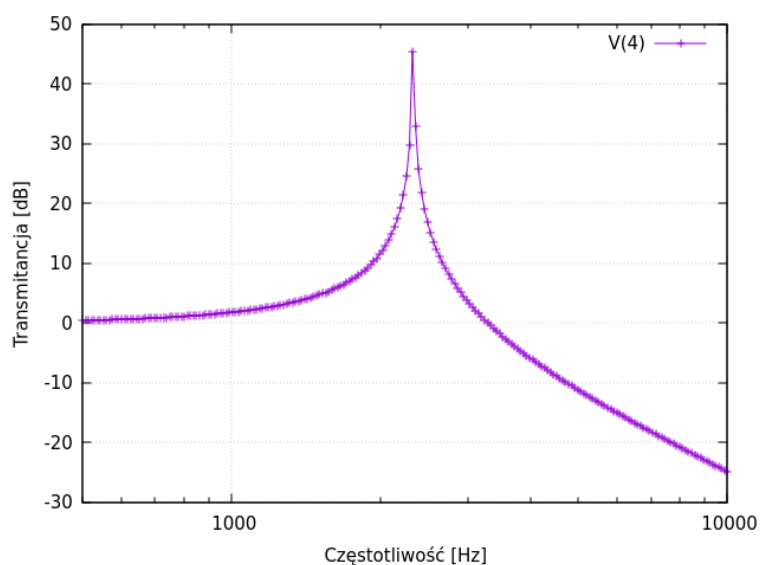


Rysunek 2.3 Odpowiadający układ w LTspice

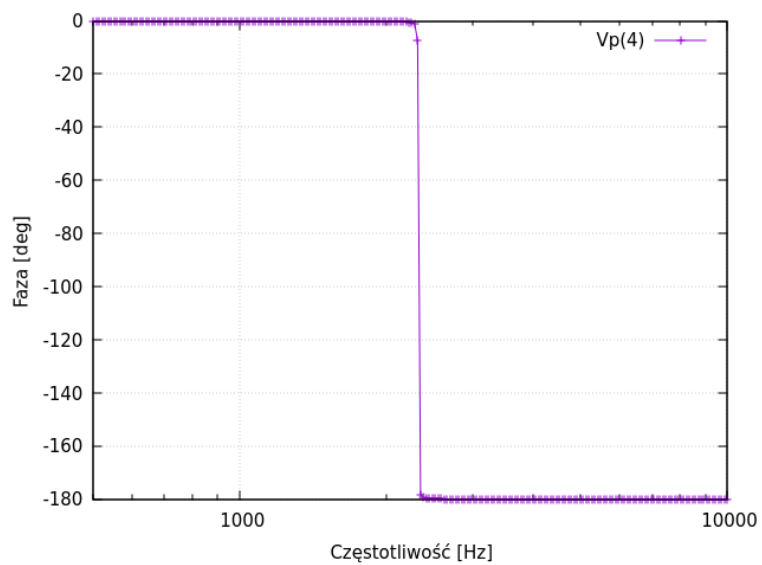
2.0.1.3 Układ rezonansowy RLC

Przykładowa plik wejściowy - analiza odpowiedzi częstotliwościowej i przesunięcia fazy przez układ RLC w zakresie 100Hz-10kHz:

```
RLC circuit
V1 1 0 1 AC 1
R1 1 2 1m
L1 2 4 100u
C1 4 0 47u
.ac oct 50 500 10k
.print ac V(4) Vph(4)
```



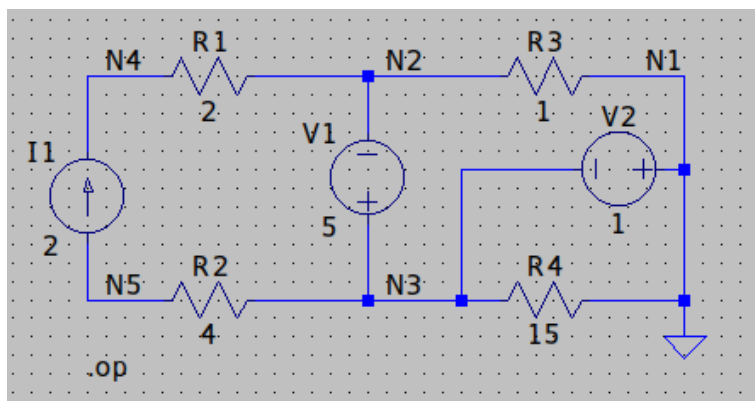
Rysunek 2.4 Transmitancja układu



Rysunek 2.5 Przesunięcie fazy na wyjściu układu

Rozdział 3

Sprawdzenie poprawności



Netlista w formacie określonym przez zadanie:

```
R 4 2 2
I 5 4 2
R 5 3 4
E 3 2 -5
E 3 1 1
R 1 2 1
R 3 1 15
```

Netlista SPICE:

```
test
R1 N2 N4 2
R2 N3 N5 4
R3 0 N2 1
R4 0 N3 15
I1 N5 N4 2
V1 N3 N2 5
V2 0 N3 1
.op
.backanno
.end
```

3.0.1 myspice

```
Potencjaly wezlowe:
V(1) = 0 V
V(2) = -6 V
V(3) = -1 V
V(4) = -2 V
V(5) = -9 V
E1 - [3, 2]:
V(E1) = -5 V
I(E1) = 8 A
P(E1) = -40 W
E2 - [3, 1]:
```

```

V(E2) = 1 V
I(E2) = -6.06667 A
P(E2) = -6.06667 W
I1 - [5, 4]:
V(I1) = 7 V
I(I1) = -2 A
P(I1) = -14 W
R1 - [4, 2]:
V(R1) = -4 V
I(R1) = -2 A
P(R1) = 8 W
R2 - [5, 3]:
V(R2) = 8 V
I(R2) = 2 A
P(R2) = 16 W
R3 - [1, 2]:
V(R3) = -6 V
I(R3) = -6 A
P(R3) = 36 W
R4 - [3, 1]:
V(R4) = 1 V
I(R4) = 0.0666667 A
P(R4) = 0.0666667 W
Moc calkowita: 60.0667 W.

```

3.0.2 LTSpice

```

--- Operating Point ---
V(n2):  -6          voltage
V(n4):  -2          voltage
V(n3):  -1          voltage
V(n5):  -9          voltage
I(I1):   2          device_current
I(R4):  0.0666667   device_current
I(R3):   6          device_current
I(R2):   2          device_current
I(R1):  -2          device_current
I(V2):  -6.06667    device_current
I(V1):  -8          device_current

```

3.0.3 ngspice

```

Circuit: test
Doing analysis at TEMP = 27.000000 and TNOM = 27.000000
No. of Data Rows : 1

```

Node	Voltage
n5	-9.00000e+00
n3	-1.00000e+00
n4	-2.00000e+00
n2	-6.00000e+00

Source	Current
v1#branch	-8.00000e+00
v2#branch	-6.06667e+00

```

Resistor models (Simple linear resistor)
model      R
rsh        0
narrow     0
short      0
tc1        0
tc2        0
tce        0
defw       1e-05
l          1e-05
kf         0
af         0
r          0
bv_max     1e+99
lf         1
wf         1
ef         1
Isource: Independent current source
device     i1
dc         2
m          1
acmag      0
pulse      -
sin        -
exp        -

```

```

    pwl      -
    sffm     -
    am       -
    trnoise  -
    trrandom -
    v        7
    p        -14
    current  2
Resistor: Simple linear resistor
  device    r4          r3          r2
  model     R          R          R
  resistance 15          1          4
  ac         15          1          4
  dtemp     0           0           0
  bv_max     1e+99      1e+99      1e+99
  noisy     1           1           1
  i          0.0666667  6           2
  p          0.0666667  36          16
Resistor: Simple linear resistor
  device    r1
  model     R
  resistance 2
  ac         2
  dtemp     0
  bv_max     1e+99
  noisy     1
  i          -2
  p          8
Vsource: Independent voltage source
  device    v2          v1
  dc         1          5
  acmag      0          0
  pulse      -          -
  sin        -          -
  exp        -          -
  pwl        -          -
  sffm       -          -
  am         -          -
  trnoise    -          -
  trrandom   -          -
  i          -6.06667   -8
  p          6.06667    40
Total analysis time (seconds) = 0.001
Total elapsed time (seconds) = 0.008
Total DRAM available = 32045.613 MB.
DRAM currently available = 22306.926 MB.
Maximum ngspice program size = 15.723 MB.
Current ngspice program size = 9.492 MB.
Shared ngspice pages = 8.105 MB.
Text (code) pages = 5.090 MB.
Stack = 0 bytes.
Library pages = 1.848 MB.

```


Rozdział 4

Indeks przestrzeni nazw

4.1 Lista przestrzeni nazw

Tutaj znajdują się wszystkie przestrzenie nazw wraz z ich krótkimi opisami:

mna	Solver układów metodą MNA	21
---------------------	-------------------------------------	--------------------

Rozdział 5

Indeks hierarchiczny

5.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

ac_analysis_params	23
mna::admittance	25
circuit_component	30
bipole_component	26
current_source	43
passive_component	64
capacitor	27
inductor	45
resistor	70
voltage_source	74
opamp	60
circuit_simulation	31
circuit_solver	32
mna::current_source	42
matrix< T >	48
matrix< std::complex< double > >	48
mna::mna_problem	55
mna::mna_solution	57
mna::opamp	62
probe	68
current_probe	39
power_probe	66
voltage_probe	72
mna::voltage_source	77

Rozdział 6

Indeks klas

6.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

ac_analysis_params	Parametry do analizy AC	23
mna::admittance	Admitancja międzywęzłowa	25
bipole_component	Klasa bazowa dla elementów z dwoma wyprowadzeniami	26
capacitor	Idealny kondensator	27
circuit_component	Klasa bazowa dla wszystkich komponentów, które mogą się znaleźć w obwodzie	30
circuit_simulation	Symulacja układu	31
circuit_solver	Analizator układów liniowych	32
current_probe	Miernik prądu płynącego przez element	39
mna::current_source	Idealna siła prądomotoryczna	42
current_source	Idealna siła prądomotoryczna	43
inductor	Prawie idealna cewka	45
matrix< T >	Klasa ułatwiająca operacje macierzowe	48
mna::mna_problem	Układ do rozwiązania metodą MNA	55
mna::mna_solution	Wynik analizy układu metodą MNA	57
opamp	Idealny wzmacniacz operacyjny	60
mna::opamp	Idealny wzmacniacz operacyjny	62
passive_component	Klasa bazowa dla komponentów posiadających admitancję zależną od częstotliwości	64
power_probe	Miernik mocy wydzielanej na elemencie	66

probe		
	Uogólnienie mierników	68
resistor		
	Idealny rezystor	70
voltage_probe		
	Miernik napięcia międzywęzłowego/na elemencie	72
voltage_source		
	Idealna siła elektromotoryczna	74
mna::voltage_source		
	Idealna siła elektromotoryczna	77

Rozdział 7

Indeks plików

7.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

src/circuit.cpp	Implementacja funkcji związanych z circuit i jego elementami	79
src/circuit.hpp	Definicje typów circuit , circuit_solver i elementów obwodu	80
src/extended.cpp	Rozszerzona część programu	82
src/extended.hpp	Plik nagłówkowy rozszerzonej wersji programu	88
src/legacy.cpp	Główny plik podstawowej wersji programu	90
src/legacy.hpp	Główny plik nagłówkowy podstawowej wersji programu	94
src/matrix.hpp	Definiuje klasę matrix do operacji macierzowych	96
src/mna.cpp	Implementacja algorytmu MNA z eliminacją Gaussa	99
src/mna.hpp	Definicje typów solvera MNA	101
src/my spice.cpp	Główny plik programu	103

Rozdział 8

Dokumentacja przestrzeni nazw

8.1 Dokumentacja przestrzeni nazw mna

Solver układów metodą MNA.

Komponenty

- struct [admittance](#)
Admitancja międzywęzłowa.
- struct [current_source](#)
Idealna siła prądomotoryczna.
- struct [mna_problem](#)
Układ do rozwiązania metodą MNA.
- class [mna_solution](#)
Wynik analizy układu metodą MNA.
- struct [opamp](#)
Idealny wzmacniacz operacyjny.
- struct [voltage_source](#)
Idealna siła elektromotoryczna.

8.1.1 Opis szczegółowy

Solver układów metodą MNA.

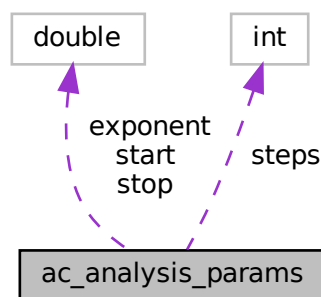
Rozdział 9

Dokumentacja klas

9.1 Dokumentacja struktury `ac_analysis_params`

Parametry do analizy AC.

Diagram współpracy dla `ac_analysis_params`:



Atrybuty publiczne

- `double start`
Dolna częstotliwość [Hz].
- `double stop`
Górna częstotliwość [Hz].
- `double exponent`
Wykładnik sweep'a. 0 to sweep liniowy.
- `int steps`
Liczba punktów na exponent-krotną zmianę częstotliwości/łącznie.

9.1.1 Opis szczegółowy

Parametry do analizy AC.

9.1.2 Dokumentacja atrybutów składowych

9.1.2.1 exponent

```
double ac_analysis_params::exponent
```

Wykładnik sweep'a. 0 to sweep liniowy.

9.1.2.2 start

```
double ac_analysis_params::start
```

Dolna częstotliwość [Hz].

9.1.2.3 steps

```
int ac_analysis_params::steps
```

Liczba punktów na exponent-krotną zmianę częstotliwości/łącznie.

9.1.2.4 stop

```
double ac_analysis_params::stop
```

Górna częstotliwość [Hz].

Dokumentacja dla tej struktury została wygenerowana z pliku:

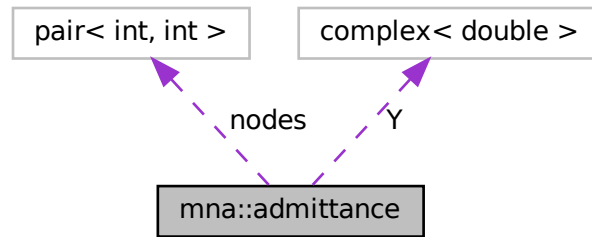
- [src/extended.cpp](#)

9.2 Dokumentacja struktury mna::admittance

Admitancja międzywęzłowa.

```
#include <mna.hpp>
```

Diagram współpracy dla mna::admittance:



Atrybuty publiczne

- `std::pair<int, int>` [nodes](#)
- `std::complex<double>` [Y](#)

9.2.1 Opis szczegółowy

Admitancja międzywęzłowa.

Każdy element pasywny jest do takiej uogólniany.

9.2.2 Dokumentacja atrybutów składowych

9.2.2.1 nodes

```
std::pair<int, int> mna::admittance::nodes
```

9.2.2.2 Y

```
std::complex<double> mna::admittance::Y
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- [src/mna.hpp](#)

9.3 Dokumentacja struktury bipole_component

Klasa bazowa dla elementów z dwoma wyprowadzeniami.

```
#include <circuit.hpp>
```

Diagram dziedziczenia dla bipole_component

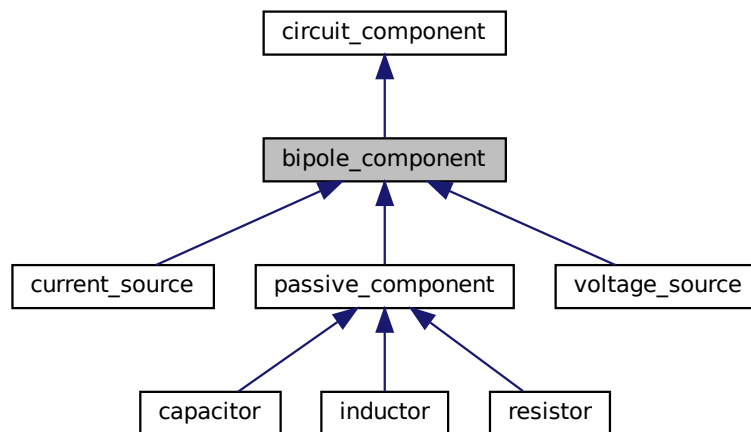
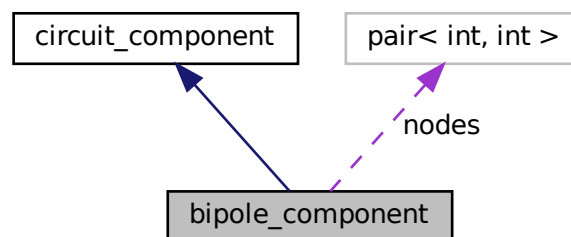


Diagram współpracy dla bipole_component:



Metody publiczne

- `bipole_component` (const std::pair< int, int > &p)

Atrybuty publiczne

- std::pair< int, int > `nodes`
Para węzłów połączonych przez komponent.

9.3.1 Opis szczegółowy

Klasa bazowa dla elementów z dwoma wyprowadzeniami.

9.3.2 Dokumentacja konstruktora i destruktora

9.3.2.1 bipole_component()

```
bipole_component::bipole_component (
    const std::pair< int, int > & p ) [inline]
```

9.3.3 Dokumentacja atrybutów składowych

9.3.3.1 nodes

```
std::pair<int, int> bipole_component::nodes
```

Para węzłów połączonych przez komponent.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- [src/circuit.hpp](#)

9.4 Dokumentacja struktury capacitor

Idealny kondensator.

```
#include <circuit.hpp>
```

Diagram dziedziczenia dla capacitor

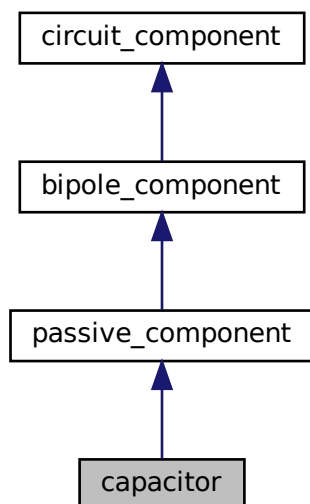
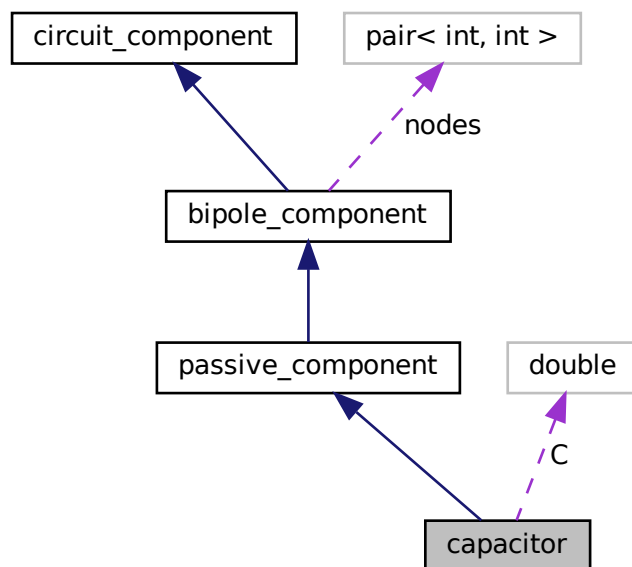


Diagram współpracy dla capacitor:



Metody publiczne

- `capacitor` (`const std::pair< int, int > &p, double c`)

- `std::complex< double > admittance` (double omega) const override
Admitancja kondensatora.

Atrybuty publiczne

- double `C`
Pojemność [F].

9.4.1 Opis szczegółowy

Idealny kondensator.

9.4.2 Dokumentacja konstruktora i destruktora

9.4.2.1 capacitor()

```
capacitor::capacitor (  
    const std::pair< int, int > & p,  
    double c ) [inline]
```

9.4.3 Dokumentacja funkcji składowych

9.4.3.1 admittance()

```
std::complex< double > capacitor::admittance (  
    double omega ) const [override], [virtual]
```

Admitancja kondensatora.

Implementuje `passive_component`.

9.4.4 Dokumentacja atrybutów składowych

9.4.4.1 C

```
double capacitor::C
```

Pojemność [F].

Dokumentacja dla tej struktury została wygenerowana z plików:

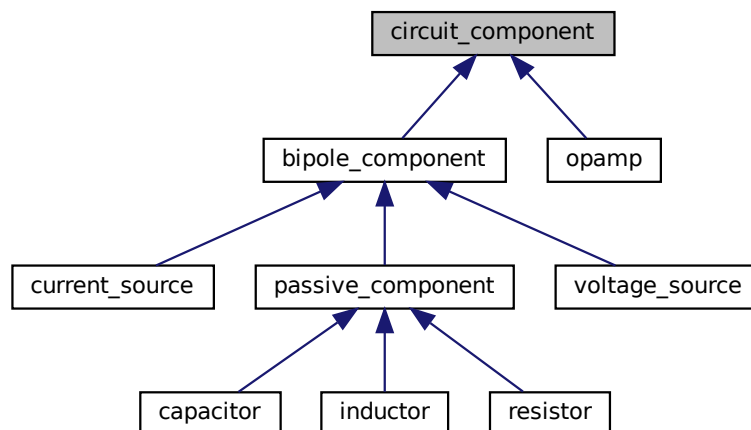
- [src/circuit.hpp](#)
- [src/circuit.cpp](#)

9.5 Dokumentacja struktury circuit_component

Klasa bazowa dla wszystkich komponentów, które mogą się znaleźć w obwodzie.

```
#include <circuit.hpp>
```

Diagram dziedziczenia dla circuit_component



Metody publiczne

- `virtual ~circuit_component()`

9.5.1 Opis szczegółowy

Klasa bazowa dla wszystkich komponentów, które mogą się znaleźć w obwodzie.

9.5.2 Dokumentacja konstruktora i destruktor

9.5.2.1 ~circuit_component()

```
virtual circuit_component::~~circuit_component ( ) [inline], [virtual]
```

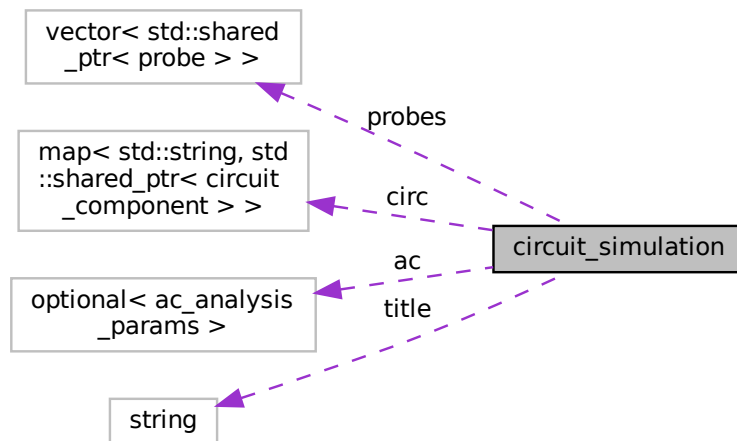
Dokumentacja dla tej struktury została wygenerowana z pliku:

- [src/circuit.hpp](#)

9.6 Dokumentacja struktury circuit_simulation

Symulacja układu.

Diagram współpracy dla circuit_simulation:



Atrybuty publiczne

- `std::string` `title`
- `circuit` `circ`
- `std::optional< ac_analysis_params >` `ac`
- `std::vector< std::shared_ptr< probe > >` `probes`

9.6.1 Opis szczegółowy

Symulacja układu.

9.6.2 Dokumentacja atrybutów składowych

9.6.2.1 ac

```
std::optional<ac_analysis_params> circuit_simulation::ac
```

9.6.2.2 circ

```
circuit circuit_simulation::circ
```

9.6.2.3 probes

```
std::vector<std::shared_ptr<probe> > circuit_simulation::probes
```

9.6.2.4 title

```
std::string circuit_simulation::title
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- [src/extended.cpp](#)

9.7 Dokumentacja klasy circuit_solver

Analizator układów liniowych.

```
#include <circuit.hpp>
```

Metody publiczne

- `circuit_solver` (const `circuit` &circ)
Tworzy solver układów.
- void `update` ()
Aktualizuje mapę numeracji węzłów i rozwiązanie (jeżeli było gotowe)
- void `solve` (double omega)
Poddaje obwód analizie dla zadanej pulsacji.
- const `mna::mna_solution` & `get_solution` () const
Zwraca rozwiązanie jako `mna_solution`.
- const std::map< int, int > & `get_node_map` () const
Zwraca mapowania węzłów (tylko do odczytu)
- double `get_solution_omega` () const
Zwraca pulsację dla której wyznaczone zostało rozwiązanie.
- std::complex< double > `voltage` (int pos, int neg=0) const
Pomiar napięcia między węzłami.
- std::complex< double > `voltage` (const `circuit_component` &comp) const
Pomiar napięcia na komponencie.
- std::complex< double > `current` (const `circuit_component` &comp) const
Pomiar prądu płynącego przez komponent.
- std::complex< double > `power` (const `circuit_component` &comp) const
Pomiar mocy traconej na komponencie.
- std::complex< double > `voltage` (const std::string &ref) const
Pomiar spadku napięcia na komponencie.
- std::complex< double > `current` (const std::string &ref) const
Pomiar prądu płynącego przez komponent.
- std::complex< double > `power` (const std::string &ref) const
Pomiar mocy traconej na komponencie.

9.7.1 Opis szczegółowy

Analizator układów liniowych.

Analizator jest tworzony na podstawie układu (`circuit`). Pozwala na przeprowadzenie analizy metodą MNA wykorzystując `mna::mna_problem`. Głównym zadaniem tej klasy jest wprowadzenie dodatkowej abstrakcji - pozwala ona na tworzenie obwodów poprzez proste dodawanie różnych elementów do powiązanej klasy `circuit` i rozluźnia wymagania dot. numeracji węzłów.

Po wywołaniu `solve()`, możliwy jest pomiar napięć, prądów i mocy w układzie za pomocą `voltage()`, `current()` i `power()`.

9.7.2 Dokumentacja konstruktora i destruktor

9.7.2.1 `circuit_solver()`

```
circuit_solver::circuit_solver (
    const circuit & circ ) [explicit]
```

Tworzy solver układów.

9.7.3 Dokumentacja funkcji składowych

9.7.3.1 current() [1/2]

```
std::complex< double > circuit_solver::current (
    const circuit_component & comp ) const
```

Pomiar prądu płynącego przez komponent.

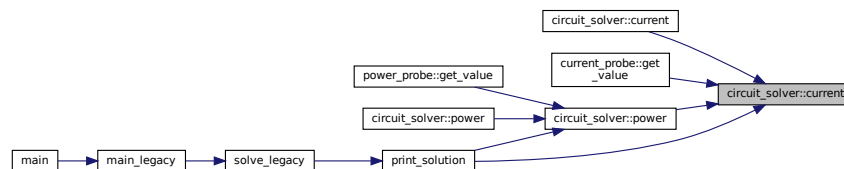
Nota

Pomiar prądu jest możliwy tylko na elementach z dwoma wyprowadzeniami i na wzmacniaczach operacyjnych (prąd wyjścia)

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

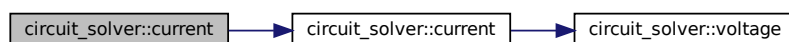


9.7.3.2 current() [2/2]

```
std::complex< double > circuit_solver::current (
    const std::string & ref ) const
```

Pomiar prądu płynącego przez komponent.

Oto graf wywołań dla tej funkcji:

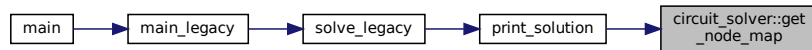


9.7.3.3 `get_node_map()`

```
const std::map< int, int > & circuit_solver::get_node_map ( ) const
```

Zwraca mapowania węzłów (tylko do odczytu)

Oto graf wywołań tej funkcji:



9.7.3.4 `get_solution()`

```
const mna::mna_solution & circuit_solver::get_solution ( ) const
```

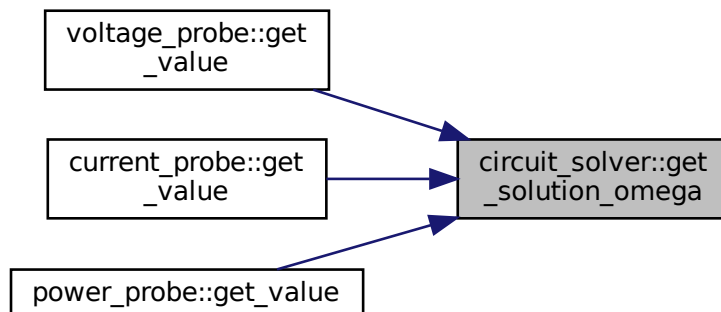
Zwraca rozwiązanie jako `mna_solution`.

9.7.3.5 `get_solution_omega()`

```
double circuit_solver::get_solution_omega ( ) const
```

Zwraca pulsację dla której wyznaczone zostało rozwiązanie.

Oto graf wywołań tej funkcji:

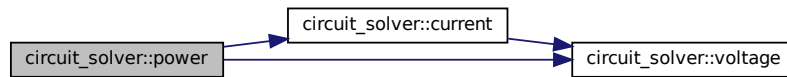


9.7.3.6 power() [1/2]

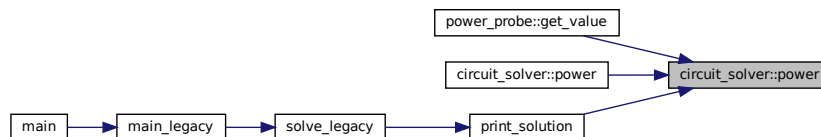
```
std::complex< double > circuit_solver::power (
    const circuit_component & comp ) const
```

Pomiar mocy traconej na komponentcie.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

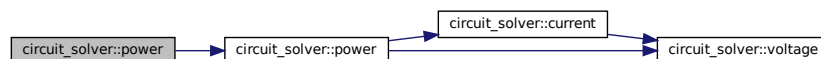


9.7.3.7 power() [2/2]

```
std::complex< double > circuit_solver::power (
    const std::string & ref ) const
```

Pomiar mocy traconej na komponentcie.

Oto graf wywołań dla tej funkcji:



9.7.3.8 solve()

```
void circuit_solver::solve (
    double omega )
```

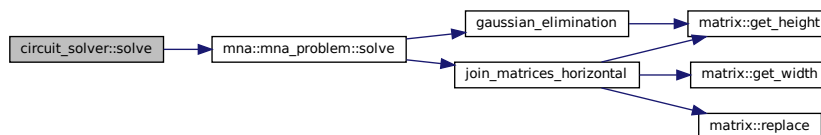
Poddaje obwód analizie dla zadanej pulsacji.

Przy analizie AC wszystkie źródła DC są pomijane i na odwrót. Analiza DC uruchamiana jest przez podanie `omega = 0`.

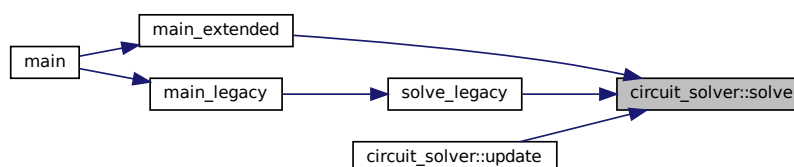
Parametry

<i>omega</i>	pulsacja sygnału źródeł AC. 0 oznacza analizę DC.
--------------	---

Oto graf wywołań dla tej funkcji:



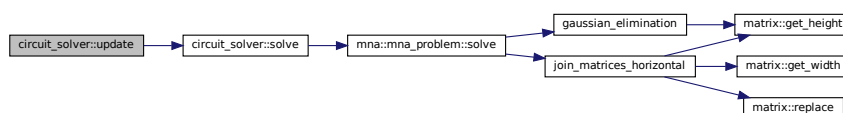
Oto graf wywoływań tej funkcji:

9.7.3.9 `update()`

```
void circuit_solver::update ( )
```

Aktualizuje mapę numeracji węzłów i rozwiązanie (jeżeli było gotowe)

Oto graf wywołań dla tej funkcji:



9.7.3.10 voltage() [1/3]

```
std::complex< double > circuit_solver::voltage (
    const circuit_component & comp ) const
```

Pomiar napięcia na komponencie.

Nota

Pomiar napięcia jest możliwy tylko na elementach z dwoma wyprowadzeniami

Oto graf wywołań dla tej funkcji:



9.7.3.11 voltage() [2/3]

```
std::complex< double > circuit_solver::voltage (
    const std::string & ref ) const
```

Pomiar spadku napięcia na komponencie.

Oto graf wywołań dla tej funkcji:



9.7.3.12 voltage() [3/3]

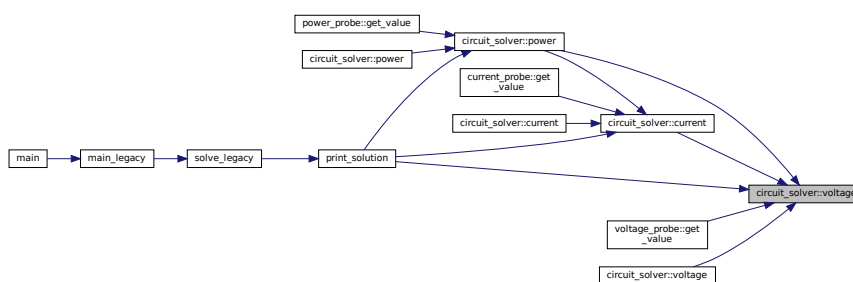
```
std::complex< double > circuit_solver::voltage (
    int pos,
    int neg = 0 ) const
```

Pomiar napięcia między węzłami.

Parametry

<i>pos</i>	Numer mierzonego węzła
<i>neg</i>	Numer węzła odniesienia

Oto graf wywoływań tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z plików:

- [src/circuit.hpp](#)
- [src/circuit.cpp](#)

9.8 Dokumentacja klasy `current_probe`

Miernik prądu płynącego przez element.

Diagram dziedziczenia dla `current_probe`

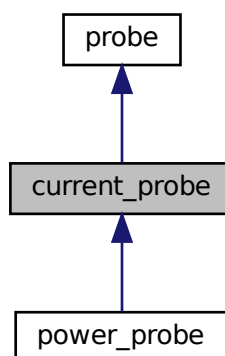
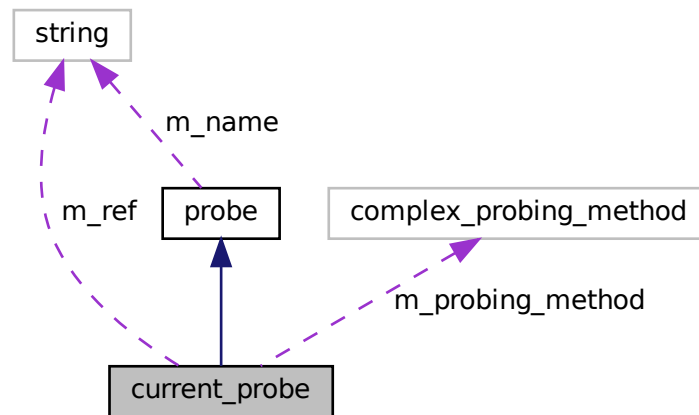


Diagram współpracy dla `current_probe`:



Metody publiczne

- `current_probe` (const `circuit` &circ, const std::string &ref, `complex_probing_method` pm)
- double `get_value` (const `circuit_solver` &solver) const override

Atrybuty chronione

- std::string `m_ref`
- `complex_probing_method` `m_probing_method`

9.8.1 Opis szczegółowy

Miernik prądu płynącego przez element.

9.8.2 Dokumentacja konstruktora i destruktor

9.8.2.1 `current_probe()`

```
current_probe::current_probe (
    const circuit & circ,
    const std::string & ref,
    complex\_probing\_method pm ) [inline]
```

Oto graf wywołań dla tej funkcji:



9.8.3 Dokumentacja funkcji składowych

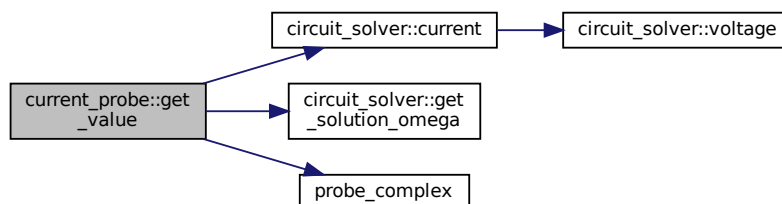
9.8.3.1 `get_value()`

```
double current_probe::get_value (
    const circuit\_solver & solver ) const [inline], [override], [virtual]
```

Implementuje [probe](#).

Reimplementowana w [power_probe](#).

Oto graf wywołań dla tej funkcji:



9.8.4 Dokumentacja atrybutów składowych

9.8.4.1 m_probing_method

```
complex_probing_method current_probe::m_probing_method [protected]
```

9.8.4.2 m_ref

```
std::string current_probe::m_ref [protected]
```

Dokumentacja dla tej klasy została wygenerowana z pliku:

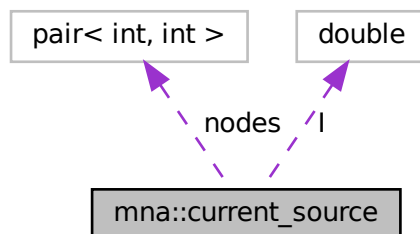
- [src/extended.cpp](#)

9.9 Dokumentacja struktury mna::current_source

Idealna siła prądomotoryczna.

```
#include <mna.hpp>
```

Diagram współpracy dla mna::current_source:



Atrybuty publiczne

- `std::pair< int, int >` [nodes](#)
- `double` [I](#)

9.9.1 Opis szczegółowy

Idealna siła prądomotoryczna.

Nota

Przyjmujemy, że pierwszy węzeł to "plus"

9.9.2 Dokumentacja atrybutów składowych

9.9.2.1 I

```
double mna::current_source::I
```

9.9.2.2 nodes

```
std::pair<int, int> mna::current_source::nodes
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- [src/mna.hpp](#)

9.10 Dokumentacja struktury current_source

Idealna siła prądomotoryczna.

```
#include <circuit.hpp>
```

Diagram dziedziczenia dla current_source

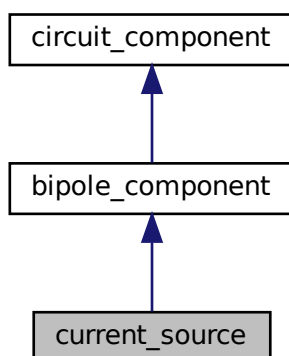
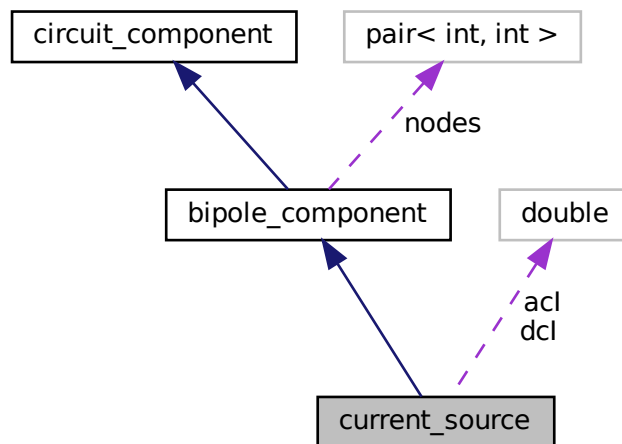


Diagram współpracy dla `current_source`:



Metody publiczne

- `current_source` (`const std::pair< int, int > &p, double i, double ac=0.0`)

Atrybuty publiczne

- `double dcl`
Wartość prądu [A] dla analizy DC.
- `double acI`
Wartość prądu [A] dla analizy AC.

9.10.1 Opis szczegółowy

Idealna siła prądomotoryczna.

9.10.2 Dokumentacja konstruktora i destruktora

9.10.2.1 `current_source()`

```

current_source::current_source (
    const std::pair< int, int > & p,
    double i,
    double ac = 0.0 ) [inline]
  
```

9.10.3 Dokumentacja atrybutów składowych

9.10.3.1 acI

```
double current_source::acI
```

Wartość prądu [A] dla analizy AC.

9.10.3.2 dcI

```
double current_source::dcI
```

Wartość prądu [A] dla analizy DC.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- [src/circuit.hpp](#)

9.11 Dokumentacja struktury inductor

Prawie idealna cewka.

```
#include <circuit.hpp>
```

Diagram dziedziczenia dla inductor

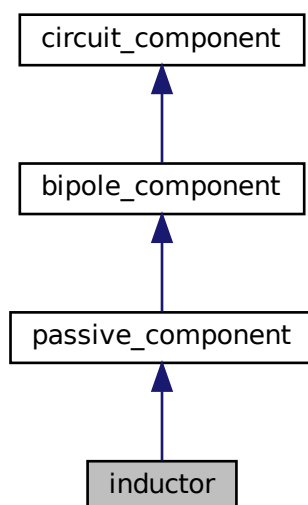
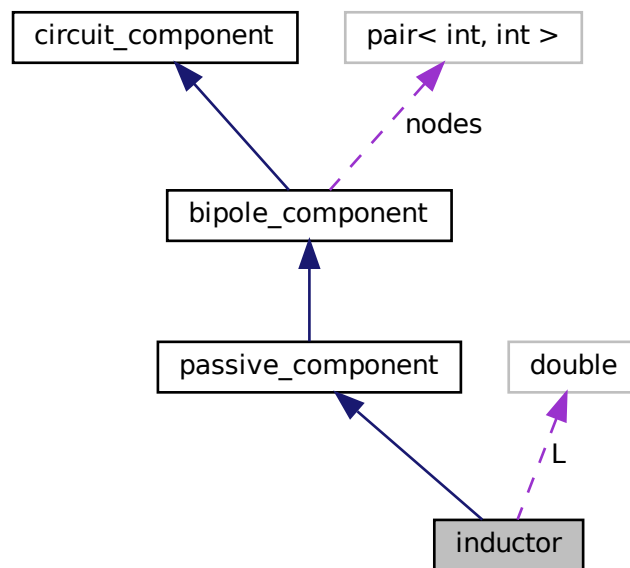


Diagram współpracy dla inductor:



Metody publiczne

- `inductor` (`const std::pair< int, int > &p, double l`)
- `std::complex< double > admittance` (`double omega`) `const override`
Admitancja cewki.

Atrybuty publiczne

- `double L`
Indukcyjność [H].

9.11.1 Opis szczegółowy

Prawie idealna cewka.

Nota

Dla $\omega = 0$ przyjmowana jest minimalna rezystancja, by nie dopuścić do dzielenia przez 0

9.11.2 Dokumentacja konstruktora i destruktora

9.11.2.1 inductor()

```
inductor::inductor (
    const std::pair< int, int > & p,
    double l ) [inline]
```

9.11.3 Dokumentacja funkcji składowych

9.11.3.1 admittance()

```
std::complex< double > inductor::admittance (
    double omega ) const [override], [virtual]
```

Admitancja cewki.

Nota

Przy analizie DC cewka jest zastępowana minimalną rezystancją (wielką admitancją), by nie dopuścić do dzielenia przez 0.

Implementuje [passive_component](#).

9.11.4 Dokumentacja atrybutów składowych

9.11.4.1 L

```
double inductor::L
```

Indukcyjność [H].

Dokumentacja dla tej struktury została wygenerowana z plików:

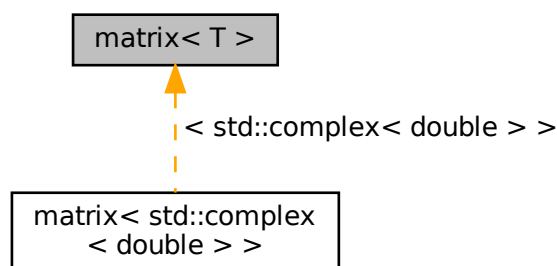
- [src/circuit.hpp](#)
- [src/circuit.cpp](#)

9.12 Dokumentacja szablonu klasy `matrix< T >`

Klasa ułatwiająca operacje macierzowe.

```
#include <matrix.hpp>
```

Diagram dziedziczenia dla `matrix< T >`



Metody publiczne

- `matrix()`
- `matrix(int h, int w)`
Inicjalizuje macierz o określonych rozmiarach.
- `~matrix()`=default
- `matrix(const matrix< T > &)=default`
- `matrix(matrix< T > &&) noexcept=default`
- `matrix< T > & operator= (const matrix< T > &)=default`
- `matrix< T > & operator= (matrix< T > &&) noexcept=default`
- `int get_width() const`
Zwraca szerokość macierzy.
- `int get_height() const`
Zwraca wysokość macierzy.
- `T * data()`
Dostęp do danych w podlegającym macierzy `std::vector`.
- `T & operator()(int row, int col)`
Dostęp do danych w macierzy.
- `const T & operator()(int row, int col) const`
Dostęp do danych w macierzy.
- `const T & at(int row, int col) const`
Dostęp do danych w macierzy.
- `T & at(int row, int col)`
Dostęp do danych w macierzy.
- `void replace(int row, int col, const matrix< T > &mat)`
- `matrix< T > transpose() const`
Zwraca transpozycję macierzy.
- `matrix< T > & operator* (const T &scalar)`
Możenie macierzy przez skalar.
- `matrix< T > & operator+ (const T &scalar)`
Dodawanie skalara do macierzy.

9.12.1 Opis szczegółowy

```
template<typename T>
class matrix< T >
```

Klasa ułatwiająca operacje macierzowe.

9.12.2 Dokumentacja konstruktora i destruktor

9.12.2.1 `matrix()` [1/4]

```
template<typename T >
matrix< T >::matrix ( ) [inline]
```

9.12.2.2 `matrix()` [2/4]

```
template<typename T >
matrix< T >::matrix (
    int h,
    int w ) [inline]
```

Inicjalizuje macierz o określonych rozmiarach.

Parametry

<i>h</i>	wysokość
<i>w</i>	szerokość

9.12.2.3 `~matrix()`

```
template<typename T >
matrix< T >::~~matrix ( ) [default]
```

9.12.2.4 `matrix()` [3/4]

```
template<typename T >
matrix< T >::matrix (
    const matrix< T > & ) [default]
```

9.12.2.5 `matrix()` [4/4]

```
template<typename T >
matrix< T >::matrix (
    matrix< T > && ) [default], [noexcept]
```

9.12.3 Dokumentacja funkcji składowych

9.12.3.1 `at()` [1/2]

```
template<typename T >
T& matrix< T >::at (
    int row,
    int col ) [inline]
```

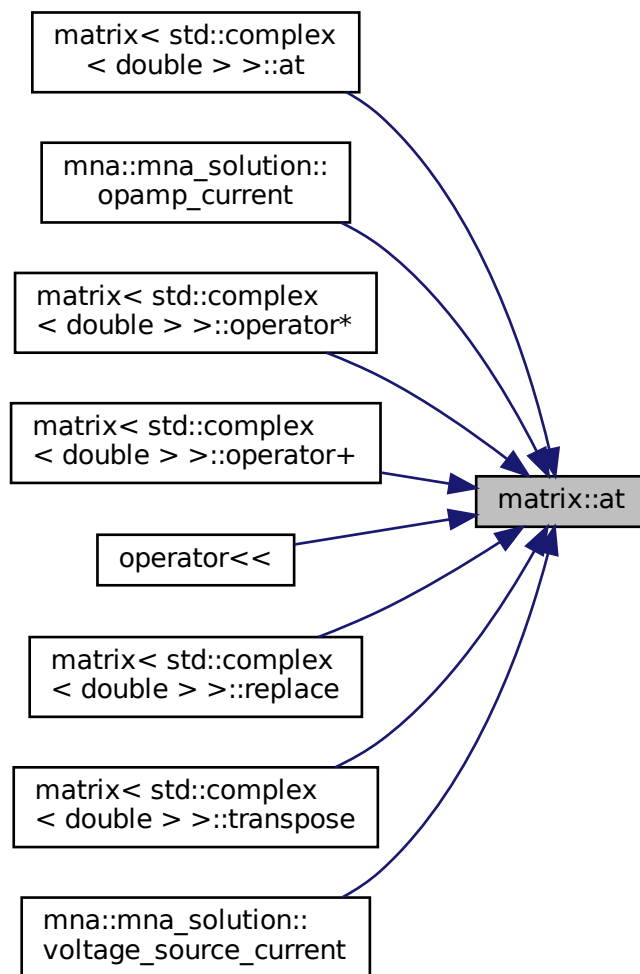
Dostęp do danych w macierzy.

9.12.3.2 `at()` [2/2]

```
template<typename T >
const T& matrix< T >::at (
    int row,
    int col ) const [inline]
```

Dostęp do danych w macierzy.

Oto graf wywoływań tej funkcji:



9.12.3.3 `data()`

```
template<typename T >
T* matrix< T >::data ( ) [inline]
```

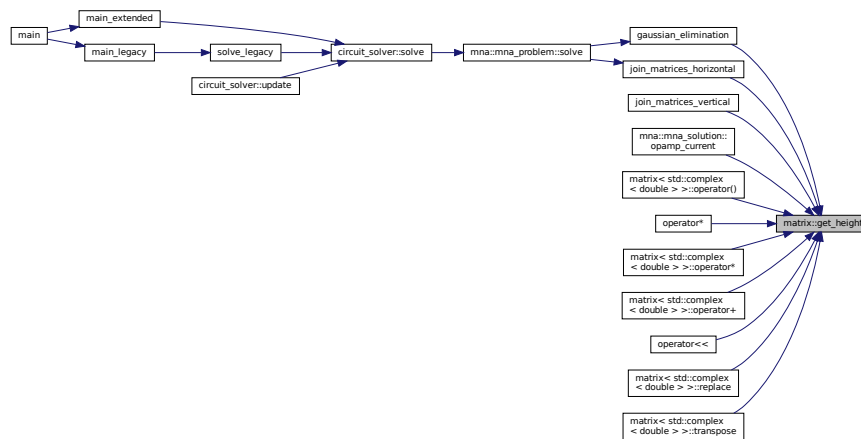
Dostęp do danych w podlegającym macierzy `std::vector`.

9.12.3.4 get_height()

```
template<typename T >
int matrix< T >::get_height ( ) const [inline]
```

Zwraca wysokość macierzy.

Oto graf wywołań tej funkcji:

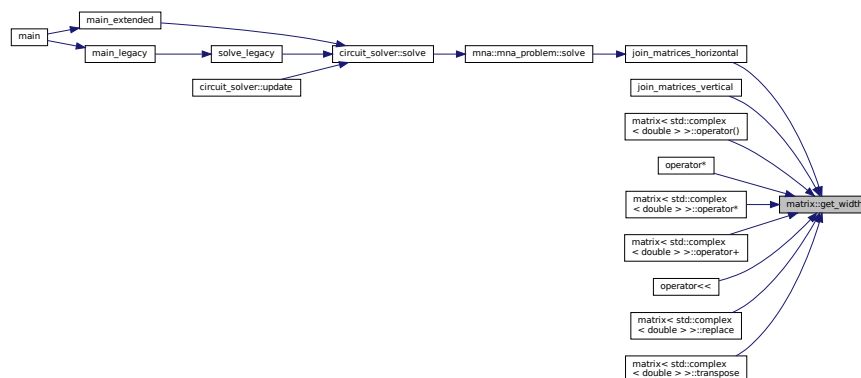


9.12.3.5 get_width()

```
template<typename T >
int matrix< T >::get_width ( ) const [inline]
```

Zwraca szerokość macierzy.

Oto graf wywołań tej funkcji:

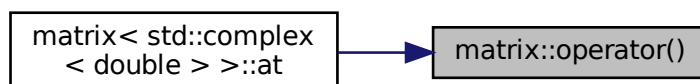


9.12.3.6 `operator()()` [1/2]

```
template<typename T >
T& matrix< T >::operator() (
    int row,
    int col ) [inline]
```

Dostęp do danych w macierzy.

Oto graf wywoływań tej funkcji:



9.12.3.7 `operator()()` [2/2]

```
template<typename T >
const T& matrix< T >::operator() (
    int row,
    int col ) const [inline]
```

Dostęp do danych w macierzy.

9.12.3.8 `operator*()`

```
template<typename T >
matrix<T>& matrix< T >::operator* (
    const T & scalar ) [inline]
```

Możenie macierzy przez skalar.

9.12.3.9 `operator+()`

```
template<typename T >
matrix<T>& matrix< T >::operator+ (
    const T & scalar ) [inline]
```

Dodawanie skalara do macierzy.

9.12.3.10 operator=() [1/2]

```
template<typename T >
matrix<T>& matrix< T >::operator= (
    const matrix< T > & ) [default]
```

9.12.3.11 operator=() [2/2]

```
template<typename T >
matrix<T>& matrix< T >::operator= (
    matrix< T > && ) [default], [noexcept]
```

9.12.3.12 replace()

```
template<typename T >
void matrix< T >::replace (
    int row,
    int col,
    const matrix< T > & mat ) [inline]
```

Nadpisuje fragment macierzy inną mniejszą macierzą

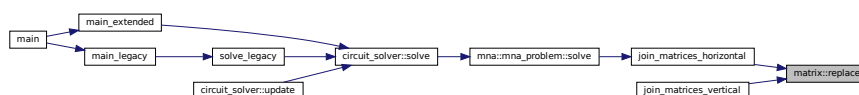
Parametry

<i>row</i>	początkowy wiersz
<i>col</i>	początkowa kolumna
<i>mat</i>	macierz do wpisania

Wyjątki

<code>std::out_of_range</code>	jeśli operacja wymagałaby wykroczenia poza macierz
--------------------------------	--

Oto graf wywołań tej funkcji:



9.12.3.13 transpose()

```
template<typename T >
matrix<T> matrix< T >::transpose ( ) const [inline]
```

Zwraca transpozycję macierzy.

Dokumentacja dla tej klasy została wygenerowana z pliku:

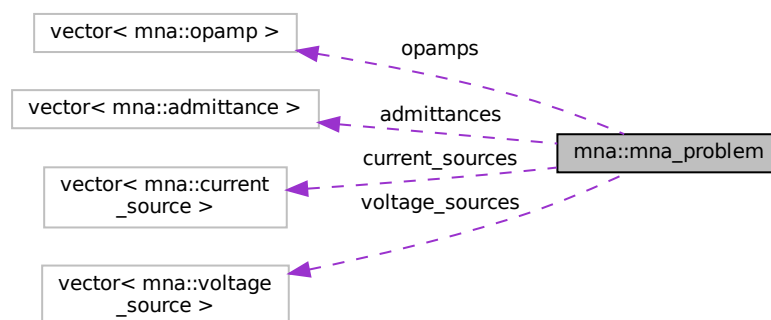
- [src/matrix.hpp](#)

9.13 Dokumentacja struktury mna::mna_problem

Układ do rozwiązania metodą MNA.

```
#include <mna.hpp>
```

Diagram współpracy dla mna::mna_problem:



Metody publiczne

- `mna_solution solve () const`
Wyznacza i zwraca rozwiązanie (potencjały węzłowe) układu.

Atrybuty publiczne

- `std::vector< admittance > admittances`
- `std::vector< voltage_source > voltage_sources`
- `std::vector< current_source > current_sources`
- `std::vector< opamp > opamps`

9.13.1 Opis szczegółowy

Układ do rozwiązania metodą MNA.

Układ jest zdegenerowany do listy admitancji międzywęzłowych, sił napięciowych i prądowych.

Nota

Węzły poniżej 0 to napięcie odniesienia (masa).

Numeracja węzłów wg. macierzy - użycie wysokich liczb w tej strukturze skutkuje obliczeniami na dużej macierzy.

9.13.2 Dokumentacja funkcji składowych

9.13.2.1 solve()

```
mna_solution mna_problem::solve ( ) const
```

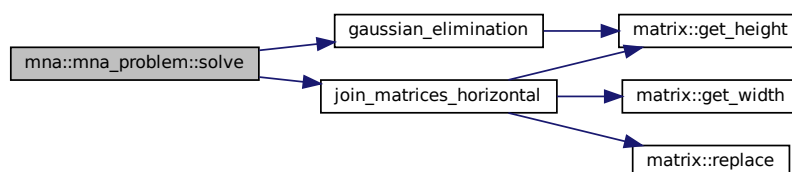
Wyznacza i zwraca rozwiązanie (potencjały węzłowe) układu.

Implementuje macierzowy algorytm opisany [tutaj](#).

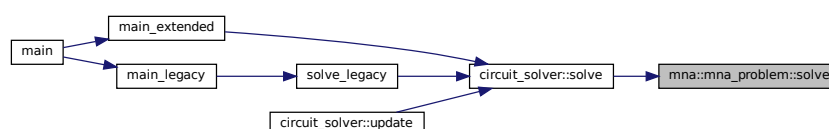
Zobacz również

<https://www.swarthmore.edu/NatSci/echeevel/Ref/mna/MNA3.html>

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



9.13.3 Dokumentacja atrybutów składowych

9.13.3.1 admittances

```
std::vector<admittance> mna::mna_problem::admittances
```

9.13.3.2 current_sources

```
std::vector<current_source> mna::mna_problem::current_sources
```

9.13.3.3 opamps

```
std::vector<opamp> mna::mna_problem::opamps
```

9.13.3.4 voltage_sources

```
std::vector<voltage_source> mna::mna_problem::voltage_sources
```

Dokumentacja dla tej struktury została wygenerowana z plików:

- [src/mna.hpp](#)
- [src/mna.cpp](#)

9.14 Dokumentacja klasy `mna::mna_solution`

Wynik analizy układu metodą MNA.

```
#include <mna.hpp>
```

Metody publiczne

- [mna_solution](#) (const [matrix](#)< std::complex< double >> &solution, int node_count, int vs_count)
Tworzy klasę zawierającą rozwiązanie na podstawie wektora napięć i prądów płynących przez siły elektromotoryczne.
- std::complex< double > [voltage](#) (int pos, int neg=-1) const
Zwraca napięcie między dwoma węzłami.
- std::complex< double > [voltage_source_current](#) (int id) const
Zwraca prąd pobierany ze źródła napięciowego.
- std::complex< double > [opamp_current](#) (int id) const
Zwraca prąd pobierany z wyjścia wzmacniacza operacyjnego.
- const [matrix](#)< std::complex< double >> [get_matrix](#) () const
Zwraca macierz zawierająca rozwiązanie.

9.14.1 Opis szczegółowy

Wynik analizy układu metodą MNA.

Wynik analizy układu. Dostarcza informacje o potencjałach węzłowych i prądach płynących przez siły elektromotoryczne.

9.14.2 Dokumentacja konstruktora i destruktora

9.14.2.1 mna_solution()

```
mna_solution::mna_solution (
    const matrix< std::complex< double >> & solution,
    int node_count,
    int vs_count )
```

Tworzy klasę zawierającą rozwiązanie na podstawie wektora napięć i prądów płynących przez siły elektromotoryczne.

Parametry

<i>solution</i>	Macierz zawierająca rozwiązanie
<i>node_count</i>	Liczba węzłów w układzie
<i>vs_count</i>	Liczba SEM w układzie (nie licząc wzmacniaczy operacyjnych)

9.14.3 Dokumentacja funkcji składowych

9.14.3.1 get_matrix()

```
const matrix< std::complex< double >> mna_solution::get_matrix ( ) const
```

Zwraca macierz zawierająca rozwiązanie.

9.14.3.2 opamp_current()

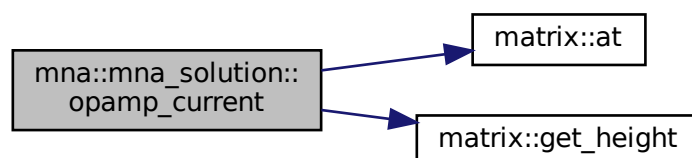
```
std::complex< double > mna_solution::opamp_current (
    int id ) const
```

Zwraca prąd pobierany z wyjścia wzmacniacza operacyjnego.

Parametry

<i>id</i>	numer wzmacniacza
-----------	-------------------

Oto graf wywołań dla tej funkcji:

9.14.3.3 `voltage()`

```
std::complex< double > mna_solution::voltage (
    int pos,
    int neg = -1 ) const
```

Zwraca napięcie między dwoma węzłami.

Nota

Węzły poniżej 0 traktowane są jako masa.

Parametry

<i>pos</i>	numer mierzonego
<i>neg</i>	numer węzła odniesienia (domyślnie -1)

9.14.3.4 `voltage_source_current()`

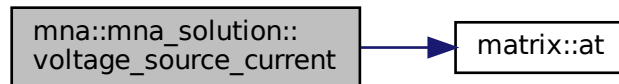
```
std::complex< double > mna_solution::voltage_source_current (
    int id ) const
```

Zwraca prąd pobierany ze źródła napięciowego.

Parametry

<i>id</i>	numer źródła napięciowego (numeracja od 0)
-----------	--

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z plików:

- [src/mna.hpp](#)
- [src/mna.cpp](#)

9.15 Dokumentacja struktury opamp

Idealny wzmacniacz operacyjny.

```
#include <circuit.hpp>
```

Diagram dziedziczenia dla opamp

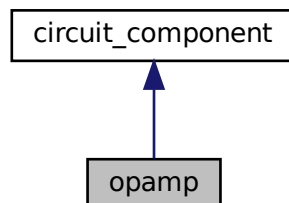
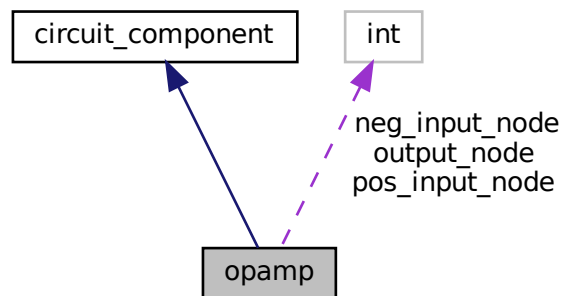


Diagram współpracy dla opamp:



Metody publiczne

- `opamp` (int pos, int neg, int out)

Atrybuty publiczne

- int `pos_input_node`
- int `neg_input_node`
Numer węzła wejścia nieodwracającego.
- int `output_node`
Numer węzła wejścia odwracającego.

9.15.1 Opis szczegółowy

Idealny wzmacniacz operacyjny.

9.15.2 Dokumentacja konstruktora i destruktora

9.15.2.1 `opamp()`

```

opamp::opamp (
    int pos,
    int neg,
    int out ) [inline]
  
```

9.15.3 Dokumentacja atrybutów składowych

9.15.3.1 neg_input_node

```
int opamp::neg_input_node
```

Numer węzła wejścia nieodwracającego.

9.15.3.2 output_node

```
int opamp::output_node
```

Numer węzła wejścia odwracającego.

9.15.3.3 pos_input_node

```
int opamp::pos_input_node
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

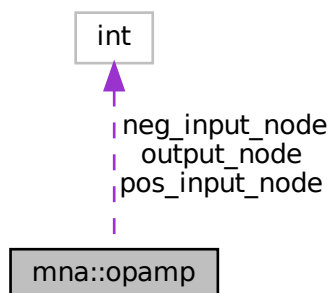
- [src/circuit.hpp](#)

9.16 Dokumentacja struktury mna::opamp

Idealny wzmacniacz operacyjny.

```
#include <mna.hpp>
```

Diagram współpracy dla mna::opamp:



Atrybuty publiczne

- int [pos_input_node](#)
- int [neg_input_node](#)
- int [output_node](#)

9.16.1 Opis szczegółowy

Idealny wzmacniacz operacyjny.

Ostrzeżenie

Zakładamy, że wzmacniacz pracuje z ujemnym sprzężeniem zwrotnym. Analiza opiera się na założeniu, że napięcie na wejściach jest równe. Dodatkowym skutkiem ubocznym jest fakt, że podłączenie wzmacniacza "na odwrót" niczego nie zmienia.

9.16.2 Dokumentacja atrybutów składowych

9.16.2.1 neg_input_node

```
int mna::opamp::neg_input_node
```

9.16.2.2 output_node

```
int mna::opamp::output_node
```

9.16.2.3 pos_input_node

```
int mna::opamp::pos_input_node
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- [src/mna.hpp](#)

9.17 Dokumentacja struktury passive_component

Klasa bazowa dla komponentów posiadających admitancję zależną od częstotliwości.

```
#include <circuit.hpp>
```

Diagram dziedziczenia dla passive_component

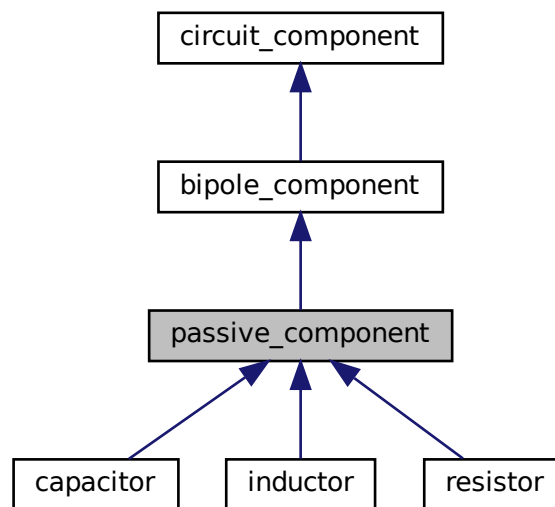
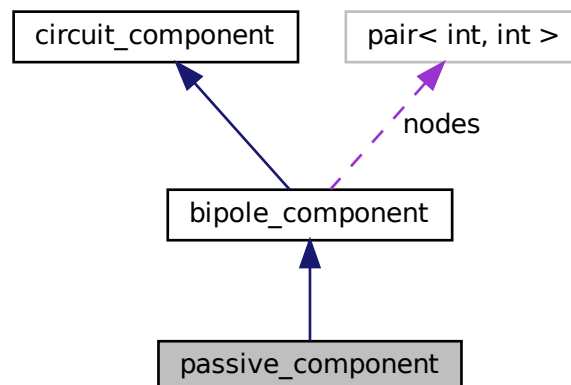


Diagram współpracy dla passive_component:



Metody publiczne

- `virtual std::complex< double > admittance (double omega) const =0`
- `bipole_component (const std::pair< int, int > &p)`

Dodatkowe Dziedziczone Składowe

9.17.1 Opis szczegółowy

Klasa bazowa dla komponentów posiadających admitancję zależną od częstotliwości.

9.17.2 Dokumentacja funkcji składowych

9.17.2.1 `admittance()`

```
virtual std::complex<double> passive_component::admittance (  
    double omega ) const [pure virtual]
```

Implementowany w `capacitor`, `inductor` i `resistor`.

9.17.2.2 `bipole_component()`

```
bipole_component::bipole_component [inline]
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- `src/circuit.hpp`

9.18 Dokumentacja klasy power_probe

Miernik mocy wydzielanej na elemencie.

Diagram dziedziczenia dla power_probe

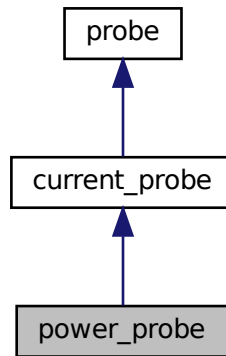
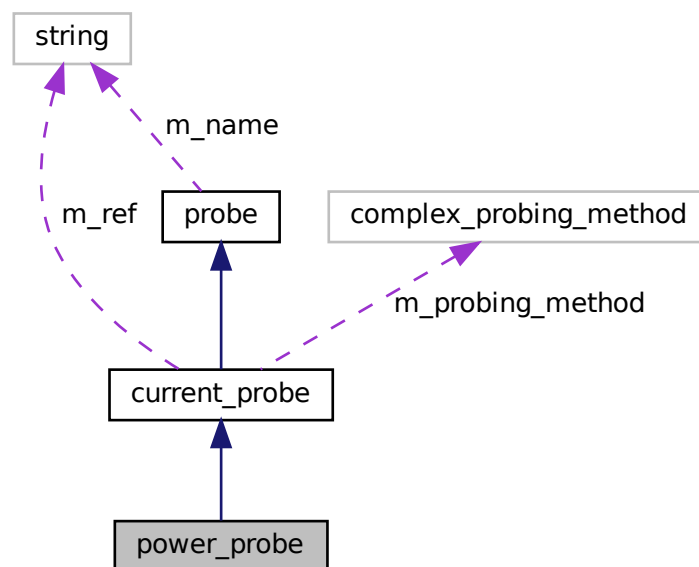


Diagram współpracy dla power_probe:



Metody publiczne

- [power_probe](#) (const [circuit](#) &circ, const std::string &ref, [complex_probing_method](#) pm)
- double [get_value](#) (const [circuit_solver](#) &solver) const override

Dodatkowe Dziedziczone Składowe

9.18.1 Opis szczegółowy

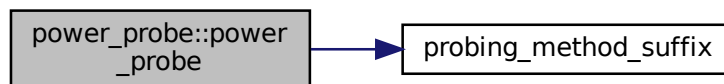
Miernik mocy wydzielanej na elemencie.

9.18.2 Dokumentacja konstruktora i destruktora

9.18.2.1 power_probe()

```
power_probe::power_probe (
    const circuit & circ,
    const std::string & ref,
    complex\_probing\_method pm ) [inline]
```

Oto graf wywołań dla tej funkcji:



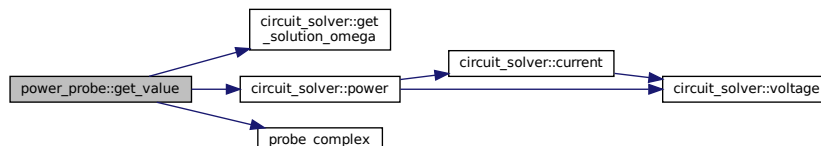
9.18.3 Dokumentacja funkcji składowych

9.18.3.1 get_value()

```
double power_probe::get_value (
    const circuit\_solver & solver ) const [inline], [override], [virtual]
```

Reimplementowana z [current_probe](#).

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [src/extended.cpp](#)

9.19 Dokumentacja klasy probe

Uogólnienie mierników.

Diagram dziedziczenia dla probe

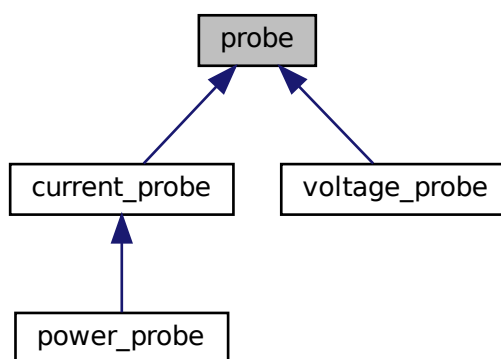
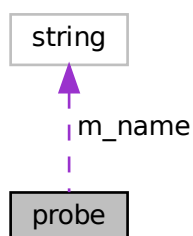


Diagram współpracy dla probe:



Metody publiczne

- `std::string get_name () const`
- `virtual double get_value (const circuit_solver &solver) const =0`

Atrybuty chronione

- `std::string m_name`

9.19.1 Opis szczegółowy

Uogólnienie mierników.

9.19.2 Dokumentacja funkcji składowych

9.19.2.1 get_name()

```
std::string probe::get_name ( ) const [inline]
```

9.19.2.2 get_value()

```
virtual double probe::get_value (
    const circuit\_solver & solver ) const [pure virtual]
```

Implementowany w [power_probe](#), [current_probe](#) i [voltage_probe](#).

9.19.3 Dokumentacja atrybutów składowych

9.19.3.1 m_name

```
std::string probe::m_name [protected]
```

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [src/extended.cpp](#)

9.20 Dokumentacja struktury resistor

Idealny rezystor.

```
#include <circuit.hpp>
```

Diagram dziedziczenia dla resistor

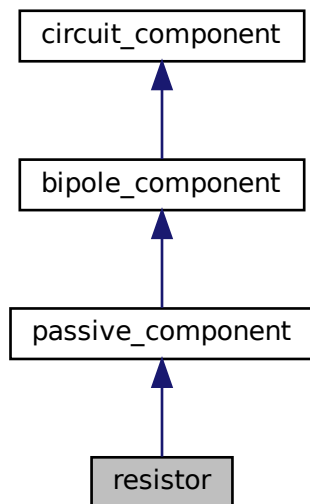
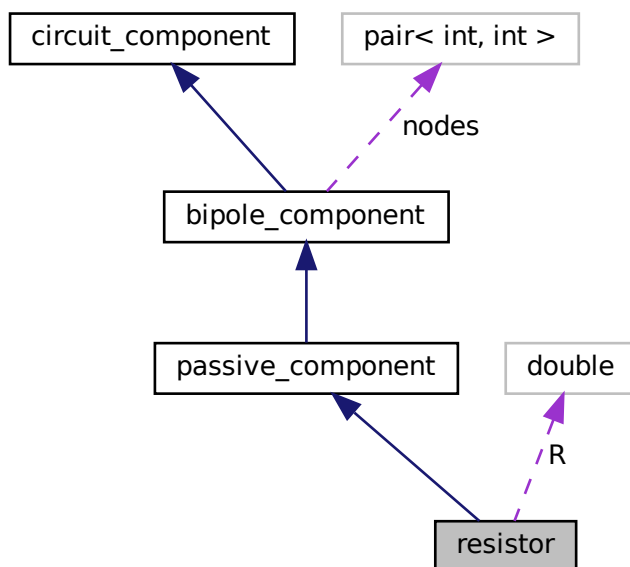


Diagram współpracy dla resistor:



Metody publiczne

- `resistor` (`const std::pair< int, int > &p, double r`)
- `std::complex< double > admittance` (`double omega`) `const override`
Admitancja rezystancji.

Atrybuty publiczne

- `double R`
Rezystancja [Ohm].

9.20.1 Opis szczegółowy

Idealny rezystor.

9.20.2 Dokumentacja konstruktora i destruktor

9.20.2.1 resistor()

```
resistor::resistor (  
    const std::pair< int, int > & p,  
    double r ) [inline]
```

9.20.3 Dokumentacja funkcji składowych

9.20.3.1 admittance()

```
std::complex< double > resistor::admittance (  
    double omega ) const [override], [virtual]
```

Admitancja rezystancji.

Implementuje `passive_component`.

9.20.4 Dokumentacja atrybutów składowych

9.20.4.1 R

```
double resistor::R
```

Rezystancja [Ohm].

Dokumentacja dla tej struktury została wygenerowana z plików:

- [src/circuit.hpp](#)
- [src/circuit.cpp](#)

9.21 Dokumentacja klasy voltage_probe

Miernik napięcia międzywęzłowego/na elemencie.

Diagram dziedziczenia dla voltage_probe

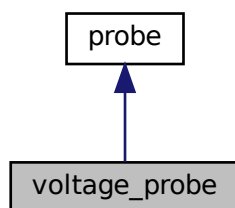
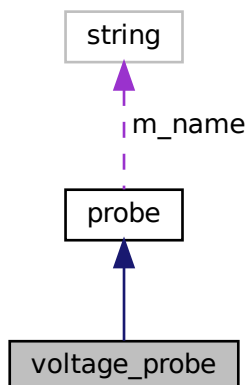


Diagram współpracy dla voltage_probe:



Metody publiczne

- `voltage_probe` (const `circuit` &*circ*, const std::string &*ref*, `complex_probing_method` *pm*)
- `voltage_probe` (int *pos*, int *neg*, `complex_probing_method` *pm*)
- double `get_value` (const `circuit_solver` &*solver*) const override

Dodatkowe Dziedziczone Składowe

9.21.1 Opis szczegółowy

Miernik napięcia międzywęzłowego/na elemencie.

9.21.2 Dokumentacja konstruktora i destruktor

9.21.2.1 `voltage_probe()` [1/2]

```
voltage_probe::voltage_probe (  
    const circuit & circ,  
    const std::string & ref,  
    complex_probing_method pm ) [inline]
```

Oto graf wywołań dla tej funkcji:



9.21.2.2 `voltage_probe()` [2/2]

```
voltage_probe::voltage_probe (  
    int pos,  
    int neg,  
    complex_probing_method pm ) [inline]
```

Oto graf wywołań dla tej funkcji:



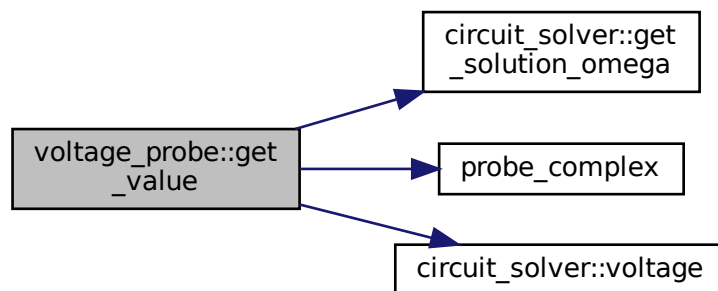
9.21.3 Dokumentacja funkcji składowych

9.21.3.1 get_value()

```
double voltage_probe::get_value (
    const circuit\_solver & solver ) const [inline], [override], [virtual]
```

Implementuje [probe](#).

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- [src/extended.cpp](#)

9.22 Dokumentacja struktury voltage_source

Idealna siła elektromotoryczna.

```
#include <circuit.hpp>
```

Diagram dziedziczenia dla voltage_source

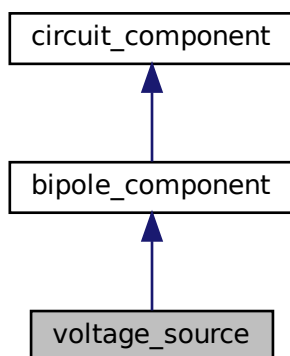
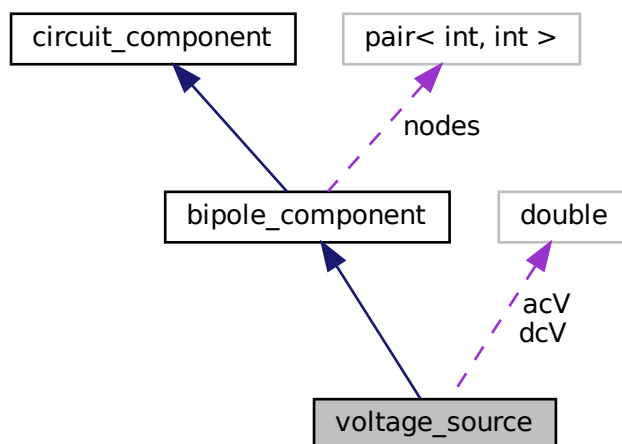


Diagram współpracy dla voltage_source:



Metody publiczne

- `voltage_source` (const std::pair< int, int > &p, double v, double ac=0.0)

Atrybuty publiczne

- double `dcV`
Wartość napięcia [V] dla analizy DC.
- double `acV`
Wartość napięcie [V] dla analizy AC.

9.22.1 Opis szczegółowy

Idealna siła elektromotoryczna.

9.22.2 Dokumentacja konstruktora i destruktora

9.22.2.1 voltage_source()

```
voltage_source::voltage_source (
    const std::pair< int, int > & p,
    double v,
    double ac = 0.0 ) [inline]
```

9.22.3 Dokumentacja atrybutów składowych

9.22.3.1 acV

```
double voltage_source::acV
```

Wartość napięcie [V] dla analizy AC.

9.22.3.2 dcV

```
double voltage_source::dcV
```

Wartość napięcia [V] dla analizy DC.

Dokumentacja dla tej struktury została wygenerowana z pliku:

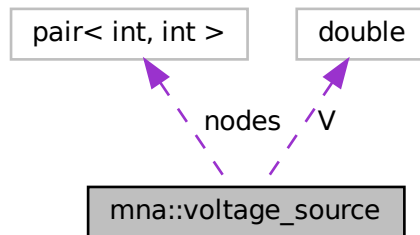
- [src/circuit.hpp](#)

9.23 Dokumentacja struktury mna::voltage_source

Idealna siła elektromotoryczna.

```
#include <mna.hpp>
```

Diagram współpracy dla mna::voltage_source:



Atrybuty publiczne

- `std::pair< int, int >` [nodes](#)
- `double` [V](#)

9.23.1 Opis szczegółowy

Idealna siła elektromotoryczna.

Nota

Przyjmujemy, że pierwszy węzeł to "plus"

9.23.2 Dokumentacja atrybutów składowych

9.23.2.1 nodes

```
std::pair<int, int> mna::voltage_source::nodes
```

9.23.2.2 V

```
double mna::voltage_source::V
```

Dokumentacja dla tej struktury została wygenerowana z pliku:

- [src/mna.hpp](#)

Rozdział 10

Dokumentacja plików

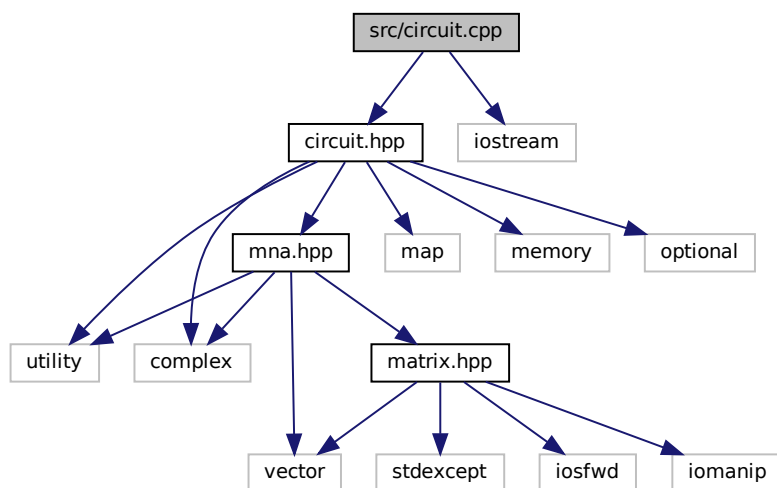
10.1 Dokumentacja pliku myspice.dox

10.2 Dokumentacja pliku src/circuit.cpp

Implementacja funkcji związanych z [circuit](#) i jego elementami.

```
#include "circuit.hpp"  
#include <iostream>
```

Wykres zależności załączania dla circuit.cpp:



10.2.1 Opis szczegółowy

Implementacja funkcji związanych z [circuit](#) i jego elementami.

Autor

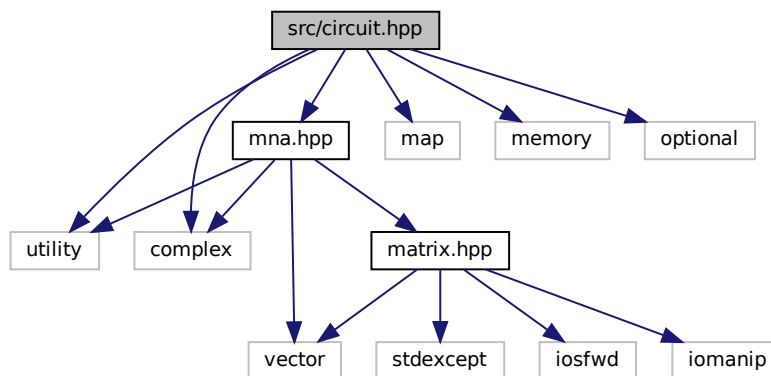
Jacek Wieczorek

10.3 Dokumentacja pliku src/circuit.hpp

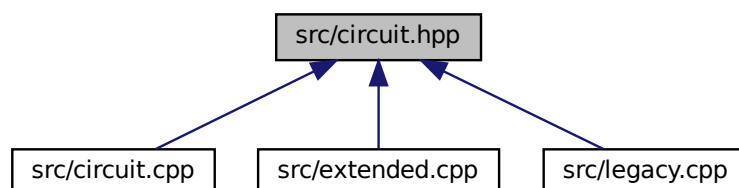
Definicje typów [circuit](#), [circuit_solver](#) i elementów obwodu.

```
#include <utility>
#include <map>
#include <memory>
#include <complex>
#include <optional>
#include "mna.hpp"
```

Wykres zależności załączania dla circuit.hpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- struct [circuit_component](#)
Klasa bazowa dla wszystkich komponentów, które mogą się znaleźć w obwodzie.
- struct [bipole_component](#)
Klasa bazowa dla elementów z dwoma wyprowadzeniami.
- struct [passive_component](#)
Klasa bazowa dla komponentów posiadających admitancję zależną od częstotliwości.

- struct `voltage_source`
Idealna siła elektromotoryczna.
- struct `current_source`
Idealna siła prądomotoryczna.
- struct `resistor`
Idealny rezystor.
- struct `inductor`
Prawie idealna cewka.
- struct `capacitor`
Idealny kondensator.
- struct `opamp`
Idealny wzmacniacz operacyjny.
- class `circuit_solver`
Analizator układów liniowych.

Definicje typów

- using `circuit` = `std::map< std::string, std::shared_ptr< circuit_component > >`
Obwód elektryczny - zbiór nazwanych elementów.

10.3.1 Opis szczegółowy

Definicje typów `circuit`, `circuit_solver` i elementów obwodu.

Autor

Jacek Wieczorek

10.3.2 Dokumentacja definicji typów

10.3.2.1 circuit

```
using circuit = std::map<std::string, std::shared_ptr<circuit_component> >
```

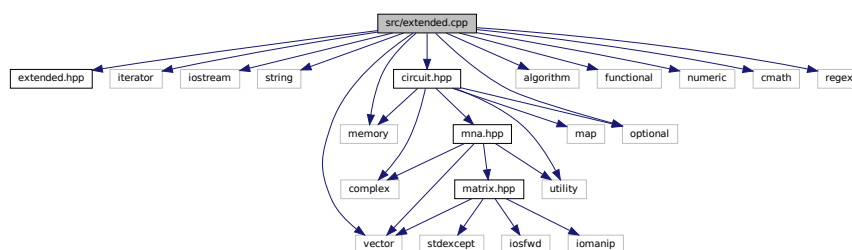
Obwód elektryczny - zbiór nazwanych elementów.

10.4 Dokumentacja pliku src/extended.cpp

Rozszerzona część programu.

```
#include "extended.hpp"
#include <iterator>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <functional>
#include <memory>
#include <numeric>
#include <optional>
#include <cmath>
#include <regex>
#include "circuit.hpp"
```

Wykres zależności załączania dla extended.cpp:



Komponenty

- struct [ac_analysis_params](#)
Parametry do analizy AC.
- class [probe](#)
Uogólnienie mierników.
- class [voltage_probe](#)
Miernik napięcia międzywęzłowego/na elemencie.
- class [current_probe](#)
Miernik prądu płynącego przez element.
- class [power_probe](#)
Miernik mocy wydzielanej na elemencie.
- struct [circuit_simulation](#)
Symulacja układu.

Wyliczenia

- enum [complex_probing_method](#) {
[complex_probing_method::DEFAULT](#), [complex_probing_method::MAGNITUDE](#), [complex_probing_method::PHASE](#),
[complex_probing_method::REAL](#),
[complex_probing_method::IMAGINARY](#) }
Metoda pomiaru zespolonych wielkości fizycznych.

Funkcje

- double `probe_complex` (std::complex< double > c, `complex_probing_method` method, double omega=0)
Zwraca określony komponent zespolonej wielkości fizycznej.
- std::string `probing_method_suffix` (`complex_probing_method` method)
Zwraca suffix dodawany do oznaczenia mierzonej wartości zespolonej.
- static std::string `tolower` (std::string s)
Zamienia litery w stringu na małe.
- static std::vector< std::string > `tokenize_string` (const std::string &s, std::function< bool(char c)> predicate)
Zwraca wektor fragmentów tekstu rozdzielonych znakami białymi.
- static double `si_string_to_double` (const std::string &s)
Zamienia tekst będący liczbą z przedrostkiem SI na wartość
- static std::shared_ptr< `circuit_component` > `create_component` (const std::vector< std::string > &tokens)
Tworzy komponent na podstawie "tokenizowanej" linii pliku SPICE.
- static `circuit_simulation` `read_spice_file` (std::istream &netlist)
Tworzy symulację na podstawie pliku częściowo kompatybilnego z formatem SPICE.
- int `main_extended` (int argc, char *argv[])
Funkcja main dla rozszerzonej wersji programu.

10.4.1 Opis szczegółowy

Rozszerzona część programu.

Autor

Jacek Wieczorek

10.4.2 Dokumentacja typów wyliczanych

10.4.2.1 `complex_probing_method`

```
enum complex_probing_method [strong]
```

Metoda pomiaru zespolonych wielkości fizycznych.

Wartości wyliczeń

DEFAULT	
MAGNITUDE	
PHASE	
REAL	
IMAGINARY	

10.4.3 Dokumentacja funkcji

10.4.3.1 create_component()

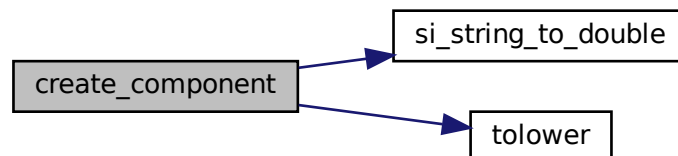
```
static std::shared_ptr<circuit_component> create_component (
    const std::vector< std::string > & tokens ) [static]
```

Tworzy komponent na podstawie "tokenizowanej" linii pliku SPICE.

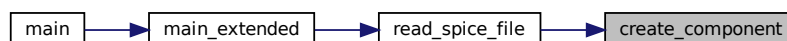
Nota

Akceptuje nazwy węzłów typu '2z'. Nie jest to piękne, ale też na razie nie ma potrzeby żeby to na siłę naprawiać.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

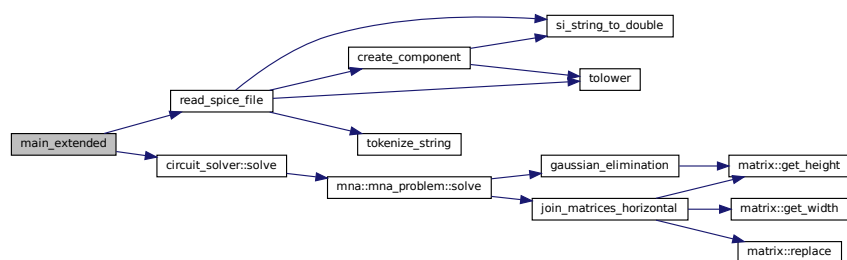


10.4.3.2 main_extended()

```
int main_extended (
    int argc,
    char * argv[] )
```

Funkcja main dla rozszerzonej wersji programu.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

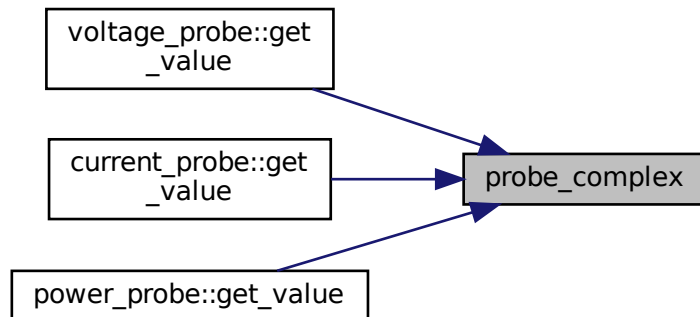


10.4.3.3 probe_complex()

```
double probe_complex (
    std::complex< double > c,
    complex_probing_method method,
    double omega = 0 )
```

Zwraca określony komponent zespolonej wielkości fizycznej.

W trybie pomiaru DEFAULT zwraca część rzeczywistą dla pulsacji równej 0, a w trybie AC zwraca moduł. Oto graf wywołań tej funkcji:

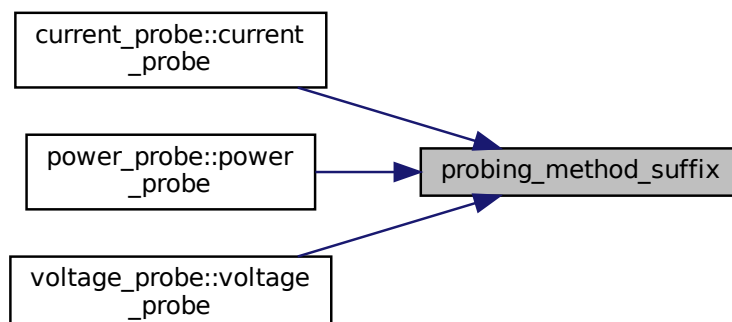


10.4.3.4 `probing_method_suffix()`

```
std::string probing_method_suffix (  
    complex_probing_method method )
```

Zwraca suffix dodawany do oznaczenia mierzonej wartości zespolonej.

Oto graf wywołań tej funkcji:

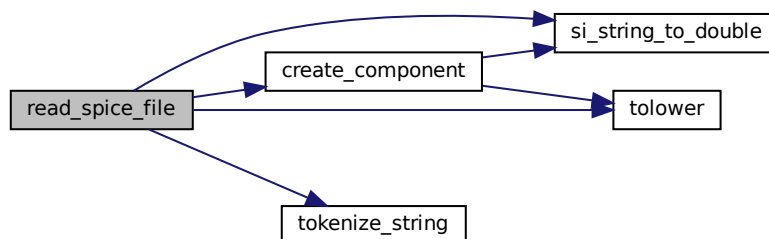


10.4.3.5 read_spice_file()

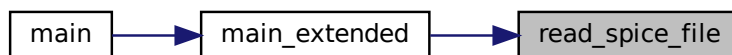
```
static circuit_simulation read_spice_file (  
    std::istream & netlist ) [static]
```

Tworzy symulację na podstawie pliku częściowo kompatybilnego z formatem SPICE.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



10.4.3.6 si_string_to_double()

```
static double si_string_to_double (  
    const std::string & s ) [static]
```

Zamienia tekst będący liczbą z przedrostkiem SI na wartość

Oto graf wywoływań tej funkcji:



10.4.3.7 tokenize_string()

```
static std::vector<std::string> tokenize_string (  
    const std::string & s,  
    std::function< bool(char c)> predicate ) [static]
```

Zwraca wektor fragmentów tekstu rozdzielonych znakami białymi.

Oto graf wywołań tej funkcji:

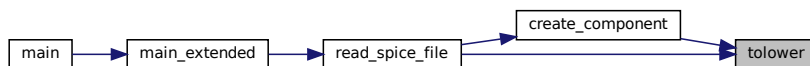


10.4.3.8 tolower()

```
static std::string tolower (  
    std::string s ) [static]
```

Zamienia litery w stringu na małe.

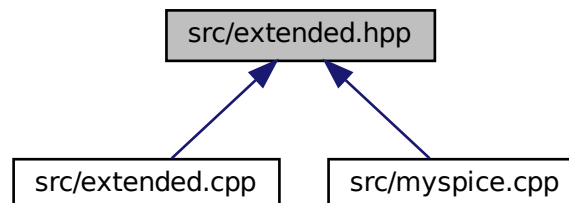
Oto graf wywołań tej funkcji:



10.5 Dokumentacja pliku src/extended.hpp

Plik nagłówkowy rozszerzonej wersji programu.

Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Funkcje

- int `main_extended` (int argc, char *argv[])

Funkcja main dla rozszerzonej wersji programu.

10.5.1 Opis szczegółowy

Plik nagłówkowy rozszerzonej wersji programu.

Autor

Jacek Wieczorek

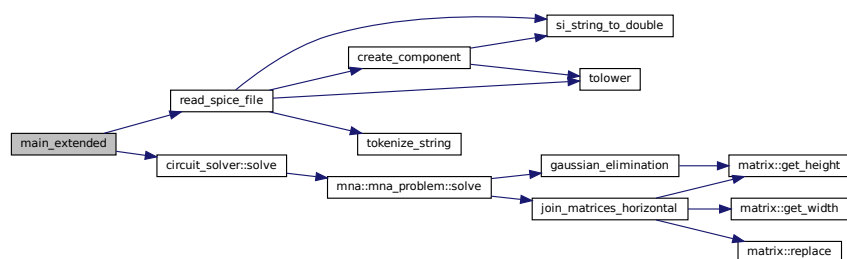
10.5.2 Dokumentacja funkcji

10.5.2.1 main_extended()

```
int main_extended (
    int argc,
    char * argv[] )
```

Funkcja main dla rozszerzonej wersji programu.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

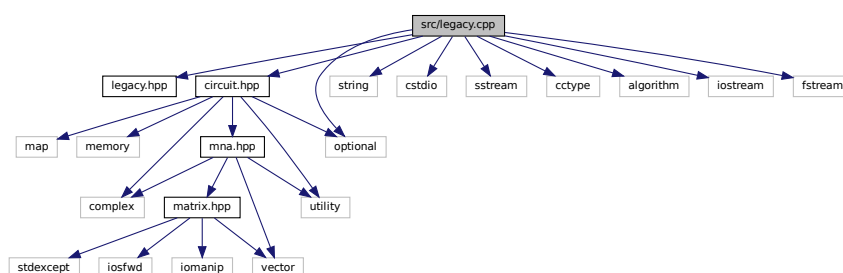


10.6 Dokumentacja pliku src/legacy.cpp

Główny plik podstawowej wersji programu.

```
#include "legacy.hpp"
#include "circuit.hpp"
#include <string>
#include <cstdio>
#include <sstream>
#include <cctype>
#include <algorithm>
#include <iostream>
#include <fstream>
#include <optional>
```

Wykres zależności załączania dla legacy.cpp:



Funkcje

- static [circuit read_netlist](#) (std::istream &netlist)
Przekształca prostą netlistę na obwód.
- static void [print_solution](#) (const [circuit](#) &circ, const [circuit_solver](#) &sol, std::ostream &f)
Wypisuje wynik symulacji układu do podanego strumienia.
- static void [solve_legacy](#) (std::istream &netlist, std::ostream &output)
Rozwiązuje układ zgodny ze specyfikacją zadania.
- static void [help](#) ()
Wypisuje informację pomocy.
- int [main_legacy](#) (int argc, char *argv[])
Główna funkcja uproszczonej wersji programu.

10.6.1 Opis szczegółowy

Główny plik podstawowej wersji programu.

Autor

Jacek Wieczorek

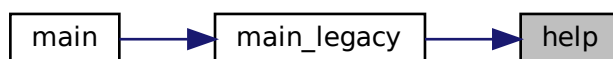
10.6.2 Dokumentacja funkcji

10.6.2.1 help()

```
static void help ( ) [static]
```

Wypisuje informację pomocy.

Oto graf wywołań tej funkcji:

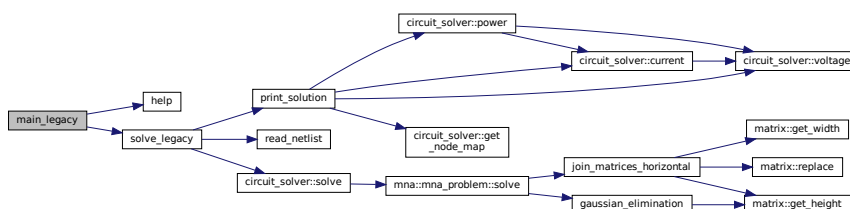


10.6.2.2 main_legacy()

```
int main_legacy (
    int argc,
    char * argv[] )
```

Główna funkcja uproszczonej wersji programu.

Oto graf wywołań dla tej funkcji:



Oto graf wywołań tej funkcji:



10.6.2.3 print_solution()

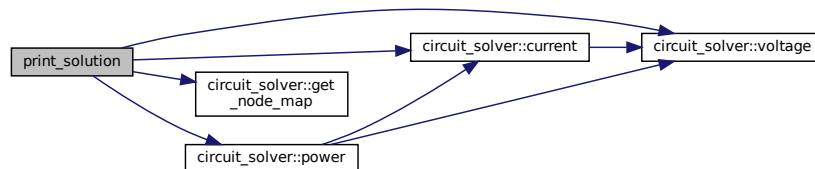
```
static void print_solution (
    const circuit & circ,
    const circuit_solver & sol,
    std::ostream & f ) [static]
```

Wypisuje wynik symulacji układu do podanego strumienia.

Parametry

<i>circ</i>	Obwód
<i>sol</i>	Solver z rozwiązaniem
<i>f</i>	Strumień wyjściowy

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



10.6.2.4 read_netlist()

```
static circuit read_netlist (
    std::istream & netlist ) [static]
```

Przekształca prostą netlistę na obwód.

Parametry

<i>netlist</i>	Strumień dostarczający netlistę
----------------	---------------------------------

Zwraca

Obwód opisany przez netlistę

Wczytywana netlista musi być zgodna z formatem opisanym w treści zadania. Najbardziej bolesne jest to, że netlista, która ma być wczytana przez program, nie podaje, który węzeł jest węzłem odniesienia (masą). Wybór węzła odniesienia jest jednak niezbędny do przeprowadzenia analizy układu. Z tego powodu zakładam, przesuwam numerację węzłów o jeden - w zadaniu zaczyna się od 1, a tutaj od zera.

Ostrzeżenie

Brak węzła o numerze 1 może poskutkować błędem symulacji lub otrzymaniem niepoprawnych wyników.

Zobacz również

[circuit_solver](#)

Para węzłów jest wczytywana w odwrotnej kolejności. To przez to, że mój solver przyjmuje numerację węzłów w SEM i SPM zgodną ze SPICE, a nie z treścią zadania. Oto graf wywołań tej funkcji:



10.6.2.5 solve_legacy()

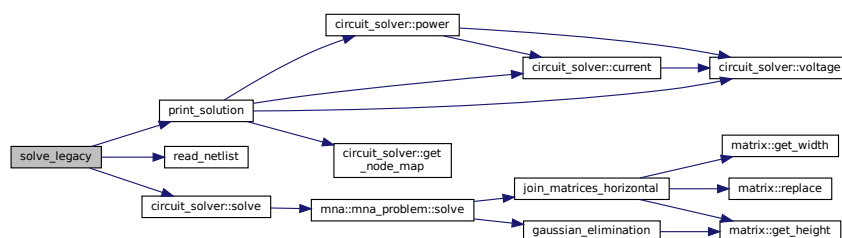
```
static void solve_legacy (
    std::istream & netlist,
    std::ostream & output ) [static]
```

Rozwiązuje układ zgodny ze specyfikacją zadania.

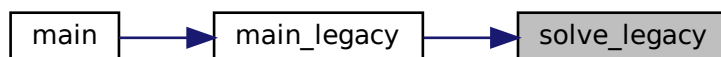
Parametry

<i>netlist</i>	Strumień dostarczający netliste
<i>output</i>	Strumień wyjściowy gdzie ma być wypisany wynik

Oto graf wywołań dla tej funkcji:



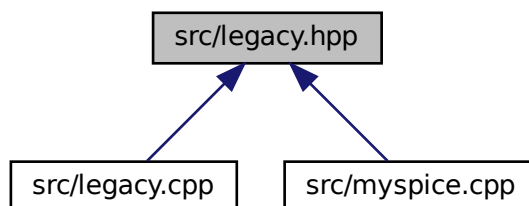
Oto graf wywoływań tej funkcji:



10.7 Dokumentacja pliku src/legacy.hpp

Główny plik nagłówkowy podstawowej wersji programu.

Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Funkcje

- int [main_legacy](#) (int argc, char *argv[])

Główna funkcja uproszczonej wersji programu.

10.7.1 Opis szczegółowy

Główny plik nagłówkowy podstawowej wersji programu.

Autor

Jacek Wieczorek

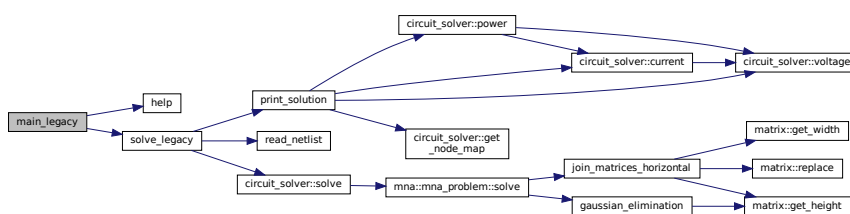
10.7.2 Dokumentacja funkcji

10.7.2.1 main_legacy()

```
int main_legacy (  
    int argc,  
    char * argv[] )
```

Główna funkcja uproszczonej wersji programu.

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

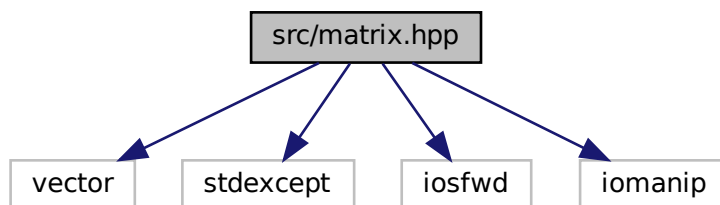


10.8 Dokumentacja pliku src/matrix.hpp

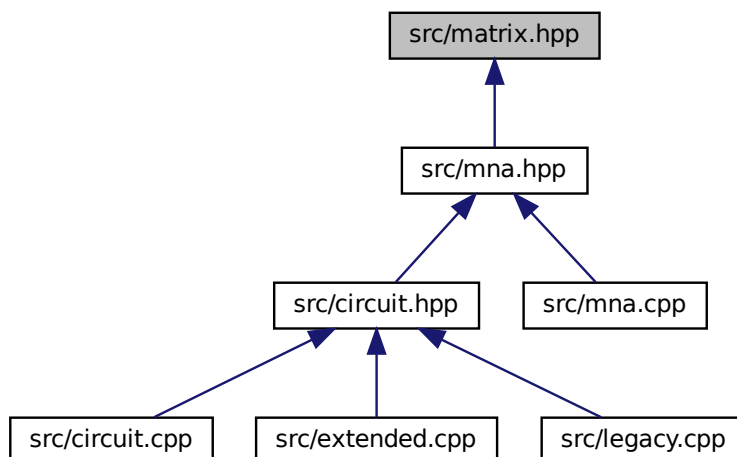
Definiuje klasę `matrix` do operacji macierzowych.

```
#include <vector>
#include <stdexcept>
#include <iosfwd>
#include <iomanip>
```

Wykres zależności załączania dla matrix.hpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class `matrix< T >`

Klasa ułatwiająca operacje macierzowe.

Funkcje

- `template<typename T >`
`matrix< T > join_matrices_horizontal (const matrix< T > &l, const matrix< T > &r)`
Złącza macierze w poziomie.
- `template<typename T >`
`matrix< T > join_matrices_vertical (const matrix< T > &u, const matrix< T > &d)`
Złącza macierze w pionie.
- `template<typename T , typename U , typename V = std::common_type_t<T, U>>`
`matrix< V > operator* (const matrix< T > &lhs, const matrix< U > &rhs)`
Operator mnożenia macierzowego.
- `template<typename T >`
`std::ostream & operator<< (std::ostream &s, const matrix< T > &mat)`
Operator wypisania dla macierzy.

10.8.1 Opis szczegółowy

Definiuje klasę `matrix` do operacji macierzowych.

Autor

Jacek Wieczorek

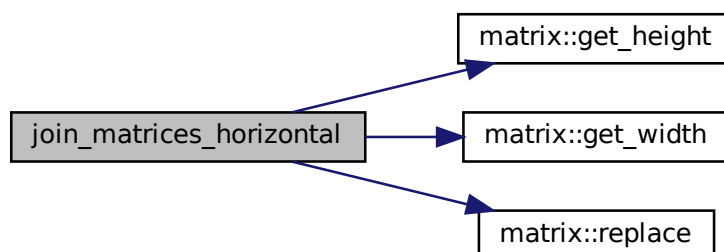
10.8.2 Dokumentacja funkcji

10.8.2.1 `join_matrices_horizontal()`

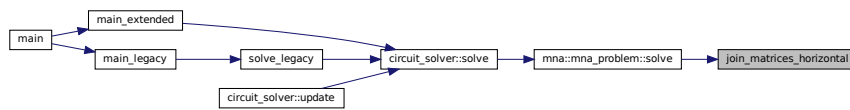
```
template<typename T >
matrix<T> join_matrices_horizontal (
    const matrix< T > & l,
    const matrix< T > & r )
```

Złącza macierze w poziomie.

Oto graf wywołań dla tej funkcji:



Oto graf wywołań tej funkcji:



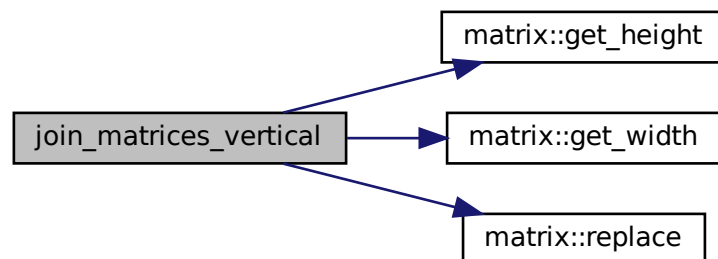
10.8.2.2 join_matrices_vertical()

```

template<typename T >
matrix<T> join_matrices_vertical (
    const matrix< T > & u,
    const matrix< T > & d )
  
```

Złącza macierze w pionie.

Oto graf wywołań dla tej funkcji:



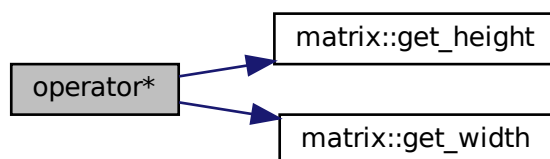
10.8.2.3 operator*()

```

template<typename T , typename U , typename V = std::common_type_t<T, U>>
matrix<V> operator* (
    const matrix< T > & lhs,
    const matrix< U > & rhs )
  
```

Operator mnożenia macierzowego.

Oto graf wywołań dla tej funkcji:

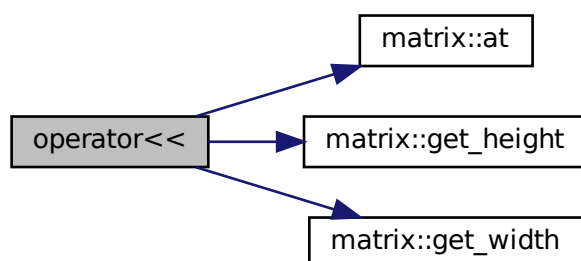


10.8.2.4 operator<<()

```
template<typename T >
std::ostream& operator<< (
    std::ostream & s,
    const matrix< T > & mat )
```

Operator wypisania dla macierzy.

Oto graf wywołań dla tej funkcji:

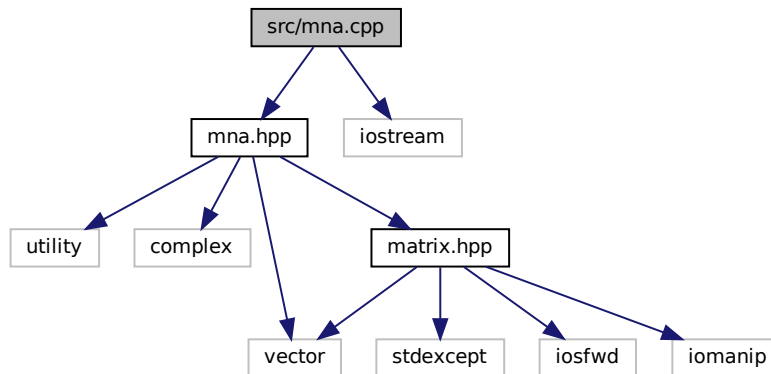


10.9 Dokumentacja pliku src/mna.cpp

Implementacja algorytmu MNA z eliminacją Gaussa.

```
#include "mna.hpp"
#include <iostream>
```

Wykres zależności załączania dla mna.cpp:



Funkcje

- static `matrix< std::complex< double > >` `gaussian_elimination` (`matrix< std::complex< double > >` mat)

Rozwiązuje układ zespolonych równań liniowych metodą eliminacji Gaussa.

10.9.1 Opis szczegółowy

Implementacja algorytmu MNA z eliminacją Gaussa.

Ten plik implementuje narzędzia do analizy układów na "najniższym poziomie". Analizowany układ musi zostać uprzednio zdegenerowany do opisu problemu, na który składają się admitancje międzywęzłowe, źródła napięciowe i prądowe oraz idealne wzmacniacze operacyjne (pracujące z ujemnym sprzężeniem zwrotnym).

Na tym poziomie obowiązuje numeracja węzłów od 0. Węzły o numerach ujemnych traktowane są jako napięcie odniesienia (masa).

Zobacz również

<https://www.swarthmore.edu/NatSci/echeevel/Ref/mna/MNA3.html>

Autor

Jacek Wiczorek

10.9.2 Dokumentacja funkcji

10.9.2.1 `gaussian_elimination()`

```
static matrix<std::complex<double> > gaussian_elimination (
    matrix< std::complex< double > > mat ) [static]
```

Rozwiązuje układ zespolonych równań liniowych metodą eliminacji Gaussa.

Parametry

<i>mat</i>	Macierz o rozmiarze $N \times (N+1)$ opisująca układ równań
------------	---

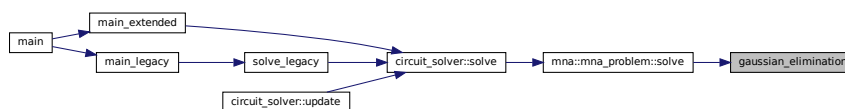
Zwraca

Macierz o rozmiarze $N \times 1$ zawierająca rozwiązanie

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:

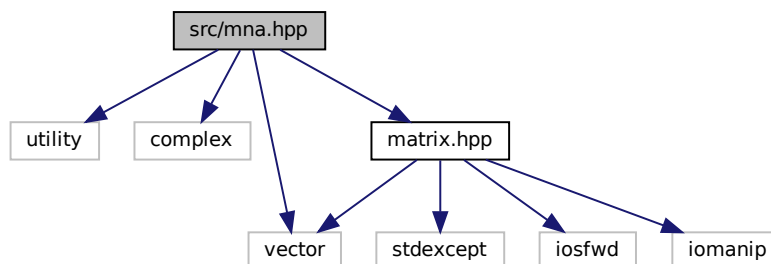


10.10 Dokumentacja pliku src/mna.hpp

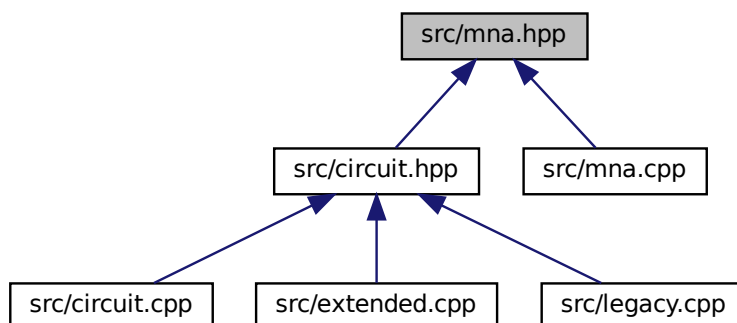
Definicje typów solvera MNA.

```
#include <utility>
#include <complex>
#include <vector>
#include "matrix.hpp"
```

Wykres zależności załączania dla mna.hpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- struct `mna::admittance`
Admitancja międzywęzłowa.
- struct `mna::voltage_source`
Idealna siła elektromotoryczna.
- struct `mna::current_source`
Idealna siła prądomotoryczna.
- struct `mna::opamp`
Idealny wzmacniacz operacyjny.
- class `mna::mna_solution`
Wynik analizy układu metodą MNA.
- struct `mna::mna_problem`
Układ do rozwiązania metodą MNA.

Przestrzenie nazw

- `mna`
Solver układów metodą MNA.

10.10.1 Opis szczegółowy

Definicje typów solvera MNA.

Autor

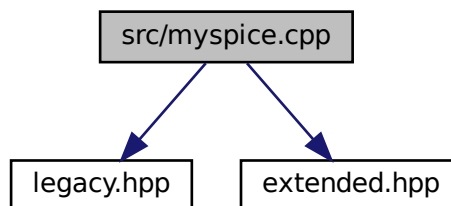
Jacek Wieczorek

10.11 Dokumentacja pliku src/myspice.cpp

Główny plik programu.

```
#include "legacy.hpp"  
#include "extended.hpp"
```

Wykres zależności załączania dla myspice.cpp:



Funkcje

- `int main (int argc, char *argv[])`

10.11.1 Opis szczegółowy

Główny plik programu.

Autor

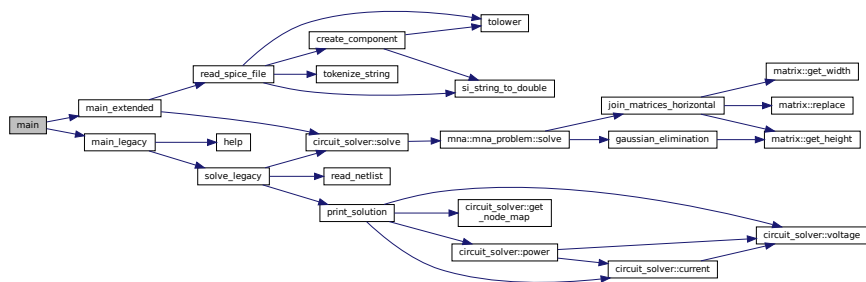
Jacek Wieczorek

10.11.2 Dokumentacja funkcji

10.11.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

Oto graf wywołań dla tej funkcji:



Indeks

- ~circuit_component
 - circuit_component, 30
- ~matrix
 - matrix< T >, 49
- ac
 - circuit_simulation, 31
- ac_analysis_params, 23
 - exponent, 24
 - start, 24
 - steps, 24
 - stop, 24
- acI
 - current_source, 45
- acV
 - voltage_source, 76
- admittance
 - capacitor, 29
 - inductor, 47
 - passive_component, 65
 - resistor, 71
- admittances
 - mna::mna_problem, 57
- at
 - matrix< T >, 50
- bipole_component, 26
 - bipole_component, 27
 - nodes, 27
 - passive_component, 65
- C
 - capacitor, 29
- capacitor, 27
 - admittance, 29
 - C, 29
 - capacitor, 29
- circ
 - circuit_simulation, 32
- circuit
 - circuit.hpp, 81
- circuit.hpp
 - circuit, 81
- circuit_component, 30
 - ~circuit_component, 30
- circuit_simulation, 31
 - ac, 31
 - circ, 32
 - probes, 32
 - title, 32
- circuit_solver, 32
 - circuit_solver, 33
 - current, 34
 - get_node_map, 34
 - get_solution, 35
 - get_solution_omega, 35
 - power, 35, 36
 - solve, 36
 - update, 37
 - voltage, 37, 38
- complex_probing_method
 - extended.cpp, 83
- create_component
 - extended.cpp, 84
- current
 - circuit_solver, 34
- current_probe, 39
 - current_probe, 40
 - get_value, 41
 - m_probing_method, 41
 - m_ref, 42
- current_source, 43
 - acI, 45
 - current_source, 44
 - dcl, 45
- current_sources
 - mna::mna_problem, 57
- data
 - matrix< T >, 51
- dcl
 - current_source, 45
- dcV
 - voltage_source, 76
- DEFAULT
 - extended.cpp, 83
- exponent
 - ac_analysis_params, 24
- extended.cpp
 - complex_probing_method, 83
 - create_component, 84
 - DEFAULT, 83
 - IMAGINARY, 83
 - MAGNITUDE, 83
 - main_extended, 84
 - PHASE, 83
 - probe_complex, 85
 - probing_method_suffix, 86
 - read_spice_file, 86

- REAL, 83
- si_string_to_double, 87
- tokenize_string, 87
- tolower, 88
- extended.hpp
 - main_extended, 89
- gaussian_elimination
 - mna.cpp, 100
- get_height
 - matrix< T >, 51
- get_matrix
 - mna::mna_solution, 58
- get_name
 - probe, 69
- get_node_map
 - circuit_solver, 34
- get_solution
 - circuit_solver, 35
- get_solution_omega
 - circuit_solver, 35
- get_value
 - current_probe, 41
 - power_probe, 67
 - probe, 69
 - voltage_probe, 74
- get_width
 - matrix< T >, 52
- help
 - legacy.cpp, 90
- I
 - mna::current_source, 43
- IMAGINARY
 - extended.cpp, 83
- inductor, 45
 - admittance, 47
 - inductor, 46
 - L, 47
- join_matrices_horizontal
 - matrix.hpp, 97
- join_matrices_vertical
 - matrix.hpp, 98
- L
 - inductor, 47
- legacy.cpp
 - help, 90
 - main_legacy, 91
 - print_solution, 91
 - read_netlist, 92
 - solve_legacy, 93
- legacy.hpp
 - main_legacy, 95
- m_name
 - probe, 69
- m_probing_method
 - current_probe, 41
- m_ref
 - current_probe, 42
- MAGNITUDE
 - extended.cpp, 83
- main
 - myspice.cpp, 103
- main_extended
 - extended.cpp, 84
 - extended.hpp, 89
- main_legacy
 - legacy.cpp, 91
 - legacy.hpp, 95
- matrix
 - matrix< T >, 49
- matrix< T >, 48
 - ~matrix, 49
 - at, 50
 - data, 51
 - get_height, 51
 - get_width, 52
 - matrix, 49
 - operator*, 53
 - operator(), 52, 53
 - operator+, 53
 - operator=, 53, 54
 - replace, 54
 - transpose, 54
- matrix.hpp
 - join_matrices_horizontal, 97
 - join_matrices_vertical, 98
 - operator<<, 99
 - operator*, 98
- mna, 21
- mna.cpp
 - gaussian_elimination, 100
- mna::admittance, 25
 - nodes, 25
 - Y, 25
- mna::current_source, 42
 - I, 43
 - nodes, 43
- mna::mna_problem, 55
 - admittances, 57
 - current_sources, 57
 - opamps, 57
 - solve, 56
 - voltage_sources, 57
- mna::mna_solution, 57
 - get_matrix, 58
 - mna_solution, 58
 - opamp_current, 58
 - voltage, 59
 - voltage_source_current, 59
- mna::opamp, 62
 - neg_input_node, 63
 - output_node, 63
 - pos_input_node, 63

- mna::voltage_source, 77
 - nodes, 77
 - V, 77
- mna_solution
 - mna::mna_solution, 58
- mypice.cpp
 - main, 103
- mypice.dox, 79
- neg_input_node
 - mna::opamp, 63
 - opamp, 62
- nodes
 - bipole_component, 27
 - mna::admittance, 25
 - mna::current_source, 43
 - mna::voltage_source, 77
- opamp, 60
 - neg_input_node, 62
 - opamp, 61
 - output_node, 62
 - pos_input_node, 62
- opamp_current
 - mna::mna_solution, 58
- opamps
 - mna::mna_problem, 57
- operator<<
 - matrix.hpp, 99
- operator*
 - matrix< T >, 53
 - matrix.hpp, 98
- operator()
 - matrix< T >, 52, 53
- operator+
 - matrix< T >, 53
- operator=
 - matrix< T >, 53, 54
- output_node
 - mna::opamp, 63
 - opamp, 62
- passive_component, 64
 - admittance, 65
 - bipole_component, 65
- PHASE
 - extended.cpp, 83
- pos_input_node
 - mna::opamp, 63
 - opamp, 62
- power
 - circuit_solver, 35, 36
- power_probe, 66
 - get_value, 67
 - power_probe, 67
- print_solution
 - legacy.cpp, 91
- probe, 68
 - get_name, 69
 - get_value, 69
 - m_name, 69
- probe_complex
 - extended.cpp, 85
- probes
 - circuit_simulation, 32
- probing_method_suffix
 - extended.cpp, 86
- R
 - resistor, 71
- read_netlist
 - legacy.cpp, 92
- read_spice_file
 - extended.cpp, 86
- REAL
 - extended.cpp, 83
- replace
 - matrix< T >, 54
- resistor, 70
 - admittance, 71
 - R, 71
 - resistor, 71
- si_string_to_double
 - extended.cpp, 87
- solve
 - circuit_solver, 36
 - mna::mna_problem, 56
- solve_legacy
 - legacy.cpp, 93
- src/circuit.cpp, 79
- src/circuit.hpp, 80
- src/extended.cpp, 82
- src/extended.hpp, 88
- src/legacy.cpp, 90
- src/legacy.hpp, 94
- src/matrix.hpp, 96
- src/mna.cpp, 99
- src/mna.hpp, 101
- src/mypice.cpp, 103
- start
 - ac_analysis_params, 24
- steps
 - ac_analysis_params, 24
- stop
 - ac_analysis_params, 24
- title
 - circuit_simulation, 32
- tokenize_string
 - extended.cpp, 87
- tolower
 - extended.cpp, 88
- transpose
 - matrix< T >, 54
- update
 - circuit_solver, 37

V

- mna::voltage_source, [77](#)

voltage

- circuit_solver, [37](#), [38](#)

- mna::mna_solution, [59](#)

voltage_probe, [72](#)

- get_value, [74](#)

- voltage_probe, [73](#)

voltage_source, [74](#)

- acV, [76](#)

- dcV, [76](#)

- voltage_source, [76](#)

voltage_source_current

- mna::mna_solution, [59](#)

voltage_sources

- mna::mna_problem, [57](#)

Y

- mna::admittance, [25](#)