Metody obliczeniowe w nauce i technice 2

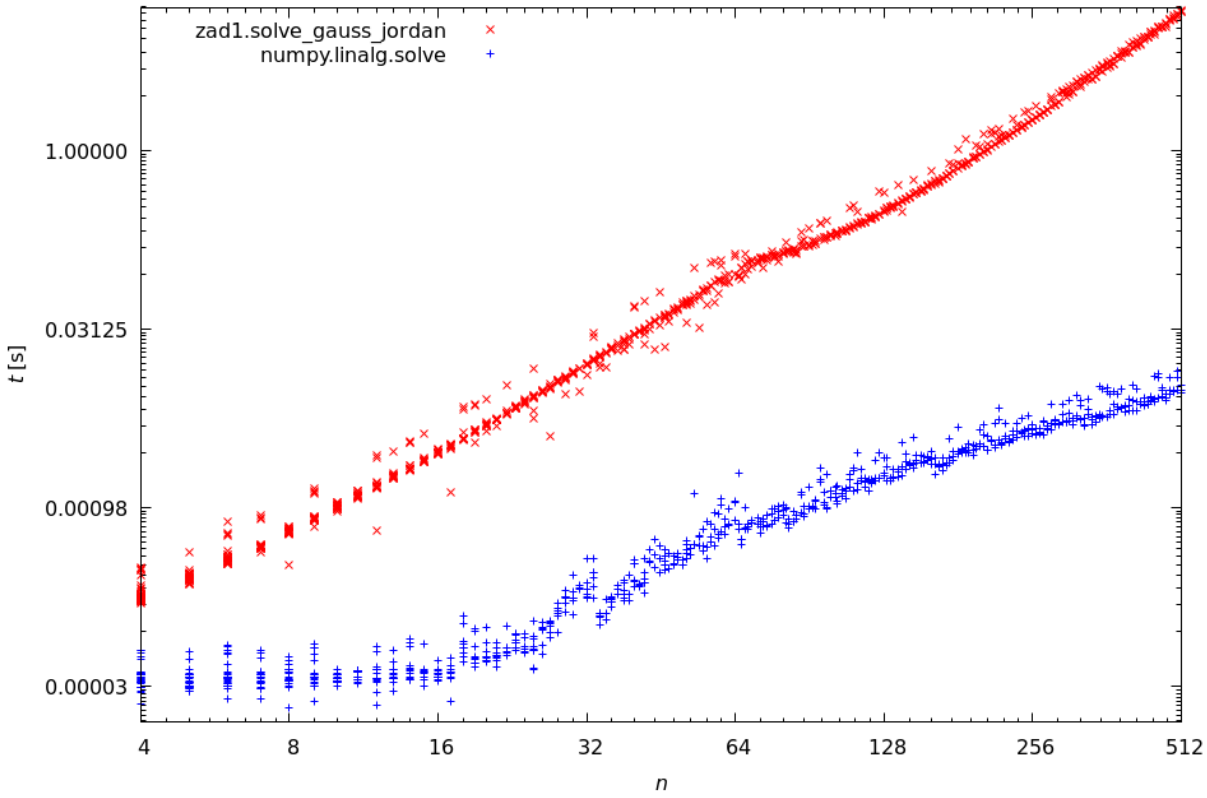
Tomasz Zawadzki

27 marca 2019

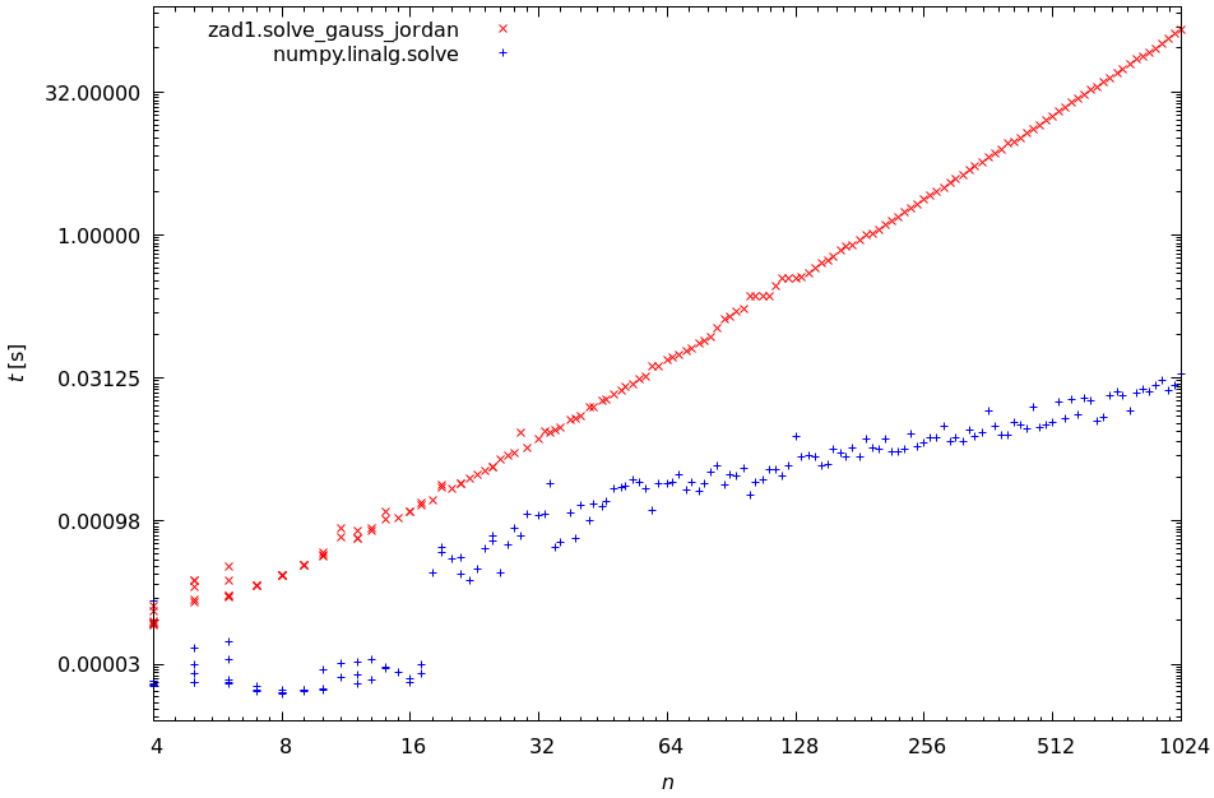
Spis treści

Zadanie 1. Metoda Gaussa-Jordana

Algorytm rozwiązywania układu równań liniowych metodą Gaussa-Jordana wraz z pełnym wyszukiwaniem elementu głównego (ang. *full pivoting*) został zaimplementowany w języku Python. Jego złożoność jest znacznie gorsza w porównaniu do algorytmu wykorzystywanego przez funkcję biblioteczną `numpy.linalg.solve`. Czas potrzebny na rozwiązanie jednego układu równań jest kilkanaście razy dłuższy, a różnica rośnie wraz ze wzrostem liczby równań w układzie.



Rys. 1. Czas rozwiązywania układu równań liniowych $n \times n$ metodą Gaussa-Jordana przy użyciu funkcji `zad1.solve_gauss_jordan` oraz `numpy.linalg.solve`.



Rys. 2. Czas rozwiązywania układu równań liniowych $n \times n$ metodą Gaussa-Jordana przy użyciu funkcji `zad1.solve_gauss_jordan` oraz `numpy.linalg.solve`.

Zadanie 2. Faktoryzacja LU

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix}}_{U} \tag{1}$$

$$Ax = b \quad \Longleftrightarrow \quad \begin{cases} Ux = z \\ Lz = b \end{cases} \tag{2}$$

Korzystając ze wzoru pozwalającego obliczyć wyrazy a_{ij} macierzy będącej iloczynem dwóch macierzy oraz z faktu, że macierze L, U są trójkątne otrzymujemy

$$a_{ij} = \sum_{s=1}^{n} l_{is}u_{sj} = \sum_{s=1}^{\min\{i,j\}} l_{is}u_{sj} \quad (1 \leqslant i, j \leqslant n) \tag{3}$$

Po podstawieniu $i = j = k$ do powyższego wzoru mamy

$$a_{kk} = \sum_{s=1}^{k} l_{ks}u_{sk} = \left(\sum_{s=1}^{k-1} l_{ks}u_{sk}\right) + l_{kk}u_{kk}, \tag{4}$$

co pozwala obliczyć wyraz u_{kk} macierzy U, przyjmując $l_{kk} = 1$ (metoda Doolittle'a).

Jeśli podstawimy $i = k$, to zależność przyjmuje postać

$$a_{kj} = \sum_{s=1}^{k} l_{ks}u_{sj} = \left(\sum_{s=1}^{k-1} l_{ks}u_{sj}\right) + l_{kk}u_{kj} \quad (k+1 \leqslant j \leqslant n), \tag{5}$$

natomiast po podstawieniu $j = k$ można otrzymać

$$a_{ik} = \sum_{s=1}^{k} l_{is}u_{sk} = \left(\sum_{s=1}^{k-1} l_{ik}u_{sk}\right) + l_{ik}u_{kk} \quad (k+1 \leqslant i \leqslant n), \tag{6}$$

co pozwala obliczyć wyrazy w k-tym wierszu macierzy trójkątnej dolnej L oraz w k-tej kolumnie macierzy trójkątnej górnej U.

Algorytm faktoryzacji LU w wersji wraz ze skalowaniem wierszy został zaimplementowany w języku Python. Oto fragment kodu źródłowego funkcji, która dokonuje rozkładu LU macierzy:

```
# LU factorization
L = np.identity(n) # Doolittle's method
U = np.zeros((n, n))
for k in range(n):
    for j in range(k, n):
        U[k,j] = A[k,j] - sum([L[k,s]*U[s,j] for s in range(k)])
    for i in range(k+1, n):
        L[i,k] = (A[i,k] - sum([L[i,s]*U[s,k] for s in range(k)])) / U[k,k]
return L, U
```

Rozkład LU przykładowej macierzy rozmiaru 3×3:

$$\begin{bmatrix} 5 & 3 & 2 \\ 1 & 2 & 0 \\ 3 & 0 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{5} & 1 & 0 \\ \frac{3}{5} & -\frac{9}{7} & 1 \end{bmatrix} \begin{bmatrix} 5 & 3 & 2 \\ 0 & \frac{7}{5} & -\frac{2}{5} \\ 0 & 0 & \frac{16}{7} \end{bmatrix} \tag{7}$$

Wynik wykonania programu `zad2/example.py`:

```
A = [[5. 3. 2.]
     [1. 2. 0.]
     [3. 0. 4.]]
L = [[ 1.          0.          0.        ]
     [ 0.2         1.          0.        ]
     [ 0.6        -1.28571429  1.        ]]
U = [[ 5.          3.          2.        ]
     [ 0.          1.4        -0.4       ]
     [ 0.          0.          2.28571429]]
```

Zadanie 3. Analiza obwodu elektrycznego

Algorytm obliczający natężenia prądu w każdej części obwodu został zaimplementowany w języku Python. Obwód elektryczny jest reprezentowany jako graf, który jest tworzony oraz przetwarzany przy użyciu biblioteki `networkx`. Rozwiązanie układu równań liniowych jest obliczane przy użyciu pakietu numerycznego `numpy`. Wizualizacja grafu jest realizowana przy pomocy biblioteki `matplotlib`.

Tworzenie grafu skierowanego reprezentującego obwód elektryczny

Obwód elektryczny jest reprezentowany przez skierowany graf ważony $G = (V, E)$. Wagą krawędzi jest opór odpowiadającego jej rezystora, a zwrot krawędzi jest określony przez kolejność węzłów we wczytywanej parze liczb. Skierowanie krawędzi jest równoważne założeniu, że prąd płynie zgodnie ze zwrotem tej krawędzi. Ujemne wartości natężenia prądu będą oznaczały, że w rzeczywistości prąd płynie przeciwnie niż wskazuje zwrot danej krawędzi.

```
# construct graph representing electric circuit
edges = [(1, 2, 100), (2, 3, 200), ...]
G = nx.DiGraph()
for (u, v, R) in edges:
    G.add_edge(u, v, R=R)
```

Przyłożenie siły elektromotorycznej

Trójka liczb (s, t, E) reprezentuje dwa węzły s oraz t, pomiędzy które została przyłożona siła elektromotoryczna E. Z fizycznego punktu widzenia jest to więc fragment przewodu o zerowym oporze i napięciu E pomiędzy końcami, który jest elementem obwodu, zatem należy dodać taką krawędź do grafu.

```
# apply the electromotive force
G.add_edge(s, t, R=0)
```

Tworzenie macierzy oraz wektora prawej strony

Kolejnym etapem jest wygenerowanie równań, które pozwolą obliczyć natężenia prądu płynącego przez poszczególne części obwodu. W tym celu tworzona jest macierz rozmiaru $|E| \times |E|$, a także $|E|$-elementowy wektor współczynników prawych stron wszystkich równań.

```
# create matrix
e = G.number_of_edges()
M = np.zeros((e, e))
B = np.zeros(e)
edgelist = list(G.edges()) # for column indexing
```

Tworzona jest również lista wszystkich krawędzi grafu, która stanowi bijekcję między zbiorem wszystkich krawędzi grafu a indeksami kolejnych wierszy macierzy. Kolejne współrzędne wektora rozwiązania tego układu równań będą natężeniami prądu płynącego przez krawędzie na tej liście w odpowiedniej kolejności.

Zastosowanie drugiego prawa Kirchhoffa dla cykli bazowych

Funkcja biblioteczna `nx.cycle_basis(G)` zwraca listę wszystkich cykli bazowych grafu nieskierowanego. Utworzona wcześniej macierz jest uzupełniana zgodnie z następującymi regułami. Dla każdego cyklu bazowego należy zastosować prawo napięć Kirchhoffa

$$\sum_i U_i = 0, \tag{8}$$

a dla każdej krawędzi reprezentującej rezystor należy wykorzystać prawo Ohma

$$R = \frac{U}{I} \quad \text{lub} \quad U = IR. \tag{9}$$

Cykl jest reprezentowany przez listę węzłów. Każde dwa kolejne węzły na liście (a także ostatni z pierwszym) reprezentują krawędź cyklu, skierowaną zgodnie z kierunkiem przechodzenia cyklu. Dla każdego cyklu C natężenia prądu płynącego przez jego krawędzi spełniają równanie

$$\sum_{(u,v)\in C} U_{(u,v)} = 0, \tag{10}$$

które po przekształceniu można zapisać w postaci

$$\sum_{i=1}^{|E|} \pm R_i I_i = \pm R_1 I_1 + \pm R_2 I_2 + \ldots + \pm R_{|E|} I_{|E|} = \pm E, \tag{11}$$

gdzie: I_i to niewiadome oznaczające natężenie prądu płynącego przez i-tą krawędź (zgodnie z kolejnością na zapisanej wcześniej liście); współczynniki R_i oznaczają wagi poszczególnych krawędzi (opory rezystorów), jeśli krawędź występuje w cyklu, albo 0, jeśli krawędź nie należy do cyklu; znaki przy współczynnikach R_i oraz E zależą od zgodności kierunków krawędzi grafu oraz kierunku przechodzenia cyklu. Jeśli krawędź reprezentująca źródło siły elektromotorycznej nie występuje w cyklu, to prawą stroną równania jest 0.

```
# apply the Kirchhoff's second law for cycle basis
cycles = nx.cycle_basis(G.to_undirected())
for i, cycle in enumerate(cycles):
    for pair in pairs(cycle):
        if pair == (s, t):
            B[i] = E
        elif pair == (t, s):
            B[i] = -E
        else:
            (u, v) = pair
            if (u, v) in edgelist:
                j = edgelist.index((u, v))
                M[i][j] = G[u][v]['R']
            else:
                j = edgelist.index((v, u))
                M[i][j] = -G[v][u]['R']
```

Zastosowanie pierwszego prawa Kirchhoffa dla pozostałych węzłów

Dla każdego węzła obwodu możemy zastosować pierwsze prawo Kirchhoffa (prawo natężeń)

$$\sum_i I_i = 0, \tag{12}$$

które można zapisać równoważnie w postaci

$$\sum_{i=1}^{|V|} X_i I_i = 0, \tag{13}$$

gdzie I_i to niewiadome oznaczające natężenie prądu płynącego przez i-tą krawędź (zgodnie z kolejnością na zapisanej wcześniej liście), a współczynniki X_i przyjmują wartość 1, jeśli krawędź jest skierowana do węzła; -1, jeśli krawędź jest skierowana od węzła; 0, jeśli krawędź nie jest incydentna z danym węzłem.

Warto zauważyć, że układ $|V|$ równań powstałych w wyniku zastosowania pierwszego prawa Kirchhoffa dla każdego węzła nie jest liniowo niezależny. Można natomiast udowodnić, że w przypadku obwodu reprezentowanego przez graf spójny układ dowolnych $|V| - 1$ równań jest liniowo niezależny.

Dla każdego grafu spełniona jest zależność

$$|C| = |E| - |V| + c, \tag{14}$$

gdzie $|C|$ oznacza liczbę cykli w grafie, natomiast c – liczbę składowych grafu. W przypadku grafu spójnego $c = 1$, więc po przekształceniu

$$|E| = |C| + |V| - 1. \tag{15}$$

Wynika z tego, że układ $|C|$ równań powstałych przez zastosowanie drugiego prawa Kirchhoffa oraz dodatkowych $|V| - 1$ równań wynikających z pierwszego prawa Kirchhoffa, daje układ $|E|$ równań liniowych, z których można obliczyć natężenia prądu płynącego przez poszczególnych $|E|$ krawędzi grafu reprezentującego obwód elektryczny.

```
# apply the Kirchhoff's first law for each node
C = len(cycles)
for i, x in enumerate(G.nodes()):
    if C+i == len(edgelist):
        break
    for (u, x) in G.in_edges(x):
        j = edgelist.index((u, x))
        M[C+i,j] = 1
    for (x, v) in G.out_edges(x):
        j = edgelist.index((x, v))
        M[C+i,j] = -1
```

Rozwiązanie układu równań liniowych

W celu rozwiązania powstałego układu równań wykorzystana została funkcja `numpy.linalg.solve` z pakietu numerycznego NumPy. Aby móc wizualizować przepływ prądu przez kolory krawędzi, potrzebna będzie jeszcze maksymalna wartość natężenia prądu.

```
# solve linear system
I = np.linalg.solve(M, B)
I_max = max(I)
```

Dodawanie obliczonych natężeń do grafu skierowanego

Jeśli obliczone natężenie prądu płynącego przez krawędź jest ujemne, oznacza to, że w rzeczywistości prąd płynie przez krawędź przeciwnie do jej zwrotu - w takiej sytuacji należy odwrócić krawędź oraz zmienić znak natężenia prądu na dodatni.

```
# add currents to graph
for i, (u, v) in enumerate(G.copy().edges()):
    if I[i] < 0: # reverse edge if current is negative
        R = G.edges[u, v]['R']
        G.remove_edge(u, v)
        G.add_edge(v, u, R=R)
        (u, v), I[i] = (v, u), -I[i]
    G.edges[u, v]['I'] = I[i]
```

Weryfikacja rozwiązania

Mając graf skierowany z nieujemnymi wartościami natężenia prądu dla wszystkich krawędzi, można dokonać weryfikacji rozwiązania w oparciu o pierwsze i drugie prawo Kirchhoffa. Z uwagi na niedoskonałości arytmetyki zmiennoprzecinkowej, a w szczególności wykorzystywanych metod numerycznych, otrzymana suma może być niezerowa, dlatego jest porównywana do zera z dokładnością określoną przez parametry `I_eps` oraz `U_eps`.

Zastosowanie drugiego prawa Kirchhoffa dla cykli bazowych

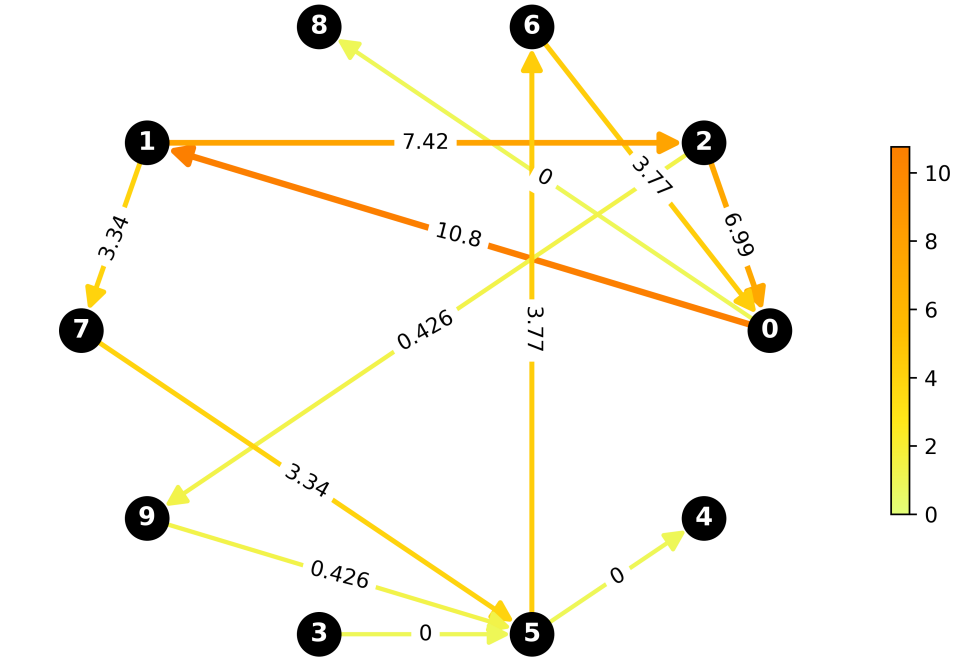
```
# verify the Kirchhoff's second law for cycle basis
for x in G.nodes():
    I = 0
    for e in G.in_edges(x):
        I += G.edges[e]['I']
    for e in G.out_edges(x):
        I -= G.edges[e]['I']
    if I > I_eps * I_max:
        raise Exception("Verification failed for Kirchhoff's second law")
```

Zastosowanie pierwszego prawa Kirchhoffa dla pozostałych węzłów

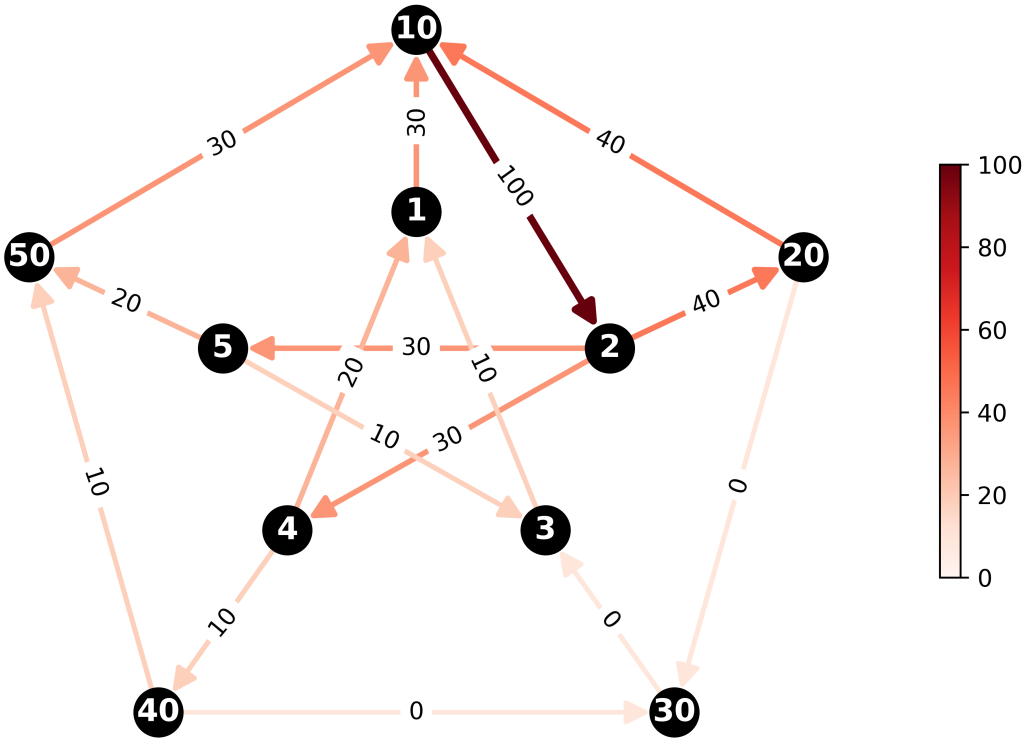
```
# verify the Kirchhoff's first law for each node
for i, cycle in enumerate(cycles):
    U = 0
    for pair in pairs(cycle):
        if pair == (s, t):
            U += E
        elif pair == (t, s):
            U -= E
        else:
            (u, v) = pair
            if (u, v) in G.edges():
                U -= G.edges[u, v]['R'] * G.edges[u, v]['I']
            else: # (v, u) in G.edges()
                U += G.edges[v, u]['R'] * G.edges[v, u]['I']
    if U > U_eps * E:
        raise Exception("Verification failed for Kirchhoff's first law")
```

Wizualizacja obwodu elektrycznego

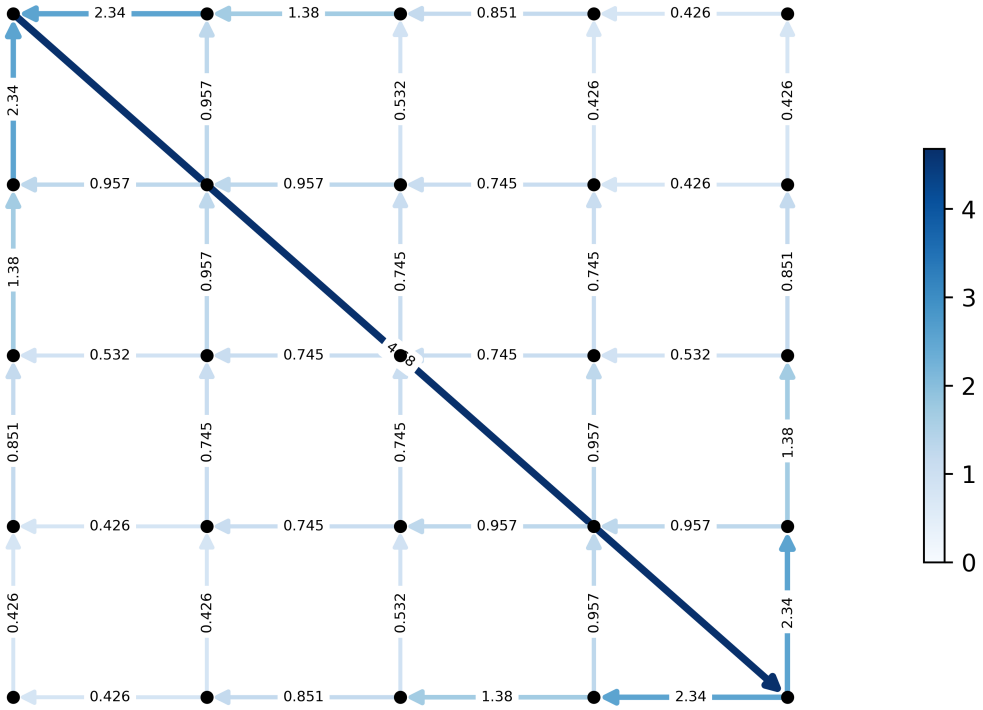
W zależności od przekazanych parametrów, odpowiednia wizualizacja obwodu elektrycznego jest wyświetlana na ekranie albo zapisywana do pliku.



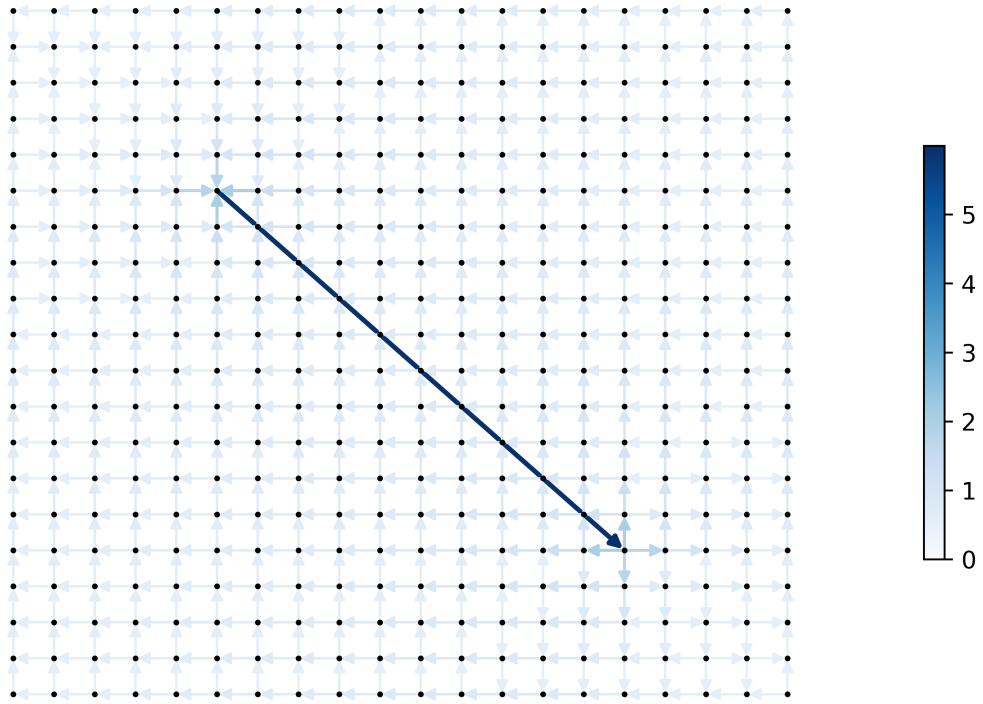
Rys. 3. Graf losowy, $|V| = 10$



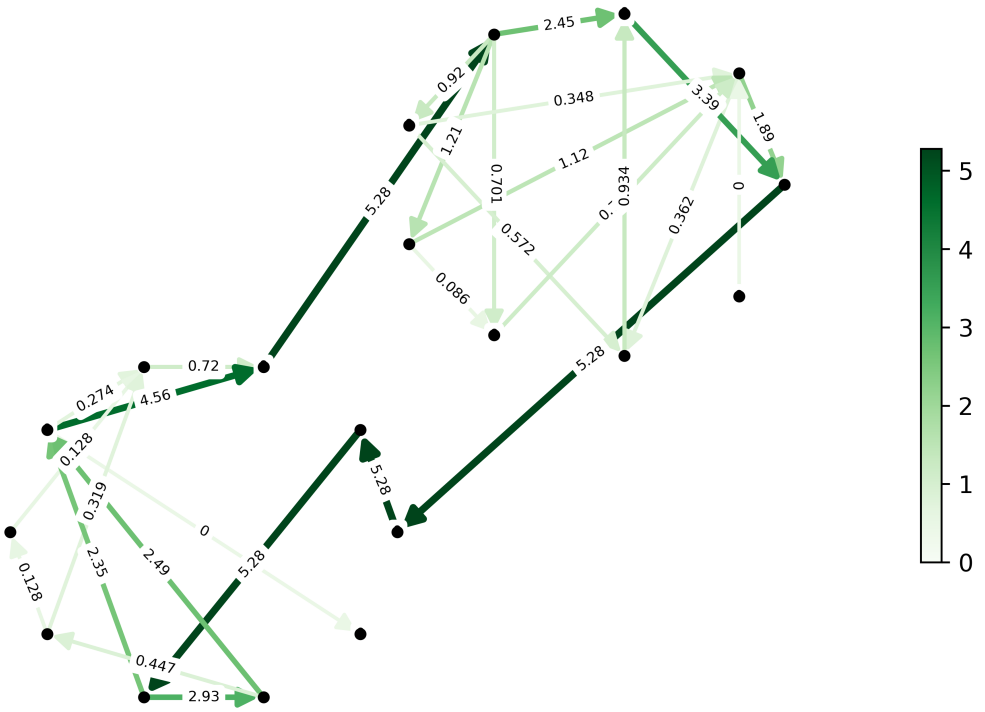
Rys. 4. Graf 3-regularny kubiczny (Petersena)



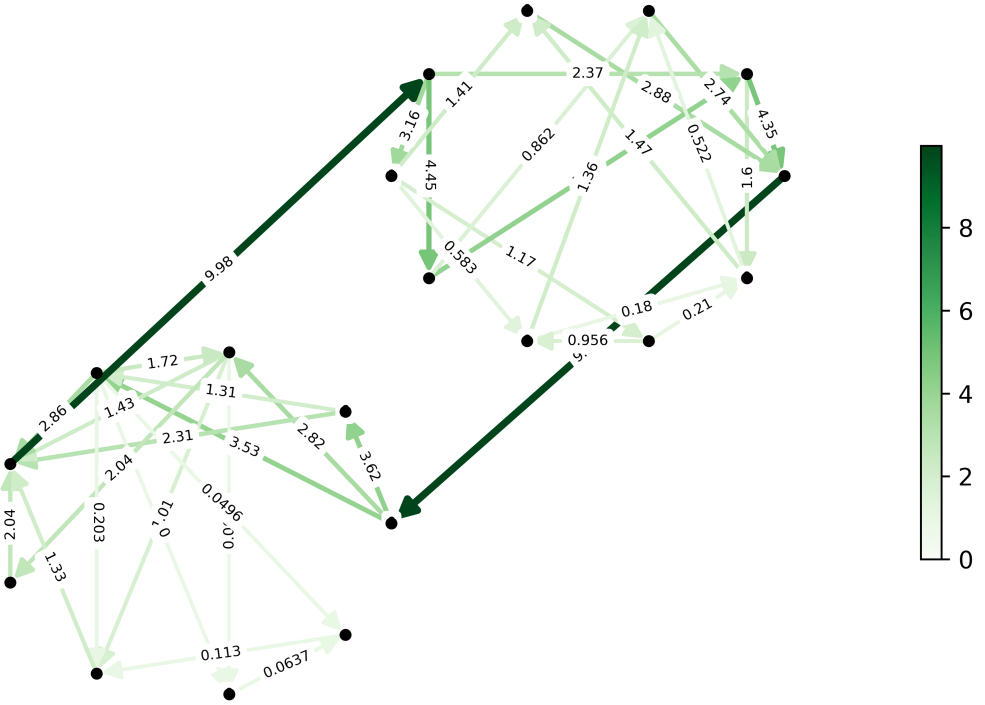
Rys. 5. Graf siatka 5×5



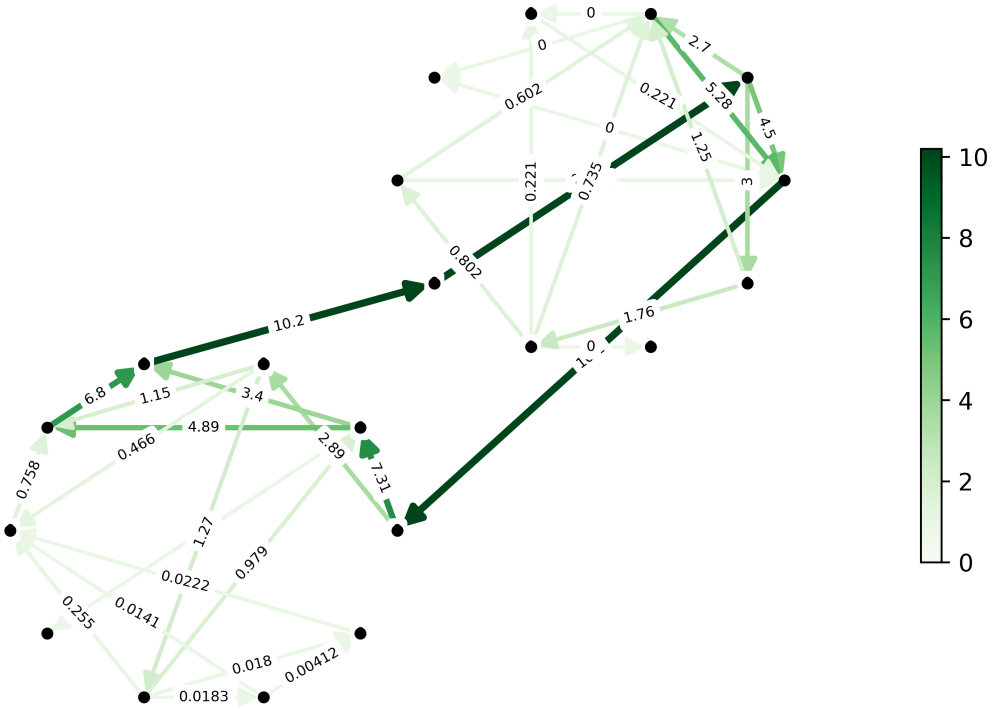
Rys. 6. Graf siatka 20×20



Rys. 7. Graf złożony z dwóch grafów losowych połączonych mostem, $|V| = 20$



Rys. 8. Graf złożony z dwóch grafów losowych połączonych mostem, $|V| = 20$



Rys. 9. Graf złożony z dwóch grafów losowych połączonych mostem, $|V| = 20$

Bibliografia

[1] D. Kincaid, W. Cheney "Numerical Mathematics and Computing"

[2] `https://en.wikipedia.org/wiki/Cycle_space#Circuit_rank`

[3] `https://docs.scipy.org/doc/`

[4] `https://matplotlib.org/contents.html`

[5] `https://networkx.github.io/documentation/latest/`