Numeryczne rozwiązywanie układów równań liniowych

Implementacja praktyczna

Wstęp

Poniższa implementacja praktyczna stanowi uzupełnienie teoretycznych rozważań na temat metod numerycznych rozwiązywania układów równań liniowych i obejmuje następujące zagadnienia:

- Rozwiązywanie układów z trójkątną macierzą współczynników przy użyciu metody podstawień
- Metoda eliminacji Gaussa bez oraz z częściowym i całkowitym wyborem elementu głównego
- Metoda LU na bazie zmodyfikowanej metody eliminacji Gaussa oraz metody Doolittle'a
- Metoda iteracyjna Jacobiego
- Metoda iteracyjna Gaussa-Seidla

Pomocne narzędzia

Poniżej zdefiniowałem garść pomocnych narzędzi do oceniania jakości rozwiązania:

Poniższa funkcja porównuje uzyskane wyniki (wektor **x**) z wynikami obliczonymi bezpośrednio z zależności: $\mathbf{x = A^{-1}b}$ i zwraca błąd bezwzględny.

$$\text{blad\_rozwiazania}(A, b, x) := A^{-1} b - x$$

Kolejna funkcja służy do poziomego 'sklejenia' dwóch macierzy o takiej samej liczbie wierszy:

$$\text{sklej\_macierze\_poziomo}(lhs, rhs) := \left| \begin{array}{l} \text{return } (-1) \text{ if } \text{rows}(lhs) \neq \text{rows}(rhs) \\ \text{for } i \in 0..\text{cols}(rhs) - 1 \\ \quad lhs^{\langle \text{cols}(lhs)+i \rangle} \leftarrow rhs^{\langle i \rangle} \\ \text{return } lhs \end{array} \right.$$

Następne dwie funkcje realizują czynność odwrotną, tj. służą do wydzielenia lewej i prawej strony macierzy, tak by lewa macierz miała podaną liczbę kolumn.

$$\text{wydziel\_macierz\_lewa}(lhs, leftCols) := \left| \begin{array}{l} \text{for } i \in 0..leftCols - 1 \\ \quad leftMatrix^{\langle i \rangle} \leftarrow lhs^{\langle i \rangle} \\ \text{return } leftMatrix \end{array} \right.$$

$$\text{wydziel\_macierz\_prawa}(lhs, leftCols) := \left| \begin{array}{l} \text{for } i \in leftCols..\text{cols}(lhs) - 1 \\ \quad rightMatrix^{\langle i-leftCols \rangle} \leftarrow lhs^{\langle i \rangle} \\ \text{return } rightMatrix \end{array} \right.$$

Metody dokładne

Wstęp do metod dokładnych

Do badań w tym sprawozdaniu użyłem układu równań liniowych postaci **Ax=b** przy:

$$A := \begin{pmatrix} 1 & 8 & 2 \\ -20 & 22 & 8 \\ -3 & 5 & 17 \end{pmatrix} \qquad b := \begin{pmatrix} 44 \\ 88 \\ 99 \end{pmatrix}$$

Metoda podstawień

Implementacja metody podstawień jest konieczna w celu realizacji pozostałych metod rozwiązywania układów równań. Ponizsza funkcja spodziewa się, że macierz **A** jest macierzą trójkątną górną. Ponadto zakładamy, że nie posiada ona elementów zerowych na diagonali (głównej przekątnej).

$$\text{podstawienia\_U}(A,b) := \left| \begin{array}{l} x_{\text{last}(b)} \leftarrow \dfrac{b_{\text{last}(b)}}{A_{\text{last}(b),\text{last}(b)}} \\ \text{for } i \in \text{last}(b) - 1 .. 0 \\ \quad x_i \leftarrow \dfrac{b_i - \sum_{j=i+1}^{\text{last}(b)} \left(A_{i,j} \cdot x_j\right)}{A_{i,i}} \\ \text{return } x \end{array} \right.$$

Przetestujmy metodę podstawień dla prostego układu:

$$\text{testA} := \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{pmatrix} \qquad \text{testB} := \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

$$\text{blad\_rozwiazania}(\text{testA}, \text{testB}, \text{podstawienia\_U}(\text{testA}, \text{testB})) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Metoda eliminacji Gaussa

Obliczenia metodą eliminacji Gaussa składają się zawsze z dwóch etapów:

- Przekształcenie równania tak, by macierz współczynników była macierzą trójkątną (w tym sprawozdaniu trójkątną górną)
- Rozwiązanie równania metodą podstawień

Poniżej zaimplementowałem trzy warianty pierwszego kroku - podstawową oraz z częściowym i całkowitym wyborem elementu głównego.

Wariant podstawowy

Metoda ta obsługuje jedynie nieosobliwe macierze współczynników, w których nie ma zer na diagonali. W celu zachowania równości układu najpierw dołącza macierz **b** do macierzy **A**.

$$\begin{aligned}
&\text{gauss_basic}(A,b) := \left|\begin{array}{l}
A \leftarrow \text{sklej_macierze_poziomo}(A,b) \\
\text{for } i \in 0..\text{rows}(A)-2 \\
\quad \text{for } r \in i+1..\text{rows}(A)-1 \\
\quad\quad \left|\begin{array}{l}
\text{☼} \leftarrow \dfrac{A_{r,i}}{A_{i,i}} \\
\text{for } j \in i..\text{cols}(A)-1 \\
\quad A_{r,j} \leftarrow A_{r,j} - A_{i,j}\cdot\text{☼}
\end{array}\right. \\
\text{return } A
\end{array}\right.
\end{aligned}$$

Użyjmy tej metody do rozwiązania równania:

$$C := \text{gauss\_basic}(A,b) = \begin{pmatrix} 1 & 8 & 2 & 44 \\ 0 & 182 & 48 & 968 \\ 0 & 3.553\times 10^{-15} & 15.352 & 76.758 \end{pmatrix}$$

$$U := \text{wydziel\_macierz\_lewa}(C,3) = \begin{pmatrix} 1 & 8 & 2 \\ 0 & 182 & 48 \\ 0 & 3.553\times 10^{-15} & 15.352 \end{pmatrix}$$

$$B := \text{wydziel\_macierz\_prawa}(C,3) = \begin{pmatrix} 44 \\ 968 \\ 76.758 \end{pmatrix}$$

$$x := \text{podstawienia\_U}(U,B) = \begin{pmatrix} 2 \\ 4 \\ 5 \end{pmatrix}$$

Sprawdźmy jeszcze dokładność wyniku:

$$\text{blad\_rozwiazania}(A,b,x) = \begin{pmatrix} 0 \\ 0 \\ -1.776\times 10^{-15} \end{pmatrix}$$

Wariant z częściowym wyborem elementu głównego

Z powodów wyjaśnionych w części teoretycznej bardzo popularna jest metoda częściowego wyboru elementu głównego. Rozpatrujemy w niej nie element na głównej przekątnej, ale wybieramy największy co do modułu element w części kolumny od przekątnej w dół, a następnie (jeśli element na przekątnej nie był tym największym co do modułu) przestawiamy miejscami wiersze tak, by ten element znalazł się na diagonali.

```
gauss_partial(A, b) := | A ← sklej_macierze_poziomo(A, b)
                       | for i ∈ 0 .. rows(A) − 2
                       |     maxIdx ← i
                       |     for maxEl ∈ i + 1 .. rows(A) − 1
                       |       maxIdx ← maxEl if |A_(maxIdx, i)| < |A_(maxEl, i)|
                       |     for swp ∈ i .. cols(A) − 1
                       |         swap ← A_(i, swp)
                       |         A_(i, swp) ← A_(maxIdx, swp)
                       |         A_(maxIdx, swp) ← swap
                       |     for r ∈ i + 1 .. rows(A) − 1
                       |         ✲ ← A_(r, i) / A_(i, i)
                       |         for j ∈ i .. cols(A) − 1
                       |           A_(r, j) ← A_(r, j) − A_(i, j) · ✲
                       | return A
```

$$C := \text{gauss\_partial}(A, b) = \begin{pmatrix} -20 & 22 & 8 & 88 \\ 0 & 9.1 & 2.4 & 48.4 \\ 0 & 0 & 15.352 & 76.758 \end{pmatrix}$$

$$U := \text{wydziel\_macierz\_lewa}(C, 3) = \begin{pmatrix} -20 & 22 & 8 \\ 0 & 9.1 & 2.4 \\ 0 & 0 & 15.352 \end{pmatrix}$$

$$B := \text{wydziel\_macierz\_prawa}(C, 3) = \begin{pmatrix} 88 \\ 48.4 \\ 76.758 \end{pmatrix}$$

$$x := \text{podstawienia\_U}(U, B) = \begin{pmatrix} 2 \\ 4 \\ 5 \end{pmatrix}$$

Zauważmy, że choć uzyskane macierze U i B różniły się od tych z podstawowej metody eliminacji Gaussa, to rozwiązanie równania jest identyczne.

Wariant z całkowitym wyborem elementu głównego

Istnieją takie metody implementacji tej metody, które pozwalają uniknąć fizycznej zamiany kolejności elementów w przetwarzanej macierzy, jednak w celu ustalenia uwagi w tym sprawozdaniu poniższe implementacje dokonują takich przestawień.

W myśl o poprawieniu czytelności oraz z powodu błędów w oprogramowaniu Mathcad (plotki z Wikipedii głoszą, że firma, która ma prawa do tej wersji celowo wypuszcza oprogramowanie z błędami i każe użytkownikom płacić dodatkowo za support i patche) zmuszony zostałem do rozbicia metody na całą serię funkcji.

Funkcja koduj_macierz() dopisuje do macierzy dodatkową kolumnę z kolejnymi liczbami

całkowitymi począwszy od 0. Funkcja dekoduj_macierz() sortuje kolumny macierzy względem ostatniej kolumny, a następnie obcina ostatnią.

$$\text{funkcja\_kodujaca}(i,j) := i$$

$$\text{koduj\_macierz}(A) := A^{\langle \text{cols}(A) \rangle} \leftarrow \text{matrix}(\text{rows}(A), 1, \text{funkcja\_kodujaca})$$

$$\text{dekoduj\_macierz}(A) := \left\| \begin{array}{l} A \leftarrow \text{csort}(A, \text{cols}(A) - 1) \\ A \leftarrow \text{wydziel\_macierz\_lewa}(A, \text{cols}(A) - 1) \end{array} \right.$$

W dalszej części tego sprawozdania termin *macierz zakodowana* oznacza macierz z dołączonymi informacjami o pierwotnej kolejności. Termin *dekodowanie macierzy* oznacza takie przekształcenie macierzy, by jej zawartość była zgodna z pierwotną kolejnością elementów.

Poniższy algorytm został przystosowany do korzystania z funkcji kodujących i dekodujących macierz. Ponieważ wybór całkowity elementu głównego wiąże się z zamianą kolejności równań w układzie, dodałem do obliczeń informację o pierwotnej kolejności równań (kodowanie macierzy).

Funkcja select_main_total() implementuje całkowity wybór elementu głównego. Bierze ona poprawkę na fakt, że przekazana jej macierz A jest macierzą zakodowaną, tj. posiada informacje o kolejności.

$$\text{select\_main\_total}(A, i) := \left\| \begin{array}{l} \text{maxIdx} \leftarrow \begin{pmatrix} i \\ i \end{pmatrix} \\ \text{for } \text{maxY} \in i .. \text{rows}(A) - 1 \\ \quad \text{for } \text{maxX} \in i .. \text{cols}(A) - 3 \\ \quad\quad \text{maxIdx} \leftarrow \begin{pmatrix} \text{maxY} \\ \text{maxX} \end{pmatrix} \text{ if } \left| A_{\left(\text{maxIdx}_0, \text{maxIdx}_1\right)} \right| < \left| A_{\text{maxY}, \text{maxX}} \right| \\ \text{swap} \leftarrow A^{\langle (\text{maxIdx}_1) \rangle} \\ A^{\langle (\text{maxIdx}_1) \rangle} \leftarrow A^{\langle i \rangle} \\ A^{\langle i \rangle} \leftarrow \text{swap} \\ \text{swp} \leftarrow A_{i, \text{cols}(A)-1} \\ A_{i, \text{cols}(A)-1} \leftarrow A_{\left(\text{maxIdx}_1, \text{cols}(A)-1\right)} \\ A_{\text{maxIdx}_1, \text{cols}(A)-1} \leftarrow \text{swp} \\ \text{for } \text{swp} \in i .. \text{cols}(A) - 2 \\ \quad \left\| \begin{array}{l} \text{swap2} \leftarrow A_{\left(\text{maxIdx}_0, \text{swp}\right)} \\ A_{\left(\text{maxIdx}_0, \text{swp}\right)} \leftarrow A_{i, \text{swp}} \\ A_{i, \text{swp}} \leftarrow \text{swap2} \end{array} \right. \\ \text{return } A \end{array} \right.$$

Poniższa funkcja realizuje sam algorytm eliminacji zmiennych metodą Gaussa przy całkowitym wyborze elementu głównego. Zwracany wynik jest zakodowany.

```
gauss_total(A, b) := | A ← sklej_macierze_poziomo(A, b)
                     | A ← koduj_macierz(A)
                     | for i ∈ 0 .. rows(A) − 2
                     |     A ← select_main_total(A, i)
                     |     for r ∈ i + 1 .. rows(A) − 1
                     |         ☼ ← A_{r,i} / A_{i,i}
                     |         for j ∈ i .. cols(A) − 2
                     |             A_{r,j} ← A_{r,j} − A_{i,j} · ☼
                     | return (A)
```

$$C := \text{gauss\_total}(A, b) = \begin{pmatrix} 22 & 8 & -20 & 88 & 1 \\ 0 & 15.182 & 1.545 & 79 & 2 \\ 0 & 0 & 8.365 & 16.731 & 0 \end{pmatrix}$$

$$U := \text{wydziel\_macierz\_lewa}(C, 3) = \begin{pmatrix} 22 & 8 & -20 \\ 0 & 15.182 & 1.545 \\ 0 & 0 & 8.365 \end{pmatrix}$$

$$B := \text{wydziel\_macierz\_prawa}(C, 3) = \begin{pmatrix} 88 & 1 \\ 79 & 2 \\ 16.731 & 0 \end{pmatrix}$$

Po wykonaniu etapu eliminacji zmiennych wyodrębniłem informacje o kolejności, a pozostałą część macierzy przekazałem do funkcji realizującej metodę podstawień.

$$\text{kolejnosc} := \text{wydziel\_macierz\_prawa}(B, 1) = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}$$

$$B := \text{wydziel\_macierz\_lewa}(B, 1) = \begin{pmatrix} 88 \\ 79 \\ 16.731 \end{pmatrix}$$

$$x := \text{podstawienia\_U}(U, B) = \begin{pmatrix} 4 \\ 5 \\ 2 \end{pmatrix}$$

W celu ułożenia elementów rozwiązania w tej samej kolejności, w jakiej występują one w pierwotnym układzie równań dokleiłem do wyniku informacje o kolejności, a następnie rozkodowałem.

$$x := \text{sklej\_macierze\_poziomo}(x, \text{kolejnosc}) = \begin{pmatrix} 4 & 1 \\ 5 & 2 \\ 2 & 0 \end{pmatrix}$$

$$x := \text{dekoduj\_macierz}(x) = \begin{pmatrix} 2 \\ 4 \\ 5 \end{pmatrix}$$

Jak widzimy, uzyskany wynik jest identyczny z rozwiązaniem wyliczonym poprzednimi wersjami metody Gaussa.

Metoda LU

W metodzie LU macierz współczynników układu zostaje rozbita na dwie macierze **L** (*ang. lower*) i **U** (*ang. upper*), które są odpowiednio macierzami trójkątnymi dolną i górną, a ich iloczyn równy jest rozbijanej macierzy, tj.:
LU = A

Mając już rozkład LU macierzy musimy rozwiązać dwa równania macierzowe w celu uzyskania wektora wyników **x**, który jest rozwiązaniem zadanego układu równań liniowych.
Ly=b
Ux=y
Ponieważ macierze **L** i **U** są macierzami trójkątnymi, oba te równania możemy rozwiązać przy użyciu metody podstawień.

Do rozwiązywania tych równań przyda nam się alternatywna funkcja realizująca metodę podstawień, tym razem dla macierzy trójkątnej dolnej:

$$\text{podstawienia\_L}(A, b) := \left| \begin{array}{l} x_0 \leftarrow \dfrac{b_0}{A_{0,0}} \\ \text{for } i \in 1 .. \text{last}(b) \\ \quad x_i \leftarrow \dfrac{b_i - \sum_{j=0}^{i} \left(\text{if}\left(i \neq j, A_{i,j} \cdot x_j, 0\right)\right)}{A_{i,i}} \\ \text{return } x \end{array} \right.$$

W tym sprawozdaniu omówiłem dwie metody uzyskiwania rozkładu LU macierzy.

Zmodyfikowana metoda Gaussa

W tej metodzie stosujemy podstawową eliminację Gaussa (warianty z częściowym i całkowitym wyborem elementu głównego mogą niekoniecznie prowadzić do uzyskania rozkładu LU) w celu uzyskania macierzy górnej (**U**), równocześnie wyliczając 'między krokami' wartości macierzy dolnej (**L**).

Ponieważ w tym przypadku diagonala macierzy dolnej składa się z samych jedynek (których, mając tę informację, można nie przechowywać), algorytm tej metody można napisać w taki sposób, że będzie ona zapisywać obliczenia bezpośrednio do przeliczanej macierzy współczynników, dzięki czemu nie jest konieczna dodatkowa alokacja pamięci na wynikowe macierze **L** i **U**. W tym sprawozdaniu implementacja zwraca macierz zawierającą w sobie jednocześnie macierze **L** i **U** zgodnie z duchem oszczędności pamięci, ale nie zmienia przekazywanej jej macierzy współczynników. Poniższa funkcja realizuje rozkład LU:

```
gaussian_LU(A) := | for i ∈ 0 .. rows(A) − 2
                  |   for r ∈ i + 1 .. rows(A) − 1
                  |     ☼ ← A_{r,i} / A_{i,i}
                  |     for j ∈ i .. cols(A) − 1
                  |       A_{r,j} ← A_{r,j} − A_{i,j} · ☼
                  |     A_{r,i} ← ☼
                  | return A
```

Do weryfikacji obliczeń przydadzą się funkcje wydzielające macierze L i U:

```
wydziel_L(A) := | for i ∈ 0 .. rows(A) − 1
                |   for j ∈ 0 .. i
                |     L_{i,j} ← A_{i,j}  if i ≠ j
                |     L_{i,j} ← 1  otherwise
                | return L
```

```
wydziel_U(A) := | for i ∈ 0 .. rows(A) − 1
                |   for j ∈ i .. cols(A) − 1
                |     U_{i,j} ← A_{i,j}
                | return U
```

Rozkład LU dla zadanej macierzy A:

$$LU := gaussian\_LU(A) = \begin{pmatrix} 1 & 8 & 2 \\ -20 & 182 & 48 \\ -3 & 0.159 & 15.352 \end{pmatrix}$$

$$L := wydziel\_L(LU) = \begin{pmatrix} 1 & 0 & 0 \\ -20 & 1 & 0 \\ -3 & 0.159 & 1 \end{pmatrix}$$

$$U := wydziel\_U(LU) = \begin{pmatrix} 1 & 8 & 2 \\ 0 & 182 & 48 \\ 0 & 0 & 15.352 \end{pmatrix}$$

Sprawdźmy, czy rozkład funkcjonuje dokonany poprawnie:

$$L \cdot U = \begin{pmatrix} 1 & 8 & 2 \\ -20 & 22 & 8 \\ -3 & 5 & 17 \end{pmatrix} \qquad A = \begin{pmatrix} 1 & 8 & 2 \\ -20 & 22 & 8 \\ -3 & 5 & 17 \end{pmatrix} \qquad L \cdot U - A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -3.553 \times 10^{-15} & 0 \end{pmatrix}$$

Rozkład LU został wykonany poprawnie, więc poniżej przystępujemy do rozwiązania dwóch równań macierzowych w celu uzyskania rozwiązania naszego głównego równania **Ax=b**.

$$y := \text{podstawienia\_L}(L, b) = \begin{pmatrix} 44 \\ 968 \\ 76.758 \end{pmatrix}$$

$$x := \text{podstawienia\_U}(U, y) = \begin{pmatrix} 2 \\ 4 \\ 5 \end{pmatrix}$$

Rozkład LU przy użyciu zmodyfikowanej metody eliminacji Gaussa daje poprawne rozwiązanie.

Metoda Doolittle'a

Metoda Doolittle'a (nie mylić z dr. med. Dolittle) to alternatywny sposób wyliczenia rozkładu LU. W przeciwieństwie do poprzedniej metody nazwa nie mówi nam wiele na temat samego algorytmu (nazwisko Doolittle można próbować odczytywać jako 'do little' - z ang. robić niewiele...), jednak podobnie jak w przypadku zmodyfikowanej metody Gaussa możemy wyliczać rozkład *in sito*, w tej samej macierzy, którą badamy. Poniższa funkcja implementuje rozkład metodą Doolittle'a:

$$\begin{aligned}
&\text{doolittle}(A) := \Big| \ \text{for } j \in 0..\text{cols}(A) - 1 \\
&\quad A_{0,j} \leftarrow \frac{A_{0,j}}{A_{0,0}} \\
&\quad \text{for } i \in 1..\text{rows}(A) - 1 \\
&\quad\quad \text{for } j \in i..\text{cols}(A) - 1 \\
&\quad\quad\quad A_{i,j} \leftarrow A_{i,j} - \sum_{k=0}^{i-1} \left(A_{i,k} \cdot A_{k,j}\right) \\
&\quad\quad\quad A_{j,i} \leftarrow \frac{\left[A_{j,i} - \sum_{k=0}^{i-1} \left(A_{j,k} \cdot A_{k,i}\right)\right]}{A_{i,i}} \quad \text{if } j \neq i \\
&\quad \text{return } A
\end{aligned}$$

$$LU := \text{doolittle}(A) = \begin{pmatrix} 1 & 8 & 2 \\ -20 & 182 & 48 \\ -3 & 0.159 & 15.352 \end{pmatrix}$$

$$L := \text{wydziel\_L}(LU) = \begin{pmatrix} 1 & 0 & 0 \\ -20 & 1 & 0 \\ -3 & 0.159 & 1 \end{pmatrix}$$

$$U := \text{wydziel\_U}(LU) = \begin{pmatrix} 1 & 8 & 2 \\ 0 & 182 & 48 \\ 0 & 0 & 15.352 \end{pmatrix}$$

$$L \cdot U = \begin{pmatrix} 1 & 8 & 2 \\ -20 & 22 & 8 \\ -3 & 5 & 17 \end{pmatrix} \qquad A = \begin{pmatrix} 1 & 8 & 2 \\ -20 & 22 & 8 \\ -3 & 5 & 17 \end{pmatrix}$$

Rozkład LU przebiegł prawidłowo z czego wynika, że użycie tego rozkładu w rozwiązywaniu układu równań da poprawny wynik. Dla pewności sprawdziłem:

$$y := \text{podstawienia\_L}(L, b) = \begin{pmatrix} 44 \\ 968 \\ 76.758 \end{pmatrix}$$

$$x := \text{podstawienia\_U}(U, y) = \begin{pmatrix} 2 \\ 4 \\ 5 \end{pmatrix}$$

Istotnie, rozkład LU wykonany metodą Doolittle'a pozwala uzyskać poprawne rozwiązanie układu równań.

Metody iteracyjne

Wstęp do metod iteracyjnych

Iteracyjne metody rozwiązywania układów równań liniowych pozwalają w teorii na zmniejszenie nakładu obliczeń do stopnia zależnego bardziej od żądanej dokładności wyniku i rodzaju konkretnych danych wejściowych niż od ich ilości. Rodzina metod iteracyjnych zaimplementowanych w tym sprawozdaniu została omówiona dokładniej w części teoretycznej. Spełnia ona poniższe równanie:

$$\mathbf{x^{(i+1)} = Bx^{(i)} + c}$$

gdzie $\mathbf{x^{(n)}}$ oznacza **n**-ty krok iteracji.

Metody te to:

- Metoda Jacobiego
- Metoda Gaussa-Seidla

W części poświęconej metodom iteracyjnym przyjąłem też nowy układ równań do rozwiązania:

$$A := \begin{pmatrix} 2 & 5 & 0 \\ 0 & 2 & 2 \\ 1 & 0 & 9 \end{pmatrix} \qquad b := \begin{pmatrix} 36 \\ 30 \\ 84 \end{pmatrix}$$

Prawidłowe rozwiązanie, które posłużyło do weryfikacji implementowanych poniżej metod wygląda następująco:

$$x := A^{-1} b = \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

Rozwiązywanie metodami iteracyjnymi przerywa się po osiągnięciu zadanej dokładności, która w naszym przypadku wyniosła:

$$tolerance := \pi \cdot 10^{-6}$$

Przydatne narzędzia dla metod iteracyjnych

Dla czytelnej implementacji potrzebne były też funkcje pomocnicze wydzielające macierz trójkątną dolną z zerami na diagonali (poddiagonalną, **L**), trójkątną górną z zerami na diagonali (naddiagonalną, **U**) i macierz diagonalną (**D**) w taki sposób, by ich suma dawała w wyniku macierz współczynników, tj.:
L+D+U = A

$$wydziel\_L(A) := \begin{| for\ i \in 0..rows(A)-1 \\ \quad for\ j \in 0..cols(A)-1 \\ \quad\quad L_{i,j} \leftarrow A_{i,j}\ \ if\ i > j \\ \quad\quad L_{i,j} \leftarrow 0\ \ otherwise \\ return\ L}$$

$$wydziel\_D(A) := \begin{| for\ i \in 0..rows(A)-1 \\ \quad for\ j \in 0..cols(A)-1 \\ \quad\quad D_{i,j} \leftarrow A_{i,j}\ \ if\ i = j \\ return\ D}$$

$$wydziel\_U(A) := \begin{| for\ i \in 0..rows(A)-1 \\ \quad for\ j \in 0..cols(A)-1 \\ \quad\quad U_{i,j} \leftarrow A_{i,j}\ \ if\ i < j \\ \quad\quad U_{i,j} \leftarrow 0\ \ otherwise \\ return\ U}$$

Cała rodzina zaimplementowanych tu metod iteracyjnych korzysta z równania iteracyjnego tej samej postaci. W poniższych funkcjach parametr **xs** to pierwsze przybliżenie rozwiązania.

Poniższy algorytm iteracyjny napisany jest w taki sposób, by można nim było wykonywać zarówno pojedyncze kroki jak i iterować aż do osiągnięcia zadanej dokładności rozwiązania.

$$iter\_krok(B, c, xs) := B \cdot xs + c$$

Podana poniżej funkcja realizuje rozwiązanie układu równań liniowych metodą iteracyjną. Parametr **nmax** oznacza liczbę obiegów algorytmu. Gdy **nmax** < 0, to parametr **ε** oznacza dokładność, która musi zostać osiągnięta przed zwróceniem wyniku. Dla **nmax** >= 0 parametr **ε** jest ignorowany. Do wyniku doklejona jest jedna kolumna, której wszystkie wartości są sobie równe i mówią ile iteracji musiał wykonać algorytm by osiągnąć żądaną dokładność.

$$\text{iter\_rozwiaz}(B, c, xs, \varepsilon, nmax) := \left| \begin{array}{l} xn \leftarrow \text{iter\_krok}(B, c, xs) \\ licznik \leftarrow 0 \\ \text{while } licznik < nmax \vee \left(nmax < 0 \wedge \max\left(\overrightarrow{|xn - xs|}\right) > \varepsilon\right) \\ \quad \left| \begin{array}{l} xs \leftarrow xn \\ xn \leftarrow \text{iter\_krok}(B, c, xs) \\ licznik \leftarrow licznik + 1 \end{array} \right. \\ \text{return } \text{sklej\_macierze\_poziomo}(xn, xn \cdot 0 + licznik) \end{array} \right.$$

W zaimplementowanej tu rodzinie metod iteracyjnych istotnym pojęciem jest promień spektralny; warunkiem zbieżności wszystkich zaimplementowanych tu metod jest, by promień spektralny jakiejś macierzy byłmniejszy od 1. Przydatna okazała się więc funkcja obliczająca promień spektralny, oznaczona symbolem Słoneczka ☼:.

$$☼(arg) := \max\left(\overrightarrow{|\text{eigenvals}(arg)|}\right)$$

Promień spektralny macierzy to największy moduł wartości własnej tej macierzy.

Sprawdzenie funkcji dotyczących promienia spektralnego oraz dokonujących rozkładu macierzy A:

$$☼(A) = 9.193 \qquad \text{eigenvals}(A) = \begin{pmatrix} 9.193 \\ 1.903 + 1.175i \\ 1.903 - 1.175i \end{pmatrix}$$

$$L := \text{wydziel\_L}(A) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$D := \text{wydziel\_D}(A) = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 9 \end{pmatrix} \qquad A = \begin{pmatrix} 2 & 5 & 0 \\ 0 & 2 & 2 \\ 1 & 0 & 9 \end{pmatrix}$$

$$U := \text{wydziel\_U}(A) = \begin{pmatrix} 0 & 5 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{pmatrix} \qquad L + D + U = \begin{pmatrix} 2 & 5 & 0 \\ 0 & 2 & 2 \\ 1 & 0 & 9 \end{pmatrix}$$

Metoda Jacobiego

Równanie iteracyjne metody Jacobiego wygląda następująco:

$\mathbf{Dx^{(i+1)} = -(L + U)x^{(i)} + b}$, a po przekształceniu:

$\mathbf{x^{(i+1)} = -D^{-1}(L + U)x^{(i)} + D^{-1}b}$

Porównując to z wzorem ogólnym rodziny widzimy, że macierz Jacobiego równa jest:

$\mathbf{B_J = -D^{-1}(L + U)}$

Z tego wszystkiego wynika, że:

- Aby równanie dało się rozwiązać metodą Jacobiego, macierz współczynników nie może mieć zerowych elementów na przekątnej (diagonala tej macierzy zapisana jest

w macierzy **D**, która musi dać się odwrócić)

- Metoda Jacobiego jest zbieżna wtedy i tylko wtedy, gdy promień spektralny macierzy Jacobiego jest mniejszy od 1, tj:

  $\varrho(-D^{-1}(L+U)) < 1$

Przed policzeniem rozwiązania nowym algorytmem należy najpierw sprawdzić, czy:

1. Diagonala macierzy **A** nie zawiera elementów zerowych
2. Promień spektralny macierzy Jacobiego jest mniejszy od 1

$L \leftarrow \text{wydziel\_L}(A)$

$D \leftarrow \text{wydziel\_D}(A)$

$U \leftarrow \text{wydziel\_U}(A)$

Sprawdzenie założenia 1:

$$D = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

Diagonala nie zawiera elementów zerowych.

Sprawdzenie założenia 2:

$$B\_j := -D^{-1} \cdot (L + U) = \begin{pmatrix} 0 & -2.5 & 0 \\ 0 & 0 & -1 \\ -0.111 & 0 & 0 \end{pmatrix}$$

$\varrho(B\_j) = 0.652$

Promień spektralny macierzy Jacobiego jest mniejszy od 1.

Rozwiązanie metodą Jacobiego:

$\underset{\wedge\wedge\wedge}{c} := D^{-1} b$

$$res := \text{iter\_rozwiaz}(B\_j, c, c \cdot 0, tolerance, -1) = \begin{pmatrix} 3 & 39 \\ 6 & 39 \\ 9 & 39 \end{pmatrix}$$

$$x := \text{wydziel\_macierz\_lewa}(x, 1) = \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

Widać, że metoda Jacobiego obliczyła wynik z żądaną dokładnością po 39 obiegach.

Metoda Gaussa-Seidla

Metoda ta jest bardzo szczególną wersją metod iteracyjnych, ponieważ do obliczenia **n**-tej iteracji używa wartości z **n-1**-szej i **n**-tej iteracji. Równanie iteracyjne wygląda następująco:

$\mathbf{Dx^{(i+1)} = -Lx^{(i+1)} - Ux^{(i)} + b}$, po przekształceniu:

$\mathbf{(D+L)x^{(i+1)} = -Ux^{(i)} + b}$

$\mathbf{x^{(i+1)} = -(D+L)^{-1}Ux^{(i)} + (D+L)^{-1}b}$

Macierz Gaussa-Seidla: $\mathbf{M_{gs} = -(D + L)^{-1}U}$

Warunkiem poprawności metody jest - podobnie jak w przypadku metody Jacobiego - by macierz **D** (a co za tym idzie **D+L**) była odwracalna. Warunkiem zbieżności jest, by promień spektralny macierzy $\mathbf{M_{gs}}$ był mniejszy od 1.

Podobnie jak poprzednio, należy najpierw sprawdzić, czy:
1. Diagonala macierzy **A** nie zawiera elementów zerowych
2. Promień spektralny macierzy Gaussa-Seidla jest mniejszy od 1

$L \leftarrow \text{wydziel\_L}(A)$

$D \leftarrow \text{wydziel\_D}(A)$

$U \leftarrow \text{wydziel\_U}(A)$

Sprawdzenie założenia 1:

$$D = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

Diagonala nie zawiera elementów zerowych.

Sprawdzenie założenia 2:

$$B\_gs := -(D + L)^{-1} \cdot U = \begin{pmatrix} 0 & -2.5 & 0 \\ 0 & 0 & -1 \\ 0 & 0.278 & 0 \end{pmatrix}$$

$\varrho(B\_gs) = 0.527$

Promień spektralny macierzy Gaussa-Seidla jest mniejszy od 1.

Rozwiązanie metodą Gaussa-Seidla:

$c := (D + L)^{-1} b$

$$res := \text{iter\_rozwiaz}(B\_gs, c, c \cdot 0, tolerance, -1) = \begin{pmatrix} 3 & 27 \\ 6 & 27 \\ 9 & 27 \end{pmatrix}$$

$$x := \text{wydziel\_macierz\_lewa}(x, 1) = \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

Porównanie metod iteracyjnych

Metoda Gaussa-Siedla obliczyła wynik z żądaną dokładnością po 27 obiegach, a więc zdecydowanie szybciej niż metoda Jacobiego. Ciekawym jest temat, w jaki sposób metody te zbiegają do właściwego wyniku. Postanowiłem przedstawić to na wykresie.
Doświadczenie pokazało, że rozsądnie jest wyskalować wykres do 20 na osi X - dalej różnice między metodami oraz między nimi a właściwym wynikiem są znikome.
(z powodu błędów w programie Mathcad nie byłem w stanie lepiej opisać ani sformatować poniższych wykresów)

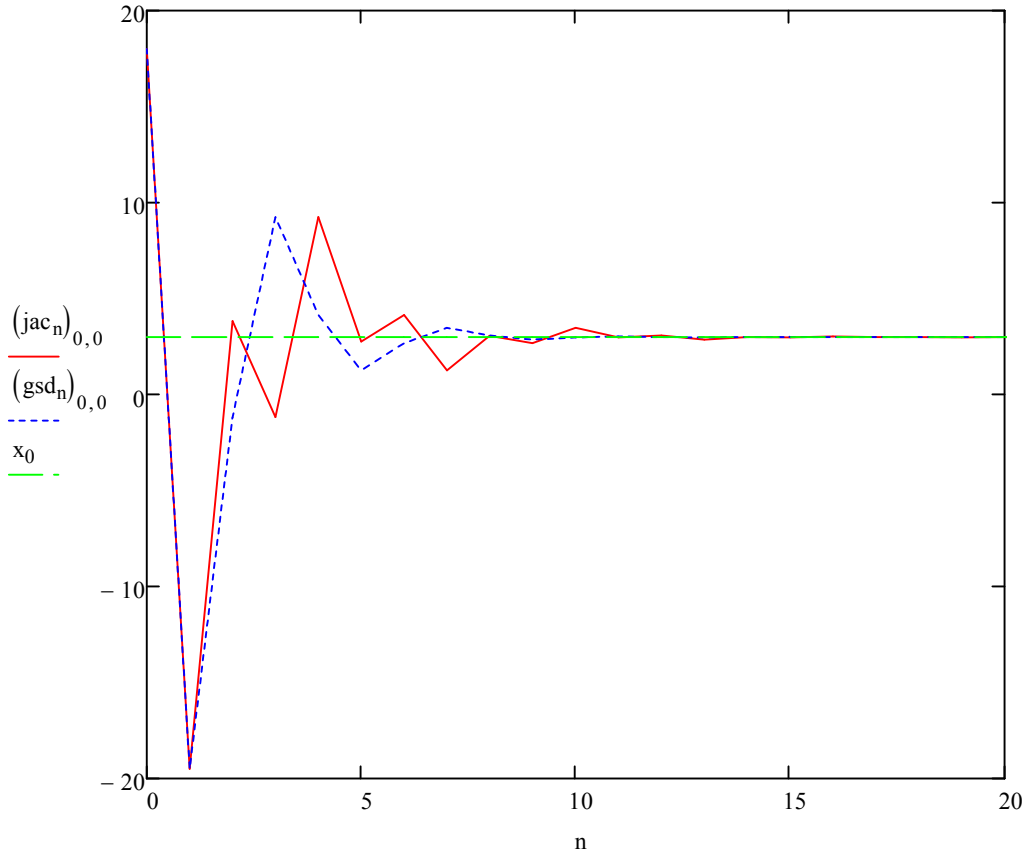
$n := 0..40$

$c\_j := D^{-1} b$ $\qquad$ $c\_g := (D + L)^{-1} b$

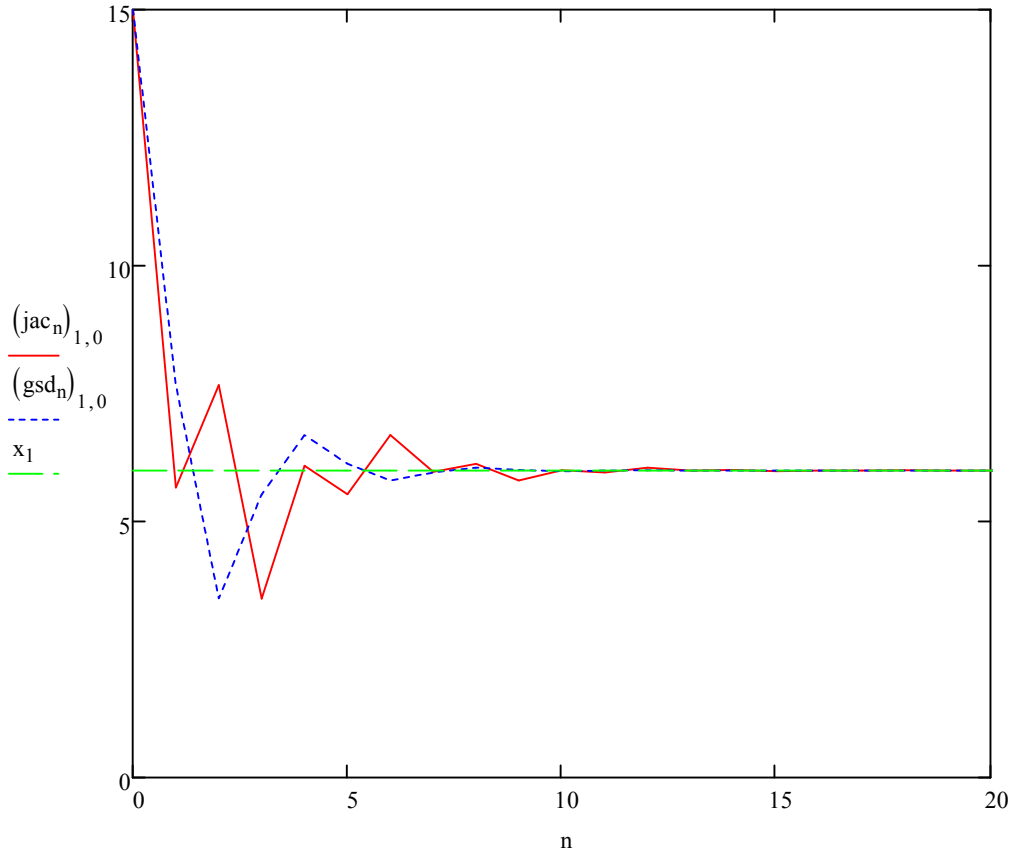
$jac_n := iter\_rozwiaz(B\_j, c\_j, c \cdot 0, tolerance, n) = ...$

$gsd_n := iter\_rozwiaz(B\_gs, c\_g, c \cdot 0, tolerance, n) = ...$

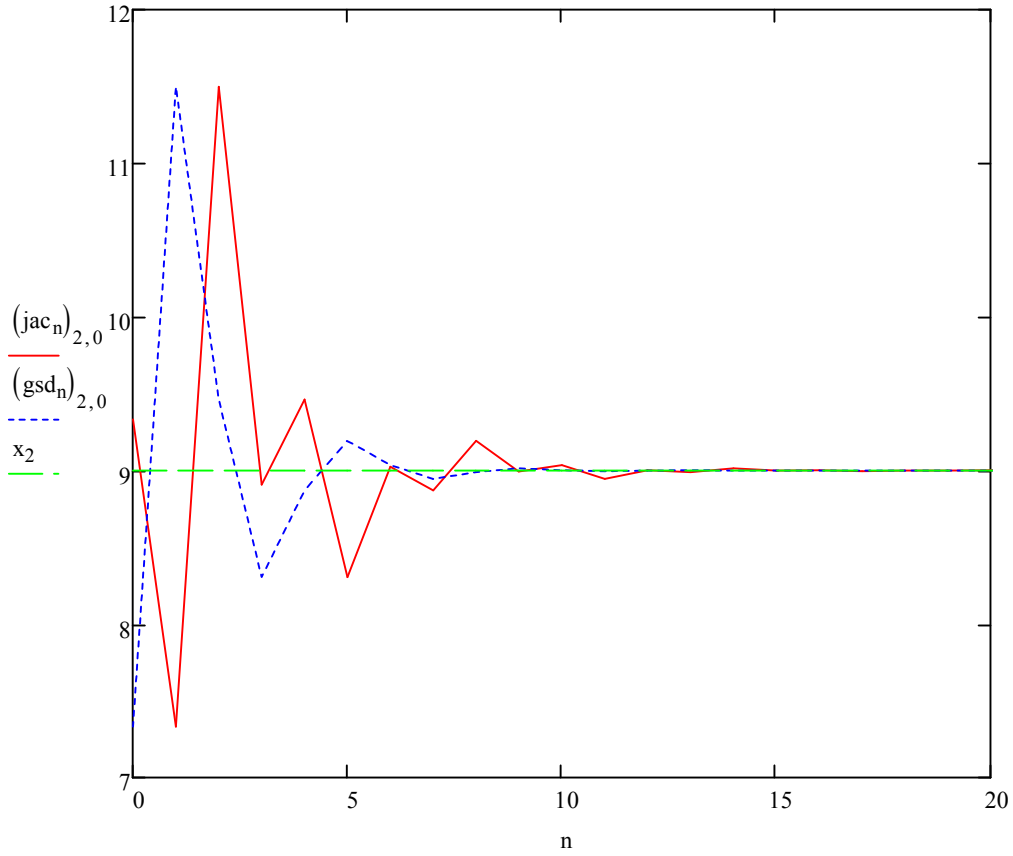
Zbieżność rozwiązania x_1



Zbieżność rozwiązania x_2



Zbieżność rozwiązania x_3



(ja nie żartuję sobie z tymi błędami Mathcada -
http://en.wikipedia.org/wiki/Mathcad#Controversy)