Metody numeryczne w elektrotechnice

Ćwiczenie 11 — Rozwiązywanie zagadnień początkowych dla równań różniczkowych zwyczajnych

Dariusz Borkowski

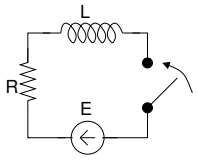
27 maja 2012

Wstęp

Poznasz numeryczne sposoby rozwiązywania równań (i układów równań) różniczkowych zwyczajnych w Simulinku, Matlabie i w C.

1 Najprostszy obwód RL — 2 pkt

Aby zrozumieć sposób w jaki pakiety symulacyjne (jak Simulink) wyznaczają przebiegi sygnałów (zmiennych stanu) układów dynamicznych najpierw rozwiąż na papierze najprostsze zadanie metodą Eulera. Wyznacz przebiegi prądu $i(t)$ w szeregowym obwodzie RL pokazanym na rysunku 1 po zamknięciu wyłącznika.



Rysunek 1: Szeregowy obwód RL.

Obwód ten można opisać równaniem różniczkowym pierwszego rzędu[1]

$$L\frac{di}{dt} + Ri = E \tag{1}$$

co można przekształcić do postaci

$$\frac{di}{dt} = \frac{1}{L}(E - Ri) \tag{2}$$

lub ogólniej[2], przyjmując x za i (prąd i jest funkcją czasu $i(t)$)

$$\frac{dx}{dt} = f(t, x) \tag{3}$$

Podstawiając do (3) iloraz różnicowy $\frac{x_{k+1}-x_k}{h}$ w miejsce pochodnej $\frac{dx}{dt}$ po przekształceniu otrzymujemy iteracyjny wzór Eulera na kolejną wartość x_{k+1}

$$x_{k+1} = x_k + hf(t_k, x_k) \tag{4}$$

gdzie h jest krokiem całkowania, a k jest numerem kroku ($t_k = kh$). Widać, że niezbędna jest znajomość warunku tzw. początkowego czyli $x_0 = x(t_0)$.

W tym przypadku krok całkowania ma wymiar czasu i jest przedziałem czasu, co jaki wyznaczone zostaną kolejne wartości (próbki) prądu. Natomiast kolejne wartości przebiegu prądu opisane są równaniem

$$i_{k+1} = i_k + h\frac{E - Ri_k}{L} \tag{5}$$

[1] Jedno równanie wyższego rzędu (np. obwód RLC opisany jest równaniem rzędu drugiego) przekształca się do postaci układu równań rzędu pierwszego, otrzymując równanie stanu, jak na wcześniejszych zajęciach. Równanie stanu z macierzą stanu jest po prostu układem równań różniczkowych zwyczajnych.

[2] W ogólnym przypadku x może być wektorem złożonym z wielu zmiennych stanu.

Przyjmując wartości $E = 10$ V, $L = 2$ H, $R = 4\ \Omega$ i warunek początkowy $i_0 = i(t = 0) = 0$ A oblicz na papierze wartości prądu w pierwszych 10–ciu krokach dla $h = 0,1$, w pierwszych 5–ciu krokach dla $h = 0,2$. Podaj na końcu wartość prądu po 1 sekundzie od zamknięcia przełącznika. Dodatkowo, korzystając częściowo z obliczonych kroków oblicz wartość prądu po pierwszej sekundzie gdy w pierwszych 6 krokach przyjąłeś $h = 0,1$, a dalej już $h = 0,2$. Porównaj wyniki i uzasadnij różnice. Wyniki zapisuj z dokładnością do 4 miejsc po przecinku, a w obliczeniach powinieneś korzystać z większej. Najłatwiej zrobić to w Matlabie wykorzystując fakt, że za każdym razem w linii poleceń Matlaba wynik umieszczany jest w zmiennej `ans`, którą możesz wykorzystać w iteracyjnych obliczeniach wykonanych ręcznie.

Dowiedziałeś się jak działa najprostsza metoda całkowania równań różniczkowych. Istnieje wiele innych metod, dokładniejszych, często działających wolniej i znacznie bardziej skomplikowanych (patrz wykład). Ważnym spostrzeżeniem powinno być, że kompromis pomiędzy dokładnością obliczeń można uzyskać zmieniając dynamicznie krok całkowania. W miejscach gdzie zmiany zmiennych stanu są duże, należy zmniejszać krok całkowania dla zwiększenia dokładności, a w miejscach o małych zmianach wydłużyć krok całkowania dla przyspieszenia obliczeń bez znacznej starty dokładności.

Jak widzisz rozwiązanie tego zadania wymagało wyprowadzenia równań opisujących obwód. Simulink służy do symulacji obiektów dynamicznych i na podstawie modelu sam tworzy te równania i rozwiązuje. Problemem jest za to zbudowanie z dostępnych bloków modelu[3] analizowanego obiektu, lecz to zagadnienie wykracza poza ramy tego przedmiotu. Tutaj pokazany zostanie najprostszy przykład.

Przenieś ten przykład do Simulinka. Otwórz nowy model. Przenieś na niego następujące bloczki z biblioteki Simulinka: `Sources/Constant`, `Sinks/Scope`, `Continous/Integrator`, `Math/Sum` i dwukrotnie `Math/Gain`. `Constant` będzie odpowiadał napięciu E. Kliknij na nim dwukrotnie i podaj wartość 10. Bloczki `Gain` to wzmocnienie. Jeden z nich odpowiada rezystancji R i ma mieć wartosc 4, a drugi odwrotności indukcyjności L i ma mieć wartość $1/2$. `Integrator` to blok całkujący, dzięki któremu z pochodnej prądu $\frac{di}{dt}$ otrzymasz prąd $i(t)$.

Teraz połacz wejścia i wyjścia bloczków tak, aby odwzorować równanie (2). Zacznij od bloku `Sum` który odzwierciedla sumę w liczniku (2). Ustaw „minus“ na wejściu, na które podasz iloczyn Ri, a „plus“ na wejściu E. Do wyjścia bloku `Sum` podłącz wejście bloku `Gain` odpowiadającemu $1/L$. Na wyjściu tak połączonych bloków jest pochodna prądu $\frac{di}{dt}$. Pochodną scałkuj podając ją na wejście bloku `Integrator`. Na wyjściu otrzymałeś prąd i. Podaj prąd na wejście drugiego bloku `Gain`, aby na jego wyjściu otrzymać iloczyn Ri, który podasz na wolne wejście (z „minusem“) bloku `Sum`. Dołącz na wejście sumatora blok `Constant` odpowiadający napięciu E. Aby zobaczyć na wykresie przebieg prądu i podłącz blok `Scope` do wyjścia bloku `Integrator`.

Otwórz (z menu) okno `Simulation Parameters` z ustawieniami symulacji. Czas zakończenia symulacji ustaw na 2 sekundy. W opcjach solvera wybierz typ algorytmu całkowania `Fixed-step` (stałokrokowy) i algorytm `ode1` (Eulera). Ustaw krok całkowania na 0,2. Kliknij dwukrotnie na bloku `Scope` aby otworzyć okno wykresu. Uruchom symulację. Porównaj wynik z uzyskanym wcześniej na papierze. Otwórz parametry wykresu i w zakładce `Data history` wyłącz ograniczenie pamiętanych danych do 5000 próbek. Teraz, dla zwiększenia dokładności symulacji, ustaw krok całkowania 0,00001 i zobacz jak długo potrwa symulacja. Wybierz algorytm całkowania `ode4` (Rungego–Kutty) i powtórz symulacje. W tym przypadku nie widać znaczącej różnicy w prędkości działania. Zmień rodzaj algorytmu na zmiennokrokowy `Variable-step` i ustaw maksymalną długość kroku na `auto`, a minimalną na 0,00001. Zobacz jak długo trwa symulacja.

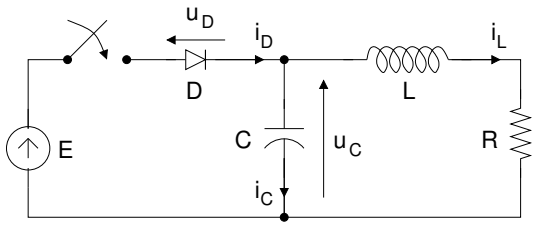
Jak widzisz, bez górnego ograniczenia kroku całkowania symulacja trwa bardzo krótko (w tym przypadku nawet przy zwiększeniu zadanej dokładności do 1e–15). Dlatego wskazane jest używanie algorytmów zmiennokrokowych. W przypadku układów sztywnych, opisanych źle uwarunkowaną macierzą Jacobiego, warto korzystać z algorytmów zmiennokrokowych oznaczonach jako `stiff`.

Teraz zbuduj model szeregowego układu RLC, o dwóch zmiennych stanu (prąd cewki i napięcie kondensatora) i wyznacz przebiegi tych zmiennych w Simulinku. Przyjmij wartości $E = 10$ V, $L = 2$ H, $C = 2$ F, $R = 2\ \Omega$ oraz zerowe warunki początkowe.

2 Wyznaczenie prądu obciążenia prostownika jednopołówkowego — 2 pkt

Twoim zadaniem jest wyznaczenie przebiegu prądu obciążenia i_L od chwili włączenia zasilania prostownika jednopołówkowego pokazanego na rysunku 2 do chwili $t_K = 0,6\ [s]$.

[3] Modele obwodów elektrycznych w Simulinku łatwiej jest budować za pomocą bloków Power Sim Blockset dla Simulinka, choć nie zawsze są one dostępne.



Rysunek 2: Schemat prostownika jednopołówkowego z obciążeniem R.

Prostownik jest zasilany napięciem sinusoidalnym

$$E = A\sin(\omega t), \tag{6}$$

którego pulsacja ω wynosi $2\pi 50$ [rad/s].

Nieliniowa charakterystyka diody napięciowo–prądowa diody jest następująca

$$i_D = i_S\left[\exp\left(\frac{u_D}{mU_T}\right) - 1\right] \tag{7}$$

gdzie $i_S = 100$ nA to prąd nasycenia, a iloczyn w temperaturze pokojowej mU_T wynosi 30 mV.

Korzystając z praw Kirchoffa dla obwodu, a następnie podstawiając zależności (6), (7) oraz zależność na sumę napięć w lewym oczku $u_D = E - u_C$, układ z rysunku można opisać dwoma równaniami różniczkowymi (z których pierwsze jest nieliniowe!):

$$i_S\left[\exp\left(\frac{E-u_C}{mU_T}\right) - 1\right] = i_L + C\frac{du_C}{dt} \tag{8}$$

$$u_C = L\frac{di_L}{dt} + Ri_L \tag{9}$$

Poszukiwane są czasowe przebiegi zmiennych stanu czyli napięcia na kondensatorze u_C i prądu cewki (obciążenia) i_L. Aby je wyznaczyć równania (8) i (9) należy przekształcić tak, aby po lewej stronie znalazły się pochodne zmiennych stanu u_C i i_L (zauważ podobieństwo do równania z poprzedniego zadania) czyli do postaci:

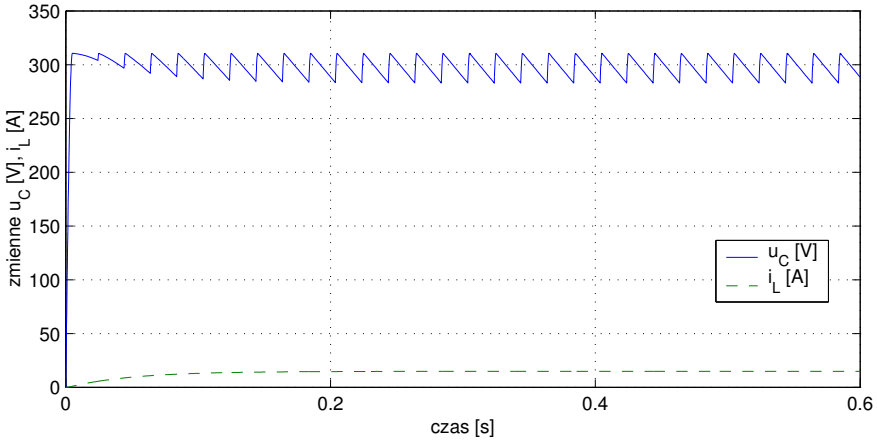
$$\frac{\partial u_C}{\partial t} = f_1(u_C, i_L, t) \tag{10}$$

$$\frac{\partial i_L}{\partial t} = f_2(u_C, i_L, t) \tag{11}$$

Przyjmij wartości elementów $R = 20\ \Omega$, $L = 1$ H, $C = 10$ mF, $A = 220\sqrt{2}$ V. Spróbuj rozwiązać równanie w Matlabie metodą `ode45` dla zerowych warunków początkowych. W tym celu zapisz pod nową nazwą przykładowy program `vdpode` obliczający rozwiązanie równania Van Der Pola. Następnie zmodyfikuj go tak, aby funkcja `f` przekazywana metodzie rozwiązywania, zwracała wektor pochodnych $[\frac{du_C}{dt} \quad \frac{di_L}{dt}]^T$ zmiennych stanu w chwili t. Możesz zrezygnować z użycia macierzy Jacobiego. Pamiętaj, że napięcie E jest także funkcją czasu.

Na początek przyjmij domyślne wartości opcji metody podając w wywołaniu `ode45` pusty wektor `[]` w miejsce opcji. Wyświetl obliczone w kolejnych chwilach czasu wartości zmiennych stanu. Widzisz, że metoda jest niezbieżna. Wypróbuj inne metody, t.j `ode113`, `ode23`. Nadal nie otrzymałeś rozwiązań. Wypróbuj metody dla układów sztywnych `ode23tb`, `ode23t` i `ode15s`. Dopiero ostatnia z metod jest zbieżna, lecz rozwiązanie nie przypomina prawidłowego rozwiązania pokazanego na rysunku 3. Wynika to z domyślnych opcji działania metod całkowania. Za pomocą funkcji `odeset` ustaw następujące opcje: tolerancja względna `RelTol` i bezwzględna `AbsTol` równe 10^{-6}, maksymalna długość kroku całkowania `MaxStep` równa 10^{-3}. Teraz wypróbuj wszystkie metody jeszcze raz. Metody te adresowane są do różnych problemów (patrz dokumentacja programu Matlab) i charakteryzują się m. in. różną prędkością działania. W celu zweryfikowania liczby kroków, obliczeń funkcji i jej pochodnych włącz opcję `Stats`. Funkcji `tic` i `toc` użyj w celu sprawdzenia, która z metod jest w tym zadaniu najszybsza.

Zdefiniuj funkcję wyznaczającą Jakobian (jak w przykładzie `vdpode`) dla funkcji f_1 i f_2 po obu zmiennych stanu i sprawdź jak jego wykorzystanie wpływa na czas obliczeń (użyj opcji `Jacobian`).



Rysunek 3: Przebiegi napięcia u_C i prądu i_L w układzie z rysunku 2 po zamknięciu przełącznika.

Sprawdź dla jakiej wartości indukcyjności L wygładzającej przebieg prądu obciążenia i_L widoczne (przy automatycznej skali wykresu) są tętnienia w prądzie obciążenia.

3 Rozwiązywanie układów równań różniczkowych zwyczajnych w GSL–u — 3 pkt

GSL udostępnia funkcje służące do rozwiązywania układów równań różniczkowych zwyczajnych pierwszego rzędu. Dostępne są zarówno funkcje pozwalające na precyzyjną kontrolę nad przebiegiem całkowania równań (włącznie z modyfikowaniem długości kroku przez programistę podczas całkowania) jak i funkcje wyższego poziomu wykonujące całkowanie automatycznie, praktycznie bez potrzeby kontroli ich działania. Dostępne jest 10 różnych schematów adaptacyjnego wyznaczania długości kroku całkowania.

W zadaniu tym rozwiążesz równanie oscylatora Van der Pola. Jest to równanie różniczkowe drugiego rzędu

$$\frac{d^2x}{dt^2} + \mu\frac{dx}{dt}\left(x(t)^2 - 1\right) + x(t) = 0 \tag{12}$$

które stosując podstawienia $y_1 = x$, $y_2 = \frac{dx}{dt} = \frac{dy_1}{dt}$, $\frac{dy_2}{dt} = \frac{d^2x}{dt^2}$ można zapisać w postaci układu dwóch równań rzędu pierwszego

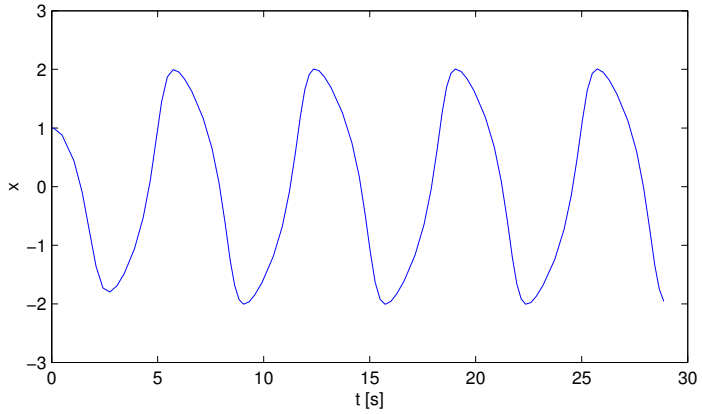
$$\frac{dy_1}{dt} = y_2 \tag{13}$$

$$\frac{dy_2}{dt} = \mu(1 - y_1^2)y_2 - y_1 \tag{14}$$

zależnych od jednego parametru μ. W zależności od wartości μ układ równań może być układem sztywnym (ang. *stiff*) lub nie. Np. dla $\mu = 1000$ jest to układ sztywny.

Otwórz szkielet programu znajdujący się w pliku `lab11z03.c`. Uzupełnij brakujące fragmenty, tak aby program wyznaczał rozwiązanie równania (12) w przedziale czasu od 0 do 100 sekund, przy podanych w pliku warunkach początkowych. Przed rozwiązaniem układu równań należy wybrać typ całkowania oraz przydzielić pamięć dla kilku niezbędnych obiektów, a po opuszczeniu pętli całkowania, należny zwolnic pamięć po obiektach. Należy uzupełnić program o funkcje zapisującą wartości rozwiązania do pliku tekstowego. Poniżej znajduje się opis niezbędnych funkcji. Funkcje opisałem w kolejności, w jakiej mają wystąpić w programie. Przed programem głównym `main` należy zdefiniować funkcję (wektor funkcji) opisujących pochodne zmiennych stanu oraz, jeśli użyty algorytm tego wymaga, również Jakobian układu równań różniczkowych.

`gsl_odeiv2_step_type` jest typem danych określającym typ metody całkowania (sposób wyznaczania długości kroku całkowania). Typ definiuje się nastepująco: `const gsl_odeiv2_step_type * T = gsl_odeiv2_step_rk4;`. Określenie typu `T` powinno się znajdować, przed opisanymi niżej funkcjami. W dokumentacji GSL–a pod adresem `http://www.gnu.org/software/gsl/manual/gsl-ref_25.html` są opisane pozostałe typy metod całkowania.



Rysunek 4: Przebiegi rozwiązania równania (12) dla $\mu = 1$.

`gsl_odeiv2_step * gsl_odeiv2_step_alloc (const gsl_odeiv2_step_type * T, size_t dim)` zwraca wskaźnik do nowego wystąpienia funkcji „kroczącej“ (ang. *stepping functions*) typu `T` dla systemu o `dim` wymiarach.

`gsl_odeiv2_control * gsl_odeiv2_control_y_new (double eps_abs, double eps_rel)` tworzy nowy obiekt kontrolny, który jest odpowiedzialny za utrzymanie lokalnego błędu całkowania w granicach okreslonych przez tolerancję względną `eps_rel` i bezwzględną `eps_abs`. Dostępne są również inne sposoby kontroli lokalnego błędu.

`gsl_odeiv2_evolve * gsl_odeiv_evolve2_alloc (size_t dim)` zwraca wskaźnik do nowo utworzonej funkcji ewolucyjnej (autorzy GSL–a postęp całkowania układu równań różniczkowych nazywają ewolucją) rozwiązujące układ o `dim` wymiarach. Funkcja ewolucyjna jest interfejsem wysokiego poziomu do rozwiązywania układu równań różniczkowych zwyczajnych. Funkcja ta automatycznie wywołuje funkcje kroczącą oraz funkcje kontrolującą błąd i jeśli jest taka potrzeba wykonuje krok wstecz i wyznacza na nowo rozwiązania.

`gsl_odeiv2_system` jest typem danych opisującym rozwiązywany układ równań różniczkowych (system). System można zdefiniować np. tak `gsl_odeiv2_system sys = {f,j,n,params}`, gdzie:
`f` jest wskaźnikiem do funkcji zwracającej wartość pochodnych zmiennych stanu
`int (* function) (double t, const double y[], double dydt[], void * params)` ;
`j` jest wskaźnikiem do macierzy Jacobiego pochodnych cząstkowych funkcji `f`
`int (* jacobian) (double t, const double y[], double * dfdy, double dfdt[], void * params)` należy dodać, że nie każdy algorytm wykorzystuje macierz Jacobiego (jedynie te najbardziej zaawansowane), ale wygodnie jest zdefiniować Jakobian, aby móc przetestować każdy rodzaj algorytmu. W przypadku niedefiniowania Jakobianu należy zamiast `j` podać `NULL`; `n` jest rozmiarem systemu (ilością równań) typu `size_t`;
`params` jest wskaźnikiem typu `void` do struktury zawierającej parametry funkcji `f`.

`int gsl_odeiv2_evolve_apply (gsl_odeiv2_evolve * e, gsl_odeiv2_control * c, gsl_odeiv2_step * s, const gsl_odeiv2_system * dydt, double * t, double t1, double * h, double y[])` jest wywołaniem funkcji ewolucyjnej, która wyznacza kolejne rozwiązania systemu. Przy wywołaniu należy podać wskaźniki do allokowanych wcześniej obiektów `s, c, e`. `dydt` jest wskaźnikiem do obiektu opisującego system. `t` jest wskaźnikiem do aktualnego czasu w symulacji, a `h` do aktualnej długości kroku. `y[]` jest tablicą (czyli właściwie wskaźnikiem na jej zerowy element) zawierającą aktualną wartość (wektor) rozwiązań. Na początku symulacji za `y` podajemy warunek początkowy, a za `t` początkowy czas symulacji. Po każdym wywołaniu rozwiązanie w aktualnym kroku zapisywane jest do `y`, a czas w aktualnym kroku do `t`. Również wartość kroku całkowania `h` jest uaktualniana w każdym kroku `t1` jest czasem końca symulacji. Funkcja ta powinna działać w pętli, sprawdzającej czy czas `t` nie przekroczył czasu `t1`. Funkcja zwraca wartość statusu. Jeśli funkcja zwróci `GSL_SUCCESS` to można przerwać wykonanie pętli.

`void gsl_odeiv2_step_free (gsl_odeiv_step * s)` zwalnia pamięć użytą przez funkcję kroczącą.

`void gsl_odeiv2_control_free (gsl_odeiv_control * c)` zwalnia pamięć po obiekcie kontrolnym.

`void gsl_odeiv2_evolve_free (gsl_odeiv_evolve * e)` zwalnia pamięć po funkcji ewolucyjnej.

Sprawdź działanie różnych algorytmów (metod całkowania) opisanych w dokumentacji. Zadaj $\mu = 1000$, aby otrzymać układ sztywny (dla różnych μ otrzymasz różny okres oscylacji rozwiązania!). Zauważ, że teraz symulacja obejmująca ten sam czas co poprzednio (100 sekund) trwa o wiele dłużej (dla przyspieszenia działania możesz wyłączyć wypisywanie wyników na ekran) i tworzy znacznie dłuższy wektor wyników. Aby uniknąć tworzenia tak długiego wektora wyników, nie należy zapisywać do pliku wartości po każdym kroku całkowania, lecz co zadany przedział czasu np. 1 sekundę. Korzystając z przykładu podanego w dokumentacji GSL–a zmodyfikuj program tak, aby zapisywać rozwiązania w wybranych chwilach, z jednakowym odstępem czasu. Wyrysuj w Matlabie rozwiązania dla $\mu = 1$ i $\mu = 100$.

Na następne zajęcia

Równania różniczkowe cząstkowe, metody różnicowe i metoda elementów skończonych.

Literatura

[1] Guziak T., Kamińska A., Pańczyk B., Sikora J. *Metody numeryczne w elektrotechnice* Wydawnictwo Politechniki Lubelskiej, Lublin 2002.

[2] Edward Kącki, Andrzej Małolepszy, Alicja Romanowicz *Metody numeryczne dla inżynierów* Wydawnictwo Politechniki Łódzkiej, Łódź 1997.