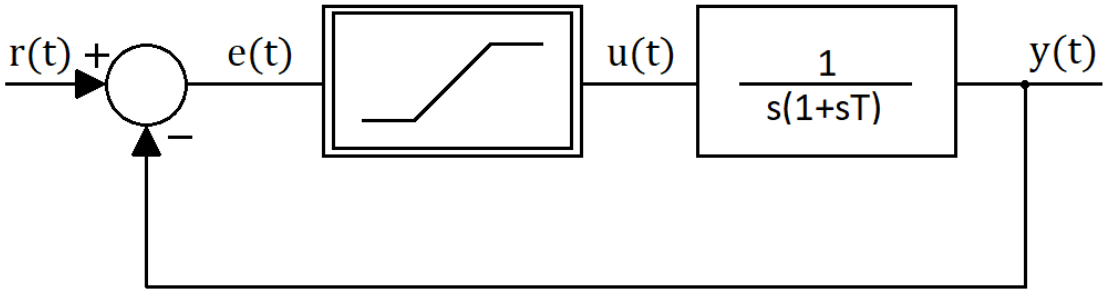
Sprawozdanie

Projekt Metody Modelowania Matematycznego

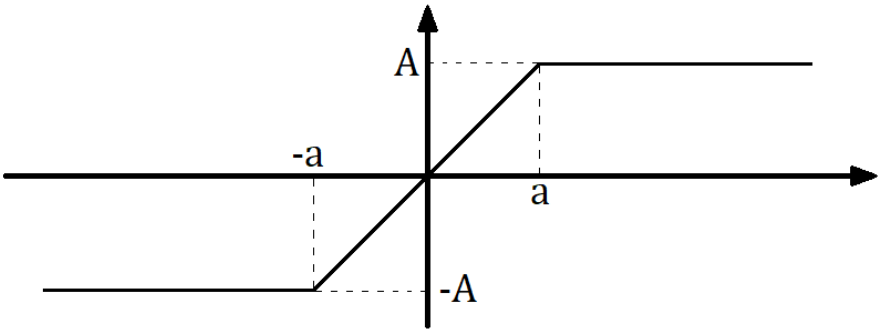
Temat 13

Polecenie

Celem projektu było zamodelowanie i przeprowadzenie symulacji działania systemu o następującym schemacie blokowym:



Nieliniowość ze schematu blokowego jest zależna od dwóch parametrów, **a** oraz **A** i przedstawia się następująco:



W menu programu użytkowanik powinien mieć możliwość definiowania parametrów ***a***, ***A*** oraz ***T***, a sam program powinien wyświetlać bieżące wartości wyjścia ***y(t)*** i uchybu ***e(t)***. Ponadto wymagane jest, aby umożliwić wprowadzenie do systemu trzech rodzajów pobudzeń: fali prostokątnej, fali trójkątnej oraz sinusoidy.

Realizacja

Wstępna analiza

Realizację zadania zaczęliśmy od teoretycznej analizy systemu. Kluczowe było dobre opisanie członu liniowego z powodu złożoności jego działania. Zdecydowaliśmy się na opis metodą równań stanu, ponieważ dawała ona najbardziej klarowne rozwiązanie. Efekty matamatycznych rozważań znajdują się poniżej:

$$\begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{1}{T} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot u$$

$$y = \begin{bmatrix} \frac{1}{T} & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Taką postać przyjęły równania stanu członu liniowego naszego układu. Na ich podstawie wyznaczyliśmy 3 kolejne równania:

$$x'_1 = x_2$$

$$x'_2 = -\frac{1}{T} \cdot x_2 + u$$

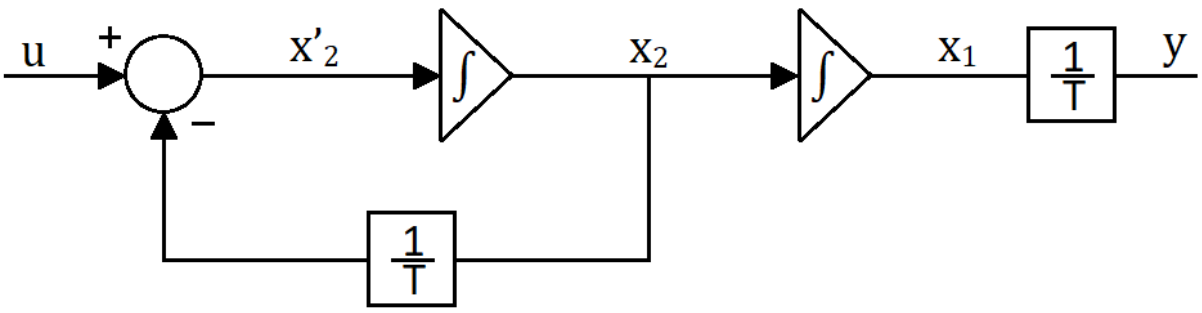
$$y = \frac{1}{T} \cdot x_1$$

Następnym krokiem było scałkowanie dwóch pierwszych równań w celu uzyskania pierwotnych wartości pochodnych x'_1 oraz x'_2:

$$x_1 = \int x_2 \mathrm{d}t$$

$$x_2 = \int \left(-\frac{1}{T} \cdot x_2 + u \right) \mathrm{d}t$$

Po opisaniu zmiennych skonstruowaliśmy schemat, który uporządkował wszystkie zależności i znacznie ułatwił późniejsze pisanie programu:

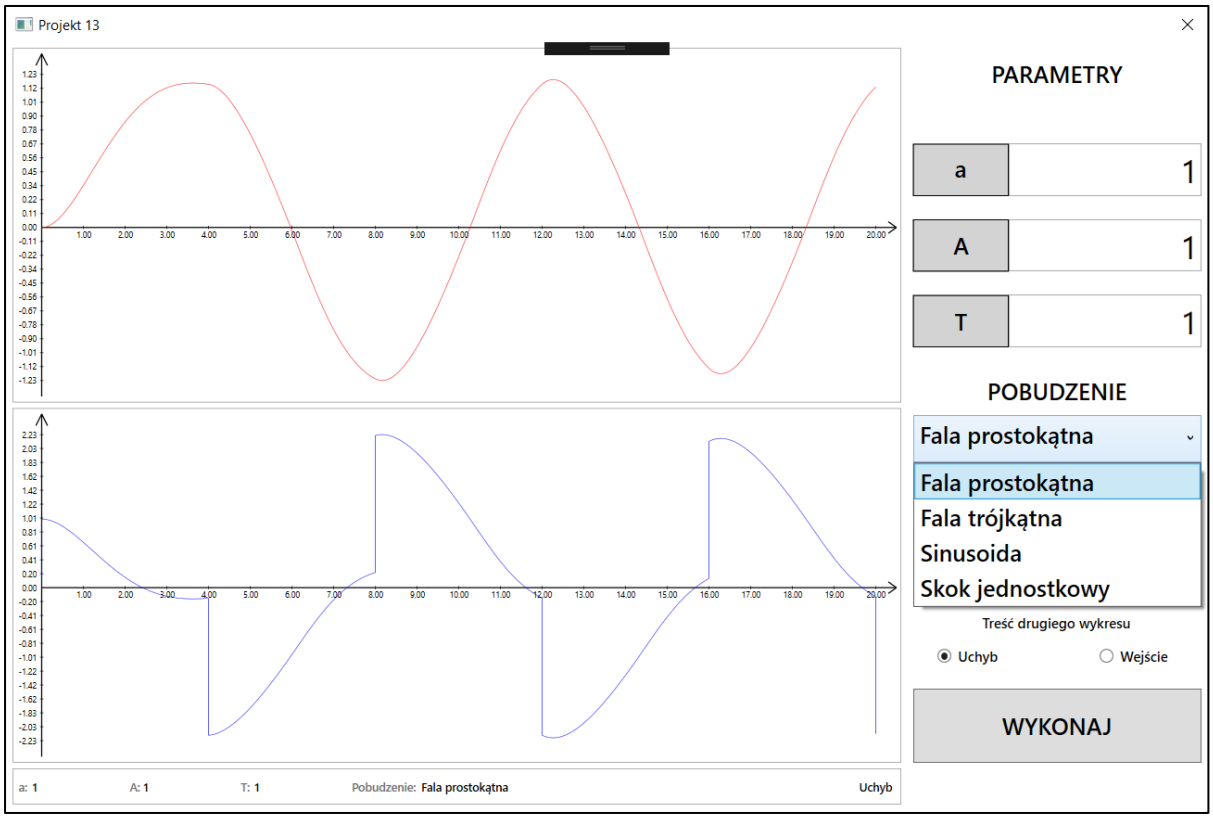


Na jego podstawie wnioskujemy, że aby otrzymać wyjście w danej chwili czasu będziemy musili dwukrotnie przeprowadzić operację całkowania. Ponadto widzimy pętlę sprzężenia zwrotnego, która ozancza, że wartość w danej chwili czasu będzie zależeć od stanu x_2 z poprzedniej chwili czasu.

Kod programu

Program postanowiliśmy napisać na platformie Windows Presentation Foundation, a więc w języku C#. Taki wybór był podyktowany łatwości prezentacji wyników i tworzenia GUI na tej platformie oraz wygodną pisanie w C#.

Na początku zajęliśmy się tworzeniem okienka programu. Na tym etapie skupiliśmy się na tym, aby zawrzeć w nim wszystkie funkcjonalności narzucone przez polecenie oraz na tym, żeby GUI było intuicyjne dla użytkowanika. Finalnie okno prezentuje się następująco:



W prawym górnym rogu znajdują się trzy pola tekstowe służące do definiowania parametrów układu. Poniżej nich umieściliśmy rozwijany pasek, który umożliwia wybranie odpowiedniego pobudzenia układu. Poza trzema podstawowymi pobudzeniami, o których była mowa w poleceniu dodaliśmy jeszcze opcję pobudzenia układu skokiem jednostkowym. W centrum okienka znajdują się dwa pola, na których rysowane są odpowiednie wykresy. Górny wykreśla odpowiedź układu, a dolny uchyb lub sygnał wejściowy w zależności od preferencji użytkownika (przyciski opcji pod rozwijanym paskiem). Oś odciętych obu wykresów prezentuje upływający czas, a rzędnych wartość sygnału. W obydwu przypadkach wyniki podawane są z dokładnością do dwóch miejsc po przecinku. Za każdy razem kiedy zlecimy programowi wykonanie operacji (przycisk WYKONAJ), wszystkie osie skalowane są na nowo w celu zapewnienia użytkownikowi możliwie jak najbardziej satysfakcjonujących doznań wizualnych. Pod wykresami znajduje się pasek informujący o tym, dla jakich parametrów została wykonana operacja.

Po stworzeniu okienka przystąpiliśmy do projektowania logiki programu. Najlepszym sposobem na skrótowe opisanie zasady działania programu będzie wypunktowanie kolejnych kroków, które są wykonywane przez program.

Zatem po wciśnięciu przycisku WYKONAJ:

1. Program zczytuje wartości zadanych parametrów, a także rodzaj pobudzenia i umieszcza te wartości w odpowidnich zmiennych.
2. Wątek jest wysyłany do „głównej" metody 'Operacje', która zarządza wszystkim tym, co dzieje się w trakcie wykonywanej przez program pracy.
3. Tam najpierw deklarowane są wszystkie niezbędne zmienne, a następnie rozpoczyna się praca pętli for, która imituje działanie zegara taktującego układ.
4. Każdy obrót pętli powoduje odwołanie się do czterech różnych metod, które wykonują następujące operacje:
    - Obliczenie aktualnej wartości sygnału podawanego na wejście układu - jako argument zadawany jest czas w jakim znajduje się układ w danym momencie.
    - Obliczanie aktualnego uchybu – wykorzystujemy wcześniej obliczoną, aktualną wartość na wejściu oraz wartość poprzedniego wyjścia układu; przy pierwszej pętli wynosi ona zero.
    - Obliczanie wartości sygnału po przejściu przez człon nieliniowy – w obliczeniach korzystamy z uchybu oraz zadeklarowanych przez użytkownika parametrów a i A.
    - Obliczenie ostatecznego wyjścia układu – w tej metodzie wszystkie kolejne kroki są wykonywane zgodnie z opracowanym schematem członu liniowego zamieszczonym wcześniej. Odbywa się tam dwukrotne całkowanie, o które dokładniej opiszemy w dalszej części.

    Na końcu każdej iteracji informacje na temat aktualnego uchybu i aktualnego wyjścia są umieszczane w dwóch osobnych listach, na podstawie których program wykonuje wykresy w następnym kroku.

5. Po wyjściu z pętli następuje dwukrotne odesłanie do metody 'Rysuj_wykres', która zgodnie z przypuszczeniami i intuicją czytelnika odpowiada ze rysowanie wykresów w głównym oknie programu. Odbywa się tam również skalowanie i odpowiednie oznaczanie osi wykresów.

Podczas pisania programu staraliśmy się zachować pewną przejrzystość kodu oraz odrębność w kwestii wykonywanych przez program kolejnych operacji. Strona logiczna oraz wizualna są od siebie odzielone, co w naszym odczuciu jest dość istotne ze względu na chaos jaki może wprowadzić połączenie i przeplatanie tych dwóch aspektów ze sobą np. w jednej metodzie.

Całkowanie numeryczne

Do zrealizowania polecenia projektu konieczne było zastosowanie całkowania numerycznego. Spośród trzech najpopularniejszych sposobów (metoda prostokątów, trapezów oraz parabol) wybraliśmy metodę trapezów. Wybór podyktowany był tym, że metoda ta stanowi pewien kompromis dwóch pozostałych. Jest tylko trochę bardziej skomplikowana niż metoda prostokątów i jednocześnie bliska dokładności metody parabol.

Istotą całkowanie numerycznego jest obliczenie pola powierzchni pod wykresem zadanej funkcji na ograniczonym zakresie (odpowiednik całki oznaczonej). Metoda trapezów polega na podzieleniu tego obszaru na wiele trapezów i policzeniu sumy pól wszystkich z nich; w ten sposób uzyskujemy

ostateczny wynik. W teorii, im więcej figur utworzymy, tym dokładniejsza będzie wartość całki. Aby obliczyć pole pojedynczego trapezu potrzebujemy aktualnej oraz poprzedniej wartości funkcji, ponieważ stanowią one długości podstaw trapezu. Jako wysokość przyjmujemy tak zwany krok całki, czyli odległość pomiędzy kolejnymi wielkościami na osi odciętych, dla któych liczymy wartość funkcji. Z takim kompletem danych, obliczenie pola figury jest już formalnością. Realizację tych wszystkich kroków odnajdziemy w metodach 'Oblicz_wyjscie' oraz 'Calkowanie'. W celu ułatwienia implementacji całkowania stworzyliśmy również prostą, aczkolwiek bardzo pomocną klasę 'Calka', której obiekty przechowują informację o wartości poprzedniej oraz aktualnej próbki, całkowitej bieżącej wartości danej całki oraz statyczną zmienną określającą wielkość kroku. Według nas takie rozwiązanie poprawiło czytelność kodu oraz zapobiegło chaosowi wynikającego z natłoku zbędnych zmiennych.

Wnioski i uwagi

Naszym zdaniem realizacja tego projektu była bardzo ciekawym doświadczeniem pod wieloma względami. Przede wszystkim pozwoliła nam lepiej zrozumieć w jaki sposób działają układy automatyki oraz z jakimi trudnościami możemy się spotkać przy pracy z nimi. Wykonując zadanie wykorzystaliśmy wiedzę zdobytą podczas realizowania Podstaw Automatyki oraz Metod Modelowania Matematycznego i połączyliśmy ją z aspektem praktycznym. Uważamy, że ostateczny rezultat naszych starań jest zadowalający. Wszystkie założenia udało się zrealizować bez większych przeszkód.

Poniżej przedstawimy kilka wniosków, które udało nam się wysnuć podczas pracy nad projektem:

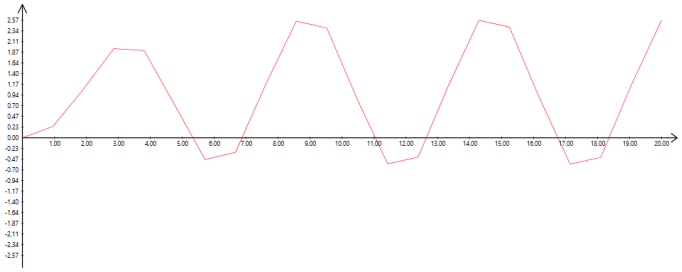
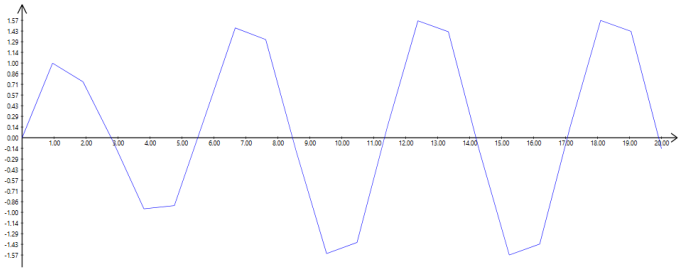
Odnośnie układu automatyki

- Wprowadzenie do układu członu z nieliniowością pozwala na stosunkowo proste ingerowanie w jakość procesów przejściowych układu. Podanie odpowiedniej kombinacji wartości parametrów umożliwia uzyskanie pożądanych właściwości systemu.
- Dla bardzo małych wartości parametru a, nieliniowość typu nasycyenie (w przybliżeniu) staje się nieliniowością typu przekaźnik dwupozycyjny. Wtedy:
    - Przy zwiększaniu wartości parametru A odpowiedź układu jest coraz szybsza, a przeregulowanie coraz większe. Efekt jest odwrotny w przypadku zmniejszania tej wartości.
    - Zwiększanie parametru T powoduje pogorszenie parametrów układu. Działa on wolniej i z większym przeregulowaniem.
- Przy $a >> A$ układ działa bardzo wolno, przy praktycznie zerowym przeregulowaniu
- Kiedy przyjmiemy, że $a = A$, to jesteśmy w stanie otrzymać układ czysto liniowy. Na przykład, jeżeli amplituda pobudzeń jest równa 1, to dla wartości parametrów powyżej 2 nieliniowości nie ma, układ zachowuje się tak samo, niezależnie jak bardzo zwiększamy tę wartość. Dla nierówności $0 < a = A < 2$ i przy podawaniu fali prostokątnej na wejście, zmniejszanie wartości parametrów sprawia, że układ działa nieco wolniej przy nieco mniejszym przeregulowaniu. Dla skoku jednostkowego analogiczne działanie obserwujemy dla nierówności $0 < a = A < 1$.
- Badanie układu przy jednoczesnej dużej wartości a i małej A nie ma większego sensu, ponieważ układ jest ekstremalnie wolny.

Odnośnie całkowania numerycznego

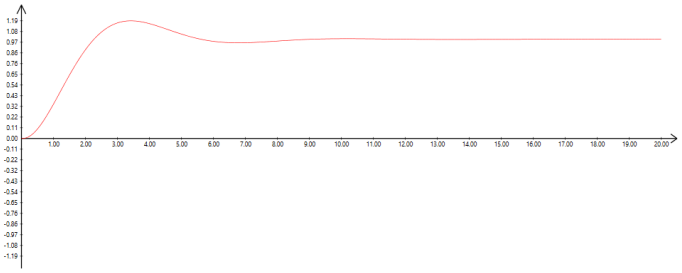
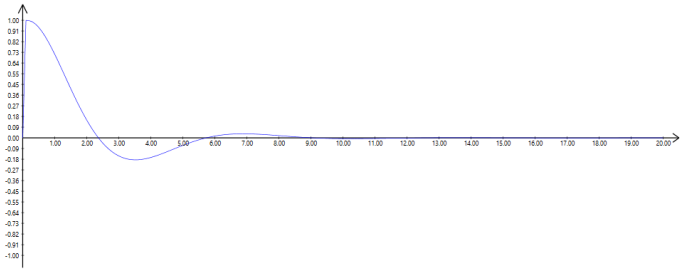
Wielkość kroku całki ma wpływ na dokładność uzyskanych wyników. Im większy krok, tym mniej dokładny rezultat otrzymamy. Porównajmy pięć przypadków zakładając, że porównanie wykonujemy dla skoku jednostkowego przy $a = A = T = 1$:

- Krok całki równy **1**:

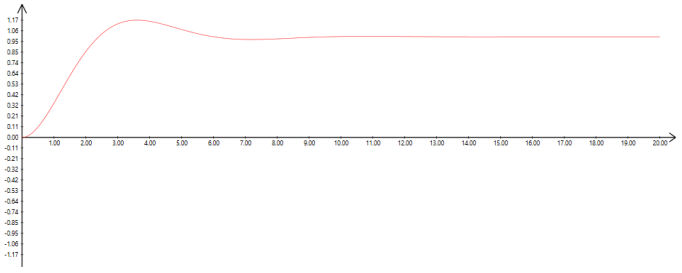
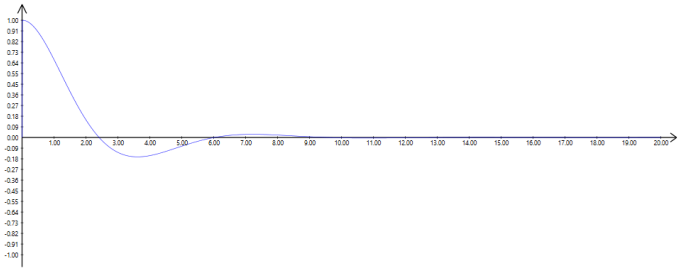
Uzyskane wykresy zdecydowanie odbiegają od tego, czego powinniśmy się spodziewać. Są one błędne i nie można z nich odczytać żadnych przydatnych informacji. Wykres został wykonany na podstawie jedynie 20 próbek.

- Krok całki równy **0,1**:

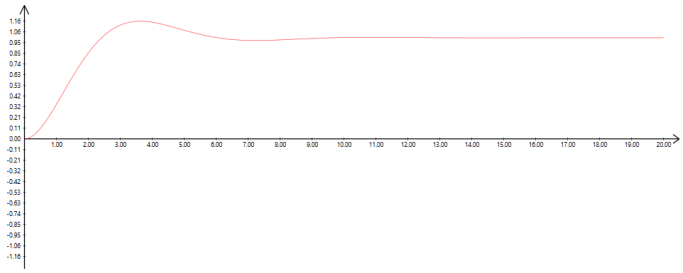
Widzimy znaczną poprawę w jakości wykresów. Tym razem przypominają one kształtem spodziewany wykres, jednak nie są doskonałe. Na wykresie uchybu widać skok zaraz przy osi rzędnych, który nie powinien się tam znaleźć. Liczba próbek wzrosła do 200. Aktualne, wskazywane przez program przeregulowanie wynosi 19%.

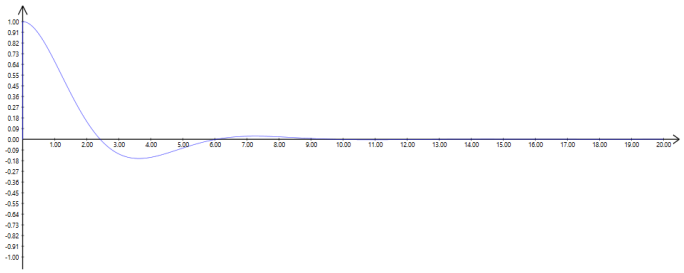
- Krok całki równy **0,01**:

Teraz wykres jest wykonany na podstawie 2000 próbek. Zmiany względem ostatniego punktu są niewielkie. Zniknął niepożądany skok na wykresie uchybu, a przeregulowanie wynosło tym razem 17%. Wyniki zyskały na dokładności.

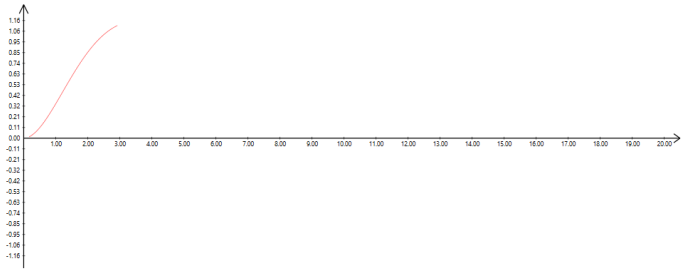
- Krok całki równy 0,001:

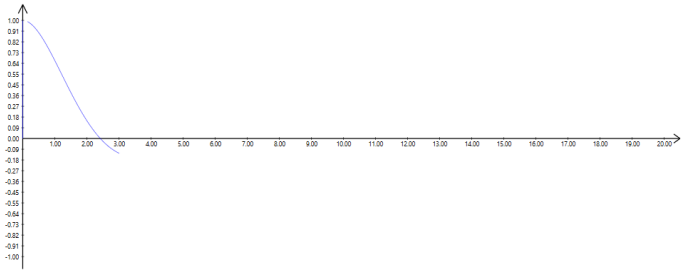




Przy kolejnym, dziesięciokrotnym wzroście liczby próbek, jedyną zmianą względem poprzedniego punktu jest redukcja wartości przeregulowania do 16%. Wynik jest coraz bardziej precyzyjny.

- Krok całki równy 0,0001:





Uzyskany rezultat jest niespodziewany. Widzimy, że wykres jest niekompletny, rysowanie zakończyło się po mniej więcej trzech sekundach. Liczba 200 000 próbek okazała się zbyt wymagająca dla komputera. Zawiódł prawdopodobnie sam mechanizm rysowania funkcji. Platforma WPF nie jest dobrze przystosowana do tworzenia wykresów na podstawie aż tylu punktów, stąd widoczne usterki. Widzimy natomiast, że przeregulowanie nie uległo zmianie i nadal wynosi 16%. Możemy zatem wnioskować, że wynik uzyskany w poprzednim punkcie jest dostatecznie dokładny.

Widzimy zatem, że najdokładnieszy wynik jaki potrafimy uzyskać otrzymamy stosując krok 0,001. Jest to powód, dla którego właśnie tę wartość ustawiliśmy jako domyślną w naszym programie. Całkiem dobry rezultat dał krok 0,01 i zastosowanie tej wartości nie byłoby dużym błędem, jednak naszym zdaniem jest to wielkość graniczna, nie powinniśmy stosować większego kroku jeżeli chcemy zachować wiarygodność uzyskiwanych rezultatów.

Odnośnie procesu pisania programu

- W kwestii pobudzeń układu największą trudność sprawiło zaprogramowanie sygnału trójkątnego. Naszym pomysłem na rozwiązanie tego problemu było podzielenie jednego okresu sygnału na cztery części i zapisanie wzoru na każdą z nich. Pomocne okazało się zastosowanie działania modulo.
- Wartości wyjścia i uchybu, które otrzymujemy przy każdej iteracji pętli mogłyby być rysowane w postaci wykresu zaraz po ich uzyskaniu. To jednak rodziłoby problem, ponieważ w takim

wypadku skalowanie byłoby bardzo utrudnione. Aby temu zapobiec, otrzymane wyniki zapisujemy do listy, dzięki czemu proces rysowania może odbyć się dopiero po zgromadzeniu wszystkich danych. Mając całe spektrum wartości funkcji możemy przeszukać utworzoną listę w celu ustalenia największej bezwzględnej wielkości, która będzie stanowiła poszukiwaną skalę dla reszty wyników.