

Składam serdeczne podziękowania
dr. inż. Pawłowi Sytemu za
poświęcony czas i pomoc w
przygotowaniu niniejszej pracy

Spis treści

CEL PRACY	9
WSTĘP	11
1 PLATFORMA .NET I JĘZYK C#	13
1.1 HISTORIA FRAMEWORKA .NET	13
1.2 HISTORIA JĘZYKA C#	14
1.3 ZASADA DZIAŁANIA	15
1.4 STOS PLATFORMY.....	16
1.5 WINFORMS CZY WPF	18
2 PROJEKT I IMPLEMENTACJA APLIKACJI.....	21
2.1 PAKIET EVALUATION – UZYSKIWANIE DEFINIOWANEJ PRZEZ UŻYTKOWNIKA FUNKCJI W TRAKCIE DZIAŁANIA PROGRAMU	23
2.1.1 <i>Parsowanie wyrażeń matematycznych</i>	23
2.1.2 <i>Dynamiczna kompilacja funkcji</i>	23
2.1.3 <i>Klasa StringParser – część odpowiedzialna za przetwarzanie i normalizację wyrażenia</i>	24
2.1.4 <i>Klasa Evaluator – dynamiczna kompilacja funkcji i uzyskiwanie delegata na funkcję</i>	26
2.2 PAKIETY FUNCTIONS I CONSTANTS – IMPLEMENTACJA FUNKCJI I STAŁYCH.....	29
2.2.1 <i>Stałe matematyczne – klasa MathematicalConstants</i>	29
2.2.2 <i>Stałe fizyczne – klasa PhysicalConstants</i>	30
2.2.3 <i>Funkcje elementarne – klasa ElementaryFunctions</i>	31
2.2.4 <i>Funkcje specjalne – klasa SpecialFunctions</i>	33
2.3 OBLCZENIA NUMERYCZNE – PAKIET NUMERICALCALCULATIONS	37
2.3.1 <i>Calkowanie numeryczne – klasa Integral</i>	37
2.3.2 <i>Różniczkowanie numeryczne – klasa Derivative</i>	41
2.3.3 <i>Szukanie pierwiastków funkcji – klasa FunctionRoot</i>	44
2.4 WYKRESY FUNKCJI ZMIENNEJ RZECZYWISTEJ – KLASA CHART2D	49
2.4.1 <i>Błędy w klasie Chart</i>	51
2.4.2 <i>Inicjalizacja klasy Chart2D</i>	54
2.4.3 <i>Reprezentacja funkcji matematycznych w klasie Chart2D</i>	55
2.4.4 <i>Rysowanie wykresów</i>	57
2.4.5 <i>Zmiana obszaru wykresu</i>	58
2.4.6 <i>Obsługa zdarzeń w klasie Chart2D i autoskalowanie</i>	60
2.4.7 <i>„Instalowanie” klasy Chart w innych kontrolkach</i>	62
2.5 WYKRESY FUNKCJI ZMIENNEJ ZESPOLONEJ – KLASA COMPLEXCHART	64
2.5.1 <i>Problem niewystarczającej liczby wymiarów</i>	67
2.5.2 <i>Metoda kolorowanie domeny</i>	67
2.5.3 <i>Reprezentacja funkcji matematycznych w klasie ComplexChart</i>	70

2.5.4	<i>Własności klasy ComplexChart</i>	71
2.5.5	<i>Obliczenia wartości funkcji i kolorów.....</i>	72
2.5.6	<i>Rysowanie wykresu</i>	76
2.5.7	<i>Obsługa zdarzeń w klasie ComplexChart</i>	78
3	EFEKTY I TESTY.....	79
3.1	<i>INTERFEJS APLIKACJI.....</i>	79
3.2	<i>EXPRESSIONTEXTBOX</i>	81
3.2.1	<i>Estetyczność wpisywanych formuł</i>	81
3.2.2	<i>Podpowiedzi do wpisywanych formuł</i>	83
3.3	<i>KALKULATOR</i>	85
3.4	<i>WYKRESY FUNKCJI RZECZYWISTYCH</i>	87
3.5	<i>WYKRESY FUNKCJI ZESPOLONYCH.....</i>	92
3.6	<i>OBLCZENIA NUMERYCZNE</i>	95
	PODSUMOWANIE I WNIOSKI.....	98
	LITERATURA.....	101
	DODATEK A. SPIS ZAWARTOŚCI DOŁĄCZONEJ PŁYTY CD	104
	DODATEK B. BŁĄD NR 1 ZNALEZIONY W KLASIE DATAVISUALIZATION.CHART	105
	DODATEK C. BŁĄD NR 2 ZNALEZIONY W KLASIE DATAVISUALIZATION.CHART	106
	DODATEK D. BŁĄD NR 3 ZNALEZIONY W KLASIE DATAVISUALIZATION.CHART	107

Cel pracy

Celem niniejszej pracy jest opracowanie aplikacji desktopowej, wspomagającej obliczenia naukowe i inżynierskie z uwzględnieniem zbioru liczb zespolonych. Aplikacja taka powinna umożliwiać obliczanie dowolnie złożonych wyrażeń algebraicznych z uwzględnieniem:

- podstawowych operatorów arytmetycznych (+, *, /)
- liczb rzeczywistych i liczb zespolonych w dowolnej reprezentacji
- wszystkich popularnych funkcji elementarnych zmiennej rzeczywistej jak i zespolonej
- popularnych i często używanych w nauce i inżynierii funkcji specjalnych zmiennej rzeczywistej i zespolonej

Aby umożliwić inżynierom i naukowcom łatwiejszą pracę zdefiniowano niezbędne minimum funkcji, jakie aplikacja powinna posiadać:

- kalkulator wspomnianych wcześniej dowolnych wyrażeń algebraicznych
- wykresy definiowanych przez użytkownika funkcji na podstawie dostępnych operatorów, funkcji elementarnych i funkcji specjalnych
- obliczenia numeryczne, takie jak całkowanie, wartość pochodnej funkcji w punkcie i pierwiastki funkcji na przedziale, wykonywane są na zdefiniowanych przez użytkownika funkcjach

Celem pracy jest realizacja takiej właśnie aplikacji, z zastrzeżeniem, że funkcje specjalne nie są priorytetem (wynika to z faktu, że niektóre funkcje specjalne nie posiadają dobrej implementacji w postaci ogólnie dostępnych bibliotek, a napisanie własnej implementacji biblioteki obliczającej wartości wszystkich popularnych funkcji specjalnych, to tak naprawdę temat na oddzielną pracę inżynierską z powodu złożoności i trudności zagadnienia).

Zakres pracy obejmuje między innymi opis platformy programistycznej .NET i języka C#, opis funkcji matematycznych, algorytmów numerycznych, projekt struktury aplikacji, model klas i pakietów tak dobranych, aby łatwe było dalsze rozwijanie aplikacji, opracowanie algorytmów ewaluacji i parsowania wyrażeń, opis dostępnych

funkcjonalności w programie, pokazanie efektów działania aplikacji, w tym testy samej aplikacji – m.in. testy poprawności obliczeń.

Wstęp

Wraz z rozwojem cywilizacyjnym wzrasta złożoność i trudność wykonywanych przez inżynierów i naukowców zadań w codziennej pracy. Jest to zrozumiałe, gdyż „prostsze” zadania zostały już wykonane i często nie ma sensu powielać tych samych zadań (chyba, że w celach edukacyjnych). Korzysta się więc z wyników wcześniejszych, a nowe odkrycia dotyczą zagadnień coraz bardziej trudnych.

Jednocześnie obserwujemy szybki wzrost mocy obliczeniowej komputerów i oprogramowania. Według prawa Moore'a liczba tranzystorów w procesorach komputerów podwaja się co dwa lata. Mamy więc do czynienia z wykładniczym wzrostem liczby tranzystorów. Co prawda, liczba tranzystorów nie musi się przenosić bezpośrednio na wydajność obliczeniową, ale w praktyce obserwuje się przynajmniej o 40% większą wydajność obliczeniową procesorów przy tym samym poborze energii, co dwa lata [3]. Co oznacza, że z biegiem czasu możemy sobie pozwolić na coraz dokładniejsze i bardziej kosztowne wydajnościowo obliczenia, gdyż mamy więcej mocy do dyspozycji.

Aby nieco ułatwić, przyśpieszyć pracę inżynierów i naukowców niezbędne są coraz lepsze narzędzia, które w tej pracy im pomagają. W przypadku, gdy inżynier/naukowiec wykonuje obliczenia, lub ogólnie działa w sferze matematyki, ma do czynienia z funkcjami, analizą matematyczną i wizualizacją wyników, (a dzieje się tak często) niezwykłą popularność zdobywając systemy obliczeń symbolicznych i numerycznych (takie jak np. prawdopodobnie najpopularniejszy – Mathematica firmy Wolfram Research). Dostępne oprogramowanie często jest jednak zbyt kosztowne i rozrosło się do ogromnych rozmiarów, które utrudniają efektywną pracę. Liczba dostępnych funkcji często zamiast zachęcać – przytłacza. Pewnie znajdzie się wielu inżynierów/naukowców którym wystarczą podstawowe obliczenia numeryczne, wygodny kalkulator naukowy i wizualizacja wyników czy funkcji w zamian za wygodny interfejs, szybki i zgrabny program, który uruchamia się w kilka sekund i nie pochłania zbyt dużej ilości zasobów.

W niniejszej pracy postawiono sobie za cel realizację takiego właśnie programu, wspomagającego obliczenia naukowe i inżynierskie, z uwzględnieniem również zbioru liczb zespolonych. Wybrano platformę technologiczną Microsoft .NET i język C# z powodu bardzo dużej ilości dostępnych klas w bibliotece standardowej [4] i docelowych platform (Windows 7, Windows Vista, Windows XP) oraz możliwości późniejszego

przeniesienia programu na inne platformy desktopowe (Linux, Mac OS X), jak i mobilne (Android, iOS, Windows Phone) za pomocą otwartoźródłowego kompilatora Mono [5].

Pierwszy rozdział pracy, to opis środowiska i języka programistycznego wykorzystywanego w projektowanej aplikacji. Rozdział drugi dotyczy realizacji aplikacji od strony algorytmicznej i technicznej, w tym również szczegóły projektowe i implementacyjne. Z kolei rozdział trzeci dotyczy uzyskanych efektów i testów aplikacji, w nim można również obejrzeć opis wszystkich funkcji i możliwości aplikacji.

Jak uważny czytelnik mógłby zauważyc, autor postanowił nieco rozbudować realizowaną aplikację w stosunku do tego, na co mógłby wskazywać tytuł, w aplikacji mamy znacznie więcej możliwości, niż tylko obliczenia na liczbach zespolonych. Z funkcjonalności aplikacji zostały zrealizowane między innymi:

- Wykresy dowolnych funkcji elementarnych zmiennej rzeczywistej
- Wykresy popularnych funkcji specjalnych zmiennej rzeczywistej
- Wykresy dowolnych funkcji elementarnych zmiennej zespolonej
- Wykresy wybranych funkcji specjalnych zmiennej zespolonej
- Kalkulator naukowy, uwzględniający liczby rzeczywiste i zespolone oraz wszystkie zaimplementowane w programie funkcje (elementarne i specjalne, zmiennej rzeczywistej i zespolonej)
- Moduł do obliczeń numerycznych, uwzględniający całkowanie numeryczne różnymi metodami, wartość pochodnej w punkcie różnymi metodami oraz numeryczne wyznaczanie pierwiastków funkcji na danym przedziale

W planach na przyszłość do zrealizowania pozostał moduł obliczeń symbolicznych i implementacja kolejnych funkcji specjalnych, zwłaszcza dla zmiennej zespolonej, gdyż to właśnie funkcji specjalnych zmiennej zespolonej jest w aplikacji najmniej (wynika to m.in. z trudności implementacyjnych).

1 Platforma .NET i język C#

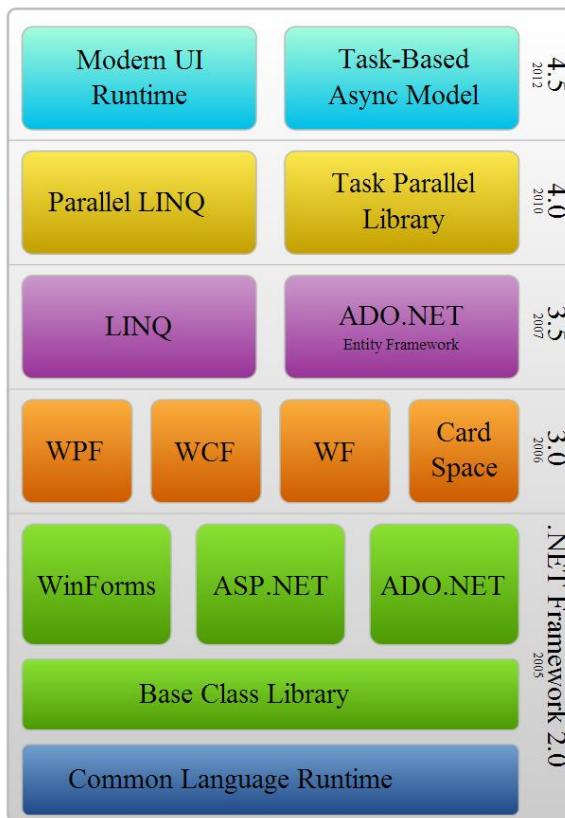
Framework .NET, to platforma programistyczna stworzona przez firmę Microsoft w celu prostszego i szybszego pisania aplikacji głównie dla systemów z rodziną Microsoft Windows. Posiada ogromną bibliotekę standardową zawierającą klasy ułatwiające zwykłe zadania, które należą do desktopowych aplikacji, takie jak np. obsługa wejścia/wyjścia, obsługa sprzętu, grafika komputerowa, łączność internetowa i sieciowa, operacje we współpracy z bazą danych, przetwarzanie plików XML, informacje o plikach i folderach i wiele innych. Jest platformą niezależną od języka programowania i umożliwia współpracę komponentów napisanych w różnych językach. Programy napisane dla platformy .NET są wykonywane na maszynie wirtualnej znanej jako „Common Language Runtime” (CLR). Platforma umożliwia zarządzanie pamięcią (garbage collector), kodem i wyjątkami. Jest uznawana za bezpieczną platformę bardziej niż Java, mimo że ta druga również posiada maszynę wirtualną. Większość powstających na systemy Windows aplikacji korzysta z platformy .NET [6]. Istnieje kilka alternatywnych implementacji platformy .NET w tym otwartoźródłowa implementacja multiplatformowa Mono, firmy Xamarin.

Język C#, to wieloparadygmatowy język programowania obejmujący m.in. silne typowanie, zorientowanie obiektowe (oparte na klasach), obsługę zdarzeń. Stworzony został specjalnie z myślą o platformie .NET, składniowo oparty na językach Java i C++. Miał na celu połączyć efektywność/produktywność w programowaniu w języku Java z wydajnością języka C++. C# na platformie .NET komplikowany jest do wspólnego dla platformy języka pośredniego „Common Intermediate Language” (CIL).

1.1 Historia frameworka .NET

Framework .NET, to efekt wieloletniej pracy firmy Microsoft, którego początek sięga 13 lutego 2002 roku, prawie 12 lat wcześniej. Początki platformy .NET jak i jej powstanie trzeba przypisać silnemu wpływowi platformy programistycznej Java. Powszechnie mówi się, że Microsoft „pozazdrościł” firmie Sun Microsystems platformy Java i filozofii „write once, run anywhere”. Microsoftowi, co prawda nie zależało na wszystkich systemach operacyjnych, ale chciał umożliwić pisanie aplikacji działających na wszystkich wersjach systemu Windows, na które dostępny jest instalator frameworka .NET. Platforma miała być wydajniejsza od Javy i jeszcze bardziej produktywna.

Framework przez ten cały okres nieustannie się rozwijał. O ile na początku pojawiały się opinie, że .NET to jedynie „kopia” Javy, to później wraz z dalszym dynamicznym rozwojem już przy wersji 3.5 przewyższył (choćby pod względem liczby klas w bibliotece standardowej) ówczesną wersję Javy [4][7]. Dzisiaj gdy framework doczekał się już wersji 4.5.1, wydaje się, że zdominował aplikacje desktopowe, co potwierdzają opinie niezależnych ekspertów (zwłaszcza w klasie aplikacji biznesowych) [8].



Rysunek 1.1 Schemat rozwoju framework'a .NET [6]

1.2 Historia języka C#

Historia języka C# przebiegała podobnie jak samej platformy .NET i ze względu na to, że jest to język dedykowany tej platformie, rozwijał się on razem z nią. Twórcą języka jest oczywiście firma Microsoft, ale zespół rozwijający ten język jest pod przywództwem Anders'a Hejlsberg'a – osoby która była oryginalnym autorem Turbo Pascala i głównym architektem Delphi [16]. Język początkowo spotkał się z dużą falą krytyki, zwłaszcza ze słów twórcy Javy Jamesa Goslinga [1]. Podobnie jednak, jak w przypadku rozwoju platformy .NET tak i w przypadku języka, szybko wyewoluował on z czegoś, co było nazywane jedynie „podróbką” Javy do potężnego języka o wielu zastosowaniach i ogromnej liczbie możliwości, jakim jest dzisiaj. Rozwijając się wraz z platformą .NET,

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

język ten wprowadzał również udoskonalenia w składni, i tak wraz z dodaniem do niego m.in. wyrażeń lambda i języka zapytań Linq, zyskał on pewne cechy języka funkcyjnego. Umożliwiło to pisanie krótszego i bardziej czytelnego kodu. Na dzień dzisiejszy język ten wydaje się, że znacznie wyprzedził swój pierwotny w postaci Javy, zwłaszcza jeżeli chodzi o liczbę funkcjonalności, możliwości i innych cech [9]. Co ciekawe, język stale się rozwija (nawet dynamiczniej niż sama platforma .NET) i w planach dla nowej wersji jest wśród wielu ekscytujących nowości między innymi „Roslyn” - kompilator jako usługa, które to rozwiązanie ma między innymi zbliżyć wydajność kodu C# do języków kompilowanych do kodu maszynowego takich jak C++.

	C# 2.0	C# 3.0	C# 4.0
Features added	<ul style="list-style-type: none">• Generics• Partial types• Anonymous methods• Iterators• Nullable types• Private setters (properties)• Method group conversions (delegates)• Covariance and Contra-variance• Static classes	<ul style="list-style-type: none">• Implicitly typed local variables• Object and collection initializers• Auto-Implemented properties• Anonymous types• Extension methods• Query expressions• Lambda expressions• Expression trees• Partial Methods	<ul style="list-style-type: none">• Dynamic binding• Named and optional arguments• Generic co- and contravariance• Embedded interop types ("NoPIA")

Rysunek 1.2 Rozwój języka C# w kolejnych wersjach aż do wersji wykorzystywanej w niniejszej pracy

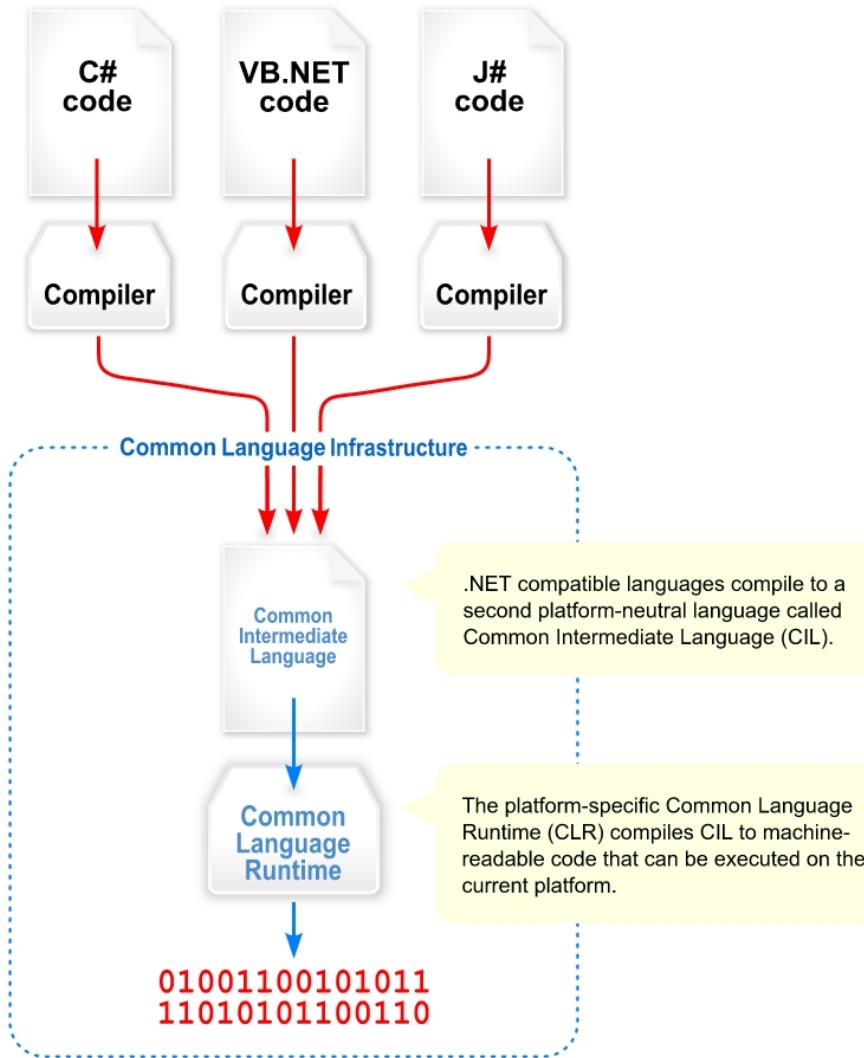
	C# 5.0 [31]	Future
Features added	<ul style="list-style-type: none">• Asynchronous methods• Caller info attributes	<p>C# 5.0</p> <ul style="list-style-type: none">• Compiler-as-a-service (Roslyn) <p>C# 6.0</p> <ul style="list-style-type: none">• Import type members into namespace• Succinct syntax for primary constructors• Readonly properties• Property expressions (property lambdas)• Method expressions• Parameter arrays for I Enumerable interfaces• Succinct null checking• Multiple return values• Constructor type inference

Rysunek 1.3 Rozwój języka C# w kolejnych wersjach - wersja najnowsza i planowana

1.3 Zasada działania

Jak zostało wspomniane wcześniej platforma jest niezależna od języka, a wszelki kod jest kompilowany do postaci języka pośredniego, który jest przetwarzany na maszynie wirtualnej. Istnieje więc pośrednik, w tym mamy również zarządzanie pamięcią, kodem i wyjątkami, wszystko to wpływa negatywnie na wydajność, ale pozytywnie na

produktywność i bezpieczeństwo. Kod pośredni jest kompilowany na maszynie wirtualnej (Common Language Runtime) do kodu maszynowego używając metod kompilacji JIT (just in time compilation).



Rysunek 1.4 Schemat kompilacji i działania aplikacji [6]

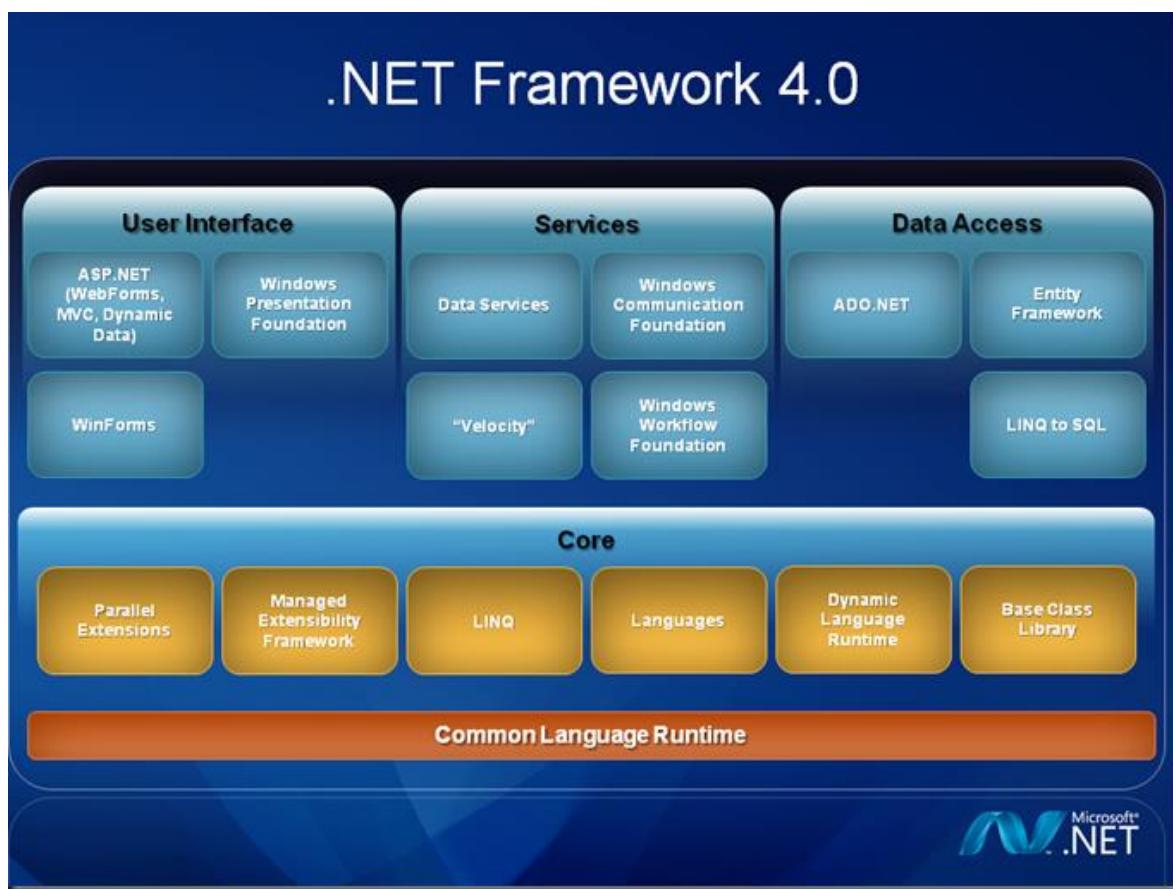
1.4 Stos platformy

Jak zostało wspomniane wcześniej platforma .NET składa się z bardzo wielu klas, pogrupowane są one w przestrzenie nazw (namespaces) – odpowiedniki pakietów z Java, zależnie od tego, jaką funkcjonalność spełniają. W samym Base Class Library (BCL) mamy do czynienia z siedemnastoma ustandaryzowanymi przestrzeniami nazw i dziewiętnaścioma nieustandaryzowanymi przestrzeniami nazw [10]. Klasy w nich zawarte służą do najrozmaitszych zadań, które może wykonywać aplikacja w systemie – np. czytanie i zapisywanie plików, rysowanie grafiki, interakcje z bazą danych, interakcje z

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

dokumentami XML i wiele innych. Nadzbiorem BCL jest Framework Class Library (FCL) zawierające BCL i inne biblioteki klas obecne w stosie platformy (często używana nazwa określająca całość bibliotek platformy .NET, to właśnie stos platformy).

Niestety, wraz z rozwojem stosu, platforma zmuszona była iść na przód i porzucać to, co przestarzałe i tak wersja 4.5 .NET framework nie obsługuje już systemu Windows XP, nadal dosyć popularnego wśród wielu użytkowników (w Polsce jest to 24,56% komputerów podłączonych do internetu [11]). System XP został porzucony z różnych przyczyn (także marketingowych), jedną z nich mógł być fakt, że nie obsługuje on wszystkich znaków standardu Unicode, co w dobie internacjonalizacji i lokalizacji oprogramowania, (idei od początku wspieranych przez .NET framework) wygląda bardzo niekorzystnie. Uwzględniając dodatkowo fakt, że zmiany z wersji 4.0 na 4.5 nie są duże i dotyczą głównie modern UI obecnego w systemach pokroju Windows 8, w niniejszej pracy inżynierskiej zdecydowano się na kompromis i wykorzystanie wersji 4.0 frameworka.



Rysunek 1.5 Stos platformy / schemat budowy platformy [2]

Najważniejsze (wybrane) części stosu platformy to:

- Common Language Runtime – wirtualna maszyna, na której wykonuje się aplikacja
- BaseClassLibrary – opisana wcześniej biblioteka klas podstawowych – zapis plików, obsługa wejścia/wyjścia, operacja na plikach XML i wiele innych
- Dynamic Language Runtime – działa nad CLR i zapewnia wsparcie dla języków dynamicznych, takich jak IronRuby
- Linq – zapewnia obsługę języka zapytań na obiektach w składni przypominającej język SQL
- Parallel Extensions – biblioteka umożliwiająca pisanie współbieżnych aplikacji, składa się z biblioteki Parallel Linq (czyli Linq ale zrównoległy, szybszy dzięki współbieżności) i Task Parallel Library – prawdziwa biblioteka do realizacji wygodnej współbieżności w aplikacjach
- Windows Presentation Foundation (WPF) – podsystem do tworzenia GUI rysowanego w trakcie działania aplikacji
- WinForms – podsystem zawierające API do tworzenia GUI poprzez wywoływanie natywnych metod z Win32 API
- ADO.NET – komponenty do współpracy z bazą danych i serwisami danych

1.5 **WinForms czy WPF**

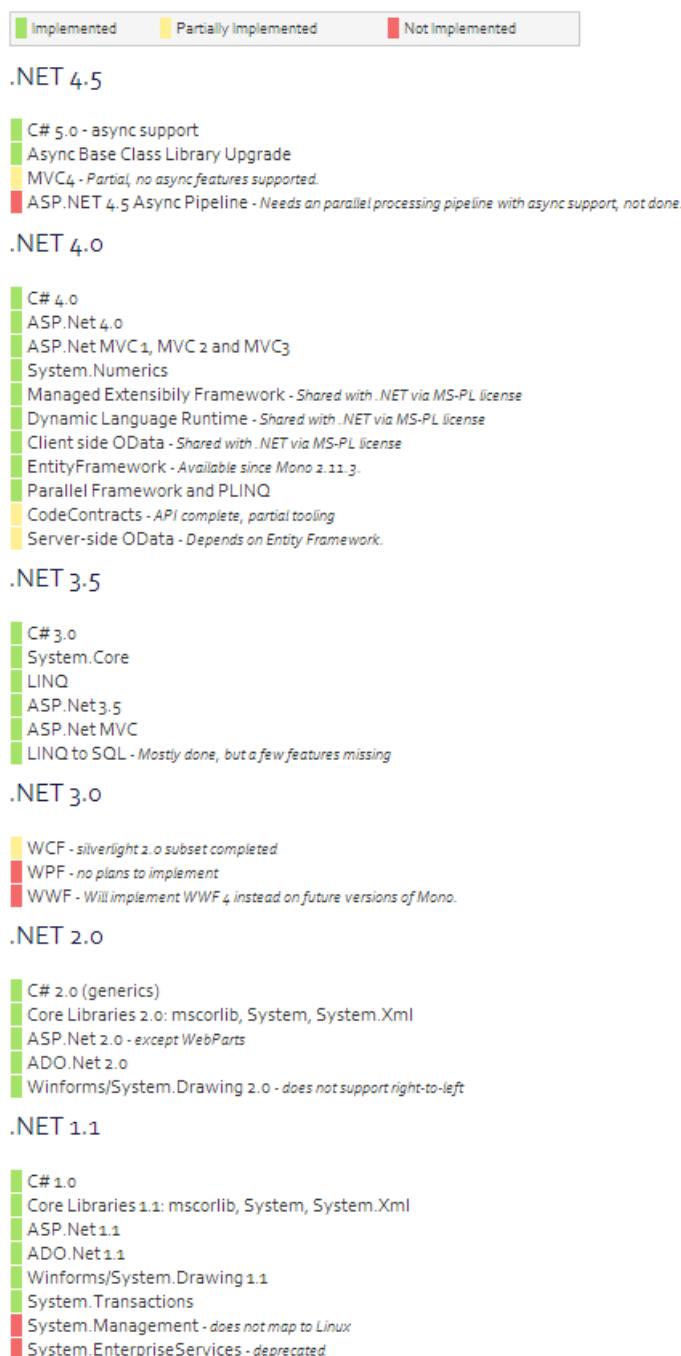
O ile większość z komponentów .NET może być używana łącznie, o tyle jako graficzny podsystem do programowania GUI w aplikacji należało wybrać tylko jeden. WPF wydaje się być naturalnym następcą WinForms, ponieważ ma znacznie więcej możliwości.

WPF umożliwia dostosowanie bardzo wielu parametrów używanych kontrolek do własnych potrzeb. Wynika to między innymi z faktu, że GUI w WPF jest renderowane (DirectX) w przeciwieństwie do WinFormsa, który tak naprawdę jest bardziej zestawem API napisanym w kodzie zarządzalnym do wywoływania natywnych windowsowych metod z Win32 API [13][14]. Tak więc w WPF możemy wszystko sobie dostosować pod nasze potrzeby, wygląd, kolor, a nawet cały styl przycisków i kontrolek. Istnieje nawet cały nowy język do tworzenia interfejsów – XAML, oparty na XML-u. W WPF mamy również lepszy „data binding” i wiele innych cech, których w WinForms nie znajdziemy.

Mimo istotnych zalet WPF-a niektórzy eksperci uważają, że ten nie jest przeznaczony, aby zastąpić WinFormsa, a oba podsystemy mogą być używane przez developerów do różnych celów [12].

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

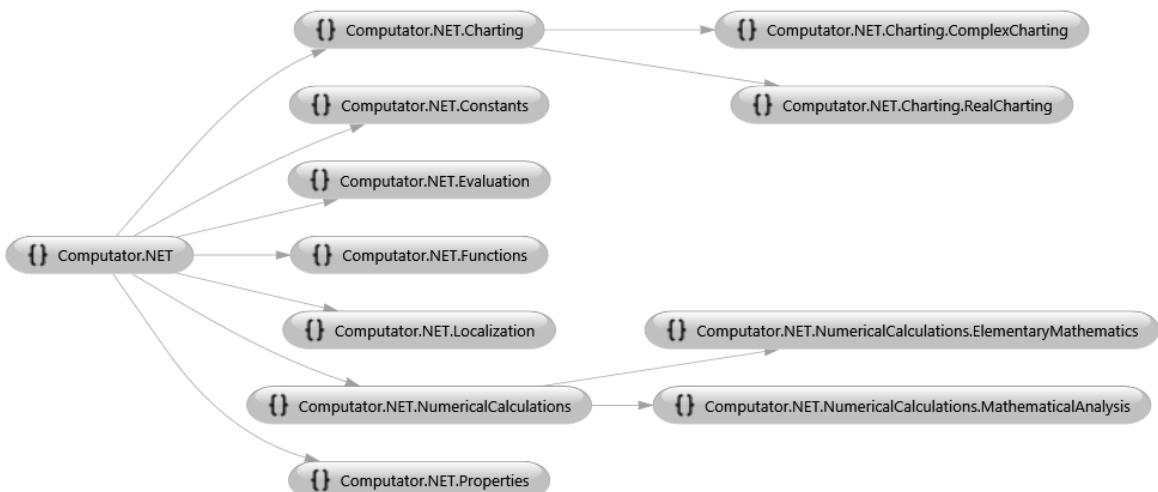
Chociaż WPF wydaje się bardziej przyszłościowy, w tej pracy do realizacji aplikacji wybrano WinForms. Zadecydowało o tym większe doświadczenie autora w programowaniu w WinForms oraz fakt, że tworzenie GUI za pomocą designer'a w WinForms jest o wiele prostsze niż w WPF-ie. Bardzo istotny okazał się przy tej decyzji również fakt, iż WPF nie jest (i nie będzie, jak widać na rysunku 1.6) wspierany przez kompilator otwartoźródłowy Mono Xamarin, a co za tym idzie aplikacji na WPF-ie nie udałoby się przenieść na inne systemy operacyjne (co jest w planach dalszego rozwoju aplikacji) [15].



Rysunek 1.6 Kompatybilność pomiędzy .NET Framework a kompilatorem Mono [15]

2 Projekt i implementacja aplikacji

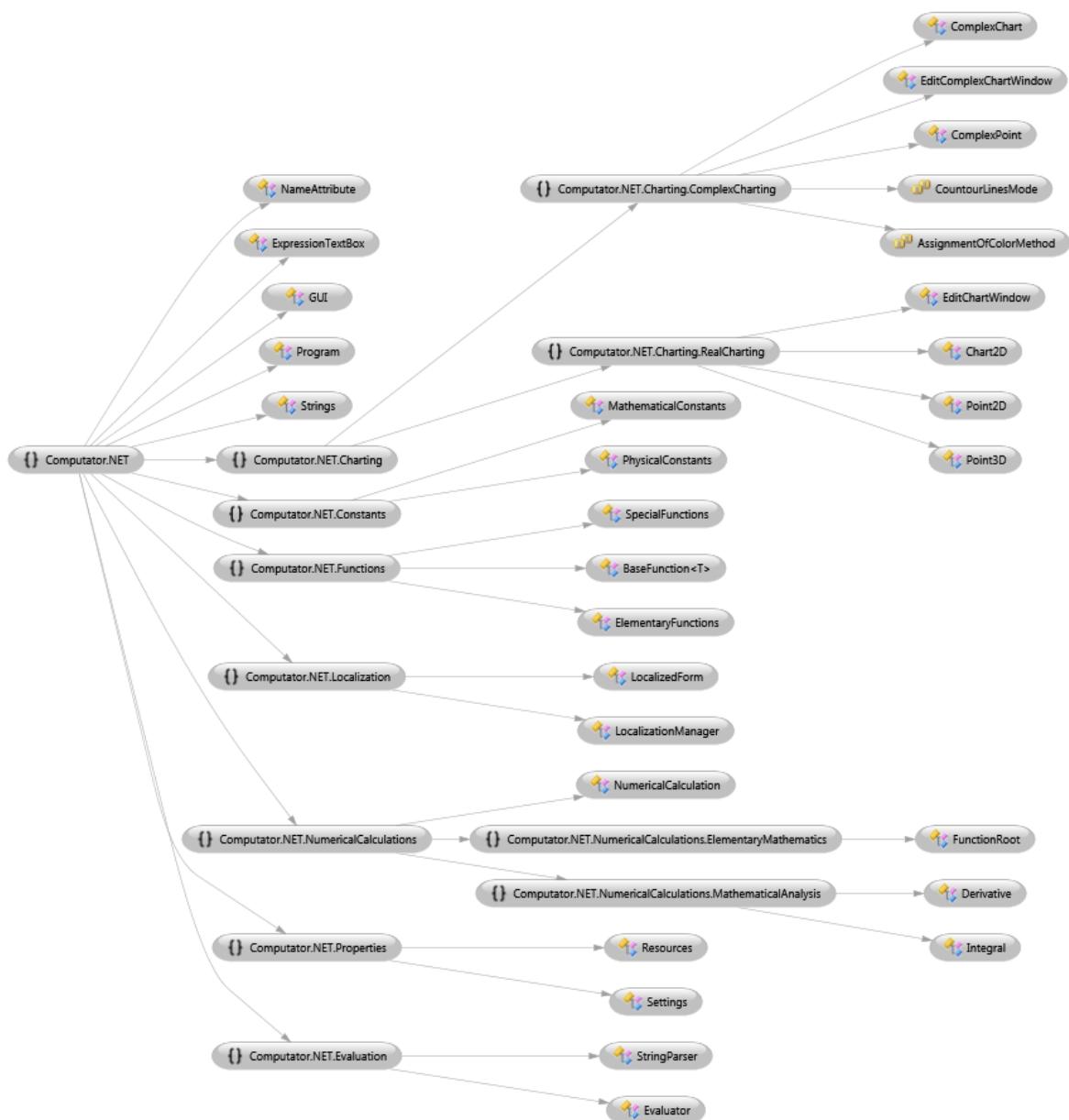
W programowaniu obiektowym, zanim przystąpi się do programowania aplikacji, należy najpierw projektować. Jest to niezwykle ważny okres w cyklu życia aplikacji, gdyż jego efekty wpłyną na późniejsze tempo rozwoju programu. Aby zapewnić programowi szybki i uporządkowany rozwój, należy wymyśleć odpowiednie nazwy klas modelujące w odpowiedni sposób rzeczywistość. Bardzo ważny jest podział na klasy oraz odpowiednie uporządkowanie tych klas w pakiety/przestrzenie nazw na podstawie tego, czy dane klasy dotyczą danego większego zagadnienia. W przypadku języka C# przestrzenie nazw i nazwy folderów, to niekoniecznie to samo. W projekcie natomiast zostało przyjęte, zgodnie z tym, jak to wygląda w Javie, że nowa przestrzeń nazw, to nowy folder o adekwatnej nazwie. W końcowej fazie projektowania uzyskano widoczny na rysunku 2.1 diagram pakietów.



Rysunek 2.1 Diagram przestrzeni nazw użytych w projekcie aplikacji

Dalszą fazą projektowania jest wymyślenie odpowiednich nazw klas (w praktyce te fazy często się pokrywają i projekt zmienia się w czasie programowania aplikacji). Projektowanie nabiera jeszcze większej wagi wraz ze wzrostem rozmiarów projektu. Jest to o tyle bardzo istotne, że w dużej aplikacji łatwiej się zgubić, poza tym często bywa tak, że jeżeli nad aplikacją mają pracować zespoły osób, to łatwiej jest rozdzielić pracę na kilka programistów, jeżeli struktura projektu jest podzielona na pakiety i klasy. Odpowiednie klasy zawierają metody odpowiadające możliwym działaniom na obiektach, jednakże w przypadku programowania aplikacji wspomagającej obliczenia naukowe i inżynierskie, często pojawiają się klasy wypełnione metodami statycznymi – czyli takimi,

które nie działają na instancji klasy (na obiekcie). W przypadku bytów, takich jak matematyczne funkcje elementarne czy funkcje specjalne, istnieją one po prostu jako metody statyczne wewnętrz klas odpowiednio identyfikujących rodzaj funkcji. Podobnie to wygląda z operacjami numerycznymi. Dlatego też (zwłaszcza z powodu ogromnej ilości zaimplementowanych w aplikacji funkcji) liczba metod statycznych obecnych w odpowiednich klasach (ElementaryFunctions i SpecialFunctions) jest na tyle duża, że nie da się przedstawić ich jako szczegółów poszczególnych klas na wspólnym diagramie klas. Aby jednak ukazać lepiej strukturę projektu na rysunku 2.2 przedstawiono diagram klas bez szczegółów, tj. bez ukazanych metod, własności i pól.



Rysunek 2.2 Diagram klas użytych w projekcie aplikacji

2.1 Pakiet Evaluation – uzyskiwanie definiowanej przez użytkownika funkcji w trakcie działania programu

Pierwszym i podstawowym problemem w pisaniu aplikacji wspomagającej obliczenia, jest uzyskiwanie funkcji na podstawie łańcucha znaków napisanych przez użytkownika w czasie wykonywanie programu (ang. „at runtime”).

2.1.1 Parsowanie wyrażeń matematycznych

Znane są przynajmniej dwa sposoby podejścia do tego problemu – jeden najbardziej powszechny i implementowany w większości aplikacji, to „parsowanie” (przetwarzanie) łańcucha znaków tak, aby uzyskać odpowiednią formę wyrażenia, którą można policzyć. W przetwarzaniu takim jest kilka kroków m.in. zamienia się wyrażenia matematyczne w reprezentacji infixowej, takiej jak $(5+2)\cdot 3$ na reprezentację, którą można łatwo policzyć – np. odwrotna notacja polska (RPN – reverse polish notation [24]), która eliminuje nawiasy i korzystając ze stosu (stos to struktura danych typu FIFO – first in, first out) ułatwia policzenie wartości wyrażenia. Poza tym, często w przypadku użycia funkcji, które są zdefiniowane w programie i użytkownik wprowadza je jako łańcuch znaków, identyfikacja ich i zamiana na wartości, wymaga tworzenia abstrakcyjnych drzew składni (drzewo jako struktura danych), tak jak w algorytmie Shunting-yard [23].

2.1.2 Dynamiczna komplikacja funkcji

Drugim, niezbyt popularnym rozwiązaniem, jest dynamiczna komplikacja funkcji i uzyskanie delegata na tę funkcję (analog wskaźnika na funkcję z języka C) dzięki mechanizmowi refleksji na podstawie danego łańcucha znaków w czasie działania programu. Być może to rozwiązanie jest mało popularne z powodu nieobecności w wielu językach i technologiach dynamicznej komplikacji i refleksji. W języku C# obie technologie są dostępne już od wersji 1.1 .NET Framework. Za tym rozwiązaniem przemawia jednak prostota w implementacji (nie trzeba przetwarzać wyrażeń, wystarczy przekazać je do komplikacji) i bezpieczeństwo – mniejsze ryzyko popełnienia błędów w kodzie z powodu łatwiejszego kodu do napisania. Ryzyko usterki, błędów i nieścisłości jest znacznie większe w przypadku przetwarzania wyrażeń, gdyż po prostu algorytmy są znacznie bardziej skomplikowane i stale trzeba je dostosowywać do dodawanych nowych funkcji.

Ostatecznie wybrano rozwiązanie z dynamiczną komplikacją z powodu przedstawionych zalet, ograniczonego czasu na projekt i chęci dalszej, prostej rozbudowy

aplikacji o nowe funkcje. Także fakt, że rozwiążanie to jest mało popularne, zachęcił autora do realizacji problemu ewaluacji wyrażeń właśnie w ten sposób.

W efekcie powstał pakiet Evaluation zawierający dwie klasy: StringParser i Evaluator. Pakiet został częściowo oparty o kod z odpowiedzi użytkownika forum stackoverflow na podobny problem programistyczny [22].

2.1.3 Klasa StringParser – część odpowiedzialna za przetwarzanie i normalizację wyrażenia

Klasa StringParser odpowiada za przetwarzanie wyrażenia, gdyż mimo wyboru metody dynamicznej komilacji, niektóre znaki wprowadzane przez użytkownika należy zamieniać na inne. Metody w niej zawarte, to tak naprawdę metody rozszerzające funkcjonalności klasy string (extension methods – popularna cecha języka, pozwalająca rozbudować istniejące klasy o dodatkowe metody). Przykładem jest znak potęgi w wyrażeniu 5^2 , który należy zamienić na funkcję pow(5,2), gdyż składnia języka C# nie dopuszcza znaku „ \wedge ” jako znaku potęgi. Realizują to trzy metody, z czego dwie publiczne są wywoływanie na przetwarzanym wyrażeniu w kolejności jedna po drugiej przed procesem dynamicznej komilacji.

```
public static string ReplacePow(this string input)
{
    string result = input.ReplacePow(@"(\d*x)\^(\d+\.\?\d*)");
    return result.ReplacePow(@"\(([^\^]+)\)\^(\d+\.\?\d*)");
}

public static string ReplacePow2(this string input)
{
    string result = input.ReplacePow(@"(\d+\.\?\d*)\^(\d*x)");
    return result.ReplacePow(@"(\d+\.\?\d*)\^\\(([^\^]+)\)");
}

private static string ReplacePow(this string input, string toReplace)
{
    return Regex.Replace(input, toReplace,
        "Math.Pow((double)($1),(double)($2))");
}
```

Pierwsza metoda używa wyrażeń regularnych do zamiany fragmentów typu x^2 na Math.Pow(x,2) druga natomiast w sytuacji odwrotnej, np. 2^x należy zamienić na Math.Pow(2,x). Dodatkowo, aby uniknąć wszelkich problemów z typami (C# jest type-safe) dokonuje się konwersji na typ zmienoprzecinkowy wyższej precyzji (8 bajtów). Analogiczne metody istnieją dla liczb zespolonych, jedyną różnicą jest to, że korzysta się tam z funkcji Complex.Pow.

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

Innym pojawiającym się problemem jest odpowiednie parsowanie liczby zespolonej. Jednym z bardzo dobrych rozwiązań do aplikacji wykorzystujących metodę parsowania wyrażeń jest projekt [27] napisany w VisualBasic. Okazało się jednak, że w przypadku dynamicznej komplikacji można o wiele prościej rozwiązać ten problem. Zakładając postać wyrażenia $x+yi$ wystarczy tak naprawdę zdefiniować stałą i w dynamicznej komplikacji oraz zamieniać yi na y^*i .

```
public static string ReplaceComplexNumbers(this string input)
{
    return Regex.Replace(input, @"(\d+\.\?\d*)i", "$1*i");
}
```

W kodzie wysyłanym do dynamicznej komplikacji zdefiniowana jest stała „i” w prosty sposób jako właściwość (property), wraz z opisem wyświetlonym w czasie działania aplikacji.

```
[Name("Imaginary unit"),
Category("Complex mathematical constants"),
Description("The imaginary unit or unit imaginary number, denoted as i, is a mathematical concept which extends the real number system  $\mathbb{R}$  to the complex number system  $\mathbb{C}$ , which in turn provides at least one root for every polynomial  $P(x)$  (see algebraic closure and fundamental theorem of algebra). The imaginary unit's core property is that  $i^2 = -1$ . The term imaginary is used because there is no real number having a negative square.")]
public static Complex i { get { return Complex.ImaginaryOne; } }
```

Opis oczywiście nie jest obowiązkowy (i nie istnieje dla wszystkich funkcji i stałych obecnych w aplikacji), ale autor projektu uważa, że wyświetlany w czasie działania aplikacji opis funkcji i stałych może zwiększyć atrakcyjność aplikacji.

2.1.4 Klasa Evaluator – dynamiczna kompilacja funkcji i uzyskiwanie delegata na funkcję

Klasa Evaluator odpowiada głównie za zbudowanie kompilowanego w pamięci kodu, za wstępne przetwarzanie wyrażenia za pomocą metod klasy StringParser i późniejsze doklejenie wyrażenia do kompilowanego kodu i samą dynamiczną kompilację za pomocą klasy CSharpCodeProvider [25]. Następnie, dzięki mechanizmowi refleksji wyszukuje się utworzoną w czasie runtime'u funkcję i delegata na nią. Cały ten proces wykonuje się w konstruktorze klasy.

```
public Evaluator(string input, bool isComplex = false)
{
    this.isComplex = isComplex;

    if (isComplex)
        Begin += complexModeLambda;
    else
        Begin += realModeLambda;

    normalized = Normalize(input);

    var provider = new CSharpCodeProvider();
    var parameters = new CompilerParameters {GenerateInMemory = true};
    parameters.ReferencedAssemblies.Add("System.dll");
    parameters.ReferencedAssemblies.Add("System.Numerics.dll");
    parameters.ReferencedAssemblies.Add("Meta.Numerics.dll");
    CompilerResults results =
provider.CompileAssemblyFromSource(parameters, Begin + normalized + End);

    try
    {
        Type cls = results.CompiledAssembly.GetType("FunctionsCreatorNS
.FunctionsCreator");
        MethodInfo method = cls.GetMethod("CustomFunction",
BindingFlags.Static | BindingFlags.Public);
        evaluatedFunction = (method.Invoke(null, null) as Delegate);
    }
    catch (FileNotFoundException)
    {
        throw new ArgumentException("Error in expression syntax. One of
used functions does not exist / is incompatible with given arguments or you ju
st made a mistake writing expression.");
    }
}
```

Metoda Normalize wykonuje wspomniane wcześniej parsowanie.

```
private string Normalize(string input)
{
    if (isComplex)
        return input.ReplaceComplexPow().ReplaceComplexPow2()
.ReplaceMultiplying().ReplaceComplexNumbers();
    else
        return input.ReplacePow().ReplacePow2().ReplaceMultiplying();
}
```

Parsowanie jest wykonywane za pomocą extension methods z klasy StringParser, tak jak wspomniano wcześniej (i opisane niektóre z tych metod). Kompilowany kod powstaje z trzech łańcuchów znaków: Begin – który jest stały i zawiera wstęp oraz definicje wszystkich funkcji, normalized, który jest sparsowaną postacią podanego przez użytkownika wyrażenia/funkcji i End, który zawiera znaki zakończenia kodu. String Begin budowany jest na podstawie kodu wszystkich funkcji matematycznych i stałych fizycznych/matematycznych zaimplementowanych w projekcie. Jest to niezbędne, aby funkcje były dostępne użytkownikowi w swoich krótkich, naturalnych nazwach i aby cały kod poprawnie się kompilował. Zdefiniowany jest on następująco:

```
private readonly string Begin =
    @"using System;using System.Numerics;using Meta.Numerics.Functions;
using System.ComponentModel;
namespace FunctionsCreatorNS
{
    public static class FunctionsCreator
    {
        " + MathematicalConstants.ToString().Replace("[", @"\/*[").Replace("]", @"\]*/")
        + PhysicalConstants.ToString().Replace("[", @"\/*[").Replace("]", @"\]*/")
        + ElementaryFunctions.ToString().Replace("[", @"\/*[").Replace("]", @"\]*/")
        + SpecialFunctions.ToString().Replace("[", @"\/*[").Replace("]", @"\]*/");
    }
}
```

Statyczne metody ToString() zwracają cały kod źródłowy (wszystkich metod statycznych) zawarty wewnętrz odpowiadnych klas (MathematicalConstants, PhysicalConstants, ElementaryFunctions, SpecialFunctions), pochodzących z odpowiednich pakietów (Functions i Constants). Jest to rozwiązań „brzydkie”, ale na dzień dzisiejszy jedyne [28], które umożliwia w dynamicznej komplikacji dostępność funkcji w swoich krótkich nazwach (np. sin(x) zamiast Math.Sin(x)). Dwa razy użyte metody Replace na każdym stringu zwracanym przez każdą metodę ToString(), mają na celu zamienić na komentarz opisy funkcji i stałych zawarte oryginalnie w klasach. Jest to robione tylko w celach optymalizacyjnych, gdyż kompilowanie funkcji i stałych wraz z opisami zwiększyłoby czas kompilacji bardziej niż zamienienie tych łańcuchów znaków na komentarz i nie byłoby wykorzystywane w programie.

Na koniec, w bloku try uzyskuje się delegat do skompilowanej w pamięci funkcji. Gdyby dynamiczna kompilacja przebiegła niepoprawnie, to zostanie wyrzucony wyjątek. Ten wyjątek zostaje obsłużony w taki sposób, że w bloku catch tworzony jest nowy „opisowy” wyjątek, tym razem czytelny dla użytkownika (tak, aby domyślał się, co mógł zrobić źle).

Po pomyślnej konstrukcji obiektu klasy Evaluator, dostępna jest skompilowana, stworzona przez użytkownika funkcja. Można ją wywołać poprzez publiczną, generyczną metodę Invoke (metoda jest generyczna, ponieważ możemy tworzyć zarówno funkcje zmiennej rzeczywistej jak i zespolonej)

```
public T Invoke<T>(T x)
{
    if (evaluatedFunction == null)
        throw new NullReferenceException("No function to invoke");

    T result = default(T);

    try
    {
        result = (T)evaluatedFunction.DynamicInvoke(x);
    }

    catch (Exception ex2)
    {
        if (x is double)
            result = (T)(object)double.NaN;
        else if (x is Complex)
            result = (T)(object)(new Complex(double.NaN, double.NaN));

        if (ex2 is Meta.Numerics.NonconvergenceException)
            throw ex2;
        else if (ex2 is Meta.Numerics.DimensionMismatchException)
            throw ex2;
        else if (ex2 is ArgumentException ||
ex2 is ArgumentOutOfRangeException)
            throw ex2;
        else
            throw new Meta.Numerics.NonconvergenceException("For chosen
values one (or more) of the functions in your expression cannot converge\n"
+ ex2.Message + "\n" + ex2.Source);
    }

    return result;
}
```

Metoda tam mogłaby wyglądać o wiele prościej, bo przecież na pozór wystarczyłoby wywołanie uzyskanego wcześniej delegata evaluatedFunction. Niestety, kod kompilowany w czasie wykonania aplikacji może generować wiele wyjątków, ponieważ zdarza się, że użytkownik nieumiejemnie definiuje funkcję. Np. czasami jedna z funkcji nie istnieje dla

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

danego argumentu, w takim przypadku najlepszą praktyką byłoby zwrócenie „nie liczby” (np. double.NaN), jednak autorzy używanych bibliotek niekoniecznie muszą przestrzegać tej dobrej praktyki. Też może się zdarzyć sytuacja, że użytkownik podał liczby, dla których funkcja istnieje i jest poprawna dla danych parametrów, ale używane biblioteki lub własna implementacja używają często szeregow do obliczania wartości funkcji. Zwłaszcza dotyczy to funkcji specjalnych, które rzadko są liczone bezpośrednio z definicji, a częściej z postaci szeregu. Niestety istnieją parametry, dla których taki szereg w biblioteчnej implementacji po prostu nie jest zbieżny, wtedy jedyne co można zrobić, to wyrzucić wyjątek o tym informujący NonconvergenceException().

2.2 Pakiety Functions i Constants – implementacja funkcji i stałych

Aby użytkownik mógł tworzyć swoje wyrażenia w programie, potrzeba cegiełek, z których buduje się własne funkcje i wyrażenia – czyli dobrze zdefiniowanych funkcji elementarnych i specjalnych oraz stałych - fizycznych i matematycznych.

2.2.1 Stałe matematyczne – klasa MathematicalConstants

Implementacja stałych sprowadza się do bardzo prostego procesu przypisywania im wartości numerycznych i zgodnie z tym, co było opisane w podrozdziale 2.1 na potrzeby dynamicznej komplikacji, istnieje metoda ToString(), której zadaniem jest eksport tego kodu jako łańcuch znaków.

Rysunek 2.3 Klasa MathematicalConstants wraz z zawartością

W matematyce nie ma zbyt wielu popularnych i często używanych stałych, więc wewnętrze tej klasy należy uważać bardziej za propozycję, która będzie w przyszłości rozbudowywana o nowe stałe, jeżeli zajdzie taka potrzeba.

Kod takiej klasy wygląda bardzo prosto, wystarczy chyba pokazać kilka linijek, żeby zrozumieć ideę. Oczywiście zgodnie z pomysłem autora stałe zawierają opis lub są przygotowane na dodanie takiego opisu.

```
[Name("name"),Category("Math constants"),Description("description")]
public static readonly double PI = Math.PI;

[Name("name"),Category("Math constants"),Description("description")]
public static readonly double e = Math.E;

[Name("name"),Category("Math constants"),Description("description")]
public static readonly double ConwaysConstant = 1.30357;

[Name("name"),Category("Math constants"),Description("description")]
public static readonly double KhinchinsConstant = 2.6854520010;
```

Of course, the method `ToString()` simply returns all this code as a string type.

2.2.2 Stałe fizyczne – klasa `PhysicalConstants`

Podobnie wygląda kod źródłowy klasy `PhysicalConstants`. Jednak w jej przypadku nie można już mówić o małej liczbie używanych stałych. Mimo dużej liczby stałych fizycznych autor zamieścił ich jedynie kilka, gdyż klasa ma służyć jedynie za przykład i umożliwić późniejszą dalszą rozbudowę.

Rysunek 2.4 Klasa `PhysicalConstants` wraz z zawartością

Jak widać i jak zapewne czytelnik się domyśla, klasa ta jest zupełnie analogiczna do klasy `MathematicalConstants`.

2.2.3 Funkcje elementarne – klasa ElementaryFunctions

W przypadku funkcji sprawa wygląda o wiele ciekawiej. Klasa ElementaryFunctions zawiera bardzo wiele znanych (i mniej znanych) funkcji elementarnych. Duża część z nich była już zaimplementowana w BaseClassLibrary, niektóre z nich są zaimplementowane używając definicji, a jeszcze inną część jest napisana samodzielnie. Funkcje bardziej złożone, których implementacja jest często trudna, zazwyczaj pochodzą z biblioteki Meta.Numerics [29] używanej w projekcie (dotyczy to jednak głównie funkcji specjalnych). Aby zaoszczędzić kod i stworzyć bardziej użyteczny i nowoczesny, zdecydowano się na napisanie funkcji szablonowych łącząc w ten sposób funkcję w jednośc i zależnie od typu argumentu wywołuje się funkcje dla typu zespolonego lub rzeczywistego. Przykładem jest szablonowa funkcja sinus.

```
public static T sin<T>(T value)
{
    if (value is double)
        return (T)(object)Math.Sin((double)((object)(value)));
    else if (value is Complex)
        return (T)(object)Complex.Sin((Complex)((object)(value)));
    else
        throw new ArgumentException();
}
```

Jak widać jest ona sparametryzowana parametrem T oznaczającym typ argumentu. I zależnie od tego, jakiego typu argument do niej przekażemy, to inna funkcja z biblioteki standardowej się wywoła. Jest to coś, czego brakuje w standardowej klasie matematycznej języka C#. Jeżeli typ nie będzie pasował do żadnego ze sprawdzanych typów, to zostanie wyrzucony wyjątek.

Niektóre funkcje, których implementacji brakowało w bibliotece standardowej, często można było zaimplementować w prosty sposób jako przekształcenie, korzystające z innych funkcji. Przykładem są choćby funkcje area hiperboliczne, takie jak area tangens hiperboliczny.

```
public static T artanh<T>(T value)
{
    if (value is double)
        return (T)(object)(0.5 * Math.Log((1 + (double)((object)(value))) /
(1 - (double)((object)(value)))));

    else if (value is Complex)
        return (T)(object)(0.5 * Complex.Log((1 + (Complex)((object)(value))) /
(1 - (Complex)((object)(value)))));

    else
        throw new ArgumentException();
}
```

Niektóre funkcje pojawiające się w nauce i inżynierii są nieciągłe, implementacji ich często brakuje w wielu bibliotekach, a napisanie ich jest bardzo łatwe.

```
public static double DiracDelta(double x)
{
    if (x != 0.0)
        return 0;
    else
        return double.PositiveInfinity;
}

public static double Heaviside(double x)
{
    if (x > 0.0)
        return 1;
    else if (x < 0.0)
        return 0;
    else
        return 0.5;
}
```

Te dwie powyższe, to chyba bardzo znane przykłady. Czasami są one określane funkcjami specjalnymi. Ze względu jednak na łatwość ich implementacji zostały zamieszczone w klasie ElementaryFunctions (delta Diraca przez niektórych nie jest uznawana nawet za funkcję [31]).

Niektóre funkcje mogą być czasochłonne w samodzielnej implementacji (chociaż w przypadku funkcji elementarnych nie jest to zazwyczaj więcej niż kilka do kilkunastu minut), a nie są zaimplementowane w bibliotece standardowej, więc warto skorzystać z zewnętrznej biblioteki numerycznej, takiej jak meta numerics [29]. Jeżeli typy odpowiadają, to można po prostu utworzyć delegat Func.

```
public static Func<long, long, long> GCF = AdvancedIntegerMath.GCF,
NWD = AdvancedIntegerMath.GCF;

public static Func<long, long, long> LCM = AdvancedIntegerMath.LCM,
NWW = AdvancedIntegerMath.LCM;
```

Spostrzegawczy czytelnik zauważyczy tutaj jeszcze jedną rzecz – w celu umożliwienia wielojęzyczności aplikacji, często należy tworzyć wiele takich samych funkcji, ale o innych nazwach – tak jak obecny tutaj największy wspólny dzielnik dwóch liczb typu long (NWD) i greatest common factor (GCF), jest to niestety wada dynamicznej komilacji.

Jak wspomniano wcześniej, czasami trzeba trochę zawałczyć ze złymi praktykami autora biblioteki i zamiast wyrzucać wyjątek w funkcji dla niepoprawnych liczb - zwracać „nieliczbę”

```
public static Func<double, double> sgn = (x) => (double.IsNaN(x)) ? dou
ble.NaN : (double)(Math.Sign(x));
```

Szczególnie jest to ważne w przypadku, gdy tak, jak w pisanej aplikacji mamy do czynienia z rysowaniem wykresu funkcji na przedziale. „Nieliczby” oznaczają po prostu, że funkcja w danym miejscu nie istnieje.

2.2.4 Funkcje specjalne – klasa SpecialFunctions

Ostatnią klasą z opisywanych pakietów jest klasa SpecialFunctions. Jak już wspomniano wcześniej w tej pracy, samodzielna implementacja funkcji specjalnych, to złożone zadanie, prawdopodobnie dobrze nadające się na oddzielną pracę inżynierską. Jednak wspomniana już biblioteka meta.numerics została wybrana do tego projektu właśnie z powodu implementacji niektórych funkcji specjalnych.

Function	Real	Complex	Notes
Gamma	✓	✓	also $\ln \Gamma$, incomplete Gamma
Psi (Digamma)	✓	✓	also polygamma $\psi^{(n)}$
Beta	✓		also incomplete Beta
Error Function	✓	✓	also erfc , erf^{-1} , Faddeeva, Fresnel C and S
Exponential Integrals	✓	✓	includes E_{in} , E_i , E_n , and trigonometric integrals C_i and S_i
Bessel J and Y	✓		also for non-integer orders, spherical Bessel j and y
Modified Bessel I and K	✓		also for non-integer orders, Airy Ai and Bi
Coulomb Wave Functions F and G	✓		accurate even in quantum tunneling region
Reimann Zeta	✓		also Dirichlet η
Dilogarithm Li_2 (Spence's Function)	✓	✓	
Orthogonal polynomials	✓		Chebyshev T , Hermite H , Legendre P , Laguerre L , Zernike R
Elliptic Integrals	✓		Legendre F, K, E; Carlson R_F and R_D

Rysunek 2.5 Funkcje specjalne zaimplementowane w bibliotece Meta.Numerics [32]

Oczywiście, jak w [32] możemy przeczytać, biblioteka numeryczna Meta.Numerics „porządnie” napisana jest dedykowana dla środowiska .NET i zawiera też inne numeryczne możliwości. W niniejszej pracy postanowiono jednak używać zewnętrznej biblioteki tylko do funkcji specjalnych (i dosłownie dwóch-trzech funkcji elementarnych).

Tak więc w klasie SpecialFunctions znajdują się głównie aliasy, uogólnienia i kilka przekształceń funkcji zawartych w opisywanej wyżej bibliotece numerycznej. Przykładem implementacji szablonowej funkcji jest choćby popularna gamma Eulera. Tak samo, jak w przypadku szablonowej funkcji sinus w klasie ElementaryFunctions, tak tutaj, zależnie od typu argumentu wywołujemy odpowiednią funkcję.

```
public static T gamma<T>(T value)
{
    if (value is double)
        return (T)(object)AdvancedMath.Gamma((double)((object)value));
    else if (value is Complex)
        return (T)(object)cmplxFromMeta(AdvancedComplexMath.Gamma(cmplx
ToMeta((Complex)(object)value)));
    else
        throw new ArgumentException();
}

public static T Γ<T>(T value)
{
    if (value is double)
        return (T)(object)AdvancedMath.Gamma((double)((object)value));
    else if (value is Complex)
        return (T)(object)cmplxFromMeta(AdvancedComplexMath.Gamma(cmplx
ToMeta((Complex)(object)value)));

    else
        throw new ArgumentException();
}
```

Niestety biblioteka Meta.Numerics używa własnego typu liczby zespolonej (nie korzysta z bibliotecznego System.Numerics.Complex) i niezbędna jest konwersja w obie strony między tymi dwoma implementacjami liczby zespolonej za pomocą metod cmplxFromMeta i cmplxToMeta. Używając unikodu udało się również stworzyć dodatkowy alias dla funkcji gamma, za pomocą często używanego do jej oznaczenia symbolu greckiej dużej litery gamma Γ , może to umożliwić zwiększenie czytelności, gdy użytkownik wprowadza formułę do policzenia czy wykreślenia w aplikacji.

Czasami, gdy brakuje implementacji funkcji dla zmiennej zespolonej w bibliotece można po prostu utworzyć delegat Func na funkcję biblioteczną, przykładem jest funkcja poligamma.

```
public static Func<int, double, double> ψn = AdvancedMath.Psi;
```

Znowu zostały wykorzystane znaki specjalne ze standardu unicode. Symbol używany do zapisu funkcji powinien być maksymalnie zbliżony do zapisu znanego z matematyki $\psi^{(n)}$.

Z niewiadomych przyczyn, biblioteka liczyła niepoprawnie wartość funkcji Eta Dirichleta dla argumentów nieujemnych. Na szczęście istnieje formuła zastępcza na liczenie tej funkcji z użyciem Zety Riemanna. Przy użyciu „lambda expressions” i krótkiego wyrażenia warunkowego rozwiązano ten problem.

```
public static Func<double, double> DirichletEta = (x) => x >= 0 ?
AdvancedMath.DirichletEta(x) : (1 - Math.Pow(2, 1 - x)) * RiemannZeta(x);
```

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

Podsumowując, należy przedstawić zawartość obu klas funkcji, tak aby pokazać wszystkie zaimplementowane i dostępne dla użytkownika funkcje.

[] []

**Rysunek 2.6 Zawartość klasy
ElementaryFunctions**

Rysunek 2.7 Zawartość klasy SpecialFunctions

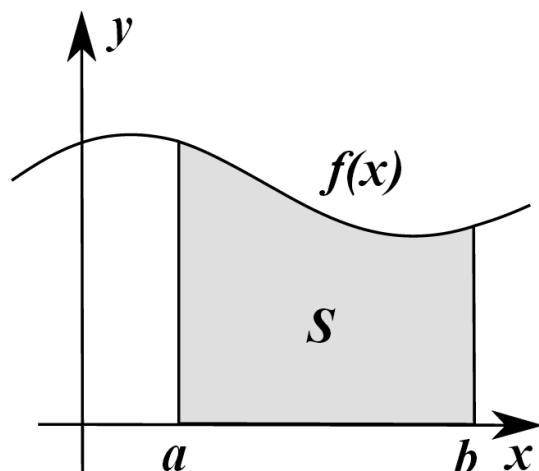
Mimo bardzo dużej liczby dostępnych funkcji, z uwzględnieniem faktu, że niektóre funkcje pominięto (pominięto głównie aliasy – różne nazwy na tę samą funkcję) w celu zmieszczenia czytelnych rysunków na jednej kartce, nadal istnieje wiele funkcji matematycznych (zwłaszcza funkcji specjalnych), które można by zaimplementować. Nie pozwolił jednak na to ograniczony czas i często trudności implementacyjne. Przykładem po kontakcie z autorem biblioteki meta.numerics okazało się, że zaimplementowanie niektórych funkcji specjalnych dla zmiennej zespolonej jest znacznie trudniejsze niż dla zmiennej rzeczywistej [33].

2.3 Obliczenia numeryczne – pakiet NumericalCalculations

Pakiet do podstawowych obliczeń numerycznych na funkcjach był w planach aplikacji od samego początku, jednak został zrealizowany dopiero na samym końcu z powodu pozornej trywialności zagadnienia. Jak się jednak później okazało, zagadnienie jest ciekawe i ukrywa w sobie kilka trudności, jednak z braku czasu zrealizowane zostały tylko najprostsze operacje numeryczne w dodatku najbardziej znymi i prostymi metodami.

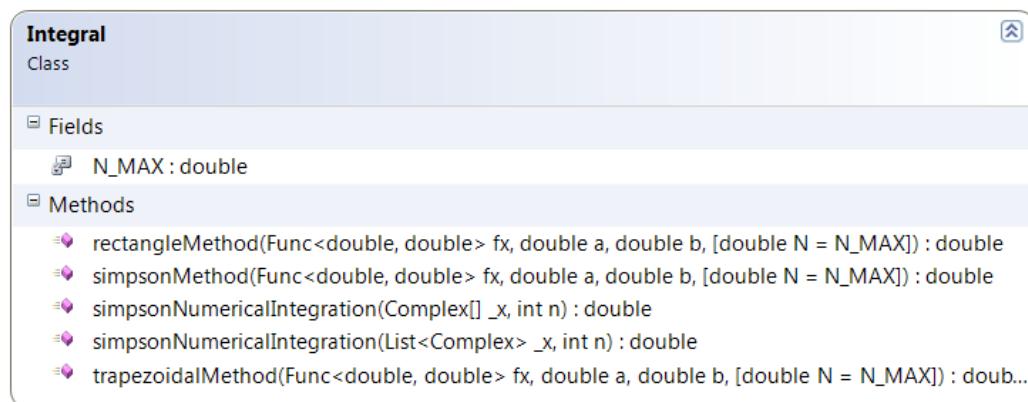
2.3.1 Całkowanie numeryczne – klasa Integral

Całkowanie numeryczne polega na numerycznym znalezieniu pola powierzchni zakreślonego przez funkcję na przedziale $[a,b]$ wraz z osią OX przy interpretacji pola powyżej osi OX jako dodatniego, a pola poniżej tej osi, jako ujemnego.



Rysunek 2.8 Całkowanie numeryczne polega na znalezieniu przybliżonej wartości S

Za operację całkowania numerycznego odpowiadają w aplikacji statyczne metody klasy Integral.



Rysunek 2.9 Zawartość klasy Integral

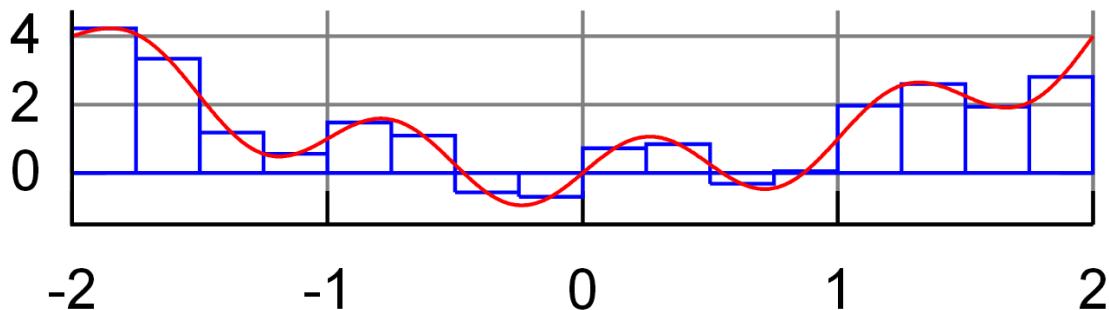
2.3.1.1 Metoda prostokątów

W najprostszej metodzie całkowania numerycznego nazywanej metodą prostokątów korzystamy tak naprawdę bezpośrednio z definicji całki według Riemanna i co się później w rozdziale ostatnim okaże, to właśnie ta „wzgardzana” przez wielu metoda jest najlepsza dla dużych przedziałów $[a,b]$ i dla wartości $N \rightarrow \infty$ ilości przedziałów na które dzieli metodą przedział $[a,b]$.

Algorytm metody wygląda bardzo prosto i polega na:

1. Podzieleniu przedziału $[a,b]$ na N różnych przedziałów tak, że wartość x w i -tym przedziale wynosi $x_i = a + i \cdot h$, gdzie $h = (b-a)/N$
2. Sumowaniu kolejnych pól powierzchni prostokątów tworzonych przez przedział h i wartość funkcji w punkcie $x_{i+0,5}$
3. Zwróceniu uzyskanej sumy

Ideę metody można dodatkowo zobrazować na rysunku.



Rysunek 2.10 Metoda prostokątów zobrazowana [34]

Implementacja tej metody w języku C# jest bardzo łatwa, ale nie należy zapominać o pewnych optymalizacjach, w miarę efektywną implementację napisano w sposób poniższy.

```
public static double rectangleMethod(Func<double, double> fx,
double a, double b, double N = N_MAX)
{
    double h = Math.Abs(b - a) / N;
    double s = 0, x;

    for (int i = 0; i < N; i++)
    {
        x = a + (i+0.5) * h;
        s += fx(x);
    }
    return h*s;
}
```

Jak widać, metoda przyjmuje delegat $\text{Func}<\text{double}, \text{double}\rangle$ - przyjmujący i zwracający typ double . Przedział całkowania $[a,b]$ i parametr N , który przyjmuje wartość domyślną w

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

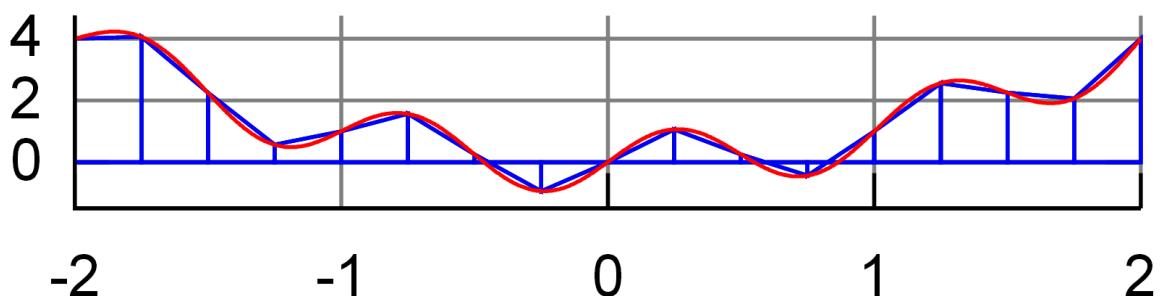
przypadku niepodania go w wywołaniu. N_MAX, to stała w klasie Integral, wynosząca 1e5, nazwa wskazuje na to, że jest to stosunkowo maksymalna użyteczna wartość N. Bardziej chodzi tutaj o pragmatyczny punkt widzenia, dla N większych od N_MAX operacje całkowania po prostu trwają za długo (więcej niż około jednej sekundy, co jest maksimum aby nie zaburzyć dobrego doświadczenia z użytkowania programu).

2.3.1.2 Metoda trapezów

Kolejną zaimplementowaną metodą całkowania numerycznego jest metoda trapezów, dla małych przedziałów znacznie dokładniejsza niż metoda prostokątów. Algorytm metody jest prosty i polega na:

1. Podzieleniu przedziału $[a,b]$ na N różnych przedziałów tak, że wartość x w i-tym przedziale wynosi $x_i = a + i \cdot h$, gdzie $h = (b-a)/N$
2. Sumowaniu kolejnych N-1 pól utworzonych przez trapez z punktu i do punktu $i+1$ o podstawach równych odpowiednio wartości funkcji w punkcie i oraz $i+1$ oraz wysokości trapezu równej $h = (b-a)/N$
3. Zwróceniu uzyskanej sumy

Ideę algorytmu świetnie obrazuje następujący obrazek



Rysunek 2.11 Metoda trapezów zobrazowana [34]

Implementacja w C# wygląda następująco

```
public static double trapezoidalMethod(Func<double, double> fx,
double a, double b, double N=N_MAX)
{
    double h = Math.Abs(b - a) / N;
    double s=0,x1,x2;

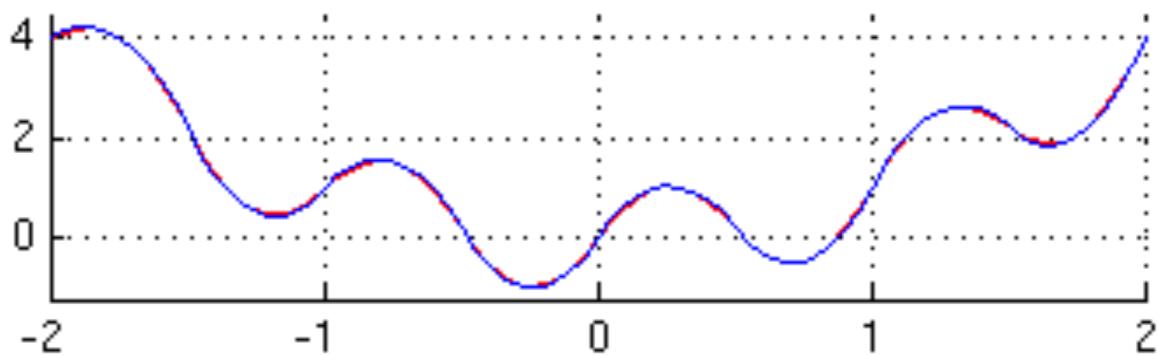
    for(int i=0;i<N;i++)
    {
        x1=a+i*h;
        x2=a+(i+1)*h;
        s += 0.5 * h * (fx(x2) + fx(x1));
    }
    return s;
}
```

2.3.1.3 Metoda Simpsona

Metoda Simpsona, zwana również metodą parabol występuje nieco rzadziej w literaturze czy materiałach edukacyjnych. Jest natomiast w wielu przypadkach bardzo dobrą i uniwersalną metodą, używa jej się między innymi w wielu popularnych programach do analizy Fouriera.

Algorytm metody polega na:

1. Podzieleniu przedziału $[a,b]$ na N różnych przedziałów tak, że mamy wartość x w i -tym przedziale wynoszącą $x_i = a + i \cdot h$, gdzie $h = (b-a)/N$ i N jest liczbą parzystą
2. Sumowaniu kolejnych iloczynów wartości funkcji w punktach od $i=1$ do $i=N-1$ i liczby 4 jeżeli i jest nieparzyste, a liczby 2 jeżeli i jest parzyste
3. Dodaniu do uzyskanej sumy wartości funkcji w punktach a i b
4. Przemnożeniu uzyskanego wyniku przez $h/3$ i zwrócenie go jako wynik całkowania



Rysunek 2.12 Ilustracja metody Simpsona [34]

Implementacja w aplikacji wygląda następująco:

```
public static double simpsonMethod(Func<double, double> fx,
double a, double b, double N = N_MAX)
{
    if (N % 2 != 0) //not even
        N++;

    double h = Math.Abs(b - a) / N;
    double s = 0, x;

    for (int i = 1; i < N; i++)
    {
        x = a + i * h;

        if(i%2==1)
            s += 4 * fx(x);
        else
            s += 2 * fx(x);
    }

    return h*(fx(a) + s + fx(b))/3.0;
}
```

2.3.2 Różniczkowanie numeryczne – klasa Derivative

Różniczkowanie numeryczne polega na wyznaczeniu wartości pochodnej funkcji w określonym punkcie. Opierając się lub rozwijając definicję pochodnej w punkcie:

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Gdzie $f(x)$, to funkcja, której pochodną w punkcie x mamy wyznaczyć, a $f'(x)$, to właśnie szukana pochodna.

Naiwny algorytm wyznaczania pochodnej zaimplementowany w języku C# mógłby wyglądać następująco:

```
public static double derivativeAtPoint(Func<double, double> f, double x)
{
    return (f(x + double.Epsilon) - f(x)) / double.Epsilon;
}
```

Gdzie `double.Epsilon` to najmniejsza dodatnia i różna od zera możliwa wartość zmiennej typu `double`. Bo skoro Δx ma dążyć do zera to powinniśmy je zaimplementować jako najmniejszą możliwą wartość. Niestety ten naiwny algorytm nie ma prawa zadziałać z powodu samej reprezentacji liczby zmiennoprzecinkowej za pomocą mantysy i wykładnika w pamięci komputera. Otóż jeżeli x jest wartością w miarę normalną np. $x=1$, to za pomocą wykładnika jest to tak zapisane, że gdyby zwiększyć wartość x o najmniej znaczący bit, zwiększylibyśmy wartość x o znacznie większą wielkość niż `double.Epsilon`, gdyż mamy do dyspozycji 64bit. Natomiast `double.Epsilon` to około $1e-324$, jest to wielokrotności mniej niż najmniej znaczący bit w reprezentacji liczby 1, więc dodanie tych dwóch liczb do siebie nie zmienia niczego, tj. $x+double.Epsilon=x$, tak jakby dodać do x zero, chociaż jest to niezgodne z matematyką, jest to jak najbardziej zgodne z informatyką.

Potrzeba więc innej implementacji niż naiwna, albo Δx musi być odpowiednio duże (ale nie za duże), albo musimy stosować jakieś sztuczki, aby pominąć liczenie pochodnej z najbardziej podstawowej definicji, tj. liczyć ją w jakiś inny sposób lub z innej postaci definicji. Jak autor ma zwyczaju zaimplementuje różne metody, tak aby użytkownik programu mógł sobie porównać wyniki z różnych metod.

Klasa składa się więc z różnych metod umożliwiających liczenie pochodnej funkcji w punkcie na różne sposoby. Autor opracował własny sposób wyznaczania Δx w czasie wykonania metody, chociaż w niektórych metodach zaimplementowano uznane na świecie metody liczenia numeryczne pochodnej funkcji w punkcie zgodnie z [38].

Rysunek 2.13 Zawartość klasy Derivative

Najprostsza metoda, to po prostu skończona różnica dwóch funkcji w dwóch bardzo bliskich punktach.

```
public static double finiteDifferenceFormula(Func<double, double> fx,
double x)
{
    double dx = x * EPS;
    if (dx == 0)
        dx = EPS_MAX;

    return (fx(x + dx)-fx(x)) / dx;
}
```

Należy zauważyć, że Δx jest liczone względem wartości x , jest to zrobione w celu zachowania małej różnicy między dwoma punktami oraz uniknięcia błędu dyskretyzacji, występującego we wspomnianym wcześniej algorytmie naiwnym. EPS, to odpowiednio mała wartość domyślnie zainicjowana jako $1e-9$.

Algorytm można nieco poprawić, zwiększając stabilność i dokładność poprzez liczenie różnicy z dwóch punktów.

```
public static double twoPointfiniteDifferenceFormula(Func<double, double> fx, double x)
{
    double dx = x * EPS;
    if (dx == 0)
        dx = EPS_MAX;
    return (fx(x + dx) - fx(x - dx)) / (2 * dx);
}
```

W literaturze [38] można spotkać również metodę obliczenia pochodnej w punkcie uznawaną za bardziej stabilną i dokładną numerycznie.

```
public static double stableFiniteDifferenceFormula(Func<double, double> fx, double x)
{
    double dx;
    double h = Math.Sqrt(EPS) * x;
    if (h == 0.0)
        h = Math.Sqrt(EPS_MAX);

    double xph = x+h;

    dx=xph-x;
    return (fx(xph) - fx(x)) / dx;
}
```

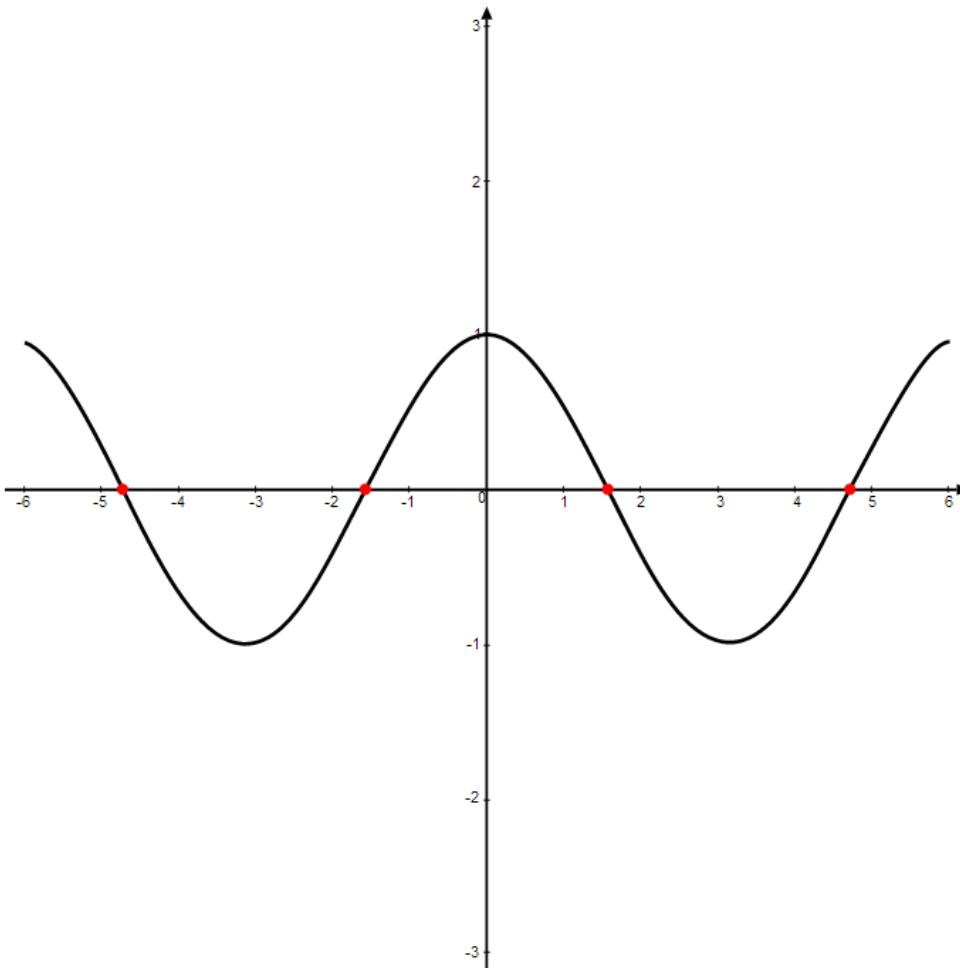
Chodzi o to, że tutaj h jest odpowiednio większe niż w innych przypadkach, a dx jest liczony dokładnie względem różnicy pomiędzy dwoma punktami – jest to bliższe prawdy ze względu na tzw. rounding error.

Często wykorzystywana jest też tzw. postać centrowa pięciopunktowa, jest to chyba najpowszechniej wykorzystywane metoda obliczania numeryczne pochodnej funkcji w punkcie.

```
public static double centeredFivePointMethod(Func<double, double> fx, double x)
{
    double dx = x * EPS;
    if (dx == 0)
        dx = EPS_MAX;
    // [f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)] / 12h
    return (fx(x-2*dx)-8*fx(x-dx)+8*fx(x+dx)-fx(x+2*dx))/12*dx;
}
```

2.3.3 Szukanie pierwiastków funkcji – klasa FunctionRoot

Zagadnieniem znanim jeszcze z bardzo wczesnych etapów edukacji jest szukanie zer funkcji (pierwiastków funkcji), czyli miejsc gdzie funkcja przyjmuje wartość zero. Sposób analityczny na rozwiązyaniu tego problemu zależy od badanej funkcji, ale sprowadza się do znalezienia takiego x które jest rozwiązaniem równania $f(x) = 0$.



Rysunek 2.14 Czerwone punkty to miejsca zerowe (pierwiastki) funkcji [39]

Metody numeryczne znajdowania pierwiastków funkcji działające na dowolnej funkcji dzielą się na metody szukające zer funkcji na zamkniętym przedziale (metoda bisekcji i reguła falsi) i metody niezależne od przedziału (metoda Newtona, metoda siecznych, interpolacja i odwrócona interpolacja), istnieją też metody będące kombinacją innych metod (np. metoda Brenta) [40].

Jeżeli wiemy o funkcji dodatkowe informacje, (np. wiemy, że jest wielomianem), to istnieją specyficzne dla danego typu funkcji metody znajdowania pierwiastków (jak np. w przypadku równania kwadratowego wyznaczanie x_1 i x_2 na podstawie odpowiedniego wzoru).

W aplikacji postanowiono zaimplementować przykładowe dwie popularne metody szukania pierwiastków funkcji w przypadku ogólnym (tj. w przypadku gdy nie mamy żadnych dodatkowych informacji o funkcji). Wybrano metodę bisekcji i metodę siecznych jako najbardziej popularne metody. Oczywiście nic nie stoi na przeszkodzie, żeby z czasem rozbudować klasę FunctionRoot o nowe metody szukania pierwiastków funkcji.

=

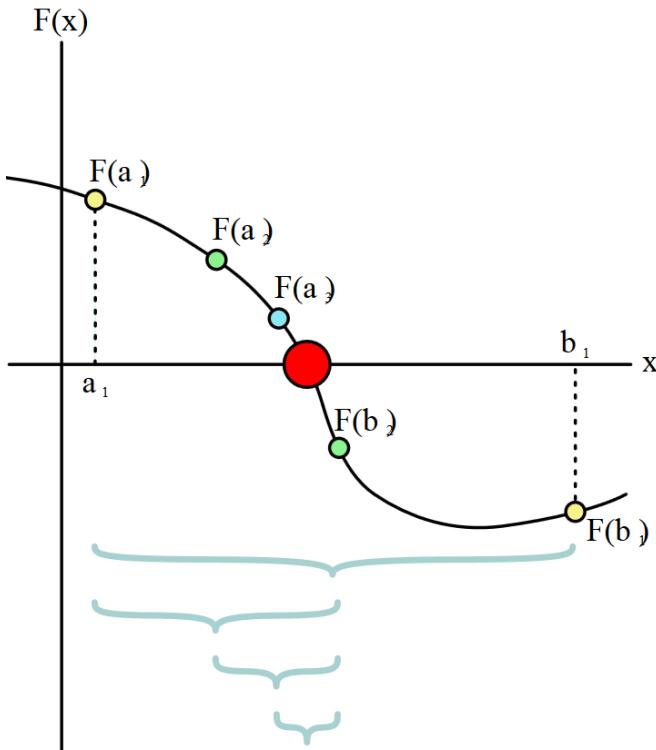
Rysunek 2.15 Zawartość klasy FunctionRoot

2.3.3.1 Metoda bisekcji

Metoda bisekcji, zwana również metodą połowienia przedziału, to bardzo prosta metoda poszukiwania miejsca zerowego funkcji w danym przedziale $[a,b]$. Generalnie opiera się ona na założeniu, że funkcja przechodząc przez miejsca zerowe zmienia znak (np. z dodatniego na ujemny lub odwrotnie).

Algorytm, przebiega następująco.

1. Wykonanie N razy (gdzie N jest duże) odpowiedniej sekwencji (punkty 2-6)
2. Wyznaczenie $x = (a+b)/2$, czyli wartości x w poowie przedziału $[a,b]$
3. Jeżeli $f(a) \cdot f(x)$ jest mniejsze od zera (funkcja zmienia znak w przedziale $[a,x]$), to do b przypisuje się wartość x ($b := x$)
4. Jeżeli jest inaczej i $f(a) \cdot f(x)$ jest mniejsze od zera (czyli funkcja zmienia znak w przedziale $[x,b]$), to do a przypisuje się wartość x ($a := x$)
5. Jeżeli $f(a)$ jest równe zero lub $f(b)$ jest równe zero to algorytm kończy się szybciej niż w N krokach i zwracane jest odpowiednio a lub b.
6. Jeżeli a jest równe b, to algorytm kończy się (szybciej niż w N krokach) i zwracane jest a.



Rysunek 2.16 Ilustracja graficzna metody bisekcji [41]

Implementacja w języku C# jest przedstawiona poniżej.

```
public static double bisectionMethod(Func<double, double> fx,
double a, double b, double N = N_MAX)
{
    double x = (a + b) / 2.0;

    for (int i = 0; i < N; i++)
    {
        //first interval - [a,x]
        if (fx(a) * fx(x) < 0)
            b = x;

        //second interval - [x,b]
        else if (fx(b) * fx(x) < 0)
            a = x;
        else
        {
            if (fx(a) == 0)
                return a;
            else if (fx(b) == 0)
                return b;
        }

        if (a == b) return a;
        else x = (a + b) / 2.0;
    }
    return x;
}
```

2.3.3.2 Metoda siecznych

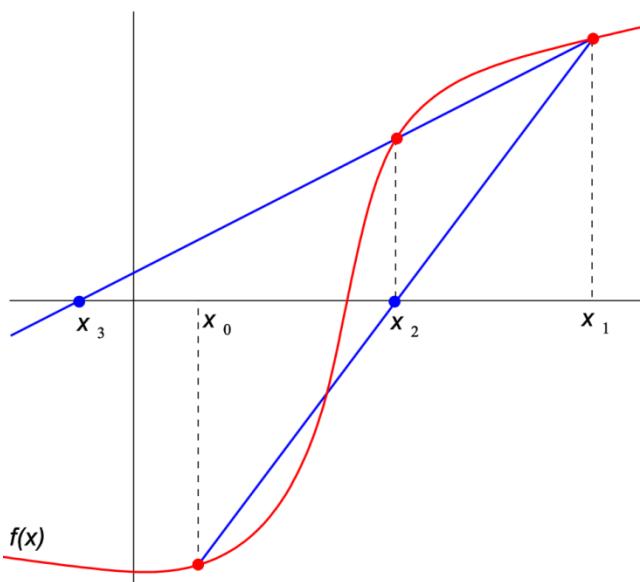
Metoda siecznych jak sama nazwa mówi używa siecznych funkcji $f(x)$, której miejsca zerowego szukamy, do interpolacji funkcji na niewielkich przedziałach tak, aby wyznaczać coraz dokładniejsze przybliżenia wartości pierwiastków funkcji badanej $f(x)$.

Algorytm jest tylko odrobinę bardziej złożony niż w metodzie połowienia przedziału. Głównie opera się on na zależności rekurencyjnej na kolejne przybliżenia wartości pierwiastka funkcji:

$$x_n = x_{n-1} - f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$$

Przebieg algorytmu jest następujący:

1. Zaczynamy od przypisania zmiennym x_1 i x_2 wartości a i b, gdzie $b > a$
2. Pomiędzy punktami o współrzędnych $(x_1, f(x_1))$ i $(x_2, f(x_2))$ zostaje utworzona wirtualna linia
3. Szukamy pierwiastka równania tej linii, tzn. takiego x , dla którego równanie tej linii przyjmuje wartość 0
4. Wartość tego pierwiastka zostaje przypisana do wartości x_2 , natomiast x_1 przyjmuje starą wartość x_2
5. Punkty 2-4 są powtarzane N razy, chociaż algorytm może się skończyć szybciej, jeżeli uzyskane x_2 już jest dokładnym miejscem zerowym funkcji (tj. $f(x_2)=0$) lub jeżeli różnica między x_2 i x_1 jest równa zero lub jeżeli różnica w wartości funkcji w punkcie x_1 i x_2 jest równa zero (funkcja może być wolnozmienna)
6. Zwracana jest wartość x_2 jako przybliżona wartość pierwiastka funkcji badanej f



Rysunek 2.17 Ilustracja algorytmu działania metody siecznych [42]

Implementacja w aplikacji opiera się mocno na wcześniej opisanym algorytmie słownym.

```
public static double secantMethod(Func<double, double> fx,
double a, double b, double N = N_MAX)
{
    double x1=a, x2=b, xOld;

    for (int i = 0; i < N; i++)
    {
        if (x2 - x1 == 0.0 ||(fx(x2) - fx(x1)) == 0.0 || fx(x2) == 0.0)
            break;
        xOld = x2;
        x2 = x2 - (fx(x2) * (x2 - x1)) / (fx(x2) - fx(x1));
        x1 = xOld;
    }
    return x2;
}
```

2.4 Wykresy funkcji zmiennej rzeczywistej – klasa Chart2D

Napisanie klasy rysującej poprawne wizualnie, konfigurowalne i podobajace się nawet wybrednym matematykom i fizykom wykresy, to nie lada wyzwanie. Tutaj ujawnia się też inna, niewspomniana wcześniej przyczyna, dla której wybrano WinForms – jest w nim obecna gotowa i naprawdę rozbudowana klasa do tworzenia różnych typów konfigurowalnych wykresów – klasa Chart [43].

Klasa Chart nie została jednak napisana od podstaw przez firmę Microsoft lub jej pracowników, a została kupiona wraz z prawami autorskimi majątkowymi od firmy Dundas już w kwietniu 2007 [45][46]. Występują co prawda niewielkie różnice pomiędzy implementacją Microsoftu (pracownicy Microsoftu wycięli z kodu pewne funkcje), jednak zmiany ogólnie są bardzo niewielkie [46].

Użycie jednak bezpośrednio klasy Chart w projekcie, nie byłoby dobrym pomysłem. Jest to bardzo rozbudowana klasa do różnych typów wykresów i reprezentacji danych, np. dane giełdowe, wykresy kolumnowe itd. Wydaje się więc, że klasa Chart nie nadaje się za bardzo do wykresów funkcji matematycznych zmiennej rzeczywistej, a bardziej do prezentacji np. danych biznesowych [44].

Z powodów wyżej wymienionych zdecydowano się na dziedziczenie po klasie Chart i zaimplementowaniu nowych metod, pól i własności w klasie potomnej pozwalających na użyteczne wykorzystanie nowej klasy Chart2D jako klasy do wyświetlania wykresów funkcji zmiennej rzeczywistej, nie wykluczając jednak możliwości reprezentacji na wykresie np. punktów wczytanych z pliku. Dodano również obsługę zdarzeń tak, aby była możliwa łatwa obsługa wykresu (np. łatwe zbliżanie i oddalanie wykresu za pomocą kółka myszy).

Klasa Chart charakteryzuje się wieloma możliwościami konfiguracji, posiada bardzo wiele własności i metod. Aby ułatwić zadanie postanowiono w implementowanej klasie Chart2D ustawić już w konstruktorze wszystkie domyślnie odpowiednie parametry dla klasy robiącej wykresy funkcji. Np. sposób rysowania wybrano jako FastLine, który jest bardzo szybkim sposobem na narysowanie interpolowanej z wielu punktów funkcji. Wyróżnia się również tym, że powstający wykres funkcji jest gładki, a nie tak, jak w najwnej metodzie obliczania wielu punktów i łączenia ich prostymi - schodkowy. W dalszym ciągu przedstawiono zawartość klasy Chart2D.



Rysunek 2.18 Zawartość klasy Chart2D

2.4.1 Błędy w klasie Chart

„Współpraca” z klasą Chart z przestrzeni nazw System.Windows.Forms.DataVisualization.Charting okazała się ciekawym i unikalnym doświadczeniem. Mimo początkowego zachwytu możliwościami tej klasy w trakcie używania jej metod i możliwości napotkano na aż trzy wyraźne błędy w tej właśnie klasie, jest to unikalne doświadczenie w tym sensie, że autor nigdy wcześniej nie spotkał się z jakimkolwiek błędem w tak oficjalnym, profesjonalnym wydawać by się mogło oprogramowaniu (w dodatku ustandaryzowanym). Także w czasie implementacji aplikacji napotkano wielokrotnie na błędy i trudności wewnątrz biblioteki Chart, które dopiero po długim czasie i analizie zidentyfikowano.

2.4.1.1 Pierwszy błąd – wyjątek 'System.OverflowException' w System.Windows.Forms.DataVisualization.dll „Value was either too large or too small for a Decimal”

Pierwszym z trzech na jakie się natknęto było wewnętrzne użycie w Chart zmiennych typu decimal do operacji przeskalowywania mimo przyjmowania wartości punktów x i y jako zmiennych typu double [47]. Naiwne rozwiązanie tego problemu polegałoby na niedodawaniu wartości niemieszczących się w przedziale wartości zmiennej decimal, który to przedział wynosi -7.9×10^{28} do 7.9×10^{28} [48]. W dodatku B zamieszczono wyjątek, który się pojawia w tym błędzie wraz z zawartością stosu wywoływań i co ciekawe, ograniczenie się do przedziału typu decimal nie rozwiązuje problemu. Należy zmniejszyć przedział jeszcze trochę [47], prawdopodobnie w środku klasy Chart są jeszcze wykonywane dodatkowe operacje, które zwiększają wartość zmiennych typu decimal.

Ostatecznie, rozwiązaniem tego problemu było zaimplementowanie w klasie potomnej stałych pól UNDERFLOW_VALUE i OVERFLOW_VALUE odpowiednio, poniżej i powyżej których wartości nie zostawały dodawane do wykresu, polom tym przypisano wartości zgodne z proponowanymi w [47].

2.4.1.2 Drugi błąd – wyjątek 'System.OverflowException' w System.Drawing.dll “Overflow error”

Drugi znaleziony jest o wiele bardziej uciążliwy. Właściwie na początku wydawało się, że pojawia się znikąd, ustalenie źródła błędu było bardzo trudne. Pomogła w tym wnikliwa analiza zawartości stosu wywoływań zamieszczonego w dodatku C.

Co ciekawe, problem jest opisany w internecie od długiego czasu. Na forum stackoverflow znajduje się niejedno pytanie z prośbą o pomoc, które do dzisiaj pozostaje otwarte i bez odpowiedzi [49], również inne fora zawierają pytania o ten problem jak [50]. Jeszcze ciekawiej robi się, jeżeli zajrzymy na forum firmy Dundas – pierwotnego autora klasy Chart – tam też jest temat z opisanym tym właśnie błędem i też pozostał zostawiony bez odpowiedzi [51]. Temat został poruszony nawet na forum firmy Microsoft w dziale wsparcia dla kontrolki Chart i mimo ciągłego „podbijania” tematu przez niezadowolonych programistów oraz pięcioletniej obecności wątku na tym forum, problem nie tylko nie został rozwiązany, ale nawet nie doczekał się oficjalnej odpowiedzi Microsoftu w tej sprawie [52].

Autorska dogłębiańska analiza sytuacji pozwoliła ustalić pewne fakty dotyczące występowania problemu.

1. Problem występuje przy rysowaniu funkcji, które w niektórych miejscach przedziału rysowania bardzo szybko rosną lub maleją, jak np. sferyczne funkcje Bessela drugiego rodzaju w okolicach zera.
2. Problem występuje tylko, jeżeli wykres ma być liniowy (Line lub jeszcze częściej FastLine), nie występuje przy wykresie punktowym
3. Analiza stosu wywołań wskazuje, że wyjątek występuje po wywołaniu funkcji DrawLine z biblioteki GDI+, która jest wywoływana przez funkcję FastLineChart.DrawLine
4. Współrzędne punktów w metodzie FastLineChart.DrawLine różnią się bardzo mocno, np. $x_0=0$ a $x_1=1e10$

Do rozwiązania problemu nie udałoby się dojść gdyby nie zdano sobie sprawy z tego, że to funkcja DrawLine z GDI+ jest przyczyną wywołania wyjątku, a więc coś jest nie tak z rysowaniem linii. Przeszukując internet w poszukiwaniu przyczyn wyjątków wywoływanych przez funkcję DrawLine z GDI+ natrafiono na [53], gdzie się okazało, że GDI ma pewne wewnętrzne ograniczenia co do współrzędnych, na których może rysować linię, tj. wartość przy rysowaniu linii z punktu (x_1, y_1) do (x_2, y_2) żadna z wartości

współrzędnych nie może być większa ani mniejsza od pewnych wartości i nie chodzi tutaj o ograniczenie wartości wykorzystanych typów liczbowych int lub float tylko o wewnętrzne ograniczenie pola rysowania w GDI, jak opisano w [54][55].

To skłoniło do śmiaływniosków, że funkcja FastLineChart.DrawLine nie robi żadnego skalowania wartości punktów na współrzędne ekranu ani żadnych przeliczeń czy sprawdzeń tych wartości tylko wywołuje funkcję DrawLine z GDI+ prawdopodobnie tylko rzutując floaty na inty. Jest to ewidentny bug albo przynajmniej nieprzemyślana wada konstrukcyjna w klasie Chart.

Rozwiązanie jest podobne do rozwiązania błędu pierwszego, tylko, że tutaj należy naprawdę całkiem mocno ograniczyć wartości dodawanych do wykresu punktów, gdyż według [53] limity to: minimum -1 073 741 760, maksimum 1 073 741 951. Co ciekawe po zaimplementowaniu tych zmian nadal błąd występował, chociaż zniknął w przypadku używania trybu „Line”, a pozostał jedynie w trybie „FastLine”. Jak opisano jednak w [56] tryb FastLine może dokonywać różnych interpolacji i innych operacji matematycznych na punktach w celu przyśpieszenia rysowania, można więc przypuszczać, że wartości punktów rysowanych mogą być wyższe niż podawanych z liczeniem wartości zadanej przez użytkownika funkcji.

Przeprowadzono więc testy empiryczne w wyniku których wykazano, że dopiero przy 500-krotnym zmniejszeniu wartości podanych wyżej jako minimum i maksimum dla współrzędnych rysowania w GDI+ błąd przestał się pojawiać. Jest to spore ograniczenie, ponieważ w ten sposób na wykresie nigdy nie pojawią się wielkości większe niż mniej więcej 0.2e7 i mniejsze niż -0.2e7. Ale dopóki Microsoft nie naprawi tego błędu (na co się nie zanosi, błąd w najnowszej wersji framework'a .NET 4.5.1 nadal występuje) jest to jedyna opcja.

2.4.1.3 Trzeci błąd - wyjątek 'System.Runtime.InteropServices.ExternalException' w System.Drawing.dll “A generic error occurred in GDI+”

Błąd trzeci występuje najbardziej losowo i w dodatku pojawia się niedeterministycznie, tzn. że nie zawsze ta sama odtworzona sytuacja zwraca wyjątek. Na podstawie zamieszczonej zawartości stosu wywołań w dodatku D i testów empirycznych ustalone generalnie, że ma on związek z obsługą zdarzeń myszki i wyświetlaniem ToolTip'a w czasie powiększania lub pomniejszania wykresu. Niestety w internecie brakuje

jakiejkolwiek wzmianki o tym błędzie a również w aplikacji pojawia on się nieczęsto i nie zawsze w tych samych sytuacjach.

Z powodu braku pomysłów na rozwiązywanie i marginalnej częstości pojawiania się błędu rozwiązywanie problemu, przynajmniej na razie, porzucono.

2.4.2 Inicjalizacja klasy Chart2D

Jak zostało wcześniej wspomniane, przynajmniej częściowym powodem stworzenia klasy Chart2D dziedziczącej po klasie Chart była chęć prostego ustawiania konfiguracji parametrów klasy Chart już przy inicjalizacji, w konstruktorze

Zadanie jest wyjątkowo łatwe, po prostu mamy rozbudowany konstruktor

```
public Chart2D()
{
    this.functions = new List<Functions.BaseFunction<double>>();
    this.MouseLeave += new EventHandler(_MouseLeave);
    this.MouseEnter += new EventHandler(_MouseEnter);
    this.MouseDoubleClick += new MouseEventHandler(_MouseDoubleClick);
    this.MouseClick += new MouseEventHandler(_MouseClick);
    this.MouseWheel += new MouseEventHandler(_MouseWheel);
    this.MouseDown += new MouseEventHandler(_MouseDown);
    this.MouseUp += new MouseEventHandler(_MouseUp);
    this.chartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.FastPoint;
    scalingFactor = 1;
    xOnlyZoomMode = yOnlyZoomMode = false;
    initChart();
}
```

W konstruktorze dokonujemy inicjalizacji innych obiektów oraz dodajemy funkcje obsługujące odpowiednie zdarzenia. Ustawienia parametrów dziedziczących z klasy Chart dokonuje się w funkcji initChart(). Jest w niej bardzo dużo ustawiania różnych parametrów (tak aby wykres wyglądał jak wykres funkcji a nie jak wykres np. giełdowy), poniżej przedstawiono część kodu funkcji initChart().

```
private void initChart()
{
    //.....
    chartArea1.AxisX.Crossing = 0D;
    chartArea1.AxisX.InterlacedColor = System.Drawing.Color.White;
    chartArea1.AxisX.IsLabelAutoFit = false;
    chartArea1.AxisX.LabelAutoFitMaxFontSize = 13;
    chartArea1.AxisX.LabelStyle.Font =
new System.Drawing.Font("Microsoft Sans Serif", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(23
8)));
    //.....
    this.Titles.Add(title1);
    this.Dock = DockStyle.Fill;
}
```

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

Ogółem linii w metodzie initChart() jest ponad 60, widać dzięki temu jak bardzo konfigurowalna jest sama klasa Chart.

2.4.3 Reprezentacja funkcji matematycznych w klasie Chart2D

Bardzo istotnym problemem jest „trzymanie” w klasie Chart2D dodanych funkcji matematycznych, bo przecież jeżeli ma to być wygodne dla użytkownika, to trzymanie samych policzonych punktów zdecydowanie odpada. Należy przecież umożliwiać rozszerzanie obszaru wykresu itd., co wymaga obliczenia wartości funkcji w nowych punktach.

2.4.3.1 Kolekcja dodanych funkcji

Kluczowe okazuje się tutaj trzymanie odpowiedniej reprezentacji dodawanej funkcji w odpowiedniej strukturze danych.

```
private List<Functions.BaseFunction<double>> functions;
```

Mamy więc listę obiektów klasy generycznej BaseFunction typu double. Opisu klasy BaseFunction nie ma co prawda w niniejszej pracy, ale jak można się domyśleć jest to po prostu klasa która zawiera delegat Func<T,T> typu szablonowego T (w tym przypadku double) i łańcuch znaków reprezentujący nazwę funkcji.

```
class BaseFunction<T>
{
    public BaseFunction(Func<T,T> function, T argmin, T argmax, string name)
    {
        this.f = function;
        this.argmax = argmax;
        this.argmin = argmin;
        this.name = name;
    }

    public BaseFunction(Func<T, T> function, string name)
    {
        this.f = function;
        this.name = name;
    }

    public Type getType()
    {
        return typeof(T);
    }

    public T eval(T x)
    {
        return f(x);
    }
}
```

```
    public string name { get; set; }
    public T argmin { get; set; }
    public T argmax { get; set; }

    private Func<T,T> f;
}
```

Jak widać w klasie BaseFunction nie ma nic, co mogłoby kogokolwiek zaskoczyć. Właściwości argmin i argmax służą do określenia przedziału na którym funkcja ma być rysowana, nie są jednak wykorzystywane w klasie Chart2D.

2.4.3.2 Dodawanie kolejnych funkcji do wykresu

Dla klasy Chart2D prawdopodobnie najważniejszą metodą jest metoda pozwalająca na dodawanie nowych funkcji do wykresu. Jak zostało omówione w podrozdziale 2.4.1 po procesie konstrukcji obiektu klasy Evaluator uzyskujemy delegat na dynamicznie komplikowaną funkcję. Optymalnym rozwiązaniem będzie więc przyjmowanie jako argument takiej metody m. in. właśnie delegata na funkcję. A w związku z tym, że klasa Chart2D odpowiada za wizualizację funkcji rzeczywistej jednej zmiennej, będzie to delegat Func<double,double>.

```
public void addFx(Func<double, double> fx, int N, string name)
{
    this.N = N;

    if (!Series.IsUniqueName(name)) //nothing new to add
        return;

    functions.Add(new Functions.BaseFunction<double>(fx, name));
    _addNewFunction();
}
```

Gdzie metoda _addNewFunction() najpierw tworzy nowy obiekt serii danych. Następnie wykonuje obliczenia punktów nowej serii danych wykresu z dokładnością N na przedziale od XMin do XMax i ostatecznie dodaje tę serię do wykresu. W metodzie tej warto też sprawdzać czy np. nie dodajemy drugi raz tej samej funkcji co jest bez sensu i tylko pochłonęłoby niepotrzebnie moc obliczeniową. Nie są też dodawane do wykresu „nieliczby” i nieskończoność dodatnia lub ujemna. Jak pisano w podrozdziale 2.4.1 nie należy dodawać wartości powyżej pewnych wielkości zdefiniowanych jako stałe pola.

```
private void _addNewFunction()
{
    var series =
new System.Windows.Forms.DataVisualization.Charting.Series()
```

```
{ ChartType=chartType };

    if (functions.Last().name != "")
        series.Name = functions.Last().name;

    double x, y, dx = (Math.Abs(XMin - XMax)) / N;

    for (int i = 0; i <= N; i++)
    {
        x = XMin + i * dx;
        y = functions.Last().eval(x);

        if (double.IsInfinity(x) || double.IsNaN(x) ||
double.IsInfinity(y) || double.IsNaN(y))
            continue;

        if (x > OVERFLOW_VALUE || y > OVERFLOW_VALUE ||
x < UNDERFLOW_VALUE || y < UNDERFLOW_VALUE)
            continue;

        series.Points.AddXY(x, y);
    }
    Series.Add(series);

    foreach (var s in this.Series)
        s.ToolTip = "x = #VALX\ny = #VALY";
    reloadChartSeriesComboBox();
}
```

Klasa Chart2D zawiera też wiele innych metod do dodawania np. samych punktów wraz z nazwą serii, ale nie zostały one tutaj opisane z racji tego, że skupiamy się na zastosowaniach naukowych i inżynierskich pisanej aplikacji oraz ich budowa jest naprawdę prosta.

2.4.4 Rysowanie wykresów

Wszystkie procedury związane z rysowaniem wykresu dzieją się w dziedziczonej i nie nadpisanej funkcji OnPaint(). W implementowanej klasie Chart2D nie zmieniono nic w procedurach rysowania wykresu - czyli całość odbywa się tak samo jak w Chart – ale zbiór punktów jest obliczany na podstawie funkcji matematycznych dodanych do wykresu przy każdej zmianie obszaru wykresu. Autor nie wyklucza natomiast nadpisania funkcji OnPaint w przyszłości i zmianę sposobu rysowania lub rysowanie nowych elementów (np. dorysowywanie asymptot funkcji).

2.4.5 Zmiana obszaru wykresu

Każda osoba, czy to naukowiec, czy student, nie wyobraża sobie dobrej aplikacji robiącej wykresy bez możliwości szybkiej i wygodnej zmiany obszaru wykresu, bo przecież często chcemy badać funkcję na różnych przedziałach.

Tak naprawdę problem ten rozwiązyano (i należy rozwiązać) na dwa sposoby, udostępniając użytkownikowi:

- Własności których numeryczna zmiana powoduje odświeżenie wykresu
- Zdarzeń (np. kółko myszki), które wpływają na numeryczną wartość obszaru wykresu

Dziedzicząc po klasie Chart napisano dodatkowo wygodne i prosto nazwane własności określające przedział który będzie pokazywał wykres.

```
public double XMin { get { return this.ChartAreas[0].AxisX.Minimum; }  
set { if (this.ChartAreas[0].AxisX.Minimum != value)  
{ this.ChartAreas[0].AxisX.Minimum = value; _refreshFunctions(); } } }  
  
public double XMax { get { return this.ChartAreas[0].AxisX.Maximum; }  
set { if (this.ChartAreas[0].AxisX.Maximum != value)  
{ this.ChartAreas[0].AxisX.Maximum = value; _refreshFunctions(); } } }  
  
public double YMin { get { return this.ChartAreas[0].AxisY.Minimum; }  
set { if (this.ChartAreas[0].AxisY.Minimum != value)  
{ this.ChartAreas[0].AxisY.Minimum = value; _refreshFunctions(); } } }  
  
public double YMax { get { return this.ChartAreas[0].AxisY.Maximum; }  
set { if (this.ChartAreas[0].AxisY.Maximum != value)  
{ this.ChartAreas[0].AxisY.Maximum = value; _refreshFunctions(); } } }
```

Widoczna jest skromna optymalizacja zawarta w tych „properties”, jeżeli ustawiana zostaje taka sama wartość jakiegoś przedziału, to nie ma sensu odświeżać wykresu, co kosztuje sporo czasu i mocy obliczeniowej. Jeżeli jednak rzeczywiście zmiana ma miejsce, to odświeżamy obszar wykresu metodą `_refreshFunctions()`.

```
private void _refreshFunctions()  
{  
    Series.Clear();  
    foreach (var fx in functions)  
    {  
        var series = new System.Windows.Forms.DataVisualization.Charting.Series();  
        chartType = fx.type;  
        if (fx.name != "")  
            series.Name = fx.name;  
  
        double x, y, dx = (Math.Abs(XMin - XMax)) / N;  
        for (int i = 0; i <= N; i++)  
        {
```

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

```
x = XMin + i * dx;
y = fx.eval(x);

    if (double.IsInfinity(x) || double.IsNaN(x) || double.IsInfinity(y) || double.IsNaN(y))
        continue;

    if (x > OVERFLOW_VALUE || y > OVERFLOW_VALUE || x < UNDERFLOW_VALUE || y < UNDERFLOW_VALUE)
        continue;

    series.Points.AddXY(x, y);
}
Series.Add(series);
}

foreach(var s in this.Series)
    s.ToolTip = "x = #VALX\ny = #VALY";

reloadChartSeriesComboBox();
}
```

Metoda ta, tak naprawdę bardzo przypomina metodę `_addNewFunction()` i w istocie prawidłowo, ponieważ robi właściwie to samo, tylko zamiast dodawać wartości punktów nowej funkcji usuwa wszystkie serie punktów i dodaje od nowa punkty wszystkich funkcji zawartych w kolekcji `functions`.

Drugim podejściem do zmiany obszaru wykresu jest dokonywanie tych zmian za pomocą zdarzeń. Bardzo popularne w podobnym oprogramowaniu jest udostępnianie użytkownikowi powiększania i pomniejszania obszaru wykresu za pomocą kółka myszki i tak tutaj również zostało to zaimplementowane.

```
private void _MouseWheel(object s, MouseEventArgs e)
{
    if (e.Delta > 0)
        zoomIn();
    else if (e.Delta < 0)
        zoomOut();
}
```

Jak widać nie było to zbyt problematyczne, żeby jednak udostępnić użytkownikowi więcej możliwości, należy umożliwić powiększanie/pomniejszanie tylko na jednej z wybranej osi. Zostało to zaimplementowane poprzez badanie wcisniętych dodatkowych klawiszy myszy.

```
void _MouseUp(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Right ||
e.Button == MouseButtons.XButton2) xOnlyZoomMode = false;

    if (e.Button == MouseButtons.XButton1) yOnlyZoomMode = false;
}
```

```
void _MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Right || 
e.Button == MouseButtons.XButton2) xOnlyZoomMode = true;

    if (e.Button == MouseButtons.XButton1) yOnlyZoomMode = true;
}
```

Żeby ukazać pełen obraz tego, jak to działa, należy jeszcze pokazać wewnętrzne funkcji zoomIn() i zoomOut()

```
public void zoomIn()
{
    bool any = false;
    if (Math.Abs(XMax - XMin) > 2 * scalingFactor && !yOnlyZoomMode)
    {
        this.ChartAreas[0].AxisX.Minimum += scalingFactor;
        this.ChartAreas[0].AxisX.Maximum -= scalingFactor;
        any = true;
    }
    if (Math.Abs(YMax - YMin) > 2 * scalingFactor && !xOnlyZoomMode)
    {
        this.ChartAreas[0].AxisY.Minimum += scalingFactor;
        this.ChartAreas[0].AxisY.Maximum -= scalingFactor;
        any = true;
    }
    if (any)
        _refreshFunctions();
}

public void zoomOut()
{
    if (!yOnlyZoomMode)
    {
        this.ChartAreas[0].AxisX.Minimum -= scalingFactor;
        this.ChartAreas[0].AxisX.Maximum += scalingFactor;
    }
    if (!xOnlyZoomMode)
    {
        this.ChartAreas[0].AxisY.Minimum -= scalingFactor;
        this.ChartAreas[0].AxisY.Maximum += scalingFactor;
    }
    _refreshFunctions();
}
```

Metody są trywialne, ogólnie, jak widać, jest to najprostsze w świecie skalowanie obszarów wykresu za pomocą kółka myszki i dodatkowych przycisków myszy jeżeli chcemy tylko przybliżać wybraną oś.

2.4.6 Obsługa zdarzeń w klasie Chart2D i autoskalowanie

Częściowo obsługa zdarzeń została już opisana w poprzednim podrozdziale. Dodatkowo jednak warto mieć w aplikacji autoskalowanie wykresu funkcji, jest to bardzo wygodne,

gdyż zamiast ręcznie ustalać dobry obszar, zrobi to za nas aplikacja dzięki tylnemu kliknięciu.

```
void _MouseDoubleClick(object sender, MouseEventArgs e)
{ if (e.Button == MouseButtons.Middle) autoScaleHard(); }

void _MouseClick(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Middle)
        autoScaleSmooth();
    if (e.Button == MouseButtons.Right || 
e.Button == MouseButtons.XButton2)
        xOnlyZoomMode = true;
}
```

Jak widać zróżnicowano pomiędzy kliknięciem pojedynczym kółka myszki, a podwójnym. Zrobiono tak, ponieważ zaimplementowano dwa rodzaje skalowania – skalowanie gładkie które ma na celu dobrze wyeksponować wartości funkcji na osiach i skalowanie ostre/twarde które po prostu eksponuje funkcję.

Skalowanie gładkie korzysta tak naprawdę z domyślnie zaimplementowanego w klasie bazowej skalowania.

```
private void autoScaleSmooth()
{
    this.ChartAreas[0].AxisY.Minimum = double.NaN;//min;
    this.ChartAreas[0].AxisY.Maximum = double.NaN; //max;
    ChartAreas[0].RecalculateAxesScale();//this.Update();
}
```

Inaczej sytuacja wygląda w przypadku skalowania ostrego. Tutaj zaimplementowano algorytm który znajduje największe i najmniejsze wartości na osi x i y i właśnie te wartości są ustawiane jako granice tych osi. Ma to niekorzystny wpływ na liczone na osiach etykiety liczbowe (uzyskujemy liczby z wieloma cyframi po przecinku), ale za to maksymalnie eksponuje funkcję na całym dostępnym dla wykresu obszarze.

```
private void autoScaleHard()
{
    double Xmax = double.MinValue, Xmin = double.MaxValue;
    double Ymax = double.MinValue, Ymin = double.MaxValue;

    foreach (var serie in this.Series)
    {
        var f = serie.Points.Max((p) => p.YValues.Max());
        if (f > Ymax)
            Ymax = f;

        f = serie.Points.Min((p) => p.YValues.Min());
        if (f < Ymin)
            Ymin = f;

        f = serie.Points.Max((p) => p.XValue);
```

```
    if (f > Xmax)
        Xmax = f;

    f = serie.Points.Min((p) => p.XValue);
    if (f < Xmin)
        Xmin = f; }
this.ChartAreas[0].AxisX.Minimum = Xmin;
this.ChartAreas[0].AxisX.Maximum = Xmax;

this.ChartAreas[0].AxisY.Minimum = Ymin;//min;
this.ChartAreas[0].AxisY.Maximum = Ymax; //max;
}
```

Jak widać miejscami sprytnie wykorzystano wyrażenia lambda tak, aby skrócić kod.

2.4.7 „Instalowanie” klasy Chart w innych kontrolkach

Żeby mieć jak największą kontrolę nad ustawieniami wykresu w aplikacji z graficznym interfejsem wymyślono, aby napisać metody które wypełniałyby obiekty odpowiednich kontrolek możliwymi wartościami ustawień wykresu.

```
public void initChart(ToolStripComboBox[] owners)
{
    setupChartTypes(owners[0]);
    setupChartSeries(owners[1]);
    setupChartColors(owners[2]);
    setupChartLegendPositions(owners[3]);
    setupChartLegendAlignments(owners[4]);
}
```

Wszystkie przekazywane jako tablica do tej metody kontrolki to ComboBox'y – bo są one wygodną metodą zmiany wielu właściwości wykresu.

Przykładowa metoda „instalująca” do ComboBox'a parametry klasy Chart2D wygląda następująco.

```
private void setupChartColors(ToolStripComboBox owner)
{
    Palette = System.Windows.Forms.DataVisualization.Charting.ChartColorPalette.Berry;
    List<ChartColorPalette> items =
        Enum.GetValues(typeof(
System.Windows.Forms.DataVisualization.Charting.ChartColorPalette))
            .Cast<System.Windows.Forms.DataVisualization.Charting.ChartColorPalette>()
            .ToList<System.Windows.Forms.DataVisualization.Charting.ChartColorPalette>();

    foreach (var v in items)
        owner.Items.Add(v.ToString());
```

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

```
        owner.DropDownStyle = ComboBoxStyle.DropDownList;
        owner.AutoSize = true;

        owner.SelectedItem = "BrightPastel";
    }
```

W takim ComboBox'ie należy jeszcze dodać obsługę zdarzenia OnSelectedIndexChanged, a w niej wywołać metodę, która służy do zmiany właściwego parametru wykresu po łańcuchu znaków z ComboBox'a.

```
public void changeChartColor(String chartColor)
{
    Palette = (System.Windows.Forms.DataVisualization.Charting
.ChartColorPalette)Enum.Parse(typeof(System.Windows.Forms.DataVisualization
.Charting.ChartColorPalette), chartColor);
}
```

2.5 Wykresy funkcji zmiennej zespolonej – klasa ComplexChart

W niniejszym podrozdziale dochodzi się właściwie do jednego z najważniejszych aspektów pracy. W aplikacji wspomagającej obliczenia na liczbach zespolonych, jedną z najważniejszych możliwości jest wizualizacja funkcji zmiennej zespolonej. Zagadnienie jest o wiele trudniejsze niż wykresy funkcji rzeczywistej jednej zmiennej.

Od strony implementacyjnej zrealizowano klasę ComplexChart jako kontrolkę dziedziczącą po klasie Control. Nie utworzono natomiast żadnej wspólnej klasy bazowej dla klas ComplexChart i Chart2D chociaż być może myślenie obiektywe, by coś takiego nakazywało. Jednak jak zostało napisane wcześniej klasa Chart2D dziedziczy już po klasie Chart a w języku C# nie ma wielodziedziczenia jak w C++.

..

Rysunek 2.19 Dziedziczenie klas Chart2D i ComplexChart

Alternatywną opcją była implementacja jakiegoś wspólnego interfejsu i wydaje to się bardzo poprawne na pozór, aczkolwiek w praktyce jednak klasy ComplexChart i Chart2D zachowują się inaczej. Jak się okaże w tym podrozdziale metody klasy ComplexChart są inne oraz działają inaczej niż metody klasy Chart2D. Również z pewnych przyczyn niektóre możliwości klasy Chart2D nie będą dostępne w klasie

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

ComplexChart i odwrotnie. Problem rysowania wykresu funkcji zespolonej jest po prostu bardzo odmienny w stosunku do rysowania wykresu funkcji rzeczywistej.

Rysunek 2.20 Zawartość klasy ComplexChart

2.5.1 Problem niewystarczającej liczby wymiarów

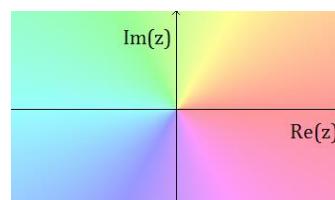
Wykres funkcji rzeczywistej jednej zmiennej $f : \mathbb{R} \rightarrow \mathbb{R}$ (takiej jak np. $f(x)=x^3$) może zostać przedstawiony za pomocą dwóch współrzędnych kartezjańskich (x,y) na płaszczyźnie dwuwymiarowej. Odwzorowanie $\mathbb{R} \rightarrow \mathbb{R}$ to odwzorowanie przestrzeni rzeczywistej (jednowymiarowej) w przestrzeń rzeczywistą (jednowymiarową), mamy więc dwa wymiary odwzorowania w sumie. Żyjąc w przestrzeni trójwymiarowej możemy z łatwością wykonać wizualizację takiej funkcji.

Wykres funkcji zespolonej jednej zmiennej $g : \mathbb{C} \rightarrow \mathbb{C}$ (takiej jak np. $g(z) = z^3+i$) to natomiast odwzorowanie $\mathbb{C} \rightarrow \mathbb{C}$ czyli przestrzeni zespolonej, która sama w sobie jest dwuwymiarowa ($\text{Re}(z)$, $\text{Im}(z)$) w zespoloną. Mamy więc cztery wymiary odwzorowania w sumie, żyąc w przestrzeni trójwymiarowej brakuje nam wymiarów do poprawnej wizualizacji [57].

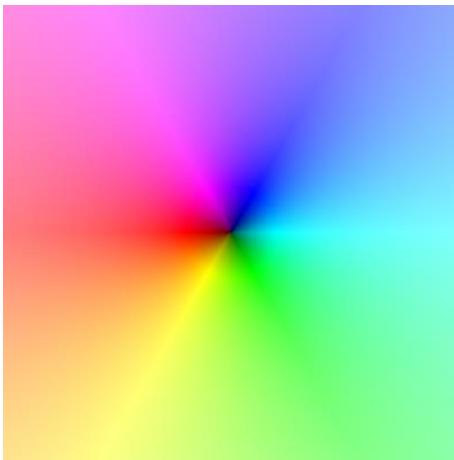
Problem ten rozwiązuje się w praktyce różnymi sposobami. Jednym z najprostszych rozwiązań jest reprezentacja dwóch wykresów trójwymiarowych (lub tzw. map ciepła). Jeden pokazuje zależność modułu funkcji od $\text{Im}(z)$ i $\text{Re}(z)$, a drugi pokazuje argument (fazę) funkcji zespolonej w zależności od $\text{Im}(z)$ i $\text{Re}(z)$. Takie rozwiązanie jest m. in. wykorzystywane w komercyjnym oprogramowaniu Mathematica [63]. W niniejszej pracy zdecydowanie zostało ono odrzucone, ponieważ jest nieoptymalne - wizualizacja musi być czytelna i nie może być nadmiarowa. Zebranie różnych metod takiej wizualizacji zostało przedstawione w [59].

2.5.2 Metoda kolorowanie domeny

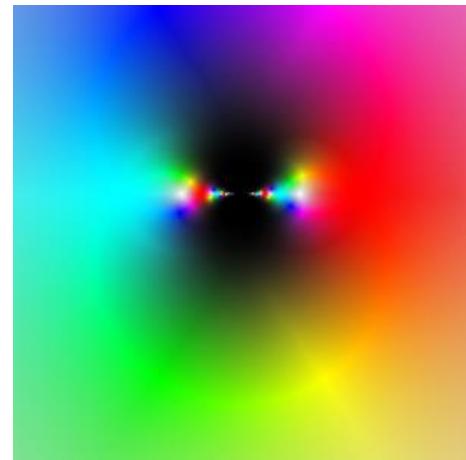
Rozwiązanie problemu niewystarczającej liczby wymiarów które przypadło szczególnie do gustu autorowi ze względu na czytelność i używanie zaledwie dwóch wymiarów, jest kolorowanie domeny [57]. Alternatywy są zdecydowanie trudniejsze w implementacji i/lub mniej czytelne jak przestrzenie Riemanna [58], obrazy konformalne [64] i wykres wektorowy [59]. Dla problemu wizualizacji funkcji zespolonej wykorzystuje się wykresy kolorowe wykresy kołowe [60]. Osie X i Y w takiej metodzie wizualizacji reprezentują odpowiednio $\text{Re}(z)$ i $\text{Im}(z)$.



Rysunek 2.21 Osie układu współrzędnych w metodzie kolorowania domeny



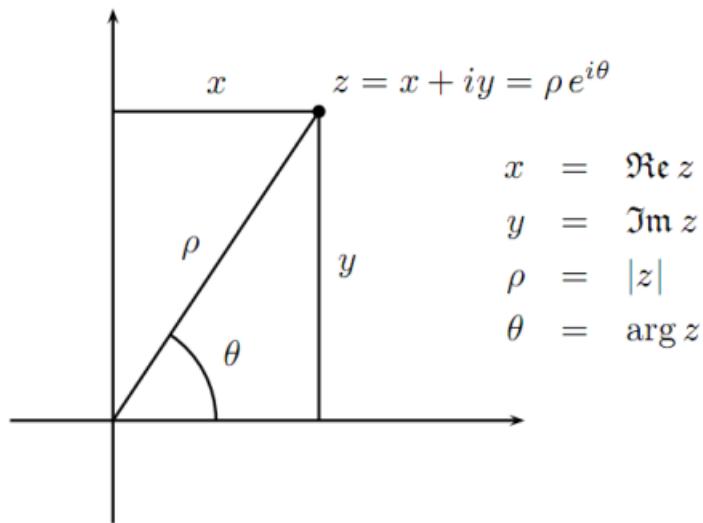
Rysunek 2.22 Kolorowanie domeny - funkcja $f(z)=z$, przyjmując, że im jaśniej tym większa wartość modulu [60]



Rysunek 2.23 Kolorowanie domeny - funkcja $f(z)=\sin(1/z)$ przyjmując, że im ciemniej tym większa wartość modulu [60]

Istnieją różne warianty metody kolorowania domeny, ale generalną zasadą jest wykorzystanie faktu identycznej okresowości barwy (hue) w modelu barwnym HSV i okresowości fazy (argumentu) zmiennej zespolonej. Aby stworzyć wykres funkcji zespolonej metodą kolorowania domeny należy przede wszystkich korzystać z biegunowej postaci zmiennej zespolonej

$$z = |z|e^{i\varphi}$$

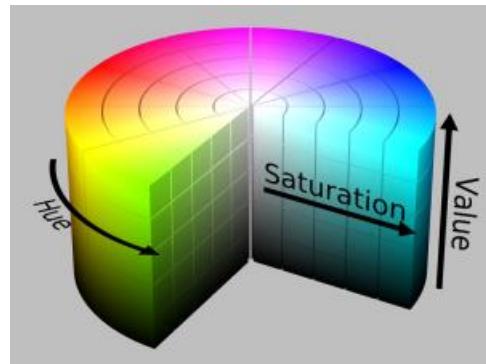


Rysunek 2.24 Reprezentacja biegunowa zmiennej zespolonej

Na wykresie wtedy wartość barwy (hue) jest bezpośrednio równa wartości fazy $\text{hue}=\varphi$. Wartość modułu natomiast jest kodowana za pomocą wartości (value) i nasycenia (saturation) w taki sposób, że zależnie od przyjętego modelu ciemniejsze obszary wykresu,

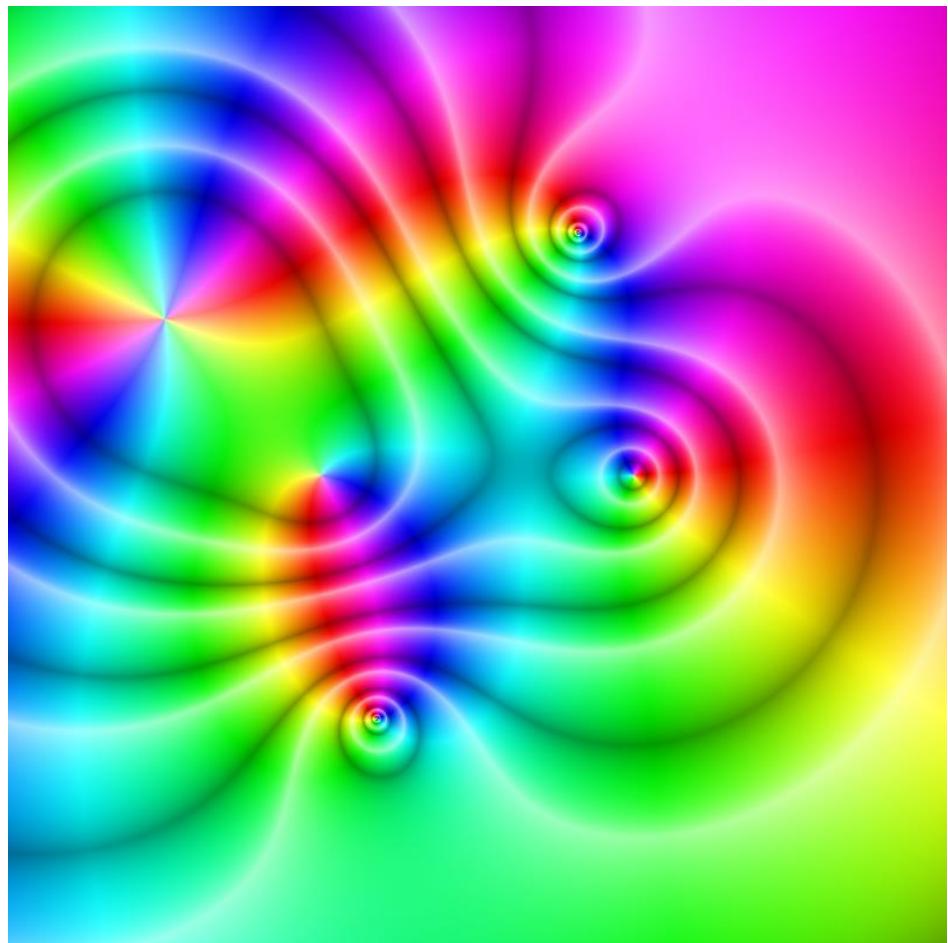
Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

to mniejsze wartości, a jaśniejsze większe lub odwrotnie [60]. albo stosuje się linie konturowe łączące punkty o tej samej wartości modułu jak w [61].



Rysunek 2.25 Model barwny HSV

W implementowanej aplikacji wybrano metodę z liniami konturowymi [61][62], ponieważ wydaje się najbardziej czytelna i ładna wizualnie.



Rysunek 2.26 Funkcja $f(z) = (z^2 - 1)(z - 2 - i)^2 / (z^2 + 2 + 2i)$ zobrazowana metodą kolorowania domeny z liniami konturowymi [65]

W aplikacji istnieje jednak możliwość zmiany metody na metodę z rysunku 2.21 lub 2.22 przez odpowiednie "properties" obecne w implementacji wykresu.

2.5.3 Reprezentacja funkcji matematycznych w klasie ComplexChart

Reprezentacja w sensie „trzymania” funkcji matematycznych zmiennej zespolonej wygląda inaczej w przypadku klasy ComplexChart niż w przypadku Chart2D. Przede wszystkim z racji wybranej metody kolorowania domeny nie da się rysować wielu funkcji na jednym wspólnym wykresie tak, jak można to robić w przypadku wykresu zmiennej rzeczywistej na płaszczyźnie. W związku z tym zamiast kolekcji funkcji mamy jedną funkcję.

```
private BaseFunction<Complex> function = null;
```

Nadal jednak jak widać mamy do czynienia z klasą szablonową BaseFunction opisaną wcześniej. Jedyna różnica jest taka, że teraz jest ona oczywiście od typu Complex.

Dodanie nowej funkcji do wykresu w przypadku klasy ComplexChart po prostu następuje starą i przebiega dosyć prosto.

```
public void addFx(Func<Complex, Complex> Fz, double x0, double xN,
double y0, double yN, string name)
{
    function = new BaseFunction<Complex>(Fz, new Complex(x0, y0),
new Complex(xN, yN), name);
    title = name;
    this.Invalidate();
}
```

W przypadku rysowanej kontrolki takiej jak ta należy też trzymać informacje o wartościach funkcji w punktach i kolorze każdego piksela.

```
private ComplexPoint[,] pointsValues;
private Color[,] pointsColors;
```

Wartości te i kolory są obliczane w metodzie calculateValuesAndColors(), cały proces jest opisany w rozdziale 2.5.5

Warto jeszcze przybliżyć zawartość klasy ComplexPoint

```
public ComplexPoint(Complex z, Complex fz)
{
    this.Z = z;
    this.Fz = fz;
}

public Complex Z { get; set; }
public Complex Fz { get; set; }
}
```

Jak widać klasa jest dokładnie tym, czego można by się spodziewać po nazwie, mamy więc współrzędne punktu w przestrzeni zespolonej i wartość funkcji zespolonej w tym punkcie.

2.5.4 Właściwości klasy ComplexChart

Tak samo jak w przypadku klasy Chart2D tak i tutaj mamy do czynienia z wieloma „properties”, które mają umożliwić maksymalne dostosowanie wykresu do potrzeb użytkownika.

```
public double countourLinesStep { get; set; }
public CountourLinesMode countourMode { get; set; }
public AssignmentOfColorMethod colorAssignmentMethod { get; set; }

        public double XMin {
get { if (function != null) return function.argmin.Real; else return -5; }
set { if (function != null) function.argmin =
new Complex(value, function.argmin.Imaginary); } }

        public double XMax {
get { if (function != null) return function.argmax.Real; else return 5; }
set { if (function != null) function.argmax = new Complex(value, function.argmax.Imaginary); } }
        public double YMin {
get { if (function != null) return function.argmin.Imaginary; else return -3; }
set { if (function != null) function.argmin =
new Complex(function.argmin.Real, value); } }

        public double YMax {
get { if (function != null) return function.argmax.Imaginary; else return 3; }
set { if (function != null) function.argmax =
new Complex(function.argmax.Real, value); } }

        public string xLabel { get; set; }
        public string yLabel { get; set; }

        public Font titleFont { get; set; }
        public Font labelsFont { get; set; }
        public Color labelsColor { get; set; }
        public Color titleColor { get; set; }
        public Color axesColor { get; set; }
        public double axisArrowRelativeSize {get;set;}
        public string title { get; set; }
        public bool shouldDrawAxes { get; set; }
```

Niektóre właściwości przypominają te obecne w klasie wykresu rzeczywistego, tak jak dotyczące obszaru rysowania wykresu (XMin,XMax), (YMin,YMax). Inne, typu string, Color i Font dotyczą wyglądu elementów wykresu takich jak napisy, etykiety i osi. Właściwość countourLinesStep dotyczy natomiast rysowania samej wizualizacji. Jest to krok oddzielający kolejne linie konturowe. Może on oddzielać je jednak w różnych trybach obliczeniowych, wybrany tryb kryje się pod zmienna countourMode i bierze istotną rolę w algorytmie obliczania kolejnych linii konturowych.

```
enum CountourLinesMode
{
    NoCountourLines, Logarithmic, Linear, Exponential
}
```

W przypadku gdy wybrany jest tryb „NoCountourLines” w algorytmie rysowania zaczyna odgrywać rolę zmienna colorAssignmentMethod, która jest tak naprawdę jedną z dwóch opcji rysowania wykresu metodą kolorowania domeny bez linii konturowych, jak widać na rysunkach 2.21 i 2.22.

```
enum AssignmentOfColorMethod
{
    GreaterIsDarker, //the origin is white, 1 is red, -1 is cyan,
    //and a point at the infinity is black.

    GreaterIsLighter //the origin is black, 1 is red, -1 is cyan,
    //and a point at the infinity is white.
}
```

2.5.5 Obliczenia wartości funkcji i kolorów

Kluczowym zagadnieniem dla wykresu funkcji zespolonej metodą kolorowania domeny, obok rysowania samego wykresu, jest obliczanie wartości funkcji w konkretnych punktach wykresu, a na ich podstawie wyznaczanie koloru w modelu barwnym HSV (czyli wyznaczanie barwy, wartości i nasycenia).

Jako, że założeniem niniejszej pracy było napisanie dokładnej aplikacji, zdecydowano się na liczenie każdego piksela wykresu po kolej, co jest najbardziej dokładne jak się da, ale też bardzo kosztowne pod względem czasu obliczeń. Metoda obliczająca calculateValuesAndColors() rzutuje więc obszar, na którym ma być zobrazowana funkcja na obszar rzeczywisty wykresu. Następnie oblicza dla każdego piksela wartość zmiennej z (na podstawie obszaru funkcji który użytkownik chce wyrysować) w tym punkcie i wartość funkcji $f(z)$ w tym punkcie, zapisuje to w dwuwymiarową tablicę pointsValues indeksowaną współrzędnymi x i y piksela. Po obliczeniu wartości na podstawie tejże, zastosowany w tej pracy algorytm kolorowania domeny, oblicza kolor w tym punkcie i ten kolor zapisywany jest w dwuwymiarowej tablicy pointsColors indeksowanej współrzędnymi piksela.

Niestety we wczesnych wersjach aplikacji ten proces trwał zbyt długo. Liczenie każdego piksela po kolej nawet przy mało złożonej funkcji typu $f(z)=z$ trwało zdecydowanie zbyt długo i jeszcze się wydłużało na urządzeniach z wysoką rozdzielczością. Na pozór wydawało się, że nie ma możliwości optymalizacji tego procesu bez straty dokładności tworzonych wykresów. Po głębszej analizie okazało się jednak, że jak najbardziej jest taka możliwość – skoro każdy piksel jest liczony niezależnie od pozostałych, całą operację można zrównoleglić używając wielu wątków procesora. Na pomoc wychodzi tutaj biblioteka TaskParallelLibrary (TPL) która pojawiając się w .NET Framework 4.0 w

końcu sprawiła, że obliczenia równoległe stały się proste. Wystarczyło zamienić najbardziej zewnętrzna pętlę for na funkcję Parallel.For, aby uzyskać zależnie od ilości rdzeni procesora wielokrotne przyspieszenie procesu obliczeniowego. Po tej optymalizacji metoda calculateValuesAndColors zyskała poniższą implementację.

```
private void calculateValuesAndColors()
{
    Parallel.For(0, Width, (x) =>
    {
        double re = function.argmin.Real +
x * (function.argmax.Real - function.argmin.Real) / Width;
        for (int y = 0; y < Height; y++)
        {
            double im = function.argmax.Imaginary -
y * (function.argmax.Imaginary - function.argmin.Imaginary) / Height;

            Complex z = new Complex(re, im);

            Complex fz = function.eval(z);
            pointsValues[x, y] = new ComplexPoint(z, fz);

            if (Double.IsInfinity(fz.Real) || Double.IsNaN(fz.Real) ||
Double.IsInfinity(fz.Imaginary) ||
                Double.IsNaN(fz.Imaginary))
            { }
            else
            {
                pointsColors[x, y] = ComplexToColor(fz);
                //image.SetPixel(x, y, ComplexToColor(fz));
            }
        });
    });
}
```

Wydawać by się mogło, że można by zoptymalizować tę metodę jeszcze bardziej. Nie zostało wspomniane, że to co widzi użytkownik jako kontrolkę, to wyrysowany obrazek kryjący się pod zmienną image. Należy zwrócić teraz uwagę na zakomentowaną linię w kodzie powyżej. O wiele optymalniej byłoby wykonywać linię zakomentowaną zamiast tej wyższej – od razu dodawać piksele do obrazka, który potem rysujemy, niepotrzebny wtedy jest pośrednik w postaci tablicy pointsColors. Otóż okazuje się jednak, że jest to niemożliwe, gdyż metoda SetPixel na zmiennej image typu BitMap nie dopuszcza równoległego wykonywania i co ciekawe dotyczy to wielu metod związanych z rysowaniem. Mając więc linię image.SetPixel wewnętrz Parallel.For uzyskujemy wprost wyjątk informujący o błędzie. Dlatego też niezbędny jest pośrednik w postaci pointsColors.

Ostatnią i z punktu widzenia poprawnej wizualizacji, najważniejszą rzeczą jest to, co dzieje się w funkcji ComplexToColor. Jak można się domyśleć właśnie w niej wykonuje się mapowanie z zmiennej typu Complex na kolor, który jest rysowany.

```

private Color ComplexToColor(Complex z)
{
    double m = z.Magnitude, t = z.Phase;

    while (t < 0.0) t += 2 * Math.PI;
    while (t >= 2 * Math.PI) t -= 2 * Math.PI;

    double h = t / (2 * Math.PI), s = 0, v = 0;
    double r0 = 0, r1 = 1;

    switch (countourMode)
    {
        case CountourLinesMode.Logarithmic:
            //Based on Claudio Rocchini C++ algorithm for complex functions
domain coloring http://en.wikipedia.org/wiki/File:Color_complex_plot.jpg
            while (m > r1)
            {
                r0 = r1;
                r1 = r1 * countourLinesStep;
            }
            break;

        case CountourLinesMode.Linear:
            r1 = countourLinesStep;
            while (m > r1)
            {
                r0 = r1;
                r1 = r1 + countourLinesStep;
            }
            break;
            //.....
    }

    if (countourMode != CountourLinesMode.NoCountourLines)
    {
        //Based on Claudio Rocchini C++ algorithm for complex functions
domain coloring http://en.wikipedia.org/wiki/File:Color_complex_plot.jpg
        double r = (m - r0) / (r1 - r0);
        double p = r < 0.5 ? 2.0 * r : 2.0 * (1.0 - r);
        double q = 1.0 - p;
        double p1 = 1 - q * q * q;
        double q1 = 1 - p * p * p;
        s = 0.4 + 0.6 * p1;
        v = 0.6 + 0.4 * q1;
    }
    else
    {
        //.....
    }

    return ColorFromHSV(h, s, v);
}

```

Funkcja bazuje na algorytmie po raz pierwszy wymyślonym przez wikipedystę Claudio Rocchini [61] opisany i zaimplementowanym również w artykule [62]. Algorytm wyróżnia się tym, że używa wartości (value) i nasycenia (saturation) z modelu barwnego HSV do zaznaczania linii konturowych łączących punkty o tej samej wartości modułu funkcji zespolonej w taki sposób, że dobrze zobrazowana jest odległość (w sensie liczbowym) innych punktów od linii konturowej. Linie konturowe są niemal białe, im dalej od nich tym ciemniej, a dokładnie w połowie wybranej skali (np. logarytmicznej) pomiędzy liniami konturowymi pojawia się dodatkowa linia konturowa – barwy niemal czarnej – patrz rysunek 2.25.

Autorska wersja tego algorytmu wprowadza między innymi inne tryby kładzenia linii konturowych, np. zamiast kłaść je logarytmicznie jak w oryginalnym algorytmie (linie konturowe w $e^0, e^1, e^2, e^3, \dots$) można wybrać opcję kładzenia liniowego (linie konturowe w $0e, 1e, 2e, 3e, \dots$) lub jeszcze inne dostępne opcje. Także dzięki implementacji właściwości countourLinesStep krokiem między liniami konturowymi nie musi być koniecznie liczba Eulera, a może być dowolna inna liczba. Dodatkowo algorytm autorski przewiduje też w przypadku wybrania opcji bez linii konturowych rysowanie, jak w tradycyjnym kolorowaniu domeny – patrz rysunki 2.21, 2.22.

Do wyznaczenia dla liczby zespolonej koloru w systemie HSV najpierw uzyskujemy barwę (hue). Robimy to normalizując wartość fazy do przedziału $[0, 2\pi]$, następnie dzielimy ją przez 2π i otrzymujemy wartość barwy (hue) $[0, 1]$. W kolejnym kroku zależnie od wybranego trybu i kroku obliczamy, jak blisko linii konturowej znajduje się wartość modułu. Używając metody obliczeniowej z algorytmu Claudio Rocchini dzięki informacji o odległościach od mniejszej i większej linii konturowej wyznaczamy wartość s i v. Potem mając już wartości w systemie barwnym HSV wywołujemy tylko funkcję konwertującą z systemu barwnego HSV na RGB i zwracamy obliczony kolor.

Ze względu na ograniczone miejsce w zamieszczonym powyżej algorytmie pominięto niektóre fragmenty, zaznaczone to jako ”//....”.

2.5.6 Rysowanie wykresu

W poprzednim punkcie podrozdziału napisano jak oblicza się wartość koloru dla każdego z pikseli. Naturalnym kolejnym krokiem po obliczeniu koloru wszystkich pikseli wykresu jest ich narysowanie. Procedura przebiega dosyć standardowo, jako, że klasa ComplexChart dziedziczy po kontrolce Control należy nadpisać funkcję OnPaint i w niej zawrzeć wszystkie procedury rysujące wykres.

```
protected override void OnPaint(PaintEventArgs pe)
{
    if (function != null)
    {
        calculateValuesAndColors();
        for (int x = 0; x < Width; x++)
            for (int y = 0; y < Height; y++)
                image.SetPixel(x, y, pointsColors[x, y]);
        drawn = true;
        pe.Graphics.DrawImage(image, 0, 0);
    }
    else
        drawn = false;
    if (shouldDrawAxes)
        drawAxes(pe);
}
```

Jak widać procedura OnPaint wygląda bardzo standardowo. Po obliczeniu wartości kolorów wszystkich pikseli jedyne, co należy teraz zrobić, to nanieść te wartości na bitmapę, którą potem rysuje się na powierzchni kontrolki.

Dodatkowo jednak, jeżeli taka opcja jest pożądana przez użytkownika, są rysowane osie wykresu w funkcji drawAxes().

```
private void drawAxes(PaintEventArgs pe)
{
    using (System.Drawing.Pen myPen = new System.Drawing.Pen(axesColor))
    {
        Point middlePoint = getMiddlePoint();

        Point xEnd = new Point(Width, middlePoint.Y);
        Point yEnd = new Point(middlePoint.X, Height);

        Point xEnd1Axis = new Point((int)(Width * (1 - axisArrowRelativeSize)), (int)(middlePoint.Y * (1 - axisArrowRelativeSize)));
        Point xEnd2Axis = new Point((int)(Width * (1 - axisArrowRelativeSize)), (int)(middlePoint.Y * (1 + axisArrowRelativeSize)));

        Point yEnd1Axis = new Point((int)(middlePoint.X * (1 - axisArrowRelativeSize)), (int)(Height * axisArrowRelativeSize));
        Point yEnd2Axis = new Point((int)(middlePoint.X * (1 + axisArrowRelativeSize)), (int)(Height * axisArrowRelativeSize));

        Point xStart = new Point(0, middlePoint.Y);
        Point yStart = new Point(middlePoint.X, 0);
```

```
//Y axis (ImZ)
pe.Graphics.DrawLine(myPen, middlePoint, yEnd);
pe.Graphics.DrawLine(myPen, middlePoint, yStart);
pe.Graphics.DrawLine(myPen, yStart, yEnd1Axis);
pe.Graphics.DrawLine(myPen, yStart, yEnd2Axis);
pe.Graphics.DrawString(yLabel, labelsFont,
new SolidBrush(labelsColor), middlePoint.X - 60, 15);

//X axis(ReZ)
pe.Graphics.DrawLine(myPen, middlePoint, xEnd);
pe.Graphics.DrawLine(myPen, middlePoint, xStart);
pe.Graphics.DrawLine(myPen, xEnd, xEnd1Axis);
pe.Graphics.DrawLine(myPen, xEnd, xEnd2Axis);
pe.Graphics.DrawString(xLabel, labelsFont,
new SolidBrush(labelsColor), Width - 60, middlePoint.Y + 10);
    }
}
```

Tak naprawdę nie ma tutaj nic niezwykłego – rysujemy osie układu współrzędnych i ich etykiety. Jedynym ciekawym zagadnieniem może być procedura znajdowania środka układu współrzędnych, która jest obsługiwana przez funkcję getMiddlePoint().

```
private Point getMiddlePoint()
{
    int x = (int)((Math.Abs(XMin) / (XMax - XMin)) * Width);
    int y = (int)((Math.Abs(YMax) / (YMax - YMin)) * Height);

    if (XMax <= 0)//we have only negative numbers for x
        x = Width - 1;
    else if (XMin >= 0)//we have only positive numbers for x
        x = 0;

    if (YMax <= 0)
        y = 0;
    else if (YMin >= 0)
        y = Height - 1;

    return new Point(x, y);
}
```

W najprostszym przypadku punkt środka układu współrzędnych, to oczywiście tam, gdzie znajduje się punkt (0,0). Jednak czasami może się zdarzyć, że użytkownik będzie chciał wyrysować sobie obszar funkcji gdzie nie ma punktu (0,0) – np. chce zobaczyć jak funkcja zachowuje się dla dużych dodatnich wartości $\operatorname{Im}(z)$ i $\operatorname{Re}(z)$. W takim bądź razie w naiwnym podejściu nie pojawiłyby mu się osie układu współrzędnych – a przecież mogą one pomóc zwiększyć czytelność wykresu. Funkcja getMiddlePoint() zwraca punkt w którym mają się skrzyżować osie $\operatorname{Re}(z)$ i $\operatorname{Im}(z)$ uwzględniając ten fakt. Przykładowo dla dużych dodatnich wartości obszarów $\operatorname{Re}(z)$ i $\operatorname{Im}(z)$ osie skrzyżują się po prostu w lewym dolnym rogu wykresu – wygląda to estetycznie i jest poprawne.

2.5.7 Obsługa zdarzeń w klasie ComplexChart

W przypadku tej kontrolki nie można sobie pozwolić na tak wygodną i dynamiczną zmianę obszaru wykresu, za pomocą kółka myszy lub ustawianych właściwości jak w kontrolce Chart2D. Wynika to z faktu, że obliczanie wszystkich pikseli po kolej mimo podjętych optymalizacji trwa stosunkowo długo. Nie oznacza to jednak, że nie korzysta się tutaj z obsługi zdarzeń, gdyż znalazło się nawet istotniejsze zadanie do obsługi. Otóż na wykresie rzeczywistym wszystkie liczby widać dzięki osiom X, Y. Tutaj natomiast widać jedynie kolory i linie konturowe – a nie widać wartości modułu i fazy funkcji (lub części rzeczywistej i urojonej funkcji) explicite. W celu ich pokazania obsługuje się zdarzenie MouseClick i używając ToolTip'a wyświetla brakującą informację.

```
private void _MouseClick(object s, MouseEventArgs e)
{
    if (drawn)
    {
        Complex fz = pointsValues[e.Location.X, e.Location.Y].Fz;
        Complex z = pointsValues[e.Location.X, e.Location.Y].Z;

        toolTip.SetToolTip(this, String.Format(
            "f(z) = {0:0.###}{1:+0.###;-0.###}i = {2:0.###} exp({3:0.###})\n"
            "z = {4:0.###}{5:+0.###;-#0.###}i = {6:0.###} exp({7:0.###})",
            fz.Real, fz.Imaginary,
            fz.Magnitude, fz.Phase,
            z.Real, z.Imaginary,
            z.Magnitude, z.Phase));
        toolTip.ShowAlways = true;
    }
}
```

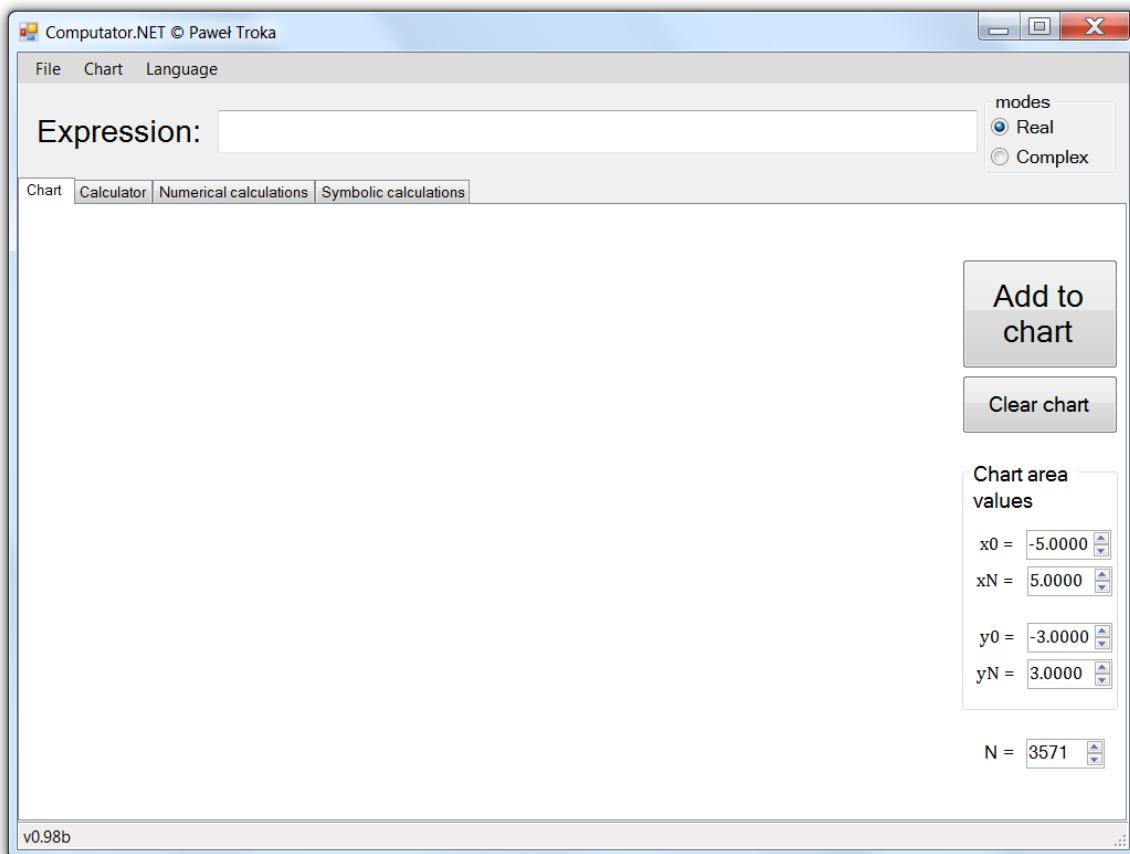
Odpowiednie sformatowanie wyświetlonej informacji jest kluczowe, aby była ona czytelna. Osiągnięto je za pomocą wywołania metody String.Format [66].

3 Efekty i testy

Aplikacja zostanie w niniejszym rozdziale przedstawiona z punktu widzenia potencjalnego użytkownika. W kolejnych podrozdziałach będą rozpatrywane jej kolejne aspekty, a wewnątrz tych podrozdziałów będzie opis i demonstracja możliwości aplikacji z danego aspektu.

3.1 Interfejs aplikacji

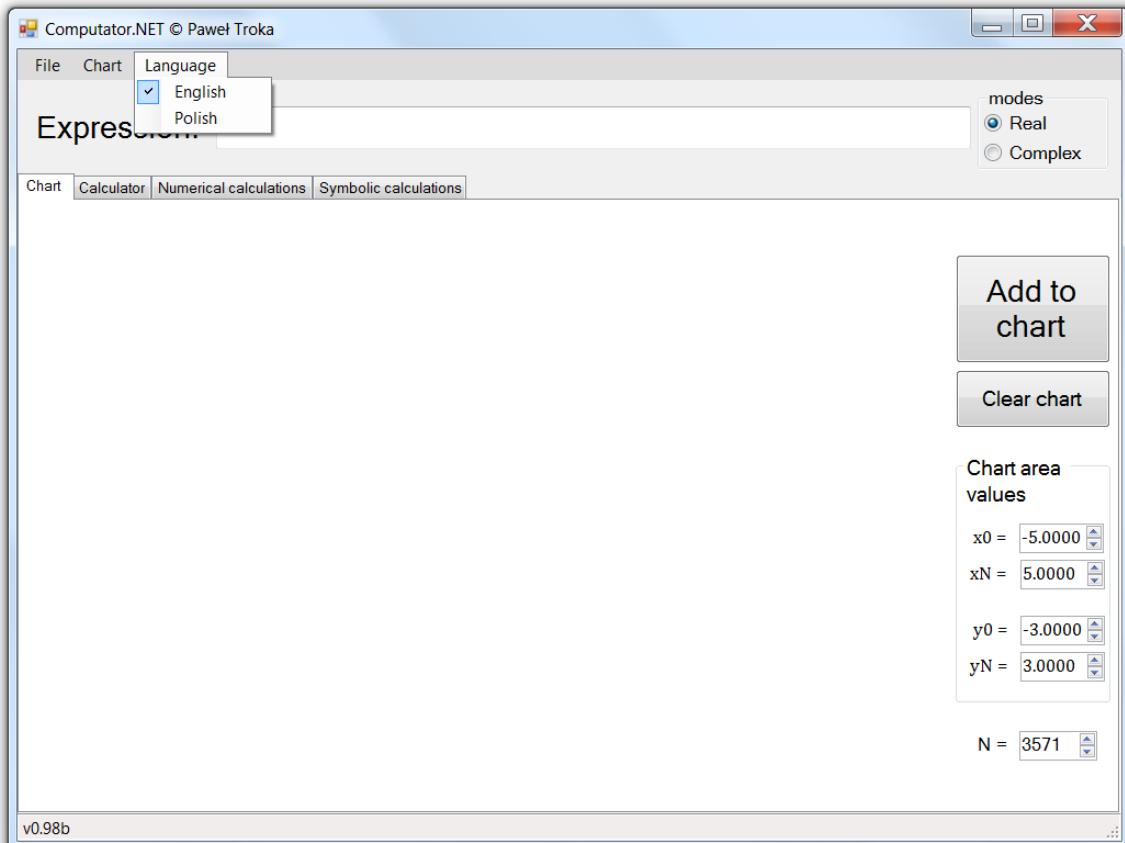
Jednym z celów pracy było stworzenie możliwie najbardziej przejrzystego i wygodnego interfejsu. Wydaje się, że cel się został osiągnięty, aplikacja jest wygodna w użytkowaniu.



Rysunek 3.1 Aplikacja tuż po uruchomieniu

Tuż po uruchomieniu od razu rzuca się w oczy duże pole „Expression” wspólne dla wszystkich modułów (Chart, Calculator, Numerical calculations, Symbolic calculations) aplikacji. Modułem, który jest domyślnie wyświetlany jako pierwszy jest moduł Chart. Od razu widać też duży, czytelny napis „Add to chart”. Górnego menu, również wspólne dla

modułów, jest dosyć standardowe i zawiera podstawowe opcje dla zmiany ustawień wykresu, wczytywanie plików i zmianę języka aplikacji.



Rysunek 3.2 Zmiana języka aplikacji

Wspólne dla wszystkich modułów jest również ustawienie trybu działania aplikacji (zbiór liczb zespolonych bądź rzeczywistych) (Real, Complex). Po przemyśleniu tego elementu interfejsu, autor stwierdza, że jest to najgorszy i najmniej wygodny element aplikacji, ponieważ dużo lepiej byłoby, gdyby aplikacja po wprowadzonym wyrażeniu sama ustalała tryb. Było to jednak rozwiązanie proste i bezpieczne, więc usprawiedliewia się w pewien sposób.

3.2 ExpressionTextBox

Duże pole „Expression”, które od razu rzuca się w oczy po uruchomieniu będąc wspólne dla wszystkich modułów aplikacji, jest jednym z najważniejszych elementów interfejsu użytkownika. To tutaj będzie wpisywane każde wyrażenie, każde działanie i każda funkcja na której jakąś operację chce wykonać użytkownik. Dlatego autorowi zależało aby ten element był maksymalnie użyteczny i dopracowany. Też z tego powodu nie mógł być to zwykły TextBox, należało dziedziczyć po elemencie TextBox i rozwinąć go o nowe funkcje i właściwości tak aby zmaksymalizować „user experience”.

Rysunek 3.3 Zawartość klasy ExpressionTextBox

3.2.1 Estetyczność wpisywanych formuł

Pierwszym celem było sprawienie, aby formuły wpisywane w ExpressionTextBox wyglądały trochę bardziej matematycznie, a mniej jak zwykły tekst – w końcu użytkownik wpisuje formuły matematyczne.

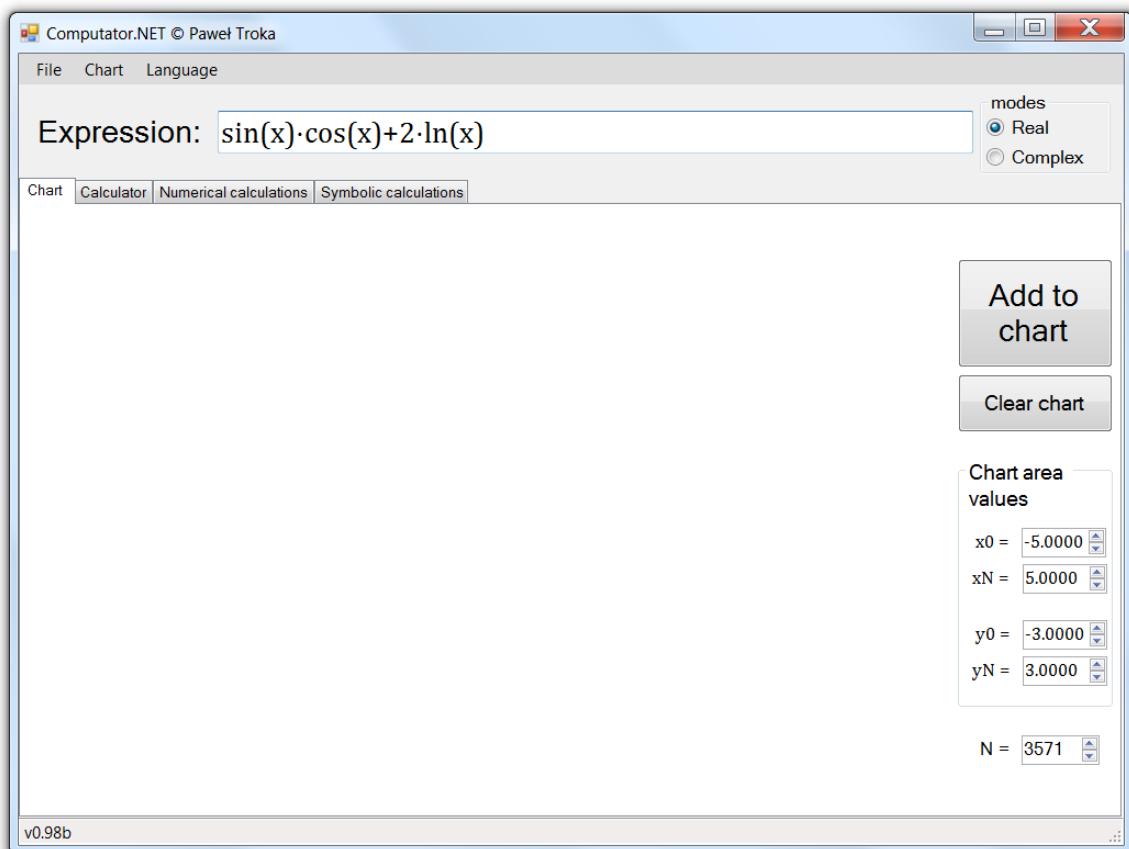
Ciekawą możliwością było zastosowanie czcionki autorstwa Microsoftu „Cambria Math”, która jest wykorzystywana m.in. przez pakiet Microsoft Office do wyświetlania wzorów matematycznych. Niestety z powodu błędów w Microsoft’owej implementacji tego fontu nie jest możliwe normalne wykorzystanie go poza Office’em bez uzyskiwania otaczających każdą literkę dużych pustych przestrzeni [67]. Dosyć podobną czcionką jest Cambria – więc ją wybrano w zastępstwie.

Kolejną koniecznością w celu zwiększenia estetyki wpisywanych formuł było posiadanie ładnych operatorów matematycznych. W zasadzie wszystkie operatory są w porządku, oprócz jednego – znak gwiazdki '*' jako znak mnożenia wygląda brzydko, a jednak użytkownicy są przyzwyczajeni do jego używania. Postanowiono więc przy wyświetlaniu podmieniać znak '*' na prawdziwy znak mnożenia z standardu unicode '..'. Do celów dynamicznej kompilacji wyrażeń potrzebujemy oczywiście znak gwiazdki, ponieważ taki jest przyjęty w językach programowania, zamieniamy więc znak mnożenia na gwiazdkę przy wysyłaniu wyrażenia do kompilacji.

```
private void ExpressionTextBox_KeyPress(object s, KeyPressEventArgs e)
{
    if (e.KeyChar == '*')
        e.KeyChar = '.';
}

public override string Text {
get { return base.Text.Replace("*", ".."); }
set { base.Text = value.Replace("*", ".."); } }

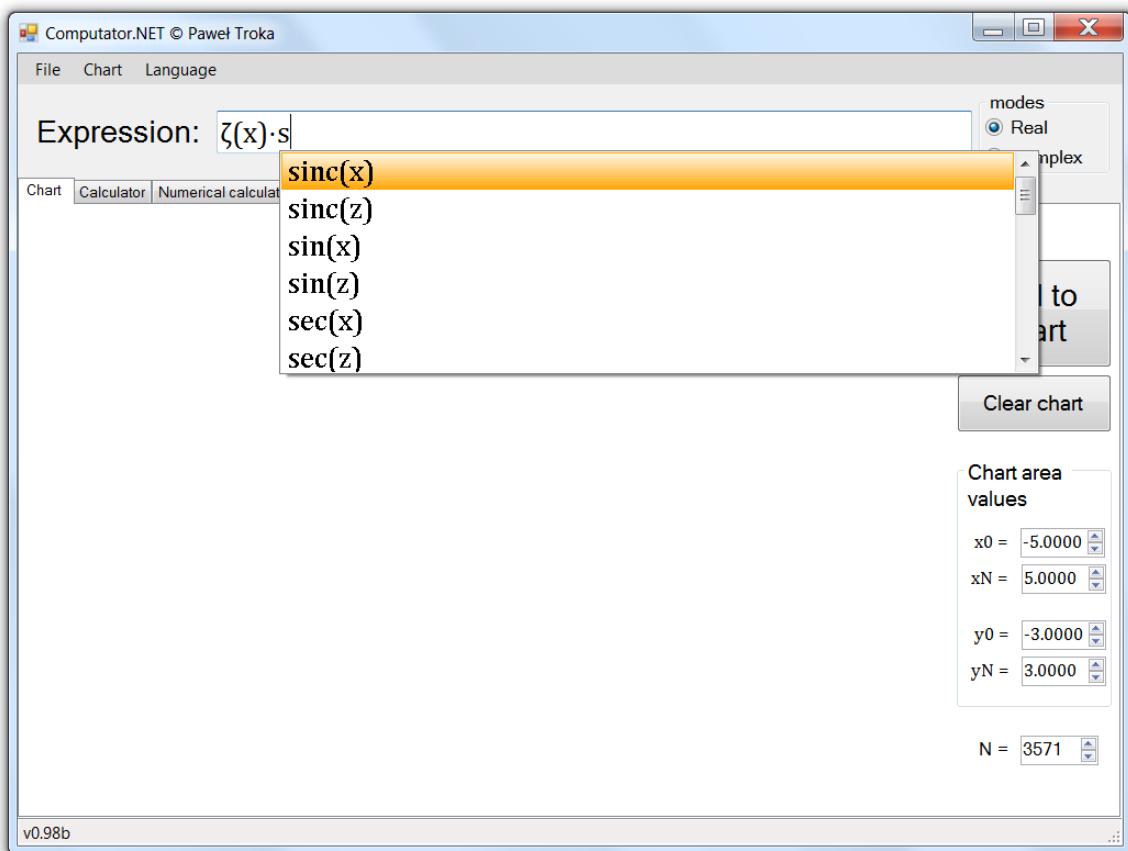
public string Expression { get { return base.Text.Replace(".", "*"); } }
```



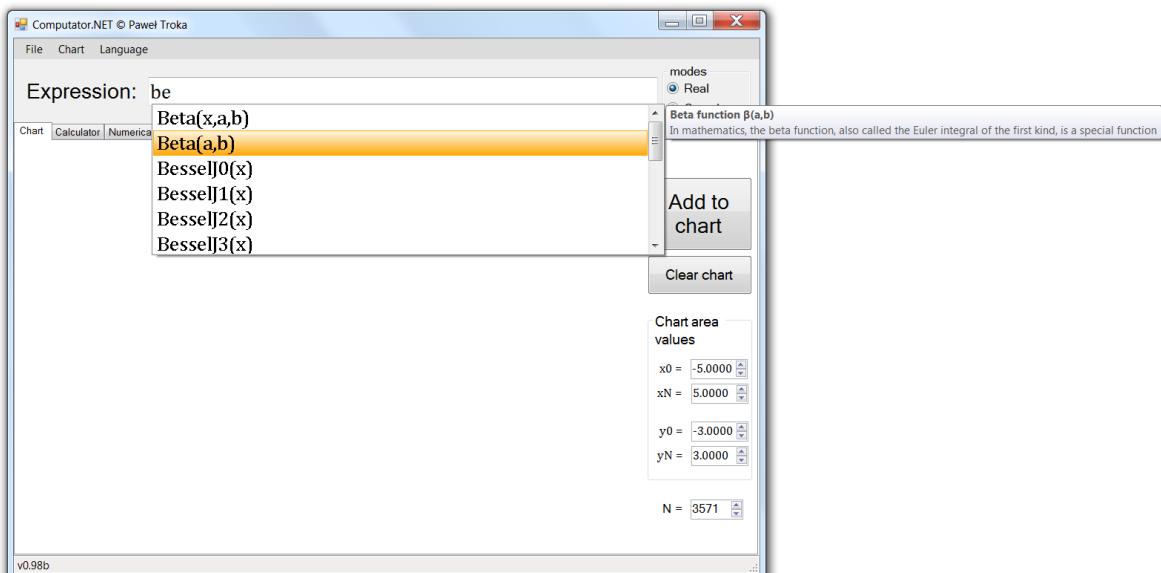
Rysunek 3.4 Wygląd wpisywanych formuł

3.2.2 Podpowiedzi do wpisywanych formuł

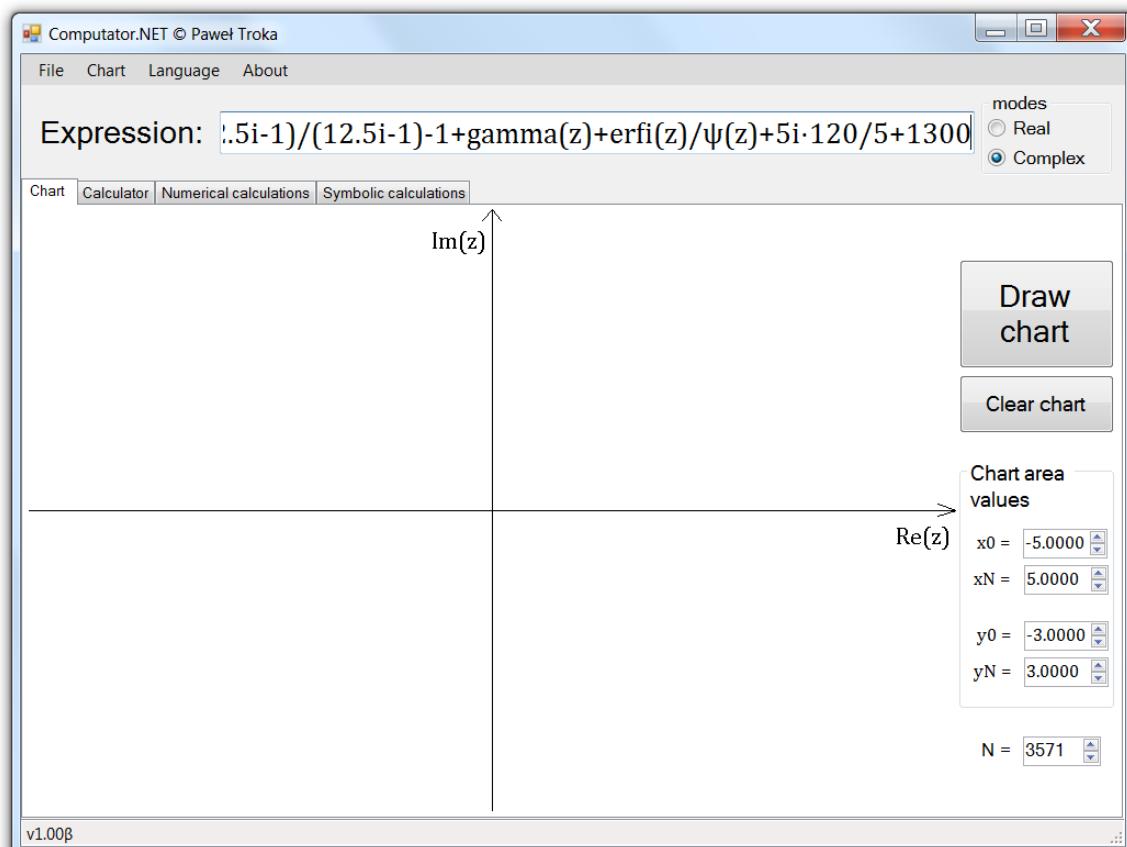
W przypadku dużej liczby zaimplementowanych funkcji tak jak w niniejszej pracy samo pole do wpisywania wyrażeń i funkcji to trochę za mało. Użytkownik tak samo jak programista oczekuje podpowiedzi składni jak coś pisze. Rzecz nie jest jakoś szczególnie łatwa w implementacji dlatego nie była uznawana za priorytet. Sytuacja się zmieniła kiedy autor natrafił w internecie na nagrodzony wyróżnieniem „Best C# article of April 2012” artykuł na stronie codeproject.com opisujący implementację klasy AutocompleteMenu rozwiązującej dosyć dobrze podany problem [68]. Biblioteka w formie pliku dll została dołączona do projektu a ExpressionTextBox wzbogacił się o nowe możliwości. Odpowiednia inicjalizacja i ustawienie obiektu autocompleteMenu i przede wszystkim wpisanie nazw funkcji i stałych zaimplementowanych w aplikacji było czasochłonne, ale efekt końcowy jest zadowalający.



Rysunek 3.5 Podpowiedzi nazw funkcji w programie



Rysunek 3.6 Krótkie opisy funkcji w programie

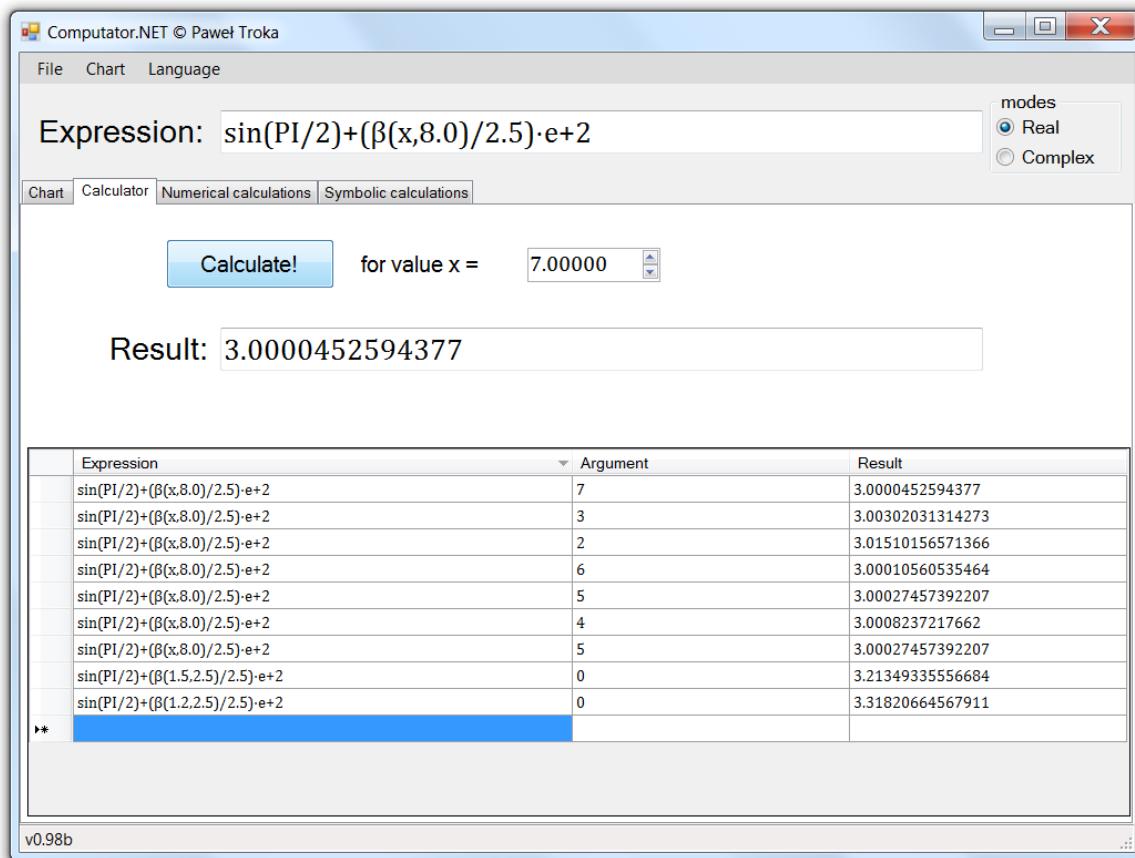


Rysunek 3.7 ExpressionTextBox umożliwia wpisywanie dowolnie długich wyrażeń

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

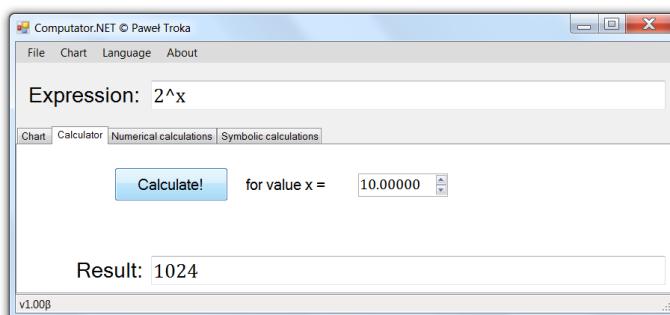
3.3 Kalkulator

Realizacja modułu obliczeniowego nazywanego kalkulatorem była jednym z najprostszym celów niniejszej pracy. Dynamiczna komplikacja sprawiła, że zadanie to było wyjątkowo łatwe. Odpowiednie przetwarzanie wyrażeń również pomogło w realizacji wygodnego kalkulatora nieograniczonego długością wyrażenia czy brakiem funkcji.



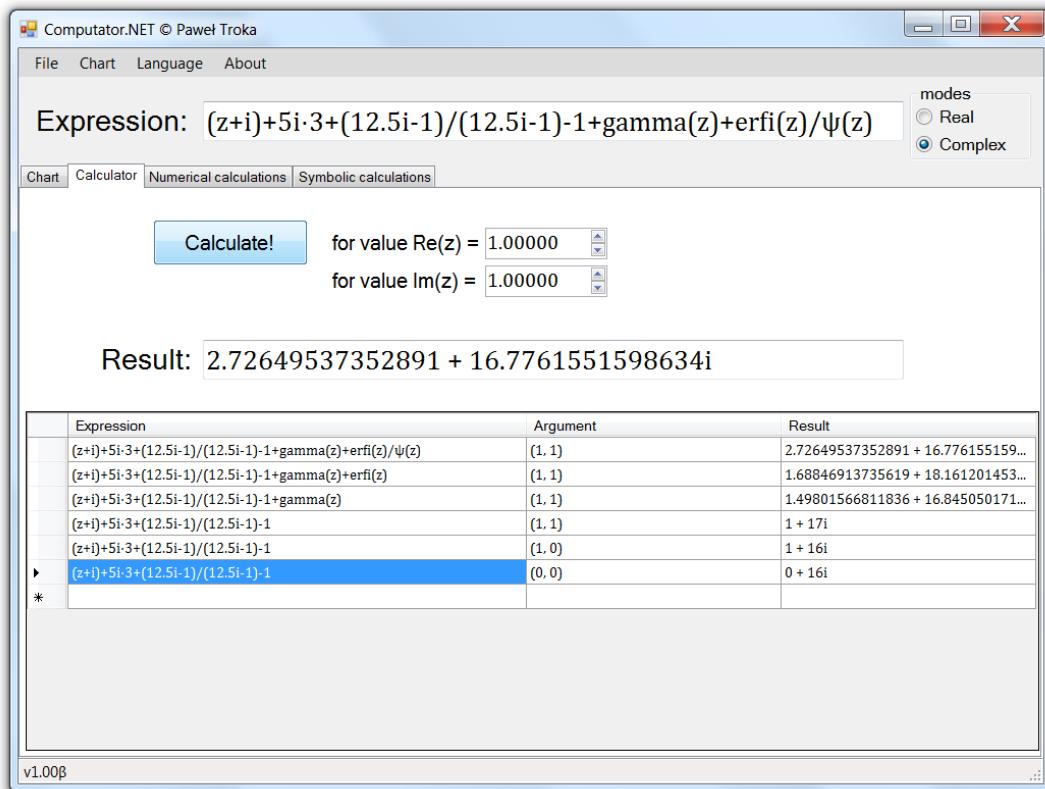
Rysunek 3.8 Kalkulator – efekty

Jak widać zastosowanie kontrolki DataGridView do przechowywania wyników poprzednich obliczeń może mieć ogromne znaczenie przy procesach badania zachowania niektórych funkcji czy wyrażeń.

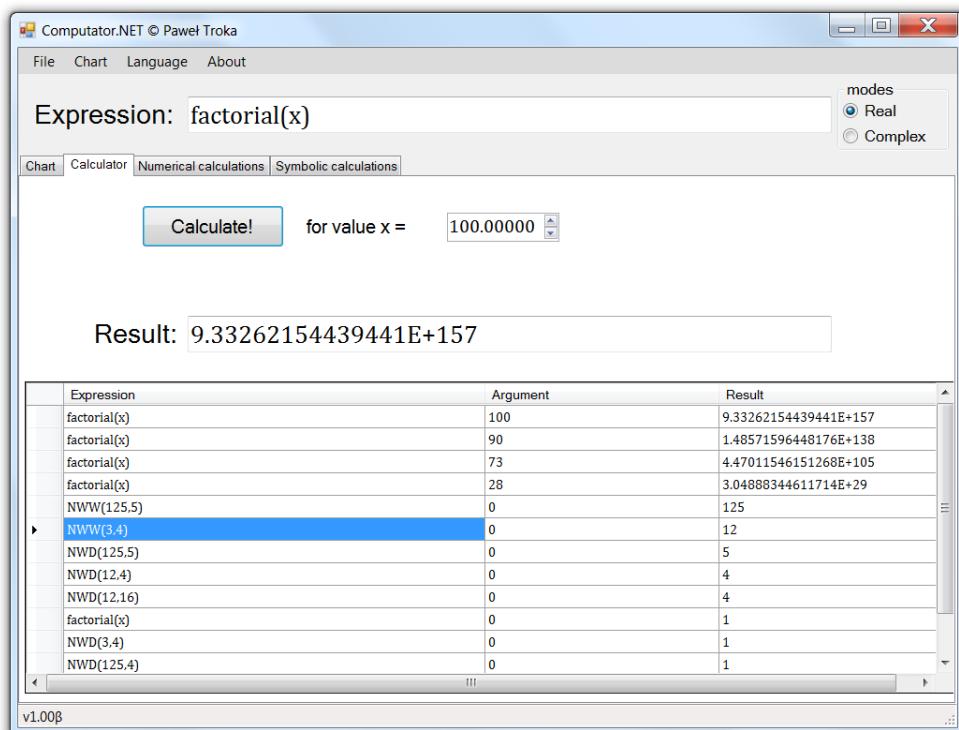


Rysunek 3.9 Kalkulator może też być małym, poręcznym narzędziem

Również zastosowanie kalkulatora do obliczeń na liczbach zespolonych, uwzględniając funkcje zmiennej zespolonej daje bardzo dobre efekty jak widać na rysunku 3.10.



Rysunek 3.10 Kalkulator - obliczenia na liczbach zespolonych

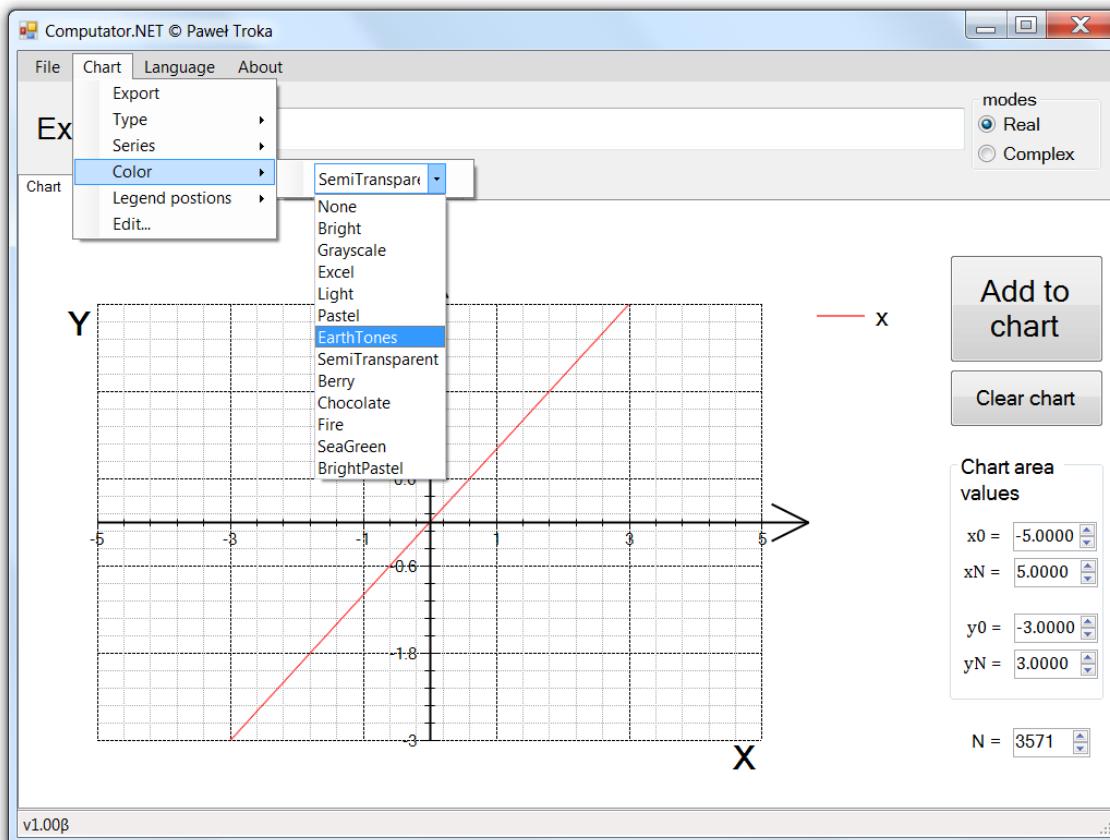


Rysunek 3.11 Kalkulator umożliwia też obliczenia funkcji liczb całkowitych i sortowanie wyników

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

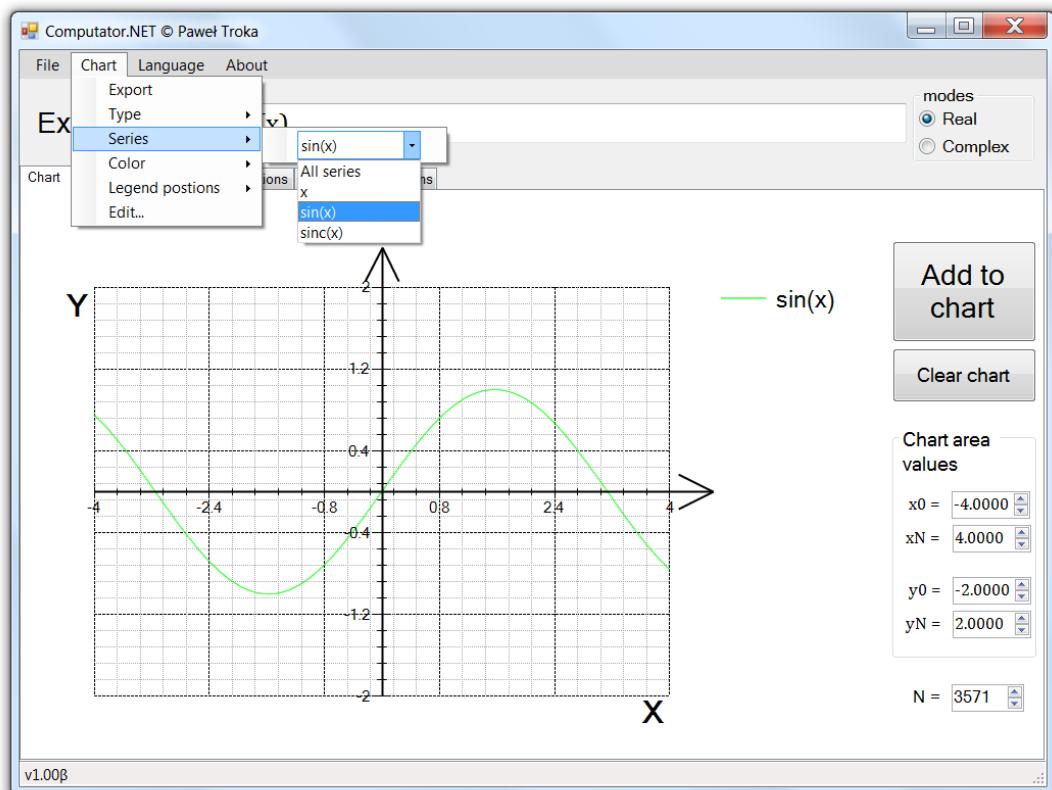
3.4 Wykresy funkcji rzeczywistych

Zgodnie z tym co opisywano w rozdziale drugim kontrolka Chart2D odpowiedzialna za wyświetlanie wykresów funkcji rzeczywistej posiada bardzo dużo możliwości konfiguracji. Aby udostępnić je użytkownikowi powstało menu dostępne z poziomu menu górnego w którym można zmieniać bardzo wiele opcji wykresu.

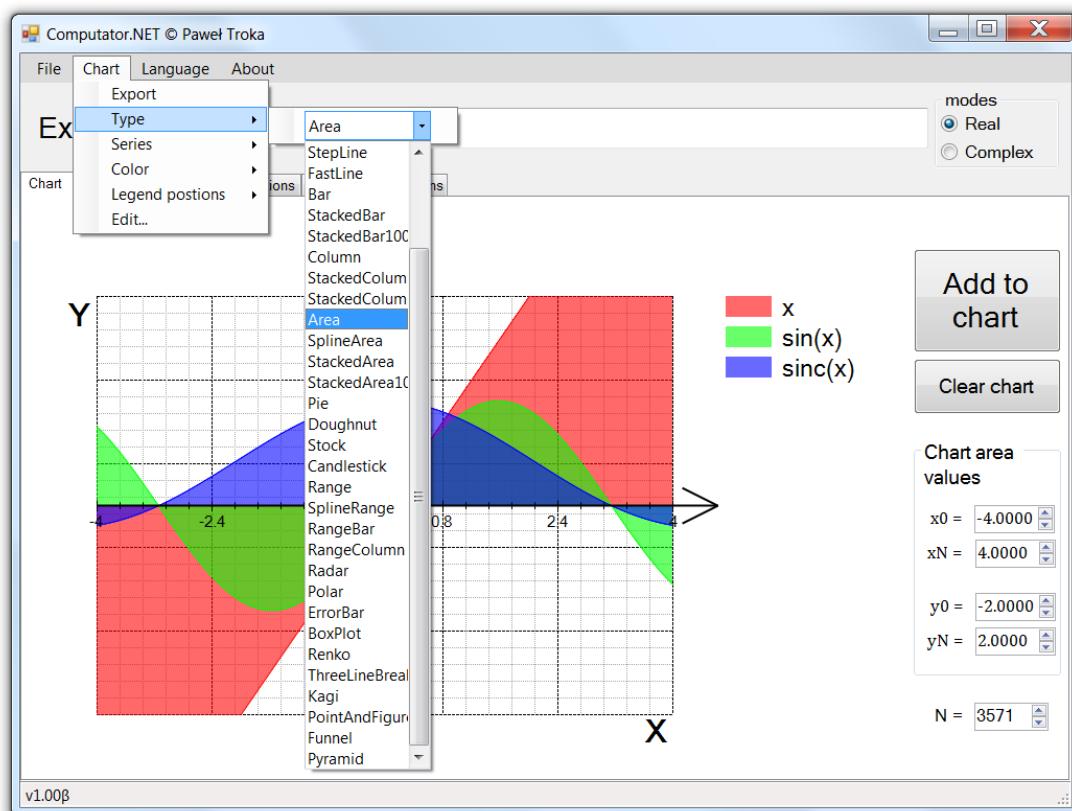


Rysunek 3.12 Menu górne edycji opcji wykresu

Jak widać na rysunku 3.12 w menu „Chart” można zmieniać naprawdę wiele. To menu umożliwia m. in. wybór kolorów na wykresie, ustawnienia pozycji legendy względem wykresu, wyboru konkretnej serii danych i zmianę typu wykresu. Poza tym przycisk „Export” umożliwia zapisanie wykresu jako obrazka – może to być niezwykle przydatne jeżeli za pomocą aplikacji miałyby być tworzone jakieś opracowania naukowe. Na rysunku 3.13 zostało pokazane jak wygodnie można wybierać sobie tylko jedną serię danych z posiadanych przez wykres. W przypadku gdy użytkownik chciałby porównywać na tym samym obszarze wartości różnych funkcji ale przytaczałoby go wyświetlanie ich wszystkich na raz taka opcja mogłaby okazać się niezbędna. Natomiast na rysunku 3.14 zostały zaprezentowane dostępne typy wykresów, jest ich naprawdę sporo.



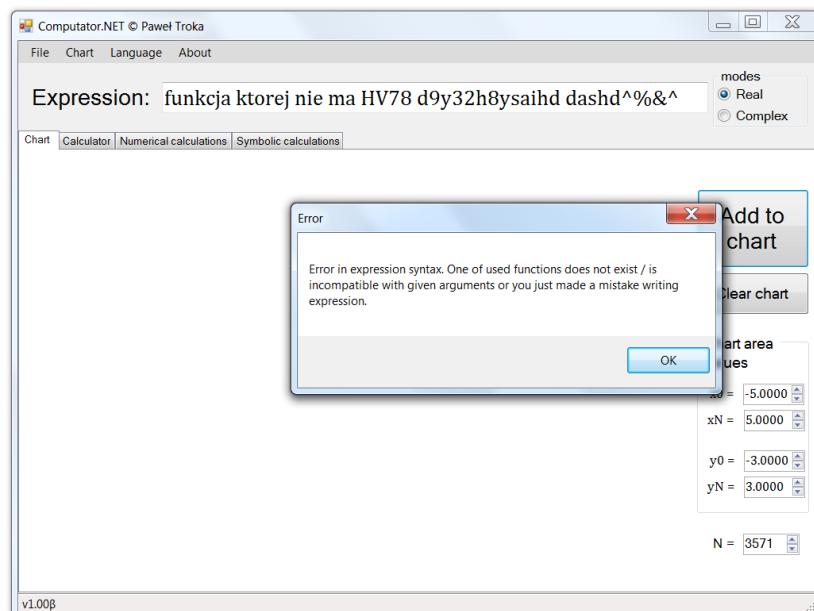
Rysunek 3.13 Możliwość wyboru serii na wykresie



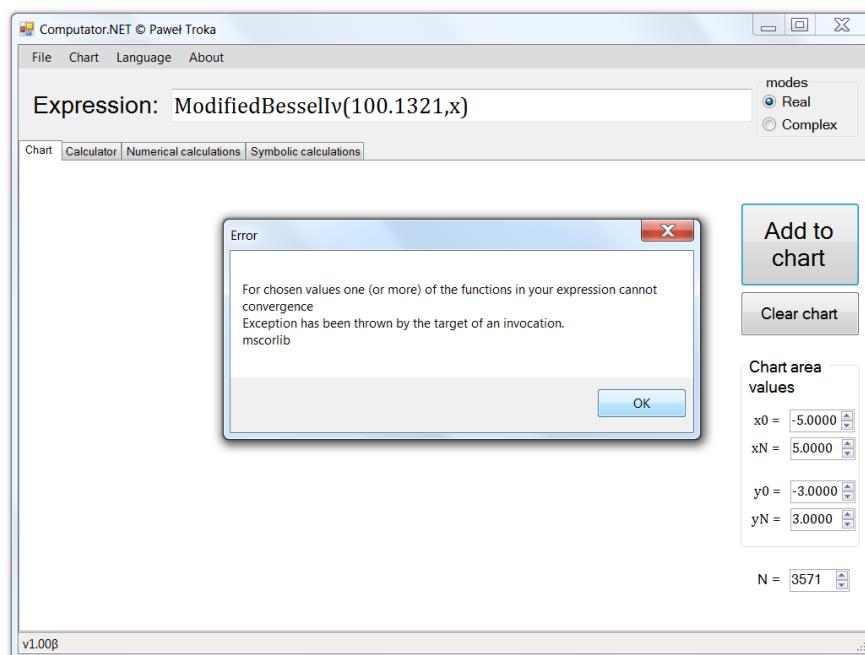
Rysunek 3.14 Różne typy wykresów

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

W rozdziale implementacyjnym zostało wspomniane, że niezwykle ważna jest obsługa błędów użytkownika. Pokazano tam m.in. obsługę mniej informacyjnych wyjątków tak aby bardziej użyteczny dla użytkownika wyjątek został wywołany. Ostatecznie wyjątek jest zawsze na końcu obsługiwany przez GUI. I to właśnie tutaj nie doda się do wykresu nieistniejąca funkcja a użytkownik otrzyma o tym czytelną informację jak zostało pokazane na rysunku 3.15. Czasami jednak winę ponosi ułomność implementacji o czym jest informowany użytkownik jak widać na rysunku 3.16.

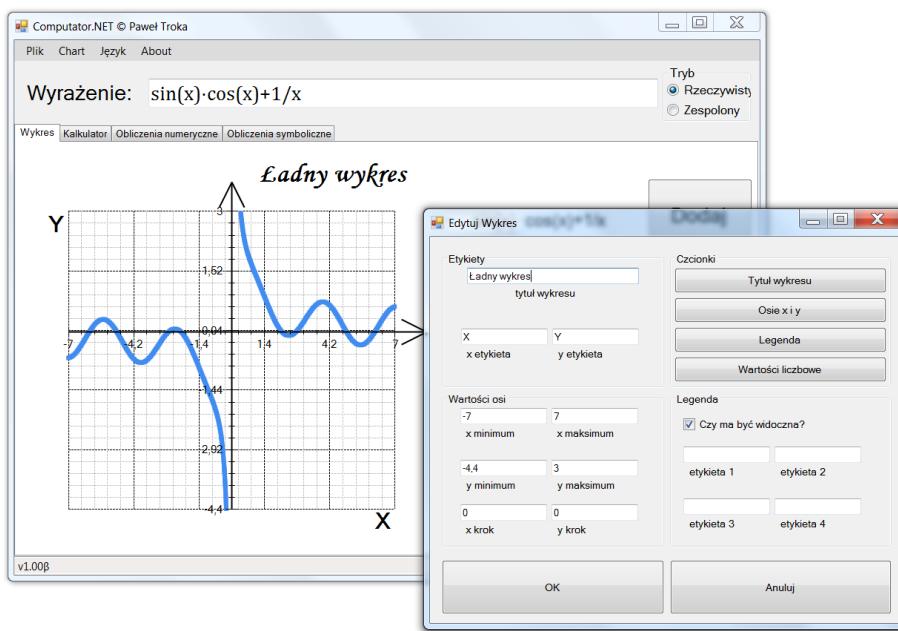


Rysunek 3.15 Obsługa błędów użytkownika

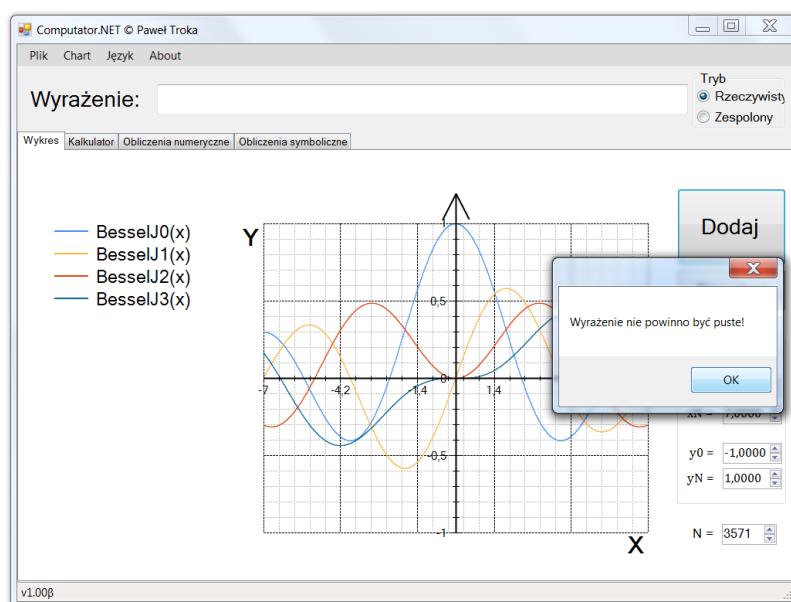


Rysunek 3.16 Szereg używany do obliczania funkcji dla danych parametrów nie jest zbieżny

Jak zostało wspomniane wcześniej aplikacja w założeniu ma być multiplatformowa (w planach) i ogólnie dostępna. Jednym z elementów dostępności jest język. Aplikacja została napisana w języku angielskim z ze względu na globalny zasięg tego języka i jego uniwersalność. Wersje obcojęzyczne są jednak w planach i już w trakcie powstawania aplikacji obok języka angielskiego stworzono również wersję polskojęzyczną. Widać to na rysunku 3.17 na którym zaprezentowano również formularz umożliwiający edycję różnych właściwości wykresu. Również obsługa błędów użytkownika w języku polskim jest równie dobra co w oryginalnej wersji, patrz rysunek 3.18.



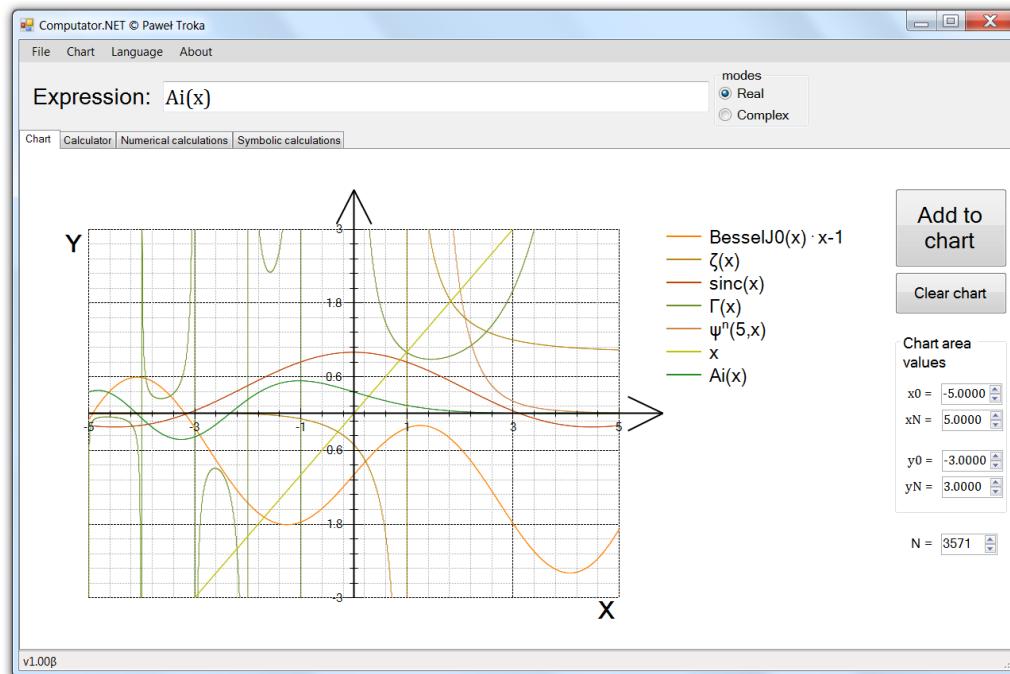
Rysunek 3.17 Formularz edycji różnych właściwości wykresu i polskojęzyczny interfejs



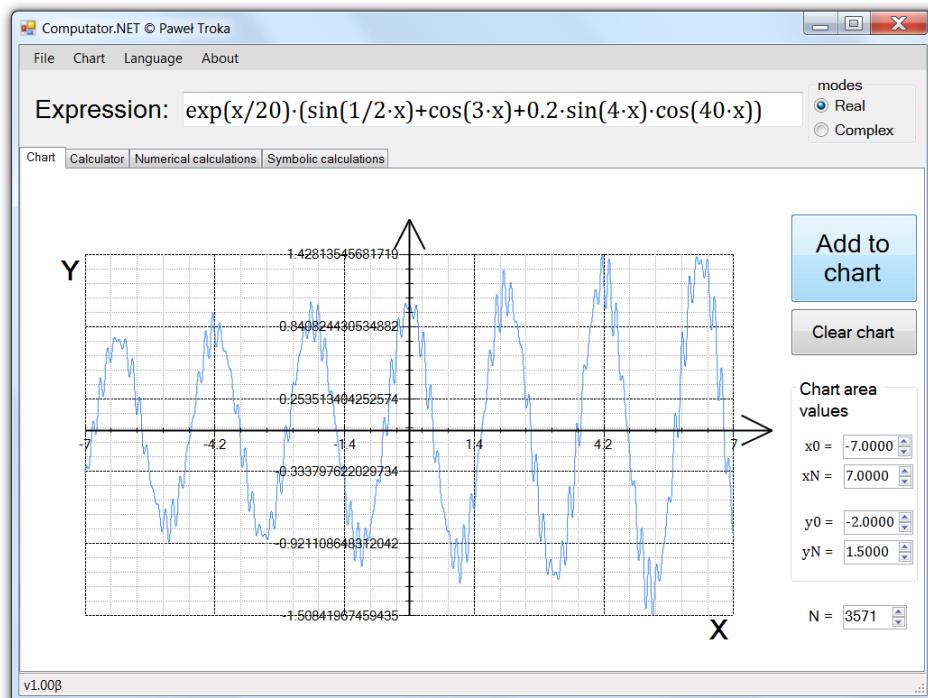
Rysunek 3.18 Obsługa błędów użytkownika w polskojęzycznym interfejsie

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

Tak naprawdę gdy już użytkownik się zapozna z możliwościami rysującymi i konfiguracyjnymi wykresu to najważniejsze okazują się zaimplementowane w programie funkcje. Na rysunku 3.19 pokazano wykresy niektórych funkcji specjalnych i elementarnych. Na rysunku 3.20 natomiast można zobaczyć wykres funkcji w tryb ostrego/twardego skalowania z wyłączoną legendą.



Rysunek 3.19 Przykładowe wykresy funkcji



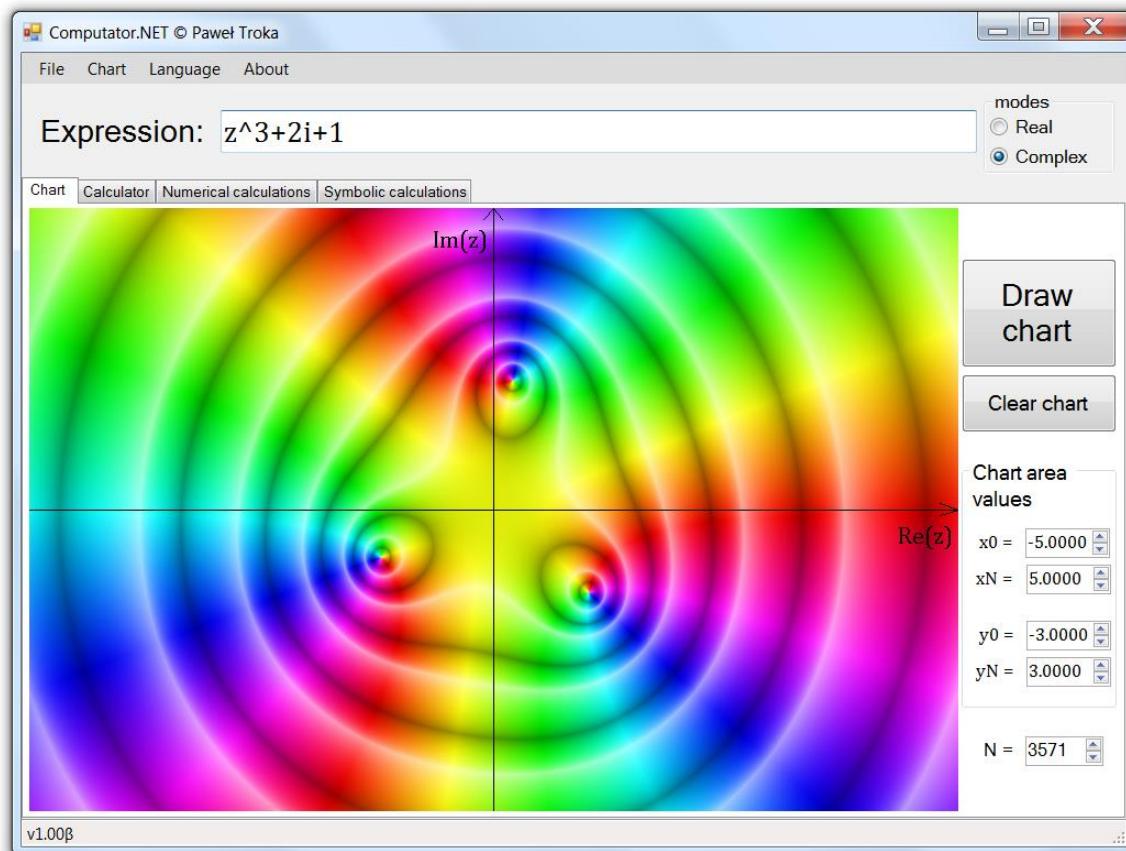
Rysunek 3.20 Ostre/twarde skalowanie i wykres bez legendy

3.5 Wykresy funkcji zespolonych

Poprawna wizualizacja funkcji zespolonej była nie lada wyzwaniem. Jednak dzięki opisanej w rozdziale implementacyjnym metody kolorowania domeny i odpowiednim algorytmom udało się zrealizować tę część pracy na bardzo wysokim poziomie.

Wykresy funkcji zespolonych uzyskane w aplikacji są ładne i jak widać na rysunku 3.21 dobrze odwzorowują zachowanie się funkcji zespolonej. Takie elementy jak biegunki czy miejsca zerowe są dobrze wyeksponowane. Również widać bardzo dużą czytelność uzyskanych linii konturowych, użytkownik nie powienien mieć problemu zlokalizowania miejsc o tych samych wartościach modułu funkcji a dzięki mocno nasyconym kolorom ma także pełną informacje o zachowaniu fazy funkcji zespolonej.

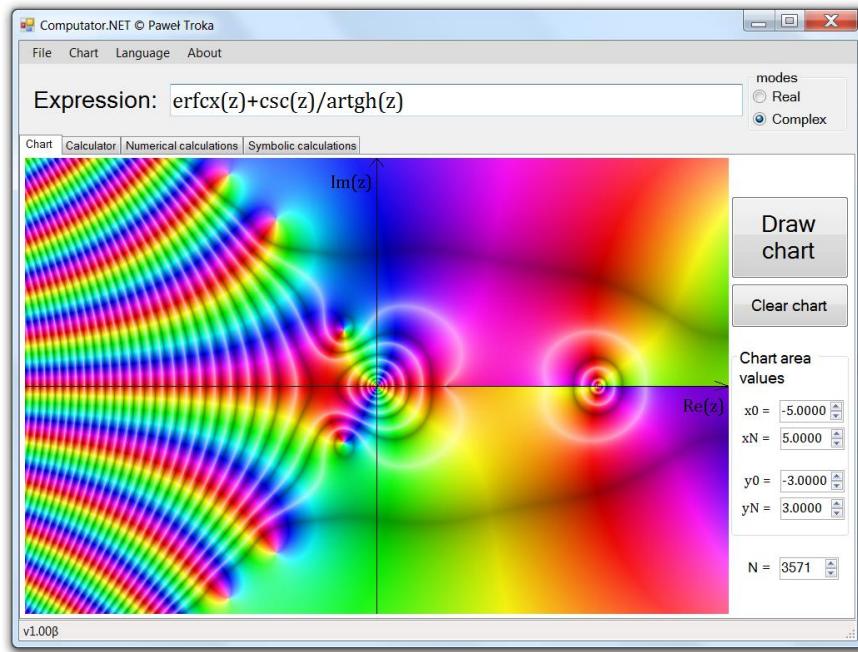
Jedynie wydaje się teraz, że informacja o wartościach na osiach $\operatorname{Re}(z)$ i $\operatorname{Im}(z)$ jest nieco ukryta. Nie rysuje się żadnych etykiet liczbowych na tych osiach co może przeszkadzać w szybkim zrozumieniu zachowania funkcji, mamy co prawda wartości przedziału rysowania z boku w bolu „Chart area values” ale podane explicite wartości liczbowe na osiach jednak zwiększyłyby czytelność wykresu. Zostanie to dodane prawdopodobnie w przyszłości.



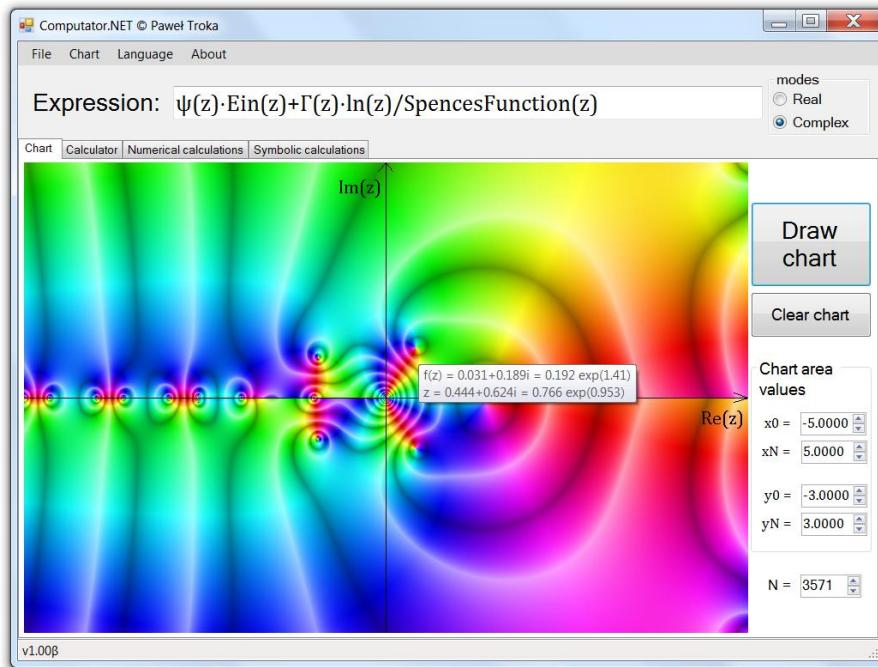
Rysunek 3.21 Przykład wykresu funkcji zespolonej - elementarny wielomian trzeciego stopnia

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

Na rysunkach 3.22 i 3.23 ukazano wykresy niektórych funkcji specjalnych zmiennej zespolonej (a właściwie ich arytmetycznej kombinacji). Na pewno wykresy te dobrze nam przybliżają zachowanie się tych funkcji, ale z powodu brakującej informacji liczbowej zaimplementowano ToolTip wyświetlający po kliknięciu na obszar wykresu wartości argumentu i funkcji w danym punkcie, widoczne jest to na rysunku 3.23.

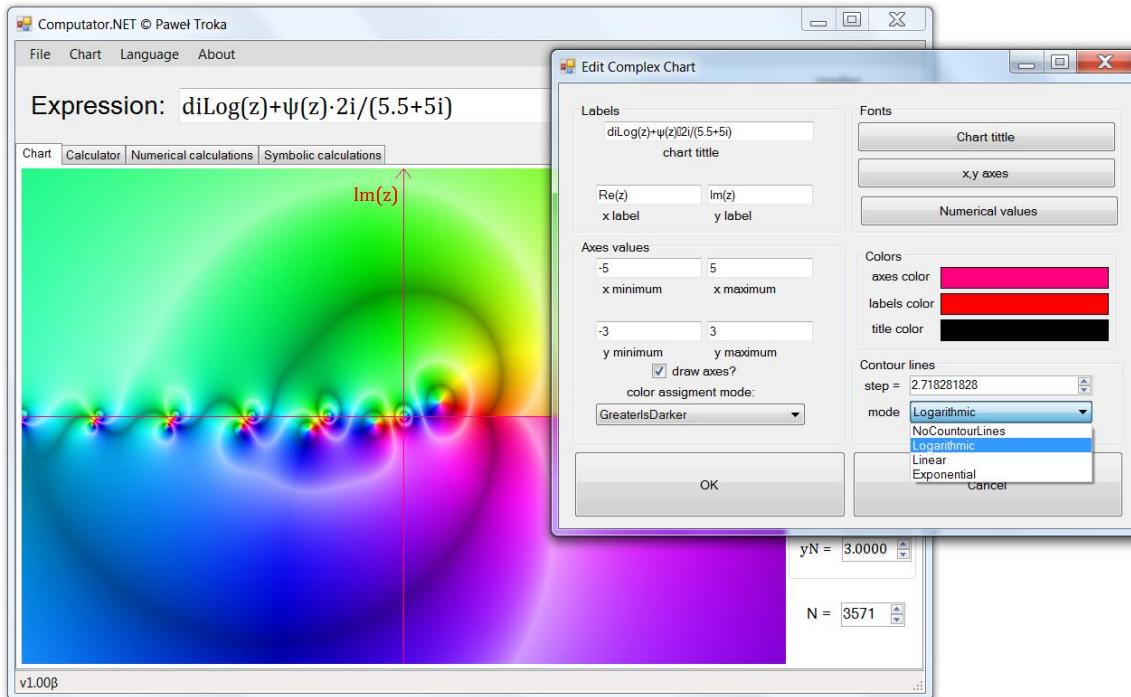


Rysunek 3.22 Przykład złożonej funkcji zespolonej na wykresie

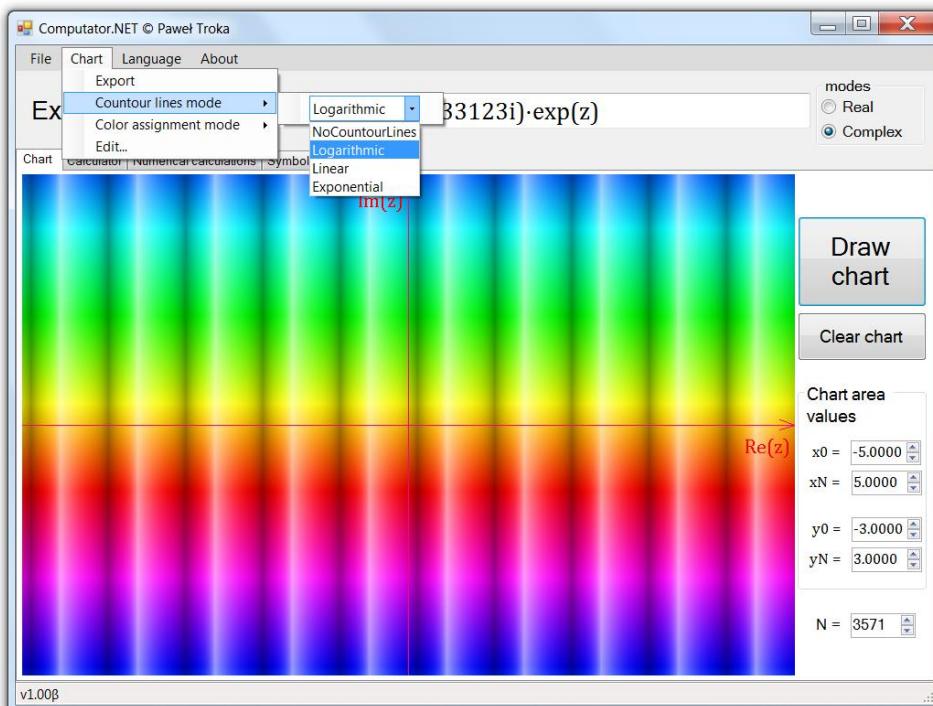


Rysunek 3.23 Przykładowe funkcje specjalne argumentu zespolonego na wykresie i ToolTip z wartościami punktu

Wykres zespolony tak samo jak rzeczywisty w implementowanej aplikacji udostępnia pewne właściwości konfiguracyjne. Na rysunku 3.24 widać formularz zmiany różnych opcji wykresu, natomiast na rysunku 3.25 widać menu wykresu które też udostępnia możliwość zmiany niektórych parametrów wizualizacji (i nie tylko).



Rysunek 3.24 Edycja opcji wykresu zespolonego

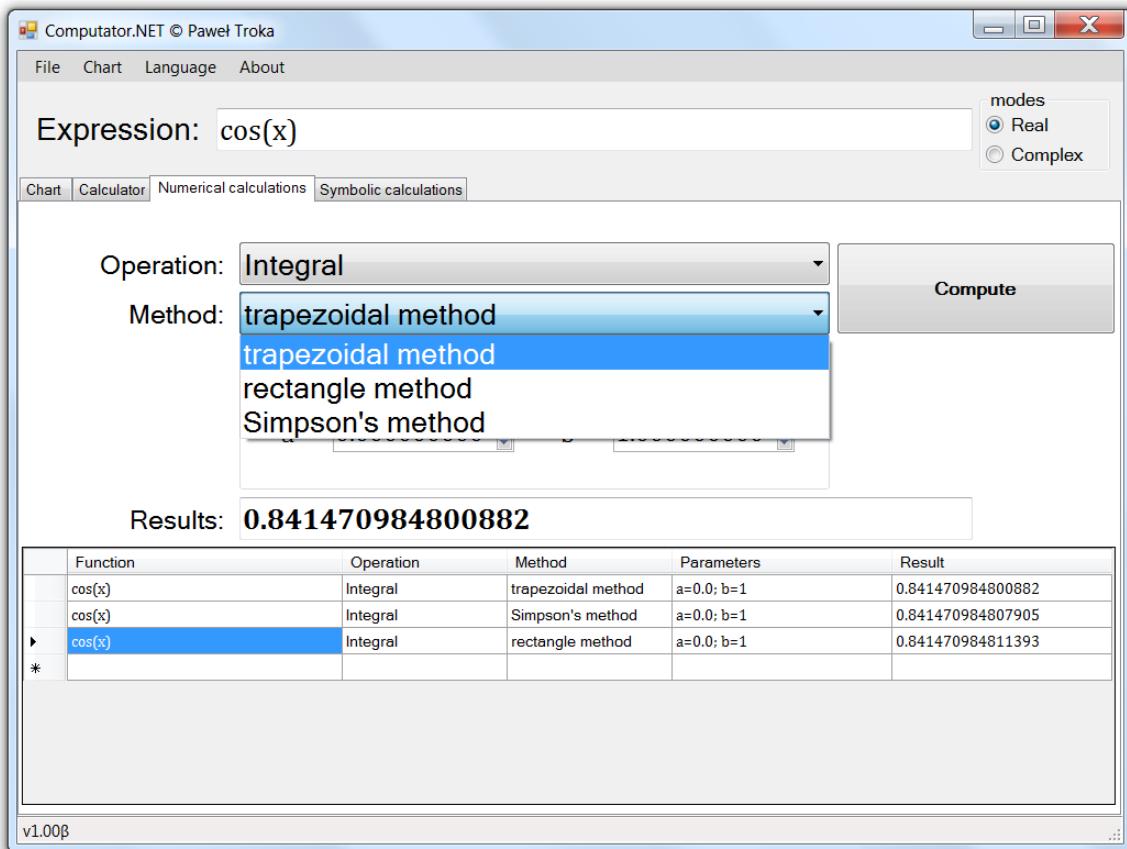


Rysunek 3.25 Różne tryby rysowania linii konturowych

3.6 Obliczenia numeryczne

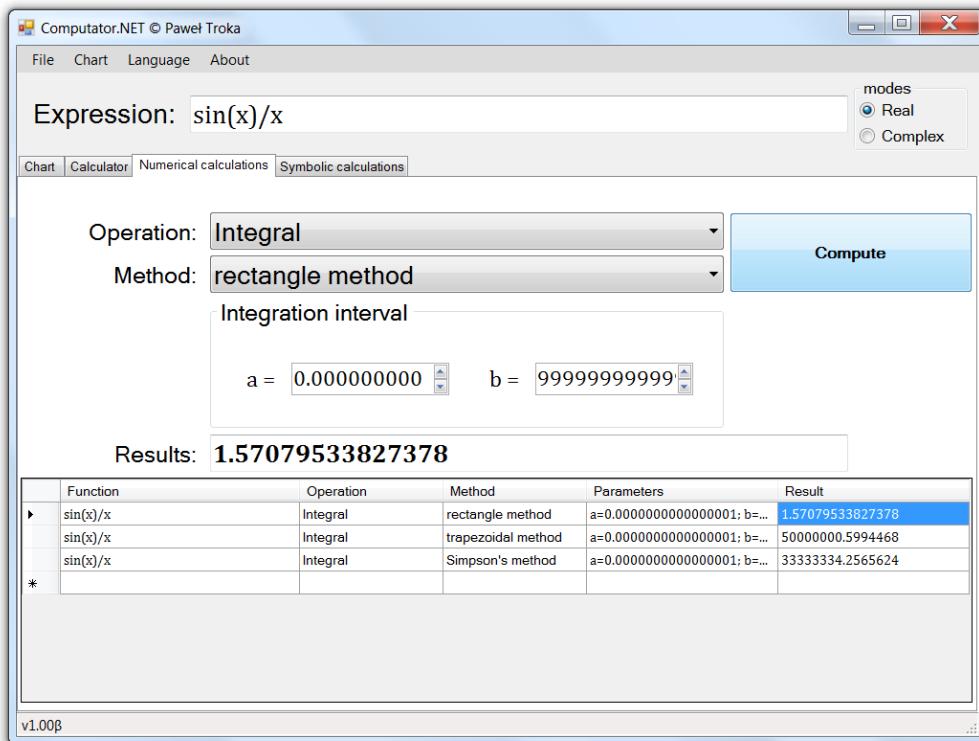
W tym podrozdziale zostaną zaprezentowane efekty implementacji metod numerycznych z podrozdziału 2.3.

Na rysunku 3.26 jest pokazane porównanie poprzez wyniki z kontrolki DataGridView całkowania numerycznego różnymi metodami. Najdokładniejsza na tym niewielkim przedziale okazała się metoda trapezów.



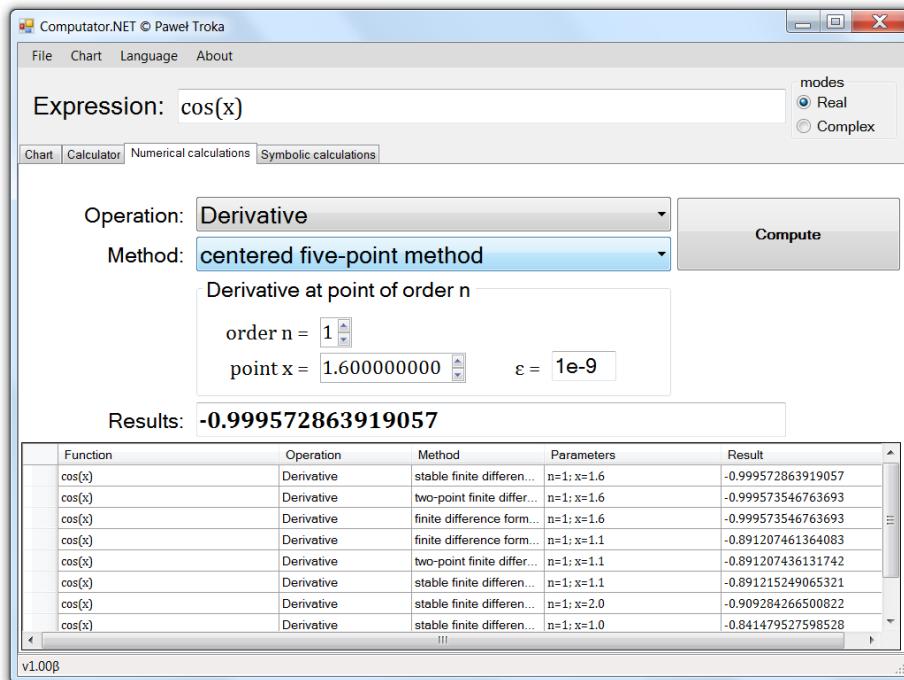
Rysunek 3.26 Przykład numerycznego całkowania funkcji

Jak zostało wspomniane w podrozdziale 2.3 w punkcie 1 najbardziej uniwersalną metodą całkowania jest najprostsza metoda prostokątów. Dzieje się tak z powodu, że jako jedyna liczy całkę tak naprawdę z definicji wg. Riemanna. Na rysunku 3.27 pokazano, że jako jedyna policzyła poprawnie całkę z bardzo dużego przedziału. Błąd obliczenia całki w pozostałych dwóch metodach bardzo szybko rośnie wraz ze wzrostem długości przedziału i w efekcie na bardzo długim przedziale metody te okazały się bezużyteczne. Należy o tym pamiętać używając aplikacji, że nie każda metoda nadaje się do każdego przypadku i dotyczy to zresztą nie tylko całkowania.



Rysunek 3.27 Przykład ukazujący wady niektórych metod całkowania

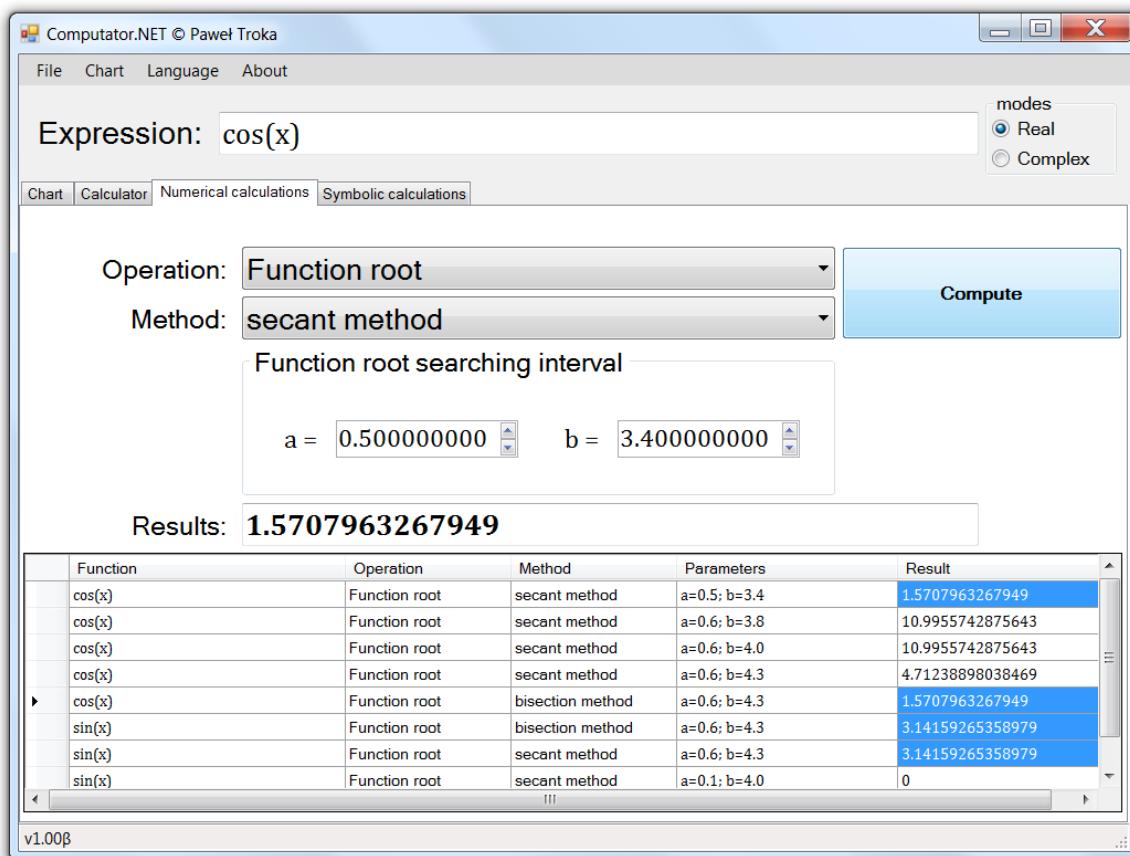
Pochodną funkcji w punkcie można też w implementowanej aplikacji liczyć różnymi sposobami. Co ciekawe jednak wyniki różnią się na tyle niewiele (i to niemal niezależnie od badanej funkcji, chyba, że jej tempo zmian na jakimś przedziale jest bardzo niestandardowe), że każda metoda się nadaje.



Rysunek 3.28 Przykład obliczeń numerycznych pochodnej funkcji w punkcie

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

Ostatnia z trzech zaimplementowanych w aplikacji metod numerycznych – szukanie pierwiastków funkcji zaskakuje swoją dokładnością. Mimo szybkiego czasu wykonania metod zwracane wyniki są dokładne co do ostatniej zapisanej w kontrolce DataGridView cyfry. Obie metody również zwracają dokładnie te same wyniki (nie licząc przypadków gdy metoda siecznych znajdzie pierwiastek poza badanym przedziałem, ale i tak nie jest to wtedy błąd z racji tego, że jest ona metodą otwartą jak napisano w podrozdziale 2.3 punkcie 3) dla tych samych przedziałów.



Rysunek 3.29 Przykład numerycznego znajdowania pierwiastków funkcji

Podsumowanie i wnioski

Uzyskana w efekcie pracy inżynierskiej aplikacja jest pełnoprawnym narzędziem wspomagającym pracę inżyniera i naukowca, uwzględniając również zbiór liczb zespolonych. Testy aplikacji wykazały poprawność jej działania, dokładność wykonywanych obliczeń oraz dużą czytelność tworzonych przez aplikację wykresów. Również dzięki optymalizacji niektórych działań (jak zrównoleglenie procesu rysowania wykresu zespolonego dla funkcji zdefiniowanej przez użytkownika), uzyskano wysoką jak na taką aplikację wydajność.

Spełnione zostały wszystkie cele zdefiniowane na początku pracy. Nie zrealizowano co prawda, pojawiających się w bardzo wczesnych wersjach celów pracy, obliczeń symbolicznych, ale jest to rzecz nieprosta implementacyjnie i potrzeba po prostu więcej czasu aby to zrobić. Udało się zrealizować natomiast dostępność bardzo wielu funkcji elementarnych (właściwie wszystkich popularnych) i specjalnych (wielu), jedynie wydaje się, że jest trochę niedużo funkcji specjalnych zmiennej zespolonej. Jak napisano jednak wcześniej są one trudne w implementacji. Jednakże dzięki kontaktowi z autorem biblioteki Meta.Numerics wydaje się, że w przyszłości poprzez współpracę z autorem biblioteki uda się uzyskać wszystkie funkcje specjalne, te, które są dla argumentów rzeczywistych również dla zmiennej zespolonej.

Interfejs użytkownika i doświadczenie w pracy z aplikacją przeszło wszelkie oczekiwania. W planach był jedynie wygodny interfejs, natomiast dzięki takim udogodnieniom jak kontrolka dataGridView wyświetlająca historię operacji wraz z wynikami czy podpowiedzi nazw funkcji (wraz z opisami) aplikacja zyskała obok użyteczności również charakter dydaktyczny. Mało jest w końcu aplikacji do obliczeń numerycznych, które pozwalają na wygodne porównanie wyników z różnych metod.

Ostatecznie wydaje się, że wybrana metoda uzyskiwania wyrażenia lub funkcji definiowanej w trakcie działania programu przez użytkownika poprzez komplikację dynamiczną, okazała się dobrym wyborem. Co prawda, występowały różne komplikacje takie jak np. utrudnione debugowanie kodu kompilowanego dynamicznie. Poza tym część rozwiązań do których zmusiła dynamiczna komplikacja nie należy do rozwiązań zbyt eleganckich (jak np. doklejanie kodu funkcji w tym procesie). Ale łatwość rozbudowy aplikacji o nowe funkcje i zminimalizowanie ryzyka ukrytych błędów oraz ułatwienie implementacji całości aplikacji zdecydowanie obroniły metodę.

Również zarządzanie wyjątkami i kontrolowanie błędów użytkownika zostało zrealizowane na tyle dobrze (mimo utrudnień wynikających właśnie ze wspomnianej dynamicznej komplikacji, gdzie trudniej uchwycić wyjątek), że użytkownik w większości przypadku wie, dlaczego wyświetlił mu się błąd.

Bardzo dobrym pomysłem okazał się również podział aplikacji na moduły poprzez dostępny w WinForms tabControl. Zaistniał także pomysł, aby w przyszłości udostępnić użytkownikom możliwość tworzenia własnych modułów dla aplikacji. Przy dzisiejszej budowie aplikacji trzeba jednak przyznać, że to raczej odległa przyszłość.

Dalszy rozwój aplikacji wydaje się bardzo obiecujący, gdyż dzięki dobrze przemyślanej budowie opartej o poprawnie nazwane i podzielone pakiety i klasy, aplikacja niezwykle wygodnie się rozwija. W planach na przyszłość są między innymi:

- Obliczenia symboliczne
- Dokładne, wielojęzyczne opisy funkcji i stałych wyświetlane podczas używania programu
- Znalezienie i poprawienie wszystkich błędów oraz jak najlepszy „workaround” błędów istniejących w używanych kontrolkach (np. błędy w klasie Chart)
- Wielojęzyczny interfejs (przynajmniej angielski, polski, niemiecki)
- Multiplatformowa wersja na systemy inne niż Windows takie jak Linux i Mac OS X dzięki kompilatorowi Mono
- Wersje na smartfony - system Android, system iOS i system Windows Phone dzięki kompilatorowi Mono i Mono.Touch
- Więcej funkcji specjalnych, zwłaszcza zmiennej zespolonej
- Dodanie wszystkich popularnych stałych fizycznych i matematycznych
- Więcej operacji numerycznych, również więcej metod tych samych operacji (np. kolejne metody całkowania funkcji)
- Operacje numeryczne również dla funkcji zmiennej zespolonej
- Zwiększenie wydajności wszelkich procedur rysujących (o ile to możliwe)
- Wykresy funkcji dwóch zmiennych (w przestrzeni 3D)

Podsumowując, aplikacja powinna sprawdzić się w rękach inżynierów i naukowców, a także studentów, czy nawet licealistów dzięki wygodnemu i przemyślanemu interfejsowi, opisom funkcji i stałych i wielu innym elementom. Fakt, że aplikacja jest wydajna, wygodna i dokładna w obliczeniach oraz wysoce konfigurowalna (cecha której brakuje dzisiaj wielu programom) i w planach multiplatformowa (patrz Mono) zachęca do używania jej na uczelniach, w ośrodkach badawczych i dydaktycznych.

Literatura

- [1] [http://en.wikipedia.org/wiki/C_Sharp_\(programming_language\)](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)) [dostęp 22.12.2014]
- [2] Wykłady z przedmiotu “Platformy Technologiczne” autorstwa mgra inż. Tomasza Gawrona w roku akademickim 2012/2013, wydział ETI, Politechnika Gdańska http://www.eti.pg.gda.pl/katedry/kask/dydaktyka/Platformy_technologiczne/.NET/2013/CS.pptx [dostęp 22.12.2014]
- [3] http://en.wikipedia.org/wiki/Moore%27s_law#Consequences_and_limitations [dostęp 23.12.2013]
- [4] Konferencja “C++11, VC++11, and Beyond”, Herb Sutter, Going Native 2012, Redmond, WA, USA, 3 luty 2013, dzień 2. <http://ecn.channel9.msdn.com/events/GoingNative12/GN12HerbSutterCpp11VCBeyondDay2.pptx> [dostęp 23.12.2013]
- [5] [http://en.wikipedia.org/wiki/Mono_\(software\)](http://en.wikipedia.org/wiki/Mono_(software)) [dostęp 26.12.2013]
- [6] http://en.wikipedia.org/wiki/.NET_Framework [dostęp 26.12.2013]
- [7] http://en.wikipedia.org/wiki/.NET_Framework_version_history [dostęp 26.12.2013]
- [8] <http://www.lhotka.net/weblog/DoesNETHaveAFuture.aspx> [dostęp 26.12.2013]
- [9] http://en.wikipedia.org/wiki/Comparison_of_C_Sharp_and_Java [dostęp 26.12.2013]
- [10] http://en.wikipedia.org/wiki/Base_Class_Library [dostęp 27.12.2013]
- [11] <http://ranking.pl/pl/rankings/operating-systems.html> [dostęp 27.12.2013]
- [12] <http://joshsmithonwpf.wordpress.com/2007/09/05/wpf-vs-windows-forms> [dostęp 28.12.2013]
- [13] http://en.wikipedia.org/wiki/Windows_Presentation_Foundation [dostęp 28.12.2013]
- [14] http://en.wikipedia.org/wiki/Windows_Forms [dostęp 28.12.2013]
- [15] <http://www.mono-project.com/Compatibility> [29.12.2013]
- [16] http://en.wikipedia.org/wiki/Anders_Hejlsberg [dostęp 30.12.2013]
- [17] <http://en.wikipedia.org/wiki/ADO.NET> [dostęp 30.12.2013]
- [18] http://en.wikipedia.org/wiki/Dynamic_Language_Runtime [dostęp 30.12.2013]
- [19] http://en.wikipedia.org/wiki/Language_Integrated_Query [dostęp 31.12.2013]
- [20] http://en.wikipedia.org/wiki/Common_Intermediate_Language [dostęp 31.12.2013]
- [21] http://en.wikipedia.org/wiki/Parallel_Extensions [dostęp 31.12.2013]
- [22] <http://stackoverflow.com/questions/15143967/mathevaluator-parser> [dostęp 2.01.2014]

- [23] http://en.wikipedia.org/wiki/Shunting-yard_algorithm [dostęp 7.01.2014]
- [24] http://en.wikipedia.org/wiki/Reverse_Polish_notation [dostęp 7.01.2014]
- [25] <http://msdn.microsoft.com/en-us/library/microsoft.csharp.csharpcodeprovider.aspx>
[dostęp 2.01.2014]
- [26] Reflection in the .NET Framework <http://msdn.microsoft.com/en-us/library/f7ykdhshy.aspx> [dostęp 4.01.2014]
- [27] <http://www.codeproject.com/Articles/199800/Evaluate-complex-and-real-math-calculator> [dostęp 8.01.2014]
- [28] <http://stackoverflow.com/questions/21161984/how-to-get-whole-code-body-of-public-static-class-in-one-tostring-method> [dostęp 16.01.2014]
- [29] <http://metanumerics.codeplex.com> [dostęp 9.01.2014]
- [30] <http://stackoverflow.com/questions/1749966/c-sharp-how-to-determine-whether-a-type-is-a-number/1750093> [dostęp 9.01.2014]
- [31] Vladimirov, V. S. (1971), Equations of mathematical physics, Marcel Dekker, ISBN 0-8247-1713-9.
- [32] <http://www.meta-numerics.net/Pages/Specs.aspx> [dostęp 06.01.2014]
- [33] <http://metanumerics.codeplex.com/workitem/7722> [dostęp 07.01.2014]
- [34] http://en.wikipedia.org/wiki/Numerical_integration [dostęp 13.01.2014]
- [35] http://en.wikipedia.org/wiki/Rectangle_method [dostęp 13.01.2014]
- [36] http://en.wikipedia.org/wiki/Trapezoidal_rule [dostęp 13.01.2014]
- [37] http://en.wikipedia.org/wiki/Simpson%27s_rule [dostęp 13.01.2014]
- [38] http://en.wikipedia.org/wiki/Numerical_differentiation [dostęp 15.01.2014]
- [39] http://en.wikipedia.org/wiki/Zero_of_a_function [dostęp 18.01.2014]
- [40] http://en.wikipedia.org/wiki/Root-finding_algorithm [dostęp 18.01.2014]
- [41] http://en.wikipedia.org/wiki/Bisection_method [dostęp 18.01.2014]
- [42] http://en.wikipedia.org/wiki/Secant_method [dostęp 18.01.2014]
- [43] <http://msdn.microsoft.com/library/system.windows.forms.datavisualization.charting.chart.aspx> [dostęp 22.01.2014]
- [44] <http://www.dotnetperls.com/chart> [dostęp 22.01.2014]
- [45] <http://www.dundas.com> [dostęp 22.01.2014]
- [46] <http://blogs.msdn.com/b/alexgor/archive/2008/11/07/microsoft-chart-control-vs-dundas-chart-control.aspx> [dostęp 22.01.2014]
- [47] <http://stackoverflow.com/questions/17210257/microsoft-chart-control-redraw-chart-after-failure-red-cross> [dostęp 22.01.2014]

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

- [48] <http://msdn.microsoft.com/library/364x0z75.aspx> [dostęp 22.01.2014]
- [49] <http://stackoverflow.com/questions/3826724/mschart-unhandled-overflow-exception-after-zooming> [dostęp 22.01.2014]
- [50] <http://forums.roguewave.com/showthread.php?499-C-Imsl.Chart2D.Chart-throws-System.OverflowException> [dostęp 22.01.2014]
- [51] <http://support2.dundas.com/forum/tm.aspx?m=16609> [dostęp 22.01.2014]
- [52] <http://social.msdn.microsoft.com/Forums/vstudio/en-US/233153fb-81e8-4faf-9919-9ee2c640c94d/unhandled-overflow-exception-after-zooming?forum=MSWinWebChart> [dostęp 22.01.2014]
- [53] <http://stackoverflow.com/questions/3468495/what-are-the-hard-bounds-for-drawing-coordinates-in-gdi> [dostęp 22.01.2014]
- [54] <http://bobpowell.net/extents.aspx> [dostęp 22.01.2014]
- [55] <http://www.mail-archive.com/mono-patches@lists.ximian.com/msg31263.html> [dostęp 22.01.2014]
- [56] <http://msdn.microsoft.com/en-us/library/dd489249.aspx> [dostęp 22.01.2014]
- [57] http://en.wikipedia.org/wiki/Domain_coloring [dostęp 23.01.2014]
- [58] http://en.wikipedia.org/wiki/Riemann_surface [dostęp 23.01.2014]
- [59] <http://www.pacifict.com/ComplexFunctions.html> [dostęp 23.01.2014]
- [60] http://en.wikipedia.org/wiki/Color_wheel_graphs_of_complex_functions [dostęp 23.01.2014]
- [61] Claudio Rocchini C++ algorithm for complex functions domain coloring
http://en.wikipedia.org/wiki/File:Color_complex_plot.jpg [dostęp 23.01.2014]
- [62] <http://www.codeproject.com/Articles/80641/Visualizing-Complex-Functions> [dostęp 23.01.2014]
- [63] <http://mathworld.wolfram.com/GammaFunction.html> [dostęp 23.01.2014]
- [64] http://en.wikipedia.org/wiki/Conformal_pictures [dostęp 23.01.2014]
- [65] http://en.wikipedia.org/wiki/Complex_analysis [dostęp 23.01.2014]
- [66] <http://msdn.microsoft.com/library/system.string.format.aspx> [dostęp 26.01.2014]
- [67] <http://stackoverflow.com/questions/8624229/cambria-math-big-top-and-bottom-margin> [dostęp 26.01.2014]
- [68] <http://www.codeproject.com/Articles/365974/Autocomplete-Menu> [dostęp 26.01.2014]

Dodatek A. Spis zawartości dołączonej płyty CD

Praca.pdf – niniejszy dokument

Computator.NET – folder zawierający projekt i kod źródłowy zrealizowanej aplikacji

Computator.NET v1.0 – folder zawierający skompilowaną aplikację

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

Dodatek B. Błąd nr 1 znaleziony w klasie DataVisualization.Chart

```
A first chance exception of type 'System.OverflowException' occurred in System.Windows.Forms.DataVisualization.dll
Additional information: Value was either too large or too small for a Decimal.

Stack trace:
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Axis.RoundedValues(double inter = 2.0E+28, bool shouldStartFromZero = true, bool autoMax = true, bool autoMin, ref double min = -6.0E+28, ref double max = 0.0) + 0x20e bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Axis.EstimateNumberAxis(ref double minValue = -6.0E+28, ref double maxValue = 0.0, bool shouldStartFromZero, int preferredNumberOfIntervals, bool autoMaximum, bool autoMinimum) + 0x2bd bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Axis.EstimateAxis(ref double minValue, ref double maxValue, bool autoMaximum, bool autoMinimum) + 0xb0 bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Axis.EstimateAxis() + 0x164 bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.ChartArea.SetDefaultAxesValues() + 0x180 bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.ChartArea.SetData(bool initializeAxes, bool checkIndexedAligned) + 0x8ce bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.ChartArea.ReCalcInternal() + 0x76 bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.ChartPicture.Paint(System.Drawing.Graphics graph, bool paintTopLevelElementOnly = false) + 0x45c bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Chart.OnPaint(System.Windows.Forms.PaintEventArgs e = {ClipRectangle = {System.Drawing.Rectangle}}) + 0x508 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.PaintWithErrorHandling(System.Windows.Forms.PaintEventArgs e, short layer) + 0x96 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.WmPaint(ref System.Windows.Forms.Message m) + 0x326 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.WndProc(ref System.Windows.Forms.Message m) + 0x3a7 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.ControlNativeWindow.OnMessage(ref System.Windows.Forms.Message m) + 0x11 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.ControlNativeWindow.WndProc(ref System.Windows.Forms.Message m) + 0x35 bytes
System.Windows.Forms.dll!System.Windows.Forms.NativeWindow.DebuggableCallback(System.IntPtr hWnd, int msg = 15, System.IntPtr wParam, System.IntPtr lParam) + 0x5e bytes
[Native to Managed Transition]
[Managed to Native Transition]
System.Windows.Forms.dll!System.Windows.Forms.Application.ComponentManager.System.Windows.Forms.UnsafeNativeMethods.IManager.FPushMessageLoop(System.IntPtr dwComponentID, int reason = -1, int pvLoopData = 0) + 0x24d bytes
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoopInner(int reason = -1, System.Windows.Forms ApplicationContext context = {System.Windows.Forms ApplicationContext}) + 0x155 bytes
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoop(int reason, System.Windows.Forms ApplicationContext context) + 0x4a bytes
> System.Windows.Forms.dll!System.Windows.Forms.Application.Run(System.Windows.Forms.Form mainForm) + 0x31 bytes
    Computator.NET.exe!Computator.NET.Program.Main() Line 17          C#
[Native to Managed Transition]
[Managed to Native Transition]
mscorlib.dll!System.AppDomain.ExecuteAssembly(string assemblyFile, System.Security.Policy.Evidence assemblySecurity, string[] args) + 0x6b bytes
0x27 bytes
Microsoft.VisualStudio.HostingProcess.Utilities.dll!Microsoft.VisualStudio.HostingProcess.HostProc.RunUsersAssembly() + 0x27 bytes
mscorlib.dll!System.Threading.ThreadHelper.ThreadStart_Context(object state) + 0x6f bytes
mscorlib.dll!System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, object state, bool preserveSyncCtx) + 0xa7 bytes
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, object state, bool preserveSyncCtx) + 0x16 bytes
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, object state) + 0x41 bytes
mscorlib.dll!System.Threading.ThreadHelper.ThreadStart() + 0x44 bytes
[Native to Managed Transition]
```

Dodatek C. Błąd nr 2 znaleziony w klasie DataVisualization.Chart

```
A first chance exception of type 'System.OverflowException' occurred in System.Drawing.dll
Additional information: Overflow error.

Stack trace:
System.Drawing.dll!System.Drawing.Graphics.CheckErrorStatus(int status) + 0x58 bytes
System.Drawing.dll!System.Drawing.Graphics.DrawLine(System.Drawing.Pen pen, float x1, float y1, float x2, float y2) + 0
x46 bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.GdiGraphics.DrawLine(System.
Drawing.Pen pen, float x1, float y1, float x2, float y2) + 0x1a bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.ChartTypes.FastLineChart.Dra
wLine(System.Windows.Forms.DataVisualization.Charting.Series series, System.Windows.Forms.DataVisualization.Charting.DataPoint point
= {System.Windows.Forms.DataVisualization.Charting.DataPoint}, System.Windows.Forms.DataVisualization.Charting.DataPoint pointMin, Sy
stem.Windows.Forms.DataVisualization.Charting.DataPoint pointMax, int pointIndex = 1788, System.Drawing.Pen pen = {Color = {System.Dr
awing.Color}}, float firstPointX = 716.1116, float firstPointY = 1.51106478E+10, float secondPointX = 716.1116, float secondPointY =
-1.51106468E+10) + 0x25a bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.ChartTypes.FastLineChart.Pai
nt(System.Windows.Forms.DataVisualization.Charting.ChartGraphics graph = {System.Windows.Forms.DataVisualization.Charting.ChartGraphi
cs}, System.Windows.Forms.DataVisualization.Charting.CommonElements common, System.Windows.Forms.DataVisualization.Charting.ChartArea
area = {System.Windows.Forms.DataVisualization.Charting.ChartArea}, System.Windows.Forms.DataVisualization.Charting.Series seriesToD
raw) + 0xd33 bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.ChartArea.Paint(System.Windo
ws.Forms.DataVisualization.Charting.ChartGraphics graph = {System.Windows.Forms.DataVisualization.Charting.ChartGraphics}) + 0xce3 by
tes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.ChartPicture.Paint(System.Dr
awing.Graphics graph, bool paintTopLevelElementOnly = false) + 0xa77 bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Chart.OnPaint(System.Windows
.Forms.PaintEventArgs e = {ClipRectangle = {System.Drawing.Rectangle}}) + 0x40a bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.PaintWithErrorHandling(System.Windows.Forms.PaintEventArgs e, sho
rt layer) + 0x96 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.WmPaint(ref System.Windows.Forms.Message m) + 0x326 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.WndProc(ref System.Windows.Forms.Message m) + 0x348 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.ControlNativeWindow.OnMessage(ref System.Windows.Forms.Message m)
+ 0x11 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.ControlNativeWindow.WndProc(ref System.Windows.Forms.Message m) +
0x39 bytes
System.Windows.Forms.dll!System.Windows.Forms.NativeWindow.DebuggableCallback(System.IntPtr hWnd, int msg = 15, System.
IntPtr wparam, System.IntPtr lparam) + 0x5e bytes
[Native to Managed Transition]
[Managed to Native Transition]
System.Windows.Forms.dll!System.Windows.Forms.Application.ComponentManager.System.Windows.Forms.UnsafeNativeMethods.IMs
oComponentManager.FPushMessageLoop(System.IntPtr dwComponentID, int reason = -1, int pvLoopData = 0) + 0x24d bytes
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoopInner(int reason = -
1, System.Windows.Forms ApplicationContext context = {System.Windows.Forms ApplicationContext}) + 0x155 bytes
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoop(int reason, System.Windows.Forms
.ApplicationContext context) + 0x4a bytes
System.Windows.Forms.dll!System.Windows.Forms.Application.Run(System.Windows.Forms.Form mainForm) + 0x31 bytes
>     Computator.NET.exe!Computator.NET.Program.Main() Line 17           C#
[Native to Managed Transition]
[Managed to Native Transition]
mscorlib.dll!System.AppDomain.ExecuteAssembly(string assemblyFile, System.Security.Policy.Evidence assemblySecurity, st
ring[] args) + 0x6b bytes
Microsoft.VisualStudio.HostingProcess.Utilities.dll!Microsoft.VisualStudio.HostingProcess.HostProc.RunUsersAssembly() +
0x27 bytes
mscorlib.dll!System.Threading.ThreadHelper.ThreadStart_Context(object state) + 0x6f bytes
mscorlib.dll!System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext executionContext, System.T
hreading.ContextCallback callback, object state, bool preserveSyncCtx) + 0xa7 bytes
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext executionContext, System.Threading
.ContextCallback callback, object state, bool preserveSyncCtx) + 0x16 bytes
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext executionContext, System.Threading
.ContextCallback callback, object state) + 0x41 bytes
mscorlib.dll!System.Threading.ThreadHelper.ThreadStart() + 0x44 bytes
[Native to Managed Transition]
```

Error! Use the Home tab to apply Nagłówek 1 to the text that you want to appear here.

Dodatek D. Błąd nr 3 znaleziony w klasie DataVisualization.Chart

```
'System.Runtime.InteropServices.ExternalException' occurred in System.Drawing.dll
Additional information: A generic error occurred in GDI+.

Stack trace:
System.Drawing.dll!System.Drawing.Drawing2D.GraphicsPath.IsVisible(System.Drawing.PointF pt, System.Drawing.Graphics graphics) + 0x92 bytes
System.Drawing.dll!System.Drawing.Drawing2D.GraphicsPath.IsVisible(float x, float y) + 0x1b bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Selection.HitTest(int x, int y, bool ignoreTransparent = true, System.Windows.Forms.DataVisualization.Charting.ChartElementType[] requestedElementTypes) + 0x4bc bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Selection.EvaluateToolTip(System.Windows.Forms.MouseEventArgs e = {X = 264 Y = 262 Button = None}) + 0x48 bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Selection.Selection_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e) + 0x71 bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Chart.OnChartMouseMove(System.Windows.Forms.MouseEventArgs e = {X = 264 Y = 262 Button = None}) + 0x12b bytes
System.Windows.Forms.DataVisualization.dll!System.Windows.Forms.DataVisualization.Charting.Chart.OnMouseMove(System.Windows.Forms.MouseEventArgs e) + 0x5 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.WmMouseMove(ref System.Windows.Forms.Message m) + 0x69 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.WndProc(ref System.Windows.Forms.Message m) + 0x45c bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.ControlNativeWindow.OnMessage(ref System.Windows.Forms.Message m) + 0x11 bytes
System.Windows.Forms.dll!System.Windows.Forms.Control.ControlNativeWindow.WndProc(ref System.Windows.Forms.Message m) + 0x35 bytes
System.Windows.Forms.dll!System.Windows.Forms.NativeWindow.DebuggableCallback(System.IntPtr hWnd, int msg = 512, System.IntPtr wParam, System.IntPtr lParam) + 0x5e bytes
[Native to Managed Transition]
[Managed to Native Transition]
System.Windows.Forms.dll!System.Windows.Forms.Application.ComponentManager.System.Windows.Forms.UnsafeNativeMethods.IManager.PushMessageLoop(System.IntPtr dwComponentID, int reason = -1, int pvLoopData = 0) + 0x24d bytes
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoopInner(int reason = -1, System.Windows.Forms ApplicationContext context = {System.Windows.Forms ApplicationContext}) + 0x155 bytes
System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoop(int reason, System.Windows.Forms ApplicationContext context) + 0xa4 bytes
System.Windows.Forms.dll!System.Windows.Forms.Application.Run(System.Windows.Forms.Form mainForm) + 0x31 bytes
> Computator.NET.exe!Computator.NET.Program.Main() Line 17           C#
[Native to Managed Transition]
[Managed to Native Transition]
mscorlib.dll!System.AppDomain.ExecuteAssembly(string assemblyFile, System.Security.Policy.Evidence assemblySecurity, string[] args) + 0x6b bytes
Microsoft.VisualStudio.HostingProcess.Utilities.dll!Microsoft.VisualStudio.HostingProcess.HostProc.RunUsersAssembly() + 0x27 bytes
mscorlib.dll!System.Threading.ThreadHelper.ThreadStart_Context(object state) + 0x6f bytes
mscorlib.dll!System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, object state, bool preserveSyncCtx) + 0xa7 bytes
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, object state, bool preserveSyncCtx) + 0x16 bytes
mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, object state) + 0x41 bytes
mscorlib.dll!System.Threading.ThreadHelper.ThreadStart() + 0x44 bytes
[Native to Managed Transition]
```