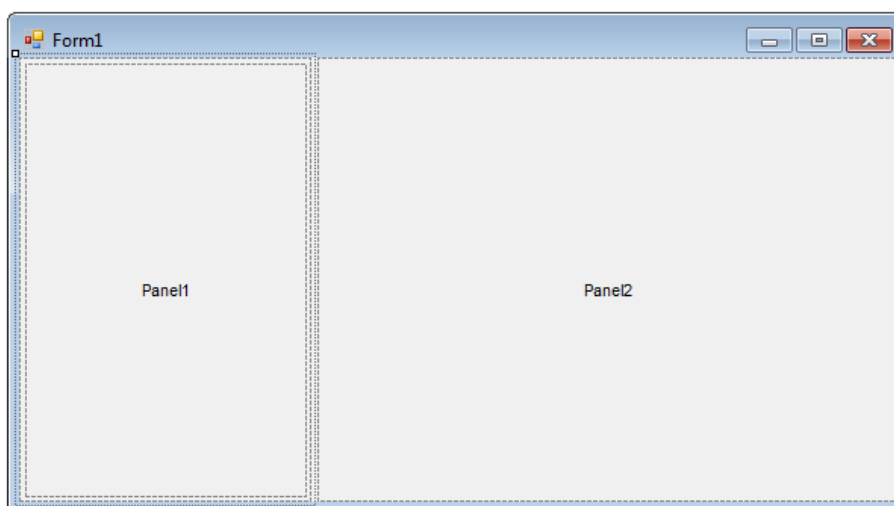


Wykres

SplitContainer

SplitContainer dzieli okno na dwie części (dwa panele), w poziomie albo w pionie; Zależnie od potrzeb umożliwia dowolną zmianę wielkości paneli przez użytkownika albo proporcjonalną zmianę wielkości paneli przy zmianie wielkości okna albo utrzymuje stały rozmiar jednego panelu.

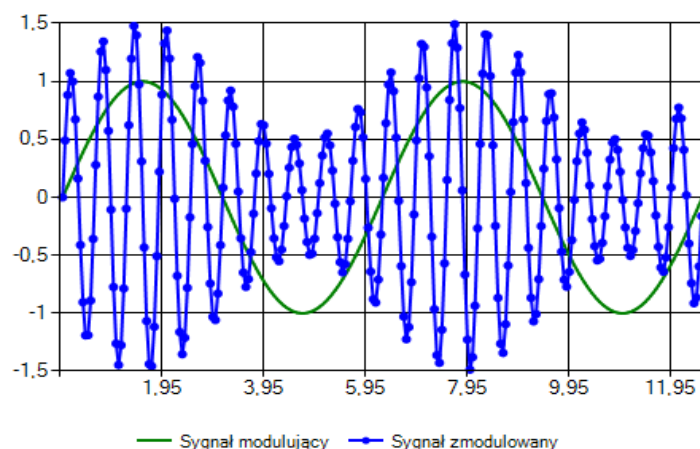


Właściwości:

- **Dock** – automatyczne dostosowanie wielkości i położenia, domyślnie *Fill*,
- **Orientation** – podział okna w poziomie lub pionie, domyślnie *Vertical* (w pionie),
- **FixedPanel** – panel zablokowany przy zmianie rozmiaru okna, możliwe wartości to *None*, *Panel1*, *Panel2*; Niezależnie od ustawienia, użytkownik może zmieniać rozmiar paneli (chyba że właściwość *IsSplitterFixed* jest ustawiona na *true*),
- **SplitterDistance** – odległość krawędzi podziału od krawędzi komponentu (lewej lub górnej, zależnie od *Orientation*)
- **Panel1MinSize**,
Panel2MinSize – minimalna wielkość paneli (szerokość lub wysokość),
- **IsSplitterFixed** – oba panele zablokowane, bez możliwości zmiany rozmiaru

Chart

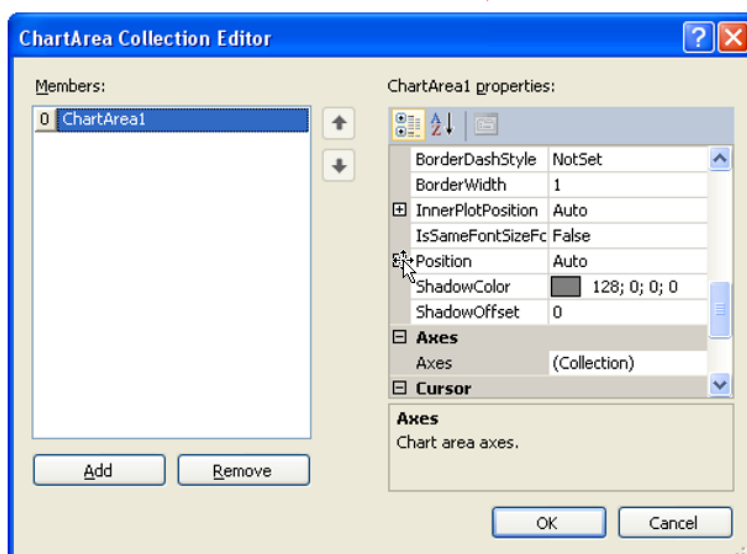
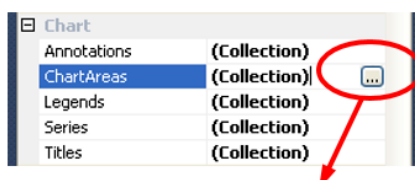
Komponent *Chart* pozwala tworzyć efektowne i zróżnicowane wykresy.



Na wykres składa się:

- Jeden lub więcej *Title* (tytuł)
- Jeden lub więcej *ChartArea* (obszar wykresu)
- Jedna lub więcej *Legend* (legenda)
- Jeden lub więcej *Series* (zbiór danych) na każdym z obszarów

Poszczególne elementy wykresu występują jako kolekcje, z których każda ma odrębny edytor właściwości.



Istnieje też możliwość programowego dodawania, modyfikowania oraz usuwania wszystkich elementów wykresu, ze wszystkich kolekcji:

```
ChartArea a;  
a = chart1.ChartAreas[0];  
a.AxisX.Minimum = 0.0;  
  
Series s;  
s = new Series();  
s.ChartType = SeriesChartType.Line;  
chart1.Series.Add (s);
```

- Elementy kolekcji są indeksowane od 0, a nie od 1.
- Należy dodać klauzulę *using*:
`using System.Windows.Forms.DataVisualization.Charting;`
(pozwala to stosować nazwy klas związanych z wykresem, np. *Series*, *ChartArea* itp.).

Właściwości

- **BackColor** – kolor tła; Inne elementy wykresu (np. poszczególne obszary) mogą mieć własne tło lub mogą być przezroczyste,
- **BackSecondaryColor**, **BackGradientStyle** – tło w postaci gradientu,
- **BackImage**, **BackImageWrapMode** – obrazek jako tło,
- **Palette** – paleta kolorów do prezentacji serii danych,
- **PaletteCustomColors** – własna paleta kolorów (jeżeli *Palette* jest *None*).

ChartArea

Obszar wykresu, mogący prezentować kilka serii danych

Właściwości

- **BackColor**, **BackImage** itp. – tło,
- **Axes** – kolekcja osi wykresu (w praktyce lepiej odwoływać się bezpośrednio do poszczególnych osi),
- **AxisX**, **AxisX2** – główna i pomocnicza oś X,
- **AxisY**, **AxisY2** – główna i pomocnicza oś Y.

Axis

Oś liczbowa wykresu.

- Każda seria danych przypisana do obszaru wykresu, może być przypisana do głównej lub pomocniczej osi X i osi Y,
- Każda oś ma kilkadziesiąt (!) właściwości określających jej wygląd (kolor, podział na zakresy, etykiety, tytuł, ...).

Wybrane właściwości

- **Minimum**, **Maximum** – domyślnie "Auto", ale można podać
- **IsLogarithmic** – skala logarytmiczna

Series

Seria (zbiór) danych prezentowanych na wykresie.

- Każda seria jest przypisana do jednego z obszarów danych, a także do głównej lub pomocniczej osi X i osi Y tego obszaru,
- Każda seria może mieć inny typ (kolumny, linie, punkty, ...), jednak nie wszystkie kombinacje są możliwe.

Wybrane właściwości

- **ChartType** – rodzaj wykresu (np. *Line*, *Point*, *Column*, ...),
- **ChartArea** – obszar, na którym seria jest wyświetlana,
- **XAxisType**, **YAxisType** – przypisanie do osi *Primary* / *Secondary*,
- **LegendText** – opis wyświetlany w legendzie,
- **BorderWidth** – dla typu *Line* – grubość linii,
- **MarkerStyle** – rodzaj znacznika (np. *Square*, *Circle*, ...),
- **Enabled** – dostępność (widoczność) serii na wykresie, odpowiednik *Visible*.

Legend

Legenda wykresu

- Może być wspólna dla wszystkich obszarów albo każdy obszar może mieć oddzielną; Serie danych można przypisywać do wybranej legendy.

Wybrane właściwości

- **DockedToChartArea** – przypisanie do obszaru,
- **IsDockedInsideChartArea** – czy legenda ma być wewnątrz obszaru, czy obok niego,
- **Docking** – po której stronie obszaru ma być legenda (*Top*, *Right*, *Left*, *Bottom*),
- **Alignment** – sposób wyrównania (*Near*, *Center*, *Far*),
- **Enabled** – dostępność (widoczność) legendy, odpowiednik *Visible*.

Wykorzystanie komponentu Chart

Można rozważyć różne strategie prezentacji danych i wykorzystania komponentu *Chart*. W najprostszym rozwiązaniu na etapie projektowania GUI są dostosowane właściwości obszaru wykresu i legendy oraz są utworzone i odpowiednio dostosowane wszystkie potrzebne serie, aplikacja tylko oblicza wartości, które mają być prezentowane na wykresie, po czym dodaje je do odpowiednich serii danych. W najbardziej złożonych aplikacjach na etapie projektowania GUI są dostosowane jedynie właściwości obszaru wykresu, ale nie ma żadnych serii danych, aplikacja dynamicznie dodaje serie danych zgodnie z wyborem użytkownika – należy wówczas programowo utworzyć i zmienić właściwości serii.

Do ćwiczeń zostanie zastosowana strategia najprostsza, ale z możliwością wzbogacenia aplikacji o dynamiczną zmianę wybranych właściwości wykresu po uruchomieniu programu, np. włączenie lub wyłączenie znaczników, ukrycie legendy itp.

Zakładając, że są dostosowane właściwości obszaru wykresu i legendy oraz są utworzone i odpowiednio dostosowane wszystkie potrzebne serie, aby wyświetlić nowy wykres, należy postępować według schematu:

- Zapisać w zmiennych referencje do wszystkich serii danych, np.:

```
Series s1, s2;  
s1 = chart1.Series[0];  
s2 = chart1.Series[1];
```

- Usunąć uprzednio wyświetlane dane – należy skorzystać z metody *Clear* kolekcji *Points*; Można użyć referencji do serii danych albo pętli *foreach*, np.:

```
s1.Points.Clear();  
s2.Points.Clear();
```

- Odczytać parametry obliczeń, np.:

```
Double tau;  
tau = Double.Parse(textBox.Text);
```

- Dodać nowe dane do wykresu – należy użyć pętli *for* (albo *while*) oraz metody *AddXY* kolekcji punktów wykresu *Points*, np.:

```
for(x=0.0; x<=10.0; x+=0.25)  
{  
    s1.Points.AddXY(x, Math.Sqrt(tau * x));  
    s2.Points.AddXY(x, Math.Log(tau * x));  
}
```

Programowa zmiana właściwości:

- Właściwości konkretnych, pojedynczych elementów – przez referencję i indeks:

```
Series s;
```

```
s = chart1.Series[0];  
s.ChartType = SeriesChartType.Line;  
s.MarkerStyle = MarkerStyle.None;
```

```
ChartArea a;
```

```
a = chart1.ChartAreas[0];  
a.AxisX.Minimum = 0;  
a.AxisX.Maximum = 5;
```

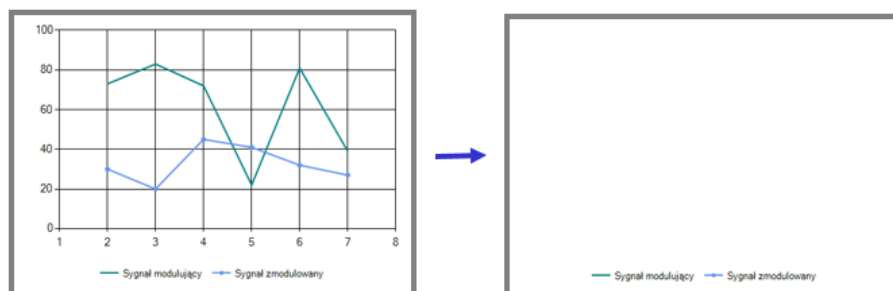
- Właściwości wszystkich elementów kolekcji – przez pętlę *foreach*:

```
foreach (Series s in chart1.Series)  
{  
    s.ChartType = SeriesChartType.Line;  
    s.MarkerStyle = MarkerStyle.Circle;  
}
```

```
foreach (Series s in chart1.Series)  
{  
    s.Points.Clear();  
}
```

Znikający wykres?

Po umieszczeniu komponentu na formularzu można go dostosować do potrzeb aplikacji – dodać i dostosować obszary, serie danych, osie, legendę itd. W tym czasie na wykresie są przedstawione przykładowe dane, jednak po uruchomieniu programu widać tylko legendę. Dzieje się tak, ponieważ serie nie zawierają żadnych danych; Dopiero po dodaniu danych pojawiają się one na wykresie;



Strumienie i pliki

Języki C/C++/C# stosują model komunikacji ze światem zewnętrznym oparty na plikach i strumieniach

- Plikiem jest nie tylko plik na dysku, ale też konsola, port szeregowy, port równoległy, drukarka, wirtualne połączenie między programami (tzw. *pipeline*), itd.
- Strumień jest programową reprezentacją pliku, definiuje dostępne operacje wejścia/wyjścia

Takie podejście umożliwia m.in. przetwarzanie potokowe, czyli przekierowanie strumieni wejścia/wyjścia pomiędzy różnymi programami i plikami, np.:

```
dir | more
dir > plik.txt
```

Pliki można podzielić na 2 kategorie:

- Pliki tekstowe – zawierają wyłącznie kody znaków, w określonym rodzaju ich kodowania (np. Unicode, ISO, Windows, ...), oraz kody sterujące (jak np. CR, LF),
- Pliki binarne – pliki o dowolnej zawartości, oprócz tekstowych;
Zazwyczaj sposób reprezentacji danych w plikach binarnych jest taki sam, jak w pamięci operacyjnej komputera – do plików binarnych można zapisać wartości dowolnych zmiennych bez ich konwersji na tekst, w dokładnie takiej samej postaci, w jakiej istnieją w programie;

Zapis i odczyt plików binarnych jest szybszy (nie ma konwersji z/na tekst) i mają one mniejszą objętość, ale ich obsługa jest bardziej złożona i bardziej podatna na błędy

Klasa Encoding

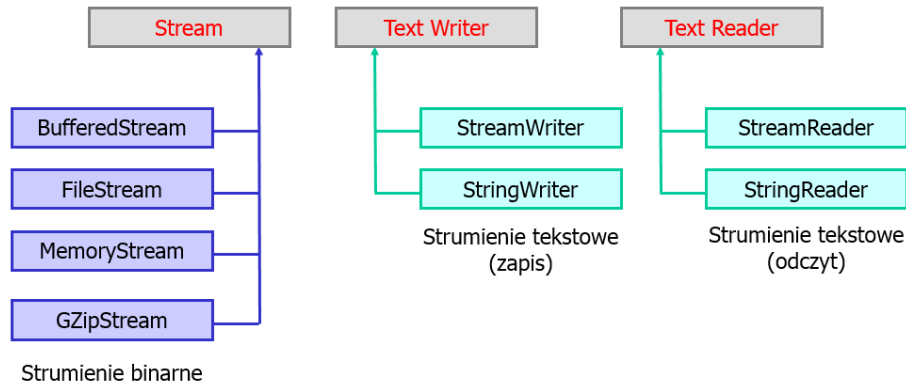
Encoding to klasa statyczna, reprezentuje sposób kodowania znaków

- Kodowanie dotyczy wyłącznie odczytu i zapisu tekstu poza program C# - wewnątrz znaki i łańcuchy są zawsze zapisane w Unicode (UTF-16)
- Nie można utworzyć obiektu klasy *Encoding* przy pomocy operatora *new*, zamiast tego można wykorzystać kodowania standardowe, dostępne jako właściwości klasy *Encoding*:

- `Encoding.ASCII` - kod ASCII 7-bitowy
- `Encoding.Unicode` - Unicode, kodowanie UTF-16
- `Encoding.UTF8` - Unicode, kodowanie UTF-8

Hierarchia klas strumieni

Strumienie w C# zostały podzielone na kilka kategorii, m.in.: *Stream* – strumień binarny, obsługujący zapis i odczyt danych, *TextWriter* – strumień tekstowy, tylko do zapisu danych, *TextReader* – strumień tekstowy, tylko do odczytu danych



Wszystkie strumienie (a także inne klasy związane z plikami i katalogami, np. *File*, *Directory*, *Path*) są zdefiniowane w przestrzeni nazw *System.IO*.

StreamReader

StreamReader jest strumieniem tekstowym, służącym do odczytu plików

- Konstruktor *StreamReader* ma kilka wersji, m.in.:

```
StreamReader(
    String path,
    Encoding encoding)
```

- `path` – kwalifikowana nazwa pliku,
- `encoding` – kodowanie znaków, domyślnie `Encoding.UTF8`;

- Metody *Read* i *ReadLine* – odczytują tekst; Działają identycznie, jak metody klasy *Console*, tyle że czytają z pliku,
- Właściwość *EndOfStream* – pozwala sprawdzić, czy została już odczytana cała zawartość pliku,
- Metoda *Close* – zamyka plik; Dopóki plik jest otwarty, jest zablokowany dla innych programów i zużywa zasoby – kiedy plik nie jest już potrzebny, należy go zamknąć.

```
StreamReader sr;
String ln;

sr = new StreamReader(openFileDialog.FileName);

while (!sr.EndOfStream)
{
    ln = sr.ReadLine();
}

sr.Close();
```

StreamWriter

StreamWriter jest strumieniem tekstowym, służącym do zapisu plików

- Konstruktor *StreamWriter* ma kilka wersji, m.in.:

```
StreamWriter(  
    String path,  
    Boolean append,  
    Encoding encoding)
```

- `path` – kwalifikowana nazwa pliku;
- `append` – dopisywanie do pliku, domyślnie *false*;
- `encoding` – kodowanie znaków, domyślnie `Encoding.UTF8`;

Jeżeli plik nie istnieje, to zostanie utworzony, jeżeli istnieje i *append* ma wartość *false* to zostanie usunięty i utworzony na nowo, jeżeli istnieje i *append* ma wartość *true* to zostanie otwarty i tekst będzie dopisywany na jego końcu.

- Metody *Write* i *WriteLine* – zapisują tekst; Działają identycznie, jak metody klasy *Console*, tyle że zapisują do pliku,
- Metoda *Close* – zamyka plik; Dopóki plik jest otwarty, jest zablokowany dla innych programów i zużywa zasoby – kiedy plik nie jest już potrzebny, należy go zamknąć.

```
StreamWriter sw;  
  
sw = new StreamWriter(saveFileDialog.FileName);  
  
sw.WriteLine("Dziś mamy {0}", DateTime.Now);  
  
sw.Close();
```

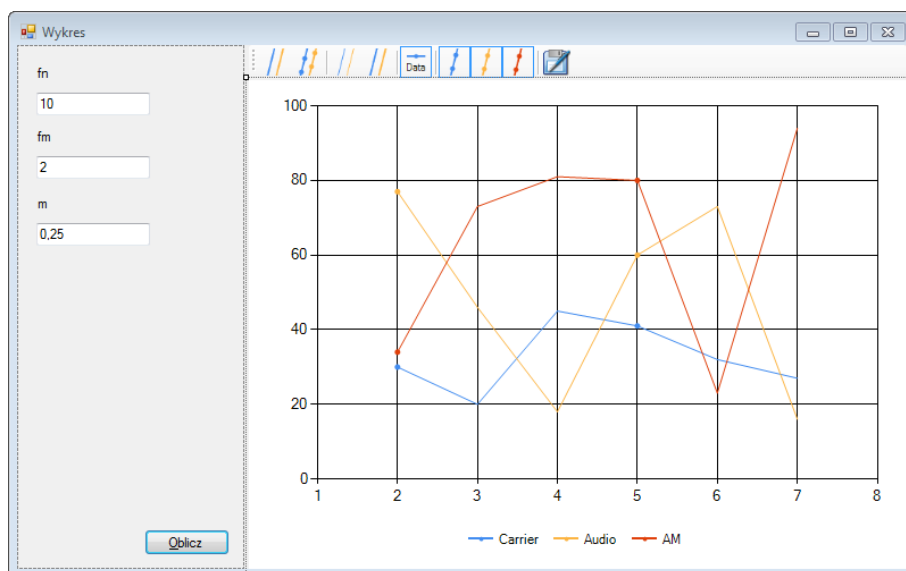
Zadania

Celem ćwiczenia jest opracowanie programu wyświetlającego wykresy wybranych funkcji, z możliwością edycji parametrów funkcji oraz, opcjonalnie, wzbogaconego o programową zmianą wybranych właściwości wykresu.

Podstawowe funkcje programu

1. Okno i zasoby aplikacji

- Utworzyć nowy projekt, język C#, typ projektu Windows Forms,
- Umieścić na formularzu komponenty wg wzoru na rysunku poniżej: *SplitContainer*, następnie pola tekstowe z etykietami na parametry funkcji i przycisk *Oblicz* w jednym panelu, a pasek narzędzi i komponent *Chart* w drugim panelu kontenera,
- Ustawić właściwości okna i komponentów: okno powinno mieć odpowiedni tytuł, kontener powinien mieć zablokowany panel z polami tekstowymi (z możliwością regulacji rozmiaru przez użytkownika lub bez takiej możliwości), wykres powinien zajmować całe dostępne miejsce, przycisk powinien być domyślny dla formularza, etykiety i przycisk powinny mieć skróty klawiaturowe, a pola tekstowe powinny zawierać domyślne wartości parametrów funkcji,
- Do zasobów aplikacji dodać obrazki, do wykorzystania w pasku narzędzi.



2. Wykres

- Ustawić odpowiednio właściwości komponentu *Chart*, np. paletę kolorów,
- Ustawić odpowiednio właściwości legendy wykresu, np. jej położenie,
- Ustawić odpowiednio właściwości głównej osi X, np. minimum i maksimum, **Ważne!** Przy zakresie osi X 0...1 przykładowy wykres zniknie (ma wartości x z zakresu od 2...7); Aby zobaczyć efekty dostosowania serii danych, ten krok można wykonać na końcu, już po dostosowaniu serii, albo realizować go programowo,
- Do komponentu *Chart* dodać odpowiednią liczbę serii danych i ustawić odpowiednio ich właściwości – wykres typu *Line* i dodatkowo włączone wyświetlanie znaczników, należy też wpisać tekst legendy, odpowiednio do obliczanych w programie funkcji.

3. Obliczenia

- Dodać obsługę kliknięcia przycisku *Oblicz*, w której należy (1) usunąć wcześniej wyświetlone dane, przy pomocy funkcji *Clear* kolekcji *Points*, kolejno dla wszystkich serii danych, (2) odczytać wartości parametrów funkcji, konwertując je do właściwego typu, zazwyczaj *Double*, oraz (3) wyświetlić wykres, obliczając i dodając wartości do kolekcji *Points* wszystkich serii danych, w pętli *for* lub *while*
- Obliczenia wykonać dla zmiennej niezależnej z zakresu od 0 do 1, z krokiem co 0,001.

Propozycje funkcji do obliczeń:

- Modułacja amplitudy:
 $n(t) = \sin(2\pi f_n t)$
 $u(t) = \sin(2\pi f_m t)$
 $s(t) = (1 + m \cdot u(t)) \cdot n(t)$
gdzie: $n(t)$ – sygnał nośny, $u(t)$ – sygnał modulujący, $s(t)$ – sygnał zmodulowany,
 f_n – częstotliwość nośna, f_m – częstotliwość m.cz., m – index modulacji, $0 < m \leq 0,5$,
domyślne wartości: $f_n = 20$, $f_m = 2$, $m = 0,5$
- Modułacja fazy:
 $n(t) = \sin(2\pi f_n t)$
 $u(t) = \sin(2\pi f_m t)$
 $s(t) = \sin(2\pi f_n t + m \cdot u(t))$
gdzie: $n(t)$ – sygnał nośny, $u(t)$ – sygnał modulujący, $s(t)$ – sygnał zmodulowany,
 f_n – częstotliwość nośna, f_m – częstotliwość m.cz., m – index modulacji, $m > 0$,
domyślne wartości: $f_n = 20$, $f_m = 2$, $m = 2$
- Drgania narastające:
 $o(t) = 1 - \exp(-\tau \cdot t)$
 $u(t) = o(t) \cdot \cos(2\pi f t)$
 $i(t) = o(t) \cdot \sin(2\pi f t)$
gdzie: $o(t)$ – obwiednia, $u(t)$ – napięcie, $i(t)$ – prąd,
 f – częstotliwość rezonansowa, τ – stała czasowa,
domyślne wartości: $f = 20$, $\tau = 5$

4. ** Dodatkowe parametry obliczeń

- Dodać dodatkowe pola tekstowe, z wartościami domyślnymi i etykietami:
 - zakres obliczeń – wartości minimum i maksimum zmiennej niezależnej,
 - liczba kroków obliczeń *albo* odstęp między wartościami zmiennej niezależnej,
- Zmodyfikować kod w funkcji kliknięcia przycisku, aby uwzględnić te parametry
Ważne! Przy zmianie zakresu obliczeń, należy zmienić nie tylko same obliczenia, ale także zmienić programowo właściwości głównej osi X wykresu.

Ustawienia aplikacji

Należy wybrać jeden wariant zmiany wyświetlania serii danych, tj. zadanie 6 *albo* 7.

W zadaniach 6, 7, 9 i 10 najlepiej jest użyć pętli *foreach*, ale można też użyć indeksów serii danych i powtarzać czynności kolejno dla wszystkich serii danych

5. * Ukrywanie legendy

- Do paska narzędzi dodać przycisk ukrywania legendy, przekształcić go w *CheckBox* – ustawić właściwości *Check* oraz *CheckOnClick* na *true*,
- Utworzyć funkcję obsługi zdarzenia *Click* dla tego przycisku i dodać instrukcję ukrywającą lub pokazującą legendę wykresu.

6. * Ukrywanie znaczników

- Do paska narzędzi dodać przycisk ukrywania znaczników, przekształcić go w *CheckBox* – ustawić właściwości *Check* oraz *CheckOnClick* na *true*,
- Utworzyć funkcję obsługi zdarzenia *Click* dla tego przycisku i dodać instrukcję ukrywającą lub pokazującą znaczniki wszystkich serii danych.

7. ** Wybór typu wykresu

- Do paska narzędzi dodać przyciski wyboru typu wykresu (tylko linia, linia i znaczniki, tylko znaczniki), ustawić właściwość *Check* pierwszego z tych przycisków na *true*, **Ważne!** Wszystkie te przyciski muszą mieć właściwość *CheckOnClick* ustawioną na *false*, ponieważ mają się zachowywać jak *RadioButton*, a nie jak *CheckBox*,
- Utworzyć funkcje obsługi zdarzenia *Click* dla tych przycisków. W każdej z nich należy zmieniać ustawienia wszystkich serii danych, ale też wartość właściwości *Check* przycisków – klikniętego na *true*, pozostałych dwóch na *false*.

8. * Ukrywanie serii danych

- Do paska narzędzi dodać przyciski ukrywania serii danych – tyle, ile serii jest na wykresie, następnie każdy z nich przekształcić w *CheckBox* – ustawić właściwości *Check* oraz *CheckOnClick* na *true*,
- Utworzyć funkcje obsługi zdarzenia *Click* dla tych przycisków i dodać instrukcje ukrywające lub pokazujące odpowiednią do przycisku serię danych.

9. * Zmiana grubości linii

- Do paska narzędzi dodać przycisk grubości linii, przekształcić go w *CheckBox* – ustawić właściwości *Check* na *false* oraz *CheckOnClick* na *true*,
- Utworzyć funkcję obsługi zdarzenia *Click* dla tego przycisku i dodać instrukcję zmieniającą szerokość linii wszystkich serii danych, np. na 1 kiedy przycisk jest zwolniony albo na 2 kiedy jest wciśnięty.

10. * Zmiana wielkości znaczników

- Do paska narzędzi dodać przycisk wielkości znaczników, przekształcić go w *CheckBox* – ustawić właściwości *Check* na *false* oraz *CheckOnClick* na *true*,
- Utworzyć funkcję obsługi zdarzenia *Click* dla tego przycisku i dodać instrukcję zmieniającą wielkości znaczników wszystkich serii danych, np. na 4 kiedy przycisk jest zwolniony albo na 6 kiedy jest wciśnięty.

Funkcje zaawansowane

11. *** Eksport wyników do pliku CSV

- Dodać do projektu komponent *SaveFileDialog* i ustawić odpowiednio jego opcje:
 - **Filter** – tylko jeden rodzaj plików – pliki CSV,
 - **DefaultExtension** – ".csv" (kropka csv),
- Do paska narzędzi dodać przycisk eksportu wyników,
- Utworzyć funkcję obsługi zdarzenia *Click* dla tego przycisku i dodać kod zapisywania wyników obliczeń do pliku CSV, wskazanego przez *SaveFileDialog*,
Ważne! Najprościej będzie skopiować kod funkcji obsługi kliknięcia przycisku *Oblicz* i tylko dodać kod zapisania wyników do pliku przy użyciu strumienia *StreamWriter*,
- Sprawdzić możliwość wczytywania pliku CSV przez program Excel pakietu Office,
- Sprawdzić poprawność formatu liczb w pliku CSV, np. tworząc wykres.

Ważne! W polskiej wersji językowej pakietu Office, w plikach CSV (ang. *comma-separated values*, wartości rozdzielone przecinkiem), wbrew nazwie, wartości należy oddzielać średnikami; Każda linia pliku CSV powinna zawierać wartości jednego wiersza, np. `x;y1;y2`; Łańcuchy tekstowe niezawierające średnika nie wymagają specjalnych zabiegów, ale zawierające go powinny być zapisane w cudzysłowach (w języku C# należy użyć sekwencji ucieczki, `\`). Do zapisania wartości zmiennych warto użyć łańcucha formatującego – działa tak samo, jak w przypadku klasy *Console*. Przed średnikiem i po nim nie powinno być spacji, chyba że spacja należy do danych.

```
sw.WriteLine("czas;m. cz.;w. cz.");  
sw.WriteLine("{0};{1};{2};", x, y1, y2);
```

