Politechnika Warszawska

WYDZIAŁ SAMOCHODÓW
I MASZYN ROBOCZYCH

Praca dyplomowa inżynierska

na kierunku Mechanika Pojazdów i Maszyn Roboczych
w specjalności Wspomaganie Komputerowe Prac Inżynierskich

Aplikacja webowa CAE dedykowana do projektowania wałów maszynowych

numer pracy według wydziałowej ewidencji prac: 115B-ISP-MM/275446/1190838

Jan Wojciech Zembowicz

numer albumu 275446

promotor
dr inż. Krzysztof Twardoch, dr hab. inż. Piotr Skawiński

konsultacje
—

WARSZAWA 2019

Jan Zembowicz

OŚWIADCZENIE

Jako autor pracy dyplomowej pt.: ,,Aplikacja webowa CAE przeznaczona do projektowania wałów maszynowych", którą wykonałem samodzielnie przestrzegając zasad ochrony własności intelektualnej zezwalam na publiczne udostępnienie pracy i wyrażam zgodę na jej udostępnienie w Bibliotece Wydziału Samochodów i Maszyn Roboczych Politechniki Warszawskiej w ramach realizacji zadań statutowych biblioteki.

Warszawa, dnia 26.04.2019

SŁOWA KLUCZOWE

1. Aplikacja internetowa
2. Wał maszynowy
3. Metoda elementów skończonych
4. Analiza składniowa wyrażeń algebraicznych

STRESZCZENIE:

Obecnie dostępne rozwiązania CAD/CAE nie zapewniają odpowiednich funkcjonalności potrzebnych do prostej i szybkiej weryfikacji zadań projektowych wałów maszynowych. Potrzebne było stworzenie intuicyjnego systemu, który można uruchomić na dowolnej platformie i nie wiązałaby się z dodatkowymi kosztami takimi jak zakup oprogramowania CAE.

Shaft.js wykorzystuje do działania technologie typowe dla stron internetowych. Interfejs programu może dostosować się do wyświetlania na dowolnym typie urządzenia i ekranu. Funkcje programu są rozmieszczone w menu w postaci zakładek HTML.

Aplikacja udostępnia edytor stanu obciążenia, pozwala definiować je na 2 sposoby. Udostępnia również edytor kształtu, z funkcjami wspomagającymi proces doboru parametrów geometrycznych.

Moduł do obliczeń wytrzymałościowych wykorzystuje analizator składniowy przetwarzający automatycznie generowane ciągi znakowe zawierające równania momentów.

Do obliczeń sztywnościowych wykorzystana jest metoda elementów skończonych, z algorytmem automatycznie dzielącym wał na elementy skończone, na podstawie jego kształtu i obciążenia

Wyniki obliczeń, dzięki zastosowanym metodom, są niemal identyczne, jak wyniki obliczeń analitycznych.

SUMMARY:

Currently available CAD/CAE solutions' functionalities are not a simple and efficient enough for didactic mechanical design tasks.

Shaft.js uses typical web technologies and solutions. Interface of the application, can be displayed on any type of device with a web browser.

Application contains rich load editor with 2 ways of defining it. Program also has deeply automated shape editor.

Strength calculation module uses the algebraic parser analysing the torque equations, which are created with the load data arrays.

Stiffness module uses the finite element analysis with automatic discretization.

Calculation results are almost the same, as the analytic methods results, thanks to the workflow and calculation methods of the application.

1. Wstęp

1.1 Przedmowa

Metodyka pracy inżynierów zmienia się nieustająco. Nieodzownymi narzędziami są kartka papieru i długopis (kiedyś narzędzia jedyne). Na przestrzeni czasu wraz z postępem i rozwojem nauki powstawały nowe. Tyczy się to używanych narzędzi, ale też metodyki rozwiązań problemów. Współcześnie trudno sobie wyobrazić pracę inżyniera, która nie opierałaby się na narzędziach CAx, czyli komputerowo wspomaganych systemach do projektowania geometrii (CAD), procesów technologicznych i ich składowych (CAM), oraz systemów opartych na metodzie elementów skończonych (FEA). Niemniej jednak do projektowania tak podstawowych klas części maszyn, jak belki, koła zębate czy wały i osie maszynowe, często stosuje się metody klasyczne.

1.2. Cel

Celem pracy, było stworzenie prostego systemu projektowania wałów maszynowych z warunku wytrzymałościowego i warunku sztywnościowego, którego głównym zadaniem jest weryfikacja zadań projektowych na przedmiotach: Projektowanie Podstaw Konstrukcji Maszyn oraz Projektowanie Napędów Mechanicznych. Rozwiązanie ma być alternatywą dla używanego dotychczas programu “Wał ‘99”. Program został nazwany ,,Shaft.js’’. Nazwa stanowi nawiązanie do poprzednika oraz technologii w jakiej został napisany.

Niniejsza praca jest swego rodzaju dokumentacją techniczną programu, kataloguje i wyjaśnia użyte w nim technologie internetowe, algorytmy, struktury danych, a także zasady i praktyki projektowe, stosowane podczas budowy i projektowania maszyn.

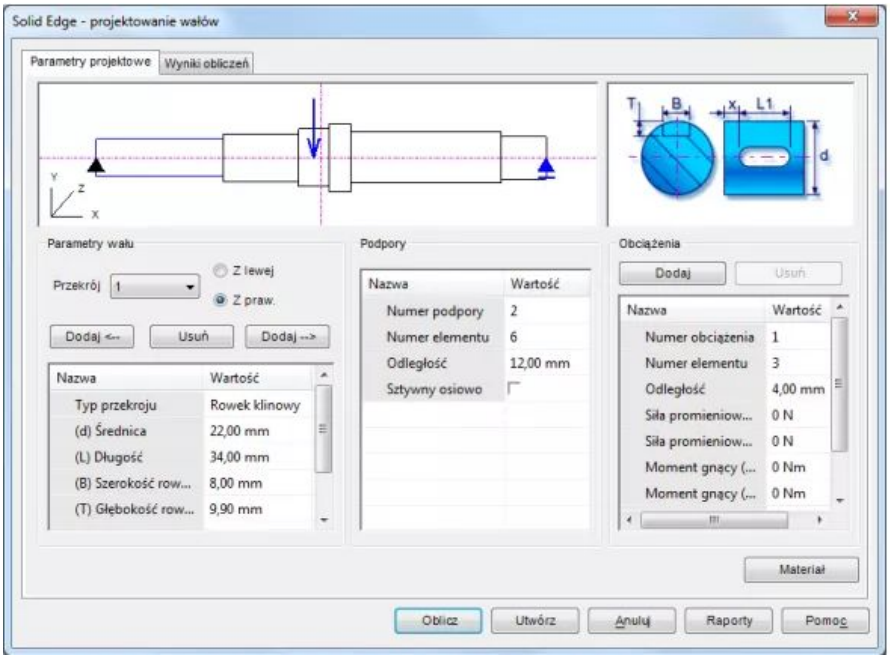
1.3. Założenia projektu

• Program musi mieć możliwość uruchomienia na dowolnym systemie operacyjnym, dowolnej architekturze sprzętu, oraz rozdzielczości ekranu.
• Program nie może wymagać żadnych dodatkowych programów do stworzenia pełnego raportu z obliczeń.
• Należy zachować pełną kompatybilność zarówno ze starszymi urządzeniami, jak z tymi, które dopiero powstaną w przyszłości.
• Proces wprowadzania danych, oraz jego poprawki muszą być intuicyjne i wymagać jak najmniejszej ilości dodatkowych instrukcji, interfejs musi być schludny i czytelny.
• Program musi być w stanie zweryfikować przynajmniej wszystkie zadania projektowe zadawane na wyżej wymienionych przedmiotach konstruktorskich.

2. Przegląd wspomaganych komputerowych rozwiązań dostępnych na rynku oraz w praktyce inżynierskiej

2.1. SolidEdge ST10 - Podręcznik inżyniera

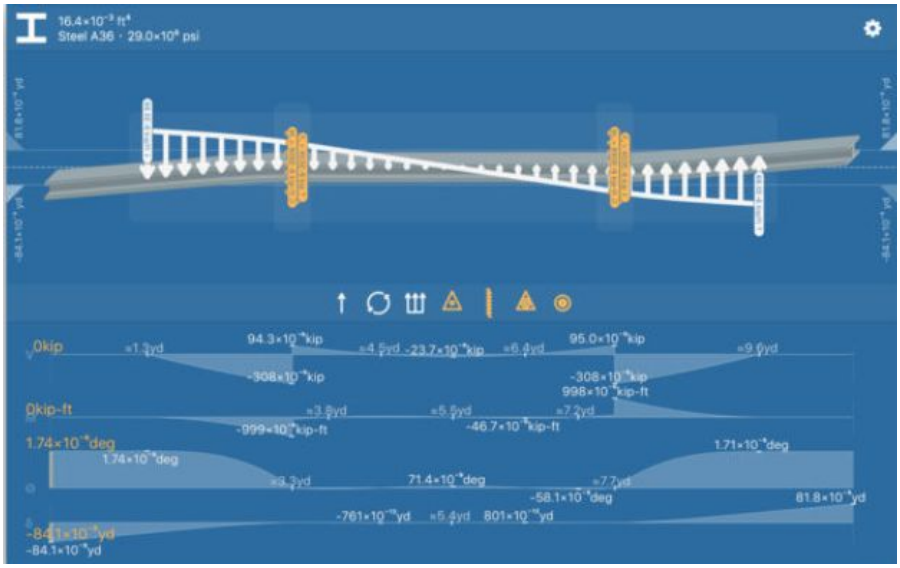
Jedną z funkcji zaawansowanego, aczkolwiek niezbyt popularnego w Polsce programu CAD jest Podręcznik Inżyniera z pakietu SolidEdge (rys. 2.1). To zestaw skryptów pozwalających na projektowanie podstawowych elementów, takich jak przekładnie, belki itp. Zaletą systemu jest podgląd parametrów wytrzymałościowych wału w czasie rzeczywistym, takich jak: przebieg momentu gnącego, lub kąt ugięcia itd. Pozwala również na obliczenia wałów statycznie niewyznaczalnych. Wadą programu jest trudna dostępność, gdyż wymaga on licencji na cały system CAD SolidEdge oraz fakt, że jest on trudno dostępny w interfejsie. Program jest dostępny tylko na systemy Microsoft Windows.



Rys. 2.1. Podręcznik inżyniera aplikacji SolidEdge (https://cador.pl/solid-edge-engineering-reference.html)

2.2. Deflection

Innym programem wspomagającym pracę inżyniera jest Deflection. To aplikacja na urządzenia mobilne, służąca do analizy belek. Podgląd wszystkich wielkości również na żywo. Wygodny interfejs (rys. 2.2), dostosowany do nowoczesnych pojemnościowych multi-dotykowych ekranów urządzeń mobilnych. Wadą programu jest możliwość analizowania jedynie prostych stanów obciążenia.



Rys. 2.2. Widok aplikacji "Deflection" (https://appsliced.co/app?n=beam-deflection-for-mechanical-engineering)

2.3. Wał maszynowy '99

Wał maszynowy '99 (rys. 2.3) to praca dyplomowa absolwentów wydziału Samochodów i Maszyn Roboczych. To zaawansowany program, jak na swoje czasy i wykorzystywaną technologie. Pozwala na projektowanie części z warunku wytrzymałościowego jak i sztywnościowego, dzięki prostemu modułowi MES (elementy belkowe, jeden stopień to jeden element, chyba, że to stopień konstrukcyjny, czyli, w tym kontekście, przenosi obciążenie). Obciążenie jest definiowane przez "płaszczyzny obciążenia", które są prostymi modelami dowolnego typu przekładni.

Rys. 2.3. Ekran powitalny programu Wal '99

3. Biblioteki, rozszerzenia i technologie informatyczne wykorzystywane w programie Shaft.js

3.1. XHTML5 [8]

```
<!DOCTYPE HTML>
<html>
        <head>
                <title>Moja Strona</title>
         </head>
         <body>
                <p>.....</p>
                <img src = ” … ”>.....</img>
        </body>
 </html>
```

XHTML (ang. eXtensible HyperText Markup Language) to rozszerzalny hipertekstowy język znaczników. Język ten pozwala stworzyć strukturę danych wewnątrz strony internetowej i innych dokumentów z rozszerzeniem .html/.htm, nadając znaczenie poszczególnym fragmentom tekstu, poprzez umieszczanie ich w, tzw. znacznikach. Dzięki temu formowane są hiperłącza, akapity, nagłówki, listy oraz osadzone w tekście dokumentu obiekty plikowe np. multimedia bądź elementy baz danych np. interaktywne formularze.

XHTML ma swoje początki w 1980 roku, kiedy pracujący w CERN fizyk Tim Bernes Lee stworzył nowy system zapisu dokumentów. Pierwsza wersja posiadała 22 znaczniki. Trzynaście z nich istnieje do tej pory w specyfikacji ()normie obecnej wersji.

Język HTML składa się z kilku kluczowych składowych elementów [6]:
• znaczników i ich atrybutów
• typów danych
• referencji znakowych
• odwołań w postaci encji
• deklaracji typu dokumentu

Jednymi z kluczowych atrybutów są `class`, definiujący klasy znaczników (jedna klasa - zestaw dowolnej ilości identycznych właściwości i cech, analogicznie do klas w konwencjonalnych językach programowania), oraz `id`, dzięki któremu możliwe jest zdefiniowanie właściwości unikalnych obiektów. Dobrą praktyką jest stosowanie `class` do stylowania w CSS, a `id` do podpinania znaczników do JavaScript (patrz niżej). Obecnie obowiązującym standardem jest, wykorzystany w projekcie, HTML 5.

3.2. CSS3 [8]

```
body {
        background-color: lightblue;
}
h1{
        color: white;
        text-align: center;
 }
```

CSS - Cascading Style Sheets (z ang. Kaskadowe Arkusze Stylów), nietypowy język opracowany w jednym celu. CSS stosuje się do agregacji danych o wyglądzie całej strony internetowej, podzielonej na wiele plików HTML.

Kaskadowość CSS oznacza, że każdy arkusz ma przypisany priorytet. Zależy on, od miejsca zdefiniowania stylu w kodzie. Przykładowo, styl zakodowany w znaczniku jest ważniejszy niż styl zakodowany w sekcji <head>, który jest ważniejszy, niż styl zakodowany w pliku CSS. Stylowany element przyjmuje taki wygląd, jaki jest zdefiniowany przez arkusz o najwyższym priorytecie. Możliwe jest złamanie tego porządku, poprzez dodanie parametru !important, nie jest to jednak dobra praktyka.

Przed opracowaniem CSS'a stylowanie było przeprowadzane poprzez ustawianie odpowiednich atrybutów w unikalnych znacznikach. Wadą tego rozwiązania jest fakt, że żeby zmienić cechę (na przykład kolor) danego elementu, należałoby dokonać zmiany w we wszystkich dokumentach, we wszystkich miejscach, w których on występuje. Dzięki wprowadzeniu stylów CSS, wszystkie polecenia dotyczące formatowania można umieścić w jednym pliku i powiązać je z konkretnymi elementami, zdefiniowanymi za pomocą czystego (X)HTML. Taka koncepcja sprawia, że modyfikacja wyglądu stron może przebiegać dużo szybciej.

Uwaga: HTML i CSS nie są językami programowania, mimo, że współpracujący ze sobą HTML5 i CSS3 są kompletne w sensie Turinga. Kompletność oznacza możliwość zaprogramowania dowolnego algorytmu, niezależnie od stopnia skomplikowania jego realizacji oraz wydajności obliczeniowej. Analogicznie, kompletny zakład krawiecki, jest w stanie uszyć dowolny rodzaj ubrania, niezależnie ile czasu zajmie skompletowanie materiałów projekt itp, oraz ilu krawców będzie do tego potrzebnych.

3.3. SASS

Sass (Syntactically Awesome Style Sheets) to preprocesor CSS, który pozwala na znacznie szybszą i wydajniejszą pracę z arkuszami stylów. Pozwala na wprowadzenie zmiennych do arkuszy CSS, co przyspiesza pracę i ułatwia automatyzację zarówno procesu tworzenia jak

i obsługi strony wraz z jej stylami. Wynikiem przetwarzania kodu SASS są pliki CSS. W przypadku wpisania kodu CSS do SASS, otrzymamy taki sam kod.

3.4. JavaScript ES6 [8]

```
function helloWorld(){
        console.log("Hello World");
};
```

JavaScript to skryptowy język programowania, stworzony przez firmę NetScape w 1995 roku. Początkowo przeznaczony był tylko do tworzenia skryptów na stronach internetowych, zastępując powolna technologie Flash. Obecnie, dzięki Node.js, czyli specjalnemu środowisku, zawierającego V8, (silnik i interpreter tego języka, pochodzące z przeglądarki Google Chrome), można go uruchomić niemal w każdym zastosowaniu, począwszy od serwerów internetowych, przez aplikacje GUI, po systemy wbudowane.

JavaScriptu nie należy mylić z Javą, która jest zupełnie inną technologią. Nazwa JS-a została nadana z przyczyn marketingowych, gdyż lata 90, były okresem ogromnego wzrostu popularności Javy. Wpłynęła ona na JS tak mocno, że nawet upodobniono do niej jego składnie. Przykład: alokacja pamięci i stworzenie obiektu pies na podstawie konstruktora zwierze wygląda identycznie:

```
var pies = New Zwierze();
```

ES6 to skrót od ECMAscript 6, standardu normalizującego składnię kodu JS-a. Nie jest on wykorzystywany tylko dla JavaScriptu, ale też TypeScriptu, lub JScriptu (technologia służąca do automatyzacji pracy w programach firmy ANSYS)

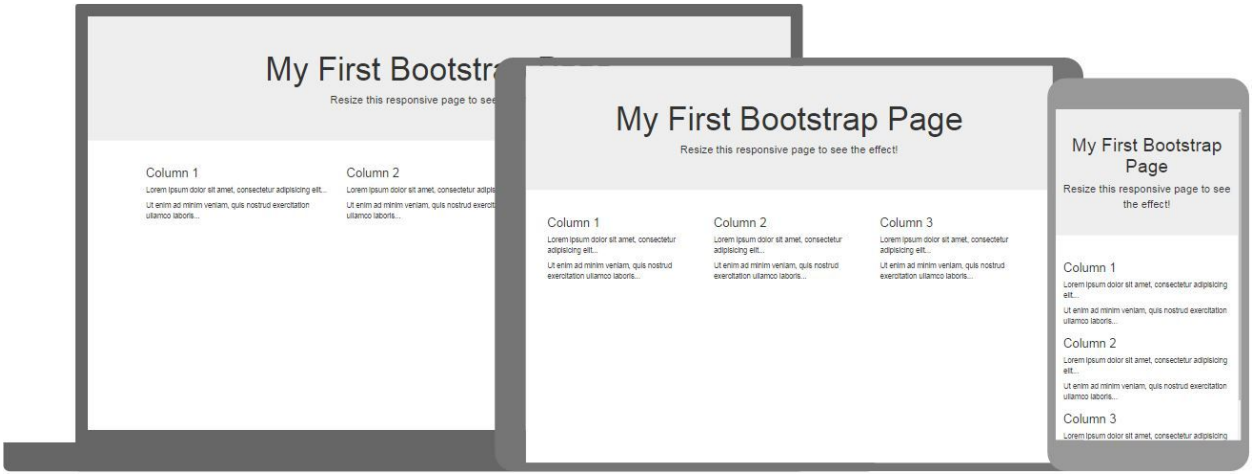
Charakterystyczne cechy JS-a

1) Język wieloparadygmatowy, głównie obiektowy, aczkolwiek programowanie funkcyjne staje się coraz bardziej popularne, wykorzystywane jest między innymi we frameworku React.js
2) Dynamiczne, słabe typowanie - zmienne otrzymują swój typ danych w momencie działania programu, typ zmiennej może zostać w dowolnym momencie zmieniony.
3) Łatwość rozbudowy, niezliczona liczba frameworków i modułów
4) Synchroniczność, instrukcje wykonywane zawsze jedna po drugiej
5) Ograniczony dostęp do systemu plików i ogólnie środowiska poza przeglądarką

Inną nazwą “czystego” JavaScriptu, bez frameworków i rozszerzeń, jest używana dalej **vanilla.js**

3.5. Bootstrap [11]

Bootstrap pozwala na tworzenie responsywnych stron internetowych bez konieczności jawnego wykorzystania JavaScriptu. Zapewnia również kompatybilność wsteczną ze starszymi przeglądarkami. Framework ten, jest przeznaczony do współpracy z HTML. Działa dzięki zestawowi arkuszy styli, LESS (inny preprocessor CSS-a) oraz skryptom w JQuery i JS. Aby skorzystać z wbudowanych w niego funkcji, musimy dodać odpowiednią klasę do znacznika HTML.



Rys. 3.1. Ilustracja działania frameworku Bootstrap (https://www.w3schools.com/bootstrap4/)

Responsywność, ang. RWD - *Responsive Web Design*, jest to pewne założenie w tworzeniu stron, w której zawartość sama dostosowuje się do kształtu i wymiarów ekranu. Schemat działania przedstawiony jest na rysunku 3.1. Obecnie tworzenie tego typu stron jest bardziej premiowane przez algorytmy wyszukujące, niż stworzenie osobnej domeny dla urządzeń mobilnych.

3.6. JQuery [8]

```
$(document).ready(function(){
        $("p").click(function(){
                $(this).hide();
        });
});
```

JQuery to framework rozszerzający funkcjonalność czystego *JS*-a. Jego jądrem jest specjalny obiekt zwracany przez `$()`, a jego metody są funkcjami udostępnianymi przez bibliotekę. Główne jego zalety to:

1) Uproszczona składnia

Przykład: zapisanie zawartości znacznika blokowego `<div>` o identyfikatorze "container" do zmiennej typu ciągu znakowego:

```
//vanilla.js:
var x = document.getElementById("container").innerHTML
//JQuery:
var x = $("#container").html()
```

Techniczna różnica między tymi dwoma zapisami, polega na tym, że w czystym JS-sie "chwytamy" element o identyfikatorze "container" i zapisujemy do zmiennej jego właściwość `.innerHTML`. W JQuery podajemy go jako parametr, do globalnego obiektu `JQuery`, a następnie korzystamy z jego metody .html(), która zwraca string z zawartością.

2) Kompatybilność wsteczna ze starszymi przeglądarkami internetowymi

Obiekt `$()` używa różnych metod, automatycznie dostosowanych do przeglądarki użytkownika.

3) Ułatwione tworzenie dynamicznych stylów i animacji

Dzięki metodzie `.animate()` jesteśmy w stanie animować dowolną właściwość CSS, którą można opisać pewną początkową i końcową liczbą(np.: szerokość, margines itp.).

3.7. Math.js [10]

Kluczowy framework w tym projekcie, udostępnia zaawansowane matematyczne funkcje niedostępne w vanilla.js. Najważniejszą jest analizator składniowy, który szerzej scharakteryzowano w rozdziałach poświęconym realizacji obliczeń.

3.8. Plotly.js [12]

Biblioteka pozwalająca na dynamiczne generowanie wykresów. Danymi wejściowymi jest obiekt, którego właściwości "x" i "y" (oraz "z" w przypadku wykresów trójwymiarowych) zawierają informacje argumentach i wartościach funkcji. W pozostałych propercjach zapisane są informacje o formatowaniu wykresów.

Wygenerowane wykresy są responsywne, Plot.ly udostępnia wiele sposobów na ich analizę, edycje, a nawet zapis do plików graficznych .png.

3.9. JSON [8]

```
{
     "property1":"value1"
     "nestedObject":
      {
           "property2":"value2"
      }
}
```

JavaScript Object Notation - nowoczesny sposób zapisu i przechowywania danych. Znajduje zastosowanie w wielu innych językach, nie tylko w JS. JSON jest znacząco lżejszy niż starszy XML, oraz nie wymaga specjalnych funkcji przepisujących obiekt do danej postaci, wystarczy zastosować rzutowanie (konwersję) jego typu na `string` (ciąg znakowy).

4. Założenia programistyczne

1) **Szybkość obliczeń**

Program musi być napisany w taki, sposób, aby jak najlepiej wykorzystać dostępną moc obliczeniową. JavaScript, ze względu na dynamiczne typowanie oraz fakt, że jest to język interpretowany, nie jest szybką technologią.

2) **Prostota i skalowalność**

Skrypt ma wykorzystywać możliwie jak najmniejszą ilość funkcji (w sensie programistycznym), przy zachowaniu pryncypiów inżynierii oprogramowania oraz stanowić łatwą bazę do rozbudowy i aktualizacji.

3) **Kompatybilność wsteczna oraz kompatybilność w przód**
4) **Czytelność i porządek kodu, liczne komentarze**
5) **Zabezpieczenie przed błędami**
6) **Całość skryptu ma zostać wykonana po stronie klienta**

5. Shaft.js a Wał ‘99

1) **Szybkość**

Najdłuższy czas oczekiwania na wynik w programie shaft.js wynosił 2.21s. Dla Wału ‘99 ponad 22,9s. Różnica wynika z faktu, że wyniki w starszym programie są przedstawiane w środowisku

programu Microsoft Excel. Po refaktoryzacji i odejściu od poprzedniej architektury funkcyjnej, przeciętny czas obliczeń zmniejszył się do wartości poniżej 0.5s.

2) **Kompatybilność**

Wał ‘99 był projektowany, aby uruchamiać się na systemach o architekturze x86-32, przez co w przyszłości mogą pojawić się problemy z uruchomieniem go na nowszych komputerach. Co więcej, do uruchomienia wymagana jest w pełni seryjna (niezmodyfikowana) wersja systemu Windows. Nie można go również uruchomić na innych urządzeniach niż komputery klasy PC. Shaft.js nie korzysta ze środowiska uruchomieniowego systemu operacyjnego, lecz z przeglądarki internetowej, która pośredniczy między kodem a OS-em.

3) **Zasoby potrzebne do uruchomienia**

Do uruchomienia Wału ‘99 potrzebujemy komputera z zainstalowanym pakietem bibliotek Microsoft Visual Redistributable. Do tego do wyświetlenia wyników wymagany jest płatny program Microsoft Excel. Shaft.js wymaga jedynie przeglądarki internetowej, preinstalowanej z niemal każdym systemem operacyjnym.

6. Wstęp do opisu budowy aplikacji

Budowę każdej aplikacji komputerowej można podzielić na dwie części. Tego typu podział, w dokumentacji komputerowych programów, jest dość naturalny i powszechny, niezależnie od platformy i technologii.

Pierwsza część, to część “Front-Endowa”, czyli ta z którą ma styczność użytkownik. Składają się na nią: kod HTML, definiujący strukturę danych, w której zapisane są poszczególne składniki interface’u, takie jak pola tekstowe, przyciski, pola grafiki wektorowej, akapity tekstu itd. oraz kod CSS, który definiuje wygląd wcześniej wspomnianych elementów. Jako, że oba powyższe języki są statyczne, towarzyszy im również język skryptowy, który definiuje ich reakcje na pojawiające się zdarzenia, takie jak: naciśnięcie przycisku, zmiana rozmiaru strony itd. Tę rolę pełni JavaScript.

Front-End to nie wyłącznie interfejs użytkownika. Należą do niego również funkcje przetwarzające i zapisujące wprowadzane dane oraz funkcje wyświetlające wyniki obliczeń oraz ich graficzne interpretacje. Do tej części należy się również kod odpowiedzialny za responsywność strony (Bootstrap i kilka autorskich funkcji).

Drugą częścią jest “Back-end”. Zawiera on funkcje wczytujące i przetwarzające dane, a także struktury przechowujące informacje. Tę część stanowi niemal wyłącznie JavaScript,

wspomagany bibliotekami rozszerzającymi jego funkcjonalność (np.:Math.js, który zapewnia zaawansowane funkcje matematyczne).

Opis Front-endu i Back-endu “od ogółu do szczegółu” przedstawiają kolejne rozdziały pracy, od interfejsu użytkownika i komunikacji z silnikiem aplikacji, aż do algorytmów obliczeń w poszczególnych etapach projektu.

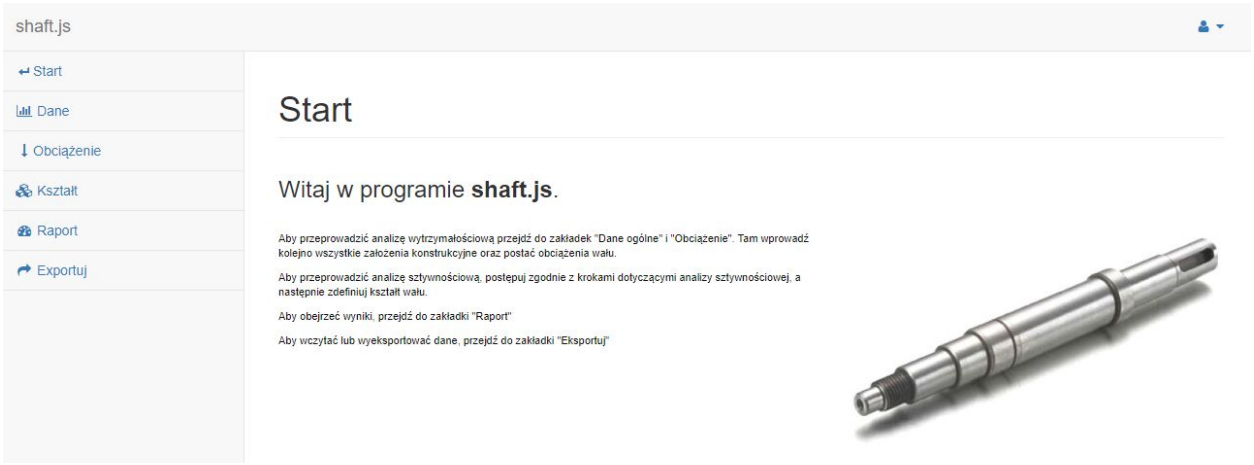
7. Front-end i składniki interface’u

7.1. Interface programu

Program składa się z 6 ekranów, ułożonych w kolejności zgodnej z procedurą projektowania wału.

7.1.1. Start

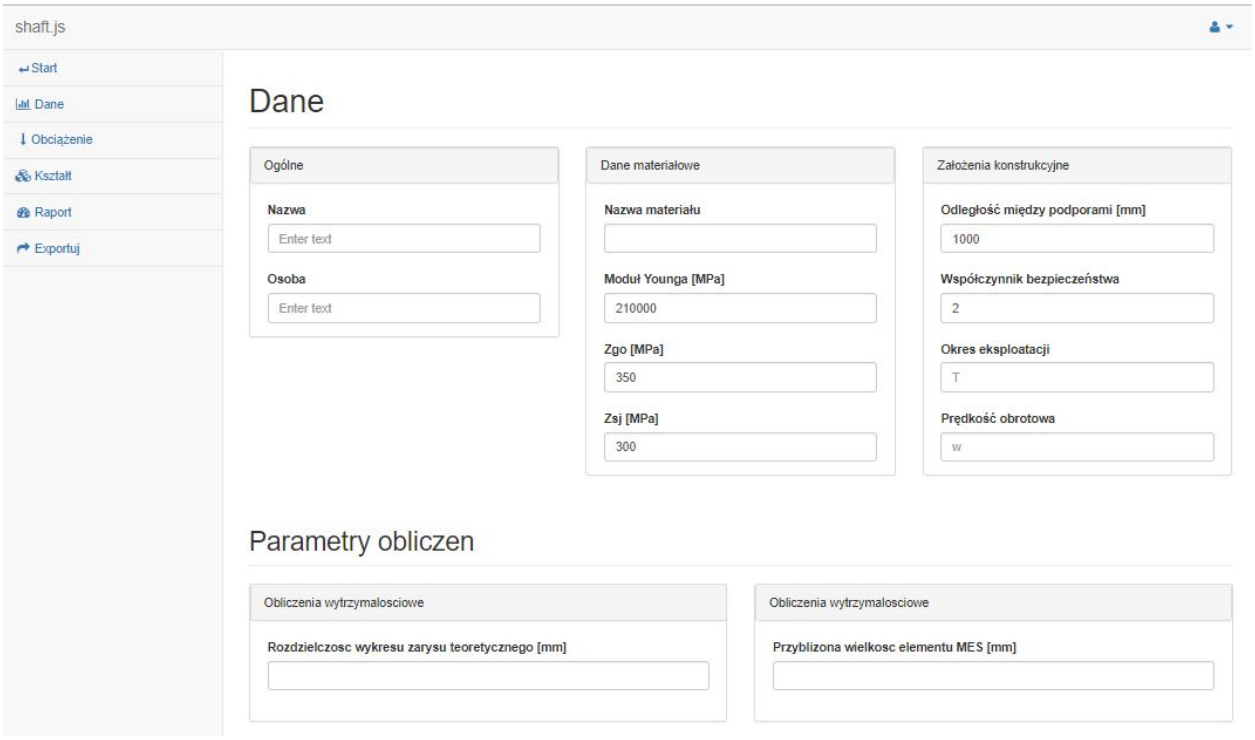
Zakładka ,,Start’’ jest zakładką powitalną (ang. *Splash screen*). W tym miejscu znajdziemy wskazówki gdzie znaleźć interesujące nas funkcjonalności, historię kolejnych wydań programu oraz listę zmian w nich wprowadzonych.



Rys. 7.1. Ekran powitalny

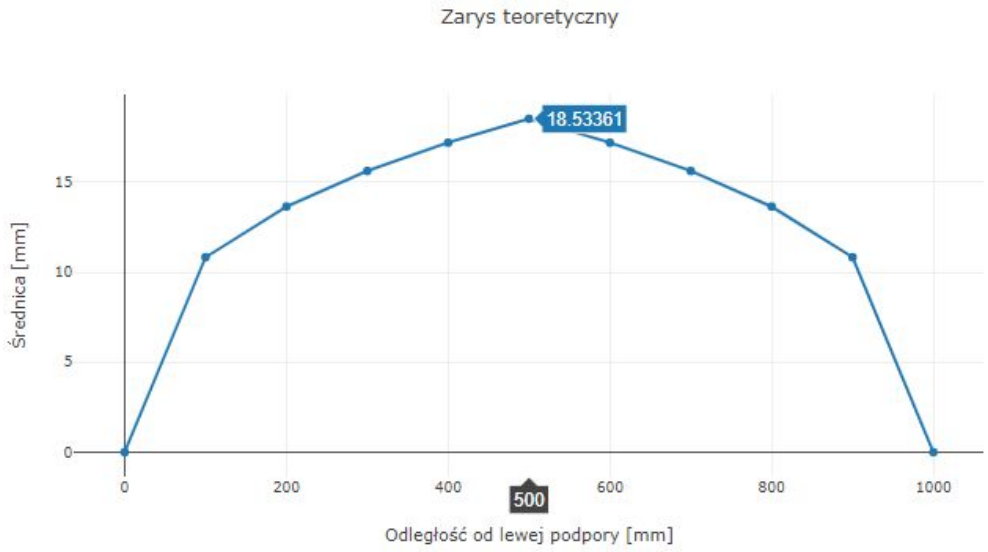
7.1.2. Dane

W zakładce ,,Dane’’, wprowadzane są założenia podstawowe założenia projektowe, na których oparta jest analiza. Ponadto w zakładce tej definiowane są charakterystyczne parametry (wpływające znacząco na złożoność obliczeniową) takie jak:

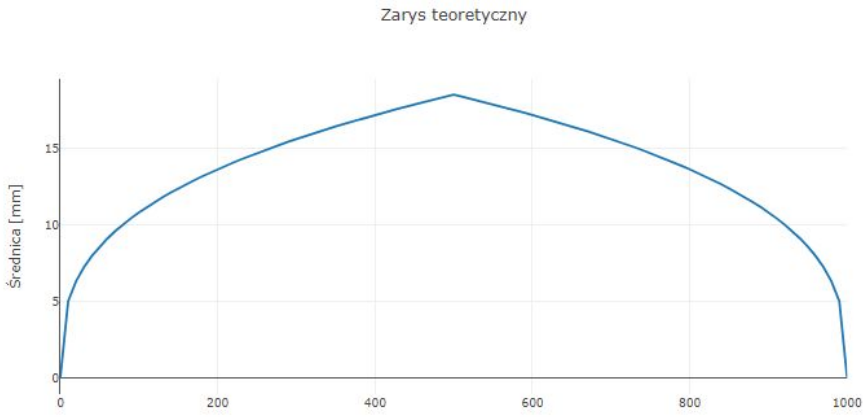


Rys. 7.2. Zakładka "Dane" – kolumny danych i parametrów obliczeń

1) Rozdzielczość wykresu zarysu teoretycznego - ustalany jest krok obliczeniowy, z jakim wyznaczany jest zarys teoretyczny. Im mniejsza jest jego wartość, tym otrzymany wykres będzie dokładniejszy (rys. 7.3 i 7.4). Domyślna długość to 10 mm. Zapotrzebowanie na moc obliczeniową rośnie odwrotnie proporcjonalnie do wielkości kroku.



Rys. 7.3. Wykres zarysu teoretycznego przy kroku obliczeniowym równym 1/10 długości wału



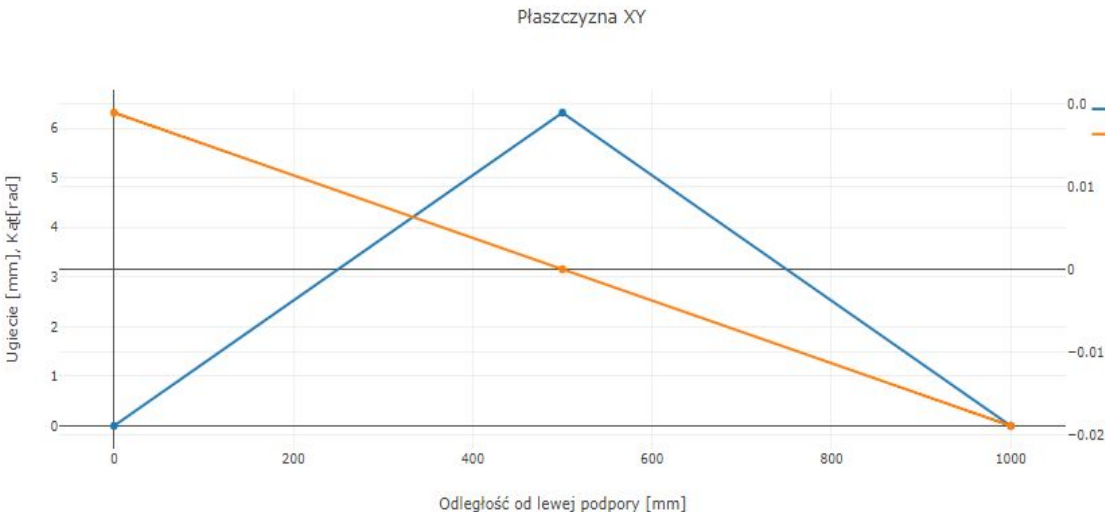
Rys. 7.4. Wykres zarysu teoretycznego przy kroku obliczeniowym równym 1/100 długości wału

2) Przybliżona wielkość elementu MES - parametr obliczeń sztywnościowych. Domyślnie algorytm dzieli wał na elementy skończone zgodnie z postulatem:

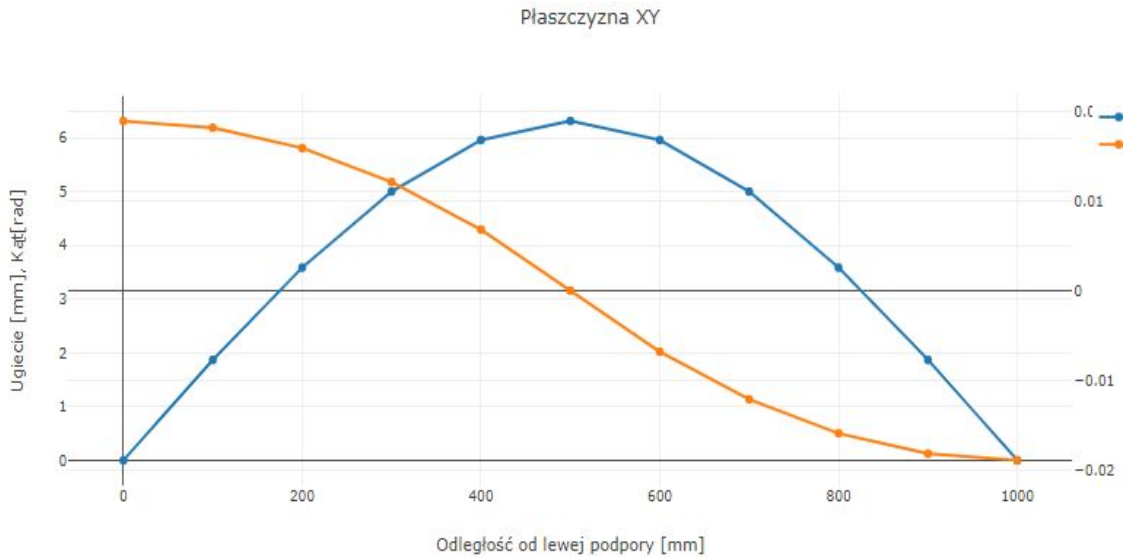
- Stopień nieobciążony - jeden element
- Stopień obciążony - tyle elementów w stopniu, ile punktów przyłożenia obciążenia + 1

W razie potrzeby wyższej dokładności lub lepszej wizualizacji wyników, użytkownik może zdefiniować ręcznie długość elementu.

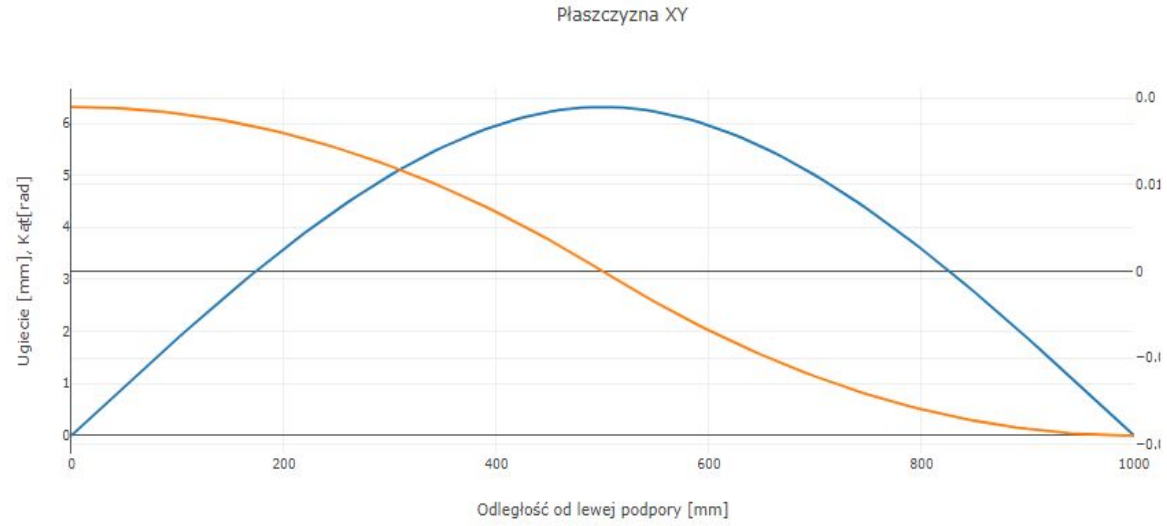
Zapotrzebowanie na moc obliczeniową rośnie wraz z połową kwadratu liczby stworzonych elementów skończonych [7]. Dla obliczeń belki o jednorodnym przekroju, obciążonej siłą skupioną w taki sposób, że jej odległość od obu podpór jest równa, wykresy ugięć, różnych rozdzielczościach, zostały przedstawione na rysunkach 7.5, 7.6, i 7.7. Kolorem niebieskim oznaczono wykres ugięcia, a kolorem pomarańczowym oznaczono wykres kąta ugięcia wału.



Rys. 7.5. Wykres ugięć bez definicji wielkości elementu, standardowe obliczenia wału



Rys. 7.6. Wykres ugięć przy podziale na elementy skończone o długości 1/10 długości wału (WŁĄCZONE automatyczne zaznaczanie pozycji węzłów)



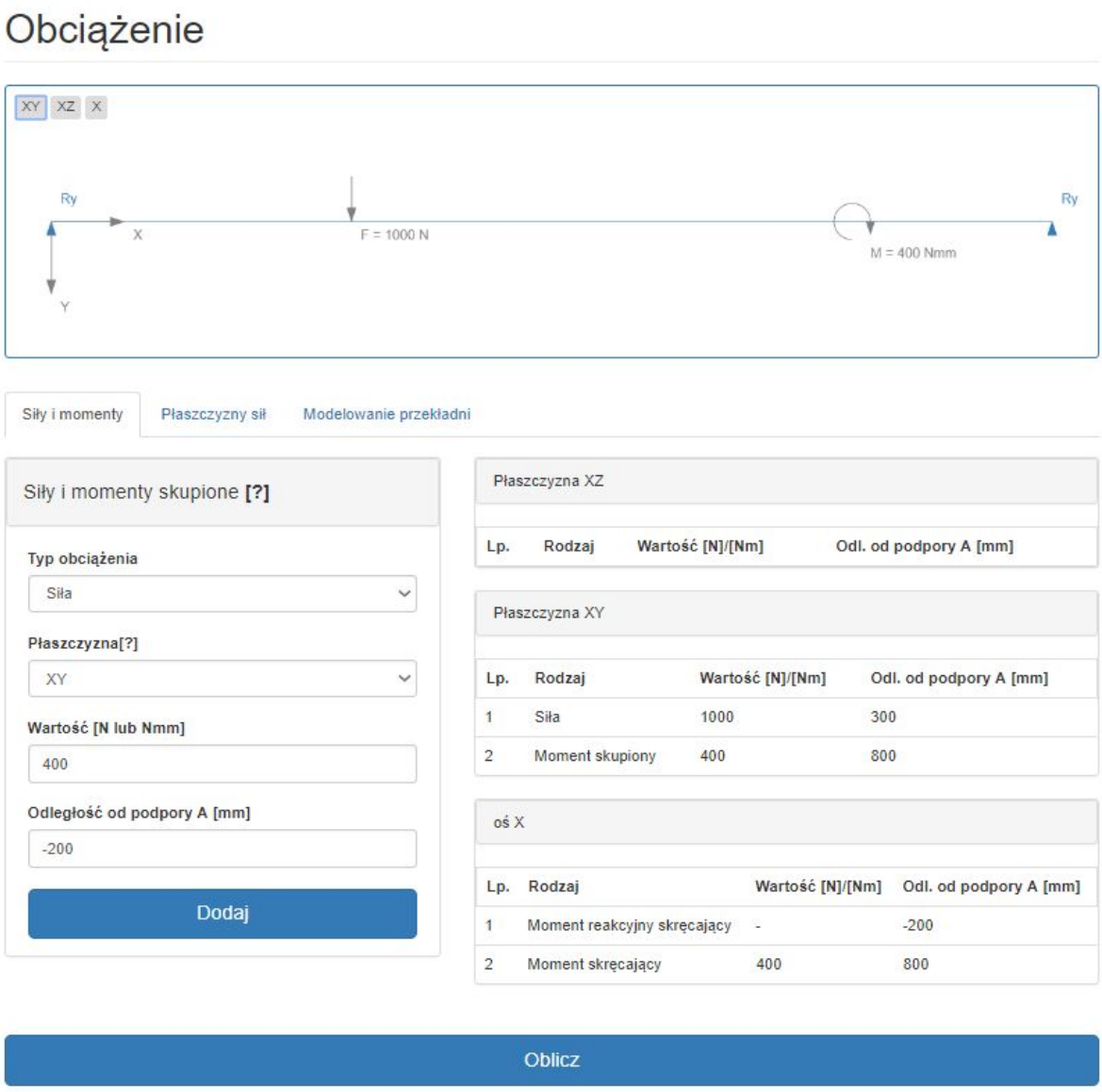
Rys. 7.7. Wykres ugięć przy podziale na elementy skończone o długości 1/100 długości wału (WYŁĄCZONE automatyczne zaznaczanie pozycji węzłów – ze względu na duże ich zagęszczenie)

Sposób realizacji powyższych ustawień w kodzie źródłowym i ich wpływ na algorytmy przetwarzania danych, jest przedstawiony w następnych rozdziałach, dotyczących obliczeniowej części programu.

7.1.3. Obciążenie

W zakładce ,,Obciążenie'', definiowany jest stan obciążenia, któremu poddawany jest badany wał lub oś. Wprowadzone dane są wykorzystywane w obliczeniach sztywnościowych i wytrzymałościowych.

W programie można zadać obciążenie na dwa sposoby. Możliwe jest ręczne wprowadzenie pojedynczych sił lub momentów skupionych, lub na stworzenie specjalnych płaszczyzn obciążenia, które są geometrycznymi modelami dowolnego typu przekładni.



Rys. 7.8. Zakładka "Obciążenie" – konfiguracja warunków obciążenia

Definicja poprzez wprowadzenie sił i momentów skupionych

Pierwszy sposób definicji obciążenia polega na wprowadzeniu sił skupionych. Do dyspozycji użytkownika oddane są następujące typy obciążenia:

- Siła
- Moment skupiony
- Siła osiowa
- Moment skręcający
- Moment skręcający reakcyjny

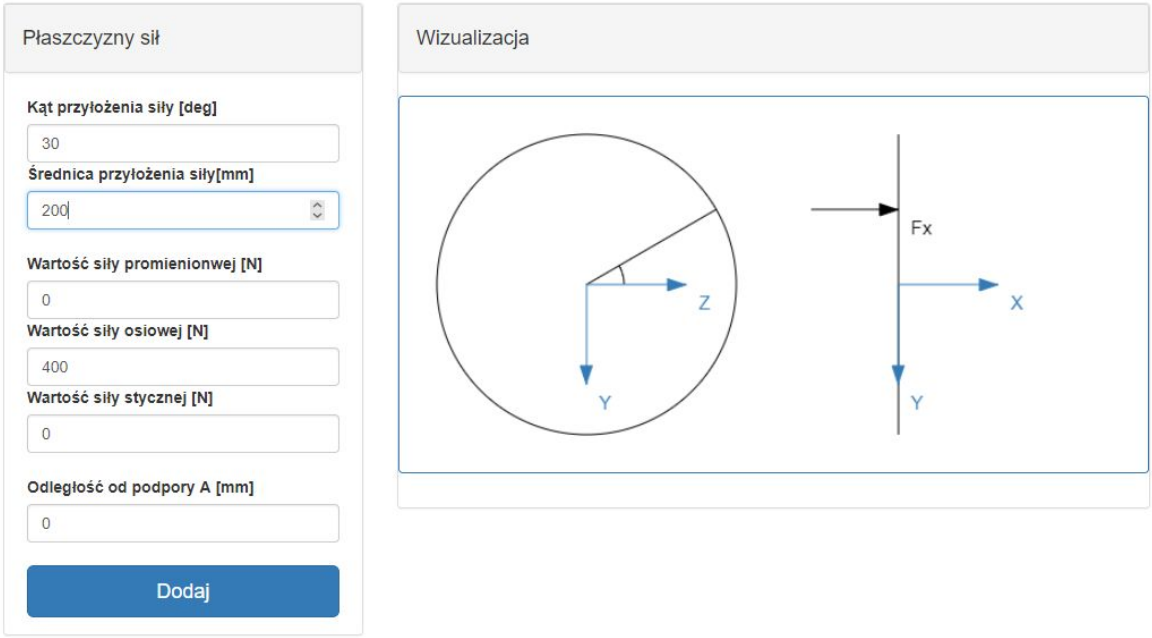
Oprócz tego, program wykorzystuje jeszcze typ obciążenia ”reakcja”,który nie jest on dostępny z poziomu użytkownika. Obciążenia typu “Siła” i “Moment skupiony ” można zadać w płaszczyźnie XY i prostopadłej do niej XZ. Odpowiadają one płaszczyznom horyzontalnym i wertykalnym. Pozostałe typy obciążeń zadawane są w osi X.

Moment skręcający reakcyjny to specjalny typ obciążenia. Użytkownik może zadać jedynie jego położenie. Jego wartość dobierana jest w trakcie przeprowadzania obliczeń i wynosi ona tyle, że spełniony jest następujący warunek: Suma momentów skręcających i momentu skręcającego reakcyjnego wynosi 0.

Definicja poprzez wprowadzenie płaszczyzn obciążenia

Obciążenie jest definiowane przez dodanie do wału modelu przekładni. Na okręgu odpowiadającym okręgowi tocznemu przekładni wyznaczany jest punkt. Do niego przykładane są 3 siły:

- Siła promieniowa
- Siła osiowa
- Siła styczna



Rys. 7.9. Zakładka "Obciążenie" – podzakładka konfiguracji obciążenia w płaszczyznach sił

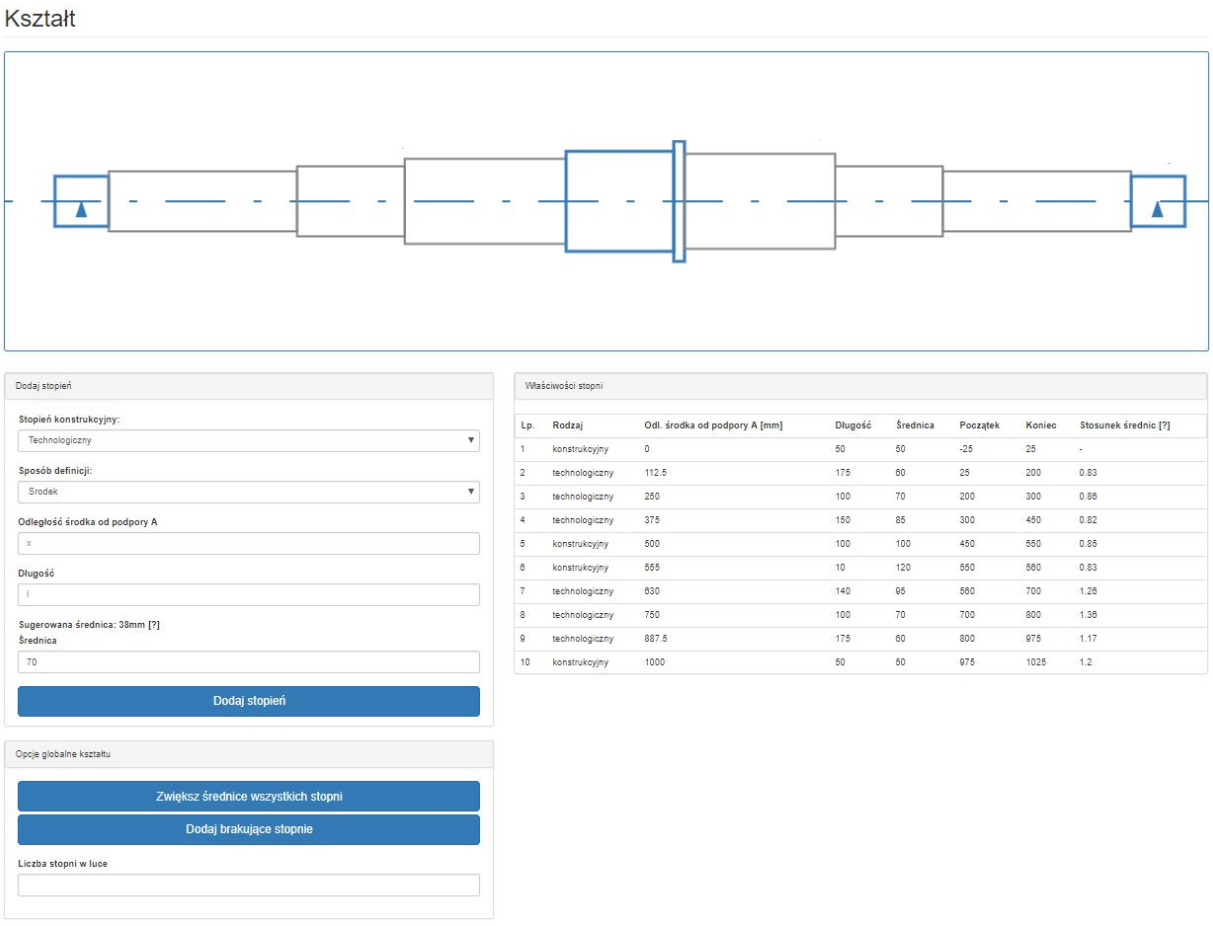
Po zdefiniowaniu pojedynczej płaszczyzny obciążenia, program przelicza i dodaje do bazy danych wejściowych siły skupione odpowiadające tej płaszczyźnie.

Powyższa metoda została wprowadzona na wzór domyślnego sposobu wprowadzania obciążenia programu Wał 99. Sprawdza się ona dobrze przy modelowaniu przekładni, w innych przypadkach prostszym sposobem jest użycie metody sił skupionych. Przykładowo, modelowanie sprzęgła wymaga wprowadzenia dwóch płaszczyzn w tej samej współrzędnej, dobranie odpowiednich kątów, sił promieniowych itd. W metodzie obciążeń skupionych wystarczy jedynie wprowadzić w miejscu sprzęgła pojedyncze obciążenie typu “Moment skręcający” lub “Moment skręcający reakcyjny”

Niezależnie od sposobu definiowania obciążenia, początek układu współrzędnych umiejscowiony jest zawsze w lewej podporze. Wymiar gabarytowy jest automatycznie wyznaczany na podstawie odległości między skrajnymi obciążeniami lub podporami.

7.1.4. Kształt

W zakładce ,,Kształt’’, wprowadzane są parametry geometryczne wału.



Rys. 7.10. Zakładka "Kształt" – modelowanie (parametryzacja) geometrii wału

Oprócz definiowania stopni, użytkownik ma do dyspozycji szereg funkcji, pozwalających na proste edytowanie i zarządzanie kształtem części.

Definiowanie kształtu wału

Parametry geometryczne stopni wału definiowane są w oknie po lewej. Użytkownik może wybrać, czy definiowany stopień będzie pełnił funkcję technologiczną lub konstrukcyjną. Nie ma to wpływu na późniejsze przetwarzanie, pełni wyłącznie rolę poglądową. Położenie stopnia względem podpór oraz jego długość można zdefiniować na dwa sposoby:

1) Definicja poprzez wprowadzenie współrzędnej środka i długości
2) Definicja poprzez wprowadzenie współrzędnych końca i początku

Na podstawie przeprowadzonych wcześniej obliczeń wytrzymałościowych, program wyznacza minimalną średnicę wału. Wyznaczana jest ona na podstawie położenia końca i początku aktualnie definiowanego stopnia. Użycie mniejszej średnicy, spowoduje, że wał, przy za stanie obciążenia będzie pracował w zakresie plastycznym.

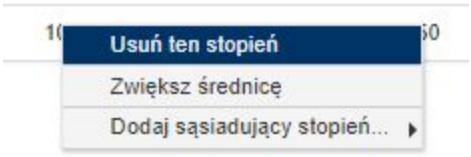
Tabela

W tabeli znajdują się wszystkie informacje o kształcie wału. Istotną informacje zawiera kolumna o tytule ,,iloraz średnic''. Jest to parametr mówiący o tym, ile razy większa jest średnica kolejnego stopnia. Na jego podstawie użytkownik może ocenić, czy uzyskana liczba stopni jest odpowiednia, lub oszacować, czy można wykorzystać powierzchnię oporową kolejnego stopnia.

Globalne i lokalne funkcje modelowania kształtu

Do użytku zostało oddanych kilka funkcjonalności automatyzujących kształtowanie. Dotyczą one pojedynczych stopni (funkcje lokalne) lub całego wału (funkcje globalne).

Funkcje lokalne można odnaleźć klikając lewym przyciskiem myszy na wiersz w tabeli agregującej stopnie. Zostanie wówczas wyświetlone menu kontekstowe.



Rys. 7.11. Menu kontekstowe edycji stopnia

Korzystając z powyższego menu możemy:

- Usunąć stopień
- Edytować jego średnice

- Dodać stopień sąsiadujący - program przełącza się w tryb definiowania stopni “Początek i koniec”, po czym wypełnia pole współrzędnej końca lub początku odpowiednią współrzędną stopnia-rodzica.

Sterowanie funkcjami globalnymi znajduje się pod spodem panelu definiowania stopni. Program udostępnia dwie funkcje:

- Edycja średnicy wszystkich stopni - program zwiększa lub zmniejsza średnice o zadaną przez użytkownika wartośc
- Wypełnienie luk między stopniami

Aplikacja wykrywa, czy wał jest ciągły. W przypadku wykrycia nieciągłości w jej miejsce definiowany jest jeden (domyślnie) lub więcej stopni. Ich średnice dobierane są w taki sposób, że ich wartości stanowią kolejne wyrazy skończonego ciągu arytmetycznego. Jego pierwszy i ostatni wyraz jest równy średnicom stopni, pomiędzy którymi znajduje się nieciągłość. Na podstawie tych średnic oraz zadanej liczbie stopni, obliczana jest różnica tego ciągu i jego pozostałe wyrazy.

7.1.5. Raport

Zakładka agregująca wyniki analizy. Znajdziemy tam wszystkie wcześniej wprowadzone dane ogólne, graficzną interpretacje zadanego obciążenia, i wszystkie interesujące wykresy.

Raport składa się z dwóch niezależnych części wyświetlających wyniki obliczeń wytrzymałościowych i sztywnościowych. Jego zawartość została tak zaprojektowana, aby wydrukowana stanowiła kompletny zestaw wynikowych danych opisujący proces kształtowania i projektowania wału.

7.1.6. Eksportuj

W tym miejscu znajdują się wszystkie funkcje potrzebne do zarządzania wynikami obliczeń.

7.2. Grafika Wektorowa

Shaft.js wykorzystuje grafikę wektorową w kilku funkcjonalnościach. W dynamicznie generowanych wykresach, oraz przy wizualizowaniu zadawanego przez użytkownika obciążenia i kształtu.

7.2.1. Wykresy

Funkcja pośrednicząca między częścią obliczeniową a biblioteką Plot.ly ma za zadanie wprowadzać odpowiednie dane wejściowe, niezależnie od typu tworzonego wykresu. Wykorzystywany jest tu fakt, że JS obsługuje specyficzne przeładowywanie metod - nie wymaga obecności wszystkich parametrów funkcji w momencie jej inwokacji. Dzięki temu, ta sama funkcja, w zależności od zadanych parametrów, może wyświetlić wiele postaci wykresów.

Oprócz tworzenia wykresów, Plot.ly udostępnia następujące funkcje

- Zapis do pliku .png
- Edycja
- Skalowanie i przybliżanie

7.2.3. Canvas i mapa bitowa

Zasada działania funkcji generującej grafikę na mapach bitowych wynika bezpośrednio z metody zapisu danych obu klas obiektów. Wywołanie funkcji zaczyna się od obliczenia poszczególnych współczynników skalujących. Wyznaczane są one, na podstawie rozmiaru rysowanego kształtu oraz rozmiarów elementu `<canvas>`. Następnie za pomocą metody `forEach()` i instrukcji warunkowej dla każdego elementu tablicy rysowany jest odpowiedni kształt i kolor, odpowiadający typowi obciążenia lub stopnia. W momencie "wykrycia" przez program lewej podpory, rysowany jest również układ współrzędnych.

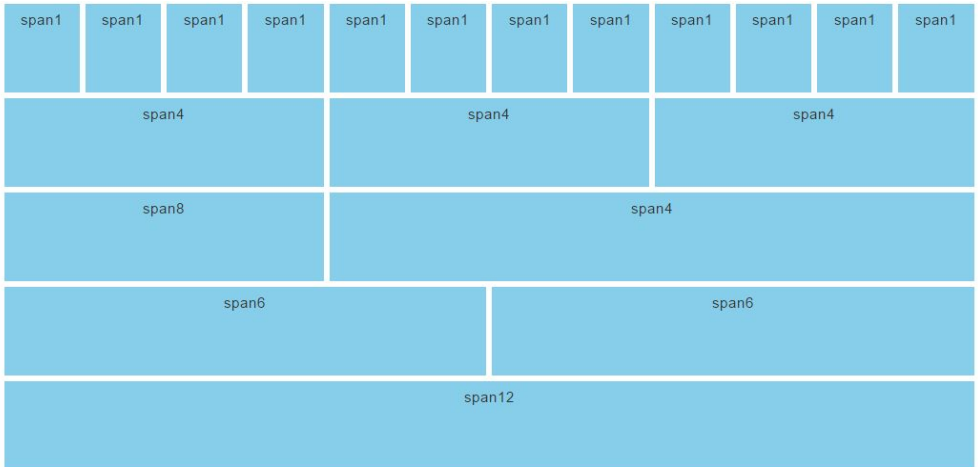
Warto zaznaczyć również w jaki sposób rysowane są podpory wału. Na początku wywołania funkcji rysującej tworzona jest kopia tablicy kształtu, do której dodawane są specjalne dwa obiekty klasy Step, których właściwość “typ” to “podpora”, a współrzędne odpowiadają współrzędnym podpór. Nie posiadają one żadnych innych właściwości W trakcie iteracji przez tablicę, w momencie wykrycia tego typu obiektu, program rysuje symbol podpory. Dzięki sortowaniu po współrzędnej środka, zostają one narysowane w odpowiednim miejscu, odpowiadającym położeniu podpory względem wału.

8. Programistyczna struktura interface’u

Struktura programu od strony kodu HTML jest systemem zakładek dostarczanym przez Bootstrap. Sterowanie odbywa się przez menu z lewej strony ekranu, na urządzeniach o wysokich rozdzielczościach (Komputery, laptopy) lub rozwijanego z góry ekranu (Telefony komórkowe, smartfony, tablety). W momencie kliknięcia na przycisk tego menu, poszczególne elementy blokowe (kontenery podstron), mają zmieniane odpowiednie właściwości CSS. Odpowiadają za ich wyświetlanie, pozycjonowanie i widoczność. Dzięki temu klikając na

poszczególne przyciski z lewej strony, mamy wrażenie poruszania się po menu, prowadzącego do odpowiednich funkcji programu.

Dostosowanie się wymiarami i rozmieszczeniem elementów zapewnia wyżej wymieniony framework Bootstrap. Dzieli on przestrzeń okna lub dowolnego elementu blokowego na 12 kolumn o takiej samej szerokości (rys. 8.1). Do tego, przyjmowane jest 5 możliwych szerokości ekranu.



Rys. 8.1. Siatka Bootstrapa (https://www.formget.com/bootstrap-grid/)

Dla każdego znacznika ** ** automatycznie dobierane jest ile zajmie on kolumn w zależności od szerokości ekranu.

Dla elementów blokowych, tabel, menu itp., tę funkcjonalność zapewnia Bootstrap. Nie wpływa on jednak na właściwości CSS, elementów, oraz rozdzielczości grafik w nich zawartych. Reguluje on wyłącznie szerokość i rozmieszczenie elementów. Aby zachować poprawne wyświetlanie generowanych przez program grafik wektorowych, należy zadbać o zachowanie odpowiednich proporcji ich szerokości i wysokości.

Aby zapewnić powyższe funkcjonalności dla znacznika **<canvas>** (generator i kontener grafik wektorowych) wymagana jest autorska funkcja. Uruchamiana jest w momencie ładowania strony oraz przy zmianie wymiarów okna (wywołane zdarzenie "`onResize`"). Wczytywana jest wówczas szerokość jednego z nagłówków, w zależności od tego, który jest aktualnie widoczny. Następnie zostaje ona przypisana do widocznych w tym momencie canvasów.

Za responsywność wykresów odpowiada Plot.ly inna autorska funkcja. Jest to wymagane, gdyż w momencie dodawania wykresu do struktury HTML, kod Plot.ly pobiera szerokość elementu - rodzica. Ponieważ rodzic jest niewidoczny (znajduje się w ukrytej w tym momencie zakładce ,,Raport''), zostaje przypisana niepoprawna szerokość równa 0 pikseli. Funkcja pobiera wartość chwilowej szerokości ekranu, po czym przypisuje odpowiednią właściwość CSS ukrytemu wykresowi. Jej algorytm został opisany w rozdziale 13.

9. Weryfikacja wprowadzanych danych

Weryfikacja wprowadzanych danych jest dwuetapowa:

1) Pierwszy etap zapewnia dostarczany przez Bootstrap rodzaj pola tekstowego, `input.type="number"`. Różni się on od zwykłego `input.type = "text"`, tym, że pozwala wprowadzić jedynie liczbę. Dzięki temu nie trzeba stosować weryfikacji wyrażenia regularnego (*RegEx* - ang. *regular expression*) .Posiada również dwa przyciski pozwalające regulować wprowadzaną wartość za pomocą myszki. Dzięki temu już z poziomu kodu HTML, sprawdzane jest czy użytkownik wprowadzi wartość niemożliwą do skonwertowania na liczbę zmiennoprzecinkową.

Niestety zastosowanie tego typu pola nie zabezpiecza przed dwoma możliwościami wpisania nieobsługiwanych danych. Okazuje się, że Bootstrap dopuszcza wprowadzanie znaku “e”. Intuicyjnie można by było się spodziewać, że zapisaną wartością będzie stała Eulera. Tak się jednak nie dzieje, zapisany zostaje ciąg znaków: “e”. Drugim wyjątkiem jest możliwość zapisania pustego ciągu znaków.

2) Drugi etap realizowany jest przez autorską funkcję. Jej zadaniem jest uniemożliwienie realizacji obliczeń gdy jeden z dowolnie dobranych warunków nie zostanie spełniony. Składa się ona z dwóch poziomów instrukcji warunkowych. Pierwszy sprawdza, jaki typ obliczeń jest właśnie przeprowadzany. W drugim zawarte są warunki dotyczące obliczeń sztywnościowych, wytrzymałościowych lub obu tych typów. Funkcja została napisana w taki sposób, aby móc wyświetlić komunikaty o wszystkich błędach popełnionych przez użytkownika, a nie, tylko jeden, pierwszy na liście sprawdzanych warunków.

Obecnie sprawdzane są następujące warunki:

I) “Czy użytkownik w polach liczbowych wpisał wyłącznie dane liczbowe?”(oba rodzaje obliczeń)
II) “Czy zadano obciążenie?” (oba rodzaje obliczeń)
III) “Czy zadano kształt?” (obliczenia wytrzymałościowe)
IV) “Czy zadany kształt jest ciągły - nie istnieją luki pomiędzy zadanymi stopniami wału?” (obliczenia wytrzymałościowe)

Uwaga: JavaScript jest językiem dynamicznie typowanym. Oznacza to, że typy danych są dobierane i modyfikowane podczas działania programu i wykonywania instrukcji [9]. Co to oznacza? Program nie zawiesi się lub nie pokaże błędu w momencie kiedy przeprowadzimy operację na niekompatybilnych typach. Pomimo tego kontrola wprowadzonych typów jest konieczna. Przykładowo:

- “2” + 2 - suma liczby i ciągu znakowego. Zwracana wartość to “22”

- 2 - “2” - Zwracana wartość to 0
- “Pies” / 2 - Zwracana wartość to NaN (ang. Not a Number, typ danych wynikający z zastosowania błędnych dla danego operatora wejściowych typów danych)

Jak widać powyżej, za każdym razem zostaje zwrócona błędna wartość. Program za każdym razem inwokuje funkcje obliczeniowe, jednak nie są one wykonywane poprawnie. Funkcje wyświetlające dane nie będą mogły zostać uruchomione prawidłowo. Zdecydowanie nie powoduje to pozytywnego doświadczenia w użytkowaniu programu (ang. User Experience)

10. Teoretyczne podstawy obliczeń

Niniejsza praca dyplomowa została napisana w stylu dokumentacji naukowo-technicznej, stąd też oprócz charakterystyki aplikacji wartością dodaną jest prezentacja wykorzystanych w projekcie zależności oraz praw mechaniki i wytrzymałości materiałów.

10.1. Obliczenia wytrzymałościowe [5]

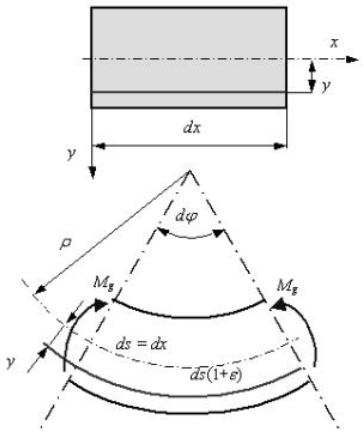
Celem przeprowadzania obliczeń wytrzymałościowych, jest znalezienie minimalnej średnicy wału (danego przekroju), zdolnej przenieść zadane obciążenie. Wały maszynowe są częściami poddanymi złożonym stanom naprężeń.

10.1.1. Płaskie zginanie:

Mechanizm zginania belki(założenia):

- Przekrój płaski pozostaje po odkształceniu płaski oraz prostopadły do osi belki
- Warstwa obojętna to warstwa prostopadła do płaszczyzny działania momentu gnącego
- W przekroju poprzecznym występują wyłącznie naprężenia normalne, w przekroju wzdłużnym naprężeń normalnych i tnących brak.

Rozpatrzmy element belki o długości dx przed i po odkształceniu:



Rys. 10.1. Belka o długości dx, przed i po odkształceniu [5]

Weźmy pod uwagę włókno znajdującę się w odległości y od warstwy obojętnej. Długość włókna przed odkształceniem to dx = ds. Po zadziałaniu obciążenia wydłuża się o ε i wynosi ds (1+ ε).

Z prostych zależności geometrycznych otrzymujemy następujące wyrażenie:

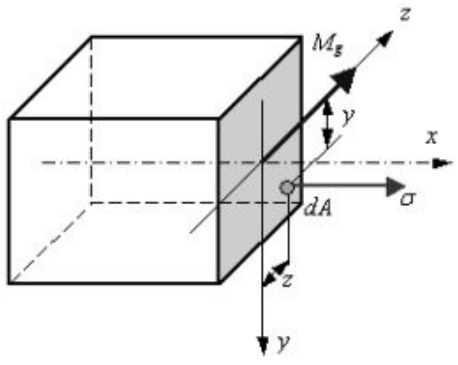
$$\frac{ds(1+\varepsilon)}{-\rho+y}=\frac{ds}{-\rho} \tag{10.1}$$

Po przekształceniu:

$$\varepsilon=-\frac{y}{\rho} \tag{10.2}$$

Powyższy wzór oznacza, że wydłużenie włókien jest proporcjonalne do ich odległości od warstwy obojętnej i posiada ten sam znak, co współrzędna y.

Po wprowadzeniu warunków równowagi zgodnie z rysunkiem 10.2, otrzymujemy:



Rys. 10.2. Ilustracja warunków równowagi podczas zginania samym momentem [5]

$$\sum F_x = 0 \quad \int_A \sigma \, dA = 0$$

$$\sum M_y = 0 \quad \int_A \sigma z \, dA = 0$$

$$\sum M_z = 0 \quad -\int_A \sigma y \, dA + M_g = 0$$

(10.3,10.4,10.5)

Dla materiałów liniowo sprężystych stosuje się prawo Hooke'a. Po uwzględnieniu tego prawa wzór uzyskuje następującą postać:

$$\sigma = E\varepsilon = -\frac{E}{\rho} y$$

(10.6)

Związek ten przedstawia zależność opisującą rozkład naprężeń normalnych w przekroju podczas zginania. Są one wprost proporcjonalne do odległości od osi obojętnej rozważanego przekroju. Po podstawieniu tej zależności do równań równowagi (3,4,5) otrzymujemy:

$$\int_A y \, dA = 0$$

$$\int_A y z \, dA = 0$$

$$\frac{E}{\rho} \int_A y^2 \, dA = -M_g$$

(10.7,10.8,10.9)

Ostatni wzór, zawiera w sobie formułę na moment bezwładności, względem osi prostopadłej do płaszczyzny działania momentu gnącego.

$$\frac{1}{\rho} = -\frac{M_g}{EJ_z}$$

(10.10)

Podstawiając do wzoru () promień krzywizny ze wzoru () wyznaczamy zależność na naprężenia normalne w dowolnym punkcie przekroju w funkcji obciążenia (momentu gnącego)

$$\sigma = \frac{M_g y}{J_z}$$

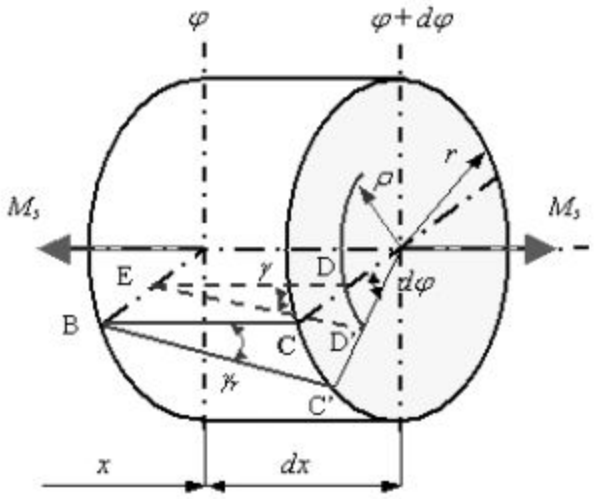
(10.11)

10.1.2. Skręcanie prętów:

Mechanizm skręcania prętów (założenia):

- Hipoteza płaskich przekrojów - przekroje po zadaniu obciążenia pozostają równoległe do siebie oraz prostopadłe do osi obojętnej.
- Kształt przekroju nie ulega zmianie
- Wzajemna odległość przekrojów nie ulega zmianie

Parametry geometryczne zawarte są na następującym rysunku, przedstawiającym fragment pręta o grubości dx, poddanemu obciążeniu skrętnemu.



Rys. 10.3. Fragment belki, o długości dx, poddanej obciążeniu momentem skrętnym [5]

Przekroje na końcu i początku obracają się względem siebie o kąt dϕ. Obrotowi przekrojów odpowiada kąt odkształcenia postaciowego γ, którego wartość na obwodzie wynosi γr . Na podstawie zależności geometrycznych można stworzyć następującą zależność.

$$CC' = dx\,\gamma_r = r\,d\varphi \tag{10.12}$$

Stąd, po przekształceniu:

$$\gamma_r = \frac{d\varphi}{dx} r \tag{10.13}$$

gdzie: dx/dϕ oznacza jednostkowy kąt skręcenia. Dla współśrodkowej płaszczyzny walcowej, w dowolnej odległości (ro), prawdziwa jest również zależność:

$$DD' = dx\,\gamma = \rho\,d\varphi \tag{10.14}$$

Analogicznie:

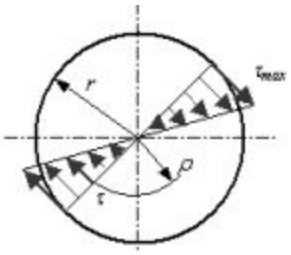
$$\gamma = \frac{d\varphi}{dx}\rho \tag{10.15}$$

Prawo Hooke'a dla ścinania, jest określane w następujący sposób:

$$\gamma = \frac{\tau}{G} \tag{10.16}$$

Gdzie symbol G odpowiada modułowi Kirchhoffa. Po podstawieniu, otrzymamy wzór definiujący naprężenia styczne:

$$\tau = G\rho\frac{d\varphi}{dx} \tag{10.17}$$



Rys. 10.4. Rozkład naprężeń stycznych, w przekroju kołowym, podczas skręcania [5]

Dla dowolnego przekroju, jednostkowy kąt skręcenia jest stały, a zatem wartość naprężeń stycznych jest proporcjonalna do odległości od środka przekroju. Naprężenia styczne w przekroju skręcanego pręta charakteryzuje rozkład liniowy,kierunek wektorów naprężeń jest prostopadły do dowolnego promienia przekroju pręta kołowego.

Równanie równowagi momentów, względem osi przekroju ma postać:

$$\int_A \tau\rho\, dA - M_s = 0 \tag{10.18}$$

Po podstawieniu wzoru na naprężenia styczne:

$$G\frac{d\varphi}{dx}\int_A \rho^2 dA - M_s = 0 \tag{10.19}$$

Otrzymujemy wzór, w którym zawiera się inny wzór, definiujący równanie na biegunowy moment bezwładności.

$$\int_A \rho^2 dA = J_0 \tag{10.20}$$

Z równania równowagi wyznaczamy kąt skręcenia:

$$\frac{d\varphi}{dx} = \frac{M_s}{GJ_0} \tag{10.21}$$

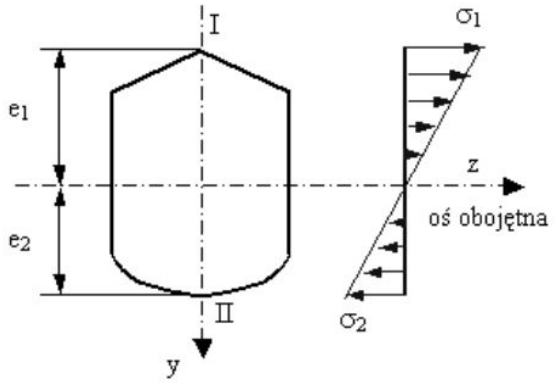
Wzór na naprężenia styczne w funkcji obciążenia i promienia powstaje poprzez podstawienie, do wyrażenia, formuły na naprężenia styczne podstawiamy zależność (10.19) określającą jednostkowy kąt skręcenia, otrzymujemy:

$$\tau = \frac{M_s}{J_0}\rho \tag{10.22}$$

Ponieważ, wartości naprężeń stycznych są wprost proporcjonalne do odległości od osi obojętnej przekroju, aby uzyskać maksymalną ich wartość, należy podstawić do za ρ promień.

10.1.3. Wskaźniki wytrzymałości

Ponieważ to maksymalne naprężenie styczne definiuje de facto maksymalne obciążenie, jakie może przenieść dana część, to właśnie ta wartość jest najczęściej poszukiwana. W celu uproszczenia toku pracy inżyniera wprowadzono wskaźniki wytrzymałości dla różnych typów obciążenia. W przypadku zginania:



Rys. 10.5. Rozkład naprężeń normalnych, w przekroju, podczas zginania [5]

$$W_1 = \frac{J_z}{e_1} \tag{10.23}$$

$$W_2 = \frac{J_z}{e_2}.$$

(10.24)

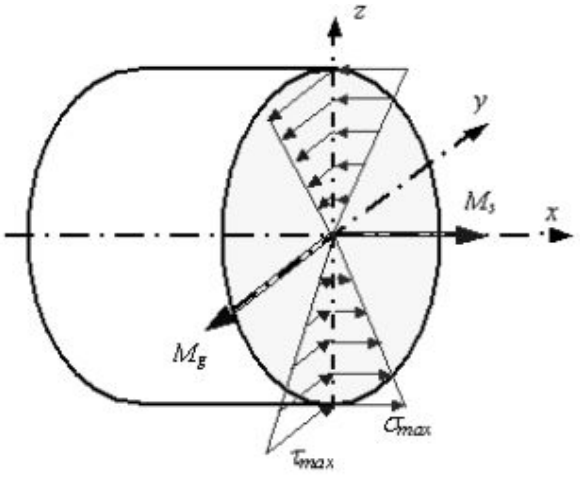
Dla przekrojów symetrycznych względem osi płaszczyzny wyznaczanej przez oś obojętna oraz której wektor normalny jest równoległy do osi prostopadłej do osi obojętnej (osi centralnej przekroju) oba te wskaźniki są równe.

W przypadku skręcania:

$$W_0 = \frac{J_0}{r}$$

(10.25)

10.1.4. Zginanie ze skręcaniem - złożony stan obciążenia



Rys. 10.6. Ilustracja rozkładu naprężeń normalnych i tnących w złożonym stanie naprężenia [5]

Największe naprężenia znajdują się na zewnętrznej powierzchni pręta, czyli punktach o największej odległości od osi obojętnej. Wyznacza je się za pomocą wzorów zawierających wcześniej wspomniane wskaźniki:

$$\sigma_{\max} = \frac{M_g}{W}, \qquad \tau_{\max} = \frac{M_s}{W_0}$$

(10.26,10.27)

Zakładając liniowo sprężyste właściwości materiałowe, możliwe jest zastosowanie hipotezy wytężeniowej. Przykładowo, dla płaskiego stanu naprężenia określonego składowymi σx = σmax, σy = 0, τxy = τmax naprężenia zredukowane według hipotezy Misesa wynoszą:

$$\sigma_{red} = \sqrt{\sigma_x^2 + \sigma_y^2 - \sigma_x\sigma_y + 3\tau_{xy}^2} = \sqrt{\sigma_{\max}^2 + 3\tau_{\max}^2}$$

(10.28)

Po podstawieniu zależności:

$$\sigma_{red} = \sqrt{\left(\frac{M_g}{W}\right)^2 + 3\left(\frac{M_s}{W_0}\right)^2} \tag{10.29}$$

Uwzględniając w (12.6) związek między wskaźnikami wytrzymałości na zginanie i skręcanie w prętach o przekroju kołowym i pierścieniowym **W0 = 2W:**

$$\sigma_{red} = \frac{\sqrt{M_g^2 + \frac{3}{4}M_s^2}}{W} \tag{10.30}$$

W ten sposób definiowany jest moment zredukowany.

$$M_{red} = \sqrt{M_g^2 + \frac{3}{4}M_s^2} \tag{10.31}$$

Warunek wytrzymałości wału zginanego i skręcanego zapisujemy w następujący sposób:

$$\sigma_{red} = \frac{M_{red}}{W} \leq \sigma_{dop} \tag{10.32}$$

Ocena wytrzymałości polega na porównaniu naprężeń zredukowanych, z naprężeniami dopuszczalnymi.

10.1.5. Moment zredukowany, a praktyka projektowa [3]

Wykorzystanie naprężeń zredukowanych w swojej klasycznej formie niesie ze sobą jeden problem. Tego typu obliczenia, nie mówią, jakiego typu zniszczenia można się spodziewać. Dlatego w praktyce projektowej, po wyznaczeniu przebiegów obciążeń gnących i skrętnych, sprawdza się, który typ obciążenia ma przewagę.

W zależności od przeważającego typu naprężeń, można je zredukować na dwa sposoby:

Przewaga naprężeń normalnych:

$$\sigma_z = \sqrt{\sigma^2 + (m\tau)^2} \tag{10.33}$$

Przewaga naprężeń stycznych:

$$\tau_z = \sqrt{\frac{\sigma^2}{m^2} + \tau^2}$$

(10.34)

Współczynnik m wyznaczany jest z zasady proporcjonalności naprężeń dopuszczalnych. W zależności, do jakich naprężeń zostaje zredukowany stan obciążenia, stosuje się odpowiedni wzór momentu zredukowanego, który uzyskujemy podstawiając następujące zależności:

$$\tau_s = \frac{M_s}{W_0}, \qquad \sigma_g = \frac{M_g}{W_x}, \qquad W_x = 0{,}5W_0$$

(10.35, 10.36, 10.37)

Przewaga naprężeń normalnych:

$$M_Z = \sqrt{{M_g}^2 + (\frac{m}{2}M_s)^2}$$

(10.38)

$$d \geq \sqrt[3]{\frac{32M_Z}{\pi k_g}}$$

(10.39)

Przewaga naprężeń normalnych:

$$M_Z = \sqrt{(\frac{2M_g}{m})^2 + {M_s}^2}$$

(10.40)

$$d \geq \sqrt[3]{\frac{16M_Z}{\pi k_s}}$$

(10.41)

10.2. Obliczenia sztywnościowe [4]

Warunek sztywnościowy jest równie ważnym kryterium projektowym. Zapewnia poprawne funkcjonowanie części takich jak przekładnie czy łożyska. Wały maszynowe, biorąc pod uwagę

ilość zmian przekrojów oraz przyłożonych obciążeń, są elementami trudnymi to analizowania za pomocą klasycznych metod badających sztywność (metody energetyczne oraz metoda Clebscha). Z pomocą przychodzi tu metoda elementów skończonych

10.2.1. Metoda elementów skończonych

Istota tej metody, polega na podziale kontinuum, pewnej ciągłości ciała, na skończoną liczbę podobszarów, nazywanymi elementami skończonymi (dyskretyzacja układu). Przed podziałem, badany przedmiot ma nieskończoną liczbę stopni swobody, która zmienia się w skończoną po jego przeprowadzeniu.

Załóżmy, że element skończony opisywany jest w kartezjańskim układzie współrzędnych x,y,z. Wektor u, opisuje przemieszczenie dowolnego punktu elementu i jest wyrażony za pomocą składowych zmiennych [u,v,w]^T - kolejno przemieszczenia w osiach x,y,z. Przez d oznaczony jest wektor postępowych przemieszczeń węzłowych elementu.

$$d = \lfloor d_i \rfloor \qquad i = 1,2,...,n_{en},$$

$$d_i = \lfloor d_{xi}, \quad d_{yi}, \quad d_{zi} \rfloor .$$

(10.42)

Inne typy przemieszczeń (obroty, promienie krzywizny etc.) również mogą być w nim zawarte. Analogicznie, macierz sił węzłowych:

$$p = \lfloor p_i \rfloor \qquad i = 1,2,...,n_{en},$$

$$p_i = \lfloor p_{xi}, \quad p_{yi}, \quad p_{zi} \rfloor .$$

(10.43)

Pole przemieszczeń pojedynczego elementu w funkcji przemieszczeń węzłów ma postać:

$$u = N \cdot d .$$

(10.44)

Gdzie N odpowiada macierzy kształtu. Ponieważ wektor u ma wymiary (3x1) , zaś wektor przemieszczeń węzłów wymiary liczby stopni swobody elementu 3x n, więc macierz funkcji próbnych, inaczej zwanych funkcjami kształtu, jest macierzą prostokątną o wymiarach 3xn.

$$\varepsilon = L \cdot u \;\Rightarrow\; \varepsilon = L \cdot N \cdot d \;\Rightarrow\; \varepsilon = B \cdot d$$

(10.45)

Każda ze składowych macierzy jest funkcją i określa wpływ danej składowej wektora przemieszczeń na przemieszczenie dowolnego punktu elementu. Zależność ε (u) otrzymuje się przez różniczkowanie stosownych wyrażeń na przemieszczenia. Macierz B opisuje odkształcenia w dowolnym punkcie elementu, powstałe przez jednostkowe przemieszczenie kolejnych stopni swobody węzłów. Z prawa Hooke'a:

$$\sigma = D \cdot \varepsilon \;\Rightarrow\; \sigma = D \cdot B \cdot d\,,$$

(10.46)

Gdzie iloczyn macierzy D i B opisuje zmiany naprężeń w funkcji przemieszczeń węzłów.

Zasada prac wirtualnych, mówi że jeżeli ciało w równowadze, jest podddane wirtualnym przemieszczeniom, wówczas praca wirtualna zewnętrznych obciążeń jest równa wirtualnej energii odkształcenia naprężeń wewnętrznych. W zapisie macierzowym:

$$\delta U_e = \delta W_e\,,$$

(10.47)

U - energia wewnętrzna

W - praca sił zewnętrznych

ð - wariacja (stan wirtualny, zgodny z więzami)

Wirtualny stan przemieszczeń węzłowych: ðd = ðdi (1,2,...,) Wówczas wirtualną energię układu i pracę sił wewnętrznych można wyrazić w postaci:

$$\delta U_e = \int_V \delta\varepsilon^T \cdot \sigma \cdot dV \qquad \text{i} \quad \delta W_e = \delta\, p^T \cdot p + \int_V \delta u^T \cdot b \cdot dV$$

(10.48, 10.49)

Podstawiając do zależności zasady prac wirtualnych (10.47):

$$\int_V \delta\varepsilon^T \cdot \sigma \cdot dV = \delta\, p^T \cdot p + \int_V \delta u^T \cdot b \cdot dV\,.$$

(10.50)

Uwzględniając wyrażenia (10.46 i 10.49) otrzymane zostaje:

$$\delta d^T \int_V B^T \cdot D \cdot \varepsilon \cdot dV = \delta d^T \cdot p + \delta d^T \int_V N^T \cdot b \cdot dV ,$$

(10.51)

Po kolejnym wykorzystaniu (10.47) i uproszczeniu przez **ðd^T:**

$$\left(\int_V B^T \cdot D \cdot B \cdot dV \right) \cdot d = p + \int_V N^T \cdot b \cdot dV$$

(10.52)

I ostatecznie:

$$K \cdot d = p$$

(10.53)

Gdzie K jest macierzą sztywności elementu, której składowe oznaczają fikcyjne siły w węzłach, spowodowane ich jednostkowymi przemieszczeniami.

Otrzymane w wcześniej równania uzyskuje się też poprzez użycie twierdzenia o minimum całkowitej energii potencjalnej. Mówi ono, że spośród wszystkich kinematycznie dopuszczalnych pól przemieszczeń spełnia się to, które całkowitej energii potencjalnej zapewnia minimum. Całkowita energia potencjalna układu wyraża się jako:

$$\Pi = U - W ,$$

(10.54)

U oznacza energię sprężystą a W jest pracą sił zewnętrznych. Dla ciała sprężystego, Π jest funkcjonałem kwadratowym, więc ma jedno globalne minimum. Energię Π można zapisać w postaci:

$$\Pi = \frac{1}{2} \int_{V_e} d^T \cdot B^T \cdot D \cdot B \cdot d \cdot dV - \int_{V_e} d^T \cdot N^T \cdot b \cdot dV - \int_{S_\sigma} d^T \cdot N^T \cdot p^* \cdot dS ;$$

(10.55)

$$\Pi = \frac{1}{2} \int_V \sigma^T \cdot \varepsilon \cdot dV - \int_V u^T \cdot b \cdot dV - \int_S u^T \cdot p^* \cdot dS ,$$

(10.56)

Gdzie p* jest obciążeniem brzegu S. Przyjmując interpolacje dla elementu o znanym kształcie:

$$u = N^T \cdot d, \qquad \varepsilon = B \cdot d, \qquad \sigma = D \cdot \varepsilon,$$

(10.57, 10.58, 10.59)

Ze stacjonarności wyrażenia wynika równowaga elementu:

$$\frac{\partial \Pi_e}{\partial d} = 0 \qquad \Rightarrow \qquad \int_{V_e} B^T \cdot D \cdot B \cdot dV \cdot d - p_e = K \cdot d - p_e ,$$

(10.60)

Gdzie pe opisuje siły węzłowe danego elementu, jako efekt obciążeń masowych b i powierzchniowych p*. Energia sprężysta pojedynczego elementu belkowego, pomijając wpływ ścinania:

$$U_e = \frac{1}{2}\int_{V_e} \sigma^T \cdot \varepsilon \cdot dV = \frac{1}{2}\int_{V_e} \varepsilon^T \cdot D \cdot \varepsilon \cdot dV = \frac{E}{2}\int_{V_e} \varepsilon_x{}^2 \cdot dV .$$

(10.61)

Przyjmując klasyczne założenie Belki Bernoulli'ego, czyli:

$$\varepsilon_x = -y \cdot \frac{d^2 v}{dx^2},$$

(10.62)

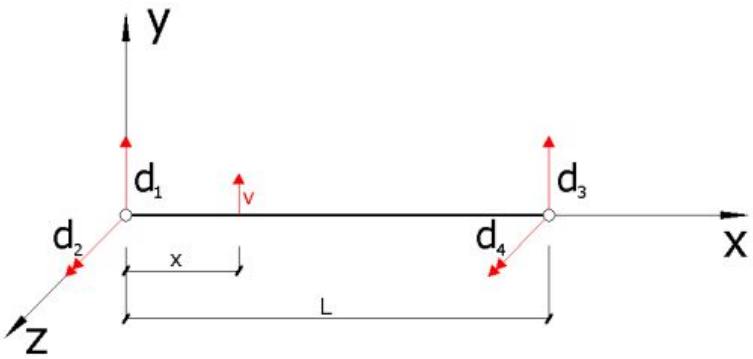
Wówczas energia wewnętrzna elementu zginanego wynosi:

$$U_e = \frac{E}{2}\int_{V_e}\left(-y \cdot \frac{d^2 v}{dx^2}\right)^2 \cdot dV = \frac{E}{2}\int_{V_e} y^2 \cdot \left(\frac{d^2 v}{dx^2}\right)^2 \cdot dV = \frac{E}{2}\int_0^1 \left(\frac{d^2 v}{dx^2}\right)^2 \iint_A dA\, dx = \frac{E \cdot I}{2}\int_0^1 \left(\frac{d^2 v}{dx^2}\right)^2 dx .$$

(10.63)

Równanie ilustruje energię wewnętrzną jako funkcje przemieszczeń.

10.2.2. Wyprowadzenie elementu belkowego



Rys. 10.7. Schemat belkowego elementu metody elementów skończonych [4]

Rozpatrzmy element belkowy, płaski, dwuwęzłowy zginany w płaszczyźnie XY. Wektor przemieszczeń węzłowych przyjmiemy w postaci d , gdzie przez oznaczono przemieszczenie

prostopadłe do osi pręta, zaś jest kątem obrotu przekroju. Indeksy 1,2 odnoszą się do numeracji węzłów. Zgodnie z założeniami klasycznej teorii belek kąty obrotu są pochodnymi przemieszczeń:

$$\phi_1 = \frac{dv_1}{dx}, \qquad \phi_2 = \frac{dv_2}{dx}. \tag{10.64}$$

Odpowiedni wektor sił węzłowych zawiera siły skupione działające w kierunku przemieszczeń oraz momenty zginające zgodne z kątami obrotów przekrojów. p = [p1 , m1 , p2, m2] Załóżmy funkcję przemieszczeń w postaci kompletnego wielomianu trzeciego stopnia:

$$v(x) = c_1 + c_2 \cdot x + c_3 \cdot x^2 + c_4 \cdot x^3. \tag{10.65}$$

Współczynniki c wyznaczone zostają z warunków brzegowych, wielkości przemieszczeń i kątów obrotów na końcu i początku elementu, równym składowym wektora d.

$$\text{dla } x = 0 \qquad v(0) = v_1 \qquad \text{oraz} \qquad \frac{dv(0)}{dx} = \phi_1$$

$$\text{dla } x = l \qquad v(l) = v_2 \qquad \text{oraz} \qquad \frac{dv(l)}{dx} = \phi_2 \tag{10.66, 10.67}$$

Po wyznaczeniu stałych, macierz funkcji kształtu:

$$N = \frac{1}{l^3} \cdot \left[2 \cdot x^3 - 3 \cdot l \cdot x^2 + l^3,\ l \cdot x^3 - 2 \cdot l^2 \cdot x^2 + x \cdot l^3,\ -2 \cdot x^3 + 3 \cdot l \cdot x^2,\ l \cdot x^3 - l^2 \cdot x^2 \right] \tag{10.68}$$

Funkcja kształtu, opisuje zmianę przemieszczenie, spowodowaną jednostkowymi przemieszczeniami węzłów. Jeśli założymy dalej prawdziwość hipotezy Eulera - Bernoulliego: wówczas przemieszczenie podłużne wyniesie:

$$u(x) = -y \cdot \frac{dv}{dx}, \tag{10.69}$$

a odkształcenie:

$$\varepsilon_x = \frac{du}{dx} = -y \cdot \frac{d^2v}{dx^2} = -y \cdot \kappa, \quad \text{gdzie} \quad \kappa = \frac{d^2v}{dx^2}. \tag{10.70}$$

Stąd, operator różniczkowy L, transformujący przemieszczenie w funkcji odległości od początku

elementu w odkształcenie epsilonx ma postać:

$$L = -y \cdot \frac{d^2}{dx^2},$$

(10.71)

Stąd macierz B = L*N:

$$B = L \cdot N = -\frac{y}{l^3} \cdot \left[12 \cdot x - 61,\ 61 \cdot x - 41^2,\ -12 \cdot x + 61,\ 61 \cdot x - 21^2\right].$$

(10.72)

Pamiętając, że dla tego prostego przypadku związek fizyczny ma postać σ = Eϵ (czyli operator D = E), otrzymujemy macierz sztywności dla elementu belkowego:

$$K_e = \frac{E \cdot I_z}{l^3} \cdot \begin{bmatrix} 16 & 6 \cdot l & -12 & 6 \cdot l \\ 6 \cdot l & 4 \cdot l^2 & -6 \cdot l & 2 \cdot l^2 \\ -12 & -6 \cdot l & 12 & -6 \cdot l \\ 6 \cdot l & 2 \cdot l^2 & -6 \cdot l & 4 \cdot l^2 \end{bmatrix},$$

(10.73)

Gdzie Iz oznacza moment bezwładności przekroju względem osi centralnej.

11. Backend, struktury danych

11.1. Globalna baza danych - obiekt *shaft* i *calculatedData*

Wszystkie informacje wprowadzane przez użytkownika przechowywane są w obiekcie `shaft`. Dane powstałe w wyniku przeprowadzania obliczeń zapisywane są w obiekcie `calculatedData`. Obydwa zostały zainicjowane jako literały obiektowe, odpowiadające wzorcowi projektowemu o nazwie “*Singleton*” (Istnieje tylko jedna instancja złożonego obiektu). Taka struktura danych jest łatwa w zarządzaniu i modyfikowaniu. Jej eksport polega wyłącznie na konwersji na ciąg znakowy zgodny z JSON. (W celu opisania, jaki typ danych domyślnie znajdzie się w danej właściwości obiektu, zastosowane są komentarze, czyli tekst po znakach ”`//`”, ignorowany przez interpreter).

```
        var shaft = {
            "FEM":{},// Zagnieżdżony obiekt, zawiera informacje o globalnych
właściwościach modelu MES
            "general":{ // Zagnieżdżony obiekt, zawiera informacje o danych
```

```
ogólnych. “” oznacza ciąg znaków, dowolna cyfra typ zmiennoprzecinkowy.
                "name":"",
                "person":"",
                "c": 1000,
                "T": 0,
                "k": 0,
                "omega": 0,
                "material":"",
                "E":210000,
                "zgo":1,
                "zsj":1,

            },
            "load": { // Zbiór tablic obiektów Load, modelujących obciążenie
w dwóch płaszczyznach, oraz obciążenie osiowe i skręcanie
                "XY":  [],
                "XZ":  [],
                "X": [],
            },
            "shape" :  [] // Tablica z obiektami Step, zapisuje dane o
kształcie
        };

        var calculatedData = { // Wykresy i równania momentów gnących
// plot - obiekt z właściwościami “x” i “y”, definiującymi współrzędne
punktów
// equationsList - tablica obiektów TorqueEquation - równań momentów
            "bending" : {
                "XY" : {
                    "plot" : {},
                    "equationsList": []
                },
                "XZ" : {
                    "plot" : {},
                    "equationsList": []
                },
            },
            "torsion" : {
                "plot" : {},
                "equationsList": []
            },
```

```
            "resultant" :{
                "bending" : {
                    "plot" : {},
                    "equationsList": []
                },
                "fullLoad" : {
                    "plot" : {},
                    "equationsList": []
                },
            "fullLoadCorrectedPlot" : {},
            "diameterPlot" : {}
            }
        };
```

Poszczególne właściwości obiektów są inicjalizowane w momencie uruchomienia programu. Zostają do nich przypisane kontenery, które posiadają określony z góry typ, lecz nie mają w sobie zapisanych danych (przykładowo: `[]`, czyli pusta tablica).

11.2. Przechowywanie lokalnych danych o poszczególnych obciążeniach i stopniach

Zarówno poszczególne stopnie wału jak zadawane obciążenia są zapisywane za pomocą instancji klas Step i Load. Udostępniają one następujące właściwości:

Klasa Load

```
function Load(typ, wartosc, coordinate){
    this.typ = typ; // Typ obciążenia, np.:Siła, moment skupiony...
    this.odleglosc = parseFloat(coordinate); // Współrzędna obciążenia
    if (typ !== "momentskrecajacyreakcyjny"){
        this.wartosc = wartosc; // Wartość obciążenia
    }
}
```

Klasa Step

```
function Step(type, diameter, centerCoordinate, start, end, length,FEM){

    this.type = type; // Typ stopnia, konstrukcyjny lub technologiczny
    this.diameter = parseFloat(diameter); // Średnica
      this.centerCoordinate = parseFloat(centerCoordinate); // Współrzędna
środka
    this.start = parseFloat(start); // Współrzędna początku
```

```
    this.end = parseFloat(end); // Współrzędna końca
    this.length = parseFloat(length); // Długość stopnia
     this.FEM = {}; // Obiekt zawierający dane do analiz MES, nie inicjowany
 w momencie inicjalizacji obiektu step

    this.momentOfInertia = { // Moment bezwładności wzgl. osi centralnej
        bending : Math.PI * Math.pow(diameter,4) / 64,
    };
 };
```

Jak widać powyżej, rolę klasy w JS, w wydaniu ES5 pełni jej konstruktor, można go odróżnić od zwykłej funkcji, dzięki temu, że jego nazwa rozpoczyna się od wielkiej litery. W zastosowaniu składni ES6 definicja powyższych klas wygląda następująco:

Klasa Load

```
class Load{
       constructor(typ, wartosc, coordinate ,plane){
        this.typ = typ;
        this.odleglosc = coordinate;
        this.plane = plane; //used in FEM calculations
        if (typ !== "momentskrecajacyreakcyjny"){
               this.wartosc = wartosc;
        }
}
```

Klasa Step

```
class Load{
       constructor(type, diameter, centerCoordinate, start, end, length,FEM){


    this.type = type;
    this.diameter = parseFloat(diameter);
    this.centerCoordinate = parseFloat(centerCoordinate);
    this.start = parseFloat(start);
    this.end = parseFloat(end);
    this.length = parseFloat(length);
```

```
    this.FEM = {};

    this.momentOfInertia = {
        bending : Math.PI * Math.pow(diameter,4) / 64,
    };
};

}
```

Klasa Step udostępnia więcej właściwości, niż jest potrzebnych do pełnej definicji stopnia. Dzięki temu komputer oblicza wszystkie potrzebne dane tylko jednokrotnie, podczas tworzenia obiektu. Zarówno obiekty z danymi o stopniach i obciążeniach, są przechowywane w tablicach, które są zapisane jako właściwości zagnieżdżonych obiektów. Tablice obciążenia zapisywane są w osobnych właściwościach. Taki sposób został wybrany ze względu na udostępniane przez JS przydatne właściwości:

1) Dynamiczny rozmiar tablic - rozmiar tablicy określany jest w trakcie działania programu. Dzięki temu nie musi być on znany przed jej deklaracją i inicjalizacją (potocznie: stworzeniem tablicy)

2) Możliwość dynamicznego odwołania się do właściwości obiektu - program w trakcie działania jest w stanie odwołać się do danych zapisanych pod konkretnymi właściwościami: obiekty to listy indeksowane za pomocą nazw właściwości. Pozwala to stworzyć funkcję, która dzięki parametrowi obsługuje wszystkie płaszczyzny obciążenia, zamiast tworzyć 3 osobne. Przykład i porównanie składni:

Odwołanie statyczne (właściwość, do której uzyskiwany jest dostęp, definiowana jest tylko przez programistę):

```
shaft.load.XY // Zwraca tablicę dla płaszczyzny XY
```

Odwołanie dynamiczne (w trakcie działania programu):

```
var plane = "XY";

shaft.load [plane]  // Zwraca tablicę dla płaszczyzny o nazwie zapisanej
w zmiennej plane
```

W obydwu przypadkach kod zwróci listę obciążeń w płaszczyźnie “XY”. Niemniej jednak w drugim przypadku, odwołanie do konkretnej właściwości realizowane jest z poziomu programu. Daje to duże możliwości w paradygmacie funkcyjnym.

3) Metoda `forEach()` - metoda tablicy przyjmująca jako parametr funkcje, która jest wykonywana dla każdego elementu tablicy, jest wygodniejszą formą klasycznej pętli. Jako

parametr przyjmowana jest referencja do funkcji, która później wykonywana jest dla każdego elementu tej struktury danych.

11.3. Analizator składniowy [13]

Analizator składniowy, nazywany też *parserem*, jest kluczowym elementem każdego ekosystemu języka programowania. Jego zadaniem jest analiza składniowa danych tekstowych pochodzących z plików kodu źródłowego. Pozwala ona na uzyskanie informacji o strukturze gramatycznej. Dzięki temu kompliator lub interpreter może “zrozumieć” intencje programisty, oraz może przepisać go na instrukcje zrozumiałe dla maszyny (kod maszynowy).

Parser zastosowanej w projekcie biblioteki Math.js, identyczne zadanie, jednak przedmiotem jego analizy są wyrażenia algebraiczne.. Konwertuje on je do postaci prostszej do komputerowego przetwarzania, a następnie wyznacza ich wartość

Poniżej przykładowy kod z wykorzystaniem *parsera* (w komentarzu zwracana wartość) [10]:

```
var f = math.parse('3x + x');
var simplified = math.simplify(f);
simplified.toString(); // '4 * x'
simplified.eval({x: 5}); // 20
```

Trzecia linijka zwraca string z uproszczonym równaniem, natomiast czwarta, wynik po podstawieniu za x liczby 4. Co ważne, równie dobrze w stringu może znajdować się więcej niewiadomych. Program je rozpoznaje, dzięki definiowanemu dynamicznie obiektowi, którego nazwy właściwości są jednocześnie nazwami niewiadomych, a ich wartości, podstawianymi liczbami.

11.3.1. Odwrócona notacja polska [13]

Odwrócona notacja polska (dalej ONP) - sposób zapisu wyrażeń algebraicznych w którym operatory działań matematycznych są zapisane na końcu danego działania w wyrażeniu (zapis postfiksowy) a nie pomiędzy liczbami i zmiennymi (operandami). Taki sposób zapisu pozwala na pełne odejście od wykorzystywania nawiasów, gdyż kolejność wykonywania działań jest określona przez kolejność operatorów i operandów.

Przykłady:

(2 + 3) * 5 ⇔ 2 3 + 5 *

((2 + 7) / 3 + (14 - 3) * 4) / 2 ⇔ 2 7 + 3 / 14 3 - 4 * + 2 /

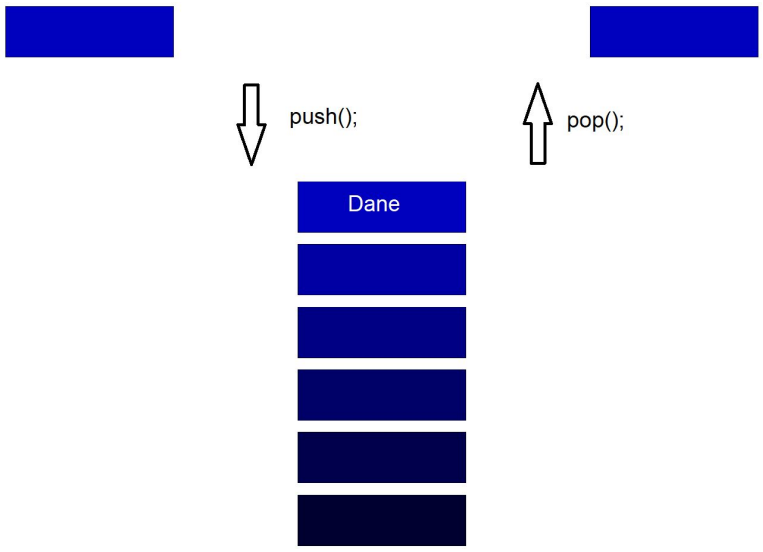
ONP upraszcza wykonywanie na komputerze obliczeń. Zarówno algorytm konwersji notacji konwencjonalnej (infiksowej) na odwrotną notację polską (postfiksową), jak i algorytm obliczania wartości wyrażenia w ONP są bardzo proste i wykorzystują stos.

11.3.2. Stos

Stos to struktura danych wczytująca zawarte w niej informacje zgodnie z zasadą *LIFO - last in, first out* (rys. 11.1.) - obiekt ostatnio odłożony na stos, będzie pierwszym obiektem, który zostanie z niego wczytany.

Na tej strukturze danych można wykonywać następujące operacje:

- push(obiekt) – czyli odłożenie obiektu na stos
- pop() – ściągnięcie obiektu ze stosu i zwrócenie jego wartości
- isEmpty() - sprawdzenie czy na stosie znajdują się już jakieś obiekty

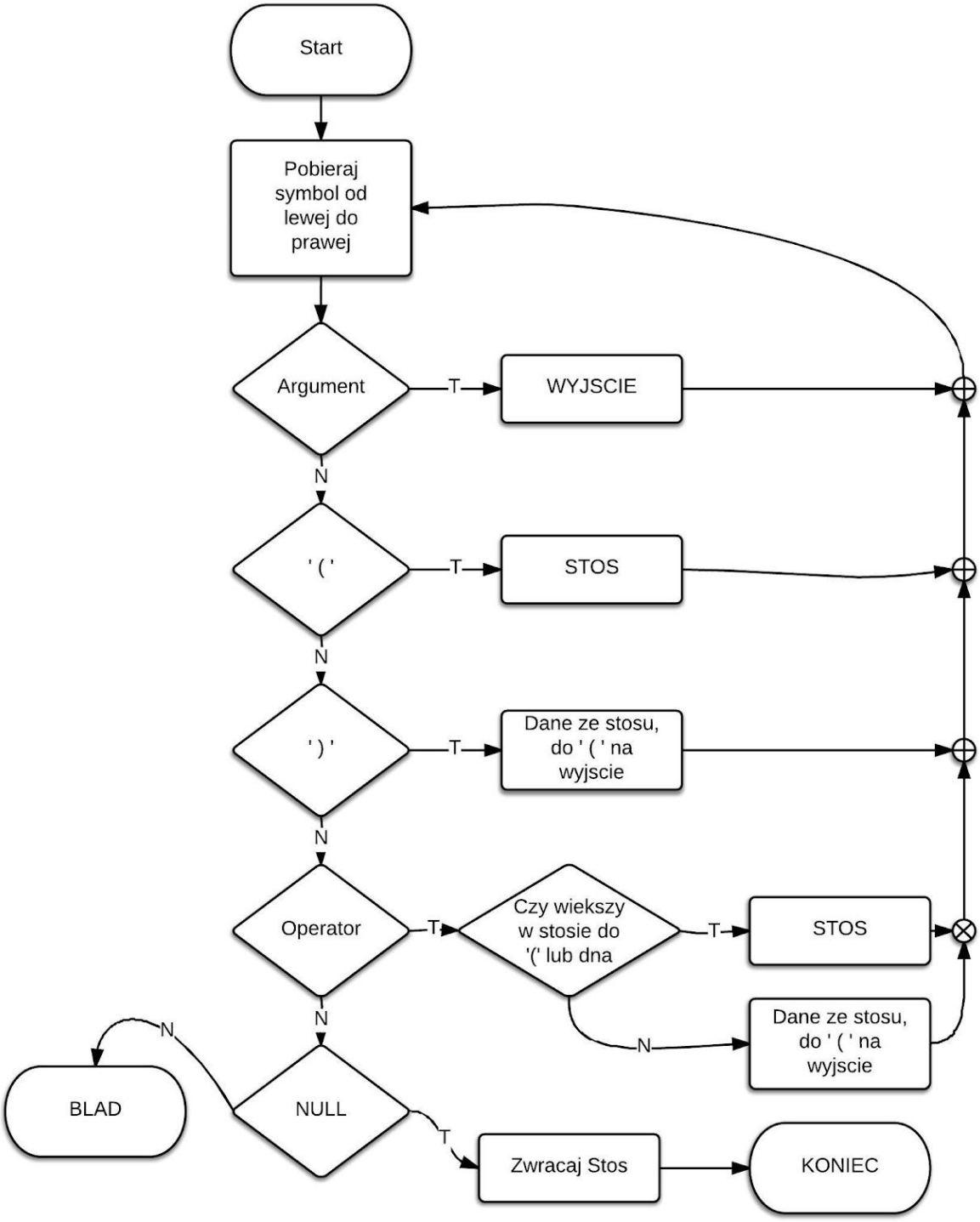


Rys. 11.1. Wizualizacja działania stosu

11.3.3. Algorytm “Stacji rozrządowej”

Jest to algorytm pozwalający na konwersję zapisu wyrażeń algebraicznych do ONP. Jego autorem jest Edsger Dijkstra. Zasada działania została przedstawiona na rysunku 11.2.

Pojawiający się w ostatnim blokach decyzyjnym symbol ,,NULL” warunkuje obecność innych symboli niż operatory lub argumenty. Zazwyczaj pojawia się on na końcu W momencie, kiedy w transformowanym lub obliczanym wyrażeniu znajdzie się nieobsługiwany znak (przykładowo: ‘@’), algorytm zwraca błąd.



Rys. 11.2. Algorytm Dijkstry - konwersja wyrażenia w notacji infixowej na ONP

Działanie algorytmu na przykładzie:

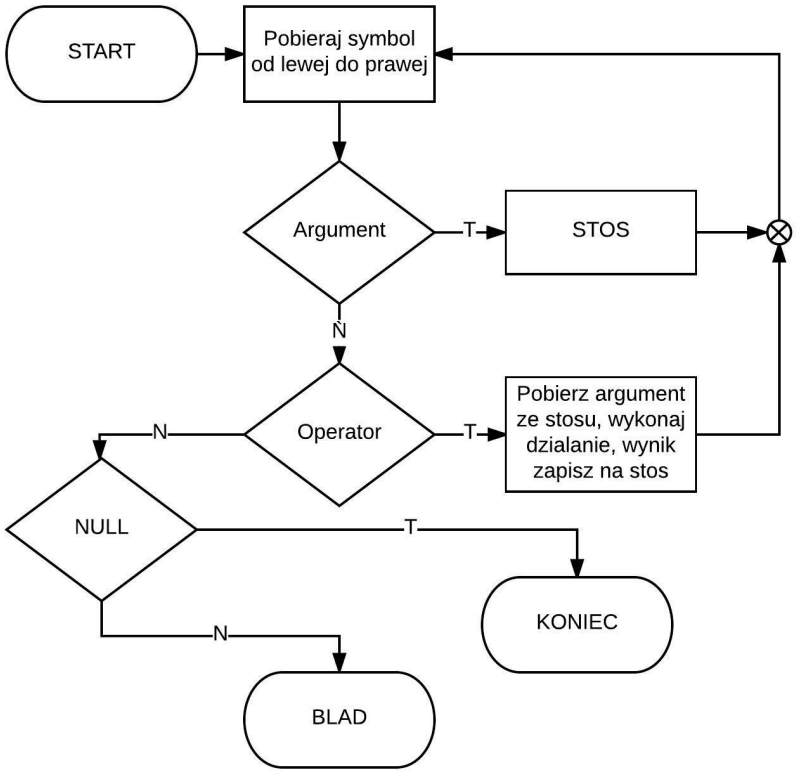
23 + a * (b * c + 55 / e) ⇔ 23 a b c * 55 e / + * +

Tabl. 11.1. Transformacja wyrażenia algebraicznego na odwróconą notację polską

Krok	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Wejście	23	+	a	*	(	b	*	c	+	55	/	e	)		Koniec
Stos		+	+	*+	*+	(*+	*(*+	*(*+	+*(*+	+*(*+	/+*(*+	/+*(*+	/+*(*+	*+	
Wyjście	23		a			b		c	*	55		e	/+	*+	

11.3.4. Algorytm obliczania wartości wyrażenia zapisanego w odwróconej notacji polskiej

Do obliczenia wartości wyrażenia ONP potrzebny jest dodatkowy algorytm (rys. 11.3), który również wykorzystuje stos.



Rys. 11.3. Algorytm Dijkstry - obliczanie wartości wyrażenia zapisanego w ONP

Przykład: Wyrażenie ONP 37 2 3 4 * 10 5 / +*+

Tabl. 11.2. Obliczanie wartości wyrażenia zapisanego w odwróconej notacji polskiej

Krok	1	2	3	4	5	6	7	8	9	10	11
Wejście	37	2	3	4	*	10	5	/	+	*	+
Stos	37	37 2	37 2 3	37 2 3 4	37 2 12	37 2 12 10	37 2 12 10 5	37 2 12 12 2	372 14	37 28	75

12. Komputerowa realizacja obliczeń w programie Shaft.js

12.1. Obliczenia wytrzymałościowe

Dzięki zastosowaniu *parsera* tok obliczeń sztywnościowych wygląda niemal identycznie jak podczas rozwiązywania klasycznego zadania projektowego. Dla każdego etapu zostaną przedstawione dane wejściowe, wyjściowe, ich struktura w programie oraz w niektórych przypadkach rysunek, wizualizacja pomagająca zrozumieć działanie etapu obliczeniowego. W tym celu równania sumy geometrycznej momentów celowo zostaną przedstawione w pełnej formie, bez redukcji wyrazów podobnych.

12.1.1. Obliczenie pionowych reakcji podpór

Do wyznaczenia reakcji w obu płaszczyznach wykorzystywane są równania równowagi sił i momentów. Układ współrzędnych, zawsze przyjęty jest w lewej podporze.

Program traktuje siły reakcji jako siły skupione. Po ich obliczeniu, dodawane są one do odpowiadającej im płaszczyzny obciążenia. W rezultacie po zsumowaniu wszystkich sił z danej tablicy, otrzymamy zero.

Jest to jedyna funkcja obliczeniowa, która musi zostać zmodyfikowana w celu wprowadzenia obsługi projektowania wałów jednokrotnie i wielokrotnie statycznie niewyznaczalnych.

Dane wejściowe: - tablica sił i momentów skupionych

```
shaft.load.XY
▼(3) [Load, Load, Load]
  ▶0: Load {typ: "sila", odleglosc: 300, plane: "XY", wartosc: 600}
  ▶1: Load {typ: "momentskupiony", odleglosc: 750, plane: "XY", wartosc: 200000}
  ▶2: Load {typ: "momentskupiony", odleglosc: 1100, plane: "XY", wartosc: 300}
   length: 3
```

Dane wyjściowe: - tablica sił i momentów skupionych zawierająca również siły reakcji podpór

```
shaft.load.XY
▼(5) [Load, Load, Load, Load, Load]
  ▶0: Load {typ: "reaction", odleglosc: 0, plane: "XY", wartosc: -219.7}
  ▶1: Load {typ: "sila", odleglosc: 300, plane: "XY", wartosc: 600}
  ▶2: Load {typ: "momentskupiony", odleglosc: 750, plane: "XY", wartosc: 200000}
  ▶3: Load {typ: "reaction", odleglosc: "1000", plane: "XY", wartosc: -380.3}
  ▶4: Load {typ: "momentskupiony", odleglosc: 1100, plane: "XY", wartosc: 300}
   length: 5
```

12.1.2. Obliczenie reakcji osiowej oraz wartości momentu skręcającego reakcyjnego

Ta funkcja jest uruchamiana tylko jeśli w badanym przypadku występuje siła osiowa i/lub obciążenie skręcające. Wyznacza reakcję skrętną (wartość momentu reakcyjnego skręcającego) oraz reakcję osiową. Również wykorzystuje równania statyki.

Dane wejściowe - tablica shaft.load.X zawierające dane o momentach skręcających i silach osiowych.

```
▼(4) [Load, Load, Load, Load]
  ▶0: Load {typ: "momentskrecajacy", odleglosc: 100, plane: "XY", wartosc: 300}
  ▶1: Load {typ: "silaosiowa", odleglosc: 300, plane: "XY", wartosc: 500}
  ▶2: Load {typ: "momentskrecajacyreakcyjny", odleglosc: 600, plane: "XY", value: ƒ}
  ▶3: Load {typ: "momentskrecajacy", odleglosc: 700, plane: "XY", wartosc: 500}
   length: 4
  ▶__proto__: Array(0)
```

Dane wyjściowe - tablica shaft.load.X zawierające dane o momentach skręcających i siłach osiowych zawierająca obliczoną wartość momentu reakcyjnego skręcającego

```
▼(4) [Load, Load, Load, Load]
  ▶0: Load {typ: "momentskrecajacy", odleglosc: 100, plane: "XY", wartosc: 300}
  ▶1: Load {typ: "silaosiowa", odleglosc: 300, plane: "XY", wartosc: 500}
  ▶2: Load {typ: "momentskrecajacyreakcyjny", odleglosc: 600, plane: "XY", value: ƒ, wartosc: -800}
  ▶3: Load {typ: "momentskrecajacy", odleglosc: 700, plane: "XY", wartosc: 500}
   length: 4
  ▶__proto__: Array(0)
```

12.1.3. Wyznaczenie równań momentów

Funkcja ta jest przeznaczona do budowy równań momentu, względem początku układu współrzędnych, czyli pierwszej podpory.

Omawiana funkcja jest uniwersalna, więc może zostać wykorzystana, zarówno do wyznaczenia równań momentów gnących jak i skręcających. Dla momentów skręcających wymagane jest umieszczenie przez program, dodatkowych, zerowych obciążeń skrętnych w miejscu podpór wału. Powyższy warunek wynika z metodyki obliczania przebiegów momentu wypadkowego.

Instrukcje wykonywane przez funkcję możemy podzielić na 3 etapy:

Preprocessing

Po inwokacji, funkcja sprawdza, czy parametr wejściowy jest ciągiem znaków, czy referencją do tablicy obciążenia (specyficzna implementacja przeładowywania metod). Następnie sprawdzane jest, czy wejściowa tablica reprezentuje płaszczyznę obciążenia (XY i XZ), czy oś X. W drugim przypadku, wykonywana jest jej kopia, do której dodawane są wyżej wspomniane obciążenia skrętne. Nie mają one wpływu na przebieg, gdyż ich wartość wynosi 0 Nmm. Tak przygotowane tablice trafiają do następnego etapu. Definiowana i inicjalizowana jest również zmienna `eq`, zawierająca powstałe w przyszłości równanie, oraz zwracana tablica `Arr`.

Właściwa budowa równań momentu gnącego

Uruchamiana zostaje metoda `forEach()`, która dla każdego elementu tablicy, wykonuje wprowadzoną jako parametr anonimową funkcję. Dla każdego elementu wykonywana jest następująca procedura:

1.1) Jeżeli elementem jest siła, lub reakcja dodaj do `eq` wypadkowy ciąg znakowy:

```
element.wartość + "*" "( x - " + (element.odleglosc).toString() + " ) +0"
```

1.2) Jeżeli elementem jest moment skupiony, dodaj do eq ciąg znaków zapisany pod zmienną:

```
element.wartość
```

2) Po dodaniu, stwórz obiekt klasy `TorqueEquation` i zainicjuj jego właściwości w następujący sposób:

`this.equation` - Stworzone wcześniej równanie
`this.start` - współrzędna siły
`this.end` - współrzędna następnej siły

3) Dodaj obiekt do zwracanej tablicy `Arr`

Uwaga: po zakończeniu pojedynczej iteracji w pętli `forEach()`, zawartość zmiennej eq nie jest kasowana. Do budowy równań pochodzących od kolejnych obciążeń (n, numer pozycji), wykorzystujemy poprzednio stworzone równanie (równanie dla obciążenia n-1). Proces ma charakter przyrostowy. Dzięki temu pętla wykonuje mniej iteracji, a program wykonuje obliczenia szybciej.

Dodanie ciągu znakowego `"+0",` jest wymagane ze względu na zasadę działania *parsera*. Aby połączyć poprzednie równanie z następnym, wykorzystywany jest `"+"` na końcu. Jednak, w momencie, gdy *parser* analizuje równanie, po rozpoznaniu plusa, oczekuje następującej po nim liczby. W momencie, gdy jej nie ma, zwraca błąd.

Postprocessing

Zostają sprawdzone następujące warunki dla kolejnych obciążeń:

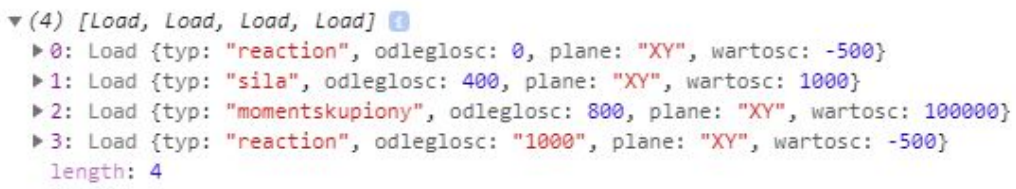
1) Czy istnieje równanie, dla którego początek i koniec przedziału są równe? - jest to efekt zamodelowania dwóch lub więcej sił w tym samym miejscu. Program nie pozwoli wprowadzić użytkownikowi takich samych typów obciążenia w jednym punkcie, lecz błąd występuje również przy dwóch różnych typach. W przypadku wykrycia, takie równanie jest usuwane.

2) Czy istnieje równanie pochodzące od ostaniej siły? W takim przypadku również jest ono usuwane.

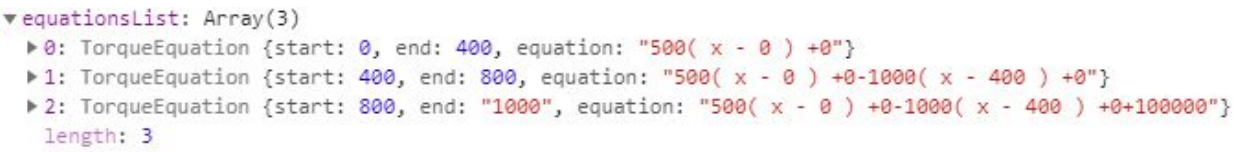
Na końcu zwrócona zostaje tablica `Arr`, która przypisywana jest do odpowiedniej właściwości obiektu `calculatedData`.

Uwaga: biblioteka math.js oferuje również możliwość upraszczania wyrażeń algebraicznych. Powoduje to jednak spadek wydajności oraz nie ilustruje zasady działania tak dobrze, jak przed uproszczeniem.

Dane wejściowe - tablica sił i momentów skupionych z obliczonymi reakcjami podpór



Dane wyjściowe - tablica równań przebiegów momentów gnących



12.1.4. Wyznaczanie równań geometrycznej sumy przebiegu momentów (równania momentu wypadkowego)

Ponieważ równania i przebiegi momentów definiują ich wartość jedynie na końcu i początku danego przedziału, niemożliwe jest proste dodanie ich wartości. Dzieje się tak, gdyż granice przedziałów w obu płaszczyznach mają różne wartości współrzędnej x. Istnieją dwa rozwiązania:

1) Dodaniu siły w dowolnej płaszczyźnie musi towarzyszyć dodanie zerowego obciążenia, w tym punkcie, w pozostałych płaszczyznach - wtedy wszystkie przedziały są identyczne.

2) Podczas dodawania równań, tworzony jest nowy przebieg, z nowymi równaniami, z przedziałami, których początki i końce pochodzą od sił z obu płaszczyzn.

W programie zostało zastosowane rozwiązanie nr 2. Większa liczba obciążeń, nawet tych o zerowej wartości, wymaga przeprowadzenia większej liczby iteracji przy niemalże każdej funkcji.

Funkcję można podzielić na dwa etapy:

1) Preprocessing

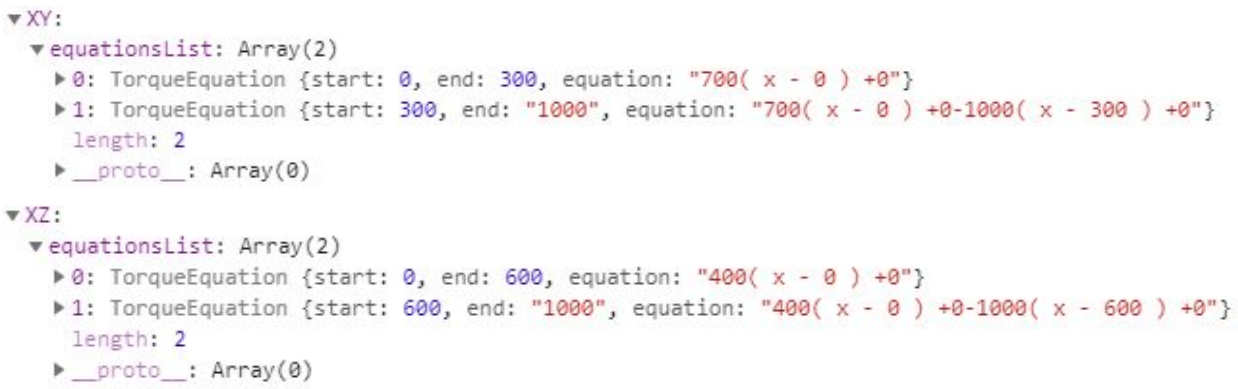
W pierwszym etapie, tworzona jest tablica zawierająca wszystkie współrzędne końca i początku przedziałów, które pochodzącą z obu płaszczyzn. Program zaczyna od ich sortowania, następnie usuwa wartości występujące więcej niż dwukrotnie. Na końcu usuwa duplikaty pierwszej i ostatniej współrzędnej. W efekcie powstaje tablica, której kolejne pary elementów, stanowią granice nowych przedziałów.

2) Właściwa funkcja

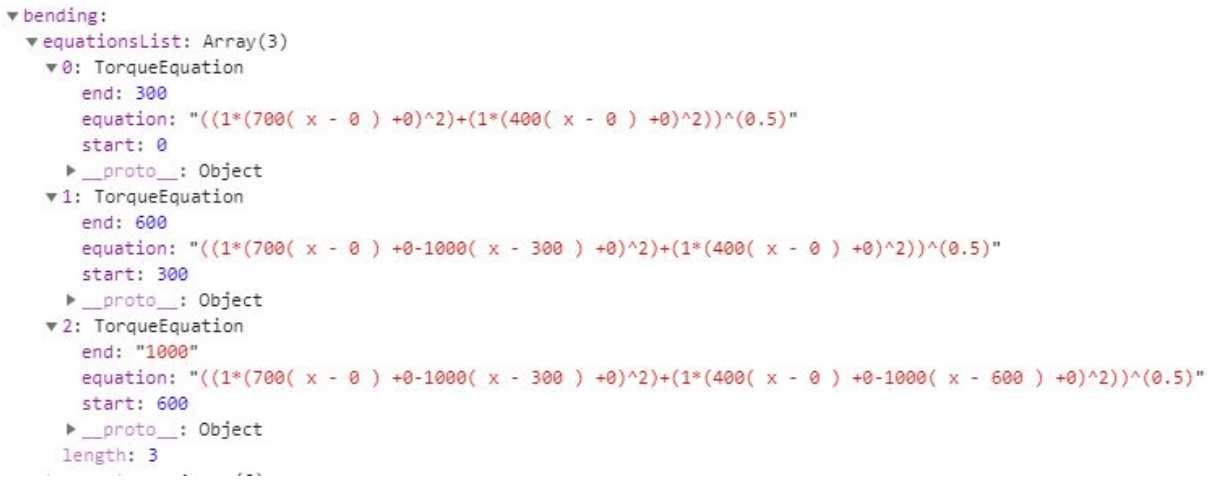
Tablica argumentów zostaje podzielona na n- tablic dwuelementowych, które reprezentują początek i koniec przedziału momentu. Przy czym ,,n'' oznacza liczbę przedziałów. Dla każdego nowego przedziału dobierane są równania w taki sposób, że nowy przedział zawiera się w dziedzinach dwóch równań wejściowych.

Dodatkowo funkcja udostępnia możliwość zdefiniowania parametru, mnożnika poszczególnych równań składowych - dzięki czemu może zostać wykorzystana również do wyznaczenia przebiegu momentu zredukowanego.

Dane wejściowe - dwie tablice z przebiegami momentów gnących i odpowiadające im współczynniki



Dane wyjściowe - tablica równań przebiegów wypadkowych momentów



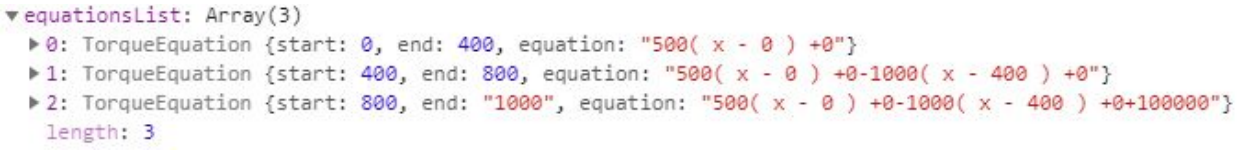
12.1.5. Wyznaczenie punktów przebiegów momentów

W tej funkcji wykorzystywany jest *parser*. Jako parametr przyjmowana jest tablica z równaniami lub ciąg znaków definiujący płaszczyznę. Zwraca ona obiekt `data` z dwiema właściwościami: x i y, będącymi tablicami zawierającymi współrzędne punktów osi rzędnych i odciętych, czyli domyślny typ danych obsługiwany przez Plot.ly (biblioteka generująca wykres). Swoje zastosowanie ma tu też wyżej opisana metoda `forEach()`. Dla każdego z równań wykonywana jest procedura:

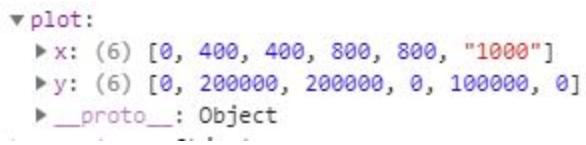
1) Dodaj do tablicy `data.x` argumentów współrzędną początku równania
2) Dodaj do tablicy `data.x` argumentów współrzędną końca równania
3) Za pomocą *parsera*,oblicz wartość odpowiadającej współrzędnej początku równania i dodaj ją do tablicy `data.y`
4) Za pomocą *parsera*, oblicz wartość odpowiadającej współrzędnej końca równania i dodaj ją do tablicy `data.y`

Po iteracji następuje zwrot obiektu `data`. Przypisywany jest on do odpowiadającej mu właściwości obiektu z wynikami obliczeń (`calculatedData`). Tak przygotowane dane są gotowe do wyświetlenia przez Plot.ly

Dane wejściowe - tablica równań przebiegów wypadkowych momentów



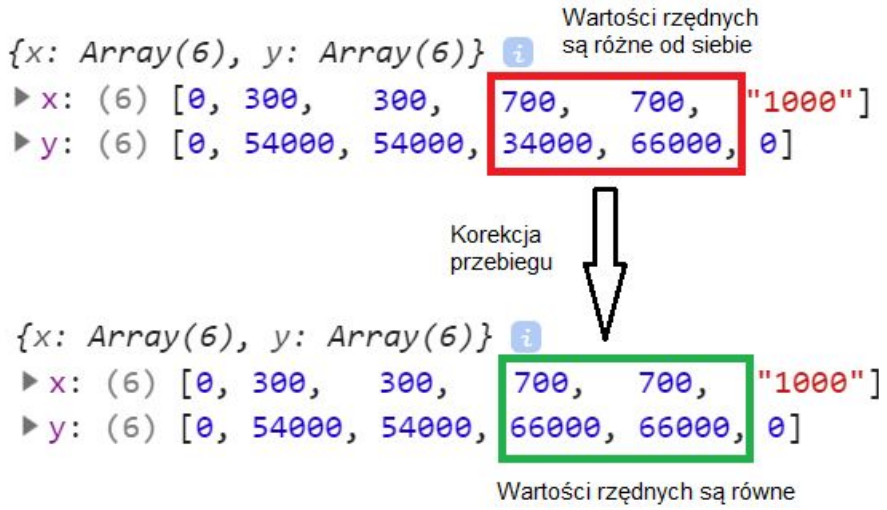
Dane wyjściowe - obiekt z dwoma właściwościami: x i y, będącymi tablicami zawierającymi współrzędne punktów



12.1.6.Korekcja wykresu wypadkowego obciążenia

W wyniku obecności momentów skupionych w obciążeniu, na wykresie momentu gnącego powstaje skok . Brak skorygowania objawiałby się skokami na wykresie zarysu teoretycznego.

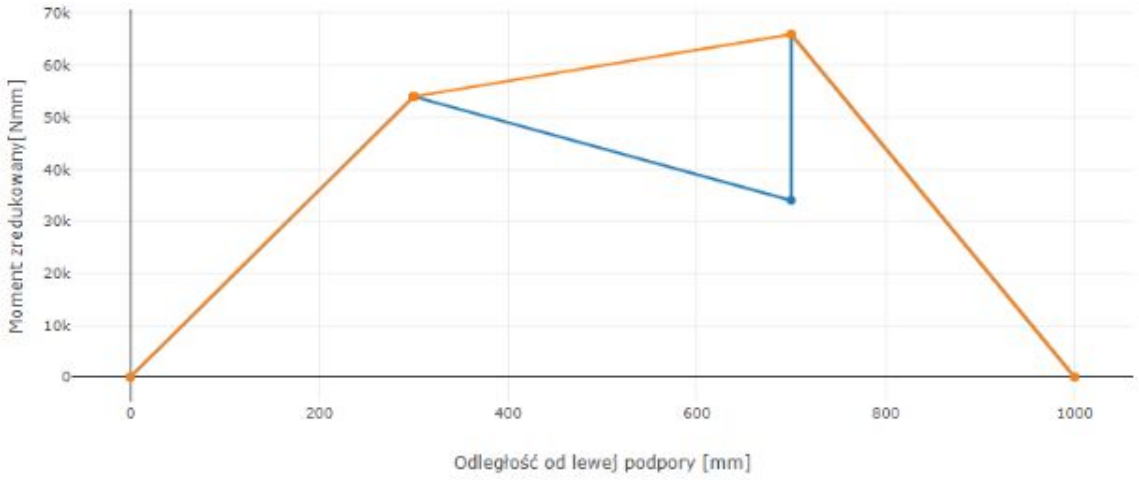
Każdy wykres momentów jest reprezentowany przez zbiór punktów, powstałych przez agregację współrzędnych granic przedziałów. Granice każdego przedziału jest reprezentowana przez dwa punkty, których odcięte (pierwsze współrzędne, x) są sobie równe, natomiast rzędne tych punktów (drugie współrzędne, y) mogą być równe lub różne od siebie. Jeżeli rzędne różnią się od siebie, to program realizuje funkcję nadpisującą wartość mniejszej rzędnej, przypisując jej wartości większej rzędnej (rys. 12.1). W ten sposób eliminowane są skoki wartości momentu zastępczego, a jego przebieg reprezentowany jest przez punkty o wartościach maksymalnych momentu w kolejnych przedziałach, zależnie od zadanego obciążenia na długości wału.



Rys. 12.1. Ilustracja zasady działania funkcji korygującej przebiegi wypadkowego momentu gnącego - widok tablic

Dane wejściowe - obiekt z dwoma właściwościami: x i y, będącymi tablicami zawierającymi współrzędne punktów

Dane wyjściowe - j.w. lecz z odpowiednio zmienionymi wartościami w tablicy y



Rys. 12.2. Ilustracja działania funkcji korygującej przebiegi wypadkowego momentu gnącego. Na niebiesko zaznaczono wykres przed korekcją, a na pomarańczowo wykres po korekcji

12.1.7. Wyznaczenie zarysu teoretycznego

Na podstawie przygotowanego przez poprzednie funkcje przebiegu momentu zredukowanego generowany jest zarys teoretyczny (rys. 12.2). Funkcja otrzymuje w parametrze tablicę

z przebiegiem momentu zredukowanego, po czym dzieli go na przedziały. Dla każdego przedziału, składającego sie z dwóch skrajnych punktów, o współrzędnych x1 i y1 oraz x2 i y2 wyznaczany jest wzór przechodzącej przez nie prostej:

$$y = ax + b \tag{12.1}$$

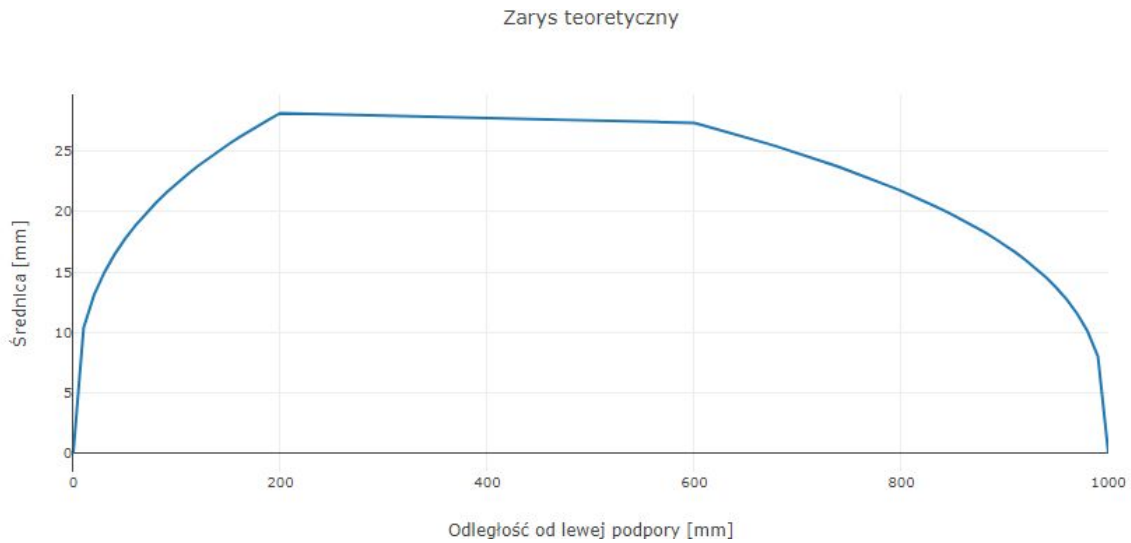
gdzie:

$$a = \frac{y_2 - y_1}{x_2 - x_1} \tag{12.2}$$

$$b = \frac{y_2 - y_1}{x_2 - x_1} - y_1 \tag{12.3}$$

Na podstawie powyższych wyrażeń, wyznaczany jest moment zredukowany w funkcji odległości od podpory. Z jego przebiegu wyznaczany jest zarys teoretyczny. Krok obliczeniowy ustala się na podstawie danych wprowadzonych przez użytkownika w zakładce “Dane”.

W wersji zrefaktoryzowanej, daną wejściową jest nie przebieg momentu zredukowanego, lecz jego równania. Zostały one już i tak wyznaczone w poprzednim etapie obliczeń. Dzięki temu udało się znacząco zredukować zapotrzebowanie na moc obliczeniową.
Dane wejściowe - przebieg momentu zredukowanego (rys. 12.1), a w nowszej wersji jego równania
Dane wyjściowe - wykres zarysu teoretycznego (rys. 12.2)



Rys. 12.3. Zarys teoretyczny generowany przez program

12.2. Obliczenia sztywnościowe

12.2.1. Funkcje pomocnicze

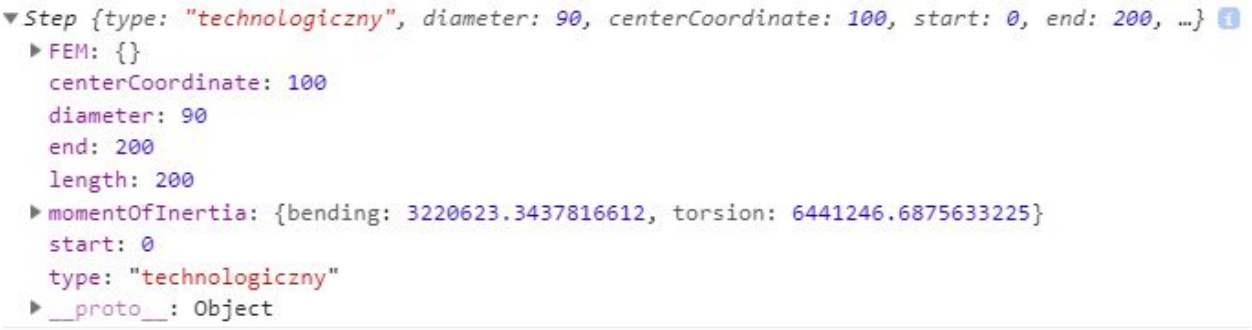
W obliczeniach sztywnościowych wykorzystane są następujące funkcje pomocnicze, uczestniczące w budowie modelu MES wału.

1) Budowa macierzy sztywności elementu belkowego

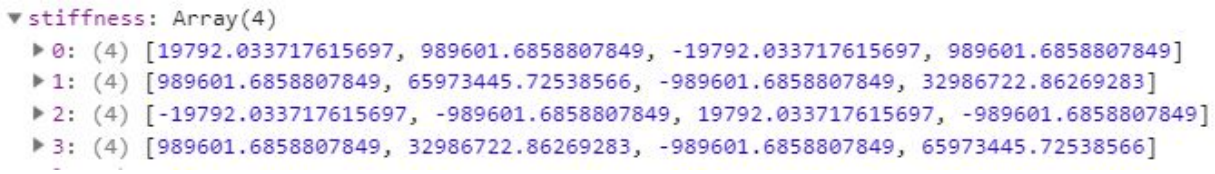
Funkcja ta, za pomocą zagnieżdżonych pętli tworzy i zwraca macierz sztywności dla zadanych w parametrach danych. W macierzy znajdują się już przeliczone współczynniki.

Przyjmuje ona dwa parametry. Pierwszy, `step` jest obowiązkowy i jest obiektem, którego właściwości definiują kształt elementu MES. Drugi jest opcjonalny i zawiera w sobie zadaną długość elementu skończonego. Jeśli funkcja będzie inwokowana bez niego, zostanie zwrócony element o pełnej długości odczytanej ze `step`.

Dane wejściowe - obiekt definiujący średnicę i długość elementu skończonego



Dane wyjściowe - lokalna macierz sztywności dla wejściowych danych



2) Agregacja wektorów obciążeń i macierzy sztywności

Bliźniacze funkcje, różniące się liczbą wymiarów wejściowych i wyjściowych danych. Dodają do siebie macierze w specjalny, wymagany w MES sposób.

Warto zwrócić uwagę na sposób “przepływu danych” w funkcji. W przypadku złożonych typów

danych, JavaScript przekazuje “wskazanie” (ByReference) na ten obiekt. W następującym przypadku:

```
var a =  [1,2,3]; // Tablica liczb
var b = a; // Przypisanie "wskazania"
b.push(4); // Dodanie czwartego elementu
```

Zmienne a i b, po wyświetleniu, zwrócą następującą tablicę: `[1,2,3,4]`.

Aby obejść ten problem należało wymusić kopiowanie tablic za pomocą metody **`.splice()`**, domyślnie używanej do zamiany wartości poszczególnych elementów.

Dane wejściowe - Dwa wektory obciążenia

```
var a = [1,1,1,1]
var b = [1,1,1,1]
```

Dane wyjściowe - Nowy wektor obciążenia o długości równej l1 + l2 - 2

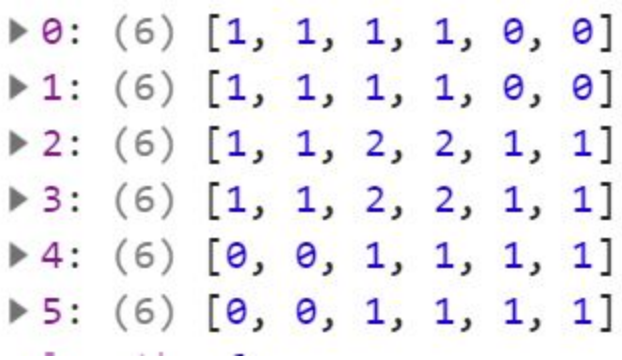
```
[1, 1, 2, 2, 1, 1]
```

Dane wejściowe - Dwie dowolne kwadratowe macierze, o dowolnych wymiarach boków (docelowo macierze sztywności)

```
var a = [[1,1,1,1],
         [1,1,1,1],
         [1,1,1,1],
         [1,1,1,1]]

var b = [[1,1,1,1],
         [1,1,1,1],
         [1,1,1,1],
         [1,1,1,1]]
```

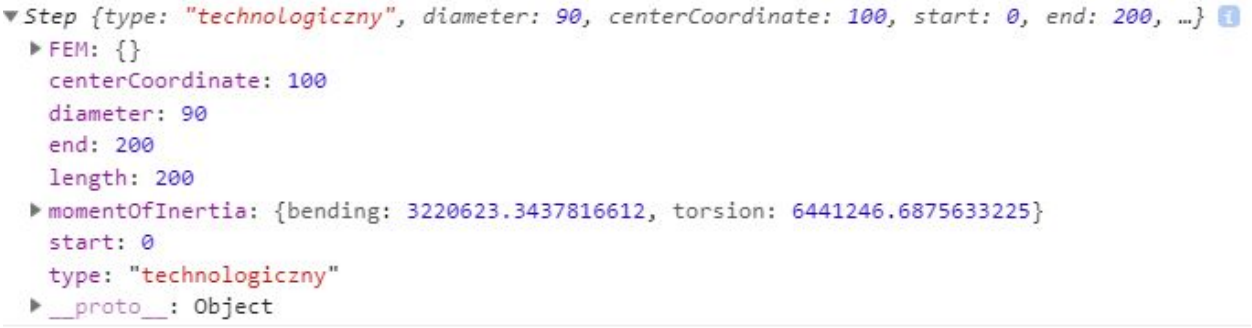
Dane wyjściowe - Jedna kwadratowa macierz obciążenia o długości boku równej l1 + l2 - 2



3) Weryfikacja obciążenia stopnia

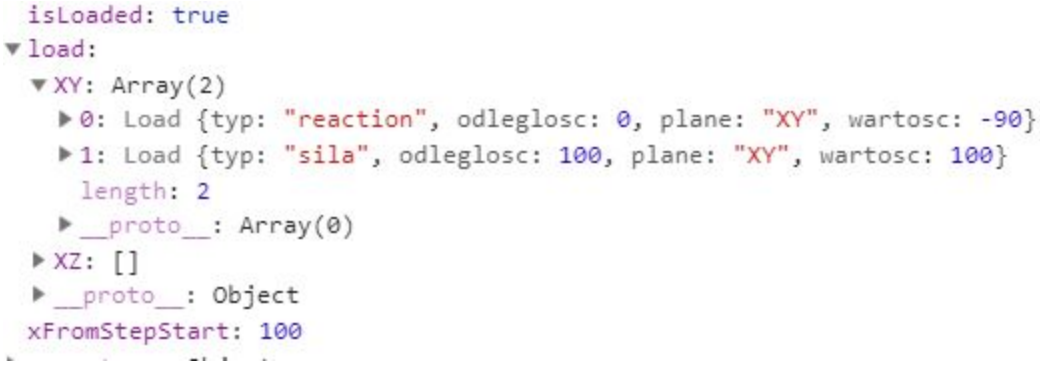
Funkcja sprawdza, czy pomiędzy początkiem i końcem zadanego stopnia znajduje się obciążenie. Jeśli tak, zwraca obiekt, który zawiera ich listę Jeśli nie, zwraca wartość logiczną `false`.

Dane wejściowe: - obiekt zawierający dane o parametrach geometrycznych stopnia



oraz tablica z zagregowanym obciążeniem ze wszystkich tablic.

Dane wyjściowe: obiekt zawierający informacje logiczną o obciążeniu (w danym przedziale istnieje lub nie), oraz obiekt, którego zagnieżdżonymi właściwościami są tablice zawierające informacje o obciążeniu tego stopnia na poszczególnych płaszczyznach



Dla stopnia nieobciążonego zwracany jest obiekt z pojedynczą właściwością `isLoaded`:

```
isLoaded: false
```

12.2.2. Procedura budowy i analizy numerycznej modelu MES

1) Budowa macierzy sztywności poszczególnych stopni oraz ich wektorów obciążeń

Model MES wału jest budowany na podstawie kształtu i obciążenia. Węzły elementów są umieszczane w miejscach zmiany przekroju lub przyłożenia obciążenia dowolnego typu.

Dla każdego stopnia wału, przy pomocy metody `forEach()` sprawdzane jest, czy stopień jest obciążony. Jeśli nie, tworzona dla niego jest macierz sztywności, oraz wektor obciążenia równy `[0,0,0,0]`.

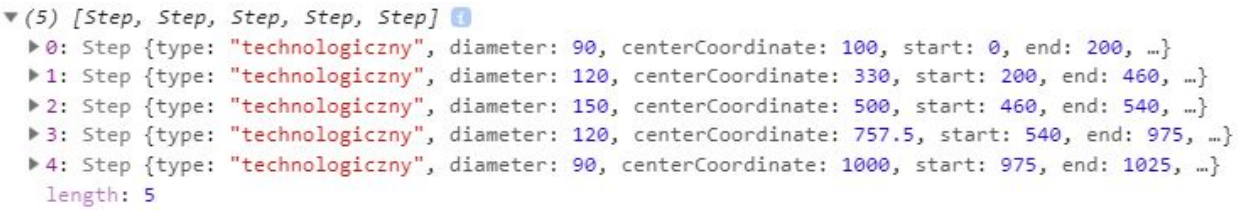
Jeśli stopień jest obciążony, funkcja sprawdza, ile obciążeń zostało przyłożonych oraz jak są rozlokowane na stopniu. Na podstawie informacji o ich położeniu generowane są węzły a później macierze sztywności elementów belkowych. Na końcu, dzięki agregacji tych macierzy, tworzona jest macierz sztywności danego stopnia.

Funkcja sprawdza obciążenie jednocześnie dwie płaszczyzny działania obciążenia. Węzły pochodzą od sił działających w obu płaszczyznach. Pozwala to na stworzenie jednej macierzy sztywności, która może być użyta w obliczeniach ugięć dwukrotnie (dla płaszczyzny XZ i XY). W wersji przed refaktoryzacją, program generował unikalną macierz dla każdej płaszczyzny, co skutkowało zauważalnie dłuższymi czasami obliczeń.

Równolegle generowane są wektory obciążeń dla każdej płaszczyzny. Dla każdego węzła następuje konkatenacja istniejącego już wektora obciążenia i dwuelementowej tablicy reprezentującej obciążenie węzła.

W przypadku, gdy użytkownik zada wielkość elementu skończonego, do tablicy agregującej wszystkie obciążenia dodawane są siły skupione, o wartości 0 N, z krokiem o zadanej wcześniej długości.Sprawdzane jest, czy współrzędna zerowej siły nie będzie pokrywała się ze współrzędną już istniejącej. Dzięki temu, wszystkie stopnie traktowane są, jakby były obciążone, a elementy generowane są “gęściej”.

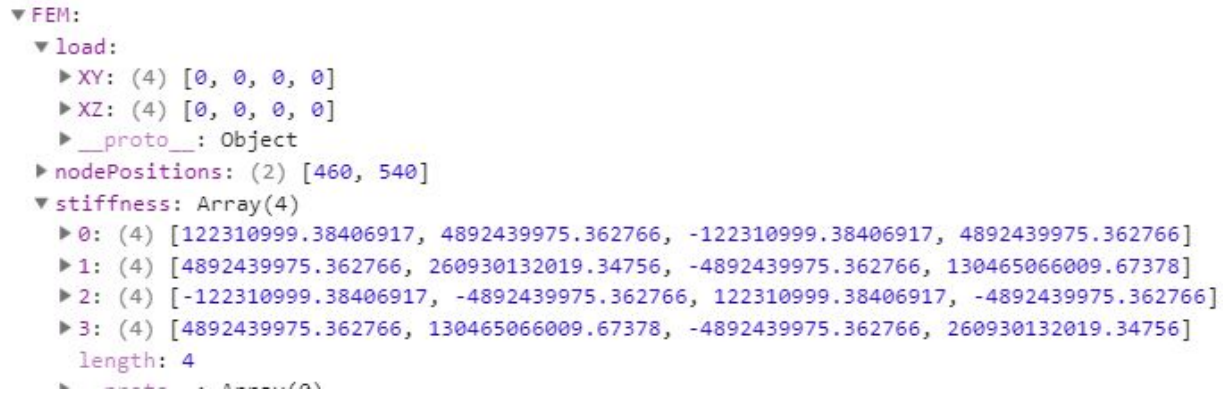
Dane wejściowe - Tablica z obiektami zawierającymi dane definiujące geometrie i położenie stopni (przykład dla wału o 5 stopniach)



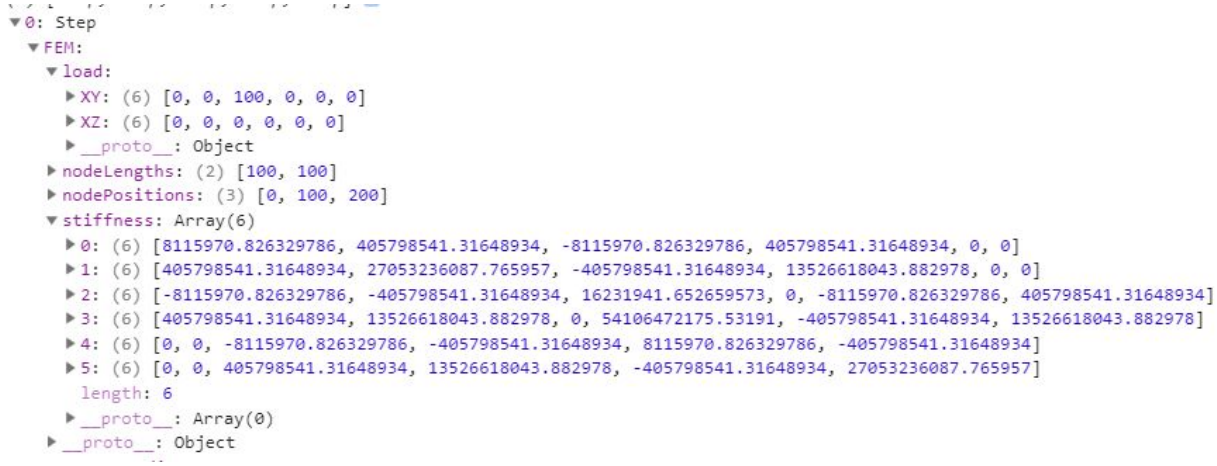
oraz tablice zawierająca obiekty klasy Load, przechowujące dane o obciążeniach ze wszystkich płaszczyzn.

Dane wyjściowe - Macierz sztywności i wektory z wartościami obciążeń dla każdego stopnia

Stopień nieobciążony:



Stopień obciążony (pojedyncza siła o wartości 100 N,w płaszczyźnie XY)**:**

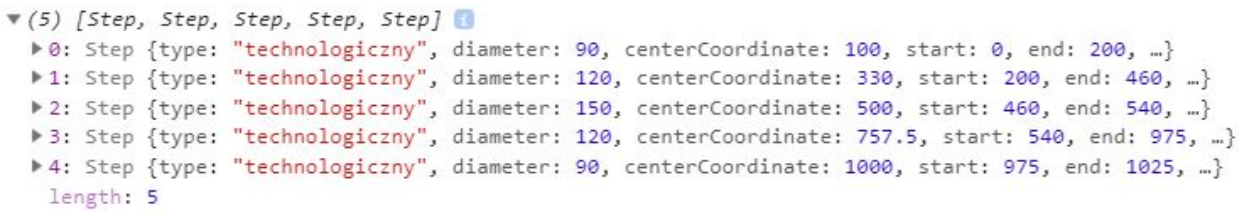


2) Budowa globalnej macierzy sztywności oraz globalnych wektorów obciążeń

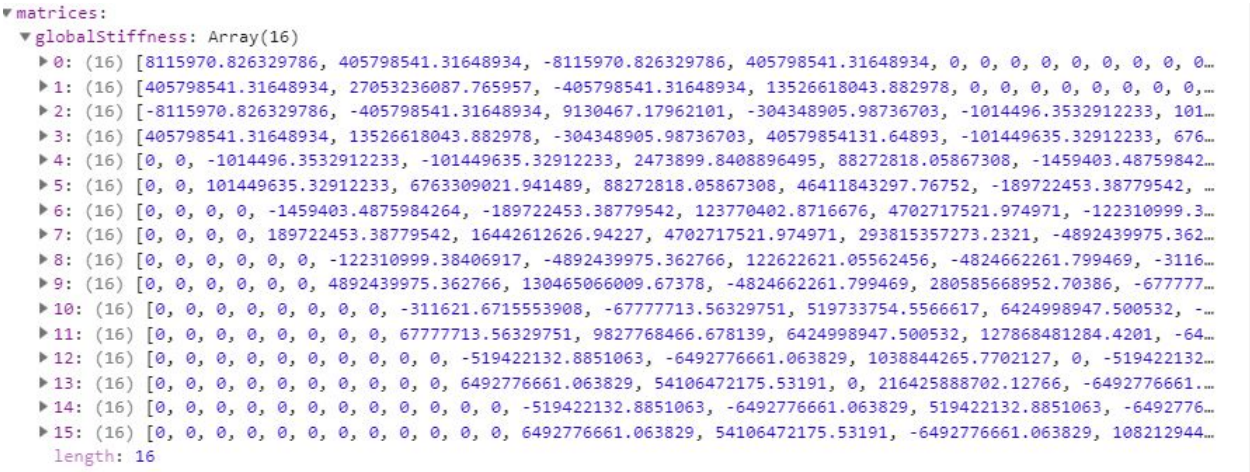
Lokalne macierze i wektory poszczególnych stopni, są dodawane za pomocą odpowiednich dla

nich funkcji agregujących.

Dane wejściowe - Kolekcja obiektów zawierających lokalne macierze sztywności oraz lokalne wektory obciążeń - te obiekty są zagnieżdżonymi właściwościami klasy `Step` (podwłaściwości obiektu `FEM`)



Dane wyjściowe - Globalna macierz sztywności i wektory z wartościami obciążeń (w przykładzie)



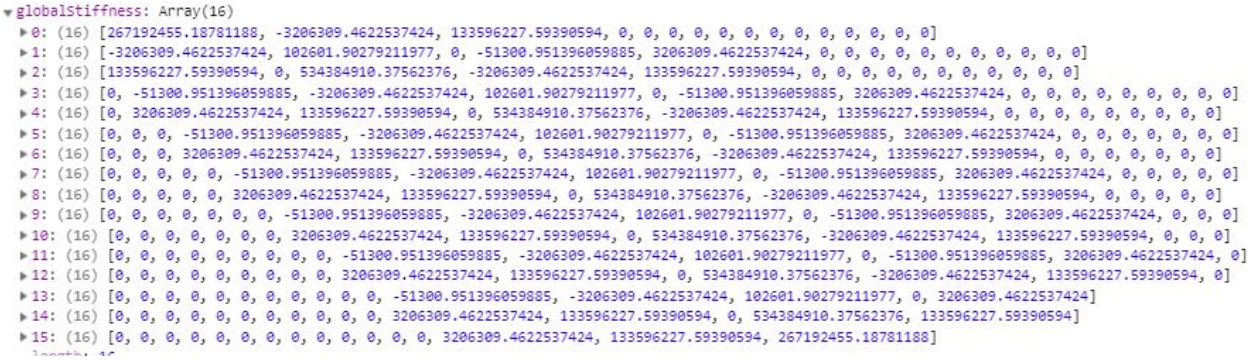
3) Preprocessing

Z macierzy sztywności usunięte zostają kolumny i wiersze odpowiadające węzłom podpór. W wektorach obciążeń usuwane są siły pochodzące od obciążeń. Zapisane zostaje również ich położenie.

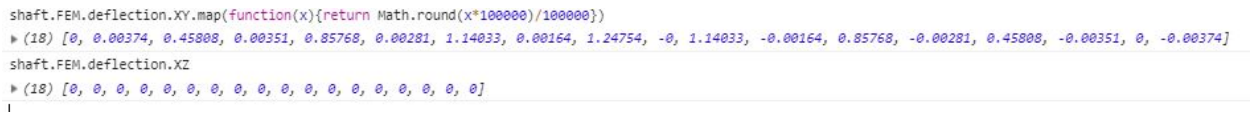
4) Odwrócenie i mnożenie macierzy

Funkcja wyznacza odwrotną macierz sztywności oraz mnoży ją przez wektory sił. W wyniku powstają macierze przemieszczeń węzłowych. Metodę odwracającą i mnożącą macierze dostarcza Math.js

Dane wejściowe - Globalna macierz sztywności i wektory z wartościami obciążeń z usuniętymi węzłami podpór



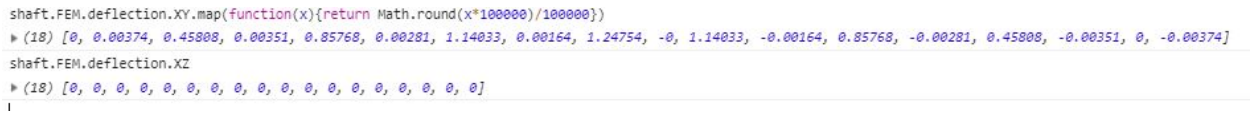
Dane wyjściowe - Macierz przemieszczeń()



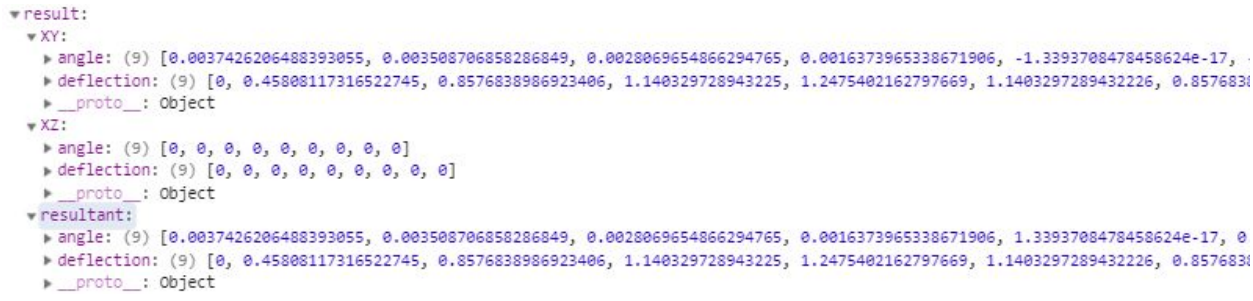
5) Postprocessing

W macierzy przemieszczeń zostają dodane węzły podpór, dla których przemieszczenie wynosi 0 mm. Następnie macierze dzielone są na przemieszczenia kątowe i postępowe. Obliczane są również przemieszczenia wypadkowe. Na końcu, na podstawie długości węzłów, tworzone są wartości kolejnych argumentów (oś x). Tak przygotowane dane są gotowe do wyświetlenia przez Plot.ly

Dane wejściowe - Macierze przemieszczeń



Dane wyjściowe - Obiekt, zawierający dane o położeniach węzłów i odpowiadające im przemieszczenia kątowe i postępowe dla obu płaszczyzn, oraz przemieszczenia wypadkowe.

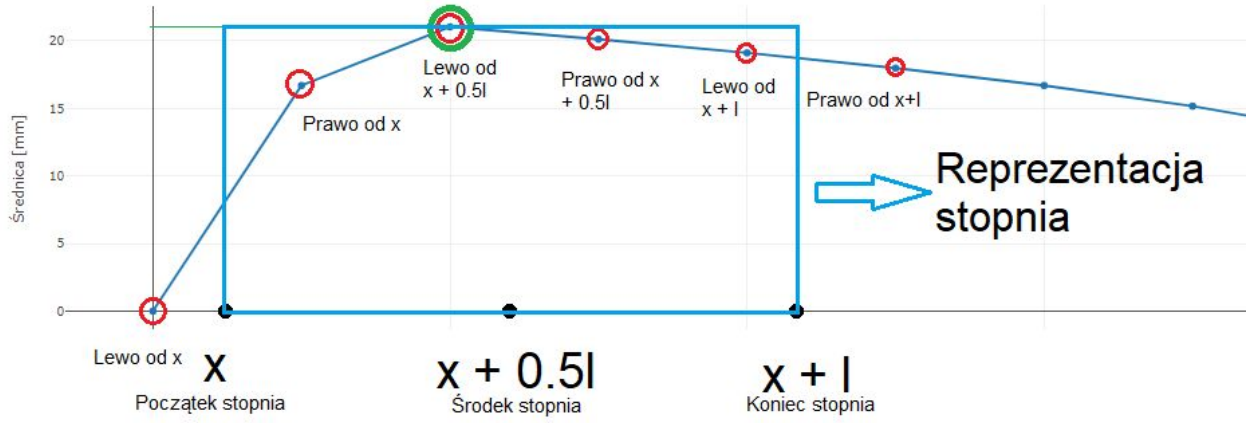


13. Wybrane pozostałe funkcje

13.1. Sugerowana średnica stopnia

Na podstawie wyznaczonego zarysu teoretycznego, funkcja dobiera minimalną średnicę danego przekroju. Idea jest prosta i przypomina odczytywanie z dyskretnego wykresu największej wartości funkcji w zadanym przedziale. Po wczytaniu, algorytm znajduje największą wartość, zaokrągla ją do jedności. Następnie zostaje ona wyświetlona odpowiednim polu.

Na przykładzie z rysunku 13.1, na czerwono zostały zaznaczone wartości pobierane przez funkcję. Na zielono podświetlony jest punkt, którego wartość średnicy (z zarysu teoretycznego) jest największa. Ta wartość po zaokrągleniu zostanie zasugerowana użytkownikowi. Na rysunku, niebieskim prostokątem zaznaczono reprezentacje stopnia, spełniającego wymagania wytrzymałościowe w tym punkcie wału.



Rys. 13.1. Graficzna reprezentacja zasady działania funkcji sugerującej średnicę

13.2. Realizacja kodu forEach();

```
Array.prototype.forEach = function(func(item,index){<<ciało funkcji>>}){
      for(let i = 0; i<this.length;i++){
          func(this [i],i);
      };
};
```

Pętla ta stanowi klasyczny przykład programowania w paradygmacie funkcyjnym.

14. Zapis, odczyt oraz eksport raportu

Domyślnym sposobem zapisu i odczytu wcześniej przygotowanego projektu jest wykorzystanie notacji JSON. Poniżej przedstawiono przykładowy sposób zapisu danych wejściowych (zginanie trzy-punktowe):

{"FEM":{},"general":{"name":"brak danych","person":"brak danych","c":"1000","T":"brak danych","k":"brak danych","omega":"brak danych","material":"brak danych","E":210000,"zgo":1,"zsj":1},"load":{"XY":[{"typ":"sila","odleglosc":500,"plane":"XY","wartosc":500}],"XZ": [],"X": [],"plane": []},"shape":[{"type":"technologiczny","diameter":40,"centerCoordinate":500,"start":0,"end":1000,"length":1000,"FEM":{},"momentOfInertia":{"bending":125663.70614359173,"torsion":251327.41228718346}}]}

W pierwszej wersji programu eksport danych (po przeprowadzeniu obliczeń) polegał na ręcznym skopiowaniu ciągu znaków JSON z odpowiedniego pole tekstowego i zapisaniu go w pliku - z kolei import na wklejeniu tego ciągu znaków z pliku tekstowego w to samo pole.

W nowej wersji użytkownikom udostępniono możliwość eksportu plików.

15. Refaktoryzacja kodu

Metodyka tworzenia i projektowania aplikacji ma charakter ewolucyjny, polega na wielokrotnym tworzeniu i ulepszaniu fragmentów kodu. Wynika to z naturalnego faktu, że w zasadzie na początku programista szuka najbardziej oczywistego, działającego rozwiązania, przymykając oko na jego wady lub po prostu ich nie zauważając. W kolejnych iteracjach kod jest ulepszany. Refaktoryzacja nie ominęła również Shaft.js. Poza elementami wymienionymi wyżej:

1) Zupełnie nowy wzorzec projektowy realizujący obliczenia wytrzymałościowe

Przed refaktoryzacja: Obliczenia wytrzymałościowe realizowane były przez funkcje, których struktura inwokacji przypominała piramidę. Kolejne funkcje wyższego poziomu obliczeń, przyjmowały w parametrach wartości zwracane przez funkcje niższego poziomu. Przykład:

Oblicz reakcję podpór > Wyświetl
Oblicz reakcje podpór > Oblicz równania momentu gnącego > Oblicz przebiegi momentu gnącego > Wyświetl

Realizacja w kodzie:

```
displayReactions(reactionForces("XY"));
//Obliczenie i wyświetlenie reakcji w płaszczyźnie XY

displayPlot(
    calculateTorquePlot(
        createTorqueEquations(
             reactionForces("XY")
        )
     )
);

//Obliczenie i wyświetlenie przebiegu momentu gnącego w płaszczyźnie XY
```

Taka struktura programu byłaby skuteczna w momencie, gdyby dostępne zasoby pamięci byłyby wyjątkowo niskie a moc obliczeniowa w nadmiarze. JS to język interpretowany, co wpływa niekorzystnie na jego wydajność. Co więcej, taki kod nie cechuję się czytelnością oraz trudno poszukiwać w nim błędów.

Po refaktoryzacji: Wszystkie wyniki obliczeń zapisywane są w globalnym obiekcie, mogą być odczytane przez kolejno inwokowane funkcje. Obiekt jest w każdej chwili dostępny z poziomu konsoli przeglądarki internetowej. Wielokrotnie zmniejszyła się też liczba wywołań funkcji.

2) Zadawanie wielkości elementu skończonego

Występowanie w tablicy kilku obciążeń o tej samej współrzędnej sprawia, że program uwzględnia w obliczeniach tylko jedno z nich. Dzieje się tak ze względu na zasadę działania funkcji obliczeń sztywnościowych i wytrzymałościowych.

Może się zdarzyć, że użytkownik zada obciążenie w tym samym punkcie, w którym zostanie dodana zerowa siła generująca elementy skończone. Aby uniknąć nieprzewidywalnego zachowania, wymagana jest weryfikacja wejściowej tablicy obciążenia przed przeprowadzeniem obliczeń sztywnościowych, polegająca na usunięciu duplikatów lub niedopuszczeniu do ich powstania.

Przed refaktoryzacją: Funkcja najpierw dodaje do tablicy zerowe obciążenia w pętli, następnie w drugiej pętli sprawdza, czy istnieją duplikaty (siły zerowe i siły zadawane przez użytkownika o tych samych współrzędnych)
Po refaktoryzacji: Sprawdzanie duplikatów odbywa się w pierwszej pętli, podczas dodawania sił, innymi słowy, jeśli funkcja wykryje zadaną przez użytkownika siłę w tej samej współrzędnej, nie doda zerowej.

3) Pobieranie szerokości okna

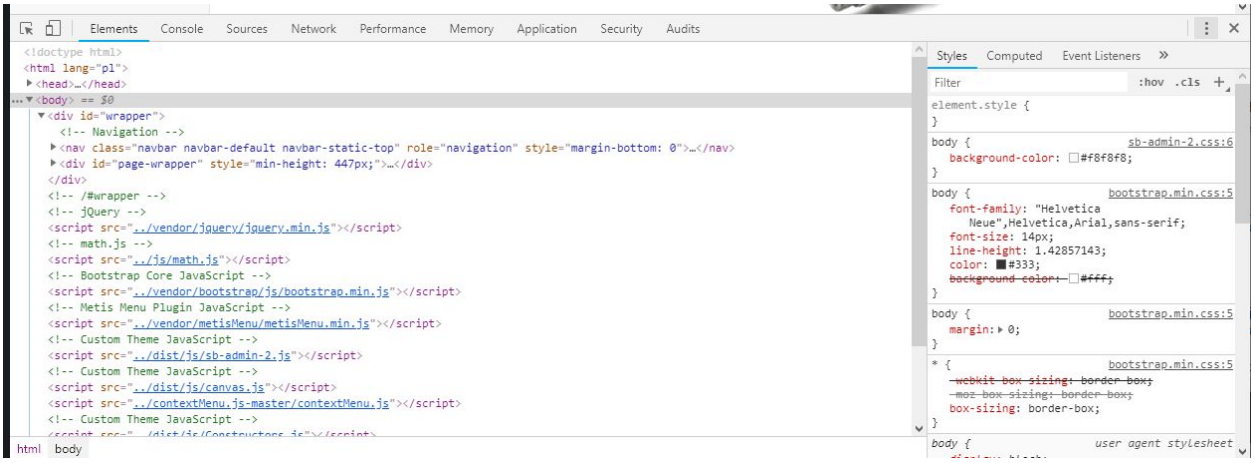
Przed refaktoryzacją: Funkcja za pomocą pętli sprawdza szerokość każdego nagłówka i za pomocą algorytmu wybiera wartość maksymalną
Po refaktoryzacji: Ponieważ szerokość niewidocznego elementu zawsze wynosi 0, funkcja zwraca sumę szerokości wszystkich potencjalnie widocznych nagłówków. Zmniejsza to wykonywaną ilość operacji.

16. Metodyka i automatyzacja testów

Weryfikacja i walidacja tworzonego kodu wymaga stosunkowo dużo czasu. Testy powinno się przeprowadzać nawet przy najmniejszej zmianie w kodzie. Dążąc do maksymalnej możliwej oszczędności czasu, należało stworzyć lub wykorzystać już istniejące narzędzia przyspieszające ten proces. Dużą część czasu poświęconego na testy zajmowało zdefiniowanie danych o obciążeniu i stopniach wału. W związku z tym została stworzona funkcje automatycznie wprowadzające te informacje. Tester ma do wyboru dwie możliwości, w zależności od badanych właściwości programu:

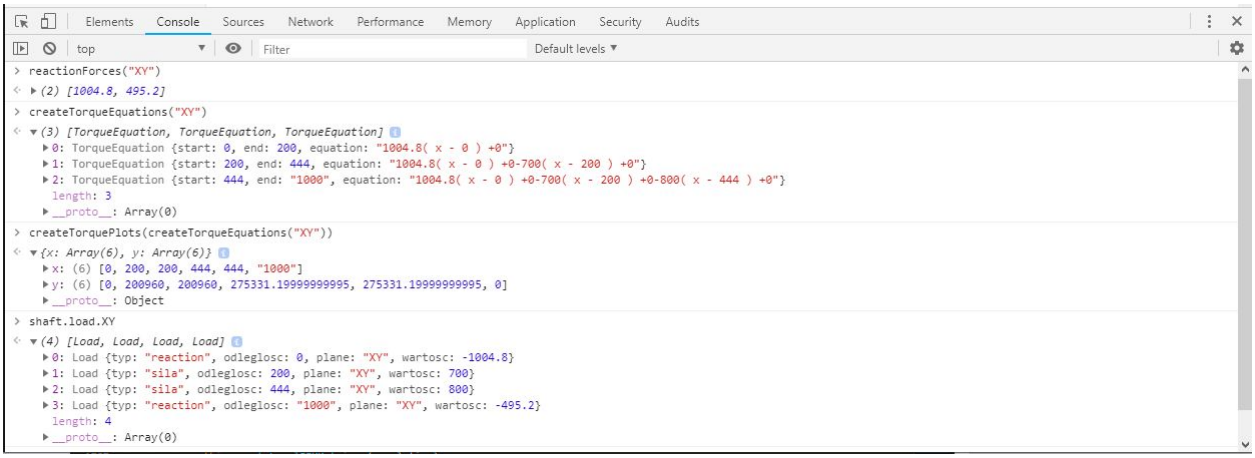
1) Automatyczne wpisywanie w pola tekstowe odpowiednich wcześniej zdefiniowanych wartości i automatyczne klikanie przycisków
2) Bezpośrednie dodanie do bazy, danych o obciążeniach i inwokacja funkcji odpowiedzialnych przypisanych do przycisków

16.1. Konsola deweloperska



Rys. 16.1. Widok struktury HTML w konsoli deweloperskiej

Dzięki temu narzędziu, możemy w dowolnym momencie działania programu wywołać dowolną funkcję, podejrzeć aktualnie znajdujące się wartości w zmiennych, oraz wprowadzić tzw *break-pointy* - miejsca, linie kodu, w którym wykonywanie programu jest zatrzymywane. Konsola umożliwia również wskazanie dowolnego elementu HTML w kodzie strony, po wskazaniu go myszką na wygenerowanej stronie. Monitoruje również zużycie mocy obliczeniowej oraz wykorzystanie pamięci. Można dzięki niej również symulować uruchomienie strony na urządzeniu mobilnym.



Rys. 16.2. Wizualizacja struktury danych JSON w konsoli deweloperskiej

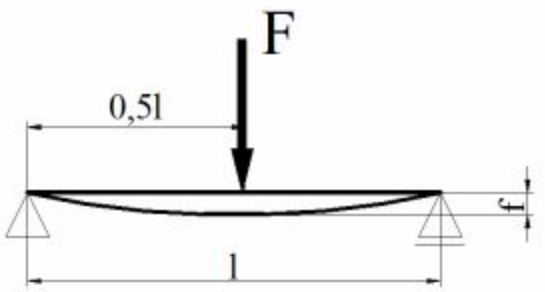
Dzięki temu, programista nie musi przypisywać wywołania funkcji do zdarzeń, w celu weryfikacji ich poprawnego działania oraz na bieżąco śledzić zmiany i przepływy danych. Korzystając z tych zasobów, sprawdzenie poprawnego działania programu jest proste i skuteczne.

17. Weryfikacja przykładów

Shaft.js jest programem uniwersalnym. Jego możliwości nie kończą się na projektowaniu wałów. Możliwe jest rozwiązanie dowolnego zadania belkowego, pod warunkiem, że jest statycznie wyznaczalne oraz nie występuje w nim obciążenie ciągłe. W celu porównania wyników obliczeń aplikacji oraz wyników obliczeń analitycznych, został zbadany następujący przykład.

Zginanie trzy punktowe belki o przekroju kołowym

Zadanie - znaleźć minmalną średnice belki o przekroju kołowym, pozwalającym na przeniesienie zadanego obciążenia, oraz obliczyć ugięcie dla dobranej średnicy w punkcie przyłożenia siły.
Dane:
F = 500N
L = 1m
σ_{dop}= 200 MPa



Rys. 17.1. Schemat obciążenia w badanym przypadku (http://mechanik.edu.pl/learning)

Obliczenia analityczne

Do obliczeń został wykorzystany program SMath, darmowy odpowiednik programu MathCad.

C

Zginanie belki o przekroju kołowym: obliczenia analityczne

Dane:

Siła $F := 500\ \mathrm{N}$

Długość belki $l := 1000\ \mathrm{mm}$

Moduł Younga $E := 210000\ \mathrm{MPa}$

Naprężenia dopuszczalne: $\sigma := 200\ \mathrm{MPa}$

Obliczenie reakcji podpór: przypadek symetryczny więc:

$$R_1 := \frac{F}{2}$$

$$R_2 := R_1$$

Równania momentu gnącego mają postać:

$0 < x < l/2$	$l/2 < x < l$
$Mg_1(x) := R_1 \cdot x$	$Mg_2(x) := R_1 \cdot x - F \cdot \left(x - \frac{l}{2}\right)$
$Mg_1(0) = 0$	$Mg_2\left(\frac{l}{2}\right) = 125\ \mathrm{N\,m}$
$Mg_1\left(\frac{l}{2}\right) = 125\ \mathrm{N\,m}$	$Mg_2(l) = 0$

Obliczenie minimalnej średnicy przekroju:

$$d := \left(32 \cdot \frac{Mg_1\left(\frac{l}{2}\right)}{\pi \cdot \sigma} \right)^{\frac{1}{3}} = 18.5336\ \text{mm}$$

Przyjmuję średnicę

$d := 20\ \text{mm}$

Moment bezwładności przekroju względem osi bezwładności:

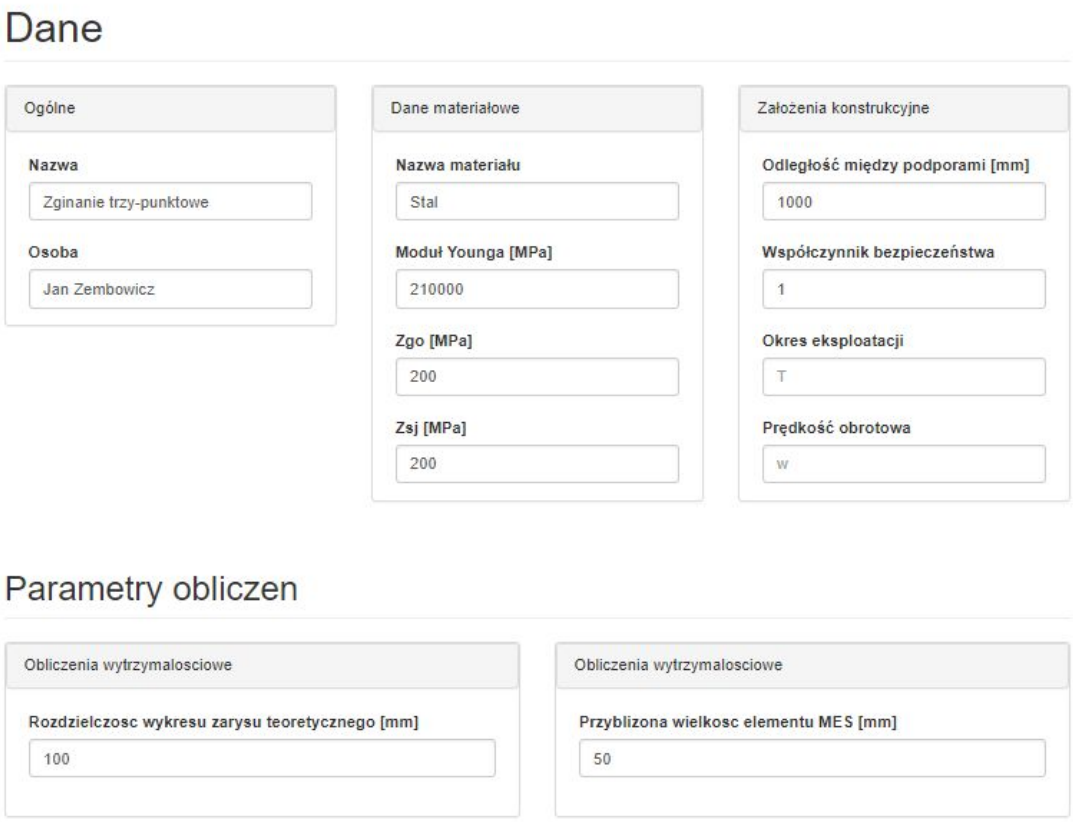
$$I := \frac{\pi \cdot d^4}{64} = 7853.9816\ \text{mm}^4$$

Ugięcie belki w miejscu przyłożonej siły

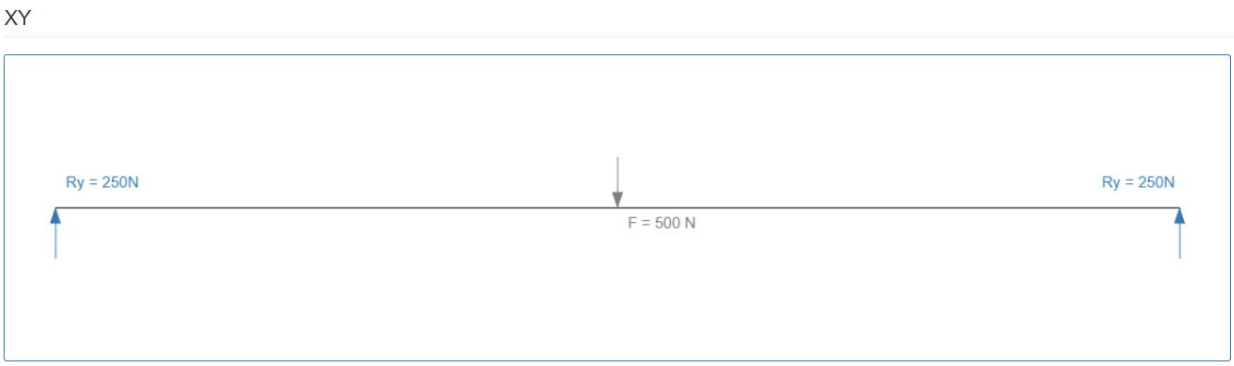
$$f := F \cdot \frac{l^3}{48 \cdot E \cdot I} = 6.3157\ \text{mm}$$

Źródło wzoru na ugięcie:
(http://mechanik.edu.pl/learning/projektowanie_z_ppcm/wzory_strzalek_ugiecia_belek.pdf)

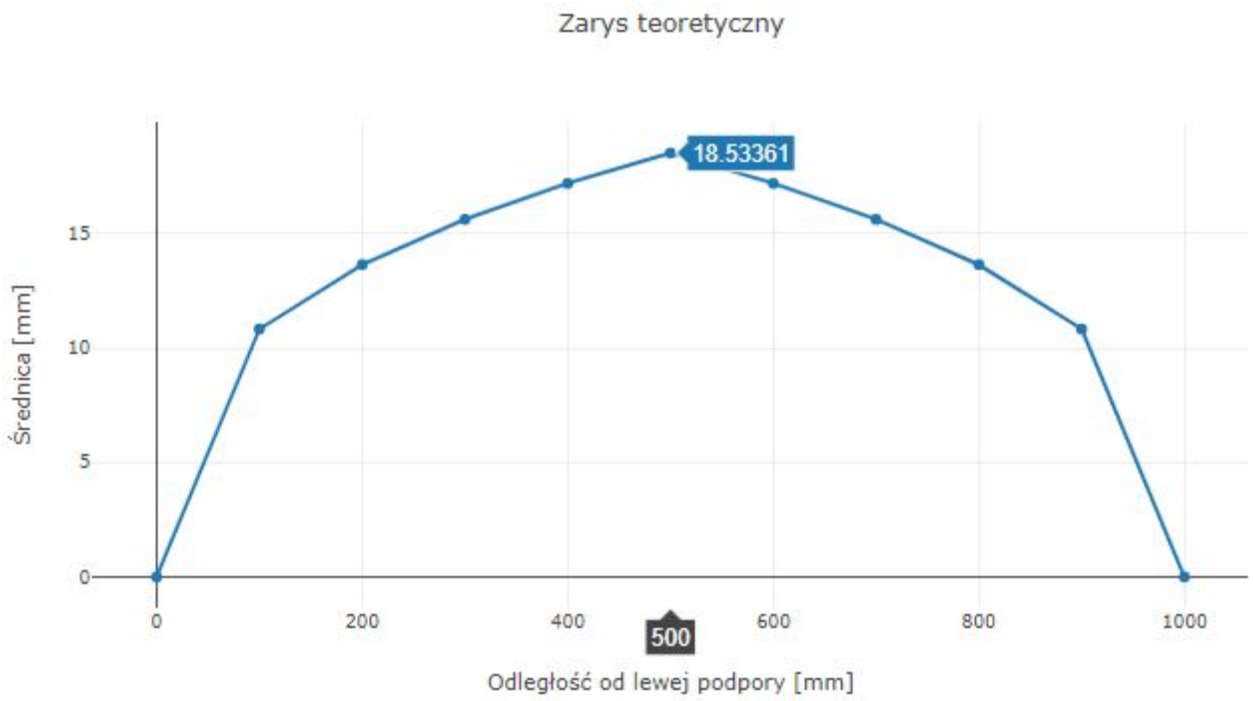
Obliczenia przeprowadzone w shaft.js



Rys. 17.2. *Parametry obliczeń*

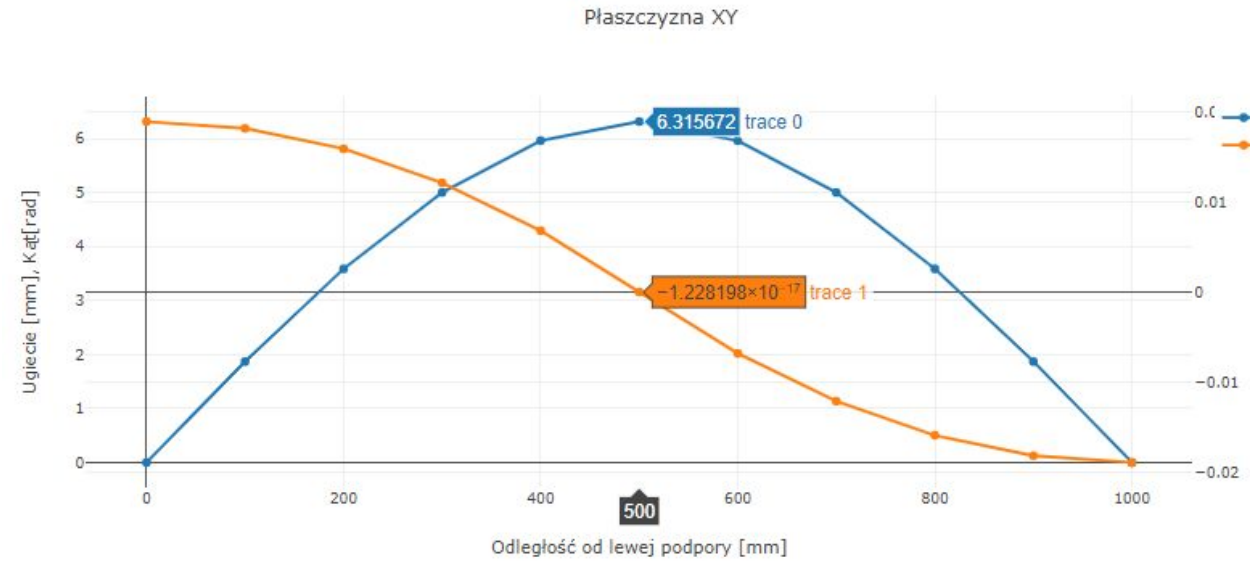


Rys. 17.3. Zamodelowane obciążenie i wyznaczone reakcje podpór



Rys. 17.4. Zarys teoretyczny badanego przypadku

Ugięcie w mm wyznaczone w miejscu przyłożenia siły (rys 17.4). Błąd wartości kątowych wynika z przybliżeń i komputerowego sposobu zapisu liczb zmiennoprzecinkowych.



Rys. 17.5. Wykres ugięć i kątów ugięcia badanego przypadku

18. Podsumowanie, wnioski i uwagi końcowe

Wyniki obliczeń analitycznych oraz tych przeprowadzonych przez shaft.js są bardzo zbliżone. Dzieje się tak dzięki zastosowaniu w programie metod obliczeniowych, które są implementacją klasycznych metod analitycznych. Wynikowo, program jest dobrym narzędziem do weryfikacji zadań projektowych.

Zastosowana architektura obiektowa, jest znacznie szybsza od architektury funkcyjnej, która była zaimplementowana w poprzednich ewolucjach programu. Znajduje to bezpośredni pozytywny wpływ na komfort i odczucia z użytkowania aplikacji.

JavaScript to technologia, która standardowo nie jest stosowana w programach CAD/CAE. W każdym razie, dynamiczne typowanie, możliwość tworzenia obiektów i funkcji ,,w locie'', pozwala na szybkie stworzenie programu o zaawansowanej i wielofunkcyjnej strukturze, co zostało wykorzystane przy tworzeniu aplikacji Shaft.js.

Na obecnym etapie rozwoju programu nie przewidziano zakładki umożliwiającej dobór łożysk. Jednak ponieważ jest to aplikacja webowa, możliwe jest zaimplementowanie modułu komunikującego się z dowolnym internetowym katalogiem łożysk, z którego będą pobierane dane o częściach znormalizowanych.

Mając na uwadze charakter rozwojowy programu, rozważane jest przeniesienie części obliczeń na stronę serwera i wykorzystanie żądań asynchronicznych AJAX. Pozwoli to na analizę bardzo złożonych przypadków w czasie rzeczywistym. Wymagane jest jednak szybkie łącze internetowe.

19. Bibliografia i źródła internetowe

1. Orłowski C., Lipski J., Loska A. (2012): Informatyka i komputerowe wspomaganie prac inżynierskich, Polskie Wydawnictwo Ekonomiczne SA, Warszawa.
2. Kurmaz L., Kurmaz O. (2011): Projektowanie węzłów i części maszyn, wyd. Politechnika Świętokrzyska, Kielce.
3. Klekot G. (2019): Podstawy konstrukcji maszyn - wykładowe materiały pomocnicze str. 240-288, Instytut podstaw budowy maszyn, Wydział samochodów i maszyn roboczych, Politechnika Warszawska, Warszawa.

4. Łodygowski T., Kąkol W. (1991): Metoda elementów skończonych w wybranych zagadnieniach mechaniki konstrukcji inżynierskich, rozd. 5. [online], Instytut konstrukcji budowlanych. Politechnika Poznańska, Poznań.
5. Pietrzakowski M. (2011): Wytrzymałość materiałów, rozd. 6., 8., 12. [online], Instytut podstaw budowy maszyn, Wydział samochodów i maszyn roboczych, Politechnika Warszawska, Warszawa.
6. Freeman E., Freeman E. (2011): Head First HTML5 Programming: Building Web Apps with JavaScript, wyd. O'Reilly Media, Sebastopol, Kalifornia, USA.
7. Goto K. ,Van De Geijn. R. (2008): Anatomy of high-performance matrix multiplication, The university of Texas at Austin, Austin, Texas, USA.

8. World Wide Web Consorcium, https://www.w3schools.com/ [dostęp: 05.04.2019].
9. Alicea T. (2018): Understanding weird parts of JavaScript, https://www.udemy.com [dostęp: 04.11.2018].
10. Dokumentacja biblioteki Math.js, http://mathjs.org/ [dostęp: 06.03.2019].
11. Dokumentacja biblioteki Bootstrap 4, http://getbootstrap.com/docs/4.1/ [dostęp 06.03.2019].
12. Dokumentacja biblioteki Plot.ly, https://plot.ly/javascript/ [dostęp 06.03.2019].
13. Brown B. (2005): Postfix notation mini lecture [online]. http://ksuweb.kennesaw.edu/faculty/rbrow211/web_lectures/postfix/ Information Technology Department, College of Computing and Software Engineering, Kennesaw State University, Kennesaw, Georgia, USA [dostęp 02.10.2018].
14. Repozytorium projektu (kod źródłowy): https://github.com/JWZ1996/shaft.js [dostęp 21.04.2019].

20. Spis Tabel i rysunków