İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

MATRİS DEPLASMAN YÖNTEMİYLE İKİ BOYUTLU GEMİ ENİNE ÇERÇEVELERİNİN YAPISAL ANALİZİNİ YAPAN BİR BİLGİSAYAR PROGRAMI HAZIRLANMASI

YÜKSEK LİSANS TEZİ

Ömer BİRLER

Gemi İnşaatı ve Gemi Makinaları Mühendisliği Ana Bilim Dalı

Gemi İnşaatı ve Gemi Makinaları Mühendisliği Programı

ARALIK 2018

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

MATRİS DEPLASMAN YÖNTEMİYLE İKİ BOYUTLU GEMİ ENİNE ÇERÇEVELERİNİN YAPISAL ANALİZİNİ YAPAN BİR BİLGİSAYAR PROGRAMI HAZIRLANMASI

YÜKSEK LİSANS TEZİ

Ömer BİRLER
(508161025)

Gemi İnşaatı ve Gemi Makinaları Mühendisliği Ana Bilim Dalı

Gemi İnşaatı ve Gemi Makinaları Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Ertekin BAYRAKTARKATAL

ARALIK 2018

İTÜ, Fen Bilimleri Enstitüsü'nün 508161025 numaralı Yüksek Lisans Öğrencisi Ömer BİRLER, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "MATRİS DEPLASMAN YÖNTEMİYLE İKİ BOYUTLU GEMİ ENİNE ÇERÇEVELERİNİN YAPISAL ANALİZİNİ YAPAN BİR BİLGİSAYAR PROGRAMI HAZIRLANMASI" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı : **Doç. Dr. Ertekin BAYRAKTARKATAL** .....
İstanbul Teknik Üniversitesi

Jüri Üyeleri : **Prof. Dr. Ahmet ERGİN** .....
İstanbul Teknik Üniversitesi

Prof. Dr. Mehmet Ömer BELİK .....
Piri Reis Üniversitesi

Teslim Tarihi : 16 Kasım 2018
Savunma Tarihi : 14 Aralık 2018

Kardeşlerim ve aileme,

ÖNSÖZ

Tez çalışmam boyunca bana her daim yardımcı olan, beni cesaretlendiren ve bana çok şey katan değerli hocam Doç. Dr. Ertekin BAYRAKTARKATAL'a ve bana programlamayı sevdiren ve bu bitirme çalışmasında bana yol gösterip yardımını esirgemeyen çok değerli arkadaşım Yusuf TEMİZ'e sonsuz teşekkür ederim.

Kasım 2018

Ömer BİRLER
Gemi İnşaatı ve Gemi Makinaları Mühendisi

İÇİNDEKİLER

Sayfa

KISALTMALAR

C# **:** C Sharp Programlama Dili

WPF : Windows Presentation Foundation

XAML **:** Extensible Application Markup Language

SEMBOLLER

$\boldsymbol{w(x)}$ **:** Yayılı Yük Dağılımı

L **:** Kirişi Uzunluğu

$\boldsymbol{M_A}$ **:** Kirişin Solundaki Düğüm Momenti

$\boldsymbol{M_B}$ **:** Kirişin Sağındaki Düğüm Momenti

$\boldsymbol{M_0(x)}$ **:** Kirişin İzostatik Halindeki Moment Dağılımı

$\boldsymbol{F_0(x)}$ **:** Kirişin İzostatik Halindeki Kuvvet Dağılımı

E **:** Elastisite Modülü

I **:** Atalet Momenti

q(x) **:** Yayılı Yük Dağılım Fonksiyonu

σ **:** Gerilme

$\boldsymbol{e}$ **:** Asal eksen mesafesi

$\boldsymbol{d}$ **:** Kirişin En Kesit Yüksekliği

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

MATRİS DEPLASMAN YÖNTEMİYLE İKİ BOYUTLU GEMİ ENİNE ÇERÇEVELERİNİN YAPISAL ANALİZİNİ YAPAN BİR BİLGİSAYAR PROGRAMI HAZIRLANMASI

ÖZET

Bu çalışmada gemi enine çerçevelerinin yapısal analizini yapabilecek bir bilgisayar programı geliştirilmesi amaçlanmıştır. Gemilerin ön dizaynında yapı hakkında fazla ayrıntı gerektirmeden çözüm yapabilen pratik analiz programlarına ihtiyaç duyulur. Gemi çerçeveleri değişken kesitli olabilmektedir. Bu yüzden programın değişken kesitli çerçeveleri de çözebilmesi istenmiştir. Kullanımının kolay olması, görsel arayüzünün olması, yazılan kodun sadeliğini ve okunabilirliğini arttırmak için nesne yönelimli programlama yaklaşımlarını uygulaması, gelişen teknolojiyle beraber artan bilgisayar kapasitelerini yüksek oranda ve verimli kullanabilmesi için modern uygulama programlama arayüzlerini kullanması hedeflenmiştir.

Gemilerin ön dizaynında karşılaşılan çerçeve yapılarının yapısal analizinde Matris Deplasman Metodu en çok kullanılan yöntemlerden birisidir. Metot bir çeşit sonlu elemanlar metodudur. Direkt metotlar arasında yer alır. Bazı sonlu elemanlar yöntemlerindeki gibi şekil fonksiyonlarına ihtiyaç duymaz. Katılık matrisi direkt olarak kiriş teorisinden çıkartılır.

Matris deplasman metodu kiriş elemanının bütün serbestlik dereceleri doğrultusundaki deplasmanlar sıfır iken her bir serbestlik derecesi doğrultusunda birim deplasman oluşturacak kuvvetleri elde ederek katılık matrisi oluşturur. Matris olarak ifade edilmiş eleman rijitlik ilişkileri daha sonra kod numaraları yöntemiyle çerçeve sistemini temsil eden tek müşterek matris haline getirilir. Aynı şekilde elemanlara etkiyen kuvvetlerden de müşterek bir kuvvet vektörü elde edilir. Elde edilen matris sistemi çözülerek deplasman vektörü elde edilir. Daha sonra bu deplasman vektörünün elemanları yine kod numaraları yöntemiyle elemanlara aktarılır ve eleman kuvvet dengesi yazılarak elemanın katılık matrisi deplasman vektörüyle çarpılıp endirekt yük vektörüyle toplanır. Bunun sonucunda müşterek eksenlerde ifade edilmiş eleman yük vektörü elde edilir. Elde edilen kuvvet vektörü transformasyon matrisiyle çarpılarak elemanın kendi eksenlerindeki kuvvet vektörü bulunur. Bu kuvvet vektörü elemanın mesnet tepkilerini içerir. Bu vektör elde edildikten sonra elemanın kesit tesir diyagramları bulunabilir.

Bu çalışmada geliştirilmiş olan program hedeflendiği gibi değişken kesitli enine çerçevelerinin çözümünü yapabilmektedir. Değişken kesitli durumda iken elemanın uç momentleri Üç Moment Yönteminin en genel haliyle bulunur. Program kullanıcıdan eklenen kiriş parçalarının her birinin atalet ve kesit alanı dağılımlarını polinom şeklinde ister. Kullanıcı kirişlere yayılı yükleri ve tekil yükleri arayüzden ekler. Program yayılı yüklerin de polinom şeklinde ifade edilmesini bekler. Kullanıcı istediği çerçeveyi oluşturduktan sonra çözüm butonuna basar ve program sistemi çözer. Çıktı olarak eğilme momenti dağılımı, kesme kuvveti dağılımı, eksenel kuvvet dağılımı ve gerilme dağılımı verir.

Program nümerik integral hesaplarında Simpson Yöntemini kullanır. Sabit kesitli çerçeve gibi basit durumlarda program, geliştirilen polinom sınıfı sayesinde analitik integraller alabilmekte ve nümerik çözüme gerek kalmadan hızlı bir şekilde hesap yapabilmektedir. Ayrıca program, paralel programlama yöntemlerini kullanarak çok çekirdekli bilgisayarlarda kısa sürede sonuç üretebilmektedir.

Bu çalışmada geliştirilen program analitik çözüm ve muadili programlarla değişik çerçevelerde verdiği sonuçlar bakımından karşılaştırılmış ve sonuçların doğru olduğu anlaşılmıştır. Sonuç olarak hedeflenen program geliştirilmiş ve gemi çerçevelerinde kullanılabilecek pratik, kullanımı kolay ve kullanışlı bir analiz programı elde edilmiştir.

DEVELOPING A COMPUTER PROGRAM TO CONDUCT STRUCTURAL ANALYSIS OF TWO DIMENSIONAL SHIP FRAMES USING MATRIX DISPLACEMENT METHOD

SUMMARY

In this study, it is aimed to develop a computer program to analyze two dimensional ship frames. Preliminary design of ships requires using practical structural analysis programs that do not require detailed information about the structure. Since ship frames can have varying inertia, like when brackets are involved, the program should have the ability to solve ship frames with varying inertia. The program is desired to have a well-designed graphical user interface to make it easy to use, to implement object oriented programming paradigms to keep the code structured and readable, to use a modern programming application interfaces to maximize the utilization of computer resources in efficient way.

In preliminary design phase, it is easy to model the ship as frames when not much detail about the structure of the ship is known. It also makes it easy to solve the problem. In the past, conventional methods like Moment Distribution Method have been used for analyzing ship frames. After computer technologies showed up, scientists started to search for computer-automated methods for structural analysis. Then, Matrix Displacement Method is developed. The method was a preface for Finite Element Method and it is considered one of the direct methods of Finite Element Method.

Matrix Displacement Method discretizes the system into smaller, idealized elements. For each beam element, when all displacements are zero, the forces that are required to create unit displacement for each degree of freedoms are obtained and stored into a matrix, which is called the element stiffness matrix. For each beam element, the forces that are acting on beams are stored in element indirect force vector and the forces that are acting on nodes are stored in element direct force vector. All forces, vectors and stiffness matrices are express in terms of global axes by using transformation matrix. After that, by using Code Numbers Method, the element stiffness matrices are combined in a global stiffness matrix. Likewise, the element force vectors are also combined in global force vector. The global system is then solved to find global displacement vector. After the global solution is done, the displacement vector is imposed in element force equations. The parts of the displacement vector in elements are multiplied by element stiffness matrices and the results are summed with indirect force vector to get the final force vector. Again, by using transformation matrix, the local force vectors are obtained. After the local force vectors are known, bending moment distributions, shear force distributions, axial force distributions and stress distributions can be found.

In case of varying inertia, stiffness matrices depend on the base stiffness coefficients, which can be found by some integrations that are derived by Moment-Area Method. The method that is used to find support reactions is also changed in case of varying

inertia. Clapeyron's Method of Three Moment is used in this case with its the most general form to calculate support reactions when both sides of the beam are fixed end supports.

There are many codes written in the past that implement Matrix Displacement Method. Most of them are written in either Basic or Fortran. These programs generally require a third-party software to analyze the results and draw the distributions. These programs also need entering the data in specific format, which can be hard for users.

The program that was developed in this study is able to solve many bending problems, not just for ships. It also does not need any additional software to present the analysis it made. It uses numerical calculation techniques such as Simpson Method when it has to but it can also solve the problems analytically if the inertia distributions of the beams are constant, which provides extra accuracy and gives the ability to solve the problem with minimum time.

The program was written in C# language. C# is a strongly typed, object oriented and multi-paradigm programming language that was developed by Microsoft in 2002. It can be used in variety of applications and it is much easier to develop programs than lower level languages. The program is based on Windows Presentation Foundation (WPF) platform, which allows the developers to design user interfaces in Extensible Application Programming Language (XAML). By using modern programming language enables the program to use modern libraries and conduct parallel computations when the user wants to.

The program consists of a drawing area and the buttons around it. The user, by clicking the buttons, can add a beam; specify the inertia and the area distribution in polynomials. Since it takes only distributions in polynomials, it can be used for every frames independent of the cross-section shape. The program also accepts distributed load in polynomial form. After the user created the desired frame, he clicks the solve button and the solution initializes. First, for every beam in the frame, support moments are calculated by implementing Three Moment Theorem. Then, the force vector is created, the transformation matrix is formed based on the beam's angle. Base stiffness coefficients are calculated according to the area and the inertia distributions. Then, the local stiffness matrix is created. These processes can be run in parallel. After that, the global stiffness matrix and the global force vector are created. By using a linear equation solver that implements Gauss Elimination Method, the global displacement vector is created. Then, the displacement vector elements sent as parameters in beams, which also start element solutions. The force vectors and the local force vectors are then calculated. After the local force vectors created, moment distributions are obtained based on the end moments in the vectors. Moment distributions derivate analytically to obtain shear force vectors. By using the local force vectors, axial force distributions are also calculated. If the user wanted to perform a stress analysis, the program calculates stress distributions by using moment distributions and inertia distributions. The graphics are then drawn. The program uses Cardinal Spline library to draw the distributions. The program warns the user if maximum allowable stress is exceeded in any beam. The program also gives polynomial expression of moment distributions and shear distributions and axial distributions. It also finds minimum and maximum values of the distributions.

As a result, a practical and easy to use computer program that conducts structural analysis of two dimensional ship frames has been developed. The program meets the

desired specifications explained at the beginning. In the end, different frames, both with varying and constant inertia, have been analyzed using this program and other programs that can calculate the given frame. The results based on end moments have been compared. It is understood that the program gives the correct results.

1. GİRİŞ

Gemilerin ön dizaynında yapısal analizlerde kiriş halinde modelleme sıklıkla yapılır. Ön dizayn sürecinde henüz elde yapıyla ilgili ayrıntılı bir bilgi yok iken bu modelleme kolaylık sağlar. Ayrıca kiriş olarak modellendiğinde problem kolaylaşır. Yapı kirişler halinde modellendiği zaman çerçeve sistemleri ortaya çıkar. Bu hiperstatik sistemlerin el ile çözümüne yönelik Cross Metodu gibi geleneksel çözüm yöntemleri bilgisayar teknolojisinin gelişmesiyle 1950'li yıllardan sonra yerini nümerik yöntemlere bırakmıştır (Bayraktarkatal, 1995). Bu yöntemlerden birisi Matris Deplasman Yöntemidir. Matris Deplasman Yöntemi bir çeşit Sonlu Elemanlar uygulamasıdır. Kiriş elemanlarının şekil fonksiyonları kullanmadan direkt olarak sonlu elemanlar modellemesine izin verir. Bilgisayar programlamasına uygun bir yöntemdir.

Temelde yöntem, kiriş teorilerinden yararlanarak her serbestlik derecesi yönünde birim deplasmanlar oluşturan kuvvetleri elde edip katılık matrisi adı altında toplamaya dayanır. Böylece elemanlar matrisler halinde ifade edilip müşterek matris oluşturulur ve matris bilgisayar tarafından çözülerek bilinmeyen deplasmanlar bulunur (Tezcan, 1970). Yöntemin matris tabanlı olması bilgisayar programlamasına oldukça elverişli kılar.

Bu çalışmanın temel amacı gemi enine çerçevelerinin çözümünü yapabilen bir bilgisayar programı oluşturmaktır. Geçmişte Matris Deplasman Metodunu uygulayan birçok bilgisayar programı yazılmıştır. Bunların çoğu Basic ve Fortran dili kullanılarak geliştirilen konsol programlarıdır. Bu programlar girdileri belirli bir formatta alırlar ve kullanımı kullanıcı açısından kolay değildir. Burada ise modern bir programlama dili kullanılarak görsel grafik arayüzü olan, pratik ve kullanışlı bir paket programı geliştirilmesi amaçlanmıştır. Geliştirilen programın aynı zamanda modern uygulama geliştirme kütüphanelerini kullanması ve günümüz bilgisayar teknolojisinden yararlanarak hızlı ve verimli çözüm yapabilmesi hedeflenmiştir. Diğer bir hedef ise nesne yönelimli programlama yaklaşımlarının kullanılmasıdır. Böylece düzenli ve disiplinli bir kod tabanı oluşturulup herkesin okuyabileceği

sadelikte, bakımı kolay bir proje elde edilecektir. Ayrıca programın başka hiçbir programa ihtiyaç duymadan analiz yapabilmesi de amaçlanmıştır.

Çalışmanın ikincil amacı ise geliştirilmeye açık modüler bir kod tabanı oluşturmaktır. İleride, çözüm metodu örneğin uzay çerçeveler için çözüm yapacak şekilde geliştirilmek istendiğinde mevcut kod kütüphanesinden maksimum seviyede yararlanılması hedeflenmiştir.

Literatürde bu alanda çeşitli çalışmalar yürütülmüştür. Bunlardan bir tanesi Doç. Dr. Ertekin Bayraktarkatal'ın doktora tezi kapsamında 1995 yılında geliştirdiği, gemiyi uzay çerçeve olarak modelleyen Fortran programlarıdır. 3 program gemi kirişleri üzerindeki moment ve gerilme dağılımlarını elde eder. Sabit kesitli gemi kirişleri için çözüm yapmaktadır.

Başka bir çalışma ise Yrd. Doç. Dr. Adnan Karaduman tarafından yüksek lisans tezi kapsamında 1993 yılında geliştirilen Basic programlama dilinde yazılmış bir bilgisayar programıdır. 2 boyutlu enine çerçeveleri çözen bu program bazı braketli durumlarda değişken kesit çözümü yapabilmektedir.

Diğer bir çalışma ise Assoc. Prof. Dr. Luiz Fernando Martha tarafından geliştirilen Ftool adlı 2 boyutlu çerçeve analiz programıdır. Sabit kesitli çerçeveler için geliştirilmiştir. Programın ayrıca ticari bir versiyonu da vardır.

Bir diğer çalışma ise Gemi İnşaat Mühendisi Ömer Birler tarafından 2016 yılında geliştirilmiş olan Mesnet adlı 2 boyutlu çerçeve analiz programıdır. Bu program 2 boyutlu gemi enine çerçevelerin yapısal analizini Cross Metodu kullanarak analiz eder. Program değişken kesitli çerçeveleri de çözebilmektedir.

Bu çalışmanın ikinci bölümünde çözüm yöntemi ayrıntılı olarak anlatılmıştır. Matris Deplasman Metodu ve Üç Moment Metodu açıklanmış, katılık matrisinin ve yük vektörlerinin elde edilmesi gösterilmiştir. Müşterek sistemin oluşturulması ve çözülmesi bir örnekle açıklanmıştır. Değişken kesitli durumda hesaplanması gereken temel rijitlik katsayılarının denklemlerle elde edilmesi gösterilmiştir.

Çalışmanın üçüncü bölümünde geliştirilen programdan bahsedilmiştir. Eklerde verilen kod parçalarına atıfta bulunarak programın çözümü nasıl yapığı anlatılmıştır. Programın çalışması için gereken sistem gereksinimleri açıklanmıştır.

Dördüncü bölümde ise programın arayüzü tanıtılmıştır. Daha sonra programın kullanılışı anlatılmış, basit bir çerçevenin programda oluşturulması, çözülmesi ve sonuçların elde edilmesi gösterilmiştir.

Beşinci bölümde sabit kirişli ve değişken kirişli çerçeve örnekleri verilmiş ve bu programda ve başka programlarda çerçeveler çözülerek sonuçların doğruluğu karşılaştırılmıştır.

Altıncı bölümde ise elde edilen sonuçlar açıklanmış ve program hakkında önerilerden, geliştirmelerden ve eksikliklerden bahsedilmiştir.

2. MATRİS DEPLASMAN YÖNTEMİ

Matris Deplasman Yöntemi ya da diğer bir adıyla Direkt Deplasman Yöntemi bir yapısal analiz metodudur. Bilgisayar programlamasına oldukça uygundur. Statikçe belirsiz karmaşık kiriş problemlerini çözebilecek kabiliyete sahiptir. En çok kullanılan sonlu elemanlar uygulamalarından biridir. Yapısal analizde tek boyutlu ve iki boyutlu kiriş sistemleri analizi için en ideal sonlu elemanlar yöntemidir.

Sonlu elemanlar yöntemi birçok mühendislik alanında kullanılabilen bir sayısal çözüm yöntemidir. Yapısal analiz, ısı geçişi, sıvı akışı elektromanyetizma başta olmak üzere birçok mühendislik alanında uygulanmaktadır. Mühendislik hesaplarında genelde olayın fiziği kısmi diferansiyel denklemler şeklinde matematiksel olarak ifade edilir. Genelde bu diferansiyel denklemlerin analitik çözümü zordur ve bazen imkânsızdır. Ayrıca hesap yapılacak elemanın şeklinin karmaşık olması da hesabı zorlaştırır. Sonlu elemanlar yöntemi, sayısal yöntemleri kullanarak bu çözülmesi zor problemlere bir çözüm sunar. Sonlu elemanlar yöntemi temelde direkt metotlar, ağırlıklı artım metodu, varyasyonel metotlar ve enerji metodu olarak ayrılabilir. Matris deplasman metodu direkt metotlar sınıfına girer.

Direkt metotlarda diğer sonlu elemanlar metotlarında olduğu gibi yaklaşık şekil fonksiyonları kullanılmaz. Bunun yerine problemin fiziği kullanılarak katılık matrisleri elde edilir. Diğer sonlu elemanlar yöntemlerinde olduğu gibi problem ayrıklaştırılarak idealize edilmiş ve nodların birbirine bağlanmasıyla oluşan elemanlar olarak modellenir. Daha sonra elemanların katılık özellikleri tek bir matriste toplanır. Elde edilen müşterek matris çözülerek yapının bilinmeyen deplasmanları ve kuvvetleri bulunur.

Metot, çerçeve ve kafes sistemlerine uygulanabilir. Yaklaşık fonksiyonlar kullanmadığı için sonuç kiriş teorisindeki sonuçla aynı elde edilir. Diğer sonlu elemanlar yöntemlerindeki gibi yakınsama analizine bu yöntemde gerek yoktur.

2.1 Tarihçe

Matris deplasman metoduna ilk olarak havacılık alanında ihtiyaç duyulmuştur. Akademisyenler karmaşık uçak çerçeve yapılarının çözümü için bilgisayarla otomatikleştirilebilecek bir yöntem arayışı içine girmişlerdir. O zamanlarda bu kompleks yapıların çözümü için Açı Metodu ve Cross Metodu kullanılmaktaydı. Gelişen bilgisayar teknolojisine uygun bir metot geliştrilmesi için çalışmalara başlanmıştır. Sonlu elemanlar yöntemi ilk olarak matris deplasman metodu ile ortaya çıkmış ve diğer metotlar sonradan geliştirilerek bugünkü halini almıştır.

Sonlu elemanlar yönteminin teorisinin ilk izleri 1934'de bulunan matris deplasman metodu (Direct Stiffness Method) ile görülmüştür (Felippa, 2017).

1940'ların başlarında bina, uçak ve uzay yapılarının gerilme analizlerinin yapılabilmesi için sistematik bir yöntem arayışı ihtiyacı doğmuştur. Bu kapsamda A. Hrennikoff (Hrennikoff, 1941) ve R. Courant (Courant, 1943) tarafından yapılan çalışmalarla yöntemin temelleri atılmıştır. Bu iki akademisyen faklı yöntemler kullanmış olsalar da çözümün ağ yapıları (mesh) ile daha küçük çalışma alanlarına ayrıklaştırma yönünden yaptıkları çalışmalar aynıdır (Crahmaliuc, 2018).

Sonlu elemanlar yöntemi ilk olarak 1950 yılında uzay mühendisliğinde Boeing, Bell Aerospace ve Rolls Royce firmaları tarafından ilk olarak kullanılmıştır. Yöntemin ana fikrini oluşturan ilk makale 1956 yılında Turner ve arkadaşları tarafından yayınlanmıştır (Güler & Şen, 2015). Turner matris deplasman metodunu genelleştirip geliştirmiştir. Turner'ın çalışmaları Boing tarafından desteklenmiştir. Turner sonlu elemanlar yönteminin nonlineer problemlere uygulanmasına öncü olmuştur.

B. M. Irons izoparametrik modelleri geliştirmiş, şekil fonksiyonları ve yama testi metotlarını icat etmiştir. R. J. Melosh, Rayleigh-Ritz metodu ile varyasyonel olarak katılık matrislerini sistematik olarak türetmiştir. E. L. Wilson ilk açık kaynak sonlu elemanlar metodu çözücü programı geliştirmiştir (Felippa, 2017).

1959'da General Motors ve IBM otomobil dizaynı için sonlu elemanlar yöntemiyle kullanılmak üzere DAC-1 (Design Augmented by Computers) bilgisayar sistemini geliştirmiştir.

1965'te NASTRAN (NASA Structural Analysis) programı yapısal analiz çözücü olarak geliştirilmiştir.

1970’de şuan sonlu elemanlar için en genel ve en yaygın kullanılan bilgisayar programı Ansys geliştirilmiştir.

1973’de Gilbert Strang ve George Fix, sonlu elemanlarla ilgili ilk kitap olan “An Analysis of the Finite Element Method” kitabını yayınlayarak yönteme matematiksel olarak büyük katkı sağlamıştır (Crahmaliuc, 2018).

1977’de ilk profesyonel sonlu elemanlar p-versiyon kodu olan FIESTA Alberto Peano tarafından yazılmaya başlanmıştır.

1982’de Barna Szabo ve Kent Myer tarafından uçak ve uzay uygulamaları için ilk endüstriyel sonlu elemanlar p-versiyon yazılımı olan PROBE geliştirilmiştir.

1987’de MECHANICA, RASNA şirketi tarafından geliştirilmiştir.

2001’de P-versiyon sonlu elemanlar yöntemi plastisite uygulamaları için en verimli yöntem olduğu A. Duster tarafından ispatlanmıştır.

2006’da Amerikan Makine Mühendisleri Topluluğu tarafından Hesaplamalı Katı Mekaniğinde Doğrulama ve Onaylama kılavuzu yayınlanmıştır.

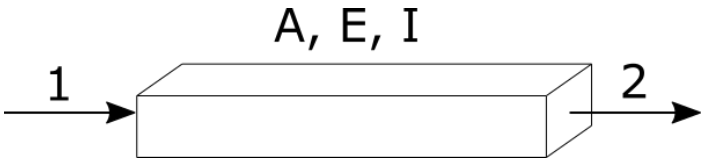
2008’de NASA model ve simülasyon geliştirilmesi için standart yayınlamıştır (Crahmaliuc, 2018).

2.2 Katılık Matrisi

Katılık en genel haliyle belirli bir doğrultuda birim deplasman elde edebilmek için belirli bir doğrultuda uygulanması gereken kuvvettir. Katılık matrisi $[k]$ le gösterilir. Matrisin herhangi bir elemanı k_{ij}, taşıyıcı elemanın bütün serbestlik dereceleri doğrultusundaki deplasmanlar sıfır iken j yönünde birim bir deplasman oluşturabilmek için i yönünde uygulaması gereken kuvvet anlamına gelir (Tezcan, 1970).

2.2.1 Eksenel yönde zorlanan kiriş elemanı

(Şekil 2.1)’de gösterilen kiriş elemanını ele alalım:



Şekil 2.1 : İki serbestlik dereceli kiriş elemanı.

2.2.1.1 Rijitlik matrisinin elde edilmesi

Taşıyıcı elemanın sadece 1 ve 2 yönlerinde serbestlik derecesi olduğunu varsayalım. k_{11} katılık elemanını bulalım. Bu katılık elemanının anlamı 1 yönünde birim deplasman oluşturmak için 1 yönünde uygulanması gereken kuvvettir. Mukavemetten;

$$\sigma = E\varepsilon = \frac{P}{A} \tag{2.1}$$

$$\varepsilon = \frac{\Delta L}{L} \tag{2.2}$$

Denklem (2.2), denklem (2.1)'de yerine koyulursa;

$$P = \frac{\Delta L E A}{L} \tag{2.3}$$

elde edilir. Tanım gereği ΔL birim deplasman olduğu için denklem 2.3'te $\Delta L = 1$ yazılırsa;

$$k_{11} = \frac{EA}{L} \tag{2.4}$$

olarak bulunur.

k_{12} katılık elemanı 2 yönünde birim deplasman oluşturmak için 1 yönünde uygulanması gereken kuvvettir. 1 yönündeki uygulanan kuvvet 2 yönünde ters yönde etki eder. Dolayısıyla 2 yönündeki deplasman oluşturmak için k_{11}'deki kuvvetin tersinin uygulanması gerekmektedir.

$$k_{12} = -\frac{EA}{L} \tag{2.5}$$

k_{21} katılık elemanı aynı mantıkla bulunur.

$$k_{21} = -\frac{EA}{L} \tag{2.6}$$

k_{22} ise k_{11} ile aynı olacağı anlaşılabilir.

$$k_{22} = \frac{EA}{L} \tag{2.7}$$

Bütün bu 4 durumunda elemana aynı anda etkidiği durumda 1 ve 2 yönündeki kuvvetler aşağıdaki gibi bulunabilir:

$$\begin{aligned} P_1 &= k_{11}d_1 + k_{12}d_2 \\ P_2 &= k_{21}d_1 + k_{22}d_2 \end{aligned} \tag{2.8}$$

Bu denklem takımı matris notasyonuyla aşağıdaki gibi yazılabilir:

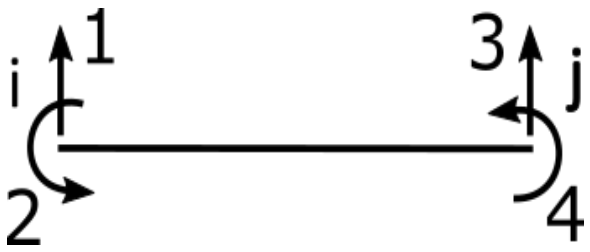
$$\begin{bmatrix} P_1 \\ P_2 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{2.9}$$

Bu denklem elemanın kuvvet denklemidir. Denklemin sol tarafındaki vektör yük vektörü, sağ tarafındaki matris katılık matrisi, sağ taraftaki vektör ise deplasman vektörüdür (Tezcan, 1970). Denklem 2.9 açık olarak aşağıdaki gibi yazılabilir:

$$\begin{bmatrix} P_1 \\ P_2 \end{bmatrix} = \frac{AE}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \tag{2.10}$$

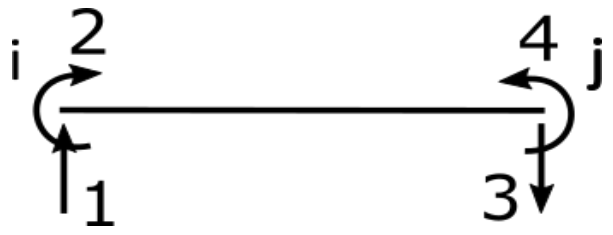
2.2.2 Eğilmeye zorlanan kiriş elemanı

Serbestlik dereceleri (Şekil 2.2)'deki gibi taşıyıcı elemanı ele alalım. Elemanın her iki ucunda bir öteleme bir de dönme olmak üzere toplamda 4 serbestlik derecesi olduğunu varsayalım.



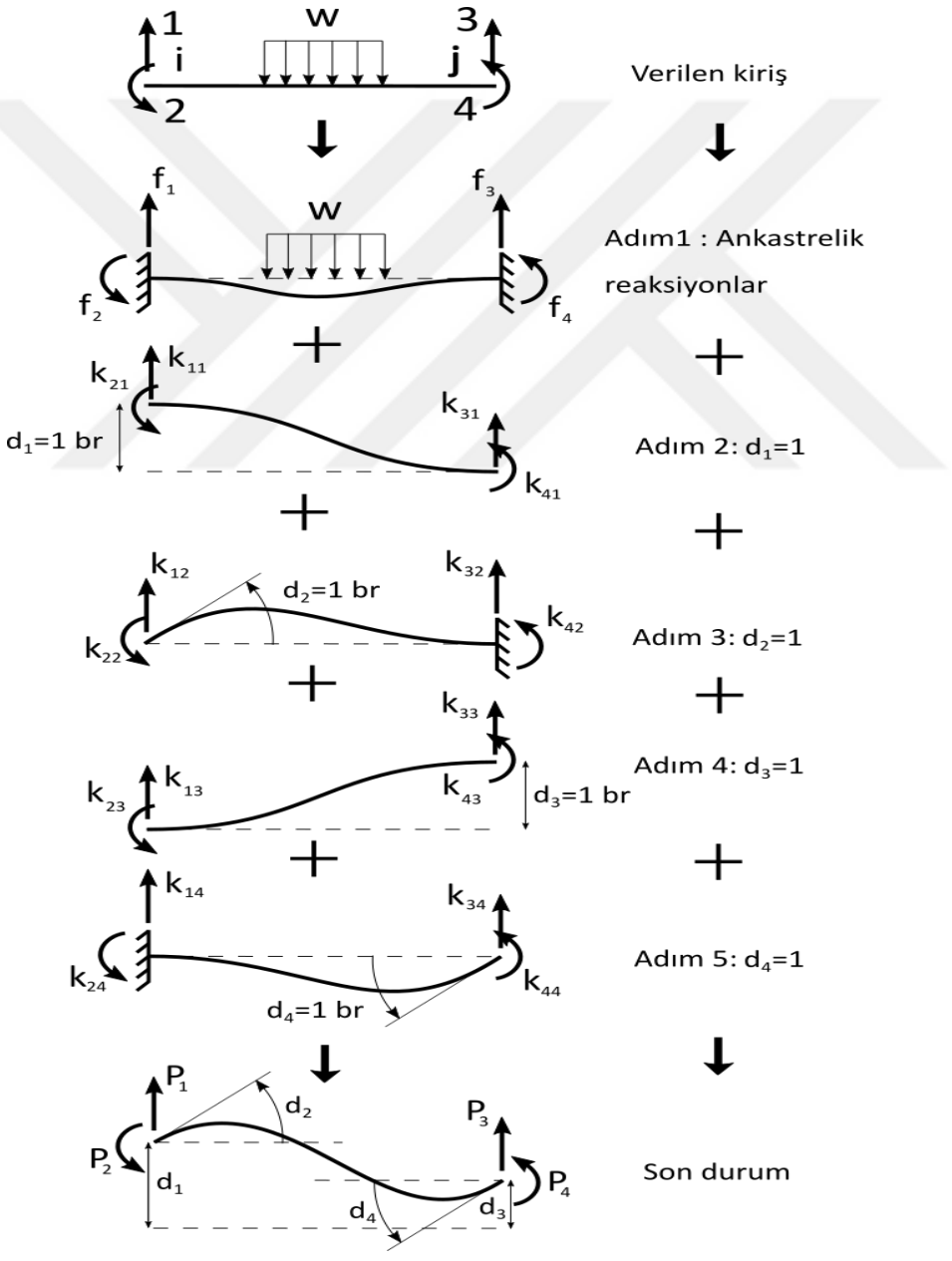
Şekil 2.2 : 4 serbestlik dereceli kiriş elemanı.

Pozitif serbestlik derecelerinin yönünün kiriş teorisindeki yönlere uymadığına burada dikkat edilmelidir. Kiriş teorisindeki serbestlik dereceleri yönü (Şekil 2.3)'te gösterilmiştir.



Şekil 2.3 : Kiriş teorisindeki pozitif serbestlik dereceleri.

Katılık matrisini elde etmek için (Şekil 2.4)'teki yol izlenebilir. Kirişin maruz kalacağı en genel şekil değiştirmeler süperpozisyon kanunu kullanarak her serbestlik derecesi yönünde birim deplasman etkidiği durumların toplamı olarak bulunabilir.

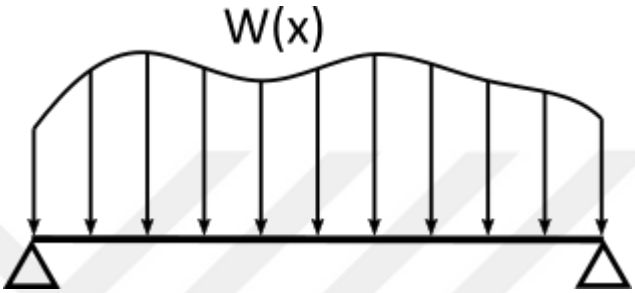


Şekil 2.4 : Düzlem eğilmeye maruz kiriş (Tezcan, 1970).

2.2.2.1 Moment alan teoremi

Moment alan teoremi eğilmeye maruz kalan kirişlerdeki sehim ve dönmenin türetilmesi için kullanılır. Mohr tarafından geliştirilen teorem tekil yüklerin yüklendiği durumlarda da ve atalet momentinin değişken olduğu durumlarda da sehim ve dönmenin elde edilmesine olanak veren bir metottur. Rijitlik matrisi hesaplarında moment alan yöntemi sıkça kullanılacaktır.

(Şekil 2.5)'te herhangi bir yayılı yüke maruz kalan bir kiriş verilmiştir.

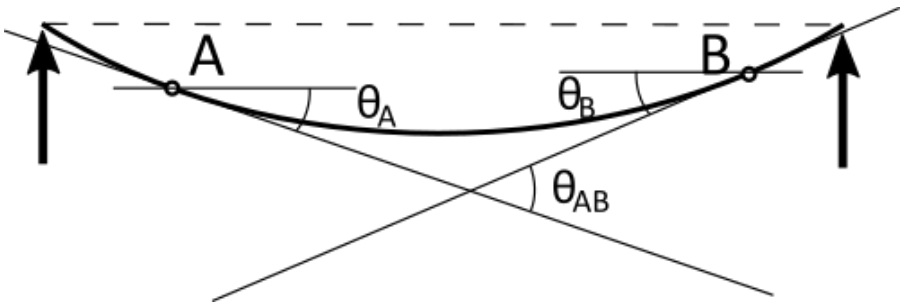


Şekil 2.5 : Herhangi bir yayılı yükle yüklenmiş eğilmeye maruz kiriş.

Bu yükten dolayı kirişin (Şekil 2.6)'daki gibi sehim yaptığı varsayılsın.

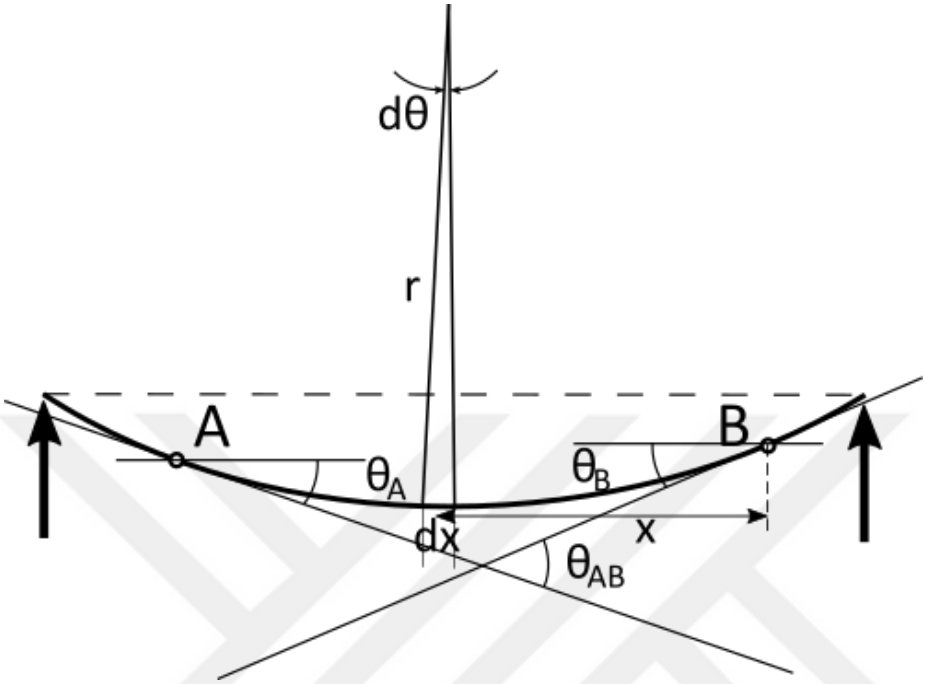
Şekil 2.6 : Yayılı yük ile yüklenen kirişin sehimi.

Kiriş üzerinde A ve B olmak üzere iki nokta seçilsin. Bu noktalardan çizilen elastik eğriye çizilen teğetler (Şekil 2.7)'deki gibi olur. Kiriş üzerindeki A ve B noktalarından çizilen teğetlerin eğimleri sırasıyla θ_A ve θ_B'dir. Kirişlerin kesişim açısı ise θ_{AB}'dir.



Şekil 2.7 : Kirişin üzerindeki iki noktadan çizilen teğetler.

(Şekil 2.8)'deki Elastik eğri üzerinde r yarıçaplı elastiklik merkezinden $d\theta$'lık bir yay ele alalım. Bu yayın yatay uzunluğu dx olsun ve B noktasından itibaren uzaklığı x olsun.



Şekil 2.8 : Kirişin elastik eğrisinin üzerindeki $d\theta$'lık yay.

(Şekil 2.8)'den;

$$dx = rd\theta \tag{2.11}$$

$$d\theta = \frac{1}{r}dx \tag{2.12}$$

Ayrıca kirişin eğilmesinden;

$$\frac{1}{r} = \frac{M}{EI} \tag{2.13}$$

Kirişin eğilme rijitliği EI'nın sabit olduğunu varsayarak denklem 2.13, denklem 2.12'de yerine konulursa;

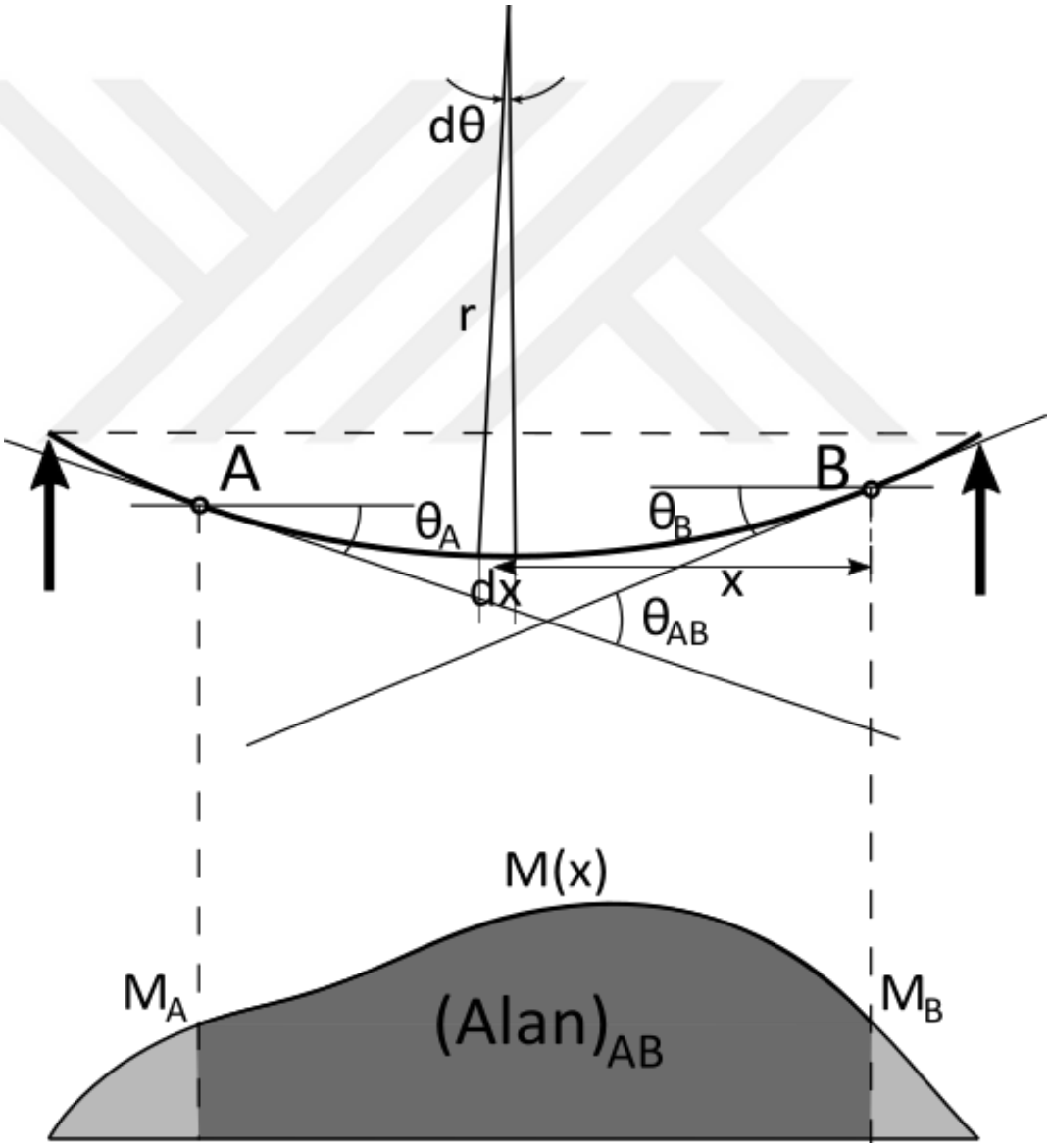
$$d\theta = \frac{M}{EI}dx \tag{2.14}$$

Her tarafın integrali alınırsa;

$$\int_{\theta_A}^{\theta_B} d\theta = \int_{x_A}^{x_B} \frac{M}{EI} dx \tag{2.15}$$

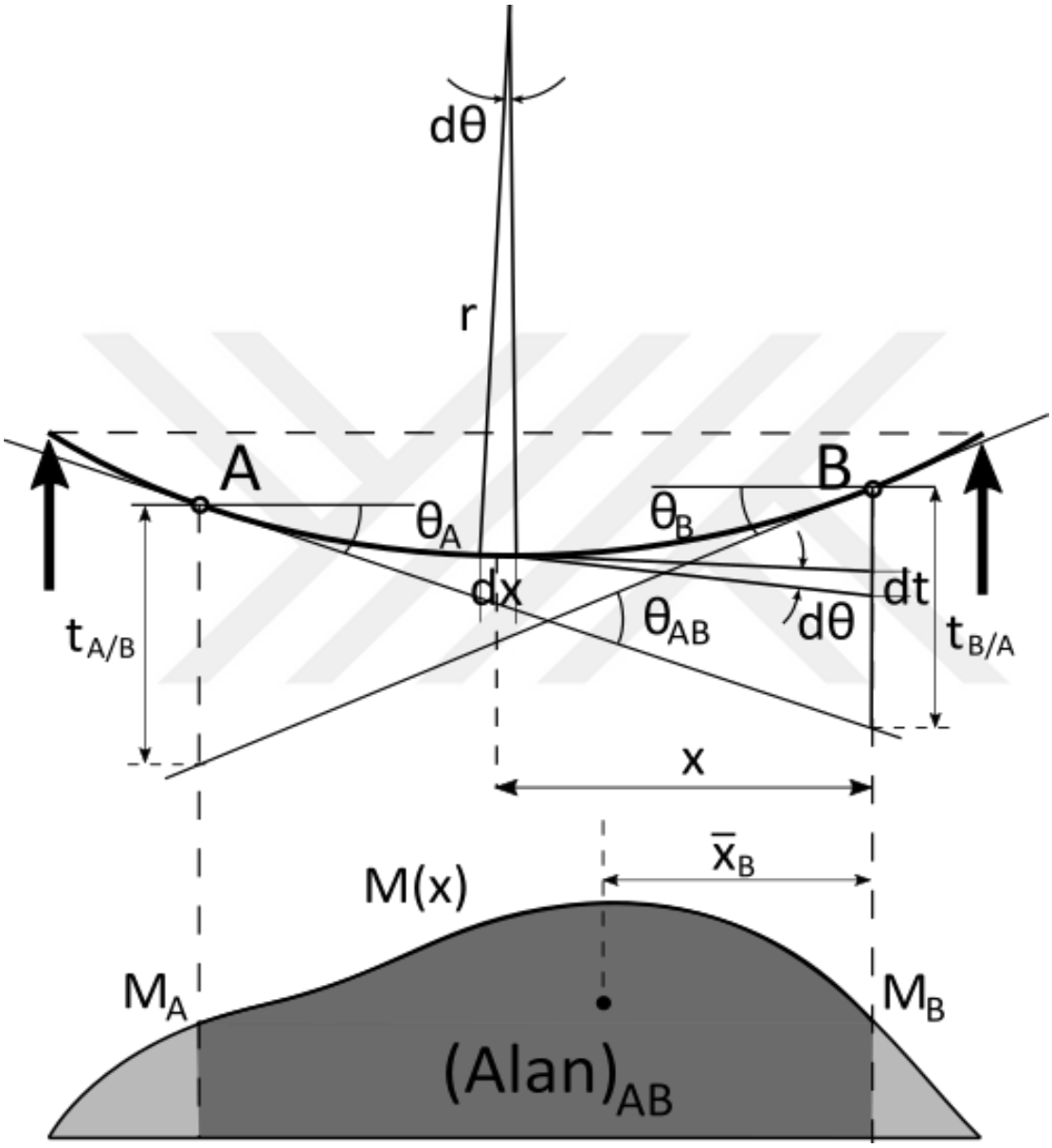
$$\theta_{AB} = \frac{1}{EI} (Alan)_{AB} \tag{2.16}$$

Bu, birinci moment alan teoremidir. Sözlü olarak ifade edilirse: Elastik eğriye üzerindeki herhangi iki noktadan çizilen teğetler arasındaki açı değişimi o noktalar arasındaki moment alanının eğilme rijitliğine bölümüne eşittir (Limbrunner & Spiegel, 2009). (Şekil 2.9)'da bu durum detaylı olarak gösterilmiştir.



Şekil 2.9 : Kiriş üzerinde birinci moment alan teoremi gösterimi.

A noktasından çizilen teğetin B noktasına olan düşey uzunluğuna sapma denir ve $t_{B/A}$ ile gösterilir. dx uzunluklu yaydan B noktasındaki düşey sapma üzerine aralarında $d\theta$ açısı olacak şekilde doğrular çizilirse, doğruların $t_{B/A}$'yi kestiği mesafe dt olur. (Şekil 2.10)'da bu durum gösterilmiştir.



Şekil 2.10 : Kiriş üzerinde ikinci moment alan teoremi gösterimi.

Şekilde de görüldüğü gibi;

$$dt = xd\theta \tag{2.17}$$

$d\theta$ ifadesinin denklem 2.14'teki değeri yazılırsa;

$$dt = x\left(\frac{M}{EI}dx\right) \tag{2.18}$$

Her tarafın integrali alınırsa;

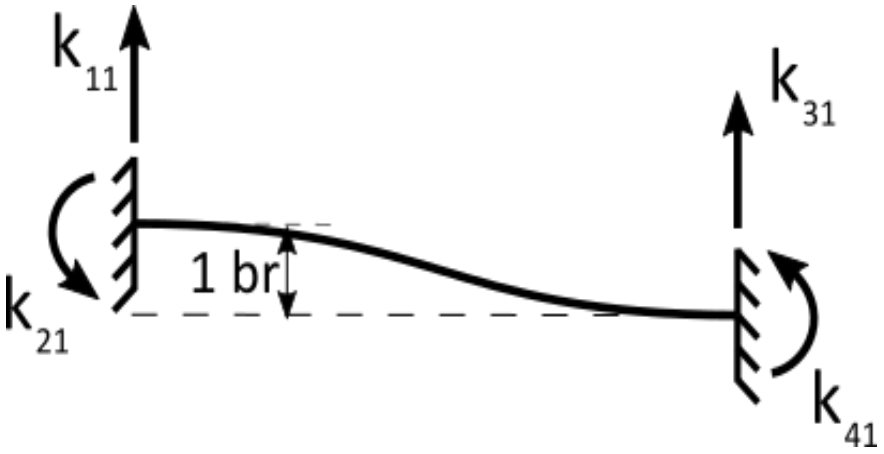
$$\int_A^B dt = \frac{1}{EI}\int_{x_A}^{x_B} xMdx \tag{2.19}$$

$$t_{B/A} = \frac{1}{EI}(Alan)_{AB}\bar{x}_B \tag{2.20}$$

Burada $\bar{x}_B$ A ve B noktaları arasındaki moment diyagramının alan merkezinin B noktasına uzaklığıdır. Bu, ikinci moment alan teoremidir. Yine sözlü olarak ifade edilirse: Elastik eğri üzerindeki herhangi iki A ve B noktaları için, A'dan çizilen teğet doğrusunun B noktasına olan dik uzaklığı, moment alanının eğilme rijitliğine bölümünün statik momentine eşittir (Limbrunner & Spiegel, 2009).

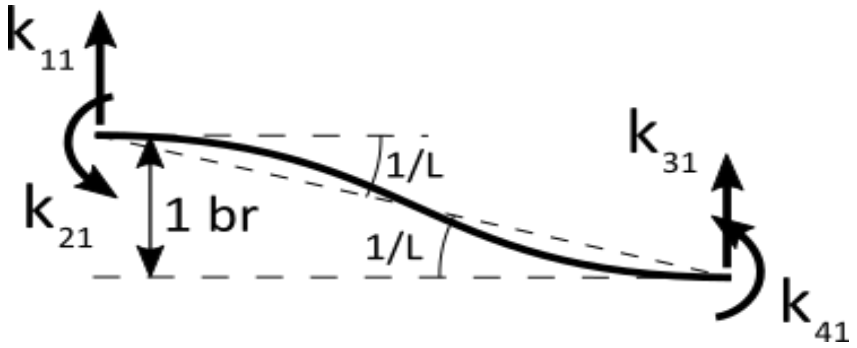
2.2.2.2 Rijitlik matrisinin elde edilmesi

1 numaralı serbestlik derecesi yönünde birim deplasman uygulamak için gereken kuvvetler daha önce yapılan tanıma göre (Şekil 2.11)'deki gibi olacaktır.



Şekil 2.11 : 1 numaralı serbestlik derecesi yönünde birim deplasman oluşturmak için gerekli kuvvetler.

Burada geçen kuvvet tanımı genel kuvvettir. Yani hem eğilme momenti hem de kesme kuvveti genel kuvvet olarak sayılmaktadır. $tan\alpha \cong \alpha$ yaklaşık eşitliğini kullanarak (Şekil 2.11)'deki kiriş (Şekil 2.12)'deki gibi olur:



Şekil 2.12 : (Şekil 2.11)'deki kirişte $tan\alpha \cong \alpha$ olması hali.

Kuvvet eşitliği kullanılarak;

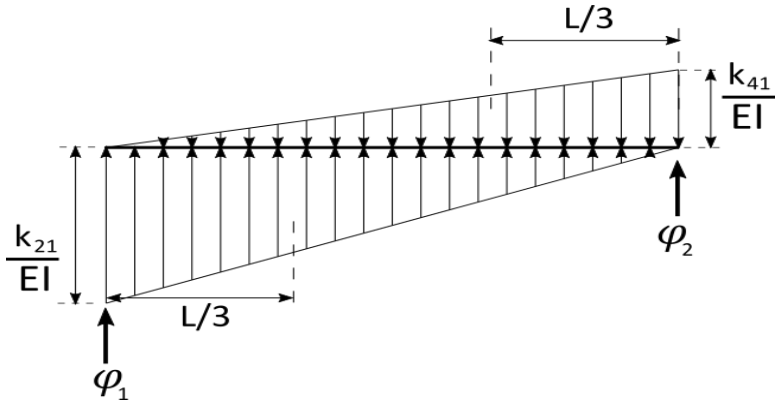
$$k_{11} + k_{31} = 0 \tag{2.21}$$

$$k_{11} = -k_{31} \tag{2.22}$$

Bu aşamada kiriş teorisi işaret sisteminde çalışıldığı belirtilmelidir. Kirişin sağ ucuna göre moment alınırsa;

$$k_{11}L - k_{21} - k_{41} = 0 \tag{2.23}$$

Elemanın bu yükler altında moment dağılımı bulunup birinci Moment-Alan teoremi uygulandığında durum (Şekil 2.13)'deki gibi olur.



Şekil 2.13 : 1 numaralı serbestlik derecesi yönünde birim deplasman uygulanan kirişe Moment-Alan teoreminin uygulanması.

Bilindiği üzere Moment-Alan teoremi kirişteki eğilme momenti dağılımının kirişe yük olarak yüklenmesi ile kirişteki dönme ve sehimlerin bulunmasını içerir. Moment

dağılımı kirişe yüklenmeden önce EI'ya bölünür. Elde edilen yeni yükleme tipinde mesnet reaksiyon kuvvetleri o noktalardaki dönmeye karşılık gelir.

Bu durumda φ_1 kirişin sol ucundaki dönme, φ_2 kirişin sağ ucundaki dönme olacaktır. (Şekil 2.12)'de $\varphi_1 \cong -1/L$ olduğu görülebilir. Burada eksi işareti dönmenin kiriş teorisindeki dönme yönüne ters olduğu içindir. φ_2'nin ankastre mesnetten dolayı sıfır olacağı ilk olarak düşünülebilir. Ancak burada kirişin ekseni değişmiştir. Dolayısıyla hala bir dönme vardır ve bu dönme yine (Şekil 2.12)'den $\varphi_2 = 1/L$ olduğu anlaşılır. Dönme yönü kiriş teorisindekiyle aynı olduğu için φ_2 pozitiftir. Düşey kuvvetlerin eşitliği yazılırsa;

$$\varphi_1 + \varphi_2 + \frac{k_{21}}{EI}\frac{L}{2} - \frac{k_{41}}{EI}\frac{L}{2} = 0 \tag{2.24}$$

$$-\frac{1}{L} + \frac{1}{L} + \frac{k_{21}}{EI}\frac{L}{2} - \frac{k_{41}}{EI}\frac{L}{2} = 0 \tag{2.25}$$

$$k_{21} = k_{41} \tag{2.26}$$

elde edilir. Kirişin sağ uç noktasına göre moment alınarak ikinci Moment-Alan Teoremi uygulanırsa;

$$\varphi_1 L + \frac{k_{21}}{EI}\frac{L}{2}\frac{2L}{3} - \frac{k_{41}}{EI}\frac{L}{2}\frac{L}{3} = 0 \tag{2.27}$$

Denklem 2.26, denklem 2.27'de yerine yazılırsa;

$$-\frac{1}{L}L + \frac{k_{21}}{EI}\frac{L}{2}\frac{2L}{3} - \frac{k_{21}}{EI}\frac{L}{2}\frac{L}{3} = 0 \tag{2.28}$$

$$k_{21} = k_{41} = \frac{6EI}{L^2} \tag{2.29}$$

Denklem 2.29'daki bulunan katılık elemanları denklem 2.23'te yerine konulursa;

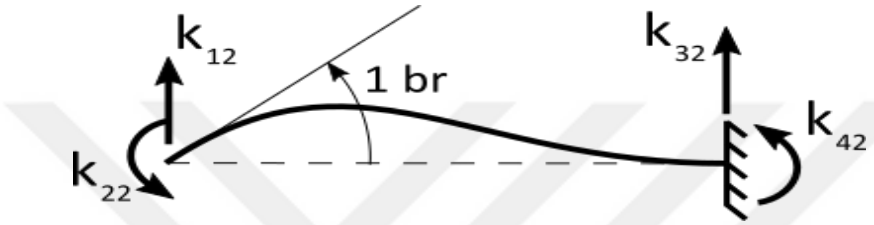
$$k_{11}L - \frac{6EI}{L^2} - \frac{6EI}{L^2} = 0 \tag{2.30}$$

$$k_{11} = \frac{12EI}{L^3} \tag{2.31}$$

Denklem 2.22'den yararlanarak;

$$k_{13} = -\frac{12EI}{L^3} \tag{2.32}$$

2 numaralı serbestlik derecesi yönünde birim deplasman uygulamak için gereken kuvvetler daha önce yapılan tanıma göre (Şekil 2.14)'teki gibi olacaktır.



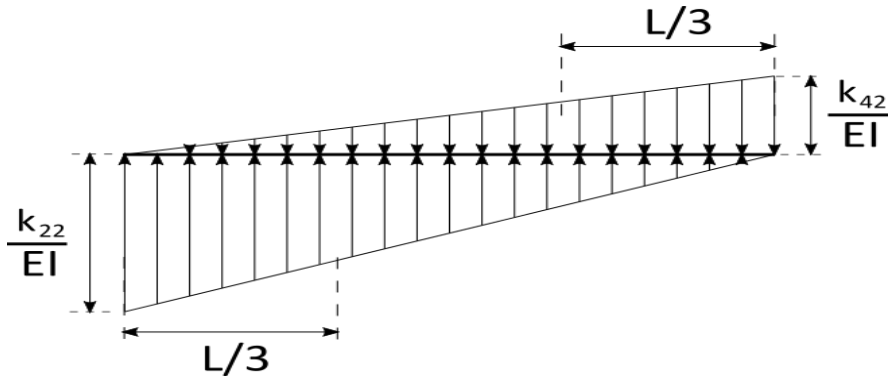
Şekil 2.14 : 2 numaralı serbestlik derecesi yönünde birim deplasman oluşturmak için gerekli kuvvetler.

Yine kuvvet ve moment dengesinden;

$$k_{12} = -k_{32} \tag{2.33}$$

$$k_{12}L - k_{22} - k_{42} = 0 \tag{2.34}$$

Tekrardan kirişe (Şekil 2.15)'teki gibi Moment-Alan teoremi uygulansın.



Şekil 2.15 : 2 numaralı serbestlik derecesi yönünde birim deplasman uygulanan kirişe Moment-Alan teoreminin uygulanması.

Düşey kuvvetlerin eşitliğinden;

$$\varphi_1 + \varphi_2 + \frac{k_{22}}{EI}\frac{L}{2} - \frac{k_{42}}{EI}\frac{L}{2} = 0 \tag{2.35}$$

Burada sağ uçta ankastre mesnet olduğu için buradaki dönme sıfırdır. Yani $\varphi_2 = 0$ olur. Burada birim dönme olduğu için $\varphi_1 = -1$ radyandır. Kiriş teorisindeki yönün tersi yönde dönme olduğu için negatiftir. Bu değerler denklem 2.35'te yerine konulur ve denklem düzenlenirse;

$$k_{22} - k_{42} = \frac{2EI}{L} \tag{2.36}$$

Kirişin sol uç noktasına göre moment alınarak ikinci Moment-Alan teoremi uygulanırsa;

$$\frac{k_{42}}{EI}\frac{L}{2}\frac{2L}{3} - \frac{k_{22}}{EI}\frac{L}{2}\frac{L}{3} = 0 \tag{2.37}$$

$$k_{22} = 2k_{42} \tag{2.38}$$

Denklem 2.38'deki ilişki denklem 2.36'da yerine yazılırsa;

$$2k_{42} - k_{42} = \frac{2EI}{L} \tag{2.39}$$

$$k_{42} = \frac{2EI}{L} \tag{2.40}$$

$$k_{22} = \frac{4EI}{L} \tag{2.41}$$

Bu değerler denklem 2.34'te yerine konulursa;

$$k_{12}L - \frac{4EI}{L} - \frac{2EI}{L} = 0 \tag{2.42}$$

$$k_{12} = \frac{6EI}{L^2} \tag{2.43}$$

Denklem 2.33'ten;

$$k_{32} = -\frac{6EI}{L^2} \tag{2.44}$$

Burada yapılan hesaplar 3 numaralı ve 4 numaralı serbestlik dereceleri için de yapılırsa aşağıdaki sonuçlar elde edilir:

$$k_{13} = -\frac{12EI}{L^3} \tag{2.45}$$

$$k_{23} = -\frac{6EI}{L^2} \tag{2.46}$$

$$k_{33} = \frac{12EI}{L^3} \tag{2.47}$$

$$k_{34} = -\frac{6EI}{L^2} \tag{2.48}$$

$$k_{14} = \frac{6EI}{L^2} \tag{2.49}$$

$$k_{24} = \frac{2EI}{L} \tag{2.50}$$

$$k_{34} = -\frac{6EI}{L^2} \tag{2.51}$$

$$k_{44} = \frac{4EI}{L} \tag{2.52}$$

Sonuç olarak bütün elde edilen değerler yerine konulursa katılık matrisi aşağıdaki gibi olur:

$$k = \begin{bmatrix} \frac{12EI}{L^3} & \frac{6EI}{L^2} & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ \frac{6EI}{L^2} & \frac{4EI}{L} & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ \frac{6EI}{L^2} & \frac{2EI}{L} & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \tag{2.53}$$

2.2.3 6 serbestlik dereceli çerçeve elemanı

Denklem 2.10'daki katılık matrisi $6x6$ boyutunda olacak şekilde aşağıdaki gibi genişletilebilir:

$$k_1 = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.54}$$

Aynı şekilde denklem 2.43'teki matris de $6x6$ boyutunda olacak şekilde aşağıdaki gibi genişletilebilir:

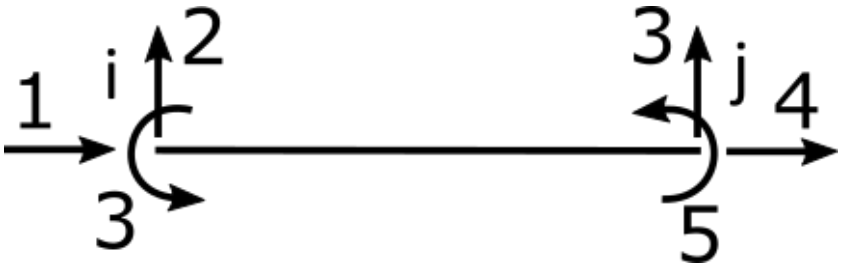
$$k_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \tag{2.55}$$

Bu iki katılık matrisi toplanırsa aşağıdaki matris elde edilir:

$$k = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \tag{2.56}$$

Bu matris, serbestlik dereceleri (Şekil 2.16)'da gösterilen 6 serbestlik dereceli kirişin katılık matrisidir (Omurtag, 2010). Kirişin hem yatay öteleme, hem dikey öteleme, hem de dönme serbestliklerine izin veren bu katılık matrisi düzlem çerçeve

elemanlarının en genel halidir. Bu çalışmada kirişin bu 6 serbestlik dereceli hali kullanılmıştır.



Şekil 2.16 : 6 serbestlik dereceli kiriş.

2.2.4 Değişken kesitli kiriş durumu

Taşıyıcı kirişin yüksekliği ve genişliği değişken olduğu durumda ya da braketlerle desteklendiği durumda katılık matrisi de değişir. Bu değişim temel rijitlik katsayıları yardımıyla katılık matrisine yansıtılır. Sabit kesitli durumda $n_{ii} = 1$, $m_{ii} = 4$, $m_{jj} = 4$ ve $m_{ij} = 2$ olur (Karaduman, 1993).

Temelde n_{ii}, m_{ii}, m_{jj} ve m_{ij} olmak üzere 4 tane rijitlik katsayısı vardır.

$m_{ii}\frac{EI_0}{L}$: Kirişin i ucunu birim döndürmek için i ucuna uygulanması gereken momenttir.

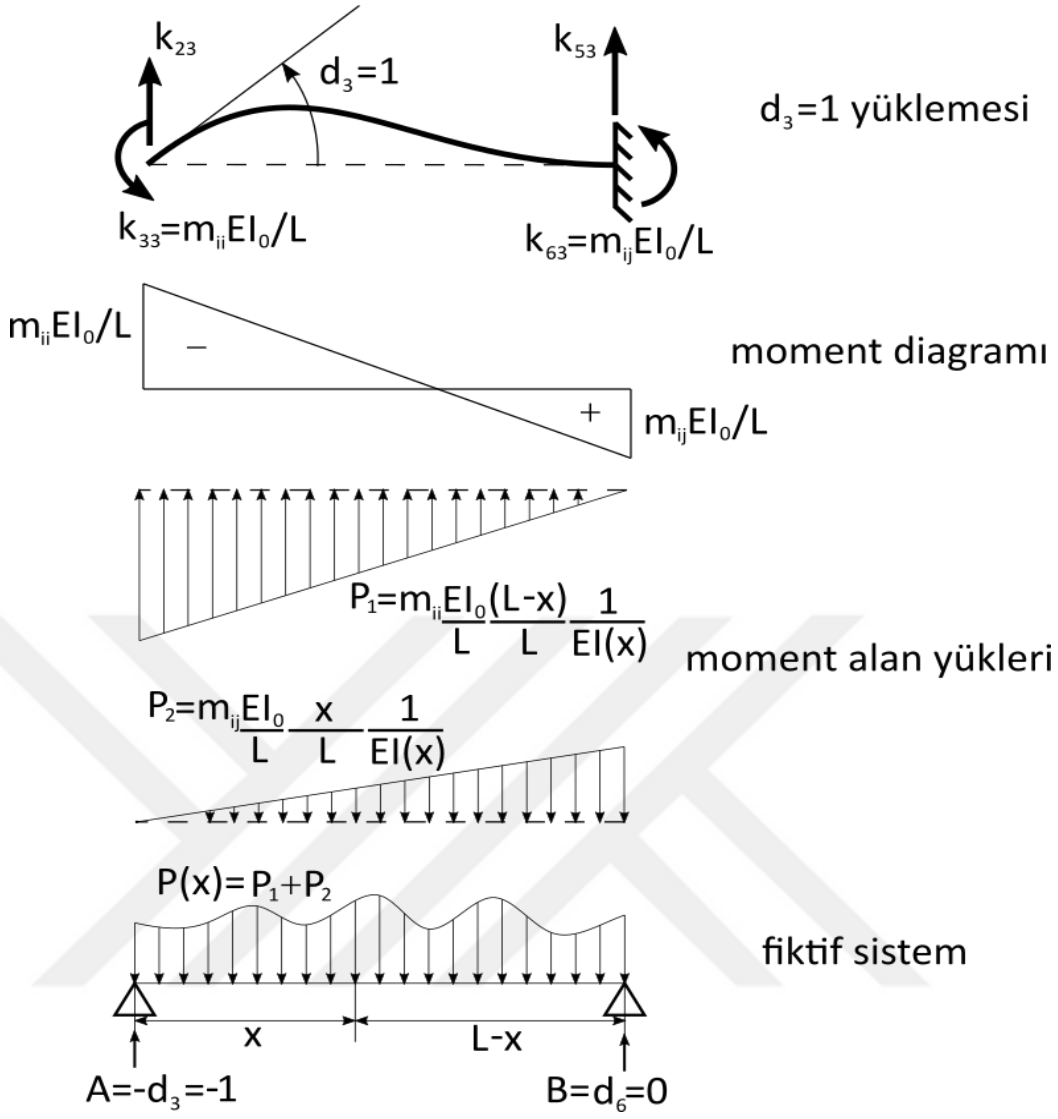
$m_{jj}\frac{EI_0}{L}$: Kirişin j ucunu birim döndürmek için j ucuna uygulanması gereken momenttir.

$m_{ij}\frac{EI_0}{L}$: Kirişin i ucunu birim döndürmek için j ucuna uygulanması gereken momenttir. Ayrıca bunun tersi de doğrudur; $m_{ij} = m_{ij}$ eşitliği her zaman sağlanır.

$n_{ii}\frac{EA_0}{L}$: Alan değişken olduğu durumda elemana birim deplasman uygulanınca eleman üzerinde oluşan eksenel kuvvettir.

Burada I_0 kesitin minimum atalet momenti, A_0 kesitin kirişin minimum alanı, L kirişin uzunluğu, E kirişin elastisite modülüdür.

6 serbestlik dereceli kirişte d_3 birim dönme durumu için moment alan yöntemi atalet momentinin değişken olduğu göz önüne alınarak uygulanırsa yapılan tanımlara göre $k_{33} = m_{ii}\frac{EI_0}{L}$ ve $k_{63} = m_{ij}\frac{EI_0}{L}$ olacaktır. (Şekil 2.17)'de bu uygulama gösterilmiştir.



Şekil 2.17 : $d_3 = 1$ ve değişken kesitli durum için moment alan yönteminin uygulanması (Karaduman, 1993).

k_{33} ve k_{63} değerlerinin bilinmesi için m_{ii} ve m_{ij} bulunması gerekir. Fiktif sistemde $M_B = 0$ eşitliği yazılırsa;

$$-AL - m_{ii}\frac{EI_0}{L^2}\int_0^L \frac{(L-x)^2}{EI(x)}dx + m_{ij}\frac{EI_0}{L^2}\int_0^L \frac{x(L-x)}{EI(x)}dx \tag{2.57}$$

Fiktif sistemde $M_A = 0$ eşitliği yazılırsa;

$$BL - m_{ii}\frac{EI_0}{L^2}\int_0^L \frac{x(L-x)}{EI(x)}dx - m_{ij}\frac{EI_0}{L^2}\int_0^L \frac{x^2}{EI(x)}dx \tag{2.58}$$

$d_6 = 1$ için de moment alan teoremi benzer şekilde yazılarak m_{jj} ifadesine ait denklemler bulunabilir.

$$I_1 = \int_0^L \frac{x^2}{EI(x)}dx\,, \qquad I_2 = \int_0^L \frac{1}{EI(x)}dx\,, \qquad I_3 = \int_0^L \frac{x}{EI(x)}dx \tag{2.59}$$

Tanımları yapılarak temel rijitlik katsayıları aşağıdaki gibi elde edilir (Karaduman, 1993):

$$m_{ii} = \frac{I_1 L}{I_0(I_1 I_2 - I_3^2)}\,, \qquad m_{ij} = \frac{L(I_3 L - I_1)}{I_0(I_1 I_2 - I_3^2)}\,,$$
$$m_{jj} = \frac{L(-I_2 L^2 - 21 L I_3 - I_3)}{I_0(I_1 I_2 - I_3^2)} \tag{2.60}$$

(Şekil 2.1)'deki eksenel yüklü kiriş için $d_1 = 1$ olduğu durumda kirişte $-n_{ii}\frac{EA_0}{L}$ eksenel kuvveti oluşur (Karaduman, 1993). Bu durumda elemanın boyca uzuma denklemi yazılırsa;

$$\Delta L = d_2 - d_1 = -1 = -n_{ii}\frac{EA_0}{L}\int_0^L \frac{1}{EA(x)} \tag{2.61}$$

$$n_{ii} = \frac{L}{A_0}\left(\int_0^L \frac{1}{A(x)}\right)^{-1} \tag{2.62}$$

elde edilir.

Bu tanımlar yapıldıktan sonra 6 serbestlik dereceli kiriş için rijitlik matrisi elemanları aşağıdaki gibi olacaktır (Karaduman, 1993):

$$k_{11} = \frac{n_{ii}EA_0}{L}, k_{14} = -k_{11}\,, \qquad k_{22} = \frac{(m_{ii} + m_{jj} + 2m_{ij})EI_0}{L^3}\,,$$
$$k_{23} = \frac{(m_{ii} + m_{ij})EI_0}{L^2}\,, k_{25} = -k_{22}\,, k_{26} = \frac{(m_{jj} + m_{ij})EI_0}{L^2}\,,$$
$$k_{33} = \frac{m_{ii}EI_0}{L}\,, k_{35} = -k_{23}\,, \qquad k_{36} = \frac{m_{jj}EI_0}{L^2}\,, \qquad k_{44} = k_{11}\,, \tag{2.63}$$
$$k_{55} = k_{22}\,, \qquad k_{56} = k_{26}\,, \qquad k_{66} = \frac{m_{jj}EI_0}{L}\,,$$
$$k_{12} = k_{13} = k_{15} = k_{16} = k_{24} = k_{34} = k_{45} = k_{46} = 0$$

Ayrıca simetriden dolayı $k_{ij} = k_{ij}$'dir.

2.3 Yük Vektörü

Katılık katsayılarıyla deplasmanlar arasındaki ilişki aşağıdaki gibi yazılır:

$$\begin{aligned}
P_1 &= k_{11}d_1 + k_{12}d_2 + k_{13}d_3 + k_{14}d_4 + k_{15}d_5 + k_{16}d_6 + f_1 \\
P_2 &= k_{21}d_1 + k_{22}d_2 + k_{23}d_3 + k_{24}d_4 + k_{25}d_5 + k_{26}d_6 + f_2 \\
P_3 &= k_{31}d_1 + k_{13}d_2 + k_{33}d_3 + k_{34}d_4 + k_{35}d_5 + k_{36}d_6 + f_3 \\
P_4 &= k_{41}d_1 + k_{42}d_2 + k_{43}d_3 + k_{44}d_4 + k_{45}d_5 + k_{46}d_6 + f_4 \\
P_5 &= k_{51}d_1 + k_{52}d_2 + k_{53}d_3 + k_{54}d_4 + k_{55}d_5 + k_{56}d_6 + f_5 \\
P_6 &= k_{61}d_1 + k_{62}d_2 + k_{63}d_3 + k_{64}d_4 + k_{65}d_5 + k_{66}d_6 + f_6
\end{aligned} \tag{2.64}$$

Denklem takımı matris formunda yazılırsa:

$$\begin{Bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \end{Bmatrix} = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} & k_{16} \\ k_{21} & k_{22} & k_{23} & k_{24} & k_{25} & k_{26} \\ k_{31} & k_{32} & k_{33} & k_{34} & k_{35} & k_{36} \\ k_{41} & k_{42} & k_{43} & k_{44} & k_{45} & k_{46} \\ k_{51} & k_{52} & k_{53} & k_{54} & k_{55} & k_{56} \\ k_{61} & k_{62} & k_{63} & k_{64} & k_{65} & k_{66} \end{bmatrix} \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{Bmatrix} + \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{Bmatrix} \tag{2.65}$$

ya da,

$$\{P\} = [k]\{d\} + \{f\} \tag{2.66}$$

Burada $\{P\}$ direkt yük vektörü, $[k]$ katılık matrisi, $\{d\}$ deformasyon vektörü ve $\{f\}$ endirekt yük vektörüdür.

Direkt yük vektörü nodlara gelen yüklerden oluşur. Endirekt yük vektörü ise kiriş üzerindeki yüklerin kirişin ucundaki nodlarda oluşturduğu tepkilerden oluşur. Katılık katsayıları tanım gereği serbestlik dereceleri yönündeki bütün deplasmanlar sıfır iken oluşturulur. Sistem endirekt yük vektörü de bu durumda oluşan yüklerdir. Diğer bir deyişle endirekt yükler kirişin her iki ucu ankastre mesnet iken bu mesnetlerde oluşan tepki kuvvetleri ve momentlerdir.

Bu çalışmada ele alınan problem kirişlerdeki atalet dağılımının değişken olmasına da izin verir. Atalet dağılımının değişken olması kirişteki moment dağılımına da etki eder. Bu durumda hesap yöntemi de değişir. Clapeyron'un Üç Moment Yöntemi bu

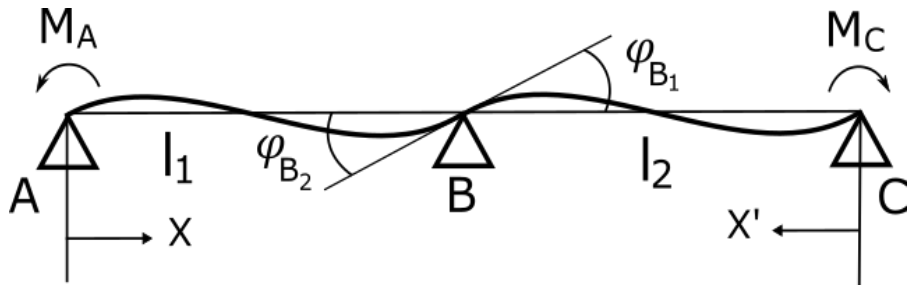
durumdaki mesnet tepkilerini hesaplayabilecek kapasitede bir metottur. Bu çalışmada bu yöntem kullanılmaktadır.

2.3.1 Üç moment metodu

Üç Moment Metodu sürekli kirişlerin mesnet momentlerini bulmak için kullanılan bir yöntemdir. Benoît Paul Émile Clapeyron tarafından 1857 yılında geliştirilmiştir. Geliştirilmesine kiriş eğilmelerinin diferansiyel denklemi kullanılmıştır (Gavin, 2009). Üç Moment Metodu elastik ve analitik bir analiz metodudur. Birbirine bağlı herhangi iki kiriş parçası için yazılabilir (Fertis, 1997). Ancak bir mesnede ikiden fazla kiriş bağlandığı durumlarda bu metot kullanılamaz. Oldukça geniş bir kullanım alanı vardır. Kirişlerin kendi içinde atalet momentlerinin değiştiği durumlarda, her türlü eğilme yüklenmesinde kullanılabilir.

Üç Moment Metodu aslında kendi başına sürekli kirişleri çözebilen bir metottur. Bu çalışmada geliştirilen programda Matris Deplasman Metodunda tanım gereği her iki ucun ankastre mesnet olduğu ve atalet momentinin değişken olduğu durumda mesnet tepkilerini ve dolayısıyla endirekt yük vektörlerini hesaplamak için kullanılmıştır.

(Şekil 2.18)'deki gibi bir sürekli kirişin herhangi üç mesnetli kısmını ele alalım.



Şekil 2.18 : Sürekli bir kirişin üç mesnetli kısmı (Savcı, 1988).

Bu parçalarında uç momentler M_A, M_B ve M_C olsun. B mesnedindeki durumu inceleyelim;

$$M_{B_1} = M_{B_2} = M \tag{2.67}$$

$$\varphi_{B_1} = -\varphi_{B_2} \tag{2.68}$$

Mohr teoremine göre;

$$i(x) = \frac{I_0}{I(x)} \tag{2.69}$$

$$i(x') = \frac{I_0}{I(x')} \tag{2.70}$$

Burada I_0 mesnedin herhangi bir noktasındaki sabit bir atalet momentidir. $I(x)$ sol taraftaki kirişin atalet momenti fonksiyonudur. $I(x')$ sağ taraftaki kirişin atalet momenti fonksiyonudur. Mohr teoreminde;

$$EI_0\varphi_B = -\frac{1}{l}\int_0^l M(x)i(x)xdx \tag{2.71}$$

Buradan;

$$EI_0\varphi_{B_1} = -\frac{1}{l_1}\int_0^{l_1} M(x)i(x)xdx \tag{2.72}$$

$$EI_0\varphi_{B_2} = -\frac{1}{l_2}\int_0^{l_2} M(x')i(x')x'dx' \tag{2.73}$$

Denklem 2.68'deki açıların eşitliğinden aşağıdaki ifade yazılabilir (Savcı, 1988);

$$\frac{1}{l_1}\int_0^{l_1} M(x)i(x)xdx + \frac{1}{l_2}\int_0^{l_2} M(x')i(x')x'dx' = 0 \tag{2.74}$$

Ayrıca yine Mohr teoreminden;

$$M(x) = M_0(x) + M_A + \frac{x}{l_1}(M_B - M_A) \tag{2.75}$$

$$M(x') = M_0(x') + M_C + \frac{x'}{l_1}(M_B - M_C) \tag{2.76}$$

Burada $M_0(x)$ kirişin her iki ucunun serbest mesnet olduğu durumdaki moment dağılımıdır. Denklem 2.76 ve denklem 2.75, denklem 2.74'te yerine konulursa aşağıdaki denklem elde edilir;

$$\frac{1}{l_1}\int_0^{l_1} M_0(x)i(x)xdx + \frac{1}{l_1}\int_0^{l_1} M_A(x)i(x)xdx + \frac{1}{l_1^2}\int_0^{l_1}(M_B - M_A)i(x)x^2dx + \frac{1}{l_2}\int_0^{l_2} M_0(x')i(x')x'dx' + \frac{1}{l_2}\int_0^{l_2} M_C(x')i(x')x'dx' + \frac{1}{l_2^2}\int_0^{l_2}(M_B - M_C)i(x')x'^2dx' = 0 \quad (2.77)$$

Denklem düzenlenirse;

$$M_A\left[\frac{1}{l_1}\int_0^{l_1} i(x)xdx - \frac{1}{l_1^2}\int_0^{l_1} i(x)x^2dx\right] + M_B\left[\frac{1}{l_1^2}\int_0^{l_1} i(x)x^2dx + \frac{1}{l_2^2}\int_0^{l_2} i(x')x'^2dx'\right] + M_C\left[\frac{1}{l_2}\int_0^{l_2} i(x')x'dx' - \frac{1}{l_2^2}\int_0^{l_2} i(x')x'^2dx'\right] = -\frac{1}{l_1}\int_0^{l_1} M_0(x)i(x)xdx - \frac{1}{l_2}\int_0^{l_2} M_0(x')i(x')x'dx' \quad (2.78)$$

Bu denklem Clapeyron'un Üç Moment Denklemidir (Savcı, 1988).

Kirişlerin atalet momentleri kendi içinde sabit olduğu durumda;

$$i(x) = i_1 = \frac{I_0}{I_1},\ i(x') = i_2 = \frac{I_0}{I_2} \quad (2.79)$$

$$M_A i_1 l_1 + 2M_B(i_1 l_1 + i_2 l_2) + M_C i_2 l_2 = -\frac{6i_1}{l_1}\int_0^{l_1} M_0(x)xdx - \frac{6i_2}{l_2}\int_0^{l_2} M_0(x')x'dx' - \quad (2.80)$$

$$K_A = \frac{6}{l_1^2}\int_0^{l_1} M_0(x)xdx,\ K_C = \frac{6}{l_2^2}\int_0^{l_2} M_0(x')x'dx' \quad (2.81)$$

K_A ve K_C denklem 2.80'de yerine konulursa (Savcı, 1988);

$$M_A i_1 l_1 + 2M_B(i_1 l_1 + i_2 l_2) + M_C i_2 l_2 = -i_1 l_1 K_A - i_2 l_2 K_C$$

$$M_A \frac{I_0}{I_1} l_1 + 2M_B \left(\frac{I_0}{I_1} l_1 + \frac{I_0}{I_2} l_2\right) + M_C \frac{I_0}{I_2} l_2 = -\frac{I_0}{I_1} l_1 K_A - \frac{I_0}{I_2} l_2 K_C \tag{2.82}$$

$$M_A \frac{l_1}{I_1} + 2M_B \left(\frac{l_1}{I_1} + \frac{l_2}{I_2}\right) + M_C \frac{l_2}{I_2} = -\frac{l_1}{I_1} K_A - \frac{l_2}{I_2} K_C$$

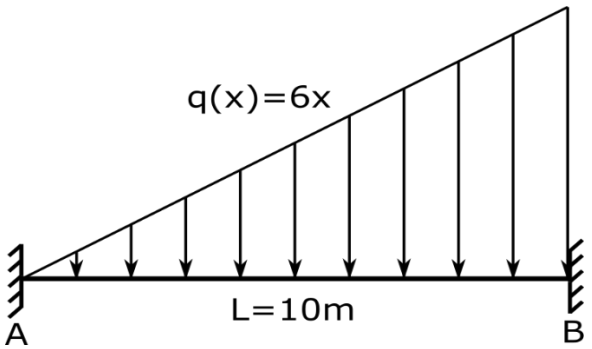
Üç Moment Denkleminin bu hali en bilinen halidir (Savcı, 1988). Pratikte çok kullanılan hali de budur. K_A ve K_C değerleri yüke göre değişir ve tablolardan bulunabilir.

Bu çalışma kapsamında geliştirilen programda kiriş içinde atalet momenti değişken olarak ve parçalı bir polinom halinde ifade edilebilmektedir. Kullanıcı $I(x)$ fonksiyonunu kendisi girer. Bu yüzden Clapeyron'un metodundaki $i(x) = \frac{I_0}{I(x)}$ ifadesi programda kullanılmamaktadır. Bunun yerine $I(x)$ fonksiyonu kullanılır. Programda kullanılan denklem 2.78'in farklı bir halidir ve denklem 2.83'te ifade edilmiştir.

$$M_A \left[\frac{I_1}{l_1}\int_0^{l_1} \frac{dx}{I(x)} - \frac{I_1}{l_1^2}\int_0^{l_1} \frac{x^2 dx}{I(x)}\right] + M_B \left[\frac{I_1}{l_1^2}\int_0^{l_1} \frac{x^2 dx}{I(x)} + \frac{I_2}{l_2^2}\int_0^{l_2} \frac{x'^2 dx'}{I(x')}\right]$$
$$+ M_C \left[\frac{I_2}{l_2}\int_0^{i_2} \frac{x' dx'}{I(x')} - \frac{I_2}{l_2^2}\int_0^{l_2} \frac{x'^2 dx'}{I(x')}\right] \tag{2.83}$$
$$= -\frac{I_1}{l_1}\int_0^{l_1} \frac{M_0(x) x dx}{I(x)} - \frac{I_2}{l_2}\int_0^{l_2} \frac{M_0(x') x' dx'}{I(x')}$$

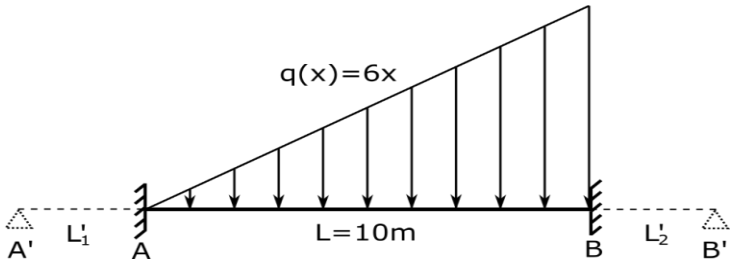
2.3.2 Örnek bir kiriş için yük vektörünün bulunması

(Şekil 2.19)'daki gibi yüklü bir kirişi ele alalım.



Şekil 2.19 : İki ucu ankastre mesnetli kiriş.

Bu gibi problemler endirekt yük vektörünün hesaplanmasında ortaya çıkar. Bütün nodlardaki deplasmanların sıfır olduğu durum o nodlarda ankastre mesnet olmasına karşılık gelir. Üç moment yönteminde ankastre mesnet olduğu durumda (Şekil 2.20)'deki gibi sanal kirişler konularak hesap yapılır.



Şekil 2.20 : Sanal kirişlerin eklendiği kiriş sistemi.

AA' kirişi ve BB' kirişleri sanal kiriş olduğu için kiriş uzunlukları, ve atalet momentleri 0 olacaktır. Ayrıca A' ve B' serbest mesnet olacağı için buradaki momentler de 0 olacaktır. $AA'B$ kiriş sistemi için denklem 2.78 aşağıdaki gibi sadeleşir;

$$M_A\left[\frac{1}{l_2^2}\int_0^{l_2} x'^2 dx'\right] + M_B\left[\frac{1}{l_2}\int_0^{i_2} x' dx' - \frac{1}{l_2^2}\int_0^{l_2} x'^2 dx'\right] = -\frac{1}{l_2}\int_0^{l_2} M_0(x')x' dx' \tag{2.84}$$

Burada önemli bir nokta vardır. $AA'B$ kirişi için Clapeyron'un denklemindeki x için koordinat sisteminin orijini A' noktasıdır ve $+x$ yönü sağ tarafa doğrudur. x' için ise orijin B noktasıdır ve $+x'$ yönü sol tarafa doğrudur. Bu durumda $M_0(x') = (10 - x')^3 - 100(10 - x')$ olarak statik denklemlerinden kolayca bulunabilir. O halde;

$$M_A\left[\frac{1}{10^2}\int_0^{10} x'^2 dx'\right] + M_B\left[\frac{1}{10}\int_0^{10} x' dx' - \frac{1}{10^2}\int_0^{10} x'^2 dx'\right] = -\frac{1}{10}\int_0^{10} ((10 - x')^3 - 100(10 - x'))x' dx' \tag{2.85}$$

$$2M_A + M_B = 700 \tag{2.86}$$

ABB' kiriş sistemi için Clapeyron denklemi aşağıdaki gibi sadeleşir;

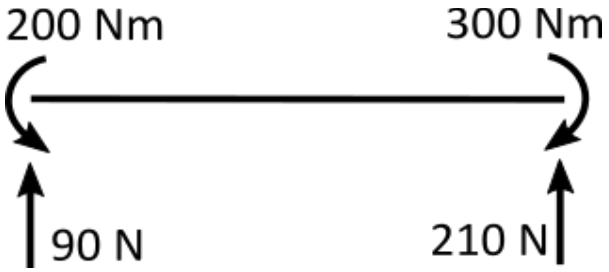
$$M_A\left[\frac{1}{l_2}\int_0^{l_2} xdx - \frac{1}{l_2^2}\int_0^{l_2} x^2dx\right] + M_B\left[\frac{1}{l_2^2}\int_0^{l_2} x^2dx\right] = -\frac{1}{l_2}\int_0^{l_2} M_0(x)xdx \tag{2.87}$$

Burada x için koordinat sisteminin orijini A noktasıdır ve $+x$ yönü sağ tarafa doğrudur. x' için ise orijin B' noktasıdır ve $+x'$ yönü sol tarafa doğrudur. Bu durumda $M_0(x) = x^3 - 100x$ olarak bulunabilir.

$$M_A\left[\frac{1}{l_2}\int_0^{l_2} xdx - \frac{1}{l_2^2}\int_0^{l_2} x^2dx\right] + M_B\left[\frac{1}{l_2^2}\int_0^{l_2} x^2dx\right] = -\frac{1}{l_2}\int_0^{l_2} M_0(x)xdx \tag{2.88}$$

$$M_A + 2M_B = 800 \tag{2.89}$$

Denklem 2.86 ve denklem 2.87 çözülürse $M_A = 200\ Nm$ ve $M_B = 300\ Nm$ bulunur. Bulunan bu değerler üç moment denklemi işaret kabulüne göredir. Mukavemetten mesnet tepkileri de hesaplandığında kirişteki mesnet tepkileri (Şekil 2.21)'deki gibi bulunur.



Şekil 2.21 : Kirişteki mesnet tepkileri.

Kirişte eksenel kuvvet yoktur. Endirekt yük vektörünün ilk elemanı 1 numaralı serbestlik derecesi yönündeki kuvvettir. 2. Elemanı ise 2 numaralı serbestlik derecesi yönündeki kuvvet ve diğer elemanları kendilerine karşılık gelen serbestlik derecesi yönündeki kuvvet ya da momenttir. Elemanın endirekt yük vektörü denklem 2.90'daki gibi olur.

$$f = \begin{Bmatrix} 0 \\ 90 \\ 200 \\ 0 \\ 210 \\ -300 \end{Bmatrix} \tag{2.90}$$

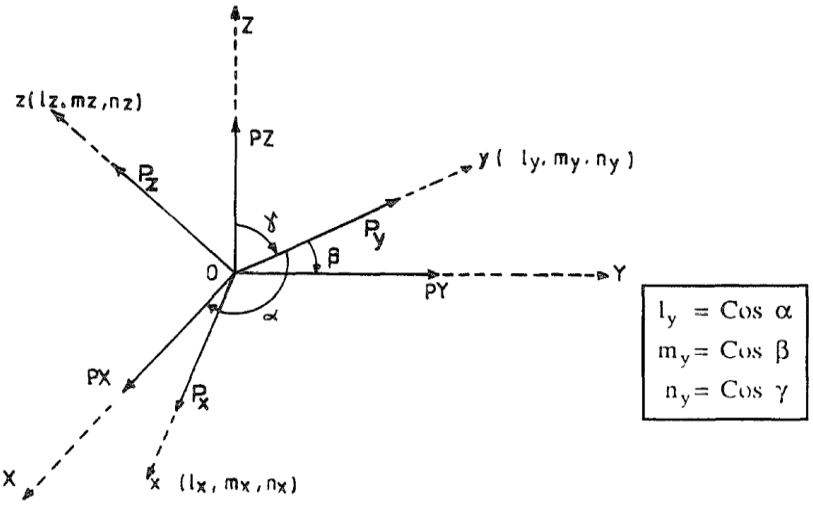
Vektörün son elemanının negatif olması kirişin sağ ucundaki momentin yönünün matris deplasman yöntemindeki dönme yönün tersinde dönme olmasından dolayıdır. Kirişin nodlarına herhangi bir direkt kuvvet etkimediği için elemanın direkt yük vektörünün bütün elemanları sıfırdır.

2.4 Müşterek Sistem

Metotta eleman matrisleri ve yük vektörleri tek bir müşterek denklem takımında toplanır. Eleman matrisleri özel bir yöntemle birleştirilerek müşterek katılık matrisini oluşturulur. Elemanların direkt ve endirekt yük vektörleri de yine özel bir yöntemle birleştirilerek müşterek yük vektörü oluşturulur. Oluşan müşterek matris sistemi çözülerek uç deplasmanları bulunur. Bulunan müşterek deplasmanlar tekrar elemanların matris denklemine entegre edilerek elemanlardaki bilinmeyen deplasmanlar bulunmuş olur.

2.4.1 Kiriş eksenlerinin transformasyonu

(Şekil 2.22)'de bir p vektörünün XYZ ve xyz olmak üzere iki eksen takımlarında iz düşümü gösterilmiştir.



Şekil 2.22 : Bir noktada eksen transformasyonu (Karaduman, 1993).

XYZ eksen takımındaki iz düşümleri kullanılarak xyz eksen takımlarındaki iz düşümler bulunmaya çalışılırsa;

$$p_x = p_X l_x + p_Y m_x + p_Z p n_x$$
$$p_y = p_X l_y + p_Y m_y + p_Z p n_y \quad (2.91)$$
$$p_z = p_X l_z + p_Y m_z + p_Z p n_z$$

yazılabilir. Yukarıdaki denklemleri matris notasyonunda yazarsak;

$$\begin{Bmatrix} p_x \\ p_y \\ p_z \end{Bmatrix} = \begin{bmatrix} l_x & m_x & n_x \\ l_y & m_y & n_y \\ l_z & m_z & n_z \end{bmatrix} \begin{Bmatrix} p_X \\ p_Y \\ p_Z \end{Bmatrix} \quad (2.92)$$

l, m, n'lerden oluşan bu matrise doğrultu kosinüsleri matrisi denir. $[t]$ ile gösterilir.

$$\{p\}_{xyz} = [t]\{p\}_{XYZ} \quad (2.93)$$

Denklem 2.93'e ortogonal transformasyon denklemi denir. Düzlem çerçevelerde x ekseni ortak eksene paralel, y ekseni de çubuk eksenine diktir. Bu durumda $[t]$ matrisi aşağıdaki gibi olur:

$$[t] = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.94)$$

Kiriş elemanının i ucunun koordinatları (x_i, y_i) ve j ucunun koordinatları (x_j, y_j) olarak kabul edilirse:

$$L = \sqrt{\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2} \quad (2.95)$$

$$\cos\beta = \frac{\left(x_j - x_i\right)}{L} \quad (2.96)$$

$$\sin\beta = \frac{\left(y_j - y_i\right)}{L} \quad (2.97)$$

Düzlem çerçevelerde 6 serbestlik derecesi olduğu için bu matrisi $6x6$ boyutuna aşağıdaki gibi getirmek gerekir:

$$[T] = \begin{bmatrix} [t] & 0 \\ 0 & [t] \end{bmatrix}_{6x6} \tag{2.98}$$

Bu durumda eleman bazındaki direkt, endirekt yük vektörleri ve deplasman vektörü aşağıdaki gibi global eksen takımından eleman eksen takımına yazılabilir:

$$\{p\}_{xyz} = [T]\{p\}_{XYZ} \tag{2.99}$$

$$\{d\}_{xyz} = [T]\{d\}_{XYZ} \tag{2.100}$$

$$\{f\}_{xyz} = [T]\{f\}_{XYZ} \tag{2.101}$$

Transformasyon matrisinin $[T]^{-1} = [T]^T$ özelliği kullanılıp aşağıdaki gibi ters transformasyon yapılabilir:

$$\{p\}_{XYZ} = [T]^T\{p\}_{xyz} \tag{2.102}$$

$$\{d\}_{XYZ} = [T]^T\{d\}_{xyz} \tag{2.103}$$

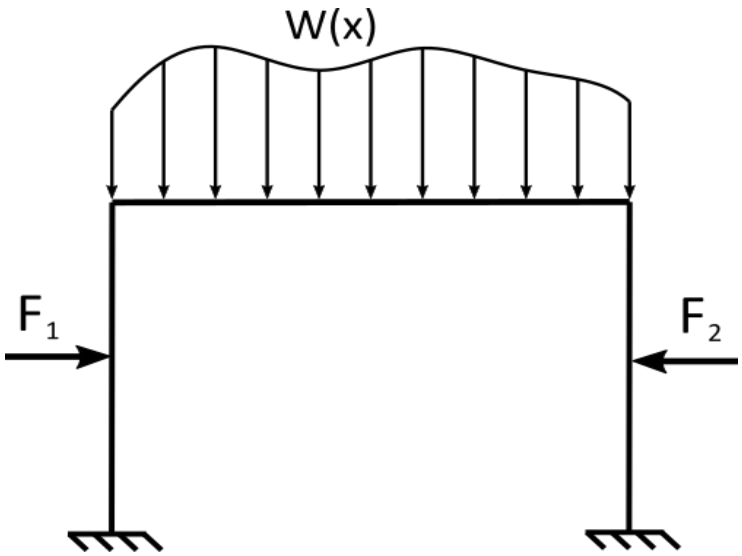
$$\{f\}_{XYZ} = [T]^T\{f\}_{xyz} \tag{2.104}$$

Katılık matrisinin müşterek eksene transformasyonu aşağıdaki gibi yapılır:

$$[k]_{XYZ} = [T]^T[k]_{xyz}[T] \tag{2.105}$$

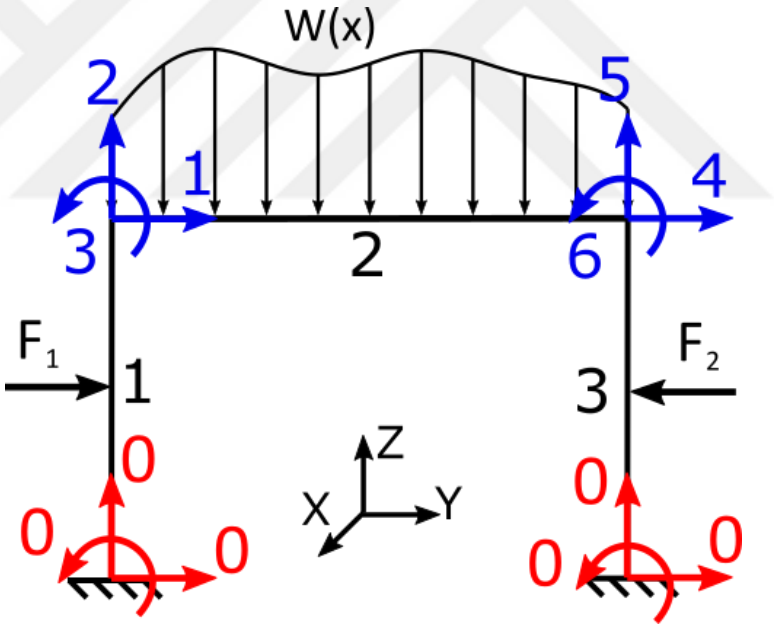
2.4.2 Müşterek sistemin elde edilmesi ve çözümü

(Şekil 2.23)'teki enine çerçeve örneği ele alınsın. Çerçeve üç ayrı kirişten oluşmaktadır ve tekil ve yayılı yüklerin etkisi altındadır.



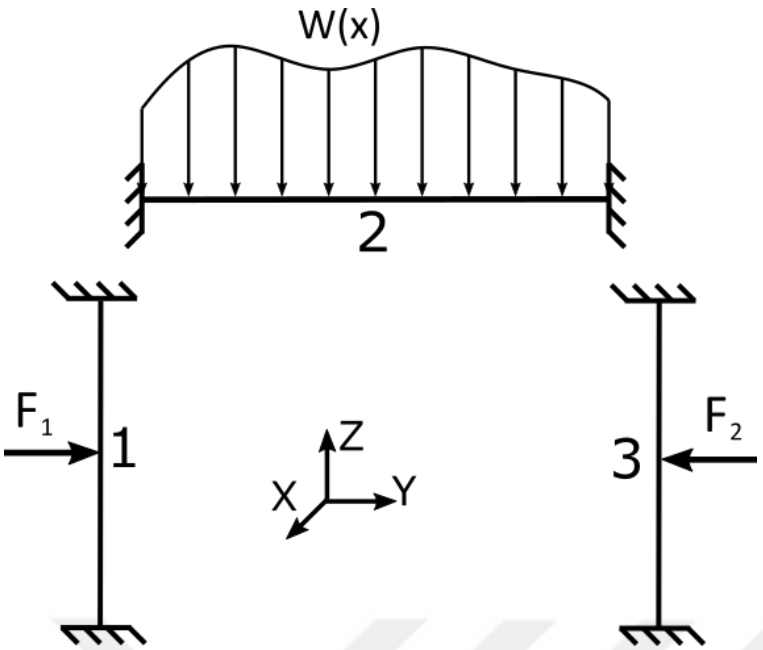
Şekil 2.23 : İki boyutlu enine çerçeve örneği.

Öncelikle kirişler numaralandırılıp sistemin serbestlik derecesi belirlenir. Sistem 6 serbestlik derecelidir. Serbestlik dereceleri (Şekil 2.24)'te gösterilmiştir.



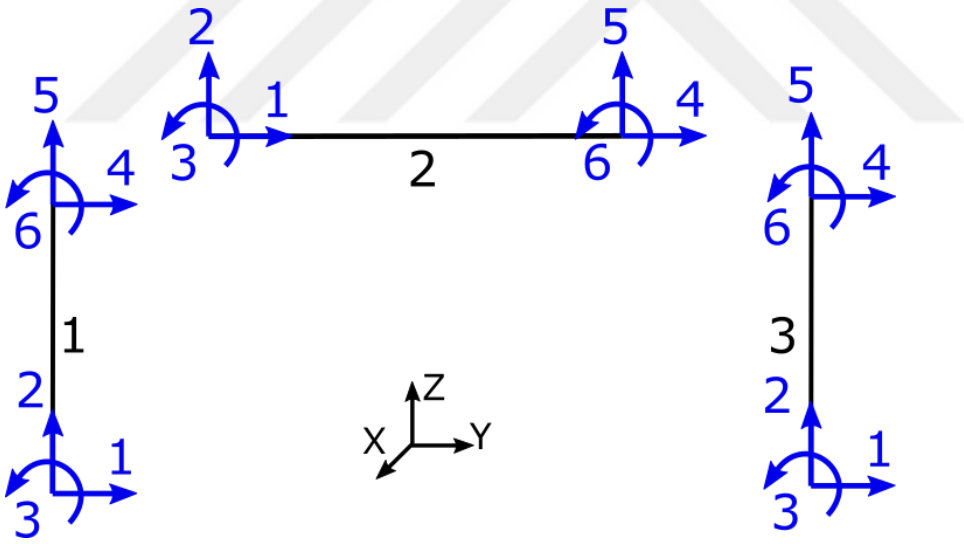
Şekil 2.24 : İki boyutlu enine çerçevenin serbestlik dereceleri.

Ankastre mesnetlerde her üç serbestlik dereceleri yönünde deplasmanlar sıfır olacağı için serbestlik dereceleri yoktur. 6 tane serbestlik derecesi olduğundan dolayı global matris $6x6$ boyutunda olacaktır. Bir sonraki adım çerçevedeki kirişlerin (Şekil 2.25)'teki halinin direkt ve endirekt yük vektörlerini bulmaktır.



Şekil 2.25 : İki boyutlu enine çerçevenin endirekt yüklerinin bulunması.

Çerçeve elemanlarının müşterek eksenlerdeki serbestlik dereceleri (Şekil 2.26)'daki gibi olacaktır.



Şekil 2.26 : İki boyutlu enine çerçevedeki elemanların serbestlik dereceleri.

Sisteme direkt yük etkimediği için tüm elemanların direkt yük vektörleri sıfır olacaktır. Üç Moment Yöntemi kullanılarak endirekt yük vektörleri bulunabilir. 1 numaralı eleman için $\cos\beta = 0$, $\sin\beta = 1$, 2 numaralı eleman için $\cos\beta = 1$, $\sin\beta = 0$ ve 3 numaralı eleman için $\cos\beta = 0$, $\sin\beta = 1$ olacaktır. Dolayısıyla her üç eleman için transformasyon matrisleri bellidir. Global eksen takımına göre

$\{f\}_{XYZ} = [T]^T\{f\}_{xyz}$ denklemi kullanılarak endirekt yük vektörlerinin aşağıdaki gibi elde edildiği varsayılsın:

$$f^1_{XYZ} = \begin{Bmatrix} f^1_1 \\ f^1_2 \\ f^1_3 \\ f^1_4 \\ f^1_5 \\ f^1_6 \end{Bmatrix}, \qquad f^2_{XYZ} = \begin{Bmatrix} f^2_1 \\ f^2_2 \\ f^2_3 \\ f^2_4 \\ f^2_5 \\ f^2_6 \end{Bmatrix}, \qquad f^3_{XYZ} = \begin{Bmatrix} f^3_1 \\ f^3_2 \\ f^3_3 \\ f^3_4 \\ f^3_5 \\ f^3_6 \end{Bmatrix} \tag{2.106}$$

Global eksen takımına göre eleman katılık matrislerinin $[k]_{XYZ} = [T]^T[k]_{xyz}[T]$ denklemiyle aşağıdaki gibi elde edildiği varsayılsın:

$$k^1{}_{XYZ} = \begin{bmatrix} k^1_{11} & k^1_{12} & k^1_{13} & k^1_{14} & k^1_{15} & k^1_{16} \\ k^1_{21} & k^1_{22} & k^1_{23} & k^1_{24} & k^1_{25} & k^1_{26} \\ k^1_{31} & k^1_{32} & k^1_{33} & k^1_{34} & k^1_{35} & k^1_{36} \\ k^1_{41} & k^1_{42} & k^1_{43} & k^1_{44} & k^1_{45} & k^1_{46} \\ k^1_{51} & k^1_{52} & k^1_{53} & k^1_{54} & k^1_{55} & k^1_{56} \\ k^1_{61} & k^1_{62} & k^1_{63} & k^1_{64} & k^1_{65} & k^1_{66} \end{bmatrix} \tag{2.107}$$

$$k^2{}_{XYZ} = \begin{bmatrix} k^2_{11} & k^2_{12} & k^2_{13} & k^2_{14} & k^2_{15} & k^2_{16} \\ k^2_{21} & k^2_{22} & k^2_{23} & k^2_{24} & k^2_{25} & k^2_{26} \\ k^2_{31} & k^1_{32} & k^2_{33} & k^2_{34} & k^2_{35} & k^2_{36} \\ k^2_{41} & k^2_{42} & k^2_{43} & k^2_{44} & k^2_{45} & k^2_{46} \\ k^2_{51} & k^2_{52} & k^2_{53} & k^2_{54} & k^2_{55} & k^2_{56} \\ k^2_{61} & k^2_{62} & k^2_{63} & k^2_{64} & k^2_{65} & k^2_{66} \end{bmatrix} \tag{2.108}$$

$$k^3{}_{XYZ} = \begin{bmatrix} k^3_{11} & k^3_{12} & k^3_{13} & k^3_{14} & k^3_{15} & k^3_{16} \\ k^3_{21} & k^3_{22} & k^3_{23} & k^3_{24} & k^3_{25} & k^3_{26} \\ k^3_{31} & k^3_{32} & k^3_{33} & k^3_{34} & k^3_{35} & k^3_{36} \\ k^3_{41} & k^3_{42} & k^3_{43} & k^3_{44} & k^3_{45} & k^3_{46} \\ k^3_{51} & k^3_{52} & k^3_{53} & k^3_{54} & k^3_{55} & k^3_{56} \\ k^3_{61} & k^3_{62} & k^3_{63} & k^3_{64} & k^3_{65} & k^3_{66} \end{bmatrix} \tag{2.109}$$

(Şekil 2.25)'de kiriş elemanlarının global eksen takımına göre serbestlik dereceleri verilmiştir. Her bir kiriş için, serbestlik derecelerinin global serbestlik derecelerindeki karşılığı tablo halinde yazılabilir. Çizelge 2.1'de bu tablo gösterilmiştir.

Çizelge 2.1 : Düzlem çerçeve için kod numaraları tablosu.

Kiriş No	1	Eleman	1	2	3	4	5	6
		Sistem	0	0	0	1	2	3
	2	Eleman	1	2	3	4	5	6
		Sistem	1	2	3	4	5	6
	3	Eleman	1	2	3	4	5	6
		Sistem	4	5	6	0	0	0

Bu yönteme kod numaraları yöntemi denir. Global matris bu tabloya bakılarak oluşturulur. Elemanların serbestlik dereceleri indekslerinin global serbestlik derecelerinde hangi indekslere karşılık geldiği bulunarak global katılık matrisinin elemanları doldurulur. Örneğin 1 numaralı elemanın k_{44}^1 elemanı global matriste k_{11}'e karşılık gelmektedir. k_{45}^1 elemanı k_{12}'ye karşılık gelir. k_{46}^1 ise k_{13}'e karşılık gelir. 2 numaralı elemanın k_{11}^2 elemanı global matriste k_{11}'e, k_{12}^2 elemanı global matriste k_{12}'ye karşılık gelir. Bu mantık ile bütün elemanların indisleri tarandığında global matris oluşturulmuş olur. Elemanların katılık matrisi elemanları global matriste aynı elemana karşılık geliyorsa katılık elemanlarının toplamı global matrisin o elemanını oluşturur. Global matris sonuç olarak denklem 2.108'deki gibi elde edilir.

$$k = \begin{bmatrix} k_{44}^1 + k_{11}^2 & k_{45}^1 + k_{12}^2 & k_{46}^1 + k_{13}^2 & k_{14}^2 & k_{15}^2 & k_{16}^2 \\ k_{54}^1 + k_{21}^2 & k_{55}^1 + k_{22}^2 & k_{56}^1 + k_{23}^2 & k_{24}^2 & k_{25}^2 & k_{26}^2 \\ k_{64}^1 + k_{31}^2 & k_{65}^1 + k_{32}^2 & k_{66}^1 + k_{33}^2 & k_{34}^2 & k_{35}^2 & k_{36}^2 \\ k_{41}^2 & k_{41}^2 & k_{43}^2 & k_{11}^3 + k_{44}^2 & k_{12}^3 + k_{45}^2 & k_{13}^3 + k_{46}^2 \\ k_{51}^2 & k_{51}^2 & k_{53}^2 & k_{21}^3 + k_{54}^2 & k_{22}^3 + k_{55}^2 & k_{23}^3 + k_{56}^2 \\ k_{61}^2 & k_{61}^2 & k_{63}^2 & k_{31}^3 + k_{64}^2 & k_{32}^3 + k_{65}^2 & k_{33}^3 + k_{66}^2 \end{bmatrix} \quad (2.110)$$

Kod numaraları yöntemi yük vektörlerine de uygulanır. Denklem 2.111'de yük vektörlerinin elemanlarının global yük vektöründe hangi numaraya karşılık geldiği gösterilmiştir.

$$f_{XYZ}^1 = \begin{Bmatrix} f_1^1 \\ f_2^1 \\ f_3^1 \\ f_4^1 \\ f_5^1 \\ f_6^1 \end{Bmatrix} \begin{matrix} \rightarrow 0 \\ \rightarrow 0 \\ \rightarrow 0 \\ \rightarrow 1 \\ \rightarrow 2 \\ \rightarrow 3 \end{matrix}, \quad f_{XYZ}^2 = \begin{Bmatrix} f_1^2 \\ f_2^2 \\ f_3^2 \\ f_4^2 \\ f_5^2 \\ f_6^2 \end{Bmatrix} \begin{matrix} \rightarrow 1 \\ \rightarrow 2 \\ \rightarrow 3 \\ \rightarrow 4 \\ \rightarrow 5 \\ \rightarrow 6 \end{matrix}, \quad f_{XYZ}^3 = \begin{Bmatrix} f_1^3 \\ f_2^3 \\ f_3^3 \\ f_4^3 \\ f_5^3 \\ f_6^3 \end{Bmatrix} \begin{matrix} \rightarrow 0 \\ \rightarrow 0 \\ \rightarrow 0 \\ \rightarrow 4 \\ \rightarrow 5 \\ \rightarrow 6 \end{matrix} \quad (2.111)$$

Kod numaralarındaki aynı mantık ile global endirekt yük vektörü aşağıdaki gibi elde edilir:

$$f = \begin{Bmatrix} f_1^1 + f_1^2 \\ f_2^1 + f_2^2 \\ f_3^1 + f_3^2 \\ f_4^2 + f_4^3 \\ f_5^2 + f_5^3 \\ f_6^2 + f_6^3 \end{Bmatrix} \tag{2.112}$$

Denklem 2.3 tekrar yazılsın;

$$\{P\} = [k]\{d\} + \{f\} \tag{2.113}$$

$\{f\}$ vektörü karşı tarafa atılırsa;

$$\{P\} - \{f\} = [k]\{d\} \tag{2.114}$$

olur. Burada tek bilinmeyen $\{d\}$ vektörüdür. Sistem Cramer Kuralı veya Gauss Eliminasyon Yöntemi gibi lineer denklem çözücü herhangi bir yöntemle çözülerek bilinmeyen $\{d\}$ vektörü bulunabilir. Sonuçta denklem 2.115'teki gibi bir vektör elde edilir. Vektörün boyutu müşterek sistemin serbestlik derecesi sayısıyla aynı olacaktır.

$$d = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{Bmatrix} \tag{2.115}$$

Bundan sonra elde edilen bu deplasmanlar eleman denklemlerinde kod numaraları mantığıyla yerine konularak elemanın serbestlik dereceleri yönündeki uç kuvvetleri bulunur.

$$\{P^1\}_{XYZ} = [k_{XYZ}^1] \begin{Bmatrix} 0 \\ 0 \\ 0 \\ d_1 \\ d_2 \\ d_3 \end{Bmatrix} + \{f_{XYZ}^1\} \tag{2.116}$$

$$\{P^2\}_{XYZ} = [k^2_{XYZ}] \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{Bmatrix} + \{f^2_{XYZ}\} \quad (2.117)$$

$$\{P^3\}_{XYZ} = [k^3_{XYZ}] \begin{Bmatrix} 0 \\ 0 \\ 0 \\ d_4 \\ d_5 \\ d_6 \end{Bmatrix} + \{f^3_{XYZ}\} \quad (2.118)$$

Daha sonra elemanların bulunan bu müşterek yük vektörlerinden lokal yük vektörlerine geçilir.

$$\{P^1\}_{xyz} = [T]\{P^1\}_{XYZ} \quad (2.119)$$

$$\{P^2\}_{xyz} = [T]\{P^2\}_{XYZ} \quad (2.120)$$

$$\{P^3\}_{xyz} = [T]\{P^3\}_{XYZ} \quad (2.121)$$

Bu aşamada kiriş hakkında her şey bilinmektedir. Bu uç kuvvetler ve momentlerden artık kesit tesir diyagramlar ve sehim eğrileri çizilebilir, gerilme analizleri yapılabilir.

3. PROGRAMIN GELİŞTİRİLMESİ

Bu bölümde bitirme çalışmasında gemi enine çerçevelerinin yapısal analizini icra etmek üzere geliştirilen bilgisayar programından bahsedilecektir.

3.1 Programın Altyapısı

Programda nesne yönelimli programlama disiplinleri yoğun olarak kullanılmıştır. Nesne yönelimli programlama class denilen sınıf yapılarıyla programlamadır. Her sınıf kendi içinde değişkenler ve fonksiyonlar barındırır. Sınıflar belirli parametrelerle ya da parametresiz olarak başlatılabilir. Erişim düzenleyici anahtar kelimelerle sınıf içindeki değişkenler ve fonksiyonlar dışarıdan erişime açılabilir ya da kapatılabilir. Bir sınıf başka bir sınıfı baz olarak alabilir. Bütün bu nesne yönelimli programlama paradigmaları program ne kadar fazla kod içerse de, ne kadar karmaşık olsa da bir düzen sağlar ve sistematik bir yapı oluşturmaya olanak tanır. Ayrıca modülerliğe de oldukça katkıda bulunur.

Program geliştirilirken C# programlama dili kullanılmıştır. C#, 2000 yılında Microsoft tarafından geliştirilmiş nesne yönelimli, genel maksatlı bir programlama dilidir (C# Language Specification, 2006). C#, Microsoft Windows platformlarında .Net Framework uygulama geliştirme kütüphanesini kullanarak çalışır. C++ üzerine inşa edilmiş dil Java'dan etkilenmiştir. C#, yüksek seviyeli bir dildir. C ve C++ gibi düşük seviyeli dillerden farklı olarak hafıza yönetimlerini dil kendisi otomatik olarak yapar. Dolayısıyla C++ ve C dibi makine koduna yakın programlama dillerine göre çok daha hızlı yazılım geliştirilmesine olanak tanır. Buna karşılık performansı bu sayılan düşük seviyeli dillerden daha düşüktür.

Uygulamada platform olarak WPF (Windows Presentation Foundation) platformu kullanılmıştır. WPF, C# ile görsel program yazmaya olanak tanıyan bir platformdur. Microsoft tarafından 2008 yılında kullanıma sunulmuştur. Eski nesil form uygulamalarının aksine WPF vektör tabanlıdır ve yazılan program değişen ekran boyutlarına ve yoğunluklarına otomatik adapte olur. WPF ayrıca programın görsel ara yüzünün XAML (Extensible Application Markup Language) ile geliştirilmesine

izin verir. XAML, Microsoft tarafından XML'den türetilmiş ve programların ara yüz tasarımında kullanılan bir biçimlendirme dilidir. Programların görsel olarak tasarımında oldukça kolaylık sağlar.

Program .Net Framework uygulama geliştirme kütüphanesini kullanmaktadır. Programın çalışması için .Net Framework kütüphanelerinin hedef bilgisayarda yüklü olması gerekir. .Net Framework, programların kullandığı fonksiyonları barındıran bir platformdur. .Net Framework desteklenmediği için bu program Windows XP'den önce çıkmış Windows işletim sistemlerinde çalışmaz. Ayrıca program sadece Windows işletim sistemlerinde çalışmaktadır.

3.2 Programda Kullanılan Kütüphaneler

Bir önceki bölümde bahsedilen temel platformlardan başka bu bilgisayar yazılımda dışarıdan alınmış bazı kütüphane ve kod parçaları kullanılmıştır. Bu bölümde bunlardan bahsedilecektir. Programda kiriş ve mesnetlerin oluşturulup grafik çizimlerinin yapıldığı grafik çizme elementi olan Canvas sınıfı kullanılmıştır. Bu sınıfa ek olarak yine bu sınıfın üzerine inşa edilmiş ve yakınlaştırma ve taşıma gibi işlevleri icra eden bir görsel sınıf kullanılmıştır (Davis, 2011).

Kirişlerin atalet momentleri, yayılı yükler gibi kiriş özelliklerinin değişken olarak tanımlanmasına imkan tanıyan bir polinom sınıfı kullanılmıştır (Alikhani, 2010). Bu sınıfın içine kirişler, mesnetler ve yükler ile ilgili olan fonksiyonlar geliştirilip eklenmiştir. Yine bu polinom sınıfın üzerine parçalı fonksiyon polinomu sınıfı geliştirilmiştir. Kirişlerde kullanılan sınıf da budur. EK A'da *Poly* sınıfı gösterilmiştir.

Yayılı yükleri çizmek, moment, kuvvet, gerilme ve atalet momenti dağılımlarının çizildiği Cardinal Spline adında bir eğri sınıfı kullanılmıştır (Floris, 2009). Bu sınıf noktaları verilen eğriyi görsel olarak çizen ve Kübik Hermit Spline'dan türetilmiş bir şekil sınıfıdır.

3.3 Programın Akışı

Program ilk olarak görsel ara yüzün dilini ayarlayarak başlar. Eğer kullanıcı bir dili seçtiyse o dildeki çevirmeleri yükler. Eğer kullanıcı hiçbir dili özel olarak seçmediyse program işletim sisteminin diline göre dili ayarlar. Eğer işletim

sisteminin dili programın çeviri dilleri veri tabanında mevcut değil ise program İngilizce dilinde açılır. Şu an için programda İngilizce ve Türkçe dilleri desteklenmektedir.

Kullanıcı çizim ortamına kiriş ve mesnet ekledikçe program bu kiriş ve mesnetlerin birbirine bağlantılarını yapar. Kullanıcı sistemi istediği gibi oluşturup çözüm butonu basınca programda bir pencere belirir ve çözüm ilerleme çubuğu görünür. Programda çözüm aşamasındaki işlemler ana iş parçacığından (main thread) ayrı bir iş parçacığında gerçekleşir. Böylece çözüm esnasında programın donması engellenmiş olur. Ayrıca kullanıcı ayarlar menüsünden çözüm sırasında tek veya çok iş parçacığının kullanılmasını seçebilir. Bu durumda program çalıştığı bilgisayarın çekirdek sayısı kadar iş parçacığı üretecektir. Bu iş parçacıkları aynı anda farklı kirişlerde işlem yaparak tek iş parçacığına göre daha hızlı çözüme ulaşılacaktır. Bu durum karmaşık bir gemi enine çerçevesinde çözüm süresini oldukça kısaltır.

Burada çözüm üç başlık altında anlatılacaktır. Çözüm öncesi işlemler, çözümdeki işlemler ve çözüm sonrası işlemler. Bu sınıflandırma, müşterek sistem matrisine göre yapılmıştır. müşterek sistemin çözümünden önceki bütün işlemler çözüm öncesi işlemler başlığı altında incelenecektir. Müşterek sistem çözüldükten sonraki işlemler de çözüm sonrasındaki işlemler başlığı altında incelenecektir.

3.3.1 Çözüm öncesi işlemler

Çözüm işlemi başladıktan sonra ilk olarak her kiriş için $Calculate$ fonksiyonu çalıştırılır. Kiriş sınıfı olan $Beam$ EK B'de verilmiştir. Kirişteki $Calculate$ fonksiyonunda ilk olarak $findconcentratedsupportforces$ ve $finddistributedsupportforces$ fonksiyonları çağırılır. Bu fonksiyonlar kiriş üzerindeki tekil ve yayılı yükler durumdaki kirişin mesnet tepkilerini bulur.

Daha sonra $findconcentratedzeroforce$ ve $finddistributedzeroforce$ fonksiyonları çağırılarak tekil ve yayılı yükler için ayrı olarak yine kirişin her iki ucunun serbest mesnet olduğu durumda kesme kuvveti dağılımlarını bulunur. Tabii bu kuvvet dağılımları parçalı polinom sınıfından (EK A'daki $PiecewisePoly$ sınıfı) oluşacaktır. Daha sonra bu dağılımlar süper pozisyon ilkesiyle toplanarak $F_0(x)$ arçalı polinomu elde edilir. Bu polinom kirişin her iki ucunun serbest mesnet olması halinde etkiyen yükün oluşturduğu kesme kuvveti dağılımıdır.

Sonra $findzeromoment$ fonksiyonu çağırılarak $F_0(x)$ polinomu integre edilip $M_0(x)$ bulunur. $M_0(x)$, daha önce Clapeyron yönteminde geçen serbest mesnetli haldeki moment dağılımıdır. Burada yapılan integrasyon analitiktir.

Sonrasında $canbesolvedanalytically$ fonksiyonu çağırılarak kesit tesir dağılımını elde etmek için analitik çözümün yapılıp yapılamayacağı kontrol edilir. Analitik çözüm yapılabilmesi için kirişin atalet dağılımı parçalı polinomu sabit olmalı ve tek bir polinom içermelidir. Böylece Üç Moment Denkleminde paydadaki $I(x)$ fonksiyonu sabit olacak ve integral sadece $M(x)$ polinomuna bağlı olacaktır. Burada geliştirilmiş olan polinom sınıfı bu polinomu analitik olarak integre edebilir. Eğer atalet fonksiyonu sabit değilse Simpson Yöntemiyle integraller nümerik olarak hesaplanır.

$mdsupportcases$ fonksiyonu çağırılarak üç moment denklemi çözülür. Nümerik integraller Simpson Metodu ile alınır. Burada kullanılan Simpson sınıfı EK A'daki $SimpsonsFirstIntegrator$ adlı sınıftır. Kirişin ucundaki ankastrelik momentler bu aşamada elde edilmiş olur.

Sonra $findfixedendmomentclapeyron$ fonksiyonu çağırılarak denklem 2.75 kullanılıp nihai moment dağılımı bulunur. Bu işlemden sonra $updateforces$ fonksiyonu çağırılır ve nihai kesme kuvveti dağılımı elde edilir.

Bu aşamadan sonra kirişlerdeki hesaplar bitmiş olur. $createbasestiffnescoefficients$ çağırılarak katılık matrisini oluşturmak için gereken temel rijitlik katsayıları hesaplanır. Eğer atalet momenti dağılımı ve alan dağılımı sabitse integralleri almaya gerek yoktur. Eğer değilse yine Simpson Metodu ile nümerik integraller alınır.

Daha sonra $createstiffnessmatrix$ fonksiyonu çağırılıp katılık matrisi hesaplanır. Önce kiriş açısı sıfır iken, eleman eksenlerinde tanımlı katılık matrisi elde edilir. Sonrasında transformasyon matrisiyle denklem 2.105'teki gibi çarpılarak kirişin nihai katılık matrisi elde edilir. Bu işlemler yapılırken EK A'daki $transpose$ ve $matrixmultiply$ fonksiyonlarından yararlanılır.

Son olarak ise $createforcevector$ fonksiyonu çağırılır. Burada direkt ve endirekt kuvvet vektörleri elde edilip denklem 2.114'teki formu elde etmek için birbirinden çıkarılıp sistem çözümünde kullanılacak nihai kuvvet vektörü elde edilir.

Bütün bu işlemler sistemdeki bütün kirişler için yapıldıktan sonra müşterek sistem çözümüne geçilir.

3.3.2 Çözümdeki işlemler

Bütün kirişlerin için bir önceki bölümdeki hesaplar yapıldıktan sonra EK C'deki $MDSolver$ sınıfının $Calculate$ fonksiyonu çağırılır. Bu sınıf müşterek sistemi oluşturmak, çözmek ve kirişlere gerekli deplasmanları iletmekle görevlidir.

$Calculate$ fonksiyonu ilk olarak $createdofpairs$ adlı fonksiyonu çağırır. Öncelikle EK C'deki DOF adlı sınıftan oluşan bir liste oluşturulur. Her bir DOF sınıfı, müşterek sistemindeki bir serbestlik derecesine karşılık gelir. Sınıfta serbestlik derecesinin tipi ve o serbestlik derecesinin hangi kirişlerin serbestlik derecelerine karşılık geldiği bilgileri tutulur.

Sonrasında $createglobalstiffnessmatrix$ fonksiyonu çağırılıp daha önce elde edilen listeden yararlanılarak müşterek katılık matrisi oluşturulur. Katılık matrisinin boyutları listenin uzunluğu kadardır. Listedeki serbestlik dereceleri taranarak her bir serbestlik derecesine karşılık gelen kirişlerin ilgili katılık matrisi elemanları toplanıp matris oluşturulur.

Daha sonra $createglobalforcevector$ fonksiyonu çağırılarak katılık matrisindeki gibi serbestlik derecesi listesinden yararlanılarak müşterek kuvvet vektörü oluşturulur.

Sonra $solvethesystem$ fonnksiyonu çağırılarak müşterek matris sistemi $LinearEquationSolver$ fonksiyonuyla sistem çözülür ve müşterek deplasmanlar bulunur. $LinearEquationSolver$ fonksiyonu Gauss Eliminasyon yöntemiyle çözüm yapar. Matris değerleri $double$ veri yapısıyla tutulduğu için herhangi bir pivotlama tekniği uygulamasına gerek görülmemiştir çünkü $double$ veri yapısı kesme hatalarına karşı 15 ile 17 basamak arasında hassasiyet sağlar (Microsoft, 2018).

Son olarak $obtainbeamdisplacements$ fonksiyonu çağırılarak elde edilen müşterek deplasman vektöründen yine serbestlik derecesi listesindeki bilgiler kullanılarak kirişlere gerekli olan deplasman elemanları verilir. Bu işlem kiriş sınıfının $UpdateDirectForceVector$ fonksiyonu kiriş için oluşturulan eleman deplasman vektörü parametresiyle çağırılarak yapılır. Bu fonksiyonda denklem 2.116'daki işlem yapılır. Kirişin eleman katılık matrisi, parametre olarak verilen deplasman vektörüyle

EK A'daki $DotProduct$ fonksiyonu kullanılarak çarpılır ve elde edilen sonuç yine EK A'daki $AddVector$ fonksiyonu kullanılarak endirekt yük vektörüyle toplanır. Sonuç olarak kirişin nihai kuvvet vektörü çözüme göre güncellenmiş olur.

3.3.3 Çözüm sonrası işlemler

Kirişlerin yük vektörleri güncellendikten sonra bütün kirişlerin $PostProcessUpdate$ fonksiyonu çağırılır.

Bu fonksiyon önce $obtainlocalforces$ fonksiyonunu çağırır. Burada fonksiyon denklem 2.102'deki ifadeyi kullanarak güncellenmiş, müşterek eksende ifade edilmiş eleman nihai yük vektörünü kirişin transformasyon matrisiyle çarparak lokal eksenlerdeki karşılığını bulur.

Daha sonra $updatemoments$ fonksiyonu çağırılır. Lokal eksene göre hesaplanmış olan bu yük vektörünün 3. ve 6. Elemanlarını alır. Bu elemanlar sırasıyla kirişin sol ve sağ uçlarındaki momentlere karşılık gelir. Sonra yine denklem 2.75'teki ifade kullanılarak kirişin nihai moment dağılımı oluşturulur.

Sonra $updateforces$ fonksiyonu çağırılır. Elde edilen nihai moment dağılımının türevi alınarak kirişin kesme kuvveti dağılımı oluşturulur.

Sonrasında $updateaxialforces$ fonksiyonu çağırılarak eksenel kuvvet dağılımı lokal eksene göre hesaplanmış yük vektörünün 1. ve 4. Elemanı kullanılarak elde edilir.

Son olarak eğer kullanıcı gerilme analizi yapmak istediyse $updatestresses$ fonksiyonu çağırılarak denklem 3.1 kullanılarak gerilme dağılımı oluşturulur.

$$\sigma(x) = \frac{M(x)y(x)}{I(x)} \tag{3.1}$$

3.4 Sistem Gereksinimleri

Programın çalışması için önerilen sistem gereksinimleri aşağıdaki gibi sıralanabilir;

- Windows XP veya daha sonrasında çıkmış Windows işletim sistemleri
- .Net 4.5 Framework veya üzeri versiyonları
- Minimum 512 MB Ram

- Minimum 256 MB DirectX 9 veya üzerini destekleyen Ekran kartı
- Minimum 1 GHz işlemci hızı

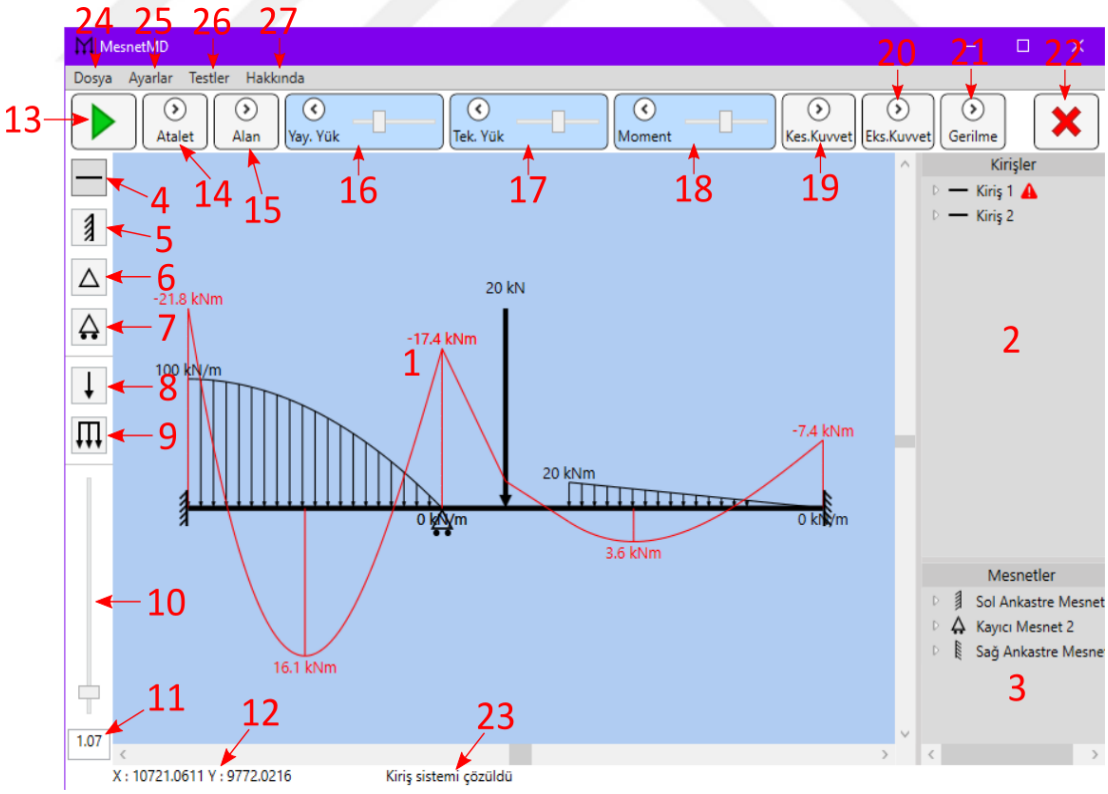
Bu gereksinimlerde ilk ikisi sağlanmadıkça programın çalışması imkânsızdır. Diğer gereksinimler programın verimli ve yeterince hızlı bir şekilde çalışması için gereklidir ve henüz denenmemiş olsa da bu verilen değerlerin altında da çalışabilir.

4. PROGRAMIN KULLANIMI

Bu tez kapsamında geliştirilen program görsel kullanıcı arayüzüne sahiptir. Proje adı MesnetMD olan programın bu bölümde kullanıcılar tarafından nasıl kullanılacağı anlatılacaktır.

4.1 Programa Genel Olarak Bakış

Programın arayüzü (Şekil 4.1)'deki gibidir. Açık mavi renkle görülen bir çizim alanı vardır. Oluşturulan kiriş sistemi burada görülür. Yanlarda kiriş, mesnet ve yük eklemek için butonlar bulunmaktadır. Üstte çözüm butonu ve silme butonu, en üstte de menü butonları vardır. Sağ tarafta eklenen kiriş ve mesnetler hakkında ayrıntılı bilgi içeren bir ağaç yapısı vardır. Aşağıda bütün görsel öğeler ayrıntılı olarak açıklanmıştır.



Şekil 4.1 : Programın ekran görüntüsü.

1) Programdaki çizim alanıdır. Bu alana kirişler, mesnetler ve yükler eklenir. Kuvvet, moment, gerilme ve atalet momenti dağılımları bu alanda gösterilir.

2) Eklenen kirişleri ve bu kirişler hakkındaki bilgileri gösteren ağaç görünümü (Tree View). Kirişin uzunluğu, üzerindeki tekil ve yayılı yükler, kirişin atalet momenti, kirişin üzerindeki yük, moment, kuvvet ve gerilme dağılımının bilgiler gösterilir. Kirişin her iki ucuna bağlanan mesnetler gösterilir.

3) Eklenen mesnetlerin ve bu mesnetler hakkındaki bilgilerin gösterildiği ağaç görünümüdür. Mesnede hangi kirişlerin hangi tarafının bağlandığı burada gösterilir.

4) Kiriş ekleme butonudur. Bu butona basınca kiriş ekleme penceresi çıkar. Kullanıcı pencereden istediği kirişin özelliklerini girerek çizim alanına ekler.

5) Ankastre mesnet ekleme butonudur. Kullanıcı kirişin sağ veya sol ucunu seçerek bu butona basar. Seçilen uca ankastre mesnet eklenmiş olur.

6) Basit mesnet ekleme butonudur. Kullanıcı kirişin sağ veya sol ucunu seçerek bu butona basar. Seçilen uca basit mesnet eklenmiş olur.

7) Kayıcı mesnet ekleme butonudur. Kullanıcı kirişin sağ veya sol ucunu seçerek bu butona basar. Seçilen uca kayıcı mesnet eklenmiş olur.

8) Tekil yük ekleme butonudur. Kullanıcı herhangi bir kirişi seçerek bu butona basar. Tekil yük ekleme penceresi açılır. Bu pencereden tekil yükün yeri ve şiddeti girilerek kirişe tekil yük eklenir.

9) Yayılı yük ekleme butonudur. Kullanıcı herhangi bir kirişi seçerek bu butona basar. Yayılı yük penceresi açılır. Yayılı yük veya yüklerin fonksiyonu ve etki aralığı bu pencereden seçilerek kirişe yayılı yük eklenir.

10) Ölçek çubuğudur. Kullanıcı bu çubuğu aşağı ya da yukarı kaydırarak hızlı bir şekilde çizim alanını yakınlaştırıp uzaklaştırabilir.

11) Ölçek kutucuğudur. Kullanıcı çizim alanının ölçeğini buradan görür ve istediği ölçeği girerek yakınlaştırma veya uzaklaştırma yapabilir.

12) Koordinat etiketidir. Orijine göre imlecin konumunu anlık olarak gösterir.

13) Çözüm butonudur. Kullanıcı tarafından oluşturulan düzlem çerçeve sisteminin çözümünün başlatıldığı butondur. Bu butona basılınca çözüm

penceresi açılır ve penceredeki ilerleme çubuğunda çözümün ilerleme durumu görülür. Çözüm bittiğinde bu pencere kapanır ve elde edilen sonuçlar kullanıcıya sunulur.

14) Atalet momenti dağılımları grafiği gösterme/gizleme butonudur.

15) Kesit alanı dağılımları grafiği gösterme/gizleme butonudur.

16) Yayılı yük dağılımları grafiği gösterme/gizleme butonudur.

17) Tekil yük dağılımları grafiği gösterme/gizleme butonudur.

18) Eğilme moment dağılımları grafiği gösterme/gizleme butonudur.

19) Kesme kuvveti dağılımları grafiği gösterme/gizleme butonudur.

20) Eksenel kuvvet dağılımları grafiği gösterme/gizleme butonudur.

21) Gerilme dağılımları grafiği gösterme/gizleme butonudur.

22) Çerçeve sistemi silme butonudur. Çizim alanındaki bütün kiriş ve mesnetleri siler.

23) Mesaj kutusudur. Programın işleyişi hakkında kullanıcıyı bilgilendirir.

24) Dosya menüsüdür. Kullanıcı oluşturduğu kiriş sistemini bir dosyaya kaydedebileceği ya da daha önce kaydedilmiş bir kiriş sistemini yükleyebileceği menüleri içerir.

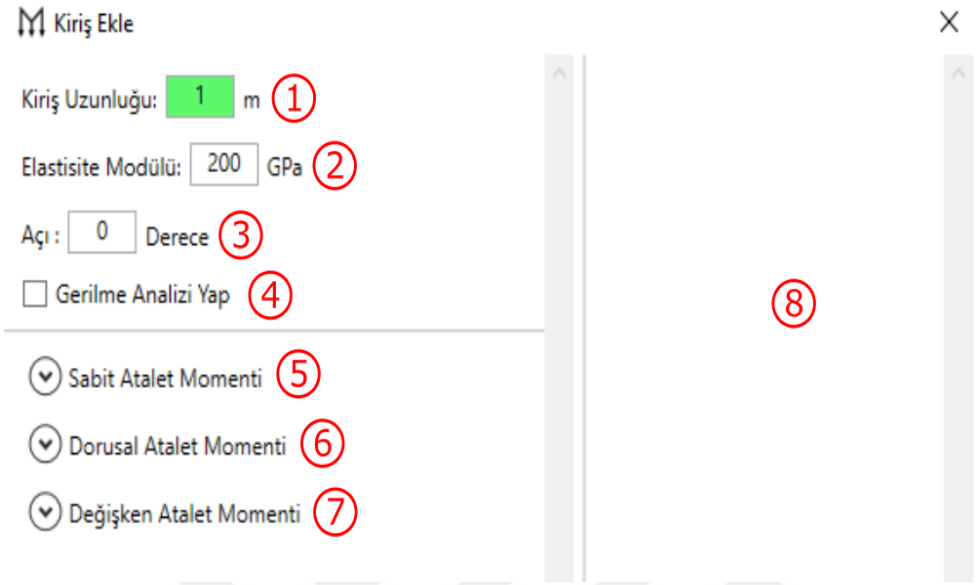
25) Ayarlar menüsüdür. Bu butona basınca ayarlar penceresi açılır. Kullanıcı buradan programın dilini ve hesaplama yöntemini seçebilir. Programın dili İngilizce veya Türkçe olabilir. Hesaplama yöntemi çok işlem parçacıklı ya da tek işlem parçacıklı olabilir. Çok işlem parçacıklı hesap yönteminde bilgisayarın gücüne bağlı olarak aynı anda birden fazla kirişin hesabı yapılır. Genelde tek iş parçacıklı hesaba göre daha kısa sürer.

26) Testler menüsüdür. Hali hazırda program içinde tanımlanan çerçeve sistemleri burada mevcuttur. Herhangi bir çerçeve sistemi seçildiğinde program o çerçeve sistemini yükler. Programda çözümün doğruluğunun test edilmesi amacıyla eklenmiştir.

27) Hakkında menüsüdür. Programın geliştiricisi, danışmanı, programın lisansı ile ilgili bilgilerin yer aldığı pencereyi açar.

4.2 Programda Bir Kirişin Eklenmesi

(Şekil 4.2)'deki resimde kiriş ekleme penceresi gösterilmiş ve numaralandırılmıştır.



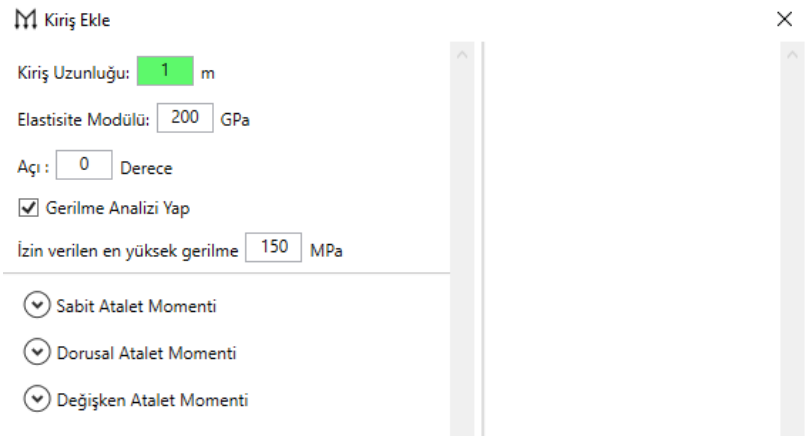
Şekil 4.2 : Kiriş ekleme penceresi.

1) Kullanıcı tarafından kiriş uzunluğunun girildiği yazı kutucuğudur. Kiriş uzunluğu metre cinsinden girilmelidir. Eğer geçersiz bir giriş yapılırsa kutucuğun arka plan rengi kırmızı geçerli giriş yapılırsa yeşil olur.

2) Kullanıcı tarafından kirişin Elastisite modülünün girildiği yazı kutucuğudur. Elastisite modülü GPa cinsinden girilmelidir. Eğer geçersiz bir giriş yapılırsa kutucuğun arka plan rengi kırmızı geçerli giriş yapılırsa yeşil olur.

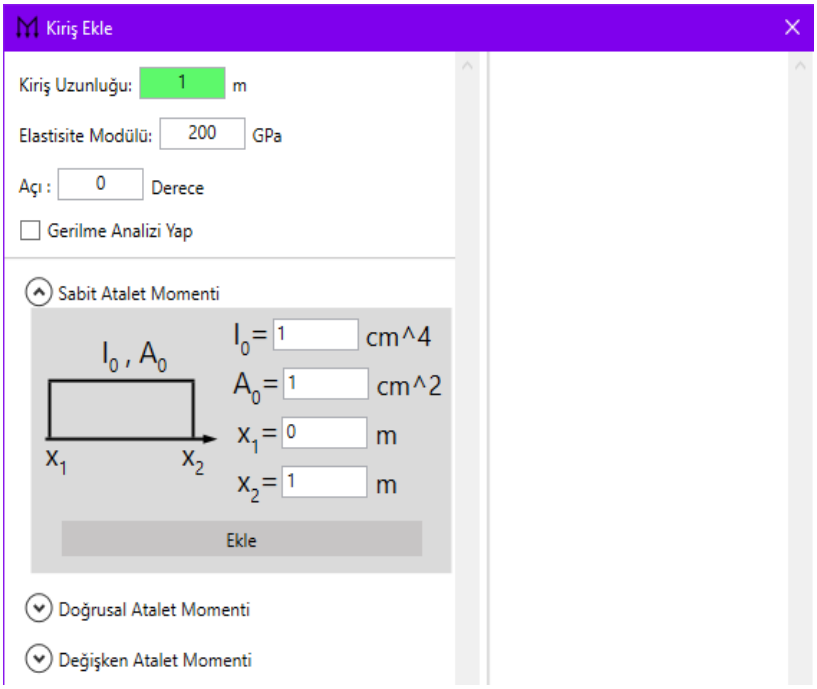
3) Kirişin açısının girildiği yazı kutucuğudur. Varsayılan değeri sıfırdır. Açı değeri derece cinsinden girilmelidir. Eğer geçersiz bir giriş yapılırsa kutucuğun arka plan rengi kırmızı geçerli giriş yapılırsa yeşil olur. Kiriş eklenirken bu açı değerine göre döndürülür. Açı değeri aynı zamanda kiriş eklendikten sonra da döndürme butonundan değiştirilebilir.

4) Gerilme analizinin yapılıp yapılmayacağı ile ilgili olan işaret kutucuğudur. Eğer gerilme analizi yapılacaksa bu kutucuk işaretlenir. İşaretlendikten sonra bu kutucuğun altında maksimum müsaade edilen gerilmenin MPa olarak girileceği bir kutucuk daha çıkar. Maksimum müsaade edilen gerilme değeri varsayılan olarak 150 MPa'dır ve kullanıcı bu değeri değiştirebilir. (Şekil 4.3)'te gerilme analizi kutucuğunun işaretlendiği durum gösterilmiştir.



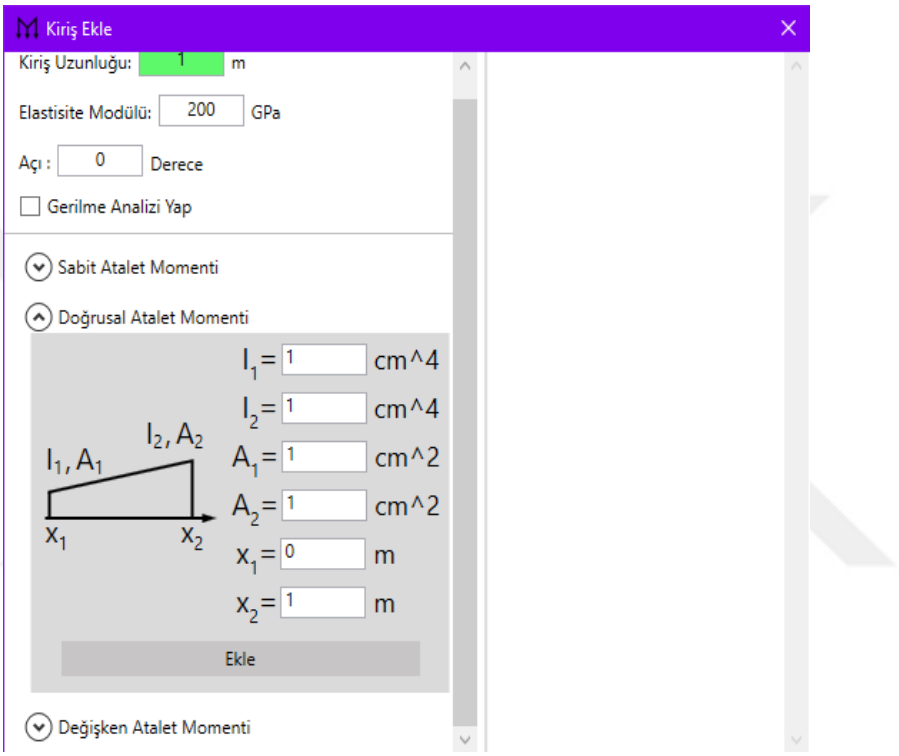
Şekil 4.3 : Gerilme analizi kutucuğu işaretlendiğinde ortaya çıkan müsaade edilen gerilmenin girildiği kutucuğun göründüğü ekran görüntüsü.

5) Kirişe sabit atalet momentinin eklenmek istendiği zaman tıklanan açılır kapanır alandır. Bu alana tıklandığında (Şekil 4.4)'teki alan ve atalet momenti giriş alanı açılacaktır. Atalet momentinin cm^4 cinsinden, alan cm^2 girilmesi gerekmektedir. x_1 ve x_2 değerleri girilen alan ve atalet momenti değerinin kirişteki başlangıç ve bitiş noktalarıdır. Bu noktalar varsayılan değer olarak 0 ve kiriş uzunluğu kadardır. Yani program girilecek alan ve atalet momentinin kiriş boyunca etkili olduğunu başlangıçta kabul eder. Kullanıcı bu değerleri değiştirerek kirişin farklı bölgelerine farklı alan ve atalet momenti atayabilir.



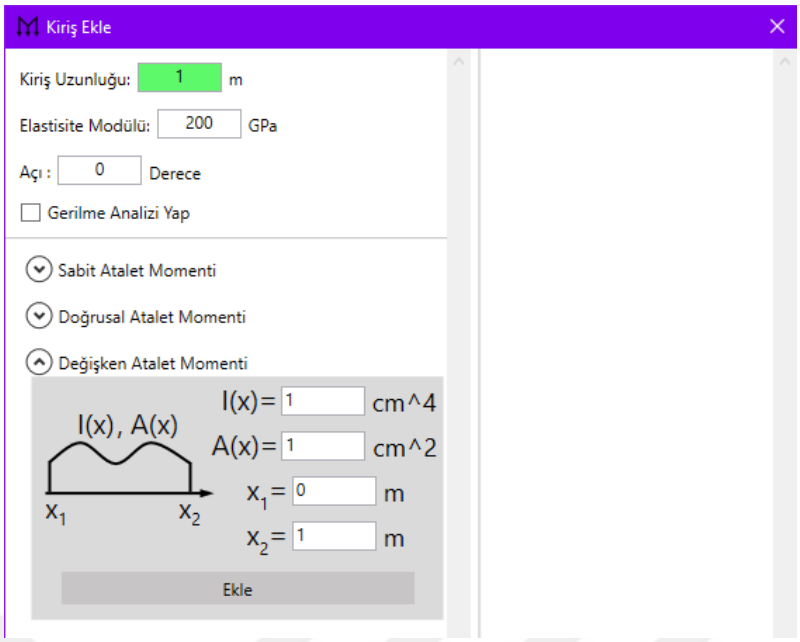
Şekil 4.4 : Sabit atalet momentinin girildiği alanı gösteren kiriş ekleme penceresi.

6) Kirişe doğrusal değişen alan ve atalet momentinin eklenmek istendiği zaman tıklanan açılır kapanır alandır. Dikdörtgen kesitli bir kirişte yüksekliğin değişmediği ancak genişliğin doğrusal olarak arttığı durum buna örnek olarak gösterilebilir. Bu durumda atalet momenti kiriş boyunca ya da kirişin bir kısmında doğrusal olarak artacaktır. (Şekil 4.5)'te bu alan gösterilmiştir. Bu durumda sabit atalet momentinden farklı olarak baştaki ve sondaki atalet momentinin yine cm^4 cinsinden alanın yine cm^2 girilmesi gerekecektir.



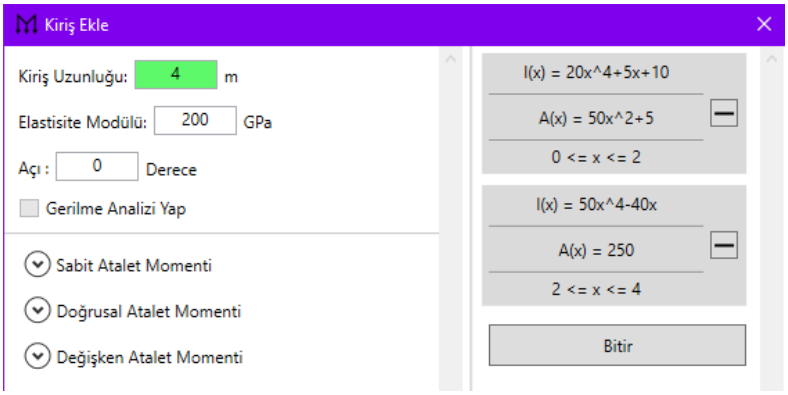
Şekil 4.5 : Doğrusal değişen atalet momentinin girildiği alanı gösteren kiriş ekleme penceresi.

7) Kirişe değişken alanın ve atalet momentinin eklenmek istendiği zaman tıklanan açılır kapanır alandır. (Şekil 4.6)'da bu alan gösterilmiştir. Burada alan ve atalet momenti olarak katsayıları reel sayı ve üsleri pozitif reel sayı olan polinomlar girilmesi istenmektedir. Tabii bu polinomların kirişin fiziksel olarak anlamsız olmaması için hiçbir noktasında alanın ve atalet momentinin sıfıra eşit ya da sıfırdan küçük olmaması gerekir. Bu durum program tarafından kontrol edilir. Eğer polinom herhangi bir noktada sıfıra eşit ya da sıfırdan küçük ise kullanıcı uyarılır ve bu polinom alan ya da atalet momenti fonksiyonu olarak kabul edilmez.



Şekil 4.6 : Değişken atalet momentinin girildiği alanı gösteren kiriş ekleme penceresi.

8) Eklenen atalet momentlerinin gösteriliği alandır. Hangi atalet momentinin hangi nokta aralıklarında atandığı buradan görülebilir. Alttaki bitir butonuna basılınca kiriş ekleme penceresi kapanır ve kiriş çizim alanına eklenir. (Şekil 4.7)'de gösterilmiştir.

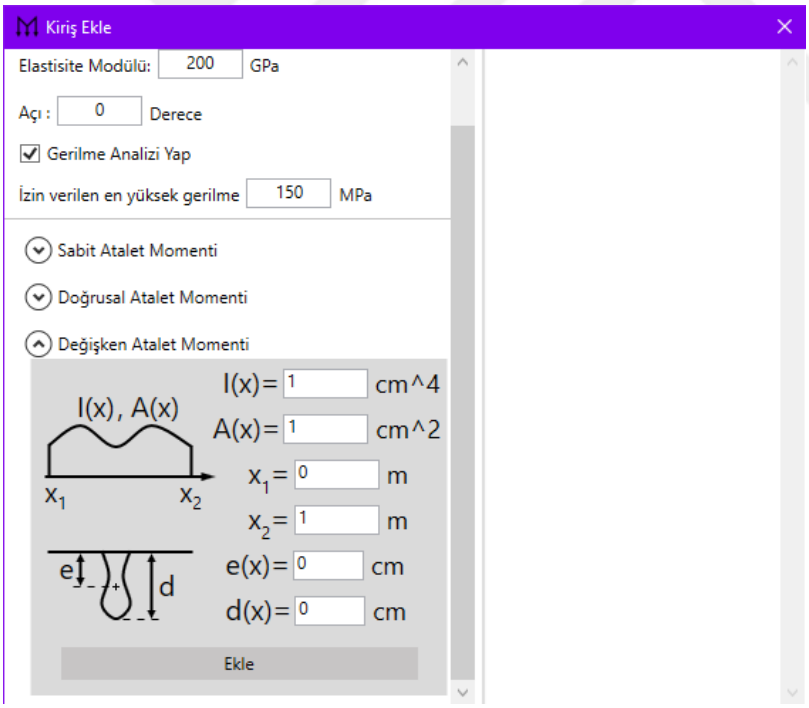


Şekil 4.7 : 2 farklı atalet momenti ve alan atanmış kiriş penceresi.

Kirişe parçalı fonksiyon halinde polinom olarak değişen, lineer olarak değişen ve sabit atalet momentleri istenilen sayıda eklenebilmektedir. Burada eklenen parçalı fonksiyonlarının hiçbirinde x aralıkları birbirinin içine geçmemelidir. Program bu durumu kontrol eder ve böyle bir atalet momentinin eklenmesini kabul etmez. Belli bir aralıkta sadece 1 tane atalet momenti tanımlı olması gerekmektedir. Ayrıca bitir tuşuna basıldığında program atalet momenti tanımlanmamış herhangi bir x aralığı

olmadığını da kontrol eder ve bu duruma izin vermez. Çünkü gerçek bir kirişin her noktasında bir atalet momenti olması gerekir. 8 numaralı maddede bahsedilen alanda aynı zamanda eklenen atalet momentleri çıkarılabilir. Bunun için (Şekil 4.7)'de görülen her atalet momentinin sağında eksi işaretli bir buton bulunmaktadır. Bu butona basılarak ilgili atalet momenti çıkarılır. Yanlışlıkla eklenen atalet momentleri böylece düzeltilebilir.

Eğer kullanıcı gerilme analizi yapmak için işaret kutucuğunu işaretlemişse bu kiriş için her noktada e ve d değerleri istenecektir. Burada d kirişin o noktadaki kesitinin yüksekliği ve e tarafsız eksenin mesafesidir. Bu iki değer cm olarak istenmektedir. Dolayısıyla bu değerlerin girileceği özel bir alan açılır. (Şekil 4.8)'de değişken atalet momenti alanında bu durum gösterilmiştir. Atalet momenti polinom olarak değiştiği için e ve d polinom olarak istenecektir. Bu değişkenler doğrusal atalet momentinde de polinom olarak alınır. Sadece sabit atalet momentinde bu iki değer sabit olarak istenir.

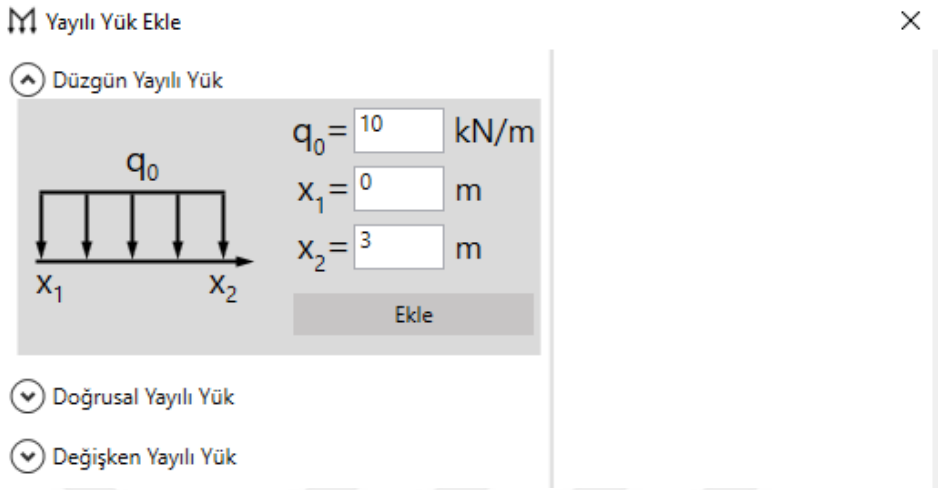


Şekil 4.8 : Gerilme analizi yapılacağı durumdaki atalet momenti alanı.

Gerilme analizi yapmak üzere tasarlanmış kirişte diğer kirişlerden farklı olarak gerilme dağılımı grafiği eklenebilir ve kirişin kritik kesitinin olup olmadığı kolayca görülebilir.

4.3 Programda Kirişe Yük Eklenmesi

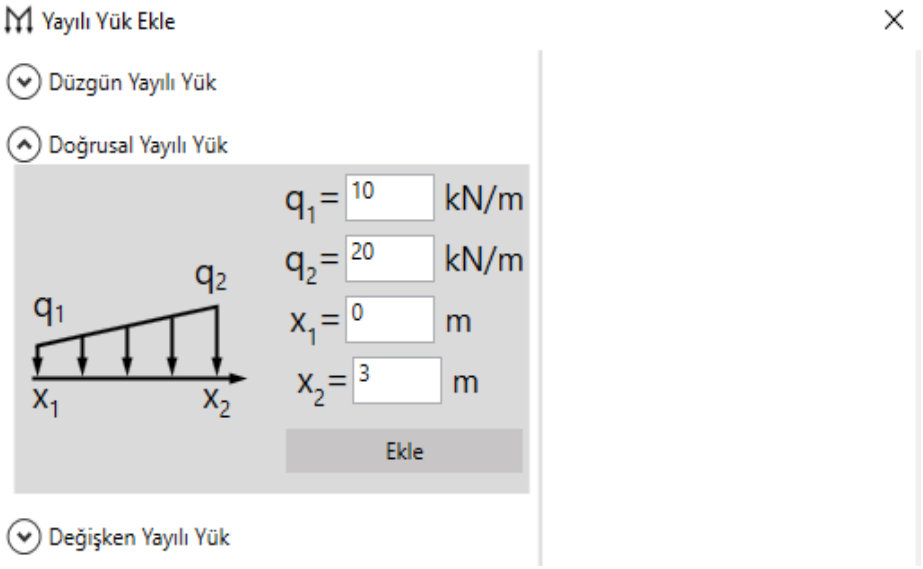
(Şekil 4.9)'da yayılı yük ekleme penceresi gösterilmiştir.



Şekil 4.9 : Yayılı yük ekleme penceresi.

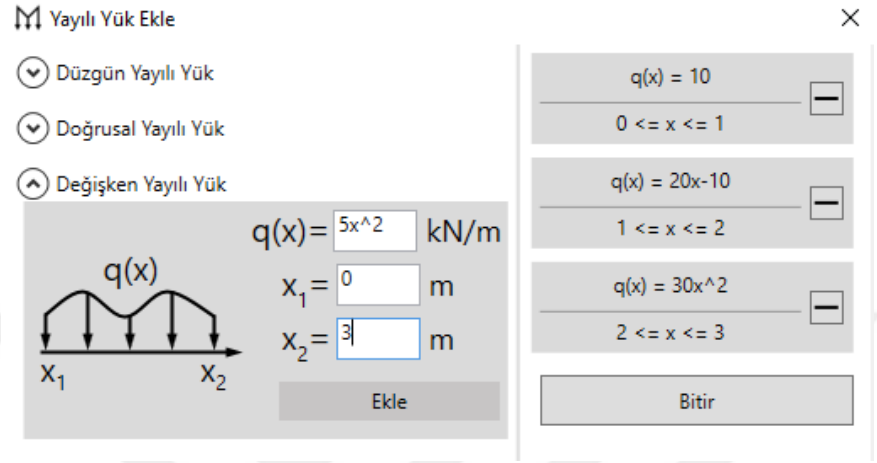
Kullanıcı düzgün, doğrusal ve değişken yayılı yük olmak üzere 3 farklı yük ekleyebilir. Düzgün yayılı yükte kullanıcı sabit yok olan q_0 değerini girmelidir. Bu penceredeki tüm yükler kN/m cinsinden girilir. x_1 ve x_2 yayılı yükün kiriş üzerindeki başlangıç ve bitiş noktasıdır. Ekle tuşuna basılınca yayılı yükün eklendiği sağ taraftaki alanda görülecektir.

(Şekil 4.10)'da doğrusal yayılı yük ekleme alanı gösterilmiştir. Burada q_1 ve q_2 yayılı yükün başındaki ve sonundaki değerlerdir.



Şekil 4.10 : Yayılı yük ekleme penceresinde doğrusal yayılı yük alanın görünüşü.

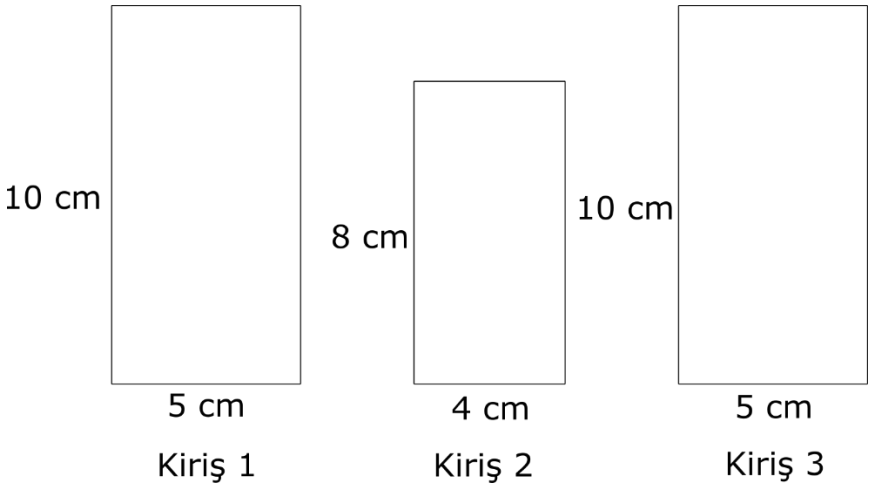
(Şekil 4.11)'de değişken yayılı yük ekleme alanı gösterilmiştir. Burada q(x) yayılı yükün polinomudur. Sağ taraftaki alanda görüldüğü gibi kirişe hem düzgün hem doğrusal hem de değişken yayılı yük eklenmiştir. Eklenen yayılı yüklerden herhangi sağ tarafındaki eksi butonun basılarak çıkarılabilir. Böylece hatayla eklenen yayılı yükün çıkarılmasına imkan tanınmış olur.



Şekil 4.11 : Yayılı yük ekleme penceresinde değişken yayılı yük alanın görünüşü.

4.4 Programda Örnek Bir Kiriş Sistemi Oluşturulması ve Çözülmesi

En kesitleri (Şekil 4.12)'deki gibi olan 3 kiriş eklenerek örnek bir gemi çerçevesi oluşturulacaktır.



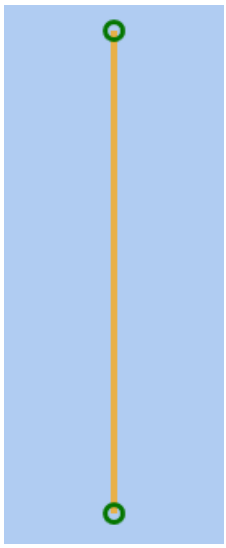
Şekil 4.12 : Eklenecek kirişlerin kesitleri.

Öncelikle kiriş butonuna basılır ve kiriş penceresi otomatik olarak açılır. Kiriş uzunluğunu 3 metre ve açı 90 derece girilmiştir. Bu durumda 3 metrelik dikey bir kiriş oluşturulacaktır. Elastisite modülü varsayılan değer yani 200 GPa olarak

seçilmiştir. Gerilme analizi yapmak üzere tıklanmıştır. (Şekil 4.12)'den anlaşılacağı gibi atalet momenti $416.667\ cm^4$ ve alan $50\ cm^2$ olacaktır. Ayrıca tarafsız eksen mesafesi $e = 5\ cm$ ve kirişin yüksekliği $d = 10\ cm$ olacaktır. Her 3 kirişin atalet momenti kiriş boyunca kendi içlerinde sabit olacaktır. Bu veriler girildikten sonra atalet momenti eklenerek bitir tuşuna basılır. Kiriş eklenmiş olacaktır. Kirişin görünümü (Şekil 4.13)'teki gibi olacaktır.

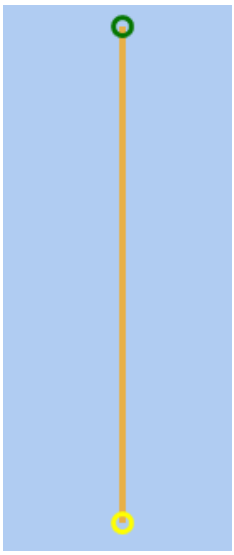
Şekil 4.13 : Kiriş 1.

Kirişin üzerine tıklanarak seçilir. (Şekil 4.14)'te seçilmiş kiriş gösterilmiştir.



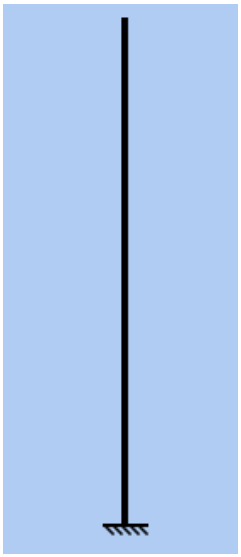
Şekil 4.14 : Seçilmiş olan kirişin görünüşü.

Görüldüğü gibi seçilen kirişin her iki ucunda yeşil çemberler ortaya çıkar. Alttaki çembere tıklanırsa (Şekil 4.15)'teki gibi çemberin rengi sarı olur.



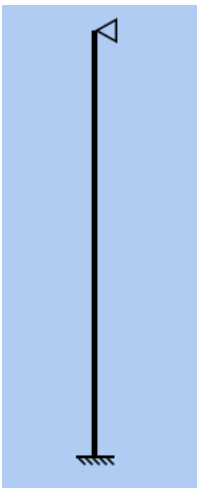
Şekil 4.15 : Alt çemberi seçilmiş kirişin görünüşü.

Kirişin bu ucu seçilmiştir. Şu durumda basit ve ankastre mesnet butonları aktif olur. Ankastre mesnet butonuna basınca bu seçilmiş olan çemberin olduğu yere ankastre mesnet eklenecektir. (Şekil 4.16)'daki ankastre mesnet eklenmiş hali gösterilmiştir. Ankastre mesnet eklendiğinde kiriş tekrar seçildiğinde ankastre mesnet eklenen tarafta çember görülmeyecektir. Bunun nedeni ankastre mesnet eklenen tarafa başka bir mesnet ya da kiriş eklenemeyecek olmasıdır.



Şekil 4.16 : Ankastre mesnet eklenmiş kirişin görünüşü.

Kiriş yeniden seçilip diğer uçtaki çember seçilsin ve basit mesnet tuşuna basılsın. Bu durumda diğer uca basit mesnet eklenmiş olacaktır. (Şekil 4.17)'de kirişin yeni hali gösterilmiştir.

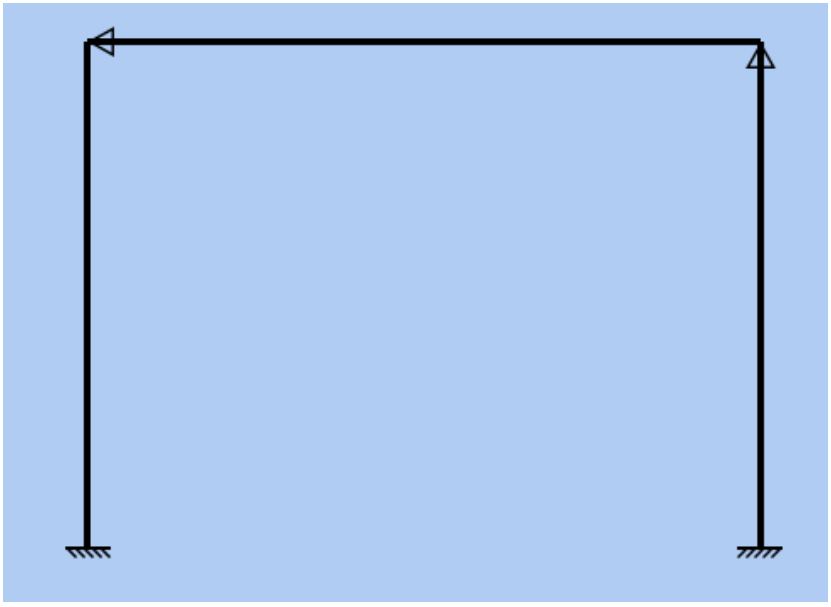


Şekil 4.17 : Ankastre ve basit mesnet eklenmiş kirişin görünüşü.

Kiriş yeniden seçildiğinde sadece basit mesnet tarafındaki çember görünür olacaktır. Bu çembere yeniden tıklanır ve kiriş butonuna basılırsa ilk kirişteki gibi kiriş ekleme penceresi açılacaktır. Buraya ikinci kirişin özellikleri girilecektir. Uzunluğu 4 metre olacaktır. (Şekil 4.12)'ten anlaşılacağı gibi atalet momenti $170.667\ cm^4$ ve alanı $24\ cm^4$ olacaktır. Tarafsız eksen mesafesi $e = 4\ cm$ ve kirişin yüksekliği $d = 8\ cm$ olacaktır. Bu kirişin açı değeri 0 derece olarak bırakılacaktır. Böylece bitir tuşuna basılınca yatay bir kiriş eklenecektir. (Şekil 4.18)'deki kiriş sistemi ortaya çıkacaktır.

Şekil 4.18 : İkinci kirişin eklendikten sonra sistemin görünüşü.

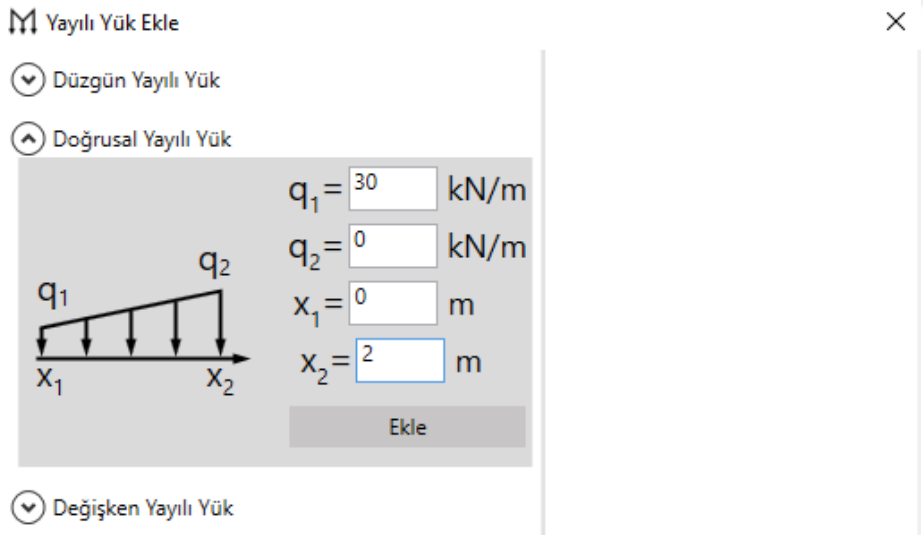
İkinci kirişin diğer ucuna basit mesnet daha önce anlatıldığı gibi eklenir. Daha sonra yine bu uçtaki çember seçilerek kiriş butonuna basılır ve üçüncü kiriş için kiriş ekleme penceresi görünür. Bu kirişin de uzunluğu 3 metre olaraktır. Açısı -90 derece girilir. Atalet momenti, tarafsız eksen uzunluğu ve kirişin yüksekliği ilk kirişin aynısı olacaktır. Üçüncü kiriş de eklendiğinde alt ucuna ankastre mesnet eklenince (Şekil 4.19)'daki sistem ortaya çıkacaktır.



Şekil 4.19 : Üçüncü kirişin eklendikten sonra sistemin görünüşü.

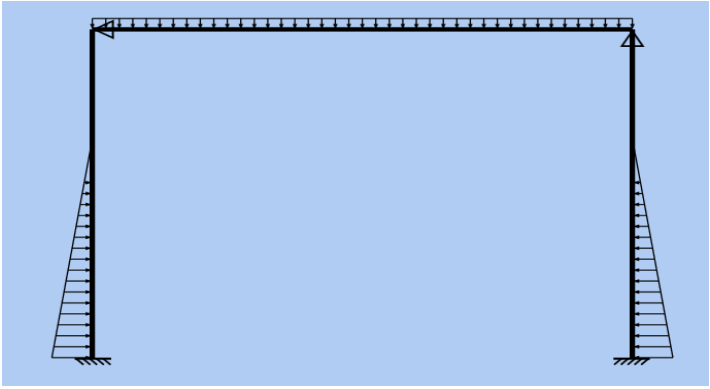
Aslında bu kiriş sistemi basit bir gemi çerçevesidir. Geminin dip yapısının ataleti çok yüksek olduğu için postaların dibe bağlandığı yerler çoğunlukla ankastre kabul edilir.

Birinci kirişi seçip yayılı yük butonuna basılarak (Şekil 4.20)'de gösterilen yayılı yük eklensin.



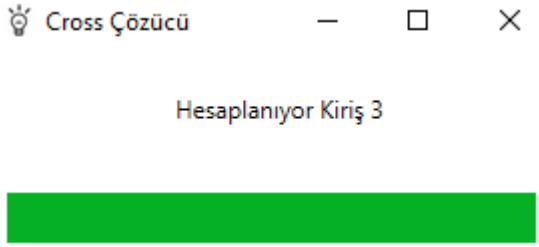
Şekil 4.20 : Birinci eklenecek olan yayılı yükün görünüşü.

Üçüncü kirişe de aynı yayılı yük eklensin. Bu yayılı yük draftı 2 metre olan geminin bordasındaki hidrostatik basıncın mukavemet modelidir. İkinci kirişe de $10\,kN/m$'lik kirş boyunca düzgün yayılı yük eklensin. Bu güverte yükünü temsil eder. Son durumda sistem (Şekil 4.21)'deki gibi olacaktır.



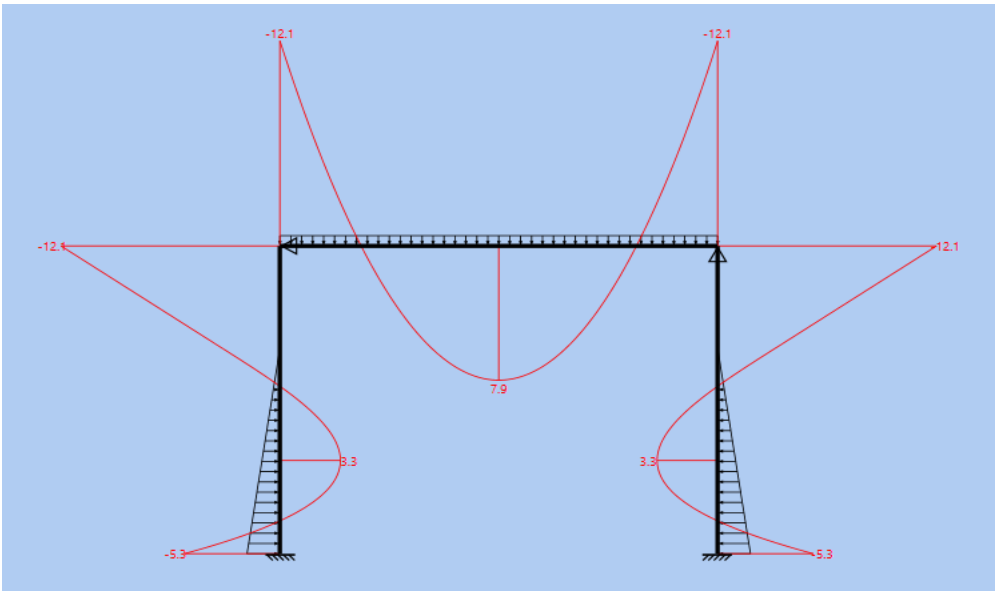
Şekil 4.21 : Yayılı yükleri eklenmiş kiriş sisteminin görünüşü.

Bütün kirişler ve yükler eklendiğine göre artık çözüme geçilebilir. Bunun için yapılması gereken tek şey sol üst taraftaki menü çubuğundaki çöz tuşuna basmaktır. Tuşa basıldığında (Şekil 4.22)'deki pencere açılacaktır.



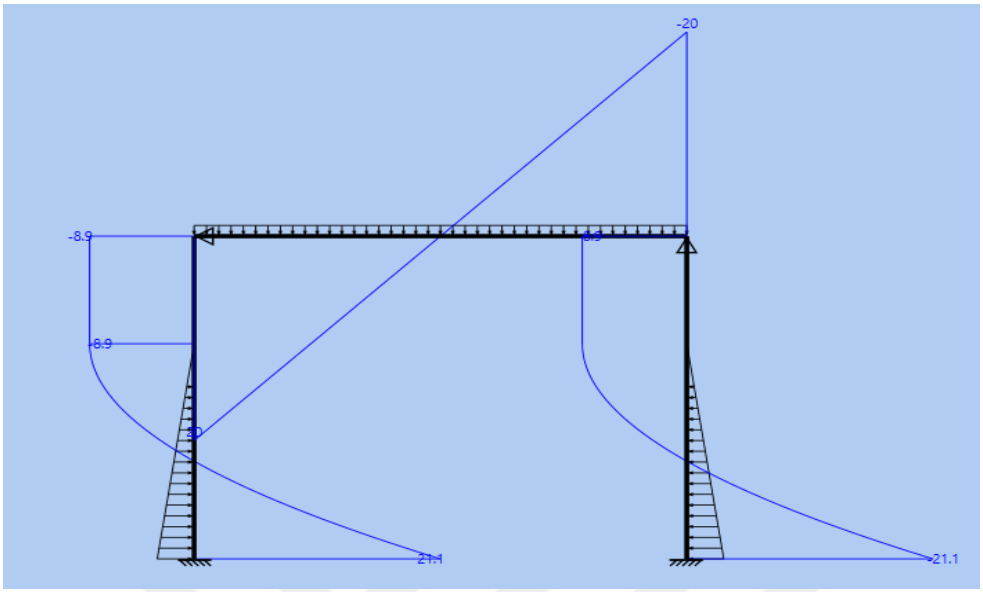
Şekil 4.22 : Kiriş sistemi çözülürken açılan pencere.

Çözüm işlemi bittikten sonra pencere kapatılacak ve (Şekil 4.23)'teki moment dağılımları ortaya çıkacaktır.



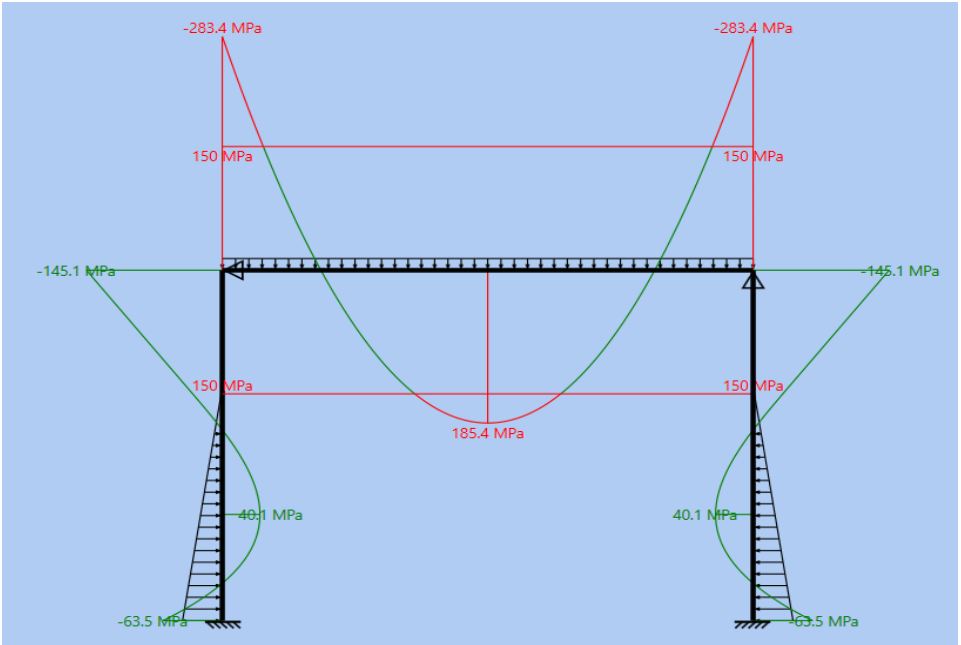
Şekil 4.23 : Kiriş sisteminin eğilme moment dağılımı.

Yine yukarıdaki menü çubuğundan moment dağılımları gizlenip kuvvet dağılımlarını gösterilirse (Şekil 4.24)'teki kuvvet dağılımı ortaya çıkacaktır.



Şekil 4.24 : Kiriş sisteminin kesme kuvveti dağılımı.

Kuvvet dağılımı gizlenip gerilme dağılımı gösterildiğinde (Şekil 4.25)'teki görünüm elde edilecektir. Kiriş sisteminde müsaade edilen maksimum gerilme $150\,MPa$ olarak girilmiştir. İkinci kirişte gerilme değerleri bu sınır değeri aştığı için bu aşan yerler kırmızı renkte gösterilmiştir. Ayrıca $150\,MPa$ sınırı grafik üzerinde çizilmiştir.

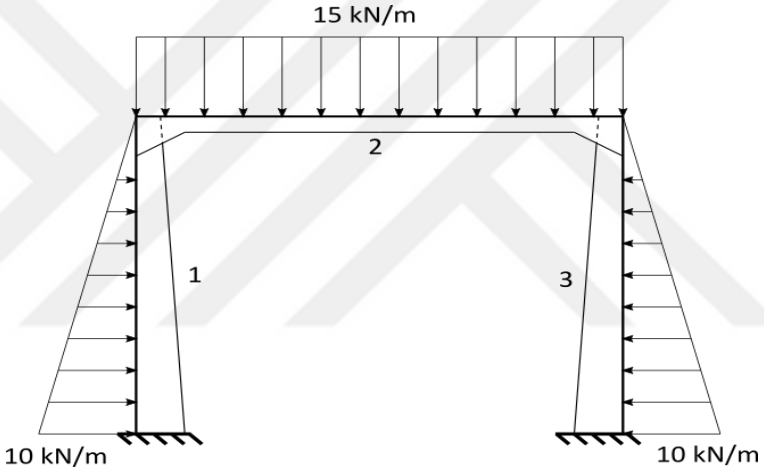


Şekil 4.25 : Kiriş sisteminin gerilme dağılımı.

5. SAYISAL ÇALIŞMA

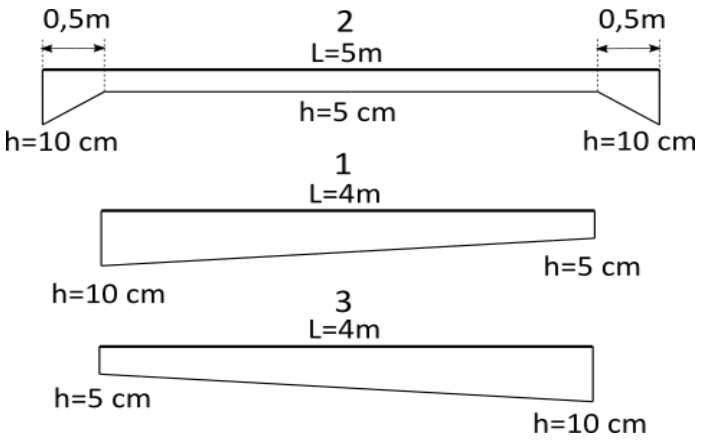
5.1 Değişken Kesitli Çerçeve Örneği

(Şekil 5.1)'de değişken kesitli bir çerçeve gösterilmiştir. 1 ve 3 kirişlerinin yükseklikleri doğrusal olarak değişmektedir. 2 kirişi ise uçlarından braketlenmiş olarak düşünülebilir. 2 numaralı kiriş 15 kN/m sabit yayılı yüküyle yüklenmiştir. 1 ve 3 numaralı kirişler ise maksimum değeri 10 kN/m olan ve kiriş sonunda sıfıra giden doğrusal yayılı yükle yüklenmiştir.



Şekil 5.1 : Değişken kesitli enine çerçeve.

(Şekil 5.2)'de kirişlerin kesit yükseklikleri gösterilmiştir. Ayrıca her bir kirişin derinliği 30 cm'dir.



Şekil 5.2 : Değişken kesitli enine çerçeve kirişlerinin boyutları.

Öncelikle kirişlerin her iki ucu ankastre mesnet iken moment dağılımı Üç Moment Yöntemiyle bulunmalıdır.

$$h_1(x) = h_3(x) = 10 - 1.25x \tag{5.1}$$

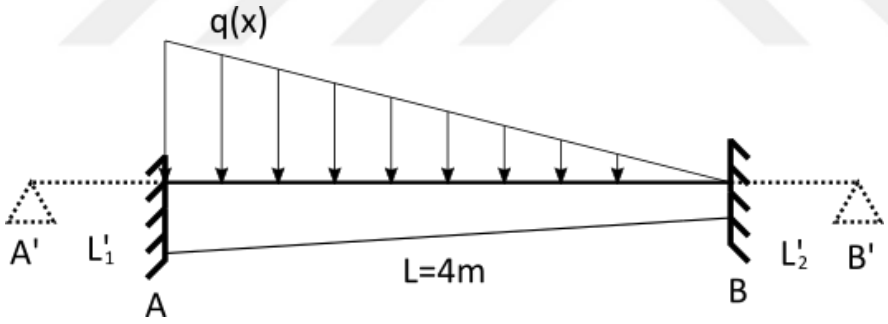
$$I_1(x) = I_3(x) = 2500 - 937.5x + 117.1875x^2 - 4.8828x^3 \tag{5.2}$$

$$h_2(x) = \begin{cases} 10 - 10x, & 0 \le x \le 0.5 \\ 5, & 0.5 \le x \le 4.5 \\ 10x - 40, & 4.5 \le x \le 5 \end{cases} \tag{5.3}$$

$$I_2(x) = \begin{cases} 2500 - 7500x + 7500x^2 - 2500x^3, & 0 \le x \le 0.5 \\ 312.5, & 0.5 \le x \le 4.5 \\ -160000 + 120000x - 30000x^2 + 2500x^3, & 4.5 \le x \le 5 \end{cases} \tag{5.4}$$

5.1.1 Analitik çözüm

(Şekil 5.3)'te 1 numaralı kiriş için sanal kiriş kabulü yapılmıştır.



Şekil 5.3 : Değişken kesitli enine çerçeve kirişi için sanal kiriş kabulü.

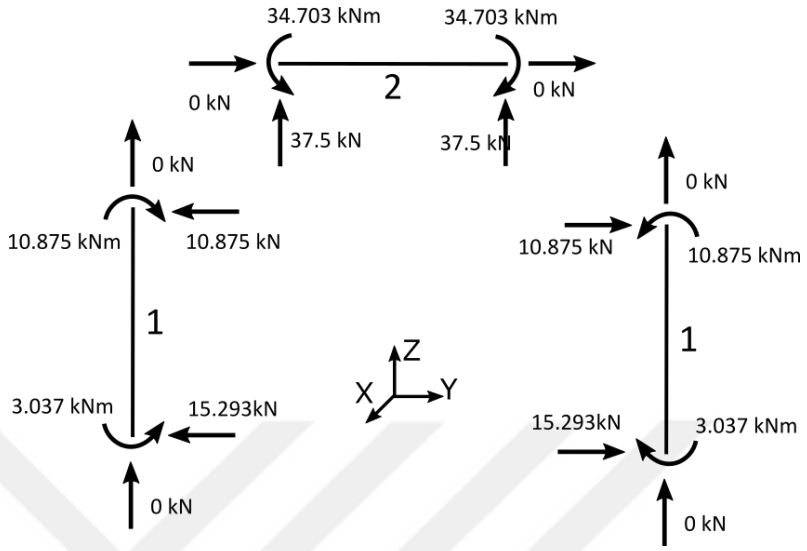
Bu sanal kiriş kabulü yapılan kiriş sisteminde denklem 2.82'deki Üç Moment Denklemi her iki kiriş parçacığı için yazılıp atalet dağılımı yerine konulursa aşağıdaki sonuçlar elde edilir.

$$M_A = -10.875\ kNm, \quad M_B = -3.0368\ kNm \tag{5.5}$$

Aynı işlemler 2 numaralı kiriş için yapıldığında aşağıdaki sonuçlar elde edilir.

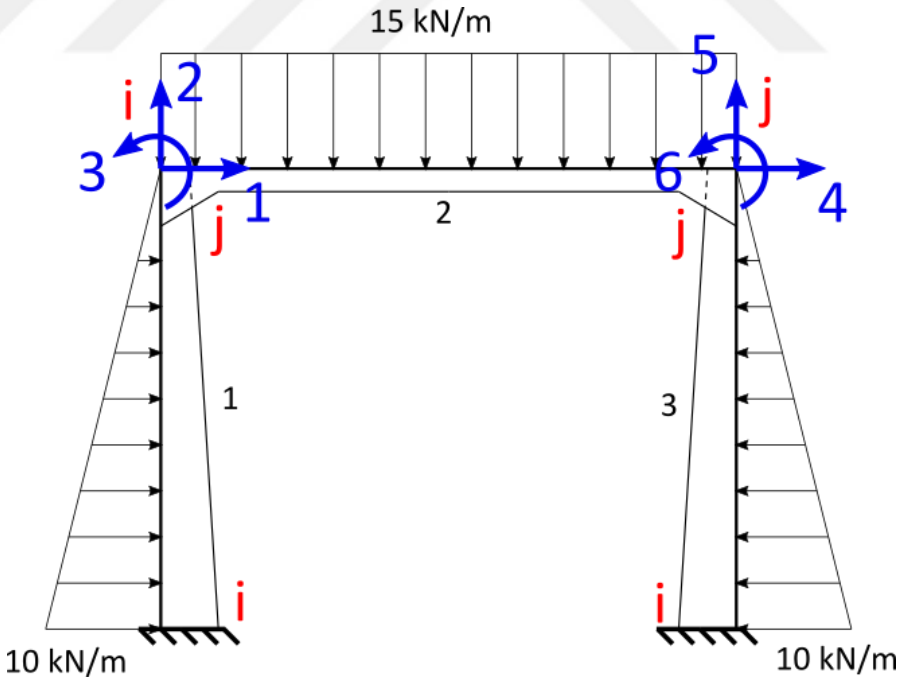
$$M_A = -34.7032\ kNm, \quad M_B = -34.7032\ kNm \tag{5.6}$$

3 numaralı kirişte bulunacak sonuçlar 1 numaralı kirişle aynı olacaktır. Burada kullanılan işaret sistemi kiriş teorisi işaret sistemidir. Mukavemet hesaplarından yararlanılarak kirişlerin mesnet tepkileri (Şekil 5.4)'teki gibi olur.



Şekil 5.4 : Değişken kesitli enine çerçeve kirişlerinin mesnet tepkileri.

Sistemin serbestlik dereceleri (Şekil 5.5)'teki gibi olacaktır.



Şekil 5.5 : Değişken kesitli enine çerçeve sisteminin serbestlik dereceleri.

Kiriş 1 ve 3'ün açısal konumu müşterek eksen takımına göre 90°'dir. Kiriş 2'nin ise açısal konumu 0°'dir. Bölüm 1.2.4'teki değişken kesitli kirişler için atalet ve alan

fonksiyonları konularak hesaplar yapılırsa temel rijitlik katsayıları aşağıdaki gibi elde edilir.

$$m_{ii}^{1} = 19.4515, \quad m_{jj}^{1} = 6.8642, \quad m_{ij}^{1} = 5.7265, \; n_{ii}^{1} = 1.4428 \tag{5.7}$$

$$m_{ii}^{2} = 5.5441, \quad m_{jj}^{2} = 5.5441, \quad m_{ij}^{1} = 3.2584, \; n_{ii}^{1} = 1.0654 \tag{5.8}$$

$$m_{ii}^{3} = 19.4515, \quad m_{jj}^{3} = 6.8642, \quad m_{ij}^{3} = 5.7265, \; n_{ii}^{3} = 1.4428 \tag{5.9}$$

(5.7)

Bütün kirişler için elastisite modülü $210\,GPa$ alınmıştır. Kirişlerin müşterek eksenlere göre katılık matrisi aşağıdaki gibi elde edilecektir.

$$k^{1}{}_{XYZ} = \begin{bmatrix} 1.1362 & 0. & 0. & -1.1362 & 0. & 0. \\ 0. & 0.0039 & 0.001 & 0. & -0.0004 & 0.0005 \\ 0. & 0.001 & 0.0003 & 0. & -0.001 & 0.0009 \\ -1.1362 & 0. & 0. & 1.1362 & 0. & 0. \\ 0. & -0.0004 & -0.001 & 0. & 0.0004 & -0.0005 \\ 0. & 0.0005 & 0.0009 & 0. & -0.0005 & 0.0011 \end{bmatrix} \tag{5.10}$$

$$k^{2}{}_{XYZ} = \begin{bmatrix} 0.6712 & 0. & 0. & -0.6712 & 0. & 0. \\ 0. & 0.0001 & 0.0002 & 0. & -0.0001 & 0.0002 \\ 0. & 0.0002 & 0.0007 & 0. & -0.0002 & 0.0004 \\ -0.6712 & 0. & 0. & 0.6712 & 0. & 0. \\ 0. & -0.0001 & -0.0002 & 0. & 0.0001 & -0.0002 \\ 0. & 0.0002 & 0.0004 & 0. & -0.0002 & 0.0007 \end{bmatrix} \tag{5.11}$$

$$k^{3}{}_{XYZ} = \begin{bmatrix} 1.1362 & 0. & 0. & -1.1362 & 0. & 0. \\ 0. & 0.0039 & 0.001 & 0. & -0.0004 & 0.0005 \\ 0. & 0.001 & 0.0003 & 0. & -0.001 & 0.0009 \\ -1.1362 & 0. & 0. & 1.1362 & 0. & 0. \\ 0. & -0.0004 & -0.001 & 0. & 0.0004 & -0.0005 \\ 0. & 0.0005 & 0.0009 & 0. & -0.0005 & 0.0011 \end{bmatrix} \tag{5.12}$$

Kirişlerin müşterek eksenlere göre endirekt yük vektörleri aşağıdaki gibi elde edilecektir.

$$f_{XYZ}^{1} = \begin{Bmatrix} -15292.8740 \\ 0 \\ 10875.0095 \\ -4707.1260 \\ 0 \\ -3036.8471 \end{Bmatrix} \tag{5.13}$$

$$f_{XYZ}^{2} = \begin{Bmatrix} 0 \\ 37500 \\ 34703.2014 \\ 0 \\ 37500 \\ -24703.2014 \end{Bmatrix} \tag{5.14}$$

$$f_{XYZ}^{3} = \begin{Bmatrix} -15292.8740 \\ 0 \\ 10875.0095 \\ -4707.1260 \\ 0 \\ -3036.8471 \end{Bmatrix} \tag{5.15}$$

Kod numaraları tablosu oluşturulup, oluşturulan tablo yardımıyla müşterek matris aşağıdaki gibi elde edilir.

$$k_{XYZ} = \begin{bmatrix} 0.6716 & 0. & 0.0005 & -0.6712 & 0. & 0. \\ 0. & 1.1364 & 0.0002 & 0. & -0.0001 & 0.0002 \\ 0.0005 & 0.0002 & 0.0019 & 0. & -0.0002 & 0.0004 \\ -0.6712 & 0. & 0. & 0.6712 & 0. & 0.0005 \\ 0. & -0.0001 & -0.0002 & 0. & 1.1364 & -0.0002 \\ 0. & 0.0002 & 0.0004 & 0.0005 & -0.0002 & 0.0019 \end{bmatrix} \tag{5.16}$$

Buradaki sonuçlar virgülden sonra 4 basamak olacak şekilde yuvarlatılarak verilmiştir. Müşterek endirekt yük vektörü aşağıdaki gibi olacaktır.

$$f_{XYZ} = \begin{Bmatrix} 4707.1260 \\ -37500 \\ -31666.3543 \\ -4707.1260 \\ -37500 \\ -31666.3543 \end{Bmatrix} \tag{5.17}$$

Direkt yük vektörü sıfır olduğu için endirekt yük vektörü denklem 2.112'deki gibi karşıya atılarak müşterek matris sistemi çözülürse aşağıdaki deplasman vektörü elde edilir.

$$D_{XYZ} = \begin{Bmatrix} 12128.3275 \\ -33005.4308 \\ -22208359.4908 \\ -11964.6512 \\ -33005.4119 \\ -22208359.7442 \end{Bmatrix} \tag{5.18}$$

Denklem 2.114'teki ifade kullanılarak kirişlerin eleman kuvvet vektörleri elde edilir.

$$\{P^1\}_{XYZ} = k^1{}_{XYZ}\begin{Bmatrix}0\\0\\0\\D_1\\D_2\\D_3\end{Bmatrix} + \{f^1_{XYZ}\} = \begin{Bmatrix}-3828.7704\\37500.0107\\-9977.5384\\-16171.2296\\-37500.0107\\-28040.7139\end{Bmatrix} \quad (5.19)$$

$$\{P^2\}_{XYZ} = [k^2_{XYZ}]\begin{Bmatrix}D_1\\D_2\\D_3\\D_4\\D_5\\D_6\end{Bmatrix} + \{f^2_{XYZ}\} = \begin{Bmatrix}16171.2296\\37500.0107\\28040.7134\\-16171.2296\\37499.9893\\-28040.6597\end{Bmatrix} \quad (5.20)$$

$$\{P^3\}_{XYZ} = [k^3_{XYZ}]\begin{Bmatrix}0\\0\\0\\D_4\\D_5\\D_6\end{Bmatrix} + \{f^3_{XYZ}\} = \begin{Bmatrix}3828.7704\\37499.9893\\9977.5921\\16171.2296\\-37499.9893\\28040.6597\end{Bmatrix} \quad (5.21)$$

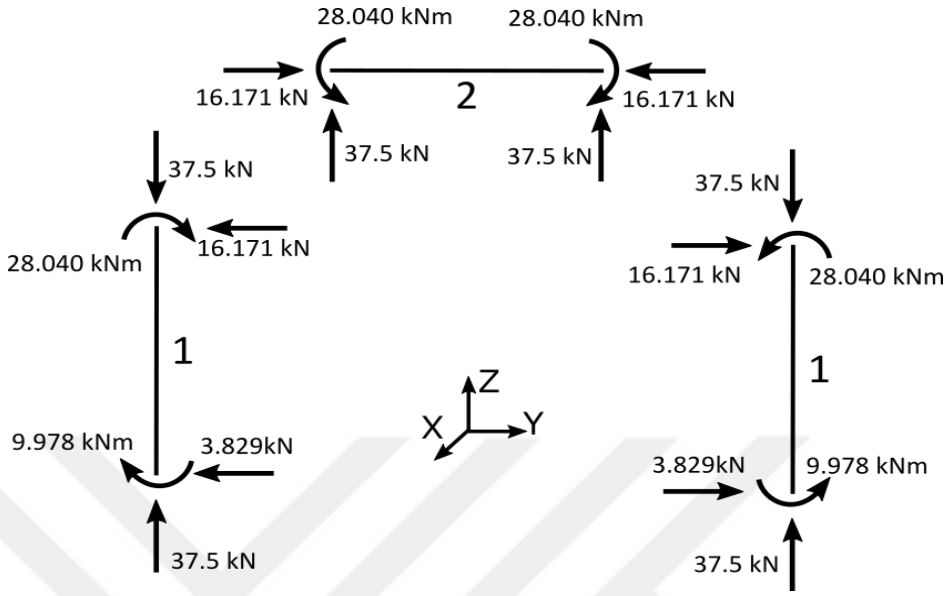
Daha sonra müşterek eksene göre tanımlanmış kuvvet vektörleri transformasyon matrisleriyle çarpılarak lokal kuvvet vektörleri elde edilir. Bu vektörler kirişlerdeki uç tepki kuvvetlerine karşılık gelir.

$$\{P^1\}_{xyz} = [T]\{P^1\}_{XYZ} = \begin{Bmatrix}37500.0107\\3828.7704\\-9977.5384\\-37500.0107\\16171.2296\\-28040.7134\end{Bmatrix} \quad (5.22)$$

$$\{P^2\}_{xyz} = [T]\{P^2\}_{XYZ} = \begin{Bmatrix}16171.2296\\37500.0107\\28040.7134\\-16171.2296\\37499.9893\\-28040.6597\end{Bmatrix} \quad (5.23)$$

$$\{P^3\}_{xyz} = [T]\{P^3\}_{XYZ} = \begin{Bmatrix}37499.9893\\-3828.7704\\9977.5921\\-37499.9893\\-16171.2296\\28040.6597\end{Bmatrix} \quad (5.24)$$

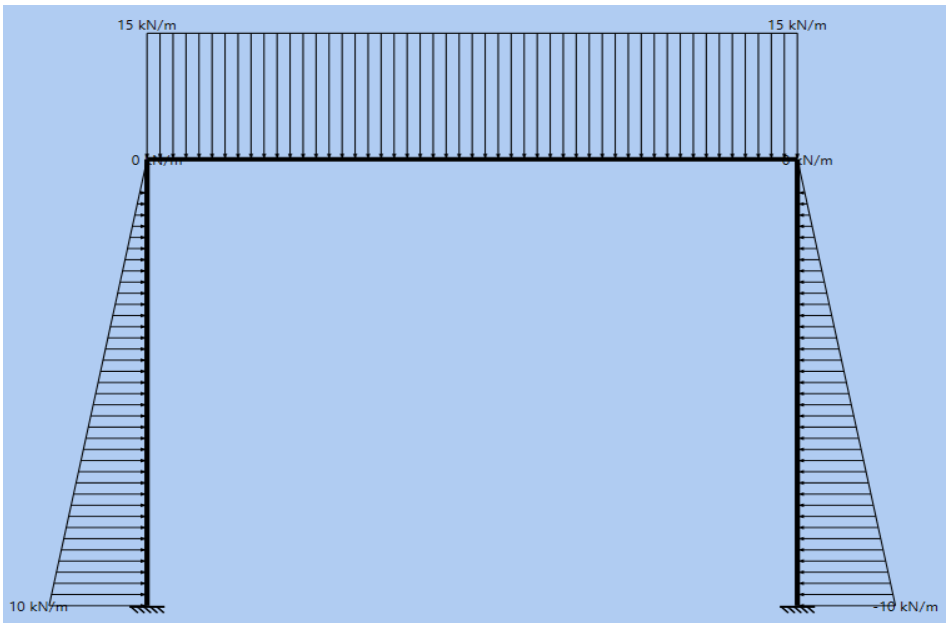
(Şekil 5.6)'da kirişlerin elde edilen uç tepki kuvvetleri gösterilmiştir. Bu tepki kuvvetleri bilindikten sonra kesit tesir diyagramları çizilebilir, eğilme momenti, kesme kuvveti, eksenel kuvvet, sehim ve gerilme dağılımları elde edilebilir.



Şekil 5.6 : Değişken kesitli enine çerçeve sisteminin kirişlerinin uç tepki kuvvetleri.

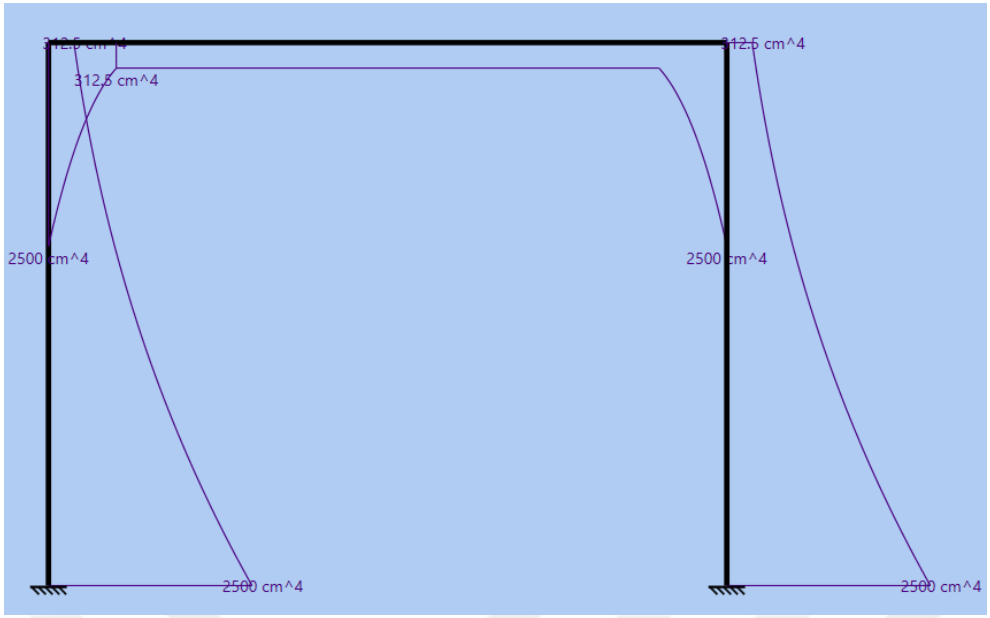
5.1.2 Program ile çözüm

Bölüm 4'te anlatıldığı gibi kirişlerin örnek çerçevedeki atalet ve alan değerleri girildiğinde programda çerçeve sistemi (Şekil 5.7)'deki gibi görünecektir.



Şekil 5.7 : Geliştirilen programda örnek çerçevenin görünüşü.

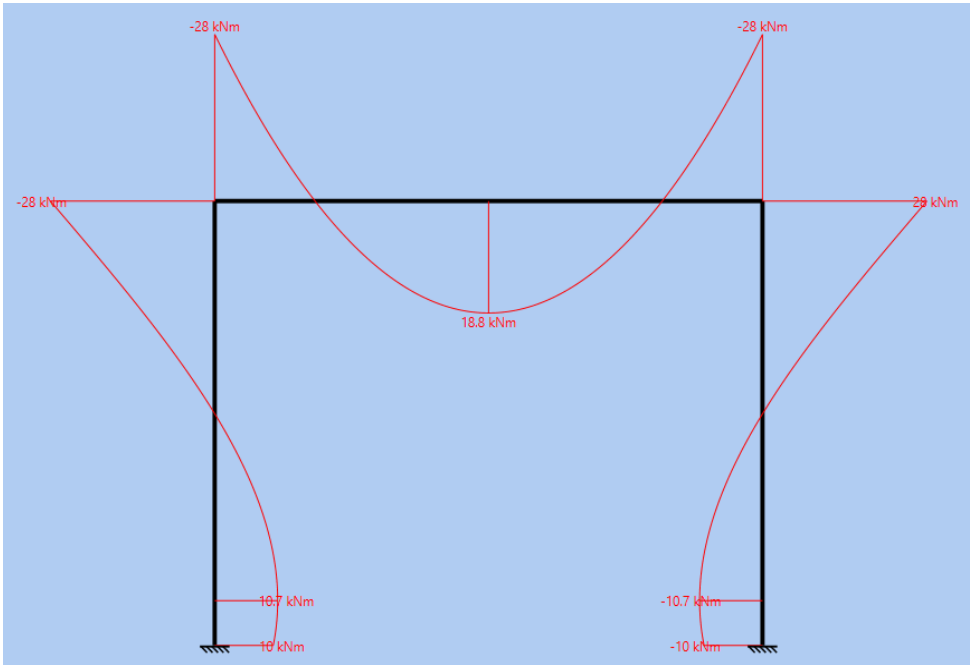
Çerçevenin atalet momenti dağılımı (Şekil 5.8)'deki gibi olacaktır.



Şekil 5.8 : Geliştirilen programda örnek çerçevenin atalet dağılımı.

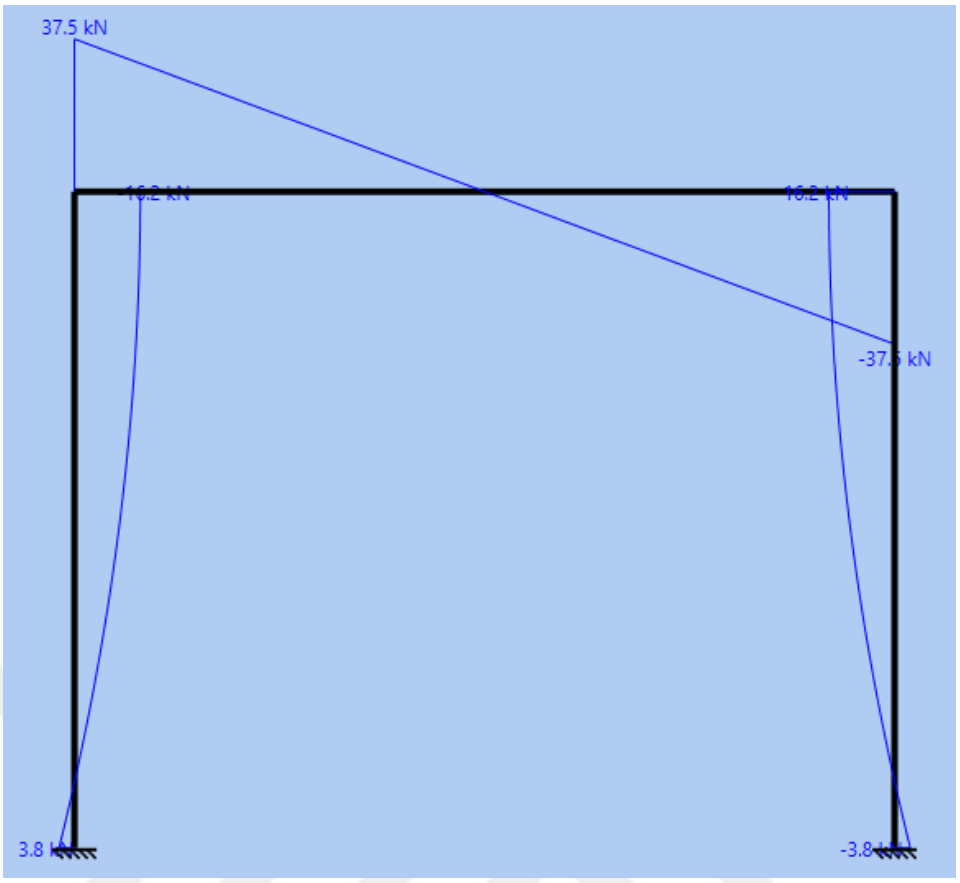
Burada çizimin simetrik olmamasının nedeni 3 numaralı kirişin 1 ile aynı yönde eklenmesidir. Bu durumda kirişin pozitif yükleme yönü değişecektir. Nitekim (Şekil 5.6)'da yayılı yükün uç değerinin 3 numaralı kirişte $-10\ kN/m$ olduğu görülmektedir.

Kiriş sistemi çözümü yapıldığında moment dağılımı (Şekil 5.9)'daki gibi olacaktır.



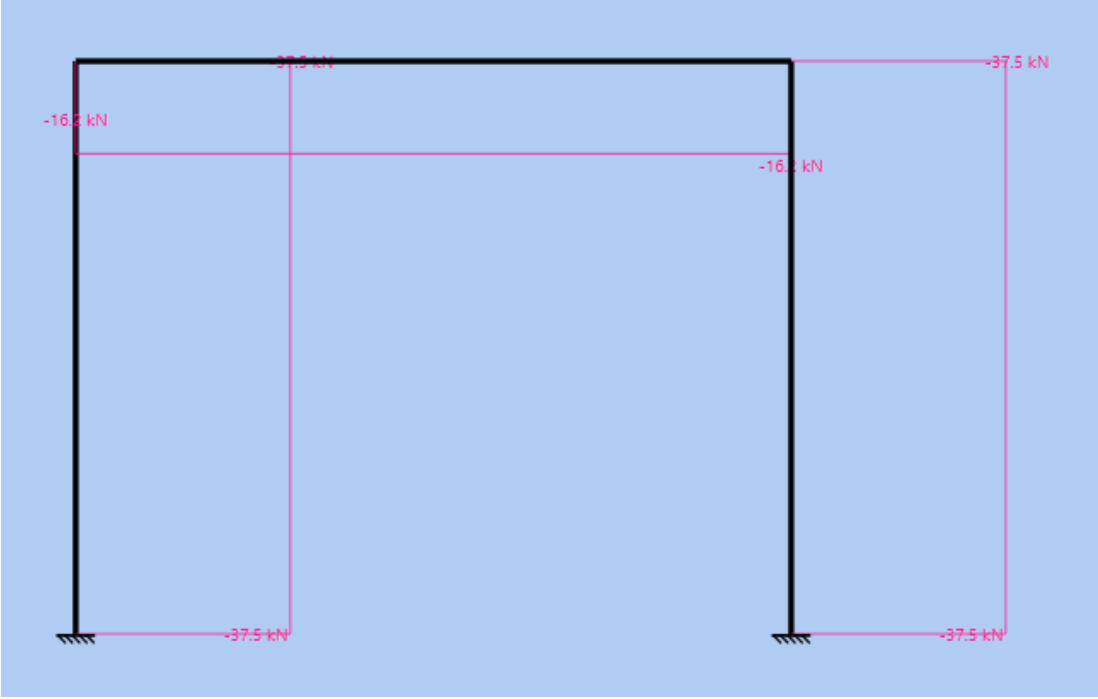
Şekil 5.9 : Geliştirilen programda örnek çerçevenin moment dağılımı grafiği.

Kesme kuvveti dağılımı (Şekil 5.10)'daki gibi olacaktır.



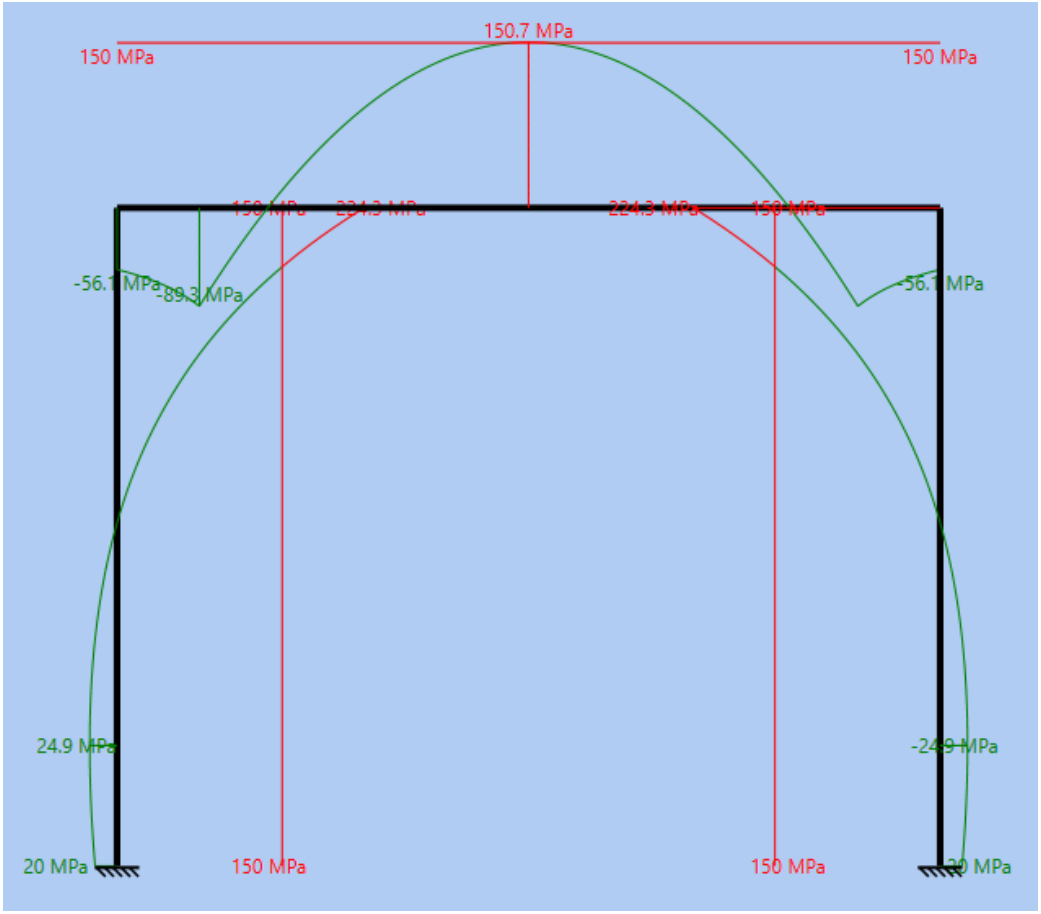
Şekil 5.10 : Geliştirilen programda örnek çerçevenin kesme kuvveti grafiği.

Eksenel kuvvet dağılımı (Şekil 5.11)'deki gibi olacaktır.



Şekil 5.11 : Geliştirilen programda örnek çerçevenin eksenel kuvvet grafiği.

Kiriş sisteminin gerilme dağılımı (Şekil 5.12)'deki gibi olacaktır.



Şekil 5.12 : Geliştirilen programda örnek çerçevenin gerilme grafiği.

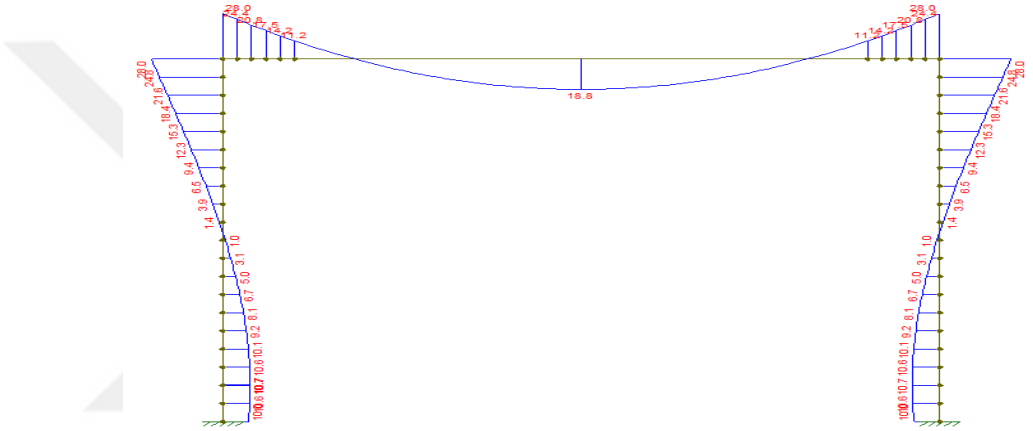
Burada müsaade edilen en yüksek gerilme $150\,MPa$ olarak kabul edilmiş ve program ona göre hesap yapmıştır. Grafikteki kırmızı eğriler o bölgede müsaade edilen en yüksek gerilmenin aşıldığını göstermektedir. Bu durumda her 3 kirişte de müsaade edilen gerilme aşılmıştır ve güvenli değildir. Program bu durumda kiriş ağacında müsaade edilen gerilmenin aşıldığı kirişin yanında (Şekil 5.13)'teki gibi kırmızı bir ünlem işareti gösterecektir.

Şekil 5.13 : Müsaade edilen en yüksek gerilmenin aşıldığı kirişlerdeki uyarı işareti.

Bu geliştirilen programdan başka Ftool adında Brezilya'lı bir inşaat mühendisi tarafından geliştirilen ve yine matris deplasman metoduyla hesap yapan bir çerçeve analiz programıyla örnek çerçeve çözülmüştür.

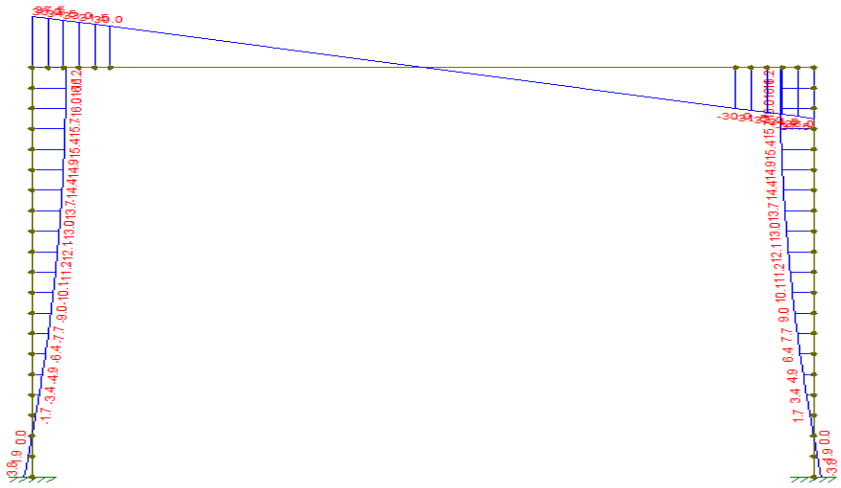
Ftool yalnızca sabit kesitli kirişler için hesap yapmaktadır. Bu nedenle değişken kesitler için kiriş parçalara bölünerek her kiriş parçacığına denk gelen ortalama atalet momenti atanarak çözüm yapılmıştır. Bu kapsamda 1 ve 3 numaralı kirişler $0{,}2\ m$ aralıklara bölünmüştür. 2 numaralı kirişin ise değişken atalet momentli uçları $0{,}1\ m$ aralıklara bölünmüş, ortadaki sabit ataletli bölüm tek bir parça olarak modellenmiştir.

Eğilme momenti dağılımı (Şekil 5.14)'teki gibi elde edilmiştir.



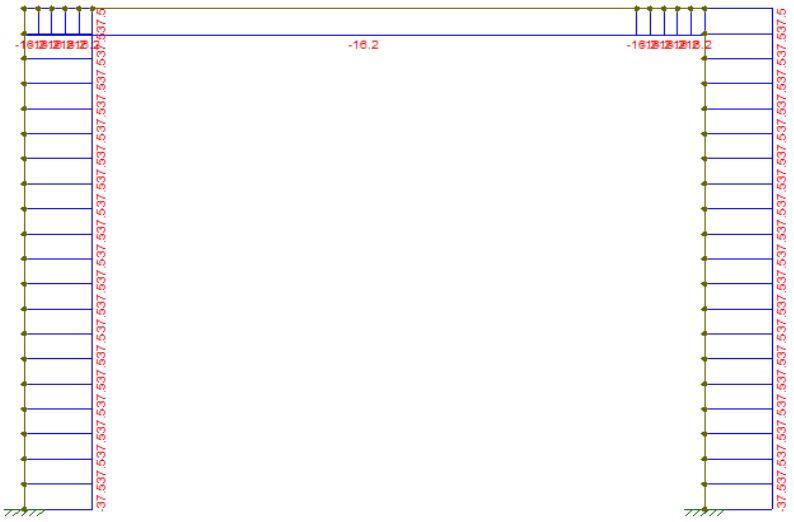
Şekil 5.14 : Örnek çerçevenin Ftool adındaki programla çözümünden elde edilen moment dağılımı.

Kesme kuvveti (Şekil 5.15)'teki gibi elde edilmiştir.



Şekil 5.15 : Örnek çerçevenin Ftool adındaki programla çözümünden elde edilen kesme kuvveti dağılımı.

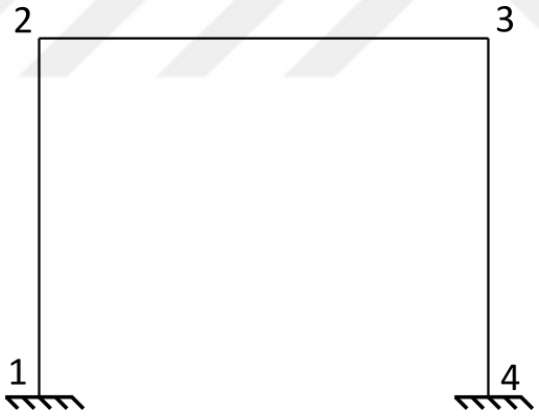
Kesme kuvveti (Şekil 5.16)'daki gibi elde edilmiştir.



Şekil 5.16 : Örnek çerçevenin Ftool adındaki programla çözümünden elde edilen eksenel kuvvet dağılımı.

5.1.3 Sonuçların karşılaştırılması

Çerçeve sisteminin düğüm noktaları (Şekil 5.17)'deki gibi numaralandırılmıştır.



Şekil 5.17 : Örnek çerçevenin düğümlerinin numaralandırılması.

Çizelge 5.1'de (Şekil 5.17)'de yapılan numaralandırmaya göre düğümlerde elde edilen momentler karşılaştırılmıştır.

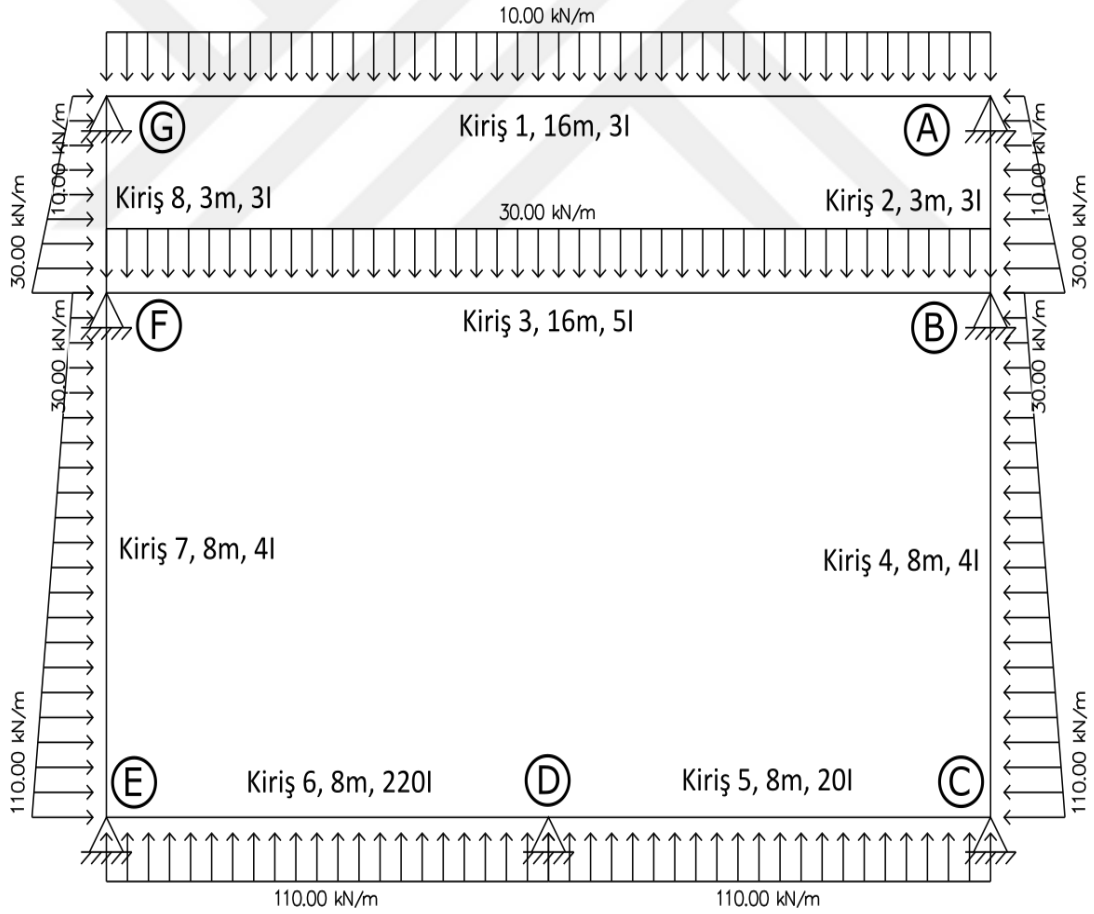
Çizelge 5.1 : Örnek çerçevenin düğüm momentlerinin değişik çözüm yöntemleriyle karşılaştırılması.

Çözüm Yöntemi	Düğüm numaraları			
	1	2	3	4
MesnetMD	9.9775	28.0407	28.0407	9.9776
Ftool	10.0	28.0	28.0	10.0
Analitik	9.978	28.04	28.04	9.978

Burada MesnetMD, bu tez çalışması kapsamında geliştirilen programın proje adıdır. Görüldüğü gibi programın çıktıları hem analitik çözüm ile hem de Ftool programıyla uyumludur.

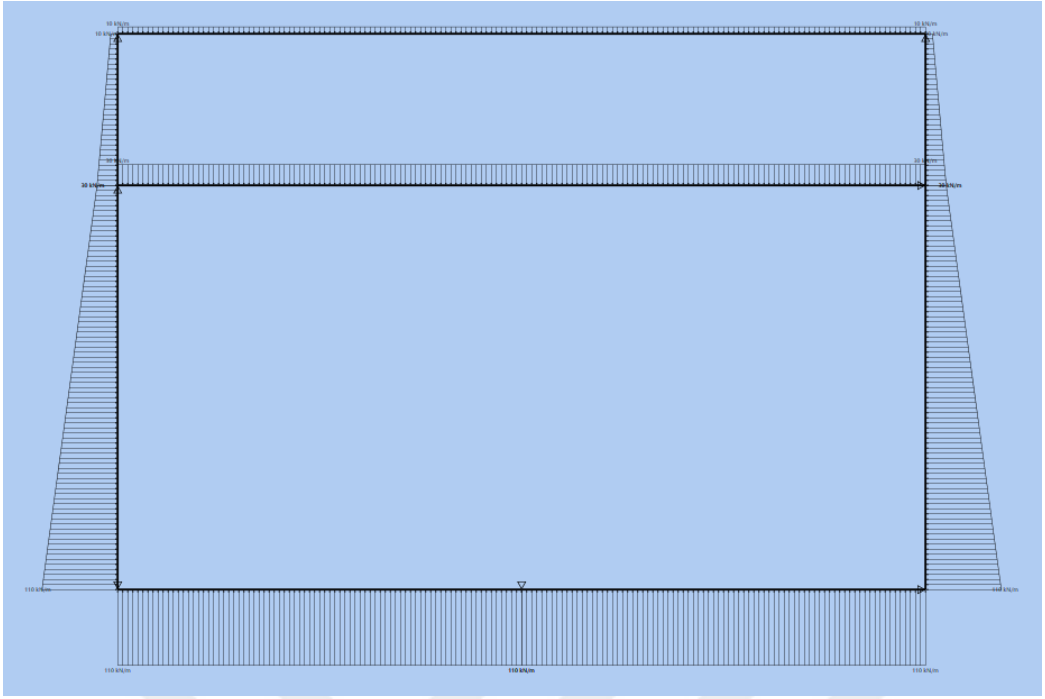
5.2 Sabit Kesitli Gemi Enine Çerçeve Örneği

Ele alınacak çerçeve sistemi (Şekil 5.18)'de gösterilmiştir. Bütün kirişlerin uzunluğu, atalet momenti ve kirişleri etkiyen yükler ve mesnetlerin isimleri şekilde gösterilmiştir. Bütün düğümler basit mesnet olarak kabul edilmiştir. Bu durumda eksenel deplasmanlar olmayacağı için çerçeve kirişlerinin alanları önemsizdir ve hesaplarda birim alan kullanılmıştır. Örnek çerçeve ara güverteli bir gemi çerçevesidir. Dip yapısındaki merkez iç omurgadan dolayı bu nokta basit mesnet olarak alınmıştır. Gemi çerçevesinin bordalarına ve dibine etkiyen hidrostatik basınç yayılı yükler olarak temsil edilmiştir.



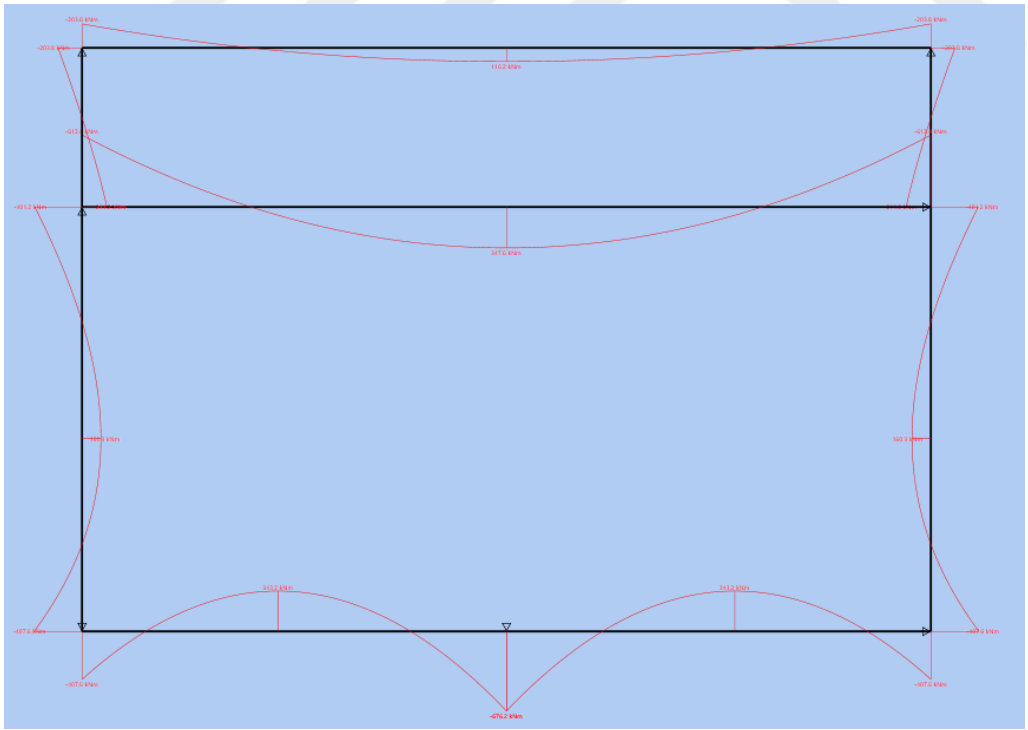
Şekil 5.18 : Ara güverteli gemi çerçevesi.

Programda bu çerçeve oluşturulmuştur. Çerçevenin görünümü (Şekil 5.19)'da gösterilmiştir.



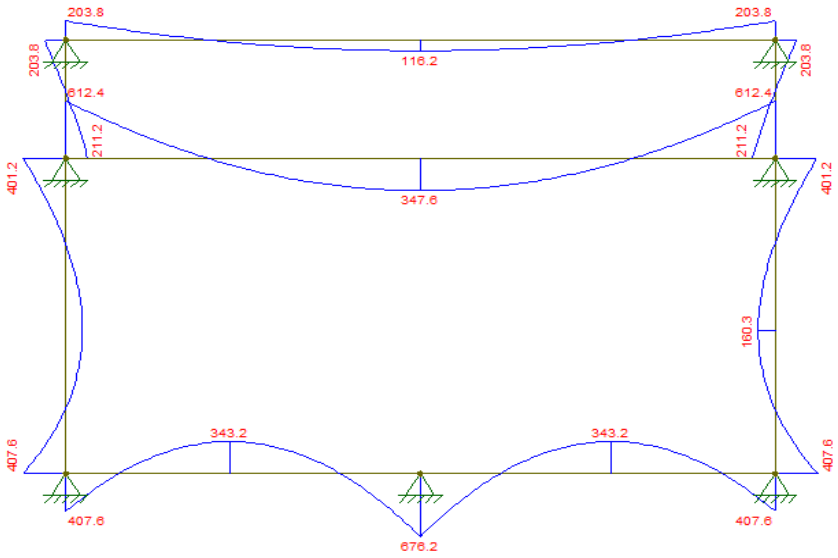
Şekil 5.19 : Ara güverteli gemi çerçevesinin programdaki görünüşü.

Programda oluşturulan bu gemi enine çerçevesinin çözümü yapılmış ve (Şekil 5.20)'deki moment dağılımı elde edilmiştir.



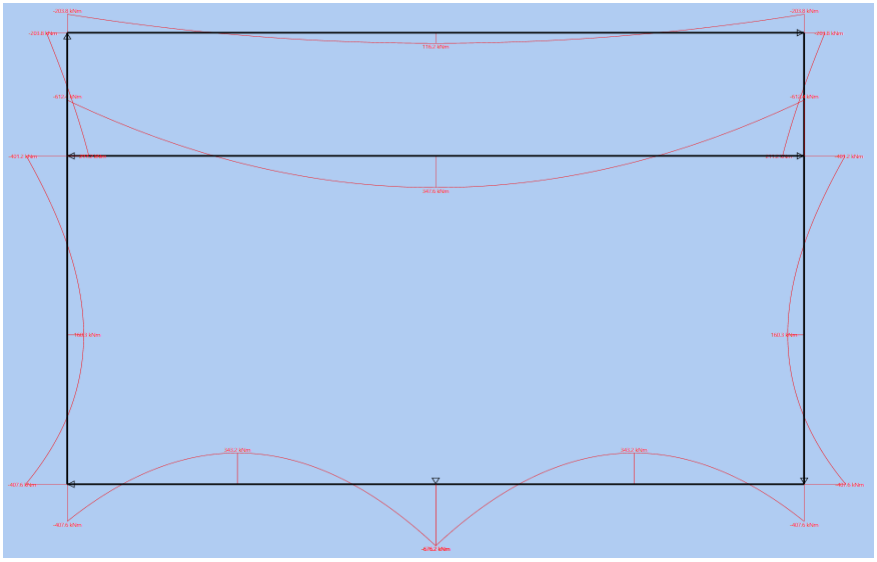
Şekil 5.20 : Ara güverteli gemi çerçevesinin programdaki moment dağılımı.

Aynı gemi çerçevesi Ftool adlı programda da çözülmüş ve (Şekil 5.21)'deki moment dağılımı elde edilmiştir.



Şekil 5.21 : Ara güverteli gemi çerçevesinin Ftool adlı programdaki moment dağılımı.

Yine aynı gemi çerçevesi Mesnet adlı programda çözülmüştür. Mesnet, 2 boyutlu gemi enine çerçevelerini Cross Metodu ile çözen analiz programıdır (Birler, 2018). Serbestlik dereceleri Matris Deplasman Metodu ile aynı olduğu durumda aynı sonuç vermesi beklenir. Mesnet, kullandığı metot gereği düğümlerde sadece dönme serbestlik derecesi olmasına müsaade eder. Bu gemi enine çerçevesinde bütün düğümler basit mesnetlidir. Dolayısıyla düğümlerde sadece dönme serbestlik derecesi vardır. Mesnet programındaki bu çerçeve için elde edilen moment dağılımı (Şekil 5.22)'deki gibi elde edilmiştir.



Şekil 5.22 : Ara güverteli gemi çerçevesinin Mesnet adlı programdaki moment dağılımı.

Görüldüğü gibi Mesnet programı arayüzü bu tez kapsamında geliştirilen programa çok benzemektedir. Bu çalışmada geliştirilen program yazarın daha önce lisans bitirme tezi kapsamında yaptığı Mesnet adlı analiz programı üzerine kurulmuştur. Arayüz kodları büyük oranda aynıdır.

5.2.1 Sonuçların karşılaştırılması

Karşılaştırmada her kirişin düğümlere karşılık gelen momentleri baz alınmıştır. Çerçeve 3 ayrı program ile çözülmüştür. Çizelge 5.2'de (Şekil 5.18)'de gösterilen düğüm noktalarında oluşan momentlerin karşılaştırılması verilmiştir.

Çizelge 5.2 : Örnek çerçevenin düğüm momentlerinin değişik çözüm yöntemleriyle karşılaştırılması.

Çözüm Programı	Düğümler						
	A1	A2	B2	B3	B4	C4	C5
Mesnet	203,7811	203,7811	211,2261	612,378	401,1519	407,6151	407,6151
Ftool	203,8	203,8	211,2	612,4	401,2	407,6	407,6
MesnetMD	203,7811	203,7811	211,2261	612,378	401,1519	407,6151	407,6151

Çözüm Programı	Düğümler						
	D5	D6	E6	E7	F7	F3	F8
Mesnet	676,1924	676,1924	407,6151	407,6151	401,1519	612,378	211,2261
Ftool	676,2	676,2	407,6	407,6	401,2	612,4	211,2
MesnetMD	676,1924	676,1924	407,6151	407,6151	401,1519	612,378	211,2261

Görüldüğü gibi programın bu çerçeve için çıktıları hem Mesnet programıyla ile hem de Ftool programıyla uyumludur. Burada Ftool, sonuçları virgülden sonra 1 basamak hassasiyetle vermektedir.

6. SONUÇ VE ÖNERİLER

Bu çalışmada gemi enine çerçevelerinin yapısal analizini Matris Deplasman Metodu ile yapan bir bilgisayar programı geliştirilmesi amaçlanmıştır. Geliştirilen programın gemi enine çerçevelerinde sık karşılaşılan, kesitin değişken olması durumlarında dahi çerçeve çözümü yapabilmesi hedeflenmiştir. Aynı zamanda geliştirilen programın modern uygulama programlama arayüzlerini kullanması, görsel arayüze sahip olması ve kullanımının kolay olması istenmiştir.

Matris Deplasman Metodu değişken kesitli olarak ifade edilmiş ve Üç Moment Yöntemi yardımıyla değişken kesitli durumlarda yük vektörünün hesaplanması anlatılmıştır. Çalışmadaki teori kullanılarak C# programlama diliyle kullanıcı arayüzüne sahip bir program geliştirilmiştir. Programın çıktıları analitik çözümle ve diğer muadil programlarla karşılaştırılarak doğru çözüm yaptığı ispatlanmıştır.

Sonuçta gemilerin ön dizaynında kullanılabilecek pratik ve kullanışlı, Windows işletim sistemlerinde çalışabilen bir bilgisayar yazılımı ortaya çıkmıştır. Program geliştirilen matematik sınıfları sayesinde mümkün olduğunda analitik olarak hesap yapabilmektedir. Böylece sabit kesitli çerçeveleri çok hızlı bir şekilde hesaplayabilmektedir. Program gerektiğinde paralel hesaplama tekniklerini de kullanarak gelişen bilgisayar teknolojilerini mümkün olduğunca kullanmakta ve çözüm süresini minimum düzeye indirebilmektedir.

Program aynı zamanda nesne yönelimli programlama yaklaşımlarını kullanmaktadır. Böylece geliştirici dışında başka kişilerin programın kaynak kodlarına baktığında kolayca anlayabilmesi sağlanmaktadır.

Programın Matris Deplasman Metodu çözümleriyle alakalı bazı kaynak kodları bu çalışmanın ek belgelerine konulmuştur. İleride program belirli bir seviyede geliştirildiği zaman kaynak kodları açılacak ve dünyadaki uygulama geliştiricilerinin programı anlamasına ve geliştirmesine olanak tanınacaktır.

Programda matematiksel altyapısı hazır olmasına karşın arayüzde ve programın işleyişinde karşılaşılan bazı zorluklar nedeniyle bazı özelllikler henüz

uygulanmamıştır. Düğümlere direkt yük ekleme özelliği yine bu nedenlerden dolayı henüz uygulanmamıştır. Tekil moment ekleme, mafsal gibi başka mesnetlerin eklenebilme özelliği de yine uygulanmamıştır.

İleride bu bahsedilen özellikler kodlanarak programa kazandırılacaktır. Sehim grafikleri çizebilme yeteneği yine programa eklenecektir. Optimizasyonlar yapılarak çözüm süresi kısaltılacak ve hafıza yönetimleri iyileştirilerek programın ram kullanımı azaltılacaktır.

Programa ileride titreşim analizinin de eklenmesi planlanmaktadır. Bunların dışında bu programdan elde edilen bilgiler ve geliştirilen kod kütüphanesi kullanılarak 3 boyutlu uzay çerçeve sistemi çözümü yapabilen bilgisayar programı geliştirilmesi hedeflenmektedir. Böylece gemi bir bütün olarak uzay çerçeve olarak modellenebilecek ve daha iyi yapısal analizlerin yapılabilmesine olarak tanınacaktır.

KAYNAKLAR

Alikhani, M. (2010, Mayıs 24). *Polynomial.Net.* Aralık 8, 2016 tarihinde Code Project: https://www.codeproject.com/articles/83394/polynomial-net adresinden alındı

Bayraktarkatal, E. (1995, Nisan). Gemi Elemanlarının Boyutlandırılması için Gerilme Analizi (Doktora Tezi). İstanbul Teknik Üniversitesi, İstanbul.

Birler, Ö. (2018, Kasım 8). *MESNET: 2 DIMENSIONAL FRAME ANALYZER WITH MOMENT DISTRIBUTION METHOD.* Bitbucket Version Control Website: https://bitbucket.org/omerbirler/mesnet adresinden alındı

C# Language Specification. (2006, Haziran). Aralık 12, 2016 tarihinde European Computer Manufacturers Association International: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf adresinden alındı

Courant, R. (1943). Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society,* 1-23.

Crahmaliuc, R. (2018, Nisan 17). 75 *Years of the Finite Element Method (FEM).* SimScale: https://www.simscale.com/blog/2015/11/75-years-of-the-finite-element-method-fem/ adresinden alındı

Davis, A. (2011, Mart 21). *A WPF Custom Control for Zooming and Panning.* Aralık 9, 2016 tarihinde Code Project: https://www.codeproject.com/Articles/85603/A-WPF-custom-control-for-zooming-and-panning adresinden alındı

Felippa, C. (2017, Aralık 17). Introduction to Finite Element Methods. University of Colorado:https://www.colorado.edu/engineering/CAS/courses.d/IFEM. d/IFEM.AppO.d/IFEM.AppO.pdf adresinden alındı

Fertis, D. G. (1997). *Infrastructure Systems: Mechanics, Design, and Analysis of Components.* New york: John Wiley & Sons, Inc.

Floris. (2009, Nisan 1). *DrawCurve / AddCurve for WPF / Cardinal Spline.* Aralık 15, 2016 tarihinde a coder named Floris: http://floris.briolas.nl/floris/2009/04/addcurve-for-wpf-cardinal-spline/comment-page-1/ adresinden alındı

Gavin, H. P. (2009). *CEE 201L. Uncertainty, Design, and Optimization*. Aralık 15, 2016 tarihinde Department of Civil and Environmental Engineering: people.duke.edu/~hpgavin/cee201/three-moment.pdf adresinden alındı

Güler, M. S., & Şen, S. (2015). Sonlu Elemanlar Yöntemi Hakkında Genel Bilgiler. *Ordu Üniversitesi Bilim ve Teknoloji Dergisi,* 56-66.

Hrennikoff, A. (1941). Solution of problems of elasticity by the framework method. *Journal of applied mechanics*, 169-175.

Karaduman, A. (1993, Şubat 16). *Değişken kesitli düzlem taşıyıcı sistemlerin matris deplasman yöntemi ile statik çözümü.* Selçuk Üniversitesi Dijital Arşiv Sistemi: http://acikerisim.selcuk.edu.tr:8080/xmlui/bitstream/handle/123456789/2443/029281.pdf?sequence=1&isAllowed=y adresinden alındı

Limbrunner, G. F., & Spiegel, L. (2009). *Applied Statics and Strength of Materials* (5. b.). New Jerseu, United States of America: Pearson Education International.

Microsoft. (2018, Kasım 14). *double (C# Reference).* Microsoft Docs: https://docs.microsoft.com/tr-tr/dotnet/csharp/language-reference/keywords/double adresinden alındı

Omurtag, M. H. (2010). *Çubuk Sonlu Elemanlar.* İstanbul: Birsen Yayınevi.

Savcı, M. (1988). Clapeyron Denklemleri. M. Savcı içinde, *Gemi Kirişleri Mukavemeti* (s. 12). İstanbul: İstanbul Teknik Üniversitesi Gemi İnşaatı ve Deniz Bilimleri Fakültesi Ofset Baskı Atölyesi.

Tezcan, S. (1970). *Çubuk Sistemlerin Elektronik Hesap Makineleri İle Çözümü.* İstanbul: Arı Kitabevi Matbaası.

EKLER

EK A: Programdaki Matematik Tabanlı Kodlar.

EK B: Programdaki Mukavemet Elemanları Kodları.

EK C: Matris Deplasman Metodu Global Sistemi Çözücü Kodları.

EK A: Programdaki Matematik Tabanlı Kodlar

```
public class Poly
{
        public Poly(string PolyExpression)
        {
            this._Terms = new TermCollection();
            this.ReadPolyExpression(PolyExpression);
        }
        public Poly(string PolyExpression, double startpoint, double endpoint)
        {
            this._Terms = new TermCollection();
            this.ReadPolyExpression(PolyExpression);
            _startpoint = startpoint;
            _endpoint = endpoint;

        }
        public double Calculate(double x)
        {
            double result = 0;
            foreach (Term t in this.Terms)
            {
                result += t.Coefficient *System.Math.Pow(x, t.Power);
            }
            return result;

        }
        public static bool ValidateExpression(string Expression)
        {
            if (Expression.Length == 0)
                return false;

            Expression = Expression.Trim();
            Expression = Expression.Replace(" ", "");
              while (Expression.IndexOf("--") > -1 | Expression.IndexOf("++") >
              -1 | Expression.IndexOf("^^") > -1 | Expression.IndexOf("xx") > -
              1)
            {
                Expression = Expression.Replace("--", "-");
                Expression = Expression.Replace("++", "+");
                Expression = Expression.Replace("^^", "^");
                Expression = Expression.Replace("xx", "x");
            }
            string ValidChars = "+-x1234567890^.E";
            bool result = true;
            foreach (char c in Expression)
            {
                if (ValidChars.IndexOf(c) == -1)
                {
                    result = false;
                }
            }
            return result;

        }
        private void handleterm(string term)
        {
            Term termitem;
            if (term.Contains("E"))
            {
                var coeffs = term.Split('E');
```

```
            double c = Convert.ToDouble(coeffs[0]);
            if (coeffs[1].Contains("x^"))
            {
                var epxs= coeffs[1].Split(new string[] { "x^" },
                StringSplitOptions.None);
                double p1 = Convert.ToDouble(epxs[0]);
                double p2 = Convert.ToDouble(epxs[1]);
                termitem = new Term();
                termitem.Coefficient = c * System.Math.Pow(10, p1);
                termitem.Power = p2;
                Terms.Add(termitem);
            }
            else if(coeffs[1].Contains("x"))
            {
                                    var epxs = coeffs[1].Split(new
                string[] { "x" }, StringSplitOptions.None);
                double p1 = Convert.ToDouble(epxs[0]);
                termitem = new Term();
                termitem.Coefficient = c * System.Math.Pow(10, p1);
                termitem.Power = 1;
                Terms.Add(termitem);
            }
            else
            {
                double p = Convert.ToDouble(coeffs[1]);
                var termItem = new Term();
                termItem.Coefficient = c * System.Math.Pow(10, p);
                Terms.Add(termItem);
            }
        }
        else
        {
            termitem = new Term(term);
            Terms.Add(termitem);
        }

    }
    public Poly Integrate()
    {
        var terms = new TermCollection();
        foreach (Term t in this.Terms)
        {
            var pow = t.Power + 1;
            var coeff = t.Coefficient / (t.Power + 1);
            terms.Add(new Term(pow, coeff));
        }
        return new Poly(terms);

    }
    public double DefiniteIntegral(double start, double end)
    {
        double result = 0;

        Poly integral = Integrate();

        result = integral.Calculate(end) - integral.Calculate(start);

        return result;

    }
    public Poly Derivate()
    {
```

```
            var terms = new TermCollection();
            foreach (Term t in this.Terms)
            {
                var pow = t.Power - 1;
                var coeff = t.Coefficient * t.Power;
                terms.Add(new Term(pow, coeff));
            }

            return new Poly(terms);
        }
}
public bool IsLinear()
{
    if (Terms.Count > 2)
    {
        return false;
    }

    foreach (Term term in Terms)
    {
            if (System.Math.Abs(term.Power - 1.0) > 0.000001 &&
          System.Math.Abs(term.Power) > 0.000001)
        {
            return false;
        }
    }

    return true;
}
public static Poly operator +(Poly p1, Poly p2)
{
    if (p1.ToString() == "0")
    {
        return p2;
    }
    else if (p2.ToString() == "0")
    {
        return p1;
    }

    Poly result = new Poly(p1.ToString());
    foreach (Term t in p2.Terms)
        result.Terms.Add(t);
    return result;
}
...
}


public class Term
{
        public Term(double power,double coefficient)
        {
            this.Power = power;
            this.Coefficient = coefficient;

        }
```

```
        public Term(string TermExpression)
         {
             if (TermExpression.Length > 0)
             {
                 if (TermExpression.IndexOf("x^") > -1)
                 {
                     string CoefficientString = TermExpression.Substring(0,
                     TermExpression.IndexOf("x^"));
                     int IndexofX = TermExpression.IndexOf("x^");
                     string PowerString = TermExpression.Substring(IndexofX +
                     2, (TermExpression.Length -1) - (IndexofX + 1));
                     if (CoefficientString == "-")
                         this.Coefficient = -1;
                     else if (CoefficientString == "+" | CoefficientString ==
                     "")
                         this.Coefficient = 1;
                     else
                         this.Coefficient = double.Parse(CoefficientString);

                     this.Power = double.Parse(PowerString);
                 }
                 else if (TermExpression.IndexOf("x") > -1)
                 {
                     this.Power = 1;
                     string CoefficientString = TermExpression.Substring(0,
                     TermExpression.IndexOf("x"));
                     if (CoefficientString == "-")
                         this.Coefficient = -1;
                     else if (CoefficientString == "+" | CoefficientString ==
                     "")
                         this.Coefficient = 1;
                     else
                         this.Coefficient = double.Parse(CoefficientString);
                 }
                 else
                 {
                     this.Power = 0;
                     this.Coefficient = double.Parse(TermExpression);
                 }
             }
             else
             {
                 this.Power = 0;
                 this.Coefficient = 0;
             }

         }
...
}
public class PiecewisePoly:CollectionBase
{

        public PiecewisePoly(List<Poly> polylist)
         {
             initialize(polylist);

         }

        private void initialize(List<Poly> polylist)
         {
             _sortlist = polylist;
```

```
        _sortlist.Sort((a, b) => a.StartPoint.CompareTo(b.StartPoint));

        foreach (Poly poly in _sortlist)
        {
            List.Add(poly);
        }
    }

    public void Add(Poly poly)
    {
        List.Add(poly);
        Sort();
    }

    public PiecewisePoly Integrate()
    {
        var ppoly = new PiecewisePoly();
        foreach (Poly poly in List)
        {
            var ply = poly.Integrate();
            ply.StartPoint = poly.StartPoint;
            ply.EndPoint = poly.EndPoint;
            ppoly.Add(ply);
        }
        return ppoly;
    }

    public PiecewisePoly Derivate()
    {
        var ppoly = new PiecewisePoly();
        foreach (Poly poly in List)
        {
            var ply = poly.Derivate();
            ply.StartPoint = poly.StartPoint;
            ply.EndPoint = poly.EndPoint;
            ppoly.Add(ply);
        }
        return ppoly;
    }

    public double DefiniteIntegral(double start, double end)
    {
        double value = 0;
        foreach (Poly poly in List)
        {
            if (poly.StartPoint >= start || poly.EndPoint <= end)
            {
                double left = 0;
                double right = 0;
                if (poly.StartPoint >= start)
                {
                    left = poly.StartPoint;
                }
                else if (poly.StartPoint < start)
                {
                    left = start;
                }

                if (poly.EndPoint <= end)
                {
```

```
                    right = poly.EndPoint;
                }
                else if (poly.EndPoint > end)
                {
                    right = end;
                }

                if (left >= end)
                {
                    break;
                }
                value += poly.DefiniteIntegral(left, right);
            }
        }

        return value;

    }

    public double Calculate(double x)
    {
        double value = 0;
        foreach (Poly poly in List)
        {
            if (x >= poly.StartPoint && x <= poly.EndPoint)
            {
                value = poly.Calculate(x);
                return value;
            }
        }
        return value;

    }

    public bool IsConstant()
    {
        if (List.Count == 1)
        {
            var poly = List[0] as Poly;
            if (poly.IsConstant())
            {
                return true;
            }
        }
        return false;

    }

}

public static class Algebra
{

    public static double[] LinearEquationSolver(double[,] coefficients,
    double[] results)
    {
          if (coefficients.GetLength(0) != coefficients.GetLength(1) &&
          coefficients.GetLength(0) != results.Length)
        {
            throw new ArgumentException("Different array sizes");
        }

        int count = coefficients.GetLength(0);
```

```
        for (int i = 0; i < count - 1; i++)
        {
            for (int j = i + 1; j < count; j++)
            {
                double s = coefficients[j, i] / coefficients[i, i];
                for (int k = i; k < count; k++)
                {
                    coefficients[j, k] -= coefficients[i, k] * s;
                }
                results[j] -= results[i] * s;
            }
        }

        for (int i = count - 1; i >= 0; i--)
        {
            results[i] /= coefficients[i, i];
            coefficients[i, i] /= coefficients[i, i];
            for (int j = i - 1; j >= 0; j--)
            {
                double s = coefficients[j, i] / coefficients[i, i];
                coefficients[j, i] -= s;
                results[j] -= results[i] * s;
            }
        }

        return Enumerable.Range(0, count).Select(i => results[i] /
        coefficients[i, i]).ToArray();

    }

    public static double[] DotProduct(double[,] m, double[] v)
    {
        if (m.GetLength(1) != v.GetLength(0))
        {
            throw new InvalidOperationException("This Matrix and this
          vector can not be multiplied!");
        }

        var result = new double[m.GetLength(0)];
        double sum = 0;

       for (int i = 0; i < m.GetLength(0); i++)
        {
            sum = 0;
            for (int j = 0; j < m.GetLength(1); j++)
            {
                sum += m[i, j] * v[j];
            }

            result[i] = sum;
        }

        return result;
    }
    public static double[] AddVectors(double[] v1, double[] v2)
    {
        if (v1.GetLength(0) != v2.GetLength(0))
        {
            throw new InvalidOperationException("These two vectors can not
          be added");
        }
        var result = new double[v1.GetLength(0)];
```

```
            for (int i = 0; i < v1.GetLength(0); i++)
            {
                result[i] = v1[i] + v2[i];
            }

            return result;
        }
        public static double[,] MultiplyMatrix(double[,] m1, double[,] m2)
        {
            int rm1 = m1.GetLength(0);
            int cm1 = m1.GetLength(1);
            int rm2 = m2.GetLength(0);
            int cm2 = m2.GetLength(1);
            double temp = 0;
            double[,] result = new double[rm1, cm2];
            if (cm1 != rm2)
            {
                 throw new InvalidOperationException("This matrices can not be
              multiplied!");
            }
            else
            {
                for (int i = 0; i < rm1; i++)
                {
                    for (int j = 0; j < cm2; j++)
                    {
                        temp = 0;
                        for (int k = 0; k < cm1; k++)
                        {
                            temp += m1[i, k] * m2[k, j];
                        }
                        result[i, j] = temp;
                    }
                }
                return result;
            }
        }
        public static double[,] Transpose(double[,] matrix)
        {
            int w = matrix.GetLength(0);
            int h = matrix.GetLength(1);

            double[,] result = new double[h, w];

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    result[j, i] = matrix[i, j];
                }
            }

            return result;

        }

...

}

public class SimpsonsFirstIntegrator : SimpsonBase
    {
```

```
     public SimpsonsFirstIntegrator(double deltax) :
    base(Global.SimpsonIntegrationType.First, deltax)
     {
     }

     public override void Calculate()
     {
         for (int i = 0; i < datas.Count; i++)
         {
             if (i == 0)
             {
                 _sum += datas[i];
             }
             else if (i == datas.Count - 1)
             {
                 _sum += datas[i];
             }
             else if (i % 2 == 0)
             {
                 _sum += 2 * datas[i];
             }
             else if (i % 2 == 1)
             {
                 _sum += 4 * datas[i];
             }
         }
         _result = _h / 3 * _sum;
     }
    ...
}
```

EK B: Programdaki Mukavemet Elemanları Kodları

```
public class Beam : SomItem, ISomItem
{
        public Beam(double length)

         {
             InitializeComponent(length);

         }

        public void AddInertia(PiecewisePoly inertiappoly)
         {
             _inertiappoly = inertiappoly;
             _izero = _inertiappoly.PreciseMin;
             _maxinertia = _inertiappoly.Max;

         }

        public void AddElasticity(double elasticitymodulus)
         {
             _elasticity = elasticitymodulus;

         }

        public void AddArea(PiecewisePoly areappoly)
         {
             _areappoly = areappoly;
             _azero = _areappoly.PreciseMin;
             _maxarea = _areappoly.Max;

         }

        /// <summary>
        /// Adds the distributed load to beam with specified direction.
        /// </summary>
        ///  <param  name="loadppoly">The  desired  distributed  load  piecewise
        polynomial.</param>

        public void AddLoad(PiecewisePoly loadppoly)
         {
             _distributedloads = loadppoly;
             _maxdistload = _distributedloads.Max;
             _maxabsdistload = _distributedloads.MaxAbs;

         }

         /// <summary>
         /// Adds the concentrated load to beam with specified direction.
         /// </summary>
         /// <param name="load">The desired list of concentrated load key value
        pair.</param>
         public void AddLoad(KeyValueCollection loadpairs)
         {
             _concentratedloads = loadpairs;
             _maxconcload = _concentratedloads.YMax;
             _maxabsconcload = _concentratedloads.YMaxAbs;

         }

        public void Calculate()
         {
```

```
        //First, find reaction forces for concentrated loads
        findconcentratedsupportforces();

        //Then, find reaction forces for distributed loads
        finddistributedsupportforces();

        //Then, find shearForce distribution for concentrated loads in
        zero case (when both side of the beam bound with free supports
        findconcentratedzeroforce();

        //Then, find shearForce distribution for distributed loads in zero
        case
        finddistributedzeroforce();

        //Super position to find resultant zero shearForce distribution
        _zeroforceppoly = _zeroforceconcpploy + _zeroforcedistppoly;

        //Then, find zero memont distribution according to resultant zero
        shearForce distribution
        findzeromoment();

        //Check if analytical solution is possible
        canbesolvedanalytically();

        //Then, find end moments for each beam based on cross support
        cases (assumes supports connecting
        //more than on beams as fixed support
        mdsupportcases();

        //Then, find fixed end moments according to calculated end moments
        findfixedendmomentclapeyron();

        //Unlike cross methos, we need to have fixedendforce before the
        solution so that the shearForce vector can be calculated.
        updateforces();

        //Create base stiffness coefficients
        createbasestiffnescoefficients();

        //Create transformation matrix
        createtransformationmatrix();

        //Create element stiffness matrix
        createstiffnessmatrix();

        //Create element shear force vector
        createforcevector();

    }

    private void findconcentratedsupportforces()
    {
        double resultantforce = 0;
        double resultantforcedistance = 0;
        double multiply = 0;

        if (_concentratedloads?.Count > 0)
        {
            //Moment from left support point
            double leftmoment = 0;
           foreach (KeyValuePair<double, double> force in
      _concentratedloads)
            {
```

```
                leftmoment += force.Key * force.Value;
            }
            _rightsupportforceconc = leftmoment / _length;

            //Moment from right support point
            double rightmoment = 0;
           foreach (KeyValuePair<double, double> force in
          _concentratedloads)
            {
                rightmoment += (_length - force.Key) * force.Value;
            }
            _leftsupportforceconc = rightmoment / _length;
        }
        else
        {
            _leftsupportforceconc = 0;
            _rightsupportforceconc = 0;
        }
    }

    private void finddistributedsupportforces()
    {
        double resultantforce = 0;
        double resultantforcedistance = 0;
        double multiply = 0;

        if (_distributedloads?.Count > 0)
        {
            var forcelist = new List<KeyValuePair<double, double>>();

            foreach (Poly load in _distributedloads)
            {
                var forces = load.CalculateMagnitudeAndLocation();
                forcelist.AddRange(forces);
            }

            //Moment from left support point
            double leftmoment = 0;
            foreach (var force in forcelist)
            {
                leftmoment += force.Key * force.Value;
            }
            _rightsupportforcedist = leftmoment / _length;

            //Moment from right support point
            double rightmoment = 0;
            foreach (var force in forcelist)
            {
                rightmoment += (_length - force.Key) * force.Value;
            }
            _leftsupportforcedist = rightmoment / _length;
        }
        else
        {
            _leftsupportforcedist = 0;
            _rightsupportforcedist = 0;
        }
    }

    private void findconcentratedzeroforce()
    {
```

```
        _zeroforceconcpploy = new PiecewisePoly();

        if (_concentratedloads?.Count > 0)
        {
            double leftforce = _leftsupportforceconc;

            if (_concentratedloads[0].Key > 0)
            {
                var poly1 = new Poly(leftforce.ToString());
                poly1.StartPoint = 0;
                poly1.EndPoint = _concentratedloads[0].Key;
                _zeroforceconcpploy.Add(poly1);
            }

            for (int i = 0; i < _concentratedloads.Count; i++)
            {
                leftforce = leftforce - _concentratedloads[i].Value;

                var poly = new Poly(leftforce.ToString());

                poly.StartPoint = _concentratedloads[i].Key;
                if (i + 1 < _concentratedloads.Count)
                {
                    poly.EndPoint = _concentratedloads[i + 1].Key;
                }
                else
                {
                    poly.EndPoint = _length;
                }

                _zeroforceconcppoly.Add(poly);
            }
        }

    }

    private void finddistributedzeroforce()
    {
        _zeroforcedistppoly = new PiecewisePoly();

        if (_distributedloads?.Count > 0)
        {
            if (_distributedloads[0].StartPoint != 0)
            {
                var ply = new Poly(_leftsupportforcedist.ToString());
                ply.StartPoint = 0;
                ply.EndPoint = _distributedloads[0].StartPoint;
                _zeroforcedistppoly.Add(ply);
            }

            foreach (Poly load in _distributedloads)
            {
                var index = _distributedloads.IndexOf(load);

                double weightsbefore = findforcebefore(index);

                if (index > 0)
                {
                       if (_distributedloads[index - 1].EndPoint !=
                       _distributedloads[index].StartPoint)
                    {
                        var ply = new Poly(weightsbefore.ToString());
```

```
                    ply.StartPoint = _distributedloads[index -
                   1].EndPoint;
                    ply.EndPoint =
                   _distributedloads[index].StartPoint;
                    _zeroforcedistppoly.Add(ply);
                }
            }

            var poly = new Poly();

            var integration = load.Integrate();
            var zerovalue =
            load.Integrate().Calculate(load.StartPoint);
            if (zerovalue != 0)
            {
                if (weightsbefore != 0)
                {
                   poly = new Poly(weightsbefore.ToString()) -
                   integration + new Poly(zerovalue.ToString());
                }
                else
                {
                   poly = -1 * integration + new
                   Poly(zerovalue.ToString());
                }
            }
            else
            {
                if (weightsbefore != 0)
                {
                   poly = new Poly(weightsbefore.ToString()) -
                   integration;
                }
                else
                {
                    poly = -1 * integration;
                }
            }
            poly.StartPoint = load.StartPoint;
            poly.EndPoint = load.EndPoint;
            _zeroforcedistppoly.Add(poly);
        }
        _zeroforcedistppoly.Sort();

        if (_distributedloads.Last().EndPoint != _length)
        {
            var weights = findforcebefore(_distributedloads.Count);
            var ply = new Poly(weights.ToString());
            ply.StartPoint = _distributedloads.Last().EndPoint;
            ply.EndPoint = _length;
            _zeroforcedistppoly.Add(ply);
        }
    }

}

private void findzeromoment()
{
    _zeromomentppoly = new PiecewisePoly();

    foreach (Poly force in _zeroforceppoly)
    {
        var index = _zeroforceppoly.IndexOf(force);
```

```
            var poly = new Poly();
            var integration = force.Integrate();
            var momentsbefore = findmomentbefore(index);
            var zerovalue = force.Integrate().Calculate(force.StartPoint);
            var constant = momentsbefore - zerovalue;

            if (constant != 0)
            {
                poly = integration + new Poly(constant.ToString());
            }
            else
            {
                poly = integration;
            }

            poly.StartPoint = force.StartPoint;
            poly.EndPoint = force.EndPoint;
            _zeromomentppoly.Add(poly);
            _zeromomentppoly.Sort();
        }
    }

    private void canbesolvedanalytically()
    {
        //Check inertia ppoly has only one poly
        if (_inertiappoly.Count > 1)
        {
            _analyticalsolution = false;
            return;
        }

        //Check if inertia ppoly is constant or not dependant on x
        if (_inertiappoly.Degree() > 0)
        {
            _analyticalsolution = false;
            return;
        }

        //Check if zero moment ppoly has any term with non-integer power
        if (_zeromomentppoly.Count > 0)
        {
            foreach (Poly poly in _zeromomentppoly)
            {
                foreach (Term term in poly.Terms)
                {
                    if (term.Power % 1 != 0)
                    {
                        _analyticalsolution = false;
                        return;
                    }
                }
            }
        }
        _analyticalsolution = true;
    }

    private void mdsupportcases()
    {
          //In matrix displacement method, the force vector is calculated
          when all of the displacement are zero (both side is fixed
          support)
```

```
if (_zeromomentppoly.Count > 0)
{
    double ma1 = 0;
    double ma2 = 0;
    double mb1 = 0;
    double mb2 = 0;
    double r1 = 0;
    double r2 = 0;

    ////////////////////////////////////////////////////////
    /////////////////Left Equation Solve////////////////////
    ////////////////////////////////////////////////////////

    var xsquare = new Poly("x^2");
    xsquare.StartPoint = 0;
    xsquare.EndPoint = _length;

    var x = new Poly("x");
    x.StartPoint = 0;
    x.EndPoint = _length;

    var xppoly = new PiecewisePoly();
    xppoly.Add(x);

    if (_analyticalsolution)
    {
        //When the inertia distribution is constant dont waste
        time and cpu with simpson numerical integration, integrate
        it analytically.
        //Since izero equals inertia the expression can be
        simplified
        ma1 = _length / 3;
        mb1 = _length / 2 - ma1;
        var moxp = _zeromomentppoly.Propagate(_length) * xppoly;
        r1 = -1 / _length * moxp.DefiniteIntegral(0, _length);
        ma2 = _length / 6;
        mb2 = _length / 3;
        var mox = _zeromomentppoly * xppoly;
        r2 = -1 / _length * mox.DefiniteIntegral(0, _length);
    }
    else
    {
        //When the inertia distribution is not constant, there is
        no choice but to use numerical integration
        //since the integration can not be solved analytically
        using polynomials in this program.
        var conjugateinertia = _inertiappoly.Conjugate(_length);
        var simpson1 = new
        SimpsonsFirstIntegrator(Config.SimpsonStep);

        for (double i = 0; i <= _length; i = i +
        Config.SimpsonStep)
        {
               simpson1.AddData(_izero /
               conjugateinertia.Calculate(i) *
               xsquare.Calculate(i));
        }

        simpson1.Calculate();
        ma1 = 1 / System.Math.Pow(_length, 2) * simpson1.Result;

        ////////////////////////////////////////////////////////
```

```
                    var simpson2 = new
                    SimpsonsFirstIntegrator(Config.SimpsonStep);

                    for (double i = 0; i <= _length; i = i +
                    Config.SimpsonStep)
                    {
                           simpson2.AddData(_izero /
                           conjugateinertia.Calculate(i) * x.Calculate(i));
                    }

                    simpson2.Calculate();

                    var value1 = 1 / _length * simpson2.Result;

                    mb1 = value1 - ma1;

////////////////////////////////////////////////////////

                    var simpson3 = new
                    SimpsonsFirstIntegrator(Config.SimpsonStep);

                    var conjugatemoment = _zeromomentppoly.Conjugate(_length);

                    for (double i = 0; i <= _length; i = i +
                    Config.SimpsonStep)
                    {

                           simpson3.AddData(conjugatemoment.Calculate(i) *
                           _izero / conjugateinertia.Calculate(i) *
                           x.Calculate(i));
                    }
                    simpson3.Calculate();
                    r1 = -1 / _length * simpson3.Result;

////////////////////////////////////////////////////////
/////////////////Right Equation Solve///////////////////////

////////////////////////////////////////////////////////

                    var simpson4 = new
                    SimpsonsFirstIntegrator(Config.SimpsonStep);

                    for (double i = 0; i <= _length; i = i +
                    Config.SimpsonStep)
                    {
                           simpson4.AddData(_izero /
                           _inertiappoly.Calculate(i) * xsquare.Calculate(i));
                    }
                    simpson4.Calculate();
                    var value2 = 1 / System.Math.Pow(_length, 2) *
                    simpson4.Result;

                    var simpson5 = new
                    SimpsonsFirstIntegrator(Config.SimpsonStep);

                    for (double i = 0; i <= _length; i = i +
                    Config.SimpsonStep)
                    {
                           simpson5.AddData((_izero /
                           _inertiappoly.Calculate(i)) * xppoly.Calculate(i));
                    }
                    simpson5.Calculate();
```

```
                    ma2 = 1 / _length * simpson5.Result - value2;

////////////////////////////////////////////////////////////
                    mb2 = value2;
////////////////////////////////////////////////////////////

                    var simpson6 = new
                    SimpsonsFirstIntegrator(Config.SimpsonStep);

                    for (double i = 0; i <= _length; i = i +
                    Config.SimpsonStep)
                    {

                        simpson6.AddData(_zeromomentppoly.Calculate(i) *
                        (_izero / _inertiappoly.Calculate(i)) *
                        xppoly.Calculate(i));
                    }

                    simpson6.Calculate();
                    r2 = -1 / _length * simpson6.Result;
                }

                double[,] coefficients =
                {
                    {ma1, mb1},
                    {ma2, mb2},
                };

                double[] results =
                {
                    r1, r2
                };

                ////////////////////////////////////////////////////////

                var moments =
                MesnetMD.Classes.Math.Algebra.LinearEquationSolver(coefficient
                s, results);
                _ma = moments[0];
                _mb = moments[1];
            }
            else
            {
                _ma = 0;
                _mb = 0;
            }

        }
        private void findfixedendmomentclapeyron()
         {
            var polylist = new List<Poly>();

            var constant = (_mb - _ma) / _length;

            if (System.Math.Abs(constant) < 0.00000001)
            {
                constant = 0.0;
            }

            foreach (Poly moment in _zeromomentppoly)
            {
                var resultpoly = new Poly();
```

```
            resultpoly.StartPoint = moment.StartPoint;
            resultpoly.EndPoint = moment.EndPoint;
                var mapoly = new Poly(_ma.ToString(), moment.StartPoint,
                moment.EndPoint);
            var xpoly = new Poly("x", moment.StartPoint, moment.EndPoint);

            if (!constant.Equals(0.0))
            {
                var cpoly = new Poly(constant.ToString(),
                moment.StartPoint, moment.EndPoint);
                resultpoly = moment + mapoly + xpoly * cpoly;
            }
            else
            {
                resultpoly = moment + mapoly;
            }
            resultpoly.StartPoint = moment.StartPoint;
            resultpoly.EndPoint = moment.EndPoint;
            polylist.Add(resultpoly);
        }
        _fixedendmomentppoly = new PiecewisePoly(polylist);
    }

    private void updateforces()
     {
         _fixedendforceppoly = _fixedendmomentppoly.Derivate();

         _maxforce = _fixedendforceppoly.Max;

         _maxabsforce = _fixedendforceppoly.MaxAbs;

         _minforce = _fixedendforceppoly.Min;
      }

    private void createbasestiffnescoefficients()
     {
         if (_inertiappoly.IsConstant())
         {
             _mii = 4;
             _mjj = 4;
             _mij = 2;
         }
         else
         {
             var x = new Poly("x");
             var xsquare = new Poly("x^2");

             var simpson1 = new
             SimpsonsFirstIntegrator(Config.SimpsonStep);

             for (double i = 0; i <= _length; i = i + Config.SimpsonStep)
             {
                 simpson1.AddData(xsquare.Calculate(i) /
                 _inertiappoly.Calculate(i));
             }
             simpson1.Calculate();
             double i1 = simpson1.Result;

             var simpson2 = new
             SimpsonsFirstIntegrator(Config.SimpsonStep);
             for (double i = 0; i <= _length; i = i + Config.SimpsonStep)
             {
```

```
                simpson2.AddData(1 / _inertiappoly.Calculate(i));
            }
            simpson2.Calculate();
            double i2 = simpson2.Result;

            var simpson3 = new
            SimpsonsFirstIntegrator(Config.SimpsonStep);

            for (double i = 0; i <= _length; i = i + Config.SimpsonStep)
            {
                simpson3.AddData(x.Calculate(i) /
                _inertiappoly.Calculate(i));
            }
            simpson3.Calculate();
            double i3 = simpson3.Result;

            double det = -_izero / _length * (i1 * i2 -
            System.Math.Pow(i3, 2));

            _mii = -i1 / det;
                _mjj = (-System.Math.Pow(_length, 2) * i2 + 2 * _length *
                i3 - i1) / det;
            _mij = -(_length * i3 - i1) / det;
        }
        if (_areappoly.IsConstant())
        {
            _nii = 1;
        }
        else
        {
             var simpson4 = new
          SimpsonsFirstIntegrator(Config.SimpsonStep);

            for (double i = 0; i <= _length; i = i + Config.SimpsonStep)
            {
                simpson4.AddData(1 / _areappoly.Calculate(i));
            }
            simpson4.Calculate();
            double i4 = simpson4.Result;
            _nii = _length / (_azero * i4);
        }
    }

    private void createtransformationmatrix()
    {
        _transformationmatrix = new double[6,6];

        _transformationmatrix[0, 0] = Algebra.CosD(_angle);
        _transformationmatrix[0, 1] = Algebra.SinD(_angle);
        _transformationmatrix[0, 2] = 0;
        _transformationmatrix[0, 3] = 0;
        _transformationmatrix[0, 4] = 0;
        _transformationmatrix[0, 5] = 0;
        _transformationmatrix[1, 0] = -Algebra.SinD(_angle);
        _transformationmatrix[1, 1] = Algebra.CosD(_angle);
        _transformationmatrix[1, 2] = 0;
        _transformationmatrix[1, 3] = 0;
        _transformationmatrix[1, 4] = 0;
        _transformationmatrix[1, 5] = 0;
        _transformationmatrix[2, 0] = 0;
        _transformationmatrix[2, 1] = 0;
        _transformationmatrix[2, 2] = 1;
```

```
        _transformationmatrix[2, 3] = 0;
        _transformationmatrix[2, 4] = 0;
        _transformationmatrix[2, 5] = 0;
        _transformationmatrix[3, 0] = 0;
        _transformationmatrix[3, 1] = 0;
        _transformationmatrix[3, 2] = 0;
        _transformationmatrix[3, 3] = Algebra.CosD(_angle);
        _transformationmatrix[3, 4] = Algebra.SinD(_angle);
        _transformationmatrix[3, 5] = 0;
        _transformationmatrix[4, 0] = 0;
        _transformationmatrix[4, 1] = 0;
        _transformationmatrix[4, 2] = 0;
        _transformationmatrix[4, 3] = -Algebra.SinD(_angle);
        _transformationmatrix[4, 4] = Algebra.CosD(_angle);
        _transformationmatrix[4, 5] = 0;
        _transformationmatrix[5, 0] = 0;
        _transformationmatrix[5, 1] = 0;
        _transformationmatrix[5, 2] = 0;
        _transformationmatrix[5, 3] = 0;
        _transformationmatrix[5, 4] = 0;
        _transformationmatrix[5, 5] = 1;
    }

    private void createstiffnessmatrix()
    {
        var E = _elasticity;
         var I = _izero * System.Math.Pow(10, -8); //Conversion from cm^4
         to m^4
        var A = _azero * System.Math.Pow(10, -4); //Conversion from cm^2
        to m^2
        var L = _length;
        var L2 = System.Math.Pow(_length, 2);
        var L3 = System.Math.Pow(_length, 3);
        var nii = _nii;
        var mii = _mii;
        var mjj = _mjj;
        var mij = _mij;

        _stiffnessmatrix = new double[6, 6];
        _stiffnessmatrix[0, 0] = nii * E * A / L;
        _stiffnessmatrix[0, 1] = 0;
        _stiffnessmatrix[0, 2] = 0;
        _stiffnessmatrix[0, 3] = -nii * E * A / L;
        _stiffnessmatrix[0, 4] = 0;
        _stiffnessmatrix[0, 5] = 0;
        _stiffnessmatrix[1, 0] = 0;
        _stiffnessmatrix[1, 1] = (mii + mjj + 2 * mij) * E * I / L3;
        _stiffnessmatrix[1, 2] = (mii + mij) * E * I / L2;
        _stiffnessmatrix[1, 3] = 0;
        _stiffnessmatrix[1, 4] = -(mii + mjj + 2 * mij) * E * I / L3;
        _stiffnessmatrix[1, 5] = (mjj + mij) * E * I / L2;
        _stiffnessmatrix[2, 0] = 0;
        _stiffnessmatrix[2, 1] = (mii + mij) * E * I / L2;
        _stiffnessmatrix[2, 2] = mii * E * I / L;
        _stiffnessmatrix[2, 3] = 0;
        _stiffnessmatrix[2, 4] = -(mii + mij) * E * I / L2;
        _stiffnessmatrix[2, 5] = mij * E * I / L;
        _stiffnessmatrix[3, 0] = -nii * E * A / L;
        _stiffnessmatrix[3, 1] = 0;
        _stiffnessmatrix[3, 2] = 0;
        _stiffnessmatrix[3, 3] = nii * E * A / L;
        _stiffnessmatrix[3, 4] = 0;
```

```
        _stiffnessmatrix[3, 5] = 0;
        _stiffnessmatrix[4, 0] = 0;
        _stiffnessmatrix[4, 1] = -(mii + mjj + 2 * mij) * E * I / L3;
        _stiffnessmatrix[4, 2] = -(mii + mij) * E * I / L2;
        _stiffnessmatrix[4, 3] = 0;
        _stiffnessmatrix[4, 4] = (mii + mjj + 2 * mij) * E * I / L3;
        _stiffnessmatrix[4, 5] = -(mjj + mij) * E * I / L2;
        _stiffnessmatrix[5, 0] = 0;
        _stiffnessmatrix[5, 1] = (mjj + mij) * E * I / L2;
        _stiffnessmatrix[5, 2] = mij * E * I / L;
        _stiffnessmatrix[5, 3] = 0;
        _stiffnessmatrix[5, 4] = -(mjj + mij) * E * I / L2;
        _stiffnessmatrix[5, 5] = mjj * E * I / L;

        //kXYZ=T^T.kxyz.T => Matrix transformation

        var T = _transformationmatrix;
        var TT = Algebra.Transpose(T);

        var m1 = Algebra.MultiplyMatrix(TT, _stiffnessmatrix);
        var m2 = Algebra.MultiplyMatrix(m1, T);
        _stiffnessmatrix = m2;
    }

    private void createforcevector()
    {
        createindirectforcevector();
        createdirectforcevector();
        _forcevector = new double[6];
    }

    private void createindirectforcevector()
    {
        _idforcevector = new double[6];
        //Converting kN to N by multiplying with 1000
          _idforcevector[0] = -_fixedendforceppoly.Calculate(0) *
          Math.Algebra.SinD(_angle) * 1000;
          _idforcevector[1] = _fixedendforceppoly.Calculate(0) *
          Math.Algebra.CosD(_angle) * 1000;
        _idforcevector[2] = -_fixedendmomentppoly.Calculate(0) * 1000;
          _idforcevector[3] = _fixedendforceppoly.Calculate(_length) *
          Math.Algebra.SinD(_angle) * 1000;
        _idforcevector[4] = -_fixedendforceppoly.Calculate(_length) *
        Math.Algebra.CosD(_angle) * 1000;
        _idforcevector[5] = _fixedendmomentppoly.Calculate(_length) *
        1000;
        }
    }

    public void PostProcessUpdate()
    {
        obtainlocalforces();
        updatemoments();
        updateforces();
        updateaxialforces();
        if (_stressanalysis)
        {
            updatestresses();
        }
    }
    private void obtainlocalforces()
    {
```

```
        _localforcevector = Algebra.DotProduct(_transformationmatrix,
        _forcevector);
    }

    private void updatemoments()
    {
        double constant = 0;

        if (_zeromomentppoly.Length > 0)
        {
            var ma = -_localforcevector[2] / 1000;

            var mb = _localforcevector[5] / 1000;

            constant = (mb - ma) / _length;
            var terms = new TermCollection();
            if (constant != 0)
            {
                if (ma != 0)
                {
                    var term1 = new Term(1, constant);
                    var term2 = new Term(0, ma);
                    terms.Add(term1);
                    terms.Add(term2);
                }
                else
                {
                    var term1 = new Term(1, constant);
                    terms.Add(term1);
                }
            }
            else
            {
                if (ma != 0)
                {
                    var term2 = new Term(0, ma);
                    terms.Add(term2);
                }
                else
                {
                    //nothing to do
                }
            }

            if (terms.Length > 0)
            {
                var mdpoly = new Poly(terms, 0, _length);
                var polies = new List<Poly>();
                polies.Add(mdpoly);
                var mdpolies = new PiecewisePoly(polies);

                _fixedendmomentppoly = _zeromomentppoly + mdpolies;
            }
            else
            {
                _fixedendmomentppoly = _zeromomentppoly;
            }
        }
        else
        {
            //There is no load on this beam
            var ma = -_localforcevector[2] / 1000;
```

```
            var mb = _localforcevector[5] / 1000;

            constant = (mb - ma) / _length;
            var terms = new TermCollection();
            if (constant != 0)
            {
                if (ma != 0)
                {
                    var term1 = new Term(1, constant);
                    var term2 = new Term(0, ma);
                    terms.Add(term1);
                    terms.Add(term2);
                }
                else
                {
                    var term1 = new Term(1, constant);
                    terms.Add(term1);
                }
            }
            else
            {
                if (ma != 0)
                {
                    var term2 = new Term(0, ma);
                    terms.Add(term2);
                }
            }

            if (terms.Length > 0)
            {
                var mdpoly = new Poly(terms, 0, _length);
                var polies = new List<Poly>();
                polies.Add(mdpoly);
                var mdpolies = new PiecewisePoly(polies);

                _fixedendmomentppoly = mdpolies;
            }
            else
            {
                var termzero = new Term(0, 0);
                var termszero = new TermCollection();
                termszero.Add(termzero);
                var polyzero = new Poly(termszero);
                var zeropolies = new List<Poly>();
                zeropolies.Add(polyzero);
                var zeroppoly = new PiecewisePoly(zeropolies);
                _fixedendmomentppoly = zeroppoly;
            }
        }


        _maxmoment = _fixedendmomentppoly.Max;
        _maxabsmoment = _fixedendmomentppoly.MaxAbs;
        _minmoment = _fixedendmomentppoly.Min;
    }

    private void updateforces()
    {
        _fixedendforceppoly = _fixedendmomentppoly.Derivate();
        _maxforce = _fixedendforceppoly.Max;
        _maxabsforce = _fixedendforceppoly.MaxAbs;
        _minforce = _fixedendforceppoly.Min;
```

```
 }

private void updateaxialforces()
 {
     var terms = new TermCollection();
     var ma = -_localforcevector[0] / 1000;
     var mb = _localforcevector[3] / 1000;
     double constant = (mb - ma) / _length;
     if (System.Math.Abs(constant) > System.Math.Pow(10, -8))
     {
         if (System.Math.Abs(ma) > System.Math.Pow(10, -8))
         {
             var term1 = new Term(1, constant);
             var term2 = new Term(0, ma);
             terms.Add(term1);
             terms.Add(term2);
         }
         else
         {
             var term1 = new Term(1, constant);
             terms.Add(term1);
         }
     }
     else
     {
         if (System.Math.Abs(ma) > System.Math.Pow(10, -8))
         {
             var term1 = new Term(0, ma);
             terms.Add(term1);
         }
         else
         {
             var term1 = new Term(0, 0);
             terms.Add(term1);
         }
     }

     var mdpoly = new Poly(terms, 0, _length);
     _axialforceppoly = new PiecewisePoly(mdpoly);
     _maxaxialforce = _axialforceppoly.Max;
     _maxabsaxialforce = _axialforceppoly.MaxAbs;
     _minaxialforce = _axialforceppoly.Min;
 }

private void updatestresses()
 {
     double precision = 0.001;
     _stress = new KeyValueCollection();
     double stress = 0;
     double y = 0;
     double e = 0;
     double d = 0;
     for (int i = 0; i < _length / precision; i++)
     {
         e = _eppoly.Calculate(i * precision);
         d = _dppoly.Calculate(i * precision);
         if (e > d - e)
         {
             y = e;
         }
         else
         {
```

```
                    y = d - e;
                }
              stress = System.Math.Pow(10, 3) *
              _fixedendmomentppoly.Calculate(i * precision) * y /
              (_inertiappoly.Calculate(i * precision));
                _stress.Add(i * precision, stress);
            }

            if (!_stress.ContainsKey(_length))
            {
                e = _eppoly.Calculate(_length);
                d = _dppoly.Calculate(_length);
                if (e > d - e)
                {
                    y = e;
                }
                else
                {
                    y = d - e;
                }
              stress = System.Math.Pow(10, 3) *
              _fixedendmomentppoly.Calculate(_length) * y /
              (_inertiappoly.Calculate(_length));
                _stress.Add(_length, stress);
            }
            _maxstress = _stress.YMax;
            _maxabsstress = _stress.YMaxAbs;
        }

       public void UpdateDirectForceVector(double[] diplacement)
        {
            var cross = Algebra.DotProduct(_stiffnessmatrix, diplacement);
            var result = Algebra.AddVectors(cross, _idforcevector);
            _forcevector = result;
        }
       ...
}

public class SupportItem : SomItem
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="SupportItem"/> class.
        /// All types of supports are derived from this class
        /// </summary>
        public SupportItem(Global.ObjectType type)
        {
            Type = type;
            if (type != Global.ObjectType.FictionalSupport)
            {
                SupportId = supportcount++;
            }

            switch (type)
            {
                case Global.ObjectType.BasicSupport:
                    DOFCount = 1;
                    break;
                case Global.ObjectType.SlidingSupport:
                    DOFCount = 2;
                    break;
                case Global.ObjectType.FictionalSupport:
                    DOFCount = 3;
```

```
                    break;
                case Global.ObjectType.RightFixedSupport:
                    DOFCount = 0;
                    break;
                case Global.ObjectType.LeftFixedSupport:
                    DOFCount = 0;
                    break;
            }

            DegreeOfFreedoms = new List<DOF>();
        }
        ...
    }

public class BasicSupport : SupportItem, ISomItem, ISupportItem,
IFreeSupportItem
    {
        public BasicSupport() : base(ObjectType.BasicSupport)
        {
            InitializeComponent();
            Members = new List<Member>();
            Name = "Basic Support " + SupportId;
            var rdof = new DOF(Global.DOFType.Rotational);
            DegreeOfFreedoms.Add(rdof);
        }
       public void AddBeam(Beam beam, Global.Direction direction)
       {
       if (!Members.Contains(member))
            {
                Members.Add(member);

                if (Members.Count == 1)
                {
                    switch (direction)
                    {
                        case Direction.Left:
                            Canvas.SetLeft(this, beam.LeftPoint.X - Width/2);
                            Canvas.SetTop(this, beam.LeftPoint.Y - Height);
                            beam.LeftSide = this;
                            //Add rotational dof member
                           var ldofmember = new DOFMember(beam,
                           DOFLocation.LeftRotational);
                            DegreeOfFreedoms[0].Members.Add(ldofmember);
                            break;
                        case Direction.Right:
                            Canvas.SetLeft(this, beam.RightPoint.X - Width/2);
                            Canvas.SetTop(this, beam.RightPoint.Y - Height);
                            beam.RightSide = this;
                            var rdofmember = new DOFMember(beam,
                           DOFLocation.RightRotational);

                            DegreeOfFreedoms[0].Members.Add(rdofmember);
                            break;
                    }
                }
                else
                {
                    switch (direction)
                    {
                        case Direction.Left:
                            beam.LeftSide = this;
                            beam.IsBound = true;
```

```
                        //Add rotational dof member
                        var ldofmember = new DOFMember(beam,
                        DOFLocation.LeftRotational);
                        DegreeOfFreedoms[0].Members.Add(ldofmember);
                        break;
                    case Direction.Right:
                        beam.RightSide = this;
                        beam.IsBound = true;
                        var rdofmember = new DOFMember(beam,
                        DOFLocation.RightRotational);
                        DegreeOfFreedoms[0].Members.Add(rdofmember);
                        break;
                }
            }
        }
    }
    ...
}

public class SlidingSupport : SupportItem, ISomItem, ISupportItem,
IFreeSupportItem
    {
        public SlidingSupport() : base(ObjectType.SlidingSupport)
        {
            InitializeComponent();
            Members = new List<Member>();
            Name = "Sliding Support " + SupportId;
            var hdof = new DOF(Global.DOFType.Horizontal);
            var rdof = new DOF(Global.DOFType.Rotational);
            DegreeOfFreedoms.Add(hdof);
            DegreeOfFreedoms.Add(rdof);
        }
        public void AddBeam(Beam beam, Global.Direction direction)
        {
            var member = new Member(beam, direction);
            if (!Members.Contains(member))
            {
                Members.Add(member);

                if (Members.Count == 1)
                {
                    switch (direction)
                    {
                        case Direction.Left:
                            Canvas.SetLeft(this, beam.LeftPoint.X - Width /
                            2);
                            Canvas.SetTop(this, beam.LeftPoint.Y - Height);
                            beam.LeftSide = this;
                            var lhdmember = new DOFMember(beam,
                            DOFLocation.LeftHorizontal);

                            DegreeOfFreedoms[0].Members.Add(lhdmember);
                            var lrdmember = new DOFMember(beam,
                            DOFLocation.LeftRotational);
                            DegreeOfFreedoms[1].Members.Add(lrdmember);
                            break;
                        case Direction.Right:
                            Canvas.SetLeft(this, beam.RightPoint.X - Width /
                            2);
                            Canvas.SetTop(this, beam.RightPoint.Y - Height);
                            beam.RightSide = this;
```

```
                        var rhdmember = new DOFMember(beam,
                        DOFLocation.RightHorizontal);
                        DegreeOfFreedoms[0].Members.Add(rhdmember);
                        var rrdmember = new DOFMember(beam,
                        DOFLocation.RightRotational);
                        DegreeOfFreedoms[1].Members.Add(rrdmember);
                        break;
                    }
                }
                else
                {
                    switch (direction)
                    {
                        case Direction.Left:
                            beam.LeftSide = this;
                            beam.IsBound = true;
                            var lhdmember = new DOFMember(beam,
                            DOFLocation.LeftHorizontal);
                            DegreeOfFreedoms[0].Members.Add(lhdmember);
                            var lrdmember = new DOFMember(beam,
                            DOFLocation.LeftRotational);
                            DegreeOfFreedoms[1].Members.Add(lrdmember);
                            break;
                        case Direction.Right:
                            beam.RightSide = this;
                            beam.IsBound = true;
                            var rhdmember = new DOFMember(beam,
                            DOFLocation.RightHorizontal);
                            DegreeOfFreedoms[0].Members.Add(rhdmember);
                            var rrdmember = new DOFMember(beam,
                            DOFLocation.RightRotational);
                            DegreeOfFreedoms[1].Members.Add(rrdmember);
                            break;
                    }
                }
            }
        }
        ...
}

public class LeftFixedSupport : SupportItem, ISomItem, ISupportItem,
IFixedSupportItem
{
        public LeftFixedSupport() : base(ObjectType.LeftFixedSupport)
        {
            InitializeComponent();
            Name = "Left Fixed Support " + SupportId;
        }
        public void AddBeam(Beam beam)
        {
            Canvas.SetLeft(this, beam.LeftPoint.X - Width);
            Canvas.SetTop(this, beam.LeftPoint.Y - Height/2);
            Member = new Member(beam, Direction.Left);
            beam.LeftSide = this;
        }
        ...
}

public class RightFixedSupport : SupportItem, ISomItem, ISupportItem,
IFixedSupportItem
{
        public RightFixedSupport() : base(ObjectType.RightFixedSupport)
```

```
        {
            InitializeComponent();
            Name = "Right Fixed Support " + SupportId;
        }
        public void AddBeam(Beam beam)
        {
            Canvas.SetLeft(this, beam.RightPoint.X);
            Canvas.SetTop(this, beam.RightPoint.Y - Height/2);
            Member = new Member(beam, Direction.Right);
            beam.RightSide = this;
            SetAngle(beam.Angle);
        }
        ...
}

public class FictionalSupport : SupportItem
{
        public FictionalSupport(): base(Global.ObjectType.FictionalSupport)
        {
            Members = new List<Member>();
            FID = fcount++;
            Name = "Fictional Support "+ FID;
            var hdof = new DOF(Global.DOFType.Horizontal);
            var vdof = new DOF(Global.DOFType.Vertical);
            var rdof = new DOF(Global.DOFType.Rotational);
            DegreeOfFreedoms.Add(hdof);
            DegreeOfFreedoms.Add(vdof);
            DegreeOfFreedoms.Add(rdof);
            Global.AddObject(this);
        }
        public void AddBeam(Beam beam, Global.Direction direction)
        {
            var member = new Member(beam, direction);
            if (!Members.Contains(member))
            {
                Members.Add(member);
                switch (direction)
                {
                    case Global.Direction.Left:
                        beam.LeftSide = this;
                           var lhdofmember = new DOFMember(beam,
                           Global.DOFLocation.LeftHorizontal);
                        DegreeOfFreedoms[0].Members.Add(lhdofmember);
                           var lvdofmember = new DOFMember(beam,
                           Global.DOFLocation.LeftVertical);
                        DegreeOfFreedoms[1].Members.Add(lvdofmember);
                           var lrdofmember = new DOFMember(beam,
                           Global.DOFLocation.LeftRotational);
                        DegreeOfFreedoms[2].Members.Add(lrdofmember);
                        break;

                    case Global.Direction.Right:
                        beam.RightSide = this;
                           var rhdofmember = new DOFMember(beam,
                           Global.DOFLocation.RightHorizontal);
                        DegreeOfFreedoms[0].Members.Add(rhdofmember);
                           var rvdofmember = new DOFMember(beam,
                           Global.DOFLocation.RightVertical);
                        DegreeOfFreedoms[1].Members.Add(rvdofmember);
                           var rrdofmember = new DOFMember(beam,
                           Global.DOFLocation.RightRotational);
                        DegreeOfFreedoms[2].Members.Add(rrdofmember);
```

```
                        break;
                    }
                }
            }
            ...
}
```

EK C: Matris Deplasman Metodu Global Sistemi Çözücü Kodları

```
public static class MDSolver
    {
        public static void Calculate()
        {
            createdofpairs();
            createglobalstiffnessmatrix();
            createglobalforcevector();
            solvethesystem();
            obtainbeamdisplacements();
        }

        private static void createdofpairs()
        {
            GlobalDofs = new List<DOF>();
            foreach (var pair in Global.Objects)
            {
                switch (pair.Value.Type)
                {
                    case Global.ObjectType.BasicSupport:
                        var bs = pair.Value as BasicSupport;
                        var rbdof = bs.DegreeOfFreedoms[0];
                        GlobalDofs.Add(rbdof);
                        break;

                    case Global.ObjectType.SlidingSupport:
                        var ss = pair.Value as SlidingSupport;
                        var hsdof = ss.DegreeOfFreedoms[0];
                        GlobalDofs.Add(hsdof);
                        var rsdof = ss.DegreeOfFreedoms[1];
                        GlobalDofs.Add(rsdof);
                        break;

                    case Global.ObjectType.FictionalSupport:
                        var fs = pair.Value as FictionalSupport;
                        var hfdof = fs.DegreeOfFreedoms[0];
                        GlobalDofs.Add(hfdof);
                        var vbdof = fs.DegreeOfFreedoms[1];
                        GlobalDofs.Add(vbdof);
                        var rfdof = fs.DegreeOfFreedoms[2];
                        GlobalDofs.Add(rfdof);
                        break;
                }
            }

            DofCount = GlobalDofs.Count;
        }

        private static void createglobalstiffnessmatrix()
        {
            GlobalStiffnessMatrix = new double[DofCount, DofCount];

            for (int i = 0; i < GlobalDofs.Count; i++)
            {
                for (int j = 0; j < GlobalDofs.Count; j++)
                {
                    double value = 0;
                    foreach (var member in GlobalDofs[i].Members)
                    {
                        var beam = member.Beam;
```

```
                int index = getbeamindex(GlobalDofs[j].Members, beam);
                if (index > -1)
                {
                    int row = (int) member.Location;
                    int col = (int)
                   GlobalDofs[j].Members[index].Location;
                    value += beam.StiffnessMatrix[row, col];
                }
            }
            GlobalStiffnessMatrix[i, j] = value;
        }
    }
}

private static void createglobalforcevector()
{
    GlobalForceVector = new double[DofCount];
    for (int i = 0; i < GlobalDofs.Count; i++)
    {
        double value = 0;
        foreach (var member in GlobalDofs[i].Members)
        {
            var beam = member.Beam;
            var index = (int) member.Location;
            value += beam.ForceVector[index];
        }
        GlobalForceVector[i] = value;
    }
}

private static void solvethesystem()
{
     GlobalDisplacementVector =
     MesnetMD.Classes.Math.Algebra.LinearEquationSolver(GlobalStiffnes
     sMatrix, GlobalForceVector);
}

private static void obtainbeamdisplacements()
{
    for (int j = 0; j < Global.Objects.Count; j++)
    {
        switch (Global.Objects[j].Type)
        {
            case Global.ObjectType.Beam:
                var displacement = new double[6];
                var beam = Global.Objects[j] as Beam;
                for (int i = 0; i < GlobalDofs.Count; i++)
                {
                    foreach (var member in GlobalDofs[i].Members)
                    {
                        if (Equals(beam, member.Beam))
                        {
                            int index = (int)member.Location;
                            displacement[index] =
                            GlobalDisplacementVector[i];
                            break;
                        }
                    }
                }

                beam.UpdateDirectForceVector(displacement);
                break;
```

```
                }
            }
        }
       ...
   }

public class DOF
    {
        public DOF(Global.DOFType type)
        {
            Type = type;
            Members = new List<DOFMember>();
        }
        public Global.DOFType Type;
        public List<DOFMember> Members;
    }

    public class DOFMember
    {
        public DOFMember(Beam beam, Global.DOFLocation location)
        {
            Beam = beam;
            Location = location;
        }
        public Beam Beam;
        public Global.DOFLocation Location;
    }
```

ÖZGEÇMİŞ

Ad-Soyad **:** Ömer BİRLER

Doğum Tarihi ve Yeri **:** 14/05/1992 - Bursa

E-posta **:** omer.birler@gmail.com

ÖĞRENİM DURUMU:

- **Lisans** **:** 2016, İstanbul Teknik Üniversitesi, Gemi İnşaatı ve Deniz Bilimleri Fakültesi, Gemi İnşaatı ve Gemi Makinaları Mühendisliği
- **Lisans** **:** 2018, İstanbul Teknik Üniversitesi, Makina Fakültesi, Makina Mühendisliği
- **Yüksek Lisans** **:** 2018, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Gemi İnşaatı ve Gemi Makinaları Mühendisliği