CEE6501 — Course Introduction

Matrix Structural Analysis (Official title)

Structural Form Finding and Matrix Analysis (Unofficial title)

Course Focus: From Analysis to Form Finding

Historically, this course was taught as a traditional **Matrix Structural Analysis** class, focused on analyzing structures with **fixed geometry**.

This semester, the course is reframed around a broader and more modern question:

> ***How do we use computational methods not only to analyze structures, but also to generate efficient structural forms?***

We will still cover classical matrix methods, but extend them to **nonlinear, geometry-driven, and equilibrium-based problems** that are directly relevant to:

- lightweight structures
- long-span systems
- cable nets and membranes
- gridshells and lattices
- funicular masonry and compression shells

Defining the Types of Analysis in this Course

Traditional Structural Analysis

Structural analysis asks:

> ***Given this geometry, how does the structure respond to loads?***

- Geometry is known
- Loads are applied
- Displacements and internal forces are computed
- Members are then sized for strength and serviceability

This is the dominant paradigm in most structural engineering education and practice.

Structural Form Finding

Form finding asks the inverse question:

> ***What geometry naturally carries these loads in equilibrium?***

- Connectivity and supports are given
- Forces or stress states, or other constraints, are prescribed
- Geometry is *solved for* the applied loads

Instead of checking whether a shape works, we compute a shape that works.

This can lead to:

- axial-force-dominated structures
- efficient use of material
- geometries that directly reflect force flow

Why This Matters

Certain structural systems are:

- geometrically complex
- highly flexible
- driven by fabrication and construction constraints

For these systems:

> ***Geometry, forces, and fabrication are tightly coupled.***

You cannot treat form as fixed and only analyze afterward — the form itself must be part of the solution.

How the Course Is Organized

Section 1 — Linear Analysis (Weeks 1–4)

Focus:

- Traditional Matrix Structural Analysis (MSA)
- Direct stiffness method
- Linear systems
- Trusses and space frames (axial elements)
- Plane frames (beam elements)

Purpose:

- refresh structural mechanics
- establish computational workflow
- build matrix intuition

Section 2 — Nonlinear Analysis and Form Finding (Weeks 6–9)

Focus:

- geometric nonlinearity
- large displacements
- equilibrium-driven geometry

Methods:

- Force Density Method
- Dynamic Relaxation
- nonlinear stiffness formulations

Purpose:

- shift from analysis of form → generation of form
- understand equilibrium as a geometric process

Section 3 — Miscellaneous Topics (Weeks 12–15)

Possible topics:

- graphical statics
- rigidity theory
- structural optimization
- guest lectures and applied workflows

Purpose:

- connect form, forces, and design
- prepare for final project applications

Structural Systems We Will Study

Traditional Systems (Review + Computational Grounding)

- Plane trusses
- Space trusses
- Plane frames
- Space frames

These are primarily analyzed using:

- linear stiffness methods
- small displacement assumptions

They provide the **foundation** for everything that follows.

Form-Active and Geometry-Driven Systems

- Cable nets
- Membranes
- Elastic rod networks
- Lattices and gridshells
- Tensegrity-like systems

These require:

- nonlinear equilibrium formulations
- geometry updates during solution
- iterative numerical methods

This is where **form finding becomes essential**.

Dual Learning Objectives of the Course

This class has two equally important goals.

🧠 Goal 1: Structural Theory and Form-Finding Methods

By the end of the course, you will be able to:

- explain the theoretical foundations of:
    - the Direct Stiffness Method
    - the Force Density Method
    - Dynamic Relaxation
    - graphical equilibrium methods (including graphic statics and rigidity concepts)
- identify the physical assumptions embedded in each method
- determine when a given method is appropriate for a specific structural system

The focus is not just on analysis, but on:

> ***understanding validity, limitations, and applicability.***

💻 Goal 2: Computational Implementation

By the end of the course, you will be able to:

- translate mathematical formulations into working Python implementations
- assemble and solve large-scale matrix systems numerically
- interpret structural behavior through visualization and post-processing
- apply version control to track, document, and collaborate on code
- communicate computational methods and results using Jupyter notebooks
- connect analysis models to parametric geometry workflows (Rhino/Grasshopper in later parts of the course)

The emphasis is not on using black-box analysis software, but on:

> ***understanding how to implement the algorithms.***

What You Should Expect From This Course

This is not a passive lecture course.

You will be expected to:

- follow along with live coding
- modify and extend notebook templates
- debug numerical methods
- think about structure as a computational system

Many assignments will require you to:

- translate equations into code
- verify behavior visually
- reflect on structural response

Brief Historical Context

From Classical Analysis → Matrix Methods → Form Finding

Classical Structural Analysis

- Developed for **hand calculations**
- Targeted specific systems:
    - trusses, beams, plane frames
- Based on:
    - equilibrium
    - compatibility
    - member force–deformation relations

Figure suggestion: Cremona force diagram (truss) or moment distribution sketch

Limits of Classical Methods

- Increasingly hard for:
    - indeterminate structures
    - many degrees of freedom
- Required manual solution of large equation systems
- Often relied on simplifying assumptions
- Not designed for systematic programming

Figure suggestion: multi-bay frame with DOF counting + hand-calculation table

Foundations and Precursors of Matrix Methods

- Maxwell (1864): the method of consistent deformations
- Maney (1915): slope-deflection
- Classical precursors of:
    - matrix flexibility
    - matrix stiffness
- Biggest disadvantage was these methods required the direct solution of simulataneous algebraic equations

Figure suggestion: slope-deflection joint schematic (moments/rotations/translations)

Computers Change Everything (1940s–1950s)

- Digital computers make large linear solves practical
- Rapid development of computational methods:
    - Levy (1947): matrix flexibility (generalizing the classical method of consistent deformations)
    - Livesley (1954): matrix stiffness
    - Turner, Clough, Martin, Topp (1956): direct stiffness method
    - Livesley (1956): nonlinear formulation of the stiffness method for frame stability analysis

Figure suggestion: global stiffness assembly pipeline (element → assemble → solve)

Matrix Methods vs Finite Element Methods

Both approaches are based on the same fundamentals:

- equilibrium
- compatibility
- constitutive (material) laws

But they differ in how the structure is **discretized** and how element behavior is defined.

Matrix (Direct Stiffness) Methods:

- structure modeled as nodes + **structural members**
- member stiffness from **exact analytical solutions** of beam/truss theory
- very efficient for framed systems (trusses, frames, space frames)
- relatively few degrees of freedom

Finite Element Method (FEM):

- structure modeled as nodes + **continuum elements**
- stiffness from assumed displacement (shape) functions and energy principles
- applicable to plates, shells, solids, and complex geometries
- accuracy depends on mesh density and element order

For linear framed structures:

- beam and truss finite elements use the **same stiffness matrices**
- global assembly and solution are identical
- FEM can be viewed as a **generalization** of matrix stiffness methods

Figure suggestion: frame discretization vs solid FE mesh

Physical Form Finding: Origins

Form finding is not a new idea — it predates digital computation.

Early designers used **physical systems** that naturally satisfy equilibrium:

- Gaudí's hanging chain models (catenary → compression form)
- Frei Otto's soap films and tensile models

These systems minimize:

- potential energy
- internal force imbalance

Equilibrium was found by **letting physics solve the problem**.

Figure suggestion: Gaudí chain model and inverted masonry arch

Computational Form Finding: Force Density Method

A major breakthrough came with:

- Heinz-Josef Schek (1974)
- Force Density = force / length prescribed per element

Key idea:

- transforms nonlinear equilibrium equations
- into a **linear system in nodal coordinates**

This makes form finding:

- fast
- robust
- well suited for cable nets and membranes

This was one of the first widely used **digital form-finding methods**.

Figure suggestion: cable net before/after equilibrium using FDM

Dynamic Relaxation

Dynamic Relaxation treats equilibrium as the steady state of a **fictitious dynamic system**:

- masses assigned to nodes
- damping added
- system is time-stepped until motion stops

Originally developed for:

- highly nonlinear problems
- large deformations
- complex boundary conditions

Used not only in structures, but also in:

- cloth simulation
- particle systems
- physical animation

Equilibrium is found by **numerical dissipation**, not matrix solves.

Figure suggestion: iterative relaxation sequence of cable or fabric mesh

Beyond Classical Form Finding

Modern computational design uses many related approaches:

- Thrust Network Analysis (compression-only funicular forms)
- Graphical statics (vector-based equilibrium constructions)
- Structural and topology optimization

Today, these are often integrated with:

- parametric modeling
- fabrication constraints
- robotic construction planning

Form finding is no longer **just** about shape — it is about **designing feasible construction processes**.

Figure suggestion: funicular shell with thrust network overlay

The Unifying Equation

Almost everything in this course — linear or nonlinear — starts from the same fundamental equilibrium statement:

$$\mathbf{K}\,\mathbf{u} = \mathbf{f}$$

$$\mathbf{K}\,\mathbf{u} = \mathbf{f}$$

Where:

- **K** = stiffness matrix
- **u** = displacement vector
- **f** = force vector

This equation encodes:

- equilibrium
- compatibility
- material behavior

What Changes Across the Course

In **Section 1 (Linear Analysis)**:

- **K** is constant
- geometry is fixed
- solution is direct

In **Section 2 (Nonlinear / Form Finding)**:

- **K** depends on geometry
- geometry depends on forces
- solution requires iteration

In **Section 3 (Miscellaeous Methods)**:

- equilibrium may be expressed in alternative forms
- geometry and forces may be coupled graphically or optimally

But the core idea — equilibrium through system equations — remains.

Big Picture Takeaway for Today

By the end of this course, you should be comfortable with:

- matrix-based structural formulations
- solving both linear and nonlinear equilibrium problems
- understanding when geometry must be part of the solution
- implementing structural methods computationally

Ultimately, the goal is to shift your mindset from:

> ***"analyze this structure"***
> *to*
> ***"compute a structure that satisfies equilibrium and design intent."***

That transition — from analysis to form generation — is at the heart of modern computational structural design.

Traditionally, structural methods ask:

> *Is this shape acceptable?*

They are used as:

> ***tools for evaluating a chosen form***

Geometry is assumed → performance is checked.

Form-finding methods instead ask:

> *What shape should this be?*

They are used as:

> ***tools for generating structural form***

Forces and constraints are prescribed → geometry is solved for.

In this course, you will learn methods that function as:

> ***both tools for analysis and tools for design***

where geometry is not just an input — it is an output of the computation.

Assignments and Homework Workflow

What a Homework Looks Like

- Most homework is a **Jupyter notebook** template
- You will:
    - fill in code
    - answer short prompts
    - generate plots/figures to verify behavior
- Grading emphasizes:
    - correctness
    - clarity
    - interpretation ("does this result make sense?")

Screenshot placeholder:

```
![Example homework notebook](images/hw_notebook_example.png)
```

Getting the Starter Files

- Homework templates will be distributed via the course repo / Canvas links
- Typical workflow:
    1. open the notebook (local or Colab)
    2. run the setup cell(s)
    3. complete tasks in order

Screenshot placeholder:

```
![Where to find assignments](images/hw_where_to_find.png)
```

Submitting Homework (Typical)

- You will submit on **Canvas**
- Most common submission formats:
    - a **PDF export** of the notebook, or
    - a **Colab link** (when specified)
- Each assignment will state the required format explicitly

Screenshot placeholder:

```
![Canvas submission page](images/hw_canvas_submit.png)
```

Collaboration and Integrity

- You may discuss concepts and debug strategies with classmates
- But submitted work must be your own:
    - do not copy/paste complete solutions
    - you should be able to explain every line you submit
- When in doubt: ask (office hours / forum)

Screenshot placeholder:

```
![Integrity policy snippet](images/hw_integrity_policy.png)
```