DOT/FAA/TC-24/13

Federal Aviation Administration
William J. Hughes Technical Center
Aviation Research Division
Atlantic City International Airport
New Jersey, 08405

Enhancing Pavement Assessment with Dynamic Backcalculation: A Dynamic Finite Element Approach

August 2024

Final Report

This document is available to the U.S. public through the National Technical Information Services (NTIS), Springfield, Virginia 22161.

This document is also available from the Federal Aviation Administration William J. Hughes Technical Center at actlibrary.tc.faa.gov.

U.S. Department of Transportation
Federal Aviation Administration

NOTICE

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for the contents or use thereof. The United States Government does not endorse products or manufacturers. Trade or manufacturer's names appear herein solely because they are considered essential to the objective of this report. The findings and conclusions in this report are those of the author(s) and do not necessarily represent the views of the funding agency. This document does not constitute FAA policy. Consult the FAA sponsoring organization listed on the Technical Documentation page as to its use.

This report is available at the Federal Aviation Administration William J. Hughes Technical Center's Full-Text Technical Reports page: actlibrary.tc.faa.gov in Adobe Acrobat portable document format (PDF).

Technical Report Documentation Page

1. Report No. DOT/FAA/TC-24/13	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle ENHANCING PAVEMENT ASSESSMENT WITH DYNAMIC BACKCALCULATION: A DYNAMIC FINITE ELEMENT APPROACH		5. Report Date August 2024 6. Performing Organization Code
7. Author(s) Elie Y. Hajj , Rami Skaff , Peter E. Sebaaly, Xiaolong Ma, Heyang Qin, and Feng Yan		8. Performing Organization Report No.
9. Performing Organization Name and Address Department of Civil and Environmental Engineering University of Nevada 664 North Virginia Street Reno, NV 89557		10. Work Unit No. (TRAIS) 11. Contract or Grant No. #692M15-20-T-00032
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Airport Engineering Division 800 Independence Ave., SW Washington, DC 20591		13. Type of Report and Period Covered Final Report 14. Sponsoring Agency Code AAS-110

15. Supplementary Notes

The Federal Aviation Administration Aviation Research Division COR was Matthew Brynick.

16. Abstract

The Heavy Weight Deflectometer(HWD)/Falling Weight Deflectometer (FWD) plays a crucial role in assessing pavement's structural condition. It consists of applying an impulse load to simulate a moving wheel and measuring pavement surface deflections. Pavement variables, such as moduli, are determined through backcalculation, which involves a forward model and optimization. The FAA currently employs static analysis in the BAKFAA software but encounters limitations, particularly with thick and stiff airfield pavements.

To address these challenges, a dynamic finite element (FE) model was developed under this study that incorporates subgrade damping to improve the reliability of backcalculation. The dynamic FE model was incorporated in a new tool called *PULSE*_FE to accelerate the computational runtime and improve efficiency. Dynamic backcalculation has been successfully applied to a Construction Cycle-9 flexible test item, and a parametric study has identified key FWD parameters for dynamic backcalculation.

Furthermore, an optimization framework was established to streamline the dynamic backcalculation process, seamlessly integrating modeling, FE analysis, and multiple optimizers. Constrained optimization was employed to enhance the practicality of solutions. Additionally, a user-friendly graphical user interface (GUI) program for BAKFAA Dynamic Backcalculation (DynaBAKFAA) was designed to simplify the tasks from mesh generation to dynamic backcalculation.

In summary, this project introduces a proficient FE model, advances in dynamic backcalculation, and an automated optimization framework, all poised to enhance pavement assessment and analysis. These developments are supported by a user-friendly interface designed for practical use.

17. Key Words Asphalt Concrete, Dynamic Backcalculation, Finite Element (FE) Model, Optimization, Master Curve, Heavy Weight Deflectometer (HWD), Falling Weight Deflectometer (FWD), Pavement Assessment, Impulse Load, Surface Deflections, Forward Model, Moduli, BAKFAA, Subgrade Damping, *PULSE*_FE, CC-9 Flexible Test Item, FWD Parameters, Optimization Framework, Constrained Optimization, GUI Program, Automated Optimization Framework, Pavement Analysis		18. Distribution Statement This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161. This document is also available from the Federal Aviation Administration William J. Hughes Technical Center at actlibrary.tc.faa.gov.	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 179	22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

ACKNOWLEDGEMENTS

The Federal Aviation Administration (FAA) deserves acknowledgement for funding this project and offering continuous support. Ms. Nicole Elias from the University of Nevada, Reno (UNR) merits recognition for her valuable assistance in the literature review. Dr. Gabriel Bazi's contribution to the project, including work on finite element (FE) analysis, development of the FE standalone model, evaluation of the Newton-Raphson method for predicting the asphalt concrete (AC) master curve, and backcalculation of layer variables, is also appreciated. Dr. Bazi provided guidance to the research team throughout the project's duration.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

2D	Two-dimensional
3D	Three-dimensional
AASHTO	American Association of State Highway and Transportation Officials
AC	Asphalt concrete
AI	Artificial intelligence
ANN	Artificial neural network
API	Application programming interface
AWS	Amazon Web Services
B	Boeing
BFGS	Broyden–Fletcher–Goldfarb–Shanno
CAD	Computer-aided design
CC	Construction Cycle
COM	Component object model
COO	Coordinate list
COV	Coefficient of variation
CSC	Compressed sparse column
CSR	Compressed sparse row
CSV	Comma-separated values
DFP	Davidon–Fletcher–Powell
DynaBAKFAA	BAKFAA Dynamic backcalculation
EI	Expected improvement
EKF	Extended Kalman filter
FAA	Federal Aviation Administration
FE	Finite element
FWD	Falling weight deflectometer
GUI	Graphical user interface
HHT	Hilber-Hughes-Taylor method
HWD	Heavy weight deflectometer
ISU	Iowa State University
JSON	JavaScript Object Notation
L-BFGS-B	Limited-memory BFGS with bound constraints
LET	Linear elastic theory
LFS	Low-strength subgrade flexible pavement with a stabilized base
LTPP	Long-term pavement performance
LVE	Linear viscoelastic
MSU	Michigan State University
NAPTF	National Airport Pavement Test Facility
NN	Neural network
nnz	Non-zero elements
nr	Number of rows
OLE	Object Linking and Embedding
POI	Probability of improvement
RMSRE	Root-mean-square relative error
SDK	Software Development Kit
SGD	Stochastic Gradient Descent

SR1	Symmetric rank 1
SVD	Singular-value decomposition
TTSP	time-temperature superposition principle
UCB	Upper confidence bound
UNR	University of Nevada, Reno

EXECUTIVE SUMMARY

The Heavy Weight Deflectometer (HWD)/Falling Weight Deflectometer (FWD) is a nondestructive tool typically used for assessing pavement conditions. An impulse load, simulating the effects of a moving wheel, is applied to the pavement surface, and the resulting surface deflections are collected. Pavement variables, such as moduli, are then determined through a method known as backcalculation. This process employs a forward model to compute pavement surface deflections and an optimization model to progressively minimize the differences between the measured and computed deflections.

Presently, the Federal Aviation Administration (FAA) relies on linear elastic theory (LET) and static analysis within its BAKFAA software to estimate layer moduli for both flexible and rigid pavements. However, static analysis has limitations, particularly when dealing with thick and stiff airfield pavement structures.

To address these limitations, a finite element (FE) model was developed to incorporate the subgrade's damping behavior, which is a key consideration given the short duration for the FWD impulse load. This dynamic model facilitated the backcalculation of layer moduli for various pavement structures constructed over the same subgrade at the FAA National Airport Pavement Test Facility (NAPTF). The model is also validated by comparing the calculated responses, including data from pressure cells, strain gauges, and multi-depth deflectometers, with the respective measured responses during FAA Construction Cycle (CC)-1 tests. In summary, dynamic backcalculation emerges as the preferred approach over static backcalculation due to its ability to account for inertial effects, viscoelastic behavior of the asphalt concrete (AC) layer, and material damping.

In this project, an FE tool known as *PULSE*_FE is developed that can assess the responses of multilayer pavement structures under static or dynamic impulse loading with linear elastic and viscoelastic isotropic materials. *PULSE*_FE was validated against the ABAQUS FE, yielding identical results within 1 to 3 percent of the time ABAQUS typically needs. This considerable improvement in computation time marks a significant milestone, rendering dynamic backcalculation a feasible approach.

The dynamic backcalculation process was successfully applied to a CC-9 flexible test item, yielding reliable layer parameters. A comprehensive parametric study was conducted, involving 15,552 pavement structures through FE modeling, resulting in a preliminary list of key FWD parameters for the backcalculation process.

An optimization framework was developed to streamline the backcalculation procedure. The framework integrates pavement structure modeling, preprocessing, FE modeling, and analysis, enabling the direct retrieval of calculated parameters without manual intervention. Multiple optimizers, including variants of Newton-Raphson, Quasi-Newton, Powell, Nelder–Mead, Bayesian, and Kalman, were implemented and evaluated within this optimization framework. Constrained optimization techniques were employed to enhance the generation of practical solutions. Experimental evaluations, conducted using both synthetic and field measured data, provided strong evidence of the effectiveness of these optimization methods and the reliable recovery of pavement variables through this approach.

Additionally, a user-friendly graphical user interface (GUI) program for the BAKFAA Dynamic Backcalculation (DynaBAKFAA) software was developed. This software allows users to perform a range of tasks, including generating a mesh for the pavement structure domain, creating FWD input files, inputting and editing material properties for each pavement layer, conducting forward analyses to determine pavement responses at various locations within the structure, and performing dynamic backcalculation using a variety of optimizers to ascertain pavement variables.

In summary, this work has led to the development of a competent FE module, an advanced dynamic backcalculation process, and an automated optimization framework. These advancements are expected to enhance pavement assessment and analysis, with the user-friendly GUI program further facilitating the adoption of these techniques in practical applications.

1. INTRODUCTION

The heavy weight deflectometer (HWD)/falling weight deflectometer (FWD) is a nondestructive pavement evaluation device commonly used to assess pavement condition. It operates by applying a short-duration impulse load (simulating a moving wheel load) to the pavement surface and measuring the corresponding deflections. The pavement variables (e.g., moduli) are then determined through a process called backcalculation, using a forward model to calculate the deflections and an optimization model to iteratively improve the variables. A reliable backcalculation of the layer variables using FWD data is critical for the structural evaluation of pavements.

The Federal Aviation Administration (FAA) currently uses the linear elastic theory (LET), which is based on static analysis in its BAKFAA (2023) software for the backcalculation of layer moduli for flexible and rigid pavements. Static analysis presents numerous constraints during the backcalculation process, particularly for airports where rigid or thick and stiff flexible pavement structures are prevalent. In a previous FAA study, researchers developed a simple and robust finite element (FE) model based on dynamic analysis to address the limitations of existing models. They also properly incorporated the subgrade's damping behavior, which had to be considered due to the short FWD impulse load (Bazi, Mansour, Sebaaly, & Hajj, 2018; Bazi, Gagnon, Sebaaly, & Ullidtz, 2020). The model was used at the FAA National Airport Pavement Test Facility (NAPTF) for the backcalculation of layer moduli for flexible and rigid pavement structures built over the same subgrade. The new model's backcalculation generated reasonable layer moduli that aligned with the type of material in each layer, and, most importantly, it yielded nearly identical layer moduli for the same subgrade material under various pavement structures and types. The generated layer variables (including moduli) were validated by comparing the measured responses from the FAA Construction Cycle (CC)-1 test items with responses calculated using three-dimensional (3D) FE models under simulated Boeing (B)747 and B777 moving wheel loads (Bazi, Mansour, Sebaaly, Ji, & Garg, 2019). The measured responses, which included data from pressure cells, strain gauges, and multi-depth deflectometers, closely matched the calculated responses.

In summary, dynamic backcalculation is recommended over static backcalculation because it enables the consideration of inertial effects, the viscoelastic behavior of the asphalt concrete (AC) layer, and the material damping.

As part of the current project, the research team is developing a standalone FE module (*PULSE*_FE) based on dynamic analysis and evaluating different optimization techniques for upgrading the BAKFAA (2023) software. The *PULSE*_FE has static and dynamic modeling capabilities along with linear elastic and viscoelastic isotropic materials. The FE module is introduced and compared to the commercial ABAQUS FE software (ABAQUS, 2019). The *PULSE*_FE effectiveness and efficiency are demonstrated by producing identical results to ABAQUS in only 1 to 3 percent the runtime that ABAQUS takes. The improvement in the calculation time is a significant achievement, making dynamic backcalculation feasible.

Furthermore, the update to the dynamic backcalculation involves (1) attempting to predict the AC layer master curve using synthetic FWD data, (2) assessing a CC-9 flexible pavement test item, and (3) conducting a parametric study to determine the most significant FWD parameters for use in the dynamic backcalculation. The analyses indicate that predicting the master curve is challenging, whereas conducting the dynamic backcalculation is feasible and capable of generating reliable results. The analysis of the CC-9 test item reveals that unbound materials, including the aggregate base and subgrade layers, exhibit mild stress-softening behavior that require evaluation through backcalculation at various FWD load levels. It is noteworthy that the stress sensitivity obtained from backcalculation is not as pronounced as the material non-linearity observed in laboratory testing. Previous research has demonstrated that the confinement effect resulting from the stiffness of the layers above an unbound layer significantly influences that layer and should undergo further evaluation, potentially being considered in pavement analysis procedures (Bazi, Saboundjian, Bou Assi, & Diab, 2020).

The remaining sections of the report are structured as follows: Section 2 explains the theoretical background of FE modeling, with a focus on the details of the axisymmetric case. Section 3 provides an overview of the FE module, covering topics such as the mesh generator, matrix storage, and a comparison between *PULSE*_FE and ABAQUS. Section 4 showcases the improvements in dynamic backcalculation using both simulated and real FWD data. Section 5 describes the development of an Optimization Technique. Section 6 offers a summary of the BAKFAA Dynamic Backcalculation (DynaBAKFAA) deliverable. Section 7 details the established database for the *PULSE*_FE program. Section 8 presents the summary and conclusions, and Section 9 contains the list of references.

2. FINITE ELEMENT METHOD

The FE method operates on the fundamental concept of dividing a continuum into a finite number of smaller regions known as finite elements. These elements must neither overlap nor have gaps between them. Each element is characterized by a set of key points referred to as nodes, which govern the element's behavior. Typically, these elements exhibit simpler geometries, load conditions, boundary conditions, and so on compared to the original continuum. This simplification ensures that stresses and displacements within each element vary in a monotonous manner. Consequently, deformation within each element can be approximated using displacement functions. By establishing dependencies between displacements or stresses at any point within an element and those at its nodes, a finite set of differential equations of motion can be formulated for these nodes. This approach allows users to transform a problem with an infinite number of degrees of freedom into one with a finite number, streamlining the solution process (Qu, 2004).

2.1 MECHANICAL RESPONSE OF SOLIDS AND STRUCTURES TO EXTERNAL FORCES

Loads, or external forces, subject solids and structures to stress, resulting in nonuniform stresses and measurable strains or even observed deformation/displacement. Solid or structural mechanics establish relationships among stresses, strains, displacements, and forces, under proper boundary conditions for solids and structures. These relationships are crucial for modeling, simulating, and designing engineered structural systems (Liu & Quek, 2013).
Forces can be static or dynamic. Static forces concern solids and structures under static loads, while dynamic forces induce vibrations with time-dependent stress, strain, and displacement.

Dynamics principles and theories apply in such cases. Statics can be derived as a special case of dynamics by omitting dynamic terms from the general dynamic equations (Liu & Quek, 2013).

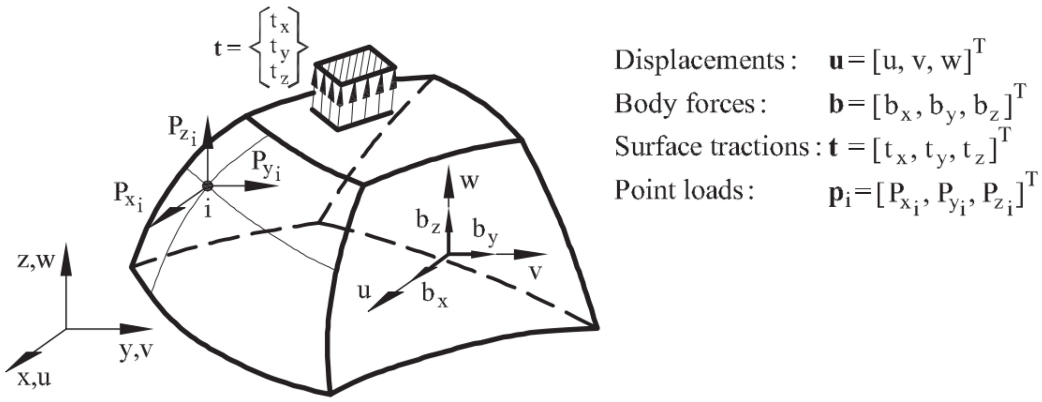
2.2 THREE-DIMENSIONAL SOLIDS

2.2.1 Displacement Vector and Motion in 3D Solids

A point's motion in a 3D solid (Figure 1) is determined by a displacement vector $\boldsymbol{U}$ comprising three components:

$$\boldsymbol{U} = [u, v, w]^T \tag{1}$$

The displacements of the point in the Cartesian axe's directions x, y, and z are represented by *u*, *v*, and *w*, respectively.



$\mathbf{t} = \begin{Bmatrix} t_x \\ t_y \\ t_z \end{Bmatrix}$
P_{z_i}
P_{y_i}
P_{x_i}
i
w
b_z
b_y
v
u
b_x
z,w
y,v
x,u
Displacements: $\mathbf{u} = [u, v, w]^T$
Body forces: $\mathbf{b} = [b_x, b_y, b_z]^T$
Surface tractions: $\mathbf{t} = [t_x, t_y, t_z]^T$
Point loads: $\mathbf{p}_i = [P_{x_i}, P_{y_i}, P_{z_i}]^T$

Figure 1. Displacements and Loads for a 3D Solid (Oñate, 2009)

2.2.2 Strain Components and Their Mathematical Formulation in 3D Solids

Strain represents the change in displacements per unit length, and, as such, the strain components in a 3D solid are determined by calculating derivatives of the displacements. The strain field is defined by six strain components (ε_x, ε_y, ε_z, γ_{xy}, γ_{xz}, and γ_{yz}), forming a strain vector ε.

$$\varepsilon = \left[\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \gamma_{yz}, \gamma_{xz}, \gamma_{xy}\right]^T \tag{2}$$

$$\varepsilon_{xx} = \frac{\partial u}{\partial x} \qquad \varepsilon_{yy} = \frac{\partial v}{\partial y} \qquad \varepsilon_{zz} = \frac{\partial w}{\partial z}$$

$$\gamma_{xy} = 2\varepsilon_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \qquad \gamma_{xz} = 2\varepsilon_{xz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \qquad \gamma_{yz} = 2\varepsilon_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}$$

where ε_x, ε_y and ε_z are the normal strains, and γ_{xy}, γ_{xz}, and γ_{yz} denote the tangential or shear strains. It is important to note that the engineering notation of γ_{ij}, referred to as engineering shear strain, is used instead of the tensor notation of ε_{ij} ($=\frac{\gamma_{ij}}{2}$) for the shear strain components in the vector form of strains. This distinction arises because tensors are mathematical entities subject to specific rules governing their transformation between coordinate systems.

2.2.3 Stress Components and Stress Tensors in 3D Solids

In a 3D solid, stress components are represented on the surface of an infinitesimal cubic volume. Each surface features one normal stress component and two shear stress components. The stress designation employs subscripts, with the first subscript indicating the acting surface, and the second subscript indicating the stress direction.

Six independent stress components exist at a point within 3D solids, as depicted in Figure 2. These stress components are termed stress tensors because they adhere to the rules of coordinate transformation for tensors and can be expressed in vector form, as shown in Equation 3:

$$\boldsymbol{\sigma} = \left[\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \tau_{yz}, \tau_{xz}, \tau_{xy}\right]^T \tag{3}$$

In this equation, σ_x (σ_{xx}), σ_y $\left(\sigma_{yy}\right)$, and σ_z (σ_{zz}) represent the normal stresses, and τ_{xy}, τ_{xz}, and τ_{yz} denote the tangential or shear stresses. The shear equivalence relations ($\tau_{ij} = \tau_{ji}$) are validated through the equilibrium state, considering the moments of forces about the central axes of the cube.

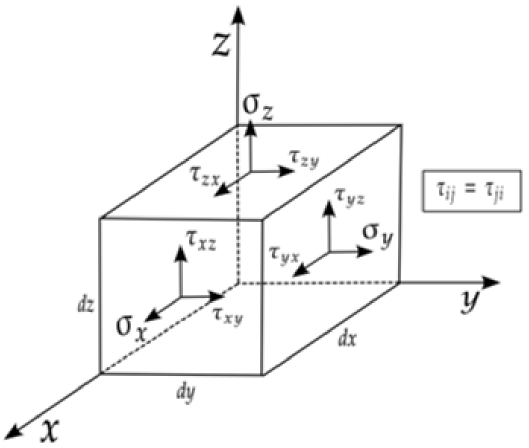


Figure 2. Stresses in a 3D Solid Element (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

2.2.4 Stress-Strain Relationship

Hooke's law, a constitutive equation, defines the relationship between stress and strain in a material. The following matrix form provides the generalized Hooke's law for 3D anisotropic materials, as shown in Equation 4:

$$\boldsymbol{\sigma} = \boldsymbol{D}\varepsilon \tag{4}$$

In this equation, ***D*** represents the 6×6 elasticity matrix of material constants. The constitutive equation is written explicitly as:

$$\begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \tau_{yz} \\ \tau_{xz} \\ \tau_{xy} \end{Bmatrix} = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} & D_{15} & D_{16} \\ & D_{22} & D_{23} & D_{24} & D_{25} & D_{26} \\ & & D_{33} & D_{34} & D_{35} & D_{36} \\ & & & D_{44} & D_{45} & D_{46} \\ & & & & D_{55} & D_{56} \\ sy. & & & & & D_{66} \end{bmatrix} \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{yz} \\ \gamma_{xz} \\ \gamma_{xy} \end{Bmatrix} \tag{5}$$

For a fully anisotropic material, there are 21 independent material constants D_{ij} (since $D_{ij} = D_{ji}$). For linear elastic isotropic materials, **D** is reduced to:

$$\boldsymbol{D} = E\overline{\boldsymbol{D}} = E \begin{bmatrix} D_{11} & D_{12} & D_{12} & 0 & 0 & 0 \\ & D_{11} & D_{12} & 0 & 0 & 0 \\ & & D_{11} & 0 & 0 & 0 \\ & & & D_{44} & 0 & 0 \\ & & & & D_{44} & 0 \\ sy. & & & & & D_{44} \end{bmatrix} \tag{6}$$

$$D_{11} = \frac{(1-\mu)}{(1-2\mu)(1+\mu)} \qquad D_{12} = \frac{\mu}{(1-2\mu)(1+\mu)} \qquad D_{44} = \frac{1}{2(1+\mu)}$$

where *E* is the Young's modulus, μ is the Poisson's ratio, *G* is the shear modulus, and $\overline{\boldsymbol{D}}$ is the dimensionless elasticity matrix. There are two independent constants among *E*, μ, and *G*. Given any two of the three constants, the other one is calculated using the following equation:

$$G = \frac{E}{2(1+\mu)} \tag{7}$$

The inverse of the elasticity matrix **D** is referred to as the flexibility or compliance matrix, which is defined differently for isotropic, orthotropic, and anisotropic material. For isotropic materials, $\mathbf{D}^{-1}$ matrix is given in the following form:

$$\boldsymbol{D}^{-1} = \frac{1}{E} \begin{bmatrix} 1 & -\mu & -\mu & 0 & 0 & 0 \\ & 1 & -\mu & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \\ & & & 2(1+\mu) & 0 & 0 \\ & & & & 2(1+\mu) & 0 \\ sy. & & & & & 2(1+\mu) \end{bmatrix} \tag{8}$$

2.3 AXISYMMETRIC MODELING FOR SIMPLIFIED ANALYSIS OF 3D SYMMETRIC PROBLEMS

When dealing with a 3D problem characterized by symmetric geometry, loadings, boundary conditions, and materials with respect to an axis, one resolves it as an axisymmetric problem (Figure 3) by employing two-dimensional (2D) finite elements. Typically, cylindrical coordinates (r, θ, z) are used, where r denotes the radial direction from the axis of rotation, z signifies the direction along the axis of rotation, and θ represents the circumferential direction (Hutton, 2004). In this scenario, displacement remains unaffected by the tangential coordinate θ, resulting in a stress analysis that is mathematically 2D, dependent on radial and axial coordinates, even though the physical problem is 3D.

For the analysis of pavement structures under FWD loading, it is best to formulate them using axisymmetric models. This approach simplifies and accelerates the analysis compared to comprehensive 3D analyses. It must be stated that when identical input parameters are applied, the 2D axisymmetric and 3D models yield identical results.

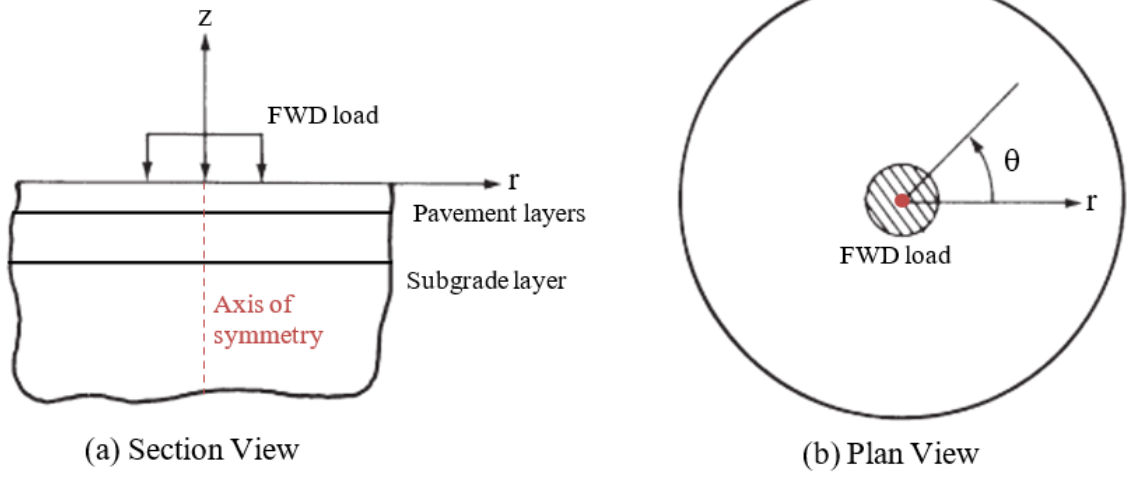


Figure 3. Axisymmetric Pavement Problem under FWD Loading (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

2.3.1 Radial and Axial Displacement Field

The radial (u) and axial (w) displacements define the movement of a point in an axisymmetric solid (Figure 4), with the following displacement vector $\boldsymbol{U}$:

$$\boldsymbol{U} = [u, w]^T \tag{9}$$

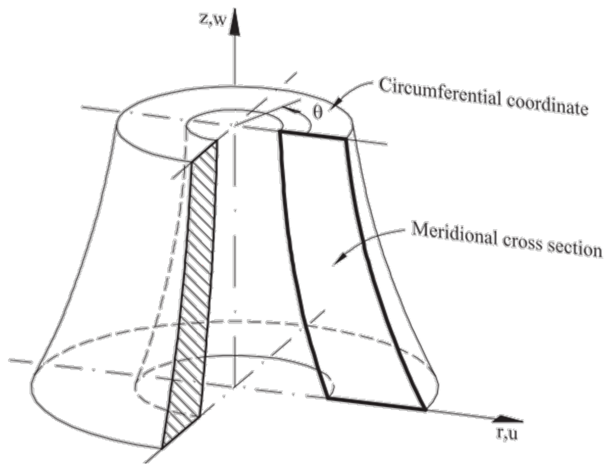


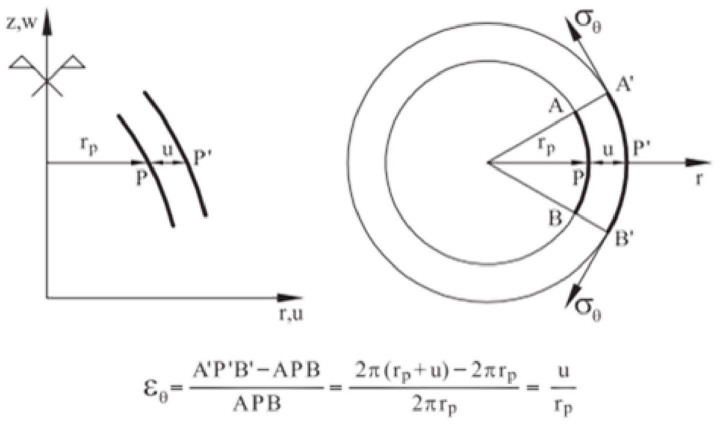
Figure 4. Axisymmetric Solid (Oñate, 2009)

2.3.2 Strain Analysis in Axisymmetric Solids with Symmetrical Displacements

Because of the symmetry in geometry, material properties, boundary conditions, and loads about the axis of symmetry, the displacements u and w become independent of the circumferential coordinate θ. Consequently, the tangential strains $\gamma_{r\theta}$ and $\gamma_{z\theta}$ become zero.

Points located on a circumference of radius r experience movement, due to the axial deformation, to a circumference of radius $r + u$. This results in a circumferential strain, defined as the relative elongation between these two circumferences (Figure 5).

$$\varepsilon_\theta = \frac{2\pi(r+u)-2\pi r}{2\pi r} = \frac{u}{r} \tag{10}$$



$$\varepsilon_\theta = \frac{A'P'B' - APB}{APB} = \frac{2\pi(r_p+u)-2\pi r_p}{2\pi r_p} = \frac{u}{r_p}$$

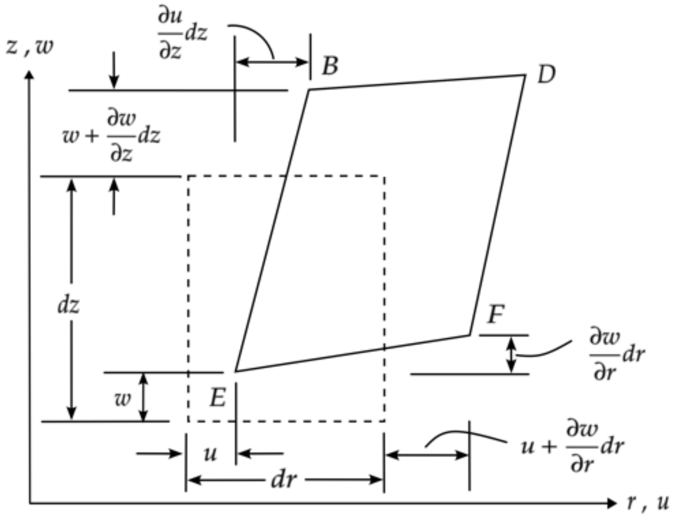
Figure 5. Derivation of the Circumferential (Hoop) Strain $\boldsymbol{\varepsilon_\theta}$ (Oñate, 2009)

The radial, axial, and shear strains are obtained by derivatives of the displacements, as illustrated in Figure 6. The strain vector ε reduces to:

$$\varepsilon = [\varepsilon_r, \varepsilon_z, \varepsilon_\theta, \gamma_{rz}]^T \tag{11}$$

$$\varepsilon_r = \frac{\partial u}{\partial r} \qquad \varepsilon_z = \frac{\partial w}{\partial z} \qquad \varepsilon_\theta = \frac{u}{r} \qquad \gamma_{rz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial r}$$

where ε_r, ε_z, and ε_θ represent the normal strains (radial, axial, and circumferential or hoop, respectively), and γ_{rz} signifies the tangential or shear strain.



z, w
$\frac{\partial u}{\partial z}dz$
B
D
$w + \frac{\partial w}{\partial z}dz$
dz
F
$\frac{\partial w}{\partial r}dr$
w
E
u
dr
$u + \frac{\partial w}{\partial r}dr$
r, u

Figure 6. Derivation of the Radial, Axial, and Shear Strains
(Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

2.3.3 Vector Representation of Stresses and Their Components

The stresses are written in a vector form:

$$\boldsymbol{\sigma} = [\sigma_r, \sigma_z, \sigma_\theta, \tau_{rz}]^T \tag{12}$$

In this representation, σ_r, σ_z, and σ_θ denote the normal radial, axial, and circumferential stresses, respectively. As shown in Figure 7, τ_{rz} represents the tangential or shear stress.

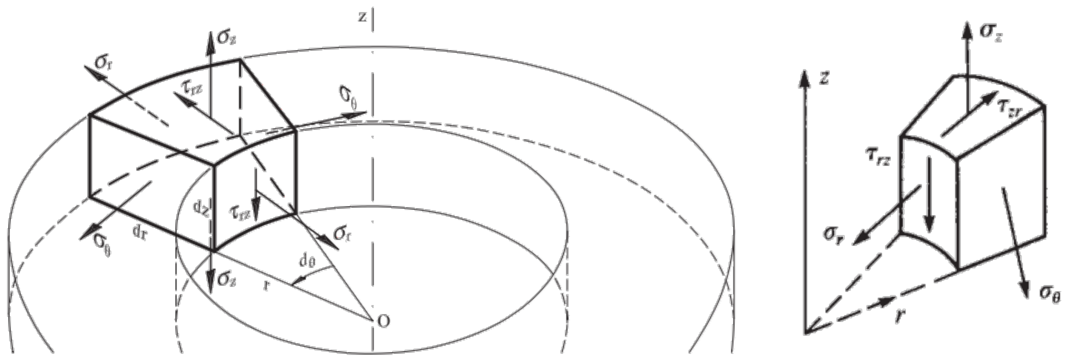


Figure 7. Stresses Acting on Differential Volume of Axisymmetric Solid (Oñate, 2009)

2.3.4 Stress-Strain Relationship

The elasticity matrix, $\boldsymbol{D}$, for a 2D axisymmetric problem is derived from that of a 3D solid by imposing the conditions of $\gamma_{r\theta} = \gamma_{z\theta} = 0$ and by assuming that the shear strain, γ_{rz}, is not coupled with the hoop stress, σ_θ. For a 2D axisymmetric isotropic material, the matrix of elastic constants is given as:

$$\boldsymbol{D} = E\overline{\boldsymbol{D}} = E \begin{bmatrix} D_{11} & D_{12} & D_{12} & 0 \\ \square & D_{11} & D_{12} & 0 \\ \square & \square & D_{11} & 0 \\ sy. & \square & \square & D_{44} \end{bmatrix} \tag{13}$$

$$D_{11} = \frac{(1-\mu)}{(1-2\mu)(1+\mu)} \qquad D_{12} = \frac{\mu}{(1-2\mu)(1+\mu)} \qquad D_{44} = \frac{1}{2(1+\mu)}$$

2.4 LINEAR VISCOELASTIC MATERIAL BEHAVIOR AND RHEOLOGICAL MODELS

Viscoelastic materials are modeled to determine their stress and strain interactions, as well as their temporal dependencies. Their response depends not only on the deformation but also on the rate of deformation when loaded. The material also experiences relaxation, in which the stress gradually decreases when deformation is constant, or creep, in which the deformation gradually increases when the load is kept constant.

Viscoelastic behavior comprises elastic and viscous components modeled as linear combinations of springs and dashpots, respectively. The elastic spring is called Hooke element, while the dashpot is referred to as the Newton element (see Figure 8).

The elastic material constant, denoted as E, provides the linear relationship for the Hooke element between elastic strain, ε^e, and elastic stress, σ^e.

$$\sigma^e = E\varepsilon^e \tag{14}$$

The viscous stress σ^v of the Newton element relates to the strain rate $\dot{\varepsilon}^v$ using the coefficient of viscosity η.

$$\sigma^v = \eta \frac{d\varepsilon^v}{dt} = \eta \dot{\varepsilon}^v \tag{15}$$

The viscosity η can be expressed in terms of the elastic constant E by introducing the relaxation time τ.

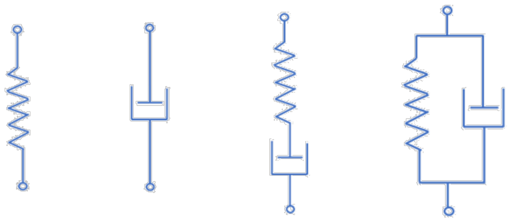


Figure 8. Hooke, Newton, Maxwell, and Kelvin-Voigt Elements (from left) (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

Combining the Hooke and Newton elements in series yields the Maxwell element, while their parallel combination results in the Kelvin-Voigt model (Figure 8). The Maxwell model can predict stress relaxation in materials, whereas the Kelvin-Voigt model can predict creep. More complex models can be constructed using different combinations of springs and dashpots to better fit experimental data from creep, relaxation, or frequency-dependent tests. A complex viscoelastic rheological model typically takes the form of the generalized Maxwell model or the generalized Kelvin chain.

2.4.1 Viscoelastic Modeling and Generalized Maxwell Elements

In a Maxwell element, the total strain ε is the sum of an elastic, ε^e, and a viscous ε^v component, while the stress remains consistent in both rheological elements.

$$\varepsilon = \varepsilon^e + \varepsilon^v \tag{16}$$
$$\sigma = E\varepsilon^e = \eta\dot{\varepsilon}^v \tag{17}$$

By differentiating Equation 16 with respect to time and applying the constitutive relations for both the spring and dashpot, the differential equation for the Maxwell model is derived:

$$\dot{\varepsilon} = \dot{\varepsilon}^e + \dot{\varepsilon}^v = \frac{\dot{\sigma}}{E} + \frac{\sigma}{\eta} \tag{18}$$

Solving this equation in a relaxation experiment (Figure 9) yields the following:

$$\frac{\sigma(t)}{\varepsilon_0} = Ee^{-\frac{t}{\tau}} = R(t) \tag{19}$$

where $\tau = \frac{\eta}{E}$ represents the relaxation time, i.e., the time needed to reduce the stress to e^{-1} of its initial value after imposing the strain, and $R(t)$ is the relaxation function.

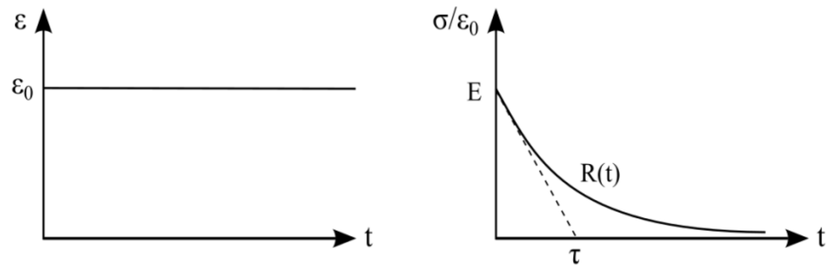


Figure 9. Relaxation Test with Maxwell Element: Strain History (Left) and Resulting Stress Response (Right) (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

The general solution of the differential equation of the Maxwell element with arbitrary strain histories is obtained through the convolution integral:

$$\sigma(t) = E \int_0^t e^{-\frac{t-s}{\tau}} \dot{\varepsilon} ds = \int_0^t R(t-s) \dot{\varepsilon}\, ds \tag{20}$$

This integral is known as the Hereditary Integral and is closely related to the Boltzmann Superposition Principle for linear isotropic viscoelastic materials.

While the basic Maxwell model qualitatively captures the material behavior, it might fall short in providing a quantitative representation. To address this, Generalized Maxwell models are systematically developed to improve accuracy (see Figure 10). The Generalized Maxwell model, also known as the Wiechert model, consists of a finite number of separate Maxwell elements arranged in parallel with an elastic Hooke element. This model considers relaxation that occurs at multiple times, not just a single time.

The relaxation function, $R(t)$, can be expressed in terms of a series of negative exponentials, forming the Prony series that mathematically characterizes the Generalized Maxwell model:

$$R(t) = E_\infty + \sum_{j=1}^{N} E_j e^{-\frac{t}{\tau_j}} \tag{21}$$

Here, E_∞ represents the long-term equilibrium modulus; E_j and τ_j denote the elastic stiffness and viscous relaxation time associated with each element in the generalized Maxwell model, respectively; and N represents the number of spring-dashpot Maxwell elements.

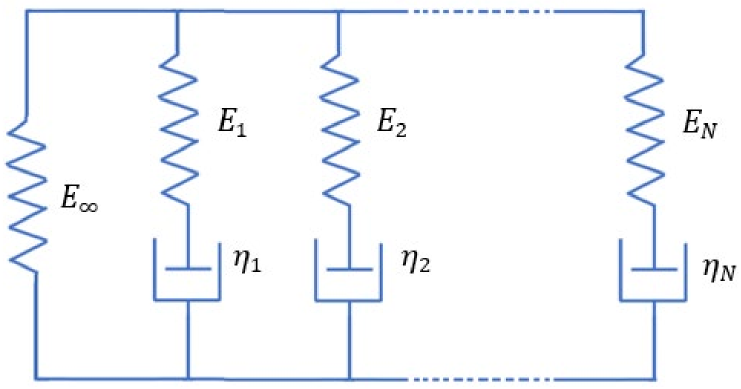


Figure 10. Generalized Maxwell Model (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

At time $t = 0$, the instantaneous modulus, E_0, can be determined as:

$$E_0 = E_\infty + \sum_{j=1}^{N} E_j \tag{22}$$

where $E_\infty = E_0\left(1 - \sum_{j=1}^{N} \alpha_j\right)$, and $\alpha_j = \frac{E_j}{E_0}$ represents the relative modulus of term j.

The Prony series representation of viscoelastic material behavior is incorporated into nearly every state-of-the-art FE software, including ABAQUS, and plays a vital role in the characterization of viscoelastic materials.

2.4.2 Numerical Model Development and Viscoelastic Stress Decomposition

The development of a numerical model commences with the general integral representation of linear viscoelasticity, using Equation 20 to decompose the stress into elastic and viscoelastic components (Kaliske & Rothert, 1997).

$$\sigma^{t+\Delta t} = \sigma_\infty^{t+\Delta t} + \sum_{j=1}^{N} M_j^{t+\Delta t} \tag{23}$$

The equation above is reformulated in the following manner:

$$\sigma^{t+\Delta t} = V\bar{D}\varepsilon^{t+\Delta t} - q^t \tag{24}$$

Here, V represents the viscoelastic tangent modulus multiplier, and q^t is a stress vector dependent on variables known at the start of the time step. $M_j^{t+\Delta t}$ signifies the internal stress variables. It is evident that q needs to be incorporated into the right-hand side of the equation of motion and can be regarded as a pseudo-load vector.

$$V = E_0\left(1 - \sum_{j=1}^{N} O_j\right) \tag{25}$$

$$O_j = \alpha_j \frac{\tau_j}{\Delta t}\left(\frac{\Delta t}{\tau_j} + e^{-\frac{\Delta t}{\tau_j}} - 1\right) \tag{26}$$

$$q^t = \sum_{j=1}^{N}\left[\alpha_j \frac{1-e^{-\frac{\Delta t}{\tau_j}}}{\frac{\Delta t}{\tau_j}} \overline{D} E_0 \varepsilon^t - e^{-\frac{\Delta t}{\tau_j}} M_j^t\right] \tag{27}$$

$$M_j^t = e^{-\frac{\Delta t}{\tau_j}} M_j^{t-\Delta t} + \alpha_j \frac{1-e^{-\frac{\Delta t}{\tau_j}}}{\frac{\Delta t}{\tau_j}} \overline{D} E_0 (\varepsilon^t - \varepsilon^{t-\Delta t}) \tag{28}$$

2.5 FINITE ELEMENT FORMULATION FOR LINEAR AXISYMMETRIC TRIANGULAR ELEMENTS

This section demonstrates the FE formulation for a straightforward linear three-noded axisymmetric triangular element. It is worth noting that all axisymmetric solid elements exhibit an annular shape, even though the element integrals are calculated within the 2D section.

2.5.1 Finite Element Mesh Generation and Structured vs Unstructured Meshes

First, discretizing the problem domain involves dividing it into a union of elements, which can consist of a single type or a combination of different types. This union of elements forms what is commonly referred to as the FE mesh. The process of creating an FE mesh is often termed mesh generation (Zienkiewicz, Taylor, & Zhu, 2013).

A mesh with a high degree of ordering, such as a Cartesian grid, is classified as structured, while a mesh without this level of order is referred to as unstructured. Figure 11 illustrates an example of 2D structured and unstructured meshes (Woodbury, 2008).

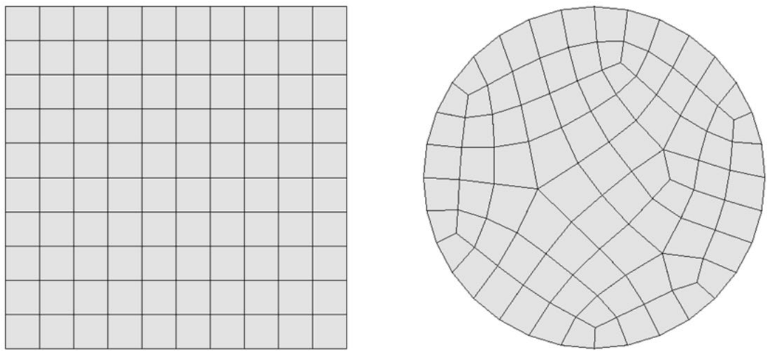


Figure 11. Two-Dimensional Structured (Left) and Unstructured (Right) Meshes (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

Most unstructured mesh generation methods are designed to create triangular elements in 2D and tetrahedral elements in 3D (referred to as simplex forms). These simplex forms offer a straightforward discretization of 2D and 3D domains of varying shapes, especially when meshes with different element sizes in various regions of the domain are needed. Numerous automatic unstructured mesh generation algorithms are available, with the most widely used algorithms being based on one or a combination of the three fundamentally distinctive methods (Zienkiewicz, Taylor, & Zhu, 2013), which are: (1) the Delaunay triangulation method, (2) the advancing-front method, and (3) the tree methods (the finite quadtree method in 2D and the finite octree method in 3D).

2.5.1.1 Selection of 2D Elements for Axisymmetric Analysis

In axisymmetric cases, any 2D element can be used. Figure 12 illustrates the most common linear and higher-order (quadratic) triangular and quadrilateral elements.

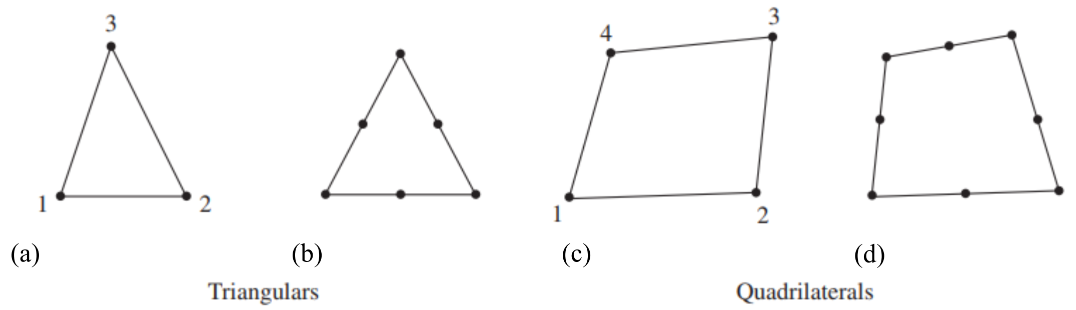


Figure 12. Linear and Quadratic Triangular and Quadrilateral Elements
(Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

2.5.1.2 Modeling Infinite Media in FE Analysis

In generating an FE model, three methods are commonly employed to replicate infinite media. These methods are: (1) the far boundary, (2) the infinite element boundary, and (3) the viscous damping boundary.

The far boundary method, as the name implies, involves shifting the boundary a considerable distance away from the center of the structure until the boundary's influence becomes negligible. The infinite element boundary method emulates unbounded soil boundaries (see Figure 13) by employing specialized shape functions, which cause the nodes on the boundary side to extend infinitely (Cook, 1995).

The viscous damping boundary method uses a series of viscous dampers to absorb the radiating wave energy.

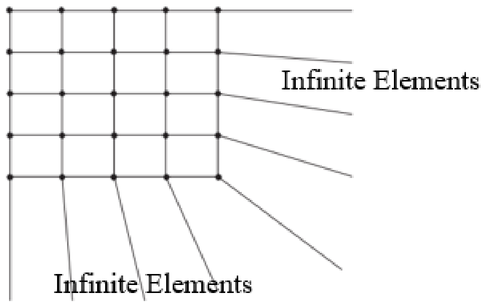


Figure 13. Infinite Elements Attached to Boundary of Standard FE Mesh (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

2.5.2 Discretization of the Displacement Field

Figure 14 displays an axisymmetric linear triangular element. This element comprises three nodes, each with two degrees of freedom per node (u_i and w_i).

The element adopts linear displacement functions, as defined in Logan (2017):

$$u(r,z) = a_1 + a_2 r + a_3 z \tag{29}$$
$$w(r,z) = a_4 + a_5 r + a_6 z \tag{30}$$

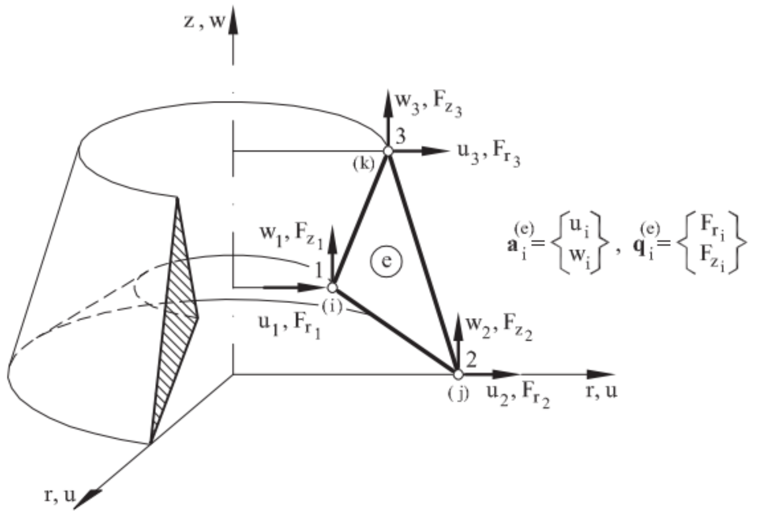

z, w
w_3, F_{z_3}
3
(k)
u_3, F_{r_3}
$\mathbf{a}_i^{(e)} = \begin{Bmatrix} u_i \\ w_i \end{Bmatrix}, \mathbf{q}_i^{(e)} = \begin{Bmatrix} F_{r_i} \\ F_{z_i} \end{Bmatrix}$
w_1, F_{z_1}
1
e
(i)
u_1, F_{r_1}
w_2, F_{z_2}
2
(j)
u_2, F_{r_2}
r, u
r, u

Figure 14. Axisymmetric Three-Noded Triangular Element (Oñate, 2009)

The total number of introduced generalized coordinates a_i in the displacement functions matches the total number of degrees of freedom for the element. The generalized coordinates represent the displacement amplitudes. The nodal displacements for nodes i, j, and k are as follows:

$$\{d\} = \begin{Bmatrix} \{d_i\} \\ \{d_j\} \\ \{d_k\} \end{Bmatrix} = \begin{Bmatrix} u_i \\ w_i \\ u_j \\ w_j \\ u_k \\ w_k \end{Bmatrix} \tag{31}$$

and u and w evaluated at node i as:

$$u(r_i, z_i) = u_i = a_1 + a_2 r_i + a_3 z_i \tag{32}$$
$$w(r_i, z_i) = w_i = a_4 + a_5 r_i + a_6 z_i \tag{33}$$

The displacement function is then expressed in a matrix form:

$$\{\psi\} = \begin{Bmatrix} u \\ w \end{Bmatrix} = \begin{Bmatrix} a_1 + a_2 r + a_3 z \\ a_4 + a_5 r + a_6 z \end{Bmatrix} = \begin{bmatrix} 1 & r & z & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & r & z \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{Bmatrix} \tag{34}$$

a_1 through a_6 can be determined by substituting the coordinates of the nodal points in the above equation, and performing the inversion operations:

$$\begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \begin{bmatrix} 1 & r_i & z_i \\ 1 & r_j & z_j \\ 1 & r_k & z_k \end{bmatrix}^{-1} \begin{Bmatrix} u_i \\ u_j \\ u_k \end{Bmatrix} = \frac{1}{2A} \begin{bmatrix} \alpha_i & \alpha_j & \alpha_k \\ \beta_i & \beta_j & \beta_k \\ \gamma_i & \gamma_j & \gamma_k \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \\ u_k \end{Bmatrix} \tag{35}$$

$$\begin{Bmatrix} a_4 \\ a_5 \\ a_6 \end{Bmatrix} = \begin{bmatrix} 1 & r_i & z_i \\ 1 & r_j & z_j \\ 1 & r_k & z_k \end{bmatrix}^{-1} \begin{Bmatrix} w_i \\ w_j \\ w_k \end{Bmatrix} = \frac{1}{2A} \begin{bmatrix} \alpha_i & \alpha_j & \alpha_k \\ \beta_i & \beta_j & \beta_k \\ \gamma_i & \gamma_j & \gamma_k \end{bmatrix} \begin{Bmatrix} w_i \\ w_j \\ w_k \end{Bmatrix} \tag{36}$$

$$\begin{array}{lll} \alpha_i = r_j z_k - z_j r_k & \alpha_j = r_k z_i - z_k r_i & \alpha_k = r_i z_j - z_i r_j \\ \beta_i = z_j - z_k & \beta_j = z_k - z_i & \beta_k = z_i - z_j \\ \gamma_i = r_k - r_j & \gamma_j = r_i - r_k & \gamma_k = r_j - r_i \end{array}$$

Area of triangular element $A = \frac{1}{2} \begin{vmatrix} 1 & r_i & z_i \\ 1 & r_j & z_j \\ 1 & r_k & z_k \end{vmatrix}$

The shape functions N_i, N_j, and N_k, commonly referred to as interpolation or blending functions, depict how the field variable varies within the FE (Hutton, 2004). Figure 15 demonstrates the shape functions, and they are defined as follows:

$$N_i = \frac{1}{2A}(\alpha_i + \beta_i r + \gamma_i z) \tag{37}$$

$$N_j = \frac{1}{2A}(\alpha_j + \beta_j r + \gamma_j z) \tag{38}$$

$$N_k = \frac{1}{2A}(\alpha_k + \beta_k r + \gamma_k z) \tag{39}$$

$$u(r,z) = a_1 + a_2 r + a_3 z = N_i u_i + N_j u_j + N_k u_k \tag{40}$$

$$w(r,z) = a_4 + a_5 r + a_6 z = N_i w_i + N_j w_j + N_k w_k \tag{41}$$

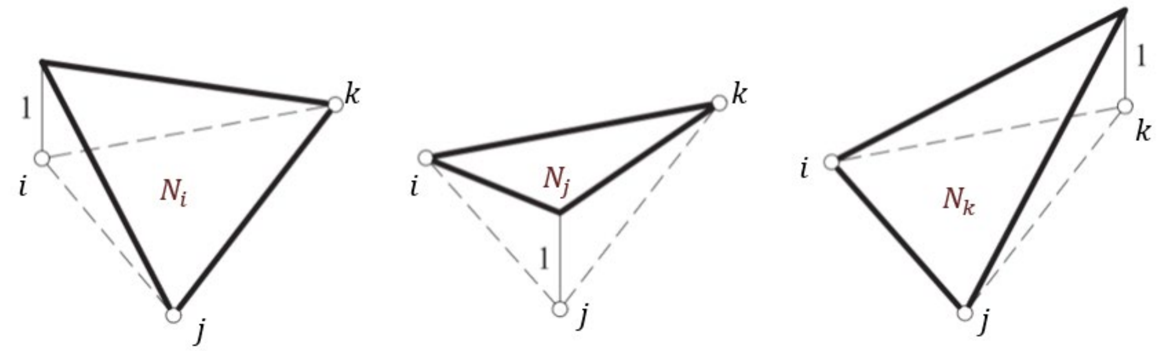


Figure 15. Shape Functions N_i, N_j, and N_k for Three-Noded Triangular Element (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

The general displacement function is expressed in terms of the shape functions:

$$\{\psi\} = \begin{Bmatrix} u \\ w \end{Bmatrix} = \begin{bmatrix} N_i & 0 & N_j & 0 & N_k & 0 \\ 0 & N_i & 0 & N_j & 0 & N_k \end{bmatrix} \begin{Bmatrix} u_i \\ w_i \\ u_j \\ w_j \\ u_k \\ w_k \end{Bmatrix} = [N]\{d\} \tag{42}$$

2.5.3 Discretization of the Strain and Stress Fields

The strain vector is related to the nodal displacements using the gradient matrix, as defined below:

$$\varepsilon = \begin{Bmatrix} \varepsilon_r \\ \varepsilon_z \\ \varepsilon_\theta \\ \gamma_{rz} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial w}{\partial z} \\ \frac{u}{r} \\ \frac{\partial u}{\partial z} + \frac{\partial w}{\partial r} \end{Bmatrix} = \begin{Bmatrix} a_2 \\ a_6 \\ \frac{a_1}{r} + a_2 + \frac{a_3 z}{r} \\ a_3 + a_5 \end{Bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{1}{r} & 1 & \frac{z}{r} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{Bmatrix} \tag{43}$$

$$\varepsilon = \begin{Bmatrix} \varepsilon_r \\ \varepsilon_z \\ \varepsilon_\theta \\ \gamma_{rz} \end{Bmatrix} = \frac{1}{2A} \begin{bmatrix} \beta_i & 0 & \beta_j & 0 & \beta_k & 0 \\ 0 & \gamma_i & 0 & \gamma_j & 0 & \gamma_k \\ \frac{\alpha_i}{r} + \beta_i + \frac{\gamma_i z}{r} & 0 & \frac{\alpha_j}{r} + \beta_j + \frac{\gamma_j z}{r} & 0 & \frac{\alpha_k}{r} + \beta_k + \frac{\gamma_k z}{r} & 0 \\ \gamma_i & \beta_i & \gamma_j & \beta_j & \gamma_k & \beta_k \end{bmatrix} \begin{Bmatrix} u_i \\ w_i \\ u_j \\ w_j \\ u_k \\ w_k \end{Bmatrix} \tag{44}$$

$$\varepsilon = \begin{Bmatrix} \varepsilon_r \\ \varepsilon_z \\ \varepsilon_\theta \\ \gamma_{rz} \end{Bmatrix} = \left[[B_i][B_j][B_k] \right] \begin{Bmatrix} u_i \\ w_i \\ u_j \\ w_j \\ u_k \\ w_k \end{Bmatrix} = [B]\{d\} \tag{45}$$

$$[B] = \left[[B_i][B_j][B_k] \right] \qquad B_i = \frac{1}{2A} \begin{bmatrix} \beta_i & 0 \\ 0 & \gamma_i \\ \frac{\alpha_i}{r} + \beta_i + \frac{\gamma_i z}{r} & 0 \\ \gamma_i & \beta_i \end{bmatrix}$$

$[B]$ is called the gradient matrix, and B_j and B_k can be obtained from B_i by replacing the i subscript by j and k, respectively. It is important to note that $[B]$ is a function of r and z, also indicating that ε_θ is not constant and varies with the polar coordinates.

The stress is then calculated using the following formula, where $[\boldsymbol{D}]$ is the elasticity matrix, $[B]$ is the gradient matrix and $\{d\}$ is the nodal displacements vector.

$$\boldsymbol{\sigma} = \boldsymbol{D}\varepsilon = [\boldsymbol{D}][B]\{d\} \tag{46}$$

2.5.4 Formulation of FE Equations for Dynamic Analysis

Three primary methods exist for deriving the FE equations of a physical system:

(1) The direct method, or direct equilibrium method, commonly applied to structural analysis problems
(2) Variational methods, which encompass subsets like energy methods and the principle of virtual work
(3) Weighted residual methods

The equations of equilibrium governing the dynamic response of a system of FEs are represented as follows (Bathe, 2014):

$$\boldsymbol{M\ddot{U}} + \boldsymbol{C\dot{U}} + \boldsymbol{KU} = \boldsymbol{R} \tag{47}$$

Here, $\boldsymbol{M}$, $\boldsymbol{C}$, and $\boldsymbol{K}$ are the mass, damping, and stiffness matrices, respectively. $\boldsymbol{R}$ is the vector of externally applied loads, and $\boldsymbol{\ddot{U}}$, $\boldsymbol{\dot{U}}$, and $\boldsymbol{U}$ are the acceleration, velocity, and displacement vectors of the FE assembly. The damping matrix is often expressed simply as proportional of the mass and stiffness matrices, known as proportional damping. The equation can be further expressed as:

$$\boldsymbol{F_I(t)} + \boldsymbol{F_D(t)} + \boldsymbol{F_E(t)} = \boldsymbol{R(t)} \tag{48}$$

Where $\boldsymbol{F_I(t)}$ represents the inertia forces, $\boldsymbol{F_I(t)} = \boldsymbol{M\ddot{U}}$; $\boldsymbol{F_D(t)}$ represents the damping forces, $\boldsymbol{F_D(t)} = \boldsymbol{C\dot{U}}$; and $\boldsymbol{F_E(t)}$ represents the elastic forces, $\boldsymbol{F_E(t)} = \boldsymbol{KU}$, all of which are time-dependent. Therefore, in dynamic analysis, static equilibrium at time $\boldsymbol{t}$ is considered, incorporating the effects of acceleration-dependent inertia forces and velocity-dependent damping forces. In contrast, static analysis neglects the effects of inertia and damping.

2.5.5 Stiffness Matrix

The element stiffness matrix is calculated as follows:

$$[k] = \iiint_V [B]^T[D][B]dV = 2\pi \iint_A [B]^T[D][B]rdrdz \tag{49}$$

2.5.6 Mass Matrix

The element mass matrix is calculated as follows:

$$[m] = \iiint_V \rho[N]^T[N]dV = 2\pi \iint_A \rho[N]^T[N]rdrdz \tag{50}$$

The consistent-mass matrix of order 6×6 for a linear triangular element is given by:

$$[m] = \frac{\pi\rho A}{10}\begin{bmatrix} \frac{4}{3}r_1 + 2\bar{r} & 0 & 2\bar{r} - \frac{r_3}{3} & 0 & 2\bar{r} - \frac{r_2}{3} & 0 \\ & \frac{4}{3}r_1 + 2\bar{r} & 0 & 2\bar{r} - \frac{r_3}{3} & 0 & 2\bar{r} - \frac{r_2}{3} \\ & & \frac{4}{3}r_2 + 2\bar{r} & 0 & 2\bar{r} - \frac{r_1}{3} & 0 \\ & & & \frac{4}{3}r_2 + 2\bar{r} & 0 & 2\bar{r} - \frac{r_1}{3} \\ & & & & \frac{4}{3}r_3 + 2\bar{r} & 0 \\ sy. & & & & & \frac{4}{3}r_3 + 2\bar{r} \end{bmatrix} \tag{51}$$

where ρ is the density; and r_1, r_2, and r_3 are the radial coordinates of nodes i, j, and k, respectively.

$$\bar{r} = \frac{r_1 + r_2 + r_3}{3}$$

The lumped matrix, which is a diagonal matrix, is obtained by adding each row of the consistent matrix onto the diagonal. ABAQUS uses the lumped mass matrix for the first-order triangular elements, as it gives more accurate results in numerical experiments that calculate the natural frequencies of simple models.

2.5.7 Damping Matrix

Damping is considered in the form of Rayleigh damping, where the damping matrix is formed as a linear combination of the mass and the stiffness matrices using the Rayleigh damping coefficients α_R and β_R.

$$[c] = \alpha_R[m] + \beta_R[k] \tag{52}$$

2.5.8 Surface Forces

The nodal force vector includes the body and surface forces, where the surface force is used to simulate an FWD load in accordance with the following formula:

$$\{f_s\} = \iint_S [N_s]^T\{T\}dS \tag{53}$$

where $[N_s]$ denotes the shape function matrix evaluated along the surface where the surface traction $\{T\}$ acts.

The nodal forces for an asymmetric first-order triangular element due to a surface traction are illustrated in Figure 16 and calculated using the following formulas for a unit pressure:

$$F_1 = \frac{2\pi}{6}(r_1^2 + r_0r_1 - 2r_0^2) \tag{54}$$

$$F_2 = \frac{2\pi}{6}(2r_1^2 - r_0r_1 - r_0^2) \tag{55}$$

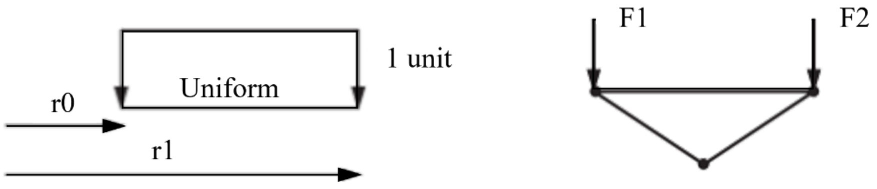


Figure 16. Nodal Forces Due to Surface Traction (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

2.6 TRANSIENT DYNAMIC ANALYSIS

Two effective procedures for transient dynamic analysis are proposed: (1) direct integration and (2) mode superposition, commonly used in the frequency domain.

In direct integration, the equations of equilibrium are integrated using a numerical step-by-step procedure, where the term *direct* means that prior to the numerical integration, no transformation of the equations into a different form is carried out. There are two main types of direct integration method: *implicit* and *explicit*. *Implicit* methods are generally more efficient for a relatively slow phenomenon, and *explicit* methods are more efficient for a very fast phenomenon, such as impact and explosion loadings (Bathe, 2014).
Different direct integration schemes are available, such as the central difference method, the Houbolt method, the Newmark (1959) integration procedure, and the Bathe method (Bathe, 2014). The central difference and Newmark methods are the most used methods.

In the central difference algorithm, the solutions (displacement, velocity, and acceleration) are obtained without solving any matrix form of system equations, which is therefore considered an explicit method. The time marching in explicit methods is therefore extremely fast, and the coding is also very straightforward. It is particularly suited for simulating highly nonlinear, large deformation, contact, and extremely fast events of mechanics. The central difference method, like most explicit methods, is conditionally stable, meaning that the time step Δt must be lower than a critical time step in order not to make the computed solution unstable.

Newmark's method is the most widely used implicit algorithm. The procedure involves matrix inversion that is analogous to solving a matrix of equations. This makes it an implicit method, meaning that one needs to solve a set of linear algebraic equations to obtain a solution at every time step. Because at each time step, the matrix system must be solved, which can be very time-consuming, the implicit algorithm is a very slow time stepping process. Newmark's method, like most implicit methods, is unconditionally stable. Unconditionally stable methods are those in which the size of the time step, Δt, will not affect the stability of the solution, but rather it is governed by accuracy considerations.

Oller (2014) compared an explicit solution with an implicit one, and provided the following aspects:

Explicit time integration methods:

1. The solution algorithm is simple in terms of logic and structure, and it allows carrying out a simple treatment of the different nonlinearities.
2. It requires less memory storage.
3. It does not need expensive tangent operators that are usually found in implicit methods.
4. The explicit methods lead to reliable algorithms.
5. The solution time increment is bounded, requiring small time steps for analysis. This makes the solution at very large time domains time-consuming.

Implicit time integration methods:

1. The implicit methods are very robust and stable.
2. The time increments can be much larger than in explicit methods, preserving the solution stability.
3. They allow more precise solutions with lower error tolerances.
4. A relative drawback is the linearization of the solution through Newton-Raphson, which requires tangent operators that are usually very difficult to obtain.
5. Another drawback is the large storage demand when using direct solution methods for the system of equations.

Because of the robustness of the implicit methods, the Newmark's method is illustrated in the following section.

2.6.1.1 Newmark-β Method

In 1959, N. M. Newmark developed a family of time-stepping methods based on the following equations:

$$\boldsymbol{M}\,{}^{t+\Delta t}\ddot{\boldsymbol{U}} + \boldsymbol{C}\,{}^{t+\Delta t}\dot{\boldsymbol{U}} + \boldsymbol{K}\,{}^{t+\Delta t}\boldsymbol{U} = {}^{t+\Delta t}\boldsymbol{R} \tag{56}$$

$${}^{t+\Delta t}\boldsymbol{U} = {}^{t}\boldsymbol{U} + {}^{t}\dot{\boldsymbol{U}}\Delta t + \left[\left(\frac{1}{2} - \beta\right){}^{t}\ddot{\boldsymbol{U}} + \beta\,{}^{t+\Delta t}\ddot{\boldsymbol{U}}\right]\Delta t^2 \tag{57}$$

$${}^{t+\Delta t}\dot{\boldsymbol{U}} = {}^{t}\dot{\boldsymbol{U}} + \left[(1-\gamma)\,{}^{t}\ddot{\boldsymbol{U}} + \gamma\,{}^{t+\Delta t}\ddot{\boldsymbol{U}}\right]\Delta t \tag{58}$$

The integration parameters β and γ determine the stability, accuracy, dissipative and dispersion characteristics of the system. For linear systems, unconditional stability is achieved when $2\beta \geq \gamma \geq \frac{1}{2}$, and second-order accuracy is obtained for $\gamma = \frac{1}{2}$, i.e., the error decreases proportional to Δt^2. For $\gamma \geq \frac{1}{2}$ and $2\beta < \gamma$, conditional stability is obtained, when the time step is limited to a critical value (De Borst et al., 2012).

Several well-known time integration schemes can be conceived as special cases of the Newmark family. For $\beta = \frac{1}{4}$ and $= \frac{1}{2}$, the average acceleration scheme, or trapezoidal rule is obtained, which is unconditionally stable and second-order accurate in the time step (Figure 17). Other implicit schemes are obtained for $\beta = \frac{1}{6}$ and $= \frac{1}{2}$, the linear acceleration scheme, and for $\beta = \frac{1}{12}$ and $\gamma = \frac{1}{2}$, the Fox–Goodwin scheme. Neither scheme, although each has an implicit format, is unconditionally stable, and the time step is limited to a critical value. Also, some explicit integration schemes can be considered as special cases of the Newmark family, for instance the central difference scheme, which is obtained for $\beta = 0$ and $= \frac{1}{2}$.

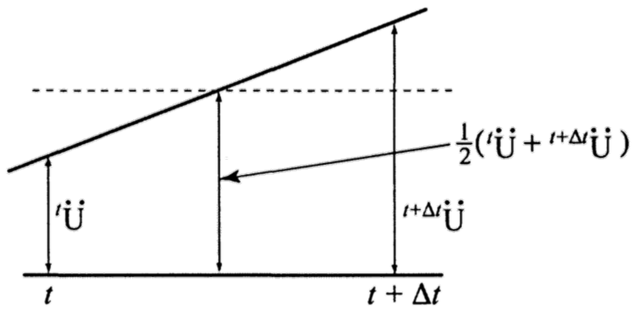

$\frac{1}{2}({}^{t}\ddot{U} + {}^{t+\Delta t}\ddot{U})$
${}^{t}\ddot{U}$
${}^{t+\Delta t}\ddot{U}$
t
$t + \Delta t$

Figure 17. Newmark's Constant-Average Acceleration Scheme (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020)

The Newmark displacement ${}^{t+\Delta t}\boldsymbol{U}$ and velocity ${}^{t+\Delta t}\dot{\boldsymbol{U}}$ equations is rearranged to obtain ${}^{t+\Delta t}\ddot{\boldsymbol{U}}$ and ${}^{t+\Delta t}\dot{\boldsymbol{U}}$ as a functi~~o~~n of ${}^{t+\Delta t}\boldsymbol{U}$.

$$ {}^{t+\Delta t}\dot{\boldsymbol{U}} = -\frac{\gamma}{\beta\Delta t}\,{}^{t}\boldsymbol{U} + \left(1-\frac{\gamma}{\beta}\right){}^{t}\dot{\boldsymbol{U}} + \Delta t\left(1-\frac{\gamma}{2\beta}\right){}^{t}\ddot{\boldsymbol{U}} + \frac{\gamma}{\beta\Delta t}\,{}^{t+\Delta t}\boldsymbol{U} \tag{59} $$

$$ {}^{t+\Delta t}\ddot{\boldsymbol{U}} = \frac{1}{\beta\Delta t^2}\left[-{}^{t}\boldsymbol{U} - \Delta t\,{}^{t}\dot{\boldsymbol{U}} - \Delta t^2\left(\frac{1}{2}-\beta\right){}^{t}\ddot{\boldsymbol{U}} + {}^{t+\Delta t}\boldsymbol{U}\right] \tag{60} $$

The former two equations are rewritten as a function of the integration constants a_0 through a_7 to simplify the calculations:

$$ {}^{t+\Delta t}\dot{\boldsymbol{U}} = {}^{t}\dot{\boldsymbol{U}} + a_6\,{}^{t}\ddot{\boldsymbol{U}} + a_7\,{}^{t+\Delta t}\ddot{\boldsymbol{U}} = a_1\left({}^{t+\Delta t}\boldsymbol{U} - {}^{t}\boldsymbol{U}\right) - a_4\,{}^{t}\dot{\boldsymbol{U}} - a_5\,{}^{t}\ddot{\boldsymbol{U}} \tag{61} $$

$$ {}^{t+\Delta t}\ddot{\boldsymbol{U}} = a_0\left({}^{t+\Delta t}\boldsymbol{U} - {}^{t}\boldsymbol{U}\right) - a_2\,{}^{t}\dot{\boldsymbol{U}} - a_3\,{}^{t}\ddot{\boldsymbol{U}} \tag{62} $$

$a_0 = \frac{1}{\beta\Delta t^2}$ | $a_1 = a_0 a_7 = \frac{\gamma}{\beta\Delta t}$ | $a_2 = \frac{1}{\beta\Delta t}$ | $a_3 = \frac{1}{2\beta} - 1$

$a_4 = a_2 a_7 - 1 = \frac{\gamma}{\beta} - 1$ | $a_5 = a_3 a_7 - a_6 = \frac{\Delta t}{2}\left(\frac{\gamma}{\beta} - 2\right)$ | $a_6 = (1-\gamma)\Delta t$ | $a_7 = \gamma\Delta t$

Replacing ${}^{t+\Delta t}\ddot{\boldsymbol{U}}$ and ${}^{t+\Delta t}\dot{\boldsymbol{U}}$ as a function of ${}^{t+\Delta t}\boldsymbol{U}$ in the general equation of motion (equation 56), the new equation becomes:

$$\boldsymbol{M}\left[a_0\,{}^{t+\Delta t}\boldsymbol{U} - a_0\,{}^{t}\boldsymbol{U} - a_2\,{}^{t}\dot{\boldsymbol{U}} - a_3\,{}^{t}\ddot{\boldsymbol{U}}\right] + \boldsymbol{C}\left[a_1\,{}^{t+\Delta t}\boldsymbol{U} - a_1\,{}^{t}\boldsymbol{U} - a_4\,{}^{t}\dot{\boldsymbol{U}} - a_5\,{}^{t}\ddot{\boldsymbol{U}}\right] + \boldsymbol{K}\,{}^{t+\Delta t}\boldsymbol{U} = {}^{t+\Delta t}\boldsymbol{R} \tag{63}$$

which is rewritten in the following effective form:

$$\widehat{K}\,{}^{t+\Delta t}\boldsymbol{U} = {}^{t+\Delta t}\widehat{\boldsymbol{R}} \tag{64}$$

where $\widehat{K}$ is the effective (tangential) stiffness matrix and ${}^{t+\Delta t}\widehat{\boldsymbol{R}}$ is the effective load.

$$\widehat{K} = a_0\boldsymbol{M} + a_1\boldsymbol{C} + \boldsymbol{K} \tag{65}$$

$${}^{t+\Delta t}\widehat{\boldsymbol{R}} = \boldsymbol{M}\left(a_0\,{}^{t}\boldsymbol{U} + a_2\,{}^{t}\dot{\boldsymbol{U}} + a_3\,{}^{t}\ddot{\boldsymbol{U}}\right) + \boldsymbol{C}\left[a_1\,{}^{t}\boldsymbol{U} + a_4\,{}^{t}\dot{\boldsymbol{U}} + a_5\,{}^{t}\ddot{\boldsymbol{U}}\right] + {}^{t+\Delta t}\boldsymbol{R} \tag{66}$$

The effective load is rewritten as follows to improve the efficiency by calculating the expressions between parentheses outside the time steps loop:

$${}^{t+\Delta t}\widehat{\boldsymbol{R}} = (a_0\boldsymbol{M} + a_1\boldsymbol{C})\,{}^{t}\boldsymbol{U} + (a_2\boldsymbol{M} + a_4\boldsymbol{C})\,{}^{t}\dot{\boldsymbol{U}} + (a_3\boldsymbol{M} + a_5\boldsymbol{C})\,{}^{t}\ddot{\boldsymbol{U}} + {}^{t+\Delta t}\boldsymbol{R} \tag{67}$$

After solving for ${}^{t+\Delta t}\boldsymbol{U}$, ${}^{t+\Delta t}\dot{\boldsymbol{U}}$ and ${}^{t+\Delta t}\ddot{\boldsymbol{U}}$ are calculated using ${}^{t+\Delta t}\boldsymbol{U}$. The procedure then repeats, moving forward in time until arriving at the final desired time.

The system of algebraic equations represented by Equation 64 can be solved at each time step for the unknown displacements, ${}^{t+\Delta t}\boldsymbol{U}$. For a constant time step Δt, the effective stiffness matrix $\widehat{K}$ is constant and needs be computed only once. The effective load on the right-hand side, ${}^{t+\Delta t}\widehat{\boldsymbol{R}}$, must be updated at each time step. By back substitution through the appropriate equations, the velocities ${}^{t+\Delta t}\dot{\boldsymbol{U}}$ and accelerations ${}^{t+\Delta t}\ddot{\boldsymbol{U}}$ are obtained. For the next time step, ${}^{t+\Delta t}\ddot{\boldsymbol{U}}$, ${}^{t+\Delta t}\dot{\boldsymbol{U}}$, and ${}^{t+\Delta t}\boldsymbol{U}$ are set equal to ${}^{t}\ddot{\boldsymbol{U}}$, ${}^{t}\dot{\boldsymbol{U}}$, and ${}^{t}\boldsymbol{U}$, respectively.

2.6.1.2 Hilber-Hughes-Taylor-α Method

Numerical (artificial) dissipation can be desirable in a number of cases, e.g., to filter out the high-frequency modal components that are introduced by the spatial discretization. Numerical dissipation can be introduced in the Newmark scheme for $\gamma > \frac{1}{2}$. Unfortunately, the second-order accuracy is then lost. To avoid this problem, Hilber, Hughes, & Taylor (1977) developed the α-method (HHT-α) with controllable numerical damping, while maintaining Newmark's assumption that the acceleration varies linearly over the time step.

The damping is the most valuable variable in the automatic time-stepping scheme, because the slight, high-frequency numerical noise inevitably introduced when the time step is changed is removed rapidly by a small amount of numerical damping. Each time step change introduces some slight noise or "ringing" into the solution; a little numerical damping quickly removes this high frequency noise without having any significant effect on the meaningful, lower frequency response (ABAQUS, 2019).

The α-method reduces to Newmark's method for $\alpha = 0$, but introduces numerical dissipation for $-\frac{1}{3} < \alpha < 0$, while second-order accuracy is preserved for $\beta = \frac{1}{4}(1-\alpha)^2$ and $\gamma = \frac{1}{2} - \alpha$. ABAQUS sets $\alpha = -0.05$ in its implicit integration scheme for slight numerical damping or transient fidelity.

The equation of motion for time steps can be written as (Hilber, Hughes, & Taylor, 1977):

$$\boldsymbol{M}\,{}^{t+\Delta t}\ddot{\boldsymbol{U}} + (1+\alpha)\boldsymbol{C}\,{}^{t+\Delta t}\dot{\boldsymbol{U}} - \boldsymbol{C}\,{}^{t}\dot{\boldsymbol{U}} + (1+\alpha)\boldsymbol{K}\,{}^{t+\Delta t}\boldsymbol{U} - \alpha\boldsymbol{K}\,{}^{t}\boldsymbol{U} = (1+\alpha)\,{}^{t+\Delta t}\boldsymbol{R} - \alpha\,{}^{t}\boldsymbol{R} \tag{68}$$

Replacing ${}^{t+\Delta t}\dot{\boldsymbol{U}}$ (Equation 61) and ${}^{t+\Delta t}\ddot{\boldsymbol{U}}$ (Equation 62) as a function of ${}^{t+\Delta t}\boldsymbol{U}$ in Equation 68, the equation of motion becomes:

$$\boldsymbol{M}\left[a_0\,{}^{t+\Delta t}\boldsymbol{U} - a_0\,{}^{t}\boldsymbol{U} - a_2\,{}^{t}\dot{\boldsymbol{U}} - a_3\,{}^{t}\ddot{\boldsymbol{U}}\right] + (1+\alpha)\boldsymbol{C}\left[a_1\,{}^{t+\Delta t}\boldsymbol{U} - a_1\,{}^{t}\boldsymbol{U} - a_4\,{}^{t}\dot{\boldsymbol{U}} - a_5\,{}^{t}\ddot{\boldsymbol{U}}\right] - \alpha\boldsymbol{C}\,{}^{t}\dot{\boldsymbol{U}} + (1+\alpha)\boldsymbol{K}\,{}^{t+\Delta t}\boldsymbol{U} - \alpha\boldsymbol{K}\,{}^{t}\boldsymbol{U} = (1+\alpha)\,{}^{t+\Delta t}\boldsymbol{R} - \alpha\,{}^{t}\boldsymbol{R} \tag{69}$$

which can be rewritten in the following effective form:

$$\widehat{K}\,{}^{t+\Delta t}\boldsymbol{U} = {}^{t+\Delta t}\widehat{\boldsymbol{R}} \tag{70}$$

where $\widehat{K}$ is the effective (tangential) stiffness matrix and ${}^{t+\Delta t}\widehat{\boldsymbol{R}}$ is the effective load.

$$\widehat{K} = a_0\boldsymbol{M} + a_1(1+\alpha)\boldsymbol{C} + (1+\alpha)\boldsymbol{K} \tag{71}$$

$${}^{t+\Delta t}\widehat{\boldsymbol{R}} = \boldsymbol{M}\left(a_0\,{}^{t}\boldsymbol{U} + a_2\,{}^{t}\dot{\boldsymbol{U}} + a_3\,{}^{t}\ddot{\boldsymbol{U}}\right) + \boldsymbol{C}\left[a_1(1+\alpha)\,{}^{t}\boldsymbol{U} + \{a_4(1+\alpha)+\alpha\}\,{}^{t}\dot{\boldsymbol{U}} + a_5(1+\alpha)\,{}^{t}\ddot{\boldsymbol{U}}\right] + \alpha\boldsymbol{K}\,{}^{t}\boldsymbol{U} + (1+\alpha)\,{}^{t+\Delta t}\boldsymbol{R} - \alpha\,{}^{t}\boldsymbol{R} \tag{72}$$

The effective load can be written in the following form for computational efficiency:

$${}^{t+\Delta t}\widehat{\boldsymbol{R}} = \left[a_0\boldsymbol{M} + a_1(1+\alpha)\boldsymbol{C} + \alpha\boldsymbol{K}\right]{}^{t}\boldsymbol{U} + \left[a_2\boldsymbol{M} + \{a_4(1+\alpha)+\alpha\}\boldsymbol{C}\right]{}^{t}\dot{\boldsymbol{U}} + \left[a_3\boldsymbol{M} + a_5(1+\alpha)\boldsymbol{C}\right]{}^{t}\ddot{\boldsymbol{U}} + (1+\alpha)\,{}^{t+\Delta t}\boldsymbol{R} - \alpha\,{}^{t}\boldsymbol{R} \tag{73}$$

3. FINITE ELEMENT MODULE

*PULSE*_FE is programmed following the concepts outlined in Section 2 using the C# (C-Sharp) programming language in Microsoft® Visual Studio® Community.

An input file is generated after meshing the pavement structure using Gmsh, which is an open-source 2D and 3D FE mesh generator with a built-in computer-aided design (CAD) engine and post-processor (Geuzaine & Remacle, 2009). The input file is then manually processed before it becomes readily available for use by *PULSE*_FE. It should be noted that the input file has the same format as ABAQUS, making the validation process simpler.

*PULSE*_FE then reads the nodes' coordinates and the elements' connectivity from the input file and performs the static or dynamic analysis, based on the user's preference, following the flowchart shown in Figure 18.

The FE mass, damping, and stiffness matrices are sparse and are stored in a compressed sparse column format for reducing the storage requirements and for allowing faster matrices operations.

*PULSE*_FE has no built-in system of units. All input data must be specified in consistent units, with examples shown in Table 1. The comparison between *PULSE*_FE and ABAQUS, described in Section 3.3, uses the U.S. Unit (inch) system.

In the following sections, the mesh generator Gmsh is illustrated for a three-layer pavement structure. Then, the storage of the matrices using the different formats is discussed. Finally, the capabilities of the *PULSE*_FE module are demonstrated for a three-layer pavement structure under static and dynamic loading by comparing the results to the surface deflections obtained using ABAQUS.

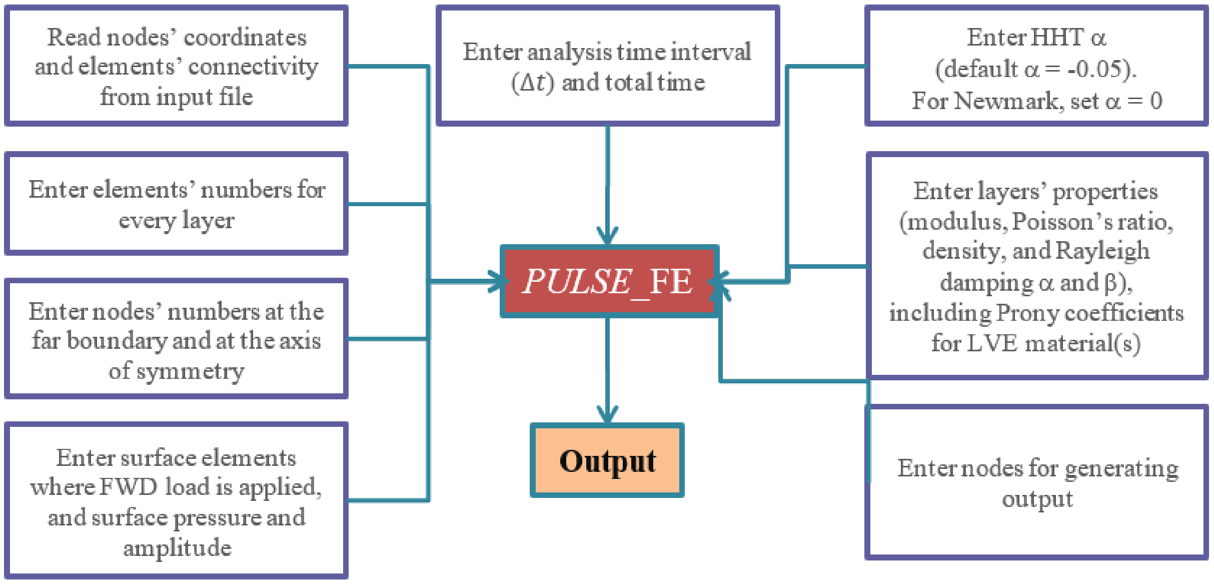

Read nodes' coordinates and elements' connectivity from input file
Enter elements' numbers for every layer
Enter nodes' numbers at the far boundary and at the axis of symmetry
Enter surface elements where FWD load is applied, and surface pressure and amplitude
Enter analysis time interval (Δt) and total time
PULSE_FE
Output
Enter HHT α (default α = -0.05). For Newmark, set α = 0
Enter layers' properties (modulus, Poisson's ratio, density, and Rayleigh damping α and β), including Prony coefficients for LVE material(s)
Enter nodes for generating output

Figure 18. *PULSE*_FE Flowchart

Table 1. Consistent Units

Quantity	SI	SI (mm)	U.S. Unit (ft)	U.S. Unit (in.)
Length	m	mm	ft	in.
Force	N	N	lbf	lbf
Mass	kg	tonne (10^3 kg)	slug	lbf.s^2/in.
Time	s	s	s	s
Stress	Pa (N/m^2)	MPa (N/mm^2)	lbf/ft^2	psi (lbf/in.^{2})
Energy	J	mJ (10^{-3} J)	lbf.ft	lbf.in
Density	kg/m^3	tonne/mm^3	slug/ft^3	lbf.s^2/in.^{4}

J = Joules
N = Newtons
Pa = Pascal
SI = International System of Units

mJ = Megajoule
MPa = MegaPascal

3.1 GMSH

The selection of a structured mesh versus a combination of structured and unstructured meshes depends on the modeling of the infinite media. The far boundary method that requires the use of an unstructured mesh is initially adopted. The use of triangular or quadrilateral elements in the infinite media is very time efficient since the size of the elements increases as they get closer to the far boundary and does not generate excessively large models.

Figure 19 shows a thin flexible pavement sample model, consisting of a 3-inch AC layer, a 12-inch aggregate base layer, and a subgrade layer that extends to 250 ft to simulate the infinite media using the far boundary method. The Front-Delaunay 2D algorithm is used to generate the mesh using Gmsh. A sensitivity analysis was performed to generate the optimal model, in terms of accuracy and mesh density. The elements are as small as a quarter of an inch under the load, and they grow monotonically to the far boundary. An unstructured mesh is selected because it is more appropriate for this model, where the elements expand proportionally with distance without producing distorted elements.

This model consists of 19,560 nodes and 38,421 linear triangular elements. An equivalent structured mesh with quarter-inch elements would require 48,000 square elements or 96,000 triangular elements for the two 200-inch-wide surface layers, and a much higher number of elements for the entire model.

It is important to note that Gmsh (gmsh.info) has an application programming interface (API) that can be used to integrate it into *PULSE*_FE.

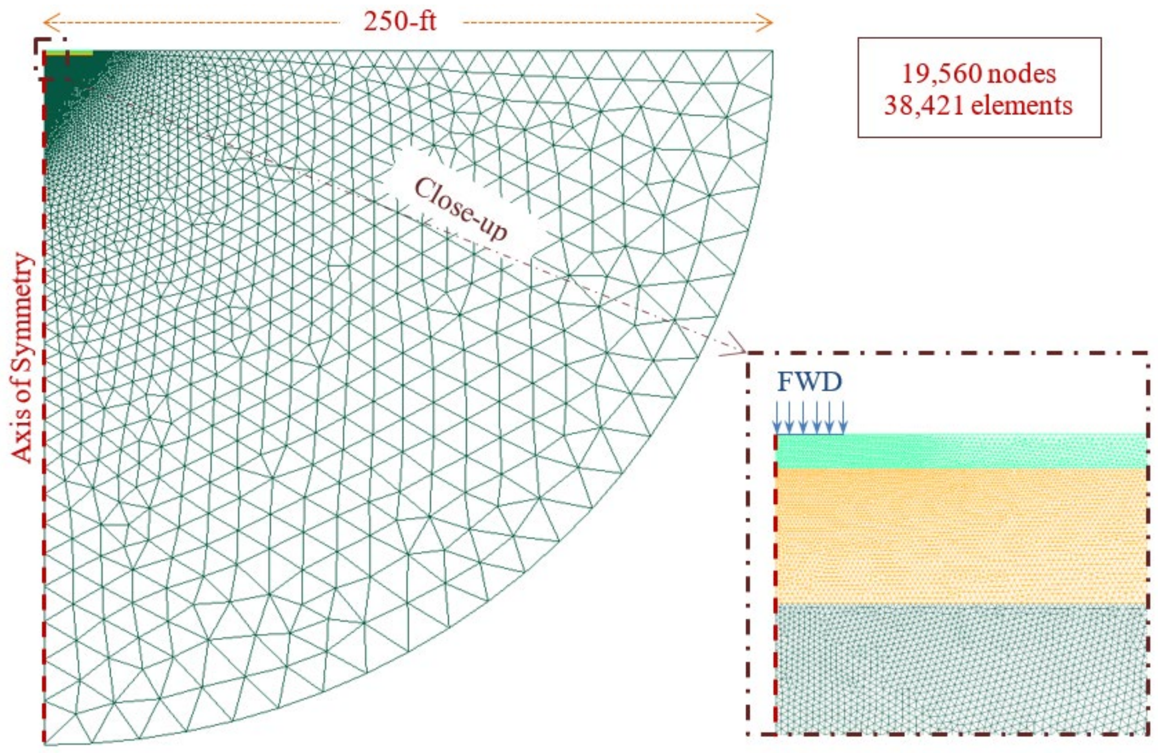


Figure 19. Thin Flexible Pavement Model Produced using Gmsh (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

3.2 MATRICES STORAGE

A sparse matrix is one that is composed of mostly zero values. Sparse matrices are distinct from matrices with mostly non-zero values, which are referred to as dense matrices. A matrix is typically stored in a 2D array, where each element is identified by two indices representing the row and column indices. The amount of memory required to store large matrices is significant, therefore sparse matrices are typically stored in different formats. Sparse matrices reduce the memory required by storing only the non-zero elements and allow for faster matrices operations. The FE stiffness, damping, and mass matrices are sparse matrices because most of the elements are zeros.

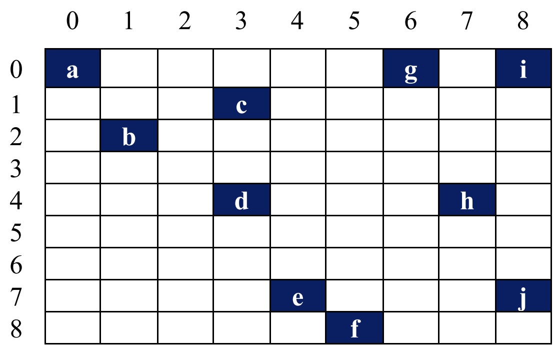
Different formats are available to store sparse matrices including, but not limited to, coordinate list (COO), compressed sparse row (CSR), and compressed sparse column (CSC). COO is a fast format for constructing the sparse matrices or generating the sparsity pattern. COO is then converted into CSR or CSC for efficient access and matrix operations. CSR is good for row-wise slicing, whereas CSC is good for column-wise slicing.

The COO format stores the row indices, column indices, and values for the non-zero elements (nnz) in three arrays, where each array has a length equal to the number of the nnz. The CSR and CSC are similar to COO, but compress the row indices and the column indices, respectively.

The CSR has the same column indices and values arrays as the COO following a row-major sorting order. The third array is a row pointer having a length equal to nr + 1, where nr is the number of rows. The row pointer array has one element per row showing the index where the given row starts, and its last element is set equal to nnz.

The CSC has the same row indices and values arrays as the COO following a column-major sorting order. The third array is a column pointer having a length equal to nc + 1, where nc is the number of columns. The column pointer array has one element per column showing the index where the given column starts, and its last element is set equal to nnz.

Figure 20 shows an example of the COO, CSR, and CSC storage formats for a sparse matrix with 9 rows, 9 columns, and 10 non-zero elements. The row and column indices use zero-based indexing, i.e., the initial position in each array is zero.



COO Format:

Row Index	0	2	1	4	7	8	0	4	0	7
Column Index	0	1	3	3	4	5	6	7	8	8
Value	a	b	c	d	e	f	g	h	i	j

CSR Format:

Row Pointer	0	3	4	5	5	7	7	7	9	10

Column Index	0	6	8	3	1	3	7	4	8	5
Value	a	g	i	c	b	d	h	e	j	f

CSC Format:

Column Pointer	0	1	2	2	4	5	6	7	8	10

Row Index	0	2	1	4	7	8	0	4	0	7
Value	a	b	c	d	e	f	g	h	i	j

Figure 20. Sparse Matrix COO, CSR, and CSC Examples (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

This matrix in Figure 20 has a density of 12 percent and a sparsity of 88 percent, where the density and sparsity are calculated in accordance with the following formulas.

$$\text{Density (\%)} = \frac{\text{nnz}}{\text{nr} \times \text{nc}} \times 100 \qquad (74)$$

$$\text{Sparsity (\%)} = 100 - \text{Density (\%)} \quad (75)$$

A typical pavement model with 24,000 nodes or 48,000 degrees of freedom has matrices with 0.03 percent density or 99.97 percent sparsity, indicating that most elements are zeros.

Finally, the COO format is used in the *PULSE*_FE module for generating the sparsity pattern, and CSC format is used for storing and processing all matrices over the related CSR format to ensure compatibility with the CSparse.Net library.

3.3 STATIC AND DYNAMIC ANALYSES COMPARISON WITH ABAQUS

The *PULSE*_FE module is validated by comparing the surface deflections to the ABAQUS software under static and dynamic loading. A three-layer flexible pavement structure consisting of a 3-inch AC layer, a 12-inch aggregate base layer, and a subgrade layer is considered. The pavement structure is meshed using Gmsh with only 986 nodes and 1,827 linear triangular elements. A coarse mesh is used to limit the number of nodes to 1,000.

The model is loaded using a 9,000-lb simulated FWD load applied uniformly over a circular area with a 6-inch radius. For the dynamic analysis, the FWD load is modeled using a 40-ms haversine pulse. All layers are modeled as linear elastic isotropic materials for the static analysis as illustrated in Table 2, and the AC layer is modeled as linear viscoelastic (LVE) for the dynamic analysis.

Table 3 shows the Prony coefficients for the AC layer. The modulus of elasticity and Poisson's ratio are used to generate the stiffness matrix for the static and dynamic analysis, respectively. The density and Rayleigh damping coefficients are used to generate the mass and damping matrices for the dynamic analysis, respectively.

It is important to note that ABAQUS is limited to a maximum of 13 Prony coefficients, which is not the case for *PULSE*_FE. This limitlessness is important for modeling, and specifically for the dynamic backcalculation of the master curve, which will be illustrated in future studies.

Table 2. Layer Thicknesses and Properties

Layer	**Type**	**Thickness (in.)**	**Modulus (ksi)**	**Poisson's Ratio**	**Density (pcf)**	**Rayleigh Damping Coefficients**	
						α_R	β_R
AC	Linear Elastic	3	500	0.35	150	10	0.001
	LVE	3	2,760	0.35	150	0	0
Aggregate Base	Linear Elastic	12	50	0.40	120	20	0.002
Subgrade	Linear Elastic	—	5	0.45	100	30	0.003

Table 3. Prony Coefficients

Term j	$\log(\tau_j)$	Relative Modulus α_j
1	-7	0.252
2	-6	0.108
3	-5	0.227
4	-4	0.18
5	-3	0.1355
6	-2	0.058
7	-1	0.0242
8	0	0.0074
9	1	0.00337
10	2	0.0011
11	3	0.0007
12	4	0.0001
13	5	0.00045

3.3.1 Static Analysis

Figure 21 shows the surface deflection basins calculated using the *PULSE*_FE module and ABAQUS. The deflections are considered to be identical for practical purposes as demonstrated with a maximum difference of 1.68×10^{-6} mils or 5.34×10^{-8} percent between the results.

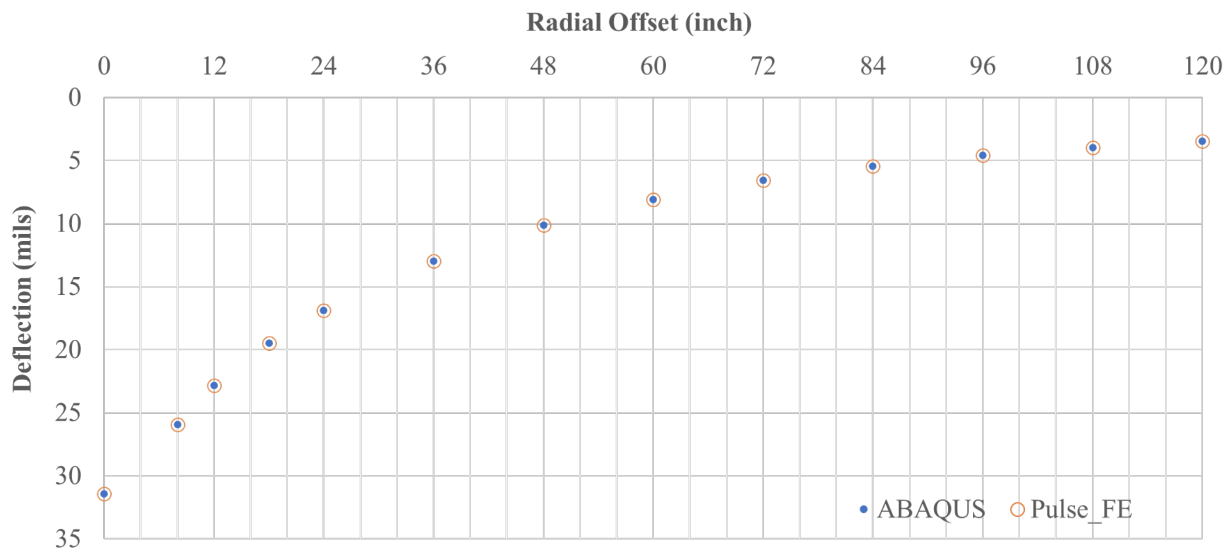


Figure 21. *PULSE*_FE and ABAQUS FWD Surface Deflections (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

3.3.2 Dynamic Analysis

Figure 22 shows the *PULSE*_FE deflection time histories at various offsets ranging from 0 to 72 inches, along with the simulated FWD dynamic load. The HHT-α numerical integration procedure is followed with $\alpha = -0.05$ resulting in $\beta = 0.275625$ and $\gamma = 0.55$.

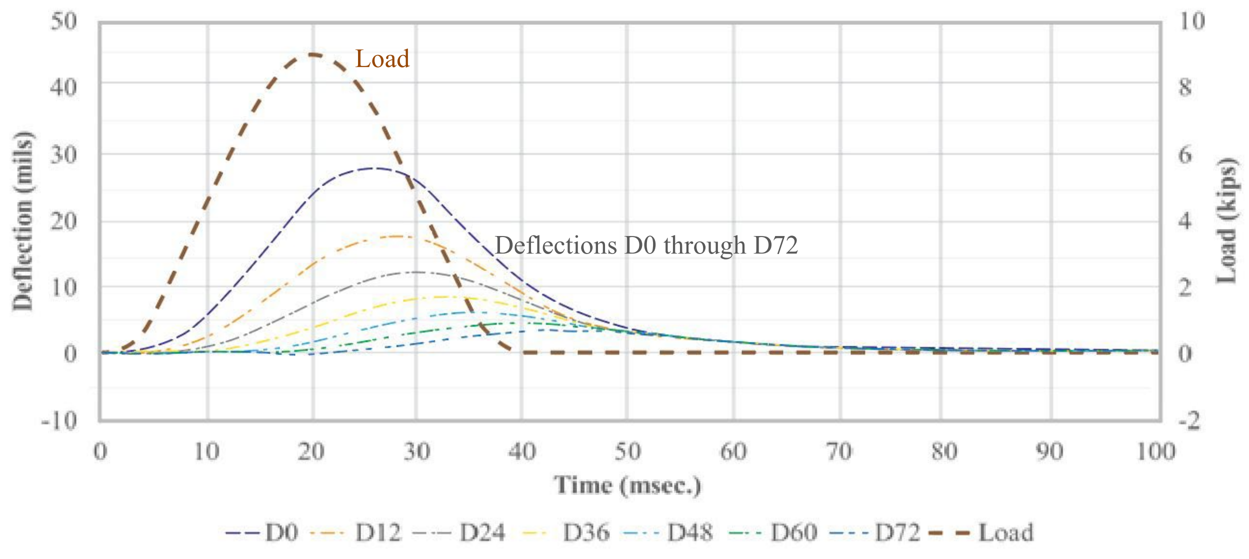


Figure 22. *PULSE*_FE Deflection Time Histories at Various Radial Offsets ((Bazi, Saboundjian, Bou Assi, & Diab, 2020)

Figures 23 and 24 compare the vertical (axial) and radial deflections, respectively, for selected offsets using *PULSE*_FE and ABAQUS. The deflections are identical with maximum difference of 4.49×10^{-7} mils or 2.34×10^{-6} percent. It should be noted that the radial deflections are not considered in the dynamic backcalculation, and they are simply compared for validation purposes.

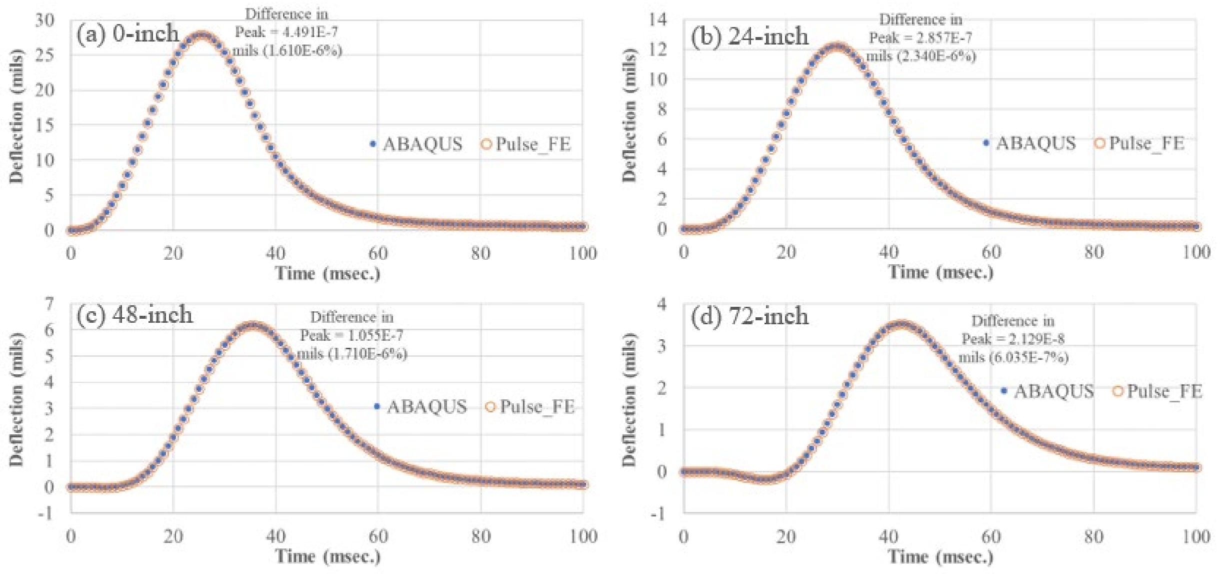


Figure 23. *PULSE*_FE and ABAQUS Vertical Surface Deflections at 0-, 24-, 48-, and 72-inch Offsets using HHT-α Method (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

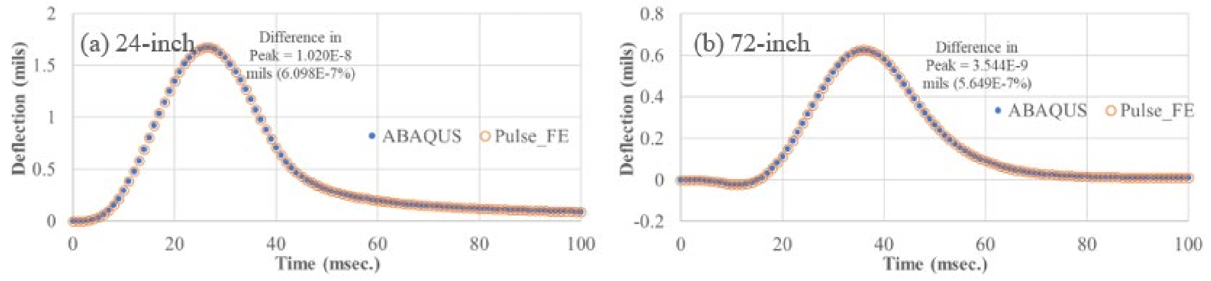


Figure 24. *PULSE*_FE and ABAQUS Radial Surface Deflections at 24- and 72-inch Offsets using HHT-α Method ((Bazi, Saboundjian, Bou Assi, & Diab, 2020)

3.3.3 *PULSE*_FE Computational Efficiency

*PULSE*_FE is currently programmed to use one computer processor. Using a laptop with an Intel core i7 processor and Microsoft® Windows® 10 Pro operating system (a standard laptop), *PULSE*_FE is able to solve the system for a linear elastic dense model with 24,000 nodes in 5 seconds using 100 time steps (e.g., every 1 ms for a 100 ms duration) and in 6 seconds using 200 time steps. On the other hand, ABAQUS completed the same tasks with one processor in about 5 and 10 minutes for the 100 and 200 steps, respectively. Increasing the number of parallel processors in ABAQUS from one to three reduces the analysis time in half.

When LVE material is considered in the modeling, the *PULSE*_FE computation time is slightly increased from 5 seconds to under 10 seconds using 100 time steps. The increase in time is linearly proportional to the number of LVE elements and the number of time steps.

In summary, *PULSE*_FE is shown to be computationally efficient by completing the task in 1 to 3 percent less time than ABAQUS takes, making the dynamic backcalculation feasible.

4. DYNAMIC BACKCALCULATION UPDATE

An application, *PULSE* 2019, was developed for the dynamic backcalculation that uses FE modeling for forward calculation and the Newton-Raphson method for improving the variables (Bazi & Bou Assi, 2022). The application reliably predicted the AC master curve for several simulated pavement structures and mixes at different temperatures. The application was upgraded to *PULSE* 2020, as part of the FAA project, to improve the master curve prediction. This improvement was achieved by better estimating the Jacobian matrix and by using additional FWD parameters that are critical for the backcalculation process.

4.1 MASTER CURVE PREDICTION HISTORICAL BACKGROUND

Over the past several years, researchers from Michigan State University (MSU) and Iowa State University (ISU) attempted to develop the AC master curve from FWD data using dynamic backcalculation. The initial results showed the need for more research and validation before a software tool could be made available.

Kutay, Chatti, and Lei (2011) from MSU pioneered the effort to predict the damaged master curve of the AC layer from FWD data. In their method, they used a layered, viscoelastic-forward algorithm in an iterative backcalculation procedure. The master curve was reliably predicted using simulated FWD data for frequencies above 10^{-3} Hz. Researchers recommend improvements in FWD technology and test procedures.

Varma, Kutay, and Chatti (2013) from MSU indicated that the master curve development requires more data than the surface deflection time-histories of a single FWD drop. They suggest performing the FWD testing at different temperatures in the range of 68 °F to 122 °F to maximize the portion of the master curve that can be reliably backcalculated.

Gopalakrishnan et al. (2014, 2015) from ISU investigated the feasibility of employing neural networks (NNs) to backcalculate the master curve using the same layered viscoelastic forward analysis tool developed by MSU. Researchers indicated that the current prediction accuracies are not sufficient to recommend these models for practical implementation.

Zaabar, Chatti, Lee, and Lajnef (2014) from MSU used a time-domain viscoelastic dynamic solution as a forward routine and a genetic algorithm for backcalculation analysis. Field FWD load and deflection time histories from three sites were used for validation. The new algorithm was capable of reliably backcalculating the master curve of the AC layer.

Varma and Kutay (2016) from MSU used a layered viscoelastic-nonlinear forward model to develop a genetic algorithm-based backcalculation scheme. The study showed that running FWD at two different temperatures can be sufficient to compute the master curve of asphalt pavements and the nonlinear properties of unbound layers. The algorithm is validated using two FWD test runs at a long-term pavement performance (LTPP) test section.

Lee, Ayyala, and Von Quintus (2017) conducted dynamic backcalculation on two LTPP sections. The backcalculated master curves were significantly different from those constructed using laboratory testing data.

Hamim et al. (2020) recently used artificial neural network (ANN) models designed using the FWD deflection-time history data obtained by the FE method to predict the master curve. The study evaluated two ANN models with one model demonstrated to be more accurate than the other.

4.2 *PULSE* 2020 UPGRADE

Several features were added to the *PULSE* 2020 application, as discussed in this report, to better estimate the Jacobian matrix for the Newton-Raphson method and to evaluate additional deflection parameters for improving the AC master curve prediction. The application was also upgraded to the same .NET framework using the C# programming language. The ABAQUS FE solver was used in this study for the forward calculation of the pavement surface deflections through a dynamic implicit analysis. The *PULSE*_FE module was used for the forward calculation.

The AC layer for flexible pavements was modeled as an LVE material, and all other layers, including the Portland cement concrete layer for rigid pavements, were modeled as linear elastic with damping. Figure 25 shows a detailed flowchart of the *PULSE* application.

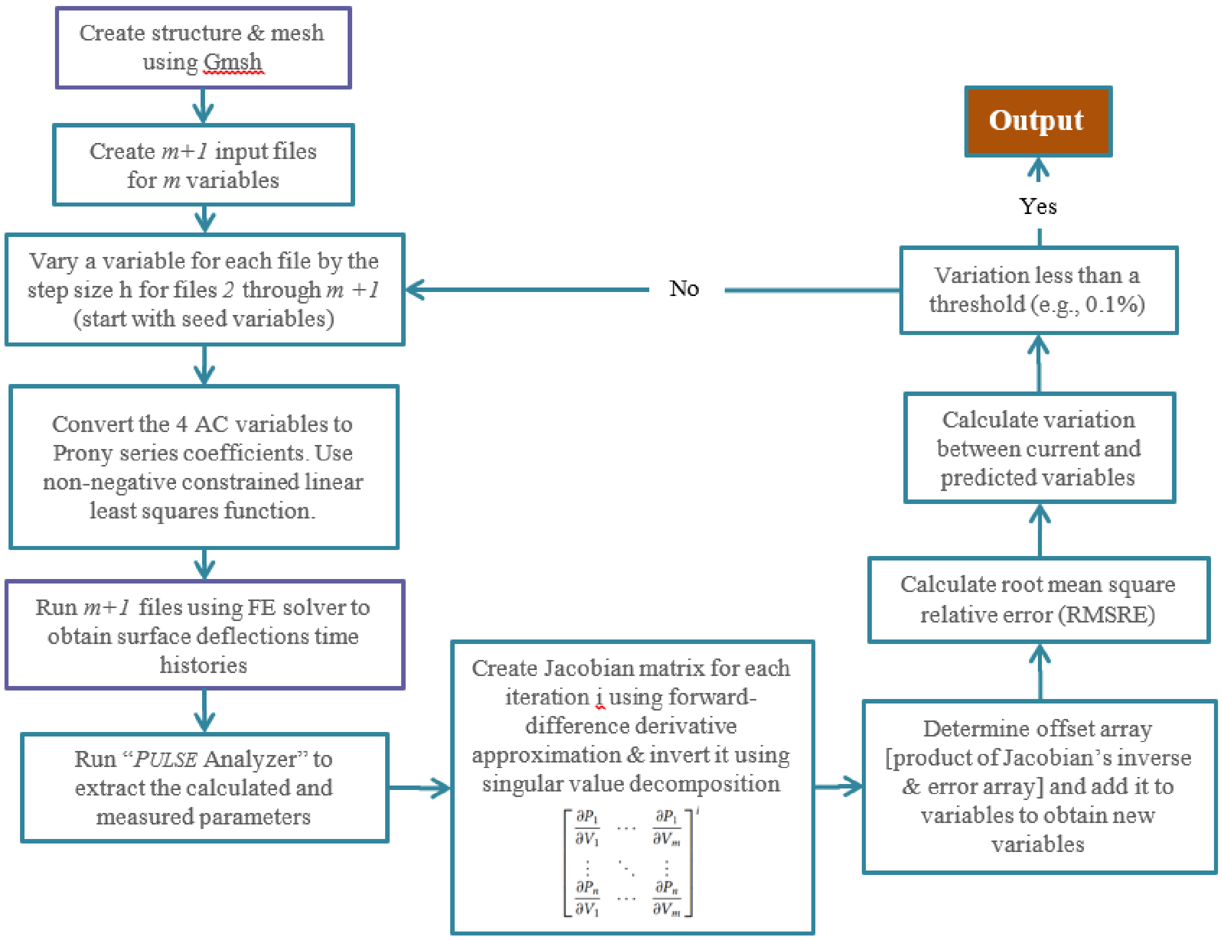

Create structure & mesh using Gmsh
Create $m+1$ input files for m variables
Vary a variable for each file by the step size h for files 2 through $m+1$ (start with seed variables)
Convert the 4 AC variables to Prony series coefficients. Use non-negative constrained linear least squares function.
Run $m+1$ files using FE solver to obtain surface deflections time histories
Run "PULSE Analyzer" to extract the calculated and measured parameters
Create Jacobian matrix for each iteration i using forward-difference derivative approximation & invert it using singular value decomposition
Determine offset array [product of Jacobian's inverse & error array] and add it to variables to obtain new variables
Calculate root mean square relative error (RMSRE)
Calculate variation between current and predicted variables
Variation less than a threshold (e.g., 0.1%)
No
Yes
Output

Figure 25. *PULSE* 2020 Application Flowchart (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

4.2.1 Asphalt Concrete LVE Behavior

The LVE behavior of AC is considered using the master curve's reduced sigmoidal function (Equation 76). This model is slightly different than the typical model, where a coefficient β' is used instead of β. The sigmoidal function's exponent $\beta + \gamma \times \log(f_r)$ in the standard model is rewritten in the form of $\beta' + \gamma \times \log(f)$, where $\beta' = \beta + \gamma \times \log[a(T)]$. This substitution allows the determination of the four sigmoidal coefficients at any temperature T, without including a time-temperature superposition model (Bazi & Bou Assi, 2022). The master curve at the reference temperature is then determined using backcalculated variables from a minimum of two temperatures.

$$logE = \delta + \frac{\alpha}{1+e^{\beta' + \gamma\times\log(f)}} \tag{76}$$

where:

> E = time–temperature-dependent relaxation modulus
> δ, α, β', and γ = fit constants
> 10^{δ} = minimum modulus
> $10^{\delta+\alpha}$ = maximum modulus
> γ = steepness of the function
> $\beta' = \beta + \gamma \times \log[a(T)]$
> f = frequency
> T = Temperature
> log[a(T)] = shift factor

The four master curve coefficients (variables) are used to calculate the Prony series coefficients for use in the FE model (Zaabar, Chatti, Lee, & Lajnef, 2014; Bazi & Bou Assi, 2022).

4.2.2 Newton's Method for Approximating Roots

Newton's method, also known as the Newton-Raphson method, is a root-finding algorithm that produces iteratively better approximations to the roots of a function following Equation 77.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{77}$$

The process starts with an initial guess, and a better approximation of the root is obtained after every iteration. Figure 26 provides an illustration of how the root of a function f(x) = -0.5 -2 × ln(2-x) is better approximated starting at x_0 = 1.9500 and reaching the root x_5 = 1.2212 in five iterations, where the root is accurate to four decimal figures. The ordinates and slopes are provided in Figure 26 for verification using Equation 77. Newton's method is efficient in reaching a solution in a few iterations, which is also the proven case for dynamic backcalculation.

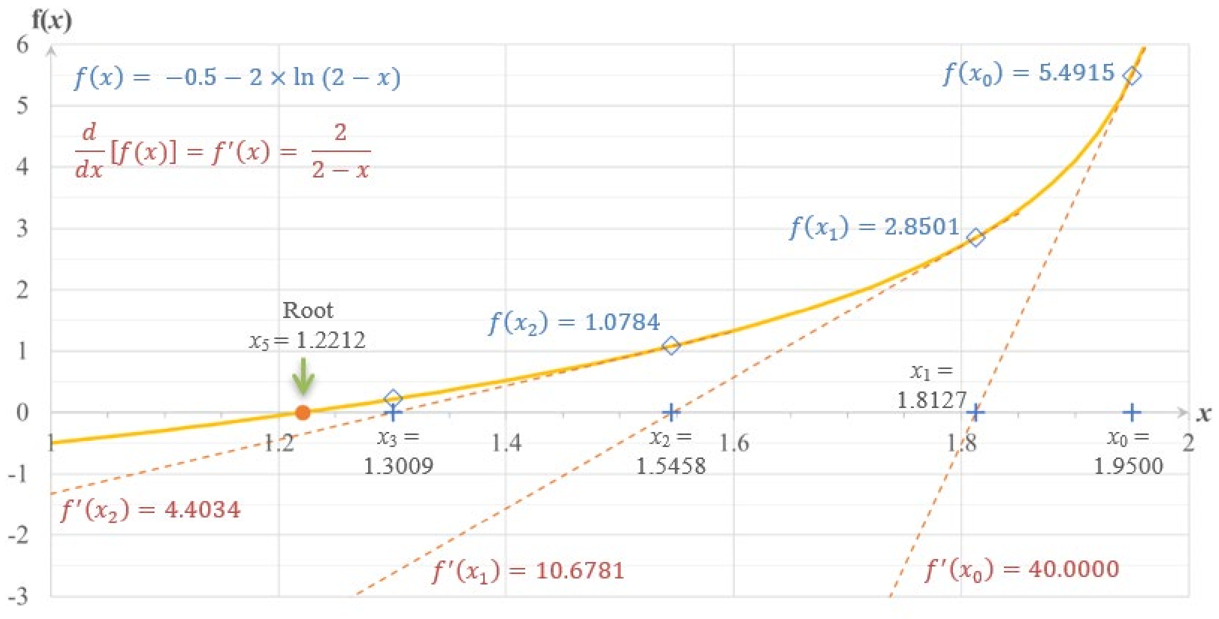


Figure 26. Example of Newton's Method for Obtaining the Root of f(x) = -0.5 -2×ln(2-x) Starting with x_0 = 1.9500 (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

The multivariate Newton's method is used in the dynamic backcalculation by formulating the problem using Equation 78 (Harichandran et al., 1993; Chatti, Ji, & Harichandran, 2004; Bazi & Bou Assi, 2022).

$$\begin{bmatrix} \frac{\partial P_1}{\partial V_1} & \cdots & \frac{\partial P_1}{\partial V_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial P_n}{\partial V_1} & \cdots & \frac{\partial P_n}{\partial V_m} \end{bmatrix}^i \begin{bmatrix} \partial V_1 \\ \vdots \\ \partial V_m \end{bmatrix}^i = \begin{bmatrix} P_1 \\ \vdots \\ P_n \end{bmatrix} - \begin{bmatrix} \widehat{P_1} \\ \vdots \\ \widehat{P_n} \end{bmatrix}^i = \begin{bmatrix} P_1 - \widehat{P_1}^{\,i} \\ \vdots \\ P_n - \widehat{P_n}^{\,i} \end{bmatrix} \tag{78}$$

where:

$\widehat{P_{1:n}}$ = Calculated parameters

$P_{1:n}$ = Measured parameters

$V_{1:m}$= Variables

$\frac{\partial P_{1:n}}{\partial V_{1:m}}$ = First-order derivative (slope)

i = Iteration number

The offset array, $\begin{bmatrix} \partial V_1 \\ \vdots \\ \partial V_m \end{bmatrix}^i$, is determined for every iteration by multiplying the inverse of the Jacobian matrix (matrix with first-order derivatives) by the error array, $\begin{bmatrix} P_1 - \widehat{P_1}^i \\ \vdots \\ P_n - \widehat{P_n}^i \end{bmatrix}$, and adding it to the previously determined variables. The Jacobian matrix is inverted using the singular-value decomposition (SVD) process.

4.2.2.1 Newton's Method Step Size

The Jacobian matrix is populated numerically using the finite-difference derivative approximation since the analytical expressions of the partial derivatives are not available. A step size, h, is used to vary one variable at a time and see the effect of such variation on the response parameters. In the former *PULSE* 2019 application, the master curve variables δ, α, β′, and γ were simply multiplied by 1+h using the forward finite-difference, which results in a variation that is different than h for the AC moduli. This is depicted by the dotted lines in Figure 27, where a step size of 0.1, or 10 percent, results in a variation in the AC moduli in the range of –19.9 percent and 137.4 percent at the FWD's most dominant frequency of 17 Hz [log(17) = 1.23].

To address this limitation and to improve the slope calculation, four equations (79 through 82) were developed to calculate the master curve variables (δ_h, α_h, β'_h, and γ_h) that would cause a step size (=h) variation in the AC moduli at a preselected frequency. This is illustrated by the solid lines in Figure 27, where the variation is exactly equal to 10 percent at 17 Hz. All equations shown below are frequency dependent, except for the δ_h equation.

$$\delta_h = \delta + log(1 + h) \tag{79}$$

$$\alpha_h = \left[\frac{\alpha}{1+e^{\beta' + \gamma.\log f}} + log(1 + h)\right] \times \left(1 + e^{\beta' + \gamma.\log f}\right) \tag{80}$$

$$\beta'_h = ln\left[\frac{\alpha}{\frac{\alpha}{1+e^{\beta' + \gamma.\log f}} + log(1+h)} - 1\right] - \gamma.\log f \tag{81}$$

$$\gamma_h = \frac{ln\left[\frac{\alpha}{\frac{\alpha}{1+e^{\beta' + \gamma.\log f}} + log(1+h)} - 1\right] - \beta'}{log f} \tag{82}$$

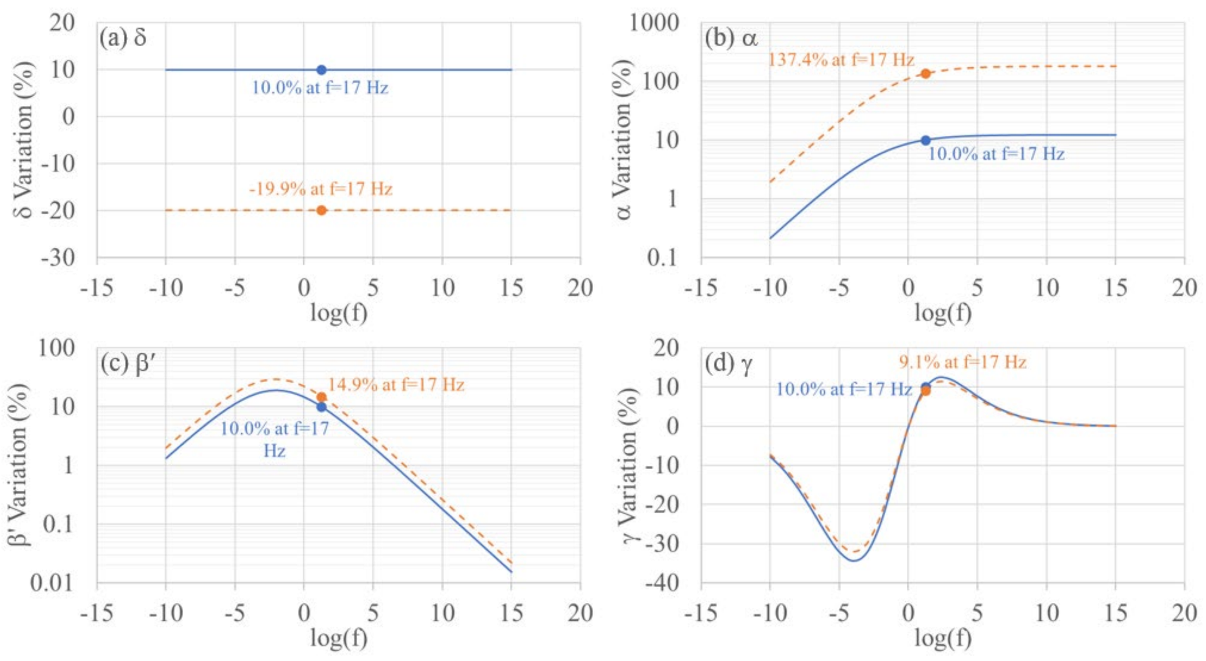


Figure 27. Variations in AC Moduli Due to Variations in Sigmoidal Function Coefficients using a Step Size h = 0.1 (10 percent)

4.2.3 Backcalculation Parameters

The FWD deflection time histories contain information that is unique to each pavement structure. The static backcalculation methods only use peak deflections, whereas the traditional dynamic backcalculation methods use peak deflections and the time delay or lag between the pulses. The parameters that use peak deflections and time delays may be adequate for the dynamic backcalculation of the pavement variables, but they are not sufficient to reliably obtain the AC master curve. Bazi and Bou Assi (2022) used additional parameters in the *PULSE* 2019 application to capture the shape and magnitude of the deflection time history for every FWD sensor, and they were successful in obtaining the master curve for most of the evaluated mixes. In the *PULSE* 2020 application, Bazi and Bou Assi (2022) explored additional parameters that are critical for the backcalculation process.

The *PULSE* 2019 parameters, shown in orange in Figure 28, include the peak deflection (D_{Peak}) that occurs at time T_{DPeak}; the times to the left and right of the peak that correspond to 50 percent of the peak deflection T_{50L} and T_{50R}, respectively; and the duration of the pulse at 50 percent of the peak deflection calculated as the difference between T_{50R} and T_{50L} (Dur_{50}).

In addition to the 5 parameters that are listed above, 17 additional parameters are explored for use in the *PULSE* 2020 application—for a total of 22 parameters—and they are shown in bold in Figures 28 and 29. Figure 28 shows the time history of the surface deflection, whereas Figure 29 shows the time history of the surface velocity.

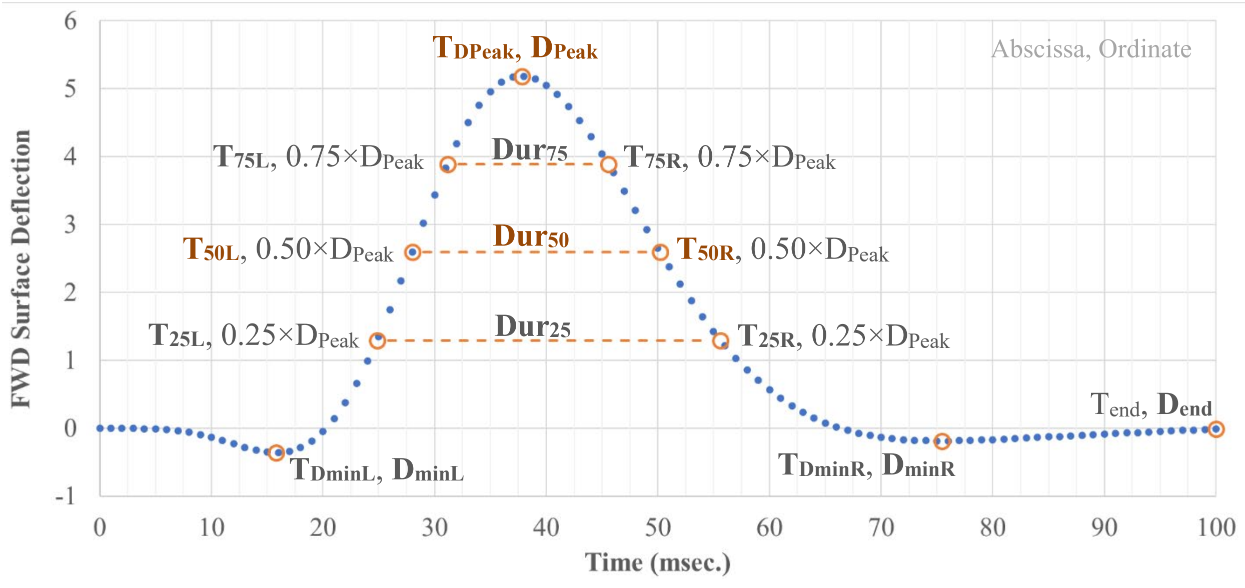


Figure 28. Typical FWD Surface Deflections Time Histories

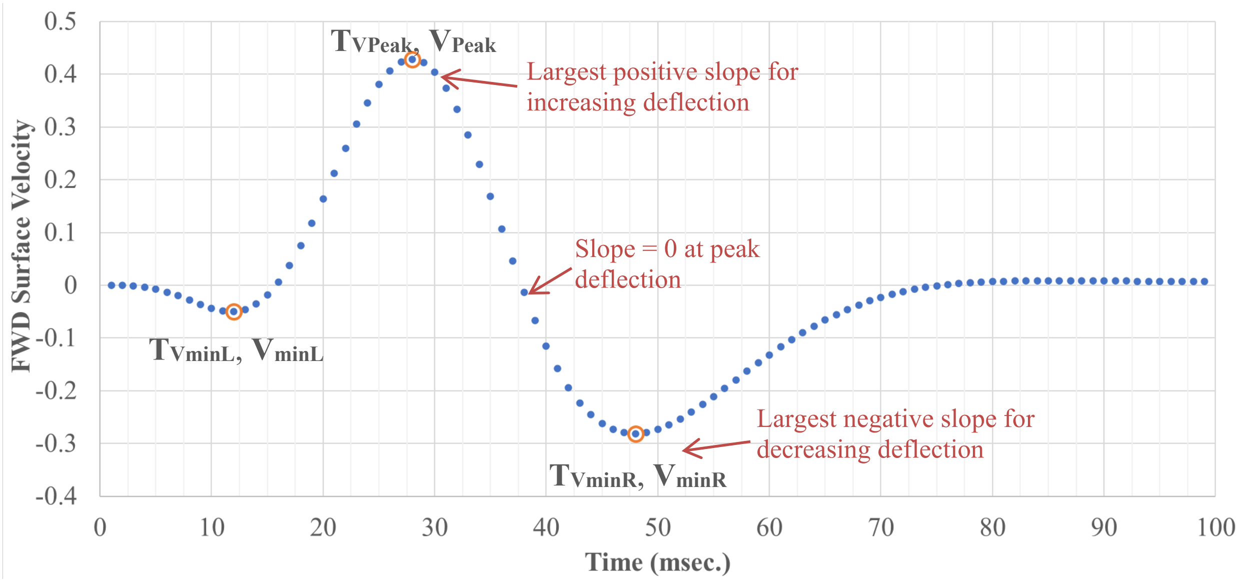


Figure 29. Typical FWD Surface Velocities Time Histories

All 22 parameters are extracted automatically for every sensor using a module named "*PULSE* Analyzer." The module uses a high-degree polynomial fitting along with its derivatives to accurately quantify all the parameters. The fitting is important to accurately obtain the FWD parameters since the FWD measured and calculated data are discrete and not continuous. The importance of fitting is illustrated in Figure 28 for T_{75R}, where the time does not coincide with a discrete data point (discrete points shown in blue). In this case, T_{75R} is the time to the right of the peak that corresponds to 75 percent of the peak deflection.

The plot of the surface velocity is determined using the central-difference finite approximation. The parameters are determined through data fitting of the *deflection* time histories.

Certain parameters, such as T_{DminL}, D_{minL}, T_{DminR}, D_{minR}, T_{VminL} and V_{minL}, are not available for all sensors and pavement structures; but when present, they need to be evaluated because they provide information about the pavement structures to be included in the dynamic backcalculation process. For example, the minimum negative deflection to the right of the peak, represented by T_{DminR} and D_{minR}, indicates the presence of a shallow stiff layer that produces the surface bouncing. Significant errors would then be expected if the stiff layer is not considered in the backcalculation model.

4.3 MASTER CURVE PREDICTION

The capabilities of the *PULSE* 2020 application are demonstrated for the same combinations of simulated flexible pavement structures and AC mixes used by Bazi and Bou Assi (2022). Those combinations consist of three flexible pavement structures with varying AC layer thicknesses ranging from 3 inches to 7 inches for mix A and one flexible structure with a 7-inch-thick AC layer for mix B at temperatures ranging from 30 °F to 100 °F.

Each combination had eight variables: four for the AC layer and two each for the aggregate base and subgrade layers. The target and seed (starting) values for the eight variables along with their absolute differences are shown in Table 4.

The backcalculation test results using *PULSE* 2019 are shown in Table 5 without any highlights for the 32 combinations. The frequency range (on a log scale), for which the AC moduli are determined to accuracies less than 1 percent, is shown for every combination. The maximum error in the AC moduli at 17 Hz and the unbound layers variables is shown between brackets. The AC master curve for frequencies larger than 10^{-1} or 10^{-2} Hz (log frequency = –1 or –2) are determined to less than 1 percent error for 17 out of the 32 combinations (53 percent).

The pavement variables are improved using the *PULSE* 2020 application (cells with green highlights) for the combinations where the maximum moduli are not obtained to accuracies less than 1 percent, thus, resulting in improved prediction of the AC master curves. The number of combinations, where the AC maximum moduli are predicted to accuracies less than 1 percent, has significantly increased from 17 to 28 combinations (an increase from 53 to 88 percent). This improvement can be attributed to (1) the use of four additional parameters (T_{VPeak}, V_{Peak}, T_{VminR}, and V_{minR}) for a total of nine parameters, and (2) the improvement in the Jacobian matrix calculation by using δ_h, α_h, β'_h, and γ_h, and the reduction in the step size from 0.1 to 0.01. V_{Peak} refers to the peak velocity or deflection slope that occurs at time T_{VPeak}. V_{minR} refers to the minimum velocity to the right of the peak deflection that occurs at time T_{VminR}. For the combinations where the master curve is better predicted, the errors in the AC modulus at 17 Hz and sublayer variables are also improved.

Table 4. Target and Seed Values of Variables[1, 2]

Layer	Absolute Difference	Variable	Target Value	Seed Value
AC (h_1 = 3, 5, & 7 inch)	Maximum E = $10^{\delta+\alpha}$ Mix A: 24% Mix B: 21% E at 17 Hz & 68 °F Mix A: 18% Mix B: 6%	Sigmoidal coefficient δ	Mix A: –0.966 Mix B: –0.134	Mix A: –1 Mix B: –0.3
		Sigmoidal coefficient α	Mix A: 4.523 Mix B: 3.703	Mix A: 4.65 Mix B: 3.95
		Sigmoidal coefficient β'	$\beta' = \beta + \gamma \times \log[a(T)]$ Mix A: –1.188–0.494×log[a(T)] Mix B: –1.417–0.548×log[a(T)]	Exact β′ + 0.2
		Sigmoidal coefficient γ	Mix A: –0.494 Mix B: –0.548	–0.65
Aggregate Base (h_2 = 12 inch)	25%	Modulus E_2	40 ksi	50 ksi
	50%	Rayleigh damping coefficient β_R	0.002 sec.	0.003 sec.
Subgrade	40%	Modulus E_3	5 ksi	7 ksi
	50%	Rayleigh damping coefficient β_R	0.002 sec.	0.001 sec.

[1]E_i = modulus of elasticity of ith layer; h_i = layer thickness of ith layer; i = 1, 2, and 3 correspond to surface, base, and subgrade layers, respectively.
[2]Ea is the activation energy used in the shift factor in accordance with the Arrhenius law (Ea =181,000 for mix A & 170,500 for mix B).
Reference temperature = 68 °F (293.15 K).

Table 5. Dynamic Backcalculation Frequency Range (log) and Maximum Error[1, 2]

Temp. (°F)	Mix A AC Thickness 3 in.	Mix A AC Thickness 5 in.	Mix A AC Thickness 7 in.	Mix B AC Thickness 7 in.
30	–1.6 to Max. [0.04%]	–2.1 to Max. [0.1%]	–2.0 to Max. [0.4%]	1.3 to 1.8 [1.0%]
				–5.1 to Max. [0.01%]
40	–3.9 to Max. [0.01%]	1.1 to 1.4 [1.4%]	1.0 to 2.6 [1.2%]	0.3 to Max. [2.7%]
		–1.4 to Max. [0.005%]	–0.7 to Max. [1.0%]	
50	–3.1 to Max. [0.02%]	–3.4 to Max. [0.06%]	–3 to Max. [0.1%]	–2.4 to Max. [0.3%]
60	–2.3 to Max. [0.008%]	–2.1 to Max. [0.03%]	–2.2 to Max. [0.07%]	–3.6 to Max. [1.2%]
70	–1.9 to Max. [0.003%]	–1.5 to Max. [0.02%]	–2.2 to 5.7 [0.2%]	–0.9 to Max. [0.14%]
			–1.9 to Max. [0.01%]	
80	–1.8 to Max. [0.001%]	–1.5 to 7.0 [0.007%]	0.6 to 2.1 [1.5%]	–0.7 to 6.7 [0.04%]
		–1.4 to 6.8 [0.009%]	–1.8 to Max. [0.005%]	–1.3 to Max. [0.03%]
90	–2.1 to 6.5 [0.005%]	–2.1 to 4.3 [0.03%]	0.5 to 2.5 [0.5%]	–0.6 to 4.9 [0.03%]
	–2.3 to 6.3 [0.02%]	–2.2 to 5.9 [0.03%]	–3.1 to Max. [0.01%]	–2.5 to Max. [0.005%]
100	–1.3 to 5.9 [0.02%]	–1.2 to 5.1 [0.01%]	–1.1 to 4.7 [0.01%]	0.4 to 2.2 [1.4%]
	–1.4 to 5.6 [0.01%]	–3.0 to Max. [0.002%]	–4.1 to Max. [0.001%]	–2.3 to Max. [0.003%]

[1] Cells highlighted in green were run using *PULSE* 2020.
[2] Max. = Maximum modulus.

4.4 PARAMETRIC STUDY

A parametric study was performed using FE axisymmetric modeling of FWD load to evaluate the influence of layer thicknesses, material properties, and presence of stiff layers on the FWD parameters. The parametric study consists of 15,552 combinations. The AC was modeled as LVE material by considering the master curve sigmoidal function variables δ, α, β, and γ (Table 6).

Table 6. Parametric Study Combinations

Pavement Type		Flexible[1]
Surface Layer	Thickness	*2 levels:* h_1 = 3 & 6 inch
	Property [Linear Viscoelastic][2]	*3 levels:* $\delta = -1.3, -1, \& -0.82$
		3 levels: $\alpha = 4.15, 4.5, \& 4.7$
		3 levels: $\beta = -1.57, -1.2, \& -0.73$
		3 levels: $\gamma = -0.8, -0.5, \& -0.12$
Aggregate Base Layer	Thickness	*2 levels:* h_2 = 6 & 12 inch
	Property [Linear Elastic]	*2 levels:* E_2 = 30 & 60 ksi
		1 level: $\beta_R = 0.002$
Subgrade Layer [Linear Elastic]		*2 levels:* E_3 = 5 & 15 ksi
Stiff Layer	Thickness[3] and Property	*3 levels:* None, 10 feet with E_4 = 100 ksi & 20 feet with E_4 = 100 ksi
Rayleigh Damping Parameters (Subgrade & Stiff Layers)		*2 levels:* α_R = 0 & 50
		2 levels: β_R = 0.002 & 0.006
FWD Load Level		*1 level:* 40 msec. haversine pulse with radius of loaded area = 6 inch, and Load level = 9 kips

[1]E_i = modulus of elasticity of ith layer; h_i = layer thickness of ith layer; i = 1, 2, 3, and 4 correspond to the surface, base, subgrade, and stiff layers, respectively.
[2]Variables were determined to cause a 50 percent variation in the modulus of the FWD's most dominant frequency of 17 Hz.
[3]Stiff layer thickness determined from surface.

The 22 FWD parameters were determined for each sensor of the 15,552 combinations, and a regression analysis was performed to understand the effect (positive or negative) and to quantify the contribution of the various predictor variables (pavement variables) on the response variables (FWD parameters). The best subset of four predictor variables was selected for each response variable, producing total contributions in the range of 40 to 95 percent for each response variable. An average contribution of 81 percent is calculated (Table 7). A similar analysis was conducted in previous research (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020), where only three FWD parameters were considered, and two levels of LVE AC behavior were modeled.

Table 7. Effects and Percent Contributions of Model Parameters on Response Variables

Response Variable at Offset (inch)		Predictor Variables (%)											Sum (%)
		h_1	δ	α	β	γ	h_2	E_2	E_3	h_{SL}	α_R	β_R	
DPeak	0	–23.24					–7.49	–5.99	–33.43				70.2
	24	–1.9					–3.64		–82.57			–2.52	90.6
	48								–86.46	+0.55	–4.23	–3.99	95.2
	72								–71.31	+3.15	–8.04	–6.88	89.4
T_{DPeak}	0	+11.08				–13.09			–53.54			+8.09	85.8
	24								–68.35	+1.48	–1.33	+21.13	92.3
	48	–2.99							–70.65	+4.13		+6.11	83.9
	72	–2.08					–2.47		–74.91	+6.46			85.9
DminL	0	–23.24					–7.49	–5.99	–33.43				70.2
	24	–1.90					–3.64		–82.57			–2.52	90.6
	48								–86.18	+0.54	–4.11	–4.12	94.9
	72								–72.03	+2.60	–6.60	–7.69	88.9
T_{DminL}	0												—
	24												—
	48	–15.57					–18.81	–9.68	–17.46				61.5
	72	–6.09					–9.11		–53.99			–5.90	75.1
DminR	0	–21.84					–6.81	–5.37	–35.83				69.8
	24						–2.99		–76.93		–3.28	–4.08	87.3
	48								–74.25	–1.73	–8.05	–6.29	90.3
	72								–62.79	–0.83	–13.09	–9.65	86.3
T_{DminR}	0								–41.66	+2.71	+0.28	+8.92	53.6
	24								–51.65	+5.75	+0.10	+8.39	65.9
	48								–50.79	+8.61	+0.01	+7.75	67.1
	72								–47.85	+8.81	+0.01	+5.88	62.5
Dur_{25}	0	+12.27				–11.13			–35.98			+15.50	74.9
	24	+3.0							–44.11	+11.63		+19.53	78.3
	48	+1.47							–29.67	+28.80		+17.73	77.6
	72								–4.28	+28.74	–0.80	+6.34	40.1
Dur_{50}	0	+13.08				–11.06			–39.53			+14.77	78.4
	24	+3.24							–48.45	+7.3		+20.47	79.5
	48	+1.76							–34.54	+20.75		+18.93	76.0
	72						+1.16		–14.95	+40.21		+18.92	75.2
Dur_{75}	0	+13.01				–10.94			–42.26			+15.76	82.0
	24	+3.08							–49.90	+6.19		+21.68	80.8
	48	+1.92							–35.74	+17.50		+19.54	74.7
	72						+1.37		–14.66	+36.33		+20.61	73.0
D_{End}	0	–23.94					–7.22	–5.75	–32.12				69.0
	24	–1.89					–3.57		–79.94			–1.66	87.1
	48								–76.31	+0.58	–3.53	–2.16	82.6
	72								–54.58	+1.76	–6.43	–3.60	66.3
T_{25L}	0	+7.98		+3.68		–14.74			–57.72				84.1
	24	–4.26				+1.78			–66.16			+13.98	86.2
	48	–10.18					–8.07	–3.79	–55.60				77.6
	72	–5.00					–6.92	–2.12	–68.57				82.6

Table 7. Effects and Percent Contributions of Model Parameters on Response Variables (Continued)

Response Variable at Offset (inch)		Predictor Variables (%)											Sum (%)
		h_1	δ	α	β	γ	h_2	E_2	E_3	h_{SL}	α_R	β_R	
T_{25R}	0	+11.98				–12.13			–40.63			+13.40	78.1
	24	+1.23							–52.82	+9.40		+20.45	83.9
	48				+0.03				–48.87	+23.10		+13.65	85.6
	72								–26.73	+24.80	–0.36	+3.39	55.3
T_{50L}	0	+9.37				–14.09			–56.44			+4.69	84.6
	24	–1.97							–68.77		–1.64	+17.04	89.4
	48	–7.70					–5.88	–2.76	–63.26				79.6
	72	–4.18					–5.42	–1.73	–73.19				84.5
T_{50R}	0	+12.19				–12.55			–46.85			+11.06	82.6
	24								–61.06	+4.63	–0.77	+21.43	87.9
	48						–0.21		–61.23	+13.11		+12.14	86.7
	72						–0.44		–56.55	+23.09		+7.09	87.2
T_{75L}	0	+10.17				–13.65			–55.41			+5.97	85.2
	24	–0.88							–69.42		–1.61	+18.89	90.8
	48	–5.76					–4.34		–67.68			+2.78	80.6
	72	–3.42					–4.25		–75.55	+2.57			85.8
T_{75R}	0	+11.73				–12.75			–50.68			+10.07	85.2
	24								–65.00	+2.91	–0.98	+21.91	90.8
	48	–0.90							–67.51	+8.25		+9.92	86.6
	72						–1.03		–66.35	+14.81		+4.74	86.9
VminL	0	–26.28					–7.43	–6.34	–26.44				66.5
	24	–2.70					–4.31		–77.13			–4.89	89.0
	48						–0.91		–79.93		–3.57	–8.12	92.5
	72						–1.47		–66.63		–4.40	–13.25	85.7
T_{VminL}	0												
	24												
	48	–16.54					–16.72	–11.47	–13.85				58.6
	72	–6.75					–7.91		–51.94			–6.78	73.4
VminR	0	–28.12				+6.65	–7.47		–21.19				63.4
	24	–3.71					–5.05		–68.03			–8.45	85.2
	48						–0.68		–71.43		–5.33	–13.44	90.9
	72	–0.51							–58.32		–9.37	–19.49	87.7
T_{VminR}	0	+10.71				–11.84			–55.73			+6.12	84.4
	24				+0.50				–70.00		–3.44	+17.64	91.6
	48	–1.65							–76.42		–2.78	+5.04	85.9
	72						–1.57		–79.15	+1.63		+2.28	84.6
VPeak	0	–26.28					–7.43	–6.34	–26.44				66.5
	24	–2.70					–4.31		–77.13			–4.89	89.0
	48						–0.51		–81.69		–3.93	–7.77	93.9
	72								–68.28	+1.17	–6.19	–12.54	88.2
T_{VPeak}	0	+9.90				–13.87			–56.12			+4.63	84.5
	24	–1.76							–68.61		–1.82	+17.69	89.9
	48	–7.03					–4.95	–2.39	–65.71				80.1
	72	–3.80					–4.27	–1.55	–75.78				85.4

The predictor variables in Table 7 consist of the AC layer thickness h_1, the AC master curve variables (δ, α, β, and γ), the base layer thickness h_2 and modulus E_2, the subgrade modulus E_3, the depth to the stiff layer h_{SL}, and the subgrade's Rayleigh damping coefficients α_R and β_R. A review of Table 7 shows that the subgrade modulus E_3 is the most significant predictor variable for all FWD parameters (response variables).

The peak deflection under the load (D_{Peak} at 0-inch offset) is the most affected by the thickness of the surface layer and the subgrade modulus; an increase h_1 and E_3 decreases the deflection under the load [negative (–) effect]. The outer deflections, on the other hand, are the most affected by the subgrade modulus and the Rayleigh damping coefficient β_R.

The time of the peak deflection (T_{DPeak}) is the most affected by h_1, γ, E_3, h_{SL}, and β_R. For example, an increase in h_1 and β_R increases the time of the peak deflection under the load (i.e., delays the peak), whereas an increase in γ and E_3 decreases the time of the peak deflection. An increase in γ produces a lower AC modulus.

The minimum deflection (D_{minL}) to the left of the peak is common for the outer sensors when testing relatively thin and soft pavement structures, and it is the most affected by h_1, h_2, and E_3. The minimum deflection (D_{minR}) to the right of the peak is common for pavement structures with shallow stiff layers, and it is the most affected by E_3, h_{SL}, and β_R.

The deflection pulse durations at 25, 50, and 75 percent of the peak deflection (Dur_{25}, Dur_{50}, and Dur_{75}, respectively) are the most affected by h_1, E_3, h_{SL}, and β_R.

The times to the left of the peak corresponding to 25, 50, and 75 percent of the peak deflection (T_{25L}, T_{50L}, and T_{75L}, respectively) are the most affected by h_1 and E_3, whereas the times to the right of the peak (T_{25R}, T_{50R}, and T_{75R}) are the most affected by E_3 and β_R.

The velocities (V_{minL}, V_{Peak}, and V_{minR}) are affected similar to their deflection counterparts (D_{minL}, D_{Peak}, and D_{minR}).

Considering all contributions for the various response variables, the subgrade modulus predominantly controls the FWD parameters with a 70 percent contribution, as illustrated in Figure 30, followed by the Rayleigh damping coefficient β_R, the AC thickness, and the depth to the stiff layer, when present.

The contribution of the AC master curve variables is minimal relative to the major contributors. As a result, the backcalculation of the AC master curve is increasingly challenging.

Finally, the sum of the contributions (last column in Table 7) is used to determine the FWD parameters with the largest contributions for use in dynamic backcalculation. The following 12 FWD parameters are shown to have the most significant effect:

- D_{minL}, D_{Peak}, T_{Peak}, D_{minR}
- T_{50L}, T_{50R}, T_{75L}, T_{75R}
- V_{minL}, V_{Peak}, T_{VPeak}, V_{minR}

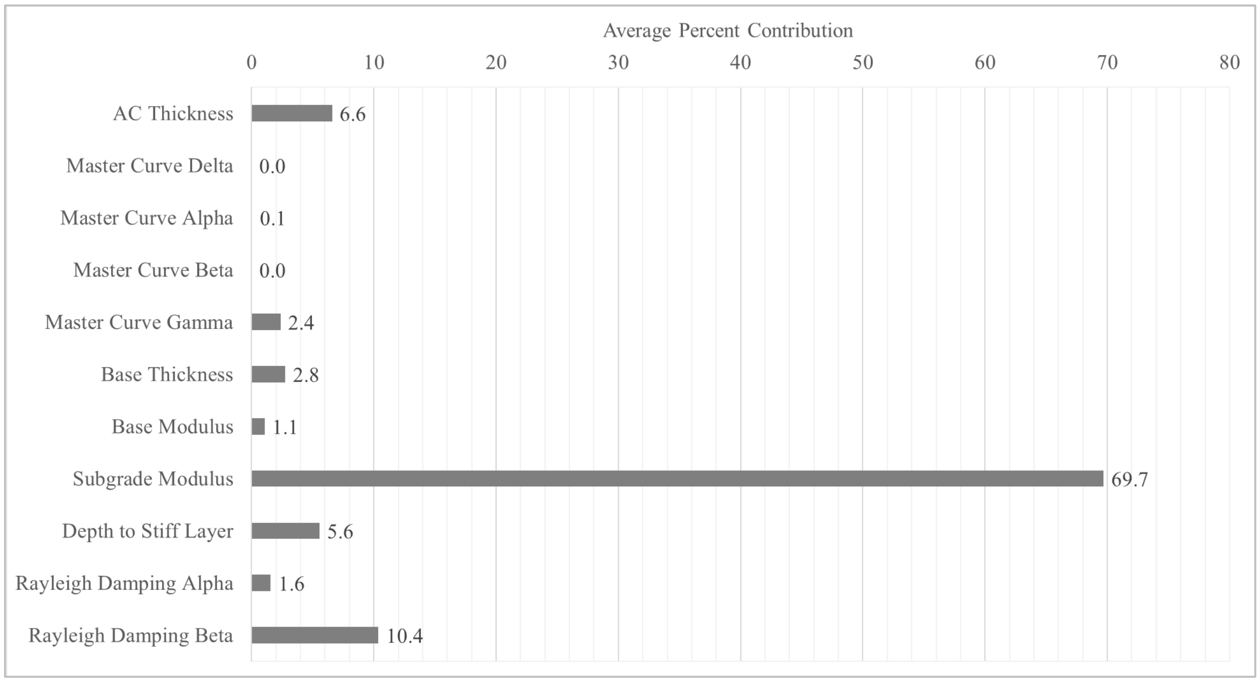


Figure 30. Overall Contribution of the Predictor Variables (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

4.5 DYNAMIC BACKCALCULATION OF FAA SECTION

The *PULSE* 2020 application was used for the dynamic backcalculation of a flexible pavement structure built at the FAA NAPTF. The NAPTF is a fully enclosed instrumented test track 900 feet long by 60 feet wide, where flexible and rigid sections are built, instrumented, and trafficked to evaluate different pavement technologies and to advance airport pavement design and evaluation methods.

The pavement structures of CC-9 were built in December 2019 and consist of 10 flexible pavement test items designed to address multiple objectives. Four test items within low-strength subgrade flexible pavement with stabilized base (LFS)-1 and LFS-2 are designed to analyze the effect of P-403- and P-209-layer thicknesses on the fatigue life. Four test items within LFC-3 and LFC-4 are comparable pavement structures designed to analyze the effect of geosynthetic materials and cement-treated permeable base material. Two test items within LFC-5 are designed to analyze the strain criteria for flexible pavement allowable overload.

The southern LFS-2 section (LFS-2S) was selected for this study (Figure 31), and the structure consisted of 4-inch P-401, 5-inch P-403, and 30-inch P-209 over a Dupont clay subgrade (P-152). P-401 and P-403 refer to the AC layers, P-209 refers to the crushed aggregate base layer, and P-152 refers to the subgrade layer as defined in FAA Advisory Circular (FAA AC) 150/5370-10H.

FWD testing is performed periodically to evaluate the uniformity of the sections before trafficking and to quantify the damage during trafficking. Pre-traffic FWD testing was performed on December 30, 2019, and January 16, 2020, and the data from the December testing were used for the analysis.

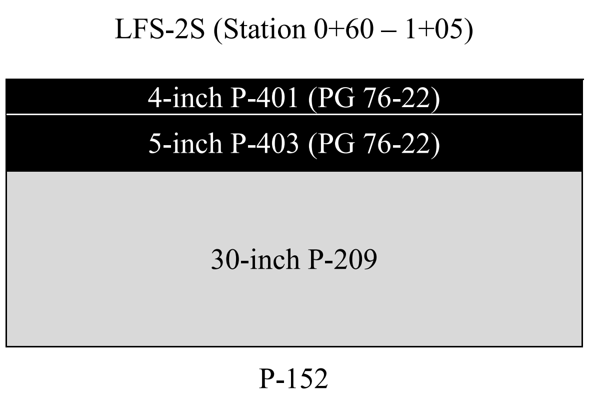


Figure 31. Pavement Structure for CC-9 LFS-2S (left) and FWD at NAPTF (right) (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

A plot of the deflection basins at three load levels, ranging from 13 kips to 37 kips, is shown in Figure 32. Overall, the FWD-measured surface deflections were relatively small due to thick and stiff pavement structure at the tested temperature; where a mid-depth AC temperature of 50 °F, a surface temperature of 52 °F, and an air temperature of 54 °F were recorded at the time of testing (11:24 AM).

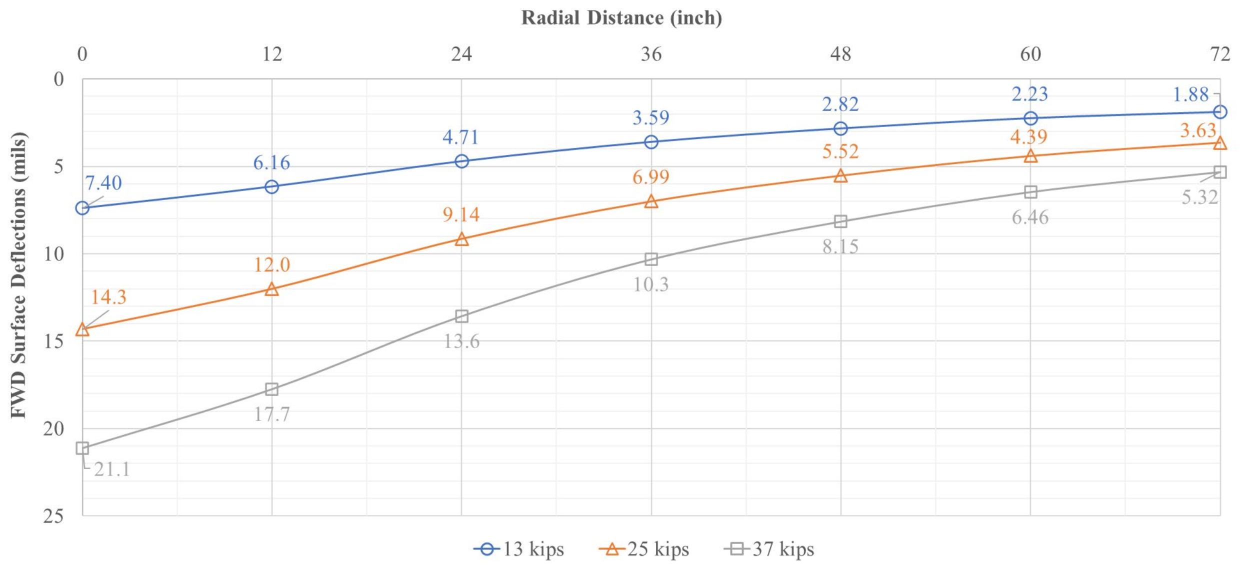


Figure 32. Deflection Basins for CC-9 LFS-2S at Three Load Levels on December 30, 2019 (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

Figure 33 shows the surface moduli plot as determined using the Boussinesq equations, where the calculation is based on the FWD load and the corresponding surface deflections. The surface moduli at the center of the FWD load plate were calculated using the Boussinesq distributed load equation, whereas the surface moduli at any radial distance from the center of the load plate are calculated using the Boussinesq point load equation. The surface moduli provide equivalent stiffness assuming the pavement is composed of a semi-infinite half-space.

The FWD loads, for the various drops, were proportional to the surface deflections resulting in almost identical surface moduli. This observation also indicates that the pavement materials mainly behave as stress-independent materials (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020).

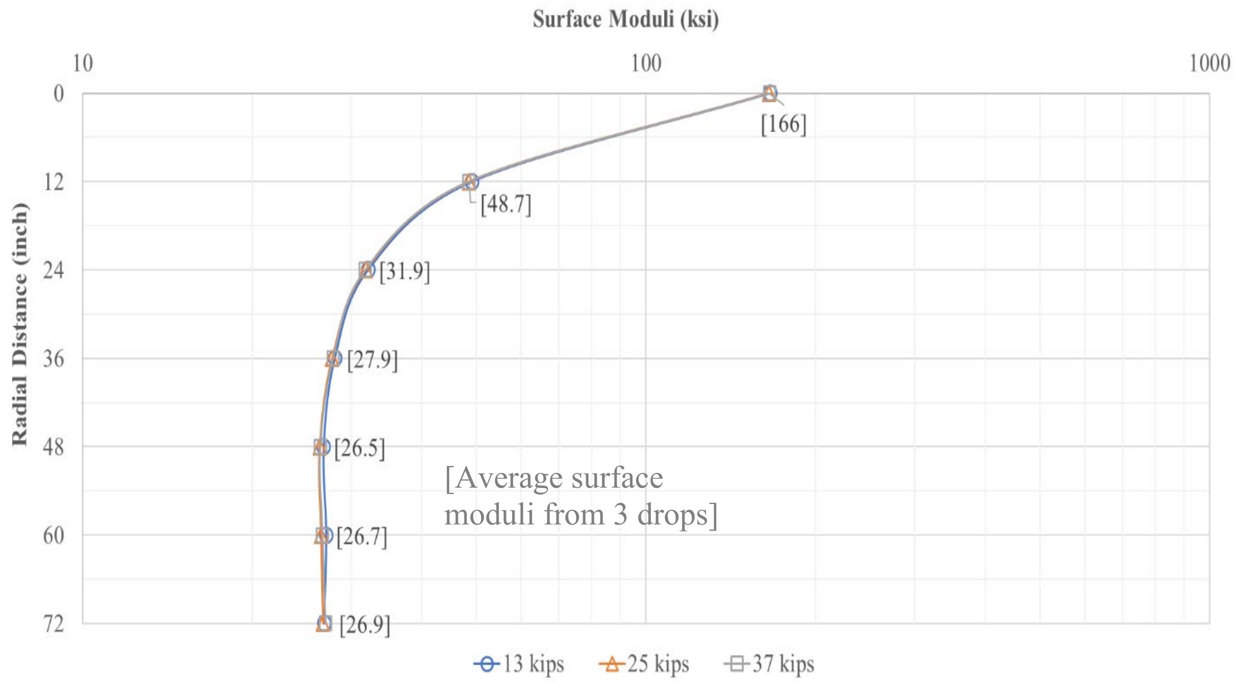


Figure 33. Surface Moduli for CC-9 LFS-2S at Three Load Levels on December 30, 2019 (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

The dynamic backcalculation was performed for the three load levels using the recommended parameters from the previous section, except for D_{minL} and V_{minL} because those parameters are not present for this thick and stiff structure.

The backcalculated layer variables (moduli) are presented in Table 8, where the AC moduli are reported at the tested temperature of 50 °F and at a frequency of 17 Hz, where the 17 Hz is considered to be the most dominant FWD frequency (Sebaaly et al., 1985 and 1986; Kim, Xu, & Kim, 2000; Chatti & Lei 2012; Bazi & Bou Assi, 2022; Fu et al., 2020).

The Rayleigh damping coefficients, β_R, are determined as 0.0034 and 0.0023 for the P-209 and P-152 layers, respectively, and are kept constant for the various drop levels to have a one-to-one comparison of the layer moduli.

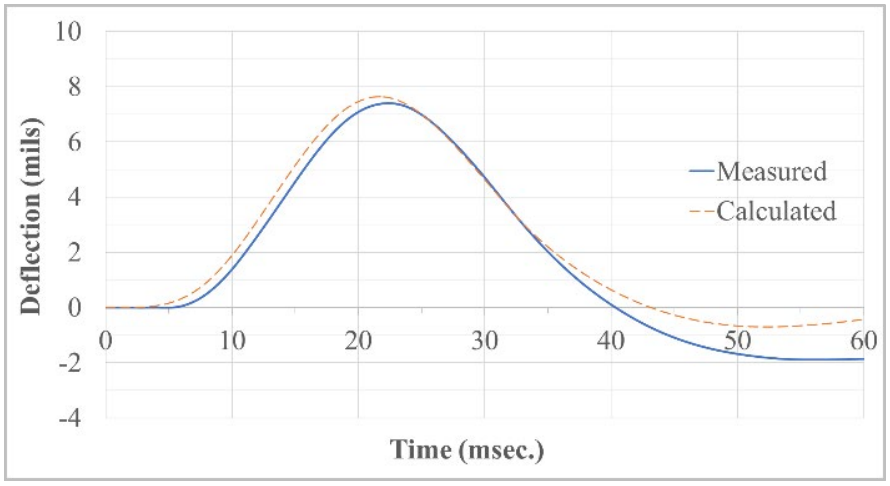
Table 8. Backcalculated Layer Moduli

Layer		CC-9 LFS-2S Station 0+85 Offset +15S		
		Load Level = 13 kips	**Load Level = 25 kips**	**Load Level = 37 kips**
AC [P-401/P-403][1]		1,170 ksi at 17 Hz	1,130 ksi at 17 Hz	1,100 ksi at 17 Hz
Crushed Aggregate Base [P-209]		46.8 ksi	42.8 ksi	41.6 ksi
Subgrade [P-152]		13.9 ksi	13.1 ksi	12.7 ksi
RMSRE	D_{Peak} only	3.9%	3.2%	3.1%
	All parameters	2.9%	3.2%	3.6%

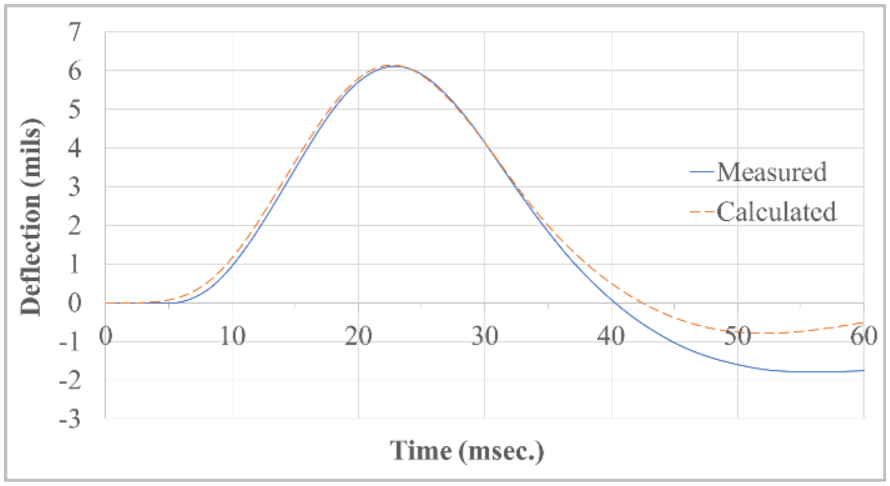
[1]The P-401/P-403 asphalt surface and asphalt base layers were combined during backcalculation.

The root-mean-square relative errors (RMSREs) between the measured and calculated parameters, as reported in Table 8, are acceptable for the three load levels by considering the peak deflections (D_{Peak} only) and by considering all FWD parameters used in the backcalculation. The calculated and measured time histories at the first load level of 13 kips are shown in Figure 34, where the fit is adequate. It is important to note that several parameters are being fitted by varying the pavement variables, and this process does not simply consider the peak deflections (D_{Peak}) used in static backcalculation or the peak deflections and time lag used in traditional dynamic backcalculation.

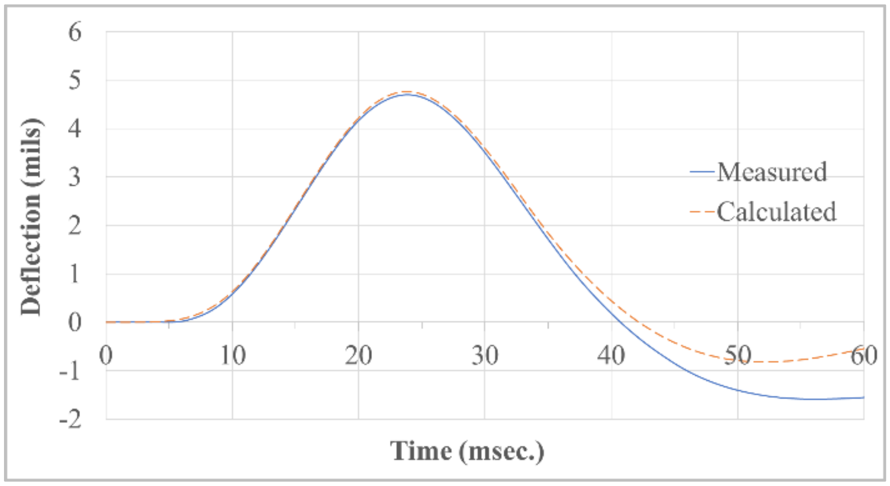
(a) 0-inch



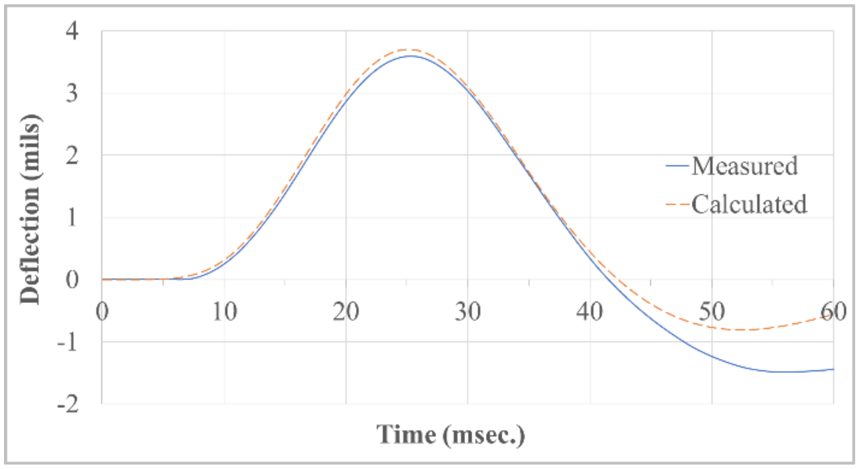
(b) 12-inch



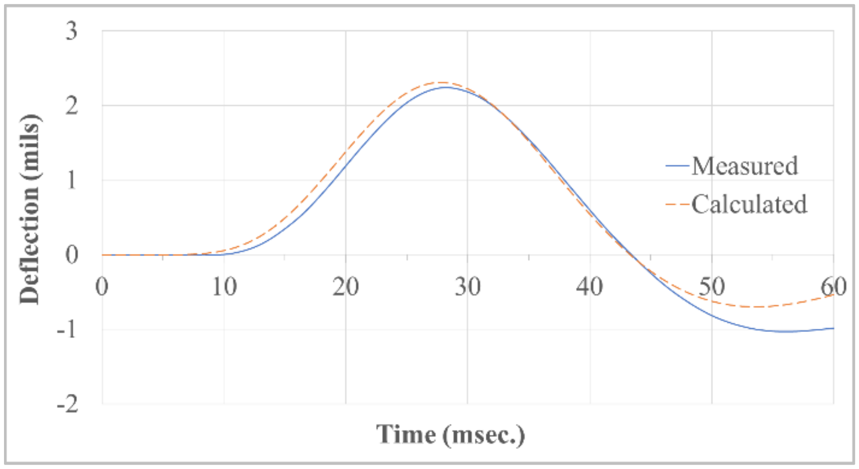
(c) 24-inch



(d) 36-inch



(e) 60-inch



(f) 72-inch

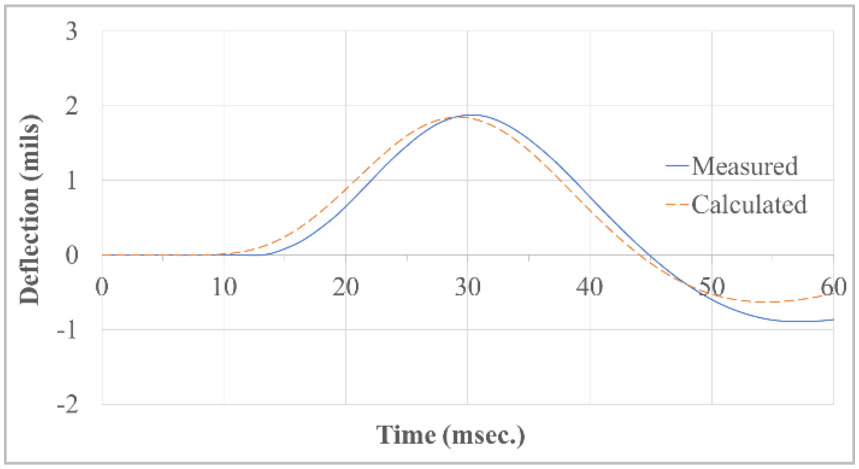


Figure 34. Measured vs Calculated FWD Deflections at 13 kips for December 30, 2019 (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

A comparison of the crushed aggregate base (P-209)- and subgrade (P-152)-layer moduli at the various load levels in Table 8 shows that the unbound layers are slightly stress-dependent, where the moduli decrease for an increase in load level. The moduli stress-softening is about 10 percent for a 185 percent increase in load level, and this behavior can only be captured by analyzing the FWD data at different load levels.

Fine-grained materials (e.g., P-152) typically exhibit stress-softening behavior, which was confirmed from dynamic backcalculation for the subgrade layer. Conversely, coarse-grained materials (e.g., P-209) typically exhibit stress-stiffening behavior based on laboratory testing, but this observation is contrary to the results obtained from dynamic backcalculation. Overall, coarse- and fine-grained materials exhibited a mildly stress-softening behavior based on dynamic backcalculation. This observation was also reported in previous research for an LTPP section (Bazi, Saboundjian, Bou Assi, & Diab, 2020).

In the same LTPP study, it was shown that the confinement effect resulting from the stiffness of the layers above an unbound layer has a major effect on that layer. The confinement effect, which is more pronounced than the mild stress-softening effect, was not studied for the FAA section since the temperatures during the two FWD testing periods (December 2019 and January 2020) were identical, resulting in similar AC moduli and confinement.

Triaxial resilient modulus testing was performed by the FAA on the P-152 subgrade material at different combinations of confining and cyclic stresses in accordance with American Association of State Highway and Transportation Officials (AASHTO) T 307-99 (2021). Figure 35 shows a plot of the subgrade resilient moduli vs the cyclic (deviatoric) stresses at three confining stresses (S3) of 6 psi, 4 psi, and 2 psi. A stress-softening behavior was observed, which is expected for a fine-grained material.

The figure also shows the backcalculated subgrade layer moduli at the three load levels of 13 kips, 25 kips, and 37 kips. The moduli are plotted against the calculated maximum vertical stresses due to the FWD loading on top of the subgrade layer, as obtained from the FE model. The mild nonlinearity of the backcalculated subgrade layer moduli is visible, and the moduli match, to a certain extent, the triaxial resilient moduli at a confining stress of 2 psi.

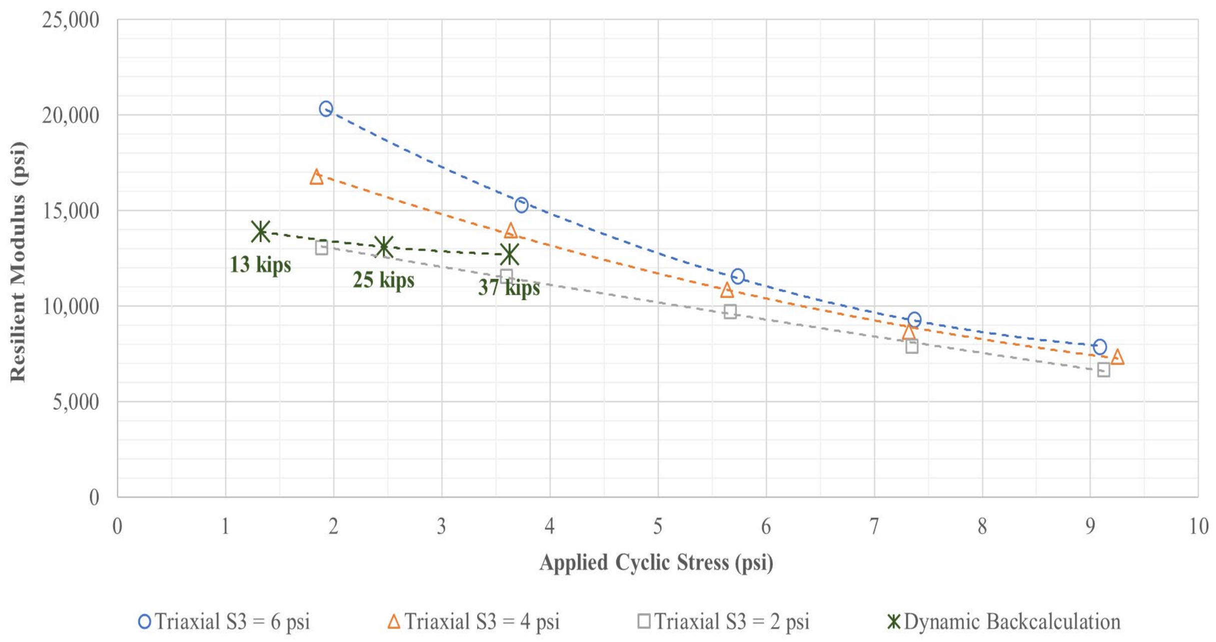


Figure 35. P-152 Moduli from Triaxial Testing and Dynamic Backcalculation
(Bazi, Saboundjian, Bou Assi, & Diab,2020)

Finally, the backcalculation was also performed using state-of-the-practice software based on static analysis. Unreliable subgrade moduli were obtained for the various drops using the static analysis, where the subgrade moduli were higher by a factor of 2 to 2.5 when compared to the dynamic backcalculation results. Such variation is expected for rigid pavements or thick and stiff flexible pavements that are common for airport pavement structures (Bazi, Gagnon, Sebaaly, & Ullidtz, 2020). It is important to note that the subgrade moduli obtained from static backcalculation are almost equal to the surface moduli for the outer sensors, as depicted by Figure 33.

5. OPTIMIZATION TECHNIQUE

Optimization is the act of obtaining the best result under given circumstances. An optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within allowed ranges to compute the value of the function.

Optimization techniques are used in the backcalculation process to estimate the pavement layers variables that would minimize the error between the measured and calculated FWD deflection time histories, or, more specifically, between time histories parameters. In this application, the layers variables are denoted as V, and the evaluation parameters are denoted as P. For a given load D, modeling the FWD testing is formulated as a mapping $M(D, V) \rightarrow P$. If the measured FWD data are denoted as $\hat{P}$, the modeling error is computed by RMSRE (Khetan & Karnin, 2020) as:

$$L(P, \hat{P}) = \sqrt{\frac{1}{n}\sum_{1}^{n}\left(\frac{P_i - \hat{P}_i}{P_i}\right)^2} \tag{83}$$

Thus, the optimization problem of the backcalculation process is formulated as:

$$\underset{V}{\operatorname{argmin}}\, L\big(M(D,V),\hat{P}\big) \tag{84}$$

5.1 FULLY AUTOMATED AND GENERAL OPTIMIZATION FRAMEWORK

The research team developed a fully automated and general optimization framework to bridge the *PULSE* application and various optimization methods. The framework has three main components: (1) an input generator, (2) a middleware integrated with *PULSE*, and (3) a cross-platform plugin interface. This optimization framework enables automated generation of input data, easy incorporation of any optimization algorithms in implementations (supporting different programming languages), and construction of the entire workflow in a fully automated manner. Figure 36 shows an overview of the fully automated and general optimization framework developed in this research effort.

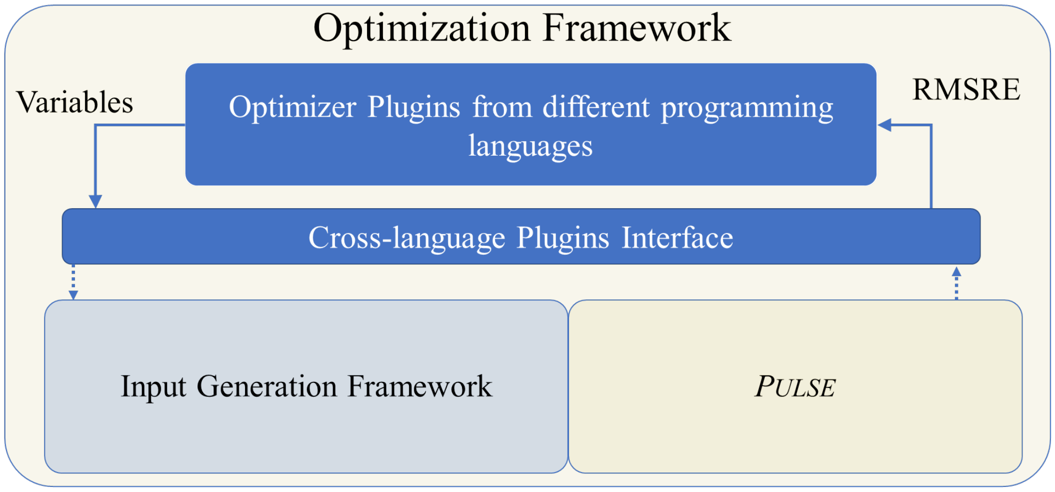

Optimization Framework
Variables
Optimizer Plugins from different programming languages
RMSRE
Cross-language Plugins Interface
Input Generation Framework
PULSE

Figure 36. Overview of the Optimization Framework

5.1.1 Motivation of the Optimization Framework

The *PULSE* application, programmed in C#, can only handle a specific optimization method implemented in the C# programming language. This study evaluated the implementation of different optimization methods. However, some optimization methods are difficult to implement in the C# programming language. For example, the machine-learning-based optimizer, Reinforcement Learning, is one of the approaches the researchers plan to develop. Yet its implementation relies heavily on deep-learning frameworks, such as TensorFlow (Abadi et al., 2016) or PyTorch (Paszke et al., 2019), which are programmed only for Python and C++ languages. Given these deep-learning frameworks have huge codebases (developed by thousands of full-time engineers in several years) and are optimized by numerous hardware-level techniques, it is not feasible to import them into C#. Therefore, it is critical to have a framework that can accommodate optimizers in different programming languages and become compatible with *PULSE*.

In addition, it is important to automate the optimization workflow. As shown in *PULSE* 2020 flowchart (Figure 25), several steps in the workflow need significant amounts of manual effort, which can be time-consuming. Examples include (1) creating structure and mesh using Gmsh (refer to Section 3.1 for more details), and (2) preparing the input files for FE solver. These manual tasks can become a bottleneck as different optimization methods are evaluated and many experiments need to be performed, which could take a significant amount of time to complete. Fully automating the workflow would significantly speed up the development and evaluation process of different optimization methods.

5.1.2 Input Generation Framework

To develop and evaluate the optimization methods, several input files need to be generated and prepared for the FE solver (mainly *PULSE*_FE, or ABAQUS for verification). The backbones of the pavement structures (e.g., points, lines, surface) are first calculated and modeled with the aid of Microsoft® Excel®. Then the information is manually imported into Gmsh (Geuzaine & Remacle, 2009) to mesh the pavement structure. To prepare the final input files, the users need to manually analyze the generated meshing and retrieve the following information for the FE solver: the starting/ending indexes for each element set, nodes on the far boundary, nodes on the axis of symmetry, evaluation nodes, and elements on the far boundary. Such manual efforts usually take at least 10 minutes even for experienced users, which is time-consuming and labor intensive to complete a large number of experiments.

To relieve the burden on the users and speed up the process of generating and preparing input files, an input generation framework that can automate this process and reduce the time to less than 10 seconds was developed as part of this study. Figure 37 demonstrates an overview comparison of the previous manual input framework and the new fully automated input generation framework. The following sections present details on each step.

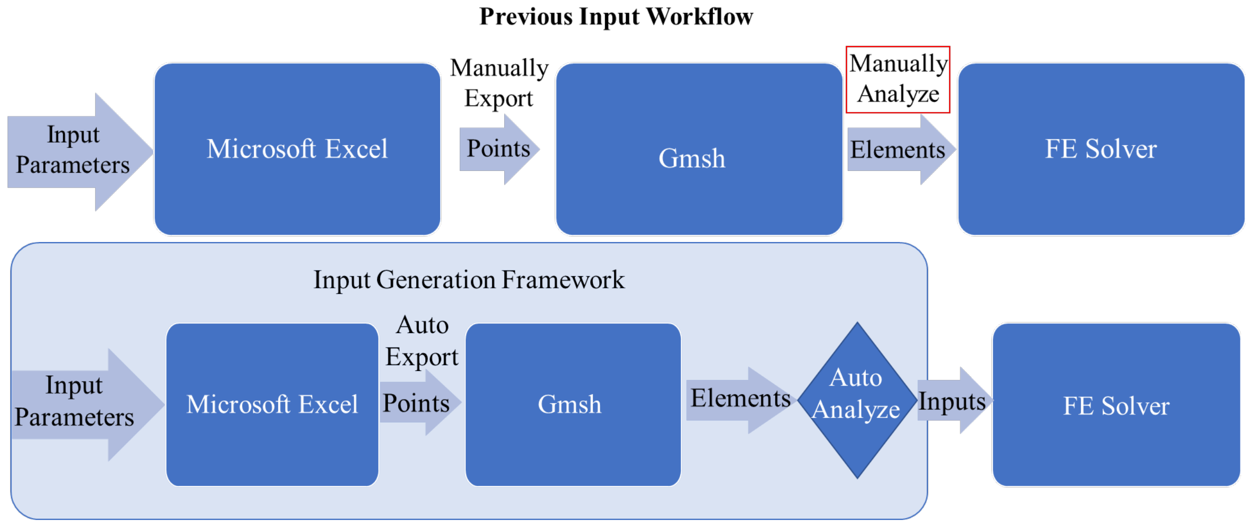


Figure 37. Overview Comparison of the Previous Manual Input Framework and the New, Fully Automated Input Generation Framework (Bazi, Saboundjian, Bou Assi, & Diab, 2020)

5.1.2.1 Pavement Structure Model

In the previous workflow, Excel was manually invoked to prepare the pavement structure backbones (e.g., points, lines, and surface) to be meshed. Automating this step is quite challenging given the large number of built-in equations and calculations, and Excel is well known to be automation unfriendly. To solve this challenge, Excel was programmed through the component object model (COM) provided by Windows. The COM is a platform-independent, distributed, object-oriented system for creating interactive binary software components. COM is the base of Microsoft's Object Linking and Embedding (OLE) (compound documents) and ActiveX (Internet-enabled components). COM objects can be created with a variety of programming languages. Specifically, the input-generation framework binds Excel by its ProgID as a COM object. By using COM, the input-generation framework has full control over Excel. In this manner, the input-generation framework can fill in the variables to the Excel file, and then Excel can perform the calculations to get the final output for developing the mesh. Given the calculation part is exactly the same, the results will be consistent with those from the manual process.

5.1.2.2 Developing the Mesh

To eliminate the manual operation on Gmsh, a Python wrapper of the Gmsh Software Development Kit (SDK) was developed. Specifically, the input-generation framework reads the pavement structure (e.g., points, lines, and surface) from the Excel output in previous step and composes a *.geo file as the input to Gmsh. Then, the input-generation framework calls the Gmsh application programming interface to synchronize its internal CAD representation with the Gmsh model, which creates the relevant Gmsh data structures for deriving the 2D meshing. The meshing using the new generation framework was compared to and checked against the manual generation meshing, which showed excellent agreement.

5.1.2.3 Analysis

The input-generation framework performs a series of analyses that resemble the manual efforts once the generated nodes and elements are acquired from Gmsh. Given the time-consuming nature of these analyses, automating these steps can save much of the manual effort and make the whole process much faster. Figure 38 shows the necessary analysis steps to compose the final input file. A few key steps are highlighted here: (1) locate the start and end elements for each surface to compose the element sets (each layer), (2) calculate distance from origin for each node to filter out the nodes on the far boundary, (3) find out nodes on the y-axis but not on far boundary, and (4) find out nodes and elements on the pavement surface where the simulated FWD load is applied.

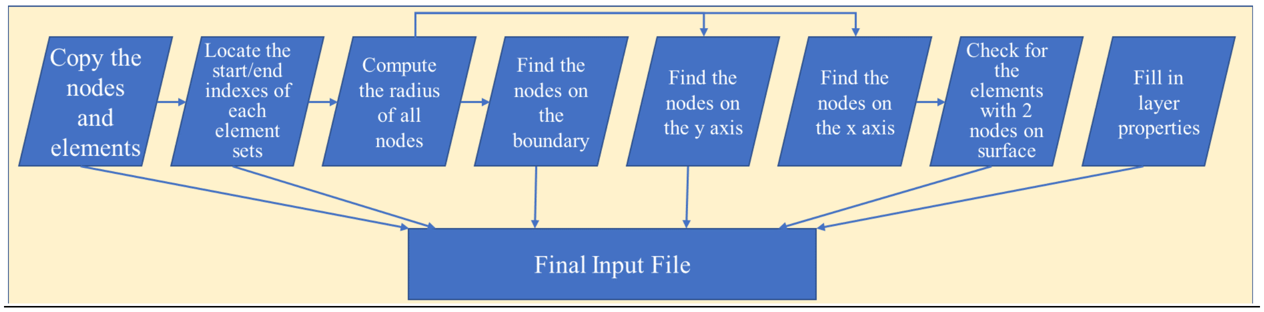

Copy the nodes and elements
Locate the start/end indexes of each element sets
Compute the radius of all nodes
Find the nodes on the boundary
Find the nodes on the y axis
Find the nodes on the x axis
Check for the elements with 2 nodes on surface
Fill in layer properties
Final Input File

Figure 38. Necessary Analysis Steps to Compose the Final Input File

5.1.2.4 Evaluation

As a key component of the optimization framework, it is important for the input-generation framework to complete the generation workflow in a fast manner. The performance of the input-generation framework was evaluated on a workstation with Intel Core i7-8700 Processor and 16-GB memory. A virtual machine with Windows 10 version 2004 was used. The evaluation included 30 independent runs of using the input generation framework to compose the ABAQUS-style input for a three-layer pavement structure. The clock running time for each run is shown in Figure 39. As shown in the figure, all 30 runs finished within 3 seconds, which is a significant speedup compared with the manual processing that usually takes about 10 minutes. Note that the time shown here also includes the time used in external applications such as Excel and Gmsh.

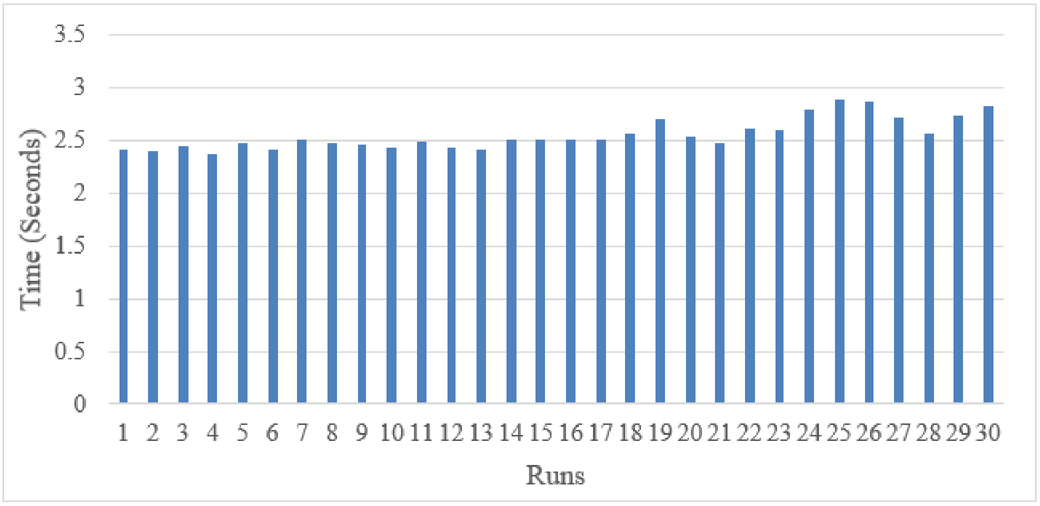


Figure 39. Running Time of Input Generation Framework for 30 Runs to Generate Input for a Three-Layer Pavement Structure

5.1.3 *PULSE* Application Wrapper

The *PULSE* application wrapper was developed to automatically parse the configurations for *PULSE*_FE from ABAQUS-style inputs and to bridge the output from *PULSE*_FE to *PULSE*_Analyzer. The *PULSE*_FE is a powerful and better replacement for ABAQUS. It is faster and easier to use. The downside is that it requires users to manually read some parameters from input files, like the ABAQUS format, and feed them to *PULSE*_FE (refer to Section 3 for more details). The developed *PULSE* application wrapper can automate this process to greatly improve user experience by offering a one-stop solution. It automatically converts the ABAQUS inputs to the final calculated parameters and hides all details from users. Table 9 lists the *PULSE*_FE parameters that can be automatically parsed from ABAQUS inputs by the developed *PULSE* application wrapper.

Table 9. Parameters that can be Automatically Parsed from ABAQUS Input Files

Parameter Name	Data Type	Explanation
layerElements	Dictionary	The index of all elements of a certain layer
NsetFarBoundary	List	The index of nodes on the far boundary
NsetSymmetry	List	The index of nodes on the axis of symmetry (Y axis)
ElsetLoad	List	The index of elements on the X axis
surfaceNumber	String	Surface number to indicate where pressure is applied
surfacePressure	Double	The pressure applied to the surface
ampIN	Dictionary	FWD amplitude data

This list covers most of the parameters that are available in ABAQUS input files. By automatically parsing these parameters, considerable amount of manual parsing efforts is avoided. Combining with the aforementioned input generation framework, these two techniques together make the entire simulation and analysis workflow fully automated. Specifically, with a set of input variables, the framework derives the corresponding calculated parameters after the finite element modelling without manual assistance.

5.1.4 Cross-Language Plugin Interface

The core functionality of the optimization framework lies in its ability to work with optimizers from different programming languages. To achieve this goal, a cross-language plugin interface was developed with three key features: (1) a cross-language protocol that allows the exchange of arbitrary basic data types and even complex or custom data structures such as dictionary, objects, and class; (2) a flexible scalar/vector interface that was designed to be compatible with optimizers regardless of whether they expect vector or scalar output, and (3) a client-server design that allows the optimizer to be placed on remote server or even cloud servers. Each feature is briefly described in the following sections.

5.1.4.1 Data Structure Exchange Protocol

Passing data from one programming language to another is challenging as they are represented and stored in quite different ways. Thus, it is important to have an intermediate layer that can connect different programming languages. The developed plugin interface employs the JSON (JavaScript Object Notation) protocol, which is a lightweight data-interchange format that is broadly supported by multiple programming languages including C#, C++, Java, Python (Pezoa, et al., 2016). JSON uses a text-based approach to store data, which makes the programming language irrelevant and the information readable for human audiences. The basic data types in

JSON are string, number, bool, and null. The fundamental data structures for JSON are (1) a collection of name/value pairs, which in other programming languages would be interpreted as object, record, struct, dictionary, hash table, keyed list, or associative array; and (2) an ordered list of values that are commonly interpreted as array, vector, list, or sequence. Note that the data structures can be nested, which enables JSON to store and represent complex data structure or classes.

5.1.4.2 Flexible Scalar/Vector Interface

Different optimizers are designed for different types of optimization problems. Some optimizers can correctly handle the case where the optimization target is a vector (a series of values), whereas a more common case is when the target is a scalar (a single value). To accommodate different optimizers, the interface is implemented to have two sets of outputs: a vector of calculated parameters and a scalar RMSRE calculated from difference between the calculated parameters and the measured parameters from FWD surface deflections. Thus, for vector-based optimizers like Newton's method, they have all the needed details, which leads to fast convergence. For scalar-based optimizers like Bayesian and Reinforcement Learning, they use the RMSRE scalar, so they can still work well.

5.1.4.3 Client-Server Design

In the ideal case, the *PULSE* and optimizer would run side-by-side on the same device for a seamless data exchange and latency-free feedback. However, sometimes the optimization problem can be complex, and its resource demands can go beyond the capacity of a single machine. The plugin interface provides extra flexibility for this scenario by using a client-server design that can have *PULSE* and the optimizer run on different machines. The text-based data exchange protocol JSON ensures that the data exchange can stay untouched even if the sender and receiver are from different machines. This enables the user to offload the computation of optimizer to a remote server or even cloud to improve the optimization speed.

5.1.5 Summary

The complete overview of the optimization framework is shown in Figure 40. The overall workflow of the developed optimization framework is summarized as follows:

- ***Step 1.*** *PULSE* gives an initial start point (a set of variables) x_0 and passes it to the optimizer through the cross-language plugin interface.
- ***Step 2.*** The optimizer decides the next point to try x_{t+1}, and passes it to input generation framework through the cross-language plugin interface.
- ***Step 3.*** Input generation framework performs the analysis to prepare the ready-to-use inputs to *PULSE* application wrapper.
- ***Step 4.*** *PULSE* application wrapper performs the FE modeling and analyzes the results using *PULSE* Analyzer to get the final calculated parameters and pass it to optimizer through the cross-language plugin interface.
- ***Step 5.*** Optimizer checks the calculated parameters to see whether the optimization process has converged. If not, the process goes back to step 2.

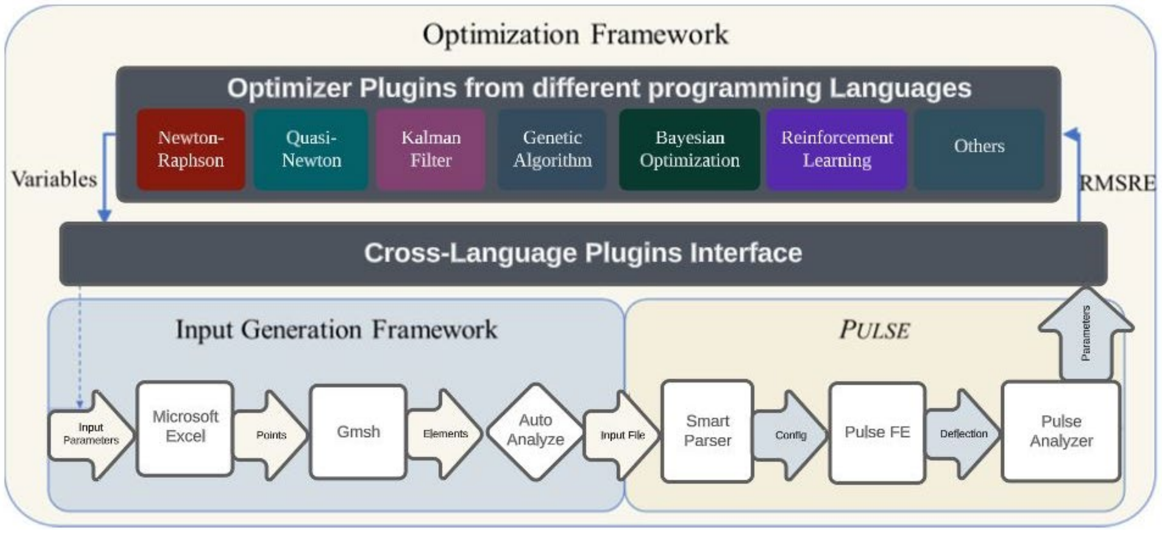


Figure 40. Complete View of All Components of Optimization Framework.

5.2 OPTIMIZATION PROBLEM FORMULATION

This section presents two problem formulations that correspond to two different types of optimization strategies. In both formulations, the choice of parameters is consistent with the previous analysis in Section 4.

5.2.1 Vector-Based Problem Formulation

For the optimizers that can work with vector outputs, the optimization target is to have the calculated parameters approach the measured parameters as close as possible. Thus, the problem is formulated as:

$$\overrightarrow{\boldsymbol{P_c}} = f(\overrightarrow{\boldsymbol{V}}) \tag{85}$$

$$Minimize\ ||\overrightarrow{\boldsymbol{P_c}} - \overrightarrow{\boldsymbol{P_m}}\,||\ \ w.r.t.\ \ \overrightarrow{\boldsymbol{V}} \in \boldsymbol{R} \tag{86}$$

where f is the abstraction of the process of input generation, FE modeling, and parameter analysis. $\overrightarrow{\boldsymbol{V}}$ is the variables vector, $\overrightarrow{\boldsymbol{P_c}}$ is the calculated parameters vector, and $\overrightarrow{\boldsymbol{P_m}}$ is the measured parameters vector. Due to differences in scales of the parameters, the optimizers potentially focus more on the large-scale variables. Fortunately, the proposed Newton's method is based on individual gradients, which are robust to the variables' scales. This problem is solved by using normalization techniques.

5.2.2 Scalar-Based Problem Formulation

A more general formulation is where the optimizers expect the output to be a scalar so that the output vector can be wrapped by RMSRE. The optimization target is thus to minimize the RMSRE scalar between the calculated parameters and the measured parameters. The problem formulation is defined as follows. Note that the optimization target is the RMSRE value, which is a scalar.

$$\overrightarrow{\boldsymbol{P_c}} = f(\vec{\boldsymbol{V}}) \tag{87}$$

$$R = RMSRE\left(\overrightarrow{\boldsymbol{P_c}}, \overrightarrow{\boldsymbol{P_m}}\right) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{P_c^i - P_m^i}{P_m^i}\right)^2} \tag{88}$$

$$Minimize \quad R \quad w.r.t.\ \vec{\boldsymbol{V}} \in \boldsymbol{R} \tag{89}$$

5.2.3 Comparison of the Two Formulations

Each of the aforementioned formulations has advantages and disadvantages. For the vector-based problem formulation, the key advantage is that it reveals more details and avoids the ill-condition issue. For example, suppose at point X the measured parameters are *a*, *b*, and *c*; then the optimizer tries a nearby point Y, which gives estimated parameters of $1.1a$, $0.9b$, and *c.* For the vector-based problem formulation, the change from X to Y leads parameter *a* to increase and *b* to decrease. However, for the scalar-based problem formulation the two calculated parameters at points X and Y might give similar RMSRE values, which does not give explicit clues to the optimizer about the possible impact of the change from X to Y. Such confusion could mislead the optimizer to make wrong optimization decisions and thus achieve suboptimal results. A key advantage of the scalar-based problem formulation is its generalization. Almost all optimizers can work with the scalar-based problem formulation, yet only a few can handle the vector-based problem formulation. Especially for the learning-based optimizers, they are designed to work on scalar outputs. Considering the reasons stated, both problem formulations are implemented in this study.

5.3 TRADITIONAL OPTIMIZERS—NEWTON-RAPHSON METHOD

With the help of the developed optimization framework, several different optimizers from the family of Newton-Raphson are implemented and evaluated (Ypma, 1995). The Newton-Raphson method is a classic numerical/mathematical-based optimization method.

The Newton-Raphson method, also called Newton's method, is an iterative algorithm that gradually approaches the root of target function by computing its derivative. Based on whether the target function is the optimization target itself or its derivative, the Newton's method is categorized into first-order Newton's method (also known as the root-finding algorithm) and second-order Newton's method (the optimization algorithm). Note that the first-order Newton's method is the default optimization approach in *PULSE.*

5.3.1 First-Order Newton's Method

The intuition of the first-order Newton's method is straightforward. For example, suppose there is a point x_n on function f. The x-intercept of its tangent line is likely to be closer to the root of f (see Figure 41 for an example). Thus, by iteratively doing this, one can get closer to the root that solves the problem.

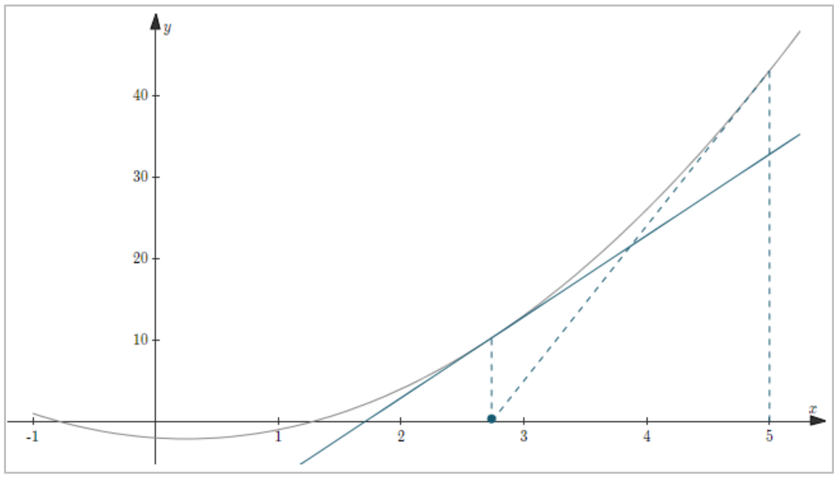


Figure 41. Example of How Newton's Method Approaches the Root of a Quadratic Function from Initial Point x = 5

In a former way, Taylor's expansion is used on function $f(x)$ at x_n :

$$y = f(x_n) + f'(x_n)(x - x_n) + \frac{1}{2}f'(x_n)(x - x_n)^2 + \cdots \tag{90}$$

Ignoring the higher-order items in Equation 90 gives the tangent line of x_n:

$$y = f'(x_n)(x - x_n) + f(x_n) \tag{91}$$

The goal is to get the x-intercept (the x that makes $y = 0$). By denoting the x-intercept as x_{n+1}, the y function becomes:

$$y = f'(x_n)(x_{n+1} - x_n) + f(x_n) = 0 \tag{92}$$

Thus, the updated equation for first-order Newton's method is:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{93}$$

In the subject problem, for the scalar-based problem formulation, the updated equation is:

$$\boldsymbol{V_{n+1}} = \boldsymbol{V_n} - \frac{f(\boldsymbol{V_n})}{f'(\boldsymbol{V_n})} \tag{94}$$

where $\boldsymbol{V_n}$ is the current variables and $\boldsymbol{V_{n+1}}$ is the variables to try next. Given $R = f(\boldsymbol{V_n})$ is a scalar, the corresponding derivative $f'(\boldsymbol{V_n})$ is a vector commonly called gradient.

For the vector-based problem formulation, the multivariable equation is:

$$\boldsymbol{V_{n+1}} = \boldsymbol{V_n} - \boldsymbol{J}^{-1}(\boldsymbol{V_n})\boldsymbol{f}(\boldsymbol{V_n}) \tag{95}$$

It is different from the scalar-based equation as the $\boldsymbol{f(V_n)}$ is a vector that makes its derivative $\boldsymbol{J(V_n)}$ a matrix called the Jacobian matrix. To adapt to matrix operations, the division becomes a left-multiplying of the inverse of Jacobian matrix and the corresponding vector output.

For a multivariate Newton optimization problem, the transition from a scalar variable, x_n, to a vector, V_n , necessitates a corresponding transformation of the scalar derivative, $f'(x_n)$, into the Jacobian matrix, $\boldsymbol{J(V_n)}$. Consequently, leveraging matrix operations become feasible, wherein the inversion of the Jacobian matrix, $\boldsymbol{J^{-1}(V_n)}$, followed by its multiplication with the vector, $\boldsymbol{f(V_n)}$, facilitates the derivation of the succeeding optimal solution, V_{n+1}, as shown in equation 95. Multivariate optimization, though derived from single-variable optimization, is distinguished by its transition from scalar output and scalar derivative to vector output and matrix derivatives.

The advantage of the first-order Newton's method is its simplicity and the ability to work with vector-based problem formulation. The drawbacks include the dependence on derivative, which is often not available or hard to get for optimization problems (including the subject problem), and the sensitivity to the choice of initial point. In addition, the first-order Newton's method does not provide convergence guarantee.

5.3.2 Second-Order Newton's Method

The second-order Newton's method focuses on finding the root of target function. In some scenarios, the target is more than just the root but the stationary (maximal/minimal) points of the target function. Given that the maximal/minimal points are only achieved when the derivative is zero, finding the root of the derivative means finding the maximal/minimal points of the target function. Recalling the Taylor expansion from first-order Newton's method:

$$y = f(x_n) + f'(x_n)(x - x_n) + \frac{1}{2}f'(x_n)(x - x_n)^2 + \cdots \tag{96}$$

Truncating high-order items and applying the derivative gives:

$$y' = f''(x_n)(x_{n+1} - x_n) + f'(x_n) = 0 \tag{97}$$

Thus,

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} \tag{98}$$

where $f''(x_n)$ is the second derivative of the target function f, which is why the method is named the second-order method. For the scalar-based problem formulation, the update equation is:

$$\boldsymbol{V_{n+1}} = \boldsymbol{V_n} - \boldsymbol{H^{-1}}(\boldsymbol{V_n})f'(\boldsymbol{V_n}) \tag{99}$$

Similar to the first order's formulas, $\boldsymbol{V_n}$ represents the current variables, and $\boldsymbol{V_{n+1}}$ represents the variables to try next, and the division becomes a left-multiplying of the inverse. Here, the second-order derivative $\boldsymbol{H}(\boldsymbol{V_n})$ is called the Hessian matrix. For the vector-based problem formulation, the update equation is:

$$\boldsymbol{V_{n+1}} = \boldsymbol{V_n} - \boldsymbol{H}^{-1}(\boldsymbol{V_n})\boldsymbol{J}(\boldsymbol{V_n}) \quad (100)$$

Here, the second-order derivative $\boldsymbol{H}(\boldsymbol{V_n})$ is the Hessian tensor. Compared with the first-order method, the second-order method is less sensitive to the choice of initial point and can work on complex/large-scale optimization problems. However, the Hessian matrix is a p-by-p matrix where p is the number of variables. For large number of variables, computing, inverting, and storing the Hessian matrix can be very expensive and, in some cases, unfeasible.

5.3.3 Finite Difference

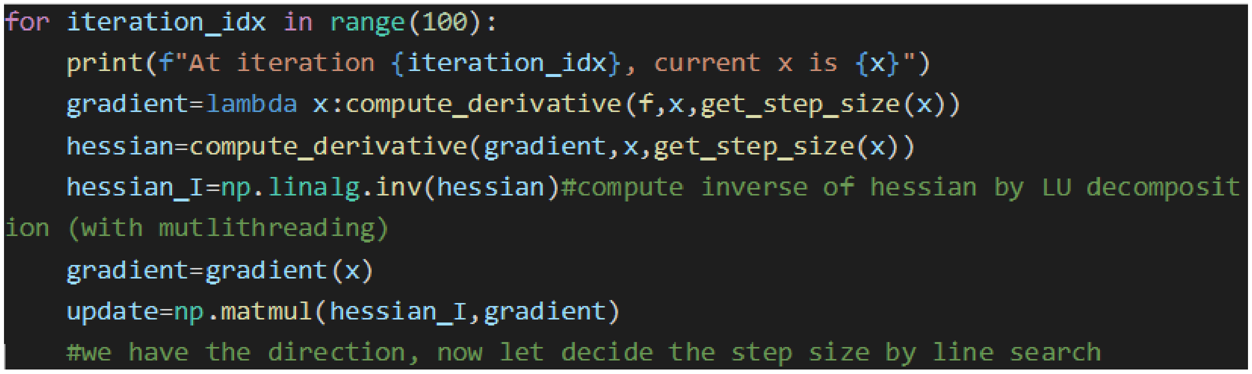
Newton's method, either the first or second order, depends on the derivatives. Yet the target function in the optimization framework is an abstract of the process of input generation, FE modeling, and deflection analysis, which is non-differentiable. Thus, the finite difference is used as an approach to approximate the derivative of the target function. Commonly used finite difference types include forward-difference, backward-difference, and central-difference. Researchers use forward-difference to reduce the number of function evaluations during optimization. The forward-difference equation is:

$$f' \quad (x) \approx \frac{f(x+h) - f(h)}{h} \quad (101)$$

where h is the step size. Typically, 1 percent of x is used as the step size but, for some variables, special values are used to make the approximation more accurate.

5.3.4 Implementation

With the cross-language optimizer interface, the optimizers can be implemented in any programming language. The Newton's method optimizers are implemented in Python to leverage its high-performance matrix operation such as matrix multiplication and inversing and the built-in Lambda function factory, which makes computation of higher-order derivatives much faster. A high-level example of the optimizer workflow is shown in Figure 42.



```
for iteration_idx in range(100):
    print(f"At iteration {iteration_idx}, current x is {x}")
    gradient=lambda x:compute_derivative(f,x,get_step_size(x))
    hessian=compute_derivative(gradient,x,get_step_size(x))
    hessian_I=np.linalg.inv(hessian)#compute inverse of hessian by LU decomposit
ion (with mutlithreading)
    gradient=gradient(x)
    update=np.matmul(hessian_I,gradient)
    #we have the direction, now let decide the step size by line search
```

Figure 42. Code Snippet of Newton Optimizer Implementation

5.3.5 Evaluation

The Newton method optimizers are evaluated using both a synthetic one-layer pavement structure and a synthetic three-layer pavement structure. Overall, the evaluation results suggest that the implementation of the optimization framework and the optimizers are correct and effective. The evaluation on real-world measured data is also conducted.

5.3.5.1 One-Layer Pavement Structure

The evaluation of the one-layer pavement structure is performed to check the correctness of the developed optimizer since the ground truth can be easily derived. The target vertical deflections that are supposed to be fitted are shown in Figure 43. These curves are generated by using the following variable set {E = 20,000 psi, Rayleigh alpha = 20, Rayleigh beta = 0.002}. The optimizers are given initial variables set of {E = 5,000 psi, Rayleigh alpha = 5, Rayleigh beta = 0.006}, and they are expected to tweak the variables to recover the ground truth variable set that is used to generate target vertical deflections. Note the ground truth variables are unknown to the optimizers.

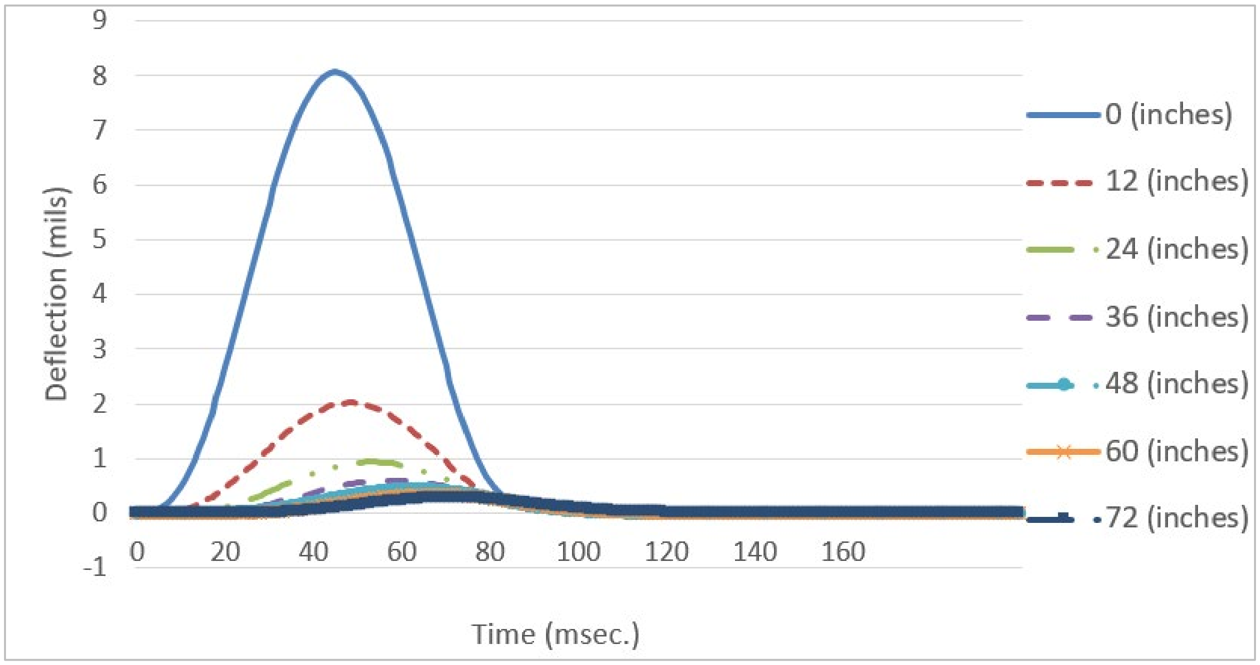


Figure 43. Synthetic Vertical Deflections for One-Layer Pavement Structure

The first-order Newton optimizer is first evaluated using both the scalar- and vector-based formulations, as shown in Figures 44 and 45, respectively. As the optimization progress curves show, the Newton optimizer can reduce the RMSRE to less than 0.1 percent (more than 99.9 percent accuracy) in both scalar- and vector-based formulations, yet the convergence rate (speed) differs. Specifically, the Newton optimizer on scalar-based formulation takes 30 iterations to converge to a reasonably good RMSRE, and the final recovered variables are {E: 19980, Rayleigh alpha: 20.755, Rayleigh beta: 0.001979}; whereas the Newton optimizer on vector-based formulation takes only 5 iterations to converge to a similarly good RMSRE level. This difference in speed matches the previous theoretical analysis. Note that because derivatives are approximated by finite difference, the vector-based formulation would need more calls to the target function. For example, in scalar-based formulation, the number of calls it takes to get the derivative (gradient) is equal to the number of variables plus one; whereas in vector-based formulation, the number of calls it takes to get derivative (Jacobian matrix) is equal to the number of variables multiplied by the number of parameters. That means the actual difference between these two formulations is smaller than it appears.

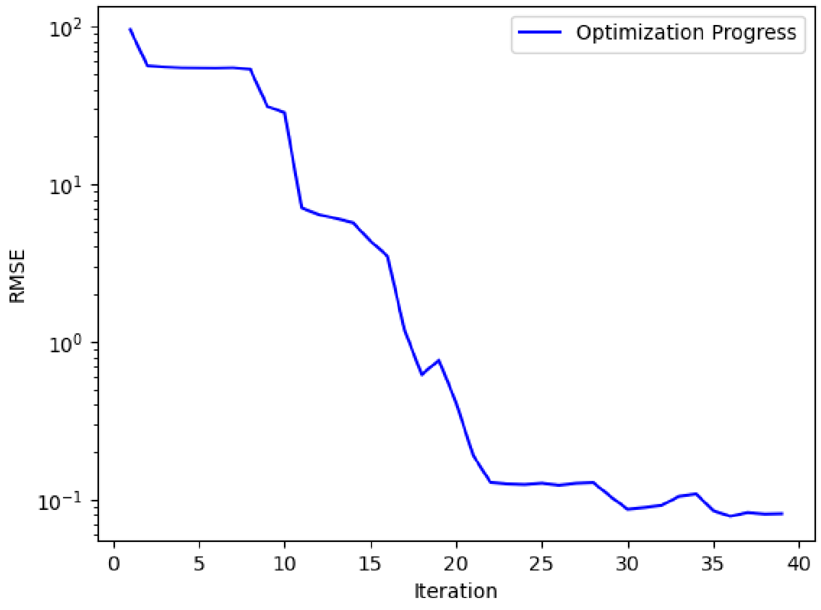


Figure 44. First-Order Newton Optimizer Fitting One-Layer Pavement Structure using Scalar-Based Problem Formulation

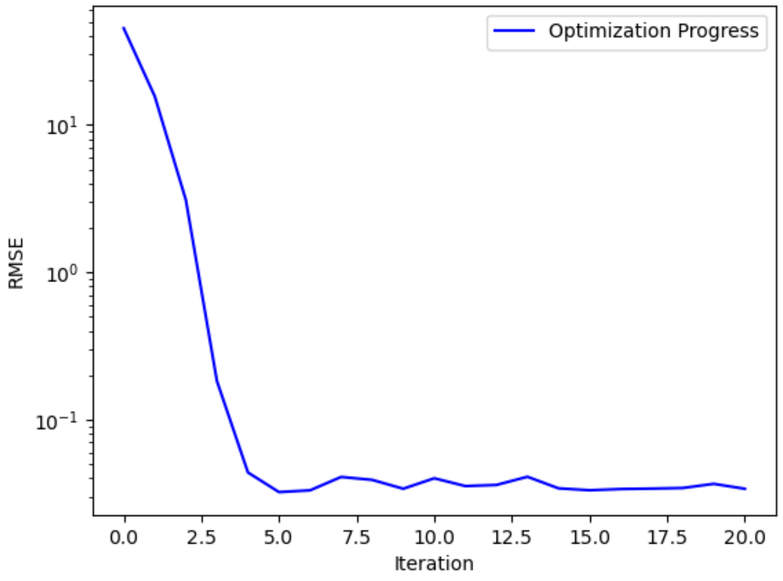


Figure 45. First-Order Newton Optimizer Fitting One-Layer Pavement Structure using Vector-Based Problem Formulation

The second-order Newton optimizer is also evaluated using the vector-based problem formulation. The results are shown in Figure 46, which indicates that its convergence is not as fast as the first-order methods. Researchers attempted to adjust the selection of parameters (see Section 4.4), but observed results were not better than the first-order method.

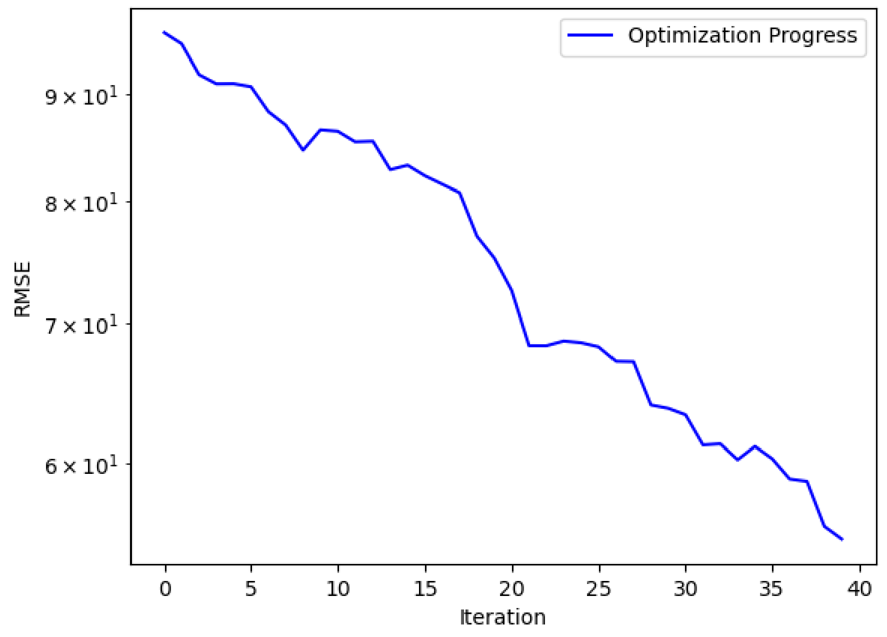


Figure 46. Second-Order Newton's Method Optimizer Fitting One-Layer Pavement Structure using Vector-Based Problem Formulation

5.3.5.2 Three-Layer Pavement Structure

Next, a more complicated three-layer pavement structure was evaluated. The target vertical deflections that were to be fitted are shown in Figure 47. These curves are generated by using the variables set shown in Table 10.

Similar to the one-layer case, the ground truth variables set that is used to generate the target vertical deflections was targeted. Note the ground truth variables are also unknown to the optimizers.

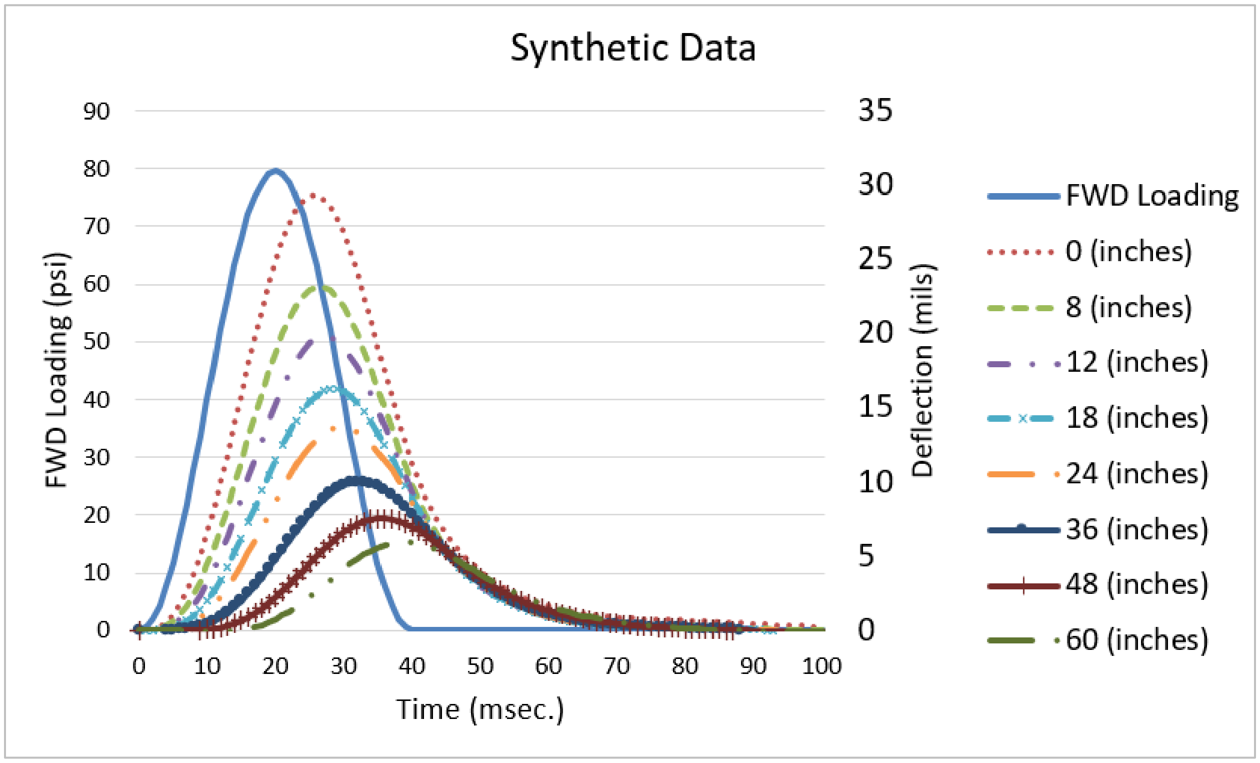


Figure 47. Synthetic Vertical Deflections and FWD Loading Time Histories for Three-Layer Pavement Structure

Table 10. Results of First-Order Newton Optimization Performance in Three-Layer System

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal function coefficients	delta	–0.134	–1	–0.920
		alpha	3.703	4.65	4.941
		betaPrime	–0.465118	– 0.265118	–0.468556
		gamma	–0.548	–0.65	–0.398
	Modulus at 17 Hz (ksi)	E1	469	286	447
Base	Modulus (ksi)	E2	40	50	40
	Rayleigh Damping Coefficient β_R	Bbase	0.002	0.003	0.00193
Subgrade	Modulus (ksi)	E3	5	7	5
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0.00197

For the first-order Newton optimizer on vector-based problem formulation, the setting that works best in the one-layer structure is evaluated for the three-layer system. The optimization history of

40 iterations is shown in Figure 48. The final RMSRE is 0.056 percent (over 99.9 percent accuracy). The final recovered variables are:

- Delta = –0.92007093
- Alpha = 4.49417398
- BetaPrime = –0.46855696
- Gamma = –0.39846419
- E2 = 40,229.533 psi
- BBASE = 0.0019321
- E3 = 5,000.955 psi
- BSG = 0.0019732

The robustness of Newton's method is tested when the seed variables are arbitrary. The evaluation is performed again with native seeds (See Table 11), and it converges with RMSRE of 0.0758 percent. The results suggest that Newton's method is robust to different seed values.

It can be concluded that even for the much more complicated three-layer system with eight variables, the first-order Newton's method is still capable of recovering the ground truth variables. Further evaluation using real-world measured data is conducted.

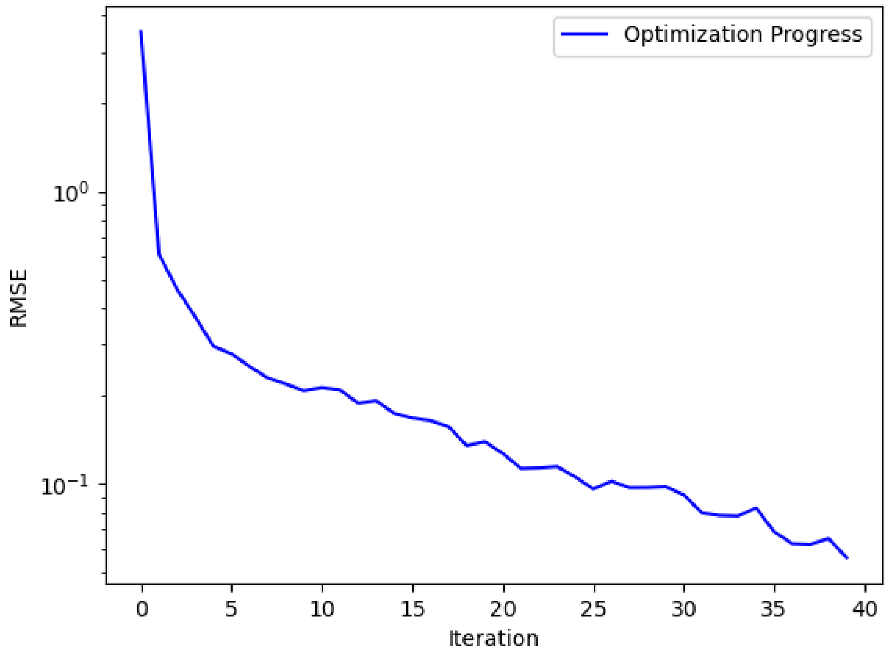


Figure 48. First-Order Newton's Method Optimizer Fitting Three-Layer Pavement Structure under Vector-Based Problem Formulation

Table 11. Cross-Comparison of Optimization Performance in Three-Layer System with Unoptimized Seeds

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.134	0	0.937
		alpha	3.703	1	1.998
		betaPrime	–0.465118	0	0.421
		gamma	–0.548	0	–2.165
	Modulus at 17 Hz (ksi)	E1	469	3	553.6
Base	Modulus (ksi)	E2	40	50	39.6
	Rayleigh Damping Coefficient β_R	Bbase	0.002	0.003	0.0021
Subgrade	Modulus (ksi)	E3	5	7	4.9
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0.002

The second-order Newton's method is also evaluated with the three-layer system and the results are shown in Figure 49. It was observed that the second-order method did not progress with a stable convergence. This could be improved by integrating a line search algorithm to adjust the step size of second-order Newton's method, which is also known as damped or relaxed Newton's method. In damped Newton's method, the updated equation is:

$$\boldsymbol{V_{n+1}} = \boldsymbol{V_n} - \gamma \boldsymbol{H^{-1}}(\boldsymbol{V_n}) f'(\boldsymbol{V_n})$$

where $0 < \gamma < 1$. This extra parameter γ is to help Newton's method converge better when facing overlarge gradient values. There is an example in Appendix A.

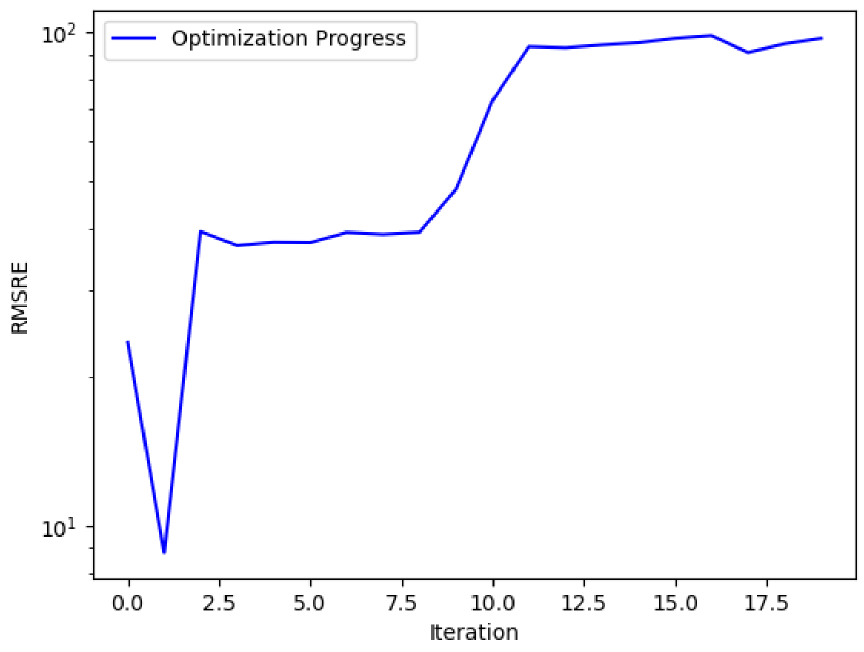


Figure 49. Second-Order Newton's Method Optimizer Fitting Three-Layer Pavement Structure under Vector-Based Problem Formulation

5.3.5.3 Evaluation of Field-Measured Deflection Data

Researchers further evaluated the performance of the Newton's method optimizers with field-measured deflection data on a three-layer flexible pavement system. The field-measured data were obtained from the LTPP database for test section 46-0804 in South Dakota. The section was built in June 1993, and it consisted of a 7.1-inch AC layer and a 12-inch unbound, granular base layer built over an untreated, silty clay subgrade. The FWD data used in this study were collected on June 8, 1994, using a Dynatest® FWD, on the eastbound lanes at milepost 400 of South Dakota Highway 1804, which is 5.5 miles northwest of Pollock, South Dakota. The surface and air temperatures at the time of testing were 48 °F and 57 °F, respectively. The measured deflections are shown in Figure 50.

Both vector- and scalar-based problem definitions were evaluated on the measured data. For the vector-based problem definition, the optimization progress of first-order Newton's method is shown in Figure 51. As the figure shows, the first-order Newton's method works well for field data. Specifically, the optimization progress converges at iteration 3, which indicates the power of Newton's method optimizer.

The final results are shown in Table 12 with a final RMSRE of 0.007 percent. From the perspective of function evaluations, the Newton's method achieves the RMSRE with less than 25 function evaluations, as shown in Figure 52, which demonstrates its efficiency.

Given the promising results of Newton's method with the vector-based problem definition, both first- and second-order Newton's method with scalar-based problem definition were further evaluated. Both methods diverged at iteration 1 when working on field data. This evaluation shows the limitations of Newton's method when working with scalar-based problem definition and the necessity of using different problem definitions for different optimizers.

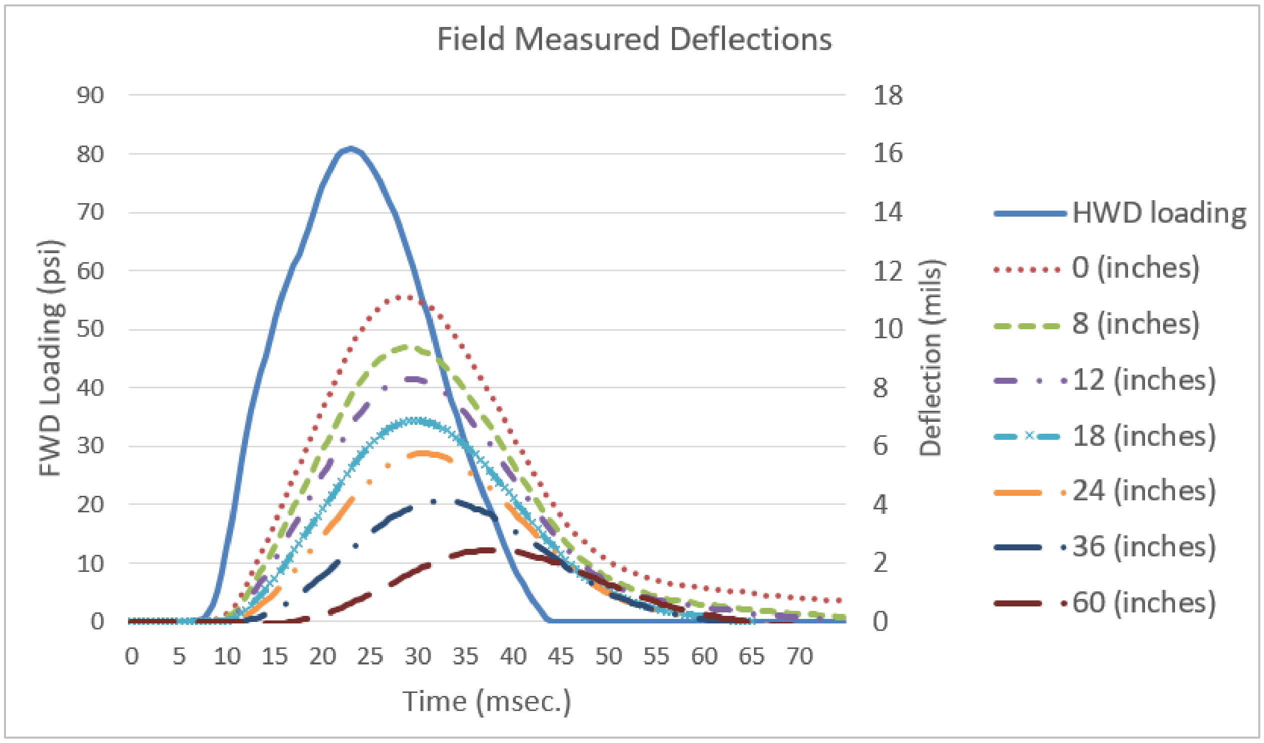


Figure 50. Field-Measured Deflections and FWD Loading Time Histories from an Actual Three-Layer System

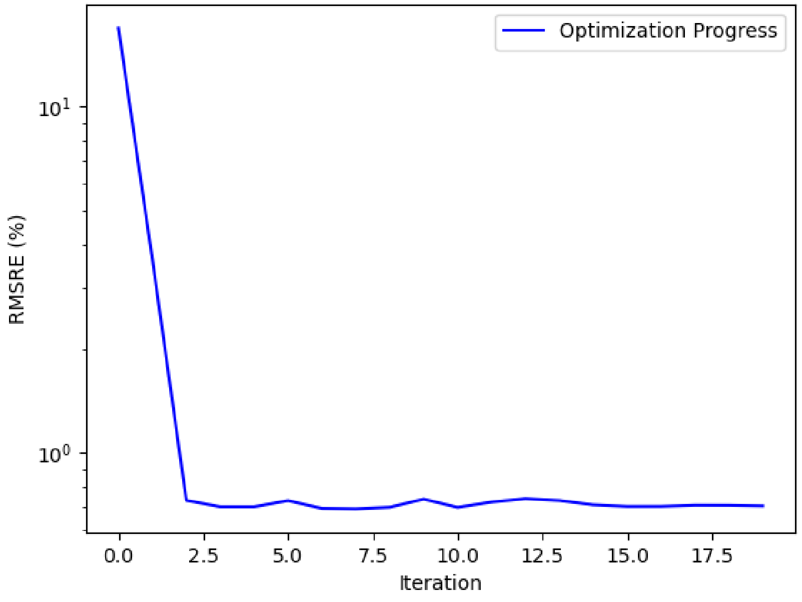


Figure 51. First-Order Newton's Method Optimizer Fitting Field Deflection Data on a Three-Layer Pavement under Vector-Based Problem Formulation

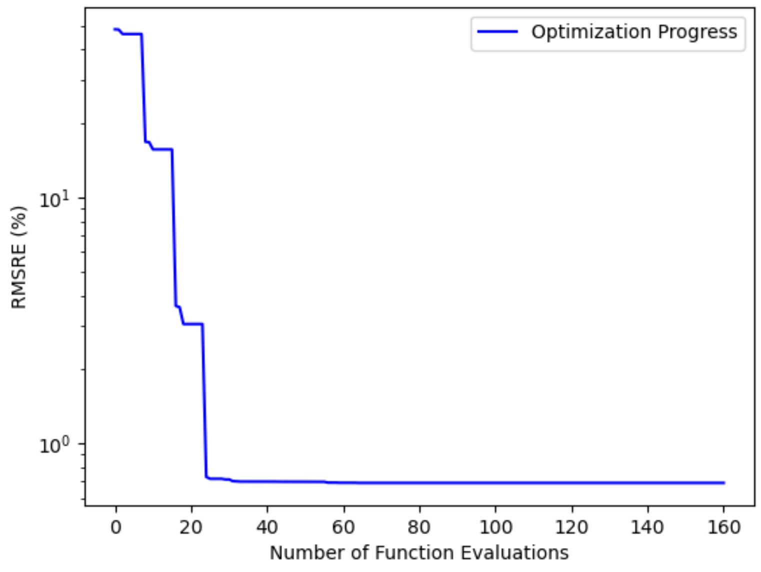


Figure 52. First-Order Newton Optimizer Fitting Field Deflection Data on a Three-Layer Pavement under Vector-Based Problem Formulation with Respect to Number of Function Evaluations

Table 12. Final Results of First-Order Newton's Method on Field-Measured Data with Vector-Based Problem Definition

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.9	–1.068
		alpha	4.5	5.160
		betaPrime	–0.7	–0.6974
		gamma	–0.4	–0.2837
	Modulus at 17 Hz (ksi)	E1	356	563.0
Base	Modulus (ksi)	E2	20	40.0
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.002448
Subgrade	Modulus (ksi)	E3	5	13.1
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.002448

5.4 TRADITIONAL OPTIMIZERS—QUASI-NEWTON METHOD

In Section 5.3.2, the second-order Newton's method that relies on the second partial-derivative Hessian matrix rather than the gradient to perform optimization was evaluated. This required computing and storing the inverse of Hessian matrix, which could be computationally expensive ($O(n^3)$ computational complexity) even with the help of mathematical approximation techniques like SVD. The Quasi-Newton methods are proposed to approximate the Hessian matrix (Chen et al., 2012) without computing it at every iteration to speed up the computation. The approximation methods include the Broyden–Fletcher–Goldfarb–Shanno (BFGS) (Moritz et al., 2016), Broyden (Huang et al., 2015), Davidon–Fletcher–Powell (DFP) (Pu & Wu, 1990), and Symmetric Rank 1 (SR1) (Brust et al., 2016) algorithms. The general idea is to compute the Hessian matrix only once and update it at every iteration instead of computing it from scratch every time. In this way, the per-iteration computation cost could be reduced at the price of potential robustness loss.

5.4.1 Broyden–Fletcher–Goldfarb–Shanno Algorithm

The BFGS algorithm (Moritz et al., 2016) is one of the most popular Quasi-Newton methods. Its core idea is to gradually approximate the Hessian matrix by a generalized secant method based on gradients. By using linear algebra tricks, the matrix inversion could be avoided, and the overall computational complexity could be reduced to $O(n^2)$, which is one magnitude lower than $O(n^3)$ of the second-order Newton's method. Specifically, B denotes the BFGS approximation of the Hessian matrix, x denotes the variables, and g denotes the gradient of target function $f(x)$. During initialization, B_0 is set to I. At iteration t, the following relations are obtained:

$$x_{t+1} = x_t - B_t^{-1} g \tag{102}$$
$$s_t = x_{t+1} - x_t \tag{103}$$
$$y_t = g_{t+1} - g_t \tag{104}$$
$$B_{t+1} = B_t - \frac{B_t s_t s_t^T B_t^T}{s_t^T B_t s_t} + \frac{y_t y_t^T}{y_t^T s_t} \tag{105}$$

By iteratively updating the approximation of the Hessian matrix, the optimization process approaches the target variables. For the object problem, the target function is not differentiable. Thus, the gradient g is approximated by finite-difference as discussed in Section 5.3.3.

5.4.2 Numerical Stability

The BFGS algorithm was implemented and evaluated in both the one- and three-layer systems with scalar-based problem definition. All settings were consistent with previous evaluation in Section 5.3.5. In the one-layer system, the BFGS was able to quickly converge to an optimal point of 0.533 percent RMSRE, as shown in Figure 53. Compared with the Newton's method results, the RMSRE was higher, yet the optimization took less time, which matched exactly with previous analysis.

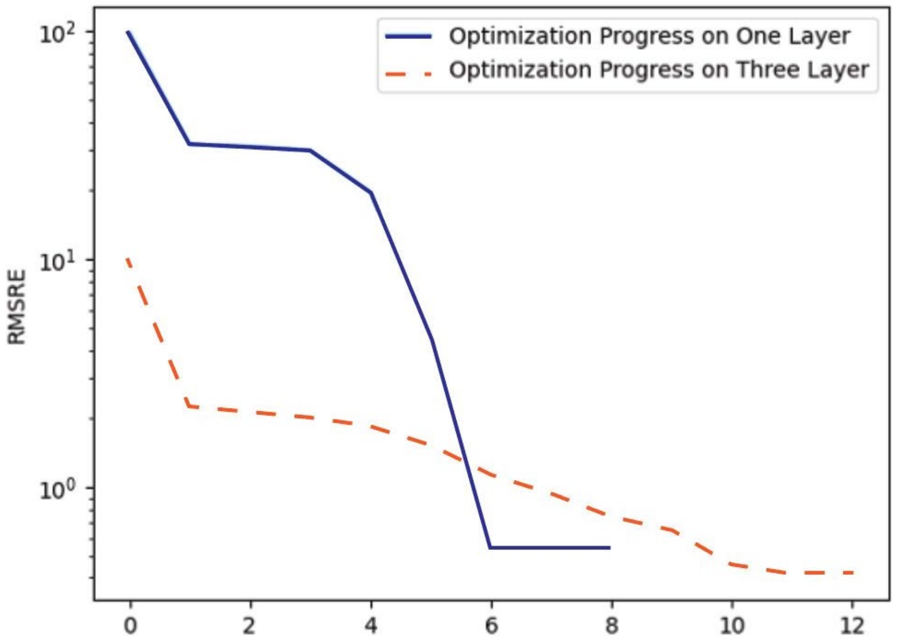


Figure 53. Optimization Progress of BFGS on One- and Three-Layer Systems

Note that the optimization on the three-layer system ends prematurely as the BFGS gives illegal input to the finite element model. For example, as early as iteration 2 on the three-layer system, the BFGS tries to set the Rayleigh beta damping coefficients of the second and third layers to be –0.32 and –0.36, respectively. This directly stops the optimization process, as shown in Figure 54. Such behavior is expected as the optimization algorithm has no knowledge of the physical meaning of the variables. As a result, it will arbitrarily explore the parameter space. Researchers first tried to work around this by implementing a guard procedure in the middleware (see Section 5.1.4) to fix the input when it is illegal. For example, the Rayleigh beta damping coefficients are set as a small positive number if they are non-positive. However, experimental evaluation shows that such patch often confuses the optimizer and thus makes the optimization diverge. To solve this problem, the framework will return a large punishment RMSRE value (e.g., 300 percent) to the optimizer if illegal inputs are detected. Evaluation suggests that this patch works very well. As shown in Figure 53, the BFGS method converges to optimal point of 0.420 percent RMSRE within 10 iterations and the recovered variables are shown in Table 13.

Table 13. Optimization Performance of BFGS in Three-layer System

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.134	–1	–0.925
		alpha	3.703	4.65	4.682
		betaPrime	–0.465118	–0.265118	–0.289
		gamma	–0.548	–0.65	–0.697
	Modulus at 17 Hz (ksi)	E1	469	286	424.7
Base	Modulus (ksi)	E2	40	50	39.8
	Rayleigh Damping Coefficient β_R	BBase	0.002	0.003	0.00150
Subgrade	Modulus (ksi)	E3	5	7	4.9
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0.00198

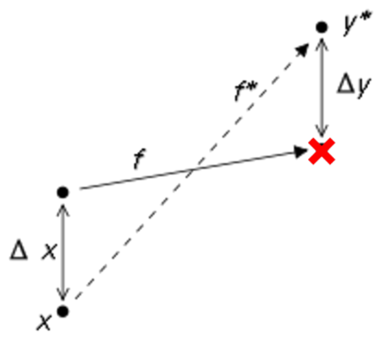


Figure 54. How Lack of Numerical Stability Stops Optimization from $\boldsymbol{x}$ to $\boldsymbol{y}^*$ (The red X represents the crash due to illegal input.)

5.4.3 Ablation Study

Next, researchers extended the evaluations of BFGS optimizer to field data, which was more challenging for the optimization than the previously tested synthetic data. The BFGS optimization on field data resulted in an RMSRE of 47.97 percent, which was far from the desired level of error. Thus, an ablation study of various possible improvements of the BFGS with field data was run to further understand performance impact of different improvements on the BFGS.

The first optimization that was applied was the step size tuning. By default, the BFGS optimizer uses a very small (1e–6 or smaller) step size for finite difference computation. For the target problem, this step size was trivial and did not cause any difference in the deflections. Thus, the step size was updated to 1 percent of the seed values. The evaluation shows this led to a better RMSRE of 34.43 percent compared to the previous value of 47.97 percent.

Normalization was then implemented for improvement. Normalization is a common action in optimization that is used to fix the problem that occurs when variables with large values receive overwhelming attention from the optimizer while the variables with small values are ignored. Specifically, normalization works by rescaling variables with different scales into the same scale so that all variables influence the optimizer to the same extent. Figure 55 shows an example of how normalization reshapes a 2D optimization space and makes it easier to find optimal point.

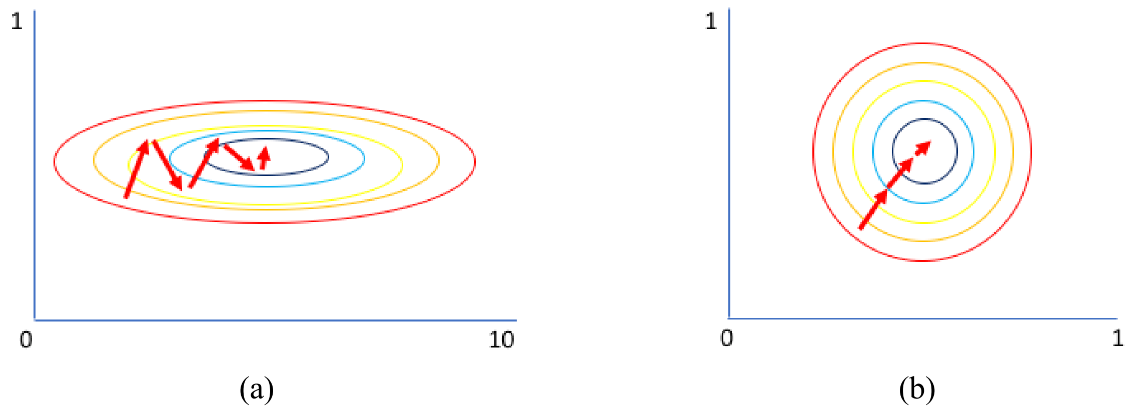


Figure 55. How Normalization Impacts Optimization: (a) Optimization Space without Normalization and (b) Optimization Space with Normalization (Arrows represent optimization steps.)

In the target problem, the range of the variables were different from the levels of 0.001 to 100. Thus, normalization was a favorable way to improve the BFGS optimizer performance. In summary, the implementation of normalization happens in the middle layer of the optimization framework. The middle layer uses the initial values (seed values) of all variables as the normalization base, so that each variable will be divided by its initial value before passing to the optimizer. There are two advantages of this implementation. First, all optimization operations happen at middle layer where both the FE simulation and the optimizer workflows stay untouched. This makes the normalization transparent, and no modification is needed on the FE simulation or the optimizer. Second, by using initial values as normalization bases, the optimization framework does not need prior knowledge of the variables, which makes it a more generalized and flexible approach.

The BFGS with normalization was evaluated on field data. The results aligned with the researchers' expectation: BFGS with normalization had an improved RMSRE of 34.54 percent compared with the previous RMSRE of 47.97 percent of the original BFGS, which is also referred to as *vanilla* BFGS.

The BFGS with both the tuned step size and the normalization improvements was further evaluated. The optimization process is shown in Figure 56. The figure shows that the optimization ran smoothly, and the final RMSRE was 13.20 percent.

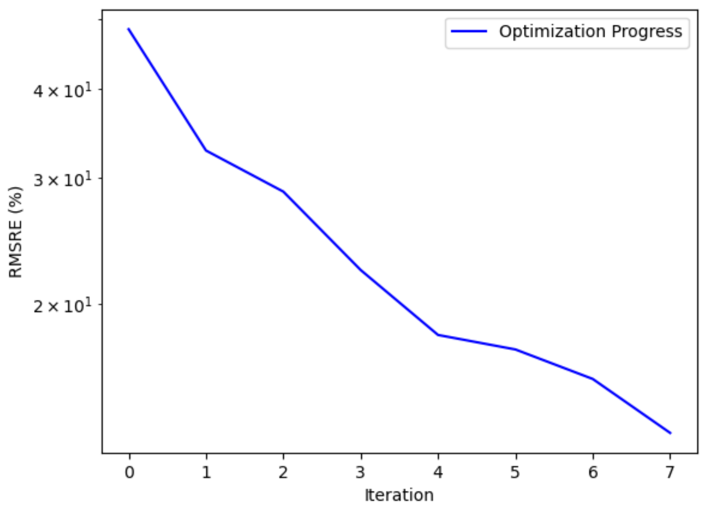


Figure 56. Optimization Progress of BFGS Optimizer on Field-Measured Data with Tuned Step Size and Normalization

The impacts of the two improvements are summarized in Table 14. Although the final RMSRE, even with two improvements, was still above the desired levels, it shows that these two improvements are useful and could be applied to other optimizers.

Table 14. Optimization Performance of BFGS on Field-Measured Data with Different Improvements

Name	Final RMSRE (%)
Original BFGS	47.97
BFGS with tuned step size	34.43
BFGS with normalization	34.54
BFGS with both tuned step size and normalization	13.20

5.4.4 Limited-Memory BFGS with Bound Constraints

The limited-memory BFGS with bound constraints (L-BFGS-B) algorithm (Byrd et al., 1995) is an extension of the BFGS algorithm. The main differences are: (1) it avoids the potentially memory-expensive matrix operation of $B_t^{-1}g$ (see Equation 102 in Section 5.4.1), and (2) it is able to handle simple box constraints (bound constraints). To do that, it uses a recursion loop to compute $B_t^{-1}g$ and a gradient method to identify the free variables (those inside the constraints) and fixed variables (those on the boundaries of constraints). Given its promising features, the L-BFGS-B was applied to solve the object optimization problem.

5.4.5 Evaluation

The L-BFGS-B algorithm was evaluated on the one- and three-layer systems with a scalar-based problem definition. All settings were consistent with the previous evaluation described in Section 5.3.5.

As shown in Figure 57, the L-BFGS-B algorithm performed similarly to the original BFGS. It converged to an optimal point of 1.273 percent RMSRE on the one-layer system and an optimal point of 1.361 percent RMSRE on the three-layer system. The details of recovered variables are shown in Table 15.

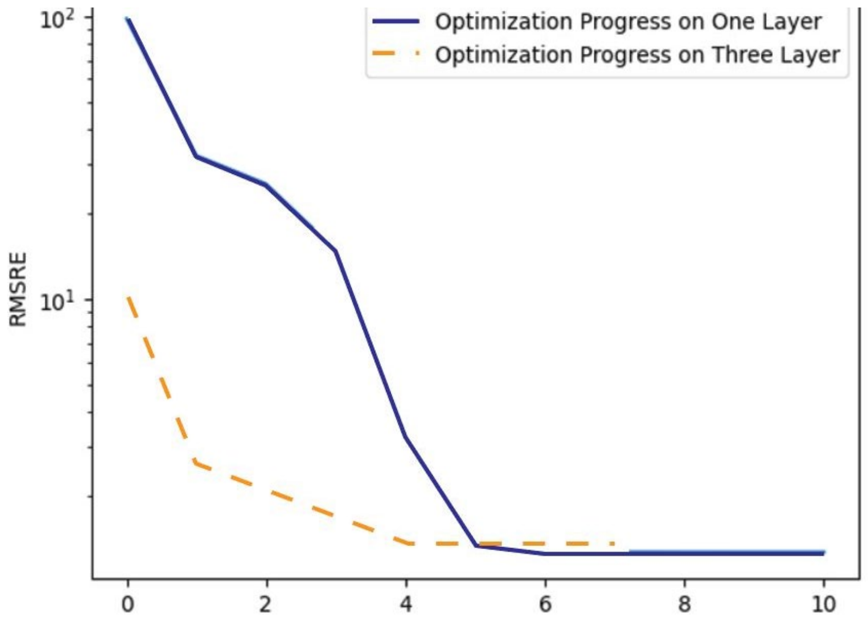


Figure 57. Optimization Progress of L-BFGS-B in One- and Three-Layer Systems

Table 15. Optimization Performance of L-BFGS-B in a Three-Layer System

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.134	–1	–1.008
		alpha	3.703	4.65	4.520
		betaPrime	–0.465118	–0.265118	–0.262
		gamma	–0.548	–0.65	–0.647
	Modulus at 17 Hz (ksi)	E1	469	286	222.4
Base	Modulus (ksi)	E2	40	50	49.8
	Rayleigh Damping Coefficient β_R	BBase	0.002	0.003	0.00301
Subgrade	Modulus (ksi)	E3	5	7	5,1
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0.00100

Researchers further evaluated L-BFGS-B performance on the field-measured data. The optimization progress is shown in Figure 58. Compared with BFGS, which converges with large RMSRE, the L-BFGS-B ran smoothly and converged at iteration 7 with RMSRE of 1.4689 percent. The recovered variables are shown in Table 16. Other classical optimizers that do not rely on derivatives to function were evaluated next.

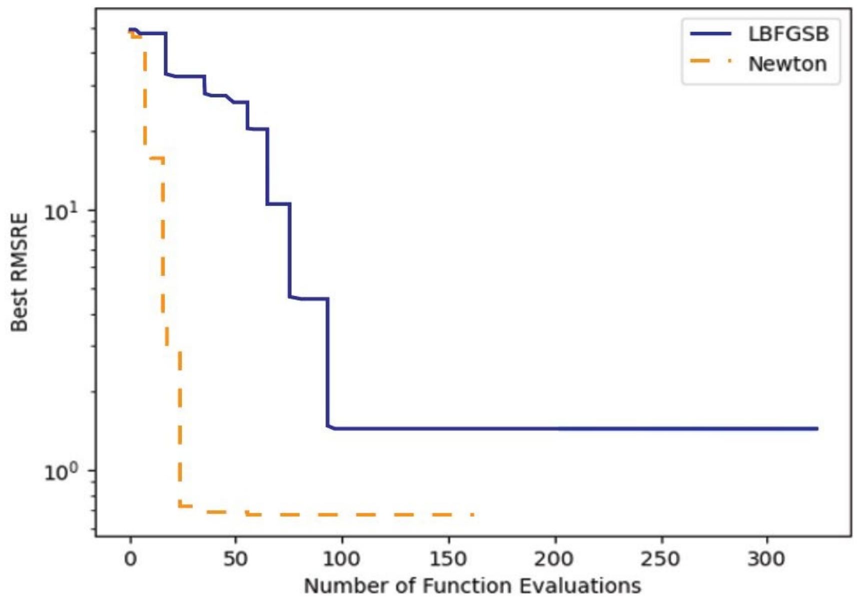


Figure 58. Optimization Progress of L-BFGS-B on Field-Measured Data for a Three-Layer System

Table 16. Optimization Performance of L-BFGS-B on Field-Measured Data for a Three-Layer System

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.9	–0.8284
		alpha	4.5	4.787
		betaPrime	–0.7	–0.8438
		gamma	–0.4	–0.3495
	Modulus at 17 Hz (ksi)	E1	356	816.8
Base	Modulus (ksi)	E2	20	23.2
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.002788
Subgrade	Modulus (ksi)	E3	5	13.6
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.002788

5.5 POWELL'S CONJUGATE DIRECTION METHOD

Powell's conjugate direction method (Powell, 1964), commonly referred to as Powell's method, is a heuristic algorithm that does not need derivatives to work. It is also capable of working with constraints, which makes it a promising approach for solving the object optimization problem.

5.5.1 Algorithm

The core idea of Powell's method is that the local extremums could form a line that conjugates to the direction towards global extremums, which is why it is named conjugate direction method. Specifically, Powell's method initializes a set of search direction vectors S to be the unit vectors on each dimension of search space. At each iteration, x denotes the current point and $f(x)$ denotes the target function.

$$x_t = x \tag{106}$$

For each direction s in S:

$$\gamma = argmin(f(x_t + \gamma s)) \tag{107}$$

$$x_t = x_t + \gamma s \tag{108}$$

then:

$$s^* = x - x_t \tag{109}$$

$$\gamma^* = argmin(x + \gamma^* s^*) \tag{110}$$

$$x = x + \gamma^* s^* \tag{111}$$

Update S with s^*. From this algorithm, it is clear that Powell's method does not need the derivative information.

5.5.2 Evaluation

Powell's method was evaluated on the one- and three-layer systems with a scalar-based problem definition. All settings were consistent with the previous evaluation described in Section 5.3.5. The optimization progress is shown in Figure 59. In the one-layer system, the final RMSRE was 0.0290 percent, and in the three-layer system, the final RMSRE was 0.125 percent. Table 17 shows the recovered variables.

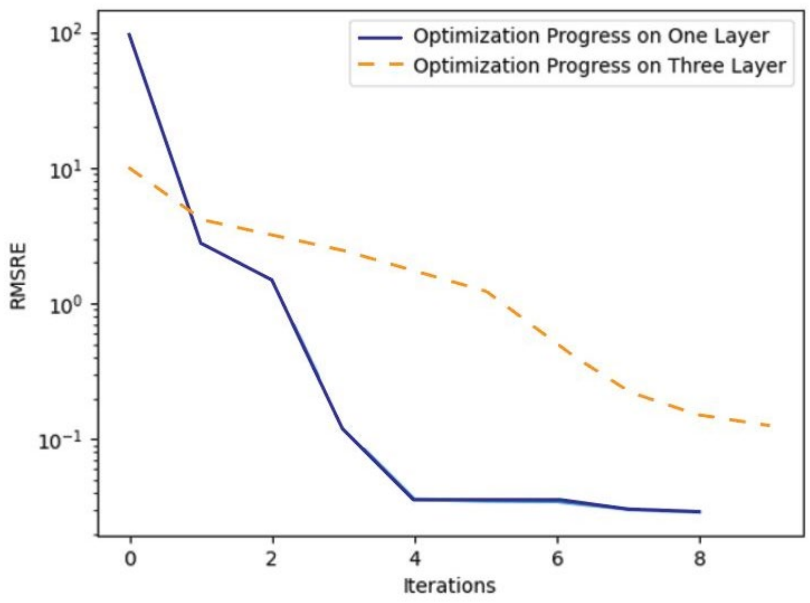


Figure 59. Optimization Progress of Powell's Method in One- and Three-Layer Systems

Table 17. Optimization Performance of Powell's Method in the Three-Layer System

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal Coefficients	delta	–0.134	–1	–1.316
		alpha	3.703	4.65	4.451
		betaPrime	–0.465118	–0.265118	–0.897
		gamma	–0.548	–0.65	–1.151
	Modulus at 17 Hz (ksi)	E1	469	286	542.3
Base	Modulus	E2	40	50	39.0
	Rayleigh Damping Coefficient β_R	BBase	0.002	0.003	0.00196
Subgrade	Modulus (ksi)	E3	5	7	4.9
	Rayleigh Damping Coefficient β_R (ksi)	BSG	0.002	0.001	0.00198

It is worth noting that Powell's method uses a great number of function evaluations in each iteration as it needs to do a line search along every direction (see Equation 107). This can be considered a drawback compared to the previous derivative-based approaches. Next, the Powell's method optimizer on field-measured deflections of a three-layer system was evaluated (Figure 60 and Table 18).

From the optimization progress in Figure 60, the Powell's method optimizer seemed to run smoothly on field-measured data. The final RMSRE was 1.95 percent. However, the recovered variables (Table 18) were unreasonable. Notably, the gamma was a positive value, 0.19.

According to Equation 76, this means the E (time-temperature-dependent relaxation modulus) would decrease as frequency increases, which is unrealistic. This is understandable because optimizers have no background knowledge about what the physical meaning of variables are. Therefore, they will arbitrarily explore the optimization space and stop at a point with low RMSRE, which might not be a feasible solution in practice. Thus, constrained optimization was evaluated as a way to fix this issue.

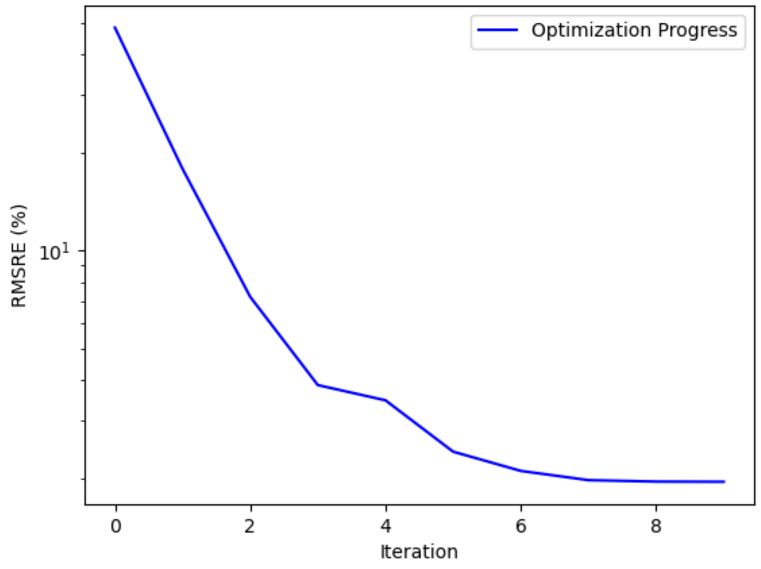


Figure 60. Optimization Progress of Powell's Method on Field-Measured Deflections of a Three-Layer System

Table 18. Optimization Performance of Powell's Method on Field-Measured Deflections of a Three-Layer System

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.9	–0.9958
		alpha	4.5	4.515
		betaPrime	–0.7	–0.9186
		gamma	–0.4	0.1958
	Modulus at 17 Hz (ksi)	E1	356	99.6
Base	Modulus (ksi)	E2	20	41.9
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.003179
Subgrade	Modulus (ksi)	E3	5	12.6
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.003179

5.5.3 Powell's Method Optimizer with Constraints

Given that the Powell's method optimizer provided unpractical solutions, researchers proposed to use constrained optimization to instruct the optimizer to stay in a certain area of the optimization space to avoid infeasible solutions. For the target problem, researchers constrained the Powell's method optimizer to only consider the solutions with gamma smaller than zero. The optimization progress with constraints is shown in Figure 61. Note that the x-axis of this figure is the number of function evaluations. The final RMSRE of 8.092 percent was higher than the RMSRE without constraints. This was expected as the optimizer with constraints cannot freely explore the optimization space and can only pick solutions from a subset of all possible solutions, which makes the RMSRE higher.

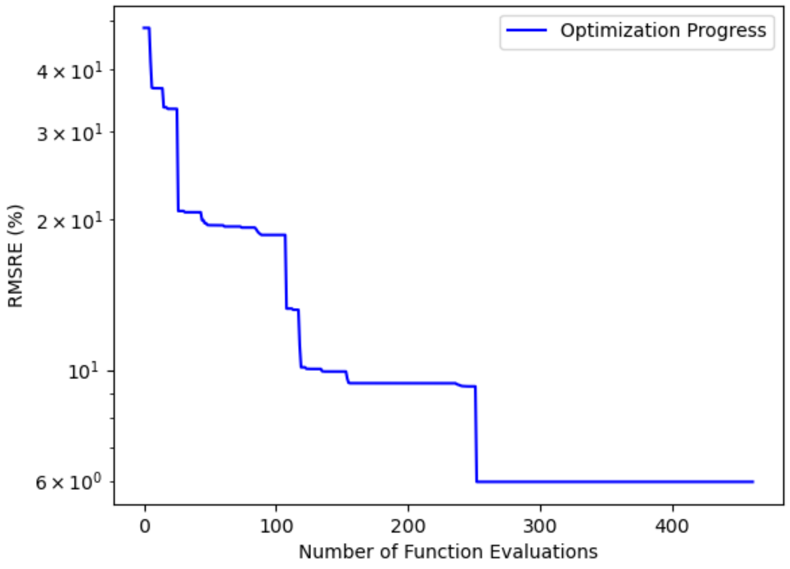


Figure 61. Optimization Progress of Powell's Method on Field-Measured Deflection of a Three-Layer System with Constraints (Note the x-axis is number of function evaluations.)

The recovered variables are summarized in Table 19, which shows the Powell's method optimizer followed the constraints well and set the recovered gamma to a negative value. However, it should also be noted that this gamma value is close to zero, which suggests that Powell's method could still prefer a positive gamma but was unable to do so as it was limited by the constraints. From an optimization perspective, this phenomenon is called local optimal, where the optimizer mistakenly selects a non-optimal solution as the optimal one.

Table 19. Optimization Performance of Powell's Method in the Field-Measured Three-Layer System with Constraints

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.9	–0.9958
		alpha	4.5	4.299
		betaPrime	–0.7	–0.9141
		gamma	–0.4	–0.002975
	Modulus at 17 Hz (ksi)	E1	356	119.1
Base	Modulus (ksi)	E2	20	91.4
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.005986
Subgrade	Modulus (ksi)	E3	5	11.6
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.005986

To improve the Powell's method performance with constraints, different seed (initial) values were used to get the Powell optimizer out of the local optimal. The first set of seeds evaluated was a native one with all variables set to zero. This set led the FE model to crash. Then researchers tried a different set by cutting all seed values to half of the original values. This one worked well, and the optimization progress is shown in Figure 62. The final RMSRE is 3.956 percent, and the recovered values are shown in Table 20 together with the recovered variables with original seed values as comparison. It is clear that the seed values play an important role in the convergence point of optimizers. The impact of seed values was then systematically evaluated for all optimizers.

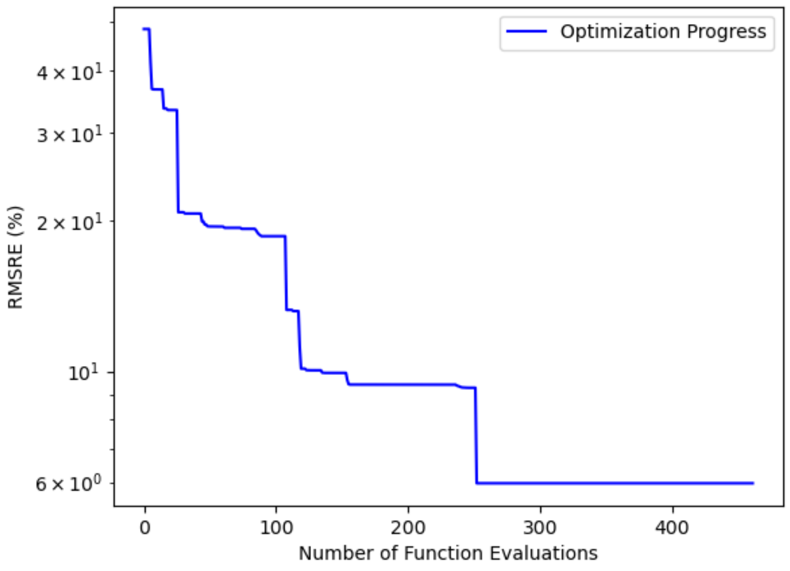


Figure 62. Optimization Progress of Powell's Method on Field-Measured Deflections of a Three-Layer System with Constraints and a Different Seed Set (Note the x-axis is the number of function evaluations.)

Table 20. Optimization Performance of Powell on Field-Measured Deflections of a Three-Layer System with Constraints and Two Seed Sets

Layer	Variable		Previous Seed Value	Previous Recovered Value	Seed Value	Recovered Value
AC	Sigmoidal Coefficients	delta	–0.9	–0.9958	–0.45	–0.1693
		alpha	4.5	4.299	2.25	3.129
		betaPrime	–0.7	–0.9141	–0.35	–2.207
		gamma	–0.4	–0.002975	–0.2	–1.00471
	Modulus at 17 Hz (ksi)	E1	356	119.1	10.0	729.1
Base	Modulus (ksi)	E2	20	91.4	10.0	17.8
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.005986	0.0015	0.002948
Subgra de	Modulus (ksi)	E3	5	11.6	2.5	15.6
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.005986	0.0015	0.002948

5.6 NELDER–MEAD METHOD

The Nelder–Mead (Gao & Han, 2012) method, also known as downhill simplex method, is a commonly used heuristic numerical method for multidimensional optimization. Like Powell's method, the Nelder–Mead method is a direct-search algorithm that does not rely on derivatives to work.

5.6.1 Algorithm

The Nelder–Mead method works by gradually moving and shrinking a simplex (polytope) to an optimal point on the optimization space. Specifically, a simplex is initialized around the initial point. For an optimization space of n dimension, the simplex will have $n + 1$ vertices $x_1, x_2, \dots, x_{n+1}$. For example, a triangle on a plane, a tetrahedron in 3D space, and so forth. Then the algorithm begins as follows:

- Step 1: Calculate the centroid of current simplex as x_0.
- Step 2: Calculate next point $x_t = x_0 + \alpha(x_0 - x_{n+1})$.
- Step 3: If x_t is the best point among all vertices $x_1, x_2, \dots, x_{n+1}$, it means the optimal point is likely outside current simplex. Thus, the need to expand the simplex. The expand point $x_e = x_0 + \gamma(x_t - x_0)$.
- Step 4: Contract the simplex by $x_c = x_0 + \rho(x_{n+1} - x_0)$.
- Step 5: If none of x_t, x_e, x_c is better than original vertices $x_1, x_2, \dots, x_{n+1}$, shrink all vertices $x_1, x_2, \dots, x_{n+1}$ by $x_i = x_1 + \sigma(x_i - x_1)$ except the best vertex. Or, if some vertices in x_t, x_e, x_c are better than the original vertices $x_1, x_2, \dots, x_{n+1}$, pick n best vertices from $x_t, x_e, x_c, x_1, x_2, \dots, x_{n+1}$ as the new simplex.
- Step 6: Go to Step 1.

The $\alpha, \gamma, \rho, \sigma$ are parameters and by default $\alpha = 1, \gamma = 2,\ \rho = \frac{1}{2}$, and $\sigma = \frac{1}{2}$. This algorithm is different from previous optimizers' algorithms, where some mathematical information (for example, derivative/gradients) are computed and used to update the solution. As a heuristic algorithm, the Nelder–Mead method tries to resemble how a human thinks and behaves when facing an optimization problem.

5.6.2 Evaluation

The Nelder–Mead method was evaluated with the same synthetic three-layer pavement structure used for previous methods. The details of the target system are shown in Figure 47. The optimization progress of Nelder–Mead is shown in Figure 64. The method took almost 300 function evaluations to converge. It should be noted that the final 50 function evaluations had RMSRE within the 1 percent range. The final RMSRE was 0.5835 percent, and the recovered variables are shown in Table 21.

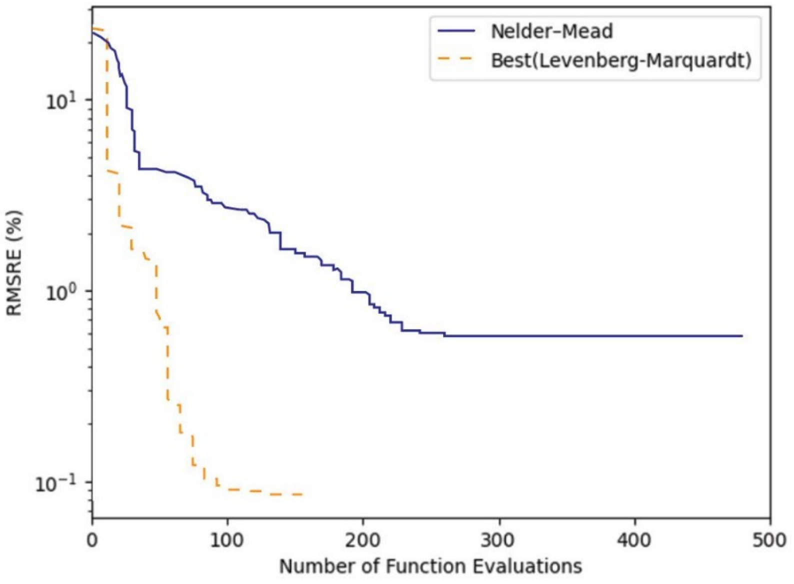


Figure 63. Optimization Progress of the Nelder–Mead Method on Deflections of a Synthetic Three-Layer Pavement Structure

Table 21. Optimization Performance of the Nelder–Mead Method in the Three-Layer Pavement Structure

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal Coefficients	delta	–0.134	–1	–0.0070439
		alpha	3.703	4.65	5.321895
		betaPrime	–0.465118	–0.265118	–0.00271253
		gamma	–0.548	–0.65	–0.0129539
	Modulus at 17 Hz (ksi)	E1	469	286	477.2
Base	Modulus (ksi)	E2	40	50	38.3
	Rayleigh Damping Coefficient β_R	BBase	0.002	0.003	0.0030104
Subgrade	Modulus (ksi)	E3	5	7	5.0
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0.00216480

Next, the Nelder–Mead method was evaluated with the field-measured deflection data of a three-layer pavement structure. The optimization progress is shown in Figure 64. The Nelder–Mead method converged at an RMSRE of 2.346 percent. Note that although Nelder–Mead used more iterations, its heuristic algorithm took fewer function evaluations at each iteration. Thus, the number of function evaluations as a metric was used when comparing the various evaluated optimizers.

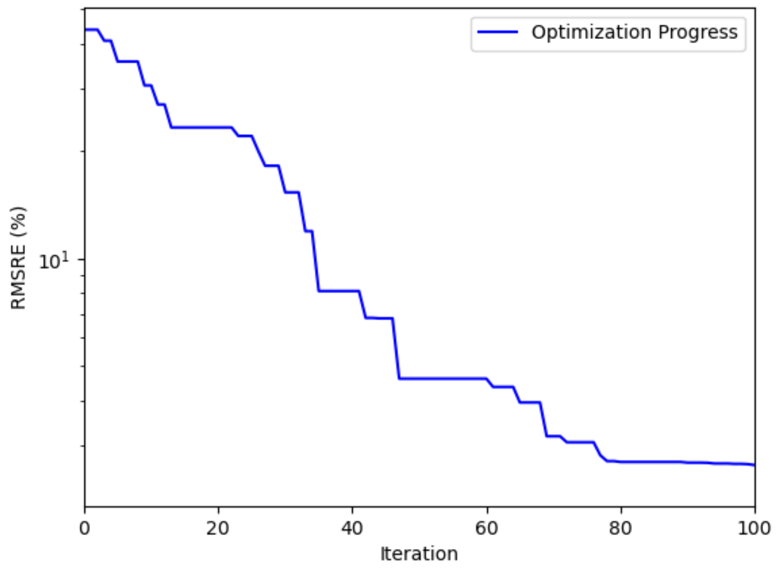


Figure 64. Optimization Progress of the Nelder–Mead Method on Field-Measured Deflections of a Three-Layer Pavement Structure

The recovered variables are shown in Table 22. The Nelder–Mead method has a similar problem to the Powell optimizer wherein the gamma value in the solution is positive, thus the solution is not practical. However, unlike Powell's method, the Nelder–Mead method does not support constrained optimizations, so it is impossible to limit it to only search solutions with negative gamma values. Different seed values were explored similar to what was discussed in Section 5.5.3 to see if they led to negative gamma values. The first seed values diverged with RMSRE of over 300 percent and the second one converged with gamma of 0.1. Thus, the Nelder–Mead might not be a suitable solution for the target problem.

Table 22. Optimization Performance of the Nelder–Mead Method on Field-Measured Deflections of a Three-Layer Pavement Structure

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal Coefficients	delta	–0.9	–0.4969
		alpha	4.5	5.184
		betaPrime	–0.7	–0.6342
		gamma	–0.4	0.03316
	Modulus at 17 Hz (ksi)	E1	356	695.5
Base	Modulus (ksi)	E2	20	17.8
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.003916
Subgrade	Modulus (ksi)	E3	5	13.2
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.003916

5.7 BAYESIAN OPTIMIZATION METHOD

As traditional optimization methods have been extensively evaluated ranging from Newton-based methods to heuristic optimizers, the next step was to evaluate learning-based optimizers. The most fundamental difference between traditional and learning-based optimizers is that learning-based optimizers usually use information from all iterations, whereas traditional optimizers only examine what happens at current or recent iterations. For example, when making decisions at iteration 100, most learning-based optimizers would consider results from all previous 99 iterations, whereas the traditional optimizers like Newton's method would only consider the gradients of current iteration. In other words, for Newton's methods, as long as the current variables' values and gradients do not change, the output would be the same regardless of the previous results. For learning-based optimizers, even if current variables' values and gradients do not change, the output would be dependent on previous results. As a result, the learning-based optimizers are usually more robust but have more parameters, which are more challenging to implement and tune.

The Bayesian optimization, a well-known learning-based algorithm, is based on a simple intuition: for target function $f(x) = y$, suppose it is known that $f(5) = 2$, then for the x near 5, its function value should be more likely near 2. This could be formulated from the probability perspective. For the unknown target function, it could be represented by a probability model where the known points (like the $f(5) = 2$ in the example) has 100 percent probability of yielding its corresponding results and the distanced points have little probability of yielding that result. In this way, the more known points there are, the more accurate the probability model will be. Finally, when the probability model is accurate enough, the optimal points can be naturally inferred from the probability model. This is called surrogate optimization, in which a surrogate model is first fitted to the target function and then the optimization takes place on this surrogate model.

Bayesian optimization is widely used in practice. For example, AlphaGo, Google's® chess-playing artificial intelligence (AI) that has beaten top-rated human players, uses the Bayesian optimization. In this section, researchers presented data from their previous work, as shown in Figure 65 (Chen et al., 2018), to more intuitively show how Bayesian optimization works.

In Figure 65, the target function $f(\theta)$ to be optimized is the red-dotted line, and it is unknown to the optimizer. The blue line is the surrogate model. The red circles on the red-dotted line are the previous known points, and the white circle is the current tested point. The light blue region is the probability distribution the optimizer concluded from the known points. As the figure shows, at first the Bayesian optimizer has little knowledge about the target function, so both the probability distribution and surrogate model are way off the target function. Along the training, the optimizer has more known points and converges the probability distribution to the true distribution as previously discussed. When the surrogate model is fitted to the target function at $t = 5$, the optimal point is also achieved.

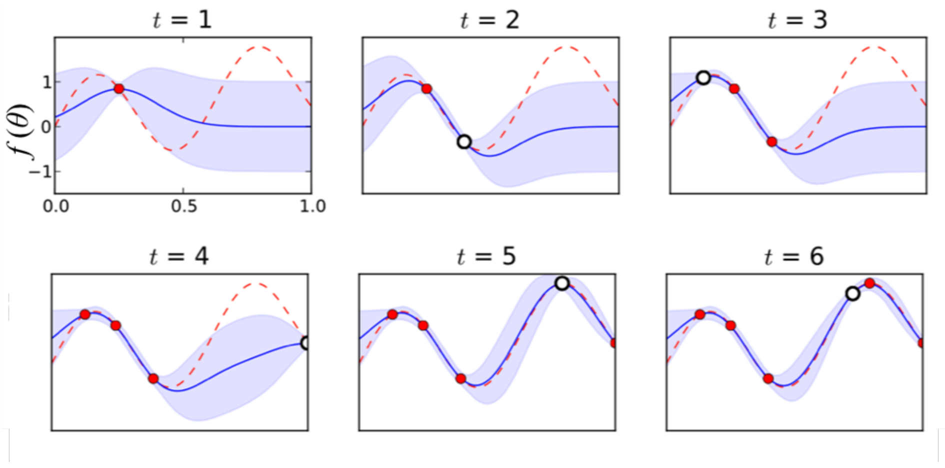


Figure 65. Bayesian Optimization Working on Example Function

5.7.1 Algorithm

On the algorithm level, Bayesian optimization is more challenging than it appears. The first challenge researchers need to address is the balance between exploitation and exploration, which is a long-time problem in the field of learning-based optimizers. Basically, the problem can be described in the following two questions:

- Suppose tests are done about the target function and a point with relatively good results is found, should the point be fine-tuned hoping for better results (i.e., exploitation)?
- Should the rest of the unknown space be explored to find another point with better results (i.e., exploration)?

For Bayesian optimization, three different strategies are used to handle the exploitation versus exploration tradeoff: Upper Confidence Bound (UCB), Expected Improvement (EI), and Probability of Improvement (POI). The UCB is a straightforward strategy that picks the maximal/minimal points from the probability distribution, which can be roughly understood as the edge of the light-blue-shaded region shown in Figure 65. The EI picks the next point with the highest expected improvement. The POI aims to pick the next point that has the highest probability of being better than best known point.

The next question to consider is what surrogate model should be used to approximate the target function? The most commonly used one is the Gaussian process. It is based on the multivariate Gaussian distribution (or sometimes known as joint normal distribution), which is a high-dimension extension of normal/gaussian distribution. For a scalar variable X, a normal distribution is defined as $X \sim N(\mu, \sigma^2)$, where μ is the mean and σ is the standard deviation. For a more complicated high-dimensional case where X is a vector, components of X could have potential impact on each other, so their covariance must be formulated. Specifically, a multivariate Gaussian distribution of vector X can be defined as $X \sim N(\mu, \Sigma)$, where mean vector $\mu = E(X)$ and elements in covariance matrix $\Sigma_{i,j} = E\left[(X_i - \mu_i)(X_j - \mu_j)\right]$. For Bayesian optimization, the target is to predict the distribution of the next point x_* with the knowledge of all previous known points $x_1 \dots x_t$ and their results $f_1 \dots f_t$. For conciseness, in the following discussions, known points will be referred to as x and their corresponding results as f. To predict next point x_*, the joint distribution of x and x_* is written as:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim N\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma_* \\ \Sigma_*^{\mathrm{T}} & \Sigma_{**} \end{bmatrix}\right) \tag{112}$$

Using the formula for conditioning a joint Gaussian distribution gives:

$$\mathrm{f}_* \mid \mathrm{f} \sim N(\mu_* + \Sigma_*^{\mathrm{T}} \Sigma^{-1} (\mathrm{f} - \mu), \Sigma_{**} - \Sigma_*^{\mathrm{T}} \Sigma^{-1} \Sigma_*) \tag{113}$$

Note that in the above equation, $f, \mu, \sum$ are already known and μ_* can be assumed to be same as μ since the target problem is the same. Thus, the only unknown items are the covariance terms $\sum_*$ and $\sum_{**}$. In other words, if there was a way to fit/model the covariance, the problem would be solved. Recall the basic intuition discussed earlier: closer inputs should yield similar outputs, and distanced inputs are likely to give different outputs. That means the covariance could be solely modelled based on inputs. The numerical models used to approximate the covariance are called kernel functions k. In practice, there are many limitations on the choice of kernel functions, but they are beyond the scope of this report. The most commonly used kernel function is the radial-basis function kernel (also known as squared-exponential kernel/Gaussian kernel):

$$k(x_i, x_j) = \exp\left(-\frac{d(x_i,\ x_j)^2}{2l^2}\right) \tag{114}$$

where the d is the Euclidean distance and l is a parameter.

However, this kernel function is often too simple to fit the real covariance in practice. Thus, one of its generalization forms called Matérn kernel is often used instead:

$$k(x_i, x_j) = \frac{1}{\Gamma(v)2^{v-1}} \left(\frac{\sqrt{2v}}{l} d(x_i, x_j)\right)^v K_v\left(\frac{\sqrt{2v}}{l} d(x_i, x_j)\right) \quad (115)$$

where Γ is the gamma function, K_v is the modified Bessel function of the second kind, and l and v are positive parameters. Note that when v goes to infinite, the Matérn kernel degenerates to the radial-basis function kernel. In the actual implementation, all parameters of kernel functions such as l and v are automatically inferred from the known points.

5.7.2 Evaluation

The Bayesian optimizer is implemented and evaluated on the measured deflection of actual three-layer pavement structure. Different from all previous traditional optimizers, the Bayesian optimizer cannot work with seed values. Rather, it needs a range (upper/lower bounds) of every single variable. Researchers constructed a native range from the seed values, as shown in Table 23. Note that the seed values in Table 23 are for informational purposes only and are not given to the optimizer.

Table 23. Initial Range and Optimization Performance of Bayesian Optimizer on Field-Measured Deflections of a Three-Layer Pavement Structure

Layer	Variable		Seed Value (*Not Used*)	Lower Bound	Upper Bound	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.9	–1.8	0	–1.4617
		alpha	4.5	0	9	5.4675
		betaPrime	–0.7	–1.4	0	–1.2872
		gamma	–0.4	–0.8	0	-0.053885
	Modulus at 17 Hz (ksi)	E1	356	0.016	31622	764.4
Base	Modulus (ksi)	E2	20	1	40	33.6
	Rayleigh Damping Coefficient β_R	BBase	0.003	0	0.006	0.0043194
Subgrade	Modulus (ksi)	E3	5	1	10	9.9
	Rayleigh Damping Coefficient β_R	BBase	0.003	0	0.006	0.0043194

The detailed optimization progress is shown in Figure 66. The Bayesian optimizer took around 450 function evaluations to optimize the RMSRE to 6.74 percent. This was not a good convergence compared with previous optimizers. However, it should be noted that all previous optimizers used seed values while the Bayesian did not. This means the Bayesian could have an advantage if seed values are not well-tuned or not given. Researchers plan to further evaluate this scenario.

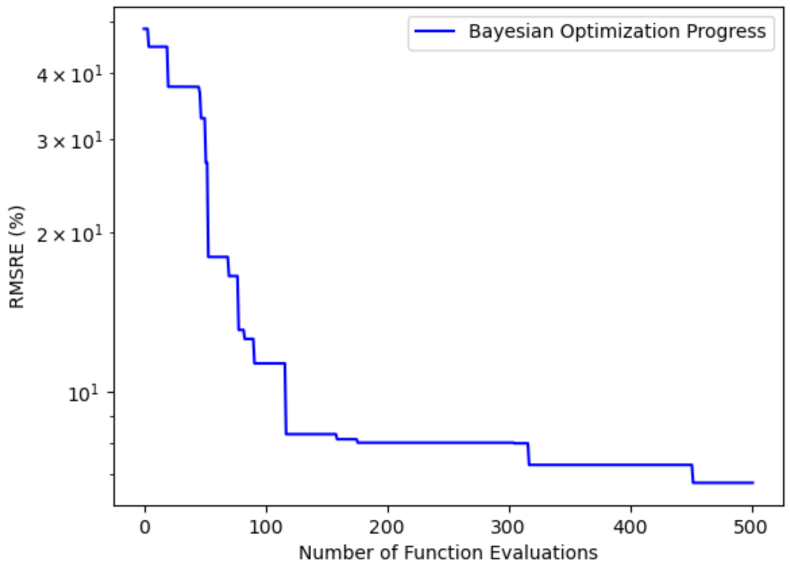


Figure 66. Optimization Progress of Bayesian Optimizer on Field-Measured Deflections of a Three-Layer Pavement Structure

The Bayesian optimizer was further evaluated on the synthetic three-layer pavement system. Similar to the setting of previous evaluation, the seed values were not used by the Bayesian optimizer, and these seeds were converted to the range instead as shown in Table 24. The optimization progress is shown in Figure 67, where the Bayesian converged to an RMSRE of 5 percent within around 60 evaluations and stayed at a similar level until the end. The final RMSRE was 3.85 percent. These results suggest that the Bayesian optimizer can quickly optimize the variables to a reasonably good level but fail to find the exact values for variables, which is a similar pattern to what was observed in the field-measured system evaluation.

Table 24. Initial Range and Optimization Performance of Bayesian Optimizer on a Synthetic Three-Layer Pavement Structure

Layer	Variable		Target Value	Seed Value (*Not Used*)	Lower Bound	Upper Bound	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.134	–1	–2	0	–1.99115
		alpha	3.703	4.65	0	9.3	4.96230
		betaPrime	–0.465118	–0.265118	–0.530236	0	–0.475363
		gamma	–0.548	–0.65	–1.3	0	–0.238817
	Modulus at 17 Hz (ksi)	E1	469	286	0.01	44668	25.0
Base	Modulus (ksi)	E2	40	50	0.001	100	66.8
	Rayleigh Damping Coefficient β_R	BBase	0.002	0.003	0	0.006	0.00340631
Subgrade	Modulus (ksi)	E3	5	7	0.001	14	5.7
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0	0.002	0.00180675

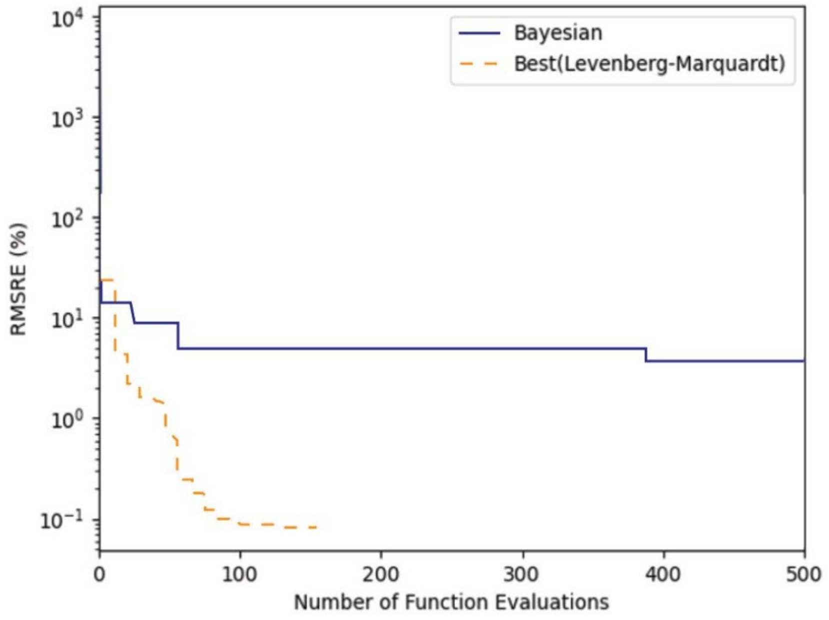


Figure 67. Optimization Progress of Bayesian Optimizer on a Synthetic Three-Layer Pavement Structure

5.7.3 Parametric Study

As discussed in Section 5.7.1, the choice of exploitation versus exploration strategies and its trade-offs could have a potential impact on the performance of the Bayesian optimizer. Thus, a parametric study was overperformed using different strategies and kappa values. Kappa values control how much the Bayesian optimizer would prefer exploration over exploitation.

The strategies tested were UCB, EI, and POI, and the kappa values tested were 1.0, 2.5, 5.0, and 10. The optimization progresses are shown in Figure 68, and the final results are shown in Table 25. The results indicate that while different strategies and kappa values do make the optimization different, they do not significantly change the final RMSRE.

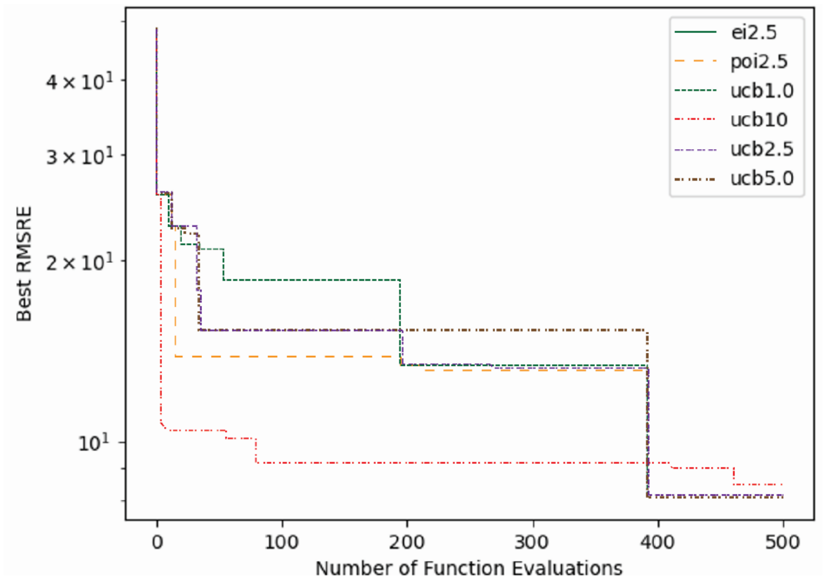


Figure 68. Optimization Progress of Parametric Study of the Bayesian Optimizer on Field-Measured Deflections of a Three-Layer Pavement Structure

Table 25. Parametric Study of the Bayesian Optimizer's Final RMSRE on Field-Measured Deflections of a Three-Layer Pavement Structure

Kappa	UCB	EI	POI
2.5	8.094%	8.094%	8.094%
1.0	8.094%		
5.0	8.094%		
10	8.501%		

Notably, these results are different from those in Section 5.7.2, which could be due to different random seed values. Most learning-based algorithms would give different results across different runs due to randomness, even if the target problem, the data, and the parameters were the same.

5.8 LEVENBERG–MARQUARDT ALGORITHM

In this effort, another family of optimization algorithms, the least-squares algorithms, was also evaluated. The term least-squares refers to the approach of minimizing the difference between the observed values and the predicted values by each individual equation of the target problem. This kind of approach naturally aligns with the vector problem definition (as discussed in Section 5.2.1), allowing for a potentially faster convergence.

The Levenberg-Marquardt (Moré, 1978) algorithm is a popular least-squares algorithm. It can be described as a combination of Newton's method and the gradient descent method. It is proposed to deal with Newton's method's drawbacks: the sensitivity to initial point (seed) and the requirement that Jacobian matrix must be invertible. Specifically, the Levenberg-Marquardt algorithm uses an adaptive parameter to control the interpolation between Newton's method and the gradient descent method. It acts more like a gradient descent method when the parameters are far from their optimal values and acts more like the Gauss-Newton method when the parameters are close to their optimal values.

5.8.1 Algorithm

The objective is to find a p, where $f(p) = x$ and x is the target value. Similar to Section 5.3.1, the Levenberg-Marquardt algorithm first uses the Taylor expansion:

$$f(\mathrm{p} + \delta_{\mathrm{p}}) \approx f(\mathrm{p}) + \mathrm{J}\delta_{\mathrm{p}} \tag{116}$$

Here the J is the Jacobian matrix. Then:

$$|\mathbf{x} - f(\mathbf{p} + \delta_{\mathbf{p},k})| \approx |\mathbf{x} - f(\mathbf{p}) - \mathbf{J}\delta_{\mathbf{p,k}}| = |\epsilon_k - \mathbf{J}\delta_{\mathbf{p,k}}| \tag{117}$$

The objective is to make $f(p)$ as close to x as possible. Thus, minimizing the above equation gives:

$$(\mathbf{J}^T\mathbf{J})\delta_{\mathrm{p}} = \mathbf{J}^T\epsilon_k \tag{118}$$

Levenberg's contribution is to replace the above equation by a "damped version":

$$[\mu\mathbf{I} + (\mathbf{J}^T\mathbf{J})]\delta_{\mathrm{p}} = \mathbf{J}^T\epsilon_k \tag{119}$$

The two terms on the left side of equation 119 give the solution of gradient descent (William, 1992) and Newton's methods (Björck, 1996), respectively. Thus, the parameter μ controls whether the solution is closer to gradient descent or Newton. For adaptive control, μ is adjusted at each iteration based on the reduction of distance between $f(p)$ and x. If the reduction is large, μ is decreased, which makes the solution closer to Newton's method and vice versa. In this way, the Levenberg-Marquardt algorithm takes advantage of the robustness of gradient descent when the current value is away from the optimal value and uses the fast and good convergence of the Newton's method as the value approaches the optimal.

5.8.2 Evaluation

Researchers evaluated the Levenberg-Marquardt algorithm with both the three-layer synthetic and field-measured FWD data, respectively. The Jacobian matrix is approximated by finite difference, as described in Section 5.3.3.

As shown in Figure 69, the Levenberg-Marquardt algorithm achieves a fast convergence on the three-layer synthetic structure. It uses less than 50 function evaluations to reach RMSRE of 1 percent and 85 evaluations to reach 0.1 percent RSMRE. The final RMSRE is 0.085 percent and the corresponding variables are shown in Table 26. Notably, the sigmoidal coefficients recovered by the Levenberg-Marquardt algorithm are more similar to the target values than previous optimization methods.

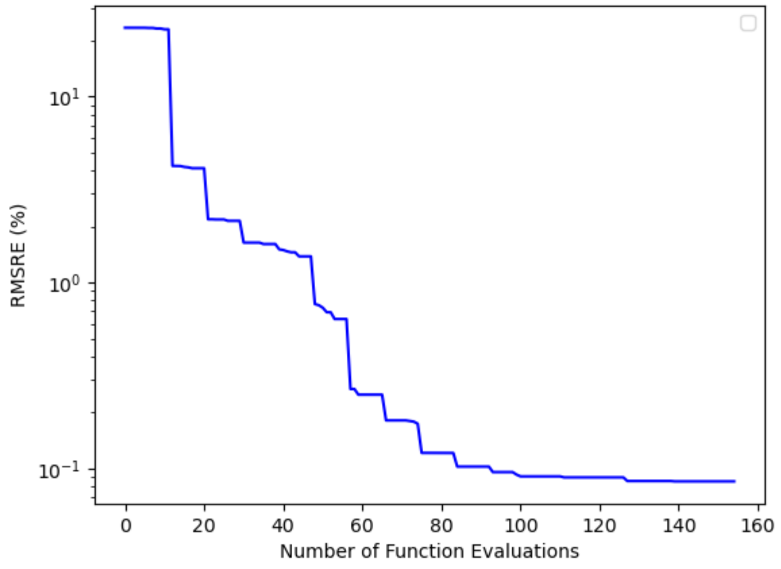


Figure 69. Optimization Progress of Levenberg-Marquardt on Deflections of Synthetic Three-Layer Pavement Structure

Table 26. Optimization Performance of Levenberg-Marquardt in the Synthetic Three-Layer Pavement Structure

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	Delta	–0.134	–1	-0.185169
	Sigmoidal coefficients	Alpha	3.703	4.65	3.70327
	Sigmoidal coefficients	betaPrime	–0.465118	– 0.265118	–0.622483
	Sigmoidal coefficients gamma	Gamma	–0.548	–0.65	–0.490351
	Modulus at 17 Hz (ksi)	E1	469	286	475.8

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
Base	Modulus (ksi)	E2	40	50	39.7
	Rayleigh Damping Coefficient β_R	BBase	0.002	0.003	0.00213204
Subgrade	Modulus (ksi)	E3	5	7	5.0
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0.00202223

For the field-measured FWD data, the Levenberg-Marquardt algorithm still achieved a relatively fast convergence by using less than 30 function evaluations to reduce the RMSRE to less than 1 percent. The optimization progress is shown in Figure 70 with a final RMSRE of 0.67 percent. The comparison between Levenberg-Marquardt and other optimizers is discussed in Section 5.14.

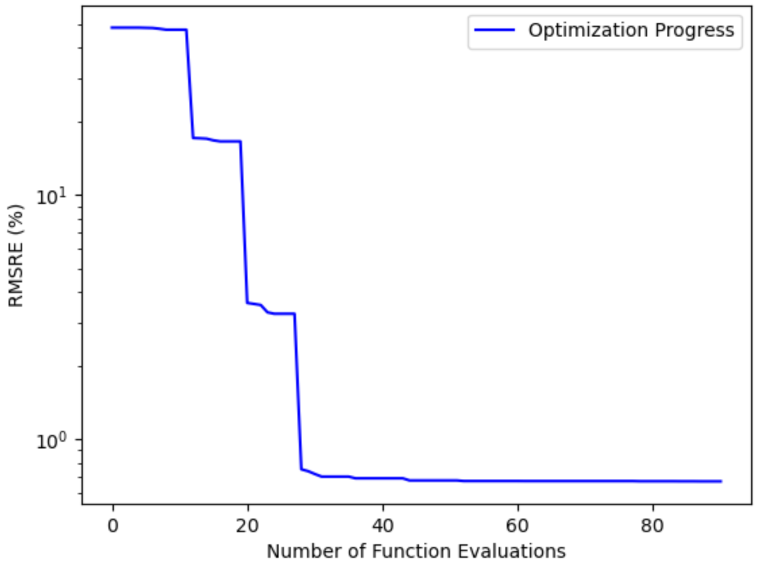


Figure 70. Optimization Progress of Levenberg-Marquardt on Deflections of Field-Measured Three-Layer Pavement Structure

Table 27. Optimization Performance of Levenberg-Marquardt on Field-Measured Deflections of a Three-Layer Pavement Structure

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	Delta	–0.9	–0.985555
	Sigmoidal coefficients	Alpha	4.5	4.40188
	Sigmoidal coefficients	betaPrime	–0.7	–0.942652
	Sigmoidal coefficients	Gamma	–0.4	–0.644878
	Modulus at 17 Hz (ksi)	E1	356	571.3
Base	Modulus (ksi)	E2	20	41.9
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.00228940
Subgrade	Modulus (ksi)	E3	5	13.1
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.00228940

Levenberg-Marquardt uses both Newton's and gradient descent methods, so, theoretically, it takes twice as many function evaluations. Yet the results show that the number of evaluations is comparable to the original Newton's method. This is because the caching mechanism implemented in the middle layer of the optimization framework (described in Section 5.1) helped to avoid all the repeated function evaluations during optimization. With the help of this mechanism, the Levenberg-Marquardt algorithm has shown promising results in terms of both the speed of convergence and the good final RMSRE.

5.9 TRUST REGION ALGORITHM

Inspired by the promising results of the Levenberg-Marquardt algorithm, researchers followed up with other least-squares algorithms. The trust region algorithm is another popular least-squares algorithm. Its concept is as follows: there are multiple ways to approximate the target function $f()$ at a given point x (e.g., Taylor expansion). These approximations are most accurate around the point x but not elsewhere. Thus, there is a trust region Δ around x, where one can safely trust the approximation and perform optimization on it. The trust region is like a dynamic boundary on the optimization. With such a boundary, the optimization is more stable and can avoid numerical problems like those observed in Section 5.4.2.

5.9.1 Algorithm

Let the target function be F and the approximated function be f.

$$g = F'(x),\ H = F''(x) \tag{120}$$

By the Taylor expansion, F can be approximated as:

$$f(x) = \frac{1}{2}x^T Hx + x^T g \tag{121}$$

Optimizing the approximated function gives:

$$s = argmin(f(x), x < \Delta) \tag{122}$$

where Δ is the trust region. If $F(x+s) < F(x)$, it means the current trust region is valid: $x = x + s$, increase Δ; or, it means the current trust region is too large, then decrease Δ. On the implementation level, there are some improvements. For example, instead of using a scaler as Δ, one could use the shape of the constraints as the shape of trust region. This is called the trust region reflective algorithm. It allows the users to have finer control over the trust region.

5.9.2 Evaluation

The trust region algorithm was evaluated on both the synthetic and field-measured, three-layer FWD data. On the synthetic pavement structure, the trust region algorithm used about 45 function evaluations to reach RMSRE of 1 percent and 120 evaluations to reach RMSRE below 0.1 percent, as shown in Figure 71. The recovered variables are shown in Table 28 and the final RMSRE is 0.098 percent. The comparison of trust region and other optimizers is in Section 5.14.

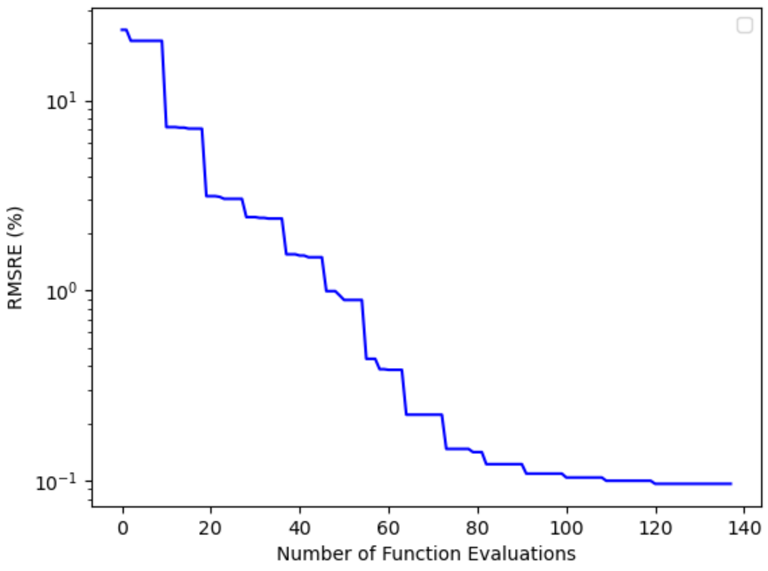


Figure 71. Optimization Progress of Trust Region on Deflections of Synthetic Three-Layer Pavement Structure

Table 28. Optimization Performance of Trust Region in the Synthetic Three-Layer Pavement Structure

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	Delta	−0.134	−1	0.953666
	Sigmoidal coefficients	Alpha	3.703	4.65	2.72996
	Sigmoidal coefficients	betaPrime	−0.465118	−0.265118	0.0806200
	Sigmoidal coefficients	Gamma	−0.548	−0.65	−0.503267
	Modulus at 17 Hz (ksi)	E1	469	286	476.0
Base	Modulus (ksi)	E2	40	50	39.7
	Rayleigh Damping Coefficient β_R	BBase	0.002	0.003	0.00214440
Subgrade	Modulus (ksi)	E3	5	7	5.0
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0.00200754

The convergence pattern was similar for the field-measured FWD data. The trust region algorithm used about 35 evaluations to reach RMSRE of 1 percent. The final RMSRE was 0.6938 percent, which was among the top-performing optimizers evaluated. The details are presented in Figure 72 and Table 29.

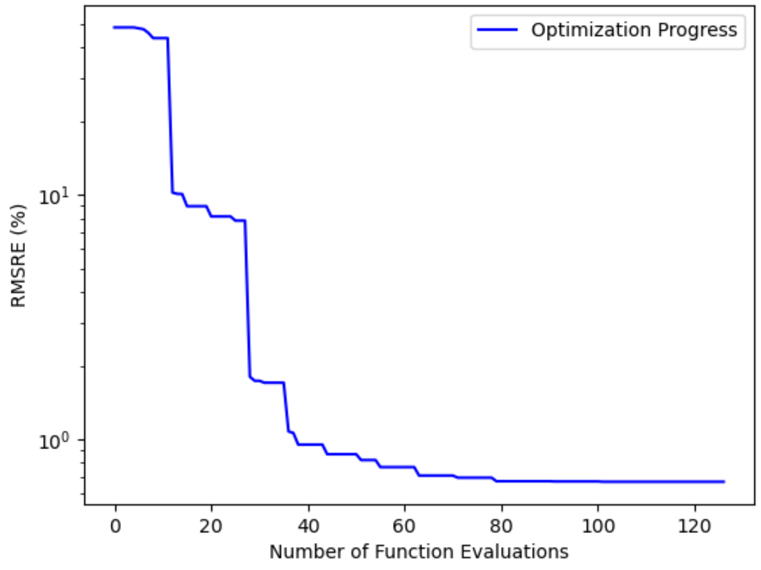


Figure 72. Optimization Progress of Trust Region on Deflections of Field-Measured Three-Layer Pavement Structure

Table 29. Optimization Performance of Trust Region on Field Measured Deflections of a Three-Layer Pavement Structure

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	Delta	–0.9	–0.422799
	Sigmoidal coefficients	Alpha	4.5	5.41625
	Sigmoidal coefficients	betaPrime	–0.7	–0.0632498
	Sigmoidal coefficients	Gamma	–0.4	–0.227358
	Modulus at 17 Hz (ksi)	E1	356	556.2
Base	Modulus (ksi)	E2	20	40
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.00237904
Subgrade	Modulus	E3	5	13
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.00237904

5.10 DOGLEG ALGORITHM

The Levenberg–Marquardt algorithm can be considered an enhancement to Newton's method by including the gradient descent algorithm to improve its robustness and convergence speed. The trust region algorithm can be considered an enhancement to Newton's method by making its step size adaptive. A natural question is whether it is possible to combine the two improvements. This combination is referred to as Powell's dogleg method. It combines the Newton's method and the gradient descent as the Levenberg–Marquardt algorithm and then limits the solution by the trust region.

5.10.1 Algorithm

The dogleg algorithm is a heuristic algorithm, as shown in Figure 73. If the current point is x, the algorithm needs to decide the next point x_{t+1}:

- **Step 1**—Get next point x_{gn} from Gauss-Newton method.
- **Step 2**—If x_{gn} is within trust region Δ: $x_{t+1} = x_{gn}$.
- **Step 3**—If x_{gn} is outside trust region Δ: get next point x_{st} from gradient descent method.
- **Step 4**—If x_{st} is outside trust region Δ: cap x_{st} to the boundary of Δ; $x_{t+1} = x_{st\_capped}$.
- **Step 5**—If x_{st} is within trust region Δ: $x_k = x_{gn} + x_{st}$. Cap x_k to the boundary of Δ: $x_{t+1} = x_{k\_capped}$.

In this way, the shape of the actual optimization step is like a dogleg hole in golf; hence, the algorithm's name.

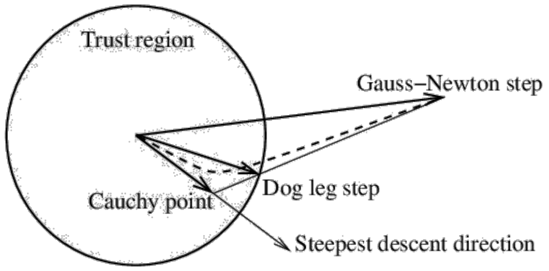


Figure 73. How Dogleg Algorithm Works (Lourakis & Argyros, 2005)

5.10.2 Evaluation

Similar to previous evaluations, the dogleg algorithm was evaluated with the synthetic and field-measured, three-layer system FWD data. On the synthetic data, the dogleg algorithm shows a fast convergence of less than 75 function evaluations to reach the RMSRE of 0.1 percent, as shown in Figure 74. The final RMSRE is 0.055 percent, which was the best result among all optimizers evaluated so far. The comparison of the dogleg and other optimizers is discussed in Section 5.14. The recovered variables are shown in Table 30.

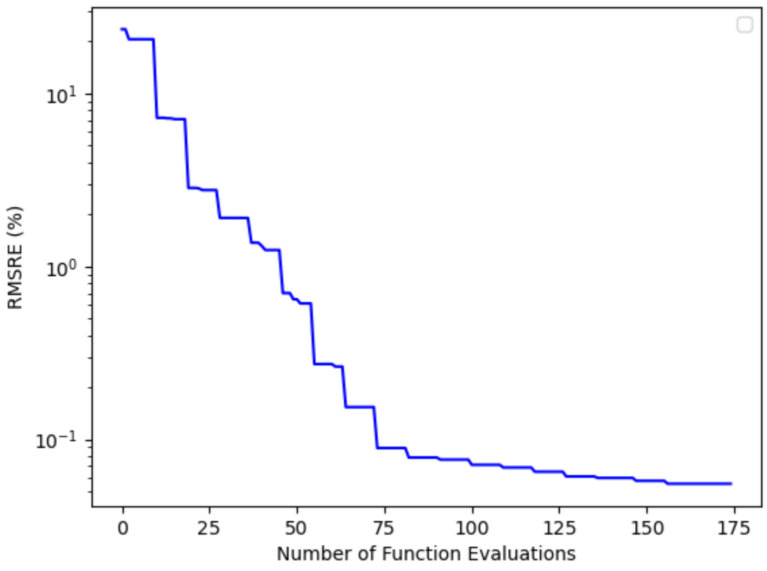


Figure 74. Optimization Progress of the Dogleg Algorithm on Deflections of Synthetic Three-Layer Pavement Structure

Table 30. Optimization Performance of the Dogleg in the Synthetic Three-Layer Pavement Structure

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	Delta	–0.134	–1	0.242861
	Sigmoidal coefficients	Alpha	3.703	4.65	3.3323
	Sigmoidal coefficients	betaPrime	–0.465118	–0.265118	–0.211929
	Sigmoidal coefficients	Gamma	–0.548	–0.65	–0.614591
	Modulus at 17 Hz (ksi)	E1	–	–	454.9
Base	Modulus (ksi)	E2	40	50	40.3
	Rayleigh Damping Coefficient β_R	BBase	0.002	0.003	0.00190876
Subgrade	Modulus	E3	5	7	4,9
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0.00198551

For the field-measured data, the dogleg algorithm took about 30 function evaluations to achieve RMSRE of 1 percent, and the final RMSRE was 0.6763 percent. The optimization progress is shown in Figure 75, and the recovered results are shown in Table 31.

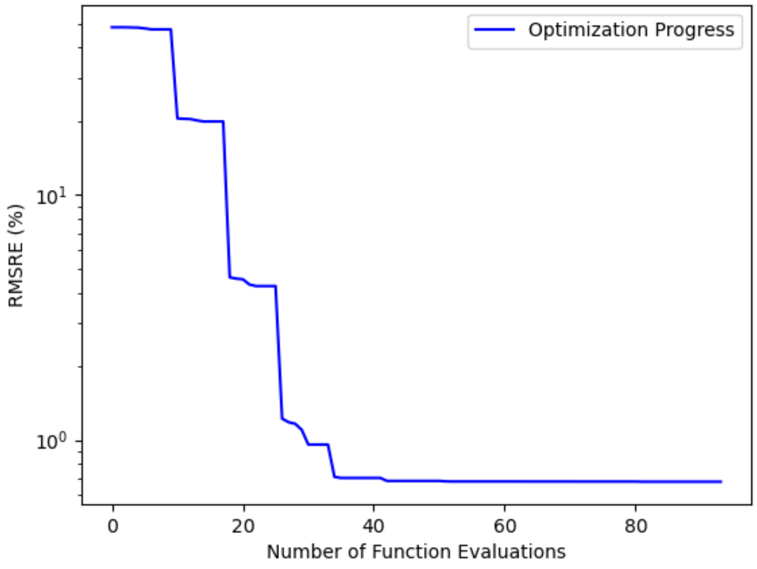


Figure 75. Optimization Progress of the Dogleg Algorithm on Deflections of Field-Measured Three-Layer Pavement Structure

Table 31. Optimization Performance of the Dogleg Algorithm on Field-Measured Deflections of a Three-Layer Pavement Structure

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal coefficients	delta	–0.9	–0.369329
	Sigmoidal coefficients	alpha	4.5	5.54109
	Sigmoidal coefficients	betaPrime	–0.7	0.0133356
	Sigmoidal coefficients	gamma	–0.4	–0.214682
	Modulus at 17 Hz (ksi)	E1	356	558.2
Base	Modulus (ksi)	E2	20	40
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.00234077
Subgrade	Modulus	E3	5	13
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.00234077

Similar to the Levenberg–Marquardt algorithm, the theoretical number of function evaluations needed was higher than the ones shown in Table 31. The caching mechanism reduced the number of function evaluations.

5.11 KALMAN FILTER

The Kalman filter is a widely used algorithm that estimates system states under noisy measurements and external disturbance. The key idea is to fit a joint probability distribution over the variables for each timeframe. This algorithm is widely used in industrial controllers such as guidance, navigation, and control of vehicles and aircraft. Several related works (Choi et al., 2010; Wu et al., 2021) proposed the use of the Kalman filter to estimate the layer modulus in backcalculation of AC structures. The challenge is that the target of this project is to estimate a wide range of variables including sigmoidal coefficients and Rayleigh damping coefficients instead of just the layer modulus. Thus, an enhanced implementation of the Kalman Filter was employed in this project to deal with the multidimensional parameters space-of-target problem.

5.11.1 Problem Definition

The Kalman filter estimates the system state at the current time step based on three factors: the previous system state, the current measurements (also called control signals), and noise:

$$x_k = Fx_{k-1} + Bu_{k-1} + w_{k-1} \tag{123}$$

where F is the state transition matrix, B is the control-input matrix, and u is the measurements, w is the noise assumed to be from a multidimensional zero-mean Gaussian distribution with covariance Q. The Kalman filter also assumes that current measurements are related to current system state by:

$$z_k = H\boldsymbol{x}_k + \boldsymbol{v}_k \tag{124}$$

where *H* is the measurement matrix, and *v* is the measurement noise that is also assumed to be from a multidimensional zero-mean Gaussian distribution with covariance *R*. Under this problem setting, any system in the Kalman filter could by defined by the five matrices *F, B, H, Q,* and *R.*

5.11.2 Algorithm

The Kalman filter can be seen as a two-stage algorithm. It first predicts the system state *x* and error covariance *P* based on information from the previous step:

$$\hat{x}_k^- = F\hat{x}_{k-1}^+ + Bu_{k-1} \tag{125}$$

$$P_k^- = FP_{k-1}^+ F^T + Q \tag{126}$$

The subscripts – and + indicate "predicted" and "refined" variables, respectively. The Kalman filter algorithm refines these predicted values based on the collected measurement information.

$$\widetilde{\boldsymbol{y}}_k = z_k - H\hat{x}_k^- \tag{127}$$

$$K_k = P_k^- H^T (R + HP_k^- H^T)^{-1} \tag{128}$$

$$\widehat{\boldsymbol{x}}_k^+ = \widehat{\boldsymbol{x}}_k^- + K_k \widetilde{\boldsymbol{y}} \tag{129}$$

$$P_k^+ = (I - K_k H)P_k^- \tag{130}$$

where *y* is the difference between predicted and actual measurements, called measurement residual. *K* is Kalman gain, which serves as a weight the algorithm uses to balance between the measurements and system state estimations. Figure 76 shows a high-level workflow of the described Kalman filter algorithm.

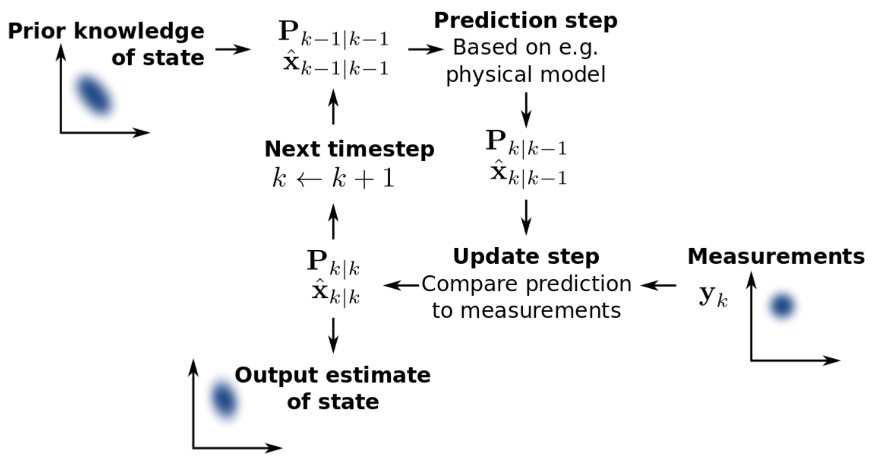

Prior knowledge of state
$\mathbf{P}_{k-1|k-1}$ $\hat{\mathbf{x}}_{k-1|k-1}$
Prediction step
Based on e.g. physical model
Next timestep
$k \leftarrow k+1$
$\mathbf{P}_{k|k-1}$ $\hat{\mathbf{x}}_{k|k-1}$
$\mathbf{P}_{k|k}$ $\hat{\mathbf{x}}_{k|k}$
Update step
Compare prediction to measurements
Measurements
$\mathbf{y}_k$
Output estimate of state

Figure 76. The Kalman Filter Workflow Showing how the Algorithm Estimates Current System States

The Kalman filter algorithm can be more intuitively understood by comparing it to a Newton's method algorithm with a finite difference and a fixed step size of 1. In that scenario, the measurement residual is similar to the derivative, and the system-state estimation update can be considered a damped version of the Newton's method updated equation.

5.11.3 Problem Formulation

The Kalman filter algorithm could not be directly applied to the target problem as it holds several assumptions that are not applicable to the backcalculation problem. Thus, the Kalman filter was tailored to fit the target problem.

The first challenge was to formulate the backcalculation problem into a form that the Kalman filter could work on. The Kalman filter expects a system with varying states and external disturbances, yet the target backcalculation problem is static. Thus, variables were used as the system state, and the parameters were used as the measurement, as suggested by the related work. This seemed reasonable because the Kalman filter takes the current shape of deflections (i.e., the parameters) as the observation and then uses it to update its estimation to the backcalculated variables. However, this approach did not work in practice, and all optimization processes were either diverged or stuck at the starting point even after non-trivial engineering/tuning effort. Based on related work research, researchers realized that the Kalman filter expects the current measurement to be correlated with target system states rather than current estimated states. For a specific backcalculation problem, the target variables are always fixed. That means the Kalman filter expects the same measurement throughout the optimization process. Yet that leads to a critical problem: if the only inputs to Kalman filter are the same measurement and a static Gaussian noise, then it will not be able to do any optimization since it is not getting any feedback of its estimations.

5.11.4 Extended Kalman Filter

Given the analysis discussed in Section 5.11.3, researchers confirmed the reason the optimization failed. Yet the expectation of correlation between observation and target is a fundamental part of the Kalman filter. To have the Kalman filter work with the backcalculation problem, researchers chose to use the Extended Kalman Filter (EKF) as a workaround. The key difference between the EKF and the Kalman Filter is that the Kalman Filter assumes there exists a static (user-defined) correlation H between current measurement and target system states (see 5.11.2), whereas the EKF approximates this measurement matrix H by using Jacobian matrix. Thus, the EKF can work with the backcalculation problem even if the measurements are static. Specifically, researchers set the state transition matrix F to identity matrix to reflect the static nature of the backcalculation problem. Choi,Wu, Pestana, and Harvey (2010) suggests some values for the uncertainty matrices Q and R, but researchers evaluated those values and found that they did not work well in practice. Thus, these matrices are set to identity matrices since the backcalculation problem is deterministic and, thus, there is no uncertainty. In each step, the parameters of the target variables/measured deflections will be fed as the measurement z. In the meantime, the Hessian matrix of the current estimation is computed and used as the measurement matrix H.

5.11.5 Evaluation

With the EKF algorithm applied and using the Hessian matrix as measurement matrix H, the performance of the optimizer was evaluated on the synthetic three-layer pavement structure used in previous evaluations. The optimization process is shown in Figure 77. The EKF was able to get a final RMSRE of 0.291 percent with about 100 function evaluations. Although the EKF showed convergence speeds and final RMSRE comparable to other optimizers, it seemed to be more sensitive to the seed values. This could be a potential disadvantage of this optimizer.

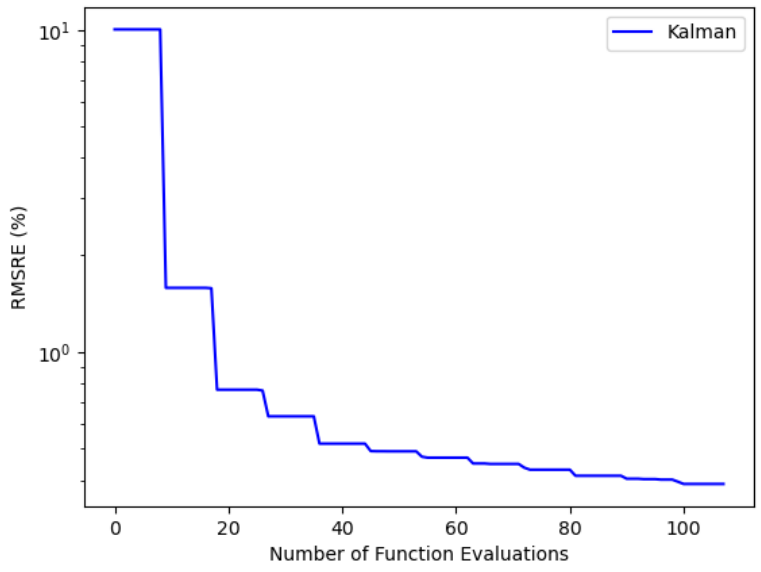


Figure 77. Optimization Progress of the EKF on Deflections of Synthetic Three-Layer Pavement Structure

Table 32. Optimization Performance of the EKF on Synthetic Deflections of a Three-Layer Pavement Structure

Layer	Variable		Target Value	Seed Value	Recovered Value by Optimizer
AC	Sigmoidal Coefficients	Delta	–0.134	–1	–0.927409
	Sigmoidal Coefficients	Alpha	3.703	4.65	4.761513
	Sigmoidal Coefficients	betaPrime	–0.465118	–0.265118	–0.277922
	Sigmoidal Coefficients	Gamma	–0.548	–0.65	–0.631047
	Modulus at 17 Hz (ksi)	E1	469	286	401.6
Base	Modulus (ksi)	E2	40	50,000	41.3
	Rayleigh Damping Coefficient β_R	BBase	0.002	0.003	0.00163970
Subgrade	Modulus	E3	5	7	4.9
	Rayleigh Damping Coefficient β_R	BSG	0.002	0.001	0.00187731

In terms of variables recovery, the EKF performed similarly to other derivative-based optimizers. It was able to recover the modulus and the Rayleigh damping coefficient reasonably well but missed the AC sigmoidal coefficients. It should be noted that this project is the first study that successfully applied the Kalman Filter to such a large group of variables. Previous work in this line of research only applied Kalman filter to modulus (Choi et al., 2010).

5.12 REINFORCEMENT LEARNING OPTIMIZER

Reinforcement Learning is a powerful tool for general optimization problems that dates back to the 1990s. In recent years, researchers combined classical reinforcement learning with the neural network into an even more powerful approach called deep-reinforcement learning, which is the technique being evaluated for the target problem (Sutton & Barto, 2018; Jaeger & Geiger, 2023).

The core concept of reinforcement learning treats the algorithm itself as an agent that interacts with the environment with certain actions and learns from the consequences of those actions. Specifically, reinforcement learning algorithms usually require three types of information:

- "state"—the description of all information about the current environment
- "action"—the available actions the algorithm can take
- "reward"—a quantitative description of the consequences of taking the action in current state

The workflow is straightforward: for the current state, take the action that maximizes expected rewards.

The advantages of reinforcement learning include its ability to tackle unknown problems iteratively by learning and its high flexibility as the state/action/reward are all user-defined.

Reinforcement learning has had notable achievements in several challenging real-world scenarios, such as recently winning against a top-rated human player in the game, “Go.”

There are different types of reinforcement learning algorithms. They can be roughly categorized into value-based, policy-based, actor-critic reinforcement learning algorithms, and others. Value-based algorithms, such as the Q-Learning function, try to accurately predict the value (reward) of each action in current situation (state). Policy-based algorithms try to iteratively improve an implicit policy (usually a neural network) so it could directly decide the next action to take without the need to know the exact value. Actor-critic algorithms are similar to a combination of the former two approaches. They work by having a policy based on predicted values and then update both the policy and the prediction of values in each step.

5.12.1 Problem Definition

In this study, the popular Q-Learning algorithm was first selected as the optimizer for the target problem. As previously discussed, all “states,” “actions,” and “rewards” need to be defined so the Q-Learning algorithm can work.

There are several possible ways to define the “state,” which is the description of current problem. First, an extensive description of the pavement structure can be used as the state. For example, a state can be described as a predefined, long vector covering each layer in the pavement structure and all the parameters. This approach would make the algorithm universal as long as the target problem could be described. However, this would make the state space infinitely large (number of possible values to the power of vector length). Thus, the algorithm would take orders of magnitude more data to train. An alternative way to define a state is to set it as empty. This is feasible because the target pavement structure is static for a certain optimization. In this case, there is no need for the algorithm to know the details of this pavement structure, and it only needs to know that the pavement structure is going to be consistent across the optimization. This significantly reduces the training time of the algorithm and makes the training much smoother as all the data are based on the same pavement structure. This also makes sure the algorithm is able to handle every optimization problem, as there is no need to fit the description of the pavement structure into the predefined vector. The disadvantage of this approach is it needs to start optimization from scratch every time as it cannot use the data from previous runs.

It is infeasible to directly use the variables as “actions,” which are possible actions the algorithm could take, for two reasons. First, the actions are discrete, whereas the target problem needs very fine-grained, continuous variables. Second, there could be many variables, and the number of combinations is large. To get around this issue, in this study, the actions were set to be the portion of change based on current variables. For example, the algorithm picks from the list (-5 percent, -1 percent, 0, +1 percent, +5 percent) and then applies it to the current variable. In this way, the actions are fixed so the algorithm will not be confused, and it enables arbitrary fine-grained change to the variables when multiple changes are combined together (e.g., two actions combined together 0.99*1.01 = 0.9999 could reduce the variable by 0.01 percent). The only disadvantage is that a positive variable cannot be turned into a negative one and vice versa. This means the algorithm requires seed values for the variables.

The “reward” part is very straightforward as the RMSRE metric is a natural inverse reward. Researchers use the -1*RMSRE as the reward. Thus, the algorithm aims to reduce RMSRE when pursuing larger rewards.

5.12.2 Algorithm

With the state/action/reward being properly defined, the algorithm of Q-Learning can be summarized as follows:

- Initialization: neural network *N*.
- Step 1: Fetch the current state *s*.
- Step 2: For every possible action *a*, get the predicted value *N(s,a)* from neural network.
- Step 3: Decide the next action *a*, based on predicted values.
- Step 4: Execute action *a*, and observe the consequences to get reward *r*.
- Step 5: Add the actual value *V(s,a) = r* to the training set of neural network *N*.
- Step 6: Train the neural network. Go to step 1.

Step 3 is a key component. The way algorithm decides its next action, called policy, needs to be chosen carefully. A native, or greedy, approach is to always choose the action with the best predicted values. Because the problem definition uses a static state, the greedy approach would result in the optimizer choosing the same action throughout the optimization. On the other hand, if the action is chosen randomly, then the optimization degenerates to random search, which is inefficient given the large search space the target problem has, as discussed previously. Thus, a good policy needs to consider the predicted values while having enough randomness. This is called the exploit-exploration balance, which is a common challenge for reinforcement learning algorithms. There are several approaches to achieve this balance. For example, the algorithm could randomly choose from the top N best actions so that balance is achieved. Also, the algorithm could assign a certain probability to every action based on their predicted value and then sample one action based on the assigned probability, which is called the Boltzmann distribution. Neither of these methods were selected in this study because they both introduce extra hyperparameters that require tuning effort, which limits the generalization and robustness of the optimizer. The final policy for the reinforcement learning optimizer was set as gradient descent with random start. In this policy, the optimizer randomly starts from an action and then repeatedly tries to move to another action in the neighborhood if that action has better predicted values. This policy has all the merits discussed.

5.12.3 Input Proposal

Section 5.12.1 discussed the problem setting for reinforcement learning algorithms to function and the related challenges. With the current workaround applied, the problem setting was capable of handling continuous variables with different orders of magnitude scales. The way the reinforcement learning optimizer chose its next action was further improved by fixing these three key disadvantages:

- Local optimality. The actions currently used are the portion of changes based on current variables. That means the next action would always be in the neighborhood of the current variables. Once the optimization hits a local optimal, it is difficult to get out of it in this manner.

- Scalability. Every action is a change to a certain variable. When the number of variables increases, the number of actions to update the variables grows exponentially.
- Quadrant. The proportional change makes the variables unable to cross quadrant.

To address the disadvantages, a key feature of neural networks, called the differentiability, is used. All neural networks have to be differentiable so that they can use a gradient-based approach in training. Such differentiability means it is possible to analytically compute the partial derivative of the output of neural network (the predicted value of action) with respect to the input (action). The best part of this approach is that most modern machine-learning frameworks have automatic differentiation engines so that the proposed partial derivative can be computed automatically during runtime without the need for user inputs, as shown in Figure 78.

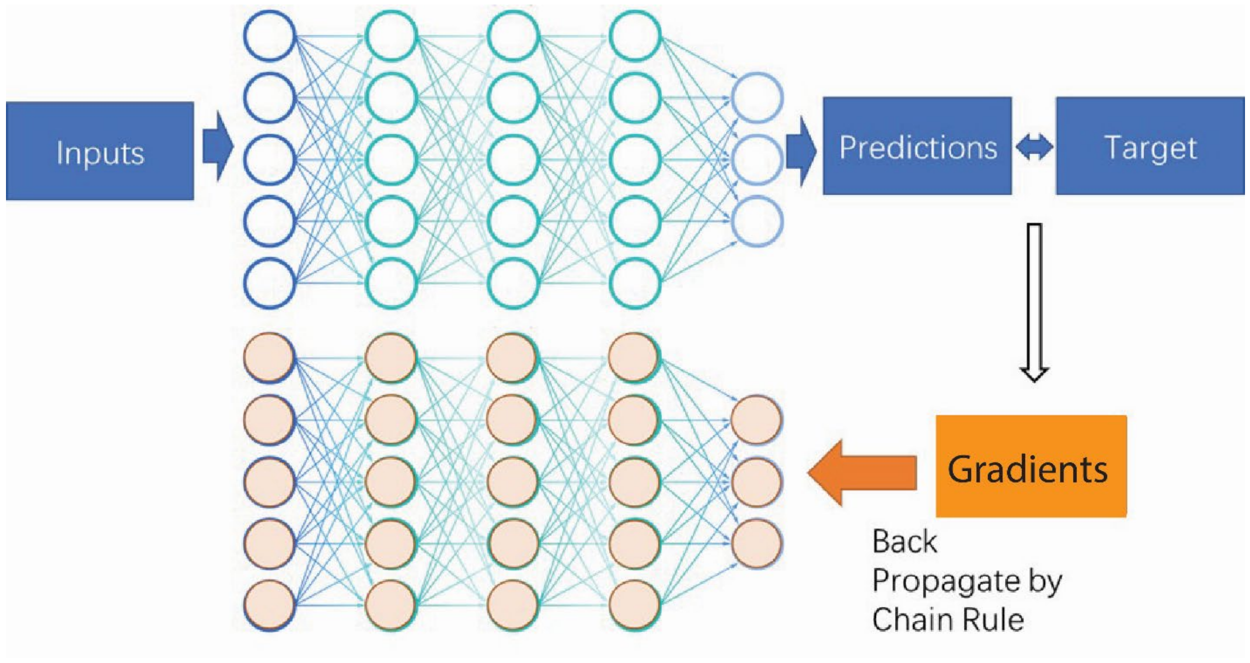


Figure 78. How a Neural Network Analytically Computes its Partial Derivative Automatically

Specifically, the neural network acts as a mapping between the input *a*, network weights *w*, and the output *v*.

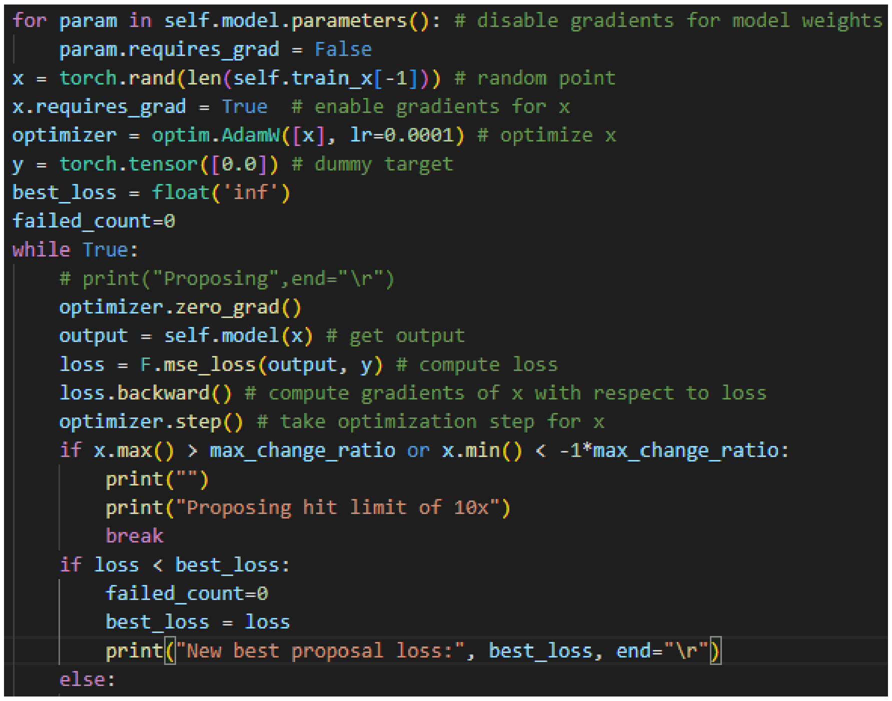
$$F(a,w) = v \tag{131}$$

This means for a specific combination of input *a* and network weights *w*, the output *v* is determined. The target is to find the action a' that maximizes the output.

$$a' = argmin\big(F(a,w)\big) = a - s\frac{\partial v}{\partial a} \tag{132}$$

Here, $\frac{\partial v}{\partial a}$ is the partial derivative, s is the step size, and a is current action. In practice, the updated equation is more sophisticated and includes momentum and approximation of second derivatives.

In this way, the optimizer can computationally get the action that the neural network believes to be best. In this study, this technique is called "input proposal." Because the neural network is randomly initialized and the starting point of action is also random, the optimization has enough randomness to keep a good exploit-exploration balance, as discussed in Section 5.12.2. For this study, a safeguard was implemented to limit the range of update of the action to be within 10 times of current action. This was to prevent the neural network from giving unrealistic suggestions, which is possible at the beginning of training when the neural network does not have considerable knowledge of the target problem. The high-level workflow of the input proposal in shown in Figure 79.



```
for param in self.model.parameters(): # disable gradients for model weights
    param.requires_grad = False
x = torch.rand(len(self.train_x[-1])) # random point
x.requires_grad = True  # enable gradients for x
optimizer = optim.AdamW([x], lr=0.0001) # optimize x
y = torch.tensor([0.0]) # dummy target
best_loss = float('inf')
failed_count=0
while True:
    # print("Proposing",end="\r")
    optimizer.zero_grad()
    output = self.model(x) # get output
    loss = F.mse_loss(output, y) # compute loss
    loss.backward() # compute gradients of x with respect to loss
    optimizer.step() # take optimization step for x
    if x.max() > max_change_ratio or x.min() < -1*max_change_ratio:
        print("")
        print("Proposing hit limit of 10x")
        break
    if loss < best_loss:
        failed_count=0
        best_loss = loss
        print("New best proposal loss:", best_loss, end="\r")
    else:
```

Figure 79. High-Level Workflow of How Input Proposal Technique is Implemented

5.12.4 Evaluation

Like previous optimizers, the reinforcement learning optimizer was evaluated on both the synthetic three-layer structure and the field-measured data. The optimization progress on the synthetic three-layer structure is shown in Figure 80. The final RMSRE was 1.24 percent.

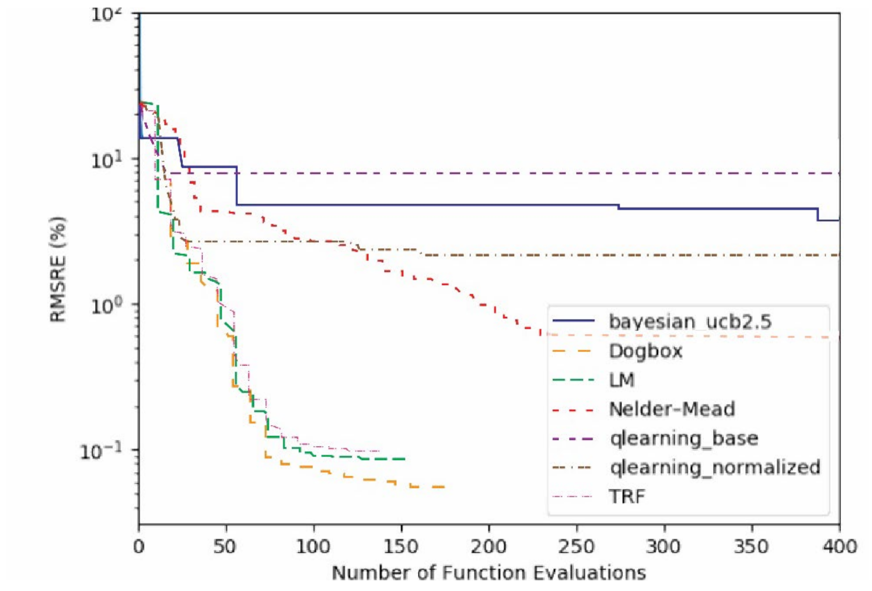


Figure 80. Optimization Progress of Reinforcement Learning Optimizer on the Synthetic Three-Layer Structure

The performance of the reinforcement learning optimizer was further evaluated on the field-measured data. The recovered variables are shown in Table 33. The final RMSRE was 1.1259 percent. This is comparable with traditional optimizers.

Table 33. Optimization Performance of Q Learning Optimizer on Field-Measured Deflections

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal Coefficients	Delta	–0.9	–0.45020
	Sigmoidal Coefficients	Alpha	4.5	5.81494
	Sigmoidal Coefficients	betaPrime	–0.7	–0.09034
	Sigmoidal Coefficients	Gamma	–0.4	–0.16938
	Modulus at 17 Hz (ksi)	E1	356	770.1
Base	Modulus (ksi)	E2	20	28.1
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.0024465
Subgrade	Modulus (ksi)	E3	5	13.4
	Rayleigh Damping Coefficient β_R	BBase	0.003	0.0024465

5.13 ENSEMBLE LEARNING

To date , several popular optimization methods have been implemented and evaluated. These methods have shown strong heterogeneity in terms of robustness, convergence speed, and performance. A natural question is whether these optimizers should be combined together to get the best of them all. This is possible if the framework was carefully designed to handle the combination of optimizations. This technique is called ensemble learning, which entails having an optimizer start from where a previous optimizer ends. Ensemble learning is frequently used for classification problems, as shown in Figure 81, and it is a great match for the backcalculation optimization problem as a variety of optimization methods are developed.

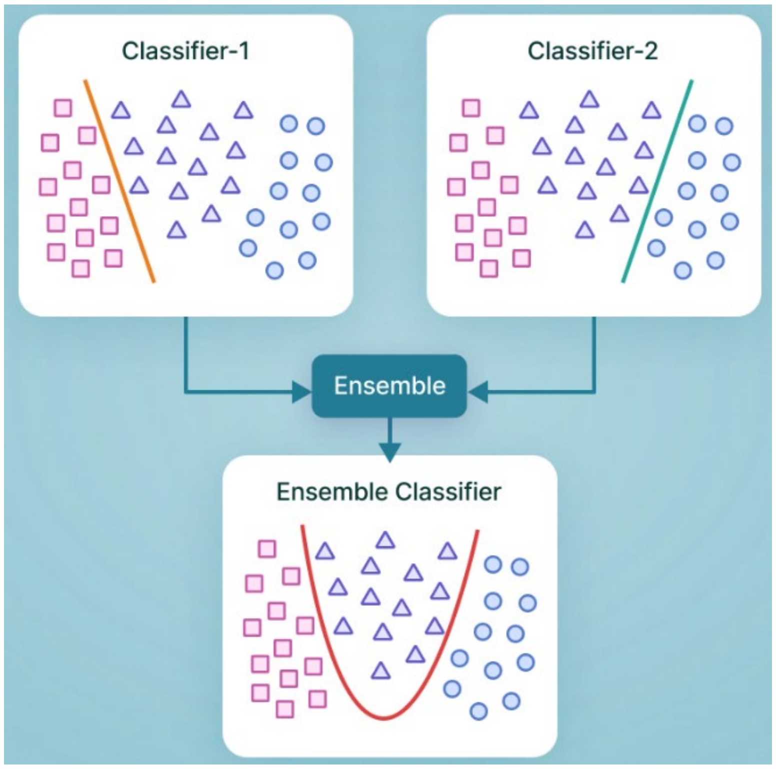


Figure 81. Ensemble Learning by V7aLab (Kundu, 2022)

5.13.1 Framework Update

The optimization framework (Figure 82) is designed at the beginning of the optimization efforts with several powerful features to support the smooth and highly efficient usage of optimizers. However, multiple optimizers working together is not part of expected use cases for the optimization framework. Thus, the framework is patched in aspects described in Sections 5.13.1.1 through 5.13.3.

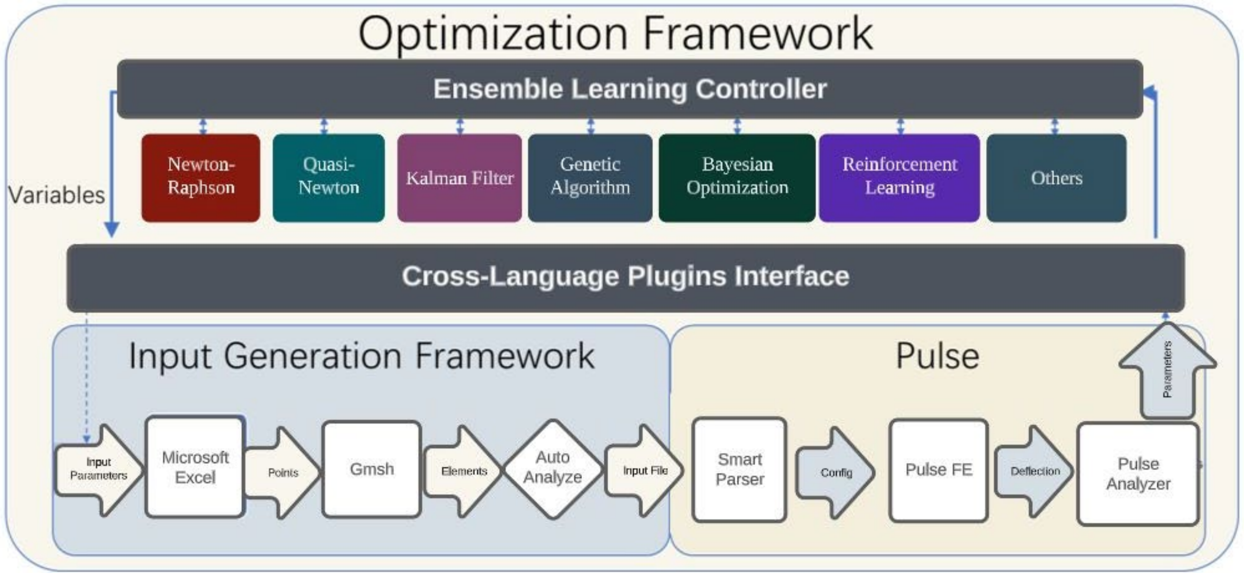


Figure 82. Optimization Framework Overview with Ensemble Learning Controller

5.13.1.1 Ensemble Learning Controller

In the original design of the optimization framework, the optimization control is entrusted to the optimizer. This includes where to try the next set of variables, when the optimization is considered finished, etc. For a multi-optimizer case, optimization fails if there are multiple control flows at the same time. Thus, a unified controller is implemented for the optimization workflow. The updated control workflow and framework is shown in Figure 82. This is a significant engineering effort as the optimizer needs to be refactored in a three-fold way. First, the control part and the optimization part need to be separated. Next, the optimization part needs to be tailored to fit the API of the controller. Finally, the optimizer needs to be modularized and self-contained, which means it takes care of all its intermediate states and variables so they do not get confused among optimizers and can be easily used by the controller. These three requirements mean the optimizer has to be completely reworked, which adds up to a non-trivial effort. The good news is that such effort is well-compensated as the reworked optimizer not only enables ensemble learning but also enables many sophisticated controls that benefit the overall optimization performance.

A key example is the adaptive stopping of optimization, which is related to a simple yet critical question of when the optimization should be considered finished. Two commonly used criteria are when a certain number of iterations are done and when the results do not change much in the last certain iterations. In this case, those two criteria are not applicable. For the first criterion, large numbers of iterations would prolong the optimization without improving the results, whereas insufficient iterations would lead to suboptimal results. For the second criterion, many optimizers would jump around the optimal point even if it converges; so, this criterion does not apply in this case. With a unified controller, this dilemma can be easily alleviated by using adaptive stopping. Adaptive stopping is a progress-based criterion where each optimizer starts with certain initial budget/patience, and the budget/patience would be restored every time the optimization progresses (i.e., achieves a new best RMSRE). In this manner, the optimization would run indefinitely as long as the optimizer is still improving the results and would end when the optimization converges.

5.13.1.2 Cache Sharing

Caching is a key mechanism in the optimization framework that greatly reduces the number of actual *PULSE*_FE calculations, especially for the classic derivative-based optimizers. In previous optimization workflows, optimizers report to the cache, yet with different standards. For example, some optimizers take RMSRE larger than 100 percent as a failure, whereas others use 300 percent or an infinitely large number. In the updated framework, the cache has a hierarchy. The system/framework cache is global, unique, and directly connected to the *PULSE*_FE so that only the raw results are recorded. The optimizer cache is optimizer dependent and is periodically synchronized with the system cache. In this manner, the *PULSE*_FE calculation will always be recorded and available to all optimizers, no matter which optimizer invoked it. Also, the optimizer can still have independent standards for the cache, such as failure.

5.13.1.3 Automatic Error Recovery

Note that the idea of ensemble learning is to have another optimizer take over where previous optimization ends. Yet a very common reason for the optimization to end is the optimizer hitting a dead-end or illegal point. For most non-learning-based optimizers, starting with a dead-end or illegal point simply leads to failed optimization. To take on this challenge, an integrity check should be implemented to detect if the current point is a bad starting point when switching optimizers.

If the current point is detected to be a bad starting point, it should be determined how the framework would pick a good starting point instead. A straightforward solution is to use a point from the optimization history, yet there is a dilemma. On the one hand, an ideal starting point should be far away from the current bad point, so the optimization does not fall into this bad point again. On the other hand, an ideal point should be close to where the previous optimization ends so the framework does not waste time redoing the same optimization. Researchers addressed this challenge by using the Maxwell–Boltzmann distribution to pick a safe point according to a probabilistic distribution based on RMSRE instead of distance. The Maxwell–Boltzmann distribution originates from the thermodynamic theory that describes the distribution of speeds among the particles in a sample of gas at a given temperature. The Maxwell–Boltzmann distribution can be intuitively understood as:

$$prob(a) = \frac{\exp(a)}{\sum \exp(A)},\ a \in A \tag{133}$$

The chance of an item with value *a* being picked is equal to the exponential value of *a* over the sum of exponential value of all the items. For backcalculation problem, the RMSRE could vary with different orders of magnitude and guaranteed to be non-negative. Thus, a simpler and more feasible form of Maxwell–Boltzmann distribution is used:

$$prob(a) = \frac{\frac{1}{a}}{\sum(\frac{1}{A})},\ a \in A \tag{134}$$

In this way, any failed optimization would be recovered to a random point with good RMSRE for the next optimizer to start from, as shown in Figure 83.

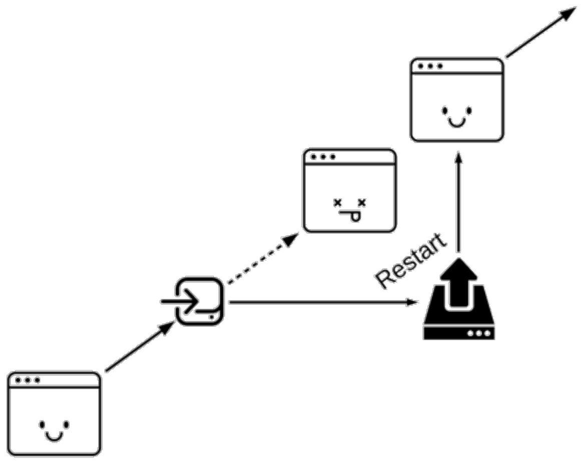


Figure 83. Automatic Error Recovery Starts Next Optimizer When Previous One Fails

5.13.2 Evaluation

The performance of ensemble learning is assessed by combining the Newton's method optimizer and reinforcement learning, using the field-measured data. These data originate from the LTPP database for test section 01-0101 in Alabama, where the pavement section consists of a 7.4-inch AC layer and a 7.9-inch unbound granular base layer constructed over untreated subgrade material. The FWD data used in this study were collected on March 11, 1998, with a surface temperature of 87 °F.

In this ensemble setting, Newton's method optimizer and reinforcement learning were employed interchangeably with a patience threshold of three. This means that when one optimizer fails to show progress in three consecutive iterations, it yields control to the other optimizer. The overall optimization progress is assessed by RMSRE in relation to the total number of Pulse calculations, as depicted in Figure 84.

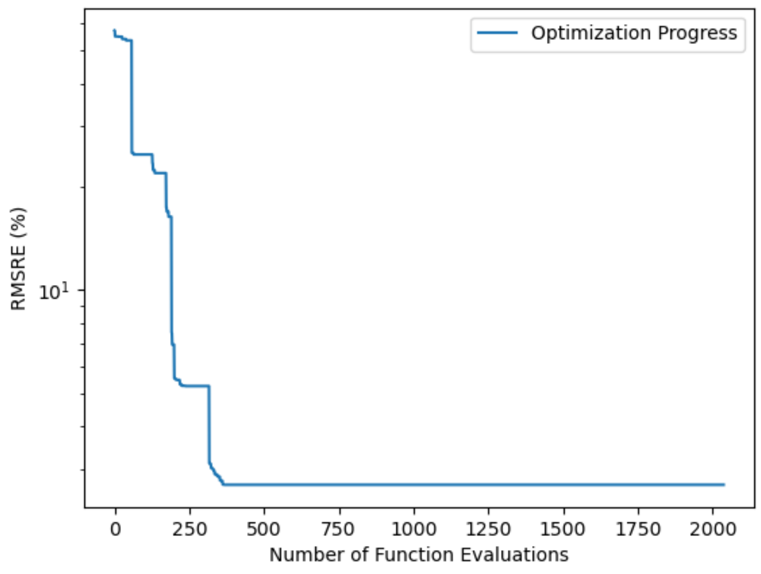


Figure 84. Ensemble Learning's Optimization Progress Using Alabama Field-Measured Data

The final RMSRE is 2.705 percent, and the recovered variables are shown in Table 34. If the optimizers are applied alone, then the Newton's method optimizer converges at 50 percent RMSRE as the seed variables are not close enough to the optimal points, and the reinforcement learning optimizer converges at around 10–20 percent RMSRE. Yet when combined by the ensemble learning, these two optimizers with the interleaving process get the RMSRE to as low as 2.7 percent, which demonstrates the potential of ensemble learning.

Table 34. Optimization Performance of Ensemble Learning on Alabama Field-Measured Data

Layer	Variable		Seed Value	Recovered Value by Optimizer
AC	Sigmoidal Coefficients	delta	–0.9	–0.2677
	Sigmoidal Coefficients	alpha	4.5	3.9329
	Sigmoidal Coefficients	betaPrime	–0.7	–0.0824
	Sigmoidal Coefficients	gamma	–0.4	–0.6468
	Modulus at 17 Hz (ksi)	E1	356	324.1
Base	Modulus (ksi)	E2	20	6.9
	Rayleigh Damping Coefficient, α_R (1/s)	alphaR	0.003	–46.6318
Subgrade	Modulus (ksi)	E3	5	33.3
	Rayleigh Damping Coefficient, β_R (1/s)	betaR	0.003	–5.973e–05

5.14 CROSS-COMPARISON OF OPTIMIZATION METHODS

While several of the evaluated optimizers were able to optimize the target problem reasonably well, there is a need to evaluate the optimizers more extensively in terms of their recovered master curves, robustness, etc. This section presents a cross comparison of optimization methods implemented so far to give a better understanding of their relative strengths and weaknesses.

5.14.1 Comparison of Convergence Speed

In the earlier evaluations, the iteration number was used as the metric of convergence speed. It is a common metric and works well under the same optimizer. Yet as there are heterogeneous optimizers in this study, the iteration number is no longer a good choice for comparison. For example, the Powell's method optimizer uses dozens of function evaluations for each iteration whereas the Nelder–Mead optimizer uses about three function evaluations each iteration. Thus, researchers decided to use the number of function evaluations (i.e., the number of calls to FE model) as the metric for comparison and rerun the earlier evaluation based on the new metric. The results are shown in Figure 85.

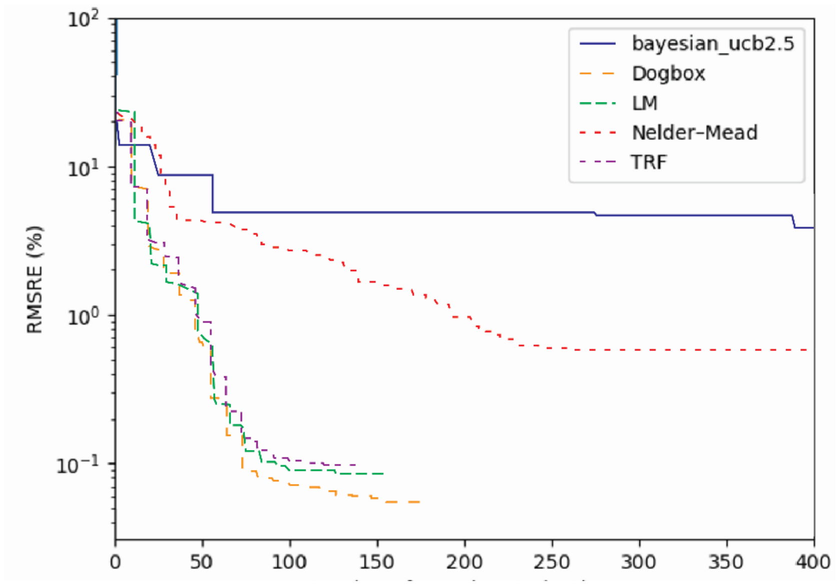


Figure 85. Comparison of Optimizers' Convergence on Synthetic Three-Layer Pavement Structure (The y-axis is log-scale.)

5.14.2 Recovered Variables for Synthetic Three-Layer System

Table 35 shows all recovered variables from the optimizers in an aggregated view. It can be concluded that the success rate of recovering variables depends on variable types. Most optimizers can find the target moduli for layers, whereas only some optimizers can find the right Rayleigh damping coefficients. For the sigmoidal coefficients, further evaluation is needed as there could be multiple sets of sigmoidal coefficients that all show a similar pattern in a certain range of the master curve (certain range of frequencies or time).

Table 35. Recovered Variables from All Optimizers for Synthetic Three-Layer System

Optimizer	delta	alpha	betaPrime	gamma	E1 at 17 Hz (ksi)	E2 (ksi)	BBase	E3 (ksi)	BSG
Target Values	–0.13400	3.7030	–0.46512	–0.54800	469.12	40.000	0.002000	5.000	0.002000
Newton	–0.92000	4.9410	–0.46856	–0.39800	447.96	40.229	0.001930	5.000	0.001970
BFGS	–0.92500	4.6820	–0.28900	–0.69700	424.76	39.882	0.001500	4.959	0.001980
L–BFGS–B	–1.0080	4.5200	–0.26200	–0.64700	222.49	49.881	0.003010	5.101	0.001000
Powell	–1.3160	4.4510	–0.89700	–1.15100	542.34	39.089	0.001960	4.999	0.001980
Nelder–Mead	–0.00704	5.3218	–0.00271	–0.01290	477.22	38.316	0.003010	5.022	0.002160
Bayesian	–1.9911	4.9623	–0.47536	–0.23882	25.11	66.877	0.003406	5.773	0.001807
Levenberg–Marquardt	–0.18517	3.7032	–0.62250	–0.49035	476.20	39.786	0.002132	5.003	0.002022
Trust Region	0.95367	2.7299	0.08060	–0.50326	476.01	39.727	0.002144	5.006	0.002007
Dogleg	0.24286	3.3323	–0.21190	–0.61459	454.94	40.314	0.001908	4.997	0.001985

5.14.3 Recovered Master Curves

One interesting phenomenon from previous evaluation results is that every optimizer gave different master curve variables, yet most of them still achieved good RMSRE. Therefore, the reasonableness of the recovered variables from the various optimizers must be evaluated by comparing the backcalculated master curves for the AC layer. Evaluations were carried out on the synthetic three-layer pavement structure. Results, shown in Figure 86, indicate that most optimizers were able to fit the target master curve (i.e., the input master curve in the forward calculation) suitably within the frequency range of the measurements induced by the FWD testing (~10–100 Hz). This demonstrates the effectiveness of the optimizers in backcalculating the portion of the master curve within the excited frequencies under the load.

Additional ways to recover the target master curve more accurately were explored. The key challenge was that the target line segment (10~100Hz) was too short, whereas the fitting equation had four free variables (see Section 4.2.1). Therefore, multiple sets of variables could have fit the target line segment reasonably well, as shown in Figure 86. However, not all of those fitted variables are feasible in practice. Thus, researchers proposed to co-optimize deflections measured at different temperatures on the same pavement structure to improve the feasibility of the recovered variables. The key idea of this proposal is that the change of temperature would cause the master curve to shift horizontally because of the time-temperature superposition principle (TTSP). Thus, by using multiple temperatures there would be multiple target line segments, which could be an improvement.

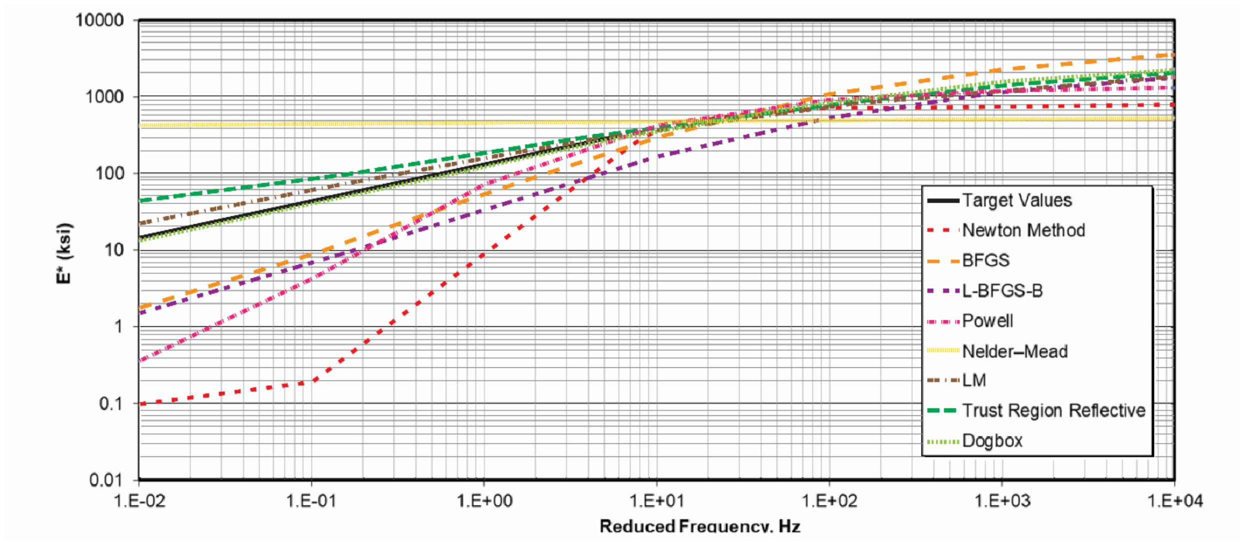


Figure 86. Comparison Between the Backcalculated Master Curves from All Optimizers and the Target Master Curve

5.14.4 Trustiness of the Recovered Master Curves

Based on the research up to this point, it could be concluded that the recovered master curves matched the target master curve only in the test frequency range (i.e., excited load-induced frequency range). Therefore, it should be determined how users know which part to trust in the recovered master curves. A key observation of this study was that different optimizers gave diverse master curves. However, all the best-fit curves tended to overlap within the test frequency range. Considering that most classic optimizers, like the Newton optimizer, do not have convergence guarantee, it means for different seeds, these classic optimizers will give different results. Thus, researchers proposed to generate the trust region directly from the deflections without the need of a priori knowledge, i.e., to use different seeds to get different master curves. Then, the overlapping area can be used to get the test frequency range.

To determine how the trust region of the master curve is located using a Newton optimizer, multiple optimization processes were run with different initial variables (seeds). All the proposed master curves from these optimization processes were recorded, as shown in Figure 87; the blue curves indicate the proposed master curves, and the thick black curve shows the target master curve.

As shown in Figure 87, there is no obvious pattern because the master curves from the optimizer are randomly overlapped, and some curves are showing even infeasible patterns due to the unconstrained nature of Newton's method. Thus, the curves are filtered with the RMSRE metric. The assumption is that if a master curve can generate deflections very similar to the target deflections, then it should be closer to the target master curve.

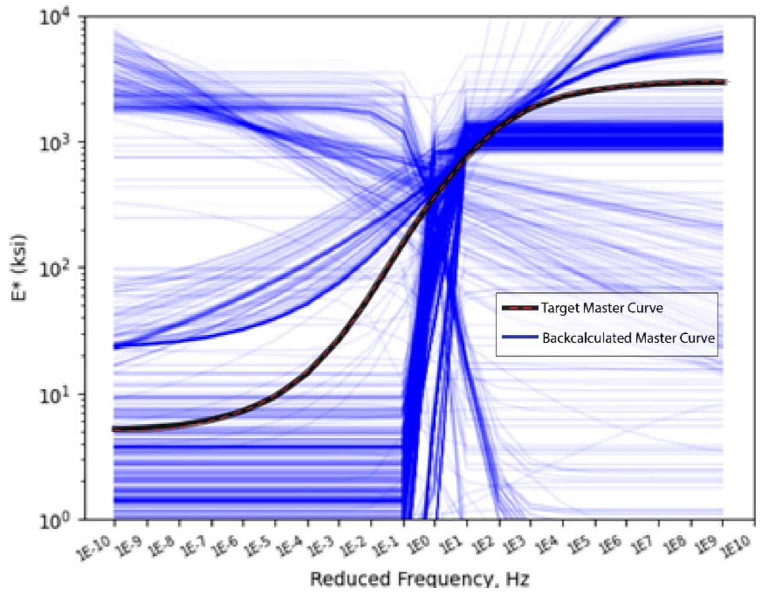


Figure 87. Comparison of the Backcalculated Master Curves from the Newton Optimizer with Different Seeds

Figure 88 shows the master curves with an RMSRE of 5 percent of less. The pattern is very clear in the figure where the master curves diverge in all other areas but the ~10–100Hz range. When comparing the overlapped regions to the target master curve, they are nearly identical. That means this approach can successfully locate the trust region in a master curve without the knowledge of the actual test frequency range used in the field experiments (i.e., during HWD/FWD testing).

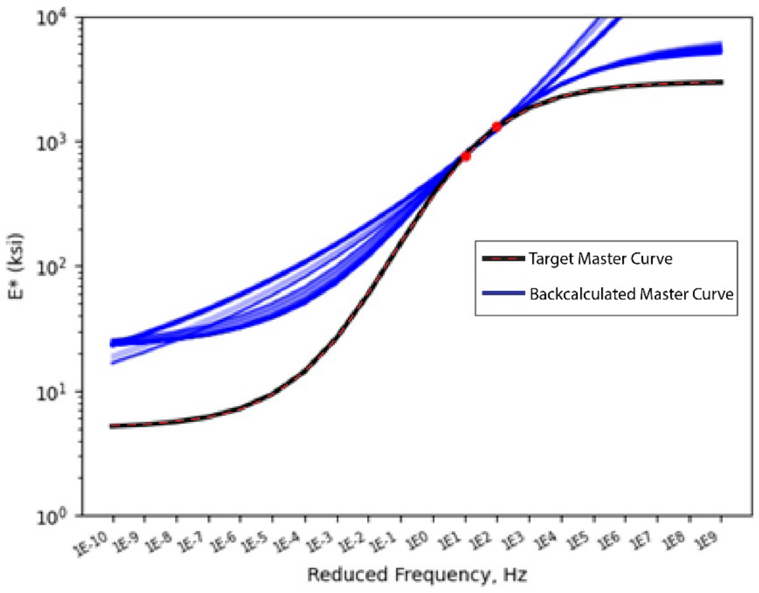


Figure 88. The Backcalculated Master Curves from the Newton Optimizer with Different Seeds Showing only the Curves with an RMSRE of Five Percent or Less

An *automatic* approach was proposed in this study to determine this trust region by considering the coefficient of variation (COV) in terms of each frequency. COV is a statistic of a variable that stays independent of its mean (scale). Thus, the COV at each frequency gives a numerical measurement of how much the master curves overlap at each frequency as shown in Figure 89.

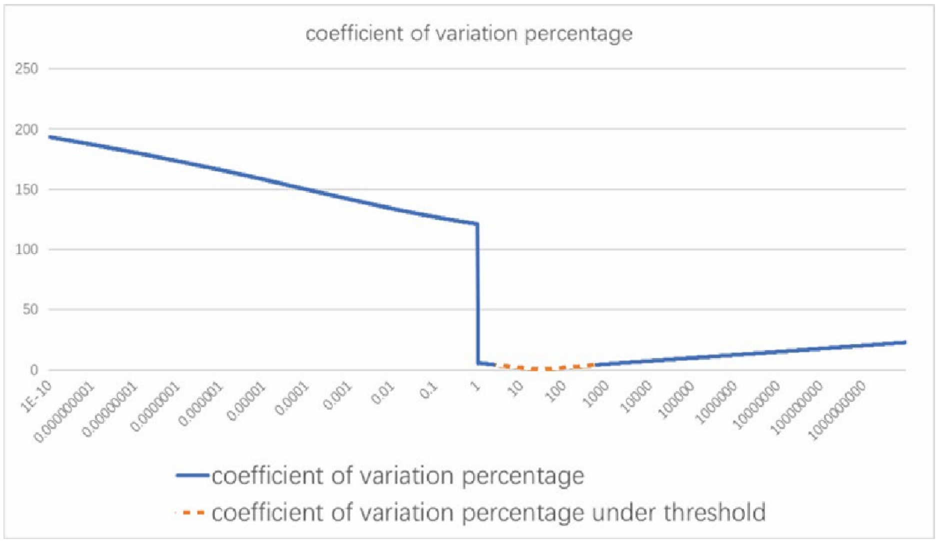


Figure 89. The COV Percentage at Each Frequency for the Master Curves

The next step was to determine what should be the threshold for a frequency that is considered to be within the test frequency range. As shown in Figure 90, the acceptable repeatability recommended by AASHTO T 378 (2017) was implemented for the expected moduli.

Nominal Maximum Aggregate Size, mm	Average |E*|, MPa	Dynamic Modulus					
		S_r% %	Acceptable Range for n Specimens, % of Average				
			n=2	n=3	n=4	n=5	n=6
19	≥ 137 to < 200	20	56	66	72	78	80
19	≥ 200 to < 500	16	46	54	59	64	65
19	≥ 500 to < 1,000	14	38	45	49	53	55
19	≥ 1,000 to < 2,000	12	32	38	42	45	46
19	≥ 2,000 to < 5,000	9	27	31	34	37	38
19	≥ 5,000 to < 10,000	8	22	26	28	31	32
19	≥ 10,000 to < 16,400	7	19	22	24	26	27

Note: S_r% = repeatability coefficient of variation for $|E^*|$, percent

Figure 90. Repeatability COV Suggested by AASHTO T 378 (Red box points to the acceptable COV values for a modulus between 1,000 and 5,000 MPa.)

To validate the correctness of the frequency inferred by this approach, the recovered master curve is compared to the target master curve under three different temperatures: 39.2 °F, 68 °F, and 104 °F. As shown in Figures 91 through 93, the recovered master curves match closely with the target curves within the inferred frequency range regardless of seed values used by the Newton optimizer. This suggests the complex potential of the proposed method.

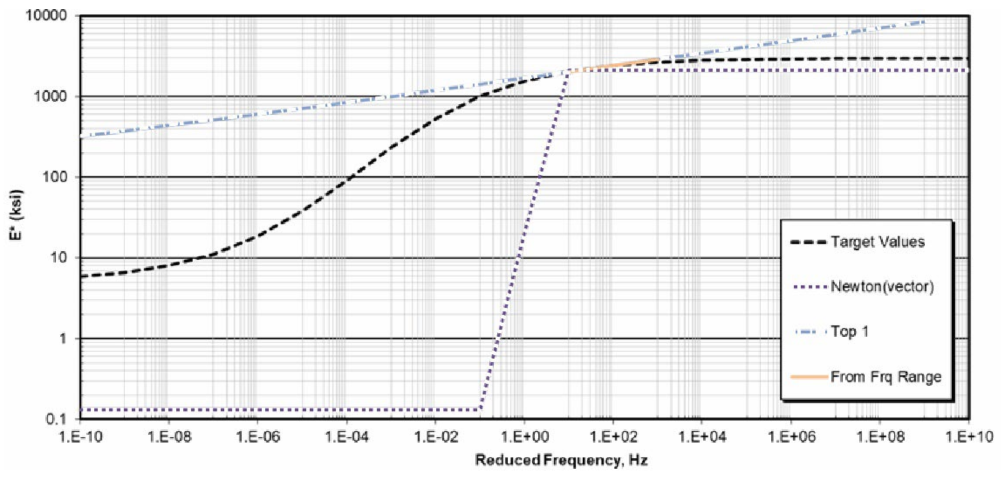


Figure 91. Comparison of the Recovered Master Curve and the Target Master Curve in Trust Frequency Range at Temperature 39.2 °F

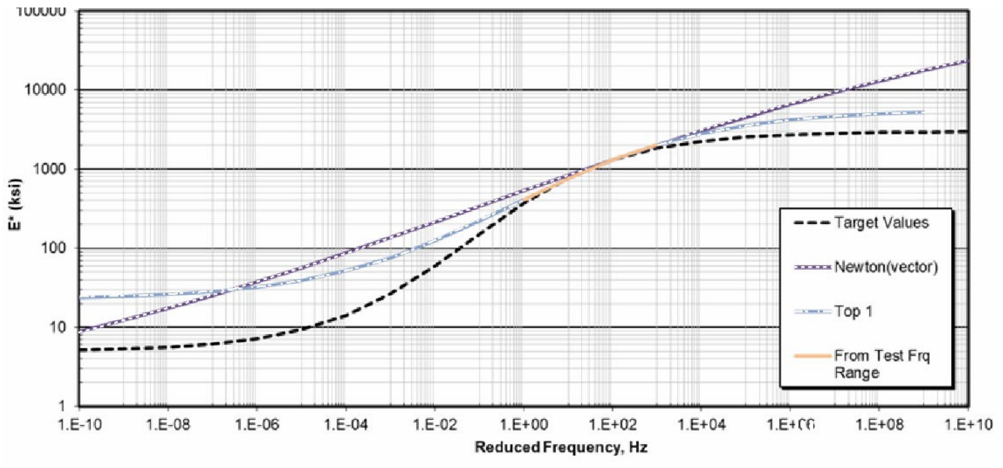


Figure 92. Comparison of the Recovered Master Curve and the Target Master Curve in Trust Frequency Range at Temperature 68 °F

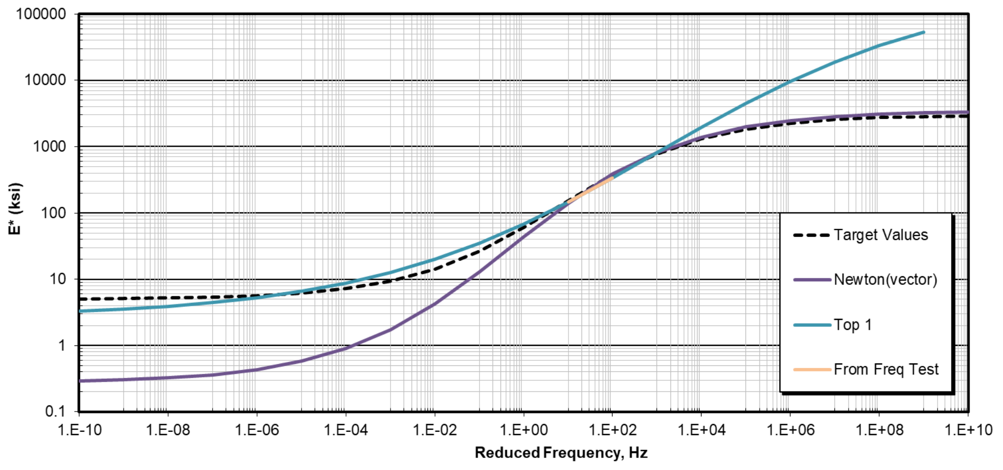


Figure 93. Comparison of the Recovered Master Curve and the Target Master Curve in Trust Frequency Range at Temperature 104 °F)

Now that the correctness of the inferred frequency range is validated, the trust frequency ranges inferred from different temperatures are used to improve the quality of the recovered master curves. The key idea is that the master curve of the same pavement structure at different temperatures shares all variables, but the betaPrime and the relationship of the target master curves are shifting horizontally. Thus, when there are three target master curve segments, it is possible to compute the shifting of each temperature:

$$\log(f_r) = \log(f) + a_1(T_R - T) + a_2(T_R - T)^2 \tag{135}$$

$$\log(a_T)= a_1(T_R - T) + a_2(T_R - T)^2 \tag{136}$$

More details of these equations can be found in Section 4.2.1.c. Based on previous analysis, the fitted a_1 and a_2 are found to be 0.06561 and 0.000106, respectively. The fitted line is shown in Figure 94. With the known horizontal shifting, the master curve segments could be moved to a reference temperature and used to fit the target master curve more preciously than fitting with only one segment. The recovered master curve by this approach is shown in Figures 95, 96, and 97. For all temperatures, the fitted master curve closely overlaps with the target even outside the test frequency range. This advantage is more obvious when comparing with the data in Figure 86, suggesting the proposed method is more effective. Table 36 summarizes the variables determined from the optimizer along the variables from the fitted master curves at three temperatures.

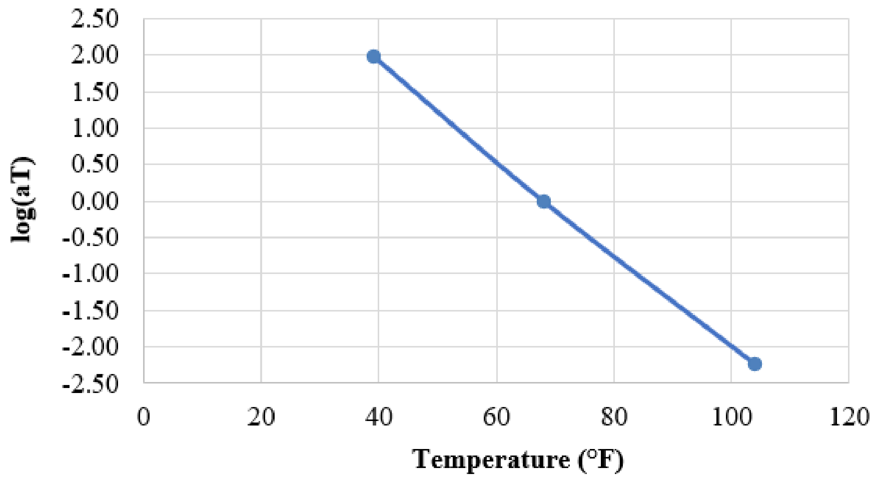


Figure 94. Linear Fitting of Temperature vs Shifting of the Master Curve log(aT)

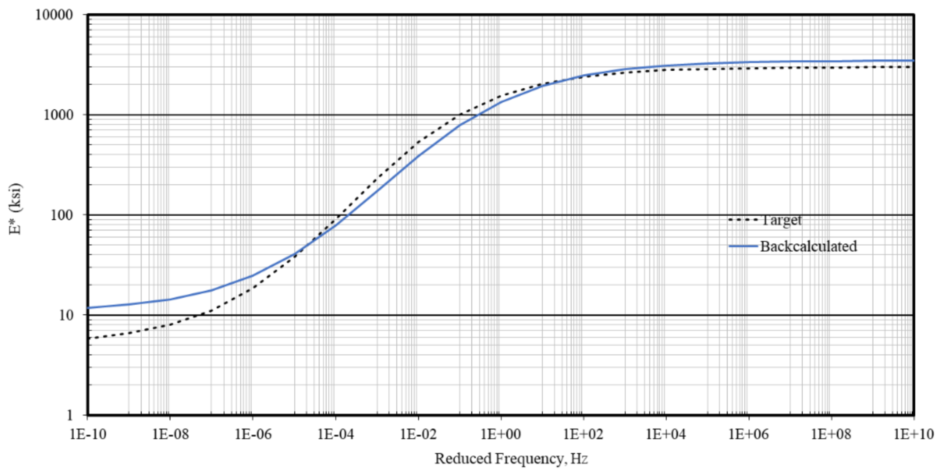


Figure 95. Fitted Master Curve with Three Master Curve Segments Shifted from Other Temperatures at 39.2 °F

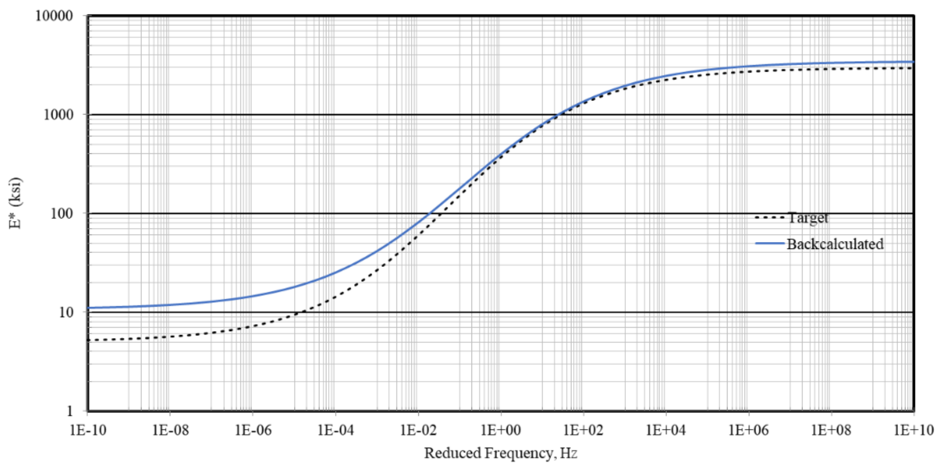


Figure 96. Fitted Master Curve with Three Master Curve Segments Shifted from Other Temperatures at 68 °F

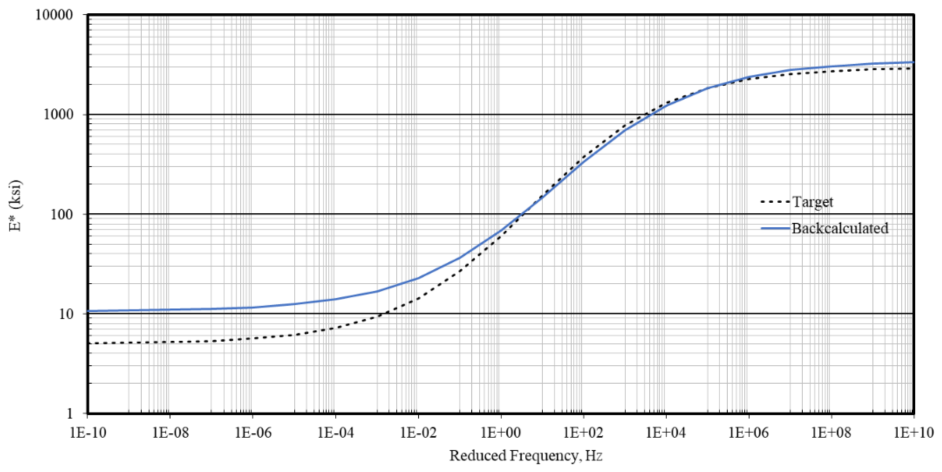


Figure 97. Fitted Master Curve with Three Master Curve Segments Shifted from Other Temperatures at 104 °F

Table 36. Comparison of the Variables from the Optimizers vs the Variables from the Fitted Master Curves at Three Temperatures

Parameters	Fitted Parameters from Optimizer	Fitted Parameters from Master Curves	Target Parameters
39.2 °F (10–1,000Hz)			
delta	0.7756	1.0260	0.6991
alpha	5.5526	2.5156	2.7761
gamma	–0.0560	–0.5655	–0.5887
beta′	0.2347	–1.6305	–2.1747
E2	34.872	–	35
E3	7.013	–	7
alphaR	29.74	–	30
betaR	0.0030	–	0.003
68 °F (1–1,000Hz)			
delta	1.3473	1.0260	0.6991
alpha	2.4079	2.5156	2.7761
gamma	–0.4551	–0.5655	–0.5887
beta′	–0.1075	–0.51218	–0.72193
E2	34.975	–	35
E3	7.003	–	7
alphaR	29.930	–	30
betaR	0.002997	–	0.003
104 °F (10–100Hz)			
delta	0.4332	1.0260	0.6991
alpha	4.9758	2.5156	2.7761
gamma	–0.3083	–0.5655	–0.5887
beta′	0.9344	0.7459	0.4457
E2	35.029	–	35
E3	7.002	–	7
alphaR	29.921	–	30
betaR	0.003003	–	0.003

–Not applicable

This further demonstrates the power of the proposed method by comparing the original variables provided by optimizer, the variables from fitted master curves, and the target variables. The variables from the fitted master curves are more similar to the target variables. This suggests that this approach could serve as a complementary part to existing optimization approaches.

5.14.5 Summary

Table 37 summarizes the optimizers that were evaluated in this study along with their properties in terms of:

- Algorithm computation cost
- Well-fitting synthetic one-layer system deflections
- Well-fitting synthetic three-layer system deflections
- Well-fitting field-measured three-layer system deflections
- Constrained optimization

Table 37. Summary of Optimization Methods and Their Properties

Optimizer	Algorithm Computation Cost	Well-Fitting, Synthetic, One-Layer System Deflections	Well-Fitting, Synthetic, Three-Layer System Deflections	Well-Fitting, Field-Measured, Three-Layer System Deflections	Constrained Optimization
First-Order Newton	Low	Yes	Yes (0.05%)	Yes (0.007%)	No
Second-Order Newton	Low	No	No (Diverged)	No (Diverged)	No
BFGS	Low	Yes	Yes (0.42%)	No (13.20%)	No
L-BFGS-B	Low	Yes	Yes (1.36%)	Yes (1.46%)	Yes
Powell	Low	Yes	Yes (0.12%)	Yes (1.95%)	Yes
Nelder–Mead	Low	–	Yes (0.58%)	Moderate (2.34%)	No
Bayesian	Moderate	–	Moderate (3.85%)	No (6.74%)	Yes
Levenberg–Marquardt	Low	–	Yes (0.085%)	Yes (0.67%)	Yes
Trust Region	Low	–	Yes (0.098%)	Yes (0.693%)	Yes
Dogleg	Low	–	Yes (0.055%)	Yes (0.676%)	Yes
Reinforcement Learning	High	–	Yes (1.24%)	Yes (1.125%)	Yes

–Not evaluated

6. GRAPHICAL USER INTERFACE

With the many developments in the FE methodology and optimization, the end goal of this project is to deliver an easy-to-use tool for pavement engineers. This study delivered Graphic user interface (GUI)-based software. The software is:

1. Graphical—users are able to use all functionality of this software by interacting with its graphic interface.

2. Compatible—the software supports all modern operating systems including Windows 7, 10, and 11.

The software GUI uses a button-page-based design. Currently, it consists of the *Create Mesh and FWD File* button (call and execute mesh generator program), the *Material Property* button (material property setting), the *Forward Analysis* button (perform forward calculation), and the *Dynamic Backcalculation* button (perform the backcalculation). These four buttons on the main page split the whole workflow into four steps with their corresponding contents. Users need to finish the work on *Create Mesh and FWD File* button before working on the *Material Property* button. The *Forward Analysis* button cannot be enabled before finishing the content in buttons *Create Mesh and FWD File* and *Material Property*. At the same time, the software allows dual unit systems throughout the program.

The following deliverables are targeted for the GUI software; it is a standalone software tool that performs backcalculation and optimization. It ships with:

- Installer
- Easy-to-use GUI
- Constraints for optimization
- Robust error handling

The software is finished with python code and properly aligns with the optimization code naturally.

6.1 DESIGN METHODOLOGY

Several important principles need to be enforced on the design of the BAKFAA Dynamic Backcalculation (DynaBAKFAA) software GUI before GUI coding can be conducted (note: Software GUI design refers to the process of creating user interfaces for software applications).

- ***Simplicity.*** Keep the interface simple and easy to use. Avoid clutter, unnecessary features, and complex designs. A simple and intuitive interface can help users to quickly understand how to use the software. In this GUI program, layout is kept simple and easy to understand.
- ***Consistency.*** Consistency in design means that all elements of the interface should look and function the same way throughout the application. This helps to avoid confusion and makes the interface more predictable.
- ***Feedback.*** The interface should provide clear and timely feedback to the user. For example, when a button is clicked, the user should see a response or an indication that the action was successful.
- ***Error prevention and recovery.*** The interface should be designed to prevent errors and to help users recover from them if they do occur. For example, providing helpful error messages can help users to correct mistakes or avoid them in the first place.
- ***Visibility.*** The interface should make important information and features clearly visible to the user. This can include using color, size, or placement to draw attention to important elements.

- ***Responsiveness.*** The GUI should be responsive and efficient, with fast loading times, smooth animations, and minimal lag. This helps to make the software feel more polished and professional.
- ***Aesthetics.*** The interface should be visually appealing and well-designed. This can include using color, typography, and other design elements to create a pleasing and engaging interface.

By following these principles, the software interfaces were created in a manner to be easy to use, visually appealing, and effective at helping users to accomplish their goals.

6.2 ARCHITECTURE

GUI architecture refers to the design and organization of the various components and modules that make up the user interface of a software application. A DynaBAKFAA GUI architecture can provide a good user experience by making the application easier to use and navigate.

The DynaBAKFAA GUI consists of four main buttons, each of which corresponds to a distinct functional module (Figure 98). Each module is independent and can interact with others to some extent.

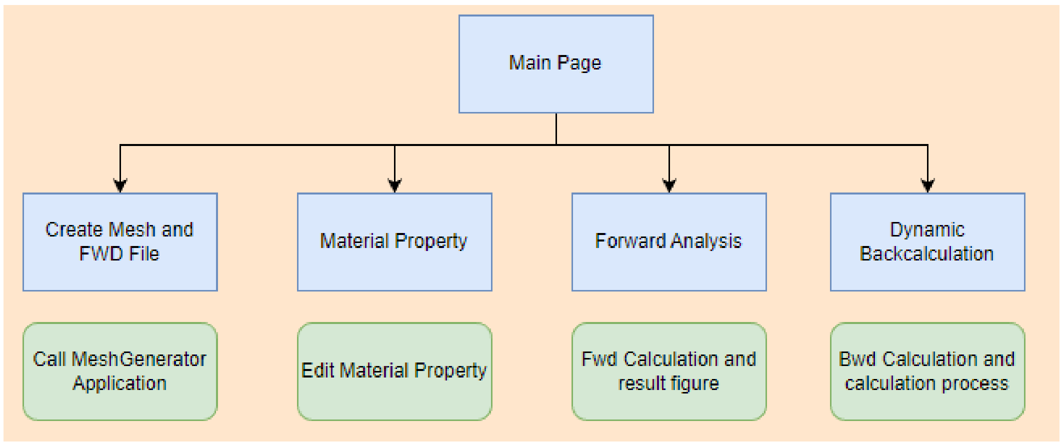


Figure 98. DynaBAKFAA Architecture of Software GUI

6.2.1 Main Page

The main interface is mainly composed of four buttons, each of which lead the user to four different main functional pages, as shown in Figure 99.

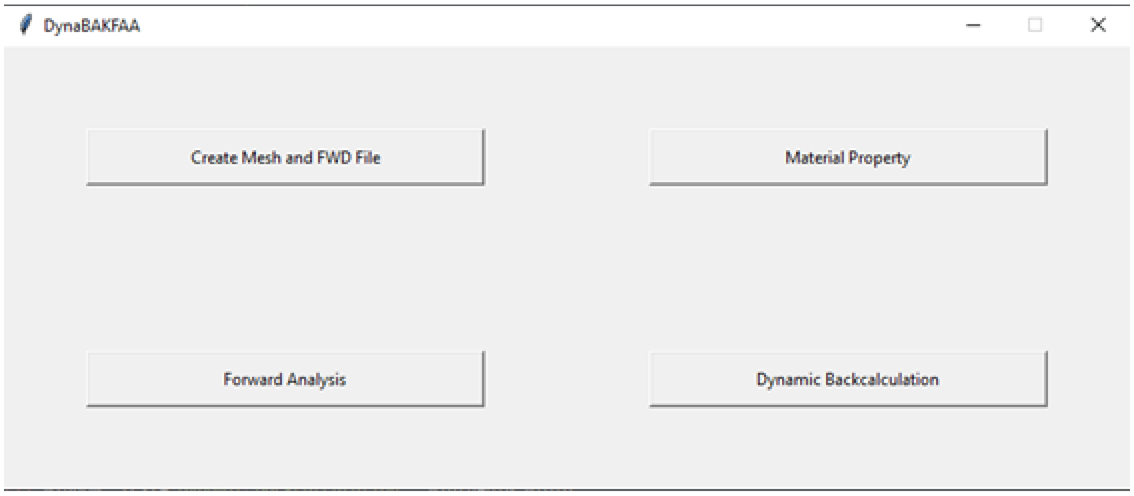


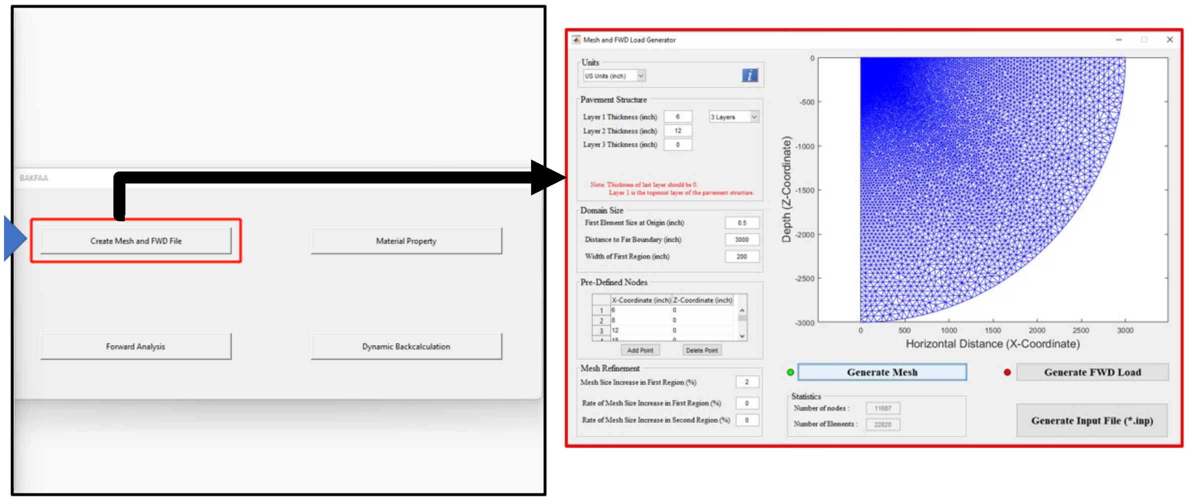
Figure 99. Main Page of DynaBAKFAA Software GUI

The detailed introductions of the main page are as follows:

- Create Mesh and FWD File
    - Directly call and run the mesh generator program (Application).
    - Users need to fill out the input information and generate the mesh file.
- Material Property
    - "Create Mesh and FWD File" button content needs to be finished before this page.
    - Users can edit material properties for each layer and save the changes.
- Forward Analysis
    - "Material Property" button content needs to be finished before this page.
    - Make forward analysis calculation and plot the results.
    - Users can edit the location settings before running the forward analysis.
    - Users can save comma-separated values (CSV) to customized path.
- Dynamic Backcalculation
    - Backcalculation page.
    - Users receive real-time update of the calculation progress.

6.2.2 Create Mesh and FWD File

The Create Mesh and FWD File page forks a process and calls the MeshGenerator program asynchronously (Figure 100). Once the MeshGenerator program is launched, other pages are disabled before users finish the MeshGenerator program.



Units
US Units (inch)

Pavement Structure
Layer 1 Thickness (inch) 6 — 3 Layers
Layer 2 Thickness (inch) 12
Layer 3 Thickness (inch) 0

Note: Thickness of last layer should be 0.
Layer 1 is the topmost layer of the pavement structure.

Domain Size
First Element Size at Origin (inch) 0.5
Distance to Far Boundary (inch) 3000
Width of First Region (inch) 200

Pre-Defined Nodes

	X-Coordinate (inch)	Z-Coordinate (inch)
1	6	0
2	8	0
3	12	0
4	18	0

Mesh Refinement
Mesh Size Increase in First Region (%) 2
Rate of Mesh Size Increase in First Region (%) 0
Rate of Mesh Size Increase in Second Region (%) 0

Statistics
Number of nodes : 11687
Number of Elements : 22820

Figure 100. Create Mesh and FWD File Button Call MeshGenerator Program

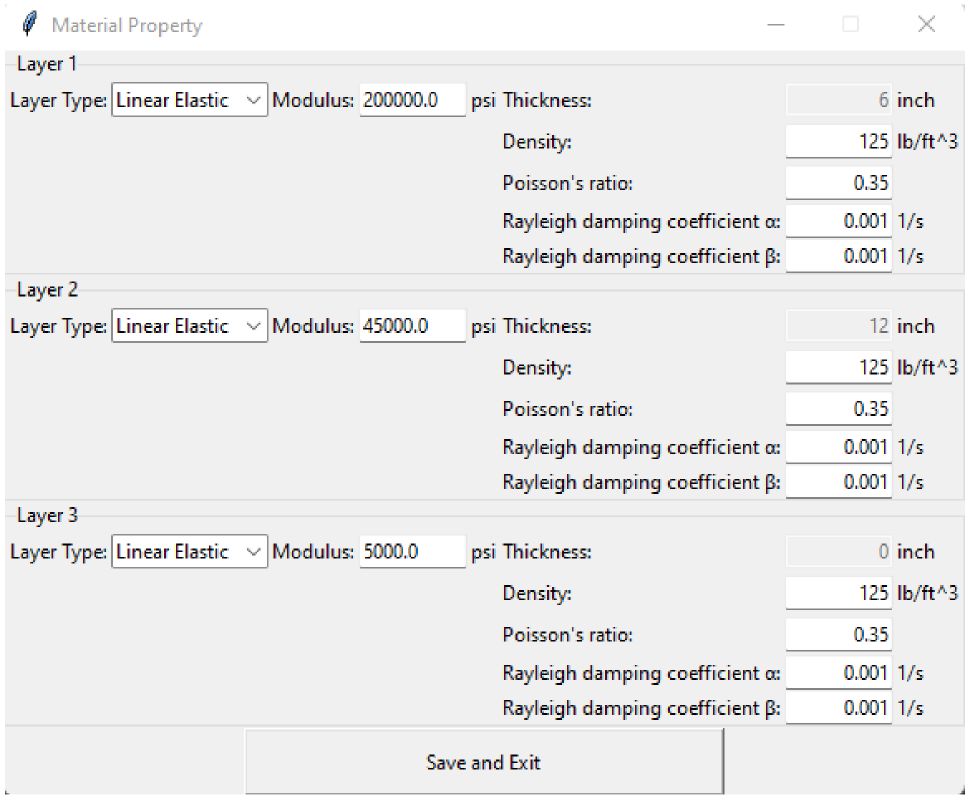
The GUI of the MeshGenerator is shown in Figure 100. Users need to generate the mesh file (.inp format) and the mesh file is used in the upcoming steps. At the same time, once the MeshGenerator program is terminated, it signals the DynaBAKFAA GUI program to enable the disabled buttons. The generated mesh file is saved to a path within the working directory and the path is under a hidden directory without user's access. The mesh file and the material property file are used to perform the forward and backward calculations.

6.2.3 Material Property

The Material Property page is where users specifically define all the properties of each pavement layer. The number of layers on this page is flexible and determined by the user's input from the MeshGenerator program. The GUI can display any number of layers, and users are able to customize all necessary material properties within this page. For each layer, there are certain properties that can be set and customized as per the user's requirements. These properties include (Figure 101):

- Layer Type: Linear Elastic, Viscoelastic
- Modulus
- Layer Thickness
- Density
- Poisson's Ratio
- Rayleigh damping coefficients α and β
- Sigmoidal function coefficients for viscoelastic modulus α, β', γ, and δ

These are the properties of the material for each layer, and this information is used to run the forward analysis in the next step.


Material Property

Layer 1

Layer Type:	Linear Elastic	Modulus:	200000.0 psi
Thickness:	6 inch		
Density:	125 lb/ft^3		
Poisson's ratio:	0.35		
Rayleigh damping coefficient α:	0.001 1/s		
Rayleigh damping coefficient β:	0.001 1/s		

Layer 2

Layer Type:	Linear Elastic	Modulus:	45000.0 psi
Thickness:	12 inch		
Density:	125 lb/ft^3		
Poisson's ratio:	0.35		
Rayleigh damping coefficient α:	0.001 1/s		
Rayleigh damping coefficient β:	0.001 1/s		

Layer 3

Layer Type:	Linear Elastic	Modulus:	5000.0 psi
Thickness:	0 inch		
Density:	125 lb/ft^3		
Poisson's ratio:	0.35		
Rayleigh damping coefficient α:	0.001 1/s		
Rayleigh damping coefficient β:	0.001 1/s		

Figure 101. Material Property Page of Software GUI

6.2.3.1 Handling US and SI Units

DynaBAKFAA GUI supports both US and SI unit systems simultaneously. This means that users can choose to work with either unit system based on their preference or the requirements of their own project.

The US unit system is commonly used in the United States and includes units such as inches, feet, pounds, and gallons. The SI unit system is the international standard and includes units such as meters, kilograms, and liters.

With the GUI software's ability to support both unit systems, users can easily switch between them and work with the one that is most convenient for their task. This flexibility can save time and effort for users who might need to work with different unit systems in their work or research.

6.2.3.2 Persistent Changes

The Save and Exit button allows users to save any changes made to a cached file and exit this page. The input information will be saved and cached in the designated path within the software, with a cache lifespan that lasts for the duration of the software's runtime. This means that when the software is closed completely, the input information from the previous session will not be retained, and when the software is opened again, the input information will revert to the default values.

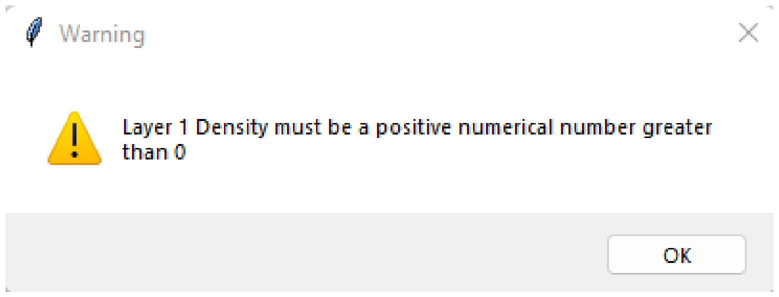
6.2.3.3 Input Validation Check for Data Entry

Input validation check for every entry means the process of verifying and validating the data entered by the user before it is retained to the cached file by the GUI. This is an important step to help ensure the accuracy and integrity of the data and prevent errors or issues from occurring during the processing of the data.

The DynaBAKFAA input validation checks include checking for the correct data type, verifying that the data fall within acceptable ranges or limits, and ensuring that the data are in the correct format.

By performing input validation checks for every entry, DynaBAKFAA GUI software can ensure that the data entered by the user are accurate and valid before processing. This helps prevent errors and issues that might arise from processing invalid data, such as crashes, incorrect calculations, or incorrect outputs.

With the input validation check, the software shows a warning (example shown in Figure 102) that details the error types and where the error happened so that users can identify and modify the error conveniently.


Warning

Layer 1 Density must be a positive numerical number greater than 0

Figure 102. Error Warning Information

6.2.4 Forward Analysis

The Forward Analysis button takes the mesh file generated by the first button and the material properties information stored in a cached JSON format file in the second button, along with the number of nodes as parameters and passes them into the calculation program *PULSE*_FE to perform the forward analysis.

After running the *PULSE*_FE calculation program with the previous input, the corresponding results are plotted based on the distance parameters as shown in Figure 103. On the left Y-axis, surface deflection is represented, on the right Y-axis FWD load is represented, and the X-axis represents time. Different curves correspond to results from different radial distances.

An option (Export to CSV button) is also provided to store the detailed results generated by the calculation to a customized path within a CSV format file. This option can be used by researchers for more in-depth exploration.

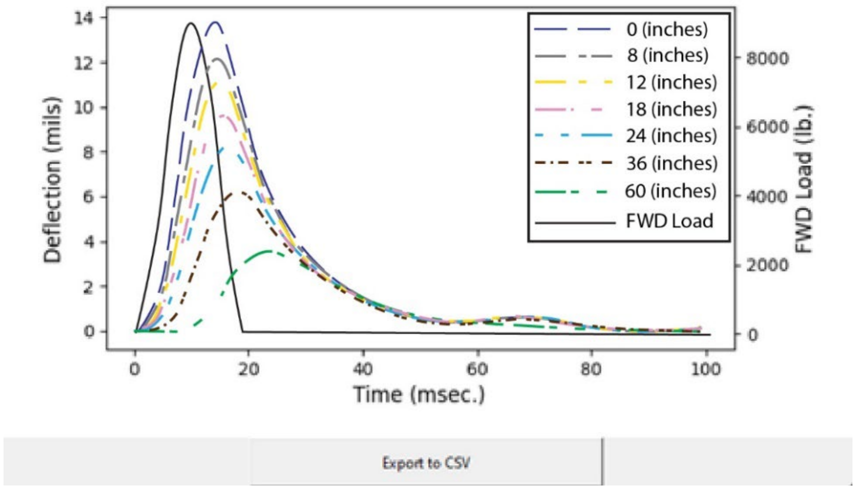
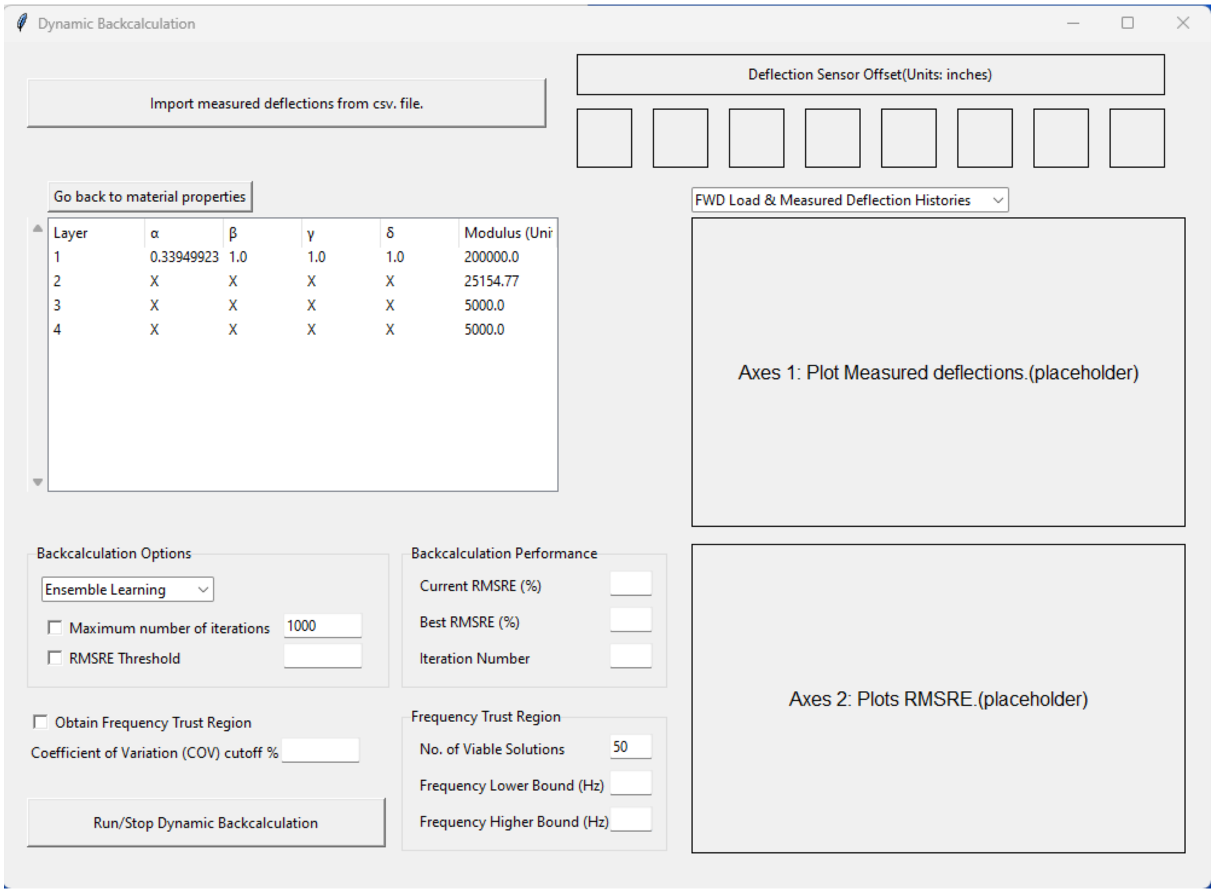


Figure 103. Deflection Plotting Page of Software GUI

6.2.5 Dynamic Backcalculation

The Dynamic Backcalculation page of software GUI layout is shown in Figure 104. The target page has the full functionality of the most viable optimization methods discussed in Section 5.



Dynamic Backcalculation
Import measured deflections from csv. file.
Deflection Sensor Offset(Units: inches)
Go back to material properties
FWD Load & Measured Deflection Histories
Axes 1: Plot Measured deflections.(placeholder)
Axes 2: Plots RMSRE.(placeholder)
Run/Stop Dynamic Backcalculation

Layer	α	β	γ	δ	Modulus (Uni
1	0.33949923	1.0	1.0	1.0	200000.0
2	X	X	X	X	25154.77
3	X	X	X	X	5000.0
4	X	X	X	X	5000.0

Backcalculation Options
Ensemble Learning
☐ Maximum number of iterations 1000
☐ RMSRE Threshold

☐ Obtain Frequency Trust Region
Coefficient of Variation (COV) cutoff %

Backcalculation Performance
Current RMSRE (%)
Best RMSRE (%)
Iteration Number

Frequency Trust Region
No. of Viable Solutions 50
Frequency Lower Bound (Hz)
Frequency Higher Bound (Hz)

Figure 104. Dynamic Backcalculation Page of Software GUI

6.3 ROBUSTNESS

6.3.1 User Input Check

Users need to give the pavement structure layer information that is in different orders of magnitude. Thus, it is important the software correctly parse user input. Currently the software can take arbitrary numbers/floats even with scientific notations and will prompt for incorrect inputs.

6.3.2 Applied Loads

*PULSE*_FE expects the applied loads to start from (0,0), but that is not always the case as the load level for location 0 is often missing. Thus, the Dynamic Backcalculation software will automatically set (0,0) if it detects the load information for location 0 is missing.

7. DATABASE

The ensemble learning training phase, while yielding good results, often imposes a notable time burden, potentially extending to a duration of up to 20 hours. Such a prolonged timeframe presents an obstacle for most users, for whom obtaining prompt and quality outcomes is of importance. Considering this challenge, strategies to expedite this training process were explored to make the use of the tool more practical and accessible to users.

The solution to this predicament centers on a two-fold approach, the pivotal element of which revolves around harnessing the power of a precomputed *PULSE*_FE data set. This data set is generated through extensive prior runs and then cataloged within a dedicated database. The creation of the database benefits the training process in two aspects: (1) a good initial starting point could largely speed up the convergence process and (2) the cached database entries could be reused by the program.

7.1 MOTIVATIONS

The *PULSE*_FE program requires a significant amount of time to produce results—typically ranging between 40 to 80 seconds—and is contingent upon varying material properties and mesh file sizes. In the developed operational workflow, these *PULSE*_FE simulation outcomes serve as the foundation for the machine-learning training. Notably, the machine-learning training endeavor necessitates numerous iterations, often numbering in the hundreds or even thousands, to attain convergence.

In the context of the established workflow, each training iteration hinges upon the availability of *PULSE*_FE results. Consequently, a single iteration demands at least 40 seconds to complete. Based on prior experiences, the cumulative training process can extend beyond 20 hours in pursuit of an acceptable solution.

The primary rationale behind establishing a dedicated database lies in its capacity to significantly curtail the requisite training time. By conducting an exhaustive execution of more than 3 million instances of the *PULSE*_FE program and systematically cataloging the resulting data, the database offers the ability to query these outcomes without the need for program re-execution. This database infrastructure affords the valuable capability to selectively identify optimal initial values and to capitalize on previously generated results throughout the course of the training process, thereby significantly enhancing its efficiency.

The database is built with a key-value pair structure, wherein the key corresponds to a specific parameter configuration, and the associated value captures the *PULSE*_FE results. This paired information is stored within the files, enabling swift and efficient querying during database use.

The process of ensemble-learning training, as shown in Figure 105, has the possibility of yielding highly favorable outcomes. It frequently entails a considerable investment of time, often spanning several hours or even extending to tens of hours, before reaching an optimal solution. Within this framework, two paramount components exist that have a substantial influence on the training speed:

- **Initial Point Selection:** The choice of the initial point plays a pivotal role in influencing the trajectory of the training process. Ensuring an informed selection at this juncture can significantly expedite the convergence of the ensemble-learning process.

- **Running times of *PULSE*_FE:** An integral facet of the training workflow, the *PULSE*_FE running time stands out as the most computing-intensive element. It can act as a bottleneck, impeding the overall training speed.

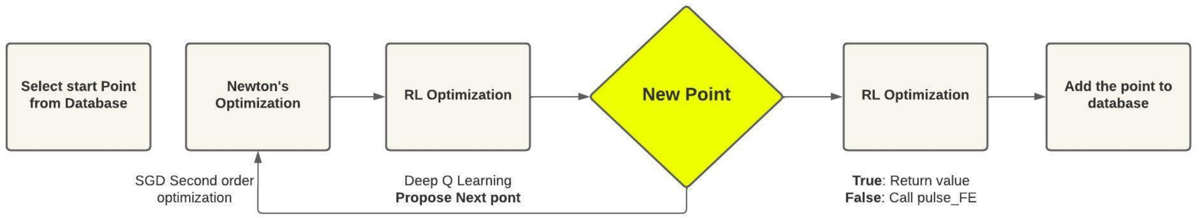


Figure 105. Database Involved in the Training Process

7.1.1 Selection of Initial Points

The machine-learning training process is based on Stochastic Gradient Descent (SGD). In a large search space, the selection of the starting point is crucial to the training performance. The selection of initial points refers to the starting point or initial values chosen for the model's parameters before the optimization process begins. In machine learning and optimization, the choice of initial values can have a significant impact on the convergence and efficiency of the optimization algorithm, as shown in Figure 106.

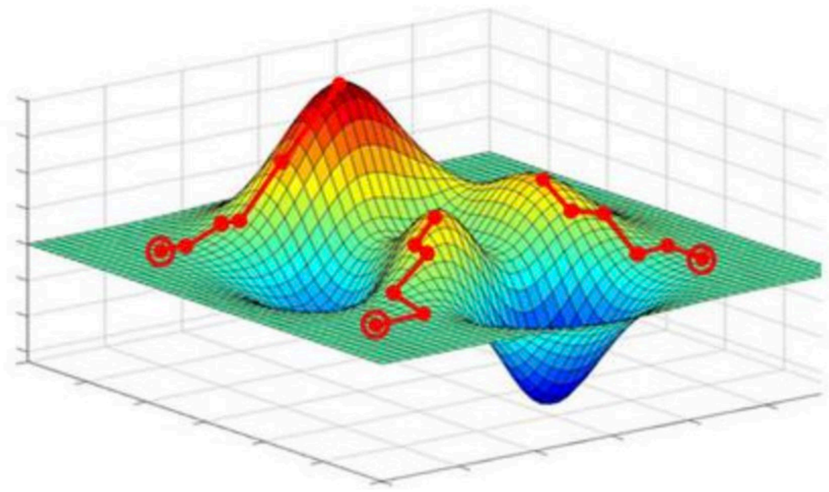


Figure 106. Selection of Initial Starting Point

How initial points impact SGD can be summarized as follows:

- Convergence Speed—The initial values can affect how quickly the optimization algorithm converges to the optimal solution. Poorly chosen initial values might result in slow convergence or the algorithm getting stuck in suboptimal solutions.

- Convergence to Global Optimum—For complex loss surfaces, such as in high-dimensional spaces, different initial values might lead to different local minima. Starting closer to the global optimum could increase the chances of finding a better solution.

- Stability—Poorly chosen initial values might lead to numerical instability during the optimization process. This instability can result in erratic behavior of the optimization algorithm and potentially prevent it from finding a good solution.

- Generalization—The initial points can also influence the generalization performance of the trained model. Depending on the optimization path followed, the model might generalize better or worse to new, unseen data.

- Avoiding Plateaus—In some cases, starting from certain initial points might lead to getting stuck on plateaus or flat regions of the loss surface, slowing down the optimization process.

To mitigate the impact of initial points in SGD and other optimization algorithms, practitioners often use techniques such as:

- Random Initialization—Initialize the parameters with random values drawn from a suitable distribution. This can help in exploring different regions of the loss surface.

- Pretraining—In some cases, pretraining a model on related tasks or using unsupervised learning can provide better initializations for the final optimization task.

- Learning Rate Scheduling—Adjust the learning rate during training to account for the initial point's impact and potentially reduce the chances of overshooting or getting stuck.

- Weight Regularization—Applying weight regularization techniques, such as L1 or L2 regularization, can help control the initial point impact by encouraging the model to start from a more centered position.

- Ensemble Methods—Training multiple models with different initializations and averaging their predictions can help mitigate the risk of getting stuck in poor solutions.

The choice of initial values in SGD and other optimization algorithms can play a crucial role in determining the optimization process efficiency, convergence, and the quality of the final solution. Experimenting with different initialization strategies and monitoring the optimization process are important practices to achieve better results.

In the established workflow, the RMSRE can be calculated beforehand to measure whether the point is good or not. Thus, the database can play an important role in this process because all the points inside the database can be iterated to find the best initial point at the start of the training process.

7.1.2 Cached Training Results

In the training process, each run generates a new data point, which is stored in the program's cache for future use. During subsequent runs, before starting the *PULSE*_FE computation, the process checks the cache. If the data point is already cached from previous runs, rerunning *PULSE*_FE is avoided, and the cached result is directly used. This prevents unnecessary repetitions and reduces the total training time by leveraging existing data points. This approach helps to optimize efficiency and resource usage in the established training workflow.

7.2 DATABASE CREATION

The database scope is determined by a fusion of various feature dimensions working in tandem, including:

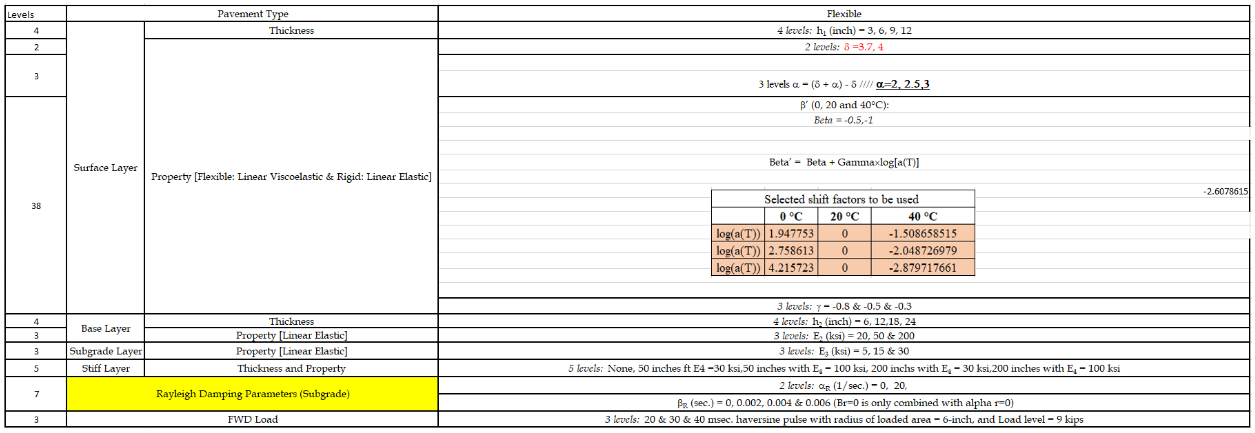
- FWD Load Duration (ms)
- Layer 1 Thickness (inches)
- Layer 2 Thickness (inches)
- Layer 3 Thickness (inches)
- Sigmoidal alpha
- Sigmoidal Beta
- Sigmoidal Gamma
- Sigmoidal Delta (psi)
- Modulus Layer 2 (psi)
- Modulus Layer 3 (psi)
- Rayleigh Alpha (1/s)

- Rayleigh Beta
- Modulus Layer 4 (psi)

For the running results from the *PULSE*_FE program, the following five values for each of the nodes, 0, 8, 12, 18, 24, 36, and 60, are collected. There are 35 values in total as the value of the key-value pair.

- Peak deflection
- Fifty-percent duration
- Time at peak
- Fifty-percent left time
- Fifty-percent right time

The amalgamation of these dimensions results in a total of 3,265,920 combinations. A preliminary assessment of the aggregate runtime for these combinations suggests an approximate duration of 1 month when leveraging 50 computing nodes. The matrix of the combination calculation process is shown in Figure 107.



Levels	Pavement Type		Flexible
4	Surface Layer	Thickness	*4 levels:* h_1 (inch) = 3, 6, 9, 12
2		Property [Flexible: Linear Viscoelastic & Rigid: Linear Elastic]	*2 levels:* δ =3.7, 4
3			3 levels α = (δ + α) - δ //// α=2, 2.5,3
38			β′ (0, 20 and 40°C): *Beta* = -0.5,-1 Beta′ = Beta + Gamma×log[a(T)] -2.6078615 *3 levels:* γ = -0.8 & -0.5 & -0.3
4	Base Layer	Thickness	*4 levels:* h_2 (inch) = 6, 12,18, 24
3		Property [Linear Elastic]	*3 levels:* E_2 (ksi) = 20, 50 & 200
3	Subgrade Layer	Property [Linear Elastic]	*3 levels:* E_3 (ksi) = 5, 15 & 30
5	Stiff Layer	Thickness and Property	*5 levels:* None, 50 inches ft E4 =30 ksi,50 inches with E_4 = 100 ksi, 200 inchs with E_4 = 30 ksi,200 inches with E_4 = 100 ksi
7	Rayleigh Damping Parameters (Subgrade)		*2 levels:* α_R (1/sec.) = 0, 20, β_R (sec.) = 0, 0.002, 0.004 & 0.006 (Br=0 is only combined with alpha r=0)
3	FWD Load		*3 levels:* 20 & 30 & 40 msec. haversine pulse with radius of loaded area = 6-inch, and Load level = 9 kips

Selected shift factors to be used			
	0 °C	20 °C	40 °C
log(a(T))	1.947753	0	-1.508658515
log(a(T))	2.758613	0	-2.048726979
log(a(T))	4.215723	0	-2.879717661

Figure 107. Matrix of Database Combinations

Initially, computing resources from the University of Nevada, Reno (UNR) supercomputing center were pursued. Although access to these resources was granted at no cost, the challenges to execute the *PULSE*_FE program on a Linux system were insurmountable. This is most likely because the program is a complicated Windows program with various versions of dependency. Despite concerted efforts, this path ended in a setback.

Accordingly, cloud computing was pursued. Cloud computing managed to successfully execute the program and create a comprehensive database. The cloud computing effort yielded a formidable cache of approximately 4 million data points, fortifying the database, and affording users a potent repository for future utilization.

7.3 CLOUD COMPUTING

After thorough investigation, it was determined that cloud computing stands as an optimal solution for executing the database generation workflow. By briefly analyzing the computing scales with the following cloud-computing details, the process was profiled on multiple types of Amazon Web Services (AWS) cloud instances.

- 3,265,920 data points
- Can run each single *PULSE*_FE run in up to 40 seconds
- Runs parallel computations on 50 nodes
- About 1 month computation time
- 3,265,920 results checkpoints
- Combines all files to a general database

The t2.medium instance was found as the optimum choice to use for the following reasons:

- The t2.medium instance offers compatibility with the Windows operating system.
- Among the available computing nodes, the t2.medium instance is the most suitable choice to effectively handle the execution of *PULSE*_FE.
- The t2.medium instance offers the most cost-effective pricing.

During the evaluation, a comparative analysis of prices across various regions revealed a notable discrepancy. Specifically, the cost for identical instances in the U.S. Eastern region (Northern Virginia) was discernibly 13 percent lower than that in the U.S. Western region (California). Given the intrinsic nature of the *PULSE*_FE program, the selection of the geographical region holds no bearing on its performance. Thus, the U.S. Eastern region (Northern Virginia), was selected.

7.3.1 Preparation of the Code

The AWS Boto3 python package was used as the code API to interact with the actions on the cloud instances. AWS Boto3 is the AWS SDK for Python. It provides a convenient and programmatic way to interact with various AWS services using Python code. Boto3 allows developers to write scripts, applications, and automation tools to manage and interact with AWS resources.

With Boto3, a wide range of tasks can be performed, such as creating and managing EC2 instances, interacting with S3 buckets, managing DynamoDB tables, and configuring networking resources. It abstracts the complexities of making API requests and handling authentication, making it easier for developers to integrate AWS services into their Python applications.

Boto3 provides a comprehensive set of functionalities to interact with AWS services, and it is widely used by developers and development and operations (DevOps) professionals for cloud infrastructure management, data processing, and building serverless applications, among other use cases.

7.3.2 Running the Database Generation Process

An impressive fleet of more than 70 AWS t2.medium instances was effectively deployed on the cloud and strategically harnessed to execute the *PULSE*_FE program in parallel. During the initial phase, a cautious approach was adopted, initiating operations with a subset of 20 nodes. This preliminary testing phase ensured the seamless functionality and flawless execution of the program. Subsequently, operations were progressively scaled up to encompass 50 nodes during less demanding periods. Moreover, during peak hours characterized by heightened computational demand, the full complement of 70 nodes was engaged to accommodate the substantial processing requirements of the program. This strategic use of computing resources optimized efficiency and productivity in support of the project goals.

7.3.3 Finalizing the Database

Following an intensive 30-day effort, a total of 3,265,920 data points from 70 nodes were successfully collected and consolidated into a single file. These results were analyzed, and data from checkpoints were parsed to establish key-value pairs using the specified features and values.

7.4 SUMMARY

Upon successfully creating the database and strategically curating an optimal initial starting point, the convergence of training iterations becomes significantly streamlined. As an example, Figure 108 illustrates the training process for a specific pavement structure (FWD 30 ms, layer 1 thickness:14.2 inches, layer 2 thickness: 7 inches).

For the specified pavement structure, a selection of candidate points is made, considering the similarity of the pavement structures in the database. Through an iterative process involving these candidate points and the calculation of corresponding RMSRE values, the best candidate point is identified as the initial training point. Figure 108 illustrates the training process, demonstrating rapid convergence and achieving a favorable RMSRE of 1.43 percent within 150 iterations. Initial evaluations indicate that, in most instances, training can be completed within an hour, a significant improvement compared to the previous range of 20 hours when starting points are not guided.

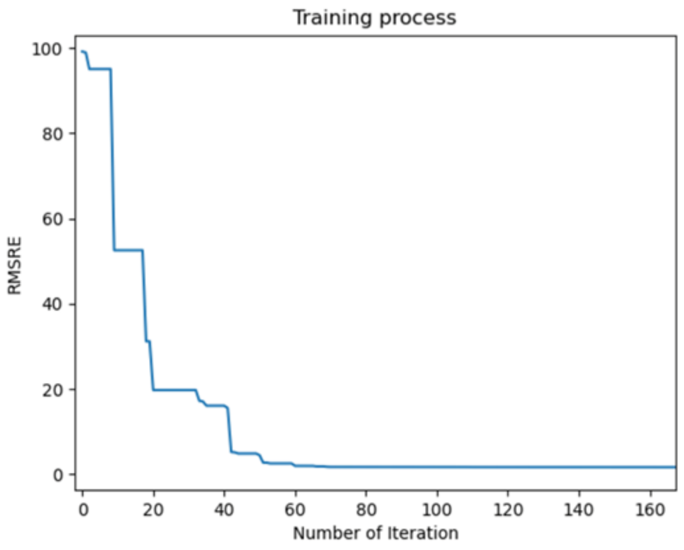


Figure 108. The Training Process with the Selection of the Initial Point

8. OVERALL SUMMARY AND CONCLUSIONS

This study created an effective and capable finite element (FE) module to assess responses from multilayer pavement structures. It handles linear elastic and viscoelastic isotropic materials under both static and dynamic heavy weight deflectometer (HWD)/falling weight deflectometer (FWD) loading conditions. Validation involved comparing surface deflections with ABAQUS, yielding identical results. Notably, the module completes calculations in just 1 to 3 percent of the time ABAQUS requires, a significant achievement for dynamic backcalculation feasibility.

Dynamic backcalculation employed the Newton-Raphson root-solving algorithm. Demonstrating the capacity to predict the asphalt concrete (AC) master curve, however, proved challenging. Backcalculation for a Construction Cycle (CC)-9 flexible test item delivered reliable layer variables. Analysis of various FWD drops revealed mild stress-softening behavior in the aggregate base and subgrade layers. A parametric study involved 15,552 pavement structures via FE modeling, yielding a preliminary list of significant FWD parameters for the backcalculation process.

An optimization framework was established to automate the backcalculation, seamlessly linking pavement structure modeling, preprocessing, FE modeling, and analysis. Calculated parameters could then be directly obtained from the specified variables without manual intervention. Implementing various optimizers like Newton-Raphson, Quasi-Newton, Powell, Nelder–Mead, Bayesian, and Kalman, coupled with different problem formulations, aided the development and evaluation of the optimization framework. Constrained optimization enhanced practical solution generation. Experimental evaluation, using synthetic and field-measured data, confirmed the effectiveness of the optimization methods and the reliable recovery of model variables through the developed approach.

Furthermore, a user-friendly GUI program for the BAKFAA Dynamic Backcalculation (DynaBAKFAA) software was introduced. This software facilitated tasks such as generating a mesh for the pavement structure domain, creating the FWD input files, inputting and editing material properties pavement layers, conducting forward analyses to determine pavement responses, and performing dynamic backcalculation with various optimizers to ascertain pavement variables.

9. REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In K. Keeton, & T. Roscoe (Chairs.), *OSDI '16: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* pp. 265–283.

ABAQUS. (2019). Finite Element Computer Program. Dassault Systèmes Simulia Corp., France.

American Association of State Highway and Transportation Officials (AASHTO). (2017). *Standard method of test for determining the dynamic modulus and flow number for asphalt mixtures using the asphalt mixture performance tester (AMPT)* (AASHTO T 378). Washington, DC.

AASHTO. (2021). *Standard method of test for determining the resilient modulus of soils and aggregate materials* (AASHTO T 307). Washington, DC.

BAKFAA. Federal Aviation Administration. Last accessed September 21, 2023: https://www.airporttech.tc.faa.gov/Products/Airport-Safety-Papers-Publications/Airport-Safety-Detail/bakfaa-342-1.

Bathe, K. J. (2014). *Finite element procedures* (2nd ed.). Klaus-Jürgen Bathe, Watertown, MA.

Bazi, G., Mansour, E., Sebaaly, P. E., & Hajj, E. Y. (2018). *FAA pavement evaluation and design model research grant number 16-G-018* (Final Report). University of Nevada, Reno.

Bazi, G., Mansour, E., Sebaaly, P., Ji, R., & Garg, N. (2019). Instrumented flexible pavement responses under aircraft loading. *International Journal of Pavement Engineering*, *22*(10), 1213–1225. https://doi.org/10.1080/10298436.2019.1671589

Bazi, G., Gagnon, J., Sebaaly, P., & Ullidtz, P. (2020). Effects of Rayleigh damping on the subgrade's apparent nonlinearity. *Journal of Transportation Engineering, Part B: Pavements*, *146*(3)., https://doi.org/10.1061/JPEODX.0000194

Bazi, G., Saboundjian, S., Bou Assi, T., & Diab, M. (2020, January 12–16). *Assessment of a low-volume flexible pavement through dynamic backcalculation* [Paper presentation]. Transportation Research Board (TRB) 99th Annual Meeting. Washington, DC.

Bazi, G. & Bou Assi, T. (2022). Asphalt concrete master curve using dynamic backcalculation. *International Journal of Pavement Engineering*, *23*(1), 95–106. https://doi.org/10.1080/10298436.2020.1733567

Björck, Å. (1996). Numerical methods for least squares problems. *Society for Industrial and Applied Mathematics*. https://doi.org/10.1137/1.9781611971484

Brust, J., Burdakov, O., Erway, J. B., Marcia, R. F., & Yuan, Y.-X. (2016). Algorithm XXX: SC-SR1: Matlab software for solving shape-changing L-SR1 trust-region subproblems. Accessed March 2024, eprint arXiv:1607.03533. https://doi.org/10.48550/arXiv.1607.03533

Byrd, R H., Lu, P., Nocedal, J., & Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, *16*(5), 1190-1208. https://doi.org/10.1137/0916069

Chatti, K., Ji, Y., Harichandran, R. S., & Lee, H. S. (2004). *Development of a computer program for dynamic backcalculation of flexible pavement layer moduli*(RC-1450 Final report). Michigan Department of Transportation, Lansing, MI.

Chatti, K., & Lei, L. (2012). Forward calculation of subgrade modulus using falling weight deflectometer time histories and wave propagation theory In R.D. Hryciw, A. Athanasopoulos-Zekkos, &N. Yesiller (Eds.), *GeoCongress 2012: State of the Art and Practice in Geotechnical Engineering* (Geotechnical Special Publication No. 225, pp. 1400-1409). https://doi.org/10.1061/9780784412121.144

Chen, X., Lin, Q., Kim, S., Carbonell, J. G., & Xing, E. P. (2012). Smoothing proximal gradient method for general structured sparse regression. *The Annals of Applied Statistics*, 6(2), 719–752. https://doi.org/10.1214/11-AOAS514

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., & de Freitas, N. (2018). *Bayesian Optimization in AlphaGo*. arXiv preprint arXiv:1812.06855. https://doi.org/10.48550/arXiv.1812.06855

Choi, J. W., Wu, R., Pestana, J. M., & Harvey, J. (2010). New layer-moduli back-calculation method based on the Constrained Extended Kalman Filter. *Journal of Transportation Engineering*, *136*(1). https://doi.org/10.1061/(ASCE)0733-947X(2010)136:1(20)

Cook, R. D. (1995). *Finite element modeling for stress analysis.* John Wiley & Sons.

De Borst, R., Crisfield, M. A., Remmers, J. J., & Verhoosel, C. V. (2012). *Non-linear finite element analysis of solids and structures*. John Wiley & Sons. https://doi.org/10.1002/9781118375938

Fu, G., Zhao, Y., Zhou, C., & Liu, W. (2020). Determination of effective frequency range excited by falling weight deflectometer loading history for asphalt pavement. *Construction and Building Materials*, *235*, Article number117792. https://doi.org/10.1016/j.conbuildmat.2019.117792

Jaeger, B., & Geiger, A. (2023). An Invitation to Deep Reinforcement Learning. arXiv preprint arXiv:2312.08365. https://arxiv.org/abs/2312.08365

Gao, F., & Han, L. (2012). Implementing the Nelder-Mead simplex algorithm with adaptive parameters. *Computational Optimization and Applications*, *51*, 259–277. https://doi.org/10.1007/s10589-010-9329-3

Geuzaine, C., & Remacle, J-F. (2009). Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, *79*(11), 1309–1331. https://doi.org/10.1002/nme.2579

Gopalakrishnan, K., Kim, S., Ceylan, H., & Kaya, O. (2014). *Development of asphalt dynamic modulus master curve using falling weight deflectometer measurements* (Tech Transfer Summary TR-659). Iowa State University, Institute for Transportation. https://rosap.ntl.bts.gov/view/dot/58914

Gopalakrishnan, K., Kim, S., Ceylan, H., & Kaya, O. (2015, June 10–12). Use of neural networks enhanced differential evolution for backcalculating asphalt concrete viscoelastic properties from falling weight deflectometer time series data. *Proceedings of the 6th International Conference on Bituminous Mixtures and Pavements*. Thessaloniki, Greece.

Hamim, A., Yusoff, N. I. M., Omar, H. A., Jamaludin, N. A. A., Hassan, N. A., El-Shafie, A., & Ceylan, H. (2020). Integrated finite element and artificial neural network methods for constructing asphalt concrete dynamic modulus master curve using deflection time-history data. *Construction and Building Materials*, *257*, Article number 119549. https://doi.org/10.1016/j.conbuildmat.2020.119549

Harichandran, R. S., Mahmood, T., Raab, A. R., & Baladi, G. Y. (1993). Modified Newton algorithm for backcalculation of pavement layer properties. *Transportation Research Recor*d, *1384*, 15–22. Strength and Deformation Characteristics of Pavement Structures (trb.org)

Hilber, H. M., Hughes, T. J., & Taylor, R. L. (1977). Improved numerical dissipation for time integration algorithms in structural dynamics. *Earthquake Engineering & Structural Dynamics*, *5*(3), 283–292. https://doi.org/10.1002/eqe.4290050306

Huang, Wen & Gallivan, Kyle & Absil, P.-A. (2015). A Broyden Class of Quasi-Newton Methods for Riemannian Optimization. SIAM Journal on Optimization. 25. 1660-1685. 10.1137/140955483.

Hutton, D.V. (2004). *Fundamentals of finite element analysis*. McGraw Hill.

Jaeger, B., &Geiger, A. (2023). *An invitation to deep reinforcement learning*. arXiv preprint arXiv:2312.08365. https://arxiv.org/abs/2312.08365

Kaliske, M., & Rothert, H. (1997). Formulation and implementation of three-dimensional viscoelasticity at small and finite strains. *Computational Mechanics*, *19*(3), 228–239. https://doi.org/10.1007/s004660050171

Khetan, A. & Karnin, Z. (2020, July 5–10). *schuBERT: Optimizing elements of BERT* (arXiv:2005.06628v1 [cs.CL]) [Long paper]. Submitted to the *58th Annual Meeting of the Association for Computational Linguistics* (virtual). https://doi.org/10.48550/arXiv.2005.06628

Kim, Y. R., Xu, B., & Kim, Y. (2000). A new backcalculation procedure based on dispersion analysis of FWD time-history deflections and surface wave measurements using artificial neural networks. In D. Shiraz & E. O. Lukanen (Eds.), *Non-destructive Testing of Pavements and Backcalculation of Moduli: Third Volume*. ASTM International. https://doi.org/10.1520/STP14774S

Kundu, R. (2022, March 1). *The complete guide to ensemble learning. V7*. https://www.v7labs.com/blog/ensemble-learning

Kutay, M. E., Chatti, K., & Lei, L. (2011). Backcalculation of dynamic modulus master curve from falling weight deflectometer surface deflections. *Transportation Research Record*, *2227*(1), 87–96. https://doi.org/10.3141/2227-10

Lee, H. S., Ayyala, D., & Von Quintus, H. (2017). Dynamic backcalculation of viscoelastic asphalt properties and master curve construction. *Transportation Research Record*, *2641*(1), 29–38. https://doi.org/10.3141/2641-05

Liu, G. R., & Quek, S. S. (2013). *The finite element method: A practical course* (2nd ed.). Elsevier.

Logan, D. L. (2017). *A First Course in the Finite Element Method* (6th ed.). Cengage Learning, Boston, MA.

Lourakis, M. I. A., & Argyros, A. A. (2005). Is Levenberg-Marquardt the Most Efficient Optimization Algorithm for Implementing Bundle Adjustment? In Proceedings / *IEEE International Conference on Computer Vision* (pp. 1526–1531). IEEE. https://doi.org/10.1109/ICCV.2005.128

Moré, J. J. (1978). The Levenberg-Marquardt algorithm: Implementation and theory in G.A. Watson (Ed.), *Lecture Notes in Mathematics*, *630*, 105-116. Springer. https://doi.org/10.1007/BFb0067700

Moritz, P., Nishihara, R., & Jordan, M. (2016). A Linearly-Convergent Stochastic L-BFGS Algorithm. In A. Gretton & C. C. Robert (Eds.), *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (pp. 249-258). PMLR. https://proceedings.mlr.press/v51/moritz16.html

Newmark, N. M. (1959). A method of computation for structural dynamics. *Journal of the Engineering Mechanics Division*, *85*(3), 67–94. https://doi.org/10.1061/JMCEA3.0000098

Oller, S. (2014). *Nonlinear dynamics of structures*. Springer International Publishing. https://doi.org/10.1007/978-3-319-05194-9

Oñate, E. (2009). *Structural analysis with the finite element method. Linear statics: Volume 1: Basis and solids*. Springer Science & Business Media. https://doi.org/10.1007/978-1-4020-8733-2

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*. https://doi.org/10.48550/arXiv.1912.01703

Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., & Vrgoč, D. (2016, April 11-15). Foundations of JSON schema. In J. Bourdeau, J. A. Hendler, & R. Nkambou (Chairs.), *WWW '16: Proceedings of the 25th International Conference on World Wide Web*, pp. 263–273. Quebec, Montreal, Canada. https://doi.org/10.1145/2872427.2883029

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, *7*(2), 155–162. https://doi.org/10.1093/comjnl/7.2.155

Pu, D., & Yu, W. (1990). On the convergence property of the DFP algorithm. *Annals of Operations Research*, *24*, 175–184. https://doi.org/10.1007/BF02216822.

Qu, Z. Q. (2004). *Model order reduction techniques with applications in finite element analysis*. Springer-Verlag London Ltd. https://doi.org/10.1007/978-1-4471-3827-3

Sebaaly, B., Davis, T. G., & Mamlouk, M. S. (1985). Dynamics of falling weight deflectometer. *Journal of Transportation Engineering*, *111*(6),618–632. https://doi.org/10.1061/(ASCE)0733-947X(1985)111:6(618)

Sebaaly, B. E., Mamlouk, M. S., & Davies, T. G. (1986). Dynamic Analysis of Falling Weight Deflectometer Data. *Transportation Research Record*, *1070*, 63–68.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

Varma, S., Kutay, M.E., & Chatti, K. (2013). Data requirements from falling weight deflectometer tests for accurate backcalculation of dynamic modulus master curve of asphalt pavements. In I. L. Al-Qadi & S. Murrell (Eds.), *Airfield and Highway Pavement 2013: Sustainable and Efficient Pavements* (pp. 445-455). American Society of Civil Engineers. https://doi.org/10.1061/9780784413005.035

Varma, S., & Kutay, M. E. (2016). Backcalculation of viscoelastic and nonlinear flexible pavement layer properties from falling weight deflections. *International Journal of Pavement Engineering*, *17*(5), 388–402. https://doi.org/10.1080/10298436.2014.993196

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256. https://doi.org/10.1007/BF00992696

Woodbury, A.C. (2008). *Localized coarsening of conforming all-hexahedral meshes* [Master's Thesis, Brigham Young University] Brigham Young University ScholarsArchive. https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=2535&context=etd

Wu, F., Luo, H., Jia, H., Zhao, F., Xiao, Y., & Gao, X. (2021). Predicting the noise covariance with a multitask learning model for Kalman Filter-Based GNSS/INS integrated navigation. *IEEE Transactions on Instrumentation and Measurement*, *70*, 8500613. https://doi.org/10.1109/TIM.2020.3024357

Ypma, T. J. (1995). Historical development of the Newton–Raphson method. *SIAM Review*, *37*(4), 531–551. https://doi.org/10.1137/1037125

Zaabar, I., Chatti, K., Lee, H. S., & Lajnef, N. (2014). Backcalculation of asphalt concrete modulus master curve from field-measured falling weight deflectometer data: Using a new time domain viscoelastic dynamic solution and genetic algorithm. *Transportation Research Record*, *2457*(1), 80–92. https://doi.org/10.3141/2457-09

Zienkiewicz, O. C., Taylor, R. L., & Zhu, J.Z. (2013). *The Finite Element Method: Its Basis and Fundamentals*. Elsevier. https://doi.org/10.1016/C2009-0-24909-9

APPENDIX A—OPTIMIZATION TECHNIQUE EXAMPLES

In this appendix, a simple one-layer system with three variables {E, Rayleigh alpha, Rayleigh beta} is used to detail how the optimization methods work in the proposed framework.

To be consistent with Section 5.3.5 of the main document, the deflections are generated with {E = 20,000 psi, Rayleigh alpha = 20, Rayleigh beta = 0.002}, and the optimizer is given an initial variables set of {E = 5,000 psi, Rayleigh alpha= 5, Rayleigh beta = 0.006}. Optimizers are expected to tweak the variables to recover the ground truth variables set that are used to generate target vertical deflections. Note the ground truth variables are unknown to the optimizers.

A.1 Newton-Raphson Method

Here is a full update iteration using first-order Newton-Raphson method. According to the update Equation 65 of Section 5.3.1,

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

$f'(x_0)$ is needed to update x. As the target function is not differentiable, the finite difference equation $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ from Section 5.3.3 is used to approximate $f'(x_0)$. Note that h is the step size. Typically, 1 percent of x is used as the step size.

Table A-1. Example of Calculation for First-Order Derivative Parameters

Rayleigh alpha	Rayleigh beta	E	RMSRE	Note
5	0.006	5,000	98.139	Initial point $f(x_0)$
5.05	0.006	5,000	98.101	$f(x_0 + 1\%\text{Rayleigh alpha})$
5	0.00606	5,000	97.748	$f(x_0 + 1\%\text{Rayleigh beta})$
5	0.006	5,050	96.881	$f(x_0 + 1\%\text{E})$

Table A-2. Example of First-Order Partial Derivative Calculation

F(x)	F(x+h)	h	Partial Derivative	Note
98.139	98.101	0.05	-0.741	Rayleigh alpha
98.139	97.748	0.00006	-6503	Rayleigh beta
98.139	96.881	50	-0.0251	E

Thus, the gradient $f'(x_0)$ = [-0.741 -6503 -0.0251]

x is updated as

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

= [5. 0.006 5000] - 98.139 / [-0.741 -6503 -0.0251]

= [137.374 0.0210 8903]

The root-mean-square relative error (RMSRE) for x_1 is 75.066. Compared with 98.139 of x_0, there is a 25 percent improvement. By iteratively updating x in this way, the target variables can be approached to resemble the target deflections.

A.2 Second-Order Newton-Raphson Method

The updated equation of second-order Newton-Raphson is:

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

To compute $f''(x_n)$ by finite difference:

$$f''(x_n) = \frac{f(x+hk) - f(x+h) - f(x+k) + f(x)}{hk}$$

Similar to first-order computation, the derivative parameters for both first order and second order are as follows:

Table A-3. Example of Calculation for First-Order Derivative Parameters

Rayleigh alpha	E	Rayleigh beta	RMSRE	Note
5	5,000	0.006	98.139	Initial point $f(x_0)$
5.05	5,000	0.006	98.101	$f(x_0 + 1\%$Rayleigh alpha$)$
5	5,000	0.00606	97.748	$f(x_0 + 1\%$Rayleigh beta$)$
5	5,050	0.006	96.881	$f(x_0 + 1\%\mathrm{E})$

Thus, the gradient $f'(x_0)$ = [-0.741 -6503 -0.0251].
For second-order terms, they are as follows

Table A-4. Example of Calculation for Second-Order Derivative Parameters

Rayleigh alpha	E	Rayleigh beta	RMSRE	Note
5.05	5,050	0.006	96.844	$f(x_0 + 1\%$Rayleigh alpha $+ 1\%\mathrm{E})$
5.05	5,000	0.00606	97.711	$f(x_0 + 1\%$Rayleigh beta $+ 1\%$Rayleigh beta$)$
5	5,050	0.00606	96.495	$f(x_0 + 1\%\mathrm{E}$ $+ 1\%$Rayleigh beta$)$
5.1	5,000	0.006	98.062	$f(x_0 + 1\%$Rayleigh alpha $+ 1\%$Rayleigh alpha$)$
5	5,100	0.006	95.648	$f(x_0 + 1\%\mathrm{E} + 1\%\mathrm{E})$
5	5,000	0.00612	97.358	$f(x_0 + 1\%$Rayleigh beta $+ 1\%$Rayleigh beta$)$

Based on Finite Difference equation, Hessian matrix can be computed:

$$\begin{bmatrix} 0.849 & 25.193 & 0.000283 \\ 25.193 & 62112 & 1.222 \\ 0.000283 & 1.222 & 0.00000943 \end{bmatrix}$$

The exact invert of Hessian is computationally expensive, and thus LU decomposition is used to approximate it.

$$\begin{bmatrix} -1.189 & -0.000333 & 7.557 \\ -0.000333 & -0.00000462 & 0.589 \\ 7.557 & 0.589 & 29809 \end{bmatrix}$$

Thus x can be updated by $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$ = [2.138 -0.00949 9591]. Unfortunately, the updated x leads to an infinite large RMSRE, and thus the optimization diverges.

A.3 BROYDEN–FLETCHER–GOLDFARB–SHANNO ALGORITHM (BFGS)

This example shows a breakdown of how BFGS, the most classical Quasi-Newton method, works on the one-layer system.

Given the BFGS updated equations:

$$\begin{aligned} x_{t+1} &= x_t - B_t^{-1} g \\ s_t &= x_{t+1} - x_t \\ y_t &= g_{t+1} - g_t \\ B_{t+1} &= B_t - \frac{B_t s_t s_t^T B_t}{s_t B_t s_t} + \frac{y_t y_t^T}{y_t s_t} \end{aligned}$$

and the fact that $B_0 = I$, it can be concluded that the first step of BFGS is identical to Newton's method. Thus, researchers use the conclusion from Appendix A.1 to set:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = [137.374\ 0.0210\ 8903]$$

Next is step 2 of the BFGS algorithm:

$$s_0 = x_1 - x_0 = [\ 137.374\ \ 0.0210\ \ 8903\ ] - [\ 5.\ 0.006\ 5000\ ] = [132.374\ \ 0.015\ \ 3903]$$

To compute $y_0 = g_1 - g_0$, finite difference (see Appendix A.1) is used to approximate g_1 :

Table A-5. Example of Calculation for First-Order Derivative Parameters

Rayleigh alpha	Rayleigh beta	E	RMSRE	Note
137.374	0.0210	8,903	74.920	Initial point $f(x_1)$
138.747	0.0210	8,903	75.301	$f(x_1 + 1\%\text{Rayleigh alpha})$
137.374	0.0212	8,903	75.277	$f(x_1 + 1\%\text{Rayleigh beta})$
137.374	0.0210	8,992	74.475	$f(x_1 + 1\%\text{E})$

Table A-5. Example of First-Order Partial Derivative Calculation

F(x)	F(x+h)	h	Partial derivative	Note
74.920	75.301	1.373	0.277	Rayleigh alpha
74.920	75.277	0.0002	1785	Rayleigh beta
74.920	74.475	89	-0.005	E

Thus, the gradient $g_1 = f'(x_1)$ = [0.277 1785 -0.005]

$$y_0 = g_1 - g_0 = [\ 0.277\ 1785\ -0.005] - [\ -0.741\ -6503\ -0.0251] = [\ 1.018\ 8288\ 0.0201]$$

With s_0 and y_0, the Hessian matrix could be approximated as

$$B_1 = B_0 - \frac{B_0 s_0 s_0^T B_0^T}{s_0^T B_0 s_0} + \frac{y_0 y_0^T}{y_0^T s_0} =$$

$$\begin{bmatrix} 1.001 & 24.997 & 0.0338 \\ 24.997 & 203513 & 0.493 \\ 0.0338 & 0.493 & 0.00115 \end{bmatrix}$$

And its inverse $B_1^{-1} =$

$$\begin{bmatrix} 10565489 & 2053 & 0.311519660 \\ 2053 & 0.399 & 60538 \\ 311519660 & 60538 & 9185046009 \end{bmatrix}$$

Thus

$$B_1^{-1} g = [\ -2295954\ 446.187\ -67695387],$$

which is the direction of update. The actual update is computed with a Wolfe line search by defining $f(\gamma) = f(x_1 - \gamma B_1^{-1} g)$ and solving $argmin(f(\gamma))$ with respects to $\gamma > 0$. The details of this line search algorithm can be found in Wright and Nocedal, *Numerical Optimization*, 1999, pp. 59–61. For demonstration purpose here, a simple step size, 0.00001, is used.

$$x_2 = x_1 - \gamma B_1^{-1} g = [160.333\ 0.0165\ 9579]$$

The RMSRE for x_2 is 68.255, which is better than 75.066 of x_1. By iteratively updating x in this way, the target variables can be approached to resemble the target deflections.