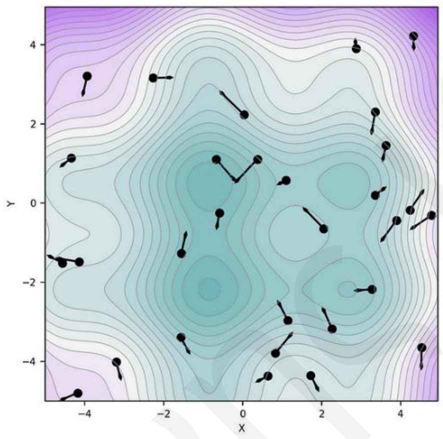
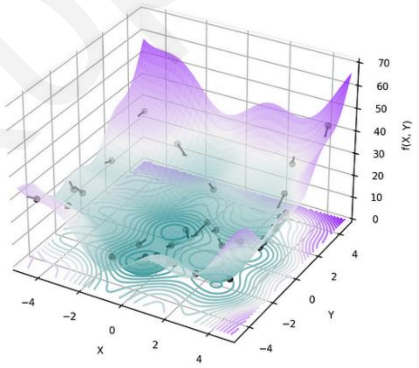
University of Cyprus
Department of Civil and Environmental Engineering

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Calibration of Viscoelastic Chain using the PSO algorithm to match various Creep Models

Ιωάννης Κυριάκου





CONTENTS

List of Tables

List of Figures

Abstract

This paper employs an optimization algorithm to approximate the creep-induced behavior of a concrete material as predicted by various well-established models of concrete that exist in the literature. The models in question are i) the B3 Model [1] ii) the CEB-FIB 2010 model code [2] and iii) the model in the current AASHTO LRFD specifications [3].

The topic of concrete creep is of utmost significance for the maintenance of infrastructure such as highway bridges, where the Dead Load is much larger than the live loads originating from traffic. Recent studies have shown that simplified, design-oriented equations may not always yield conservative predictions of long-term deformations. The economic effects of this shortcoming are enormous, while in some cases the structural integrity of the structure may be compromised.

The present study used a finite element analysis of a simple model through which the long-term deformation of a uniaxial element was analyzed based on the constitutive law provided by the above-mentioned model codes. The constitutive law used in the analyses consists of a solidifying viscoelastic chain, calibrated to match the creep curves generated by the three different models.

Introduction

It is well established that concrete, the solid that forms at room temperature by combining Portland cement with water, sand and aggregates, exhibits long-time deformations under load, even if the load is constant. This phenomenon is called *creep* and has been a subject of scientific study since the early 20th century, resulting in a plethora of mathematical models, each with its own set of relevant parameters, to estimate the time-dependent deformations of concrete to varying degrees of success.

In an ideal scenario there would be a minimal, consistent set of industry-standard models to predict long-term concrete behavior. This is currently an insurmountable task due to the large variety of concrete products that can be achieved as well as the inherent variability of the physicochemical processes that take place during the hydration process. The multi-phased nature of concrete and the reliance of the final product on the properties of its constituents is a source of significant uncertainty in the modelling process. Consequently, the topic of concrete creep must undergo a probabilistic treatment, with models calibrated to minimize the error between observations of long-term deformations and the predicted values.

The matter is further complicated by the prevalence of modern admixtures (e.g. plasticizers) which result in material behavior that deviates from historical observations, limiting the usefulness of long-time deformation measurements of old structures which were constructed in the absence of such admixtures. This results in a demand for more advanced and all-encompassing models. In any case, the datasets needed to produce satisfactory values for each model's parameters are enormous.

In the present study, a selection of three models in the literature was used to generate the long-term (i.e. 50-year duration) deformation curve of a hypothetical box-girder bridge with known geometry and material properties (Figure 1). This stage was followed by setting up a simple model in a custom finite element analysis program with material parameters that were iteratively calibrated with respect to the deformation curve generated by each of the three model codes. This was achieved by using the Kelvin model of a viscoelastic chain as described in [1].
The remainder of this paper examines the model codes and the process of calibrating a viscoelastic chain so as to approximate the creep curves predicted by each.

Literature Review: Creep Models

Compliance Function

The mathematical description of creep begins at (1) which separates the concrete strain into two components: the instantaneous strain, which depends on the elastic modulus at the time of loading, and the creep strain, the calculation of which is based on the concept of the creep coefficient, $\varphi(t, t0)$.

The creep coefficient is a concept adopted by all major concrete models and is usually (although other, less used variants exist) defined as the ratio of the additional, monotonically increasing creep strain to the initial strain induced by the first application of load. Since it is a strain value, it involves the use of an elastic modulus which - by convention - is defined as the mean value of the 28-day elastic modulus of concrete. This is a simple recognition of the well-known fact that the mechanical properties of concrete are time dependent. A direct implication of this fact is that creep is never fully reversible when the load is removed by which point the elastic properties of the material have changed.

$$\varepsilon_{c,total}(t) = \varepsilon_{ci}(t) + \varepsilon_{cc}(t, t_0) \qquad (1)$$

$$\varepsilon_{cc}(t) = \frac{\sigma_c(t_0)}{E_{ci}} \varphi(t, t0) \qquad (2)$$

$\varepsilon_{cc}(t)$ in (2) is the time-dependent creep strain of concrete. Bundling these two components into a singular expression and setting the induced stress $\sigma_c(t_0) = 1\, F \cdot M^{-2}$, we obtain the compliance function $J(t, t_0)$, which describes the strain response of concrete per unit stress. Within the range of applicability of these models, stress is linearly related to the strain via the compliance function in the form of (4). It should be noted that all three of the model codes examined in this paper [1,2,3] agree that this proportionality limit (that is, where strain is linearly related to stress via the compliance function) is exceeded when the compressive stress in the concrete core exceeds ~40% of its ultimate value.

$$\varepsilon_{c,total}(t) = \varepsilon_{ci}(t) + \varepsilon_{cc}(t, t_0) \qquad (3)$$

$$\varepsilon_{c,total}(t) = \sigma(t_0) \cdot \left[\frac{1}{E_{ci}(t_0)} + \frac{\varphi(t,t0)}{E_{ci}}\right] = \sigma(t_0) \cdot J(t, t_0) \qquad (4)$$

Furthermore, the creep coefficient is split up into two components: the basic creep coefficient $\varphi_{bc}(t, t0)$, which is what is measured on sealed concrete specimens, and the drying creep coefficient $\varphi_{dc}(t, t0)$, which is the additional strain caused by the drying process [1]. This phenomenon of concrete creep increasing in the presence of drying (i.e. when exposed to environmental conditions, as is almost always the case) was first observed by G. Pickett in the tests [7] conducted on specimens loaded under different humidity conditions as is shown in Figure 1 taken from [1], where,

D: drying specimen, no load

L: loaded specimen, no drying (sealed specimen – **basic creep**)

LD: loaded specimen in the presence of drying

LC: loaded specimen subjected to cycles of drying and re-wetting

The specimens were plain concrete beams of square cross section (with side 50.8 mm, span 813 mm, and midspan load 222 N) [7]. While the specimen series **L, LD & LC** were identically loaded, the two specimen-series which included drying (**LD & LC)** exhibited increased deflection because of the movement of the interlayer water within the concrete core [1, 7].

The additional creep strain accumulated when concrete is simultaneously drying necessitates the use of an additional creep component ("drying creep" as previously mentioned) whose mathematical description has been defined by all three model codes in the form of (5).

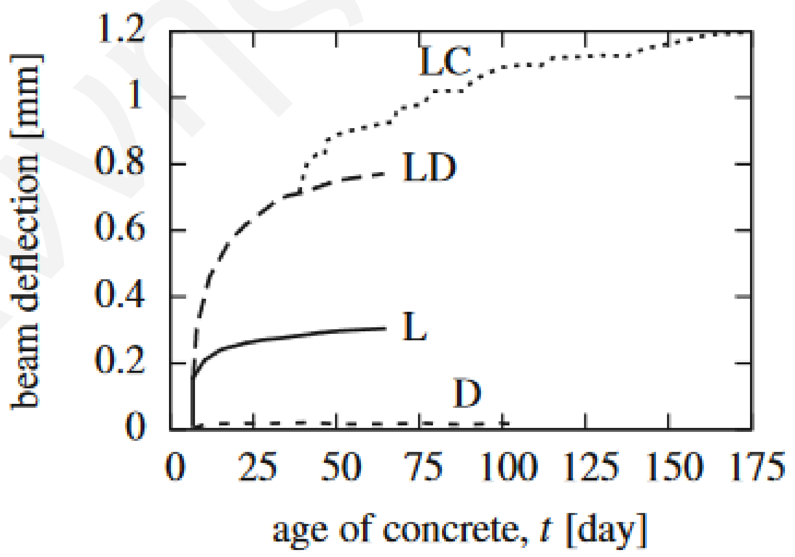


Figure 1

$$\varphi(t, t0) = \varphi_{bc}(t, t0) + \varphi_{dc}(t, t0) \quad (5)$$

CEB-FIB 2010

The CEB-FIB 2010 Model Code was authored by the International Federation for Structural Concrete to serve as a basis for code-making committees for concrete structures and related structural materials. This section summarizes the concrete creep model provided therein and its relevant input parameters.

The creep model provided by CEB-FIB2010 is applicable [2] to ordinary concrete (mean compressive strength in the range 20MPa – 130MPa) cured in moist conditions at normal temperatures and environmental humidity (5°C to 30°C). The model breaks down outside of this ranges. It is also assumed that the stress in the concrete is less than 40% of its ultimate strength; past this point, creep becomes non-linear, and its mathematical description is more involved.

Symbol	**Depends on**				**Reference**
s	Strength class of cement				[2] Table 5.1-9
	32.5N	32.5R, 42.5N	42.5R, 52.5N, 52.5R		
	0.38	0.25	0.20		
a_E	Aggregate Type				[2] Table 5.1-6
	Basalt	Quartzite	Limeston e	Sandston e	
	1.2	1.0	0.9	0.7	
f_{ck}	Characteristic strength in MPa				[2] Table 5.1-3
E_{ci}	Elastic Modulus at 28 days ([2] convention for creep coefficient)				[2] Eq. 5.1-56
E_{c0}	reference elastic modulus for quartzite aggregate: $21.5 * 10^3 MPa$				[2] Eq. 5.1-60
t_0	age at loading in days				[2] Eq. 5.1-60
RH	Relative Humidity as a percentage				[2] Eq. 5.1-56
A_c	Cross sectional area of Beam				[2] Eq. 5.1-56
u	Perimeter of cross section exposed to the atmosphere				[2] Eq. 5.1-70

Table 1 – Input Parameters

Symbol	Depends on	Reference
$\boldsymbol{\varphi(t, t0)}$	Creep Coefficient	[2] Eq. 5.1-56
$\boldsymbol{\beta_{cc}(t)}$	Concrete strength as a function of time	[2] Eq. 5.1-51
$\boldsymbol{\beta_{E}(t)}$	Elastic Modulus as a function of time	[2] Eq. 5.1-56
$\boldsymbol{h}$	Representative cross-sectional depth	[2] Eq. 5.1-56

Table 2 - Derived quantities

(Eq. 6)

$$\mathrm{E_{ci}} = \mathrm{E_{c0}} \cdot \mathrm{a_E} \cdot \left(\frac{\mathrm{f_{ck}} + \Delta \mathrm{f}}{10}\right)^{\frac{1}{3}}$$

$$\Delta f = 8\, MPa$$

$$\mathrm{E_{ci}(t)} = \beta_\mathrm{E}(\mathrm{t}) \cdot \mathrm{E_{ci}}$$

$$\beta_\mathrm{E}(\mathrm{t}) = [\beta_\mathrm{CC}(\mathrm{t})]^{0.5}$$

$$\beta_\mathrm{CC}(\mathrm{t}) = \exp\{\mathrm{s} \cdot [1 - \left(\frac{28}{\mathrm{t}}\right)^{0.5}]\}$$

$$\varphi_\mathrm{bc}(\mathrm{t}, \mathrm{t0}) = \beta(f_{cm}) \cdot \beta_{bc}(t, t0)$$

$$\varphi_\mathrm{dc}(\mathrm{t}, \mathrm{t0}) = \beta_{dc}(f_{cm}) \cdot \beta(RH) \cdot \beta_{dc}(t_0) \cdot \beta_{dc}(t, t_0)$$

Where,

$$\beta(f_{cm}) = \frac{1.8}{(f_{cm})\text{^}0.7}$$

$$\beta_{bc}(t, t0) = \ln\left(\left(\frac{30}{t_0} + 0.035\right)^2 \cdot (t - t0) + 1\right)$$

$$\beta_{dc}(f_{cm}) = \frac{412}{f_{cm}^{1.4}}$$

$$\beta(RH) = \frac{1 - \frac{RH}{100}}{\sqrt[3]{0.1 \cdot \frac{h}{100}}}$$

$$\beta_{dc}(t_0) = \frac{1}{0.1 + t_0^{0.2}}$$

$$\beta_{dc}(t, t_0) = \left[\frac{t - t_0}{\beta_h + (t - t_0)}\right]^{\gamma(t_0)}$$

$$\beta_h = \min\{1.5 \cdot h + 250 a_{fcm}, 1500 a_{fcm}\}$$

$$a_{fcm} = (35/f_{cm})^{0.5}$$

$$\gamma(t_0) = \frac{1}{2.3 + 3.5/\sqrt{t_0}}$$

$$h = 2 \cdot \frac{A_c}{u}$$

ACI 209

The ACI 209 Report, published in May 2008 by the American Concrete Institute is appropriately named “Guide for Modeling and Calculating Shrinkage and Creep in Hardened Concrete” and is the model adopted by the current AASHTO LFRD specifications. It should be noted **[3]** that the presence of admixtures such as silica fume, fly ash and natural pozzolans completely invalidates the predictions of this model.

Symbol	Depends on	Reference
N/A	Curing Conditions: Moist vs Steam	[3] Eq. A-22,23
s	Concrete slump	[3] Eq. A-28
f_{28}	Conventional strength at 28 days	[3] Eq. A-17
$E_{c,t0}$	Elastic modulus at the time of loading, t0	[3] Eq. A-18
ψ	Fine aggregate percentage	[3] Eq. A-29
t_0	age at loading in days	[3]
h	Relative Humidity	[3] Eq. A-24
V/S	Volume to exposed surface ratio: for a prismatic beam equivalent to the ratio Ac/u	[3] Eq. A-24
u	Perimeter of cross section exposed to the atmosphere	[3] Eq. A-24
α	Air content as a percentage	[3] Eq. A-30
γ_c	Unit weight of concrete (not to be confused with creep correction factor)	[3] Eq. A-16

Table 3 - Input Parameters

Symbol	Depends on	Reference
f_{c,t_0}	Concrete strength at the time of loading	[3] Eq. A-17
$E_{c,t0}$	Elastic modulus at the time of loading, t0	[3] Eq. A-16
γ_c	Creep correction factors	[3] Eq. A-16

Table 4 – Derived quantities

(Eq. 7)

$$\varphi_u = 2.35\gamma_c$$

$$\gamma_c = \gamma_{c,t0} \cdot \gamma_{c,RH} \cdot \gamma_{c,vs} \cdot \gamma_{c,s} \cdot \gamma_{c,\psi} \cdot \gamma_{sh,a}$$

$$\gamma_{c,t0} = 1.25 t_0^{-0.118} \qquad \gamma_{c,RH} = 1.27 - 0.67 \cdot \max\{h, 0.40\}$$

$$\gamma_{c,vs} = \frac{2}{3}(1 + 1.13 e^{\{-\frac{0.54V}{s}\}}) \qquad \gamma_{c,s} = 0.82 + 0.067s$$

$$\gamma_{c,\psi} = 0.88 + 0.0024\psi \qquad \gamma_{sh,a} = 0{,}43 + 0{,}09\alpha \geq 1$$

The value $\mathbf{2.35}$ in (7) is the asymptotic limiting value of the creep coefficient, meaning that the value of the creep strain is $\varphi_u = 2.35\gamma_c$ times the value of the instantaneous (elastic strain). As expected, this value is affected by the composition of the concrete mix, as indicated by the correction factor γ_c. Table 4 shows the parameters that determine the correction factors (through its partial factors).

At any point in time, the value of the creep coefficient is calculated by equation A-18 of the ACI 209 Report.

$$\varphi(t, t_0) = \frac{(t - t_0)}{f + (t - t_0)} \cdot \varphi_u \qquad (8)$$

$$\text{where } f = 26.0e\text{^}\{0.36(V/S)\} \qquad (9)$$

From (9) it is evident that the creep coefficient asymptotically approaches its ultimate value φ_u calculated in (7) according to the composition of the concrete mix and the element geometry.

The stress response of the material is (similar to CEB-FIB2010) calculated through the compliance function.

$$J(t, t_0) = \frac{1 + \varphi(t, t_0)}{E_{c,t0}}$$

where is the instantaneous concrete elastic modulus at the time of loading, given by

$$E_{c,t_0} = 33 \cdot \gamma_c^{1.5} \sqrt{f_{cm,t_0}}$$

Bazant's B3 Model

The B3 Model is one of the most recent models in the literature and is the result of the collaboration between Zdeněk P. Bažant and Milan Jirásek as presented in their joint book [1]. The model is based upon Bazant's solidification theory [1] and its form is summarized as such:

Symbol	Depends on			Reference
a_2	Curing Conditions			[1] C.17 pg. 714
	Steam	Normal	Moist (submerged in water)	
	0.75	1.2	1.0	
a_1	Cement type			[1] C.17 pg. 714
	Type I	Type II	Type III	
	1.0	0.85	1.1	
k_s	Shape Factor			[1] pg. 44
	Slab	Cylinder	Square Prism	
	1.00	1.15	1.25	
V/S	Volume to exposed surface ratio: for a prismatic beam equivalent to the ratio Ac/u			[1] C.17 pg. 711
f_c	Mean compressive strength			[1] C.17 pg. 711
w	Water content (mass) in concrete mix			[1] C.17 pg. 711
c	Cement content (mass) in concrete mix			[1] C.17 pg. 711
a	Aggregate content (mass) in concrete mix			[1] C.17 pg. 711
h_{env}	Relative Humidity			[1] C.17 pg. 711
t_0	Age of concrete at the time of loading, in days			[1] C.17 pg. 711
t'	Age of concrete at the end of curing, in days			[1] C.17 pg. 711

Table 5 – Input Parameters

The compliance function is given by

$$J(t,t_0) = \mathrm{q}_1 + \mathrm{q}_2 Q(t,t_0) + \mathrm{q}_3 \ln[1+(t-t_0)^{0.1}] + \mathrm{q}_4 \ln\left(\frac{t}{t_0}\right) + \mathrm{q}_5\sqrt{e^{-g(t-t')} - e^{-g(t-t_0)}} \qquad (10)$$

The auxiliary functions are calculated as:

(Eq. 11)

$$Q(t,t_0) = Q_f(t_0)\left[1+\left(\frac{Q_f(t_0)}{Z(t,t_0)}\right)^{r(t_0)}\right]^{-1/r(t_0)}$$

$$r(t_0) = 1.7\cdot(t_0)^{0.12} + 8$$

$$Z(t,t_0) = (t_0)^{-0.5}\ln\left[1 + (t - t_0)^{0.1}\right]$$

$$Q_f(t_0) = \left[0.086(t_0)^{2/9} + 1.21(t_0)^{4/9}\right]^{-1}$$

$$\varepsilon_{sh} = -\varepsilon_{sh}^{\infty}(1 - h_{env}^3) \cdot S(t - t_0)$$

$$\varepsilon_{sh}^{\infty} = 0.57514 \cdot a_1 \cdot a_2(0.019w^{2.1}f_c^{-0.28} + 270) \cdot \sqrt{3 + \frac{14}{(t_0 + k_t(k_s \cdot D)^\wedge 2\,)}}$$

$$g(\Delta t) = 8 \cdot [1 - (1 - h_{env}) \cdot S(\Delta t)]$$

$$S(\Delta t) = tanh\sqrt{\frac{\Delta t}{k_t(k_s \cdot D)^\wedge 2}}$$

$$k_t = 0.085t_0^{-0.08} \cdot f_c^{-1/4}$$

Where the value of k_s is found in Table 6.

The constant coefficients q_1 through q_5 **[1]** all have the units of MPa^{-1} and are given by:

$$\mathrm{q}_1 = 126.77f_c^{-0.5} \cdot 10^{-6}$$

$$\mathrm{q}_2 = 185.4 \cdot c^{0.5} \cdot 10^{-6}$$

$$\mathrm{q}_3 = 0.29\left(\frac{w}{c}\right)^4 \cdot \mathrm{q}_2$$

$$\mathrm{q}_4 = 20.3 \cdot \left(\frac{a}{c}\right)^{-0.7} \cdot 10^{-6}$$

$$\mathrm{q}_5 = 7.57 \cdot f_c^{-1}(\varepsilon_{sh}^{\infty})^{-0.6} \cdot 10^{-1}$$

Comparison between Models

All models recognize the inverse correlation between the age at loading, and the developed creep strains as the most significant factor. To account for the drying component of creep, the relative humidity and the curing conditions (e.g. normal ("moist") curing, steam curing) also affect the model equations.

All three models exhibit a close dependence on the composition of the concrete mix – the ACI209 model requires a more thorough knowledge of the fresh mix's properties such as its air content as well as the slump, and all three models are affected by the type (and quantity) of aggregate used. Recognizing that these values may not be entirely known to the designer at the time of modelling (whether it be an existing structure or a new one), the ACI209 Report is based on typical values of these parameters; the designer may use the creep correction factors to account for each special case.

Concurrently, all three models consider the size of the concrete specimen through its effective depth meaning the ratio of exposed volume to exposed specimen surface.

Generation of Deformation Curves

The input parameters and equations for each model were input into an Excel Spreadsheet to generate the total deformation of our specimen for the next 50 years (18,250 days). The deformation curve generated in each case corresponded to unit stress, so essentially what was generated was a graph of the compliance function. As previously mentioned, concrete creep is linear with respect to stress until the stress reaches around 40% of its ultimate value [1].

The horizontal axis (time) was set to a logarithmic scale in order to accurately visualize the evolution of creep for the target time span, from the first application of load at 8 days, kept constant for the next 50 years.

The parameters that concern the concrete composition (water, cement and aggregate content) were taken from a sample mix [5] and the element geometry (Figure 1) as well as the initial loading at the 8th day, were taken from [4].

Model Inputs

	CEB2010		**ACI209**	**B3**	
Strength class	42.5R	s = 0.20	N/A	Type III Cement	$a_1 = 1.1$
Aggregate type	Limestone	$a_E = 0.9$	N/A	N/A	
Aggregate content	N/A		N/A	$a = 1909\,\frac{kg}{m^3}$	
Fine Aggregate %	N/A		32.6%	N/A	
Air content %	N/A		6%	N/A	
Slump	N/A		69 mm	N/A	
Relative humidity	0.7		0.7	0.7	
Age at loading, t_0	8 days		8 days	8 days	

Representative depth, h	184mm	184mm	184mm
Water content, w	N/A	N/A	115.2 kg/m^3
Cement content, c	N/A	N/A	288.3 kg/m^3
Aggregate content, a	N/A	N/A	1758.4 kg/m^3
Compressive strength	4 ksi (28MPa)	4 ksi (28MPa)	4 ksi (28MPa)

Table 6 – Model inputs

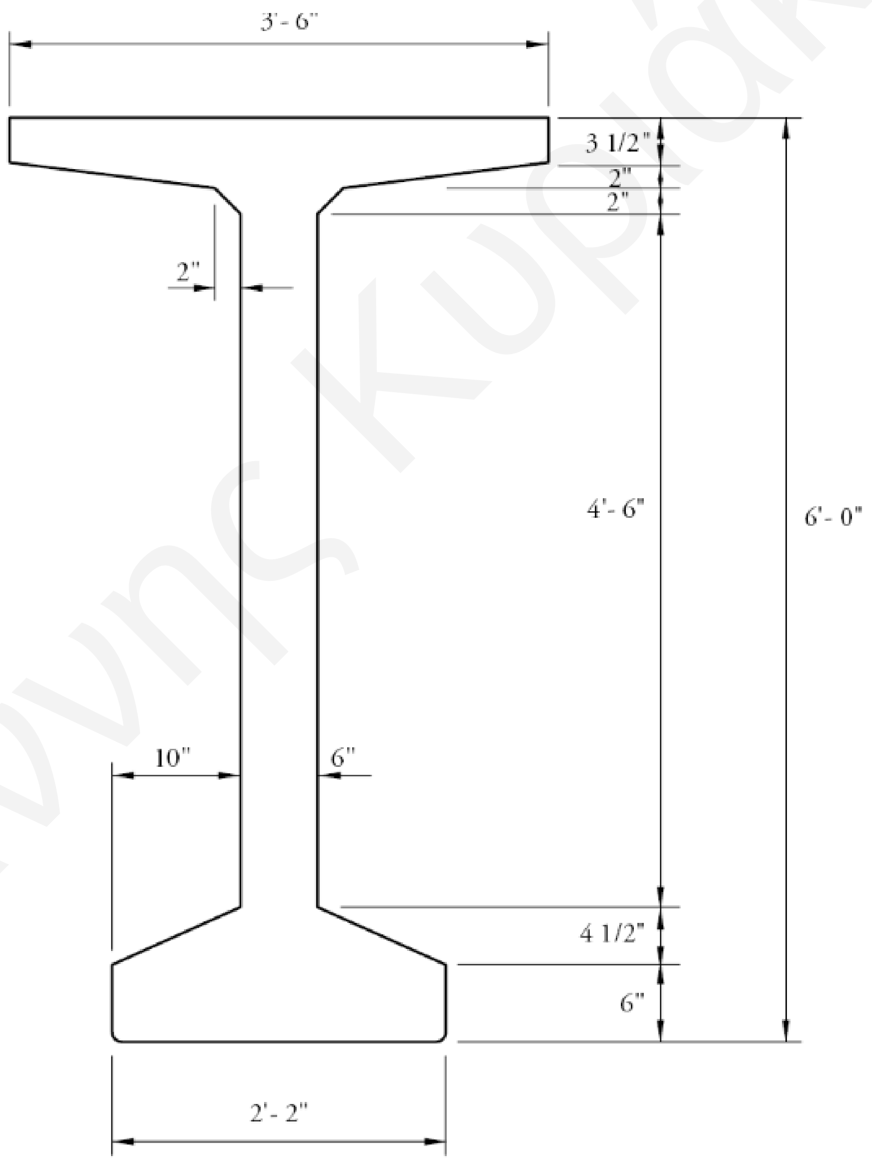


Figure 2

For the model inputs in Table 6, the final values of the compliance functions (strains corresponding to unit stress) for each model (after 50 years of sustained loading) are as such:

B3	**CEB2010**	**ACI209**
$10.8 \cdot 10^{-7}$	$9.15 \cdot 10^{-7}$	$7.09 \cdot 10^{-7}$

Table 7 – Final Strain values

The greatest creep deformation is predicted by the B3 Model, in which creep is assumed to not have a finite bound but instead increases indefinitely in a logarithmic manner. ACI209 was expected to yield the lowest value given that the model is set up in such a way as to approach a given asymptotic value.

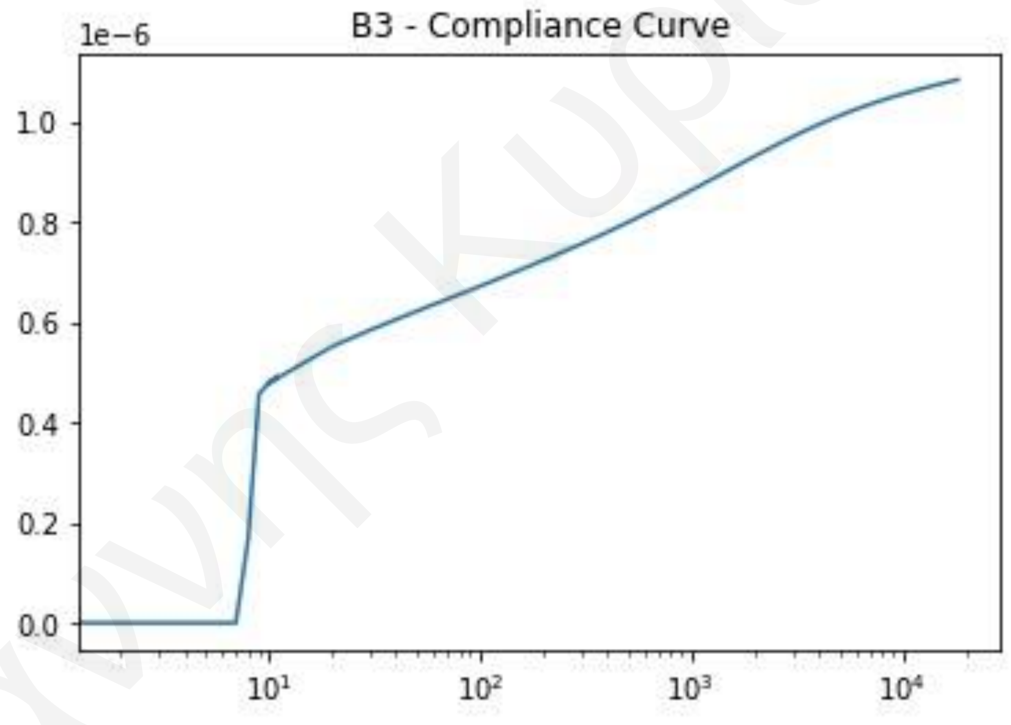


Figure 3

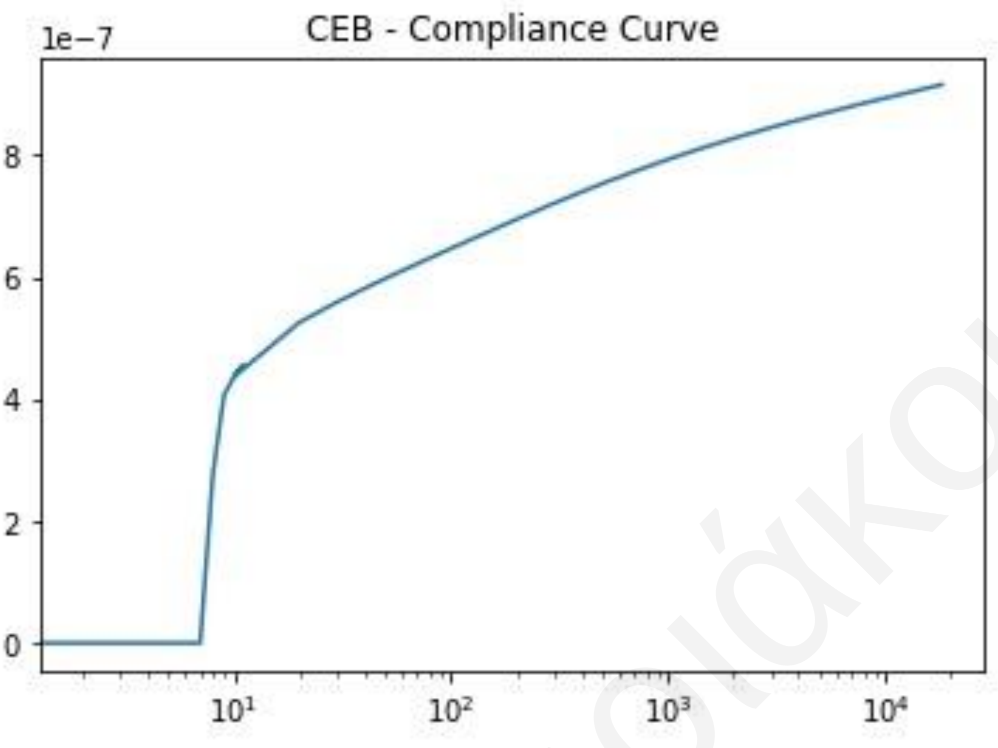


Figure 4

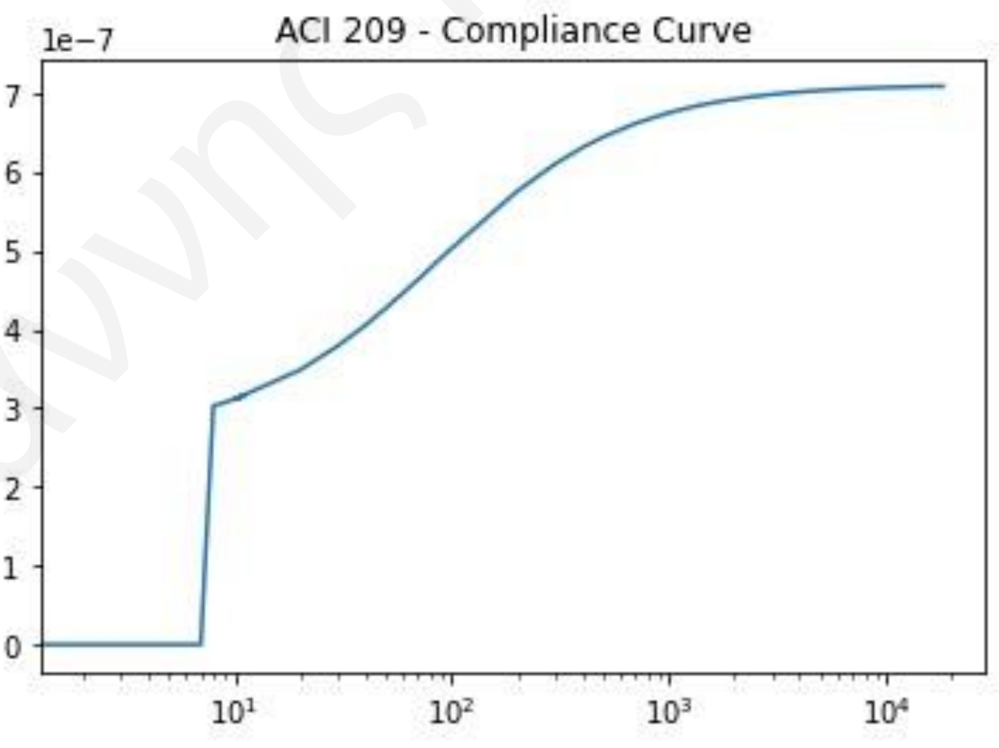


Figure 5

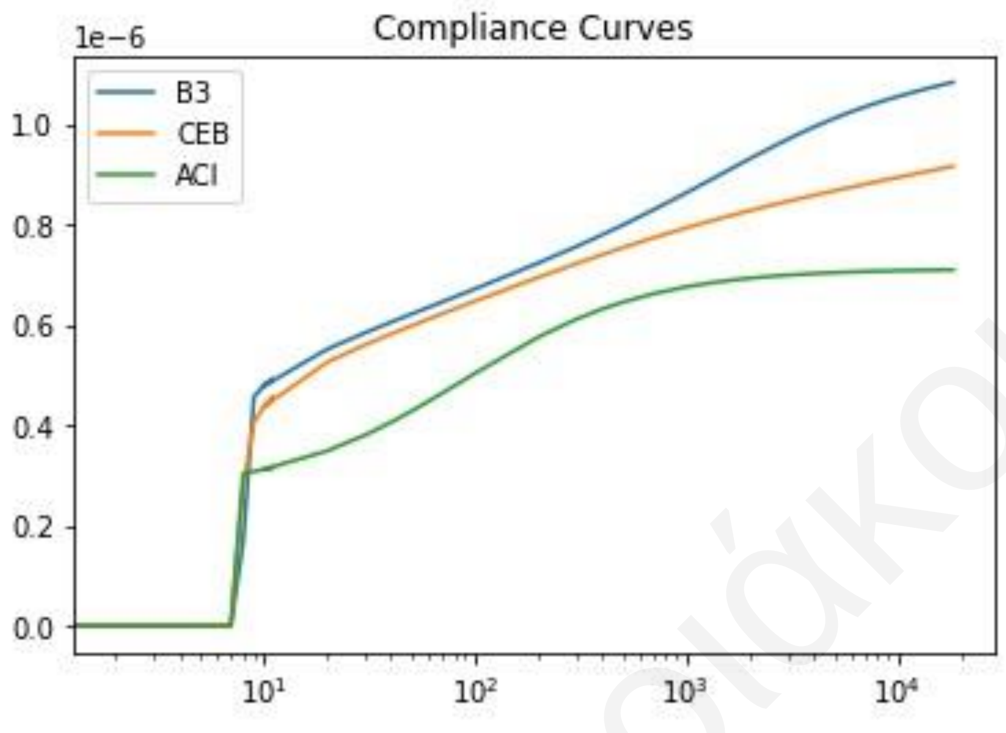


Figure 6

Particle Swarm Optimization

As mentioned in **[1]**, concrete is an aging material that belongs to the realm of linear viscoelasticity; in simple terms this means that concrete exhibits time-dependent deformation. In the present paper, the fundamental material model selected to describe the long-term deformations of concrete is the Kelvin chain, as was done by Bazant in **[1]** Fig. 6) and whose compliance function is given by (12). The derivation of the compliance function can be found in **[1]**.

$$\Phi(t) = \sum_{1}^{\mu} \frac{1 - e^{-\frac{t}{\tau_\mu}}}{E_\mu} \qquad (12)$$

Equation (12) is the sum of the compliances of the μ units that make up the chain; in the realm of viscoelasticity each unit has an *elastic* component and a *viscous* component working in parallel and respectively characterized by the elastic stiffness E_μ and viscosity η_μ. The ratio of the latter to the former is defined **[1]** as the *retardation time* of the unit, $\tau_\mu = \eta_\mu / E_\mu$, and has the appropriate units of time. It is a measure of the time period within which the particular unit in the chain has an appreciable influence on the total sum as implied by the negative exponential term in the numerator of the sum; for times t beyond the retardation time τ_μ the contribution of the unit vanishes rapidly.

According to Bazant's solidification model **[1]**, the compliance function $\Phi(t)$ of the Kelvin chain is influenced the solidification coefficient $v(t)$ via equation 13. The parameters α, λ_0, m are empirical constants.

A full exposition on the solidification theory and the derivation of its formulas can be found in **[1]**.

$$\frac{1}{v(t)} = 1 + \frac{1}{a}\left(\frac{\lambda_0}{t}\right)^m \qquad (13)$$

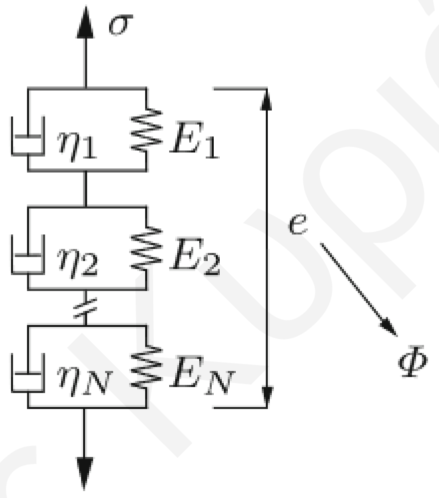


Figure 7

The goal is then to find the values of the parameters α, λ_0, m as well as the values for E_μ, τ_μ (corresponding to the μ units in the Kelvin chain) that best approximate the generated deformation (strain) values, leaving us with a total of $2\mu\,(i.e.\ E_\mu\ \&\ \tau_\mu\,) + 3(i.e.\ \alpha, \lambda_0, m\,)$ decision variables. In the present paper the number of units in the Kelvin chain was six, for a total of 15 decision variables. Following Bazant's **[1]** recommendation of separating the retardation times by an order of magnitude to cover as large a time period as needed, and given that the values resulting from the calibration were subsequently applied to the box-girder bridge for a 50-year duration (18250 days), it follows that the number of units in the chain is six, with the following fixed retardation times,

$$\tau_1 = 0.1, \tau_2 = 1.0, \tau_3 = 10.0, \tau_4 = 100.0, \tau_5 = 1000.0, \tau_6 = 10000.0\ days$$

The PSO (Particle Swarm Optimization) [6] algorithm was used for the purpose of calibrating the viscoelastic chain's response to match the generated creep compliance for the three model codes in the present study.

Algorithm Description

The Particle Swarm Optimization routine belongs to the class of meta-heuristic algorithms that iteratively improves a set of candidate solutions to a problem with respect to a given measure of quality. This family of algorithms also includes the well-known Genetic Algorithms which operate in a similar fashion.

In this case, the goal was to minimize the SRSS (Square Root of Squared Sums) error between the strain values predicted by the three models and the material's deformation response, as determined by the Finite Element analysis of a uniaxially loaded beam of unit cross-sectional area and length, so that deformations are equal to strains. The algorithm was appropriately named as such because it seeks to emulate the behavior of birds or fish (*particles*) that instinctively organize themselves in swarms in search of resources and whose individual movement is influenced (but not entirely determined) by the movement of the whole group.

Each *particle* represents a possible solution to the problem (a specific set of values for the $2\mu + 3$ decision variables) which is subsequently altered in each iteration if a superior set of values is found. To effectively exploit the computing power of the HPC supercomputer, a number of 400 particles and a number of 200 total iterations were selected to carry out the optimization, although a smaller number of particles can be used if computing resources are limited.

Algorithm Progression

In the very first iteration of the algorithm run (initialization phase) it is necessary to supply all particles with their initial values and determine the best (see section Measure of Fitness) solution among them. Once this is done, the swarming phase of the algorithm can begin.

In every subsequent iteration, the algorithm records the latest values of the decision variables for each particle i; this set of values is the particle's *position vector* $\boldsymbol{X^i}$ in the search space ($\mathbf{R^{2\mu+3}}$) which is subject to improvements in every iteration. Keeping up with the swarm analogy, it is then necessary to define the *velocity vector* $\boldsymbol{V^i}$ (15) which will determine the particle's next position in the search space according to (14) and after doing so, evaluate each new solution's fitness and record the most optimal vector of values achieved thus far as the particle's new personal best $(pbest)$. Finally, the best solution among the particles is selected as the swarm's global best $gbest$.

In the following equations the superscript i refers to the particle i and the index variable n refers to the current iteration of the algorithm; one can notice that the "global best" vector $gbest$ is not denoted by a superscript since it concerns the whole swarm rather than an individual particle.

The application of the two random coefficients r_1, r_2 in (15) assists the algorithm in reaching new solutions and not remain trapped in local minima (valleys) of the cost function used to evaluate the fitness of each solution, as is the case with some popular deterministic algorithms (e.g. Gradient Descent).

$$\boldsymbol{X^i(n+1) = X^i(n) + V^i(n+1)} \quad (14)$$

$$\boldsymbol{V^i(n+1) = wV^i(n) + c_1 r_1 \left(pbest^i - X^i(n)\right) + c_2 r_2 (gbest - X^i(n))} \quad (15)$$

The various parameters appearing in (15) can be summarized as such:

$$w \in [0,1]: inertia\ weight\ constant$$

As implied by its name, the inertia constant affects how much the previous velocity vector influences the new one, determining the extent to which the particle will deviate from its previous path.

$$c_1 \in [0,1]: cognitive\ (exploration) coefficient,$$

which determines how much the particle will remain tethered to its own personal best.

$$c_2 \in [0,1]: social\ (exploitation) coefficient,$$

which determines how much the particle will move towards the swarm's global best achieved thus far in the iteration.

$$r_1 \in [0,1]: random\ component$$

$$r_2 \in [0,1]: random\ component$$

The selection of the values for the coefficients w, c_1, c_2 – especially the interplay between the individual (c_1) and social (c_2) components – is a whole subject of study and beyond the scope of this paper. As previously mentioned, the values of r_1 and r_2 are randomly generated (from a uniform probability distribution in our). For the purposes of the calibration the following values were used:

$$w = 0.8, \qquad c_1 = 0.7, \qquad c_2 = 0.5$$

The inertial and individual components, w and c_1, are appropriately larger than the social component c_2 to take advantage of the large number of particles in finding new, potentially superior, solution sets.

Ranges of Values for the Decision Variables

As mentioned above, the retardation times were fixed to their sequential values, starting from the value of 0.1 days, and increasing by an order of magnitude for each subsequent unit, as is recommended by Bazant in **[1]**. The elastic moduli were granted an enormous range of possible values - namely between 0.1 and 1000 - since the number of particles itself was quite large; this would prevent the algorithm from remaining trapped in locally optimal solutions.

The remaining three decision variables stemming from the solidification theory were allowed the following ranges of values:

$$\alpha \in [0.00001, 0.1], \qquad \lambda_0 \in [0.001, 10], \qquad m \in [0.1, 2]$$

While seemingly random, it must be noted that the upper and lower bounds for these variables were progressively modified throughout the various calibration runs to improve the output of the algorithm. They are empirical constants with no particular physical meaning.

Measure of Fitness

The goal of each iteration was to evaluate the quality of the newfound solutions, namely the set of values for the decision variables. For this task, it is necessary to define a *target* and a method of determining the “closeness” of the candidate solution to said target. In the present study, the target for each calibration run was the generated strain curve originating from the three model codes examined, and the measure of closeness of a candidate solution to the target was the error between the values predicted by the model curves and the actual material response (as determined by the Finite Element Analysis of the uniaxially loaded beam equipped with the Kelvin chain compliance function). Specifically, the Square Root of Squared Sums of the differences (errors) was used. This measure of closeness is referred to in the literature as the *cost* *function, objective function, etc.,* and the goal in this case was to select the vector of values that minimizes its value i.e. best approach the target curves.

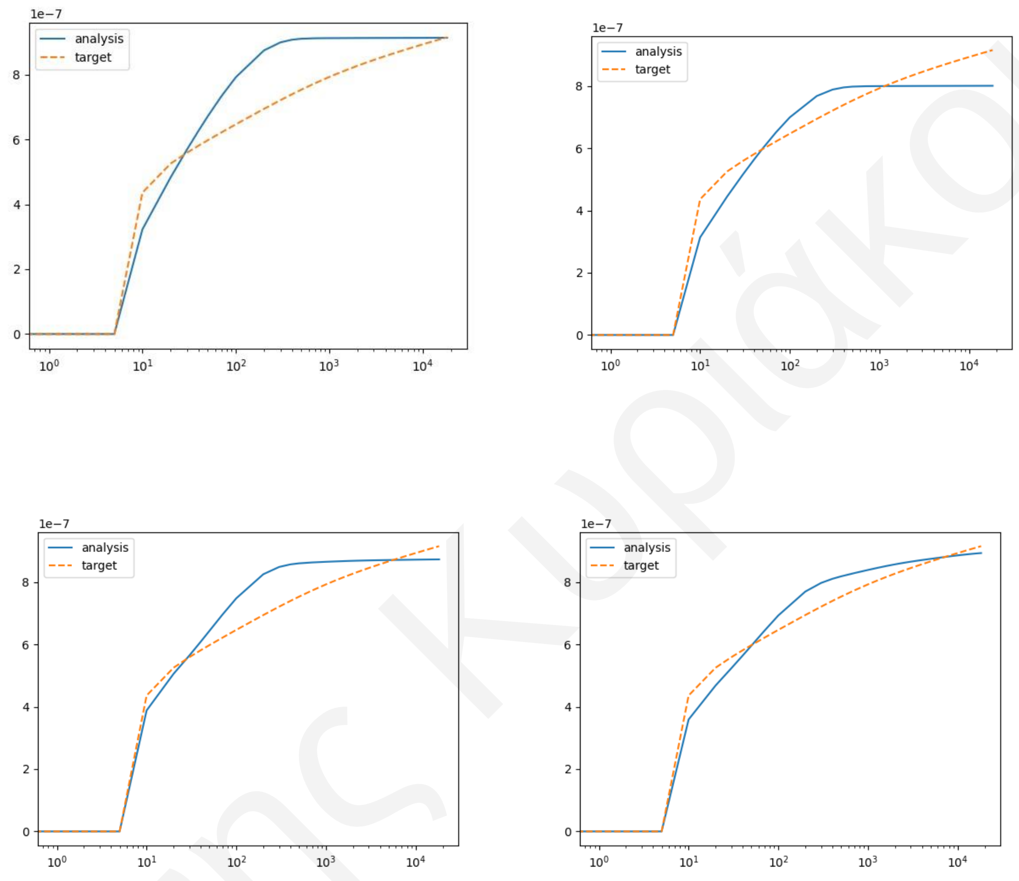
Python Source Code

The Python code to perform the optimization was split up into five source files. Due to the memory and processing-intensive nature of the algorithm, it was executed in a multi-threaded environment using the university's HPC supercomputer; the parameters determined at each iteration were fed through a Finite Element Analysis program, which applied a unit load and recorded the deformation history. To carry out the calibration algorithm and to produce the necessary visualizations of the various results, several well-known Python libraries were used e.g. *numpy, pickle, matplotlib* as well as several built-in libraries to handle the generated data files. The actual Python Source code is included in Appendix A.

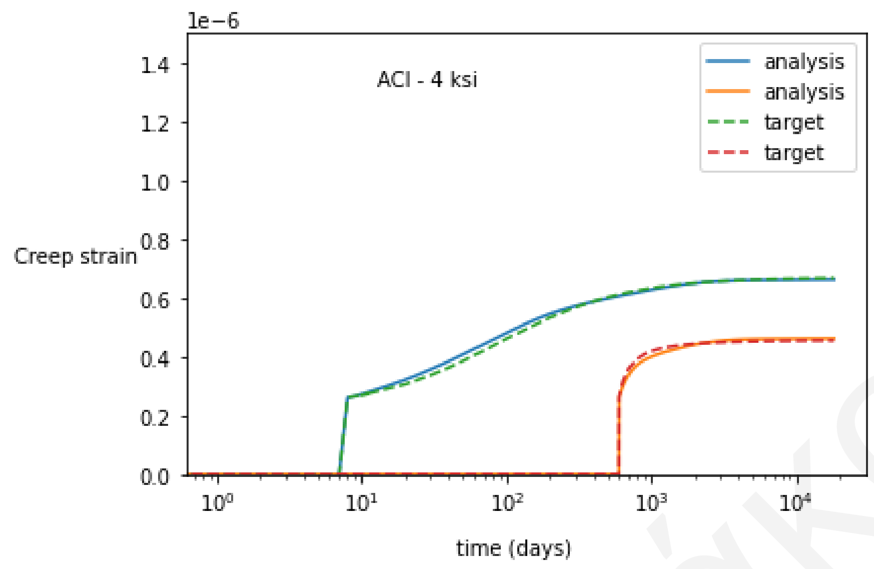
Calibration Results

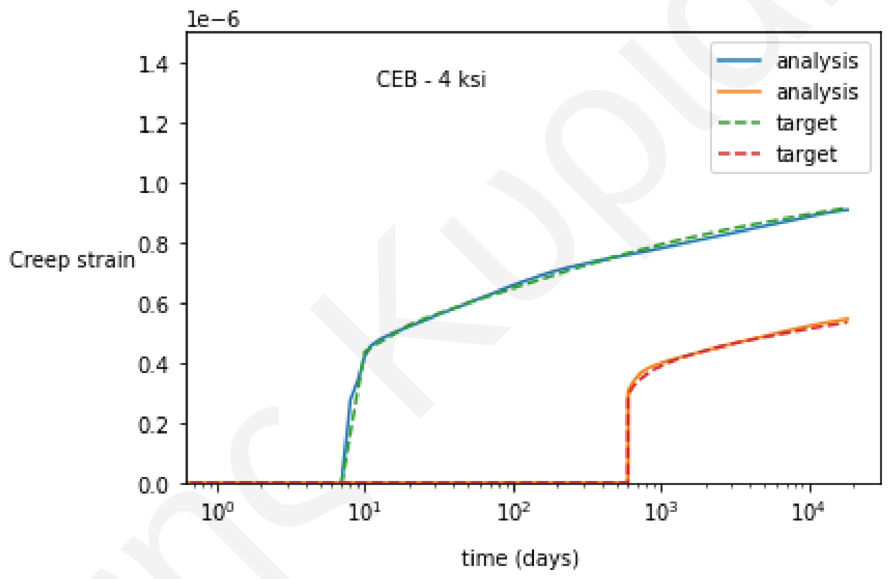
In each iteration the algorithm modifies the decision variables to gradually approach the target curve (dashed line in figures 8, 9, 10, 11 labeled 'target'). Figures 8, 9, 10, 11 showcase some sample steps for each iteration and the progressive improvement of our Kelvin chain as it approaches the deformation curve generated by each model. Because of the long-duration nature of the analysis and the fact that concrete creep displays logarithmic growth [1, 2, 3, 7] it is necessary to plot the time axis (x-axis) in logarithmic scale to properly visualize the evolution of the compliance function.

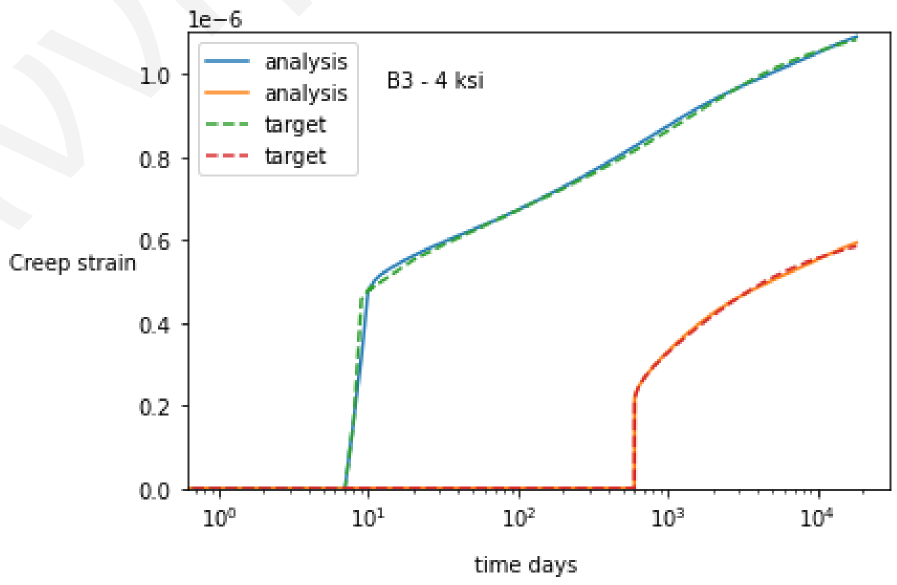
The final strain values (instantaneous + creep strain) for all three models are shown in Figures 12, 13, 14 where the calibration was performed for loading at ages 8 (green lines) and 600 days (red lines). In every case, there was a better approximation when the model was loaded at 600 days, by which point the concrete compliance functions "level off" due to the extensive maturing of the material. The material response taken from the analysis is represented by the continuous line, which in every case is accompanied by the target it attempts to approach, represented by the dashed line.



Figures 8-11 – Progressive Calibration of Kelvin Chain







Figures 12-14 – Final Calibration Results

Analysis of Prestressed Beam

The final values of the chain parameters taken from the calibration were then used as the concrete constitutive law in the three-span prestressed beam model [4] and the relaxation of the tendons due to the creep deformation of the concrete beams was recorded. The longitudinal profile and cross section of the bridge are shown in Figure 15, and the evolution of the tendon stress for each model is shown in Figure 16. As predicted, the accumulation of deformation in the concrete induces relaxation of the pre-stressed tendons; the model calibrated according to the B3 Model exhibits the most tendon relaxation of all due to the increased creep strain.

Figure 9.2.1-1
Longitudinal Section

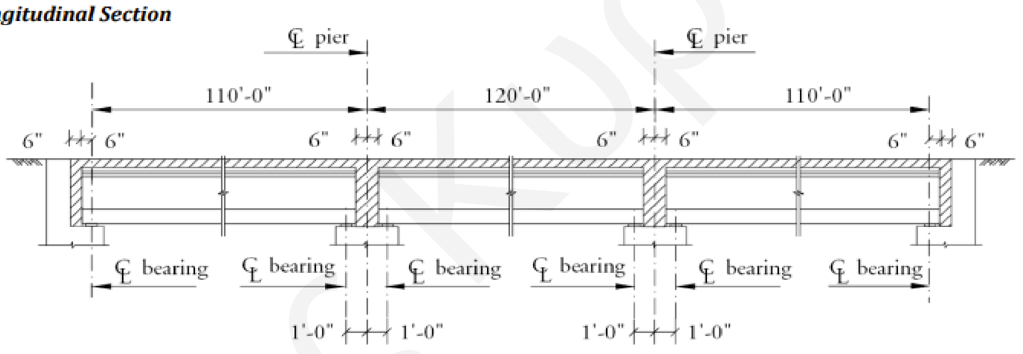


Figure 9.2.1-2
Bridge Cross Section

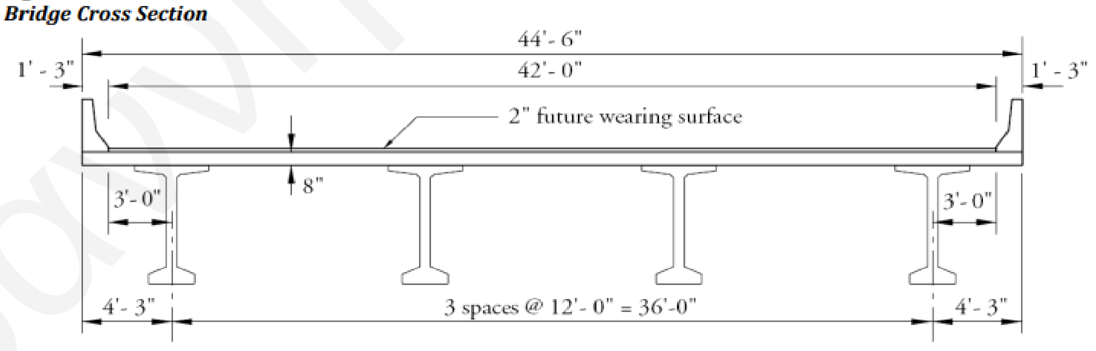


Figure 15 – Example bridge profile (PCI Manual)

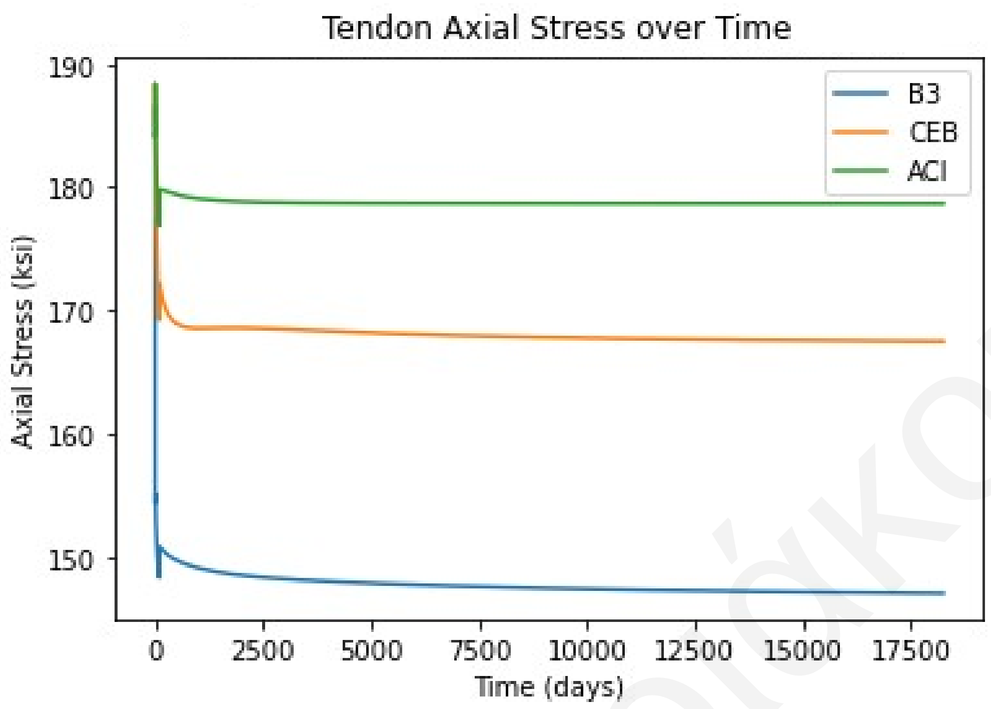


Figure 16

Conclusions

There seems to be a good agreement between the creep deformations predicted by the model codes and the final parameters of the Kelvin chain, which undergoes aging according to the viscoelastic model and Bazant's solidification theory [1]. These models could thus be incorporated into a finite element analysis software package to predict the time-dependent deformations of creep-sensitive structures such as nuclear tanks and bridges, where it could furthermore be used to calculate the prestress losses of post-tensioning steel and so on.

Appendix A: Python Source Code

```
import os
import shutil
import subprocess
import numpy as np
import sys
import copy
import random
import math
import time
import pickle
from generate_runs import run_multiphys
import matplotlib.pyplot as plt
import matplotlib
curDir = os.getcwd()
#**********************
# LOAD INPUTS
#**********************
os.chdir(curDir)
import task_inputs
taskName = task_inputs.taskName
numVariables = task_inputs.numVariables
numParticles = task_inputs.numParticles
vmax = task_inputs.vmax
vmin = task_inputs.vmin
wmax = task_inputs.wmax
wmin = task_inputs.wmin
kmax = task_inputs.kmax
c1 = task_inputs.c1
c2 = task_inputs.c2
p1 = task_inputs.p1
p2 = task_inputs.p2
optOption = task_inputs.optOption
numProcesses = task_inputs.numProcesses
lt_1 = task_inputs.lt_1
lt_2 = task_inputs.lt_2
# infoNames = task_inputs.infoNames
#**********************
# MISC. SET UP
#**********************

n = 0
k = 1
DB = {}
DB[k] = {}
DB[k]['iter'] = k
fname = []
fname2 = []
runNumber = 'Null'
```

```
try:
    runNumber = sys.argv[1]
except IndexError:
    runNumber = '000'
    for i in range(1,numParticles+1):
    DB[k][i] = {}
    DB[k][i]['par'] = i
    DB[k][i]['V']=[]
#   DB[k][i]['locx']=[]
#   DB[k][i]['locz']=[]
    for j in range(0,numVariables):
        DB[k][i]['V'].append(copy.deepcopy(random.uniform(0.01, 1.)*vmax[j]))
    fname.append('Par'+str(i).zfill(3)+'i')
    fname2.append('Par'+str(i).zfill(3)+'a')
#**********************
# WRITE HEADERS TO .OUT FILES:
#**********************

Outfile = open('Run'+runNumber+'_'+taskName+'_rawC.txt','a')
Outfile.write('iter,par,f,pBest,gBest,g,')
for var in range(0, numVariables):
    Outfile.write('X['+str(var)+'],')
for var in range(0, numVariables):
    Outfile.write('V['+str(var)+'],')
# for item in infoNames:
#   Outfile.write(str(item)+',')
Outfile.write('\n')
Outfile.close()
#**********************
# INITIALIZE PARTICLES
#**********************
for i in range(0, 1+int(math.floor(max(numParticles,numProcesses)/numProcesses))):
    parRun = []
    for j in range(0, numProcesses):
        parRun.append(int(i*numProcesses+j+1))
        if(parRun[j]<=numParticles):
            execString = ''
            execString = execString + 'pp' + str(parRun[j]).zfill(3) + ' = '
            execString                =                execString              +
"subprocess.Popen(['python','initializeParticle.py', fname[parRun[j]-1]])"
            exec(execString)
            time.sleep(0.01)
    for j in range(0, numProcesses):
        if(parRun[j]<=numParticles):
            execString = ''
            execString = execString + 'out'+str(parRun[j]).zfill(3)+' = '
            execString = execString + 'pp'+str(parRun[j]).zfill(3)+'.wait()'
            exec(execString)
#**********************
# LOAD INITIALIZED PARTICLES FROM PICKLE FILES,
# SET UP DB DICTIONARY
```

```
#*********************
for i in range(1,numParticles+1):
    with  open(curDir+"/"+fname[i-1]+"/"+fname[i-1]+'.pkl', 'rb') as handle:
        DB[k][i]['f'] = pickle.load(handle)
        DB[k][i]['X'] = pickle.load(handle)
        m = pickle.load(handle)
        DB[k][i]['data'] = pickle.load(handle)
        n += m
        handle.close()
        DB[k][i]['pBest'] = DB[k][i]['f']
        DB[k][i]['XpBest'] = DB[k][i]['X']
#**********************
# Identify global best position
#**********************
DB[k]['gBest'] = DB[k][1]['f']
DB[k]['XgBest'] = DB[k][1]['X']
DB[k]['dataBest'] = DB[k][1]['data']
for i in range(1,numParticles+1):
    f = DB[k][i]['f']
    X = DB[k][i]['X']
    data = DB[k][i]['data']
    if (optOption.lower()=='minimize'):
        if f < DB[k]['gBest']:
            DB[k]['gBest'] = f
            DB[k]['XgBest'] = X
            DB[k]['dataBest'] = data
    elif (optOption.lower()=='maximize'):
        if f > DB[k]['gBest']:
            DB[k]['gBest'] = f
            DB[k]['XgBest'] = X
            DB[k]['dataBest'] = data
    else:
        print ('ERROR! optOption is not properly defined')
        exit()
#**********************
# Iteration loop:
#**********************

DB[k]['n'] = n
done = False
while not done:
    g_vmax = -1.
    for i in range(1,numParticles+1):
        X = DB[k][i]['X']
        f = DB[k][i]['f']
        data = DB[k][i]['data']
        # g = DB[k][i]['g']
        if (optOption.lower()=='minimize'):
            if f < DB[k][i]['pBest']:
                DB[k][i]['pBest'] = f
                DB[k][i]['XpBest'] = X
                DB[k][i]['dataBest'] = data
            if f < DB[k]['gBest']:
                DB[k]['gBest'] = f
                DB[k]['XgBest'] = X
                DB[k]['dataBest'] = data
        elif (optOption.lower()=='maximize'):
            if f > DB[k][i]['pBest']:
                DB[k][i]['pBest'] = f
                DB[k][i]['XpBest'] = X
                DB[k][i]['dataBest'] = data
            if f > DB[k]['gBest']:
                DB[k]['gBest'] = f
                DB[k]['XgBest'] = X
                DB[k]['dataBest'] = data
        else:
            print ('ERROR! optOption is not properly defined')
            exit()

        p_vmax = max([abs(x) for x in DB[k][i]['V']])
        if p_vmax > g_vmax:
            g_vmax = p_vmax

    #if g_vmax < vmin:
        #print 'g_vmax < vmin'
        #done = True
    if k > kmax:
        print ('k >= kmax')
        done = True
    if not done:
        DB[k+1] = {}
        DB[k+1]['iter'] = k+1
        w = wmax-(wmax-wmin)*k/kmax
        DB[k]['w'] = w
        for i in range(1,numParticles+1):
            DB[k+1][i] = {}
            DB[k+1][i]['par'] = i
            DB[k+1][i]['pBest'] = DB[k][i]['pBest']
            DB[k+1][i]['XpBest'] = DB[k][i]['XpBest']
            with open(fname2[i-1]+str(k).zfill(3)+'.pkl', 'wb') as output:
                pickle.dump(DB[k],output,protocol=pickle.HIGHEST_PROTOCOL)
            output.close()
        for                     i                     in                  range(0,
1+int(math.floor(max(numParticles,numProcesses)/numProcesses))):
            parRun = []
            for j in range(0, numProcesses):
                parRun.append(int(i*numProcesses+j+1))
                if(parRun[j]<=numParticles):
                    execString = ''
                    execString = execString + 'pp'+str(parRun[j]).zfill(3)+' = '
                    execString = execString +
"subprocess.Popen(['python','updateParticle.py', fname2[parRun[j]-
1]+str(k).zfill(3),"
                    execString = execString + "str(k), str("+str(parRun[j])+")])"
                    exec(execString)
                    time.sleep(0.01)
            for j in range(0, numProcesses):
                if(parRun[j]<=numParticles):
                    execString = ''
                    execString = execString + 'out'+str(parRun[j]).zfill(3)+' = '
                    execString = execString + 'pp'+str(parRun[j]).zfill(3)+'.wait()'
                    exec(execString)
                    time.sleep(0.01)
        for i in range(1,numParticles+1):
            with open(curDir+"/"+fname2[i-1]+"/"+fname2[i-
1]+str(k).zfill(3)+'.pkl', 'rb') as handle:
                DB[k+1][i]['f'] = pickle.load(handle)
                DB[k+1][i]['X'] = pickle.load(handle)
                DB[k+1][i]['V'] = pickle.load(handle)
                m = pickle.load(handle)
                DB[k+1][i]['data'] = pickle.load(handle)
                n += m
                handle.close()
                DB[k+1][i]['pBest'] = DB[k][i]['pBest']
                DB[k+1][i]['XpBest'] = DB[k][i]['XpBest']
                os.remove(fname2[i-1]+str(k).zfill(3)+'.pkl')
                pkl_there=False
        DB[k+1]['gBest'] = DB[k]['gBest']
        DB[k+1]['XgBest'] = DB[k]['XgBest']
        DB[k+1]['dataBest'] = DB[k]['dataBest']
        DB[k+1]['n'] = n
        # Plot the best solution
        target=np.loadtxt('targeted_plot_ACI_1.txt')
        #target[:,1]=target[:,1]-target[lt_1,1]

        fig1,ax1=plt.subplots()
        #      ax1.plot(DB[k]['dataBest'][0],DB[k]['dataBest'][1], linestyle='-
',label='analysis')
        ax1.plot(target[:,0],DB[k]['dataBest'][0][1], linestyle='-
',label='analysis')
        ax1.plot(target[:,0],target[:,1], linestyle='--',label='target')
        # ax1.set_ylabel('Base Shear (kN)')
        # ax1.set_xlabel('Drift Ratio(%)')
        # ax1.set_xlim([0,3 ])
        # ax1.set_ylim([0,400 ])
        ax1.legend()
        ax1.set_xscale('log')

        fig1.savefig(str(k)+'_1_.png')
        plt.close(fig1)
        target=np.loadtxt('targeted_plot_ACI_2.txt')
        #target[:,1]=target[:,1]-target[lt_2,1]

        fig1,ax1=plt.subplots()
        #      ax1.plot(DB[k]['dataBest'][0],DB[k]['dataBest'][1],      linestyle='-
',label='analysis')
        ax1.plot(target[:,0],DB[k]['dataBest'][1][1],                   linestyle='-
',label='analysis')
        ax1.plot(target[:,0],target[:,1], linestyle='--',label='target')
        # ax1.set_ylabel('Base Shear (kN)')
        # ax1.set_xlabel('Drift Ratio(%)')
        # ax1.set_xlim([0,3 ])
        # ax1.set_ylim([0,400 ])
        ax1.legend()
        ax1.set_xscale('log')
        #plt.show()
        fig1.savefig(str(k)+'_2_.png')
        plt.close(fig1)

        Outfile = open('Run'+runNumber+'_'+taskName+'_rawA.txt','a')
        Outfile.write('iter    =    %i,    n    =    %i,    gBest    =    %g'    %
(DB[k]['iter'],DB[k]['n'],DB[k]['gBest']))
        for var in range(0, numVariables):
                Outfile.write(', X['+str(var)+'] = ')
                Outfile.write(str(DB[k]['XgBest'][var]))
        Outfile.write('\n')
        Outfile.close()

        Outfile = open('Run'+runNumber+'_'+taskName+'_rawB.txt','a')
        Outfile.write('iter = %g, gBest = %g\n' % (DB[k]['iter'], DB[k]['gBest']))
        for i in range(1,numParticles+1):
            Outfile.write('par = ')
            Outfile.write(str(DB[k][i]['par']))
            Outfile.write(', f = ')
            Outfile.write(str(DB[k][i]['f']))
            Outfile.write(', pBest = ')
            Outfile.write(str(DB[k][i]['pBest']))
            for var in range(0, numVariables):
                Outfile.write(', X['+str(var)+'] = ')
                Outfile.write(str(DB[k][i]['X'][var]))
            for var in range(0, numVariables):
                Outfile.write(', V['+str(var)+'] = ')
                Outfile.write(str(DB[k][i]['V'][var]))
            Outfile.write('\n')
        Outfile.write('*****************************************************\n')
        Outfile.close()

        Outfile = open('Run'+runNumber+'_'+taskName+'_rawC.txt','a')
        for i in range(1,numParticles+1):
            Outfile.write(str(DB[k]['iter']))
            Outfile.write(',')
            Outfile.write(str(DB[k][i]['par']))
            Outfile.write(',')
            Outfile.write(str(DB[k][i]['f']))
            Outfile.write(',')
            Outfile.write(str(DB[k][i]['pBest']))
            Outfile.write(',')
            Outfile.write(str(DB[k]['gBest']))
            Outfile.write(',')
            for var in range(0, numVariables):
                Outfile.write(str(DB[k][i]['X'][var]))
                Outfile.write(',')
            for var in range(0, numVariables):
                Outfile.write(str(DB[k][i]['V'][var]))
                Outfile.write(',')
            Outfile.write('\n')
        Outfile.close()

        print('iteration='+str(k))
        k += 1
```

```
import os
import shutil
import subprocess
import time
from lasso.dyna import D3plot, ArrayType as dt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

def read_spc(name):
    time=[]
    data={}
    nodes=[]
    first=1
    f = open(name+"/spcforc", "r")
    for x in f:
        if x[1:7]=="output":
            time.append(float(x.split()[4]))
            first=first-1
        elif x[1:5]=="node" and x[25:31]=="forces" and first==0 :
            nodes.append(int(x.split()[1]))
            data[nodes[-1]]=[]
    f.close()

    return time

def read_deplot_disp(d3plot):
    #d3plot = D3plot(name+'/d3plot')
    disp = d3plot.arrays["node_displacement"]
    return disp
```

```
def drift(disp, node1,node2, direction,h):
    re_disp_1=[x[node1-1][direction]-disp[0][node1-1][direction] for x in disp]
    re_disp_2=[x[node2-1][direction]-disp[0][node2-1][direction] for x in disp]
    re_disp=[x-y for x, y in zip(re_disp_1, re_disp_2)]
    drift=[x/h for x in re_disp]
    return drift

def run_multiphys(X,filename,target,lt,f):
    t1 = X[0]
    t2 = X[1]
    t3 = X[2]
    t4 = X[3]
    t5 = X[4]
    t6 = X[5]
    e1 = X[6]
    e2 = X[7]
    e3 = X[8]
    e4 = X[9]
    e5 = X[10]
    e6 = X[11]
    a  = X[12]
    lam0  = X[13]
    m  = X[14]

    #Assemble swap array
    swap = []
    swap.append(('{t1}',str(t1)))
    swap.append(('{t2}',str(t2)))
    swap.append(('{t3}',str(t3)))
    swap.append(('{t4}',str(t4)))
    swap.append(('{t5}',str(t5)))
    swap.append(('{t6}',str(t6)))
    swap.append(('{e1}',str(e1)))
    swap.append(('{e2}',str(e2)))
    swap.append(('{e3}',str(e3)))
    swap.append(('{e4}',str(e4)))
    swap.append(('{e5}',str(e5)))
    swap.append(('{e6}',str(e6)))
    swap.append(('{a}',str(a)))
    swap.append(('{lt1}',str(lt)))
    swap.append(('{lt2}',str(lt-1)))
    swap.append(('{lt3}',str(50*365+1)))
    swap.append(('{lam0}',str(lam0)))
    swap.append(('{m}',str(m)))

    curDir = os.getcwd()
    newDir = os.path.join('.',str(filename))
    if not os.path.exists(newDir):
        os.makedirs(newDir)

    os.chdir(newDir)
    print(newDir)
    shutil.copy('../FE_MultiPhys_2023.01.11','./FE_MultiPhys_2023.01.11')
    shutil.copy('../readinp.txt','./readinp.txt')
    shutil.copy('../libiomp5md.dll','./libiomp5md.dll')
    shutil.copy('../slurm_script','./slurm_script')

    template=open('../creep_template.txt','r')
    runFile=open('creep_template.txt','w')

    trl = template.readline()
    while trl:
        for i in range(len(swap)):
            if str(swap[i][0]) in trl:
                trl=trl.replace(swap[i][0], swap[i][1])
        runFile.write(trl)
        trl = template.readline()
    template.close()
    runFile.close()

    return_code = subprocess.Popen(['sbatch -W slurm_script'],shell=True)
    return_code.wait()
    time.sleep(0.01)

    # d3plot = D3plot(curDir+'\\'+filename+'\\d3plot')

    D3plot_there=True
    time_sleep=0
    while D3plot_there:
        try:

dr=D3plot(curDir+'/'+filename+'/d3plot',state_array_filter=['node_displacement'])
            D3plot_there=False
        except:
            time_sleep=time_sleep+0.1
            print('TIME SLEEP   ' +time_sleep)
            time.sleep(0.1)
    a = dr.arrays['node_displacement']
    b = a - a[0]
    #disp = b[:, 1, 0]-b[lt, 1, 0]
    disp = b[:, 1, 0]

    index_t=[int(i) for i in target[:,0]]

    data = [dr.n_timesteps, disp[index_t]]

    data = np.array(data)

    if lt<500:
        f = f + (np.sum((data[1]-np.transpose(target)[1])**2)**0.5)*2
    else:
        f = f + np.sum((data[1]-np.transpose(target)[1])**2)**0.5
    # f = np.sum((data[1]-target[1])**2)**0.5

    # fig1,ax1=plt.subplots()
    # ax1.plot(ana_time,disp, linestyle='-',label='analysis')
    # ax1.plot(target[:,0],target[:,1], linestyle='--',label='target')
    # # ax1.set_ylabel('Base Shear (kN)')
    # # ax1.set_xlabel('Drift Ratio(%)')
    # # ax1.set_xlim([0,3 ])
    # # ax1.set_ylim([0,400 ])
    # ax1.legend()


    # # f=1
    return (f,X,data)
```

```
import random
import sys
from math import sqrt, pi
import subprocess
import time
import os
import pickle as pickle
from generate_runs import run_multiphys
import numpy as np

import task_inputs
xMax = task_inputs.xMax
xMin = task_inputs.xMin
numVariables = task_inputs.numVariables
lt_1 = task_inputs.lt_1
lt_2 = task_inputs.lt_2
rand = task_inputs.rand

fileName = sys.argv[1]

xInit = [0.0]*numVariables

par=int(fileName[3:6])

if rand=='y':

    for i in range(0,numVariables):
        xInit[i] = xMin[i] + random.uniform(0.0, 1.0)*(xMax[i]-xMin[i])
else:
    with open('Run000_creep_rawC_ss.txt', 'r') as f:
        par_line = f.readlines()[-400+par-1]

    ll=par_line.split(',')
    for i in range(0,numVariables):
        xInit[i]=float(ll[i+5])

X = xInit

f=0
data_all=[]
target=np.loadtxt('targeted_plot_ACI_1.txt')
#target[:,1]=target[:,1]-target[lt_1,1]
f, X,data = run_multiphys (X,fileName,target,lt_1,f)

data_all.append(data)
os.chdir("..")
```

```
target=np.loadtxt('targeted_plot_ACI_2.txt')
#target[:,1]=target[:,1]-target[lt_2,1]
f, X,data = run_multiphys (X,fileName,target,lt_2,f)

data_all.append(data)
#print(data)

m=1

with open(fileName+'.pkl', 'wb') as output:
    # pickle.dump(a, handle, protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(f,output,protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(X,output,protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(m,output,protocol=pickle.HIGHEST_PROTOCOL)

    pickle.dump(data_all,output,protocol=pickle.HIGHEST_PROTOCOL)
```

```
import random
import sys
from math import sqrt, pi
import subprocess
import time
import os
import pickle as pickle
from generate_runs import run_multiphys
import numpy as np

import task_inputs
xMax = task_inputs.xMax
xMin = task_inputs.xMin
numVariables = task_inputs.numVariables
lt_1 = task_inputs.lt_1
lt_2 = task_inputs.lt_2
rand = task_inputs.rand

fileName = sys.argv[1]

xInit = [0.0]*numVariables

par=int(fileName[3:6])

if rand=='y':

    for i in range(0,numVariables):
        xInit[i] = xMin[i] + random.uniform(0.0, 1.0)*(xMax[i]-xMin[i])
else:
    with open('Run000_creep_rawC_ss.txt', 'r') as f:
        par_line = f.readlines()[-400+par-1]

    ll=par_line.split(',')
    for i in range(0,numVariables):
        xInit[i]=float(ll[i+5])

X = xInit
```

```
f=0
data_all=[]
target=np.loadtxt('targeted_plot_ACI_1.txt')
#target[:,1]=target[:,1]-target[lt_1,1]
f, X,data = run_multiphys (X,fileName,target,lt_1,f)

data_all.append(data)
os.chdir("..")
target=np.loadtxt('targeted_plot_ACI_2.txt')
#target[:,1]=target[:,1]-target[lt_2,1]
f, X,data = run_multiphys (X,fileName,target,lt_2,f)

data_all.append(data)
#print(data)

m=1

with open(fileName+'.pkl', 'wb') as output:
    # pickle.dump(a, handle, protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(f,output,protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(X,output,protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(m,output,protocol=pickle.HIGHEST_PROTOCOL)

    pickle.dump(data_all,output,protocol=pickle.HIGHEST_PROTOCOL)
```

```
#Task Name:
taskName = 'creep'

#Optimization Option ('maximize' or 'minimize')
optOption ='minimize'

#Number of Variables (Number of particle DOF)
numVariables = 15

#Max Number of Particles
numParticles = 400
#numParticles = 2

#Max Number of Concurrent subProcesses
numProcesses = 100
#numProcesses = 1

#Max Number of iterations
kmax = 200
#kmax = 3

#loading time
lt_1 = 8
lt_2 = 600

#Initialize xMax and xMin
xMax=[1]*numVariables
xMin=[0]*numVariables
```

```
# t1
xMax[0] = 0.1
xMin[0] = 0.1

# t2
xMax[1] = 1
xMin[1] = 1

# t3
xMax[2] = 10
xMin[2] = 10

# t4
xMax[3] = 100
xMin[3] = 100

# t5
xMax[4] = 1000
xMin[4] = 1000

# t6
xMax[5] = 18000
xMin[5] = 10000

# e1
xMax[6] = 1000
xMin[6] = 0.01

# e2
xMax[7] = 1000
xMin[7] = 0.01

# e3
xMax[8] = 1400
xMin[8] = 0.01

# e4
xMax[9] = 1000
xMin[9] = 0.01

# e5
xMax[10] = 1000
xMin[10] = 0.01

# e6
xMax[11] = 1000
xMin[11] = 0.01

# a
xMax[12] = 0.1
xMin[12] = 0.00001

# lam0
xMax[13] = 10
```

```
xMin[13] = 0.0001

# m
xMax[14] = 2
xMin[14] = 0.1

vmax = [0]*numVariables
vmin = [0]*numVariables
for i in range(0,numVariables):
       vmax[i]=(xMax[i]-xMin[i])/5
       vmin[i]=(xMax[i]-xMin[i])/1000
wmax = 1.0
wmin = 0.8
c1 = 0.7
c2 = 0.5
p1 = 0.6
p2 = 0.8
```

```
import random
import sys
from math import sqrt, pi
import subprocess
import time
import os
import pickle
from generate_runs import run_multiphys
import numpy as np

import task_inputs
xMax = task_inputs.xMax
xMin = task_inputs.xMin
numVariables = task_inputs.numVariables
numParticles = task_inputs.numParticles
vmax = task_inputs.vmax
vmin = task_inputs.vmin
wmax = task_inputs.wmax
wmin = task_inputs.wmin
kmax = task_inputs.kmax
c1 = task_inputs.c1
c2 = task_inputs.c2
p1 = task_inputs.p1
p2 = task_inputs.p2
optOption = task_inputs.optOption
lt_1 = task_inputs.lt_1
lt_2 = task_inputs.lt_2

#**********************

with  open(sys.argv[1]+'.pkl', 'rb') as handle:
        DB = pickle.load(handle)


k = int(sys.argv[2])
i = int(sys.argv[3])

w = DB['w']

XgBest = DB['XgBest']

XNext = []
VNext = []
```

```
X = DB[i]['X']
V = DB[i]['V']
XpBest = DB[i]['XpBest']

for j in range(numVariables):

    # p = random.random()
    r1 = random.uniform(0.0, 1.0)
    r2 = random.uniform(0.0, 1.0)

    # if p < p1:
    #     alpha = random.random()
    #     beta = 1.
    #     gamma = 1.
    # elif p > p2:
    #     alpha = 0.
    #     beta = 0.
    #     gamma = 1.
    # else:
    #     alpha = 0.
    #     beta = 1.
    #     gamma = 1.

    #  VNext.append(max(-vmax[j],min(vmax[j],alpha*w*V[j]  +  beta*c1*r1*(XpBest[j]-
X[j]) + gamma*c2*r2*(XgBest[j]-X[j]))))
    Vtemp=w*V[j]+c1*r1*(XpBest[j]-X[j])+c2*r2*(XgBest[j]-X[j])
    if abs(Vtemp)>vmax[j]:
        VNext.append(np.sign(Vtemp)*vmax[j])
    elif abs(Vtemp)<vmin[j]:
        VNext.append(np.sign(Vtemp)*vmin[j])
    else:
        VNext.append(Vtemp)

    Xtemp=X[j] + VNext[j]
    if Xtemp>xMax[j]:
        XNext.append(xMax[j])
    elif Xtemp<xMin[j]:
        XNext.append(xMin[j])
    else:
        XNext.append(Xtemp)
    # XNext.append()

f=0
data_all=[]
target=np.loadtxt('targeted_plot_ACI_1.txt')
#target[:,1]=target[:,1]-target[lt_1,1]

f, X, data = run_multiphys (XNext,sys.argv[1][:-3],target,lt_1,f)
data_all.append(data)
os.chdir("..")
target=np.loadtxt('targeted_plot_ACI_2.txt')
#target[:,1]=target[:,1]-target[lt_2,1]

f, X, data = run_multiphys (XNext,sys.argv[1][:-3],target,lt_2,f)
data_all.append(data)

m=1

with open(sys.argv[1]+'.pkl', 'wb') as output:
    pickle.dump(f,output,protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(XNext,output,protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(VNext,output,protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(m,output,protocol=pickle.HIGHEST_PROTOCOL)
    pickle.dump(data_all,output,protocol=pickle.HIGHEST_PROTOCOL)
```

References

[1] Z. Bažant, *Creep and Hygrothermal Effects in Concrete Structures, 2018*

[2] International Federation for Structural Concrete (fib), *Model Code for Concrete Structures, 2010*

[3] ACI Committee 209, *Guide for Modeling Creep in Hardened Concrete, 2008*

[4] Prestress Concrete Institute, *Bridge Design Manual Example 9*

[5] Mix Design , *https://www.engineeringcivil.com/mix-design-m-50-grade.html*

[6] A Gentle Introduction to Particle Swarm Optimization *https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/*

[7] Pickett, G. (1942), *The effect of change in moisture content on the creep of concrete under a sustained load,* Journal of the American Concrete Institute, 38, 333–355.

[8] Particle Swarm Optimization, Wikipedia, *https://en.wikipedia.org/wiki/Particle_swarm_optimization*