Numerical Simulation of the Response of Sandy Soils Treated with Pre-Fabricated Vertical Drains

Antonios Vytiniotis

Diploma in Civil Engineering (2005)

National Technical University of Athens, Department of Civil Engineering

Submitted to the Department of Civil and Environmental Engineering in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Civil and Environmental Engineering

at the

Massachusetts Institute of Technology

February 2009

Numerical Simulation of the Response of Sandy Soils Treated with Pre-Fabricated Vertical Drains

Antonios Vytiniotis

Submitted to the Department of Civil and Environmental Engineering on January 25, 2009, in Partial Fulfillment of the Requirements for the Degree of Master of Science in Civil and Environmental Engineering

Abstract

This research is part of the ongoing effort of the Seismic Risk Mitigation for Port Systems Grand Challenge. It addresses the problem of numerically simulating the response of sandy soils treated with earthquake drains, for liquefaction risk mitigation. This thesis describes 1)the implementation of finite 1-D elements to simulate the uncoupled mechanical and flow properties of perforated vertical (PV) drains, 2) the investigation of scaling laws for laminar and turbulent flow inside a PV-drain, 3) the validation of the numerical models using a centrifuge experiment (SSK01) performed at UC-Davis (Kamai, et al., 2008).

The mechanical and flow behavior of the drains are assumed to be uncoupled. The mechanical behavior is treated as a truss element, taking into account the axial stiffness and assuming zero bending stiffness. The flow behavior is treated using the phenomenological Darcy-Weisbach equation. The elements are implemented in the Opensees framework. Two implementations are presented, one for laminar drains, and one for fully turbulent drains. Both of these implementations are used to estimate also the effect of drain storage capacity.

It has been illustrated that the flow in the drains in model scale and in prototype scale might be qualitatively different. If the centrifuge model is scaled N times Reynolds number (Re) is N times larger in prototype scale, so under common situations model scale flow can be laminar even if at the prototype scale flow is fully turbulent. A methodology is presented to select properties of model scale drains (where flow is laminar) to represent prototype drains (where flow is turbulent).

Validation has been performed against SSK01 centrifuge test. Results show good agreement with experimental data. Limitations of the constitutive soil model and the selected input parameters are discussed. Model scale results validate the consideration of the storage capacity effect, and thus use of the implemented drain elements. On the other hand the need for turbulent flow (rather than laminar) drains does not affect significantly the results of the specific test used.

Thesis Supervisor:	Prof. Andrew J. Whittle
Title:	Professor of Civil and Environmental Engineering

Contents

1 Introduction

The seismic performance of major port facilities is controlled, in large part, by the response of waterfront structures and their interactions with soil and rock fills. Widespread failures of waterfront structures in recent earthquakes, most notably at the port of Kobe during the 1995 Hyogoken-Nambu, Kobe earthquake (where 181 out of 187 berths were destroyed, Fig. 1.1) have generated substantial international efforts to improve the analysis and design of waterfront structures. A working group of the International Navigation Association recently published a set of seismic design guidelines for port structures (PIANC, 2002). This volume proposes performance based seismic design methods that classify structures according to their acceptable level of damage for an expected magnitude of seismic event. The most critical port structures (S-class) must be repairable under even the largest expected seismic design events. PIANC (2002) recommend non-linear dynamic analyses of soil structure interaction for all S-class structures.

In practice, many failures of waterfront facilities are attributed to the poor performance of soil fills, including large permanent displacements of quay walls and pile-supported wharf structures due to liquefaction of poorly compacted hydraulic backfills (Port of Oakland in the Loma Prieta earthquake, 1989; and Kobe port, 1995). The spatial variability in the composition (e.g., fines content) and compaction state of these materials, makes reliable predictions of site specific ground response very difficult in practice. The modeling of dynamic soil-structure interactions is further complicated by difficulties in accurately representing the constitutive behavior of the soil (especially in the measurement of input parameters). There has been much previous research in the development of: a) robust, dynamic FE analyses that can handle coupled flow and deformation within the soil and realistic soil-structure interactions; b) constitutive formulations (mainly based on plasticity theory; e.g., Elgamal et al., 2003) for modeling the behavior of soil under cyclic loading; c) incorporation of spatial variability in stochastic FE analyses; and d) applications of these methods in predictions for centrifuge models (notably in the NSF supported Velacs project; Arulanandan & Scott, 1993). To date, most dynamic analyses of waterfront structures have focused on reproducing first order field observations from case studies (e.g., Iai et al., 1998).

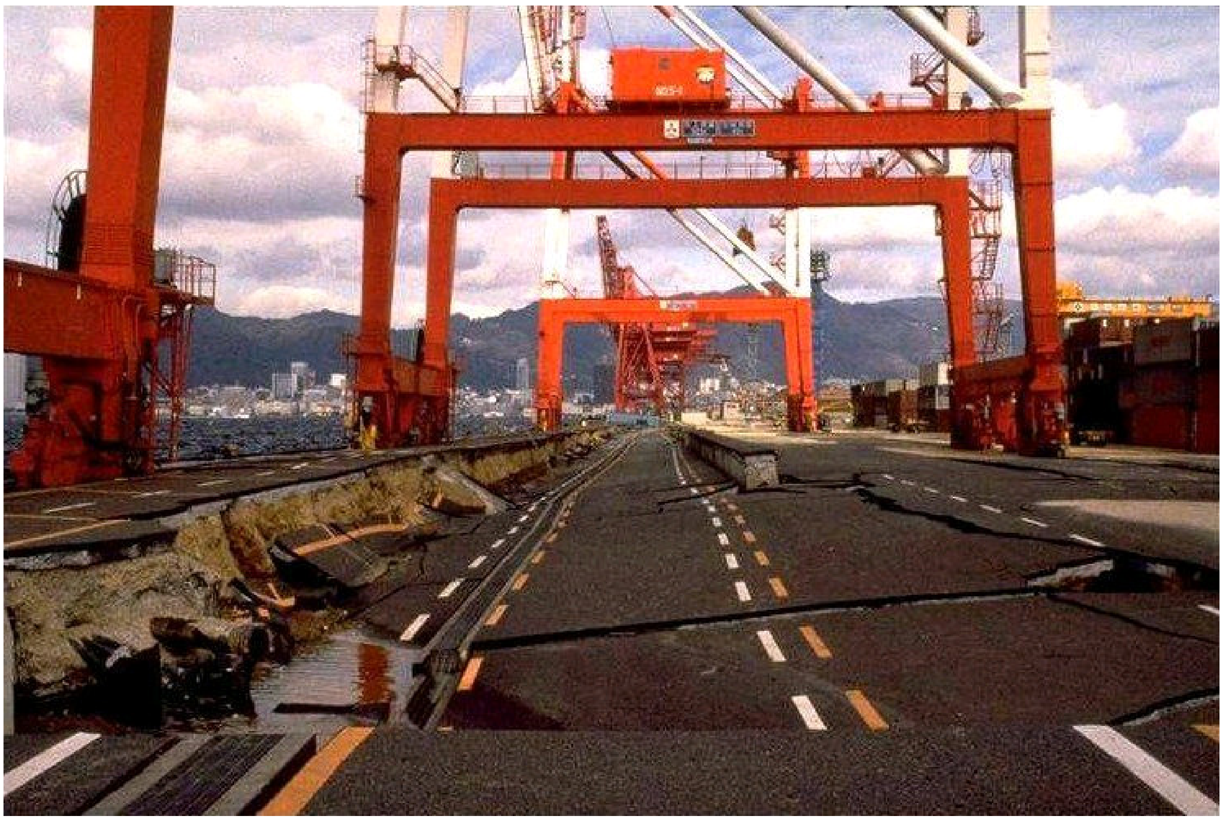
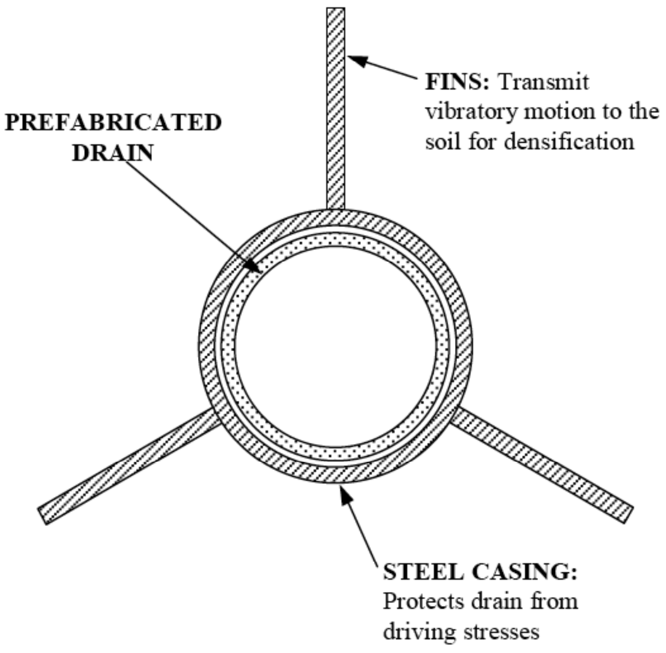


Figure 1-1 Significant damage in the Kobe port facilities due to the 1995 Kobe earthquake

The reduction of seismic risk associated with potential failures of waterfront structures can be accomplished through in-situ remediation schemes that involve strengthening structures and improving the soil fills. There are several established methods for remediating against liquefaction (PHRI, 1997) by either 1) strengthening the soil through compaction, in-situ cementation (e.g., deep-mixed soilcrete columns); or 2) limiting the potential generation of pore pressures within the fill by installing stone or gravel drains. Most of these ground improvement strategies are quite disruptive to existing port operations, and their as-built effectiveness has not been measured in any systematic manner.

Recently, Rathje et al. (2004) have proposed the use of Prefabricated Vertical (PV) drains for mitigating the build-up of excess pore pressures within the soil fill. These drains comprise perforated, corrugated plastic pipes (typically 75mm – 200mm diameter) encased in a geosynthetic fabric (geo-textile), Figure 1.2. There is already extensive experience in the design of geocomposite filtration and drainage materials and specialized equipment for field installation (Fig. 1.3). The installation of PV drains causes limited compaction of the adjacent soil and minimal stiffening of the in-situ soil mass (in contrast to conventional gravel drains) and can be achieved with much less disruption of existing port operations. However, the effectiveness of PV drains as a method of limiting the magnitude of permanent ground deformations caused by strong ground shaking has yet to be proven.

This research considers the effectiveness of PV drains for controlling ground movements and preventing liquefaction under seismic loading conditions. The main focus is the development of numerical methods for modeling both laminar and turbulent flow within PV drains embedded within the soil mass. Special PV drain elements have been developed and implemented within the open source, finite element code Opensees (McKenna & Fenves, 2001). The analyses are compared with results of physical model tests performed at the UC Davis national geotechnical centrifuge facility (Kamai, et al., 2008):



FINS: Transmit vibratory motion to the soil for densification
PREFABRICATED DRAIN
STEEL CASING: Protects drain from driving stresses

Figure 1-2 Cross-section of casing and prefabricated drain (Pestana et al., 1997)

Chapter 2 introduces the finite element formulation used to represent the coupled flow and deformation within the soil mass and validates the u-p approximation used within the OpenSees code. Chapter 3 describes the analysis of PV drains for controlling the development of seismic-induced pore pressures within the soil mass. The chapter describes the formulation and validation of new 1-D finite elements for representing laminar and turbulent flow regimes in PV drains. Chapter 4 summarizes the design and instrumentation of a centrifuge model tests performed at UC Davis, 2006 that is used in subsequent model validation. Chapter 5 describes numerical analyses of the centrifuge model tests and includes parametric studies to evaluate factors affecting the measured ground response. The summary, conclusions and recommendations of this study are in Chapter 6.

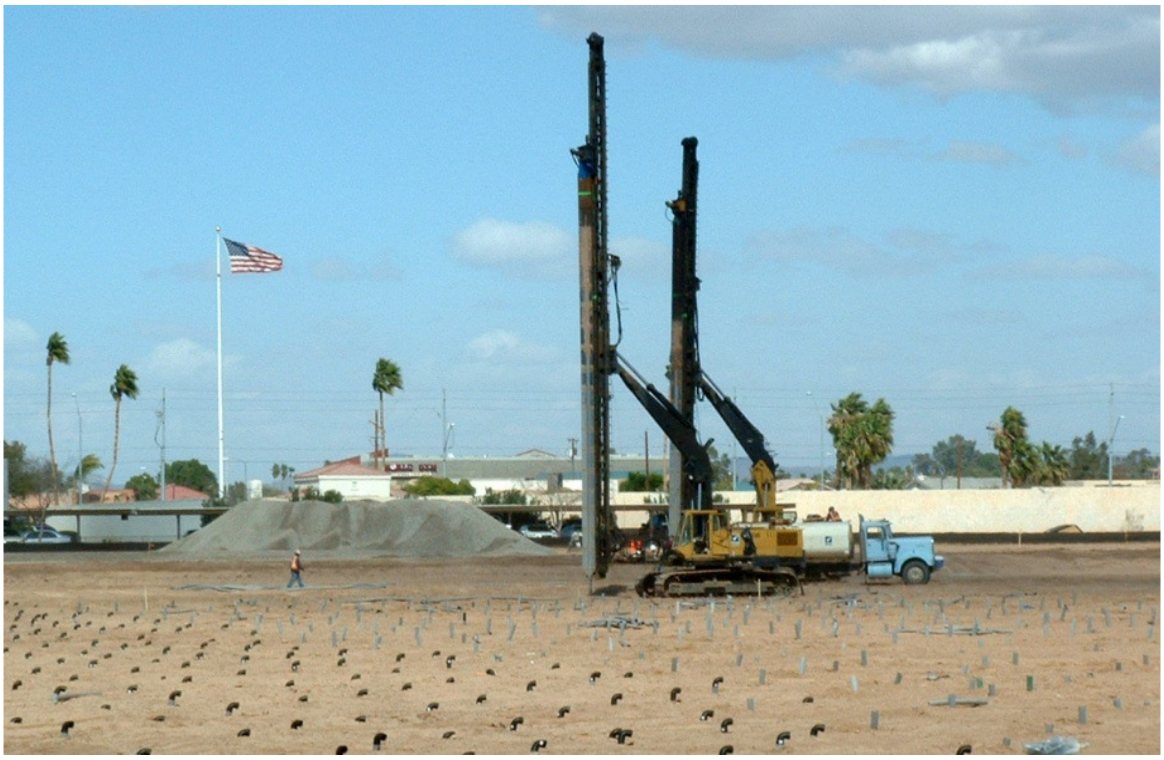


Figure 1-3 Installation of PV drains (Ellington Cross)

2 Finite Element model

2.1 Governing Equations

The governing equations for modeling coupled flow and deformation within a continuous, fully saturated, soil mass were originally formulated by Biot (1956). This section provides a basic summary following the presentation by Zienkiewicz et al. (1999). The three main equations describe the conservation of momentum, diffusion of pore fluid and conservation of fluid mass.

Momentum:
$$S^T\sigma - \rho\ \ddot{u} - \rho_f[\dot{w} + w\nabla^T w] + \rho b = 0 \tag{2.1}$$

where S is the divergence matrix, ∇ is the divergence operator, σ the (total) stress matrix, u the soil displacement vector, w is the average superficial velocity of the percolating water (relative to the soil skeleton), b the vector of body forces, ρ is the total mass density of the soil and ρ_f the mass density of the pore fluid,

Diffusion:
$$-\nabla p - R - \rho_f\ \ddot{u} - \frac{\rho_f[\dot{w} + w\nabla^T w]}{n} + \rho b\ = 0 \tag{2.2}$$

where R are the viscous drag forces, ρ_f is the fluid density, n is the porosity,

Mass:
$$\nabla^T w + \alpha m\ \dot{\varepsilon} + \frac{\rho}{Q} + n \cdot \frac{\dot{\rho_f}}{\rho_f} + \dot{s_0} = 0 \tag{2.3}$$

where ε is the strain. α is the Biot pore pressure coefficient that controls the definition of the effective stresses,σ', within the soil mass:

$$\sigma' = \sigma + \alpha m^T p \tag{2.4}$$

Assuming that the soil and fluid particles are incompressible, α = 1 for most saturated soils in accordance with the conventional Terzaghi definition of effective stress.

The skeletal bulk modulus, Q, is defined from the bulk stiffness parameters of the pore fluid and solid particles (K_f and K_s, respectively):

$$\frac{1}{Q} = \frac{n}{K_f} + \frac{a - n}{K_s} \tag{2.5}$$

Zienkiewicz et al. (1999) have shown that it is possible to simplify this system of equations by neglecting the relative acceleration of the water with respect to the soil skeleton (in eqns. 2.1, 2.2). This leads to an approximate form of the governing equations referred to as the u-p formulation that can be used in consolidation problems, and in coupled dynamic pore pressure displacement analyses.

u-p momentum: $$S^T \sigma - \rho\ \ddot{u} + \rho b = 0 \tag{2.6}$$

u-p mass and diffusion: $$\nabla^T k\left(-\nabla p - \rho_f\ \ddot{u} + \rho_f b\right) + \alpha m\ \dot{\epsilon} + \frac{\dot{p}}{Q} + \dot{s_0}\quad = 0 \tag{2.7}$$

where k is the conventional hydraulic conductivity [L/T] used in seepage analyses.

An alternative simplification of the governing equations is to ignore only the convective terms of the fluid acceleration; this results in the so-called u-p-U approximation:

u-p-U: $$S^T \sigma + \alpha Q(\alpha - n)\nabla(\nabla^T u) + \alpha Q_n \nabla(\nabla^T U) - (1 - n)\rho\ddot{u} - \rho_f n\ddot{U} + \rho b = 0 \tag{2.8}$$

u-p-U: $$(\alpha - n)Q\nabla(\nabla^T u) + nQ\nabla(\nabla^T U) - k^{-1}(nU - nu) - \rho_f U + \rho_f b \quad = 0 \tag{2.9}$$

where U is the water displacement relative to the soil skeleton.

There are several commercially available codes available for the analyses of geotechnical earthquake problems. Table 2-1 compares four programs that have been considered for this research; Flac, ABAQUS, and DYNAFLOW, and OPENSEES. All four programs use either u-p or u-p-U formulations.

This research uses Opensees, “an object-oriented software framework for simulation applications in earthquake engineering using finite element methods”(Mazzoni et al., 2005), for its capabilities, modularity, and open source development.

Table 2-1 Comparison between available software for geotechnical earthquake engineering simulations

Software	Advantages	Disadvantages
ABAQUS *u-p*	Finite Element Implicit integration Good pre- and post processing Open architecture: User elements & User models	No coupled pore pressure-displacement elements for dynamics
DYNAFLOW *u-p-U*	Finite Element Implicit integration Advanced soil models (Prevost) available	Requires separate pre- post- processing capability Closed architecture No user support
FLAC *u-p*	Finite difference Explicit integration Open architecture - FISH functions Advanced soil models (Papadimitriou)	Accuracy & error control? Numerical efficiency
OPENSEES *u-p, u-p-U*	Finite element Implicit integration Open source code Advanced soil models available (Elgamal)	Requires separate pre- post-processor (GID used here) u-p-U formulation is only available for 3D elements

2.2 Verification through a one-dimensional analytical solution

In order to verify Opensees the results of numerical analyses are compared with analytical solutions for a reference problem (Zienkiewicz et al ,1999). Figure 2.1 shows the geometry of a 10m high column of saturated soil subjected to a sinusoidal vertical pressure. The soil exhibits linear, elastic behavior and there is free drainage at the ground surface. The analytical steady state solution for this problem is summarized in Appendix A. Figure 2.2 shows the analytical solutions at four different angular frequencies *ω = 0.1, 1, 10, 100rad/s*. Figures 2.2a and 2.2c show the analytical soil deformations and pore pressures at one selected time (5 secs), while Figures 2.2b and d show the maximum displacements and pore pressures for steady state conditions. The pore pressures conform to the boundary conditions, and at the lowest frequency (0.1 rad/sec) the results converge towards the static (drained) solution. At higher frequencies there are much smaller deformations in the soil, while the zone of maximum pore pressures extends up towards the free surface.

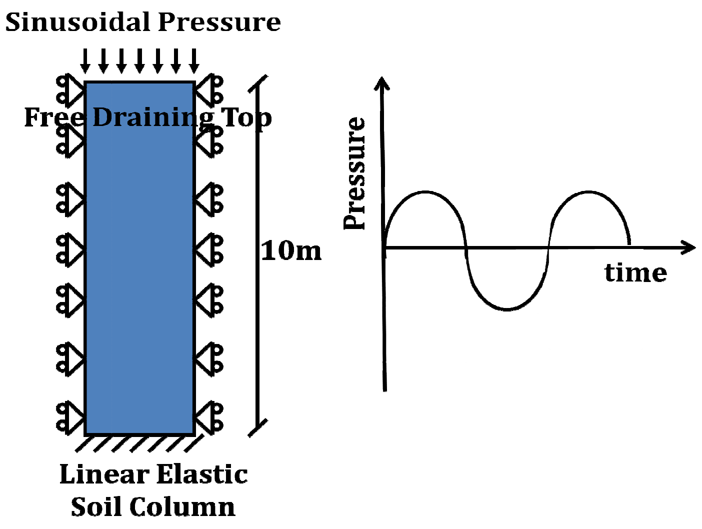


Figure 2-1 Geometry of the verification problem

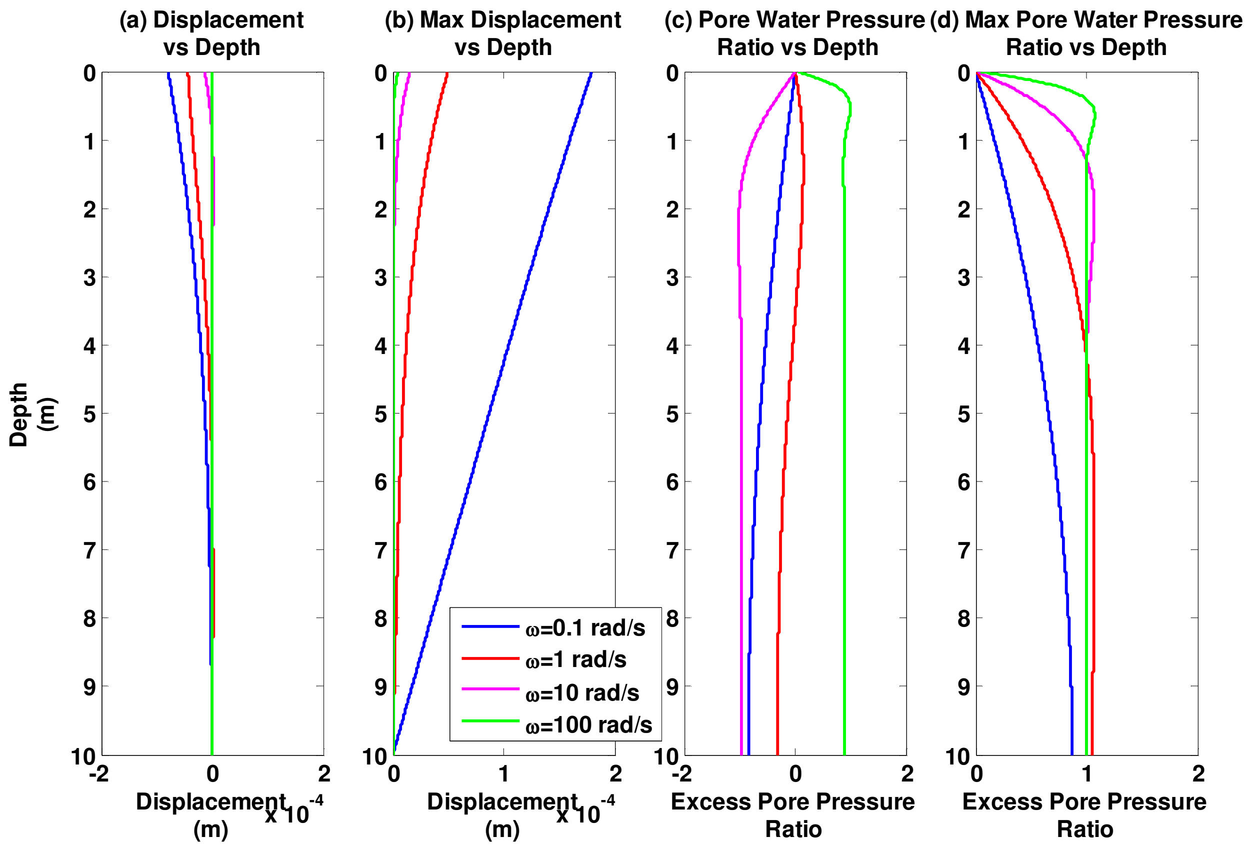
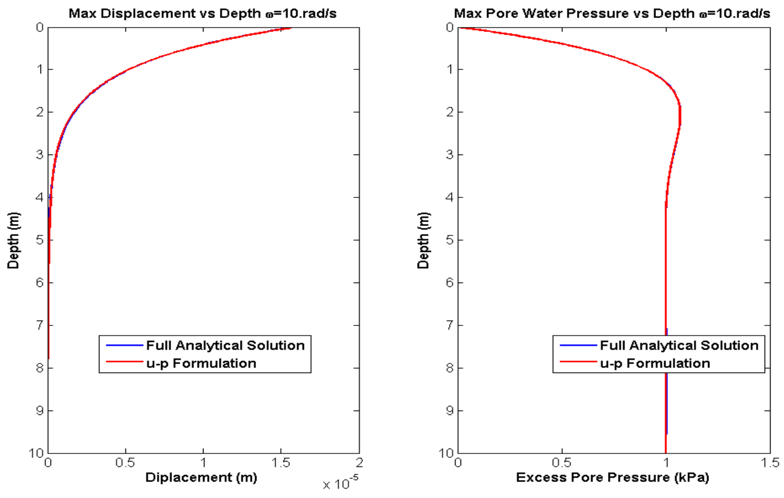


Figure 2-2 Comparison of maximum displacement, displacement at t=5s, maximum excess pore water pressure and pore water pressure for various angular frequencies

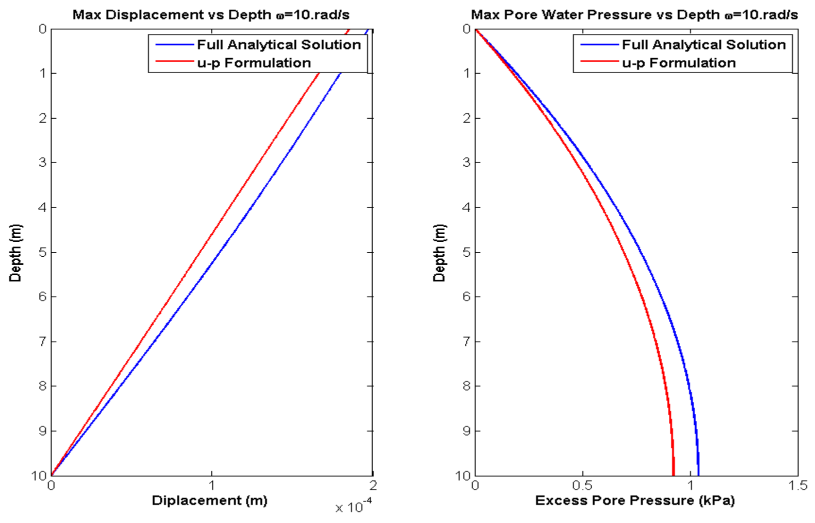
2.2.1 Comparison of the full formulation and the u-p formulation

In this section we compare the analytical solutions for the reference problem using the complete formulation (i.e., accounting for pore fluid acceleration terms and the convective terms, eqns. A.31, A.48; Appendix A), with results obtained using the u-p approximation (Appendix A, eqns. A.49 - A.54). A sample analysis has been performed for a loading frequency of *ω = 10rad/s* (typical for earthquake problems) and hydraulic conductivity values, *k = 0.001, 0.1* and *0.2m/s* (note k = 0.001 m/sec is typical for loose sand, the other values are much higher than expected for real soils). The results from these analyses are summarized in Figure 2-3. For k = 0.001 m/sec (Fig. 2-3a), there is almost perfect agreement between the u-p and fully coupled analytical solutions. However, differences in the deformations and pore pressures become apparent for k = 0.1, 0.2 m/s[1] where fluid velocity is very high. It is important to note that this reference problem involves p-wave propagation, and different behavior should be expected for 1-D shear waves. In a shear wave propagation problem smaller discrepancies between the two formulations should be expected, since the coupling between shear deformation and volumetric response introduces a smaller driving force due to relative movement between the pore fluid and the soil skeleton. Hence, the u-p approximation appears fully justified for typical geotechnical earthquake analyses.

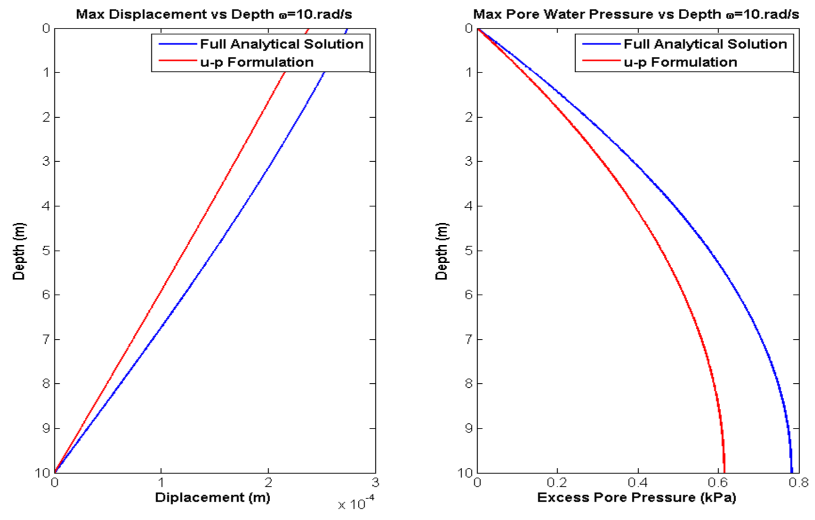


a) k = 0.001 m/sec

[1] Typical range of sand permeability is 10^{-2} to 10^{-5} m/s



b) k = 0.1 m/sec



c) k = 0.2m/sec

Figure 2-3 Effect of u-p approximation for reference problem based on analytical solutions (Appendix A)

2.2.2 Validation of OPENSEES

Figure 2-4 compares numerical simulations of the reference problem using the Opensees code with the analytical solutions described above. The Opensees model uses four-noded 'QuadUP' elements[2] to represent the 10m soil column. The solutions are presented for the 'worst case' condition (i.e., involving the largest expected errors for the u-p formulation) with *ω=10rad/s* and *k=0.2m/s*. The time step selected was 1/20th of the frequency of the applied pulse. The results show excellent agreement between Opensees and the analytical solutions for both deformations and pore pressures confirming the accuracy of the numerical methods used by Opensees.

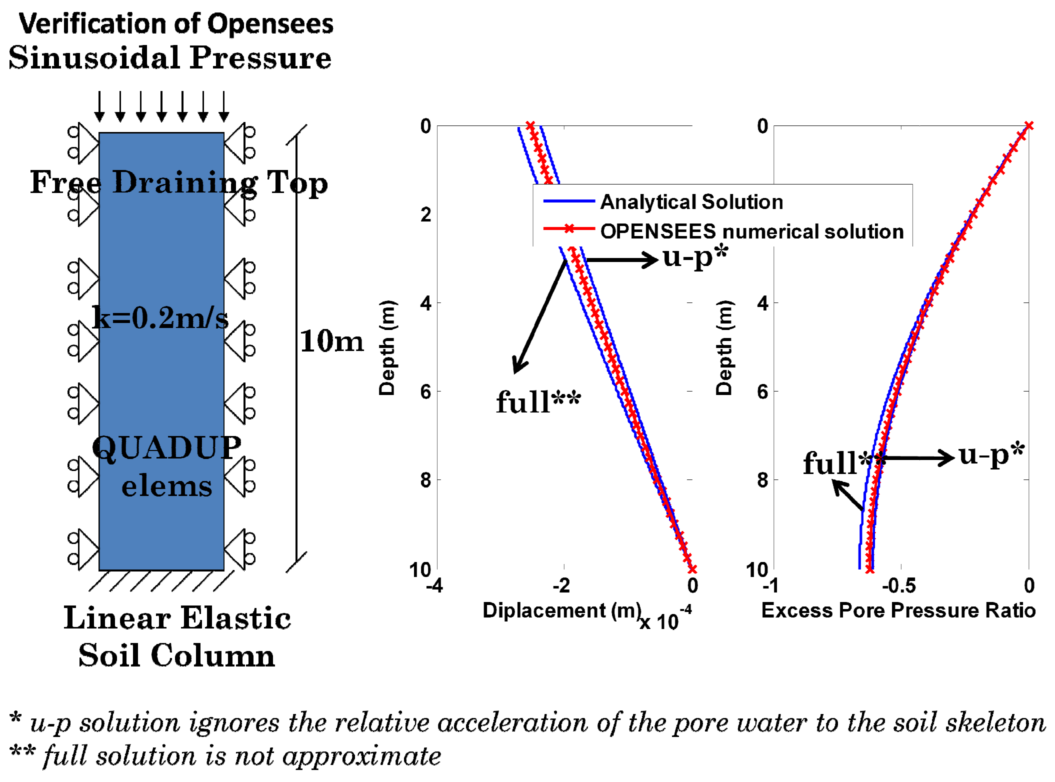


Figure 2-4 Verification of the coupled pore pressure displacement solver in Opensees, snapshot at maximum displacement on top (ω=10rad/s)

[2] These are 4-noded elements that use bilinear isoparametric formulation. Each element node has 2 degrees of freedom (DOFs) for the displacement of the soil skeleton and 1 DOF for pore pressure.

2.3 Constitutive soil model

The accuracy of numerical predictions for the cyclic response of soils during seismic events is controlled, in large part, by the capabilities of the constitutive models that are used to represent the mechanical (i.e., stress-strain-strength) behavior of the pertinent soils. There are two relatively advanced elasto-plastic soil models that are integrated within Opensees and directly available for this research:

1. Pressure Independent, Multi-Yield Surface model (PI-MYS; Mazzoni et al., 2005). The volumetric stress strain reponse is linear-elastic. Plasticity occurs only in the deviatoric stress-strain response and is insensitive to the confining effective stress. The model is primarily applicable for low permeability soils that remain undrained during seismic loading events.

2. Pressure Dependent, Multi-Yield Surface (PD-MYS02) that was developed by Yang et al. (2002, 2003) and is a direct extension of earlier formulations presented by Prevost (1985)[3]. Yang et al. (2002) have shown that the PD-MYS02 model can simulate shear-induced volume contraction or dilation, cyclic mobility and the onset of liquefaction observed in laboratory cyclic shear tests on sands. However, the model requires several input parameters that vary with the initial void ratio and hence, a separate calibration is required for each in situ density condition.

Mazzoni et al. (2005) present typical input parameters from prior calibrations of the PI-MYS and PD-MYS02 models for two reference materials, Nevada fine sand and Yolo loam that are widely

[3] Available in Dynaflow.

used in physical model testing for geotechnical earthquake engineering (following Arulanandan & Scott, 1993) as shown in Tables 2.2 and 2.3.

The capabilities of these models have been evaluated using a simple verification problem comprising a single (quad-up, coupled) plane strain element subject to cyclic shearing along the top surface (the top nodes are constrained to have the same vertical and horizontal displacements) with drainage on top, Figure 2-5. The model has dimensions 1m x 1m, the soil is initially in a K_0-normally consolidated condition with K_0 =0.47 and is assigned a hydraulic conductivity, k = 3x10-5 m/s such that partial drainage can occur. The example problems consider cyclic loading with τ = ±40kPa at a frequency, ω = 1 rad/s for a period of 15secs. This model represents typical 'simple shear' conditions for shearing due to a vertically propagating shear wave within a 1-D soil column (and includes effects of soil inertia).

Figure 2-6 and Figure 2-7 compare results of simulations for a medium-loose Nevada sand at initial vertical effective stress levels, σ'_{v0} = 50kPa and 20kPa, respectively, and a dense Nevada sand at initial vertical effective stress level, σ'_{v0} = 50kPa. The stress path at σ'_{v0} = 50kPa (Figs. 2-7a, 2-8a), shows little accumulation of shear-induced pore pressures through three cycles of loading with cyclic strains in the range 1-2%. In contrast, at the lower confining pressure, σ'_{v0} = 20kPa the model simulates 'cyclic mobility' with large shear-induced pore pressures in each shearing branch (Figs. 2-7b, 2-8b) corresponding to conditions where effective stress paths cross the phase transformation line and much larger shear strains (~10%). Despite the differences in cyclic stress-strain response, the model predicts $\sigma'h \approx \sigma'v$ after just one load cycle.

Figure 2-6cFigure 2-7c show a similar set of results for simple shearing of dense Nevada sand (Dr = 80%, Table 2.3). In this case, the soil exhibits an elastic shakedown with no further accumulation of pore pressures or shear strains after two cycles of loading. Strains are much smaller than those computed for the medium loose sand, and again $\sigma'_h \approx \sigma'_v$ after one load cycle.

Figure 2-8 Figure 2-9 show the results for simple shearing of soft clay (Table 2-2 Input parameters for Yolo Loam using the PI-MYS model). The soil exhibits almost perfectly elastic hysteresis cycles and again $\sigma'_h \approx \sigma'_v$ after one load cycle.

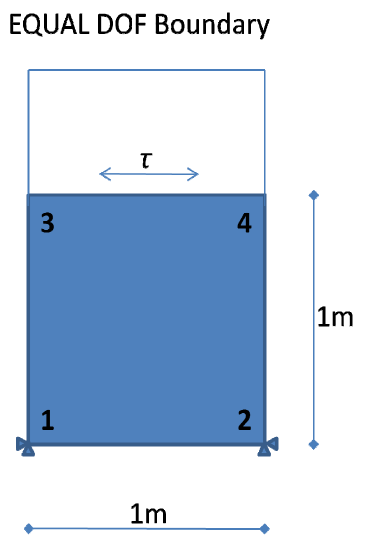


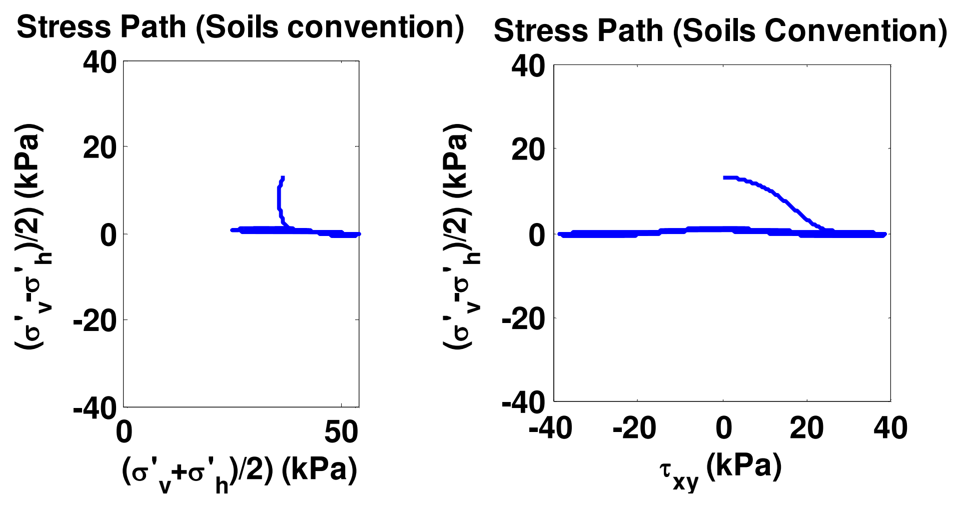
Figure 2-5 Geometry of the reference cyclic shear test

Table 2-2 Input parameters for Yolo Loam using the PI-MYS model

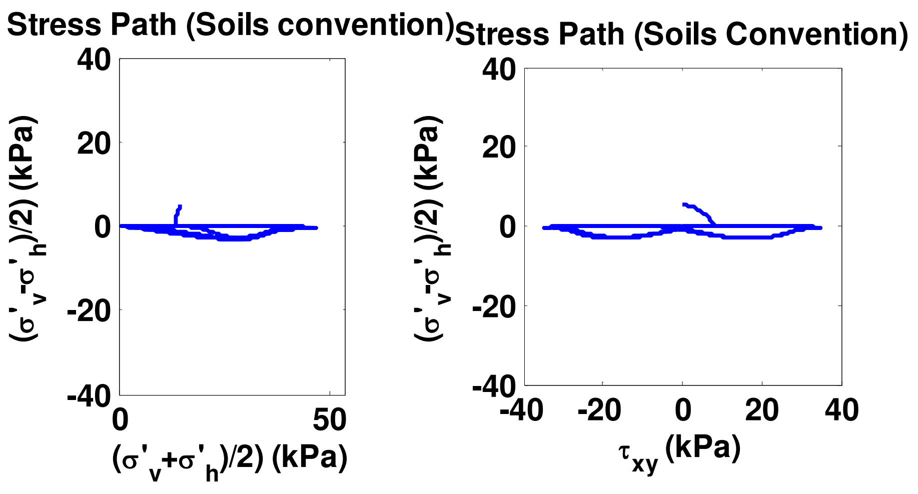
Parameter	Physical Meaning	Yolo Loam
ρ *(ton/m^3)*	Density	1.3
G_{ref} (kPa)	Elastic shear modulus	13000
K_{ref} (kPa)	Elastic bulk modulus	65000
c (kPa)	Cohesion	18.0
γ_{peak}	Peak Shear Strain	0.1

Table 2-3 Input parameters for Nevada fine sand using the PD-MYS02 model

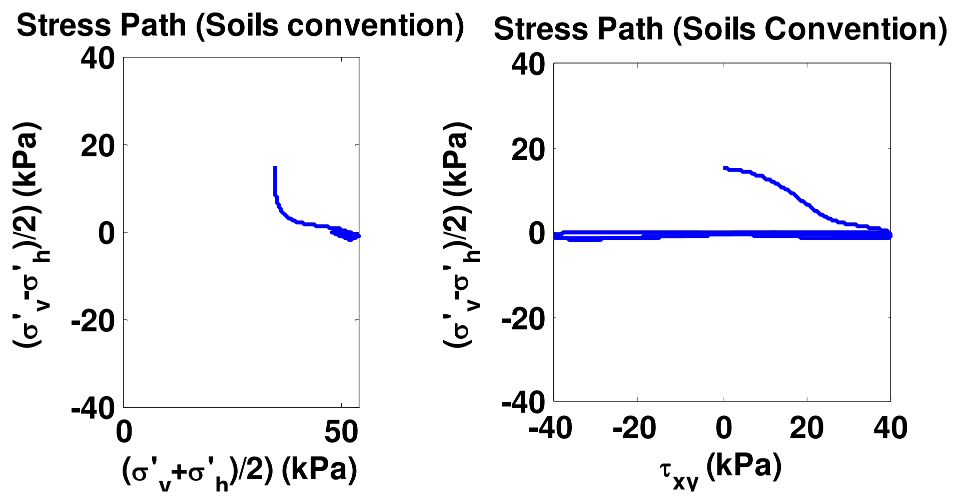
Parameter	Physical Meaning	Dense: D_r = 80%	Medium-Loose: D_r = 40%
ρ*(ton/m^3)*	Density	2.07	1.98
G_{ref} (kPa)	Elastic shear modulus	130000	90000
K_{ref} (kPa)	Elastic bulk modulus	260000	220000
ϕ	Friction angle	36.5	32.0
γ_{peak}	Peak Shear Strain	0.1	0.1
p_{ref} (kPa)	Reference Pressure	80	80
ψ_{PT}	Phase transformation angle	26.0	26.0
c_1	Contraction coefficient	0.013	0.067
c_3	Contraction coefficient	0.0	0.23
d_1	Dilation coefficient	0.3	0.06
d_3	Dilation coefficient	0.0	0.27



a) Medium-Loose Sand, $\sigma'_{v,initial}$=50kPa

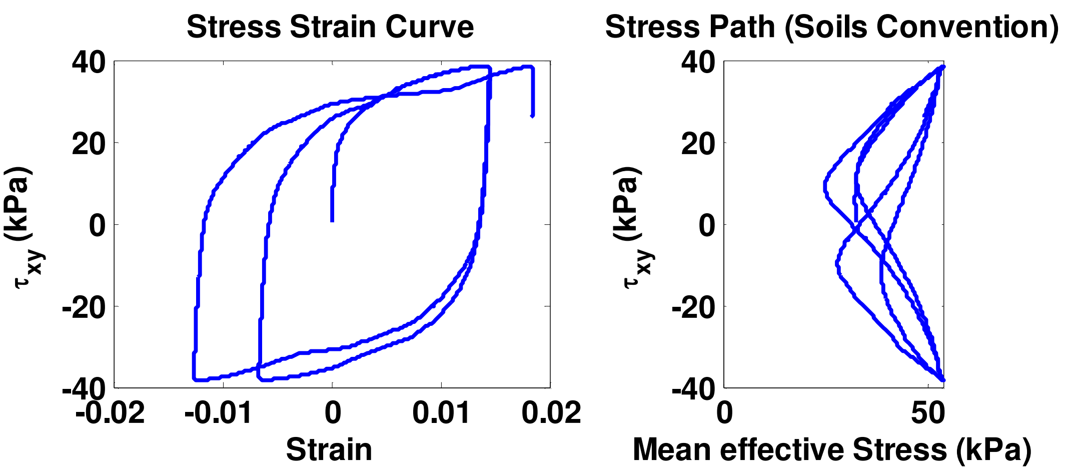


b) Medium-Loose Sand, $\sigma'_{v,initial}$=20kPa

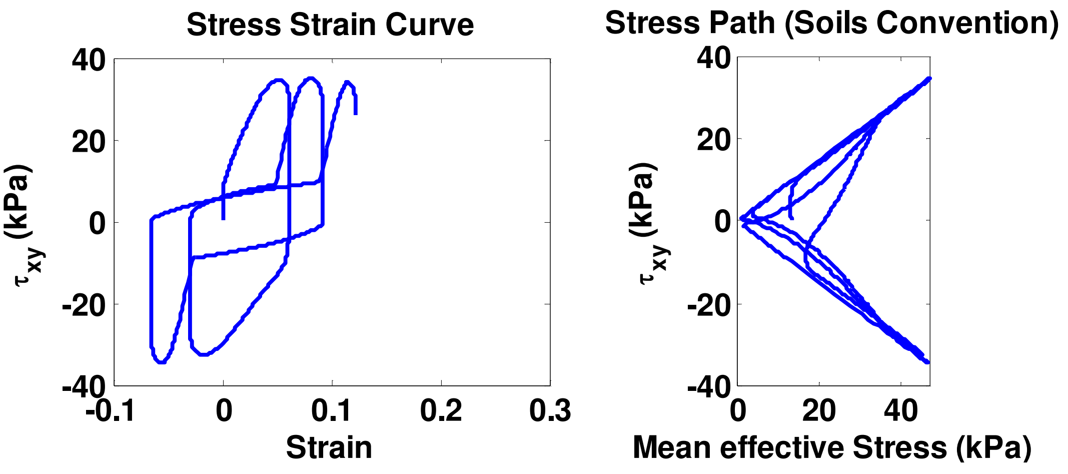


c) Dense Sand, $\sigma'_{v,initial}$=50kPa

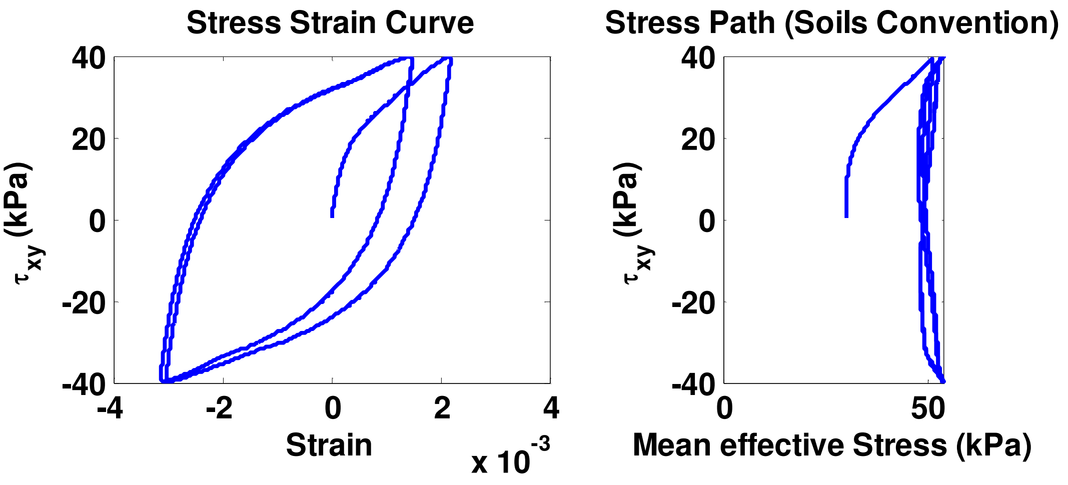
Figure 2-6 Stress paths for reference cyclic shear test using the PD-MYS02 soil model with parameters for Nevada sand



a) Medium-Loose Sand, $\sigma'_{v,initial}$=50kPa



b) Medium-Loose Sand, $\sigma'_{v,initial}$=20kPa



c) Dense Sand, $\sigma'_{v,initial}$=50kPa

Figure 2-7 Stress paths and stress strain for reference cyclic shear test using the PD-MYS02 soil model with parameters for Nevada sand

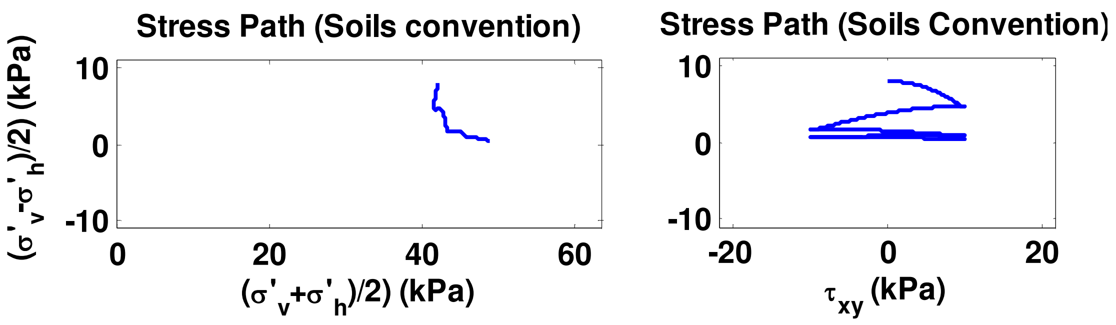


Figure 2-8 Stress paths for reference cyclic shear test of a Soft Clay for $\sigma'_{v,initial}$=50kPa

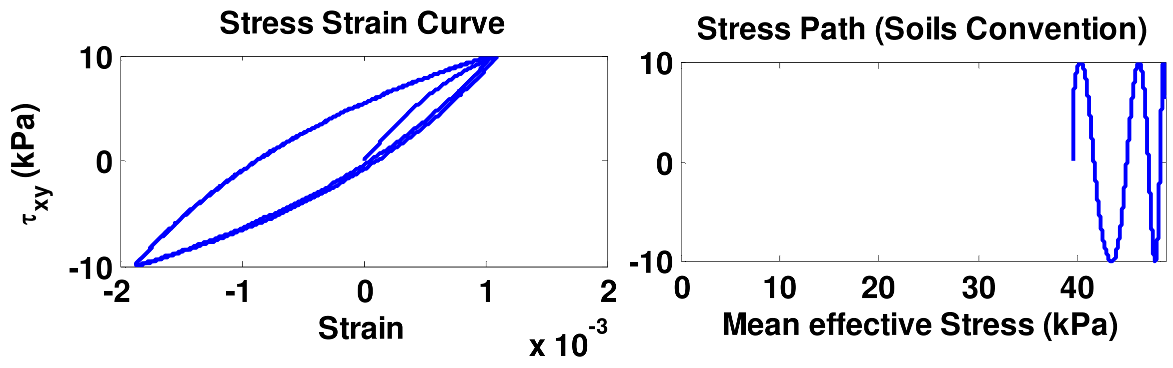


Figure 2-9 Stress path and stress strain curve for a DSS test of a Soft Clay for $\sigma'_{v,initial}$=50kPa

3 Modeling of PV earthquake drains

3.1 Prior Analyses

Earthquake drains are presented schematically in Figure 3-1. During a seismic event the excess pore pressure gradient drives vertical flow inside the soil towards the free surface and radial or horizontal flow towards the vertical drains. This allows for dissipation of the excess pore pressure and mitigation of liquefaction risk.

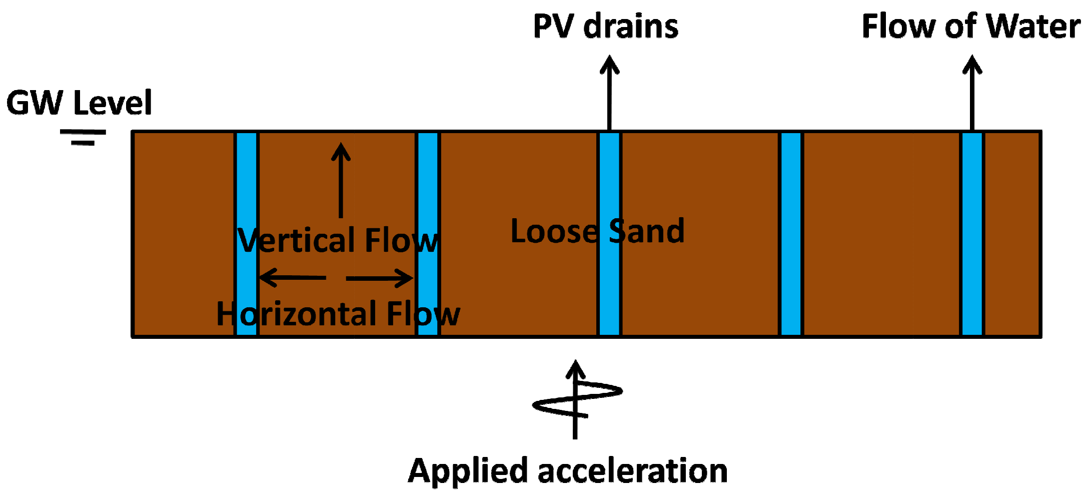


Figure 3-1 Schematic mechanisms of liquefaction mitigation using earthquake drains

Seed & Booker, (1977) were the first to analyze the role of vertical drains for mitigating liquefaction risks. Their analyses assume radial dissipation of excess pore pressures within the soil, while 1-D vertical strains are caused by changes in effective vertical stresses:

$$\frac{\partial}{\partial r}\left(k\frac{1}{r}\frac{\partial u}{\partial r}\right) = m_v\left(\frac{\partial u}{\partial t} - \frac{\partial u_g}{\partial t}\right) \tag{3.1}$$

where k is the hydraulic conductivity, m_v the 1-D compressibility, u the excess pore pressures and u_g the excess pore pressures generated by cyclic loading.

The drain itself acts as a perfect sink (with zero excess pore pressure) providing unlimited vertical transmission of pore fluid, an assumption subsequently defined as a 'perfect drain' condition. The solution of equation 3.1 requires the specification of an empirical model to characterize the generation of excess pore pressures in cyclic loading. This is usually accomplished using empirical data from undrained cyclic shear tests (either triaxial or simple shear tests), where pore pressures are reported as a function of the number of uniform load cycles, N. DeAlba et al. (1975) proposed that the excess pore pressure ratio, r_u (= u_g/σ'_0) can be estimated as follows:

$$r_u = 2\sin^{-1}\left(\frac{N}{N_L}\right)^{\frac{1}{2\theta}} = m_v\left(\frac{\partial u}{\partial t} - \frac{\partial u_g}{\partial t}\right) \qquad (3.2)$$

where N_L is defined as the number of cycles required to initiate liquefaction, and θ is an empirical constant. Figure 3-2 shows that θ = 0.7 provides a good approximation based on results from undrained direct simple shear tests.

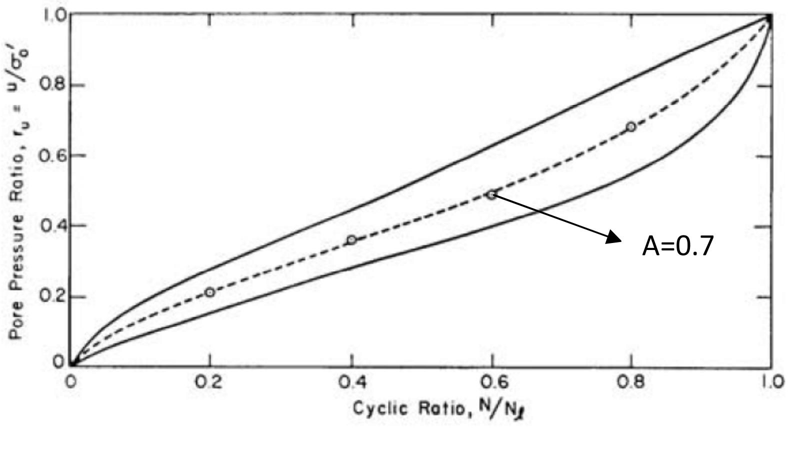


Figure 3-2 Pore pressure generation from undrained direct simple shear tests: data range and (DeAlba et al., 1975)

Using these empirical functions the generation of excess pore pressures can be estimated using the approach suggested by Seed et al. (1975):

$$\frac{\partial u_g}{\partial t} = \frac{\partial u_g}{\partial N}\frac{\partial N}{\partial t} \approx \frac{\partial u_g}{\partial N}\frac{N_{eq}}{t_d} \tag{3.3}$$

where t_d is the total duration of earthquake shaking and N_{eq} is the equivalent number of uniform load cycles for a given design earthquake.

The value of m_v can be determined by means of a cyclic loading triaxial compression test, as described by Lee & Albaisa, 1974. Seed and Booker created design charts (Figure 3-3) which for a given seismic event, and a given spacing ratio (ratio of drain diameter to drain to drain distance) predict the expected excess pore pressure ratio. In order to design a drainage-based earthquake mitigation technique, a target maximum excess pore pressure ratio is selected, and the appropriate drain spacing ratio is evaluated. They, also found that the effect of vertical drainage on the maximum pore pressure ratio within the sand has minimal effect on the maximum pore pressure ratio and hence can be ignored in the analysis.

Experimental work has been performed by Onoue et al (1987) in order to verify Seed and Booker's method. Gravel drains have been constructed in a sandy soil to a depth of 11m in field scale, with various spacing ratios of 0.25, 0.333, and 0.417. The experimental results are printed on top of the Seed and Booker design chart in Figure 3-4, and the read values from the class A prediction are also shown. Onoue et al (1987) concluded that disregarding well resistance must be considered in the analysis, since the values of r_u from the Seed and Booker diagrams were considerably smaller than the measured values.

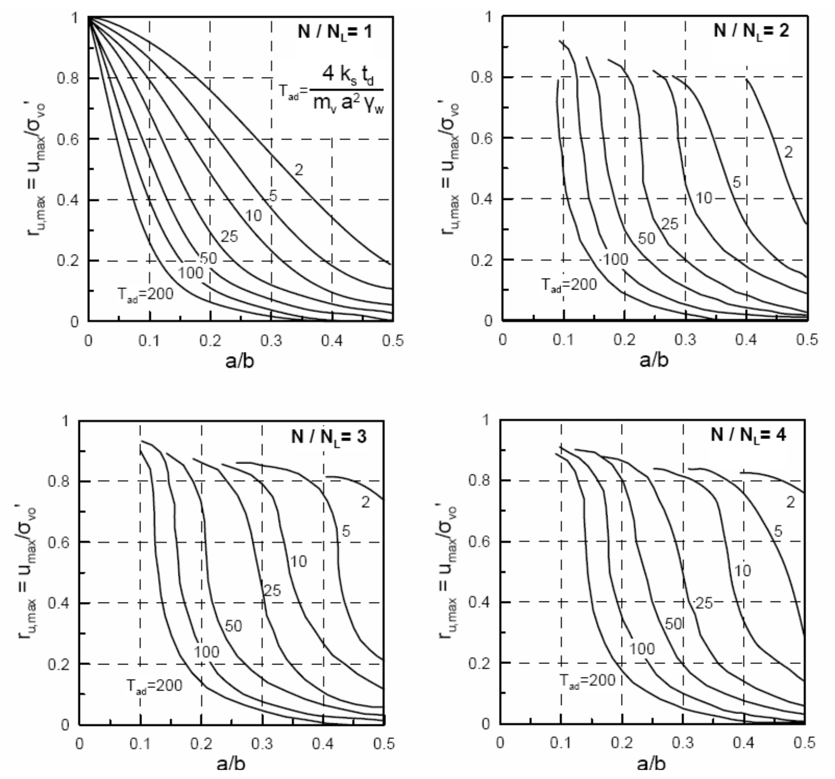


Figure 3-3 Design charts for groups of gravel drains against liquefaction (Seed & Booker, 1977). N is the number of cycles in the design scenario, N_L is the number of cycles to reach liquefaction without the drains, a/b is the spacing ratio, $r_{u,max}$ is the maximum expected pore pressure ratio in the soil stratum, k_s is the soil hydraulic conductivity, t_d is the duration of the design event, m_v is the vertical soil compressibility, a is the drain radius, and γ_w is the water unit weight.

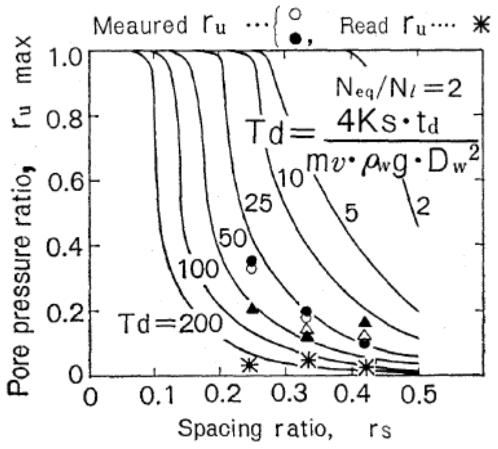


Figure 3-4 Effect of well resistance on excess pore pressures in the soil shown on top of the Seed and Booker design charts. 'Read values' refer to theoretical solutions from Seed & Booker (1977), measured data are provided from field scale experiments Onoue et al (1987)

Onoue (1988) has used the same axi-symmetric diffusion equation, and empirical pore pressure generation model, as Seed and Booker, but he also included the effect of well resistance in design charts. It is found that when the cycle ratio (number of cycles divided by the number of cycles to reach liquefaction) $N/N_L \leq 1$ then there is a significant impact of vertical flow in the sand, while for $N/N_L > 1$ this effect can be disregarded. Figure 3-5 show design charts which take into account the effect vertical flow while Figure 3-6 shows design chars which exclude vertical flow.

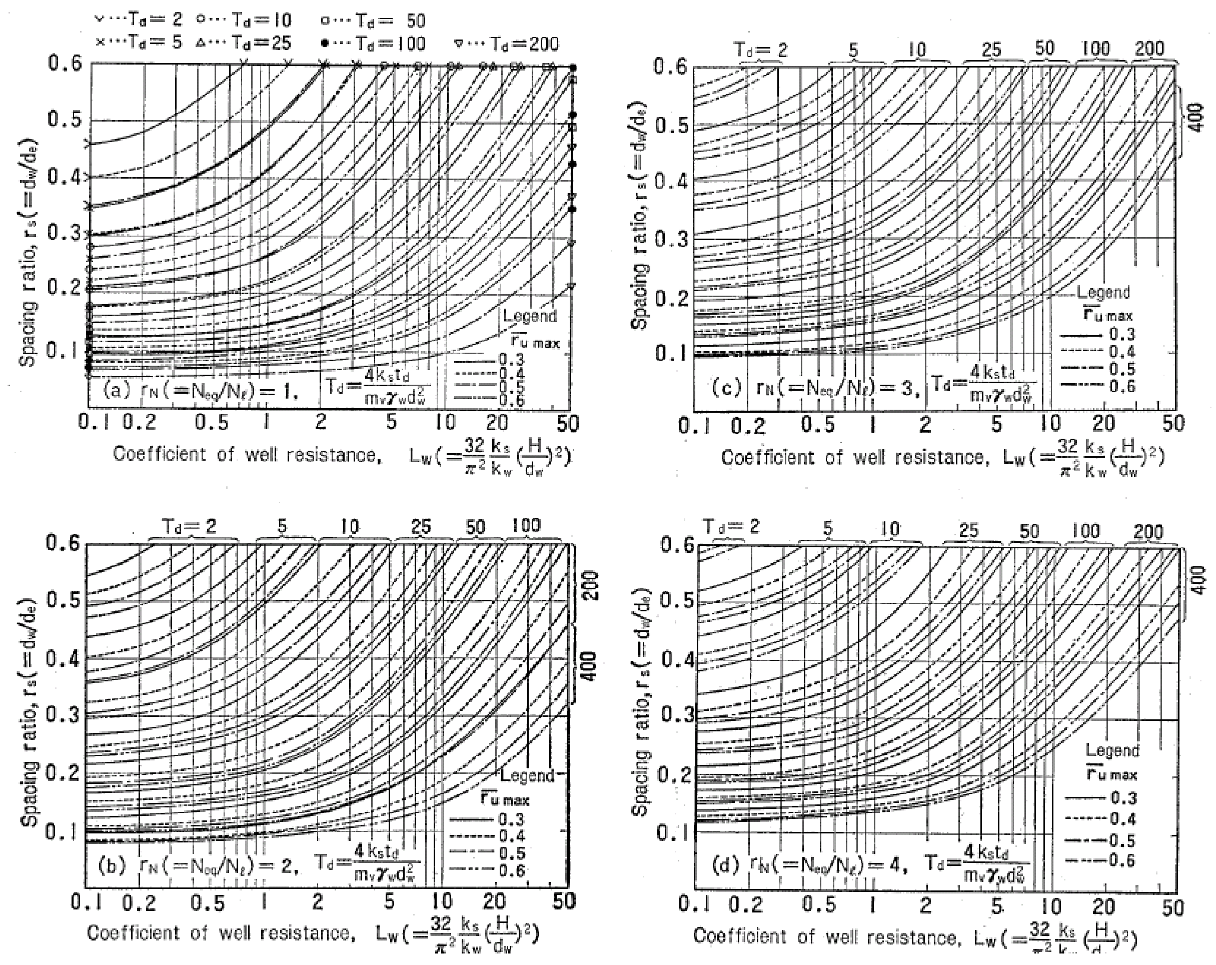


Figure 3-5 Design charts for groups of gravel drains against liquefaction (Onoue, 1988). N_{eq} is the number of cycles in the design scenario, N_l is the number of cycles to reach liquefaction without the drains, r_s is the spacing ratio, $r_{u,max}$ is the maximum expected pore pressure ratio in the soil stratum, k_s is the soil hydraulic conductivity, k_w is the drain material hydraulic conductivity, H is the soil strata height, t_d is the duration of the design event, m_v is the vertical soil compressibility, a is the drain radius, and γ_w is the water unit weight.

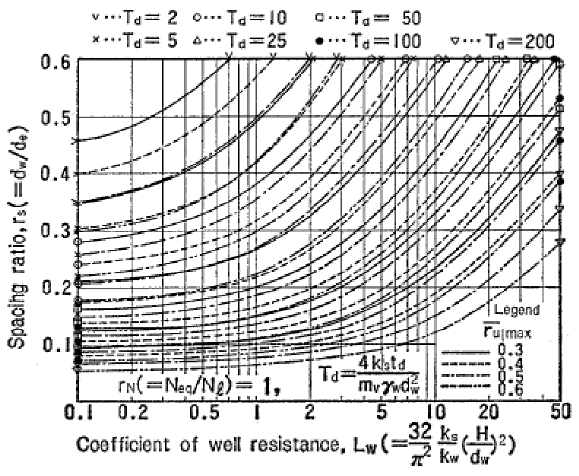


Figure 3-6 Design charts for groups of gravel drains against liquefaction (Onoue, 1988), in the case where vertical direction de-watering is disregarded. N_{eq} is the number of cycles in the design scenario, N_l is the number of cycles to reach liquefaction without the drains, r_s is the spacing ratio, $r_{u,max}$ is the maximum expected pore pressure ratio in the soil stratum, k_s is the soil hydraulic conductivity, k_w is the drain material hydraulic conductivity, H is the soil strata height, t_d is the duration of the design event, m_v is the vertical soil compressibility, a is the drain radius, and γ_w is the water unit weight.

Pestana et al. 1997 have extended the analysis of Onoue to include the effects of storage capacity. In many cases, the phreatic level of water inside a drain is not at the top of the drain. So, as the excess pore pressure develops, the water level will first rise up to the top of the drain (water will be stored inside the drain), before overtopping occurs; Storage capacity is defined as the amount of water that will be stored in the drain before outflow occurs.

Analyses of the effect of storage capacity have been performed, with perfect drains, drains with finite permeability, drains with variable initial water level, and drains with presence of reservoir (storage capacity) of varying size. From these analyses, a combined plot showing the effect of storage capacity drain permeability is presented in Figure 3-7. From this plot, by comparing the k_d/k_s for infinity and 1000, we can see that the perfect drain assumption is not valid in many circumstances. Also, not introducing the effect of storage capacity can lead to unconservative design.

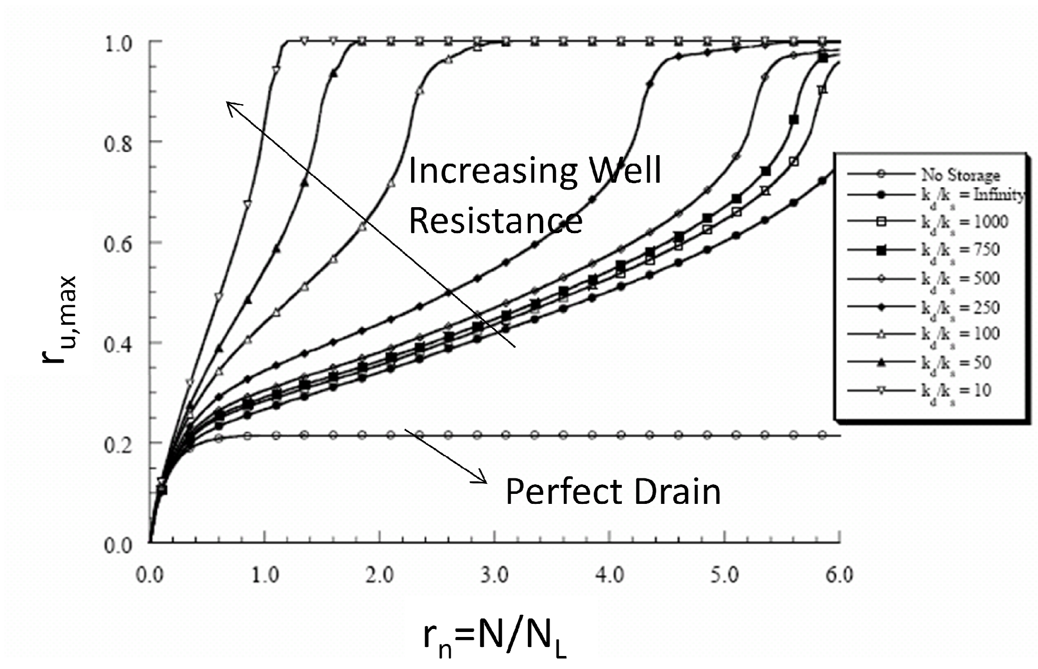


Figure 3-7 Pore pressure ratio with storage and varying drain resistance, where k_s is the permeability of the soil, k_d is the permeability of the drain, r_N is the cycle ratio N/N_L, and $R_{u,max}$ is the maximum excess pore pressure ratio. The drain spacing is s/d=5 and the water level inside the drain is 1m below ground surface. (Pestana et al 1997)

3.2 Prior numerical two-dimensional numerical analyses

Apart from one-dimensional work, a two-dimensional finite difference code, FLAC, and an advanced elasto-plastic bounding surface plasticity soil model (Andrianopoulos, 2006), have been utilized for the simulation of soil improved with infinite permeability stone columns (Papadimitriou, et al, 2007). They examined the feasibility of performing coupled pore pressure displacement analysis in a vertical shear wave propagation problem. Their results are not directly compared to Seed and Booker but a short summary is presented in Figure 3-8 in the form of excess pore pressure ratio vs time plots for four different analyses. Their analyses correctly predict a decreasing $r_{u,max}$ with increasing spacing ratio.

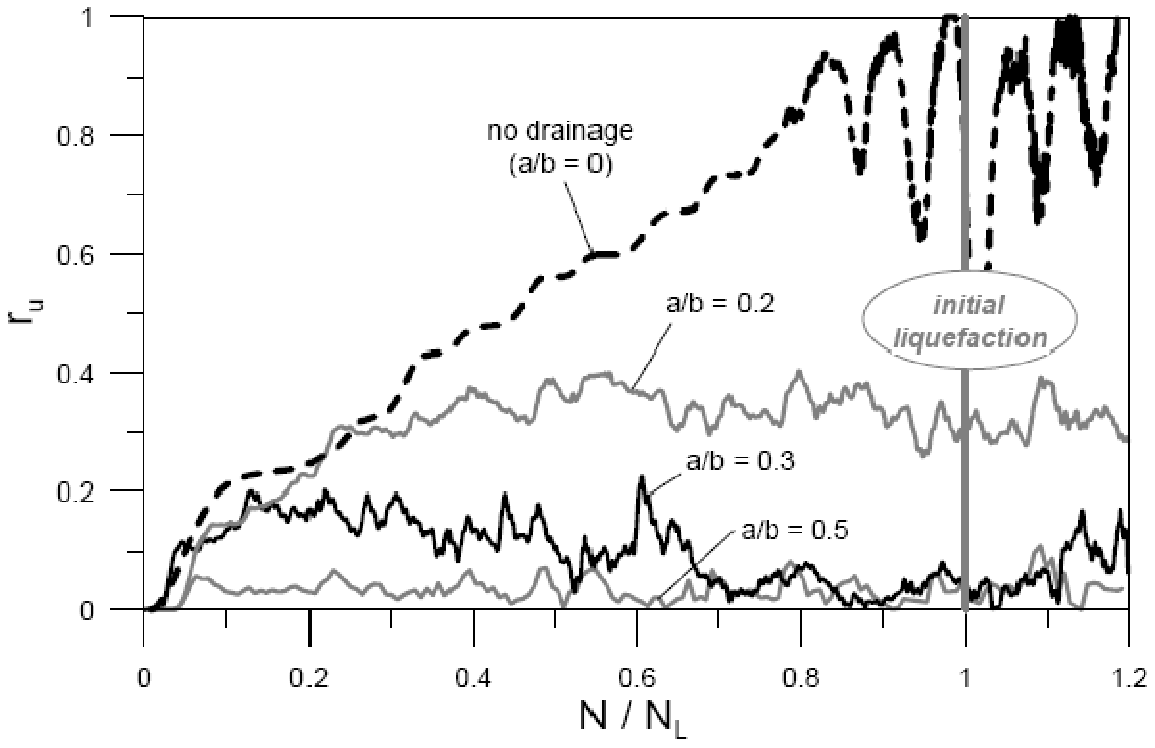


Figure 3-8 Rate of excess pore pressure buildup from analyses for various improvement ratios a/b (drain radius/drain spacing) and comparison with the analysis for fully undrained conditions (a/b=0)

3.3 Hydraulics of vertical drains

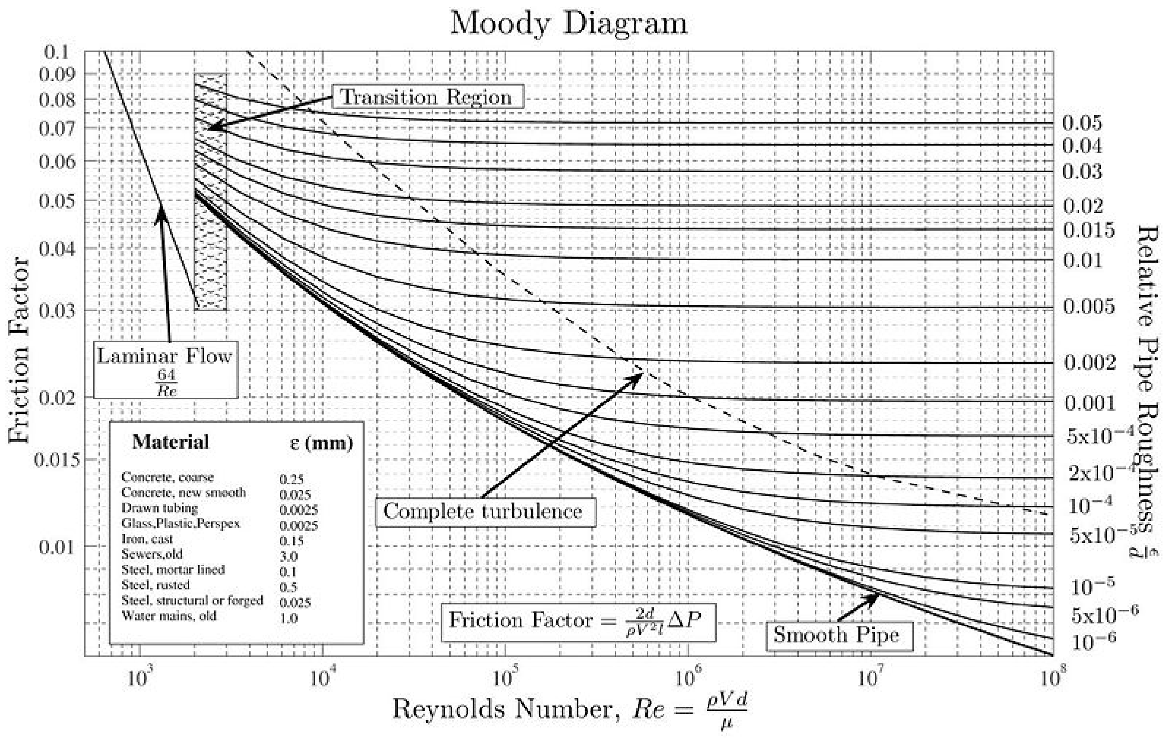
The prior analyses have considered vertical drains as either ideal conduits with unlimited fluid transmissivity, or have represented well resistance by assuming continued validity of Darcy's law with an equivalent hydraulic conductivity. This section considers the hydraulics of flow within these vertical pipes.

The Darcy–Weisbach equation is a widely used phenomenological equation used in hydraulics. It relates pressure loss due to friction to the average velocity of the fluid flow:

$$\Delta P = \lambda \cdot \frac{L}{D} \cdot \frac{\rho V^2}{2} \qquad (3.4)$$

Where λ is a dimensionless coefficient of laminar or turbulent flow, L is the length of the pipe, D is the diameter of the pipe, ρ is the density of the water, V is the average velocity of the flow. λ is equivalent to the Darcy friction factor (f) and can be estimated for both laminar and turbulent flow from the Moody diagram (Figure 3-9).

In a typical drain the roughness ε can be as low as 0.0025mm, but could increase substantially due the holes on the side of the drain and the accumulation of debris inside a PV-drain.



Material	ε (mm)
Concrete, coarse	0.25
Concrete, new smooth	0.025
Drawn tubing	0.0025
Glass,Plastic,Perspex	0.0025
Iron, cast	0.15
Sewers,old	3.0
Steel, mortar lined	0.1
Steel, rusted	0.5
Steel, structural or forged	0.025
Water mains, old	1.0

Figure 3-9 Estimation of the Darcy friction factor for laminar and turbulent flow

3.4 Finite Element Implementation of 1-D Drain Elements

3.4.1 Truss theory

A truss finite element is being used to predict the mechanical part of the drain element. Simple truss theory is very well established in a finite element context. The definitions for this element are presented in Figure 3-10. We define the force vector in global coordinates:

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{3.5}$$

and the displacements vector:

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{3.6}$$

The stiffness matrix for this truss element is:

$$k = \begin{bmatrix} \cos^2(\theta) & \cos(\theta)\cdot\sin(\theta) & -\cos^2(\theta) & -\cos(\theta)\cdot\sin(\theta) \\ \cos(\theta)\cdot\sin(\theta) & \sin^2(\theta) & -\cos(\theta)\cdot\sin(\theta) & -\sin^2(\theta) \\ -\cos^2(\theta) & -\cos(\theta)\cdot\sin(\theta) & \cos^2(\theta) & \cos(\theta)\cdot\sin(\theta) \\ -\cos(\theta)\cdot\sin(\theta) & -\sin^2(\theta) & \cos(\theta)\cdot\sin(\theta) & \sin^2(\theta) \end{bmatrix} \cdot \frac{AE}{L} \tag{3.7}$$

With the above definitions the equilibrium is defined as:

$$q = k \cdot u \tag{3.8a}$$

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \frac{AE}{L} \cdot \begin{bmatrix} \cos^2(\theta) & \cos(\theta)\cdot\sin(\theta) & -\cos^2(\theta) & -\cos(\theta)\cdot\sin(\theta) \\ \cos(\theta)\cdot\sin(\theta) & \sin^2(\theta) & -\cos(\theta)\cdot\sin(\theta) & -\sin^2(\theta) \\ -\cos^2(\theta) & -\cos(\theta)\cdot\sin(\theta) & \cos^2(\theta) & \cos(\theta)\cdot\sin(\theta) \\ -\cos(\theta)\cdot\sin(\theta) & -\sin^2(\theta) & \cos(\theta)\cdot\sin(\theta) & \sin^2(\theta) \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{3.8b}$$

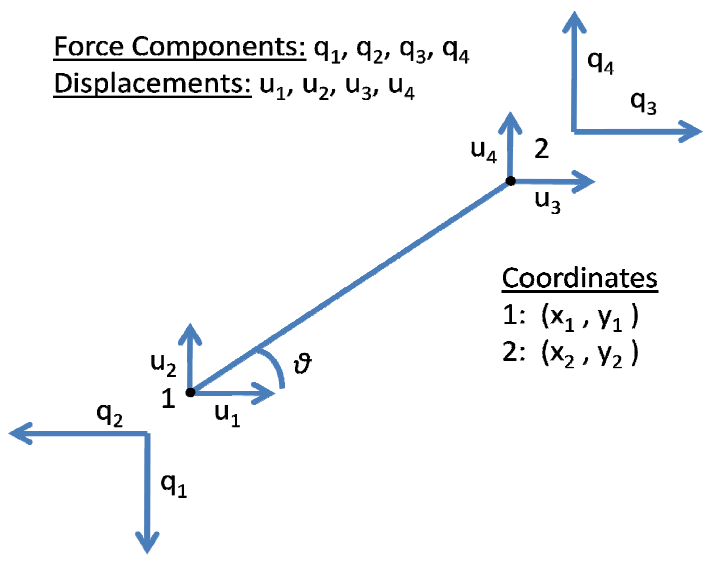

Force Components: q1, q2, q3, q4
Displacements: u1, u2, u3, u4
q4
q3
u4
2
u3
Coordinates
1: (x1 , y1)
2: (x2 , y2)
u2
ϑ
1
u1
q2
q1

Figure 3-10 Coordinates, displacements, and forces definitions for truss element

3.4.2 Laminar Flow

3.4.2.1 Flow equations

Laminar flow occurs when a fluid flows smoothly or in regular paths. In laminar flow, sometimes called streamline flow, the velocity, pressure, and other flow properties at each point in the fluid remain constant. Laminar flow over a horizontal surface may be thought of as consisting of thin layers, or laminae, all parallel to each other. Inside a pipe, the fluid in contact with the pipe is stationary, but all the other layers slide over each other.

In laminar flow, it is possible to simplify the Darcy-Weisbach equation by a a linear relation between the pressure loss and the average velocity of water (or flow). For laminar flow:

$$\lambda = \frac{64}{Re} \tag{3.9}$$

where Re is the Reynolds number.

For a circular tube filled with water:

$$\mathrm{Re} = \frac{\mathrm{VD}}{\nu} \tag{3.10}$$

where ν is the kinematic viscosity of water.

Using the above equations:

$$\Delta P = \frac{64\,\nu}{VD} \cdot \frac{L}{D} \cdot \frac{\rho V^2}{2} \Rightarrow \tag{3.11a}$$

$$\Delta P = 64 \cdot \nu \cdot \frac{L}{D^2} \cdot \frac{\rho V}{2} \Rightarrow \tag{3.11b}$$

$$\Delta P = \frac{32 \cdot \nu \cdot L \cdot \rho}{A \cdot D^2} Q \Rightarrow \tag{3.11c}$$

$$\Delta P = \frac{32 \cdot \mu \cdot L}{A \cdot D^2} Q \Rightarrow \tag{3.11d}$$

$$Q = \frac{A \cdot D^2}{32 \cdot \mu \cdot L} \Delta P \Rightarrow \tag{3.12a}$$

$$Q = C_l i \tag{3.12b}$$

where μ is the dynamic viscosity of water (or any fluid in general), i = Dp/L is the pressure gradient. The coefficient C_l [$L^6F^{-1}T^{-1}$] is defined by:

$$C_l = \frac{A \cdot D^2}{32 \cdot \mu} \tag{3.13}$$

3.4.2.2 Finite Element approximation

Using this formulation we can define a one-dimensional element in a two dimensional space relating water pressure to flow. We first define the vector of flow (equivalent to the vector of external forces in the truss element):

$$Q = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \tag{3.14}$$

And the vector of pressures:

$$p = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \tag{3.15}$$

By using equation 3.13 we can define a transmissivity matrix:

$$T = \begin{bmatrix} -\dfrac{A \cdot D^2}{32 \cdot \mu \cdot L} & \dfrac{A \cdot D^2}{32 \cdot \mu \cdot L} \\ \dfrac{A \cdot D^2}{32 \cdot \mu \cdot L} & -\dfrac{A \cdot D^2}{32 \cdot \mu \cdot L} \end{bmatrix} \tag{3.16}$$

Then the equilibrium equation in this element is defined as:

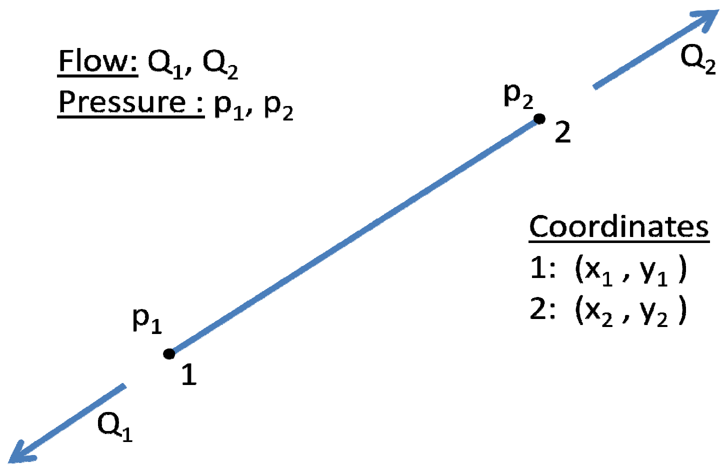


Figure 3-11 Coordinates, pressure, and flow definitions for one-dimensional pipe element

$$Q = T \cdot p \tag{3.17a}$$

$$\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = \begin{bmatrix} -\dfrac{A \cdot D^2}{32 \cdot \mu \cdot L} & \dfrac{A \cdot D^2}{32 \cdot \mu \cdot L} \\ \dfrac{A \cdot D^2}{32 \cdot \mu \cdot L} & -\dfrac{A \cdot D^2}{32 \cdot \mu \cdot L} \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \tag{3.17b}$$

Next, we assume uncoupled behavior between the mechanical behavior of the truss and the flow taking place inside the pipe. We now define the generalized force vector in global coordinates:

$$\hat{q} = \begin{bmatrix} q_1 \\ q_2 \\ Q_1 \\ q_3 \\ q_4 \\ Q_2 \end{bmatrix} \tag{3.18}$$

and the generalized displacements vector:

$$\hat{u} = \begin{bmatrix} u_1 \\ u_2 \\ p_1 \\ u_3 \\ u_4 \\ p_2 \end{bmatrix} \tag{3.19}$$

By combining the previous forms we have the generalized equilibrium:

$$\hat{q} = k \cdot \hat{u} \tag{3.20a}$$

$$\begin{bmatrix} q_1 \\ q_2 \\ Q_1 \\ q_3 \\ q_4 \\ Q_2 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & 0 & -k_{11} & -k_{12} & 0 \\ k_{12} & k_{22} & 0 & -k_{12} & -k_{22} & 0 \\ 0 & 0 & k_{33} & 0 & 0 & -k_{33} \\ -k_{11} & -k_{12} & 0 & k_{11} & k_{12} & 0 \\ -k_{12} & -k_{22} & 0 & k_{12} & k_{22} & 0 \\ 0 & 0 & -k_{33} & 0 & 0 & k_{33} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ p_1 \\ u_3 \\ u_4 \\ p_2 \end{bmatrix} \tag{3.20b}$$

$$k_{11} = \cos^2(\theta) \tag{3.20c}$$

$$k_{12} = \cos(\theta) \cdot \sin(\theta) \tag{3.20d}$$

$$k_{22} = \sin^2(\theta) \tag{3.20e}$$

$$k_{33} = -\frac{A \cdot D^2}{32 \cdot \mu \cdot L} \tag{3.20f}$$

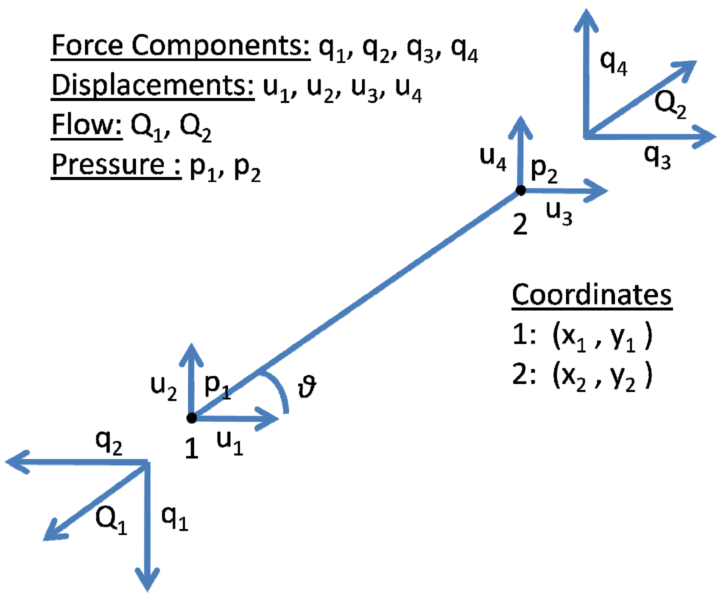


Figure 3-12 Coordinates, pressure, flow, force, and displacement definitions for the uncoupled drain element

3.4.2.3 Opensees Implementation

The finite element is implemented in the Opensees Software framework, to be used together with quad_up elements, in order to predict drainage inside soil layers by means of drains(e.g. PV-drains or stone columns). A new command is added in the interpreter that takes the arguments:

element Pipelin2 eleid node1 node2 Material Area C_l γ_w

eleid is the id of the element, a unite integer number assigned to this element, **node1** and **node2** are the start and end nodes of the element, **Material** is a an integer number already defined assigning a specific constitutive material for the mechanical response, **Area** is the area of the drain that contributes to the mechanical behavior, $\boldsymbol{\gamma_w}$ is the density of the water times the acceleration of gravity (e.g. negative when pointing downwards). The source code of the implementation is presented in Appendix F.

3.4.3 Turbulent Flow

3.4.3.1 Flow equations

Turbulent flow is a flow regime characterized by chaotic property changes. For fully turbulent flow according to the Moody Diagram (Figure 3-9) λ is a constant. From the Darcy-Weisbach equation:

$$\Delta P = \lambda \cdot \frac{L}{D} \cdot \frac{\rho V^2}{2} \tag{3.21a}$$

$$\Delta P = \lambda \cdot \frac{L}{D} \cdot \frac{\rho (\frac{Q}{A})^2}{2} \Rightarrow \tag{3.21b}$$

$$\Delta P = \frac{\lambda \cdot L \cdot \rho}{2 \cdot D \cdot A^2} \cdot Q^2 \Rightarrow \tag{3.21c}$$

$$\mathrm{Q} = \sqrt{\frac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \sqrt{\Delta P} \tag{3.22a}$$

$$\mathrm{Q} = C_t \cdot \sqrt{\mathrm{i}} \tag{3.22b}$$

the coefficient C_t [$L^{4.5}F^{-0.5}T^{-1}$] is defined by:

$$C_t = \sqrt{\frac{2 \cdot D \cdot A^2}{\lambda \cdot \rho}} \tag{3.23}$$

which can be written in a rate form:

$$\dot{\mathrm{Q}} = \sqrt{\frac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \sqrt{\dot{\Delta P}} \Rightarrow \tag{3.24a}$$

$$\dot{\mathrm{Q}} = \sqrt{\frac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \frac{1}{2\sqrt{\Delta \mathrm{P}}} \cdot \Delta \dot{P} \tag{3.24b}$$

3.4.3.2 *Finite Element Approximation*

One can define a one-dimensional element in a two dimensional space relating water pressure to flow, with the same conventions used in Figure 3-11. In the finite element approximation of the turbulent flow regime for the drains we are using a consistent Jacobian formulation.

We first define the vector of rate of flow (equivalent to the vector of external forces in the truss element):

$$\dot{Q} = \begin{bmatrix} \dot{Q}_1 \\ \dot{Q}_2 \end{bmatrix} \tag{3.25}$$

$$\Delta Q = \begin{bmatrix} \Delta Q_1 \\ \Delta Q_2 \end{bmatrix} \tag{3.26}$$

And the vector of rates of pressures:

$$\dot{p} = \begin{bmatrix} \dot{p}_1 \\ \dot{p}_2 \end{bmatrix} \tag{3.27}$$

$$\Delta p = \begin{bmatrix} \Delta p_1 \\ \Delta p_2 \end{bmatrix} \tag{3.28}$$

During the nth step of the integration:

$$\mathrm{Q}_{1(\mathrm{n})} = sign(i_n) \cdot C_t \cdot \sqrt{\mathrm{i}_{(\mathrm{n})}} \tag{3.29}$$

And at the n+1th step:

$$\mathrm{Q}_{1(\mathrm{n+1})} = sign(i_n) \cdot C_t \cdot \sqrt{\mathrm{i}_{(\mathrm{n+1})}} \tag{3.30}$$

So:

$$\Delta\mathrm{Q}_{(\mathrm{n})} = Q_{1(n+1)} - Q_{1(n)} \tag{3.31}$$

We also define $\Delta p_{1(n)}$ and $\Delta p_{2(n)}$ as the increments of pore pressure in node 1 and 2 of the drain element respectively. Also:

$$\Delta p_{(n)} = \Delta p_{2(n)} - \Delta p_{1(n)} \tag{3.32}$$

Using this we can write the consistent transmissivity matrix:

$$T_{cons} = \begin{bmatrix} -\dfrac{\Delta Q_n}{\Delta p_{(n)}} & \dfrac{\Delta Q_n}{\Delta p_{(n)}} \\ \dfrac{\Delta Q_n}{\Delta p_{(n)}} & -\dfrac{\Delta Q_n}{\Delta p_{(n)}} \end{bmatrix} \tag{3.33}$$

Then the equilibrium equation in this element is defined as:

$$\Delta Q = T_{cons} \cdot \Delta p \tag{3.34a}$$

$$\begin{bmatrix} \Delta Q_1 \\ \Delta Q_2 \end{bmatrix} = \begin{bmatrix} -\dfrac{\Delta Q_n}{\Delta p_{(n)}} & \dfrac{\Delta Q_n}{\Delta p_{(n)}} \\ \dfrac{\Delta Q_n}{\Delta p_{(n)}} & -\dfrac{\Delta Q_n}{\Delta p_{(n)}} \end{bmatrix} \cdot \begin{bmatrix} \Delta p_1 \\ \Delta p_2 \end{bmatrix} \tag{3.34b}$$

This formulation has a disadvantage. $\Delta p_{(n)}$ might be very low or zero, and this could cause numerical errors, or no convergence. When $\Delta p_{(n)}$ is very small then we can instead use the continuum Jacobian. By using equation 6.35 we can define a continuous transmissivity matrix:

$$T_{cont} = \begin{bmatrix} -\sqrt{\dfrac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \dfrac{1}{2\sqrt{\Delta \mathrm{P}}} & \sqrt{\dfrac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \dfrac{1}{2\sqrt{\Delta \mathrm{P}}} \\ \sqrt{\dfrac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \dfrac{1}{2\sqrt{\Delta \mathrm{P}}} & -\sqrt{\dfrac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \dfrac{1}{2\sqrt{\Delta \mathrm{P}}} \end{bmatrix} \tag{3.35}$$

Then the equilibrium equation in this element is defined as:

$$\dot{Q} = T_{cont} \cdot \dot{p} \tag{3.36a}$$

$$\begin{bmatrix} \dot{Q}_1 \\ \dot{Q}_2 \end{bmatrix} = \begin{bmatrix} -\sqrt{\dfrac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \dfrac{1}{2\sqrt{\Delta \mathrm{P}}} & \sqrt{\dfrac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \dfrac{1}{2\sqrt{\Delta \mathrm{P}}} \\ \sqrt{\dfrac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \dfrac{1}{2\sqrt{\Delta \mathrm{P}}} & -\sqrt{\dfrac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \dfrac{1}{2\sqrt{\Delta \mathrm{P}}} \end{bmatrix} \cdot \begin{bmatrix} \dot{p}_1 \\ \dot{p}_2 \end{bmatrix} \tag{3.36b}$$

As we can see a problem still rises in the continuum Jacobian matrix, when ΔP is very close to zero, or else when i, the hydraulic gradient, is very small. Remembering that we only need the continuous Jacobian when the $\Delta p_{(n)}$ is very close to zero then, we derive one more scheme to be used numerically when both (a) i is very small (b) $\Delta p_{(n)}$ is very small. Under these circumstances we linearize the Q vs i equation, and we assume a linear region of size $2d_c$.

$$\mathrm{Q} = \frac{C_t}{\sqrt{d_c}} \cdot \frac{\Delta P}{L} \tag{3.37}$$

Which can be written in a form similar to laminar flow:

$$T = \begin{bmatrix} -\dfrac{C_t}{\sqrt{d_c} \cdot L} & \dfrac{C_t}{\sqrt{d_c} \cdot L} \\ \dfrac{C_t}{\sqrt{d_c} \cdot L} & -\dfrac{C_t}{\sqrt{d_c} \cdot L} \end{bmatrix} \tag{3.38}$$

Next, we assume uncoupled behavior between the mechanical behavior of the truss and the flow taking place inside the pipe. We now define the generalized force rate vector in global coordinates:

$$\dot{\hat{q}} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{Q}_1 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{Q}_2 \end{bmatrix} \tag{3.39}$$

and the generalized displacement rate vector:

$$\dot{\hat{u}} = \begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{p}_1 \\ \dot{u}_3 \\ \dot{u}_4 \\ \dot{p}_2 \end{bmatrix} \tag{3.40}$$

The generalized equilibrium has the form:

$$\dot{\hat{q}} = \hat{k} \cdot \dot{\hat{u}} \tag{3.41a}$$

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{Q}_1 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{Q}_2 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & 0 & -k_{11} & -k_{12} & 0 \\ k_{12} & k_{22} & 0 & -k_{12} & -k_{22} & 0 \\ 0 & 0 & k_{33} & 0 & 0 & -k_{33} \\ -k_{11} & -k_{12} & 0 & k_{11} & k_{12} & 0 \\ -k_{12} & -k_{22} & 0 & k_{12} & k_{22} & 0 \\ 0 & 0 & -k_{33} & 0 & 0 & k_{33} \end{bmatrix} \cdot \begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{p}_1 \\ \dot{u}_3 \\ \dot{u}_4 \\ \dot{p}_2 \end{bmatrix} \tag{3.41b}$$

$$k_{11} = \cos^2(\theta) \tag{3.41c}$$

$$k_{12} = \cos(\theta) \cdot \sin(\theta) \tag{3.41d}$$

$$k_{22} = \sin^2(\theta) \tag{3.41e}$$

$\Delta p_{(n)} > d_c$:

$$k_{33} = -\frac{\Delta Q_n}{\Delta p_{(n)}} \tag{3.41f}$$

$\Delta p_{(n)} < d_c$ and $i > d_c$:

$$k_{33} = -\sqrt{\frac{2 \cdot D \cdot A^2}{\lambda \cdot L \cdot \rho}} \cdot \frac{1}{2\sqrt{\Delta P}} \tag{3.41g}$$

$\Delta p_{(n)} < d_c$ and $i < d_c$:

$$k_{33} = -\frac{C_t}{\sqrt{d_c} \cdot L} \tag{3.41h}$$

In Figure 3-13 a summary of the used approximations and regimes is shown.

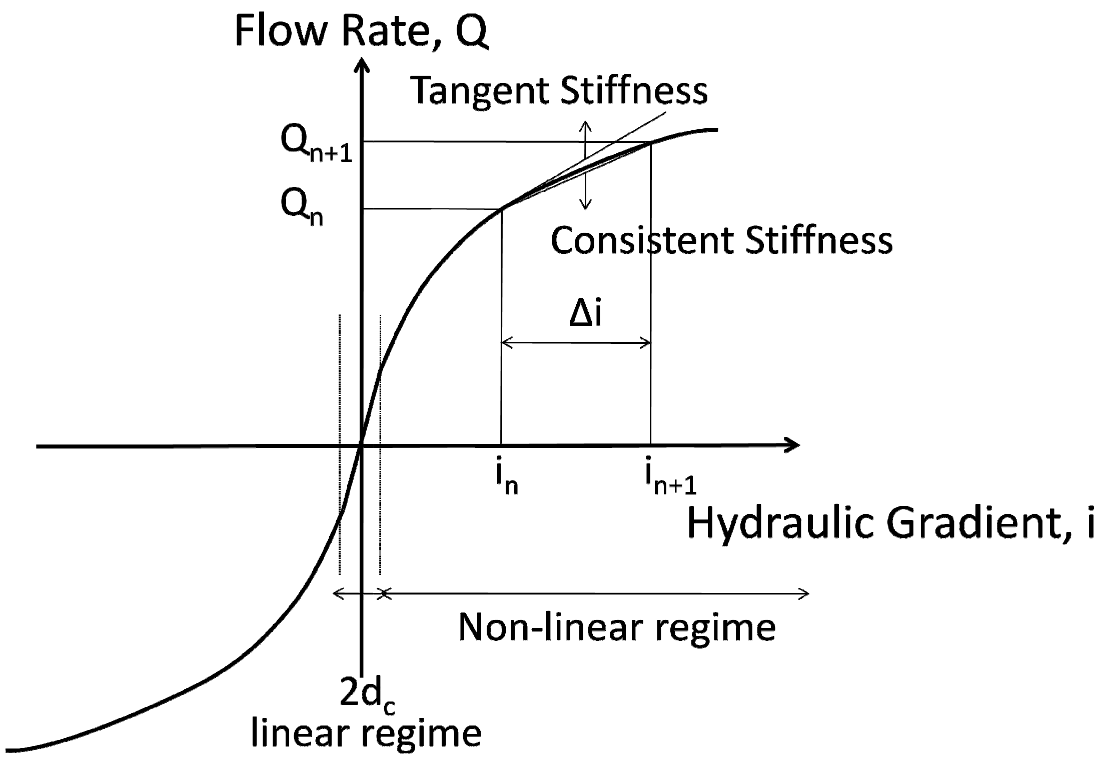


Figure 3-13 Various regimes and definitions used to integrate the Darcy-Weisbach equation for fully turbulent flow

3.4.3.3 Opensees Implementation

The finite element is implemented in the Opensees Software framework, to be used together with quadUP elements, in order to predict drainage inside soil layers by means of drains of any type. A new command is added in the interpreter that takes the arguments:

element Pipelin2 eleid node1 node2 Material Area C_t γ_w

eleid is the id of the element, a unite integer number assigned to this element, **node1** and **node2** are the start and end nodes of the element, **Material** is a an integer number already defined assigning a specific constitutive material for the mechanical response, **Area** is the area of the drain that contributes to the mechanical behavior, γ_w is the density of the water times the acceleration of gravity (e.g. negative when pointing downwards). The relevant source code is presented in Appendix G.

3.4.4 Storage Capacity

In typical applications the water level inside the drain is not at the ground surface. Hence, there is a storage effect within the drain as the water rises above the ambient groundwater water table. Pestana et al. (1997) have shown that the drain storage capacity can reduce the effectiveness and applicability of PV drains for liquefaction risk mitigation.

Since, the proposed elements compute the flow inside the drain (with fluid supplied from the surrounding soil) it is possible to integrate water coming out of drain and calculate the pore pressure condition on the top of the drain based on the height of water inside the pipe. A varying water level inside the pipe means a variable boundary condition at the top of the pipe. According to Figure 3-14, the pore pressure condition at point A is updated at every time step according to water level inside the drain H.

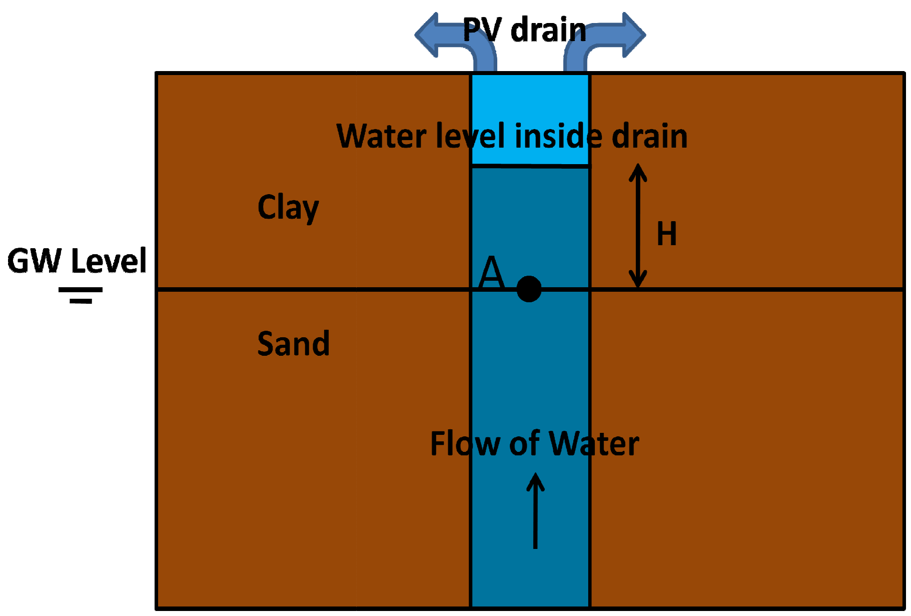
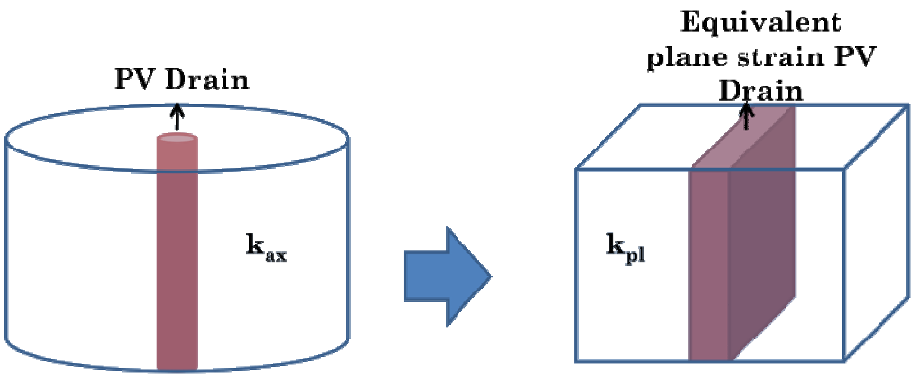


Figure 3-14 Schematic view of the storage capacity effect mechanisms

3.4.5 Representation of drain elements in 2-D plane strain FE analyses

Although in principle the proposed drain elements can be readily implemented together with coupled 3-D soil elements, the high computational cost of 3-D coupled, non-linear analyses is beyond the scope of the current thesis. In the current work the drain elements are encoded with 2-D quad-up elements for plane strain analyses. In the plane strain FE models, there is a geometric modification from radial to planar flow into the drains.

Hird et al, (1992) have investigated the use of two-dimensional, plane strain approximations for modeling the consolidation (of low permeability clays) with arrays of PV drains. They propose an equivalence that allows a true-radial consolidation problem around a drain to be simulated in the plane strain model, using assumptions of equal strain. Their results enable selection of equivalent drain spacing or equivalent soil permeability based in order to achieve the same average degree of consolidation within the soil layer. Their findings are also applicable for PV drains in high permeability sand deposits. Their methodology and equivalent hydraulic conductivity are summarized in Figure 3.11 for the case where the spacing between the drains is the same in the 3-D (actual, real world) scenario and in the 2D plane strain model. Although the Hird et al. (1992) analyses match the average degree of consolidation in the two spaces, it should be noted that the plane strain model will not represent accurately the excess pore pressures at all points in the soil mass. Verification analyses are presented in Appendix I.



For **infinite permeability** drains inside a **uniform** soil:

$$k_{pl} = \frac{2k_{ax}}{3\left[\ln(n) - \frac{3}{4}\right]}$$

k_{ax} : true soil permeability,
k_{pl} : equivalent soil permeability in a plane stain analysis
n :drain spacing ratio

Figure 3.11 Axisymmetric to plane strain equivalence, in the case the distance between the drains is the same in the 3D and in the plane strain model (after Hird et al, 1992)

Following Hird's solution, the axisymmetric transmissivity of a laminar flow drain must be scaled, in order to model the effect of the drain in a plane strain model. Assuming the same drain spacing for both 3-D (physical space) and in the 2-D model, then the equivalent laminar drain coefficient is:

$$C_l^{pl} = \frac{2}{\pi R} C_l^{ax} \qquad (3.42)$$

where R is the drain spacing in 3D.

Equation 3.42 was derived by solving analytically the consolidation problem of a plane strain unit cell with a laminar flow drain. It is more difficult to find the equivalence for fully-turbulent flow. In this case, consolidation with the fully turbulent flow is matched to equivalent laminar drain properties (using equation 4.23). Since the drain properties C_l and C_t are linearly dependent on each other, then the same scaling factor can be applied for fully turbulent flow cases:

$$C_t^{pl} = \frac{2}{\pi R} C_t^{ax} \sqrt{w} \qquad (3.43)$$

where w is the width of the plane strain finite element grid (in general w=1m for plane strain analyses)

4 Centrifuge Experiments

4.1 Introduction

Apart from measurements at well instrumented field sites (e.g., Lotung; Zeghal et al., 1995), there are practically no direct measurements of soil performance during real earthquakes. As a result, laboratory experiments play a vital role in the validation of numerical analyses. In practice there are only two classes of laboratory experiment that have been used to study problems of soil-structure interaction i) centrifuge models, and ii) shaking table experiments.

Centrifuge models simulate gravitational stress fields within a soil mass at reduced geometrical scale though centrifugal loading. The key components of successful centrifuge model tests for dynamic soil-structure interaction problems are: i) careful application of scaling laws (e.g., Schofield & Steedman, 1988); ii) quality of base shaking actuator; and iii) design and calibration of instrumentation for operation at high centrifugal accelerations. There has been a substantial investment in geotechnical centrifuge facilities around the US (Figure 4.1), including an NSF-funded national test facility at the University of California, Davis. In contrast, most of the large-scale shaking table experiments have been performed in Japan (facilities include the Port and Harbour Research Institute, PHRI, and Public Works Research Institute, PWRI).

The use of PV drains for mitigation of liquefaction has recently been investigated in centrifuge model tests performed at UC Davis (Kamai et al., 2008). These data are compared with numerical simulations using Opensees (and the proposed drain elements) in Chapter 5 of this thesis. This chapter considers the scaling laws for designing centrifuge model tests on geotechnical earthquake problems, and specifically considers the scaling of flow in PV drains (Section 4.3). Section 4.3 gives details of the centrifuge model reported by Kamai et al. (2008).

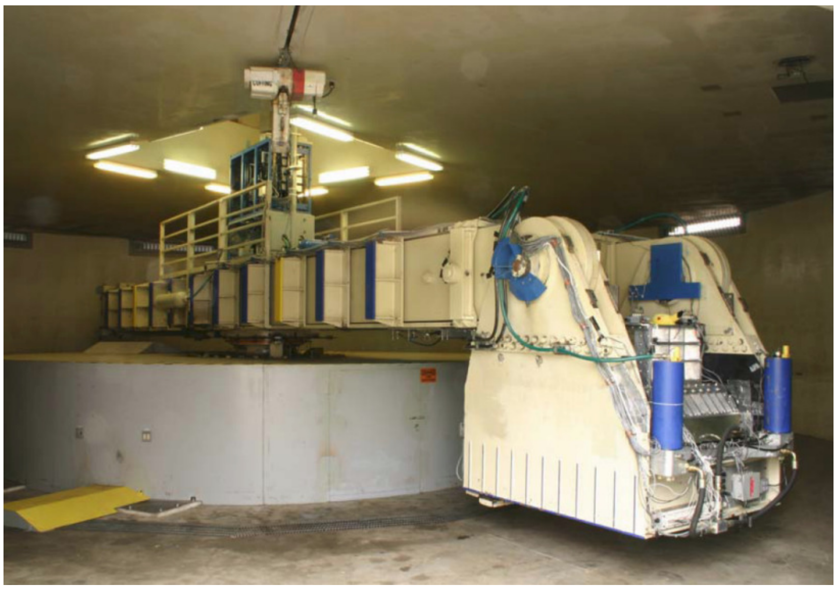


Figure 4-1 The UC-Davis geotechnical centrifuge

4.2 Scaling principles for geotechnical earthquake problems

In a centrifuge test, it is important to relate parameters measured in the scale model (M) to the prototype (P) full scale situation. In the following we present the basic scaling laws with particular focus on the use of PV earthquake drains. The equilibrium equations (eqn. 2.1) for the coupled pore-pressure displacement problem (neglecting convective terms) are:

Model:
$$\frac{\partial \sigma^M}{\partial x^M} + \rho^M b^M - \rho^M a^M - \rho_f^M \dot{w}^M = 0 \qquad (4.1a)$$

Prototype:
$$\frac{\partial \sigma^P}{\partial x^P} + \rho^P b^P - \rho^P a^P - \rho_f^P \dot{w}^P = 0 \qquad (4.1b)$$

Where σ is the stress matrix, *x* is the length, ρ is the material total density, b is the body force vector, α is the acceleration of the soil skeleton, and w is the velocity of water relative to the soil skeleton.

In the model, the physical length scale is reduced by a factor N, corresponding to the gravitational acceleration applied in the centrifuge:

$$x^M = x^P/N \tag{4.2}$$

$$a^M = Na^P \tag{4.3}$$

This is possible only if the timescale is also scaled according to:

$$t^M = t^P/N \tag{4.4}$$

The body forces and average acceleration of the pore fluid are also scaled by N:

$$b^M = Nb^P \tag{4.5}$$

$$\dot{w}^M = N\dot{w}^P \tag{4.6}$$

However, in order to match the diffusion equation in the soil (eqn. 2.2), the hydraulic conductivity of the pore fluid must also be scaled:

$$k^M = k^P/N \tag{4.7}$$

In order to achieve this scaling requirement in the centrifuge model, one can either use a soil material with lower conductivity than the prototype; or use a pore fluid with lower viscosity but the same density as water (e.g., silicon oil). The first approach is problematic as changing soil type can also affect important mechanical properties, while changes in the pore fluid can also create practical difficulties.

It is clear that scaling limitations are significant for modeling dynamic problems with diffusion of pore fluid. These difficulties are further confounded when dealing with partially saturated soils. Here it is impossible to scale consistently diffusion and inertial forces. Other similitude problems arise when modeling dynamic coupled pore pressures and displacement problems. For example, the undrained shear strength of clays typically increases by 5% to 15% for every log cycle of strain rate (Lacasse et al., 1970; Randolph et al., 2005). The effects of particle size are not well understood. However, through modeling of models, the effects of particle scale are eliminated in problems of soil-structure interaction by ensuring minimum ratios of characteristic structural width to particle size:

Footings $$\frac{B}{d_{50}} > 35 \tag{4.8a}$$

Piles $$\frac{B}{d_{50}} > 45 \quad or\ 60 \tag{4.8b}$$

where B is the width or diameter of the model foundation.

However, the development of shear banding seems to be significantly affected by the grain size, as shown from model tests on a trap door and on a cavity collapse problem (Stone & Muir, 1992). It is also important to account of non-uniformities in the centrifuge acceleration field on the vertical stress in soil samples (Schofield, 1980). By generating shear stresses on vertical planes, variable acceleration produces a 'silo/arching effect' that can affect significantly the vertical stress in depth. The non uniformity of the gravitational field significantly influences the measured vertical displacements, since every horizontal soil section wants to move to an equipotential level, thus curving towards the center of rotation.

4.3 Scaling laws for PV-drains

The flow coming out a drain in the prototype is related to the flow coming out of the model drain in the following way:

$$\mathrm{Q^P} = \frac{\Delta V^P}{T^P} = \frac{\Delta V^M \cdot N^3}{T^M \cdot N} = \frac{\Delta V^M}{T^M} \cdot N^2 = Q^M \cdot N^2 \tag{4.9}$$

So for laminar flow:

$$\mathrm{Q} = \mathrm{C_l} \cdot \frac{\Delta P}{\mathrm{L}} \tag{4.10a}$$

$$\mathrm{C_l^P} \cdot \frac{\Delta P}{\mathrm{L^P}} = \mathrm{C_l^M} \cdot \frac{\Delta P}{\mathrm{L^M}} N^2 \tag{4.10b}$$

$$\mathrm{C_l^P} \cdot \frac{\Delta P}{\mathrm{N \cdot L^M}} = \mathrm{C_l^M} \cdot \frac{\Delta P}{\mathrm{L^M}} N^2 \tag{4.10c}$$

$$\mathrm{C_l^P} = \mathrm{C_l^M} \cdot N^3 \tag{4.10d}$$

And for turbulent flow:

$$\mathrm{Q} = \mathrm{C_t} \cdot \sqrt{\frac{\Delta P}{L}} \tag{4.11a}$$

$$\mathrm{C_t^p} \cdot \sqrt{\frac{\Delta P}{L^P}} = \mathrm{C_t^M} \cdot \sqrt{\frac{\Delta P}{L^M}} N^2 \tag{4.11b}$$

$$\mathrm{C_t^p} \cdot \sqrt{\frac{\Delta P}{N \cdot L^M}} = \mathrm{C_t^M} \cdot \sqrt{\frac{\Delta P}{L^M}} N^2 \tag{4.11c}$$

$$\mathrm{C_t^P} = \mathrm{C_t^M} \cdot N^{2.5} \tag{4.11d}$$

It can be seen that the parameter relating the pore pressure gradient to the flow of water has to be N^3 times larger in the prototype scale when dealing with laminar flow drains and $N^{2.5}$

times larger for turbulent flow. From the above analysis a problem can arise due to difference in the Reynolds number at prototype and model scales:

$$\mathrm{Re^P} = \frac{\mathrm{V^P D^P}}{\mathrm{v}} \Rightarrow \tag{4.12a}$$

$$\mathrm{Re^P} = \frac{\frac{\mathrm{Q^P}}{A^P}\mathrm{D^P}}{\mathrm{v}} \Rightarrow \tag{4.12b}$$

$$\mathrm{Re^P} = \frac{\frac{\mathrm{N^2Q^M}}{N^2A^M}N\mathrm{D^M}}{\mathrm{v}} \Rightarrow \tag{4.12c}$$

$$\mathrm{Re^P} = \frac{Q^M\,\mathrm{D^M}}{\mathrm{A^M v}}\mathrm{N} \Rightarrow \tag{4.12d}$$

$$\mathrm{Re^P} = \mathrm{Re^M N} \tag{4.12e}$$

So a flow that is laminar in the model scale might be turbulent in the prototype scale. This is important because the experiments do not preserve Re at model scale.

Alternatively if a different pore fluid is used in the centrifuge model (to scale the diffusion process), then the scaling ratio for Reynolds number is further increased:

$$\mathrm{Re^P} = \mathrm{Re^M N^2} \tag{4.13}$$

4.4 Design of model scale PV-drains

When designing a scale model with drains one should design the transmissivity of the scale drains so that it matches the extra resistance caused by the turbulence in the real-scale drains, since it is very common that the model scale flow is mostly laminar and the prototype scale flow is mostly turbulent. A methodology is proposed at this point that would allow selection of pipes for scale modeling purposes.

In prototype scale we need to find a laminar flow parameter C_l that gives the smallest error compared with the flow coming out of a drain with parameter $C_{t,}$ for a specific range of i (pore pressure gradient). We define a function that is the square of the difference of the laminar flow calculations to the turbulent flow calculations.

$$f = \left(Q_t^P - Q_l^P\right)^2 \tag{4.14}$$

$$f = \left(C_t^P\sqrt{i} - C_l^P i\right)^2 \tag{4.15}$$

$$f = C_t^{P^2} i + C_l^{P^2} i^2 - 2C_t^P C_l^P i^{\frac{3}{2}} \tag{4.16}$$

Next we define the integral of function f:

$$F = \int_0^{imax} f\,di \tag{4.17}$$

$$F = \int_0^{imax} (C_t^{P^2} i + C_l^{P^2} i^2 - 2C_t^P C_l^P i^{\frac{3}{2}})\,di \tag{4.18}$$

$$F = \frac{C_t^{P^2} i_{max}^2}{2} + \frac{C_l^{P^2} i_{max}^3}{3} - \frac{4}{5} C_t^P C_l^P i_{max}{}^{\frac{5}{2}} \tag{4.19}$$

In order for the flow calculation for turbulent flow to match the calculation for the laminar flow we need to minimize F.

$$\frac{\partial F}{\partial C_l^P} = \frac{2}{3} C_l^P \; i_{max}{}^3 - \frac{4}{5} C_t^P i_{max}{}^{\frac{5}{2}} \tag{4.20}$$

$$\frac{2}{3} C_l^P \; i_{max}{}^3 - \frac{4}{5} C_t^P i_{max}{}^{\frac{5}{2}} = 0 \tag{4.21}$$

$$\frac{\partial^2 F}{\partial C_l^{P^2}} = \frac{2}{3} i_{max}{}^3 > 0 \; for \; i_{max} > 0 \; without \; loss \; of \; generality \tag{4.22}$$

$$C_l^P \; = \frac{6C_t^P}{5\sqrt{i_{max}}} \tag{4.23}$$

$$\mathrm{C_l^M} = \frac{6\mathrm{C_t^P}}{5\sqrt{i_{max}} \cdot N^3} \tag{4.24a}$$

If we replace the expressions for C_l and C_t we have:

$$\mathrm{C_l^M} = \frac{6}{5\sqrt{i_{max}} \cdot N^3}\sqrt{\frac{2 \cdot D^P \cdot {A^P}^2}{\lambda \cdot \rho}} \Rightarrow \tag{4.24b}$$

$$\frac{A^M \cdot {D^M}^2}{32 \cdot \mu} = \frac{6}{5\sqrt{i_{max}} \cdot N^3}\sqrt{\frac{2 \cdot D^P \cdot {A^P}^2}{\lambda \cdot \rho}} \Rightarrow \tag{4.24c}$$

$$D^M = \left(\frac{76.8\ \mu}{\sqrt{i_{max}} \cdot N^3}\sqrt{\frac{{D^P}^5}{2 \cdot \lambda \cdot \rho}}\right)^{\frac{1}{4}} \tag{4.25}$$

Equation 4.25 defines the design diameter for the model drains, given the parameters of the prototype drains. This relation has been produced under the assumption that the flow is laminar in the model scale and turbulent in the model scale. This allows for the model scale laminar flow to be as similar as it can be to the prototype scale turbulent flow by minimizing the squares of the distances between the flow calculated by the two theories. This relation can be inverted, in order to examine what is the prototype drain that the model drain is representing.

$$D^P = \left(\frac{i_{max} \cdot N^6 \cdot \lambda \cdot \rho}{2949.12\mu^2} {D^M}^8\right)^{\frac{1}{5}} \tag{4.26}$$

Also, a reasonable range for PV drains that would not make us loose accuracy in the range of small gradient, but sufficient to capture large gradients would be:

$$i_{max} = \frac{\Delta P}{\Delta h \cdot \gamma_w}\gamma_w \tag{4.27}$$

A reasonable value, considering the fact that in a normal setup very few excess pore pressures will develop along the line of the drain, in accordance with the FE simulations performed in this project, would be:

$$\frac{\Delta P}{\Delta h \cdot \gamma_w} = 1/100 \Rightarrow \tag{4.28a}$$

$$i_{max} = \frac{\gamma_w}{100} \tag{4.28b}$$

So the aforementioned relationships simplify as:

$$D^M = \left(\frac{768\ \mu}{\sqrt{\gamma_w} \cdot N^3} \sqrt{\frac{D^{P^5}}{2 \cdot \lambda \cdot \rho}} \right)^{\frac{1}{4}} \tag{4.29}$$

$$D^P = \left(\frac{\gamma_w \cdot N^6 \cdot \lambda \cdot \rho}{294912 \mu^2} D^{M^8} \right)^{\frac{1}{5}} \tag{4.30}$$

Also, one needs to notice that a fluid of different viscosity might be used to match diffusion between the scale model and the prototype, according to the following equation:

$$\mu^P = \frac{\mu^M}{N} \tag{4.31}$$

So we re-write the equations in the following form:

$$D^M = \left(\frac{76.8\ \mu^M}{\sqrt{i_{max}} \cdot N^3} \sqrt{\frac{D^{P^5}}{2 \cdot \lambda \cdot \rho}} \right)^{\frac{1}{4}} \Rightarrow \tag{4.32a}$$

$$D^M = \left(\frac{76.8\ \mu^P}{\sqrt{i_{max}} \cdot N^2} \sqrt{\frac{D^{P^5}}{2 \cdot \lambda \cdot \rho}} \right)^{\frac{1}{4}} \tag{4.32b}$$

$$D^P = \left(\frac{i_{max} \cdot N^6 \cdot \lambda \cdot \rho}{2949.12 \mu^{M^2}} D^{M^8} \right)^{\frac{1}{5}} \Rightarrow \tag{4.33a}$$

$$D^P = \left(\frac{i_{max} \cdot N^4 \cdot \lambda \cdot \rho}{2949.12 {\mu^P}^2} {D^M}^8 \right)^{\frac{1}{5}} \tag{4.33b}$$

So, under the assumption that:

$$\frac{\Delta P}{\Delta h \cdot \gamma_w} = 1/100 \Rightarrow \tag{4.34}$$

We have:

$$D^M = \left(\frac{768\ \mu^M}{\sqrt{\gamma_w} \cdot N^3} \sqrt{\frac{{D^P}^5}{2 \cdot \lambda \cdot \rho}} \right)^{\frac{1}{4}} \tag{4.35a}$$

$$D^M = \left(\frac{768\ \mu^P}{\sqrt{\gamma_w} \cdot N^2} \sqrt{\frac{{D^P}^5}{2 \cdot \lambda \cdot \rho}} \right)^{\frac{1}{4}} \tag{4.35b}$$

$$D^P = \left(\frac{\gamma_w \cdot N^6 \cdot \lambda \cdot \rho}{294912 {\mu^M}^2} {D^M}^8 \right)^{\frac{1}{5}} \tag{4.36a}$$

$$D^P = \left(\frac{\gamma_w \cdot N^4 \cdot \lambda \cdot \rho}{294912 {\mu^P}^2} {D^M}^8 \right)^{\frac{1}{5}} \tag{4.36b}$$

4.5 Centrifuge models of PV drains

In order to evaluate the effectiveness of PV drains for liquefaction remediation (Earthquake drains) a centrifuge test was performed at the centrifuge facility at UC Davis, at March 2007 (SSK01) (Kamai, et al., 2008). The test compares the response of two similar facing slopes with a central channel. Beneath the left side slope, is a 5m thick (in prototype scale) layer of loose sand containing an array of PV-drains, while beneath the right side slope is the loose sand is un-treated. The two sides were symmetrically sloped at 3° towards a 200 mm wide central channel and were both comprised of three distinct layers (Figure 4-2): (1) a bottom layer of dense Nevada Sand, overlain by (2) a liquefiable layer of loose Nevada Sand, which was overlain by (3) a layer of compacted Yolo Loam. The Yolo Loam was used to impede the vertical dissipation of pore water pressure out of (DeAlba, Chan, & Seed, 1975) the liquefiable layer. A series of harmonic shaking events were applied to the model at a centrifugal acceleration of approximately 15g. All events were applied transverse to the central channel in the (longitudinal) direction.

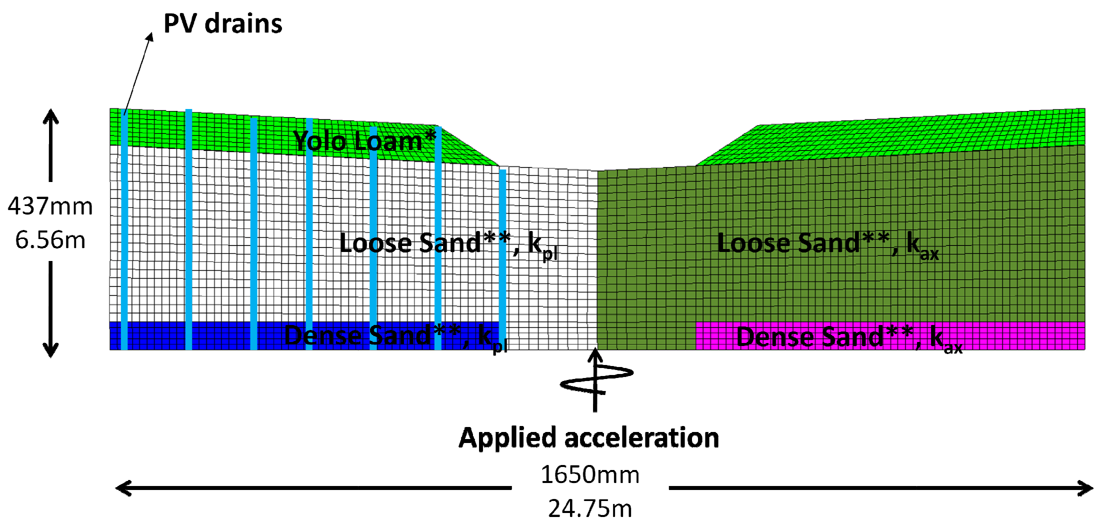


Figure 4-2 Conceptual diagram of the PV drains centrifuge model

The test had two important goals. The first was to provide a proof-of-concept, that, PV drains can be effective in reducing liquefaction resistance. The second was to provide reliable measurements of performance for validation of numerical analyses.

4.5.1 Model Preparation

The first stage in model preparation involved installation of the PV drains. These comprised 7mm ID nylon tubes with 1.5mm holes drilled every 5mm. The tubes were wrapped twice with a Precision Woven Polypropylene Mesh, to keep sand grains from clogging or entering the tube during liquefaction. The drains were placed in a triangular grid using wires for temporary support (see Fig. 4-8).

A large box pluviator was used to deposit the basal layer of dense (D_r = 84%) Nevada sand, while a barrel pluviator was used to construct the rest of the model of the loose sand layer (D_r = 40%). Subsequent measurements of relative density (based on volumes and weights) suggest that the loose sand may actually have been deposited at a slightly lower relative density (D_r = 30%)[4].

The crust material was constructed using natural Yolo Loam, which was sun dried and sieved (to pass a #10 sieve). Then water was added to reach the optimum water content of this soil (15%). The crust was placed in three layers.

Figure 4-3 shows the soil preparation procedures caused small movements of the PV drains from their initial positions.

Finally, the model was flooded with CO_2 and placed under a vacuum of approximately 90kPa to remove air within the soil. Then water was slowly dripped into saturation troughs, slowly saturating the model from bottom to top. Saturation was targeted so that the entire liquefiable layer would be saturated.

During construction of the scale model three types of sensors were installed. Pore pressure transducers to monitor excess pore pressure (Figure 4-5), displacement transducers (Figure 4-6) to measure horizontal and vertical displacements, and accelerometers (Figure 4-4) to measure

[4] A 10% difference in relative density corresponds to only a 2mm difference in height of the model layer.

amplification or de-amplification of the input motion in various locations. Kamai et al. (2008) give full details of the instrumentation used in the test. It should be noted that some of the embedded pore pressure transducers and accelerometers underwent large net settlements due to liquefaction induced in the test as shown in Figures 4-4a and 4-5a.

4.5.2 Scale Factors

The factors used to convert the data to prototype scale are indicated in Table 4-1.

Table 4-1 Scale Factors

Quantity	Prototype/Model Dimension
Time	15/1
Displacement, Length	15/1
Acceleration, Gravity	1/15
Pressure, Stress	1/1
Permeability	15/1

Based on the detailed evaluation of scaling for PV drains, it is interesting to evaluate the design of model drains used in SSK01. In this section we use the aforementioned scaling laws for the centrifuge experiment in order to examine similitude issues between the model and prototype drains. We know for our case that:

$$\gamma_w = 9810 N/m^3 \tag{4.37}$$

$$N = 15 \tag{4.38}$$

$$\frac{\epsilon}{d} = \frac{0.0025mm}{7mm} = 3.57 \cdot 10^{-4} \tag{4.39}$$

$$\lambda = 0.0085\ (approximate\ for\ prototype\ scale) \tag{4.40}$$

(from Moody's diagram for ε/d=2.4*10^{-5})

$$D^M = 0.007m \tag{4.41}$$

$$\mu = 0.001Pa \cdot s \tag{4.42}$$

$$\rho = 1000kg/m^3 \tag{4.43}$$

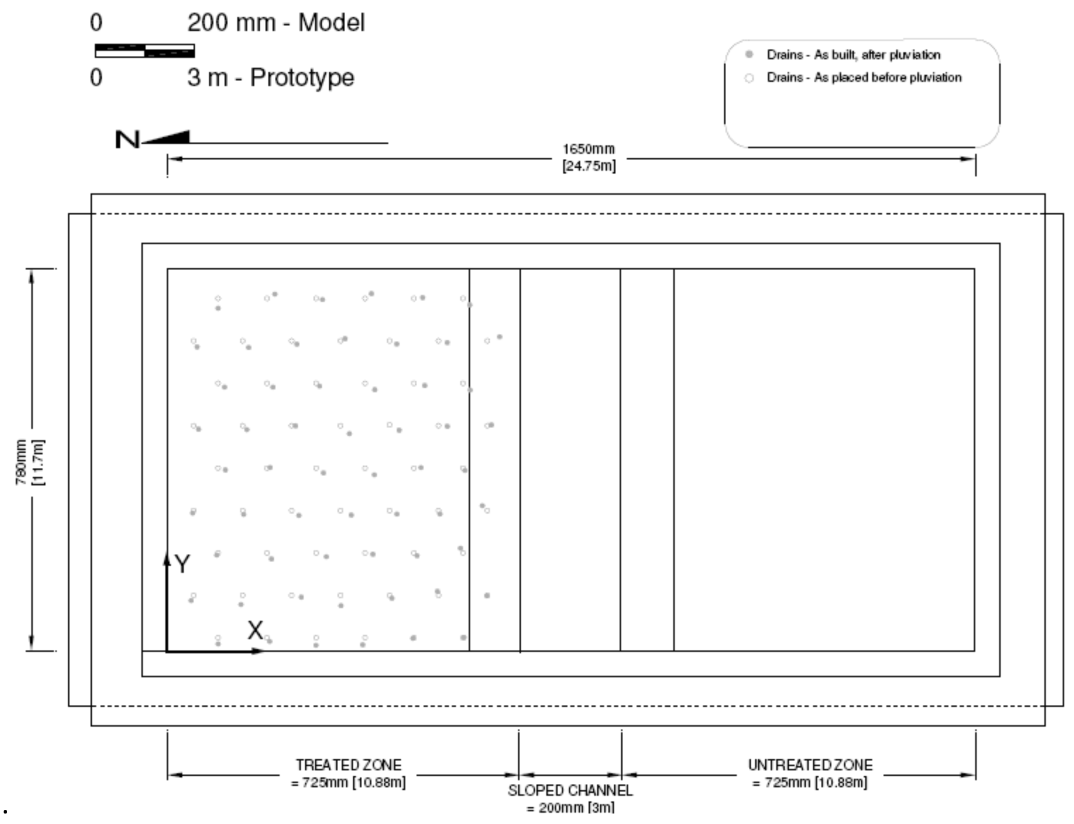
Which gives, using eqn. 4.26:

$$D^P = \left(\frac{25 \cdot 98.1 \frac{N}{m^3} \cdot 15^6 \cdot 0.0085 \cdot 1000 \frac{kg}{m^3}}{73728 \cdot 0.001^2} 0.007^8 \right)^{\frac{1}{5}} \tag{4.44}$$

$$D^P = 0.1132m \tag{4.45}$$

This results shows that the prototype drain should have a diameter, D^P = 113mm. This is slightly larger than expected from direct length scaling (i.e., D^p = 15(D^m) = 105mm). This means that under this range of i's the assumption of full turbulence in the drains makes them less permeable. For all practical purposes, under the above assumptions, the drain behavior should scale well for both laminar and turbulent flow.

It should be noted that the drains used in this experiment have a very smooth surface while actual PV drains will have much rougher surfaces due to bacterial growth (biofouling) and material deposition. Also, the maximum gradient used here might not be always that small depending on the permeability of the drains, the drain spacing, and the permeability of the soil.



0 200 mm - Model
0 3 m - Prototype
N
● Drains - As built, after pluviation
○ Drains - As placed before pluviation
1650mm [24.75m]
780mm [11.7m]
Y
X
TREATED ZONE = 725mm [10.88m]
SLOPED CHANNEL = 200mm [3m]
UNTREATED ZONE = 725mm [10.88m]

Figure 4-3 Location of the drains before and after pluviation

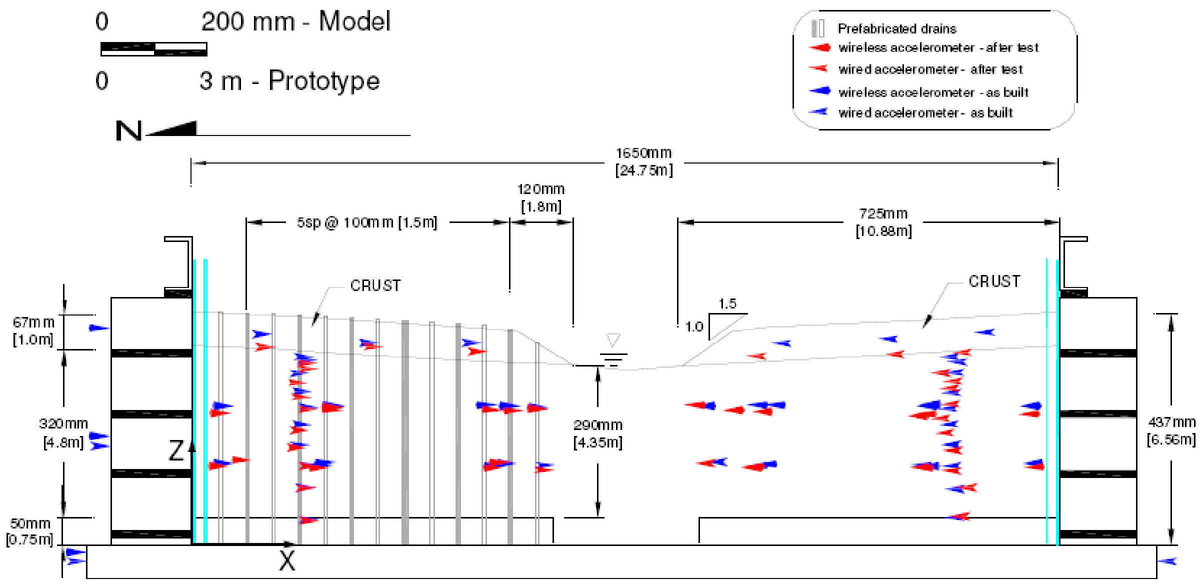
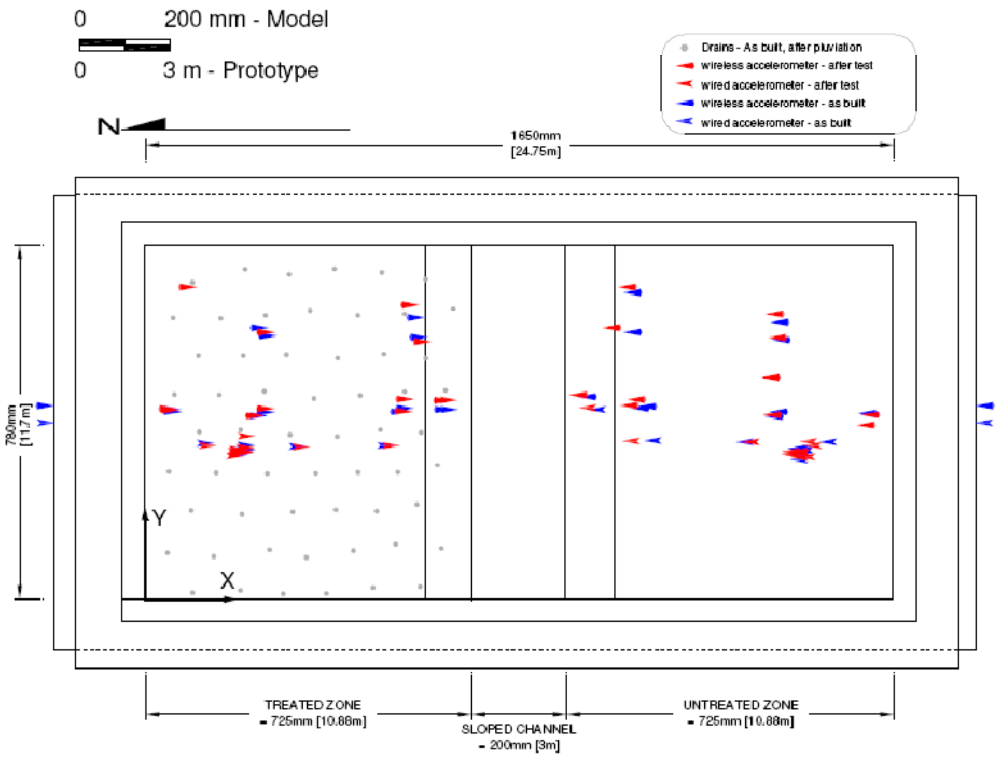


Figure 4-4 Accelerometer locations: cross section (as built and after test)



0 200 mm - Model
0 3 m - Prototype
N
Drains - As built, after pluviation
wireless accelerometer - after test
wired accelerometer - after test
wireless accelerometer - as built
wired accelerometer - as built
1650mm [24.75m]
780mm [11.7m]
Y
X
TREATED ZONE - 725mm [10.88m]
SLOPED CHANNEL - 200mm [3m]
UNTREATED ZONE - 725mm [10.88m]

Figure 4-5 Accelerometer locations: plan view (as built and after shaking)

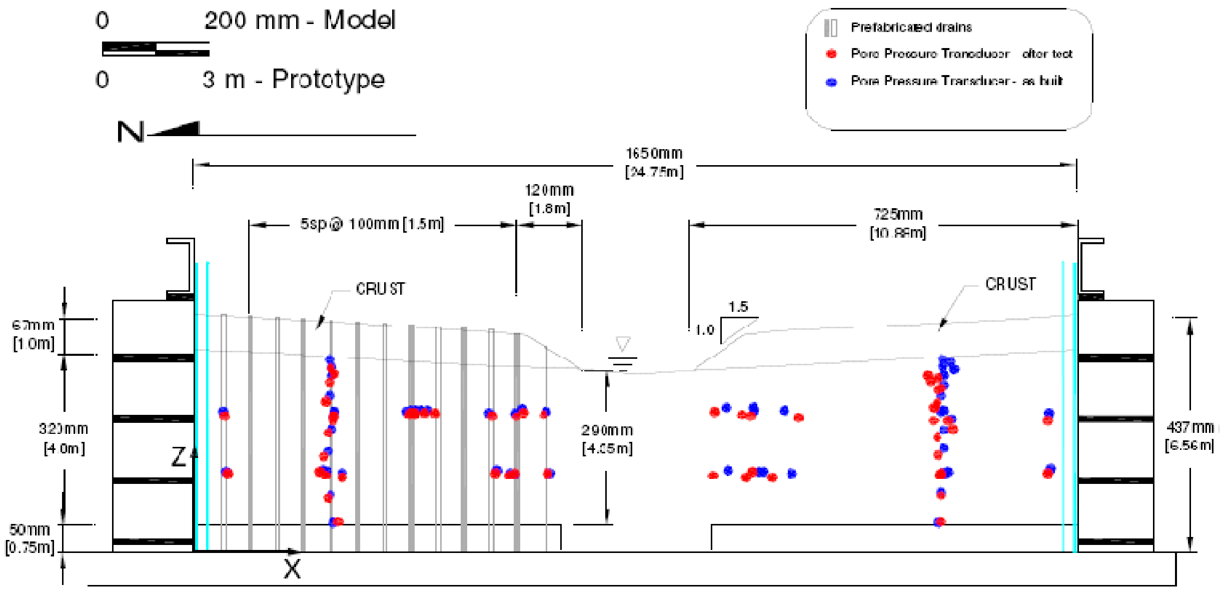


Figure 4-6 Pore pressure transducers: cross section (before and after shaking)

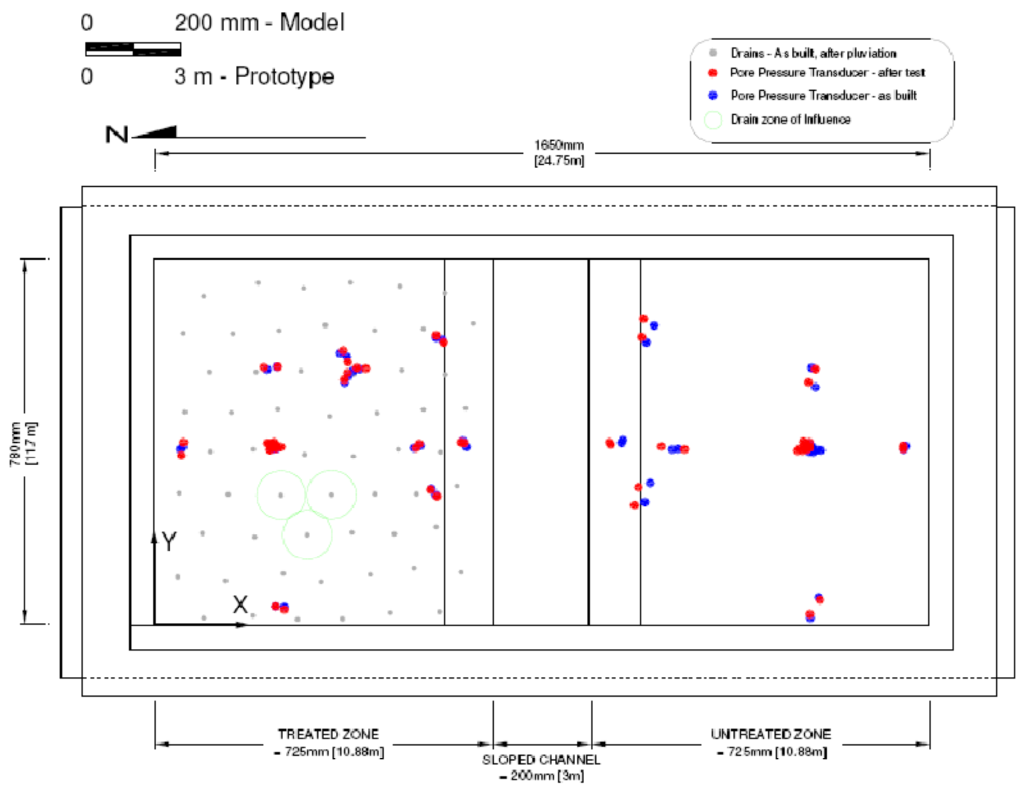

0 200 mm - Model
0 3 m - Prototype
N
Drains - As built, after pluviation
Pore Pressure Transducer - after test
Pore Pressure Transducer - as built
Drain zone of Influence
1650mm [24.75m]
780mm [11.7 m]
Y
X
TREATED ZONE - 725mm [10.88m]
SLOPED CHANNEL - 200mm [3m]
UNTREATED ZONE - 725mm [10.88m]

Figure 4-7 Pore pressure transducers: plan view (before and after shaking)

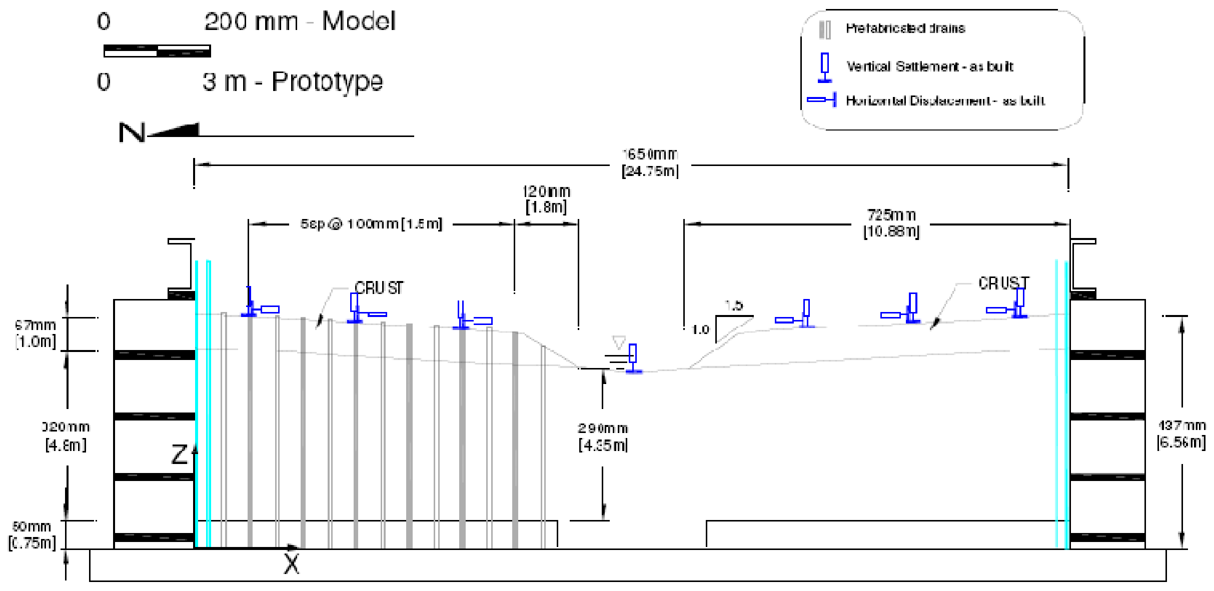


Figure 4-8 Displacement transducers: cross section

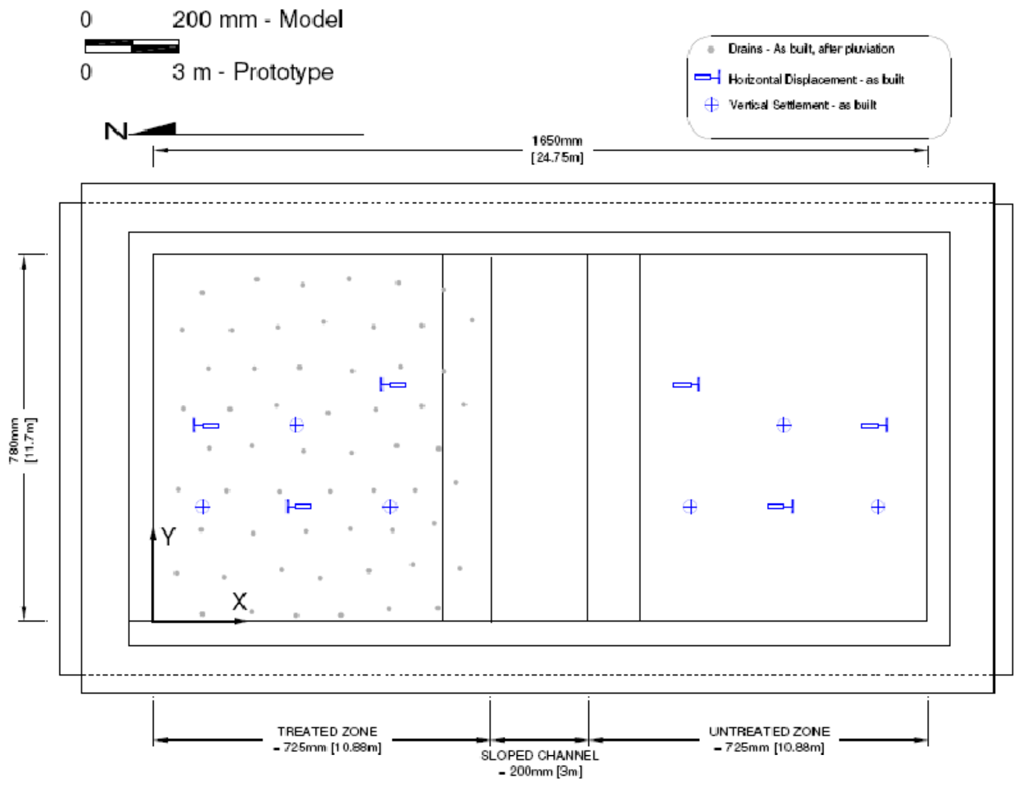

0
200 mm - Model
0
3 m - Prototype
N
Drains - As built, after pluviation
Horizontal Displacement - as built
Vertical Settlement - as built
1650mm [24.75m]
780mm [11.7m]
Y
X
TREATED ZONE - 725mm [10.88m]
SLOPED CHANNEL - 200mm [3m]
UNTREATED ZONE - 725mm [10.88m]

Figure 4-9 Displacement transducers: plan view

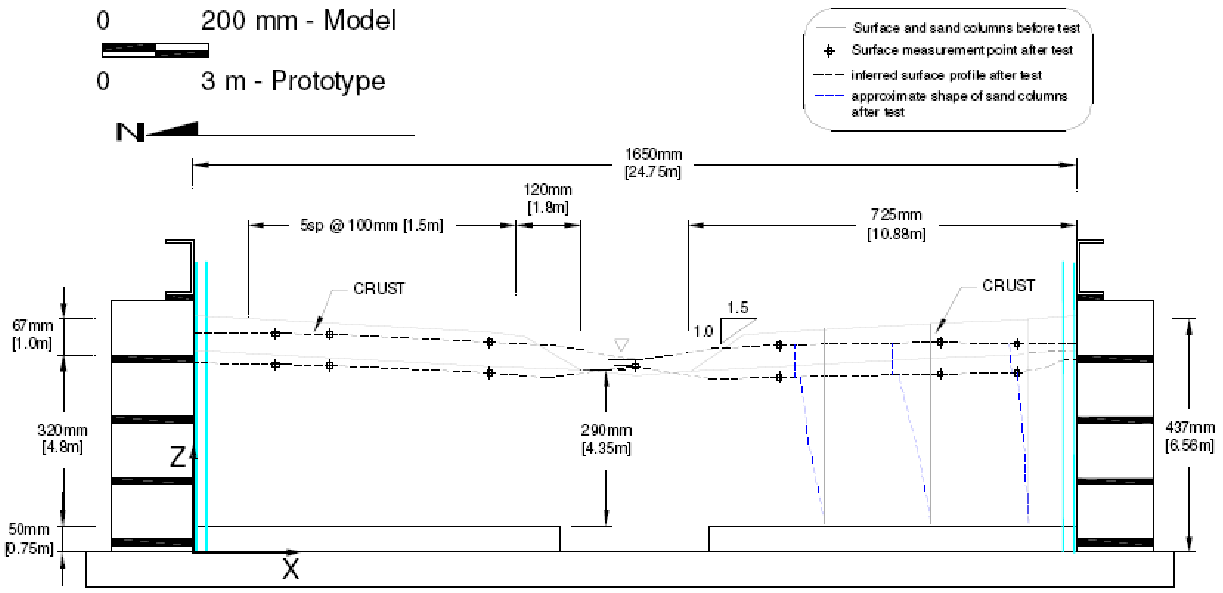


Figure 4-10 Initial and final deformed shape: cross section

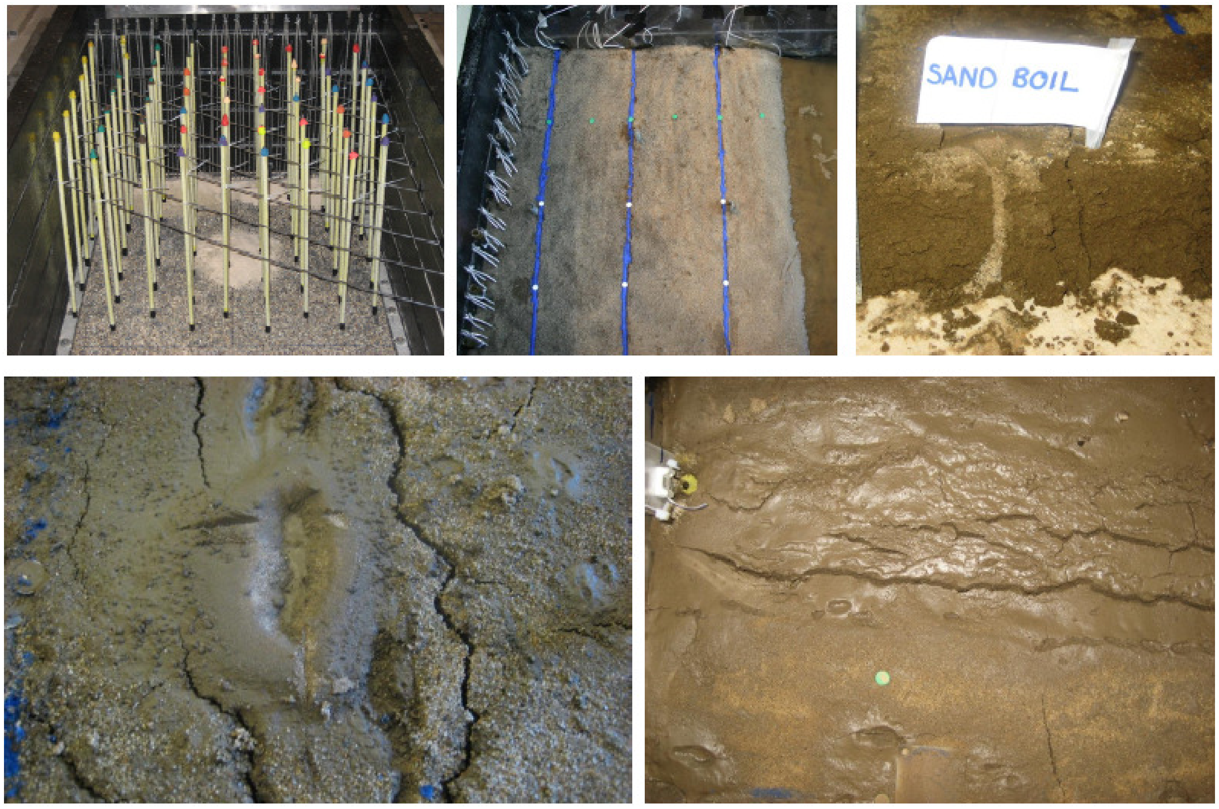


Figure 4-11 Model pictures – from top left corner, clockwise: (1) drains placed before pluviation (2) surface markers on untreated side (3) sand boil – cross section through crust (4)&(5) the untreated side after the test – cracking and sand boils.

5 Numerical Analyses of PV Drain Performance

This chapter describes 2-D plane strain analyses of PV drain performance through finite element modeling of the reference centrifuge model test SSK01 (Kamai et al., 2008) described in section 4.5. The finite element model incorporates the proposed drain elements for laminar and turbulent flow (Chapter 3).

5.1 Finite element model

Figures 5-1 and 5-2 illustrate the main features of the Opensees finite element model used to simulate centrifuge test SSK01:

1. Coupled flow and deformation in the high permeability sand layers are represented by QuadUP elements which include 4-nodes for bilinear interpolation or displacements and 4 nodes for bilinear interpolation of pore pressures. The current analyses use a regular grid of uniform-sized elements. The effective stress-strain-strength properties of the Nevada sand is characterized by the pressure dependent multi-yield surface model (PD-MYS02; Yang et al., 2002), with separate sets of input parameters for the dense and loose sand layers.

2. The capping layer of low permeability, compacted Yolo loam is represented by 8-noded Quad elements (quadratic displacement interpolation using total stresses), and its undrained mechanical properties are represented by the pressure independent multi-yield surface model (PI-MYS; section 2.3). The QuadUP and Quad domains are connected with equalDOF objects (Figure 5-1), a connection that ties the displacement DOF in a node of Quad element to a DOF in an adjacent node of a QuadUP element.

3. The centrifuge model is built within a laminar box (i.e., a shear box made from an assembly of hollow steel plates separated by bearings which minimize friction). This design ensures equal horizontal displacements at the lateral boundaries of the centrifuge model. These periodic boundary conditions are represented in the finite element model by constraining nodes at the left and right boundaries to with equal displacement degrees of freedom, while the mass of the plates is added to these boundary nodes (Fig. 5-3). The sand layers are continuous across the model (Fig. 5-1) and hence, remain in contact with the box walls throughout shaking. In contrast, the Yolo loam forms a partial cap and is absent in the central channel. During shaking events, the loam can separate from the walls of the laminar box. This behavior is modeled by introducing zero-thickness, no-tension elements between the Yolo loam and the walls of the box (Fig. 5-3).

4. The array of PV drains is installed on the left-side of the centrifuge model and is represented in the finite element model by a series of uniformly-spaced line elements (Fig. 5-2). Perfect drains are represented by imposing boundary conditions of zero excess pore pressure, while finite transmissivity drains are represented using the proposed drain elements (Section 3.3) for conditions of laminar or turbulent flow. The water table is located at the top of the sand layer in the centrifuge model, while the PV drains discharge at the ground surface (i.e, above the Yolo loam). Hence, there is a significant storage effect (Storage Volume/Drain Section Area=1m) which is also represented by the drain elements. The finite element approximation of planar flow (vs radial flow in the centrifuge model) is represented using an equivalent hydraulic conductivity (after Hird et al., 1992) to match the average degree of consolidation within the surrounding soil mass.

5. Seismic loading is represented by applying a uniform basal excitation (acceleration) across all nodes. The load history used in SSK01 mode comprises of five cycles of shaking as presented in Table 5-1. The first 2 cycles are used to test the model and the monitoring equipment. The last three cycles (a_{max}=0.07, 0.11, 0.3 g) constitute the part of the experiment that is going to be used for validation.

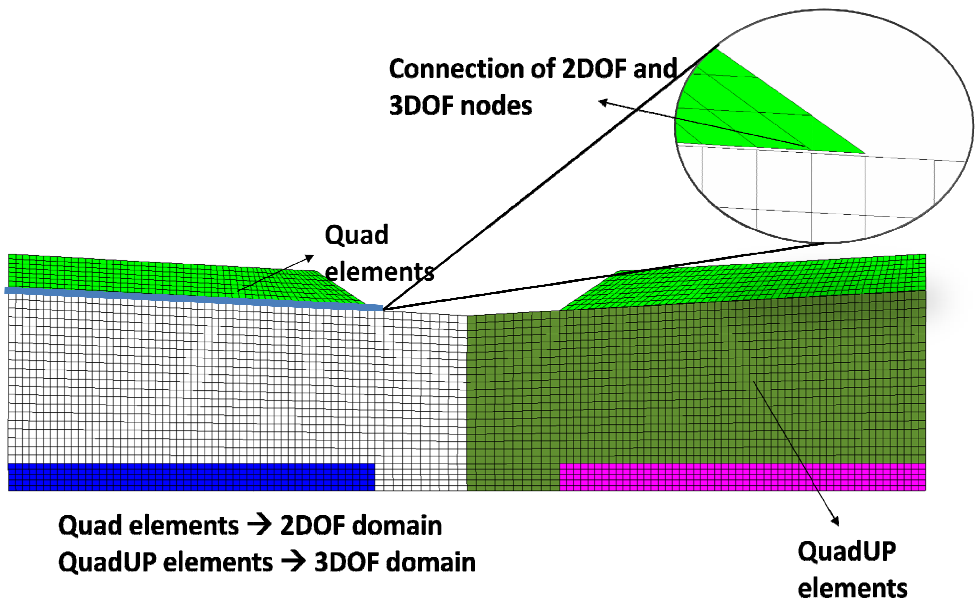

Connection of 2DOF and 3DOF nodes
Quad elements
Quad elements → 2DOF domain
QuadUP elements → 3DOF domain
QuadUP elements

Figure 5-1 Different domains employed in the FE analysis

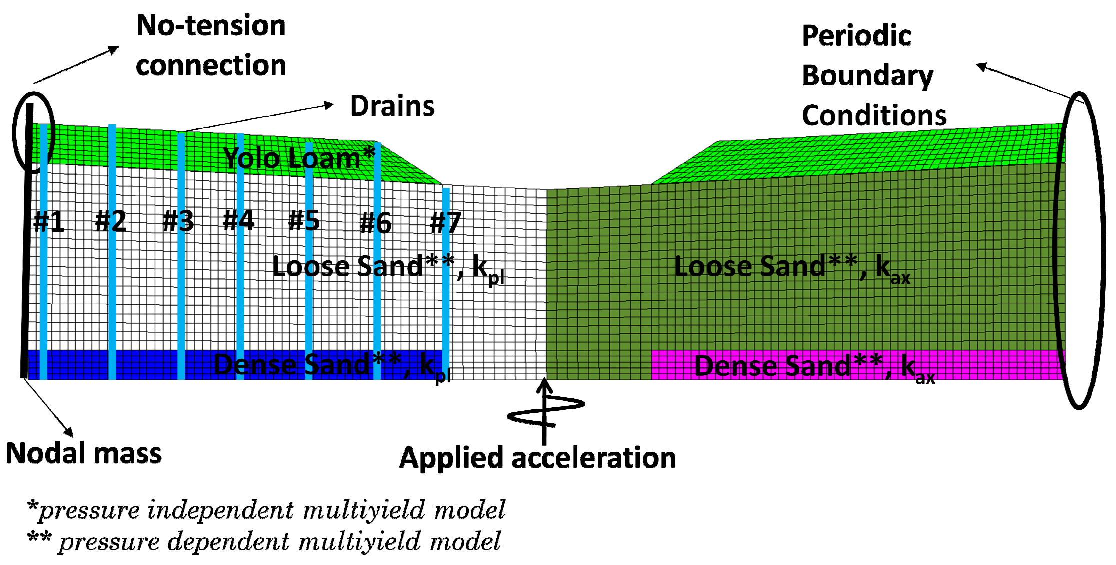


Figure 5-2 FE Model Setup

Table 5-1 Shaking sequence

Spin	Event ID	Centrifuge Acceleration (g)	Motion Amplitude[(1)] (g)	Time	Date
Spin 5	Spin Up	---	---	4:15 PM	3/9/2007
	SSK01_08	15	0.01	4:45 PM	3/9/2007
	SSK01_09	15	0.03	5:06 PM	3/9/2007
	SSK01_10	15	0.07	5:32 PM	3/9/2007
	SSK01_11	15	0.11	6:19 PM	3/9/2007
	SSK01_12	15	0.3	6:45 PM	3/9/2007
	Spin Down	---	---	6:51 PM	3/9/2007

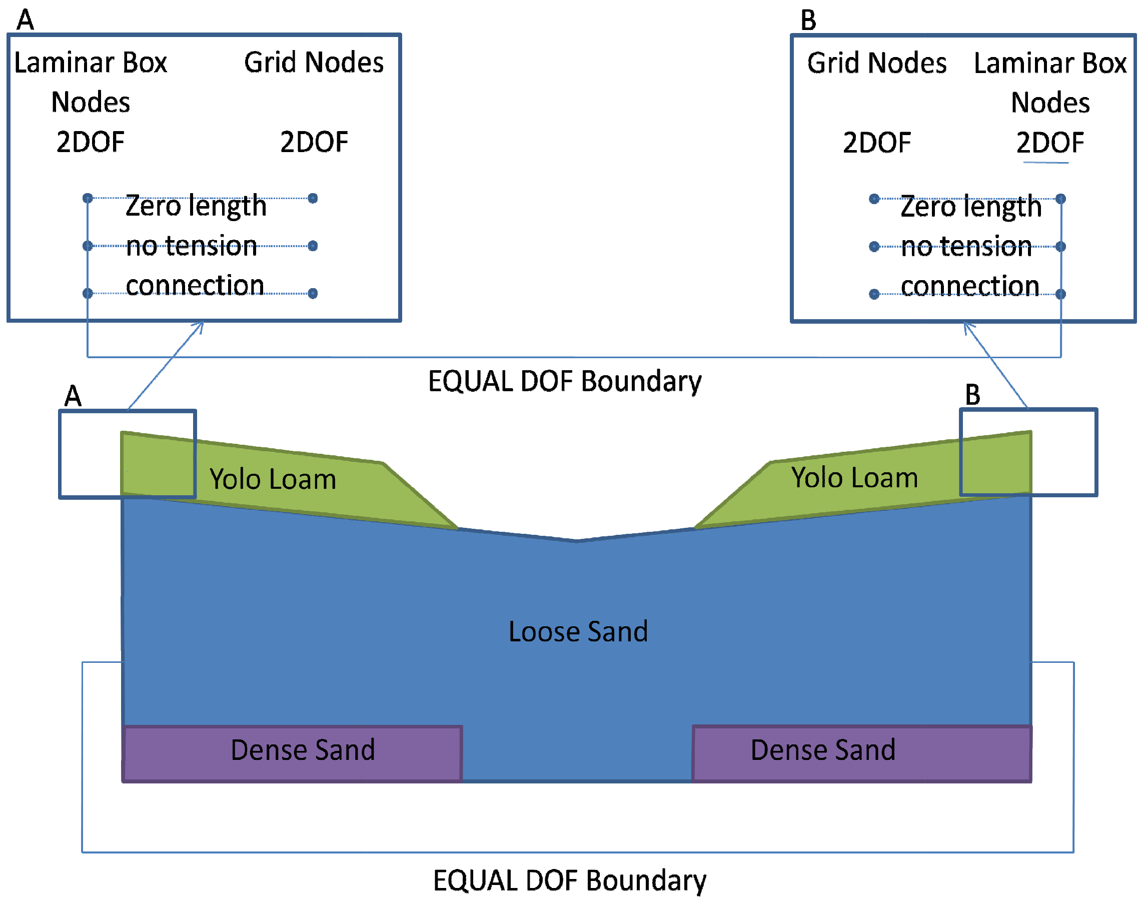


Figure 5-3 Boundary conditions of the FE model

5.1.1 Model parameters

5.1.1.1 Soil parameters

Although there are some uncertainties in the as-built density of the centrifuge model, the current analyses assume that the dense and loose Nevada sand are prepared uniformly with relative density, D_r = 80% and 40%, respectively. Table 5-2 summarizes the model input parameters used to represent these layers using PD-MYS02 model. These parameters are based on prior published calibrations for Nevada sand by Mazzoni et al. (2005). The assumed value of the coefficient of lateral earth pressures is K_0=0.48.

Table 5-2 Input parameters for Nevada sand using PD-MYS02 model

Parameter	Dense sand (D_r = 80%)	Loose sand (D_r =40%)
ρ *(ton/m^3)*	2.07	1.98
G_{ref} *(kPa)*	130000	90000
K_{ref} *(kPa)*	260000	220000
Φ	36.5	32.0
γ_{peak}	0.1	0.1
p_{ref} *(kPa)*	80	80
ψ_{PT}	26.0	26.0
c_1	0.013	0.067
c_3	0.0	0.23
d_1	0.3	0.06
d_3	0.0	0.27

Mazzoni et al. (2005) have also recommended input parameters for the compacted Yolo Loam (Table 5.2) using the PI-MYS model in Opensees. There no little basis for changing these parameters. However, it should be noted that the compacted Yolo loam is partially saturated and appears to undergo volume change during the centrifuge experiment. The initial total unit weight was initially measured as γ_t = 13kN/m^3. However, samples taken after the test found γ_t = 18kN/m^{3}and the height of the Yolo loam shrinks from 1m to approximately 0.8m. This is partly due to compaction during the consolidation phase and the shaking event. The current finite element analyses assume γ_t = 13kN/m^3 and 1m of Yolo loam thickness.

The shear strength of the Yolo loam (c, Table 5-3) is important in the finite element model as it controls the shear resistance along the loam-sand interface (there are no special slide line elements used in the model). The interface shear resistance can be estimated from measurements of relative accelerations above and below the Yolo loam-Nevada sand interface. Figure 5.4 shows the relative acceleration measured during the first phase of shaking in test SSK01. This represents the difference between the absolute acceleration measurements of accelerometers U56 (right above the interface) and U38 (right below the interface).

Using simple mechanics, the maximum shear stress at the interface $\tau_{max} \approx \sigma_v a_{max}$, where σ_v is the overburden pressure. Based on the results in Figure 5-4, $a_{max} \approx 0.5 m/sec^2$. If $\sigma_v \approx 13kPa$, then $\tau_{max} \approx 6.5kPa$. This is quite similar in magnitude to the drained shear resistance on the sand ($\tau = \sigma'_v \tan\phi'$), and can also be represented by assuming that cohesion in the Yolo loam, c = 6.0 kPa.

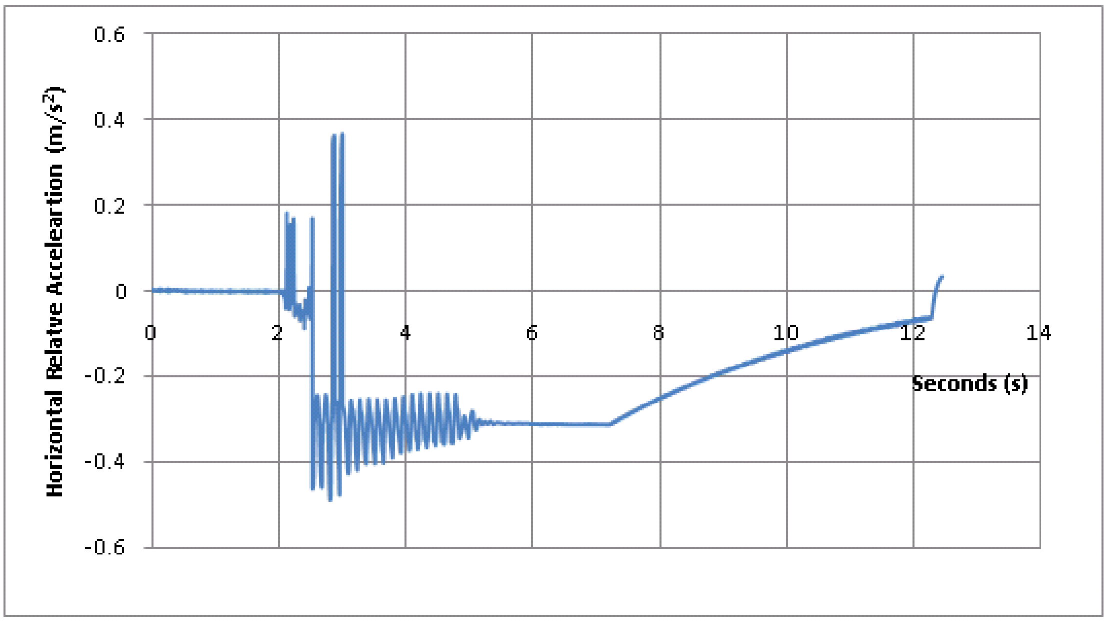


Figure 5-4 Relative acceleration below and above the Loose Sand - Yolo Loam interface for the SSK01-10 phase of shaking

Table 5-3 Yolo Loam model properties for PI-MYS model

Parameter	Input
ρ *(ton/m³)*	1.3
G_{ref} (kPa)	13000
K_{ref} (kPa)	65000
C (kPa)	6.0*
γ_{peak}	0.1

* c = 18kPa was originally recommended by Mazzoni et al. (2005)

The hydraulic conductivity of the Nevada sand was measured at model scale:

$$k^m = 2 \cdot 10^{-5} m/s \tag{5.1}$$

So at the prototype scale the hydraulic conductivity must be scaled by N = 15 in order to match diffusion times:

$$k^p = 3 \cdot 10^{-4} m/s \tag{5.2}$$

On the left side of the models where PV drains are installed, the hydraulic conductivity is scaled in order to match the average degree of consolidation in the plane strain FE model and 3D physical models (after Hird et al., 1992):

$$k_{pl}^P = \frac{2 \cdot 3 \cdot 10^{-4}}{3(\ln(15.5) - \frac{3}{4})} m/s \tag{5.3a}$$

$$k_{pl}^P = 1.0046 \cdot 10^{-4} m/s \tag{5.3b}$$

Hence there is a factor of 3 decrease in the hydraulic conductivity within the sand on the left-hand side of the model.

5.1.1.2 Laminar box parameters

The approximate weight of the FSB3 laminar box used for tests SSK01 is approximately the same as an earlier design (FSB2) whose properties are listed in Table 5.3. These masses are uniformly distributed along the lateral boundaries. This is evenly assigned to the side nodes of the centrifuge model as: (.0132Mgr +.0475Mgr +.0475Mgr +0.09Mgr)/19=0.0104Mgr/side node.

Table 5-4 Design details of container FSB2 for the large centrifuge

Ring location	Material	Section size	Mass of ring (kg)	Area of rubber on lower face (m^2)	Mass of rubber on lower face (kg)
1 (top)	aluminum	2" x 4" channel	13.2	0.258	3.7
2	aluminum	6" x 4" tubing, ¼" wall	47.5	0.439	6.3
3	aluminum	6" x 4" tubing, ¼" wall	47.5	0.538	7.7
4	aluminum	6" x 4" tubing, ½" wall	90	0.538	7.7
5 (bottom)	stainless steel	4" x 4" and 6" x 4" tubing, ¼" wall	113	0.597	8.5

5.1.1.3 Parameters for PV-drains

The input parameters needed for the PV drains are the transmissivity for laminar flow and fully turbulent flow, the axial stiffness, and the storage capacity. The inner diameter of the drain at model scale, D^m = 7mm (and the outer diameter is 9mm). So the effective area that influences the mechanical truss behavior is:

$$A^M = \frac{\pi}{4}(9^2 - 7^2)mm^2 \Rightarrow \tag{5.4a}$$

$$A^M = 25.1328mm^2 \Rightarrow \tag{5.4b}$$

$$A^P = A^M N^2 \Rightarrow \tag{5.5a}$$

$$A^P = 0.005655m^2 \tag{5.5b}$$

The storage capacity corresponds to the volume of the drain that needs to fill up is a cylinder with diameter the inner diameter of the drain. Thus the cross section of the drain needed to estimate the effect of storage capacity is:

$$A^M = \frac{\pi}{4} 7^2 mm^2 \Rightarrow \tag{5.6a}$$

$$A^M = 38.4845 mm^2 \Rightarrow \tag{5.6b}$$

$$A^P = A^M N^2 \Rightarrow \tag{5.7a}$$

$$A^P = 0.0087 m^2 \tag{5.7b}$$

The material for the drains is nylon with an elastic modulus of:

$$E = 3000000 kPa \tag{5.8}$$

Now we need to estimate the transmissivity parameters. We estimate C$_l$ initially for the model scale (μ=10^{-3}Pa·s, for water at 20°C):

$$C_l = \frac{A \cdot D^2}{32 \cdot \mu} \Rightarrow \tag{5.9a}$$

$$C_l^M = \frac{38.48 \cdot 10^{-6} m^2 \cdot 0.007^2 m^2}{32 \cdot 10^{-3} \frac{N}{m^2} s} \Rightarrow \tag{5.9b}$$

$$C_l^M = 5.892 \cdot 10^{-5} \frac{m^6}{kN \cdot s} \tag{5.9c}$$

$$C_l^P = C_l^M N^3 \tag{5.10a}$$

$$C_l^M = 0.1989 \frac{m^6}{kN \cdot s} \tag{5.10b}$$

This is the transmissivity parameter we should use if we were solving the true 3D radial drainage problem in the prototype scale. In reality we solve a plane strain problem, so we need to scale the transmissivity according to Hird et al (1992):

$$C_l^{pl} = \frac{2}{\pi R} C_l^{ax} \tag{5.11a}$$

$$C_l^{pl} = \frac{2}{3.14 \cdot \frac{1.57m}{2}} 0.1989 \frac{m^6}{kN \cdot s} \tag{5.11b}$$

$$C_l^{pl} = 0.1614 \frac{m^5}{kN \cdot s} \tag{5.11c}$$

which is the parameter we are using.

For the fully turbulent drains we introduce a similar type of analysis. We estimate initially C_t for the model scale:

$$C_t = \sqrt{\frac{2 \cdot D \cdot A^2}{\lambda \cdot \rho}} \tag{5.12a}$$

$$C_t^M = \sqrt{\frac{2 \cdot 0.007m \cdot (38.48 \cdot 10^{-6} m^2)^2}{0.017 \cdot 1 \frac{Mgr}{m^3}}} \tag{5.12b}$$

$$C_t^M = 3.492 \cdot 10^{-5} \frac{m^{4.5}}{kN^{0.5}s} \tag{5.12c}$$

$$C_t^P = C_t^M N^{2.5} \tag{5.13a}$$

$$C_t^M = 0.0304 \frac{m^{4.5}}{kN^{0.5}s} \tag{5.13b}$$

We now scale the transmissivity according to Hird et al (1992):

$$C_t^{pl} = \frac{2}{\pi R} C_t^{ax} \sqrt{w} \tag{5.14a}$$

$$C_t^{pl} = \frac{2}{3.14 \cdot \frac{1.57m}{2}} 3.043 \cdot 10^{-5} \frac{m^{4.5}}{kN^{0.5}s} \sqrt{1m} \tag{5.14b}$$

$$C_t^{pl} = 0.0247 \frac{m^4}{kN^{0.5}s} \tag{5.14c}$$

which is the parameter we are using in our analysis.

A summary of results is presented in Table 5-5.

Table 5-5 Drain properties in prototype scale

Mechanical Deformation	
$Cross\ Section$	$0.005655m^2$
E	$3000000kPa$
Storage Capacity	
$Cross\ Section$	$0.0087m^2$
$Maximum\ stored\ height$	$1m$
Drain properties	
C_l^{pl}	$0.1614\dfrac{m^5}{kN}$
C_t^{pl}	$0.0247\dfrac{m^4}{kN^{0.5}s}$

5.2 Base case analysis

This section presents detailed results from a base case finite element analysis of centrifuge model test, SSK01, and compares results with experimental measurements reported by Kamai et al. (2008). The base case analysis assumes laminar flow in the PV drains and includes the storage effect in each row of drains.

Figure 5-5 shows typical results for flow in one row of drains (#2, see Figure 5-2) during the first cyclic loading event (10 cycles of loading with a_{max} = 0.69m/s^2 with period T=0.5s, over a 13s period). The drain elements enable calculation of the flow rate (Figure 5-5a) and fluid volume discharged in each row of drains (Figure 5-5b). The actual flow rates in the drain (Figure 5-5a) exceed the limit for laminar flow at prototype scale but are well within the laminar range at model scale. The discharge volume increases approximately linearly during the shaking event (Figure 5-5b). Figure 5-5c compares the displacement computed directly at the top of the drain with an indirect estimate based on fluid volume discharged by the drain. The indirect calculation generates higher settlements and reflects other sources of ground movements including cyclic shear-induced volume changes in the sand and fluid inflow from the untreated side of the centrifuge model.

5.2.1 Predicted Excess Pore Pressures

In Figure 5-6 to Figure 5-8 compare the predicted and measured pore pressures at six points within the loose Nevada sand in three harmonic shaking events of increased intensity; a_{max} = 0.69, 1.07 and 2.94 m/sec^2). The points A, B, C are located within the PV drain array; while D, E and F are at similar locations in the untreated side of the model. It is readily apparent from the measured data that the PV drains are effective in reducing excess pore pressures generated within the sand (compare time series for A vs D at the top, B vs E in the middle or C vs F towards the base of the sand).

The numerical analyses provide very good predictions of the measured excess pore pressures in the middle and lower parts of the untreated sand (points E and F) during shaking at all three levels of shaking. The analyses underestimate the pore pressures at the top of the untreated sand (point D) where liquefaction[5] is measured at all three levels of shaking. These results suggest limitations of the constitutive model (and/or the selected input parameters[6]) for reproducing the onset of liquefaction in the loose Nevada sand. The numerical analyses also predict much more rapid dissipation of excess pore pressures after cessation of shaking (observed most clearly at E and F, Figs. 5-7, 5-8). This latter effect is related to the consolidation coefficient in

[5] The data at D show $u \approx 20$kPa. This is approximately equal to the overburden pressure, implying that $\sigma'_v \approx 0$ for all three events. It should also be noted that the pressure transducer at D also sank significantly during these events (see Fig. 4.5a).

[6] One possible problem are deviations between the actual sand density in the centrifuge model and the relative density assumed for the PD-MYS02 model parameters.

the sand and can reflect an overestimate of the hydraulic conductivity and/or the stiffness properties of the soil skeleton (linked to input parameters for the loose Nevada sand in the PD-MYS02 model).

On the 'treated' side of the model, direct comparison of the computed and measured pore pressures (points A, B and C) should be interpreted with caution. The parameters assumed in the plane strain finite element model aim to match the average degree of consolidation in the sand surrounding the drains, but do not reproduce accurately the spatial variation of excess pore pressures. The finite element model appears to describe quite well the pore pressure generated at locations B and C at the end of shaking (for all three events), but the analyses seem to lag the measured development during the cyclic loading.

The analysis predicts minimal pore pressure development at point A (top of sand) compared with the measured data. This suggests limitations of the selected input parameters of the soil model, or misleading measurements due to sinking of the pore pressure transducers close to the soil surface.

5.2.2 Accelerations

Figures 5-9, 5-10, and 5-11 shows a comparable set of analyses and measurements for the horizontal acceleration at points A-F in the three SSK01 shaking events. On the untreated side, the numerical analysis grossly overestimates the accelerations in the upper part of the sand layer. For the moderate loading event (a_{max} = 1.07m/s^2, Fig. 5-10) the data show de-amplification at point D while for higher intensity loading (a_{max} = 2.94m/s^2, Fig. 5-11) de-amplification occurs at both D and E (progression of a liquefaction front from the top to the middle of the sand layer).

These results reflect constitutive model limitations in simulating the onset of liquefaction in the model.

For the treated side of the model, the analyses are generally in reasonable agreement with the measured acceleration data at C and B, but a significant underestimate of peak accelerations measured at the top of the sand (point A), especially at higher levels of shaking.

5.2.3 Displacements

Figures 5-12, 5-13, and 5-14 summarize the computed and measured horizontal deformations at a series of 6 points along the surface of the Yolo loam cap, for the same three shaking events. The numerical analyses generally underestimates the lateral movements on the untreated side of the model (points D, E, F) but is in good agreement with the smaller movements measured above the PV drains. Unfortunately there are no continuous deformation-time data within the soil mass. Figures 5-16, 5-17, and 5-18 show the numerical simulations of lateral deformations along two vertical sections (A - treated and B - untreated) during each of the shaking events. For the untreated side, these figures show that lateral spreading within the loose sand is partially constrained by the overlying Yolo loam cap. This is contrary to the centrifuge data (Fig. 4-7) where there is a clear displacement discontinuity at the loam-sand interface. This discrepancy again reflects the limitations of the numerical analyses in replicating the observed liquefaction event. There is much smaller lateral spreading on the treated side of the model, confirming the efficacy of the PV drains in mitigating effects of liquefaction.

Figures 5-18, 5-19 and 5-20 show computed and measured vertical deformations at 6 points along the surface of the Yolo Loam. The finite element analyses consistently underestimate the surface settlements on the treated side (points A, B and C) at all three levels of shaking. The comparisons on the untreated side vary widely from large underprediction at D (after low intensity shaking, Fig. 5-18) to significant overprediction at D after more intense shaking (Fig. 5-20). This behavior is difficult to explain from comparisons at discrete spatial points but become more apparent when considering the whole field of soil deformations (end of test), Figure 5-21. Here it can be seen that the numerical predictions relate to a large rotational mechanism within the untreated side of the model. This behavior appears to exaggerate the measured vertical surface displacements (D, E, F; Fig. 5-20) and may be attributed, in part, to variations in the gravitational field across the model. In a typical centrifuge test, the effect of the non-uniformity of the gravitational field tends to create heave on the sides and settlement on the center of the model. Due to this effect, simulated settlement at points D and E (close to the sides of the model) is more than the measured one, and at point F (in between the sides and the middle of the model) the simulated settlements compare much better to the measured ones.

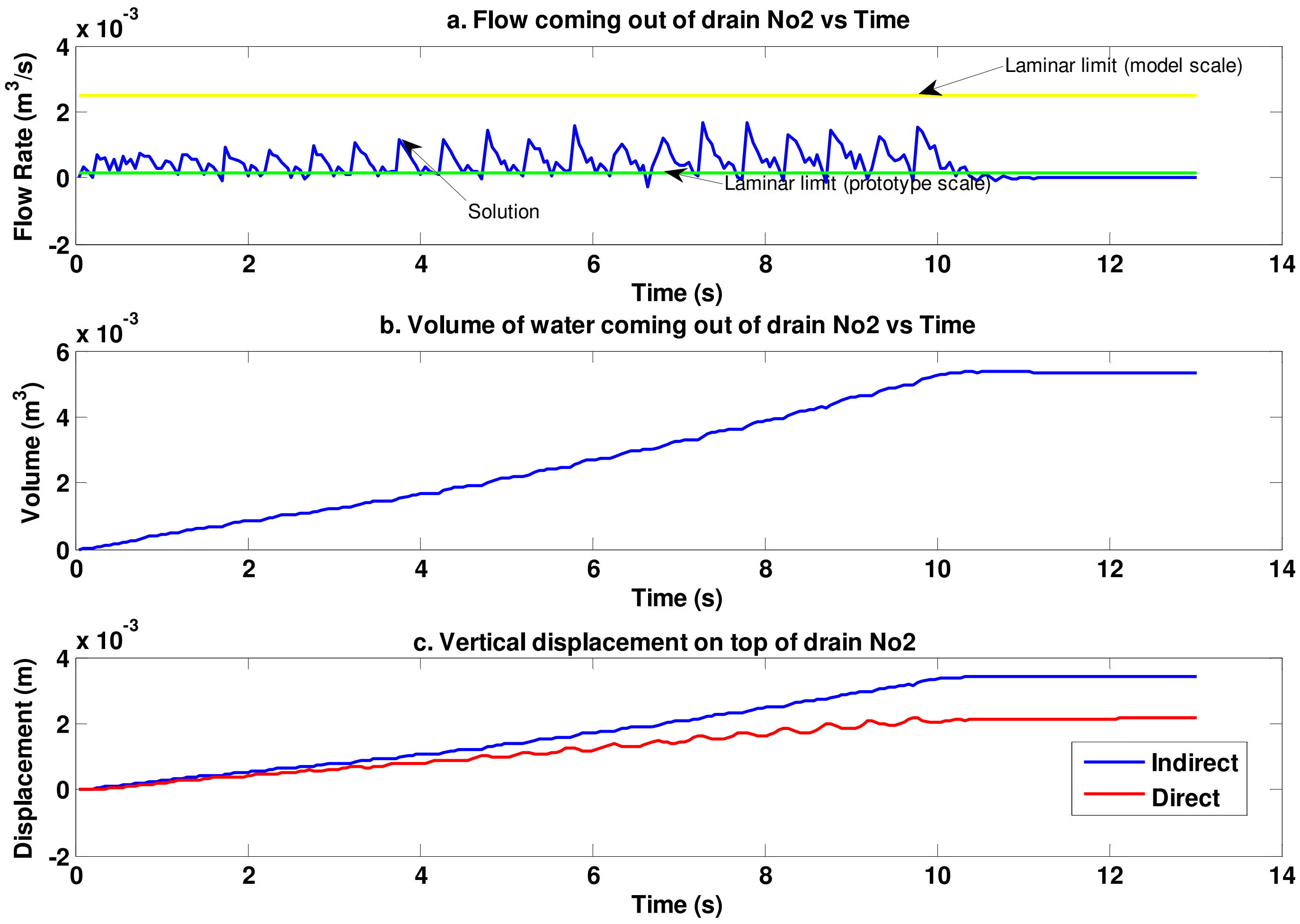


Figure 5-5 Flow and volume of water coming out of drain No2, and comparison of directly and indirectly predicted displacements on the top of the drain during the first shaking event (a_{max}=0.687m/s^2)

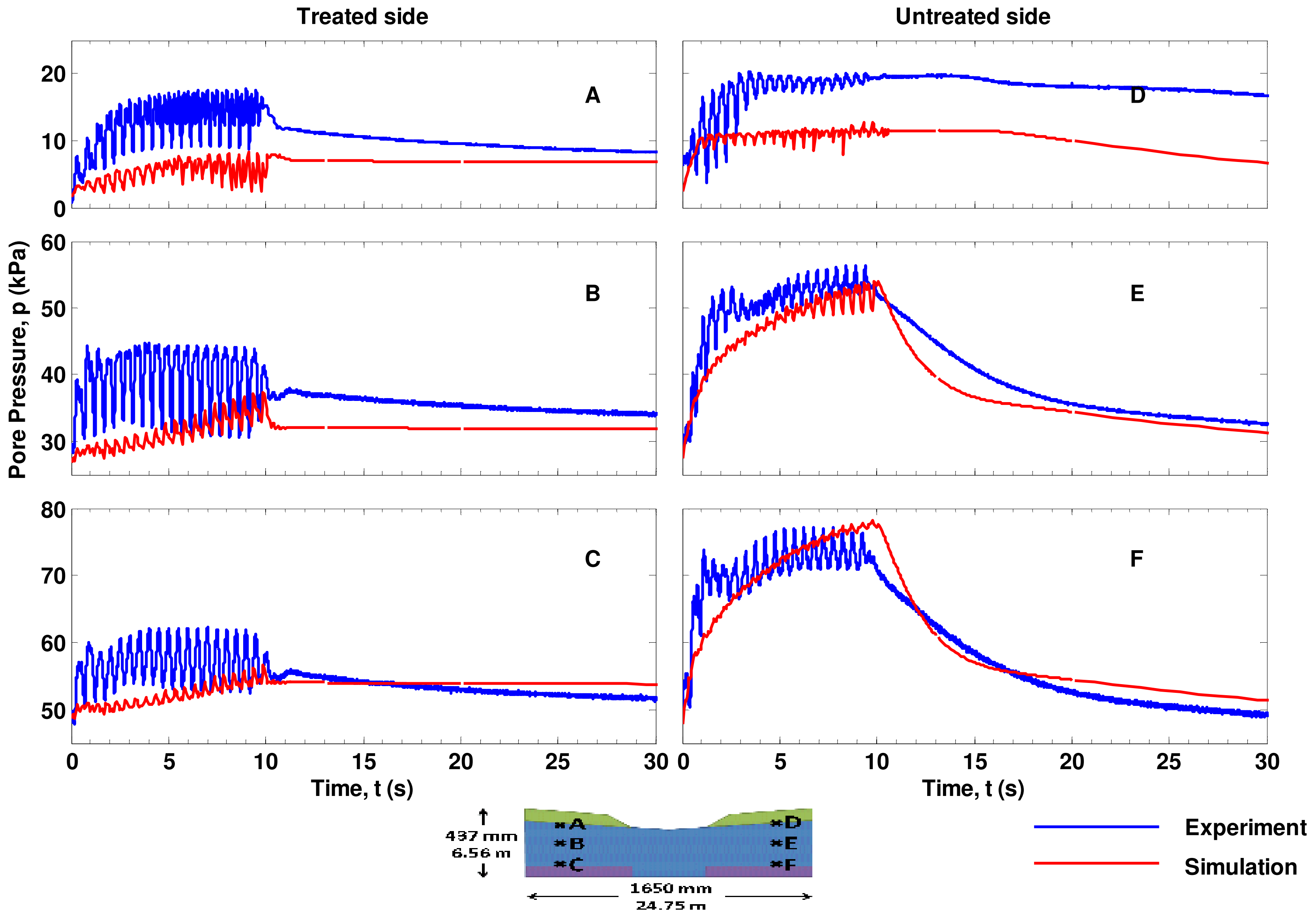


Figure 5-6 Comparison of measured and simulated pore pressures for the first phase of shaking (a_{max}=0.687m/s^2)

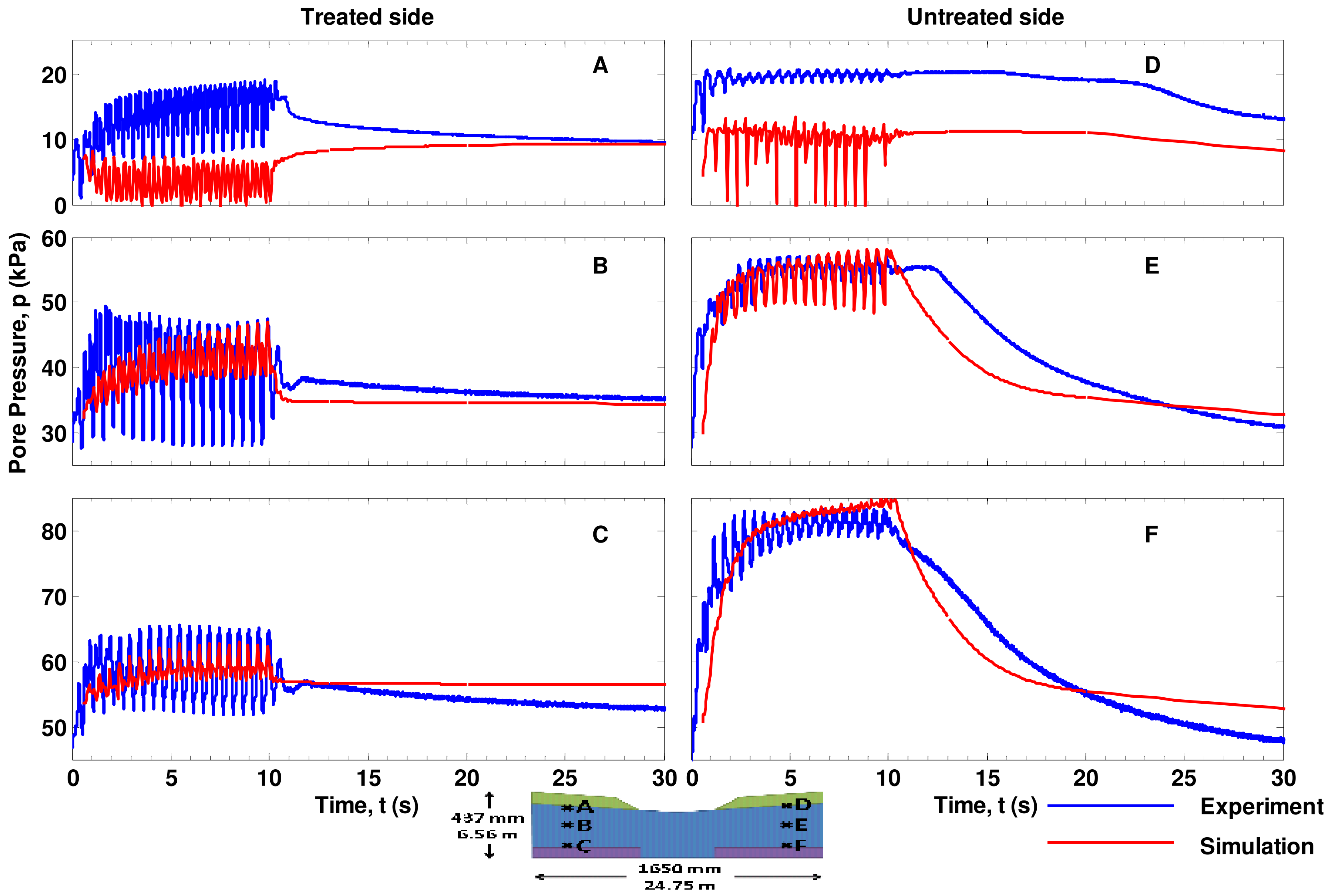


Figure 5-7 Comparison of measured and simulated pore pressures for the second phase of shaking (a_{max}=1.071m/s^2)

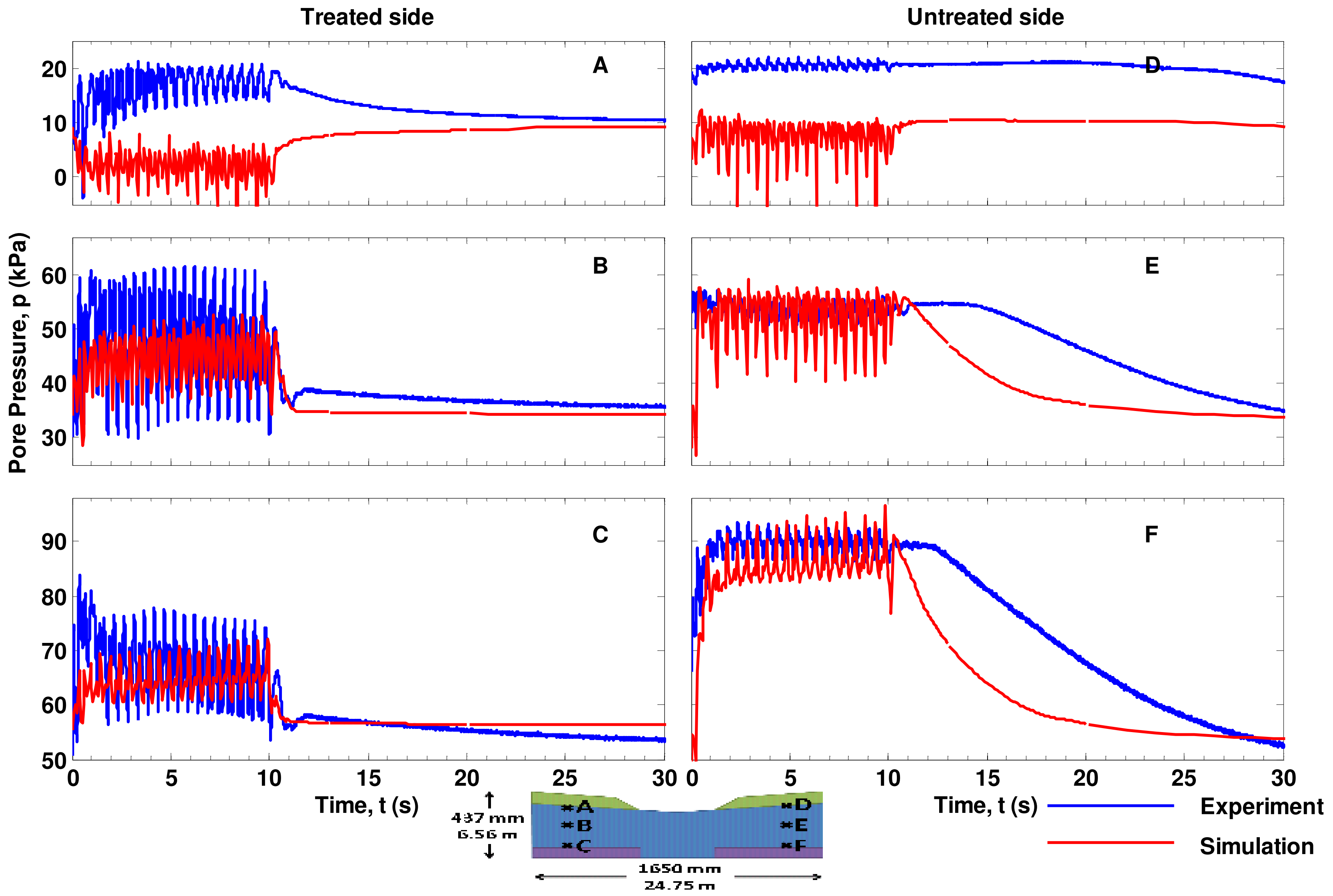


Figure 5-8 Comparison of measured and simulated pore pressures for the third phase of shaking (a_{max}=2.943m/s^2)

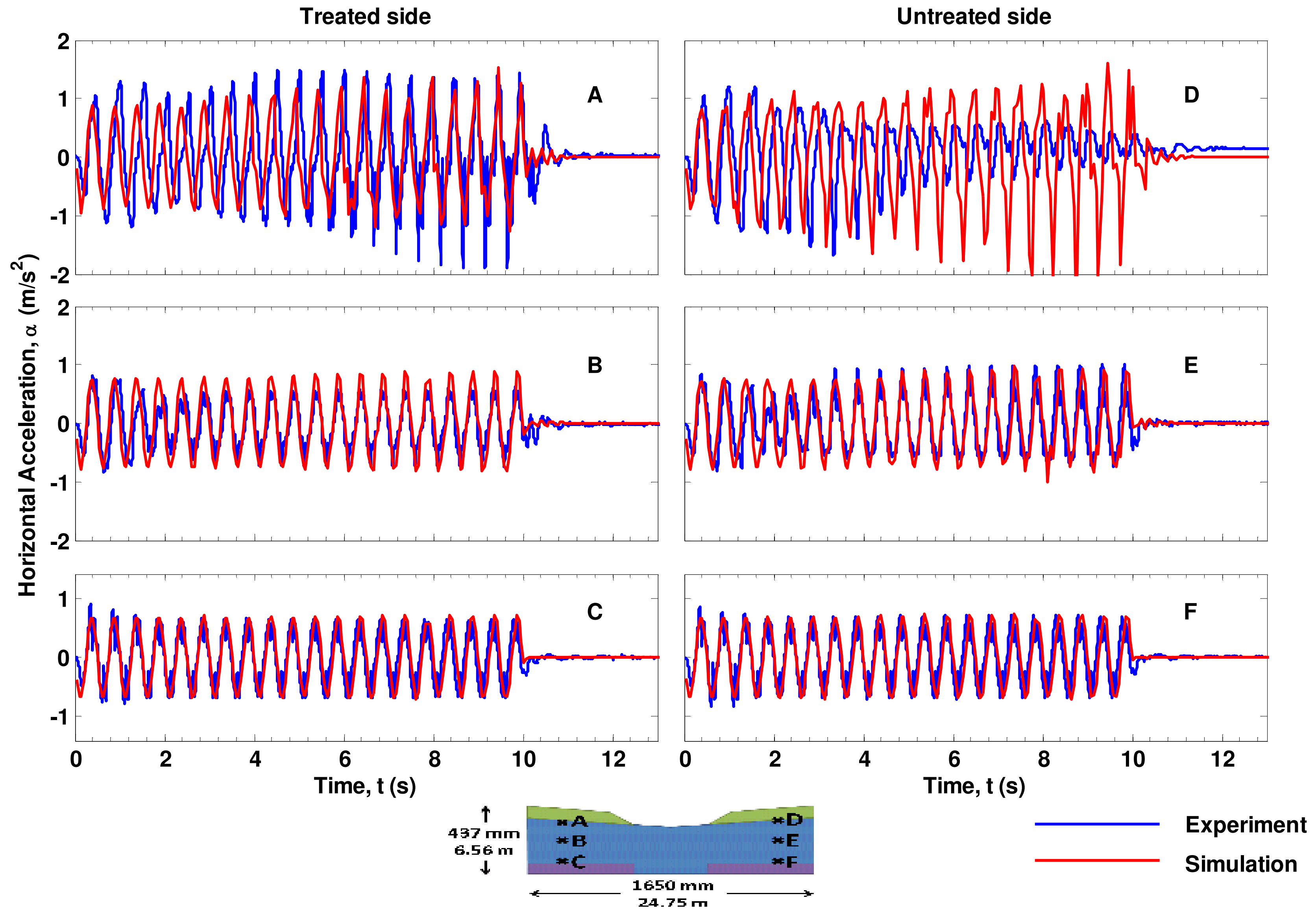


Figure 5-9 Comparison of measured and simulated horizontal accelerations for the first phase of shaking (a_{max}=0.687m/s^2)

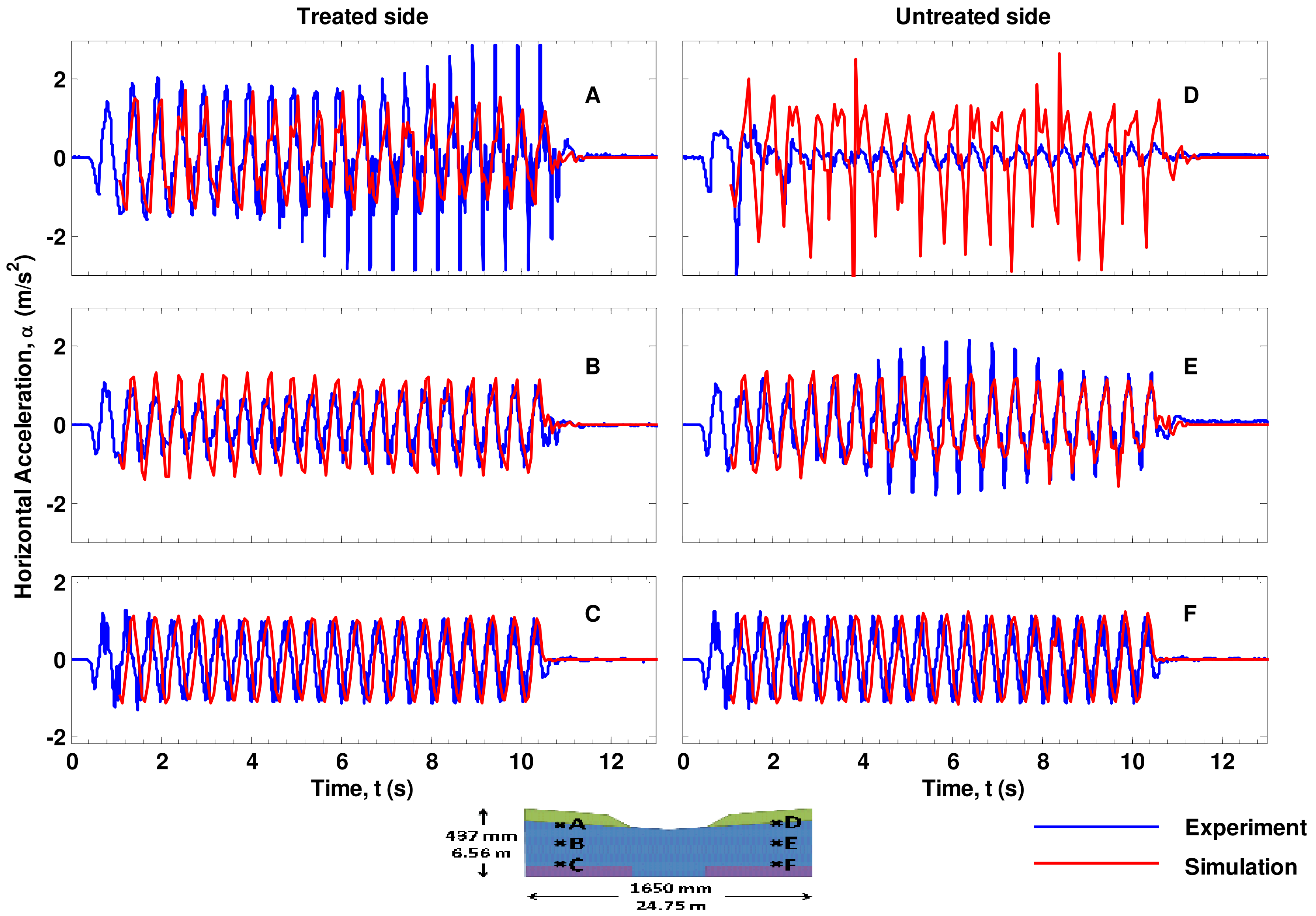


Figure 5-10 Comparison of measured and simulated horizontal accelerations for the second phase of shaking (a_{max}=1.071m/s^2)

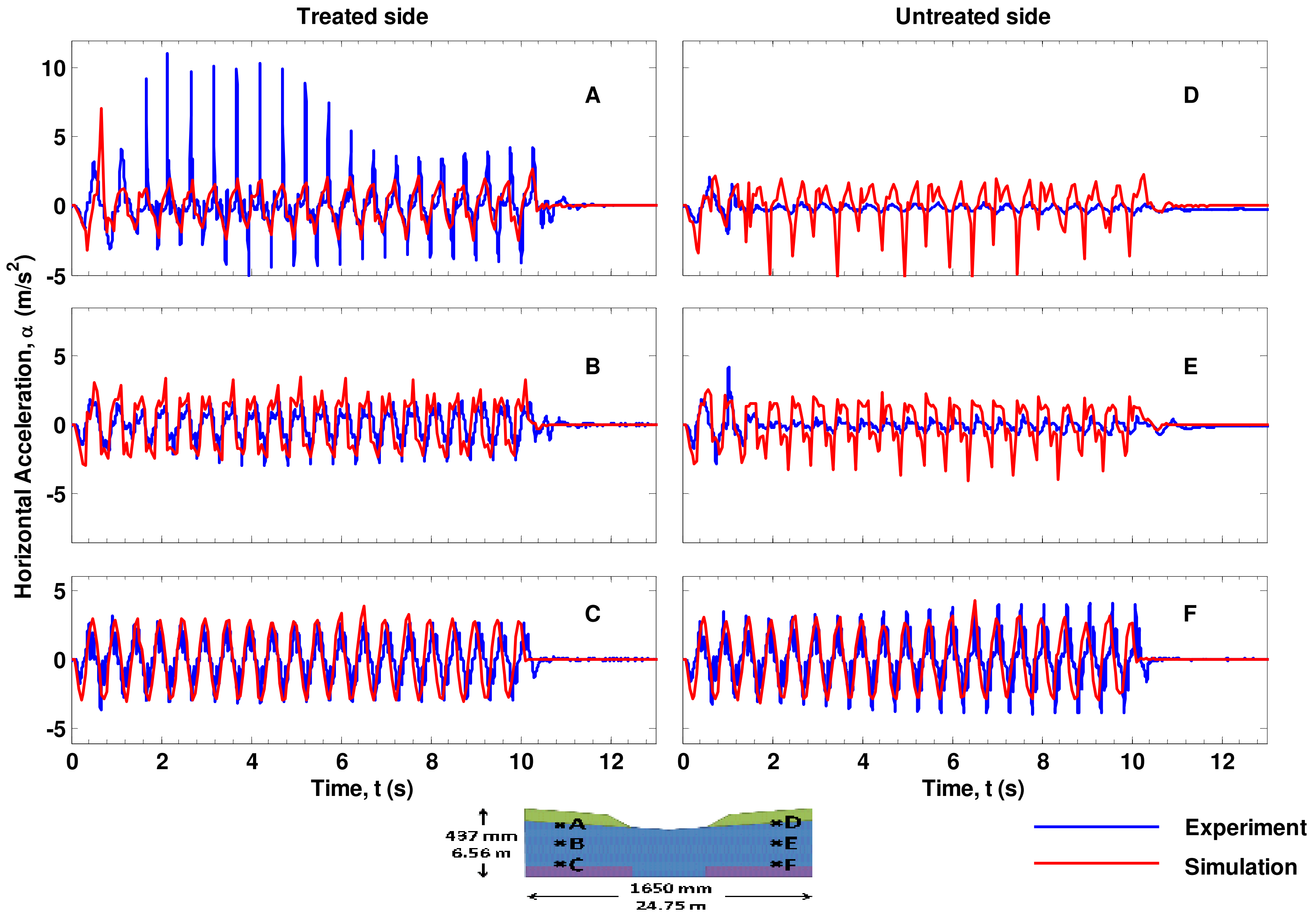


Figure 5-11 Comparison of measured and simulated horizontal accelerations for the third phase of shaking (a_{max}=2.943m/s^2)

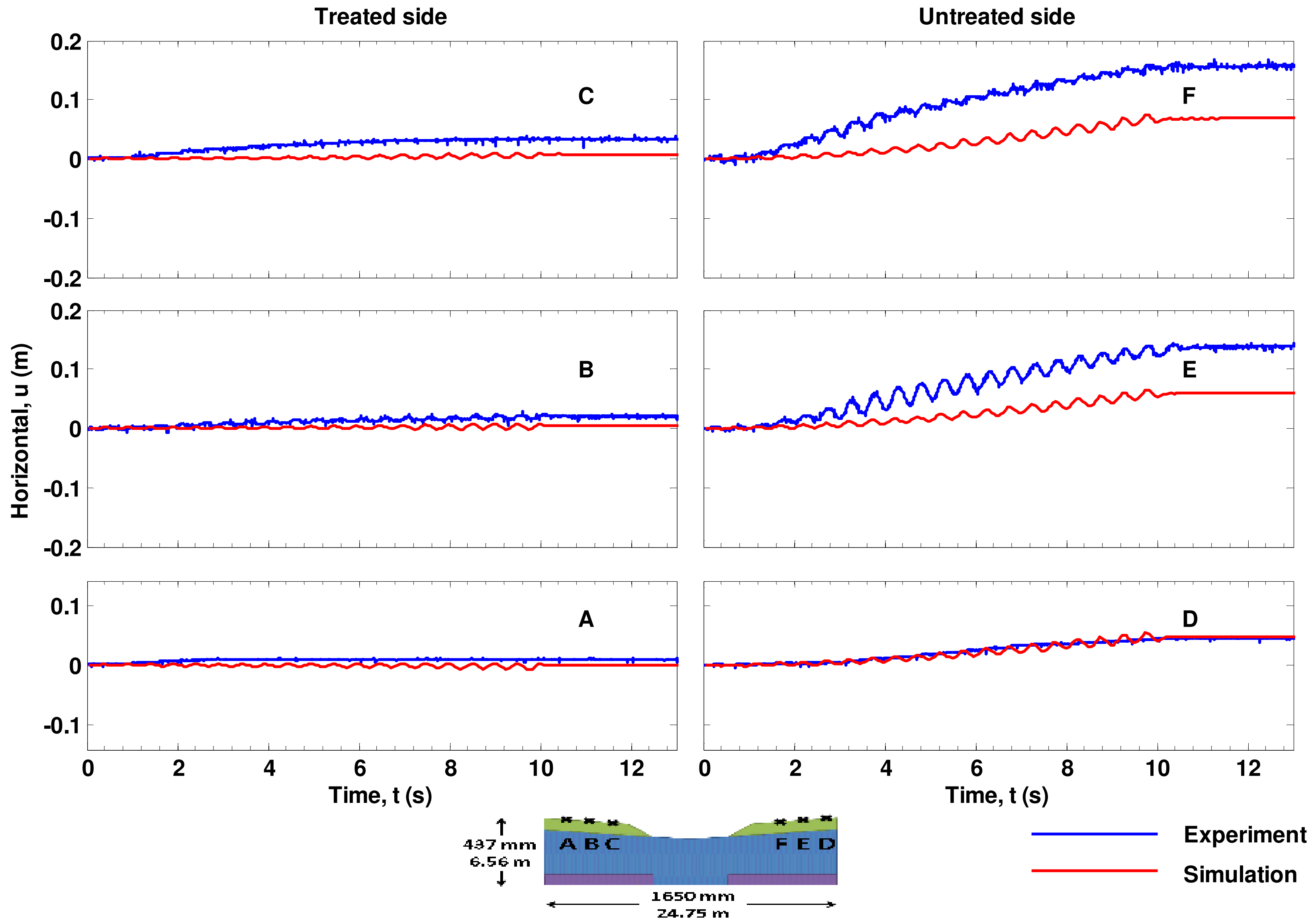


Figure 5-12 Comparison of measured and simulated horizontal displacements for the first phase of shaking (a_{max}=0.687m/s^2)

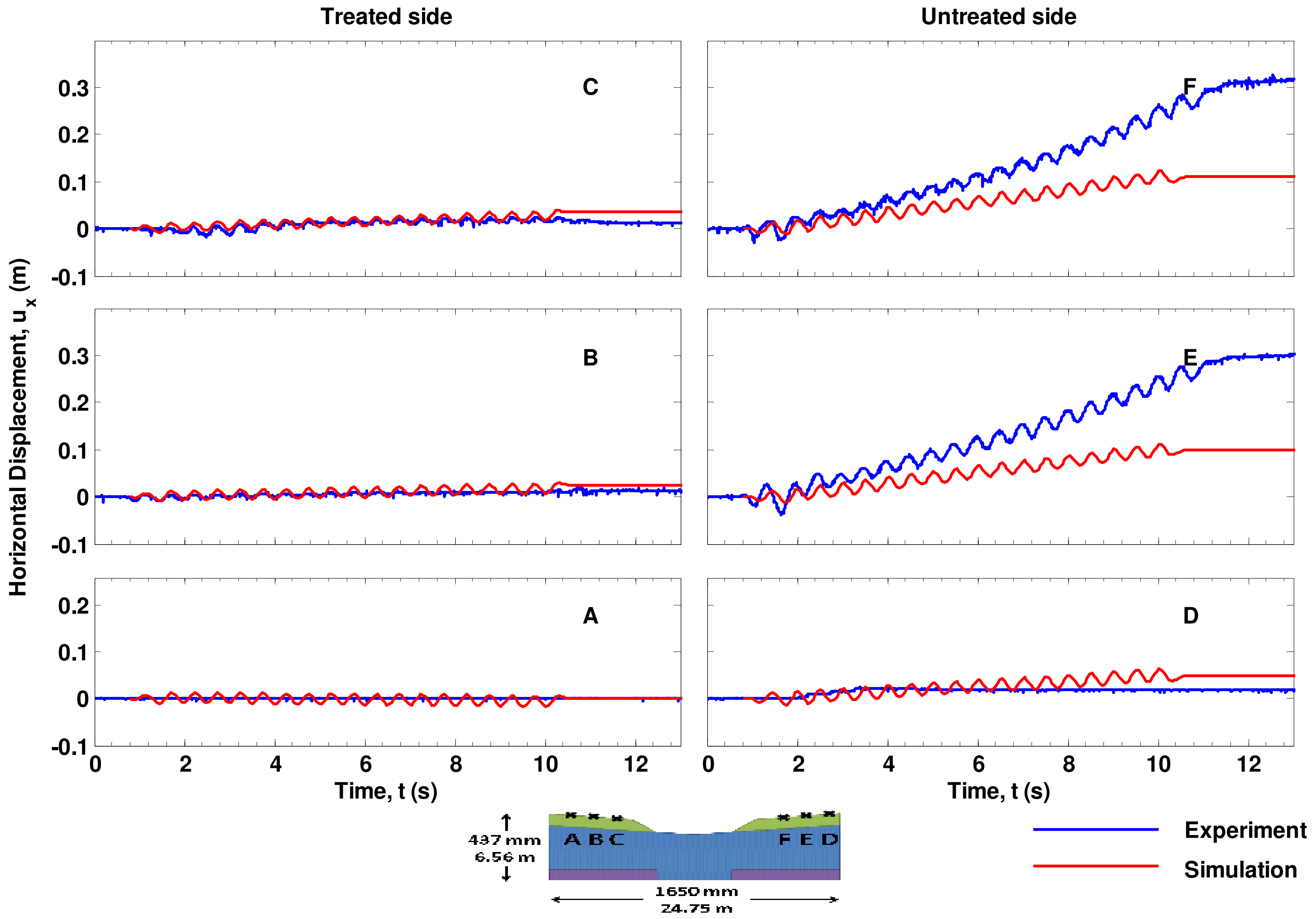


Figure 5-13 Comparison of measured and simulated horizontal displacements for the second phase of shaking (a_{max}=1.071m/s^2)

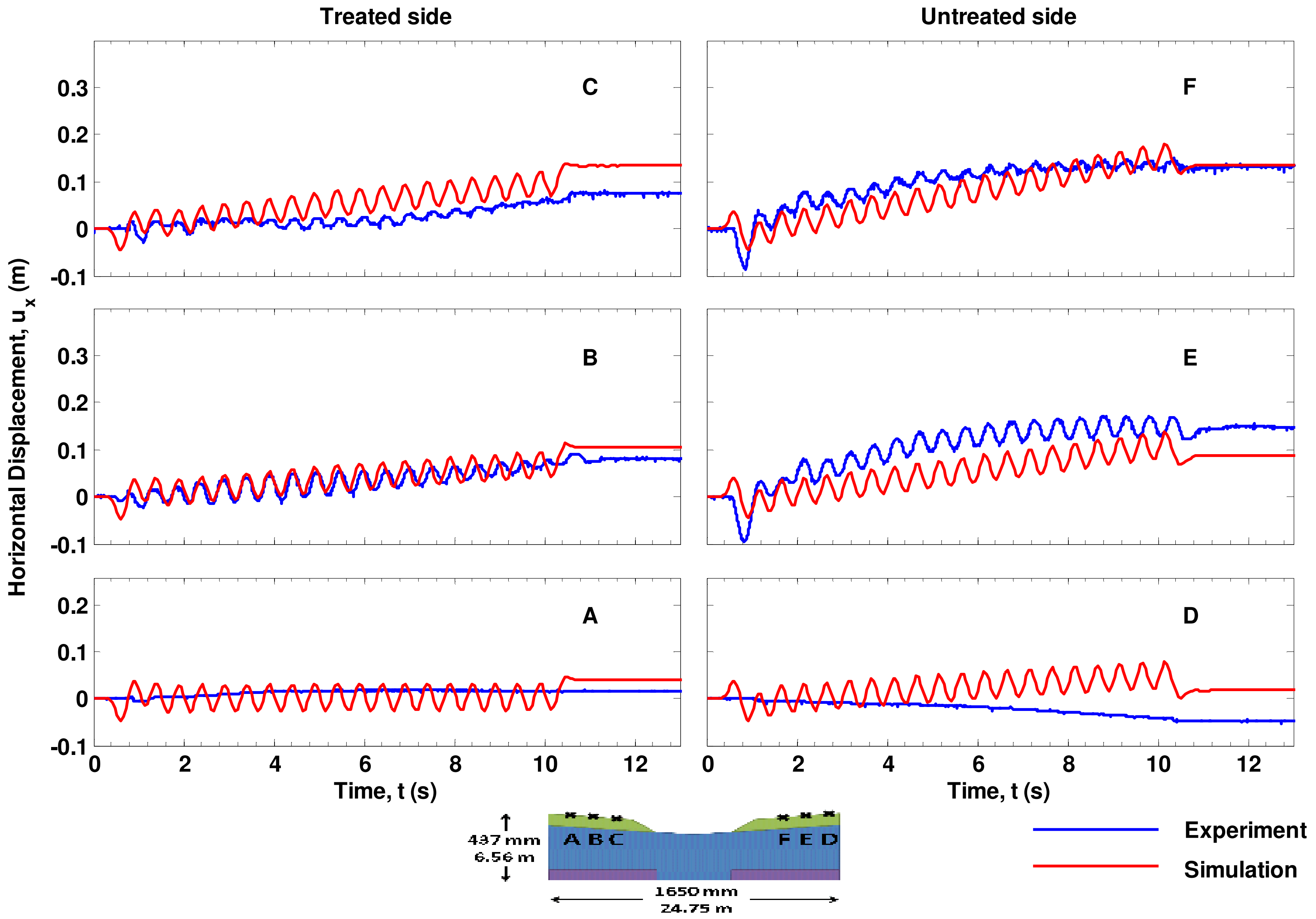


Figure 5-14 Comparison of measured and simulated horizontal displacements for the third phase of shaking (a_{max}=2.943m/s^2)

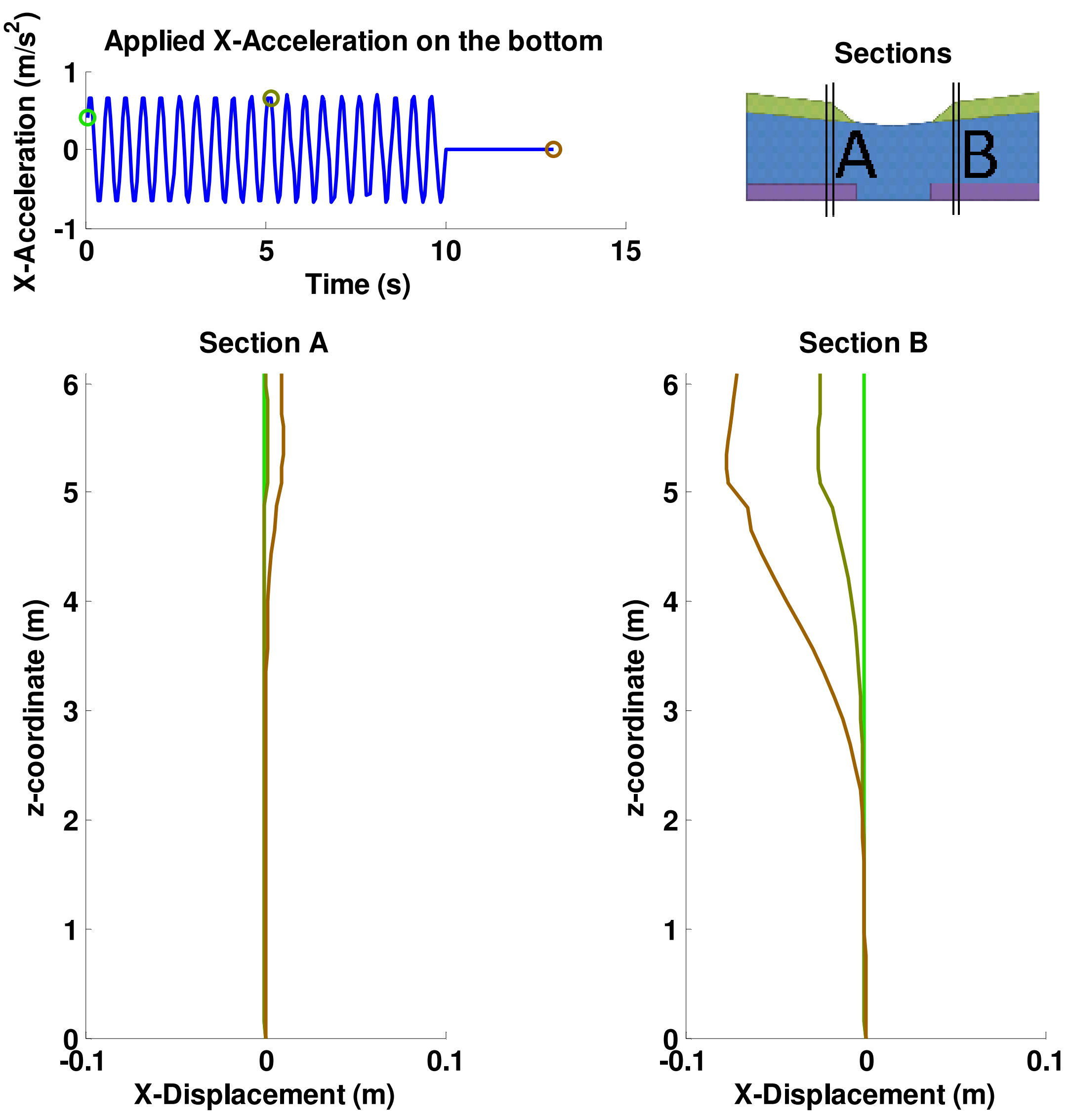


Figure 5-15 Horizontal displacements profiles during the first phase of shaking (a_{max}=0.687m/s^2)

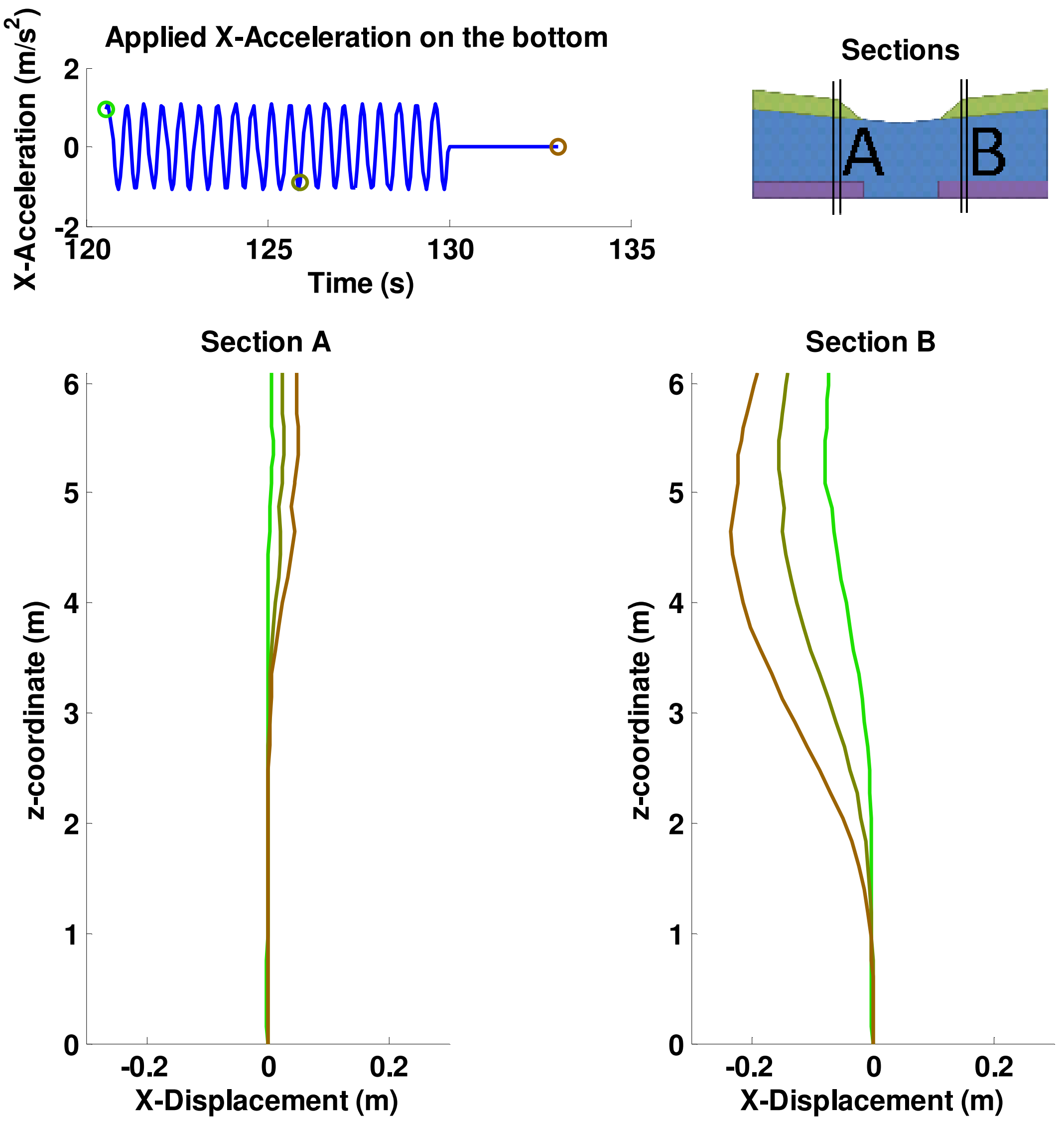


Figure 5-16 Horizontal displacements profiles during the second phase of shaking (a_{max}=1.071m/s^2)

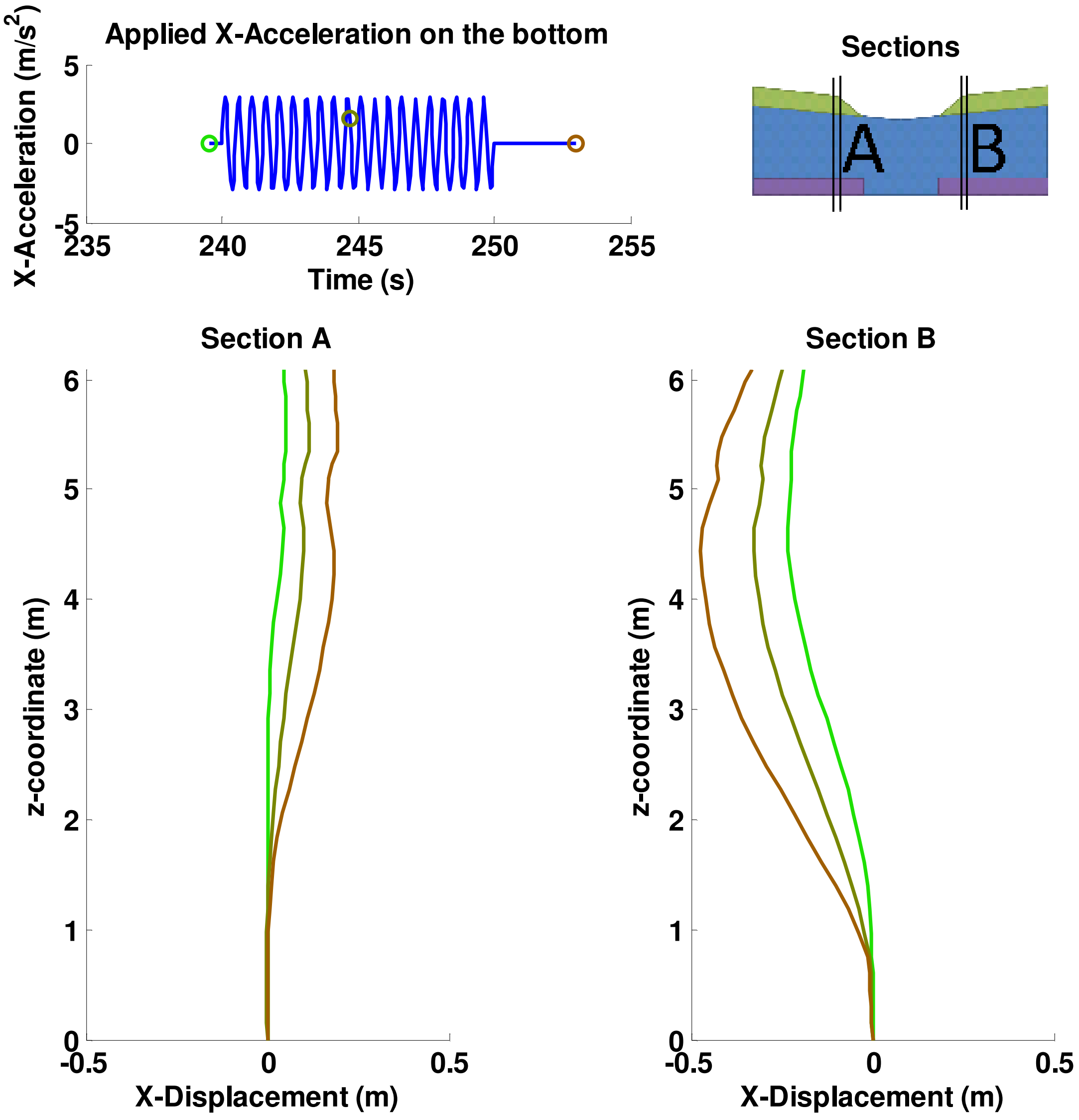


Figure 5-17 Horizontal displacements profiles during the third phase of shaking (a_{max}=2.943m/s^2)

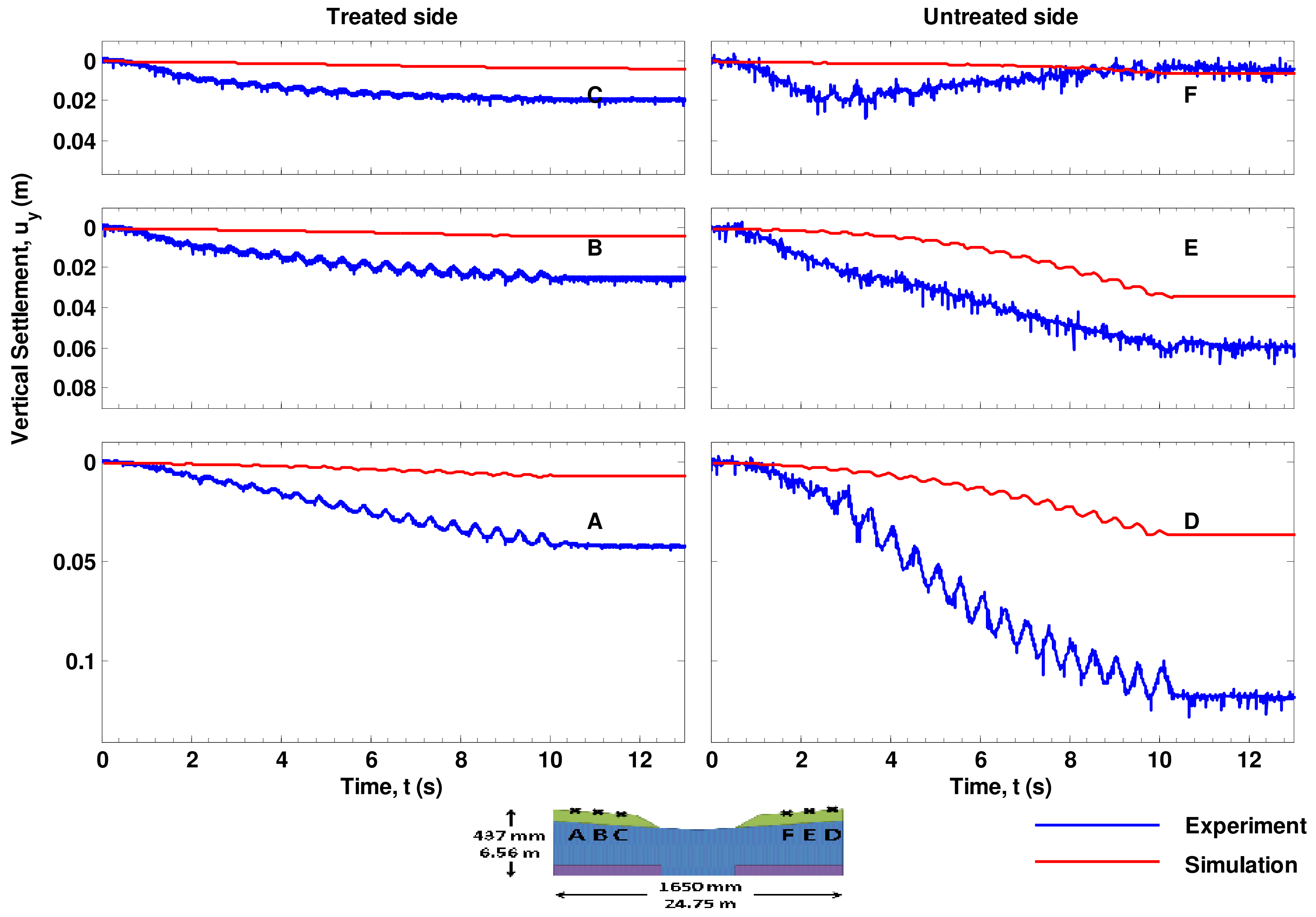


Figure 5-18 Comparison of measured and simulated settlements for the first phase of shaking (a_{max}=0.687m/s^2)

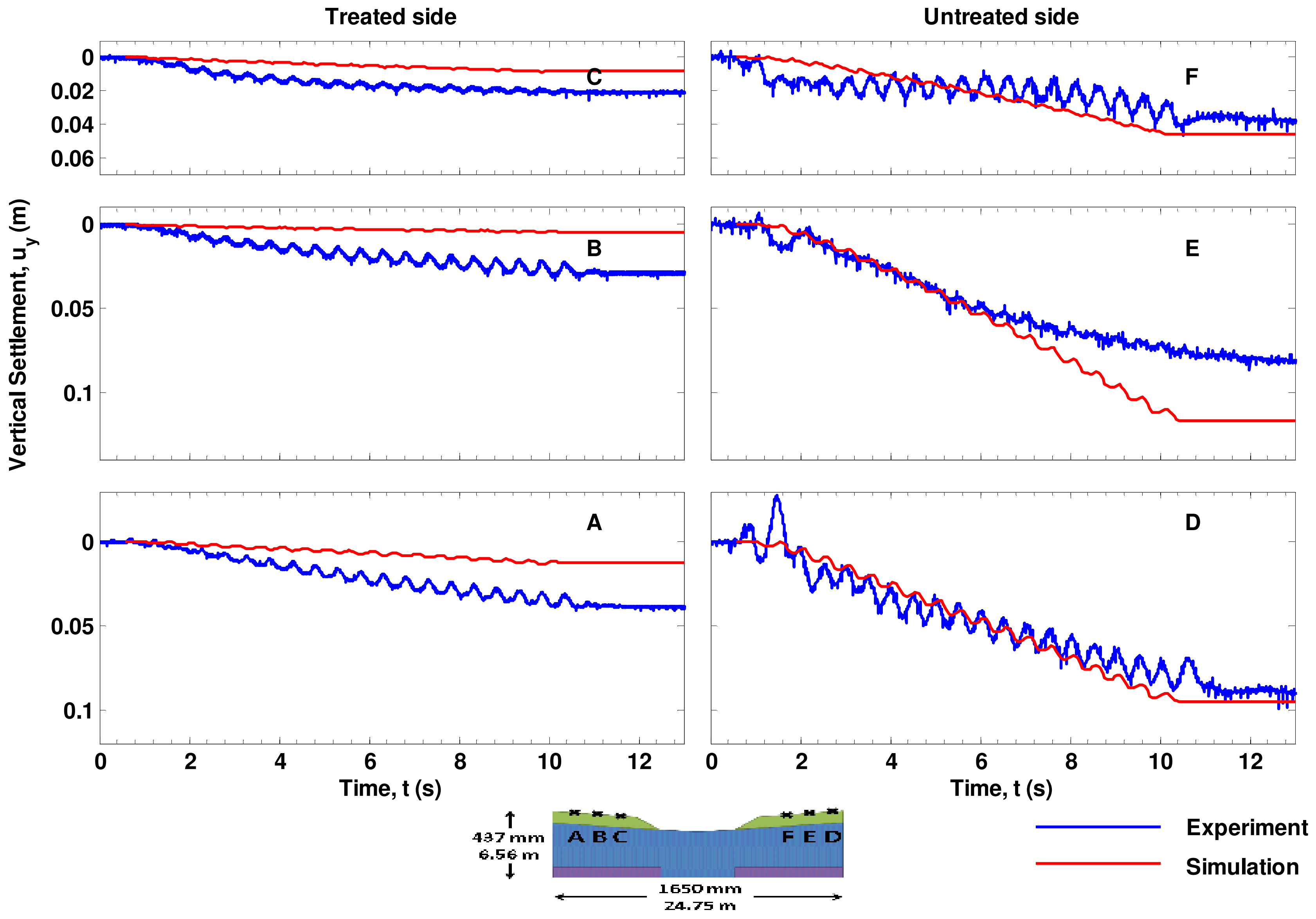


Figure 5-19 Comparison of measured and simulated settlements for the second phase of shaking (a_{max}=1.071m/s^2)

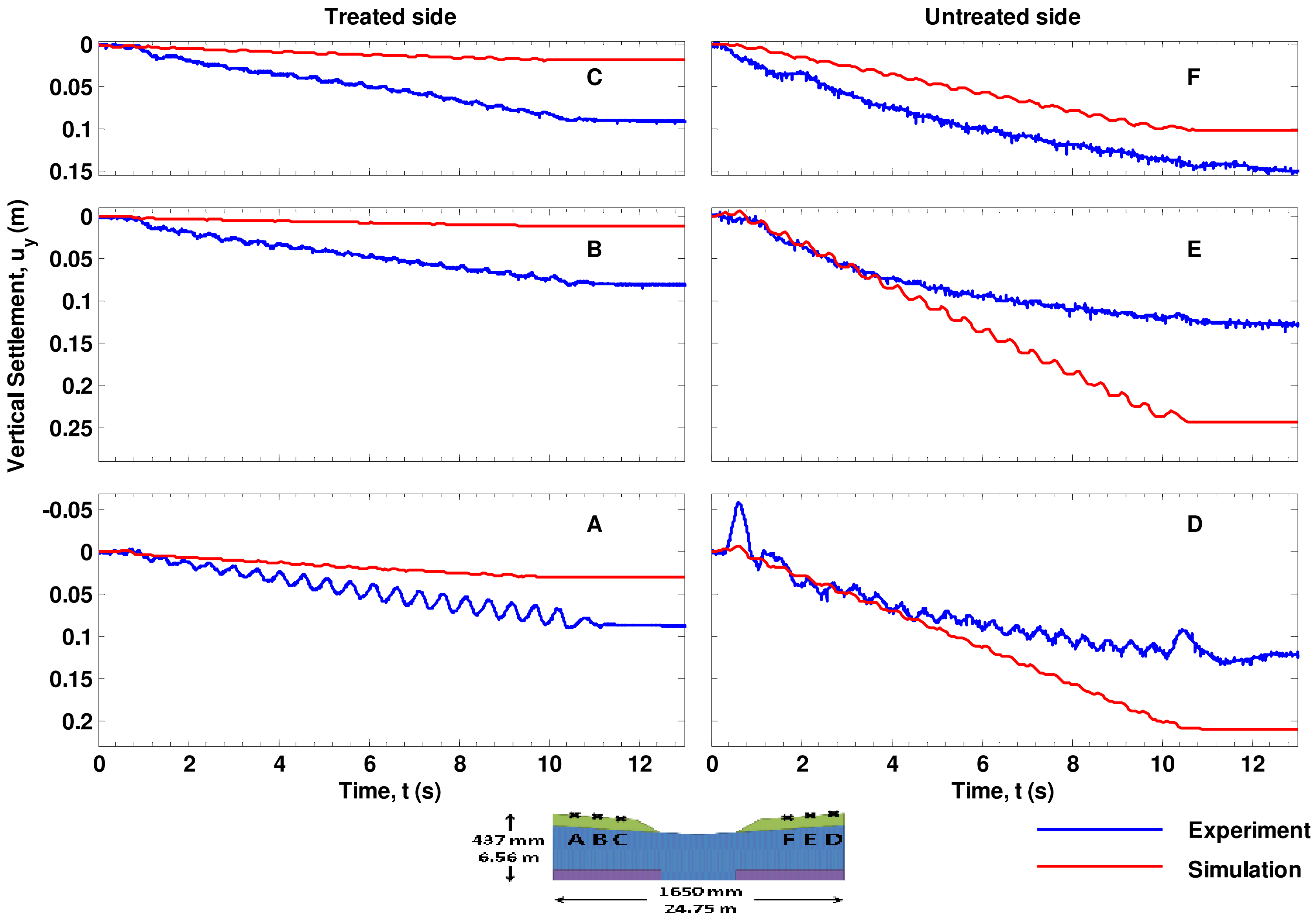


Figure 5-20 Comparison of measured and simulated settlements for the third phase of shaking (a_{max}=2.943m/s^2)

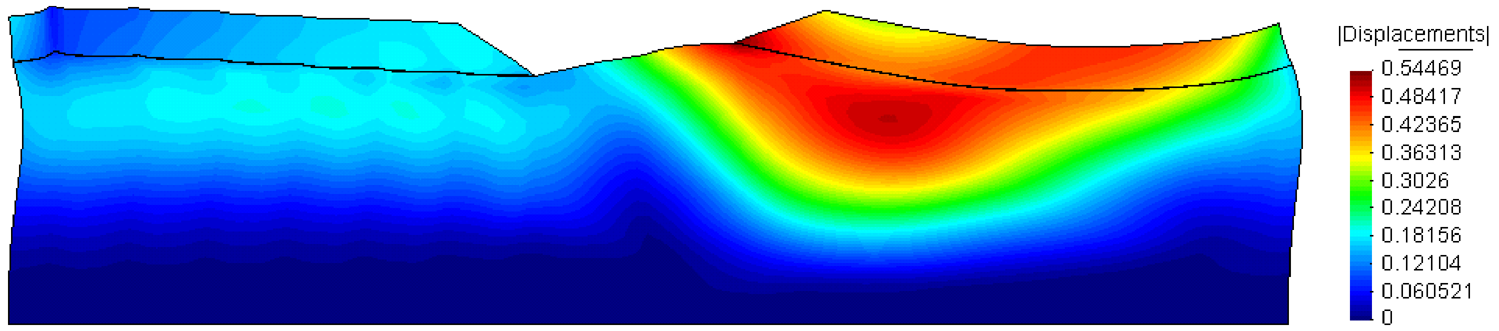


Figure 5-21 Final predicted deformed shape

5.3 Effect of different approximations in PV-drains simulations

5.3.1 Drain resistance

Figure 5-22 compares the pore pressures at two points in the middle of the loose Nevada sand layer (A - treated, B - untreated) for the base case analysis (laminar PV drains) with computations assuming perfect drains. The results at A are practically identical for all three loading events. This result reflects the high transmissivity of the PV drains used in the centrifuge model.

5.3.2 Drain stiffness

In this section we compare an analysis with laminar drains including the drain stiffness to one ignoring the drain stiffness. Figure 5-23 compares the pore pressures for the base case analysis with computations assuming that the drains are infinitely compressible; it is shown that for the second phase of shaking the predicted pore pressures are higher when the drain stifness effect is included, whereas at the third phase they are lower. This is attributed to two competing effects: the drains do not allow the soil to settle, reducing the excess pore pressure, whereas they impede the soil from undergoing large shear deformations, after the dilation angle, incresing the excess pore pressure. These effects do not reflect clearly on the predicted accelerations (Figure 5-24), but are better illustrated in the predicted horizontal displacements shown in Figure 5-25. It should be noted that in order to model realistically the effect of the axial drain stiffness the drain should be connected to the soil grid with frictional elements, something that has not been considered in these analyses.

5.3.3 Drain storage capacity

In this section we compare simulations with drains, with and without storage capacity. A comparison using laminar drains is presented in Figure 5-26 to Figure 5-28, where the pore pressures, the horizontal accelerations, and the horizontal displacements are plotted. When an analysis does not consider the effect of storage capacity, the water pressures in the drains at the clay-sand interface are assumed to stay constant. In an analysis considering the storage effect, the pore pressure conditions at the interface changes, as the water level inside the drain increases up the top of the clay layer. Taking into account the effect of storage capacity, increases the simulated excess pore pressures, thus the predicted permanent displacements on top of the treated side.

5.3.4 Drain turbulence

A comparison of the predicted pore pressures, under the assumption that the flow in the drains is fully turbulent vs. the assumption that the flow in the drains is laminar, is plotted in Figure 5-29. Due to the significant transmissivity of the drains and the large permeability of the soil used in the centrifuge model, we cannot see discrepancies between the results using laminar drains and fully turbulent drains.

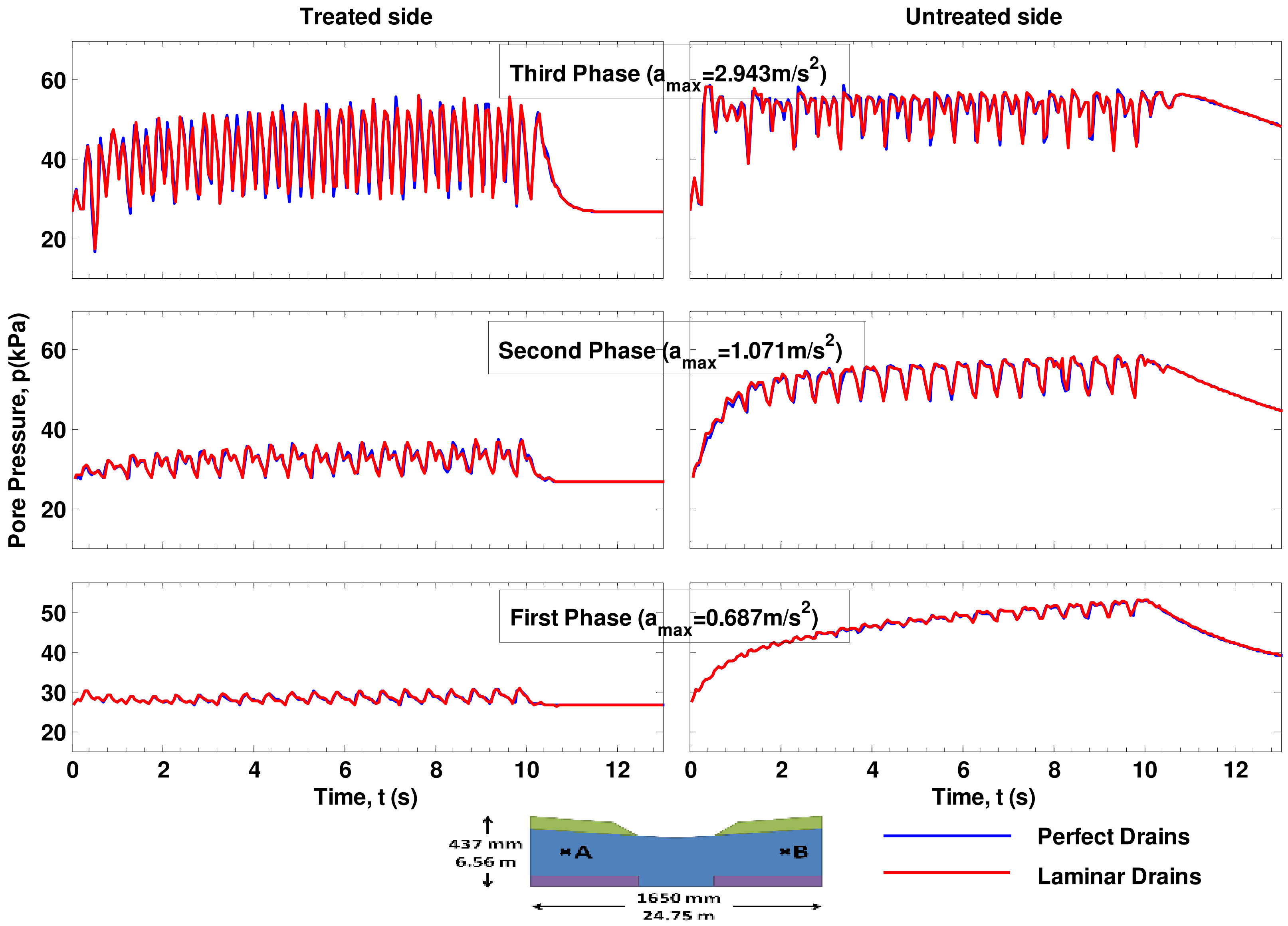


Figure 5-22 Comparison of predicted pore pressures using the perfect drains vs. laminar drains assumption (base case: laminar drains)

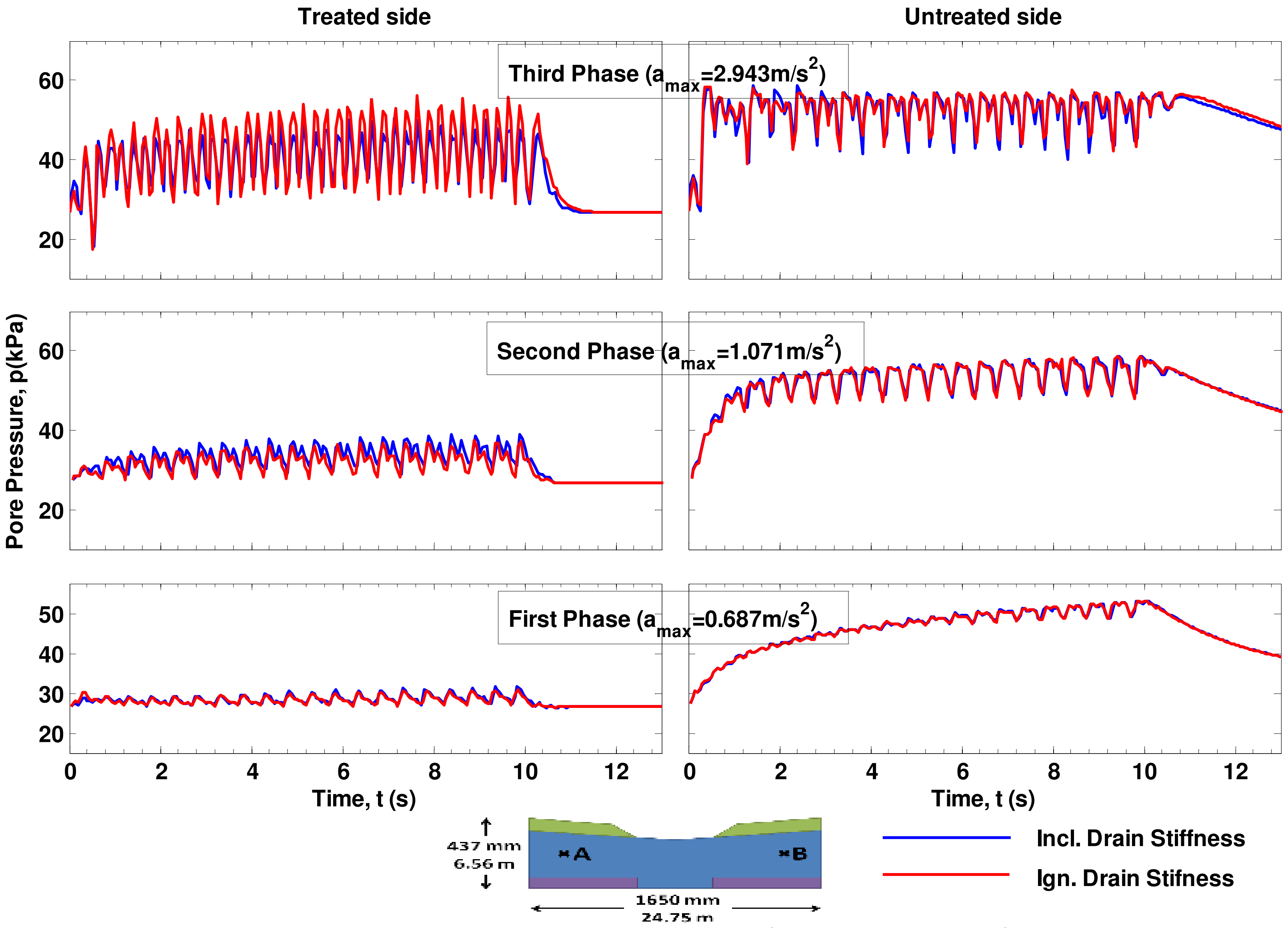


Figure 5-23 Comparison of predicted pore pressures including and ignoring the effect of drain stiffness (base case: incl. drain stiffness)

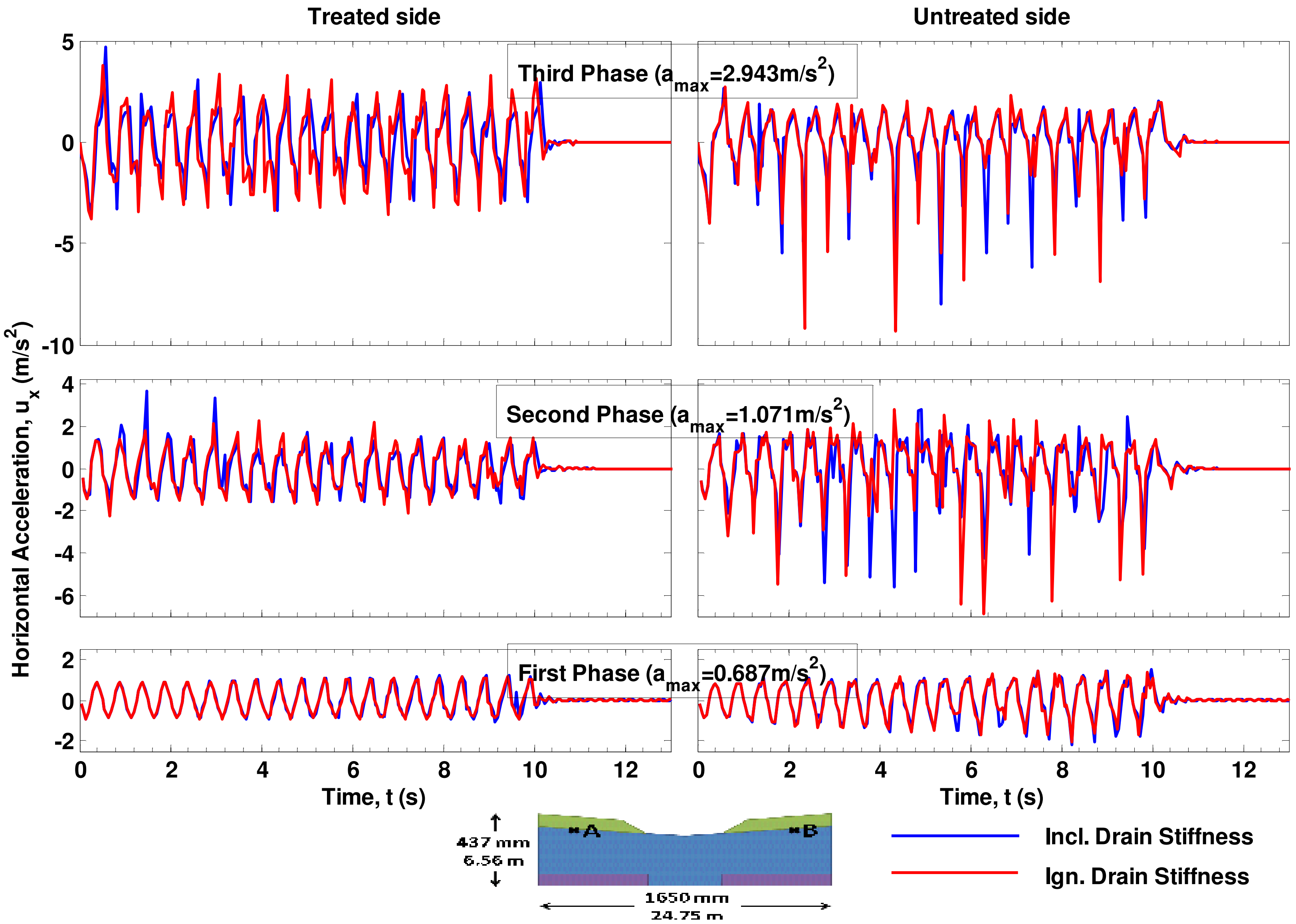


Figure 5-24 Comparison of predicted horizontal accelerations including and ignoring the effect of drain stiffness (base case: incl. drain stiffness)

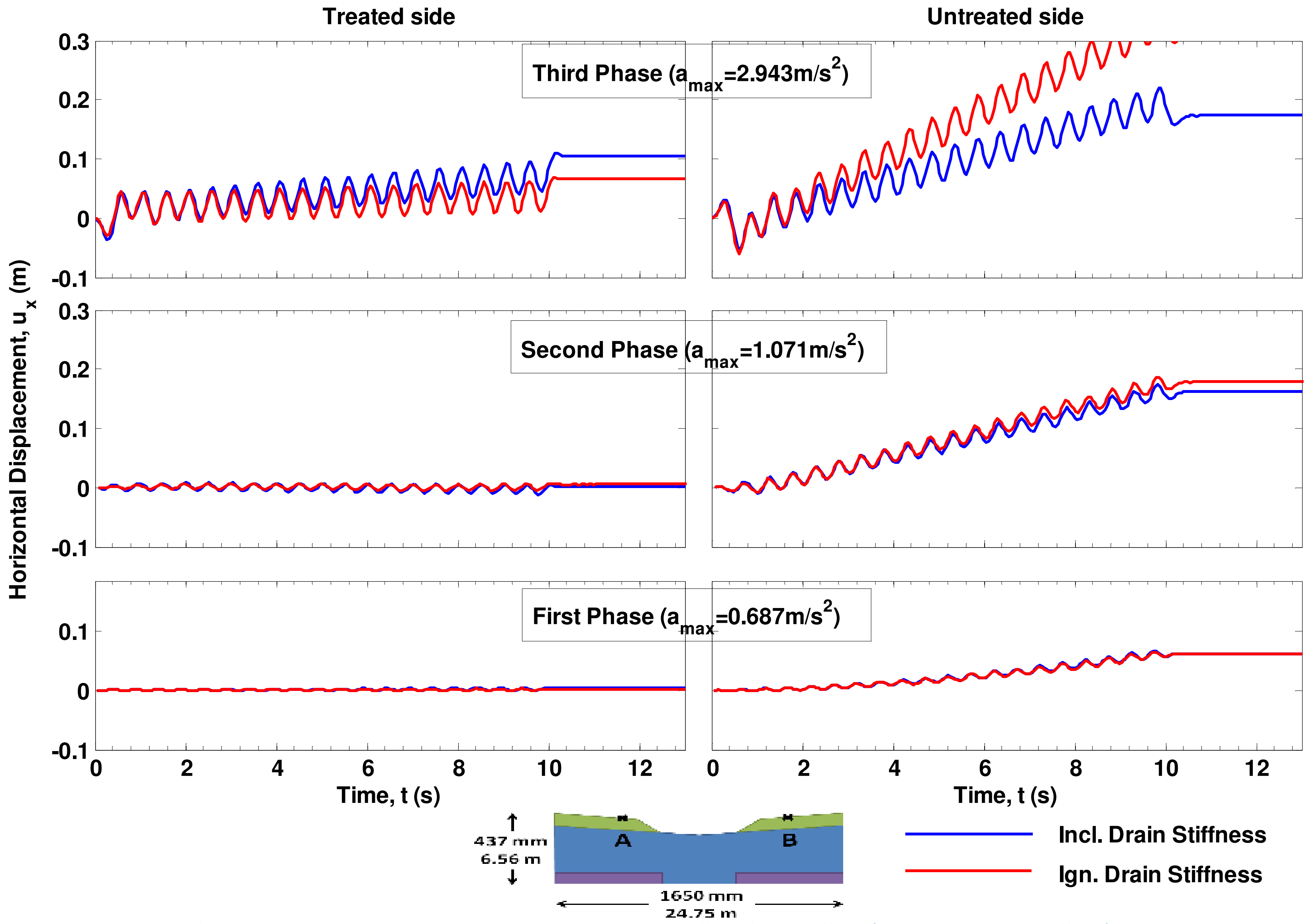


Figure 5-25 Comparison of predicted horizontal displacements including and ignoring the effect of drain stiffness (base case: incl. drain stiffness)

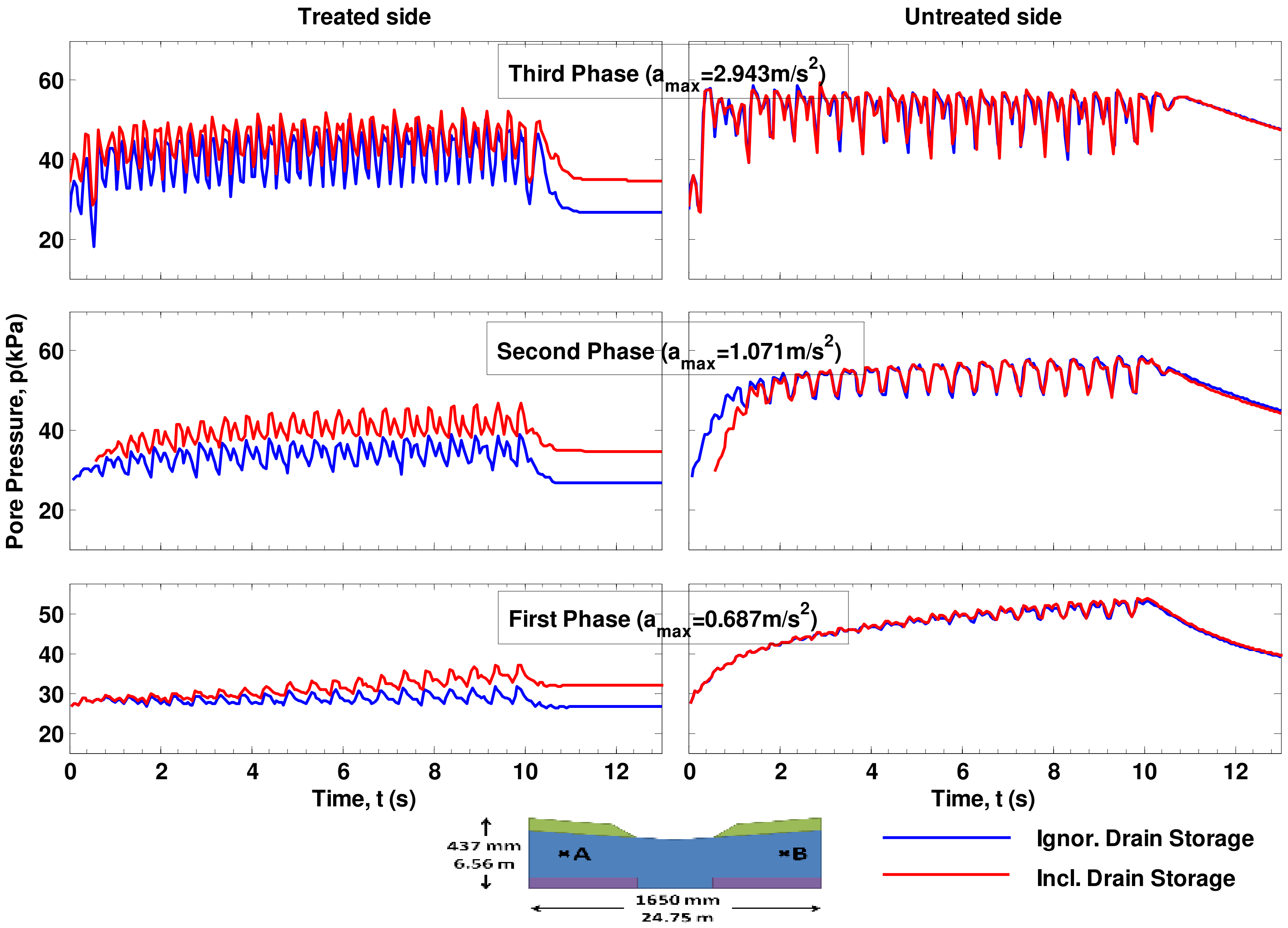


Figure 5-26 Comparison of predicted pore pressures illustrating the effect of drain storage (base case: incl. drain storage)

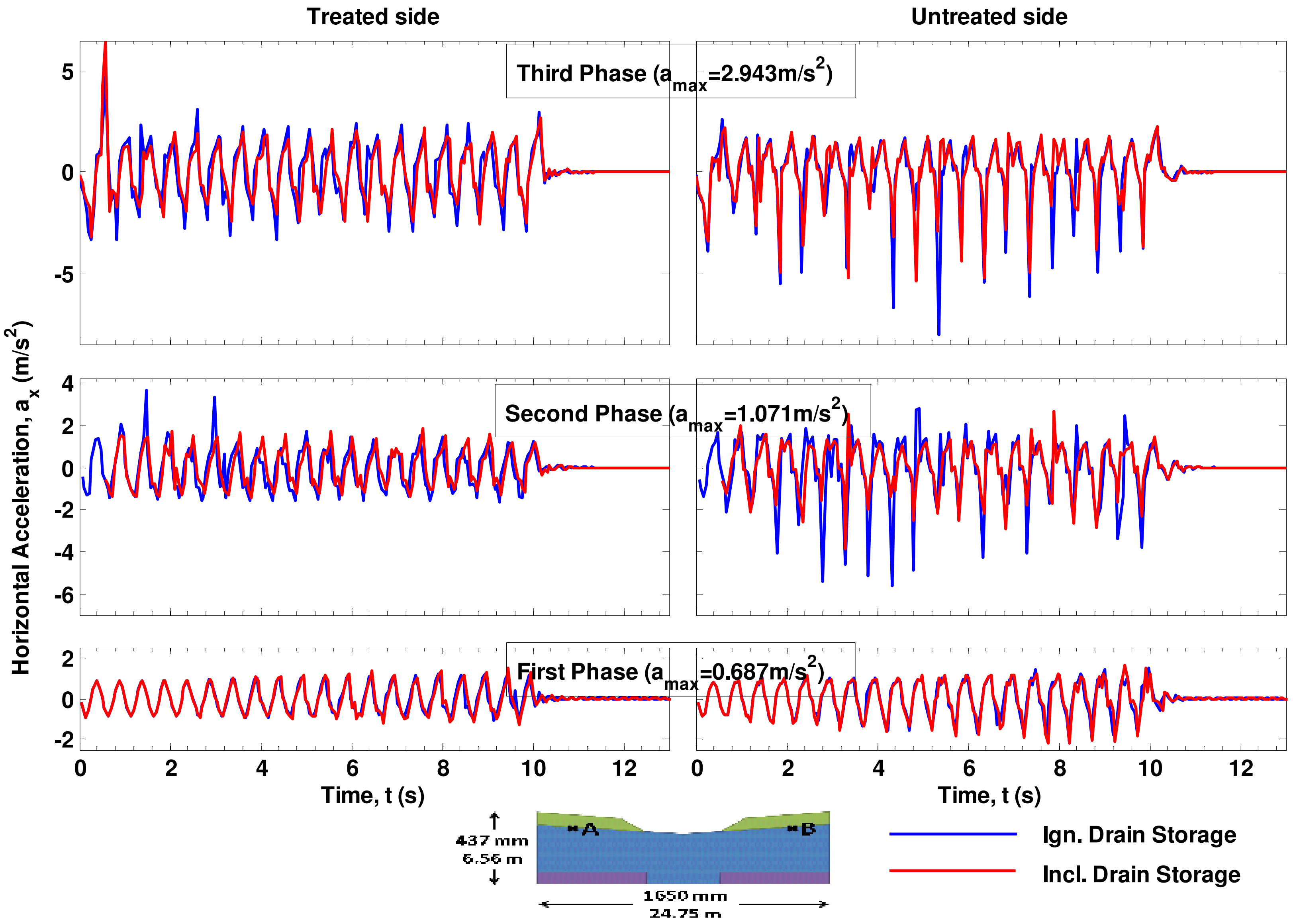


Figure 5-27 Comparison of predicted horizontal accelerations illustrating the effect of drain storage (base case: incl. drain storage)

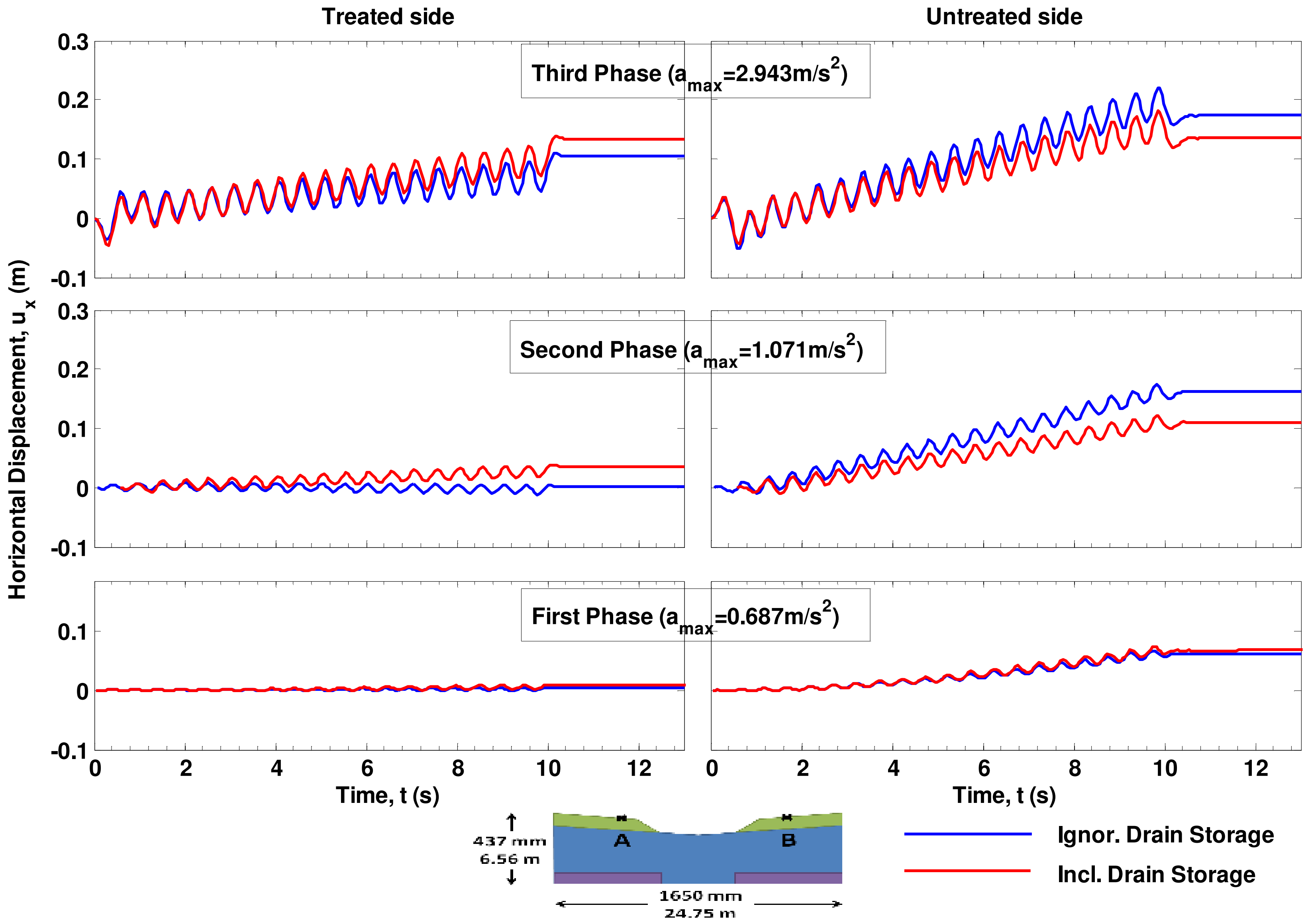


Figure 5-28 Comparison of predicted horizontal displacements illustrating the effect of drain storage (base case: incl. drain storage)

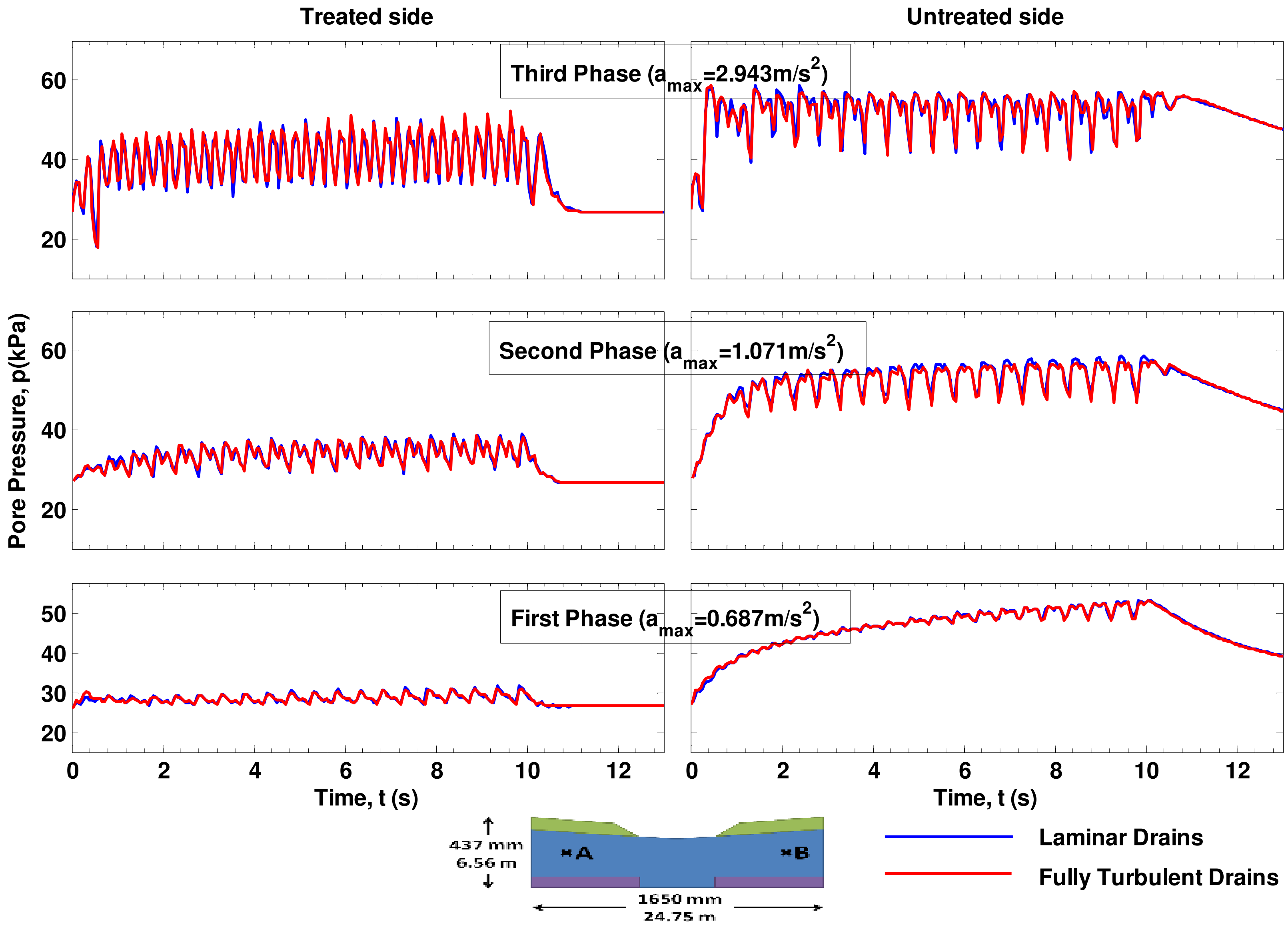


Figure 5-29 Comparison of predicted pore pressures using laminar vs. fully turbulent drains (base case: laminar drains)

6 Summary, Conclusions, and Recommendations

This thesis focuses on three different issues: (a) it establishes a method to estimate the response of soil with vertical drains, (b) it discusses similitude issues for centrifuge modeling, and (c) it performs validation analyses of the implemented numerical methods.

6.1 Simulating Vertical Drains

This thesis used an uncoupled theory of mechanical deformation and flow inside a drain in order to investigate the effect of earthquake drains during cyclic loading of sandy deposits. The mechanical part of the drain's response is idealized as a truss[7]. The pore pressure flow part of the drain's response is assumed to be either fully turbulent or laminar.

Classes written in C++ have been implemented in the OpenSees FEM framework in order to simulate the drains. Both formulations have been shown to work effectively, and give reasonable results, within acceptable convergence levels and speed. For the fully turbulent drains, a consistent Jacobian integration scheme is used due to its superior performance compared to the continuous Jacobian.

The effect of storage capacity has also been successfully modeled. Care should be taken by researchers to achieve strict convergence when they use this feature, since small spurious perturbations in the predicted flow inside the drain could significantly affect the storage effect.

[7] Hence the drain carries axial load applied to through nodal connections by the adjacent soil.

6.2 Similitude Issues

This thesis investigated similitude issues having to do with the design of the model scale drains. It is shown that if both in the model scale and in the prototype scale the flow is laminar then the transmissivity parameter C_l is N^3 times larger in the prototype scale. It is also shown that if both flow are turbulent, then the transmissivity parameter C_t is $N^{2.5}$ times larger in the prototype scale.

Also, it is shown that the Reynolds number in prototype scale and in model scale in the drains is different. It is actually possible that model scale flow is laminar whereas prototype scale flow is turbulent. For this situation, a methodology is proposed for experimental design, in order to choose a model-scale diameter for the drains (where flow is laminar) that best fits the prototype response (where fully turbulent flow is expected), the prototype diameter D^P should correspond to model diameter D^M of:

$$D^M = \left(\frac{76.8\ \mu^M}{\sqrt{i_{max}} \cdot N^3} \sqrt{\frac{{D^P}^5}{2 \cdot \lambda \cdot \rho}} \right)^{\frac{1}{4}} \tag{6.1}$$

6.3 Validation

The implementation of the PV drains has been validated against centrifuge experiments performed at UC-Davis by Kamai et al. (2008). Results showed great accordance with the experiment. Discrepancies between the simulated and experimental results are attributed mostly to uncertainties of the soil permeability and relative density. The validation illustrates that the storage effect is significant for the numerical analysis of the cyclic response of soils treated with

PV-drains. Also, with the design parameters used in the centrifuge model, it is found that the drains behave almost as perfect drains. A different set of tests is needed to evaluate the differences between perfect drain, laminar drain, or fully turbulent drain assumption.

6.4 Future research

The following issues need to be addressed in future research:

- Further investigate the need to model flow in the drains as fully turbulent, by examining real-life scenarios rather than model scale setups.
- Improve the constitutive soil model predictions. Most important is the ability to be able to simulate the response of layers of the same sand at different stress levels, with different void ratios using the same set of parameters.
- Implement the mechanical deformation of the drain using beam theory rather than truss theory. This would allow use of the drain elements for simulation of the response of layers improved with stone columns.
- Apart from acquiring expertise in the simulation of a geotechnical earthquake engineering problem consisting of shear wave propagation in soil layers with pre-installed PV drains, it is very important issue to create design charts for engineers. These should give recommendations for engineers for various levels of shaking (1), number of cycles (2), frequency of loading (3), stress level (4), and soil type (5), in order to:
    - evaluate the applicability of the improvement method by means of estimating the maximum pore pressure ratio (u_{max}/σ_{v0})
    - estimate the spacing and the types of the drains
    - estimate the size of the zone that needs improvement

7 Bibliography

Andrianopoulos, K. I. (2006). *Numerical Modeling of Static & Dynamic Behavior of Elastoplastic Soils.* Athens: Geotechnical Division NTUA (in Greek).

Arulanandan, K., & Subico, J. J. (1992). Post liquefaction settlement of sands. *Proceedings of the Wroth Memorial Symposium.* England: Oxford University.

Biot, M. (1956). Theory of Propagation of Elastic Waves in a Fluid-Saturated Porous Solid. I. Low-Frequency range. *The Journal of the Acoustical Society of America , 28* (2), 168-178.

Conlee, C., Gallagher, P., Kamai, R., Boulanger, R., & Rix, G. (2008). *Evaluation of the Effectiveness of Colloidal Silica for Liquefaction Remediation: Centrifuge Data Report for CTC01.* Center for Geotechnical Modeling, University of California at Davis.

Corral, G. *Cyclic Undrained Behavior of Loose Sand Treated by Colloidal Silica.* MIT. Unpublished.

Dafalias, Y. F., & Manzani, M. T. (2004). Simple Plasticity Sand Model Accounting for Fabric Change Effects. *Journal of Engineering Mechanics* , 622-634.

Dafalias, Y. (1994). Overview of constitutive models used in VELACS. *Verification of Numerical Procedures for the Analysis of Soil Liquefaction Problems.* Rotterdam: A.A. Balkema Publishers.

DeAlba, P., Chan, C. K., & Seed, H. B. (1975). *Determination of Soil Characteristics by Large Scale Laboratoty Tests.* Eathquake Engineering Research Center.

Elgamal, A., Yang, Z., & Parra, E. (2002). Computational Modeling of Cyclic Mobility and Post-Liquefaction Site Response. *Soil Dynamics and Earthquake Engineering , 22*, 259-271.

Gallagher, P. M., & Mitchell, J. (2002). Influence of Colloidal Silica Grout on Liquefaction Potential and Cyclic Undrained Behavior of Loose Sand. *Soil Dynamics and Earthquake Engineering , 22*, 1017-1026.

Gallagher, P. M., Pamuk, A., & Abdoun, T. (2007). Stabilization of Liquefiable Soils Using Colloidal Silica Grout. *Journal of Materials in Civil Engineering , 19* (1), 33-40.

Hird, C, C., Pyrah, I. C., & Russell, D. (1992). Finite Element Modelling of Vertical Drains beneath Embankments on Soft Ground. *Geotechnique , 42* (3), 499-511.

Hughes, T., & Pister, K. (1978). Consistent linearization in mechanics of solids and structures. *Computers and Structures , 8*, 391-397.

Jeremic, B. (2008). *Lecture Notes on Computational Geomechanics: Inelastic Finite Elements for Pressure Sensitive Materials.* Department of Civil and Environmental Engineering, University of California, Davis.

Kamai, R., Howell, R., Conlee, C., Boulanger, R., Marinucci, A., Rathje, E., et al. (2008). *Evaluation of the Effectiveness of Prefabricated Vertical Drains for Liquefaction Remediation - Centrifuge Data Report for RNK01.* Department of Civil & Environmental Engineering, College of Engineering, University of California at Davis.

Kamai, R., Kano, S., Conlee, C., Marinucci, A., Boulanger, R., Rathje, E., et al. (2008). *Evaluation of the Effectiveness of Prefabricated Vertical Drains for Liquefaction Remediation: Centrifuge Data Report for SSK01.* Center for Goetechnical Modeling, University of California at Davis.

Kammerer, A. M., Pestana, J. M., & Seed, R. (2002). *Undrained response of monterey 0/30 sand under multidirectional cyclic simple shear loading conditions.* University of California, Berkeley.

Kodaka, T., Oka, F., Ohno, Y., Takyu, T., & Yamasaki, N. (2003). Modeling of Cyclic Deformation and Strength Characteristics of Colloidal Silica Treated Sand. *Geomechanics 2003* (pp. 205-216). ASCE.

Kolymbas, D. (2000). *Introduction to Hypoplasticity.* Taylor & Francis.

Lambe, T. W. (1973). Predictions in soil engineering. *Geotechnique , 23* (2), 149-202.

Lee, K., & Albaisa, A. (1974). Earthquake Induced Settlements in Saturated Sands. *Journal of the Geotechnical Engineering Division , 100* (GT4), 387-406.

Li, X. S., & Dafalias, Y. F. (2002). Constitutive Modeling of Inherently Anisotropic Sand Behavior. *Journal of Geotechnical and Geoenvironmental Engineering , 128* (10), 868-880.

Liao, H., Huang, C., & Chao, B. (2003). Liquefaction Resistance of a Colloidal Silica Grouted Sand. *Grouting 2003*, (pp. 1305-1313).

Mazzoni, S., McKenna, F., & Fenves, G. L. (2005). *Opensees Command Language Manual.*

Nunez, I. L. (1988). Driving and tesion loading of piles in sand on a centrifuge. *Centrifuge 1988*, (pp. 353-362). Balkema, Rotterdam.

Oka, F., Yashima, A., Tateishi, A., Taguchi, Y., & Yamashita, S. (1999). A cyclic elasto-plastic model constitutive model for sand considering a plastic-strain dependence of the shear modulus. *Geotechnique , 49* (5), 661-680.

Onoue, A. (1988). Diagrams considering Well Resistance for Designing Spacing Ratio of Gravel Drains. *Soils and Foundations , 28* (3), 160-168.

Onoue, A., Mori, N., & Takano, J. (1987). In-situ experiment and analysis on well resistance of gravel drains. *Soils and Foundations , 27* (2), 42-60.

Papadimitriou, A. G., & Bouckovalas, G. D. (2001). Plasticity Model for Sand under Small and Large Cyclic Strains: a Multi-axial Formulation. *Soil Dynamics and Earthquake Engineering , 22* (3), 191-204.

Papadimitriou, A. G., Bouckovalas, G. D., & Dafalias, Y. F. (2001). Plasticity Model for Sand under Small and Large Cyclic Strains. *Journal of Geotechnical and Geoenvironmental Engineering , 127* (11), 973-983.

Papadimitriou, A., Moutsopoulou, M.-E., Bouckovalas, G., & Brennan, A. (2007). Numerical Investigation of Liquefaction Mitigation using Gravel Drains. *4th International Conference on Earthquake Geotechnical Engineering.* Thessaloniki - GREECE 2007: 4ICEGE.

Pestana, J. M., & Whittle, A. J. (2002). Evaluation of a constitutive model for clays and sands: Part I - sand behaviour. *Iinternational Journal for Numerical and Analytical Methods in Geomechanics , 26*, 1097-1121.

Pestana, J. M., & Whittle, A. J. (2002). Evaluation of a constitutive model for clays and sands: Part II - clay behaviour. *Iinternational Journal for Numerical and Analytical Methods in Geomechanics , 26*, 1123-1146.

Pestana, J. M., & Whittle, A. J. (1999). Formulation of a unified constitutive model for clays and sands. *Iinternational Journal for Numerical and Analytical Methods in Geomechanics , 23*, 1215-1243.

Pestana, J. M., Hunt, C. E., Goughnour, R. R., & Kammerer, A. M. (1997). *Effect of Storage Capacity on Vertical Drain Performance in Liquefiable Sand Deposits.*

Prevost, J. (1985). A Simple Plasticity Theory for Frictional Cohesionless Soils. *Soil Dynamics and Earthquake Engineering , 4* (1), 9-17.

Randolph, M. F., Cassidy, M. J., Gourvenec, S. M., & Erbrich, C. (2005). Challenges of offshore geotechnical engineering. *Proceedings of the 16th International Conference on Soil Mechanics and Geotechnical Engineering*, (pp. 123-176). Osaka, Japan.

Remaud, D. (1999). *Pieux sous charges latérales: Etude expérimentale de l'effet de groupe.* Thése de Doctorat de l'Université de Nantes (in French).

Schofield, A. (1980). Cambridge geotechnical centrifuge operations. *Géotechnique , 25* (4), 227-268.

Seed, H. B., & Booker, J. R. (1977). Stabilization of Potentially Liquefiable Sand Deposits using Gravel Drains. *Journal of the Geotechnical Engineering Division , 103* (GT7), 757-768.

Simo, J., & Hughes, T. (1998). *Computational Inelasticity.* New York: Springer-Verlang.

Spencer, L., Rix, G., & Gallagher, P. (2007). Dynamic Properties of Colloidal Silica Gel and Sand Mixtures. *Fourth International Conference on Earthquake Geotechnical Engineering.*

Stone, K., & Muir, W. (1992). Effects of dilatancy and particle size observed in model tests on sand. *Soils and Foundations , 32* (4), 43-47.

Ternet, O. (1999). Reconstitution et caractérization des massifs de sable: application aux essais en centrifugeuse et en chambre de calibration. *Thése de doctorat* . Université de Caen (in French).

Towhata, I. (2007). Developments of Soil Improvement Technologies for Mitigation of Liquefaction Risk. *4th Internationl Conference on Earthquake Geotechnical Engineering* (pp. 355-383). Thessaloniki: Springer.

Whittle, A., & Kavvadas, M. (1994). Formulation of MIT-E3 Constitutive Model for Overconsolidated Clays. *Journal of Geotechnical Engineering , 120* (1), 173-198.

Yang, Z., Elgamal, A., & Parra, E. (2003). Computational Model for Cyclic Mobility and Associated Shear Deformation. *Journal of Geotechnical and Geoenvironmental Engineering , 129* (12), 1119-1127.

Zienkiewicz, O., Chan, A., Pastor, M., Schrefler, B., & Shiomi, T. (1999). *Computational Geomechanics.* Chichester: John Wiley & Sons Ltd.

Appendix A

One-dimensional dynamic response of a fully saturated soil column

The reference problem chosen is finding the steady state dynamic response of a fully saturated soil column in which free drainage and a sinusoidal pressure are applied at the top (Zienkiewicz et al, 1999). The material is linear elastic and the pore water is considered to be incompressible.

$$\sigma = \sigma^{'} - p \tag{A.1}$$

$$\epsilon = \frac{\partial u}{\partial z} \tag{A.2}$$

$$\sigma' = D \cdot \epsilon \tag{A.3}$$

$$\frac{d\sigma}{dz} = \rho\ddot{u} + \rho_f\ddot{w} \tag{A.4}$$

$$-\frac{dp}{dz} = \rho_f\ddot{u} + \frac{\rho_f}{n}\ddot{w} + \frac{\rho_f g}{k}\dot{w} \tag{A.5}$$

$$\dot{\epsilon} + \frac{dw}{dz} = -\frac{\dot{p}n}{K_f} \tag{A.6}$$

where σ is the vertical stress, σ' is the effective stress, ε is the vertical strain, u is the vertical displacement, D is the one-dimensional compression modulus, ρ is the density of the total composite, ρ_f is the density of the fluid phase, w is the pore water displacement relative to the soil skeleton, K_f is the compressibility of the fluid phase, p is the pore water pressure and n is the porosity. From these equations we can get a system of ordinary differential equations of u and w:

$$\left(D + \frac{K_f}{n}\right)\frac{d^2u}{dz^2} + \frac{K_f}{n}\frac{dw^2}{dz^2} = \rho\ddot{u} + \rho_f\ddot{w} \tag{A.7}$$

$$\left(\frac{d^2u}{dz^2} + \frac{dw^2}{dz^2}\right)\frac{K_f}{n} = \rho_f\ddot{u} + \frac{\rho_f}{n}\ddot{w} + \frac{\rho_f g}{k}\dot{w} \tag{A.8}$$

Next we apply separation of variables to sole the above system. In the case of a sinusoidal excitation:

$$q \ = q_0 e^{i\omega t} \tag{A.9}$$

the solution of the σ, u, w variables is of the form of

$$X e^{i\omega t} \tag{A.10}$$

where X is a space function. Using the above we have:

$$u = \bar{u} e^{i\omega t} \tag{A.11}$$

$$w = \overline{w} e^{i\omega t} \tag{A.12}$$

$$\left(D + \frac{K_f}{n}\right)\frac{d^2\bar{u}}{dz^2} + \frac{K_f}{n}\frac{d\overline{w}^2}{dz^2} \ = -\omega^2 \rho \bar{u} - \omega^2 \rho_f \overline{w} \tag{A.13}$$

$$\left(\frac{d^2\bar{u}}{dz^2} + \frac{d\overline{w}^2}{dz^2}\right)\frac{K_f}{n} \ = -\omega^2 \rho_f \bar{u} - \omega^2 \frac{\rho_f}{n}\overline{w} + i\omega \frac{\rho_f g}{k}\ \overline{w} \tag{A.14}$$

Note that the results will be complex numbers. We define:

$$\kappa = \frac{\frac{K_f}{n}}{D + \frac{K_f}{n}} \tag{A.15}$$

$$\beta = \frac{\rho_f}{\rho} \tag{A.16}$$

$$V_c^2 = \frac{D + \frac{K_f}{n}}{\rho} \tag{A.17}$$

$$\bar{z} = \frac{z}{L} \tag{A.18}$$

$$T = \frac{2\pi}{\omega} \tag{A.19}$$

$$\hat{T} = \frac{2L}{V_c} \tag{A.20}$$

$$\Pi_1 = \left(\frac{2}{\beta\pi}\right)\frac{k}{g}\frac{T}{\hat{T}^2} \tag{A.21}$$

$$\Pi_2 = \pi^2 \left(\frac{\hat{T}}{T}\right)^2 \tag{A.22}$$

The previous equations take the following form:

$$\frac{d^2\bar{u}}{dz^2} + \kappa \frac{d^2\overline{w}}{dz^2} = -\Pi_2\bar{u} - \beta\Pi_2\overline{w} \tag{A.23}$$

$$\kappa \frac{d^2\bar{u}}{dz^2} + \kappa \frac{d^2\overline{w}}{dz^2} = -\beta\Pi_2\bar{u} - \frac{\beta}{n}\Pi_2\overline{w} + \frac{i}{\Pi_1}\overline{w} \tag{A.24}$$

The above equations can be written in a canonical form:

$$\frac{d^2\bar{u}}{dz^2} = \frac{\beta\Pi_2 - \Pi_2}{1-\kappa}\bar{u} + \frac{\frac{\beta}{n}\Pi_2 - \frac{i}{\Pi_1} - \beta\Pi_2}{1-\kappa}\overline{w} \tag{A.25}$$

$$\frac{d^2\overline{w}}{dz^2} = \frac{-\beta\Pi_2 - \frac{(\beta-1)\Pi_2\kappa}{1-\kappa}}{\kappa}\bar{u} + \frac{-\frac{\beta}{n} + \frac{i}{\Pi_1} - \frac{\kappa(\frac{\beta}{n}\Pi_2 - \frac{i}{\Pi_1} - \beta\Pi_2)}{1-\kappa}}{\kappa}\overline{w} \tag{A.26}$$

And we define:

$$A = \frac{\beta\Pi_2 - \Pi_2}{1-\kappa} \tag{A.27}$$

$$B = \frac{\frac{\beta}{n}\Pi_2 - \frac{i}{\Pi_1} - \beta\Pi_2}{1-\kappa} \tag{A.28}$$

$$\Gamma = \frac{-\beta\Pi_2 - \frac{(\beta\Pi_2 - \Pi_2)\kappa}{1-\kappa}}{\kappa} \tag{A.29}$$

$$\Delta = \frac{-\frac{\beta}{n} + \frac{i}{\Pi_1} - \frac{\kappa(\frac{\beta}{n}\Pi_2 - \frac{i}{\Pi_1} - \beta\Pi_2)}{1-\kappa}}{\kappa} \tag{A.30}$$

The solution of the system of PDE's is:

$$u = C_i B e_i^{\lambda_i \bar{z}} \tag{A.31}$$

$$w = C_i(\lambda_i^2 - A)e_i^{\lambda_i \bar{z}} \tag{A.32}$$

Where λ_i's are the solutions of the characteristic equation:

$$\lambda^4 - (\mathrm{A} - \Delta)\lambda^2 + \mathrm{A} \cdot \Delta - \mathrm{B} \cdot \Gamma = 0 \tag{A.33}$$

Next, we apply the boundary conditions. At the surface:

$$\bar{z} = 0, \qquad \bar{\sigma} = \bar{q}, \qquad \bar{p} = 0 \Rightarrow \tag{A.34}$$

$$\frac{d\bar{u}}{d\bar{z}} = \frac{\bar{q}L}{D} \tag{A.35}$$

$$\frac{d\bar{w}}{d\bar{z}} = -\frac{\bar{q}L}{D} \tag{A.36}$$

And at the bottom,

$$\bar{z} = 1, \qquad \bar{u} = 0, \qquad \frac{d\bar{p}}{d\bar{z}} = 0 \Rightarrow \tag{A.37}$$

$$\bar{u} = 0 \tag{A.38}$$

$$\bar{w} = 0 \tag{A.39}$$

The boundary conditions give us the following system of equations that should be solved:

$$C_i\lambda_i = \frac{\bar{q}L}{DB} \tag{A.40}$$

$$C_i(\lambda_i^2 - A)\lambda_i = \frac{\bar{q}L}{D} \tag{A.41}$$

$$C_i\lambda_i = 0 \tag{A.42}$$

$$C_i(\lambda_i^2 - A)\lambda_i = 0 \tag{A.43}$$

Finally, we need to estimate p, the pore pressure vs depth:

$$-\frac{dp}{dz} = \rho_f\ddot{u} + \frac{\rho_f}{n}\ddot{w} + \frac{\rho_f g}{k}\dot{w} \tag{A.44}$$

$$\frac{dp}{dz} = \left(\omega^2\rho_f\bar{u} + \omega^2\frac{\rho_f}{n}\bar{w} - i\omega\frac{\rho_f g}{k}\bar{w}\right)e^{i\omega t} \tag{A.45}$$

$$dp = \left(\omega^2\rho_f\bar{u} + \omega^2\frac{\rho_f}{n}\bar{w} - i\omega\frac{\rho_f g}{k}\bar{w}\right)e^{i\omega t}d\bar{z}L \tag{A.46}$$

$$p = \int_0^{\bar{z}} \left(\omega^2 \rho_f \bar{u} + \omega^2 \frac{\rho_f}{n} \overline{w} - i\omega \frac{\rho_f g}{k} \overline{w}\right) e^{i\omega t} d\bar{z} L \tag{A.47}$$

$$p = \left(\omega^2\ C_i \frac{\mathrm{B}}{\lambda_i} e_i^{\lambda_i \bar{z}-1} + (\frac{\rho_f}{n}\omega^2 - \rho_f g\ i\frac{\omega}{k}) \frac{C_i}{\lambda_i} (\lambda_i^2 - \mathrm{A})(e_i^{\lambda_i \bar{z}} - 1)\ \right) L \tag{A.48}$$

In order to find a solution to u-p approximation of this problem, we omit the relative acceleration of the pore fluid to the soil skeleton:

$$\frac{d^2 \bar{u}}{dz^2} + \kappa \frac{d^2 \overline{w}}{dz^2} = -\Pi_2 \bar{u} \tag{A.49}$$

$$\kappa\ \frac{d^2 \bar{u}}{dz^2} + \kappa \frac{d^2 \overline{w}}{dz^2} = -\beta \Pi_2 \bar{u} + \frac{i}{\Pi_1}\ \overline{w} \tag{A.50}$$

The solution is the same as above, except for the fact than A, B, Γ, and Δ are defined:

$$A = \frac{\beta \Pi_2 - \Pi_2}{1-\kappa} \tag{A.51}$$

$$B = \frac{-\frac{i}{\Pi_1}}{1-\kappa} \tag{A.52}$$

$$\Gamma = \frac{-\beta \Pi_2 - \frac{(\beta \Pi_2 - \Pi_2)\kappa}{1-\kappa}}{\kappa} \tag{A.53}$$

$$\Delta = \frac{\frac{i}{\Pi_1} - \frac{\kappa(-\frac{i}{\Pi_1})}{1-\kappa}}{\kappa} \tag{A.54}$$

Appendix B

Matlab Code Solving Analytically the full formulation of the dynamic response of a soil column (Figure 2-1)

```
function [u max_u p max_p w H L]=coupl_1d(t)
% This function solves analytically the 1d coupled (pore
% pressure-displacement) dynamic response problem of a soil column.
% t is the time of interest
% It uses the real part of exp(i*omega*t) so the excitation is a cosine

K_f=2200000;    % Volumetric compressibility of the fluid
n=0.333;      % Porosity
E=30000;     % Elastic modulus of the soil skeleton
v=0.3;      % Poisson’s ratio
rho_f=1.; % Fluid density
rho=2.;   % Average density of multi-phase medium
L=10;       % Height of soil column
omega=10.;   % Natural Frequency of the Applied Load
q=1;        % Amplitude of the Applied Load
g=10;     % Acceleration of gravity
kappa=0.2;    % Permeability (Hydraulic Conductivity)

rho_dry=rho-n*rho_f
e=n/(1-n)

D_oned=E*(1-v)/((1+v)*(1-2*v))
k=(K_f/n)/(D_oned+K_f/n)
V_c2=(D_oned+K_f/n)/rho;
beta=rho_f/rho
sqrt(V_c2)

T=2*pi/omega
T_star=2*L/sqrt(V_c2);

Pi_1=(2/beta/pi)*kappa*T/g/(T_star^2)
Pi_2=pi^2*(T_star/T)^2

% Now we solve the system of differential equations:
% Look at the theory
% The equivalent equations are:
% d2u/dz2=A*u+B*w
% d2w/dz2=C*u+D*w
% The general solution is:
% u=Ci*b*e^(lambda_i*z) (Einstein summation convention)

A=(beta*Pi_2-Pi_2)/(1-k);
B=(beta/n*Pi_2-i/Pi_1-beta*Pi_2)/(1-k);
C=-beta*Pi_2-k/(1-k)*(beta*Pi_2-Pi_2);
C=C/k;
D=-beta/n*Pi_2+i/Pi_1-k/(1-k)*(beta/n*Pi_2-i/Pi_1-beta*Pi_2);
D=D/k;

% Now we create the characteristic polynomial
% The solution of the characteristic polynomial are the lambda's
P_char=[1 0 -(A+D) 0 (A*D-B*C)];
lambda=roots(P_char);

if (A*D-B*C)==0
    'Beware A*D-B*C=0'
end

if (A-D)^2+4*B*C==0
    'Beware (A-D)^2+4*B*C=0'
end

% We solve the system L_m*X=R
% The solution of this system is the C_i's
L_m=[exp(lambda(1)) exp(lambda(2)) exp(lambda(3)) exp(lambda(4));...
    (lambda(1)^2-A)*exp(lambda(1))  (lambda(2)^2-A)*exp(lambda(2))  (lamb-
da(3)^2-A)*exp(lambda(3))  (lambda(4)^2-A)*exp(lambda(4));...
    lambda(1) lambda(2) lambda(3) lambda(4);...
    (lambda(1)^2-A)*lambda(1)  (lambda(2)^2-A)*lambda(2)  (lambda(3)^2-
A)*lambda(3)  (lambda(4)^2-A)*lambda(4)];

R=[0 ; 0; q*L/D_oned/B; -q*L/D_oned];
X=L_m\R;

% t=1          % The absolute time

n_inc=1000;
L_inc=L/n_inc;
z=0;
i_n=0;
H=0;

% We calculate the displacements
while z<=1
    i_n=i_n+1;
    z=z+L_inc/L;
    u(i_n)=0;
    for i_it=1:4
        u(i_n)=u(i_n)+X(i_it)*B*exp(lambda(i_it)*z);
    end
    temp_u=u(i_n);
    u(i_n)=abs(u(i_n));
    max_u(i_n)=u(i_n);
    u(i_n)=u(i_n)*real(exp(i*(omega*t-phase(temp_u))));
    H(i_n)=z*L;
end

% We calculate the fluid displacement
z=0;
i_n=0;
while z<=1
    i_n=i_n+1;
    z=z+L_inc/L;
    w(i_n)=0;
    for i_it=1:4
```

```
        w(i_n)=w(i_n)+X(i_it)*(lambda(i_it)^2-A)*exp(lambda(i_it)*z);
    end
    temp_w=w(   i_n);
    w(i_n)=abs(w(i_n));
    w(i_n)=w(i_n)*real(exp(i*(omega*t-phase(temp_w))));
end


% We calculate the pore pressure
z=0;
i_n=0;
while z<=1
    i_n=i_n+1;
    z=z+L_inc/L;
    p(i_n)=0;
    for i_it=1:4

p(i_n)=p(i_n)+omega^2*rho_f*X(i_it)*B/lambda(i_it)*(exp(lambda(i_it)*z)-
1)+...
            (rho_f*omega^2/n-rho_f*g*i*omega/kappa)*X(i_it)*(lambda(i_it)^2-
A)/lambda(i_it)*...
            (exp(lambda(i_it)*z)-1);
    end
    p(i_n)=p(i_n)*L;
    temp_p=p(i_n);
    p(i_n)=abs(p(i_n));
    max_p(i_n)=p(i_n);
    p(i_n)=p(i_n)*real(exp(i*(omega*t-phase(temp_p))));
end
```

Appendix C

Matlab Code Solving Analytically the u-p formulation of the dynamic response of a soil column

```
function [u max_u p max_p w H L]=coupl_1d_up(t)
% This function solves analytically the 1d coupled (pore
% pressure-displacement) dynamic response problem of a soil column.
% This function follows the u-p formulation ignoring the acceleration terms
% for the pore fluid.
% t is the time of interest
% It uses the real part of exp(i*omega*t) so the excitation is a cosine

K_f=2200000;    % Volumetric compressibility of the fluid
n=0.333;      % Porosity
E=30000;     % Elastic modulus of the soil skeleton
v=0.3;      % Poisson's ratio
rho_f=1; % Fluid density
rho=2.;    % Average density of multi-phase medium
L=10;       % Height of soil column
omega=10.;   % Natural Frequency of the Applied Load
q=1;        % Amplitude of the Applied Load
g=10;     % Acceleration of gravity
kappa=0.2;    % Permeability (hydraulic conductivity)

rho_dry=rho-n*rho_f
e=n/(1-n)

D_oned=E*(1-v)/((1+v)*(1-2*v))
k=(K_f/n)/(D_oned+K_f/n)
V_c2=(D_oned+K_f/n)/rho
beta=rho_f/rho
sqrt(V_c2)

T=2*pi/omega
T_star=2*L/sqrt(V_c2);

Pi_1=(2/beta/pi)*kappa*T/g/(T_star^2)
Pi_2=pi^2*(T_star/T)^2

% Now we solve the system of differential equations:
% Look at the theory
% The equivalent equations are:
% d2u/dz2=A*u+B*w
% d2w/dz2=C*u+D*w
% The general solution is:
% u=Ci*b*e^(lambda_i*z) (Einstein summation convention)

A=(beta*Pi_2-Pi_2)/(1-k);
B=(-i/Pi_1)/(1-k);
C=-beta*Pi_2-k/(1-k)*(beta*Pi_2-Pi_2);
C=C/k;
D=i/Pi_1-k/(1-k)*(-i/Pi_1);
D=D/k;
```

```
% Now we create the characteristic polynomial
% The solution of the characteristic polynomial are the lambda's
P_char=[1 0 -(A+D) 0 (A*D-B*C)];
lambda=roots(P_char);

if (A*D-B*C)==0
    'Beware A*D-B*C=0'
end

if (A-D)^2+4*B*C==0
    'Beware (A-D)^2+4*B*C=0'
end

% We solve the system L_m*X=R
% The solution of this system is the C_i's
L_m=[exp(lambda(1)) exp(lambda(2)) exp(lambda(3)) exp(lambda(4));...
    (lambda(1)^2-A)*exp(lambda(1))  (lambda(2)^2-A)*exp(lambda(2))  (lamb-
da(3)^2-A)*exp(lambda(3))  (lambda(4)^2-A)*exp(lambda(4));...
    lambda(1) lambda(2) lambda(3) lambda(4);...
    (lambda(1)^2-A)*lambda(1)  (lambda(2)^2-A)*lambda(2)  (lambda(3)^2-
A)*lambda(3)  (lambda(4)^2-A)*lambda(4)];

R=[0 ; 0; q*L/D_oned/B; -q*L/D_oned];
X=L_m\R;

% t=1          % The absolute time

n_inc=1000;
L_inc=L/n_inc;
z=0;
i_n=0;
H=0;

% We calculate the displacements
while z<=1
    i_n=i_n+1;
    z=z+L_inc/L;
    u(i_n)=0;
    for i_it=1:4
        u(i_n)=u(i_n)+X(i_it)*B*exp(lambda(i_it)*z);
    end
    temp_u=u(i_n);
    u(i_n)=abs(u(i_n));
    max_u(i_n)=u(i_n);
    u(i_n)=u(i_n)*real(exp(i*(omega*t-phase(temp_u))));
    H(i_n)=z*L;
end

% We calculate the fluid displacement
z=0;
i_n=0;
while z<=1
    i_n=i_n+1;
    z=z+L_inc/L;
    w(i_n)=0;
    for i_it=1:4
        w(i_n)=w(i_n)+X(i_it)*(lambda(i_it)^2-A)*exp(lambda(i_it)*z);
    end
    temp_w=w(   i_n);
    w(i_n)=abs(w(i_n));
    w(i_n)=w(i_n)*real(exp(i*(omega*t-phase(temp_w))));
end


% We calculate the pore pressure
z=0;
i_n=0;
while z<=1
    i_n=i_n+1;
    z=z+L_inc/L;
    p(i_n)=0;
    for i_it=1:4

p(i_n)=p(i_n)+omega^2*rho_f*X(i_it)*B/lambda(i_it)*(exp(lambda(i_it)*z)-
1)+...
            (-rho_f*g*i*omega/kappa)*X(i_it)*(lambda(i_it)^2-
A)/lambda(i_it)*...
            (exp(lambda(i_it)*z)-1);
    end
    p(i_n)=p(i_n)*L;
    temp_p=p(i_n);
    p(i_n)=abs(p(i_n));
    max_p(i_n)=p(i_n);
    p(i_n)=p(i_n)*real(exp(i*(omega*t-phase(temp_p))));
end
```

Appendix D

Opensees tcl/tk Code to test the u-p approximation

```
# This analysis solve the dynamic response of a soil column
# upon the application of a constant load on top
# The problem is saturated
wipe
model BasicBuilder -ndm 2 -ndf 3

# Below we define variables for the analysis
set E 30000
set nu 0.3
set numXele 1; # number of elements in x (H) direction
set numYele 40; # number of elements in y (V) direction
set xSize .5; # Element size in x direction
set ySize .25; # Element size in z direction
set numXnode [expr $numXele+1]
set numYnode [expr $numYele+1]
set smass 2.;
set peak_shear_strain 10000.
set c 100.
set G [expr $E/(2*(1+$nu))]
set B [expr $E/(3*(1-2*$nu))]
set i 1
set j 1
set pi 3.141593

# Define material
nDMaterial PressureIndependMultiYield 1 2 $smass $G $B $c $peak_shear_strain\
0. 100. 0. 1

# Define nodes
for {set i 1} {$i <= $numXnode} {incr i 1} {
for {set j 1} {$j <= $numYnode} {incr j 1} {
set xdim [expr ($i-1)*$xSize]
set ydim [expr ($j-1)*$ySize]
set nodeNum [expr $i + ($j-1)*$numXnode]
node $nodeNum $xdim $ydim
}
}

# define elements
set k 0.2
set k [expr $k/10/1.] ;#actual value used in computation
set gravX 0.0
set gravY 0.0
set press 0.0
set bulk_f 2.2e6
set n_por 0.333
set bulk [expr $bulk_f/$n_por]
for {set i 1} {$i <= $numXele} {incr i 1} {
for {set j 1} {$j <= $numYele} {incr j 1} {
set eleNum [expr $i + ($j-1)*$numXele]
set n1 [expr $i + ($j-1)*$numXnode]
set n2 [expr $i + ($j-1)*$numXnode + 1]
set n4 [expr $i + $j*$numXnode + 1]
set n3 [expr $i + $j*$numXnode]
element quadUP $eleNum $n1 $n2 \
$n4 $n3 1.0 1 $bulk 1. $k $k $gravX $gravY $press
}
}
```

```
#vees
# Fix Base:
for {set i 1} {$i <= $numXnode} {incr i 1} {
fix $i 1 1 0
}

# Fix X Direction: (To all but the top nodes)
for {set i [expr $numXnode+1]} {$i <= [expr $numXnode*($numYnode-1)]} {incr i 1} {
fix $i 1 0 0
}

# Drainage on top: (And x-Fixity)

for {set i 1} {$i <= $numXnode} {incr i 1} {
fix [expr $i + ($numYnode-1)*$numXnode] 1 0 1;
}
set omega 10.

# Calculate the period from the timeseries
set T [expr 2*$pi/$omega]
set Timeseries "Sine 0.0 50. $T -shift [expr $pi/2]"
pattern Plain 1 $Timeseries {
for {set i 1} {$i <= $numXnode} {incr i 1} {
load [expr $i + ($numYnode-1)*$numXnode] 0. -.25 0.;
}
}

#build
recorder Node -file output_disp.txt -time \
-node 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39\
41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 -dof 1 2 disp
recorder Node -file porepress.txt -time \
-node 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39\
41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 -dof 3 vel
recorder Node -file output_react.txt -time -node 1 2 -dof 1 2 reaction

#recorder Element -file output_stress.txt material 1 stiffness
set gamma 0.5
test NormDispIncr 1.0e-5 10 0;
algorithm Newton
integrator Newmark $gamma [expr pow($gamma+0.5, 2)/4] \
0.00 0.0 0.002 0.0
analysis Transient
set startT [clock seconds]
analyze 500 0.01
set endT [clock seconds]
puts "Execution time: [expr $endT-$startT] seconds."
```

Appendix E

Opensees tcl/tk code to test the constitutive model

```
model BasicBuilder -ndm 2 -ndf 3
node 1 0.000 0.000
node 2 1.000 0.000
node 3 0.000 1.000
node 4 1.000 1.000

# Depending on the material one should comment/uncomment the respective material
# Dense
# nDMaterial PressureDependMultiYield02 1 2 2.1 130000 260000 36.5 0.1 101. .5 26.
0.013 0.0 0.3 0.0
# Loose
# nDMaterial PressureDependMultiYield02 1 2 1.7 60000 160000 31. 0.1 101. .5 31. 0.087
0.18 0. 0.0
# Medium-Loose
# nDMaterial PressureDependMultiYield02 1 2 1.8 90000 220000 32. 0.1 101. .5 26. 0.067
0.23 0.27 0.77
# Soft Clay
nDMaterial PressureIndependMultiYield 1 2 2.1 13000 65000 18. 0.1

# Define Elements
set gravX 0.0
set gravY 0.
set press 0.
set bulk_f 2.2e9
set n_por 0.333
set bulk [expr $bulk_f/$n_por]
element quadUP 1 3 1 2 4 1.0 1 $bulk 1. [expr 0.00003/9.81/1.] [expr 0.00003/9.81/1.]
$gravX $gravY
fix 1 1 1 0
fix 2 1 1 0
fix 3 0 0 1
fix 4 0 0 1
equalDOF 3 4 1 2

# Depending on the excitation one should change the Timeseriesini factor
set Timeseriesini "Constant -factor 50."

# pattern UniformExcitation 1 1 -accel $Timeseries;
pattern Plain 1 $Timeseriesini {
load 4 0. -.5 0
load 3 0. -.5 0
}

# Set material to elastic for gravity loading
updateMaterialStage -material 1 -stage 0

# GRAVITY APPLICATION (elastic behavior)
set gamma 1.5

# create the SOE, ConstraintHandler, Integrator, Algorithm and Numberer
integrator Newmark $gamma [expr pow($gamma+0.5, 2)/4] \
0.00 0.0 0.002 0.0
test NormDispIncr 1.0e-5 5 1;
constraints Transformation
algorithm Newton
numberer RCM
system ProfileSPD
analysis Transient
```

```
analyze 3 5.e5
puts "End of Elastic Phase of Gravity Application"

# Set material to elasto-plastic for the rest of the loading
updateMaterialStage -material 1 -stage 1
analyze 5 5.e5
puts "End of Gravity Application. Starting Dynamic Excitation..."

# rezero time
wipeAnalysis
setTime 0.0
set omega 1.
set pi 3.141593

# Calculate the period from the timeseries
set T [expr 2*$pi/$omega]
set Timeseries "Sine 0.0 50. $T -shift 0.0 -factor 20."
pattern Plain 2 $Timeseries {
load 4 1. 0. 0
load 3 1. 0. 0
}

set gamma 0.6
test NormDispIncr 1.0e-4 200 1;
integrator Newmark $gamma [expr pow($gamma+0.5, 2)/4] 0.00 0.0 0.002 0.0
constraints Transformation
algorithm KrylovNewton
numberer RCM
system ProfileSPD

# create the analysis object
analysis Transient

# Recorders
recorder Node -file output_disp.txt -time -dof 1 2 disp
recorder Node -file output_pore.txt -time -dof 3 vel
recorder Element -file stress_1.txt -time -eleRange 1 1 material 1 stress
recorder Element -file stress_2.txt -time -eleRange 1 1 material 2 stress
recorder Element -file stress_3.txt -time -eleRange 1 1 material 3 stress
recorder Element -file stress_4.txt -time -eleRange 1 1 material 4 stress
recorder Element -file strain_1.txt -time -eleRange 1 1 material 1 strain
recorder Element -file strain_2.txt -time -eleRange 1 1 material 2 strain
recorder Element -file strain_3.txt -time -eleRange 1 1 material 3 strain
recorder Element -file strain_4.txt -time -eleRange 1 1 material 4 strain

# perform analysis
set startT [clock seconds]
analyze 1500 0.01
set endT [clock seconds]
puts "Execution time: [expr $endT-$startT] seconds."
```

Appendix F

Laminar Drain Source Code

1. Class Definition

```
/* ****************************************************************** **
**    OpenSees - Open System for Earthquake Engineering Simulation    **
**          Pacific Earthquake Engineering Research Center            **
**                                                                    **
**                                                                    **
** (C) Copyright 1999, The Regents of the University of California    **
** All Rights Reserved.                                               **
**                                                                    **
** Commercial use of this program without express permission of the  **
** University of California, Berkeley, is strictly prohibited.  See   **
** file 'COPYRIGHT'  in main directory for information on usage and   **
** redistribution,  and for a DISCLAIMER OF ALL WARRANTIES.           **
**                                                                    **
** Developed by:                                                      **
**   Frank McKenna (fmckenna@ce.berkeley.edu)                         **
**   Gregory L. Fenves (fenves@ce.berkeley.edu)                       **
**   Filip C. Filippou (filippou@ce.berkeley.edu)                     **
**                                                                    **
** ****************************************************************** */

// $Revision: 1.00 $
// $Date: 2008/07/18 18:05:53 $
// $Source: /usr/local/cvs/OpenSees/SRC/element/pipe/Pipe.h,v $


// Written: Antonios Vytiniotis
// Created: 07/08
// Revision: A
//
// Description: This file contains the definition for the Pipelin2. A
Pipelin2 object
// provides the abstraction of the small deformation bar element plus
predicts the
// uncoupled pore pressure change according to Darcy Weisbach equation for
laminar flow.
// Each pipe object is associated with a material object dealing with the
axial compressibility
// of the drain. This Pipelin2 element will work in 2d problems in a 3DOF
domain.
//
// What: "@(#) Pipelin2.h, revA"


#ifndef Pipelin2_h
#define Pipelin2_h


#include <Element.h>
#include <Matrix.h>
```

```
class Node;
class Channel;
class UniaxialMaterial;


class Pipelin2:public Element {
public:
      //constructors
      Pipelin2 (int tag, int Nd1, int Nd2, UniaxialMaterial &theMaterial,
double A, double C_3, double Grav=0.0);
      Pipelin2();
      //destructor
      ~Pipelin2();

      //public methods to obtain information about dof & connectivity
      int getNumExternalNodes(void) const;
      const ID &getExternalNodes(void);
      int getNumDOF(void);
    Node **getNodePtrs(void);

      //public methods to set the state of the element
      void setDomain(Domain *theDomain);
      int commitState(void);
      int revertToLastCommit(void);
      int revertToStart(void);
      int update(void);

      //public methods to obtain stiffness, mass, damping, and residual
information
      const Matrix &getTangentStiff(void);
      const Matrix &getInitialStiff(void);
      const Matrix &getDamp(void);
      const Matrix &getMass(void);

      void zeroLoad(void);
    int addLoad(ElementalLoad *theLoad, double loadFactor);
    int addInertiaLoadToUnbalance(const Vector &accel);
      const Vector &getResistingForce(void);
      const Vector &getResistingForceIncInertia(void);

      //public methods for output
      int sendSelf(int commitTag, Channel &theChannel);
      int recvSelf(int commitTag, Channel &theChannel, FEM_ObjectBroker
&theBroker);
      int displaySelf(Renderer &theViewer, int displayMode, float fact);
      void Print(OPS_Stream &s, int flag=0);
      Response *setResponse(const char **argv, int argc, OPS_Stream &s);
      int getResponse(int responseID, Information &eleInformation);

//protected:
private:
      //private member function - only availabe to objects of the class
      double computeCurrentStrain(void) const;

      //private attributes - a copy for each object of the class
      UniaxialMaterial *theMaterial; //pointer to a material
      ID externalNodes; // contains the id's of end nodes
      Matrix trans; //hold the transformation matrix
//    Vector *theLoad;     // pointer to the load vector P

      double L; //length of Pipe based on undeformed configuration
      double C_3;
      double A;
      double d_y_class;
      int eletag;
      double Gamma;     //weight per unit volume
      Node *end1Ptr, *end2Ptr;  //two pointer to the trusses nodes
      Node *theNodes[2];                  //two pointer to the trusses nodes
in a matrix form  (AV)

      //private class attribute
      static Matrix trussK;
      static Matrix trussD;
      static Matrix trussM;
      static Vector trussR;

};
#endif
```

2. Class Implementation

```
/* ****************************************************************** **
**    OpenSees - Open System for Earthquake Engineering Simulation    **
**          Pacific Earthquake Engineering Research Center            **
**                                                                    **
**                                                                    **
** (C) Copyright 1999, The Regents of the University of California    **
** All Rights Reserved.                                               **
**                                                                    **
** Commercial use of this program without express permission of the  **
** University of California, Berkeley, is strictly prohibited.  See  **
** file 'COPYRIGHT'  in main directory for information on usage and   **
** redistribution,  and for a DISCLAIMER OF ALL WARRANTIES.           **
**                                                                    **
** Developed by:                                                      **
**   Frank McKenna (fmckenna@ce.berkeley.edu)                         **
**   Gregory L. Fenves (fenves@ce.berkeley.edu)                       **
**   Filip C. Filippou (filippou@ce.berkeley.edu)                     **
**                                                                    **
** ****************************************************************** */

// $Revision: 1.00 $
// $Date: 2008/07/18 18:05:53 $
// $Source: /usr/local/cvs/OpenSees/SRC/element/Pipe/Pipelin2.cpp,v $


// Written: Antonios Vytiniotis
// Created: 07/08
// Revision: A
//
// Description: This file contains the implementation for the Pipelin2 class.
//
```

```
#include "Pipelin2.h"
#include <Information.h>
#include <Parameter.h>

#include <Domain.h>
#include <Node.h>
#include <Channel.h>
#include <FEM_ObjectBroker.h>
#include <UniaxialMaterial.h>
#include <Renderer.h>

#include <math.h>
#include <stdlib.h>
#include <string.h>

#include <ElementResponse.h>

#include <Matrix.h>
#include <Vector.h>

#include <ElasticMaterial.h>

// initial the class wide variables
Matrix Pipelin2::trussK(6,6);
Matrix Pipelin2::trussM(6,6);
Matrix Pipelin2::trussD(6,6);
Vector Pipelin2::trussR(6);


Pipelin2::Pipelin2(int tag,
                         int Nd1, int Nd2,
                         UniaxialMaterial &theMat,
                         double a, double c3, double g)
                         :Element(tag,ELE_TAG_Pipelin2),
                         theMaterial(0),
                         externalNodes(2),
                         trans(1,4),L(0.0), A(a), C_3(c3), Gamma(g),
end1Ptr(0), end2Ptr(0), eletag(tag),
                         d_y_class(0.0)
{
      //create a copy of the material object
      theMaterial=theMat.getCopy();


      //fill in the ID containing external node info with node id's
      externalNodes(0)=Nd1;
      externalNodes(1)=Nd2;
      for (int i=0; i<2; i++)
    theNodes[i] = 0;
      trussR.Zero();
}

//constructor which should be invoked by an FE_ObjectBroker only
Pipelin2::Pipelin2()
                  :Element(0,ELE_TAG_Pipelin2),
                  theMaterial(0),
                  externalNodes(2),
                  trans(1,4), L(0.0), A(0.0), C_3(0.0), Gamma(0.0),
end1Ptr(0), end2Ptr(0),
                  d_y_class(0.0)
            {
            for (int i=0; i<2; i++)
            theNodes[i] = 0;
            }

            Pipelin2::~Pipelin2()
            {
                  if (theMaterial !=0)
                        delete theMaterial;
            }


            int Pipelin2::getNumExternalNodes(void) const
            {
                  return 2;
            }

            const ID &
                  Pipelin2::getExternalNodes(void)
            {
                  return externalNodes;
            }

            int
                  Pipelin2::getNumDOF(void){
                        return 6;
            }

            Node **
            Pipelin2::getNodePtrs(void)
            {
            return theNodes;
            }

            void
                  Pipelin2::setDomain(Domain *theDomain)
            {
                  //first ensure nodes exist in Domain and set the node
pointers
                  int Nd1 =externalNodes(0);
                  int Nd2 =externalNodes(1);
                  end1Ptr =theDomain->getNode(Nd1);
                  end2Ptr =theDomain->getNode(Nd2);
                  theNodes[0] = theDomain->getNode(Nd1);
                  theNodes[1] = theDomain->getNode(Nd2);

                  if (theNodes[0]==0)
                        return;
                  if (theNodes[1]==0)
                        return;

                  // call the DomainComponent class method
                  this->DomainComponent::setDomain(theDomain);
```

```
                    //ensure connected nodes have corrent number of dof's
                    int dofNd1=theNodes[0]->getNumberDOF();
                    int dofNd2=theNodes[1]->getNumberDOF();
                    if ((dofNd1 !=3) || (dofNd2 !=3))
                          return; //don't go any further otherwise segmentation
fault

                    //now determine the length & transformation matrix
                    const Vector &end1Crd=theNodes[0]->getCrds();
                    const Vector &end2Crd=theNodes[1]->getCrds();

                    double dx= end2Crd(0)-end1Crd(0);
                    double dy= end2Crd(1)-end1Crd(1);

                    d_y_class=dy;
                    L=sqrt(dx*dx+dy*dy);

                    if (L==0.0)
                          return;

                    double cs=dx/L;
                    double sn=dy/L;
                    trans(0,0)=-cs;
                    trans(0,1)=-sn;
                    trans(0,2)= cs;
                    trans(0,3)= sn;

//                  // determine the nodal mass for lumped mass approach
//                  M=M*A*L/2; //M was set to rho by the constructor
              }


              int
                    Pipelin2::commitState()
              {
                    return theMaterial->commitState();
              }

              int
                    Pipelin2::revertToLastCommit()
              {
                    return theMaterial->revertToLastCommit();
              }

              int
                    Pipelin2::revertToStart()
              {
                    return theMaterial->revertToStart();
              }

              int
                    Pipelin2::update()
              {
                    //determine the current strain given trial displacements at
nodes
                    double strain=this->computeCurrentStrain();
```

```
                //set the strain in the materials
                theMaterial->setTrialStrain(strain);

                return 0;
        }


        const Matrix &
                Pipelin2::getTangentStiff(void)
        {
                if (L==0) {//if length ==zero - we zero and return
                        trussK.Zero();
                        return trussK;
                }

                //get the current E from the material for the strain that
was set
                // at the material when the update() method was invoked

                double E = theMaterial->getTangent();

                //form the tangent stiffness matrix
                Matrix K_temp(4,4);
                K_temp =trans^trans; //This is a temporary matrix
containing the truss stiffness parameters
                K_temp *=A*E/L;

//              trussK.Zero();
                // Truss stiffness components:
                trussK(0,0)=K_temp(0,0);
                trussK(1,0)=K_temp(1,0);
                trussK(0,1)=K_temp(0,1);
                trussK(1,1)=K_temp(1,1);
                trussK(3,3)=K_temp(2,2);
                trussK(4,3)=K_temp(3,2);
                trussK(3,4)=K_temp(2,3);
                trussK(4,4)=K_temp(3,3);
                trussK(2,2)=0.;
                trussK(5,5)=0.;
                trussK(5,2)=0.;
                trussK(2,5)=0.;

                return trussK;
        }
        const Matrix &
                Pipelin2::getInitialStiff(void)
        {
                if (L==0) {
                        trussK.Zero();
                        return trussK;
                }

                //get the current strain from the material
                double strain = theMaterial->getStrain();

                //get the current stress from the material
                double stress = theMaterial->getStress();
```

```
			//compute the tangent
			double E=stress/strain;

			//form the tangent stiffness matrix
			Matrix K_temp(4,4);
			K_temp =trans^trans; //This is a temporary matrix
containing the truss stiffness parameters
			K_temp *=A*E/L;

//			trussK.Zero();
			// Truss stiffness components:
			trussK(0,0)=K_temp(0,0);
			trussK(1,0)=K_temp(1,0);
			trussK(0,1)=K_temp(0,1);
			trussK(1,1)=K_temp(1,1);
			trussK(3,3)=K_temp(2,2);
			trussK(4,3)=K_temp(3,2);
			trussK(3,4)=K_temp(2,3);
			trussK(4,4)=K_temp(3,3);
			trussK(2,2)=0.;
			trussK(5,5)=0.;
			trussK(5,2)=0.;
			trussK(2,5)=0.;

//			opserr << "Componenents of the matrix K11" << trussK(3,3)
<<endln;
			return trussK;
		}

		const Matrix &
			Pipelin2::getDamp(void)
		{
			//No damping associated with this type of element
			trussD.Zero();
			trussD(2,2)=-C_3/L;
			trussD(5,5)=-C_3/L;
			trussD(5,2)=C_3/L;
			trussD(2,5)=C_3/L;
			double deleteme=C_3/L;
			double deleteme2=C_3/L;
			return trussD;
		}

		const Matrix &
			Pipelin2::getMass(void)
		{
			if (L==0){
				trussM.Zero();
				return trussM;
			}

//			At this point we have zero lumped mass
			trussM.Zero();
			return trussM;
		}
```

```
            void
                  Pipelin2::zeroLoad(void)
            {
                  //does nothing - no element load associated with this
object
            }

            int
                  Pipelin2::addLoad(ElementalLoad *theLoad, double
loadFactor)
            {
            opserr <<"MyTruss::addLoad - load type unknown for truss with
tag: " << this->getTag() << endln;

            return -1;
            }

            int
                  Pipelin2::addInertiaLoadToUnbalance(const Vector &accel)
            {
                  return 0;
            }

            const Vector &
                  Pipelin2::getResistingForce()
            {
                  if (L==0) {//if length ==zero - zero and return
                        trussR.Zero();
                        return trussR;
                  }
                  // R=Ku-Pext
                  //force =F*transformation
                  double force = A* theMaterial->getStress();
                  trussR(0)= trans(0,0)*force;
                  trussR(1)= trans(0,1)*force;
                  trussR(3)= trans(0,2)*force;
                  trussR(4)= trans(0,3)*force;

                  const Vector &vel1 = theNodes[0]->getTrialVel();
            const Vector &vel2 = theNodes[1]->getTrialVel();

//          This is the linear element with total disp (no need for state
params)
//                Domain::update(double a, double b);
//                double dt=;
//
      trussR(2)=trussD(2,2)*vel1(2)+trussD(2,5)*vel2(2)+trussD(2,2)*d_y_class
*Gamma;
//                trussR(5)=trussD(5,2)*vel1(2)+trussD(5,5)*vel2(2)-
trussD(5,5)*d_y_class*Gamma;
                  trussR(2)=-C_3/L*vel1(2)+C_3/L*vel2(2)-
C_3/L*d_y_class*Gamma;
                  trussR(5)=C_3/L*vel1(2)-
C_3/L*vel2(2)+C_3/L*d_y_class*Gamma;
                  return trussR;
            }
            const Vector &
                  Pipelin2::getResistingForceIncInertia()
            {

                  this->getResistingForce();

                  //No inertia is included in the in this element formulation
                  return trussR;
            }


            int
                  Pipelin2::sendSelf (int commitTag, Channel &theChannel)
            {
                  int dataTag=this->getDbTag();

                  // Pipelin2 packs it's data into a Vector and sends this to
theChannel
                  //along with it's dbTag and the commitTag passed in the
arguments

                  Vector data(6);
                  data(0)= this->getTag();
                  data(1)=A;
                  data(4)=C_3;
                  data(5)=Gamma;
                  data(2)=theMaterial->getClassTag();
                  int matDbTag=theMaterial->getDbTag();
                  if (matDbTag==0) {
                        matDbTag =theChannel.getDbTag();
                        if (matDbTag !=0)
                              theMaterial->setDbTag(matDbTag);
                  }
                  data(3)=matDbTag;

                  theChannel.sendVector (dataTag, commitTag, data);

                  theChannel.sendID(dataTag, commitTag, externalNodes);

                  theMaterial->sendSelf(commitTag, theChannel);

                  return 0;
            }
            int
                  Pipelin2::recvSelf(int commitTag, Channel &theChannel,
FEM_ObjectBroker &theBroker)
            {
                  int dataTag= this->getDbTag();
                  Vector data(6);
                  theChannel.recvVector(dataTag, commitTag, data);

                  this->setTag((int)data(0));
                  A=data(1);
                  C_3=data(4);
                  Gamma=data(5);
                  theChannel.recvID(dataTag, commitTag, externalNodes);
```

```
        int matClass=data(2);
        int matDb = data(3);
        theMaterial= theBroker.getNewUniaxialMaterial(matClass);

        theMaterial->setDbTag(matDb);
        theMaterial->recvSelf(commitTag, theChannel, theBroker);

        return 0;
}

int
        Pipelin2::displaySelf(Renderer &theViewer, int displayMode,
float fact)
{
        const Vector &end1Crd= end1Ptr->getCrds();
        const Vector &end2Crd= end2Ptr->getCrds();
        const Vector &end1Disp=end1Ptr->getDisp();
        const Vector &end2Disp=end2Ptr->getDisp();

        Vector v1(3);
        Vector v2(3);
        for (int i=0; i<2;i++) {
                v1(i)=end1Crd(i)+end1Disp(i)*fact;
                v2(i)=end2Crd(i)+end2Disp(i)*fact;
        }
        if (displayMode==3) {
                //use the strain as the drawing measure
                double strain = theMaterial->getStrain();
                return theViewer.drawLine(v1, v2, strain, strain);
        }
        else if (displayMode==2){
                //otherwise use the material stress
                double stress =A*theMaterial->getStress();
                return theViewer.drawLine(v1,v2,stress, stress);
        }
        else{
                //use the axial force
                double force = A*theMaterial->getStress();
                return theViewer.drawLine(v1,v2,force,force);
        }
}

void
        Pipelin2::Print(OPS_Stream &s, int flag)
{
        //compute the strain and axial force in the member
        double strain, force;
        if (L==0.0) {
                strain=0;
                force=0.0;
        }
        else{
                strain = theMaterial->getStrain();
                force=A*theMaterial->getStress();
        }
        trussR(0)= trans(0,0)*force;
        trussR(1)= trans(0,1)*force;
```

```
		trussR(3)= trans(0,2)*force;
		trussR(4)= trans(0,3)*force;

		const Vector &vel1 = theNodes[0]->getVel();
	const Vector &vel2 = theNodes[1]->getVel();
		trussR(2)=-C_3/L*vel1(2)+C_3/L*vel2(2)-
C_3/L*d_y_class*Gamma;
		trussR(5)=C_3/L*vel1(2)-
C_3/L*vel2(2)+C_3/L*d_y_class*Gamma;

		if (flag==0) {//print everythin
			s<< "Element: " <<this->getTag();
			s<< " type: My Truss iNode: "<< externalNodes(0);
			s<< " jNode: "<<externalNodes(1);
			s<< " Area: "<< A;
			if (Gamma!=0) s << "Gamma: "<<Gamma;

			s<< " \n\t strain: " <<strain;
			s<< " axial load: " <<force;
			s<< " \n\t unbalanced load: " <<trussR;
			s<< " \t Material: " << *theMaterial;
			s<< endln;
		} else if (flag==1) {//just print ele id, strain and force
			s<< this->getTag() << "  " <<strain << "  " << force
<<endln;
		}
	}


	Response *
		Pipelin2::setResponse(const char **argv, int argc ,
OPS_Stream &s)
	{
		// we compare arg(0) for known response types for the Truss

		//axial force

		if(strcmp(argv[0], "axialForce")==0)
			return new ElementResponse(this, 1, 0.0);

		//a material quantity
		else if (strcmp(argv[0], "material")==0)
			return theMaterial->setResponse(&argv[1], argc-1, s);
		else
			return 0;
	}

	int
		Pipelin2::getResponse(int responseID, Information
&eleInformation)
	{
		switch (responseID){
			case -1:
				return -1;

			case 1:
```

```
                                return eleInformation.setDouble (A*theMaterial-
>getStress());
                          default:
                                return 0;
                    }
              }

              double
                    Pipelin2::computeCurrentStrain(void) const
              {
                    //determine the strain
                    const Vector &disp1=end1Ptr->getTrialDisp();
                    const Vector &disp2=end2Ptr->getTrialDisp();

                    double dLength=0.0;
                    for (int i=0;i<2;i++)
                          dLength -= (disp2(i)-disp1(i))*trans(0,i);

                    double strain =dLength/L;

                    return strain;
              }
```

3. Tcl/tk command interpreter

```
/* ****************************************************************** **
**    OpenSees - Open System for Earthquake Engineering Simulation    **
**          Pacific Earthquake Engineering Research Center            **
**                                                                    **
**                                                                    **
** (C) Copyright 1999, The Regents of the University of California    **
** All Rights Reserved.                                               **
**                                                                    **
** Commercial use of this program without express permission of the  **
** University of California, Berkeley, is strictly prohibited.  See   **
** file 'COPYRIGHT'  in main directory for information on usage and   **
** redistribution,  and for a DISCLAIMER OF ALL WARRANTIES.           **
**                                                                    **
** Developed by:                                                      **
**   Frank McKenna (fmckenna@ce.berkeley.edu)                         **
**   Gregory L. Fenves (fenves@ce.berkeley.edu)                       **
**   Filip C. Filippou (filippou@ce.berkeley.edu)                     **
**                                                                    **
** ****************************************************************** */

// $Revision: 1. $
// $Date: 2008/07/20 19:20:46 $
// $Source: /usr/local/cvs/OpenSees/SRC/element/pipe/TclPipelin2Command.cpp,v
$


// File: ~/element/TclPipelin2Command.C
//
// Written: avytin
// Created: 09/08
// Revision: A
```

```
//
// Description: This file contains the implementation of the
TclModelBuilder_Pipelin2()
// command.
//
// What: "@(#) TclModelBuilder.C, revA"

#include <stdlib.h>
#include <string.h>
#include <Domain.h>

#include "Pipelin2.h"
#include <TrussSection.h>
#include <TclModelBuilder.h>
#include <CorotTruss.h>
#include <CorotTrussSection.h>

extern void printCommand(int argc, TCL_Char **argv);

int
TclModelBuilder_Pipelin2(ClientData clientData, Tcl_Interp *interp, int argc,
                                    TCL_Char **argv, Domain*theTclDomain,
TclModelBuilder *theTclBuilder,
                                    int eleArgStart){
                                          //make sure at least one other
argument to contain type of system
                                          if (argc!=8 && argc!=9){
                                                interp->result = "WARNING bad
command - Pipelin2 eleId iNode jNode matID Area c_3 Gamma";
                                                return TCL_ERROR;
                                          }

                                          //get the id, x_loc, y_loc
                                          int trussId, iNode, jNode, matID;
                                          double A, C_3, Gamma=0.0;
                                          if (Tcl_GetInt(interp,argv[2],
&trussId)!= TCL_OK){
                                                interp->result = "WARNING
invalid eleId - Pipelin2 eleId iNode jNode matID Area c_3 Gamma";
                                                return TCL_ERROR;
                                          }

                                          if (Tcl_GetInt(interp, argv[3],
&iNode) != TCL_OK) {
                                                interp->result = "WARNING
invalid iNode - Pipelin2 eleId iNode jNode matID Area c_3 Gamma";
                                                return TCL_ERROR;
                                          }

                                          if (Tcl_GetInt(interp, argv[4],
&jNode) != TCL_OK) {
                                                interp->result = "WARNING
invalid jNode - Pipelin2 eleId iNode jNode matID Area c_3 Gamma";
                                                return TCL_ERROR;
                                          }
                                          if (Tcl_GetInt(interp, argv[5],
&matID) != TCL_OK) {
                                                interp->result = "WARNING
invalid matID - Pipelin2 eleId iNode jNode matID Area c_3 Gamma";
                                                return TCL_ERROR;
                                          }

                                          if (Tcl_GetDouble(interp, argv[6],
&A) != TCL_OK) {
                                                interp->result = "WARNING
invalid Area - Pipelin2 eleId iNode jNode matID Area c_3 Gamma";
                                                return TCL_ERROR;
                                          }

                                          if (Tcl_GetDouble(interp, argv[7],
&C_3) != TCL_OK) {
                                                interp->result = "WARNING
invalid C_3 - Pipelin2 eleId iNode jNode matID Area c_3 Gamma";
                                                return TCL_ERROR;
                                          }

                                          if (Tcl_GetDouble(interp, argv[8],
&Gamma) != TCL_OK) {
                                                interp->result = "WARNING
invalid C_3 - Pipelin2 eleId iNode jNode matID Area c_3 gamma";
                                                return TCL_ERROR;
                                          }
                                          UniaxialMaterial *theMaterial =
theTclBuilder->getUniaxialMaterial(matID);

                                          if (theMaterial ==0) {
                                                opserr << "WARNING
TclPipelin2 - Pipelin2  - no Material found with tag ";
                                                opserr << matID << endln;
                                                return TCL_ERROR;
                                          }

                                          //now create the truss and add it
to the domain
                                          Element *theTruss = 0;
                                          theTruss=new
Pipelin2(trussId,iNode,jNode,*theMaterial,A,C_3,Gamma);
                                          if (theTruss==0) {
                                                opserr << "WARNING
TclPipelin2 - Pipelin2 - ran out of memory for node ";
                                                opserr << trussId << endln;
                                                return TCL_ERROR;
                                          }

                                          if (theTclDomain-
>addElement(theTruss)==false) {
                                                delete theTruss;
                                                opserr << "WARNING
TclPipelin2 - Pipelin2 - could not add Pipelin2 to the domain";
                                                opserr << trussId << endln;
                                                return TCL_ERROR;
                                          }
```

```
                                                 //Everything is OK

                                                 return TCL_OK;
}
```

Appendix G

Fully Turbulent Flow Drains Source Code

1. Class Implementation

```
/* ****************************************************************** **
**    OpenSees - Open System for Earthquake Engineering Simulation    **
**          Pacific Earthquake Engineering Research Center            **
**                                                                    **
**                                                                    **
** (C) Copyright 1999, The Regents of the University of California    **
** All Rights Reserved.                                               **
**                                                                    **
** Commercial use of this program without express permission of the  **
** University of California, Berkeley, is strictly prohibited.  See   **
** file 'COPYRIGHT'  in main directory for information on usage and   **
** redistribution,  and for a DISCLAIMER OF ALL WARRANTIES.           **
**                                                                    **
** Developed by:                                                      **
**   Frank McKenna (fmckenna@ce.berkeley.edu)                         **
**   Gregory L. Fenves (fenves@ce.berkeley.edu)                       **
**   Filip C. Filippou (filippou@ce.berkeley.edu)                     **
**                                                                    **
** ****************************************************************** */

// $Revision: 1.00 $
// $Date: 2008/07/18 18:05:53 $
// $Source: /usr/local/cvs/OpenSees/SRC/element/pipe/Pipe.h,v $


// Written: Antonios Vytiniotis
// Created: 07/08
// Revision: A
//
// Description: This file contains the definition for the Pipe3. A Pipe3
object
// provides the abstraction of the small deformation bar element plus
predicts the
// uncoupled pore pressure change according to Darcy Weisbach equation. Each
pipe
// object is associated with a material object dealing with the axial
compressibility
// of the drain. This Pipe3 element will work in 2d problems in a 3DOF
domain.
//
// What: "@(#) Pipe3.h, revA"


#ifndef Pipe3_h
#define Pipe3_h


#include <Element.h>
#include <Matrix.h>
```

```
class Node;
class Channel;
class UniaxialMaterial;

//#define ELE_TAG_MyTruss 4002

// This is a trial implementation of a simle 2-d Pipe3 element
class Pipe3:public Element {
public:
      //constructors
      Pipe3 (int tag, int Nd1, int Nd2, UniaxialMaterial &theMaterial, double
A, double C_3, double Gamma=0.0, double D_C=0.0);
      Pipe3();
      //destructor
      ~Pipe3();

      //public methods to obtain information about dof & connectivity
      int getNumExternalNodes(void) const;
      const ID &getExternalNodes(void);
      int getNumDOF(void);
    Node **getNodePtrs(void);

      //public methods to set the state of the element
      void setDomain(Domain *theDomain);
      int commitState(void);
      int revertToLastCommit(void);
      int revertToStart(void);
      int update(void);

      //public methods to obtain stiffness, mass, damping, and residual
information
      const Matrix &getTangentStiff(void);
      const Matrix &getInitialStiff(void);
      const Matrix &getDamp(void);
      const Matrix &getMass(void);

      void zeroLoad(void);
    int addLoad(ElementalLoad *theLoad, double loadFactor);
    int addInertiaLoadToUnbalance(const Vector &accel);
      const Vector &getResistingForce(void);
      const Vector &getResistingForceIncInertia(void);

      //public methods for output
      int sendSelf(int commitTag, Channel &theChannel);
      int recvSelf(int commitTag, Channel &theChannel, FEM_ObjectBroker
&theBroker);
      int displaySelf(Renderer &theViewer, int displayMode, float fact);
      void Print(OPS_Stream &s, int flag=0);
      Response *setResponse(const char **argv, int argc, OPS_Stream &s);
      int getResponse(int responseID, Information &eleInformation);

//protected:
private:
      //private member function - only availabe to objects of the class
      double computeCurrentStrain(void) const;

      //private attributes - a copy for each object of the class
      UniaxialMaterial *theMaterial; //pointer to a material
      ID externalNodes; // contains the id's of end nodes
      Matrix trans; //hold the transformation matrix
//    Vector *theLoad;     // pointer to the load vector P

      double L; //length of Pipe3 based on undeformed configuration
      double C_3;
      double A;
      double D_C;
      double d_y_class;
      int eletag;
      double Gamma;     //weight per unit volume
      Node *end1Ptr, *end2Ptr;  //two pointer to the trusses nodes
      Node *theNodes[2];                  //two pointer to the trusses nodes
in a matrix form  (AV)


      //private class attribute
      static Matrix trussK;
      static Matrix trussD;
      static Matrix trussM;
      static Vector trussR;

};
#endif
```

2. Class Definition

```
/* ****************************************************************** **
**    OpenSees - Open System for Earthquake Engineering Simulation    **
**          Pacific Earthquake Engineering Research Center            **
**                                                                    **
**                                                                    **
** (C) Copyright 1999, The Regents of the University of California    **
** All Rights Reserved.                                               **
**                                                                    **
** Commercial use of this program without express permission of the   **
** University of California, Berkeley, is strictly prohibited.  See   **
** file 'COPYRIGHT'  in main directory for information on usage and   **
** redistribution,  and for a DISCLAIMER OF ALL WARRANTIES.           **
**                                                                    **
** Developed by:                                                      **
**   Frank McKenna (fmckenna@ce.berkeley.edu)                         **
**   Gregory L. Fenves (fenves@ce.berkeley.edu)                       **
**   Filip C. Filippou (filippou@ce.berkeley.edu)                     **
**                                                                    **
** ****************************************************************** */

// $Revision: 1.00 $
// $Date: 2008/07/18 18:05:53 $
// $Source: /usr/local/cvs/OpenSees/SRC/element/Pipe/Pipe3.cpp,v $


// Written: Antonios Vytiniotis
// Created: 07/08
// Revision: A
```

```
//
// Description: This file contains the implementation for the Pipe3 class.
// The current implementation does not work very well for Newton and
EnergyIncr
// (Flow does not go to zero at end of analysis)
// It works well for DispIncr and KrylovNewton Analysis eventhough some
leakage
// seem to be happening after the end of the analysis. More verification is
needed
// in order to see the effect of the d_c parameter and incrementation and
integration
// schemes. Also, minor controls might be needed to calculate R at very low
i.
//

#include "Pipe3.h"
#include <Information.h>
#include <Parameter.h>

#include <Domain.h>
#include <Node.h>
#include <Channel.h>
#include <FEM_ObjectBroker.h>
#include <UniaxialMaterial.h>
#include <Renderer.h>

#include <math.h>
#include <stdlib.h>
#include <string.h>

#include <ElementResponse.h>

#include <Matrix.h>
#include <Vector.h>

#include <ElasticMaterial.h>

// initial the class wide variables
Matrix Pipe3::trussK(6,6);
Matrix Pipe3::trussM(6,6);
Matrix Pipe3::trussD(6,6);
Vector Pipe3::trussR(6);

Pipe3::Pipe3(int tag,
                         int Nd1, int Nd2,
                         UniaxialMaterial &theMat,
                         double a, double c3, double g, double dc)
                         :Element(tag,ELE_TAG_Pipe3),
                         theMaterial(0),
                         externalNodes(2),/*theLoad(0),*/
                         trans(1,4),L(0.0), A(a), C_3(c3), Gamma(g), D_C(dc),
end1Ptr(0), end2Ptr(0), eletag(tag),
                         d_y_class(0.0)
{
      //create a copy of the material object
      theMaterial=theMat.getCopy();
      //fill in the ID containing external node info with node id's
      externalNodes(0)=Nd1;
      externalNodes(1)=Nd2;
      for (int i=0; i<2; i++)
    theNodes[i] = 0;
}

//constructor which should be invoked by an FE_ObjectBroker only
Pipe3::Pipe3()
                  :Element(0,ELE_TAG_Pipe3),
                  theMaterial(0),
                  externalNodes(2),/*theLoad(0),*/
                  trans(1,4), L(0.0), A(0.0), C_3(0.0), Gamma(0.0),
D_C(0.0),end1Ptr(0), end2Ptr(0),
                  d_y_class(0.0)
{
            for (int i=0; i<2; i++)
            theNodes[i] = 0;
            }

            Pipe3::~Pipe3()
            {
                  if (theMaterial !=0)
                        delete theMaterial;
            }


            int Pipe3::getNumExternalNodes(void) const
            {
                  return 2;
            }

            const ID &
                  Pipe3::getExternalNodes(void)
            {
                  return externalNodes;
            }

            int
                  Pipe3::getNumDOF(void){
                        return 6;
            }

            Node **
            Pipe3::getNodePtrs(void)
            {
            return theNodes;
            }

            void
                  Pipe3::setDomain(Domain *theDomain)
            {
                  //first ensure nodes exist in Domain and set the node
pointers
                  int Nd1 =externalNodes(0);
                  int Nd2 =externalNodes(1);
                  end1Ptr =theDomain->getNode(Nd1);
```

```
                                end2Ptr =theDomain->getNode(Nd2);
                                theNodes[0] = theDomain->getNode(Nd1);
                                theNodes[1] = theDomain->getNode(Nd2);

                                if (theNodes[0]==0)
                                      return;
                                if (theNodes[1]==0)
                                      return;

                                // call the DomainComponent class method
                                this->DomainComponent::setDomain(theDomain);

                                //ensure connected nodes have corrent number of dof's
                                int dofNd1=theNodes[0]->getNumberDOF();
                                int dofNd2=theNodes[1]->getNumberDOF();
                                if ((dofNd1 !=3) || (dofNd2 !=3))
                                      return; //don't go any further otherwise segmentation
fault

                                //now determine the length & transformation matrix
                                const Vector &end1Crd=theNodes[0]->getCrds();
                                const Vector &end2Crd=theNodes[1]->getCrds();

                                double dx= end2Crd(0)-end1Crd(0);
                                double dy= end2Crd(1)-end1Crd(1);

                                d_y_class=dy;
                                L=sqrt(dx*dx+dy*dy);

                                if (L==0.0)
                                      return;

                                double cs=dx/L;
                                double sn=dy/L;
                                trans(0,0)=-cs;
                                trans(0,1)=-sn;
                                trans(0,2)= cs;
                                trans(0,3)= sn;

//                              // determine the nodal mass for lumped mass approach
//                              M=M*A*L/2; //M was set to rho by the constructor
                        }


                        int
                                Pipe3::commitState()
                        {
                                return theMaterial->commitState();
                        }

                        int
                                Pipe3::revertToLastCommit()
                        {
                                return theMaterial->revertToLastCommit();
                        }

                        int
```

```
                Pipe3::revertToStart()
        {
                return theMaterial->revertToStart();
        }

        int
                Pipe3::update()
        {
                //determine the current strain given trial displacements at
nodes
                double strain=this->computeCurrentStrain();

                //set the strain in the materials
                theMaterial->setTrialStrain(strain);

                return 0;
        }


        const Matrix &
                Pipe3::getTangentStiff(void)
        {
                if (L==0) {//if length ==zero - we zero and return
                        trussK.Zero();
                        return trussK;
                }

                //get the current E from the material for the strain that
was set
                // at the material when the update() method was invoked

                double E = theMaterial->getTangent();

                //form the tangent stiffness matrix
                Matrix K_temp(4,4);
                K_temp =trans^trans; //This is a temporary matrix
containing the truss stiffness parameters
                K_temp *=A*E/L;

//              trussK.Zero();
                // Truss stiffness components:
                trussK(0,0)=K_temp(0,0);
                trussK(1,0)=K_temp(1,0);
                trussK(0,1)=K_temp(0,1);
                trussK(1,1)=K_temp(1,1);
                trussK(3,3)=K_temp(2,2);
                trussK(4,3)=K_temp(3,2);
                trussK(3,4)=K_temp(2,3);
                trussK(4,4)=K_temp(3,3);

                trussK(2,2)=0.0;
                trussK(5,5)=0.0;
                trussK(5,2)=0.0;
                trussK(2,5)=0.0;

                return trussK;
        }
```

```
            const Matrix &
                  Pipe3::getInitialStiff(void)
            {
                  if (L==0) {
                        trussK.Zero();
                        return trussK;
                  }

                  //get the current strain from the material
                  double strain = theMaterial->getStrain();

                  //get the current stress from the material
                  double stress = theMaterial->getStress();

                  //compute the tangent
                  double E=stress/strain;

                  //form the tangent stiffness matrix
                  Matrix K_temp(4,4);
                  K_temp =trans^trans; //This is a temporary matrix
containing the truss stiffness parameters
                  K_temp *=A*E/L;

//                trussK.Zero();
                  // Truss stiffness components:
                  trussK(0,0)=K_temp(0,0);
                  trussK(1,0)=K_temp(1,0);
                  trussK(0,1)=K_temp(0,1);
                  trussK(1,1)=K_temp(1,1);
                  trussK(3,3)=K_temp(2,2);
                  trussK(4,3)=K_temp(3,2);
                  trussK(3,4)=K_temp(2,3);
                  trussK(4,4)=K_temp(3,3);

                  trussK(2,2)=0.0;
                  trussK(5,5)=0.0;
                  trussK(5,2)=0.0;
                  trussK(2,5)=0.0;

                  return trussK;
            }

            const Matrix &
                  Pipe3::getDamp(void)
            {
                  //No damping associated with this type of element
                  trussD.Zero();

                  // Darcy-Weisbach components
                  const Vector &vel1 = end1Ptr->getTrialVel();
                  const Vector &vel2 = end2Ptr->getTrialVel();
//                const Vector &disp1 = theNodes[0]->getIncrDisp();
//                const Vector &disp2 = theNodes[1]->getIncrDisp();

                  double i_a;
                  i_a=vel2(2)/L-vel1(2)/L+d_y_class*Gamma/L;  //Element
hydraulic gradient
```

```
//                  Find the sign of the gradient:
                    double sign_i_a=1;
                    if (i_a<0)
                          sign_i_a=-1;
//Control for singularities in the incrementaly linearized equation (can be
changed)
                    if (sign_i_a*i_a<D_C)
                          i_a=sign_i_a*D_C;
//                        i_a=D_C;

                    trussD(2,2)=-C_3/2/sqrt(sign_i_a*i_a)/L;
                    trussD(5,5)=-C_3/2/sqrt(sign_i_a*i_a)/L;
                    trussD(5,2)=C_3/2/sqrt(sign_i_a*i_a)/L;
                    trussD(2,5)=C_3/2/sqrt(sign_i_a*i_a)/L;
                    return trussD;
              }

              const Matrix &
                    Pipe3::getMass(void)
              {
                    if (L==0){
                          trussM.Zero();
                          return trussM;
                    }

//                  At this point we have zero lumped mass
                    trussM.Zero();
                    return trussM;
              }

              void
                    Pipe3::zeroLoad(void)
              {
                    //does nothing - no element load associated with this
object
              }

              int
                    Pipe3::addLoad(ElementalLoad *theLoad, double loadFactor)
              {
              opserr <<"MyTruss::addLoad - load type unknown for truss with
tag: " << this->getTag() << endln;

              return -1;
              }

              int
                    Pipe3::addInertiaLoadToUnbalance(const Vector &accel)
              {
                    return 0;
              }

              const Vector &
                    Pipe3::getResistingForce()
              {
                    if (L==0) {//if length ==zero - zero and return
                          trussR.Zero();
                          return trussR;
                    }
                    // R=Ku-Pext
                    //force =F*transformation
                    double force = A* theMaterial->getStress();
                    trussR(0)= trans(0,0)*force;
                    trussR(1)= trans(0,1)*force;
                    trussR(3)= trans(0,2)*force;
                    trussR(4)= trans(0,3)*force;

                    const Vector &vel1 = theNodes[0]->getTrialVel();
              const Vector &vel2 = theNodes[1]->getTrialVel();

                    if ((vel2(2)-vel1(2)+d_y_class*Gamma)>0.0)
                    {
                          trussR(2)=C_3*sqrt((vel2(2)-
  vel1(2)+d_y_class*Gamma)/L);
                          trussR(5)= -C_3*sqrt((vel2(2)-
  vel1(2)+d_y_class*Gamma)/L);
                          double deleteme4=(vel2(2)-vel1(2)+d_y_class*Gamma);
                          double deleteme5=vel2(2);
                          double deleteme6=vel1(2);
                          double deleteme7=vel1(2);
                    }
                    else
                    {
                          trussR(2)=-C_3*sqrt((-vel2(2)+vel1(2)-
  d_y_class*Gamma)/L);
                          trussR(5)= +C_3*sqrt((-vel2(2)+vel1(2)-
  d_y_class*Gamma)/L);
                          double deleteme4=(-vel2(2)+vel1(2)-d_y_class*Gamma);
                          double deleteme5=vel2(2);
                          double deleteme6=vel1(2);
                          double deleteme7=vel1(2);
                    }
                    double deleteme2=trussR(2);
                    double deleteme3=trussR(5);
                    return trussR;
              }

              const Vector &
                    Pipe3::getResistingForceIncInertia()
              {

                    this->getResistingForce();

                    //No inertia is included in the in this element formulation
                    return trussR;
              }


              int
                    Pipe3::sendSelf (int commitTag, Channel &theChannel)
              {
                    int dataTag=this->getDbTag();
```

```
                        // Pipe3 packs it's data into a Vector and sends this to
theChannel
                        //along with it's dbTag and the commitTag passed in the
arguments

                        Vector data(7);
                        data(0)= this->getTag();
                        data(1)=A;
                        data(4)=C_3;
                        data(5)=Gamma;
                        data(6)=D_C;
                        data(2)=theMaterial->getClassTag();
                        int matDbTag=theMaterial->getDbTag();
                        if (matDbTag==0) {
                             matDbTag =theChannel.getDbTag();
                             if (matDbTag !=0)
                                  theMaterial->setDbTag(matDbTag);
                        }
                        data(3)=matDbTag;

                        theChannel.sendVector (dataTag, commitTag, data);

                        theChannel.sendID(dataTag, commitTag, externalNodes);

                        theMaterial->sendSelf(commitTag, theChannel);

                        return 0;
                   }
                   int
                        Pipe3::recvSelf(int commitTag, Channel &theChannel,
FEM_ObjectBroker &theBroker)
                   {
                        int dataTag= this->getDbTag();
                        Vector data(7);
                        theChannel.recvVector(dataTag, commitTag, data);

                        this->setTag((int)data(0));
                        A=data(1);
                        C_3=data(4);
                        Gamma=data(5);
                        D_C=data(6);
                        theChannel.recvID(dataTag, commitTag, externalNodes);

                        int matClass=data(2);
                        int matDb = data(3);
                        theMaterial= theBroker.getNewUniaxialMaterial(matClass);

                        theMaterial->setDbTag(matDb);
                        theMaterial->recvSelf(commitTag, theChannel, theBroker);

                        return 0;
                   }

                   int
                        Pipe3::displaySelf(Renderer &theViewer, int displayMode,
float fact)
                   {
```

```
                const Vector &end1Crd= end1Ptr->getCrds();
                const Vector &end2Crd= end2Ptr->getCrds();
                const Vector &end1Disp=end1Ptr->getDisp();
                const Vector &end2Disp=end2Ptr->getDisp();

                Vector v1(3);
                Vector v2(3);
                for (int i=0; i<2;i++) {
                      v1(i)=end1Crd(i)+end1Disp(i)*fact;
                      v2(i)=end2Crd(i)+end2Disp(i)*fact;
                }
                if (displayMode==3) {
                      //use the strain as the drawing measure
                      double strain = theMaterial->getStrain();
                      return theViewer.drawLine(v1, v2, strain, strain);
                }
                else if (displayMode==2){
                      //otherwise use the material stress
                      double stress =A*theMaterial->getStress();
                      return theViewer.drawLine(v1,v2,stress, stress);
                }
                else{
                      //use the axial force
                      double force = A*theMaterial->getStress();
                      return theViewer.drawLine(v1,v2,force,force);
                }
          }

          void
                Pipe3::Print(OPS_Stream &s, int flag)
          {
                //compute the strain and axial force in the member
                double strain, force;
                if (L==0.0) {
                      strain=0;
                      force=0.0;
                }
                else{
                      strain = theMaterial->getStrain();
                      force=A*theMaterial->getStress();
                }
                trussR(0)= trans(0,0)*force;
                trussR(1)= trans(0,1)*force;
                trussR(3)= trans(0,2)*force;
                trussR(4)= trans(0,3)*force;

                const Vector &vel1 = theNodes[0]->getVel();
          const Vector &vel2 = theNodes[1]->getVel();

                if ((vel2(2)-vel1(2)+d_y_class*Gamma)>0.0)
                {
                      trussR(2)=C_3*sqrt((vel2(2)-
vel1(2)+d_y_class*Gamma)/L);
                      trussR(5)= -C_3*sqrt((vel2(2)-
vel1(2)+d_y_class*Gamma)/L);
                      double deleteme4=(vel2(2)-vel1(2)+d_y_class*Gamma);
                      double deleteme5=vel2(2);
```

```
                        double deleteme6=vel1(2);
                        double deleteme7=vel1(2);
                  }
                  else
                  {
                        trussR(2)=-C_3*sqrt((-vel2(2)+vel1(2)-
d_y_class*Gamma)/L);
                        trussR(5)= +C_3*sqrt((-vel2(2)+vel1(2)-
d_y_class*Gamma)/L);
                  }

                  if (flag==0) {//print everythin
                        s<< "Element: " <<this->getTag();
                        s<< " type: My Truss iNode: "<< externalNodes(0);
                        s<< " jNode: "<<externalNodes(1);
                        s<< " Area: "<< A;
                        if (Gamma!=0) s << "Gamma: "<<Gamma;

                        s<< " \n\t strain: " <<strain;
                        s<< " axial load: " <<force;
                        s<< " \n\t unbalanced load: " <<trussR;
                        s<< " \t Material: " << *theMaterial;
                        s<< endln;
                  } else if (flag==1) {//just print ele id, strain and force
                        s<< this->getTag() << "  " <<strain << "  " << force
<<endln;
                  }
            }


            Response *
                  Pipe3::setResponse(const char **argv, int argc , OPS_Stream
&s)
            {
                  // we compare arg(0) for known response types for the Truss

                  //axial force

                  if(strcmp(argv[0], "axialForce")==0)
                        return new ElementResponse(this, 1, 0.0);

                  //a material quantity
                  else if (strcmp(argv[0], "material")==0)
                        return theMaterial->setResponse(&argv[1], argc-1, s);
                  else
                        return 0;
            }

            int
                  Pipe3::getResponse(int responseID, Information
&eleInformation)
            {
                  switch (responseID){
                        case -1:
                              return -1;

                        case 1:
                                return eleInformation.setDouble (A*theMaterial-
>getStress());
                        default:
                                return 0;
                }
        }

        double
                Pipe3::computeCurrentStrain(void) const
        {
                //determine the strain
                const Vector &disp1=end1Ptr->getTrialDisp();
                const Vector &disp2=end2Ptr->getTrialDisp();

                double dLength=0.0;
                for (int i=0;i<2;i++)
                        dLength -= (disp2(i)-disp1(i))*trans(0,i);

                double strain =dLength/L;

                return strain;
        }
```

3. Tcl/tk command interpreter

```
/* ****************************************************************** **
**    OpenSees - Open System for Earthquake Engineering Simulation    **
**          Pacific Earthquake Engineering Research Center            **
**                                                                    **
**                                                                    **
** (C) Copyright 1999, The Regents of the University of California    **
** All Rights Reserved.                                               **
**                                                                    **
** Commercial use of this program without express permission of the  **
** University of California, Berkeley, is strictly prohibited.  See   **
** file 'COPYRIGHT'  in main directory for information on usage and   **
** redistribution,  and for a DISCLAIMER OF ALL WARRANTIES.           **
**                                                                    **
** Developed by:                                                      **
**   Frank McKenna (fmckenna@ce.berkeley.edu)                         **
**   Gregory L. Fenves (fenves@ce.berkeley.edu)                       **
**   Filip C. Filippou (filippou@ce.berkeley.edu)                     **
**                                                                    **
** ****************************************************************** */

// $Revision: 1. $
// $Date: 2008/07/20 19:20:46 $
// $Source: /usr/local/cvs/OpenSees/SRC/element/pipe/TclPipe3Command.cpp,v $


// File: ~/element/TclPipe3Command.C
//
// Written: avytin
// Created: 09/08
// Revision: A
```

```
//
// Description: This file contains the implementation of the
TclModelBuilder_Pipe3()
// command.
//
// What: "@(#) TclModelBuilder.C, revA"

#include <stdlib.h>
#include <string.h>
#include <Domain.h>

#include "Pipe3.h"
#include <TrussSection.h>
#include <TclModelBuilder.h>
#include <CorotTruss.h>
#include <CorotTrussSection.h>

extern void printCommand(int argc, TCL_Char **argv);

int
TclModelBuilder_Pipe3(ClientData clientData, Tcl_Interp *interp, int argc,
                                    TCL_Char **argv, Domain*theTclDomain,
TclModelBuilder *theTclBuilder,
                                    int eleArgStart){
                                          //make sure at least one other
argument to contain type of system
                                          if (argc!=10){
                                                interp->result = "WARNING bad
command - Pipe3 eleId iNode jNode matID Area c_3 Gamma d_c";
                                                return TCL_ERROR;
                                          }

                                          //get the id, x_loc, y_loc
                                          int trussId, iNode, jNode, matID;
                                          double A, C_3, Gamma, D_C;
                                          if (Tcl_GetInt(interp,argv[2],
&trussId)!= TCL_OK){
                                                interp->result = "WARNING
invalid eleId - Pipe3 eleId iNode jNode matID Area c_3 Gamma d_c";
                                                return TCL_ERROR;
                                          }

                                          if (Tcl_GetInt(interp, argv[3],
&iNode) != TCL_OK) {
                                                interp->result = "WARNING
invalid iNode - Pipe3 eleId iNode jNode matID Area c_3 Gamma d_c";
                                                return TCL_ERROR;
                                          }

                                          if (Tcl_GetInt(interp, argv[4],
&jNode) != TCL_OK) {
                                                interp->result = "WARNING
invalid jNode - Pipe3 eleId iNode jNode matID Area c_3 Gamma  d_c";
                                                return TCL_ERROR;
                                          }
```

```
                                                    if (Tcl_GetInt(interp, argv[5],
&matID) != TCL_OK) {
                                                          interp->result = "WARNING
invalid matID - Pipe3 eleId iNode jNode matID Area c_3 Gamma d_c";
                                                          return TCL_ERROR;
                                                    }

                                                    if (Tcl_GetDouble(interp, argv[6],
&A) != TCL_OK) {
                                                          interp->result = "WARNING
invalid Area - Pipe3 eleId iNode jNode matID Area c_3 Gamma  d_c";
                                                          return TCL_ERROR;
                                                    }

                                                    if (Tcl_GetDouble(interp, argv[7],
&C_3) != TCL_OK) {
                                                          interp->result = "WARNING
invalid C_3 - Pipe3 eleId iNode jNode matID Area c_3 Gamma d_c";
                                                          return TCL_ERROR;
                                                    }

                                                    if (Tcl_GetDouble(interp, argv[8],
&Gamma) != TCL_OK) {
                                                          interp->result = "WARNING
invalid Gamma - Pipe3 eleId iNode jNode matID Area c_3 Gamma d_c";
                                                          return TCL_ERROR;
                                                    }

                                                    if (Tcl_GetDouble(interp, argv[9],
&D_C) != TCL_OK) {
                                                          interp->result = "WARNING
invalid d_c - Pipe3 eleId iNode jNode matID Area c_3 Gamma  d_c";
                                                          return TCL_ERROR;
                                                    }

                                                    UniaxialMaterial *theMaterial =
theTclBuilder->getUniaxialMaterial(matID);

                                                    if (theMaterial ==0) {
                                                          opserr << "WARNING TclPipe3 -
Pipe3  - no Material found with tag ";
                                                          opserr << matID << endln;
                                                          return TCL_ERROR;
                                                    }

                                                    //now create the truss and add it
to the domain
//                                                  MyTruss *theTruss = new
MyTruss(trussId,iNode,jNode,*theMaterial,A,M);
                                                    Element *theTruss = 0;
                                                    theTruss=new
Pipe3(trussId,iNode,jNode,*theMaterial,A,C_3,Gamma, D_C);
                                                    if (theTruss==0) {
                                                          opserr << "WARNING TclPipe3 -
Pipe3 - ran out of memory for node ";
                                                          opserr << trussId << endln;
                                                          return TCL_ERROR;
```

```
                                          }

                                          if (theTclDomain-
>addElement(theTruss)==false) {
                                                delete theTruss;
                                                opserr << "WARNING TclPipe3 -
Pipe3 - could not add Pipe3 to the domain";
                                                opserr << trussId << endln;
                                                return TCL_ERROR;
                                          }

                                          //Everything is OK

                                          return TCL_OK;
}
```

Appendix H

Validation of drain elements

i. One dimensional Problems

For verification purposes tests on the element level have been performed using pressure control and flow control, using one and two drain elements. The results showed great match with hand calculations. The simulations performed are summarized on the next table.

Table G-1 Validation tests for laminar and turbulent flow drains

	Pore Pressure Control	Flow Control
One element	●	●
Two Elements	●	●

ii. Plane Strain Consolidation

The consolidation of a plane strain unit cell, with elastic material governing the response of the soil skeleton has also been examined. The geometry used is shown in Figure 3.12, and is descretized in 400 quadup elements. The boundaries are all impermeable except for the top of the drain. The soil material is linear elastic (E=30 000kPa, v=0.3). The described pipe elements have been used with parameters C_t=0.1, C_l=0.001, and very small stiffness.

Figure 3.13 summarizes the results for this verification. The figure compares the directly calculated vertical settlement at the top of the drain to the indirectly calculated settlement found by computing the mass balance of water flowing through the drain. For this indirect calculation it is assumed that all vertical deformations are due to displacement of pore fluid within the soil skeleton, and that no water is coming out of the soil layer through the soil. For the laminar drain, the directly calculated vertical displacement is very close to the indirectly calculated one.

The agreement is not perfect because there is flow of water coming out of the soil without entering the drain, through point A. On the other hand, for the turbulent drains we can see that the results match very closely each other, this time because the permeability of the drain is very large, and almost all of the flow comes out of drain (very small flow is coming out of the soil mass).

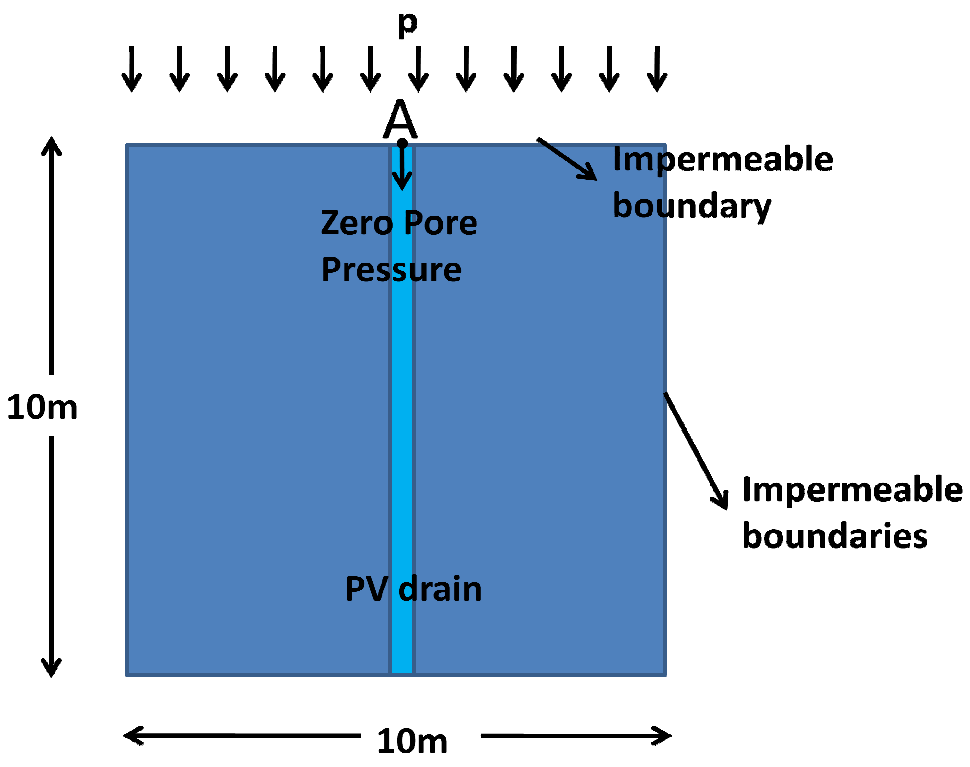


Figure G-1 Geometry of the plane strain consolidation verification problem. Zero pore pressure boundary condition is only applied at point A.

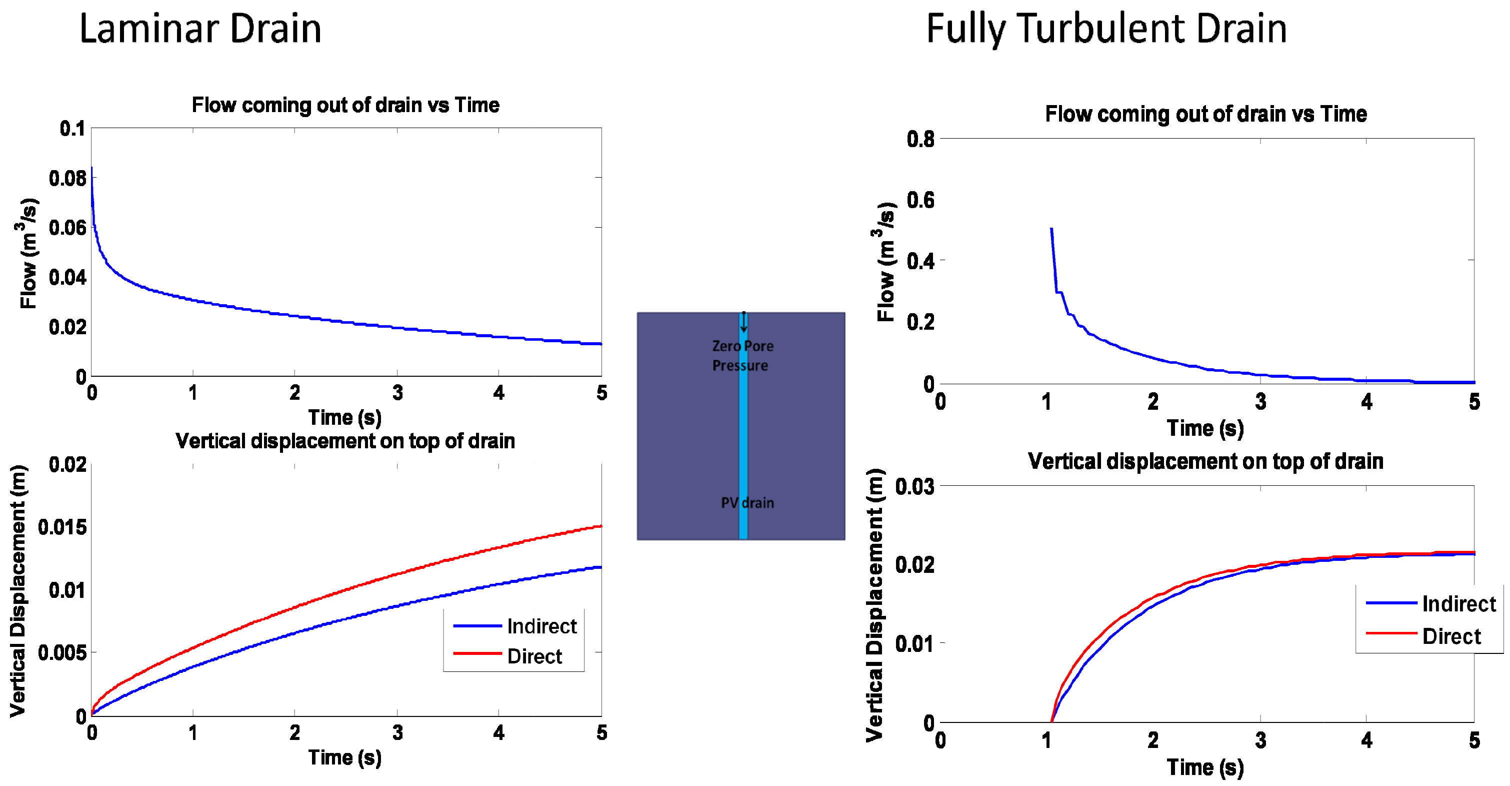


Figure G-2 Validation for laminar and fully turbulent flow drains

Appendix I

Verification of Hird axisymmetric to plane strain drain equivalence theory

Analyses have been performed using ABAQUS, simulating the consolidation of a plane strain and an axisymmetric unit cell around a perfect drain in an elastic soil layer, using the up formulation (E=50 000kPa, v=0.32, k=0.0003m/s). The results presented in Figure H-1 show that the average degree of consolidation is matched greatly. Small discrepancies can be observed: dissipation happens faster in the plane strain unit cell close to the drain, and slower at the boundaries of the unit cell.

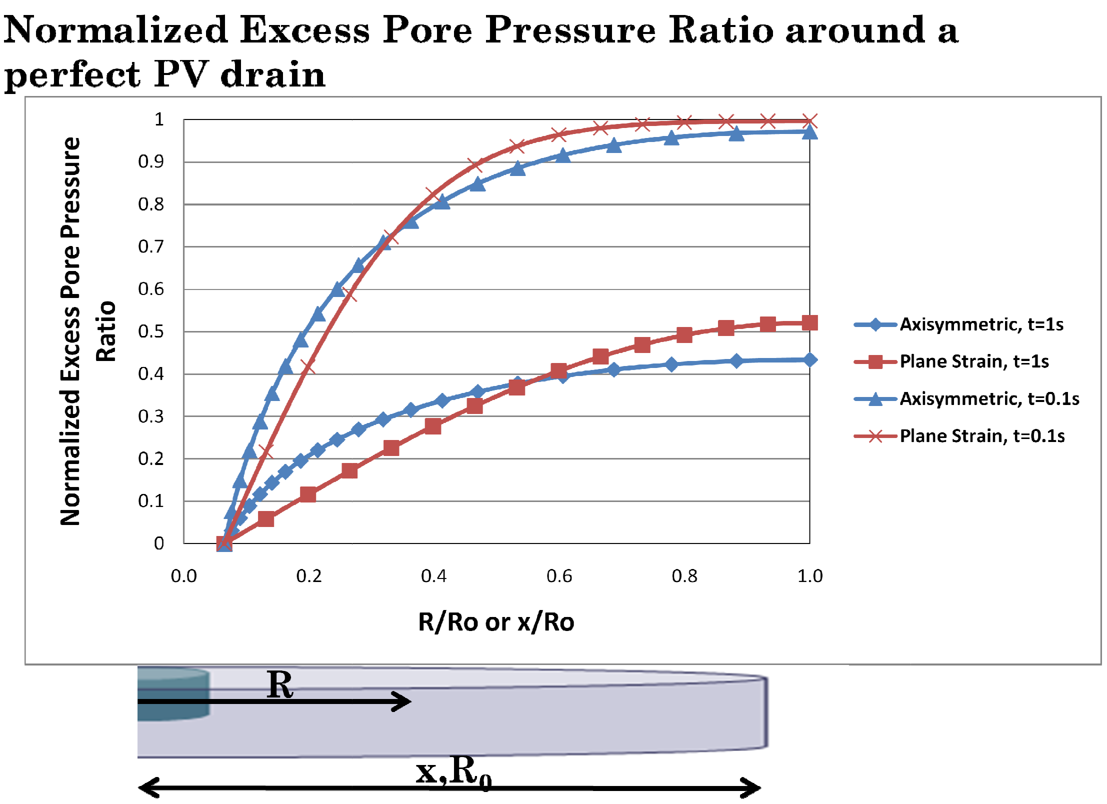


Figure H-1 Comparison excess pore pressures around an axisymmetric and an equivalent plane strain perfect drain