Title
Progress Toward the Next Generation of Bioreactors for 3D Tissue Engineering

Author
Youssef, Khalid

Publication Date
2015

Peer reviewed|Thesis/dissertation

University of California

Los Angeles

Progress Toward the Next Generation of Bioreactors for 3D Tissue Engineering

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Biomedical Engineering

by

Khalid Youssef

2015

Abstract of the Dissertation

Progress Toward the Next Generation of Bioreactors for 3D Tissue Engineering

by

Khalid Youssef

Doctor of Philosophy in Biomedical Engineering

University of California, Los Angeles, 2015

Professor Louis Bouchard, Chair

This research is part of an interdisciplinary project to achieve composite tissue transplantation through novel technology that induces the growth of thick, immunocompatible, tissue implants that are compatible with the patient's immune system. While the common approach to growing cells in vitro utilizes scaffolds embedded in perfusion bioreactors, state of the art methods still suffer from many limitations such as failing to accommodate the dynamical aspect where different stages of development lead to changes in the material properties. The aim of this work is to introduce a new generation of perfusion bioreactors not bound by current limitations. This is achieved by developing noninvasive feedback visualization techniques, new scaffold-patterning technology, new approach to bioreactors design, and control and optimization methods. These novel tools presented herein will contribute to the development of stable vascular networks that are functionally competent to sustain physiological flow and permeate tissue grafts.

The dissertation of Khalid Youssef is approved.

Joseph DiStefano III

Dejan Markovic

William Gelbart

Louis Bouchard, Committee Chair

University of California, Los Angeles

2015

Table of Contents

List of Figures

List of Tables

Vita

2010-2015	University of California, Los Angeles , Ph.D. candidate , Biomedical Engineering.
2006–2010	Northern Illinois University , M.Sc. , Electrical Engineering.
2000–2005	Business and Computer Univ. College , B.E. , Computer & Communications Engineering.
2010–2015	University of California Los Angeles, Graduate Student Researcher / Teaching Assistant, Department of Chemistry & Biochemistry. Teaching Assistant, Department of Neuroscience.
2006–2010	Northern Illinois University , Graduate Assitant / Teaching Assitant , Department of Electrical Engineering. Graduate Assistant, Outreach Broadband Developments.

Publications

Youssef K, Jarenwattananon N, Bouchard LS. Engineering Flow Fields in Bioreactor Scaffolds. Submitted, under review.

Youssef K, Jarenwattananon N, Bouchard LS. Feature-Preserving Noise Removal. To appear: IEEE Transactions on Medical Imaging.

Mack JJ, Youssef K, Lake MP, Noel O, Wu A, Iruela-Arispe ML, Bouchard LS. Real-Time Maps of Fluid Flow Fields in Porous Biomaterials. Biomaterials 34:

1980-1986 (2013)

Youssef K, Woo P-Y. Difference of the Absolute Differences A New Method for Motion Detection. Int. J. Intell. Syst. App. 4(9): 1-14 (2012)

Youssef K, Mack JJ, Iruela-Arispe ML, Bouchard LS. Macro-Scale Topology Optimization for Controlling Internal Shear Stress in a Porous Scaffold Bioreactor. J. Biotech. Bioeng. 109(7): 1844-1854 (2012)

Youssef K. Bare-Hand Tracking Based on the New DAD Object Tracking Method. Masters Dissertation, published by ProQuest. ISBN: 9781124024363. (2010)

Youssef K, Woo P-Y. Efficient Music Note Recognition Based on Self-Organizing Map Tree and Linear Vector Quantization. Soft. Comp. J. 13: 1187-1198 (2009)

Youssef K. Bare Hand Tracking: Comprehensive Literature Review. NIU Engineering Review Journal. 1(2): 1-6 (2009)

Youssef K, Woo P-Y. Music Note Recognition Based on Neural Networks. Natural Computation, 2008. Fourth International Conference. 2: 474-478 (2008)

Youssef K, Woo P-Y. A Novel Approach to Using Color Information in Improving Face Recognition Systems Based on Multi-Layer Neural Networks. Book chapter. Recent advances in face recognition. ISBN 978-953-7619-34-3. pp 223-236 (2008)

Youssef K, Woo P-Y. Robotic Position/Orientation Control Using Neural Networks. Neural Networks, 2008. IEEE World Congress on Computational Intelligence. IEEE International Joint Conference. pp 310-314 (2008)

Youssef K, Woo P-Y. Instrument Sound Separation in Songs. Electro/Information Technology, 2008. IEEE International Conference. pp 442-447 (2008)

Youssef K, Woo P-Y. A New Method for Face Recognition Based on Color Information and a Neural Network. Third International Conference on Natural Computation (ICNC 2007), pp 585-589 (2007)

CHAPTER 1

Introduction

Repairing damaged organs in severely wounded patients requires implanting thick composite tissue/organoids consisting of several types of cells. Cells can be derived from the patient, resulting in an implant that is compatible with the patient's immune system and has the potential to generate multiple tissue types. However, tissue grown from stem cells in culture cannot grow thicker than 300μm in the absence of vascular supply [1]. For example, liver organoids that are grown and implanted into a mouse fail to thrive after implantation due to the lack of well established vessels. This problem has prevented on-demand production of thick tissue implants. For larger and more complex tissues, blood vessels from the host are unable to invade and vascularize the tissue. For these, integration with hierarchic blood vessels is necessary for adequate delivery of gases and nutrients, and is also an essential source of signals for organization and development of functional parts of the organ [2].

A common approach to growing cells in vitro utilizes perfusion bioreactors, which feature suitable fluid and nutrient mass transport, enable cells to grow and migrate, and permit the application of a variety of stimuli to the cells to enhance growth and differentiation. More importantly, bioreactors are employed because the organization of individual cells into functional structures requires a 3D context. A scaffold provides the medium for cell growth. Ideally, it will be a spongy-like material composed of biodegradable polymers. Nutrients and other factors needed for cell growth are delivered through an inlet, while waste products

exit through an outlet. An ideal bioreactor would enable the growth of tissue using appropriate cell mixtures, growth factors and nutrients through the presence of a dynamic environment where growth media is flowed in a controlled manner.

The presence of flow, while unavoidable, can be both beneficial if well controlled, or highly detrimental if uncontrolled. In any case, its role must be better understood. Experimental results show that flow is critical to the growth and organization of stable vascular networks both in 2D and 3D. Shear forces associated with applied flow have been shown to enhance cell growth and differentiation [3, 4, 5]. In addition, interstitial flow distributions bias the extracellular transport and gradients of growth factors, which in turn affect cell-cell signaling and morphogenesis [6]. Mechanical stimulation plays a critical role in many cellular processes, including cell division, contractility, differentiation and motility. It has been an active area of research in regenerative medicine and tissue regeneration [3, 4, 5]. The research performed to date has demonstrated that mechanical stress (shear forces) is an important regulator of cell function and a relevant component for pre-conditioning cells prior to transplantation into live organisms [7, 8, 9, 10, 11, 12].

Several groups have focused on developing perfusion bioreactors that impose flow-induced stresses on cells residing within the scaffold [6, 13, 14, 15]. Parameters such as perfusion rate, flow and stresses are typically selected by trial-and-error. However, growing adequately vascularized thick composite tissue in vitro requires more precise control over the distribution of forces. Without such control, the forces can easily fluctuate (spatially or temporally) across a bioreactor, resulting in damage to the cells. Furthermore, different developmental stages of tissue generally require different inputs and stimuli. Protocols become more complex and begin to evolve with maturation time in bioreactors. It is recognized that culture conditions need to adapt to tissue maturation evolution.

Until recently, researchers have achieved only crude control of mechanical

forces, such as the average value of the forces throughout a scaffold. Desired averages of mechanical stress were achieved through manipulating scaffold porosity, a micro-structural property of the scaffold, and fluid flow rates through the chamber's inlet/outlet. Recent studies, have demonstrated the presence of complex internal micro-fluidic patterns within cell growth medium that can alter cell development [16, 17]. Considerable efforts have been devoted to tailoring shear stress distributions arising from flow fields applied to bioreactors as a means to study the cellular response due to spatial gradients [18, 19, 20, 21, 22]. State of the art approaches involve creating materials with local porosity gradients [23, 24, 25, 26, 19, 20, 27, 28]. While this is suitable for setting initial conditions in a scaffold at the expense of high material complexity, it fails to accommodate the dynamical aspect where different stages of development lead to changes in the material properties.

In the next-generation of bioreactors, adaptive control will be crucial to insure regenerated tissue quality. In addition to the need to adapt culture conditions to tissue maturation evolution, due to many variability sources, a culture conditions sequence that may work to bring a construct from its initial conditions to its targeted state in one case, may fail in another [29]. However, it is presently difficult to experimentally determine let alone control flow profiles and flow heterogeneity within a 3D matrix. Such knowledge would enable optimizing the growth of functional tissue, which otherwise suffers from the inability to grow beyond small clusters of cells, through an adaptive control mechanism with feedback.

The work presented herein contributes to advancing spatiotemporal control in bioreactor scaffolds through two routes; 1. developing noninvasive visualization techniques using an NMR phase-contrast velocimetry method that can obtain spatially accurate distributions of velocity fields inside biomaterials. 2. developing flow control, scaffold-patterning technology, and control techniques that convey precise mechanical stresses to cells that are, in addition, spatially and temporally

distributed according to predetermined parameters. From the engineering standpoint, knowledge of the flow fields may reveal degradation/changes of the scaffold over time that can be extremely informative feedback for optimization. The main advantage of using NMR is that it's nondestructive to the cells. Furthermore, the obtained information will be essential to study cellular responses to flow and evaluate flow-induced cell behavior. The aim is to help provide control over the growth of vascularized tissue in real time, and contribute to the growth of large vascularized tissue. The rationale is that these novel tools will enable the development of stable vascular networks that are functionally competent to sustain physiological flow and permeate tissue grafts.

A new technique to controlling mechanical stress values through manipulating the scaffold macrostructure by patterning specific networks of microfluidic flow channels, in lieu of targeting scaffold microstructure, is presented. It is shown that spatial distributions of fluid shear forces can be precisely controlled by creating engraved patterns on the face of the scaffold to redirect the path of fluid flow. A novel optimization technique is developed based on computational intelligence algorithms, where the engraved pattern corresponding to a desired distribution can be determined. This allows for creating desired distributions and gradients of mechanical stress on demand. Two scaffolds made from identical porous materials can be made to yield (if desired) entirely different flow properties.

As the tissue grows, the resistance to flow will be altered over time. To overcome this problem, a real-time control technique is presented. It involves measuring flow patterns in real time using computational flow dynamics techniques combined with non-invasive real-time visualization methods to derive a variety of relevant hydrodynamic parameters, and adjustment of the flow field using a flexible multi-channel bioreactor. The readouts can help assess and track cell development inside a scaffold by estimating cell growth from the change in porosity in a noninvasive manner. In addition to more precise spatial control, using a cus-

tomized bioreactor containing ten inlets in lieu of the typical single-inlet design can also offer temporal control of flow fields. The inlet flow rate is used to alter the stress distribution throughout the scaffold at will. By using multiple inlets, flow entering the chamber from different inlets at different rates interacts to create complex flow field patterns. Using an adaptive control technique developed based on artificial neural networks, the combination of flow rates at each individual inlet that correspond to a desired flow pattern can be determined. Different combinations yield different flow patterns throughout the same scaffold. This offers the flexibility of using different distributions at different times, or adaptively maintaining the same distribution as the medium structure changes with cell growth. Because tissue/cell growth occurs randomly, the tissue regeneration process must make use of adaptive real-time feedback control.

Chapter 2 presents a novel computational optimization method to control shear stress distributions in porous media, by optimizing macro-scale channel topology to target a desired shear stress within a porous scaffold. The approach is validated by MRI flow experiments on polymeric scaffolds. It addresses the design of biomaterial scaffolds for controlling fluid flow and shear stress. Furthermore, it presents a technique to experimentally map the flow field and measure shear stress distributions throughout porous scaffolds.

Chapter 3 presents real-time, non-invasive measures of local hydrodynamics in 3D biomaterials based on nuclear magnetic resonance. Microflow maps were further used to derive pressure, shear and fluid permeability fields. Finally, remodeling of collagen gels in response to precise fluid flow parameters was correlated with structural changes. Accurate flow maps within 3D matrices will be a critical step towards understanding cell behavior in response to controlled flow dynamics.

Chapter 4 presents a new NMR denoising technique with superior feature-preserving abilities. Denoising is efficiently done using a nonlinear filter, which operates along patch neighborhoods and multiple copies of the original image. The

use of patches enables the algorithm to account for spatial correlations in the random field whereas the multiple copies are used to recognize the noise statistics. The nonlinear filter, which is implemented by a hierarchical multistage system of multilayer perceptrons, outperforms state-of-the-art denoising algorithms such as those based on collaborative filtering and total variation. Compared to conventional denoising algorithms, the developed filter can restore images without blurring them, making it attractive for use in medical imaging where the preservation of anatomical details is critical, and an important tool for speeding up NMR data acquisition in order to facilitate real-time feedback.

Chapter 5 presents a new method for controlling interstitial flow fields in scaffolds used for tissue engineering applications. A new bioreactor design utilizing ten fluid inlets is introduced, together with an algorithm that dynamically solves the inverse problem of computing inlet pressures required to obtain a target flow field, where arbitrary flow fields can be generated in a single scaffold by individually adjusting inlet pressures. As most bioreactors used to date have been designed to operate with a single inlet for fluid flow, this approach can introduce several advantages over existing methods. These include more accurate and complex flow fields, real-time control in response to cell growth without altering the composition or structure of the scaffold material itself, and reducing the need for engineering the scaffold material properties and alleviating manufacturing complexity. This constitutes a new platform for studies of cellular responses to mechanical forces in complex environments. It opens potentially transformative possibilities for tissue engineering, and will facilitate studying the effect of mechanical force distributions on tissue development in a bioreactor scaffold beyond initial stages of cell growth.

CHAPTER 2

Macro-Scale Topology Optimization for Controlling Internal Shear Stress in a Porous Scaffold Bioreactor

Article published: Biotechnology and Bioengineering Volume 109, Issue 7, pages 1844-1854, July 2012

2.1 Introduction

Mechanical stimulation of cells has been an important area of research in regenerative medicine and tissue regeneration [4, 3, 5]. Several groups have focused on developing perfusion bioreactors that impose flow-induced shear stresses on cells residing within a porous matrix [6, 14, 15, 13]. The research performed to date has demonstrated that shear stress is an important regulator of cell function and a relevant component for preconditioning cells prior to transplantation into live organisms [8, 12, 10, 11, 7, 9]. In previous work, local shear stresses were defined as a function of media flow rate and dynamic viscosity, bioreactor configuration, and porous scaffold micro-architecture [30, 31, 32]. Target shear stress values were estimated as averages over the entire scaffold and changes in the overall average were achieved by varying the scaffold micro-architecture under similar flow rates [33]. Recent studies, however, have demonstrated the presence of complex internal microfluidic patterns within the architecture of the porous scaffolds that can alter cell growth in culture [16, 17]. Nonetheless, methods that convey pre-

cise shear stresses to cells that are, in addition, spatially distributed according to predetermined parameters are yet to be developed. Although modeling of scaffold architecture and flow conditions have been considered, no published work, to our knowledge, has systematically optimized the macroarchitecture of channel topology within a porous scaffold to achieve a target shear stress under defined flow inputs. Such a method could have important advantages in that macroscopic flow conditions are easier to alter than the microstructural parameters, especially if real-time flow control is required.

In this article, we present a method to define channel topologies that yield a desired shear stress distribution. In our computational approach, channel topologies are governed by a genetic algorithm (GA) and shear stress distributions assessed by simulating flow using a Lattice Boltzmann Method (LBM). The GA achieves optimization by minimizing a cost function based on fuzzy logic rules. The optimized topologies demonstrate the ability to obtain a target shear stress with a more narrow distribution when compared to the case of a non-optimized topology. The method could be adapted to generate any desired distribution of internal shear stresses by modification of the cost function. To validate the optimization results experimentally, the trend in shear stress distributions under real flow conditions in porous scaffolds were derived from experimental measurements of flow assisted by magnetic resonance imaging (MRI) techniques. MRI is suitable for imaging flows non-invasively in optically opaque media (such as porous polymer scaffolds). This article provides proof-of-principle for the algorithm in two dimensions (2D) and its experimental realization by approximation using stacks of thin slices of scaffold material. Our method could be extended to optimizing flows by patterning channels at the microfluidic level, using micro-computed tomography (m-CT) images as inputs to the pore geometry, and using three-dimensional (3D) prototyping techniques for the structural reconstitution of the scaffolds. The MRI technique has the advantage that it can measure flows noninvasively in real-time,

and should cell-seeded scaffolds be used, it may be possible to devise adaptive schemes to adjust the flow patterns as needed to account for the effects of cell growth.

2.2 Methods

2.2.1 Lattice Boltzmann Method (LBM)

To model flows through porous media, a computational fluid dynamics program was developed based on LBM with no-slip boundary conditions [34]. Pores are modeled as open spaces in the lattice within which the fluid is allowed to flow. On the other hand, the walls surrounding a pore are modeled as closed spaces representing a solid, such that the fluid bounces back upon collision. A D2Q9 Bhatnagar-Gross-Krook (BGK) scheme was adapted to satisfy the evolution equation of the density distribution function and the conditions defined by Equations (1) to (3)

$$\Delta_i f_i = -\omega(f_i - f_i^e) \tag{2.1}$$

$$\sum_i f_i^e = \sum_i f_i = \rho \tag{2.2}$$

$$\sum_i f_i^e c_{ia} = \sum_i f_i c_{ia} = \rho u_a \tag{2.3}$$

where f_i is the density distribution function, f_i^e is the equilibrium density distribution function, ρ is the density, $a = x, y$ and $u = (u_x, u_y)$ as velocity, c the velocity direction and $\omega = (\frac{1}{t})$ with t as the time scale. These conditions lead to a generic family of LBGK equilibria that are expressed by Equation (4) in which Q_i is the projector along the i^{th} discrete direction. For the D2Q9 scheme, values for the weights w_i and the speed of sound C_S are fixed according to Equation (5).

$$f_i^e = \rho w_i(1 + \frac{c_{ia}u_a}{c_s^2} + \frac{Q_{iab}u_a u_b}{2c_s^4}) \tag{2.4}$$

$$c_s^2 = \frac{1}{3}; w_9 = \frac{4}{9}; w_{1,2,3,4} = \frac{1}{9}; w_{5,6,7,8} = \frac{1}{36} \tag{2.5}$$

Shear stress was calculated under the Newtonian fluid assumption using Equation (6)

$$\overset{\leftrightarrow}{\tau} = \nu \left(\frac{\nabla U + \nabla U^T}{2} \right) \tag{2.6}$$

where $\overset{\leftrightarrow}{\tau}$ is the shear stress tensor, ν is the dynamic viscosity, and U is the 2D velocity vector field. Shear stress τ was calculated as the largest eigenvalue of the tensor $\overset{\leftrightarrow}{\tau}$. The derivatives of the velocity field for the gradient were calculated by finite-difference approximation for the four partial derivatives at each point on the lattice corresponding to the fluid phase. The finite-difference approximations are given in Equations (7) to (10):

$$\frac{\partial U_x(i,j)}{\partial x} \approx \frac{U_x(i+1,j) - U_x(i-1,j)}{2} \tag{2.7}$$

$$\frac{\partial U_x(i,j)}{\partial y} \approx \frac{U_x(i+1,j) - U_x(i,j-1)}{2} \tag{2.8}$$

$$\frac{\partial U_y(i,j)}{\partial x} \approx \frac{U_y(i+1,j) - U_y(i-1,j)}{2} \tag{2.9}$$

$$\frac{\partial U_y(i,j)}{\partial y} \approx \frac{U_y(i,j+1) - U_y(i,j-1)}{2} \tag{2.10}$$

Simulations were performed on a 128×128 lattice and the method was validated against "textbook" problems with known analytical solutions: for laminar flows, benchmark models such as the parallel-plate model were used (the simulation results were compared with the analytical solutions for flow between two infinite plates); for turbulent flows, vortex streets around a cylinder were simulated.

2.2.2 Generation of the Channel Topology

To generate channel topologies, we create a set of curves, each of which is represented by a $(\frac{L}{2})$-dimensional vector where L is the length of the flow chamber (length is measured along the principal direction of flow). This vector is obtained by calculating the inverse discrete cosine transform (IDCT) of a second vector of an equal dimension. The second vector is zero-filled, except for a window near the start of the vector, and these non-zero entries determine the spatial frequency content of the curve. The number of discrete cosine components used determines the complexity of the curve, which is reflected by the maximum number of turns. A large number of components, however, increases the parameter space for the topology search and channel complexity and could potentially lead to long optimization times and difficulty in patterning the channels.

A set of curves is obtained, such as illustrated in (Fig. 2.1) A and B. These curves are superimposed, as shown in (Fig. 2.1) C, forming one quadrant of the 128×128 lattice. They are then symmetrically replicated to cover the entire lattice (Fig. 2.1) D. Finally, image-processing techniques are used to obtain the desired channel width and the final topology as shown in (Fig. 2.1) E.

The DCT is known for its energy compaction property, whereby signal information tends to be concentrated in the lower frequency components of Fourier space. Using a topology generation technique that is based on the superposition of curves, we take advantage of the DCT compaction property to greatly reduce the number of parameters in the search space. This makes the optimization more tractable by reducing the search time. This approach also offers additional advantages, such as the ease of manufacturing and simulation. In principle, however, it should be possible to realize more complex scaffold geometries using 3D prototyping printers. In this case, one could envisage using entirely different methods for generating the topology.

2.2.3 Topology Optimization by GA

The GA finds an optimal channel configuration under a set of fixed parameters, i.e., porosity, channel width, number of channels, and flow rate. We note that the flow of fluid requires a connected pore space. This can be achieved by considering only pore spaces that are connected; however, doing so would require a detailed simulation of the 3D pore space geometry, leading to substantial increases in computational requirements. In our case, we have instead simulated the pore space using an "effective-medium" approach and our way of ensuring a connected pore space is to fix the porosity to a sufficiently large value. The search is performed in an evolutionary manner for a set of discrete cosine components that generate a channel topology using the mechanism described in Generation of the Channel Topology Section to obtain an optimal channel geometry that yields a uniform target shear stress throughout the porous media.

The search begins with a population of random solutions in the search space, which is defined by all combinations of non-zero entries of each vector. It then converges towards an optimal solution by evaluating the performance of each member in the population and passing the best two members to the next generation in a survival-of-the-fittest manner. The genes of the members with the best performance are crossed over and a random mutation is performed. The resulting offsprings are then added to the population as new members. As the generation number increases, members with improved performance evolve. An illustration of the evolutionary process carried out by the GA is shown in (Fig. 2.2) where the shear stress histograms and distributions for members in subsequent generations are shown. Each member in the population represents a channel topology and the performance evaluated by simulating fluid flow using the LBM to determine shear stress distribution and assign a score based on a cost function. The ultimate goal is to find the channel topology that leads to a minimum of the cost function.

2.2.4 Cost Function

The $L-1$ norm error, ε, can be used to measure the difference between the shear stress distribution $\tau(i,j)$ of a given channel topology and a target distribution γ as described in Equation (11)

$$\varepsilon = \sum_{(i,j)\in\Lambda} |\tau(i,j) - \gamma| \tag{2.11}$$

where Λ is the set of lattice coordinates.

The goal of this optimization is to determine a channel topology that produces a target average shear stress that is uniformly distributed. The cost function can also be defined by the combination of the average and the standard deviation as in Equation (12).

$$\varepsilon = w_1 \cdot (\mu - \gamma) + w2 \cdot \left(\frac{\delta}{\mu}\right) \tag{2.12}$$

where μ is the average shear stress and δ is the standard deviation. Parameters w_1 and w_2 are weights that determine the influence of each component on the total error. This cost function is a linear relation that can be interpreted as a decision process that attempts to estimate the performance based on two separate values: $(\mu - \gamma)$ and (δ/μ) For example, if $(\mu - \gamma)$ is deemed a more important factor in determining the error, it is assigned a larger weight. Equation (12) is not the best cost function to use if we want to assign more weight to (δ/μ) when its value is exceedingly low, which is usually the case when optimizing for a narrow distribution of shear stress. An interesting alternative is to use fuzzy logic to model the decision process. Fuzzy logic is a mathematical tool for dealing with uncertainty. It provides a mechanism for representing linguistic constructs such as “small, “medium, “good and “few to model the ambiguity associated with vagueness and imprecision [35]. A possible fuzzy logic system is to evaluate the

performance of a given channel topology based on a set of four rules: if $(\mu - \gamma)$ is small and (δ/μ) is small then error is small; if $(\mu - \gamma)$ is big and (δ/μ) is small then error is medium; if $(\mu - \gamma)$ is small and (δ/μ) is big then error is medium; if $(\mu - \gamma)$ is big and (δ/μ) is big then error is large. Compared to the L-1 norm, this fuzzy logic method gives an improvement of $\sim$15% in shear stress standard deviation when the average stress matched the target shear stress. A comparison of the relation between $(\mu - \gamma)$ and (δ/μ) using Equation (12) versus using fuzzy logic is shown in (Fig. 2.3). Membership functions were represented by Gaussian curves. The non-linear relationship enables the GA to reach an optimal configuration more efficiently.

2.2.5 Flow Experiments on Polymeric Scaffolds

2.2.5.1 Scaffold Fabrication

Our theoretical results (to be described in Results Section) were validated in experiments on porous polymer scaffolds fabricated with flow channels of various topologies. Porous scaffolds were prepared by a porogen leaching method to achieve high porosity. Sugar crystals (250355 μm size distribution) served as the porogen and were added to a 20 weight percent solution of polycaprolactone (PCL) in dichloromethane and thoroughly mixed to form a viscous paste (14:1, sugar:PCL). The sugar/PCL paste was then added to a Teflon mold (2×2 cm^2) machined with the desired channel topology as displayed in (Fig. 2.4) A. A Teflon plunger was applied to uniformly distribute the paste within the mold and compress the sugar crystals, thereby resulting in an interconnected network of pores upon removal of the sugar. Once the paste has been distributed and compressed, the scaffold was allowed to cure via solvent evaporation overnight. To achieve open porosity, the scaffold was leached in deionized water for several days, refreshing the water several times a day. To achieve a desired thickness, multiple

scaffolds can be stacked as shown in (Fig. 2.4) B.

2.2.5.2 Microstructural Characterization of Scaffolds

X-ray μ-CT of the porous scaffolds revealed pore sizes on the order of $\sim$250 mm; a representative microtomographic image is displayed in (Fig. 2.4) C. The μ-CT data was obtained on a SkyScan 1172 with 13 μm spatial resolution and then analyzed using Matlab (Mathworks, Natick, MA) and iMorph softwares (http://imorph.sourceforge.net). The 3D data was used to compute the porosity (>70%), specific surface (20,164 $\frac{m^2}{m^3}$) and pore size distribution as displayed in (Fig. 2.4) D and E. These parameters (obtained in this study via μ-CT) should not be taken as true values upon which the success of any method hinges, as the actual values could be different, depending on the spatial resolution and imaging modality used to characterize the scaffold topography. For example, scanning electron microscopy revealed more intricate pore features than is observed in the μ-CT scan (see (Fig. 2.4) F).

2.2.5.3 Bioreactor Construction

To test the porous scaffolds under flow, a cylindrical bioreactor was custom designed and built. The bioreactor was machined from Teflon in two half-sections to house a 2×2 cm^2 square and 1 cm-deep flow chamber, shown in (Fig. 2.4) G. Inlet and outlet ports were drilled through the Teflon to allow fluid to enter and exit the flow chamber. Plastic tubing was connected via Swagelok brass fittings to the inlet and outlet ports. To perform the flow experiments, the scaffold of interest was placed at the base of the flow chamber so that the inlet and outlet were centered at the scaffold surface. To fill the chamber, two non-channeled porous scaffolds were then added and compressed by tightening four nylon screws to seal the bioreactor.

2.2.5.4 Flow Imaging

A 400 MHz vertical-bore Varian NMR system was used in all experiments. The bioreactor was placed in the bore of a 40 mm-i.d. imaging probe. For flow imaging, a spin-echo multi-slice (SEMS) sequence was modified to perform phase-contrast MRI velocimetry [36] as follows. All existing gradients, except the phase encoding gradient, were flow compensated. The flow compensation (F.C.) gradients added are shown as green lobes in (Fig. A.3). Pairs of flow weighting (F.W.), bipolar gradients were added along x, y, and z axes to select the gradient first moment (M_1). Two experiments were first performed under flow with two gradient values, $+M_1$ and $-M_1$, where the value of M_1 was chosen large enough to include the highest anticipated flow velocity and avoid phase wrap-around effects. The two experiments were subtracted in order to obtain a velocity map. The velocity maps, however, contained artifacts from gradient non-idealities (e.g., eddy currents and non-linearities) and therefore the contribution from non-idealities was subtracted from an identical experiment performed in the absence of flow. With this approach, we were able to measure flow velocities below 1 mm/s using a total of 12 scans: 3 gradient directions $(x, y, z) \times 2$ gradient reversals $(M_1, -M_1) \times 2$ runs (flow, no flow). The imaging slice (1.0mm thick) was centered on the inlet/outlet of the flow chamber, which fed into the patterned surface of the porous scaffold. The remaining imaging parameters were set to TR=5 s, TE=50 ms, 256×256 matrix and a field of view equal to 25×25mm^2. The source code for the Varianpulse sequence used in these experiments is available for download free of charge at our web site: http:// 3Dneovascular.mcdb.ucla.edu

2.3 Results

2.3.1 LBM Simulations

LBM simulations were performed for three different scaffold configurations. The first configuration is a porous scaffold with no channels. The second configuration is a porous scaffold with four parallel channels connected to the flow chamber inlet and outlet. We will refer to this configuration as the "manual design". The third configuration is a porous scaffold with channel topology that was optimized using the topology optimization method. Since this study is a proof-of-concept for the optimization strategy, a number of constraints were imposed for time efficiency and to allow a fair comparison to the manual design topology. Accordingly, the optimization was performed by enforcing comparable channel width and number of channels. Although this limited the full potential of optimization, the resulting topology demonstrated a substantial improvement over the control (manual design). All configurations simulated a scaffold porosity of 95% (in 2D) and average flow velocity at the inlet of $\sim$26 mm/s. Based on results for random sphere packings in n-dimensions (Torquato, 2001), a 95% porosity in 2D likely corresponds to a lower porosity in 3D.

The simulation results of the three configurations are shown in (Fig. 2.5) A$-$C. The shear stress histograms are normalized by the standard deviation and the shear stress maps are plotted on a logarithmic scale to simplify comparison of the three channel topologies. When a target shear stress value of 2 (logarithmic scale) was set for the optimization, an improvement in shear stress uniformity and distribution is observed for the optimized configuration. (Fig. 2.5) shows the standard deviation divided by mean shear stress (δ/μ) for each configuration as a measurement of distribution uniformity. The optimized configuration was more uniform than both the no-channel (10.6%) and the manual design (5%). We note than 10% improvement in the quantity (δ/μ) is actually quite a large improvement

in the shear stress uniformity: the value (δ/μ) optimized for is volume-averaged over the entire bioreactor including regions near the wall, which see zero flow. It is, for example, impossible to improve on the values of this ratio near the reactor boundaries due to the no-slip condition. The best way to see the substantial improvement that can be achieved using our method is to compare the results of (Fig. 2.5) (bioreactor with no channels) to the optimization of (Fig. 2.7) presented later.

In all cases, simulations produced a non-linear relationship between input pressure and average shear stress. For example, in the case of the no-channel configuration, a maximum input pressure was reached after which no further increases in the average shear stress were observed. The introduction of macro-scale channels to the scaffold alters the average shear stress limit and by implementing our optimization method, we were able to systematically vary the average shear stress limit to a desired target, whereas without topology optimization, this value remained below the plateau maximum. Furthermore, our optimization method offers flexibility to control different aspects of shear stress distribution by modifying the rules of the fuzzy logic cost function. In other designs (not shown here), the cost function was modified to assign more weight to minimize standard deviation thereby producing a channel topology with a value of $\sim$0.7. This relative deviation can be further optimized by increasing the number of channels and reducing the width of the channels. However, pushing the level of complexity makes it more challenging to realize the patterned scaffold experimentally. This is where improvements in our method could be realized using a 3D prototyping printer instead of CNC milling.

2.3.2 Flow Rate Simulations

Varying the inlet flow rate varies the applied pressure difference across the flow chamber of the bioreactor. Simulations were performed for 10 different flow rate

values that ranged from very low (0.1 mm/s) to very high (5 mm/s). Five simulations for each flow rate were performed with the same target shear stress and uniformity measured as the ratio between standard deviation and average shear stress. (Fig. 2.6) plots the relationship between shear stress uniformity and flow rate. Analyzing this relationship, the ideal flow rate for a target shear stress value can be determined and the overall performance (least fluctuations between runs) can be identified. Large fluctuations were observed for very low and very high flow rates. At very low flow rates, fluctuations are mainly due to diffusion. At very high flow rates, the fluctuations are likely due to an inability to find a channel topology that can match the high flow with the given target shear stress. In other words, if the flow rate is too high for the target shear stress, no channel topology can be found to match the target prescription. This is the main cause for the observed random fluctuations from one run to another.

2.3.3 Flow Rate Optimization

In the previous optimizations, the flow rate was kept fixed during the optimization. We also investigated the effects of flow rate on shear stress distribution when the flow rate is allowed to vary during the optimization. In this case, the GA searches for a combination of topology and flow rate to achieve an optimal distribution for a target shear stress. The resulting channel topology is displayed in (Fig. 2.7), where the flow rate corresponds to an inlet velocity value of 91 mm/s. Improvement was observed both visually and quantitatively since a smaller (δ/μ) value was obtained. This suggests that removal of the flow rate constraint gives the GA more freedom to optimize the channel design.

2.3.4 Flow Imaging Experiments on Fabricated Scaffolds

Each of the fabricated 2×2 cm^2 porous scaffolds was placed into the bioreactor flow chamber and water was flowed at a constant rate of $\sim$5mL/min using a syringe pump (Harvard Apparatus). The flow chamber was filled using additional non-channeled porous scaffold material to minimize water flowing outside the channel region and making sure that the surface of each scaffold of interest (see (Fig. 2.8) A$-$C) was aligned with the chamber's inlet and outlet. The 2D MRI slices were imported into MATLAB (The Mathworks, Natick, MA) for analysis. The flow velocity maps yielded three orthogonal components of the velocity, $\vec{v} = (v_x, v_y, v_z)$, at each point in space. A plot of the magnitude of the velocity, $|\vec{v}|$, is shown in (Fig. 2.8) D$-$F. Velocity fluctuations were estimated from data acquired at zero flow and yielded errors (1σ limits) on the order of 0.05 mm/s. From the velocity maps, shear rates were calculated by finite-differences in the plane of the slice (flow was in-plane) using Equation (13).

$$\dot{\gamma} = \frac{\partial_x v_y + \partial_y v_x}{2} \tag{2.13}$$

The absolute values of the shear rates at each pixel were retained for analysis and the resulting shear rate maps are shown in (Fig. 2.8) G$-$I. Histograms of the shear rate distributions were plotted normalized to the number of values in (Fig. 2.8) J$-$L. We note that this analysis neglects the z-component of velocity, which could reach up to 30% of the velocity magnitude in some regions of the scaffold. The values of (δ/μ) for the shear rate distribution are listed in the insets of (Fig. 2.8) J$-$L. Substantial improvement was observed for the manual design compared to the case of no channels. Likewise, a narrower distribution of shear rates is measured in the topology-optimized scaffold as compared to the manual design and the no-channel scaffold. This demonstrates our ability to control the variance of the shear rate distribution. We note that the shear rate maps (Fig. 2.8)

G−I do not match exactly those of the simulations performed (Fig. 2.5) and we speculate that the divergence is likely due to the fact that the experimental flow patterns are inherently 3D, whereas the shear rate analysis was performed using only two of the velocity components. True 3D analysis of the flow field could be possible by adapting our MRI flow sequence to perform 3D tomography instead of 2D slice readouts. Nonetheless, it is noteworthy that the trends in (δ/μ) measured experimentally match those predicted theoretically and performance of the topology-optimized scaffold exceeded our theoretical predictions.

2.4 Discussion

All the simulations in this work are done in 2D. The same method could be extended to control shear stress distributions in 3D. The problem of computation time could be solved using parallel processing techniques. In our 2D calculations, the population size used in the GA is 20 and the number of generations is 100, for a total of 2,000 runs. Considering the size of the search space, the number of runs performed by the GA is quite low. For example, the number of parameters used by the GA in the flow rate simulations is 28. If we consider a resolution of 10^3 for each parameter, this produces a search space of $1{,}000^{28}$ possibilities. Using an average personal computer, each run takes an average of 9 s for a 128×128 lattice, or ∼5 h to perform a single optimization. To determine the computational time required for 3D optimizations using porosity maps from experimental data, we performed small-scale 3D LBM simulations of the flow field on the m-CT data. The 3D LBM simulations showed a linear relationship in computational time with the 2D LBM simulations. This means that a 128×128×128 3D lattice requires 128 times the time required by a 128×128 2D lattice for equivalent time steps. Additionally, the convergence time will increase with lattice size and the 3D lattice will require more time steps to converge.

In preliminary tests conducted on a GPU array (nVIDIA 460GTX) the speed enhancements observed were as high as 28 times, compared to a 3.2 GHz Intel CPU (non-parallel computation). The 460 GTX card contains 336 cores with 650MHz clock speed and 1GB memory. On a nVIDIA TESLA card containing 448 cores with 1.15 GHz clock speed and 3GB memory, speed enhancements of 50 times would be expected. Given the increase in computational time due to large lattice size, convergence time and speed enhancement from GPU implementation, an estimate for 3D simulations is approximately 20 times the duration of 2D simulations. Thus, a $128\times128\times128$ 3D optimization is expected to take approximately 4 days.

In future work, the biological response to the scaffold channel topology could be experimentally studied. Control over flow and shear stress within a 3D porous scaffold is essential in order to correlate cell response to a defined shear stress distribution. By controlling the scaffold macroarchitecture, we can control flow in 3D and study the effects of shear stress, uniform or varied, on cell growth (i.e., cell proliferation and migration). The ability to vary the scaffold topology allows us to control patterns of shear stress within the scaffold. In this study, we focused on producing uniform patterns of shear stress. However, the spatiotemporal patterns of shear stress required for optimal cell growth and proliferation over time are presently unknown. Therefore, the ability to alter the flow pattern in both space and time will be paramount to understanding a biological response to flow and correlated mechanical forces. This need for adaptable flow patterns is especially relevant considering cell growth within a scaffold is expected to increase the resistance to flow. The MRI technique is particularly suited for use in adaptive control schemes during growth, due to its ability to provide real-time, non-invasive measurements. To control flow in 3D, multiple inlets can be arranged such that patterns of shear stress are applied to stimulate cell growth within the scaffold and ultimately improve ex-vivo growth of complex tissues in 3D. While

we have not tested the effects of patterned flows on cell growth in this study, many research groups are pursuing such investigations. The highporosity and large-pore-size PCL scaffolds used in our experiments feature open pores on the surface that facilitate the seeding of the scaffold via injection or gravity-mediated transport and diffusion. We note that any other scaffold material could potentially be used instead of PCL provided that it is permeable to fluid flow and the porosity is known to a reasonable degree.

2.5 Conclusion

In this paper, we presented an algorithm to control shear stress distributions in porous polymeric scaffolds through optimization of macro-scale channel topology. Experimental results confirm the validity of our 2D optimization methods and future work will focus on extending these methods to control shear stress distribution in 3D. Parallel processing techniques are under consideration to reduce computation time and extend the optimization method to 3D geometries. Providing a means to control shear stress through macro-architecture of a scaffold could facilitate investigations of shear stress distribution on cell proliferation and gene expression under flow in a 3D scaffold. The main advantage of redirecting macroscopic flows is the ability to optimize the shear stresses throughout the entire bioreactor volume without the need to optimize the scaffold microstructure. The work could be extended to include 3D prototyping techniques for the production of scaffolds with more complex geometries. In this case, both microfluidic and macroscopic control of the flow topology may be achievable.

Figures

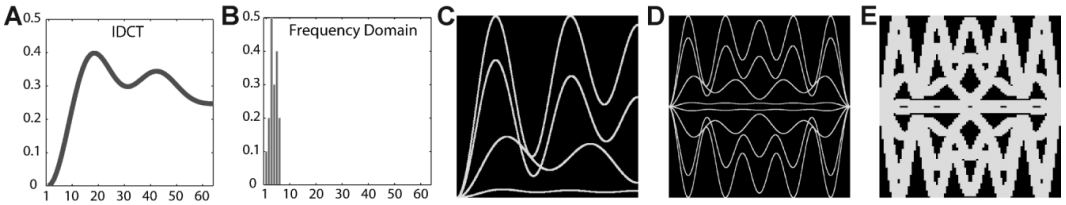

Figure 2.1: Illustration of the generation of a channel topology

A: An initial curve is formed by the IDCT of the vector of discrete cosine components shown in (B) where only the first six elements are allowed to take non-zero values. C: Four curves are superimposed followed by (D) symmetrical replication (up/down, left/right) to yield the final topology, (E).

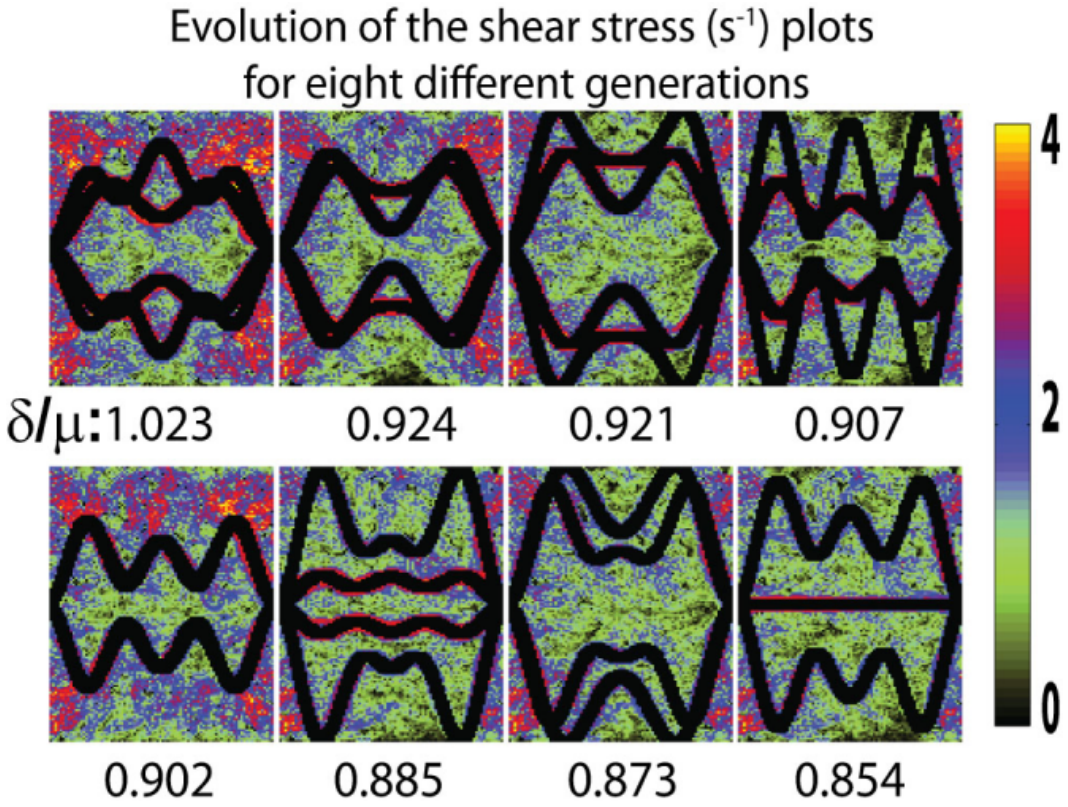

Figure 2.2: Shear stress distributions

for eight different generations sampled from a simulation comprising a total of 100 generations. The topology optimization by GA results in: (1) a target shear stress value of 1; (2) more uniform shear stress distribution around the target value.

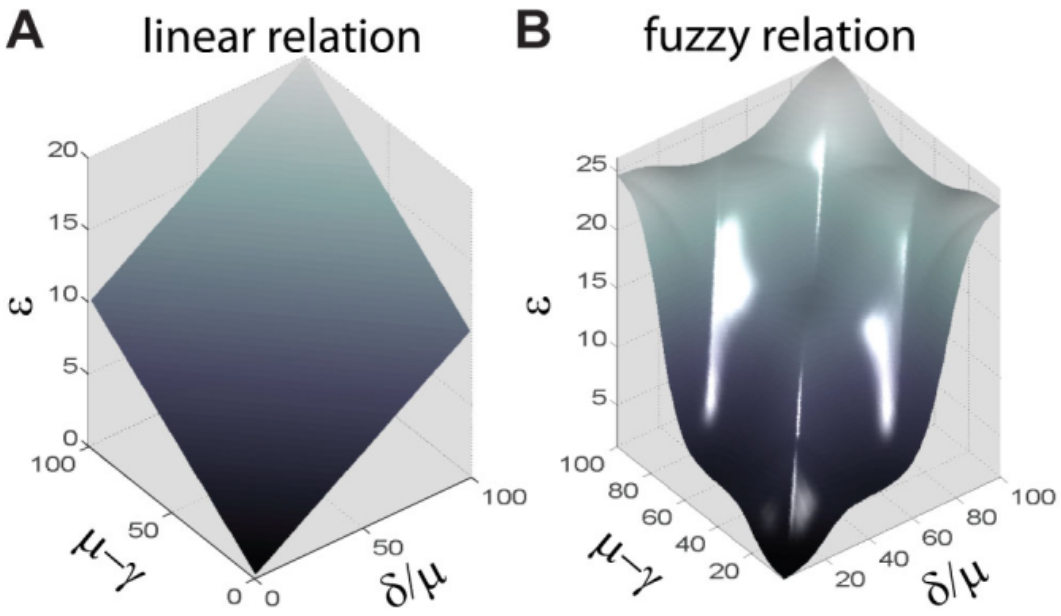

Figure 2.3: Comparison of error surfaces
in the case of the (A) linear relation versus (B) fuzzy relation.

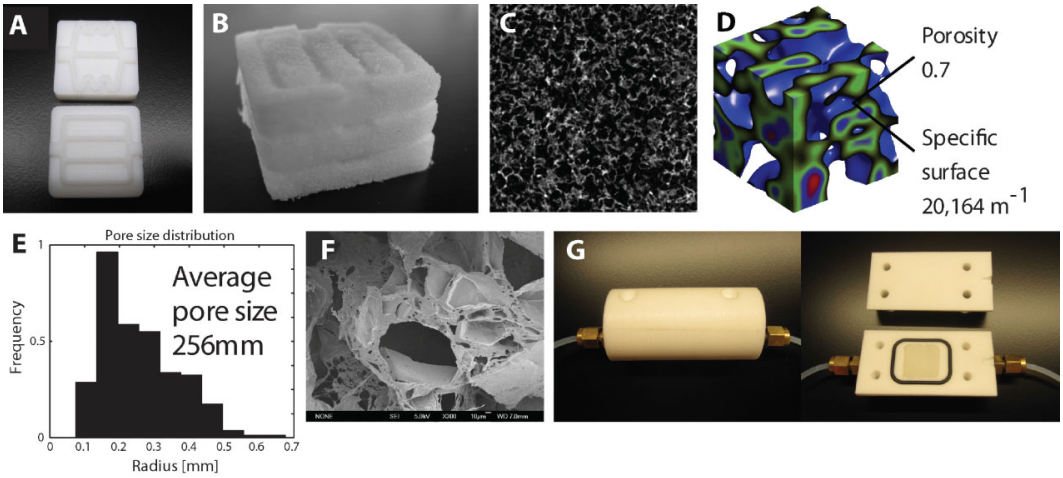

Figure 2.4: Flow Experiments on Polymeric Scaffolds

A: Teflon molds used to cast sugar/PCL paste and fabricate 3D scaffolds. B: The leached scaffolds are highly porous and can be stacked to achieve the desired thickness. C: X-ray μ-CT image revealing open porosity of scaffold, scanned at 13 μm resolution. D: 3D rendering of μ-CT data using MATLAB to estimate porosity and specific surface. E: Pore size distribution as determined by μ-CT data. F: Scanning electron micrograph of a cross-section representative of the scaffold. G: Teflon bioreactor shown assembled (left) and opened (right). The interior of the flow chamber is $2\times2\times1$ cm^3.

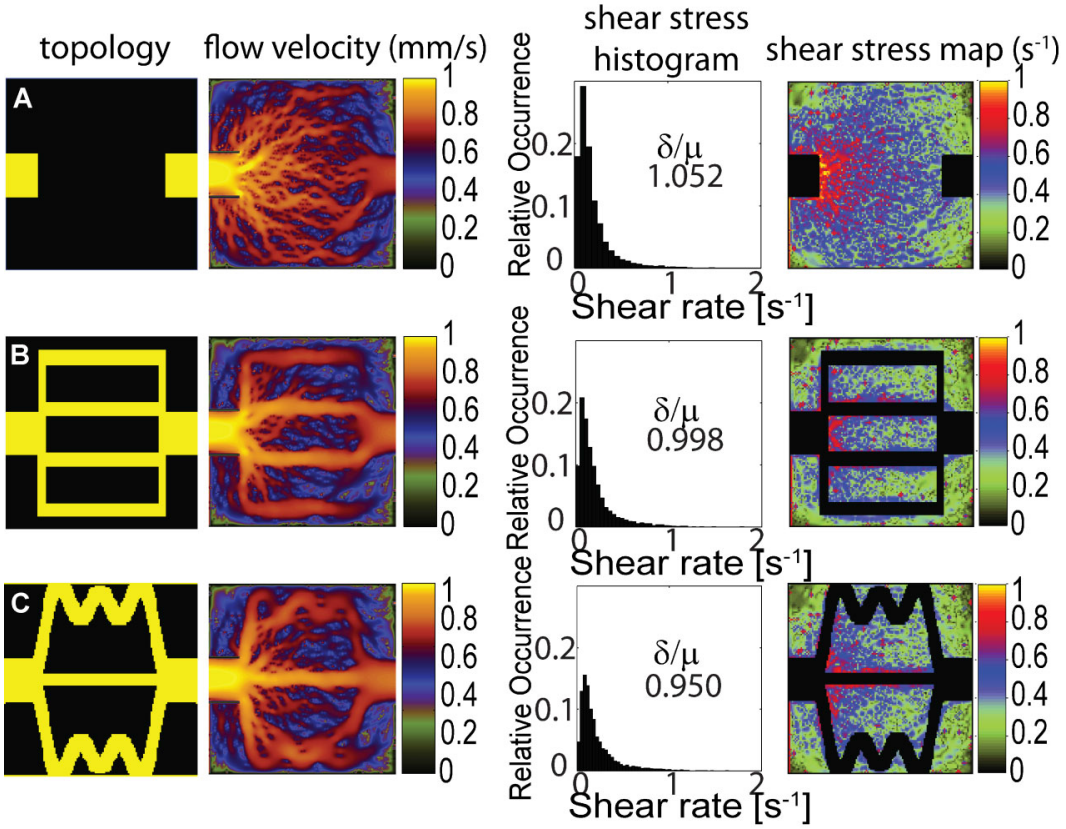

Figure 2.5: Computer simulation results

for channel topologies and corresponding input velocities: (A) no channel, (B) manual design, and (C) optimized channel design. The optimization is limited to a very small number of discrete cosine components to keep the design simple for machining purposes (see (Fig. 2.8) C). A better optimization utilizing more discrete cosine components is shown in (Fig. 2.7). For display purposes, the logarithm of the flow velocity and shear stress maps is shown.

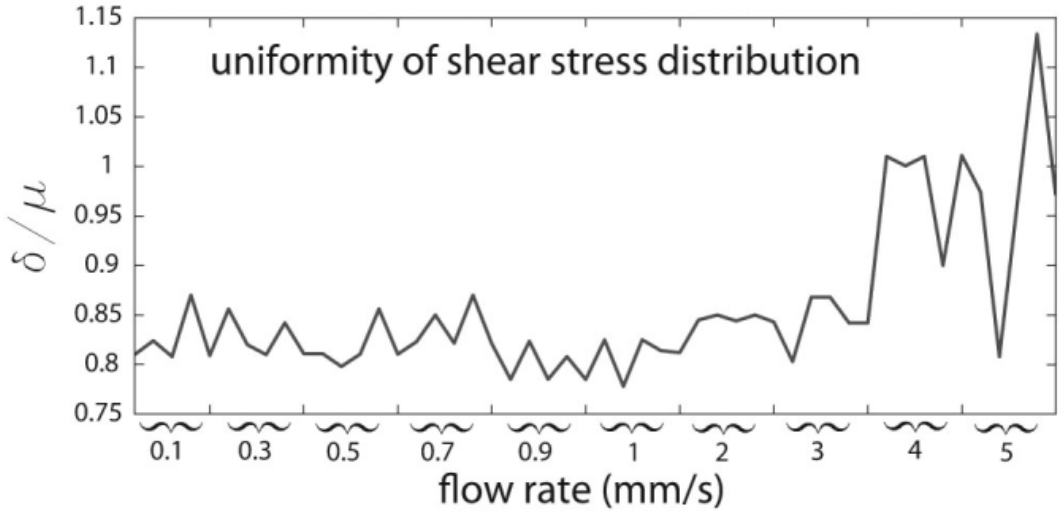

Figure 2.6: Uniformity of the shear stress distribution
for different flow rates corresponding to input velocity values ranging from 0.1 to 5 mm/s.

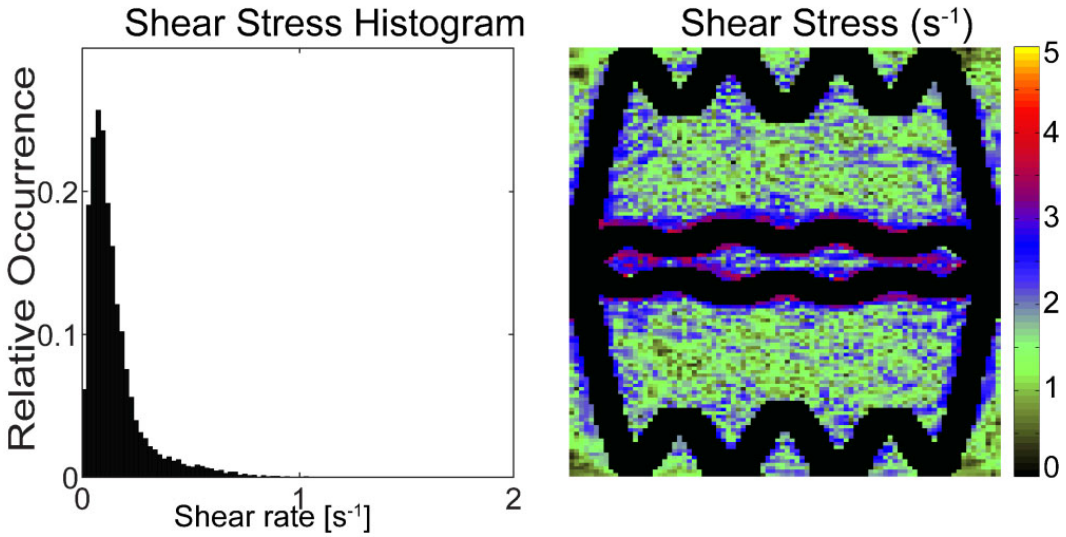

Figure 2.7: Optimized topology
shear stress histogram (left) and shear stress distribution (right). The flow rate was allowed to vary and more discrete cosine components were used than in the results of (Fig. 2.5). This resulted in improved performance.

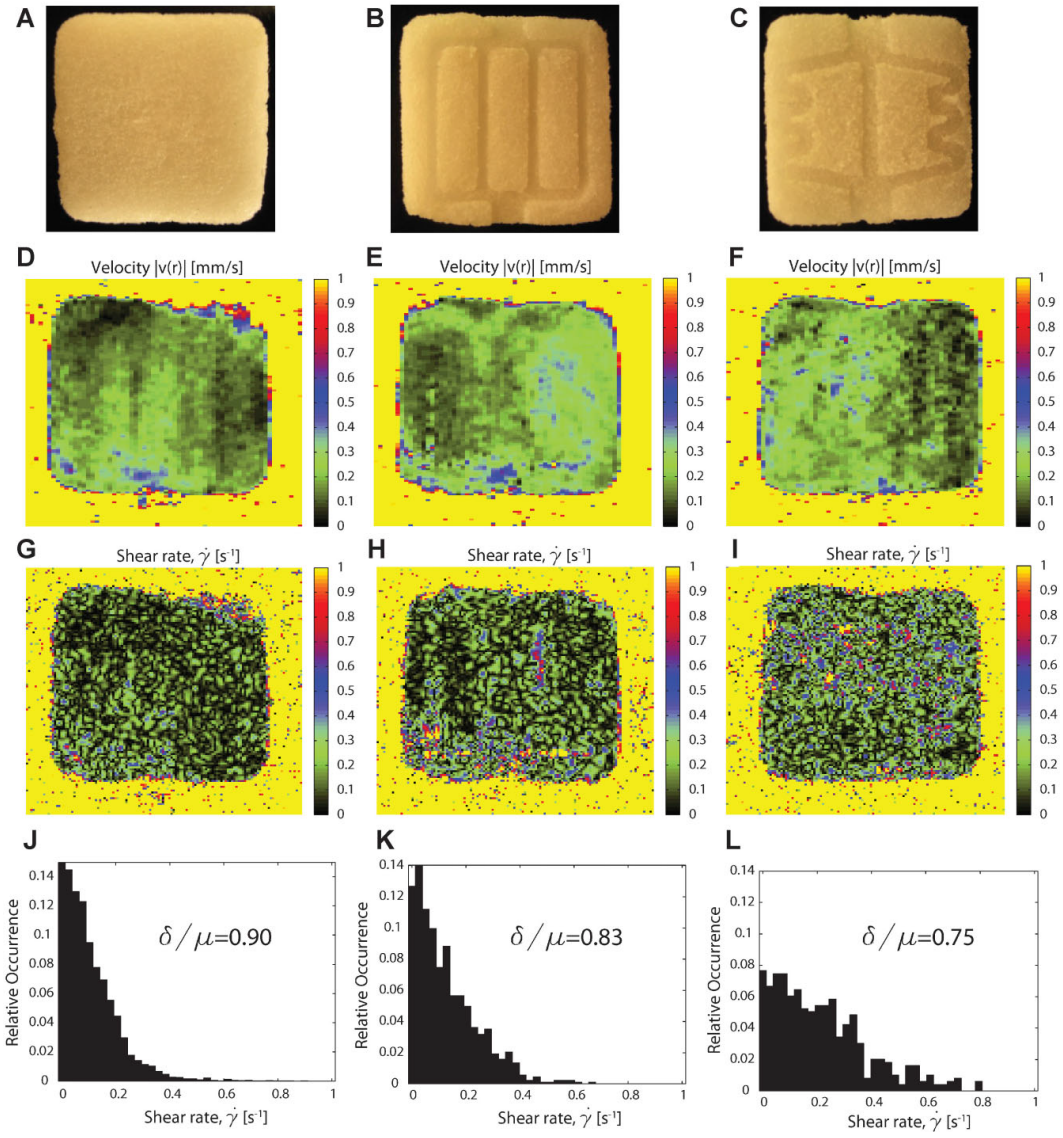

Figure 2.8: Comparison of MRI-derived shear rate distributions for water flowing in three different porous scaffolds: (A) no channel ($\frac{\delta}{\mu} \sim 0.90$), (B) manual design of the channel topology ($\frac{\delta}{\mu} \sim 0.83$), and (C) topology-optimized channel design ($\frac{\delta}{\mu} \sim 0.75$). In (A−C), a photograph of the 2×2 cm^2 porous scaffold shows the channel topology used in the experiments. The flow rate in each scaffold was 5 mL/min at the inlet and resulted in flow velocities in the range of 0−1mm/s inside the porous scaffold. Shear rate ($\dot{\gamma}$) distributions (G−L) inside the scaffold were derived from the appropriate flow map (D−F).

CHAPTER 3

Real-Time Maps of Fluid Flow Fields in Porous Biomaterials

Article published: Biomaterials Volume 34, Issue 8, Pages 1980-1986, March 2013

3.1 Introduction

Tissue engineering approaches to regenerative medicine generally employ bioreactors for the growth of tissues because the organization of individual cells into functional structures requires a 3D context [37, 13]. Biomaterials have been successfully used as 3D scaffolds to mimic specific biochemical and physical environments [23, 38, 39, 40]. Naturally, the optimization of tissue growth would benefit from methods to visualize changes occurring inside the biomaterial over time. This would enable the control of mechanical and biochemical inputs, such as fluid flow and transport of growth factors, cytokines and nutrients. However, to monitor and improve upon the effect of these transport processes within 3D biomaterials, it is essential to develop technology that can accurately determine local microflow profiles in real time.

Fluid flow, and in particular, shear forces associated with applied flow have been shown to enhance cell growth and differentiation [5, 41, 42]. In addition to shear stress, interstitial flow distributions bias the extracellular transport and gradients of growth factors, which in turn affect cell-cell signaling and morphogenesis [43]. It is difficult to experimentally determine flow profiles and flow

heterogeneity within a 3D matrix. A first step in solving this problem is to obtain spatially accurate distributions of velocity fields inside biomaterials. This information will be essential to study cellular responses to flow and evaluate flow-induced cell behavior. Furthermore, from the engineering standpoint, knowledge of the flow fields may reveal degradation/changes of the scaffold over time that can be extremely informative towards future optimizations.

Current approaches engage computational flow modeling to estimate patterns of flow and shear stress based on known or assumed material properties, primarily fluid permeability and porosity [44, 20]. However, computational predictions are of limited value compared to real-time measurements as material properties change over time. Although optical imaging techniques such as Doppler optical coherence tomography [45] have been employed, they are hindered by their reliance on optical transparency of the medium and/or the use of particle tracers[46]. As such, the applicability of optical methods is restricted in opaque and low permeability materials such as hydrogels.

Here we present a methodology which enables the visualization of interstitial flow within biomaterials. The nuclear magnetic resonance (NMR) technique is well suited for this task, due to its ability to probe material properties even in opaque media. This article evaluates the potential of NMR velocity imaging for investigating the hydrodynamic properties of biomaterials [47, 48, 49, 36] and to provide measurements of slow flows. We also discuss the derivation of shear, hydraulic pressure and fluid permeability maps from a single NMR velocimetry experiment.

3.2 Materials and Methods

3.2.1 Implementation of the NMR Velocimetry Technique

A spin-echo multi-slice (SEMS) sequence was modified to compensate for flow on all gradients except the phase encoding gradient. In the calculations for flow compensation, the full trapezoidal shape of the gradient pulses was used (as opposed to rectangular approximations). The sequence of pulses is shown in (Fig. A.3). Pairs of flow weighting (F.W.), trapezoidal bipolar gradients were added along x, y and z axes to select the gradient first moment (M_1). In the case of no flow, stationary spins experience positive and negative gradients of the same magnitude leading to a total phase accumulation from the first gradient equal and opposite to the phase accrued under the second gradient (zero net phase accumulation). In the case of constant-velocity fluid flow, spins move between the two gradients and phase cancellation is incomplete, leading to a residual phase that is proportional to the velocity [36].

Two experiments were performed under flow, with two gradient values: $+M_1$ and M_1, where the value of M_1 was chosen large enough to include the highest anticipated flow velocity and avoid phase wrap-around effects. These two experiments were subtracted to obtain a velocity map. This velocity map for slow flows in biomaterials, however, contains artifacts from gradient nonidealities (e.g., eddy currents and nonlinearities). Therefore, a flow/no-flow subtraction procedure was utilized to overcome this problem: for each velocity component, two scans were subtracted from experiments performed with flow on versus flow off. This yielded an error-corrected velocity map. With this approach, we are able to routinely acquire flow maps of the three orthogonal components of the velocity: $\overrightarrow{v} = (v_x, v_y, v_z)$ for voxels of typical size 90 μm $\times$ 90 μm $\times$ 1 mm with accuracy of 0.05 mm/s using a total of 12 scans: 3 gradient directions $(x, y, z) \times 2$ gradient reversals $(M_1, -M_1) \times 2$ runs (flow, no flow). The accuracy of this subtraction

technique was validated in experiments involving clear flow in a pipe and porous media flow against a known average flow rate, for several different flow rates down to 0.01 mm/s. In terms of temporal resolution, our technique yields 12 scans in 8 minutes of acquisition. The imaging slice (1 mm thick) was positioned in the middle of the flow chambers. The remaining imaging parameters were: TR=3s, TE=20ms, 128×128 matrix and field of view (FOV) = 12 mm × 12 mm.

3.2.2 Scaffold Fabrication

3.2.2.1 Porous PCL Scaffold

Porous polymer scaffolds were prepared by a porogen leaching method to achieve high porosity ($\tilde{9}0\%$). Sugar crystals (size distribution in the range 250-355 μm) served as the porogen and were added to a 20 wt. % solution of polycaprolactone (PCL; Polysciences) in dichloromethane and thoroughly mixed to form a viscous paste (14:1, sugar:PCL). The sugar/PCL paste was cast into a 1 cm diameter Teflon mold and compressed to a height of 3.5 mm with a plunger to uniformly distribute the paste within the mold and compress the sugar crystals, thereby resulting in an interconnected network of pores upon removal of the sugar. Once the paste was distributed and compressed, the scaffold was allowed to cure via solvent evaporation overnight followed by freeze-drying to remove any residual solvent. The scaffold was leached in deionized water for several days to remove the porogen.

3.2.2.2 Hydrogel Matrix

Hydrogel matrices were prepared using a mixture of Matrigel (BD Biosciences), fibrinogen (Sigma Aldrich) and type 1collagen (BD Biosciences) at a volume composition of 10%, 65% and 25%, respectively. To prepare the hydrogel, an aliquot of 10 mg/ml solution of collagen was first diluted in 10X Dulbecco's Modified

Eagle's Medium (DMEM; Sigma-Aldrich) and neutralized with 1M NaOH. A 10 mg/mL fibrin solution was prepared separately in serum-free 1X DMEM (Invitrogen). From the stock solutions, collagen, fibrin and Matrigel were mixed at the indicated volumetric ratios to yield a final concentration of 3 mg/mL collagen and 6 mg/mL fibrin. Solutions were kept on ice to prevent polymerization of collagen.Thrombin (Sigma Aldrich) was then added at a concentration of 50 U/mL to allow for crosslinking of the fibrin component. An aliquot of the hydrogel mixture was added to the appropriate flow chamber and allowed to polymerize at 37 $^{\circ}C$ for 45 minutes. This hydrogel mixture was chosen for sufficient mechanical strength and permeability to flow.

3.2.3 Material Analysis

3.2.3.1 Scanning Electron Microscopy (SEM)

Hydrogel samples were fixed for 24-48 hours in Karnovsky's fixative under agitation at 4 $^{\circ}C$. Samples were subsequently dehydrated in increasing concentrations of ethanol (30%, 50%, 70%, 80%, 90%, 96%, 100%) for 15-20 minutes at each concentration and then dried using critical point drying (Tousimis, Andromegasamdri-915-B). To image the hydrogel and PCL scaffolds, freeze dried samples were freeze fractured in liquid nitrogen. Fractured samples were then mounted, gold coated (Denton Desk V, HP Cold Sputter Coater w/ Etch Mode) and imaged (Nova 230 NanoSEM).

3.2.3.2 Micro-computed Tomography (μ-CT)

The μ-CT data for the porous PCL scaffold was obtained using a SkyScan 1172 scanner with 13 μm spatial resolution. The scan data was reconstructed using cluster reconstruction software (NRecon). For 3D visualization, a 550 μm thick section of the scaffold was selected and rendered using CTvol software to view the

interconnected pore network.

3.2.4 Flow Chambers

Two NMR-compatible flow chambers were constructed from Teflon with brass inlet and outlet. The fluid permeability flow chamber ((Fig. 3.1) A) was designed for the measurement of flow through porous media. The inlet and outlet screw directly into the body of the chamber, allowing for a tight seal and an accurate reading of the pressure drop across the sample. The sample region has a 10 mm-diameter circular cross-section with a height of 3.5 mm and is sandwiched by 2 large pore-density filters, each 3 mm thick. The sample region holds 500 μL of the hydrogel mixture or a 3.5 mm thick scaffold. The bioreactor vessel ((Fig. 3.2) A and B) has the shape of a cylindrical bioreactor with inner rectangular flow chamber with dimensions 2 cm $\times$ 2 cm $\times$ 1 cm and 1.5 mm diameter inlet/outlet. The upper component is sealed hermetically to the bottom component with an o-ring and nylon screws. For flow experiments, the hydrogel mixture (4.5 mL) was polymerized around a 1 mm diameter Teflon rod bridged between the ports of the inlet and outlet. Removal of the Teflon rod yielded a single 1 mm diameter channel at the center of the hydrogel matrix.

3.2.5 Flow Set-up

Continuous flow was facilitated either by syringe pump (Harvard Apparatus) or by pressurizing a sample cylinder (Swagelok), containing either 1X DMEM (without serum) or DI water. The syringe pump was connected directly to the inlet of the flow chamber via $\frac{1}{4}$ inch Teflon tubing. In this configuration, inlet velocities were controlled by user inputs. In the sample cylinder configuration, compressed nitrogen gas was used to drive fluid flow via precision micro-pressure regulator (Model PR4033-400, Ingersoll-Rand/Aro). A digital hydraulic pressure gauge

(Model DPG8000-30, Omega Engineering Inc.) was placed at the inlet and outlet of the flow chambers to provide readouts of the pressure drop. To image fluid flow, a NMR system consisting of a 9.4 T, 400 MHz vertical bore (89 mm diameter) Varian instrument and a 40 mm-i.d. imaging probe was used. Flow chambers were placed vertically in the imaging probe. Flow rates at the outlet were measured using the bucket-and-stopwatch method.

3.2.6 Shear Rate Calculation

From a 2D slice, the in-plane shear rate induced by the fluid flow in the porous matrix can be calculated from digital images using finite differences and the formula, $\dot{\gamma} = \frac{\sigma_x v_y + \sigma_x v_x}{2}$

3.2.7 Hydraulic Pressure Calculation

To derive hydraulic pressure maps, we first take the divergence of the steady-state Navier-Stokes equation (NSE) for an incompressible fluid and then obtain a Poisson-type equation for the pressure:

$$\nabla^2 p = -\rho \frac{\partial v_i}{\partial x_k} \frac{\partial v_k}{\partial x_i} = -\rho \frac{\partial^2 v_i v_k}{\partial x_k \partial x_i} \tag{3.1}$$

where the velocity field,$v(x)$, is known from MRI experiments,ρ is the fluid density and the pressure field p is subjected to Dirichlet boundary conditions at the inlet and outlet as measured by hydraulic pressure gauges and Neumann boundary conditions $\hat{n} \cdot \nabla p = 0$ at the vessel wall. Previously measured pressure gradients via MRI used the NSE directly [50]. However, pressures cannot be obtained reliably with this method due to the path-dependence that arises from integrating the pressure term. The reason for path dependence is because the velocity terms in the NSE (such as the convective and viscous terms) are not conservative fields. Solutions to the Poisson equation, such as used here, do not

suffer from this problem.

3.2.8 Fluid Permeability Measurements

The slow flow of an incompressible fluid through porous media is typically described by Darcy's law [51], $U(x) = -\frac{k}{\mu} \cdot \nabla p(x)$, where U is the average fluid velocity, ∇p is the applied pressure gradient,μ is the dynamic viscosity, and $\boldsymbol{k}$ is the fluid permeability tensor. In isotropic media, is taken to be a constant: $\boldsymbol{k} = k\boldsymbol{I}$, where is the unit tensor. The permeability k, which has dimensions of length squared, is roughly the effective pore channel area of the dynamically connected part of the pore space. In general, it must be measured since it cannot be obtained from simple pore statistics such as porosity or specific surface. A peculiar feature of the flow field is that only a subset of the pore space contributes to the fluid permeability. The fluid permeability is calculated as the ratio, $k = -\frac{Q\mu L}{A(p_2 - p_1)}$, where L is the length of the vessel, Q is the volumetric flow rate (customarily measured with the bucket-and-stopwatch method), A is its cross-sectional area and $(p_2 - p_1)$ is the pressure drop between the inlet and outlet.

3.2.9 Fluid Permeability Maps

The permeability maps are obtained by using Darcy's law, $U(x) = -\frac{k}{\mu} \cdot \nabla p(x)$, where U is fluid velocity, ∇p is pressure gradient, μ is the dynamic viscosity, and k is the fluid permeability (scalar). Flow maps obtained from MRI measurements are used to determine local velocity values. Since the resolution of the MRI flow maps is comparable to pores sizes, the resolution is scaled down four-fold. The derived pressure maps are used to determine local pressure gradients. Average permeability is calculated by taking the arithmetic mean of the map horizontally (main direction of flow), and the geometric mean vertically as described by the following equation, $K = \frac{m}{n}\left(\sum_{i=1}^{m}\left(\sum_{j=1}^{n} k_{i,j}\right)^{-1}\right)^{-1}$, where K is the average

permeability of the map and $k_{i,j}$ is the local permeability at the i^{th} row and j^{th} column, such that the map consists of m rows and n columns.

3.3 Results and Discussion

3.3.1 Hydrodynamics in a Porous PCL Scaffold

Flow experiments were performed using a NMR-compatible custom flow chamber which features a 3.5 mm thick sample chamber sandwiched between two filter discs ((Fig. 3.1) A). The first biomaterial studied was a porous polymer scaffold made from polycaprolactone (PCL) via porogen leaching method to yield interconnected pores with average pore size of 256 μm as determined by micro computed tomography (μ-CT). Micro-CT-derived 3D rendering of the scaffold ((Fig. 3.1) B) and scanning electron microscopy (SEM) images ((Fig. 3.1) C) demonstrate large interconnected pores as well as micropores in the scaffold walls. Two different volumetric flow rates, 5 ml/min and 10 ml/min, were applied via syringe pump to the PCL scaffold in the flow vessel and a MRI scan was used to probe the corresponding velocity field $\overrightarrow{v} = (v_x, v_y, v_z)$ with voxel size (spatial resolution) of 90 μm $\times$ 90 μm 1 mm. A map of the flow speed (magnitude of the velocity) is shown in (Fig. 3.1) D and H where dotted regions indicate the location of the PCL scaffold. For both applied flow rates, the resulting flow fields show substantial heterogeneity due to the scaffold's internal pore network and the chamber wall boundaries. The shear rates were derived from the gradient of the velocity field and plotted ((Fig. 3.1) E and I) to further evidence the fluid flow heterogeneity within the scaffold. From the velocity data and the measured hydraulic pressure drop across the flow chamber region, we were able to map the internal hydraulic pressures ((Fig. 3.1) F and J) relative to the outlet pressure.

Using the velocity and hydraulic pressure maps, the fluid permeability can be computed through the scaffold region as the ratio of velocity to pressure gradi-

ent. Fluid permeability is a fundamental property of a porous medium and is of paramount importance in the context of biomaterials where flow plays a critical role [52, 53]. Permeability is traditionally measured by flowing liquid into a chamber containing the porous medium. The volumetric flow rate resulting from an applied pressure differential establishes a proportionality constant between the two. However, permeability determined by a bulk measurement will not reflect any local variability due to inhomogeneous flow and/or matrix property changes. The fluid permeability maps for the PCL scaffold is displayed in (Fig. 3.1) G and K. Using the traditional method of the bucket-and-stopwatch, we calculated a value for the fluid permeability across the PCL scaffold material to be $(3.9 \pm 1.0)\, 10^{-13} m^2$. This is to be compared with the average permeability values calculated for the scaffold region from the fluid permeability maps: $(2.82 \pm 0.29) \times 10^{-13}\ m^2$ and $(2.69 \pm 0.19) \times 10^{-13}\ m^2$. In the presence of macroscopic fluctuations, fluid permeability becomes a local property of the material that should be averaged over a region larger than the pore size, but smaller than the scaffold. The MRI flow mapping technique elucidates these local variations that otherwise would be averaged out in a bulk measurement. As will be discussed below, this type of local flow measurement is especially useful when the material properties change over time due to variations on cell growth, increase in matrix production and/or degradation of the scaffold.

3.3.2 Microflows in a Biopolymer Hydrogel

We next applied the flow mapping technique to biopolymer hydrogels. Hydrogels are commonly used in 3D cell culture owing to their semblance to extracellular matrix and the ability to functionalize the fibrillar components to aid cell growth [54, 55] however their fragility and low permeability make it challenging to implement flow. The low permeability is due to the small pore size of the interconnecting fibril network and therefore fluid flow is typically described as in-

terstitial [25]. Additional issues arise when applying flow to small pore (<5 μm) hydrogels including erosion of the fibril network and even collapse of the network under high fluid pressures. To investigate these effects, we formed a biopolymer hydrogel consisting of a mixture of fibrin, collagen and Matrigel inside the 1 cm diameter flow chamber ((Fig. 3.1) A). For this gel, an applied input flow rate of 50 μl/min was sufficiently low to prevent collapse of the hydrogel under the applied fluid pressure. From the velocity field data, the velocity components (v_x, v_y, v_z) could be resolved in the horizontal, vertical and through-plane flow directions, respectively ((Fig. 3.2) A). As evidenced by the larger magnitude of the v_x component we note that primary flow occurs along the horizontal direction, which was the direction of the applied fluid pressure, and therefore velocity components orthogonal to the main direction of flow are small.

When the applied input flow rate was increased to 100 μl/min, irreversible compression occurred in the gel ((Fig. 3.2) B right image). The flow maps show a significant change in the hydrodynamic response under the 100 μl/min applied flow rate, as compared to the same hydrogel at 50 μl/min, by erosion of the gel under the applied fluid pressure ((Fig. 3.2) B). This compression was confirmed by SEM ((Fig. 3.2) C). Prior to flow, the collagen and fibrin fiber networks are observed to be fairly uniform ((Fig. 3.2) C left image). After the application of high flow, however, the hydrogel was severely compressed and the SEM images show an accordion-like collapse of the gel under the direction of applied flow that is reminiscent of the lamellar structure of elastin fibers in a blood vessel ((Fig. 3.2) C right image). The flow mapping technique successfully depicts structural integrity of the gel. Without such maps, it is nearly impossible to determine whether changes to the material have occurred unless the experiment is stopped and the gel is sectioned.

3.3.3 Interstitial Flow from a Channel

The ability to map perfusion flows in the region surrounding a central channel was investigated by forming the same biopolymer hydrogel in a Teflon vessel containing a rectangular flow chamber ((Fig. 3.3) A and B) with 1.5 mm diameter inlet and outlet. A single channel (1 mm diameter) was patterned in the hydrogel during polymerization to match the inlet/outlet of the chamber walls. This geometry yields a model of perfusion from a main flow source to mimic interstitial flow. The channel provided a primary path for fluid to investigate the path of perfusion flows into the surrounding gel matrix from the central flow channel ((Fig. 3.3) C). A volumetric flow rate of 5 ml/min was applied for 15 and 60 minutes, respectively. The velocity maps $\overrightarrow{v} = (v_x, v_y, v_z)$ acquired under steady flow conditions and are shown in (Fig. 3.3) D and E. Even after 15 minutes of applied flow, non-negligible erosion of the hydrogel is seen near the inlet leading to escape of fluid through the region between the gel and vessel wall as evidenced by the larger v_y component and visual inspection of the hydrogel post-experiment. Escaping fluid is seen at the bottom of the v_x component map and erosion leading to vertical flow is seen in the v_y component.

The flow mapping technique enables the visualization of time-dependent erosion of the gel. With extended perfusion of the hydrogel for 60 minutes, erosion was substantial, as seen in the velocity maps for v_x, v_y, v_z ((Fig. 3.3) E) and enlarged central channel width. Flow was observed to be a combination of perfusion flow through the gel matrix and time-dependent erosion around the channel. Flow speeds in the range 0.1 - 0.8 mm/s were measured in the matrix region surrounding the channel ((Fig. 3.3) F). Fluctuations were estimated by taking the standard deviation of the flow field in regions far away from the flow channel, where flow was weakest and mostly uniform, to yield an error bar of 0.05 mm/s. A one-dimensional profile around the channel, averaged along the direction of flow, was plotted to show that flow was detected outside the channel ((Fig. 3.3) G). These

flows result in erosion of the hydrogel with time as evidenced by the NMR imaging technique and confirmed by sectioning and visual inspection of the hydrogel post-experiment. As expected, shear rates ((Fig. 3.3) H) were seen to be strongly peaked near the channel.

3.3.4 Implications of the Results

The capability to non-invasively measure fluid flows in 3D biomaterials in real time is a technological advantage that can be applied to many fields including tissue engineering, cell biology and biomechanics. Prior to this study, the conditions of the biomaterial would typically be assessed at the beginning and end points of an experiment; and the flow conditions were mainly obtained through computer modeling or through simple input-output relationships. The phase-cycled, subtraction NMR technique yields accurate, non-invasive measurements of fluid flow and scaffold integrity in real time during an experiment. We anticipate that this will be an important tool in regenerative medicine, where the state of a 3D cell culture in a bioreactor can be monitored over time, enabling real-time control of external inputs in order to provide optimal conditions for cellular development. Experimental determination of flow is especially important when the flow is complex, as is the case with biomaterials characterized by variable porosity and tortuosity, because such systems are difficult to model computationally. The method can also assess the deformability of a biomaterial. Such deformability effects can be problematic, as they often lead to unwanted transient or even permanent effects. Examples of such effects include compression of the material, erosion of the matrix and escape of fluid along the walls of the flow chamber. Real-time fluid flow maps in these systems provide windows into the condition of the system.

3.4 Conclusion

In this article, we have successfully generated detailed flow maps in biomaterials. We have demonstrated how to derive, from a single experiment, local hydrodynamic properties such as shear rate, hydraulic pressures and fluid permeability as well as important structural information about the biomaterial condition. We anticipate applications of the imaging technique to 3D cell cultures in establishing quantitative relationships between fluid mechanics, cell growth and organization. The main strength of the NMR approach is the ability to map flows in optically opaque media, regardless of the presence or absence of vascular networks. The NMR readout provides a volume-average flow measurement (averaged over the image voxel, which is much smaller than 1 mm^3, in the present study), making it an ideal technique for probing interstitial flows.

Acknowledgements

This study was supported by funds from the UCLA Stem Cell Center Innovator Award to M.L.I.A. and the Ruth L. Kirschstein National Research Service Award (T32HL69766 to J.J.M. and O.D.V.N.). Research funding to L.S.B. from the Camille & Henry Dreyfus Foundation and the Arnold and Mabel Beckman Foundation are acknowledged. The authors have no conflicting financial interests to disclose. L.S.B., J.M., M.L.I.A. designed experiments; L.S.B., J.M., O.N., M.L., performed experiments; L.S.B., A.W., O.N., J.M. designed the flow chambers; L.S.B., J.M. conducted data processing and error analysis; J.M., L.S.B. wrote and tested flow pulse sequence; K.Y. performed computations on the fluid flow field; L.S.B., J.M., M.L.I.A. wrote the paper. The authors declare no competing financial interests. We thank Melody Swartz and Petros Koumoutsakos for critical reviews on the manuscript.

Figures

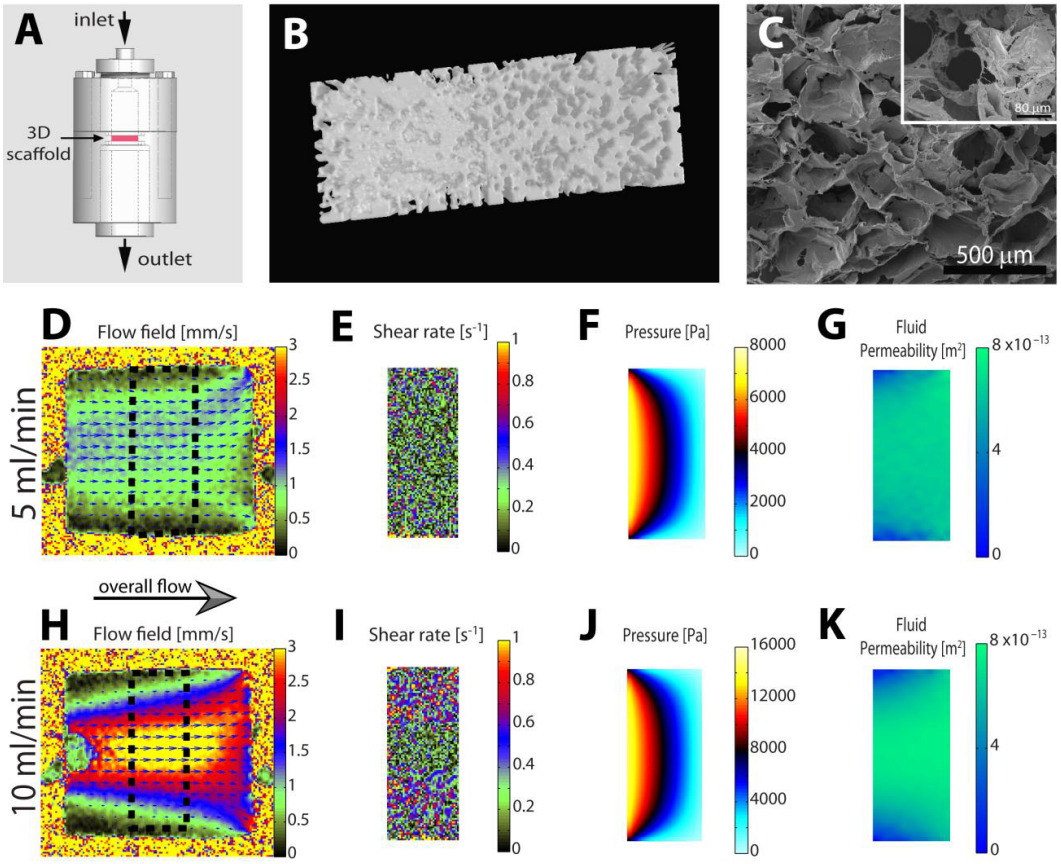

Figure 3.1: Spatially-resolved measurements of flow and fluid permeability for a porous polymer scaffold.

(A) Schematic of the flow chamber with indications for the fluid inlet, outlet and location of the porous scaffold. (B) 3D rendering based on μ-CT scan data for a 550 μm thick section of the porous polymer scaffold used in the permeability experiments. (C) SEM images of the porous network with evidence of micropores (see inset) in the struts of the scaffold pores. Applied flow rates: (D-G) 5 ml/min and (H-K) 10 ml/min. From left to right: (D,H) MRI flow fields where the blue arrows indicate the direction of flow; (E,I) shear rate maps derived from the flow maps; (F,J) pressure field calculated from the MRI flow map; (G,K) corresponding fluid permeability maps computed from the pressure and velocity fields. The scaffold (3.5 mm thick) region is indicated in (D,H) by dotted lines, sandwiched by two filters. The average fluid permeability calculated for the scaffold region in (G) is $2.6910^{-13}m^2$ and $2.8210^{-13}m^2$ in (K).

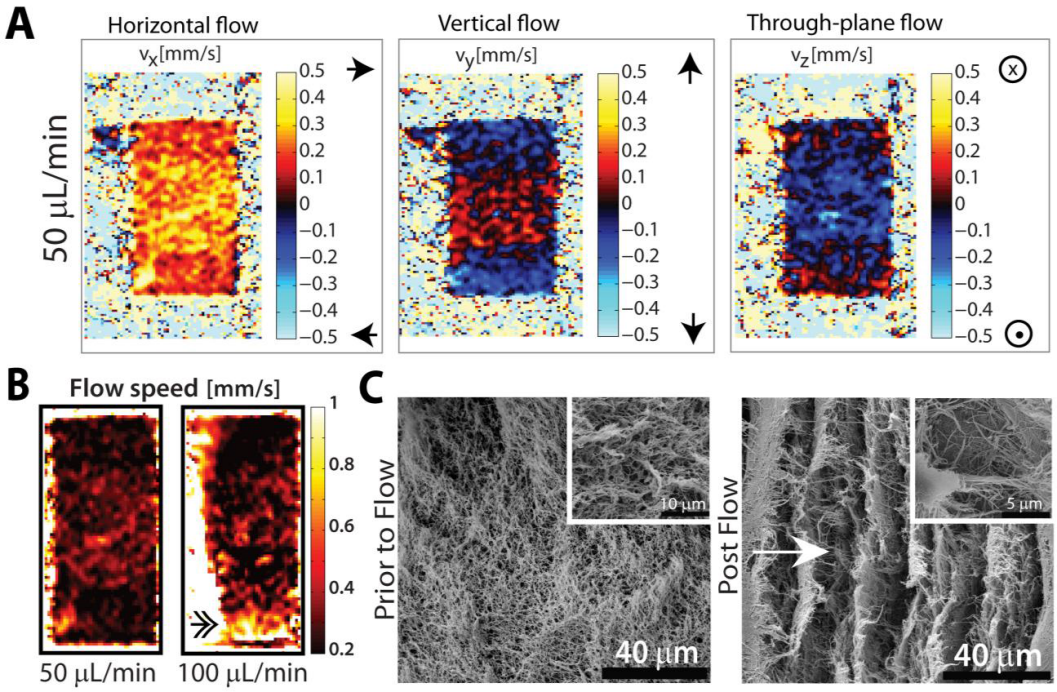

Figure 3.2: MRI visualization of flow fields in a hydrogel matrix.

(A) MRI flow maps are plotted for the v_x, v_y, v_z velocity components of an applied flow rate of 50 μL/min. For the flow maps, the field of view was 6 mm $\times$ 12 mm with an imaged slice thickness of 1 mm. (B) Maps of flow speed are plotted for the same hydrogel under both 50 μl/min and 100μl/min applied flow rates. (C) SEM images of the hydrogel matrix prior to and post flow where compression of the hydrogel is clearly observed by collapse of the fibril network.

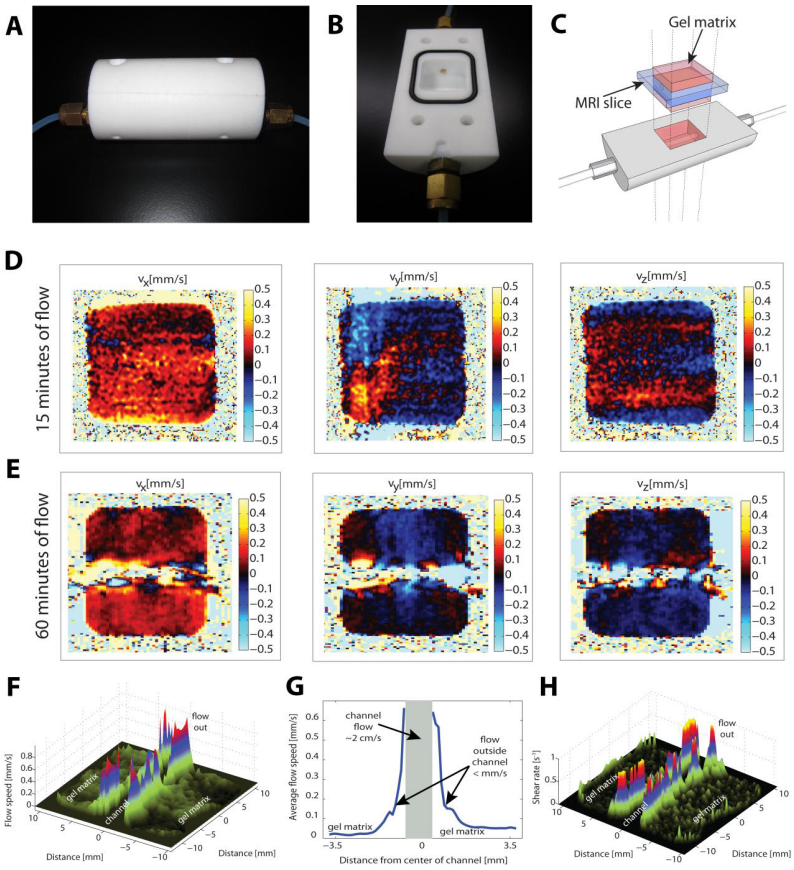

Figure 3.3: Flow mapping in a channeled hydrogel.

Bioreactor displayed (A) sealed and (B) opened to reveal the 2 cm × 2 cm × 1 cm flow chamber. (C) Schematic of the bioreactor indicating the channeled hydrogel matrix within the flow chamber. (D) Vector components (v_x, v_y, v_z) of the MRI-derived flow map after 15 minutes of applied 5 mL/min fluid flow in the channeled hydrogel. Overall flow is from left to right, as can be seen by the v_x component, which dominates over other components. (E) Vector components of the MRI-derived flow map after 60 minutes of applied flow. Flow speed is obtained from the velocity maps and averaged along the direction of flow to yield (F) flow map and (G) 1D profile of average flow speed from the central channel out into the surrounding hydrogel matrix. Flow velocities within the channel, well in excess of 2 cm/s, are not shown due to being too far off scale. (H) The shear rates $(\dot{\gamma})$ were derived from the flow map to yield a shear rate map inside the hydrogel.

CHAPTER 4

Feature-Preserving Noise Removal

Article will appear in next issue of IEEE Transactions on Medical Imaging

4.1 Introduction

Theprocess of removing noise in an image requires *a priori* knowledge of the noise distribution. In Wiener filtering, for example, the optimal filter requires knowledge of the power spectra of signal and noise [56]. It is common practice to perform denoising in some transform domain where the signal is sparse and discard the transform coefficients that do not overlap with those containing the signal [57, 58, 59]. Multi-resolution techniques such as the wavelet transform can achieve better sparsity and separation of the noise depending on the type of basis element that are used and the overcompleteness of the basis [60, 61]. A recent method, Block-matching and 3D filtering (BM3D), also termed "collaborative filtering" has shown state-of-the-art performance by grouping patches that look similar into 3D blocks and performing transform domain filtering of each 3D similarity block [62]. These algorithms yield outstanding performance as long as the images obey certain conditions consistent with algorithm assumptions, such as noise statistics or the type of patterns contained in the image. If the assumptions are not met, this can give rise to artifacts or losses in the image fine structure [63].

From an experimental standpoint, the simplest method to reduce noise is signal averaging. In the case of additive white Gaussian noise (AWGN) signal averaging

can be performed until the mean value of the signal is sufficiently high relative to the noise. In magnetic resonance imaging (MRI), signal transients from each phase-encode step are acquired and averaged to produce a mean-value image with higher signal-to-noise ratio (SNR). In photography, the analogous process consists of increasing the exposure time. The "mean-value" image in the case of AWGN generally reveals better contrast-to-noise. In the general case, there are three potential problems with the signal-averaging approach. First, there is an implicit assumption about the noise statistics. The mean value is the best estimate of the signal only in special cases. For example, signal averaging fails to remove multiplicative noise. Second, averaging does not account for spatial correlations in the image. Third, in the process of averaging data from multiple transients into single values (the mean values), much information is discarded about the noise, whose true nature can only be revealed upon realizations of the random experiment.

In this paper, we propose to denoise each pixel of an image using a nonlinear filter that operates along a data block which consists of patch neighborhoods of a pixel and multiple copies of the same image. Thus, an image pixel is denoised using information from one block of size $D = r \times N_p$, where N_p is the number of pixels in a patch and r is the number of copies. In this study, we use rectangular patches of size $N_p = 17 \times 17$ and $r = 7$ copies of the image. The nonlinear filter is designed based on multilayer perceptrons (MLP), which have been shown to be universal function approximators [64]. The purpose of operating along patches is to account for possible spatial correlations in the random field of the image. The method makes no assumptions about the noise statistics. There is also no guesswork involved in determining suitable thresholds, parameters or dictionaries. Because every algorithm deserves an acronym, we propose to call it multiple copies-multilayer perceptrons (MC-MLP).

Our test case will be MRI images, due to the common practice in MRI to per-

form signal averaging as a way to improve SNR. Poor SNR in MRI may arise in low magnetic fields or during the detection of metabolites or insensitive nuclei such as ^{23}Na. Such an MRI experiment can be readily configured to save individual transients separately to provide multiple copies of the same image. Magnitude-mode MRI images typically feature Rician noise. We validate MC-MLP by comparing its performance to the total variation method (TV) [65, 66], and a Rician noise MRI implementation of the state-of-the-art denoising algorithm Block-matching and 4D filtering (BM4D) [67], which performs collaborative filtering. MC-MLP outperforms both methods in terms of conventional methods such as peak signal-to-noise ratio (PSNR), feature similarity (FSIM), and mean structural similarity (MSSIM), as well as subjective visual quality metrics [68, 69]. We also demonstrate excellent performance in the removal of multiplicative noise, illustrating the independence of the method to the type of noise.

4.2 Denoising by multiple copies

4.2.1 Nonlinear filter design with MLP

An image Y is a mapping

$$Y : \{1, \ldots, N_x\} \times \{1, \ldots, N_y\} \to \mathcal{S}, \tag{4.1}$$

where $\mathcal{S}$ is a set of allowed pixel values. The Cartesian product $\{1, \ldots, N_x\} \times \{1, \ldots, N_y\}$ will be indexed by $t = (i, j)$. An experimentally measured image contains noise and is therefore a random field X whose realization is denoted by $X(\omega)$. It is convenient to denote $X_t(\omega)$ the value of the t-th pixel in the image $X(\omega)$ (a matrix of size $N_x \times N_y$). The probability space is $(\Omega, \mathcal{F}, P)$, where $\omega \in \Omega$ and Ω is the set of all possible outcomes

$$\Omega = \{I_1, I_2, \ldots, I_{N_I}\}, \quad N_I = (\#\mathcal{S})^{N_x \times N_y} \tag{4.2}$$

where I_j $(j = 1, \ldots, N_I)$ is a $N_x \times N_y$ matrix whose elements belong to the set $\mathcal{S}$ and $I_i \neq I_j$ for $i \neq j$. There is no fundamental restriction on $\mathcal{S}$. For example, if $\mathcal{S}$ is a discrete set such as $\mathcal{S} = \{0, 1, \ldots, 255\}$, then the σ-algebra on Ω can be taken to be the power set $\mathcal{F} = 2^{\Omega}$ whereas if $\mathcal{S}$ is a continuous interval such as $[0, 255]$ then Ω can be taken to be the Borel σ-algebra generated by the interval.

P is a probability measure depending on the nature of the experiment and could be unknown or arbitrary. Its structure can be inferred from individual realizations $\omega_1, \omega_2, \ldots$. For fixed $\omega \in \Omega$, the mapping $X_t(\omega) \to \mathcal{S}$ as a function of t yields a realization (sample path) of the random field. The sample path is an image in the sense of the mapping (A.1).

Denoising by multiple copies is the task of finding an estimator $\tilde{Y}$ of the true image Y given the prior information from r realizations of the sample path $\{X(\omega_1), X(\omega_2), \ldots, X(\omega_r)\}$ where $X(\omega) = \eta(\omega, Y)$, such that $\tilde{Y} \approx Y$ according to a suitable distance metric. Here, $\eta(\omega, \cdot)$ stands for the noise function, which is determined by the probability measure, P. An example of $\eta(\omega, \cdot)$ is AWGN, which takes the form, $X(\omega) = Y + \Gamma(\omega)$, where $\Gamma(\omega)$ is a $N_x \times N_y$ matrix of random values that are Gaussian-distributed.

To obtain a good estimate $\tilde{Y} \approx Y$, we must reduce the uncertainty of the estimate. There are at least two ways to do this. The first is to look at more instances of X, for example, by having several copies of the same image, i.e. $\{X(\omega_1), X(\omega_2), \ldots, X(\omega_r)\}$ where r is sufficiently large. Since $X(\omega)$ varies with each instance ω according to the noise distribution whereas Y is independent of ω, the more instances ω we have, the more certain we can be about the value of our estimate. Although in practice, this increases computational cost, we found that $r = 7$ is a good compromise.

The second approach to obtain more information is to look at the neighborhood of the pixel $X_t(\omega)$. We will denote the coordinates of the neighborhood by the set of points in a square region U_t centered on $t = (i, j)$:

$$U_t = \{t' = (i', j') | i' \in [i - d, i + d], j' \in [j - d, j + d]\} \tag{4.3}$$

U_t is referred to as a patch in the text below, where d is the number of pixels included away from the pixel's coordinates $t = (i, j)$. This patch region U_t contains $N_p = (2d+1)^2$ pixels. When denoising images it is important to account for spatial correlations in the random field $X(\omega)$ due to the shape of the deterministic function Y, or possible spatial correlations in the noise function (if any). Modern denoising algorithms operate on patches on the premise that images contain specific shapes and patterns such as curves, edges and plain surfaces. Denoising algorithms often take the mean of all available copies and recognize one or more type of low level patterns in patches such as edges. Yet a general searching algorithm can learn higher level patterns and can increase certainty by taking separate copies into consideration rather than just their mean.

The nonlinear filter design utilizes the information from r copies and a patch U_t centered on the pixel X_t with neighborhood distance d and finds a function

$$\exists f(X(\omega_1)|_{U_t}, X(\omega_2)|_{U_t}, \ldots, X(\omega_r)|_{U_t}) = \tilde{Y}_t^{\circ} \tag{4.4}$$

where $X(\omega_1)|_{U_t}$ denotes the restriction of the matrix $X(\omega_1)$ to the U_t neighborhood. It is a $(2d + 1) \times (2d + 1)$-dimensional matrix with entries taking values in $\mathcal{S}$. $\tilde{Y}^{\circ}$ is the best estimate of Y that can be obtained from the information provided by all r copies $X(\omega_1)|_{U_t}, X(\omega_2)|_{U_t}, \ldots, X(\omega_r)|_{U_t}$ of the U_t neighborhood, for all such neighborhoods ($\forall t$). The r two-dimensional matrices $X(\omega_1)|_{U_t}, X(\omega_2)|_{U_t}, \ldots, X(\omega_r)|_{U_t}$ are reshaped into 1D vectors of length N_p, then concatenated into a 1D vector of length $D = r \times N_p$ denoted by $\vec{x}_t$ and used inputs

to a MLP whose transfer function is a hyperbolic tangent. Thus, a MLP with D inputs, K outputs, one hidden layer with M nodes yields a K-dimensional output vector $\vec{\tilde{y}}_t$ whose k-th component is given by the iterated hyperbolic tangents:

$$\tilde{y}_t(k) = \tanh\left(\sum_{l=0}^{M} \theta_{l,k}^{(2)} \tanh\left(\sum_{j=0}^{D} \theta_{j,l}^{(1)} x_t(j)\right)\right) \tag{4.5}$$

where $z(l) = \tanh\left(\sum_{j=0}^{D} \theta_{j,l}^{(1)} x_t(j)\right)$, for $l = 0, \ldots, M$ are the outputs of the hidden layer. We use the convention where $x_t(0) = 1$ and $z(0) = 1$, so that $\theta_{0,l}^{(1)}$ and $\theta_{0,k}^{(2)}$ represent biases to the transfer function. The generalization to arbitrary numbers of hidden layers is straightforward by nesting additional hyperbolic tangents. The calculation of the vector $\vec{\tilde{y}}$ is called feed forward propagation. In our implementation, each MLP has a single output corresponding to a single pixel in the image. Thus, $K = 1$ and we may drop the vector notation, writing $\tilde{y}_t$ instead of $\vec{\tilde{y}}_t$.

Let $\vec{\Theta} = \left[\theta_{j,l}^{(1)}|_{j=0..D}^{l=0..M}, \theta_{l,k}^{(2)}|_{l=0..M}^{k=0..K}\right]$ be a vector of length m containing weights and bias values for all the nodes. The MLP is trained to solve for f by searching for an optimal $\vec{\Theta}$ that minimizes the sum of square errors in (4.6)

$$E(\vec{\Theta}) = \frac{1}{2}\sum_{s=1}^{S} e_s(\vec{\Theta})^2, \tag{4.6}$$

where $e_s(\vec{\Theta}) = y_s - \tilde{y}_s$ is the MLP error corresponding to sample s for a given set of MLP parameters. Here, y_s is the desired target value from a low noise image for an input vector $\vec{x}_s$ from a noisy training sample, and $\tilde{y}_s$ is the MLP estimate of y_s. While the coordinate t is a suitable index in the feed forward phase where every pixel in the image is processed, we denote it by s in the training phase where an error is calculated. Training samples are input-output pairs $(\vec{x}_s, y_s)$ picked from training images in no specific order where the entire image or only parts of the image might be used for training. s corresponds to the sample number in the training dataset. S is the total number of training samples.

The algorithm minimizes the errors at all nodes. Back-propagation uses the output error in (4.6) to determine errors of individual nodes in the remaining layers [70]. From $n(k) = \sum_{l=0}^{M} \theta_{l,k}^{(2)} \tanh\left(\sum_{j=0}^{D} \theta_{j,l}^{(1)} x(j)\right)$, the partial error for a weight $\theta_{l,k}^{(2)}$ is obtained by

$$\frac{\partial E(\vec{\Theta})}{\partial \theta_{l,k}^{(2)}} = \frac{\partial E(\vec{\Theta})}{\partial n(k)} z(l). \tag{4.7}$$

The error at a node is determined by taking into account the sum of partial errors of weights for all connections emanating from it. $\vec{\Theta}$ is iteratively updated using Levenberg-Marquardt search [71]. $\Delta\vec{\Theta}$ is calculated at each iteration using the update rule,

$$\Delta\vec{\Theta} = -[J^T J + \mu \mathbf{1}]^{-1} J^T \vec{e}, \tag{4.8}$$

and is added to $\vec{\Theta}$. $\vec{e} = (e_1, e_2, \ldots, e_S)$ is a vector of MLP errors for all samples.

Here, $\mathbf{1}$ is an identity matrix and

$$J = \begin{bmatrix} \frac{\partial e_1(\vec{\Theta})}{\partial \Theta_1} & \frac{\partial e_1(\vec{\Theta})}{\partial \Theta_2} & \cdots & \frac{\partial e_1(\vec{\Theta})}{\partial \Theta_m} \\ \frac{\partial e_2(\vec{\Theta})}{\partial \Theta_1} & \frac{\partial e_2(\vec{\Theta})}{\partial \Theta_2} & \cdots & \frac{\partial e_2(\vec{\Theta})}{\partial \Theta_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_S(\vec{\Theta})}{\partial \Theta_1} & \frac{\partial e_S(\vec{\Theta})}{\partial \Theta_2} & \cdots & \frac{\partial e_S(\vec{\Theta})}{\partial \Theta_m} \end{bmatrix} \tag{4.9}$$

is the Jacobian matrix containing first derivatives of MLP errors with respect to the $\vec{\Theta}$ parameters. When μ is large the method behaves like a steepest descent method. When μ is small the method is equivalent to a Gauss-Newton method. μ is updated at each iteration depending on how E changes.

While denoising by MLP has been shown to be possible given enough layers, nodes and training samples, several challenges exist in practice to make it computationally feasible. Burger and co-workers [72] studied the use of MLPs as a pure learning denoising approach by training a relatively large MLP to denoise images corrupted by AWGN. Their MLP did not use multiple copies. It was trained to

denoise patches of size 17×17 stitched together after denoising to reconstruct the final image. Their MLP contained four hidden layers with 2,047 nodes each, and was trained with 362 million samples derived from a database of 1,500 images and requires approximately one month of training time on a GPU for a specific noise level. While their MLP could compete with BM3D, it remains impractical in real applications due to its exceedingly high computational cost and lengthy execution times.

Here we use an entirely different approach where the system's architecture (Fig. 4.1) uses multiple stages of MLP. We divide our system into two phases, a training phase and a feed forward phase. The training phase is where the MLPs learn to build an optimized model for the application at hand. This is where the nonlinear filter is designed. This phase can take anywhere from 15 minutes to several hours on a modern laptop computer, depending on the noise level. Once the training phase is complete, the MLPs operate in feed forward mode. This is the phase where the nonlinear filter is applied to the image data. The feed forward phase is much faster than the training phase. The time required to denoise an image is on the order of several seconds to a few minutes, depending on the size of the image.

4.2.2 Training Phase

4.2.2.1 Multiple stages

The first step to reducing computational cost is using several small MLPs trained in multiple stages instead of one large MLP. While the MLP architecture can be optimized to further enhance performance, architecture optimization is a topic on its own and will not be considered here. The MLPs used in our tests feature 6 hidden layers each with 10 nodes per layer. Performance is better with more training samples, but larger datasets require more nodes, increasing MLP size and

computation time. The lower the noise level, the less training is required. Thus, we do training in a first stage of small MLPs with a relatively small dataset to minimize noise to a high degree, albeit not state-of-the-art level. When training is done, first stage MLPs operate in feed forward mode and are used to denoise original training images. The end result is a set of estimates with arbitrary residual errors, yielding arbitrary noise distributions with much smaller standard deviation. Seven first stage MLPs are shown in the diagram and yield seven estimates for each training image with much lower noise than original copies. Estimates are used to generate a new dataset for training the second denoising stage MLPs. Multiple MLP stages can be added in a similar manner and trained hierarchically. A third denoising stage uses dataset generated from estimates of the second denoising stage MLPs and so on. In our tests, four MLP stages were used. For clarity, only two MLP denoising stages are shown in (Fig. 4.1). The last stage in the diagram is a little different and will be discussed separately.

This multistage denoising approach is extremely powerful and is a major driving force behind the performance of our algorithm. Multistage denoising is, unfortunately, not possible with other denoising methods. This is because other methods make assumptions about the noise statistics whereas the MLP approach is noise independent. For example, an algorithm that is designed for use with Rician noise cannot be used for multistage denoising because its output does not necessarily have a Rician noise distribution.

4.2.2.2 Multiple Copies

Before patches from r noisy copies are introduced to the input layers of the first denoising stage MLPs, they are first grouped into r combinations of $(r-1)$ copies. This produces r distinct realizations of an image random field, but with reduced noise levels. This is denoted by the $\binom{7}{6}$ block in the diagram, where 7 combinations of 6 copies are grouped and added together. This reduces noise levels at the input

to enable shorter training times. This operation is made possible because the MLP denoising process is noise independent. This technique is also not applicable in general to other methods where assumptions are made about noise distribution. For example, adding images with Rician noise produces an image with non-Rician noise, violating the basic assumption of the algorithm.

4.2.2.3 Patch size

Using $d = 8$ for (4.3) produces patches of size 17×17. When 7 copies are used this yields an input vector of length 2023. Multistage training allows using smaller patches per stage while still allowing the system to use information from a large patch size. This concept is illustrated in Fig. 4.3. For $d = 2$ for first stage MLPs, each output represents a center pixel from patches of size 5×5 from original noisy copies. Using $d = 2$ for second stage, each output represents a center pixel from the 5×5 patches from the first denoising stage MLPs estimates. This collectively gives an effective patch size of 9×9 from original noisy copies to be used as inputs for the second denoising stage MLPs. In general, the effective d value for a stage is the sum of individual d values from previous stages. This technique reduces processing and memory requirements, making our method applicable to devices with low computational resources. Using a smaller d value in a consecutive stage yields a reduction in dimensionality, giving the option of optimizing for speed or memory. Larger d values are typically assigned for the first training stage where smaller data sets can be used. Decreasing d in subsequent stages allows for larger data sets.

4.2.2.4 Feature extraction

In our discussion so far, each stage includes multiple MLPs. Ultimately, we need one final value for each pixel. One way to do this is by averaging values of all

estimates from the final stage MLPs. Alternatively, an additional stage can be added with one MLP and $d = 0$ to get a final value. However, instead of using raw MLP outputs to train this additional (now final) stage, we use feature extraction to enhance generalizing ability. This is indicated by the feature extraction block where the mean, standard deviation, minimum and maximum of outputs from stage 2 are used to train the MLPs in stage 3 to get a final result.

4.2.3 Feed forward phase

After completing MLP training for all stages, the system is used in feed forward mode where image denoising is performed. In our case, we require 7 noisy copies of an image to produce a clean estimate. Patches are extracted for each pixel from its surrounding neighbors for all 7 copies, producing 7 patches of size $N_p = (2d+1)^2$. Denoising is performed hierarchically. While MLPs in each stage are independent and can be processed in parallel, the performance at each stage depends on results from preceding stage. Pixels estimates from the first stage are reorganized into their corresponding positions in the image. The same data acquisition process is performed on the first stage image estimates using d values of the second stage. The final stage produces one final estimate for each pixel. The estimates are regrouped to produce a final estimate of the denoised image. The total time for denoising an image depends on its size. The computer used in our work is a laptop equipped with a 4-core Intel® Core™ i7-3610QM CPU @ 2.30 GHz per core. The average time required for denoising of an 128×128 image was approximately 15 s. Time grows linearly with the number of pixels, i.e. a 256×256 image requires approximately 15 s $\times$ 4 = 60 s.

4.3 Results

Results obtained for various noise levels and distributions indicate that our method can outperform the current state-of-the-art denoising methods provided the algorithm is given enough training time and samples, under the condition of reasonable training time. The longest training time we encountered was less than 10 hours. Because our method allows optimizing performance for specific applications while maintaining good generalizing ability, datasets need only be on the order of hundreds of thousands training samples. We first perform a time comparison to find out how much training time it takes our system to achieve comparable results with the benchmark MLP from [72] and BM3D. We then apply our method to denoising MRI images with Rician noise and compare our results under different noise levels with BM4D and TV. We also apply our method to denoising images under arbitrary noise including multiplicative and additive components for different noise levels. Finally, we apply our method to real MRI data and compare our results to an array of MRI denoising algorithms. In order to keep comparisons consistent, we use implementation demos provided by each group for their algorithm on their website. Links for each method can be found in the reference section. Note that as our approache is learning-based, noise estimation is not explicitly required by our method. In practical applications, learning-based denoising approaches implicitly learn the noise distribution for the specific application from training data and do not require a separate noise estimation step. When a learning-based approach is used for general denoising, a database for different noise levels is created and noise estimation methods are considered at that point, however this is outside the scope of this paper.

4.3.1 Time comparison with benchmark MLP and BM3D

A demo for the BM3D method is provided by [62] which can be downloaded at the URL [73]. The demo provided by [72] contains weights for an MLP that was trained for 1 month on a GPU system to denoise images with pixel values in the range $[0, 255]$ contaminated by AWGN of standard deviation 25 (in units of the pixel range). The demo for the MLP method can be downloaded at the URL [74]. We applied our MC-MLP method to denoise an MRI image after adding AWGN with equivalent standard deviation to the one provided by the MLP method of [72] and measured the time it takes to train our system to achieve comparable results. Given that our method uses 7 copies, we use a standard deviation of $25\sqrt{7}$ for the AWGN introduced to each copy as shown in (4.10),

$$X_t(\omega) = Y_t + 25\sqrt{7}\Gamma_t(\omega), \tag{4.10}$$

where the probability measure P is defined as follows: Γ_t is a zero-mean Gaussian random field which is spatially uncorrelated in the sense that Γ_t is independent of Γ_u whenever $t \neq u$. Thus, the spatial correlations in X_t, as seen by the denoising algorithm, are due to the signal Y_t only.

We generate 7 copies $X_t(\omega_1)$, $X_t(\omega_2)$, ..., $X_t(\omega_7)$ with a standard deviation of $25\sqrt{7}$ and use them as inputs to the MC-MLP algorithm. The individual copies are averaged to generate one copy with a standard deviation of 25 to be denoised by the other algorithms. Our method outperformed the MLP from [72] after only 1 hour of training on a standard laptop computer, as compared to 1 month of training in their case. Our method also outperformed BM3D after only 10 hours of training. The results are shown in Fig. 4.4.

4.3.2 Rician denoising comparison with BM4D and TV

BM3D is adapted to Rician noise by using a transform to map it to AWGN and then perform AWGN denoising. The Rician noise implementation was applied to volumetric MRI data. This extension to volumetric data is called BM4D. Instead of grouping similar 2D patches together in 3D blocks, BM4D groups similar 3D patches in 4D blocks. A demo with a dataset of volumetric MRI brain image is provided at the URL [75]. A minimum of 9 slices is required for the demo to work. We apply our method to the same problem and compare its performance on the same dataset used by the BM4D demo for different noise levels by changing the value of σ in the Rician noise distribution added to images. Images with Rician noise are generally defined as shown in (4.11),

$$X_t(\omega) = \sqrt{(Y_{tR} + \sigma\Gamma_t^{(1)}(\omega))^2 + (Y_{tI} + \sigma\Gamma_t^{(2)}(\omega))^2}. \tag{4.11}$$

where Y_{tR} and Y_{tI} are the real and imaginary signal components respectively. P is defined as follows: $\Gamma_t^{(1)}$ and $\Gamma_t^{(2)}$ are zero-mean Gaussian random fields which are statistically independent of each other and spatially uncorrelated in the sense that $\Gamma_t^{(i)}$, $i = 1, 2$ is independent of $\Gamma_u^{(i)}$ whenever $t \neq u$. Thus, the spatial correlations in X_t are due to the signal Y_t only.

We generated 7 copies for each noise level to be used by our method. Since BM4D requires 9 slices, we generate 7 noisy copies for 9 adjacent slices to be denoised by BM4D. Only the last slice is denoised by our method and compared to BM4D. This provides the BM4D implementation 9 times more data than our method. To keep the comparison as fair as possible, we resized slices used for BM4D by a factor of 3×3, yielding the same amount of data for each method. Unlike the case with AWGN, taking the mean of copies with Rician noise distribution does not produce a Rician noise distribution. Instead, we performed BM4D separately on each copy and took the average of all the outcomes.

For further validation, we also compared our method to a TV implementation for Rician noise provided by Center of Domain Specific Computing (CSDC) at UCLA [76]. (Fig. 4.5) and (Fig. 4.6) show results for low and extreme noise levels, respectively. Results from all noise levels are summarized in (Fig. 4.7). According to all three metrics used here, peak signal to noise ratio (PSNR), feature similarity (FSIM) and mean structural similarity (MSSIM), our method displayed superior performance across all noise levels. But aside from these metrics, the best assessment is the ability to identify specific anatomical features. It is clear by comparing (say) (Fig. 4.6) j to (Fig. 4.6) h or (Fig. 4.6) i that our method preserves the anatomical features whereas competing algorithms merely smooth and blur out important details. In particular, the gray to white matter contrast remains clearly defined with our denoising method. From a clinical point of view, this property is most important. Our method is able to depict much of the layering cortex whereas information is lost in the images from competing algorithms (Fig. 4.6) h and i compared to the original image (Fig. 4.6) g, for example, in the dark signal regions in the inner surface of the sulcus. These results demonstrate the ability of our method to capture finer features than conventional denoising algorithms are able to.

4.3.3 Noise with a multiplicative component

In this test, we demonstrate the ability of our method to perform in the presence of multiplicative noise. We applied our method to denoising MRI images of a cherry tomato contaminated by the noise distribution of (4.12),

$$X_t(\omega) = \sigma_1 \Gamma_t^{(1)}(\omega)(Y_t + \sigma_2 \Gamma_t^{(2)}(\omega)), \tag{4.12}$$

where P is defined as follows: $\Gamma_t^{(1)}$ and $\Gamma_t^{(2)}$ are zero-mean Gaussian random fields which are statistically independent of each other and spatially uncorrelated in the

sense that $\Gamma_t^{(i)}$, $i = 1, 2$ is independent of $\Gamma_u^{(i)}$ whenever $t \neq u$. Thus, the spatial correlations in X_t are due to the signal Y_t only.

We tested our method with different noise levels by varying the value of σ_2. Since we are not aware of other methods designed for this type of noise, we compared the results of our system to the mean value of the noisy copies. The latter approach is the most commonly used method for reducing noise in experiments. The results are shown in (Fig. 4.8). By comparing (Fig. 4.8) c to (Fig. 4.8) b [relative to the reference image, (Fig. 4.8) g, it is clear that the method does a very good job at removing the noise even under conditions of extreme noise.

4.3.4 Application to MRI images with real noise

So far we have evaluated the performance of the algorithm using MRI images that were contaminated by noise, as is common practice when evaluating novel denoising algorithms. This enabled us to select the noise distribution and type. In this section, we collected noisy MRI data sets and evaluated the denoising performance of our algorithm against a wide variety of high-performance denoising algorithms. The nature of the noise was not known *a priori*. Noisy MRI data was obtained using a spin-echo imaging pulse sequence with different repetition times (TR) to alter the SNR. All measurements were performed on a 9.4 T vertical bore Varian VNMRS micro-imaging system, using a 40 mm-i.d. imaging probe. The imaging parameters were: TR = 100 ms for noise level 1, 50 ms for noise level 2, TR = 30 ms for noise level 3 (Fig. 4.9), and 2000 ms for the high SNR original image (Fig. 4.10). TE = 19 ms for all noise levels (Fig. 4.9), and TE = 18 ms for original image (Fig. 4.10). Matrix size = 512 × 512, and field of view (FOV) = 45 mm × 30 mm. We compared our method to BM4D, TV,adaptive non-local means filter [77] (AONLM), adaptive multiresolution non-local means filter [78] (ONLM), optimized blockwise Rician non local means filter [79, 80, 81] (ORNLM), oracle-based 3D discrete cosine transform filter [82] (ODCT), and

Table 4.1: PSNR, FSIM, and MSSIM values corresponding to each image in (Fig. 4.9).

	1Copy	Mean	MC-MLP	BM4D	TV	ORNLM	AONLM	ONLM	ODCT	PRINLM
Noise Level 1 (PSNR, FSIM, MSSIM)	17.95, 0.72, 0.09	20.61, 0.87, 0.33	26.13, 0.93, 0.81	22.56, 0.92, 0.79	25.82, 0.92, 0.78	25.78, 0.90, 0.70	23.07, 0.90, 0.73	25.71, 0.90, 0.71	23.13, 0.89, 0.71	24.01, 0.92, 0.78
Noise Level 2 (PSNR, FSIM, MSSIM)	14.02, 0.57, 0.04	16.46, 0.72, 0.09	21.71, 0.85, 0.70	19.50, 0.85, 0.58	19.48, 0.79, 0.42	20.77, 0.76, 0.41	19.25, 0.76, 0.37	21.15, 0.76, 0.44	19.17, 0.78, 0.47	20.30, 0.85, 0.52
Noise Level 3 (PSNR, FSIM, MSSIM)	12.95, 0.53, 0.03	13.85, 0.58, 0.04	19.39, 0.78, 0.60	17.81, 0.76, 0.33	16.73, 0.60, 0.12	18.33, 0.76, 0.31	16.54, 0.54, 0.11	18.28, 0.54, 0.18	16.54, 0.63, 0.26	17.06, 0.78, 0.33

prefiltered rotationally invariant nonlocal means filter [82] (PRINLM). Our MC-MLP algorithm outperformed all other methods in terms of all the metrics used (PSNR, FSIM, and MSSIM) for quantification. The results are shown in (Fig. 4.9) and in Table 4.1.

4.4 Conclusion

We presented a feature-preserving image denoising algorithm in which a nonlinear filter is designed using a hierarchical multistage system of MLPs. From the point of view of conventional metrics (PSNR, FSIM, MSIM), the algorithm outperforms state-of-the art methods from low to high noise levels, and can handle both additive and muliplicative noises, including Gaussian and signal-dependent Rician nosies. For moderate to low noise levels, competing algorithms are limited to special cases where the known noise distribution meets narrow criteria. Our approach is general and is applicable to situations with arbitrary noise distributions and can even operate under extreme noise levels. It can also be used in situations where the noise distribution is not known, where it can still learn to model it from experimental data.

The filtering is computationally efficient and shows that multiple copies of

the same image enable more effective noise removal with better preservation of anatomical features. Competing denoising algorithms tend to smooth images to the point where important anatomical details are lost. There are several possible scenarios in which our method could be applied. One such application is MRI, where low SNR or low contrast-to-noise ratio situations frequently arise. Namely, with low-field MRI, MRI of lower sensitivity nuclei (such as ^{23}Na or ^{31}P), diffusion tensor imaging in the presence of strong diffusion gradients, MR spectroscopy of metabolites at low concentrations or functional MRI. Other scenarios could include x-ray, position emission tomography and ultrasound imaging. Our method could also be applied to video data using neighboring frames provided that the motion is not too large or that motion tracking is used. We have shown that as little as 7 copies are required for good performance, making the method practical in terms of data acquisition times, as low-SNR situations generally require far more than 7 signal averages.

Figures

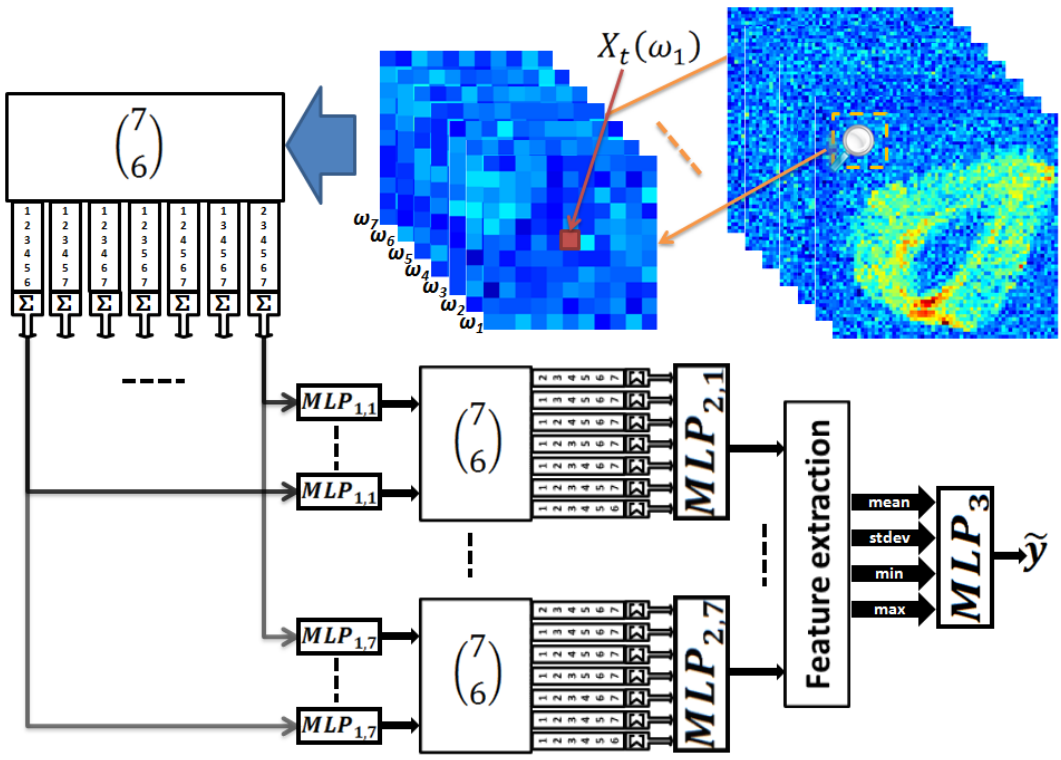

Figure 4.1: System design for the MC-MLP denoising algorithm as implemented by multiple stages.

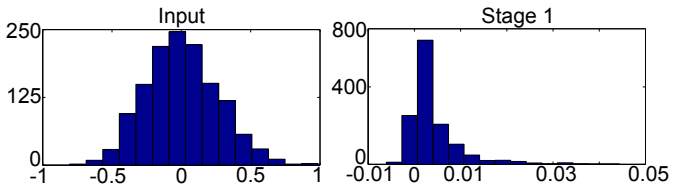

Figure 4.2: Example of the evolution of the noise distribution after a denoising stage.

Here (left) the Gaussian input noise distribution is very different from the (right) noise distribution after the first denoising stage.

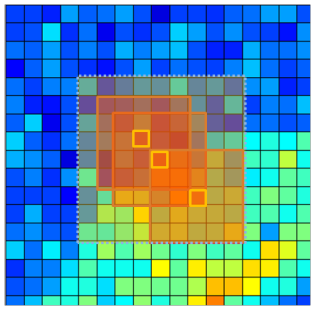

Figure 4.3: Multiple stage training allows for using smaller patches at each stage. Here, a 17×17 patch is subdivided into smaller patches which are sent to the multiple stages of denoising for processing.

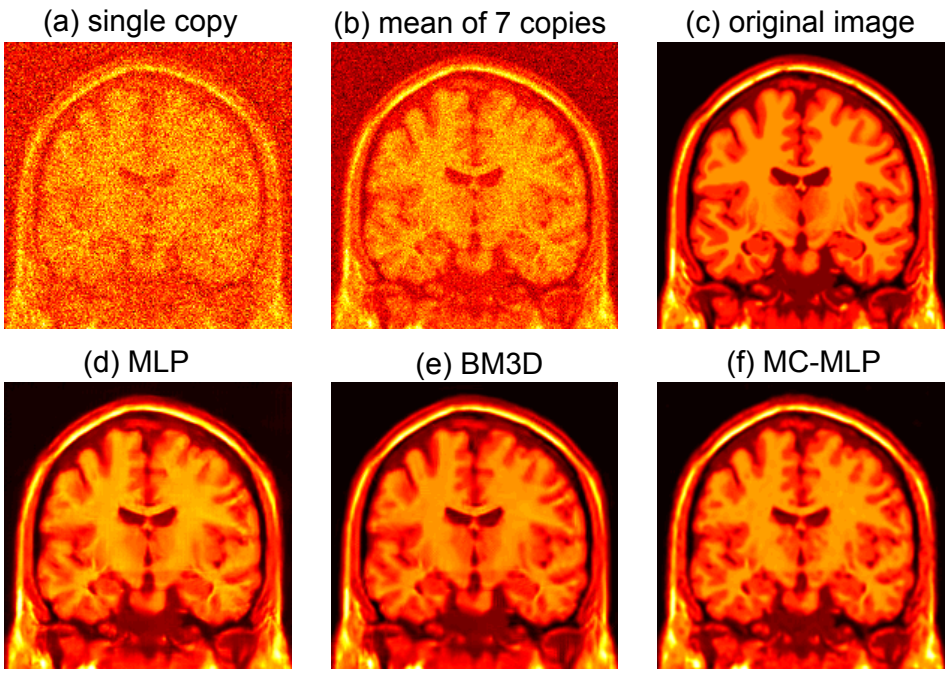

Figure 4.4: Comparison of MC-MLP with MLP and BM3D after 10 hours of training at noise level $\sigma = 25$.

The goal here is to demonstrate computational efficiency of MC-MLP as compared to MLP, by measuring the training time required to obtain similar performance as BM3D. (a) Noisy image (1 copy). PSNR: 11.723 dB. FSIM: 0.450. MSSIM 0.135. (b) The average of 7 copies. PSNR 20.242 dB. FSIM: 0.687. MSSIM 0.425. (c) Original (ideal) image with high SNR used as the gold standard for comparison. (d) MLP method applied to average of the 7 copies. PSNR 30.615 dB. FSIM: 0.922. MSSIM 0.869. (e) BM3D method applied to average of the 7 copies. PSNR 31.228 dB. FSIM:0.933. MSSIM 0.908. (f) Denoising using the MC-MLP method with 7 copies. PSNR 31.242 dB. FSIM:0.936. MSSIM 0.919.

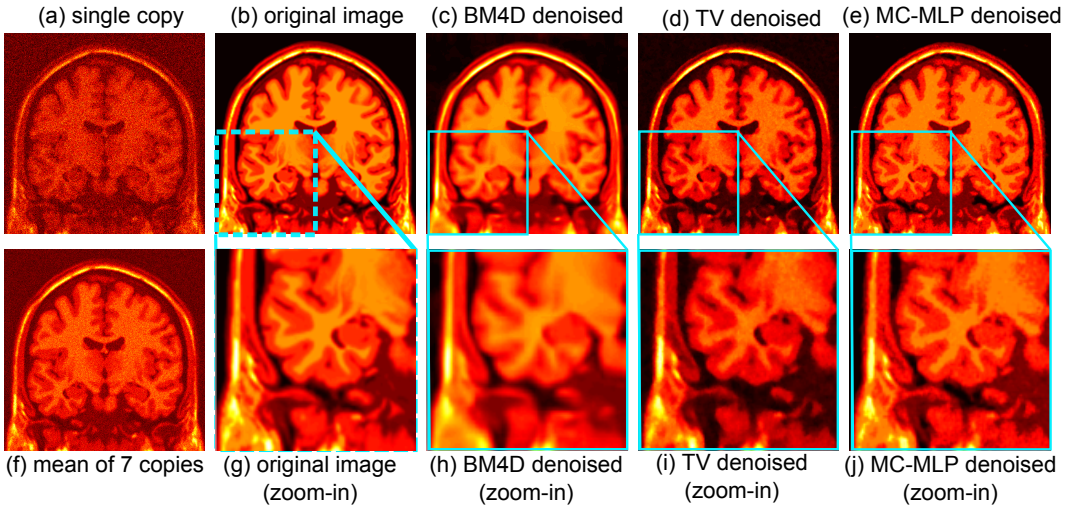

Figure 4.5: Comparison of MC-MLP with BM4D and TV methods ($\sigma = 20$). (a) Noisy image (1 copy). PSNR 13.378 dB. FSIM 0.658. MSSIM 0.073. (b) Original (ideal) image with high SNR used as the gold standard for comparison. (c) BM4D method applied to each of the 7 copies. The average of 7 denoised copies is shown. PSNR 28.789 dB. FSIM 0.894. MSSIM 0.804. (d) TV method applied to each of the 7 copies. The average of 7 denoised copies is shown. PSNR 21.793 dB. FSIM 0.896. MSSIM 0.703. (e) Denoising using the MC-MLP method with 7 copies. PSNR 33.314 dB. FSIM 0.951. MSSIM 0.907. (f) Signal-averaged image corresponding to the mean of 7 copies. PSNR 16.718 dB. FSIM 0.832. MSSIM 0.273. (g) Close-up on the lower left quadrant of (b) (h) Close-up on the lower left quadrant of (c). (i) Close-up on the lower left quadrant of (d). (j) Close-up on the lower left quadrant of (e).

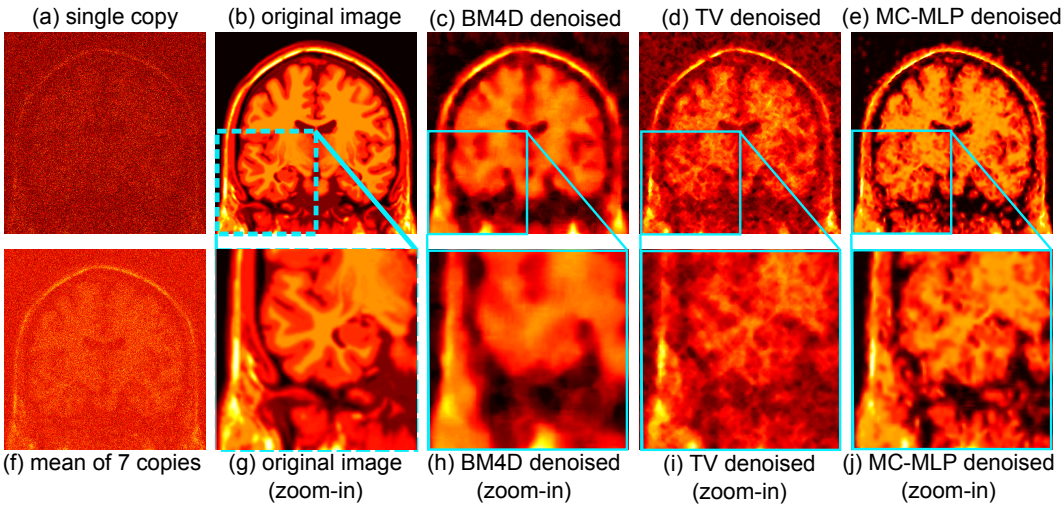

Figure 4.6: Comparison of MC-MLP with BM4D and TV methods ($\sigma = 70$). (a) Noisy image (1 copy). PSNR 1.863 dB. FSIM 0.373. MSSIM 0.005. (b) Original (ideal) image with high SNR used as the gold standard for comparison. (c) BM4D method applied to each of the 7 copies. The average of 7 denoised copies is shown. PSNR 20.658 dB. FSIM 0.743. MSSIM 0.594. (d) TV method applied to each of the 7 copies. The average of 7 denoised copies is shown. PSNR 18.647 dB. FSIM 0.708. MSSIM 0.486. (e) Denoising using the MC-MLP method with 7 copies. PSNR 22.678 dB. FSIM 0.806. MSSIM 0.652. (f) Signal-averaged image corresponding to the mean of 7 copies. PSNR 3.523 dB. FSIM 0.560. MSSIM 0.030. (g) Close-up on the lower left quadrant of (b) (h) Close-up on the lower left quadrant of (c). (i) Close-up on the lower left quadrant of (d). (j) Close-up on the lower left quadrant of (e).

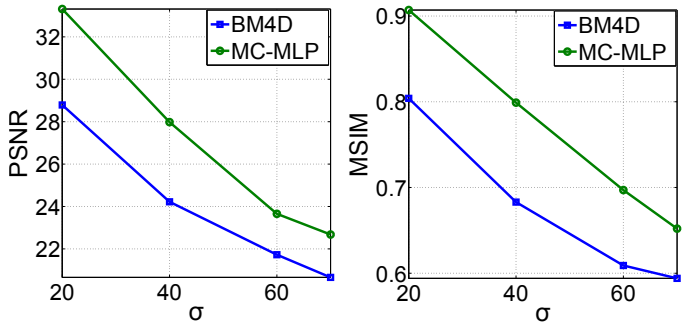

Figure 4.7: PSNR and MSIM comparison of MC-MLP with BM4D method for different noise levels ($\sigma = 20, 40, 60$ and 70).

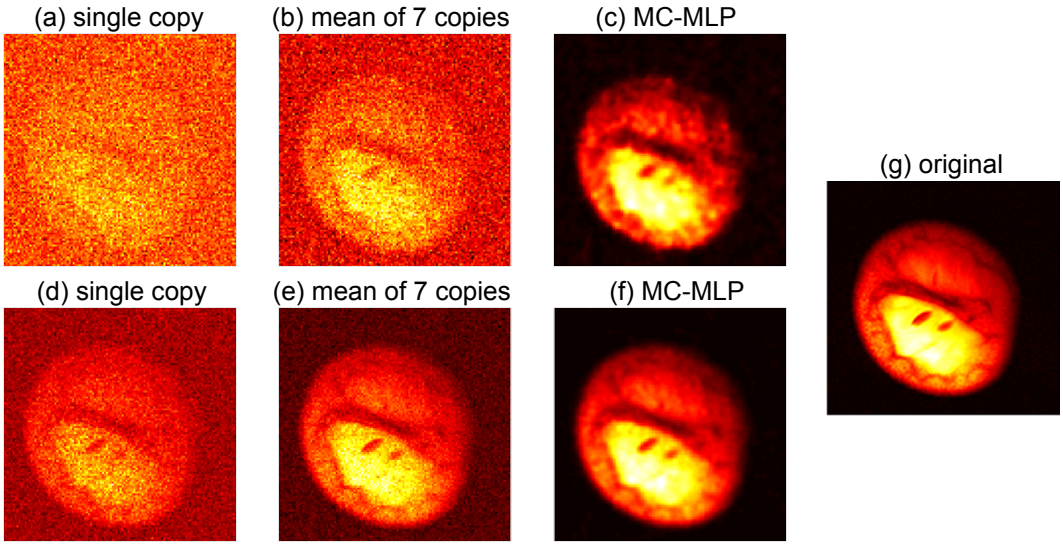

Figure 4.8: Comparison of MC-MLP with mean for arbitrary noise including multiplicative component.

(a) Noisy image (1 copy) $\sigma_2 = 50$. (b) The average of 7 copies ($\sigma_2 = 50$). PSNR 13.82 dB. FSIM 0.37. (c) Denoising using the MC-MLP method with 7 copies $\sigma_2 = 50$. PSNR 26.88 dB. FSIM 0.87. (d) Noisy image (1 copy) $\sigma_2 = 10$. (e) The average of 7 copies ($\sigma_2 = 10$). PSNR 26.52 dB. FSIM 0.79. (f) Denoising using the MC-MLP method with 7 copies $\sigma_2 = 10$. PSNR 33.77 dB. FSIM 0.95. (g) Original (ideal) image with high SNR used as the gold standard for comparison.

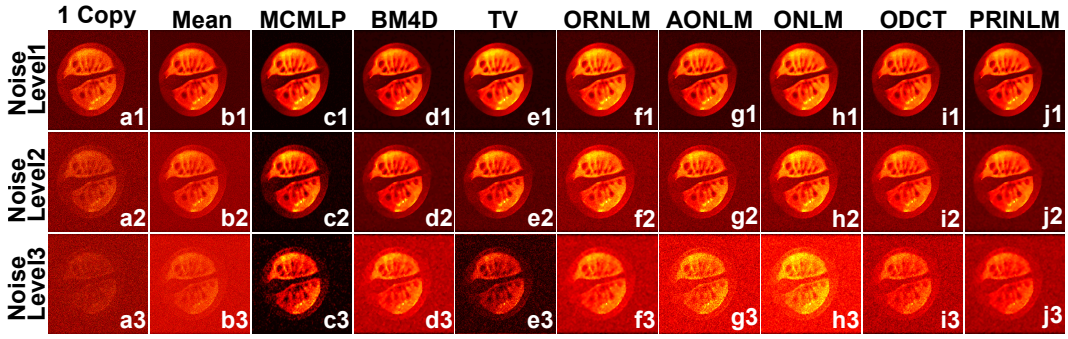

Figure 4.9: Comparison of MC-MLP algorithm with several MRI denoising methods

Methods compared: (BM4D, TV, ORNLM, AONLM, ONLM, ODCT, PRINLM) applied to a T_1-weighted image of a cherry tomato acquired on a Varian 9.4 T microimaging system using a spin-echo imaging sequence. Different noise levels (noise level 1, 2 and 3) were created by adjusting the TR (repetition time) value in the pulse sequence. The performance of each denoising algorithm is evaluated using the performance metrics of PSNR, FSIM, and MSSIM and the values are given in Table 4.1. The MC-MLP algorithm outperformed other methods for all noise levels, according to all performance metrics. A high SNR image of the cherry tomato is shown in (Fig. 4.10) for comparison. The salient feature of our algorithm is that not only the SNR of the denoised image is higher, even under conditions of extreme noise levels, but the features of the image are preserved as opposed to blurred out, as is the case for conventional algorithms.

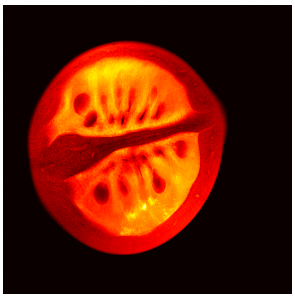

Figure 4.10: High resolution MRI image of the cherry tomato.
Used for evaluating denoising performance of the results of (Fig. 4.9).

CHAPTER 5

Engineering Flow Fields in Bioreactor Scaffolds

5.1 Introduction

Regenerative medicine aims to produce fully functional tissue for implantation, often in an *ex vivo* setting, for purposes of replacing or regenerating organs. A common TE protocol is to seed cells into pre-determined tissue scaffolds, adjust the physical conditions such as flow rate, nutrients and growth factors, and then measure cell response. In recent years, the role of mechanical forces, such as shear stress, has been recognized in the context of regulating proliferation, migration and morphogenesis [83, 84, 85, 86, 87, 88]. Consequently, considerable efforts have been devoted to tailoring shear stress distributions arising from flow fields applied to bioreactors as a means to study the cellular response due to spatial gradients [18, 19, 20, 21, 22]. The state of the art approach involves creating materials with local porosity gradients [23, 24, 25, 26, 19, 20, 27, 28]. While this approach is suitable for setting initial conditions in a scaffold at the expense of high material complexity, it fails to accommodate the dynamical aspect where different stages of development lead to changes in the material properties.

Herein, we present a general methodology to control mechanical forces inside bioreactors both in space and in time. Advantages include the creation of complex flow fields and the possibility of real-time control in response to cell growth without altering the composition or structure of the scaffold material itself. Arbitrary flow fields are generated by individually adjusting inlet pressures to a multiple-inlet

bioreactor, generating various mechanical force distributions in a single scaffold. Real-time control can be achieved by employing this flow control strategy in a feedback control loop guided by non-invasive imaging. The spatial distribution of mechanical forces can be altered on demand by dialing-in the appropriate inlet pressure at various inlets, thereby offering flexible means of dynamic control that can adapt to structural changes.

Our approach operates in two phases. The first phase is virtual: a multilayer perceptron (MLP) is trained to learn the nonlinear relationship between flows applied at the bioreactor's inlets and the shear rate maps resulting in the scaffold. The MLP was trained with examples obtained from computational fluid dynamics (CFD) simulations that estimate patterns of flow and shear rate based on known material properties, namely fluid permeability and porosity [44, 20]. A set of inlet flow speeds is found that can be dialed-in to produce a desired shear rate distribution in the scaffold. The second phase is experimental, involving the selection of target shear rate maps, application of the required inlet pressures to generate this map, followed by experimental verification of the flow field through non-invasive measurements of flow velocity. The design and construction of a 10-inlets bioreactor is shown in (Fig. 5.1). The bioreactor is made of non-magnetic materials for compatibility with a nuclear magnetic resonance (NMR) environment where maps of fluid velocity [36] were generated inside the reactor. From the flow maps, we calculated the shear rate distributions using finite-difference approximations [89]. This provides an estimate of interstitial flows in the porous scaffold matrix. NMR is the method of choice to probe material properties of opaque media in a non-invasive manner. Namely, it can be used for investigating the hydrodynamic properties of biomaterials through measurements of shear rate, hydraulic pressure and fluid permeability, as demonstrated in recent studies [49, 89].

5.2 Materials and methods

5.2.1 10-channel bioreactor design.

A bioreactor possessing 10 input channels and one output channel was designed to test our method, as shown in (Fig. 5.1). Inlets are distributed around a chamber that holds a 3 mm-thick scaffold. There are 3 inlets at a 45° angle on each side, 4 inlets at the bottom and 1 output channel at the top. (Fig. 5.1) a and b show the CAD drawings of the bioreactor design. (Fig. 5.1) c and d show the actual bioreactor, and (Fig. 5.1) e shows the fully-assembled bioreactor. The bioreactor and its parts were made of NMR-compatible materials such as PTFE. The bioreactor is designed to fit in a 40 mm-i.d. NMR imaging probe. Inlets are connected to current-controlled mini proportional valves (Kelly Pneumatics, Inc., Costa Mesa, CA). A programmable current regulating circuit was designed to receive commands from a computer program to set the current value at each valve. This enabled computer controlled flow speeds at each inlet. A cellulose scaffold with high porosity ($\sim 90\%$) was used and experiments were performed using water as the input fluid.

5.2.2 Adaptive control algorithm.

Our algorithm consists of a fast search method designed to guide a MLP to perform an initial estimation of required inlet speeds using CFD simulations, by learning the nonlinear relationship between inlet flow speeds and target parameters. The scaffold was treated as a grid with nine equal regions. The algorithm was configured to create shear rate patterns by simultaneously controlling shear rate mean values in these regions. Thus, the problem in our case involves ten inlets and nine parameters. The system design diagram is shown in (Fig. A.1). A description of the algorithm, pseudo code and a table of main parameters are available in (App. A). Parameter values can be adjusted depending on the application

requirements.

5.2.3 CFD shear rate simulation.

Finite element analysis software COMSOL Multiphysics (COMSOL Inc., Burlington, MA) was used to solve Navier-Stokes equations for incompressible Newtonian laminar flow with no slip boundary conditions:

$$\rho(U \cdot \nabla)U = \nabla \cdot [-P + \nu(\nabla U + (\nabla U)^T))] - \rho G, \quad \rho \nabla \cdot U = 0, \quad (U = 0 \text{ at walls}).$$

U denotes the velocity vector field ($\mathrm{m \cdot s^{-1}}$), P is the pressure (Pa), ρ is density ($\mathrm{kg \cdot m^3}$) and ν is the dynamic viscosity (Pa·s). The term $\rho \cdot G$ takes gravity into consideration, where G is the gravitational acceleration ($\mathrm{m \cdot s^{-2}}$). At the inlets, we set $U = U_{in}$ according to the input speed determined by the algorithm. No viscous stress ($P = P_0$) boundary condition was selected at the outlet. The velocity field is obtained by solving the Navier-Stokes equation. Shear rate ς ($\mathrm{s^{-1}}$) is then calculated as a function of the velocity field using finite difference approximations of $\varsigma = (\nabla U + \nabla U^T)/2$. Shear stress τ (Pa) is obtained by multiplying shear rate values with dynamic viscosity, i.e. $\tau = \nu\varsigma$.

5.2.4 NMR shear rate measurement.

The pulse sequence for the NMR imaging experiment is shown in (Fig. A.3). A standard spin-echo imaging experiment with slice-select (S.S.), readout (R.O.), and phase-encode (P.E.) gradients was modified to include phase-contrast velocimetry as in refs. [49, 89]. To generate 2-D images, the sequence is repeated n times while looping of the P.E. gradient. Flow-compensation (F.C.) gradients were added along all directions except the P.E. direction. Bipolar, trapezoidal flow-weighting (F.W.) gradients are added along the x, y, and z directions in order to select the gradient first moment M_1. In the case of no flow, stationary nuclear spins experience the positive and negative lobes of the bipolar F.W. gra-

dient with the same magnitude. For stationary spins, the phase accumulation from the positive lobe is equal and opposite to the phase accumulation from the negative lobe, resulting in zero net phase accumulation. For moving spins, phase cancellation is incomplete [36] and the residual phase accumulation is proportional to velocity. Pulse sequence parameter values are listed in Supplementary Methods.

5.3 Results

A variety of shear rate maps were studied; two examples of spatial patterns are shown in (Fig. 5.2). (Fig. 5.2) a shows a non-trivial pattern whereas (Fig. 5.2) d shows a uniform pattern. CFD simulations of flows within the scaffold of the bioreactor were performed and the results are shown in (Fig. 5.2). The first simulation (Fig. 5.2) a−c was used to test the algorithm's ability to find inlet flow speeds in the case of a complex shear rate distribution. The pattern consists of a non-linear gradient featuring low shear rates in the lower right hand corner juxtaposed against a high shear rate region located in the upper left hand corner (Fig. 5.2) a and b. Along the path connecting these two corners, the shear rate values range from $0\ s^{-1}$ to $6\ s^{-1}$. The convergence plot (Fig. 5.2) 2c demonstrates that the algorithm converged to a reasonable solution in less than 25 iterations, and found an accurate solution in less than 50 iterations.

The second simulation (Fig. 5.2) d−f was designed to test the algorithm's ability to adapt to changes. The algorithm was required to find inlet flow speeds to generate a uniform shear rate distribution (shown in (Fig. 5.2) d). The target shear rate value was originally set to $10\ s^{-1}$. After 100 iterations, the algorithm converged to a solution for the $10\ s^{-1}$ value. At that point, the target shear rate value was changed to $11\ s^{-1}$ and the convergence speed to the new value was assessed. The convergence plot (Fig. 5.2) f shows that the algorithm quickly

adapts to the new value (Fig. 5.2) e, where only a small number of iterations were required to find a new solution. This simulation also illustrates the algorithm's ability to generate a highly uniform shear rate distribution. It is generally difficult to create a uniform shear rate distribution in a bioreactor with only a single inlet; but with 10 inlets excellent results could be obtained. Shear rate values in the scaffold varied between 10 s^{-1} and 12 s^{-1}, when the target value selected was 11 s^{-1}.

Experimental validations were conducted by comparing shear rate distributions from simulations with actual shear rate distributions measured experimentally using the technique of NMR velocimetry [36, 49, 89]. Three different complex shear rate distributions generated in the same scaffold are shown in Fig. 3, along with the corresponding maps of shear rates as measured experimentally. We observed excellent agreement between theory and experiments. This demonstrates that the 10-inlet bioreactor design possesses the ability to dynamically generate versatile and complex mechanical force distributions within a scaffold, where the spatial distributions of shear rate values within the scaffold region can be altered to generate arbitrary patterns by merely changing inlet flow speeds.

5.4 Discussion

Tissue engineering studies of the effects of shear flows have been hampered by the lack of suitable platforms to control flow fields. Previous attempts at controlling mechanical forces were limited to altering the scaffold structure and bioreactor geometry. Most bioreactors used to date have been designed to operate with a single inlet. Parameters such as perfusion rate, flow, and mechanical stress are typically selected by trial-and-error. Even then, due to many sources of variability, a protocol that works to bring a particular construct to a desired stage may likely fail to work for another construct [29]. Thus, the importance of adaptive control

with feedback. Given a feedback mechanism, many possible solutions, e.g., single parameter control, or a proportional-integral-derivative controller, may exist for a single-inlet bioreactor. However, a single inlet does not permit fine tuning of the flow field at the microscale, which is an essential element for the study of cellular responses to flow. Bioreactors with multiple inlets, such as the one presented herein, reduce the need for engineering the scaffold material properties, may alleviate manufacturing complexity while enabling the creation of more accurate and complex flow fields. Spatiotemporal control of mechanical force distributions in scaffolds can be used to dynamically control the bioreactor for applications in TE.

Because the relationship between shear maps and inlet pressure is highly nonlinear, the task of finding a set of inlet pressures to generate a desired flow pattern in a 10-inlets bioreactor is a complex adaptive control problem possessing no known analytical solutions. Thus, the adaptive control algorithm presented herein constitutes a crucial part of our approach. The algorithm, whose implementation is described in the Supplementary Methods section, demonstrates high efficiency, as it converges to a solution with a small number of iterations. Furthermore, it can learn complex input/output relationships between inlet flow speeds and mechanical force distributions, which gives it the ability to quickly adapt to changes.

5.5 Conclusions

The concepts presented here are general and could be applied to controlling any other force field, or the flow of substances (e.g., nutrients, gases). Although not demonstrated here, scaffold structure, bioreactor geometry and inlet positions can easily be included as additional parameters to be optimized along with inlet speeds. The extension to 3-D should be straightforward because it does not require a redesign of the algorithm. An interesting open question would be to verify what

shear rate distributions lead to tissue development in a bioreactor scaffold beyond initial stages of cell growth. A possible extension of this work to provide additional control of the flow patterns would be to add multiple outlets to the bioreactor; outlets add more degrees of freedom to the problem where pressure can be released locally.

Acknowledgements

Funding support from the Arnold and Mabel Beckman Foundation and the National Institutes of Health (NIH) award 5 R01 HL114086-03 is acknowledged.

Figures

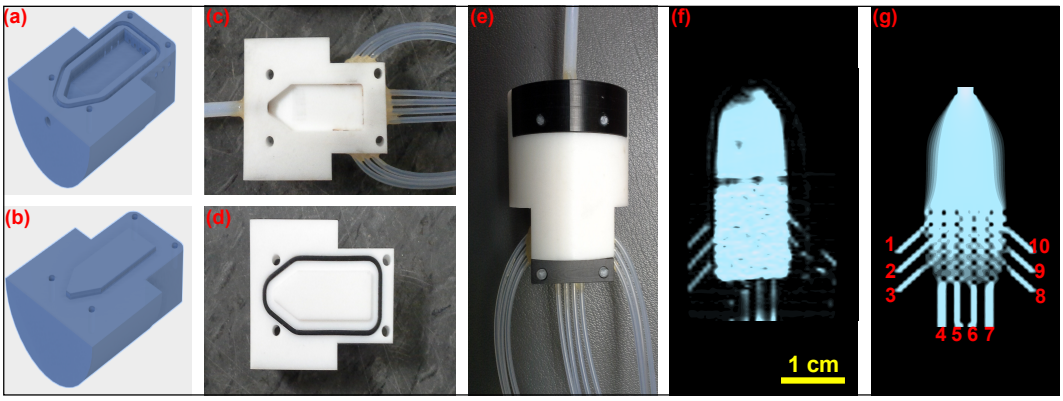

Figure 5.1: Ten-inlet bioreactor design

(a) CAD drawing, bottom part; (b) CAD drawing, top part. Construction: (c) Actual reactor, bottom part; (d) Actual reactor, top part; (e) Fully assembled 10-inlet bioreactor. Fluid region: (f) Proton density NMR image showing the fluid region; (g) Corresponding CFD simulation. The inlets are labeled in (g) for use in (Fig. 5.2) and (Fig. 5.3). In (f) ad (g) the yellow box indicates the region shown in (Fig. 5.3).

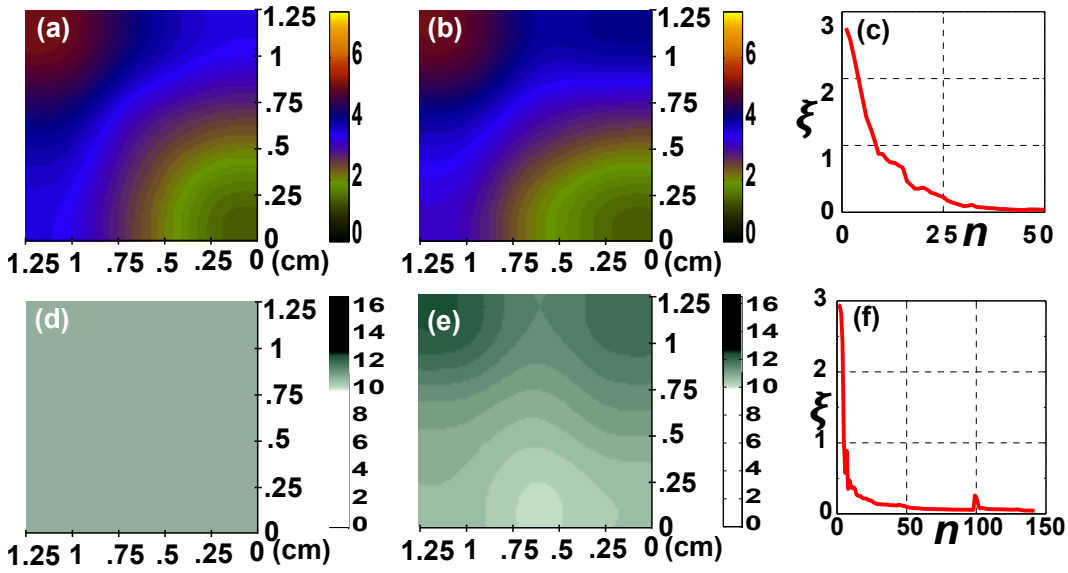

Figure 5.2: Inverse problem solved by multi-layer perceptron.

Results from CFD simulations are shown: (a) Desired target shear pattern; (b) Outcome of applying input flow speeds calculated to generate the pattern shown in (a), rounded to the nearest 0.05 mm/s from inlet #1 to inlet #10 are: [0.50, 0.45, 0.40, 0.15, 0.30, 0.40, 0.80, 1.40, 0.60, 1.20] mm/s, respectively; (c) Convergence plot with the y-axis corresponding to the cost function and x-axis corresponding to iteration number; (d) Uniform shear rate pattern at 11 s^{-1}; (e) Outcome of applying input flow speeds calculated by algorithm to generate the pattern shown in (d), rounded to the nearest 0.05 mm/s from inlet #1 to inlet #10 were: [0.00, 0.05, 1.70, 1.95, 3.90, 2.75, 3.50, 0.05, 1.20, 0.00] mm/s, respectively; (f) Convergence plot with y-axis corresponding to the cost function and x-axis corresponding to the iteration number.

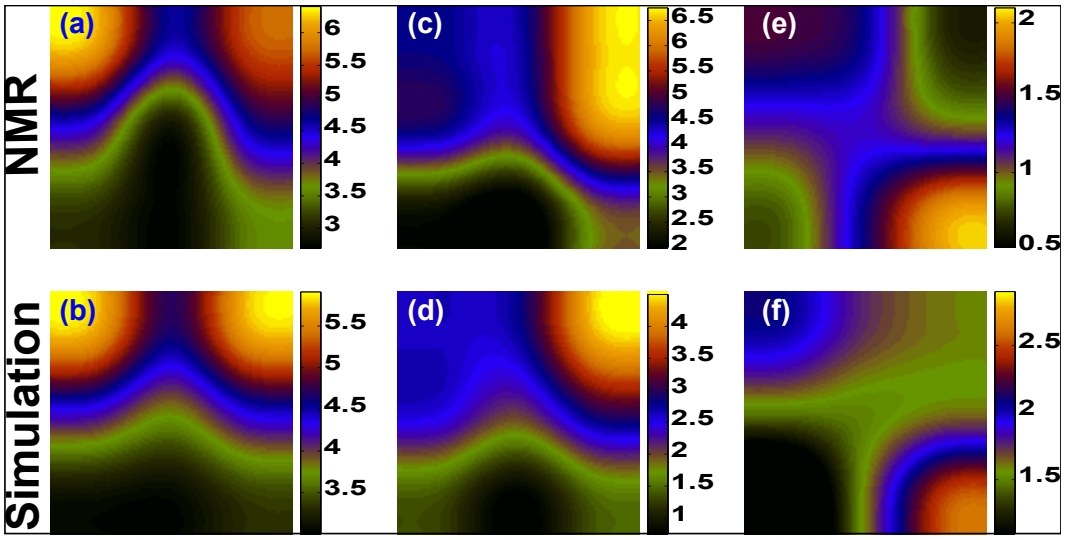

Figure 5.3: Validation of CFD results by NMR flow maps.
Results from NMR velocimetry (a) and CFD simulation (b) for the case where all inlets have same flow speed of 1 mm/s. (c) & (d) are NMR and CFD simulation results, respectively, where inlets [3, 8, 9 & 10] have a flow speed of 1.5 mm/s, and inlets [1, 2, 4, 5, 6 & 7] have zero flow speed. (e) & (f) are NMR and CFD simulation results, respectively, where inlets [4, 5, 8, 9 & 10] have zero flow speed, inlets [6 & 7] have a flow speed of 1 mm/s, and inlets [1, 2 & 3] have a flow speed of 0.5 mm/s.

CHAPTER 6

Future Directions

6.1 Introduction

Contributions presented herein will be integrated with work of other people in the group to ultimately reach the overall goal of growing complex organoids with proper vascularization. Experiments and simulations conducted so far concern flow and tissue mapping as well as flow control. In future work, the real-time information about the state of the tissue growth will be extended by adding cell growth to the process. Optimization experiments can be modified to use information from real-time images of tissue growth. Our group have developed quantitative MRI cell and tissue mapping readouts based on measurements of the T_2 relaxation time. This information can be included to develop real-time metabolic imaging readouts based on ^{13}C-labeled glucose NMR spectroscopy measurements. With tissue and metabolic information, the optimization will maximize metabolic activity throughout the bioreactor by controlling the temporal and spatial distributions of flow and nutrient inputs and adjusting these inputs as needed, based on real-time readouts.

Some projects currently in progress that extend the work presented in this dissertation are briefly discussed here, and some initial results are displayed.

6.2 Noise Estimation

A noise estimation method has been developed based on the MC-MLP algorithm presented in chapter 4. With learning based algorithms, training data can be prepared such that the performance is optimal for a noise distribution at a specific noise level. The high accuracy and robust performance of MC-MLP allows the implementation of an accurate noise estimation method. The method is explained here using Rician noise as an example.

A noise range is specified and divided into intervals of several noise levels. A separate MC-MLP is trained for each noise level, using an artificial noise distribution accurately generated for the specific noise level. In the Racian noise example presented here, a separate MC-MLP is trained with noise generated using (4.11) for σ values between 15% and 40%, more specifically: $\sigma \in \{15, 17, 20, 22, 25, 27, 30, 32, 35, 37, 40\}$.

The key for this method lies in finding a way to measure the performance of a MC-MLP trained on a specific noise level when used to denoise an image. This is achieved using the following steps; 1, The image is denoised. 2, The denoised image is subtracted from the original image to produce the estimated noise removed by the MC-MLP. 3, the distribution of the estimated noise is compared to a Rician noise distribution generated using a σ value similar to the one used to train the MC-MLP. 4, A distance measure is produced by calculating the absolute value of the difference between the standard deviations of the estimated and generated noise distributions. The MC-MLP that yields the smallest distance corresponds to the noise level closest to the noise in the image.

In practice, only a small section of the image needs to be evaluated to speed up the process, it is unnecessary to denoise the entire image. To further speed up processing time we do not measure the distance for each noise level, binary search is used instead. (Fig. 6.1) demonstrates the steps taken by the method to

estimate the noise of an image contaminated with Rician noise with $\sigma = 20$.

6.3 1D denoising

Suppose we have an NMR Free Induction Decay signal (FID) signal f with additive ξ and multiplicative η noise as in (6.1). ω denotes a specific realization of the random process, and t denotes time. We drop both t and ω from the notation and consider a fixed moment in time t. Solving for f we get (6.2). The goal is to find an estimator $\hat{f}$ for f such that the SNR of the spectrum, defined in (6.3), is maximized.

$$s(t,\omega) = \eta(t,\omega)f(t) + \xi(t,\omega) \tag{6.1}$$

$$f = \frac{s - \xi}{\eta} \tag{6.2}$$

$$SNR = \frac{\sum |FFT(f)^2|}{\sum \left(|FFT(f) - |FFT(\hat{f})| \right)^2} \tag{6.3}$$

A 1D version of the the MC-MLP algorithm is being developed and customized to NMR spectrum denoising. It will be used to speed up metabolic imaging readouts and allow real-time feedback, by greatly reducing the number of copies needed for a clear signal. The additive noise component can be estimated from the tail of the FIDs. The problem lies in estimating the multiplicative noise. The MC-MLP method is a perfect candidate for this problem. A number of FIDs with spectrums of varying peak widths and heights are used to train the MLPs. Additive noise, with a distribution similar to the one estimated from the tail of the measured FIDs, is added to the training FIDs where multiple noisy copies of each signals are generated. The MLPs are trained to estimate the actual values of data points in an FID spectrum given information from multiple copies exposed to additive noise. The measured signal containing multiplicative noise appears to

the MLPs as a distorted variation of the data it was trained to recognize. The robustness of the MC-MLP method allows denoising the multiplicative component without the need of an explicit knowledge of its distribution.

Initial results from simulations and real data are shown in (Fig. 6.2) and (Fig. 6.3) for simulated and experimentally acquired data respectively.

6.4 3D denoising

A 3D version of the MC-MLP algorithm is being developed for denoising volumetric data. This will be used to help extend the methods presented herein to 3D measurements and optimization by speeding up acquisition and allowing real-time readouts. A collaboration with a sodium imaging group has been made to test the algorithm. ^{23}Na is an insensitive nucleus in MRI imaging characterized with poor SNR. The challenge in the case of sodium imaging lies in the difficulty to obtain a clean sodium image for training the algorithm. Thus, a more careful preparation of training data is needed. Such situations in general, i.e. when original clean sample images are not available for training, are approached by using images with low level/general features such as simple curves and edges. This ensures that the algorithm is not optimized for anatomical features not present in the type of images its used for denoising.

Although 3D training has not been implemented yet, initial results using 2D training for denoising volumetric images, one slice at a time, are already encouraging. Some initial results are displayed in (Figs. 6.4, 6.5 and 6.6), where performance is further enhanced using a technique we call multi-directional denoising. A 3D volumetric image can be traversed slice by slice along each of its three axis. In brain imaging notation, this allows viewing slices in the sagittal, axial, and coronal planes. Thus, each voxel in the volume is shared by three planes with the exception of voxels on the edges, these can be dealt with using zero padding.

The idea behind the multi-directional denoising technique is that by denoising the slices along the three axis each voxel is denoised three times, where the mean of the three results is then calculated.

The initial results demonstrate clear superiority of MC-MLP in terms of SNR. However, this is still a work in progress. We have not yet determined the anatomical advantage of our algorithm at this stage. Sodium imaging highlights specialized anatomical features that need to be carefully interpreted by sodium imaging experts. There is still a lot of room for improvement in implementing MC-MLP for 3D denoising.

Figures

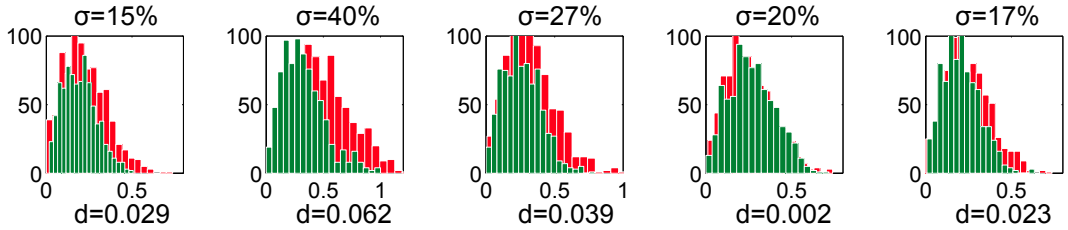

Figure 6.1: Binary search steps.

Noise levels checked by the binary search in the order visited from left to right. A comparison between the noise distribution of noise obtained from a MC-MLP denoising and the expected noise for the corresponding σ is shown for each level. d is a distance measure that indicates how close the two noise distributions are.

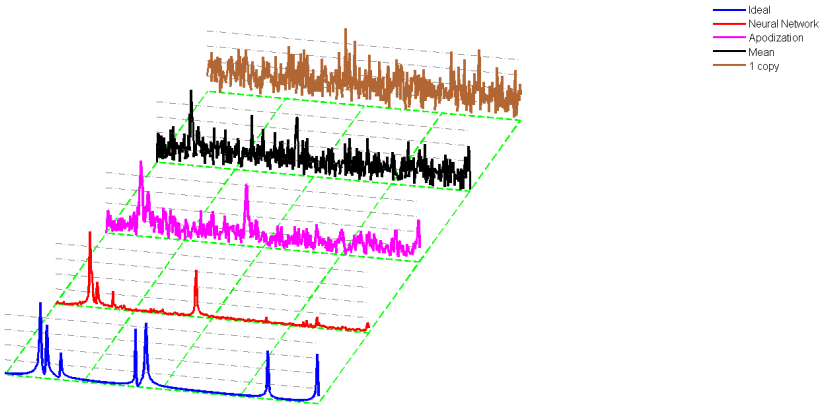

Figure 6.2: Simulation results.

Number of copies: 100. MC-MLP/Apodization SNR: 6.7. MC-MLP/Mean SNR: 16.3.

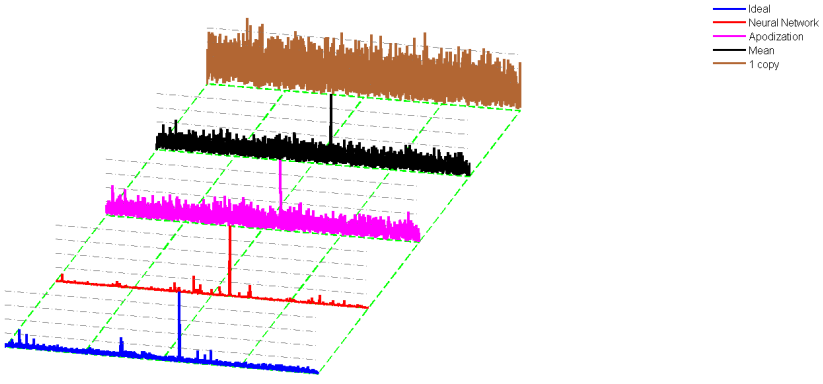

Figure 6.3: Real data results.

Number of copies: 200. MC-MLP/Apodization SNR: 21.8. MC-MLP/Mean SNR: 21.9.

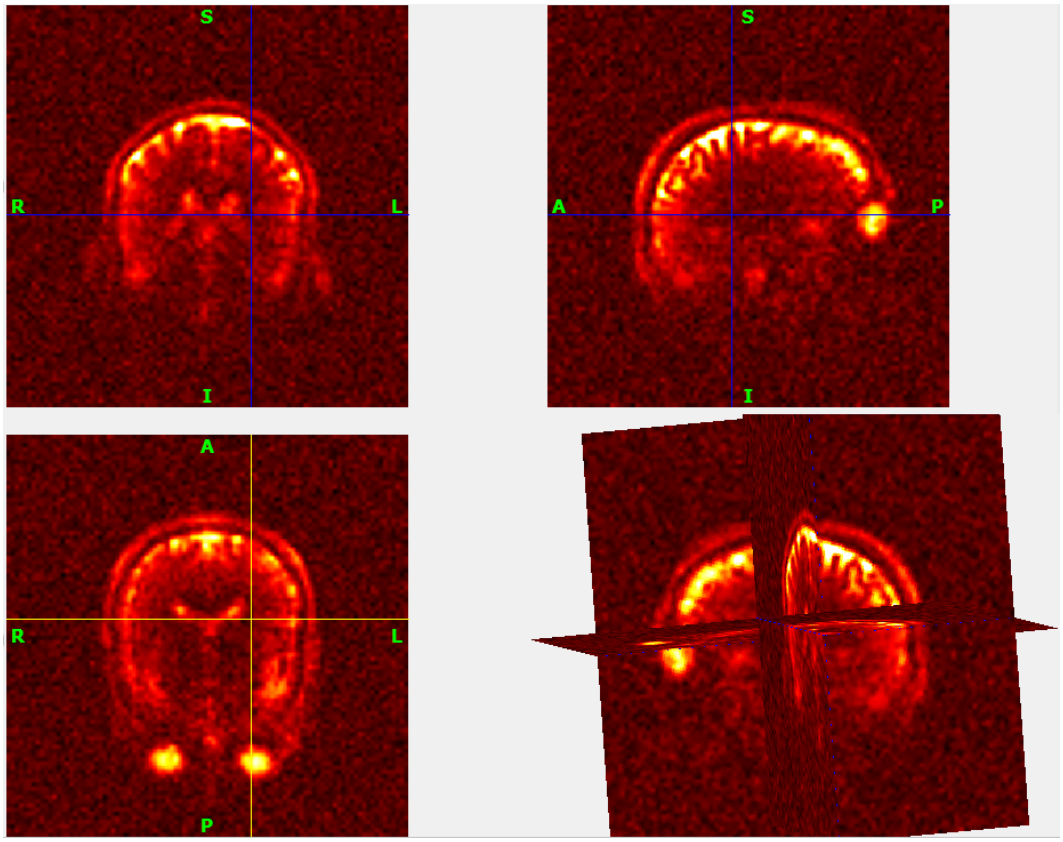

Figure 6.4: Sample mean of 7 sodium MRI recordings.

All sodium images acquired with FLORET sequence at 7T on brain with 15-channel coil, and reconstucted with regridding and sum-of-square of the 15 channels. Real (Nyquist) resolution for data acquisition = 3 mm isotropic = nominal (reconstructed) resolution = 3 mm isotropic (96x96x96 voxels for FOV 288x288x288 mm). Resized to 255x256x256 voxels. Sodium MRI images courtesy of Dr. Guillaume Madelin @ NYU. Plotted using BrainSuite software [90]

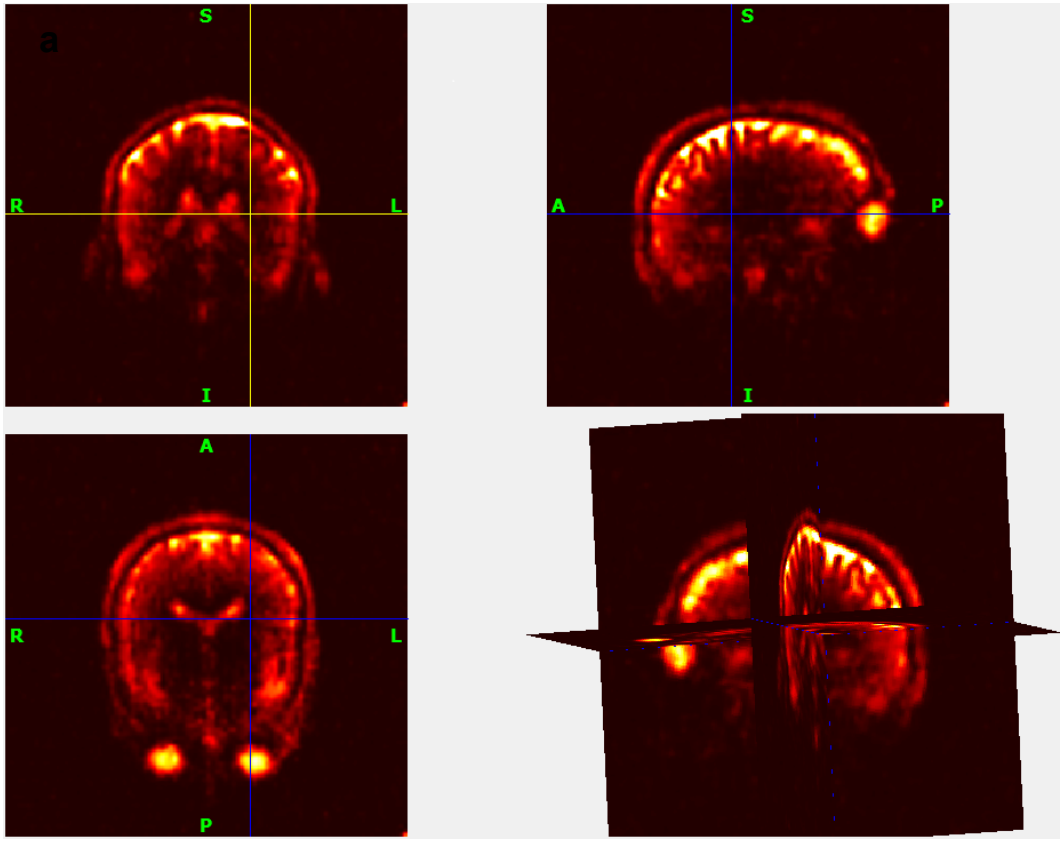

Figure 6.5: MC-MLP 3D denoising

MC-MLP algorithm denoising of (Fig. 6.4). Axial, sagittal, coronal and 3D views are shown for the denoised sample. Plotted using BrainSuite software [90]

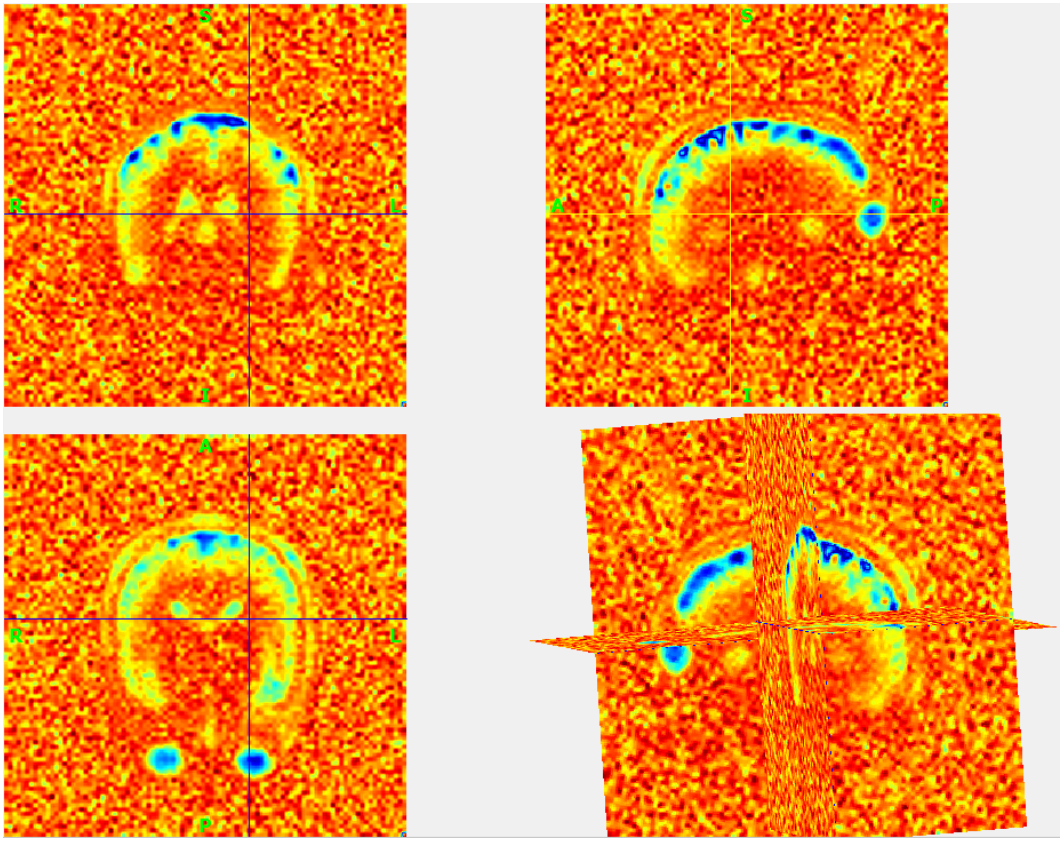

Figure 6.6: Removed noise.

Absolute difference of The MC-MLP denoised results and the mean. Plotted using BrainSuite software [90]

APPENDIX A

Supplementary Information for Chapter 5

A.1 Supplementary methods

A.1.1 Adaptive control algorithm

The working principle of the search algorithm is illustrated in (Fig. A.1), the pseudocode is provided in Algorithm 1 below whereas its parameters are listed in Table A.1. The search algorithm typically starts with an arbitrary shear rate map specified by the user. Mean values from different regions of the map are fed to the multi-layer perceptron (MLP) as its input, and the output produced by the MLP is used to determine the flow speeds of all 10 inlets required to reproduce the desired target map as closely as possible. A new shear rate map corresponding to the inlets' flow speeds generated by the MLP is obtained and compared to the desired pattern using the cost function ξ of Eq. (A.3). Typically a cost function that measures the Euclidian distance between a target vector and an evaluated vector can be used as in Eq. (A.1). Including a Gaussian membership function as shown in Eq. (A.2) can relax the measurement constraints by allowing a small margin of error where an approximate measurement is considered acceptable. d determines the width of the Gaussian function. Using a d value between 2 and 3 "smoothes" the error space, which leads to faster convergence. Finally, in order to prevent the search algorithm from being biased towards relatively large target vector components, a normalization is performed by dividing by the target vector as shown in Eq. (A.3). For example, a target shear rate map can have high

variations in shear rate values between different regions. Regions with relatively high shear rate values will have more weight in the cost function. This can cause the obtained solution to be less accurate in regions with relatively low shear rate values. Normalizing as in Eq. (A.3) allows each region to have an equal weight in the cost function regardless of the target value in that region. When O_j and T_j are close, the Gaussian function yields a value close to one, and the contribution of the corresponding measurement to the cost function is small. When they are far, the Gaussian function yields a value close to zero, and the contribution to the cost function is large. O_j denotes a measurement (observed value) whereas T_j denotes the corresponding desired measurement (target value). By measurement we refer to a mean shear rate value associated with one of the scaffold regions. In our case, we have partitioned the scaffold into $D = 9$ regions.

The initial MLP output does not yield an accurate solution. The search algorithm aims to find a solution that is slightly better than the one provided by the MLP, by following a number of rules to introduce changes to the MLP output as shown in the pseudocode. The new solution is used to update the MLP weights as explained in the following section. This guides the MLP to gradually learn the relationship between the inlets' flow speeds and the target map after a number of iterations. The algorithm was validated by applying it to a 2-input control problem where the convergence path through the error space can be visualized on a 3D plot. Results from the validation are shown in (Figure A.2).

$$\xi = \sqrt{\sum_{j=1}^{D} (O_j - T_j)^2} \tag{A.1}$$

$$\xi = \sqrt{\sum_{j=1}^{D} \left(T_j - T_j e^{\frac{-(O_j - T_j)^2}{2d^2}} \right)^2} \tag{A.2}$$

$$\xi = \sqrt{\sum_{j=1}^{D} \left(\frac{T_j - T_j e^{\frac{-(O_j - T_j)^2}{2d^2}}}{T_j} \right)^2} = \sqrt{\sum_{j=1}^{D} \left(1 - e^{\frac{-(O_j - T_j)^2}{2d^2}} \right)^2} \tag{A.3}$$

Algorithm 1: The control algorithm is described by the pseudo-code below:

Initiate MLP inputs
Evaluate shear stress map for MLP outputs
Compare to target
$\xi_b \leftarrow \xi$
$\xi_1 \leftarrow \xi$
count = 0
Reset
flag= 0
Reset input picking
Pick
Randomly pick one of the inputs
Increase its speed by *ds*
Evaluate shear stress map for resulting input flow speeds
if $\xi > \xi_b$ **then**
    Decrease its speed by *ds*
    Evaluate shear stress map for resulting input flow speeds
    if $\xi > \xi_b$ **then**
        goto *Check*
    else
        goto *update*
else
update
    $\xi_b \leftarrow \xi$

```
    count = 0
    Update MLP
    flag = 1
Check
    if All inputs picked then
        if flag = 1 then
            goto Reset
        else
            count = count + 1
            if count > bcount then
                ξ_b ← ξ
                count = 0
                Update MLP
                goto Reset
            else
Perturb
                Add random vector multiple of ds to input flow speeds
                Evaluate shear stress map for resulting input flow speeds
                if ξ > ξ_b then
                    goto Perturb
                else
                    ξ_b ← ξ
                    count = 0
                    Update MLP
                    goto Reset
    else
        goto Pick
```

Table A.1: List of the algorithm's main parameters.

Parameter	Description	Range	Used Value
μ	Learning rate	μ_{min}:μ_{max}	Automated
a	Momentum	a_{min}:a_{max}	Automated
μ_{min}	μ lower bound	>0	1.00E-06
μ_{max}	μ upper bound	>μ_{min}	5.00E-02
a_{min}	a lower bound	>0	1.00E-06
a_{max}	a upper bound	>a_{min}	5.00E-02
dso	Search step size multiplier	>0	5.00E-01
ds	Search step size: ξ.dso	>0	Automated
ξ	Cost function value	NA	Automated
D	Number of MLP inputs	>0	9
K	Number of MLP outputs	>0	10
boutput	MLP output with current minimum cost	NA	NA
ξ_b	Current best cost value	NA	NA
bcount	Maximum MLP training epochs for current best output	>0	3
d	membership function dilatation parameter	>0	3

A.1.2 Multi-layer perceptron (MLP)

MLP have been shown to be universal function approximators [56]. A MLP with D inputs, K outputs, one hidden layer with M nodes yields a K-dimensional output vector $\vec{\tilde{y}}_t$ whose k-th component is given by the iterated hyperbolic tangents:

$$\tilde{y}_t(k) = \tanh\left(\sum_{l=0}^{M} \theta_{l,k}^{(2)} \tanh\left(\sum_{j=0}^{D} \theta_{j,l}^{(1)} x_t(j)\right)\right) \tag{A.4}$$

where

$$z(l) = \tanh\left(\sum_{j=0}^{D} \theta_{j,l}^{(1)} x_t(j)\right) \tag{A.5}$$

for which $l = 0, \ldots, M$ are the outputs of the hidden layer. We use the convention where $x_t(0) = 1$ and $z(0) = 1$, so that $\theta_{0,l}^{(1)}$ and $\theta_{0,k}^{(2)}$ represent biases to the transfer function. The generalization to arbitrary numbers of hidden layers is straightfor-

ward by nesting additional hyperbolic tangents. The calculation of the vector $\vec{\tilde{y}}$ is called feed forward propagation.

A MLP with 9 inputs corresponding to the 9 scaffold regions, and 10 outputs corresponding to the 10 channels is used. Mean values from nine regions of the shear rate map are stored in a vector denoted by $\vec{x}_t$ and used as inputs to a MLP whose transfer function is a hyperbolic tangent, has 10 outputs corresponding to the 10 inlets of the bioreactor, and includes two hidden layers with 10 neurons each. The Back-Propagation with Adaptive Learning rate and Momentum term method (BPALM) from [58] was used to update the MLP weights. This technique is suitable in our case since the MLP is trained online one target vector at a time, and since it offers convergence acceleration as compared to the original back propagation method proposed by [57] by introducing an adaptive learning rate and momentum term.

A.1.3 Nuclear magnetic resonance imaging of 10-channel bioreactor

A spin-echo nuclear magnetic resonance (NMR) imaging pulse sequence (Fig. A.3) was modified for phase-contrast flow velocimetry [36] as in [49, 89] All gradients except for the phase-encoding (P.E.) gradient include flow-compensation (F.C.). The F.C. calculations utilize the full trapezoidal shape of the gradient pulse. Bipolar flow-weighting (F.W.) gradients were added along x,y, and z directions to select the gradient first moment (M_1). With a stationary (no-flow) sample, the nuclear spins experience the full magnitude of both positive and negative lobes of the bipolar gradient, resulting in zero net phase accumulation. In the case of constant-velocity flow, the nuclear spins move during the application of the two gradients, and phase-cancellation is incomplete, giving a residual phase proportional to the velocity [36]. Two experiments were performed under flow with two gradient values $+M_1$ and $-M_1$. M_1 is chosen to include the highest anticipated flow velocity and thus avoid phase wrap-around artifacts. The two experiments

are subtracted in order to calculate a velocity map. In order to compensate for gradient non-idealities (e.g. eddy currents and nonlinearities seen during slow fluid flow in biomaterials), a "flow/no flow" subtraction procedure was utilized; for each direction of flow, the contribution from non-idealities was subtracted from an identical experiment performed without fluid flow. The imaging parameters were: TR = 1 s, TE = 34 ms, 128 $\times$ 64 matrix, and field of view (FOV) was 10 cm $\times$ 3 cm. All measurements were performed on a 9.4 T vertical bore Varian VNMRS micro-imaging system, using a 40 mm-i.d. imaging probe. Shear measurements were calculated as in [49, 89].

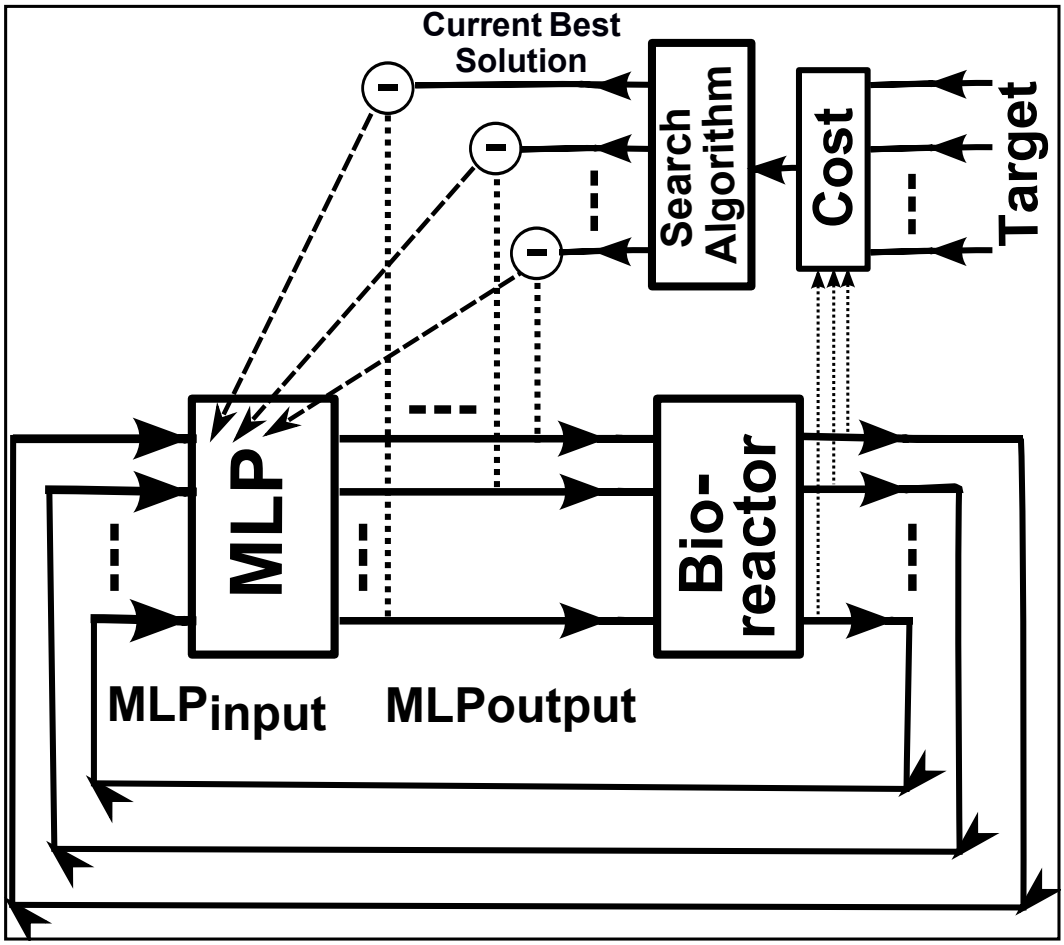

Figure A.1: System design of the control algorithm.

Multi layer perceptron block (MLP) gets its inputs from shear map measurements of the scaffold inside the bioreactor, and produces outputs to update inlets' flow speeds such that the shear rate map in the scaffold is closer to the target shear rate map. The MLP training is guided by the search algorithm that evaluates the MLP performance using the cost function evaluation block (Cost), and determines the training samples presented to the MLP.

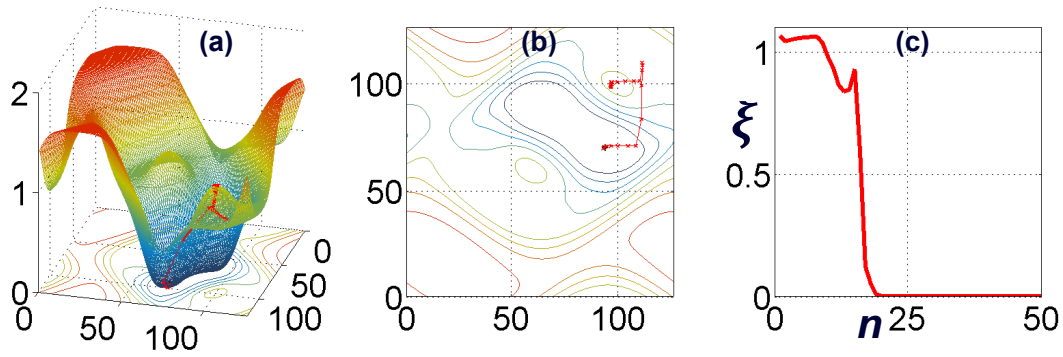

Figure A.2: 2-input control validation results.

The convergence path shows the search starts at a maxima, avoids getting stuck in a local minima and finds its way to a global minima in a small number of steps. (a) 3D plot of the error space. (b) 2D contour projection of the error space. (c) Convergence plot with y-axis corresponding to the cost function and x-axis corresponding to the iteration number.

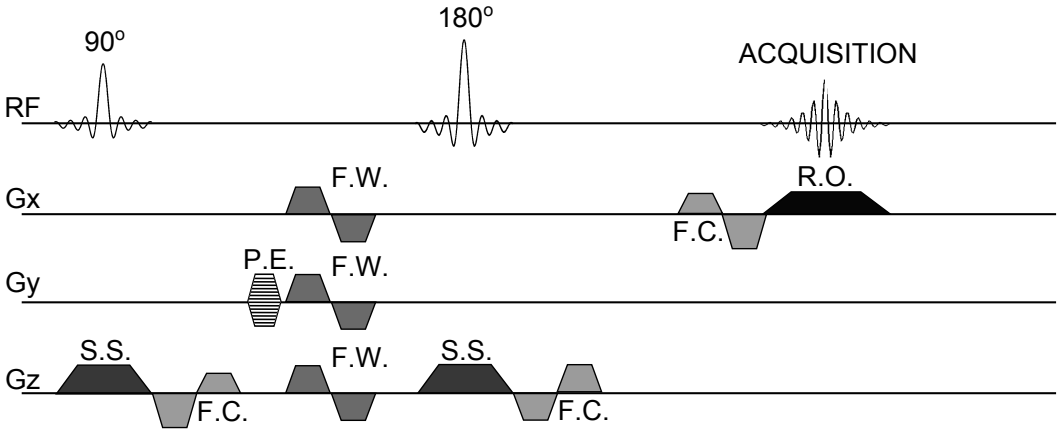

Figure A.3: NMR imaging pulse sequence.

Slice-selective (S.S.) three-dimensional phase-contrast velocimetry spin-echo nuclear magnetic resonance (NMR) imaging pulse sequence. In order to get a 2-D image, the sequence is repeated n times while stepping the phase-encode (P.E.) gradient through the phase-encoding scheme. Bipolar trapezoidal flow-weighting (F.W.) gradients were added along the x,y, and z directions to select the gradient first moment (M_1). Flow-compensation (F.C) gradients are added along x and z directions. Additional abbreviations: S.S. slice select gradient; R.O. read out (frequency encode) gradient.

APPENDIX B

Code

The code was implemented and tested using MATLAB (Mathworks, Natick, MA) version R2013a.

B.1 Genetic Algorithm Function

This function needs to be executed from the optimization toolbox:

solver: ga - Genetic Algorithm

Fitness function: @LBM_genSChv3128a5

Number of variables: 18

```
function error = LBM_genSChv3128a5(xx)
tic
warning off all
xx = xx - min(xx);
xx = xx / max(xx);
goal = 0.5e-5;
goalprc = 20;
% %%%%%%%%%%%%%%%%%%%%initialize%%%%%%%%%%%%%%%%%
scl = 2;
omega=1;
density=1;
t1=4/9;
t2=1/9;
t3=1/36;
c_squ=1/3;
nx=64*scl-1;
ny=64*scl-1;
tau = 1/omega;
viscosity = (tau-0.5)/3;
F=repmat(density/9,[nx ny 9]);
FEQ=F;
msize=nx*ny;
CI=[0:msize:msize*7];
BOUND=(rand(nx,ny)>0.6);
BOUND(1,1:ny)=1;
BOUND(nx,1:ny)=1;
BOUND(1:nx,1)=1;
BOUND(1:nx,ny)=1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = 800;
W = 800;
w = 2;
vrs = 4;
mnsz = L/4;

A = zeros(L,W);
kk = 1;
for ii = 1:w
    x = 0;
    x(1:vrs) = xx(kk:kk+vrs-1);
    kk = kk + vrs;
x(W) = 0;
    y = idct(x);
Y = y - min(y);
    Y = abs(Y - Y(1));
    Y = Y - min(Y);
    Y = floor(xx(kk)*(L-1)*Y/max(Y)) + 1;
    kk = kk + 1;


    for i = 1:W
        A(i,Y(i)) = 1;
    end
end
```

```
Geom = A';

mGeom = 0;
mGeom(W,L) = 0;
for i = 1:L
    mGeom(:,i) = Geom(:,L-i+1);
end

Geom(:,L+1:2*L) = mGeom;

mGeom = 0;
mGeom(W,2*L) = 0;
for i = 1:W
    mGeom(i,:) = Geom(W-i+1,:);
end

tGeom = Geom;
Geom = mGeom;
Geom(W+1:2*W,:) = tGeom;


Geom = double(imresize(Geom,[mnsz,mnsz]) ~= 0);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Geom2 = zeros(nx,ny);
Geom2(:,floor(ny/7)+1:floor(ny-ny/7)) = double(imresize(Geom,[nx,floor(ny-
(2*ny/7))]) ~= 0);

se = strel('line',3,45);
bw1 = imdilate(Geom2,se);
se = strel('line',3,-45);
bw2 = imdilate(Geom2,se);
bw = bw1 | bw2;

timg = Geom2*255;
timgc(:,:,1) = timg(:,:,1);
timgsp = not(timgc' == 0);
timgss = timgc ~= 0 & timgc ~= 255;
timgsp = imresize(timgsp,[nx,ny]);
timgss = imresize(timgss,[nx,ny]);
BOUND = not(not(BOUND) | timgsp');
BOUND = BOUND | timgss;

flnm = ['C:\Users\arj\Documents\MATLAB2\2D9QLBM\mask12.bmp'];
timg1 = imread(flnm);
timgc1(:,:,1) = timg1(:,:,1);
timgsp1 = not(timgc1' == 0);
timgss1 = timgc1 ~= 0 & timgc1 ~= 255;
timgsp1 = imresize(timgsp1,[nx,ny]);
timgss1 = imresize(timgss1,[nx,ny]);
BOUND = not(not(BOUND) | timgsp1');
BOUND = BOUND | timgss1;
```

```
BOUND = BOUND';

ON=find(BOUND);
TO_REFLECT=[ON+CI(1) ON+CI(2) ON+CI(3) ON+CI(4) ...
            ON+CI(5) ON+CI(6) ON+CI(7) ON+CI(8)];
REFLECTED= [ON+CI(5) ON+CI(6) ON+CI(7) ON+CI(8) ...
            ON+CI(1) ON+CI(2) ON+CI(3) ON+CI(4)];
avu=1;
prevavu=1;
ts=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xx(kk) = xx(kk)*1e-2;
xx(kk) = max(xx(kk),1e-7);
xx(kk) = min(xx(kk),1e-2);
deltaU = xx(kk);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
kk = kk +1;

numactivenodes=sum(sum(1-BOUND));

dUXbdx = 0;
dUXbdy = 0;
dUYbdx = 0;
dUYbdy = 0;
dUXbdx(nx,ny) = 0;
dUXbdy(nx,ny) = 0;
dUYbdx(nx,ny) = 0;
dUYbdy(nx,ny) = 0;
posx = 0;
szzx = nx;
posy = 0;
szzy = ny;
SS = 0;
SS(szzx-posx,szzy-posy) = 0;
SSv = 0;
SSv(szzx-posx*szzy-posy) = 0;

hold off

while (ts<2000)

    F(:,:,4)=F([2:nx 1],[ny 1:ny-1],4);
    F(:,:,3)=F(:,[ny 1:ny-1],3);
    F(:,:,2)=F([nx 1:nx-1],[ny 1:ny-1],2);
    F(:,:,5)=F([2:nx 1],:,5);
    F(:,:,1)=F([nx 1:nx-1],:,1);
    F(:,:,6)=F([2:nx 1],[2:ny 1],6);
    F(:,:,7)=F(:,[2:ny 1],7);
    F(:,:,8)=F([nx 1:nx-1],[2:ny 1],8);
    BOUNCEDBACK=F(TO_REFLECT);
    DENSITY=sum(F,3);

    UX=(sum(F(:,:,[1 2 8]),3)-sum(F(:,:,[4 5 6]),3))./DENSITY;
```

```
    UY=(sum(F(:,:,[2 3 4]),3)-sum(F(:,:,[6 7 8]),3))./DENSITY;

    UX(1,1:ny)=UX(1,1:ny)+deltaU;

    UX(ON)=0;
    UY(ON)=0;
    DENSITY(ON)=0;
    U_SQU=UX.^2+UY.^2;
    U_C2=UX+UY;
    U_C4=-UX+UY;
    U_C6=-U_C2;
    U_C8=-U_C4;

    FEQ(:,:,9)=t1*DENSITY.*(1-U_SQU/(2*c_squ));
    FEQ(:,:,1)=t2*DENSITY.*(1+UX/c_squ+0.5*(UX/c_squ).^2-U_SQU/(2*c_squ));
    FEQ(:,:,3)=t2*DENSITY.*(1+UY/c_squ+0.5*(UY/c_squ).^2-U_SQU/(2*c_squ));
    FEQ(:,:,5)=t2*DENSITY.*(1-UX/c_squ+0.5*(UX/c_squ).^2-U_SQU/(2*c_squ));
    FEQ(:,:,7)=t2*DENSITY.*(1-UY/c_squ+0.5*(UY/c_squ).^2-U_SQU/(2*c_squ));
    FEQ(:,:,2)=t3*DENSITY.*(1+U_C2/c_squ+0.5*(U_C2/c_squ).^2-
U_SQU/(2*c_squ));
    FEQ(:,:,4)=t3*DENSITY.*(1+U_C4/c_squ+0.5*(U_C4/c_squ).^2-
U_SQU/(2*c_squ));
    FEQ(:,:,6)=t3*DENSITY.*(1+U_C6/c_squ+0.5*(U_C6/c_squ).^2-
U_SQU/(2*c_squ));
    FEQ(:,:,8)=t3*DENSITY.*(1+U_C8/c_squ+0.5*(U_C8/c_squ).^2-
U_SQU/(2*c_squ));
    F=omega*FEQ+(1-omega)*F;
    F(REFLECTED)=BOUNCEDBACK;
    ts=ts+1;
end

dUXbdx(2:end-1,2:end-1) = (UX(3:end,2:end-1) - UX(1:end-2,2:end-1))/2;
dUXbdy(2:end-1,2:end-1) = (UX(2:end-1,3:end) - UX(2:end-1,1:end-2))/2;
dUYbdx(2:end-1,2:end-1) = (UY(3:end,2:end-1) - UY(1:end-2,2:end-1))/2;
dUYbdy(2:end-1,2:end-1) = (UY(2:end-1,3:end) - UY(2:end-1,1:end-2))/2;

for ssi = posx+1:posx+szzx-1
    for ssj = posy+1:posy+szzy-1
            ssU = [dUXbdx(ssi,ssj) dUXbdy(ssi,ssj); dUYbdx(ssi,ssj)
dUYbdy(ssi,ssj)];
            ssU = ssU + ssU';
            SS(ssi-posx,ssj-posy) = max(abs(eig(ssU)))* viscosity;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%error%%%%%%%%%%%%%%%%%%%%%%%%
SSm = 0;
for ssi = 20:107
    for ssj = 2:126
        if timgc(ssj,ssi) == 0
        SSm(ssi-19,ssj-1) = SS(ssi,ssj);
        end
    end
 end
SSmskh = double(SSm > goal + goal*(goalprc/100));
SSmskl = double(SSm < goal - goal*(goalprc/100));
for ssi = 1:length(SSm(:,1))
```

```
SSv((ssi-1)*(length(SSm(1,:)))+1:ssi*(length(SSm(1,:)))) = SSm(ssi,:);
end

ssk = 1;
ssk2 = 1;
SSvnz = 0;
SSvmsk = SSv > 0;
SSvnz(sum(SSvmsk)+1) = 0;
SSvnznh = 0;
SSvnznh2 = 0;

for ssi = 1: length(SSv)
if SSv(ssi) > 0
    SSvnz(ssk) = SSv(ssi);
    if SSv(ssi) < 2*10^-5
        SSvnznh(ssk) = SSv(ssi);
            ssk = ssk +1;
    end
end
    if SSv(ssi) > 1e-8 && SSv(ssi) < 6e-5
        SSvnznh2(ssk2) = SSv(ssi);
            ssk2 = ssk2 +1;
    end
end

inrng = (sum(sum((SSm>(goal - goal*(goalprc/100)) & SSm<(goal +
goal*(goalprc/100))))))/sum(sum(SSm>0)));
soa = (std(SSvnz)/mean(SSvnz));
par1 = 0.75e6*abs(goal - mean(SSvnz));
par2 = 5.5*soa;
par3 = inrng*40;
pars = [par1 par2 par3];
error = generrfuz1(pars)*1000/30;

if sum(sum(timgsp)) < 5000
    error = inf;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Error =
csvread('C:\Users\arj\Documents\MATLAB2\2D9QLBM\SCh128a5\errorbksch.txt');

if error(end) < Error(end) - 1

    Error(end+1) = error(end);
    strng = [num2str(error(end)) ', ' num2str(par1) ', ' num2str(par2) ', '
num2str(par3)];
   disp(strng);


   U_SQUn = U_SQU - min(min(U_SQU));
U_SQUn = U_SQUn / max(max(U_SQUn));
subplot(1,3,1)
```

```
pp = ((U_SQUn(1:nx,:)*255)') + 50*(((BOUND(1:nx,:)==0)')) -
0*(((BOUND(1:nx,:)==1)'));
image(pp);
colormap(jet(255))

title(['Flow field']);xlabel(['x   input:'  num2str(deltaU)]);ylabel('y');
subplot(1,3,2)
hist((SSvnznh2 - mean(SSvnznh2))./(std(SSvnznh2)),50);
title(['Shear Stress Distribution']);
xlabel(['Error: ',num2str(error/1e0)])
ylabel(['std/avg: ',num2str(std(SSvnz)/mean(SSvnz))])

subplot(1,3,3)
imagesc(log(10^6*SSm'+1))
caxis([0 4])
title(['Shear Stress']);
xlabel(['Average Shear Stress: ',num2str(mean(SSvnz))])
ylabel(['In range: ',num2str(sum(sum((SSm>(goal - goal*(goalprc/100)) &
SSm<(goal + goal*(goalprc/100)))))/sum(sum(SSm>0)))])

%%%%%%%%%%%%%%%%%%%%%%display%%%%%%%%%%%%%%%%%%%%%%

csvwrite('C:\Users\arj\Documents\MATLAB2\2D9QLBM\SCh128a5\errorbksch.txt',Err
or);

   step = csvread('2D9QLBM\SCh128a5\step.txt');
   step = step + 1;
   csvwrite('C:\Users\arj\Documents\MATLAB2\2D9QLBM\SCh128a5\step.txt',step);
   flnm = ['C:\Users\arj\Documents\MATLAB2\2D9QLBM\SCh128a5\opt'
int2str(step) '.tif'];
   saveas(gcf,flnm,'tif');
   flnmt = ['C:\Users\arj\Documents\MATLAB2\2D9QLBM\SCh128a5\opt'
int2str(step) '.txt'];
   csvwrite(flnmt,xx);

   pause(0.01)
end

warning on all
tym = csvread('C:\Users\arj\Documents\MATLAB2\2D9QLBM\SCh128a5\time.txt');
   tym = tym + toc;
   csvwrite('C:\Users\arj\Documents\MATLAB2\2D9QLBM\SCh128a5\time.txt',tym);
   tym/60
end
```

B.2 Computing Hydraulic Conductivity

```
vx = res_vz(20:115,80:120); %Adjust to NMR data
vy = res_vy(20:115,80:120); %Adjust to NMR data
v = (vx.^2+vy.^2).^.5;

P = zeros(length(vx(:,1)),length(vx(1,:)));

density = 1000;%kg/m^3
visc = 0.001;%kg/(m.s)
kvisc = visc/density;
dx = 1e-4;
dy = dx;

h = 1e-4;%m

for i = 1:length(P(:,1))-1
    for j = 1:length(P(1,:))-1
        gvs(i,j) = ((vx(i+1,j)- vx(i,j) )*(1/dx)*( vx(i+1,j)- vx(i,j)*(1/dx))
+ ...
                (vx(i,j+1)- vx(i,j) )*(1/dx)*( vy(i+1,j)- vy(i,j)*(1/dx)) +
...
                (vx(i,j+1)- vx(i,j) )*(1/dx)*( vy(i+1,j)- vy(i,j)*(1/dx))
+...
                (vy(i,j+1)- vy(i,j) )*(1/dx)*( vy(i,j+1)- vy(i,j))*(1/dx));
    end
end
b = -density*gvs*(h^2);

cns = inf;
Pt = P;
while cns > sum(sum(P.*P))*1e-10
    for i = 2:length(P(:,1))-1
        for j = 2:length(P(1,:))-1
            P(i,j) = ((P(i-1,j)+P(i+1,j)+P(i,j-1)+P(i,j+1))+b(i,j))/(4);
        end
    end
    P(:,1) = P(:,2);
    P(:,end) = P(:,end-1);
    P(1,:) = P(2,:);
    P(end,:) = P(end-1,:);
    P(:,1) = 26600/2;%kg/(m.s^2)
    P(21:79,end) = 0;%Adjust to outlet position

   imagesc(P)
   axis square
   title(num2str(sum(sum(P))))
   pause(0.001)
   cns = sum(sum((Pt - P).*(Pt - P)));
   Pt = P;
end

imagesc(v*1e7)
hold on
contour(P);
colorbar
hold off
```

```
axis square
title('pressure')
pause

for i = 2:length(P(:,1))-1
    for j = 2:length(P(1,:))-1
        gpx(i,j) = (P(i+1,j)- P(i,j));
        gpy(i,j) = (P(i,j+1)- P(i,j));
    end
end

gp = (gpx.^2+gpy.^2).^.5;

imagesc(v)
axis square
title('v')
colorbar
pause
contour(gp)
axis square
title(['\delta' 'p'])
colorbar
pause

imagesc(vx)
axis square
title('vx')
colorbar
pause
contour(abs(gpx))
axis square
title(['\delta' 'px'])
colorbar
pause

imagesc(vy)
axis square
title('vy')
colorbar
pause
contour(abs(gpy))
axis square
title(['\delta' 'py'])
colorbar
pause

k = (h*visc*v(2:end-2,2:end-2)./abs(gp(2:end-1,2:end-1)));

kx = (h*visc*abs(vx(2:end-2,2:end-2))./abs(gpx(2:end-1,2:end-1)));
ky = (h*visc*abs(vy(2:end-2,2:end-2))./abs(gpy(2:end-1,2:end-1)));

for i = 2:length(gp(:,1))-1
    for j = 2:length(gp(1,:))-1
        A = [abs(gpx(i,j)); abs(gpy(i,j))];
        B = [abs(vx(i,j)); abs(vy(i,j))];
```

```
        xx = (inv(A'*A))*A'*B;
        k2(i,j) = xx;
    end
end

kx = kx(2:end-1,2:end-1);
ky = ky(2:end-1,2:end-1);
k2 = k2(2:end-1,2:end-1);

k = k(2:end-1,2:end-1);
imagesc(v(2:end-2,2:end-2)*2e-5)
hold on
contour(k)
axis square
axis off
title('permeability')
colorbar
hold off
pause

average_k = mean(mean(k))

imagesc(log(k(2:end-3,2:end-2)*1e15));
c=colorbar;
caxis([0 7]);
t=title(['log permeability 100' '\mu' 'L']);
axis square; axis off;
set(c,'fontsize',16); set(t,'fontsize',16);
```

B.3 MCMLP

B.3.1 MCMLP training

```
clear
close all

tic

Resol = 192;
TRes = 192;
NL = 1;
sa = 10
sm = 1;
rad = 2;
rad2 = 2;
w2d = (rad*2+1)^2;
w2d2 = (rad2*2+1)^2;
w3d = 0;
wg = 10;
wg1 = 3;
fthresh = 1;
thresh = .025;
noc = 7;

comb = nchoosek(1:noc,(noc-1));
nocom = length(comb(:,1));

BrTstDt = squeeze(mean(imread('sample1.Jpg'),3));
ReConI = BrTstDt;
%%%%%%%%%%%%Load original image for comparison complete.
ReConI =abs(imresize(ReConI,[Resol,Resol]));
ReConI = ReConI / max(max(ReConI));
ReConI2 = ReConI;

C = ReConI2;
Resolx = size(ReConI,2);
Resoly = size(ReConI,1);
sigma = sa;
map = 1;
for jj = 1:noc
    imwrite(sqrt((C+sa/100.*randn(size(C))).^2 +
(C+sa/100.*randn(size(C))).^2),'NoisyTmp.Jpg');
    TMP = squeeze(mean(imread('NoisyTmp.Jpg'),3));
    TMP =abs(imresize(TMP,[Resol,Resol]));
    TMP = TMP / max(max(TMP));
    Data2bR0(jj,:,:) = zeros(Resoly+2*rad+1,Resolx+2*rad+1);
    Data2bR0(jj,rad+1:Resoly+rad,rad+1:Resolx+rad) = TMP;
end
Noisy = squeeze((Data2bR0(1,rad+1:Resoly+rad,rad+1:Resolx+rad)));

comb = nchoosek(1:noc,(noc-1));
nocom = length(comb(:,1));
for jj = 1:noc
    Data2bR(jj,:,:) = squeeze(mean(Data2bR0(comb(jj,:),:,:)));
end
%%%%%%%%%%Introducing noise to image complete.
```

```
UKData = zeros(noc,w2d,(Resolx)*(Resoly));
for jj = 1:noc
    CIm = squeeze(Data2bR(jj,:,:));
    CDt = zeros(w2d,(Resolx)*(Resoly));
    CTg = zeros(1,(Resolx)*(Resoly));
    l = 1;
        for j = 1+rad:Resoly+rad
            for k = 1+rad:Resolx+rad
                CDt(:,l) = reshape(CIm(j-rad:j+rad,k-
rad:k+rad),[1,(2*rad+1)^2]);
                CTg(l) = ReConI(j-rad,k-rad);
                l = l + 1;
            end
        end
    UKData(jj,:,:) = CDt;
end
Goal = CTg;
ReConO = reshape(Goal,[Resoly,Resolx])';
Mean = reshape(squeeze(mean(UKData(:,ceil(end/2),:))),[Resoly,Resolx])';

for jj = 1:noc
    TMP = UKData(comb(jj,:),:,:);
    for kk = 1:(noc-1)
        UKDataMC(jj,(kk-1)*w2d+1:kk*w2d,:) = squeeze(TMP(kk,:,:));
    end
end
%%%%%%%%%%%%Unknown sample data preperation complete.

IData2o(1,:,:) = squeeze(mean(imread('brtr1.Jpg'),3));
IData2o(2,:,:) = squeeze(mean(imread('brtr2.Jpg'),3));
IData2o(3,:,:) = squeeze(mean(imread('brtr3.Jpg'),3));
IData2o(4,:,:) = squeeze(mean(imread('brtr4.Jpg'),3));
IData2o(5,:,:) = squeeze(mean(imread('brtr5.Jpg'),3));
IData2o(6,:,:) = squeeze(mean(imread('brtr6.Jpg'),3));
IData2o(7,:,:) = squeeze(mean(imread('brtr7.Jpg'),3));
IData2o(8,:,:) = squeeze(mean(imread('brtr8.Jpg'),3));
IData2o(9,:,:) = squeeze(mean(imread('brtr9.Jpg'),3));

IData2 = abs(imresize(squeeze(IData2o(1,:,1:round(end/2))),[TRes,TRes]));
IData2 = IData2 / max(max(IData2));

IData2(end+1:end+TRes,:) =
abs(imresize(squeeze(IData2o(5,:,:)),[TRes,TRes]));
IData2(end-TRes+1:end,:) = IData2(end-TRes+1:end,:) / max(max(IData2(end-
TRes+1:end,:)));

IData2(end+1:end+TRes,:) =
abs(imresize(squeeze(IData2o(9,:,:)),[TRes,TRes]));
IData2(end-TRes+1:end,:) = IData2(end-TRes+1:end,:) / max(max(IData2(end-
TRes+1:end,:)));

Resolxt = size(IData2,2);
Resolyt = size(IData2,1);
L = [(Resolyt/Resolxt)*Resolxt^2, (Resolyt/Resolxt)*Resolxt^2,
(Resolyt/Resolxt)*Resolxt^2, (Resolyt/Resolxt)*Resolxt^2];
%%%%%%%%%%%%Loading training data complete.
```

```
C = IData2;
for jj = 1:noc
    imwrite(sqrt((C+sa/100.*randn(size(C))).^2 +
(C+sa/100.*randn(size(C))).^2),'NoisyTmp.Jpg');
    TMP = squeeze(mean(imread('NoisyTmp.Jpg'),3));
    TMP = abs(imresize(TMP,[size(IData2,1),size(IData2,2)]));
    TMP = TMP / max(max(TMP));
    TRDataa(jj,:,:) = zeros(Resolyt+2*rad,Resolxt+2*rad);
    TRDataa(jj,rad+1:Resolyt+rad,rad+1:Resolxt+rad) = TMP;
end

for jj = 1:noc
    TRData(jj,:,:) = squeeze(mean(TRDataa(comb(jj,:),:,:)));
end
%%%%%%%%%%Introducing noise to image complete.

TRData0 = zeros(noc,w2d,(Resolxt)*(Resolyt));
for jj = 1:noc
    CIm = squeeze(TRData(jj,:,:));
    CDt = zeros(w2d,(Resolxt)*(Resolyt));
    CTg = zeros(1,(Resolxt)*(Resolyt));
    l = 1;
        for j = 1+rad:Resolyt+rad
            for k = 1+rad:Resolxt+rad
                CDt(:,l) = reshape(CIm(j-rad:j+rad,k-
rad:k+rad),[1,(2*rad+1)^2]);
                CTg(l) = IData2(j-rad,k-rad);
                l = l + 1;
            end
        end
    TRData0(jj,:,:) = CDt;
end
TGt0 = CTg;

for jj = 1:noc
    TMP = TRData0(comb(jj,:),:,:);
    for kk = 1:(noc-1)
        TRData0MC(jj,(kk-1)*w2d+1:kk*w2d,:) = squeeze(TMP(kk,:,:));
    end
end
%%%%%%%%%%Training data preperation complete.

display('Stage1 training begins...')
figure
subplot(3,3,1)
imagesc(ReConO);
axis square
colormap hot
title('Original');
subplot(3,3,2)
imagesc(Noisy);
axis square
colormap hot
title(['Noisy PSNR: ' mat2str(getPSNR(Noisy, ReConO, 1))]);
for jj = 1:noc
    net1 = feedforwardnet([10 10 10 10 10 10]);
```

```
    net1.trainParam.epochs = 40;
    net1 =
train(net1,[squeeze(TRData0MC(jj,:,1:L(1)))],[TGt0(1:L(1))],'useParallel','ye
s');

    NET1(jj).net = net1;
    Xmin1(jj,:) = min(squeeze(TRData0MC(jj,:,1:L(1)))');
    Xmax1(jj,:) = max(squeeze(TRData0MC(jj,:,1:L(1)))');
    Tmin1(jj,:) = min(TGt0(1:L(1)));
    Tmax1(jj,:) = max(TGt0(1:L(1)));

    TRData1f2(jj,:) = net1(squeeze(TRData0MC(jj,:,:)));
    TMP = reshape(TRData1f2(jj,:),[Resolxt,Resolyt])';
    RESULT1Tr(jj,:,:) = TMP;

    UKData1f2(jj,:) = net1(squeeze(UKDataMC(jj,:,:)));
    TMP = reshape(UKData1f2(jj,:),[Resolx,Resoly])';
    RESULT1(jj,:,:) = TMP;
    subplot(3,3,jj+2)
    imagesc(TMP);
    axis square
    colormap hot
    title(['N' mat2str(jj) ' PSNR: ' mat2str(round(getPSNR(TMP, ReConO,
1)*100)/100)]);
    PSNR1(jj) = getPSNR(TMP, ReConO, 1);
    [FSIM, FSIMc] = FeatureSIM(ReConO*255, TMP*255);
    FSIM1(jj) = FSIM;
    [mssim ssim_map] = ssim_index(ReConO*255, TMP*255);
    MSSIM1(jj) = mssim;
    title(['N' mat2str(jj) ' PSNR: ' mat2str(PSNR1(jj))]);
    pause(.01)
end
for jj = 1:noc
    TMP = reshape(TRData1f2(jj,:),[Resolxt,Resolyt])';
    TRData1f2tr(jj,:,:) = zeros(Resolyt+2*rad2,Resolxt+2*rad2);
    TRData1f2tr(jj,rad2+1:Resolyt+rad2,rad2+1:Resolxt+rad2) = TMP;
    TMP = reshape(UKData1f2(jj,:),[Resolx,Resoly])';
    UKData1tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
    UKData1tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
end
display('Stage1 training complete')
fprintf('Av. PSNR: %.4fdB, Av. FSIM: %.4fdB, Av. MSSIM: %.4fdB
\n',mean(PSNR1),mean(FSIM1),mean(MSSIM1));
%%%%%%%%%%Stage1 training complete
TRData0MC=0;
UKDataMC=0;

for jj = 1:noc
    UKData1ctr(jj,:,:) = squeeze(mean(UKData1tr(comb(jj,:),:,:)));
end

UKData1 = zeros(noc,w2d2,(Resolx)*(Resoly));
for jj = 1:noc
    CIm = squeeze(UKData1ctr(jj,:,:));
    CDt = zeros(w2d2,(Resolx)*(Resoly));
    l = 1;
```

```
        for j = 1+rad2:Resoly+rad2
            for k = 1+rad2:Resolx+rad2
                CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                l = l + 1;
            end
        end
    UKData1(jj,:,:) = CDt;
end

for jj = 1:noc
    TMP = UKData1(comb(jj,:),:,:);
    for kk = 1:(noc-1)
        UKData1MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
    end
end

for jj = 1:noc
    TRData1f2ctr(jj,:,:) = squeeze(mean(TRData1f2tr(comb(jj,:),:,:)));
end

TRData1 = zeros(noc,w2d2,(Resolxt)*(Resolyt));
for jj = 1:noc
    CIm = squeeze(TRData1f2ctr(jj,:,:));
    CDt = zeros(w2d2,(Resolxt)*(Resolyt));
    CTg = zeros(1,(Resolxt)*(Resolyt));
    l = 1;
        for j = 1+rad2:Resolyt+rad2
            for k = 1+rad2:Resolxt+rad2
                CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                l = l + 1;
            end
        end
    TRData1(jj,:,:) = CDt;
end

for jj = 1:noc
    TMP = TRData1(comb(jj,:),:,:);
    for kk = 1:(noc-1)
        TRData1MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
    end
end
%%%%%%%%%%%Training data preperation complete.

display('Stage2 training begins...')
figure
subplot(3,3,1)
imagesc(ReConO);
axis square
colormap hot
title('Original');
subplot(3,3,2)
imagesc(Noisy);
axis square
colormap hot
```

```
title(['Noisy PSNR: ' mat2str(getPSNR(Noisy, ReConO, 1))]);
for jj = 1:noc
    net2 = feedforwardnet([10 10 10 10 10 10]);
    net2.trainParam.epochs = 30;
    net2 =
train(net2,[squeeze(TRData1MC(jj,:,1:L(2)))],[TGt0(1:L(2))],'useParallel','ye
s');

    NET2(jj).net = net2;
    Xmin2(jj,:) = min(squeeze(TRData1MC(jj,:,1:L(1)))');
    Xmax2(jj,:) = max(squeeze(TRData1MC(jj,:,1:L(1)))');
    Tmin2(jj,:) = min(TGt0(1:L(1)));
    Tmax2(jj,:) = max(TGt0(1:L(1)));

    TRData2f3(jj,:) = net2(squeeze(TRData1MC(jj,:,:)));
    TMP = reshape(TRData2f3(jj,:),[Resolxt,Resolyt])';
    RESULT2Tr(jj,:,:) = TMP;

    UKData2f3(jj,:) = net2(squeeze(UKData1MC(jj,:,:)));
    TMP = reshape(UKData2f3(jj,:),[Resolx,Resoly])';
    RESULT2(jj,:,:) = TMP;
    subplot(3,3,jj+2)
    imagesc(TMP);
    axis square
    colormap hot
    title(['N' mat2str(jj) ' PSNR: ' mat2str(round(getPSNR(TMP, ReConO,
1)*100)/100)]);
    PSNR1(jj) = getPSNR(TMP, ReConO, 1);
    [FSIM, FSIMc] = FeatureSIM(ReConO*255, TMP*255);
    FSIM1(jj) = FSIM;
    [mssim ssim_map] = ssim_index(ReConO*255, TMP*255);
    MSSIM1(jj) = mssim;
    title(['N' mat2str(jj) ' PSNR: ' mat2str(PSNR1(jj))]);
    pause(.01)
end
for jj = 1:noc
    TMP = reshape(TRData2f3(jj,:),[Resolxt,Resolyt])';
    TRData2f3tr(jj,:,:) = zeros(Resolyt+2*rad2,Resolxt+2*rad2);
    TRData2f3tr(jj,rad2+1:Resolyt+rad2,rad2+1:Resolxt+rad2) = TMP;

    TMP = reshape(UKData2f3(jj,:),[Resolx,Resoly])';
    UKData2tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
    UKData2tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
end
display('Stage2 training complete')
fprintf('Av. PSNR: %.4fdB, Av. FSIM: %.4fdB, Av. MSSIM: %.4fdB
\n',mean(PSNR1),mean(FSIM1),mean(MSSIM1));
%%%%%%%%%%Stage2 training complete
TRData1MC=0;
UKData1MC=0;

for jj = 1:noc
    UKData2ctr(jj,:,:) = squeeze(mean(UKData2tr(comb(jj,:),:,:)));
end

UKData2 = zeros(noc,w2d2,(Resolx)*(Resoly));
```

```
for jj = 1:noc
    CIm = squeeze(UKData2ctr(jj,:,:));
    CDt = zeros(w2d2,(Resolx)*(Resoly));
    l = 1;
        for j = 1+rad2:Resoly+rad2
            for k = 1+rad2:Resolx+rad2
                CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                l = l + 1;
            end
        end
    UKData2(jj,:,:) = CDt;
end

for jj = 1:noc
    TMP = UKData2(comb(jj,:),:,:);
    for kk = 1:(noc-1)
        UKData2MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
    end
end

for jj = 1:noc
    TRData2f3ctr(jj,:,:) = squeeze(mean(TRData2f3tr(comb(jj,:),:,:)));
end

TRData2 = zeros(noc,w2d2,(Resolxt)*(Resolyt));
for jj = 1:noc
    CIm = squeeze(TRData2f3ctr(jj,:,:));
    CDt = zeros(w2d2,(Resolxt)*(Resolyt));
    l = 1;
        for j = 1+rad2:Resolyt+rad2
            for k = 1+rad2:Resolxt+rad2
                CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                l = l + 1;
            end
        end
    TRData2(jj,:,:) = CDt;
end

for jj = 1:noc
    TMP = TRData2(comb(jj,:),:,:);
    for kk = 1:(noc-1)
        TRData2MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
    end
end
%%%%%%%%%%%Training data preperation complete.

display('Stage3 training begins...')
figure
subplot(3,3,1)
imagesc(ReConO);
axis square
colormap hot
title('Original');
subplot(3,3,2)
```

```
imagesc(Noisy);
axis square
colormap hot
title(['Noisy PSNR: ' mat2str(getPSNR(Noisy, ReConO, 1))]);
for jj = 1:noc
    net3 = feedforwardnet([10 10 10 10 10 10]);
    net3.trainParam.epochs = 20;
    net3 =
train(net3,[squeeze(TRData2MC(jj,:,1:L(3)))],[TGt0(1:L(3))],'useParallel','ye
s');

    NET3(jj).net = net3;
    Xmin3(jj,:) = min(squeeze(TRData2MC(jj,:,1:L(1)))');
    Xmax3(jj,:) = max(squeeze(TRData2MC(jj,:,1:L(1)))');
    Tmin3(jj,:) = min(TGt0(1:L(1)));
    Tmax3(jj,:) = max(TGt0(1:L(1)));

    TRData3f4(jj,:) = net3(squeeze(TRData2MC(jj,:,:)));
    TMP = reshape(TRData3f4(jj,:),[Resolxt,Resolyt])';
    RESULT3Tr(jj,:,:) = TMP;

    UKData3f4(jj,:) = net3(squeeze(UKData2MC(jj,:,:)));
    TMP = reshape(UKData3f4(jj,:),[Resolx,Resoly])';
    RESULT3(jj,:,:) = TMP;
    subplot(3,3,jj+2)
    imagesc(TMP);
    axis square
    colormap hot
    title(['N' mat2str(jj) ' PSNR: ' mat2str(round(getPSNR(TMP, ReConO,
1)*100)/100)]);
    PSNR1(jj) = getPSNR(TMP, ReConO, 1);
    [FSIM, FSIMc] = FeatureSIM(ReConO*255, TMP*255);
    FSIM1(jj) = FSIM;
    [mssim ssim_map] = ssim_index(ReConO*255, TMP*255);
    MSSIM1(jj) = mssim;
    title(['N' mat2str(jj) ' PSNR: ' mat2str(PSNR1(jj))]);
    pause(.01)
end
for jj = 1:noc
    TMP = reshape(TRData3f4(jj,:),[Resolxt,Resolyt])';
    TRData3f4tr(jj,:,:) = zeros(Resolyt+2*rad2,Resolxt+2*rad2);
    TRData3f4tr(jj,rad2+1:Resolyt+rad2,rad2+1:Resolxt+rad2) = TMP;

    TMP = reshape(UKData3f4(jj,:),[Resolx,Resoly])';
    UKData3tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
    UKData3tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
end
display('Stage3 training complete')
fprintf('Av. PSNR: %.4fdB, Av. FSIM: %.4fdB, Av. MSSIM: %.4fdB
\n',mean(PSNR1),mean(FSIM1),mean(MSSIM1));
%%%%%%%%%%Stage3 training complete
TRData2MC=0;
UKData2MC=0;
```

```
for jj = 1:noc
    UKData3ctr(jj,:,:) = squeeze(mean(UKData3tr(comb(jj,:),:,:)));
end

UKData3 = zeros(noc,w2d2,(Resolx)*(Resoly));
for jj = 1:noc
    CIm = squeeze(UKData3ctr(jj,:,:));
    CDt = zeros(w2d2,(Resolx)*(Resoly));
    l = 1;
        for j = 1+rad2:Resoly+rad2
            for k = 1+rad2:Resolx+rad2
                CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                l = l + 1;
            end
        end
    UKData3(jj,:,:) = CDt;
end

for jj = 1:noc
    TMP = UKData3(comb(jj,:),:,:);
    for kk = 1:(noc-1)
        UKData3MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
    end
end

for jj = 1:noc
    TRData3f4ctr(jj,:,:) = squeeze(mean(TRData3f4tr(comb(jj,:),:,:)));
end

TRData3 = zeros(noc,w2d2,(Resolxt)*(Resolyt));
for jj = 1:noc
    CIm = squeeze(TRData3f4ctr(jj,:,:));
    CDt = zeros(w2d2,(Resolxt)*(Resolyt));
    l = 1;
        for j = 1+rad2:Resolyt+rad2
            for k = 1+rad2:Resolxt+rad2
                CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                l = l + 1;
            end
        end
    TRData3(jj,:,:) = CDt;
end

for jj = 1:noc
    TMP = TRData3(comb(jj,:),:,:);
    for kk = 1:(noc-1)
        TRData3MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
    end
end
%%%%%%%%%%%Training data preperation complete.
```

```
display('Stage4 training begins...')
figure
subplot(3,3,1)
imagesc(ReConO);
axis square
colormap hot
title('Original');
subplot(3,3,2)
imagesc(Noisy);
axis square
colormap hot
title(['Noisy PSNR: ' mat2str(getPSNR(Noisy, ReConO, 1))]);
for jj = 1:noc
    net4 = feedforwardnet([10 10 10 10 10 10]);
    net4.trainParam.epochs = 20;
    net4 =
train(net4,[squeeze(TRData3MC(jj,:,1:L(4)))],[TGt0(1:L(4))],'useParallel','ye
s');

    NET4(jj).net = net4;
    Xmin4(jj,:) = min(squeeze(TRData3MC(jj,:,1:L(1)))');
    Xmax4(jj,:) = max(squeeze(TRData3MC(jj,:,1:L(1)))');
    Tmin4(jj,:) = min(TGt0(1:L(1)));
    Tmax4(jj,:) = max(TGt0(1:L(1)));

    TRData4f5(jj,:) = net4(squeeze(TRData3MC(jj,:,:)));
    TMP = reshape(TRData4f5(jj,:),[Resolxt,Resolyt])';
    RESULT4Tr(jj,:,:) = TMP;

    UKData4f5(jj,:) = net4(squeeze(UKData3MC(jj,:,:)));
    TMP = reshape(UKData4f5(jj,:),[Resolx,Resoly])';
    RESULT4(jj,:,:) = TMP;
    subplot(3,3,jj+2)
    imagesc(TMP);
    axis square
    colormap hot
    title(['N' mat2str(jj) ' PSNR: ' mat2str(round(getPSNR(TMP, ReConO,
1)*100)/100)]);
    PSNR1(jj) = getPSNR(TMP, ReConO, 1);
    [FSIM, FSIMc] = FeatureSIM(ReConO*255, TMP*255);
    FSIM1(jj) = FSIM;
    [mssim ssim_map] = ssim_index(ReConO*255, TMP*255);
    MSSIM1(jj) = mssim;
    title(['N' mat2str(jj) ' PSNR: ' mat2str(PSNR1(jj))]);
    pause(.01)
end
for jj = 1:noc
    TMP = reshape(TRData4f5(jj,:),[Resolxt,Resolyt])';
    TRData4f5tr(jj,:,:) = zeros(Resolyt+2*rad2,Resolxt+2*rad2);
    TRData4f5tr(jj,rad2+1:Resolyt+rad2,rad2+1:Resolxt+rad2) = TMP;

    TMP = reshape(UKData4f5(jj,:),[Resolx,Resoly])';
    UKData4tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
    UKData4tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
end
```

```
display('Stage4 training complete')
fprintf('Av. PSNR: %.4fdB, Av. FSIM: %.4fdB, Av. MSSIM: %.4fdB
\n',mean(PSNR1),mean(FSIM1),mean(MSSIM1));
%%%%%%%%%%Stage4 training complete

ReConN = squeeze(mean(RESULT4));

szx = size(ReConN,2)+2*wg;
szy = size(ReConN,1)+2*wg;
zfReConN = zeros(szy,szx);
zfReConN(wg+1:end-wg,wg+1:end-wg) = ReConN;
GIm = 0;
A = 1;
x0 = 0; y0 = 0;
sigma_x = 1;
sigma_y = 2;
[X, Y] = meshgrid(-wg:1:wg, -wg:1:wg);
for theta = 0:pi/100:pi
    a = cos(theta)^2/2/sigma_x^2 + sin(theta)^2/2/sigma_y^2;
    b = -sin(2*theta)/4/sigma_x^2 + sin(2*theta)/4/sigma_y^2 ;
    c = sin(theta)^2/2/sigma_x^2 + cos(theta)^2/2/sigma_y^2;
    Z = A*exp( - (a*(X-x0).^2 + 2*b*(X-x0).*(Y-y0) + c*(Y-y0).^2)) ;
end
for j = wg+1:szy-wg
for i = wg+1:szx-wg
tmp = zfReConN(j-wg:j+wg,i-wg:i+wg);
GIm(j-wg,i-wg) = mean(mean(tmp .* Z));
end
end
GMean = GIm;
GMean = max(max(ReConO)).*(GMean / max(max(GMean)));

t = toc/60

figure
subplot(2,2,1)
imagesc(rot90(ReConO));
axis square; axis off
colormap hot
title('Original');
subplot(2,2,2)
imagesc(rot90(Noisy));
axis square; axis off
colormap hot
PSNR = getPSNR(Noisy, ReConO, 1);
[FSIM, FSIMc] = FeatureSIM(ReConO*255, Noisy*255);
FSIM = FSIM;
[mssim ssim_map] = ssim_index(ReConO*255, Noisy*255);
MSSIM = mssim;
title(sprintf('1 Copy (PSNR %.3f dB, FSIM %.3f, MSSIM %.3f)', ...
    PSNR, FSIM, mssim));
subplot(2,2,3)
imagesc(rot90(Mean));
axis square; axis off
colormap hot
PSNR = getPSNR(Mean, ReConO, 1);
```

```
[FSIM, FSIMc] = FeatureSIM(ReConO*255, Mean*255);
FSIM = FSIM;
[mssim ssim_map] = ssim_index(ReConO*255, Mean*255);
MSSIM = mssim;
title(sprintf('Mean (PSNR %.3f dB, FSIM %.3f, MSSIM %.3f)', ...
    PSNR, FSIM, mssim));
subplot(2,2,4)
imagesc(rot90(ReConN));
axis square; axis off
colormap hot
PSNR = getPSNR(ReConN, ReConO, 1);
[FSIM, FSIMc] = FeatureSIM(ReConO*255, ReConN*255);
[mssim ssim_map] = ssim_index(ReConO*255, ReConN*255);
FSIM = FSIM;
MSSIM = mssim;
title(sprintf('MCMLP (PSNR %.3f dB, FSIM %.3f, MSSIM %.3f)', ...
    PSNR, FSIM, mssim));

PNN = getPSNR(ReConN, ReConO, 1);
PMean = getPSNR(Noisy, ReConO, 1);
PGMean = getPSNR(GMean, ReConO, 1);
[FSIM, FSIMc] = FeatureSIM(ReConO*255, ReConN*255);
FNN = FSIM;
[FSIM, FSIMc] = FeatureSIM(ReConO*255, Noisy*255);
FMean = FSIM;
[FSIM, FSIMc] = FeatureSIM(ReConO*255, GMean*255);
FGMean = FSIM;
[mssim ssim_map] = ssim_index(ReConO*255, ReConN*255);
MNN = mssim;
[mssim ssim_map] = ssim_index(ReConO*255, Noisy*255);
MMean = mssim;
[mssim, ssim_map] = ssim_index(ReConO*255, GMean*255);
MGMean = mssim;

fprintf('PSNR: Noisy: %.4fdB, NN: %.4fdB, GMean: %.4fdB
\n',PMean,PNN,PGMean);
fprintf('FSIM: Noisy: %.4fdB, NN: %.4fdB, GMean: %.4fdB
\n',FMean,FNN,FGMean);
fprintf('MSSIM: Noisy: %.4fdB, NN: %.4fdB, GMean: %.4fdB
\n',MMean,MNN,MGMean);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

B.3.2 MCMLP demo

```
warning off;
clc;
clf;
clear all;
close all

fprintf('%s \n', 'Welcome to MCMLP Denoising Demo')
fprintf('%s \n\n', 'Khalid Youssef')


[namein, pathin, filterindex] = uigetfile({  '*.Jpg','JPEG image (*.Jpg)'},
'Select one or several files (using +CTRL or +SHIFT)','MultiSelect', 'on');
if isequal(namein,0) | isequal(pathin,0)
    disp('User pressed cancel')
else
    disp(['Input file : ', fullfile(pathin, namein)])
    [pathstr, name_s, ext]=fileparts(fullfile(pathin, namein));

    noc = 7;
    rad = 2;
    rad2 = 2;
    w2d = (rad*2+1)^2;
    w2d2 = (rad2*2+1)^2;
    wg = 3;

    flag = 0;
    for jj = 1:noc
        str = [namein(1:end-5) mat2str(jj) '.JPG'];
        disp(['Noisy Copy ' mat2str(jj) ': ', fullfile(pathin, str)])
        [pathstr, name_s, ext]=fileparts(fullfile(pathin, str));

        TMP = imread(str);
        TMP = squeeze(mean(TMP,3))/255;

        DATA(jj,:,:) = TMP;
    end


    for jj = 1:noc
    TMP = squeeze(DATA(jj,:,:));
    Resolx = size(TMP,2);
    Resoly = size(TMP,1);

    Data2bR0(jj,:,:) = zeros(Resoly+2*rad+1,Resolx+2*rad+1);
    Data2bR0(jj,rad+1:Resoly+rad,rad+1:Resolx+rad) = TMP;
    end

    Noisy = squeeze((Data2bR0(1,rad+1:Resoly+rad,rad+1:Resolx+rad)));

    comb = nchoosek(1:noc,(noc-1));
    nocom = length(comb(:,1));
    for jj = 1:noc
        Data2bR(jj,:,:) = squeeze(mean(Data2bR0(comb(jj,:),:,:)));
    end
```

```
    %%%%%%%%%%Introducing noise to image complete.

    UKData = zeros(noc,w2d,(Resolx)*(Resoly));
    for jj = 1:noc
        CIm = squeeze(Data2bR(jj,:,:));
        CDt = zeros(w2d,(Resolx)*(Resoly));
        CTg = zeros(1,(Resolx)*(Resoly));
        l = 1;
            for j = 1+rad:Resoly+rad
                for k = 1+rad:Resolx+rad
                    CDt(:,l) = reshape(CIm(j-rad:j+rad,k-
rad:k+rad),[1,(2*rad+1)^2]);
                    l = l + 1;
                end
            end
        UKData(jj,:,:) = CDt;
    end
    Mean = reshape(squeeze(mean(UKData(:,ceil(end/2),:))),[Resoly,Resolx])';

    for jj = 1:noc
        TMP = UKData(comb(jj,:),:,:);
        for kk = 1:(noc-1)
            UKDataMC(jj,(kk-1)*w2d+1:kk*w2d,:) = squeeze(TMP(kk,:,:));
        end
    end
    %%%%%%%%%%Unknown sample data preperation complete.


%%%%%%%%%%%%%%%%%%%%%%%%%noise estimation%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
opt = [15, 17, 20, 22, 25, 27, 30, 32, 35, 37, 40];

mn = 1;
mx = length(opt);

pn = 1;

[D1,h1,h2] =
MCMLP_EstimateJPG(Data2bR0(:,1:128+2*rad+1,1:128+2*rad+1),opt(mn));
subplot(3,3,pn)
pn = pn + 1;
if (std(h1) - std(h2)) <= 0
    hist(h2,20)
    hold on
    hist(h1,20)
    h = findobj(gca,'Type','patch');
    set(h(1),'FaceColor',[0 .5 .2],'EdgeColor','w');
    set(h(2),'FaceColor',[1 0 .1],'EdgeColor','w');
else
    hist(h1,20)
    hold on
    hist(h2,20)
    h = findobj(gca,'Type','patch');
    set(h(1),'FaceColor',[0 .5 .2],'EdgeColor','w');
    set(h(2),'FaceColor',[1 0 .1],'EdgeColor','w');
end
```

```
title(mat2str(opt(mn)))
xlabel(mat2str(D1))

[D2,h1,h2] =
MCMLP_EstimateJPG(Data2bR0(:,1:128+2*rad+1,1:128+2*rad+1),opt(mx));
subplot(3,3,pn)
pn = pn + 1;
if (std(h1) - std(h2)) <= 0
    hist(h2,20)
    hold on
    hist(h1,20)
    h = findobj(gca,'Type','patch');
    set(h(1),'FaceColor',[0 .5 .2],'EdgeColor','w');
    set(h(2),'FaceColor',[1 0 .1],'EdgeColor','w');
else
    hist(h1,20)
    hold on
    hist(h2,20)
    h = findobj(gca,'Type','patch');
    set(h(1),'FaceColor',[0 .5 .2],'EdgeColor','w');
    set(h(2),'FaceColor',[1 0 .1],'EdgeColor','w');
end
title(mat2str(opt(mx)))
xlabel(mat2str(D2))
while mx ~= mn
if D1 < D2
    mx = mx - ceil((mx-mn)/2);
    if mx ~= mn
        [D2,h1,h2] =
MCMLP_EstimateJPG(Data2bR0(:,1:128+2*rad+1,1:128+2*rad+1),opt(mx));
        subplot(3,3,pn)
pn = pn + 1;
        if (std(h1) - std(h2)) <= 0
    hist(h2,20)
    hold on
    hist(h1,20)
    h = findobj(gca,'Type','patch');
    set(h(1),'FaceColor',[0 .5 .2],'EdgeColor','w');
    set(h(2),'FaceColor',[1 0 .1],'EdgeColor','w');
else
    hist(h1,20)
    hold on
    hist(h2,20)
    h = findobj(gca,'Type','patch');
    set(h(1),'FaceColor',[0 .5 .2],'EdgeColor','w');
    set(h(2),'FaceColor',[1 0 .1],'EdgeColor','w');
end
title(mat2str(opt(mx)))
xlabel(mat2str(D2))
    end
else
    mn = mn + ceil((mx-mn)/2);
    if mx ~= mn
        [D1,h1,h2] =
MCMLP_EstimateJPG(Data2bR0(:,1:128+2*rad+1,1:128+2*rad+1),opt(mn));
        subplot(3,3,pn)
pn = pn + 1;
```

```
        if (std(h1) - std(h2)) <= 0
    hist(h2,20)
    hold on
    hist(h1,20)
    h = findobj(gca,'Type','patch');
    set(h(1),'FaceColor',[0 .5 .2],'EdgeColor','w');
    set(h(2),'FaceColor',[1 0 .1],'EdgeColor','w');
else
    hist(h1,20)
    hold on
    hist(h2,20)
    h = findobj(gca,'Type','patch');
    set(h(1),'FaceColor',[0 .5 .2],'EdgeColor','w');
    set(h(2),'FaceColor',[1 0 .1],'EdgeColor','w');
end
title(mat2str(opt(mn)))
xlabel(mat2str(D1))
    end
end
end
pause(.1)

opt2 = MCMLP_EstimateJPG2(Data2bR0(:,1:128+2*rad+1,1:128+2*rad+1),opt(mx));
[val,ind] = min(abs(opt-opt2));
if (opt2 < 15 - 1.25) || (opt2 > 35 + 1.25)
    display('Noise is out of range')
else
    if (opt(mx) < 35) && (opt2 > opt(ind) + 1.25)
        ind = ind + 1;
    end


if mod(opt(ind),5) == 0
    str = [mat2str(opt(ind)/100)];
else
    str = [mat2str((opt(ind)+.5)/100)];
end
display(['Estimated sigma: ' str])
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if mod(opt(ind),5) == 0
    str = ['demo' mat2str(opt(ind)) 'JPG'];
else
    str = ['demo' mat2str(opt(ind)) 'hJPG'];
end
    load(str)

    display('Stage1 denoising begins...')
    for jj = 1:noc

        xmin = Xmin1(jj,:);
        xmax = Xmax1(jj,:);
        tmin = Tmin1(jj,:);
        tmax = Tmax1(jj,:);
```

```
        IW = cell2mat(S1IW(jj,1));
        IB = cell2mat(S1B(jj,1));
        LB1 = cell2mat(S1B(jj,2));
        LB2 = cell2mat(S1B(jj,3));
        LB3 = cell2mat(S1B(jj,4));
        LB4 = cell2mat(S1B(jj,5));
        LB5 = cell2mat(S1B(jj,6));
        LB6 = cell2mat(S1B(jj,7));
        LW1 = cell2mat(S1LW(jj,2,1));
        LW2 = cell2mat(S1LW(jj,3,2));
        LW3 = cell2mat(S1LW(jj,4,3));
        LW4 = cell2mat(S1LW(jj,5,4));
        LW5 = cell2mat(S1LW(jj,6,5));
        LW6 = cell2mat(S1LW(jj,7,6));

        tst = squeeze(UKDataMC(jj,:,:));
        UKData1f2(jj,:) = tmin + (tmax-
tmin)*((LW6*tansig(LW5*tansig(LW4*tansig(LW3*tansig(LW2*tansig(LW1*tansig...
            (IW*(-1+ 2*((tst-repmat(xmin',[1 size(tst,2)]))))./...
            (repmat(xmax',[1 size(tst,2)])- repmat(xmin',[1
size(tst,2)])))...
            +repmat(IB,[1 size(tst,2)]))+repmat(LB1,[1
size(tst,2)]))+repmat(LB2,[1 size(tst,2)]))+...
            repmat(LB3,[1 size(tst,2)]))+repmat(LB4,[1
size(tst,2)]))+repmat(LB5,[1 size(tst,2)]))+LB6) +1)/2;

        TMP = reshape(UKData1f2(jj,:),[Resolx,Resoly])';
        RESULT1(jj,:,:) = TMP;
    end

    for jj = 1:noc
        TMP = reshape(UKData1f2(jj,:),[Resolx,Resoly])';
        UKData1tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
        UKData1tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
    end
    display('Stage1 denoising complete')
    %%%%%%%%%%Stage1 denoising complete

    for jj = 1:noc
        UKData1ctr(jj,:,:) = squeeze(mean(UKData1tr(comb(jj,:),:,:)));
    end

    UKData1 = zeros(noc,w2d2,(Resolx)*(Resoly));
    for jj = 1:noc
        CIm = squeeze(UKData1ctr(jj,:,:));
        CDt = zeros(w2d2,(Resolx)*(Resoly));
        l = 1;
            for j = 1+rad2:Resoly+rad2
                for k = 1+rad2:Resolx+rad2
                    CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                    l = l + 1;
                end
            end
        UKData1(jj,:,:) = CDt;
    end
```

```
    for jj = 1:noc
        TMP = UKData1(comb(jj,:),:,:);
        for kk = 1:(noc-1)
            UKData1MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
        end
    end


    display('Stage2 denoising begins...')
    for jj = 1:noc

        xmin = Xmin2(jj,:);
        xmax = Xmax2(jj,:);
        tmin = Tmin2(jj,:);
        tmax = Tmax2(jj,:);


        IW = cell2mat(S2IW(jj,1));
        IB = cell2mat(S2B(jj,1));
        LB1 = cell2mat(S2B(jj,2));
        LB2 = cell2mat(S2B(jj,3));
        LB3 = cell2mat(S2B(jj,4));
        LB4 = cell2mat(S2B(jj,5));
        LB5 = cell2mat(S2B(jj,6));
        LB6 = cell2mat(S2B(jj,7));
        LW1 = cell2mat(S2LW(jj,2,1));
        LW2 = cell2mat(S2LW(jj,3,2));
        LW3 = cell2mat(S2LW(jj,4,3));
        LW4 = cell2mat(S2LW(jj,5,4));
        LW5 = cell2mat(S2LW(jj,6,5));
        LW6 = cell2mat(S2LW(jj,7,6));

        tst = squeeze(UKData1MC(jj,:,:));
        UKData2f3(jj,:) = tmin + (tmax-
tmin)*((LW6*tansig(LW5*tansig(LW4*tansig(LW3*tansig(LW2*tansig(LW1*tansig...
            (IW*(-1+ 2*((tst-repmat(xmin',[1 size(tst,2)]))))./...
            (repmat(xmax',[1 size(tst,2)])- repmat(xmin',[1
size(tst,2)])))...
            +repmat(IB,[1 size(tst,2)]))+repmat(LB1,[1
size(tst,2)]))+repmat(LB2,[1 size(tst,2)]))+...
            repmat(LB3,[1 size(tst,2)]))+repmat(LB4,[1
size(tst,2)]))+repmat(LB5,[1 size(tst,2)]))+LB6) +1)/2;

        TMP = reshape(UKData2f3(jj,:),[Resolx,Resoly])';
        RESULT2(jj,:,:) = TMP;
    end

    UKData1MC = 0;

    for jj = 1:noc
        TMP = reshape(UKData2f3(jj,:),[Resolx,Resoly])';
        UKData2tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
        UKData2tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
    end
```

```
    display('Stage2 denoising complete')
    %%%%%%%%%%Stage2 denoising complete


    for jj = 1:noc
        UKData2ctr(jj,:,:) = squeeze(mean(UKData2tr(comb(jj,:),:,:)));
    end

    UKData2 = zeros(noc,w2d2,(Resolx)*(Resoly));
    for jj = 1:noc
        CIm = squeeze(UKData2ctr(jj,:,:));
        CDt = zeros(w2d2,(Resolx)*(Resoly));
        l = 1;
            for j = 1+rad2:Resoly+rad2
                for k = 1+rad2:Resolx+rad2
                    CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                    l = l + 1;
                end
            end
        UKData2(jj,:,:) = CDt;
    end

    for jj = 1:noc
        TMP = UKData2(comb(jj,:),:,:);
        for kk = 1:(noc-1)
            UKData2MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
        end
    end

    display('Stage3 denoising begins...')
    for jj = 1:noc

    xmin = Xmin3(jj,:);
    xmax = Xmax3(jj,:);
    tmin = Tmin3(jj,:);
    tmax = Tmax3(jj,:);

    IW = cell2mat(S3IW(jj,1));
    IB = cell2mat(S3B(jj,1));
    LB1 = cell2mat(S3B(jj,2));
    LB2 = cell2mat(S3B(jj,3));
    LB3 = cell2mat(S3B(jj,4));
    LB4 = cell2mat(S3B(jj,5));
    LB5 = cell2mat(S3B(jj,6));
    LB6 = cell2mat(S3B(jj,7));
    LW1 = cell2mat(S3LW(jj,2,1));
    LW2 = cell2mat(S3LW(jj,3,2));
    LW3 = cell2mat(S3LW(jj,4,3));
    LW4 = cell2mat(S3LW(jj,5,4));
    LW5 = cell2mat(S3LW(jj,6,5));
    LW6 = cell2mat(S3LW(jj,7,6));

    tst = squeeze(UKData2MC(jj,:,:));
```

```
    UKData3f4(jj,:) = tmin + (tmax-
tmin)*((LW6*tansig(LW5*tansig(LW4*tansig(LW3*tansig(LW2*tansig(LW1*tansig...
            (IW*(-1+ 2*((tst-repmat(xmin',[1 size(tst,2)]))./...
            (repmat(xmax',[1 size(tst,2)])- repmat(xmin',[1
size(tst,2)])))...
            +repmat(IB,[1 size(tst,2)]))+repmat(LB1,[1
size(tst,2)]))+repmat(LB2,[1 size(tst,2)]))+...
            repmat(LB3,[1 size(tst,2)]))+repmat(LB4,[1
size(tst,2)]))+repmat(LB5,[1 size(tst,2)]))+LB6) +1)/2;

        TMP = reshape(UKData3f4(jj,:),[Resolx,Resoly])';
        RESULT3(jj,:,:) = TMP;
    end

    UKData2MC = 0;

    for jj = 1:noc
        TMP = reshape(UKData3f4(jj,:),[Resolx,Resoly])';
        UKData3tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
        UKData3tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
    end
    display('Stage3 denoising complete')
    %%%%%%%%%%Stage3 denoising complete


    for jj = 1:noc
        UKData3ctr(jj,:,:) = squeeze(mean(UKData3tr(comb(jj,:),:,:)));
    end

    UKData3 = zeros(noc,w2d2,(Resolx)*(Resoly));
    for jj = 1:noc
        CIm = squeeze(UKData3ctr(jj,:,:));
        CDt = zeros(w2d2,(Resolx)*(Resoly));
        l = 1;
            for j = 1+rad2:Resoly+rad2
                for k = 1+rad2:Resolx+rad2
                    CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                    l = l + 1;
                end
            end
        UKData3(jj,:,:) = CDt;
    end

    for jj = 1:noc
        TMP = UKData3(comb(jj,:),:,:);
        for kk = 1:(noc-1)
            UKData3MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
        end
    end

    display('Stage4 denoising begins...')
    for jj = 1:noc

        xmin = Xmin4(jj,:);
```

```
        xmax = Xmax4(jj,:);
        tmin = Tmin4(jj,:);
        tmax = Tmax4(jj,:);

        IW = cell2mat(S4IW(jj,1));
        IB = cell2mat(S4B(jj,1));
        LB1 = cell2mat(S4B(jj,2));
        LB2 = cell2mat(S4B(jj,3));
        LB3 = cell2mat(S4B(jj,4));
        LB4 = cell2mat(S4B(jj,5));
        LB5 = cell2mat(S4B(jj,6));
        LB6 = cell2mat(S4B(jj,7));
        LW1 = cell2mat(S4LW(jj,2,1));
        LW2 = cell2mat(S4LW(jj,3,2));
        LW3 = cell2mat(S4LW(jj,4,3));
        LW4 = cell2mat(S4LW(jj,5,4));
        LW5 = cell2mat(S4LW(jj,6,5));
        LW6 = cell2mat(S4LW(jj,7,6));

        tst = squeeze(UKData3MC(jj,:,:));
        UKData4f5(jj,:) = tmin + (tmax-
tmin)*((LW6*tansig(LW5*tansig(LW4*tansig(LW3*tansig(LW2*tansig(LW1*tansig...
            (IW*(-1+ 2*((tst-repmat(xmin',[1 size(tst,2)]))./...
            (repmat(xmax',[1 size(tst,2)])- repmat(xmin',[1
size(tst,2)])))...
            +repmat(IB,[1 size(tst,2)]))+repmat(LB1,[1
size(tst,2)]))+repmat(LB2,[1 size(tst,2)]))+...
            repmat(LB3,[1 size(tst,2)]))+repmat(LB4,[1
size(tst,2)]))+repmat(LB5,[1 size(tst,2)]))+LB6) +1)/2;

        TMP = reshape(UKData4f5(jj,:),[Resolx,Resoly])';
        RESULT4(jj,:,:) = TMP;
    end

    UKData3MC = 0;
    display('Stage4 denoising complete')
    %%%%%%%%%%Stage4 denoising complete


    ReConN = squeeze(mean(RESULT4));
    Denoised = ReConN;

    figure
    subplot(1,3,1)
    imagesc((Noisy));
    axis square; axis off
    colormap hot
    title('1 Copy','FontSize',18)

    subplot(1,3,2)
    imagesc((Denoised));
    axis square; axis off
    colormap hot
    title('NN','FontSize',18)
    pause(.1)
```

```
[namein, pathin, filterindex] = uigetfile({  '*.Jpg','JPEG image (*.Jpg)'},
'Select one or several files (using +CTRL or +SHIFT)','MultiSelect', 'on');
if isequal(namein,0) | isequal(pathin,0)
    disp('User pressed cancel')
else
    disp(['Input file : ', fullfile(pathin, namein)])
    [pathstr, name_s, ext]=fileparts(fullfile(pathin, namein));

    ReConI = imread(namein);
    ReConI = squeeze(mean(ReConI,3));
    ReConI = ReConI / max(max(ReConI));
    Original = ReConI;

    subplot(1,3,3)
    imagesc((ReConI));
    axis square; axis off
    colormap hot
    title('Original','FontSize',18);

    Diff = Denoised - Original;
    RMSE = sqrt(mean(Diff(:).^2));
    PSNR = 20*log10(1/RMSE)

    [FSIM, FSIMc] = FeatureSIM(Denoised*255,Original*255);
    display('Comp')
    FSIM
    [mssim ssim_map] = ssim_index(Denoised*255,Original*255);
    display('Comp')
    mssim
end
end
end
```

B.3.3 MCMLP noise estimation function

```
function [dist,h1,h2] = MCMLP_EstimateJPG(Data,sa)

    Try_sigma = sa/100
    if mod (sa,5) == 0
        str = ['demo' mat2str(sa) 'JPG'];
    else
        str = ['demo' mat2str(floor(sa)) 'hJPG'];
    end
    load(str)

    NL = 1;
    rad = 2;
    rad2 = 2;
    noc = 7;
    w2d = (rad*2+1)^2;
    w2d2 = (rad2*2+1)^2;
    w3d = 0;
    wg = 3;
    wg1 = 3;
    fthresh = 1;
    thresh = .025;

    Data2bR0 = Data;
    Resolx = 128;
    Resoly = 128;
    Noisy = squeeze((Data2bR0(1,rad+1:Resoly+rad,rad+1:Resolx+rad)));

    comb = nchoosek(1:noc,(noc-1));
    nocom = length(comb(:,1));
    for jj = 1:noc
        Data2bR(jj,:,:) = squeeze(mean(Data2bR0(comb(jj,:),:,:)));
    end

    %%%%%%%%%%Introducing noise to image complete.

    UKData = zeros(noc,w2d,(Resolx)*(Resoly));
    for jj = 1:noc
        CIm = squeeze(Data2bR(jj,:,:));
        CDt = zeros(w2d,(Resolx)*(Resoly));
        l = 1;
            for j = 1+rad:Resoly+rad
                for k = 1+rad:Resolx+rad
                    CDt(:,l) = reshape(CIm(j-rad:j+rad,k-
rad:k+rad),[1,(2*rad+1)^2]);
                    l = l + 1;
                end
            end
        UKData(jj,:,:) = CDt;
    end
    Mean = reshape(squeeze(mean(UKData(:,ceil(end/2),:))),[Resoly,Resolx])';

    for jj = 1:noc
        TMP = UKData(comb(jj,:),:,:);
        for kk = 1:(noc-1)
            UKDataMC(jj,(kk-1)*w2d+1:kk*w2d,:) = squeeze(TMP(kk,:,:));
```

```
        end
    end
    %%%%%%%%%%%%Unknown sample data preparation complete.

    display('.')
    for jj = 1:noc

        xmin = Xmin1(jj,:);
        xmax = Xmax1(jj,:);
        tmin = Tmin1(jj,:);
        tmax = Tmax1(jj,:);

        IW = cell2mat(S1IW(jj,1));
        IB = cell2mat(S1B(jj,1));
        LB1 = cell2mat(S1B(jj,2));
        LB2 = cell2mat(S1B(jj,3));
        LB3 = cell2mat(S1B(jj,4));
        LB4 = cell2mat(S1B(jj,5));
        LB5 = cell2mat(S1B(jj,6));
        LB6 = cell2mat(S1B(jj,7));
        LW1 = cell2mat(S1LW(jj,2,1));
        LW2 = cell2mat(S1LW(jj,3,2));
        LW3 = cell2mat(S1LW(jj,4,3));
        LW4 = cell2mat(S1LW(jj,5,4));
        LW5 = cell2mat(S1LW(jj,6,5));
        LW6 = cell2mat(S1LW(jj,7,6));

        tst = squeeze(UKDataMC(jj,:,:));
        UKData1f2(jj,:) = tmin + (tmax-
tmin)*((LW6*tansig(LW5*tansig(LW4*tansig(LW3*tansig(LW2*tansig(LW1*tansig...
            (IW*(-1+ 2*((tst-repmat(xmin',[1 size(tst,2)]))))./...
            (repmat(xmax',[1 size(tst,2)])- repmat(xmin',[1
size(tst,2)])))...
            +repmat(IB,[1 size(tst,2)]))+repmat(LB1,[1
size(tst,2)]))+repmat(LB2,[1 size(tst,2)]))+...
            repmat(LB3,[1 size(tst,2)]))+repmat(LB4,[1
size(tst,2)]))+repmat(LB5,[1 size(tst,2)]))+LB6) +1)/2;

        TMP = reshape(UKData1f2(jj,:),[Resolx,Resoly])';
        RESULT1(jj,:,:) = TMP;
    end

    for jj = 1:noc
        TMP = reshape(UKData1f2(jj,:),[Resolx,Resoly])';
        UKData1tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
        UKData1tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
    end
    %%%%%%%%%%Stage1 denoising complete


    for jj = 1:noc
        UKData1ctr(jj,:,:) = squeeze(mean(UKData1tr(comb(jj,:),:,:)));
    end

    UKData1 = zeros(noc,w2d2,(Resolx)*(Resoly));
```

```
    for jj = 1:noc
        CIm = squeeze(UKData1ctr(jj,:,:));
        CDt = zeros(w2d2,(Resolx)*(Resoly));
        l = 1;
            for j = 1+rad2:Resoly+rad2
                for k = 1+rad2:Resolx+rad2
                    CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                    l = l + 1;
                end
            end
        UKData1(jj,:,:) = CDt;
    end

    for jj = 1:noc
        TMP = UKData1(comb(jj,:),:,:);
        for kk = 1:(noc-1)
            UKData1MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
        end
    end


    display('.')
    for jj = 1:noc

        xmin = Xmin2(jj,:);
        xmax = Xmax2(jj,:);
        tmin = Tmin2(jj,:);
        tmax = Tmax2(jj,:);


        IW = cell2mat(S2IW(jj,1));
        IB = cell2mat(S2B(jj,1));
        LB1 = cell2mat(S2B(jj,2));
        LB2 = cell2mat(S2B(jj,3));
        LB3 = cell2mat(S2B(jj,4));
        LB4 = cell2mat(S2B(jj,5));
        LB5 = cell2mat(S2B(jj,6));
        LB6 = cell2mat(S2B(jj,7));
        LW1 = cell2mat(S2LW(jj,2,1));
        LW2 = cell2mat(S2LW(jj,3,2));
        LW3 = cell2mat(S2LW(jj,4,3));
        LW4 = cell2mat(S2LW(jj,5,4));
        LW5 = cell2mat(S2LW(jj,6,5));
        LW6 = cell2mat(S2LW(jj,7,6));

        tst = squeeze(UKData1MC(jj,:,:));
        UKData2f3(jj,:) = tmin + (tmax-
tmin)*((LW6*tansig(LW5*tansig(LW4*tansig(LW3*tansig(LW2*tansig(LW1*tansig...
            (IW*(-1+ 2*((tst-repmat(xmin',[1 size(tst,2)]))./...
            (repmat(xmax',[1 size(tst,2)])- repmat(xmin',[1
size(tst,2)])))...
            +repmat(IB,[1 size(tst,2)]))+repmat(LB1,[1
size(tst,2)]))+repmat(LB2,[1 size(tst,2)]))+...
```

```
            repmat(LB3,[1 size(tst,2)]))+repmat(LB4,[1
size(tst,2)]))+repmat(LB5,[1 size(tst,2)]))+LB6) +1)/2;

        TMP = reshape(UKData2f3(jj,:),[Resolx,Resoly])';
        RESULT2(jj,:,:) = TMP;
    end

    UKData1MC = 0;

    for jj = 1:noc
        TMP = reshape(UKData2f3(jj,:),[Resolx,Resoly])';
        UKData2tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
        UKData2tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
    end
    %%%%%%%%%%Stage2 denoising complete

    for jj = 1:noc
        UKData2ctr(jj,:,:) = squeeze(mean(UKData2tr(comb(jj,:),:,:)));
    end

    UKData2 = zeros(noc,w2d2,(Resolx)*(Resoly));
    for jj = 1:noc
        CIm = squeeze(UKData2ctr(jj,:,:));
        CDt = zeros(w2d2,(Resolx)*(Resoly));
        l = 1;
            for j = 1+rad2:Resoly+rad2
                for k = 1+rad2:Resolx+rad2
                    CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                    l = l + 1;
                end
            end
        UKData2(jj,:,:) = CDt;
    end

    for jj = 1:noc
        TMP = UKData2(comb(jj,:),:,:);
        for kk = 1:(noc-1)
            UKData2MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
        end
    end

    display('.')
    for jj = 1:noc

    xmin = Xmin3(jj,:);
    xmax = Xmax3(jj,:);
    tmin = Tmin3(jj,:);
    tmax = Tmax3(jj,:);

    IW = cell2mat(S3IW(jj,1));
    IB = cell2mat(S3B(jj,1));
    LB1 = cell2mat(S3B(jj,2));
    LB2 = cell2mat(S3B(jj,3));
    LB3 = cell2mat(S3B(jj,4));
```

```
    LB4 = cell2mat(S3B(jj,5));
    LB5 = cell2mat(S3B(jj,6));
    LB6 = cell2mat(S3B(jj,7));
    LW1 = cell2mat(S3LW(jj,2,1));
    LW2 = cell2mat(S3LW(jj,3,2));
    LW3 = cell2mat(S3LW(jj,4,3));
    LW4 = cell2mat(S3LW(jj,5,4));
    LW5 = cell2mat(S3LW(jj,6,5));
    LW6 = cell2mat(S3LW(jj,7,6));

    tst = squeeze(UKData2MC(jj,:,:));
    UKData3f4(jj,:) = tmin + (tmax-
tmin)*((LW6*tansig(LW5*tansig(LW4*tansig(LW3*tansig(LW2*tansig(LW1*tansig...
            (IW*(-1+ 2*((tst-repmat(xmin',[1 size(tst,2)]))))./...
            (repmat(xmax',[1 size(tst,2)])- repmat(xmin',[1
size(tst,2)])))...
            +repmat(IB,[1 size(tst,2)]))+repmat(LB1,[1
size(tst,2)]))+repmat(LB2,[1 size(tst,2)]))+...
            repmat(LB3,[1 size(tst,2)]))+repmat(LB4,[1
size(tst,2)]))+repmat(LB5,[1 size(tst,2)]))+LB6) +1)/2;

        TMP = reshape(UKData3f4(jj,:),[Resolx,Resoly])';
        RESULT3(jj,:,:) = TMP;
    end

    UKData2MC = 0;

    for jj = 1:noc
        TMP = reshape(UKData3f4(jj,:),[Resolx,Resoly])';
        UKData3tr(jj,:,:) = zeros(Resoly+2*rad2,Resolx+2*rad2);
        UKData3tr(jj,rad2+1:Resoly+rad2,rad2+1:Resolx+rad2) = TMP;
    end
    %%%%%%%%%%Stage3 denoising complete

    for jj = 1:noc
        UKData3ctr(jj,:,:) = squeeze(mean(UKData3tr(comb(jj,:),:,:)));
    end

    UKData3 = zeros(noc,w2d2,(Resolx)*(Resoly));
    for jj = 1:noc
        CIm = squeeze(UKData3ctr(jj,:,:));
        CDt = zeros(w2d2,(Resolx)*(Resoly));
        l = 1;
            for j = 1+rad2:Resoly+rad2
                for k = 1+rad2:Resolx+rad2
                    CDt(:,l) = reshape(CIm(j-rad2:j+rad2,k-
rad2:k+rad2),[1,(2*rad2+1)^2]);
                    l = l + 1;
                end
            end
        UKData3(jj,:,:) = CDt;
    end

    for jj = 1:noc
        TMP = UKData3(comb(jj,:),:,:);
```

```
        for kk = 1:(noc-1)
            UKData3MC(jj,(kk-1)*w2d2+1:kk*w2d2,:) = squeeze(TMP(kk,:,:));
        end
    end

    display('.')
    for jj = 1:noc

        xmin = Xmin4(jj,:);
        xmax = Xmax4(jj,:);
        tmin = Tmin4(jj,:);
        tmax = Tmax4(jj,:);

        IW = cell2mat(S4IW(jj,1));
        IB = cell2mat(S4B(jj,1));
        LB1 = cell2mat(S4B(jj,2));
        LB2 = cell2mat(S4B(jj,3));
        LB3 = cell2mat(S4B(jj,4));
        LB4 = cell2mat(S4B(jj,5));
        LB5 = cell2mat(S4B(jj,6));
        LB6 = cell2mat(S4B(jj,7));
        LW1 = cell2mat(S4LW(jj,2,1));
        LW2 = cell2mat(S4LW(jj,3,2));
        LW3 = cell2mat(S4LW(jj,4,3));
        LW4 = cell2mat(S4LW(jj,5,4));
        LW5 = cell2mat(S4LW(jj,6,5));
        LW6 = cell2mat(S4LW(jj,7,6));

        tst = squeeze(UKData3MC(jj,:,:));
        UKData4f5(jj,:) = tmin + (tmax-
tmin)*((LW6*tansig(LW5*tansig(LW4*tansig(LW3*tansig(LW2*tansig(LW1*tansig...
            (IW*(-1+ 2*((tst-repmat(xmin',[1 size(tst,2)]))))./...
            (repmat(xmax',[1 size(tst,2)])- repmat(xmin',[1
size(tst,2)])))...
            +repmat(IB,[1 size(tst,2)]))+repmat(LB1,[1
size(tst,2)]))+repmat(LB2,[1 size(tst,2)]))+...
            repmat(LB3,[1 size(tst,2)]))+repmat(LB4,[1
size(tst,2)]))+repmat(LB5,[1 size(tst,2)]))+LB6) +1)/2;

        TMP = reshape(UKData4f5(jj,:),[Resolx,Resoly])';
        RESULT4(jj,:,:) = TMP;
    end

    UKData3MC = 0;
    %%%%%%%%%%Stage4 denoising complete

    ReConN = squeeze(mean(RESULT4));
    Denoised = ReConN;

    esigma = sa/100;
    S = ReConN(:);
    kk = 1;
    for jj = 1:length(S)
    [val,ind] = min(S);
    if kk < .05*length(S);
```

```
        In(kk) = ind;
        kk = kk + 1;
    end
    S(ind) = inf;
    end
    N = Noisy(:);
    M = N(In);
    Es = sqrt((randn(length(M),1)*esigma).^2+(randn(length(M),1)*esigma).^2);
    dist = abs(std(M) - std(Es));
    distance = dist

    h1 = M;
    h2 = Es;
end
```

B.4 10 Channel Bioreactor Control

This program requires COMSOL server for MATLAB (COMSOL Inc., Burlington, MA). Version 4.2 was used in this work.

```
clear
close all
scale = 0;

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelNode.create('mod1');

model.geom.create('geom1', 2);

model.mesh.create('mesh1', 'geom1');

model.physics.create('spf', 'LaminarFlow', 'geom1');

model.study.create('std1');
model.study('std1').feature.create('stat', 'Stationary');

model.geom('geom1').lengthUnit('mm');
model.geom('geom1').feature.create('imp1', 'Import');
model.geom('geom1').feature('imp1').set('filename',
'C:\Users\arj\Documents\MATLAB\BioReact\geom5.mphbin');
model.geom('geom1').feature('imp1').importData;
model.geom('geom1').run;

model.material.create('mat1');
model.material.remove('mat1');
model.material.create('mat1');
model.material('mat1').name('Water, liquid');
model.material('mat1').set('family', 'water');
model.material('mat1').propertyGroup('def').set('dynamicviscosity',
'eta(T[1/K])[Pa*s]');
model.material('mat1').propertyGroup('def').set('ratioofspecificheat',
'1.0');
model.material('mat1').propertyGroup('def').set('electricconductivity',
'5.5e-6[S/m]');
model.material('mat1').propertyGroup('def').set('heatcapacity',
'Cp(T[1/K])[J/(kg*K)]');
model.material('mat1').propertyGroup('def').set('density',
'rho(T[1/K])[kg/m^3]');
model.material('mat1').propertyGroup('def').set('thermalconductivity',
'k(T[1/K])[W/(m*K)]');
model.material('mat1').propertyGroup('def').set('soundspeed',
'cs(T[1/K])[m/s]');
model.material('mat1').propertyGroup('def').func.create('eta', 'Piecewise');
model.material('mat1').propertyGroup('def').func('eta').set('funcname',
'eta');
model.material('mat1').propertyGroup('def').func('eta').set('arg', 'T');
model.material('mat1').propertyGroup('def').func('eta').set('extrap',
'constant');
model.material('mat1').propertyGroup('def').func('eta').set('pieces',
{'273.15' '413.15' '1.3799566804-0.021224019151*T^1+1.3604562827E-4*T^2-
4.6454090319E-7*T^3+8.9042735735E-10*T^4-9.0790692686E-13*T^5+3.8457331488E-
```

```
16*T^6'; '413.15' '553.75' '0.00401235783-2.10746715E-5*T^1+3.85772275E-
8*T^2-2.39730284E-11*T^3'});
model.material('mat1').propertyGroup('def').func.create('Cp', 'Piecewise');
model.material('mat1').propertyGroup('def').func('Cp').set('funcname', 'Cp');
model.material('mat1').propertyGroup('def').func('Cp').set('arg', 'T');
model.material('mat1').propertyGroup('def').func('Cp').set('extrap',
'constant');
model.material('mat1').propertyGroup('def').func('Cp').set('pieces',
{'273.15' '553.75' '12010.1471-80.4072879*T^1+0.309866854*T^2-5.38186884E-
4*T^3+3.62536437E-7*T^4'});
model.material('mat1').propertyGroup('def').func.create('rho', 'Piecewise');
model.material('mat1').propertyGroup('def').func('rho').set('funcname',
'rho');
model.material('mat1').propertyGroup('def').func('rho').set('arg', 'T');
model.material('mat1').propertyGroup('def').func('rho').set('extrap',
'constant');
model.material('mat1').propertyGroup('def').func('rho').set('pieces',
{'273.15' '553.75' '838.466135+1.40050603*T^1-0.0030112376*T^2+3.71822313E-
7*T^3'});
model.material('mat1').propertyGroup('def').func.create('k', 'Piecewise');
model.material('mat1').propertyGroup('def').func('k').set('funcname', 'k');
model.material('mat1').propertyGroup('def').func('k').set('arg', 'T');
model.material('mat1').propertyGroup('def').func('k').set('extrap',
'constant');
model.material('mat1').propertyGroup('def').func('k').set('pieces', {'273.15'
'553.75' '-0.869083936+0.00894880345*T^1-1.58366345E-5*T^2+7.97543259E-
9*T^3'});
model.material('mat1').propertyGroup('def').func.create('cs',
'Interpolation');
model.material('mat1').propertyGroup('def').func('cs').set('funcname', 'cs');
model.material('mat1').propertyGroup('def').func('cs').set('interp',
'piecewisecubic');
model.material('mat1').propertyGroup('def').func('cs').set('extrap',
'const');
model.material('mat1').propertyGroup('def').func('cs').set('table', {'273'
'1403'; '278' '1427'; '283' '1447'; '293' '1481'; '303' '1507'; '313' '1526';
'323' '1541'; '333' '1552'; '343' '1555'; '353' '1555'; '363' '1550'; '373'
'1543'});
model.material('mat1').propertyGroup('def').addInput('temperature');
model.material('mat1').set('family', 'water');

model.physics('spf').feature.create('out1', 'Outlet', 1);
model.physics('spf').feature('out1').selection.set([24]);
model.physics('spf').feature.create('inl1', 'Inlet', 1);
model.physics('spf').feature('inl1').selection.set([5]);
model.physics('spf').feature.create('inl2', 'Inlet', 1);
model.physics('spf').feature('inl2').selection.set([3]);
model.physics('spf').feature.create('inl3', 'Inlet', 1);
model.physics('spf').feature('inl3').selection.set([1]);
model.physics('spf').feature.create('inl4', 'Inlet', 1);
model.physics('spf').feature('inl4').selection.set([18]);
model.physics('spf').feature.create('inl5', 'Inlet', 1);
model.physics('spf').feature('inl5').selection.set([22]);
model.physics('spf').feature.create('inl6', 'Inlet', 1);
model.physics('spf').feature('inl6').selection.set([28]);
model.physics('spf').feature.create('inl7', 'Inlet', 1);
model.physics('spf').feature('inl7').selection.set([35]);
```

```
model.physics('spf').feature.create('inl8', 'Inlet', 1);
model.physics('spf').feature('inl8').selection.set([49]);
model.physics('spf').feature.create('inl9', 'Inlet', 1);
model.physics('spf').feature('inl9').selection.set([50]);
model.physics('spf').feature.create('inl10', 'Inlet', 1);
model.physics('spf').feature('inl10').selection.set([51]);
model.physics('spf').feature.create('vf1', 'VolumeForce', 2);
model.physics('spf').feature('vf1').selection.set([1]);
model.physics('spf').feature('vf1').set('F', {'0' '-9810' '0'});
model.physics('spf').feature('inl1').set('U0in', 1, '0.1');

model.mesh('mesh1').autoMeshSize(3);
model.mesh('mesh1').run;

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').feature.create('st1', 'StudyStep');
model.sol('sol1').feature('st1').set('study', 'std1');
model.sol('sol1').feature('st1').set('studystep', 'stat');
model.sol('sol1').feature.create('v1', 'Variables');
model.sol('sol1').feature.create('s1', 'Stationary');
model.sol('sol1').feature('s1').feature.create('fc1', 'FullyCoupled');
model.sol('sol1').feature('s1').feature('fc1').set('initstep', 0.01);
model.sol('sol1').feature('s1').feature('fc1').set('minstep', 1.0E-6);
model.sol('sol1').feature('s1').feature('fc1').set('dtech', 'auto');
model.sol('sol1').feature('s1').feature('fc1').set('maxiter', 25);
model.sol('sol1').feature('s1').feature.create('d1', 'Direct');
model.sol('sol1').feature('s1').feature('d1').set('linsolver', 'pardiso');
model.sol('sol1').feature('s1').feature('fc1').set('linsolver', 'd1');
model.sol('sol1').feature('s1').feature('fc1').set('initstep', 0.01);
model.sol('sol1').feature('s1').feature('fc1').set('minstep', 1.0E-6);
model.sol('sol1').feature('s1').feature('fc1').set('dtech', 'auto');
model.sol('sol1').feature('s1').feature('fc1').set('maxiter', 25);
model.sol('sol1').feature('s1').feature.remove('fcDef');
model.sol('sol1').attach('std1');

model.result.create('pg1', 2);
model.result('pg1').set('data', 'dset1');
model.result('pg1').feature.create('surf1', 'Surface');
model.result('pg1').feature('surf1').set('expr', {'spf.U'});
model.result('pg1').set('frametype', 'spatial');
model.result('pg1').name('Velocity (spf)');
model.result.create('pg2', 2);
model.result('pg2').set('data', 'dset1');
model.result('pg2').feature.create('con', 'Contour');
model.result('pg2').feature('con').set('expr', {'p'});
model.result('pg2').set('frametype', 'spatial');
model.result('pg2').name('Pressure (spf)');
model.result('pg2').feature('con').set('number', 40);
model.sol('sol1').runAll;
model.result('pg1').run;

noin = 9;%%%%%% inpt vector length
noout = 10;%%%%%% number of outputs
nohn = 10;%%%%%% number of neurons in hidden layer
nohn2 = 10;
```

```
Whl_i = rand(nohn,noin)/100;
Bhl = rand(nohn,1)/100;
Whl_2 = rand(nohn2,nohn)/100;
Bhl2 = rand(nohn2,1)/100;
Wo_hl2 = rand(noout,nohn2)/100;
Bo = rand(noout,1)/100;
mu = .01;
h1 = ones(nohn,1)*mu;
h2 = ones(nohn2,1)*mu;
h3 = ones(noout,1)*mu;
mo = 0.025;
a1 = ones(nohn,1)*mo;
a2 = ones(nohn2,1)*mo;
a3 = ones(noout,1)*mo;
u = .1;
v = .2;
ds = 1e-1;
dso = ds;
maxms = 5e-2;
minms = 1e-6;
bcount = 0;
mfd = 3;

target = 10*ones(3,3);
inspeed = rand(1,10);
Err =  0;
Errs = zeros(9,1);
errn = zeros(1,10);
errb = inf;
outptb = zeros(10,1);
inpt = ones(noin,1)/100;
inptP = inpt;
fl = ones(3,3);
flb = fl;
flag = 0;
vv = 1;

x0 = -.5:.05:3;
y0 = -.5:.05:3;
[x,y] = meshgrid(x0,y0);
xx = [x(:),y(:)];

ii = 1;
ds = 5e-1;
dso = ds;
iii = ii;

figure
imagesc(imresize(target,[100,100]))
figure

for ii = iii:iii+49
    if ii > 10
        if mod(ii,2) == 0
        scale = scale + .001;
        end
```

```
    else
        scale = 0;
    end
    outpt  = purelin(Bo + Wo_hl2*logsig(Bhl2+ Whl_2*logsig(Bhl +
Whl_i*inpt)));
    hl = logsig(Bhl + Whl_i*inpt);
    hl2 = logsig(Bhl2+ Whl_2*logsig(Bhl + Whl_i*inpt));
    inspeed = min(max(abs(outpt),0),100);
    inspeedP = inspeed;

    model.physics('spf').feature('inl1').set('U0in', 1, [mat2str(inspeed(1))
'[mm/s]']);
    model.physics('spf').feature('inl2').set('U0in', 1, [mat2str(inspeed(2))
'[mm/s]']);
    model.physics('spf').feature('inl3').set('U0in', 1, [mat2str(inspeed(3))
'[mm/s]']);
    model.physics('spf').feature('inl4').set('U0in', 1, [mat2str(inspeed(4))
'[mm/s]']);
    model.physics('spf').feature('inl5').set('U0in', 1, [mat2str(inspeed(5))
'[mm/s]']);
    model.physics('spf').feature('inl6').set('U0in', 1, [mat2str(inspeed(6))
'[mm/s]']);
    model.physics('spf').feature('inl7').set('U0in', 1, [mat2str(inspeed(7))
'[mm/s]']);
    model.physics('spf').feature('inl8').set('U0in', 1, [mat2str(inspeed(8))
'[mm/s]']);
    model.physics('spf').feature('inl9').set('U0in', 1, [mat2str(inspeed(9))
'[mm/s]']);
    model.physics('spf').feature('inl10').set('U0in', 1,
[mat2str(inspeed(10)) '[mm/s]']);
    model.sol('sol1').runAll;
    model.result('pg1').run;
    shear = mphinterp(model,'spf.sr','coord',xx');
    shear = reshape(shear,length(x0),length(y0));
    for i1 = 1:71
    for j1 = 1:71
    if isnan(shear(i1,j1)) == 1
    shear(i1,j1) = 0;
    end
    end
    end
    smapsim = shear(end-60:end-41,11:31);;
    if ii == 1
        Vid1(:,:,vv) = imresize(imresize(abs(smapsim),[3,3]),[100,100]);
        vv = vv + 1;
    end

    fl = imresize(abs(smapsim),[3,3]);
    flP = fl;
    Inspeed = inspeed'

    errs = (target - target.*exp(-(fl-target).^2./(2*mfd^2)))./target;
    eerrs = reshape(errs,[1,9])
    err = sum(reshape(errs,[1,9]).^2)^.5;
    if err == 0
        break
```

```
    end
    Errs(:,ii) = eerrs';
    Err(ii) = err;

    dst = ds;
    if flag == 0
        order = randperm(10);
        for jj = 1:10
            refresh

            outptt = outpt;
            outptt(order(jj)) =
outptt(order(jj))+dst*sign(outptt(order(jj)));
            inspeed = inspeedP;
            inspeed(order(jj)) = min(max(abs(outptt(order(jj))),0),100);
            model.physics('spf').feature(['inl'
mat2str(order(jj))]).set('U0in', 1, [mat2str(inspeed(order(jj))) '[mm/s]']);
            model.sol('sol1').runAll;
            model.result('pg1').run;
            shear = mphinterp(model,'spf.sr','coord',xx');
            shear = reshape(shear,length(x0),length(y0));
            for i1 = 1:71
            for j1 = 1:71
            if isnan(shear(i1,j1)) == 1
            shear(i1,j1) = 0;
            end
            end
            end

            smapsim = shear(end-60:end-41,11:31);
            flt = flP;
            flt = imresize(abs(smapsim),[3,3]);
            errst = (target - target.*exp(-(flt-
target).^2./(2*mfd^2)))./target;

            errt = sum(reshape(errst,[1,9]).^2)^.5;
            if errt > errb && bcount < 3
                errt = errb;
                outptt = outptb;
                flt = fltb;
                bcount = bcount + 1;
            else
                errb = errt;
                outptb = outptt;
                fltb = flt;
                bcount = 0;
            end
            if errt < err
                Vid1(:,:,vv) =
imresize(imresize(abs(smapsim),[3,3]),[100,100]);
                vv = vv + 1;
                errn(order(jj)) = outptt(order(jj)) - outpt(order(jj));
                fl = flt;

                %**************************************Update parameters
                hlerror2 = Wo_hl2'*errn';
```

```
                hlerror = Whl_2'*hlerror2;
                if ii > 1
                    e1 = (hlerror - hlerrorP)./(hlerror+(hlerror==0)*1);
                    e2 = (hlerror2 - hlerror2P)./(hlerror2+(hlerror2==0)*1);
                    e3 = (errn' - errnP')./(errn'+(errn'==0)*1);

                    h1 = max(min(h1.*(1-sign(e1)*u.*exp(-e1)),maxms),minms);
                    h2 = max(min(h2.*(1-sign(e2)*u.*exp(-e2)),maxms),minms);
                    h3 = max(min(h3.*(1-sign(e3)*u.*exp(-e3)),maxms),minms);

                    a1 = max(min(a1.*(1-sign(e1)*v.*exp(-e1)),maxms),minms);
                    a2 = max(min(a2.*(1-sign(e2)*v.*exp(-e2)),maxms),minms);
                    a3 = max(min(a3.*(1-sign(e3)*v.*exp(-e3)),maxms),minms);
                end
                dWhl_i =  repmat(h1,[1,noin]).*((hlerror.*(hl.*(1-
hl)))*inpt');
                dBhl = h1.*(hlerror.*(hl.*(1-hl)));
                dWhl_2 =  repmat(h2,[1,nohn]).*((hlerror2.*(hl2.*(1-
hl2)))*hl');
                dBhl2 = h2.*(hlerror2.*(hl2.*(1-hl2)));
                dWo_hl2 =  repmat(h3,[1,nohn2]).*(errn'*hl2');
                dBo = h3.*errn';

                Whl_i = Whl_i + dWhl_i;
                Bhl = Bhl + dBhl;
                Whl_2 = Whl_2 + dWhl_2;
                Bhl2 = Bhl2 + dBhl2;
                Wo_hl2 = Wo_hl2 + dWo_hl2;
                Bo = Bo + dBo;

                if ii > 1%%%%%%%%%%%%%%%%%%%%%Add momentum
                Whl_i = Whl_i + repmat(a1,[1,noin]).*dWhl_ip;
                Bhl = Bhl + a1.*dBhlp;
                Whl_2 = Whl_2 + repmat(a2,[1,nohn]).*dWhl_2p;
                Bhl2 = Bhl2 + a2.*dBhl2p;
                Wo_hl2 = Wo_hl2 + repmat(a3,[1,nohn2]).*dWo_hlp;
                Bo = Bo + a3.*dBop;
                end

                dWhl_ip = dWhl_i;
                dBhlp = dBhl;
                dWhl_2p = dWhl_2;
                dBhl2p = dBhl2;
                dWo_hlp = dWo_hl2;
                dBop = dBo;
                %**************************************

                subplot(2,2,3)
                imagesc(imresize(flt,[100,100]));
                title(mat2str(order(jj)))
                caxis([0,1.5*max(max(target))])
                colorbar
                axis square; axis off
                pause(.01)
            else
                refresh
```

```
                outptt = outpt;
                outptt(order(jj)) = outptt(order(jj))-
dst*sign(outptt(order(jj)));
                inspeed = inspeedP;
                inspeed(order(jj)) = min(max(abs(outptt(order(jj))),0),100);
                model.physics('spf').feature(['inl'
mat2str(order(jj))]).set('U0in', 1, [mat2str(inspeed(order(jj))) '[mm/s]']);
                model.sol('sol1').runAll;
                model.result('pg1').run;
                shear = mphinterp(model,'spf.sr','coord',xx');
                shear = reshape(shear,length(x0),length(y0));
                for i1 = 1:71
                for j1 = 1:71
                if isnan(shear(i1,j1)) == 1
                shear(i1,j1) = 0;
                end
                end
                end

                smapsim = shear(end-60:end-41,11:31);;
                flt = flP;
                flt = imresize(abs(smapsim),[3,3]);
                errst = (target - target.*exp(-(flt-
target).^2./(2*mfd^2)))./target;
                errt = sum(reshape(errst,[1,9]).^2)^.5;
                if errt > errb && bcount < 3
                    errt = errb;
                    outptt = outptb;
                    flt = fltb;
                    bcount = bcount + 1;
                else
                    errb = errt;
                    outptb = outptt;
                    fltb = flt;
                    bcount = 0;
                end
                if errt < err
                    Vid1(:,:,vv) =
imresize(imresize(abs(smapsim),[3,3]),[100,100]);
                    vv = vv + 1;
                    errn(order(jj)) = outptt(order(jj)) - outpt(order(jj));
                    fl = flt;

                   %************************************Update parameters
                    hlerror2 = Wo_hl2'*errn';
                    hlerror = Whl_2'*hlerror2;
                    if ii > 1
                        e1 = (hlerror - hlerrorP)./(hlerror+(hlerror==0)*1);
                        e2 = (hlerror2 -
hlerror2P)./(hlerror2+(hlerror2==0)*1);
                        e3 = (errn' - errnP')./(errn'+(errn'==0)*1);

                        h1 = max(min(h1.*(1-sign(e1)*u.*exp(-
e1)),maxms),minms);
                        h2 = max(min(h2.*(1-sign(e2)*u.*exp(-
e2)),maxms),minms);
```

```
                        h3 = max(min(h3.*(1-sign(e3)*u.*exp(-
e3)),maxms),minms);

                        a1 = max(min(a1.*(1-sign(e1)*v.*exp(-
e1)),maxms),minms);
                        a2 = max(min(a2.*(1-sign(e2)*v.*exp(-
e2)),maxms),minms);
                        a3 = max(min(a3.*(1-sign(e3)*v.*exp(-
e3)),maxms),minms);
                    end
                    dWhl_i =  repmat(h1,[1,noin]).*((hlerror.*(hl.*(1-
hl)))*inpt');
                    dBhl = h1.*(hlerror.*(hl.*(1-hl)));
                    dWhl_2 =  repmat(h2,[1,nohn]).*((hlerror2.*(hl2.*(1-
hl2)))*hl');
                    dBhl2 = h2.*(hlerror2.*(hl2.*(1-hl2)));
                    dWo_hl2 =  repmat(h3,[1,nohn2]).*(errn'*hl2');
                    dBo = h3.*errn';

                    Whl_i = Whl_i + dWhl_i;
                    Bhl = Bhl + dBhl;
                    Whl_2 = Whl_2 + dWhl_2;
                    Bhl2 = Bhl2 + dBhl2;
                    Wo_hl2 = Wo_hl2 + dWo_hl2;
                    Bo = Bo + dBo;

                    if ii > 1%%%%%%%%%%%%%%%%%%%%%Add momentum
                    Whl_i = Whl_i + repmat(a1,[1,noin]).*dWhl_ip;
                    Bhl = Bhl + a1.*dBhlp;
                    Whl_2 = Whl_2 + repmat(a2,[1,nohn]).*dWhl_2p;
                    Bhl2 = Bhl2 + a2.*dBhl2p;
                    Wo_hl2 = Wo_hl2 + repmat(a3,[1,nohn2]).*dWo_hlp;
                    Bo = Bo + a3.*dBop;
                    end

                    dWhl_ip = dWhl_i;
                    dBhlp = dBhl;
                    dWhl_2p = dWhl_2;
                    dBhl2p = dBhl2;
                    dWo_hlp = dWo_hl2;
                    dBop = dBo;
                    %**************************************

                    subplot(2,2,3)
                    imagesc(imresize(flt,[100,100]));
                    title(mat2str(order(jj)))
                    caxis([0,1.5*max(max(target))])
                    colorbar
                    axis square; axis off
                    pause(.01)
                else
                    refresh
                    errn(order(jj)) = 0;
                end
            end
        end
```

```
    else
        outptt = outpt;
        outptt = outptt+dst*(rand(10,1)-.5);
        inspeed = inspeedP;
        inspeed = min(max(abs(outptt),0),100);
        for kk = 1:10
            model.physics('spf').feature(['inl' mat2str(kk)]).set('U0in', 1,
[mat2str(inspeed(kk)) '[mm/s]']);
        end
        model.sol('sol1').runAll;
        model.result('pg1').run;
        shear = mphinterp(model,'spf.sr','coord',xx');
        shear = reshape(shear,length(x0),length(y0));
        for i1 = 1:71
        for j1 = 1:71
        if isnan(shear(i1,j1)) == 1
        shear(i1,j1) = 0;
        end
        end
        end

        smapsim = shear(end-60:end-41,11:31);;
        flt = flP;
        flt = imresize(abs(smapsim),[3,3]);
        errst = (target - target.*exp(-(flt-target).^2./(2*mfd^2)))./target;
        errt = sum(reshape(errst,[1,9]).^2)^.5;
        if errt > errb && bcount < 3
            errt = errb;
            outptt = outptb;
            flt = fltb;
            bcount = bcount + 1;
        else
            errb = errt;
            outptb = outptt;
            fltb = flt;
            bcount = 0;
        end
        if errt < err
            Vid1(:,:,vv) = imresize(imresize(abs(smapsim),[3,3]),[100,100]);
            vv = vv + 1;
            errn = outptt' - outpt';
            fl = flt;

            %**************************************Update parameters
            hlerror2 = Wo_hl2'*errn';
            hlerror = Whl_2'*hlerror2;
            if ii > 1
                e1 = (hlerror - hlerrorP)./(hlerror+(hlerror==0)*1);
                e2 = (hlerror2 - hlerror2P)./(hlerror2+(hlerror2==0)*1);
                e3 = (errn' - errnP')./(errn'+(errn'==0)*1);

                h1 = max(min(h1.*(1-sign(e1)*u.*exp(-e1)),maxms),minms);
                h2 = max(min(h2.*(1-sign(e2)*u.*exp(-e2)),maxms),minms);
                h3 = max(min(h3.*(1-sign(e3)*u.*exp(-e3)),maxms),minms);

                a1 = max(min(a1.*(1-sign(e1)*v.*exp(-e1)),maxms),minms);
```

```
            a2 = max(min(a2.*(1-sign(e2)*v.*exp(-e2)),maxms),minms);
            a3 = max(min(a3.*(1-sign(e3)*v.*exp(-e3)),maxms),minms);
        end
        dWhl_i =  repmat(h1,[1,noin]).*((hlerror.*(hl.*(1-hl)))*inpt');
        dBhl = h1.*(hlerror.*(hl.*(1-hl)));
        dWhl_2 =  repmat(h2,[1,nohn]).*((hlerror2.*(hl2.*(1-hl2)))*hl');
        dBhl2 = h2.*(hlerror2.*(hl2.*(1-hl2)));
        dWo_hl2 =  repmat(h3,[1,nohn2]).*(errn'*hl2');
        dBo = h3.*errn';

        Whl_i = Whl_i + dWhl_i;
        Bhl = Bhl + dBhl;
        Whl_2 = Whl_2 + dWhl_2;
        Bhl2 = Bhl2 + dBhl2;
        Wo_hl2 = Wo_hl2 + dWo_hl2;
        Bo = Bo + dBo;

        if ii > 1%%%%%%%%%%%%%%%%%%%%Add momentum
        Whl_i = Whl_i + repmat(a1,[1,noin]).*dWhl_ip;
        Bhl = Bhl + a1.*dBhlp;
        Whl_2 = Whl_2 + repmat(a2,[1,nohn]).*dWhl_2p;
        Bhl2 = Bhl2 + a2.*dBhl2p;
        Wo_hl2 = Wo_hl2 + repmat(a3,[1,nohn2]).*dWo_hlp;
        Bo = Bo + a3.*dBop;
        end

        dWhl_ip = dWhl_i;
        dBhlp = dBhl;
        dWhl_2p = dWhl_2;
        dBhl2p = dBhl2;
        dWo_hlp = dWo_hl2;
        dBop = dBo;
        %****************************************

        subplot(2,2,3)
        imagesc(imresize(flt,[100,100]));
        title('random itteration')
        caxis([0,1.5*max(max(target))])
        colorbar
        axis square; axis off
        pause(.01)
    else
        refresh
        errn = zeros(1,10);
    end
end
errn = errn
Errn(:,ii) = errn';

if ii > 3 && sum(sum(abs(Errn(:,ii-2:ii)))) == 0 && Err(ii) > .05
    flag = 1;
else
    flag = 0;
end
inpt = reshape(fl,[9,1])/100;
ds = dso*err;
```

```
    hlerrorP = hlerror;
    hlerror2P = hlerror2;
    errnP = errn;

    if ii < 51
        subplot(2,2,2)
        plot((Err*1000),'Color','r','LineWidth',3);
        axis square
        title(mat2str(ii))
        grid on
        subplot(2,2,4)
        plot((Errs'*1000));
        title(mat2str(ds))
        axis square
        grid on
        pause(.1)
    else
        subplot(2,2,2)
        plot((Err(:,end-50:end)*1000),'Color','r','LineWidth',3);
        axis square
        title(mat2str(ii))
        grid on
        subplot(2,2,4)
        plot((Errs(:,end-50:end)'*1000));
        title(mat2str(ds))
        axis square
        grid on
        pause(.1)
    end

    subplot(2,2,1)
    imagesc(imresize(fl,[100,100]));
    caxis([0,1.5*max(max(target))])
    axis image; axis off
    subplot(2,2,3)
    hold off
    x0 = -.5:.05:3;
    y0 = -.5:.05:3;
    [x,y] = meshgrid(x0,y0);
    xx = [x(:),y(:)];
    vi = mphinterp(model,'v','coord',xx');
    vi = reshape(vi,length(x0),length(y0));
    for i1 = 1:71
    for j1 = 1:71
    if isnan(vi(i1,j1)) == 1
    vi(i1,j1) = 0;
    end
    end
    end
    imagesc(imresize(flipud(fliplr(abs(vi(:,1:41)))),[1024,512]))
    axis image; axis off
    pause(.01)
    Vid2(:,:,ii) = flipud(fliplr(abs(vi(:,1:41))));
    refresh
end
```

```
pause
for j = 1:ii
subplot(2,2,1)
imagesc(squeeze(Vid1(:,:,j)))
title('approximation')
caxis([0,1.5*max(max(target))])
colorbar
axis image; axis off
subplot(2,2,3)
imagesc(imresize(target,[100,100]))
caxis([0,1.5*max(max(target))])
title('target')
colorbar
axis image; axis off
subplot(1,2,2)
imagesc(imresize(squeeze(Vid2(:,:,j)),[1024,512]))
axis image; axis off
pause(.1)
end

subplot(2,2,1)
imagesc(imresize(fl,[100,100]));
caxis([0,1.5*max(max(target))])
colorbar
axis image; axis off
subplot(2,2,3)
imagesc(imresize(target,[100,100]));
caxis([0,1.5*max(max(target))])
colorbar
axis image; axis off
subplot(2,2,2)
plot((Err*1000),'Color','r','LineWidth',3);
axis square
title(mat2str(ii))
grid on
subplot(2,2,4)
x0 = -.5:.05:3;
y0 = -.5:.05:3;
[x,y] = meshgrid(x0,y0);
xx = [x(:),y(:)];
vi = mphinterp(model,'v','coord',xx');
vi = reshape(vi,length(x0),length(y0));
for i1 = 1:71
for j1 = 1:71
if isnan(vi(i1,j1)) == 1
vi(i1,j1) = 0;
end
end
end
imagesc(imresize(flipud(fliplr(abs(vi(:,1:41)))),[1024,512]))
axis image; axis off

figure

subplot(1,3,1)
imagesc(imresize(target,[100,100]));
```

```
caxis([0,1.5*max(max(target))])
axis image; axis off
title('Target','FontSize',28)
subplot(1,3,2)
imagesc(imresize(fl,[100,100]));
caxis([0,1.5*max(max(target))])
axis image; axis off
title('Result','FontSize',28)
subplot(1,3,3)
plot((Err*1000),'Color','r','LineWidth',3);
axis square
title(mat2str(ii))
grid on
xlabel('Itteration','FontSize',28)
ylabel('Cost','FontSize',28)
title('Convergence','FontSize',28)
subplot(1,3,1)
colorbar
subplot(1,3,2)
colorbar
```

References

[1] G. Vunjak-Novakovic and D. Scadden, "Biomimetic platforms for human stem cell research," *Cell Stem Cell*, vol. 8, no. 3, pp. 252–261, 2011.

[2] J. Hofmann, A. Zovein, H. Koh, F. Radtke, G. Weinmaster, and M. Iruela-Arispe, "Jagged1 in the portal vein mesenchyme regulates intrahepatic bile duct development: insights into alagille syndrome." *Development*, vol. 137, no. 23, pp. 4061–4072, 2010.

[3] M. Nollert, S. Diamond, and L. Mclntire, "Hydrodynamic shear stress and mass transport modulation of endothelial cell metabolism," *Biotechnology and Bioengineering*, vol. 38, no. 6, pp. 588–602, 1991.

[4] N. Datta, Q. Pham, U. Sharma, V. Sikavitsas, J. Jansen, and A. Mikos, "In vitro generated extracellular matrix and fluid shear stress synergistically enhance 3d osteoblastic differentiation," *PNAS*, vol. 103, no. 8, pp. 2488–2493, 2006.

[5] S. Stolberg and K. McCloskey, "Can shear stress direct stem cell fate?" *Biotechnology Progress*, vol. 25, no. 1, pp. 10–19, 2009.

[6] G. Bancroft, V. Sikaviitsas, and A. Mikos, "Design of a flow perfusion bioreactor system for bone tissue-engineering applications," *Tissue Engineering*, vol. 9, no. 3, pp. 549–554, 2003.

[7] K. Reich and J. Frangos, "Effect of flow on prostaglandin e2 and inositol trisphosphate levels in osteoblasts," *American Journal of Physiology*, vol. 261, no. 3, pp. 428–432, 1991.

[8] M. Hillsley and J. Frangos, "Alkaline phosphatase in osteoblasts is downregulated by pulsatile fluid flow," *Calcified Tissue Int.*, vol. 60, no. 1, pp. 48–53, 1997.

[9] S. R., F. Mitchell, R. Howard, and T. Chambers, "Induction of no and prostaglandin e2 in osteoblasts by wall-shear stress but not mechanical strain," *American Journal of Physiology*, vol. 273, no. 36, pp. 751–758, 1997.

[10] K.-N. J., M. Helfrich, J. Sterck, H. MacPherson, M. Joldersma, S. Ralston, C. Semeins, and E. Burger, "Nitric oxide response to shear stress by human bone cell cultures is endothelial nitric oxide synthase dependent," *Biochemical and Biophysical Research Communications*, vol. 250, no. 1, p. 108114, 1998.

[11] T. McAllister, T. Du, and J. Frangos, "Fluid shear stress stimulates prostaglandin and nitric oxide release in bone marrow derived preosteoclast-like cells," *Biochemical and Biophysical Research Communications*, vol. 270, no. 2, pp. 643–648, 2000.

[12] G. Jiang, C. White, H. Stevens, and J. Frangos, "Temporal gradients in shear stimulate osteoblastic proliferation via erk 1/2 and retinoblastoma protein," *American Journal of Physiology*, vol. 283, no. 2, pp. 383–389, 2002.

[13] S. Dermenoudis and Y. Missirlis, "Bioreactors in tissue engineering," *Advanced Materials*, vol. 12, no. 11, pp. 592–608, 2010.

[14] M. Brown, R. Iyer, and R. M., "Pulsatile perfusion bioreactor for cardiac tissue engineering," *Biotechnology Progress*, vol. 24, no. 4, pp. 907–920, 2008.

[15] N. Choi, M. Cabodi, B. Held, J. Gleghorn, L. Bonassar, and A. Stroock, "Microfluidic scaffolds for tissue engineering," *Nature Materials*, vol. 6, no. 11, pp. 908–915, 2007.

[16] K. A.K.M.B., O. I.T., and B. Koc, "Engineered tissue scaffolds with variational porous architecture," *Biomechanical Engineering*, vol. 133, no. 1, p. 011001, 2010.

[17] F. Okkels, M. Dufva, and H. Bruus, "Optimal homogenization of perfusion flows in microfluidic bio-reactors: A numerical study," *PLoS ONE*, vol. 6, no. 1, p. e14574, 2011.

[18] A. Yeatts and J. Fisher, "Bone tissue engineering bioreactors: dynamic culture and the influence of shear stress," *Bone*, vol. 48, no. 2, pp. 171–181, 2011.

[19] F. Melchels, B. Tonnarelli, A. Olivares, I. Martin, D. Lacroix, J. Feijen, D. Wendt, and D. Grijpma, "The influence of the scaffold design on the distribution of adhering cells after perfusion cell seeding," *Biomaterials*, vol. 32, no. 11, pp. 2878–2884, 2011.

[20] A. Lesman, Y. Blinder, and S. Levenberg, "Modeling of flow-induced shear stress applied on 3d cellular scaffolds: Implications for vascular tissue engineering," *Biotechnol. Bioeng.*, vol. 105, no. 3, pp. 645–654, 2009.

[21] C. Potter, S. Schobesberger, M. Lundberg, P. Weinberg, J. Mitchell, and J. Gorelik, "Shape and compliance of endothelial cells after shear stress in vvitro or from different aortic regions: scanning ion conductance microscopy study," *PLoS ONE*, vol. 7, no. 5, pp. 1–5, 2012.

[22] Y.-C. Toh and J. Voldman, "Fluid shear stress primes mouse embryonic stem cells for differentiation in a self-renewing environment via heparan sulfate proteoglycans transduction," *FASEB*, vol. 25, no. 4, pp. 1208–1217, 2011.

[23] L. Griffith and M. Swartz, "Capturing complex 3d tissue physiology in vitro," *Nat. Rev. Mol. Cell Bio.*, vol. 7, pp. 211–224, 2006.

[24] J. Culver, J. Hoffmann, R. Poché, J. Slater, J. West, and M. Dickinson, "Three-dimensional biomimetic patterning in hydrogels to guide cellular organization," *Adv. Mater.*, vol. 24, no. 17, pp. 2344–2348, 2012.

[25] M. Song, D. Dean, and M. Knothe Tate, "Mechanical modulation of nascent stem cell lineage commitment in tissue engineering scaffolds," *Biomaterials*, vol. 34, no. 23, pp. 5766–5775, 2013.

[26] P. Tseng and D. Di Carlo, "Substrates with patterned extracellular matrix and subcellular stiffness gradients reveal local biomechanical responses," *Adv. Mater.*, vol. 26, no. 8, pp. 1242–1247, 2013.

[27] Z. Zhang, L. Yuan, P. Lee, E. Jones, and J. Jones, "Modeling of time dependent localized flow shear stress and its impact on cellular growth within additive manufactured titanium implants," *J. Biomed. Mater. Res., Part B*, vol. 102, no. 8, pp. 1689–1699, 2014.

[28] R. McCoy, C. Jungreuthmayer, and F. O'Brien, "Influence of flow rate and scaffold pore size on cell behavior during mechanical stimulation in a flow perfusion bioreactor," *Biotechnol. Bioeng.*, vol. 109, no. 6, pp. 1583–1594, 2012.

[29] F. Couet, S. Meghezi, and D. Mantovani, "Fetal development, mechanobiology and optimal control processes can improve vascular tissue regeneration in bioreactors: an integrative review," *Med. Eng. Phys.*, vol. 34, no. 3, pp. 269–278, 2012.

[30] S. VanGordon, R. Voronov, T. Blue, R. Shambaugh, D. Papavassiliou, and V. Sikavitsas, "Effects of scaffold architecture on preosteoblastic cultures under continuous fluid shear," *Industrial & Engineering Chemistry Research*, vol. 50, pp. 620–629, 2011.

[31] R. Voronov, S. VanGordon, V. Sikavitsas, and D. Papavassiliou, "Distribution of flow-induced stresses in highly porous media," *Applied Physics Letters*, vol. 97, no. 2, pp. 024 101–024 103, 2010.

[32] ——, "Computational modeling of flow-induced shear stresses within 3d saltleached porous scaffolds imaged via micro-ct," *Biomechanics*, vol. 43, no. 7, p. 12791286, 2010.

[33] B. Porter, R. Zauel, H. Stockman, R. Guldberg, and D. Fyhrie, "3-d computational modeling of media flow through scaffolds in a perfusion bioreactor," *Biomechanics*, vol. 38, no. 3, pp. 543–549, 2005.

[34] S. S, *The Lattice Boltzmann equation for fluid dynamics and beyond: Numerical mathematics and scientific computation.* Ocford Science Publications, 2001.

[35] S. Sivanandam, S. Sumathi, and S. Deepa, *Introduction to fuzzy logic using MATLAB*. Springer, 2007.

[36] P. Callaghan, *Principles of nuclear magnetic resonance microscopy*. Clarendon Press Oxford, 1991, vol. 3.

[37] S. Hollister, "Porous scaffold design for tissue engineering," *Nature Material*, vol. 4, no. 7, pp. 518–524, 2005.

[38] E. Place, N. Evans, and M. Stevens, "Complexity in biomaterials for tissue engineering," *Nat. Mater.*, vol. 8, pp. 457–470, 2009.

[39] L. Adamo, O. Naveiras, P. Wenzel, S. McKinney-Freeman, P. Mack, and J. Gracia-Sancho, "Biomechanical forces promote embryonic hematopoiesis," *Nature*, vol. 459, no. 7250, pp. 1131–1135, 2009.

[40] E. Tzima, M. Irani-Tehrani, W. Kiosses, E. Dejana, D. Schultz, and B. Engelhardt, "A mechanosensory complex that mediates the endothelial cell response to fluid shear stress," *Nature*, vol. 437, no. 7057, pp. 426–431, 2005.

[41] S. Obi, K. Yamamoto, N. Shimizu, S. Kumagaya, T. Masumura, and T. Sokabe, "Fluid shear stress induced arterial differentiation of endothelial progenitor cells," *Applied Physiology*, vol. 106, no. 1, pp. 203–211, 2009.

[42] C. Hahn and M. Schwartz, "Mechanotransduction in vascular physiology and atherogenesis," *Nature Reviews Molecular Cell Biology*, vol. 10, no. 1, pp. 53–62, 2009.

[43] C. Helm, M. Fleury, A. Zisch, F. Boschetti, and M. Swartz, "Synergy between interstitial flow and vegf directs capillary morphogenesis in vitro through a gradient amplification mechanism," *PNAS*, vol. 102, no. 44, pp. 15 779–15 784, 2005.

[44] F. Boschetti, M. Raimondi, F. Migliavacca, and G. Dubini, "Prediction of the micro-fluid dynamic environment imposed to three-dimensional engineered cell systems in bioreactors," *Biomechanics*, vol. 39, no. 3, pp. 418–425, 2006.

[45] Y. Jia, P. Bagnaninchi, Y. Yang, A. Haj, M. Hinds, and S. Kirkpatrick, "Doppler optical coherence tomography imaging of local fluid flow and shear stress within microporous scaffolds," *Journal of Biomedical Optics*, vol. 14, no. 3, 2009.

[46] J. Santiago, S. Wereley, C. Melnhart, D. Beebe, and R. Adrian, "A particle image velocimetry system for microfluidics," *Experiments in Fluids*, vol. 25, no. 4, pp. 316–319, 1998.

[47] M. Britton and P. Callaghan, "Two-phase shear band structures at uniform stress," *Physical Review Letters*, vol. 78, no. 26, pp. 4930–4933, 1997.

[48] P. Swider, M. Conroy, A. Pedrono, D. Ambard, S. Mantell, and K. Soballe, "Use of high resolution mri for investigation of fluid flow and global permeability in a material with interconnected porosity," *Biomechanics*, vol. 40, no. 9, pp. 2112–2118, 2007.

[49] K. Youssef, J. Mack, M. Iruela-Arispe, and L.-S. Bouchard, "Macro-scale topology optimization for controlling internal shear stress in a porous scaffold bioreactor," *Biotechnol. Bioeng.*, vol. 109, no. 7, pp. 1844–1854, 2012.

[50] S. Urchuk and D. Plewes, "Mr measurements of pulsatile pressure gradients," *Journal of Magnetic Resonance Imaging*, vol. 4, no. 6, pp. 829–836, 1994.

[51] A. Scheidegger, *The physics of flow through porous media*. University of Toronto Press, 1974.

[52] F. Brandl, F. Sommer, and A. Goepferich, "Rational design of hydrogels for tissue engineering: impact of physical factors on cell behavior," *Biomaterials*, vol. 28, no. 2, pp. 134–146, 2007.

[53] J. Drury and D. Mooney, "Hydrogels for tissue engineering: scaffold design variables and applications," *Biomaterials*, vol. 24, no. 24, p. 43374351, 2003.

[54] M. Lutolf and J. Hubbell, "Synthetic biomaterials as instructive extracellular microenvironments for morphogenesis in tissue engineering," *Nature Biotechnology*, vol. 23, no. 1, pp. 47–55, 2005.

[55] S. D., "Designing cell-compatible hydrogels for biomedical applications," *Science*, vol. 336, no. 6085, pp. 1124–1128, 2012.

[56] W. John, "The extrapolation, interpolation and smoothing of stationary time series with engineering applications." *Journal of the American Statistical Association*, vol. 47, pp. 319–321, 1952.

[57] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," *Proceedings of the 26th Annual International Conference on Machine Learning*, vol. 8, pp. 689–696, 2009.

[58] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image Processing*, vol. 15, pp. 3736–3745, 2006.

[59] M. Protter and M. Elad, "Image sequence denoising via sparse and redundant representations," *IEEE Transactions on Image Processing*, vol. 18, pp. 842–861, 2010.

[60] C. S. Anand and J. S. Sahambi, "Wavelet domain non-linear filtering for mri denoising," *Magnetic Resonance Imaging*, vol. 28, pp. 175–191, 1961.

[61] R. Yan, L. Shao, and Y. Liu, "Nonlocal hierarchical dictionary learning using wavelets for image denoising," *IEEE Transactions on Image Processing*, vol. 22, pp. 4689–4698, 2013.

[62] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3d transform-domain collaborative filtering," *IEEE Transactions on Image Processing*, vol. 16, pp. 1–16, 2007.

[63] A. Buades, B. Coll, and J. M. Morel, "A review of image denoising algorithms, with a new one," *Multiscale Modeling Simulations*, vol. 4, pp. 490–530, 2005.

[64] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[65] R. W. Liu, L. Shi, W. Huang, J. Xu, S. C. H. Yu, and D. Wang, "Generalized total variation-based mri rician denoising model with spatially adaptive regularization parameters," *Magnetic Resonance Imaging*, vol. 32, pp. 702–720, 2014.

[66] P. Getreuer, "Rudin-osher-fatemi total variation denoising using split bregman," *Image Processing On Line*, vol. 4.2, pp. 74–95, 2012.

[67] M. Maggioni and A. Foi, "Nonlocal transform-domain denoising of volumetric data with groupwise adaptive variance estimation," *Proceedings SPIE 8296, Computational Imaging*, 2012.

[68] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, pp. 1–14, 2004.

[69] L. Zhanga, L. Zhanga, X. Moub, and D. Zhang, "Fsim: a feature similarity index for image quality assessment," *IEEE Transactions on Image Processing*, vol. 20, pp. 2378–2386, 2011.

[70] E. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 9, pp. 533–536, 1986.

[71] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, pp. 989–993, 1994.

[72] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising: can plain neural networks compete with bm3d?" *EEE Conference on Computer Vision and Pattern Recognition*, pp. 2392–2399, 2012.

[73] K. Dabov, A. Danieyan, and A. Foi. (2014) Bm3d demo software for image/video restoration and enhancement public release v2.00. http://www.cs.tut.fi/∼foi/GCF-BM3D/.

[74] H. C. Burger. (2012) Image denoising with multi-layer perceptrons. http://people.tuebingen.mpg.de/burger/neural_denoising/cvpr2012.html.

[75] M. Maggioni and A. Foi. (2013) Bm4d software for volumetric data denoising and reconstruction public release ver. 2.3. http://www.cs.tut.fi/∼foi/GCF-BM3D/.

[76] P. Getreuer. (2009) reciandenoise: 2d and 3d total variation based rician denoising. https://code.google.com/p/cdsc-image-processing-pipeline/downloads/list.

[77] J. Manjon, P. Coupe, L. Marti-Bonmati, D. Collins, and M. Robles, "Adaptive non-local means denoising of mr images with spatially varying noise levels," *Journal of Magnetic Resonance Imaging*, pp. 192–203, 2010.

[78] P. Coupe, J. Manjon, M. Robles, and D. Collins, "Adaptive multiresolution non-local means filter for three-dimensional magnetic resonance image denoising," *IET Image Processing*, pp. 558–568, 2012.

[79] P. Coupe, J. V. Manjon, E. Gedamu, D. Arnold, M. Robles, and D. L. Collins, "Robust rician noise estimation for mr images," *Medical Image Analysis*, pp. 483–493, 2010.

[80] P. Coupe, P. Yger, S. Prima, P. Hellier, C. Kervrann, and C. Barillot, "An optimized blockwise non local means denoising filter for 3d magnetic resonance images," *IEEE Transactions on Medical Imaging*, pp. 425–441, 2008.

[81] N. Wiest-Daessle, S. Prima, P. Coupe, S. Morrissey, and C. Barillot, "Rician noise removal by non-local means filtering for low signal-to-noise ratio mri: applications to dt-mri," 2008, pp. 171–179.

[82] J. Manjon, P. Coupe, A. Buades, D. Collins, and M. Robles, "New methods for mri denoising based on sparseness and self-similarity," *Medical Image Analysis*, pp. 18–27, 2012.

[83] A. Engler, S. Sen, H. Sweeney, and D. Discher, "Matrix elasticity directs stem cell lineage specification," *Cell*, vol. 126, no. 4, pp. 677–89, 2006.

[84] J. Rutkowski and M. Swartz, "A driving force for change: interstitial flow as a morphoregulator," *Trends Cell Biol.*, vol. 17, no. 1, pp. 44–50, 2007.

[85] K. Saha, A. Keung, E. Irwin, Y. Li, L. Little, and D. Schaffer, "Substrate modulus directs neural stem cell behavior," *Biophys. J.*, vol. 95, no. 9, pp. 4426–4438, 2008.

[86] Z. Syedain, J. Weinberg, and R. Tranquillo, "Cyclic distension of fibrin-based tissue constructs: Evidence of adaptation during growth of engineered connective tissue," *Proc. Nat. Acad. Sci. USA*, vol. 105, no. 18, pp. 653–654, 2008.

[87] D. Fletcher and R. Mullins, "Cell mechanics and the cytoskeleton," *Nature*, vol. 463, pp. 485–492, 2010.

[88] R. Udan, T. Vadakkan, and M. Dickinson, "Dynamic responses of endothelial cells to changes in blood flow during vascular remodeling of the mouse yolk sac," *Development*, vol. 140, no. 19, pp. 4041–4050, 2013.

[89] J. Mack, K. Youssef, O. Noel, M. Lake, A. Wu, M. Iruela-Arispe, and L.-S. Bouchard, "Real-time maps of fluid flow fields in porous biomaterials," *Biomaterials*, vol. 34, no. 8, pp. 1980–1986, 2013.

[90] D. Shattuck and R. Leahy, "Brainsuite: An automated cortical surface identification tool," *Medical Image Analysis*, vol. 8, no. 2, pp. 129–142, 2002.