Individually-controllable magnetic artificial cilia for microfluidic manipulation tasks

Nathan Banka

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2017

Reading Committee:

Santosh Devasia, Chair

Brian Fabien

Martin Berg

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

Individually-controllable magnetic artificial cilia for microfluidic manipulation tasks

Nathan Banka

Chair of the Supervisory Committee:
Professor Santosh Devasia
Mechanical Engineering

This thesis presents the design, modeling, and control of a magnetic artificial cilia system in which the cilia are individually controllable. In nature, cilia exhibit metachronal waves, or a phase difference between adjacent cilia that results in a traveling wave, which may improve pumping performance or efficiency of biological cilia. However, existing magnetic artificial cilia devices typically use actuation by a rotating field generated by Helmholtz coils or by a rotating permanent magnet. These field sources cannot apply a phase shift to the cilia array and therefore cannot generate a metachronal wave. Nevertheless, magnetic actuation remains desirable for cilia devices as it allows for biocompatibility, precise control of system inputs, and low-cost fabrication of the cilia. In this thesis, a new design for magnetic artificial cilia is presented in which the actuating magnetic field is localized, enabling individual actuation. However, this design decision leads to challenging research problems in input-pattern identification, nonlinear systems modeling, and control. In addressing these challenges, the contributions of this thesis are to (i) demonstrate that individual control can improve performance in cilia-based devices, (ii) present accurate nonlinear models for predicting the static response, and (iii) develop a machine-learning-based system identification and control strategy for output tracking.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This work was supported by NSF grant CMMI 1000404.

DEDICATION

For Kaitlin

Chapter 1
INTRODUCTION

1.1 Research Goal and Impact

My research goal is to develop a magnetic artificial cilia system in which the cilia are individually controllable. This is an important project for the following reasons:

1.1.1 Individual control is important for cilia systems

In nature, cilia exhibit metachronal waves, or a phase difference between adjacent cilia that results in a traveling wave[14]. This phenomenon is well-known, having been observed as early as the nineteenth century[55], and is now thought to be driven by hydrodynamic coupling between the individual cilia[78]. Numerical work has also shown that pumping performance or efficiency of an array of artificial cilia is improved by a metachronal wave actuation strategy[35]. Hence developing a system capable of individual control would enable the study of the benefits of metachronal waves as well as likely improve performance of microfluidic devices.

1.1.2 Magnetic actuation is an important category for cilia devices

Magnetic actuation offers many benefits compared to other actuation strategies. Unlike the electrostatic actuation approach[23], the magnetic field does not interact with the working fluid and is biocompatible. Compared to exciting the cilia mechanically at resonance[58], the magnetic actuation approach allows for many more actuation speeds and strategies. And compared to pneumatic actuation[33], the magnetic cilia are much simpler to fabricate and allow for more design freedom in the cilium geometry, and have no risk of rupture or permeation. Additionally, the low cost of materials and fabrication of magnetic cilia is

important for biological applications in which the cilia and chamber can only be used one time to avoid cross-contamination.

1.1.3 Existing magnetic cilia devices are not capable of individual actuation

Most existing magnetic cilia devices focus on emulating the size and quantity of cilia arrays[9, 19, 21, 28, 43]. However, working with dense arrays of microcilia requires applying the magnetic field to the whole array at once. Usually this takes the form of a rotating field generated by Helmholtz coils[19, 21] or by a rotating permanent magnet[9]. These field sources cannot apply a phase shift to the cilia array and therefore cannot generate a metachronal wave, limiting pumping performance.

1.2 The proposed solution

In order to enable individual control, the magnetic field must be localized so that it only influences a single cilium. Since the magnetic field decays as $\frac{1}{r^2}$, a field source that is on the scale of the cilia will apply a force that depends strongly on its relative location. This observation leads to the proposed design, shown in Figure 1.1. The field source, which in practice can be implemented by an electromagnet or a moving permanent magnet to create a time-varying field, preferentially actuates a single cilium due to its biased location. As long as the intercilia spacing is sufficiently large, the relatively small far field of the magnet will not cause the other cilium to deflect.

1.3 Challenge of the Research

The proposed individual actuation approach introduces several research questions:

1. *Input pattern design:* The performance benefit of individual control can be demonstrated by applying the individually-controlled cilia to a mixing task. However, the design space for inputs to an array of cilia is rather large, even when the number of cilia is relatively small: for example, the cilia could be actuated simultaneously, or with

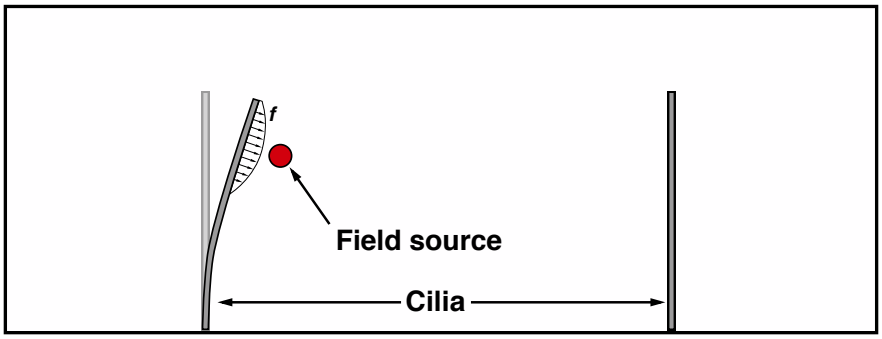


Figure 1.1: A conceptual schematic of the proposed design for a cilia-based magnetic device capable of individual actuation. When the field source is active on the left side, the left cilium deflects due to a distributed magnetic force f. Since the other cilium is far from the field source, it is not affected

a phase difference, or one cilium could beat several times alone, etc. For the case of mixing in laminar flow, an additional challenge is that steady advection alone is not sufficient to cause mixing faster than diffusion.

2. *Magnet placement:* Since the magnetic field is localized, the position of the magnet relative to the cilium has a large impact on the amplitude of the deflection. This is a necessary byproduct of the individual actuation, since the field must be relatively strong at the nearer cilium and relatively weak at the more distant cilium. The magnet placement is also has an impact on other design variables; for example, placing the magnet farther from the actuated cilium also requires an increase to the spacing between the two cilia to maintain the separable actuation. Conversely, the cilia may have a smaller spacing if the magnet-cilium distance is decreased. Understanding and predicting the effects of placement on performance of the device is therefore a critical question for the system design, which has been addressed using a combination of numerical and experimental work. However, the interaction between the magnet and the cilium is highly nonlinear because it is subject to large-deformation beam nonlin-

earities and the magnetic force is nonlinear and deformation-dependent, which makes modeling and simulation challenging.

3. *Control:* Several factors make control of the individually-controllable magnetic cilia challenging. Since the output is estimated offline from video analysis, feedback control is not applicable in the current embodiment. Also, manufacturing variability and high sensitivity to magnet placement make estimation of a general dynamic model difficult—in practice, the parameters of each cilium in a cilia-based device would need to be individually estimated and may vary based on changes in ambient conditions.

1.4 Research contributions

The main contributions of this work are as follows:

1. *Blinking vortex-based actuation patterns:* To limit the search space for actuation patterns, a certain class of input patterns was selected, inspired by the concept of the blinking vortex and other chaotic advection strategies, and applied to a mixing task: one cilium beats a certain number of times, then the other beats the same number of times, and the cycle repeats. This input strategy was found to significantly improve mixing times compared to a simultaneous-actuation strategy in which the two cilia beat synchronously. Simulations based on the blinking vortex model of chaotic advection predict similar trends to the experiments. The research on input patterns is presented in Chapter 3.

2. *Nonlinear models for magnet placement:* To address the placement question, a nonlinear finite element model was developed that couples the large-deformation structural problem with models of the magnetic field in a curved thin film and the corresponding magnetic force distribution. With only one free parameter this model was able to predict the magnet placement that maximizes tip deflection to within 5%. The modeling results are presented in Chapter 4.

3. *Iterative machine learning for control:* To enable output tracking of desired trajectories, i.e. to move the tip of the cilium in a specified pattern, an iterative learning control was used. The iterations were governed by an estimated inverse system model generated by Gaussian Process regression, a model-free machine learning algorithm that seeks to learn a non-parametric unknown function from noisy measurements. Using this method, inputs were successfully found to reduce the tracking error to within the measurement accuracy (about 3% error). The control approach is presented in Chapter 5.

Chapter 2
LITERATURE REVIEW

2.1 Biological cilia

Cilia are flexible rods found on the surface of cells. In nature, they are used as soft actuators to manipulate flows. For example, in humans cilia are found in dense arrays in the trachea and are used to move mucus away from the lungs. Cilia are also used for propulsion by organisms such as *Paramecium*. In most of these biological examples, cilia are found at the microscale, with lengths on the order of 10–15 µm; however, macroscopic examples are found in Ctenophora, a group of jellyfish that swim using "comb plates", which are rows of paddles that are formed from 100,000 or more cilia fused together and can reach lengths up to 2 µm [2].

Due to the small length scales in which cilia are found, they operate at very low Reynolds numbers. In the case of Stokes flow ($\mathrm{Re} \ll 1$), flow generation by moving structures is bound by the Scallop theorem, which states that reciprocal motion generates no net flow, even in the presence of temporal asymmetry[61]. That is, in the absence of inertial effects, the fluid-structure interaction is reversible, so if the forward and return strokes follow the same path, no net change is made, even if the speeds of the two motions are different. Cilia overcome the limitations of the Scallop theorem by exploiting spatial asymmetry. A typical cilium's beat involves a effective stroke in which the cilium is straightened, while the recovery stroke returns to the initial position with the cilium curled on itself, staying close to the cell wall.

A notable feature of biological cilia is that they exhibit non-simultaneous actuation patterns known as metachronal waves[14]. A metachronal wave is a phase difference across an array of cilia, and have been observed to move in the direction of the flow (symplectic metachronal wave), against the direction of the flow (antiplectic), or orthogonally to the flow

direction (dexioplectic or laeoplectic). These coordinated beating patterns are thought to arise through interactions between the adjacent cilia[78], and are hypothesized to improve efficiency or pumping speeds of the cilia array[35].

Metachronal waves have also been observed in the Ctenophora, providing a convenient organism for examining the coordination mechanism. In one study [70], the author attempted to prevent transmission of the metachronal wave by several different means, including mechanically restricting the motion of one or more of the comb plates. In some species, any method of preventing the motion of one plate would stop the metachronal wave from progressing past that point, indicating that hydrodynamic interactions are responsible for the coordination. However, other species feature an interplate ciliated groove, a channel between adjacent comb plates that was shown to be responsible for wave transmission: in these species, restricting the motion of a plate did not interrupt the metachronal wave, but interfering with the interplate ciliated groove prevented wave transmission. In all cases, inhibiting neural activity in the jellyfish did not prevent transmission of metachronal waves, suggesting that (at least for Ctenophora) the waves are an emergent behavior, not a controlled one.

2.2 Artificial cilia devices

The features of the biological cilia provide a useful model for the development of microfluidic devices, which many researchers have used as inspiration. Since microfluidic channels also feature low Reynolds number flow and soft actuation is desirable for interacting with biological samples, it is reasonable to expect that the biological solution for microfluidic tasks is in some sense optimized. To that end, several devices have been proposed. Since the complex internal structure of the biological cilia can not be reproduced at microscale by current engineering techniques, the artificial cilia systems generally consist of high-aspect-ratio structures that respond to a certain type of externally-applied stimulus.

2.2.1 Artificial cilia driven by electrostatic interaction

One early approach was to use bilayer artificial cilia driven by electrostatic methods for mixing in a microfluidic channel [23]. These cilia were formed by patterning layers of chromium and polyimide on a surface to form two-layer plates; these plates curl up under internal elastic stresses but straighten when voltage is applied. These cilia operated at Reynolds numbers on the order of 1 and were shown to be effective at mixing colored fluid. However, electric actuation has disadvantages due to the high electric field required, which may interact with the fluid and the sample, and which is incompatible with conductive fluids [24].

2.2.2 Artificial cilia driven by mechanical excitation

Another approach for microfluidic mixing is to use mechanical excitation to actuate an array of cilia. This method was demonstrated for batch mixing in microwells[46]. A row of cilia made from polydimethylsiloxane (PDMS) was mounted to the base of a microwell, and the whole system was vibrated using a piezoelectric actuator. Actuating at the resonant frequency of the cilia led to eight-fold improvements in mixing time compared to a microwell not containing a cilia assembly.

2.2.3 Artificial cilia driven by pneumatic pressure

A recent development in artificial cilia was a demonstration of a row of artificial cilia driven by pneumatic actuation[33]. The cilia were formed by molding hollow rods with eccentric cavities. When a pressure is applied, the expansion of the cavity causes a bending deformation in the direction of the thick wall of the cilium. This technique allows for each cilium to be individually controlled, which enables study of the effects of metachronal waves on flow generation. To that end, a row of six individually-controlled cilia were placed in a closed-loop channel and different phases were compared. It was found that the peak flow speed was generated using symplectic metachrony; however, the authors noted that the slender geometry of their cilia may reduce the overall effectiveness of their device since the cilia are

small relative to the channel width. An unresolved issue with the pneumatic actuation was that the polymer used to make the cilia (PDMS) is permeable, so air bubbles were slowly introduced into the channel through the cilium wall when the actuators were pressurized.

2.2.4 Artificial cilia driven by magnetic field interaction

Likely the most active area in artificial cilia research is in cilia driven by magnetic fields, which have several beneficial features. A major advantage of this method is that compared to electrostatic or mechanical actuation, the magnetic field does not interact with most samples of interest and is biocompatible. Additionally, the magnetic cilia can be relatively simple: only a single layer of uniform composition is required, typically using magnetic micro- or nano-particles suspended in a polymer. This fabrication simplicity is in contrast to precise cavity molding in the pneumatic case or multiple-layer processes for the electrostatic case. Essentially this means that all of the complexity is taken outside of the fluid region; this may be important for biological applications, in which the cilia can only be used once to prevent cross-contamination of samples.

In the literature studying arrays of magnetic artificial cilia, two main strategies are found. The first is to use a uniform magnetic field to drive the cilia[19, 21]. Using two sets of Helmholtz coils allows for a uniform field to be generated whose direction in the plane can be controlled. Since a high-aspect-ratio structure will tend to align itself with an external magnetic field, changing the direction of this uniform field allows for control of the cilia in the array. The second common method is to use a magnetic field source, either a permanent magnet [9, 28] or an electromagnet[62], located beneath the bases of the cilia. This magnetic field source is typically of much larger scale than the cilia array. By changing the voltage (in the case of the electromagnet) or rotating or moving the permanent magnet, the cilia are influenced by changing magnetic fields and undergo deflection.

Both of these methods preclude individual actuation of the cilia. In the case of a uniform field, all cilia are acted on by the same force, so no metachronal wave (or other individual actuation pattern) can be established. Similarly, in the case of a single large field source,

any differences in magnetic forcing across the array are not controllable and are likely minimal, since the variation of the field on the scale of the cilia array will be small. However, a substantial effort has been put into numerical simulation of individually-actuated cilia, particularly in the case of a metachronal wave along a single row of cilia. Results for a single cilium[40] have clarified the types of symmetry-breaking that allow for net flow generation, while studies of flow due to cilia beating in antiplectic and symplectic metachronal waves have predicted improved performance compared to simultaneous beating[41]. Additionally, it has been demonstrated numerically that even if the individual cilia beat symmetrically in both time and space, the presence of a metachronal wave allows net flow generation, since the overall motion of the array is non-symmetric[42]. These numerical studies, together with the demonstrated results using pneumatic cilia, emphasize the importance of individual actuation for cilia arrays.

2.3 Chaotic advection

2.3.1 Deterministic chaos

Deterministic chaos is the study of systems which do not feature any stochastic processes but which nevertheless appear to have random behavior. This feature is driven by a sensitivity to initial conditions: that is, small differences in the initial state of the system lead to large changes in the outcome. Since the initial state of a physical system can only be measured to finite accuracy, sensitivity to initial conditions leads to unpredictable outcomes in real systems. An example is the discrete sequence generated by the equation

$$x_{k+1} = 2x_k \mod 1,\ k \geq 1 \tag{2.1}$$

which for small changes in the initial condition x_0 results in vastly different trajectories through the interval $[0, 1]$, as shown in Figure 2.1.

This property that small changes in initial conditions leads to separate trajectories has important connections to mixing. If two fluid particles are initially close to one another and remain close to one another, intuitively one would not expect that motion to have mixing

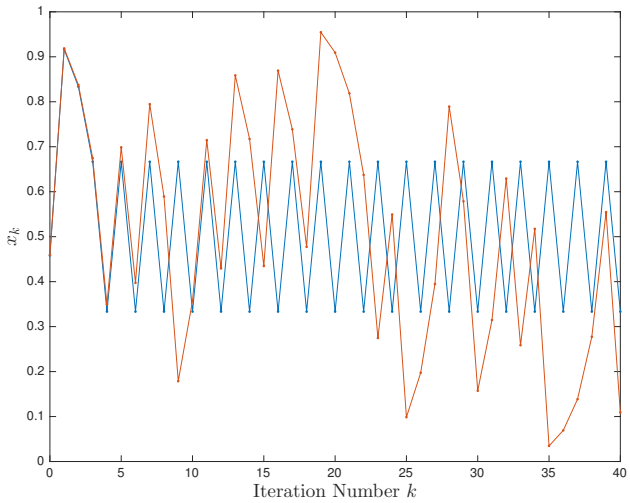


Figure 2.1: Figure showing the trajectories taken under the sequence in Equation 2.1. The nominal initial condition is $x_0 = 11/24$ (blue line); for the orange line, x_0 is perturbed by 0.001. This small change in initial condition causes very different trajectories.

properties. Conversely, if two initially-close fluid particles follow very different trajectories, mixing can be expected. This is formalized by defining the following properties of a mapping [69]:

1. *Ergodicity:* A mapping that is ergodic is essentially one that cannot be separated into two or more non-interacting mappings. Equivalently, this means that a tracer particle in the domain must go everywhere in the domain as the mapping is iterated: if there is a region not visited by a particle, then that region does not interact with the rest of the domain. However, ergodic mappings can be constructed that do not appear to "mix" the domain: an example is a rigid translation modulo the boundaries; iterated positions may cover the whole domain but two particles that start together will remain together throughout the process.

2. *Mixing:* A mapping is mixing if elements of a set A in the domain are evenly spread

across the whole domain as the mapping is iterated. This is essentially what is expected from a definition of mixing: if a colored dye is deposited in a region, a mixing process disperses it evenly throughout the domain. This is a stronger condition than ergodicity because it requires that initially-nearby particles do not remain together.

3. *Bernoulli property:* A mapping that possesses the Bernoulli property is one in which the mapped states behave like a random (Bernoulli) process. (Formally, a mapping is Bernoulli if it is isomorphic to the Bernoulli shift operator). This is an even stronger condition than mixing because it says the result of a Bernoulli mapping is indistinguishable from a randomization of the domain.

It can be shown[69] that the Bernoulli property implies mixing and mixing implies ergodicity.

2.3.2 The blinking vortex

The field of chaotic advection stems from the seminal 1984 paper by Hassan Aref[6], in which the author demonstrated via numerical simulation that the unsteady flow generated by two alternately-activated stirrers could produce chaotic or non-chaotic behavior, depending on the system parameters. This is known as the "blinking vortex", and the remainder of this section will summarize that work.

The system in consideration is a circular chamber of radius a in two dimensions; the fluid is considered to be inviscid and incompressible. The flow is generated by an ideal point vortex whose location alternates between $(x, y) = (b, 0)$ and $(-b, 0)$ at a fixed period T. The streamlines of the flow generated by the vortex at a fixed location can be found via the method of images: the streamfunction ψ for a point vortex in an infinite domain,

$$\psi = -\frac{\Gamma}{2\pi} \log r, \tag{2.2}$$

where r is the distance from the vortex center and Γ the vortex circulation, is linearly combined with additional streamfunctions ("images" of the true vortex) that enforce the boundary conditions. In the case of the circular chamber, only one image is required; the

image vortex has circulation $-\Gamma$ and is located at $(a^2/b, 0)$ for a true vortex at $(b, 0)$. The streamlines are then the level curves of the resulting linear combination. By representing the positions in the complex plane, $z = x + iy$, the streamfunction can be shown to be

$$\psi(z) = -\frac{\Gamma}{2\pi}(\ln|z - b| - \ln|z - a^2/b|) \tag{2.3}$$

$$= -\frac{\Gamma}{2\pi}\ln\left|\frac{z - b}{z - a^2/b}\right| \tag{2.4}$$

and so the level curves of $\psi(z)$ are the loci for which the argument of the logarithm is constant, i.e.

$$\left|\frac{z - b}{z - a^2/b}\right| = \lambda. \tag{2.5}$$

Via complex analysis, these level sets can be shown to be circles with well-defined centers and radii; Aref derived an implicit relation for the angle traversed by a particle with initial location $z_0 \equiv z_c(\lambda) + e^{i\phi}$ along its streamline, namely

$$\Delta\left(\phi - \frac{2\lambda}{1 + \lambda^2}\sin\phi\right) = 2\pi\Delta t/T_\lambda, \tag{2.6}$$

where T_λ is the period of rotation for that streamline and can be found explicitly. The right-hand side is known but the left-hand side must be inverted numerically to find the change in angle. The result is a mapping that can be quickly computed for arbitrarily long times Δt without loss of accuracy due to integration error. Finally, to compute the advection due to the vortex when it is located at $(-b, 0)$, the same mapping computed above can be reused by reflecting the advected particle positions about the origin, computing the map, and then reflecting again.

With the mapping defined, variations on the vortex spacing and period of alternation can be considered. Aref found trends with respect to these two parameters via numerical simulation of different test conditions. The results indicate that there is a critical value of the alternation period to achieve chaotic behavior, which depends on the spacing (with the vortices spaced farther apart, a longer alternation period is required). A larger vortex spacing was also found to result in a larger relative area of the chaotic region at fixed period.

2.3.3 Applications to DNA microarray experiments

In DNA microarray experiments, known probe strands of DNA are attached to a substrate, typically a standard 25 mm×75 mm glass slide. Unknown fluorescent-labeled DNA samples ("target") in solution are then placed on top of the probe-patterned area and allowed to hybridize with the probe strands. Then the target DNA may be identified by observing which probes have been bound by the target. In order for the results to be meaningful, it is critical that the target DNA be allowed to circulate past all probe DNA. Common practice is for the mixing to be solely diffusion-driven, and 18 or more hours are allowed for the DNA strands to circulate. However, since the diffusion rate of DNA is very low, reaching equilibrium would actually take days[65], which not only reduces the throughput of the method but can also introduce systemic bias in the results[68]. Since the DNA is usually delivered in a very thin layer of fluid, precluding turbulent mixing flow, there is substantial interest [20, 26, 76] in generating chaotic advection flows for microarray applications.

One such device is commercially available under the name ArrayIt. The operating principle is that four inlets are placed near the corners of a mixing chamber. In one half of the mixing cycle, one corner inlet acts as a source while the opposite corner inlet acts as a sink. In the other half of the cycle, the other opposite pair of inlets is active in a source-sink pair. This pattern is called a pulsed source-sink device, and is known to exhibit chaotic behavior[20, 39]. Since fluid is removed from the chamber via the sink, it is reinjected on the next half-cycle through the neighboring source. This process requires substantial additional working volume compared to a diffusion-based process: not only is the volume of fluid in the chamber substantially larger, but a significant quantity of additional fluid is contained in the mixing loop and unavailable for reaction with the probe strands. In the ArrayIt device, 350 µL of working fluid is required, of which only 60 µL is contained in the chamber. Despite the resulting decreased target DNA concentration, the chaotic advection method decreases the overall process time substantially: the manufacturer's documentation suggests hybridization times on the order of two hours are achievable (compared to 18 hours or more for the

diffusion-based process). The performance of this device suggests that chaotic advection is a viable direction for mixing applications and a device that reduces the quantity of nonreactive fluid would be an important advancement.

2.4 Iterative learning control

Soft actuators, typically motivated by bioinspired design, are of interest for robotics as they may reduce damage to manipulated objects and improve grip during manipulation tasks [56, 75], and have also been implemented as electro-active polymers [17] or pneumatic artificial muscles [22]. For output trajectory tracking, there is a need to experimentally identify the dynamic response model to find exact-tracking inputs. Iterative methods have previously been successfully applied to soft actuator control[47, 50]. Additionally, iterative corrections can improve the output tracking precision [15, 37, 54, 74].

Conditions for convergence of iterative methods have been well studied in literature, e.g., see [3]. For example, the need to invert the system G to find the perfect input u^* for tracking a desired output y_d has motivated the use of the inverse $\hat{G}^{-1}$ of the known model $\hat{G}$ of the system G in early iterative control development [7]. Since convergence depends on the size of modeling error, improvements of the model through parameter adaptation with data acquired during the iteration was studied in, e.g., [10, 11, 30, 57]. For nonminimum-phase systems (which include flexible structures such as the magnetic actuator being considered in this article), a non-causal inverse was initially proposed for iterative learning control in [32] and frequency domain implementation of the non-causal inverse was studied in, e.g., [36, 72]. Convergence to the desired output can be guaranteed if the phase uncertainty in the model is less than 90 degrees and the iteration gain is sufficiently small [72, 73]. Such dependence on the phase uncertainty was also developed using a discrete Fourier transform approach in [29, 36]. However, there can be regions in the frequency domain where convergence cannot be guaranteed, e.g., where the phase uncertainty in the model $\hat{G}$ is greater than 90 degrees. Frequency regions where convergence cannot be guaranteed can be reduced by using the input-output data generated during the iteration procedure [45]. Nevertheless,

such input-output data might have substantial error at some frequencies where the signal to noise ratio is small, which can, in turn, lead to error in the acquired models. Therefore, methods are needed to manage the model-estimation in the presence of noise.

Chapter 3

INDIVIDUALLY-CONTROLLABLE MAGNETIC CILIA: MIXING APPLICATION

3.1 Introduction

This chapter introduces a new design for individually-controlled magnetic artificial cilia for use in fluid devices, and specifically intended to improve the mixing in DNA microarray experiments. The design has been implemented using a low-cost prototype that can be fabricated using polydimethylsiloxane (PDMS) and off-the-shelf parts, and achieves large cilium deflections (59% of the cilium length). The device's performance is measured via a series of mixing experiments using different actuation patterns inspired by the blinking vortex theory. The experimental results, quantified using the relative standard deviation of the color when mixing two colored inks, show that exploiting the individual control leads to faster mixing (38% reduction in mixing time) than when operating the device in a simultaneous-actuation mode with the same average cilium beat frequency. Furthermore, the experimental results show an optimal beating pattern that minimizes the mixing time. The existence and character of this optimum is predicted by simulations using a blinking-vortex approach for 2D ideal flow, suggesting that the blinking-vortex model can be used to predict the effect of parameter variation on the experimental system.

3.2 System design and fabrication

3.2.1 Mixing chamber design

To compare performance with and without individual control, the actuation method shown in Figure 4.1 has been applied to a batch-mixing process on a standard 25 mm×75 mm glass slide, as is commonly used for DNA microarray experiments[1, 26, 76]. The chamber was

made from polydimethylsiloxane (PDMS) via a molding process and has interior dimensions of 40 mm×18 mm×3.8 mm.

The cilia presented here have lengths of 13 mm. This cilia size has not been optimized, but is similar to recent work[33] demonstrating effective pumping using pneumatically-actuated artificial cilia that were 8 mm long and 1 mm in diameter, and used a similar ratio of cilium length to channel width. The mixing strategy of the present work is based on the theory of the blinking vortex[6, 77], which involves two spatially-overlapping vortices that are alternately activated; this leads to the natural choice to use two cilia, each of which can generate a vortex.

3.2.2 Apparatus construction

The prototype implementation of the conceptual design shown in Figure 4.1 uses two cilia to generate mixing flows. The magnetic field sources are two wheels of 16 mm radius which each have three cylindrical permanent magnets mounted along the circumference (shown in Figure 3.1); the permanent magnets are grade N50 NdFeB (i.e. their stated maximum energy product is $(BH)_{max} = 50$ MG $\cdot$ Oe), and measure 4.2 mm in diameter and 4 mm tall. Each wheel is assigned its own motor and can be rotated individually, producing a time-varying magnetic field. The wheels and the support structure have been constructed from Lego parts; the driving motors and software are from a Lego Mindstorms NXT set. Photos of the apparatus are shown in Figure 3.2. From the perspective of prototyping, the use of Lego parts was useful to test different design concepts and actuation patterns; the system size could be reduced by using smaller structural components and more compact motors. However, the current size of the device, measuring 18 cm×26.5 cm×14 cm, is consistent with existing commercial devices[26]: as an example, the manufacturer's documentation for the ArrayIt TrayMix S4 reports dimensions of 35 cm×26.5 cm×14 cm.

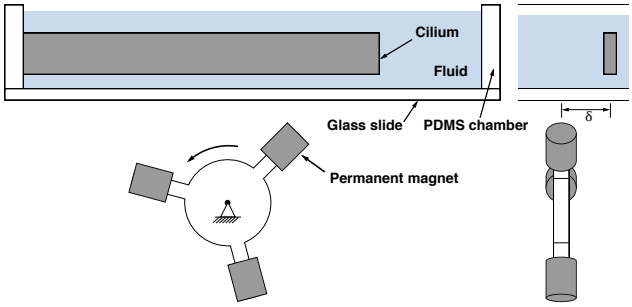


Figure 3.1: Schematic of the proposed actuation approach for each cilium, showing the magnet wheel used as field source. As the magnet wheel rotates, the magnets move closer and farther from the cilium, causing a time-varying force to be applied. As shown in the side view on the right, the magnet wheel is offset from the cilium axis by a distance δ, resulting in transverse deflection of the cilium

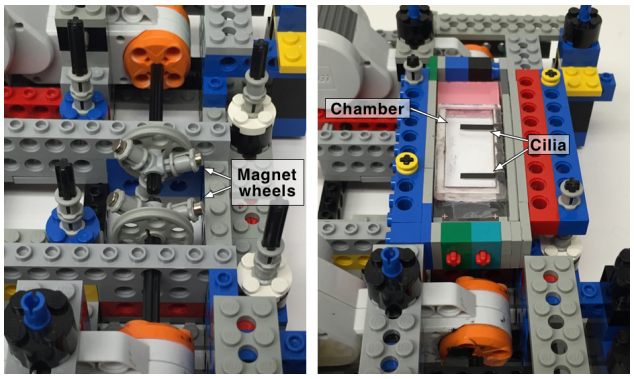


Figure 3.2: Photos showing the support structure of the cilia-mimetic mixer prototype. Left: the device with the mixing chamber removed to show the magnet wheels. Right: the apparatus with the cilia, chamber, and supporting slide mounted in a slide holder

3.2.3 Magnetic cilia

The cilia are flexible rectangular cantilevers measuring approximately 13 mm in length, with cross-section 2.5 mm×0.3 mm, made from a composite of PDMS and carbonyl iron microparticles with a stated grain size of 5–9 µm (Sigma-Aldrich). Similar composites have previously been used in the magnetic artificial cilia literature and have been reported to have fatigue life in the tens of millions of cycles[62]. The process of fabricating the Fe-PDMS cilia and mounting them in a chamber is shown in Figure 3.3, and is as follows: the PDMS prepolymer and curing agent were combined at a 10:1 mass ratio. Thereafter, the iron powder was added and mixed by hand, with the final mixture having 80% iron by mass. The uncured Fe-PDMS mixture was poured onto a glass plate and placed in a vacuum chamber until most air bubbles were removed (Figure 3.3a). A second glass plate was placed on top of the Fe-PDMS so that both surfaces of the cured sheet would be smooth and flat; to ensure uniform thickness, a weight was placed on top and a length of thin-gauge wire used as a spacer between the two plates (Figure 3.3b-c). The assembly was then cured at 80 °C in an oven for two hours, and (after cooling) the glass plates pried apart (Figure 3.3d) and the Fe-PDMS sheet cut into cilia using a knife (Figure 3.3e). Because of the difficulty removing air bubbles from the Fe-PDMS mixture, care was taken to cut the cilia from bubble-free regions of the Fe-PDMS sheet. The cilia were then mounted into a rectangular PDMS chamber that was made in a mold; the mounting procedure involves cutting through the wall of the chamber (Figure 3.3f) and gluing the cilia in the gap created using a small amount of uncured PDMS (Figure 3.3g), followed by a final curing step (Figure 3.3i). In this final step, excess pieces of Fe-PDMS were placed under the cilia (Figure 3.3h) to ensure the cilia were mounted level and to create an air gap between the cilia and the glass slide, which is important to allow the cilia to vibrate freely. After curing, these support structures are removed.

Table 3.1: Placement data corresponding to the numbered locations in Figure 3.4

Location	1–3, 6–9	4	5, 11, 12	10
# Beats	No crossing	5	2	4

3.2.4 Magnet placement

Because the magnetic force is localized and the cilia are very flexible, the performance of the device is dependent on the position of the magnets. A good placement will balance the competing goals of producing large deflections (and therefore generating strong flow), minimizing frictional interactions between the slide and the cilium, and reducing shear stresses on sensitive samples, which can be damaged by aggressive agitation[71]. The placement was selected experimentally, based on the size and apparent strength of the vortex generated by the beating of the cilium, as assessed by filling the chamber with water and adding a drop of ink. Since the blinking vortex theory suggests vortex-vortex overlap is key for mixing[6], the metric used for assessing placement was the number of beats required to generate a vortex that crosses over the centerline between the two cilia. The first round of trials, shown as squares in Figure 3.4, resulted in only a few viable placements: locations 6–9 were too far from the cilium and resulted in no deflection, while locations 1–3 exhibited saturation because the out-of-plane forces caused the cilium to contact the slide. This out-of-plane motion typically occurs when the cilium is directly above the magnet, where the magnetic field component out of the plane is large. Since so few viable placements were found, a second set of trials, shown in circles in Figure 3.4, was performed. Of these, locations 11 and 12 showed similar results, so a location between them was selected as the final placement, shown as the red circle in Figure 3.4. These results also indicate that 7 mm is a critical magnet-cilium distance, beyond which the deflection is very small. This implies that for the selected placement the minimum intercilia spacing is about 12 mm—at this spacing, a magnet at the selected placement for the left cilium (4.8 mm away) is sufficiently far from

the right cilium (more than 7 mm) to avoid actuating both cilia at once.

3.2.5 Cilium beat period

To maximize the influence of each beat, i.e. to allow the generated vortex sufficient time to propagate, the wheel angular speed was chosen to be quite slow; the motors were driven at 10% power (below which the wheel speed was not reliably constant), which results in a beat frequency of 0.8 Hz, or a wheel angular rotation speed of $\omega = 1.67$ rad/s. This results in a time period between successive beats, T, of

$$T = \frac{2\pi}{\omega N_{mag}}, \tag{3.1}$$

or 1.25 seconds when the number of magnets on the wheel, N_{mag}, is three.

3.2.6 Actuation patterns

The individually-controlled patterns, modeled after the blinking vortex[6], are shown in Figure 3.5. Since only full beats of the cilia are considered, the actuation patterns are discrete sequences, and the closest analogy to the blinking vortex is to beat the left cilium n times, then beat the right cilium n times, and repeat. This pattern is denoted Ln-Rn and the selected patterns are shown in Figure 3.5.

For comparison, a simultaneous-actuation case is considered with the same average cilium beat frequency as the alternating Ln-Rn patterns when compared in the limit over many time-steps. This condition corresponds to an equal energy input rate between the simultaneous and alternating patterns. The resulting simultaneous-actuation pattern, denoted "Simul" in the figures to follow, involves beating both cilia together on odd-numbered time-steps, and beating neither cilium on even-numbered time-steps.

3.3 Experiments

The performance of the prototype cilia-mimetic mixer was evaluated using a series of experiments to determine the mixing rates when using the different actuation patterns shown in

Figure 3.5 and discussed in Section 3.2.6.

3.3.1 Chamber preparation procedure

The chamber was prepared by filling it with 3 mL water using a pipette. Then, blue and red ink droplets (each 2.5 µL) were placed at the cilium tips using a micropipette—see Figures 3.10 to 3.12 for the ink drop placement. Then the cilia device was activated and the resulting mixing was recorded to digital video using a smartphone with an image acquisition rate of 30 Hz. Finally, the fluid was pipetted out of the chamber to make it ready for the next experiment.

3.3.2 Quantification of mixing performance

The progress of the mixing process was assessed using the statistical distribution of the color of the water in the chamber, as represented by the blue-yellow b^* channel of a Lab color space (CIELAB 1976[59]). A Lab color space treats lightness information as its own channel, L^* (ranging from 0-100), separate from hue information, which helps to correct perceived color variations due to shadows in the experimental video. The hue information is contained in the other two channels, a^* and b^*, which quantify the hue space from green to red and from blue to yellow, respectively. The numerical range of the hue channels is from 0-255.

Due to the way the a^* and b^* values map on to the sRGB color space representable by computers, for the relatively low lightness values ($L^* < 50$) present in the video images, the difference between the red and blue inks is much smaller along the red-green a^* axis than along the blue-yellow b^* axis. The result of this fact, as can be seen in Figure 3.6, is that when using the a^* channel, the red and blue areas tend to be considered similar and the white areas (i.e. areas with no ink) are considered extreme. On the other hand, when using the b^* channel, the three dominant colors in the scene (red, blue, and white) can be differentiated: b^* is small where there is blue ink, moderate where there is no ink, and large where there is red ink. Additionally, the much larger range of b^* values reduces the influence of noise on

the computation of the color variation in the chamber. Therefore, the blue-yellow channel b^* has been used to quantify mixing in the experiments to follow.

The relative standard deviation, denoted c_v, was used as a measure of the variation of the blue-yellow b^* channel and hence the overall color variation in the chamber. This measure, also called the coefficient of variation (hence the symbol c_v), is a standard metric in mixing[51, 60]. The relative standard deviation of the blue-yellow b^* channel is computed as

$$c_v(b^*) = \frac{\sigma(b^*)}{\mu(b^*)}, \tag{3.2}$$

where $\sigma(b^*)$ and $\mu(b^*)$ are the standard deviation and mean of the blue-yellow b^* color data in each frame of the video.

3.3.3 Effect of actuation pattern on mixing rate

To measure the influence of actuation pattern on the mixing rate, experiments with different actuation patterns were performed. Ten experiments were performed for each of the patterns in Figure 3.5 in a randomized sequence; for each video, the degree of mixing, quantified using the relative standard deviation $c_v(b^*)$ in Equation 3.2, was computed twice per second (independent of the cilium motion) over the pixels in each image (of which there were 3.5×10^5 per image) and plotted as a function of time. This image analysis was initiated just before the first beat of the cilia to capture the initial conditions. The patterns were compared on the basis of the mixing time, which was defined as the time it takes for the relative standard deviation $c_v(b^*)$ to fall below a threshold value $\bar{c}_v = 0.05$, which is a typical value in mixing literature[60] and represents a level at which the color variations transition from noticeable to unnoticeable.

3.4 Blinking vortex simulations

The blinking vortex theory[6] concerns chaotic behavior of iterated alternating mappings in inviscid, incompressible, two-dimensional flow. In its original development, the fluid domain

is taken to be a circular region in the complex plane $\mathbb{C}$ with radius a. An ideal point vortex with circulation Γ is introduced into this domain whose position alternates between $z = \pm b$ with alternation time period τ. Although the flow for each half-cycle is non-mixing, alternating between them can lead to chaotic behavior. It was shown that the chaotic or non-chaotic nature of the flow is determined by the vortex position b and alternation period τ or their non-dimensional counterparts

$$\beta = \frac{b}{a}, \qquad \mu = \frac{\Gamma\tau}{2\pi a^2}. \tag{3.3}$$

In the remainder of this section, the blinking vortex theory is applied to a rectangular domain in the complex plane with corners at $z = 0$, W, iH, and $W + iH$.

3.4.1 Mapping between rectangle and half-plane

The equations of motion in the rectangular domain are stiff, making numerical solution difficult. The dynamics can be made more tractable by the mapping

$$w = \wp(z), \tag{3.4}$$

where $\wp(z)$ is the Weierstrass elliptic function with half-periods $\omega_1 = W$ and $\omega_3 = iH$. The function $\wp(z)$ is a one-to-one conformal map from the rectangle in the complex z-domain to the lower-half-plane (i.e. $\operatorname{Im} w \leq 0$), and satisfies the differential equation

$$(\wp'(z))^2 = 4(\wp(z))^3 - g_2\wp(z) - g_3, \tag{3.5}$$

where g_2 and g_3 are constants that depend on the dimensions W and H[25]. Details on computation of the Weierstrass elliptic function $\wp$ and the associated constants can be found in Section 23.22 of Ref. 25.

The reverse map, $z = \wp^{-1}(w)$, can be written as [4]

$$z = (e_1 - e_3)^{-\frac{1}{2}} \int_0^{\phi} \frac{d\theta}{\sqrt{1 - m\sin^2\theta}}, \tag{3.6}$$

where

$$(\sin\phi)^2 = \frac{e_1 - e_3}{w - e_3}, \qquad\qquad m = \frac{e_2 - e_3}{e_1 - e_3}, \tag{3.7}$$

and e_1, e_2, and e_3 are the roots of the polynomial in Equation 3.5. To accelerate this inversion procedure, and since the derivative of the Weierstrass elliptic function is readily computed, the Newton-Raphson method can be used first to numerically solve the equation $w - \wp(z) = 0$; the elliptic integral in Equation 3.6 is used for any values of w for which the Newton-Raphson method does not converge within a small number (e.g., 15) of iterations.

3.4.2 Solution in half-plane

The conjugate velocity $\dot{\overline{w}}$ of a particle in the half-plane may be computed using the chain rule as

$$\dot{\overline{w}} = \frac{d\overline{z}}{dt}\frac{d\overline{w}}{d\overline{z}} = \overline{\wp}'(z)\dot{\overline{z}}. \tag{3.8}$$

To obtain the velocities in closed form, the dependence on z, the position in the rectangle, should be eliminated. In the w-plane, the streamlines may be computed as the level curves of the Hamiltonian H. If the vortex in the z-plane is located at z_1, the corresponding vortex in the w-plane is at $w_1 = \wp(z_1)$ and the boundary conditions are satisfied with a single image at $\overline{w}_1$, leading to the Hamiltonian

$$H = \frac{\Gamma}{2\pi}\ln\left|\frac{w - w_1}{w - \overline{w}_1}\right| = \frac{\Gamma}{2\pi}\ln\left|\frac{\wp(z) - \wp(z_1)}{\wp(z) - \wp(\overline{z}_1)}\right|, \tag{3.9}$$

which, since the mapping in Equation 3.4 is conformal, is also valid for the z-plane. Then the velocities in the z-plane are

$$\dot{\overline{z}} = -\frac{dH}{dy} - i\frac{dH}{dx} = \frac{2}{i}\frac{dH}{dz} = \frac{2}{i}\frac{dw}{dz}\frac{dH}{dw}, \tag{3.10}$$

and combining Equations 3.5, 3.8 and 3.10 gives the velocities in terms of the mapped position w alone as

$$\dot{\overline{w}} = |\wp'(z)|^2\frac{2}{i}\frac{dH}{dw} = |4w^3 - g_2 w - g_3|\frac{2}{i}\frac{dH}{dw}. \tag{3.11}$$

The streamlines in the w-plane are the level curves of Equation 3.9, which are equivalent to the level curves of the argument of the logarithm, i.e.

$$\left|\frac{w-w_1}{w-\overline{w}_1}\right| = \lambda, \qquad 0 \le \lambda < 1. \tag{3.12}$$

This level set is a set of circles given by $w = w_c + re^{i\theta}$, with

$$w_c = \frac{w_1 - \lambda^2 \overline{w}_1}{1-\lambda^2}, \qquad r = \frac{\lambda}{1-\lambda^2}\left|w_1 - \overline{w}_1\right|. \tag{3.13}$$

Inserting $w = w_c + re^{i\theta}$ into Equation 3.11 and solving for the angular velocity $\dot{\theta}$ leads to

$$\dot{\theta} = \frac{\Gamma}{2\pi r^2}\frac{|4w^3 - g_2 w - g_3|(1-\lambda^2)}{\lambda^2 - 2\lambda\sin\theta + 1}, \tag{3.14}$$

which fully describes the particle motion in the mapped w-domain in terms of the angular position θ alone, since the streamline constant λ can be computed for the initial conditions using Equation 3.12 and remains constant during the motion. These dynamics are non-stiff since the particle trajectories are bounded and solutions remain on their streamlines.

3.4.3 Blinking vortex simulations in the rectangle

To simulate the flow in the rectangular chamber, the vortex center location $z_1 = (6 + 15\mathrm{i})\,\mathrm{mm}$ and circulation $\Gamma = 350\,\mathrm{mm}^2/\mathrm{s}$ were estimated from the experimental videos. To compare to the experiments, the domain width $W = 25.5\,\mathrm{mm}$ is taken to be the intercilia spacing and the domain height $H = 17.7\,\mathrm{mm}$ is selected to match the chamber height; this approach neglects the flow past the cilia tips. Blue and red tracer particles are used to visualize the results of iterating the dynamics (Equation 3.14) and mapping back to the z-plane, alternating between z_1 and the symmetrically-located position every nT seconds, where n is the beat number and T is from Equation 3.1. The mixing time was computed as in the experiments: the simulation domain was subdivided into pixel-sized bins; for each bin, the difference between the number of red and blue tracer particles was used to compute the "color" of that bin as

$$b_s^* = \mathrm{sat}(k(N_{\mathrm{red}} - N_{\mathrm{blue}}) + 128) \tag{3.15}$$

where sat$(\cdot)$ limits the argument to the range $[0, 255]$ and the scaling constant k is chosen such that the initial conditions of the tracers result in saturation of the corresponding bins. Then the coefficient of variation $c_v(b_s^*)$ is used to assess the mixing progress after each cycle of the simulated mixing process.

3.5 Results and discussion

3.5.1 Vortex generated by an individual cilium

The mechanism by which the cilia device generates vortices is illustrated in Figure 3.7. As a magnet approaches the glass slide (see Figure 3.1), the cilium deflects and fluid is drawn into the space behind it (Figure 3.7c-h). As the cilium moves above the magnet, magnetic forces normal to the glass slide start to become significant, eventually resulting in the cilium deflecting such that it touches the glass slide (Figure 3.7e-h). This leads to friction forces that constrain the deflection of the cilium, and introduce a bend in the cilium near the tip. When the magnet moves too far from the cilium to hold it in place, the internal elastic forces cause the cilium to snap back to its original position (Figure 3.7i-j), creating fluid flow toward the center of the slide along the top of the image, causing a vortex to form that rotates clockwise on the left side and counter-clockwise on the right.

The magnetic actuation approach is able to produce large deflections of the cilium. Figure 3.7i shows a deflection of 59% of the cilium's length, and an angle of 42.5° from the undeflected position. This large deflection, combined with the fast snap-back (about two frames, or 0.07 s), allows each beat of the cilium to have a strong influence on the flow—compare the ink pattern in Figure 3.7a to the pattern one beat later, in Figure 3.7s.

The role of inertial effects (as compared to viscous effects) can be assessed by using the still images in Figure 3.7 to estimate the operating Reynolds number,

$$\mathrm{Re} = \frac{VL}{\nu}, \tag{3.16}$$

where V and L are fluid velocity and characteristic length of the device, and $\nu = 1.1 \times 10^{-6}\,\mathrm{m}^2/\mathrm{s}$ is the kinematic viscosity of water. For this device, the tip speed of the cilium is taken as

the maximum fluid velocity (i.e. a no-slip condition) and the length of the cilium as the characteristic length[23, 33]. By measuring the tip position in each frame, the tip speed can be estimated as

$$V \approx \frac{\Delta x}{\Delta t}, \tag{3.17}$$

where Δx is the change in tip position and Δt is the time step, i.e. one frame (0.03 s). Since the images are taken from above the slide, the out-of-plane motion is not included in the estimates of Δx; however, since the gap between the cilium and the slide is small compared to the in-plane deflections (0.4 mm vs 7.7 mm), most of the deflection is in-plane. Averaging over ten beats of the cilium results in an estimated peak tip speed of 161 mm/s for the left cilium and 140 mm/s for the right cilium; using Equation 3.16, the Reynolds number is computed as 1903 for the left cilium and 1655 for the right cilium, which are similar to the device presented by Gorissen et al[33]; the discrepancy between the two cilia is likely due to differences in the geometry of the cilia and the placement of the two magnet wheels. These Reynolds numbers indicate that inertial effects do play a significant role in the performance of the device.

The relative role of advection and diffusion can be estimated using the Péclet number, which may be computed as the ratio

$$\mathrm{Pe} = \frac{VL}{D} \sim 10^3\text{–}10^4 \tag{3.18}$$

where the diffusion coefficient D for ink in water has been estimated as 0.2–0.5 mm^2/s based on experimental results using a similar ink[48]. This value is in line with existing cilia-based mixers[23] and biological cilia[18], and suggests that advective transport is substantially more important than diffusive transport for the cilia device. This is confirmed by the results of observing the undisturbed diffusive mixing in the device, as shown in Figure 3.8. After four minutes, the coefficient of variation remains well over the threshold (with $c_v(b^*) = 0.33$) and mixing has not taken place. Since the diffusion coefficient for DNA is even lower than for ink, on the order of 10^{-4}–10^{-5} mm^2/s[51], advection would be expected to be even more critical for a DNA mixing process.

Table 3.2: Summarized experimental results. Mixing time is defined as the time to reach a relative standard deviation of $c_v(b^*) < 0.05$

	Mixing time		Peak $c_v(b^*)$	
Pattern	Avg	Std. Dev.	Avg	Std. Dev.
Diffusion-only	Did not reach mixing criterion			
Simultaneous	123.4	16.8	0.398	0.021
L1-R1	113.4	7.7	0.385	0.012
L2-R2	76.8	9.5	0.340	0.016
L5-R5	87.2	8.6	0.436	0.010
L10-R10	110.7	11.9	0.449	0.005

3.5.2 Simultaneous control leads to stagnation

The mixing rates for all of the patterns are shown in Figure 3.9, which plots the relative standard deviation $c_v(b^*)$ against the elapsed time for a selection of experiments. At the start of the experiment, most of the color is concentrated in relatively small ink drops and are dispersed by the first few beats of the cilia. The result is that the relative standard deviation $c_v(b^*)$ starts at an artificially small value before increasing to a maximum, as can be observed in Figure 3.9. When evaluating the experimental data to determine mixing times, data points before this peak were not considered. Table 3.2 summarizes the average mixing time for each pattern, as defined in Section 3.3.3.

The flow produced when operating the device in simultaneous-actuation mode is shown in Figure 3.10. If the flow were laminar and the device were perfectly symmetric, the expected outcome would be that mixing would occur only by diffusion along the line of symmetry. However, it is clear from the video images (see, e.g., Figure 3.10d) that, due to unsteady flow, ink is also transferred from one side to the other, primarily along the bottom of the image. The result of the unsteady flow where the two vortices meet is that the results

for the simultaneous pattern show much higher variance than the alternating patterns (see Table 3.2).

3.5.3 Individual control promotes mixing

The mixing action of the cilia device using individual actuation was inspired by the blinking vortex mixer[6], and the device uses a similar process to mix the two ink drops. This process is illustrated in Figure 3.11. As the left cilium beats, the nearby fluid is circulated. After many beats, a uniform blue region results. Since the two vortices overlap, the blue region covers more than half of the chamber and extends into the area that will be circulated by the right cilium (Figure 3.11b). Then, when the actuation pattern switches, the right cilium will cause the fluid on the right side of the chamber to circulate. Because the previous actuation of the left cilium introduced blue fluid into the region of circulation, the result after many beats will be a uniform region whose color is a red-purple (Figure 3.11c). On subsequent cycles, just after the actuation pattern switches, the part of the chamber circulated by the newly-active cilium will be the less uniform side of the chamber; through the beating of the cilium, its side of the chamber will be made uniform and the other side of the chamber made non-uniform; and after each cycle, both halves of the chamber will converge toward a mean color, as seen in Figure 3.11d-f.

However, a practical actuation pattern Ln-Rn, with finite n, will not necessarily achieve the full uniformity shown in Figure 3.11. As one cilium beats, it tends to make the fluid on its side more uniform, which results in diminishing returns as the number of beats per cycle, n, increases. The circulation of the fluid in the chamber is most effective at mixing the colors just after the actuation switches, when the fluid on the active side of the chamber is least uniform. On the other hand, it takes several beats for the vortex to reach into the other half of the chamber as shown in Figure 3.11b. These competing effects lead to an optimal number of beats n in the Ln-Rn patterns that balances the diminishing returns on successive beats against the time for the vortex to fully develop.

While in general the optimal number of beats can be expected to depend on parameters

such as the magnet placement and fluid properties, for the conditions of the experiments performed here, the summarized data in Table 3.2 shows that the L2-R2 pattern mixes the fastest. Compared to the simultaneous pattern, the L2-R2 pattern reaches a mixed state (corresponding to relative standard deviation $c_v(b^*) < 0.05$) in 38% less time. The tradeoffs between small n and large n in the Ln-Rn pattern, discussed above, can be seen by comparing images of the flow from the L2-R2 pattern in Figure 3.12 and the L10-R10 pattern in Figure 3.11. As discussed above, the longer actuation pattern (larger n) results in a larger, more uniform region of each color, but each cycle takes more time. An additional benefit of a shorter actuation pattern (smaller n) is also evident in the lower maximum relative standard deviation $c_v(b^*)$ shown in Table 3.2—about 0.34 in the L2-R2 case compared to about 0.4 in the simultaneous case and about 0.45 in the L10-R10 case—which indicates that for the shorter L2-R2 pattern, substantial mixing is occurring before the two ink droplets have been fully dispersed. In the video images, this behavior can be seen when comparing Figure 3.10c to Figure 3.12c; the color on the right half of the chamber shows that in the alternating case, the blue ink has already started to mix into the red, creating a dark purple color; on the other hand, the red in the simultaneous case is bright and relatively unmixed.

3.5.4 The simulations predict similar trends

While there are differences between the idealized simulations and the experiments, the predictions are similar to the observed behavior in the experiments discussed above. In particular, the simulations assume ideal flow (2D, inviscid, incompressible) with no inertial effects, which leads to vortices that form and vanish instantaneously; conversely, the experiments take place in a three-dimensional domain with both viscous and inertial contributions, resulting in vortices that move and decay after being shed from the cilia tips. Despite these differences, the simulation results in Figure 3.13 show that the trend with respect to alternation period (i.e. beat number n in the Ln-Rn pattern) is qualitatively similar.

In the simultaneous case (approximated with an alternation rate of 0.25 s), the symmetry leads to a non-mixing flow with the red and blue tracer particles remaining partitioned

on the left and right halves of the chamber according to their initial conditions, as shown in Figure 3.14b. As the alternation rate between the two vortex positions increases, the mixing time decreases until it eventually reaches a minimum. With the nominal parameters estimated from the experimental videos, the optimal pattern closely agrees with the experimental results: the best performance is found for L2-R2 or L3-R3 patterns, shown in Figure 3.14c-d, which have similar mixing performance. Additionally, the slopes of the experimental curve in Figure 3.13 seem to be matched by the simulations. However, the values of the mixing times are generally not well-predicted, with the simulated times substantially shorter than the experimental mixing times (except for the simultaneous case, in which the perfect symmetry dominates the simulation results). This is likely due to the vortices forming and vanishing instantaneously in the simulations, whereas it takes time for the vortices to build and decay in the experiment. Additionally, since the vortices have not fully stopped when the actuation switches, the newly-formed vortex must work against the vortex already present.

The simulations can also be used to predict the effect of parameter variation on mixing performance. Based on the results of the original blinking vortex theory[6], it is expected that the spacing between the vortices is a key parameter for the overall mixing performance. Figure 3.15 shows the results of perturbing the distance between the vortices by 10%. The data indicate that moving the vortices closer together would shorten the mixing time and favor a shorter beat pattern; however, since only integer values of the beat number n are feasible for the real system, the best performance would still be expected to be at the L2-R2 pattern. Conversely, if the vortices were spaced farther apart, the mixing time would be longer and longer beating patterns would expected to be optimal.

3.5.5 Achieved mixing rate

The mixing rate of the device presented here is competitive with existing micromixers on a time-per-volume basis. Although the scale of this device, targeted toward the DNA microarray application, is large compared to typical micromixers [23], the achieved mixing time of

0.026 s/µL (3 mL in 77 s) is in the range of 0.025–700 s/µL reported for prior micromixers[23].

Existing work on mixing for DNA microarrays uses substantially larger volumes than micromixers for other tasks and microarray protocols typically call for long mixing times to improve reproducibility. While a typical micromixer volume is less than 1 µL[23], commercial rotating-bubble mixers[67] use mixing chamber volumes on the order of 60 µL; commercial chaotic advection mixers[26] use 350 µL, of which 290 µL is non-reactive since it is in a mixing loop; and experimental devices use chamber volumes in the tens of microliters[20, 76]. These devices report hybridization times from minutes [20, 76] to hours [26, 67]; the mixing time of 77 s for the present work compares favorably.

Since the present work used ink rather than DNA molecules, there may be other factors such as particle size and reaction time that influence the performance of the device presented here, and hybridization experiments are necessary to confirm the performance estimates and applicability to DNA microarrays. However, prior work by McQuain et al. [51] characterized a mixing device using ink mixing and subsequently tested the hybridization rates; the findings indicate that the ink mixing took roughly 16 minutes and the DNA hybridization reaction saturated after about 10 minutes. These results indicate that the rate of mixing of ink may predict the rate of microarray hybridization.

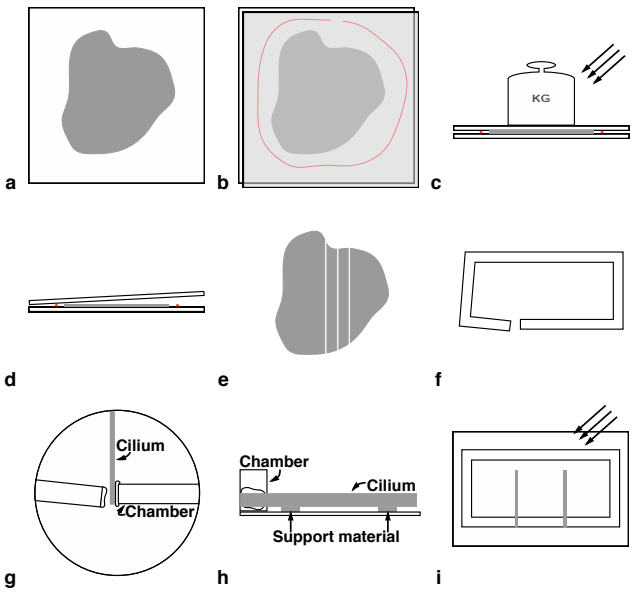


Figure 3.3: Schematic of the cilia fabrication process. **a** The Fe-PDMS mixture is poured onto a glass plate and bubbles removed in vacuum. **b** Narrow-gauge wire is used to create a constant-width spacer. A second glass plate is placed on top of the Fe-PDMS. **c** A weight is used to compress the Fe-PDMS sheet to ensure uniform thickness, and the assembly is placed in an oven to cure. **d** The top glass is pried away after curing. **e** The cured Fe-PDMS sheet is sliced to make cilia. **f** A rectangular PDMS chamber is cut where each cilium is to be mounted. **g** A drop of uncured PDMS is used on each side of the gap to glue the cilium in place. **h** Excess Fe-PDMS is used as support material (removed after curing) to create a gap between the cilium and the slide and ensure the cilium will be parallel to the slide. **i** A final curing step completes the fabrication of the cilia system

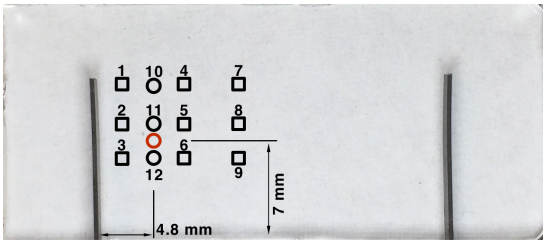


Figure 3.4: Several locations were considered for the magnet (specifically, the position of the magnet when it is closest to the glass slide). A first set of trials is depicted by squares. The placements farthest from the cilium (7-9) were ineffective, so a second set of trials was performed (shown as circles). Finally, a placement was chosen (red circle) as discussed in Section 3.2.4

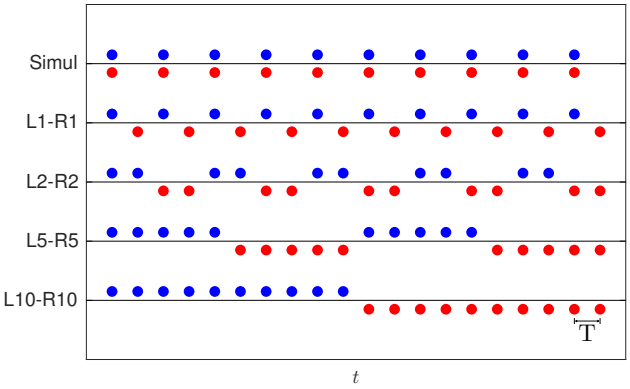


Figure 3.5: Comparison of the beating patterns tested in the experiments of this chapter. The top pattern represents the simultaneous case. The blue and red dots each represent one beat of the left and right cilia, respectively

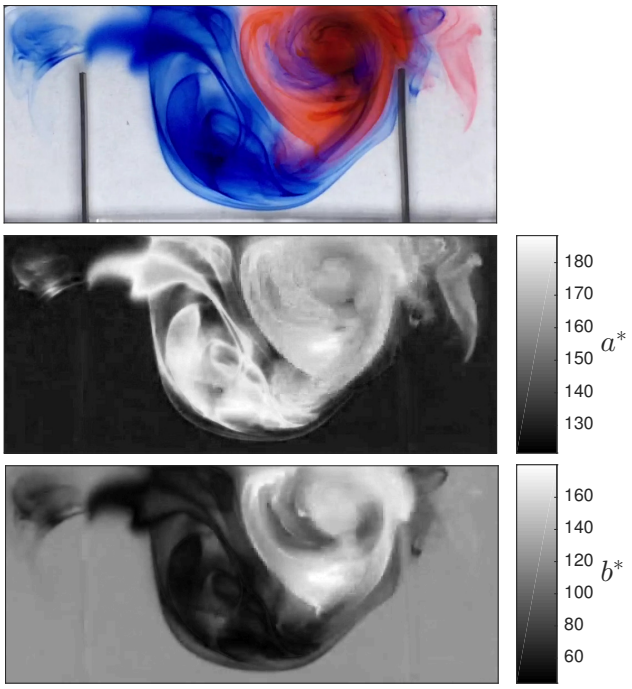


Figure 3.6: Comparison of the red-green a^* channel (middle) and yellow-blue b^* channel (bottom) for assessing the color variation in the image. The a^* channel distinguishes between areas with and without ink, but draws little distinction between the red and blue areas. The b^* channel distinguishes between the red ink, blue ink, and no-ink areas

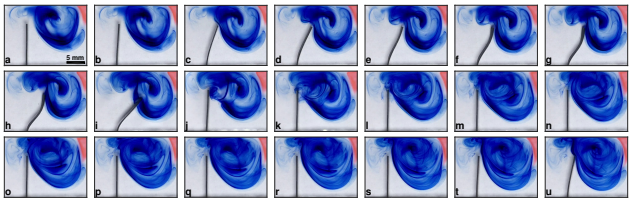


Figure 3.7: Sequence of images showing one beat of the left cilium to illustrate the flow generation mechanism. This sequence shows the third beat of an L10-R10 actuation pattern, and the time between images is two frames (0.07 s). **a-d** As the cilium deflects, fluid is drawn into the expanding area to its left. **e-h** In these four images, the cilium touches the glass slide due to magnetic forces normal to the slide. The resulting friction forces cause the cilium to bend. **i-j** When the magnet moves too far from the cilium to overcome elastic forces, the cilium straightens and snaps back to its undeflected configuration. The snapback takes about 2 frames, or 0.07 s. **k-t** In these images, clockwise rotation of the flow to the right of the cilium can be observed. **u** Here the cilium has begun to deflect once more

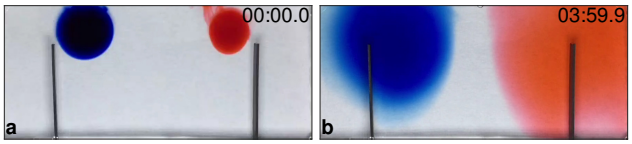


Figure 3.8: Photos showing diffusion in the cilia device. **a** Initial ink distribution. **b** After four minutes, no mixing has occurred; $c_v(b^*) = 0.33$.

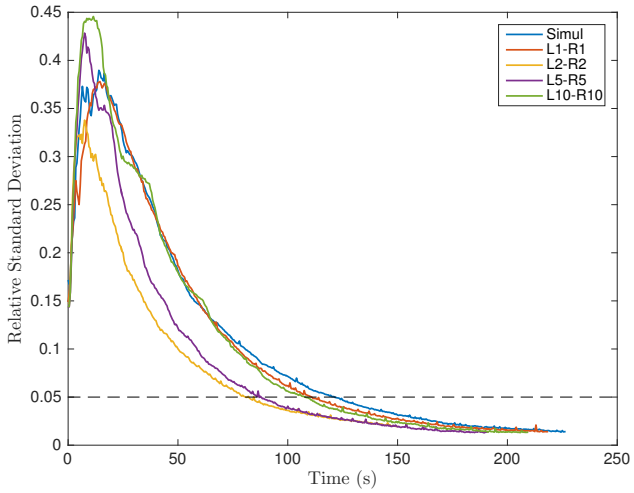


Figure 3.9: Degree of mixing, as quantified using the relative standard deviation $c_v(b^*)$, plotted against time for representative experimental runs for each beat pattern; the data shown are those with mixing times close to the average for each actuation pattern given in Table 3.2. Lower values of $c_v(b^*)$ indicate better-mixed fluid in the chamber. The dashed line indicates the threshold for the relative standard deviation, $c_v(b^*)$, of 0.05, below which the chamber is considered mixed

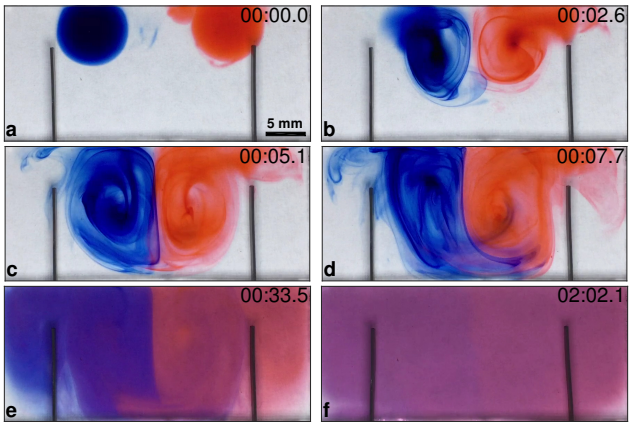


Figure 3.10: Experimental video images with the simultaneous beat pattern (see Figure 3.5). Timestamps (mm:ss) are shown in the top corner of each image. **a-d** Initial distribution of ink and result after each of the first three cycles ($\frac{t}{T} = 0, 2, 4, 6$). A boundary is formed between the right and left sides due to symmetry (**c**); however, unsteady processes sometimes break symmetry and promote mixing (**d**). **e** As the experiment continues, the boundary remains and mixing largely occurs when ink moves from one side to the other along the bottom boundary. **f** Image with $c_v(b^*) = 0.05$

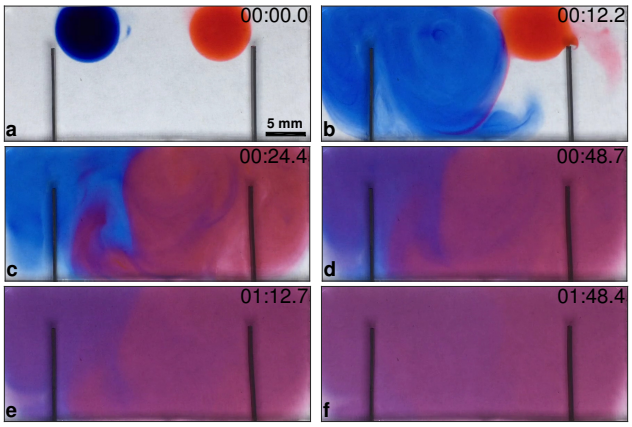


Figure 3.11: Experimental video images with the L10-R10 beat pattern (see Figure 3.5). Timestamps (mm:ss) are shown in the top corner of each image. **a** Initial distribution of ink. **b** After half a cycle ($\frac{t}{T} = 10$), a uniform blue region is found in the region of influence of the left cilium. **c** After a full cycle ($\frac{t}{T} = 20$), the action of the right cilium has made the right half uniform, drawing in some of the blue ink. **d-e** After two further cycles ($\frac{t}{T} = 40, 60$) it is observed that the boundary between colors is consistent after each full cycle. However, the color of the two uniform regions converges to a uniform value after each cycle. **f** After sufficient cycles, the color reaches the threshold, $c_v(b^*) = 0.05$.

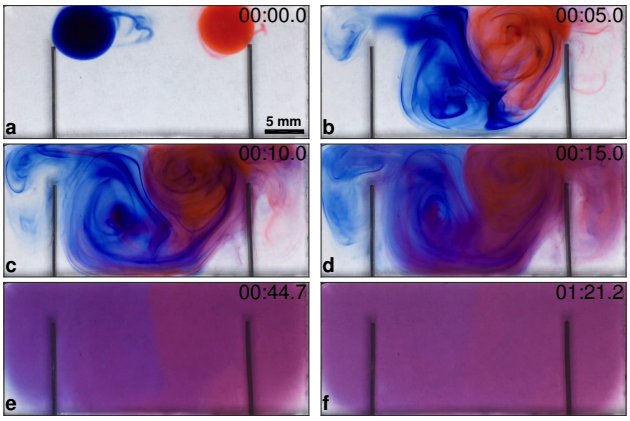


Figure 3.12: Experimental video images with the L2-R2 beat pattern (see Figure 3.5). Timestamps (mm:ss) are shown in the top corner of each image. **a-d** Images showing the initial development of the flow after each of the first three cycles ($\frac{t}{T} = 0, 4, 8, 12$). **e** Image showing the progression of the mixing flow. **f** Image with $c_v(b^*) = 0.05$

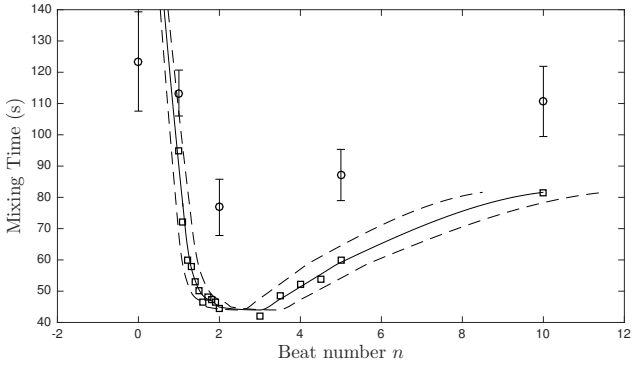


Figure 3.13: Comparison of the simulations (□) with experimental results (○). The solid line is a smoothed and interpolated fit to the simulated data; the dashed lines indicate the result of perturbing the vortex circulation Γ by $\pm 15\%$. Error bars for the experimental results indicate one standard deviation

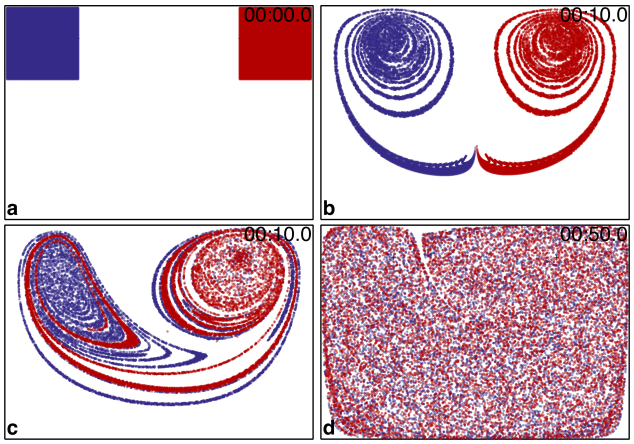


Figure 3.14: Selected images from the simulations described in Section 3.4. Timestamps (mm:ss) show elapsed time. **a** Initial placement of tracer particles. **b** Results after 10 s of the simultaneous pattern. In this case, symmetry prevents mixing. **c** Results after 10 s (two cycles) of the L2-R2 pattern, showing tracer particles moving between the two sides. **d** Results after 50 s (10 cycles) of the L2-R2 pattern, showing that the chamber is mixed.

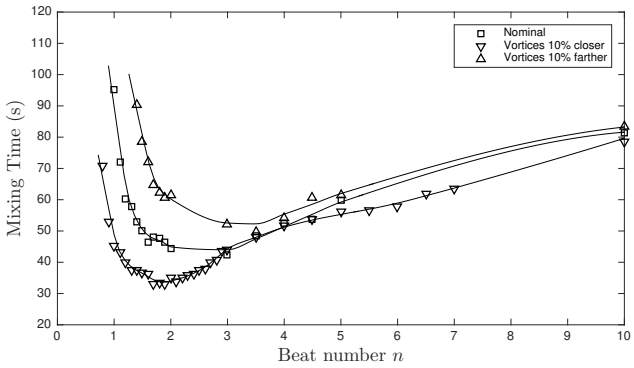


Figure 3.15: Simulated mixing trials showing the effect of perturbing the vortex position $x_1 = \mathrm{Re}(z_1)$ by $\pm 10\%$. The markers indicate simulated trials, and the solid lines represent smoothed and interpolated fits. For low beat number ($n < 1$), mixing time goes to infinity as the conditions approach the simultaneous-activation case, making computational costs prohibitive. As in the original blinking vortex theory, the vortex spacing is a key parameter for the overall mixing performance

Chapter 4

NONLINEAR MODELS FOR MAGNET PLACEMENT IN INDIVIDUALLY-ACTUATED CILIA DEVICES

4.1 Introduction

This chapter presents a model for predicting the optimal magnet placement in magnetic cilia devices that achieve individual control via localization of the driving magnetic field. In this configuration, each cilium is controlled by a magnetic field source which is limited in spatial extent, and the cilia are spaced sufficiently-far apart that the control remains uncoupled. An implementation is presented using an electromagnetic field source to attain large-deformation actuation (transverse deflections of 47% of the length). The large deformations are achieved by exploiting the nonlinear response of a flexible cantilever in a non-uniform magnetic field. However, the same non-linearities also pose a modeling challenge: the overall performance is sensitive to the location of the electromagnet and the location that produces the largest deflections is non-linearly dependent on the strength of the magnetic field. The nonlinear displacement of the cilium is predicted using a finite element model of the coupled magnetic-structural equations for static inputs at varying field strengths and magnet positions. The deflection at the model-predicted optimal placement is within 5% of the experiment-predicted optimal placement. Moreover, actuator placement using a model that does not include the nonlinearities is estimated to result in performance loss of about 50% peak deflection. This result emphasizes the importance of capturing nonlinearities in system design.

4.2 Experiments

To aid in understanding of the modeling process and the nature of the placement problem, the experimental system is described first.

4.2.1 Experimental system

The experimental system consists of a magnetic artificial cilium and an electromagnet which serves as the magnetic field source. The cilium is a cantilever made from a composite of polydimethylsiloxane (PDMS) and carbonyl iron and fabricated and installed into a PDMS chamber on a standard glass slide as discussed in Chapter 3. The magnetic field source is a custom solenoidal electromagnet wound with 28AWG magnet wire and controlled by an analog circuit. The electromagnet core was machined from a soft iron rod, resulting in an I-shaped cross-section as shown in Figure 4.1.

The current in the electromagnet coil is controlled by an input voltage signal V produced by a function generator (Rigol DG1022) and sent through an analog current amplification circuit to produce a current

$$i = \frac{V_{\text{ref}}}{R_g} \tag{4.1}$$

where R_g is the resistance of the gain resistor, as shown in Figure 4.2. The circuit was designed with a nominal value of $R_g = 1\,\Omega$. The coil current was estimated by measuring the voltage across the gain resistor using a microcontroller (Pololu Wixel) with a 12-bit analog-to-digital converter at a sample rate of 1 kHz.

4.2.2 Experimental procedure

In each experiment, a desired input voltage signal V was programmed into the function generator and the resulting response of the cilium was recorded to digital video using a smartphone recording at a frame rate of 60 Hz for static experiments or 240 Hz for dynamic experiments. To synchronize the recorded coil current i with the video, a light-emitting diode (LED) was placed such that it is visible on the video, and the microcontroller used for sampling the coil current was also programmed to toggle the LED (on/off) to indicate the start and end of the experiment.

Image analysis was used to automatically estimate the response of the cilium frame-by-frame for each video recording. First, the videos were manually cropped to focus on the

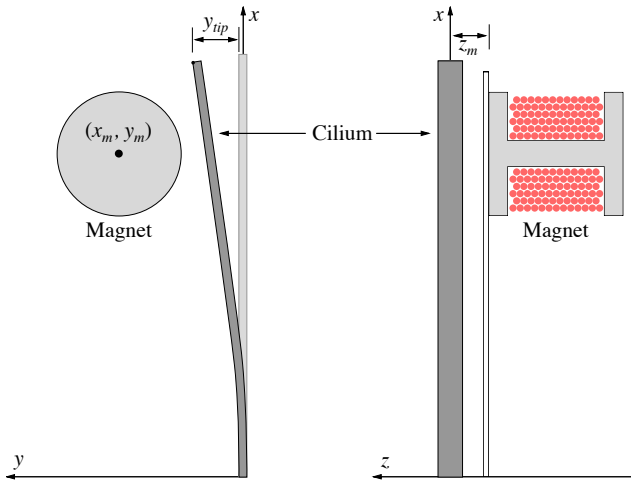


Figure 4.1: Schematic of the experimental system showing top and side views. The artificial cilium is a flexible magnetic cantilever that is deflected due to a field produced by the electromagnet located at $\mathbf{X}_m = (x_m,\ y_m,\ z_m)$.

region of interest, and the positions of the magnet and cilium base, $\mathbf{X}_m$ and $\mathbf{X}_b$, were manually labeled for each video. Then each frame was converted to grayscale as shown in Figure 4.3a and automatically processed as shown in Figure 4.3: contrast enhancement (Figure 4.3b), thresholding (Figure 4.3c), smoothing of the resulting significant features (Figure 4.3d), and selection of the region that includes the labeled base position (Figure 4.3e). However, in some cases additional pixels were misidentified as belonging to the cilium, typically due to low contrast between the cilium and shadows on or around the magnet. An adaptive Hough transform[38, 49] was used to estimate the curvature of the cantilever in the presence of error in the image analysis, shown in Figure 4.3f.

4.3 Modeling

Consider a magnetizable rectangular cantilever (i.e., artificial cilium) of known dimensions whose base position $\mathbf{X}_b$ is at the origin and a magnetic field source at position $\mathbf{X}_m \equiv (x_m,\ y_m,\ z_m)$. It is assumed that the magnetic field source produces a magnetic field $\mathbf{H}_c$

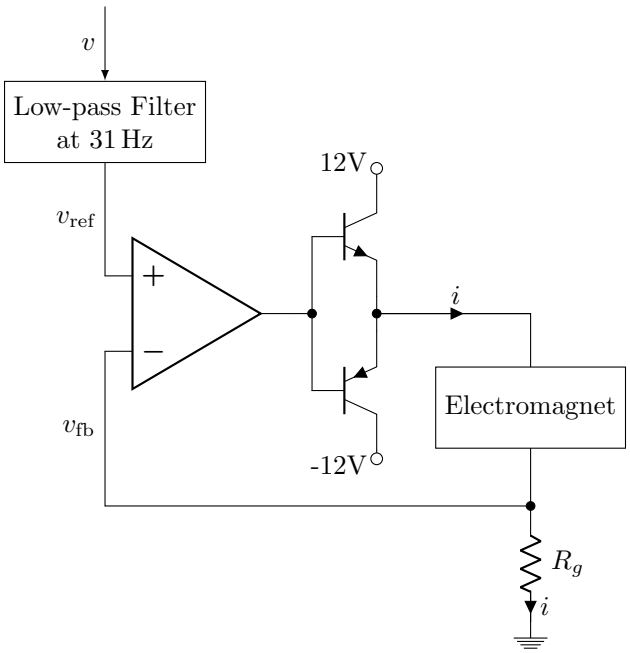


Figure 4.2: Control of the current i through the electromagnet coil is achieved using an analog feedback loop (push-pull amplifier) implemented using op-amps. Measuring the voltage $V_{\text{fb}} = iR_g$ across the gain resistor provides an estimate of the coil current. The voltage signal V is prefiltered with an active low-pass filter with 31 Hz cutoff frequency.

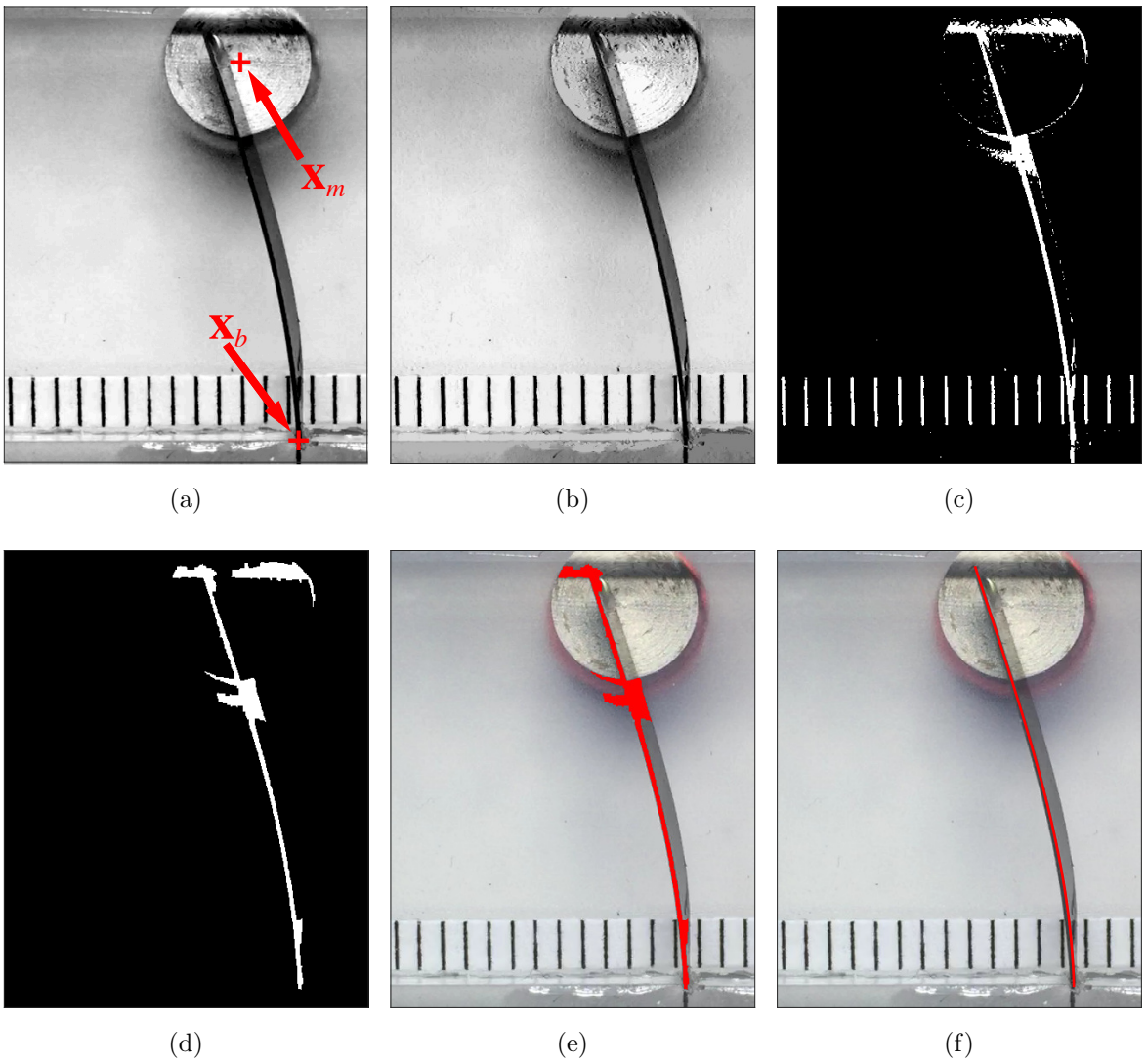


Figure 4.3: Image processing sequence for a single frame of video. (a) Greylevel image showing the manually-identified magnet and base positions $\mathbf{X}_m$ and $\mathbf{X}_b$. (b) The greylevel image is processed to enhance contrast. (c) Thresholding. (d) Large connected regions are selected and smoothed. (e) The region touching the manually-labeled base position is retained, shown here overlaid on the original image. Due to low contrast, the algorithm misidentifies shadows as part of the cilium. (f) Shape identification via adaptive Hough transform[38, 49].

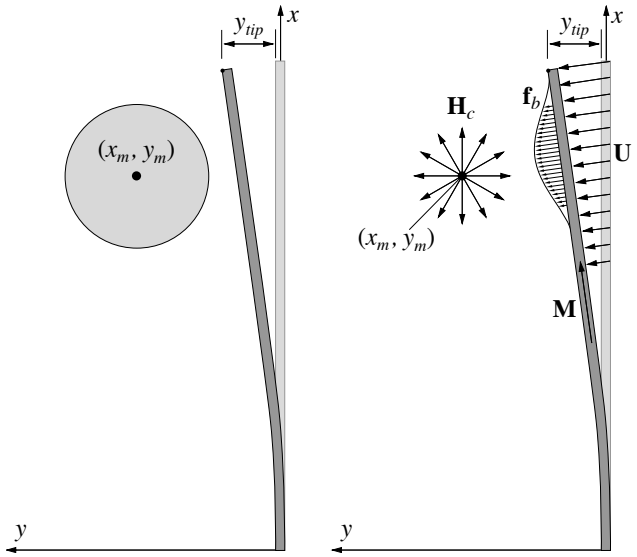


Figure 4.4: Top view of the model representation (right) of the experimental system (left). A field source at (x_m, y_m) produces a field $\mathbf{H}_c$ which induces a magnetization $\mathbf{M}$ in a rectangular cantilever (i.e., artificial cilium). The interaction of the magnetization $\mathbf{M}$ with the applied field $\mathbf{H}_c$ generates a magnetic body force density $\mathbf{f}_b$ on the cantilever, resulting in a deflection field $\mathbf{U}$ and corresponding tip deflection y_{tip}.

that is known everywhere in the model domain and whose amplitude can be controlled by the input voltage V. The modeling problem, depicted in Figure 4.4, is to predict the cantilever's tip deflection y_{tip} under the influence of the generated magnetic field $\mathbf{H}_c$ as a function of magnet position $\mathbf{X}_m$ and input voltage V, i.e.

$$(\mathbf{X}_m,\ V) \longmapsto y_{\text{tip}} \tag{4.2}$$

which will be solved using a finite element approach to find the relation between the deflection y_{tip} and voltage V at various magnet positions $\mathbf{X}_m$. The system is modeled for the static case, since slow trajectories will be used to avoid damaging samples in biological applications. This is also similar to the beating of biological cilia, which operate at oscillatory Reynolds number $\text{Re} \sim 10^{-3}$, where

$$\text{Re} = \frac{\omega L^2}{\nu}. \tag{4.3}$$

Here ω is the beating frequency in radians per second, L is the length of the cilium, and ν is the kinematic viscosity of the medium[14]. Since the experimental cilium has length $L \sim 1 \times 10^{-2}\,\text{m}$, matching the Reynolds number of biological cilia would result in a beating frequency of $\omega \sim 1 \times 10^{-5}\,\text{rad/s}$ when operated in water (kinematic viscosity $\nu \sim 1 \times 10^{-6}\,\text{m}^2/\text{s}$). The solution process involves four main modeling components, as summarized below.

The magnetic field $\mathbf{H}_c(x)$ produced by the magnetic field source at an arbitrary point $\mathbf{x}$ is computed from the input voltage V and magnet position $\mathbf{X}_m$, i.e.

$$(\mathbf{X}_m,\ V) \longmapsto \mathbf{H}_c(\mathbf{x}). \tag{4.4}$$

Application of a magnetic field induces a magnetization $\mathbf{M}(\mathbf{x})$ at each point $\mathbf{x}$, which can be computed from the applied field $\mathbf{H}_c$:

$$\mathbf{H}_c(\mathbf{x}) \longmapsto \mathbf{M}(\mathbf{x}). \tag{4.5}$$

A magnetized body in a magnetic field is subject to a magnetic body force density $\mathbf{f}_b$, which is a function of the magnetization and the applied field,

$$(\mathbf{H}_c(\mathbf{x}),\ \mathbf{M}(\mathbf{x})) \longmapsto \mathbf{f}_b(\mathbf{x}). \tag{4.6}$$

Finally, applying the magnetic body force density $\mathbf{f}_b$ to a nonlinear structural model allows for prediction of the cantilever deflection field $\mathbf{U}$,

$$\mathbf{f}_b(\mathbf{x}) \longmapsto \mathbf{U}(\mathbf{x}), \tag{4.7}$$

from which the tip deflection in the y direction, y_{tip}, can be read out as one of the nodal deflections,

$$\mathbf{U}(\mathbf{x}) \longmapsto y_{\text{tip}}. \tag{4.8}$$

Equations 4.4 to 4.7 represent the four major components of the model: (1) Computation of the field generated by the electromagnet as in Equation 4.4; (2) a magnetic potential model to compute the magnetization as in Equation 4.5; (3) a magnetic force model to compute the load on the structure as in Equation 4.6; and (4) a nonlinear structural model to compute the deformation of the structure as in Equation 4.7. In the following sections, the modeling approach for each component is described in further detail.

4.3.1 Field generated by the electromagnet

To estimate the applied field $\mathbf{H}_c(\mathbf{x})$, an axisymmetric model of the experimental electromagnet was developed in the open-source software Finite Element Method Magnetics (FEMM). The output of the FEMM model is the radial and axial field components of the applied field $\mathbf{H}_c$, which gives a prediction of the applied field as

$$\mathbf{H}_c(r',\ z') = A(r',\ z'), \tag{4.9}$$

where $A(r',\ z')$ describes the spatial variation of the field, computed when the coil is subject to a nominal current of $i = 1\,\text{A}$, the magnet core was modeled using the "Pure Iron" preset,

and the coil was modeled as 127 turns of 28 AWG magnet wire. The coordinates of the FEMM model are converted to the global coordinates as

$$(r',\ z') = (\|(x - x_m,\ y - y_m)\|,\ z - z_m). \tag{4.10}$$

Since the control variable is the input voltage V, Equation 4.9 is restated as

$$\mathbf{H}_c(\mathbf{x}) = k_c\ V\ A(\|(x - x_m,\ y - y_m)\|,\ z - z_m), \tag{4.11}$$

where it has been assumed that the applied field $\mathbf{H}_c$ is linear with respect to the input voltage V, the fitting constant k_c is used to match the model field to the measured field of the experimental electromagnet, and the field profile A has been translated from the magnet-centered frame to the global coordinates $(x,\ y,\ z)$.

4.3.2 Magnetic potential model

Next, the magnetization $\mathbf{M}$ in the structure due to the applied field $\mathbf{H}_c$ must be computed. The magnetization is related to the total field $\mathbf{H}$ in the structure as

$$\mathbf{M} = \chi\mathbf{H} = \chi(\mathbf{H}_c + \mathbf{H}_m) \tag{4.12}$$

where $\mathbf{H}_m$ is the portion of the field that is caused by the magnetization of the structure and the magnetic susceptibility χ is a material constitutive parameter that in general may vary through the structure or depend on the applied field magnitude $\|\mathbf{H}\|$. Since the volume considered (i.e. the structure) contains no currents, the magnetic field $\mathbf{H}$ (as well as the components $\mathbf{H}_m$ and $\mathbf{H}_c$) is curl-free and can be written as the gradient of a scalar potential function[66]

$$\varphi = \phi_c + \phi_m, \tag{4.13}$$

where ϕ_c is the current-induced potential corresponding to the applied field $\mathbf{H}_c$, ϕ_m is the magnetization-induced potential corresponding to the magnetization-induced field $\mathbf{H}_m$, and their sum is the total potential φ. Then

$$\mathbf{H} = -\nabla\varphi = -\nabla\phi_c - \nabla\phi_m \tag{4.14}$$

and the magnetization problem is solved if the total potential φ can be found everywhere in the structure.

Since the applied field $\mathbf{H}_c$ is known, a current-induced potential ϕ_c that satisfies the condition $\mathbf{H}_c = -\nabla\phi_c$ can be computed by the integration

$$\phi_c(\mathbf{x}) = -\int_{\mathbf{x}_0}^{\mathbf{x}} \mathbf{H}_c \cdot d\mathbf{x}, \tag{4.15}$$

where $\mathbf{x}_0$ is an arbitrary point taken to have zero potential, which was selected as the center of the top of the electromagnet.

The second term in Equation 4.13, the magnetization-induced potential ϕ_m, can be expressed as an integral over the whole volume[16],

$$\phi_m(\mathbf{x}) = \frac{1}{4\pi}\int_{V'} \frac{\mathbf{x}-\mathbf{x}'}{|\mathbf{x}-\mathbf{x}'|^3} \cdot \mathbf{M}(\mathbf{x}')\, dV', \tag{4.16}$$

which states that the potential ϕ_m at a point $\mathbf{x}$ depends on the magnetization $\mathbf{M}$ at all points $\mathbf{x}'$ in the structure. In the finite element discretization, the potential φ at a general point $\mathbf{x}$ is related to the nodal total potential $\hat{\varphi}$ via the shape functions N as

$$\varphi(\mathbf{x}) = N(\mathbf{x})\hat{\varphi}, \tag{4.17}$$

and therefore, using Equation 4.14,

$$\mathbf{M} = -\chi\nabla N\hat{\varphi}, \tag{4.18}$$

which, when combined with Equation 4.16, results in

$$\phi_m(\mathbf{x}) = \left(-\frac{1}{4\pi}\int_{V'} \left(\frac{\mathbf{x}-\mathbf{x}'}{|\mathbf{x}-\mathbf{x}'|^3}\right)^T \chi\nabla N(\mathbf{x}')\, dV'\right)\hat{\varphi} \tag{4.19}$$

and the nodal magnetization-induced potential $\hat{\phi}_m$ can be computed by evaluating the integral in Equation 4.19 at the nodal positions $\hat{\mathbf{x}}$, resulting in

$$\hat{\phi}_m = \phi_m(\hat{\mathbf{x}}) = \mathbf{K}_{\mathbf{M}}\hat{\varphi}. \tag{4.20}$$

Finally, the nodal total potential $\hat{\varphi}$ can be computed by combining Equations 4.13 and 4.20, resulting in

$$(I - \mathbf{K_M})\hat{\varphi} = \hat{\phi}_c, \tag{4.21}$$

and the magnetization $\mathbf{M}$ is computed from Equations 4.18 and 4.21.

Remark 1 *Equation 4.21 is similar to a structural finite element equation with forcing term ϕ_c and stiffness $(I - \mathbf{K_M})$. However, unlike a structural stiffness matrix, the magnetic self-coupling $\mathbf{K_M}$ is a full matrix, so solving for the total potential φ is computationally expensive if the mesh is very fine. This issue is addressed by choosing high-order elements that are accurate even when a coarse mesh is used.*

Remark 2 *When computing an integral on a region in which the integrand has a singularity, numerical integration procedures may lose accuracy. This is observed when computing the magnetic coupling matrix $\mathbf{K_M}$ in Equation 4.19; when the distance between the source point $\mathbf{x}$ and observer point $\mathbf{x}'$ tends to zero, the fraction in the integrand tends to infinity. However, the integral is well-defined in three-dimensional analysis because the $\mathbf{x}^{-2}$ singularity has lower power than the dimensionality of the integral [53]. The inaccuracy due to singular points can be mitigated by ensuring that the singular point is at a corner of the integration domain. Since the singular points are at the nodes, this property can be achieved by subdividing the integrals where singularities occur, i.e. when the source point $\mathbf{x}$ is in the element over which the integral in Equation 4.19 is being evaluated, as shown in Figure 4.5. This subdivision also allows the singular integrals, which are less accurate, to be integrated at a higher order than the non-singular integrals.*

4.3.3 Magnetic force model

The magnetic body force density $\mathbf{f}_b$ is modeled using the Kelvin force[27],

$$\mathbf{f}_b(\mathbf{x}) = \mu_0(\mathbf{M}(\mathbf{x}) \cdot \nabla)\mathbf{H}_c(\mathbf{x}), \tag{4.22}$$

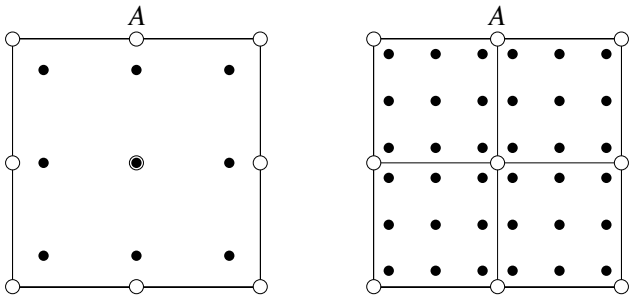


Figure 4.5: Mesh subdivision for singular integrals. Left: Nodes (○) and quadrature points (•), i.e. the points at which the singular integrand is to be evaluated, in standard 3×3 gauss quadrature. When evaluating the integral in Equation 4.19, singular points occur at the nodes, reducing accuracy because it violates smoothness assumptions (see Remark 2). Right: Subdivision of the element to lower-order elements ensures that the singularity occurs at the corner, which mitigates the error. The increased number of quadrature points also improves accuracy.

which can be derived via the principle of virtual work [16]. The equivalent nodal forces are found by integrating over the deflected structure. Since the applied field gradient $\nabla \mathbf{H}_c$ varies spatially, the magnetic body force density $\mathbf{f}_b$ depends on the structural deformation $\mathbf{U}$. The corresponding nodal force vector is computed using the finite element shape functions N,

$$\mathbf{R}(\mathbf{U}) = \int_V N^T(\mathbf{x}) \mathbf{f}_b(\mathbf{x})\, dV, \tag{4.23}$$

where the dependence on $\mathbf{U}$ is included to emphasize that the integral is taken over the volume of the deflected structure.

4.3.4 Nonlinear structural model

The final step in the finite element modeling process is, given a nodal force vector $\mathbf{R}$ due to the magnetic field at an input voltage V, to compute the corresponding deflection $\mathbf{U}$, i.e. the shape of the structure for which the externally-applied load $\mathbf{R}$ is balanced pointwise by

internal elastic forces $\mathbf{F}$. The result of the solution procedure will be the deflection $\mathbf{U}$ which satisfies the equilibrium condition

$$\mathbf{R}(V, \mathbf{U}) - \mathbf{F}(\mathbf{U}) = \mathbf{0}. \tag{4.24}$$

The deflection $\mathbf{U}$ is related to the position $\mathbf{x}$ of points in the structure by

$$\mathbf{x} = \mathbf{x_0} + \mathbf{U}, \tag{4.25}$$

where $\mathbf{x_0}$ is the initial position of the points in the structure. The internal forces $\mathbf{F}$ are computed using a Total Lagrangian formulation of continuum mechanics, which takes the undeflected configuration as the reference state and uses the second Piola-Kirchoff stress and Green-Lagrange strain to model the relationship between the deflection $\mathbf{U}$ and the forces $\mathbf{F}$—see chapter 6 of Ref. 8 for details.

4.3.5 Incremental-iterative solution procedure

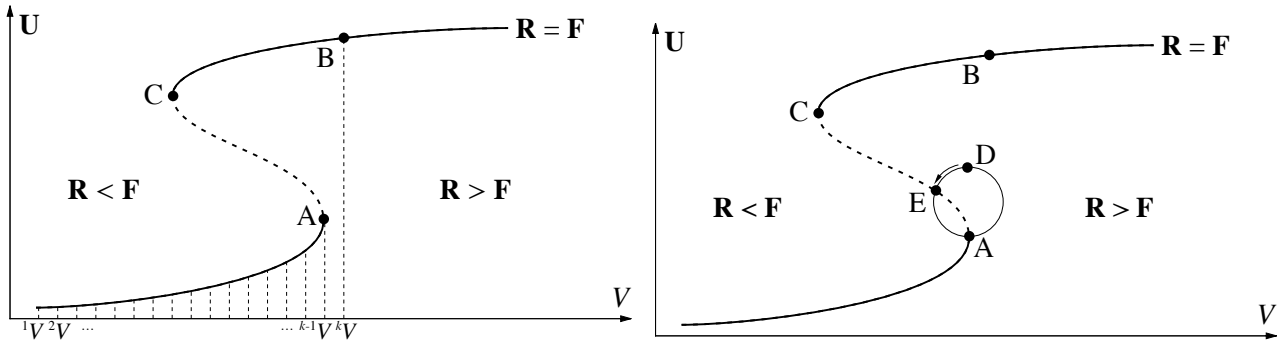
Because the equilibrium equation Equation 4.24 depends implicitly (and nonlinearly) on the deflection $\mathbf{U}$, the deflection solution $\mathbf{U}_d$ at a given input voltage V_d is solved using an incremental-iterative procedure. In this procedure, the desired solution $(V_d, \mathbf{U}_d)$ is obtained as the last of a sequence of equilibrium states $({}^kV^*,\ {}^k\mathbf{U}^*)$ that satisfy Equation 4.24. Given an equilibrium solution $({}^{k-1}V^*,\ {}^{k-1}\mathbf{U}^*)$, the next equilibrium $({}^kV^*,\ {}^k\mathbf{U}^*)$ is found iteratively, which is initialized as

$$({}^kV^{(0)},\ {}^k\mathbf{U}^{(0)}) = ({}^{k-1}V^*,\ {}^{k-1}\mathbf{U}^*) + ({}^kV_{\text{inc}},\ {}^k\mathbf{U}_{\text{inc}}), \tag{4.26}$$

with iterative corrections

$$({}^kV^{(i+1)},\ {}^k\mathbf{U}^{(i+1)}) = ({}^kV^{(i)},\ {}^k\mathbf{U}^{(i)}) + ({}^k\Delta V^{(i)},\ {}^k\Delta\mathbf{U}^{(i)}). \tag{4.27}$$

Here, as in the notation of [8], ${}^k\mathbf{U}^{(i)}$ refers to the i-th iterative trial deflection in the k-th increment and an asterisk denotes an equilibrium solution (i.e. a solution to Equation 4.24). The selection of the iterative correction $({}^k\Delta V^{(i)},\ {}^k\Delta\mathbf{U}^{(i)})$ in Equation 4.27 is described later in this section.



(a) Load control (b) Arclength method

Figure 4.6: Numerical solution in the presence of instability. At point A, the stiffness matrix is singular. The load control method (top) is similar to the experimental procedure, but fails to traverse the large jump in response from point A to point B, leading to divergence or slow convergence. In the arclength method (bottom), a spherical constraint is used so that the solutions follow the unstable path from point A to point C to point B without discontinuity.

Remark 3 *A well-known incremental-iterative method is the "load control" method, in which the deflection increment* ${}^{k}\mathbf{U}_{inc} = 0$ *and the voltage increment* ${}^{k}V_{inc}$ *is set to a predetermined value, in Equation 4.26. Additionally, the iterative correction is restricted such that* ${}^{k}\Delta V^{(i-1)} = 0$ *in Equation 4.27. However, as shown in Figure 4.6, load control can fail to converge when there are reversals in the equilibrium path, since the post-reversal solution (point B) is far from the prior solution at point A even if the size of the voltage increment* ${}^{k}V_{inc}$ *is arbitrarily small. Additionally, the steep slope at point A can result in divergence of the computational procedures.*

When load reversals are present, as they are in this system, the load control procedure fails because the response is discontinuous at a critical voltage (point A in Figure 4.6). However, there is a continuous path in (V, $\mathbf{U}$) space that connects the state at the critical voltage (V_A, $\mathbf{U}_A$) to the subsequent solution at (V_B, $\mathbf{U}_B$). This reversed path through point C can be followed by letting the input voltage V vary and solving simultaneously for

both the deflection $\mathbf{U}$ and the input voltage V in the incremental-iterative procedure. This requires an additional constraint equation (since an additional variable is being solved for). In arclength methods, the iterative solutions $({}^kV^{(i)},\ {}^k\mathbf{U}^{(i)})$ (for different i) are constrained to lie on the surface of a hypersphere centered on a point $({}^kV_+,\ {}^k\mathbf{U}_+)$, for example, from D to E on the circle representing the hypersphere in Figure 4.6b. Formally,

$$({}^k\mathbf{U}^{(i)} - {}^k\mathbf{U}_+)^2 + k_a({}^kV^{(i)} - {}^kV_+)^2 = L^2, \tag{4.28}$$

where k_a is chosen so that the units and numerical size of the variables are similar, and the arclength L determines the iteration step size through the size of the hypersphere. Since the hypersphere is fixed during the iterative corrections, divergence is avoided when solving for the next equilibrium state $({}^kV^*,\ {}^k\mathbf{U}^*)$.

Remark 4 *The constraint center $(V_+, \mathbf{U}_+)$ is commonly chosen to be the most recent equilibrium point $(k-1)$. However, this choice results in two possible intersections between the equilibrium path and the constraint surface: the next equilibrium (k) and the previous equilibrium $(k-2)$. For this work, the method proposed by Ref. 64 was used, in which the constraint center is shifted along the initial tangent such that the most recent equilibrium point $(k-1)$ is on the constraint surface,*

$$({}^kV_+,\ {}^k\mathbf{U}_+) = \left(\frac{1}{2}\,{}^kV_{inc},\ \frac{1}{2}\,{}^k\mathbf{U}_{inc}\right), \tag{4.29}$$

as shown in Figure 4.7.

The equations for the iterative corrections are as follows. A Taylor expansion (with only the first-order-derivative terms) of Equation 4.24 about $({}^kV^{(i)},\ {}^k\mathbf{U}^{(i)})$ yields

$$\begin{aligned}\mathbf{R}({}^kV^{(i+1)},\ {}^k\mathbf{U}^{(i+1)}) - \mathbf{F}({}^k\mathbf{U}^{(i+1)}) \approx\ & \mathbf{R}({}^kV^{(i)},\ {}^k\mathbf{U}^{(i)}) \\ & - \mathbf{F}({}^k\mathbf{U}^{(i)}) - {}^k\mathbf{K}^{(i)}\,{}^k\Delta\mathbf{U}^{(i)} - {}^k\mathbf{Q}^{(i)}\,{}^k\Delta V^{(i)}\end{aligned} \tag{4.30}$$

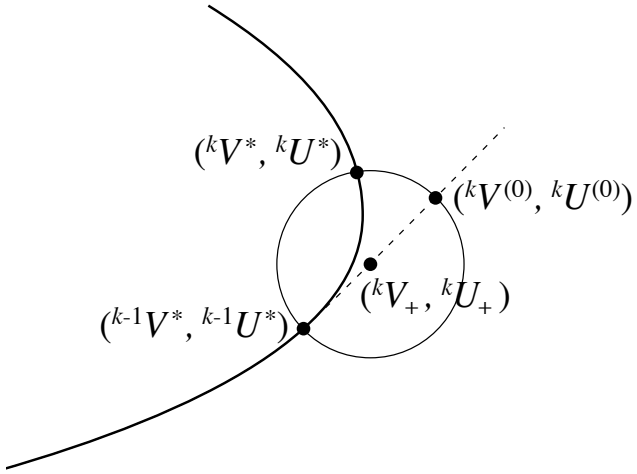


Figure 4.7: Detail of the arclength incrementation procedure. Starting from a previous solution at $({}^{k-1}V^{*},\ {}^{k-1}\mathbf{U}^{*})$, the center of the constraint surface at $({}^{k}V_{+},\ {}^{k}\mathbf{U}_{+})$ is found by projecting along the tangent to the equilibrium path. Then the first trial solution, $({}^{k}V^{(0)},\ {}^{k}\mathbf{U}^{(0)})$, is the intersection of the tangent and the circle. The iterative procedure given by Equations 4.28 and 4.33 follows the circle from $({}^{k}V^{(0)},\ {}^{k}\mathbf{U}^{(0)})$ to the equilibrium at $({}^{k}V^{*},\ {}^{k}\mathbf{U}^{*})$.

where ${}^k\Delta\mathbf{U}^{(i)} = {}^k\mathbf{U}^{(i+1)} - {}^k\mathbf{U}^{(i)}$, ${}^k\Delta V^{(i)} = {}^kV^{(i+1)} - {}^kV^{(i)}$, and

$$ {}^k\mathbf{K}^{(i)} = -\left.\frac{\partial(\mathbf{R}-\mathbf{F})}{\partial\mathbf{U}}\right|_{{}^k\mathbf{U}^{(i)},{}^kV^{(i)}} \tag{4.31} $$

$$ \begin{aligned} {}^k\mathbf{Q}^{(i)} &= -\left.\frac{\partial(\mathbf{R}-\mathbf{F})}{\partial V}\right|_{{}^k\mathbf{U}^{(i)},{}^kV^{(i)}} \\ &= -\left.\frac{\partial\mathbf{R}}{\partial V}\right|_{{}^kV^{(i)}}. \end{aligned} \tag{4.32} $$

Ideally, if the result of the iteration is an equilibrium solution, i.e., $({}^kV^{(i+1)},\ {}^k\mathbf{U}^{(i+1)}) = ({}^kV^{*},\ {}^k\mathbf{U}^{*})$, then the left-hand-side of Equation 4.30 will be zero. Based on this, Equation 4.30 can be rearranged to estimate the required iterative increments as

$$ {}^k\mathbf{K}^{(i)}\,{}^k\Delta\mathbf{U}^{(i)} + {}^k\mathbf{Q}^{(i)}\,{}^k\Delta V^{(i)} = \mathbf{R}({}^kV^{(i)},\ {}^k\mathbf{U}^{(i)}) - \mathbf{F}({}^k\mathbf{U}^{(i)}). \tag{4.33} $$

The derivative of the finite element nodal force vector $\mathbf{R}$ with respect to the deflection $\mathbf{U}$ in Equation 4.31 is computed from Equation 4.23 as

$$ \frac{\partial\mathbf{R}}{\partial\mathbf{U}} = \int_V N^T(\mathbf{x})\frac{\partial\mathbf{f}_b}{\partial\mathbf{U}}N(\mathbf{x})\,dV, \tag{4.34} $$

where, from Equation 4.22,

$$ \frac{1}{\mu_0}\frac{\partial\mathbf{f}_b(\mathbf{x})}{\partial\mathbf{U}} = \left(\frac{\partial\mathbf{M}(\mathbf{x})}{\partial\mathbf{U}}\cdot\nabla\right)\mathbf{H}_c(\mathbf{x}) + (\mathbf{M}(\mathbf{x})\cdot\nabla)\,\frac{\partial\mathbf{H}_c(\mathbf{x})}{\partial\mathbf{U}}. \tag{4.35} $$

Remark 5 *The purpose of computing the derivative of the load vector* $\mathbf{R}$ *in Equation 4.34 is to improve convergence of the finite element solution procedure. However, since the magnetization* $\mathbf{M}$ *depends on the coupling matrix* $\mathbf{K}_\mathbf{M}$ *(Equation 4.21), which is expensive to recompute as discussed in Remark 1, computation of* $\frac{\partial\mathbf{M}}{\partial\mathbf{U}}$ *in Equation 4.35 is prohibitively time intensive. Therefore the first term in Equation 4.34 is neglected, resulting in*

$$ \frac{\partial\mathbf{R}}{\partial\mathbf{U}} \approx \mu_0\int_V N^T(\mathbf{x})(\mathbf{M}(\mathbf{x})\cdot\nabla)(\nabla\mathbf{H}_c(\mathbf{x}))N(\mathbf{x})\,dV. \tag{4.36} $$

The error introduced by this approximation is also corrected through the iterative process.

Equations 4.28 and 4.33 constitute two equations which can be solved for the two unknown increments $({}^k\Delta V^{(i)},\ {}^k\Delta\mathbf{U}^{(i)})$ in Equation 4.27. For further details of the arclength incrementation procedure, see Ref. 64.

4.3.6 Model implementation details

Discretization and integration

The structure was discretized by a uniform mesh of $30 \times 1 \times 4$ 27-noded isoparametric brick elements. While a $3 \times 3 \times 3$ quadrature rule is standard for these elements for linear problems[8], the integrals were computed using a $4 \times 4 \times 4$ Gaussian quadrature rule for both the magnetic and the structural sub-problems; this was done to improve accuracy without refining the mesh, which would have increased computational cost (see Remark 1).

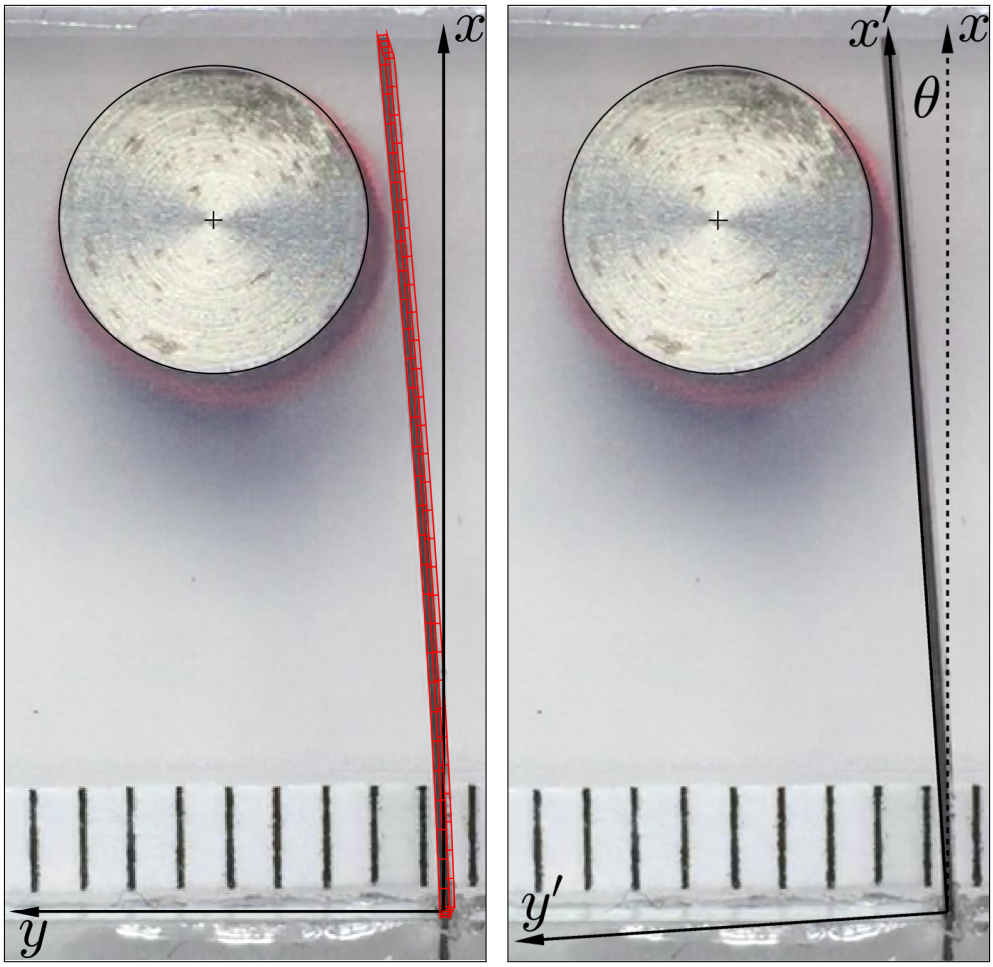
Initial deflection of the cantilever

While the model problem shown in Figure 4.4 assumed an initially-straight rectangular cantilever, the initial conditions for the experiments included a slight deflection of the artificial cilium, as shown in Figure 4.8, due perhaps to the gravity load or residual stresses from hysteretic material properties. To account for this initial deflection, two different approaches were used depending on the data being analyzed:

- When using the model to match a specific experimental trial, i.e. when fitting the model and evaluating the fit, the initial nodal positions $\hat{\mathbf{x}}_0$ are matched to the estimated curvature from image analysis was applied to the finite element mesh, treating the curvature as occurring only in the xy-plane; an example is shown in Figure 4.8a.

- When evaluating the overall placement trends, the experimental results must be corrected for varying initial conditions. In this case, the experimental magnet placements are measured in a coordinate frame (x', y') aligned with the initial angle θ of the cilium and the model initial conditions are zero. This is shown in Figure 4.8b.

In both cases, a slight rotation θ_b of the cilium about the x-axis was included to account for a twist of the experimental cilium observed from the video frames. This angle was visually estimated from the video and was assumed to be constant across all trials.



(a) (b)

Figure 4.8: Handling of initial conditions when the experimental cilium is initially deflected. (a) A finite element mesh overlaid on a photograph of the initial deflection, for comparing the model to a single experiment. (b) The rotated coordinates $(x',\ y')$ which are used to compare multiple different experiments; each placement is measured in a frame rotated by the cilium initial angle θ for that experiment as shown.

Incrementation and iteration

In the incremental-iterative solution, the arclength L, which controls the size of the increment, was initially set to 0.005 m in each increment. The iterative procedure continued until (i) the absolute error E_{abs} was less than an absolute error tolerance of 1×10^{-28} J; (ii) The relative error E_{rel} was less than a relative error tolerance of 1×10^{-16} J; or (iii) The number of iterations i exceeded 20. If convergence was not reached or if the solution converged to a repeated equilibrium point, the arclength L was halved and the iterative procedure was restarted. The absolute and relative errors were estimated using a work-based error measure[8],

$$ {}^{k}E_{abs}^{(i)} = \left| \left({}^{k}\mathbf{U}^{(i)} - {}^{k}\mathbf{U}^{(i-1)} \right) \cdot \left({}^{k}\mathbf{R}^{(i)} - {}^{k}\mathbf{F}^{(i)} \right) \right| \tag{4.37} $$

$$ {}^{k}E_{rel}^{(i)} = {}^{k}E_{abs}^{(i)} / {}^{k}E_{abs}^{(1)}. \tag{4.38} $$

The scaling parameter k_a was determined based on the mesh; for the mesh used, $k_a = 0.0145\,\mathrm{m/V}$.

The simulation was terminated when any of the following conditions was reached:

- The input voltage exceeded a maximum of 1.5 V, corresponding to the experimental input range, or
- The arclength L fell below a minimum value of 1×10^{-6} m, or
- The deflection resulted in a contact condition with the glass slide.

When presenting the results, the tip deflection y_{tip} is assumed to remain constant after contact with the glass slide. Additionally, when a load reversal is encountered, the presented results do not follow the reversal path but instead jump to the next solution point with an increased voltage, which conforms more closely to what would be expected in an experiment.

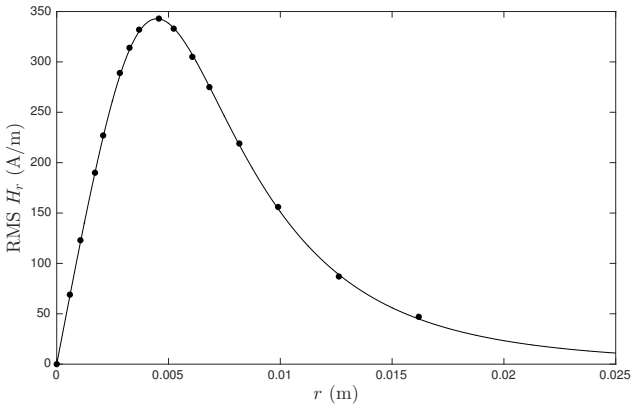


Figure 4.9: Measured radial magnetic field profile (i.e., the radial component of $\mathbf{H}_c(\mathbf{x})$) and best-fit model profile with a fit parameter $k_c = 0.722$ at an estimated height from the electromagnet top of $z_{meas} = 4.09\,\text{mm}$

4.3.7 Model fitting to experimental system and results

To compare the model predictions to experimental results, the physical constants and geometries of the system must be measured or estimated.

Electromagnet model fitting

To validate the model of the electromagnet described in Section 4.3.1, the electromagnet was driven at 20 Hz with a peak-to-peak voltage of 0.5 V and the generated field was sampled using a magnetometer (LakeShore 425). Since the active region of the magnetometer probe is uncertain, fitting the measured data to the model field profile requires simultaneously finding the fitting parameter k_c in Equation 4.11 and z_{meas}, the height from the electromagnet top at which the measurement was taken. Nonlinear least squares was used to find the fitting parameter k_c and measurement height z_{meas} that minimize the error between the measured data and the model data from Equation 4.11. The measured data points and fit are shown in Figure 4.9.

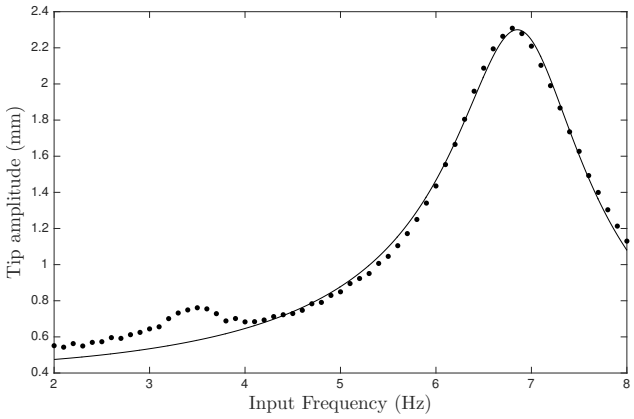


Figure 4.10: Experimental frequency response (•) and best fit via nonlinear least squares (—). The output amplitude is the peak-to-peak tip response, y_{pp}.

Material property fitting

The elastic modulus E was estimated based on the low-amplitude resonant frequency of the cilium. At an arbitrary magnet placement, a series of sinusoidal inputs,

$$V(t) = 0.2\sin(2\pi f_i t) + 0.2, \tag{4.39}$$

with f_i ranging from 1–8 Hz in 0.1 Hz increments, was applied for 5 s each. At each input frequency, the root-mean-square amplitude of the final 2 s of the response was used to estimate the peak-to-peak tip response y_{pp}. Then the resulting frequency response was fit to a second-order system model using nonlinear least squares, resulting in an estimated natural frequency of $\omega_1 = 6.9$ Hz. Finally, the elastic modulus was estimated from the beam theory result for the first cantilever mode [52],

$$\omega_1^2 = \frac{EI\beta_1^4}{\rho A} \tag{4.40}$$

where β_1 is the first root of $\cos(\beta_n L)\cosh(\beta_n L) = -1$, I is the moment of inertia, and A is the cross-sectional area of the cantilever.

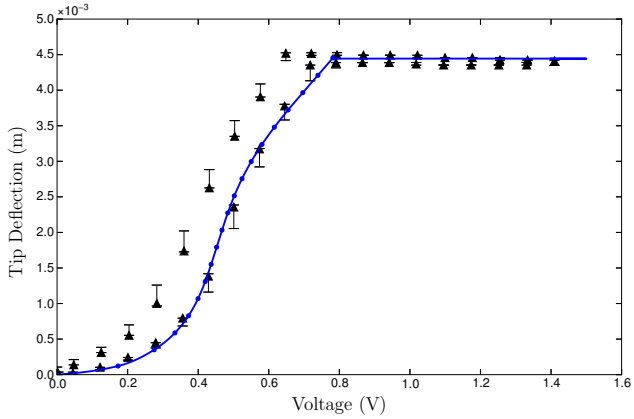


Figure 4.11: Model (–●–) and experimental (▲) results from the model fitting. Error bars indicate the range of estimated tip deflections for that voltage level.

Remark 6 *The constant offset in Equation 4.39 is required because the force is proportional to the square of the input voltage V, so the signal should be all-positive or all-negative. However, this results in a frequency component at twice the nominal frequency f_i, since*

$$(2\sin(2\pi f_i t) + 2)^2 = 8\sin(2\pi f_i t) - 2\cos(4\pi f_i t) + 6. \tag{4.41}$$

This feature can be observed as the presence of a second peak in Figure 4.10 at half the resonant frequency.

The magnetic susceptibility χ was estimated to be 10 by fitting the static deflection curve from the model to experimental data. The fitted model results and experimental data are shown in Figure 4.11. This value is similar to the values reported in [13] which, when converted from mass susceptibility to volume susceptibility given the estimated density of the Fe-PDMS composite, correspond to $\chi \sim 4$–12. For the static deflection experiment, a step-and-hold voltage pattern was used as the input, as shown in Figure 4.12. This input was selected to provide the voltage-deflection characteristic for the static case. The fitting parameters are given in Table 4.1.

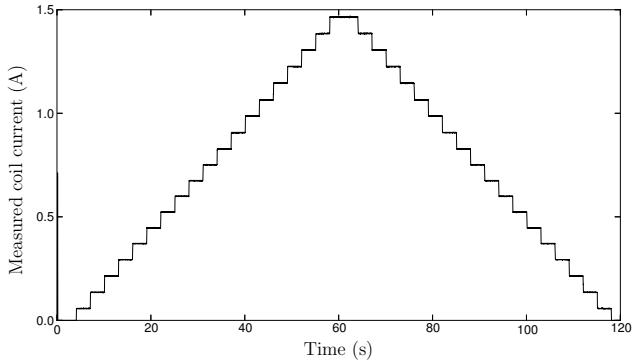


Figure 4.12: Typical measured input signal, i.e. the coil current i in Equation 4.1, for placement experiments.

Table 4.1: Model fit parameters estimated as discussed in Section 4.3.7.

Parameter	Estimated value
Elastic modulus E	13.915 MPa
Density ρ	3220 kg/m^3
Poisson's ratio ν	0.3
Cilium length L	17.1 mm
Cilium thickness W	0.19 mm
Cilium height H	2.95 mm
Cilium-to-slide distance	0.5 mm
Cilium initial rotation θ_b	3°
Applied field scaling k_c	0.75
Magnet out-of-plane position z_m	−2.975 mm[a]
Magnetic susceptibility χ	10

[a] Measured relative to the cilium center-line

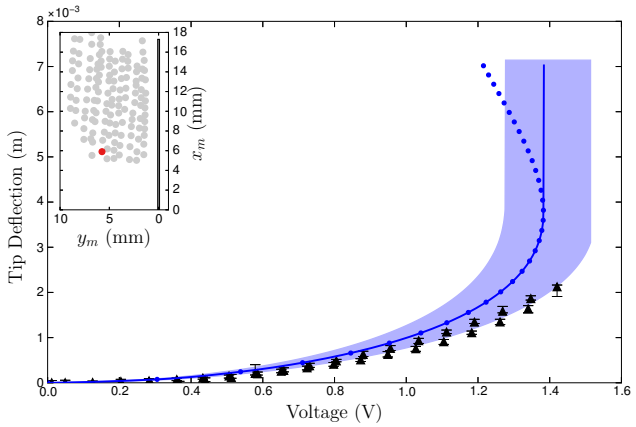


Figure 4.13: Model (—●—) and experimental (▲) results for the placement $(x_m,\ y_m) = (5.9,\ 5.74)$ shown in the inset (●). This placement is just outside the critical placement boundary, resulting in low response. Error bars indicate the range of estimated tip deflections for that voltage level. The shaded band indicates the model-predicted deflection with 5% variation in the transverse magnet position y_m. For load reversal (see Figure 4.6b), the model data points (●) follow the reversed path while the model line (—) shows the predicted response when the input voltage V is increased.

4.4 Results and discussion

4.4.1 Model validation

To assess the quality of the model fit, the static deflection experiments were repeated at different magnet placements. From these experiments, several representative placements were selected, showing responses with and without instabilities (jumps) and saturation. The experimental response at these placements is compared to model predictions in Figures 4.13 to 4.16. For the model predictions, the same fitting parameters were used as in Section 4.3.7.

The model validation runs, and the corresponding experiments, provide an illustration of the nonlinear response of the cilium when a magnetic field $\mathbf{H}_c$ is applied. At low input levels or distant placements (low applied force), the response is proportional to voltage squared, as

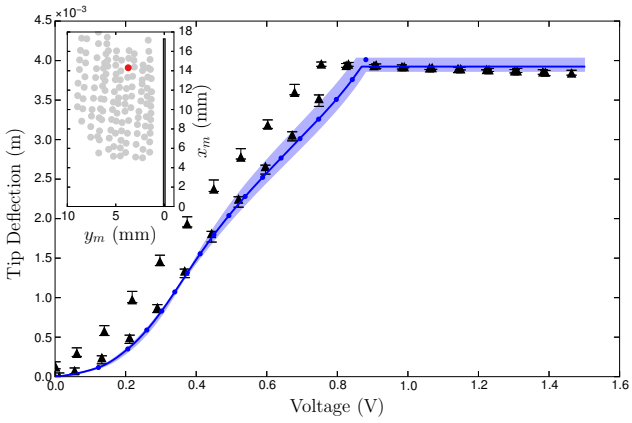


Figure 4.14: Model (—●—) and experimental (▲) results for the placement $(x_m,\ y_m) = (14.3,\ 3.72)$ shown in the inset (●). This experiment exhibited saturation due to contact and is near the optimal placement at low voltage. Error bars indicate the range of estimated tip deflections for that voltage level. The shaded band indicates the model-predicted deflection with 5% variation in the transverse magnet position y_m.

expected since the magnetic body force density $\mathbf{f}_b$ in Equation 4.22 is the product of two field quantities which are each proportional to the input voltage (Equation 4.11). However, as the cilium deflects under the magnetic load $\mathbf{R}$, the applied field $\mathbf{H}_c$ experienced by the cantilever tends to increase. The net effect is that the magnetic load $\mathbf{R}$ acts as a negative-stiffness nonlinear spring, since increased deflection results in increased load. When the magnetic load $\mathbf{R}$ increases faster than the internal elastic load $\mathbf{F}$, the system becomes unstable and a sudden increase in deflection is observed from point A to point B in Figures 4.15 and 4.16. This instability is critical for achieving high deflections.

The deflection is limited, however, by saturation due to contact. As the cilium moves over the electromagnet, the component of the applied load in the z-direction increases dramatically, pulling the cilium toward the slide. Since the cantilever aspect ratio is high, the structure resists out-of-plane bending and the force in the z-direction causes a torsional response and eventually contact between the cilium and the slide, resulting in a friction force

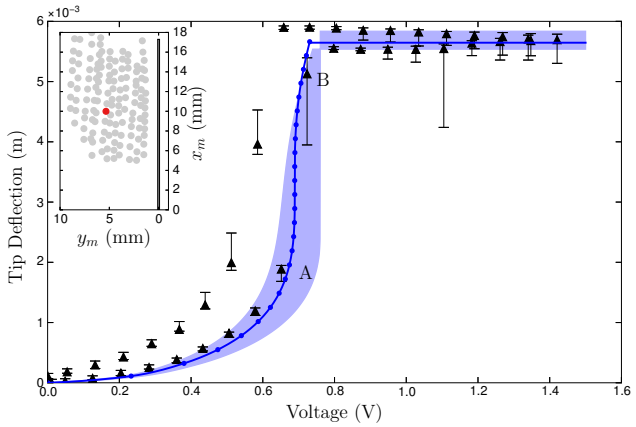


Figure 4.15: Model (—●—) and experimental (▲) results for the placement $(x_m,\ y_m) = (10,\ 5.34)$ shown in the inset (●). This experiment exhibited saturation due to contact and jump discontinuity due to negative stiffness. Error bars indicate the range of estimated tip deflections for that voltage level. The shaded band indicates the model-predicted deflection with 5% variation in the transverse magnet position y_m.

that prevents further deflection in the plane. Since the saturation occurs when part of the cilium is above the electromagnet, the saturated output level is a function of the magnet placement $\mathbf{X}_m$, and this behavior is observed in Figures 4.14 to 4.16.

The negative-stiffness load and the saturation, together, result in a bistable response for certain placements, which is best demonstrated by Figure 4.16. After point A, a critical voltage is reached that results in a sudden increase (unstable response) to point B, after which the deflection saturates due to contact. When the input voltage V is reduced the cilium remains in contact at the saturated deflection level at about 7 mm until reaching point C, at which point the magnetic force $\mathbf{R}$ is no longer strong enough to resist the internal elastic forces $\mathbf{F}$ and the cilium springs back to a nearly-undeflected position at point D. For inputs between about 0.5–1.3 V, there are two possible equilibrium solutions, one with $y_{\text{tip}} \lesssim 1$ mm and one at saturation, $y_{\text{tip}} \approx 7$ mm, i.e. the system is bistable.

These results show that, despite the nonlinear response, the model fit is reasonable even

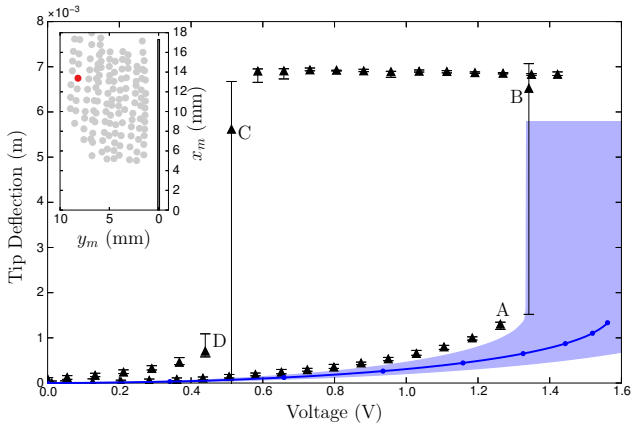
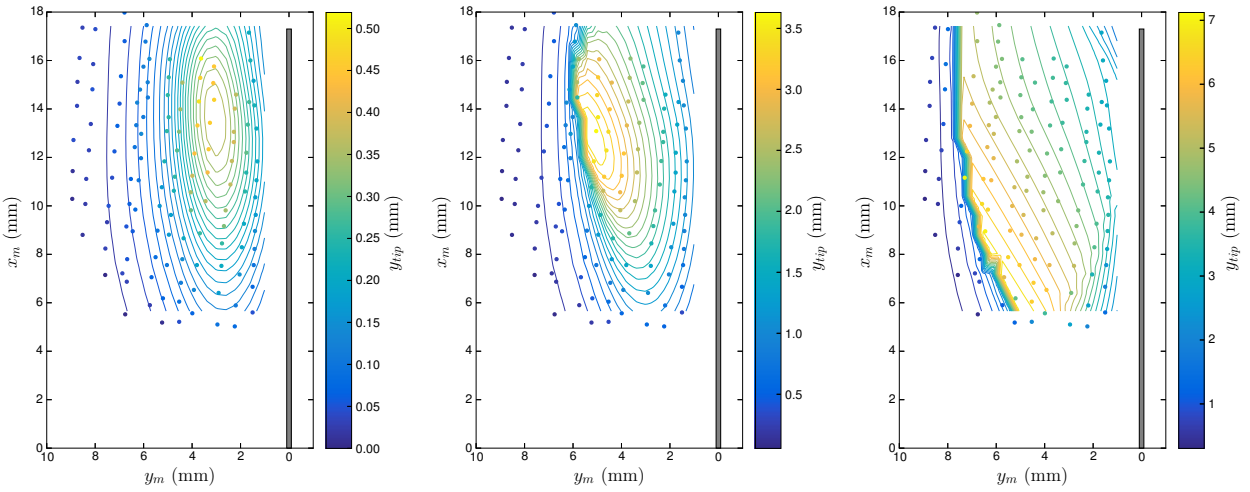


Figure 4.16: Model (—●—) and experimental (▲) results for the placement $(x_m,\ y_m) = (13.4,\ 8.18)$ shown in the inset (●). This placement is near the optimum at high voltage and the response includes jump discontinuity, saturation due to contact, and bistability. Error bars indicate the range of estimated tip deflections for that voltage level. The shaded band indicates the model-predicted deflection with 5% variation in the transverse magnet position y_m.



(a) $V = 0.2\,\mathrm{V}$ (b) $V = 0.58\,\mathrm{V}$ (c) $V = 1.19\,\mathrm{V}$

Figure 4.17: Cantilever deflections predicted by the model (contours) and measured experimentally (•).

when the magnet placement $\mathbf{X}_m$ is far from the placement used for fitting; in each case the model captures the experimental response up to a 5% error in the magnet position (shown as a shaded area). Model results are not shown for the return path since there seems to be material hysteresis which was not modeled.

4.4.2 Placement results: model vs experiment

To assess the overall quality of the model predictions, the experimental and predicted tip deflections from the static deflection experiments were mapped at fixed voltage levels, as shown in Figure 4.17, in which interpolated contours of the model results are plotted together with the corresponding experimental results. The model contours were estimated from simulations with magnet placements $\mathbf{X}_m$ sampled on a regular grid. Due to the nonlinear forcing from the magnet, the placement that produces the peak deflection varies substantially with input voltage, as does the overall shape of the deflection distribution. At low input voltage (Figure 4.17a), the contours are smooth and the optimal placement is relatively close to

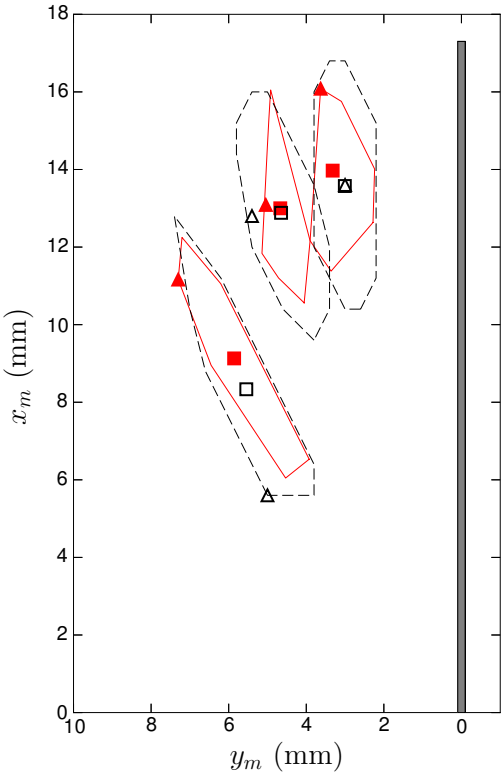


Figure 4.18: Outlines of the placements with deflections in the top 10% at the three input voltage levels shown in Figure 4.17 for the experimental (—) and model (– –) results. For each outlined region, the weighted average ($x_m^{\square}$, $y_m^{\square}$) computed from Equation 4.42 (■, □) and the placement ($x_m^{\triangle}$, $y_m^{\triangle}$) with the highest observed deflection (▲, △) are shown.

the cilium. As the input voltage increases (Figure 4.17b), the placement that produces the largest deflection moves farther from the cilium, and at high inputs also moves toward the cilium's base (Figure 4.17c). Additionally, at moderate to high voltages there is a sharp boundary, a critical distance beyond which the deflection suddenly decreases. The comparison to the model results shows that the model captures the spatial variation of the static amplitude function $y_{\text{tip}}(\mathbf{x}_m, V)$ from Equation 4.2.

As a further comparison between the model and experimental results, the optimal placement (i.e. maximum tip deflection y_{tip}) was estimated for the experimental and model results for various voltage levels. At each voltage, the location of the optimal placement was estimated as the weighted average

$$x_m^{\square} = \frac{1}{\sum y_{\text{tip}}^2} \sum x_{m,i} y_{\text{tip}}^2, \qquad y_m^{\square} = \frac{1}{\sum y_{\text{tip}}^2} \sum y_{m,i} y_{\text{tip}}^2, \tag{4.42}$$

where the sums are taken over points with tip deflections y_{tip} in the 90th percentile or above; these areas are shown in Figure 4.18. The estimated optimal placement $(x_m^{\square},\ y_m^{\square})$ is plotted as a function of voltage in Figure 4.19. The maximum errors in the axial and transverse coordinates, $x_m^{\square}$ and $y_m^{\square}$, are 11.3% and 9.7%. However, Figure 4.19 also shows that the maximal deflection at the optimal placement $(x_m^{\square},\ y_m^{\square})$ predicted by the model is within 5% of the deflection at the optimal placement predicted by the experimental data. Additionally, the overall shape of the optimal placement region seems to be well-predicted by the model, as seen in Figure 4.18.

The model and experiment can also be compared on the basis of the peak placement $(x_m^{\triangle},\ y_m^{\triangle})$ at which the highest deflection is observed at each input voltage V. The peak placements are shown as triangles in Figure 4.18 and seem to agree much less well than the weighted-average placement $(x_m^{\square},\ y_m^{\square})$. However, it should be noted that in the high-voltage case, in which the model and experimental peak placements are the farthest apart, the model contour line that passes through the estimated peak passes very close to the experimental peak placement. Also, at high input levels, the output amplitude is very sensitive near the boundary of the 90th-percentile placements; a placement slightly too far away would have

significantly lower achievable tip deflection, and performance loss could be caused by small perturbations to the system such as temperature changes or mass accumulation.

4.4.3 Optimal placement trends

The optimal placement problem is driven by the competition between the negative-stiffness load and the saturation. When the magnet is close to the cilium (small y_m) the cilium is in a region of high field initially, resulting in a strong force; when the magnet is also near the tip of the cilium (large x_m), the resulting modal force is increased since the bending moment about the base is higher. However, these conditions (small y_m, large x_m) also result in the smallest saturated deflection, as the deflection is highest near the tip, meaning the cilium will move over the magnet into the region of high out-of-plane field sooner. The result is that when the applied voltage is very low, the optimal placement is the one that maximizes the applied load, ignoring the saturation problem. Conversely, when the applied voltage is high, the optimal placement is shifted away from the cilium tip, so that the tip deflection at saturation is increased. An additional issue at high input levels is that the performance becomes very sensitive to the placement. This is because the best placements at high voltage are those at which the applied voltage is just enough to trigger the unstable response, resulting in a sharp boundary between high-deflection and low-deflection placements. For a practical design, a placement would necessarily be chosen safely inside this boundary so that system perturbations, such as temperature changes or particle attachment, do not cause a dramatic reduction in response.

4.4.4 Impact on design

The minimum cilia-cilia spacing to maintain independent control can be estimated from the optimal placement results. When the transverse placement coordinate y_m is greater than about 7 mm, the response is very low. So, for a given designed magnet placement $\mathbf{X}_m$, which should be selected appropriately for the designed peak input voltage V_{max}, the next cilium should be about 7 mm farther away. This means that if the magnet placement is selected

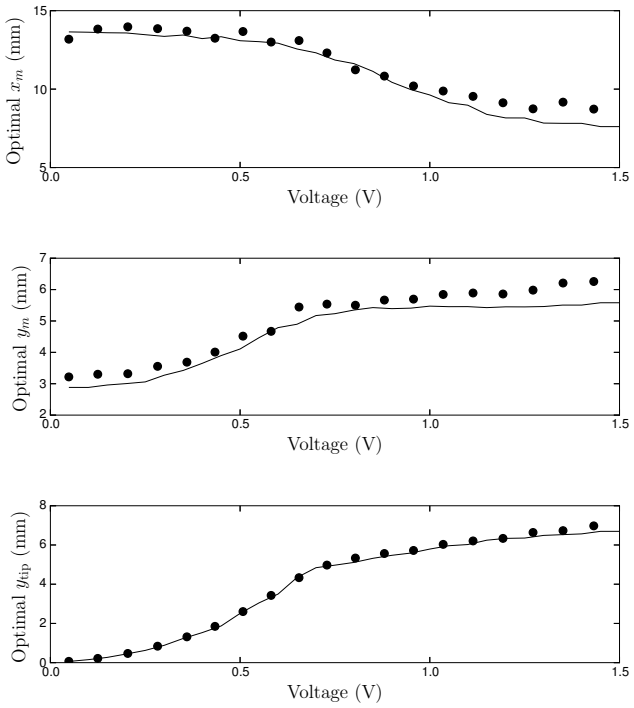


Figure 4.19: Trends in optimal placement ($x_m^{\square}$, $y_m^{\square}$) and maximal tip deflection $y_{\text{tip}}^{\square}$ with increasing input amplitude, as predicted by the model (—) and estimated from the experimental data (•). For the model-predicted tip deflection (bottom), the line indicates the experimental tip deflection linearly interpolated (via Delaunay triangulation[5]) at the optimal placement predicted by the model data.

for a low input voltage, e.g. $V_{max} = 0.2\,\text{V} \rightarrow y_m \approx 3\,\text{mm}$ based on Figure 4.19, the cilia-cilia spacing could be about 10 mm. On the other hand, if large deflections (and therefore high input voltage) is desired, the transverse placement $y_m \approx 6\,\text{mm}$, so the cilia-cilia spacing should be at least 13 mm. For denser arrays, the same modeling approach could be used to rapidly compare the placement results using different electromagnet designs to change the spatial extent of the applied field $\mathbf{H}_c$.

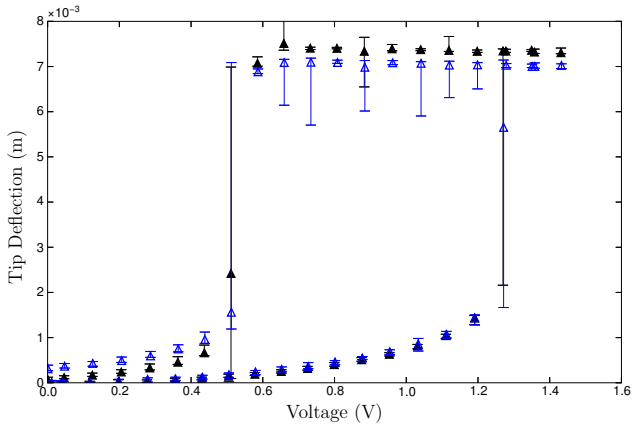


Figure 4.20: Comparison of static response in air (▲) and in water (△) with the magnet placement at $\mathbf{X}_m = (11.36, 7.22)$. This placement is near the optimal placement for $V = 1.19\,\mathrm{V}$ shown in Figure 4.18.

4.4.5 Comparison to fluid regime

Since the intended application for the cilia device is for fluidic devices, trials were run in water to compare to the experiments and model predictions in air. While a similar design has already been demonstrated to be effective for fluid manipulation tasks (see Chapter 3), this helps to validate the approach of predicting the placement from the static deflection in air. Figure 4.20 shows a comparison between the static response in water and in air near the optimal placement, which demonstrates that the results in air and in water are very similar. This is expected since the viscous forces from the fluid should be negligible in the static case. The frequency response in water and in air at the same placement is given in Figure 4.21. Although the responses differ at higher frequencies due to increased damping in water, the frequency responses also show that the responses in water and air are similar at low frequencies (close to the static case), as expected.

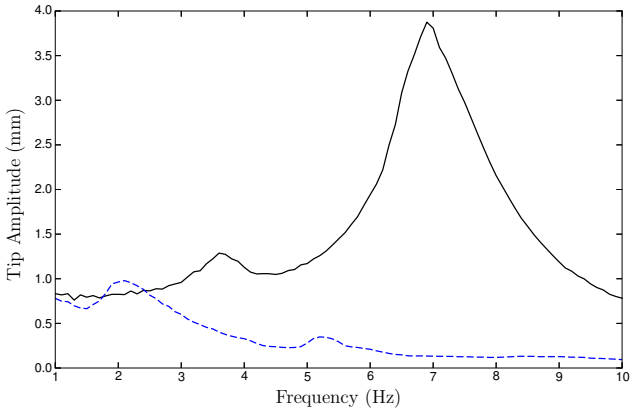


Figure 4.21: Comparison of frequency response in air (—) and in water (- -) with measured peak-to-peak input voltage 1.13 V. The peak-to-peak tip amplitude y_{pp} was estimated as in Section 4.3.7. The magnet placement was at $\mathbf{X}_m = (11.36, 7.22)$, which is near the optimal placement for $V = 1.19\,\mathrm{V}$ shown in Figure 4.18. At low frequencies (less than 2 Hz) the response is relatively constant and is similar in air and in water.

4.4.6 Impact of neglecting nonlinearities

As an illustration of the importance of capturing the nonlinear behavior, the peak experimental response at high voltage, $\mathbf{X}_{NL}^{\triangle}$, can be compared to the best placement at low voltage, $\mathbf{X}_{L}^{\triangle}$. Since the low-voltage case has a near-linear response, this is a reasonable approximation to the best-case placement prediction for a linear modeling approach. The experimental responses at these two locations are shown in Figure 4.22. Comparing the peak deflections at these two placement shows that the linear placement suffers from a reduction in amplitude of about 50% compared to the nonlinear placement. However, the nonlinear placement only achieves its peak deflection when it reaches an input voltage of at least 1.4 V; at all lower input levels, the tip deflection y_{tip} is higher at the linear placement by as much as 750%. This indicates the importance of correctly selecting the placement for the available voltage range: a placement optimized for too low of a voltage will saturate early, while a placement

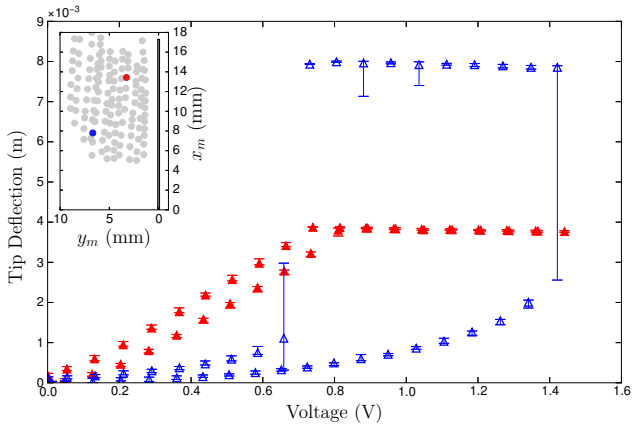


Figure 4.22: Experimental response at the low-input optimal placement $\mathbf{X}_L^{\triangle} = (13.4,\ 3.26)$ (▲) and high-input optimal placement $\mathbf{X}_{NL}^{\triangle} = (7.8,\ 6.68)$ (△). Neglecting the nonlinear effects leads to 50% reduction in tip displacement at high voltage.

optimized for too high of a voltage may not reach its critical voltage.

Chapter 5

APPLICATION OF ITERATIVE MACHINE LEARNING FOR OUTPUT TRACKING WITH MAGNETIC SOFT ACTUATORS

5.1 Introduction

This chapter presents experimental results of output tracking for highly-flexible magnetic actuators using iterative machine learning (IML) for model estimation and model uncertainty estimation. These devices are of interest as soft actuators but pose a control challenge as models of their dynamic response are sensitive to parameter variation between actuators. To overcome this challenge and improve tracking performance, complex-valued Gaussian Process regression (CGPR) is used to estimate a system inverse model from a single trial; additionally, the CGPR approach provides an uncertainty measure, which is used to design the frequency-dependent iteration gain at frequencies where the uncertainty is large to ensure convergence. This approach leads to convergence even when data near resonance is not available. Additionally, a method to improve the model estimation is proposed in which the initial training signal is augmented with additional frequency content where the nominal signal is small. Using this augmented approach, the iterative machine learning approach reduces the maximum tracking error from 19% of the range of the desired output when using the DC gain alone down to $< 3\%$, which is within the resolution limit.

5.2 Problem formulation and solution

5.2.1 System description

The nonlinear static response of the electromagnetic actuator, i.e. the transverse tip deflection y_{tip} at the selected placement, is plotted against the squared input voltage v^2 in Figure 5.1. It can be observed that the relation between tip deflection y_{tip} and squared

voltage v^2 is nearly linear for relatively low inputs, which is expected based on models of the magnetic force presented in Chapter 4. However, the magnetic force also varies with the tip deflection y, and at higher input levels, saturation is observed due to contact, as discussed in Chapter 4. Additionally, the response exhibits hysteresis, which may result from magnetic hysteresis or from viscoelastic material properties of PDMS. Because of these nonlinear features, the system input was defined to be

$$u(t) = v^2(t) - u_c, \tag{5.1}$$

where u_c is a nominal operating point, selected as $u_c = 0.2\,\mathrm{V}^2$, and the system output was defined as

$$y(t) = y_{\mathrm{tip}}(t) - y_c, \tag{5.2}$$

where y_c is the static deflection at the operating point u_c, so that $y \equiv 0$ when $u \equiv 0$. The hysteresis loop when operating at the chosen operating point of $u_c = 0.2\,\mathrm{V}^2$ is also shown in Figure 5.1, which shows that the response is relatively linear (when compared to the saturation case) even for relatively large deflections of $\pm 1\,\mathrm{mm}$ about the nominal deflection $y_{\mathrm{tip}} = y_c$; since the cantilever thickness is about $0.2\,\mathrm{mm}$, this trajectory would be expected to include large-deformation effects.

5.2.2 The output tracking problem

The tracking problem can be stated as follows: given a desired output $y_d(t)$, where in this case $y(t)$ is the transverse tip deflection of the cantilever, find an input (squared voltage) $u^*(t)$ such that $y(t) = y_d(t)$. In the frequency domain, this is expressed as finding $U^*(\omega)$ that satisfies

$$Y_d(\omega) = G(\omega)U^*(\omega) \tag{5.3}$$

where $G(\omega)$ is the transfer function of the system linearized about the nominal operating point (u_c, y_c) and the time-domain signals $u(t)$, $y(t)$ are mapped to and from the frequency domain via the Fourier transform.

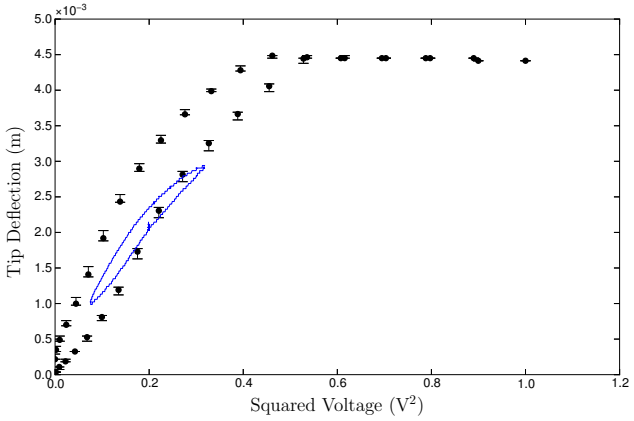


Figure 5.1: Static response of the electromagnetic actuator, i.e. the transverse tip deflection y_{tip} at the selected placement plotted against the input voltage v squared (•). The error bars indicate the range of measured values. When plotted against the input voltage squared, the nonlinearity of the response is reduced about the operating point of $u_c = 0.2\,\text{V}^2$. Moreover, if saturation is avoided, the hysteresis nonlinearity is reduced, as shown by the smaller hysteresis loop (—).

Remark 7 (Deviations from operating point) *Because the system input and output, $u(t)$ and $y(t)$, are measured relative to a nominal operating point with $u = u_c$, $y = y_c$, $G(\omega)$ can be considered to be linearized about the operating point.*

5.2.3 Iterative learning control

Since the transfer function $G(\omega)$ linearized about the operating point is not available, the desired input $u^*(t)$ will be solved through iterative corrections. Starting from an initial input

$$U_0(\omega) = Y_d(\omega)/\hat{G}(0), \tag{5.4}$$

where $\hat{G}(0)$ is an estimate of the static gain of the system, successive corrections are made based on the error between the desired output $Y_d(\omega)$ and the true output at iteration k,

$Y_k(\omega) = G(\omega)U_k(\omega)$, as

$$U_{k+1}(\omega) = U_k(\omega) + \rho(\omega)\hat{G}^{-1}(\omega)(Y_d(\omega) - Y_k(\omega)). \tag{5.5}$$

If the estimated inverse transfer function $\hat{G}^{-1}(\omega)$ has no error, i.e., $\hat{G}^{-1}(\omega) = G^{-1}(\omega)$, and the iteration gain $\rho(\omega) = 1$, then the update will result in the exact-tracking inverse input, i.e.,

$$\begin{aligned} U(\omega) &= G^{-1}(\omega)Y_d(\omega) = G^{-1}(\omega)G(\omega)U^*(\omega) \\ &= U^*(\omega). \end{aligned} \tag{5.6}$$

When modeling error is present, then exact output tracking is not achieved. The change in the error $Y_d - Y_k$ can be quantified as, dropping the argument ω for conciseness,

$$U_{k+1} - U_k = G^{-1}(Y_{k+1} - Y_k) = \rho\hat{G}^{-1}(Y_d - Y_k) \tag{5.7}$$

or, rearranging,

$$\frac{Y_d - Y_{k+1}}{Y_d - Y_k} = 1 - \rho\frac{\hat{G}^{-1}}{G^{-1}}, \tag{5.8}$$

and convergence is achieved if the right-hand-side term of Equation 5.8 is less than 1, i.e. the series of inputs $U_k(\omega)$ is a contraction. Then the iterative update law requires two quantities: the estimated inverse transfer function $\hat{G}^{-1}(\omega)$ and the iteration gain $\rho(\omega)$, which together must satisfy Equation 5.8.

5.2.4 Bounds on iteration gain

Given an estimated inverse system model $\hat{G}^{-1}(\omega)$, the iteration gain $\rho(\omega)$ must be selected to ensure convergence given some bound on the modeling error, which is expressed as

$$\frac{G^{-1}}{\hat{G}^{-1}} = \Delta_m e^{j\Delta_p} \tag{5.9}$$

where $\Delta_m(\omega)$ and $\Delta_p(\omega)$ are the magnitude and phase error at the frequency ω, respectively. It was shown in [73] that convergence is achieved, i.e. the right-hand side of Equation 5.8 is

less than one, at frequency ω if and only if

$$|\Delta_p(\omega)| < \pi/2 \tag{5.10}$$

$$0 < \rho(\omega) < \frac{2\cos\Delta_p(\omega)}{\Delta_m(\omega)}. \tag{5.11}$$

The bounds on the iteration gain $\rho(\omega)$ in Equation 5.11 may also be expressed in terms of uncertainty in the real and imaginary parts of $\hat{G}^{-1}(\omega)$. The actual and estimated inverse systems are written as

$$G^{-1}(\omega) = a + jb \tag{5.12}$$

$$\hat{G}^{-1}(\omega) = \hat{a} + j\hat{b}. \tag{5.13}$$

The absolute errors in the real and imaginary parts are assumed to be bounded at each frequency ω by non-negative real values Δ_a, Δ_b, such that

$$|a(\omega) - \hat{a}(\omega)| \leq \Delta_a(\omega), \qquad |b(\omega) - \hat{b}(\omega)| \leq \Delta_b(\omega). \tag{5.14}$$

Then the magnitude error can be rewritten as

$$\Delta_m = \left|\frac{G^{-1}}{\hat{G}^{-1}}\right| = \left|\frac{a+jb}{\hat{a}+j\hat{b}}\right| \tag{5.15}$$

and so

$$\Delta_m^2 = \frac{a^2+b^2}{\hat{a}^2+\hat{b}^2} \leq \frac{(|\hat{a}|+\Delta_a)^2 + (|\hat{b}|+\Delta_b)^2}{\hat{a}^2+\hat{b}^2} \tag{5.16}$$

$$\therefore \Delta_m \leq \frac{|(|\hat{a}|+\Delta_a) + j(|\hat{b}|+\Delta_b)|}{|\hat{a}+j\hat{b}|}. \tag{5.17}$$

A bound on the phase error Δ_p is found by taking the dot product of the true and model inverse systems, resulting in

$$\cos\Delta_p = \frac{\hat{G}^{-1}\cdot G^{-1}}{|G^{-1}||\hat{G}^{-}1|} \tag{5.18}$$

$$= \frac{a\hat{a}+b\hat{b}}{|a+jb||\hat{a}+j\hat{b}|}. \tag{5.19}$$

A lower bound on the cosine of the phase error can be obtained by minimizing the numerator and maximizing the denominator. Since a is bounded by $\hat{a} \pm \Delta_a$ and b is bounded by $\hat{b} \pm \Delta_b$, the extremal values of the numerator are

$$a\hat{a} + b\hat{b} = \hat{a}^2 \pm |\hat{a}|\Delta_a + \hat{b}^2 \pm |\hat{b}|\Delta_b, \tag{5.20}$$

which is minimized when taking both error terms to be negative. Similarly, for the denominator, $|G^{-1}|$ is maximized when the error terms Δ_a and Δ_b have the same sign as the real and imaginary components $\hat{a}$ and $\hat{b}$, i.e.,

$$\max |G^{-1}| = |(|\hat{a}| + \Delta_a) + j(|\hat{b}| + \Delta_b)|. \tag{5.21}$$

Combining Equations 5.19 to 5.21 results in a bound on the phase error,

$$\cos \Delta_p \geq \frac{\hat{a}^2 + \hat{b}^2 - |\hat{a}|\Delta_a - |\hat{b}|\Delta_b}{|\hat{a} + j\hat{b}||(|\hat{a}| + \Delta_a) + j(|\hat{b}| + \Delta_b)|}. \tag{5.22}$$

Then the convergence condition in Equation 5.11 is satisfied, substituting Equations 5.17 and 5.22, when

$$0 < \rho < 2\frac{\hat{a}^2 + \hat{b}^2 - |\hat{a}|\Delta_a - |\hat{b}|\Delta_b}{(|\hat{a}| + \Delta_a)^2 + (|\hat{b}| + \Delta_b)^2}. \tag{5.23}$$

5.2.5 Complex Gaussian Process Regression Model

The inverse model can be estimated from experimental data in the frequency domain by taking the ratio

$$\hat{G}^{-1}(\omega) = U(\omega)/Y(\omega), \tag{5.24}$$

where the frequency-domain representations $U(\omega) = \mathcal{F}(u(t))$ and $Y(\omega) = \mathcal{F}(y(t))$ are obtained via the Fast Fourier Transform (FFT). However, in applications this data may be unavailable or noisy at some frequencies. In this case, the inverse system transfer function $\hat{G}^{-1}$ can be estimated via complex-valued Gaussian Process regression (CGPR). In the

CGPR framework, the data is viewed as noisy outputs of an unknown function of frequency ω,

$$\hat{G}^{-1}(\omega) = f(\omega) + \epsilon, \tag{5.25}$$

and the noise term ϵ is assumed to be a zero-mean Gaussian random variable with variance σ_ϵ^2. The assumption of the CGPR is that similar inputs ω_1, ω_2 will result in similar function values $f(\omega_1)$, $f(\omega_2)$; the similarity is measured by the kernel function $k(\omega_1, \omega_2)$, which in this case was taken as the squared-exponential kernel [63],

$$k(\omega_1,\ \omega_2) = \sigma_f \exp\left(-\frac{1}{2}\gamma^{-2}(\omega_1 - \omega_2)^H(\omega_1 - \omega_2)\right), \tag{5.26}$$

where σ_f is the function variance, γ is the length scale of the kernel, and the superscript H denotes the complex conjugate transpose. Given known input-output pairs $(\omega_T, \hat{G}_T^{-1})$, the predictive mean and variance are given by

$$\mathbb{E}[f(\omega)] = K(\omega, \omega_T)K_{\hat{G}^{-1}}^{-1}\hat{G}_T^{-1} \tag{5.27}$$

$$\mathbb{V}[f(\omega)] = K(\omega, \omega) - K(\omega, \omega_T)K_{\hat{G}^{-1}}^{-1}K(\omega_T, \omega) \tag{5.28}$$

where $K(\omega_1, \omega_2)$ is the covariance matrix formed by evaluating the kernel function $k(\cdot, \cdot)$ on all pairs of inputs in ω_1 and ω_2 and $K_{\hat{G}^{-1}} = K(\omega_T, \omega_T) + \sigma_\epsilon^2 I$ is the covariance matrix of the noisy measurements.

Remark 8 (Uncertainty estimation) *The CGPR framework naturally provides an uncertainty measure, i.e. the predictive variance in Equation 5.28. The iterative gain can then be computed by setting the bounds Δ_a and Δ_b on the real and imaginary parts of the inverse system model $\hat{G}^{-1}$ as, e.g., twice the standard deviation estimated by the CGPR model. Since the squared-exponential kernel used here treats the real and imaginary parts as independent, this results in*

$$\Delta_a = \Delta_b = 2\sqrt{\mathbb{V}[f]}. \tag{5.29}$$

Remark 9 (Filtering of unknown frequency components) *Since the CGPR predictions tend to go to zero when the test points ω are far from the training data ω_T, computing an input signal from Equation 5.5 using the estimated inverse system model given by Equation 5.27 tends to naturally filter out frequency content where the data is poor. This helps to prevent divergence of the iterative process.*

5.2.6 Signal augmentation

To ensure that sufficient frequency data is available for model training, the initial input signal $u_0(t)$ can be augmented with frequency content where it would otherwise be small. The total input becomes

$$u_0'(t) = u_0(t) + \tilde{u}(t) = u_0(t) + \sum_i \tilde{A}\sin(\tilde{\omega}_i t) \tag{5.30}$$

where the added frequencies $\tilde{\omega}_i$ are selected to span the frequency region of interest. The augmentation magnitude $\tilde{A}$ must be large enough that the signal-to-noise ratio is reasonable at the added frequencies $\tilde{\omega}_i$ while not substantially shifting away from the nominal operating point about which the system is linearized.

5.2.7 Hyperparameter estimation

The hyperparameters of the kernel function given in Equation 5.26, i.e. the length scale γ, signal variance σ_f, and noise variance σ_ϵ, which are not known *a priori*, must be estimated from the experimental data before predictions can be generated. After applying the initial trial input $u_0(t)$ defined by Equation 5.4 to the experimental system, resulting in a response $y_0(t)$, both signals were converted to frequency domain via FFT, and the inverse model data was computed from the resulting ratio as

$$\hat{G}^{-1}(\omega) = U_0(\omega)/Y_0(\omega). \tag{5.31}$$

Based on this data, the hyperparameters were selected by maximizing the log marginal likelihood function [63],

$$\log p(G^{-1}|\omega,\theta) = -\frac{1}{2}\left(G^{-T}K_{\hat{G}^{-1}}^{-1}G^{-1} + \log|K_{\hat{G}^{-1}}| + n\log 2\pi\right), \tag{5.32}$$

where θ represents the set of hyperparameters. Since this procedure relies on sampled input-output pairs to estimate the hyperparameters, the predictions may be poor if the sampling is uneven or the data is unreliable. In particular, at frequencies where the input $U(\omega)$ or the output $Y(\omega)$ are small, the ratio $G^{-1} = U(\omega)/Y(\omega)$ is more sensitive to noise or cross-frequency interactions corrupting the model estimates. On the other hand, if the usable frequency range is not well-sampled, the length scales may not be well-estimated by maximizing the marginal likelihood. To avoid samples with high signal-to-noise ratio, only points with sufficiently large input or output magnitudes were added to the dataset for the GPR model, i.e., only those points satisfying

$$|Y_0(\omega)| \geq 0.1\max_{\omega}|Y_0(\omega)|, \qquad |U_0(\omega)| \geq 0.5\max_{\omega}|U_0(\omega)| \tag{5.33}$$

were included. For this work, the starting guess for the length scale was selected as half the sampling rate, $\gamma_0 = F_s/2 = 120\,\text{Hz}$, and the starting signal and noise standard deviations were set to the standard deviation of the noisy model-data measurements, $\sigma_{f0} = \sigma_{\epsilon 0} = \sigma(U/Y)$.

5.2.8 Algorithm

The iterative machine learning algorithm is as follows.

Algorithm 1 Iterative machine learning

Require: $G(0),\ Y_d(\omega)$

Initialize:

Apply input $u_0(t)$ (5.4), measure output $y_0(t)$

Compute: $Y_0 = \mathcal{F}(y_0)$, $U_0 = \mathcal{F}(u_0)$

$\omega_T : |Y_0| \geq 0.1 \max_\omega |Y_0|$ **and** $U_0 \geq 0.5 \max_\omega |U_0|$ (5.33)

Compute: $\hat{G}_T^{-1}(\omega_T) = U_0(\omega_T)/Y_0(\omega_T)$ (5.24)

Compute: γ, σ_f, σ_ϵ (5.32)

Compute: $\hat{G}^{-1} = \mathbb{E}[f(\omega)]$ (5.27)

Compute: $\Delta_a = \Delta_b = 2\mathbb{V}[f(\omega)]$ (5.28)

Compute: $\rho = \frac{\hat{a}^2+\hat{b}^2-|\hat{a}|\Delta_a-|\hat{b}|\Delta_b}{(|\hat{a}|+\Delta_a)^2+(|\hat{b}|+\Delta_b)^2}$ (5.23)

Compute: $U_1 = \hat{G}^{-1} Y_d$

Loop:

for $k = 1$ to 15 **do**

Apply $u_k(t)$, measure $y_k(t)$

Update: $U_{k+1} = U_k + \rho \hat{G}^{-1}(Y_d - Y_k)$ (5.5)

end for

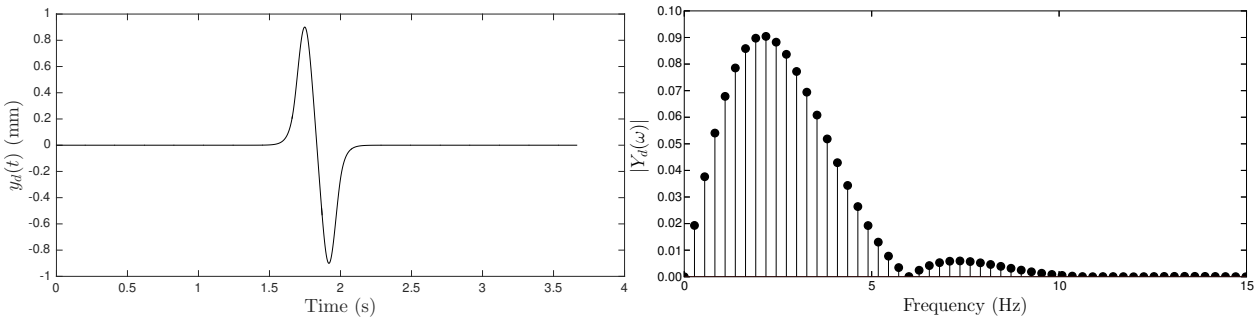
5.3 Experiments

5.3.1 Desired trajectory

The selected trajectory for the experiments was constructed as one cycle of a triangle wave using only the first two harmonics, i.e.

$$y_d(t) = \frac{8}{\pi^2}\left(\sin(\omega_0 t) + \frac{1}{9}\sin(3\omega_0 t)\right), \tag{5.34}$$

where the fundamental frequency ω_0 was taken to be 3 Hz and $0 \leq t \leq 1/3$. To allow for pre- and post-actuation in the inverse input $u^*(t)$, this signal was then padded with five periods of zeros before and after the triangle wave, smoothed using a zero-phase low-pass filter with



(a) (b)

Figure 5.2: Desired trajectory in (a) time domain and (b) frequency domain.

transfer function

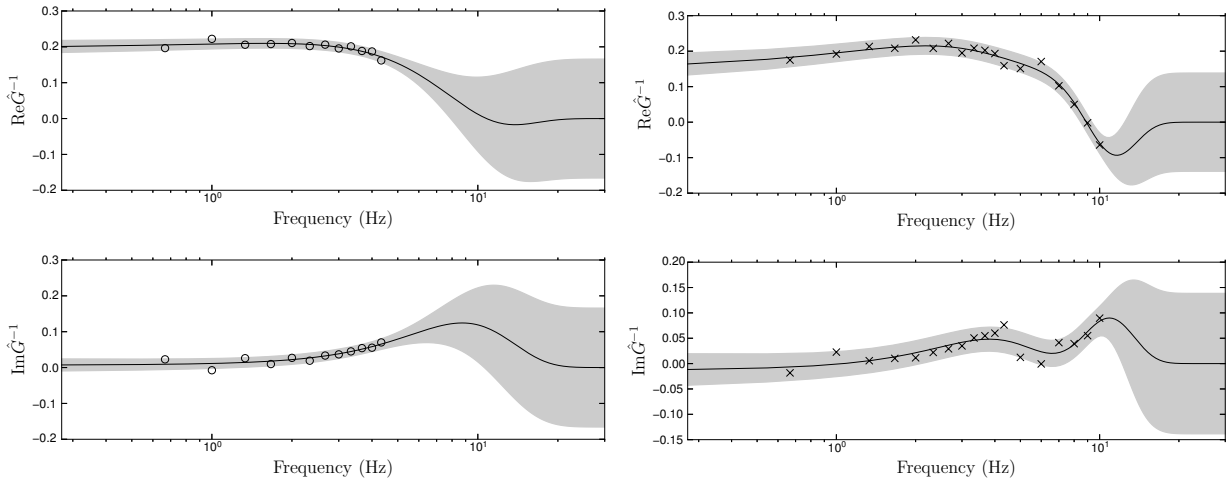
$$G_f(\omega) = \left(\frac{a}{j\omega + a}\right)^* \left(\frac{a}{j\omega + a}\right), \tag{5.35}$$

where the cutoff frequency a was selected as $1.5\omega_0$. Finally, this signal was scaled back up to its original amplitude. The resulting desired output $y_d(t)$ and frequency-domain representation $Y_d(\omega)$ are shown in Figures 5.2a and 5.2b.

5.3.2 Data-based models

Model without signal augmentation

When training the model without the augmented signal, the initial input $u_0(t)$ was computed according to Equation 5.4, with the DC gain $G(0)$ estimated from the static data shown in Figure 5.1 as $5.42\,\mathrm{mm/V^2}$. After applying this signal, optimizing the hyperparameters based on the thresholded model data, as discussed in Section 5.2.7, resulted in the following estimates: length scale $\gamma = 33.82$; signal standard deviation $\sigma_f = 0.165$; and noise standard deviation $\sigma_\epsilon = 0.012$. The real and imaginary parts of the thresholded data points and GPR fit are shown in Figure 5.3a.



(a) Unaugmented (b) Augmented

Figure 5.3: (a) Real and imaginary parts of the GPR fit without augmented input (—) and the training samples (∘). (b) Real and imaginary parts of the GPR fit with augmented input (—) and the training samples (×). In both cases, the shaded area indicates ±1 standard deviation of the noisy samples (i.e. $\Delta_{a,b} + \sigma_\epsilon$).

Model with signal augmentation

To increase the frequency range over which iterations may be applied, the first input signal was augmented as described in Section 5.2.6 at integer frequencies $\tilde{\omega}_i$ where the unaugmented signal $|U(\omega)|$ is below the threshold value in Equation 5.33. The augmentation amplitude $\tilde{A}$ was selected as $0.75 \max |U(\omega)|$ so that the added frequency content would be above the threshold value of $0.5 \max |U(\omega)|$. The resulting augmented signal $u'(t)$ and the frequency-domain representation $U'(\omega)$ are shown in Figure 5.4. Learning the hyperparameters from this data resulted in the following estimates: length scale $\gamma = 17.724$; signal standard deviation $\sigma_f = 0.137$; and noise standard deviation $\sigma_\epsilon = 0.022$. The estimated model from this fit is shown in Figure 5.3b.

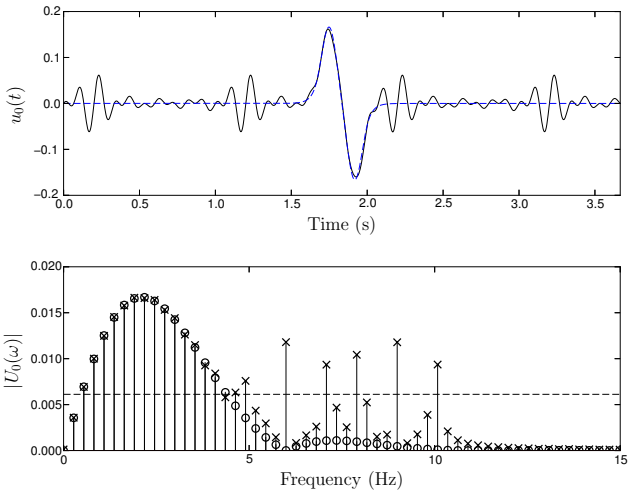


Figure 5.4: Top: Unaugmented initial input $u_0(t)$ (– –) and augmented initial input $u_0'(t)$ (—) in time domain. Bottom: Augmented ($\times$) and unaugmented ($\circ$) input signals in frequency domain. The dashed line indicates the cutoff amplitude for inclusion in the training data.

Comparison between augmented and unaugmented models

The Bode plots of the two models are shown in Figure 5.5, along with the training data. The augmented signal provides a substantially wider band of data to fit the model, resulting in a fit that includes an apparent resonance around 8.5 Hz, as evidenced by the dip in the inverse magnitude $|\hat{G}^{-1}|$ and the shift in the inverse phase $\angle\hat{G}^{-1}$. However, the phase shift would be expected to be 180° at resonance, whereas the data is only sufficient to support a shift of about 150° in the model. In comparison, the model fit without augmented data shows the same trends, extrapolating from the data available. Still, as would be expected, the unaugmented model is a poor fit to the data available to the augmented model and should be expected to be inaccurate, especially near resonance.

The iterative gain $\rho(\omega)$ for both the augmented and unaugmented models, computed as half the maximum value in Equation 5.23, is shown in Figure 5.6. Because the gain of the unaugmented model is higher for low frequencies (due to the narrower uncertainty bands

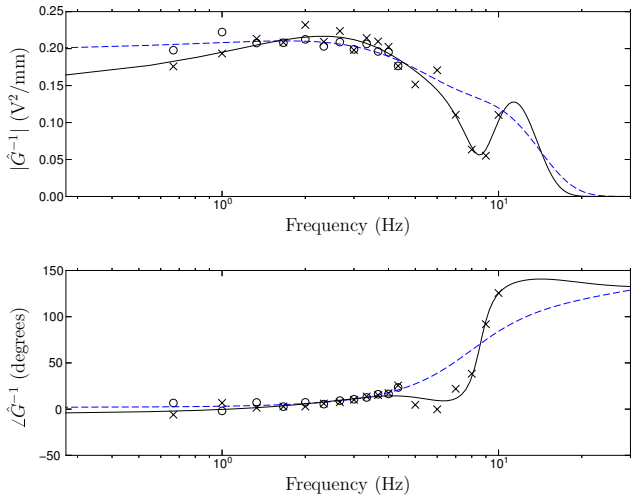


Figure 5.5: Magnitude and phase of the estimated inverse systems generated using unaugmented data (−−) and augmented data (—). The training data is also shown for the unaugmented case (×) and the augmented case(∘).

seen in Figure 5.3a vs Figure 5.3b), the convergence may be faster at those frequencies. However, the uncertainty in the unaugmented model is too high to safely apply corrections past about 8 Hz, which prevents convergence at those frequencies. In contrast, the additional data available to the augmented model allows for nonzero iteration gain ρ up to about 11 Hz, or 1 Hz above the available data.

5.3.3 Tracking results

The convergence results for the augmented and unaugmented case are shown in Figure 5.7. While both methods reduce the error, the unaugmented model does not reach the same level of accuracy as the augmented model—the best iteration in the unaugmented case resulted in about twice the error as the best-case augmented trial. This is explained by the presence of frequency content in the desired output $Y_d(\omega)$ between 8 and 10 Hz; because the iteration gain for the unaugmented model is zero in this range (see Figure 5.6), which also includes the

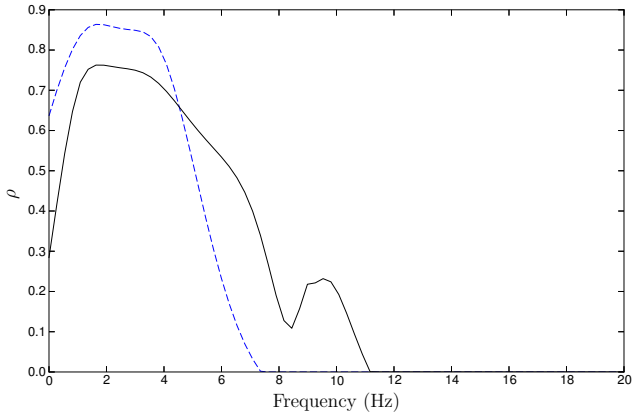


Figure 5.6: Iterative gain for the augmented (—) and unaugmented(– –) models. Because the gain for the unaugmented model is zero beyond 8 Hz, tracking of those frequency components will not be possible. In comparison, the augmented model is confident enough to apply iterative corrections up to 11 Hz.

resonance of the cantilever, the unaugmented model is unable to converge fully. However, the unaugmented model is still able to correct most of the error, reducing the error from 0.34 mm at iteration 0 (i.e. when using $u_0(t)$ from Equation 5.4) to 0.1 mm at iteration 12, which was the trial with the lowest maximum tracking error for the unaugmented case.

The error can be further reduced by including the signal augmentation, as shown in Figure 5.8. The corresponding inputs are shown in Figure 5.9. Using the DC gain to estimate the inverse input results in a maximum error of 0.34 mm, or about 19% of the desired range of motion. The error is plotted against time in Figure 5.8a, from which it can be seen that two major sources of error are an offset in the final settled position, which may be attributable to hysteresis, and ringing. Additionally, the peaks of the desired trajectory y_d are not well-tracked. After learning a model from the augmented input signal (see Figure 5.4), the input $u_1(t)$ computed from the estimated inverse system $\hat{G}^{-1}$ is able to substantially reduce the error, as shown in Figure 5.8b. In this case, the maximum error is reduced to 0.20 mm, or about 11% of the full range of motion. However, the hysteresis and

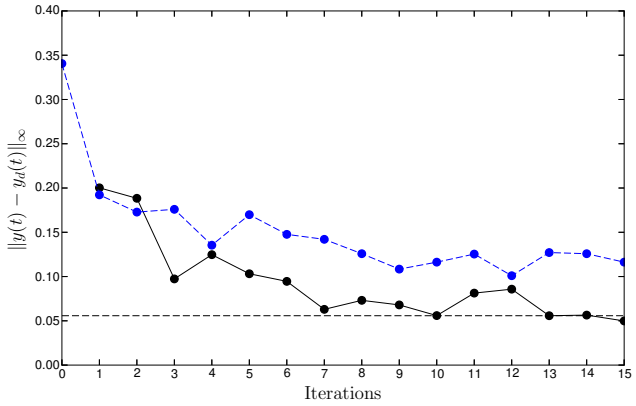
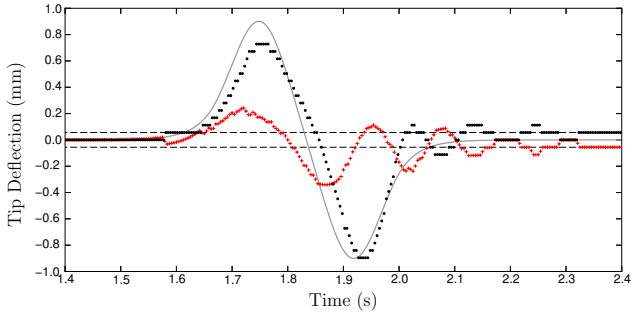
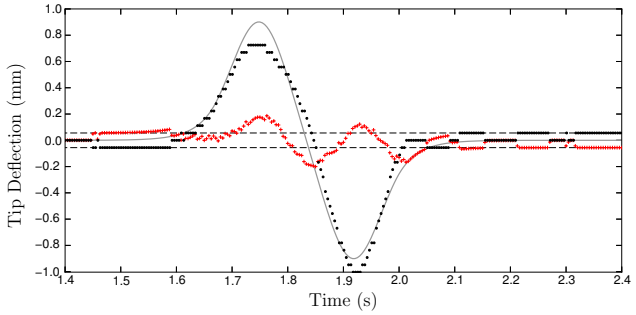


Figure 5.7: Comparison of the convergence when iterating using the unaugmented (– –) and augmented (—) GPR model fits. While the unaugmented model converges, it does not reach the resolution limit. By adding frequency content to only the first input, the learned model is able to reduce the maximum error to less than one pixel, i.e. perfect tracking is achieved.

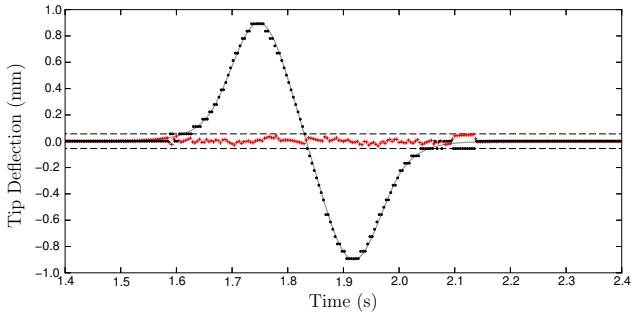
ringing are not fully compensated. After iterating 15 times, excellent tracking is achieved, as shown in Figure 5.8c. In this case the maximum tracking error is reduced to 0.05 mm, which is similar to the resolution limit of 0.056 mm. This final tracking error is less than 3% of the full range of motion.



(a) $u(t) = u_0(t)$



(b) $u(t) = u_1(t)$



(c) $u(t) = u_{15}(t)$

Figure 5.8: Comparison of tracking results for three cases: (a) using the initial (unaugmented) input $u_0(t)$ from Equation 5.4; (b) the first learned input for the augmented model; (c) the final learned input using the augmented model. Each plot shows the desired trajectory $y_d(t)$ (—), measured deflection $y_i(t)$ (•), and tracking error $y_d(t) - y_i(t)$ (+). The dashed lines indicate the resolution limits on the error, i.e. ± 1 pixel in the video frames. The measured output $y_i(t)$ is steady outside the time period plotted. The GPR-estimated model provides a substantial reduction in the tracking error (reduces the maximum error by about 50%) and reduces the maximum error to less than one pixel during the iterative process.

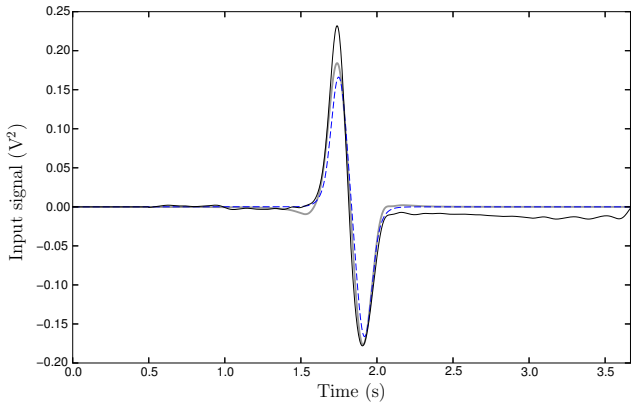


Figure 5.9: Comparison of inputs showing the initial (unaugmented) input $u_0(t)$ from Equation 5.4 (– –), the first learned input using the augmented model (—), and the final learned input using the augmented model (—).

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

6.1.1 Blinking vortex-based input patterns

A novel magnetic-cilia device was presented that achieves individual control of multiple cilia. The device produces large cilia deflections of 59% of the cilium length, which results in effective flow generation in the chamber. By exploiting the individual actuation capability, a reduction in mixing time of 38% was achieved compared to simultaneous actuation with the same average cilium beat frequency. The results show that there is an optimal beat pattern, which minimizes the mixing time. Additionally, simulations based on the blinking vortex theory show similar trends and predict a similar optimal beat pattern.

6.1.2 Nonlinear modeling for magnet placement

The nonlinear behavior of cilia actuator, including instability, bistability, and saturation, was characterized experimentally. A modeling procedure was developed and presented that accurately predicts the nonlinear response, and the model prediction of the optimal magnet placement that maximizes tip deflection was compared to experimental results with varying magnet placement. The experimental deflection at the model-predicted placement was within 5% of the optimal placement estimated from the experimental data.

6.1.3 Iterative machine learning for control

Complex-valued Gaussian Process regression (CGPR) was used to find an estimated system inverse model $\hat{G}^{-1}$ from a single experiment, and to estimate the modeling error. This

method resulted in convergence near the frequencies included in the model and the estimated uncertainty was successful in preventing divergence at frequencies far from the available data. However, learning based on the nominal input signal did not result in perfect tracking since the noise to signal ratio was large at some frequencies. To improve the performance, the training signal was augmented with additional input at frequencies where the nominal signal was small. Iterating using the resulting model led to perfect tracking—i.e. the maximum tracking error was within the resolution limit of the sensors. In comparison, using an estimated inverse input based on the measured DC gain alone resulted in 19% error.

6.2 Future work

6.2.1 Applications testing

While the two-cilia device presented in Chapter 3 works well for the batch mixing process applicable to microarrays, devices with more than two cilia may be advantageous for future work on other applications. For example, cilia devices for pumping applications using a row of cilia have been theoretically [31, 41] and experimentally[33] studied; these studies have shown that a metachronal wave, i.e. a phase difference along the row of cilia, improves pumping performance. Using the magnet wheel method, a fixed-phase array of n cilia can be actuated using a single motor and axle bearing n wheels; the phase difference between the actuation of adjacent cilia can be implemented by adjusting the angle of each magnet wheel relative to its neighbors. Using smaller-diameter magnets would reduce the range of influence of the magnetic field and allow for denser cilia spacings, smaller cilia, and/or smaller chambers.

To improve the performance of the present device for mixing performance and other tasks, different cilia arrangements may also be advantageous. Prior work [44] has shown simulated results indicating that cilia arrangements on opposite sides of a channel can be more effective for generation of chaotic advection flows. This result fits with the predictions of the blinking vortex model: with oppositely-placed cilia, there is greater control over the spacing between

the vortices since the cilia are not at risk of colliding. As the blinking vortex model predicts better chaotic behavior from closer vortex spacings, this additional design freedom could be beneficial for mixing devices. However, experimental study is necessary to confirm these results. Future work should investigate the effect of varying the cilium length, intercilia spacing, actuation speeds, and cilia orientation to assess the performance for specific tasks.

6.2.2 Miniaturization

While the experimental system presented in Chapter 4 was designed at a relatively large scale compared to other cilia devices, the excellent matching of the finite element model to the experimental placement results suggests that it can now be used to predict performance under for different parameters, e.g. when scaling the system down in size, or for different electromagnet designs that change the relative influence of the in-plane and out-of-plane field. For example, miniaturized designs could take advantage of planar MEMS electromagnet coils [34] that have lower normal magnetic field components, reducing the impact of contact.

6.2.3 Iterative Machine Learning

The iterative machine learning procedure developed in Chapter 5 has many interesting future directions. For nonlinear systems, the frequency-domain approach presented here may not be viable (i.e. the region of convergence may be small), as the nonlinearity violates assumptions on superposition. However, Gaussian Process regression has been shown to be effective for time-domain inversion of nonlinear filters in simulation [12] and may prove effective for control as well.

BIBLIOGRAPHY

[1] Microarrays: Table of suppliers. *Nature*, 442(7106):1071–1072, aug 2006.

[2] Björn A Afzelius. The fine structure of the cilia from ctenophore swimming-plates. *The Journal of biophysical and biochemical cytology*, 9(2):383–394, 1961.

[3] H.-S. Ahn, Y. Q. Chen, and K. L. Moore. Iterative learning control: Brief survey and categorization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(6):1099–1121, 2007.

[4] Naum Ilyich Akhiezer. *Elements of the theory of elliptic functions*, volume 79. American Mathematical Soc., 1990.

[5] Isaac Amidror. Scattered data interpolation methods for electronic imaging systems: a survey. *Journal of electronic imaging*, 11(2):157–176, 2002.

[6] Hassan Aref. Stirring by chaotic advection. *Journal of fluid mechanics*, 143:1–21, 1984.

[7] Ch Atkeson and Joseph McIntyre. Robot trajectory learning through practice. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 1737–1742. IEEE, 1986.

[8] Klaus-Jürgen Bathe. *Finite element procedures*. Klaus-Jurgen Bathe, 1996.

[9] Jacob Belardi, Nicolas Schorr, Oswald Prucker, and Jürgen Rühe. Artificial cilia: generation of magnetic actuators in microfluidic systems. *Advanced Functional Materials*, 21(17):3314–3320, 2011.

[10] Lennart Blanken, Frank Boeren, Dennis Bruijnen, and Tom Oomen. Batch-to-Batch Rational Feedforward Control: From Iterative Learning to Identification Approaches, With

Application to a Wafer Stage. *IEEE-ASME TRANSACTIONS ON MECHATRONICS*, 22(2):826–837, APR 2017.

[11] Joost Bolder, Tom Oomen, Sjirk Koekebakker, and Maarten Steinbuch. Using iterative learning control with basis functions to compensate medium deformation in a wide-format inkjet printer. *MECHATRONICS*, 24(8):944–953, DEC 2014.

[12] Rafael Boloix-Tortosa, F. Javier Payan-Somet, Eva Arias-de-Reyna, and Juan José Murillo-Fuentes. Proper complex gaussian processes for regression. *CoRR*, abs/1502.04868, 2015.

[13] Antonio JF Bombard, Marcelo Knobel, Maria Regina Alcantara, and Ines Joekes. Evaluation of magnetorheological suspensions based on carbonyl iron powders. *Journal of intelligent material systems and structures*, 13(7-8):471–478, 2002.

[14] Christopher Brennen and Howard Winet. Fluid mechanics of propulsion by cilia and flagella. *Annual Review of Fluid Mechanics*, 9(1):339–398, 1977.

[15] Mark Butcher and Alireza Karimi. Linear Parameter-Varying Iterative Learning Control With Application to a Linear Motor System. *IEEE-ASME TRANSACTIONS ON MECHATRONICS*, 15(3):412–420, JUN 2010.

[16] Anthony Carpentier, Nicolas Galopin, Olivier Chadebec, Gérard Meunier, and Christophe Guérin. Application of the virtual work principle to compute magnetic forces with a volume integral method. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 27(3):418–432, 2014.

[17] Federico Carpi, Roy Kornbluh, Peter Sommer-Larsen, and Gursel Alici. Electroactive polymer actuators as artificial muscles: are they ready for bioinspired applications? *Bioinspiration & biomimetics*, 6(4):045006, 2011.

[18] Julyan HE Cartwright, Oreste Piro, and Idan Tuval. Fluid-dynamical basis of the embryonic development of left-right asymmetry in vertebrates. *Proceedings of the National Academy of Sciences of the United States of America*, 101(19):7234–7239, 2004.

[19] Chia-Yuan Chen, Chia-Yun Chen, Cheng-Yi Lin, and Ya-Ting Hu. Magnetically actuated artificial cilia for optimum mixing performance in microfluidics. *Lab on a Chip*, 13(14):2834–2839, 2013.

[20] Baratunde A Cola, David K Schaffer, Timothy S Fisher, and Mark A Stremler. A pulsed source-sink fluid mixing device. *Microelectromechanical Systems, Journal of*, 15(1):259–266, 2006.

[21] Naïs Coq, Sandrine Ngo, Olivia Du Roure, Marc Fermigier, and Denis Bartolo. Three-dimensional beating of magnetic microrods. *Physical Review E*, 82(4):041503, 2010.

[22] Michaël De Volder and Dominiek Reynaerts. Pneumatic and hydraulic microactuators: a review. *Journal of Micromechanics and microengineering*, 20(4):043001, 2010.

[23] J. den Toonder, F. Bos, D. Broer, L. Filippini, M. Gillies, J. de Goede, T. Mol, M. Reijme, W. Talen, H. Wilderbeek, et al. Artificial cilia for active micro-fluidic mixing. *Lab Chip*, 8(4):533–541, 2008.

[24] Jaap M.J. den Toonder and Patrick R. Onck. Microfluidic manipulation with artificial/bioinspired cilia. *Trends in Biotechnology*, 31(2):85 – 91, 2013.

[25] *NIST Digital Library of Mathematical Functions*. http://dlmf.nist.gov/, Release 1.0.13 of 2016-09-16. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller and B. V. Saunders, eds.

[26] Vincent Dugas, Jérôme Broutin, and Eliane Souteyrand. Mixing and dispensing homogeneous compounds of a reactant on a surface, August 10 2010. US Patent 7,772,010.

[27] A Engel and R Friedrichs. On the electromagnetic force on a polarizable body. *American Journal of Physics*, 70(4):428–432, 2002.

[28] BA Evans, AR Shields, R.L. Carroll, S. Washburn, MR Falvo, and R. Superfine. Magnetically actuated nanorod arrays as biomimetic cilia. *Nano letters*, 7(5):1428–1434, 2007.

[29] C. T. Freeman, P. L. Lewin, E. Rogers, D. H. Owens, and J. J. Hatonen. Discrete fourier transform based iterative learning control design for linear plants with experimental verification. *ASME Journal of Dynamic Systems, Measurement, and Control*, 131(3):031006–1 – 031006–10, May,2009.

[30] X. Gao and S. Mishra. An iterative learning control algorithm for portability between trajectories. In *2014 American Control Conference, Portland, OR*, pages 3808–3813, June 4-6 June, 2014.

[31] Erik M Gauger, Matthew T Downton, and Holger Stark. Fluid transport at low reynolds number with magnetically actuated artificial cilia. *The European Physical Journal E*, 28(2):231–242, 2009.

[32] J. Ghosh and B. Paden. Iterative learning control for nonlinear nonminimum phase plants. *ASME Journal of Dynamic Systems, Measurement, and Control*, 123:21–20, March,2001.

[33] Benjamin Gorissen, Michaël de Volder, and Dominiek Reynaerts. Pneumatically-actuated artificial cilia array for biomimetic fluid propulsion. *Lab on a Chip*, 15(22):4348–4355, 2015.

[34] Gary D Gray and Paul A Kohl. Magnetically bistable actuator: Part 1. ultra-low switching energy and modeling. *Sensors and Actuators A: Physical*, 119(2):489–501, 2005.

[35] B. Guirao and J.F. Joanny. Spontaneous creation of macroscopic flow and metachronal waves in an array of cilia. *Biophysical journal*, 92(6):1900–1917, 2007.

[36] J. Hatonen, T.J. Harte, D.H. Owens, J. Ratcliffe, P. Lewin, and E. Rogers. Iterative learning control - what is it all about? {*IFAC*} *Proceedings Volumes*, 37(12):547 – 553, 2004. {IFAC} Workshop on Adaptation and Learning in Control and Signal Processing (ALCOSP 04) and {IFAC} Workshop on Periodic Control Systems (PSYCO 04), Yokohama, Japan, 30 August - 1 September, 2004.

[37] H Havlicsek and A Alleyne. Nonlinear control of an electrohydraulic injection molding machine via iterative adaptive learning. *IEEE-ASME TRANSACTIONS ON MECHATRONICS*, 4(3):312–323, SEP 1999.

[38] John Illingworth and Josef Kittler. The adaptive hough transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (5):690–698, 1987.

[39] Scott W. Jones and Hassan Aref. Chaotic advection in pulsed source–sink systems. *Physics of Fluids*, 31(3):469–485, 1988.

[40] SN Khaderi, MGHM Baltussen, PD Anderson, JMJ den Toonder, and PR Onck. Breaking of symmetry in microfluidic propulsion driven by artificial cilia. *Physical Review E*, 82(2):027302, 2010.

[41] SN Khaderi, CB Craus, J. Hussong, N. Schorr, J. Belardi, J. Westerweel, O. Prucker, J. Rühe, JMJ Den Toonder, and PR Onck. Magnetically-actuated artificial cilia for microfluidic propulsion. *Lab Chip*, 11(12):2002–2010, 2011.

[42] SN Khaderi, JMJ den Toonder, and PR Onck. Fluid flow due to collective non-reciprocal motion of symmetrically-beating artificial cilia. *Biomicrofluidics*, 6(1):014106, 2012.

[43] Syed Khaderi, Jeanette Hussong, Jerry Westerweel, Jaap den Toonder, and Patrick Onck. Fluid propulsion using magnetically-actuated artificial cilia–experiments and simulations. *Rsc Advances*, 3(31):12735–12742, 2013.

[44] Vinayak V Khatavkar, Patrick D Anderson, Jaap MJ den Toonder, and Han EH Meijer. Active micromixer based on artificial cilia. *Physics of Fluids (1994-present)*, 19(8):083605, 2007.

[45] K.-S. Kim and Q. Zou. A modeling-free inversion-based iterative feedforward control for precision output tracking of linear time-invariant systems. *IEEE/ASME Transactions on Mechatronics*, 18(6):1767–1777, Dec. 2013.

[46] Jiradech Kongthon. *Modeling and Control of Biomimetic Cilia-Based Devices for Microfluidic Applications*. UNIVERSITY OF WASHINGTON, 2011.

[47] Jiradech Kongthon and Santosh Devasia. Iterative control of piezoactuator for evaluating biomimetic, cilia-based micromixing. *IEEE/ASME Transactions on Mechatronics*, 18(3):944–953, jun 2013.

[48] Sanboh Lee, HY Lee, IF Lee, and CY Tseng. Ink diffusion in water. *European journal of physics*, 25(2):331, 2004.

[49] Hungwen Li, Mark A Lavin, and Ronald J Le Master. Fast hough transform: A hierarchical approach. *Computer Vision, Graphics, and Image Processing*, 36(2-3):139–161, 1986.

[50] Andrew J. McDaid, Kean C. Aw, Enrico Haemmerle, and Sheng Q. Xie. Control of IPMC actuators for microfluidics with adaptive "online" iterative feedback tuning. *IEEE/ASME Transactions on Mechatronics*, 17(4):789–797, aug 2012.

[51] Mark K McQuain, Kevin Seale, Joel Peek, Timothy S Fisher, Shawn Levy, Mark A Stremler, and Frederick R Haselton. Chaotic mixer improves microarray hybridization. *Analytical Biochemistry*, 325(2):215–226, 2004.

[52] L Meirovitch. *Fundamentals of Vibrations*. McGraw Hill, 2001.

[53] SG Mikhlin. *Multidimensional Singular Integrals and Integral Equations*. Pergamon Press, 1965.

[54] Sandipan Mishra and Masayoshi Tomizuka. Projection-Based Iterative Learning Control for Wafer Scanner Systems. *IEEE-ASME TRANSACTIONS ON MECHATRONICS*, 14(3):388–393, JUN 2009.

[55] William Daniel Moore and Th W Engelmann. On ciliary motion. *Journal of anatomy and physiology*, 3(Pt 2):420, 1869.

[56] Rahim Mutlu, Gursel Alici, and Weihua Li. A Soft Mechatronic Microstage Mechanism Based on Electroactive Polymer Actuators. *IEEE-ASME TRANSACTIONS ON MECHATRONICS*, 21(3):1467–1478, JUN 2016.

[57] M. Norrlof. An adaptive iterative learning control algorithm with experiments on an industrial robot. *IEEE Transactions on Robotics and Automation*, 18(2):245–251, Apr 2002.

[58] K. Oh, J.H. Chung, S. Devasia, and J.J. Riley. Bio-mimetic silicone cilia for microfluidic manipulation. *Lab on a Chip*, 9(11):1561–1566, 2009.

[59] International Commission on Illumination. *Recommendations on Uniform Color Spaces, Color-difference Equations, Psychometric Color Terms*. CIE publication. CIE, 1978.

[60] Edward L Paul, Victor A Atiemo-Obeng, and Suzanne M Kresta. *Handbook of industrial mixing: science and practice*. John Wiley & Sons, 2004.

[61] Edward M Purcell. Life at low reynolds number. *Am. J. Phys*, 45(1):3–11, 1977.

[62] Mona Rahbar, Lesley Shannon, and Bonnie L Gray. Microfluidic active mixers employing ultra-high aspect-ratio rare-earth magnetic nano-composite polymer artificial cilia. *Journal of Micromechanics and Microengineering*, 24(2):025003, 2014.

[63] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, MA, 2006.

[64] Manuel Ritto-Corrêa and Dinar Camotim. On the arc-length and other quadratic control methods: Established, less known and new implementation procedures. *Computers & Structures*, 86(11):1353–1368, 2008.

[65] Maureen Sartor, Jennifer Schwanekamp, Danielle Halbleib, Ismail Mohamed, Saikumar Karyala, Mario Medvedovic, and Craig R Tomlinson. Microarray results improve significantly as hybridization approaches equilibrium. *Biotechniques*, 36(5):790–796, 2004.

[66] J Simkin and CW Trowbridge. On the use of the total scalar potential on the numerical solution of fields problems in electromagnetics. *International journal for numerical methods in engineering*, 14(3):423–440, 1979.

[67] Michael Stangegaard. *DNA Microarrays for Biomedical Research: Methods and Protocols*, chapter Gene Expression Analysis Using Agilent DNA Microarrays, pages 133–145. Humana Press, Totowa, NJ, 2009.

[68] Doris Steger, David Berry, Susanne Haider, Matthias Horn, Michael Wagner, Roman Stocker, and Alexander Loy. Systematic spatial bias in dna microarray hybridization is caused by probe spot position-dependent variability in lateral diffusion. *PloS one*, 6(8):e23727, 2011.

[69] Rob Sturman, Julio M. Ottino, and Stephen Wiggins. *The Mathematical Foundations of Mixing*. Cambridge University Press (CUP), 2006.

[70] S.L. Tamm. Mechanisms of ciliary co-ordination in ctenophores. *Journal of Experimental Biology*, 59(1):231–245, 1973.

[71] CR Thomas and D Geer. Effects of shear on proteins in solution. *Biotechnology letters*, 33(3):443–456, 2011.

[72] S. Tien, Qingze Zou, and S. Devasia. Control of dynamics-coupling effects in piezo-actuator for high-speed afm operation. In *Proceedings of the 2004 American Control Conference*, volume 4, pages 3116–3121 vol.4, June 2004.

[73] Szuchi Tien, Qingze Zou, and Santosh Devasia. Iterative control of dynamics-coupling-caused errors in piezoscanners during high-speed afm operation. *IEEE Transactions on Control Systems Technology*, 13(6):921–931, 2005.

[74] Meng-Shiun Tsai, Chung-Liang Yen, and Hong-Tzong Yau. Integration of an Empirical Mode Decomposition Algorithm With Iterative Learning Control for High-Precision Machining. *IEEE-ASME TRANSACTIONS ON MECHATRONICS*, 18(3):878–886, JUN 2013.

[75] Zheng Wang, Panagiotis Polygerinos, Johannes T. B. Overvelde, Kevin C. Galloway, Katia Bertoldi, and Conor J. Walsh. Interaction Forces of Soft Fiber Reinforced Bending Actuators. *IEEE-ASME TRANSACTIONS ON MECHATRONICS*, 22(2):717–727, APR 2017.

[76] Cheng-Wey Wei, Ji-Yen Cheng, Chih-Ting Huang, Meng-Hua Yen, and Tai-Horng Young. Using a microfluidic device for 1 μl dna microarray hybridization in 500 s. *Nucleic acids research*, 33(8):e78–e78, 2005.

[77] Stephen Wiggins and Julio M Ottino. Foundations of chaotic mixing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 362(1818):937–970, 2004.

[78] Xingzhou Yang, Robert H Dillon, and Lisa J Fauci. An integrative computational model of multiciliary beating. *Bulletin of mathematical biology*, 70(4):1192–1215, 2008.

Appendix A

COMPUTER PROGRAMS

A.1 Blinking vortex simulation

This file implements the blinking vortex simulation for a rectangular or circular chamber and provides functions for plotting the iterated tracer locations.

```
function [t,cv] = sim_blinking_vortex(method,mu,times,Ntr)
    % Blinking vortex simulation.

    default('Ntr',200);
    if length(times) > 2
        subfigs = true;
        plotdim = [3 3];
    else
        subfigs = false;
        plotdim = [1 1];
    end
    wplane_axis = [-0.0065 0.0021 -0.0065 0];
    % Choose beta, mu
    a = 1; G = 2*pi;
    b = 0.5; % vortex distance from center
    Nrep = max(1,floor(mu/2));
    Nrep = 1;

    W = 25.48;
    H = 17.69;

    nybin = 400;
```

```
nxbin = round(W/H*nybin);
swapdims = false;

% Tracer ICs
tr = a/8;
[tx,ty] = meshgrid(linspace(-tr, tr, 1e2),linspace(0,2*tr,1e2));
tx = tx(:); ty = ty(:);

makeplots = nargout==0;
lotsofplots = false;
tonsofplots = false;
streamlineplots = true;

if makeplots && subfigs, f = figure; end

ttotal = tic;
switch method
     case 'contour'
          % just plot the contours and get out
          method = 'analytical';
          Ng = 1e3;
          [x,y] = meshgrid(linspace(-a,a,Ng),linspace(-a,a,Ng));
          [xp,yp,gp] = reflections(b,0,G);
          S = pvsf(x,y,xp,yp,gp);
          S2 = pvsf(x,y,-xp,-yp,gp);
          R = sqrt(x.^2+y.^2);
          smax = max(S(R<=a)); smin = min(S(R<=a));
          figure; plot_tracers(NaN,NaN);
          figure; plot_tracers(NaN,NaN); hold on;
          contour(x,y,S,linspace(smin,smax,50));
          figure; plot_tracers(NaN,NaN); hold on;
          contour(-x,-y,S,linspace(smin,smax,50));
          figure; plot_tracers(NaN,NaN); hold on;
```

```
    S = S+S2;
    smax = max(S(R<=a)); smin = min(S(R<=a));
    contour(x,y,S,linspace(smin,smax,50));
    return
case 'numerical'
    Ng = 1.5e3; % Number of cartesian grid points
    Nr = 1e3; % R discretization
    Nt = 1000; % Theta discretization
    Ns = 1000; % Streamfunction discretization
    opts = odeset('AbsTol',1e-15,'RelTol',1e-12);
    [rmesh,thmesh] =
     ↪  meshgrid(linspace(1/Nr/100,max(abs(a-b),abs(-a-b))+0.0001,Ng),linspace(0,2*pi,Ng));
    [rmap,thmap] = map_values(rmesh,thmesh,b,0,mu/2);
    map = @numerical_map;
case 'analytical'
    map = @analytical_map;
case 'analytical-interp'
    % Compute the analytical map at the grid points, but then
    % interpolate.  A way of assessing the effect of interpolation
    % on the mapping.
    Ng = 3e3; % Number of cartesian grid points
    [xmesh,ymesh] = meshgrid(linspace(-a,a,Ng),linspace(-a,a,Ng));
    [xm,ym] = analytical_map(xmesh(:),ymesh(:),mu/2);
    xmap1 = xmesh; ymap1 = ymesh;
    xmap1(:) = xm; ymap1(:) = ym;
    map = @numerical_map;
case 'rectangular-interp'
    % The same as the numerical case, but in a rectangular chamber.
    W = 25.48; %mm
    H = 18.2; %mm
    G = -425; %mm^2/s
    x0 = -W/2+7; %mm
    y0 = 15.5-H/2; %mm
```

```
    Ng = 1.5e3; % Number of cartesian grid points
    Nr = 1e3; % R discretization
    Nt = 1000; % Theta discretization
    Ns = 1000; % Streamfunction discretization
    tr = H/6;
    os = H/10;
    [tx1,ty1] = meshgrid(linspace(-W/2+os, -W/2+os+tr,
     ↪  1e2),linspace(H/2-os-tr,H/2-os,1e2));
    [tx2,ty2] = meshgrid(linspace(W/2-os-tr, W/2-os,
     ↪  1e2),linspace(H/2-os-tr,H/2-os,1e2));
    tx1 = tx1(:); ty1 = ty1(:);
    tx2 = tx2(:); ty2 = ty2(:);
    tx = {tx1, tx2}; ty = {ty1, ty2};

    opts = odeset('AbsTol',1e-15,'RelTol',1e-12,'MaxStep',0.02);
    Rmax = max(sqrt((x0+[-1;1;1;-1]*W/2).^2+(y0+[-1;-1;1;1]*H/2).^2));
    [rmesh,thmesh] = meshgrid(linspace(1/Nr/100,Rmax,Ng),linspace(0,2*pi,Ng));
    [rmap,thmap] = map_values(rmesh,thmesh,x0,y0,mu/2);
    map = @numerical_map;
case {'rectangular', 'rectangular-opposed'}
    % Blinking vortex in a rectangular chamber via Weierstrass
    % elliptic function conformal map to half-plane.

    % rectangular-opposed: Same as rectangular but with cilia in
    % opposite directions.  This means instead of (x0,y0) and
    % (W-x0,y0), the vortices should be at (x0,y0) and (W-x0,H-y0)
    % and the two vortices should be in the same direction.

    G = -350; %mm^2/s
    x0 = 6*.9;
    y0 = 15;
    Ng = 2e3; % Number of cartesian grid points
```

```
        % Get the constants for the domain
        [~,~,g2,g3,e1,e2,e3] = weierstrass_rho(x0+y0*1i,W,H);

        tr = H/3;
        os_x = H/100;
        os_y = H/100;
        avg_tracer_density = Ntr^2/nxbin/nybin;
        initial_tracer_density = Ntr^2/(tr^2/W/H*nxbin*nybin);
        cscl = 128/(initial_tracer_density);

        timers.ode = 0;
        timers.Pinv = 0;
        timers.newton = 0;

        [tx1,ty1] = meshgrid(linspace(os_x,os_x+tr,
         ↪  Ntr),linspace(H-os_y-tr,H-os_y,Ntr));
        if strcmpi(method,'rectangular')
             [tx2,ty2] = meshgrid(linspace(W-os_x-tr, W-os_x,
              ↪  Ntr),linspace(H-os_y-tr,H-os_y,Ntr));
        elseif strcmpi(method,'rectangular-opposed')
             [tx2,ty2] = meshgrid(linspace(W-os_x-tr, W-os_x,
              ↪  Ntr),linspace(os_y+tr,os_y,Ntr));
        end

        tx = [tx1(:);tx2(:)]; ty = [ty1(:);ty2(:)];
        tseg = [ones(numel(tx1),1);2*ones(numel(tx2),1)];

        xbins = linspace(0,W,nxbin);
        ybins = linspace(0,H,nybin);
        map = @theta_map;
    otherwise
        error('unknown method.')
end
```

```
x = tx; y = ty;
cv = getcv(x,y);
time = 0;
if makeplots && lotsofplots
     figure; plot_tracers(x,y);
     figure; plot_tracers_wplane(x,y);
end
[x,y] = iterate_map(x,y,times(1));

for j = 1:length(times)

     if makeplots
          if lotsofplots
               figure(f);
          end
          if subfigs
               ax = subaxis(plotdim(1),plotdim(2),j,'Spacing',0.01);
          else
               figure;
          end
          plot_tracers(x,y);
     end
     if j < length(times), [x,y] = iterate_map(x,y,times(j+1)-times(j)); end
end
toc(ttotal)
fprintf('ODE45: %g seconds\nNewton method: %g seconds\nWeierstrass inverse: %g
 ↪  seconds\n',...
     timers.ode,timers.newton,timers.Pinv)

if makeplots && subfigs
     drawnow
     AR = get(ax,'PlotBoxAspectRatio');
```

```
    sz = get(f,'PaperPosition');
    fig_AR = ((1.01*plotdim(2)-0.01)*AR(1))/((1.01*plotdim(1)-0.01)*AR(2));

    % Scale the height-- this is important because it matches how the
    % layout will happen when putting the resulting figure into an
    % article (i.e. the width is fixed to be the column width but the
    % height is flexible). Then the font size will at least be consistent
    % between figures.
    set(f,'PaperPosition',[sz(1) sz(1) 1.3*sz(3) 1.3*sz(3)/fig_AR])
    set(f,'PaperPosition',[sz(1) sz(1) 1*sz(3) 1*sz(3)/fig_AR])

    set(f,'PaperPosition',[0 0 4*plotdim(2) 4*plotdim(1)]);
end

t = time;
if length(times) == 1 && nargout == 1
    t = ax.Parent;
end

% Analytical map from Aref 1984
function [x,y] = analytical_map(x0,y0,~,~,t)
    z0 = x0+1i*y0;
    z1 = b; z2 = a^2/b;
    lam = abs((z0-z1)./(z0-z2));
    rho = lam./(1-lam.^2)*(z2-z1);
    zc = (z1-lam.^2*z2)./(1-lam.^2);
    T_l = 4*pi^2.*rho.^2/G.*((1+lam.^2)./(1-lam.^2));
    Al = @(phi,lam) phi-2*lam./(1+lam.^2).*sin(phi);
    dAl = @(phi,lam) 1-2*lam./(1+lam.^2).*cos(phi);
    phi0 = mod(angle(z0-zc),2*pi);

    % Initial Newton Method step
    err = @(phi) Al(phi,lam) - Al(phi0,lam) - 2*pi*t./T_l;
```

```
    derr = @(phi) dAl(phi,lam);
    phi = phi0-err(phi0)./derr(phi0);

    w = .5; % step size modifier

    % Bisection method on first two Newton points
    phi = (phi0+phi)/2;

    while norm(err(phi)) > 1e-6
        phi = phi - w*err(phi)./derr(phi);
    end
    z = zc+rho.*exp(1i*phi);
    x = real(z);
    y = imag(z);
end

function [S,w] = streamlines_w(x1,y1,G)
    z1 = x1+1i*y1;

    [xg,yg] = meshgrid(linspace(0,W,200),linspace(0,H,200));
    % Conformal map to lower half-plane
    [w1,~,g2,g3] = weierstrass_rho(z1,W,H);
    w = weierstrass_rho(xg+1i*yg,W,H);

    S = G/2/pi*log(abs((w-w1)./(w-conj(w1))));
end

function [x,y] = theta_map(x0,y0,x1,y1,t,G)
    z0 = x0+1i*y0;
    z1 = x1+1i*y1;

    % Conformal map to lower half-plane
    [w1,~,g2,g3] = weierstrass_rho(z1,W,H);
```

```
w0 = weierstrass_rho(z0,W,H);

% Streamlines in w-space
lam = abs((w0-w1)./(w0-conj(w1)));
rho = lam./(1-lam.^2)*abs(conj(w1)-w1);
wc = (w1-lam.^2*conj(w1))./(1-lam.^2);
Tl = (1+lam.^2)./(1-lam.^2)*4*pi^2.*rho.^2./G;
th0 = mod(angle(w0-wc),2*pi);

% Integrate dynamics in w-space
tic;
if ~lotsofplots
     TODE = [0 t/2 t];
else
     TODE = linspace(0,t,50);
end
[~, THODE] = ode45(...
     @(t,th) omega(th,lam(:),wc(:),rho(:),Tl(:),g2,g3),...
     TODE,th0(:),odeset('RelTol',1e-3));
timers.ode = timers.ode + toc;
if tonsofplots
     for row = 1:size(THODE,1)
          figure;
          plot_tracers_wplane_w(wc(:)+rho(:).*exp(1i*THODE(row,:).'));
          axis([-0.0065 0.0021 -0.0065 0])
          hl = hline(0,'k-'); hl.LineWidth = 1;
          print('-depsc2',sprintf('papers/DSCC2016-talk/figs/sim-wplane-%d.eps',row));
          close
     end
end
tonsofplots = false;

THODE = THODE(end,:); THODE = THODE(:);
```

```
TW = wc(:)+rho(:).*exp(1i*THODE);
if any(imag(TW)>0), fprintf('%d out-of-bounds in w-plane.\n',nnz(imag(TW)> 0)), end

% Now map back to z-space by inverting tw = rho(tz). First do a few
% steps of Newton method, then use the elliptic integral of the first
% kind to solve the remaining points.  Newton's method is much faster
% if it converges (and it typically does for most particles) so it
% saves time over just using the elliptic integral everywhere.
TZ = z0; % Initial guess for Newton method
tic;
[r,dr] = weierstrass_rho(TZ,W,H);
err = r-TW;
derr = dr;

nit = 1;
while norm(err) > 1e-12 && nit < 15
     TZ = TZ-err./derr; % Newton update
     TZ = mod(real(TZ),2*W)+1i*mod(imag(TZ),2*H); % Periodicity of w

     % Compute new error
     [r,dr] = weierstrass_rho(TZ,W,H);
     err = r-TW; derr = dr;

     nit = nit+1; % Increment step counter
end
timers.newton = timers.newton + toc;
oob = real(TZ) > W | imag(TZ) > H; % Converged outside the rectangle
badinv = abs(err) > 1e-6; % Error is still too high
fixme = oob | badinv;
if any(fixme) % Elliptic integral to solve remaining points
     tic;
     TZ(fixme) = (e1-e3)^-.5*ellipticF(... % P-inverse as elliptic integral F
          asin(((TW(fixme)-e3)./(e1-e3)).^-.5),...
```

```
                (e2-e3)/(e1-e3)...
                );
            timers.Pinv = timers.Pinv + toc;
        end

        x = real(TZ); y = imag(TZ);
    end

    function CV = getcv(x,y)
        ct = max(0,min(255,128+cscl*(...
            histcounts2(x(tseg==1),y(tseg==1),xbins,ybins) - ...
            histcounts2(x(tseg==2),y(tseg==2),xbins,ybins)...
            )));
        CV = std(ct(:))./mean(ct(:));
    end

    function thd = omega(theta,lambda,wc,rad,Tl,g2,g3)
        w = wc+rad.*exp(1i*theta);
        thd = 2*pi*abs(4*w.^3-g2*w-g3)./(Tl.*(1-2*lambda./(1+lambda.^2).*sin(theta)));
    end

    function [rfn,omfn,Sfn] = polar_dynamics(xc,yc)
        % Return:
        %       - R as a function of th,S
        %       - omega as a function of th,S
        %       - S as a function of x,y

        xg = xc + rmesh.*cos(thmesh);
        yg = yc + rmesh.*sin(thmesh);

        switch method
            case {'analytical','numerical'}
                [xp,yp,Gp] = reflections(xc,yc,G);
```

```
            [S,vx,vy] = pvsf(xg,yg,xp,yp,Gp);
        otherwise
            z1 = xc+W/2+1i*(yc+H/2);
            z = xg+W/2+1i*(yg+H/2);
            r1 = weierstrass_rho(z1,W,H);
            [r,dr] = weierstrass_rho(z,W,H);
            v = G/2/pi/1i*dr.*(r1-conj(r1))./((r-r1).*(r-conj(r1)));
            v(isnan(v)) = 0;
            vx = real(v); vy = -imag(v);
            S = G/2/pi*real(log(r-conj(r1))-log(r-r1));
            S(isnan(S)) = 0;
    end

    Sfn = @(xx,yy)
     ↪  interp2(rmesh,thmesh,S,sqrt((xx-xc).^2+(yy-yc).^2),mod(atan2(yy-yc,xx-xc),2*pi));
    switch method
        case {'numerical','analytical'}
            inbounds = sqrt(xg.^2+yg.^2)<=a;
            smin = min(S(inbounds));
            smax = max(S(inbounds));
            inbounds = 1:numel(S);
        case 'rectangular'
            inbounds = abs(xg) <= 1.001*W/2 & abs(yg) <= 1.001*H/2;
            smin = max(min(S,[],2));
            smax = min(max(S,[],2));
    end

    [thgrid,sgrid] = meshgrid(linspace(0,2*pi,Nt),linspace(smin,smax,Ns));

    %  compute r at the (th,s) grid points
    rgrid = griddata(thmesh(inbounds),S(inbounds),rmesh(inbounds),thgrid,sgrid);
    rfn = @(th,s) interp2(thgrid,sgrid,rgrid,mod(th,2*pi),s,'cubic');
```

```
    % ω(θ,s): angular velocity at the thgrid, sgrid points. The angular
    % component of the linear velocity is given by a rotation
    %      [Vn] = [  cos(θ) sin(θ) ] [Vx]
    %      [Vt] = [ -sin(θ) cos(θ) ] [Vy]
    % and omega = Vt/R.
    om = -vx.*sin(thmesh)./rmesh+vy.*cos(thmesh)./rmesh;
    omgrid = griddata(thmesh(inbounds),S(inbounds),om(inbounds),thgrid,sgrid);
    omfn = @(th,s) interp2(thgrid,sgrid,omgrid,mod(th,2*pi),s,'cubic',0);
end

function [rmap, thmap] = map_values(r,th,xc,yc,T)
    [rf,of,sf] = polar_dynamics(xc,yc);
    s0 = sf(r(:).*cos(th(:))+xc,r(:).*sin(th(:))+yc);
    Tsim = T/Nrep;
    [~,TH] = ode45(@(t,th) of(th(:),s0),[0 Tsim/2 Tsim],th(:),opts);

    th1 = TH(end,:)';
    r1 = rf(th1,s0);
    thmap1 = th; rmap1 = r;
    thmap1(:) = th1(:); rmap1(:) = r1(:);
    rmap = rmap1; thmap = thmap1;
end

function [x,y] = numerical_map(x0,y0,xc,yc,~)
    m = 'cubic';

    r0 = sqrt((x0-xc).^2+(y0-yc).^2);
    th0 = mod(atan2(y0-yc,x0-xc),2*pi);
    r1 = interp2(rmesh,thmesh,rmap,r0,th0,m);
    th1 = interp2(rmesh,thmesh,thmap,r0,th0,m);
    x1 = r1.*cos(th1)+xc; y1 = r1.*sin(th1)+yc;
    x = x1; y = y1;
end
```

```
function [x,y] = iterate_map(x,y,N)
     tt = 0;
     cv;
     if iscell(x) && iscell(y)
          while tt < N
               progressbar(N-1,tt,'Mapping')
               for i = 1:length(x)
                    %[x{i},y{i}] = iterate_map(x{i},y{i},1);
                    [x{i},y{i}] = map(x{i},y{i},x0,y0,mu/2,G);
               end
               for i = 1:length(x)
                    %[x{i},y{i}] = iterate_map(x{i},y{i},1);
                    [x{i},y{i}] = map(x{i},y{i},W-x0,y0,mu/2,-G);
               end
               %cv(end+1) = getcv(x,y);
               tt = tt+1;
          end
          return
     elseif iscell(x) || iscell(y)
          error('Malformed input.');
     end

     progressbar(N,0,'Mapping')
     while tt < N % correction to prevent extra cycles
          switch method
               case {'numerical','analytical'}
                    [x,y] = map(x,y,b,0,mu/2);
                    [x,y] = map(-x,-y,b,0,mu/2);
                    x = -x; y = -y;
               case {'rectangular-interp'}
                    for i = 1:Nrep, [x,y] = map(x,y,x0,y0,mu/2); end
```

```
        %f = gcf; figure; plot_tracers(x,y); figure(f);
        x = -x;
        for i = 1:Nrep, [x,y] = map(x,y,x0,y0,mu/2); end
        x = -x;
    case 'rectangular'
        [x,y] = map(x,y,x0,y0,mu/2,G);
        cv(end+1) = getcv(x,y);
        time(end+1) = time(end)+mu/2;
        if makeplots && lotsofplots
            figure; plot_tracers(x,y);
            figure; plot_tracers_wplane(x,y);
        end
        [x,y] = map(x,y,W-x0,y0,mu/2,-G);
        cv(end+1) = getcv(x,y);
        time(end+1) = time(end)+mu/2;
        if makeplots && lotsofplots
            figure; plot_tracers(x,y);
            figure; plot_tracers_wplane(x,y);
        end
    case 'rectangular-opposed'
        [x,y] = map(x,y,x0,y0,mu/2,G);
        cv(end+1) = getcv(x,y);
        time(end+1) = time(end)+mu/2;
        if makeplots && lotsofplots
            figure; plot_tracers(x,y);
            figure; plot_tracers_wplane(x,y);
        end
        [x,y] = map(x,y,W-x0,H-y0,mu/2,G);
        cv(end+1) = getcv(x,y);
        time(end+1) = time(end)+mu/2;
        if makeplots && lotsofplots
            figure; plot_tracers(x,y);
            figure; plot_tracers_wplane(x,y);
```

```
                end

            end
            tt = tt+1;
            progressbar(N,tt,'Mapping')
        end
    end

    function plot_tracers(x,y)
        switch method
            case {'numerical','analytical'}
                plot(a*cos(0:pi/200:2*pi),a*sin(0:pi/200:2*pi),'k-','LineWidth',1);
                hold on
                plot([-b b],[0 0],'k+','MarkerSize',3);
            case {'rectangular-interp'}
                plot([-1 -1 1 1 -1]*W/2, [-1 1 1 -1 -1]*H/2,'k-','LineWidth',1);
                hold on
                plot([-x0 x0],[y0 y0],'k+','MarkerSize',3);
            case 'rectangular'
                plot([0 0 1 1 0]*W,[0 1 1 0 0]*H,'k-','LineWidth',1);
                hold on
                plot([x0 W-x0],[y0 y0],'k+','MarkerSize',3);
            case 'rectangular-opposed'
                plot([0 0 1 1 0]*W,[0 1 1 0 0]*H,'k-','LineWidth',1);
                hold on
                plot([x0 W-x0],[y0 H-y0],'k+','MarkerSize',3);
        end
        ax = gca;
        if ~iscell(x) && numel(unique(tseg)) == 1
            ax.ColorOrder = [0 0 0];
            if ~swapdims
                inp = {x y};
            else
```

```
            inp = {H-y W-x};
        end
    elseif iscell(x)
        ax.ColorOrder = [.2081 .1663 .5292;.7 0 0];
        inp = {};
        for i = 1:length(x)
            if ~swapdims
                inp = [inp x(i) y(i)]; %#ok
            else
                inp = [inp H-y(i) W-x(i)]; %#ok
            end
        end
    else
        inp = {};
        ax.ColorOrder = [.2081 .1663 .5292;.7 0 0];
        tuniq = unique(tseg);
        for i = 1:numel(tuniq)
            inp = [inp {x(tseg==tuniq(i)) y(tseg==tuniq(i))}]; %#ok
        end
    end
    alpha(scatter(x,y,5,[tseg(:)==1 tseg(:)==2]*[.2081 .1663 .5292;.7 0
     ↪  0],'filled'),0.2)
    axis off equal
end

function plot_tracers_wplane(x,y)
    w = weierstrass_rho(x+1i*y,W,H);
    plot_tracers_wplane_w(w);
end

function plot_tracers_wplane_w(w)
    w0 = weierstrass_rho(x0+1i*y0,W,H);
    w0_ = weierstrass_rho(W-x0+1i*y0,W,H);
```

```
    w_1 = w(tseg==1); w_2 = w(tseg==2);

    plot([real(w0) real(w0_)],[imag(w0) imag(w0_)],'k+'); hold on
    ax = gca;
    ax.ColorOrder = [.2081 .1663 .5292;.7 0 0];
    plot(ax,real(w_1), imag(w_1), real(w_2), imag(w_2),...
        'MarkerSize',.1,'Marker','.','LineStyle','none');
    axis equal
    axis off

    if streamlineplots
        hold on
        [S,wg] = streamlines_w(x0,y0,G);
        Ssample = sort(S(abs(real(wg)-real(w0))<1e-2& imag(wg)>imag(w0)));
        Ssample = linspace(min(Ssample),max(Ssample)*.6,20);
        cn = contour(real(wg),imag(wg),S,Ssample,'k');
    end

    axis(wplane_axis);
    hl = hline(0,'k'); hl.LineWidth = 1;
end

function [xr,yr,Gr] = reflections(x,y,G)
    switch method
        case 'rectangular'
            % get the reflections of the points in the vectors x,y about the four
            % walls of the rectangular chamber
            layers = 30;

            % Need to order these so that opposite pairs are processed
            % together to avoid overflow
            [Gx,Gy] = meshgrid(-layers:layers);
            xr = (-1).^Gx*x0+Gx*W;
```

```
            yr = (-1).^Gy*y0+Gy*H;
            Gr = (-1).^(Gx+Gy)*G;
            xr = xr(:); yr = yr(:); Gr = Gr(:);
            [~,o] = sortrows([abs(xr) abs(yr)],[-1 -2]);
            xr = xr(o); yr = yr(o); Gr = Gr(o);
            Gx = Gx(o); Gy = Gy(o);

            diamond = abs(Gx)+abs(Gy) <= layers;

            xr = xr(diamond); yr = yr(diamond);
            Gr = Gr(diamond);
        case {'numerical','analytical'}
            xr = [x real(a^2/conj(x+y*1i))];
            yr = [y imag(a^2/conj(x+y*1i))];
            Gr = [G -G];
    end
end

function plot_map()
    f = gcf;
    figure;

    switch method
        case {'numerical','analytical'}
            [xmesh,ymesh] = meshgrid(linspace(-a,a,1000),linspace(-a,a,1000));
            [xmap, ymap] = map(xmesh,ymesh,b,0,mu/2);
        case 'rectangular'
            [xmesh,ymesh] = meshgrid(linspace(-W/2,W/2,Ng),linspace(-H/2,H/2,Ng));
            [xmap, ymap] = map(xmesh,ymesh,x0,y0,mu/2);
    end
    nnz(isnan(xmap))
    imagesc(xmesh(1,:),ymesh(:,1),xmap);
    axis equal tight
```

```
        colorbar
        figure(f);
    end

    % Define vortex streamlines
    function [S,vx,vy] = pvsf(x,y,xi,yi,Gi)
        S = 1;
        vx = 0; % vx = dS/dy
        vy = 0; % vy = -dS/dx
        for i = 1:length(xi)
            if Gi(i) > 0
                S = S./realsqrt((x-xi(i)).^2+(y-yi(i)).^2);
            else
                S = S.*realsqrt((x-xi(i)).^2+(y-yi(i)).^2);
            end

            vx = vx-Gi(i)/2/pi*(y-yi(i))./((x-xi(i)).^2+(y-yi(i)).^2);
            vy = vy+Gi(i)/2/pi*(x-xi(i))./((x-xi(i)).^2+(y-yi(i)).^2);
        end
        S = abs(G)/2/pi*log(S);
    end


end
```

A.2 Finite Element modeling

A.2.1 Main control function

This function is the main interface to the finite element code via key-value pairs defining the parameters of the modeled system and the simulator. Each variable is assigned a default value and so running the model does not require any input; simply calling `data = fea_simulation` will start running the model with default values.

```
function data = fea_simulation(varargin)
     % Finite element simulation with parameters and methods controlled by
     % VARARGIN.

     %%%%%%%%%%%%%%% Parse inputs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     % Any unmatched parameters should get forwarded to fea_system() and
     % fea_solver().
     params = inputParser;
     params.PartialMatching = false;
     params.KeepUnmatched = true;
     params.StructExpand = false;
     params.addParameter('use_exp_U0',true);
     params.addParameter('continue_from',[]);
     params.addOptional('sys',[]); % Overrides continue_from
     params.addOptional('solver',[]); % Overrides continue_from
     params.addParameter('KM',[]);
     params.addParameter('field_data',[]);

     argin = varargin;
     parse(params, argin{:});
     sys = []; solver = [];
     continuation = ~isempty(params.Results.continue_from);
     if continuation
          if ischar(params.Results.continue_from)
               continue_from = load(params.Results.continue_from);
          else
               continue_from = params.Results.continue_from;
          end
          sys = continue_from.sys;

          % add magnetic terms back in
          [field.A,field.dA,field.intA,field.d2A] = field_data();
          sys.field = field;
```

```
    sys.coupling = magnetic_field(sys,sys.magnetic_integration);

    solver = continue_from.solver;
end

if ~isempty(params.Results.sys), sys = params.Results.sys; end
if ~isempty(params.Results.solver), solver = params.Results.solver; end
if ~params.Results.use_exp_U0, argin = [argin {'U0',0}]; end

% Pull off any inputs that aren't key-value pairs
while ~isempty(argin) && ~ischar(argin{1}),
    argin(1) = [];
end

% Initialize the SYS and SOLVER structures if needed
if isempty(sys)
    sys = fea_system(argin{:});
elseif fea_needs_recompute(sys, argin)
    sys = fea_system(sys.ipt{:},argin{:});
end

if isempty(solver)
    solver = fea_solver(argin{:});
elseif fea_needs_recompute(solver, argin)
    solver = fea_solver(solver.ipt{:},argin{:});
end

% Convenience variables, etc
sys.Voltage = solver.Voltage;
nnodes = sys.nnodes;
xvec = fea_vec(sys.x0+sys.U0,sys.dofs);
ytip = xvec(sys.tipnode,2);
xtip = xvec(sys.tipnode,1);
```

```
ytip = 0;
ts = 0;
outputrows = 2;

% Time stamp setup
t_ = tic;

if solver.isarclength
     % INCREMENTATION PARAMETERS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     % corrector_alpha must be in [0, 1/2] and really it should just
     % either be 0 or 0.5. k should be chosen so that the scales of U
     % and V are similar numerically.
     if isnan(solver.arclength)
          solver.arclength = 3*max(sys.experiment.y);           % Radius of constraint
           ↪  surface
     end
     arclength = solver.arclength;

     x0vec = fea_vec(sys.x0,sys.dofs);
     if isnan(solver.corrector_k)
          solver.corrector_k = .1*(x0vec(:,1)-x0vec(:,2))'*(x0vec(:,1)-x0vec(:,2));
     end
end

% Load experimental or literature data for comparison
switch regexprep(func2str(solver.load_fn),'^.*/','') % Sanitize nested function names
     case {'uniform_load','saturated_uniform_load'}
          exactsolution = load('exact_uniform_load.mat');
          sys.experiment.x = exactsolution.xtip*sys.length;
          sys.experiment.V = exactsolution.n/20;
          sys.experiment.y = exactsolution.ytip*sys.length;
     case {'tip_load','tip_point_load'}
          exactsolution = load('exact_tip_load.mat');
```

```
          sys.experiment.x = exactsolution.xtip*sys.length;
          sys.experiment.V = exactsolution.n/10;
          sys.experiment.y = exactsolution.ytip*sys.length;
     otherwise
          sys.experiment = solver.expdata;
end

% Set up initial conditions (or take from continuation data)
if ~continuation % New data
     state = [];

     step = 1;
     niter = 0;
else
     state = continue_from.solution(end);
     step = length(continue_from.solution);
     chunk = state;
     niter = sum([continue_from.solution.iteration]);
end
stored_steps = 0;

fprintf('Initial conditions computed in %2.4f seconds.\n',toc(t_));

% Set up the data file (if we are saving)
if ~isempty(solver.filename)
     t_ = tic;
     nchunk = solver.nchunk;
     SYS = sys;
     sys = rmfield(sys,{'coupling','field'});

     if exist(solver.filename,'file') && ~continuation
          % File exists but we're not keeping it: delete the old one
          fprintf('Deleting previous data with the same filename.\n')
```

```
     delete(solver.filename)
     if exist('solution','var')
          save(solver.filename,'sys','solver','solution');
     else
          save(solver.filename,'sys','solver');
     end

     mf = matfile(solver.filename);

elseif ~exist(solver.filename,'file')
     % No existing file, just save a new file
     if exist('solution','var')
          save(solver.filename,'sys','solver','solution');
     else
          save(solver.filename,'sys','solver');
     end
     mf = matfile(solver.filename);

elseif continuation
     % Continuing from an existing file. Since overwriting variables
     % seems to result in ever-increasing file size, get the
     % information out of the old file, delete, and re-save using
     % SAVE.

     mf = matfile(solver.filename);
     if ~exist('solution','var')
          solution = mf.solution; %#ok
     end
     delete(solver.filename);
     save(solver.filename,'sys','solver','solution');
     clear solution
end
sys = SYS; clear SYS
```

```
     fprintf('Saved to %s in %2.4f seconds.\n',solver.filename,toc(t_));
else
     nchunk = Inf;
end

fprintf('Starting nonlinear simulation.\n')
fprintf(repmat('\n',1,outputrows-1));

% Variables not to include when saving data
rmnames = {'J','Jdet','Jinv','Hchat','Mhat','Khat',...
     'Rhat','Kload','KUU','KUP','KPP','S','K','KL','KNL',...
     'Kliter','Khiter','Qiter','dUiter','Rprederr','Fprederr','dRprediter','dFprediter',...
     'dRiter','dFiter','dKiter','Fiter','Riter','Uiter','KLiter','KNLiter',...
     'eiter','Siter','Ktipiter','modeerror','Kcond','UQ','UR','Hbar','Jni','Kiter',...
     'eigenvectors','eigenvalues','strain','Sh','damping'};
% Variables not to transfer between load increments
it_params = {'dtiter','relerr','abserr','ytiter','titer','Kliter','Khiter','Kiter',...
     'Qiter','dUiter','Rprederr','Fprederr','dRprediter','dFprediter',...
     'dRiter','dFiter','dKiter','Fiter','Riter','Uiter','KLiter','KNLiter',...
     'eiter','Siter','Ktipiter','modeerror','Kcond'};

% Run the incremental-iterative solution until reaching a termination
% condition
while termination_check
     % Compute solution at the next increment
     if solver.isdynamic
          [new_state, flag] = fea_solution_step(sys,solver,state);
          if flag < 0, fprintf('Terminating early\n'), break, end
          state = new_state; % Accept the solution
          niter = niter + new_state.iteration;

          [new_state, flag] = fea_solution_step(sys,solver,state);
          if flag < 0, fprintf('Terminating early\n'), break, end
```

```
     state = new_state; % Accept the solution
     niter = niter + new_state.iteration;
else
     [new_state, flag] = fea_solution_step(sys,solver,state);
     if flag < 0, fprintf('Terminating early\n'), break, end
     state = new_state; % Accept the solution
     niter = niter + new_state.iteration;
end

if step == 1 % Add every field in state to the chunk
     %fields_to_add = setdiff(fieldnames(state),fieldnames(chunk));
     %for f = 1:length(fields_to_add)
          %chunk.(fields_to_add{f}) = [];
     %end
     %chunk = orderfields(chunk); % important
     chunk = orderfields(state);
     chunk = rmfield(chunk,intersect(rmnames,fieldnames(chunk)));
     mf.Properties.Writable = true;
     mf.solution = chunk;
     mf.Properties.Writable = false;
end

xvec = fea_vec(sys.x0+state.U,nnodes);
Uvec = fea_vec(state.U,nnodes);
step = step+1;
xtip(step) = Uvec(sys.tipnode,1);
ts(step) = state.t; %#ok

% Store up solutions in a chunk to be saved later
fields_to_remove = union(...
     setdiff(fieldnames(state),fieldnames(chunk)),...
     intersect(rmnames,fieldnames(state)));
fields_to_add = setdiff(fieldnames(chunk),fieldnames(state));
```

```
    for f = 1:length(fields_to_add)
        state.(fields_to_add{f}) = [];
    end
    chunk(stored_steps+1) = orderfields(rmfield(state,...
        fields_to_remove));
    stored_steps = stored_steps+1;

    if stored_steps >= nchunk && ~isempty(solver.filename)
        mf.Properties.Writable = true;
        mf.solution(1,step-stored_steps+1:step) = chunk;
        mf.Properties.Writable = false;
        stored_steps = 0;
    end

    % Get the state ready to be used as an input for the next increment
    state = rmfield(state,intersect(it_params,fieldnames(state)));
    if isfield(state,'iterations'), state = rmfield(state,'iterations'); end
end

% Finally, write the remaining data (if we are saving)
if ~isempty(solver.filename)
    if stored_steps > 0
        fprintf('Final save...')
        mf.('Properties').Writable = true;
        mf.solution(1,step-stored_steps+1:step) = chunk;
        mf.('Properties').Writable = false;
        state = mf.solution;
        fprintf(' Done. \n')
    end

    if nargout > 0
        data = load(solver.filename);
    end
```

```
    else
        data = struct('sys',sys,'solver',solver,'solution',chunk);
    end

    function cont = termination_check
        cont = true;
        if isempty(state), return; end
        xvec = fea_vec(state.U+sys.x0,sys.dofs);
        if state.t > solver.tlim(2) || state.t < solver.tlim(1)
            cont = false;
        end
        if any(ytip > solver.ylim(2) | ytip < solver.ylim(1)),
            cont = false;
        end
        if sys.dofs == 3 && any(xvec(:,3) > solver.zlim(2) | xvec(:,3) < solver.zlim(1))
            cont = false;
        end

        if solver.isarclength && arclength < solver.min_arclength,
            cont = false;
        end
        if solver.isloadcontrol && abs(state.dt(end)) < solver.mindt,
            cont = false;
        end
        if niter > solver.maxiter
            cont = false;
        end
    end
end
```

A.2.2 Model system initialization

This function parses the key-value pairs corresponding to the system parameters and initializes the necessary matrices.

```
function sys = fea_system(varargin)
    t0 = tic;
    %%%%%%%%%%%%%%% Parse inputs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    parser = inputParser;
    parser.addParameter('E',20e6);
    parser.addParameter('eltsize',1e-3);
    parser.addParameter('chi',10);
    parser.addParameter('Ms',435e3);
    parser.addParameter('sigma',1);
    parser.addParameter('length', 15.5e-3);
    parser.addParameter('width', .12e-3);
    parser.addParameter('height', 2.5e-3);
    parser.addParameter('zeta',0.09);
    parser.addParameter('rho',2906.5);
    parser.addParameter('poiss',0.499);
    parser.addParameter('xm',[12e-3;5e-3;-1e-3]);
    parser.addParameter('eltstr','H27')
    parser.addParameter('U0',0); % Polynomial in x
    parser.addParameter('structure_integration',0);
    parser.addParameter('magnetic_integration',0);
    parser.addParameter('damping_alpha',[]);
    parser.addParameter('damping_beta',[]);
    parser.addParameter('coupling',[]);
    parser.addParameter('baserotation',0); % degrees
    parser.KeepUnmatched = true;
    parse(parser,varargin{:});
    sys = parser.Results;
    % Store the input required to remake this SYS structure. Must happen
    % before we assign any new fields to SYS.
```

```
sys.ipt = reshape([fieldnames(sys)';struct2cell(sys)'],...
     1,2*length(fieldnames(sys)));
clear p f

% If not provided, set damping values based on Rayleigh damping with
% damping of ZETA at (linear) resonance and half-resonance.
if isempty(sys.damping_alpha) && isempty(sys.damping_beta)
     fn = fea_cantilever_resonance(sys);
     sys.damping_alpha = 2*sys.zeta*fn/3;
     sys.damping_beta = 4/3*sys.zeta/fn;

     sys.damping_beta = 4/3*sys.zeta/fn/3; %NOTE
end

elt = fea_element(sys.eltstr,sys.structure_integration);

% Generate mesh
[sys.conn,x0] = fea_mesh([sys.length,sys.width,sys.height],elt,sys.eltsize);
nnodes = max(sys.conn(:));
sys.nnodes = nnodes;
sys.nel = size(sys.conn,1);
nel = sys.nel;

% Set zero coordinate
x0vec = fea_vec(x0,nnodes); dofs = size(x0vec,2);
[~,x_zero] = min(abs(x0vec(:,1)));
x_zero = x0vec(x_zero(1),1);
x0vec(:,1) = x0vec(:,1) - x_zero(1);
x0vec(:,2) = x0vec(:,2) - sys.width/2;
x0 = fea_unvec(x0vec);
sys.x0 = x0;

x0 = x0vec(:,1); y0 = x0vec(:,2);
```

```
elts.a = max(x0(sys.conn),[],2)-min(x0(sys.conn),[],2);
elts.b = max(y0(sys.conn),[],2)-min(y0(sys.conn),[],2);
elts.a = round(1e10*elts.a)/1e10; elts.b = round(1e10*elts.b)/1e10;
unifa = unique(elts.a); unifb = unique(elts.b);
assert(numel(unifa)==1 && numel(unifb)==1,'Non-uniform mesh')

if dofs==3
     x0vec(:,3) = x0vec(:,3) - sys.height/2;
     sys.x0 = fea_unvec(x0vec);
     z0 = x0vec(:,3);
     elts.c = max(z0(sys.conn),[],2)-min(z0(sys.conn),[],2);
     elts.c = round(1e10*elts.c)/1e10;
     unifc = unique(elts.c);
     assert(numel(unifc)==1,'Non-uniform mesh')
end

% Identify the "tip node": x=L, near the center of the tip.
tipdistance = [x0vec zeros(nnodes,3-dofs)]-...
     repmat([sys.length sys.width/2 sys.height/2],nnodes,1);
tipdistance = realsqrt(sum(tipdistance.^2,2));
[~,tipnode] = min(tipdistance);
sys.tipnode = tipnode;

clear tipdistance unifa unifb unifc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% ELEMENTAL VARIABLES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elts.rho = sys.rho*ones(nel,1);         % Element density (kg/m^3)
elts.E = sys.E*ones(nel,1);             % Element modulus (Pa)
elts.chi = sys.chi*ones(nel,1);         % Element susceptibility
elts.v = sys.poiss*ones(nel,1);         % Element Poisson ratio
sys.elt = elt;                          % Struct with useful element props
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% NODAL VARIABLES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ngauss = length(elt.r)^dofs*nel;
sys.ngauss = ngauss;

sys.chi = sys.chi*ones(nnodes,1);

% Get a list of all "exterior faces" of the system. A "face" is an edge in
% 2D or a surface in 3D. The data structure for the faces is a matrix in
% which each row is a face and the columns give the nodes that belong to
% that face.

faces = [... % First assume 2d and then check later
     sys.conn(:,elt.node_coords(:,1)== 1);...
     sys.conn(:,elt.node_coords(:,1)==-1);...
     sys.conn(:,elt.node_coords(:,2)== 1);...
     sys.conn(:,elt.node_coords(:,2)==-1);...
     ];
rface = [... % r-coordinates of face nodes
     repmat(elt.node_coords(elt.node_coords(:,1) == 1, 1)',nel,1);...
     repmat(elt.node_coords(elt.node_coords(:,1) ==-1, 1)',nel,1);...
     repmat(elt.node_coords(elt.node_coords(:,2) == 1, 1)',nel,1);...
     repmat(elt.node_coords(elt.node_coords(:,2) ==-1, 1)',nel,1);...
     ];
sface = [... % s-coordinates of face nodes
     repmat(elt.node_coords(elt.node_coords(:,1) == 1, 2)',nel,1);...
     repmat(elt.node_coords(elt.node_coords(:,1) ==-1, 2)',nel,1);...
     repmat(elt.node_coords(elt.node_coords(:,2) == 1, 2)',nel,1);...
     repmat(elt.node_coords(elt.node_coords(:,2) ==-1, 2)',nel,1);...
     ];
```

```
% Detect dofs
sys.dofs = dofs;
tface = zeros(size(rface));
if dofs==3
     faces = [faces; ...
          sys.conn(:,elt.node_coords(:,3)== 1);...
          sys.conn(:,elt.node_coords(:,3)==-1);...
     ];
     rface = [...
          rface;
          repmat(elt.node_coords(elt.node_coords(:,3) == 1, 1)',nel,1);...
          repmat(elt.node_coords(elt.node_coords(:,3) ==-1, 1)',nel,1);...
          ];
     sface = [...
          sface;
          repmat(elt.node_coords(elt.node_coords(:,3) == 1, 2)',nel,1);...
          repmat(elt.node_coords(elt.node_coords(:,3) ==-1, 2)',nel,1);...
          ];
     tface = [... % t-coordinates of face nodes
          repmat(elt.node_coords(elt.node_coords(:,1) == 1, 3)',nel,1);...
          repmat(elt.node_coords(elt.node_coords(:,1) ==-1, 3)',nel,1);...
          repmat(elt.node_coords(elt.node_coords(:,2) == 1, 3)',nel,1);...
          repmat(elt.node_coords(elt.node_coords(:,2) ==-1, 3)',nel,1);...
          repmat(elt.node_coords(elt.node_coords(:,3) == 1, 3)',nel,1);...
          repmat(elt.node_coords(elt.node_coords(:,3) ==-1, 3)',nel,1);...
          ];
end
% Element numbers corresponding to the (r,s,t) coordinates in [rst]face
faceelts = repmat((1:nel)',2*dofs,1); % By construction

% An exterior face is a face that only appears once in the face matrix.
[~,uniqfaces,~] = unique(faces,'rows');
[exterior_faces,I] = setdiff(faces,faces(setdiff(1:size(faces,1),uniqfaces),:),'rows');
```

```
faceelts = faceelts(I);
rface = rface(I,:); sface = sface(I,:); tface = tface(I,:);

% Bound and free nodes and dofs
% The BC-nodes are the set intersection of the left and exterior faces
bcnodes = intersect(exterior_faces,sys.conn(:,elt.node_coords(:,1)==-1),'rows');

alldofs = fea_vec((1:dofs*nnodes)',nnodes);
bcdofs = fea_unvec(alldofs(bcnodes,:))';
freedofs = ~ismember(1:dofs*nnodes,bcdofs);
sys.freedofs = freedofs(:);
sys.freenodes = ~ismember(1:nnodes,bcnodes);
sys.freenodes = sys.freenodes(:);

% How many surface gauss points will we have? Number of faces times gauss
% points per face.
nfgauss = length(elt.r)^(dofs-1);
nsgauss = nfgauss*size(exterior_faces,1);
sys.nsgauss = nsgauss;

sys.elts = elts;
sys.voltagegain = sys.sigma;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% INITIAL CONDITIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Now apply initial conditions via U0 - assume undeflected CL is along x
baserotation = sys.baserotation;

% rotate about the top instead of the center (to match ICs)
ztop = max(x0vec(:,3));
x0vec(:,3) = x0vec(:,3)-ztop;
x0vec(:,2) = x0vec(:,2)*cosd(baserotation)-x0vec(:,3)*sind(baserotation);
```

```
x0vec(:,3) = x0vec(:,2)*sind(baserotation)+x0vec(:,3)*cosd(baserotation);
x0vec(:,3) = x0vec(:,3)+ztop;
clear ztop;

Up = polyder(sys.U0);
for i = 1:nnodes
     if x0vec(i,1) > 0
          tn = polyval(Up,x0vec(i,1)); %tangent slope at (x,y)
          unitnormal = 1/sqrt(1+tn^2)*[-tn 1];
          x0vec(i,1:2) = [x0vec(i,1) polyval(sys.U0,x0vec(i,1))]+x0vec(i,2)*unitnormal;
     end
end

x0 = fea_unvec(x0vec); sys.x0 = x0;
sys.U0 = zeros(dofs*nnodes,1);
sys.dU0 = zeros(dofs*nnodes,1);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%% Derivatives of interpolation matrices %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% To make implementation easier later, these functions should take all
% three coordinates and just ignore t if 2D.
r = sym('r','real');
s = sym('s','real');
t = sym('t','real');

Hv = kron(elt.H(r,s,t),eye(dofs));
H1 = matlabFunction(elt.H,'vars',[r,s,t]);

dH1dr = matlabFunction(diff(H1(r,s,t),r),'vars',[r,s,t]);
dH1ds = matlabFunction(diff(H1(r,s,t),s),'vars',[r,s,t]);
```

```
dH1dt = matlabFunction(diff(H1(r,s,t),t),'vars',[r,s,t]);
gradrHv = [diff(Hv,r);diff(Hv,s);diff(Hv,t)];
gradrHv = matlabFunction(gradrHv(1:dofs^2,:),'vars',[r,s,t]);
clear r s t


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%% Constant matrices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elts = sys.elts;

% Strain-displacement matrix precursors
switch dofs
     case 2
          % BNL
          BNLfn = @(r,s,t) [1 0 0 0;0 0 1 0;0 1 0 0;0 0 0 1]*gradrHv(r,s,t);

          % BL0
          BL0mat = [1 0 0 0;0 0 0 1;0 1 1 0];

          % Cijrs
          elasticity = @(i) ... % Plane Stress
               elts.E(i)/(1-elts.v(i)^2)*[...
               1           elts.v(i) 0;...
               elts.v(i) 1         0;...
               0         0         (1-elts.v(i))/2];
     case 3
          BNLfn = @(r,s,t) [...
               1 0 0 0 0 0 0 0 0;...
               0 0 0 1 0 0 0 0 0;...
               0 0 0 0 0 0 1 0 0;...
               0 1 0 0 0 0 0 0 0;...
               0 0 0 0 1 0 0 0 0;...
```

```
            0 0 0 0 0 0 0 1 0;...
            0 0 1 0 0 0 0 0 0;...
            0 0 0 0 0 1 0 0 0;...
            0 0 0 0 0 0 0 0 1]*...
            gradrHv(r,s,t);

        BL0mat = [...
            1 0 0 0 0 0 0 0 0;...
            0 0 0 0 1 0 0 0 0;...
            0 0 0 0 0 0 0 0 1;...
            0 1 0 1 0 0 0 0 0;...
            0 0 0 0 0 1 0 1 0;...
            0 0 1 0 0 0 1 0 0;...
            ];

        el1 = @(i) elts.v(i)/(1-elts.v(i));
        el2 = @(i) (1-2*elts.v(i))/2/(1-elts.v(i));
        elasticity = @(i) ... % 3D Stress
            elts.E(i)*(1-elts.v(i))/(1-2*elts.v(i))/(1+elts.v(i))*[...
            1      el1(i) el1(i) 0      0      0;...
            el1(i) 1      el1(i) 0      0      0;...
            el1(i) el1(i) 1      0      0      0;...
            0      0      0      el2(i) 0      0;...
            0          0          0      0      el2(i) 0;...
            0          0          0      0      0      el2(i)];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% Surface load matrices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
g = length(elt.r); % number of gauss points in each direction

% If every [rst] coordinate in a face is equal, it's an [rst] face.
```

```
isrface = all(rface==repmat(rface(:,1),1,size(rface,2)),2);
issface = all(sface==repmat(sface(:,1),1,size(sface,2)),2);
istface = all(tface==repmat(tface(:,1),1,size(tface,2)),2) & dofs==3;

% Generate the set of gauss points for all the faces. Each coordinate will
% be a nfaces-by-g^(dofs-1) matrix.
[i,j] = ind2sub(g*ones(1,dofs-1),1:g^(dofs-1));
r = repmat(elt.r(i)',size(exterior_faces,1),1); % (r,s) or (r,t)
s = r;
s(~issface & ~isrface,:) = repmat(elt.r(j)',nnz(~isrface & ~issface),1); % (r,s)
t = repmat(elt.r(j)',size(exterior_faces,1),1);
alpha_s = zeros(size(exterior_faces,1),g^(dofs-1),2);

% Put in the fixed face coordinates
r(isrface,:) = repmat(rface(isrface,1),1,g^(dofs-1));
s(issface,:) = repmat(sface(issface,1),1,g^(dofs-1));
t(istface,:) = repmat(tface(istface,1),1,g^(dofs-1));
r = r'; s = s'; t = t';

% And compute the quadrature weight at all those points:
alpha_s(~isrface,:,1) = repmat(elt.al(i)',nnz(~isrface),1); % (r,s) or (r,t)
alpha_s(~issface & ~isrface,:,2) = repmat(elt.al(j)',nnz(~isrface & ~issface),1); % (r,s)
alpha_s(~issface & isrface,:,1) = repmat(elt.al(i)',nnz(~issface & isrface),1);  % (s,t)
alpha_s(~istface,:,2) = repmat(elt.al(j)',nnz(~istface),1);
alpha_s = prod(alpha_s(:,:,1:dofs-1),3)';
alpha_s = sparse(1:dofs*nsgauss,1:dofs*nsgauss,... % Gauss integration weights
     kron(alpha_s,ones(dofs,1)));

% Easiest way to make Hvs is to make Hs:
[I,~] = ind2sub([nsgauss,elt.elnodes],1:nsgauss*elt.elnodes);
Hs =
 ↪   sparse(I,kron(sys.conn(faceelts,:),ones(nfgauss,1)),H1(r(:),s(:),t(:)),nsgauss,nnodes);
Hvs = kron(Hs,speye(dofs));
```

```
% Evaluate D on the surfaces to get Ds:
[I,~] = ind2sub(...
     [nsgauss*(dofs-1),elt.elnodes],...
     1:nsgauss*(dofs-1)*elt.elnodes);
switch dofs
     case 2
          Ds = [dH1dr(r(:),s(:),t(:))'; dH1ds(r(:),s(:),t(:))'];
     case 3
          Ds = [dH1dr(r(:),s(:),t(:))'; dH1ds(r(:),s(:),t(:))'; dH1dt(r(:),s(:),t(:))'];
end
Ds = reshape(Ds,elt.elnodes,dofs*nsgauss)';

% Remove rows corresponding to derivatives we don't need, i.e. rows of Ds
% that correspond to derivatives normal to the surface
iv = [~isrface ~issface ~istface];
Ds = Ds(logical(kron(iv(:,1:dofs)',ones(1,nfgauss))),:);
Ds = sparse(I,kron(kron(sys.conn(faceelts,:),ones(nfgauss,1)),ones(dofs-1,1)),...
     Ds,...
     (dofs-1)*nsgauss,nnodes);
sys.Ds = Ds;
sys.Hvs = Hvs;
sys.alpha_s = (alpha_s);

switch dofs
     case 2
          Jds = sys.Ds*x0vec;
          Jds = Jds.*(Jds>eps);
          sys.Jds =
           ↪  spdiags(kron(realsqrt(Jds(:,1).^2+Jds(:,2).^2),[1;1]),0,2*nsgauss,2*nsgauss);
     case 3
          % This produces a matrix of stacked 2x2 jacobians for the 2D
          % faces of the mesh. However, the matrix is three columns wide;
```

```
            % one column of each 3x2 block is all-zero, depending on which
            % way the face points in element (r,s,t) coordinates. The
            % jacobians are evaluated at each surface gauss point, meaning
            % there are 2nsgauss rows here.
            Jds = sys.Ds*x0vec;

            % The final Jds matrix should be (3nsgauss)x(3nsgauss). Split
            % the temp matrix by face type:
            isrface = logical(kron(isrface,ones(g^(dofs-1)*(dofs-1),1)));
            issface = logical(kron(issface,ones(g^(dofs-1)*(dofs-1),1)));
            istface = logical(kron(istface,ones(g^(dofs-1)*(dofs-1),1)));
            Jd_r = Jds(isrface,[2 3]); Jd_r = Jd_r.*(Jd_r > eps);
            Jd_s = Jds(issface,[1 3]); Jd_s = Jd_s.*(Jd_s > eps);
            Jd_t = Jds(istface,[1 2]); Jd_t = Jd_t.*(Jd_t > eps);

            % Now do the normal 2D jacobian determinant thing for each
            % partition and recombine
            Jds = zeros(nsgauss,1);
            Jds(isrface(1:2:end)) = Jd_r(1:2:end,1).*Jd_r(2:2:end,2)...
                 -Jd_r(2:2:end,1).*Jd_r(1:2:end,2);
            Jds(issface(1:2:end)) = Jd_s(1:2:end,1).*Jd_s(2:2:end,2)...
                 -Jd_s(2:2:end,1).*Jd_s(1:2:end,2);
            Jds(istface(1:2:end)) = Jd_t(1:2:end,1).*Jd_t(2:2:end,2)...
                 -Jd_t(2:2:end,1).*Jd_t(1:2:end,2);

            % Put it on the diagonal
            sys.Jds = spdiags(kron(Jds(:),[1;1;1]),0,3*nsgauss,3*nsgauss);
end
clear Ds Hvs alpha_s isrface issface istface exterior_faces faceelts
clear rface sface tface uniqfaces faces Hs iv Jds

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% Volume load matrices %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nstraincomps = (dofs+nchoosek(dofs,2)); % Number of strain components
sys.nstraincomps = nstraincomps;

[ind,el] = ind2sub([g^dofs,nel],(1:ngauss)');
[j,i,k] = ind2sub(g*ones(1,dofs),ind);
coordinds = [i j k];
r = elt.r(i); s = elt.r(j); t = elt.r(k); % Gauss point local coordinates

% ELNODES is a matrix where the row index corresponds to gauss point
% number and the row entries are the nodes in the element that contains
% that gauss point.
elnodes = sys.conn(el,:);

% ELDOFS has a row for each gauss point and the entries in the row are
% that gauss point's element's degrees of freedom.
eldofs = reshape(alldofs(elnodes',:)',dofs*elt.elnodes,[])';

% D: make dH1dr, dH1ds, dH1dt at (r,s,t); composite to stacked blocks.
switch dofs
     case 2
          D = [dH1dr(r,s,t)'; dH1ds(r,s,t)'];
     case 3
          D = [dH1dr(r,s,t)'; dH1ds(r,s,t)'; dH1dt(r,s,t)']; % Breaks in 2D
end

% D has a row for each dof at each gauss point (and the precursor has
% elt.elnodes columns)
[I,~] = ind2sub([dofs*ngauss,elt.elnodes],1:dofs*ngauss*elt.elnodes);

% The columns of D correspond to the node indices
D = sparse(I,kron(elnodes,ones(dofs,1)),...
     reshape(D,elt.elnodes,dofs*ngauss)',...
```

```
     dofs*ngauss,nnodes);

[sys.J0,Ji0,Jd0] = fea_jacobians(D,x0vec);
sys.Ji0 = Ji0;
sys.D = D; clear D


% Dn matrix: the gradients of the shape functions, evaluated at the nodes
% rather than the gauss points. Primarily useful in computing the magnetic
% field, where the nodal derivatives of nodal variables are needed.  Can
% exploit the implementation of SPARSE, in which entries at the same
% coordinates are summed; what we want is the average, so simply divide
% each row by the number of contributions.
%
% Get the local node coordinates for each node in each element. Shared
% nodes will have repeated entries with different local coordinates. The
% entries should be sorted by node number, then element number. The
% matrices we need are:
%      - local coordinates r, s, t
%      - node number (to interpret the above vectors)
%      - element number (to interpret the coordinates)
% This can be facilitated by the sys.conn matrix, which lists which nodes
% are in each element, ordered the same way as elt.node_coords.

% Start by generating a vector that has each node number in order as many
% times as it is included in elements.
if dofs == 3
     [nodes,node_order] = sort(sys.conn(:)); % gives total repetitions of each node
     [~,n1f] = unique(nodes,'first'); [~,n1l] = unique(nodes,'last');
     node_counts = n1l-n1f+1;
     node_dofs = fea_unvec(alldofs(nodes,:));

     % Now make a vector that, for each node listed above, gives the local
```

```
    % coordinate index.
    [elts, node_index] = ind2sub(size(sys.conn),node_order);

    rn = elt.node_coords(node_index,1);
    sn = elt.node_coords(node_index,2);
    switch dofs
        case 2
            Dn = [dH1dr(rn,sn,tn)'; dH1ds(rn,sn,tn)'];
        case 3
            Dn = [dH1dr(rn,sn,tn)'; dH1ds(rn,sn,tn)'; dH1dt(rn,sn,tn)']; % Breaks in 2D
    end

    % Dn has a row for each dof at each node. Each row has elt.elnodes
    % entries, so repeat each row index that many times.
    Dn = spdiags(1./kron(node_counts,ones(dofs,1)),0,dofs*nnodes,dofs*nnodes)*sparse(...
        ... row index is the node dofs, repeated as we cycle
        ... through the columns of the Dn precursor
        repmat(node_dofs,elt.elnodes,1),...
        ... column index is the node number corresponding to the local node index
        ... for the current column of Dn and the element for the current row of Dn
        kron(sys.conn(elts,:),ones(dofs,1)),...
        reshape(Dn,elt.elnodes,numel(node_dofs))',...
        dofs*nnodes,nnodes);
    sys.Dn = Dn;

    clear Dn rn sn tn nodes node_order node_counts node_index n1f n1l In
end

% BL0,BNL,C: make elemental matrix; repmat to stacked blocks.
BNL = zeros(dofs^2*g^dofs,dofs*elt.elnodes);
C = zeros(nstraincomps*g^dofs);
for p = 1:g^dofs
    BNL(dofs^2*(p-1)+1:dofs^2*p,:) = BNLfn(r(p),s(p),t(p));
```

```
     C(nstraincomps*(p-1)+1:nstraincomps*p,...
          nstraincomps*(p-1)+1:nstraincomps*p) = elasticity(1);
end

% Change stacked-block matrices to system matrices via SPARSE.
% C:
blocksize = size(C);
[I,J] = ind2sub(blocksize.*[nel 1],(1:nel*prod(blocksize))');
C = sparse(I,J+blocksize(2)*floor((I-1)/blocksize(1)),...
     repmat(C,nel,1),blocksize(1)*nel,blocksize(2)*nel);
    sys.C = C;
clear C

% BNL: put the blocks into the matrix according to the dof table.
% Premultiply by the inverse jacobian repeated (dofs) times on the block
% diagonal. Is there an efficient way to do this?
[Jii,Jij] = find(Ji0);
Jii = Jii+floor((Jii-1)/dofs)*dofs*(dofs-1);
Jij = Jij+floor((Jij-1)/dofs)*dofs*(dofs-1);
Jirep = sparse(...
     kron((0:dofs-1)',ones(length(Jii),1))*dofs+repmat(Jii,[dofs 1]),...
     kron((0:dofs-1)',ones(length(Jij),1))*dofs+repmat(Jij,[dofs 1]),...
     repmat(nonzeros(Ji0),[dofs,1]),dofs*dofs*ngauss,dofs^2*ngauss);

blocksize = size(BNL);
[I,~] = ind2sub(blocksize.*[nel 1],(1:nel*prod(blocksize))');
sys.BNL = Jirep*sparse(I,kron(eldofs,ones(dofs^2,1)),...
     repmat(BNL,nel,1),blocksize(1)*nel,dofs*nnodes);
clear BNL Jirep Jii Jij

% Construct BL0 from BNL
BL0 = kron(speye(nel*g^dofs),sparse(BL0mat))*sys.BNL;
sys.BL0 = BL0; clear BL0
```

```
alpha = sparse(1:dofs*ngauss,1:dofs*ngauss,... % Gauss integration weights
     kron(prod(elt.al(coordinds(:,1:dofs)),2),ones(dofs,1)));
sys.alpha = alpha; clear alpha

% H: can make using H1(r,s,t). (As stacked blocks).
[I,~] = ind2sub([ngauss,elt.elnodes],1:ngauss*elt.elnodes);
H = sparse(I,elnodes,H1(r,s,t),ngauss,nnodes);

[I,J] = ind2sub([ngauss,4],1:ngauss*4);
J = J+4*floor((I-1)/g^dofs);
sys.G = sparse(I,J,[ones(ngauss,1) r(:) s(:) t(:)],ngauss,4*nel);
Hv = kron(H,eye(dofs));
sys.H = H; sys.Hv =Hv;
clear H Hv elnodes eldofs

sys.Jd0 = Jd0;
clear Jd0;

% Create DH from Jinv*D:
Ji0D = Ji0*sys.D;
[idxDHi,idxDHj] = ind2sub([dofs*ngauss,nnodes],find(Ji0D));

% Compute the mass matrix:
sys.Mass = sys.rho*sys.Hv'*sys.alpha*sys.Jd0*sys.Hv;

switch dofs
     case 2
          idxDHj = [idxDHj+nnodes*~mod(idxDHi,2) idxDHj+nnodes*mod(idxDHi,2)];
          idxDHi = [idxDHi+floor((idxDHi-1)/2) 3*ceil(idxDHi/2)];
     case 3
          idxDHj = [...
               idxDHj+nnodes*mod(idxDHi-1,3) ...
```

```
                idxDHj+nnodes*(~mod(idxDHi-1,3)+~mod(idxDHi,3)) ...
                idxDHj+nnodes*2*~~mod(idxDHi,3)];
            idxDHi = [...
                idxDHi+3*floor((idxDHi-1)/3) ...
                idxDHi+3*floor((idxDHi-1)/3)+(3*~mod(idxDHi-1,3)+2*~~mod(idxDHi-1,3)) ...
                idxDHi+3*floor((idxDHi-1)/3)+(5*~mod(idxDHi-1,3)+3*~~mod(idxDHi-1,3)) ...
                ];
end
DH = sparse(idxDHi,idxDHj,nonzeros(repmat(Ji0D,1,dofs)),nstraincomps*ngauss,dofs*nnodes);
 ↪  sys.DH = DH;
clear DH

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%% Index vectors %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If a matrix is block-diagonal with n 2x2 blocks, these get all the (i,j)
% elements of the blocks for a specific (i,j) pair

% put stacked 3x4 blocks into a block diagonal instead
% Size of L is [nstraincomps*ngauss,dofs^2]
[idxLi, idxLj] = ind2sub([nstraincomps*ngauss,dofs^2],1:nstraincomps*ngauss*dofs^2);
sys.indices.idxLj = idxLj+dofs^2*floor((idxLi-1)/nstraincomps);
sys.indices.idxLi = idxLi;
clear idxLi idxLj

% Get the strain components out of the matrix of block diagonals.
sstep = dofs^2*ngauss+dofs; send = dofs^2*ngauss^2;
if dofs == 2
    sstart = [1 2+dofs*ngauss 2];
elseif dofs == 3
    sstart = [1 2+dofs*ngauss 3+2*dofs*ngauss 2 3+dofs*ngauss 1+2*dofs*ngauss];
end
```

```
idxstrain = [];
for i = 1:length(sstart)
     idxstrain = [idxstrain;sstart(i):sstep:send]; %#ok
end

clear sstart sstep send
sys.indices.idxstrain = idxstrain(:);

% Index vectors for creating the load stiffness matrix in the magnetic
% load case
[sys.indices.MI,sys.indices.MJ] = find(logical(kron(speye(ngauss),ones(dofs))));

% Put the entries of Sh into S in the appropriate spots
% Sh in 2D is [S11 S22 S12]'
% Sh in 3D is [S11 S22 S33 S12 S23 S31]'
%
% First reshape Sh to get all like entries in columns
Shidx = reshape(1:ngauss*nstraincomps,nstraincomps,ngauss)';
% Diagonal entries of Shidx are repeated dofs times, off-diagonal elements
% are repeated 2dofs times.
Shidxdiag = Shidx(:,1:dofs); Shidxoffdiag = Shidx(:,dofs+1:end);
Shidxdiag = kron(Shidxdiag,ones(dofs,1)); Shidxoffdiag = kron(Shidxoffdiag,ones(dofs,1));
% Now stack the indices. This is the right-hand-side index vector:
%      In 2D, Sh(Shidx) = [S11 S22 S12 S21] kron [1 1]
%      In 3D, -> [11 22 33 12 23 31 21 32 13] kron [1 1 1]
sys.indices.Shidx = [Shidxdiag(:);Shidxoffdiag(:);Shidxoffdiag(:)];
clear Shidxdiag Shidxoffdiag Shidx

% Construct LHS index vector. Start by constructing an index matrix. Size
% of S is dofs^2*ngauss. We only care about the elements on the
% dofs-by-dofs block diagonal.
Si = find(logical(kron(speye(dofs*ngauss),ones(dofs)))); % -> column vector
% Now reshape to gather like entries (as above). When we do this we're
```

```
% going to get [S11 S12 S21 S22]' in 2D, or [11 12 13 21 22 23 31 32 33]
% in 3D. These need to be reordered to match the indexing operation into
% Sh.
Si = reshape(Si,dofs^2,ngauss*dofs)';
switch dofs
     case 2
          cols = [1 4 2 3];
     case 3
          cols = [1 5 9 2 6 7 4 8 3];
end
Si = Si(:,cols);
[sys.indices.SI,sys.indices.SJ] = ind2sub([ngauss*dofs^2,ngauss*dofs^2],Si(:));
clear Si

% alpha, Jd scaled to match strain components
aJd_1 = diag(sys.alpha*sys.Jd0); aJd_1 = aJd_1(1:sys.dofs:end);
sys.aJd.g = spdiags(aJd_1,0,sys.ngauss,sys.ngauss); % 1 per gauss point
sys.aJd.S = spdiags(kron(aJd_1(:),ones(sys.dofs^2,1)),... % dofs^2 per gauss pt
     0,sys.dofs^2*sys.ngauss,sys.dofs^2*sys.ngauss);
sys.aJd.st = spdiags(kron(aJd_1(:),ones(nstraincomps,1)),... % nstraincomp per gauss pt
     0,nstraincomps*sys.ngauss,nstraincomps*sys.ngauss);

%  TODO: add a way to check if the simulation needs magnetics
if true % add magnetic information
     % field data
     % TODO: add input checking for type of field (uniform, electromagnet)
     [field.A,field.dA,field.intA,field.d2A] = field_data();
     sys.field = field;

     % Magnetic coupling function
     sys.coupling = magnetic_field(sys,sys.magnetic_integration);
end
```

```
    % Keep this at the end: size of matrices, elapsed time
    info = whos('sys'); unit = floor(log(info.bytes)/log(1024));
    units = {'B','kB','MB','GB'};

    t = toc(t0);
    fprintf('Constructed %3.3g%s of system matrices in %g seconds.\n',...
        info.bytes/1024^unit,units{unit+1},t)
end
```

A.2.3 Solver initialization

This function parses the key-value pairs corresponding to the solver parameters and initializes the necessary matrices.

```
function solver = fea_solver(varargin)
    % This function parses inputs and constructs a solver structure that
    % contains all of the parameters that will be needed by the various FEA
    % functions to do option checking, etc. Continuations will be handled at a
    % higher level, this function just parses the inputs as received.

    solver_params = inputParser;
    solver_params.PartialMatching = false;
    solver_params.KeepUnmatched = true;
    solver_params.StructExpand = false;

    % Most of these just go directly into the solver structure to be read out
    % by other functions, a few need special handling after the parsing.
    solver_params.addParameter('load_fn',@fea_magnetic_load,@(x) isa(x,'function_handle'))
    solver_params.addParameter('magnetic_method','kelvin-naive')
    solver_params.addParameter('applied_field','electromagnet')
    solver_params.addParameter('expdata',0);
    solver_params.addParameter('Voltage',@(t) t)
    solver_params.addParameter('include_gravity',true);
    solver_params.addParameter('use_exp_U0',true);
```

```
solver_params.addParameter('dt',0.01)
solver_params.addParameter('RelTol',1e-6);
solver_params.addParameter('AbsTol',1e-12);
solver_params.addParameter('filename',[])
solver_params.addParameter('minstep',3);
solver_params.addParameter('maxstep',20);
solver_params.addParameter('mindt',1e-4); % Terminate if dt < mindt
solver_params.addParameter('maxdt',0.01); % Cap dt at maxdt
solver_params.addParameter('maxiter',Inf); % Terminate after N iterations
solver_params.addParameter('incrementmethod','arclength');
solver_params.addParameter('errormethod','displacement');
solver_params.addParameter('arclength',NaN);
solver_params.addParameter('initial_direction',1); % 1 or -1
solver_params.addParameter('mode','static');
solver_params.addParameter('nchunk',25);
solver_params.addParameter('min_arclength',1e-6);
solver_params.addParameter('incompressibility',false);
solver_params.addParameter('corrector_k',NaN);
solver_params.addParameter('linesearch',false);
solver_params.addParameter('linetol',0.5);
solver_params.addParameter('dsmin',0.1);
solver_params.addParameter('corrector_alpha',0.5);
solver_params.addParameter('modes',0);
solver_params.addParameter('bias_force',0);
solver_params.addParameter('bias_voltage',0); % included in static case only
solver_params.addParameter('disable_z',false,@islogical)
solver_params.addParameter('ylim',[-Inf Inf]);
solver_params.addParameter('zlim',[-Inf Inf]);
solver_params.addParameter('tlim',[-Inf Inf]);

parse(solver_params, varargin{:});
solver = solver_params.Results;
% Store the input required to remake this SOLVER structure. Must happen
```

```
    % before we assign any new fields to SOLVER.
    solver.ipt = reshape([fieldnames(solver)';struct2cell(solver)'],...
        1,2*length(fieldnames(solver)));

    solver.ismagnetic = strcmpi(func2str(solver.load_fn),'fea_magnetic_load');

    switch solver.mode
        case 'dynamic'
            solver.isdynamic = true;
            solver.isstatic = false;
        case 'static'
            solver.isdynamic = false;
            solver.isstatic = true;
            % Ensure the proper voltage function in the static case
            solver.Voltage = @(t) t + solver.bias_voltage;
    end
    solver.ismodal = solver.modes > 0;
    switch solver.incrementmethod
        case 'arclength'
            solver.isarclength = true;
            solver.isloadcontrol = false;
            assert(~solver.isdynamic,...
                'Arc length solution not valid in dynamic case.')


        case 'loadcontrol'
            solver.isarclength = false;
            solver.isloadcontrol = true;
    end

end
```

A.2.4 Element definition

This function constructs a representation of a finite element.

```
function elt = fea_element(eltstr,int_order)
    % Properties: H, elnodes, node_coords (natural coordinates), bcnodes,
    % computation of x0, padding functions, gaussquad constants, dof function
    % connectivity for plotting

    default('int_order',0);

    switch eltstr
        case 'Q8'
            elt.elnodes = 8;
            elt.node_coords = [-1 1; -1 0; -1 -1;0 1;0 -1;1 1;1 0;1 -1];
            elt.H = shape_functions(elt.node_coords);
            [elt.r,elt.al] = quadrature_rule(3+int_order);
        case 'H20'
            node_coords = coord_table(linspace(-1,1,3)',3);
            z = sum(node_coords==0,2);
            elt.elnodes = 20;
            elt.node_coords = node_coords(z<2,:);
            assert(length(elt.node_coords)==20);

            elt.H = shape_functions(elt.node_coords);
            [elt.r,elt.al] = quadrature_rule(3+int_order);
        case 'H20R'
            node_coords = coord_table(linspace(-1,1,3)',3);
            z = sum(node_coords==0,2);
            elt.elnodes = 20;
            elt.node_coords = node_coords(z<2,:);
            assert(length(elt.node_coords)==20);

            elt.H = shape_functions(elt.node_coords);
```

```
            [elt.r,elt.al] = quadrature_rule(2);
        otherwise
            [nn,dofs,q] = parse_elt_str(eltstr);
            coords = linspace(-1,1,nn)';
            elt.node_coords = coord_table(coords,dofs);
            elt.H = shape_functions(elt.node_coords);
            elt.elnodes = nn^dofs;
            [elt.r,elt.al] = quadrature_rule(q+int_order);
    end

end

function [nn,dofs,q] = parse_elt_str(eltstr)
    shape = eltstr(1);
    switch shape
        case 'Q'
            dofs = 2;
        case 'H'
            dofs = 3;
        otherwise
            error('unrecognized element string')
    end

    nnodes = str2double(eltstr(2:end));
    nn = nnodes^(1/dofs);
    assert(abs(nn-round(nn))<1e-12);
    q = round(nn);
    nn = round(nn);
end

function nc = coord_table(coords,dofs)
    nn = length(coords);
    nc = zeros(nn^dofs,dofs);
```

```
    for i = 1:dofs
        nc(:,i) = repmat(kron(coords,ones(nn^(dofs-i),1)),nn^(i-1),1);
    end

end
```

A.2.5 Shape functions

This function computes the finite element shape functions

```
function H = shape_functions(node_coords)
    % General method for generating iso-P shape functions:
    %     1. At node i, the coordinates are (ri,si) and hj(ri,si) = ∂ij.
    %     2. Iterate through the nodes and multiply all shape functions that
    %     are not this shape function AND are not already zero at (ri,si) by
    %     (r-ri)(s-si).
    %     3. Evaluate each shape function at its own node and normalize to 1.
    [N,dofs] = size(node_coords);

    % For the "Serendipity" elements, i.e. elements that are not a m-by-m
    % grid, adjust the shape function generation procedure:
    %     a. Shape functions can't include a factor of r (or s or t).
    %     b. This doesn't affect the edge nodes' shape functions because they
    %     can be completely expressed as products of (ß ± 1) for ß in [r,s,t].
    %     c. For the corner nodes, need a factor of (1-ri*r-si*s-ti*t) to
    %     complete the shape function, so we can initialize the shape functions
    %     with that factor.

    if abs(round(N^(1/dofs))-N^(1/dofs)) < 1e-12
        element_type = 'full';
    elseif N == 2^dofs + 2*factorial(dofs)
        element_type = 'serendipity';
    else
        error('Unsupported element type.');
```

```
end

method = 'sym';
switch method
     case 'sym'
          %%%%%%%% symbolic implementation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          coords = [sym('r','real');sym('s','real');sym('t','real')];
          Hsym = sym(ones(1,N));
          for i = 1:N
               ci = node_coords(i,:)';
               if strcmpi(element_type,'serendipity') && sum(ci==0)==0
                    % Multiply this corner node by (1-r*ri-s*si-t*ti)
                    Hsym(i) = Hsym(i)*(1-sum(ci.*coords(1:dofs)));
               end

               for d = 1:dofs
                    c = coords(d);
                    fixes = find(logical(subs(Hsym,c,ci(d))~=0) & ((1:N)~=i));
                    for j = fixes
                         if ci(d) ~= node_coords(j,d) ...
                                   && ~(strcmpi(element_type,'serendipity')...
                                   && ci(d)==0)
                              Hsym(j) = Hsym(j)*(c-ci(d));
                         end
                    end
               end
          end
          H = matlabFunction(Hsym,'vars',coords);
          nc = mat2cell(node_coords,N,ones(dofs,1));
          ident = H(nc{:});
          H = matlabFunction(Hsym/ident,'vars',coords);
     case 'poly'
          % since the shape functions are (simple) polynomials in two
```

```
            % variables, store as two matrices and multiply the
            % corresponding polynomials at the end.
            r = sym('r','real'); s = sym('s','real');
            order = N^(1/dofs);
            Hrpoly = mat2cell([zeros(N,order-1) ones(N,1)],ones(N,1),order); % unit
             ↪  polynomials
            Hspoly = mat2cell([zeros(N,order-1) ones(N,1)],ones(N,1),order); % unit
             ↪  polynomials
            for i = 1:n
                 ri = node_coords(i,1); si = node_coords(i,2);
                 for j = [1:i-1 i+1:N]
                      Hjr = polyval(Hrpoly{j},ri);
                      if Hjr ~= 0 && ri ~= node_coords(j,1)
                           Hrpoly{j} = conv(Hrpoly{j},[zeros(1,order-2) 1 -ri]);
                      end

                      Hjs = polyval(Hspoly{j},si);
                      if Hjs ~= 0 && si ~= node_coords(j,2)
                           Hspoly{j} = conv(Hspoly{j},[1 -si]);
                      end
                 end
            end

            Hsym = sym(zeros(1,N));
            for i = 1:n
                 Hrpoly{i} = Hrpoly{i}/polyval(Hrpoly{i},node_coords(i,1));
                 Hspoly{i} = Hspoly{i}/polyval(Hspoly{i},node_coords(i,2));
                 Hsym(i) = poly2sym(Hrpoly{i},r)*poly2sym(Hspoly{i},s);
            end
            H = matlabFunction(hsym,'vars',[r,s]);
       end
  end
```

A.2.6 Jacobian matrices

This function computes the Jacobian of a finite element quantity.

```
function [J,Ji,Jd] = fea_jacobians(D,xvec)
     % This function takes in the gradient interpolation function D and the
     % vector of point coordinates XVEC (n-by-ndof) and computes the Jacobian,
     % its inverse, and its determinant at each point in XVEC.
     J = D*xvec; dofs = size(xvec,2);  nj = size(D,1)/dofs;

     % Get rid of numerical noise
     J = J.*(abs(J)>10*eps);

     [Ji,Jj] = ind2sub([dofs*nj,dofs],1:dofs^2*nj);
     J = sparse(Ji,Jj+dofs*floor((Ji-1)/dofs),J,dofs*nj,dofs*nj);

     switch dofs
          case 2
               idxi1 = 1:2:2*nj; idxi2 = 2:2:2*nj;
               idx11 = 1:4*nj+2:4*nj^2;
               idx21 = 2:4*nj+2:4*nj^2;
               idx12 = 2*nj+1:4*nj+2:4*nj^2;
               idx22 = 2*nj+2:4*nj+2:4*nj^2;

               J11 = J(idx11); J12 = J(idx12); J21 = J(idx21); J22 = J(idx22);

               Jd = sparse(1:2*nj,1:2*nj,kron(J11.*J22-J12.*J21,[1 1]),2*nj,2*nj,(2*nj));

               Ji = sparse([idxi1 idxi1 idxi2 idxi2],[idxi1 idxi2 idxi1 idxi2],...
                    [J22 -J12 -J21 J11],2*nj,2*nj,4*nj);
          case 3
               Jx = @(i,j) J((i+(j-1)*dofs*nj):dofs^2*nj+dofs:(dofs*nj)^2);
               i1 = 1:3:3*nj; i2 = 2:3:3*nj; i3 = 3:3:3*nj;
```

```
            % Analytical inverse and determinant
            Jd = sparse(1:3*nj,1:3*nj,kron(...
                 Jx(1,1).*(Jx(2,2).*Jx(3,3)-Jx(2,3).*Jx(3,2))+...
                 Jx(1,2).*-(Jx(2,1).*Jx(3,3)-Jx(2,3).*Jx(3,1))+...
                 Jx(1,3).*(Jx(2,1).*Jx(3,2)-Jx(2,2).*Jx(3,1)),...
                 [1 1 1]),3*nj,3*nj,(3*nj));
            Ji = sparse(...
                 [i1 i2 i3 i1 i2 i3 i1 i2 i3],...
                 [i1 i1 i1 i2 i2 i2 i3 i3 i3],...
                 [Jx(2,2).*Jx(3,3)-Jx(2,3).*Jx(3,2) ...
                     -(Jx(2,1).*Jx(3,3)-Jx(2,3).*Jx(3,1)) ...
                 Jx(2,1).*Jx(3,2)-Jx(2,2).*Jx(3,1) ...
                 -(Jx(1,2).*Jx(3,3)-Jx(1,3).*Jx(3,2)) ...
                 Jx(1,1).*Jx(3,3)-Jx(1,3).*Jx(3,1) ...
                 -(Jx(1,1).*Jx(3,2)-Jx(1,2).*Jx(3,1)) ...
                 Jx(1,2).*Jx(2,3)-Jx(1,3).*Jx(2,2) ...
                 -(Jx(1,1).*Jx(2,3)-Jx(1,3).*Jx(2,1)) ...
                 Jx(1,1).*Jx(2,2)-Jx(1,2).*Jx(2,1)],...
                 3*nj,3*nj,dofs^2*nj);
        otherwise
    end
    Ji = Ji/Jd;
end
```

A.2.7 Mesh construction

This function constructs a finite element mesh from elements.

```
function [conn,x0] = fea_mesh(domainsize,elt,eltsize)
    % Generate 2D quad mesh for a specified rectangular area and element type
    % with a prescribed element size. DOMAINSIZE = [xdim, ydim, (zdim)].
    % ELTSIZE: same format.

    tic;
```

```
dofs = min(length(domainsize),size(elt.node_coords,2));
% Sanitize inputs
if length(eltsize) > dofs
     eltsize = eltsize(1:dofs);
end

% Get the size of the canonical element (assuming uniform mesh)
meshdim = max(round(domainsize(1:dofs)./eltsize),1);
eltsize = domainsize(1:dofs)./meshdim;

meshdim = meshdim(1:dofs);

nel = prod(meshdim);
X = [];
conn = zeros(nel,elt.elnodes);

% Assume the element numbering starts with the top-left element
% and goes columnwise. Iterate through the elements and number the nodes
% and assign their coordinates. Put the corner at the origin and let the
% calling function adjust the coordinate system.
for k = 1:nel
     [eli,elj,elk] = ind2sub(meshdim([2 1 3:dofs]),k);
     sub = [elj eli elk]; sub = sub(1:dofs);
     for n = 1:elt.elnodes
          coordn = (sub-1/2).*eltsize+eltsize/2.*elt.node_coords(n,:);

          % If there's already a node here, use it
          if k > 1
               existing_node = find(all(abs(X-repmat(coordn,size(X,1),1))<10*eps,2));
               assert(numel(existing_node)<2,'Something weird happened in the mesher!');
          else
               existing_node = [];
          end
```

```
            if isempty(existing_node)
                newindex = max(conn(:))+1;
                conn(k,n) = newindex;
                X(newindex,:) = coordn; %#ok
            else
                conn(k,n) = existing_node;
            end

        end
    end
    x0 = reshape(X',numel(X),1);
    t = toc;
    fprintf('Created mesh with %g nodes (%g dofs) in %g seconds.\n',newindex,dofs*newindex,t)
end
```

A.2.8 Vectorize/Unvectorize

These convenience functions convert between two different representations of vector-valued finite element quantities: one in which each column corresponds to a vector component and each row corresponds to the location ("vectorized" form) and one in which the components are stacked, $X = [x_1\ y_1\ x_2\ y_2 ...]^T$ ("unvectorized" form).

```
function xvec = fea_vec(x,sz)
    assert(size(x,2)==1)

    if length(x)/sz > sz % Then the input is DOFS
        xvec = reshape(x,sz,[])';
    else % the input is number of rows
        xvec = reshape(x,[],sz)';
    end
end
```

```
function x = fea_unvec(xvec)
    x = xvec'; x = x(:);
end
```

A.2.9 Status printout

This function prints out the current status of the simulation.

```
function fea_status_line(sys,solver,state)
    % TODO: make this configurable with a setting in SOLVER, auto layout
    % the values, figure out OUTPUTROWS automatically, etc.

    %%%% STATUS OUTPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    outputrows = 2;
    uvec = fea_vec(state.U,sys.dofs);
    totalwidth = get(0,'CommandWindowSize'); totalwidth = totalwidth(1);
    system(['tput cuu ' num2str(outputrows-1)]);
    fprintf('\r')
    N = 0;
    N = N + printct('Step %d, Itr. %d: ',state.step, state.iteration);
    N = N + printct('yt = % 3.3gmm, ',1e3*uvec(sys.tipnode,2));
    N = N + printct('V = % 5.3gV, ', state.V);
    N = N + printct('RE = % 3.1e, AE = %3.1e, ',state.relerr(end), state.abserr(end));
    if solver.isarclength
        N = N + printct('arclength = %5.3g',state.arclength);
    elseif solver.isloadcontrol
        N = N + printct('step size = %5.3g',state.dt);
    end

    fprintf(repmat(' ',totalwidth-N,1));
    fprintf('\n')

    progressbar(solver.tlim(2),state.t);
end
```

```
function N = printct(str,varargin)
     N = length(sprintf(str,varargin{:}));
     fprintf(str,varargin{:});
end
```

A.2.10 State initialization

Initialize the state structure with default values.

```
function state = fea_state_init(sys, solver, state)
     % This function has three related jobs:
     %      1. When STATE is empty, create a new state structure with the minimum
     %      fields required for functions downstream to not crash
     %      2. When STATE is a converged solution, initialize the next state,
     %      filling in the PREV fields with the appropriate data, and
     %      incrementing the timestep in the load control case.
     %      3. When STATE is a non-converged solution, reset to the beginning of
     %      the load step and cut the time step.
     %
     %
     % Set up the structure for the output of the incremental solution
     % based on the previous solution. Also used to reset the state when
     % we go over the iteration limit

     if nargin == 2 || isempty(state)
          % Initialize the starting state based on SYS and SOLVER
          state = orderfields(struct(...
               'V',0,'U',sys.U0,...
               'dU', sys.dU0,'d2U',0*sys.U0,...
               'step',0,'iteration',0,...
               't',0,'chi',sys.chi,'p',zeros(4*sys.nel,1)));
          isconverged = true;
     else
```

```
     isconverged = ~isfield(state,'relerr') || fea_convergence_check(solver, state);
end

if ~isconverged
     % Cut step size
     if solver.isloadcontrol
          dt = state.dt/2;
     elseif solver.isarclength
          arclength = state.arclength/2;
     end

     % reset the state
     fn = fieldnames(state.prev);
     for i = 1:length(fn)
          state.(fn{i}) = state.prev.(fn{i});
     end
     state.step = state.step + 1;

else % was converged
     % Reset step size to default
     dt = solver.dt;
     arclength = solver.arclength;

     prev = orderfields(struct(...
          't',state.t,'U',state.U,...
          'dU',state.dU,'d2U',state.d2U,...
          'V',state.V, 'chi', state.chi,...
          'step', state.step,'p', state.p));
     if solver.isdynamic && ...  % handle TRBDF step requirements
               isfield(state,'prev')...
               && length(state.prev) == 1
          state.prev(2) = prev;
     else
```

```
            state.prev = prev;
            state.step = state.step + 1;
        end
    end

    state.iteration = 0;
    state = fieldcat(state,'relerr',NaN);
    state = fieldcat(state,'abserr',NaN);

    % Increment the load level, if using load control.
    if solver.isloadcontrol
        state.t = state.t + dt;
        state.V = solver.Voltage(state.t);
        state.dt = dt;
    else %arclength
        state.t = state.V;
        state.arclength = arclength;
    end

    state = fea_update_state(sys, solver, state);

end
```

A.2.11 Incremental solution

This function applies one increment (i.e. load step), which are the outer loop in the incremental-iterative solution scheme.

```
function [new_state, flag] = fea_solution_step(sys, solver, state)
    flag = -10; % not converged yet
    new_state = state;
    while flag < 0
        new_state = fea_state_init(sys,solver,new_state);
        if solver.isloadcontrol && abs(new_state.dt) < solver.mindt
```

```
            break
        elseif solver.isarclength && new_state.arclength < solver.min_arclength
            new_state.VQ = NaN;
            break
        end

        [new_state, flag] = fea_iterative_solution(sys, solver, new_state);
    end

end
```

A.2.12 Iterative solution

This function applies the iterations, i.e. the inner loop of the incremental-iterative solution scheme.

```
function [state, flag] = fea_iterative_solution(sys, solver, state)
    % Start from STATE, which is out of balance, and iterate until meeting the
    % error tolerance, going over the iteration limits, or receiving an error
    % from the incrementing function.

    flag = 0;
    fea_status_line(sys, solver, state);
    if ~isfield(state,'iterations'), state.iterations = []; end
    while state.iteration < solver.maxstep && flag == 0
        % Compute the increment in U, V
        [state, flag] = fea_compute_increment(sys, solver, state);

        % Apply the increment
        state.U = state.U+state.increment.U;
        state.V = state.V+state.increment.V;
        if solver.incompressibility % apply pressure increment
            state.p = state.p + state.increment.p;
        end
```

```
state.iteration = state.iteration + 1;

% update loads
state = fea_update_state(sys,solver,state);
state.iterations = fieldcat(state.iterations,'t',state.V);
state.iterations =
 ↪  fieldcat(state.iterations,'ytip',state.U(sys.dofs*sys.tipnode-1));

% Compute error at the new state
[absolute_error, relative_error] = fea_estimate_error(sys,solver,state);
state = fieldcat(state,'relerr',relative_error);
state = fieldcat(state,'abserr',absolute_error);
fea_status_line(sys, solver, state);

% Convergence check
if flag == 0, [converged, flag] = fea_convergence_check(solver,state); end

if converged
     % Before returning, do some final state modifications
     if solver.isdynamic
          % At the end, compute the velocity/acceleration
          [state.dU, state.d2U] = fea_time_derivatives(state);
     end

     if solver.isarclength
          % Compute the next direction VQ
          DU = state.U-state.prev(end).U;
          DV = state.V-state.prev(end).V;
          test = state.UQ'*DU(sys.freedofs)+solver.corrector_k^2*DV;
          if test >= 0
               state.VQ = 1;
          else
               state.VQ = -1;
```

```
                end
            end

            return
        end
    end

    if ~converged
        flag = -10;
    end
end
```

A.2.13 State update

Given a new pair $(V, \mathbf{U})$, this function computes the internal and external loads and the stiffness matrix.

```
function state = fea_update_state(sys,solver,state)
    % Assume: STATE has STATE.U, STATE.V which are up-to-date. In a dynamic
    % simulation, STATE also has STATE.PREV.U, STATE.PREV.dU, STATE.PREV.d2U
    % (needed for derivatives) and STATE.PREV has as many elements as
    % necessary for the desired integration scheme (i.e. 1 for trapezoid, 2
    % for BDF, etc.).
    %
    % All other information in STATE (if any) is presumed to be stale and
    % ignored.

    dofs = sys.dofs; nnodes = sys.nnodes;

    % update all loads, etc.
    x = sys.x0+state.U;
    xvec = fea_vec(x,dofs);

    % Update jacobians and loads
```

```
[state.J,state.Jinv,state.Jdet] = fea_jacobians(sys.D,xvec);

% Compute the magnetic field if needed
if solver.ismagnetic && abs(state.V) > 0
     state = fea_internal_magnetic_field(sys,solver,state);
end

% R
[R, Kload] = solver.load_fn(sys, solver, state);
if solver.include_gravity
     Rg = sys.Hv'*sys.alpha*sys.Jd0*(sys.Hv*sys.Mass*9.8*kron(ones(nnodes,1),[0;0;-1]));
     R = R+Rg;
end

state = fea_structural_forces(sys, solver, state);

state.R = R;
F = state.F;
K = state.K;

if solver.isdynamic
     if state.iteration == 0
          state.damping = sys.damping_alpha*sys.Mass + sys.damping_beta*state.K;
     end
     [MK,MR] = fea_dynamic_forces(sys, state);
else
     MK = sparse(size(K,1),size(K,2));
     MR = sparse(size(R,1),size(R,2));
end

% Composite stiffnesses and loads
Khat = K + MK - Kload;
Rhat = R - F - MR + solver.bias_force;
```

```
    % Boundary conditions
    state.Khat = Khat(sys.freedofs,sys.freedofs);
    state.Rhat = Rhat(sys.freedofs);
    state.Kload = Kload(sys.freedofs,sys.freedofs);
end
```

A.2.14 Iterative update

This function computes the iterative update for either the arclength method or the load-control method.

```
function [state, flag] = fea_compute_increment(sys,solver,state)
    % Compute the increments to the displacements and load parameter according
    % to the methods listed in SOLVER.

    % If a modal solution was requested, convert to modal force
    Rhat = state.Rhat;
    Khat = state.Khat;
    if solver.ismodal
        evec = state.eigenvectors(:,1:solver.modes);
        Rhat = evec'*Rhat;
        Khat = evec'*Khat*evec;
    end

    flag = 0; % Means all is OK
    if solver.isloadcontrol
        dV = 0;
        dU = Khat\Rhat;
    elseif solver.isarclength
        % OUTLINE (ARC LENGTH METHOD):
        %      - Compute the field (iterate if necessary)
        %      - Compute the external load and load stiffness
        %      - Compute the derivative of the load vector wrt
```

```
%      Vin
%      - Compute the internal load and tangent stiffness
%      - Predictor: Compute the linearized deflection
%      due to Vin and the linearized deflection due to
%      the total tangent stiffness.  The predictor
%      solution is the (U, V) pair where U is along the
%      line given by the tangent stiffness and U·U+kV^2
%      = Δs.
%      - Corrector: update the (total) tangent stiffness
%      and find the intersections of the equilibrium
%      line with the constraint surface. Choose the
%      point closest to the current point.
%      - Convergence test: out-of-balance loads relative
%      to total load vector, or any other convergence
%      measure.


if isfield(state,'VQ')
     VQ = state.VQ;
else
     VQ = solver.initial_direction;
end
% Derivative of load vector wrt Vin via finite difference
num_delta_V = 1e-5;
if state.V < num_delta_V || VQ == -1, num_delta_V = -num_delta_V; end
temp = state;
temp.V = state.V+num_delta_V;
if solver.ismagnetic
     temp = fea_internal_magnetic_field(sys,solver,temp);
end
Rp = solver.load_fn(sys,solver,temp);
Qu = (Rp-state.R)/num_delta_V;
Qu = Qu(sys.freedofs);
```

```
UR = (Khat\Rhat);
UQ = Khat\Qu;

state.UR = UR;
state.UQ = UQ;

k = solver.corrector_k;

% Constraint implementation
if state.iteration == 0 %predictor step
     dV = VQ*state.arclength/sqrt(UQ'*UQ+k^2);
     dU = dV*UQ;
     UX = state.U; UX(sys.freedofs) = UX(sys.freedofs)+dU*solver.corrector_alpha;
     VX = state.V+dV*solver.corrector_alpha;
     state.UX = UX; state.VX = VX;
else      %corrector step
     dUt = state.U-state.UX; dUt = dUt(sys.freedofs);
     dVt = state.V-state.VX;
     state.dUt = dUt; state.dVt = dVt;

     kc = k; % spherical corrector
     %kc = 0; % cylindrical corrector
     arcc = (1-solver.corrector_alpha)*state.arclength;

     % Setup for partial correction/line search
     a0 = (UQ'*UQ+kc^2);
     b0 = (2*UQ'*dUt+2*kc^2*dVt);
     b1 = (2*UQ'*UR);
     c0 = dUt'*dUt+kc^2*dVt^2-arcc^2;
     c1 = (2*UR'*dUt);
     c2 = (UR'*UR);
     ln.a0 = a0; ln.b0 = b0; ln.b1 = b1;
```

```
ln.c0 = c0; ln.c1 = c1; ln.c2 = c2;


% What is the maximum correction to stay within
% the hypersphere?
as = b1^2-4*a0*c2;
bs = 2*b0*b1-4*a0*c1;
cs = b0^2-4*a0*c0;
if (bs/2/as)^2-cs/as < 0
     dsmax = 0;
else
     dsmax = (-bs/2/as+realsqrt((bs/2/as)^2-cs/as));
end


if dsmax < solver.dsmin % no correction, cut step size
     ds = 0;
     flag = -3; % Solution is outside the hypersphere
else
     ds = min(1,(1-1e-4)*dsmax);
     if solver.linesearch
          % Check if we need a linesearch
          Udir = UR;
          ga = Rhat'*Udir;
          sa = 0;
          g0 = ga;

          % forces at ds=1
          line_out = state;
          sb = ds;
          [dVk,ddVds] = arclength_roots(ln,sb);
          line_out.U = state.U;
          dUk = zeros(size(state.U));
```

```
dUk(sys.freedofs) = sb*UR+dVk*UQ;
line_out.U = state.U+dUk;
line_out.V = state.V+dVk;
line_out = fea_update_state(sys,solver,line_out);
Udir = UR + ddVds*UQ;
gb = line_out.Rhat'*Udir;

sk = sb;

kline = 0;
while ga*gb < 0 &&...
          abs(gb) > solver.linetol*abs(g0) &&...
          abs(sb-sa) > (sb+sa)*solver.linetol/2 &&...
          kline < 10
     kline = kline+1;
     sk = (sb-sa)/(ga-gb)*ga+sa;
     [dVk,ddVds] = arclength_roots(ln,sk);
     dUk(sys.freedofs) = sk*UR+dVk*UQ;
     line_out.U = state.U+dUk;
     line_out.V = state.V+dVk;
     line_out = fea_update_state(sys,solver,line_out);
     Udir = UR + ddVds*UQ;
     gk = line_out.Rhat'*Udir;

     %update
     if gk*gb > 0
          gb = gk; sb = sk;
     else
          ga = gb; sa = sb;
          gb = gk; sb = sk;
     end
end
ds = sk;
```

```
                end
            end
            dV = arclength_roots(ln,ds);
            dU = ds*UR+dV*UQ;
        end
    end

    % If modal, map back to real displacements
    if solver.ismodal
        dU = state.eigenvectors(:,1:solver.modes)*dU;
    end

    % Expand to include the bound dofs
    increment.U = zeros(size(state.U));
    increment.U(sys.freedofs) = dU;
    increment.V = dV;

    state.increment = increment;

end

function [dV,ddVds] = arclength_roots(ln,ds)
    a = ln.a0;
    b = ln.b0 + ln.b1*ds;
    c = ln.c0 + ln.c1*ds + ln.c2*ds^2;
    if (b/2/a)-c/a < 0
        root = 0;
    else
        root = realsqrt((b/2/a)^2-c/a);
    end
    dV1 = -b/2/a+root;
    dV2 = -b/2/a-root;
    ddVds1 = -(ln.b1/2)/ln.a0 + (b*ln.b1/2-a*ln.c1+...
```

```
        2*ds*a*ln.c2)/(2*a*sqrt((b/2)^2-a*c));
    ddVds2 = -(ln.b1/2)/ln.a0 - (b*ln.b1/2-a*ln.c1+...
        2*ds*a*ln.c2)/(2*a*sqrt((b/2)^2-a*c));
    tt = ln.b0/2;
    if tt*dV1 > tt*dV2
        dV = dV1;
        ddVds = ddVds1;
    else
        dV = dV2;
        ddVds = ddVds2;
    end
end
```

A.2.15 Convergence testing

This function computes the solution error based on one of several predefined error metrics.

```
function [absolute_error, relative_error] =  fea_estimate_error(sys,solver,state)
    % Estimate the error in the solution in STATE as indicated by
    % SOLVER.ERRORMETHOD.

    residual = state.Rhat;
    if solver.ismodal && strcmpi(solver.errormethod,'load')
        residual = state.eigenvectors(:,1:solver.modes)'*residual;
    end
    reference_value = [];

    switch solver.errormethod
        case 'work'
            % Energy-based error method from Bathe p. 765
            dU = state.increment.U;
            absolute_error =...
                abs(dU(sys.freedofs)'*residual);
        case 'displacement'
```

```
            dU = state.increment.U;
            absolute_error = norm(dU);
            reference_value = norm(state.U+eps);
        case 'load'
            absolute_error = norm(residual);
        otherwise
            error('Unrecognized error-estimation method.')
    end

    if isempty(reference_value)
        % Default behavior: use the initial absolute error
        if ~isfield(state,'abserr') || all(isnan(state.abserr) | state.abserr == 0)
            reference_value = absolute_error;
        else
            reference_value = state.abserr(find(~isnan(state.abserr) & state.abserr ~=
             ↪  0,1,'first'));
        end
    end

    relative_error = absolute_error/reference_value;
end
```

This function tests whether the error estimated from the above function satisfies the selected error tolerances.

```
function [converged, flag] = fea_convergence_check(solver,state)
    converged = false;
    flag = 0;
    if (state.relerr(end) < solver.RelTol || state.abserr(end) < solver.AbsTol)
        if solver.isloadcontrol % accept the solution
            converged = true;
        elseif solver.isarclength && state.iteration > 1
            DU = state.U-state.prev(end).U;
            DV = state.V-state.prev(end).V;
```

```
            % Reject repeated solutions. If corrector_alpha is 0.5,
            % then a repeated solution is one in which the state is
            % very close to the previous state. The arclength
            % constraint is that ΔU^2 + ΔV^2 = L^2, where (with alpha
            % 0.5) the comparison is to a point along the predictor's
            % projection. So if the threshold here is too large, sharp
            % curves in the equilibrium path will be rejected. If the
            % threshold is too small (or AbsTol/RelTol are too large)
            % there's a risk of accepting repeated solutions.
            if (DU'*DU+solver.corrector_k^2*DV^2)/solver.arclength^2 < 1e-3
                % TODO: parameterize the tolerance above
                flag = -2; % repeated solution
            else % accept the solution
                converged = true;
            end
        end
    end


end
```

A.2.16 Magnetics

```
function state = fea_applied_field(sys, state)
    % Assume: SYS has fields SYS.FIELD.A, SYS.FIELD.dA, SYS.FIELD.intA,
    % SYS.FIELD.d2A, which are function handles that take arguments X, Y, Z
    % (relative to a reference position SYS.XM) and return the appropriate
    % field quantities. STATE has fields STATE.V, STATE.U (input voltage and
    % deflection).
    V = state.V;

    xvec = fea_vec(state.U+sys.x0,sys.dofs);
    X = xvec(:,1)-sys.xm(1);
    Y = xvec(:,2)-sys.xm(2);
```

```
    if sys.dofs == 2
        Z = -sys.xm(3)*ones(sys.nnodes,1);
    else
        Z = xvec(:,3)-sys.xm(3);
    end
    state.Hchat = fea_unvec(sys.sigma*V*sys.field.A(X,Y,Z));
    state.phi_cs = -sys.sigma*V*sys.field.intA(X,Y,Z);
end

function state = fea_internal_magnetic_field(sys, solver, state)
    x = sys.x0+state.U;
    V = state.V;

    % Inverse jacobian evaluated at the nodes
    [~,Jni,~] = fea_jacobians(sys.Dn,fea_vec(x,sys.dofs));
    state.Jni = Jni;

    state = fea_applied_field(sys, state);
    chi = state.chi;

    % TODO: decide how often to recompute KM
    KM = sys.coupling(x);

    %%%%% EXTERNAL LOADS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    fielderr = 1; fieldTol = 1e-3;
    w = 1; %relaxation parameter
    while fielderr > fieldTol
        % Solve for potential at nodes
        psihat = (eye(sys.nnodes)-spdiags(chi,0,sys.nnodes,sys.nnodes)*KM)\state.phi_cs;

        Hbar = -state.Jinv*sys.D*psihat;
        Hhat = sys.Hv\Hbar;

        Hnorm = realsqrt(sum(fea_vec(Hhat,sys.dofs).^2,2));
```

```
        Hdir = fea_vec(Hhat,sys.dofs)./repmat(Hnorm,1,sys.dofs);

        if V < eps
            Mhat = zeros(size(Hhat));
            dchi = zeros(size(Hnorm));
        else
            Mhat = fea_unvec(diag(min(sys.Ms,chi.*Hnorm))*Hdir);
            Mnorm = realsqrt(sum(fea_vec(Mhat,sys.dofs).^2,2));
            dchi = Mnorm./Hnorm-chi;
        end

        fielderr = norm(dchi)/norm(chi);
        if fielderr > fieldTol % recompute KMinv with new chi
            recomps = recomps+1
            chi = chi+w*dchi;

            % Reenable later
            %KMinv = inv(eye(nnodes)-spdiags(chi,0,nnodes,nnodes)*KM);
        end
    end
    state.Hhat = Hhat; state.Mhat = Mhat;
    state.psihat = psihat; state.chi = chi;
    state.Hbar = Hbar;
end

function [fb, fb_U] = fea_magnetic_body_force(sys,solver,state)
    Mhat = state.Mhat;
    dofs = sys.dofs; ngauss = sys.ngauss;
    mu0 = 4e-7*pi;

    switch solver.magnetic_method
        case 'kelvin-naive'
            % Kelvin force using the field from the electromagnet as the
            % external H-field.
```

```
            Mbar = fea_vec(sys.Hv*Mhat,ngauss);
            xvec = fea_vec(sys.x0+state.U,sys.nnodes);
            X = sys.H*xvec(:,1)-sys.xm(1);
            Y = sys.H*xvec(:,2)-sys.xm(2);
            Z = sys.H*xvec(:,3)-sys.xm(3);

            % ∇H is a Nx3x3 array
            % First dimension: gauss point
            % Second dimension: component of H
            % Third dimension: derivative direction
            %
            % ∇H(i,j,k) = (∂_k H_j)|x_i
            gradHcbar = sys.sigma*state.V*sys.field.dA(X,Y,Z);

            % (M·∇)H = [(Σk M_k ∂_k)H_j]|x_i is a Nx3 array
            % First dimension (i): gauss point
            % Second dimension (j): component of H
            MdotgradHc = sum(permute(repmat(Mbar,[1 1 3]),[1 3 2]).*gradHcbar,3);

            fb = mu0*fea_unvec(MdotgradHc);

            d2Hbar = sys.sigma*state.V*sys.field.d2A(X,Y,Z);
            Mdotd2Hbar = sum(permute(repmat(Mbar,[1 1 3 3]),[1 3 4 2]).*d2Hbar,4);

            fb_U = sparse(sys.indices.MI,sys.indices.MJ,...
                reshape(permute(mu0*(Mdotd2Hbar),[3 2 1]),dofs^2*ngauss,1),...
                dofs*ngauss,dofs*ngauss);
        otherwise
            error('Unknown magnetic force method.')
    end
end
```

A.2.17 Structural forces

These functions compute the stress, strain, and corresponding structural finite element forces and stiffness.

```
function state = fea_stress_strain(sys, state)
    J = state.J;

    % strain
    XT = sys.Ji0*J; % transposed deformation gradient
    e = 1/2*(XT*XT'-speye(sys.dofs*sys.ngauss));
    state.strain = e(sys.indices.idxstrain);

    % Cut out numerical noise on the unity cancellations
    state.strain = state.strain.*(abs(state.strain) > eps);

    % Double the shear strain to convert to engineering strains.
    % TODO: make this more elegant
    if sys.dofs == 2
        state.strain(3:3:end) = 2*state.strain(3:3:end);
    elseif sys.dofs==3
        state.strain(4:6:end) = 2*state.strain(4:6:end);
        state.strain(5:6:end) = 2*state.strain(5:6:end);
        state.strain(6:6:end) = 2*state.strain(6:6:end);
    end

    % Sh
    state.Sh = sys.C*state.strain;

    % S via sparse indexing into Sh
    state.S = sparse(sys.indices.SI,sys.indices.SJ,...
        state.Sh(sys.indices.Shidx),...
        sys.dofs^2*sys.ngauss,sys.dofs^2*sys.ngauss);
```

```
end

function state = fea_structural_forces(sys, solver, state)
     U = state.U;
     J = state.J;

     % strain
     X = (sys.Ji0*J)'; % transposed deformation gradient
     deformation = X'*X; %
     e = 1/2*(deformation-speye(sys.dofs*sys.ngauss));
     state.strain = e(sys.indices.idxstrain);

     % Cut out numerical noise on the unity cancellations
     state.strain = state.strain.*(abs(state.strain) > eps);

     % Double the shear strain to convert to engineering strains.
     % TODO: make this more elegant
     if sys.dofs == 2
          state.strain(3:3:end) = 2*state.strain(3:3:end);
     elseif sys.dofs==3
          state.strain(4:6:end) = 2*state.strain(4:6:end);
          state.strain(5:6:end) = 2*state.strain(5:6:end);
          state.strain(6:6:end) = 2*state.strain(6:6:end);
     end

     % BL = BL0+BL1
     BL = sys.BL0+BL1(sys,U);

     % Sh
     state.Sh = sys.C*state.strain;

     % S via sparse indexing into Sh
     state.S = sparse(sys.indices.SI,sys.indices.SJ,...
          state.Sh(sys.indices.Shidx),...
```

```
        sys.dofs^2*sys.ngauss,sys.dofs^2*sys.ngauss);

    %%%% INTERNAL LOADS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % KL = int(BL'*C*BL)
    KL = BL'*(sys.aJd.st*sys.C)*BL;

    % KNL = int(BNL'*S*BNL)
    KNL = sys.BNL'*(sys.aJd.S*state.S)*sys.BNL;

    % F = int(BL'*Sh)
    F = BL'*sys.aJd.st*state.Sh;

    if sys.dofs == 2 % Use defined height
        KL = KL*sys.height;
        KNL = KNL*sys.height;
        F = F*sys.height;
    end

    state.F = F;
    state.KL = KL;
    state.KNL = KNL;
    state.K = KL+KNL;
end

function BL1 = BL1(sys, U)
    %u1 = U(1:2:end); u2 = U(2:2:end);
    uvec = [fea_vec(U,sys.dofs) zeros(sys.nnodes, 3-sys.dofs)];
    u1 = uvec(:,1); u2 = uvec(:,2); u3 = uvec(:,3);

    L = [kron(eye(sys.dofs),u1) kron(eye(sys.dofs),u2) kron(eye(sys.dofs),u3)];
    L = L(:,1:sys.dofs^2);
    L = sys.DH*L;
```

```
    L = sparse(sys.indices.idxLi,sys.indices.idxLj,L);

    BL1 = L*sys.BNL;
end
```

A.2.18 Load functions

```
function [R, Kload] = fea_magnetic_load(sys,solver,state)
    if abs(state.V) > 0
        [fb,fb_U] = fea_magnetic_body_force(sys,solver,state);
        R = sys.Hv'*sys.alpha*sys.Jd0*fb;
        Kload = sys.Hv'*sys.alpha*sys.Jd0*fb_U*sys.Hv;
    else
        R = 0;
        Kload = sparse(sys.dofs*sys.nnodes,sys.dofs*sys.nnodes);
    end

    if solver.disable_z
        R(3:3:end) = 0;
        Kload(:,3:3:end) = 0;
    end
end

function [R, Kload] = fea_tip_load(sys,~,state)
        % Scale Vin s.t. Vin = 10*n (as defined in Pai 1996) and the tip load
        % is n*pi^2*E*I22/4L^2

        I22 = sys.width^3*sys.height/12;
        n = state.V*10;

        xs = sys.Hvs*sys.x0;
        xsvec = fea_vec(xs,sys.nsgauss);

        q = n*pi^2*sys.E*I22/4/sys.length^2/sys.width;
```

```
        if sys.dofs == 3
            q = q/sys.height;
        end

        endpts = abs(xsvec(:,1)-sys.length)/(sys.length)<1e-5;
        fs = zeros(size(xsvec)); fs(endpts,2) = q;
        fs = fea_unvec(fs);

        R = sys.Hvs'*sys.alpha_s*sys.Jds*fs;
        Kload = sparse(sys.dofs*sys.nnodes,sys.dofs*sys.nnodes);
end

function [R, K] = fea_uniform_load(sys,~,state)
    % Pai 1996: load is n*EI22/L^3, n goes from 0 to 20

    I22 = sys.width^3*sys.height/12;
    n = 20*state.V;

    xs = sys.Hvs*sys.x0;
    xsvec = fea_vec(xs,sys.nsgauss);
    q = n*sys.E*I22/sys.length^3;

    if sys.dofs == 3
        q = q/sys.height;
    end

    topsurfacepts = abs(xsvec(:,2)-sys.width/2)/(sys.width/2)<1/100;
    fs = zeros(size(xsvec)); fs(topsurfacepts,2) = q;
    fs = fea_unvec(fs);

    R = sys.Hvs'*sys.alpha_s*sys.Jds*fs;
    K = sparse(sys.dofs*sys.nnodes,sys.dofs*sys.nnodes);
end
```

A.3 Complex-valued Gaussian Process regression

A.3.1 GPR prediction

```
function [Yt, Yt_var, hp] = cgpr_predict(X, Y, Xt, hp, learn)
    % CGPR_PREDICT complex gaussian process regression.
    %
    %    YT = CGPR_PREDICT(X, Y, XT) generates predicted values YT at sample
    %    points XT based on training data (X, Y).  The underlying latent
    %    function is modeled as
    %          Y = f(X) + E,
    %    where X and/or Y may be complex and E is Gaussian white noise.
    %    Predicted function values Yt = f(Xt) are generated from a complex
    %    Gaussian Process regression model.  Each row of X should represent an
    %    input variable and each column a data point.
    %
    %    [YT, YT_VAR, HP] = CGPR_PREDICT(X, Y, XT) additionally returns the
    %    model variance YT_VAR at the sample points XT and the hyperparameters
    %    HP used for the model.
    %
    %    [YT, YT_VAR, HP] = CGPR_PREDICT(X, Y, XT, HP, LEARN) uses the
    %    hyperparameter vector HP as either the starting guess for the
    %    hyperparameter optimization (if LEARN == true) or skips learning and
    %    just uses HP (if LEARN == false).  [...] = CGPR_PREDICT(X, Y, XT, HP)
    %    is the same as [...] = CGPR_PREDICT(X, Y, XT, HP, false).
    %
    %    For an M-input GPR (X has M rows), HP(1:M) are length scales for the
    %    M input variables. HP(M+1) is the overall output variance. HP(M+2) is
    %    the estimated variance of the noise variable E.
    %
    %    Examples:
    %         % To learn hyperparameters and sample at Xt based on (X, Y) data:
    %         [Yt, Yt_var, hp] = cgpr_predict(X, Y, Xt);
    %
```

```
%         % Provide an initial guess for hyperparameter learning:
%         [Yt, Yt_var, hp] = cgpr_predict(X, Y, Xt, hp0, true);
%
%         % Resample the same GP with new data:
%         X = [X X1]; Y = [Y; Y1];
%         [Yt, Yt_var] = cgpr_predict(X, Y, Xt, hp);


if nargin < 2, error('Not enough inputs.'); end
if nargin < 3, Xt = []; end

% Ensure X and Xt have appropriate dimensions.  Assumes number of data
% points is greater or equal to number of input variables.
nx = size(X,1);
if nx > size(X,2), X = X.'; nx = size(X,1); end
if size(Y,2) > size(Y,1), Y = Y.'; end
assert(size(Y,1) == size(X,2),'Sample input and output dimensions must match.');
if nx ~= size(Xt, 1), Xt = Xt.'; end
if ~isempty(Xt) && nx ~= size(Xt, 1)
     error('Training points X and sample points Xt should have the same number of rows.');
end

if nargin < 5 || isempty(learn), learn = false; end % default: no learning
if nargin < 4 || isempty(hp),
     % generate a starting guess for the hyperparameters & enable learning
     hp = ones(nx+2,1);
     hp(nx+1:nx+2) = std(Y);
     learn = true;
end

if learn, hp = learn_hyperparameters(X, Y, hp); end

if ~isempty(Xt)
```

```
        % If test points were supplied, compute estimated Yt & variance
        [Yt, Yt_cov] = gpr_predict(X, Y, Xt, hp);
        Yt_var = diag(Yt_cov);
    elseif nargout < 3
        error('No sample points provided.');
    else
        % No test points, just return the hyperparameters
        Yt = [];
        Yt_var = [];
    end

end

function p = learn_hyperparameters(X,Y,p0)
    % Hyperparameter estimation via log marginal likelihood maximization.
    % See Rasmussen & Williams, ch. 5.
    opt = optimoptions('fminunc',...
        'GradObj','on','algorithm','quasi-newton','Display','off');
    p = fminunc(@(p) log_likelihood(p, X, Y), log(p0), opt);
    p = exp(p);
end

function [K, dK] = kernel_fun(X1, X2, log_params)
    % Squared exponential kernel,
    %      k(x1, x2) = sf^2 exp( - (x1 - x2)' G (x1 - x2) )
    % where G = diag(1/gamma1^2, 1/gamma2^2, ...) is the length scales and sf
    % is the signal variance.

    n1 = size(X1,2); % number of data points in X1
    n2 = size(X2,2); % number of data points in X2
    m = size(X1,1); % number of variables at each data point

    % vectorized distance function
```

```
cdist = @(x1, x2) repmat(x1,1,n2)-repmat(x2.',n1,1);

% Initialize data structures
C = zeros(n1, n2);
dk = cell(m,1);
dK = cell(m+1,1);

% Get the parameter values from their logs
gam = max(1e-4,exp(log_params(1:m))); % length scales
sf = exp(log_params(m+1)); % signal variance

% build up the argument to exp component by component
for i = 1:m
     Ci = cdist(X1(i,:).'/gam(i), X2(i,:).'/gam(i)); % = (X1i-X2i)/Gi
     dk{i} = conj(Ci).*Ci; % = (X1i-X2i)' (X1i-X2i) / Gi^2 = Ci/Gi^2
     C = C + dk{i}; % sum over all input variables
end

% now take the exponent
K = sf^2*exp(-C/2);

% compute derivatives of K w.r.t. log of hyperparameters
% K = sf^2 * exp( -C1/2/G1^2 - C2/2/G2^2 -...)
% dK/d(log(Gi)) = K * d(-Ci/2/Gi^2)/d(log(Gi))
%                 = -K * Ci/2*d(Gi^-2)/d(log(Gi))
%                 = -K * Ci/2 * d(exp(-2 U))/d U, U = log(Gi)
%                 = -K * Ci/2 * -2 / Gi^2
%                 =  K * (Ci/Gi^2)
for i = 1:m
     dK{i} = K.*dk{i};
end

% K = sf^2 * exp( -C1/2/G1^2 - C2/2/G2^2 -...)
```

```
    % dK/d(log(sf)) = exp(...) * 2 * sf * d(sf)/d(log(sf))
    %                 = 2 * sf^2 * exp(...) = 2*K
    dK{m+1} = 2*K;
end

function [L, dL] = log_likelihood(log_params, X, Y)
    if nargout > 1
        [K, dK] = kernel_fun(X, X, log_params);
    else % don't compute derivatives if not needed
        K = kernel_fun(X,X,log_params);
    end

    n = length(K); % number of sample points

    % noise standard deviation
    sigma_e = exp(log_params(end));
    dd = 1e-12; % small regularizing term to ensure C is positive definite
    Ky = K + (sigma_e^2+dd)*eye(n); % covariance of y, Ky (including noise)

    [cholKy, flag] = chol(Ky,'lower');
    if flag ~= 0 % C was not positive definite
        L = 1e20;
        dL = zeros(size(log_params));
        return
    end

    LiY = cholKy\Y;
    KiY = cholKy'\LiY;

    % log p = -Y' K^-1 Y/2 - log(det(K))/2 - n/2*log(2 pi)
    % cf. Rasmussen & Williams p. 114
    % Cast as a minimization problem by returning -log p
    L = (1/2*(LiY'*LiY) + trace(log(cholKy)) + n/2*log(2*pi));
```

```
    if nargout > 1
        cholKyi = cholKy \ eye(n);
        dL = zeros(size(log_params));
        for i = 1:length(log_params)-1
            % d log p / d theta_i = 1/2 y' K^-1 dK/d theta_i K^-1 y
            %                          - trace(K^-1 dK/d theta_i)/2
            % NOTE: trace(A*B) = sum(sum(A.*B.'))
            dL(i) = -0.5*(KiY'*dK{i}*KiY) + 0.5*sum(sum(cholKyi.*(cholKy \ dK{i})));
        end
        % For the noise hyperparameter:
        % d C / d log(sigma_e) = d(sigma_e^2 I)/d log(sigma_e)
        %                      = 2 sigma_e * d sigma_e / d log sigma_e
        %                      = 2 sigma_e^2
        dL(end) = -sigma_e^2*(KiY'*KiY) + sigma_e^2*sum(sum(cholKyi.*cholKyi));

        % dL should be real but the complex component may not cancel
        dL = real(dL);
    end

end

function [Ye, Ye_cov] = gpr_predict(X, Y, Xt, params)
    K = kernel_fun(X, X, log(params));
    Kt = kernel_fun(Xt, X, log(params));
    nt = length(Y);
    n = size(Xt,2);
    sigma_e = params(end);
    Ky = K + sigma_e^2*eye(nt);

    try % solve using cholesky decomposition
        cholKy = chol(Ky,'lower');
        Ye = Kt*(cholKy'\(cholKy\Y));
        v = cholKy\Kt';
```

```
    catch % cholesky failed, use mldivide (slower)
        Ye = Kt*(Ky\Y);
        v = Kt'\Ky/Kt;
    end

    % cf. Rasmussen & Williams p. 19
    Ye_cov = kernel_fun(Xt,Xt,log(params)) - v'*v + sigma_e^2*eye(n);
end
```

A.3.2 Iterative gain

```
function rho = gpr_iterative_gain(Gi_est, Gicov, frac)
    % Get the iteration gain based on the GPR-estimated model error

    Gi_r = real(Gi_est); Gi_j = imag(Gi_est);

    % variance is the diagonal of the covariance
    if size(Gicov,1) == size(Gicov,2)
        Gicov = diag(Gicov); % get the variance only
    end

    var_r = (Gicov); var_j = var_r;

    % get the error in real & imaginary parts based on the variance---take the
    % error to be 2 standard deviations
    err_r = 2*sqrt(var_r); err_j = 2*sqrt(var_j);

    % Gain
    rho = max(0,...
        frac*2*(Gi_r.^2 + Gi_j.^2 - abs(Gi_r).*err_r - abs(Gi_j).*err_j)./...
        ((abs(Gi_r) + err_r).^2 + (abs(Gi_j)+err_j).^2));
end
```