Title
Controlled Pattern Formation in a Reaction-Diffusion System: A Novel Application of Non-Linear Model Predictive Control to Distributed Parameters Systems

Author
Schuler, Alejandro

Publication Date
2013

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Controlled Pattern Formation in a Reaction-Diffusion System:

A Novel Application of Non-Linear Model Predictive Control to Distributed Parameters Systems

A thesis submitted in partial satisfaction of the requirements for the degree of Master of Science in Mechanical Engineering

by

Alejandro Schuler

2013

ABSTRACT OF THE THESIS

Controlled Pattern Formation in a Reaction-Diffusion System:

A Novel Application of Non-Linear Model Predictive Control to Distributed Parameters Systems

by

Alejandro Schuler

Master of Science in Mechanical Engineering

University of California, Los Angeles, 2013

Professor Chih-Ming Ho, Chair

In this work, we examine the application of open-loop non-linear model predictive control (NMPC) to a specific partial differential equation (PDE) reaction-diffusion system. Instead of attempting to stabilize a particular state, we focus only on reaching the desired state at a given time. This problem is motivated by applications to *in-vitro* synthesis of tissues whose formation is governed by reaction-diffusion models. The PDE model is discretized into a finite set of nonlinear ordinary differential equations using finite-elements, after which a NMPC algorithm is used to derive the desired control actions. Results for one- and two- dimensional systems demonstrate the feasibility of the approach, but also highlight a dependency on the number of actuators and computational power.

The thesis of Alejandro Schuler is approved.

Panagiotis D. Christofides

James S. Gibson

Chih-Ming Ho, Committee Chair

University of California, Los Angeles

2013

for Allegra

and for my parents,

Nathalie & Carlos

Lorraine & Craig

Contents

List of Figures

Chapter 1

Introduction and Motivation

Pattern formation during the development of organisms is a process that has long interested biologists. What are the processes that lead to the creation of the multitude of patterned structures that exist in the natural world, such as the stripes and spots of animal coats, or the branching morphology of trees, blood vessels, and lungs? How is it that these intricate structures arise from a relatively homogeneous starting point such as a zygote or seed?

These questions caught the attention of Alan Turing, who was the first to propose that a coupled system of partial differential equations describing the dynamics of two chemical agents ("morphogens") interacting and diffusing could be enough to explain the phenomenon of stripe and spot formation[2]. The concept behind reaction-diffusion models is normally that one chemical serves as an activator and the other as an inhibitor; the activator promotes production of itself and of the inhibitor, whereas the inhibitor decreases production of the activator. Letting the inhibitor diffuse more quickly than the activator allows it to have an edge over the activator in a "global" sense, whereas the activator tends to dominate "locally" wherever it is being produced. Plotting the concentration of one or both of the agents in the space results in a pattern of spots or stripes. Notably, this pattern emerges naturally over time if the system starts in a state where both chemicals are distributed evenly in the domain, but with minor random perturbations- a situation analogous to what we see in biological development.

More recently, the work of Garfinkel et al. [1] proposed a specific two-agent model based on the experimental reaction dynamics of two hypothesized morphogens for vascular mesenchymal cells. They tested simulations based on their model against experiments in which they added inhibitor to cells grown in culture to verify that their model qualitatively behaves in the same manner as the process of interest. By doing so,

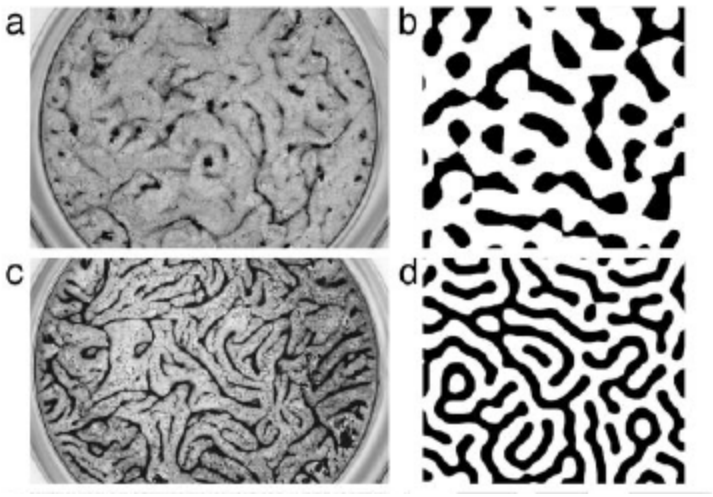


Figure 1.1: Garfinkel et al. varied the amount of an inhibiting morphogen added to their cell cultures, resulting in qualitatively different patterns. On the right, similar conditions are simulated *in silico*, demonstrating that their model captures the phenomenon [1].

they also proved that the addition of the inhibitor changed the pattern which was produced.

This work will focus on the possibility of controlling the formation of a particular pattern in Garfinkel's reaction-diffusion system. By manipulating the system at certain points in space and time, we will attempt to drive it towards a preselected pattern. To derive the necessary manipulation, we will use an open-loop model predictive control (MPC) [3] scheme. This approach is similar to work undertaken by Dubljevic et al. [4] [5] [6], in which theoretical and computational results for MPC implementation on parabolic PDEs are discussed. These works, however, are focused on deriving stabilizing controllers, while the problem we wish to address here concerns forcing the system to arrive at a desired non-equilibrium state at a given time.

Chapter 2

Problem Statement

The model is as follows: $\forall x, t \in \Omega \times [0, t_f]$,

$$\begin{aligned} \frac{\partial A}{\partial t} &= \rho \nabla^2 A + \gamma \left[\frac{A^2}{(1 + kA^2)B} - cA \right] + u_A(x,t) \\ \frac{\partial B}{\partial t} &= \nabla^2 B + \gamma(A^2 - eB) + u_B(x,t) \end{aligned} \qquad \textit{The Garfinkel Model}$$

$$\begin{aligned} A(x,0) &= A_0, \quad B(x,0) = B_0 && \textit{Initial Conditions} \\ \nabla A(x \in \partial\Omega, t) \cdot n_{\partial\Omega} &= 0, \quad \nabla B(x \in \partial\Omega, t) \cdot n_{\partial\Omega} = 0 && \textit{Zero flux on the domain boundary} \end{aligned} \tag{2.1}$$

$$y(x,t) = \Psi(A(x,t), B(x,t)) \quad \textit{Pattern, or output}$$

where D, γ, k, c, and e are constant parameters which are determined experimentally, x is a general position vector, $n_{\partial\Omega}$ is a vector normal to $\partial\Omega$ and $y(x,t)$ is defined as the pattern generated by the state of the system $A(x,t), B(x,t)$ at time t. As an example of y, dark pigmentation in an animal coat could be produced in a particular location only if the level of activator is over a certain threshold; a form of y for this case might be: $y(x,t) = H(A(x,t) - A_{threshold})$ where H is the Heaviside step function.

Our goal is to find the "best" functions $u_A(t), u_B(t)$ which will drive the system from y_0 to y_{des}, and so we wish to find a $u(t)$ to minimize

$$J = \int_\Omega \phi(x)[y(x,t_f) - y_{des}(x)]^2 dx + \nu \int_{t_0}^{t_f} \int_\Omega \psi_1(x) u_1(x,t)^2 dx dt + \nu \int_{t_0}^{t_f} \int_\Omega \psi_2(x) u_2(x,t)^2 dx dt \tag{2.2}$$

where $\phi(x)$ is a weighting function representing the relative importance of the correctness of the pattern at a particular location, $\psi_i(x)$ are weighting functions penalizing u_i differently depending on the position of the control application, and ν_i are weights emphasizing the relative importance of u_A to u_B and to the pattern error $y - y_{des}$.

Chapter 3

Model Discretization

3.1 Flux-Controlled One-Dimensional Finite Element Scheme

In order to use MPC algorithms[3] to find the desired control, we must have a finite-state, discrete-time approximation of (2.1). To this end, we take our system of partial differential equations (2.1) and discretize it spatially into a finite number of ordinary differential equations (ODEs) using a finite element scheme. The resulting ODE system is then discretized temporally into a discrete time (DT) model. The states of the ODEs associated with an finite element model describe the time evolution of the concentrations of the morphogens at specific points in space along the domain. This is a practical advantage of a finite element scheme because the resulting state variables could be directly measurable in an experimental setting.

We express the Galerkin weak form of the 1D model (with control terms removed) over a single element $x \in [x_a x_b]$

$$\int_{x_a}^{x_b} w_i(x,t)^T \begin{bmatrix} -\rho[\partial_{xx}A] - g_A(A,B) + \partial_t A \\ -\partial_{xx}B - g_B(A,B) + \partial_t B \end{bmatrix} dx = 0, \quad w_i(x,t) = \begin{bmatrix} w_{iA}(x,t) \\ w_{iB}(x,t) \end{bmatrix}$$

Where w is an arbitrary weight function and the functions $g_X(\cdot)$ are the nonlinear terms

$$g_A(A,B) = \gamma \left[\frac{A^2}{(1+kA^2)B} - cA \right], \quad g_B(A,B) = \gamma(A^2 - eB) \tag{3.1}$$

Integrating by parts,

$$0 = \int_{x_a}^{x_b} \frac{dw_i}{dx} \begin{bmatrix} \rho\partial_x A \\ \partial_x B \end{bmatrix} dx + \int_{x_a}^{x_b} w_i \begin{bmatrix} \partial_t A - g_A \\ \partial_t B - g_B \end{bmatrix} dx + w_i(x_a) \begin{bmatrix} \rho\partial_x A(x_a,t) \\ \partial_x B(x_a,t) \end{bmatrix} - w_i(x_b) \begin{bmatrix} \rho\partial_x A(x_b,t) \\ \partial_x B(x_b) \end{bmatrix} \tag{3.2}$$

Which is the desired standard weak form of the PDE. We now use the ansatz

$$w_j(x) = L_j \quad \text{and} \quad \begin{bmatrix} A(x,t) \\ B(x,t) \end{bmatrix} \approx \sum_{i=1}^{m} \begin{bmatrix} L_i(x)A_i(t) \\ L_i(x)B_i(t) \end{bmatrix} \tag{3.3}$$

and the linear interpolation functions

$$L_1 = \frac{x_b - x}{x_b - x_a} = 1 - \xi \qquad L_2 = \frac{x - x_a}{x_b - x_a} = \xi \tag{3.4}$$

in order to construct an approximative system of ODEs. With some manipulation of (3.2-3.4)we obtain the following ODE system for each element

$$0 = \frac{1}{h^e} \begin{bmatrix} \rho & -\rho & 0 & 0 \\ -\rho & \rho & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} + \frac{h^e}{6} \begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \end{bmatrix} + \begin{bmatrix} \rho\partial_x A(x_a) \\ -\rho\partial_x A(x_b) \\ \partial_x B(x_a) \\ -\partial_x B(x_b) \end{bmatrix} - h^e \begin{bmatrix} G_{A1} \\ G_{A2} \\ G_{B1} \\ G_{B2} \end{bmatrix} \tag{3.5}$$

where $h^e = x_b - x_a$ is the length of the element. In this construction, the flux terms cancel for all interior nodes such that the flux vector will only be nonzero on the node at $x = L$, where $\rho\partial_x A(L) = u_A(t)$ and $\partial_x B(L) = u_B(t)$. The vector of terms G_{Xi} requires more attention since it is, strictly speaking, a nonlinear vector function of A_1, A_2, B_1, and B_2. Expressed analytically by combining (3.1-3.5),

$$\begin{aligned} G_{Ai}(A_1, A_2, B_1, B_2) &= \int_0^1 L_i\gamma \left[\frac{(A_1L_1 + A_2L_2)^2}{[1 + k(A_1L_1 + A_2L_2)^2](B_1L_1 + B_2L_2)} - c(A_1L_1 + A_2L_2) \right] d\xi \\ G_{Bi}(A_1, A_2, B_1, B_2) &= \int_0^1 L_i\gamma[(A_1L_1 + A_2L_2)^2 - e((B_1L_1 + B_2L_2)]d\xi \end{aligned} \tag{3.6}$$

The first of these expressions (G_{Ai}) is tedious to evaluate analytically and so we continue by evaluating

G_{A1} and G_{A2} numerically. The second integral (for G_{Bi}) simplifies nicely to

$$\begin{aligned} G_{B1}(A_1, A_2, B_1, B_2) &= \frac{\gamma}{12}(3A_1^2 + 2A_1A_2 + A_2^2 - 4eB_1 - 2eB_2) \\ G_{B2}(A_1, A_2, B_1, B_2) &= \frac{\gamma}{12}(A_1^2 + 2A_1A_2 + 3A_2^2 - 2eB_1 - 4eB_2) \end{aligned} \tag{3.7}$$

We have now arrived at the desired ODE approximation of the system. We define X as a vector formed from the A_i's and B_i's and assemble the elemental matrices (3.5) appropriately to yield a global ODE system of the form

$$0 = KX + M\dot{X} + Q - G(X) \tag{3.8}$$

where Q contains the flux (controlled) terms. This system is converted to discrete time by marching forward from the initial conditions using an implicit Euler scheme

$$X_{t+\Delta t} = \underbrace{\left(\frac{1}{\Delta t}M + K\right)^{-1}\left(\frac{1}{\Delta t}M\right)}_{T} X_t + \underbrace{\left(\frac{1}{\Delta t}M + K\right)^{-1}}_{P}[G(X_t) - Q_{t+\Delta t}] \tag{3.9}$$

$$X_{t+\Delta t} = TX_t + PG(X_t) - PQ_{t+\Delta t} \tag{3.10}$$

where we have made use of the approximation $G(X_{t+\Delta t}) \approx G(X_t)$ in order to avoid a costly iterative solution procedure.

3.2 Two-Dimensional Distributed Control Finite Element Model

The construction of a two-dimensional finite-element model follows conceptually from the methods used to derive the single-dimensional model. The Galerkin weak form is given by

$$0 = \int_\Omega \frac{dw_i}{dx}\begin{bmatrix} \rho\partial_x A \\ \partial_x B \end{bmatrix} + \frac{dw_i}{dy}\begin{bmatrix} \rho\partial_y A \\ \partial_y B \end{bmatrix} dxdy + \int_\Omega w_i \begin{bmatrix} \rho\partial_t A - g_A - u_A \\ \partial_t B - g_B - u_B \end{bmatrix} dxdy + \oint_{\partial\Omega} w_i q ds \tag{3.11}$$

where the g_i terms are defined in (3.1). Using (3.4) and linear triangular finite elements we obtain the matrix equation

$$0 = K^e X^e + M^e \dot{X}^e + U^e - G^e(X^e) + Q^e \tag{3.12}$$

For $X_i^e = \begin{bmatrix} A_i \\ B_i \end{bmatrix}$ and interpolation functions L_i,

$$\begin{aligned} K_{ij}^e &= \int_\Omega \left(\frac{dL_i}{dx}\frac{dL_j}{dx} + \frac{dL_i}{dy}\frac{dL_j}{dy} \right) \begin{bmatrix} \rho & 0 \\ 0 & 1 \end{bmatrix} dxdy \\ M_{ij}^e &= \int_\Omega L_i L_j \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} dxdy \end{aligned} \tag{3.13}$$

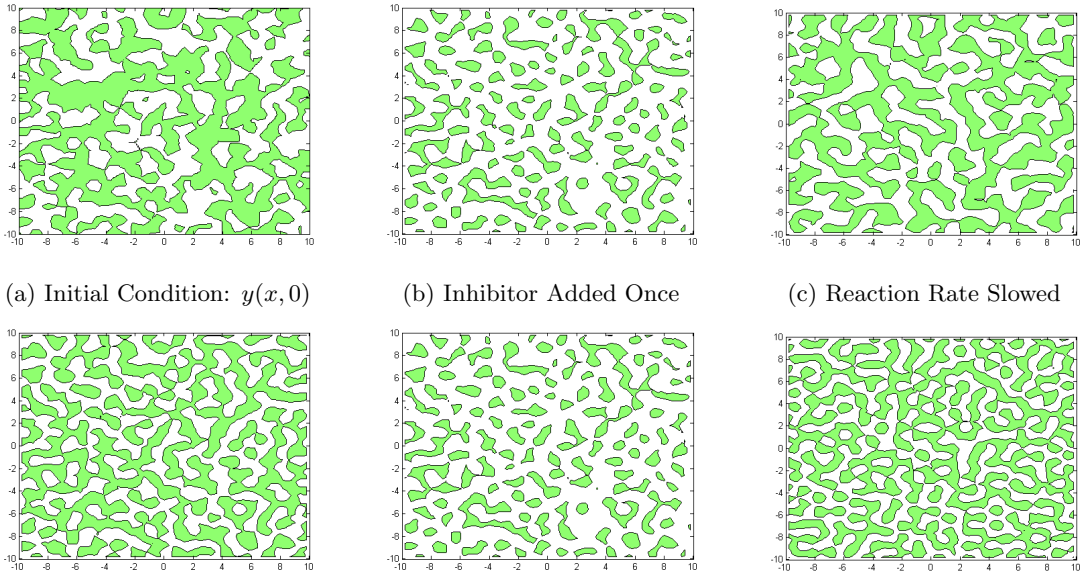
$$\begin{aligned} G_i^e &= \int_\Omega \begin{bmatrix} L_i g_A \\ L_i g_B \end{bmatrix} dxdy \\ U_i^e &= \int_\Omega \begin{bmatrix} L_i u_A \\ L_i u_B \end{bmatrix} dxdy \\ Q_i^e &= \oint_{\partial\Omega} \begin{bmatrix} L_i q_A \\ L_i q_B \end{bmatrix} ds \end{aligned} \tag{3.14}$$

The matrices in (3.13) are evaluated analytically while the matrices in (3.14) are evaluated numerically using quadrature. After global assembly, application of an implicit Euler scheme yields

$$X_{t+\Delta t} = TX_t + PG(X_t) + PU_{t+\Delta t} - PQ_{t+\Delta t} \tag{3.15}$$

where T and P are defined as in (3.9). Along with (3.10), this is the approximation of (2.1) that we will use in the proceeding analysis to compute the optimal controls to generate desired patterns.

This scheme was implemented in MATLAB[7] on a 20 by 20 square domain meshed[8] into triangles with sides of average length 0.7. Unless otherwise noted, all simulations were run using parameters $\rho = 0.005$, $c = 0.01$, $k_p = 0.65$, $e = 0.02$, and $\gamma = 100$. Simulations were run to a final time of $t_f = 3s$ with a time step $\Delta t = 0.05s$. A reproduction of the results of Garfinkel et al[1] is shown in figure 1.1.



(a) Initial Condition: $y(x, 0)$ (b) Inhibitor Added Once (c) Reaction Rate Slowed

(d) Control Case: $y(x, t_f)$ (e) Inhibitor Added Continually (f) Reaction Rate Accelerated

Figure 3.1:
Simulation results showing areas of the domain for which the concentration of A is greater than the equilibrium value $A_{ss} = 1.11031$. In other words, a plot of the function $y(x,t) = H(A(x,t) - A_{ss})$. **(a)** The randomly generated initial condition: $A(x,y) \sim \mathcal{N}(A_{ss}, 0.02A_{ss})$. **(d)** System state at t_f with no perturbation or forcing. A stripe-like pattern is visible. **(b)** The value of $B(x,y)$ is artificially increased once by 1 at $t = 0.1s$. The result y is shown at t_f and the pattern appears spot-like. **(e)** The value of $B(x,y)$ is artificially increased continually by 0.4 at each time step. The result y is shown at t_f and the pattern appears spot-like. **(c)** The result at t_f of the unperturbed dynamics using $\gamma = 50$. The characteristic size of the pattern features is increased. **(f)** The result at t_f of the unperturbed dynamics using $\gamma = 200$. The characteristic size of the pattern features is decreased.

Chapter 4

Model Predictive Control

The basic idea of an open-loop MPC scheme is the use of the knowledge of the system's dynamics to optimize the future behavior of the system [9]. Standard applications of MPC involve resolving the optimal control problem at varying intervals and pushing the solution horizon forward in time. In this way, only the temporally immediate control is ever actually applied to the system, "closing the loop" in a sense. In this application, we focus only on solving the optimal control problem at the initial time and applying the entire computed control sequence to the system. An open-loop implementation is an obvious choice for the problem at hand because we do not mean to stabilize the system at the desired pattern, only to achieve a resemblance at the terminal time, at which point we assume the system can be "frozen". For a time span $t(i) = (i-1)\Delta t : \{1, ... N\} \mapsto [0, t_f]$, restating (2.2) in a the discretized context of (3.10) and (3.15): we seek to find a vector sequence U to...

$$\begin{aligned} minimize: &\quad J = (Y(X_N(U)) - Y_{des})\Phi(Y(X_N(U)) - Y_{des}) + \sum_{i=1}^{N} U_i R U_i \\ with\ respect\ to: &\quad U \in \mathbb{R}^U \\ where: &\quad X_{i+1} = TX_i + PG(X_i) + PU_{i+1} - PQ_{i+1} \end{aligned} \tag{4.1}$$

X_N is written as $X_N(U)$ to emphasize its sole dependence on U from a uniform initial condition. This formulation of the optimization problem is known as recursive discretization and is efficient in regard to keeping the optimization variable as small as possible and eliminating constraints [9]. On the other hand, this type of optimization can be slow to converge because no information is provided in the optimization

variable about the reference trajectory. To improve performance, we use a so-called shooting method[10] which adds a vector s of length r_s to the optimization variable as well as a constraint to the problem. The entries in s represent the values of some of the system's states at particular instances in time. Specifically, for a time index $\zeta : \{1, ... \, r_s\} \mapsto \{1, ... \, N\}$ and a state index $\iota : \{1, ... \, r_s\} \mapsto \{1, ... \, n_{states}\}$,

$$s_j = X_{\zeta(j),\iota(j)} \tag{4.2}$$

Using (3.15) and (4.1-4.2), the optimization problem now becomes

$$\begin{aligned} minimize: \quad & J = (Y(X_{t_f}(U)) - Y_{des})\Phi(Y(X_{t_f}(U)) - Y_{des}) + \sum_{i=1}^{N} U_i R U_i \\ with\ respect\ to: \quad & \{U, s\} \in \mathbb{R}^U \times \mathbb{R}^{r_s} \\ subject\ to: \quad & s_j = \left[TX_{\zeta(j)-1} + PG(X_{\zeta(j)-1}) + PU_{\zeta(j)} - PQ_{\zeta(j)}\right]_{\iota(j)} \end{aligned} \tag{4.3}$$

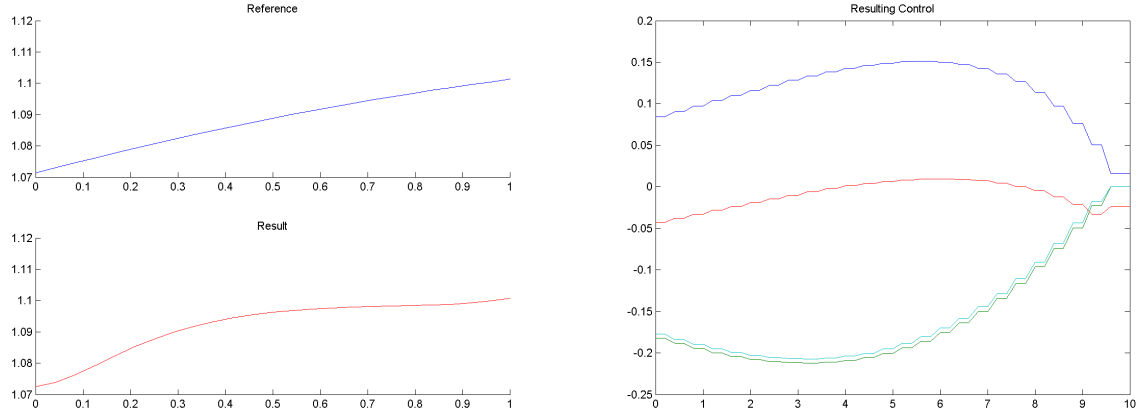
This formulation allows information about the reference trajectory to be embedded into the initial guess for the optimization variable s_0. Because the system is complex and only the desired terminal state is known a priori, it is the only usable point of reference. In our case, we postulate that the relevant output is the state of the activator, A, so we let $Y(X_t) = X_{t,\iota(A)} = A_t$ where $\iota(A)$ refers to the indices of X_t corresponding to the states A_t. The initial guess should therefore be $s_0 = y_{ref}$ with $\zeta(\cdot) = N$ and $\iota = \iota(A)$. Furthermore, we let $\Phi = I * 10^5$ and $R = I$ to emphasize the importance of the accuracy of the result over the control cost and we apply a zero-flux boundary condition: $Q = 0$ to the two-dimensional system. The problem (4.3) becomes

$$\begin{aligned} minimize: \quad & J = 10^5 \times ||A_N - Y_{des}|| + \sum_{i=1}^{N} ||U_i|| \\ with\ respect\ to: \quad & \{U, s\} \in \mathbb{R}^U \times \mathbb{R}^{r_s} \\ subject\ to: \quad & s = [TX_{N-1} + PG(X_{N-1}) + PU_N]_{\iota(A)} \end{aligned} \tag{4.4}$$

In the flux-controlled one-dimensional system, we have the equivalent:

$$
\begin{aligned}
minimize: \quad & J = 10^5 \times ||A_N - Y_{des}|| + \sum_{i=1}^{N} ||U_i|| \\
with\ respect\ to: \quad & \{U, s\} \in \mathbb{R}^U \times \mathbb{R}^{r_s} \\
subject\ to: \quad & s = [TX_{N-1} + PG(X_{N-1}) - PQ_N]_{\iota(A)}
\end{aligned}
\tag{4.5}
$$

The above are solved in MATLAB[7] using `fmincon()` with cost and constraint functions each of which call upon the finite element simulation to provide trajectories X based on U for each iteration.

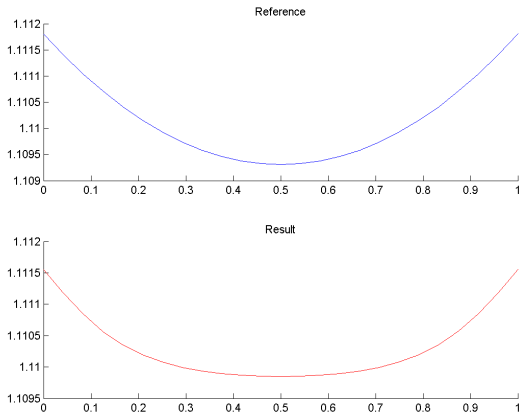


(a) Desired concentration profile at t_f and actual result (b) Result of the optimization of the applied controls

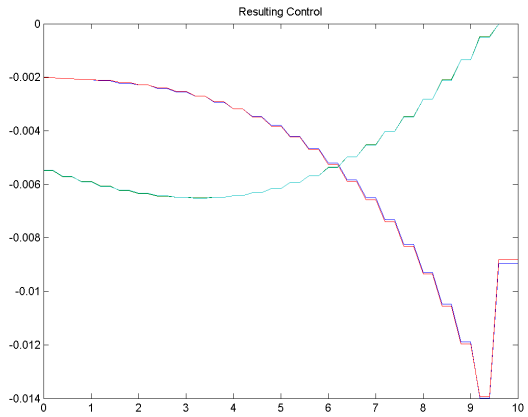
Figure 4.1: Results of the control optimization for the single-dimensional boundary-flux-controlled system. Unless otherwise stated, all results are of the 25-node simulated system driven from a steady state initial condition $A(x,0) = A_{ss}, B(x,0) = B_{ss}$ with physical parameters: $\rho = 0.005$, $c = 0.01$, $k = 0.65$, $e = 0.02$, $\gamma = 10$, $\Delta t = 0.2$. Controls are zero-order held over two timesteps in order to reduce the size of the optimization problem.

As is evident in figures 4.1-4.4, the optimization of boundary-flux controls for the single-dimensional simulation successfully achieved a qualitative replication of the desired concentration profile of the activator given a "simple-looking" reference profile. Figure 4.5 demonstrates that the same optimization algorithm fails to drive the system to a more complex state. This is intuitively due to the fact that the reaction terms in the system overpower diffusion and that the nature of the reaction is local. Controls influencing the system at given locations, therefore, will only affect the system locally. Mathematically speaking, it is also possible that the system is not differentially flat[11], although the proof would be nontrivial and will not be explored here.

There is a possibility that the generated controls are not at a true optimum and that using a better seed

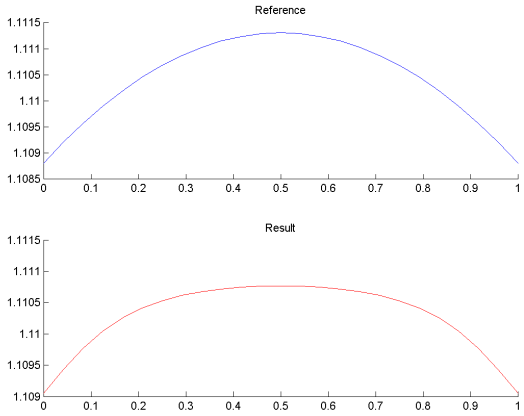


(a) Desired concentration profile at t_f and actual result

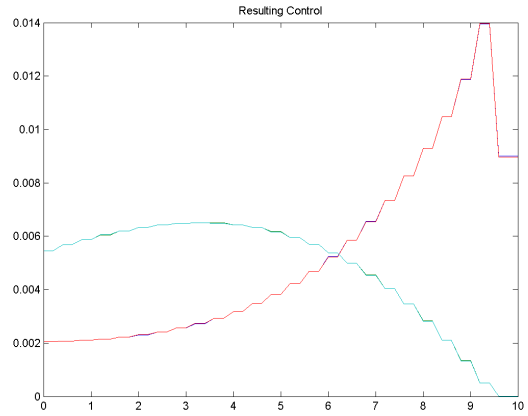


(b) Result of the optimization of the applied controls

Figure 4.2: See caption for Figure 4.1



(a) Desired concentration profile at t_f and actual result

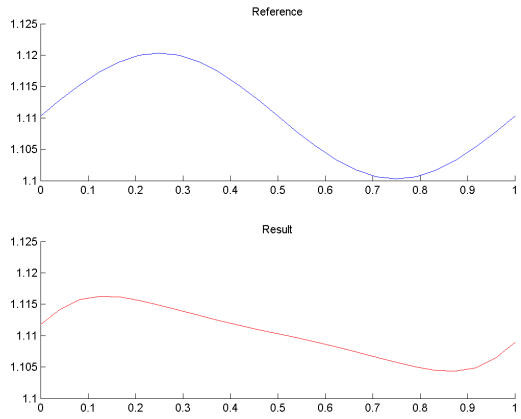


(b) Result of the optimization of the applied controls

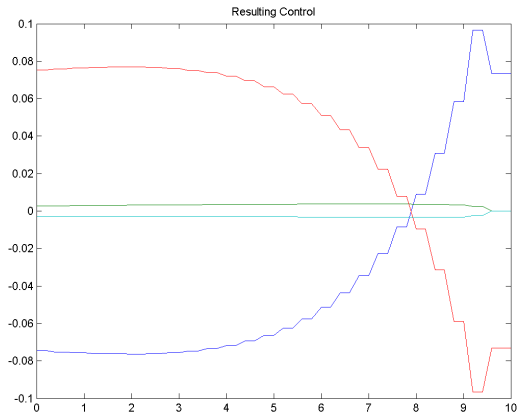
Figure 4.3: See caption for Figure 4.1

in the optimization algorithm would produce a better result. This hypothesis is unlikely, however, given that each optimization was run several times with a variety of randomized initial guesses. In each case, the result converged closely to the reported optimal control. Furthermore, several optimizations which used the stochastic optimization algorithm CMA-ES[12] (instead of the gradient-based `fmincon()`) also produced results convergent with those from `fmincon()`.

The shortcomings are more evident in the results for the distributed control of the two-dimensional simulation. Figures 4.6 and 4.7 show the sensitivity of the result to the desired concentration profile and the placement of the control point (where the activator and inhibitor can be added or removed from the

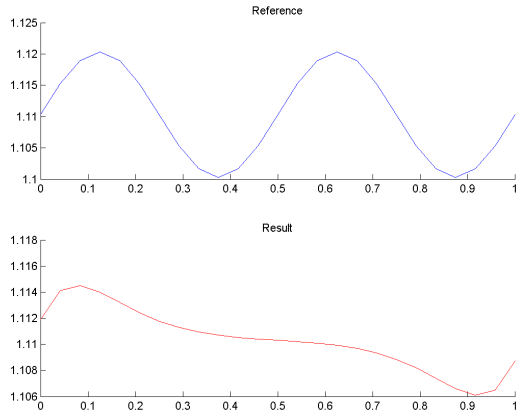


(a) Desired concentration profile at t_f and actual result

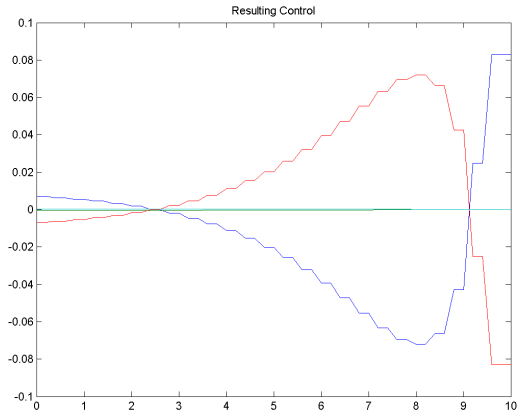


(b) Result of the optimization of the applied controls

Figure 4.4: See caption for Figure 4.1



(a) Desired concentration profile at t_f and actual result



(b) Result of the optimization of the applied controls

Figure 4.5: See caption for Figure 4.1

system). The use of more than a small number of control points is both prohibitively time-consuming in the optimization and experimentally infeasible. Theoretically, however, increasing the number of control points and intelligently distributing them would yield improved results, which is consistent with the results of Dubljevic et al. [5].

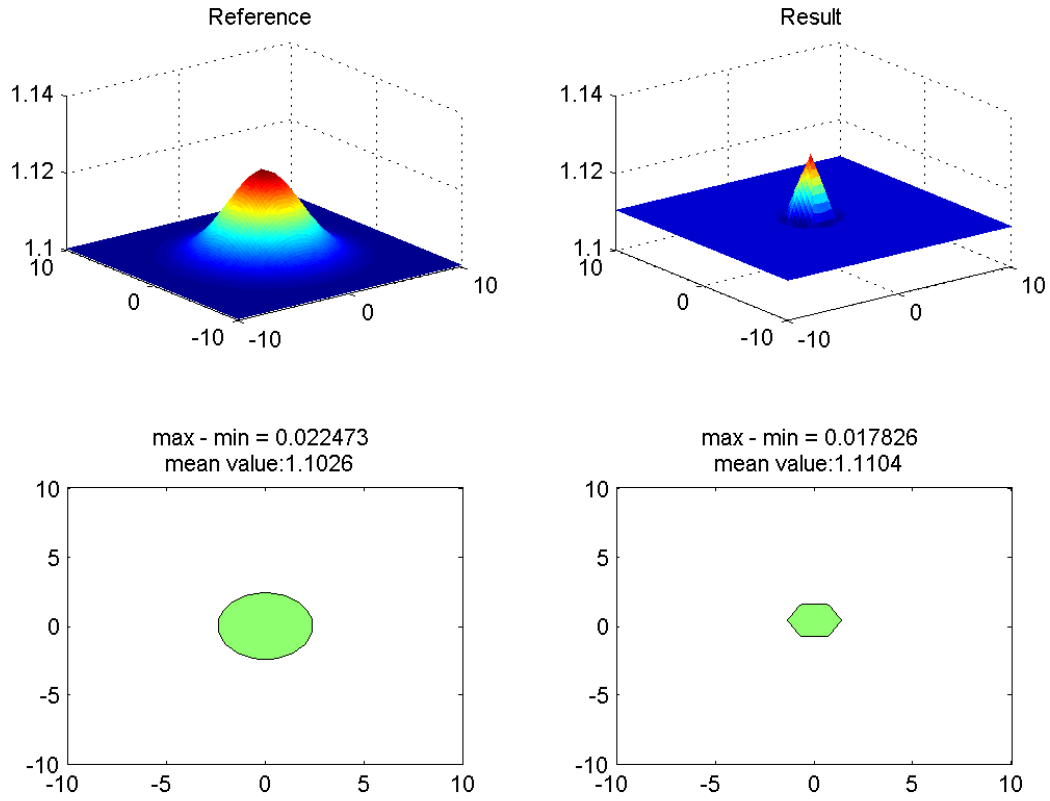


Figure 4.6: Reference and resulting terminal concentration profiles using a single centered control point. Results are of the simulated system with characteristic element size of 0.2, driven from a steady state initial condition $A(x,0) = A_{ss}, B(x,0) = B_{ss}$ with physical parameters: $\rho = 0.005$, $c = 0.01$, $k = 0.65$, $e = 0.02$, $\gamma = 200$, $\Delta t = 0.2$. Controls are zero-order held over two timesteps in order to reduce the size of the optimization problem.

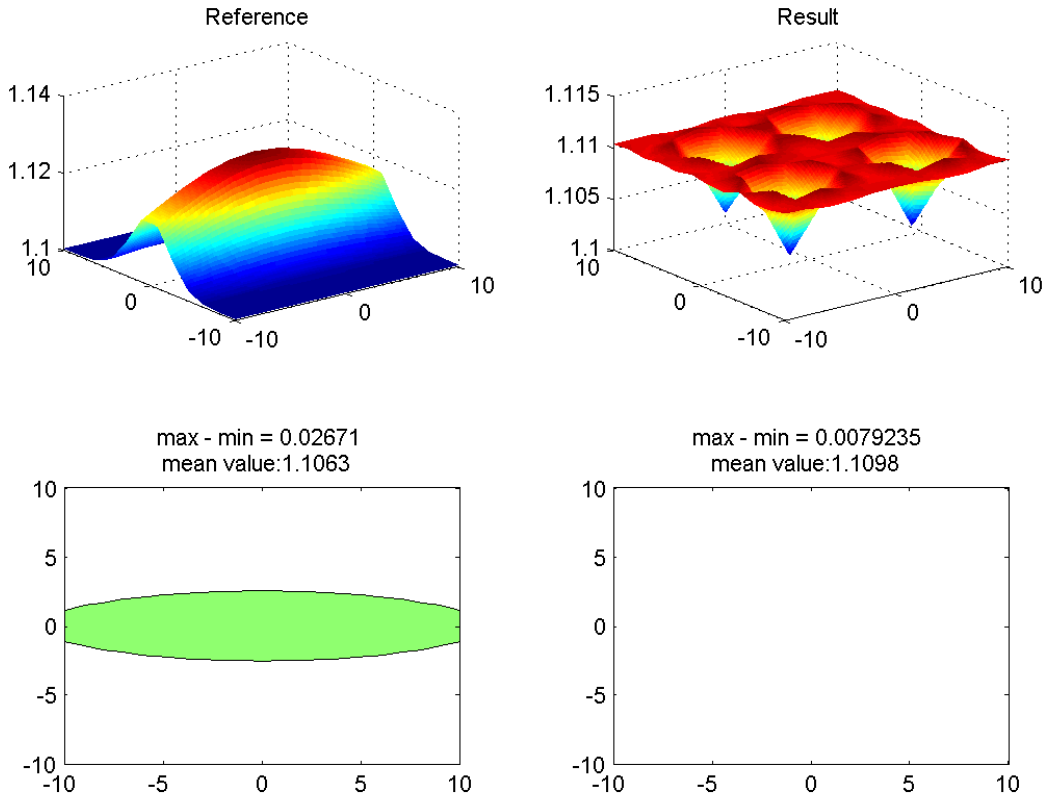


Figure 4.7: Reference and resulting terminal concentration profiles using four evenly spaced control points. Results are of the simulated system with characteristic element size of 0.2, driven from a steady state initial condition $A(x,0) = A_{ss}, B(x,0) = B_{ss}$ with physical parameters: $\rho = 0.005$, $c = 0.01$, $k = 0.65$, $e = 0.02$, $\gamma = 200$, $\Delta t = 0.2$. Controls are zero-order held over two timesteps in order to reduce the size of the optimization problem.

Chapter 5

Conclusion

An open loop MPC scheme has shown partially effective in generating controls to drive Garfinkel's reaction-diffusion system to a desired terminal activator concentration profile. Although simple patterns can be generated, the controlled creation of interesting concentration profiles is out of reach without a practically excessive number of actuation points.

Chapter 6

Code

6.1 1dopt

Executes a single run of the optimization for the single-dimensional flux-controlled system with the specified parameters. Sets up a finite element model object `garfinkelFEmodel` and defines anonymous functions based on `d1constraints()`, `d1cost()`, and `garfinkelFEmodel.fullsim()` which are called by `fmincon()`. Displays results.

```
% Parameter Setup
clear all
clc

Ass = 1.110305545592924; Bss = 61.638920228753065;
%A_t = A_xx = B_t = B_xx = 0. Does not depend on gamma or rho
rho = 0.01/2; c = 0.01; kp = 0.65; e = 0.02; %model params
D = 1; gamma = 10;
udim = 4; %number of controls: A'(0,t) and B'(0,t)

d1model = garfinkelFEmodel; %initialize model object
d1model.params = [gamma,c,kp,e,rho,D]; %parameters
d1model.dt = 0.2; %simulation time step
L = 0.25; N =25; %length of domain, number of nodes
d1model.nodes = linspace(0,L,N); %domain
d1model.setup;
```

```
%----------------------------INITIAL CONDITIONS-----------------------------------------
% A0 = Ass*( 1+0.02*randn(size(d1model.nodes)) );
% B0 = Bss*( 1+0.02*randn(size(d1model.nodes)) );
A0 = Ass*ones(size(d1model.nodes));
B0 = Bss*ones(size(d1model.nodes));
%---------------------------------------------------------------------------------------


%----------------------------TIME PARAMETERS--------------------------------------------
tf = 10; %final time
tsim = 0:d1model.dt:tf; %simulation time vector
MPCstep = d1model.dt*2; %frequency of control application
nsteps = length(0:MPCstep:tf)-1; %how many dt's we have
%---------------------------------------------------------------------------------------



%----------------------------REFERENCE SIGNAL-------------------------------------------
yref = 0.01*sin(d1model.nodes*2*pi/L)' + Ass; %squiggle
%---------------------------------------------------------------------------------------
%NEXT TRY SQUIGGLE


%----------------------------OPTIMIZATION SETUP-----------------------------------------
sdim = length(A0); %number of shooting nodes: I only care about what A looks like

xi = nsteps*ones(sdim,1); %times of shooting nodes (in dt's): final time
ind = 1:2:length([A0 B0]);%which states the shooting nodes refer to, in this case, A
s0 = yref; %initial guess for the shooting nodes is the target final state

mysys = @(U) d1model.fullsim4controls(A0,B0,U,tf); %outputs [X,As,Bs,tout].
mycost = @(z) d1cost(z,yref,nsteps,mysys,MPCstep,d1model.dt,tf,udim); %F(z) (to be minmized)
mycons = @(z) d1constraints(z,xi,ind,nsteps,mysys,MPCstep,d1model.dt,tf,udim);
%G(z)  (constraint G = 0)

zsize = udim*nsteps+sdim;
Aeq = zeros(zsize);
```

```
beq = zeros(zsize,1);
A = []; b = [];
lbu = -1*ones(udim*nsteps,1); ubu = -lbu;
lbx = zeros(sdim,1); ubx = inf*ones(sdim,1);
%lb  = [lbu ;lbx]; ub = [ubu ;ubx];
lb  = []; ub = [];

u0 = 0*randn(udim,nsteps);
z0 = [u0(:)' s0'];
%-------------------------------------------------------------------------------------------------------




%------------------TRIAL RUN-------------------------------------------------------------
name = '17'; %12 on are same settings different shapes
need = (matlabpool('size') == 0);
if need
    matlabpool open 4
end
options = optimset('Display','iter','UseParallel','always');
tic;
z = fmincon(mycost,z0,A,b,Aeq,beq,lb,ub,mycons,options);
toc
%-------------------------------------------------------------------------------------------------------




%------------------PLOTTING-------------------------------------------------------------
uopt = vec2mat(z(1:udim*nsteps),udim)'; %pull  u from z

[Uopt,~] = zohsamp(uopt,nsteps,MPCstep,d1model.dt,tsim); %get the ZOH version of u
[U0,~] = zohsamp(u0,nsteps,MPCstep,d1model.dt,tsim); %get the ZOH version of u

[~,Aopt,Bopt,~] = mysys(Uopt);
yrefpattern = step_pattern(yref);
Apattern = step_pattern(Aopt(:,end));
```

```
figure(3); clf;
title('Result')
subplot(3,1,1); plot(d1model.nodes,yref);
legend('Reference');
subplot(3,1,2); plot(d1model.nodes,Aopt(:,end),'r');
legend('Result');
subplot(3,1,3); plot(d1model.nodes,yrefpattern);
hold on; plot(d1model.nodes,Apattern,'r');
legend('Reference','Result');

figure(5);clf;
subplot(2,1,1);
title('Control');
plot(tsim,Uopt);
subplot(2,1,2);
title('Original Guess');
plot(tsim,U0);

cd Images

h=figure(3);
figure(3);
saveas(h,[name '_Result'],'fig'); %name is a string

h=figure(5);
figure(5);
saveas(h,[name '_Control'],'fig'); %name is a string

cd ..

% d1model.viewsim(Aopt,Bopt);
%
%
% yref = Ass + 0.0005*(humps(linspace(0,1,25))'-50);%desired terminal output state
% yref = Ass -(0.01*(d1model.nodes-d1model.nodes(round(end/2))).^2 - 0.001)'; %up hump
% yref = (0.01*(d1model.nodes-d1model.nodes(round(end/2))).^2 + Ass - 0.001)'; %down hump
%
%
```

```
% figure(6)
% plot(d1model.nodes,yref)
```

6.2 d1constraints

Calculates the difference between the optimization variable s and $A(x, t_f)$, therefore returning the deviance from the continuity constraint in (22). Calls `zohsamp()` and `garfinkelFEmodel.fullsim()`.

```
function [c,ceq] = d1constraints(z,xi,ind,nsteps,sys,MPCstep,dt,tf,udim)

tsim = 0:dt:tf;

u = vec2mat(z(1:udim*nsteps),udim)'; %pull u from z
s = z(udim*nsteps+1:end); %pull s from z

[U,~] = zohsamp(u,nsteps,MPCstep,dt,tsim);

[x,~,~,~] = sys(U); %get the system trajectory based on u

ceq = zeros(size(s));
for j = 1:length(s)
    ceq(j) = s(j) - x(ind(j),xi(j)+1); %continuity constraint,
    %xi(j)+1 because xi(j) = 0 should go to x(1)
end
c = -1;
```

6.3 d1cost

Calculates the cost of the trajectory and control based on (22). Calls `zohsamp()` and `garfinkelFEmodel.fullsim()`.

```
function J = d1cost(z,yref,nsteps,sys,MPCstep,dt,tf,udim)

tsim = 0:dt:tf;
u = vec2mat(z(1:udim*nsteps),udim)'; %pull u from z
```

```
[U,sampledindices] = zohsamp(u,nsteps,MPCstep,dt,tsim); %get ZOH of u to apply to system

[~,As,~,~] = sys(U); %get the system trajectory based on u
y = As; %can change this later with some function y = output(x)

ysamp = y(:,sampledindices); %sample from output at MPCstep rate

% ypattern = step_pattern(ysamp);           %H(A) PATTERN COST
% yrefpattern = step_pattern(yref);
ypattern = ysamp;                           %COST BASED ON CONTUNIUM
yrefpattern = yref;

R = 1*eye(udim);
Q = 0*eye(length(ysamp(:,1)));
F = 1*eye(size(Q));

J = 0; %initialize cost

for t = 1:nsteps
    dy = (ypattern(:,t)-yrefpattern);
    J = J + dy'*Q*dy + u(:,t)'*R*u(:,t);
end

J = J + (10^3)*dy'*F*dy; %terminal state cost
```

6.4 garfinkelFEmodel

Finite element object containing `garfinkelFEmodel.fullsim()` which executes a simulation of the system defined by the properties of the object.

```
classdef garfinkelFEmodel < handle
    properties
        params
        dt
        nodes
    end
```

```
properties (SetAccess = private)
    NN % X+ = LL*X + NN*G(X) - NN*U+
    LL
    nel %number of elements
    LM %degree of freedom connectivity
    h %element lengths
end

methods
    function [] = setup(obj)

        rho = obj.params(5); D = obj.params(6); %diffusion ratio, global diffusion

        numnode = length(obj.nodes); %number of nodes
        obj.nel = numnode -1; %number of elements
        obj.h = obj.nodes(2:end) - obj.nodes(1:end-1);

        EC = ones(2,obj.nel-1); %initialize connectivity matrix
        for el = 1:1:obj.nel %generate connectivities of elements
            EC(:,el) = [el el+1];
        end

        ID = ones(2,numnode); %initialize ID
        for node=1:numnode
            ID(:,node)=[2*node-1,2*node];
            %fill in ID AKA name the degrees of freedom: X = [A1 B1 A2 B2 A3 B3 A4 B4...]'
        end
        obj.LM = ones(4,obj.nel); %initialize LM
        for el=1:obj.nel
            nodeNames = EC(:,el); %find the nodes of the element and get their numbers
            obj.LM(:,el)=[ID(:,nodeNames(1));ID(:,nodeNames(2))]; %fill in LM
        end

        % Build Element Stiffness and Mass Matricies
        k = ones(4,4,obj.nel); %initialize stiffness matricies
        m = k;  %initialize mass matrix

        for elNum=1:obj.nel
            m(:,:,elNum) = obj.h(elNum)*(1/6)*[2 0 1 0; 0 2 0 1; 1 0 2 0; 0 1 0 2];
```

```
        %elemental masses
        k(:,:,elNum) = (1/obj.h(elNum))*D*[rho 0 -rho 0; 0 1 0 -1; ...
                -rho 0 rho 0; 0 -1 0 1];
        %elemental stiffnesses
    end

    K = zeros(2*numnode); %initialize K
    M = K; %initialize M
    for elNum=1:obj.nel %global matrix assembly
        for row=1:4
            for col=1:4
                K(obj.LM(row,elNum),obj.LM(col,elNum)) = ...
                        K(obj.LM(row,elNum),obj.LM(col,elNum)) + k(row,col,elNum);
                M(obj.LM(row,elNum),obj.LM(col,elNum)) = ...
                        M(obj.LM(row,elNum),obj.LM(col,elNum)) + m(row,col,elNum);
            end
        end
    end

    %time stepping matrices
    obj.NN = (M/obj.dt + K)\eye(size(K)); %save some computation in the loop
    obj.LL = obj.NN*M/obj.dt';
end %sets up the finite element matricies

function [Xnext] = xstep(obj,X,u) %this is just an optimized version of step()
    G = zeros(2*(obj.nel+1),1); %initialize G
    U = G;
    for elNum = 1:obj.nel %find G(X(t)) reaction term
        G(obj.LM(:,elNum)) = G(obj.LM(:,elNum)) ...
            + obj.h(elNum)*gs(X(obj.LM(:,elNum)),obj.params);
    end
    U(1:2) = obj.params(6)*[obj.params(5)*u(1);u(2)];
    %interleave UA and UB to form U. UA is multipleid by rho, whole thing by D
    Xnext = obj.LL*X + obj.NN*(G - U); %diffusion + reaction + control
    Xnext(isnan(Xnext)) = 0; Xnext(Xnext<0) = 0; %limit nonsense
end
function [X,As,Bs,tout] = fullsim(obj,A0,B0,U,tf) %full sim through tf. %U = [UA ; UB]
    tout = 0:obj.dt:tf;
    X = zeros(2*length(A0),length(tout)); %initialize state history
```

```
            X0 = [A0(:)';B0(:)']; X(:,1) = X0(:);
            %interleave A,B to form X = [A1 B1 A2 B2 A3 B3 A4 B4...]'
            for t = 1:(length(tout)-1) %the number of timesteps
                X(:,t+1) = xstep(obj,X(:,t),U(:,t)); %U = [UA ; UB]
            end
            As = X(1:2:end,:); Bs = X(2:2:end,:);
        end

        function [] = viewsim(obj,As,Bs)
            figure(1), clf; figure(2), clf;
            for frame = 1:length(As(1,:))
                figure(1); plot(obj.nodes,As(:,frame));
                figure(2); plot(obj.nodes,Bs(:,frame));
            end
        end
    end
end
```

6.5 gs (1D)

Evaluates the nonlinear term $G(X)$. Called by `garfinkelFEmodel.fullsim()`.

```
function [g] = gs(X,params)

A1 = X(1); B1 = X(2); A2 = X(3); B2 = X(4); %unpack X
gamma = params(1); c = params(2); kp = params(3); e = params(4); %unpack params

%calculate nonlinear terms in the equation for B
GB1 = (gamma*(3*A1^2 + 2*A1*A2 + A2^2 - 4*B1*e - 2*B2*e))/12;
GB2 = (gamma*(A1^2 + 2*A1*A2 + 3*A2^2 - 2*B1*e - 4*B2*e))/12;

%set up variables
dx = 0.25;
x = 0:dx:1;
L1 = 1-x; L2 = x;
A = A1*L1 + A2*L2; B = B1*L1 + B2*L2;
%calculate the integrands for the nonlinear terms in A
```

```
GA1i = L1.*( A.^2./( (1 + kp*A.^2).*B ) - c.*A );
GA2i = L2.*( A.^2./( (1 + kp*A.^2).*B ) - c.*A );

%limit nonsense
GA1i(isnan(GA1i)) = eps; GA2i(isnan(GA2i)) = eps; % 0/0 = 0

%calculate nonlinear terms in the equation for A
GA1 = gamma*trapz(x,GA1i);
GA2 = gamma*trapz(x,GA2i);

%put them all in a vector in the proper order
g = [GA1; GB1; GA2; GB2];
```

6.6 zohsamp

Takes a zero-order hold of the control for application to the system.

```
function [U,sampledindices] = zohsamp(u,nsteps,MPCstep,dt,tsim)

sampledindices = zeros(1,nsteps);
for step = 1:nsteps
    sampledindices(step) = round(step*MPCstep/dt);
    %find the sampled time steps in y. round not floor because MPCstep = n*dt
end

udim = length(u(:,1));

U = zeros(udim,length(tsim)); %implement a zero order hold of U over tsim
counter = length(u);
controltimes = fliplr(sampledindices);
controltimes(1) = length(tsim);
for i = controltimes
    for row = 1:i
        U(:,row) = u(:,counter);
    end
    counter = counter - 1;
end
```

6.7 2dopt

Executes a single run of the optimization for the two-dimensional distributedly controlled system with the specified parameters. Sets up a finite element model object `garfinkelFE2D` and defines anonymous functions based on `d2constraints()`, `d2cost()`, and `garfinkelFE.fullsim()` which are called by `fmincon()`. Displays results.

```
% Parameter Setup
clear all
clc

Ass = 1.110305545592924; Bss = 61.638920228753065;
%A_t = A_xx = B_t = B_xx = 0. Does not depend on gamma or rho

L = 10; %domain size/2
l = 0.1*2*L; %characteristic mesh size
rho = 0.01/2; c = 0.01; kp = 0.65; e = 0.02; %model params
D = 1; gamma = 200;
simdt = 0.2;


fd = @(p) drectangle(p,-L,L,-L,L);
[p,t]=distmesh2d(fd,@huniform,l,[-L,-L;L,L],[-L,-L;-L,L;L,-L;L,L]);
d2model = garfinkelFE2D;
d2model.params = [gamma,c,kp,e,rho,D];
d2model.dt = simdt;
d2model.nodes = p;
d2model.els = t;
d2model.setup


%-----------------------------INITIAL CONDITIONS-----------------------------------------
A0 = Ass*(1+0*randn(length(d2model.nodes),1)); %steady states
B0 = Bss*(1+0*randn(length(d2model.nodes),1));
%----------------------------------------------------------------------------------------
```

```
%----------------------------TIME PARAMETERS-------------------------------------------------
tf = 5; %final time
tsim = 0:d2model.dt:tf; %simulation time vector
MPCstep = d2model.dt*2; %frequency of control application
nsteps = length(0:MPCstep:tf)-1; %how many dt's we have
%--------------------------------------------------------------------------------------------


%----------------------------REFERENCE SIGNAL------------------------------------------------
yref = Ass -0.01 + mvnpdf(d2model.nodes,[0,0],[20 0; 0 5]); %middle dot
%yref = Ass -0.01 + 4*mvnpdf(d2model.nodes,[0,0],[100 0; 0 5]); %horizontal stripe
%--------------------------------------------------------------------------------------------



%----------------------------CONTROL POINTS--------------------------------------------------
% cpoints = [0  5*(tand(6) - tand(30)); -5 -5*tand(30); ... triangle
%            0  0; 5 -5*tand(30)]; %this has to be [x y; x y; ...]
% cpoints = 5*[1 1; -1 1; 1 -1; -1 -1]; %square
cpoints = [0 0]; %center point
npts = length(cpoints(:,1)); %number of target points to control
myones = ones(length(d2model.nodes(:,1)),1);
UvecID = zeros(1,npts*2);

for pt = 1:npts
    dnodes = d2model.nodes - myones*cpoints(pt,:); % node coords - point
    distances = dnodes(:,1).^2 + dnodes(:,2).^2; % dx^2 + dy^2
    [~,cnode] = sort(distances);
    UvecID((2*pt-1):(2*pt)) = [2*cnode(1)-1 2*cnode(1)];
    %find the state vector indices for that node
end
udim = 2*npts; %each point has a B and an A control
%--------------------------------------------------------------------------------------------



%----------------------------OPTIMIZATION SETUP----------------------------------------------
costtype = 'continuous'; %pattern or continuous cost grading for state
```

```
sdim = length(A0); %number of shooting nodes: I only care about what A looks like

xi = nsteps*ones(sdim,1); %times of shooting nodes (in dt's): final time
ind = 1:2:length([A0(:);B0(:)]);%which states the shooting nodes refer to, in this case, A

mysys = @(U) d2model.fullsim(A0,B0,U,UvecID,tf); %outputs [X,As,Bs,tout].
mycost = @(z) d2cost(z,yref,nsteps,mysys,MPCstep,d2model.dt,tf,udim,costtype,Ass);
%F(z) (to be minmized)
mycons = @(z) d2constraints(z,xi,ind,nsteps,mysys,MPCstep,d2model.dt,tf,udim);
%G(z)  (constraint G = 0)

s0 = yref; %initial guess for the shooting nodes is the target final state
u0 = 0*randn(udim,nsteps); %initial guess for the controls
z0 = [u0(:) ; s0(:)]; %initial optimization variable
%-------------------------------------------------------------------------------------------------------




%-----------------TRIAL RUN-----------------------------------------------------------------
name = 'cmaes1';
algo = 'cmaes';

need = (matlabpool('size') == 0);
if need
    matlabpool open 4
end
switch algo
    case 'cmaes'
        tic;
        z = mycmaes(mycost,u0(:),0.5);
        toc
    case 'fmincon'
        options = optimset('Display','iter','UseParallel','always');
        tic;
        z = fmincon(mycost,z0,[],[],[],[],[],[],mycons,options);
        toc
end
```

```
%-------------------------------------------------------------------------------------------



%------------------PLOTTING-----------------------------------------------------------------
uopt = vec2mat(z(1:udim*nsteps),udim)'; %pull  u from z

[Uopt,~] = zohsamp(uopt,nsteps,MPCstep,d2model.dt,tsim); %get the ZOH version of u
[U0,~] = zohsamp(u0,nsteps,MPCstep,d2model.dt,tsim); %get the ZOH version of u

[~,Aopt,Bopt,~] = mysys(Uopt);

figure(3); clf;
title('Result')

subplot(2,2,1); fig = gcf();
view2Dplot(d2model.nodes,yref,fig);
title('Reference')
subplot(2,2,2); fig = gcf();
view2Dplot(d2model.nodes,Aopt(:,end),fig);
title('Result')

subplot(2,2,3); fig = gcf();
view2Dpatt(d2model.nodes,yref,fig,Ass+0.005);
subplot(2,2,4); fig = gcf();
view2Dpatt(d2model.nodes,Aopt(:,end),fig,Ass+0.005);



figure(5);clf;
title('Control');
subplot(2,1,1);
plot(tsim,Uopt(1:2:end,:));
leftlabel = repmat('at (',npts,1);
rightlabel = repmat(')',npts,1);
legend([leftlabel num2str(cpoints) rightlabel]); title('A Controls');
subplot(2,1,2);
plot(tsim,Uopt(2:2:end,:));
legend([leftlabel num2str(cpoints) rightlabel]); title('B Controls');
```

```
figure(6); clf
title('Control Application Points');
plot(cpoints(:,1),cpoints(:,2),'ko','linewidth',3);
axis([-L L -L L]); axis square;

cd Images

h=figure(3);
figure(3);
saveas(h,[name '_Result'],'fig'); %name is a string

h=figure(5);
figure(5);
saveas(h,[name '_Control'],'fig'); %name is a string

h=figure(6);
figure(6);
saveas(h,[name '_Control_Points'],'fig'); %name is a string

cd ..

% d2model.viewsim(Aopt,Bopt);
```

6.8 d2constraints

Calculates the difference between the optimization variable s and $A(x, t_f)$, therefore returning the deviance from the continuity constraint in (21). Calls `zohsamp()` and `garfinkelFE2D.fullsim()`.

```
function [c,ceq] = d2constraints(z,xi,ind,nsteps,sys,MPCstep,dt,tf,udim)

tsim = 0:dt:tf;

u = vec2mat(z(1:udim*nsteps),udim)'; %pull u from z
s = z(udim*nsteps+1:end); %pull s from z

[U,~] = zohsamp(u,nsteps,MPCstep,dt,tsim);
```

```
[x,~,~,~] = sys(U); %get the system trajectory based on u

ceq = zeros(size(s));
for j = 1:length(s)
    ceq(j) = s(j) - x(ind(j),xi(j)+1); %continuity constraint,
    %xi(j)+1 because xi(j) = 0 should go to x(1)
end
c = -1;
```

6.9 d2cost

Calculates the cost of the trajectory and control based on (21). Calls `zohsamp()` and `garfinkelFE2D.fullsim()`.

```
function J = d2cost(z,yref,nsteps,sys,MPCstep,dt,tf,udim,type,cutoff)

tsim = 0:dt:tf;
u = vec2mat(z(1:udim*nsteps),udim)'; %pull u from z

[U,sampledindices] = zohsamp(u,nsteps,MPCstep,dt,tsim); %get ZOH of u to apply to system

[~,As,~,~] = sys(U); %get the system trajectory based on u
y = As; %can change this later with some function y = output(x)

ysamp = y(:,sampledindices); %sample from output at MPCstep rate

switch(type)
    case 'pattern'
        ypattern = step_pattern(ysamp,cutoff);          %H(A) PATTERN COST
        yrefpattern = step_pattern(yref,cutoff);
    case 'continuous'
        ypattern = ysamp;                           %COST BASED ON CONTUNIUM
        yrefpattern = yref;
end

R = 1*eye(udim);
```

```
Q = 0*eye(length(ysamp(:,1)));
F = 1*eye(size(Q));

J = 0; %initialize cost

for t = 1:nsteps
    dy = (ypattern(:,t)-yrefpattern);
    J = J + dy'*Q*dy + u(:,t)'*R*u(:,t);
end

J = J + (10^5)*dy'*F*dy; %terminal state cost
```

6.10 garfinkelFE2D

Finite element object containing `garfinkelFE2D.fullsim()` which executes a simulation of the system defined by the properties of the object.

```
classdef garfinkelFE2D < handle
    properties
        params %list of parameters
        dt %timestep
        nodes %Nx2 list of coordinates of each node. nodes(n,:) = [nx,ny]
        els %(aka EC) nelx3 list of nodes in each triangular element. els(elem,:) = [n1 n2 n3]
    end
    properties (SetAccess = private)
        areas %(det(J))
        NN
        LL
        nel
        LM
        Jins %inverse Jacobians of each element transformation to master element
    end

    methods
        function [] = setup(obj)

            rho = obj.params(5); D = obj.params(6); %diffusion ratio, global diffusion
```

```
numnode = length(obj.nodes); %this will return the long dimension
obj.nel = length(obj.els);
obj.areas = zeros(obj.nel,1); %a list of areas of each element. Ae = areas(e)
obj.Jins = cell(obj.nel,1); %initialize same as areas
for el = 1:obj.nel
    coords = obj.nodes(obj.els(el,:),:); %[x1 y1;x2 y2;x3 y3]
    J = [coords(1,1) - coords(3,1), coords(1,2) - coords(3,2); ...
              coords(2,1) - coords(3,1), coords(2,2) - coords(3,2)];
              %jacobian: J = [x1 - x3, y1 - y3; x2 - x3, y2 - y3];
    obj.areas(el) = det(J)/2; %area of the triangles
    obj.Jins{el} = J\eye(2);
end

ID = ones(2,numnode); %initialize ID
for node=1:numnode
    ID(:,node)=[2*node-1,2*node];
    %fill in ID AKA name the degrees of freedom: X = [A1 B1 A2 B2 A3 B3 A4 B4...]'
end
obj.LM = ones(6,obj.nel); %initialize LM
for el=1:obj.nel
    obj.LM(:,el)=[ID(:,obj.els(el,1));ID(:,obj.els(el,2));ID(:,obj.els(el,3))];
    %fill in LM
end

k = cell(obj.nel,1); %initialize elemental stiffness matrix
m = k; %initialize elemental mass matrix
for el = 1:obj.nel
    dL1 = obj.Jins{el}*[1;0]; dL2 = obj.Jins{el}*[0;1]; dL3 = obj.Jins{el}*[-1;-1];
    dLs = [dL1 dL2 dL3]; ke = zeros(6); %all dLi/dx and /dy. Initialize ki
    for i = 1:3
        for j = 1:3
            value = dLs(1,i)*dLs(1,j) + dLs(2,i)*dLs(2,j);
            ke(2*i-1:2*i,2*j-1:2*j) = [rho*value 0; 0 value]; %build elemental k
        end
    end
    k{el} = D*2*obj.areas(el)*ke;
    %Since ki(x,y) is a constant, the integration is just multiplication by area
    m{el} = 2*obj.areas(el)*(1/24)*[2 0 1 0 1 0; 0 2 0 1 0 1; 1 0 2 0 1 0; ...
            0 1 0 2 0 1; 1 0 1 0 2 0; 0 1 0 1 0 2];
```

```
    end

    K = zeros(2*numnode); %initialize K
    M = K; %initialize M
    for el=1:obj.nel
        for row=1:6
            for col=1:6
                ke = k{el}; me = m{el};
                K(obj.LM(row,el),obj.LM(col,el)) =
                        K(obj.LM(row,el),obj.LM(col,el)) + ke(row,col);
                M(obj.LM(row,el),obj.LM(col,el)) =
                        M(obj.LM(row,el),obj.LM(col,el)) + me(row,col);
            end
        end
    end

    obj.NN = (M/obj.dt + K)\eye(size(K)); %save some computation in the loop
    obj.LL = obj.NN*M/obj.dt';
end

function [Anext,Bnext] = step(obj,A,B,uA,uB)
%steps one timestep forward. Written to accomodate distributed control
    X = [A(:)';B(:)']; X = X(:); %interleave A,B to form X = [A1 B1 A2 B2 A3 B3 A4 B4...]'
    G = zeros(size(X)); %initialize G
    for elNum = 1:obj.nel %find G(X(t)) reaction term
        G(obj.LM(:,elNum)) = G(obj.LM(:,elNum)) ...
            + obj.h(elNum)*gs(X(obj.LM(:,elNum)),obj.params,obj.areas(elNum));
    end
    U = [uA(:)';uB(:)']; U = U(:);
    %interleave UA and UB to form U. UA is multipleid by rho, whole thing by D
    Xnext = obj.LL*X + obj.NN*(G+U); %diffusion + reaction + control
    Anext = Xnext(1:2:end); Bnext = Xnext(2:2:end);
end

function [X,As,Bs,tout] = fullsim(obj,A0,B0,U,UvecID,tf)
%full sim through tf. UA, UB are vectors of size tout.
    tout = 0:obj.dt:tf;
    X = zeros(2*length(A0),length(tout)); %initialize state history
    Up = X; % initialize all applied controls at all times as 0
```

```
    X0 = [A0(:)';B0(:)']; X(:,1) = X0(:);
    %interleave A,B to form X = [A1 B1 A2 B2 A3 B3 A4 B4...]'
    for i = 1:length(UvecID) %Bulid Up: all zeros except the rows which are controlled
        Up(UvecID(i),:) = U(i,:);
    end
    for t = 1:round(tf/obj.dt) %the number of timesteps
        G = zeros(size(X(:,t))); %initialize G
        U = G;
        Upt = Up(:,t); %Fetch the applied controls at time t
        for el = 1:obj.nel %find G(X(t)) reaction term and U control
            G(obj.LM(:,el)) = G(obj.LM(:,el)) ... %calculate G
                + gs(X(obj.LM(:,el)),obj.params,obj.areas(el));
            U(obj.LM(:,el)) = U(obj.LM(:,el)) ...
                + us(Upt(obj.LM(:,el)),obj.areas(el));
                %find how each node feels the applied U field
        end
        Xnext = obj.LL*X(:,t) + obj.NN*(G+U); %diffusion + reaction + control
        Xnext(isnan(Xnext)) = 0; Xnext(Xnext<0) = 0; %limit nonsense
        X(:,t+1) = Xnext;
    end
    As = X(1:2:end,:); Bs = X(2:2:end,:);
end

function [] = viewsim(obj,As,Bs,Ass,Bss)
    figure(1), clf; figure(2), clf;
    x = obj.nodes(:,1); y = obj.nodes(:,2);

    xmax = max(x); xmin = min(x); ymax = max(y); ymin = min(y);
    dx = (xmax - xmin)/100; dy = (ymax - ymin)/100;

    [X,Y] = meshgrid(xmin:dx:xmax,ymin:dy:ymax);

    for frame = 1:length(As(1,:))

        Ag = griddata(x,y,As(:,frame),X,Y);
        figure(1); contourf(X,Y,Ag,[Ass Ass]);
        title({['max - min = ' num2str(max(As(:,frame)) - min(As(:,frame)))], ...
            ['mean value:' num2str(mean(As(:,frame)))]});
```

```
                Bg = griddata(x,y,Bs(:,frame),X,Y);
                figure(2); contourf(X,Y,Bg,[Bss Bss]);
                title({['max - min:' num2str(max(Bs(:,frame)) - min(Bs(:,frame)))], ...
                    ['mean value = ' num2str(mean(Bs(:,frame)))]});
            end
        end
    end
end
```

6.11 gs (2D)

b Evaluates the nonlinear term $G(X)$. Called by `garfinkelFE2D.fullsim()`.

```
function [g] = gs(X,params,area)
    A1 = X(1); B1 = X(2); A2 = X(3); B2 = X(4); A3 = X(5); B3 = X(6); %unpack X
    gamma = params(1); c = params(2); kp = params(3); e = params(4); %unpack params

    x = [0.5 ; 0 ; 0.5]; y = [0 ; 0.5 ; 0.5]; %quadrature points

    L1 = x;
    L2 = y;
    L3 = 1 - x - y;

    A = A1*L1 + A2*L2 + A3*L3;
    B = B1*L1 + B2*L2 + B3*L3;

    gA = gamma*(A.^2./( (1 + kp*A.^2).*B ) - c.*A);
    gB = gamma*(A.^2 - e*B);

    g = 2*area*1/3*[L1'*gA ; L1'*gB ; L2'*gA ; L2'*gB ; L3'*gA ; L3'*gB];
     %numerical integration
end
```

Bibliography

[1] A. Garfinkel, Y. Tintut, D. Petrasek, K. Bostrom, and L. L. Demer, "Pattern formation by vascular mesenchymal cells," *PNAS*, 2004.

[2] A. Turing, "The chemical basis of morphogenesis," *Philosophical Transactions of the Royal Society of London*, 1952.

[3] Camacho and Bordons, *Model Predictive Control*. Springer Verlag, 2004.

[4] S. Dubljevic, P. Mhaskar, N. H. El-Farra, and P. D. Christofides, "Predictive control of diffusion-reaction processes," *American Control Conference, 2005. Proceedings of the 2005*, 2005.

[5] S. Dubljevic and P. D. Christofides, "Predictive output feedback control of parabolic partial differential equations (pdes)," *Industrial and engineering chemistry research*, 2006.

[6] S. Dubljevic, P. D. Christofides, and I. G. Kevrekidis, "Distributed nonlinear control of diffusion-reaction processes," *International journal of robust and nonlinear control*, 2004.

[7] MATLAB, *version R2012b*. Natick, Massachusetts: The MathWorks Inc., 2012.

[8] P. Persson and G. Strang, "A simple mesh generator in matlab," *SIAM Review*, 2004.

[9] Grune and Pannek, *Nonlinear Model Predictive Control Theory and Algorithms*. Springer, 2011.

[10] Stoer and Bulirsch, *Introducation to Numerical Analysis*. Springer Verlag, 1980.

[11] P. M. M. Fliess, J. L. Lvine and P. Rouchon, "Flatness and defect of non-linear systems: introductory theory and examples," *International Journal of Control*, 1995.

[12] N. Hansen, *The CMA Evolution Strategy: A Comparing Review*. Springer, 2006.