Team Control Number

For office use only
T1 _____
T2 _____
T3 _____
T4 _____

51626

Problem Chosen

A

For office use only
F1 _____
F2 _____
F3 _____
F4 _____

2016
MCM/ICM
Summary Sheet

(Your team's summary should be included as the first page of your electronic submission.)

Type a summary of your results on this page. Do not include the name of your school, advisor, or team members on this page.

Water in a bathtub is an immensely complex system, even when considering standing water, or waves in water. The system becomes more complicated with the introduction of water from a faucet and possible human motion in a bathtub. But when taking a bath, there is little consideration for the minute changes in temperature throughout the tub. Rather, one cares about achieving a *sense* of evenness with as little effort as possible. Thus, we approach the problem with the methods of control theory.

In reality, the most optimal way to keep the temperature even in a bathtub without wasting water is likely by constantly moving and slowly trickling warm water from the faucet into the tub. However, this is an unrealistic way to take a bath. We focus less on the physical behavior of the water, as much of it can be modeled by a Finite Element Simulator, and focus on clean system design. Since the behavior of the water is continuous and the actions that a bather can perform are discrete, we build a *hybrid dynamical system* to model the interaction. Hybrid systems naturally separate the continuous from the discrete, so any model of the bath water can be interfaced with the discrete part of the hybrid dynamical system without having to change it. This allows for greater flexibility, as we can vary the complexity of the model of bath water depending on computational resources. With this hybrid description, we can use hybrid system optimization tools to find the optimum strategy that stays as close to a typical bath as possible.

Optimal Bathing Strategies by Hybrid Controls

Team #51626

February 1, 2016

Abstract

In this report, we address the problem of having the perfect bathing experience. In particular, we present a complete mathematical formulation of the problem as a hybrid dynamical system as defined in the flow-jump framework. We give an explicit cost function to optimize and use hybrid system optimization tool to solve a simplified version of the problem as a demonstration. We give possible boundary conditions for the use of a Finite Element Simulator to model the behavior of the bath water. We conclude with some remarks about the benefits and pitfalls to a data-driven controls method as opposed to an analytic physic-driven method.

1 Introduction

Consider the following anecdote: an physicist draws a bath and lets the water in the tub stand for several minutes. The air will cool the surface of the water faster than the sides of the tub will, so as she enters the tub, she mixes the water with her hands to make the temperature more consistent throughout. She then lays in the tub and begins to relax. The air continues to cool the surface of the water, so she draws more hot water from the faucet and moves the water around to make the temperture even throughout again. After doing this several times, she realizes she might be able to model the heat diffusion of the water as well as the fluid dynamics of the water to find the optimal combination of motion and water input to make the temperature of the water consistent throughout the tub and waste as little water as possible.

She takes this problem to her friend studying computer science, who notes that it is an interesting problem, but no longer captures the original intention of the problem. Her friend does not know much physics, but she believes that she would not want to sit in a tub in constant motion. Instead she argues that some solution to the problem must necessarily have an optimization aspect that adds costs to unnatrual motion.

The above anecdote represents the motivation of our approach to this problem. There are many physical phenomenon that affect the temperature gradient of the water in a tub, especially with respect to the possible motions of the bather. So rather than focusing on the different possible positions and motions necessary, we focus on fixing a collection of motions and actions typical in a tub, and find the best strategy that fits the situation of a relaxing bath.

Throughout control theory, there are examples in which modeling the controls of a system carefully leads to nontrivial optimizations. The goal of our model is to see if there are any such optimizations in the case of having a relaxing bath. There are several *dueling* objectives, as the only way to heat the water is to waste more water and spend more time not relaxed. Ideally, even with extremely simplified physics, it is possible to see some nontrival decisions that a controller might make after optimization.

The structure of this report is as follows: We begin with standard definition related to hybrid control systems. Many of these definitions are found in [**?**] and are given here for completeness. We then give a complete formalization of the problem as a hybrid system optimization problem. With this formalization it is possible to apply existing algorithms [**?**]. We will then give a short section on using Finite Element Simulator for simulating water thermodynamics and how it interfaces with the hybrid system model. This will be followed by a simple demonstration of the model using a hybrid system toolbox in MATLAB [**?**].

2 Definitions

We will use the following mathematical formulation of this problem: let S denote some strategy that formally describes the behavior of a bather for H time. Let C_i denote a cost function that takes a strategy S and some vector of parameters $\overline{p}$. Our problem is to give an optimal strategy S^* that minimizes

$$\sum_{i=1}^{n-1} w_i C_i(S^*, \overline{p}_i)$$

for some fixed $\overline{w} \in \mathbb{R}^n$ and some vector of cost functions $\overline{C}$.

For example, let $u_S : \mathbb{R}^3 \times [0, H] \to (0, \infty)$ denote the temperature of the water in the tub with respect to space and time as determined by the strategy S. We will assume (and it will be

the case in our formalization) that u_S is uniquely determined. Then one cost function is

$$C(S,T) = \int_0^H \int_x \int_y \int_z u_S(x,y,z,t) - T$$

where T could be some nominal temperature. The parameters have very weak type constraints; for example, we could consider the cost function

$$C'(S,T) = \int_0^H \int_x \int_y \int_z u_S(x,y,z,t) - T(t)$$

where T is a function of temperature with respect to time.

Since the dynamics of water temperature are continuous and strategy is typically discrete, it is natural to model this problem as a hybrid dynamical system. What follows is the flow-jump framework for hybrid dynamical systems.

Definition 1. *Let the state space $\mathcal{S} \subseteq \mathbb{R}^n$ and the input space $\mathcal{I} \subseteq \mathbb{R}^m$. Let $C, D \subseteq \mathcal{S} \times \mathcal{I}$ and let $f : C \to C$ and $g : D \to \mathcal{S}$ be functions. A **Hybrid Dynamical System** is a quadruple $\mathcal{H} = \langle f, C, g, D \rangle$.*

In the above definition, f is the **flow map**, C is the **flow set**, g is the **jump map**, and D is the **jump set**. The flow set may be thought of as the domain of the continuous component of the hybrid system and the jump set may be thought of as the domain of the discrete component. The flow map defines the way the continuous component of the hybrid system evolves and the jump maps defines how and when to change discrete states.

Definition 2. *A **hybrid time domain** is a subset of $[0, H] \times \{0, 1, \ldots, H-1\}$ of the form*

$$\bigcup [t_i, t_{i+1}] \times \{i\}$$

for any $\{t_i\}$ such that $0 = t_0 \leq t_1 \cdots \leq t_n \leq H$.

Let $\dot{x}$ denote the derivative of x and let x^+ denote the next state after a change in states.

Definition 3. *A **trajectory**, or solution, of a hybrid dynamical system is a function*

$$x : dom(x) \to \mathcal{S}$$

such that $dom(x)$ is a hybrid time domain, $\dot{x} = f(x,u)$, and $x^+ = g(x,u)$.

Using the language introduced that beginning of this section, the jumps that occure in a trajectory are a strategy, and we would like to find the sequence of jumps that minimize the cost functions.

3 Assumptions

We give an explicit reference of the assumptions we make about the details of the model. Many of these simplifications have justifications.

1 **The water level in the tub is never higher than the excess drain.**
This is to assume that the water level is fixed at the maximum capacity. By solving a collection of simple fluid dynamics equations (e.g. Bernoulli's Equation) it is not hard to tell that rate at which water enters the tub would have to be unreasonably fast in order for this assumption to break.

2 **No convection occurs within the water.**
If this were the case, then the water temperature would become even more rapidly. Thus, this assumption should not affect the optimization.

3 **We assume arbitrary convection coefficients.**
This is probably our strongest assumption. These values typically lie in a range, and solving for these coefficients would require further assumptions and more time than we have.

4 **Air acts as an infinite heat sink.**
This assumption is standard and natural in this setting, as it should be nearly true for simple calculations.

5 **The tub under consideration has feet.**
We consider all porselein sides to be surrounded in air. Though this may not be realistic, it drastically simplifies the model. Since we are primarily demonstrating a framework, the assumption may be dropped in exchange for more complicated simulations.

6 **Water temperature is lower bounded by a hard constraint.**
Water should approach some ambient temperature if it is left to stand indefinitely.

7 **When relaxed, the bather is sitting along the longest part of the tub and has her feet at the end where the faucet is.**

8 **The bath has a faucet that has control of flow rate and temperature simultaneously.**
There are many baths with this design. Most ball-and-socket joints so that one direction represents temperature and one direction represents flow rate.

In addition to simplifications, there are several variables we fix for the sake of demonstation. The maximum safe temperature for the water in the tub is 110°F and the maximum temperature of the water from the faucet is 125°F. The maximum rate of water from the faucet is 7 gal/ min. The lower bound of the temperature of the water in the tub is 80°F. As a baseline, we assume that the ideal temperature is 95°F.

4 Controls and Optimization

We consider *front* and *back* to be the only two positions in the tub. In the back position, the bather is laying back and stationary. In the front position, the bather is crouching in order to have access to the temperature and flow rate of the faucet. Only in the front position does the bather have access. This captures the intuition that it is necessary for a bather to lean forward in order to turn off or turn on the faucet. Also, the bather is allowed to leave the faucet on and move to the back, but must move to the front in order to change turn the faucet off again. We enforce that the bather must spend some nonzero amount of time at the front in order to ensure that they cannot change the faucet temps instantaneously.

The automaton in Figure 1 expresses this intuition of this system. Table 1 gives details for the naming of variables in the automaton. This representation has redundant information and also does not capture the entire state space. For example, H, C, I_H, and I_C do not represent states, but rather cues for changing the states. Likewise, the state f_{on} is continuum many states representing pairs of temperatures and flow rates. In practice, it is a complete graph with nm states, where n is the number of possible discretized temperatures and m is the number of possible discretized flow rates.

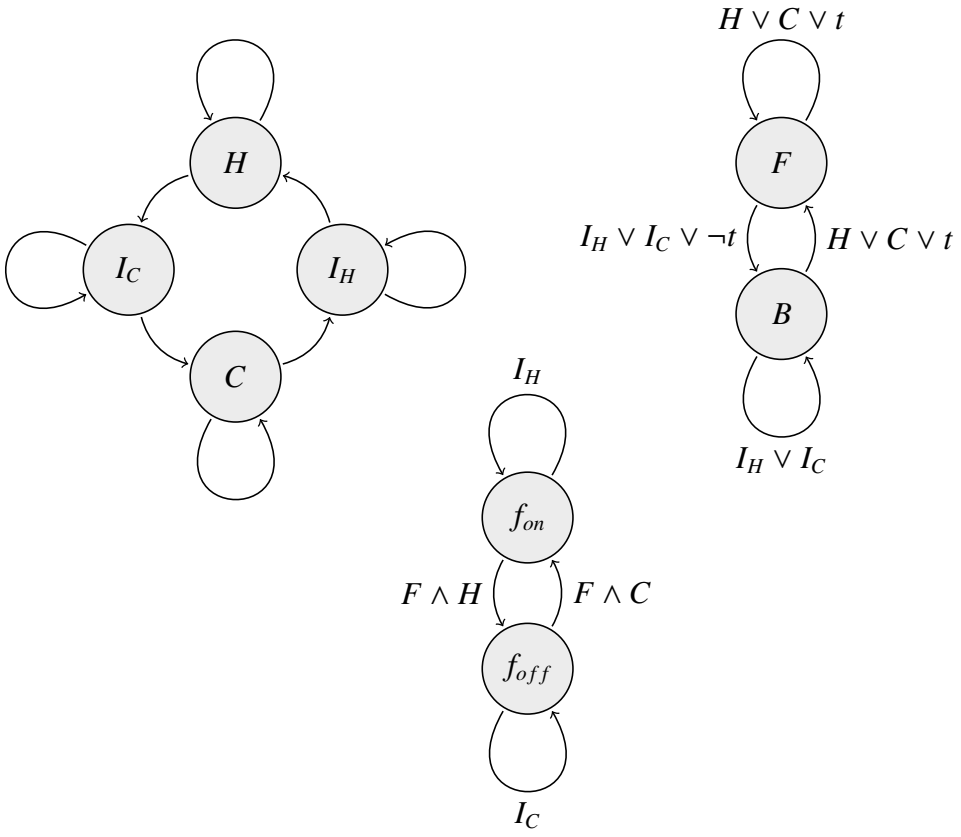

Figure 1: This automaton represents the intuitive state space of the problem. It omits many states and has redundent information, but gives a general outline of the possible control sequences.

I_C	state of intermediate cooling
I_H	state of intermediate heating
H	state in which it is necessary to start heating
C	state in which it is necessary to start cooling
f_{on}	faucet is on
f_{off}	faucet is off
F	bather is in the front position
B	bather is in the back position
t	timer for waiting in the front is running

Table 1: This table is a legend for Figure 1.

In the flow-jump model, this automaton may be though of as the discrete aspect of the the hybrid model, i.e. the automaton describes the jump function g and the logic variables are governed by the continuous dynamics of the water, which is described by f. The following is a complete formulation of the problem into this the flow-jump model.

Let the state space be

$$\begin{aligned}\mathcal{S} &= T_{bath} \times dT \times P \times T_{faucet} \times R_{faucet} \times W \times [0, w] \times [0, l] \times [0, h] \\ &= [T_{min}, T_{max}] \times \mathbb{R} \times \{0, 1\} \times [0, t_{max}] \times [0, f_{max}] \times [0, t_{cap}] \times [0, w] \times [0, l] \times [0, h] \\ &\subseteq \mathbb{R}^9\end{aligned}$$

where T_{bath} is the domain of possible temperatures of the water in the tub, dT is the domain of the derivative of the temperature of the water in the tub, P is the domain of possible positions in the tub, T_{faucet} is the domain of possible temperatures of the faucet water, R_{faucet} is the domain of possible flow rates of the faucet, and W is the domain of possible times that a timer can read. The timer domain W is included to ensure that the bather must stay put for at least t_{cap} time in the front of the tub, though she may stay longer. The last three dimensions are the space dimensions for the temperature. Let the input space be

$$\begin{aligned}\mathcal{I} &= P \times T_{faucet} \times R_{faucet} \times F_{IO} \\ &= \{0, 1\} \times [0, t_{max}] \times [0, f_{max}] \times \{0, 1\} \\ &\subseteq \mathbb{R}^4\end{aligned}$$

where F_{IO} is essentially the domain of a boolean that represents if the controller would like to change the faucet values. Let the flow set $C = \mathcal{S} \times \mathcal{I}$. Let the jump set be

$$D = \{i \times s \in C : (i_0 \wedge \neg s_2) \vee (\neg i_0 \wedge s_2 \wedge s_5) \vee (i_3 \wedge (i_1 \neq s_3 \vee i_2 \neq s_4)\}.$$

The jump set contains the tuples in which the input position does not match the state position and the timer allows the controller to move, or the input faucet boolean is set and the temperature or rate needs to change.

The next step is to define f and g. We note that one benefit of modelling the problem in this way is its modularity. There is an inherent separation between the continuous and discrete aspects of the problem. In this context, f is defined by whatever form of dynamics one uses to model the behavior of the water in the tub. In the next section, we will detail how to use a Finite Element Simulator in order to derive values for f. However, when we demonstrate the optimization of the controller, we will use an extremely simplistic and somewhat arbitrary linear model since it is easier to simulate given our time constraints. This gives our model flexibility and scalability by allowing anyone using it to decide how much computational effort (or physical effort) she wants to exert to have a working model. Thus, as of now, we assume f is given. The jump function is simple, so I will describe in words. If the system reaches a jump state, it is because (1) $i_0 \neq s_2$ or (2) i_3. In the first case, set $i_0 = s_2$ given that the timer s_5 allows it. In the second case, if $i_1 \neq s_3$, set them equal, and if $i_2 \neq s_4$, set them equal. In other words, change the system to the settings that the controller desires, as expressed by the inputs. Together this gives our hybrid system $\mathcal{H} = \langle f, C, g, D \rangle$.

Given $\mathcal{H}$ we can define the explicit cost functions for our optimization. Let β be a trajectory of $\mathcal{H}$ for H time. Informally, β is a possible strategy. Let u_β be the implicit function of

temperature for each three dimensional point in terms of time (the same function defined above in Section 2). Then

$$C_1(\beta, T) = \int_0^H \int_x \int_y \int_z u_\beta(x, y, z, t) - T.$$

Let dT_β be the implicit function giving rate of temperature change in terms of time and let

$$C_2(\beta, C) = \int_0^H dT_\beta(t) - C.$$

Let R_β be the implicit function giving rate of water in terms of time and let

$$C_3(\beta, C) = \int_0^H R_\beta(t) - C.$$

Let P_β be the implicit function giving the position in terms of time and let

$$C_4(\beta, C) = \int_0^H P_\beta(t) - C.$$

For the given problem, we would like to minimize

$$F_\beta(T_{opt}, 0, 0, 0) = w_1 C_1(\beta, T_{opt}) + w_2 C_2(\beta, 0) + w_3 C_3(\beta, 0) + w_4 C_4(\beta, 0)$$

where T_{opt} the desired temperature of the water. The C_1 terms minimizes the distance from the temperature from being constant. The C_2 terms minimizes the amount of change of the temperature. In other words, it ensures a faster convergence by the controller. The C_3 term minimizes the volume of water used. The C_4 term minimizes the amount of time spent in the front of the tub. Each w_i gives the level of importance of each optimized value. For example, if water is expensive, w_3 may be increased so that it is more important to use less water than say close to some optimal temperature.

Again, the purpose of this optimization task is to give an optimal strategy for bathing with the additional constraint that the bath should not *seem* optimized. The minimization of moving ensures that the bather is relaxing for as long as possible.

5 Water Dynamics

As stated above, we put little reliance in our model on any particular derivation of the dynamics of the temperature throughout the tub. So we give an extreme case in which we calculate the changes using Final Element Analysis (FEA). We chose to use the Partial Differential Equation Toolbox for Matlab$^{\text{tm}}$ for accomplishing our FEA. Our code is included in the Appendix.

Table **??** details all the notation we will be using to describe our model. We use the following assumptions in our model:

1. Convection within the main body of water is negligible, thus we can describe the evolution of temperature using only heat diffusion.
2. At the boundaries of the water, convection of the air and water come into play. We assume the convective heat transfer coefficients at the boundaries are constant. We picked values for these two coefficients from a range of typical values ([**?**]).
3. We model the boundaries as a one dimensional plane wall problem ([**?**]).

ρ	density of water
c_p	specific heat of water
$T(x, y, z, t)$	Temperature, which is a function of space and time. Sometimes notated as simply T
q	heat exchange rate
h_air	convective coefficient of air. Has value $1 W/m^2K$.
h_water	convective coefficient of the bath water. Has value $50 W/m^2K$
$T_ambient$	Ambient temperature- the air in the bathroom.
T_{faucet}	The temperature of the water coming out of the bath faucet.
$\dot{V}$	The faucet flow rate.
A_{faucet}	The size of the water inlet. The faucet is modeled as a 2 inch diameter circle.
$k_{porcelain}$	The thermal conductivity of porcelain: $1.5 W/mK$ [**?**]
k	The thermal conductivity of water: $.628 W/mK$ [**?**]

Table 2: This table details all the notation we will use in this section.

4. We model the faucet (water intake) as a constant heat source. We neglect the fluid dynamics of water entering the tub.
5. We neglect to model the outlet in our particular FEA, but it could be added to the side of the tub as a constant heat sink.
6. We begin all of our models assuming that the temperature within the bath is completely even.
7. We model the density and specific heat of water as constant.
8. We ignore the thermal properties of the bathtub user.

We import a Computer Aided Design (CAD) model of the bath water into the PDE toolbox to create the necessary geometries (Fig. **??**). The PDE toolbox allows us to specify boundary conditions on every geometric face. The PDE we are solving is the Heat Equation, which models heat diffusion:

$$\rho c_p \frac{\partial T}{\partial t} - k\nabla^2 T) = q \tag{1}$$

We then define all the boundary conditions. All of our conditions are constraints on the heat flux, i.e. Neumann boundary conditions:

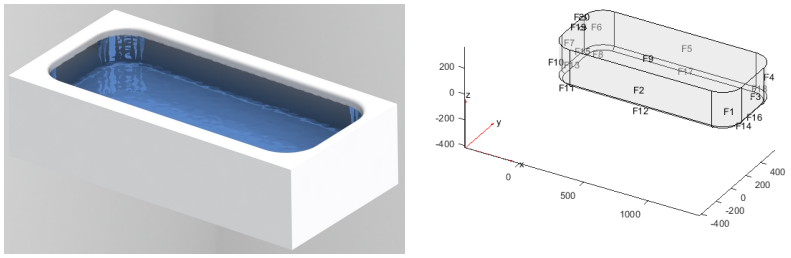

Figure 2: On the left is a render of the bath water with bath tub. The bath water is for the common 5 foot bath tub. On the right, we have the model as represented in Matlab with all of the geometric faces labeled.

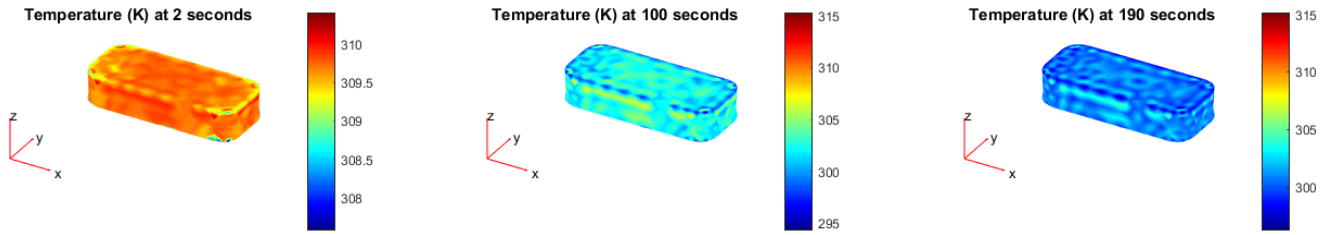

Figure 3: Evolution of the temperature distribution over time for a bath with initial water temperature $310K$, $T_{ambient} = 300K$, and the faucet in the off state.

1. Initial time condition: $T(x, y, z, 0) = T_0$ for all x, y, z.
2. Boundary condition for bathtub border: $q = 1/(1/h_{air} + 1/h_{water} + 1/k_{porcelain} * (T_{ambient} - T)$
3. Boundary condition for air boundary: $q = 1/(1/h_{air} + 1/h_{water}) * (T_{ambient} - T)$
4. Intake condition: $q = \rho c_p * T_{faucet} * \dot{V}/A_{intake}$

We have two CAD models of the bath water- one with a faucet inlet, and one without. In the case where the faucet is on, we use the former model and all the above boundary conditions. If the faucet is off, we use the latter model, and leave out the last boundary condition. The next step to setting up the simulation is generating the mesh- i.e. the nodes where the temperature will be numerically computed.

We can now run the PDE solver and get a numeric solution for how the temperature of the bath water will evolve over time. The results from one of our simulations can be found in Fig. **??**. We see that the solution follows basic intuition- the water surface cools faster than the sides, because the porcelain provides some insulation. The edges also cool much faster, which is intuitively correct. We performed multiple "sanity check" simulations to confirm our model

We use our FEA model to create a physical basis for our two metrics: unevenness, average temperature. We calculate the average temperature at each time step and we linearize the resulting curve using a simple linear regression. We define the unevenness as the mean square difference between the temperature at each point and the previously calculated average temperature. We also linearize the unevenness using linear regression. Figure **??** shows the values for our two metrics for a simulation where the initial temperature is $300K$, the $T_{faucet} = 325K$ and $T_{ambient} = 300K$. We produce linear dynamics models because of HYSDEL limitations.

6 Procedures and Results

As an example of solving for the optimal trajectory of the hybrid system $\mathcal{H}$ mentioned above, we use a naive set of linear equations to govern the dynamics of the water. We do this because, though simplifying the dynamics to linear behaviours does not properly capture the fluid and thermo-dynamics of water in a bath tub, the control and optimization framework is built such that the dynamics are modular and can be switched out for higher fidelity solutions. In addition, representing the dynamics with a set of linear equations enables us to provide a demonstration of the capabilities of the system within our personal time constraints.

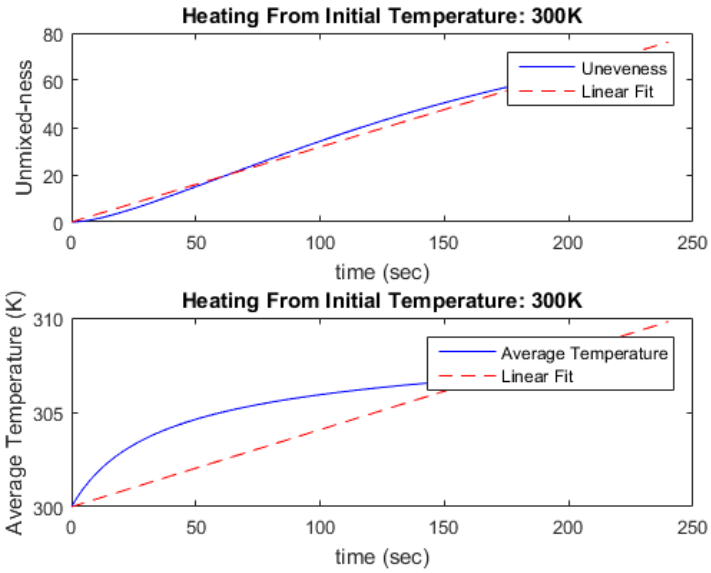

Figure 4: The calculated unevenness for a simulation where the initial temperature is $300K$, the $T_{faucet} = 325K$ and $T_{ambient} = 300K$. The unevenness of the water temperature increases with time, and so does the average temperature.

Our best option was to use the Hybrid Toolbox, which is built on top of the Hybrid Systems Description language (HYSDEL). This toolbox only supports affine dynamics so instead of performing a piecewise line regression to capture dynamics in a linear manner, we just choose even simpler linear equations that at least *characterize* the behaviors so the controller has a more realistic system to work with.

Among these approximations included adding an extra dimension in the state vector for *unevenness*, which is just meant to naively quantify the total deviation of the water's temperatures from its average temperature. In the optimization problem, the unevenness metric is minimized and the action of changing position helps decrement this value. This is meant to encapsulate the intuition that moving through the tub mixes the water.

In the Figure **??**, we can see that the controller, given a sequence of inputs, can adjust the trajectory of the water's temperature and eveness. With the full and accurate dynamics generated from the PDE toolbox, we can simply run the controller on the simulator coded in the appendix to find an optimal control sequence for a bath user to use.

7 Conclusions

The purpose of our model is to mathematically formulate the state space of being in a bath tub so that it is easier to understand how to build a strategy. In particular, we choose a state space that best represents what we believe is typical behavior. In doing so, our optimal strategy will not include atypical motion or decisions while bathing and will make comfort a high priority. This has the benefit that it completely modularizes the continuous and discrete aspects of the problem. The optimization problem has no explicit reliance on the specific state of the tub, but rather is robust for any kind of dynamics. Likewise, the problem of optimization does not change if we change the way the water dynamics are modelled. This flexibility allows us to

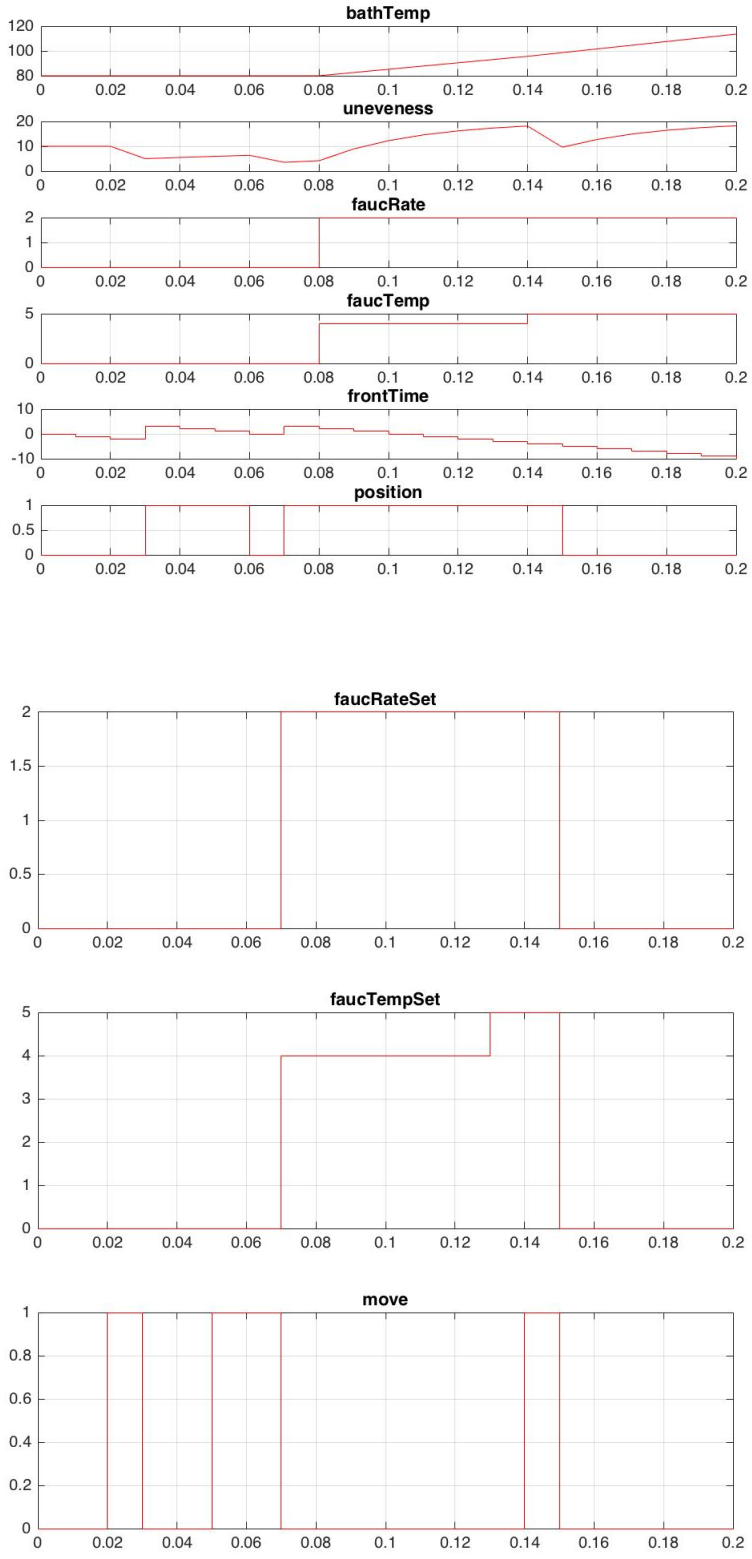

Figure 5: Notice the effects of the controller for a predetermined sequence of control inputs

focus on ensuring that the system specificatios are as desired, while also being able to decide *a posteriori* how we want to model the dynamics of the water.

It is understandable to see this as an unsatisfactory solution, as it does not directly address the problem of water dynamics, which is likely to play a large role in many other solutions. Though this is true, we believe that since we are asked to give a non-technical strategy for any bathtub user, we should give a solution that focuses more on the ability of the bather to control the bath, rather than on her ability to distribute heat based on the physical nature of the water. Another problem with our model is the lack of software supporting hybrid system optimization. There are many algorithms for optimization of hybrid systems as well as some software. But what we used, we found difficult, though powerful enough in many cases.

A question that we do not directly address in this paper is how our model is affected by the use of additives, such as bubble bath mixture or bath salts. As it is a reoccuring theme of our approach, we abstract out this question and leave it up to the modeller to decide how much of this to consider in the model. In the model as we stated it, the use of additives would simply change the boundary conditions and the constants used in our Finite Element Simulator. For example, the bath salts may change the specific heat of the water, and the bubble bath solution might change the boundary condition that specifies the affect of air on the temperatures of the water near the surface.

8 User Guide to Optimal Bathing

Purely for amusement, we format this non-technical report as a letter included in every new model of bathtub produced the fictional company BATHLAB.

Dear Customer,

Congratulations on your purchase of the BATHLAB ComforTUB51626! We know that you will enjoy this product for years to come. Our team of engineers have written this letter in order to give you some tips so you have have the *optimal* bathing experience.

References

[1] P.I. Barton, C.K. Lee, M. Yunt. Optimization of hybrid systems. *Computers and chemical engineering*, 30(10), pp.1576-1589, 2006.

[2] A. Bemporad. Hybrid Toolbox - User's Guide. `http://cse.lab.imtlucca.it / bemporad/hybrid/toolbox`, 2004.

[3] T.L. Bergman and F.P. Incropera. Fundamental of Heat and Mass Transfer. Hoboken, NJ, Wiley, 2011.

[4] R. Sanfelice, D. Copp, and P. Nanez. A toolbox for simulation of hybrid systems in Matlab/Simulink: Hybrid Equations (HyEQ) Toolbox. *In Proceedings of the 16th international conference on Hybrid systems: computation and control*, pp. 101-106. ACM, 2013.

[5] Typical Convection Coefficients to Estimate Thermal Transfer. `http://www.me.mtu.edu/ microweb/GRAPH/Intro/film.htm`

9 Code

9.1 HYSDEL Description of our Hybrid Model

```
/* Bath Time */

SYSTEM hyb_bath {

    INTERFACE {

        STATE {
            REAL bathTemp [-150, 150];
            REAL uneveness [0, 20];
            REAL faucRate [0, 5];
            REAL faucTemp [0, 5];
            BOOL position;
            REAL frontTime [-200, 200];
        }

        INPUT {
            REAL faucRateSet [-1, 5];
            REAL faucTempSet [-1, 5];
            BOOL move;
        }

        OUTPUT{
            REAL bathTempOut;
            REAL unevenessOut;
            REAL faucRateOut;
            REAL faucTempOut;
            REAL positionOut;
        }

        PARAMETER{
            REAL naturalCoolRate;
            REAL ambTemp;
            REAL naturalUnevenRate;
            REAL mixingEvenRate;
            REAL a1, a2, a3, a4, a5;
            REAL b1, b2, b3, b4, b5;
            REAL pause;
        }
    }

    IMPLEMENTATION {

        AUX {
            REAL currCoolRate;
            BOOL atAmb;
            BOOL moving, moveAllowed;
            BOOL notFaucReady0, notFaucReady1;
```

```
    BOOL d0, d1, d2, d3, d4, d5;
    BOOL e0, e1, e2, e3, e4, e5;
    BOOL f11, f12, f13, f14, f15;
    BOOL f21, f22, f23, f24, f25;
    BOOL f31, f32, f33, f34, f35;
    BOOL f41, f42, f43, f44, f45;
    BOOL f51, f52, f53, f54, f55;
    REAL tempChange11, tempChange12, tempChange13, tempChange14;
    REAL tempChange15, tempChange21, tempChange22, tempChange23;
    REAL tempChange24, tempChange25, tempChange31, tempChange32;
    REAL tempChange33, tempChange34, tempChange35, tempChange41;
    REAL tempChange42, tempChange43, tempChange44, tempChange45;
    REAL tempChange51, tempChange52, tempChange53, tempChange54;
    REAL tempChange55;
    REAL unevenessChange;
    REAL moveMixing, moveTimerSet;
    BOOL faucOff;
    REAL setFaucRate, setFaucTemp;
    REAL positionValue;
}

AD {
    atAmb = bathTemp <= ambTemp;

    moveAllowed = frontTime <= 0;
    notFaucReady0 = faucRateSet <= -1;
    notFaucReady1 = faucTempSet <= -1;
    faucOff = faucRate <= 0;

    d0 = faucRate <= 0;
    d1 = faucRate <= 1;
    d2 = faucRate <= 2;
    d3 = faucRate <= 3;
    d4 = faucRate <= 4;
    d5 = faucRate <= 5;

    e0 = faucTemp <= 0;
    e1 = faucTemp <= 1;
    e2 = faucTemp <= 2;
    e3 = faucTemp <= 3;
    e4 = faucTemp <= 4;
    e5 = faucTemp <= 5;
}

LOGIC {
    moving = move & moveAllowed;

    f11 = (~d0 & d1) & (~e0 & e1);
    f12 = (~d0 & d1) & (~e1 & e2);
    f13 = (~d0 & d1) & (~e2 & e3);
    f14 = (~d0 & d1) & (~e3 & e4);
    f15 = (~d0 & d1) & (~e4 & e5);

    f21 = (~d1 & d2) & (~e0 & e1);
```

```
    f22 = (~d1 & d2) & (~e1 & e2);
    f23 = (~d1 & d2) & (~e2 & e3);
    f24 = (~d1 & d2) & (~e3 & e4);
    f25 = (~d1 & d2) & (~e4 & e5);

    f31 = (~d2 & d3) & (~e0 & e1);
    f32 = (~d2 & d3) & (~e1 & e2);
    f33 = (~d2 & d3) & (~e2 & e3);
    f34 = (~d2 & d3) & (~e3 & e4);
    f35 = (~d2 & d3) & (~e4 & e5);

    f41 = (~d3 & d4) & (~e0 & e1);
    f42 = (~d3 & d4) & (~e1 & e2);
    f43 = (~d3 & d4) & (~e2 & e3);
    f44 = (~d3 & d4) & (~e3 & e4);
    f45 = (~d3 & d4) & (~e4 & e5);

    f51 = (~d4 & d5) & (~e0 & e1);
    f52 = (~d4 & d5) & (~e1 & e2);
    f53 = (~d4 & d5) & (~e2 & e3);
    f54 = (~d4 & d5) & (~e3 & e4);
    f55 = (~d4 & d5) & (~e4 & e5);
}

DA {
    currCoolRate = {
        IF atAmb THEN 0
        ELSE naturalCoolRate
    };

    unevenessChange = {
        IF faucOff THEN naturalUnevenRate*(10-uneveness)
        ELSE .3*(20-uneveness)
    };

    moveMixing = {
        IF moving THEN -mixingEvenRate*(uneveness)
        ELSE 0
    };

    moveTimerSet = {
        IF moving & ~position THEN pause
        ELSE frontTime - 1
    };

    setFaucRate = {
        IF ~notFaucReady0 & position THEN faucRateSet
        ELSE faucRate
    };

    setFaucTemp = {
        IF ~notFaucReady1 & position THEN faucTempSet
        ELSE faucTemp
    };
```

```
positionValue = {
    IF position THEN 1
    ELSE 0
};

tempChange11 = {
    IF f11 THEN a1*b1
    ELSE 0
};
tempChange12 = {
    IF f12 THEN a1*b2
    ELSE 0
};
tempChange13 = {
    IF f13 THEN a1*b3
    ELSE 0
};
tempChange14 = {
    IF f14 THEN a1*b4
    ELSE 0
};
tempChange15 = {
    IF f15 THEN a1*b5
    ELSE 0
};

tempChange21 = {
    IF f21 THEN a2*b1
    ELSE 0
};
tempChange22 = {
    IF f22 THEN a2*b2
    ELSE 0
};
tempChange23 = {
    IF f23 THEN a2*b3
    ELSE 0
};
tempChange24 = {
    IF f24 THEN a2*b4
    ELSE 0
};
tempChange25 = {
    IF f25 THEN a2*b5
    ELSE 0
};

tempChange31 = {
    IF f31 THEN a3*b1
    ELSE 0
};
tempChange32 = {
    IF f32 THEN a3*b2
```

```
    ELSE 0
};
tempChange33 = {
    IF f33 THEN a3*b3
    ELSE 0
};
tempChange34 = {
    IF f34 THEN a3*b4
    ELSE 0
};
tempChange35 = {
    IF f35 THEN a3*b5
    ELSE 0
};

tempChange41 = {
    IF f41 THEN a4*b1
    ELSE 0
};
tempChange42 = {
    IF f42 THEN a4*b2
    ELSE 0
};
tempChange43 = {
    IF f43 THEN a4*b3
    ELSE 0
};
tempChange44 = {
    IF f44 THEN a4*b4
    ELSE 0
};
tempChange45 = {
    IF f45 THEN a4*b5
    ELSE 0
};

tempChange51 = {
    IF f51 THEN a5*b1
    ELSE 0
};
tempChange52 = {
    IF f52 THEN a5*b2
    ELSE 0
};
tempChange53 = {
    IF f53 THEN a5*b3
    ELSE 0
};
tempChange54 = {
    IF f54 THEN a5*b4
    ELSE 0
};
tempChange55 = {
    IF f55 THEN a5*b5
```

```
                ELSE 0
            };
        }

        CONTINUOUS {
            bathTemp = bathTemp - currCoolRate +
                       tempChange11 + tempChange12 + tempChange13 +
                       tempChange14 + tempChange15 + tempChange21 +
                       tempChange22 + tempChange23 + tempChange24 +
                       tempChange25 + tempChange31 + tempChange32 +
                       tempChange33 + tempChange34 + tempChange35 +
                       tempChange41 + tempChange42 + tempChange43 +
                       tempChange44 + tempChange45 + tempChange51 +
                       tempChange52 + tempChange53 + tempChange54 +
                       tempChange55;

            uneveness = uneveness + unevenessChange + moveMixing;

            faucRate = setFaucRate;
            faucTemp = setFaucTemp;

            frontTime = moveTimerSet;
        }
        AUTOMATA {
            position = (position | move) & ~(position & move);
        }

        OUTPUT {
            bathTempOut = bathTemp;
            unevenessOut = uneveness;
            faucRateOut = faucRate;
            faucTempOut = faucTemp;
            positionOut = positionValue;

        }
    }
}
```

9.2 MATLAB simulation and optimization of controller

```
clear variables
close all

%parameters
naturalCoolRate = .005;
ambTemp = 80;
naturalUnevenRate = .1;
mixingEvenRate = .5;
a1 = .2; a2 = .4; a3 = .6; a4 = .8; a5 = 1; %linear interpolation slopes
b1 = .2; b2 = .4; b3 = .6; b4 = .8; b5 = 1;
pause = 3;
Ts = 0.01;
```

```
targetTemp = 95;

%initial conditions
initTemp = 80;

%creating dynamics object
S = mld('bath',Ts);

%open loop sim
U = [
        0, 0, 0;
        0, 0, 0;
        0, 0, 1;
        0, 0, 0;
        0, 0, 0;
        0, 0, 1;
        0, 0, 1;
        2, 4, 0;
        2, 4, 0;
        2, 4, 0;
        2, 4, 0;
        2, 4, 0;
        2, 4, 0;
        2, 5, 0;
        2, 5, 1;
        0, 0, 0;
        0, 0, 0;
        0, 0, 0;
        0, 0, 0;
        0, 0, 0;
        0, 0, 0;
    ];

x0 =[
        80; %bathTemp
        10; %uneveness
        0; %faucRate
        0; %faucTemp
        0; %frontTime
        0 %position
    ];
[X,T,D,Z,Y]=sim(S,x0,U);

%plot simulation results
figure('Name', 'State');
subplot(611)
plot(T,X(:, 1),'r')
title('bathTemp')
grid
subplot(612)
plot(T,X(:, 2),'r')
title('uneveness')
grid
subplot(613)
```

```
stairs(T,X(:, 3),'r')
title('faucRate')
grid
subplot(614)
stairs(T,X(:, 4),'r')
title('faucTemp')
grid
subplot(615)
stairs(T,X(:, 5),'r')
title('frontTime')
grid
subplot(616)
stairs(T,X(:, 6),'r')
title('position')
grid

figure('Name', 'Input');
subplot(311)
stairs(T,U(:, 1),'r')
title('faucRateSet')
grid
subplot(312)
stairs(T,U(:, 2),'r')
title('faucTempSet')
grid
subplot(313)
stairs(T,U(:, 3),'r')
title('move')
grid

set(gcf,'Position',[  328    68   560   626]);

%create controller for system
clear Q refs limits
refs.x=[1 2 3 5];
Q.x=[10 0 0 0;
     0  3 0 0;
     0  0 2 0;
     0  0 0 1];
Q.rho=Inf;
Q.norm=Inf;

N=4;
limits.umin= [-1, -1, 0];
limits.umax= [5, 5, 1];
limits.xmin=[60, 0, 0, 0, 0, -1000];
limits.xmax=[130, 20, 5, 5, 1, 3];

C=hybcon(S,Q,N,limits,refs);

nominal = [targetTemp, 10, 0, 0];
Tstop=8;
r=struct('x', nominal);
x0=[initTemp; 10; 0; 0; 0; 0];
```

```
[X,U,D,Z,T,Y]=sim(C,S,r,x0,Tstop);

%plot controller performance
figure('Name', 'Performance of HybCon');
clf
subplot(511)
plot(T,X(:,1))
title('bathTemp')
grid
axis([0 Tstop 70 120])
subplot(512)
plot(T,X(:,2))
title('uneveness')
grid
subplot(513)
stairs(T,X(:,3))
title('faucRate')
grid
subplot(514)
stairs(T,X(:,4))
title('faucTemp')
grid
subplot(515)
stairs(T,X(:,5))
title('position')
grid

figure('Name', 'HybCon Inputs');
subplot(311)
stairs(T,U(:, 1),'r')
title('faucRateSet')
grid
subplot(312)
stairs(T,U(:, 2),'r')
title('faucTempSet')
grid
subplot(313)
stairs(T,U(:, 3),'r')
title('move')
grid


%% Import Model
clear all
close all

Tinit = 310;
Tambient = 300;

model = createpde(1);
importGeometry(model,'correctWater.stl')
%h=pdegplot(model,'FaceLabels','on');
%h(1).FaceAlpha=0.3;
```

```
hair = 1;
hwater=50;
k = 1.5;

cp = 4.178/1000;
p = 971.8;

d=1*cp*p;
c=.628;
f=0;
a=0;

%% Set Boundaries and Gen Mesh

q1 = 1/(1/hair + 1/hwater);
g1 = Tambient/(1/hair + 1/hwater);


q = 1/(1/hair + 1/hwater+1/k);
g = Tambient/(1/hair + 1/hwater+1/k);


x = [1,0,0];
nx = [-1,0,0];
y = [0,1,0];
ny = [0,-1,0];
z = [0,0,1];
nz = [0,0,-1];

xg = [1,0,0];
nxg = [-1,0,0];
yg = [0,1,0];
nyg = [0,-1,0];
zg = [0,0,1];
nzg = [0,0,-1];

applyBoundaryCondition(model,'Face',9,'q',q1,'g',g1)
applyBoundaryCondition(model,'Face',18,'q',q,'g',g)
applyBoundaryCondition(model,'Face',[1,2,3,4,5,6,7,8,10,11,12,13,14,15,16,17],...
    'q',q,'g',g)

hmax = 200;
generateMesh(model,'Hmax',hmax,'GeometricOrder','quadratic')

%figure
%pdeplot3D(model)

%% Run sim

u = parabolic(Tinit,0:.2:240,model,c,a,f,d);
result = createPDEResults(model,u);
rs = result.NodalSolution;
-------------------------------------
%% Import Model
```

```
clear all
close all

Tinit = 300;
Tambient = 300;
Tfaucet=325;
FaucetRate = .00378541/60* 7;

model = createpde(1);
importGeometry(model,'waterOn.stl')
h=pdegplot(model,'FaceLabels','on');
h(1).FaceAlpha=0.3;

hair = 1;
hwater=50;
k = 1.5;

cp = 4.178/1000;
p = 971.8;

d=1*cp*p;
c=.628;
f=0;
a=0;

%% Set Boundaries and Gen Mesh

q1 = 1/(1/hair + 1/hwater);
g1 = Tambient/(1/hair + 1/hwater);


q = 1/(1/hair + 1/hwater+1/k);
g = Tambient/(1/hair + 1/hwater+1/k);

applyBoundaryCondition(model,'Face',9,'q',q1,'g',g1)
applyBoundaryCondition(model,'Face',20,'q',0,'g',p*cp*Tfaucet*FaucetRate * pi/0.0254^2
applyBoundaryCondition(model,'Face',[1,2,3,4,5,6,7,8,10,11,12,13,14,15,16,17,18,19],..
    'q',q,'g',g)

hmax = 200;
generateMesh(model,'Hmax',hmax,'GeometricOrder','quadratic')

figure
pdeplot3D(model)

%% Run sim

u = parabolic(Tinit,0:.2:240,model,c,a,f,d);
result = createPDEResults(model,u);
rs = result.NodalSolution;
-----------------------------------
temp = size(rs);
timeStep = 0.2;
Tnominal = 305;
```

```
len=temp(2);
data1 = 1:len;
data2 = 1:len;
t = (0:(len-1)) .* timeStep;

for i = 1:len
    data2(i) = sum(rs(:,i))/temp(1);
    data1(i) = sum((data2(i)-rs(:,i)).^2)/temp(1)^2;
end

slopeMix = t'\(data1-data1(1))';
subplot(2,1,1)
plot(t,data1,'-b',t,slopeMix.*t + data1(1),'--r')
legend('Uneveness','Linear Fit')
title('Heating From Initial Temperature: 300K')
xlabel('time (sec)')
ylabel('Unmixed-ness')

tempAvg = t'\(data2-data2(1))';
subplot(2,1,2)
plot(t, data2,'-b', t, tempAvg.*t + data2(1),'--r')
legend('Average Temperature', 'Linear Fit')
title('Heating From Initial Temperature: 300K')
xlabel('time (sec)')
ylabel('Average Temperature (K)')

[slopeMix,tempAvg]
```