Optimal Control of Active Nematics using COMSOL and MATLAB

Michael M. Norton,[1, 2, *] Piyush Grover,[3] Michael F. Hagan,[2] and Seth Fraden[2]

[1] *Center for Neural Engineering, Department of Engineering Science and Materials, Pennsylvania State University, University Park, Pennsylvania 16801*

[2] *Physics Department, Brandeis University, Waltham, Massachusetts 02453*

[3] *Mechanical and Materials Engineering, University of Nebraska - Lincoln, Lincoln Nebraska 68588*

(Dated: September 7, 2020)

I. INTRODUCTION

This document describes how to configure and run the MATLAB scripts that perform the optimal control calculation presented in `https://arxiv.org/abs/2007.14837`. The code utilizes the numerical technique "direct adjoint looping" [2] to iteratively converge on a spatio-temporal field for either activity strength α or applied vorticity g, through consecutive finite element simulations consisting of forward, backward (adjoint), and update (gradient descent) steps, Fig. 1.

All files are described in 1; variables and parameters in tables 2,3 and 4. For those users interested in modifying the code to suit their own PDE systems, tables 5 and 6 describe all COMSOL PDE entries and how they are enabled/disabled during the adjoint loop.

All scripts are run within MATLAB. While the final output is a COMSOL `*.mph` file, there is largely no need to access the COMSOL gui to perform any calculations.

The code will be maintained at: GitHub: `https://github.com/wearefor/activenematic_oc` and OSF: `https://osf.io/qyk9t/`.

II. SOFTWARE REQUIREMENTS

- MATLAB®, written with 2019a, compatibility with other versions is untested
- COMSOL Multiphysics®with MATLAB LiveLink module, code written for v5.2, compatibility with other versions is untested
- matlab-ascii-plot, free download from MATLAB Central [1]

III. RUNNING THE CODE

1. **Initializing:** The script `AN_adjointloop_setup.m` serves two functions. Firstly, it creates a COMSOL model with the necessary equations to model nematohydrodynamics and adjoint dynamics, as well as to perform gradient descent on the control fields. Secondly, it generates target solutions for the clockwise (CW) and counterclockwise (CCW) circulating states.

   The output of this script will be saved in a data file appended with the date and time: `LimitCycles_yyyymmdd_HHMM.mph`. Note this file name, and define the string variable `params.limitcyclestr` in the file `AN_adjointloop.m` accordingly.

[*] mike.m.norton@gmail.com

2. **Running the Adjoint Loop:** The primary execution script is `AN_adjointloop.m`. The beginning of the script defines parameters in lines ∼1-115; the adjoint loop starts on 289. When executed, the script will load the previously generated `*.mph` file (define `params.limitcyclestr`) and proceed to configure and execute forward, adjoint, and update steps through several auxiliary functions. All of the computational heavy lifting is done in COMSOL, the sole purpose of the scripts are to configure the "studies" in COMSOL.

   All simulation and numerical parameters (with the exception of the base level of active stress `alpha0`, domain size `DomRad`, flow alignment `lam1`, and perhaps a few other parameters) are defined in the beginning of this file, details are below in tables 2,3,4.

   While running, the script outputs simulation details into the terminal that I've found most useful for monitoring the progress. These include notifications when COMSOL has been asked to do a potentially time-intensive task, a breakdown of all contributions to the cost J of the current control solution, the circulation, and ascii time plots of control effort. Two `*.mat` files `debug.mat` and `params_.mat` are also generated that save the parameter structure `params` and some additional diagnostic variables.

   Direct adjoint looping can be a memory-intensive process [2]. The small computational domain and short control windows considered for the problem described in the manuscript partially alleviate this concern, but this will not be the case for all problems of interest. To avoid large data files and clear up memory, the code alternates between saving the control field in `A` and `B` datasets (in other words there are always two control solutions in memory, the current iteration and the previous), Fig. 1 illustrates the process. By contrast, forward and adjoint solutions are overwritten at each iteration since there is never a need to simultaneously use dynamics from different iterations. However, for debugging purposes it can be helpful to access intermediate steps, this can be done by setting `params.save=1`, this will perform calculations as usual but save an `*.mph` file to the disk. Because of the need to flip between different datasets, odd and even iterations require slightly different setups so that solvers for the forward and adjoint steps are using the correct control solution. See table 5 for a summary of the PDE equations and table 6 for how these PDEs are activated/deactivated. The flow chart Fig. 1 illustrates the concept.

3. **Modifying the Control Target:** By default, the scripts are configured to use either CW or CCW solutions, which are saved in the `LimitCycles...mph` file, as the target solution for $\mathbf{Q}^*$ in a tracking problem (i.e., there is a penalty incurred at all times for not matching the target trajectory). A different solution (or function) can be used instead by modifying the file `func_config_adjoint.m`; this reference field can be time varying or stationary. There are several references to these solutions to cover the different cases of the adjoint loop. A quick way to identify them all is to search for all instances of `sol_isolate_...` and edit as needed.

4. **Modifying the Physics:** Modifying the governing equations will necessitate re-deriving the adjoint equations and changing the code accordingly. When new physics involve the control field or if a new control field is introduced, the gradient of the cost function with respect to the control field will also need to be updated. The validity of the adjoint equations in the paper was numerically verified using finite differences. If changes are made, the new equations can be checked in the following way: First, we note that applying point perturbations in space/time to the control solution for the purposes of computing a numerical derivative in COMSOL is not possible. Instead, one can consider an auxiliary problem that introduces a smooth perturbation, such as a Gaussian bump, to the control field of interest, and assess the sensitivity of the cost function to the amplitude of the perturbation A by both adjoint and finite difference methods. We note that since the "control variable" A is a constant in this example, the gradient will now also be a single, scalar value rather than a function of time and space as it was in the paper (eqns. 6 and 7). More detail on the adjoint method for PDEs can be found in the following tutorial [3].

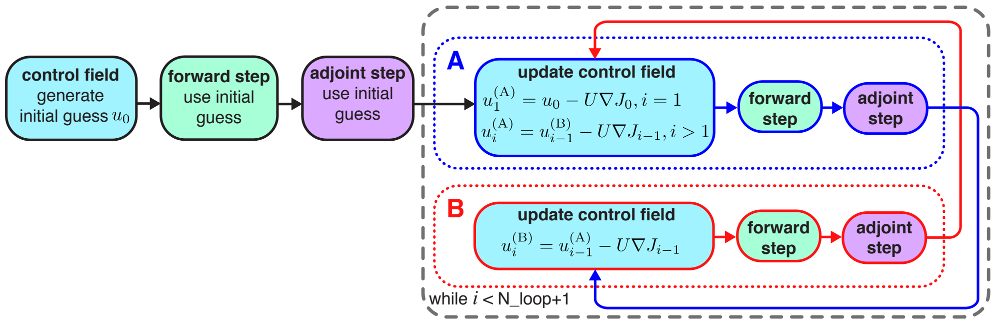

FIG. 1. A visual reprsentation of the adjoint loop as it is implemented in the provided scripts. In the diagram, u referes to either control field (α or g) and U is the gradient step size. For more detail on how equations are enabled/distabled at various stages of the loop see table 6.

file/function name	description
`AN_adjointloop_setup.m`	configuration script that defines physics, geometry, and boundary conditions, and generates reference solutions (i.e. cw and ccw attractors).
`AN_adjointloop.m`	defines control parameters (such as weights) and numerical parameters, and runs adjoint loop
`AN_plots.m`	imports data and generates field plots
`func_armijo.m`	armijo line search for determining gradient descent step size [4]
`func_config_adjoint.m`	configures adjoint physics for current
`func_config_forward.m`	configures forward
`func_config_update.m`	configures control field update
`func_cost.m`	calculate value of cost function
`func_mphsave.m`	saves comsol mph file
`func_plot_controlfield.m`	plots control field
`func_plot_director.m`	plot director field and degree of order
`func_plot_flowfield.m`	plots flow field and vorticity
`func_plotcontrol.m`	plots the spatial maximum of either α or g as a function of time in a command line asciiplot
`asciiplot.m`	generates command line ascii plots, the width and height may need to be modified to suit your window size[1]

TABLE 1. A list of all scripts and their usage.

IV. NOMENCLATURE AND QUICK REFERENCE GUIDE

symbol	COMSOL variable name	description
	forward dynamics	
$\mathbf{Q} = \begin{pmatrix} Q_{xx} & Q_{xy} \\ Q_{xy} & -Q_{xx} \end{pmatrix}$	`Qxx, Qxy`	components of nematic order tensor
$\mathbf{u} = \{u_x, u_y\}$	`ux, uy`	components of velocity field
p	`p`	pressure field
	adjoint dynamics	
$\boldsymbol{\psi} = \begin{pmatrix} \psi_{xx} & \psi_{xy} \\ \psi_{xy} & -\psi_{xx} \end{pmatrix}$	`psixx, psixy`	components of adjoint nematic order tensor field
$\boldsymbol{\nu} = \{\nu_x, \nu_y\}$	`nux, nuy`	components of adjoint velocity field
ϕ	`phi`	adjoint pressure field
	gradient descent update	
α	`Anew`	active stress field
$\frac{\partial J}{\partial \alpha}$	`Agrad`	gradient of cost function w.r.t. active stress control field
g	`Gnew`	applied vorticity field
$\frac{\partial J}{\partial g}$	`Ggrad`	gradient of cost function w.r.t. applied vorticity control field

TABLE 2. All field variables as they appear in the manuscript and their variable name in COMSOL.

symbol	MATLAB varible	COMSOL variable	value	description
constants defined in **AN_adjointloop_setup.m**				
α_0	-	`alpha0`	fixed: 5	base level of activity
λ	-	`lam1`	fixed: 1	flow alignment
R	-	`DomRad`	fixed: 6.5	domain radius
ρ_0	-	`rho0`	fixed: 1.6	nematic density
-	-	`tLCmax`	15	duration of reference solution (note that the period of the limit cycle solution used in the paper is ~ 5)
penalty weights for control				
-	`params.A_Q_weight`	`A_Q_weight`	-	terminal penalty weight on $\Delta\mathbf{Q}$
-	`params.B_Q_weight`	`B_Q_weight`	-	terminal penalty weight on $\Delta\mathbf{u}$
W	`params.C_Q_weight`	`C_Q_weight`	$\sim$100-1000	stage penalty weight on $\Delta\mathbf{Q}$
-	`params.D_Q_weight`	`D_Q_weight`	-	stage penalty weight on $\Delta\mathbf{u}$
Γ_α	-	`Gamma_alpha`	0.1	penalty on active stress gradients
Γ_g	-	`Gamma_g`	0.1	penalty on applied vorticity gradients
initial conditions, initial control guesses, control window, etc.				
t_f	`params.Tf`	-	2	control window duration
-	`params.dt`	-	0.05	time step for saving
-	`params.tlist`	-	[0:dt:Tf]	list of times
-	-	`A(t)`	`alpha0`	naive initial guess for $\alpha(\mathbf{x}, t)$
-	`params.G0`	`G0`	-0.75	strength of initial guess value for applied vorticity
-	-	`G(t)`	$G_0 (1 - \Theta (t - t_f/2))$	uniform time-varying initial guess for applied vorticity
-	`params.thetastart`	-	3	phase of initial conditions used in paper (units of time)
θ	`params.theta`	`theta`	$\in$ [`tLCmax`-t_f, `tLCmax`]	target phase (units of time): following the convention in the paper, the code shifts the lookup of the target solution such that at $t = t_f$, $\mathbf{Q}^* (\theta)$, similarly, at the beginning of the control window, the control target is $\mathbf{Q}^* (\theta - t_f)$. The range on admissable values for θ is set by the duration of the reference solution `tLCmax`

TABLE 3. List of parameters controlling the optimal control problem. If the field is empty, then the variable may not be defined in the paper and/or may only exist in MATLAB or COMSOL. For example, no penalty was assigned to the velocity field $\mathbf{u}$, but code exists to support this. Some parameters have both MATLAB and COMSOL names because they are first defined in MATLAB and then passed into COMSOL.

parameter name	value	description
`params.armijoskipfirst`	0 or 1	option to skip armijo step size optimization
`params.armijoskipcutoff`	integer	specifies number of adjoint loop steps to skip Armijo backtracking
`gradstep` (COMSOL parameter)	-	gradient step size, determined by either:
		1) the list `params.gradstep` list or 2) Armijo backtracking
		note: `gradstep` multiples gradient ∇J not $\nabla J/\lvert\nabla J\rvert$
`params.gradstep`	$10^{-7} - 10^{-4}$	sequence of gradient step sizes to use in place of backtracking
`params.armijo_gammaMax`	1	maximum gradient step size scaling factor to consider, $u_{i+1} = u_i - \gamma\nabla J/\lvert\nabla J\rvert$
`params.armijo_gradstepMin`	10^{-11}	minimum gradient step size
`params.armijo_beta`	1	step size rescaling
`params.armijo_mu`	1	search region size
`params.armijo_mureduce`	0.6	If the search region is too large a a negative cost function goal can result,
		this scaling factor is used to iteratively reduce the Armijo search region
`params.costtol`	10^{-6}	stop condition, relative tolerance on cost function
`params.Q_err_tol`	0	alternative stop condition, relative tolerance on $\Delta\mathbf{Q}\,(t = t_f)$
`params.Nloop`	$\sim$50	If tolerances are not met, the maximum number of adjoint loops to perform

TABLE 4. List of numerical tuning parameters. These are variables for controlling various aspects of adjoint looping and Armijo backtracking. These are all Matlab parameters except for **gradstep**, which exists as a Comsol parameter.

physics name	description
	Forward Dynamics:
w	Nematic Dynamics, WeakFormPDE for `{Qxx,Qxy}`
w/wfeq1	time derivative
w/wfeq3	material derivative: velocity contribution
w/wfeq4	material derivative: vorticity contribution
w/wfeq5	flow alignment
w/wfeq7	distortion relaxation
w/wfeq15	order/disorder dynamics
w/wfeq16	applied vorticity $\mathbf{g}$
w2	Stokes flow, WeakFormPDE for `{ux,uy}`
w2/wfeq1	viscous forces, pressure gradient
w2/wfeq2	uniform and steady active stress
w2/wfeq4	active stress defined during adjoint looping
w2/wfeq5	time-varying, spatially uniform initial guess for α, defined by comsol function A(t)
w2/wfeq6	time-derivative, not utilized, Re= 0
w3	WeakFormPDE for `p`
w3/wfeq1	continuity equation
	Backward/Adjoint Dynamics:
w4	WeakFormPDE for `{psixx, psixy}` (adjoint variables of nematic order tensor)
w4/wfeq1	adjoint dynamics, all contributions except those arising from control fields and stage penalties
w4/wfeq2	time-varying, spatially uniform initial guess for α, defined by comsol function A(t)
w4/wfeq3	active stress defined during adjoint looping
w4/wfeq4	stage penalty on $\Delta\mathbf{Q}$
w4/wfeq5	applied vorticity defined during adjoint looping
w4/wfeq6	uniform and steady active stress
w4/wfeq7	time-varying, spatially uniform initial guess for active vorticity g, defined by comsol function G(t)
w5	WeakFormPDE for `{nux, nuy}` (adjoint variables of velocity)
w5/wfeq1	adjoint dynamics, all contributions except those arising from stage penalties
w5/wfeq2	stage penalty on $\Delta\mathbf{u}$ (unused in paper)
w6	WeakFormPDE for `phi` (adjoint variable of pressure `p`)
w6/wfeq1	continuity equation
	Control Field Update:
w7	Active Stress Update
w7/wfeq1	gradient descent update
w7/wfeq2	define gradient, update A
w7/wfeq3	define gradient, update B
w8	Applied Vorticity Update
w8/wfeq1	gradient descent update
w8/wfeq2	define gradient, update A
w8/wfeq3	define gradient, update B

TABLE 5. Comsol physics configured in **AN_matlab_adjointloop_setup.m**. Most of the PDEs are defined symbolically in their weak form in this initial setup file. The exceptions are those that depend on solutions that are only created once the adjoint loop has begun (they will either be commented out in this this setup script or absent altogether). These will be created/enabled as needed in the files **func_config_forward.m**, **func_config_adjoint.m**, and **func_config_update.m**.

	initialization		in adjoint loop, i=1			in adjoint loop, even (B)			in adjoint loop, odd (A)		
physics/PDE	forward	adjoint	control	forward	adjoint	control	forward	adjoint	control	forward	adjoint
w	on	off	off	on	off	off	on	off	off	on	off
w/wfeq1	on	-	-	on	-	-	on	-	-	on	-
w/wfeq3	on	-	-	on	-	-	on	-	-	on	-
w/wfeq4	on	-	-	on	-	-	on	-	-	on	-
w/wfeq5	on	-	-	on	-	-	on	-	-	on	-
w/wfeq7	on	-	-	on	-	-	on	-	-	on	-
w/wfeq15	on	-	-	on	-	-	on	-	-	on	-
w/wfeq16	g*	-	-	g*	-	-	*	-	-	g*	-
w2	on	off	off	on	off	off	on	off	off	on	off
w2/wfeq1	on	-	-	on	-	-	on	-	-	on	-
w2/wfeq2	g*	-	-	g*	-	-	g*	-	-	g*	-
w2/wfeq4	off	-	-	α*	-	-	α*	-	-	α*	-
w2/wfeq5	α*	-	-	-	-	-	-	-	-	-	-
w2/wfeq6	off	-	-	off	-	-	off	-	-	off	-
w3	on	off	off	on	off	off	on	off	off	on	off
w3/wfeq1	on	-	-	on	-	-	on	-	-	on	-
w4	off	on	off	off	on	off	off	on	off	off	on
w4/wfeq1	-	on	-	-	on	-	-	on	-	-	on
w4/wfeq2	-	α*	-	-	off	-	-	off	-	-	off
w4/wfeq3	-	off	-	-	α*	-	-	α*	-	-	α*
w4/wfeq4	-	on	-	-	on	-	-	on	-	-	on
w4/wfeq5	-	off	-	-	g*	-	-	g*	-	-	g*
w4/wfeq6	-	g*	-	-	g*	-	-	g*	-	-	g*
w4/wfeq7	-	g*	-	-	off	-	-	off	-	-	off
w5	off	on	off	off	on	off	off	on	off	off	on
w5/wfeq1	-	on	-	-	on	-	-	on	-	-	on
w5/wfeq2	-	on	-	-	on	-	-	on	-	-	on
w6	off	on	off	off	on	off	off	on	off	off	on
w6/wfeq1	-	on	-	-	on	-	-	on	-	-	on
w7	off	off	α*	off	off	α*	off	off	α*	off	off
w7/wfeq1	-	-	α*	-	-	off	-	-	off	-	-
w7/wfeq2	-	-	off	-	-	off	-	-	α*	-	-
w7/wfeq3	-	-	off	-	-	α*	-	-	off	-	-
w8	off	off	g*	off	off	g*	off	off	g*	off	off
w8/wfeq1	-	-	g*	-	-	off	-	-	off	-	-
w8/wfeq2	-	-	off	-	-	off	-	-	g*	-	-
w8/wfeq3	-	-	off	-	-	g*	-	-	off	-	-

TABLE 6. **Enabled/Disabled physics lookup table.** If the field needs to be solved at a given step the "parent" entry (w,w2,etc.) will be listed as "on", however some contributions to the dynamics need are iteration-dependent and need to be disabled. Alternatively, if the parent entry is listed as "off" then all dependent contributions are automatically disabled and listed as "-". The "*" indicates dynamics conditional on the choice of control field (g*: on if using applied vorticity control OR α*: on if using active stress control). All enabling/disabling of physics is handled in the functions: **func_config_forward.m**, **func_config_adjoing.m**, **func_config_update.m**, and **func_config_cost.m**.

[1] Mikhail, matlab-ascii-plot, (https://github.com/kyak/matlab-ascii-plot/releases/tag/0.1.1), GitHub. Retrieved June 1, 2020.

[2] R. R. Kerswell, C. C. Pringle, and A. P. Willis, *An optimization approach for analysing nonlinear stability with transition to turbulence in fluids as an exemplar,* Reports on Progress in Physics **77**, 085901 (2014), arXiv:1408.3539.

[3] A. M. Bradley, PDE-constrained optimization and the adjoint method, (https://cs.stanford.edu/∼ambrad/adjoint_tutorial.pdf), Retrieved September 5, 2020

[4] M. Bangert, Optimization tutorial, (https://www.mathworks.com/matlabcentral/fileexchange/34835-optimization-tutorial), MATLAB Central File Exchange. Retrieved May 1, 2020. .