**Online Appendix to Accompany:**

# An Introduction to Differential Games for Dynamic Modelers

This online appendix provides instructions for replicating the results reported in the paper, instructions for using the accompanying code for solving new problems, and a solution to the exercise provided at the end of the paper.

1. **Instructions for replicating the results**

All models and algorithmic steps are fully documented in the paper and thus the results should be feasible to replicate without these instructions. Nevertheless, the instructions in this document provide step-by-step guidelines to replicate the results with minimum additional effort.

Note that the results reported in the paper are produced using Matlab (R2010b) and Vensim DSS (V6.0). Both software packages are proprietary; however some of the results could be directly replicated in free software packages. Specifically, the Vensim models (.mdl files) could be opened and simulated using free Vensim model reader available from [www.Vensim.com](http://www.Vensim.com). Unfortunately, the optimization processes in Vensim cannot be conducted in a free software package based on the code we offer here. But reproducing the code in a different package (e.g., R or Octave) is rather straightforward. Most (but not all) Matlab commands can be run in the free environment Octave, available at: <http://www.gnu.org/software/octave/>. The most notable exception is the call to the matlab function fminunc, which is part of the Optimization Toolbox. However, it can be replaced by a call to any other function that provides an algorithm for unconstrained minimization.

**Replicating the results in section “A Simple Example”**

The file “*SimpleExampleDocumented.m*” implements the model for the example and graphs the Figures 2-a and 2-b in the Matlab environment. With small modifications you will be able to run these commands using the Octave environment as well. Once you copy the file in one of the directories on the command path for Matlab (or Octave), you can simply type the name of the file (without “.m” at the end) in the command screen and the graphs in Figure 2 will be created. The code is commented with details on the implementation. Note that the blue arrows showing the example/hypothetical iterations of the algorithm to reach the Nash equilibrium (Figure 2-b) are added by hand on top of the Matlab graph and will not be generated using the Matlab code.

**Replicating the results in section “Algorithm for Solving Differential Games”**

The following steps specify the process for replicating the computational results of the example differential game reported in the paper. To replicate the results quickly, you can simply put all the attached files in the same folder and run the command script “BaseDifferentialGame-30.cmd” in Vensim DSS. In Matlab (or Octave) you can run the file twoFirms.m. For clarity, we provide more details on the implementation following the same logical steps outlined in the paper.

**Step 1- Building a model of system’s dynamics**

**Vensim:** The model reported in the paper is available as “*FinalImplementation-TwoFirms-Subs.mdl*” in the accompanying package. You can open and simulate the model using free Vensim Model Reader, or Vensim DSS software. Note that the payoff variables for the two firms are identified as “Payoff” and “Payoff 2”. The action taken by each firm is specified as “FractionAllocated U1” and “FractionAllocated U2”.

**Matlab:** The corresponding files are yDot.m and obj.m, which respectively implement the derivatives of equations (5)-(9) and (10). Each firm’s total payoff is then the solution of the corresponding differential equation evaluated at the final time, and this is reflected in the code.

**Step 2- Iteratively solve the policy optimization problem to find the game’s equilibrium.**

**Step 2-A: Specify a set of features**

**Vensim:** Based on the available information set for each player (in this case the four stock variables and time), we have defined features which normalize the stocks. These are found in the following variables of the model: “NormCapUD”, “NormalizedCapability”, “NormCapUD 2”, “NormalizedCapability 2”, and normalized time calculated as Time/Final Time. These features are brought together as a vector variable, “FeatureSet F” subscripted with dimension “Feature”. Note that to use both first and second order combinations of these features in the policy function (e.g. both “NormCap UD” and “NormCap UD\*NormalizedCapability”), the variable “FeatureSet F” is augmented by a 6th member, a constant term equal to one.

**Matlab:** The set of features corresponds to the variable XT (“X transpose”) in the code twoFirms.m, where X is the column vector [1; C1; C2; D1; D2; t].

**Step 2-B: Define the functional form for policy function**

**Vensim:** This step is implemented in the variables “FractionAllocated U1” and “FractionAllocated U2” and offers a logistic function that calculates the fraction allocated based on a second order function of the features above. The matrix of parameters that specify the numerical instances for this function are B1 and B2.

**Matlab:** In twoFirms.m, B1 and B2 are concatenated vertically to form a matrix called B. In this program, it is sometimes convenient to work with a vector containing only the (generally) non-zero elements of B, which is a matrix consisting of upper-triangular submatrices. The translation from a vector to an upper-triangular matrix is achieved by the function vec2ut.m.

**Step 2-C: Vectorize the implementation of the model**

**Vensim:** This step is achieved in Vensim DSS using the stochastic optimization option, which replicates the model *N* times and calculates the specified payoff summed over all these replications. To implement stochastic optimization we identify in the “Sim Control” dialogue box a vensim sensitivity control (.vsc) file that specifies the initial values for different stock variables in the *N*=100 parallel competitions we will conduct our study over. The file “*Sens.vsc*” specifies the parameters of these initial distributions, also available in the paper. A vensim save list (.lst) file is also required when working with sensitivity analysis files, which is available as “*SensSave.lst*” in the package. Vensim will use the stochastic optimization option if it detects the keyword “:STOCHASTIC” in the optimization control file (.voc), specified in the next step. One could alternatively achieve the same result by explicitly defining a new subscript for parallel optimizations and calculating the payoff as a sum of payoffs for those individual optimizations.

**Matlab:** In twoFirms.m, the computations are not performed in parallel because that would involve requiring even more functionality beyond standard Matlab. However, the optimization using 100 random initial conditions is performed by setting the flag IC100 = 1. Any other setting of IC100 indicates that the code is to perform the optimization over the specific set of initial conditions in Table 2.

**Step 2-D: Specify algorithmic parameters**

**Vensim:** The algorithmic parameters for the optimization are embedded in different files that are attached here. Specifically, the number of parallel optimizations, *N*, is specified in the sensitivity analysis file that is used for stochastic optimization by Vensim. In “*Sens.vsc*” we used the value of *N*=100. The starting policy function parameters for the second player is specified as B2=0 in the Vensim model. A more intelligent initialization could shorten the iteration time in step 2-E but we did not attempt to use more complicated initialization methods. The optimization parameters are all specified in the optimization control file used in the optimization steps (2-E). Five restarts were used in each optimization to build confidence in the generality of the optimization results, though this may not be enough to ensure global optimum. Two .voc files are used for the two players, these are called “*Firm1.voc*” and “*Firm2.voc*” and include the parts of the B1 and B2 parameter matrices that are included in the optimization (See footnote 7 in the paper), leaving the rest of the (redundant) parameters at zero.

**Matlab:** The algorithmic parameters for the optimization in Matlab are contained in twoFirms.m. No systematic exploration of these is reported.

**Step 2-E: Iteratively find the solution to the game**

**Vensim:** The command file “*BaseDifferentialGame-30.cmd*” provides the script that automates the optimization iterations in Vensim. In fact all you need to do to replicate the results is to leave all the attached files in the same folder, open this command file in Vensim DSS (using “Edit File” command under File menu), and select “Run Commands” from the File menu in Vensim.

This file will run 30 consecutive optimizations iteratively, starting the next optimization based on the best policy parameters (B matrix) found for the other player in the previous step. This is equal to 15 full iterations, as discussed in the paper. Your application may need fewer or more iterations. It is rather easy to extend this file to include more iterations, either by adding code at the end, or rerunning the same command file after adding code to read the initial policy for the iteration 1 (optimizations 1 and 2) from the .out file in iteration 15 (optimizations 29 and 30). Therefore you can use this command script for your own applications with a few modifications.

The stopping of iterations is not automated in this Vensim implementation. You should therefore inspect the difference between the resulting policy parameters in two consecutive iterations (B matrices) occasionally to detect convergence.

Note that each .out files, which includes the policy parameters from the previous run and is used as input to the next optimization, is generated by the optimization in the previous step; therefore, you will not need any .out file to start the algorithm. Two intermediate .out files that include the optimized policy parameters for firms 1 and 2 after 4 iterations (“*base7.out*” and “*base8.out*”) are included in the package to facilitate experimentation.

The 15 iterations coded in the script above are sufficient to converge in the example provided in the paper and with the optimization parameters used. Once convergence is achieved, you can extract the optimum policy by looking at the values of the B1 and B2 parameters in the model found/used in the last iteration file (e.g. “*base30.vdf”*), or in the last two .out files (“*base29.out*” and “*base30.out*” for players 1 and 2 respectively). These numerical values are shown in Table 3 in the paper.

**Matlab:** Due to the local nature of the optimization and the likelihood that Matlab and Vensim codes find different local optima, the Matlab code converges to a similar-looking but different optimal solution. Specific results are not given. The initialization of B is easy to change in the code for the purposes of experimentation.

**Step 2-F: Simulating the resulting equilibrium competition**

**Vensim:** After conducting the steps above, you can conduct a final simulation, reading the parameter values for B1 and B2 from the last two .out files from the optimization (e.g. (“*base29.out*” and “*base30.out*”) and observing the resulting trajectory of the competition. You can read these initial parameter values in Vensim by going to “Sim Control” dialogue, “Changes” tab, and add the two .out files in front of “Load changes from…”. You could also enter by hand the optimum parameter values into the equations for parameters B1 and B2. In fact, you can use the “*base7.out*” and “*base8.out*” files attached to simulate results very close to those reported in Figure 5 (for exact results you need to continue a few rounds more of iterative optimization, or use the parameters reported in Table 3 to initialize B1 and B2).

**Matlab:** To perform the analogous operation in twoFirms.m, the B that was the result of the optimization over 100 random initial conditions can be read in as the initial guess and the optimization omitted.

1. **Using the accompanying code**

You can use the accompanying code for your own analysis of differential games. To do so, you need to develop your own model of the strategic competition of interest (to replace the files *“FinalImplementation-TwoFirms-Subs.mdl”* or yDot.m and obj.m, respectively, in the steps above). You should specify the payoffs, information sets, features, and functional form of the optimum policy in these files. You can then edit the .voc files or twoFirms.m to replace B1 and B2 with the parameters of the policy function in the new model, change the number of restarts for each optimization (currently we use five restarts), and potentially change other optimization parameters. In the .vpd files, replace “Payoff” and “Payoff 2” with the parallel concepts in your new model (you could also keep the same variable names for the payoffs in the Vensim model and not change the .vpd files). You should edit the sensitivity control, “*Sens.vsc*”, to reflect the initial conditions you want to vary in the parallel competitions, and set the number of those parallel competitions, *N*. Also edit the save list, “*SensSave.lst*”, to include the variables you want to save during optimization for diagnostics. In editing the .voc, .vpd, and .vsc files, do not change the names of the files, so that you can use the attached command script. Finally, you can change the third line of the command script, “SPECIAL>LOADMODEL|FinalImplementation-TwoFirms-Subs.mdl”, to load your new model, rather than the one described in the paper. To build on the Matlab files provided, it is advisable to rename twoFirms.m to sometime more appropriate to the new problem and carefully edit the parts that must be modified or are now irrelevant. You should now be ready to run the commands and solve the differential game you have specified.

1. **Solution to the exercise**

The files attached in the folder “ExerciseSolution” include one possible solution to the exercise outlined in the paper. We go through different steps of the solution below.

**Step 1:** The model “SimpleExample-singledimfeature.mdl” includes the model for this exercise along with features and payoffs for each player. The model is developed based on the description in the main paper. We assume the game is Markovian and thus the information set available to each player includes the value of stocks and the current time.

**Steps 2-A and 2-B**: In this solution, we use a logistic function that maps a linear combination of different state variables, and (normalized) time, into an allocation fraction between 0 and 1 for each player. Specifically, defining as the vector of policy parameters for firm *i* (with subscripted elements *f1*…*f4*), we get the following functional form for the policy function:

Note that the choice of logistic function, and linear combination of states, is only based on computational efficiency and conceptual clarity, it is likely that alternative functional forms (and feature sets) could be found that produce solutions that are closer to the true Nash equilibrium.

**Steps 2-C**: The vectorization of the competitions is achieved through using Vensim’s stochastic optimization function. The file Sens.vsc provides the details of this implementation, where 20 initial positions in the state space are randomly generated from independent uniform distributions for the two firms ranging between 0 and .

**Step 2-D**: N is assumed to be 20, and specified in “*Sens.vsc*”. Second firm starts the iterations from the naïve policy of B2=0. We conducted 10 restarts for each optimization experiments.

**Step 2-E:** Conducting the iterative optimization using the modified “*BaseDifferentialGame-30.cmd*”, we converged to a stable solution after 15 iterations (30 optimizations), though a few more iterations would have reduced the convergence threshold from approximately 14% achieved here. We note that the 10 restarts are not all converging to the same peak in the optimizations, and therefore using a larger number of restarts (or increasing N to get better convergence) is warranted. The optimum B vectors found for the two firms are B1=[11.1, -5.9, 0.087, 0.86] and B2=[119.1, -100.3, 1.71, -22.0]. The .out files reporting these solutions are attached (“*base29.out*” and “*base30.out*”).

**Step 2-F**: Simulating the resulting game using the optimum allocation policies found above, we get the following behaviors for the allocation fraction, quality, and payoff (only results for the first 20 periods are shown in Figure S1, as steady state is reached and nothing changes afterwards). The first firm allocates more to its quality early on to catch up its quality value, and despite higher steady state investments than the second firm, achieves lower quality and payoff due to its shorter quality life parameter.

Note that this solution outperforms the constant policy of Ui=0.45 that we found through the analytical method described in the paper (Figure S2), in the sense that both firms get higher payoffs and are at equilibrium given the policy structure assumed in this case. The steady-state allocation for both firms is under 0.45 in this case. The constant policy was derived assuming that the two firms could not observe any information, and thus acting in the dark, had to make a single allocation decision and stick to it. However, with the information about their own quality and those of the competition, both firms can waste less resource on quality, and thus gain higher performance.



Payoff

Quality

Fraction Allocated

**Figure S1- The behavior of the two firms in the game solution in the exercise.**

****

Payoff

**Figure S2- Comparison of constant allocation policy and the game solution derived above.**