Module 4A - Genetic Algorithms

Omar Betancourt, Payton Goodrich, Emre Mengi

April 2021

Contents

Objectives: Learn the theory and practice of using one of the most commonly used machine learning algorithms.
Prerequisite Knowledge: N/A
Prerequisite Modules: 1A - Calculus, 1B - Linear Algebra, 1D - Differential Equations, 2B - Continuum Mechanics, 3C - Generic Time Stepping, 4B - Gradient-based Optimization
Difficulty: Intermediate
Summary: A spark sintering process is used as an example for applying a simple genetic algorithm for process optimization.

1 Theory

Genetic algorithms are extremely useful for multicomponent system optimization where gradient-methods are not preferred due to nonconvex, nonsmooth nature of the system. The rapid rate at which the simulations explored in the food system applications modules can be completed enables the ability to explore inverse problems seeking to determine what parameter combinations can deliver a desired result (Figure 1.1). In order to cast the objective mathematically, we set the problem up as a Machine Learning Algorithm (MLA); specifically a Genetic Algorithm (GA) variant, which is well-suited for nonconvex optimization. Following Zohdi [7, 8, 9, 10], we formulate the objective as a cost function minimization problem that seeks system parameters that match a desired response

$$\Pi^{tot}(\lambda_1, \ldots \lambda_{13}) = \text{LOSSES} \tag{1.1}$$

We systematically minimize Equation 1.1, $\min_{\Lambda} \Pi$, by varying the design parameters: $\Lambda^i \stackrel{\text{def}}{=} \{\Lambda_1^i, \Lambda_2^i, \Lambda_3^i, \ldots, \Lambda_N^i\}$ $\stackrel{\text{def}}{=}$ { flow rate, aircraft dynamics... }. The system parameter search is conducted within the constrained ranges of $\Lambda_1^{(-)} \leq \Lambda_1 \leq \Lambda_1^{(+)}, \Lambda_2^{(-)} \leq \Lambda_2 \leq \Lambda_2^{(+)}$ and $\Lambda_3^{(-)} \leq \Lambda_3 \leq \Lambda_3^{(+)}$, etc. These upper and lower limits would, in general, be dictated by what is physically feasible.

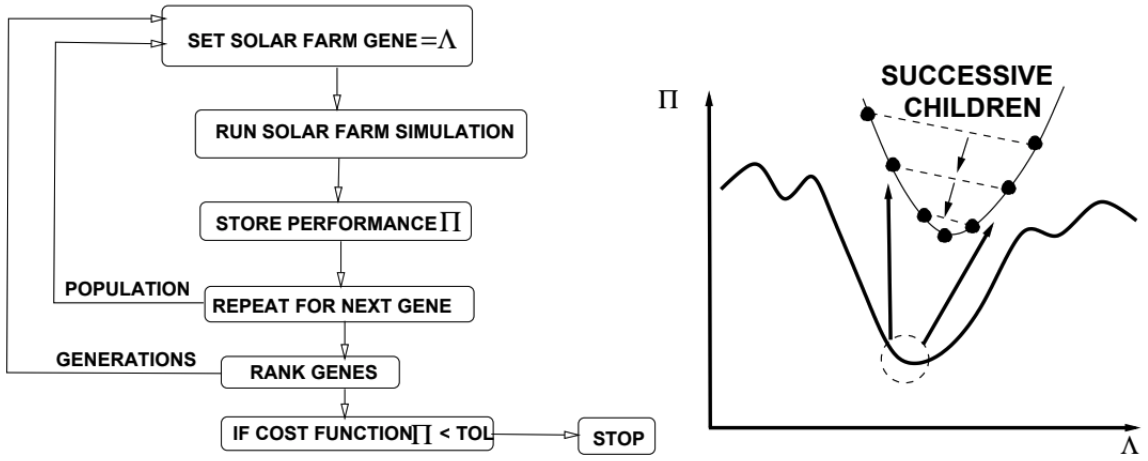

Figure 1.1: The basic action of a MLA/GA-Machine Learning Algorithm/Genetic Algorithm. Zohdi [7, 8, 9, 10] .

System parameter search: Machine Learning Algorithm (MLA)

Here we follow Zohdi [7, 8, 9, 10] in order to minimize Equation 1.1, which we will refer to as a "cost function". Cost functions such as Equation 1.1 are nonconvex in design parameter space and often nonsmooth. Their minimization is usually difficult with direct application of gradient methods. This motivates nonderivative search methods, for example those found in Machine Learn- ing Algorithms (MLA's). One of the most basic subsets of MLA's are so-called Genetic Algorithms (GA's). Typically, one will use a GA first

in order to isolate multiple local minima, and then use a gradient-based algorithm in these locally convex regions or reset the GA to concentrate its search over these more constrained regions. GA's are typically the simplest scheme to start the analysis, and one can, of course, use more sophisticated methods if warranted. For a review of GA's, see the pioneering work of John Holland (1, 2]), as well as Goldberg [3], Davis [4], Onwubiko [5] and Goldberg and Deb [6].

Generalities

The MLA/GA approach is extremely well-suited for nonconvex, nonsmooth, multicomponent, multistage systems and, broadly speaking, involves the following essential concepts:

1. **POPULATION GENERATION:** Generate a parameter population of genetic strings: $\mathbf{\Lambda}^i$
2. **PERFORMANCE EVALUATION:** Compute performance of each genetic string: $\Pi\left(\mathbf{\Lambda}^i\right)$
3. **RANK STRINGS:** Rank them $\mathbf{\Lambda}^i, i = 1, \ldots, N$
4. **MATING PROCESS:** Mate pairs/produce offspring
5. **GENE ELIMINATION:** Eliminate poorly performing genetic strings
6. **POPULATION REGENERATION:** Repeat process with updated gene pool and new random genetic strings
7. **SOLUTION POST-PROCESSING:** Employ gradient-based methods afterwards in local "valleys"- *if smooth enough*

Specifics

Following Zohdi [7, 8, 9, 10], the algorithm is as follows:

- **STEP 1:** Randomly generate a population of S starting genetic strings, $\mathbf{\Lambda}^i, (i = 1, 2, 3, \ldots, S)$: $\mathbf{\Lambda}^i \stackrel{\text{def}}{=} \left\{\Lambda_1^i, \Lambda_2^i, \Lambda_3^i, \Lambda_4^i, \ldots, \Lambda_N^i\right\}$
  where

$$\underline{\Lambda}^{(i)} = \begin{pmatrix} \Lambda_1^- \leq \Lambda_1^{(i)} \leq \Lambda_1^+ \\ \Lambda_2^- \leq \Lambda_2^{(i)} \leq \Lambda_2^+ \\ \Lambda_3^- \leq \Lambda_3^{(i)} \leq \Lambda_3^+ \\ \Lambda_4^- \leq \Lambda_4^{(i)} \leq \Lambda_4^+ \\ \ldots \\ \Lambda_N^- \leq \Lambda_N^{(i)} \leq \Lambda_N^+ \end{pmatrix}$$

- **STEP 2:** Compute fitness of each string $\Pi\left(\mathbf{\Lambda}^i\right), (\mathrm{i} = 1, \ldots, \mathrm{S})$
- **STEP 3:** Rank genetic strings: $\mathbf{\Lambda}^i, (\mathrm{i} = 1, \ldots, \mathrm{S})$
- **STEP 4 :** Mate nearest pairs and produce two offspring, $(\mathrm{i} = 1, \ldots, \mathrm{S})$: $\boldsymbol{\lambda}^i \stackrel{\text{def}}{=} \phi_i \cdot \mathbf{\Lambda}^i + (1 - \phi_i) \cdot \mathbf{\Lambda}^{i+1}$, and $\boldsymbol{\lambda}^{i+1} \stackrel{\text{def}}{=} \hat{\phi}_i \cdot \mathbf{\Lambda}^i + \left(1 - \hat{\phi}_i\right) \cdot \mathbf{\Lambda}^{i+1}$, where for this operation, ϕ_i and $\hat{\phi}_i$ are random numbers, such that $0 \leq \phi_i \leq 1, 0 \leq \hat{\phi}_i \leq 1$, which are different for each component of each genetic string.

$$\underline{\Lambda}^{(ci)} = \begin{pmatrix} \Lambda_1^{pi}\phi_1 + \Lambda_1^{p(i+1)}(1-\phi_1) \\ \Lambda_2^{pi}\phi_2 + \Lambda_2^{p(i+1)}(1-\phi_2) \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \Lambda_N^{pi}\phi_N + \Lambda_N^{p(i+1)}(1-\phi_N) \end{pmatrix}$$

where $\phi_i \in$ rand[0,1].

$$\underline{\Lambda}^{(c(i+1))} = \begin{pmatrix} \Lambda_1^{p(i+1)}\hat{\phi}_1 + \Lambda_1^{pi}(1-\hat{\phi}_1) \\ \Lambda_2^{p(i+1)}\hat{\phi}_2 + \Lambda_2^{pi}(1-\hat{\phi}_2) \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \Lambda_N^{p(i+1)}\hat{\phi}_N + \Lambda_N^{pi}(1-\hat{\phi}_N) \end{pmatrix}$$

where $\hat{\phi}_i \in$ rand[0,1].

- **STEP 5:** Eliminate the bottom $M < S$ strings and keep top $K < N$ parents and top K offspring (K offspring $+K$ parents $+M = S$)
- **STEP 6:** Repeat STEPS $1-6$ with top gene pool (K offspring and K parents), plus M new, randomly generated, strings
- **IMPORTANT OPTION:** Rescale and restart search around best performing parameter set every few generations

Remark 4 : If one selects the mating parameter Φ to be greater than one and/or less than zero, one can induce "mutations", i.e. characteristics that neither parent possesses. However, this is somewhat redundant with introduction of new random members of the population in the current algorithm.

Remark 5 : If one does not retain the parents in the algorithm above, it is possible that inferior performing offspring may replace superior parents. Thus, top parents should be kept for the next generation. This guarantees a monotone reduction in the cost function. Furthermore, retained parents do not need to be re-evaluated-making the algorithm less computationally less expensive, since these parameter sets do not have to be reevaluated (or ranked) in the next generation. Numerous studies of the author have shown that the advantages of parent retention outweighs inbreeding, for sufficiently large population sizes. Finally, we remark that this algorithm is easily parallelizable.

Remark 6: After application of such a global search algorithm, one can apply a gradient-based method, if the objective function is sufficiently smooth in that region of the parameter space. In other words, if one has located a convex portion of the parameter space with a global genetic search, one can employ gradient-based procedures locally to minimize the objective function further, since they are generally much more efficient for convex optimization of smooth functions. An exhaustive review of these methods can be found in the texts of Luenberger [11] and Gill, Murray and Wright [12].

2 Example

Introduction

The goal is to utilize genetic algorithm to design a material using desired mechanical, electrical, and thermal properties. Genetic algorithm is a type of machine learning algorithm that allows the user to apply optimization to even non-convex systems. Such a material selection process can be used in early-stages of a composite material design for an engineering system.

Objectives

The project includes calculating the mechanical, thermal, and electrical properties of the resulting material from mixing two materials. Hashin-Shtrikman bounds are used to calculate effective properties of the composite material and a multi-property genetic algorithm optimization is applied to find the optimal parameters for the second material while the properties of the first material are known.

Theory and Procedure

First, the model parameters are set. Then, the required relations for the mechanical, thermal, and electrical properties (with an isotropic response) are derived before a GA is applied:

The HS bounds for the effective bulk modulus of such a mixture is given by:

$$\kappa^{*,-}\kappa_1 + \frac{v_2}{\frac{1}{\kappa_2-\kappa_1}+\frac{3(1-v_2)}{3\kappa_1+4\mu_1}} \leq \kappa^* \leq \kappa_2 + \frac{1-v_2}{\frac{1}{\kappa_1-\kappa_2}+\frac{3v_2}{3\kappa_2+4\mu_2}}\kappa^{*,+}, \tag{2.1}$$

The analogous expression for the shear modulus is:

$$\mu^{*,-}\mu_1 + \frac{v_2}{\frac{1}{\mu_2-\mu_1}+\frac{6(1-v_2)(\kappa_1+2\mu_1)}{5\mu_1(3\kappa_1+4\mu_1)}} \leq \mu^* \leq \mu_2 + \frac{(1-v_2)}{\frac{1}{\mu_1-\mu_2}+\frac{6v_2(\kappa_2+2\mu_2)}{5\mu_2(3\kappa_2+4\mu_2)}}\mu^{*,+}, \tag{2.2}$$

where κ_2 and κ_1 are the bulk moduli and μ_2 and μ_1 are the shear moduli of the respective phases ($(\kappa_2 \geq \kappa_1)$ and $(\mu_2 \geq \mu_1)$), and where v_2 is the second phase volume fraction. Using convex combinations of the HS bounds as approximations for the effective moduli $\kappa^* \approx \phi\kappa^{*+} + (1-\phi)\kappa^{*-}$ and $\mu^* \approx \phi\mu^{*+} + (1-\phi)\mu^{*-}$, where $\phi = 0.5$.

We can also use the HS bounds for the overall electrical and thermal conductivity σ_e^* and *

$$\langle\sigma_e^{-1}(\mathbf{x})\rangle_{\mathbf{\Omega}}^{-\mathbf{1}} \leq \underbrace{\sigma_{1,e} + \frac{v_2}{\frac{1}{\sigma_{2,e}-\sigma_{1,e}}+\frac{1-v_2}{3\sigma_{1,e}}}}_{\sigma_{\mathbf{e}}^{*,-}} \leq \sigma_{\mathbf{e}}^* \leq \underbrace{\sigma_{2,e} + \frac{1-v_2}{\frac{1}{\sigma_{1,e}-\sigma_{2,e}}+\frac{v_2}{3\sigma_{2,e}}}}_{\sigma_{\mathbf{e}}^{*,+}} \leq \langle\sigma_{\mathbf{e}}(\mathbf{x})\rangle_{\mathbf{\Omega}} \tag{2.3}$$

The analogous expression for the effective thermal conductivity is:

$$\underbrace{_1 + \frac{v_2}{\frac{1}{_2-_1}+\frac{1-v_2}{3_1}}}_{^{*,-}} \leq ^* \leq \underbrace{_2 + \frac{1-v_2}{\frac{1}{_1-_2}+\frac{v_2}{3_2}}}_{^{*,+}} \tag{2.4}$$

The mismatch of the combined material properties can lead to inefficiency and locally elevated mechanical, thermal, or electrical loads. Therefore, concentration factors are defined to provide a design limitation for the composite material:

The load carried by each phase in the microstructure is characterized by stress and strain concentration tensors. In the case of an isotropic material, we can write:

$$C_\kappa^{\sigma,2}\frac{1}{v_2}\frac{\kappa_2}{\kappa^*}\frac{\kappa^*-\kappa_1}{\kappa_2-\kappa_1} \qquad \text{and} \qquad C_\mu^{\sigma,2}\frac{1}{v_2}\frac{\mu_2}{\mu^*}\frac{\mu^*-\mu_1}{\mu_2-\mu_1} \tag{2.5}$$

$$C_{\kappa}^{\sigma,1}\tfrac{1}{v_1}(1-v_2C_{\kappa}^{\sigma,2}) \qquad \text{and} \qquad C_{\mu}^{\sigma,1}\tfrac{1}{v_1}(1-v_2C_{\mu}^{\sigma,2}). \tag{2.6}$$

A key quantity of interest for many industrial applications is the amount of heat generated from running a current through a material, denoted H. We can determine the phase-wise load-shares of the Joule field, denoted $H = \mathbf{J}\cdot\mathbf{E}$, carried by the components in the heterogeneous mixture. For a two-phase mixture, the product of the concentration factors take the following form:

$$C_{E,1}C_{J,1} = \frac{\sigma_{e,1}}{\sigma_e^*}\left(\frac{1}{(1-v_2)}\left(\frac{\sigma_{e,2}-\sigma_e^*}{\sigma_{e,2}-\sigma_{e,1}}\right)\right)^2 \tag{2.7}$$

$$C_{E,2}C_{J,2} = \frac{\sigma_{e,2}}{\sigma_e^*}\left(\frac{1}{v_2}\left(\frac{\sigma_e^*-\sigma_{e,1}}{\sigma_{e,2}-\sigma_{e,1}}\right)\right)^2. \tag{2.8}$$

For thermal properties, use the following concentration tensors:

- ${}_{\theta,2}\cdot\langle\nabla\theta\rangle_\Omega = \langle\nabla\theta\rangle_{\Omega_2}$ where ${}_{\theta,2} = \frac{1}{v_2^K}({}_2{-}_1)^{-1}\cdot({}^*{-}_1)$
- ${}_{\theta,1}\cdot\langle\nabla\theta\rangle_\Omega = \langle\nabla\theta\rangle_{\Omega_1}$ where ${}_{\theta,1} = \frac{1}{v_1^K}(1-v_2^K\,{}_{\theta,2}) = \frac{1-v_2^K\,{}_{\theta,2}}{1-v_2^K}$
- ${}_{q,2}\cdot\langle\rangle_\Omega = \langle\rangle_{\Omega_2}$ where ${}_{q,2} = {}_2\cdot{}_{\theta,2}\cdot{}^{*-1}$
- ${}_{q,1}\cdot\langle\rangle_\Omega = \langle\rangle_{\Omega_1}$ where ${}_{q,1} = \frac{\mathbf{1}-v_2^K\,{}_{q,2}}{1-v_2^K}$

With many separate goals of the project defined, all the goals must be combined into a single scalar value to be minimized by the genetic algorithm. The cost function defined for mechanical, electrical, and thermal properties are:

$$\boxed{\begin{aligned}\Pi^{mechanical} &= w_1\left|\tfrac{\kappa^{*,Des}-\kappa^*}{\kappa^{*,Des}}\right| + w_1\left|\tfrac{\mu^{*,Des}-\mu^*}{\mu^{*,Des}}\right| + \hat{w}_3\left|\tfrac{C_\kappa^{\sigma,2}-TOL_\kappa}{TOL_\kappa}\right| + \hat{w}_4\left|\tfrac{C_\mu^{\sigma,2}-TOL_\mu}{TOL_\mu}\right| \\ &+\hat{w}_5\left|\tfrac{C_\kappa^{\sigma,1}-TOL_\kappa}{TOL_\kappa}\right| + \hat{w}_6\left|\tfrac{C_\mu^{\sigma,1}-TOL_\mu}{TOL_\mu}\right|\end{aligned}}$$

where:

$$\hat{w}_3 = \begin{cases} w_3 & C_\kappa^{\sigma,2} > TOL_\kappa \\ 0 & \text{otherwise}\end{cases} \tag{2.9}$$

$$\hat{w}_4 = \begin{cases} w_4 & C_\mu^{\sigma,2} > TOL_\mu \\ 0 & \text{otherwise}\end{cases} \tag{2.10}$$

$$\hat{w}_5 = \begin{cases} w_5 & C_\kappa^{\sigma,1} > TOL_\kappa \\ 0 & \text{otherwise}\end{cases} \tag{2.11}$$

$$\hat{w}_6 = \begin{cases} w_6 & C_\mu^{\sigma,1} > TOL_\mu \\ 0 & \text{otherwise}\end{cases} \tag{2.12}$$

$$\boxed{\Pi^{elec} = w_1\left|\tfrac{\sigma_e^{*,Des}-\sigma_e^*}{\sigma_e^{*,Des}}\right| + \hat{w}_2\left|\tfrac{C_{J1}C_{E1}-TOL_\sigma}{TOL_\sigma}\right| + \hat{w}_3\left|\tfrac{C_{J2}C_{E2}-TOL_\sigma}{TOL_\sigma}\right|,} \tag{2.13}$$

where:

$$\hat{w}_2 = \begin{cases} w_2 & \frac{C_{J1}C_{E1}-TOL_\sigma}{TOL_\sigma} > 0 \\ 0 & \text{otherwise}\end{cases} \tag{2.14}$$

$$\hat{w}_3 = \begin{cases} w_3 & \frac{C_{J2}C_{E2}-TOL_\sigma}{TOL_\sigma} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

$$\boxed{\Pi^{thermal} = w_1 |\frac{^{*,Des}-^*}{^{*,Des}}| + \hat{w}_2 |\frac{C_{q1}-TOL}{TOL}| + \hat{w}_3 |\frac{C_{q2}-TOL}{TOL}|} \tag{2.16}$$

where:

$$\hat{w}_2 = \begin{cases} w_2 & \frac{C_{q1}-TOL}{TOL} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

$$\hat{w}_3 = \begin{cases} w_3 & \frac{C_{q2}-TOL}{TOL} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

We can then combine the loss function to obtain the overall fitness of the design strings. The overall objective function is:

$$\boxed{\Pi^{total} = W_1\Pi^{electrical} + W_2\Pi^{thermal} + W_3\Pi^{mechanical}} \tag{2.19}$$

We consider a base matrix material of fixed material values $(\kappa_1, \mu_1, \sigma_{e,1,1})$. Using $\phi = 0.5$ for estimating the effective material properties from HS bounds, we utilize a genetic algorithm to find the optimal values for the second phase material parameters as well as the optimal volume fraction required to achieve desired material properties. For this genetic algorithm, the design string is:

$$\mathbf{\Lambda}^i = \{\kappa_2, \mu_2, \sigma_{e,2,2}, v_2\} \tag{2.20}$$

For the genetic algorithm, we consider three cases:

- Keeping the top $P = 10$ parents after each generation (10 parents, 10 offspring, 80 new random strings).
- Not keeping top $P = 10$ parents after each generation, (10 offspring, 90 new genetic strings).
- Keeping the top $P = 10$ parents after each generation, and generating new children with mutations. More specifically, instead of creating children using combinations of parents with random weights $0 \leq \phi \leq 1$, use $-.5 \leq \phi \leq 1.5$ instead. This allows for offspring to be generated outside of the hyperrectangle or "higher-dimensional box" defined by parent designs in design space.

Results

After the model has been constructed, the outputs are graphed:

Keeping the top $P = 10$ parents after each generation (10 parents, 10 offspring, 80 new random strings):

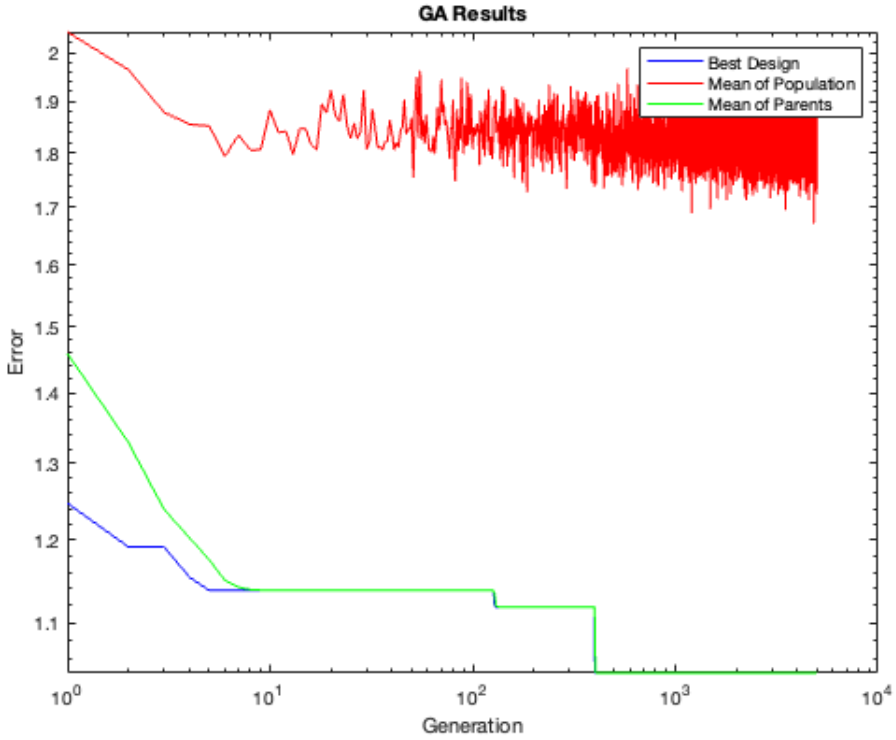

Figure 2.1: GA Results for Case 1 (Keep parents-no mutation).

Table 2.1: The top 4 system parameter performers for Case 1 (Keep parents-no mutation).

Design	κ_2	μ_2	$_{e,2}$	$_2$	υ_2	Π
1	137.5 GPa	59.3 GPa	3.31E+07 S/m	7.74 W/m-k	0.590	1.044
2	137.5 GPa	59.3 GPa	3.31E+07 S/m	7.74 W/m-k	0.590	1.044
3	137.5 GPa	59.3 GPa	3.31E+07 S/m	7.74 W/m-k	0.590	1.044
4	137.5 GPa	59.3 GPa	3.31E+07 S/m	7.74 W/m-k	0.590	1.044

Not keeping top $P = 10$ parents after each generation, (10 offspring, 90 new genetic strings):

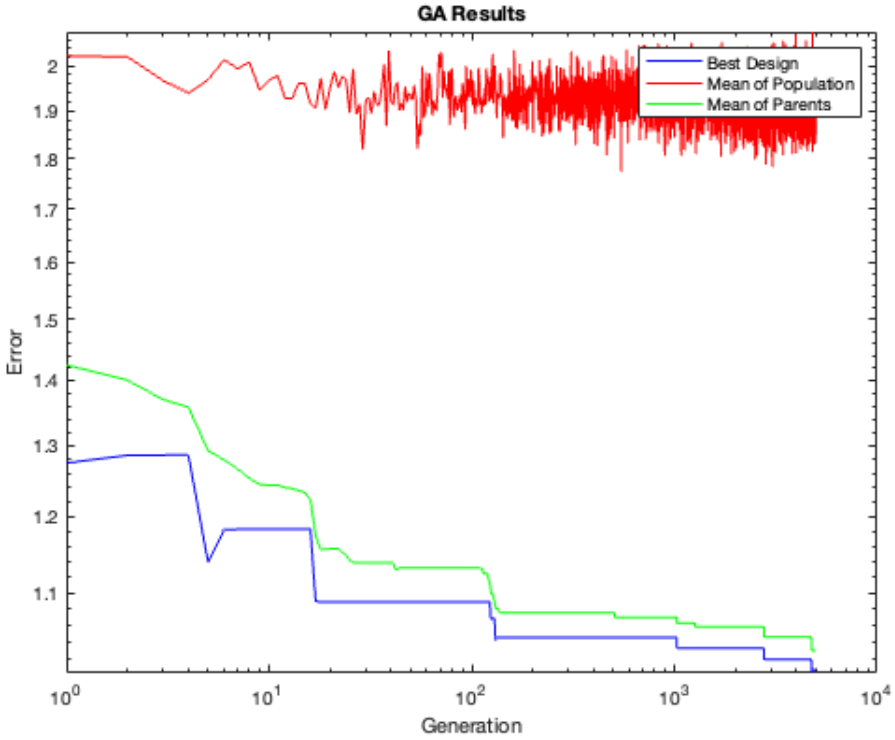

Figure 2.2: GA Results for Case 2 (Discard parents-no mutation).

Table 2.2: The top 4 system parameter performers for Case 2 (Discard parents-no mutation).

Design	κ_2	μ_2	$_{e,2}$	$_2$	υ_2	Π
1	141.1 GPa	65.0 GPa	2.95E+07 S/m	7.85 W/m-k	0.589	1.008
2	144.5 GPa	66.3 GPa	2.96E+07 S/m	7.90 W/m-k	0.584	1.008
3	135.7 GPa	64.4 GPa	2.88E+07 S/m	7.69 W/m-k	0.592	1.015
4	164.2 GPa	69.5 GPa	3.18E+07 S/m	8.46 W/m-k	0.566	1.015

Keeping the top $P = 10$ parents after each generation, and generating new children with mutations:

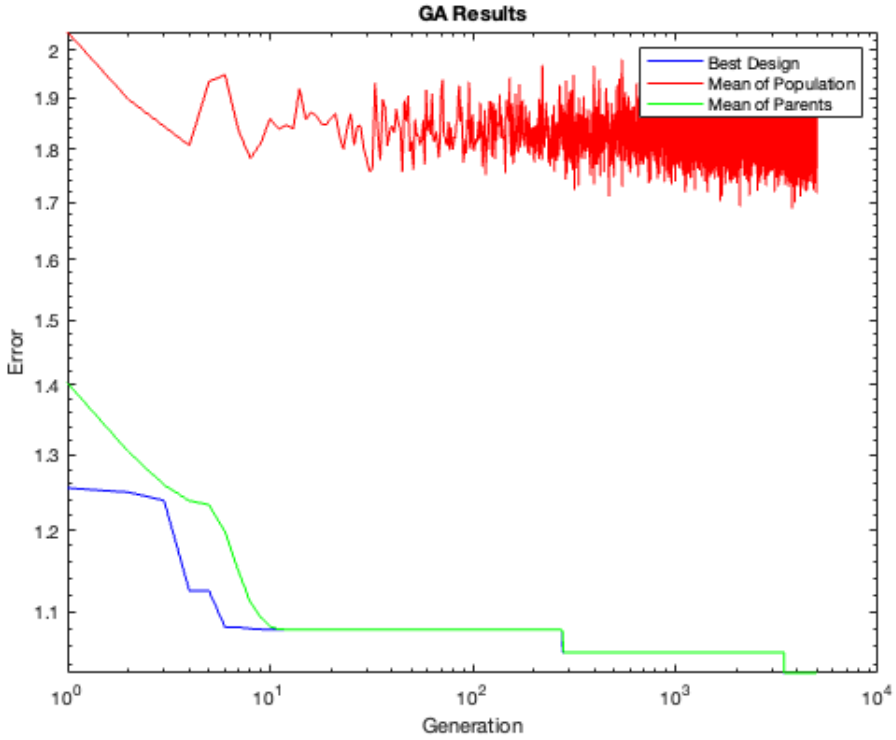

Figure 2.3: GA Results for Case 3 (Keep parents-with mutation).

Table 2.3: The top 4 system parameter performers for Case 3 (Keep parents-with mutation).

Design	κ_2	μ_2	$_{e,2}$	$_2$	v_2	Π
1	162.3 GPa	60.0 GPa	2.89E+07 S/m	7.03 W/m-k	0.634	1.031
2	162.3 GPa	60.0 GPa	2.89E+07 S/m	7.03 W/m-k	0.634	1.031
3	162.3 GPa	60.0 GPa	2.89E+07 S/m	7.03 W/m-k	0.634	1.031
4	162.3 GPa	60.0 GPa	2.89E+07 S/m	7.03 W/m-k	0.634	1.031

Observations and Discussion

Looking at Figures 2.1 - 2.3, a few comments can be made about the results. The genetic algorithm results when the parents were retained and no mutations were allowed shows the same design parameter values for the top 4 design strings, with a cost value of $\Pi = 1.044$. The cost decreases monotonically over generations as the parents are kept in the algorithm, preventing an increase in the cost function between generations. Around 10 generations, the mean cost of the parents match the best design cost.

For the second genetic algorithm where the parents are discarded while mutations are not allowed, we obtain a variety of design strings in the top 4, with a lowest cost value of $\Pi = 1.008$. The cost of these design strings are similar, meaning that it would be up to the engineer or scientist to decide on the optimal material selection as the second material of the composite. The cost function increases over generations in a few instances, but the error goes down during the overall GA run.

The third genetic algorithm where the parents are kept in the design string with mutations allowed has the same design string and cost for the top 4, with a cost of $\Pi = 1.031$. The mean cost of the parents match up

with the best design cost around 10th generation.

The difference in trends among the used genetic algorithms show that the second case has the lowest error after 5000 generations, while the first case has the highest error. However, the difference in the error margin among these three cases are too close to accurately judge the best genetic algorithm. Overall, the model constructed here can be used for fast preliminary analysis of a material design process before computationally expensive models are deployed for detailed analysis.

3 Assignment

GENETIC ALGORITHM FOR PARAMETER TUNING (100 POINTS)

Overview

In this project, you will use a genetic algorithm to find system parameters for a spark sintering process (starter code provided) that match a desired result.

Notation

This assignment involves a large number of symbols. In multiphysical problems with many parameters, there can be conflicts between conventionally-used symbols. The symbols used to represent a particular physical quantity will not always be consistent between assignments within this class to clarify conflicts within particular assignments. In this assignment, several symbol choices are likely to lead to confusion:

- The system temperature will be expressed as θ.
- The electrical conductivity will be expressed as σ^c.
- The Cauchy Stress will be expressed with the letter σ without a subscript.
- The Jacobian (determinant of deformation gradient F) will have symbol $\mathcal{J}$.
- Electrical current will have symbol J.
- Strains will have symbol E.
- Electrical field will have symbol $\mathcal{E}$.
- 1 (a bold number 1) represents the 3×3 identity matrix.

Key symbols are defined and explained in **the table on the last page**.

Overview and Introduction

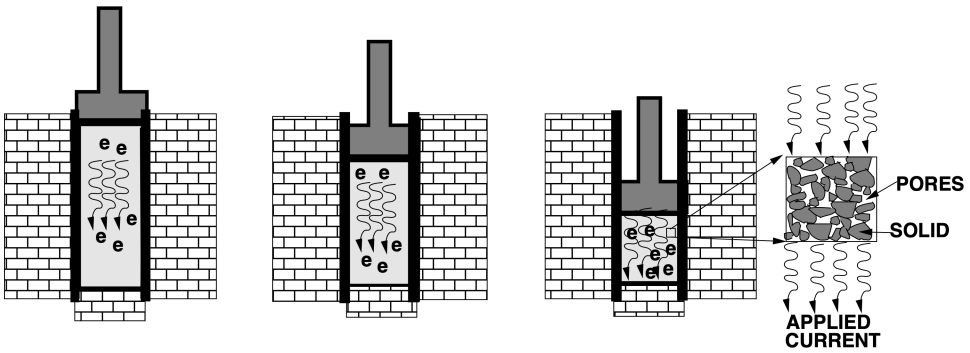

Figure 3.1: Compaction of powdered material with simultaneous applied current.

Sintering is a process by which objects are formed from a powdered stock material. The process is applied to a variety of powdered metals, ceramics, and glasses, as well as graphite and diamond.

- loose powder (stock material)
- green compact (mechanically compressed powder)
- sintered material (thermally processed, densified solid)

Sintering requires heating the compacted (green) material to $70\% - 90\%$ of its melting temperature. Sintering has several advantages over more conventional processing methods, including:

- high purity of processed materials
- the ability to process materials that cannot be easily melted

Generally, powder processing is more expensive than conventional methods like casting and machining. Ongoing research is seeking to make the process cheaper to expand the methods beyound niche markets. Electrically-aided sintering is one promising variation on the generic sintering process. One type of electrically-aided sintering is "spark-sintering". In spark sintering, loose powder is confined in a graphite mold, then simultaneously heated by an electrical current and compressed mechanically.

The objective of this project is to encode a numerical model to simulate the spark sintering process and predict the optimal processing parameters to achieve desired properties in the final product. This will require modelling the interaction of mechanical compression, electrical heating, and pore collapse in the material being processed. However, modeling of the spark sintering process is out of scope of this module, so, a starter code is provided at the end of the assignment for your use. Using the starter code, develop a genetic algorithm code to find the best design strings that achieve the desired system response.

Desired Outcomes

The desired outcomes are reflected by a scalar "cost function" that balances between the potentially conflicting factors we want to optimize.

$$\Pi = w_1 \left(\frac{\max_t \theta - \theta_{des}}{\theta_{des}} \right)^2 + w_2 d(t = t_f)$$

Note that both terms are unitless and each has a weight that captures its relative importance. Verbally, this cost function means we want the *maximum temperature at any time* to closely match a desired temperature and want complete densification by the end of the process, though we don't care about exactly when the material becomes fully dense. You will need to modify your code or the provided starter code to return the highest temperature that occurs and the final densification parameter.

If you find a design with a cost $\Pi \leq TOL$, you can terminate your algorithm early. Otherwise, terminate after a fixed number of generations. Refer to the variable glossary for values.

Design Vector

Using a genetic algorithm as outlined in the theory section, you will vary two of the system parameters to achieve the desired results, subject to constraints.

- $.9 \geq d(t = 0) \geq .5$ (the initial densification parameter)
- $1e7 \geq J \geq 0$ (the magnitude of the current density)

The design vector for this problem will contain only these two values:

$$\mathbf{\Lambda}^i = \{\Lambda_1^i, ..., \Lambda_N^i\} = \{d_o, J\}^i \tag{3.1}$$

Note that, since both design variables have upper and lower bounds, you will need to appropriately scale your random guesses to prevent unacceptable values.

Deliverables

Construct a concise technical report. In your results section, make sure to include and discuss the following:

1. Provide a convergence plot showing the cost of the best design, the mean costs of all parent designs, and the mean cost of the overall population **for each generation**. A convergence plot should show the cost over many generations. A correct implementation will show the cost going down for the entire parent population and the best design. You should use `loglog()` or `semilogy()` when plotting your results since the cost will vary over

several orders of magnitude. See the plotting examples script on bcourses for examples of good and bad plot scaling.

An example of a well-presented convergence plot is shown below.

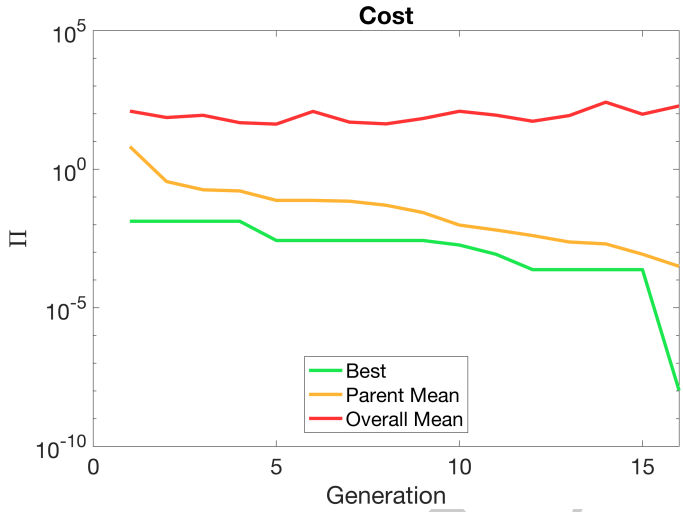

Figure 3.2: Example convergence plot

2. Report your best-performing 4 designs in a table similar to the following.

DESIGN	Λ_1	Λ_2	Λ_3	*etc*	Π
1					
2					
3					
4					

Table 3.1: The top 4 system parameter performers.

3. Include plots of the densification parameter and temperature over time for your single best design. How well does the design achieve the desired system behavior?
4. Discuss the results. How much variation does each parameter have between your top performers?
5. Compare results between several runs of your program. Are the results consistent, or do you get different answers?

VARIABLE GLOSSARY

Table 3.2: Genetic Algorithm Parameters

Symbol	Type	Units	Value	Description
`children`	scalar	none	6	Strings generated by breeding
`parents`	scalar	none	6	Surviving strings for breeding
`S`	scalar	none	20	Designs per generation
`G`	scalar	none	100	Max total generations
θ_{des}	scalar	K	650	Desired peak temperature
TOL	scalar	none	1e-6	Acceptable tolerance for early termination
w_1	scalar	none	1000	Temperature term weight
w_2	scalar	none	100	Densification term weight

Table 3.3: Project Parameters

Symbol	Type	Units	Value	Description
a	scalar	none	0.8	Joule heating absorption coefficient
A_1	scalar	none	$2.5 * 10^{-4}$	densification rate parameter
A_2	scalar	none	1.5	plasticity rate parameter
β	scalar	K^{-1}	$2 * 10^{-5}$	coefficient of thermal expansion
C	scalar	$Jkg^{-1}K^{-1}$	850	heat capacity
d	scalar	none	TBD	densification parameter
$d(t=0)$	scalar	none	$.9 \geq d(t=0) \geq .5$	initial densification parameter
$\mathcal{E}$	3×1 vector	Vm^{-1}	TBD	electrical field
E	$3 \times 3 matrix$	none	TBD	Green-Lagrange strain
H	scalar	$Js^{-1}m^{-3}$	TBD	Joule heating power
$\mathcal{J}$	scalar	none	TBD	Jacobian, $\mathcal{J} = detF$
J	3D vector	A/m^2	Je_3	electrical current density
J	Scalar	A/m^2	$1 * 10^7 \geq J \geq 0$	electrical current density magnitude
κ^o	scalar	GPa	85	dense material bulk modulus
μ^o	scalar	GPa	22	dense material shear modulus
P_1	scalar	none	10^{-3}	elasticity thermal parameter
P_2	scalar	none	10^{-3}	conductivity thermal parameter
P_3	scalar	none	10^{-3}	densification thermal parameter
P_4	scalar	none	10^{-3}	yield strength thermal parameter
ρ_o	scalar	kg/m^3	2350	reference material density
$\sigma^{o,c}$	3×3 matrix	Sm^{-1}	10^51	dense material electrical conductivity
σ^c	3×3 matrix	Sm^{-1}	TBD	effective electrical conductivity
S	$3 \times 3 matrix$	MPa	TBD	Second Piola Kirchhoff stress
σ	$3 \times 3 matrix$	MPa	TBD	Cauchy stress
σ_d^o	scalar	MPa	1.2	reference densification threshold
σ_y^o	scalar	MPa	38	reference yield strength
θ_o	scalar	K	300	reference temperature
$\theta(t=0)$	scalar	K	300	initial temperature
v_p	scalar	none	TBD	pore volume fraction
w	scalar	Jkg^{-1}	TBD	stored energy per unit mass

Starter Code

```
1
2  tic %timekeeping
3  clc; clear all; close all; %housekeeping
4
5  %% Load variables
6  a = 0.8; %Joule heating absorption coefficient
7  A_1 = 2.5e-4; %densification rate parameter
8  A_2 = 1.5; %plasticity rate parameter
9  beta = 2e-5; %coefficient of thermal expansion [1/K]
10 C = 850; %heat capacity [J/kgK]
11 d_t0 = 0.67; %initial densification parameter
12 J_e = [0 0 10^7]; %electrical current density [A/m2]
13 kappa_0 = 85e9; %dense material bulk modulus [Pa]
14 mu_0 = 22e9; %dense material shear modulus [Pa]
15 P_1 = 10^-3; %elasticity thermal parameter
16 P_2 = 10^-3; %conductivity thermal parameter
17 P_3 = 10^-3; %densification thermal parameter
```

```
P_4 = 10^-3; %yield strength thermal parameter
rho_0 = 2350; %reference material density [kg/m3]
sig_0c = 10^5 .* eye(3); %dense material electrical conductivity [S/m]
sig_d0 = 1.2e6; %reference densification threshold [Pa]
sig_y0 = 38e6; %reference yield strength [Pa]
theta_0 = 300; %reference temperature [K]
theta_t0 = 300; %initial temperature [K]

t = [0:10^-4:1]; %time stepping
del_t = 10^-4; %time step
N_t = length(t); %# of steps

%% Pre-allocate memory
p = NaN(N_t,1); %hydrostatic pressure
sig_frob = NaN(N_t,1); %frobenius norm of deviatoric stress
d = NaN(N_t,1); %densification
theta = NaN(N_t,1); %temperature
E_p_norm = NaN(N_t,1); %2-norm of plastic strain

%% Initialize parameters
d(1) = d_t0;
theta(1) = theta_t0;
E_p = zeros(3);
E_theta = zeros(3);
isDdotzero = false;

c1 = kappa_0 + (4/3)*mu_0; c2 = kappa_0 - (2/3)*mu_0;
IE_0 = zeros(6,6);
IE_0(1,1)=c1; IE_0(2,2)=c1; IE_0(3,3)=c1;
IE_0(4,4)=mu_0; IE_0(5,5)=mu_0; IE_0(6,6)=mu_0;
IE_0(2,1)=c2; IE_0(3,1)=c2; IE_0(1,2)=c2; IE_0(1,3)=c2; IE_0(3,2)=c2; IE_0
    (2,3)=c2;

%% Time-stepping

t_step = 0;
for i = 1:N_t
    %Calculate deformation
    [F,J_def] = myDeform(t_step);
    %Calculate mechanical properties
    [IE,sig_c,sig_d,sig_y] = myMechProp(IE_0,d(i),theta(i),P_1,P_2,P_3,...
    P_4,theta_0,sig_0c,sig_d0,sig_y0);
    %Calculate thermal strain
    [E_theta] = myThermalStrain(beta,theta(i),theta_0);
    %Calculate strain and stress
    [E,E_eff,S,sig] = myStrainStress(F,J_def,E_p,E_theta,IE);
    %Calculate deliverables
    [p(i),sig_frob(i),E_p_norm(i),sig_dev] = myContinuumResults(sig,E_p);
    %Calculate RHS for next time step
    [RHS_d] = myRHS_d(sig,sig_d,A_1,isDdotzero); %densification
    [RHS_Ep] = myRHS_Ep(A_2,sig_dev,J_def,F,sig_y); %plastic strain

    %Forward Euler
    if i ~= N_t
```

```
71      %[theta(i+1),E_p,d(i+1),RHS_d,isDdotzero] = myForwardEuler(RHS_theta,
            RHS_Ep,RHS_d,...
72      %theta(i),E_p,d(i),del_t);
73      [d(i+1),RHS_d,isDdotzero] = myForwardEulerD(RHS_d,d(i),del_t);
74      [RHS_theta] = myRHS_theta(E_eff,a,sig_c,J_def,J_e,S,RHS_Ep,RHS_d,...
75      P_1,theta(i),theta_0,IE_0,rho_0,C,beta,d(i)); %RHS of theta
76      [theta(i+1),E_p] = myForwardEuler(RHS_theta,RHS_Ep,...
77      theta(i),E_p,del_t);
78      %update time for next step
79      t_step = t_step + del_t;
80      end
81  end
82
83  %% Plots
84
85  %Hydrostatic pressure
86  figure(1)
87  xlim([0 1])
88  plot(t,p./10^9,'Linewidth',5)
89  title('Hydrostatic Pressure over Time')
90  xlabel('Time [s]')
91  ylabel('p [GPa]')
92  grid on
93  hold off
94
95  %Frobenius norm of deviatoric stress
96  figure(2)
97  xlim([0 1])
98  plot(t,sig_frob./10^6,'Linewidth',5)
99  title('Frobenius norm of Deviatoric Stress over Time')
100 xlabel('Time [s]')
101 ylabel('||\sigma''||_F [MPa]')
102 grid on
103 hold off
104
105 %Densification
106 figure(3)
107 xlim([0 1])
108 plot(t,d,'Linewidth',5)
109 title('Densification over Time')
110 xlabel('Time [s]')
111 ylabel('d')
112 grid on
113 hold off
114
115 %Temperature
116 figure(4)
117 xlim([0 1])
118 plot(t,theta,'Linewidth',5)
119 title('Temperature over Time')
120 xlabel('Time [s]')
121 ylabel('Temperature [K]')
122 grid on
123 hold off
```

```
124
125 %2-norm of plastic strain
126 figure(5)
127 xlim([0 1])
128 plot(t,E_p_norm,'Linewidth',5)
129 title('2-norm of Plastic Strain over Time')
130 xlabel('Time [s]')
131 ylabel('||E^p||_2')
132 grid on
133 hold off
134
135 toc %timekeeping
136
137 %% Functions
138
139 function [F,J_def] = myDeform(t) %deformation matrix and Jacobian
140
141 B = zeros(3); B(3,3) = -0.15 * t;
142 F = B + eye(3);
143 J_def = det(F);
144
145 end
146
147 function [IE,sig_c,sig_d,sig_y] = myMechProp(IE_0,d,theta,P_1,P_2,P_3,...
148     P_4,theta_0,sig_0c,sig_d0,sig_y0) %mechanical properties
149
150 %elasticity tensor
151 IE = (1-d) .* IE_0 .* exp(-P_1 * (theta - theta_0) / theta_0);
152 %electrical conductivity
153 sig_c = (1-d) .* sig_0c .* exp(-P_2 * (theta - theta_0) / theta_0);
154 %densification stress threshold
155 sig_d = sig_d0 .* exp(-P_3 * (theta - theta_0) / theta_0);
156 %yield strength
157 sig_y = sig_y0 .* exp(-P_4 * (theta - theta_0) / theta_0);
158
159 end
160
161 function [E_theta] = myThermalStrain(beta,theta,theta_0)
162
163 E_theta = beta * (theta - theta_0) .* eye(3);
164
165 end
166
167 function [E,E_eff,S,sig] = myStrainStress(F,J_def,E_p,E_theta,IE) %strain and
    stress
168
169 %Lagrangian strain
170 E = (1/2).*(F' * F - eye(3));
171 %Effective strain
172 E_eff = E - E_p - E_theta;
173 %Piola-Kirchhoff Stress 2nd
174 S_array = IE * [E_eff(1,1);E_eff(2,2);E_eff(3,3);...
175 2*E_eff(1,2);2*E_eff(2,3);E_eff(3,1)]; %piola-kirch stress 2nd
176 S = zeros(3,3);
```

```
177 S(1,1) = S_array(1);
178 S(2,2) = S_array(2);
179 S(3,3) = S_array(3);
180 S(1,2) = S_array(4);
181 S(2,1) = S_array(4);
182 S(2,3) = S_array(5);
183 S(3,2) = S_array(5);
184 S(3,1) = S_array(6);
185 S(1,3) = S_array(6);
186 sig = 1/J_def .* F * S * F';
187
188 end
189
190 function [p,sig_frob,E_p_norm,sig_dev] = myContinuumResults(sig,E_p) %
    deliverables
191
192 p = trace(sig)/3; %hydrostatic pressure
193 sig_dev = sig - eye(3) .* p; %deviatoric stress
194 sig_frob = norm(sig_dev,'fro'); %frob norm of dev stress
195 E_p_norm = norm(E_p); %2-norm of plastic strain
196
197 end
198
199 function [RHS_d] = myRHS_d(sig,sig_d,A_1,isDdotzero) %RHS of densification
200
201 if isDdotzero == false
202     if norm(trace(sig))/3 > sig_d && trace(sig)/3 <= 0
203         RHS_d = -A_1 * (norm(trace(sig)/3) / sig_d - 1);
204     else
205         RHS_d = 0;
206     end
207 else
208     RHS_d = 0;
209 end
210
211
212 end
213
214 function [RHS_Ep] = myRHS_Ep(A_2,sig_dev,J_def,F,sig_y) %RHS of plastic strain
215
216 %calculate rate parameter
217 if norm(sig_dev) > sig_y
218     lambda_dot = A_2 * (norm(sig_dev) / sig_y - 1);
219 else
220     lambda_dot = 0;
221 end
222 %calculate Eulerian plastic strain parameter
223 if sig_dev ==0
224     e_dot = zeros(3);
225 else
226     e_dot = lambda_dot .* sig_dev ./ norm(sig_dev);
227 end
228 %calculate Lagrangian plastic strain rate parameter
229 RHS_Ep = J_def .* inv(F) * e_dot * inv(F)';
```

```
230
231 end
232
233 function [RHS_theta] = myRHS_theta(E_eff,a,sig_c,J_def,J_e,S,RHS_Ep,RHS_d,...
234     P_1,theta,theta_0,IE_0,rho_0,C,beta,d) %RHS of theta
235
236 %calculate E as an array
237 E_array = [E_eff(1,1);E_eff(2,2);E_eff(3,3);2*E_eff(1,2);2*E_eff(2,3);E_eff
    (3,1)];
238 %Joule heating term
239 JH = a / (trace(sig_c)/3) * J_def .* J_e * J_e';
240 %calculate numerator of RHS_theta
241 numerator = trace(S' * RHS_Ep) + (RHS_d / 2) * exp(-P_1 * (theta - theta_0) /
    theta_0)...
242     .* E_array' * IE_0 * E_array + JH;
243 %calculate denominator of RHS_theta
244 denominator = rho_0 * C - beta * trace(S) - (1-d) / (2 * theta_0) * P_1 *...
245     exp(-P_1 * (theta - theta_0) / theta_0) .* E_array' * IE_0 * E_array;
246 %calculate RHS_theta
247 RHS_theta = numerator / denominator;
248
249 end
250
251 function [d_new,RHS_d,isDdotzero] = myForwardEulerD(RHS_d,d,del_t) % Forward
    Euler scheme for ddot
252
253 d_new = d + del_t * RHS_d; %update theta
254 if d_new < 0
255     d_new = 0;
256     isDdotzero = true;
257     RHS_d = 0;
258 else
259     isDdotzero = false;
260 end
261 if d_new > 1
262     d_new = 1;
263 end
264
265 end
266
267 function [theta_new,E_p_new] = myForwardEuler(RHS_theta,RHS_Ep,...
268     theta,E_p,del_t) % Forward Euler scheme
269
270 theta_new = theta + del_t * RHS_theta; %update theta
271 E_p_new = E_p + del_t .* RHS_Ep; %update theta
272
273 end
```

4 Solution

The assignment solution is encoded in Matlab below.

```
1
2  tic %timekeeping
3  clc; clear all; close all; %housekeeping
4  %
5
6  %%%%GENETIC ALGORITHM START%%%%%%%%
7
8  %general GA parameters
9  Sgen = 20; % # of genetic strings
10 Pgen = 6; % # of parents
11 Cgen = 6; % # of children
12 TOL = 1E-6; % error tolerance
13 GEN_LIMIT = 100; % generation limit
14 w1 = 1000; %temperature term weight
15 w2 = 100; %densification term weight
16 theta_des = 650; %desired peak temperature [K]
17
18 %%Step 1: Create S design strings
19 d_t0 = zeros(Sgen,1);
20 J_e = zeros(Sgen,3);
21
22 for i = 1:Sgen
23     d_t0(i) = 0.9 + (0.5-(0.9))*rand; %initial densification parameter
24     J_ez = 10^7 + (0-10^7)*rand;
25     J_e(i,:) = [0 0 J_ez]; %electrical current density [A/m2]
26 end
27
28 %pre-allocation
29 Pi = NaN(Sgen,1);
30
31 n_gen = 0; %number of generations counter
32 j = 1; %while loop counter
33 Pi(1)= 1000; %high Pi to start while loop
34
35 while min(Pi)>TOL
36
37     if n_gen >= GEN_LIMIT
38         break
39     end
40
41     parfor k = 1:Sgen %string loop
42
43         % Load variables
44         a = 0.8; %Joule heating absorption coefficient
45         A_1 = 2.5e-4; %densification rate parameter
46         A_2 = 1.5; %plasticity rate parameter
47         beta = 2e-5; %coefficient of thermal expansion [1/K]
48         C = 850; %heat capacity [J/kgK]
49         %d_t0 = 0.67; %initial densification parameter
50         %J_e = [0 0 10^7]; %electrical current density [A/m2]
51         kappa_0 = 85e9; %dense material bulk modulus [Pa]
```

```
52          mu_0 = 22e9; %dense material shear modulus [Pa]
53          P_1 = 10^-3; %elasticity thermal parameter
54          P_2 = 10^-3; %conductivity thermal parameter
55          P_3 = 10^-3; %densification thermal parameter
56          P_4 = 10^-3; %yield strength thermal parameter
57          rho_0 = 2350; %reference material density [kg/m3]
58          sig_0c = 10^5 .* eye(3); %dense material electrical conductivity [S/m]
59          sig_d0 = 1.2e6; %reference densification threshold [Pa]
60          sig_y0 = 38e6; %reference yield strength [Pa]
61          theta_0 = 300; %reference temperature [K]
62          theta_t0 = 300; %initial temperature [K]
63
64          t = [0:10^-4:1]; %time stepping
65          del_t = 10^-4; %time step
66          N_t = length(t); %# of steps
67
68          % Pre-allocate memory
69          p = NaN(N_t,1); %hydrostatic pressure
70          sig_frob = NaN(N_t,1); %frobenius norm of deviatoric stress
71          d = NaN(N_t,1); %densification
72          theta = NaN(N_t,1); %temperature
73          E_p_norm = NaN(N_t,1); %2-norm of plastic strain
74
75          % Initialize parameters
76          %d(1) = d_t0;
77          theta(1) = theta_t0;
78          E_p = zeros(3);
79          E_theta = zeros(3);
80          isDdotzero = false;
81
82          c1 = kappa_0 + (4/3)*mu_0; c2 = kappa_0 - (2/3)*mu_0;
83          IE_0 = zeros(6,6);
84          IE_0(1,1)=c1; IE_0(2,2)=c1; IE_0(3,3)=c1;
85          IE_0(4,4)=mu_0; IE_0(5,5)=mu_0; IE_0(6,6)=mu_0;
86          IE_0(2,1)=c2; IE_0(3,1)=c2; IE_0(1,2)=c2; IE_0(1,3)=c2; IE_0(3,2)=c2;
                IE_0(2,3)=c2;
87
88
89          d_t0s = d_t0(k);
90          d(1) = d_t0s;
91          J_es = J_e(k,:);
92
93          % Time-stepping
94          t_step = 0;
95          for i = 1:N_t
96              %Calculate deformation
97              [F,J_def] = myDeform(t_step);
98              %Calculate mechanical properties
99              [IE,sig_c,sig_d,sig_y] = myMechProp(IE_0,d(i),theta(i),P_1,P_2,P_3
                    ,...
100             P_4,theta_0,sig_0c,sig_d0,sig_y0);
101             %Calculate thermal strain
102             [E_theta] = myThermalStrain(beta,theta(i),theta_0);
103             %Calculate strain and stress
```

```
104             [E,E_eff,S,sig] = myStrainStress(F,J_def,E_p,E_theta,IE);
105             %Calculate deliverables
106             [p(i),sig_frob(i),E_p_norm(i),sig_dev] = myContinuumResults(sig,
                    E_p);
107             %Calculate RHS for next time step
108             [RHS_d] = myRHS_d(sig,sig_d,A_1,isDdotzero); %densification
109             [RHS_Ep] = myRHS_Ep(A_2,sig_dev,J_def,F,sig_y); %plastic strain
110
111             %Forward Euler
112             if i ~= N_t
113             %[theta(i+1),E_p,d(i+1),RHS_d,isDdotzero] = myForwardEuler(
                    RHS_theta,RHS_Ep,RHS_d,...
114             %theta(i),E_p,d(i),del_t);
115             [d(i+1),RHS_d,isDdotzero] = myForwardEulerD(RHS_d,d(i),del_t);
116             [RHS_theta] = myRHS_theta(E_eff,a,sig_c,J_def,J_es,S,RHS_Ep,RHS_d
                    ,...
117             P_1,theta(i),theta_0,IE_0,rho_0,C,beta,d(i)); %RHS of theta
118             [theta(i+1),E_p] = myForwardEuler(RHS_theta,RHS_Ep,...
119             theta(i),E_p,del_t);
120             %update time for next step
121             t_step = t_step + del_t;
122             end
123         end
124         %%Step 2: Evaluate fitness of strings
125         Pi(k) = w1 * ((max(theta)-theta_des) / theta_des)^2 + w2 * d(N_t);
126     end
127
128     %%Step 3: Rank the strings by ascending value of Pi
129     [Pi,ind] = sort(Pi); % keep the original indices for finding associated A
            string
130     %align des strings and sol vectors in order it was sorted above
131     d_t0 = d_t0(ind,:);
132     J_e = J_e(ind,:);
133
134     %%Step 4: Mate top strings
135
136     if n_gen>=GEN_LIMIT*3/4 % enable mutation if number of generations reach
            3/4
137         phi = -0.5+(1.5-(-0.5)).*rand(Pgen,3); % calculate phi - MUTATION
                ENABLED
138     else
139         phi = rand(Pgen,3); % calculate phi - MUTATION DISABLED
140     end
141
142     ind1 = [1:2:Pgen]'; % child 1 index pair
143     ind2 = [2:2:Pgen]'; % child 2 index pair
144     d_t0 = vertcat(d_t0(1:Pgen,:),phi(ind1).*d_t0(ind1,:)+(1-phi(ind1)).*d_t0(
            ind2,:),...
145         phi(ind2).*d_t0(ind2,:)+(1-phi(ind2)).*d_t0(ind1,:)); % concatenate c
                on p
146     J_e = vertcat(J_e(1:Pgen,:),phi(ind1).*J_e(ind1,:)+(1-phi(ind1)).*J_e(ind2
            ,:),...
147         phi(ind2).*J_e(ind2,:)+(1-phi(ind2)).*J_e(ind1,:)); % concatenate c on
                p
```

```
148
149     %%Step 5: Generate S-2P new strings
150     d_t0_new = 0.9 + (0.5-(0.9))*rand(Sgen-2*Pgen,1); %initial densification
            parameter
151     d_t0 = vertcat(d_t0,d_t0_new);
152     J_e_new = [zeros(Sgen-2*Pgen,1),zeros(Sgen-2*Pgen,1),10^7 + (0-10^7)*rand(
            Sgen-2*Pgen,1)];
153     J_e = vertcat(J_e,J_e_new); %electrical current density [A/m2]
154
155     %Save generation-based outputs
156     Min(j)=min(Pi);
157     Ave(j)=mean(Pi);
158     Ave_parent(j)=mean(Pi(1:Pgen));
159
160     n_gen = n_gen + 1; %number of generations counter
161     j = j + 1; %while loop counter
162
163
164 end
165
166 %%%% Gen Plotting %%%%%%%%%%%%%%%%%%%%%%%%%
167 figure(1)
168 semilogy(1:n_gen,Min,'b',1:n_gen,Ave,'r',1:n_gen,Ave_parent,'g')
169 legend('Best Design','Mean of Population','Mean of Parents')
170 xlabel('Generation')
171 ylabel('Error')
172 title('GA Results')
173 xtickformat('%.0f')
174 hold off
175
176
177 %%
178 %%%%Obtain Best Design Parameters%%%%%%%%%%
179 Lambda = [d_t0(1:4,:), J_e(1:4,3)];
180 Lambda(:,1)
181 Lambda(:,2)
182 Error = Pi(1);
183
184 %% Time-stepping
185
186 % Load variables
187 a = 0.8; %Joule heating absorption coefficient
188 A_1 = 2.5e-4; %densification rate parameter
189 A_2 = 1.5; %plasticity rate parameter
190 beta = 2e-5; %coefficient of thermal expansion [1/K]
191 C = 850; %heat capacity [J/kgK]
192 d_t0 = Lambda(1,1); %initial densification parameter
193 J_e = [0 0 Lambda(1,2)]; %electrical current density [A/m2]
194 kappa_0 = 85e9; %dense material bulk modulus [Pa]
195 mu_0 = 22e9; %dense material shear modulus [Pa]
196 P_1 = 10^-3; %elasticity thermal parameter
197 P_2 = 10^-3; %conductivity thermal parameter
198 P_3 = 10^-3; %densification thermal parameter
199 P_4 = 10^-3; %yield strength thermal parameter
```

```
rho_0 = 2350; %reference material density [kg/m3]
sig_0c = 10^5 .* eye(3); %dense material electrical conductivity [S/m]
sig_d0 = 1.2e6; %reference densification threshold [Pa]
sig_y0 = 38e6; %reference yield strength [Pa]
theta_0 = 300; %reference temperature [K]
theta_t0 = 300; %initial temperature [K]

t = [0:10^-4:1]; %time stepping
del_t = 10^-4; %time step
N_t = length(t); %# of steps

% Pre-allocate memory
p = NaN(N_t,1); %hydrostatic pressure
sig_frob = NaN(N_t,1); %frobenius norm of deviatoric stress
d = NaN(N_t,1); %densification
theta = NaN(N_t,1); %temperature
E_p_norm = NaN(N_t,1); %2-norm of plastic strain

% Initialize parameters
d(1) = d_t0;
theta(1) = theta_t0;
E_p = zeros(3);
E_theta = zeros(3);
isDdotzero = false;

c1 = kappa_0 + (4/3)*mu_0; c2 = kappa_0 - (2/3)*mu_0;
IE_0 = zeros(6,6);
IE_0(1,1)=c1; IE_0(2,2)=c1; IE_0(3,3)=c1;
IE_0(4,4)=mu_0; IE_0(5,5)=mu_0; IE_0(6,6)=mu_0;
IE_0(2,1)=c2; IE_0(3,1)=c2; IE_0(1,2)=c2; IE_0(1,3)=c2; IE_0(3,2)=c2; IE_0
    (2,3)=c2;



t_step = 0;
for i = 1:N_t
    %Calculate deformation
    [F, J_def] = myDeform(t_step);
    %Calculate mechanical properties
    [IE, sig_c, sig_d, sig_y] = myMechProp(IE_0,d(i),theta(i),P_1,P_2,P_3,...
    P_4,theta_0,sig_0c,sig_d0,sig_y0);
    %Calculate thermal strain
    [E_theta] = myThermalStrain(beta,theta(i),theta_0);
    %Calculate strain and stress
    [E,E_eff,S,sig] = myStrainStress(F,J_def,E_p,E_theta,IE);
    %Calculate deliverables
    [p(i),sig_frob(i),E_p_norm(i),sig_dev] = myContinuumResults(sig,E_p);
    %Calculate RHS for next time step
    [RHS_d] = myRHS_d(sig,sig_d,A_1,isDdotzero); %densification
    [RHS_Ep] = myRHS_Ep(A_2,sig_dev,J_def,F,sig_y); %plastic strain

    %Forward Euler
    if i ~= N_t
```

```
252     %[theta(i+1),E_p,d(i+1),RHS_d,isDdotzero] = myForwardEuler(RHS_theta,
            RHS_Ep,RHS_d,...
253     %theta(i),E_p,d(i),del_t);
254     [d(i+1),RHS_d,isDdotzero] = myForwardEulerD(RHS_d,d(i),del_t);
255     [RHS_theta] = myRHS_theta(E_eff,a,sig_c,J_def,J_e,S,RHS_Ep,RHS_d,...
256     P_1,theta(i),theta_0,IE_0,rho_0,C,beta,d(i)); %RHS of theta
257     [theta(i+1),E_p] = myForwardEuler(RHS_theta,RHS_Ep,...
258     theta(i),E_p,del_t);
259     %update time for next step
260     t_step = t_step + del_t;
261     end
262 end
263
264 %% Plots
265
266 %Hydrostatic pressure
267 figure(2)
268 xlim([0 1])
269 plot(t,p./10^9,'Linewidth',5)
270 title('Hydrostatic Pressure over Time')
271 xlabel('Time [s]')
272 ylabel('p [GPa]')
273 grid on
274 hold off
275
276 %Frobenius norm of deviatoric stress
277 figure(3)
278 xlim([0 1])
279 plot(t,sig_frob./10^6,'Linewidth',5)
280 title('Frobenius norm of Deviatoric Stress over Time')
281 xlabel('Time [s]')
282 ylabel('||\sigma''||_F [MPa]')
283 grid on
284 hold off
285
286 %Densification
287 figure(4)
288 xlim([0 1])
289 plot(t,d,'Linewidth',5)
290 title('Densification over Time')
291 xlabel('Time [s]')
292 ylabel('d')
293 grid on
294 hold off
295
296 %Temperature
297 figure(5)
298 xlim([0 1])
299 plot(t,theta,'Linewidth',5)
300 title('Temperature over Time')
301 xlabel('Time [s]')
302 ylabel('Temperature [K]')
303 grid on
304 hold off
```

```
305
306 %2-norm of plastic strain
307 figure(6)
308 xlim([0 1])
309 plot(t,E_p_norm,'Linewidth',5)
310 title('2-norm of Plastic Strain over Time')
311 xlabel('Time [s]')
312 ylabel('||E^p||_2')
313 grid on
314 hold off
315
316 toc %timekeeping
317
318 %% Functions
319
320 function [F,J_def] = myDeform(t) %deformation matrix and Jacobian
321
322 B = zeros(3); B(3,3) = -0.15 * t;
323 F = B + eye(3);
324 J_def = det(F);
325
326 end
327
328 function [IE,sig_c,sig_d,sig_y] = myMechProp(IE_0,d,theta,P_1,P_2,P_3,...
329     P_4,theta_0,sig_0c,sig_d0,sig_y0) %mechanical properties
330
331 %elasticity tensor
332 IE = (1-d) .* IE_0 .* exp(-P_1 * (theta - theta_0) / theta_0);
333 %electrical conductivity
334 sig_c = (1-d) .* sig_0c .* exp(-P_2 * (theta - theta_0) / theta_0);
335 %densification stress threshold
336 sig_d = sig_d0 .* exp(-P_3 * (theta - theta_0) / theta_0);
337 %yield strength
338 sig_y = sig_y0 .* exp(-P_4 * (theta - theta_0) / theta_0);
339
340 end
341
342 function [E_theta] = myThermalStrain(beta,theta,theta_0)
343
344 E_theta = beta * (theta - theta_0) .* eye(3);
345
346 end
347
348 function [E,E_eff,S,sig] = myStrainStress(F,J_def,E_p,E_theta,IE) %strain and
    stress
349
350 %Lagrangian strain
351 E = (1/2).*(F' * F - eye(3));
352 %Effective strain
353 E_eff = E - E_p - E_theta;
354 %Piola-Kirchhoff Stress 2nd
355 S_array = IE * [E_eff(1,1);E_eff(2,2);E_eff(3,3);...
356 2*E_eff(1,2);2*E_eff(2,3);E_eff(3,1)]; %piola-kirch stress 2nd
357 S = zeros(3,3);
```

```
358 S(1,1) = S_array(1);
359 S(2,2) = S_array(2);
360 S(3,3) = S_array(3);
361 S(1,2) = S_array(4);
362 S(2,1) = S_array(4);
363 S(2,3) = S_array(5);
364 S(3,2) = S_array(5);
365 S(3,1) = S_array(6);
366 S(1,3) = S_array(6);
367 sig = 1/J_def .* F * S * F';
368
369 end
370
371 function [p,sig_frob,E_p_norm,sig_dev] = myContinuumResults(sig,E_p) %
    deliverables
372
373 p = trace(sig)/3; %hydrostatic pressure
374 sig_dev = sig - eye(3) .* p; %deviatoric stress
375 sig_frob = norm(sig_dev,'fro'); %frob norm of dev stress
376 E_p_norm = norm(E_p); %2-norm of plastic strain
377
378 end
379
380 function [RHS_d] = myRHS_d(sig,sig_d,A_1,isDdotzero) %RHS of densification
381
382 if isDdotzero == false
383     if norm(trace(sig))/3 > sig_d && trace(sig)/3 <= 0
384         RHS_d = -A_1 * (norm(trace(sig)/3) / sig_d - 1);
385     else
386         RHS_d = 0;
387     end
388 else
389     RHS_d = 0;
390 end
391
392
393 end
394
395 function [RHS_Ep] = myRHS_Ep(A_2,sig_dev,J_def,F,sig_y) %RHS of plastic strain
396
397 %calculate rate parameter
398 if norm(sig_dev) > sig_y
399     lambda_dot = A_2 * (norm(sig_dev) / sig_y - 1);
400 else
401     lambda_dot = 0;
402 end
403 %calculate Eulerian plastic strain parameter
404 if sig_dev ==0
405     e_dot = zeros(3);
406 else
407     e_dot = lambda_dot .* sig_dev ./ norm(sig_dev);
408 end
409 %calculate Lagrangian plastic strain rate parameter
410 RHS_Ep = J_def .* inv(F) * e_dot * inv(F)';
```

```
411
412 end
413
414 function [RHS_theta] = myRHS_theta(E_eff,a,sig_c,J_def,J_e,S,RHS_Ep,RHS_d,...
415     P_1,theta,theta_0,IE_0,rho_0,C,beta,d) %RHS of theta
416
417 %calculate E as an array
418 E_array = [E_eff(1,1);E_eff(2,2);E_eff(3,3);2*E_eff(1,2);2*E_eff(2,3);E_eff
    (3,1)];
419 %Joule heating term
420 JH = a / (trace(sig_c)/3) * J_def .* J_e * J_e';
421 %calculate numerator of RHS_theta
422 numerator = trace(S' * RHS_Ep) + (RHS_d / 2) * exp(-P_1 * (theta - theta_0) /
    theta_0)...
423     .* E_array' * IE_0 * E_array + JH;
424 %calculate denominator of RHS_theta
425 denominator = rho_0 * C - beta * trace(S) - (1-d) / (2 * theta_0) * P_1 *...
426     exp(-P_1 * (theta - theta_0) / theta_0) .* E_array' * IE_0 * E_array;
427 %calculate RHS_theta
428 RHS_theta = numerator / denominator;
429
430 end
431
432 function [d_new,RHS_d,isDdotzero] = myForwardEulerD(RHS_d,d,del_t) % Forward
    Euler scheme for ddot
433
434 d_new = d + del_t * RHS_d; %update theta
435 if d_new < 0
436     d_new = 0;
437     isDdotzero = true;
438     RHS_d = 0;
439 else
440     isDdotzero = false;
441 end
442 if d_new > 1
443     d_new = 1;
444 end
445
446 end
447
448 function [theta_new,E_p_new] = myForwardEuler(RHS_theta,RHS_Ep,...
449     theta,E_p,del_t) % Forward Euler scheme
450
451 theta_new = theta + del_t * RHS_theta; %update theta
452 E_p_new = E_p + del_t .* RHS_Ep; %update theta
453
454 end
```

5 Ethical Considerations for this Project

A goal of this project is to enable advancements in science and engineering through to address critical national challenges associated with next generation food systems. There are deep ethical considerations associated with any technology, in particular for food systems. While technology has tremendous potential to identify greater efficiencies, when it is created without appropriate consideration for who will have access to and control over new resources, or how the new technologies will impact those who work in the system, the efficiencies identified may come at the cost of greater societal inequity. It is important to pursue harnessing technology to disrupt existing inequities, rather than further entrench existing power structures. The following areas should be considered:

- Labor: 1) occupational health, 2) food manufacturing, and 3) outdoor agriculture labor;
- Producers: 1) Small- to mid-size farms, 2) urban agriculture, and 3) research in farm transitions; Technology: 1) research in technology and democracy;
- Health Human Rights: 1) land rights, 2) social justice, and 3) decolonization in agriculture;

Please consider the following questions:

- What are the societal implications of the technology that you are developing?
- Can this technology be distributed fairly and equitably to a wide variety of entities in agricultural industry?
- Are there any potential unintended consequences of this technology becoming available?
- Are there any harmful "spinoffs" of this technology?
- Are there any useful "spinoffs" of this technology?

6 References

1. Holland, J. H. 1975. Adaptation in natural artificial systems. Ann Arbor, Mich. University of Michigan Press.
2. Holland, J.H.; Miller, J.H. (1991). Artificial Adaptive Agents in Economic Theory (PDF). Amer- ican Economic Review. 81 (2): 365-71. Archived from the original (PDF) on October 27, 2005.
3. Goldberg, D. E. 1989. Genetic algorithms in search, optimization machine learning. Addison- Wesley.
4. Davis, L. 1991. Handbook of Genetic Algorithms. Thompson Computer Press.
5. Onwubiko, C. 2000 Introduction to engineering design optimization. Prentice Hall.
6. Goldberg, D. E. Deb, K. 2000. Special issue on Genetic Algorithms. Computer Methods in Applied Mechanics Engineering. 186 (2-4) 121-124.
7. Zohdi, T. I. (2009) Mechanistic modeling of swarms. Computer Methods in Applied Mechanics and Engineering. Volume 198, Issues 21-26, Pages 2039-2051.
8. Zohdi, T. I. (2018). Multiple UAVs for Mapping: a review of basic modeling, simulation and applications. Annual Review of Environment and Resources. https://doi.org/10.1146/annurev- environ-102017-025912
9. Zohdi, T. I. (2019). The Game of Drones: rapid agent-based machine-learning models for multi- UAV path planning. Computational Mechanics. https://doi.org/10.1007/s00466-019-01761-9
10. Zohdi, T., (2020) A machine-learning framework for rapid adaptive digital-twin based fire- propagation simulation in complex environments. Computer Methods in Applied Mechanics and Engineering. https://doi.org/10.1016/j.cma.2020.112907
11. Luenberger, D. 1974. Introduction to Linear Nonlinear Programming. Addison-Wesley, Menlo Park.
12. Gill, P. Murray, W. and Wright, M. 1995. Practical optimization. Academic Press.