Genetic Algorithm for Laser Wakefield Accelerator (GALWA)[*] Documentation
Version 3.0

Sam Close, Jaspreet Aujla, John Scott & Elisabetta Boella

August 28, 2021

Contents

1 Introduction

Genetic Algorithm for Laser Wakefield Accelerator (GALWA) is a metaheuristic algorithm designed to optimise simulation outcomes of laser wakefield acceleration (LWFA) through evolutionary tailoring of input parameters. It is written in Python and currently supports OSIRIS simulations.

2 The Genetic Algorithm

2.1 Original Method

Primarily developed by Jaspreet Aujla, the genetic algorithm is the core of the system. Upon reflection, one may call it instead a quasi-genetic algorithm, since it does not display the usual characteristics of a genetic algorithm: it both clones, and breeds individuals.

[*]Work in progress name

2.1.1 Pseudo-code

Before running the core of the code, some values are initialised, namely the maximum generation number (G), and the number of individuals per generation (N). The current generation number will be g.

```
1. If this is the first generation (i.e., $g = 0$), run the populate method (step 2). Otherwise, run the repopulate method (step 4). Once $g = G$, stop.
2. Create $N$ individuals, with randomly selected characteristics.
    (a) Pass these individuals to the simulation, in order to extract their merit function.
    (b) Assign each individual their merit function.
    (c) Go to step 3.
3. In the mating stage, sort the generation by merit function.
    (a) Copy the top half of performers into the next generation. Also copy their characteristics into a 'gene pool'.
    (b) Randomly create new individuals from this gene pool. Each individual's characteristics have a fixed chance to mutate, and become another random value.
    (c) This is now the next generation (i.e., $g = g + 1$). Return to step 1.
4. Assigns merit functions to the individuals if they do not already have one. Go to step 3.
```

2.2 Sexual Method

A more typical method, this method is inspired by nature, but is considerably simpler. The main simplification is no sexes. At present, this is a separate branch on the repository.

2.2.1 Pseudo-code

The pseudo code is the same as in 2.1.1, differing only in the mating stage (3).

```
⋮

3. In the mating stage, sort the generation by merit function.
    (a) Select the top half of performers. Pair them off. Make smaller gene pools with these pairs.
    (b) Randomly create four children from this pool, with a chance to mutate for each characteristic.
    (c) Once done for all pairs, these children are the next generation. Return to step 1.

⋮
```

2.3 Asexual Method

Another typical method, which again is much simpler than real life. The main simplification is the way in which mutations occur. At present, this is a separate branch on the repository.

2.3.1 Pseudo-code

The pseudo code is the same as in 2.1.1, differing only in the mating stage (3).

```
⋮

3. In the mating stage, sort the generation by merit function.
    (a) Select the top half of performers.
    (b) For each, clone them twice. When cloning, each characteristic has a fixed chance to mutate.
    (c) These clones are the next generation. Return to step 1.

⋮
```

2.4 Performance & Convergence

Measuring the ‘success’ of a genetic algorithm is difficult, since there is no universal benchmark. This is made particularly hard if our target value has no set limit.

In our case, the merit function we seek to minimize is the fractional weighted uncertainty,

$$\text{Merit Function} = \frac{\sigma_E}{\langle E \rangle}. \tag{1}$$

These weighted values are obtained in the following process:

1. Calculate the charge-energy for each particle i, $q_i E_i$.
2. Sum all charge-energies, $\sum q_i E_i$.
3. Sum all charges to get the total charge, $Q = \sum q_i$.
4. Calculate the charge-weighted average, $\langle E \rangle = \sum q_i E_i / Q$.
5. Calculate the weighted-standard deviation,

$$\sigma_E = \sqrt{\sum_i \frac{q_i (E_i - \langle E \rangle)^2}{Q}}$$

Momentarily neglecting physics, one sees that the merit function can take a value in the interval $[0, \infty]$, but with two free parameters, it can approach these bounds from two directions. When we then consider physics, we can immediately reduce this range to $(0, \infty)$, since it is physically impossible to have either quantity be truly zero or infinite.

This can, unfortunately, lure the system into fulfilling the merit function, but not giving us the result we want. For instance, it may find that it can lower the merit function by making the plasma length very short, reducing the interacting time, and thereby reducing the spread of energy, but hardly changing the energy of the beam. Or, it may greatly accelerate the beam, but with no consideration for spread.

3 Interaction with OSIRIS

3.1 Unit System

OSIRIS uses a natural unit system, which is not always the most helpful. As such, there are a handful of core quantities which determine the other system parameters.

The most central of these is the plasma density, n, with units $[n] = \mathrm{m}^{-3}$. SI was preferred to CGS although was sometimes unavoidable. In the forthcoming, OSIRIS quantities will be primed ($'$); likewise, regular quantities will be unprimed.

3.1.1 Time & Frequency

The time scale of the system is given with respect to the plasma (angular) frequency, ω_p, calculated as

$$\omega_p = e\sqrt{\frac{n}{m_e \epsilon_0}}, \tag{2}$$

with units $[\omega_p] = \mathrm{rad\,s}^{-1}$. Therefore, in OSIRIS time, $t' := t\omega_p$.

By a similar logic, all angular frequencies are relative to the plasma frequency, therefore, $\omega' := \omega/\omega_p$.

3.1.2 Length

The primary length scale of the system is given with respect to the skin depth, which we will henceforth call τ, given as

$$\begin{aligned} \tau &= \frac{c}{\omega_p} \\ &= \frac{1}{e}\sqrt{\frac{m_e}{n\mu_0}} \end{aligned} \tag{3}$$

with units $[\tau] = \mathrm{m}$, where clearly, our OSIRIS positions are given as $\mathbf{x}' := \mathbf{x}/\tau$.

3.1.3 'Momenta'

Despite being called momenta in the documentation, this is actually a velocity term. One will see that combining the time and position redefintions leads to $\mathbf{u}' := \mathbf{u}/c$.

3.2 Plasma

The plasma density profile is roughly shaped like a double sigmoid; there are a number of functions in this family. We opted for a formulation using the hyperbolic tangent, starting from the equation

$$n(x) = \frac{1}{4}\left(1 + \tanh\left(\frac{x - x_0}{k_1}\right)\right)\left(1 + \tanh\left(\frac{x_0 + L - x}{k_2}\right)\right) \tag{4}$$

where x_0 is where the plasma starts, L is the plasma length, and k_1, k_2 are the up- and down-ramp lengths respectively. These lengths are nominal, since as this function is continuous, these must mean approximate lengths, or idealisised lengths.

(4) also needs to be normalised, to ensure that the peak value is always unity. Therefore, our true plasma density distribution is

$$\hat{n}(x) = \frac{n(x)}{n(a)} \tag{5}$$

where a is the root of $\mathrm{d}n(x)/\mathrm{d}x = 0$.

There are downsides to this formulation:

- the aforementioned nominal lengths make analysis a bit harder;
- it is poorly behaved for modelling purposes when $L \sim k_1$ or $L \sim k_2$.

It is best behaved for analysis when $k_1 \sim k_2$, and $L \gg k_1 + k_2$.

Alternatively, to make these lengths 'true', one could opt for a piecewise function such as

$$n(x) = \begin{cases} \frac{x-x_0}{k_1} & x_0 \leq x < k_1 + x_0 \\ 1 & k_1 + x_0 \leq x < k_1 + L + x_0 \\ -\frac{x-(x_0+k_1+k_2+L)}{k_2} & k_1 + L + x_0 \leq x < k_1 + k_2 + L + x_0 \\ 0 & \text{otherwise} \end{cases}. \tag{6}$$

3.3 Laser

A number of laser parameters are strictly necessary for the system. In particular, the laser wavelength λ_z[1], and the (peak) laser intensity, I_0. These determine the 'laser strength parameter' a_0, given by I_0 is typically given in $\mathrm{W\,cm^{-2}}$, and the wavelength in µm, giving a rough approximation

$$a_0 \sim 0.854\,93 \times 10^{-9} \lambda_z \sqrt{I_0}. \tag{7}$$

4 Usage

References

[1]The subscript z comes from the notation 'z-pulse' within OSIRIS