Ain Shams University

ASU - Dynamic Systems & Digitalisation Cluster

Center for Sound, Vibration & Smart Structures

Mapping of artificial acoustic emission sources on wind turbine blades

Internship Report

By
Mohamed Rezk
Undergraduate Computer Engineering

Supervised by
Dr.Ahmed Hesham
Ph.D Mechanical Engineering

Contents

Part I

Fundamentals of AE

0.1 Introduction

Acoustic Emission is kind of non-destructive passive testing, where the signal is detected by surface-mounted piezoelectric sensors as opposed to ultrasonic active testing where the signals are emitted through the specimen and transmitted back. To acquire such signals, the component under test is submitted to external stimuli, such as loads , high pressure, or relatively high temperature. As the damage grows through the component, more energy is released in the process. This kind of testing is generally used for evaluating the structural integrity of certain component to monitor defects like fatigue cracking, fiber break, and corrosion progression through the material. Also, being non-destructive, AE testing allows large structures to be monitored while they are operating causing minimal disruption.

0.2 Signal Parameters

Analysis of AE data is quite difficult as the number of data point produced from a single test is large. So Extracting features from the waveform allows comparisons between different set of data.

Amplitude: is the maximum measured voltage across the waveform. It is an important parameter as it determines the parts of the wave which will not be included in the analysis (some points do not cross the threshold defined based on the value of the amplitude).

Time of arrival : regarded as the most import feature in the waveform. ToA determines the time that sensor began to sense the wave coming form the AE event. They are some method to compute ToA, like threshold crossing and Akaike information criterion (AIC), but they will be discussed in detail later.

There are some other important signal parameters like: rise time , counts , duration; however, they are irrelevant to the scope of this project.

0.3 Data Acquisition Setup

The setup is quite easy, just four sensors placed on a known size grid. When the hit takes place each of the sensor sends its received signal to the corresponding channels to be filtered as some noise might interfere. After that the signal get recorded by

a specialized acquisition software to be further exported to CSV files , for example. This short journey can be summarized by the following figure.

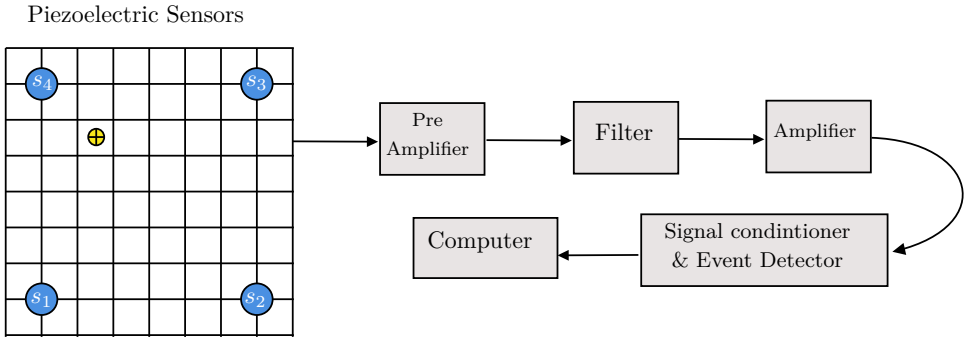


Figure 1: Data Acquisition Setup

Part II

Localization

This part of the project takes care of the most important aspect of Acoustic Emission inspection, i.e the Localization of the AE event. There are several techniques to handle such tasks; however, we are going to limit ourselves to most basic one,Time Difference of arrival. TDoA alone will not be much more beneficial as some noise and reading error are involved in the processes. As a result, an optimization algorithm has to arise to solve such a problem (all about it in optimization part). As all the materials have not the same inside-structure nor all directions within the same material are the same, we have to put some assumptions to help us unify our model.

0.4 Assumptions

- Wave path is constant; otherwise, the solution is not going to be the same for a given set of data.
- Wave velocity is the same everywhere on structure.
- Structure thickness is uniform. If not, a third dimension would be involved, which only complicates the model
- Material is homogeneous.

0.5 Techniques

There are plenty of methods to locate the source of an acoustic emission. The most basic ones are Time of arrival (ToA), do not confuse with arrival time which is signal parameter, and Time difference of arrival (TDoA).

0.5.1 Time of Arrival (ToA)

This is a fairly easy technique to implement and derive a general equation that describes this simple behavior. ToA is based on knowing the exact time the signal has been sent and the exact time the signal has reached a reference point in space, which could be a sensor. Also, knowing the velocity of signal through the material, we can compute the the distance between the event position and the reference point through the following equation:

$$d = velocity.(t_{arrival} - t_{sent})$$

Nevertheless, the equation above can be simple in the 1-D case, it yields an equation for circle in two dimensions.

$$d_i = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2}$$

where, (x_i, y_i) is the location of the i^{th} sensor on the plane and (x_s, y_s) is the AE source unknown location.Once this equation is calculated for enough reference points (sensor locations) on the same plane, the location of the AE event cab be easily obtained from the intersection point of the circles. It has to be noted that at least 3 sensors are required for the two dimensions to kill the symmetry around the line joining the original two sensors, in the same way 4 sensors for 3-D. Despite its simplicity, ToA can have some accuracy-relates issues. The first sub-figure represents the ideal case, where all the circles intersect in one and only one point (the target), but this rarely happens, the second sub-figure, on the other hand, shows the case where the circle do not meet in any point; however, there is a margin of error we can search in. This problem arises due to some issues in time synchronization and acquisition errors.

(a) Ideal case of ToA solution

(b) non-ideal case of ToA

Figure 2: ToA solution visuliazation

0.5.2 Time Difference of Arrival (TDoA)

TDoA is the second most popular technique, it is somehow more versatile than ToA as this method does not require the time that the signal was sent from the AE source, only the arrival time and signal speed are needed to localize the source. Once two

sensors at least (in 1-D) receives the signal, we can compute the difference in arrival time and hence, the difference of distance between the sensors and target. This can follow the upcoming equation:

$$\Delta d_{i,j} = v \ . \ \Delta t_{i,j}$$

In the 2-D case, however, the equation can be written as follows:

$$\Delta d_{i,j} = \sqrt{(x_i \ - x_s)^2 + (y_i \ - y_s)^2} - \sqrt{(x_j \ - x_s)^2 + (y_j \ - y_s)^2}$$
$$where: \quad 1 \leq i, j \leq n \quad , \ i \neq j$$

And n is the number of sensors in the sensor array. DToA is quite hard to solve as there is not an obvious closed form solution . The solution can rely on an optimization algorithm, which , in turn, tries to minimize the difference between the calculated time difference and the observed one:

$$f(x, y) = \sum_{i=1}^{n} (\Delta t_{i,obs} - \Delta t_{i,clc})^2$$

Where:

$$\Delta t_{i,obs} = t_i - t_1$$
$$\Delta t_{i,clc} = \left[\sqrt{(x_i \ - x_s)^2 + (y_i \ - y_s)^2} - \sqrt{(x_1 \ - x_s)^2 + (y_1 \ - y_s)^2} \ \right] / v$$

So our main goal is :

$$\min_{x,y \ \in \ R^2} \{f(x, y)\}$$

And this will be discussed in the optimization part. Now, the important question is how to compute t_i for a given waveform. Some methods are going to be introduced in the next section.

0.6 Computation of arrival time

0.6.1 Threshold Crossing

Generally, there are two main techniques used to compute the arrival time of the signal to a given sensor. The first one is threshold crossing, where a fraction α of the maximum amplitude is set as a minimum threshold, and then we have to iterate over the data points until we have the first crossing. It is somehow a simple approach, and not computationally expensive. In fact its complexity is fraction of n or formally $\theta(n)$, where n is the number of samples in the waveform.

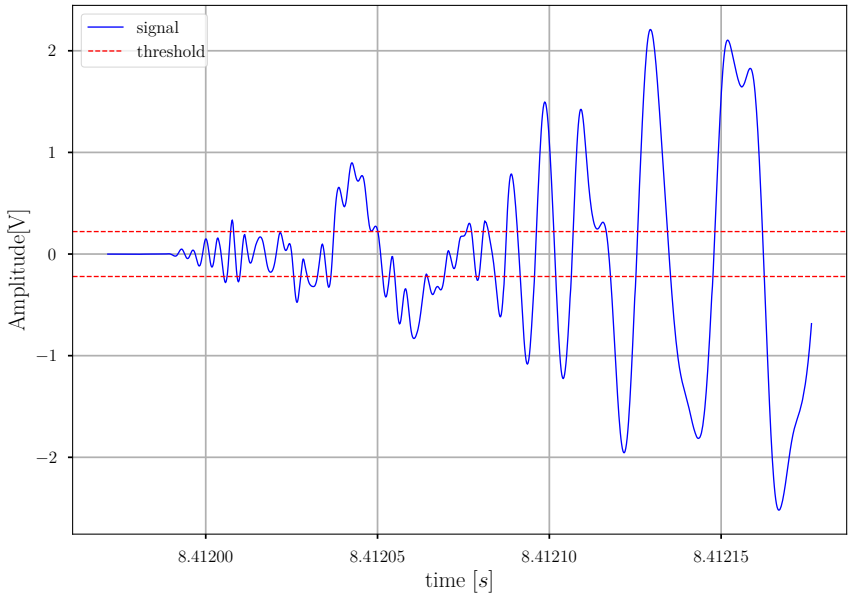


Figure 3: Threshold Crossing implementation

The downside of this method is accuracy as it might skip some data points , hence an increase in arrival time. That is because it is very sensitive to the parameter α , and here comes the second approach (AIC).

0.6.2 Akaike Information Criterion (AIC)

The Akaike Information Criterion (AIC) function was first originated in information theory and used as an estimator for relative quality of different statistical models. It can be useful to compute the arrival time of the signal to the sensor of interest

through the following equation:

$$AIC(t) = t.\log_{10}\left(var(x[1:t])\right) + (T-t-1).\log_{10}\left(var(x[t:T])\right)$$

where T is the signal duration, and var is the classical statistical variance. Since the function splits the signal into two-time vectors $[1:t]$ and $[t:T]$ and describes the similarity between them. When point t is aligned with the signal onset, the vector prior to t contains only high-entropy uncorrelated noise and the vector after t contains only low-entropy signal with marked correlation and the AIC function therefore returns a minimum. on the contrary, AIC is computationally expensive as it splits the waveform into two vectors for every time point, in fact, its algorithmic complexity can be formulated as $\theta(n^2)$. This can affect the runtime of the program especially if it is utilized in real-time; however, there is a little trick, i.e. caching the AIC data in (for example) a JSON file and loading the data after computing the function for just one time per sensor data. AIC proved useful with reasonable accuracy measures when it is compared to the threshold crossing. To sum up, there is a trade off between accuracy and computational time in two methods.

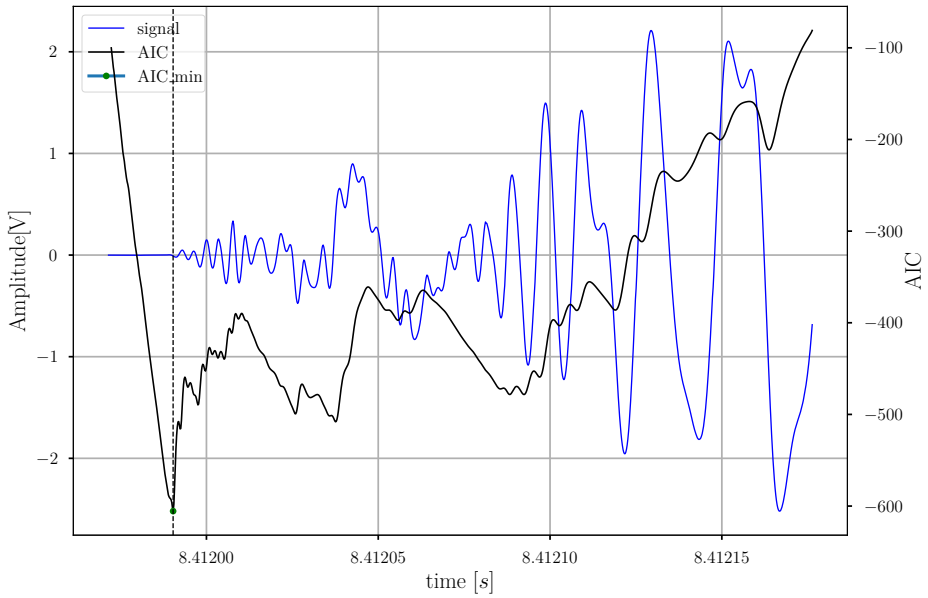


Figure 4: AIC implementation

Part III

Optimization

Particle Swarm Optimization

Particle swarm optimization (PSO) has been successfully applied to wide variety of problems in both of scientific and industrial fields alongside with other swarm intelligence based techniques. It undergoes beneath the umbrella of meta-heuristics where the global solution is not guaranteed to be found for some class of problems , but optimal or local solution can be easily found with few assumptions and lower computation resources. Also, PSO is stochastic in its nature which means the solution is not going to be the same for each time the algorithm gets executed ; however, based on the bench-marking and some handful testing, the final result fall contained within small region inside the solution space.

PSO proved to be so efficient in terms of computational time and memory compared to other methods. It also could be parallelized, but it is out of this project's scope. The downside, however, is that PSO has some hyper-parameters that have to be tuned to get reasonable solution within small margin of error. This section will provide a mathematical and a computational model of PSO.

0.7 Introduction

To understand the basic idea behind PSO, we have to know its origin. This stochastic method was proposed by Dr. Eberhart and Dr. Kennedy in 1995 inspired by the social behavior of birds or groups of fish.

Imagine that there is a group of birds trying to find food which is located only in one spot, none of the birds know exactly where the food is ; however, they have an idea of how far away might the food be. So, they have to cooperate intelligently to find the food spot.

In the same way, we have a bunch of particles (population), each one represents

a potential solution. None of the particle knows where is the global minima of the objective function, but each particle has memory of best visited location based on the value of the cost (objective) function. Within each generation (new population), the best known position to the population is updated for each iteration based on the best fitness evaluated among this particular population.

0.8 Mathematical Model of PSO

Each particle has its own position vector $\vec{x_i}(t)$ and a velocity vector $\vec{v_i}(t)$ to update the position each iteration along side with the particle's best known position $\vec{P_i}(t)$ and population's best know position $\vec{g}(t)$. This can be modeled with simple node has the aforementioned vectors and and another node representing the new state of the particle after one iteration.

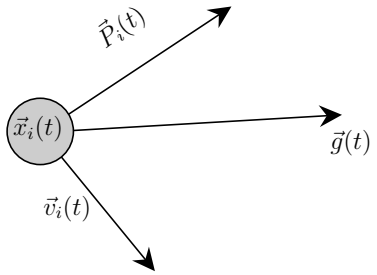


Figure 5: Initial state of particle at point of time t

Each iteration, the position of the particle has to be updated, to do so, the velocity vector must be updated first.

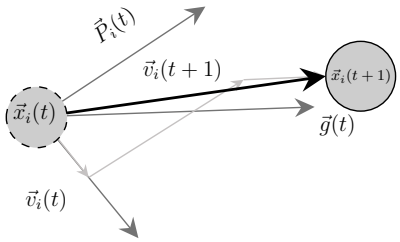


Figure 6: Initial state of particle at point of time t

As the figure shows, the velocity of each particle with a population can be updated as follows:

$$\vec{v_i}(t+1) = w.\vec{v_i}(t) + r_1.c_1.[\vec{P_i}(t) - \vec{x_i}(t)] + r_2.c_2.[\vec{g}(t) - \vec{x_i}(t)]$$

Where:

- w is the inertia weight (a scalar multiple of the velocity vector)
- r_1 and r_2 are uniformly random numbers between 0 and 1 $\rightarrow U[0,1]$
- c_1 is cognitive weight (how the particle tends to be positioned towards the its best location)
- c_2 is social weight (how the particle tends to be positioned towards the global best location)

As we have a finite search space, the velocity vector has to fall within a range:

$$\vec{v}_i(t+1) \in [\vec{v}_{min}\ ,\ \vec{v}_{max}]$$

If the condition does not prevail, the velocity is set to the extreme value it has surpassed weather minimum or maximum.
After updating the velocity, the position vector can be update as follows :

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1)$$

Following the last step, the best known position for the i^{th} particle can be improved if the the following condition holds :

$$f(\vec{x}_i(t+1)) < f(\vec{P}_i(t)) \implies \vec{P}_i(t+1) = \vec{x}_i(t+1)$$

The same way applies for the global best known position:

$$f(\vec{P}_i(t+1)) < f(\vec{g}(t)) \implies \vec{g}(t+1) = \vec{P}_i(t+1)$$

Surely, this is a simplified yet descriptive explanation of PSO; however, some additional features can be added that can assist in fast convergence to solution like simulating the death of a particle , which can be described by :

$$\vec{x}_i(t+1) \notin [x_{max}\ ,\ x_{min}] \implies \vec{x}_i(t+1) = rand([x_{max}\ ,\ x_{min}])$$

Fairly simple but still powerful to consider.

0.9 PSO PseudoCode

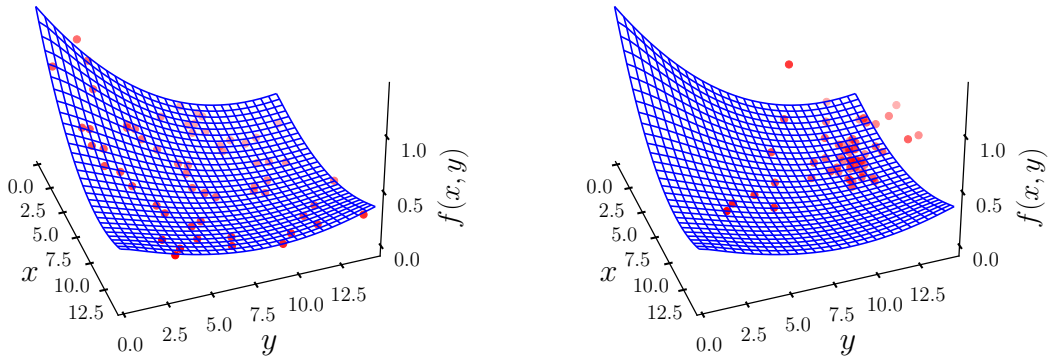
The pseudo-code of this algorithm is quite easy, we just have to compile the equations stated above, all together yield :

Randomly initialize Swarm population of N particles $\vec{x}_i, 1 \leq i \leq N$
Select hyperparameter values w, c_1 and c_2
for $k \leftarrow 1$ to *generations* **do**
  for $i \leftarrow 1$ to N **do**
    a. Compute new velocity of i^{th} particle
    $\vec{v_i}(t+1) = w.\vec{v_i}(t) + r_1.c_1.[\vec{P_i}(t) - \vec{x_i}(t)] + r_2.c_2.[\vec{g}(t) - \vec{x_i}(t)]$
    b. Check new velocity bound
    if $\vec{v}_i(t+1) < \vec{v}_{min}$ **then**
      $\vec{v}_i(t+1) = \vec{v}_{min}$
    else if $\vec{v}_i(t+1) > \vec{v}_{max}$ **then**
      $\vec{v}_i(t+1) = \vec{v}_{max}$
    end if
    c. Update the position of i^{th} particle
    $\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1)$
    d. Update new best of this particle and new best of Swarm
    if $f(\vec{x}_i(t+1)) < f(\vec{P}_i(t))$ **then**
      $\vec{P}_i(t+1) = \vec{x}_i(t+1)$
    end if
    if $f(\vec{P}_i(t+1)) < f(\vec{g}(t))$ **then**
      $\vec{g}(t+1) = \vec{P}_i(t+1)$
    end if
  end for
end for

0.10 PSO Implementation

PSO can be easily implemented in python relying on OOP paradigm. It is very suitable for the purpose and and the algorithm can be further extended to adopt more techniques. I have implemented a particle class which contains the particle's position, velocity, fitness, best position , and best fitness. That is besides a solver

class to properly transfer the data between different routines and export the necessary data for visualization. The following two figures are snapshots from the animation. In the first sub figure, the particle positions had been assigned randomly; after some iterations (generations), the particles tended to collaborate until they reached the minima of the objective function as clearly shown in the second sub-figure.



(a) Intial state of particles

(b) Final state of particles

Figure 7: PSO Animation

Simplex Optimization

0.11 Inroduction

The simplex Nelder-Mead algorithm (Amoeba search) is widely used to solve unconstrained function minimization problems even the discontinuous ones or subject to noise. The algorithm is an iterative process, meaning, it tries to replace the worst solution with a better one and reorder the simplex nodes. Each iteration, the search area is moving and deformed based on some rules. To understand the structure of the algorithm more closely, we begin with introducing some helpful stuff. Our main goal is :

$$\min_{\vec{X}\in\mathbb{R}^n} f(\vec{X})$$
$$f : \mathbb{R}^n \to \mathbb{R}$$

Where, f is the objective function we have discussed earlier and n is the dimension. And the simplex is a geometric figure that lives in that n dimension with $n+1$ vertices. In 2-D, it is a triangle, so we are going to denote it by Δ for simplicity and convenience.

$$\Delta = (\vec{x}_{best} \ , \ \vec{x}_{other} \ , \ \vec{x}_{worst})$$

where,

- $\vec{x}_{best}$ is the best solution within the vertices
- $\vec{x}_{worst}$ is the worst solution within the vertices
- $\vec{x}_{other}$ is the solution in between
  That is based on the fact that :

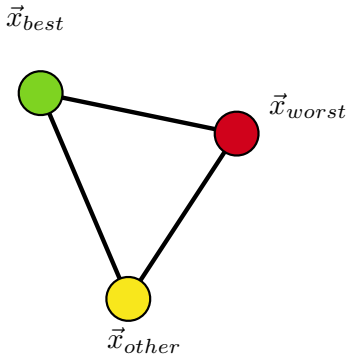


Figure 8: Initial state of particle at point of time t

$$f(\vec{x}_{best}) \leq \quad f(\vec{x}_{other}) \ \leq \ f(\vec{x}_{worst})$$

The goal now is to adjust the parameter values of the worst point so that the simplex moves towards the minimum of the function with the direction of moving determined by the centroid.

0.12 Mathematical Model of Simplex method

Compute Centroid

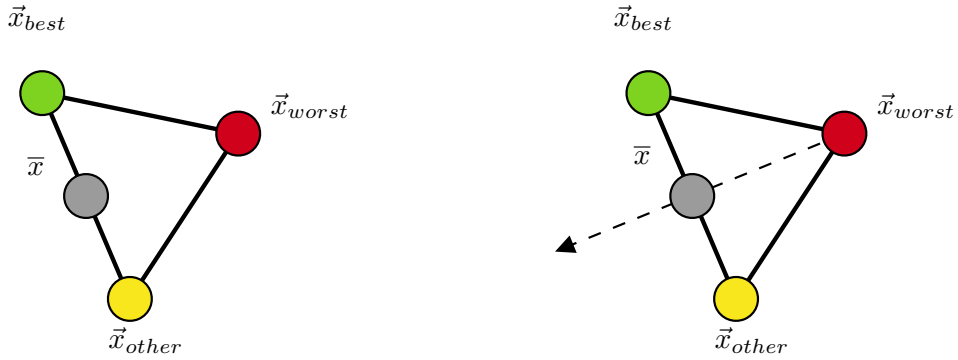
The centroid represents the geometric mean between two points. It is calculated for every pair of vertices in the simplex except for the worst point.

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

In our case $n = 2$, the centroid is calculated for $\vec{x}_{best}$ and $\vec{x}_{other}$ only through the following simple equation:

$$\overline{x} = (\vec{x}_{best} + \vec{x}_{other})/2$$

Now, all the operation we are going to execute on the simplex will take place with respect to the direction of vector $(\overline{x} - \vec{x}_{worst})$, second sub-figure.



(a) Intial state of particles (b) Final state of particles

Figure 9: PSO Animation

Reflection

Reflection is the first operation we make on the simplex data structure. It is an attempt to see whether the value of the function at the reflected point is less than the worst solution. It can be computed through the upcoming equation.

$$\vec{x}_r = \overline{x} + \alpha(\overline{x} - \vec{x}_{worst})$$

where α is the reflection coefficient, positive real-valued parameter , typically $\alpha = 1$.

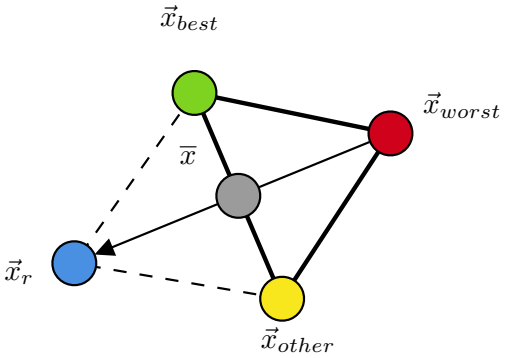


Figure 10: Initial state of particle at point of time t

If the objective function at the reflected point is less than the other point, but not the best, we set the reflected point to the worst.

$$f(\vec{x}_{best}) \leq \quad f(\vec{x}_r) \;<\; f(\vec{x}_{other}) \quad \Longrightarrow \vec{x}_{worst} = \vec{x}_r$$

And we reorder the vertices as we mentioned above; in fact, we have to reorder the vertices of the simplex after each operation, which is the whole point of the algorithm.

Expansion

If it happened and the reflected point is better than the best point (in terms of fitness) $f(\vec{x}_r) < \quad f(\vec{x}_{best})$, maybe we can find better point a bit further from the reflected point, that is the idea of expansion.

$$\vec{x}_e = \vec{x}_r + \beta(\vec{x}_r - \overline{x})$$

where β is the expansion coefficient, with $\beta > 1$, typically $\beta = 2$.

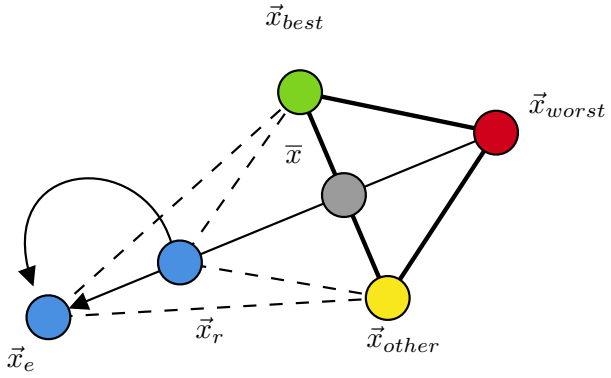


Figure 11: Initial state of particle at point of time t

Again, we check the quality of the new point $\vec{x}_e$, if it is better than the reflected point, then we are in the right direction, if not we assign the reflected point to the worst point normally which means, we have done a bad attempt .

$$f(\vec{x}_e) < \quad f(\vec{x}_r) \quad \Longrightarrow \vec{x}_{worst} = \vec{x}_e$$
$$f(\vec{x}_e) \geq \quad f(\vec{x}_r) \quad \Longrightarrow \vec{x}_{worst} = \vec{x}_r$$

Contraction

If the reflected is worse from the beginning, then we can minimize the damage by contracting the simplex whether inside or outside, whichever is better.

Inside contraction when the reflected point is worse than all including the worst point, we try to contract the simplex towards the worst point, perhaps we get a better solution.

$$f(\vec{x}_r) \geq \quad f(\vec{x}_{worst})$$
$$\vec{x}_c = \overline{x} + \gamma(\vec{x}_{worst} - \overline{x})$$

where γ is the contraction coefficient, with $0 < \gamma < 1$, typically $\gamma = \frac{1}{2}$.
if the fitness value is better at the contracted point than the worst point, we replace the worst point with the contracted one, if not we moved in the wrong direction, maybe we have to go backwards.

$$f(\vec{x}_c) < \quad f(\vec{x}_{worst}) \quad \Longrightarrow \vec{x}_{worst} = \vec{x}_c$$

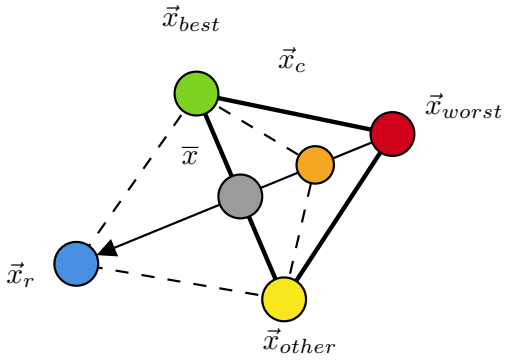


Figure 12: Initial state of particle at point of time t

Outside contraction When the reflected point is, however, worse than the "other" point , but better than the worst, we can contract in the outside direction (towards the reflected point).

$$f(\vec{x}_{other}) \leq \quad f(\vec{x}_r) \; < \; f(\vec{x}_{worst})$$
$$\vec{x}_c = \overline{x} + \gamma(\vec{x}_r - \overline{x})$$

And we also check weather we are going to accept the new point or not.

$$f(\vec{x}_c) < \quad f(\vec{x}_r) \quad \Longrightarrow \vec{x}_{worst} = \vec{x}_c$$

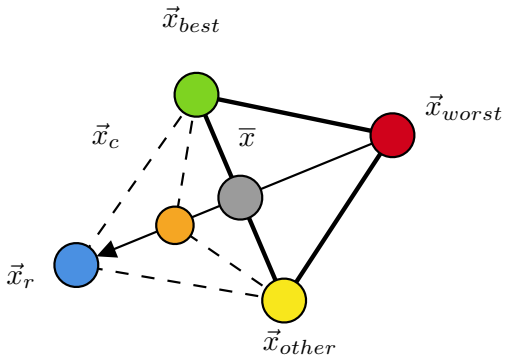


Figure 13: Initial state of particle at point of time t

Shrinking The idea of shrinking is to minimize the damage from a failed contraction, that is when $f(x_c) > \min[f(x_{worst}), f(x_r)]$, This can be done by moving the worst point and the other point towards the best point by :

$$\vec{x^*}_{worst} = \vec{x}_{best} + \delta(\vec{x}_{worst} - \vec{x}_{best})$$
$$\vec{x^*}_{other} = \vec{x}_{best} + \delta(\vec{x}_{other} - \vec{x}_{best})$$

where δ is the contraction coefficient, with $0 < \delta < 1$, traditionally $\delta = \frac{1}{2}$.

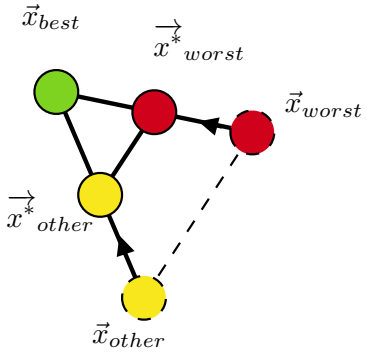


Figure 14: Initial state of particle at point of time t

We do all of the previous until we reach the predefined maximum iterations or we

cannot do better by reaching certain threshold, which can be done by checking the convergence each iteration

Convergence We say that , the simplex has converged to the best solution if it is sufficiently compact meaning that the distance between the each pair of vertices is adequately small.

$$\sigma = 2 \; . \; \left[\frac{f(\vec{x}_{worst}) - f(\vec{x}_{best})}{f(\vec{x}_{worst}) + f(\vec{x}_{best}) + \epsilon} \right]$$

where ϵ is a tolerance defined based on the nature of the problem. Each iteration we check if :

$$\sigma < \epsilon$$

if not we proceed until the condition prevails.

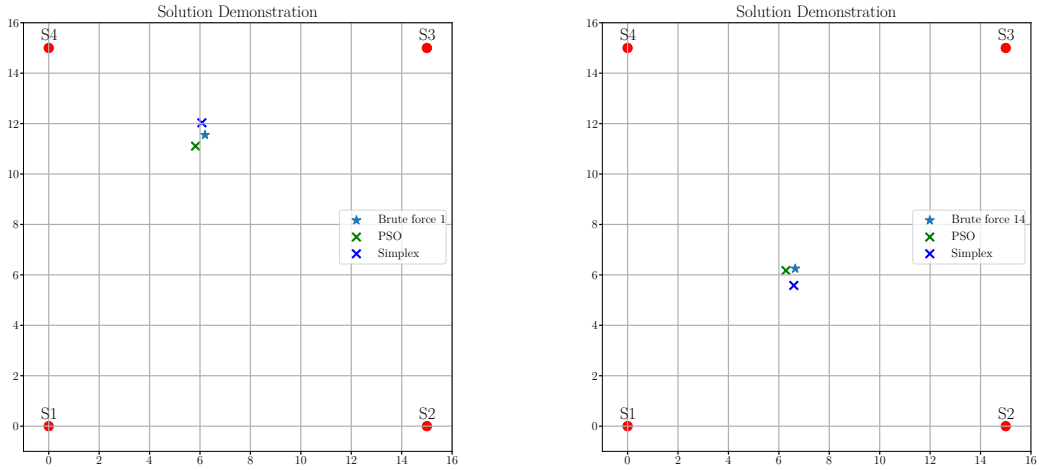
Part IV

Results

For the sake of clarity, the two algorithms have been tested for number of iterations and for different hit IDs. To ensure the quality of the optimization method used I have implemented a discrete brute force approach to compute the deviation in both x and y.

0.13 General Testing

The following figure shows the localization done by the particle swarm optimization and the simplex method, besides ,of course, the brute force.



(a) Localization demo

(b) Different Hit

Figure 15: Localization Demo for Two hits

0.14 Deviation Test

Then, we test the deviation of the two algorithms from the brute force to see a visual trend over number of iterations. For the deviation in y and x:

$$d_i^x = x_i - x_{bf}$$

$$d_i^y = y_i - y_{bf}$$

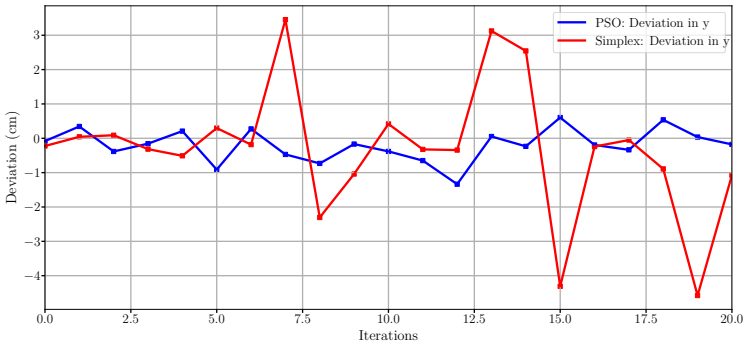


Figure 16: Deviation in y

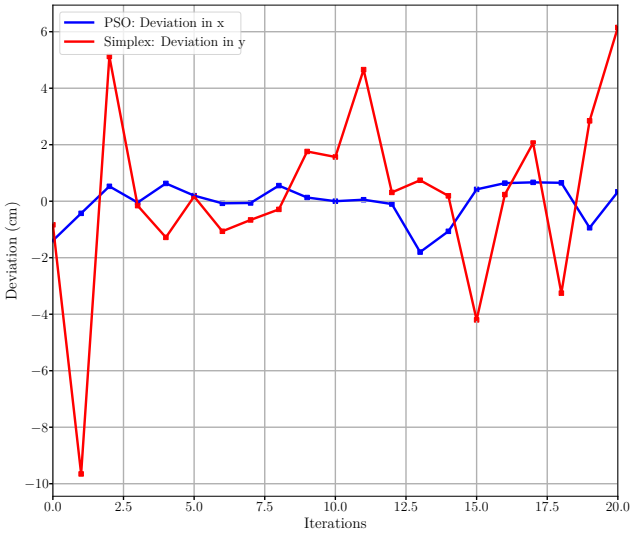


Figure 17: Deviation in x

0.15 Thoughts

Clearly, the PSO has an incredible advantage over simplex optimization in terms of deviation from the true location of the source. But let us take a moment here and analyze the situation more closely.
The algorithmic complexity of PSO is :

$$\theta(n \times m) \quad , \; m = generations \quad , \; n = particles$$

as we have to update every particle within a population over the course of the generation. For instance, I have set the number of generations $(m)$to 100 and number of

particles n to 60 which means around 6000 iterations have been consumed. On the other hand, the simplex method is linear. In fact, its algorithmic complexity can be formulated as :

$$O(n) \quad , \; n = maximim \; iterations$$

Which means that the algorithm can consume them all or any fraction of this depending on the convergence check we discussed last section. However, In average, the simplex optimization took 52 iteration to finish (no longer can better the solution), which considered amazing in terms of computational resources. PSO is superior to simplex optimization due to the swarm intelligence, thanks to the stochastic nature. Nevertheless, we can blend the two algorithms to exploit the fast convergence of the simplex method with the stochastic nature of the particle swarm, which can make (and i conjuncture) a stable algorithm.

Part V

Conclusion and Recommendations

0.16 Conclusion

Acoustic emission testing proved to be valuable in ensuring the structure integrity, and monitoring the health status of different bodies, e.g. wind turbine Blade.
The localization part of the acoustic emission is very interesting. It opens the doors to implement and test different optimization algorithms. Some are subtitle for lower computational resources , some convergences quickly, and so on . We can even blend some together to get the desired solution. Also, some techniques can be formulated from the scratch to fit the problem nature, like Delta-T mapping. As we say "All roads lead to Rome", but the important question still is *"when this method shall be used"*.

0.17 Recommendations

My recommendations for whom is going to work in similar project can be summarized in the following points:

- Hybrid algorithms tend to be more stable and converge fast.
- Implement simulated annealing or genetic algorithm, both have great performance.
- Tweak the AIC to take less time or implement a similar method.
- Implement a more structured noise filter for attenuation data.
- Avoid many loops which affects the software performance.

Part VI

References

Bibliography

[1] Francesco Ciampa and Michele Meo. Acoustic emission source localization and velocity determination of the fundamental mode a0 using wavelet analysis and a newton-based optimization technique. *Smart Materials and Structures*, 19(4): 045027, 2010.

[2] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[3] Donald M Olsson and Lloyd S Nelson. The nelder-mead simplex procedure for function minimization. *Technometrics*, 17(1):45–51, 1975.

[4] V Salinas, Y Vargas, J Ruzzante, and L Gaete. Localization algorithm for acoustic emission. *Physics Procedia*, 3(1):863–871, 2010.

[5] Christopher B Scruby. An introduction to acoustic emission. *Journal of Physics E: Scientific Instruments*, 20(8):946, 1987.

[6] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.