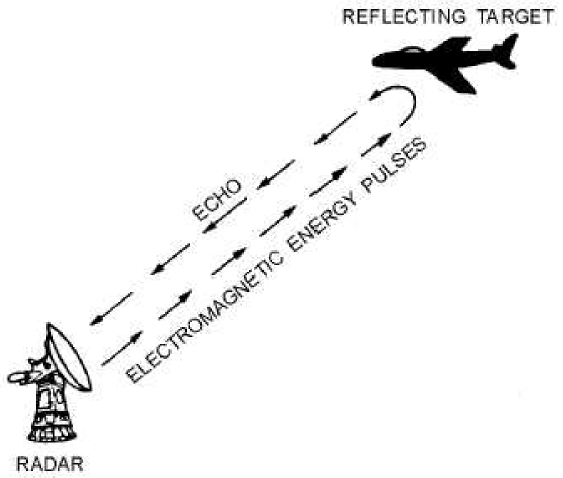
PIC Math Project – Accelerating Search for Low-Peak-Sidelobe Radar Codes

Gregory E. Coxson
United States Naval Academy
Annapolis, MD

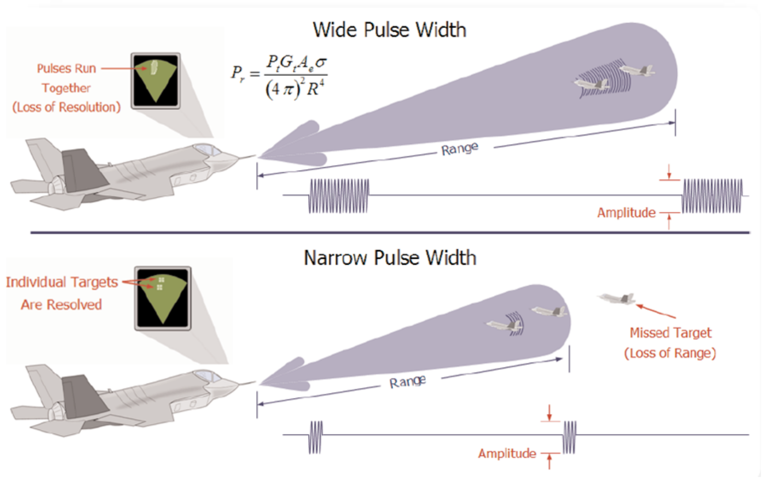
January 10, 2016

1 Motivation

Pulse compression dates to the early days of radar, during the build-up to World War II. The problem is motivated by the practical concern of achieving optimal detection of in-coming targets.

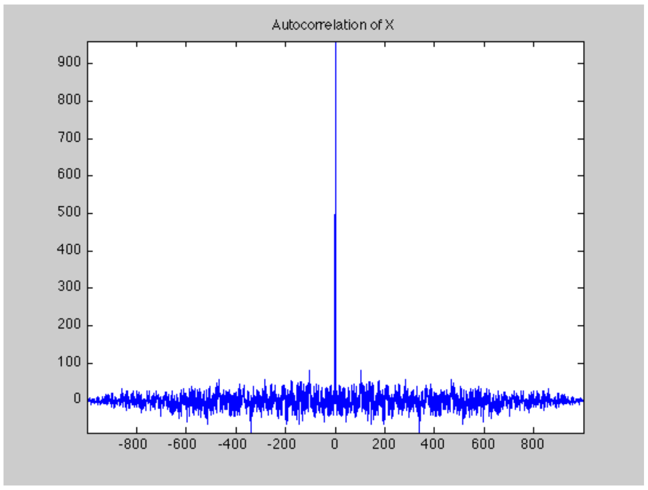


Radar signals tend to be sent as pulses of fixed power; the wider the pulse the farther out targets can be seen (because the pulse hits the target with more energy, so more energy can bounce back to the radar) but wider pulses make it harder to resolve two close-together targets.


Wide Pulse Width
Pulses Run Together (Loss of Resolution)
$P_r = \frac{P_t G_t A_e \sigma}{(4\pi)^2 R^4}$
Range
Amplitude
Narrow Pulse Width
Individual Targets Are Resolved
Missed Target (Loss of Range)
Range
Amplitude

Those early radar designers wanted both long range and good resolution of pairs of targets in close proximity. Pulse compression solved this problem, allowing both objectives to be achieved at the same time.

Pulse compression requires encoding these pulses, and this is where the Mathematics comes in. Radar engineers designing code sets for pulse compression need codes having certain nice properties in relation to the filter used when the radar receiver captures the return signal. The picture below shows what is needed after pulse compression – a sequence with a narrow spike indicating exactly the target range (equivalently, time delay) and everywhere else, near-zero outputs.



If the peak is high and the other elements (called sidelobes) are much lower, the radar detection processor can apply a simple threshold test. If after pulse compression, a peak appears above the threshold, the processor determines that a target is present; if not, then it decides there is no target present. The possibility will always exist that noise or other effects combine to give a "false alarm", which happens when the radar decides there is a target present but none exists, or to give a "missed detection", when it decides there is no target when one is present. This is why higher detection performance is desired; good-quality codes are a key part of optimizing performance.

It is possible to abstract the process of finding good codes to an mathematical and computational exercise. The essence of the problem is to find N-length binary codes in a search space of size 2^N achieving the lowest possible value of the peak sidelobe level, or PSL, of the autocorrelation function. There are limits to how low the PSL can go, but in reality what is needed more than low PSL is a ratio ratio of peak (which can be assumed to be roughly the code length N) to PSL. Achieving high peak to PSL ratios is easier with longer codes, and there is the rub – finding really good codes gets harder as the code length grows. That is why, 80 years after radar began development in earnest, the search for good codes, and methods to find good codes, continues.

Because the optimization surface for this problem is far from smooth, the task of finding the optimal codes (that is, those that achieve the minimum peak sidelobe for a given length N) cannot benefit from elegant and powerful numerical methods available for "nicer" problems. Most of the time, the last resort, or brute-force exhaustive search of the entire search space, is called upon out of a combination of necessity and pure frustration. Unfortunately, as N grows, computational effort for brute-force exhaustive search tends to grow exponentially with the code length N.

Having given the darkest and scariest prognosis, it is important to add that there are options for making exhaustive search faster. This project will focus on one of them – exploitation of symmetries of the autocorrelation function that lies at the heart of radar pulse compression.

The following theorem in Farris [8] defines a symmetry for a function of a single complex variable z (that is, $z \in C$, where C represents the set of complex numbers):

Theorem 1. We call a transformation of the complex plane, $\alpha(z)$, a symmetry of the function $f(z)$ if and only if

$$f(\alpha(z)) = f(z)$$

for all

$$z \in C.$$

By the way, if you find it fun to work with symmetries, as you will be doing in this project, Farris's book provides an interesting way to take your exploration to another level (Farris explores how to build symmetries into complex-valued functions of complex variables).

We will be dealing not with a function of a single complex variables, with a function of sequences of binary or unimodular numbers. Let's start by defining a binary ± 1 number to be $x \in \{1, -1\}$. A binary sequence of length N will be understood to be of the form $x = \{x_1, x_2, \ldots, x_N$ where $x_i \in \{1, -1\}$ for

$i = 1, \ldots, N$. We will also be interested in unimodular sequences whose elements are drawn from the unit circle. Let the unit circle be represented by

$$T = \{z \in C : ||z|| = 1\}.$$

A unimodular sequence is $z = \{z_1, z_2, \ldots, z_N\}$ where $z_i \in T$ for $i = 1, \ldots, N$. A useful special case of the unimodular sequences are the m^{th}-root-of-unity sequences, where $z_i^m = 1$ for $i = 1, \ldots, N$.

As we will see, in the binary case, the autocorrelation function has three symmetries. These may be exploited to improve the computational cost for binary-code exhaustive searches for any length N, as you will discover in this project. What is not known is whether this can be done for unimodular sequences. So the second stage of this project will be to venture into the unknown and ask whether something similar can be done for exhaustive searches for mth-root-of-unity sequences for given exponent m and length N.

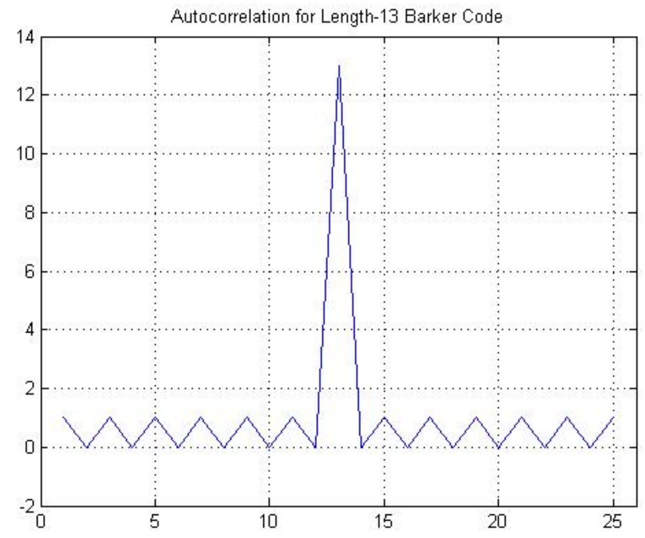
This document is organized as follows. The next section will discuss the autocorrelation function in more detail. This will be followed by a section defining and discussing the peak sidelobe level, the key metric used in this project. The section after will discuss the search space for binary codes. Then, a section for efficient listing of codes by using autocorrelation function symmetries and hexadecimal representation. The next section discusses the Barker codes, which achieve the best peak sidelobe level achievable. This is followed by a section exploiting symmetries for search efficiency. To this point, all discussion relates to binary codes, and there are questions and exercises to give you insight and understanding. Finally, in the last section, we go beyond binary codes to unimodular codes, and indicate the main research challenge.

2 The Autocorrelation Function

Pulse compression was developed in the early days of radar to resolve a trade-off between radar range and ability to resolve closely spaced targets. When the transmit power is fixed, longer ranges are achieved with wide pulses. However, wide pulses mean that returns from pairs of closely spaced targets tend to overlap, limiting the ability to discern the presence of more than one target.

The idea of pulse compression is to partition a radar pulse into N sub-pulses of equal width. Next, multiples of 1 or -1 are applied to each sub-pulse, the pattern giving a sequence that will be referred to as a "binary code" of length N, or more precisely and more verbosely, a "binary ± 1 code" of length N. An encoded transmit pulse is transmitted, bounces off objects, and then a part of it returns to the radar, where it is sampled and a filter applied with taps agreeing with the elements of the reversal of the code. This "Match filter" optimize the signal-to-noise ratio at filter output, allowing the radar signal processor to best pull the signal out of noise. Under reasonable assumptions (see [1]), the response of the match filter is the autocorrelation of the length-N code used to encode the pulse.

The autocorrelation is a symmetric sequence having a central peak, the rest of the elements termed "sidelobes." For best detection performance, the sidelobes need to be as low as possible, and ideally zero (although this ideal is unattainable). The plot below shows the autocorrelation for a length-13 binary code achieving PSL = 1.



Once formulated in terms of binary codes that minimize the maximum of autocorrelation sidelobe size (assuming a metric of choice, often the peak sidelobe level, or PSL), the task of finding good pulse compression codes becomes a mathematical exercise. Given that pulse compression has been around since World War II, one might think that this mathematical exercise has been solved to a great degree. However, as with many research areas, once you start looking and asking questions, open questions become apparent, and even proliferate.

We begin by defining a binary code x of length N as a sequence of elements x_i where $x_i \in \{-1, 1\}$ for $i = 1, \ldots, N$. It will be useful to think of x as a sequence represented as a row vector of length N:

$$x = \{x_1, \ldots, x_N\}.$$

The autocorrelation of a code x is defined as the sequence of length $2N - 1$

$$\mathrm{ACF_x = x * \overline{x}}$$

where $*$ represents aperiodic convolution and $\overline{x}$ means reversal of vector x. The elements $\mathrm{ACF_x(k)}$ for $k = 1 - N, \ldots, N - 1$ may be written explicitly as sums of pairwise products of the elements of x. Then

- For $k = 1, \ldots, N-1$:
$$\mathrm{ACF_x(k)} = \sum_{\mathrm{i=1}}^{\mathrm{N-k}} \mathrm{x_i x_{i+k}}.$$
- $\mathrm{|ACF_x(N-1)| = |x_1 x_N| = 1}$.
- When $k = 0$, $\mathrm{ACF_x(k)}$ represents the peak of the autocorrelation, which equals
$$x_1x_1 + \ldots + x_Nx_N = |x|^2 = N.$$
- For $k = 1-N, \ldots, -1$,
$$\mathrm{ACF_x(-k) = ACF_x(k)}.$$

Using MatLab, given an N-length code x, the operation **xcorr**(x, x) will return the $(2N-1)$-length autocorrelation sequence $\mathrm{ACF_x}$.

3 A Popular Sidelobe Metric – the PSL

A metric often used in radar to compare the performance of different binary codes is the peak sidelobe level, or PSL. It is defined as follows:
$$\mathrm{PSL_x} = \max_{\mathrm{k \neq 0}} \mathrm{|ACF_x(k)|}$$

Question: Given what we know of the autocorrelation function, give a lower bound for $\mathrm{PSL_x}$ for the codes of length N, for any $N \geq 1$.

Question: In terms of the usual norms used in Analysis, which one is being used in defining $\mathrm{PSL_x}$? Define at least one alternative norm to $\mathrm{PSL_x}$.

The peak sidelobe level is often used in radar because what is needed are codes with yield a filter response uniformly below some given level.

Exercise: Write a simple program that returns $\mathrm{PSL_x}$ for a given binary ± 1 code x of any length $N \geq 1$.

While PSL as defined above is necessary non-negative and an integer, radar engineers tend to use the decibel (or dB) scale. To convert to decibels, use the formula
$$\mathrm{PSL_{dB} = 20\log_{10}(PSL/N)}.$$
Here, PSL is compared to the peak of the autocorrelation (the value of ACF(k) for $k = 0$), always equal to the code length N.

4 The Space of Binary Codes of Length N

In radar waveform design, the code length N is often specified. It would be useful to know, for a given N, what is the best achievable value of $\mathrm{PSL_x}$ and the codes x that achieve it. Hence, the space of codes Q_N of binary ± 1 codes of length N.

Question: For a given $N \geq 1$, what is the cardinality of Q_N?

The metric $\mathrm{PSL_x}$ is famously difficult to work with, due to the "fractured" landscape that must be dealt with as x varies over Q_N. To find the best achievable $\mathrm{PSL_x}$ for a given N, there is no more reliable approach, or at least no *known* approach more reliable than exhaustive search. In brute-force, or "naive", exhaustive search, each code $x \in Q_N$ is generated one at a time and $\mathrm{PSL_x}$ calculated and checked against the best codes found so far.

Exercise: Write a computer program to find and return the lowest value of $\mathrm{PSL_x}$ for all $x \in Q_N$, and the codes x that achieves it, for $N = 10$. Provide the computation time as an output.

Question: How many codes did your program check? How many codes did your program find to be optimal?

A useful way to estimate the number of codes in the search space is to use the approximation $2^{10} \approx 10^3$. So, for instance, if $N = 32$, the number of codes that need to be visited is $2^{32} = (2^2) \times (2^{30}) \approx 4 \times (10^3)^3 = 4 \times 10^9$, or 4 billion codes.

Assume you can generalize your computer program to work with any $N \geq 1$ you provide as an input parameter. Then:

Question: Given the processing power of the computer(s) at your disposal, and the fact that the cardinality of Q_N doubles with each *unit* increase in N, graph the computation time needed for your exhaustive search routine as a function of N. How high can you take the value of N and return results from your exhaustive search routine within a minute, a day, or a week?

Can you think of a way to parallelize your exhaustive searches?

5 Listing Your Codes

One challenge, already discussed, is the exponential increase in computer time required for exhaustive searches for minimal-PSL codes of a given length N. As N increases, another annoyance is that the length and the number of codes returned increases. It becomes a chore to list them all in a satisfying way.

One device often used to make the code listings more compact is to use hexadecimal representation. The idea is to group the elements of a code in sub-groups of length 4, starting from the right side of the code. If N is not a multiple of 4, add zeros on the left. In addition, map each code element 1 to 1 and each code element -1 to 0. For instance, the 5-element binary ± 1 code

$$x = \{\ 1 \quad 1 \quad -1 \quad 1 \quad -1\ \}$$

is mapped to

$$x = \{\ 1 \quad 1 \quad 0 \quad 1 \quad 0\ \}.$$

However, 4 does not go evenly into 5, so an appropriate number of zeros are added on the left, yielding the 8-element binary 0/1 code

$$\tilde{x} = \{\ 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0\ \}.$$

Now, map each group of 4 to the corresponding hexadecimal character. The mapping is specified below.

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Now we can give a hexadecimal representation for the length-5 binary ± 1 code above. It is 1A.

We can also shorten our listings of codes by noticing that there are some mappings on codes that leave the peak sidelobe level the same. Consider the form of autocorrelation function again:

$$\mathrm{ACF_x} = \mathrm{x} * \overline{\mathrm{x}},$$

along with the observation that the autocorrelation itself is left-right symmetric. We can see two symmetries right away. That is, negating a code (that is, $\tilde{x} = -x$) preserves PSL. Also, reversal (that is, $\tilde{x} = \overline{x}$) preserves PSL as well.

Question: There is a third PSL-preserving transformation. Can you find it? (hint: this third symmetry preserves PSL, but not ACF).

Exercise: When you find the third "symmetry", improve your exhaustive search routine, from a previous exercise, to employ both of the techniques discussed in this section, to make the output from the routine more compact. You might want to write subroutines to do both hexadecimal-to-binary and binary-to-hexadecimal conversions to add to an available library of subroutines.

Exercise:

1. Find the full set of binary codes with minimal PSL for length $N = 10$. Determining the minimal PSL may need to be part of your search.
2. Reduce the list of optimal codes to a minimal set of representatives relative to the three symmetries that preserve PSL for binary codes.
3. Convert each code in the minimal set to hexadecimal to arrive at a final list of codes in hex.

Question The three PSL symmetries generate a finite group of order 8 under composition of operations. Prove to yourself that it is a group. Is it commutative (that is, Abelian)? Can you identify the group structure?

6 PSL and the Barker Codes

The definition of the autocorrelation function leads to the conclusion that when applied to binary ± 1 codes, it is impossible to achieve PSL < 1. But is PSL $= 1$ possible? Indeed it is; the codes that achieve it are known as the Barker codes. A famous open conjecture asks whether there are any Barker codes of length $N > 13$. By a 1961 result of Turyn and Storer [6], any Barker code longer than 13 would have to be of even length; they proved that $N = 13$ is the greatest odd length for which a Barker code exists.

Exercise: Use your exhaustive search routine to find representatives of all the Barker codes for binary code lengths $N \leq 13$.

If you did this last exercise using three PSL-preserving symmetries, try it again using just the two first symmetries we discussed, negation and reversal. How many Barker code representatives did you find each way? You might notice that for some Barker code lengths, the number of representations is the same either way. What does this suggest to you? For certain Barker code lengths, there is a shared symmetry called "skew symmetry" (coined by the Swiss physicist Marcel Golay). Identify this symmetry. You might need to separate the code lengths into two cases, modulo 4.

7 Search Efficiencies

We have identified some ways to make exhaustive search outputs more compact. But how can we make them faster? In this section, let's discuss a method to consider.

One of the most effective ways to make exhaustive search complete in a shorter time is to find a way to achieve an exhaustive search without checking every code. This may seem a strange idea at first. How is an exhaustive search possible without checking every code in the search space? We will discuss one such approach.

The approach goes back to the idea that the three symmetries generate a group. Let's refer to our three PSL-preserving operations as g_1, g_2 and g_3. Then let's define a relation between two codes x and y of equal length N by the ability to map one to the other using some combination of the three PSL-preserving operators. It is easy to show this is an equivalence relation. It follows that any $x \in Q_N$ belongs to an equivalence class, and that the codes in an equivalence class have equal PSL. So, what if, instead of checking every code in Q_N, you needed only to check a single representative of each of these equivalence class?

Consider the negation operator. The operator partitions the search space into two equal halves. Without loss of generality, one set can have the codes with the first element x_1 set to 1, and the other set can have the codes with $x_1 = -1$. Each code in the first set can be mapped to a code in the other set using the negation operator, and vice versa. It follows that only one of these sets needs to be searched, in order to achieve an exhaustive search. Now combine negation and reversal; can the search effort be reduced further? How about combining all three of the PSL-preserving operations?

Exercise: Find an algorithm which generates, for any code length N, binary codes of length N one-by-one, in such a way that every equivalence class is visited exactly once.

Once you have found such an algorithm, you will want to know how much of a speed-up is achieved. To answer this question, it will be good to know how many equivalence classes there are for any given N. Define E_N to be the number of equivalence classes for length N. The achieved speed-up factor can then be written $2^N/E_N$.

Exercise: Determine E_N for several consecutive code lengths N, say, $10 \leq N \leq 26$. Plot E_N versus N over this range. Make conjectures about the dependence of E_N as a function of N.

A more general version of the last problem would be to find the distribution of equivalence class sizes for each N. The Lagrange theorem of Group Theory implies that every equivalence class size must divide the order of the group, which is 8 in this case [7].

8 Extending to Unimodular Codes

It is natural to try to generalize from binary ± 1 codes to more general classes of codes. The most natural superclass to work with are the unimodular codes, which share with the binary ± 1 codes the property of unit magnitude for code elements. Instead of choosing code elements from the set $\{1, -1\}$, however, elements are chosen from the unit circle. we often restrict attention to the m^{th}-root of unity codes, in which each code element is selected from the set

$$\{\ 1 \quad e^{2\pi i/m} \quad e^{4\pi i/m} \quad \ldots \quad e^{2\pi i(m-1)/m}\ \}.$$

for a given integer $m \geq 2$. After binary codes, the next easiest cases to consider are $m = 3$ (the so-called ternary codes), where the elements are chosen from the set

$$\{\ 1 \quad -(1/2) + i\sqrt{3}/2 \quad -(1/2) - i\sqrt{3}/2\ \}$$

and for the case of $m = 4$ (often called the quad-phase case) the elements are chosen from the set

$$\{\ 1 \quad i \quad -1 \quad -i\ \}.$$

The binary codes are the special case when $m = 2$.

Let's focus on the quad-phase codes since (unlike ternary codes), they include the binary codes. Let's review what carries over from the binary case and what does not. To begin with, hexadecimal representations no longer work. The cardinality of the search space for a given code length N is now $m^N = 4^N$ instead of 2^N. But the definition of the autocorrelation generalizes nicely:

$$\mathrm{ACF_z = z * \overline{z^c}}$$

where z^c represents the result from applying elementwise complex conjugation to an N-length quad-phase code z. The definition of PSL carries over with no changes; specifically,

$$\mathrm{PSL_z = \max_{k \neq 0} |ACF_z(k)|},$$

where

$$\mathrm{ACF_z(k) = \sum_{i=1}^{N-k} z_i z_{i+k}^c}.$$

for $k = 1, \ldots, N-1$ and

$$\mathrm{ACF_z(-k) = ACF_z^c(k)}$$

for $k = 1-N, \ldots, -1$. Unimodular codes z with $\mathrm{PSL_z} = 1$ are called generalized Barker sequences (as opposed to the term Barker code used in the binary case).

The operations that preserve PSL generalize to the following set of four (see [5]):

1. Reversal: $g_1(z) = \overline{z}$

2. Complex Conjugation: $g_2(z) = z^c$

3. Multiplication by a unit-magnitude complex number: $g_3(z) = \mu_1 z$, where $\mu_1 = 1$

4. Progressive multiplication by a unit-magnitude complex number μ_2:

$$g_4(z) = zA_N$$

where

$$A = \begin{pmatrix} \mu_2 & 0 & 0 & \dots & 0 \\ 0 & (\mu_2)^2 & 0 & \dots & 0 \\ 0 & 0 & (\mu_2)^3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & (\mu_2)^N \end{pmatrix}.$$

Without loss of generality, we can assume that $\mu_1, \mu_2 \in \{1, i, -1, -i\}$ where $i = \sqrt{-1}$.

In the quad-phase case, for a given m and code length N, the four PSL-preserving operations listed above generate a group of order 64. It is not commutative. As for the binary ± 1 case, the space of quad-phase codes can be partitioned into equivalence classes of maximum size 64. For simplicity (and without loss of generality), set $\mu_1 = \mu_2 = i$. Two N-length quad-phase codes z_1 and z_2 are equivalent if z_2 can be produced from z_1 using some combination of the four operations listed above.

For this quad-phase framework, consider the following challenge carried over from the binary case.

Challenge (Research): For the case of $m = 4$, investigate whether a method exists to generate single representatives of each equivalence class, one equivalence class at a time, for any length N. Alternatively, establish that no such method exists. Along the way, look for ways to ex;loit symmetries to reduce the number of codes that need to be checked. Whatever determination you can achieve, try and extend your results to other values of $m > 2$.

To be clear, the challenge is to find the answer where none is apparently known. It may be surprising, but we are already on the line between questions with answers and one that do not. Be brave and venture forth. We are here to respond to questions, but realize that we may not have the answers ready. The answers will be yours to enjoy and to publish.

9 References

[1] Levanon, N. and Mozeson, E., *Radar Signals*, Wiley, NY, 2004.

[2] Richards, M., *Fundamentals of Radar Signal Processing*, second edition, McGraw-Hill, NY, 2014.

[3] Skolnik, M., *Radar Handbook*, third edition, McGraw-Hill, NY, 2008.

[4] Richards, M. and Scheer, J., *Principles of Modern Radar – Basic Principles*, SciTech, Raleigh NC, 2010.

[5] Golomb, S. and Win, M., "Recent results on polyphase sequences," *IEEE Transactions on Information Theory*, vol. 44, Mar. 1999, pages 817-824.

[6] Turyn, R. and J. Storer, J., "On binary sequences." *Proceedings of the American Mathematical Society*, vol. 12, No. 3 (Jun 1961), pages 394-399.

[7] Hungerford, T.M., *Algebra*, Springer-Verlag, NY, 1974.

[8] Farris, F., *Creating Symmetry*, Princeton University Press, Princeton, NJ, 2015.