MATLAB Costas Arrays Toolbox

User's Manual

Functions to Generate and Manipulate Costas Arrays

Ken Taylor, Konstantinos Drakakis and Scott Rickard

July 9, 2009

sfi science foundation ireland fondúireacht eolaíochta éireann

UCD DUBLIN

SSP Group UCD

1 Introduction

This manual gives information about the functions contained within the Costas toolbox. These functions can be used to generate Costas arrays, manipulate them in various ways, and test the properties which they possess. Costas arrays are permutation matrices representing frequency hop waveforms. The constraints placed on the arrays yield ideal auto-ambiguity properties, which gives rise to applications in radar and sonar. Each section of the manual is dedicated to a different type of function, with the majority of functions falling under the headings of generation, manipulation operations, and array classification.

2 Generation Functions

The following functions all use algorithms which guarantee the production of Costas arrays for certain constraints. They can be used to obtain Costas arrays of sizes relative to primes and powers of prime numbers. **golomb2.m**
Description: Generates G_2 Golomb Costas array sequences from a prime p, or power of prime q.
Usage: golomb2(q), golomb2(q, a), golomb2(q, a, b)
This function generates Golomb Costas arrays of size p - *2*. The user is required to enter a prime number p or power of a prime q to indicate the size of array required. If a prime number is entered then the power of the prime is an optional argument for inclusion, denoted by n, along with the primitive elements used for generation, a and b. Multiple primitive elements can be

entered for a, then the function will output all arrays which are generated using them. To generate one specific array, a and b can both be given a single value.

Golomb arrays are constructed by placing a dot in position (i, j) if and only if $a^i + b^j = 1$, taken modulo p. Each pair of primitive elements generate a different Costas array of size p - *2*.

golomb3.m

Description: Generates G_3 Golomb Costas array sequences from a prime p, or power of prime q.

Usage: golomb3(q), golomb3(q, a), golomb3(q, a, b)

This function generates Golomb Costas arrays of size p - *3*. The user is required to enter a prime number p or power of a prime q to indicate the size of array required. If a prime number is entered then the power of the prime is an optional argument for inclusion, denoted by n, along with the primitive elements used for generation, a and b. Multiple primitive elements can be entered for a, then the function will output all arrays which are generated using them. To generate one specific array, a and b can both be given a single value. If the G_2 arrays of the required size are already known then they can be passed straight into the function to obtain the G_3 arrays.

G_3 Golomb Costas arrays are generated by taking G_2 Golomb arrays of the relevant size and removing the corners dot at the upper left hand corner, which exist when $a + b = 1$, taken modulo p.

golomb4.m

Description: Generates G_4 Golomb Costas array sequences from a prime p, or power of prime q.

Usage: golomb4(q), golomb4(q, a), golomb4(q, a, b)

This function generates Golomb Costas arrays of size p - *4*. The user is required to enter a prime number p or power of a prime q to indicate the size of array required, however this construction only works for powers of two. If a prime number is entered then the power of the prime, denoted by n, is passed to the function as the second argument. The primitive elements used for generation, a and b, are optional arguments for inclusion. Multiple primitive elements can be entered for a, then the function will output all arrays which are generated using them. To generate one specific array, a and b can both be given a single value. If the G_2 arrays of the required size are already known then they can be passed straight into the function to obtain the G_4 arrays.

G_4 Golomb Costas arrays are generated by taking G_2 Golomb arrays of size $2^n - 2$, where n is an integer greater than one, and removing the dots at (1,1) and (2,2), which exist when $a + b = 1$, taken modulo p.

golomb4b.m

Description: Generates G_4^* Golomb Costas array sequences from a prime p, or power of prime q.

Usage: golomb4b(q), golomb4b(q, a), golomb4b(q, a, b)

This function generates Golomb Costas arrays of size p - *4*. The user is required to enter a prime number p or power of a prime q to indicate the size

of array required. If a prime number is entered then the power of the prime is an optional argument for inclusion, denoted by n, along with the primitive elements used for generation, a and b. Multiple primitive elements can be entered for a, then the function will output all arrays which are generated using them. To generate one specific array, a and b can both be given a single value. If the G_2 arrays of the required size are already known then they can be passed straight into the function to obtain the G_4^* arrays.

G_4^* Golomb Costas arrays are generated by taking G_2 Golomb arrays of the relevant size and removing the dots from (1,1) and (2,q-2), which exist when $a + b = 1$, and $a^2 + b^{-1} = 1$, taken modulo p.

golomb5b.m

Description: Generates G_5^* Golomb Costas array sequences from a prime p, or power of prime q.

Usage: golomb5b(q), golomb5b(q, a), golomb5b(q, a, b)

This function generates Golomb Costas arrays of size p - 5. The user is required to enter a prime number p or power of a prime q to indicate the size of array required. If a prime number is entered then the power of the prime is an optional argument for inclusion, denoted by n, along with the primitive elements used for generation, a and b. Multiple primitive elements can be entered for a, then the function will output all arrays which are generated using them. To generate one specific array, a and b can both be given a single value. If the G_2 arrays of the required size are already known then they can be passed straight into the function to obtain the G_5^* arrays.

G_5^* Golomb Costas arrays are generated by taking G_2 Golomb arrays of

the relevant size and removing the dots from (1,1), (2,q-2) and (q-2,2), which exist when $a + b = 1$, and $a^2 + b^{-1} = 1$, taken modulo p. This is the same as taking G_4^* Golomb arrays and removing the dot at the bottom left hand corner.

lempel2.m

Description: Generates L_2 Golomb Costas array sequences from a prime p, or power of prime q.

Usage: lempel2(q), lempel2(q, a)

This function generates L_2 Lempel arrays of size p - 2. The user is required to enter a prime number p or power of a prime q to indicate the size of array required. If a prime number is entered then the power of the prime is an optional argument for inclusion, denoted by n, along with the primitive elements used for generation, a. Multiple primitive elements can be entered for a, then the function will output all arrays which are generated.

L_2 Lempel arrays are G_2 Golomb arrays with b equal to a. Thus arrays are constructed by placing a dot in position (i, j) if and only if $a^i + a^j = 1$, taken modulo p. Each primitive element generates a different Costas array of size p - 2.

lempel3.m

Description: Generates L_3 Golomb Costas array sequences from a prime p, or power of prime q.

Usage: lempel3(q), lempel3(q, a)

This function generates Lempel Costas arrays of size p - 3. The user is

required to enter a prime number p to indicate the size of array required. The power of the prime is an optional argument for inclusion, denoted by n, which must take the value *1* for this construction. The primitive element used for generation, a is also an optional argument for inclusion. Multiple primitive elements can be entered for a, then the function will output all arrays which are generated using them.

L_3 Lempel Costas arrays are generated by taking L_2 Lempel arrays of the relevant size and removing the corners dot at the upper left hand corner, which exist when *2* is primitive in *GF(p)*.

primelem.m

Description: Generates the primitive elements for an input prime p, or power of prime q.

Usage: primelem(p), primelem(q)

Each of the Costas array generation functions uses the *primelem* function as a starting point to construct arrays. This function calculates primitive elements for prime numbers (denoted by p) used for Welch array generation, and for powers of prime numbers (denoted by q) used for Golomb array generation.

For prime numbers, the function tests all integers up to p - *1* by raising them to successive powers, taken modulo p. Should they take the value one before completing a full cycle then the starting number is not primitive is excluded. When testing is completed the function outputs a vector of all numbers which are primitive. The case for prime powers requires use of a built in MATLAB function to obtain irreducible polynomials. Use of

Euler's totient function provides the number of primitive elements, which are expressed as polynomials with reducing powers of x. Raising the polynomials to successive powers is done using convolution, and the modulo operation is performed using deconvolution, which expresses the polynomial in required terms. When testing is completed the function outputs rows of primitive element polynomials. These can then be used in the various construction functions detailed below.

taylor4.m

Description: Generates T_4 arrays, Taylor variant to the Lempel construction, from a prime p, or power of prime q.

Usage: taylor4(q), taylor4(q, a)

This function generates the Taylor variant to Lempel Costas arrays of size p - 4. The user is required to enter a prime power, or a prime as the first argument and prime as the second argument. They may also enter a primitive element if a specific one is required.

When $a^2 + a^1 = 1$ then there will be dots located at (1,2) and (2,1), which can simultaneously be removed from L_2 Lempel arrays. These dots are removed using the *remrc* function. The L_2 arrays are generated, then each is tested to see if they contain dots at (1,2) and (2,1). If so then the dots are removed and the resulting array is added to the list of output arrays.

welch1exp.m

Description: Generates W_1^{exp} Welch Costas array sequences from a prime *p*.

Usage: welch1exp(p), welch1exp(p, a), welch1exp(p, a, c,)

This function generates exponential Welch Costas arrays of size *p - 1*. The user is required to enter a prime number *p* to indicate the size of array required. The primitive elements used to generate arrays are an optional argument for inclusion, denoted by *a*. If none are entered, or *a* is given the value one, then the function uses all possible primitive elements by default. Since all circular shifts of exponential Welch Costas arrays also possess the Costas property, the user can also include a circular shift of generated arrays, denoted by *c*. Each shift moves the first element of the array to the end.

W_1^{exp} Welch Costas arrays are generated by taking a primitive element, and raising it to successive powers, taken modulo *p*. Each primitive element generates a different Costas array of size *p - 1*.

welch1.m

Description: Generates all W_1 Welch Costas arrays from a prime *p*.

Usage: welch1(p), welch1(p, a), welch1(p, a, c,)

This function generates all W1 Welch Costas array sequences (exponential and logarithmic) of size *p - 1* and includes all possible circular shifts in the output (unless *c* is specified). The user is required to enter a prime number *p* to indicate the size of array required. The primitive elements used to generate arrays are an optional argument for inclusion, denoted by *a*. If none are entered, or *a* is given the value one, then the function uses all possible primitive elements by default.

welch2exp.m

Description: Generates W_2^{exp} Welch Costas array sequences from a prime p.

Usage: welch2exp(p), welch2exp(p, a)

This function generates exponential Welch Costas arrays of size p - *2*. The user is required to enter a prime number p to indicate the size of array required. The primitive elements used to generate an array are an optional argument for inclusion, denoted by a. If none are entered, or a is given the value one, then the function uses all possible primitive elements by default.

W_2^{exp} Welch Costas arrays are generated by taking W1 Welch arrays of the relevant size and removing the corner dot at the upper left hand corner.

welch3exp.m

Description: Generates W_3^{exp} Welch Costas array sequences from a prime p.

Usage: welch3exp(p), welch3exp(p, a)

This function generates Welch Costas arrays of size p - *3*. The user is required to enter a prime number p to indicate the size of array required. The primitive element used to generate an array is an optional argument for inclusion, to follow the format of other Welch array generation functions, though for this construction it must have value *2*.

W_3^{exp} Welch Costas arrays are generated by taking W_2^{exp} Welch arrays of the relevant size and removing the corner dot at the upper left hand corner, which is possible when the primitive element used to generate the array has value *2*.

3 Emergent Functions

The following functions manipulate Costas arrays in an attempt to find other Costas arrays. This can be done in a variety of methods based on adding and removing dots to create new arrays.

rickard.m

Description: Generates Rickard arrays of size n from a stub x.

Usage: rickard(x, n)

The Rickard expansion method makes use of the periodicity properties of Welch and Golomb Costas arrays to obtain new Costas arrays. The technique used here involves vertically stacking two copies of an array of size x, with a blank row placed between the two arrays. All possible (x+n)-by-(x+n) windows are then tested by adding dots in each possible way, with the resulting Costas arrays being returned.

This function is most commonly used to obtain Costas arrays using a single or double dot extension to existing Costas arrays. The function can be used to extend arrays by more dots, but this will greatly increase the time required for the function to run as there will be more possible arrays to test. The arrays in the input set do not have to be Costas arrays, and blank columns of arrays in the input set can be represented by a 'NaN'.

drt.m

Description: Generates DRT arrays of size n from a stub x.

Usage: drt(x, n)

The DRT (Drakakis, Rickard, Taylor) expansion method is a two dimensional extension of the Rickard expansion which can be used to obtain new Costas arrays. The technique used here involves starting with four copies of an array of size x placed in a square, with a blank cross between the arrays. All possible (x+n)-by-(x+n) windows are then tested by adding dots in each possible way, with the resulting Costas arrays being returned.

This function is most commonly used to obtain Costas arrays using a single dot extension to existing Costas arrays. The function can be used to extend arrays by more dots, but this will greatly increase the time required for the function to run as there will be more possible arrays to test. The arrays in the input set do not have to be Costas arrays, and blank columns of arrays in the input set can be represented by a 'NaN'.

spinadd.m

Description: Generates Beard arrays of size *n + 1* by adding a corner dot.

Usage: spinadd(A)

This Beard expansion method searches for Costas arrays by trying to add a corner dot to to all circular shifts of all the arrays in the dihedral symmetry class of an input set of arrays.

This function starts by generating all flips and rotations of the arrays in the input set. All circular shifts of the resulting arrays are then found. A corner dot is added to all these arrays, and the Costas arrays from the resulting set are taken as output.

spinadd2.m

Description: Generates Beard arrays of size *n + 2* by adding two corner dots.

Usage: spinadd2(A)

This Beard expansion method searches for Costas arrays by trying to add two corner dots at opposing corners to to all circular shifts of all the arrays in the dihedral symmetry class of an input set of arrays.

This function starts by generating all flips and rotations of the arrays in the input set. All circular shifts of the resulting arrays are then found. Two corner dots are added to all these arrays at opposing corners, and the Costas arrays from the resulting set are taken as output.

spindrop.m

Description: Generates Beard arrays of size *n - 1* by removing a corner dot.

Usage: spindrop(A)

This Beard expansion method searches for Costas arrays by circular shifting arrays until they contain a corner dot, then removing the dot and testing to see if the result is a Costas array.

This function starts by generating all flips and rotations of the arrays in the input set. All circular shifts of the resulting arrays are then found. All arrays containing a corner dot have the dot removed and the result is tested to see if it is a Costas array.

spindrop2.m

Description: Generates Beard arrays of size *n - 2* by removing two corner dots.

Usage: spindrop2(A)

This Beard expansion method searches for Costas arrays by circular shifting arrays until they contain a corner dot, then if there is also a dot in the opposing corner both dots are removed and the result is tested to see if it is a Costas array.

This function starts by generating all flips and rotations of the arrays in the input set. All circular shifts of the resulting arrays are then found. Any arrays which contain corner dots in opposing corners have the dots removed and the result is tested to see if it is a Costas array.

taylor0.m

Description: Generates T_0 arrays, Taylor variant to the Golomb construction, from a prime p, or power of prime q.

Usage: taylor0(q), taylor0(q, a), taylor0(q, a, b)

This function generates the Taylor variant to Golomb Costas arrays of size p. The user is required to enter a prime number p or power of a prime q to indicate the size of array required. If a prime number is entered then the power of the prime is an optional argument for inclusion, denoted by n, along with the primitive elements used for generation, a and b. Multiple primitive elements can be entered for a, then the function will output all arrays which are generated using them.

When the G_2 Golomb arrays have been constructed using the parameters passed to the function, it searches for Costas arrays by adding dots to the upper left and lower right corner, and also adding dots to the upper right and lower left corner. Each resulting array is tested to see if it is a Costas array, and if so it is added to the list of output arrays.

taylor1.m

Description: Generates T_1 arrays, Taylor variant to the Golomb construction, from a prime p, or power of prime q.

Usage: taylor1(q), taylor1(q, a), taylor1(q, a, b)

This function generates the Taylor variant to Golomb Costas arrays of size p - *1*. The user is required to enter a prime number p or power of a prime q to indicate the size of array required. If a prime number is entered then the power of the prime is an optional argument for inclusion, denoted by n, along

with the primitive elements used for generation, a and b. Multiple primitive elements can be entered for a, then the function will output all arrays which are generated using them.

When the G_2 Golomb arrays have been constructed using the parameters passed to the function, it searches for Costas arrays by adding dots at each of the corners. Each resulting array is tested to see if it is a Costas array, and if so it is added to the list of output arrays.

welch0.m

Description: Generates W_0 Welch Costas array sequences from a prime p.

Usage: welch0(p), welch0(p, a), welch0(p, a, c)

This function generates Welch Costas arrays of size p. The user is required to enter a prime number p to indicate the size of array required. The primitive elements used for generation, denoted by a, are an optional argument for inclusion, along with any circular shifting if necessary.

W_0 Welch Costas arrays are generated by taking W_1 Welch arrays of the relevant size, and then trying to add a dot at each corner using the *caddul, caddur, caddll* & *caddlr* functions. Each new array is tested to see if it satisfies the conditions for Costas, and if so is added to the list of output arrays.

4 Operation Functions

addrc.m

Description: A function to add blank rows and columns to an array or multiple arrays in permutation form.

Usage: addrc(r, c, A)

This function can be used to add a row, column, or both to one or more arrays in permutation form. When calling the function, r represents the row to be added, c represents the column, and A is the array or arrays to be expanded. Setting $r = 0$ means only adding a column and vice-versa. Blank columns are represented by a 'NaN'.

adddot.m

Description: A function to add a dot to an array or multiple arrays in permutation form.

Usage: adddot(r, c, A)

This function can be used to fill in gaps in an array or multiple arrays which are currently being described as *NaN*. The first and second arguments passed to the function give the row and column location of the dot which is to be added, the third argument is the array or arrays which are to be filled. The function only adds dots to arrays which have a *NaN* in the specified location, all other arrays are ignored, except those which already have the desired value there.

circshifts.m

Description: Generates all unique circular shifts of input arrays.

Usage: circshifts(A)

This function can be used to obtain all the circular shifts of an array or multiple arrays. This is useful for the case of Welch arrays, where the periodicity property means that all circular shifts are also Costas arrays.

cadd.m

Description: A set of functions to add a corner dot to an array or multiple arrays in permutation form.

Usage: cadd('ul', A), cadd('ur', A), cadd('ll', A), cadd('lr', A), cadd('all', A)

This function can be used to augment an array or multiple arrays by adding a dot at one of the corners. The first argument passed to the function represents the corner where the dot is to be added, with 'ul' for the upper left corner, 'll' for the lower left corner, 'lr' for the lower right corner, and 'ur' for the upper right corner. The second argument passed to the function, A, should contain the array or arrays which are to be augmented. Passing 'all' as the first argument created four new arrays for each array in A, one with a dot in each corner. The function outputs all arrays, regardless of whether or not the result is a Costas permutation.

crem.m

Description: A set of functions to remove a corner dot from an array or multiple arrays in permutation form.

Usage: crem('ul', A), crem('ur', A), crem('ll', A), crem('lr', A)

This function can be used to decrease the size of an array or multiple arrays by removing a dot at one of the corners. The first argument passed to the function represents the corner where the dot is to be removed, with 'ul' for the upper left corner, 'll' for the lower left corner, 'lr' for the lower right corner, and 'ur' for the upper right corner. The second argument passed to the function, A, should contain the array or arrays which are to be reduced. The function only outputs arrays which originally had a dot at the specified location, all other arrays are ignored.

dsym.m

Description: A function to flip an array or multiple arrays in permutation form about an axis, or rotate by multiples of 90°.

Usage: dsym('hor', A), dsym('ver', A), dsym('mdiag', A), dsym('adiag', A), dsym(90, A), dsym(180, A), dsym(270, A)

This set of functions can be used to alter an array or multiple arrays by flipping about an axis. The first argument passed to the function is the axis about which the flip is performed, with 'hor' for horizontal, 'ver' for vertical, 'mdiag' for the main diagonal, and 'adiag' for the anti-diagonal. Alternatively, the number passed to the function refers to the amount of clockwise rotation. The second argument passed to the function, A, should contain the array or arrays which the operation should be performed on.

dsyms.m

Description: Generates the family of arrays in the dihedral symmetry class for the sequences in the input matrix (all rotations and flips).

Usage: dsyms(c)

This function can be used to obtain all rotations and flips of an array or multiple arrays in permutation form. The arrays are ordered lexicographically and repeat arrays are removed.

minimal.m

Description: Replaces input arrays with their flip or rotation which is minimal, then returns sorted unique arrays.

Usage: minimal(A)

This function can be used to replace an array or multiple arrays with their flip or rotation which is minimal. For each input array, all rotations and flips are generated, and the one which comes first lexicographically replaces the original array in the output.

remrc.m

Description: A function to remove rows and columns from an array or multiple arrays in permutation form.

Usage: remrc(r, c, A)

This function can be used to remove a row, column, or both from one or more arrays in permutation form. When calling the function, r represents the row to be removed, c represents the column, and A is the array or arrays to be manipulated. Setting $r = 0$ means only removing a column and vice-versa.

remdot.m

Description: A function to remove a dot to an array or multiple arrays in permutation form.

Usage: remdot(r, c, A)

This function can be used to remove dots from an array or multiple arrays. The size of the array or arrays is not altered, with removed dots being replaced by a *NaN*. The first and second arguments passed to the function give the row and column location of the dot which is to be removed, the third argument is the array or arrays which are to be edited. The function only removes dots from arrays which are not empty at the specified location, all other arrays are ignored, except those which already have a *NaN* there.

uniquearrays.m

Description: Removes rotations and flips from a set of Costas arrays.

Usage: uniquearrays(A)

This function can be used to remove rotations and flips from a set of input arrays. First the input arrays are sorted, then all rotations of the first row are removed. Next all rotations and flips of the second row are removed, and so on until all the remaining arrays are unique. The function does not output minimal arrays, only unique arrays from the input set.

5 Classification Functions

iscircshift.m

Description: Determines whether or not an array *m1* is a circular shift of array *m2*.

Usage: iscircshift(m1, m2)

This function can be used to determine if an array is a circular shifted version of another array. The function outputs '1' if the first array is a circular shift of the second, and '0' if not. It can also be used for multiple arrays, returning a column of ones and zeros.

iscostas.m

Description: Tests a sequence or multiple rows of sequences to see if Costas condition is satisfied.

Usage: iscostas(x)

This function can be used to check if a sequence satisfies the Costas condition, that all vectors connecting pairs of dots are unique. The function outputs '1' if the condition is satisfied, and '0' otherwise. It does not check if the sequence is a permutation. 'A NaN' can be used to represent a blank column for testing incomplete sequences. It can also be used to test multiple sequences, returning a column of ones and zeros.

iscostasperm.m

Description: Tests a sequence or multiple rows of sequences to see if both Costas conditions are satisfied.

Usage: iscostasperm(x)

This function checks both Costas conditions for a sequence or multiple rows of sequences. For a single sequence the function outputs '1' if both conditions are satisfied, and '0' otherwise. For multiple rows of sequences the function outputs a column of ones and zeros.

The first condition for Costas is that every row and column of the grid must contain exactly one dot. This is satisfied if the sequence is a permutation. The second condition is that all vectors connecting pairs of dots are unique. This can be tested by constructing a difference triangle for the sequence. If all rows of the difference triangle contain unique values then the condition is satisfied.

isdiagonal.m

Description: Determines whether or not an array is symmetric about the main diagonal.

Usage: isdiagonal(A)

This function can be used to determine whether or not an array is symmetric about the main diagonal. The function outputs '1' if the array is symmetric about the main diagonal, and '0' if not. It does this by checking if the array changes when flipped about the main diagonal. It can also be used for multiple arrays, returning a column of ones and zeros. Arrays which are not symmetric about the main diagonal will generate eight unique rotations and

flips, where as arrays which are symmetric about the main diagonal will only generate four unique rotations and flips.

isrotflip.m

Description: Determines whether an array *m1* is a flip or rotation of array *m2*.

Usage: isrotflip(m1, m2)

This function can be used to determine if an array is a member of the family of another array, that is whether or not an array *m1* is a member of the family of arrays which can be generated for an array *m2*. The function outputs '1' if the first array is a flip or rotation of the second, and '0' if not. It can also be used for multiple arrays, returning a column of ones and zeros.

6 Testing Functions

This section of the toolbox was added in version 1.03, and allows the user to verify that functions are working correctly. This is useful as changes can be made to functions to try and improve performance. Following such changes the user can run the tests to verify that the function still performs as expected. The testing functions work by generating a set of results and comparing them to data stored in files which we believe to be correct. Data in the testing files was obtained by other methods than direct use of the relevant function. Currently the testing only covers the generation functions of the toolbox, but this will later be extended to cover more functions.

A separate testing function exists for each generation function. Each

function outputs ‘1’ if the function has passed the test and ‘0’if it fails. In the event of a failure then the function will the display the input for which the correct output was not generated.

testall.m

Description: Runs all the test functions and checks for errors.

Usage: testall

This function runs all of the testing functions in series and displays the results of each test to the screen. It will output ‘1’ if all functions are working as expected and ‘0’ otherwise.

7 Miscellaneous Functions

Contents.m

Description: Contains a description of all the functions in the toolbox.

correlationsurface.m

Description: Generates the correlation surface for two input arrays.

Usage: correlationsurface(x,y)

This function can be used to generate the correlation surface for two input arrays x and y. Use x = y for an auto-correlation surface. After generating the correlation surface it can be viewed using the mesh function. This function can be used to view the ideal thumbtack auto-ambiguity of Costas arrays. Compare this plot with a plot of a staircase waveform to see the advantage of Costas arrays.

costas.m

Description: Outputs only the Costas arrays contained in the input set of arrays.

Usage: costas(A)

This function can be used to obtain the Costas arrays from a set of arrays in permutation form. Any arrays in the input set which are not Costas arrays are removed from the output set.

findcostas.m

Description: Generates all Costas arrays for a given input size by exhaustively searching through permutations.

Usage: findcostas(n)

This function can be used to find all Costas arrays of a specific size. The user is required to enter the size of arrays which they require, then the function tests all permutations of that size to see if they are Costas arrays. This function is only suitable for small sizes as it is a very inefficient way of searching for Costas arrays, but is a good starting point to work from.

plotcostas.m

Description: Generates a plot of the Costas array which is entered.

Usage: plotcostas(A)

This function can be used to look at the plot of a Costas array. Matrix convention is used to describe the index of array elements, so *(1,1)* is defined to be at the top left hand corner of the grid.

stubtocostas.m

Description: Exhaustively searches for Costas arrays of a specified length which begin with a certain stub.

Usage: stubtocostas(v, n)

This function can be used to try and find Costas arrays with a specified beginning. The first argument passed to the function should be the stub, and the second argument passed to the function should be the length of Costas array which it is desired to find. The function combines all permutations of the missing numbers with the original stub and tests the result to see if it is a Costas array. All Costas arrays which are found are added to the output. The function should only be used to expand stubs by a small amount.

zeropad.m

Description: Pads the border of a hit array (ha) with zeros.

Usage: zeropad(ha)

This function can be used to add a border of zeros to a hit array which has been generated using the correlationsurface function. This improves the look of a correlation surface when it is plotted using the mesh function.

8 Acknowledgements

We would like to thank Steve Eddins from MATLAB for his extensive guidance on coding style. His detailed comments were extremely useful and much appreciated.