

bent-functions

last edited on February 12, 2012 09:37 AM by admin

 Typeset
 [Worksheet](#) [Edit](#) [Text](#) [Undo](#) [Share](#) [Publish](#)

%hide

The Walsh-Hadamard transform

%hide

The *Walsh transform* of a Boolean function

$$f : GF(2)^n \rightarrow GF(2)$$

is an integer-valued function over $GF(2)^n$ that can be defined as

$$W_f(u) = \sum_{x \in GF(2)^n} (-1)^{f(x) + \langle u, x \rangle}.$$

If we order the elements of $GF(2)^n$ in some fixed way, then this expression for the Walsh transform can be rewritten as the multiplication of the column vector

$$f^* = ((-1)^{f(x)} \mid x \in GF(2)^n),$$

by the $2^n \times 2^n$ Hadamard matrix

$$H = ((-1)^{x \cdot y} \mid x, y \in GF(2)^n).$$

```
V = GF(2)^3
X = V.list()
X

[(0, 0, 0), (1, 0, 0), (0, 1, 0), (1, 1, 0), (0, 0, 1), (1, 0, 1), (0, 1, 1), (1, 1, 1)]
```

```
V = GF(2)^4
X = V.list()
X

[(0, 0, 0, 0), (1, 0, 0, 0), (0, 1, 0, 0), (1, 1, 0, 0), (0, 0, 1, 0),
 (1, 0, 1, 0), (0, 1, 1, 0), (1, 1, 1, 0), (0, 0, 0, 1), (1, 0, 0, 1),
 (0, 1, 0, 1), (1, 1, 0, 1), (0, 0, 1, 1), (1, 0, 1, 1), (0, 1, 1, 1),
 (1, 1, 1, 1)]
```

%hide

We shall use this "lexicographical" ordering. Here is the corresponding Hadamard matrix:

```
H = matrix(QQ, [[(-1)^(x.dot_product(y)) for x in X] for y in X])
```

```
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
[ 1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1  1 -1]
[ 1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1]
[ 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1]
[ 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1]
[ 1 -1  1 -1 -1  1 -1  1  1 -1 -1 -1  1 -1  1 -1]
[ 1  1 -1  1 -1 -1  1  1  1  1 -1 -1 -1 -1  1  1]
[ 1 -1  1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1 -1  1]
[ 1  1 -1 -1  1 -1 -1  1  1  1 -1 -1 -1 -1 -1  1]
[ 1 -1  1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1]
[ 1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1]
[ 1 -1  1 -1  1 -1  1 -1  1  1 -1 -1 -1 -1 -1  1]
[ 1  1 -1 -1  1 -1  1 -1 -1  1  1 -1 -1 -1 -1  1]
[ 1 -1  1 -1 -1  1 -1 -1  1  1 -1 -1 -1 -1 -1  1]
[ 1  1  1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1]
[ 1 -1  1 -1 -1 -1 -1  1  1  1  1 -1 -1 -1 -1 -1]
[ 1  1 -1 -1 -1 -1 -1 -1  1  1  1  1  1  1 -1 -1]
[ 1 -1  1 -1 -1 -1 -1 -1 -1  1  1  1  1  1  1  1]
```

```
from sage.crypto.boolean_function import *
P.<x0,x1,x2,x3> = BooleanPolynomialRing()
b = x0*x1 + x2*x3
f = BooleanFunction(b)
fstar = vector([(-1)^(int(b(x[0],x[1],x[2],x[3]))) for x in V])
print [b(x[0],x[1],x[2],x[3]) for x in V], "\n", fstar
```

```
[0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0]
(1, 1, 1, -1, 1, 1, 1, -1, 1, 1, 1, -1, -1, -1, -1, 1)
```

```
f.walsh_hadamard_transform()
```

```
(-4, -4, -4, 4, -4, -4, 4, -4, -4, -4, 4, 4, 4, 4, -4)
```

```
def walsh_transform(f, a):
    """
    This computes the Walsh(-Fourier) transform of the
    Boolean function f on $GF(2)^n$ at $a \in GF(2)^n$.

    INPUT:
        s - a sequence of n elements in GF(2), represented as a list
        k - an integer in the interval [0,n-1].
    WARNING: This seems to differ by a sign from the method
    walsh_hadamard_transform in sage.crypto.boolean_function.

    """
    F = a[0].parent()
    n = len(a)
    terms = [(-1)^(f(x)+x.dot_product(a)) for x in F**n]
    return sum(terms)
```

[evaluate](#)

```
%hide
```

Now we check if this function f is bent by computing its Walsh transform and seeing if we get $+2^{(4/2)}=+4$ or $-2^{(4/2)}=-4$.

```
f0 = lambda x: x[0]*x[1] + x[2]*x[3] # same f as above
WT = [walsh_transform(f0, a) for a in V]; WT
```

```
[4, 4, 4, -4, 4, 4, 4, -4, 4, 4, 4, -4, -4, -4, -4, 4]
```

```
H*fstar
```

```
(4, 4, 4, -4, 4, 4, -4, 4, 4, 4, -4, -4, -4, -4, 4)
```

```
from sage.crypto.boolean_function import *
P.<x0,x1,x2,x3> = BooleanPolynomialRing()
b = x0*x2 + x1*x3
f = BooleanFunction(b)
fstar = vector([(-1)^(int(b(x[0],x[1],x[2],x[3]))) for x in V])
print [b(x[0],x[1],x[2],x[3]) for x in V], "\n", fstar
```

```
[0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0]
(1, 1, 1, 1, 1, -1, 1, -1, 1, 1, -1, -1, 1, -1, -1, 1)
```

```
%hide
```

Now we check if this function f is bent by computing its Walsh transform and seeing if we get $+2^{(4/2)}=+4$ or $-2^{(4/2)}=-4$.

```
f0 = lambda x: x[0]*x[2] + x[1]*x[3] # same f as above
WT = [walsh_transform(f0, a) for a in V]; WT
```

```
[4, 4, 4, 4, -4, 4, -4, 4, 4, -4, -4, 4, -4, -4, 4]
```

```
H*fstar
```

```
(4, 4, 4, 4, -4, 4, -4, 4, 4, -4, -4, 4, -4, -4, 4)
```

```
f.walsh_hadamard_transform()
```

```
(-4, -4, -4, -4, -4, 4, -4, 4, -4, -4, 4, 4, -4, 4, 4, -4)
```