

## Order and Metric Compatible Symbolic Sequence Processing

A traditional random variable  $X$  is a function that maps from a stochastic process to the real line  $(X, \leq, d, +, .)$ , where  $R$  is the set of real numbers,  $\leq$  is the standard linear order relation on  $R$ ,  $d(x,y)=|x-y|$  is the usual metric on  $R$ , and  $(R, +, .)$  is the standard field on  $R$ . Greenhoe(2015b) has demonstrated that this definition of random variable is often a poor choice for computing statistics when the stochastic process that  $X$  maps from has structure that is dissimilar to that of the real line. Greenhoe(2015b) has further proposed an alternative statistical system, that rather than mapping a stochastic process to the real line, instead maps to a weighted graph that has order and metric geometry structures similar to that of the underlying stochastic process. In particular, ideally the structure  $X$  maps from and the structure  $X$  maps to are, with respect to each other, both isomorphic and isometric. Mapping to a weighted graph is useful for analysis of a single random variable. for example the expectation  $EX$  of  $X$  can be defined simply as the center of its weighted graph. However, the mapping has limitations with regards to a sequence of random variables in performing sequence analysis (using for example Fourier analysis or wavelet analysis), in performing sequence processing (using for example FIR filtering or IIR filtering), in making diagnostic measurements (using a post-transform metric space), or in making optimal decisions (based on distance measurements in a metric space or more generally a distance space). Rather than mapping to a weighted graph, this paper proposes instead mapping to an ordered distance linear space

$Y=(R^n, \leq, d, +, ., R, +, x)$ , where  $(R, +, x)$  is a field,  $+$  is the vector addition operator on  $R^n$   $\times R^n$ , and  $.$  is the scalar-vector multiplication operator on  $R \times R^n$ . The linear space component of  $Y$  provides a much more convenient (as compared to the weighted graph) framework for sequence analysis and processing. The ordered set and distance space components of  $Y$  allow one to preserve the order structure and distance geometry inherent in the underlying stochastic process, which in turn likely provides a less distorted (as

compared to the real line) framework for analysis, diagnostics, and optimal decision making.

# Order and Metric Compatible Symbolic Sequence Processing

DANIEL J. GREENHOE

*Abstract:* A traditional random variable  $X$  is a function that maps from a stochastic process to the “real line”  $(\mathbb{R}, \leq, d, +, \cdot)$ , where  $\mathbb{R}$  is the set of real numbers,  $\leq$  is the standard linear order relation on  $\mathbb{R}$ ,  $d(x, y) \triangleq |x - y|$  is the usual metric on  $\mathbb{R}$ , and  $(\mathbb{R}, +, \cdot)$  is the standard field on  $\mathbb{R}$ . [Greenhoe \(2015b\)](#) has demonstrated that this definition of random variable is often a poor choice for computing statistics when the stochastic process that  $X$  maps from has structure that is dissimilar to that of the real line. [Greenhoe \(2015b\)](#) has further proposed an alternative statistical system, that rather than mapping a stochastic process to the real line, instead maps to a *weighted graph* that has order and metric geometry structures similar to that of the underlying stochastic process. In particular, ideally the structure  $X$  maps from and the structure  $X$  maps to are, with respect to each other, both *isomorphic* and *isometric*.

Mapping to a weighted graph is useful for analysis of a single random variable—for example the *expectation*  $EX$  of  $X$  can be defined simply as the *center* of its *weighted graph*. However, the mapping has limitations with regards to a *sequence* of random variables in performing sequence analysis (using for example *Fourier analysis* or *wavelet analysis*), in performing sequence processing (using for example *FIR filtering* or *IIR filtering*), in making diagnostic measurements (using a post-transform metric space), or in making “optimal” decisions (based on “distance” measurements in a *metric space* or more generally a *distance space*).

Rather than mapping to a *weighted graph*, this paper proposes instead mapping to an *ordered distance linear space*  $Y \triangleq (\mathbb{R}^n, \leq, d, +, \cdot, \mathbb{R}, +, \dot{\times})$ , where  $(\mathbb{R}, +, \dot{\times})$  is a *field*,  $+$  is the vector addition operator on  $\mathbb{R}^n \times \mathbb{R}^n$ , and  $\cdot$  is the scalar-vector multiplication operator on  $\mathbb{R} \times \mathbb{R}^n$ . The *linear space* component of  $Y$  provides a much more convenient (as compared to the *weighted graph*) framework for sequence analysis and processing. The *ordered set* and *distance space* components of  $Y$  allow one to preserve the order structure and distance geometry inherent in the underlying stochastic process, which in turn likely provides a less distorted (as compared to the *real line*) framework for analysis, diagnostics, and optimal decision making.

*2010 Mathematics Subject Classification:* 60B99,60G05 (primary); 60E15,60G20,62F03,62P10,92D20 (secondary)

*Universal Decimal Classification (UDC):* 519.2+519.6+601

*Keywords:* metric linear space, signal processing, GSP, genomic signal processing, genomic sequence processing

## Contents

<b>1 Standard definitions</b>	<b>3</b>
1.1 Sets . . . . .	3
1.2 Relations . . . . .	3
1.3 Order . . . . .	5
1.4 Field operator pairs . . . . .	7
1.5 Sequences . . . . .	7
1.6 Filtering . . . . .	9



1.7 Discrete Fourier Transform . . . . .	10
1.8 Probability . . . . .	10
<b>2 Introduction</b>	<b>11</b>
2.1 Stochastic processing . . . . .	11
2.2 Outcome subspaces . . . . .	11
2.3 Single symbolic event processing using weighted graphs . . . . .	13
2.4 Symbolic sequence processing using linear spaces on $R^n$ . . . . .	14
2.5 Motivation for metrics in symbolic sequence processing . . . . .	14
<b>3 Outcome subspace sequences</b>	<b>15</b>
3.1 Definitions . . . . .	15
3.2 Examples of symbolic sequence statistics . . . . .	15
3.3 Extending to distance linear spaces . . . . .	20
3.3.1 Motivation . . . . .	20
3.3.2 Some random variables . . . . .	21
3.3.3 Some ordered distance linear spaces . . . . .	22
<b>4 Symbolic sequence processing results</b>	<b>23</b>
4.1 Low pass filtering/Smoothing . . . . .	23
* low pass filtering of real die sequence . . . . .	23
* low pass filtering of spinner sequence . . . . .	26
* low pass filtering of fair die sequence . . . . .	27
4.2 High pass filtering . . . . .	29
* high pass filtering of weighted real die sequence . . . . .	29
* high pass filtering of weighted spinner sequence . . . . .	31
* high pass filtering of weighted die sequence . . . . .	33
4.3 Fourier Analysis . . . . .	35
* length 1200 non-stationary die sequence with 10Hz oscillating mean . . . . .	35
* length 12000 non-stationary die sequence with 10Hz oscillating mean . . . . .	36
* length 12000 non-stationary die sequence with 100Hz oscillating mean . . . . .	37
* length 12000 non-stationary artificial dna sequence with 10Hz oscillating mean . . . . .	38
* Fourier analysis of Ebola DNA sequence . . . . .	39
4.4 Wavelet Analysis . . . . .	40
* statistical edge detection using Haar wavelet on non-stationary die sequence . . . . .	41
* statistical edge detection using Haar wavelet on non-stationary artificial DNA sequence . . . . .	42
* Wavelet analysis of Phage Lambda DNA sequence . . . . .	43
<b>A Distance spaces</b>	<b>45</b>
<b>B Metric spaces</b>	<b>46</b>
<b>C Lagrange arc distance</b>	<b>46</b>
<b>D C++ source code support</b>	<b>49</b>
D.1 Symbolic sequence routines . . . . .	49
D.2 Die routines . . . . .	55
D.3 Real die routines . . . . .	61
D.4 Spinner routines . . . . .	68
D.5 DNA routines . . . . .	75
<b>Reference Index</b>	<b>92</b>
<b>Subject Index</b>	<b>93</b>

## 1 Standard definitions

The definitions in this section are more or less standard. Some notation and forms of expression may be non-standard.

### 1.1 Sets

**Definition 1.1**<sup>1</sup> Let  $X$  be a *set*.<sup>2</sup> The **empty set**  $\emptyset$  is defined as  $\emptyset \triangleq \{x \in X | x \neq x\}$ .

**Definition 1.2**<sup>3</sup> Let  $\mathbb{R}$  be the **set of real numbers**. Let  $\mathbb{R}^+ \triangleq \{x \in \mathbb{R} | x \geq 0\}$  be the **set of non-negative real numbers**. Let  $\mathbb{R}^+ \triangleq \{x \in \mathbb{R} | x > 0\}$  be the **set of positive real numbers**. Let  $\mathbb{Z}$  be the **set of integers**. Let  $\mathbb{W} \triangleq \{n \in \mathbb{Z} | n \geq 0\}$  be the **set of whole numbers**. Let  $\mathbb{N} \triangleq \{n \in \mathbb{Z} | n \geq 1\}$  be the **set of natural numbers**. Let  $\mathbb{Z}^* \triangleq \mathbb{Z} \cup \{-\infty, \infty\}$  be the *extended set of integers*.

### 1.2 Relations

One of the most fundamental structures in mathematics is the *ordered pair*, and one of the most common definitions of *ordered pair* is due to Kuratowski (1921) and is presented next:<sup>4</sup>

**Definition 1.3**<sup>5</sup> The **ordered pair**  $(a, b)$  is defined as  $(a, b) \triangleq \{\{a\}, \{a, b\}\}$ .

Proposition 1.4 (next) and Corollary 1.5 demonstrate that the definition of *ordered pair* given by Definition 1.3 allows  $a$  and  $b$  to be unambiguously extracted from  $(a, b)$  and that  $(a, b)$  is well defined.

#### Proposition 1.4

$$\begin{aligned} \{a\} &= \bigcap (a, b) = \bigcap \{\{a\}, \{a, b\}\} = \{a\} \cap \{a, b\} \\ \{b\} &= \Delta(a, b) = \Delta\{\{a\}, \{a, b\}\} = \{a\} \Delta \{a, b\} \end{aligned}$$

**Corollary 1.5**<sup>6</sup>  $(a, b) = (c, d) \iff \{a = c \text{ and } b = d\}$

PROOF:

$\square$	$\{a\} = \bigcap (a, b) = \bigcap (c, d) = \{c\}$	by Proposition 1.4 and left hypothesis	$\square$
$\{b\} = \Delta(a, b) = \Delta(c, d) = \{d\}$	by Proposition 1.4 and left hypothesis		
$(a, b) = (c, d)$	by right hypothesis		

<sup>1</sup> [Halmos \(1960\) page 8](#), [Kelley \(1955\) page 3](#), [Kuratowski \(1961\), page 26](#)

<sup>2</sup> The mathematical structure called *set* is left undefined in this paper. For more information on *sets*, see for example [Zermelo \(1908a\)](#) pages 263–267 (7 axioms), [Zermelo \(1908b\)](#) ((English translation of previous reference)), [Fraenkel \(1922\)](#), [Halmos \(1960\) pages 1–6](#) (Naive set theory), [Wolf \(1998\)](#), page 139

<sup>3</sup> Notation  $\mathbb{R}, \mathbb{W}, \mathbb{N}$ , etc.: *Bourbaki notation*. References: [Davis \(2005\) page 9](#), [Cohn \(2012\) page 3](#)

<sup>4</sup> As an alternative to the Kuratowski definition, the *ordered pair* can also be taken as an *axiom*. References:

[Bourbaki \(1968\)](#), page 72, [Munkres \(2000\)](#), page 13

<sup>5</sup> [Suppes \(1972\) page 32](#), [Halmos \(1960\) page 23](#), [Kuratowski \(1961\)](#), page 39, [Kuratowski \(1921\) \(Def. V, page 171\)](#), [Wiener \(1914\)](#)

<sup>6</sup> [Apostol \(1975\) page 33](#), [Hausdorff \(1937\) page 15](#)



**Definition 1.6** <sup>7</sup> Let  $X$  and  $Y$  be sets. The **Cartesian product**  $X \times Y$  is defined as

$$X \times Y \triangleq \{(x, y) | (x \in X) \text{ and } (y \in Y)\}$$

**Definition 1.7** <sup>8</sup> Let  $X$  and  $Y$  be sets. A **relation**  $\mathbb{R}$  on  $X$  and  $Y$  is any subset of  $X \times Y$  such that  $\mathbb{R} \subseteq X \times Y$ . The set  $2^{XY}$  is the **set of all relations** in  $X \times Y$ .

**Definition 1.8** <sup>9</sup> Let  $X$  and  $Y$  be sets. A relation  $f \in 2^{XY}$  is a **function** if

$$\{(x, y_1) \in f \text{ and } (x, y_2) \in f\} \implies \{y_1 = y_2\}. \text{ The set } Y^X \text{ is the } \mathbf{\text{set of all functions}} \text{ in } 2^{XY}.$$

A function does not always have an inverse that is also a function. But unlike functions, *every* relation has an inverse that is also a relation. Note that since all functions are relations, every function *does* have an inverse that is at least a relation, and in some cases this inverse is also a function.

**Definition 1.9** <sup>10</sup> Let  $\mathbb{R}$  be a relation in  $2^{XY}$ .

$\mathbb{R}^{-1}$  is the **inverse** of relation  $\mathbb{R}$  if

$$\mathbb{R}^{-1} \triangleq \{(y, x) \in Y \times X | (x, y) \in \mathbb{R}\}$$

The inverse relation  $\mathbb{R}^{-1}$  is also called the **converse** of  $\mathbb{R}$ .

**Definition 1.10** Let  $X$  be a set. The quantity  $2^X$  is the *power set of*  $X$  such that

$$2^X \triangleq \{A \subseteq X\} \quad (\text{the set of all subsets of } X).$$

**Definition 1.11** Let  $Y$  be a set. The structure  $Y^n$  for  $n \in \mathbb{N}$  is a *set* defined as

$$Y^1 \triangleq Y \quad \text{and}$$

$$Y^n \triangleq Y \times Y^{n-1} \quad \text{for } n = 2, 3, 4, \dots$$

**Definition 1.12** The **set of complex numbers**  $\mathbb{C}$  is defined as  $\mathbb{C} \triangleq \mathbb{R}^2$ .

**Definition 1.13** Let  $Y_1, Y_2, \dots, Y_N$  be sets. The structure  $(x_1, x_2, \dots, x_n)$  is an **n-tuple** on  $Y_1 \times Y_2 \times \dots \times Y_N$  if  $(x_1, x_2, \dots, x_n)$  is an element in the set  $Y_1 \times Y_2 \times \dots \times Y_N$ . A **3-tuple** is also called an **ordered triple** or simply a **tuple**.

**Definition 1.14** <sup>11</sup> Let  $\mathbb{R} \in 2^{XY}$  be a *relation* (Definition 1.7 page 4).

The **domain** of  $\mathbb{R}$  is  $D(\mathbb{R}) \triangleq \{x \in X | \exists y \text{ such that } (x, y) \in \mathbb{R}\}$ .

The **image set** of  $\mathbb{R}$  is  $I(\mathbb{R}) \triangleq \{y \in Y | \exists x \text{ such that } (x, y) \in \mathbb{R}\}$ .

The **null space** of  $\mathbb{R}$  is  $N(\mathbb{R}) \triangleq \{x \in X | (x, 0) \in \mathbb{R}\}$ .

The **range** of  $\mathbb{R}$  is any set  $R(\mathbb{R})$  such that  $I(\mathbb{R}) \subseteq R(\mathbb{R})$ .

<sup>7</sup> ↗ Halmos (1960) page 24, G. Frege, 2007 August 25, <http://groups.google.com/group/sci.logic/msg/3b3294f5ac3a76f0>

<sup>8</sup> ↗ Maddux (2006) page 4, ↗ Halmos (1960) pages 26–30, ↗ Suppes (1972) page 86, ↗ Kelley (1955) page 10, ↗ Bourbaki (1939), ↗ Bottazzini (1986) page 7, ↗ Comtet (1974) page 4 ( $|Y^X|$ ); The notation  $2^{XY}$  is motivated by the fact that for finite  $X$  and  $Y$ ,  $|2^{XY}| = 2^{|X| \cdot |Y|}$ .

<sup>9</sup> ↗ Maddux (2006) page 4, ↗ Halmos (1960) pages 26–30, ↗ Suppes (1972) page 86, ↗ Kelley (1955) page 10, ↗ Bourbaki (1939), ↗ Bottazzini (1986) page 7, ↗ Comtet (1974) page 4 ( $|Y^X|$ ); The notation  $Y^X$  is motivated by the fact that for finite  $X$  and  $Y$ ,  $|Y^X| = |Y|^{|X|}$ .

<sup>10</sup> ↗ Suppes (1972) page 61 (Defintion 6, inverse=“converse”)

↗ Kelley (1955) page 7

↗ Peirce (1883) page 188 (inverse=“converse”)

<sup>11</sup> ↗ Munkres (2000), page 16, ↗ Kelley (1955) page 7

**Definition 1.15** The *set function*<sup>12</sup>  $|A| \in \mathbb{Z}^{*2^X}$  is the *cardinality* of  $A$  such that

$$|A| \triangleq \begin{cases} \text{the number of elements in } A & \text{for finite } A \\ \infty & \text{otherwise} \end{cases} \quad \forall A \in 2^X$$

### 1.3 Order

**Definition 1.16**<sup>13</sup> Let  $X$  be a set. A relation  $\leq$  is an **order relation** in  $2^{XX}$  (Definition 1.7 page 4) if

- |    |   |                         |                   |     |                                 |
|----|---|-------------------------|-------------------|-----|---------------------------------|
| 1. | $x \leq x$                                  | $\forall x \in X$       | (reflexive)       | and | $\boxed{\quad}$ <i>preorder</i> |
| 2. | $x \leq y$ and $y \leq z \implies x \leq z$ | $\forall x, y, z \in X$ | (transitive)      | and | $\boxed{\quad}$                 |
| 3. | $x \leq y$ and $y \leq x \implies x = y$    | $\forall x, y \in X$    | (anti-symmetric). |     |                                 |

An **ordered set** is the pair  $(X, \leq)$ . If  $\leq = \emptyset$  (Definition 1.1 page 3), then  $(X, \leq)$  is an **unordered set**. If  $x \leq y$  or  $y \leq x$ , then elements  $x$  and  $y$  are said to be **comparable**; otherwise they are **incomparable**. The relation  $<$  is the relation  $\leq \setminus =$  (“less than but not equal to”), where  $\setminus$  is the *set difference* operator, and  $=$  is the equality relation.

**Definition 1.17**<sup>14</sup> In an *ordered set*  $(X, \leq)$ ,

- the set  $[x : y] \triangleq \{z \in X | x \leq z \leq y\}$  is a **closed interval** on  $(X, \leq)$  and  
 the set  $(x : y) \triangleq \{z \in X | x < z < y\}$  is an **open interval** on  $(X, \leq)$

**Definition 1.18**<sup>15</sup> Let  $(X, \leq)$  be an *ordered set*. A subset  $D \subseteq X$  is **convex** in  $X$  if

$$x, y \in D \implies (x : y) \subseteq D.$$

**Example 1.19** Convex subsets of  $\mathbb{Z}$  under the usual integer ordering relation include

$$\emptyset, \mathbb{Z}, \mathbb{W}, \mathbb{N}, \{0, 1, 2, 3, 4, 5\}, \text{ and } \{-2, -1, 0, 1, 2, 3\}.$$

**Definition 1.20** Let  $(X, \leq)$  be an ordered set. For any set  $A \in 2^X$ ,  $c$  is an **upper bound** of  $A$  in  $(X, \leq)$  if  $x \leq c \quad \forall x \in A$ . An element  $b$  is the **least upper bound**, or **lub**, of  $A$  in  $(X, \leq)$  if  $b$  and  $c$  are *upper bounds* of  $A \implies b \leq c$ .

The least upper bound of the set  $A$  is denoted  $\bigvee A$ .

The **join**  $x \vee y$  of  $x$  and  $y$  is defined as  $x \vee y \triangleq \bigvee \{x, y\}$ .

**Definition 1.21** Let  $(X, \leq)$  be an ordered set. For any set  $A \in 2^X$ ,  $p$  is a **lower bound** of  $A$  in  $(X, \leq)$  if  $p \leq x \quad \forall x \in A$ . An element  $a$  is the **greatest lower bound**, or **glb**, of  $A$  in  $(X, \leq)$  if  $a$  and  $p$  are *lower bounds* of  $A \implies p \leq a$ .

The greatest lower bound of the set  $A$  is denoted  $\bigwedge A$ .

The **meet**  $x \wedge y$  of  $x$  and  $y$  is defined as  $x \wedge y \triangleq \bigwedge \{x, y\}$ .

<sup>12</sup>set function: Pap (1995) page 8 (Definition 2.3: extended real-valued set function), Halmos (1950) page 30 (§7. MEASURE ON RINGS)

<sup>13</sup> MacLane and Birkhoff (1999) page 470, Beran (1985) page 1, Korselt (1894) page 156 (I, II, (1)), Dedekind (1900) page 373 (I–III). An *order relation* is also called a **partial order relation**. An *ordered set* is also called a **partially ordered set** or **poset**.

<sup>14</sup> Apostol (1975) page 4, Ore (1935) page 409, Duthie (1942) page 2, Ore (1935) page 425 (quotient structures)

<sup>15</sup> Barvinok (2002) page 5

**Definition 1.22** <sup>16</sup> An algebraic structure  $L \triangleq (X, \vee, \wedge; \leq)$  is a **lattice** if

1.  $(X, \leq)$  is an ordered set and
2.  $x, y \in X \implies x \vee y \in X$  and
3.  $x, y \in X \implies x \wedge y \in X$

A *lattice L* is **linear** if for every  $x, y$  in  $X$ ,  $x \leq y$ ,  $y \leq x$ , or both.

**Definition 1.23** <sup>17</sup> Let  $X \triangleq (X, \leq)$  and  $Y \triangleq (Y, \preceq)$  be ordered sets.

A function  $\theta \in Y^X$  is **order preserving** in  $(X, Y)$  if  $(x \leq y) \implies (\theta(x) \preceq \theta(y)) \quad \forall x, y \in X$ .

**Definition 1.24** Let  $L_1 \triangleq (X, \vee, \wedge; \leq)$  and  $L_2 \triangleq (Y, \oslash, \oslash; \preceq)$  be *lattices*.

$L_1$  and  $L_2$  are **isomorphic** on  $(X, Y)$  if there exists a function  $\theta \in Y^X$  such that

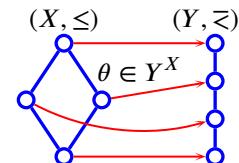
1.  $\theta(x \vee y) = \theta(x) \oslash \theta(y) \quad \forall x, y \in X \quad (\text{preserves joins})$  and
2.  $\theta(x \wedge y) = \theta(x) \oslash \theta(y) \quad \forall x, y \in X \quad (\text{preserves meets}).$

In this case, the function  $\theta$  is said to be an **isomorphism** from  $L_1$  to  $L_2$ , and the isomorphic relationship between  $L_1$  and  $L_2$  is denoted as  $L_1 \equiv L_2$ .

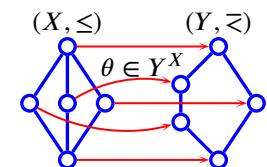
**Theorem 1.25** <sup>18</sup> Let  $(X, \vee, \wedge; \leq)$  and  $(Y, \oslash, \oslash; \preceq)$  be lattices and  $\theta \in Y^X$  be a BIJECTIVE function with inverse  $\theta^{-1} \in X^Y$ .

$$\underbrace{\begin{array}{c} x_1 \leq x_2 \implies \theta(x_1) \preceq \theta(x_2) \quad \forall x_1, x_2 \in X \\ y_1 \preceq y_2 \implies \theta^{-1}(y_1) \preceq \theta^{-1}(y_2) \quad \forall y_1, y_2 \in Y \end{array}}_{\theta \text{ and } \theta^{-1} \text{ are ORDER PRESERVING}} \quad \left. \right\} \Leftrightarrow \underbrace{(X, \vee, \wedge; \leq) \equiv (Y, \oslash, \oslash; \preceq)}_{\text{ISOMORPHIC}}$$

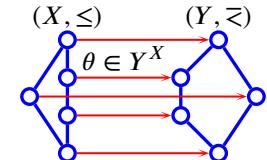
**Example 1.26** <sup>19</sup> In the diagram to the right, the function  $\theta \in Y^X$  is *order preserving* with respect to  $\leq$  and  $\preceq$ . Note that  $\theta^{-1}$  is *not order preserving* and that the ordered sets  $(X, \leq)$  and  $(Y, \preceq)$  are *not isomorphic*—as already demonstrated by Theorem 1.25 that they cannot be.



**Example 1.27** In the diagram to the right, the function  $\theta \in Y^X$  is *order preserving* with respect to  $\leq$  and  $\preceq$ . Note that  $\theta^{-1}$  is *not order preserving*. Like Example 1.26 (page 6), this example also illustrates the fact that that *order preserving* does not imply *isomorphic*.



**Example 1.28** In the diagram to the right, the function  $\theta \in Y^X$  is *order preserving* with respect to  $\leq$  and  $\preceq$ . Note that  $\theta^{-1}$  is also *order preserving* and that the ordered sets  $(X, \leq)$  and  $(Y, \preceq)$  are *not isomorphic*—as already demonstrated by Theorem 1.25 that they must be.



**Definition 1.29** Let  $(\mathbb{R}, \leq)$  be the *standard ordered set of real numbers*. The *floor function*  $\lfloor x \rfloor \in \mathbb{Z}^\mathbb{R}$  and the *ceiling function*  $\lceil x \rceil \in \mathbb{Z}^\mathbb{R}$  are defined as  $\lceil x \rceil \triangleq \bigwedge \{n \in \mathbb{Z} | n \geq x\}$  and  $\lfloor x \rfloor \triangleq \bigvee \{n \in \mathbb{Z} | n \leq x\}$ .

<sup>16</sup> MacLane and Birkhoff (1999) page 473, Birkhoff (1948) page 16, Ore (1935), Birkhoff (1933) page 442, Maeda and Maeda (1970), page 1

<sup>17</sup> Burris and Sankappanavar (2000), page 10

<sup>18</sup> Burris and Sankappanavar (2000), page 10

<sup>19</sup> Burris and Sankappanavar (2000), page 10

## 1.4 Field operator pairs

**Definition 1.30** Let  $X$ ,  $Y$ , and  $Z$  be sets. Let  $(\mathbb{R}, +, \cdot)$  be the standard field of real numbers. The **addition operator**  $\oplus : X \times Y \rightarrow Z$  is defined as shown in Table 1 (page 7). Moreover, for some sequence  $(x_n)$ ,

$$\bigoplus_{n=1}^N x_n \triangleq x_1 \oplus x_2 \oplus \cdots \oplus x_N \quad \text{and} \quad \bigoplus_{n \in \mathbb{D}} x_n \triangleq x_{\alpha \in \mathbb{D}} \oplus x_{\beta \in \mathbb{D}} \oplus \cdots \oplus x_{\gamma \in \mathbb{D}}$$

1.	If $X \times Y \triangleq \mathbb{R} \times \mathbb{R}$	then $x \oplus y$	$\triangleq$	$x + y$	$\in \mathbb{R}$	.
2.	If $X \times Y \triangleq \mathbb{C} \times \mathbb{C}$	then $(a, b) \oplus (c, d)$	$\triangleq$	$(a + c, b + d)$	$\in \mathbb{C}$	.
3.	If $X \times Y \triangleq \mathbb{R}^n \times \mathbb{R}^n$	then $(x_1, x_2, \dots, x_n) \oplus (y_1, y_2, \dots, y_n)$	$\triangleq$	$(x_1 + y_1, \dots, x_n + y_n)$	$\in \mathbb{R}^n$	.
4.	If $X \times Y \triangleq \mathbb{C}^n \times \mathbb{C}^n$	then $(x_1, x_2, \dots, x_n) \oplus (y_1, y_2, \dots, y_n)$	$\triangleq$	$(x_1 \oplus y_1, \dots, x_n \oplus y_n)$	$\in \mathbb{C}^n$	.
5.	If $X \times Y \triangleq \mathbb{R} \times \mathbb{C}$	then $x \oplus (a, b)$	$\triangleq$	$(x + a, x + b)$	$\in \mathbb{C}$	.
6.	If $X \times Y \triangleq \mathbb{C} \times \mathbb{R}$	then $x \oplus y$	$\triangleq$	$y \oplus x$	$\in \mathbb{C}$	.

Table 1: Definition of the addition operator  $\oplus : X \times Y \rightarrow Z$  (see Definition 1.30 page 7)

**Definition 1.31** Let  $X$ ,  $Y$ , and  $Z$  be sets. Let  $(\mathbb{R}, +, \cdot)$  be the standard field of real numbers. Let the juxtaposition operator  $xy$  on  $x$  and  $y$  be equivalent to the real field operator  $x \cdot y$  on  $x$  and  $y$  such that  $xy \triangleq x \cdot y$ . The **multiplication operator**  $\otimes : X \times Y \rightarrow Z$  is defined as shown in Table 2 (page 7).

1.	If $X \times Y \triangleq \mathbb{R} \times \mathbb{R}$	then $x \otimes y$	$\triangleq$	$xy$	$\in \mathbb{R}$	.
2.	If $X \times Y \triangleq \mathbb{R} \times \mathbb{C}$	then $x \otimes (a, b)$	$\triangleq$	$(xa, xb)$	$\in \mathbb{C}$	.
3.	If $X \times Y \triangleq \mathbb{C} \times \mathbb{C}$	then $(a, b) \otimes (c, d)$	$\triangleq$	$(ac - bd, ad + bc)$	$\in \mathbb{C}$	.
4.	If $X \times Y \triangleq \mathbb{R} \times \mathbb{R}^n$	then $x \otimes (y_1, y_2, \dots, y_n)$	$\triangleq$	$(xy_1, xy_2, \dots, xy_n)$	$\in \mathbb{R}^n$	.
5.	If $X \times Y \triangleq \mathbb{C} \times \mathbb{R}^n$	then $x \otimes (y_1, y_2, \dots, y_n)$	$\triangleq$	$(y_1 \otimes x, y_2 \otimes x, \dots, y_n \otimes x)$	$\in \mathbb{C}^n$	.
6.	If $X \times Y \triangleq \mathbb{C} \times \mathbb{C}^n$	then $x \otimes (y_1, y_2, \dots, y_n)$	$\triangleq$	$(x \otimes y_1, x \otimes y_2, \dots, x \otimes y_n)$	$\in \mathbb{C}^n$	.
7.	If $X \times Y \triangleq \mathbb{C} \times \mathbb{R}$	then $x \otimes y$	$\triangleq$	$y \otimes x$	$\in \mathbb{C}$	.
8.	If $X \times Y \triangleq \mathbb{R}^n \times \mathbb{R}$	then $x \otimes y$	$\triangleq$	$y \otimes x$	$\in \mathbb{R}^n$	.
9.	If $X \times Y \triangleq \mathbb{R}^n \times \mathbb{C}$	then $x \otimes y$	$\triangleq$	$y \otimes x$	$\in \mathbb{C}^n$	.
10.	If $X \times Y \triangleq \mathbb{C}^n \times \mathbb{C}$	then $x \otimes y$	$\triangleq$	$y \otimes x$	$\in \mathbb{C}^n$	.

Table 2: Definition of the multiplication operation  $\otimes : X \times Y \rightarrow Z$  (see Definition 1.31 page 7)

## 1.5 Sequences

**Definition 1.32**<sup>20</sup> A function in  $X^{\mathbb{D}}$  (Definition 1.8 page 4) is an  $X$ -valued **sequence** if  $\mathbb{D} \neq \emptyset$  and  $\mathbb{D}$  is a convex (Definition 1.18 page 5) subset of  $\mathbb{Z}$ . A sequence may be denoted in the form  $(x_n)_{n \in \mathbb{D}}$ , or simply as  $(x_n)$ .

**Definition 1.33** The sequence  $(y_n)_{\mathbb{D}_2}$  is the sequence  $(x_n)_{\mathbb{D}_1}$  **down sampled by a factor of  $M$** , where  $M \in \mathbb{N}$ , if  $n \in \mathbb{D}_2 \iff Mn \in \mathbb{D}_1$  and  $y_n = x_{Mn} \quad \forall n \in \mathbb{D}_2$ .

<sup>20</sup>  Simmons (2016) (‘Formal definition’)

**Definition 1.34** Let  $\oplus$  be the *addition operator* (Definition 1.30 page 7) and  $\otimes$  the *multiplication operator* (Definition 1.31 page 7). Let  $\mathbb{D}_1$  and  $\mathbb{D}_2$  be convex subsets of  $\mathbb{Z}$ . Let  $\mathbb{D} \triangleq (\bigwedge \mathbb{D}_1 + \bigwedge \mathbb{D}_2 - 1 : \bigvee \mathbb{D}_1 + \bigvee \mathbb{D}_2 + 1)$ . Let  $(x_n)_{\mathbb{D}_1}$  be a sequence over a field  $\mathbb{F}_1$  and  $(y_n)_{\mathbb{D}_2}$  a sequence over a field  $\mathbb{F}_2$ .

The **convolution**  $(z_n)_{\mathbb{D}} \triangleq (x_n)_{\mathbb{D}_1} \star (y_n)_{\mathbb{D}_2}$  of  $(x_n)$  and  $(y_n)$  is defined as

$$z_n \triangleq \bigoplus_{m \in \mathbb{D}_1} f(n, m) \quad \text{where } f \text{ is defined as } f(n, m) \triangleq \begin{cases} x_m \otimes y_{n-m} & \text{if } m \in \mathbb{D}_1 \text{ and } (n - m) \in \mathbb{D}_2 \\ 0 & \text{otherwise} \end{cases} \quad \forall n, m \in \mathbb{D}$$

**Proposition 1.35** Let  $(x_n)$  and  $(y_n)$  be finite sequences with lengths  $N$  and  $M$ , respectively. Then the length of  $(x_n) \star (y_n)$  is  $N + M - 1$ .

**Example 1.36**<sup>21</sup> Let  $(x_n)_{n \in \mathbb{Z}}$  and  $(y_n)_{n \in \mathbb{Z}}$  be sequences over a field  $\mathbb{F}$ . Then the domain  $\mathbb{D}$  of the convolution  $(z_n)_{n \in \mathbb{D}} \triangleq (x_n) \star (y_n)$  is  $\mathbb{D} \triangleq (\bigwedge \mathbb{Z} + \bigwedge \mathbb{Z} - 1 : \bigvee \mathbb{Z} + \bigvee \mathbb{Z} + 1) = \mathbb{Z}$  and  $z_n \triangleq \sum_{m \in \mathbb{Z}} x_m y_{n-m}$ .

**Example 1.37** Let  $(x_n)_{[0:1]} \triangleq (1, 2)$  and  $(y_n)_{[0:2]} \triangleq (10, 20, 50)$  be sequences over the field  $(\mathbb{R}, +, \cdot)$ . Then the domain  $\mathbb{D}$  of the convolution  $(z_n)_{n \in \mathbb{D}} \triangleq (x_n) \star (y_n)$  is  $\mathbb{D} \triangleq (0 + 0 - 1 : 1 + 2 + 1) = [0 : 3] = \{0, 1, 2, 3\}$  and

$$\begin{aligned} (z_n)_{n \in \mathbb{D}} &\triangleq \left( \left( \sum_{m \in \{0,1\}} f(0, m), \sum_{m \in \{0,1\}} f(1, m), \sum_{m \in \{0,1\}} f(2, m), \sum_{m \in \{0,1\}} f(3, m) \right) \right)_{\{0,1,2,3\}} \\ &\triangleq ((1 \times 10 + 0), (1 \times 20 + 2 \times 10), (1 \times 50 + 2 \times 20), (0 + 2 \times 50))_{\{0,1,2,3\}} \\ &= \left( \left( \underbrace{10}_{z_0}, \underbrace{40}_{z_1}, \underbrace{90}_{z_2}, \underbrace{100}_{z_3} \right) \right)_{\{0,1,2,3\}} \end{aligned}$$

**Example 1.38** Let  $(x_n)_{[0:1]} \triangleq (1, 2)$  and  $(y_n)_{[3:5]} \triangleq (10, 20, 50)$  be sequences over the field  $(\mathbb{R}, +, \cdot)$ . Then the domain  $\mathbb{D}$  of the convolution  $(z_n)_{n \in \mathbb{D}} \triangleq (x_n) \star (y_n)$  is  $\mathbb{D} \triangleq (0 + 3 - 1 : 1 + 5 + 1) = [3 : 6] = \{3, 4, 5, 6\}$  and

$$\begin{aligned} (z_n)_{n \in \mathbb{D}} &\triangleq \left( \left( \sum_{m \in \{0,1\}} f(3, m), \sum_{m \in \{0,1\}} f(4, m), \sum_{m \in \{0,1\}} f(5, m), \sum_{m \in \{0,1\}} f(6, m) \right) \right)_{\{3,4,5,6\}} \\ &\triangleq ((1 \times 10 + 0), (1 \times 20 + 2 \times 10), (1 \times 50 + 2 \times 20), (0 + 2 \times 50))_{\{3,4,5,6\}} \\ &= \left( \left( \underbrace{10}_{z_3}, \underbrace{40}_{z_4}, \underbrace{90}_{z_5}, \underbrace{100}_{z_6} \right) \right)_{\{3,4,5,6\}} \end{aligned}$$

**Example 1.39** Let  $(x_n)_{[0:1]} \triangleq (1, 2)$  and  $(y_n)_{[0:2]} \triangleq ((3, 4), (5, 6), (7, 8))$  be sequences.

<sup>21</sup>historical references: Cauchy (1821) (Chapter IV), Apostol (1975) page 204 (note that convolution is a single element in a series that is the “Cauchy product”), Dominguez-Torres (2010) page 20 (section 4.2: connection to the work of Cauchy), Dominguez-Torres (2015) (history of the continuous convolution operation)

Then the *domain*  $\mathbb{D}$  of the convolution  $(z_n)_{n \in \mathbb{D}} \triangleq (x_n) \star (y_n)$  is  $\mathbb{D} = \{0, 1, 2, 3\}$  and

$$\begin{aligned} (z_n)_{n \in \mathbb{D}} &\triangleq \left( \left( \bigoplus_{m \in \{0,1\}} f(0, m), \bigoplus_{m \in \{0,1\}} f(1, m), \bigoplus_{m \in \{0,1\}} f(2, m), \bigoplus_{m \in \{0,1\}} f(3, m) \right) \right)_{\{0,1,2,3\}} \\ &\triangleq ([1 \otimes (3, 4) \oplus 0], [1 \otimes (5, 6) \oplus 2 \otimes (3, 4)], [1 \otimes (7, 8) \oplus 2 \otimes (5, 6)], [0 \oplus 2 \otimes (7, 8)])_{\{0,1,2,3\}} \\ &\triangleq [(3, 4)], [(5, 6) \oplus (6, 8)], [(7, 8) \oplus (10, 12)], [(14, 16)]_{\{0,1,2,3\}} \\ &= \left( \left( \underbrace{(3, 4)}_{z_0}, \underbrace{(11, 14)}_{z_1}, \underbrace{(17, 20)}_{z_2}, \underbrace{(14, 16)}_{z_3} \right) \right)_{\{0,1,2,3\}} \end{aligned}$$

**Definition 1.40** Let  $(x_n)_{n \in \mathbb{D}}$  and  $(y_n)_{n \in \mathbb{D}}$  be sequences over a field  $\mathbb{F} \triangleq (X, +, \cdot)$ , and  $\alpha$  an element in  $\mathbb{F}$ . The operations  $\alpha + (x_n)$ ,  $(x_n) + \alpha$ ,  $\alpha(x_n)$ , and  $(x_n)\alpha$  are defined as

$$\begin{aligned} \alpha + (x_n)_{n \in \mathbb{D}} &\triangleq (x_n)_{n \in \mathbb{D}} + \alpha \triangleq (x_n + \alpha)_{n \in \mathbb{D}} \quad \forall \alpha \in \mathbb{F} \quad \text{and} \\ \alpha(x_n)_{n \in \mathbb{D}} &\triangleq (x_n)_{n \in \mathbb{D}} \alpha \triangleq (\alpha x_n)_{n \in \mathbb{D}} \quad \forall \alpha \in \mathbb{F}. \end{aligned}$$

## 1.6 Filtering

**Definition 1.41** Let  $(x_n)_{\mathbb{D}_1}$  and  $(y_n)_{\mathbb{D}_2}$  be sequences (Definition 1.32 page 7).

The sequence  $(z_n)_{n \in \mathbb{D}}$  is said to be  $(x_n)$  **filtered** by  $(y_n)$  if  $(z_n) \triangleq (x_n) \star (y_n)$  (Definition 1.34 page 8). Moreover, in this case, the operation  $\star (y_n)$  is a **filter** on the sequence  $(x_n)$ .

**Definition 1.42** A **length  $M$  low pass rectangular sequence**  $(h_n)_{n \in [0 : M-1]}$  is here defined as

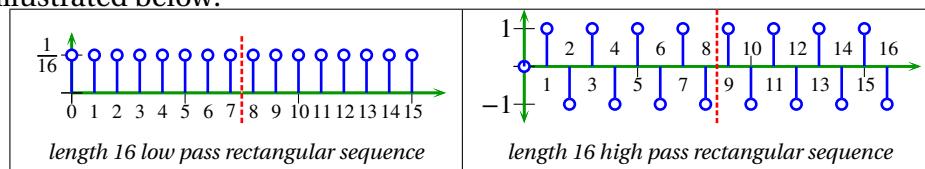
$$h_n = \frac{1}{M} \text{ for } n \in [0 : M-1].$$

**Definition 1.43** A **length  $M$  high pass rectangular sequence**  $(h_n)_{n \in [0 : M]}$  is here defined as

$$h_n \triangleq \begin{cases} 0 & \text{for } n = 0 \\ (-1)^{n+1} & \text{for } n = 1, 2, \dots, M \end{cases}$$

Note that in this definition, the sequence has been offset by 1 on the x-axis from what might normally be expected. This is for the purpose of computational convenience used in Section 4.2 (page 29).

**Example 1.44** A **length 16 low pass rectangular sequence** and **length 16 high pass rectangular sequence** are illustrated below:

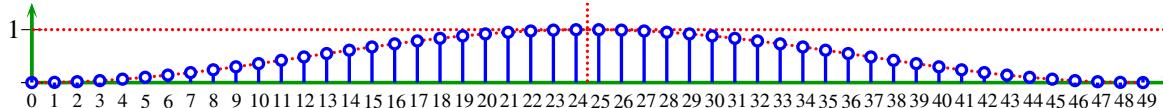


**Definition 1.45**<sup>22</sup> A **length  $M$  low pass Hanning sequence**  $(h_n)_{n \in [0 : M-1]}$  is here defined as

$$h_n \triangleq \frac{1}{2} \left[ 1 - \cos \left( \frac{2\pi n}{M-1} \right) \right] \text{ for } n = 0, 1, 2, \dots, M-1.$$

<sup>22</sup> Blackman and Tukey (1958) page 502 (B.5 Particular Pairs of Windows), Blackman and Tukey (1959), page 98 (B.5 Particular Pairs of Windows), Oppenheim and Schafer (1999) page 763, Prabhu (2013) page 148

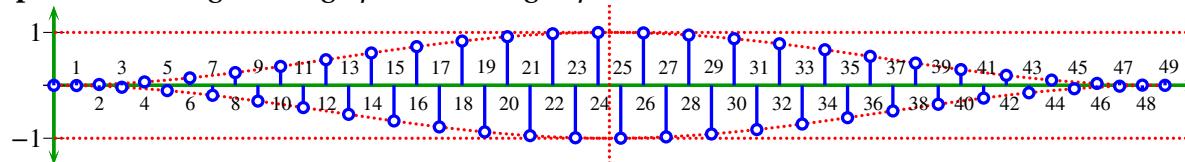
**Example 1.46** A length 50 low pass Hanning sequence is illustrated below:



**Definition 1.47** A length  $M$  high pass Hanning sequence  $(h_n)_{n \in [0 : M-1]}$  is here defined as

$$h_n \triangleq (-1)^n \frac{1}{2} \left[ 1 - \cos \left( \frac{2\pi n}{M-1} \right) \right] \text{ for } n = 0, 1, 2, \dots, M-1$$

**Example 1.48** A length 50 high pass Hanning sequence is illustrated below:



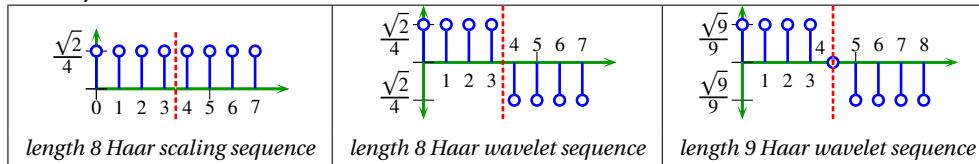
**Definition 1.49** A length  $M$  Haar scaling sequence  $(h_n)_{n \in [0 : M-1]}$  is here defined as

$$h_n = \sqrt{\frac{1}{M}} \quad \text{for } n \in [0 : M-1].$$

**Definition 1.50** A length  $M$  Haar wavelet sequence  $(h_n)_{n \in [0 : M-1]}$  is here defined as

$$h_n \triangleq \begin{cases} +\sqrt{\frac{1}{M}} & \text{for } n = 0, 1, \dots, \lfloor \frac{M}{2} \rfloor - 1 \\ -\sqrt{\frac{1}{M}} & \text{for } n = \lceil \frac{M}{2} \rceil, \lceil \frac{M}{2} \rceil + 1, \dots, M-1 \\ 0 & \text{otherwise} \end{cases}$$

**Example 1.51** A length 8 Haar scaling sequence, a length 8 Haar wavelet sequence, and length 9 Haar wavelet sequence are illustrated below:



## 1.7 Discrete Fourier Transform

**Definition 1.52** Let  $\oplus$  be the *addition operator* (Definition 1.30 page 7) and  $\otimes$  the *multiplication operator* (Definition 1.31 page 7). Let  $(x_n)_{n \in \mathbb{D}}$  be a length  $N$  sequence (Definition 1.32 page 7). The **discrete Fourier transform**  $DFT(x_n)$  of  $(x_n)$  is a sequence  $(y_k)_{k \in \mathbb{D}}$  over  $\mathbb{C}$ , where the element  $y_k$  is defined as

$$y_k \triangleq \sqrt{\frac{1}{N}} \bigoplus_{n \in \mathbb{D}} \left[ x_n \otimes \exp \left( \frac{-i2\pi nk}{N} \right) \right]$$

## 1.8 Probability

**Definition 1.53**<sup>23</sup> Let  $X$  be a set. A function  $P \in \mathbb{R}^{\vdash X}$  is a **probability function** if

- (1).  $P(1) = 1$  *(normalized)* and
- (2).  $P(x) \geq 0 \quad \forall x \in X$  *(nonnegative)* and
- (3).  $x \wedge y = 0 \implies P(x \vee y) = P(x) + P(y) \quad \forall x, y \in X$  *(additive)*.

<sup>23</sup> Papoulis (1991) pages 21–22, Kolmogorov (1933), page 2 (§1. Axioms I–V)

**Definition 1.54** <sup>24</sup> The triple  $(\Omega, \mathbb{E}, P)$  is a **probability space** if  $\Omega$  is a set,  $\mathbb{E}$  is a  $\sigma$ -algebra on  $\Omega$ , and  $P$  is a probability function in  $[0 : 1]^{\mathbb{E}}$ . In this case,  $\Omega$  is called the **set of outcomes**.

The traditional random variable is a mapping from a probability space to the real line. However, before defining random variable formally, note two things that a random variable is *not*:<sup>25</sup>

1. A random variable is **not random**.
2. A random variable is **not a variable**.

**Definition 1.55** <sup>26</sup> A **traditional random variable**  $X$  on a *probability space*  $(\Omega, \mathbb{E}, P)$  is any *function* in the set  $\mathbb{R}^{\Omega}$ .

## 2 Introduction

### 2.1 Stochastic processing

A *traditional random variable*  $X$  (Definition 1.55 page 11) is a *function* that maps from a *set of outcomes* to the real line  $(\mathbb{R}, |\cdot|, \leq)$ , where  $\mathbb{R}$  is the set of real numbers,  $\leq$  is the standard linear order relation on  $\mathbb{R}$ , and  $d(x, y) \triangleq |x - y|$  is the *usual metric* (*Euclidean metric*) on  $\mathbb{R}$ . [Greenhoe \(2015b\)](#) demonstrated that this definition of random variable is often a poor choice of a statistic when the stochastic process that  $X$  maps from is a structure other than the *real line* or some substructure of the *real line*, such as the *integer line*  $(\mathbb{Z}, |\cdot|, \leq)$ . [Greenhoe \(2015b\)](#) further proposed an alternative statistical system, that rather than mapping stochastic processes to the *real line*, instead maps to a *weighted graph* that has order and metric geometry structure compatible to that of the underlying *stochastic process*. In particular, ideally the weighted graph and stochastic process are *isomorphic* and *isometric* with respect to each other.

In this paper, instead of mapping to weighted graphs, we instead use random variables that map to *linear spaces* that are *isomorphic* and *isometric* to the stochastic processes. This approach has the advantage that *linear spaces* are well suited to sequence processing operations.

### 2.2 Outcome subspaces

Here is a review of some key definitions [Greenhoe \(2015b\)](#) which purposed a stochastic processing based on *graph theory*.

**Definition 2.1** A triple  $G \triangleq (X, \leq, d)$  is an **ordered distance space** if  $(X, d)$  is a *distance space* (Definition A.1 page 45) and  $(X, \leq)$  is an *ordered set* (Definition 1.16 page 5). The triple  $G$  is an **ordered metric space** if  $G$  is a *distance space* and  $d$  is a *metric* (Definition B.1 page 46).

**Definition 2.2** <sup>27</sup> The 6-tuple  $(\Omega, \leq, d, \mathbb{E}, P)$  is an **extended probability space** if  $(\Omega, \mathbb{E}, P)$  is a *probability space* (Definition 1.54 page 11) and  $(\Omega, \leq, d)$  is an *ordered distance space* (Definition 2.1). The 4-tuple  $(\Omega, \leq, d, \mathbb{E})$  is an **outcome subspace** if  $(\Omega, \leq, d, \mathbb{E}, P)$  is an *extended probability space*.

<sup>24</sup> [Greenhoe \(2015b\)](#)

<sup>25</sup> [Miller \(2006\) page 130](#)

<sup>26</sup> [Papoulis \(1991\), page 63](#)

<sup>27</sup> [Greenhoe \(2015b\)](#)

**Definition 2.3** A random variable  $X$  on an extended probability space  $(\Omega, \leq, d, \mathbb{E}, P)$  is any function in the set  $Y^\Omega$  (Definition 1.8 page 4), where  $Y$  is a set.

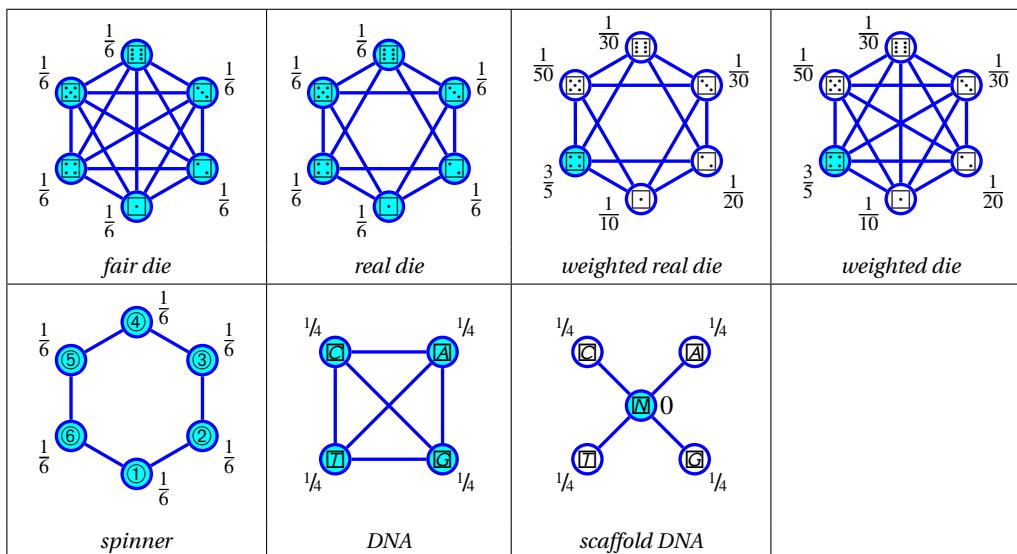


Figure 1: example *outcome subspaces* (Definition 2.2 page 11) illustrated by *weighted graphs* with shaded *expected values*. For more details, see [Greenhoe \(2015b\)](#)

**Definition 2.4**<sup>28</sup> The structure  $G \triangleq (\{\square, \square, \square, \square, \square, \square\}, d, \leq, \dot{P})$  is the **weighted die outcome subspace** if  $G$  is an *outcome subspace*,  $\leq = \emptyset$  (*unordered* Definition 1.16 page 5), and  $d$  is the *discrete metric* (Definition B.2 page 46).

**Definition 2.5**<sup>29</sup> The structure  $G \triangleq (\{\square, \square, \square, \square, \square, \square\}, d, \leq, \dot{P})$  is the **fair die outcome subspace** if  $G$  is a *weighted die outcome subspace* (Definition 2.4), and  $\dot{P}(\square) = \dot{P}(\square) = \dot{P}(\square) = \dot{P}(\square) = \dot{P}(\square) = \dot{P}(\square) = \frac{1}{6}$ .

**Definition 2.6**<sup>30</sup> The structure  $G \triangleq (\{\square, \square, \square, \square, \square, \square\}, d, \emptyset, \dot{P})$  is the **weighted real die outcome subspace** if  $G$  is an *outcome subspace*, and *metric*  $d$  is defined as in the table to the right.

$d(x, y)$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
$\square$	0	1	1	1	1	2
$\square$	1	0	1	1	2	1
$\square$	1	1	0	2	1	1
$\square$	1	1	2	0	1	1
$\square$	1	2	1	1	0	1
$\square$	2	1	1	1	1	0

**Definition 2.7**<sup>31</sup> The structure  $G \triangleq (\{\square, \square, \square, \square, \square, \square\}, d, \emptyset, \dot{P})$  is the **real die outcome subspace** if  $G$  is a *weighted real die outcome subspace* (Definition 2.6 page 12) with

$$\dot{P}(\square) = \dot{P}(\square) = \dot{P}(\square) = \dot{P}(\square) = \dot{P}(\square) = \dot{P}(\square) = \frac{1}{6}.$$

<sup>28</sup>[Greenhoe \(2015b\)](#)

<sup>29</sup>[Greenhoe \(2015b\)](#)

<sup>30</sup>[Greenhoe \(2015b\)](#)

<sup>31</sup>[Greenhoe \(2015b\)](#)

**Definition 2.8** The structure  $G \triangleq (\{①, ②, ③, ④, ⑤, ⑥\}, d, \emptyset, \dot{P})$  is the **spinner outcome subspace** if  $G$  is an *outcome subspace*,

$\dot{P}(①) = \dot{P}(②) = \dot{P}(③) = \dot{P}(④) = \dot{P}(⑤) = \dot{P}(⑥) = \frac{1}{6}$ ,  
and *metric*  $d$  is defined as in the table to the right.

$d(x, y)$	①	②	③	④	⑤	⑥
①	0	1	2	3	2	1
②	1	0	1	2	3	2
③	2	1	0	1	2	3
④	3	2	1	0	1	2
⑤	2	3	2	1	0	1
⑥	1	2	3	2	1	0

**Definition 2.9**<sup>32</sup> The structure  $H \triangleq (\{\blacksquare, \blacksquare, \blacksquare, \blacksquare, \blacksquare\}, d, \emptyset, \dot{P})$  is the **DNA outcome subspace**, or **genome outcome subspace**, if  $H$  is an *outcome subspace*, and  $d$  is the *discrete metric* (Definition B.2 page 46).

**Definition 2.10**<sup>33</sup> The structure  $H \triangleq (\{\blacksquare, \blacksquare, \blacksquare, \blacksquare, \blacksquare\}, d, \leq, \dot{P})$  is the **DNA scaffold outcome subspace**, or **genome scaffold outcome subspace**, if  $H$  is an *outcome subspace*,

$\leq = \{(\blacksquare, \blacksquare), (\blacksquare, \blacksquare), (\blacksquare, \blacksquare), (\blacksquare, \blacksquare), \}$   
( $\blacksquare < \blacksquare, \blacksquare < \blacksquare, \blacksquare < \blacksquare$ , and  $\blacksquare < \blacksquare$ , but otherwise *unordered*),  
 $\dot{P}$  is a *probability function*, and *metric*  $d$  is defined as in the table to the right.

$d(x, y)$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$	$\blacksquare$
$\blacksquare$	0	$\sqrt{2}h$	$\sqrt{2}h$	$\sqrt{2}h$	$\sqrt{2}h$
$\blacksquare$	$\sqrt{2}h$	0	1	1	1
$\blacksquare$	$\sqrt{2}h$	1	0	1	1
$\blacksquare$	$\sqrt{2}h$	1	1	0	1
$\blacksquare$	$\sqrt{2}h$	1	1	1	0

## 2.3 Single symbolic event processing using weighted graphs

Greenhoe (2015b) mapped from the *outcome subspaces* reviewed in the previous section to *weighted graphs* that had order and metric geometries compatible to the outcome subspaces; and expectation and variance operations were defined that correspond roughly to the “center” and “spread” (as defined in graph theory) of a weighted graph. An example follows.

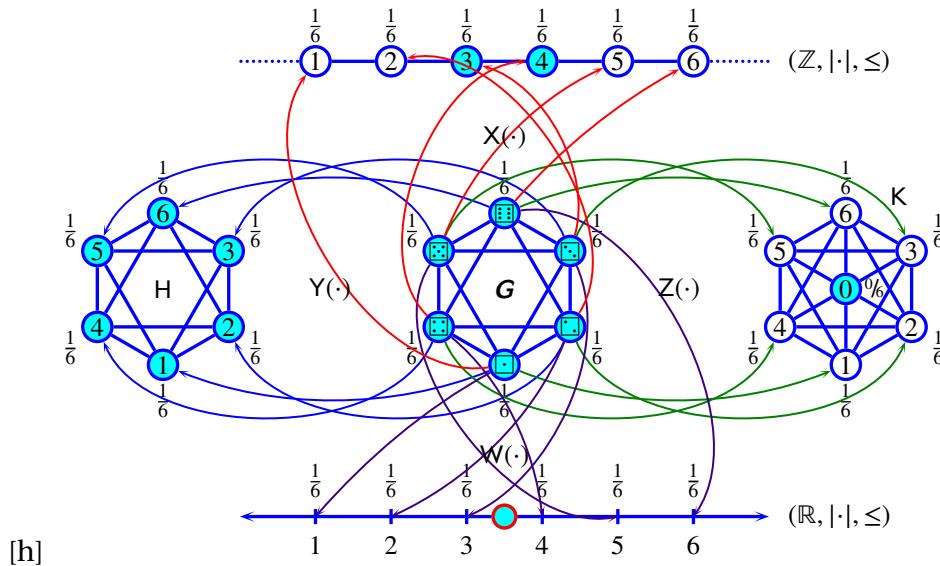


Figure 2: random variable mappings from the *real die outcome subspace* to several *ordered metric spaces* (Example 2.11 page 14)

<sup>32</sup> Greenhoe (2015b)

<sup>33</sup> Greenhoe (2015b)

**Example 2.11** (real die mappings) Let  $\mathbf{G}$  be the *real die outcome subspace*. Let  $W$ ,  $X$ ,  $Y$  and  $Z$  be *random variable* mappings as illustrated in Figure 2 (page 13). Let  $\ddot{E}$  be the *outcome expected value* function, and  $\ddot{\text{Var}}$  the *outcome variance* function, as defined in [Greenhoe \(2015b\)](#). This yields the following statistics:

outcome subspace statistics on real line:	$\ddot{E}(W) = \{3.5\}$	$\ddot{\text{Var}}(W; \ddot{E}) = \frac{35}{12} \approx 2.917$
outcome subspace statistics on integer line:	$\ddot{E}(X) = \{3, 4\}$	$\ddot{\text{Var}}(X; \ddot{E}) = \frac{20}{12} \approx 1.667$
outcome subspace statistics on isomorphic structure:	$\ddot{E}(Y) = \{1, 2, 3, 4, 5, 6\}$	$\ddot{\text{Var}}(Y; \ddot{E}) = 0$
outcome subspace statistics on extended structure:	$\ddot{E}(Z) = \{0\}$	$\ddot{\text{Var}}(Z; \ddot{E}) = 1$

PROOF: See [Greenhoe \(2015b\)](#).

## 2.4 Symbolic sequence processing using linear spaces on $R^n$

The approach using graph theory (Section 2.3) is useful for analysis of a single random variable—for example the *expectation*  $EX$  of  $X$  can be defined simply as the *center* of its *weighted graph*. However, the mapping has limitations with regards to a *sequence* of random variables in performing sequence analysis (using for example *Fourier analysis* or *wavelet analysis*), in performing sequence processing (using for example *FIR filtering* or *IIR filtering*), in making diagnostic measurements (using a post-transform metric space), or in making “optimal” decisions (based on “distance” measurements in a *metric space* or more generally a *distance space*).

Rather than mapping to a *weighted graph*, this paper proposes instead mapping to an *ordered distance linear space*  $\mathbf{Y} \triangleq (\mathbb{R}^n, \leq, d, +, \cdot, \mathbb{R}, \dot{+}, \dot{\times})$ , where  $(\mathbb{R}, +, \dot{\times})$  is a *field*,  $+$  is the vector addition operator on  $\mathbb{R}^n \times \mathbb{R}^n$ , and  $\cdot$  is the scalar-vector multiplication operator on  $\mathbb{R} \times \mathbb{R}^n$ . The *linear space* component of  $\mathbf{Y}$  provides a much more convenient (as compared to the *weighted graph*) framework for sequence analysis and processing. The *ordered set* and *distance space* components of  $\mathbf{Y}$  allow one to preserve the order structure and distance geometry inherent in the underlying stochastic process, which in turn likely provides a less distorted (as compared to the *real line*) framework for analysis, diagnostics, and optimal decision making. This paper demonstrates by several examples the usefulness of the approach.

## 2.5 Motivation for metrics in symbolic sequence processing

Why might we care about *metrics*, or more generally *distance functions*, in symbolic sequence processing? *Metric balls* in a *metric space* induce a *topology*. Topologies are necessary for the concept of *convergence*. Some topologies are also *algebra of sets*:<sup>34</sup> an algebra of sets (or in particular a *sigma-algebra*) is used for the concept of measure. Loosely speaking then, we care about *distance* and *distance spaces* for two reasons:

1. In analysis, *metric spaces* allow us to define the concepts of convergence and limit of a sequence as in  $\sum_{n=0}^{\infty} x_n \triangleq \lim_{N \rightarrow \infty} \sum_{n=0}^{N} x_n$ . That is, without the implicit or explicit definition of convergence and limit, the expression  $\sum_{n=0}^{\infty} x_n$  is meaningless.<sup>35</sup>
2. In signal processing, “optimal” decisions may be made with respect to a *distance space*. For example, a point may be selected (identified as “optimal”) based on it being measured as having the smallest distance to some reference point.

<sup>34</sup>For example on the three element set  $\{x, y, z\}$ , there are a total of 29 topologies; on of these 29, five are algebras of sets.

<sup>35</sup> [Klauder \(2010\) page 4](#)



### 3 Outcome subspace sequences

#### 3.1 Definitions

**Definition 3.1** Let  $\mathbb{D}_1$  and  $\mathbb{D}_2$  be *convex subsets* (Definition 1.18 page 5) of  $\mathbb{Z}$ .

Let  $\mathbb{D} \triangleq (\bigwedge \mathbb{D}_1 - \bigvee \mathbb{D}_2 - 1 : \bigvee \mathbb{D}_1 - \bigwedge \mathbb{D}_2 + 1)$ . Let  $(x_n)_{\mathbb{D}_1}$  and  $(y_n)_{\mathbb{D}_2}$  be *sequences* over an *outcome subspace*  $(\Omega, \leq, \dot{d}, \dot{P})$ . The **outcome subspace sequence metric**  $p((x_n), (y_n))$  is defined as

$$p((x_n), (y_n)) \triangleq \sum_{n \in \mathbb{D}} f(n) \quad \text{where} \quad f(n) \triangleq \begin{cases} \dot{d}(x_n, y_n) & \text{if } n \in \mathbb{D}_1 \text{ and } n \in \mathbb{D}_2 \\ 1 & \text{if } n \in \mathbb{D}_1 \text{ but } n \notin \mathbb{D}_2 \\ 1 & \text{if } n \notin \mathbb{D}_1 \text{ but } n \in \mathbb{D}_2 \\ 0 & \text{otherwise} \end{cases} \quad \forall n \in \mathbb{D}$$

**Proposition 3.2** Let  $(x_n)_{\mathbb{D}_1}$  and  $(y_n)_{\mathbb{D}_2}$  be *SEQUENCES* over an *OUTCOME SUBSPACE*  $(\Omega, \leq, \dot{d}, \dot{P})$ .

Let  $p$  be the *OUTCOME SUBSPACE SEQUENCE METRIC*.

$$\mathbb{D}_1 \cap \mathbb{D}_2 \neq \emptyset \implies p \text{ is a METRIC}$$

PROOF: This follows from the *Fréchet product metric* (Proposition B.4 page 46). In particular,  $p$  is a sum of metrics that include the metrics  $\dot{d}(x_n, y_n)$  and the *discrete metric* (Definition B.2 page 46).  $\square$

In standard signal processing, the *autocorrelation* of a *sequence*  $(x_n)_{n \in \mathbb{D}}$  is another *sequence*  $(y_n)_{n \in \mathbb{D}}$  defined as  $y_n \triangleq \sum_{m \in \mathbb{Z}} x_m x_{m-n}$ . However, this definition requires that the sequence  $(x_n)_{n \in \mathbb{D}}$  be constructed over a *field*. In an *outcome subspace sequence*, we in general do not have a *field*; for example, in a *die outcome subspace*, the expressions  $\square + \square$  and  $\square \times \square$  are undefined. This paper offers an alternative definition (next) for *autocorrelation* that uses the *distance*  $\dot{d}$  and that does not require a *field*.

**Definition 3.3** Let  $(x_n)_{n \in \mathbb{D}}$  and  $(y_n)_{n \in \mathbb{D}}$  be *sequences* over the *outcome subspace*  $(\Omega, \leq, \dot{d}, \dot{P})$ .

Let  $p((x_n), (y_n))$  be the **outcome subspace sequence metric** (Definition 3.1 page 15).

The **cross-correlation**  $R_{xy}(n)$  of  $(x_n)$  and  $(y_n)$  and the **autocorrelation**  $R_{xx}(n)$  of  $(x_n)$  are defined as

$$R_{xy}(n) \triangleq - \sum_{m \in \mathbb{Z}} p((x_{m-n}), (y_m)) \quad (\text{cross-correlation } R_{xy}(n) \text{ of } (x_n) \text{ and } (y_n))$$

$$R_{xx}(n) \triangleq - \sum_{m \in \mathbb{Z}} p((x_{m-n}), (x_m)) \quad (\text{autocorrelation } R_{xx}(n) \text{ of } (x_n))$$

Moreover, the  **$M$ -offset autocorrelation** of  $(x_n)$  and  $(y_n)$  is here defined as  $R_{xx}(n) + M$  (Definition 1.40 page 9).

#### 3.2 Examples of symbolic sequence statistics

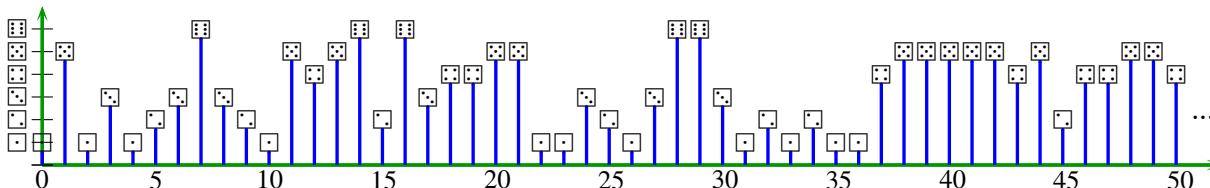
**Example 3.4** (fair die sequence) Consider the pseudo-uniformly distributed *fair die* (Definition 2.5 page 12) sequence generated by the C code<sup>36</sup>

```
1 #include<stdlib.h>
2 ...
3 srand(0x5EED);
4 for(n=0; n<N; n++) {x[n] = 'A' + rand() % 6;}
```

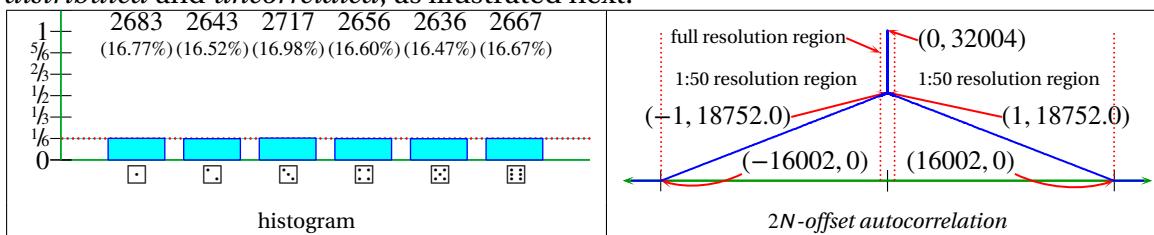
where 'A' represents  $\square$ , 'B' represents  $\square$ ,  $\vdots$ , 'F' represents  $\square$ .

The resulting sequence is partially displayed here:

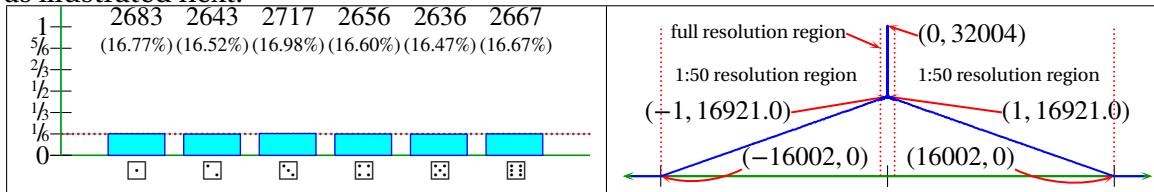




This sequence constrained to a length of  $N = 2667 \times 6 = 16002$  elements is approximately *uniformly distributed* and *uncorrelated*, as illustrated next:



**Example 3.5** (real die sequence) Consider the pseudo-uniformly distributed *real die* (Definition 2.7 page 12) sequence generated as in Example 3.4 (page 15), but with the real die metric rather than the fair die metric. This change will not affect the distribution of the sequence, but it does affect the autocorrelation, as illustrated next:



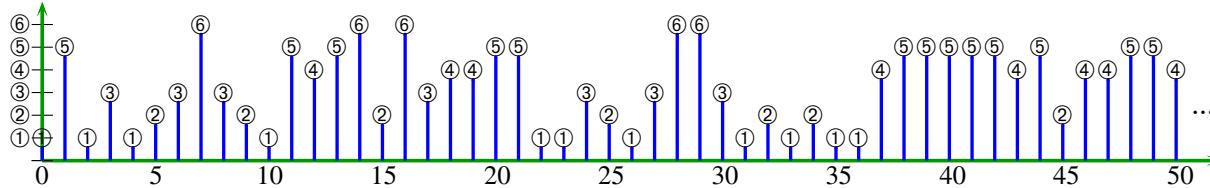
**Example 3.6** (spinner sequence) Consider the pseudo-uniformly distributed *spinner* (Definition 2.8 page 13) sequence generated by the C code<sup>37</sup>

```

1 #include<stdlib.h>
2 ...
3 srand(0x5EED);
4 for(n=0; n<N; n++) {x[n] = 'A' + rand() %6;}
```

where 'A' represents ① ,  
 'B' represents ② ,  
 :  
 'F' represents ⑥ .

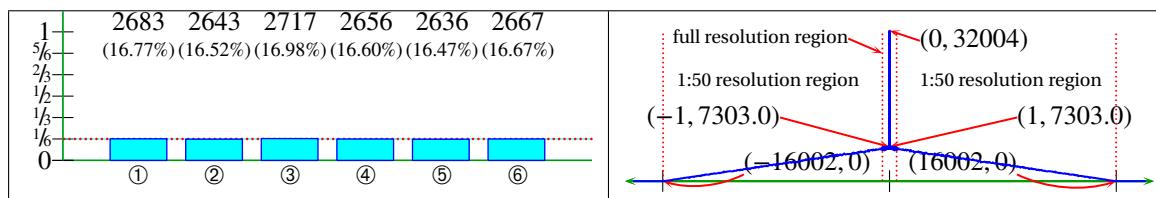
The resulting sequence is partially displayed here:



This sequence is in essence identical to the fair die sequence (Example 3.4 page 15) and real die sequence (Example 3.5 page 16) and thus yields what is essentially an identical histogram. But because the metric is different, the autocorrelation is also different. In particular, because the nodes of the spinner metric are on average farther apart with respect to the spinner metric, the sequence is less correlated (with respect to the metric), as illustrated next:

<sup>36</sup>For a more complete source code listing, see Section D.2 (page 55)

<sup>37</sup>For a more complete source code listing, see Section D.4 (page 68)



**Example 3.7** (weighted real die sequence) Consider the non-uniformly distributed *weighted real die* (Definition 2.6 page 12) sequence with

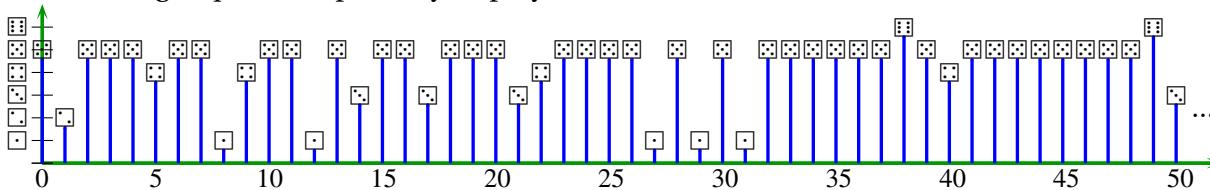
$P(\square) = 0.75$  and  $P(\square) = P(\square) = P(\square) = P(\square) = P(\square) = 0.05$ , generated by the C code<sup>38</sup>

```

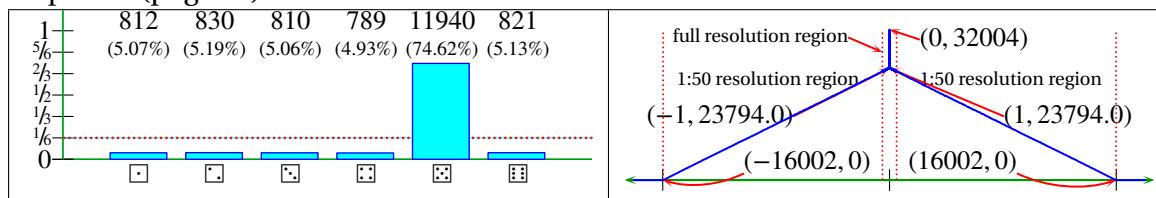
1 strand(0x5EED);
2 for(n=0; n<N; n++) { u=rand()%100;
3   if (u< 5) x[n]='A'; /* 00-04 */
4   else if(u<10) x[n]='B'; /* 05-09 */
5   else if(u<15) x[n]='C'; /* 10-14 */
6   else if(u<20) x[n]='D'; /* 15-19 */
7   else if(u<95) x[n]='E'; /* 20-94 */
8   else x[n]='F'; /* 95-99 */
  }
```

where 'A' represents  $\square$ , 'B' represents  $\square$ , 'C' represents  $\square$ , 'D' represents  $\square$ , 'E' represents  $\square$ , and 'F' represents  $\square$ .

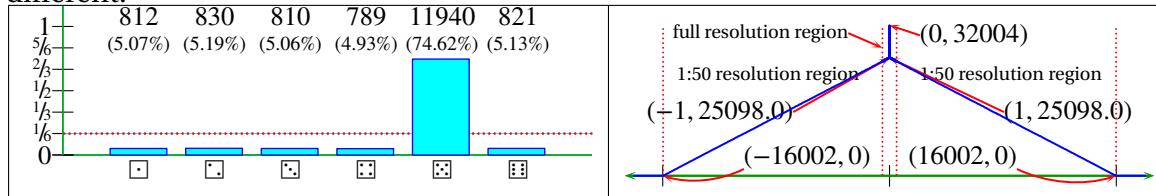
The resulting sequence is partially displayed here:



Of course the resulting histogram, as illustrated below on the left, reflects the non-uniform distribution. Also note, as illustrated below on the right, that the weighted sequence is much more correlated (as defined by Definition 3.3 page 15) as compared to the uniformly distributed *real die* sequence of Example 3.5 (page 16).



**Example 3.8** (weighted die sequence) Consider the non-uniformly distributed *weighted die* (Definition 2.4 page 12) sequence generated as in Example 3.7. Of course the resulting histogram is identical to that of Example 3.7, but because the distance function is different, the autocorrelation sequence is also different.



<sup>38</sup>For a more complete source code listing, see Section D.3 (page 61)

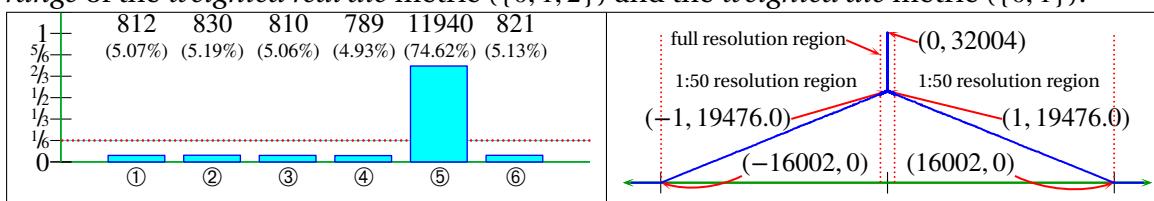
**Example 3.9** (weighted spinner sequence) Consider the non-uniformly distributed die sequence with

$P(5) = 0.75$  and  $P(1) = P(2) = P(3) = P(4) = P(6) = 0.05$ , generated by the C code<sup>39</sup>

```
1 srand(0x5EED);
2 for(n=0; n<N; n++) { u=rand()%100;
3     if (u< 5) x[n]='A'; /* 00-04 */
4     else if(u<10) x[n]='B'; /* 05-09 */
5     else if(u<15) x[n]='C'; /* 10-14 */
6     else if(u<20) x[n]='D'; /* 15-19 */
7     else if(u<95) x[n]='E'; /* 20-94 */
8     else x[n]='F'; /* 95-99 */ }
```

where 'A' represents ① ,  
 'B' represents ② ,  
 'C' represents ③ ,  
 'D' represents ④ ,  
 'E' represents ⑤ , and  
 'F' represents ⑥ .

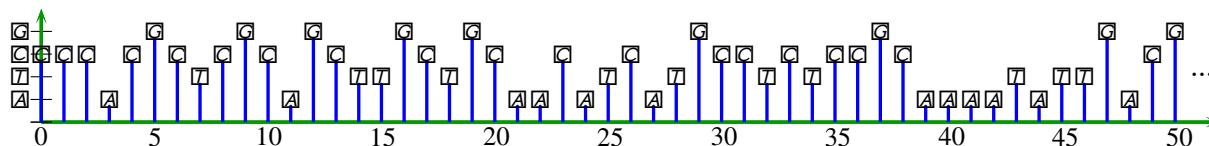
The resulting sequence and histogram is in essence the same as in the *weighted die sequence* example (Example 3.7 page 17). But note, as illustrated below on the right, that the *weighted spinner sequence* of this example is significantly less correlated than the *weighted die sequence* of Example 3.7 (page 17), presumably due to the larger *range* (Definition 1.14 page 4) of the spinner metric ( $\{0, 1, 2, 3\}$ ) as compared to the *range* of the *weighted real die* metric ( $\{0, 1, 2\}$ ) and the *weighted die* metric ( $\{0, 1\}$ ).



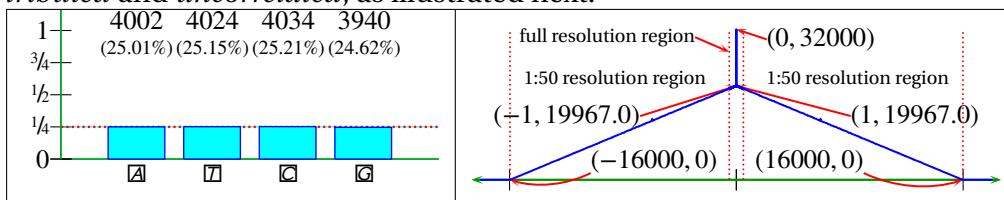
**Example 3.10** (Random DNA sequence) Consider the pseudo-uniformly distributed DNA sequence generated by the C code<sup>40</sup>

```
1 srand(0x5EED);
2 for(n=0; n<N; n++) { r=rand()%4;
3     switch(r) { case 0: x[n]='A'; break;
4                 case 1: x[n]='T'; break;
5                 case 2: x[n]='C'; break;
6                 case 3: x[n]='G'; break; }
```

where 'A' represents  ,  
           'T' represents  ,  
           'C' represents  , and  
           'G' represents  .



This sequence constrained to a length of  $4000 \times 4 = 16000$  elements is approximately *uniformly distributed* and *uncorrelated*, as illustrated next:



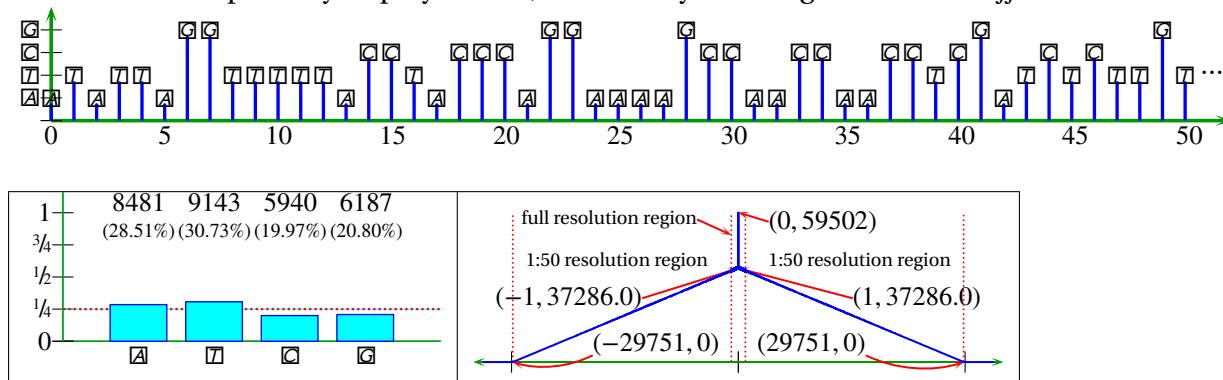
**Example 3.11** (SARS coronavirus DNA sequence) Consider the genome sequence (DNA sequence) for the SARS coronavirus with *GenBank* accession number NC\_004718.3.<sup>41</sup> This sequence is of length

<sup>39</sup>For a more complete source code listing, see Section D.4 (page 68).

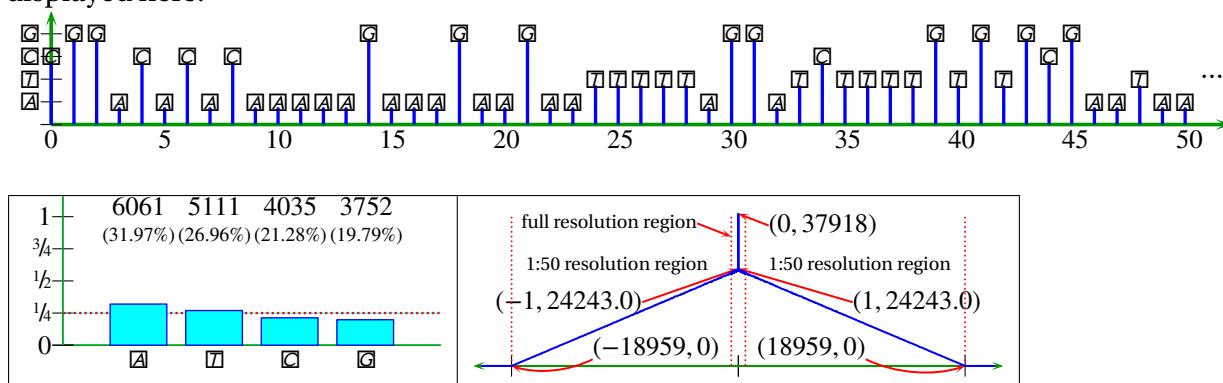
<sup>40</sup>For a more complete source code listing, see Section D.5 (page 75).

<sup>41</sup> GenBank-NC\_004718.3 (2011)

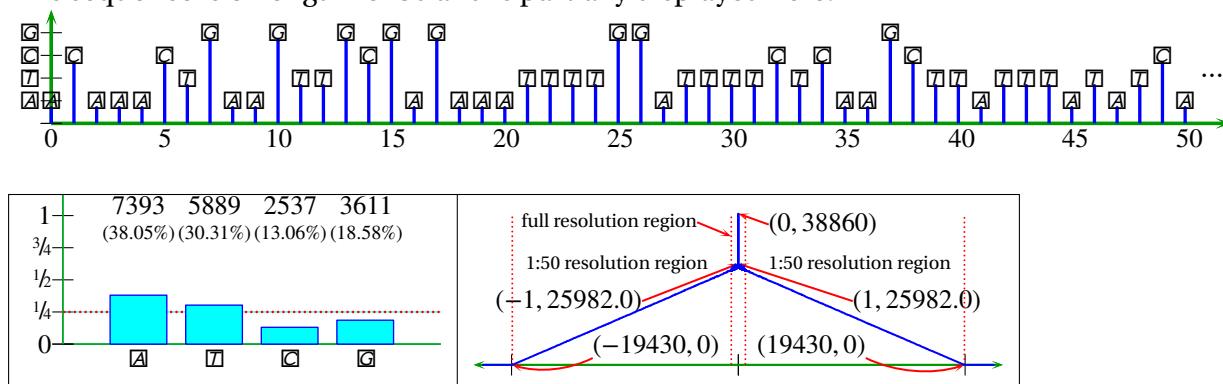
$N = 29751$  and is partially displayed here, followed by its histogram and  $2N$ -offset autocorrelation plots.



**Example 3.12** (Ebola virus DNA sequence) Consider the genome sequence (DNA sequence) for the Ebola virus with GenBank accession AF086833.2.<sup>42</sup> This sequence is of length 18959 and is partially displayed here:



**Example 3.13** (Bacterium DNA sequence) Consider the genome sequence (DNA sequence) for the bacterium *Melissococcus plutonius* strain 49.3 plasmid pMP19 with GenBank accession NZ\_CM003360.1.<sup>43</sup> This sequence is of length 19430 and is partially displayed here:



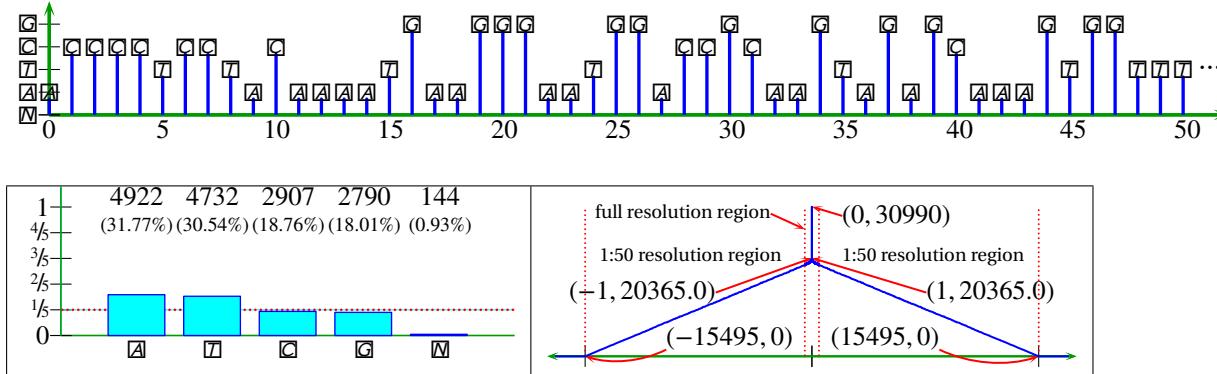
**Example 3.14** (Papaya DNA sequence)<sup>44</sup> Consider the genome sequence segment for the fruit *carica papaya* with GenBank accession DS982815.1. This sequence is not a complete genome, rather it is a

<sup>42</sup> GenBank-AF086833.2 (2013)

<sup>43</sup> GenBank-NZ\_CM003360.1 (2015)

<sup>44</sup> GenBank-DS982815.1 (2015)

“genomic scaffold” (Definition 2.10 page 13). As such, there are some elements for which the content is not known. For these locations, the symbol `N` is used. In this particular sequence, there are 144 `N` symbols. This sequence is of length 15495 and is partially displayed here:



### 3.3 Extending to distance linear spaces

### 3.3.1 Motivation

Section 2.2 (page 11) reviewed how a stochastic process could be defined as an *outcome subspace* with *order* and *metric* structures. Example 2.11 (page 14) reviewed an example of a *real die outcome subspace* that was mapped through 4 different *random variables* to 4 different *weighted graphs*. Two of these random variables ( $Y$  and  $Z$ ) mapped to structures (*weighted graphs*) that are very similar to the *real die* with respect to order and metric geometry. Two other random variables ( $W$  and  $X$ ) mapped to structures (the *real line* and the *integer line*) that are very dissimilar. The implication of this example is that if we want statistics that closely model the underlying stochastic process, then we should map to a structure that has an order structure and distance geometry similar to that of the underlying stochastic process, and not simply the one that is the most convenient. Ideally, we would like to map to a structure that is *isomorphic* (Definition 1.24 page 6) and *isometric* (Definition A.6 page 45) to the structure of the stochastic process.

However, for sequence processing using very basic methods such as FIR filtering, Fourier analysis, or wavelet analysis, we would very much like to map into the *real line*  $\mathbb{R}^1$  or possibly some higher dimensional space  $\mathbb{R}^n$ . Because the real line is often very dissimilar to the stochastic process, we are motivated to find structures in  $\mathbb{R}^n$  that *are* similar. And that is what this section presents—mapping from a stochastic process  $(\Omega, \leq, d, P)$  into an *ordered distance linear space*  $(\mathbb{R}^n, \leq, d, +, \cdot, \mathbb{R}, \dot{+}, \dot{\times})$  in which  $\mathbb{R}^n$  is an extension of  $\Omega$  and  $d$  is an extension of  $\dot{d}$ .

Thus, for sequence processing on an *outcome subspace*  $(\Omega, \leq, \dot{d}, \dot{\mathbb{P}})$ , we would like to define a *random variable*  $X$  and an *ordered distance linear space*  $(\mathbb{R}^n, \leq, d, +, \cdot, \mathbb{R}, \dot{+}, \dot{\times})$  that satisfy the following constraints:

- (1) The random variable maps the elements of  $\Omega$  into  $\mathbb{R}^n$  and
  - (2) the order relation  $\leq$  is an *extension* to  $\mathbb{R}^n$  of the order relation  $\leq$  on  $\Omega$  and
  - (3) the *distance* function  $d$  is an *extension* to  $\mathbb{R}^n$  of the distance function  $\hat{d}$  on  $\Omega$ .

### 3.3.2 Some random variables

In this section, we first define some *random variables* (Definition 2.3 page 12) that are used later in this paper.

**Definition 3.15** The **traditional die random variable**  $X$  maps from the set

$\{\square, \square, \square, \square, \square, \square\}$  into the set  $\mathbb{R}^1$  and is defined as<sup>45</sup>

$$X(\square) \triangleq 1, X(\square) \triangleq 2, X(\square) \triangleq 3, X(\square) \triangleq 4, X(\square) \triangleq 5, \text{ and } X(\square) \triangleq 6.$$

**Definition 3.16** The **PAM die random variable**  $X$  maps from the set  $\{\square, \square, \square, \square, \square, \square\}$  into the set  $\mathbb{R}^1$  and is defined as<sup>46</sup>

$$X(\square) \triangleq -2.5, X(\square) \triangleq -1.5, X(\square) \triangleq -0.5, X(\square) \triangleq +0.5, X(\square) \triangleq +1.5, \text{ and } X(\square) \triangleq +2.5.$$

**Definition 3.17** The **QPSK die random variable**  $X$  maps from the set  $\{\square, \square, \square, \square, \square, \square\}$  into the set  $\mathbb{C}^1$  and is defined as<sup>47</sup>

$$\begin{aligned} X(\square) &\triangleq \exp(30 \times \frac{\pi}{180}i), & X(\square) &\triangleq \exp(90 \times \frac{\pi}{180}i), & X(\square) &\triangleq \exp(150 \times \frac{\pi}{180}i), \\ X(\square) &\triangleq \exp(210 \times \frac{\pi}{180}i), & X(\square) &\triangleq \exp(270 \times \frac{\pi}{180}i), & X(\square) &\triangleq \exp(330 \times \frac{\pi}{180}i). \end{aligned}$$

**Definition 3.18** The  **$\mathbb{R}^3$  die random variable**  $X$  maps from the set  $\{\square, \square, \square, \square, \square, \square\}$  into the set  $\mathbb{R}^3$  and is defined as

$$\begin{aligned} X(\square) &\triangleq (1, 0, 0), & X(\square) &\triangleq (0, 1, 0), & X(\square) &\triangleq (0, 0, 1), \\ X(\square) &\triangleq (0, 0, -1), & X(\square) &\triangleq (0, -1, 0), & X(\square) &\triangleq (-1, 0, 0). \end{aligned}$$

**Definition 3.19** The  **$\mathbb{R}^6$  die random variable**  $X$  maps from the set  $\{\square, \square, \square, \square, \square, \square\}$  into the set  $\mathbb{R}^6$  and is defined as

$$\begin{aligned} X(\square) &\triangleq (1, 0, 0, 0, 0, 0), & X(\square) &\triangleq (0, 1, 0, 0, 0, 0), & X(\square) &\triangleq (0, 0, 1, 0, 0, 0), \\ X(\square) &\triangleq (0, 0, 0, 1, 0, 0), & X(\square) &\triangleq (0, 0, 0, 0, 1, 0), & X(\square) &\triangleq (0, 0, 0, 0, 0, 1). \end{aligned}$$

**Definition 3.20** The  **$\mathbb{R}^1$  spinner random variable**  $X$  maps from the set  $\{\circledcirc, \circledcirc, \circledcirc, \circledcirc, \circledcirc, \circledcirc\}$  into the set  $\mathbb{R}^1$  and is defined as<sup>48</sup>

$$X(\circledcirc) \triangleq 1, X(\circledcirc) \triangleq 2, X(\circledcirc) \triangleq 3, X(\circledcirc) \triangleq 4, X(\circledcirc) \triangleq 5, \text{ and } X(\circledcirc) \triangleq 6.$$

**Definition 3.21** The **QPSK spinner random variable**  $X$  maps from the set  $\{\circledcirc, \circledcirc, \circledcirc, \circledcirc, \circledcirc, \circledcirc\}$  into the set  $\mathbb{C}^1$  and is defined as<sup>49</sup>

$$\begin{aligned} X(\circledcirc) &\triangleq \exp(30 \times \frac{\pi}{180}i), & X(\circledcirc) &\triangleq \exp(90 \times \frac{\pi}{180}i), & X(\circledcirc) &\triangleq \exp(150 \times \frac{\pi}{180}i), \\ X(\circledcirc) &\triangleq \exp(210 \times \frac{\pi}{180}i), & X(\circledcirc) &\triangleq \exp(270 \times \frac{\pi}{180}i), & X(\circledcirc) &\triangleq \exp(330 \times \frac{\pi}{180}i). \end{aligned}$$

**Definition 3.22** <sup>49</sup> The **PAM DNA random variable**  $X$  maps from the set  $\{\square, \square, \square, \square\}$  into the set  $\mathbb{R}^1$  and is defined as<sup>50</sup>

$$X(\square) \triangleq -1.5, X(\square) \triangleq -0.5, X(\square) \triangleq +0.5, X(\square) \triangleq +1.5.$$

<sup>46</sup>PAM is an acronym for *pulse amplitude modulation* and is a standard technique in the field of digital communications.

<sup>47</sup>QPSK is an acronym for *quadrature phase shift keying* and is a standard technique in the field of digital communications.

<sup>49</sup> Galleani and Garello (2010) page 772

**Definition 3.23** <sup>51</sup> The **QPSK DNA random variable**  $X$  maps from the set  $\{\square, \square, \square, \square\}$  into the set  $\mathbb{C}^1$  and is defined as

$$\begin{aligned} X(\square) &\triangleq \exp\left(45 \times \frac{\pi}{180}i\right), & X(\square) &\triangleq \exp\left(135 \times \frac{\pi}{180}i\right), \\ X(\square) &\triangleq \exp\left(225 \times \frac{\pi}{180}i\right), & X(\square) &\triangleq \exp\left(315 \times \frac{\pi}{180}i\right). \end{aligned}$$

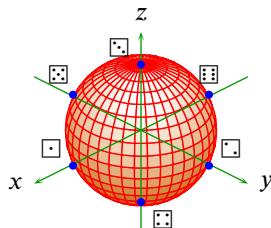
**Definition 3.24** The  $\mathbb{R}^4$  **DNA random variable**  $X$  maps from the set  $\{\square, \square, \square, \square\}$  into the set  $\mathbb{R}^4$  and is defined as<sup>52</sup>

$$X(\square) \triangleq (1, 0, 0, 0), \quad X(\square) \triangleq (0, 1, 0, 0), \quad X(\square) \triangleq (0, 0, 1, 0), \quad X(\square) \triangleq (0, 0, 0, 1)$$

### 3.3.3 Some ordered distance linear spaces

**Definition 3.25** The structure  $(\mathbb{R}^1, \leq, d)$  is the  $\mathbb{R}^1$  **die distance linear space** if  $\leq$  is the *standard ordering relation* on  $\mathbb{R}$ , and  $d(x, y) \triangleq |x - y|$  (the *Euclidean metric* on  $\mathbb{R}$ , Definition B.3 page 46).

**Definition 3.26** The structure  $(\mathbb{R}^3, \leq, d)$  is the  $\mathbb{R}^3$  **die distance linear space** if  $\leq = \emptyset$ , and  $d$  is the 2-scaled *Lagrange arc distance*  $d$  defined as follows:  $d(p, q) \triangleq 2p(p, q)$  where  $p$  is the *Lagrange arc distance* (Definition C.1 page 47).



$d(x, y)$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
$\square$	0	1	1	1	1	2
$\square$	1	0	1	1	2	1
$\square$	1	1	0	2	1	1
$\square$	1	1	2	0	1	1
$\square$	1	2	1	1	0	1
$\square$	2	1	1	1	1	0

Used together with the  $\mathbb{R}^3$  *die random variable*  $X$  (Definition 3.18 page 21), the distance  $d$  in the  $\mathbb{R}^3$  *die distance linear space* (Definition 3.26) is an extension of  $d$  in the *real die outcome subspace*  $G \triangleq (\{\square, \square, \square, \square, \square, \square\}, \leq, d, \dot{P})$  (Definition 2.7 page 12). We can also say that  $X$  is an *isometry* (Definition A.6 page 45) and that the two structures are *isometric*. For example,

$$\begin{aligned} d[X(\square), X(\square)] &= d[(1, 0, 0), (0, 1, 0)] = 1 = d(\square, \square) \text{ and} \\ d[X(\square), X(\square)] &= d[(1, 0, 0), (-1, 0, 0)] = 2 = d(\square, \square). \end{aligned}$$

As for order, the mapping  $X$  is also *order preserving* (Definition 1.23 page 6), but trivially, because the *real die outcome subspace* is *unordered* (Definition 1.16 page 5). But if we still honor the standard ordering on each dimension  $\mathbb{R}$  in  $\mathbb{R}^3$ , then the two structures are *not isomorphic* (Definition 1.24 page 6) because<sup>53</sup> the inverse  $X^{-1}$  is *not order preserving* (Theorem 1.25 page 6)—for example,  $X(\square) = (0, 0, -1) \leq (0, 0, 1) = X(\square)$ , but  $\square$  and  $\square$  are *incomparable* (Definition 1.16 page 5) in  $G$ .

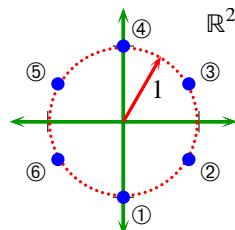
**Definition 3.27** The structure  $(\mathbb{R}^2, \leq, d)$  is the  $\mathbb{R}^2$  **spinner distance linear space** if  $\leq = \emptyset$ , and  $d$  is the 3-scaled *Lagrange arc distance*  $d$  defined as follows:  $d(p, q) \triangleq 3p(p, q)$  where  $p$  is the *Lagrange arc distance* (Definition C.1 page 47).

<sup>51</sup> [Galleani and Garello \(2010\)](#) page 772

<sup>51</sup>QPSK is an acronym for *quadrature phase shift keying* and is a standard technique in the field of digital communications.

<sup>52</sup>This type of mapping has previously been used by [Voss \(1992\)](#) in calculating the *Voss Spectrum*, (a kind of Fourier analysis) of DNA sequences. See also [Galleani and Garello \(2010\)](#) page 772.

<sup>53</sup>Note that while  $X^{-1}$  (Definition 1.9 page 4) does not exist as a *function*, it does exist as a *relation*.



$d(x, y)$	①	②	③	④	⑤	⑥
①	0	1	2	3	2	1
②	1	0	1	2	3	2
③	2	1	0	1	2	3
④	3	2	1	0	1	2
⑤	2	3	2	1	0	1
⑥	1	2	3	2	1	0

Used together with the *QPSK spinner random variable*  $X$  (Definition 3.21 page 21), the distance  $d$  in the  $\mathbb{R}^2$  *spinner distance linear space* (Definition 3.27 page 22) is an extension of  $\dot{d}$  in the *spinner outcome subspace*  $G \triangleq (\{\textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4}, \textcircled{5}, \textcircled{6}\}, \dot{d}, \leq, \dot{P})$  (Definition 2.8 page 13). We can again say that  $X$  is an *isometry* and that the two structures are *isometric*. For example,

$$\begin{aligned} d[X(\textcircled{1}), X(\textcircled{2})] &= d[(0, -1), (\sqrt{3}/2, -1/2)] = 1 = \dot{d}(\textcircled{1}, \textcircled{2}) \quad \text{and} \\ d[X(\textcircled{1}), X(\textcircled{3})] &= d[(0, -1), (\sqrt{3}/2, +1/2)] = 2 = \dot{d}(\textcircled{1}, \textcircled{3}) \quad \text{and} \\ d[X(\textcircled{1}), X(\textcircled{4})] &= d[(0, -1), (0, 1)] = 3 = \dot{d}(\textcircled{1}, \textcircled{4}) . \end{aligned}$$

The mapping  $X$  is again trivially *order preserving*. And if we again honor the standard ordering on each dimension  $\mathbb{R}$  in  $\mathbb{R}^2$ , then the two structures are *not isomorphic* (Definition 1.24 page 6) because the inverse  $X^{-1}$  is *not order preserving*—for example,  $X(\textcircled{1}) = (0, -1) \leq (0, 1) = X(\textcircled{4})$ , but  $\textcircled{1}$  and  $\textcircled{4}$  are *incomparable* in  $G$ .

**Definition 3.28** The structure  $(\mathbb{R}^6, \leq, d)$  is the  $\mathbb{R}^6$  **die distance linear space** if  $\leq = \emptyset$ , and  $d$  is defined as  $d(p, q) \triangleq \sqrt{2}p(p, q)$ , where  $p$  is the *Euclidean metric* on  $\mathbb{R}^6$  (Definition B.3 page 46).

Used together with the  $\mathbb{R}^6$  *die random variable*  $X$  (Definition 3.19 page 21), the distance  $d$  in the  $\mathbb{R}^6$  *fair die distance linear space* (Definition 3.28 page 23) is an extension of  $\dot{d}$  in the *fair die outcome subspace*  $G \triangleq (\{\square, \square, \square, \square, \square, \square\}, \dot{d}, \leq, \dot{P})$  (Definition 2.7 page 12). We can again say that  $X$  is an *isometry* and that the two structures are *isometric*. For example,

$$\begin{aligned} d[X(\square), X(\square)] &= d[(1, 0, 0, 0, 0, 0), (0, 1, 0, 0, 0, 0)] = 1 = \dot{d}(\square, \square) \quad \text{and} \\ d[X(\square), X(\square)] &= d[(1, 0, 0, 0, 0, 0), (0, 0, 1, 0, 0, 0)] = 1 = \dot{d}(\square, \square) \quad \text{and} \\ d[X(\square), X(\square)] &= d[(1, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 1)] = 1 = \dot{d}(\square, \square) . \end{aligned}$$

The mapping  $X$  is again trivially *order preserving*, and the inverse  $X^{-1}$  is trivially *order preserving* as well. And so unlike the  $\mathbb{R}^3$  *die distance linear space* (Definition 3.26) and the  $\mathbb{R}^2$  *spinner distance linear space* (Definition 3.27), this pair of structures is *isomorphic*.

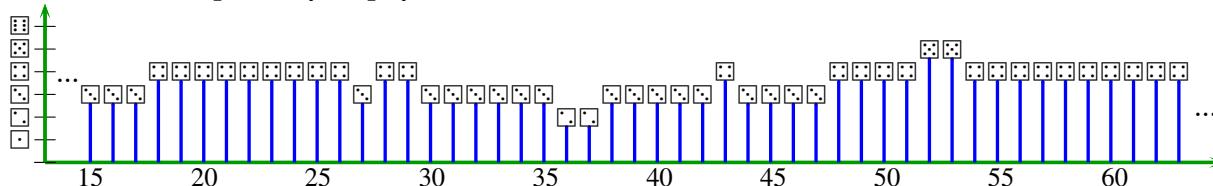
## 4 Symbolic sequence processing results

### 4.1 Low pass filtering/Smoothing

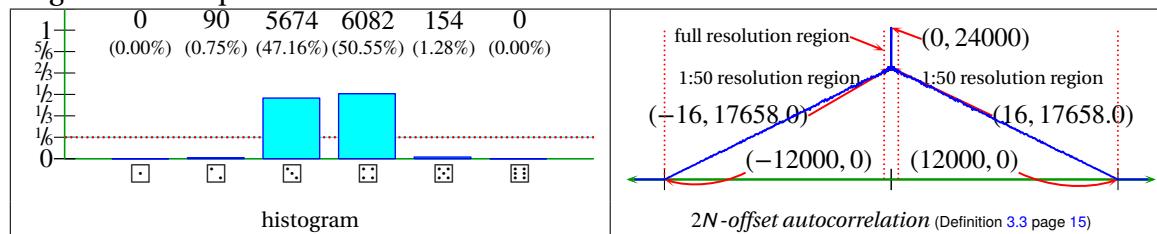
**Example 4.1** (low pass filtering of real die sequence)

- Consider the pseudo-uniformly distributed die sequence presented in Example 3.5 (page 16). Suppose we want to *filter* this sequence with a *low pass sequence* in order to “smooth out” the sequence. But to perform the actual filtering, note that the die sequence must first be mapped into a *linear space*  $\mathbb{R}^N$ .

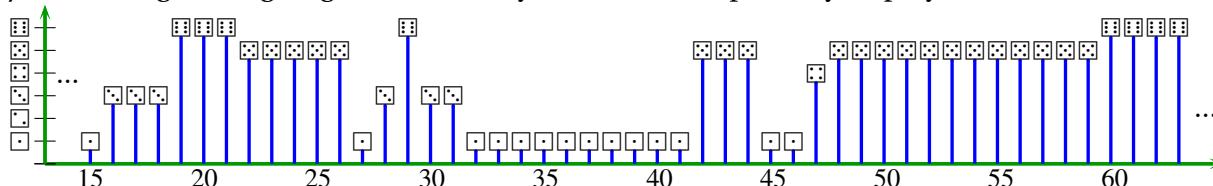
- (2) Suppose we first use the *traditional die random variable* (Definition 3.15 page 21) to map the die sequence into  $\mathbb{R}^1$ . *Filtering* (Definition 1.41 page 9) this  $\mathbb{R}$ -valued sequence using the *length 16 rectangular low pass sequence* (Example 1.44 page 9) in the  $\mathbb{R}^1$  *die distance linear space* (Definition 3.25 page 22) and then mapping the result back to a *die sequence* using the *Euclidean metric* (Definition B.3 page 46), produces the result partially displayed here:



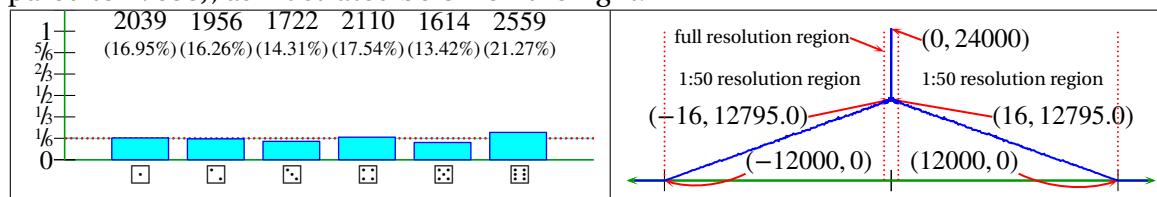
Note that the die sequence has indeed been smoothed out, but its uniform distribution has been destroyed—almost all of its values are around the “expected value” 3.5, as illustrated below on the left. Of course such filtering also introduces correlation, giving the *autocorrelation* sequence a slightly wider center lobe as illustrated below on the right. Both diagrams are calculated over a length 12000 sequence.



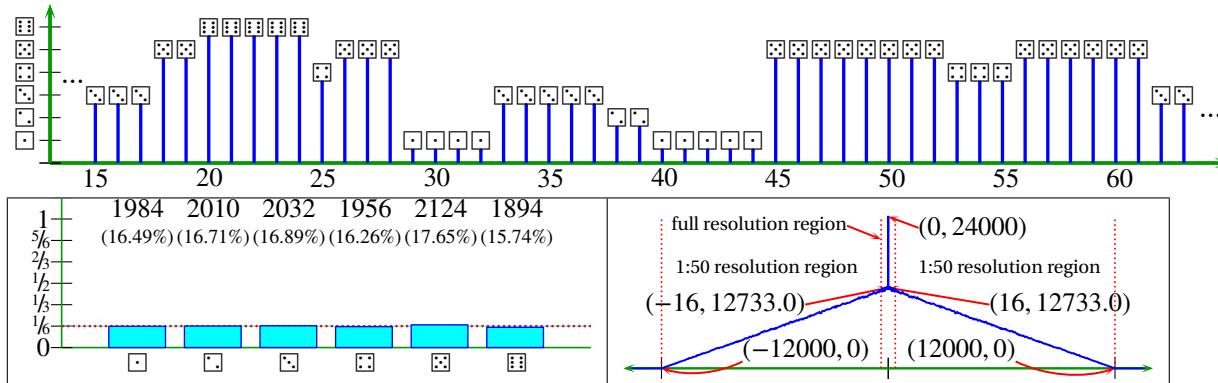
- (3) Alternatively, suppose we next try using the  $\mathbb{R}^3$  *die random variable* (Definition 3.18 page 21) to map the die sequence into  $\mathbb{R}^3$ . *Filtering* this new sequence using the *length 16 rectangular low pass sequence* in the  $\mathbb{R}^3$  *distance linear space* (Definition 3.26 page 22) and then mapping back to a *die sequence* using the *Lagrange arc distance* yields the result partially displayed here:



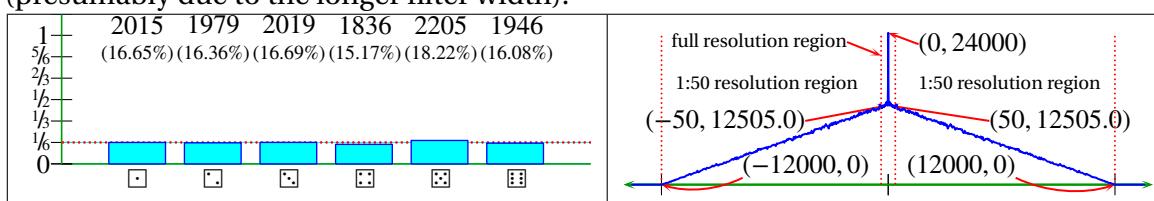
Note that the *die sequence* does appear to be “smoothed out”, but this time the distribution is much more uniform, as illustrated below on the left; and is slightly less correlated (12795 compared to 17658), as illustrated below on the right.



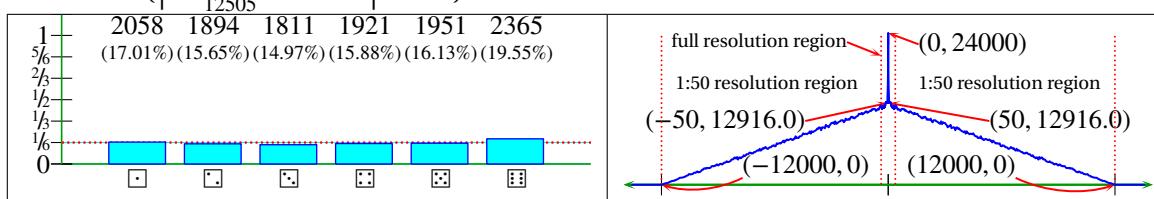
- (4) Using a *length 16 Hanning low pass sequence* (Definition 1.45 page 9) rather than the *length 16 rectangular low pass sequence* as in item (3) results in a distribution that is more uniform and in a sequence that is very slightly less correlated:



- (5) Using a *length 50 Hanning low pass sequence* (Example 1.46 page 10) rather than the *length 16 Hanning low pass sequence* as in item (4) results in about the same uniformity of distribution, about 1.8% lower side lobes in the autocorrelation sequence ( $\frac{12733-12505}{12733} \times 100 \approx 1.8$ ), but a wider main lobe (presumably due to the longer filter width):



- (6) Using a *length 50 rectangular low pass sequence* rather than the *length 50 Hanning low pass sequence* as in item (5) results in a distribution that is a little less uniform and about 3.3% more correlated ( $|\frac{12505-12916}{12505}| \times 100| \approx 3.3$ ):



- (7) Replacing the *Lagrange arc distance* by the *Euclidean metric* in this example has very little effect. More details follow:

- Using the *Euclidean metric* in  $\mathbb{R}^3$  rather than the *Lagrange arc distance* in item (3) yields sequences that are **identical**.<sup>54</sup>
- Using the *Euclidean metric* in item (4) rather than the *Lagrange arc distance* yields sequences that **differ** at 6 locations out of  $N + M + M - 1 = 12000 + 16 + 16 - 1 = 12031$  locations (differ at approximately 0.05% of the locations).<sup>55</sup>

n	Euclidean	Lagrange
1296	◻	◻
1630	◻	◻
2587	◻	◻

n	Euclidean	Lagrange
2589	◻	◻
6242	◻	◻
11888	◻	◻

- Using the *Euclidean metric* in  $\mathbb{R}^3$  rather than the *Lagrange arc distance* as in item (5) (length 50 Hanning filter) yields sequences that **differ** at 109 locations out of  $N + M + M - 1 = 12000 + 50 + 50 - 1 = 12099$  locations (differ at approximately 0.9% of the locations).<sup>56</sup>

<sup>54</sup> See experiment log file "rdie\_lp\_12000m16.xls" generated by the program "ssp.exe".

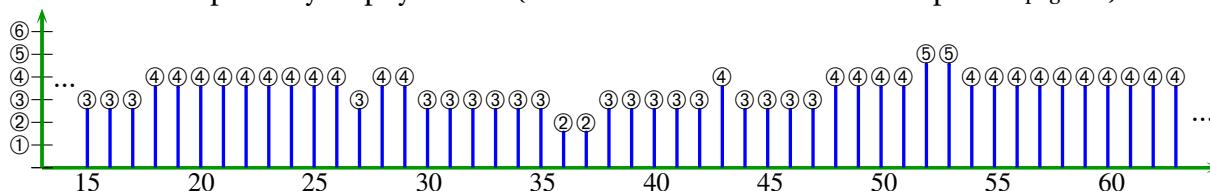
<sup>55</sup> See experiment log file "rdie\_lp\_12000m16.xls" generated by the program "ssp.exe".

<sup>56</sup> See experiment log file "rdie\_lp\_12000m50.xls" generated by the program "ssp.exe".

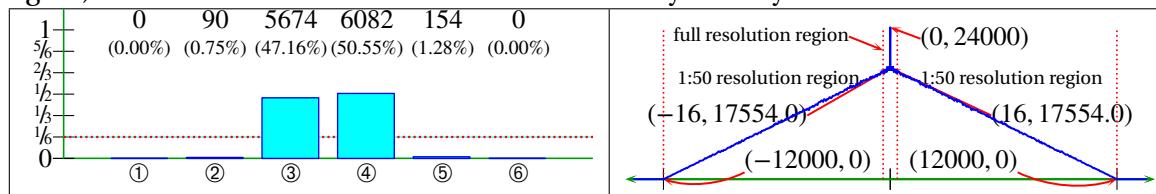
- (d) Using the *Euclidean metric* in  $\mathbb{R}^3$  rather than the *Lagrange arc distance* as in item (6) (length 50 rectangular filter) yields sequences that are **identical**.<sup>57</sup>

**Example 4.2** (low pass filtering of spinner sequence)

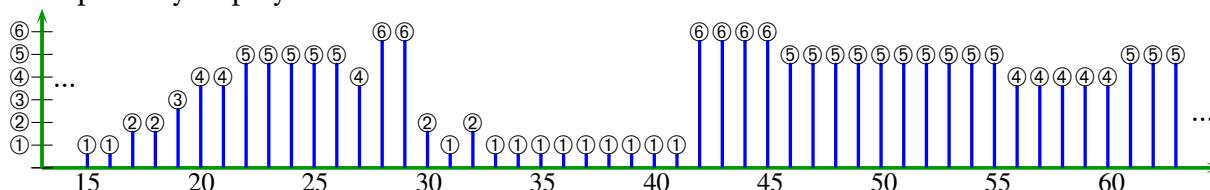
- (1) Consider the pseudo-uniformly distributed spinner sequence presented in Example 3.6 (page 16). As in Example 4.1 (page 23), suppose we want to *filter* this sequence with a *low pass rectangular sequence* in order to “smooth out” the sequence.
- (2) Suppose we first use the  $\mathbb{R}^1$  *spinner random variable* (Definition 3.20 page 21) to map the spinner sequence into  $\mathbb{R}^1$ . *Filtering* this mapped sequence using the *length 16 rectangular low pass sequence* and then mapping the result back to a *spinner sequence* using the *Euclidean metric*, produces the result partially displayed here (in essence the same as in Example 4.1 page 23):



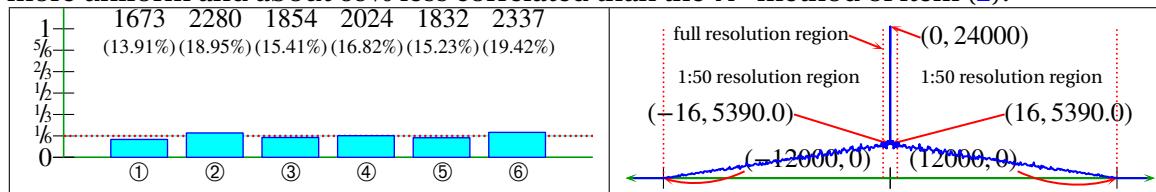
Again, it's uniform distribution has been essentially destroyed.



- (3) Alternatively, suppose we next try using the *QPSK spinner random variable* (Definition 3.21 page 21) to map the spinner sequence into  $\mathbb{C} \cong \mathbb{R}^2$ . *Filtering* this new sequence using the *length 16 rectangular low pass sequence* in the  $\mathbb{R}^2$  *spinner distance linear space* (Definition 3.27 page 22) and then mapping back to a *sequence over the spinner outcome subspace* using the *Lagrange arc distance* yields the result partially displayed here:



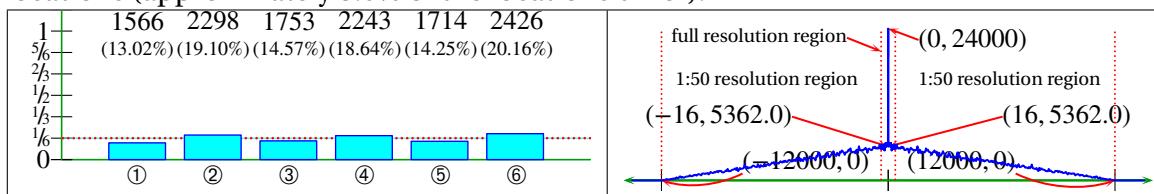
Note that the sequence does appear to be “smoothed out”, but this time the distribution is much more uniform and about 69% less correlated than the  $\mathbb{R}^1$  method of item (2):



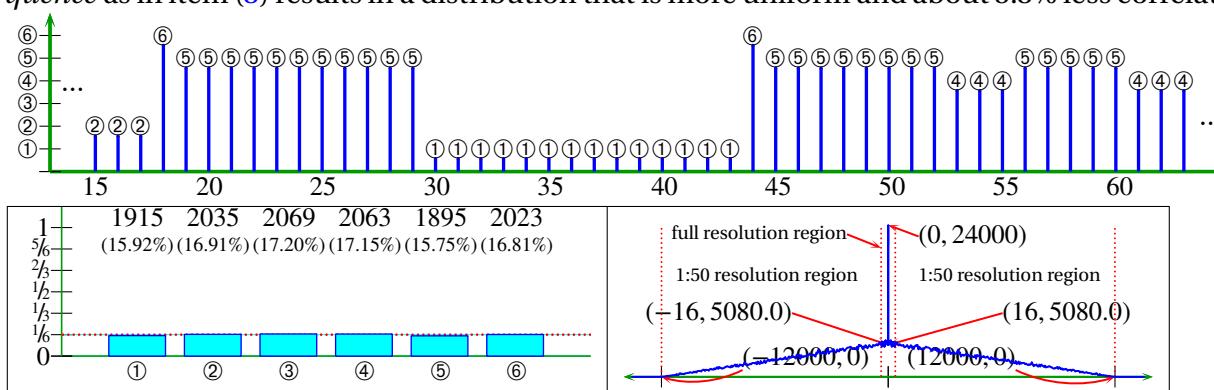
Furthermore, it is about 58% less correlated than the  $\mathbb{R}^3$  filtering for the die sequence used in item (3) of Example 4.1 (page 23).

<sup>57</sup> See experiment log file “rdie\_lp\_12000m50.xls” generated by the program “ssp.exe”.

- (4) Using the *Euclidean metric* rather than the *Lagrange arc distance* as in item (3) results in a sequence that differs at 356 different locations out of  $N + M + M - 1 = 12000 + 16 + 15 = 12031$  locations (approximately 3.0% of the locations differ).<sup>58</sup>



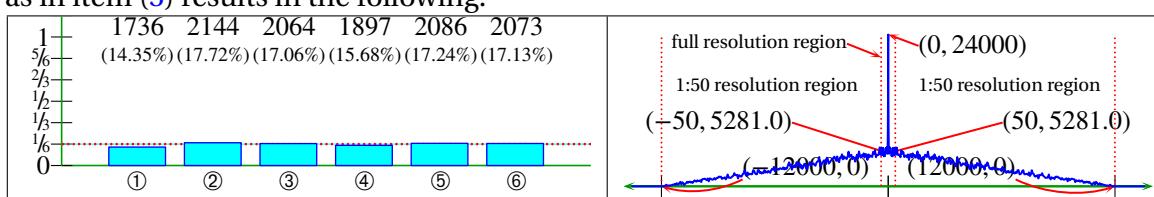
- (5) Using a *length 16 Hanning low pass sequence* rather than the *length 16 Rectangular low pass sequence* as in item (3) results in a distribution that is more uniform and about 5.3% less correlated:



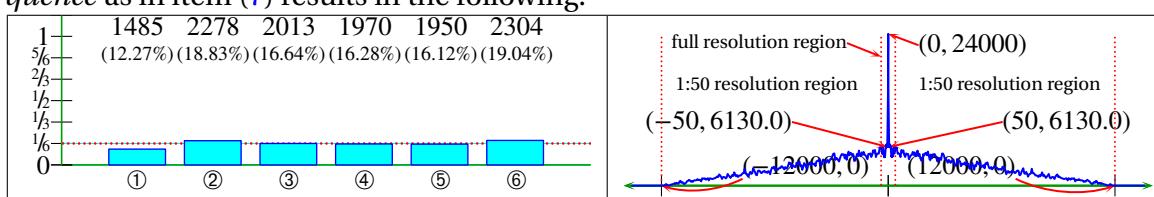
- (6) Using the *Euclidean metric* rather than the *Lagrange arc distance* as in item (5) results in a sequence that differs at exactly 2 locations (approximately 0.017%) out of 12031 locations:<sup>59</sup>

n	Euclidean	Lagrange
4149	☒	☒
5594	☒	☐

- (7) Using a *length 50 Hanning low pass sequence* rather than the *length 16 Hanning low pass sequence* as in item (5) results in the following:



- (8) Using a *length 50 Rectangular low pass sequence* rather than the *length 50 Hanning low pass sequence* as in item (7) results in the following:

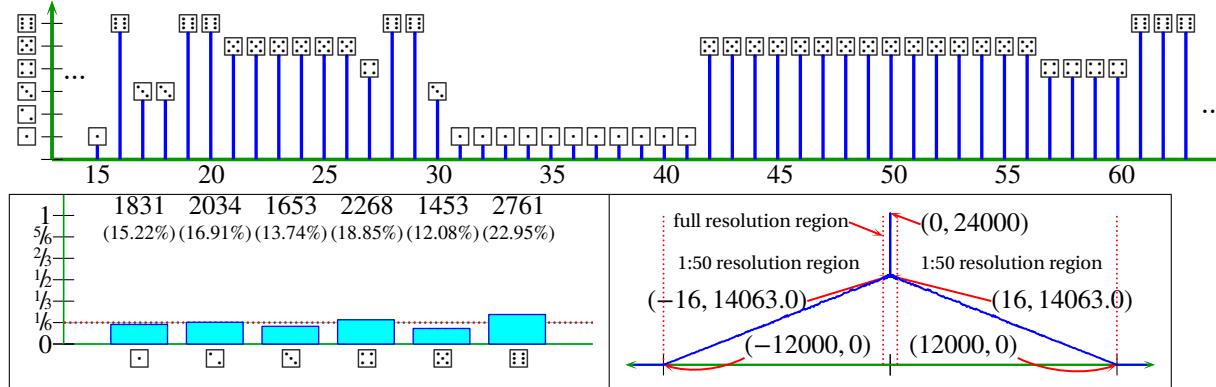


#### Example 4.3 (low pass filtering of fair die sequence)

<sup>58</sup> ⓘ See experiment log file "spin\_lp\_12000m16.xls" generated by the program "ssp.exe".

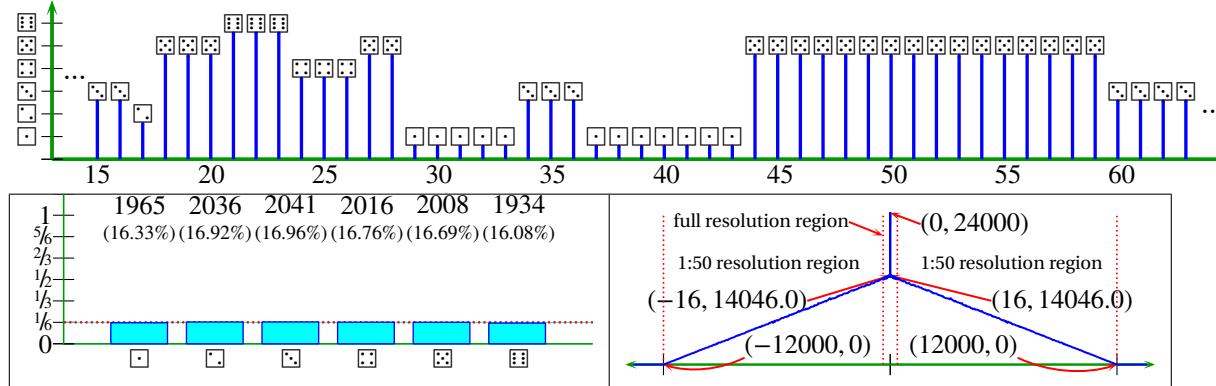
<sup>59</sup> ⓘ See experiment log file "spin\_lp\_12000m16.xls" generated by the program "ssp.exe".

- (1) Consider the pseudo-uniformly distributed die sequence presented in Example 3.4 (page 15). Suppose we want to *filter* this sequence with a *low pass sequence* in order to “smooth out” the sequence, just as in Example 4.1 (page 23).
- (2) Suppose we first use the *traditional die random variable* (Definition 3.15 page 21) to map the die sequence into  $\mathbb{R}^1$ . *Filtering* this mapped sequence using the *length 16 rectangular low pass sequence* and then mapping the result back to a *die sequence* using the *Euclidean metric*, produces a result identical to that of item (2) (page 24) of Example 4.1.
- (3) Alternatively, suppose we next use the  $\mathbb{R}^6$  *die random variable* (Definition 3.19 page 21) to map the die sequence into  $\mathbb{R}^6$ . *Filtering* this new sequence using the *length 16 rectangular low pass sequence* in the  $\mathbb{R}^6$  *die distance linear space* (Definition 3.28 page 23) and then mapping back to a *die sequence* using the *Euclidean metric* yields a much more uniform distribution and a sequence that is about 28% less correlated.



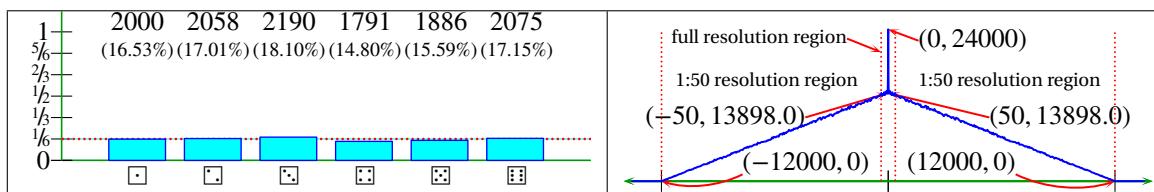
Note further that this  $\mathbb{R}^6$  technique yeilds a sequence that is about 9.0% more correlated than yielded by the  $\mathbb{R}^3$  technique used in item (3) of Example 4.1 (page 23).

- (4) Using a *length 16 Hanning low pass sequence* rather than the *length 16 Rectangular low pass sequence* as in item (3) results in a distribution that is more uniform and a sequence that is about 0.12% less correlated:



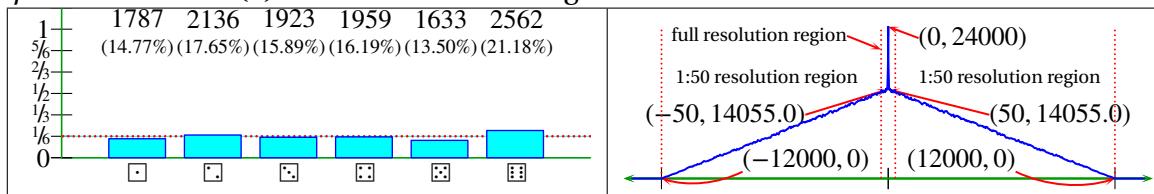
Note further that this  $\mathbb{R}^6$  technique yields a sequence that is about 10% more correlated than yielded by the  $\mathbb{R}^3$  technique used in item (4) of Example 4.1 (page 23).

- (5) Using a *length 50 Hanning low pass sequence* rather than the *length 16 Hanning low pass sequence* as in item (4) results in the following:



Note further that this  $\mathbb{R}^6$  technique yields a sequence that is about 11% more correlated than yielded by the  $\mathbb{R}^3$  technique used in item (5) of Example 4.1 (page 23).

- (6) Using a *length 50 Rectangular low pass sequence* rather than the *length 50 Hanning low pass sequence* as in item (5) results in the following:



Note further that this  $\mathbb{R}^6$  technique yeilds a sequence that is about 8.8% more correlated than yeilded by the  $\mathbb{R}^3$  technique used in item (6) of Example 4.1 (page 23).

- (7) In the *fair die outcome space*, the *Lagrange arc distance* does not seem so appropriate. That being said however, ...

- (a) using the *Lagrange arc distance* rather than the *Euclidean metric* in item (3) page 28 yields results that are **identical**<sup>60</sup>
- (b) using the *Lagrange arc distance* rather than the *Euclidean metric* in item (4) page 28 yields results that **differ** at 8 locations (differ at approximately 0.07% of the total possible  $N + M + M - 1 = 12000 + 16 + 16 - 1 = 12031$  locations):<sup>61</sup>

n	Euclidean	Lagrange
2181	◻	◻
2589	◻	◻
3521	◻	◻
5385	◻	◻

n	Euclidean	Lagrange
8729	◻	◻
10866	◻	◻
11372	◻	◻
11895	◻	◻

- (c) using the *Lagrange arc distance* rather than the *Euclidean metric* in item (5) page 28 yields results that are **identical**<sup>62</sup>
  - (d) using the *Lagrange arc distance* rather than the *Euclidean metric* in item (6) page 29 yields results that **differ** at 289 locations (differ at approximately 2.4% of the total possible 12031 locations):<sup>63</sup>
- (8) Empirical evidence observed in items 3, 4, 5, and 6, suggests that the  $\mathbb{R}^6$  technique of this example leads to about 10% more correlation than the  $\mathbb{R}^3$  technique of Example 4.1 (page 23).

## 4.2 High pass filtering

### Example 4.4 (high pass filtering of weighted real die sequence)

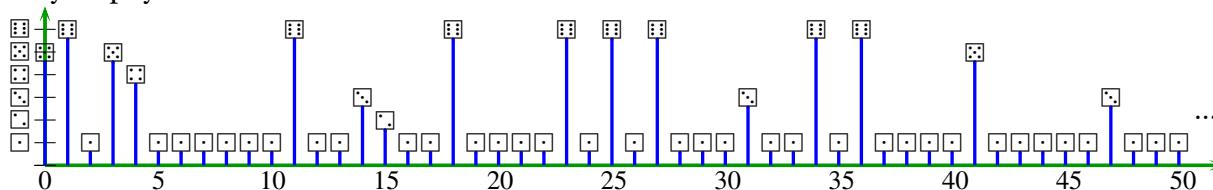
<sup>60</sup> See experiment log file “fdie\_lp\_12000m16.xls” generated by the program “ssp.exe”.

<sup>61</sup> See experiment log file “fdie\_lp\_12000m16.xls” generated by the program “ssp.exe”.

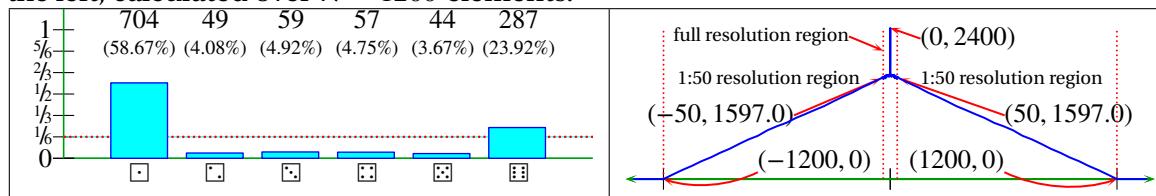
<sup>62</sup> See experiment log file “fdie\_lp\_12000m50.xls” generated by the program “ssp.exe”.

<sup>63</sup> See experiment log file “fdie\_lp\_12000m50.xls” generated by the program “ssp.exe”.

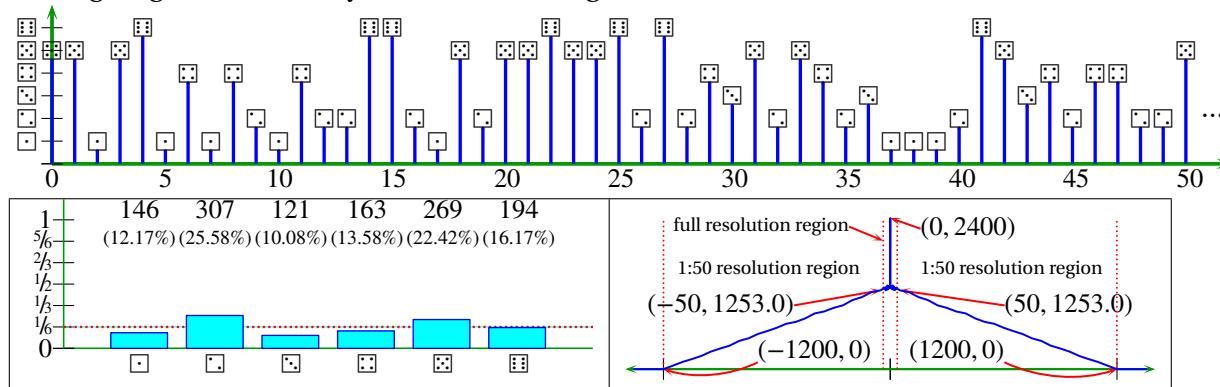
- (1) Consider a length  $50(1200+2)-(50-1) = 60051$  non-uniformly distributed die sequence generated as described in Example 3.7 (page 17). To remove the strong  $\blacksquare$  bias, we could map and *filter* (Definition 1.41 page 9) the sequence with the *length 50 high pass rectangular sequence* (Definition 1.43 page 9). Such filtering will obviously introduce correlation into the die sequence. The low pass filtering of Example 4.1 page 23 (“smoothing”) also introduced correlation, but wanting a “smooth” sequence informally implies a willingness to accept a highly correlated sequence. However in this current example, we would prefer to have an *uncorrelated* sequence. To negate the correlation introduced by filtering, we *down sample* (Definition 1.33 page 7) the filtered sequence by a factor of 50 and remove the first and last element, leaving a sequence of length 1200.
- (2) If the filtering and downsampling described in item (1) is performed in the traditional  $\mathbb{R}^1$  space, then after mapping back to a *die sequence* using the *Euclidean metric*, we obtain the result partially displayed here...



where the bias at  $\blacksquare$  has been replaced by a new bias at  $\square$ , as illustrated quantitatively below on the left, calculated over  $N = 1200$  elements.

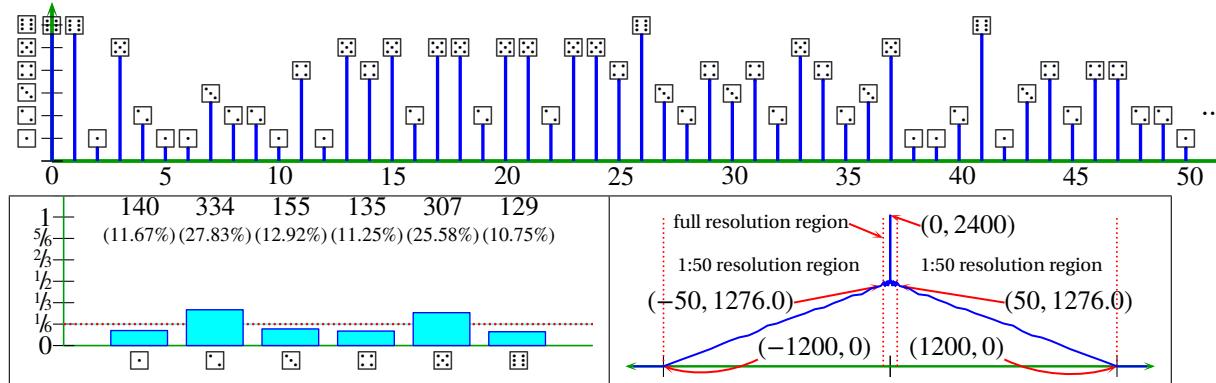


- (3) Alternatively, suppose we next use the  $\mathbb{R}^3$  *die random variable* (Definition 3.18 page 21) to map the die sequence into  $\mathbb{R}^3$ . *Filtering* this new sequence using the *length 50 rectangular high pass sequence* in the  $\mathbb{R}^3$  *distance linear space* (Definition 3.26 page 22) and then mapping back to a *die sequence* using the *Lagrange arc distance* yields the following results:



Note that neither the  $\mathbb{R}^1$  method of item (2) nor the  $\mathbb{R}^3$  method of item (3) yields a uniformly distributed sequence; but the  $\mathbb{R}^3$  method at least comes significantly closer to this end. Moreover, the  $\mathbb{R}^3$  method also yields a sequence that is less correlated.

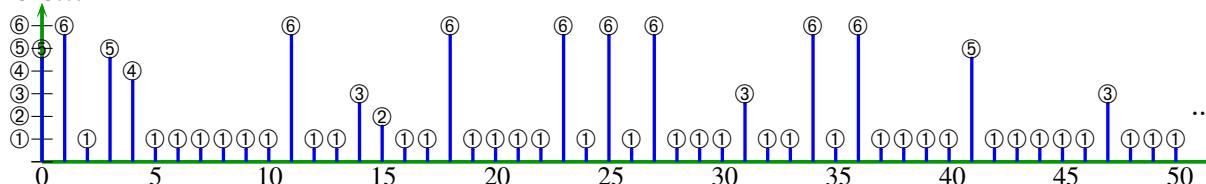
- (4) Replacing the *length 50 rectangular high pass filter* in item (3) with the *length 50 Hanning high pass filter* (Definition 1.47 page 10) yields a different sequence with similar distribution but is slightly more correlated:



- (5) Replacing the *Lagrange arc distance* by the *Euclidean metric* in this example has very little effect, even before downsampling. Before downsampling, the length of each sequence is  $M(N + 2) = 50(1202) = 60100$  elements. More details follow:
  - (a) Using the *Euclidean metric* rather than the *Lagrange arc distance* in item (3) yields results that are **identical**.<sup>64</sup>
  - (b) Using the *Euclidean metric* rather than the *Lagrange arc distance* in item (4) yields results that **differ** at 7 locations (approximately 0.01% of all the locations).<sup>65</sup>
- (6) For the type of sequence processing described in this example, item (5) very informally *suggests* the following:
  - (a) The processing is not highly sensitive to the choice of distance function.
  - (b) The processing is not heavily dependent on the *triangle inequality*.

#### Example 4.5 (high pass filtering of weighted spinner sequence)

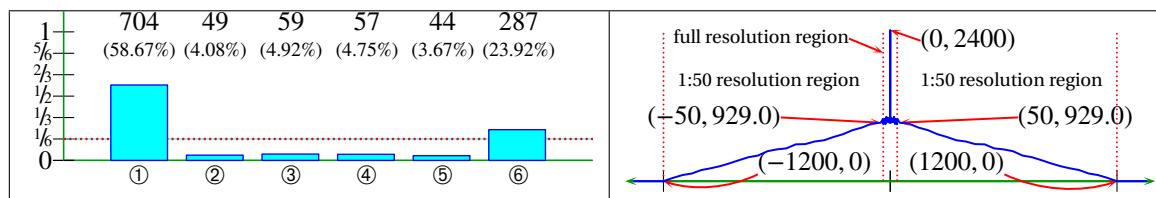
- (1) Consider a length  $50(1200 + 2) - (50 - 1) = 60051$  non-uniformly distributed *spinner sequence* generated as described in Example 3.9 (page 17). To remove the strong ⑤ bias, we could *filter* the sequence with the *length 50 high pass rectangular sequence* and down sample the filtered sequence by a factor of 50, as described in Example 4.4 (page 29).
- (2) If the filtering described in item (1) is performed in the traditional  $\mathbb{R}^1$  space, then after mapping back to a *spinner sequence* using the *Euclidean metric*, we obtain the result partially displayed here...



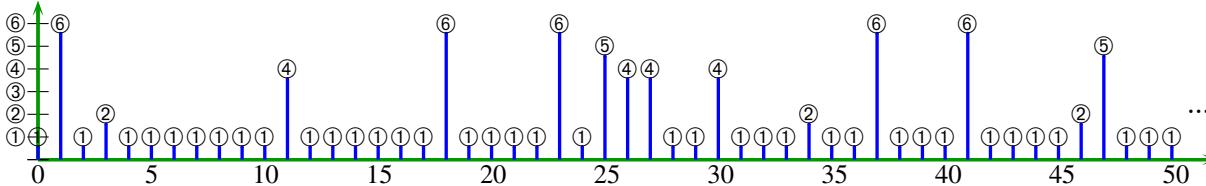
where the bias at ⑤ has been replaced by a new bias at ①, as illustrated quantitatively below on the left, calculated over 1200 elements.

<sup>64</sup> See experiment log file “wrdie\_hp\_1200m50.xls” generated by the program “ssp.exe”.

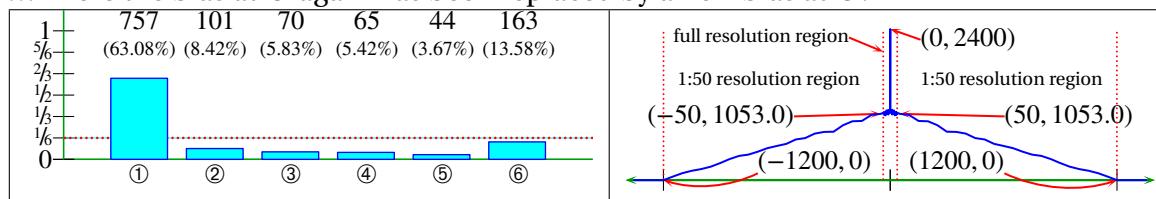
<sup>65</sup> See experiment log file “wrdie\_hp\_1200m50.xls” generated by the program “ssp.exe”.



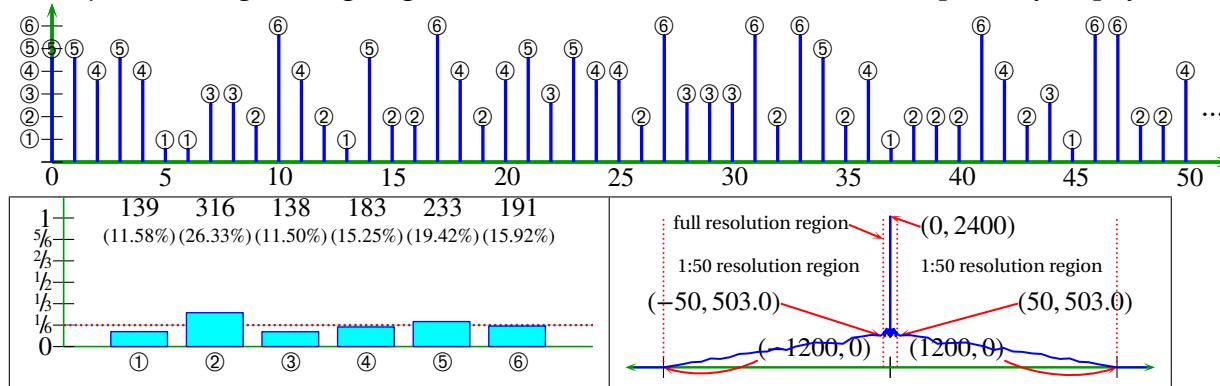
- (3) If we replace the *length 50 rectangular high pass filter* of item (2) with a *length 50 Hanning high pass filter* then we obtain the result partially displayed here...



...where the bias at ⑤ again has been replaced by a new bias at ①:



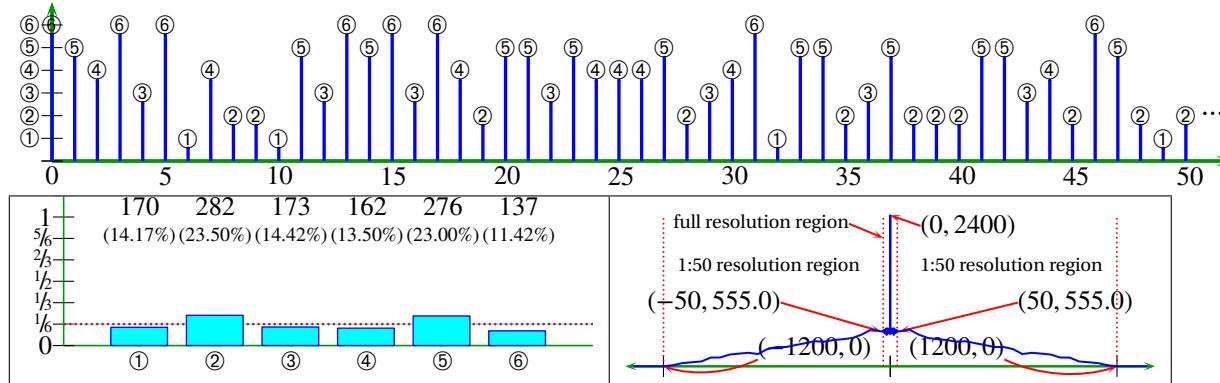
- (4) If the rectangular filtering in  $\mathbb{R}^1$  of item (2) is instead performed in  $\mathbb{R}^2$  and mapped back to a *spinner sequence* using the *Lagrange arc distance*, then we obtain the result partially displayed here:



Note that neither the  $\mathbb{R}^1$  methods (described in item (2) and item (3)) nor the  $\mathbb{R}^2$  method (described in item (4)) yields a uniformly distributed sequence; but the  $\mathbb{R}^2$  method at least comes significantly closer to this end.

- (5) Replacing the *Lagrange arc distance* by the *Euclidean metric* as in item (4) yields a sequence that differs at a total of 410 locations (approximately 0.7% of the locations).<sup>66</sup>
- (6) If instead of using the rectangular filtering (as in item (4)), we use the Hanning filtering of item (3) in  $\mathbb{R}^2$  and map back to a *spinner sequence* using the *Lagrange arc distance*, then we obtain the result partially displayed here:

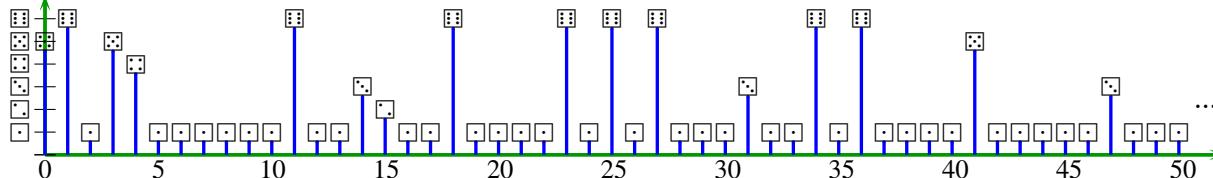
<sup>66</sup> See experiment log file "wspin\_hp\_1200m50.xlg" generated by the program "ssp.exe".



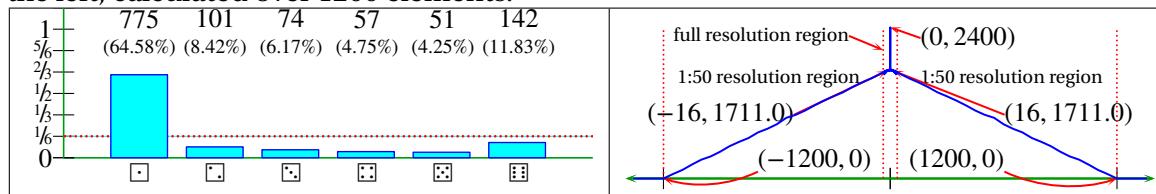
- <sup>67</sup> Replacing the *Lagrange arc distance* by the *Euclidean metric* in item (6) yields a very similar result: yields a sequence that differs at a total of 3 locations (approximately 0.005% of the locations).

**Example 4.6** (high pass filtering of weighted die sequence)

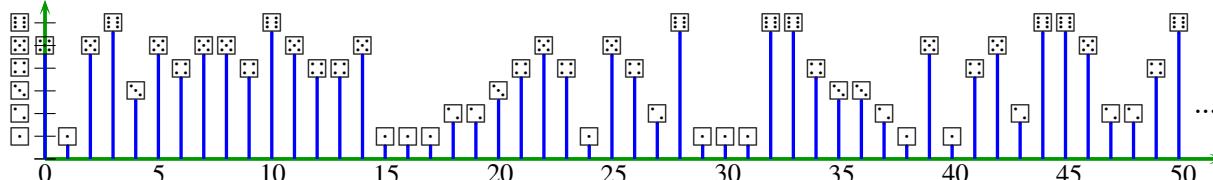
- (1) Consider a length  $50(1200 + 2) - (50 - 1) = 60051$  weighted die sequence generated as described in Example 3.8 (page 17). To remove the strong  $\square$  bias, we could map and filter the sequence with the length 16 high pass rectangular sequence (Example 1.44 page 9). To negate the correlation introduced by filtering, we down sample the filtered sequence by a factor of 16.
  - (2) If the die sequence of item (1) is mapped into  $\mathbb{R}^1$  using the traditional die random variable (Definition 3.15 page 21), filtered, down sampled, and mapped back to a die sequence using the Euclidean metric, we obtain the result partially displayed here...



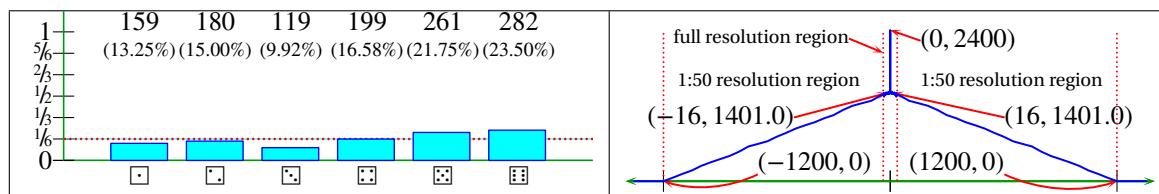
where the bias at  $\square$  has been replaced by a new bias at  $\square$ , as illustrated quantitatively below on the left, calculated over 1200 elements.



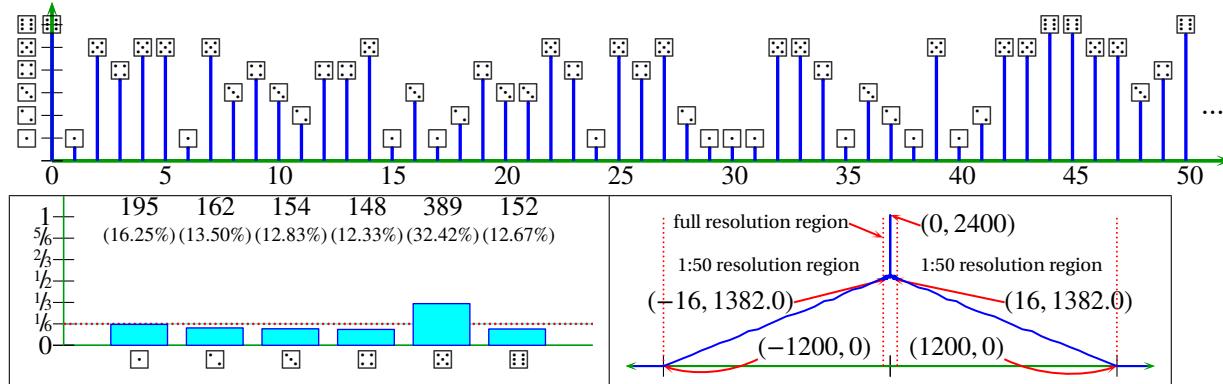
- (3) But if instead of processing the die sequence in  $\mathbb{R}^1$  as in item (2), processing is performed in  $\mathbb{R}^6$  and mapped back to a die sequence using the *Euclidean metric*, then we obtain the result partially displayed here:



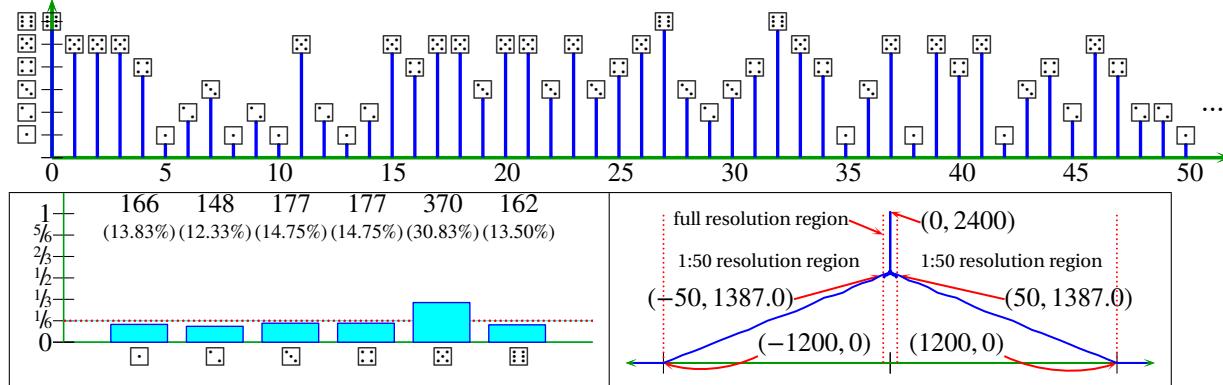
<sup>67</sup> See experiment log file “wspin\_hp\_1200m50.xlg” generated by the program “ssp.exe”.



- (4) Replacing the *length 16 rectangular sequence* in item (3) with a *length 16 Hanning sequence* in  $\mathbb{R}^6$  yields the following results:

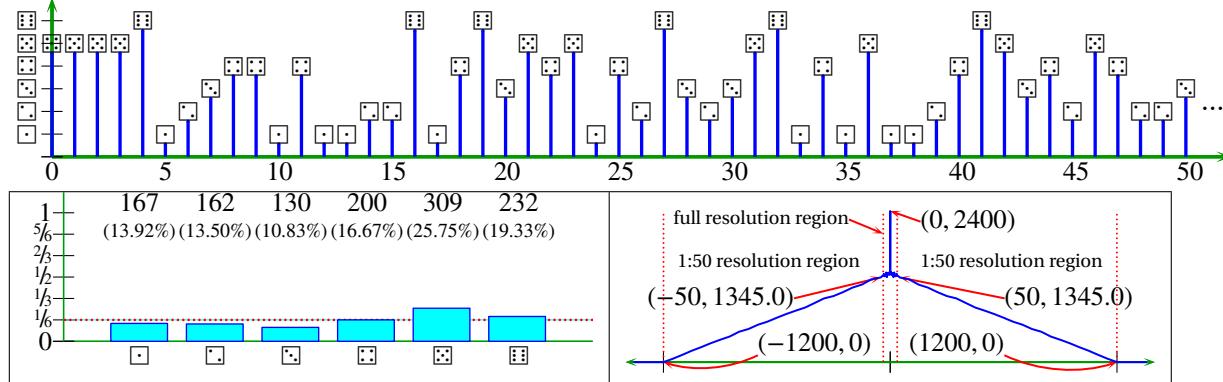


- (5) Replacing the *length 16 Hanning sequence* in item (4) with a *length 50 Hanning sequence* in  $\mathbb{R}^6$  yields the following results:



Note that this  $\mathbb{R}^6$  technique yields a sequence that is about 8.7% more correlated than yielded by the  $\mathbb{R}^3$  technique used in item (4) of Example 4.4 (page 29).

- (6) Replacing the *length 50 Hanning sequence* in item (5) with a *length 50 rectangular sequence* in  $\mathbb{R}^6$  yields the following results:



Note that this  $\mathbb{R}^6$  technique yields a sequence that is about 7.3% more correlated than yielded by the  $\mathbb{R}^3$  technique used in item (3) of Example 4.4 (page 29).

- (7) As in Example 4.3 (page 27), here again the *Lagrange arc distance* does not seem so appropriate. That again being said however, ...
  - (a) using the *Lagrange arc distance* rather than the *Euclidean metric* in item (3) page 33 yields results that are **identical**.<sup>68</sup>
  - (b) using the *Lagrange arc distance* rather than the *Euclidean metric* in item (4) page 34 yields results that **differ** at 16 locations (differ at approximately 0.08% of the total possible  $M(N + 2) = 16(1200 + 2) = 19232$  locations).<sup>69</sup>
- (8) Empirical evidence observed in items item (5) and item (6) suggests that the  $\mathbb{R}^6$  technique of this example leads to about 8% more correlation than the  $\mathbb{R}^3$  technique of Example 4.4 (page 29).

### 4.3 Fourier Analysis

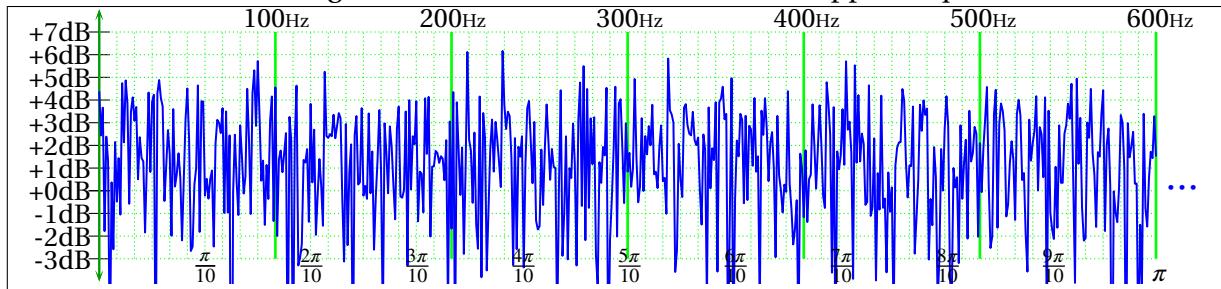
**Example 4.7** (length 1200 non-stationary die sequence with 10Hz oscillating mean)

- (1) Suppose we have a length  $N \triangleq 1200$  die sequence  $(x_n)$  with the following distribution:

$$\begin{aligned} P(\square) &= P(\square) = P(\square) = P(\square) = P(\square) = 0.15 \quad \text{and} \quad P(\square) = 0.25 \\ &\text{for } n \in \left\{ p + (2m)\frac{M}{2} \mid p = 0, 1, \dots, \frac{M}{2} - 1, m = 0, 1, 2, \dots, 9 \right\} \quad \text{and} \\ P(\square) &= P(\square) = P(\square) = P(\square) = P(\square) = 0.15 \quad \text{and} \quad P(\square) = 0.25 \\ &\text{for } n \in \left\{ p + (2m + 1)\frac{M}{2} \mid p = 0, 1, \dots, \frac{M}{2} - 1, m = 0, 1, 2, \dots, 9 \right\} \end{aligned}$$

where  $M \triangleq 120$ . That is, the distribution of the sequence oscillates every  $\frac{M}{2} = 60$  samples between one that favors  $\square$  and one that favors  $\square$ . Moreover, if we were to evaluate the sequence using a *Discrete Fourier Transform* operator  $DFT$  (Definition 1.52 page 10), we might expect to see a strong component at  $\frac{N}{M} = 10$  (or 10 Hz—the distribution goes through 10 cycles during the course of the sequence).

- (2) Suppose we first use the *PAM die random variable* (Definition 3.16 page 21) to map the sequence of item (1) into  $\mathbb{R}^1$ . The magnitude of the  $DFT : \mathbb{R}^1 \rightarrow \mathbb{C}^1$  of the mapped sequence is as follows:



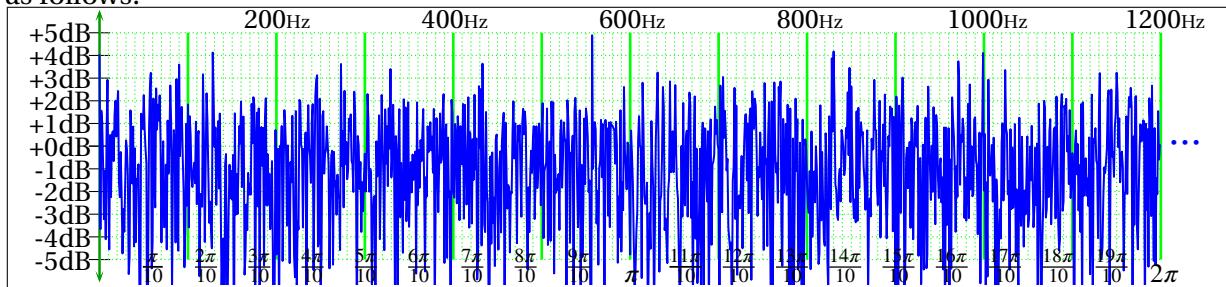
Looking at the above result, it would be next to impossible to discern that the distribution had a significantly strong oscillation of 10 cycles. In fact, the magnitude of the DFT at 10Hz is only 0.895699, or  $10 \log_{10}(0.895699) = -0.478377$  dB. There are exactly 456 out of a total  $\frac{M}{2} = 600$  values that are greater than the DFT magnitude at 10Hz.<sup>70</sup> That is, to either a human observer or a machine algorithm, the 10Hz component is effectively lost in the noise.

<sup>68</sup> See experiment log file “wdie\_hp\_1200m16.xls” generated by the program “ssp.exe”.

<sup>69</sup> See experiment log file “wdie\_hp\_1200m16.xls” generated by the program “ssp.exe”.

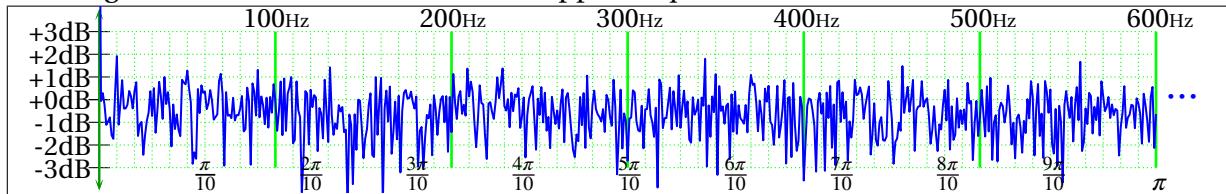
<sup>70</sup> See experiment log file “diedft\_1525\_1200m120.xls” generated by the program “ssp.exe”.

- (3) Suppose we next use the *QPSK die random variable* (Definition 3.17 page 21) to map the sequence into the complex plane. The magnitude of the  $\text{DFT} : \mathbb{C}^1 \rightarrow \mathbb{C}^1$  operation on the mapped sequence is as follows:



The magnitude of the DFT at 10Hz is 0.589990, or  $10 \log_{10}(0.589990) = -2.291552$  dB. There are exactly 831 out of a total  $N = 1200$  values that are greater than the DFT magnitude at 10Hz.<sup>71</sup> Again, the 10Hz component is effectively lost in the noise.

- (4) Suppose we next use the  $\mathbb{R}^6$  die random variable (Definition 3.19 page 21) to map the sequence into  $\mathbb{R}^6$ . The magnitude of  $\text{DFT} : \mathbb{R}^6 \rightarrow \mathbb{C}^6$  of the mapped sequence is as follows:



The magnitude at 10Hz is 1.556295, or  $10 \log_{10}(1.556295) = 1.920920$  dB. Besides the DC component (0Hz component), this is the uniquely greatest value of the 600 samples. And in fact, there are only 5 out of a total  $n_h = 600$  samples that are  $0.90 \times 1.556295$  or greater.<sup>72</sup> Thus, using the  $\mathbb{R}^6$  mapping technique of this example, it is much simpler to detect the 10Hz oscillating distribution.

#### Example 4.8 (length 12000 non-stationary die sequence with 10Hz oscillating mean)

- (1) Suppose we have a length  $N \triangleq 12000$  die sequence  $(x_n)$  with the following distribution:

$$\begin{aligned} P(\square) &= P(\square) = P(\square) = P(\square) = P(\square) = 0.16 \quad \text{and} \quad P(\square) = 0.20 \\ &\text{for } n \in \left\{ p + (2m)\frac{M}{2} \mid p = 0, 1, \dots, \frac{M}{2} - 1, m = 0, 1, 2, \dots, 9 \right\} \quad \text{and} \\ P(\square) &= P(\square) = P(\square) = P(\square) = P(\square) = 0.16 \quad \text{and} \quad P(\square) = 0.20 \\ &\text{for } n \in \left\{ p + (2m + 1)\frac{M}{2} \mid p = 0, 1, \dots, \frac{M}{2} - 1, m = 0, 1, 2, \dots, 9 \right\} \end{aligned}$$

where  $M \triangleq 1200$ . That is, the distribution of the sequence oscillates every  $M_h = 600$  samples between one that favors  $\square$  and one that favors  $\square$ . If we were to evaluate the sequence using the *Discrete Fourier Transform* operator, we again might expect to see a strong component at  $\frac{N}{M} = 10$  (or 10 Hz—the distribution goes through 10 cycles during the course of the sequence).

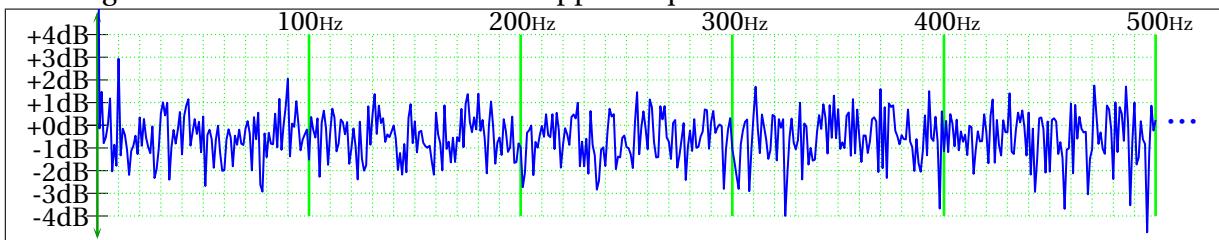
- (2) Suppose we first use the *PAM die random variable* (Definition 3.16 page 21) to map the sequence of item (1) into  $\mathbb{R}^1$ . In the magnitude of  $\text{DFT} : \mathbb{R}^1 \rightarrow \mathbb{C}^1$  there are 1130 values out of a possible  $n_h = 6000$  values greater than the value at 10Hz (that value being 2.174512).<sup>73</sup> As in Example 4.7 (page 35), the subtle 10Hz component is effectively lost in the noise.

<sup>71</sup> See experiment log file “diedft\_1525\_1200m120.xlg” generated by the program “ssp.exe”.

<sup>72</sup> See experiment log file “diedft\_1525\_1200m120.xlg” generated by the program “ssp.exe”. The 5 largest values are the points (0, 14.251433), (10, 1.556295), (344, 1.513501), (456, 1.405843) and (557, 1.468970).

<sup>73</sup> See experiment log file “diedft\_1620\_12000m1200.xlg” generated by the program “ssp.exe”.

- (3) Suppose we next use the *QPSK die random variable* (Definition 3.17 page 21) to map the sequence into the complex plane. There are exactly 1932 out of a total  $N = 12000$  values that are greater than the DFT value at 10Hz (that value being 1.348693).<sup>74</sup> As in Example 4.7 (page 35), the subtle 10Hz component is effectively lost in the noise.
- (4) Suppose we next use the  $\mathbb{R}^6$  *die random variable* (Definition 3.19 page 21) to map the sequence into  $\mathbb{R}^6$ . The magnitude of DFT :  $\mathbb{R}^6 \rightarrow \mathbb{C}^6$  of the mapped sequence is as follows:



Besides the DC component, the value at 100Hz (that value being (1.965018) is the uniquely greatest value of the  $n_h = 6000$  samples; and it is  $10 \log_{10}(1.965018/1.660189) = 0.699 \dots$  dB larger than the next largest value.<sup>75</sup> Thus, even though the oscillating distribution is very subtle (even more subtle than that of Example 4.7 (page 35)), the  $\mathbb{R}^6$  mapping technique and subsequent analysis are able to detect it.

#### Example 4.9 (length 12000 non-stationary die sequence with 100Hz oscillating mean)

- (1) Suppose we have a length  $N \triangleq 12000$  die sequence  $(x_n)$  with the following distribution:
- $$P(\square) = P(\square) = P(\square) = P(\square) = P(\square) = 0.16 \quad \text{and} \quad P(\square) = 0.20$$
- $$\text{for } n \in \left\{ p + (2m)\frac{M}{2} \mid p = 0, 1, \dots, \frac{M}{2} - 1, m = 0, 1, 2, \dots, 9 \right\} \quad \text{and}$$
- $$P(\square) = P(\square) = P(\square) = P(\square) = P(\square) = 0.16 \quad \text{and} \quad P(\square) = 0.20$$
- $$\text{for } n \in \left\{ p + (2m + 1)\frac{M}{2} \mid p = 0, 1, \dots, \frac{M}{2} - 1, m = 0, 1, 2, \dots, 9 \right\}$$
- where  $M \triangleq 120$ . That is, the distribution of the sequence oscillates every  $m_h = 60$  samples between one that favors  $\square$  and one that favors  $\square$ . If we were to evaluate the sequence using the *Discrete Fourier Transform* operator, we might expect to see a strong component at  $\frac{N}{M} = 100$  (or 100 Hz—the distribution goes through 100 cycles during the course of the sequence).
- (2) Suppose we first use the *PAM die random variable* (Definition 3.16 page 21) to map the sequence of item (1) into  $\mathbb{R}^1$ . In the magnitude DFT :  $\mathbb{R}^1 \rightarrow \mathbb{C}^1$  there are 1320 values out of a possible  $n_h = 6000$  values greater than the value at 100Hz (that value being 2.081469).<sup>76</sup> The subtle 100Hz component is effectively lost in the noise.
- (3) Suppose we next use the *QPSK die random variable* (Definition 3.17 page 21) to map the sequence into the complex plane. There are exactly 1555 out of a total  $N=12000$  values that are greater than the DFT value at 100Hz (that value being 1.425427).<sup>77</sup> The subtle 100Hz component is effectively lost in the noise.

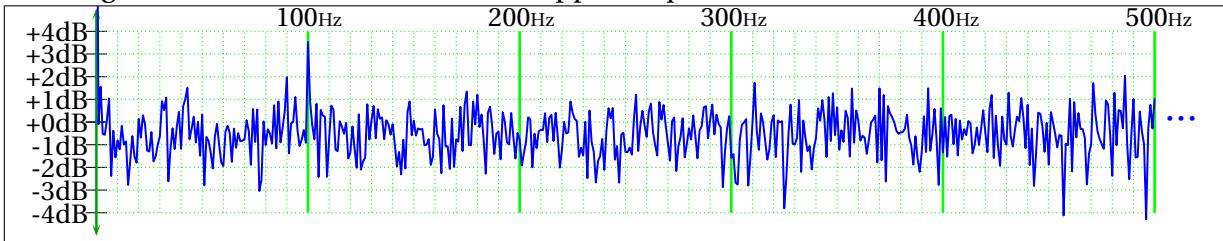
<sup>74</sup> See experiment log file “diedft\_1620\_12000m1200.xlg” generated by the program “ssp.exe”.

<sup>75</sup> See experiment log file “diedft\_1620\_12000m1200.xlg” generated by the program “ssp.exe”. The 10 largest values are (0, 44.763194), (10, 1.965018), (90, 1.602474), (1223, 1.660189), (1313, 1.555349), (2385, 1.551028), (3039, 1.550918), (4154, 1.563756), (4187, 1.586362), and (5147, 1.623052).

<sup>76</sup> See experiment log file “diedft\_1620\_12000m1200.xlg” generated by the program “ssp.exe”.

<sup>77</sup> See experiment log file “diedft\_1620\_12000m1200.xlg” generated by the program “ssp.exe”.

- (4) Suppose we next use the  $\mathbb{R}^6$  die random variable (Definition 3.19 page 21) to map the sequence into  $\mathbb{R}^6$ . The magnitude of  $DFT : \mathbb{R}^6 \rightarrow \mathbb{C}^6$  of the mapped sequence is as follows:



Besides the DC component, the value at 100Hz (that value being 2.256927) is the uniquely greatest value of the  $n_h = 6000$ . and it is  $10 \log_{10}(2.256927/1.599335) = 1.495 \dots$  dB larger than the next largest value.<sup>78</sup> Thus, even though the oscillating distribution is very subtle, the  $\mathbb{R}^6$  mapping technique and subsequent analysis are able to detect it.

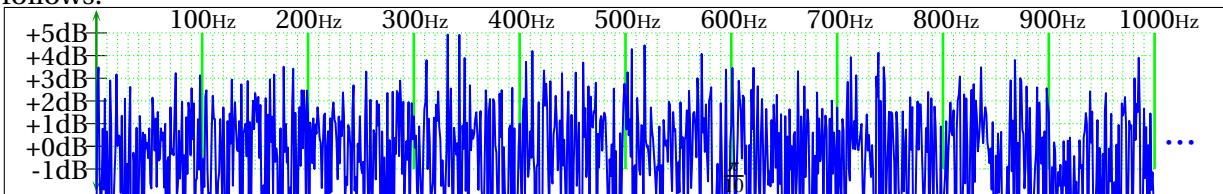
#### Example 4.10 (length 12000 non-stationary artificial DNA sequence with 10Hz oscillating mean)

- (1) Suppose we have a length  $N \triangleq 12000$  die sequence  $(x_n)$  with the following distribution:

$$\begin{aligned} P(\square) &= P(\square) = P(\square) = 0.24 \quad \text{and} \quad P(\square) = 0.28 \\ &\quad \text{for } n \in \left\{ p + 2m \frac{M}{2} \mid p = 0, 1, \dots, \frac{M}{2} - 1, m = 0, 1, 2, \dots, 9 \right\} \quad \text{and} \\ P(\square) &= P(\square) = P(\square) = 0.24 \quad \text{and} \quad P(\square) = 0.28 \\ &\quad \text{for } n \in \left\{ p + (2m + 1) \frac{M}{2} \mid p = 0, 1, \dots, \frac{M}{2} - 1, m = 0, 1, 2, \dots, 9 \right\} \end{aligned}$$

where  $M \triangleq 1200$ . That is, the distribution of the sequence oscillates every  $m_h = 600$  samples between one that favors  $\square$  and one that favors  $\square$ . Moreover, if we were to evaluate the sequence using a *Discrete Fourier Transform* (DFT) operator, we might expect to see a strong component at  $\frac{N}{M} = 10$  (or 10 Hz—the distribution goes through 10 cycles during the course of the sequence).

- (2) Suppose we first use the *PAM DNA random variable* (Definition 3.22 page 21) to map the DNA sequence into  $\mathbb{R}^1$ . The magnitude of  $DFT : \mathbb{R}^1 \rightarrow \mathbb{C}^1$  of the sequence after applying this mapping is as follows:

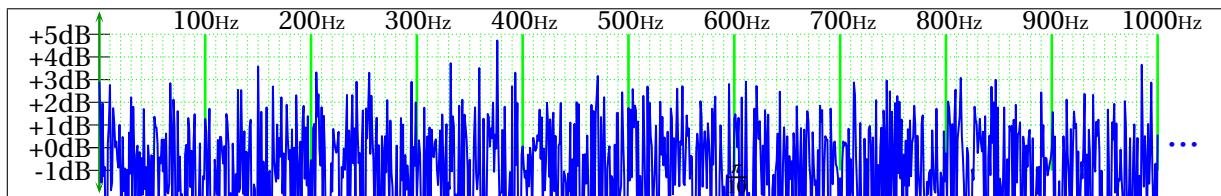


The magnitude of the DFT at 10Hz is only 1.163575 ( $10 \log_{10}(1.163575) = 0.657944$  dB). There are exactly 2023 out of a total  $n_h = 6000$  values that are greater than the DFT value at 10Hz (that value being 1.163575).<sup>79</sup> Here again, the 10Hz component is effectively lost in the noise.

- (3) Suppose we next use the *QPSK DNA random variable* (Definition 3.17 page 21) to map the DNA sequence into the complex plane. The magnitude of  $DFT : \mathbb{C}^1 \rightarrow \mathbb{C}^1$  of the sequence after applying this mapping is as follows:

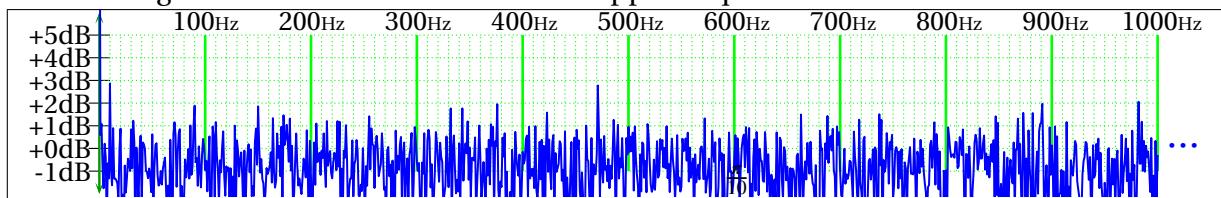
<sup>78</sup> ⓘ See experiment log file “diedft\_1620\_12000m120.xlg” generated by the program “ssp.exe”. The 10 largest values are (0, 44.763060), (90, 1.577597), (100, 2.256927), (486, 1.599335), (1223, 1.585154), (1313, 1.547956), (3039, 1.553522), (3162, 1.561863), (5147, 1.558487), and (5567, 1.533659).

<sup>79</sup> ⓘ See experiment log file “dnadft\_12000m1200.xlg” generated by the program “ssp.exe”.



The DFT at 10Hz is 1.888671, or  $10 \log_{10}(1.888671) = 2.761563$  dB. There are exactly 343 out of a total  $N = 6000$  values that are greater than the DFT value at 10Hz. (that value being 1.888671).<sup>80</sup> Using this mapping it would be difficult to detect the subtle but significant 10Hz component.

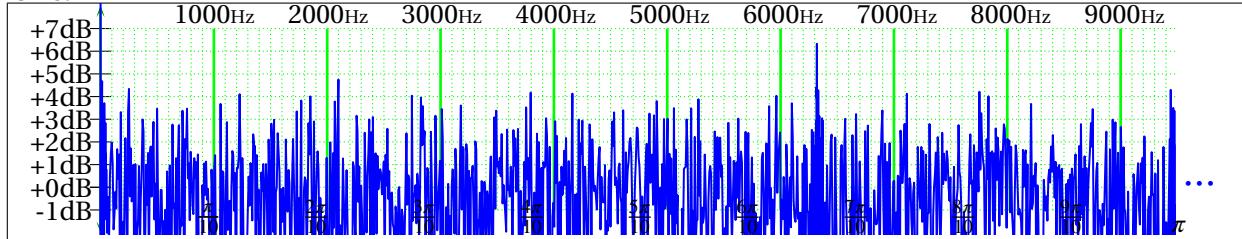
- (4) Suppose we next use the  $\mathbb{R}^4$  DNA random variable (Definition 3.24 page 22) to map the sequence into  $\mathbb{R}^4$ . The magnitude of DFT :  $\mathbb{R}^4 \rightarrow \mathbb{C}^4$  of the mapped sequence is as follows:



The magnitude of the DFT at 10Hz is 1.932042 ( $10 \log_{10}(1.932042) = 2.860166$  dB). Besides itself and the DC component, there are only two out of a total  $N/2 = 6000$  samples that are greater or equal to this value.<sup>81</sup> Thus, using the  $\mathbb{R}^4$  mapping technique and subsequent analysis of this example, it is much simpler to detect the 10Hz oscillation.

#### Example 4.11 (Fourier analysis of Ebola DNA sequence)

- (1) Consider the Ebola DNA sequence described in Example 3.12 (page 19). DNA sequences commonly exhibit a strong DFT harmonic component at  $2\pi/3$  radians.<sup>82</sup>
- (2) Suppose we first use the PAM DNA random variable (Definition 3.22 page 21) to map the DNA sequence into  $\mathbb{R}^1$ . The magnitude of DFT :  $\mathbb{R}^1 \rightarrow \mathbb{C}^1$  of the sequence after applying this mapping is as follows:



The component at  $2\pi/3$  is easy to pick out with a signal to noise ratio (SNR) of  $10 \log_{10}(4.290296/1.123163) \approx 5.8$  dB.<sup>83</sup> Here, the noise value 1.123163 is the RMS (root mean square) of the DFT magnitude sequence from  $n = 1$  to  $n = N/2 - 1$  computed as follows:

$$\sqrt{\frac{1}{N/2 - 1} \sum_{n=1}^{N/2-1} x_n^2}.$$

<sup>80</sup> █ See experiment log file “dnadft\_12000m1200.xlg” generated by the program “ssp.exe”.

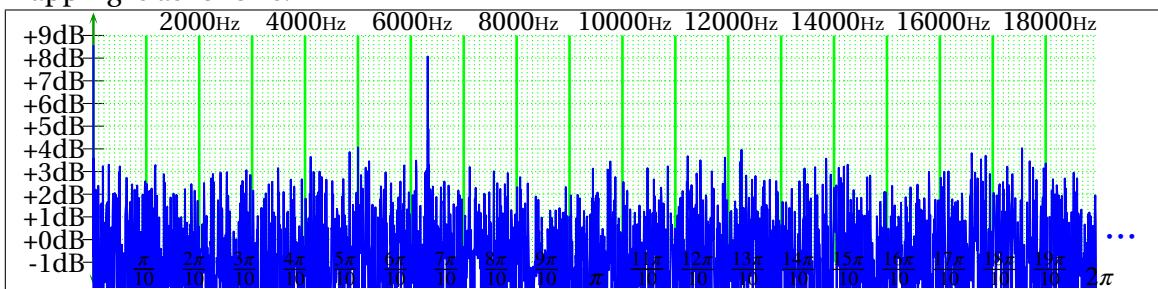
<sup>81</sup> █ See experiment log file “dnadft\_12000m1200.xlg” generated by the program “ssp.exe”.

The 4 largest values are at (0, 54.791926), (10, 1.932042), (4187, 1.962836), and (5147, 2.057553).

<sup>82</sup> █ Galleani and Garello (2010) page 771

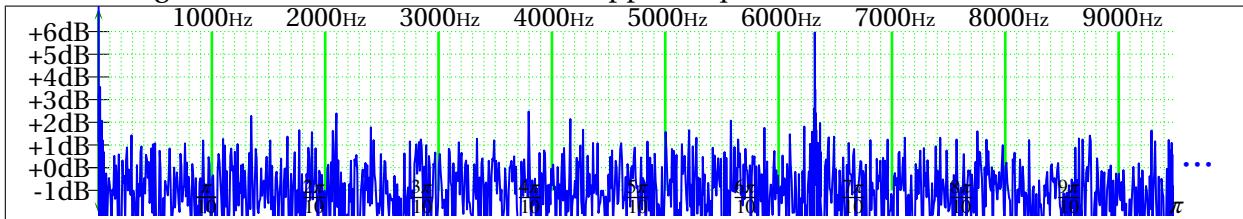
<sup>83</sup> █ See experiment log file “dna\_AF086833\_ebola\_dft.xlg” generated by the program “ssp.exe”.

- (3) Suppose we next use the *QPSK DNA random variable* (Definition 3.17 page 21) to map the dna sequence into the complex plane. The magnitude of  $\text{DFT} : \mathbb{C}^1 \rightarrow \mathbb{C}^1$  of the sequence after applying this mapping is as follows:



The component at  $2\pi/3$  is again easy to pick out with a signal to noise ratio (SNR) of  $10 \log_{10}(6.412578/0.998659) \approx 8.1$  dB.<sup>84</sup> Here, the noise value 0.998659 is the RMS of the DFT magnitude sequence from  $n = 1$  to  $n = N - 1$ .

- (4) Suppose we next use the  $\mathbb{R}^4$  DNA random variable (Definition 3.24 page 22) to map the sequence into  $\mathbb{R}^4$ . The magnitude of  $\text{DFT} : \mathbb{R}^4 \rightarrow \mathbb{C}^4$  of the mapped sequence is as follows:



The component at  $2\pi/3$  is again easy to pick out with a signal to noise ratio (SNR) of  $10 \log_{10}(3.944811/0.860665) \approx 6.6$  dB.<sup>85</sup> Here, the noise value 0.860665 is the RMS of the DFT magnitude sequence from  $n = 1$  to  $n = N/2 - 1$ .

- (5) In conclusion, for this application, there is only a small advantage to using the  $\mathbb{R}^4$  mapping (item (4)) versus the  $\mathbb{R}^1$  mapping (item (2)), and even a demonstrable disadvantage when compared to the  $\mathbb{C}^1$  mapping (item (3)).

## 4.4 Wavelet Analysis

In this section, we use what is in *essense* wavelet analysis, but yet is not truly wavelet analysis in the strict sense:

- (1) For starters, standard *wavelets* and their associated *scaling functions* are not sequences (Definition 1.32 page 7), but rather are functions with domain  $\mathbb{R}$  (not  $\mathbb{Z}$  or some convex subset of  $\mathbb{Z}$ ).
- (2) While it is true that the celebrated *Fast Wavelet Transform* (FWT) does work *internally* with sequences (using *filter banks*),<sup>86</sup> the FWT is actually defined to work on functions with domain  $\mathbb{R}$ ; and so the function to be analyzed by the FWT must first be *sampling* by a *scaling function*, which yields a sequence that can be processed by the *filter banks*.<sup>87</sup>

<sup>84</sup> See experiment log file “dna\_AF086833\_ebola\_dft.xls” generated by the program “ssp.exe”.

<sup>85</sup> See experiment log file “dna\_AF086833\_ebola\_dft.xls” generated by the program “ssp.exe”.

<sup>86</sup> Greenhoe (2013) pages 360–364 (J.6 Filter Banks)

<sup>87</sup> Greenhoe (2013) pages 369–372 (Appendix L)

- (3) Wavelet analysis is typically performed by translating the wavelet or scaling function by fixed amounts depending on the “scale” of the given wavelet. For example, a Haar wavelet of length 4000 would typically “jump” in offsets of 4000: 0, 4000, 8000, 12000, ....<sup>88</sup> As one might imagine, this may be reason for concern if you are using this wavelet to perform edge detection (you might jump over and miss detecting the edge). In this section, wavelet sequences are translated by offsets of 1, making an edge harder to miss.

**Example 4.12** (statistical edge detection using Haar wavelet on non-stationary die sequence)

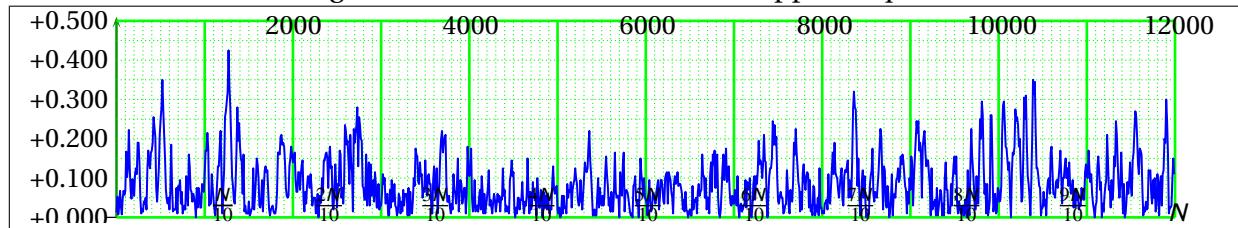
- (1) Suppose we have a length  $N \triangleq 12000$  die sequence  $(x_n)$  with the following distribution:

$$P(\square) = P(\square) = P(\square) = P(\square) = P(\square) = P(\square) = \frac{1}{6} \quad \text{for } n \in [0 : 3999] \cup [8000 : 11999]$$

$$P(\square) = P(\square) = P(\square) = P(\square) = P(\square) = P(\square) = \frac{1}{10} \text{ and } P(\square) = \frac{1}{2} \quad \text{for } n \in [4000 : 7999]$$

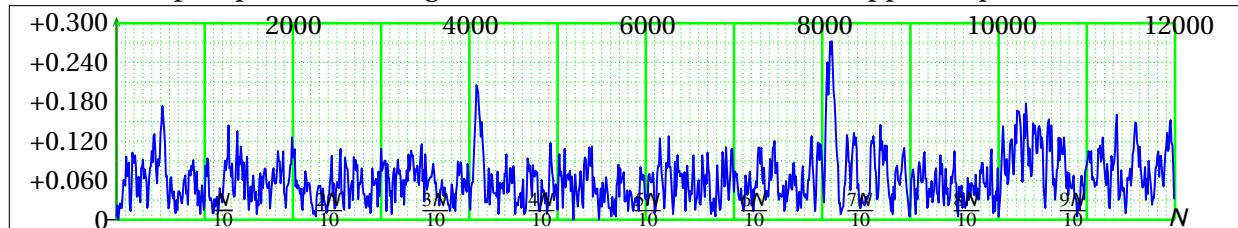
That is, the distribution of the sequence is uniformly distributed in the first and last thirds, but biased towards  $\square$  in the middle third. In this example we use a simple statistical edge detector to try to find the statistical “edges” at 4000 and 8000. The edge detector here is a *filter* operation  $\mathbf{W}$  (Definition 1.41 page 9) using a *length 200 Haar wavelet sequence* (Definition 1.50 page 10).<sup>89</sup>

- (2) Suppose we first use the *PAM die random variable* (Definition 3.16 page 21) to map the sequence of item (1) into  $\mathbb{R}^1$ . The magnitude of  $\mathbf{W} : \mathbb{R}^1 \rightarrow \mathbb{C}^1$  of the mapped sequence is as follows:<sup>90</sup>



We might expect to see strongest evidence of the edges at  $4000 + 200/2 = 4100$  and  $8100$ . But looking at the above result, this is not apparent. In fact, there are a total of 10646 values that are greater than or equal to the value at location 4100 (that value being 0.015).<sup>91</sup>

- (3) Suppose we next use the *QPSK die random variable* (Definition 3.17 page 21) to map the die sequence into the complex plane. The magnitude of  $\mathbf{W} : \mathbb{C}^1 \rightarrow \mathbb{C}^1$  of the mapped sequence is as follows:



Using this method, the edges are apparent. And the value of the peak at  $n = 4083$  (with value 0.223383) is about  $10 \log_{10}(0.223383/0.072741) \approx 4.9$  dB above the noise floor.<sup>92</sup> Here, the noise

<sup>88</sup> Greenhoe (2013) pages 27–62 (Chapter 2. The Structure of Wavelets)

<sup>89</sup> Empirical evidence due to Singh et al. (1997) suggests that the Haar wavelet performs better than several other common wavelets as an edge detector.

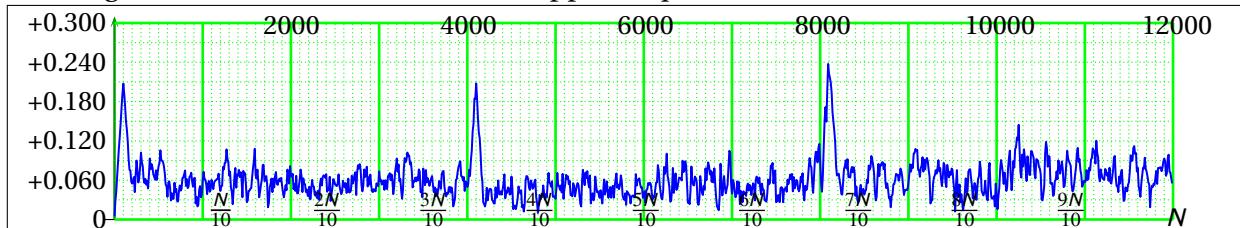
<sup>90</sup> Note that the plot in item (2) has been down sampled by a factor of 10 for practical reasons of displaying the very large data set.

<sup>91</sup> See experiment log file “diehaar\_12000m4000\_h200\_1050.xls” generated by the program “ssp.exe”.

<sup>92</sup> See experiment log file “diehaar\_12000m4000\_h200\_1050.xls” generated by the program “ssp.exe”.

value 0.072741 is the *RMS* (see item (2) of Example 4.11 page 39) of the DFT magnitude sequence computed over the domain  $n = 200 \dots N - 1$ .

- (4) Suppose we next use the  $\mathbb{R}^6$  *die random variable* (Definition 3.19 page 21) to map the sequence into  $\mathbb{R}^6$ . The magnitude of  $\mathbf{W} : \mathbb{R}^6 \rightarrow \mathbb{R}^6$  of the mapped sequence is as follows:



Using this method, the edges are also apparent. And the value of the peak at  $n = 4102$  (with value 0.209165) is about  $10 \log_{10}(0.209165/0.065768) \approx 5.0$  dB above the noise floor.<sup>93</sup> This is only a slight improvement over item (3).

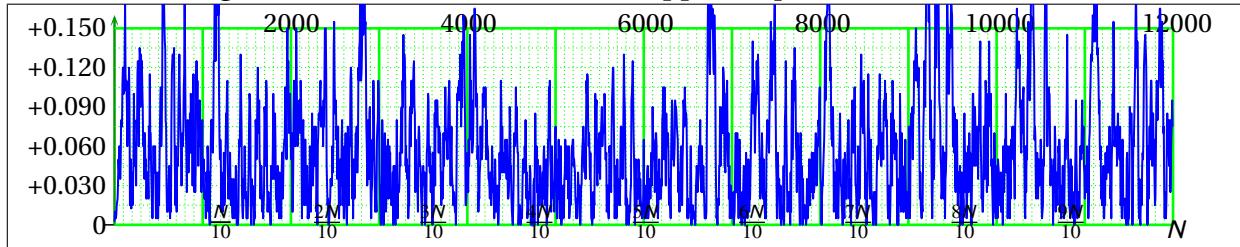
#### Example 4.13 (statistical edge detection using Haar wavelet on non-stationary artificial DNA sequence)

- (1) Suppose we have a length  $N \triangleq 12000$  dna sequence ( $x_n$ ) with the following distribution:

$$\begin{aligned} P(\square) &= P(\square) = P(\square) = P(\square) = \frac{1}{4} && \text{for } n \in [0 : 3999] \cup [8000 : 11999] \\ P(\square) &= P(\square) = P(\square) = \frac{27}{100} \text{ and } P(\square) = \frac{49}{100} && \text{for } n \in [4000 : 7999] \end{aligned}$$

That is, the distribution of the sequence is uniformly distributed in the first and last thirds, but biased towards  $\square$  in the middle third. Just as in Example 4.12, we again use a filter operation  $\mathbf{W}$  with *length 200 Haar wavelet sequence* as a simple statistical edge detector to try to locate the statistical “edges” at 4000 and 8000.

- (2) Suppose we first use the *PAM DNA random variable* (Definition 3.22 page 21) to map the DNA sequence into  $\mathbb{R}^1$ . The magnitude of  $\mathbf{W} : \mathbb{R}^1 \rightarrow \mathbb{C}^1$  of the mapped sequence is as follows:<sup>94</sup>



We might expect to see strongest evidence of the edges at or near  $4000 + 200/2 = 4100$  and  $8100$ . In fact, the sequence does have peaks at 4087 (with value 0.185) and at 8087 (with value 0.230). The peak at 4087 is about  $10 \log_{10}(0.185000/0.071355) \approx 4.1$  dB above the noise floor. However, there are 103 other values not around the  $n = 4087$  and  $n = 8087$  peaks that are 0.185 or greater. These 102 values represent roughly 11 other peaks, each of which could trigger a “false positive” decision.<sup>95</sup>

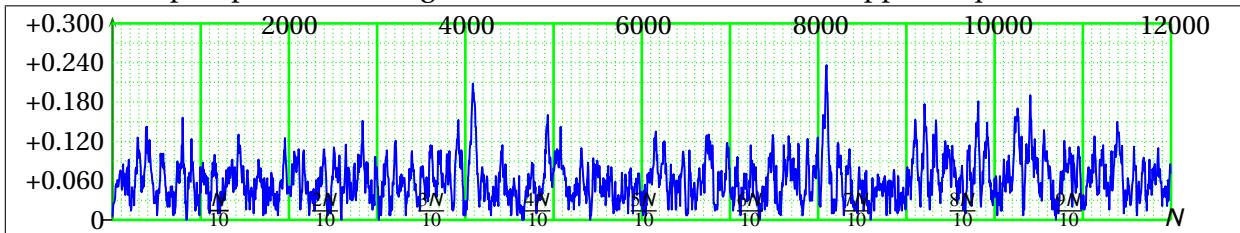
<sup>93</sup>Here the RMS noise value is computed over the domain  $n = 200 \dots N - 1$ .

<sup>94</sup>See experiment log file “diehaar\_12000m4000\_h200\_1050.xlg” generated by the program “ssp.exe”.

<sup>95</sup>Note that the plot in item (2) has been down sampled by a factor of 10 for practical reasons of displaying the very large data set.

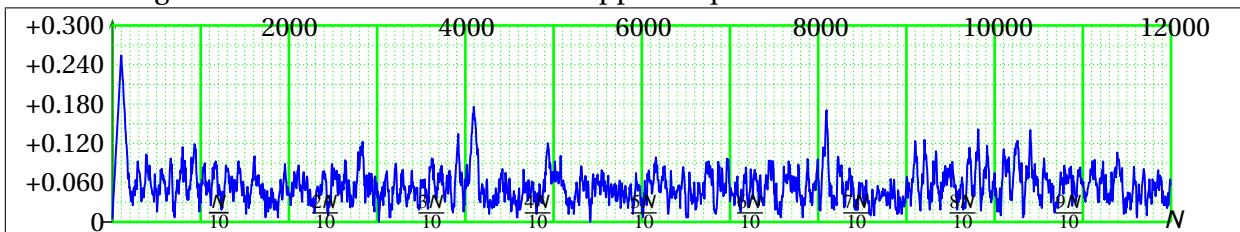
<sup>95</sup>See experiment log file “dnahaar\_12000m4000\_h200\_1749.xlg” generated by the program “ssp.exe”.

- (3) Suppose we next use the *QPSK DNA random variable* (Definition 3.17 page 21) to map the dna sequence into the complex plane. The magnitude of  $\mathbf{W} : \mathbb{C}^1 \rightarrow \mathbb{C}^1$  of the mapped sequence is as follows:



Using this method, the edges are apparent. And the value of the peak at  $n = 4086$  (with value 0.215870) is  $10 \log_{10}(0.215870/0.070068) \approx 4.9$  dB above the noise floor.<sup>96</sup>

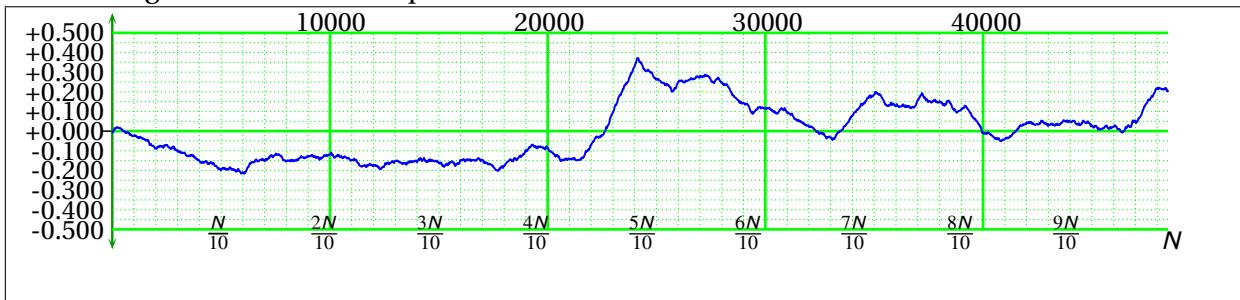
- (4) Suppose we next use the  $\mathbb{R}^4$  DNA random variable (Definition 3.24 page 22) to map the sequence into  $\mathbb{R}^4$ . The magnitude of  $\mathbf{W} : \mathbb{R}^4 \rightarrow \mathbb{C}^4$  of the mapped sequence is as follows:



Using this method, the edges are also apparent. And the value of the peak at  $n = 4096$  (with value 0.181246) is  $10 \log_{10}(0.181246/0.059594) \approx 4.8$  dB above the noise floor.<sup>97</sup> Note that this is a slight decrease in performance as compared to item (3).

#### Example 4.14 (Wavelet analysis of Phage Lambda DNA sequence)

- (1) Consider the Phage Lambda DNA sequence. It has a strong  $\text{AT}$  bias before  $n = 20000$  and a strong  $\text{CG}$  bias after,<sup>98</sup> as demonstrated next by filtering the DNA sequence with a *length 1600 Haar scaling sequence* (Definition 1.49 page 10)—such a filtering operation acts as a kind of “sliding window” histogram of the DNA sequence.

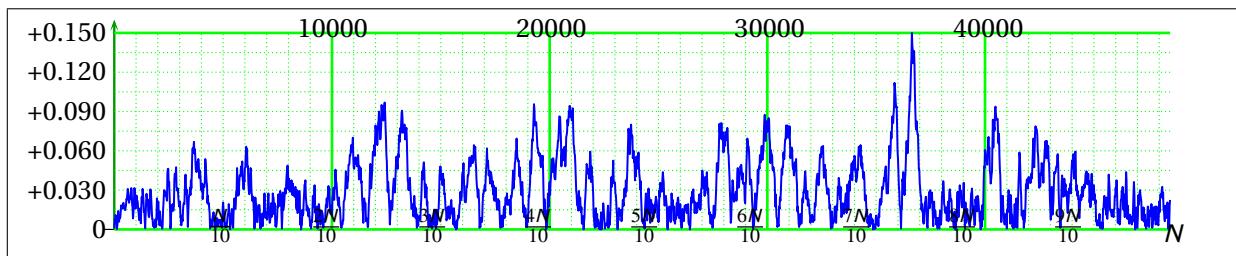


- (2) Suppose we first use the *PAM DNA random variable* (Definition 3.22 page 21) to map the DNA sequence into  $\mathbb{R}^1$ . The magnitude of the length 1600 Haar wavelet operation on the mapped sequence is as follows:

<sup>96</sup> See experiment log file “dnahaar\_12000m4000\_h200\_1749.xlg” generated by the program “ssp.exe”.

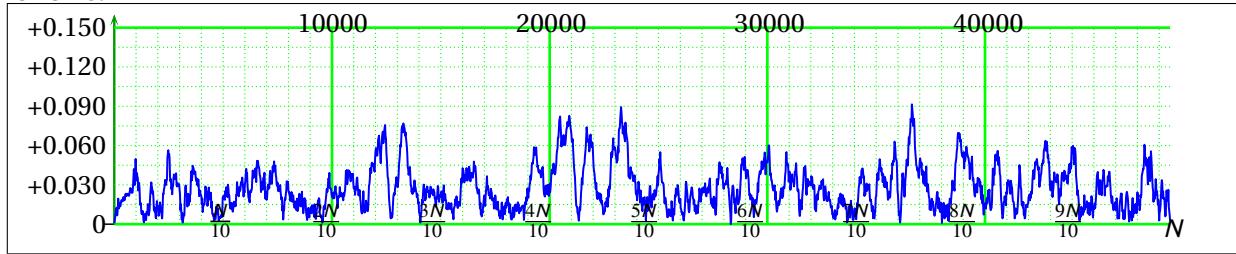
<sup>97</sup> See experiment log file “dnahaar\_12000m4000\_h200\_1749.xlg” generated by the program “ssp.exe”.

<sup>98</sup> Cristianini and Hahn (2007) page 14



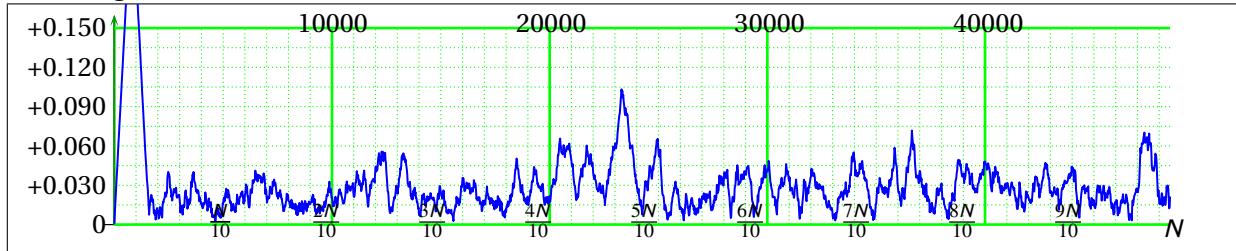
Note that it is very difficult to pick out the edge at 20000.

- (3) Suppose we next use the *QPSK DNA random variable* (Definition 3.23 page 22) to map the DNA sequence into the complex plane. The length 1600 Haar wavelet operation on the mapped sequence is as follows:



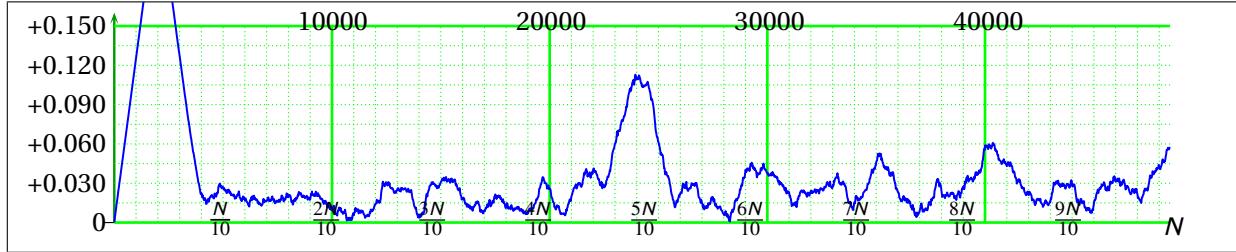
If one did not know apriori that there was an edge at 20000, it would still be difficult to identify.

- (4) Suppose we next use the  $\mathbb{R}^4$  *DNA random variable* (Definition 3.24 page 22) to map the DNA sequence into  $\mathbb{R}^4$ . Filtering the mapped sequence with a length 1600 Haar wavelet sequence results in the following:



Here there is a clear peak near 20000.

- (5) And here is the same analysis as used in item (4), but at scale 4000 (using a length 4000 Haar wavelet filter):



Again, the peak near 20000 is quite pronounced. However, at the low resolution scale (of 4000), it would be difficult to determine precisely where the statistical edge actually was.

## Appendix A Distance spaces

**Definition A.1** <sup>99</sup> A function  $d$  in the set  $\mathbb{R}^{X \times X}$  (Definition 1.8 page 4) is a **distance** if

1.  $d(x, y) \geq 0 \quad \forall x, y \in X$  (*non-negative*) and
2.  $d(x, y) = 0 \iff x = y \quad \forall x, y \in X$  (*nondegenerate*) and
3.  $d(x, y) = d(y, x) \quad \forall x, y \in X$  (*symmetric*)

The pair  $(X, d)$  is a **distance space** if  $d$  is a *distance* on a set  $X$ .

**Definition A.2** <sup>100</sup> Let  $(X, d)$  be a *distance space* and  $2^X$  be the *power set* of  $X$  (Definition 1.10 page 4). The **diameter** in  $(X, d)$  of a set  $A \in 2^X$  is  $\text{diam } A \triangleq \begin{cases} 0 & \text{for } A = \emptyset \\ \sup \{d(x, y) | x, y \in A\} & \text{otherwise} \end{cases}$

**Definition A.3** <sup>101</sup> Let  $(X, d)$  be a *distance space*. Let  $2^X$  be the *power set* (Definition 1.10 page 4) of  $X$ . A set  $A$  is **bounded** in  $(X, d)$  if  $A \in 2^X$  and  $\text{diam } A < \infty$ .

The next theorem lists five properties that do *not* in general hold in a *distance space*. Note that if a *distance space* is a *metric space*, then all five of the properties *do* hold.

**Theorem A.4** <sup>102</sup> Let  $(x_n)_{n \in \mathbb{Z}}$  be a *SEQUENCE* in a *DISTANCE SPACE*  $(X, d)$ . The *DISTANCE SPACE*  $(X, d)$  does not necessarily have all the nice properties that a *METRIC SPACE* (Definition B.1 page 46) has. In particular, note the following:

- |  |               |  |
|--|---------------|--|
| 1. $d$ is a <b>DISTANCE</b> in $(X, d)$                | $\Rightarrow$ | $d$ is <b>CONTINUOUS</b> in $(X, d)$       |
| 2. $B$ is an <b>OPEN BALL</b> in $(X, d)$              | $\Rightarrow$ | $B$ is <b>OPEN</b> in $(X, d)$             |
| 3. $B$ is the set of all <b>OPEN BALLS</b> in $(X, d)$ | $\Rightarrow$ | $B$ is a <b>BASE</b> for a topology on $X$ |
| 4. $(x_n)$ is <b>CONVERGENT</b> in $(X, d)$            | $\Rightarrow$ | <b>limit is UNIQUE</b>                     |
| 5. $(x_n)$ is <b>CONVERGENT</b> in $(X, d)$            | $\Rightarrow$ | $(x_n)$ is <b>CAUCHY</b> in $(X, d)$       |

**Definition A.5** <sup>103</sup> Let  $(X, d)$  be a *distance space* (Definition A.1 page 45). Let  $\mathbb{R}^+$  be the *set of positive real numbers* (Definition 1.2 page 3).

An **open ball** centered at  $x$  with radius  $r$  is the set  $B(x, r) \triangleq \{y \in X | d(x, y) < r\}$ .

A **closed ball** centered at  $x$  with radius  $r$  is the set  $\bar{B}(x, r) \triangleq \{y \in X | d(x, y) \leq r\}$ .

**Definition A.6** <sup>104</sup> Let  $(X, d)$  and  $(Y, p)$  be *distance spaces*.

The function  $f \in Y^X$  is an **isometry** on  $(Y, p)^{(X, d)}$  if

$$d(x, y) = p(f(x), f(y)) \quad \forall x, y \in X$$

The spaces  $(X, d)$  and  $(Y, p)$  are **isometric** if there exists an isometry on  $(Y, p)^{(X, d)}$ .

<sup>99</sup> Menger (1928) page 76 ('Abstand  $a$   $b$  definiert ist...') (distance from  $a$  to  $b$  is defined as...'), Wilson (1931) page 361 ('distance', 'semi-metric space'), Blumenthal (1938) page 38, Blumenthal (1953) page 7 ('DEFINITION 5.1. A distance space is called semimetric provided...'), Galvin and Shore (1984) page 67 ('distance function'), Laos (1998) page 118 ('distance space'), Khamsi and Kirk (2001) page 13 ('semimetric space'), Bessenyei and Pales (2014) page 2 ('semimetric space'), Deza and Deza (2014) page 3 ('distance (or dissimilarity)')

<sup>100</sup> in *metric space*: Hausdorff (1937), page 166, Copson (1968), page 23, Michel and Herget (1993), page 267, Molchanov (2005) page 389

<sup>101</sup> in *metric space*: Thron (1966), page 154 (definition 19.5), Bruckner et al. (1997) page 356

<sup>102</sup> Greenhoe (2015a), Heath (1961) page 810 ('THEOREM'), Galvin and Shore (1984) page 71 ('2.3 LEMMA')

<sup>103</sup> in *metric space*: Aliprantis and Burkinshaw (1998), page 35

<sup>104</sup> Thron (1966), page 153 (definition 19.4), Giles (1987) page 124 (Definition 6.22), Khamsi and Kirk (2001) page 15 (Definition 2.4), Kubrusly (2001) page 110



**Theorem A.7** <sup>105</sup> Let  $(X, d)$  and  $(Y, p)$  be METRIC SPACES. Let  $f$  be a function in  $Y^X$  and  $f^{-1}$  its inverse in  $X^Y$ .

$$\{f \text{ is an isometry on } (Y, p)^{(X, d)}\} \iff \{f^{-1} \text{ is an isometry on } (X, d)^{(Y, p)}\}$$

## Appendix B Metric spaces

**Definition B.1** <sup>106</sup> Let  $X$  be a set and  $\mathbb{R}^+$  the set of non-negative real numbers. Let  $d$  be a function in  $\mathbb{R}^{+^{X \times X}}$ . The pair  $(X, d)$  is a *metric space* and  $d$  is a **metric** on  $X$  if

1.  $d(x, y) \geq 0 \quad \forall x, y \in X \quad (\text{non-negative})$  and
2.  $d(x, y) = 0 \iff x = y \quad \forall x, y \in X \quad (\text{nondegenerate})$  and
3.  $d(x, y) = d(y, x) \quad \forall x, y \in X \quad (\text{symmetric})$  and
4.  $d(x, y) \leq d(x, z) + d(z, y) \quad \forall x, y, z \in X \quad (\text{subadditive/triangle inequality}).$ <sup>107</sup>

**Definition B.2** <sup>108</sup> Let  $X$  be a set and  $d \in \mathbb{R}^{X \times X}$ . The function  $d$  is the **discrete metric** on  $\mathbb{R}^{X \times X}$  if

$$d(x, y) \triangleq \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise} \end{cases} \quad \forall x, y \in X$$

**Definition B.3** Let  $X$  be a set and  $d \in \mathbb{R}^{X^N}$ . The function  $d$  is the **Euclidean metric** on  $\mathbb{R}^N$  if

$$d((x_1, x_2, \dots, x_N), (y_1, y_2, \dots, y_N)) \triangleq \sqrt{\sum_{n=1}^N (x_n - y_n)^2} \quad \forall (x_1, x_2, \dots, x_N), (y_1, y_2, \dots, y_N) \in \mathbb{R}^N$$

**Proposition B.4 (Fréchet product metric)** <sup>109</sup> Let  $X$  be a set.

$$\left\{ \begin{array}{l} 1. \{p_n\} \text{ are METRICS on } X \text{ and} \\ 2. \alpha_n \geq 0 \quad \forall n = 1, 2, \dots, N \text{ and} \\ 3. \max \{ \alpha_n | n=1,2,\dots,N \} > 0 \end{array} \right\} \implies \left\{ \begin{array}{l} d(x, y) = \sum_{n=1}^N \alpha_n p_n(x, y) \\ \text{is a METRIC on } X \end{array} \right\}$$

## Appendix C Lagrange arc distance

This paper makes use of a function introduced herein called the *Lagrange arc distance* (next definition). This function has been found useful for processing certain outcome subspaces. It is an extension of another function known as the *spherical metric* or *great circle metric*.<sup>110</sup> The *spherical metric* has domain on the surface of a sphere in  $\mathbb{R}^N$ . The *Lagrange arc distance* extends this idea to all of  $\mathbb{R}^N$  (not just the

<sup>105</sup> Thron (1966), page 153 (theorem 19.5)

<sup>106</sup> Dieudonné (1969), page 28, Copson (1968), page 21, Hausdorff (1937), page 109, Fréchet (1928),

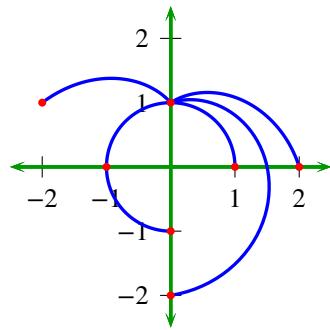
Fréchet (1906), page 30

<sup>107</sup> Euclid (circa 300BC) (Book I Proposition 20)

<sup>108</sup> Busemann (1955) page 4 (COMMENTS ON THE AXIOMS), Giles (1987), page 13, Copson (1968), page 24, Khamsi and Kirk (2001) page 19 (Example 2.1)

<sup>109</sup> Deza and Deza (2006) page 47, Deza and Deza (2009) page 84, Steen and Seebach (1978) pages 64–65 (Example 37.7), Isham (1999) page 10

<sup>110</sup> Ratcliffe (2013) pages 37–38 (The Spherical Metric), Deza and Deza (2014) page 123 (6.4 Non-Euclidean geometry), Deza and Deza (2006) page 73 (6.4 Non-Euclidean geometry)



$d((0, 1), (-1, 0)) = \frac{1}{2}$
$d((0, 1), (-1, 0)) = \frac{1}{2}$
$d((0, 1), (0, -1)) = 1$
$d((1, 0), (0, -1)) = \frac{1}{2}$
$d((1, 0), (-1, 0)) = 1$
$d((-1, 0), (0, -1)) = \frac{1}{2}$
$d((0, 1), (2, 0)) = 0.8167968 \dots$
$d((0, 1), (0, -2)) = 1.5346486 \dots$
$d((0, 1), (-2, 1)) = 0.6966032 \dots$

Figure 3: Lagrange arc distance examples in  $\mathbb{R}^2$ 

surface of the sphere). The *Lagrange arc distance*  $d(p, q)$  first draws an arc between the two points  $p$  and  $q$  using *Lagrange interpolation*, and then  $d(p, q)$  is simply the Euclidean “arc length” of this arc.

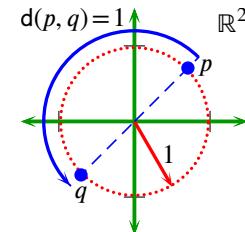
However, the extension does come at a cost—the *Lagrange arc distance* is not a *metric* (Definition B.1 page 46), but rather only a *distance* (Definition A.1 page 45, Theorem C.5 page 48). For more details about the impact of this cost, see Theorem A.4 (page 45).

**Definition C.1** Let  $p \triangleq (p_1, p_2, \dots, p_N)$  and  $q \triangleq (q_1, q_2, \dots, q_N)$  be two points in the *linear space*  $\mathbb{R}^N$ . Let  $P$  be a plane in  $\mathbb{R}^N$  that intersects the origin,  $p$  and  $q$ . Let  $(r_p, \theta_p)$  and  $(r_q, \theta_q)$  be the *polar coordinates* of  $p$  and  $q$ , respectively, in the plane  $P$ , and where the origin in  $P$  coincides with the origin in  $\mathbb{R}^N$ . Let  $A$  be an arc in  $P$  with polar equation  $(r(\theta), \theta)$  where  $\theta \in [\theta_p : \theta_q]$  and

$$r(\theta) \triangleq r_p \frac{\theta - \theta_q}{\theta_p - \theta_q} + r_q \frac{\theta - \theta_p}{\theta_q - \theta_p} \quad (\text{Lagrange interpolation}) .$$

Let  $R(p, q)$  be the Euclidean length of arc  $A$  in the plane  $P$ .<sup>111</sup>

$$d(p, q) = \begin{cases} \frac{1}{\pi} r_q & \text{if } p = (0, 0) \\ \frac{1}{\pi} r_p & \text{if } q = (0, 0) \\ \frac{1}{\pi} |r_p - r_q| & \text{if } \theta_p = \theta_q \\ \frac{1}{\pi} R(p, q) & \text{otherwise} \end{cases}$$



**Proposition C.2** Let  $R(p, q)$  be as defined in Definition C.1. If  $r_p \neq 0$ ,  $r_q \neq 0$  and  $\theta_p \neq \theta_q$  then

$$R(p, q) = \left| \frac{1}{\theta_p - \theta_q} \right| \left[ \frac{b + 2a\theta_q}{4a} \sqrt{a\theta_q^2 + b\theta_q + c} + \frac{4ac - b^2}{8a^{3/2}} \ln \left| 2a\theta_q + b + 2\sqrt{a(a\theta_q^2 + b\theta_q + c)} \right| \right] - \left| \frac{1}{\theta_p - \theta_q} \right| \left[ \frac{b + 2a\theta_p}{4a} \sqrt{a\theta_p^2 + b\theta_p + c} + \frac{4ac - b^2}{8a^{3/2}} \ln \left| 2a\theta_p + b + 2\sqrt{a(a\theta_p^2 + b\theta_p + c)} \right| \right]$$

$$\begin{aligned} \text{where } a &\triangleq (r_p - r_q)^2 \\ b &\triangleq 2(r_p - r_q)(r_q\theta_p - r_p\theta_q) \\ c &\triangleq (r_q\theta_p - r_p\theta_q)^2 + (r_p - r_q)^2 \end{aligned}$$

**Example C.3** (Lagrange arc distance in  $\mathbb{R}^2$ ) Figure 3 (page 47) illustrates the *Lagrange arc distance* on some pairs of points in  $\mathbb{R}^2$ .

<sup>111</sup>Note that path plotted with the polar coordinates  $(r(\theta), \theta)$  with  $\theta_p \leq \theta \leq \theta_q$  is an *arc* in  $P$  from  $p$  to  $q$ .

**Example C.4** (Lagrange arc distance in  $\mathbb{R}^3$ ) Here are some examples of the Lagrange arc distance in  $\mathbb{R}^3$ :

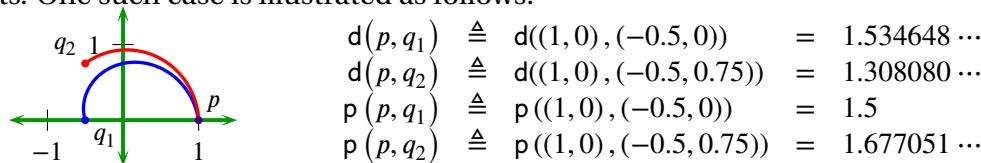
$d((0, 1, 0), (-1, 0, 0))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 1, 0), (0, 0, 1))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 1, 0), (0, 0, -1))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 1, 0), (-1, 0, 0))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 1, 0), (0, -1, 0))$	=	2	$\theta$	=	$\pi$	=	$180^\circ$
$d((1, 0, 0), (0, 0, 1))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((1, 0, 0), (0, 0, -1))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((1, 0, 0), (-1, 0, 0))$	=	2	$\theta$	=	$\pi$	=	$180^\circ$
$d((1, 0, 0), (0, -1, 0))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 0, 1), (0, 0, -1))$	=	2	$\theta$	=	$\pi$	=	$180^\circ$
$d((0, 0, 1), (-1, 0, 0))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 0, 1), (0, -1, 0))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 0, -1), (-1, 0, 0))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 0, -1), (0, -1, 0))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((-1, 0, 0), (0, -1, 0))$	=	1	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 1, 0), (2, 0, 0))$	$\approx$	1.633594	$\theta$	=	$\pi/2$	=	$90^\circ$
$d((0, 1, 0), (0, -2, 0))$	$\approx$	3.069297	$\theta$	=	$\pi$	=	$180^\circ$
$d((0, 1, 0), (-2, 1, 0))$	$\approx$	1.393206	$\theta$	$\approx$	1.107	$\approx$	63°
$d((0, 1, 0), (-1, 0, -1))$	$\approx$	1.235842	$\theta$	=	$\pi$	=	$90^\circ$
$d((1, 1, 1), (-\frac{1}{2}, \frac{1}{4}, -2))$	$\approx$	2.732537	$\theta$	$\approx$	2.2466	$\approx$	128.72°

The *Lagrange arc distance* is *not* a metric because in general the *triangle inequality* property does not hold (next theorem). Furthermore, the *Lagrange arc distance* does not induce a *norm* because it is *not translation invariant* (the *translation invariant* property is a necessary condition for a *metric* to induce a *norm*) and balls in a *Lagrange arc distance space* are in general *not convex* (balls are always *convex* in a *normed linear space*).

**Theorem C.5** In the LAGRANGE ARC DISTANCE SPACE  $(X, d)$  over a field  $\mathbb{F}$

- (1).  $d(p, r) \leq d(p, q) + d(q, r) \quad \forall p, q, r \in X$  (TRIANGLE INEQUALITY FAILS)
- (2).  $d(p+r, q+r) \neq d(p, q) \quad \forall p, q, r \in X$  (NOT TRANSLATION INVARIANT)
- (3).  $d(\alpha p, \alpha q) = |\alpha| d(p, q) \quad \forall p, q, r \in X, \alpha \in \mathbb{R}$  (HOMOGENEOUS)
- (4).  $d$  is CONTINUOUS
- (5).  $d$  does not induce a norm
- (6). balls in  $(X, d)$  are in general NOT CONVEX

**Remark C.6** (Lagrange arc distance versus Euclidean metric) The *Lagrange arc distance*  $d$  and *Euclidean metric*  $p$  are similar in the sense that they often lead to the same results in determining which of the two points  $q_1$  or  $q_2$  is “closer” to a point  $p$ . But in some cases the two metrics lead to two different results. One such case is illustrated as follows:



That is,  $q_2$  is closer than  $q_1$  to  $p$  with respect to the *Lagrange arc distance*, but  $q_1$  is closer than  $q_2$  to  $p$  with respect to the *Euclidean metric*.

## Appendix D C++ source code support

This paper seeks to conform to the principles of *Reproducible Research* as detailed at

<http://reproducibleresearch.net/>.

This section contains a partial C++ source code listing for `ssp.exe`, written by the author of this paper, which produced the `TEX` files used to generate the 128 or so data plot files presented in this paper. The complete and downloadable source code for `ssp.exe` is to accompany the online version of this paper.

### D.1 Symbolic sequence routines

```
1  /*=====
2   * Daniel J. Greenhoe
3   * header file for routines for char sequence functions
4   *=====
5  class symseq {
6  private:
7   long N;
8   char *x;
9  public:
10  symseq(const long M); //constructor initializing to '.'
11  symseq(const long M,const unsigned seed,const char *symbols); //constructor initializing
12   using seed
13  void clear(void); // fill sequence with the value 'A'
14  char get(const long n); //get a value from x at location n
15  char get(const long n,char *symbols); //get a value from x at location n but exit if not in
16   <symbols> string
17  void put(long n, char symbol);
18  void put(const long start, const long end, const char symbol);
19  const long getN(void){const long M=(const long)N; return M;} //get N
20  void downsample(int M, symseq *y); //downsample by a factor of <M> and write to <y>
21  void list(const long start, const long end, const char* str1, const char *str2, const int
22   display, FILE *fptr);
23  void list(const long start, const long end, FILE *fptr){list(start,end,"","","",1,fptr);}
24  void list(const long start, const long end, const int display, FILE *fptr){list(start,end,
25   "", "", 1,fptr);}
26  void list(const long start, const long end, const char* str1, const char *str2, FILE *fptr){
27   list(start, end, str1, str2, 1, fptr);
28 }
29  void list(long start, long end){list(start,end,"","","",NULL);} //list contents of sequence
30  void list(void){list(0, N-1,"","","",NULL);} //list contents of sequence
31  void list(long start){list(start,N-1,"","","",NULL);} //list contents of sequence
32  void shiftL(long n); //shift symseq n elements to the left
33  void shiftR(long n); //shift symseq n elements to the right
34  void prngseed(unsigned seed){rand(seed);} //
35  void randomize(const char *symbols); //randomize using a list of symbols
36  void operator=(symseq y); //x=y
37  void operator>=(long n){shiftR(n);} //shift symseq n elements to the right
38  void operator<=(long n){shiftL(n);} //shift symseq n elements to the left
39  int operator==(symseq y); //test if x==y; 1 if yes, 0 if no
40 }

extern long cmp(const symseq *x,const symseq *y, int showdiff, FILE *fptr);
void copy(const long start, const long end, const symseq *x, const symseq *y);
```

```
41 void downsample(int M, symseq *x, symseq *y);  
  
1 /*=====  
2 * Daniel J. Greenhoe  
3 * routines for Real Die symseqs  
4 *=====*/  
5 /*=====  
6 * headers  
7 *=====*/  
8 #include<stdio.h>  
9 #include<stdlib.h>  
10 #include<string.h>  
11 #include<math.h>  
12 #include<symseq.h>  
13  
14 /*  
15 * constructor initializing symseq to '.'  
16 */  
17 symseq::symseq(long M){  
18     long n;  
19     void *memptr;  
20     N=M;  
21     memptr=malloc(N*sizeof(char));  
22     if(memptr==NULL){  
23         fprintf(stderr,"symseq::symseq memory allocation for %ld elements failed\n",M);  
24         exit(EXIT_FAILURE);  
25     }  
26     x = (char *)memptr;  
27     for(n=0; n<N; n++)x[n]='.';  
28 }  
29  
30 /*  
31 * constructor initializing symseq to '.'  
32 */  
33 symseq::symseq(const long M,const unsigned seed,const char *symbols){  
34     long n;  
35     void *memptr;  
36     N=M;  
37     memptr=malloc(N*sizeof(char));  
38     if(memptr==NULL){  
39         fprintf(stderr,"symseq::symseq memory allocation for %ld elements failed\n",M);  
40         exit(EXIT_FAILURE);  
41     }  
42     x = (char *)memptr;  
43     randomize(seed,symbols);  
44 }  
45  
46 /*  
47 * fill the symseq with the value '.'  
48 */  
49 void symseq::clear(void){  
50     long n;  
51     for(n=0; n<N; n++)x[n]='.';  
52 }  
53  
54 /*  
55 * get a symbol from the symseq x at location n  
56 */  
57 char symseq::get(const long n){  
58     if(n<0 || n>N){ //domain check  
59         fprintf(stderr,"ERROR using symseq::get(n): n=%ld outside the domain [0:%ld] of the  
60             sequence.\n",n,N);  
61         exit(EXIT_FAILURE);  
62     }  
63     return x[n];
```

```
63     }
64
65     /*
66      * get a symbol from the symseq x at location n
67      * but exit if symbol is not in the string <range>
68      * example: symbol=get(n,"ABCDEF");
69      */
70 char symseq::get(const long n, char *range) {
71     int M;
72     int match;
73     char *sptr;
74     char symbol;
75     if(n<0 || n>=N) { //domain check
76         fprintf(stderr ,"\nERROR using symseq::get(n): n=%ld outside the domain [0:%ld] of the
77             sequence.\n",n,N);
78         exit(EXIT_FAILURE);
79     }
80     symbol = x[n];
81     for(match=0,sptr=range;* sptr!= '\0';sptr++) if(symbol==* sptr) match=1;
82     if (!match) { //range check
83         fprintf(stderr ,"\nERROR using symbol=symseq::get(%ld): symbol='c' (0x%02X) outside the range
84             {%s} of the sequence.\n",n,symbol,symbol,range);
85         exit(EXIT_FAILURE);
86     }
87     return symbol;
88 }
89
90 /*
91  * put a single value from the symseq x at location n
92  */
93 void symseq::put(long n, char symbol) {
94     int rval;
95     if(n<0 || n>=N) { //domain check
96         fprintf(stderr , "ERROR using symseq::put(n): n=%ld outside the domain [0:%ld] of the
97             sequence.\n",n,N-1);
98         exit(EXIT_FAILURE);
99     }
100    x[n]=symbol;
101
102 /*
103  * put the symbol <symbol> into the sequence <x>
104  * from location <start> to location <end>
105  */
106 void symseq::put(const long start, const long end, const char symbol){
107     long n;
108     for(n=start;n<=end;n++) put(n,symbol);
109
110 /*
111  * list contents of dieseq
112  */
113 void symseq::list(const long start, const long end, const char *str1, const char *str2, const int
114     display, FILE *fptr){
115     long n,m;
116     if(strlen(str1)>0){
117         if(display) printf("%s",str1);
118         if(fptr!=NULL) fprintf(fptr,"%s",str1);
119     }
120     for(n=start ,m=1; n<=end; n++,m++){
121         if(display) printf("%c",get(n));
122         if(m%50==0&&display) printf("\n");
123         else if(m%10==0&&display) printf(" ");
124         if(fptr!=NULL){
125             fprintf(fptr ,"%c",get(n));
```

```
124     if (m%50==0) fprintf(fptr ,"\n");
125     else if (m%10==0) fprintf(fptr , " ");
126   }
127 }
128 if (strlen(str2)>0){
129   if(display) printf("%s",str2);
130   if(fptr!=NULL) fprintf(fptr,"%s",str2);
131 }
132 }
133 */
134 /* downsample sequence by a factor of <M>
135 // * and write to sequence pointed to by <y>
136 // *
137 */
138 void symseq::downsample(int M, symseq *y){
139   const long Ny = y->getN();
140   long n,m;
141   if (M<1) { // check validity of factor <M>
142     fprintf(stderr,"ERROR using symseq::downsample(M,y): factor=%d must be at least 1\n",M);
143     exit(EXIT_FAILURE);
144   }
145   if (Ny != N/M) { // check validity of the length of output sequence <y>
146     fprintf(stderr,"ERROR using symseq::downsample(M,y): length %ld of output sequence y must
147       be N/M = %ld/%ld = %ld\n",Ny,N,M,N/M);
148     exit(EXIT_FAILURE);
149   }
150   for (n=0,m=0; m<Ny; n+=M,m++) y->put(m,x[n]);
151 }
152 /*
153 * shift symseq n elements to the right inserting zeros on the left
154 * example: if x = [ a b c d e f ] (N=6), then shiftR(2) results in
155 *           x = [ 0 0 a b c d ] (N=6).
156 */
157 void symseq::shiftR(long n){
158   long m;
159   for(m=N-1;m-n>=0;m--) x[m]=x[m-n];
160   for(m=0;m<n;m++) x[m]='.';
161 }
162 /*
163 * shift symseq n elements to the left inserting zeros on the right
164 * example: if x = [ a b c d e f ] (N=6), then shiftL(2) results in
165 *           x = [ c d e f 0 0 ] (N=6).
166 */
167 void symseq::shiftL(long n){
168   long m;
169   for(m=0;m<N-n;m++) x[m]=x[m+n];
170   for(m=N-2;m<N;m++) x[m]='.';
171 }
172 /*
173 * fill the sequence with uniformly distributed pseudo-random symbols
174 * from the string <symbols>
175 */
176 void symseq::randomize(const char *symbols){
177   const long N=getN();
178   const int M=strlen(symbols);
179   int r,i;
180   long n;
181   for(n=0; n<N; n++){
182     r=rand();
183     i = r%M;
184     put(n,symbols[i]);
185   }
186 }
187 }
```

```
188     }
189
190     /*=====
191     * operators
192     *=====*/
193
194     /*
195     * operator symseq x = symseq y
196     */
197
198     void symseq::operator=(symseq y) {
199         const long M=y.getN();
200         long n;
201         if (N!=M) {
202             fprintf(stderr,"ERROR using symseq x = symseq y operation: size of x (%ld) does not equal
203                     size of y (%ld)\n",N,M);
204             exit(EXIT_FAILURE);
205         }
206         for (n=0;n<N;n++)x[n]=y.get(n);
207
208     /*
209     * operator symseq x == symseq y
210     * compare x and y; return 1 if the same, 0 if different.
211     */
212
213     int symseq::operator==(symseq y) {
214         const long M=y.getN();
215         long n;
216         int retval;
217         char xsym,ysym;
218
219         if (N!=M) {
220             fprintf(stderr,"ERROR using symseq x == symseq y operation: size of x (%ld) does not equal
221                     size of y (%ld)\n",N,M);
222             exit(EXIT_FAILURE);
223         }
224         for (n=0,retval=1;n<N;n++) {
225             xsym= get(n);
226             ysym=y.get(n);
227             if (xsym!=ysym)retval=0;
228         }
229         return retval;
230     }
231
232     /*=====
233     * external functions
234     *=====*/
235
236     /*
237     * compare dieseq x and dieseq y
238     * return the number of locations in which the two sequences are different
239     * return 0 if the same
240     */
241
242     long cmp(const symseq *x, const symseq *y, int showdiff, FILE *fptr) {
243         const long N=x->getN();
244         const long M=y->getN();
245         char xsym,ysym;
246         long n;
247         long count;
248         if (N!=M) {
249             fprintf(stderr,"\nERROR using cmp(symseq *x,symseq *y): size of x (%ld) != size of y
250                     (%ld).\n",N,M);
251             exit(EXIT_FAILURE);
252         }
253         for (n=0,count=0;n<N;n++) {
254             xsym=x->get(n);
255             ysym=y->get(n);
256             if (xsym!=ysym) {
```

```

250     count++;
251     if (showdiff) fprintf(stdout, "%ld: x[%ld]=%c(0x%02x)
252                     y[%ld]=%c(0x%02x)\n", count, n, xsym, xsym, n, ysym, ysym);
253     if (fptr!=NULL) fprintf(fptr, "%ld: x[%ld]=%c(0x%02x)
254                     y[%ld]=%c(0x%02x)\n", count, n, xsym, xsym, n, ysym, ysym);
255   }
256 }
257
258 /*
259 * copy the sequence <x> = [x_start ... x_end]
260 * into the sequence <y> = [y_0      ... y_{N-1}]
261 * where N = end-start+1
262 */
263 void copy(const long start, const long end, const symseq *x, const symseq *y){
264   long Nx=x->getN();
265   long Ny=y->getN();
266   long n,m;
267   double xx;
268   y->clear();
269   if (end>=Nx){
270     fprintf(stderr,"ERROR using copy(start,end,seqR1 *x, seqR1 *y): <end>=%ld is too
271           large.\n",end);
272     exit(EXIT_FAILURE);
273   }
274   if ((end-start+1)!=Ny){
275     fprintf(stderr,"ERROR using copy(start,seqR1 *x, seqR1 *y): length of [x_start...x_end] (%ld)
276           != length of y(%ld).\n",Nx-start,Ny);
277     exit(EXIT_FAILURE);
278   }
279   for(n=start ,m=0;n<=end;n++,m++){
280     xx=x->get(n);
281     y->put(m,xx);
282   }
283
284 /*
285 * downsample sequence by a factor of <M>
286 * and write to sequence pointed to by <y>
287 */
288 void downsample(int M, symseq *x, symseq *y){
289   const long Nx = x->getN();
290   const long Ny = y->getN();
291   char symbol;
292   long n,m;
293   if (M<1){ //check validity of factor <M>
294     fprintf(stderr,"ERROR using symseq::downsample(M,y): factor=%d must be at least 1\n",M);
295     exit(EXIT_FAILURE);
296   }
297   if (Ny != Nx/M){ //check validity of the length of output sequence <y>
298     fprintf(stderr,"ERROR using symseq::downsample(M,y): length %ld of output sequence y must be
299           N/M = %ld/%ld = %ld\n",Ny,Nx,M,Nx/M);
300     exit(EXIT_FAILURE);
301   }
302   for(n=0,m=0; m<Ny; n+=M,m++){
303     symbol = x->get(n);
304     y->put(m,symbol);
305   }
306 }
```

## D.2 Die routines

```

1  /*=====
2   * Daniel J. Greenhoe
3   * header file for routines for die routines
4   * 'A'--> die face value 1
5   * 'B'--> die face value 2
6   * 'C'--> die face value 3
7   * 'D'--> die face value 4
8   * 'E'--> die face value 5
9   * 'F'--> die face value 6
10  =====*/
11 class dieseq: public symseq {
12 public:
13     dieseq(const long M) : symseq(M) {} //constructor initializing to '.'
14     dieseq(const long M, const unsigned seed) : symseq(M, seed, "ABCDEF") {} //constructor
15     //initializing random values
16     void randomize(void){symseq::randomize("ABCDEF");} ///
17     void randomize(unsigned seed){srand(seed); randomize();}
18     int randomize(long start, long end, int wA, int wB, int wC, int wD, int wE, int wF);
19     int randomize(unsigned seed, int wA, int wB, int wC, int wD, int wE, int wF){srand(seed); return
20         randomize(0, getN() - 1, wA, wB, wC, wD, wE, wF);}
21     int randomize(int wA, int wB, int wC, int wD, int wE, int wF){return
22         randomize(0, getN() - 1, wA, wB, wC, wD, wE, wF);}
23     int randomize(long start, long end, unsigned seed, int wA, int wB, int wC, int wD, int wE, int
24         wF){srand(seed); return randomize(start, end, wA, wB, wC, wD, wE, wF);}
25     char get(long n){return symseq::get(n, "ABCDEF");} //get a value from x at location n
26     void put(long n, char symbol){symseq::put(n, symbol);}
27     seqR1 dietoR1(void); //map die face values to R^1
28     seqC1 dietoC1(void); //map die face values to R^1
29     seqR1 dietoR1pam(void); //map die face values to R^1 using PAM scheme (symmetric about
30         zero)
31     seqR3 dietoR3(void); //map die face values to R^3
32     seqR1 histogram(const long start, const long end, int display, FILE *fptr); //compute,
33         display, and write histogram
34     seqR1 histogram(){return histogram(0, getN() - 1, 0, NULL);} //compute histogram
35     seqR1 histogram(const long start, const long end){return
36         histogram(start, end, 0, NULL);} //compute histogram
37     seqR1 histogram(int display, FILE *fptr){return histogram(0, getN() - 1, 1, fptr);} //print
38         histogram to file
39     seqR1 histogram(FILE *fptr){return histogram(0, getN() - 1, 0, fptr);} //print histogram to file
40     void operator=(diesequeq y); //x=y
41 };
42
43 extern int die_domain(char c); //check if value is in the domain of die
44 extern double die_dietoR1 (char c);
45 extern double die_dietoR1pam(char c);
46 extern vectR3 die_dietoR3 (char c);
47 extern vectR6 die_dietoR6 (char c);
48 extern complex die_dietoClc(char c);

```

```

1  /*=====
2   * Daniel J. Greenhoe
3   * routines for Real Die dieseqs
4   *=====
5   * headers
6   *=====
7   */
8 #include<stdio.h>
9 #include<stdlib.h>
10 #include<string.h>
11 #include<math.h>
12 #include<main.h>
13 #include<symseq.h>

```

```
14 #include<r1.h>
15 #include<r2.h>
16 #include<r3.h>
17 #include<r6.h>
18 #include<c1.h>
19 #include<die.h>
20
21 /*=====
22 * prototypes
23 =====*/
24 void phistogram(seqR1 *data, const long start, const long end, FILE *ptr);
25
26 /*
27 * fill the dieseq with weighted pseudo-random die face values
28 */
29 int dieseq::randomize(long start, long end, int wA, int wB, int wC, int wD, int wE, int wF) {
30     int r,u;
31     long n;
32     char symbol;
33     int sum=wA+wB+wC+wD+wE+wF;
34     if(sum!=100) {
35         fprintf(stderr, "dieseque::randomize error: sum of weight values = %d != 100\n",sum);
36         return -1;
37     }
38     //printf(" start=%ld end=%ld weights=(%03d %03d %03d %03d %03d %03d)\n",
39     //       start,end,wA,wB,wC,wD,wE,wF);
39     for(n=start; n<=end; n++) {
40         r=rand();
41         u = r%100;
42         if (u<wA) symbol='A';
43         else if (u<wA+wB) symbol='B';
44         else if (u<wA+wB+wC) symbol='C';
45         else if (u<wA+wB+wC+wD) symbol='D';
46         else if (u<wA+wB+wC+wD+wE) symbol='E';
47         else symbol='F';
48         put(n,symbol);
49     }
50     return 0;
51 }
52
53 /*
54 * map die face values to R^1
55 * A-->1 B-->2 C-->3 D-->4 E-->5 F-->6
56 * all other values --> 0
57 */
58 seqR1 dieseq::dietoR1(void) {
59     const long N=getN();
60     long n;
61     char sym;
62     double xR1;
63     seqR1 y(N);
64     for(n=0; n<N; n++) {
65         sym = get(n);
66         xR1 = die_dietoR1(sym);
67         y.put(n,xR1);
68     }
69     return y;
70 }
71
72 /*
73 * map die face values to R^1 using PAM scheme
74 * A-->-2.5 B-->-1.5 C-->-0.5 D-->0.5 E-->1.5 F-->2.5
75 * all other values --> 0
76 */
76 seqR1 dieseq::dietoR1pam(void) {
```

```
78 const long N=getN();
79 long n;
80 char sym;
81 double xR1;
82 seqR1 y(N);
83 for(n=0; n<N; n++){
84     sym = get(n);
85     xR1 = die_dietoR1pam(sym);
86     y.put(n,xR1);
87 }
88 return y;
89 }

91 /*
92 * map die face values to C^1
93 */
94 seqC1 dieseq::dietoC1(void){
95     const long N=getN();
96     long n;
97     char sym;
98     complex xC1;
99     seqC1 y(N);
100    for(n=0; n<N; n++){
101        sym = get(n);
102        xC1 = die_dietoC1c(sym);
103        y.put(n,xC1);
104    }
105    return y;
106 }

108 /*
109 * map die face values to R^3 sequence
110 */
111 seqR3 dieseq::dietoR3(void){
112     const long N=getN();
113     long n;
114     char sym;
115     vectR3 xR3;
116     seqR3 y(N);
117     for(n=0; n<N; n++){
118         sym = get(n);
119         xR3 = die_dietoR3(sym);
120         y.put(n,xR3);
121     }
122     return y;
123 }

125 /*
126 * compute histogram of dna sequence
127 * return seqR1 y of length 6 where
128 * y[1]-->number of dna 'A' symbols,
129 * y[2]-->number of dna 'B' symbols,
130 * y[3]-->number of dna 'C' symbols,
131 * y[4]-->number of dna 'D' symbols,
132 * y[5]-->number of dna 'D' symbols,
133 * y[6]-->number of dna 'D' symbols,
134 * y[0]-->number of all other values
135 * y[7]-->total number of symbols y[1],y[2],...,y[6]
136 */
137 seqR1 dieseq::histogram(const long start, const long end, int display, FILE *fptr){
138     seqR1 data(8);
139     long n;
140     long bin;
141     double p;
142     int i;
```

```
143 char symbol;
144 FILE *ptr;
145 data.clear();
146 for(n=start;n<=end;n++){
147     symbol=get(n);
148     switch(symbol){
149         case 'A': bin=1; break;
150         case 'B': bin=2; break;
151         case 'C': bin=3; break;
152         case 'D': bin=4; break;
153         case 'E': bin=5; break;
154         case 'F': bin=6; break;
155         default : bin=0; break;
156     }
157     if(bin!=0) data.increment(7);
158     data.increment(bin);
159 }
160 if(display) phistogram(&data,start,end,stdout);
161 if(fptr!=NULL)phistogram(&data,start,end,fptr );
162 return data;
163 }
164
165 /*
166 * print die sequence histogram with data pointed to by <data>
167 * to stream pointed to by ptr
168 */
169 void phistogram(seqRl *data, const long start, const long end, FILE *ptr){
170     const long N=end-start+1;
171     long bin;
172     fprintf(ptr, "\n");
173     fprintf(ptr, "-----\n");
174     fprintf(ptr,"| Histogram for sequence [x_n|n=%7ld-%7ld] (length %7ld)\n",
175             |\n|,start,end,N);
176     for(bin=1;bin<=6;bin++)fprintf(ptr, "%c", 'A'+(char)bin-1);
177     fprintf(ptr, " extra |\n|");
178     for(bin=1;bin<=6;bin++)fprintf(ptr, "%10.0lf",data->get(bin));
179     fprintf(ptr, "%10.0lf |\n|",data[0]);
180     for(bin=1;bin<=6;bin++)fprintf(ptr, "(%6.2lf%%)",data->get(bin)/(double)N*100.0);
181     fprintf(ptr, "(%6.2lf%%) |\n|",data->get(0)/(double)N*100.0);
182     fprintf(ptr, "-----\n");
183
184 /*
185 * =====
186 * operators
187 * ===== */
188 /*
189 * operator dieseq x = dieseq y
190 */
191 void dieseq::operator=(diesequeq y){
192     const long N= getN();
193     const long M=y.getN();
194     long n;
195     char symbol;
196     if(N!=M){
197         printf(stderr,"ERROR using dieseq x = dieseq y operation: size of x (%ld) does not equal
198             size of y (%ld)\n",N,M);
199         exit(EXIT_FAILURE);
200     }
201     for(n=0;n<N;n++){
202         symbol = y.get(n);
203         put(n,symbol);
204     }
205 }
```

```
206 /*=====
207 * external operations
208 =====*/
209 /*
210 * map die face values to R^1
211 */
212 double die_dietoR1(char c){
213     double rval;
214     switch(c){
215         case 'A': rval = 1.0; break;
216         case 'B': rval = 2.0; break;
217         case 'C': rval = 3.0; break;
218         case 'D': rval = 4.0; break;
219         case 'E': rval = 5.0; break;
220         case 'F': rval = 6.0; break;
221         default:
222             fprintf(stderr,"ERROR using die_dietoR1(c): c=%c(0x%xx) is not in the valid domain
223                 {A,B,C,D,E,F}\n",c,c);
224             exit(EXIT_FAILURE);
225     }
226     return rval;
227 }
228 /*
229 * map die face values to R^1 PAM
230 */
231 double die_dietoR1pam(char c){
232     double rval;
233     switch(c){
234         case 'A': rval = -2.5; break;
235         case 'B': rval = -1.5; break;
236         case 'C': rval = -0.5; break;
237         case 'D': rval = 0.5; break;
238         case 'E': rval = 1.5; break;
239         case 'F': rval = 2.5; break;
240         default:
241             fprintf(stderr,"ERROR using die_dietoR1pam(c): c=%c(0x%xx) is not in the valid domain
242                 {A,B,C,D,E,F}\n",c,c);
243             exit(EXIT_FAILURE);
244     }
245     return rval;
246 }
247 /*
248 * map die face values to complex plane C^1
249 *
250 *           imaginary axis
251 *           |
252 *           B=(cos90 ,sin90 )
253 *
254 * (cos150 ,sin150 )=C           A=(cos30 ,sin30 )
255 *           |
256 *           ----- real axis
257 *
258 * (cos210 ,sin210 )=D           F=(cos330 ,sin330 )
259 *           |
260 *           E=(cos270 ,sin270 )
261 *
262 *           |
263 *
264 */
265 complex die_dietoC1c(char c){
266     complex rc;
267     switch(c){
```

```

269     case 'A': rc = expi( 30.0/180.0*PI); break;
270     case 'B': rc = expi( 90.0/180.0*PI); break;
271     case 'C': rc = expi(150.0/180.0*PI); break;
272     case 'D': rc = expi(210.0/180.0*PI); break;
273     case 'E': rc = expi(270.0/180.0*PI); break;
274     case 'F': rc = expi(330.0/180.0*PI); break;
275     case '0': rc.put(0,0); break;
276     default: rc.put(0,0); break;
277     fprintf(stderr,"ERROR using dietoC1(char c): c=%c(0x%xx) is not in the valid domain
278             {0,A,B,C,D,E,F}. Returning (0,0).\n",c,c);
279 }
280 return rc;
281 }
282 /*
283 * map die face values to R^3
284 *   |+1|   | 0|   | 0|   | 0|   | 0|   |-1|
285 * A->| 0| B->|+1| C->| 0| D->| 0| E->|-1| F->| 0|
286 *   | 0|   | 0|   |+1|   |-1|   | 0|   | 0|
287 */
288 vectR3 die_dietoR3(char c){
289     vectR3 xyz;
290     switch(c){
291         case 'A': xyz.put(+1, 0, 0); break;
292         case 'B': xyz.put( 0,+1, 0); break;
293         case 'C': xyz.put( 0, 0,+1); break;
294         case 'D': xyz.put( 0, 0,-1); break;
295         case 'E': xyz.put( 0,-1, 0); break;
296         case 'F': xyz.put(-1, 0, 0); break;
297         default:
298             fprintf(stderr,"ERROR using die_dietoR3(c): c=%c(0x%xx) is not in the valid domain
299                 {A,B,C,D,E,F}\n",c,c);
300             exit(EXIT_FAILURE);
301     }
302     return xyz;
303 }
304 /*
305 * map die face values to R^6
306 * A ->(1,0,0,0,0,0) D ->(0,0,0,1,0,0)
307 * B ->(0,1,0,0,0,0) E ->(0,0,0,0,1,0)
308 * C ->(0,0,1,0,0,0) F ->(0,0,0,0,0,1)
309 * 0 ->(0,0,0,0,0,0)
310 * on ERROR return (0,0,0,0,0,0)
311 */
312 vectR6 die_dietoR6(char c){
313     vectR6 rsix;
314     switch(c){
315         case 'A': rsix.put(1,0,0,0,0,0); break;
316         case 'B': rsix.put(0,1,0,0,0,0); break;
317         case 'C': rsix.put(0,0,1,0,0,0); break;
318         case 'D': rsix.put(0,0,0,1,0,0); break;
319         case 'E': rsix.put(0,0,0,0,1,0); break;
320         case 'F': rsix.put(0,0,0,0,0,1); break;
321         default:
322             fprintf(stderr,"ERROR using dietoR6(char c): c=%c(0x%xx) is not in the valid domain
323                 {0,A,B,C,D,E,F}.\n",c,c);
324             exit(EXIT_FAILURE);
325     }
326     return rsix;
327 }
328 /**
329 // * compare dieseq x and dieseq y
330 // * return the number of locations in which the two sequences are different

```

```

331 // * return 0 if the same
332 // *
333 //long dieseq_cmp(const dieseq *x, const dieseq *y, int showdiff, FILE *fptr){
334 // const long N=x->getN();
335 // const long M=y->getN();
336 // const long NM=(N>M)?N:M; //NM = the smaller of N and M
337 // char xchar,ychar;
338 // long n;
339 // long count;
340 // if(N>M)fprintf(stderr,"Warning: In dieseq x == dieseq y operation, size of x (%ld) is smaller
341 // than size of y (%ld). The first %ld elements of x and y will be compared.\n",N,M,NM);
342 // if(N<M)fprintf(stderr,"Warning: In dieseq x = dieseq y operation, size of x (%ld) is larger
343 // than size of y (%ld). The first %ld elements of x and y will be compared.\n",N,M,NM);
344 // for(n=0,count=0;n<NM;n++){
345 //     xchar=x->get(n);
346 //     uchar=y->get(n);
347 //     if(xchar!=uchar){
348 //         count++;
349 //         if(showdiff) fprintf(stdout,"%6ld: x[%6ld]=%c(0x%02x)
350 // y[%6ld]=%c(0x%02x)\n",count,n,xchar,xchar,n,uchar,uchar);
351 //     }
352 // }
353 // return count;
354 // }
```

### D.3 Real die routines

```

1 /*=====
2 * Daniel J. Greenhoe
3 * header file for routines for real die routines
4 * 'A'--> die face value 1
5 * 'B'--> die face value 2
6 * 'C'--> die face value 3
7 * 'D'--> die face value 4
8 * 'E'--> die face value 5
9 * 'F'--> die face value 6
*=====
10 class rdieseque public dieseque {
11 public:
12     rdieseque(const long M) : dieseque(M) {};
13     rdieseque(const long M, const unsigned seed) : dieseque(M, seed) {};
14     void operator=(dieseque y); //x=y
15     void operator>=(long n){shiftR(n); } //shift rdieseque n elements to the right
16     void operator<=(long n){shiftL(n); } //shift rdieseque n elements to the left
17     int metrictbl(void);
18     int Rxx(const seqR1 *Rxx, const int showcount);
19     int Rxx(const seqR1 *Rxy, const int showcount, const long N, const long M, const long start,
20             const long finish);
21     int Rxxo(const seqR1 *rxx, const int showcount);
22     double Rxx(const long m);
23 };
24
25 extern rdieseque rdie_R1todie_euclid(seqR1 xyz);
26 extern rdieseque rdie_R3todie_larc(seqR3 xyz);
27 extern rdieseque rdie_R3todie0_larc(seqR3 xyz);
28 extern rdieseque rdie_R3todie_euclid(seqR3 xyz);
29 extern rdieseque rdie_R3todie0_euclid(seqR3 xyz);
30 extern vectR3 rdie_dietoR3(char c);
31 extern int rdie_dietoR1(char c);
32 extern int rdie_domain(char c); //check if value is in the domain of rdie
```

```
33 extern double rdiemetric(char a, char b);  
34 extern double rdiemetric(rdieseq x, rdieseq y); //metric for two sequences
```

```
63     return rval;
64 }
65
66 /*-----*
67 * autocorrelation Rxx(m)
68 *-----*/
69 double rdieseQ::Rxx(const long m) {
70     const long mm=labs(m);
71     const long N=getN();
72     long n,mm;
73     double d,sum;
74     char a,b;
75     for (n=0,sum=0;n<(N+mm);n++) {
76         nmm=n-mm;
77         a=(n < 0 || n >=N)? 0.0 : get(n);
78         b=(nmm<0 || nmm>=N)? 0.0 : get(nmm);
79         d=(a==0 || b==0)? 1.0 : rdie_metric(a,b);
80         sum+=d;
81     }
82     return -sum;
83 }
84
85 /*=====
86 * operators
87 =====*/
88 /*
89 * operator rdieseQ x = dieseQ y
90 */
91 void rdieseQ::operator=(dieseQ y) {
92     long n;
93     const long N=getN();
94     const long M=y.getN();
95     char symbol;
96     if (N!=M) {
97         printf(stderr,"ERROR using rdieseQ x = rdieseQ y operation: size of x (%ld) does not equal
98             size of y (%ld)\n",N,M);
99         exit(EXIT_FAILURE);
100    }
101    for (n=0;n<N;n++) {
102        symbol=y.get(n);
103        put(n,symbol);
104    }
105
106 /*=====
107 * external operations
108 =====*/
109 /*
110 * map die face values to R^1
111 * A-->1 B-->2 C-->3 D-->4 E-->5 F-->6
112 */
113 int rdie_dietoR1(char c) {
114     int n,rval;
115     char domain[6]={ 'A', 'B', 'C', 'D', 'E', 'F' };
116     char element;
117     for (n=0,rval=-1;n<6;n++) if (c==domain[n]) rval=n;
118     if (rval== -1) {
119         printf(stderr,"ERROR using rdie_dietoR1(char c): c=%c(0x%x) is not in the valid domain
120             {0,A,B,C,D,E,F}\n",c,c);
121         exit(EXIT_FAILURE);
122     }
123     return rval;
124 }
125 /*-----*
```

```

126 * map die face values to R^3
127 * |+1| | 0| | 0| | 0| | 0| |-1| | 0|
128 * A->| 0| B->|+1| C->| 0| D->| 0| E-->|-1| F-->| 0| 0-->| 0|
129 * | 0| | 0| |+1| |-1| | 0| | 0| | 0|
130 */
131 vectR3 rdie_dietoR3(char c){
132     vectR3 xyz;
133     switch(c){
134         case 'A': xyz.put(+1, 0, 0); break;
135         case 'B': xyz.put( 0,+1, 0); break;
136         case 'C': xyz.put( 0, 0,+1); break;
137         case 'D': xyz.put( 0, 0,-1); break;
138         case 'E': xyz.put( 0,-1, 0); break;
139         case 'F': xyz.put(-1, 0, 0); break;
140         default:
141             fprintf(stderr , "ERROR using rdie_dietoR3(char c): c=%c(0x%lx) is not in the valid domain
142                         {A,B,C,D,E,F}\n",c,c);
143             exit(EXIT_FAILURE);
144     }
145     return xyz;
146 }
147 /*
148 * map R^3 values to die face values using Lagrange Arc metric
149 */
150 rdieseq rdie_R3todie_larc(seqR3 xyz){
151     long n;
152     int m;
153     long N=xyz.getN();
154     double d[7];
155     double smallestd;
156     char closestface;
157     vectR3 p,q[7];
158     rdieseq rdie(N);
159
160     //q[0].put(0,0,0);
161     q[1]=rdie_dietoR3('A');
162     q[2]=rdie_dietoR3('B');
163     q[3]=rdie_dietoR3('C');
164     q[4]=rdie_dietoR3('D');
165     q[5]=rdie_dietoR3('E');
166     q[6]=rdie_dietoR3('F');
167
168     for(n=0; n<N; n++){
169         p.put(xyz.getx(n),xyz.gety(n),xyz.getz(n));
170         smallestd=larc_metric(p,q[1]);
171         closestface='A';
172         for(m=2;m<7;m++){
173             d[m] = larc_metric(p,q[m]);
174             if(((m&0x01) && (d[m]<smallestd)) || ((!((m&0x01)) && (d[m]<=smallestd))) {
175                 // bias odd samples
176                 // towards smaller values
177                 smallestd=d[m];
178                 closestface='A'+m-1;
179             }
180         }
181         rdie.put(n,closestface);
182     }
183     return rdie;
184 }
185
186 /*
187 * map R^3 values to die face values and (0,0,0) using Lagrange Arc metric
188 */
189

```

```
190 rdiese seq_rdie_R3todie0_larc(seqR3 xyz) {
191     long n;
192     int m;
193     long N=xyz.getN();
194     double d[7];
195     double smallestd;
196     char closestface;
197     vectR3 p,q[7];
198     rdiese seq_rdie(N);
199
200     q[0].put(0,0,0);
201     q[1]=rdie_dietoR3('A');
202     q[2]=rdie_dietoR3('B');
203     q[3]=rdie_dietoR3('C');
204     q[4]=rdie_dietoR3('D');
205     q[5]=rdie_dietoR3('E');
206     q[6]=rdie_dietoR3('F');
207
208     for(n=0; n<N; n++) {
209         p.put(xyz.getx(n),xyz.gety(n),xyz.getz(n));
210         smallestd=larc_metric(p,q[0]);
211         //smallestd=ae_metric(1,p,q[0]);
212         closestface='0';
213         for(m=1;m<7;m++) {
214             d[m] = larc_metric(p,q[m]);
215             if(((m&0x01) && (d[m]<smallestd)) || ((!(m&0x01)) && (d[m]<=smallestd))) {
216                 // bias odd samples
217                 // towards smaller values
218                 smallestd=d[m];
219                 closestface='A'+m-1;
220             }
221         }
222         rdie.put(n,closestface);
223     }
224     return rdie;
225 }
226
227 /*
228 * map R^3 values to die face values using Euclidean metric
229 * 0 A B C D E F A+...+F
230 */
231
232 rdiese seq_rdie_R3todie_euclid(seqR3 xyz) {
233     long n;
234     int m;
235     long N=xyz.getN();
236     double d[7];
237     double smallestd;
238     char closestface;
239     vectR3 p,q[7];
240     rdiese seq_rdie(N);
241
242     q[1]=rdie_dietoR3('A');
243     q[2]=rdie_dietoR3('B');
244     q[3]=rdie_dietoR3('C');
245     q[4]=rdie_dietoR3('D');
246     q[5]=rdie_dietoR3('E');
247     q[6]=rdie_dietoR3('F');
248
249     for(n=0; n<N; n++) {
250         p.put(xyz.getx(n),xyz.gety(n),xyz.getz(n));
251         smallestd=ae_metric(1,p,q[1]);
252         closestface='A';
253         for(m=2;m<=6;m++) {
254             d[m] = ae_metric(1,p,q[m]);
```

```

255     if (((m&0x01) && (d[m]<smallestd)) || ((!(m&0x01)) && (d[m]<=smallestd))) {
256         //_____
257         // bias towards smaller values      bias towards larger values
258         smallestd=d[m];
259         closestface='A'+m-1;
260     }
261 }
262     rdie.put(n,closestface);
263 }
264 return rdie;
265 }

266 /*
267 * map R^3 values to die face and (0,0,0) values using Euclidean metric
268 *   0   A   B   C   D   E   F   A+..+F
269 */
270 rdiseq rdie_R3todie0_euclid(seqR3 xyz){
271 long n;
272 int m;
273 long N=xyz.getN();
274 double d[7];
275 double smallestd;
276 char closestface;
277 vectR3 p,q[7];
278 rdiseq rdie(N);
279
280 q[0]=rdie_dietoR3('0');
281 q[1]=rdie_dietoR3('A');
282 q[2]=rdie_dietoR3('B');
283 q[3]=rdie_dietoR3('C');
284 q[4]=rdie_dietoR3('D');
285 q[5]=rdie_dietoR3('E');
286 q[6]=rdie_dietoR3('F');
287
288 for(n=0; n<N; n++){
289     p.put(xyz.getx(n),xyz.gety(n),xyz.getz(n));
290     smallestd=ae_metric(1,p,q[0]);
291     closestface='0';
292     for(m=1;m<=6;m++) {
293         d[m] = ae_metric(1,p,q[m]);
294         // if (d[m]<smallestd)
295         // if (d[m]<=smallestd)
296         // if (((m&0x01) && (d[m]<smallestd)) || ((!(m&0x01)) && (d[m]<smallestd)))
297         // if (((m&0x01) && (d[m]<=smallestd)) || ((!(m&0x01)) && (d[m]<smallestd)))
298         if (((m&0x01) && (d[m]<smallestd)) || ((!(m&0x01)) && (d[m]<=smallestd))) {
299             //_____
300             // bias towards smaller values      bias towards larger values
301             smallestd=d[m];
302             closestface='A'+m-1;
303         }
304         // if (m&0x01){ (alternative coding)
305         //   if (d[m]<smallestd){
306         //     smallestd=d[m];
307         //     closestface='A'+m-1;
308         //   }
309         // else
310         //   if (d[m]<=smallestd){
311         //     smallestd=d[m];
312         //     closestface='A'+m-1;
313         //   }
314         }
315     }
316     rdie.put(n,closestface);
317 }
318 return rdie;
319 }

```

```
320 /*-----  
321 * map R^1 values to die face values using Euclidean metric  
322 *-----*/  
323 rdieseq rdie_R1todie_euclid(seqR1 xyz){  
324     long n;  
325     long N=xyz.getN();  
326     char closestface;  
327     double p;  
328     rdieseq rdie(N);  
329  
330     for(n=0; n<N; n++){  
331         p = xyz.get(n);  
332         if(p<1.5)    closestface='A';  
333         else if(p>=5.5) closestface='F';  
334         else            closestface=(char)(p+0.5-1)+'A';  
335         rdie.put(n,closestface);  
336     }  
337     return rdie;  
338 }  
340 /*-----  
341 * real die metric d(a,b)  
342 *-----*/  
343 *      | 0   A   B   C   D   E   F   (b)  
344 *      |  
345 *      a= 0| 0   1   1   1   1   1   1  
346 *      a= A| 1   0   1   1   1   1   2  
347 *      a= B| 1   1   0   1   1   2   1  
348 *      a= C| 1   1   1   0   2   1   1  
349 *      a= D| 1   1   1   2   0   1   1  
350 *      a= E| 1   1   2   1   1   0   1  
351 *      a= F| 1   2   1   1   1   1   0  
352 * On success return d(a,b). On error return -1.  
353 */  
354 double rdie_metric(char a, char b){  
355     int ra=rdie_dietoR1(a);  
356     int rb=rdie_dietoR1(b);  
357     double d;  
358  
359     if(ra<0)fprintf(stderr,"a=%c(0x%02x) not in domain of rdie metric d(a,b)\n",a,a);  
360     if(rb<0)fprintf(stderr,"b=%c(0x%02x) not in domain of rdie metric d(a,b)\n",b,b);  
361  
362     if(ra<0)    d=-1;  
363     else if(rb<0) d=-1;  
364     else if(ra==rb) {d=0; }  
365     else if(ra==0) d=1;  
366     else if(rb==0) d=1;  
367     else if(ra+rb==7) d=2;  
368     else           d=1;  
369     return d;  
370 }  
371 /*-----  
372 * real die metric p(x,y) where x and y are rdie sequences computed as  
373 * p(x,y) = d(x0,y0) + d(x1,y1) + d(x2,y2) + ... + d(x{N-1},y{N-1})  
374 * where d(a,b) is defined above.  
375 * On success return d(x,y). On error return -1.  
376 */  
377 double rdie_metric(rdiaseq x, rdiaseq y){  
378     double rval,d;  
379     long n;  
380     long N=x.getN();  
381     long M=y.getN(); //NM = the larger of N and M  
382     long NM=(N<M)?M:N;  
383     for(n=0,d=0;n<NM;n++){  
384 }
```

```

385     rval=rdie_metric(x.get(n),y.get(n));
386     if(rval<0){d+=0.0; printf("rval=%lf ",rval);}
387     else d+=rval;
388 }
389 if(N!=M){
390   fprintf(stderr,"ERROR using rdie_metric(rdiese q x,rdiese q y): size of x (%ld) does not equal
391           the size of y (%ld).\n",N,M);
392   exit(EXIT_FAILURE);
393 }
394 return d;
395 }
```

## D.4 Spinner routines

```

1  /*=====
2  * Daniel J. Greenhoe
3  * header file for routines for spinner routines
4  * 'A'--> spin face value 1
5  * 'B'--> spin face value 2
6  * 'C'--> spin face value 3
7  * 'D'--> spin face value 4
8  * 'E'--> spin face value 5
9  * 'F'--> spin face value 6
*=====
11 class spinseq: public dieseq {
12 public:
13     spinseq(const long M) : dieseq(M) {};
14     spinseq(const long M, const unsigned seed) : dieseq(M,seed) {};
15     seqR1 spinptoR1(void);           //map spin face values to R^1
16     seqR2 spinptoR2(void);           //map spin face values to R^2
17     void operator=(spinseq y);      //x=y
18     int metrictbl(void);
19     double Rxx (const long m);
20     int    Rxx (const seqR1 *Rxx, const int showcount);
21     int    Rxo (const seqR1 *rxx, const int showcount);
22     spinseq downsample(int factor); //downsample by a factor of <factor>
23 };
24
25 extern int spin_domain(char c); //check if value is in the domain of rspin
26 extern spinseq spin_R1tospin_euclid(seqR1 xy);
27 extern spinseq spin_R2tospin_larc(seqR2 xy);
28 extern spinseq spin_R2tospin0_larc(seqR2 xy);
29 extern spinseq spin_R2tospin_euclid(seqR2 xy);
30 extern spinseq spin_R2tospin0_euclid(seqR2 xy);
31 extern vectR2 spin_spintoR2(char c);
32 extern double spin_spintoR1(char c);
33 extern double spin_metric(char a, char b);
34 extern double spin_metric(spinseq x, spinseq y); //metric for two sequences
35 //extern seqR1 spin_correlation(spinseq x, spinseq y, int showcount); //correlation
36 //extern seqR1 spin_correlation(spinseq x, spinseq y){return
37     spin_correlation(x,y,0);}// correlation

```

```

1  /*=====
2  * Daniel J. Greenhoe
3  * routines for Real spin spinseqs
4  *=====
5  /*=====
6  * headers
7  *=====
8 #include<stdio.h>
9 #include<stdlib.h>
```

```
10 #include<math.h>
11 #include<main.h>
12 #include<symseq.h>
13 #include<r1.h>
14 #include<r2.h>
15 #include<r3.h>
16 #include<r6.h>
17 #include<c1.h>
18 #include<euclid.h>
19 #include<larc.h>
20 #include<die.h>
21 #include<spinner.h>
22
23 /*
24 * display spinner metric table
25 */
26 int spinseq::metrictbl(void){
27     char a,b;
28     for(a='A';a<='F';a++){
29         for(b='A';b<='F';b++) printf ("d(%c,%c)=% .1lf ",a,b,spin_metric(a,b));
30         printf ("\n");
31     }
32     return 1;
33 }
34
35 /*
36 * autocorrelation Rxx of a spinner seqR1 x with 2N offset
37 */
38 int spinseq::Rxxo(const seqR1 *rxx, const int showcount){
39     const long N=getN();
40     int rval;
41     rval=Rxx(rxx,showcount);
42     rxx->add(2*N);
43     return rval;
44 }
45
46 /*
47 * autocorrelation Rxx of a spinner seqR1 x
48 */
49 int spinseq::Rxx(const seqR1 *rxx, const int showcount){
50     long m;
51     const long N=getN();
52     int rval=0;
53     double rx xm;
54     if(showcount) fprintf(stderr," Calculate %ld auto-correlation values ... n=",2*N+1);
55     for(m=-N;m<=N;m++){
56         if(showcount) fprintf(stderr,"%8ld ",m+N);
57         rx xm=Rxx(m);
58         if(rx xm>0) rval=-1;
59         rxx->put(m+N,rx xm);
60         if(showcount) fprintf(stderr,"\\b\\b\\b\\b\\b\\b\\b\\b\\b");
61     }
62     if(showcount) fprintf(stderr,"%8ld .... done.\n",m+N);
63     return rval;
64 }
65
66 /*
67 * autocorrelation Rxx(m)
68 */
69 double spinseq::Rxx(const long m){
70     const long mm=labs(m);
71     const long N=getN();
72     long n,mm;
73     double d,sum;
74     char a,b;
```

```
75   for (n=0,sum=0;n<(N+mm) ;n++) {  
76     nmm=n-mm;  
77     a=(n <0 || n >=N)? 0.0 : get(n);  
78     b=(nmm<0 || nmm>=N)? 0.0 : get(nmm);  
79     d=(a==0 || b==0)? 1.0 : spin_metric(a,b);  
80     sum+=d;  
81   }  
82   return -sum;  
83 }  
84  
85 /*-----  
86 * downsample sequence by a factor of <factor>  
87 */  
88 spinseq spinseq::downsample(int factor){  
89   const long N=getN();  
90   long n,m;  
91   long M;  
92   if(factor<1){  
93     fprintf(stderr,"ERROR using dieseq::downsample: factor=%d must be at least 1\n",factor);  
94     exit(EXIT_FAILURE);  
95   }  
96   M=N/factor;  
97   spinseq newseq(M);  
98   for(n=0,m=0; m<M; n+=factor ,m++)newseq.put(m, get(n));  
99   return newseq;  
100 }  
101  
102 /*-----  
103 * map spin face values to R^1  
104 * A-->1 B-->2 C-->3 D-->4 E-->5 F-->6  
105 */  
106 seqR1 spinseq::spintoR1(void){  
107   const long N=getN();  
108   long n;  
109   seqR1 y(N);  
110   for(n=0; n<N; n++)y.put(n, spin_spintoR1(get(n)));  
111   return y;  
112 }  
113  
114 /*-----  
115 * map spin face values to R^2 sequence  
116 */  
117 seqR2 spinseq::spintoR2(void){  
118   const long N=getN();  
119   long n;  
120   seqR2 seqR2(N);  
121   for(n=0; n<N; n++)seqR2.put(n, spin_spintoR2(get(n)));  
122   return seqR2;  
123 }  
124  
125  
126 /*=====  
127 * operators  
128 =====*/  
129 /*-----  
130 * operator spinseq x = dieseq y  
131 */  
132 void spinseq::operator=(spinseq y){  
133   long n;  
134   const long N=getN();  
135   const long M=y.getN();  
136   char symbol;  
137   if(N!=M){  
138     fprintf(stderr,"\nERROR using spinseq x = spinseq y: size of x (%ld) is smaller than size of
```

```
    y (%ld)\n",N,M,N);
140 exit(EXIT_FAILURE);
141 }
142 for(n=0;n<N;n++){
143     symbol=y.get(n);
144     put(n,symbol);
145 }
146 }
147
148 /*=====
149 * external operations
150 =====*/
151 /*
152 * map spin face values to R^1
153 */
154 double spin_spintoR1(char c){
155     double rval;
156     switch(c){
157         case 'A': rval = 1.0; break;
158         case 'B': rval = 2.0; break;
159         case 'C': rval = 3.0; break;
160         case 'D': rval = 4.0; break;
161         case 'E': rval = 5.0; break;
162         case 'F': rval = 6.0; break;
163     default:
164         fprintf(stderr,"ERROR using spin_spintoR1(c): c=%c(0x%x) is not in the valid domain
165             {A,B,C,D,E,F}\n",c,c);
166         exit(EXIT_FAILURE);
167     }
168     return rval;
169 }
170 /*
171 * map spinner face values to R^2
172 */
173 vectR2 spin_spintoR2(char c){
174     vectR2 xy;
175     switch(c){
176         case 'A': xy.put(0, -1.0); break;
177         case 'B': xy.put(+sqrt(3)/2,-0.5); break;
178         case 'C': xy.put(+sqrt(3)/2,+0.5); break;
179         case 'D': xy.put( 0, +1.0); break;
180         case 'E': xy.put(-sqrt(3)/2,+0.5); break;
181         case 'F': xy.put(-sqrt(3)/2,-0.5); break;
182     default:
183         fprintf(stderr,"ERROR: c=%c(0x%x) is not in the valid domain {0,A,B,C,D,E,F} in
184             spin_spintoR2(char c)\n",c,c);
185         exit(EXIT_FAILURE);
186     }
187     return xy;
188 }
189 /*
190 * map R^2 values to spin face values using Lagrange Arc distance
191 */
192 spinseq spin_R2tospin_larc(seqR2 xy){
193     long n;
194     int m;
195     long N=xy.getN();
196     double d[7];
197     double smallestd;
198     char closestface;
199     vectR2 p,q[7];
200     spinseq rspin(N);
201 }
```

```
202 //q[0].put(0,0,0);
203 q[1]=spin_spintoR2('A');
204 q[2]=spin_spintoR2('B');
205 q[3]=spin_spintoR2('C');
206 q[4]=spin_spintoR2('D');
207 q[5]=spin_spintoR2('E');
208 q[6]=spin_spintoR2('F');
209
210 for(n=0; n<N; n++){
211     p.put(xy.getx(n),xy.gety(n));
212     smallestd=larc_metric(p,q[1]);
213     closestface='A';
214     for(m=2;m<7;m++){
215         d[m] = larc_metric(p,q[m]);
216         if (((m&0x01) && (d[m]<smallestd)) || ((!(m&0x01)) && (d[m]<=smallestd))) {
217             // bias odd samples
218             // towards smaller values
219             smallestd=d[m];
220             closestface='A'+m-1;
221         }
222     }
223 }
224 rspin.put(n,closestface);
225 }
226 return rspin;
227 }
228
229 /*
230 * map R^3 values to spin face values and (0,0,0) using Lagrange Arc metric
231 */
232 spinseq spin_R2tospin0_larc(seqR2 xy){
233     long n;
234     int m;
235     long N=xy.getN();
236     double d[7];
237     double smallestd;
238     char closestface;
239     vectR2 p,q[7];
240     spinseq rspin(N);
241
242     q[0].put(0,0);
243     q[1]=spin_spintoR2('A');
244     q[2]=spin_spintoR2('B');
245     q[3]=spin_spintoR2('C');
246     q[4]=spin_spintoR2('D');
247     q[5]=spin_spintoR2('E');
248     q[6]=spin_spintoR2('F');
249
250     for(n=0; n<N; n++){
251         p.put(xy.getx(n),xy.gety(n));
252         smallestd=larc_metric(p,q[0]);
253         //smallestd=ae_metric(l,p,q[0]);
254         closestface='0';
255         for(m=1;m<7;m++){
256             d[m] = larc_metric(p,q[m]);
257             if (((m&0x01) && (d[m]<smallestd)) || ((!(m&0x01)) && (d[m]<=smallestd))) {
258                 // bias odd samples
259                 // towards smaller values
260                 smallestd=d[m];
261                 closestface='A'+m-1;
262             }
263         }
264     }
265     rspin.put(n,closestface);
266 }
```

```
267     return rspin;
268 }
269 */
270 /* map R^2 values to spin face values using Euclidean metric
271 * 0 A B C D E F A+...+F
272 */
273 */
274 spinseq spin_R2tospin_euclid(seqR2 xy){
275     long n;
276     int m;
277     long N=xy.getN();
278     double d[7];
279     double smallestd;
280     char closestface;
281     vectR2 p,q[7];
282     spinseq rspin(N);
283
284     q[1]=spin_spintoR2('A');
285     q[2]=spin_spintoR2('B');
286     q[3]=spin_spintoR2('C');
287     q[4]=spin_spintoR2('D');
288     q[5]=spin_spintoR2('E');
289     q[6]=spin_spintoR2('F');
290
291     for(n=0; n<N; n++){
292         p.put(xy.getx(n),xy.gety(n));
293         smallestd=ae_metric(1,p,q[1]);
294         closestface='A';
295         for(m=2;m<=6;m++) {
296             d[m] = ae_metric(1,p,q[m]);
297             if (((m&0x01) && (d[m]<smallestd)) || ((!(m&0x01)) && (d[m]<=smallestd))) {
298                 // bias towards smaller values
299                 smallestd=d[m];
300                 closestface='A'+m-1;
301             }
302         }
303     }
304     rspin.put(n,closestface);
305 }
306
307     return rspin;
308 }
309 */
310 /* map R^2 values to spin face and (0,0) values using Euclidean metric
311 * 0 A B C D E F A+...+F
312 */
313 spinseq spin_R2tospin0_euclid(seqR3 xy){
314     long n;
315     int m;
316     long N=xy.getN();
317     double d[7];
318     double smallestd;
319     char closestface;
320     vectR2 p,q[7];
321     spinseq rspin(N);
322
323     q[0]=spin_spintoR2('0');
324     q[1]=spin_spintoR2('A');
325     q[2]=spin_spintoR2('B');
326     q[3]=spin_spintoR2('C');
327     q[4]=spin_spintoR2('D');
328     q[5]=spin_spintoR2('E');
329     q[6]=spin_spintoR2('F');
330
331     for(n=0; n<N; n++) {
```

```

332     p.put(xy.getx(n),xy.gety(n));
333     smallestd=ae_metric(1,p,q[0]);
334     closestface='0';
335     for(m=1;m<=6;m++) {
336         d[m] = ae_metric(1,p,q[m]);
337         // if (d[m]<smallestd)
338         // if (d[m]<=smallestd)
339         // if (((m&0x01) && (d[m]<smallestd)) || ((!(m&0x01)) && (d[m]<smallestd)))
340         // if (((m&0x01) && (d[m]<=smallestd)) || ((!(m&0x01)) && (d[m]<=smallestd)))
341         if (((m&0x01) && (d[m]<smallestd)) || ((!(m&0x01)) && (d[m]<=smallestd))) {
342             //-----
343             // bias towards smaller values           bias towards larger values
344             smallestd=d[m];
345             closestface='A'+m-1;
346         }
347         //if(m&0x01){ (alternative coding)
348         //  if(d[m]<smallestd){
349         //    smallestd=d[m];
350         //    closestface='A'+m-1;
351         //  }
352         //else
353         //  if(d[m]<=smallestd){
354         //    smallestd=d[m];
355         //    closestface='A'+m-1;
356         //  }
357     }
358     rspin.put(n,closestface);
359 }
360
361 return rspin;
362
363 /*
364 * map R^1 values to spin face values using Euclidean metric
365 */
366 spinseq spin_R1tospin_euclid(seqR1 xy){
367     long n;
368     long N=xy.getN();
369     char closestface;
370     double p;
371     spinseq rspin(N);
372
373     for(n=0; n<N; n++){
374         p = xy.get(n);
375         if(p<1.5)      closestface='A';
376         else if(p>=5.5) closestface='F';
377         else            closestface=(char)(p+0.5-1)+'A';
378         rspin.put(n,closestface);
379     }
380     return rspin;
381 }
382
383 /*
384 * spinner metric d(a,b)
385 *   d(a,b) | A   B   C   D   E   F   (b)
386 *
387 *   a= A| 0   1   2   3   2   1
388 *   a= B| 1   0   1   2   3   2
389 *   a= C| 2   1   0   1   2   3
390 *   a= D| 3   2   1   0   1   2
391 *   a= E| 2   3   2   1   0   1
392 *   a= F| 1   2   3   2   1   0
393 * On success return d(a,b). On error return -1.
394 */
395 double spin_metric(char a, char b){
396     double ra=spin_spintoR1(a);

```

```

397 double rb=spin_spintoR1(b);
398 double d;
399
400 d=(double)fabs(ra-rb);
401 if      (d>3.5) d=2.0;
402 else if(d>4.5) d=1.0;
403 return d;
404 }
```

## D.5 DNA routines

```

1 /*=====
2 * Daniel J. Greenhoe
3 * header file for routines for DNA routines
4 =====*/
5 class dnaseq: public symseq {
6 public:
7     dnaseq(long M) : symseq(M) {} //constructor initializing to '.'
8     void seed(unsigned seed) {srand(seed);}   //
9     void randomize(void) {symseq::randomize("ATCG");}    //
10    void randomize(unsigned seed) {srand(seed); randomize();}
11    int randomize(long start, long end, int wA,int wT,int wC,int wG);
12    int randomize(unsigned seed,int wA,int wT,int wC,int wG) {srand(seed);return
13        randomize(0,getN()-1,wA,wT,wC,wG);}
14    int randomize(int wA,int wT,int wC,int wG){return randomize(0,getN()-1,wA,wT,wC,wG);}
15    int randomize(long start, long end, unsigned seed,int wA,int wT,int wC,int
16        wG) {srand(seed);return randomize(start,end,wA,wT,wC,wG);}
17    char get(long n){return symseq::get(n,"ATCG");} //get a value from x at location n
18    void put(long n, char symbol){symseq::put(n,symbol);}
19    void put(dnaseq *y, const long n, const char symbol);
20    void put(dnaseq *y, long n){return put(y,n,'.);}
21    void put(const long start, const long end, char c); //put a value <c> at locations start to
22        end
23    seqR2 dnatoR2(void);           //map gsp face values to R^2
24    dnaseq downsample(int factor); //downsample by a factor of <factor>
25    seqR1 dnatoR1(void);          //map dna sequence to R^1
26    seqC1 dnatoC1(void);          //map dna sequence to R^1
27    seqR1 dnatoRlpam(void);       //map dna sequence to R^1 using PAM scheme (symmetric about
28        zero)
29    seqR1 dnatoR1bin(void);       //map dna sequence to R^1 using AT-CG binary scheme
30    seqR4 dnatoR4(void);          //map dna sequence to R^4
31    double Rxx (const long m);
32    int Rxx (const seqR1 *Rxx, const int showcount);
33    int Rxxo(const seqR1 *rxx, const int showcount);
34
35    seqR1 histogram(const long start, const long end, int display, FILE *fptr); //compute,
36        display, and write histogram
37    seqR1 histogram() {return histogram(0,getN()-1,0,NULL);} //compute histogram
38    seqR1 histogram(const long start, const long end){return
39        histogram(start,end,0,NULL);} //compute histogram
40    seqR1 histogram(int display,FILE *fptr){return histogram(0,getN()-1,display,fptr);} //print
41        histogram to file
42    seqR1 histogram(FILE *fptr){return histogram(0,getN()-1,0,fptr);} //print histogram to file
43    void operator=(dnaseq y); //x=y
44};

45 extern long numsym_fasta_file(const char *filename);
46 extern int read_fasta_file(const char *filename, char *description, dnaseq *x);
47 extern int dna_domain (char symbol);
48 extern vectR2 dna_dnatoR2(char symbol);
49 extern double dna_dnatoR1(char symbol);
50 extern complex dnatoC1c (char symbol);
```

```
45 extern vectR4 dnatoR4c (char symbol);  
  
1 /*=====  
2 * Daniel J. Greenhoe  
3 * routines for Real gsp dnaseqs  
4 *=====*/  
5 /*=====  
6 * headers  
7 *=====*/  
8 #include<stdio.h>  
9 #include<stdlib.h>  
10 #include<string.h>  
11 #include<math.h>  
12 #include<main.h>  
13 #include<symseq.h>  
14 #include<r1.h>  
15 #include<r2.h>  
16 #include<r3.h>  
17 #include<r4.h>  
18 #include<r6.h>  
19 #include<c1.h>  
20 #include<euclid.h>  
21 #include<larc.h>  
22 #include<dna.h>  
23  
24 /*=====  
25 * prototypes  
26 *=====*/  
27 void dna_phistogram(seqR1 *data, const long start, const long end, FILE *ptr);  
28  
29 /*-----  
30 * constructor initializing dnaseq to <c>  
31 *-----*/  
32 //dnaseq::dnaseq(long M,char c){  
33 // long n;  
34 // void *memptr;  
35 // N=M;  
36 // memptr=malloc(N*sizeof(char));  
37 // if (memptr==NULL) fprintf(stderr,"dnaseq::dnaseq memory allocation for %ld elements  
38 // failed\n",M);  
39 // else{  
40 //   x = (char *)memptr;  
41 //   for(n=0; n<N; n++)x[n]=c;  
42 // }  
43 //  
44 //dnaseq::dnaseq(long M){  
45 // long n;  
46 // void *memptr;  
47 // N=M;  
48 // memptr=malloc(N*sizeof(char));  
49 // if (memptr==NULL) fprintf(stderr,"dnaseq::dnaseq memory allocation for %ld elements  
50 // failed\n",M);  
51 // else{  
52 //   x = (char *)memptr;  
53 //   for(n=0; n<N; n++)x[n]='.';  
54 // }  
55 //  
56 /*-----  
57 * fill the dnaseq with the value '.'  
58 *-----*/  
59 //void dnaseq::clear(void){  
60 // long n;  
61 // for(n=0; n<N; n++)x[n]='.';
```

```
62 // }
```

```
63
64
65 /*
66 * copy a dnaseq <y> into dnaseq x starting at location <n>
67 * and fill any remaining locations with <c>
68 */
69 void dnaseq::put(dnaseq *y, const long n, const char symbol) {
70     const long N=getN();
71     long i,j;
72     long M=y->getN();
73     if (n>=N) {
74         fprintf(stderr, "\nERROR using dnaseq::put(y,n,symbol): n=%ld outside sequence domain
75             [0:%ld]\n",n,N-1);
76         exit(EXIT_FAILURE);
77     }
78     if (! dna_domain(symbol)) {
79         fprintf(stderr, "\nERROR using dnaseq::put(y,n,symbol): symbol='%c'=0x%x not in sequence range
80             {A,T,C,G}\n",symbol,symbol);
81         exit(EXIT_FAILURE);
82     }
83     else {
84         for (i=0;i<n; i++) put(i,symbol);
85         for (j=0;j<M; j++,i++) put(i,y->get(j));
86         for ( ;i<N; i++) put(i,symbol);
87     }
88 }
89 /*
90 * put the value <c> into the sequence x from location <start> to <end>
91 */
92 void dnaseq::put(const long start, const long end, const char symbol) {
93     const long N=getN();
94     long n;
95     if (start<0||end>=N|| start>end) {
96         fprintf(stderr, "ERROR using dnaseq::put(%ld,%ld,'%c')\n", start ,end ,symbol);
97         exit(EXIT_FAILURE);
98     }
99     for (n=start; n<=end; n++) put(n,symbol);
100 }
101 /*
102 * fill the dnaseq with pseudo-random DNA values
103 * using seed value <seed>
104 * distributed with the weight values <wA,wT,wC,wG>
105 * where each weight value wX in an integer in the closed interval [0,100]
106 * and where the sum of the intervals must be 100.
107 */
108 int dnaseq::randomize(long start, long finish, int wA,int wT,int wC,int wG) {
109     int r,u;
110     long n;
111     int sum=wA+wT+wC+wG;
112     char symbol;
113     if (sum!=100) {
114         fprintf(stderr, "ERROR using dnaseq::randomize(start ,finish ,wA,wT,wC,wG): sum of weight values
115             = %d != 100\n",sum);
116         exit(EXIT_FAILURE);
117     }
118     for (n=start; n<=finish; n++) {
119         r=rand();
120         u = r%100;
121         if (u<wA) symbol='A';
122         else if (u<wA+wT) symbol='T';
123         else if (u<wA+wT+wC) symbol='C';
124         else symbol='G';
```

```
124     put(n,symbol);
125 }
126 return 0;
127 }
128
129 /*
130 * map dna face values to R^1 using PAM scheme
131 * A-->-1.5 T-->-0.5 C-->0.5 G-->0.5
132 * all other values --> 0
133 */
134 seqR1 dnaseq::dnatoR1pam(void) {
135     const long N=getN();
136     long n;
137     char symbol;
138     seqR1 seqR1(N);
139     for(n=0; n<N; n++) {
140         symbol=get(n);
141         switch(symbol) {
142             case 'A': seqR1.put(n, -1.5); break;
143             case 'C': seqR1.put(n, -0.5); break;
144             case 'T': seqR1.put(n, 0.5); break;
145             case 'G': seqR1.put(n, 1.5); break;
146             default:
147                 fprintf(stderr, "\nERROR using dnaseq::dnatoR1pam(): symbol='%"c'=0x%x not in sequence range
148                         {A,T,C,G}\n", symbol, symbol);
149                 exit(EXIT_FAILURE);
150         }
151     }
152     return seqR1;
153 }
154 /*
155 * map dna face values to R^1 using AT/CG binary scheme
156 * A-->1 T-->1 C-->-1 G-->-1
157 * all other values --> 0
158 */
159 seqR1 dnaseq::dnatoR1bin(void) {
160     const long N=getN();
161     long n;
162     char symbol;
163     seqR1 seqR1(N);
164     for(n=0; n<N; n++) {
165         symbol=get(n);
166         switch(symbol) {
167             case 'A': seqR1.put(n, 1); break;
168             case 'C': seqR1.put(n, -1); break;
169             case 'T': seqR1.put(n, 1); break;
170             case 'G': seqR1.put(n, -1); break;
171             default:
172                 fprintf(stderr, "\nERROR using dnaseq::dnatoR1bin(): symbol='%"c'=0x%x not in sequence range
173                         {A,T,C,G}\n", symbol, symbol);
174                 exit(EXIT_FAILURE);
175         }
176     }
177     return seqR1;
178 }
179 /*
180 * map dna face values to C^1
181 */
182 seqC1 dnaseq::dnatoC1(void) {
183     const long N=getN();
184     long n;
185     char symbol;
186     complex yy;
```

```
187 seqC1 y(N);
188 for(n=0; n<N; n++){
189     symbol = get(n);
190     yy = dnatoClc(symbol);
191     y.put(n,yy);
192 }
193 return y;
194 }

195 /*
196 * map dna face values to complex plane C^1
197 *
198 *           imaginary axis
199 *
200 *           |
201 *           |
202 * (cos135 ,sin135)=C      A=(cos45 ,sin45)
203 *
204 *           -----
205 *           real axis
206 *
207 * (cos225 ,sin225)=T      G=(cos315 ,sin315)
208 *
209 *           |
210 *           |
211 */
212 */

213 complex dnatoClc(char c){
214     complex rc;
215     switch(c){
216         case 'A': rc = expi( 45.0/180.0*PI); break;
217         case 'C': rc = expi(135.0/180.0*PI); break;
218         case 'T': rc = expi(225.0/180.0*PI); break;
219         case 'G': rc = expi(315.0/180.0*PI); break;
220         //
221         //case 'A': rc = expi( 45.0/180.0*PI); break; // Gilleans 2007 mapping
222         //case 'G': rc = expi(135.0/180.0*PI); break;
223         //case 'C': rc = expi(225.0/180.0*PI); break;
224         //case 'T': rc = expi(315.0/180.0*PI); break;
225         //
226         //case 'A': rc.put(+1,+1); break;
227         //case 'G': rc.put(-1,+1); break;
228         //case 'C': rc.put(-1,-1); break;
229         //case 'T': rc.put(+1,-1); break;
230         case '0': rc.put( 0, 0); break;
231         default: rc.put( 0, 0);
232         fprintf(stderr,"ERROR using dnatoClc(char c): c=%c(0x%02X) is not in the valid domain
233             {0,A,C,T,G}. Returning (0,0).\n",c,c);
234     }
235     return rc;
236 }

237 /*
238 * map dna values to R^4 sequence
239 */
240 seqR4 dnaseq::dnatoR4(void){
241     const long N=getN();
242     long n;
243     char yc;
244     seqR4 seq4(N);
245     for(n=0; n<N; n++)seq4.put(n,dnatoR4c(get(n)));
246     return seq4;
247 }

248 /*
249 * map dna face values to R^4
250 */
```

```
251 * A-->(1,0,0,0)    T-->(0,0,1,0)
252 * C-->(0,1,0,0)    G-->(0,0,0,1)
253 * 0-->(0,0,0,0)
254 * on ERROR return (0,0,0,0)
255 */
256 vectR4 dnatoR4c(char c){
257     vectR4 rfour;
258     switch(c){
259         case '0': rfour.put(0,0,0,0); break;
260         case 'A': rfour.put(1,0,0,0); break;
261         case 'C': rfour.put(0,1,0,0); break;
262         case 'T': rfour.put(0,0,1,0); break;
263         case 'G': rfour.put(0,0,0,1); break;
264         default: rfour.put(0,0,0,0);
265         fprintf(stderr,"ERROR using dnatoR4(char c): c=%c(0x%x) is not in the valid domain {A,C,T,G}.\n"
266             "Returning (0,0,0,0).\n",c,c);
267     }
268     return rfour;
269 }
270
271 /*
272 * list contents of dnaseq
273 */
274 //void dnaseq::list(const long start, const long end, const char *str1, const char *str2, FILE
275 //    *fptr){
276 //    long n,m;
277 //    if(strlen(str1)>0){
278 //        printf("%s",str1);
279 //        if(fptr!=NULL)fprintf(fptr,"%s",str1);
280 //    }
281 //    for(n=start,m=1; n<=end; n++,m++){
282 //        printf("%c",get(n));
283 //        if(m%50==0)printf("\n");
284 //        else if(m%10==0)printf(" ");
285 //        if(fptr!=NULL){
286 //            fprintf(fptr,"%c",get(n));
287 //            if(m%50==0)fprintf(fptr,"\n");
288 //            else if(m%10==0)fprintf(fptr," ");
289 //        }
290 //    }
291 //    if(strlen(str2)>0){
292 //        printf("%s",str2);
293 //        if(fptr!=NULL)fprintf(fptr,"%s",str2);
294 //    }
295 //}
296 /*
297 * map gsp face values to R^1
298 * A-->1  T-->2  C-->3  G-->4
299 */
300 seqR1 dnaseq::dnatoR1(void){
301     const long N=getN();
302     long n;
303     char symbol;
304     seqR1 seqR1(N);
305     for(n=0; n<N; n++){
306         symbol = get(n);
307         switch(symbol){
308             case 'A': seqR1.put(n,1); break;
309             case 'T': seqR1.put(n,2); break;
310             case 'C': seqR1.put(n,3); break;
311             case 'G': seqR1.put(n,4); break;
312             default:
313                 fprintf(stderr,"nERROR using dnaseq::dnatoR1(): symbol='%c'=0x%x not in sequence range\n",
```

```
314         {A,T,C,G}\n",symbol,symbol);
315     exit(EXIT_FAILURE);
316 }
317 return seqR1;
318 }

320 /*
321 * map gsp face values to R^2 sequence
322 */
323 seqR2 dnaseq::dnatoR2(void){
324     const long N=getN();
325     long n;
326     char symbol;
327     vectR2 yy;
328     seqR2 seqR2(N);
329     for(n=0; n<N; n++){
330         symbol=get(n);
331         yy=dna_dnatoR2(symbol);
332         seqR2.put(n,yy);
333     }
334     return seqR2;
335 }
336

337 /*
338 * downsample seqR1 by a factor of <factor>
339 */
340 dnaseq dnaseq::downsample(int factor){
341     const long N=getN();
342     long n,m;
343     long M;
344     char symbol;
345     if(factor<1){
346         fprintf(stderr ,"\nERROR using dnaseq::downsample: factor=%d must be at least 1\n",factor);
347         exit(EXIT_FAILURE);
348     }
349     M=N/factor;
350     dnaseq newseq(M);
351     for(n=0,m=0; m<M; n+=factor ,m++){
352         symbol=get(n);
353         newseq.put(m,symbol);
354     }
355     return newseq;
356 }

357 /*
358 * compute histogram of dna sequence
359 * return seqR1 y of length 6 where
360 * y[1]-->number of dna 'A' symbols,
361 * y[2]-->number of dna 'T' symbols,
362 * y[3]-->number of dna 'C' symbols,
363 * y[4]-->number of dna 'G' symbols,
364 * y[0]-->number of all other values
365 * y[5]-->total number of symbols y[1],y[2],...,y[5]
366 */
367 seqR1 dnaseq::histogram(const long start, const long end, int display, FILE *fptr){
368     seqR1 data(6);
369     seqR1 data(6);
370     long n;
371     long bin;
372     double p;
373     int i;
374     char symbol;
375     FILE *ptr;
376     data.clear();
377     for(n=start;n<=end;n++) {
```

```
378     symbol=get(n);
379     switch(symbol){
380       case 'A': bin=1; break;
381       case 'T': bin=2; break;
382       case 'C': bin=3; break;
383       case 'G': bin=4; break;
384       default : bin=0; break;
385     }
386     if(bin!=0) data.increment(5);
387     data.increment(bin);
388   }
389   if(display) dna_phistogram(&data,start,end,stdout);
390   if(fptr!=NULL) dna_phistogram(&data,start,end,fptr );
391   return data;
392 }
393 /*
394 * print DNA histogram with data pointed to by <data>
395 * to stream pointed to by ptr
396 */
397 void dna_phistogram(seqR1 *data, const long start, const long end, FILE *ptr){
398   const long N=end-start+1;
399   long bin;
400   fprintf(ptr ,"\n");
401   fprintf(ptr , "-----\n");
402   fprintf(ptr , "| Histogram for dna sequence [x_n|n=%7ld-%7ld] (length %7ld)\n");
403   fprintf(ptr , "| \n|",start,end,N);
404   fprintf(ptr , "      A          T          C          G          AT          CG          other      |\n|");
405   for(bin=1;bin<=4;bin++) fprintf(ptr , "%10.0lf",data->get(bin));
406   fprintf(ptr , "%10.0lf%10.0lf%10.0lf
407           |\n|",data->get(1)+data->get(2),data->get(3)+data->get(4),data->get(0));
408   for(bin=1;bin<=4;bin++) fprintf(ptr , "(%6.2lf%%)",data->get(bin)/(double)N*100.0);
409   fprintf(ptr , "(%6.2lf%%) (%6.2lf%%) (%6.2lf%%)
410           |\n|", (data->get(1)+data->get(2))/(double)N*100.0,(data->get(3)+data->get(4))/(double)N*100.0,data->get(0));
411   fprintf(ptr , "-----\n");
412 /*
413 * operators
414 */
415 /*
416 * operator dnaseq x = dnaseq y
417 */
418 void dnaseq::operator=(dnaseq y){
419   const long N=getN();
420   const long M=y.getN();
421   long n;
422   char symbol;
423   if(N!=M){
424     fprintf(stderr ,"\nERROR using dnaseq::operator=: length of x (%ld) differs from length of y
425           (%ld).\n",N,M);
426     exit(EXIT_FAILURE);
427   }
428   for(n=0;n<N;n++){
429     symbol = y.get(n);
430     put(n,symbol);
431   }
432 /*
433 * external operations
434 */
435 /*
436 * map dna symbols to R^1
437 * A-->1 T-->2 C-->3 G-->4 O-->0 other-->-1
```

```
439 *-----*/
440 double dna_dnatoR1(char symbol) {
441     double r;
442     switch(symbol) {
443         case 'A': r=1.0; break;
444         case 'T': r=2.0; break;
445         case 'C': r=3.0; break;
446         case 'G': r=4.0; break;
447         default :
448             fprintf(stderr,"ERROR using dna_dnatoR1(symbol): symbol='%c' (0x%lx) is not in the valid
449                 domain {A,T,C,G}\n",symbol,symbol);
450             exit(EXIT_FAILURE);
451     }
452     return r;
453 }
454 */
455 * map dna symbols to R^2
456 */
457 vectR2 dna_dnatoR2(char symbol) {
458     vectR2 r;
459     switch(symbol) {
460         case 'A': r.put( 1.0,      0 ); break;
461         case 'T': r.put(-1.0,      0 ); break;
462         case 'C': r.put( 0,      +1.0 ); break;
463         case 'G': r.put( 0,      -1.0 ); break;
464         default :
465             fprintf(stderr,"ERROR using dna_dnatoR2(symbol): symbol='%c' (0x%lx) is not in the valid
466                 domain {A,T,C,G}\n",symbol,symbol);
467             exit(EXIT_FAILURE);
468     }
469     return r;
470 }
471 */
472 * map R^2 values to dna values using Euclidean metric
473 * 0 A B C D E F A+...+F
474 */
475 dnaseq dna_R2todna_euclid(seqR2 xy) {
476     long n;
477     int m;
478     long N=xy.getN();
479     double d[5];
480     double smallestd;
481     char closestface;
482     vectR2 p,q[5];
483     dnaseq rdna(N);
484
485     //q[0].put(0,0,0);
486     q[1]=dna_dnatoR2('A');
487     q[2]=dna_dnatoR2('T');
488     q[3]=dna_dnatoR2('C');
489     q[4]=dna_dnatoR2('G');
490
491     for(n=0; n<N; n++){
492         p.put(xy.getx(n),xy.gety(n));
493         smallestd=ae_metric(1,p,q[1]);
494         closestface='A';
495         for(m=2;m<5;m++){
496             d[m] = ae_metric(1,p,q[m]);
497             if (((m&0x01) && (d[m]<smallestd)) || ((!((m&0x01))) && (d[m]<=smallestd))) {
498                 /*
499                  // bias odd samples           bias even samples
500                  // towards smaller values       towards larger values
501                 smallestd=d[m];
502             }
503         }
504     }
505 }
```

```

502     switch(m) {
503         case 1: closestface = 'A'; break;
504         case 2: closestface = 'T'; break;
505         case 3: closestface = 'C'; break;
506         case 4: closestface = 'G'; break;
507         default: fprintf(stderr,"Error in dna_R2todna_larc(seqR2 xy)\n");
508     }
509 }
510 }
511 rdna.put(n,closestface);
512 }
513 return rdna;
514 }
515
516 /*
517 * map R^1 values to gsp face values using Euclidean metric
518 */
519 dnaseq dna_R1todna_euclid(seqR1 xy) {
520     long n;
521     long N=xy.getN();
522     char closestface;
523     double p;
524     dnaseq rgsp(N);
525
526     for(n=0; n<N; n++){
527         p = xy.get(n);
528         if(p<1.5) closestface='A';
529         else if(p>=3.5) closestface='G';
530         else if(p>=2.5) closestface='C';
531         else closestface='T';
532         rgsp.put(n,closestface);
533     }
534     return rgsp;
535 }
536
537 /*
538 * dna metric d(a,b)
539 *   d(a,b) | 0   A   T   C   G   (b)
540 *   -----+-----
541 *   a= 0| 0   1   1   1   1
542 *   a= A| 1   0   1   1   1
543 *   a= T| 1   1   0   1   1
544 *   a= C| 1   1   1   0   1
545 *   a= G| 1   1   1   1   0
546 * On success return d(a,b). On error return -1.
547 */
548 double dna_metric(char a, char b){
549     int ra=dna_dnatoR1(a);
550     int rb=dna_dnatoR1(b);
551     double d;
552
553     if(ra<0)fprintf(stderr,"a=%c(0x%02x) not in domain of gsp metric d(a,b)\n",a,a);
554     if(rb<0)fprintf(stderr,"b=%c(0x%02x) not in domain of gsp metric d(a,b)\n",b,b);
555
556     if(ra<0) d=-1.0;
557     else if(rb<0) d=-1.0;
558     else if(ra==rb) d= 0.0;
559     else d= 1.0;
560     return d;
561 }
562
563 /*
564 * real gsp metric p(x,y) where x and y are rgsp sequences computed as
565 * p(x,y) = d(x0,y0) + d(x1,y1) + d(x2,y2) + ... + d(x{N-1},y{N-1})
566 * where d(a,b) is defined above.

```

```
567 * On success return d(x,y). On error return -1.  
568 */  
569 double dna_metric(dnaseq x, dnaseq y){  
570     double rval,d;  
571     long n;  
572     long N=x.getN();  
573     long M=y.getN();  
574     long NM=(N<M)?N:M; //NM = the smaller of N and M  
575     for(n=0,d=0;n<NM;n++){  
576         rval=dna_metric(x.get(n),y.get(n));  
577         if(rval<0){d+=0.0; printf("rval=%lf ",rval);}  
578         else d+=rval;  
579     }  
580     if(N!=M){  
581         fprintf(stderr,"ERROR using dna_metric(x,y): size of x (%ld) does not equal the size of y  
582             (%ld)\n",N,M);  
583         exit(EXIT_FAILURE);  
584     }  
585     return d;  
586 }  
587 /*  
588 * autocorrelation Rxx of a dna sequence x with 2N offset  
589 */  
590 int dnaseq::Rxx(const seqR1 *rxx, const int showcount){  
591     const long N=getN();  
592     int rval;  
593     rval=Rxx(rxx,showcount);  
594     rxx->add(2*N);  
595     return rval;  
596 }  
597 /*  
598 * autocorrelation Rxx of a dna sequence x  
599 */  
600 int dnaseq::Rxx(const seqR1 *rxx, const int showcount){  
601     long m;  
602     const long N=getN();  
603     int rval=0;  
604     double rxm;  
605     if(showcount)fprintf(stderr," Calculate %ld auto-correlation values ... n=",2*N+1);  
606     for(m=-N;m<=N;m++){  
607         if(showcount)fprintf(stderr,"%8ld ",m+N);  
608         rxm=Rxx(m);  
609         if(rxm>0)rval=-1;  
610         rxx->put(m+N,rxm);  
611         if(showcount)fprintf(stderr,"\\b\\b\\b\\b\\b\\b\\b\\b\\b");  
612     }  
613     if(showcount)fprintf(stderr,"%8ld .... done.\n",m+N);  
614     return rval;  
615 }  
616 /*  
617 * autocorrelation Rxx(m)  
618 */  
619 double dnaseq::Rxx(const long m){  
620     const long mm=labs(m);  
621     const long N=getN();  
622     long n,mm;  
623     double d,sum;  
624     char a,b;  
625     for(n=0,sum=0;n<(N+mm);n++){  
626         nmm=n-mm;  
627         a=(n < 0 || n >=N)? 0.0 : get(n);  
628         b=(nmm<0 || nmm>=N)? 0.0 : get(nmm);  
629     }
```

```
631     d=(a==0 || b==0)?      1.0 : dna_metric(a,b);
632     sum+=d;
633 }
634 return -sum;
635 }

636 /*
637 *  read dnna sequence from FASTA formatted file
638 *  and return how many symbols are in it.
639 *  reference: https://www.genomatix.de/online\_help/help/sequence\_formats.html
640 */
641 long numsym_fasta_file(const char *filename){
642     FILE *fptr;
643     int bufN;
644     long N=0;
645     char buffer[1024];
646
647     if(filename==NULL) fptr=stdout;
648     else fptr=fopen(filename , "r");
649     if(fptr==NULL){
650         fprintf(stderr , "Unable to open file %s for reading.\n",filename);
651         return -1;
652     }
653     while(fgets(buffer,1024,fptr)!=NULL){
654         if(buffer[0]=='>'); //printf(" description: %s",buffer);
655         else{
656             bufN = strlen(buffer);
657             N += bufN-1;
658         }
659     }
660     return N;
661 }
662 }

663 /*
664 *  read dnna sequence into <x> from FASTA formatted file
665 *  reference: https://www.genomatix.de/online\_help/help/sequence\_formats.html
666 */
667 int read_fasta_file(const char *filename , char *description , dnaseq *x){
668     FILE *fptr;
669     char buffer[1024];
670     int bufN,i;
671     long n;
672     char symbol;
673
674     if(filename==NULL) fptr=stdout;
675     else fptr=fopen(filename , "r");
676     if(fptr==NULL){
677         fprintf(stderr , "\nERROR using read_fasta_file(%s,...) : unable to open file.\n",filename);
678         exit(EXIT_FAILURE);
679     }
680     n=0;
681     sprintf(description , "No description line found in FASTA file %s.",filename); //default
682     description
683     while(fgets(buffer,1024,fptr)!=NULL){
684         if(buffer[0]=='>') strcpy(description , buffer);
685         else{
686             bufN = strlen(buffer);
687             for(i=0;i<bufN-1;i++){
688                 switch(buffer[i]){
689                     case 'A': symbol='A'; break;
690                     case 'T': symbol='T'; break;
691                     case 'C': symbol='C'; break;
692                     case 'G': symbol='G'; break;
693                     case 'a': symbol='A'; break;
694                     case 't': symbol='T'; break;
```

```
695     case 'c': symbol='C'; break;
696     case 'g': symbol='G'; break;
697     default: symbol='x'; fprintf(stderr,"unknown character %c (%02x) in
698         dnaseq\n",buffer[i],buffer[i]);
699     }
700     x->put(n,symbol);
701     n++;
702   }
703 }
704 fclose(fptr);
705 return 0;
706 }
```

## References

- Charalambos D. Aliprantis and Owen Burkinshaw. *Principles of Real Analysis*. Academic Press, London, 3 edition, 1998. ISBN 9780120502578. URL <http://www.amazon.com/dp/0120502577>.
- Tom M. Apostol. *Mathematical Analysis*. Addison-Wesley series in mathematics. Addison-Wesley, Reading, 2 edition, 1975. ISBN 986-154-103-9. URL <http://books.google.com/books?vid=ISBN0201002884>.
- Alexander Barvinok. *A Course in Convexity*, volume 54 of *Graduate studies in mathematics*. American Mathematical Society, 2002. ISBN 9780821872314. URL <http://books.google.com/books?vid=ISBN0821872311>.
- Ladislav Beran. *Orthomodular Lattices: Algebraic Approach*. Mathematics and Its Applications (East European Series). D. Reidel Publishing Company, Dordrecht, 1985. ISBN 90-277-1715-X. URL <http://books.google.com/books?vid=ISBN902771715X>.
- Mihaly Bessenyei and Zsolt Pales. A contraction principle in semimetric spaces. *arXiv.org*, January 8 2014. URL <http://arxiv.org/abs/1401.1709>.
- Garrett Birkhoff. On the combination of subalgebras. *Mathematical Proceedings of the Cambridge Philosophical Society*, 29:441–464, October 1933. URL <http://adsabs.harvard.edu/abs/1933MPCPS..29..441B>.
- Garrett Birkhoff. *Lattice Theory*. American Mathematical Society, New York, 2 edition, 1948. URL <http://books.google.com/books?vid=ISBN3540120440>.
- R. B. Blackman and J. W. Tukey. The measurement of power spectra from the point of view of communications engineering—part ii. *The Bell System Technical Journal*, 37:485–569, March 1958. URL <https://archive.org/download/bstj37-2-485>.
- R. B. Blackman and J. W. Tukey. *The Measurement of Power Spectra from the Point of View of Communications Engineering*. Dover Publications, New York, 1959. ISBN 486-60507-8. URL <https://archive.org/download/TheMeasurementOfPowerSpectra>. “unabridged and corrected republication of the work originally published in January and March, 1958, in Volume XXXVII of the *Bell System Technical Journal*”.
- Leonard Mascot Blumenthal. Distance geometries: a study of the development of abstract metrics. *The University of Missouri studies. A quarterly of research*, 13(2):145, 1938.

- Leonard Mascot Blumenthal. *Theory and Applications of Distance Geometry*. Oxford at the Clarendon Press, 1 edition, 1953. ISBN 0-8284-0242-6. URL <http://books.google.com/books?vid=ISBN0828402426>.
- Umberto Bottazzini. *The Higher Calculus: A History of Real and Complex Analysis from Euler to Weierstrass*. Springer-Verlag, New York, 1986. ISBN 0-387-96302-2. URL <http://books.google.com/books?vid=ISBN0387963022>.
- Bourbaki. *Éléments de mathématique. Première partie: Les structures fondamentales de l'analyse. Livre I: Théorie des ensembles (Fascicule des résultats)*. Hermann & Cie, Paris, 1939.
- Bourbaki. *Theory of Sets*. Elements of Mathematics. Hermann, Paris, 1968. URL <http://www.worldcat.org/oclc/281383>.
- Andrew M. Bruckner, Judith B. Bruckner, and Brian S. Thomson. *Real Analysis*. Prentice-Hall, Upper Saddle River, N.J., 1997. ISBN 9780134588865. URL <http://books.google.com/books?vid=ISBN013458886X>.
- Stanley Burris and Hanamantagida Pandappa Sankappanavar. A course in universal algebra. Re-typeset and corrected version of the 1981 edition, 2000. URL <http://www.math.uwaterloo.ca/~snburris/htdocs/ualg.html>.
- Herbert Busemann. *The Geometry of Geodesics*. Academic Press, 2 edition, 1955. ISBN 0486154629. URL <http://books.google.com/books?vid=ISBN0486154629>. a Dover 2005 edition has been published which "is an unabridged republication of the work originally published in 1955".
- Augustin-Louis Cauchy. *Part 1: Analyse Algébrique*. Cours D'Analyse de L'école Royale Polytechnique. Debure frères, Paris, 1821. ISBN 2-87647-053-5. URL <http://www.archive.org/details/coursdalanalyse00caucgoog>. Systems design course of the Polytechnic Royal School; 1st Part: Algebraic analysis).
- Paul M. Cohn. *Introduction to Ring Theory*. Springer Undergraduate Mathematics Series. Springer Science & Business Media, December 6 2012. ISBN 9781447104759. URL <http://books.google.com/books?vid=isbn9781447104759>.
- Louis Comtet. *Advanced combinatorics: the art of finite and infinite*. D. Reidel Publishing Company, Dordrecht, 1974. ISBN 978-9027704412. URL <http://books.google.com/books?vid=ISBN9027704414>. translated and corrected version of the 1970 French edition.
- Edward Thomas Copson. *Metric Spaces*. Number 57 in Cambridge tracts in mathematics and mathematical physics. Cambridge University Press, London, 1968. ISBN 978-0521047227. URL <http://books.google.com/books?vid=ISBN0521047226>.
- Nello Cristianini and Matthew W. Hahn. *Introduction to Computational Genomics: A Case Studies Approach*. Cambridge University Press, 2007. ISBN 9781139460156. URL <https://eembdersler.files.wordpress.com/2012/02/introduction-to-computational-genomics-a-case-studies-approach.pdf>.
- Sheldon W. Davis. *Topology*. McGraw Hill, Boston, 2005. ISBN 007-124339-9. URL <http://www.worldcat.org/isbn/0071243399>.
- Richard Dedekind. Ueber die von drei moduln erzeugte dualgruppe. *Mathematische Annalen*, 53: 371–403, January 8 1900. URL <http://resolver.sub.uni-goettingen.de/purl/?GDZPPN002257947>. Regarding the Dual Group Generated by Three Modules.
- Elena Deza and Michel-Marie Deza. *Dictionary of Distances*. Elsevier Science, Amsterdam, 2006. ISBN 0444520872. URL <http://books.google.com/books?vid=ISBN0444520872>.

- Michel-Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer, 2009. ISBN 3642002331. URL <http://www.uco.es/users/ma1fegan/Comunes/asignaturas/vision/Encyclopedia-of-distances-2009.pdf>.
- Michel-Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer, Bücher, 3 edition, 2014. ISBN 3662443422. URL <http://books.google.com/books?vid=ISBN3662443422>.
- Jean Alexandre Dieudonné. *Foundations of Modern Analysis*. Academic Press, New York, 1969. ISBN 1406727911. URL <http://books.google.com/books?vid=ISBN1406727911>.
- J. Alejandro Dominguez-Torres. The origin and history of convolution i: continuous and discrete convolution operations. online, November 02 2010. URL <http://www.slideshare.net/Alexdfar/origin-adn-history-of-convolution>.
- J. Alejandro Dominguez-Torres. A history of the convolution operation. *IEEE Pulse, Retroscope*, January/February 2015. URL <http://pulse.embs.org/january-2015/history-convolution-operation/>.
- W.D. Duthie. Segments of ordered sets. *Transactions of the American Mathematical Society*, 51(1):1–14, January 1942. . URL <http://www.jstor.org/stable/1989978>.
- Euclid. *Elements*. circa 300BC. URL <http://farside.ph.utexas.edu/euclid.html>.
- Abraham Fraenkel. Zu den grundlagen der cantor-zermeloschen mengenlehre. *Mathematische Annalen*, 86:230–237, 1922. URL <http://dz-srv1.sub.uni-goettingen.de/cache/toc/D36920.html>.
- Maurice René Fréchet. Sur quelques points du calcul fonctionnel (on some points of functional calculation). *Rendiconti del Circolo Matematico di Palermo*, 22:1–74, 1906. Rendiconti del Circolo Matematico di Palermo (Statements of the Mathematical Circle of Palermo).
- Maurice René Fréchet. *Les Espaces abstraits et leur théorie considérée comme introduction à l'analyse générale*. Borel series. Gauthier-Villars, Paris, 1928. URL <http://books.google.com/books?id=9czoHQAAACAAJ>. Abstract spaces and their theory regarded as an introduction to general analysis.
- Lorenzo Galleani and Roberto Garello. The minimum entropy mapping spectrum of a dna sequence. *IEEE Transactions on Information Theory*, 56(2):771–783, February 2010. ISSN 0018-9448. . URL <http://dx.doi.org/10.1109/TIT.2009.2037041>.
- Fred Galvin and Samuel David Shore. Completeness in semimetric spaces. *Pacific Journal Of Mathematics*, 113(1):67–75, March 1984. . URL <http://msp.org/pjm/1984/113-1/pjm-v113-n1-p04-s.pdf>.
- GenBank-AF086833.2. *Ebola virus—Mayinga, Zaire, 1976, complete genome*. NCBI: National Center for Biotechnology Information, Rockville Pike Bethesda MD 20894 USA, February 13 2013. URL <http://www.ncbi.nlm.nih.gov/nuccore/AF086833.2>. accession AF086833.2.
- GenBank-DS982815.1. *Carica papaya supercontig\_1446 genomic scaffold*. NCBI: National Center for Biotechnology Information, Rockville Pike Bethesda MD 20894 USA, March 11 2015. URL <http://www.ncbi.nlm.nih.gov/nuccore/DS982815.1>. accession DS982815.1.
- GenBank-NC\_004718.3. *SARS coronavirus, complete genome*. NCBI: National Center for Biotechnology Information, Rockville Pike Bethesda MD 20894 USA, September 02 2011. URL <http://www.ncbi.nlm.nih.gov/nuccore/30271926>. accession NC\_004718.3.
- GenBank-NZ\_CM003360.1. *Melissococcus plutonius strain 49.3 plasmid pMP19*. NCBI: National Center for Biotechnology Information, Rockville Pike Bethesda MD 20894 USA, August 21 2015. URL [http://www.ncbi.nlm.nih.gov/nuccore/NZ\\_CM003360.1](http://www.ncbi.nlm.nih.gov/nuccore/NZ_CM003360.1). accession NZ\_CM003360.1.

- John Robilliard Giles. *Introduction to the Analysis of Metric Spaces*. Number 3 in Australian Mathematical Society lecture series. Cambridge University Press, Cambridge, 1987. ISBN 0521359287. URL <http://books.google.com/books?vid=ISBN0521359287>.
- Daniel J. Greenhoe. *Wavelet Structure and Design*, volume 3 of *Mathematical Structure and Design series*. Abstract Space Publishing, August 2013. ISBN 9780983801139. URL <http://books.google.com/books?vid=ISBN0983801134>.
- Daniel J. Greenhoe. Properties of distance spaces with power triangle inequalities. *Carpathian Mathematical Publications*, 7(2), 2015a. ISSN 2313-0210. URL <http://www.researchgate.net/publication/281831459>.
- Daniel J. Greenhoe. Order and metric geometry compatible stochastic processing. *PeerJ PrePrints*, February 19 2015b. . URL <https://peerj.com/preprints/844/>. This paper has been submitted to the journal *Stochastic Systems* (<http://www.i-journals.org/ssy/>) on 2015 February 18 and as of 2015 November 19 (nine months) no editor decision has yet been received.
- Paul R. Halmos. *Measure Theory*. The University series in higher mathematics. D. Van Nostrand Company, New York, 1950. URL <http://www.amazon.com/dp/0387900888>. 1976 reprint edition available from Springer with ISBN 9780387900889.
- Paul Richard Halmos. *Naive Set Theory*. The University Series in Undergraduate Mathematics. D. Van Nostrand Company, Inc., Princeton, New Jersey, 1960. ISBN 0387900926. URL <http://books.google.com/books?vid=isbn0387900926>.
- Felix Hausdorff. *Set Theory*. Chelsea Publishing Company, New York, 3 edition, 1937. ISBN 0828401195. URL <http://books.google.com/books?vid=ISBN0828401195>. 1957 translation of the 1937 German *Grundzüge der Mengenlehre*.
- Robert W. Heath. A regular semi-metric space for which there is no semi-metric under which all spheres are open. *Proceedings of the American Mathematical Society*, 12:810–811, 1961. ISSN 1088-6826. URL <http://www.ams.org/journals/proc/1961-012-05/S0002-9939-1961-0125562-9/>.
- Chris J. Isham. *Modern Differential Geometry for Physicists*. World Scientific Publishing, New Jersey, 2 edition, 1999. ISBN 9810235623. URL <http://books.google.com/books?vid=ISBN9810235623>.
- John Leroy Kelley. *General Topology*. University Series in Higher Mathematics. Van Nostrand, New York, 1955. ISBN 0387901256. URL <http://books.google.com/books?vid=ISBN0387901256>. Republished by Springer-Verlag, New York, 1975.
- Mohamed A. Khamsi and W.A. Kirk. *An Introduction to Metric Spaces and Fixed Point Theory*. John Wiley, New York, 2001. ISBN 978-0471418252. URL <http://books.google.com/books?vid=isbn0471418250>.
- John R. Klauder. *A Modern Approach to Functional Integration*. Applied and Numerical Harmonic Analysis. Birkhäuser/Springer, 2010. ISBN 0817647902. URL <http://books.google.com/books?vid=isbn0817647902>. john.klauder@gmail.com.
- Andrei Nikolaevich Kolmogorov. *Foundations of the theory of probability*. Chelsea Publishing Company, New Yourk, 2 edition, 1933. ISBN B0006AUOGO. URL <http://statweb.stanford.edu/~cgates/PERSI/Courses/Phil166-266/Kolmogorov-Foundations.pdf>. 1956 2nd edition English translation of A. N. Kolmogorov's 1933 "Grundbegriffe der Wahrscheinlichkeitsrechnung".
- A. Korset. Bemerkung zur algebra der logik. *Mathematische Annalen*, 44(1):156–157, March 1894. ISSN 0025-5831. . URL <http://www.springerlink.com/content/v681m56871273j73/>. referenced by Birkhoff(1948)p.133.

- Carlos S. Kubrusly. *The Elements of Operator Theory*. Springer, 1 edition, 2001. ISBN 9780817641740.  
URL <http://books.google.com/books?vid=ISBN0817641742>.
- Kazimierz Kuratowski. Sur la notion d'ordre dans la theorie des ensembles (on the concept of order in set theory). *Fundamenta Mathematicae*, 2:161–171, 1921. URL <http://matwbn.icm.edu.pl/ksiazki/fm/fm2/fm2122.pdf>.
- Kazimierz Kuratowski. *Introduction to Set Theory and Topology*, volume 13 of *International Series on Monographs on Pure and Applied Mathematics*. Pergamon Press, New York, 1961. URL <http://www.worldcat.org/oclc/1023273>.
- Nicolas K. Laos. *Topics in Mathematical Analysis and Differential Geometry*, volume 24 of *Series in pure mathematics*. World Scientific, 1998. ISBN 9789810231804. URL <http://books.google.com/books?vid=ISBN9810231806>.
- Saunders MacLane and Garrett Birkhoff. *Algebra*. AMS Chelsea Publishing, Providence, 3 edition, 1999. ISBN 0821816462. URL <http://books.google.com/books?vid=isbn0821816462>.
- Roger Duncan Maddux. *Relation Algebras*. Elsevier Science, 1 edition, July 27 2006. ISBN 0444520139. URL <http://books.google.com/books?vid=ISBN0444520139>.
- Fumitomo Maeda and Shūichirō Maeda. *Theory of Symmetric lattices*, volume 173 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen*. Springer-Verlag, Berlin/New York, 1970. URL <http://books.google.com/books?id=4oeBAAAIAAJ>.
- Karl Menger. Untersuchungen über allgemeine metrik. *Mathematische Annalen*, 100:75–163, 1928. ISSN 0025-5831. URL <http://link.springer.com/article/10.1007/BF01455705>. (Investigations on general metric).
- Anthony N. Michel and Charles J. Herget. *Applied Algebra and Functional Analysis*. Dover Publications, Inc., 1993. ISBN 0-486-67598-X. URL <http://books.google.com/books?vid=ISBN048667598X>. original version published by Prentice-Hall in 1981.
- Gregory K. Miller. *Probability: Modeling and Applications to Random Processes*. Wiley, August 25 2006. URL <http://www.amazon.com/dp/0471458929>.
- Ilya S. Molchanov. *Theory of Random Sets*. Probability and Its Applications. Springer, 2005. ISBN 185233892X. URL <http://books.google.com/books?vid=ISBN185233892X>.
- James R. Munkres. *Topology*. Prentice Hall, Upper Saddle River, NJ, 2 edition, 2000. ISBN 0131816292. URL <http://www.amazon.com/dp/0131816292>.
- Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, 2 edition, 1999. ISBN 9780137549207. URL <http://www.amazon.com/dp/0137549202>.
- Oystein Ore. On the foundation of abstract algebra. i. *The Annals of Mathematics*, 36(2):406–437, April 1935. URL <http://www.jstor.org/stable/1968580>.
- Endre Pap. *Null-Additive Set Functions*, volume 337 of *Mathematics and Its Applications*. Kluwer Academic Publishers, 1995. ISBN 0792336585. URL <http://www.amazon.com/dp/0792336585>.
- Anthanassios Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York, 3 edition, 1991. ISBN 0-07-048477-5. URL <http://books.google.com.tw/books?vid=ISBN0070484775>.
- Charles Sanders Peirce. Note b: the logic of relatives. In *Studies in Logic by Members of the Johns Hopkins University*, pages 187–203. Little, Brown, and Co., Boston, 1883. URL <http://www.archive.org/details/studiesinlogic00peiruoft>.

- K. M. M. Prabhu. *Window Functions and Their Applications in Signal Processing*. CRC Press, 2013. ISBN 9781466515840. URL <https://books.google.com/books?vid=ISBN1466515848>.
- John Ratcliffe. *Foundations of Hyperbolic Manifolds*, volume 149 of *Graduate Texts in Mathematics*. Springer Science & Business Media, 2013. ISBN 9781475740134. URL <http://books.google.com/books?vid=ISBN1475740131>.
- David Simmons. Sequence. Technical report, University of North Texas, Department of Mathematics, 1155 Union Circle #311430, Denton, TX 76203, January 21 15:38 2016. URL <https://en.wikipedia.org/w/index.php?title=Sequence&oldid=700944642>.
- Rajeev Singh, Ramon E. Vasquez, and Reena Singh. Comparison of daubechies, coiflet, and symlet for edge detection. In Stephen K. Park and Richard D. Juday, editors, *SPIE Proceedings*, volume 3074 of *Visual Information Processing VI*, July 22 1997. . URL <http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=926491>.
- Lynn Arthur Steen and J. Arthur Seebach. *Counterexamples in Topology*. Springer-Verlag, 2, revised edition, 1978. URL <http://books.google.com.tw/books?vid=ISBN0486319296>. A 1995 “unabridged and unaltered republication” Dover edition is available.
- Patrick Suppes. *Axiomatic Set Theory*. Dover Publications, New York, 1972. ISBN 0486616304. URL <http://books.google.com/books?vid=ISBN0486616304>.
- Wolfgang J. Thron. *Topological structures*. Holt, Rinehart and Winston, New York, 1966. URL [http://books.google.com/books?id=JRM\\_AAAIAAJ](http://books.google.com/books?id=JRM_AAAIAAJ).
- Richard F. Voss. Evolution of long-range fractal correlations and 1/f noise in dna base sequences. *Physical Review Letters*, 68(25):3805–3808, June 22 1992. . URL <http://journals.aps.org/prl/abstract/10.1103/PhysRevLett.68.3805>.
- Norbert Wiener. A simplification of the logic of relations. *Proceedings of the Cambridge Philosophical Society*, 17:387–390, 1914. URL <http://books.google.com/books?vid=ISBN0674324498&pg=PA224>. reprinted in Jean Van Heijenoort's *From Frege to Gödel : A Source Book in Mathematical Logic, 1879-1931* page 224–227.
- Wallace Alvin Wilson. On semi-metric spaces. *American Journal of Mathematics*, 53(2):361–373, April 1931. URL <http://www.jstor.org/stable/2370790>.
- Robert S. Wolf. *Proof, Logic, and Conjecture; The Mathematician's Toolbox*. W.H. Freeman and Company, 1998. ISBN 0716730502. URL <http://www.worldcat.org/isbn/0716730502>.
- Ernst Zermelo. Untersuchungen über die grundlagen der mengenlehre i. *Mathematische Annalen*, 65: 261–281, 1908a. URL <http://dz-srv1.sub.uni-goettingen.de/sub/digbib/loader?ht=VIEW&did=D38200>.
- Ernst Zermelo. Investigations in the foundations of set theory. In Jean Van Heijenoort, editor, *From Frege to Gödel : A Source Book in Mathematical Logic, 1879-1931*, pages 199–215. Harvard University Press, 1967, Cambridge, Massachusetts, 1908b. URL <http://www.amazon.com/dp/0674324498>.

## Reference Index

- Ratcliffe (2013), 46  
 Aliprantis and Burkinshaw (1998), 45  
 Apostol (1975), 3, 5, 8  
 Barvinok (2002), 5  
 Beran (1985), 5  
 Bessenyei and Pales (2014), 45  
 Birkhoff (1933), 6  
 Birkhoff (1948), 6  
 Blackman and Tukey (1958), 9  
 Blackman and Tukey (1959), 9  
 Blumenthal (1938), 45  
 Blumenthal (1953), 45  
 Bottazzini (1986), 4  
 Bourbaki (1939), 4  
 Bourbaki (1968), 3  
 Bruckner et al. (1997), 45  
 Burris and Sankappanavar (2000), 6  
 Busemann (1955), 46  
 Cauchy (1821), 8  
 Cohn (2012), 3  
 Comtet (1974), 4  
 Copson (1968), 45, 46  
 Cristianini and Hahn (2007), 43  
 Davis (2005), 3  
 Dedekind (1900), 5  
 Deza and Deza (2006), 46  
 Deza and Deza (2009), 46  
 Deza and Deza (2014), 45, 46  
 Dieudonné (1969), 46  
 Dominguez-Torres (2010), 8  
 Dominguez-Torres (2015), 8  
 Duthie (1942), 5  
 Euclid (circa 300BC), 46  
 Fraenkel (1922), 3  
 Fréchet (1906), 46  
 Fréchet (1928), 46  
 Galleani and Garello (2010), 21, 22, 39  
 Galvin and Shore (1984), 45  
 Giles (1987), 45, 46  
 Greenhoe (2013), 40, 41  
 Greenhoe (2015a), 45  
 Greenhoe (2015b), 1, 11–14  
 Halmos (1950), 5  
 Halmos (1960), 3, 4  
 Hausdorff (1937), 3, 45, 46  
 Heath (1961), 45  
 Isham (1999), 46  
 Kelley (1955), 3, 4  
 Khamsi and Kirk (2001), 45, 46  
 Klauder (2010), 14  
 Kolmogorov (1933), 10  
 Korset (1894), 5  
 Kubrusly (2001), 45  
 Kuratowski (1921), 3  
 Kuratowski (1961), 3  
 Laos (1998), 45  
 MacLane and Birkhoff (1999), 5, 6  
 Maddux (2006), 4  
 Maeda and Maeda (1970), 6  
 Menger (1928), 45  
 Michel and Herget (1993), 45  
 Miller (2006), 11  
 Molchanov (2005), 45  
 Munkres (2000), 3, 4  
 GenBank-AF086833.2 (2013), 19  
 GenBank-NZ\_CM003360.1 (2015), 19  
 GenBank-DS982815.1 (2015), 19  
 GenBank-NC\_004718.3 (2011), 18  
 Oppenheim and Schafer (1999), 9  
 Ore (1935), 5, 6  
 Pap (1995), 5  
 Papoulis (1991), 10, 11  
 Peirce (1883), 4  
 Prabhu (2013), 9  
 Simmons (2016), 7  
 Singh et al. (1997), 41  
 Steen and Seebach (1978), 46  
 Suppes (1972), 3, 4  
 Thron (1966), 45, 46  
 Voss (1992), 22  
 Wiener (1914), 3  
 Wilson (1931), 45  
 Wolf (1998), 3  
 Zermelo (1908b), 3  
 Zermelo (1908a), 3

## Subject Index

$2N$ -offset autocorrelation, 16, 19, 24  
 $\mathbb{R}^1$  die distance linear space, 22, 24  
 $\mathbb{R}^1$  spinner random variable, 21, 26  
 $\mathbb{R}^2$  spinner distance linear space, 22, 23, 26  
 $\mathbb{R}^3$  die distance linear space, 22, 22, 23  
 $\mathbb{R}^3$  die random variable, 21, 22, 24, 30  
 $\mathbb{R}^3$  distance linear space, 24, 30  
 $\mathbb{R}^4$  DNA random variable, 22, 39, 40, 43, 44  
 $\mathbb{R}^6$  die distance linear space, 23, 28  
 $\mathbb{R}^6$  die random variable, 21, 23,

28, 36–38, 42  
 $\mathbb{R}^6$  fair die distance linear space, 23  
 $M$ -offset autocorrelation, 15  
 GLB, 5  
 LUB, 5  
 $\sigma$ -algebra, 11  
 3-tuple, 4  
 addition operator, 7, 7, 8, 10  
 additive, 10  
 algebra of set, 14  
 anti-symmetric, 5  
 arc, 47  
 autocorrelation, 15, 15, 24  
 Bacterium DNA sequence, 19  
 base, 45  
 bijective, 6

bound  
 greatest lower bound, 5  
 infimum, 5  
 least upper bound, 5  
 supremum, 5  
 bounded, 45  
 Bourbaki notation, 3  
 cardinality of  $A$ , 5  
 carica papaya, 19  
 Cartesian product, 4  
 Cauchy, 45  
 ceiling function, 6  
 center, 1, 14  
 closed ball, 45  
 closed interval, 5  
 comparable, 5  
 continuous, 45, 48



- convergence, 14  
 convergent, 45  
 converse, 4  
 Convex, 5  
 convex, 5, 7, 48  
 convex subset, 8, 15  
 convolution, 8, 8, 9  
 cross-correlation, 15
- definitions  
 $\mathbb{R}^1$  die distance linear space, 22  
 $\mathbb{R}^2$  spinner distance linear space, 22  
 $\mathbb{R}^3$  die distance linear space, 22  
 $\mathbb{R}^6$  die distance linear space, 23  
 Cartesian product, 4  
 closed ball, 45  
 closed interval, 5  
 distance space, 45  
 DNA outcome subspace, 13  
 DNA scaffold outcome subspace, 13  
 empty set, 3  
 extended probability space, 11  
 fair die outcome subspace, 12  
 function, 4  
 genome outcome subspace, 13  
 genome scaffold outcome subspace, 13  
 isometry, 46  
 lattice, 6  
 n-tuple, 4  
 open ball, 45  
 open interval, 5  
 ordered distance space, 11  
 ordered metric space, 11  
 ordered pair, 3  
 ordered set, 5  
 ordered triple, 4  
 outcome subspace, 11  
 partially ordered set, 5  
 poset, 5  
 probability space, 11  
 real die outcome subspace, 12  
 relation, 4  
 sequence, 7  
 set of all functions, 4  
 set of all relations, 4
- set of complex numbers, 4  
 set of integers, 3  
 set of natural numbers, 3  
 set of non-negative real numbers, 3  
 set of outcomes, 11  
 set of positive real numbers, 3  
 set of real numbers, 3  
 set of whole numbers, 3  
 spinner outcome subspace, 13  
 triple, 4  
 unordered set, 5  
 weighted die outcome subspace, 12  
 weighted real die outcome subspace, 12  
 diameter, 45  
 die outcome subspace, 15  
 die sequence, 24, 28, 30  
 Discrete Fourier Transform, 35–38  
 discrete Fourier transform, 10  
 discrete metric, 12, 13, 15, 46, 46  
 distance, 14, 15, 20, 45, 45, 47  
 distance function, 14  
 distance space, 1, 11, 14, 45, 45  
 DNA, 12  
 DNA outcome subspace, 13  
 DNA scaffold outcome subspace, 13  
 domain, 4, 8, 9  
 down sample, 30, 33  
 down sampled by a factor of  $M$ , 7
- Ebola virus DNA sequence, 19  
 empty set, 3  
 Euclidean metric, 11, 22–33, 35, 46, 48, 49  
 events, 11  
 examples  
     Bacterium DNA sequence, 19  
     Ebola virus DNA sequence, 19  
         fair die sequence, 15  
         Fourier analysis of Ebola DNA sequence, 39  
         high pass filtering of weighted die sequence, 33  
         high pass filtering of weighted real die sequence, 29  
         high pass filtering of
- weighted spinner sequence, 31  
 Lagrange arc distance in  $\mathbb{R}^2$ , 47  
 Lagrange arc distance in  $\mathbb{R}^3$ , 48  
 Lagrange arc distance versus Euclidean metric, 48  
 length 1200 non-stationary die sequence with 10Hz oscillating mean, 35  
 length 12000 non-stationary artificial DNA sequence with 10Hz oscillating mean, 38  
 length 12000 non-stationary die sequence with 100Hz oscillating mean, 37  
 length 12000 non-stationary die sequence with 10Hz oscillating mean, 36  
 low pass filtering of fair die sequence, 27  
 low pass filtering of real die sequence, 23  
 low pass filtering of spinner sequence, 26  
 Papaya DNA sequence, 19  
 Random DNA sequence, 18  
 real die mappings, 14  
 real die sequence, 16  
 SARS coronavirus DNA sequence, 18  
 spinner sequence, 16  
 statistical edge detection using Haar wavelet on non-stationary artificial DNA sequence, 42  
 statistical edge detection using Haar wavelet on non-stationary die sequence, 41  
 Wavelet analysis of Phage Lambda DNA sequence, 43  
 weighted die sequence, 17, 18  
 weighted real die sequence, 17  
 weighted spinner sequence, 18, 18  
 expectation, 1, 14  
 expected value, 12  
 extended probability space, 11, 11, 12  
 extended set of integers, 3  
 extension, 20

- fair die, 12, 15  
 fair die outcome space, 29  
 fair die outcome subspace, 12, 23  
 fair die sequence, 15  
 Fast Wavelet Transform, 40  
 field, 1, 8, 9, 14, 15  
 field of real numbers, 7  
 Filter, 24, 26, 28, 30  
 filter, 9, 23, 26, 28, 30, 31, 33, 41  
 filter bank, 40  
 filtered, 9  
 finite, 5  
 FIR filtering, 1, 14  
 floor function, 6  
 Fourier analysis, 1, 14  
 Fourier analysis of Ebola DNA sequence, 39  
 Fréchet product metric, 15, 46  
 function, 1, 4, 7, 11, 12, 22, 46  
 functions  
      $\mathbb{R}^3$  die random variable, 22  
      $\mathbb{R}^4$  DNA random variable, 39, 40, 43, 44  
      $\mathbb{R}^6$  die random variable, 23, 36–38, 42  
         autocorrelation, 15  
         cardinality of  $A$ , 5  
         ceiling function, 6  
         center, 1, 14  
         cross-correlation, 15  
         die sequence, 24, 28, 30  
         discrete metric, 12, 13, 15,  
**46**  
         distance, 14, 15, 20, 45, 45,  
**47**  
         distance function, 14  
         Euclidean metric, 11, 22–  
 33, 35, 46, 48, 49  
         expectation, 1, 14  
         floor function, 6  
         great circle metric, 46  
         isometry, 22, 23, 45  
         isomorphism, 6  
         Lagrange arc distance, 22,  
 24–27, 29–33, 35, 46–49  
         Lagrange arc distance  
         space, 48  
         Lagrange interpolation,  
**47**  
         length  $M$  Haar scaling se-  
 quence, 10  
         length  $M$  Haar wavelet se-  
 quence, 10  
         length  $M$  high pass Han-  
         ning sequence, 10  
         length  $M$  high pass rect-  
         angular sequence, 9  
         length  $M$  low pass Han-  
         ning sequence, 9  
         length  $M$  low pass rectan-  
         gular sequence, 9  
         length 16 Hanning low  
         pass sequence, 25, 27, 28  
         length 16 high pass rect-  
         angular sequence, 9  
         length 16 low pass rectan-  
         gular sequence, 9  
         length 16 Rectangular low  
         pass sequence, 27, 28  
         length 16 rectangular low  
         pass sequence, 24, 26, 28  
         length 1600 Haar scaling  
         sequence, 43  
         length 200 Haar wavelet  
         sequence, 41, 42  
         length 50 Hanning low  
         pass sequence, 25, 27, 29  
         length 50 high pass Han-  
         ning sequence, 10  
         length 50 high pass rect-  
         angular sequence, 31  
         length 50 low pass Han-  
         ning sequence, 10  
         length 8 Haar scaling se-  
         quence, 10  
         length 8 Haar wavelet se-  
         quence, 10  
         length 9 Haar wavelet se-  
         quence, 10  
         low pass rectangular se-  
         quence, 26  
         low pass sequence, 23, 28  
         metric, 12–15, 46, 47, 48  
         norm, 48  
         outcome expected value,  
**14**  
         outcome subspace se-  
 quence metric, 15, 15  
         outcome variance, 14  
         PAM die random variable,  
 35–37, 41  
         PAM DNA random vari-  
 able, 38, 39, 42, 43  
         polar coordinates, 47  
         probability function, 10,  
**13**  
         QPSK die random vari-  
 able, 36, 37, 41  
         QPSK DNA random vari-  
 able, 38, 40, 43, 44  
         QPSK spinner random  
         variable, 23  
         random variable, 12, 14,  
 20, 21  
         RMS, 39, 40, 42  
         root mean square, 39  
         scaling function, 40  
         sequence, 15, 26, 31, 45  
         set function, 5  
         spherical metric, 46  
         spinner sequence, 26, 31  
         traditional die random  
         variable, 33  
         traditional random vari-  
 able, 1, 11, 11  
         usual metric, 1, 11  
         wavelet, 40  
         weighted die sequence, 33
- GenBank, 18  
 genome outcome subspace, 13  
 genome scaffold outcome sub-  
 space, 13  
 GNU Octave  
     break, 18, 58–60, 64, 71,  
 78–80, 82–84, 86, 87  
     case, 18, 58–60, 64, 71, 78–  
 80, 82–84, 86, 87  
     char, 49–86  
     class, 49, 55, 61, 68, 75  
     const, 49–58, 61–63, 68–  
 70, 75–82, 85, 86  
     default, 58–60, 64, 71, 78–  
 80, 82–84, 87  
     double, 54–59, 61–69, 71–  
 75, 81–85  
     else, 17, 18, 51, 52, 56, 67,  
 68, 74, 75, 77, 84–86  
     extern, 49, 55, 61, 62, 68,  
**75**  
     for, 15–18, 50–54, 56–58,  
 62–67, 69–74, 77–86  
     if, 17, 18, 50–54, 56, 58, 62–  
 70, 72–75, 77, 81–86  
     int, 49–57, 61–73, 75, 77,  
 81, 83–86  
     long, 49–58, 61–86  
     operator, 49, 53, 55, 58, 61,  
 63, 68, 70, 75, 82  
     nolig, 49, 61  
     private, 49  
     public, 49, 55, 61, 68, 75  
     return, 49–51, 53–60, 62–  
 75, 78–87  
     sizeof, 50  
     switch, 18, 58–60, 64, 71,  
 78–80, 82, 83, 86

- unsigned, 49, 50, 55, 61, 68, 75  
void, 49–58, 61–63, 68–70, 75–82  
while, 86
- graph, 11
- great circle metric, 46
- greatest lower bound, 5
- high pass filtering of weighted die sequence, 33
- high pass filtering of weighted real die sequence, 29
- high pass filtering of weighted spinner sequence, 31
- homogeneous, 48
- IIR filtering, 1, 14
- image set, 4
- incomparable, 5, 22, 23
- integer line, 11, 20
- inverse, 4
- isometric, 1, 11, 20, 22, 23, 45
- isometry, 22, 23, 45, 46
- isomorphic, 1, 6, 6, 11, 20, 23
- isomorphism, 6
- join, 5
- juxtaposition, 7
- Kuratowski, 3
- Lagrange arc distance, 22, 24–27, 29–33, 35, 46–49
- Lagrange arc distance in  $\mathbb{R}^2$ , 47
- Lagrange arc distance in  $\mathbb{R}^3$ , 48
- Lagrange arc distance space, 48
- Lagrange arc distance versus Euclidean metric, 48
- Lagrange interpolation, 47
- lattice, 6, 6
- isomorphic, 6
- least upper bound, 5
- length  $M$  Haar scaling sequence, 10
- length  $M$  Haar wavelet sequence, 10
- length  $M$  high pass Hanning sequence, 10
- length  $M$  high pass rectangular sequence, 9
- length  $M$  low pass Hanning sequence, 9
- length  $M$  low pass rectangular sequence, 9
- length 1200 non-stationary die sequence with 10Hz oscillating mean, 35
- length 12000 non-stationary artificial DNA sequence with 10Hz oscillating mean, 38
- length 12000 non-stationary die sequence with 100Hz oscillating mean, 37
- length 12000 non-stationary die sequence with 10Hz oscillating mean, 36
- length 16 Hanning low pass sequence, 24, 25, 27, 28
- length 16 Hanning sequence, 34
- length 16 high pass rectangular sequence, 9, 33
- length 16 low pass rectangular sequence, 9
- length 16 Rectangular low pass sequence, 27, 28
- length 16 rectangular low pass sequence, 24, 26, 28
- length 16 rectangular sequence, 34
- length 1600 Haar scaling sequence, 43
- length 200 Haar wavelet sequence, 41, 42
- length 50 Hanning high pass filter, 30, 32
- length 50 Hanning low pass sequence, 25, 27–29
- length 50 Hanning sequence, 34
- length 50 high pass Hanning sequence, 10
- length 50 high pass rectangular sequence, 30, 31
- length 50 low pass Hanning sequence, 10
- length 50 rectangular high pass filter, 30, 32
- length 50 rectangular high pass sequence, 30
- length 50 Rectangular low pass sequence, 27, 29
- length 50 rectangular low pass sequence, 25
- length 50 rectangular sequence, 34
- length 8 Haar scaling sequence, 10
- length 8 Haar wavelet sequence, 10
- length 9 Haar wavelet sequence, 10
- linear, 6
- linear space, 1, 11, 14, 23, 47
- low pass filtering of fair die sequence, 27
- low pass filtering of real die sequence, 23
- low pass filtering of spinner sequence, 26
- low pass rectangular sequence, 26
- low pass sequence, 23, 28
- lower bound, 5, 5
- meet, 5
- Melissococcus plutonius strain 49.3 plasmid pMP19, 19
- metric, 11–15, 20, 46, 46, 47, 48
- Metric ball, 14
- metric space, 1, 14, 45, 46
- metrics, 46
- discrete, 46
- multiplication operation, 7
- multiplication operator, 7, 8, 10
- n-tuple, 4
- non-negative, 45, 46
- nondegenerate, 45, 46
- nonnegative, 10
- norm, 48
- normalized, 10
- normed linear space, 48
- not convex, 48
- not isomorphic, 6, 22, 23
- not order preserving, 6, 22, 23
- not translation invariant, 48
- null space, 4
- open, 45
- open ball, 45, 45
- open interval, 5
- operations
- $2N$ -offset autocorrelation, 16, 19, 24
- $\mathbb{R}^1$  spinner random variable, 21, 26
- $\mathbb{R}^3$  die random variable, 21, 24, 30
- $\mathbb{R}^4$  DNA random variable, 22
- $\mathbb{R}^6$  die random variable, 21, 28
- $M$ -offset autocorrelation, 15

## SUBJECT INDEX

Daniel J. Greenhoe

page 97

- addition operator, 7, 7, 8, 10  
 autocorrelation, 15, 24  
 convolution, 8, 8, 9  
 Discrete Fourier Transform, 35–38  
 discrete Fourier transform, 10  
     down sample, 30, 33  
     down sampled by a factor of  $M$ , 7  
     expected value, 12  
     Fast Wavelet Transform, 40  
     Filter, 24, 26, 28, 30  
     filter, 9, 23, 26, 28, 30, 31, 33, 41  
     filter bank, 40  
     filtered, 9  
     FIR filtering, 1, 14  
     Fourier analysis, 1, 14  
     IIR filtering, 1, 14  
     join, 5  
     juxtaposition, 7  
     Lagrange interpolation, 47  
     length 16 Hanning low pass sequence, 24, 27, 28  
     length 16 Hanning sequence, 34  
     length 16 high pass rectangular sequence, 33  
     length 16 rectangular low pass sequence, 24, 26, 28  
     length 16 rectangular sequence, 34  
     length 50 Hanning high pass filter, 30, 32  
     length 50 Hanning low pass sequence, 25, 27, 28  
     length 50 Hanning sequence, 34  
     length 50 high pass rectangular sequence, 30  
     length 50 rectangular high pass filter, 30, 32  
     length 50 rectangular high pass sequence, 30  
     length 50 Rectangular low pass sequence, 27, 29  
     length 50 rectangular low pass sequence, 25  
     length 50 rectangular sequence, 34  
     meet, 5  
     multiplication operation, 7  
     multiplication operator, 7, 8, 10  
     PAM die random variable, 21  
     PAM DNA random variable, 21  
     pulse amplitude modulation, 21  
     QPSK die random variable, 21  
     QPSK DNA random variable, 22  
     QPSK spinner random variable, 21, 26  
     quadrature phase shift keying, 21, 22  
     sampled, 40  
     set difference, 5  
     traditional die random variable, 21, 24, 28  
     Voss Spectrum, 22  
     wavelet analysis, 1, 14  
 order, 20  
 order preserving, 6, 6, 22, 23  
 order relation, 5, 5  
 ordered distance linear space, 1, 14, 20  
 ordered distance space, 11, 11  
 ordered metric space, 11, 13  
 ordered pair, 3, 3  
 ordered set, 1, 5, 5, 11, 14  
 ordered triple, 4  
 outcome expected value, 14  
 outcome subspace, 11, 12, 13, 15, 20  
 outcome subspace sequence, 15  
 outcome subspace sequence metric, 15, 15  
 outcome variance, 14  
 outcomes, 11  
 PAM die random variable, 21, 35–37, 41  
 PAM DNA random variable, 21, 38, 39, 42, 43  
 Papaya DNA sequence, 19  
 partial order relation, 5  
 partially ordered set, 5  
 polar coordinates, 47  
 poset, 5  
     order preserving, 6  
 power set, 45  
 power set of  $X$ , 4  
 preorder, 5  
 preserves joins, 6  
 preserves meets, 6  
 probability function, 10, 11, 13  
 probability measure, 11  
 probability space, 11, 11  
 properties  
     additive, 10  
     anti-symmetric, 5  
     bijective, 6  
     bounded, 45  
     Cauchy, 45  
     comparable, 5  
     continuous, 45, 48  
     convergence, 14  
     convergent, 45  
     Convex, 5  
     convex, 5, 7, 48  
     extension, 20  
     finite, 5  
     Fréchet product metric, 46  
     homogeneous, 48  
     incomparable, 5, 22, 23  
     isometric, 1, 11, 20, 22, 23, 45  
     isomorphic, 1, 6, 6, 11, 20, 23  
     linear, 6  
     metric, 20  
     non-negative, 45, 46  
     nondegenerate, 45, 46  
     nonnegative, 10  
     normalized, 10  
     not convex, 48  
     not isomorphic, 6, 22, 23  
     not order preserving, 6, 22, 23  
     not translation invariant, 48  
     open, 45  
     order, 20  
     order preserving, 6, 6, 22, 23  
 preserves joins, 6  
 preserves meets, 6  
 reflexive, 5  
 subadditive, 46  
 symmetric, 45, 46  
 transitive, 5  
 translation invariant, 48  
 triangle inequality, 31, 46, 48  
 triangle inequality fails, 48  
 uncorrelated, 16, 18, 30  
 uniformly distributed, 16, 18

- unique, 45  
unordered, 12, 13, 22  
pulse amplitude modulation, 21
- QPSK die random variable, **21**, 36, 37, 41  
QPSK DNA random variable, **22**, 38, 40, 43, 44  
QPSK spinner random variable, **21**, 23, 26  
quadrature phase shift keying, 21, 22  
quotient structures, 5
- Random DNA sequence, **18**  
random variable, **12**, 14, 20, 21  
range, 4, 18  
real die, 12, 16, 17, 20  
real die mappings, **14**  
real die outcome subspace, **12**, 13, 14, 20, 22  
real die sequence, **16**  
real line, 1, 11, 14, 20  
reflexive, 5  
relation, 4, **4**, 22  
    inverse, 4  
relations  
    converse, 4  
    domain, 4  
    image set, 4  
    inverse, 4  
    null space, 4  
    order relation, **5**  
    partial order relation, **5**  
    preorder, 5  
    range, 4  
    standard ordering relation, 22  
Reproducible Research, 49  
RMS, 39, 40, 42  
root mean square, 39
- sampled, 40  
SARS coronavirus DNA sequence, **18**  
scaffold DNA, 12  
scaling function, 40  
sequence, 1, **7**, 8–10, 14, 15, 26, 31, 45  
set, 3–5, 7, 10–12  
set difference, 5  
set function, 5  
set of all functions, **4**  
set of all relations, **4**  
set of complex numbers, **4**  
set of integers, **3**
- set of natural numbers, **3**  
set of non-negative real numbers, **3**  
set of outcomes, **11**, **11**  
set of positive real numbers, **45**  
set of postive real numbers, **3**  
set of real numbers, **1**, **3**  
set of whole numbers, **3**  
sets, 4  
sigma-algebra, 14  
source code, 49  
spherical metric, 46  
spinner, 12, 16  
spinner outcome subspace, **13**, 23, 26  
spinner sequence, **16**, 26, 31, 32  
standard ordered set of real numbers, 6  
standard ordering relation, 22  
statistical edge detection using Haar wavelet on non-stationary artificial DNA sequence, **42**  
statistical edge detection using Haar wavelet on non-stationary die sequence, **41**  
stochastic process, 1, 11  
structures  
     $\mathbb{R}^1$  die distance linear space, **22**, 24  
     $\mathbb{R}^2$  spinner distance linear space, **22**, 23, 26  
     $\mathbb{R}^3$  die distance linear space, **22**, **22**, 23  
     $\mathbb{R}^3$  distance linear space, 24, 30  
     $\mathbb{R}^6$  die distance linear space, **23**, 28  
     $\mathbb{R}^6$  fair die distance linear space, 23  
     $\sigma$ -algebra, 11  
    3-tuple, 4  
    algebra of set, 14  
    arc, 47  
    base, 45  
    Cartesian product, 4  
    closed ball, **45**  
    closed interval, **5**  
    convex subset, 8, 15  
    die outcome subspace, 15  
    die sequence, 30  
    distance space, 1, 11, 14, **45**, **45**  
    DNA, 12  
    DNA outcome subspace,
- 13**  
DNA scaffold outcome subspace, **13**  
domain, 8, 9  
empty set, **3**  
extended probability space, **11**, **11**, 12  
extended set of integers, 3  
fair die, 12, 15  
fair die outcome space, 29  
fair die outcome subspace, **12**, 23  
field, 1, 8, 9, 14, 15  
field of real numbers, 7  
function, 1, **4**, 7, 11, 12, 22, 46  
genome outcome subspace, **13**  
genome scaffold outcome subspace, **13**  
graph, 11  
integer line, 11, 20  
Lagrange arc distance space, 48  
lattice, **6**, **6**  
linear space, 1, 11, 14, 23, 47  
metric, 11, 46  
Metric ball, 14  
metric space, 1, 14, 45, 46  
metrics, 46  
n-tuple, **4**  
normed linear space, 48  
open ball, 45, **45**  
open interval, **5**  
order relation, 5  
ordered distance linear space, 1, 14, 20  
ordered distance space, 11, **11**  
ordered metric space, **11**, 13  
ordered pair, 3, **3**  
ordered set, 1, 5, **5**, 11, 14  
ordered triple, **4**  
outcome subspace, **11**, 12, 13, 15, 20  
outcome subspace sequence, 15  
partially ordered set, **5**  
poset, **5**  
power set, 45  
power set of  $X$ , 4  
probability function, 11  
probability space, 11, **11**  
range, 18

- real die, 12, 16, 17, 20  
 real die outcome subspace, **12**, 13, 14, 20, 22  
 real line, 1, 11, 14, 20  
 relation, 4, **4**, 22  
 scaffold DNA, 12  
 sequence, 1, **7**, 8–10, 14,  
**15**  
 set, 3–5, 7, 10–12  
 set function, 5  
 set of all functions, **4**  
 set of all relations, **4**  
 set of complex numbers, **4**  
 set of integers, **3**  
 set of natural numbers, **3**  
 set of non-negative real numbers, **3**  
 set of outcomes, 11, **11**  
 set of positive real numbers, 45  
 set of positive real numbers, **3**  
 set of real numbers, 1, **3**  
 set of whole numbers, **3**  
 sets, 4  
 sigma-algebra, 14  
 spinner, 12, 16  
 spinner outcome subspace, **13**, 23, 26  
 spinner sequence, 31, 32  
 standard ordered set of real numbers, 6
- stochastic process, 1, 11  
 topology, 14  
 triple, **4**  
 unordered set, **5**  
 weighted die, 12, 17, 18  
 weighted die outcome subspace, **12**, **12**  
**20**  
 weighted graph, 1, 11–14,  
 weighted graphs, 20  
 weighted real die, 12, 17,  
**18**  
 weighted real die outcome subspace, **12**, **12**  
 subadditive, 46  
 symmetric, 45, 46
- theorems  
 Fréchet product metric, **15**  
 topology, 14  
 traditional die random variable, **21**, 24, 28, 33  
 traditional random variable, 1, **11**, **11**  
 transitive, 5  
 translation invariant, 48  
 triangle inequality, 31, 46, 48  
 triangle inequality fails, 48  
 triple, **4**  
 uncorrelated, 16, 18, 30  
 uniformly distributed, 16, 18
- unique, 45  
 unordered, 12, 13, 22  
 unordered set, **5**  
 upper bound, 5, **5**  
 usual metric, 1, 11
- values  
**GLB**, **5**  
**LUB**, **5**  
 diameter, **45**  
 greatest lower bound, **5**  
 least upper bound, **5**  
 lower bound, 5, **5**  
 upper bound, 5, **5**  
 Voss Spectrum, 22
- wavelet, 40  
 wavelet analysis, 1, 14  
 Wavelet analysis of Phage Lambda DNA sequence, **43**  
 weighted die, 12, 17, 18  
 weighted die outcome subspace, **12**, **12**  
 weighted die sequence, **17**, 18, 33  
 weighted graph, 1, 11–14, 20  
 weighted graphs, 20  
 weighted real die, 12, 17, 18  
 weighted real die outcome subspace, **12**, **12**  
 weighted real die sequence, **17**  
 weighted spinner sequence, **18**, **18**

Telecommunications Engineering Department, National Chiao-Tung University, Hsinchu, Taiwan; 國立交通大學 (Gúo Lì Jiāo Tōng Dà Xué) 電信工程學系 (Diàn Xīn Gōng Chéng Xué Xì) 新竹, 台灣 (Xīn Zhú, Tái Wān)

dgreenhoe@gmail.com

Received: aa bb 20YY      Revised: cc dd 20ZZ