Chapter 15

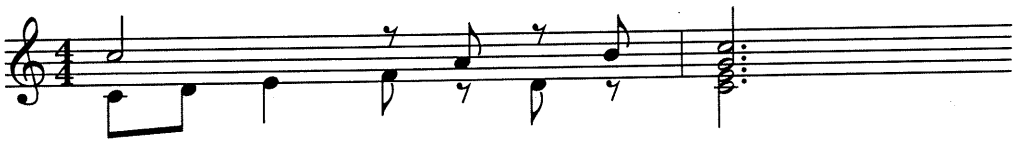
Harmonic Intervals

In Chapter 11 we examined Humdrum tools related to melodic pitch intervals. This chapter returns to the discussion of intervals by focussing on two tools pertaining to characterizing harmonic intervals: the **hint** and **ydelta** commands.

Types of Harmonic Intervals

Harmonic intervals identify pitch distances between nominally concurrently-sounding notes. Example 15.1 illustrates five types of harmonic intervals. The most obvious harmonic intervals are those that occur between pitches that have concurrent onsets. We might call these *explicit harmonic intervals.* An example of an explicit harmonic interval is the perfect octave in the first sonority of Example 15.1

Example 15.1 Types of harmonic intervals.



```
**kern    **kern
*M4/4     *M4/4
=1-       =1-
8c        2cc
8d        .
4e        .
8f        8r
8r        8a
8d        8r
8r        8b
=2        =2
2.c 2.e   2.g 2.cc
*-        *-
```

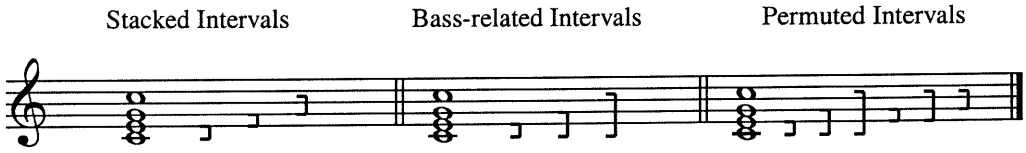
Less obvious harmonic intervals arise when tone onsets are not synchronous — yet both pitches are sustained simultaneously for a period of time. We might call these *passing harmonic intervals.*

Two examples are evident in Example 15.1: the minor seventh between the D and the sustained C, and the minor sixth between E and C. Most music theorists would argue that the minor sixth interval is more important than the minor seventh. Compared with the D, the following E has a longer duration and coincides with a stronger metrical position. Hence the interval of the minor sixth is likely to be more perceptually salient than the minor seventh interval. We might call these intervals *strong passing intervals* and *weak passing intervals* respectively.

Another type of harmonic interval may be deemed to occur even when there are no concurrently sounding pitches. In Example 15.1, the alternating F→A and D→B imply subdominant and dominant harmonic functions and so suggest the intervals of a major third followed by a major sixth. We might call these intervals *implicit harmonic intervals.* The harmonic status of the pitch sequence A→D is more contentious; many theorists would argue that no harmonic interval is implied either because of the weak→strong metric relationship or because the harmonic interval is less stereotypic of the given the cadential cliché. Once again, implicit harmonic intervals might be distinguished as weak or strong.

A further complication in identifying harmonic intervals arises when there are more than two pitches involved. In the final four-note chord of Example 15.1, what harmonic intervals are implied? Example 15.2 shows three possible break-downs of the harmonic relationships. *Stacked harmonic intervals* include only the intervals between successive pitches ordered from low to high. Four pitches thus generate three intervals. *Bass-related harmonic intervals* include all intervals calculated with respect to the lowest pitch. Four pitches again leads to three (different) intervals. This latter way of characterizing intervals is the most similar to figured bass notation. *Permuted harmonic intervals* include all intervals that can be calculated between all notes of a sonority. Four pitches leads to six intervals.

Example 15.2 Interpreting Interval Content in Chords



A final issue in characterizing harmonic intervals is whether or not the presence of unisons is important. In most applications (such as figured bass) the presence of unisons is unimportant. In other applications (such as tabulating instrument doubling in orchestration) the occurrence of unisons is noteworthy.

By way of summary, we have distinguished five types of harmonic intervals, and three measurement methods: explicit harmonic intervals, implicit harmonic intervals (both strong and weak), plus passing harmonic intervals (again both strong and weak varieties). Measurement methods include (1) stacked harmonic intervals, (2) bass-related harmonic intervals, and (3) permuted harmonic intervals.

As in the case of melodic intervals, harmonic intervals can be calculated according to a variety of units — including diatonic intervals, semitones, cents, frequency, and even cochlear coordinates. We will consider just two tools for calculating harmonic intervals: **hint** and **ydelta.**

Harmonic Intervals Using the *hint* Command

The Humdrum **hint** command calculates harmonic intervals for pitch-related representations such as `**kern`, `**pitch`, `**solfg`, and `**Tonh`. As in the case of the **mint** command, output intervals are expressed as a combination of diatonic interval size plus interval quality (such as 'perfect fourth' and 'minor ninth').

In the default operation, **hint** calculates only explicit harmonic intervals; for sonorities containing more than two pitches, only stacked harmonic intervals are calculated. The output from the **hint** command always consists of a single `**hint` spine. Any number of spines may be present in the input, but only pitch-related spines are processed. Given the default invocation, the output corresponding to Example 15.1 is as follows:

```
**hint
*M4/4
=1-
P8
-
-
-
-
-
-
=2
M3 m3 P4
*-
```

Notice that sonorities that contain only a single pitch result in the outputting of a hyphen (-). The hyphen indicates that pitched material is present, but there are no explicit harmonic intervals. Input records that contain no pitch tokens result in the outputting of a null token (.). If a single duplicated pitch is present, then the output will indicate a perfect unison (P1). Unisons can be suppressed from the output via the **-u** option for **hint**.

When more than two pitches are present in a sonority, *permuted harmonic intervals* can be calculated by invoking the **-a** option (i.e. *all* intervals). For example, with the **-a** option, the final chord in Example 15.1 would produce the following output:

```
**hint
*all
M3 P5 P8 m3 m6 P4
*-
```

Notice the presence of the `*all` tandem interpretation in the above output. This interpretation is added to the output in order to warn users that the representation should not be interpreted as stacked intervals.

Bass-related harmonic intervals can be calculated with the **-l** option. In this case, harmonic intervals are calculated with respect to the lowest pitch in the sonority. This option is helpful in determining figured bass. For the final chord in Example 12.1 the corresponding output would be

```
**hint
M3 P5 P8
*-
```

Two further options for **hint** allow the user to tailor how the intervals are represented. The **-c** option causes compound intervals such as a minor tenth (m10) to be output as non-compound equivalents (m3). This means that the interval of an octave (P8) will be rendered as a unison (P1). The **-d** option suppresses the outputting of interval qualities and results in only diatonic interval sizes being output. Again, this option is helpful in determining figured bass. The command:

```
hint -clud
```

will produce the following output for the final major chord in Example 12.1:

```
**hint
3 5
*-
```

Propagating Data Using the *ditto* Command

In the default operation, **hint** calculates intervals only between pitches that are explicitly present in an input data record. This means that passing intervals are not calculated.

In order to generate passing intervals, we will make use of the Humdrum **ditto** command. The **ditto** command replaces null tokens with the previous non-null data token in the same spine. Suppose we had an arbitrary input such as the following:

```
**flip   **flop
A        xyz
.        jkl
.        .
B        abc
.        .
C        .
.        .
*-       *-
```

The effect of **ditto** would be the following:

```
**flip   **flop
A        xyz
A        jkl
A        jkl
B        abc
B        abc
C        abc
C        abc
*-       *-
```

Each null token has been replaced by the preceding data token within the spine.

Consider the effect of **ditto** on the `**kern` data in Example 15.1:

```
ditto -p example15.1
```

The following output results:

```
**kern    **kern
*M4/4     *M4/4
=1-       =1-
8c        2cc
8d        (2cc)
4e        (2cc)
8f        8r
8r        8a
8d        8r
8r        8b
=2        =2
2.c 2.e   2.g 2.cc
*-        *-
```

Notice that the half-note C5 has been repeated. The **-p** option has caused each repetition of the data token to be placed in parentheses so they can be easily recognized. By using **ditto,** we have transformed previously passing intervals into explicit sonorities whose intervals can now be identified by **hint.** We can combine the two commands in a single pipeline:

```
ditto example15.1 | hint
```

The resulting output for Example 15.1 includes the two passing intervals (m7 and m6) in the first measure:

```
**hint
*M4/4
=1-
P8
m7
m6
-
-
-
-
=2
M3 m3 P4
*-
```

The **ditto**, command provides to additional options that are worthy of note: the **-s** and **-c** options. The **-s** option allows **ditto** to skip or ignore the presence of certain data records. Suppose, for example, that we had a barline in the midst of some null tokens:

```
A
.
=
.
```

Often, we would like to propagate certain data tokens *around* some other types of data tokens, so the result might be:

```
A
A
=
A
```

By providing **ditto** with a suitable regular expression, we can have the data token 'A' skip over the barline:

```
ditto -s ^=
```

Without this option, the final data token in the above example would be an equals-sign rather than the token 'A'.

The **-c** option for **ditto** allows the user to selectively identify which characters are propagated. For example, the following command will cause only the lower-case letters 'a' and 'b' to be propagated:

```
ditto -c ab
```

This feature allows users to replicate only certain kinds of data — such as pitches, durations, dynamic marks, etc.

As we will see in future chapters, the **ditto** command proves useful in a wide variety of situations apart from calculating intervals.

Using the *ditto* and *hint* Commands

Let's pause and consider some of the ways we might use the **ditto** and **hint** commands. First, let's determine if some input contains a particular interval. Are there any augmented sixth intervals in Bach's two-part inventions? The following commands look for explicit and passing sixths respectively. Notice the use of the **-c** option so octave equivalents will also be identified:

```
hint -c inventio* | grep A6
ditto -s ^= inventio* | hint -c | grep A6
```

Are there any diminished octave intervals between any two concurrent notes in any of Beethoven's piano sonatas?

```
ditto -s ^= sonatas* | hint -a | grep d8
```

In orchestral works, some pairs of instruments are more likely than others to double each other at

the unison or octave. What proportion of the intervals formed by the oboe and flute notes are doubled? Since we are looking for a proportion, we need to make two counts: the total number of (explicit) intervals formed by the oboe and flute, and the number of those intervals that are octave equivalents. (We will assume that there is only one oboe and one flute part in the file `Rimsky-K`:)

```
extract -i '*Ioboe,*Iflts' Rimsky-K | hint -c | rid -GLId \
   | grep -c P1
extract -i '*Ioboe,*Iflts' Rimsky-K | hint -c | rid -GLId \
   | grep -c [MmPAd]
```

The second **grep** counts the total number of intervals by looking for all of the interval qualities (major, minor, perfect, etc.)

Suppose we have extracted two horn parts from an orchestral score. Are octave intervals between the horns more likely to occur on the dominant pitch or the tonic pitch?

```
solfa horns > temp1
hint horns > temp2
assemble temp1 temp2 | grep -c ^do.*P8
assemble temp1 temp2 | grep -c ^so.*P8
```

Determining Implicit Harmonic Intervals

Recall that *implicit harmonic intervals* may be deemed to occur between tones that don't actually sound at the same time. This arises when one part has a rest while the other part is sounding. Note that if we could eliminate rest tokens, then we could use **ditto** to repeat previous pitch tokens in place of the rests and so generate implicit harmonic intervals.

The **humsed** command (described in Chapter 14) is well suited to this task. We want to transform any data token containing the letter 'r' to a null token. Consider the following substitution:

```
humsed 's/.*r.*/.     ./' example15.1
```

Unfortunately, this isn't quite right. The above substitution will find any data record containing the letter 'r' and transform the entire record to a single null data record. We need to address individual data tokens. In this example, rest tokens may be in two possible positions: the first token in the record or the last token in the record. We need two different regular expressions to address each of these conditions. First, a regular expression to identify rests in the first token:

```
/^[^    ]*r[^    ]*    /
```

(That is, the beginning of the record (^), followed by zero or more instances of any character other than a tab ([^]*), followed by the letter 'r', followed by zero or more instances of any character other than a tab ([^]*), followed by a tab.)

Similarly, our second regular expression identifies rests in the last token:

```
/    [^    ]*r[^    ]*$/
```

Now we can eliminate rest tokens using the following two substitution commands within a single invocation of **humsed**:

```
humsed 's/^[^  ]*r[^  ]*  /.  /; s/  [^  ]*r[^  ]*$/  ./' example15.1
```

The following output results:

```
**kern    **kern
*M4/4     *M4/4
=1-       =1-
8c        2cc
8d        .
4e        .
8f        .
.         8a
8d        .
.         8b
=2        =2
2.c 2.e   2.g 2.cc
*-        *-
```

If we now apply **ditto** and recalculate the intervals, the resulting output will identify some implicit intervals as well:

```
humsed 's/^[^  ]*r[^  ]*  /.  /; s/  [^  ]*r[^  ]*$/  ./' example15.1 \
   | ditto -p
```

Below we see the output assembled with the output from the corresponding **hint** command:

```
**kern    **kern   **hint
*M4/4     *M4/4    *M4/4
=1-       =1-      =1-
8c        2cc      P8
8d        (2cc)    m7
4e        (2cc)    m6
8f        (2cc)    P5
(8f)      8a       M3
8d        (8a)     P5
(8d)      8b       M6
=2        =2       =2
2.c 2.e   2.g 2.cc M3 m3 P4
*-        *-       *-
```

The *ydelta* Command

Often it is useful to represent intervals by the number of semitones (or some other numerical value). We might begin by using the **semits** command to translate Example 15.1 to a `**semits` representation.

```
semits example15.1
```

The resulting output would be as follows:

```
**semits    **semits
*M4/4       *M4/4
=1-         =1-
0           12
2           .
4           .
5           r
r           9
2           r
r           11
=2          =2
0 4         7 12
*-          *-
```

Numerical differences for values on a single data record can be computed using the **ydelta** command. The **ydelta** command is comparable to **xdelta** however, numerical differences are calculated between simultaneous numerical values (delta-*y*) rather than between successive numerical values (delta-*x*).

Like the **hint** command, **ydelta** always outputs a single spine. The user must specify which input spines are to be processed using the **-i** option. In the following command, only **semits input is to be processed:

```
semits example15.1 | ydelta -s = -i '**semits'
```

The **-s** option allows the user to identify data records to be *skipped* while processing. In this case, the regular expression '=' is used to identify barlines, so measure numbers will be excluded when processing the numerical data.

The above command yields the following output:

```
**Ysemits
*
=1-
12
.
.
.
.
.
.
=2
4 7 12
*-
```

Notice that **ydelta** prepends the upper-case letter 'Y' to the given input interpretation. All output values are calculated with respect to the lowest value in the current data record. Hence, the '4 7

12 in the last data record means that there are pitches 4 semitones above the lowest note, 7 semitones above the lowest note, and 12 semitones above the lowest note. (If necessary, the lowest or offset value for each record can be output in square brackets using the **-o** option.)

Like the **hint** command, **ydelta** calculates numerical intervals only when more than one value is present on a given input data record. As in the case of **hint,** we might use the **ditto** command to propagate pitch values — replacing all the null data tokens. A suitable command would be:

```
semits example15.1 | ditto -s = | ydelta -s = -i '**semits'
```

The resulting output would be:

```
**Ysemits
*
=1-
12
10
8
.
.
.
.
=2
4 7 12
*-
```

More Examples Using the *ydelta* Command

What is the average semitone distance between the cantus and altus voices in Lassus motets? We can answer this question by first extracting the appropriate voices and translating to a semitone representation.

```
extract -f 1,2 motet* | semits > temp1
```

Using **ditto** we can expand the pitched material so that concurrently-sounding tones will generate explicit intervals. We then use **ydelta** to calculate the actual semitone interval distances. The **rid** command can be used to eliminate non-data records, and the **grep -v** command can be used to further eliminate barlines. Finally, we can calculate the mean interval distance using the **stats** command:

```
ditto -s = temp1 | ydelta -s = -i '**semits' | rid -GLId \
   | grep -v = | stats
```

Suppose we have a two-part input. Are there tritone intervals (explicit and passing) that are not spelled as either an augmented fourth or diminished fifth? We can answer this question by using both **hint** and **ydelta** and a suitable sequence of **grep** commands. The **ditto** command is used to ensure that both explicit and passing intervals are generated.

```
ditto -s = 2part | semits | ydelta -s = -i '**semits' > temp1
ditto -s = 2part | hint > temp2
assemble temp1 temp2 | rid -GLId | grep ^6 | grep -v A4 \
    | grep -v d5
```

Notice the use of **grep -v** to first exclude any records that match an augmented fourth, and then to exclude any remaining records that match a diminished fifth.

Reprise

Harmonic intervals can be measured in a variety of ways. They can be characterized as diatonic qualities such as minor sevenths or augmented sixths. They can be measured in terms of semitone distance — or even in cents or hertz (frequency difference). Only the diatonic size may be of interest (e.g., "a fifth"), and compound intervals (e.g., major tenth) can be expressed by their non-compound equivalents (major third).

At least five types of harmonic intervals can be distinguished including *explicit harmonic intervals*, *strong passing intervals*, *weak passing intervals*, *strong implicit harmonic intervals* and *weak implicit harmonic intervals*. In addition, we distinguished three different ways of characterizing harmonic intervals: *stacked harmonic intervals*, *bass-related harmonic intervals* and *permuted harmonic intervals*.

In this chapter we have seen how to use the **hint** command to calculate these various kinds of intervals. We have also seen how **ydelta** can be used to measure purely numerical distances between concurrent values.