Capítulo 1

MODIFICACIÓN DE PARTITURAS SERIALISTAS

1.1. Escalas y funciones del experimento

1.1.1. Escalas interválicas, escalas y funciones

Una *escala interválica* es una secuencia ordenada de números naturales – una secuencia de intervalos entre notas – tales que la suma de todos ellos da 12. Así solo se consideran válidas las escalas equivalentes octava a octava: todos los intervalos de la escala deben sumar el número de semitonos de una octava. Por ejemplo, la escala diatónica jónica (o escala mayor) tiene como secuencia (2, 2, 1, 2, 2, 2, 1).

Dada una escala interválica de longitud L y una nota fija inicial, la secuencia de intervalos se plasma en una secuencia de notas de longitud L+1. Se construye comenzando por la nota inicial y sumando cada intervalo para conseguir la nota siguiente.

Con la escala mayor y la nota Re se consigue {Re, Mi, Fa#, Sol, La, Si, Do#, Re}, ya que es equivalente a {2, 2+**2**=4, 4+**2**=6, 6+**1**=7,

7+**2**=9, 9+**2**=11, 11+**2**=13, 13+**1**=14}. Por construcción, la última nota debe ser equivalente a la primera, ya que en el último paso habremos sumado a la nota inicial todos los términos de la secuencia interválica, y por definición suman 12.

De esta forma, se puede definir una *escala-k* como el conjunto de notas generadas por una escala interválica desde la nota k. Por ejemplo, el conjunto anterior sería la escala-2 mayor; es decir, la escala de Re mayor. Una escala generada por una secuencia de intervalos con longitud L tiene L notas, ya que como la última es repetida puede no tenerse en cuenta. La longitud $L \leq 12$, ya que una escala-k definida de esta forma siempre es un subconjunto de la escala cromática: $E_k \subseteq \mathbb{Z}/(12)$

Una *función a una escala-k* es una función f que transforma cada nota de la escala cromática a un valor de la escala E_k. Entonces $f : \mathbb{Z}/(12) \to E_k$ reduce las notas de una melodía a solamente la escala escogida. Las funciones a escalas se representan de la siguiente manera, con la primera fila representando el dominio de f (la escala cromática), la segunda su imagen (la escala E_k), y la tercera su secuencia interválica:

0	1	2	...	10	11
$f(0)$	$f(1)$	$f(2)$	...	$f(10)$	$f(11)$
$f(1) - f(0)$	$f(2) - f(1)$	$f(3) - f(2)$	...	$f(11) - f(10)$	$12 + f(0) - f(11)$

En realidad, la k de la escala-k no es especialmente relevante, porque una escala-$k+1$ es la transportada de una k. Se puede escoger sin pérdida de generalidad $k = 0$ a partir de ahora, y así todas comenzarán en Do.

El proceso verdaderamente interesante está en averiguar, dada una escala E, cuál es la mejor función que transforma melodías cromáticas en melodías en E. Estas son las *funciones* E-*inducidas*.

¿Cuáles serán las características de esas funciones óptimas? Deben conservar la estructura serial y deben conservar el parecido con la melodía original.

1.1.2. Funciones bien distribuidas

La mayor prioridad es conservar la estructura serial de las piezas; por tanto, todas las notas deben aparecer con la menor frecuencia posible, y se debe evitar jerarquías entre las notas en la medida de lo posible. Si $|\mathrm{E}| < 12$, f no puede ser inyectiva, por lo que va a haber elementos repetidos en la imagen. Queremos la f que mejor distribuya esas repeticiones, que distribuya las notas de E a lo largo de la escala cromática.

Lo óptimo sería que todas tuvieran la misma frecuencia. Eso solo pasará cuando $|\mathrm{E}|$ divida a 12. Por ejemplo, si $\mathrm{E} = \{a_1, a_2, a_3, a_4, a_5, a_6\}$ (entonces $|\mathrm{E}| = 6$), existen funciones tales que cada nota de la imagen se repite exactamente 2 veces. La siguiente función E-inducida f cumpliría la condición de buena distribución:

Cromática	0	1	2	3	4	5	6	7	8	9	10	11
Escala E	a_1	a_1	a_2	a_2	a_3	a_3	a_4	a_4	a_5	a_5	a_6	a_6
Intervalos						...						

En cambio, si $|\mathrm{E}|$ no divide a 12 no hay funciones E-inducidas totalmente distribuidas. No existe una sola frecuencia que puedan compartir todas las notas de E. Sin embargo, sí se pueden encontrar dos frecuencias consecutivas, c y $c+1$, tales que todos los elementos de E tengan o frecuencia c o frecuencia $c+1$. Esto es lo más parecido a que todas tengan la misma frecuencia, y se va a probar a continuación que siempre es posible.

La situación es equivalente a que E se pueda dividir en dos subconjuntos disjuntos Q y R, con $|\mathrm{Q}| = q$ y $|\mathrm{R}| = r$ (entonces $q + r = |\mathrm{E}|$), tales que la frecuencia de las notas en Q es c y la frecuencia de las notas en R es $c + 1$. En resumen, para probar que Q y R existen, debemos encontrar un c, un q y un r naturales para los que $cq + (c + 1)r = 12$.

$cq + (c+1)r = cq + cr + r = c(q+r) + r = c|\mathrm{E}| + r = 12$, lo cual se cumple por el algoritmo de la división, que asegura que al dividir 12 entre $|\mathrm{E}|$ existen su cociente c y su resto $r \geq 0$. □

El siguiente gráfico describe, para cada posible $|\mathrm{E}|$ en cada fila, la frecuencia óptima de sus elementos. Las columnas representan las frecuencias de los elementos, y los números de dentro son cada q y r (cuando es 0 no se escribe: no hay notas con esa frecuencia).

	1	2	3	4	5	6	7	8	9	10	11	12
1												1
2						2						
3				3								
4			4									
5		3	2									
6		6										
7	2	5										
8	4	4										
9	6	3										
10	8	2										
11	10	1										
12	12											

1.1.3. Funciones E-inducidas

Hay que pedir más requisitos a f para que no solo modifique las notas, sino que además las imágenes se parezcan lo máximo posible a sus preimágenes, a las notas originales.

(1) f debe ser sobreyectiva: si no, la música resultante tendría una escala más reducida de la deseada.

(2) f debe ser creciente: si no, las dos notas decrecientes se deberían intercambiar. También se podrá sumar o restar 12 a las notas que lo requieran. La monotonía debe conservarse, además, en

todas las octavas, por lo que las funciones deben acabar o por la nota con la que se empieza +12 o por una menor a ella.

(3) f debe tener el mayor número de puntos fijos posible: las notas que puedan mantenerse estables al aplicar f deben quedarse igual. Existen escalas para las que no se puede tener buena distribución, monotonía creciente y todas sus notas fijas, así que este criterio es menos prioritario.

Por ejemplo, para la escala {0, 1, 2}, si se fijan las tres notas, f empezaría con 0 1 2. Por (2), la nota asociada al 12 debe ser un 12. Entonces los otros tres 1's que faltan no se pueden asignar, ya que después del 2 no puede ir un 1 y antes del 12 no puede ir un 13. Por tanto, f no estaría bien distribuida. La solución es que solo se fijen dos de las tres notas.

Cromática	0	1	2	3	4	5	6	7	8	9	10	11	(12)
Escala E	0	1	2			~~1~~			~~13~~				(12)

(4) Por último, de entre las f que queden, se escogerán aquellas cuya distancia a la cromática sea menor. Se calculará para todo i la expresión $|f(i) - i|$, y se sumarán todos los resultados. Esa suma será la *puntuación* de dicha f. Las funciones con menor puntuación son las funciones de mejor ajuste.

Si aún quedan varias f que cumplen todos los requisitos, se escogerá la más grave, la menor de ellas. De esta manera, dada cualquier escala E, la función E-inducida queda unívocamente determinada.

En el Anexo A, página 9, se encuentra el código en Haskell de un programa que, dado una escala, produce su función inducida óptima con las propiedades descritas anteriormente. También puede encontrarse en *https://gitlab.com/dodecafonismo/f-inducida*.

Anexos

Apéndice A

Código para generar funciones E-inducidas

```
import Control.Monad (forever, guard)
import Data.List
import Data.Ord (comparing)


type Scale = [Int]


-- Es la longitud de la escala cromática: el cardinal del
long :: Int
long = 12


main :: IO String
main
  = forever
  $ do
      scale <- getLine
      putStrLn
        "Calculando...\n"
      showResult
        $ nub
        $ induce
        $ readScale
          scale
      putStrLn
        "\nTerminado."


readScale :: String -> Scale
readScale scale
  = (\x -> if length x > 12 then [] else x)
  $ map
    (flip mod long . fst . head)
  $ filter
```

```
    (not . null)
  $ map reads
  $ words
    scale


-- Mostrar una escala.
showScale :: Scale -> String
showScale scale
  = drop 1  -- Quita el primer espacio.
  $ foldl
    (\acc i ->
      acc
      ++ " "
      ++ show i
      -- Deja doble hueco en los números de una cifra.
      ++ if i >= 0 && i < 10
           then " "
           else ""
    ) "" scale


-- Mostrar varias escalas.
showScales :: [Scale] -> String
showScales scales
  = drop 1
  $ foldl
    (\acc sc ->
      acc
      ++ "\n"
      ++ showScale sc
    ) "" scales


-- Mostrar el resultado del cálculo.
```

```
showResult :: [Scale] -> IO ()
-- 'best' requiere no vacío.
-- Si ponemos la restricción en 'best', entonces falla hed
showResult []
  = return ()

showResult scales
  = do
      putStrLn
        $ showScale [ 0 .. (long - 1) ]
      putStrLn
        $ replicate (long * 3 - 1) ''
      putStrLn
        $ showScales scales
      putStrLn
        $ "\nMayor ajuste ("
        ++ show bestPunct
        ++ "):"
      putStrLn
        $ showScales bestFits
      putStrLn
        "\nLa más grave es:"
      putStrLn
        $ showScale
        $ head
        $ sort
          bestFits
    where
      (bestFits, bestPunct) =
        best scales


best :: [Scale] -> ([Scale],Int)
best scales
  = ( map fst solution
```

```
    , head $ map snd solution
    )
  where
    solution
      = head  -- El primer grupo tiene la mínima distancia.
      $ groupBy
        (\x y -> snd x == snd y)
      $ sortBy
        (comparing snd)  -- La menor distancia.
      $ map
        (\s -> (s, norm s))
        scales


norm :: Scale -> Int
norm s
  = sum  -- La suma de las distancias
  $ map  -- entre la función y la escala cromática.
    (\(x, i) -> abs (x-i))
  $ zip s [0..]
  -- = sqrt
  -- $ fromIntegral
  -- $ sum
  -- $ map
  --   (\(x, i) -> (x-i)*(x-i))
  -- $ zip s [0..]


-- Llama a la función recursiva de fixed points.
induce :: Scale -> [Scale]
induce scale =
  fxp (length scale) scale []


-- Obliga a que haya alguna fixed.
```

```
-- >Existe alguna solución que no tenga 1 fixed pero sí 0?
-- He puesto demasiados head fixed, supongamos que no.
fxp :: Int -> Scale -> [Scale] -> [Scale]
-- Si encontró solución, devuélvelo.
fxp _ _ xxs@(_:_)
  = xxs
-- Si 0 fixed points, no hay solución.
fxp 0 _ _
  = []
-- Si quedan fixed points, llama a func y disminuye nfixed
fxp nfixed scale _
  = fxp (nfixed - 1) scale (func nfixed scale)

-- TODO importarlo de sitio existente
-- #26 de H-99
combinations :: Int -> [a] -> [[a]]
combinations 0 _
  = [[]]
combinations _ []
  = []
combinations n (x:xs)
  = map
    (x:)
    (combinations (n-1) xs)
    ++ combinations n xs


dissonances :: Scale -> [Scale]
dissonances scale
  = combinations q scale
  where
    e = length scale
    q = e - mod long e
```

```
-- q es el tamaño del subconjunto de frecuencias menores.
-- c es la frecuencia menor.
-- long = qc + (e - q)(c + 1)
func :: Int -> Scale -> [Scale]
func nfixed scale
  = let e = length scale
        c = div long e
    in do
      fixed <-
        combinations nfixed scale
      minFreq <-
        dissonances scale
      start <- --[[h], [h,h], [h,h,h]]
        let
        h = head fixed
        hFreq =
          if h `elem` minFreq
            then c
            else c + 1
        in
        take hFreq $ iterate (++ [h]) [h]
      -- Quito los que tienen el 2o fixed dentro del ámbito de
      guard
        (length fixed == 1
        || (fixed !! 1) - (head fixed) >= length start)
      result <-
        let
        h = head fixed
        ( (first, freqFirst) : oldFreqs )
          = map
            (\ha ->
              if ha `elem` minFreq
                then (ha, c)
                else (ha, c + 1)
             )
```

```
            $ take e
            $ dropWhile (< h)
            $ cycle
              scale
        freqs  -- Ponemos el h al final con su frecuencia
          = oldFreqs
          ++ if freqFirst /= length start
               then [ (first, freqFirst - length start) ]
               else []
        newfixed
          = take (nfixed - 1)
          $ dropWhile (<= h)
          $ cycle
            fixed
        in return $ oneCase newfixed freqs start
      guard
        (length result == long) -- Quito los que no tengan
      return $ normalize result


-- Pongo el índice 0 otra vez al principio
-- y lo pongo en orden creciente.
normalize :: Scale -> Scale
normalize []
  = []
normalize result@(h:_)
    = map (\i -> if i > h then i - long else i) before
    ++ hh
    ++ map (\i -> if i <= h then i + long else i) after
  where
    (hhafter, before)
      = splitAt (long - h) result
    (hh, after)
      = span (== h) hhafter
```

```
-- Dados los puntos fijos, las frecuencias y el resultado
oneCase :: Scale -> [(Int, Int)] -> Scale -> Scale
oneCase _ _ []
  = []  -- Si no hay start ha habido un problema. No puede ocu
oneCase _ [] x
  = x   -- Si no quedan frecuencias, hemos acabado bien.
oneCase [] ((next, nextFreq):freqs) start
  =     -- Si no quedan fixed points, simplemente rellenar.
  oneCase
    []
    freqs $
    start ++ replicate nextFreq next

-- Si quedan fixed points:
oneCase ffixed@(f:fixed) ((next, nextFreq):freqs) start@(s:_)
    -- Si f es igual al siguiente índice, next tiene que ser e
    | f==currentIndex && f==next
    = oneCase
        fixed  -- Ya hemos usado f; lo desechamos.
        ((eraseFreq next nextFreq) ++ freqs) $
        start ++ [next]
    --Si no necesito fijar, entonces next no puede pasarse del
    | f/=currentIndex && f>=next
    = oneCase
        ffixed
        ((eraseFreq next nextFreq) ++ freqs) $
        start ++ [next]
    | otherwise = []
  where
    currentIndex
      = s + length start
    eraseFreq element freq
      = if freq == 1
          then []
```

```
    else [(element, freq-1)]
```

Bibliografía

[1] Wright, David. *Mathematics and Music*, American Mathematical Society (2009).

[2] Prof. Fernando Delgado García. Clases y material de Historia de la Música, 5° y 6° de Enseñanzas Profesionales del Conservatorio Profesional de Música Arturo Soria, cursos 2014-15 y 2015-16.

[3] Boulez, Pierre. *Schoenberg is dead*, The Score (1952).

[4] Domínguez Romero, Manuel. *Las Matemáticas en el Serialismo Musical*, Sigma n.24 (2004).

[5] Kinney, James P. *Twelve-tone Serialism: Exploring the Works of Anton Webern*, Undergraduate Honors Theses. Paper 1 (2015)

[6] Díaz de la Fuente, Alicia. *Estructura y significado en la música serial y aleatoria*, Universidad Nacional de Educación a Distancia. Tesis Doctoral en Filosofía (2005)

[7] Xiao, June. *Bach's Influences in the Piano Music of Four 20th Century Composers*, Indiana University Jacobs School of Music. Doctoral Theses in Music (2014)

[8] Clercq, Trevor de. *A Window into Tonality via the Structure of Schoenberg's "Musette" from the Piano Suite, op. 25*,

Theory/Analysis of 20th-Century Music. Prof. David Headlam (2006)

[9] Hyde, Martha. Chapter 4: "Dodecaphonism: Schoenberg", *Models of Musical Analysis: Early Twentieth-century Music*, Ed. Mark Everist and Jonathan Dunsby. Oxford: Blackwell (1993)

[10] Ilomäki, Tuukka. *On the Similarity of Twelve-Tone Rows*, Sibelius Academy (2008).

[11] Armstrong, M. A. Chapter 6: "Permutations", Chapter 17: "Actions, Orbits, and Stabilizers", Chapter 18: "Counting Orbits", *Groups and Symmetry*, New York: Springer-Verlag (1988)

[12] Golomb, S. W., Welch, L. R. *On the enumeration of polygons*, The American Mathematical Monthly, Vol. 67, 349-353 (1960).

[13] Reiner, David L. *Enumeration in Music Theory*, The American Mathematical Monthly, Vol. 92, No. 1 (1985).

[14] Basomba García, Daniel. *El último Bach y el dodecafonismo como ideal musical: una lectura estética y sociológica*, Universidad Carlos III de Madrid. Tesis Doctoral en Ciencia Política y Sociología (2013)