TRABAJO FIN DE GRADO

OBTENCIÓN DE FRACCIONES DE DISEÑOS FACTORIALES A DOS NIVELES CON 32 OBSERVACIONES

TRABAJO FIN DE GRADO PARA LA OBTENCIÓN DEL TÍTULO DE **GRADUADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES**

FEBRERO **2017**

Javier García García

DIRECTOR DEL TRABAJO FIN DE GRADO:

Carlos González Guillén

Jesús Juan Ruiz

INDUSTRIALES
ETSII | UPM

POLITÉCNICA

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mis tutores Carlos y Jesús la dedicación e implicación en mi Trabajo de Fin de Grado. Gracias a su ayuda este trabajo ha resultado mucho más ameno e interesante.

También me gustaría agradecer a Carmen y a todos mis amigos (Héctor, Fajardo, Jimbo, Eloy, Armando, Nacho, Ina, Guede, Guille, Pippo, Noel, Nova, Castell, Revu, José, Lydia, Mer, Laura, Cris…) el apoyo durante todos estos años de carrera. Su compañía ha sido fundamental para disfrutar durante todos estos años.

Por supuesto quiero agradecer a mis padres que me brindaran la oportunidad de estudiar una carrera.

Por último, me gustaría dar las gracias al resto de mi familia y amigos que no he nombrado pero me han mostrado apoyo a lo largo de estos años.

RESUMEN

Una de las aplicaciones fundamentales de la Estadística es el diseño de experimentos. En un diseño experimental se manipulan deliberadamente una o más variables (factores), para medir el efecto que tienen en otra variable de interés (variable respuesta). En un diseño factorial, es necesario medir la variable respuesta para dos o más valores de cada factor.

En los estudios exploratorios una estrategia habitual es considerar solo dos niveles o valores en cada uno de los factores. A estos diseños experimentales se le denomina diseños factoriales a dos niveles. El número de observaciones que requiere cada diseño es una potencia de dos.

En muchas aplicaciones en la industria, el número de factores es elevado y la aplicación de diseños factoriales completos exige un número muy elevado de experimentos a realizar. Cuando los recursos son limitados, el número de factores incluidos grande, o simplemente se quiere reducir el coste experimental, se utilizan las fracciones de diseños factoriales.

Un diseño factorial fraccionado utiliza un subconjunto de un diseño factorial completo, así que algunas variables aparecerán confundidas con interacciones de otras variables. En un diseño factorial fraccionado $2^{k-p} = n$, se denomina *k* al número de variables del experimento, *p* al grado de fraccionamiento y *n* al número de observaciones de un experimento.

A modo de ejemplo, se va a definir un diseño factorial fraccionado con 7 variables (*k=7*), 2 palabras generadoras (*p=2*) y 32 replicaciones (*n=32*) al que conoceremos como Ejemplo.

$$Ejemplo = \{A, B, C, D, E, AB, ABC\}$$

Al tratarse de un diseño de 32 replicaciones, el diseño factorial completo estaría compuesto por 5 variables ($2^5 = 32$). Sin embargo, de acuerdo a Ejemplo, aparecen dos variables extras (*F* y *G*) confundidas con dos interacciones (*AB* y *ABC*), a las que se conoce como *palabras generadoras p*. Estas palabras quedan representadas por las expresiones $F = AB$ y $G = ABC$, o $I = ABF$ e $I = ABCG$, siendo *I* la identidad.

Cada diseño viene caracterizado por su *ecuación generatriz*, que queda definida por los productos de las *palabras generadoras p*. En este caso la *ecuación generatriz* sería:

$$I = ABF$$

$$I = ABCG$$

$$I = CGF$$

Una forma de caracterizar la ecuación generatriz de un diseño es mediante su *Word Length Pattern*. Se define el *Word Length Pattern* de un diseño como un vector cuya componente i-ésima es el número de palabras de longitud i que hay en su ecuación generatriz completa. Así en el ejemplo anterior en el que la *ecuación generatriz completa* era:

$$I = ABF = ABCG = CGF$$

el *Word Length Pattern* sería:

$$WLP = (0, 0, 2, 1, 0, 0, 0)$$

Se define la *Resolución* del diseño como la longitud de la palabra más corta de la ecuación generatriz completa. Así en el caso anterior la resolución del diseño será 3. Como se puede apreciar, la resolución de un diseño y el *Word Length Pattern* depende de cómo hayan sido elegidos los términos independientes del mismo.

Así, si por ejemplo tenemos confundido un efecto principal con una interacción de orden cuatro sabremos con mucha certeza que la estimación se debe al efecto principal y no a la interacción de orden cuatro, pero si tenemos un efecto principal confundido con una interacción de orden dos, la estimación del factor principal puede estar muy sesgada y ya no podemos estar seguros de que dicha estimación se deba al efecto principal puesto que también puede ser debido a la interacción o a ambos.

La resolución de un diseño indicará con qué orden de interacción van a estar confundidos los factores principales interacciones de orden dos, tres, etc, o más bien con qué orden de interacción no van a estar confundidos.

A la hora de construir una determinada fracción interesa principalmente que los efectos principales e interacciones de orden bajo estén confundidos con interacciones de orden superior. Para conseguir esto se siguen los criterios de *máxima resolución* y *mínima aberración*. Estos dos criterios están basados en unas hipótesis aceptables empíricamente:

1) Las interacciones de orden bajo son más importantes que las de orden alto.

2) Las interacciones del mismo nivel son igual de importantes.

3) Sólo hay un número relativamente pequeño de efectos significativos.

Veamos a continuación estos dos criterios:

- ***Máxima resolución*:** Como se ha indicado antes, interesa que los diseños tengan máxima resolución para que los efectos principales o interacciones de orden bajo aparezcan confundidas con efectos de orden alto, para así poder estimar su efecto con elevada fiabilidad.

Para que un diseño sea interesante desde un punto de vista práctico debe tener una resolución superior a dos para que los efectos principales no aparezcan confundidos entre sí. De todas formas, es conveniente que la resolución sea como mínimo 3 o 4 para que los efectos principales no estén confundidos entre sí o con interacciones de orden dos, cuyo efecto puede ser importante.

- ***Mínima aberración*:** El concepto de aberración se usa para comparar dos diseños, así siempre diremos que un diseño tiene mayor o menor aberración que otro pero no hablaremos de la aberración de un diseño por sí mismo.

Supongamos dos diseños 2^{k-p}, D_1 y D_2 de resolución máxima, cuyos *Word Length Pattern* vienen dados por:

- $WLP_1 = (A_1^{(1)}, A_2^{(1)}, A_3^{(1)}, \ldots, A_n^{(1)})$
- $WLP_2 = (A_1^{(2)}, A_2^{(2)}, A_3^{(2)}, \ldots, A_n^{(2)})$

entonces diremos que D_1 tiene menor aberración que D_2 si el primer "i" tal que $A_i^{(1)} \neq A_1^{(2)}$ verifica que $A_1^{(1)} < A_1^{(2)}$.

Veamos un ejemplo en el que podemos comprobar la importancia de conseguir diseños con menor aberración. Consideremos los dos diseños 2^{7-2} siguientes:

$$D_1: I = DEFG = ABCDF = ABCEG$$

$$D_2: I = ABCF = ADEG = BCDEFG$$

Ambos tienen resolución 4 pero *Word Length Pattern* distintos:

$$WLP_1 = (0, 0, 0, 1, 2, 0, 0)$$

$$WLP_2 = (0, 0, 0, 2, 0, 1, 0)$$

Según lo dicho anteriormente el diseño D_1 tiene menor aberración que el D_2. Al ser ambos diseños de resolución 4 los efectos principales no estarán confundidos con interacciones de orden dos, aunque estas sí que estarán confundidas entre sí. Pero mientras que en el diseño D_1 solo aparecen tres pares de interacciones de orden dos confundidas entre sí (*DE = FG, EF = DG* y *DF = EG*) en el diseño D_2 aparecen nada menos que seis (*AB = CF, AC = BF, AD = EG, AE= DG, AF=BD* y *AG = DE*).

Esto es debido a que en el diseño D_1 únicamente existe una palabra de longitud 4 mientras que en el D_2 aparecen dos.

Dado el número total de factores a estudiar y el número de experimentos a realizar, el experimentador cuenta con varias posibilidades a la hora de elegir su diseño. No obstante, encontrar el diseño óptimo para cada caso particular es muy complejo. Debido a lo explicado en los párrafos anteriores, el diseño óptimo suele ser el de

máxima resolución y *mínima aberración*, sin embargo puede haber casos particulares en los que esto no sea así.

Para facilitar el trabajo del experimentador, en un trabajo anterior David Santos Vaquero había calculado todos los diseños posibles para diseños factoriales fraccionados con 16 observaciones (un total de 46). Como la utilidad práctica era significativa, pero el número de diseños calculados hasta el momento era pequeño, se decidió emprender la búsqueda de todos los diseños factoriales fraccionales a dos niveles de 32 observaciones con diferente *Word Length Pattern* (2^{k-p} = 32 observaciones).

El problema era de enorme complejidad por el número elevado de combinaciones a comparar y debía ser tratado computacionalmente.

Por ello, en el presente trabajo se han desarrollado tres programas en Matlab. El primer programa, al que se le ha bautizado como *"CaminosNodo"*, ofrece los diseños *k+1* con su correspondiente *Word Length Pattern* a los que es posible llegar desde un diseño concreto con *k* factores. El segundo programa, llamado *"DiseñosN32"*, permite obtener todos los diseños posibles para diseños factoriales $n=2^{k-p}$ con 32 observaciones (además es fácilmente adaptable a 16, 64, etc). Por último se ha creado, un tercer programa, *"Complementarios"*, que calcula los diseños complementarios de un diseño dado, con su correspondiente *Word Lenght Pattern*. El último programa tiene menor interés para el experimentador, ya que ha sido una herramienta de validación y verificación de los otros dos programas de acuerdo a la teoría.

Los resultados han sido satisfactorios a la par que interesantes. Se han obtenido 820 diseños con diferente *Word Length Pattern* en el rango de 32 observaciones. Además, se han encontrado fenómenos matemáticos inesperados, como la aparición de diseños con el mismo *Word Length Pattern* pero diferentes, algo que no se esperaba hasta valores de *k-p* superiores.

Asimismo los resultados de este Trabajo Fin de Grado, tienen una utilidad menos explícita ya que tienen diversas conexiones con la matemática discreta, teorema de Galois y otros problemas de la Teoría de Números, un tema que apasiona a los matemáticos. Cabe destacar la estrecha relación que guarda con problemas de codificación, concretamente con los códigos de detección de errores, a los cuáles, los resultados de este proyecto pueden ser aplicados directamente.

En definitiva, a pesar de la enorme utilidad para el desarrollo de experimentos de los programas desarrollados y los diseños encontrados en este trabajo, los diseños factoriales 2^{k-p} están ligados a numerosos problemas combinatorios en el ámbito matemático, por lo que los resultados obtenidos en este trabajo pueden ser extrapolados a otras áreas matemáticas que no sean simplemente el diseño de experimentos.

ÍNDICE

INTRODUCCIÓN

En el presente proyecto se tratarán los diseños factoriales a dos niveles (fracciones 2^{k-p}). Se parte de una tesis anterior, publicada por Gabriel Palomo y tutelada por Jesús Juan Ruiz, en la que se elaboró un mapa gráfico de todos los diseños posibles para $n=16=2^{k-p}$. El objetivo de este proyecto es continuar la línea de aquella tesis y realizar el estudio de los diseños factoriales $n=32=2^{k-p}$.

La parte central del proyecto es el desarrollo de un programa informático en Matlab (*Matrix Laboratory*) y las posibilidades que nos ofrece para el estudio de estos diseños. La finalidad de este programa es pues ofrecer un catálogo, lo más completo posible de los diseños factoriales $32=2^{k-p}$ coherentes con el criterio de optimalidad de máxima resolución. El programa concretamente analiza partiendo de un diseño, los posibles modelos matemáticos diferentes a los que se podría llegar añadiendo una nueva variable o factor al experimento.

Gracias al programa se ha podido confeccionar un estudio de todos los posibles diseños factoriales $n=32=2^{k-p}$ para cada k y ordenarlos de mejor a peor en cuanto a los criterios matemáticos de máxima resolución y mínima aberración.

Los datos de partida son, en este caso, los términos que conforman el diseño de partida.

El primer capítulo del proyecto es una introducción al marco teórico en el que se desarrolla el proyecto, los diseños factoriales a dos niveles. Si el lector está familiarizado con el tema puede pasar directamente al segundo capítulo, en el que se explica cómo se han construido los programas, detallando las variables utilizadas y exponiendo los pasos a seguir para pasar de cada diseño a los siguientes cuando se quiere añadir un nuevo factor

En el tercer capítulo, se analizan los resultados que han sido posibles de obtener con los programas.

1. MARCO TEÓRICO

1.1. Diseño de experimentos

En la investigación empírica es muy frecuente que repitiendo un experimento en condiciones indistinguibles para el investigador, los resultados obtenidos presenten variabilidad. Esta variabilidad suele ser pequeña en un laboratorio, pero en una planta industrial las diferencias de rendimiento entre lotes de características análogas pueden llegar a ser del 20%.

La metodología del diseño de experimentos estudia cómo realizar comparaciones lo más homogéneas posibles y como elegir los experimentos adecuados, para aumentar, en consecuencia, la probabilidad de detectar cambios o identificar variables influyentes. Comprobar si un tratamiento mejora un proceso requiere comparar los resultados antes y después de aplicarlo. Cuando exista mucha variabilidad entre resultados (un gran error experimental) sólo detectaremos como influyentes aquellos tratamientos que produzcan cambios muy grandes con relación al error experimental.

Solamente podemos confiar en detectar relaciones de causalidad mediante un experimento bien diseñado. Consideremos, por ejemplo, el problema de comparar el rendimiento de varias máquinas, y admitamos que la habilidad del operador influye en la productividad de la máquina. Si el análisis se efectúa sobre los datos históricos, y una de las máquinas ha sido operada por un trabajador con habilidad destacadamente mejor o peor que el resto, podemos atribuir diferencias entre máquinas las diferencias entre trabajadores. Esto mismo puede ocurrir con la influencia de cualquier otro factor; sin embargo, si se ha diseñado un experimento que controla física o estadísticamente las variables relevantes, tendremos una base firme para analizar si las diferencias pueden ser atribuidas a las condiciones experimentales y deducir una relación causal.

El objetivo de un experimento es estudiar el efecto que sobre una variable de interés (variable respuesta), tiene un conjunto de otras variables (variables experimentales o factores).

Así la variable respuesta puede ser, por ejemplo, el rendimiento de un proceso y los factores: la temperatura, la presión y la concentración. El experimento consiste en fijar estas variables a distintos niveles y observar el valor de la variable respuesta en cada unidad experimental. El número total de datos es el tamaño del experimento.

En cualquier experimento en que se investigue el efecto de un factor, existen a priori un gran número de variables que pueden influir sobre los resultados. Conceptualmente existen tres caminos para eliminar el efecto de una variable: el primero es mantenerla fija durante toda la realización del experimento, el segundo es reorganizar la estructura del experimento de manera que las comparaciones de interés se efectúen para valores fijos de esa variable, lo que supone eliminar estadísticamente su efecto, el tercero es evitar su influencia aleatorizando su aparición en los tratamientos.

Los experimentos científicos utilizan estos tres tipos de control, los dos primeros para aquellas variables controladas por el experimentador y que potencialmente a priori van a tener mayor influencia. La aleatorización se reserva para eliminar el efecto de variables fuera de nuestro control y de poca influencia esperada, cuyos efectos se englobarán dentro del error experimental.

Existen, principalmente, tres clases de diseños experimentales:

- **Diseños con un factor principal**. Para aumentar su sensibilidad podemos conseguir comparaciones más homogéneas mediante el uso de variables "bloque". La diferencia principal entra un factor cualquiera y una variable bloque es que, en general, se supone que no hay interacción entre las variables bloques y el resto de las variables factores. Por ejemplo, si queremos comparar el rendimiento de unas determinadas máquinas eliminando el efecto del trabajador, diremos que el trabajador es una variable bloque. El modelo utilizado en este caso es de *bloque aleatorizados*.

- **Diseños con varios factores de interés e interacciones**. En este caso en el que existen varios factores igualmente importantes que además pueden interaccionar entre sí, se utilizan los *diseños factoriales*. Estos diseños son la base del presente proyecto y se tratarán en el apartado siguiente.

- **Diseños jerárquicos**. Se caracterizan porque ciertos factores aparecen anidados o jerarquizados dentro de otros, careciendo de sentido entonces la idea de interacción entre ellos. Además típicamente los niveles de los factores no son fijos, sino que se seleccionan al azar de una población. El modelo más importante es el de *componentes de la varianza*.

En cualquiera de estos diseños la metodología básica es: formular el modelo matemático que se va a utilizar, estimar los parámetros, contrastar que efectos pueden suponerse nulos y analizar las hipótesis básicas.

1.2. Diseños factoriales

Como se ha dicho antes los diseños factoriales son utilizados cuando tenemos varios factores de igual importancia a priori y que, además, pueden interaccionar entre sí.

La idea básica de los diseños factoriales es cruzar los niveles de los factores en todas las combinaciones posibles. Por ejemplo, supongamos que tenemos un tratamiento o factor con Q niveles (por ejemplo en una reacción química Q temperaturas), y además tenemos R reactores diferentes donde producir la reacción, sospechando además que el usar un reactor u otro puede influir en la variable respuesta.

Podemos seleccionar Q niveles de la variable temperatura y aplicar dichos niveles en cada uno de los R reactores, tendríamos así en total QxR experimentos. Esquemáticamente, el método es pues cruzar todas las combinaciones posibles de las

dos variables. Los datos podrían representarse en una tabla de doble entrada como la que se muestra a continuación.

	Temperatura 1	…	Temperatura Q
Reactor 1	y_{11}	…	Y_{1Q}
…	…	…	…
Reactor R	y_{R1}	…	Y_{RQ}

Tabla 1. 1. Diseño factorial básico.

La formulación matemática del modelo analizado en un diseño factorial con dos factores es:

$$y_{ij} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + u_{ij}$$

Este modelo descompone la respuesta en:

1) Un efecto global μ, que mide el promedio de respuesta para todas las unidades.
2) Un efecto α_i que mide el efecto incremental de la variable α, en función de su nivel, que podría ser la temperatura en el ejemplo anterior.
3) Un efecto β_j que mide el efecto incremental de la variable β, en función de su nivel, que sería el tipo de reactor en el caso anterior.
4) El efecto incremental de la interacción de los dos efectos anteriores $(\alpha\beta)_{ij}$
5) Un efecto aleatorio, u_{ij}, que recoge el efecto de todas las restantes causas posibles de variabilidad del experimento.

Para poner de manifiesto la importancia que puede tener la interacción entre los dos factores supongamos que en el ejemplo anterior del reactor cada una de las variables está a dos niveles, es decir, tenemos dos posibles temperaturas T_0 y T_1, y dos reactores R_0 y R_1.

Supongamos que experimentamos con todas las posibilidades obteniendo los siguientes valores de rendimiento:

	Temperatura T_0	Temperatura T_1
Reactor R_0	78	60
Reactor R_1	71	85

Tabla 1. 2. Diseño factorial para dos factores.

El efecto de aumentar la temperatura en (T_0, R_0) será: 60-78= -18 y el tipo de reactor 71-78= -7. En consecuencia al ser ambos valores negativos, podríamos concluir que aumentos de temperatura, o cambiar el tipo de reactor, en el rango de valores (R_0, R_1), (T_0, T_1), disminuyen el rendimiento.

Este resultado puede ser equivocado pues es posible que si aumentamos la temperatura y cambiamos el tipo de reactor conjuntamente y medimos el rendimiento en el punto (R_1, T_1) este sea superior (85).

Esto ocurrirá si los efectos no son aditivos, es decir, si existe interacción entre ambas variables como es en este caso.

Así, en presencia de interacción, el método de mover una variable sin alterar las demás puede ser profundamente erróneo.

En el caso de tener más variables aparecerán interacciones de orden superior, así, con tres variables tendremos tres interacciones de orden dos y una de orden tres. En este caso el modelo matemático sería:

$$y_{ij} = \mu + \alpha_i + \beta_j + y_k + (\alpha\beta)_{ij} + (\alpha\gamma)_{ik} + (\beta\gamma)_{jk} + (\alpha\beta\gamma)_{ijk} + u_{ij}$$

Los tres factores principales, la media y las cuatro interacciones se podrán estimar con un mínimo de 8 experimentos.

Cada una de las variables puede estar a distintos niveles. Así si la variable α está a 2 niveles, la β a 3 niveles y la γ a 4, el número total de experimentos será: 4x3x2 = 24.

En el apartado siguiente veremos diseños factoriales a dos niveles cuyo mayor interés desde el punto de vista práctico es que reducen mucho el número de experimentos a realizar.

1.3. Diseños factoriales a dos niveles

Como se ha dicho antes para realizar un diseño factorial general, el investigador selecciona un número fijo de "niveles" para cada una de un conjunto de variables (factores) y luego hace experimentos con todas las combinaciones posibles. Si hay I_1 niveles para la primera, I_2 para la segunda, …, y I_k para la k-ésima, el conjunto de todas las I_1 x I_2 x … x I_k condiciones experimentales se llama diseño factorial I_1 x I_2 x … x I_k. Por ejemplo, un diseño factorial 2 x 3 x 5 comprende 2 x 3 x 5= 30 experimentos

elementales y un diseño factorial 2x2x2 = 2^3 comprende 8 experimentos elementales. A partir de ahora trataremos con diseños en los que cada variable ocurre únicamente a dos niveles. Estos diseños son importantes por varias razones:

Requieren relativamente pocos experimentos elementales por cada factor, y a pesar de que no permiten explorar exhaustivamente una amplia región del espacio de variabilidad de los factores, pueden indicar tendencias y así determinar una dirección prometedora para futuros experimentos.

Forman la base de los diseños factoriales fraccionales a dos niveles. Estos diseño son frecuentemente de gran utilidad en los primeros momentos de una investigación, donde suele ser aconsejable estudiar en un primer intento un gran número de variables superficialmente en lugar de estudiar intensamente un pequeño número (que puede incluir o no las variables importantes).

Estos diseños y sus correspondientes fracciones pueden ser utilizados en bloques para construir diseños con un grado de complejidad que se ajuste a la sofisticación del problema.

Las observaciones producidas por estos diseños tienen una fácil interpretación.

En estos diseños utilizaremos los signos (+) y (-) para representar los dos niveles de cada factor. Si este es cuantitativo, (+) indicará el nivel superior y (-) el inferior, en otro caso, estos niveles se definen arbitrariamente (por ejemplo, si un factor es el tipo de catalizador podemos definir (+) = catalizador A; (-) catalizador B.

Por ejemplo, en el caso de un 2^2, la disposición estándar de las 4 observaciones es:

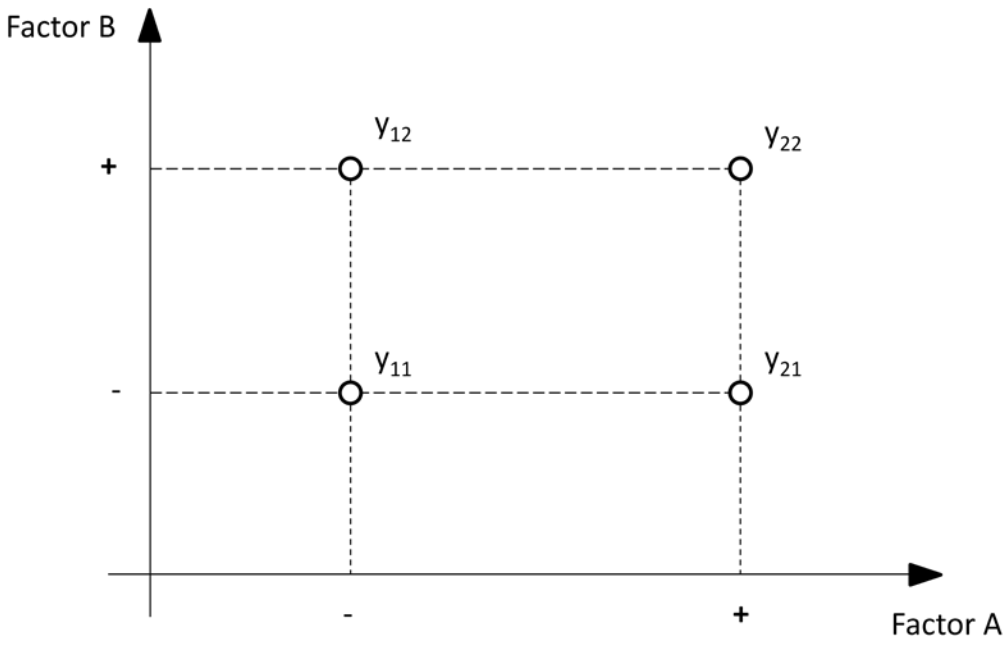


Figura 1. 1. Diseño factorial a dos niveles.

El modelo estadístico de este diseño en este caso es:

$y_{ij} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + u_{ij}$ i=1,2; j=1,2

Con las restricciones:

$\Sigma\ \alpha_i = \Sigma\ \beta_j = \Sigma(\alpha\beta)_{ij} = \Sigma(\alpha\beta)_{ij} = 0$

Estas restricciones se imponen para evitar la indeterminación en la estimación de los parámetros del modelo, la cual está fundamentada en la interpretación que los parámetros tienen, en el sentido de ser desviaciones a la media.

Así, los parámetros del modelo satisfacen:

$\alpha_1 = -\alpha_2$

$\beta_1 = -\beta_2$

$(\alpha\beta)_{i1} = (\alpha\beta)_{i2}$, i=1, 2

$(\alpha\beta)_{i1} = (\alpha\beta)_{i1}$, j =1, 2

Se define el efecto de cada factor como el incremento esperado de la respuesta cuando este pasa de (-) a (+).

Por tanto:

α = efecto de A = $\alpha_2 - \alpha_1 = 2\alpha_2$

β = efecto de B = $\beta_2 - \beta_1 = 2\beta_2$

$\alpha\beta$ = interacción de A y B = $(\alpha\beta)_{11} - (\alpha\beta)_{21} = (\alpha\beta)_{22} - (\alpha\beta)_{12} = -2(\alpha\beta)_{21} = 2(\alpha\beta)_{22}$

Si tenemos más de dos factores nos aparecerán interacciones de orden superior, así en el caso de tres factores tendremos tres interacciones de orden dos y una de orden tres.

Un procedimiento muy usado para obtener la estimación de los efectos de las diferentes interacciones y efectos principales de forma sencilla es el llamado *algoritmo de los signos* que funciona como sigue:

1) Se disponen los factores en una tabla de modo que el primer factor vaya alternando los signos (-) y (+), el segundo que alterne de dos en dos los signos (-) y (+), el tercero que los alterne de cuatro en cuatro y así sucesivamente con todos los factores, de manera que el k-ésimo alterne 2^{k-1} signos (-) con 2^{k-1} signos (+).

2) Se multiplican las columnas correspondientes para obtener cualquier interacción.

3) Cualquier efecto se estima multiplicando la columna de signos correspondiente por el vector de observaciones y dividiendo por 2^{n-1} el resultado obtenido, siendo n el número de factores.

Veamos esto con un ejemplo, supongamos un experimento factorial 2^3 en el que hay dos variables cuantitativas, temperatura (variable T) y concentración (variable C), y una cualitativa que es el tipo de catalizador (variable K). La respuesta es la cantidad de producto obtenido.

Supongamos que los resultados del experimento fueron:

Experimento	Temperatura (ºC) T	Concentración (%) C	Catalizador (A ó B) K	Producción (gramos) y
1	160(-)	20(-)	A(-)	60
2	180(+)	20(-)	A(-)	72
3	160(-)	40(+)	A(-)	54
4	180(+)	40(+)	A(-)	68
5	160(-)	20(-)	B(+)	52
6	180(+)	20(-)	B(+)	83
7	160(-)	40(+)	B(+)	45
8	180(+)	40(+)	B(+)	80

Tabla 1. 3. Diseño factorial a dos niveles para tres factores

Siguiendo los pasos 1) y 2) del algoritmo de los signos obtenemos la siguiente tabla con todas las interacciones.

Media	T	C	K	TC	TK	CK	TCK	Produc.
+	-	-	-	+	+	+	-	60
+	+	-	-	-	-	+	+	72
+	-	+	-	-	+	-	+	54
+	+	+	-	+	-	-	-	68
+	-	+	+	+	-	-	+	52
+	+	+	+	-	+	-	-	83
+	-	+	+	-	-	+	-	45
+	+	+	+	+	+	+	+	80
8	4	4	4	4	4	4	4	

Tabla 1. 4. Interacciones para un diseño factorial a dos niveles con tres factores.

La estimación de la media se calcula con la primera columna:

$$\text{Media} = \frac{60 + 72 + 54 + 68 + 52 + 83 + 45 + 80}{8} = \frac{514}{8} = 64.25$$

El efecto principal T se calcula con la segunda columna:

$$\text{T} = \frac{-60 + 72 - 54 + 68 - 52 + 83 - 45 + 80}{4} = \frac{92}{4} = 23$$

La interacción TC se calculará con la quinta:

$$\text{TC} = \frac{60 - 72 - 54 + 68 + 52 - 83 - 45 + 80}{4} = \frac{6}{4} = 1.5$$

De esta manera podemos calcular todas las interacciones obteniendo:

Efecto	**Estimación**	**Desviación Típica**
Media	64.25	0.7
T	23	1.4
C	-5	1.4
K	1.5	1.4
TC	1.5	1.4
TK	10	1.4
CK	0	1.4
TCK	0.5	1.4

Tabla 1. 5. Cálculo de interacciones para un diseño factorial a dos niveles con tres factores.

Para la estimación de la desviación típica se ha supuesto que el experimento ha sido realizado con dos replicaciones cuyo valor medio es el mostrado en la tabla anterior. Estas replicaciones son:

Nº experimento	1ª Replicación	2ª Replicación	Media
1	59	61	60
2	70	74	72
3	50	58	54
4	69	67	68
5	50	54	52
6	81	85	83
7	44	46	45
8	79	81	80

Tabla 1. 6. Valor de las replicaciones en cada experimento.

La expresión utilizada para la estimación de la desviación típica es (Box, Hunter & Hunter):

$$\sigma = \frac{4}{N} s^2$$

donde N es el número total de experimentos (16 en este caso) y s viene dado por la expresión:

$$s^2 = \frac{\sum di^2}{2g}$$

di es la diferencia entre cada una de las replicaciones y *g* es el número de grados de libertad, en nuestro caso 8 que son los factores que intervienen. Por tanto:

$$s^2 = \frac{2^2 + 4^2 + 8^2 + 2^2 + 4^2 + 4^2 + 2^2 + 2^2}{2 \times 8} = 8$$

Así:

$$\sigma^2 = \frac{4}{16} \times 8 = 2 \rightarrow \sigma = \sqrt{2} \approx 1.4$$

Veamos qué conclusiones podemos sacar de estos resultados.

La comparación de las estimaciones con sus desviaciones típicas sugiere que los efectos T, C y TK requieren una interpretación, mientras que el resto de los efectos podrán estar generados por errores experimentales.

El efecto principal de una variable debe ser interpretado individualmente sólo cuando no haya evidencia de que esta variable interacciona con otras. Cuando hay evidencia de una o más interacciones, las variables que interaccionan deben interpretarse conjuntamente.

En la tabla anterior hay un efecto grande de la temperatura, 23±1.4. Pero como la temperatura interacciona con el tipo de catalizador (TK=10±1.4), no hablaremos sobre el efecto de la temperatura individualmente. El efecto principal de la concentración es -5±1.4, y en este caso no hay evidencia de ninguna interacción en la que intervenga la concentración. Por tanto, podemos sacar las siguientes conclusiones:

1. El efecto de la concentración (C), cuando se produce un cambio de nivel, es reducir la producción en unas cinco unidades, y eso sucede independientemente de los niveles de otras variables.

2. Los efectos de la temperatura (T), y el catalizador (K) no se pueden interpretar separadamente debido a la existencia de la interacción TK. La interacción surge a causa de una diferencia de sensibilidad, en el resultado obtenido, al cambio de la temperatura de los dos catalizadores. Con el catalizador Al efecto de la temperatura sobre la variable respuesta es de 13 unidades, pero con el catalizador B es de 33 unidades.

Por último indicar en este apartado que en los diseños factoriales a dos niveles completos el número de efectos a calcular coincide con el número de experimentos a realizar 2^n y por tanto podemos estimar en todos los casos todas las interacciones por separado.

En el siguiente apartado trataremos fracciones factoriales a dos niveles que tienen la ventaja de que el número de experimentos a realizar es considerablemente menor aunque a costa de no poder estimar todos los parámetros del modelo separadamente.

1.4. Fracciones factoriales a dos niveles

En estudios exploratorios en la industria, donde se intenta estudiar el efecto simultáneo de muchos factores, los experimentos factoriales se encuentran con dos tipos de dificultades: la primera que el número de experimentos elementales puede ser muy alto, por ejemplo un diseño 2^6 requiere 64 observaciones; la segunda es que generalmente sólo unos pocos de los efectos estudiados son activos, siendo la mayoría de las interacciones de orden elevado cero o muy próximas a cero, con lo que parte de los datos del experimento se utilizan únicamente, para estimar el error experimental, en lugar de estimar parámetros relevantes.

Por ejemplo, en el caso anterior de la reacción química con los factores temperatura concentración y catalizador el efecto de la interacción entre los tres factores era prácticamente cero (TCK=0.5).

Supongamos un 2^6 en el que todas las interacciones de orden superior a dos sean nulas, casi 2/3 de los datos del experimento (42/64) se utilizarán en estimar el error experimental y solamente 1/3 (22/64 exactamente) de los datos se utilizan para estimar la media (1 parámetro), los efectos principales (6 parámetros) y las interacciones de segundo orden (15 parámetros).

Los diseños fraccionales o las fracciones de diseños factoriales tratan de resolver este problema: la idea es realizar únicamente una fracción del diseño factorial completo, elegida de manera que podamos dedicar el mayor número de grados de libertad posible a estimar parámetros de orden bajo, suponiendo cero las interacciones de orden elevado. Estos diseños se utilizan en las primeras etapas de una investigación para detectar los factores que tienen más importancia en la respuesta.

Para ver todo esto mejor vamos a poner un ejemplo; supongamos un diseño 2^4 en el que los factores son:

A: Velocidad de alimentación

B: Catalizador

C: Velocidad de agitación

D: Temperatura

Siendo la respuesta el tanto por ciento de reaccionado. Supongamos que los resultados del experimento son los que se muestran a continuación.

Experimento Elemental	VARIABLE				Respuesta % reacc.
	A	B	C	D	
*1	-	-	-	-	61
2	+	-	-	-	53
3	-	+	-	-	63
*4	+	+	-	-	61
5	-	-	+	-	53
*6	+	-	+	-	56
*7	-	+	+	-	54
8	+	+	+	-	61
9	-	-	-	+	69
*10	+	-	-	+	61
*11	-	+	-	+	94
12	+	+	-	+	93
*13	-	-	+	+	66
14	+	-	+	+	60
15	-	+	+	+	95
*16	+	+	+	+	98

Tabla 1. 7. Diseño factorial con cuatro factores.

El diseño 2^4 completo requiere 16 experimentos elementales con los que podemos estimar la media, los cuatro efectos principales, las seis interacciones de orden dos, las cuatro interacciones de orden tres y la interacción de orden cuatro obteniendo.

EFECTO	ESTIMACIÓN
Media	68.25
A	-1.5
B	17.5
C	-1.5
D	21.75
AB	3.25
AC	3.25
AD	-1.5
BC	0.75

BD	13.5
CD	2
ABC	0
ABD	0.75
ACD	-1.75
BCD	1.75
ABCD	0.5

Tabla 1. 8. Estimaciones en un diseño factorial con cuatro factores.

Supongamos que el investigador hubiera decidido realizar únicamente los 8 experimentos marcados con un asterisco en la tabla anterior, en este caso tendríamos el experimento:

Experimento Elemental	**VARIABLE**				**Respuesta % reacc.**
	A	B	C	D	
1	-	-	-	-	61
4	+	+	-	-	61
6	+	-	+	-	56
7	-	+	+	-	54
10	+	-	-	+	61
11	-	+	-	+	94
13	-	-	+	+	66
16	+	+	+	+	98

Tabla 1. 9. Diseño factorial fraccionado con cuatro factores.

Ahora como sólo tenemos 8 experimentos elementales sólo podemos estimar 8 efectos que, suponiendo el resto irrelevantes como se verá más adelante, será:

Media = 68.875	D = 21.75
A = 0.25	AB = 5.25
B = 15.75	AC = 16.75
C = -0.75	AD = -0.75

Tabla 1. 10. Estimaciones en un diseño factorial fraccionado con cuatro factores.

Comparando estos resultados con los del diseño completo 2^4 observamos que no hay muchas diferencias en cuanto a que los efectos que son significativos en un caso lo

son también en el otro (salvo el AC), por tanto parece que nos podíamos haber ahorrado la mitad del trabajo.

Veamos cómo se construyó esta fracción 2^{4-1}:

1. Se escribió un diseño completo para las tres variables A, B y C.
2. Se escribió la columna de signos para la interacción ABC y estos signos se utilizaron para definir los niveles de la variable D. Es decir, hicimos D = ABC.

1.4.1. Ecuación generatriz y confusión

En el ejemplo anterior parece que hayamos conseguido algo a cambio de nada. Veamos si esto es así o hemos perdido algo por el camino. Hemos realizado 8 experimentos elementales y hemos estimado 8 cantidades: la media, los 4 efectos principales y 3 interacciones de orden dos. Así pues quedan otras 8 cantidades para completar el 2^4 (tres interacciones de orden dos, cuatro de orden tres y una de orden cuatro).

Veamos que ha pasado con estas interacciones. Trataremos de estimar, por ejemplo, el valor de la interacción CD. Multiplicando los signos de las columnas 3 y 4 obtenemos la secuencia:

$$CD = + + - - - - + +$$

que es idéntica a:

$$AB = + + - - - - + +$$

Por tanto CD = AB y, como consecuencia se dice que las interacciones AB y CD están confundidas o son *alias*, así en el ejemplo anterior, la estimación 5.25 no sabemos si es debida a AC, a CD o a ambas.

Si obtenemos las columnas de signos correspondientes a todas las interacciones multiplicando los signos llegamos a los siguientes resultados que nos dan la estructura de alias completa o *patrón de confusión*.

A = BCD	AB = CD
B = ACD	AC = BD
C = ABD	AD = BC
D = ABC	I = ABCD

Suponiendo que las interacciones de orden tres o superior son nulas, podemos atribuir a los efectos principales la estimación en la que estos aparecen, puesto que están confundidos con interacciones de orden tres. En cambio no podemos decir lo mismo de las interacciones de orden dos puesto que estas aparecen confundidas entre sí.

Veamos ahora como podemos obtener el patrón de confusión de una forma más sencilla a través de la ecuación generatriz.

El diseño 2^{4-1} del ejemplo se construyó haciendo:

$$D = ABC$$

Esta relación se denomina *generador del diseño* o *ecuación generatriz*. Multiplicando los dos miembros por D, con la condición de que D x D = I (identidad) y, por ejemplo A x D = AD, obtenemos:

$$I = ABCD$$

Que es la habitual forma de escribir la ecuación generatriz. Una vez obtenida esta podemos ya obtener el patrón de confusión de cualquier efecto sin más que multiplicar dicho efecto por los dos miembros de la ecuación generatriz, así, por ejemplo:

$$AB \times I = AB \times ABCD \rightarrow AB = CD$$

De esta manera podemos obtener la estructura de alias complete de un modo sencillo.

En este ejemplo hemos utilizado media fracción del 2^4 obteniendo un diseño 2^{k-p} en el que k=4 y p=1. En este caso la ecuación generatriz tiene un único término. Veamos qué pasa cuando p tiene un valor superior a 1.

Una forma de construir un diseño 2^{k-p} cualquiera es escribir en primer lugar el 2^{k-p} completo con k-p variables y asociar después las p variables restantes con las interacciones entre las k-p variables anteriores. Dependiendo de las interacciones que escojamos el diseño será mejor o peor como veremos luego.

En este caso tendremos p términos independientes que forman la ecuación generatriz. Multiplicando todos estos términos independientes entre sí tendremos la ecuación generatriz completa que es la que usaremos para obtener la estructura de alias de cualquier efecto.

Veamos esto con un ejemplo. Supongamos una fracción 2^{6-3}, según lo anterior el procedimiento para construirla es comenzar con los ocho datos de un 2^3 con las variables A, B y C y asignar los tres factores adicionales (D, E y F) con sus interacciones. Así por ejemplo:

$$D = ABC$$

$$E = AB$$

$$F = BC$$

Con lo que la ecuación generatriz será:

$$I = ABCD = ABE = BDF$$

Multiplicando estos términos entre sí obtendremos los $2^p-1=7$ términos de la ecuación generatriz completa:

$$I = ABCD = ABE = CDE = BDF = ACF = ADEF = BDEF$$

Para realizar la multiplicación de los términos independientes entre sí, el método más cómodo es ir introduciendo cada uno de los términos e irlos multiplicando por todos los anteriores, así en el caso de tener tres términos independientes (A_1, A_2 y A_3) la ecuación generatriz completa vendrá dada por:

$$I = A_1 = A_2 = A_1\,A_2 = A_3 = A_1\,A_3 = A_2\,A_3 = A_1\,A_2\,A_3$$

Para obtener la estructura de confusión de cualquier efecto bastará con multiplicar dicho efecto por los términos de la anterior ecuación. Así, por ejemplo, en el caso anterior, la estructura de alias del efecto principal A será:

$$A = BCD = BE = ACDE = ABDF = CF = DEF = ABDEF$$

Otra manera de construir un diseño factorial fraccional es ir eligiendo directamente los términos independientes del diseño con los k factores, así por ejemplo, en el caso anterior podíamos haber hecho directamente:

$$I = ADE = CEF = BCED$$

La ecuación generatriz completa sería:

$$I = ADE = CEF = ACDF = BCED = ABC = BDF = ABEF$$

En este caso los experimentos a realizar son aquellos en los que se cumpla que los términos de la ecuación generatriz estén a nivel (+), que son:

Experimento Elemental	VARIABLE					
	A	**B**	**C**	**D**	**E**	**F**
1	+	+	+	-	-	-
2	-	-	+	+	-	-
3	-	+	-	-	+	-
4	+	-	-	+	+	-
5	+	-	-	-	-	+
6	-	+	-	+	-	+
7	-	-	+	-	+	+
8	+	+	+	+	+	+

Tabla 1. 11. Diseño factorial fraccionado con tres factores.

Veamos ahora el concepto de *Word Length Pattern* y *Resolución* de un diseño que nos proporcionará un criterio para elegir de una forma u otra los términos independientes de un diseño.

1.4.2. Word Length Pattern y Resolución de un diseño

Se define el *Word Length Pattern* de un diseño como un vector cuya componente i-ésima es el número de palabras de longitud i que hay en su ecuación generatriz completa. Así en el ejemplo anterior en el que la ecuación generatriz completa era:

$$I = ABCD = ABE = CDE = BDF = ACF = ADEF = BDEF$$

el Word Length Pattern será:

$$WLP = (0, 0, 4, 3, 0, 0)$$

Es decir, existen 4 palabras de longitud de tres y 3 palabras de longitud cuatro.

Se define la *Resolución* del diseño como la longitud de la palabra más corta de la ecuación generatriz completa. Así en el caso anterior la resolución del diseño será 3. Como se puede apreciar la resolución de un diseño y el Word Length Pattern depende de cómo hayan sido elegidos los términos independientes del mismo.

Así, supongamos un diseño 2^{5-2} construido de tres formas diferentes:

1) $I = ABCDE = ABCD = E \rightarrow WLP = (1, 0, 0, 1, 1)$

2) $I = ABCD = BCDE = AE \rightarrow WLP = (0, 1, 0, 2, 0)$

3) I = ABC = CDE = ABDE → WLP = (0, 0, 2, 1, 0)

La resolución será 1 en el primer caso, 2 en el segundo y 3 en el tercero.

Veamos cuál es el significado de la resolución de un diseño. En una fracción factorial interesa siempre que los efectos principales estén confundidos con las interacciones de mayor orden posible, pues a mayor orden de interacción, en general menor es su efecto y por tanto aumenta la probabilidad de que el factor principal esté correctamente estimado.

Así, si por ejemplo tenemos confundido un efecto principal con una interacción de orden cuatro sabremos con mucha certeza que la estimación se debe al efecto principal y no a la interacción de orden cuatro, pero si tenemos un efecto principal confundido con una interacción de orden dos, las estimación del factor principal puede estar muy sesgada y ya no podemos estar seguro de que dicha estimación se deba al efecto principal puesto que también puede ser debido a la interacción o a ambos.

La resolución de un diseño nos indicará con qué orden de interacción van a estar confundidos los factores principales interacciones de orden dos etc, o más bien con qué orden de interacción no van a estar confundidos. Así un diseño de resolución R es aquel en el que ningún efecto de f factores está confundido con ningún otro efecto que contenga menos de R – f factores. Por ejemplo:

1) Un diseño de resolución R = 3 no confunde los efectos principales entre sí, pero los confunde con interacciones de orden dos.

2) Un diseño de resolución R = 4 no confunde los efecto principales con interacciones de dos factores, pero confunde las interacciones de dos factores entre sí.

3) Un diseño de resolución V no confunde las interacciones de orden dos entre sí ni los efectos principales con interacciones de orden tres, pero si confunde las interacciones de orden dos con las de orden tres, etc.

Por tanto desde el punto de vista de la confusión interesa que un diseño tenga la máxima resolución posible.

Aparte de la resolución existen otros criterios de optimalidad de un diseño que los veremos en el apartado siguiente.

1.4.3. Criterios de optimalidad de un diseño

A la hora de construir una determinada fracción interesa principalmente que los efectos principales e interacciones de orden bajo estén confundidas con interacciones de orden superior. Para conseguir esto se siguen los criterios de máxima resolución y mínima aberración. Estos dos criterios están basados en unas hipótesis aceptables empíricamente:

1. Las interacciones de orden bajo son más importantes que las de orden alto.
2. Las interacciones del mismo nivel son igual de importantes.
3. Sólo hay un número relativamente pequeño de efectos significativos.

Veamos a continuación estos dos criterios.

1.4.3.1. Máxima resolución

Como se ha indicado antes, interesa que los diseños tengan máxima resolución desde el punto de vista de que los efectos principales o interacciones de orden bajo aparezcan confundidas con efectos de orden alto, para así poder estimar su efecto con elevada fiabilidad.

Para que un diseño sea interesante desde un punto de vista práctico debe tener una resolución superior a dos para que los efectos principales no aparezcan confundidos entre sí. De todas formas, es conveniente que la resolución sea como mínimo III o IV para que los efectos principales no estén confundidos entre sí o con interacciones de orden dos, cuyo efecto puede ser importante.

1.4.3.2. Mínima aberración

El concepto de aberración se usa para comparar dos diseños, así siempre hablaremos de que un diseño tiene mayor o menos aberración que otro pero no hablaremos de la aberración de un diseño por sí mismo.

Supongamos dos diseños 2^{k-p}, D_1 y D_2 de resolución máxima, cuyos Word length pattern vienen dados por:

$$WLP_1 = (A_1^{(1)}, A_2^{(1)}, A_3^{(1)}, \ldots, A_n^{(1)})$$

$$WLP_2 = (A_1^{(2)}, A_2^{(2)}, A_3^{(2)}, \ldots, A_n^{(2)})$$

entonces diremos que D_1 tiene menor aberración que D_2 si el primer "i" tal que $A_i^{(1)} \neq A_1^{(2)}$ verifica que $A_1^{(1)} < A_1^{(2)}$.

Si además ningún diseño tiene menor aberración que D_1, dentro de los 2^{k-p}, se dice que el diseño es de aberración mínima.

Este concepto de aberración mínima se usa para clasificar diseños de resolución máxima (los diseños de aberración mínima son de resolución máxima). Así es una forma de clasificar este tipo de diseños puesto que estos no se comportan igual desde el punto de vista del número de interacciones de orden bajo confundidas con los efectos principales o con las interacciones de segundo o tercer orden.

Normalmente lo más interesante es que los diseños tengan la mínima aberración, es decir, sería interesante conocer un diseño de aberración mínima conocidos los valores

de k y p. Por desgracia no existe ningún método para conseguirlo aunque Fries y Hunter (1980) proporcionaran un algoritmo para obtenerlo en algunos casos.

Veamos un ejemplo en el que podemos comprobar la importancia de conseguir diseños con aberración mínima.

Consideremos los dos diseños 2^{7-2} siguientes:

$$D_1: I = DEFG = ABCDF = ABCEG$$

$$D_2: I = ABCF = ADEG = BCDEFG$$

Ambos tienen resolución 4 pero Word length pattern distintos:

$$WLP_1 = (0, 0, 0, 1, 2, 0, 0)$$

$$WLP_2 = (0, 0, 0, 2, 0, 1, 0)$$

Según lo dicho anteriormente el diseño D_1 tiene menor aberración que el D_2. Al ser ambos diseños de resolución 4 los efectos principales no estarán confundidos con interacciones de orden dos, aunque estas sí que estarán confundidas entre sí. Pero mientras que en el diseño D_1 solo aparecen tres pares de interacciones de orden dos confundidas entre sí (DE = FG, EF = DG y DF = EG) en el diseño D_2 aparecen nada menos que seis (AB = CF, AC = BF, AD = EG, AE= DG, AF=BD y AG = DE).

Esto es debido a que en el diseño D_1 únicamente existe una palabra de longitud 4 mientras que en el D_2 aparecen dos.

Por último cabe hablar también en este apartado del concepto de aberración mínima débil.

Hasta ahora hemos hablado de dos criterios para optimizar diseños: *máxima resolución* y *mínima aberración*. Estos dos criterios están basados en las hipótesis comentadas anteriormente.

En general podemos decir que un diseño de aberración mínima es un buen diseño salvo que estas hipótesis se incumplan de un modo claro. Sin embargo, existen situaciones en la práctica en que estas condiciones se alejan bastante de la realidad y se pueden encontrar diseños mejores que el de aberración mínima.

Así existen casos en los que interesa tener unas determinadas interacciones libres de confusión con interacciones de bajo orden, en estos casos pueden interesar diseños con peor aberración. Por ejemplo, supongamos un diseño 2^{6-2} en el que las interacciones AC, AD, AG BC, CD, CE, DE, EF interesa obtenerlas por separado.

Se prueban los siguientes diseños:

$$\text{Diseño 1: } I = ABE = BCDF = ACDEF$$

$$\text{Diseño 2: } I = ABCE = BCDF = ADEF$$

cuyos WLP son:

$$WLP_1 = (0, 0, 1, 1, 1, 0)$$

$$WLP_2 = (0, 0, 0, 3, 0, 0)$$

Haciendo la estructura de confusión para interacciones de orden dos (o menores) obtenemos:

DISEÑO 1	**DISEÑO 2**
E = AB	AB = DE
B = AE	AC = BE
BC = DF	AD = EF
BD = CF	AE = BC =DF
A = BE	AF =DE
BF = CD	BD = CF
	BF = CD

Tabla 1. 12. Estructura de confusión para interacciones de orden dos.

Se puede observar como en este caso es mejor utilizar el diseño 1, que es de mayor aberración (y menor resolución), que el diseño 2.

Aunque, como acabamos de ver, puede haber casos en los que no siempre es mejor utilizar diseños de mínima aberración, en general siempre suelen ser los que mejor se adaptan s los primeros pasos de un determinado proceso experimental en el que a priori no se sabe que efectos pueden ser más importantes. Es por eso que la mayoría de las investigaciones en este campo se centran en buscar pautas o algoritmos para encontrar este tipo de diseños en todos los casos de k y p.

1.5. Gráfico con todos los diseños 2^{k-p} con 16 observaciones

Uno de los resultados de la tesis llamada *Método computacional para la obtención de fracciones factoriales a dos niveles* llevada a cabo por David Santos Vaquero y dirigida por Jesús Juan Ruiz contiene todos los posibles diseños no equivalentes para fracciones factoriales $2^{k-p} = 16$ observaciones.

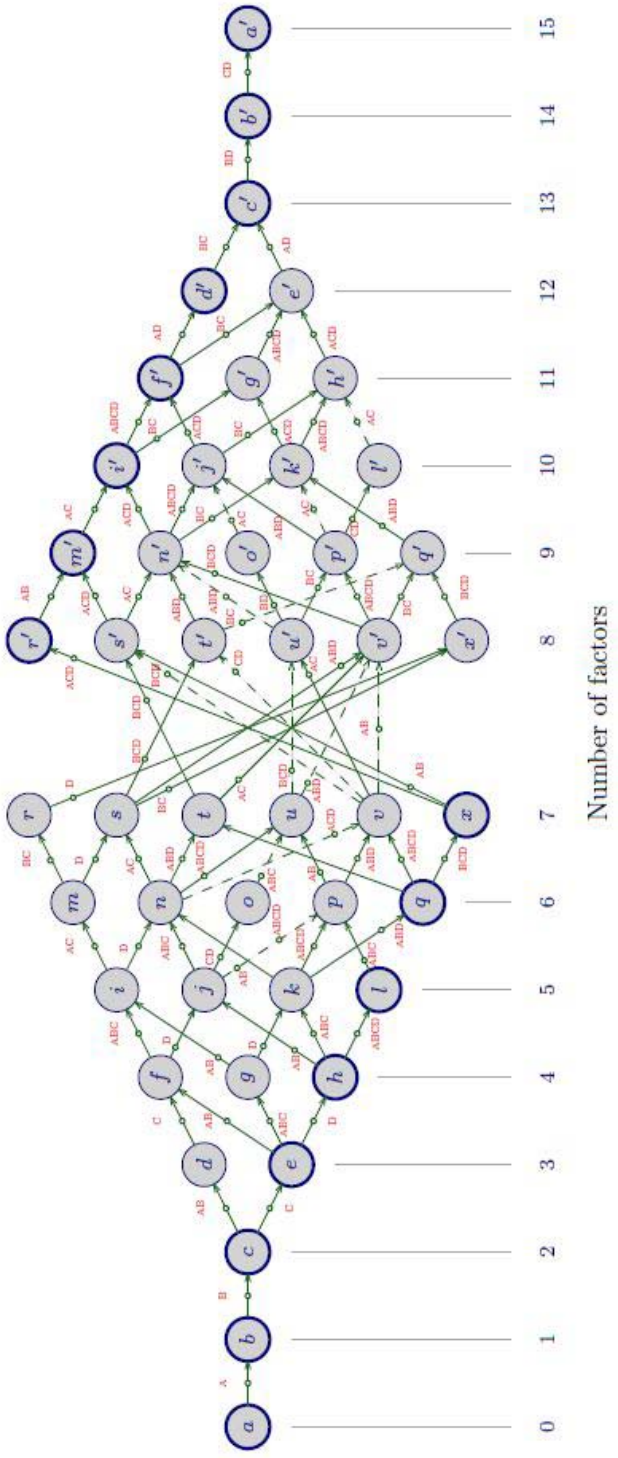


Figura 1. 2. *Diseños factoriales fraccionados a dos niveles para 16 observaciones.*

La idea básica es empezar desde un diseño nulo para acabar en un conjunto H_4 (siendo H_m un diseño con 2^m-1 elementos, en este caso 15). Cada punto o nodo del gráfico, identificado con una letra, es un diseño.

El punto inicial es un conjunto vacío (para k=0 factores). Para k=1, de todos los diseños no equivalentes se ha elegido uno de ellos {A}, que corresponde al nodo b. Para k=2, todos los conjuntos de dos elementos dan un diseño equivalente a {A, B} que da lugar al nodo c. Para k = 3 ya existen dos alternativas de diseños no equivalentes d={A, B, AB} y e={A, B, C}. Para k = 4 hay tres diseños no equivalentes representados por los nodos f, g y h. Y así continua hasta llegar a k=15 que es el conjunto H_4 del que hablábamos anteriormente.

Las flechas conectan dos diseños que solo difieren en un elemento: el diseño con k+1 columnas (es decir, al que se le ha añadido u factor) se obtiene añadiendo al diseño la palabra que aparece en el cuerpo de la flecha. Considerando lo anterior, se concluye que es posible llegar a cualquier diseño del gráfico desde el nodo inicial “a”. Y no sólo eso, prestando atención al gráfico puede observarse que existen diferentes caminos para llegar a un mismo nodo. Así por ejemplo el diseño u={A, B, C, D, AB, ABC, ABCD} puede ser alcanzado desde n={A, B, C, D, AB, ABC} o desde p={A, B, C, D, ABC, ABCD}. Sin embargo, el diseño obtenido desde “o” y que corresponde a una flecha discontinua es diferente de los otros dos. El camino que llega desde el no “o” acaba en el nodo “u”. Por lo tanto, para definir un diseño se toman los elementos que forman el camino que une el diseño o nodo con el origen, con la condición de que todas las líneas sean continuas excepto la última, que puede ser continua o discontinua. En el último caso, el diseño obetnido es nominalmente diferente pero equivalente matemáticamente a los otros.

Los puntos a resaltar del gráfico son los siguientes:

1. El número total de diseños no equivalentes para n = 16 observaciones se reduce a 46 (incluyendo el nulo).

2. El número de diseños a elegir para un determinado número de factores k es muy pequeño. En el gráfico, los diseños con el mismo número de factores están en la misma línea vertical. Por ejemplo para k=6 vemos que podemos obtener 5 diseños diferentes. El máximo abanico de posibilidades a la hora de elegir corresponde a las fracciones de 6 o 7 factores, para un número total de diseños no equivalentes de 6.

3. Una de las propiedades a destacar del gráfico es la simetría, debido al hecho que si dos diseños d_1 y d_2 son equivalentes, también lo son sus diseños complementarios $\overline{d}_1$ y $\overline{d}_2$. La notación $\overline{d}$ se usará para representar el complementario de un diseño dado *d*, donde $\overline{d}=H_m\backslash d$. Por ejemplo, el complementario de:

$$m'=\{A, B, C, D, AB, ABC, ABD, ACD, BCD\}$$

es el diseño

$$\bar{m}'=\{AC, AD, BC, BD, CD, ABCD\}$$

Nótese que $\bar{m}'$ está formado por las palabras de las flechas que unen el nodo *m'* con el *a'*. El diseño complementario de *m'* (que es el diseño $\bar{m}'$) es equivalente a m. Asimismo, dentro de los subconjuntos de H_4, hay 4 diseños diferentes para k=5, y otros 4 diferentes para k=10, que son los complementarios de los primeros. En el gráfico el complementario de un diseño de k palabras es el nodo simétricamente opuesto, que tendrá de 15-k palabras. Las relaciones entre los diseños complementarios fueron analizadas por Tang and Wu (1996). En concordancia con esto, las propiedades de cada uno de estos diseños $k\geq 8$ (la parte derecha del gráfico) puede ser deducida de la otra mitad k<8 (el lado izquierdo).

4. Por cada valor k de la figura se ha identificado el diseño de Mínima Aberración con un borde del círculo en negrita. El complementario (o simétrico) del diseño de Mínima Aberración es el peor diseño desde el punto de vista matemático de la aberración. Todos los diseños con $k\geq 8$ tiene R=III con la excepción del diseño k=8 que tiene R=IV. Al diseño r' se le conoce como diseño saturado con resolución IV. Se puede observar que todos los diseños de mínima aberración para k>8 pueden ser obtenidos de el. En el gráfico podemos ver que todas las palabras que forman el diseño r' contiene un número impar de letras.

5. Los diseños de la parte superior izquierda del gráfico *a, b, c, d, e, f, g, i, m* y *r* están exclusivamente formados por los primeros 3 factores A, B y C, lo que supone que estrictamente hablando no son fracciones factoriales de 2^4. Estos diseños corresponden a la familia de los diseños de 8 observaciones (subconjuntos de H_3) y su parte simétrica está formada por palabras pares de H_4, subconjunto isomórfico de H_3.

1.5.1. Conclusiones

El gráfico mostrado presenta un rango de diseños eficientes y fáciles de analizar e interpretar cuando se trabajen con experimentos con 16 observaciones. El problema habitual al que se suele enfrentar el experimentador es que modelo elegir. Atendiendo a la combinatoria podría parecer que las variables son grandes. Sin embargo, a la luz de los resultados, se puede observar que las opciones son sorprendentemente bajas.

Se ha decidido continuar el estudio en esta línea, concretamente la de los diseños factoriales 2^{k-p} con 32 observaciones, porque a priori, el número de alternativas que se ofrecerán al experimentador serán mucho mayores, además de ofrecer un estudio de un número de factores lo suficientemente grande para cubrir cualquier posible problema de la realidad.

Además pensamos que puede ser interesante entender mejor la estructura e ilustrar algunos conceptos de estos diseños factoriales de dos niveles.

2. PROGRAMAS

A lo largo del desarrollo de este Trabajo de Fin de Grado *"Obtención de fracciones de diseños factoriales a dos niveles con 32 observaciones"* se han desarrollado 3 programas.

Los 3 programas han sido desarrollados en Matlab. La elección de este software tiene su justificación por su capacidad para trabajar con matrices de más de dos índices, las cuales han jugado un papel fundamental en el desarrollo del programa principal del trabajo. El autor tenía conocimientos básicos de la programación en C, adquiridos durante la carrera, y le ha resultado muy fácil aprender a usar esta herramienta. Matlab es un entorno de trabajo sencillo de manejar: la programación se lleva cabo en un lenguaje muy sencillo e intuitivo, con una inicialización de variables muy simple.

Volviendo a los 3 programas que se han llevado a cabo, se va a describir brevemente su funcionamiento. El primer programa, al que se le ha bautizado como *"CaminosNodo"*, ofrece los diseños *k+1* con su correspondiente *Word Length Pattern* a los que es posible llegar desde un diseño concreto con *k* factores. El segundo programa, llamado *"DisenosN32"*, permite obtener todos los diseños posibles para diseños factoriales $n=2^{k-p}$ con n=16,32, 64, etc. Se ha optado por el nombre de 32, porque era el objetivo inicial del proyecto, si bien es cierto, que el gráfico que se ha estudiado anteriormente para n=16 experimentos, fruto del trabajo de David Santos Vaquero, es muy sencillo de obtener con el programa que se ha implementado para la obtención de los diseños con n=32 experimentos. Por último se ha creado, un tercer programa, *"Complementarios"*, que calcula los diseños complementarios de un diseño dado, con su correspondiente *Word Length Pattern*.

Los dos primeros programas son de gran utilidad. La mayor parte del código del primer programa, está incluido en el segundo, ya que es la herramienta principal que utiliza para calcular nuevos diseños. ¿Cuál es la razón entonces, de incluir este programa de forma individual? Existen tres razones fundamentales. La primera es que tiene una enorme utilidad práctica, ya que si en el mundo real, nos encontramos en un experimento concreto, se podrá deducir fácilmente en que diseño y con qué cantidad de factores (*k*) contamos. Si se tiene intención de añadir una nueva variable *k* al experimento, gracias a este programa, se podrá desplegar un abanico con todas las opciones posibles, ordenadas matemáticamente en base a los criterios de máxima resolución y mínima aberración. La segunda razón, es que la comprensión de la ejecución de este programa, puede ayudar enormemente a entender como se ha construido el programa principal *"DisenosN32"*, lo cual es tremendamente interesante dado que es el programa más relevante del trabajo. Por último, y no por ello menos importante, el interés matemático que tiene ver como se interrelacionan los diferentes diseños entre sí.

El tercer programa fue desarrollado para subsanar errores y verificar que el programa principal funcionaba correctamente de acuerdo a la teoría que explica las relaciones entre nodos simétricos. Ésta fue abordada ligeramente al final del capítulo

uno, y será desarrollada con profundidad antes de explicar cómo funciona el tercer programa.

Con cada programa, primero se procederá a una explicación de cómo ejecutar el programa a nivel usuario, para después explicar cómo se ha programado: que criterios matemáticos se han usado, que variables se han utilizado etc.

La explicación estará acompañada de un ejemplo práctico para que sea más fácil de entender.

2.1. PROGRAMA "CAMINOSNODO"

El programa *"CaminosNodo"* pretende ofrecer un catálogo de los diseños *k+1* con su correspondiente *Word Lenght Pattern*, ordenados en base a los criterios de máxima resolución y mínima aberración, a los que se es posible acceder desde un diseño concreto de *k* factores.

2.1.1. Ejecución del programa

Para ejecutar este programa bastará con seguir los siguientes pasos:

1. Abrir Matlab.
2. Abrir el archivo de la siguiente manera *File>Open* (buscarlo en la carpeta en la que se ha guardado).
3. Ejecutarlo haciendo click en *Run* que puede aparecer en la barra de comandos de Matlab si es una versión moderna o haciendo *Debug>Run*.

2.1.2. Funcionamiento del programa

Una vez se ejecute el programa este pide que se introduzcan por pantalla el número de factores k y el número de generadores p del diseño en el que se encuentra el usuario.

Se va a suponer que el usuario va a partir del diseño:

Ejemplo = {A, B, C, D, E, AB, ABC}

que como se puede comprobar tiene k=7 factores y p=2 generadores, $2^{7-2} = 32$.

Los intervalos límites para los valores de k y p en el caso de n=32 experimentos son $k \in [6, 31]$ y $p \in [1,26]$ respectivamente.

En el ejemplo dado, aparecerán las siguientes preguntas a la que se responderá con los resultados correspondientes:

Ingrese el numero de factores:7

Ingrese el numero de generadores:2

A continuación el programa solicitará que el usuario ingrese las dos palabras, a las que se ha llamado generadoras.

Nótese que para comodidad del usuario, no será necesario introducir las letras puras A, B, C, D y E pues formarán parte de todos los diseños. Esta decisión ha sido tomada con el objetivo de hacer la búsqueda una tarea más rápida.

Volviendo a las palabras generadoras, cabe recordar que en base a la teoría del capítulo uno, en un diseño factorial $2^{k-p} = 32$, cada nueva palabra que se añade a partir de

la E, se está confundiendo con una interacción de las 5 variables puras, por eso, hay que añadir su letra correspondiente.

Es decir, en el ejemplo que nos atañe, no valdría con ingresar AB y ABC, puesto que el significado real de este diseño sería que se han añadido dos variables extra para un experimento de 32 replicaciones. La primera variable extra que se ha añadido, F, se ha confundido con la interacción de segundo orden entre A y B. Mientras que la segunda variable extra que se ha añadido, G, se ha confundido con la interacción de tercer orden entre A, B y C.

Es por ello que F = AB equivale a decir I = ABF y G = ABC equivale a decir I = ABCG, siendo I la identidad.

Son estas dos palabras, las que el usuario debería ingresar en el ejemplo que se está estudiando.

Ingrese el vector generador en mayúsculas (Acuérdese de la letra extra): ABF

Ingrese el vector generador en mayúsculas (Acuérdese de la letra extra): ABCG

Una vez el usuario ha introducido todas las palabras que conforman el diseño, el programa realizará las operaciones pertinentes, y mostrará por pantalla los resultados, que en el caso particular serán:

Los diseños nuevos son añadiendo una de estas palabras:

ans =

A CDE H

A DE H

AB DE H

A CD H

DE H

A D H

AB D H

A C H

WLPnuevaspalabrasordenadas =

0	*0*	*2*	*1*	*2*	*2*	*0*	*0*
0	*0*	*2*	*2*	*1*	*1*	*1*	*0*
0	*0*	*2*	*2*	*2*	*0*	*0*	*1*
0	*0*	*2*	*3*	*2*	*0*	*0*	*0*
0	*0*	*3*	*1*	*0*	*2*	*1*	*0*
0	*0*	*3*	*2*	*1*	*1*	*0*	*0*
0	*0*	*3*	*3*	*0*	*0*	*1*	*0*
0	*0*	*4*	*3*	*0*	*0*	*0*	*0*

Los diseños a los que se podría acceder añadiendo una nueva palabra al Ejemplo = {A, B, C, D, E, AB, ABC} aparecerán mostrados en la interfaz de la pantalla.

Como puede observarse, los diseños aparecerán ordenados de mejor a peor en base a los criterios de máxima resolución (recuérdese que la resolución de un diseño es el tamaño de la palabra de menor longitud que conforma su Word Length Pattern) y mínima aberración (criterio comparativo usado en el caso que dos diseños tengan la misma resolución; tendrá menos aberración aquel con menos palabras de longitud igual a la resolución del diseño).

En el gráfico que se estudió en el capítulo uno para diseños factoriales 2^{k-p} con n=16 replicaciones, los diseños aparecían ordenados en base a los mismos criterios de optimalidad de máxima resolución y mínima aberración.

En el caso de las palabras que se muestran por pantalla, puede parecer que la vista no es la más pulcra ya que en ocasiones aparecen huecos entre letras. Se entendió que puede resultar más visual e intuitivo, que las letras que no deban aparecer en una palabra aparezcan representadas con un espacio en blanco. Puede resultar de gran utilidad cuando se manejen diseños de gran cantidad de palabras candidatas, como puede ser un diseño con k= 14 que contará con 67 palabras candidatas.

A modo de síntesis y para que no quepa ninguna duda, en el ejemplo que se había tomado hasta ahora, si se quería obtener el diseño de máxima resolución y mínima aberración habría que optar por añadir la última palabra I = ACDEH (H = ACDE) lo que supondría llegar hasta un diseño:

Ejemplofinal = {A, B, C, D, E, AB, ABC, ACDE}

con su correspondiente *Word Length Pattern*:

WLP (Ejemplofinal) = (0 0 2 1 2 2 0 0)

2.1.3. Estructura y análisis del código del programa

Se va a proceder a explicar el código del programa. Para ello, en primera instancia se explicarán las variables utilizadas así como las funciones auxiliares que Matlab pone a disposición del usuario y que han sido requeridas para el desarrollo del programa.

En segundo lugar, se explicarán los pequeños pasos que el programa recorre hasta llegar al resultado final. *"CaminosNodo"* ha sido dividido en varias partes para facilitar su comprensión.

2.1.3.1 Variables utilizadas

- **k**: Número de factores o variables.
- **p**: Número de palabras generadoras.
- **disenoinicial**: Diseño desde el que partimos. Será una matriz de p filas y k columnas que contendrá las palabras generadoras almacenadas de forma binaria.
- **z**: Es una variable auxiliar para ser usada en el bucle.
- **name**: Una variable auxiliar en la que se almacenan las palabras generadoras en forma de letras para ser convertidas a código binario.
- **x**: Contiene el número de filas de disenoinicial.
- **y**: Contiene el número de columnas de disenoinicial.
- **M**: La matriz que contiene todas las 31 palabras posibles que pueden generarse por combinación lineal de 5 letras (A, B, C, D y E). En código binario.
- **P**: Matriz identidad que se corresponde con las letras A, B, C, D y E en código binario.
- **M1**: Matriz auxiliar que se corresponde con M sin A, B, C, D y E.
- **candidatas**: Matriz que contiene las palabras a testar. Es decir, las candidatas a ver si generan nuevos diseños o no.
- **contador**: Variable auxiliar usada en el bucle para ordenar.
- **candidatasanadido**: Cantidad de columnas de ceros que hay que añadir a candidatas para añadirle la fila de unos que corresponderá a la nueva variable.
- **candidatasanadido2**: Columna de unos que corresponderá a la nueva variable que se está confundiendo con nuestras posibles candidatas.

- **candidatascompleta**: Matriz resultado de añadir a candidatas, candidatasanadido y candidatasanadido2
- **disenoinicialanadido**: Columna de ceros que hay que añadir a diseñoinicial que tenga la misma dimensión de candidatascompleta.
- **disenoinicialcompleto**: Resultado de añadir a disenoinicial, disenoinicialanadido.
- **numero**: variable auxiliar usada en el bucle
- **disenotestado**: Posible diseño a incluir en nuevaspalabras
- **G**: Ecuaciones generatrices de cada diseño testado.
- **tope**: tamaño de WLPnuevaspalabras
- **diff**: variable auxiliar para que nos indica si es igual=1 o diferente=0
- **diferente**: Suma el número de diff para saber si es diferente.
- **nuevaspalabras**: Matriz donde se almacenan de forma binaria las palabras que añadiéndolas se generan nuevos diseños para k+1 factores.
- **WLPnuevaspalabras**: Matriz donde se almacenan el WLP de las nuevas palabras
- **nuevaspalabrasordenadas**: nuevaspalabras ordenadas por los criterios de optimalidad.
- **WLPnuevaspalabrasordenadas**: WLP correspondientes a nuevaspalabrasordenadas.
- **indices**: Variable auxiliar para ordenar los diseños en función de los criterios de optimalidad.
- **colocación**: Variable auxiliar para el bucle for.

2.1.3.2. Funciones auxiliares

- **input('texto'):** Muestra el texto por pantalla
- **for:** Bucle que repite una sentencia un número determinado de veces.
- **double(letras):** Convierte letras del abecedario a código ASCII.
- **length(vector):** Calcula la longitud de un vector.
- **sum(vector):** Calcula la suma de los elementos de un vector.

- **size(matriz, 1/2):** Devuelve el numero de filas de la matriz (1) o el número de columnas (2).
- **xor(vector1, vector2):** Realiza la operación binaria xor entre dos vectores.
- **any(v(i)):** Devuelve un 1 si el elemento del vector es distinto de 0 y un 0 si es igual a 0.
- **if:** Si se cumple la condición a la derecha del if se ejecutan las sentencias que aparecen hasta el siguiente end.
- **isequal(vector1, vector2):** Devuelve un 1 si vector1=vector2, en caso contrario devuelve un 0.
- **sortrows():** Ordena las filas de una matriz de menor a mayor.

2.1.3.3. Explicación

1º PARTE: El usuario introduce el diseño en el que se encuentra

Mediante la función input el usuario introducirá manualmente las palabras generadoras.

Ingrese el vector generador en mayúsculas (Acuérdese de la letra extra): ABF

Ingrese el vector generador en mayúsculas (Acuérdese de la letra extra): ABCG

Lo que hace el programa es transformar estas variables a código binario, ya que es mucho más sencillo operar de esta manera. Para ello el programa transforma las letras a código ASCII y luego le otorga unos a la casilla de la matriz que corresponda a una letra.

Según lo explicado anteriormente el programa obtendría:

disenoinicial =

1 1 0 0 0 1 0

1 1 1 0 0 0 1

Como podemos observar la primera fila de la matriz equivale a ABF mientras que la segunda contiene la palabra ABCG.

2º PARTE: Crea la matriz de candidatas a generar un nuevo diseño

Entre las líneas 17 y 32 de código, lo que hace el programa es obtener la matriz que contiene a H_5 que como se explico en el capítulo 1, es el conjunto que contiene al diseño factorial completo para 5 factores y 32 replicaciones.

Es decir, H_5 = {A, B, C, D, E, AB, AC, AD, AE, BC, BD, BE, CD, CE, DE, ABC, ABD, ABE, ACD, ACE, ADE, BCD, BCE, BDE, CDE, ABCD, ABCE, ABDE, ACDE, BCDE, ABCDE}

Esta matriz se genera a través de las 5 letras contenidas en forma binaria en la matriz P. De forma que en estas líneas se obtendrían las palabras de H_5 que serían almacenadas en la matriz M, que sería una matriz de 31x5.

Entre las líneas 33 y 43 de código, se eliminan de la matriz M las filas correspondientes a las letras A, B, C, D y E. En este punto la matriz M cuenta con una dimensión de 26x5.

Finalmente entre las líneas 44 y 54, se le quitará a M, las palabras que se repiten, es decir, el resto de palabras de nuestro diseño, AB y ABC en el caso particular que se está estudiando. Se obtendrá de esta manera una matriz a la que se ha llamado candidatas que contiene todas las palabras que se van a evaluar a la hora de formar nuevos diseños. En el caso de estudio, candidatas sería una matriz de 24x5.

3º PARTE: Ajusta las dimensiones de las matrices candidatas y disenosiniciales

Como se ha analizado, la matriz candidatas esta formada por 5 columnas, que equivale a 5 letras. Se necesita añadirle la letra que se requiere por tratarse de una nueva palabra.

Para facilitar la comprensión acudiremos al ejemplo práctico que se está desarrollando en este hilo. Si el diseño desde el que el usuario parte es Ejemplo = {A, B, C, D, E, AB, ABC}, como ya se ha comprobado en diversas ocasiones, F = AB y G = ABC, por lo que añadir una nueva variable a un experimento de este tipo, supone confundir una nueva variable H con la palabra que se va a añadir. Es esta la razón de completar la matriz candidatas con unos en la fila correspondiente a la H.

Se debe añadir columnas de ceros en las letras, que ya se hayan usado para el diseño desde el que se parte. Las columnas de ceros se añaden por medio de candidatasnadido y la columna de unos por medio de candidatasanadido2. Posteriormente, se concatenan las tres matrices para obtener el resultado deseado en candidatas completo.

candidatas =

1	*0*	*1*	*0*	*0*
0	*1*	*1*	*0*	*0*
1	*0*	*0*	*1*	*0*

Dimensión 24x5

candidatascompleta =

1	*0*	*1*	*0*	*0*	*0*	*0*	*1*
0	*1*	*1*	*0*	*0*	*0*	*0*	*1*
1	*0*	*0*	*1*	*0*	*0*	*0*	*1*

Dimensión 24x8

El ejemplo superior es para el caso particular. Se sobreentiende que la cantidad de ceros que habrá que añadir dependerá de las palabras generadoras de nuestro diseño inicial.

Por otro lado, como se va a operar entre la matriz de disenoinicial y candidatas completa, se necesita que las dimensiones sean la mismas. Es esta la razón de añadir a disenoinicial una columna de ceros al igual que se ha visto en el ejemplo anterior.

disenoinicial =

$$\begin{matrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{matrix}$$

Dimensión 2x7

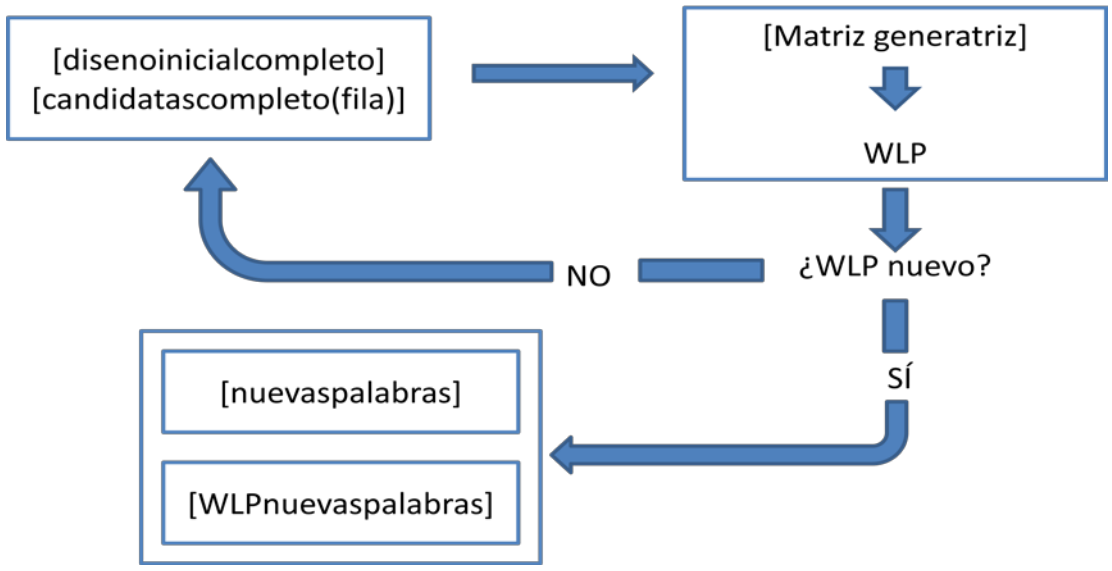
disenoinicialcompleto =

$$\begin{matrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{matrix}$$

Dimensión 2x8

4º PARTE: Se obtienen las palabras que generan nuevos diseños

El diagrama de flujo de esta parte se puede observar a continuación.



El programa sustrae cada fila de candidatascompleta y la concatena con disenoinicialcompleto para testar un posible futuro diseño.

Se calcula las ecuaciones generatrices de este diseño y genera su Word Length Pattern.

Si es un diseño nuevo almacena tanto el diseño como su WLP en las matrices nuevaspalabras y WLPnuevaspalabras respectivamente. Si el WLP es repetido, el diseño será equivalente a uno ya introducido, por lo que se desechará. Se probará entonces con la siguiente fila de la matriz candidatascompleto.

Se obtendrá en esta etapa los diseños finales con el pequeño inconveniente de que aparecerán desordenados.

nuevaspalabras =

1	*0*	*1*	*0*	*0*	*0*	*0*	*1*
1	*0*	*0*	*1*	*0*	*0*	*0*	*1*
1	*1*	*0*	*1*	*0*	*0*	*0*	*1*
1	*0*	*1*	*1*	*0*	*0*	*0*	*1*
0	*0*	*0*	*1*	*1*	*0*	*0*	*1*
1	*0*	*0*	*1*	*1*	*0*	*0*	*1*
1	*1*	*0*	*1*	*1*	*0*	*0*	*1*
1	*0*	*1*	*1*	*1*	*0*	*0*	*1*

WLPnuevaspalabras =

0	*0*	*4*	*3*	*0*	*0*	*0*	*0*
0	*0*	*3*	*2*	*1*	*1*	*0*	*0*
0	*0*	*3*	*3*	*0*	*0*	*1*	*0*
0	*0*	*2*	*3*	*2*	*0*	*0*	*0*
0	*0*	*3*	*1*	*0*	*2*	*1*	*0*
0	*0*	*2*	*2*	*1*	*1*	*1*	*0*
0	*0*	*2*	*2*	*2*	*0*	*0*	*1*
0	*0*	*2*	*1*	*2*	*2*	*0*	*0*

5º PARTE: Se ordenan los diseños anteriores

Se ordenan en función de los criterios máxima resolución y mínima aberración usando la función sortrows() de Matlab.

El resultado sería el siguiente:

nuevaspalabrasordenadas =

1	*0*	*1*	*1*	*1*	*0*	*0*	*1*
1	*0*	*0*	*1*	*1*	*0*	*0*	*1*
1	*1*	*0*	*1*	*1*	*0*	*0*	*1*
1	*0*	*1*	*1*	*0*	*0*	*0*	*1*
0	*0*	*0*	*1*	*1*	*0*	*0*	*1*
1	*0*	*0*	*1*	*0*	*0*	*0*	*1*
1	*1*	*0*	*1*	*0*	*0*	*0*	*1*
1	*0*	*1*	*0*	*0*	*0*	*0*	*1*

WLPnuevaspalabrasordenadas =

0	*0*	*2*	*1*	*2*	*2*	*0*	*0*
0	*0*	*2*	*2*	*1*	*1*	*1*	*0*
0	*0*	*2*	*2*	*2*	*0*	*0*	*1*
0	*0*	*2*	*3*	*2*	*0*	*0*	*0*
0	*0*	*3*	*1*	*0*	*2*	*1*	*0*
0	*0*	*3*	*2*	*1*	*1*	*0*	*0*
0	*0*	*3*	*3*	*0*	*0*	*1*	*0*
0	*0*	*4*	*3*	*0*	*0*	*0*	*0*

6º PARTE: Transforma nuevaspalabras a letras

Transformando cada uno en función de donde este colocado a código ASCII, posteriormente hace una lectura de la matriz como char, transformando finalmente la matriz de nuevaspalabrasordenadas a letras.

Los diseños nuevos son añadiendo una de estas palabras:

A CDE H
A DE H
AB DE H
A CD H
DE H
A D H
AB D H
A C H

2.2. PROGRAMA "DisenosN32"

Se trata del programa principal del trabajo. Es la herramienta que permite encontrar todos los diseños con diferente *Word Lenght Pattern* para diseños factoriales fraccionados a dos niveles con 32 observaciones.

Como ya se explicará más adelante en el análisis de resultados, cabe la posibilidad de que dos diseños sean diferentes a pesar de tener el mismo *Word Length Pattern.* Este es un suceso que ocurre con una frecuencia extremadamente baja. La utilidad práctica de encontrar los diseños diferentes con el mismo Word Length Pattern es nula y la pérdida de eficiencia del programa sería muy notoria, por lo que no se han incluido esos diseños, sin embargo si se han calculado aparte.

El programa ha calculado 820 diseños con diferente *Word Length Pattern* para los diseños factoriales fraccionales 2^{k-p} con 32 observaciones. Aparecen 9 diseños extra, por el fenómeno ya mencionado, que comparten *Word Length Pattern* a pesar de ser diseños diferentes.

La versatilidad del programa es muy amplia porque permite no solo calcular todos los diseños posibles, si no hasta un número determinado de variables, lo cual puede resultar interesante cuando se sepa el número máximo de variables en un experimento.

2.2.1. Ejecución del programa

Para ejecutar este programa bastará con seguir los siguientes pasos:

1. Abrir Matlab.
2. Abrir el archivo de la siguiente manera *File>Open>DisenosN32* (al ser una función el archivo debe ser guardado en la carpeta donde busca Matlab, por defecto suele ser *work*).
3. Ejecutarlo en la *Command Window*, llamando a la función como si se tratase de otra cualquiera:

DisenosN32(factores)

2.2.2. Funcionamiento del programa

El programa ha sido desarrollado a modo de función para que se ejecute de manera más rápida y eficiente en cuanto a rendimiento se refiere. Además a nivel usuario supone una forma de trabajar mucho más intuitiva.

Se ha definido la función de la siguiente manera:

function [disenosfinalesletras,diseno,WLPdiseno] = disenoN32(factores)

Se ha incluido una variable de entrada:

- **factores:** es el número de variables o factores k hasta los que le gustaría al usuario que se construyeran los posibles diseños. El intervalo límite de entrada es factores ∈ [7,31]. En k = 5, se tiene el diseño factorial completo para 5 variables {A, B, C, D y E}. Mientras que en k = 6, se obtienen los cuatro primeros diseños fraccionados: {A, B, C, D, E, AB}, {A, B, C, D, E, ABC}, {A, B, C, D, E, ABCD}, {A, B, C, D, E, ABCDE}, que son los que se incluye en el código como variables iniciales. Es por eso que el límite inferior de factores será 7; k = 7 son los primeros diseños que pueden ser calculados a partir de las variables iniciales que se han usado.

Se han incluido 3 variables de salida:

- **disenosfinalesletras:** Almacena todos los diseños con diferente *Word Length Pattern* que existen hasta el número de factores de entrada que se haya introducido.

- **diseno:** Contiene los mismos datos que *disenosfinalesletras* pero en código binario. Puede ser útil trabajar de esta manera en otros ámbitos.

- **WLPdiseno:** Almacena los *Word Length Pattern* de los diseños de *disenosfinalesletras*.

A modo de ejemplo si ejecutamos la función disenosN32 para el caso más simple, *factores = 7*:

[disenosfinalesletras,diseno,WLPdiseno] = disenosN32(7)

Los resultados que se obtendrían serían los siguientes:

disenosfinalesletras(:,:,1,7) =

ABC F
AB DE G

diseno(:,:,1,7) =

1	*1*	*1*	*0*	*0*	*1*	*0*
1	*1*	*0*	*1*	*1*	*0*	*1*

disenosfinalesletras(:,:,2,7) =

ABC F
A DE G

diseno(:,:,2,7) =

1	*1*	*1*	*0*	*0*	*1*	*0*
1	*0*	*0*	*1*	*1*	*0*	*1*

disenosfinalesletras(:,:,3,7) =

ABC F
AB D G

diseno(:,:,3,7) =

1	*1*	*1*	*0*	*0*	*1*	*0*
1	*1*	*0*	*1*	*0*	*0*	*1*

disenosfinalesletras(:,:,4,7) =

AB F
A CDE G

diseno(:,:,4,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
1	*0*	*1*	*1*	*1*	*0*	*1*

disenosfinalesletras(:,:,5,7) =

AB F
CDE G

diseno(:,:,5,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
0	*0*	*1*	*1*	*1*	*0*	*1*

disenosfinalesletras(:,:,6,7) =

AB F
A CD G

diseno(:,:,6,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
1	*0*	*1*	*1*	*0*	*0*	*1*

disenosfinalesletras(:,:,7,7) =

AB F
CD G

diseno(:,:,7,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
0	*0*	*1*	*1*	*0*	*0*	*1*

disenosfinalesletras(:,:,8,7) =

AB F
A C G

diseno(:,:,8,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
1	*0*	*1*	*0*	*0*	*0*	*1*

WLPdiseno(:,:,7) =

0	*0*	*0*	*1*	*2*	*0*	*0*	*0*
0	*0*	*0*	*2*	*0*	*1*	*0*	*0*
0	*0*	*0*	*3*	*0*	*0*	*0*	*0*
0	*0*	*1*	*0*	*1*	*1*	*0*	*0*
0	*0*	*1*	*1*	*0*	*0*	*1*	*0*
0	*0*	*1*	*1*	*1*	*0*	*0*	*0*
0	*0*	*2*	*0*	*0*	*1*	*0*	*0*
0	*0*	*2*	*1*	*0*	*0*	*0*	*0*

Resulta obvio que la utilidad del programa pasa por trabajar con un cantidad de variables o factores k infinitamente superior, pero no se ha querido abrumar al lector con resultados, si no realizar una mera explicación de cómo funciona el programa a nivel usuario.

2.2.3 Estructura y análisis del código del programa

Como en el caso anterior, primero se explicarán las variables y funciones utilizadas para posteriormente continuar con la explicación del código del programa.

2.2.3.1 Variables utilizadas

- **diseno:** Es una matriz de 4 índices [nº de filas, nº de columnas, posición del diseño, nº variables (k)]. Es la variable que almacena todos los diseños en código binario.
- **disenosfinalesletras:** Es una matriz de 4 índices [nº de filas, nº de columnas, posición del diseño, nº variables (k)]. Es la variable que almacena todos los diseños en forma de palabras.
- **WLPdiseno:** Es una matriz de 3 índices [nº filas, nº columnas, nº variables (k)]. Está directamente relacionada con diseno, ya que guarda sus Word Length Pattern.
- **letras:** el número de variables puras, que no están confundidas.
- **P:** contiene el número de letras en binario en forma de matriz. Se utiliza para generar M.
- **M:** crea la matriz que contiene el diseño factorial completo en binario. Una matriz que contiene H_m y que por tanto será dependiente de letras. Las dimensiones de M serán 2^{letras}-1x*letras*.
- **k**: Número de factores o variables.
- **p**: Número de palabras generadoras.
- **disenoinicial**: Diseño desde el que partimos. Será una matriz de p filas y k columnas que contendrá las palabras generadoras almacenadas de forma binaria.
- **z**: Es una variable auxiliar para ser usada en el bucle.
- **x**: Contiene el número de filas de diseno.
- **y**: Contiene el número de columnas de diseno.
- **M1**: Matriz auxiliar que se corresponde con M sin A, B, C, D y E.
- **candidatas**: Matriz que contiene las palabras a testar. Es decir, las candidatas a ver si generan nuevos diseños o no.
- **contador**: Variable auxiliar usada en el bucle para ordenar.
- **candidatasanadido**: Cantidad de columnas de ceros que hay que añadir a candidatas para añadirle la fila de unos que corresponderá a la nueva variable.
- **candidatasanadido2**: Columna de unos que corresponderá a la nueva variable que se está confundiendo con nuestras posibles candidatas.
- **candidatascompleta**: Matriz resultado de añadir a candidatas, candidatasanadido y candidatasanadido2

- **disenoanadido**: Columna de ceros que hay que añadir a diseno para que tenga la misma dimensión de candidatascompleta.
- **disenocompleto**: Resultado de concatenar a disenol, disenoanadido.
- **numero**: variable auxiliar usada en el bucle
- **disenotestado**: Posible diseño a incluir en nuevaspalabras
- **generatriz**: Ecuaciones generatrices de cada diseño testado.
- **tope**: tamaño de WLPnuevaspalabras
- **diff**: variable auxiliar para que nos indica si es igual=1 o diferente=0
- **diferente**: Suma el número de diff para saber si es diferente.
- **nuevaspalabras**: Matriz donde se almacenan de forma binaria las palabras que añadiéndolas se generan nuevos diseños para k+1 factores.
- **WLPnuevaspalabras**: Matriz donde se almacenan el WLP de las nuevas palabras.
- **disenosrepetidos**: Almacena todos los diseños a los que se llega desde cada diseño k-1 concreto.
- **WLPdisenosrepetidos**: Almacena los WLP de disenosrepetidos.
- **disenostotales**: Almacena los diseños que no se repiten para un determinado nº de factores k. Elimina los disenosrepetidos de diseños.
- **WLPtotales:** Almacena los WLP de disenostotales.
- **disenostotalesordenados**: disenostotales ordenados por los criterios de optimalidad.
- **WLPdisenostotalesordenados**: WLP correspondientes a nuevaspalabrasordenadas.
- **disenosparaletras**: Se almacenan los disenostotalesordenados en código ASCII.
- **disenosletras:** Se almacenan los disenostotalesordenados en forma de palabras.
- **indices**: Variable auxiliar para ordenar los diseños en función de los criterios de optimalidad.
- **colocación**: Variable auxiliar para el bucle for.

2.2.3.2. Funciones auxiliares

- **for:** Bucle que repite una sentencia un número determinado de veces.
- **sum(vector):** Calcula la suma de los elementos de un vector.
- **size(matriz, 1/2):** Devuelve el numero de filas de la matriz (1) o el número de columnas (2).
- **xor(vector1, vector2):** Realiza la operación binaria xor entre dos vectores.
- **any(v(i)):** Devuelve un 1 si el elemento del vector es distinto de 0 y un 0 si es igual a 0.
- **if:** Si se cumple la condición a la derecha del if se ejecutan las sentencias que aparecen hasta el siguiente end.
- **isequal(vector1, vector2):** Devuelve un 1 si vector1=vector2, en caso contrario devuelve un 0.
- **sortrows():** Ordena las filas de una matriz de menor a mayor.

2.2.3.3. Explicación del código

1º PARTE: Diseño de partida

En la primera tienen lugar tres hitos importantes para el correcto funcionamiento del programa.

Se introducen los diseños de partida para generar después el resto de diseños factoriales fraccionales para 32 observaciones. Se parte desde el caso k = 6:

{A, B, C, D, E, AB}	diseno(:,:,1,6)=[1 1 0 0 0 1];
{A, B, C, D, E, ABC}	diseno(:,:,2,6)=[1 1 1 0 0 1];
{A, B, C, D, E, ABCD}	diseno(:,:,3,6)=[1 1 1 1 0 1];
{A, B, C, D, E, ABCDE}	diseno(:,:,4,6)=[1 1 1 1 1 1];

A partir del mismo, se generarán el resto de diseños. Es importante resaltar que las matrices donde se almacenarán los nuevos diseños son de 4 índices. La estructura es de la siguiente manera: [filas, columnas, nº del diseño en base a criterios de optimalidad, nº de variables o factores].

Por otro lado se inicializan dos bucles vitales para el correspondiente funcionamiento del programa.

El bucle correspondiente a q, indica el número de variables o factores con los que se está trabajando, de manera que trabajar con q = 8 supondría trabajar con 8

variables (de las cuales 5 serían puras y las otras 3 estarían confundidas con las palabras generadoras).

El bucle correspondiente a w, indica dentro de un número concreto de variables k, en que diseño nos encontramos.

Por último, en esta primera parte del programa, tienen lugar la declaración al vacío de las variables que se necesita que se reseteen con cada inicialización de q.

Es importante clarificar que el procedimiento que se seguirá a lo largo del programa será: calcular todos los diseños a los que se puede llegar desde cada nodo, ver cuales se repiten, quitar los repetidos y finalmente ordenarlos.

2º PARTE: Crea la matriz de candidatas a generar un nuevo diseño

Entre las líneas 22 y 37 de código, lo que hace el programa es obtener la matriz que contiene a H_5 que como se explico en el capítulo 1, es el conjunto que contiene al diseño factorial completo para 5 factores y 32 replicaciones.

Es decir, H_5 = {A, B, C, D, E, AB, AC, AD, AE, BC, BD, BE, CD, CE, DE, ABC, ABD, ABE, ACD, ACE, ADE, BCD, BCE, BDE, CDE, ABCD, ABCE, ABDE, ACDE, BCDE, ABCDE}

Esta matriz se genera a través de las 5 letras contenidas en forma binaria en la matriz P. De forma que en estas líneas se obtendrían las palabras de H_5 que serían almacenadas en la matriz M, que sería una matriz de 31x5.

Entre las líneas 37 y 46 de código, se eliminan de la matriz M las filas correspondientes a las letras A, B, C, D y E. En este punto la matriz M cuenta con una dimensión de 26x5.

3º PARTE: Se eliminan las palabas repetidas para cada diseño

Finalmente, se le quitará a M, las palabras que se repiten, es decir, el resto de palabras de nuestro diseño, AB y ABC en el caso particular que se está estudiando. Se obtendrá de esta manera una matriz a la que se ha llamado candidatas que contiene todas las palabras que se van a evaluar a la hora de formar nuevos diseños. En el caso de estudio, candidatas sería una matriz de 24x5.

4º PARTE: Ajusta las dimensiones de las matrices candidatas y diseno

Como se ha analizado, la matriz candidatas está formada por 5 columnas, que equivale a 5 letras. Se necesita añadirle la letra que se requiere por tratarse de una nueva palabra.

Para facilitar la comprensión acudiremos al ejemplo práctico que se está desarrollando en este hilo. Si el programa se encuentra frente al diseño Ejemplo = {A, B, C, D, E, AB, ABC}, como ya se ha comprobado en diversas ocasiones, F = AB y G = ABC, por lo que añadir una nueva variable a un experimento de este tipo, supone

confundir una nueva variable H con la palabra que se va a añadir. Es esta la razón de completar la matriz candidatas con unos en la fila correspondiente a la H.

Se debe añadir columnas de ceros en las letras, que ya se hayan usado para el diseño desde el que se parte. Las columnas de ceros se añaden por medio de *candidatasnadido* y la columna de unos por medio de *candidatasanadido2*. Posteriormente, se concatenan las tres matrices para obtener el resultado deseado en *candidatascompleto*.

candidatas =

1	*0*	*1*	*0*	*0*
0	*1*	*1*	*0*	*0*
1	*0*	*0*	*1*	*0*

Dimensión 24x5

candidatascompleta =

1	*0*	*1*	*0*	*0*	*0*	*0*	*1*
0	*1*	*1*	*0*	*0*	*0*	*0*	*1*
1	*0*	*0*	*1*	*0*	*0*	*0*	*1*

Dimensión 24x8

El ejemplo superior es para el caso particular. Se sobreentiende que la cantidad de ceros que habrá que añadir dependerá de las palabras generadoras de nuestro *diseno*.

Por otro lado, como se va a operar entre la matriz de *diseno* y *candidatascompleta*, se necesita que las dimensiones sean la mismas. Es esta la razón de añadir a *diseno* una columna de ceros al igual que se ha visto en el ejemplo anterior.

disenoinicial =

1	*1*	*0*	*0*	*0*	*1*	*0*
1	*1*	*1*	*0*	*0*	*0*	*1*

Dimensión 2x7

disenoinicialcompleto =

1	*1*	*0*	*0*	*0*	*1*	*0*	*0*
1	*1*	*1*	*0*	*0*	*0*	*1*	*0*

Dimensión 2x8

5º PARTE: Se obtienen las palabras que generan nuevos diseños

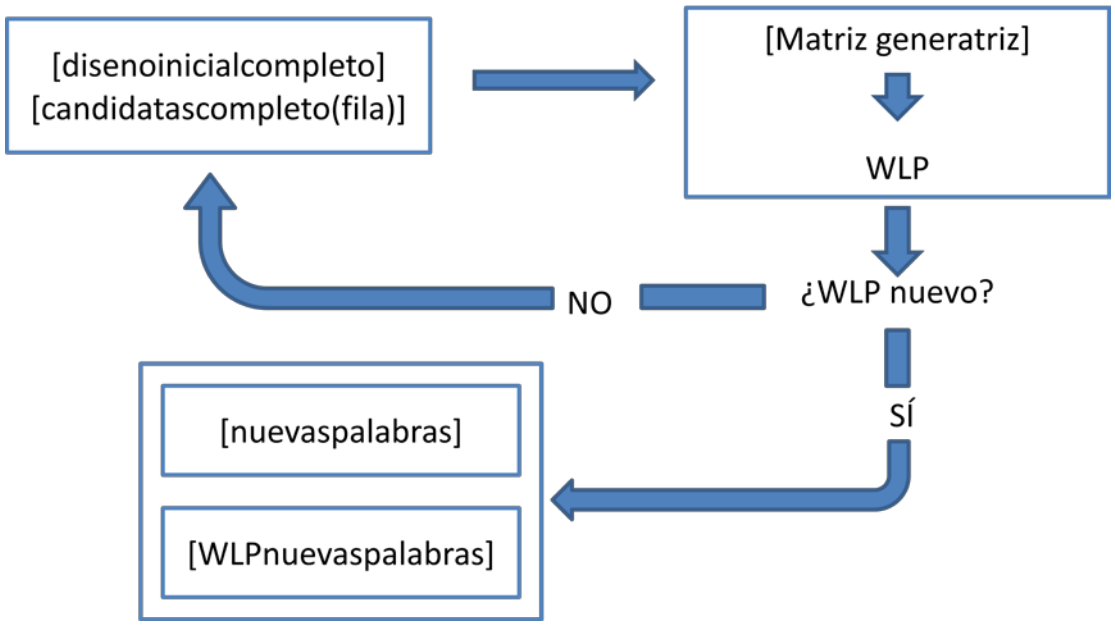
El programa sustrae cada fila de *candidatascompleta* y la concatena con *disenoinicialcompleto* para testar un posible futuro diseño.

Se calcula las ecuaciones generatrices de este diseño y genera su *Word Length Pattern.*

Si es un diseño nuevo almacena tanto el diseño como su WLP en las matrices *nuevaspalabras* y *WLPnuevaspalabras* respectivamente. Si el WLP es repetido, el diseño será equivalente a uno ya introducido, por lo que se desechará. Se probará entonces con la siguiente fila de la matriz *candidatascompleto.*

Se obtendrá en esta etapa los diseños finales con el pequeño inconveniente de que aparecerán desordenados.

El diagrama de flujo de esta parte se puede observar a continuación:



7º PARTE: Quitar los diseños que se repiten

Para un determinado número de factores k, cada diseño desde el que se parta podrá acceder a un número de diseños determinado de *k+1* factores. Habrá diseños para *k+1* factores a los que se lleguen de distintos diseños de *k* factores.

Según esta diseñado el programa, todos los diseños *k+1* a los que se puede llegar desde un diseño *k* concreto se almacenarán en una variable llamada *disenosrepetidos*.

Parece trivial deducir que para un determinado número de factores, *disenosrepetidos* contará con los mismos diseños escritos varias veces. Es por ello, que en esta parte, los diseños repetidos se eliminan.

Los diseños k+1 a los que se podrá llegar quedaran almacenados en *disenostotales.*

Todo lo anterior puede extrapolarse para los *Word Length Pattern* que quedarán almacenados en *WLPtotales*.

A modo de ejemplo se va a mostrar cómo se pasa del Word Length Pattern con gran cantidad de diseños repetidos al Word Length Pattern sin diseños repetidos.

8º PARTE: Ordenar los diseños por criterios de optimalidad

Se ordenan en función de los criterios máxima resolución y mínima aberración usando la función sortrows() de Matlab. Los mejores diseños quedarán almacenados en primer lugar para facilitar la tarea al experimentador.

A modo de ejemplo se va a mostrar una pequeña tabla para el caso de factores=8, en la que se podrá observar como después de ejecutar la función el orden cambia. En este caso, se incluirá solo el Word Lenght Pattern para no abrumar al lector con demasiados diseños.

WLPtotales =

0	0	2	2	2	0	0	1
0	0	2	1	2	2	0	0
0	0	1	2	3	1	0	0
0	0	1	3	2	0	1	0
0	0	0	3	4	0	0	0
0	0	2	2	1	1	1	0
0	0	0	5	0	2	0	0
0	0	3	3	0	0	1	0
0	0	2	3	2	0	0	0
0	0	0	7	0	0	0	0
0	0	0	6	0	0	0	1
0	0	3	1	0	2	1	0
0	0	3	2	1	1	0	0
0	0	4	3	0	0	0	0

WLPtotalesordenados =

0	0	0	3	4	0	0	0
0	0	0	5	0	2	0	0
0	0	0	6	0	0	0	1
0	0	0	7	0	0	0	0
0	0	1	2	3	1	0	0
0	0	1	3	2	0	1	0
0	0	2	1	2	2	0	0
0	0	2	2	1	1	1	0
0	0	2	2	2	0	0	1
0	0	2	3	2	0	0	0
0	0	3	1	0	2	1	0
0	0	3	2	1	1	0	0
0	0	3	3	0	0	1	0
0	0	4	3	0	0	0	0

9º PARTE: Convertir los diseños a letras

Mediante una conversión a código ASCII y luego a cadena de caracteres los diseños se convertirán a letras para facilitar la labor al experimentador.

10º PARTE: Guarda los diseños

En esta última parte, almacena los diseños obtenidos en la variable diseno para que puedan ser utilizados como diseños de entrada en la siguiente inicialización del bucle.

A modo de ejemplo para el caso de 7 factores los resultados obtenidos serían:

disenosfinalesletras(:,:,1,7) =

ABC F
AB DE G

diseno(:,:,1,7) =

1	1	1	0	0	1	0
1	1	0	1	1	0	1

disenosfinalesletras(:,:,2,7) =

ABC F
A DE G

diseno(:,:,2,7) =

1	1	1	0	0	1	0
1	0	0	1	1	0	1

disenosfinalesletras(:,:,3,7) =

ABC F
AB D G

diseno(:,:,3,7) =

1	*1*	*1*	*0*	*0*	*1*	*0*
1	*1*	*0*	*1*	*0*	*0*	*1*

disenosfinalesletras(:,:,4,7) =

AB F
A CDE G

diseno(:,:,4,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
1	*0*	*1*	*1*	*1*	*0*	*1*

disenosfinalesletras(:,:,5,7) =

AB F
CDE G

diseno(:,:,5,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
0	*0*	*1*	*1*	*1*	*0*	*1*

disenosfinalesletras(:,:,6,7) =

AB F
A CD G

diseno(:,:,6,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
1	*0*	*1*	*1*	*0*	*0*	*1*

disenosfinalesletras(:,:,7,7) =

AB F
CD G

diseno(:,:,7,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
0	*0*	*1*	*1*	*0*	*0*	*1*

disenosfinalesletras(:,:,8,7) =

AB F
A C G

diseno(:,:,8,7) =

1	*1*	*0*	*0*	*0*	*1*	*0*
1	*0*	*1*	*0*	*0*	*0*	*1*

WLPdiseno(:,:,7) =

0	*0*	*0*	*1*	*2*	*0*	*0*	*0*
0	*0*	*0*	*2*	*0*	*1*	*0*	*0*
0	*0*	*0*	*3*	*0*	*0*	*0*	*0*
0	*0*	*1*	*0*	*1*	*1*	*0*	*0*
0	*0*	*1*	*1*	*0*	*0*	*1*	*0*
0	*0*	*1*	*1*	*1*	*0*	*0*	*0*
0	*0*	*2*	*0*	*0*	*1*	*0*	*0*
0	*0*	*2*	*1*	*0*	*0*	*0*	*0*

2.3. PROGRAMA "COMPLEMENTARIOS"

El programa *"Complementarios"* no tiene una utilidad práctica tan marcada como los dos programas anteriores. Se desarrolló como una herramienta para verificar el correcto funcionamiento de los problemas anteriores.

Para explicar la comprobación que se realizó se va a tener presente el gráfico de los diseños factoriales fraccionales $2^{k-p} = 16$ observaciones.

Ya se habló en el capítulo uno, de las propiedades simétricas del gráfico. Una propiedad fundamental es que dado un diseño *m*, su diseño complementario $\bar{m}$, tiene el mismo Word Lenght Pattern que su diseño simétrico *m'*.

De acuerdo a esta propiedad, es sencillo deducir los procesos de comprobación que se han llevado a cabo. Para cada grupo de diseños k, se calculaban sus complementarios con su correspondiente Word Length Pattern, y se verificaba que coincidían con los simétricos, que eran calculados por el programa principal.

2.3.1. Ejecución del programa

Para ejecutar este programa bastará con seguir los siguientes pasos:

4. Abrir Matlab.
5. Abrir el archivo de la siguiente manera *File>Open>Complementarios* (al ser una función el archivo debe ser guardado en la carpeta donde busca Matlab, por defecto suele ser *work*).
6. Ejecutarlo en la *Command Window*, llamando a la función como si se tratase de otra cualquiera:

Complementarios(limite,letras,disenoentrada)

2.3.2. Funcionamiento del programa

El programa ha sido guardado como una función ya se estimó que podía ser una herramienta útil para alguien que trabajara en estos diseños en el futuro.

Una función se guarda de la siguiente manera en Matlab:

function [variables de salida] = nombre (variables de entrada)

Las variables de entrada que se han establecido son tres:

- **limite:** El programa está preparado para ser ejecutado en forma de bucle (t = 1:limite) de manera que si tenemos varios diseños almacenados en diferentes índices de la misma matriz (disenosentrada(:,:,t) puede recorrerlos hasta el valor final de t. Si se quiere calcular el complementario de un solo diseño, el valor de t deber ser igual a 1.

- **letras:** El programa necesita saber si estamos en diseños factoriales fraccionales de 16 observaciones (letras = 4, $2^4 = 16$), de 32 observaciones (letras = 5, $2^5 = 32$), etc

- **disenoentrada:** El diseño del que quiere calcularse el complementario.

A modo de ejemplo, se va a calcular el diseño complementario y su correspondiente Word Length Pattern para el diseño j de diseños factoriales fraccionales de 16 observaciones (se ha usado un ejemplo de 16 observaciones para no abrumar al lector con mucha cantidad de palabras).

Si el diseño j = {A, B, C, D, AB} se introduce en la *Command Window* de Matlab, lo siguiente:

```
disenoentrada=[eye(letras); 1 1 0 0];
```

Una vez se conoce el diseño del que se quiere calcular el complementario, ejecutamos la función:

```
[WLP,complementario]=funcioncomplementario(1,4,disenoentrada)
```

A lo que Matlab rápidamente responderá:

```
WLP =

     0     0     9    16    15    12     7     3     1     0


complementario =
CD
B D
BC
BCD
A  D
A C
A CD
AB D
ABC
ABCD
```

Si quisiera conocerse el valor de alguna variable más del programa, como por ejemplo, el número complementario en binario, que es como se trabaja en los otros dos programas, bastaría con añadir la variable complementario entre los corchetes de variables de salida.

2.3.3. Estructura y análisis del código del programa

Como en los casos anteriores, primero se explicarán las variables y funciones utilizadas para posteriormente continuar, con la explicación del código del programa.

2.3.3.1. Variables utilizadas

- **t:**
- **disenoentrada:**
- **letras:** el número de variables puras, que no están confundidas.
- **P:** contiene el número de letras en binario en forma de matriz. Se utiliza para generar M.
- **M:** crea la matriz que contiene el diseño factorial completo en binario. Una matriz que contiene H_m y que por tanto será dependiente de letras. Las dimensiones de M serán 2^{letras}-1x*letras*.
- **contador:** variable auxiliar para los bucles.
- **diseno:** la matriz del disenoentrada concatenada a la matriz que contiene las letras en solitario. Se asume el usuario no incluirá las variables puras en el diseño de entrada.
- **complementario:** Contiene el diseño complementario.
- **complementarioletras:** Contiene el diseño complementario en código ASCII por si se quiere mostrar por pantalla en forma de letras.
- **cambio:** Matriz donde se almacena el cambio de variables.
- **complementariosin:** Almacena el diseño complementario sin las variables que se han usado para el cambio.
- **inversa:** Matriz que despeja las variables antiguas y las pone en función de las nuevas.
- **cambiados:** Almacena complementario con el cambio de variables.
- **cambiadoss:** Les añade las variables puras al cambio de variable.
- **N:**
- **WLP:** Almacena el Word Length Pattern del diseño complementario actual.
- **WLPcomplementario:** Una matriz que almacena todos los Word Lenght Pattern de diseños complementarios por si el programa se ejecuta en bucle.

2.3.3.2. Funciones auxiliares

- **for:** Bucle que repite una sentencia un número determinado de veces.
- **size(matriz, 1/2):** Devuelve el numero de filas de la matriz (1) o el número de columnas (2).
- **xor(vector1, vector2):** Realiza la operación binaria xor entre dos vectores.
- **if:** Si se cumple la condición a la derecha del if se ejecutan las sentencias que aparecen hasta el siguiente end.
- **setdiff(matriz1,matriz2,'rows'):** Selecciona las filas de matriz1 que no están en matriz2.
- **gfrank(matriz,2):** Calcula el rango de la matriz en módulo dos.
- **inv(matriz):** Calula la inversa de una matriz.
- **round():** Redondea.
- **mod(matriz,2):** Convierte a módulo dos.
- **zeros(filas, columnas):** Crea matriz de ceros.

2.3.3.3. Explicación del código

1º PARTE: Se crea la matriz M que contiene H_{letras}

En este parte se crea la matriz que contiene el conjunto de valores que conforman un diseño factorial. En este caso se obtendría la matriz que contiene a H_4 = {A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD, ABCD} en números binarios.

2º PARTE: Se crea el diseño complementario

En esta franja del programa se crea el complementario de disenoentrada

disenoentrada =

1	*0*	*0*	*0*
0	*1*	*0*	*0*
0	*0*	*1*	*0*
0	*0*	*0*	*1*
1	*1*	*0*	*0*

Dimensión 5x4

complementario =

0	*0*	*1*	*1*
0	*1*	*0*	*1*
0	*1*	*1*	*0*
0	*1*	*1*	*1*
1	*0*	*0*	*1*
1	*0*	*1*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*0*
1	*1*	*1*	*1*

Dimensión 10x4

3º PARTE: Se convierte a letras

En esta parte se convierte el diseño complementario a código ASCII para después ser convertido a letras que es una de las salidas del programa.

complementarioletras =

CD
B D
BC
BCD
A D
A C
A CD
AB D
ABC
ABCD

4º PARTE: Se calcula el cambio de variable

El diseño complementario de j sería el siguiente $\bar{j}$ = {AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD, ABCD}. En estas condiciones no puede calcularse la ecuación generatriz, ergo tampoco el Word Length Pattern. Es por eso necesario realizar un cambio de variable.

En función de las reglas establecidas en el capítulo uno al trabajar con estos conjuntos, y dado que trabajamos en binario, es necesario imponer la condición de que la matriz de cambio de variable tenga rango 4 en módulo dos. Es decir que las 4 ecuaciones que conformen el cambio sean linealmente independientes.

cambio =

0	*0*	*1*	*1*
0	*1*	*0*	*1*
1	*0*	*0*	*1*
1	*0*	*1*	*1*

El cambio de variable sería

A'= CD
B'= BD
C'= AD
D'= ACD

5º PARTE: Quitar del complementario las palabras que se han transformado en las nuevas variables

Es decir se trata de eliminar $\bar{j}$ = {AC, ~~AD~~, BC, ~~BD~~, ~~CD~~, ABC, ABD, ~~ACD~~, BCD, ABCD}

6º PARTE: Se realiza la inversa

Consiste en despejar las viejas variables en función de las nuevas.

inversa =

1	*0*	*0*	*1*
1	*1*	*1*	*1*

El cambio de variable sería

A= A'D'
B= A'B'C'D'
C= C'D'

0 0 1 1 *D= A'C'D'*
1 0 1 1

7º PARTE: Se obtienen las nuevas palabras tras el cambio

Se multiplica las antiguas palabras por la inversa para obtener las nuevas palabras y poder calcular el Word Length Pattern.

8º PARTE: Añado las variables puras

Para tener el diseño de cambio de variable completo.

9º PARTE: Obtengo el WLP del diseño complementario

Se obtiene el Word Length Pattern del diseño complementario para poder compararlo con el diseño simétrico.

3. ANÁLISIS Y CONCLUSIONES DE LOS RESULTADOS

Una de las aplicaciones fundamentales de la Estadística es el diseño de experimentos. En un diseño experimental se manipulan deliberadamente una o más variables (factores), para medir el efecto que tienen en otra variable de interés (variable respuesta). En un diseño factorial, es necesario medir la variable respuesta para dos o más valores de cada factor.

En los estudios exploratorios una estrategia habitual es considerar solo dos niveles o valores en cada uno de los factores. A estos diseños experimentales se le denomina diseños factoriales a dos niveles. El número de observaciones que requiere cada diseño es una potencia de dos.

En muchas aplicaciones en la industria, el número de factores es elevado y la aplicación de diseños factoriales completos exige un número muy elevado de experimentos a realizar. Cuando los recursos son limitados, el número de factores incluidos grande, o simplemente se quiere reducir el coste experimental, se utilizan las fracciones de diseños factoriales.

Dado el número total de factores a estudiar y el número de experimentos a realizar, elegir el diseño óptimo para cada caso particular es muy complejo. Es por eso que el propósito inicial del trabajo era desarrollar un programa informático que dado el número de factores ofreciera todos los diseños posibles para el caso de 32 observaciones. Es decir, se quería obtener todos los diseños factoriales fraccionales a dos niveles de 32 observaciones (2^{k-p} = 32 observaciones) que tienen utilidad práctica.

El problema era de enorme complejidad por el número elevado de combinaciones a comparar y debía ser tratado computacionalmente.

Para resolverlo, se desarrollaron tres programas en Matlab. El primer programa, al que se le ha bautizado como *"CaminosNodo"*, ofrece los diseños *k+1* con su correspondiente *Word Length Pattern* a los que es posible llegar desde un diseño concreto con *k* factores. El segundo programa, llamado *"DiseñosN32"*, satisface los objetivos iniciales del trabajo ya que permite obtener todos los diseños posibles para diseños factoriales $n=2^{k-p}$ con 32 observaciones (además es fácilmente adaptable a 16, 64, etc) con diferente *Word Length Pattern*. Por último se ha creado, un tercer programa, *"Complementarios"*, que calcula los diseños complementarios de un diseño dado, con su correspondiente *Word Lenght Pattern*. El último programa tiene menor interés para el experimentador, ya que ha sido una herramienta de validación y verificación de los otros dos programas de acuerdo a la teoría.

Los resultados obtenidos han sido claramente satisfactorios a la par que interesantes. Se han obtenido 820 diseños con diferente *Word Length Pattern* en el rango de 32 observaciones. Además, se han encontrado fenómenos matemáticos inesperados, como la aparición de diseños con el mismo *Word Length Pattern* pero diferentes, algo que no se esperaba hasta valores de *k-p* superiores.

Para entender los resultados obtenidos hay que explicar qué caracteriza a un diseño, así como que hace que un diseño sea equivalente a otro.

Para explicarlo se va a seguir la notación de Box y Hunter (1961) en relación a los diseños factoriales fraccionados a dos niveles. Los número *1,2,…k* referidos a los factores representan letras que a su vez representan variables. A un producto de estas letras se le llama palabra. Para un diseño 2^{k-p} podemos escoger un conjunto (no único) de *p* palabras, las cuales generan el diseño.

Sea I la identidad, para todas las palabras *X*, $IX = XI = X$ y $X^2 = I$. Esto permite escribir el producto entre dos palabras *XW* de una forma reducida. El conjunto de palabras diferentes formadas por los todos los productos posibles incluyendo los que involucran a las *p* palabras generadoras, forman la *ecuación generatriz*, que contiene 2^p términos que si se incluye el término que hace referencia a la identidad *I* (a términos prácticos no suele incluirse). Las palabras de la *ecuación generatriz* corresponden a aquellos productos de las columnas del diseño que son iguales a la identidad. Un diseño es únicamente determinado por esta *ecuación generatriz* y viceversa.

Hasta aquí, puede decirse que es un resumen breve de lo que se desarrolló en el capítulo uno, sin embargo la última frase del párrafo anterior es de vital importancia para esclarecer la diferencia entre dos diseños diferentes valga la redundancia.

Como ya se explicó anteriormente, Burton y Connor (1957) desarrollaron una condición necesaria y suficiente para que dos diseños sean diferentes, el *Word Length Pattern*. Siempre que dos diseños tengan diferente *Word Length Pattern*, los dos diseños son necesariamente diferentes.

En un primer momento, se pensó que dos diseños con el mismo *Word Length Pattern* eran diseños equivalentes, y de hecho en un número mayoritario de casos es así. Sin embargo, Draper y Mitchel (1970) probaron que esto no tiene por que ocurrir de esta manera. Pongamos un ejemplo, que fue incluido por Jiahua Chen & Dennis K.J. Lin. en *On the identity relationships of* 2^{k-p} *designs*:

$$I=ABCFK=ABCHJL=FGHJKL=ADEFJM=BCDEGJKM=BCDEFHLM=ADEGHKLM$$

$$I=ACDFJK=ACEGHL=DEFGHJKL=ABCDEFHM=BEHJKM=BDEGLM=ABCGJKLM$$

Si se calcula el *Word Length Pattern* de ambos diseños puede observarse que coincide:

$$WLP = (0\ 0\ 0\ 0\ 0\ 4\ 0\ 3\ 0\ 0\ 0\ 0)$$

Nótese que las frecuencias con que aparecen las letras en las palabras de longitud 8 son diferentes. Debido a tales frecuencias sería imposible obtener un diseño mediante el otro con un cambio de variable, lo que hace que estos diseños sean diferentes. A este tipo de comprobación se le llamó *Letter Comparison* (comparación de letras).

En general para hacer una *Letter Comparison*, se examina la ecuación generatriz de un diseño para determinar el número, a_{ij}, de palabras de longitud *j* en las que la letra *i* aparece. Se procede a formar una matriz de dimensiones *k x k* $A = \{a_{ij}\}$ de cada diseño. Si se comparan dos diseños, aparecen entonces dos patrones de letras *A* y *A**. Para que sean equivalentes *A = P(A*)*, donde *P(A*)* es una permutación de las columnas de *A**.

Estos casos en los que a pesar de tener el mismo *Word Length Pattern* los diseños son diferentes no son habituales y ocurren en general, para un número n bastante elevado, siendo $n = 2^{k-p}$.

En el presente trabajo, el programa se desarrolló, descartando los diseños en base a su *Word Length Pattern* pues parecía muy poco probable que se dieran casos así en diseños factoriales fraccionales a dos niveles de 32 observaciones, además de carecer de utilidad práctica. Sorprendentemente para un número elevado de k, empiezan a aparecer diseños que cumplen estas condiciones.

¿Podría decirse entonces que se obtuvieron todos los diseños factoriales fraccionales a dos niveles de 32 observaciones? No, ha habido un número muy reducido de diseños, concretamente 9 de un total de 820, que no se calculan con el programa. ¿Tiene esto alguna repercusión sobre la utilidad práctica del programa? No.

Esta última afirmación enlaza con la aplicación práctica que se introdujo al principio de esta sección. Si se quiere reducir el coste experimental confundiendo una nueva variable con interacciones de otras variables, lo que interesa conocer son los diseños con diferente *Word Length Pattern* para un número *k* de factores dado, de manera que se pueda elegir el que más se adecúe al experimento.

En general, es conveniente utilizar diseños con máxima resolución y mínima aberración ya que con el mismo número de experimentos proporcionan una mayor cantidad de información útil. Que un diseño tenga máxima resolución supone que la nueva variable o factor se confundirá con una interacción de mayor orden, la cual tiene una probabilidad mucho más baja de ser significativa.

El diseño del programa no se cambió porque, la comprobación de que dos diseños sean o no equivalentes supone una cantidad muy superior de cálculos que una comparación de *Word Length Pattern*, lo que elevaría de sobremanera el tiempo de respuesta. Su aporte práctico, completamente nulo, no justificaba la pérdida de eficiencia del programa. Es por ello que se optó por mantener la estructura que se había desarrollado hasta el momento.

Según se ha desarrollado el programa el experimentador va a poder calcular para un número determinado de variables cual es el diseño de máxima resolución y mínima aberración. Además si deseara añadir una nueva variable, puede ejecutar fácilmente el programa y observar el número de diseños a los que puede acceder desde su experimento actual.

Por otro lado, si bien es cierto que la finalidad de este Trabajo de Fin de Grado era resultar de ayuda en los diseños de experimentos, los resultados del mismo tienen una utilidad no tan explícita como puede parecer, y es que el problema guarda una estrecha relación con diversas ramificaciones matemáticas.

Cada diseño, cada subconjunto de palabras, forma un grupo finito, de manera que diversas conexiones, dentro de la teoría de Galois y la matemática discreta pueden ser estudiadas. Pero esencialmente, hay que destacar que las fracciones de los diseños factoriales están estrechamente relacionadas con la codificación. Ambos son problemas combinatorios dentro de la Teoría de Números, un tema que apasiona a los matemáticos.

De manera sintética, los códigos detectores de errores trabajan para que en el envío datos codificados si se producen errores de transmisión, de manera que el paquete final sea distinto del inicial, estos puedan ser detectados y corregidos.

Hamming desarrolló la nomenclatura *(k,p)* para describir el sistema: en cada paquete de bits enviado, *k* será el número total de bits que conforma el paquete, *p* serán los bits que contienen los datos que quieren ser enviados, *k-p* será el número de bits de paridad. Los bits de paridad son los encargados de detectar los posibles errores que haya habido en la transmisión. Por último, también hay que definir el concepto de la *distancia de Hamming*. La efectividad de los códigos correctores de errores depende de la diferencia entre una palabra de código válida y otra. Cuanto mayor sea esta diferencia, menor es la posibilidad de que una palabra del código válido se transforme en otra palabra del código válido por una serie de errores. A esta diferencia se le llama distancia de Hamming, y se define como el número mínimo de bits que tienen que cambiarse para transformar una palabra de código válida en otra palabra de código válida. Si dos palabras de código están a una distancia **d**, se necesitan **d** errores para convertir una en la otra.

Se dice que un código $\boldsymbol{C}$ detecta t-errores si cualesquiera dos palabras $\mathbf{c_1}, \mathbf{c_2} \in \boldsymbol{C}$ que tienen una *distancia de Hamming* menor que *t* coinciden. Dicho de otro modo, un código detecta *t-errores* si y solo si la *distancia de Hamming* mínima entre cualesquiera dos palabras en él es al menos *t+1*.

El problema que se plantea por tanto es enviar la máxima información posible con el mínimo de bits de paridad. Esto puede formularse de la siguiente manera: dada una longitud de código *k*, requerida una distancia *d*, encontrar el número máximo de bits de datos *p*, para que la distancia sea máxima.

Es fascinante como este problema puede extrapolarse a las fracciones de diseños factoriales y es que la longitud del código *k* coincide con el número de factores de un diseño, el número de bits de datos *p* con el número de generadores y por último la distancia *d*, se trata del *Word Length Pattern.* Este tema aparece tratado con más profundidad en *Some bridges between codes and designs* de Peter J. Cameron.

Lo aprendido en el último párrafo supone que la resolución del problema de codificación pasa por encontrar el diseño de máxima resolución. Es por ello que los resultados del presente proyecto tienen cabida en el ámbito de la codificación entre otros campos matemáticos.

En definitiva, y a modo de conclusión final, cabe resaltar las siguientes características:

- Se han obtenido 820 diseños con diferente *Word Length Pattern* en el rango de 32 observaciones. Tienen una utilidad práctica muy alta en el diseño de experimentos.

- Se han encontrado fenómenos matemáticos inesperados, como la aparición de diseños con el mismo *Word Length Pattern* pero diferentes, algo que no se esperaba hasta valores de *k-p* superiores.

- A pesar de la enorme utilidad para el desarrollo de experimentos, los diseños factoriales 2^{k-p} están ligados a numerosos problemas combinatorios en el ámbito matemático, por lo que los resultados obtenidos en este trabajo pueden ser extrapolados a otras áreas matemáticas que no sean simplemente el diseño de experimentos. A modo de ejemplo, guardan una estrecha relación con los códigos de detección de errores, a los cuáles, los resultados de este proyecto pueden ser aplicados directamente.

4. LÍNEAS FUTURAS

Cualquier trabajo de investigación contribuye a despejar algunas incógnitas sobre el tema tratado, pero de forma simultánea genera nuevas preguntas, nuevas ideas y/o abre nuevas vías de trabajo. En este apartado se presentan algunas líneas futuras de investigación que pueden ser objeto de interés, atendiendo al trabajo expuesto en el presenta Trabajo de Fin de Grado.

La corriente más intuitiva parece ser explorar los diferentes diseños que aparecen con el mismo *Word Length Pattern*. Sería interesante averiguar si siguen alguna pauta de aparición y si se les podría encontrar alguna utilidad práctica. Reformular el código del programa para que encuentre todos los diseños parece una opción a seguir razonablemente natural.

El código del programa puede ser utilizado para la obtención de diseños factoriales 2^{k-p} con 64, 128… observaciones. Ahondar en la utilidad de estos resultados sería redundante puesto que ha debido quedar clara a lo largo de la lectura del trabajo. Así pues, seguir explorando los diseños factoriales fraccionados 2^{k-p} para mayor número de observaciones se trata de un camino a seguir de notable interés tanto práctico como teórico.

Por último, pienso que sería especialmente aconsejable seguir investigando como pueden relacionarse los diseños factoriales fraccionados con multitud de problemas combinatorios. Encontrar relaciones con nuevos problemas de la Teoría de Números o profundizar en los ya existentes tiene un interés innegable.

5. SOSTENIBILIDAD, PRESUPUESTO Y PLANIFICACION

5.1. SOSTENIBILIDAD

La sensibilización y concienciación medioambiental existente hoy en día se extiende a todos los ámbitos, incluido al relativo a la realización de proyectos ingenieriles. Por ello, el presente trabajo también avala esta responsabilidad medioambiental.

Este compromiso con el medioambiente puede no resultar explícito en la elaboración del trabajo, pues la temática no induce a esta preservación de una manera clara. Sin embargo, si ahondamos en lo que propone el trabajo, un abanico de posibilidades para elegir modelos estadísticos que más se adecúen a nuestras necesidades en el desarrollo de un experimento, es fácil discernir que puede suponer una verdadera ayuda para la conservación del medioambiente.

En el caso más llamativo y a la vez intuitivo que podemos elegir, un experimento químico, parece sencillo atisbar como contar con modelos estadísticos que respondan a unas pautas que se requieran en un experimento concreto puede ser beneficioso para el medioambiente. La disminución de replicaciones, cómo ejemplo más obvio, puede suponer la emisión de muchas menos sustancias contaminantes al medio. El uso de menor número de materias primas, también supone una enorme disminución del impacto ambiental.

Con todo ello, este trabajo intenta reducir al máximo los daños causados al entorno, garantizando un compromiso de respeto por el medioambiente.

5.2. PRESUPUESTO

Los costes asociados al presente trabajo pertenecen fundamentalmente a tres categorías:

- Licencias de programas.
- Amortización del equipo utilizado.
- Recursos Humanos (Autor y tutores)

Se puede observar una descomposición más detallada a continuación:

CONCEPTO	COSTE UNITARIO	UNIDADES (Ud.)	COSTE (€)
Licencia de Matlab Estudiante	150 €/unidad	1 unidad	150
Licencia de Microsoft Office	69 €/unidad	1 unidad	69
Amortización ordenador portátil personal Sony Vaio	31.563 €/unidad	11 meses	347,19
Ingeniero Junior autor del proyecto	30 €/hora	520 horas	15600
Profesor 1 Encargado del proyecto	60 €/hora	90 horas	5400
Profesor 2 Encargado del proyecto	60 €/hora	10 horas	600
SUBTOTAL	**22.166,19**		
Costes indirectos (IVA)	21%		
TOTAL	**26.821,42**		

Figura 5. 1. Presupuesto.

5.3. PLANIFICACIÓN TEMPORAL

En este apartado, se va a presentar en primer lugar una breve descripción de las tareas que se han llevado a cabo, a modo de Estructura de Descomposición del Proyecto o EDP), y posteriormente un diagrama Gantt para poder enmarcarlas cronológicamente.

5.3.1 Detalle de las tareas

1. Estudio de la teoría de diseños factoriales fraccionales a dos niveles.
2. Familiarización con la programación en Matlab.
3. Desarrollo de un pequeño programa que calcula el Word Length Pattern de un diseño.
4. Exploración y construcción manual de los primeros diseños 2^{k-p} para 32 observaciones.
5. Construcción de un programa que calcula los diseños a los que se puede llegar desde un diseño actual añadiendo una nueva variable al experimento.
6. Construcción de un programa que calcula todos los diseños posibles para diseños 2^{k-p} con 32 observaciones.
7. Construcción de un programa que calcula los diseños complementarios para verificar y validar los resultados obtenidos en el programa anterior.
8. Interpretación de los resultados.
9. Redacción del Trabajo de Fin de Grado.

5.3.2. Diagrama de Gantt

En este apartado se presenta la planificación seguida para realizar el trabajo.

En primer lugar, hay que notificar que el presente Trabajo de Fin de Grado es complejo y ha requerido 520 horas frente a las 360 horas que se estipulan en los 12 créditos que le corresponden.

El autor inició su desarrollo en Marzo de 2016 pero debido a la cantidad de asignaturas no fue capaz de completarlo para la fecha señalada en Julio. Finalmente en el año 2017 ha concluido su desarrollo. Por tanto las fechas empleadas en la ejecución del presente Trabajo Fin de Grado han sido:

- ☐ Fecha de inicio: 11 de Marzo de 2016.
- ☐ Fecha de finalización: 6 de Febrero de 2017.

Para llevar a cabo la planificación, se ha realizado una división en tareas, para la cual se ha tenido en cuenta las actividades fundamentales necesarias para el desarrollo del trabajo.

Además en cada tarea mencionada se incluye también el tiempo empleado en la redacción ya que se realizó simultáneamente.

Nombre	Fecha de inicio	Fecha de fin
1.Tareas de Estudio	11/03/16	15/04/16
1.1. Estudio de la teoría de diseños factoriales fraccionales a dos niveles	11/03/16	31/03/16
1.2. Familiarización con la programación en Matlab	1/04/16	15/04/16
2. Comienzo de la búsqueda de diseños	18/04/16	11/10/16
2.1. Desarrollo del programa que calcula el Word Length Pattern de un diseño	18/04/16	11/05/16
2.2. Exploración y construcción manual de los primeros diseños 2^(k-p) para 32 observaciones	16/09/16	11/10/16
3. Programación	13/10/16	10/01/17
3.1. Construcción del programa "CaminosNodo"	13/10/16	29/11/16
3.2. Construcción del programa "DiseñosN32"	1/12/16	30/12/16
3.3. Construcción del programa "Complementarios"	2/01/17	10/01/17
4. Fase Final	11/01/17	3/02/17
4.1. Interpretación de los resultados	11/01/17	21/01/17
4.2. Redacción del TFG	11/01/17	3/02/17

Figura 5. 2. Relación de tareas del Diagrama de Gantt.

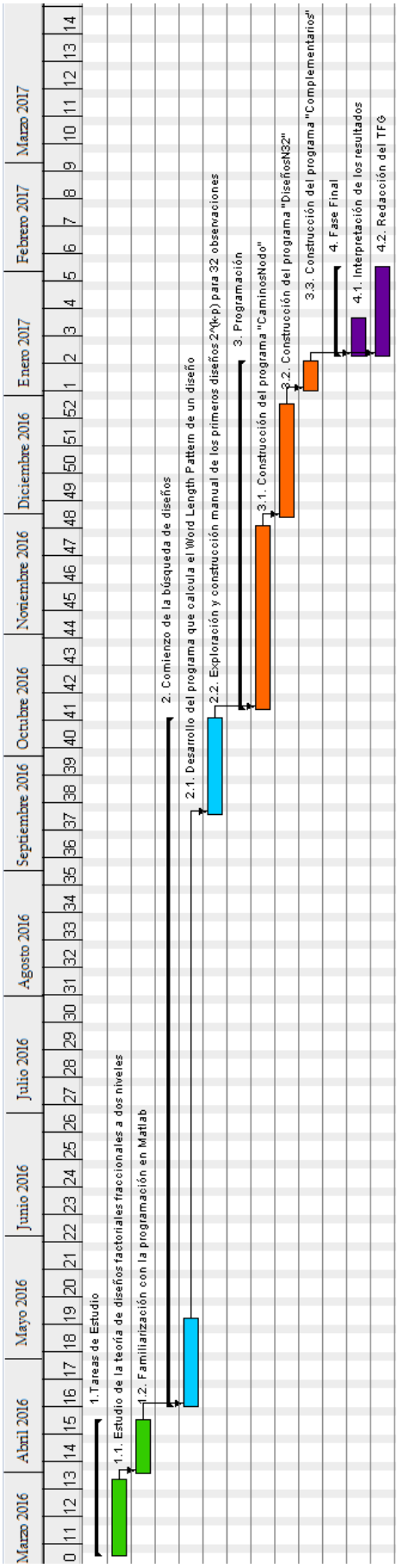


Figura 5. 3. Diagrama de Gantt

6. BIBLIOGRAFÍA

Daniel Peña. *Regresión y diseño de experimentos*. Ciencias Sociales. Alianza Editorial.

G. E. P. Box and J. S. Hunter. *The 2^{k-p} Fractional Factorial Designs Part I*. American Statistical Association and American Society for Quality. Technometrics. Vol. 3, No. 3 (Aug., 1961), pp. 311-351.

G. E. P. Box and J. S. Hunter. The *2^{k-p} Fractional Factorial Designs Part II.* Taylor & Francis Ltd. on behalf of American Statistical Association and American Society for Quality. Technometrics. Vol. 3, No. 4 (Nov., 1961), pp. 449-458.

Vaquero Santos, David (1999). *Método computacional para la obtención de fracciones factoriales a dos niveles.* Proyecto fin de carrera. Tutor: Jesús Juan Ruiz.

Fernández García, Francisco R. *Las matemáticas del mundo y el mundo de las matemáticas.*e-UMAB. ISBN: 84-8338-277-6.

R. C. Bose and D. K. RAY-Chaudhuri. *On A Class of Error Correcting Binary Group Codes.* University of North Carolina and Case Institute of Technology. INFORMATION AND CONTROL 3, 68-79 (1960).

Jiahua Chen & Dennis K.J. Lin. *On the identity relationships of 2^{k-p} designs*. Journal of Statistical Planning and Inference 28 (1991) 95-98.

Peter J. Cameron. *Some bridges between codes and designs.* School of Mathematical Sciences Queen Mary and Westfield College London E1 4NS, U.K.

7. ANEXOS

7.1. Programa “CaminosNodo”

```
%PROGRAMA CAMINOSNODO
%1 PARTE: INTRODUCES EL DISENO EN EL QUE TE ENCUENTRAS
 k=input('Ingrese el numero de factores:');
 p=input('Ingrese el numero de generadores:');
 disenoinicial=[];
 disenoinicial=zeros(p,k);
 for z=1:p
 name=input('Ingrese el vector generador en mayúsculas (Acuérdese de la letra extra): ', 's');
 name=double(name);
    for j=1:length(name)
     disenoinicial(z,name(j)-64)=1;
    end
 end
[x,y]=size(disenoinicial);
letras=5; %letras=k-p
R=disenoinicial(:,1:letras);
%2 PARTE: CREA LA MATRIZ DE PALABRAS CANDIDATAS A GENERAR UN NUEVO DISEÑO
M=[];
P=[];
P=eye(letras);
M(1,:)=P(1,:);
contador=2;
for i=2:letras
    M(2^(i-1),:)=P(i,:);
    contador=contador+1;
    for j=1:(2^(i-1)-1)
        M(contador,:)=xor(M(j,:),P(i,:));
        contador=contador+1;
    end
end
N=M;
%vamos a quitarle a la matriz M las letras A, B, C, D, E
M1=[];
contador=1;
limite=size(N);
for i=1:limite(1)
    if sum(M(i,:))~=1
       M1(contador,:)=M(i,:);
       contador=contador+1;
    end
end
M=M1; %a M le deja sin las letras en solitario
%QUITAMOS LAS PALABRAS QUE SE REPITEN
candidatas=[];
contador=1;
for i=1:size(M,1)
    for j=1:x
        a(j)=any(xor(M(i,:),R(j,:)));
    end
    if all(a)~=0
        candidatas(contador,:)=M(i,:);
        contador=contador+1;
    end
end
%3 PARTE: CONCATENA PARA QUE CANDIDATAS Y DISENOINCIAL TENGAN LA MISMA DIMENSIÓN
candidatasanadido(size(candidatas,1),size(disenoinicial,2)-letras)=0;
candidatasanadido2=ones(size(candidatas,1),1);
candidatascompleta=[candidatas candidatasanadido candidatasanadido2];
disenoinicialanadido(size(disenoinicial,1),1)=0;
disenoinicialcompleto=[disenoinicial disenoinicialanadido];
%4 PARTE: MULTIPLICO Y OBTENGO WLP DE CADA UNA
nuevaspalabras=[];
WLPnuevaspalabras=[];
numero=2;
for z=1:size(candidatascompleta,1)
    disenotestado=[disenoinicialcompleto; candidatascompleta(z,:)];
    tamano=size(disenotestado);
```

```
    %empezamos a hacernos la generatriz de este diseÑo
    G=[];
    WLP=zeros(1,tamano(2)); %la longitud del wlp
        G(1,:)=disenotestado(1,:);
        WLP(sum(G(1,:)))=WLP(sum(G(1,:)))+1;
        contador=2;

    for i=2:tamano(1)
        G(2^(i-1),:)=disenotestado(i,:);
        WLP(sum(G(2^(i-1),:)))=WLP(sum(G(2^(i-1),:)))+1;
        contador=contador+1;
        for j=1:(2^(i-1)-1)
            G(contador,:)=xor(G(j,:),disenotestado(i,:));
            WLP(sum(G(contador,:)))=WLP(sum(G(contador,:)))+1;
            contador=contador+1;
        end
    end
    if z==1
        WLPnuevaspalabras(1,:)=WLP;
        nuevaspalabras(1,:)=candidatascompleta(1,:);
    end
    tope=size(WLPnuevaspalabras);
    diferente=0;
    for x=1:tope(1)
        dif=isequal(WLP,WLPnuevaspalabras(x,:));
        if dif==0
            diferente=diferente+1;
        end
    end
    if (diferente==tope(1))&&(z~=1)
        nuevaspalabras(numero,:)=candidatascompleta(z,:);
        WLPnuevaspalabras(numero,:)=WLP;
        numero=numero+1;
    end
end
%5 PARTE:ORDENARLAS EN ORDEN DE PEOR A MEJOR(EN EL GRAFICO ESTA AL REVES)
WLPnuevaspalabrasordenadas=[];
[WLPnuevaspalabrasordenadas,indices]=sortrows(WLPnuevaspalabras);
for colocacion=1:size(indices,1)
    nuevaspalabrasordenadas(colocacion,:)=nuevaspalabras(indices(colocacion),:);
end
%6 PARTE:CONVERTIR palabras[]A LETRAS Y MOSTRAR RESULTADOS POR PANTALLA
  for i=1:size(WLPnuevaspalabrasordenadas,1)
      for j=1:size(WLPnuevaspalabrasordenadas,2)
          if nuevaspalabrasordenadas(i,j)==1
              nuevaspalabrasordenadas(i,j)=64+j;
          end
      end
   end
  disp('Los diseños nuevos son añadiendo una de estas palabras:')
  char(nuevaspalabrasordenadas)
  WLPnuevaspalabrasordenadas
```

7.2. Programa “DisenosN32”

```
function [disenosfinalesletras,diseno,WLPdiseno] = disenoN32(factores)
%1 PARTE: SE GENERA EL PRIMER DISENO
diseno=[];
WLPdiseno=[];
diseno(:,:,1,6)=[1 1 0 0 0 1];
diseno(:,:,2,6)=[1 1 1 0 0 1];
diseno(:,:,3,6)=[1 1 1 1 0 1];
diseno(:,:,4,6)=[1 1 1 1 1 1];
disenoanadido=[];
numeros=4;
for q=7:factores
[x,y,z]=size(diseno(:,:,1:numeros-1,q-1));
letras=5;
contadorglobal=0;
disenosrepetidos=[];
WLPdisenosrepetidos=[];
disenostotales=[];
disenostotalesordenados=[];
for w=1:z
clear palabras
R=diseno(:,1:letras,w,q-1);
%2 PARTE: CREA LA MATRIZ DE PALABRAS A MULTIPLICAR
M=[];
N=[];
P=eye(letras);
M(1,:)=P(1,:);
contador=2;
for i=2:letras
    M(2^(i-1),:)=P(i,:);
    contador=contador+1;
    for j=1:(2^(i-1)-1)
        M(contador,:)=xor(M(j,:),P(i,:));
        contador=contador+1;
    end
end
N=M;
%vamos a quitarle a la matriz M las letras A, B, C, D, E
M1=[];
contador=1;
for i=1:size(N,1)
    if sum(M(i,:))~=1
       M1(contador,:)=M(i,:);
       contador=contador+1;
    end
end
M=M1;
%3 PARTE: QUITAMOS LAS PALABRAS QUE SE REPITEN
candidatas=[];
contador=1;
for i=1:size(M,1)
    for j=1:x
        a(j)=any(xor(M(i,:),R(j,:)));
    end
    if all(a)~=0
        candidatas(contador,:)=M(i,:);
        contador=contador+1;
    end
end
%4 PARTE: AJUSTA LAS DIMENSIONES DE LAS MATRICES
columnas=size(diseno(:,:,:,q-1));
candidatasanadido=zeros(size(candidatas,1),columnas(2)-letras);
candidatasanadido2=ones(size(candidatas,1),1);
candidatascompleta=[candidatas candidatasanadido candidatasanadido2];
disenoanadido(columnas(1),1)=0;
disenocompleto=[diseno(:,:,w,q-1) disenoanadido];
```

```
%5 PARTE: MULTIPLICO Y OBTENGO WLP DE CADA UNA
nuevaspalabras=[];
WLPnuevaspalabras=[];
numero=2;
for z=1:size(candidatascompleta,1)
    disenotestado=[disenocompleto; candidatascompleta(z,:)];
    generatriz=[];
    WLP=zeros(1,size(disenotestado,2));
        generatriz(1,:)=disenotestado(1,:);
        WLP(sum(generatriz(1,:)))=WLP(sum(generatriz(1,:)))+1;
        contador=2;
    for i=2:size(disenotestado,1)
        generatriz(2^(i-1),:)=disenotestado(i,:);
        WLP(sum(generatriz(2^(i-1),:)))=WLP(sum(generatriz(2^(i-1),:)))+1;
        contador=contador+1;
        for jj=1:(2^(i-1)-1)
            generatriz(contador,:)=xor(generatriz(jj,:),disenotestado(i,:));
            WLP(sum(generatriz(contador,:)))=WLP(sum(generatriz(contador,:)))+1;
            contador=contador+1;
        end
    end
    if z==1
        WLPnuevaspalabras(1,:)=WLP;
        nuevaspalabras(1,:)=candidatascompleta(1,:);
    end
    tope=size(WLPnuevaspalabras);
    diferente=0;
    for xx=1:tope(1)
        dif=isequal(WLP,WLPnuevaspalabras(xx,:));
        if dif==0
            diferente=diferente+1;
        end
    end
    if (diferente==tope(1))&&(z~=1)
        nuevaspalabras(numero,:)=candidatascompleta(z,:);
        WLPnuevaspalabras(numero,:)=WLP;
        numero=numero+1;
    end
end
for i=1:size(nuevaspalabras,1)
    disenosrepetidos(:,:,contadorglobal+i)=[disenocompleto(:,:); nuevaspalabras(i,:)];
    WLPdisenosrepetidos(contadorglobal+i,:)=WLPnuevaspalabras(i,:);
end
contadorglobal=contadorglobal+size(nuevaspalabras,1);
end
%7 PARTE QUITAR LOS DISENOS QUE SE REPITEN
WLPtotales=[];
WLPtotales(1,:)=WLPdisenosrepetidos(1,:);
disenostotales(:,:,1)=disenosrepetidos(:,:,1);
numeros=2;
for v=2:size(WLPdisenosrepetidos,1)
    topes=size(WLPtotales);
    diferentes=0;
    for k=1:topes(1)
        diff=isequal(WLPdisenosrepetidos(v,:),WLPtotales(k,:));
        if diff==0
            diferentes=diferentes+1;
        end
    end
    if  diferentes==topes(1)
        WLPtotales(numeros,:)=WLPdisenosrepetidos(v,:);
        disenostotales(:,:,numeros)=disenosrepetidos(:,:,v);
        numeros=numeros+1;
    end
end
```

```
%8 PARTE: ORDENAR POR CRITERIOS DE OPTIMALIDAD
WLPtotalesordenados=[];
[WLPtotalesordenados,indices]=sortrows(WLPtotales);
for colocacion=1:size(indices,1)
    disenostotalesordenados(:,:,colocacion)=disenostotales(:,:,indices(colocacion));
end
%9 PARTE:CONVERTIR A LETRAS
disenosparaletras=[];
disenosparaletras=disenostotalesordenados;
for s=1:size(disenosparaletras,3)
    for i=1:size(disenosparaletras,1)
        for j=1:size(disenosparaletras,2)
            if disenosparaletras(i,j,s)==1
                disenosparaletras(i,j,s)=64+j;
            end
        end
    end
end
disenosletras=char(disenosparaletras);
%10 PARTE: GUARDA LOS DISENOS
[x,y,z]=size(disenostotalesordenados);
diseno(1:x,1:y,1:z,q)=disenostotalesordenados;
[x,y,z]=size(disenosletras);
disenosfinalesletras(1:x,1:y,1:z,q)=disenosletras;
[x,y]=size(WLPtotalesordenados);
WLPdiseno(1:x,1:y,q)=WLPtotalesordenados;
end
end
```

7.3. Programa “Complementarios”

```
function[WLPcomplementario, complementarioletras]=Complementarios(limite,letras,disenoentrada)
%PROGRAMA COMPLEMENTARIOS 32
%1 PARTE: CREA M
for t=1:limite
M=[];
P=[];
P=eye(letras);
M(1,:)=P(1,:);
contador=2;
for i=2:letras
    M(2^(i-1),:)=P(i,:);
    contador=contador+1;
    for j=1:(2^(i-1)-1)
        M(contador,:)=xor(M(j,:),P(i,:));
        contador=contador+1;
    end
end
%2 PARTE: TENEMOS NUESTRO DISENO, HALLAMOS EL COMPLEMENTARIO
complementario=[];
complementario=setdiff(M,disenoentrada,'rows');
%3 PARTE: convertir complementario a letras
complementarioletras=complementario;
for i=1:size(complementarioletras,1)
   for j=1:letras
           if complementarioletras(i,j)==1
               complementarioletras(i,j)=64+j;
           end
   end
end
complementarioletras=char(complementarioletras);
%4 PARTE: CALCULA MATRIZ CAMBIO VARIABLE DE RANGO 4
cambio=[];
contador=[];
cambio(1,:)=complementario(1,:);
cambio(2,:)=complementario(2,:);
contador=3;
for i=3:size(complementario,1)
    if gfrank([cambio;complementario(i,:)],2)==contador
       cambio(contador,:)=complementario(i,:);
       contador=contador+1;
    end
end

%5 PARTE:QUITO DE COMPLEMENTARIO LAS PALABRAS QUE USO PARA CAMBIO
complementariosin=[];
complementariosin=setdiff(complementario,cambio,'rows');
%6 PARTE: HAGO INVERSA DE CAMBIO PARA DESPEJAR LAS VARIABLES A,B,C,D...
inversa=[];
inversa=mod(round(inv(cambio)),2);
%7 PARTE: OBTENGO LAS NUEVAS PALABRAS
cambiados=[];
cambiados=mod(complementariosin*inversa,2);
%8 PARTE: ANADO EL EYE
cambiadoss=[];
cambiadoss=[cambiados eye(size(cambiados,1))];
%9 PARTE: OBTENGO WLP CON ESAS PALABRAS
N=[];
WLP=[];
WLP=zeros(1,size(complementario,1));
N(1,:)=cambiadoss(1,:);
WLP(sum(N(1,:)))=WLP(sum(N(1,:)))+1;
contadores=2;
for i=2:size(cambiadoss,1)
    N(2^(i-1),:)=cambiadoss(i,:);
    WLP(sum(N(2^(i-1),:)))=WLP(sum(N(2^(i-1),:)))+1;
    contadores=contadores+1;
    for j=1:(2^(i-1)-1)
        N(contadores,:)=xor(N(j,:),cambiadoss(i,:));
        WLP(sum(N(contadores,:)))=WLP(sum(N(contadores,:)))+1;
        contadores=contadores+1;
    end
end
WLPcomplementario(t,:)=WLP;
end
WLPcomplementario;
end
```

8. ÍNDICE DE TABLAS

9. ÍNDICE DE FIGURAS