Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Mecánica

PUESTA A PUNTO DE ANÁLISIS MULTIVARIANTE MEDIANTE MATLAB

Autor:

Pecharromán Vega, José Mª

Tutor:

Martín Pedrosa, Fernando

Ciencia de los Materiales e Ingeniería Metalúrgica

Valladolid, Julio 2016

RESUMEN

Se trata de hacer operativo el análisis multivariante mediante Matlab. Se implementan las técnicas de Análisis en Componentes Principales, las de Parafac y las de Tucker. Permite obtener resultados rápidamente y evaluar las implicaciones prácticas de manera sencilla. Se facilita, además, el empleo a usuarios no avanzados en estas técnicas. Se aplica todo el procedimiento a un caso práctico consistente en curvas de voltametría cíclica de una serie de vinos.

PALABRAS CLAVE

Análisis Multivariante; PCA; Parafac; Tucker; Matlab

Índice

1. Introducción

Este proyecto se ha realizado en el departamento de Ciencia de los Materiales e Ingeniería metalúrgica de la Escuela Técnica Superior de Ingenieros Industriales de la Universidad de Valladolid, donde desde hace años se trabaja en el desarrollo de sistemas olfativos y gustativos electrónicos (narices y lenguas electrónicas).

El objetivo final del grupo de trabajo es la creación de un instrumento de medida que, mediante una red de sensores y su correspondiente software de análisis, permita la caracterización organoléptica de vinos.

Debido a la complejidad del problema abordado, es necesario un grupo de colaboración entre químicos, físicos e ingenieros, de modo que se realicen los diseños y desarrollos en campos tan dispares como la síntesis de materiales, diseño de hardware, fabricación de sensores o creación de herramientas de análisis.

Inicialmente se desarrolla un novedoso grupo de sensores voltamétricos basados en pasta de carbono que conforman una *lengua electrónica*. Estos sensores se exponen a distintos vinos en el laboratorio registrando las correspondientes curvas voltamétricas.

Una vez obtenidos los resultados, se debe desarrollar un software que permita discriminar las señales permitiendo caracterizar los vinos por sus distintos sabores.

2. Objetivos

Situaremos el presente proyecto dentro del marco de análisis de resultados. El objetivo propuesto consiste en desarrollar una herramienta informática que permita el tratamiento de los datos obtenidos mediante los instrumentos de medida electroquímicos, de una forma lo más rápida y sencilla posible. El software desarrollado deberá ser capaz de distinguir los diferentes tipos de vinos analizados a partir las señales generadas por los sensores disponibles.

Las señales producidas por los sensores de líquidos son curvas que presentan una gran complejidad y además deberán ser analizadas en su totalidad por el software. Para cumplir con lo propuesto, se desarrollará un programa informático basado en MatLab que permita procesar los datos obtenidos en los ensayos realizados con la lengua electrónica, mediante técnicas de análisis multivariante y alcanzar resultado válidos y concluyentes.

Establecemos por tanto el objetivo del presente proyecto en el desarrollo de un software informático que sea capaz del análisis de datos mediante las técnicas multivariantes propuestas (Análisis en Componentes Principales, Parafac, Tucker3) y que se integre dentro del proyecto de diseño y fabricación de la lengua electrónica para la identificación de distintos vinos.

3. Análisis multivariante

Podemos definir análisis multivariante como el conjunto de métodos de análisis estadístico que tiene como finalidad determinar la contribución o influencia de varios factores o variables en un resultado. Para que un análisis se considere multivariante todas las variables deberán ser aleatoria y su influencia en el resultado no puede ser estudiada de forma individual.

Para un conjunto de datos dado, el objetivo de este tipo de análisis es obtener una combinación lineal de aquellas variables que aportan una mayor contribución a la explicación del resultado. Para ello se establecen métodos que mediante técnicas de predicción y búsqueda de patrones nos permiten buscar las relaciones causa-efecto.

Para que los métodos nos generen resultados válidos y concluyentes es necesario que los datos obtenidos de nuestros ensayos sean representativos del fenómeno a estudiar. Para ello será fundamental estudiar minuciosamente los variables a medir en los ensayos, la eliminación de los datos que no aporten información al resultado final y preparar los datos transformándolos y ordenándolos correctamente para el análisis.

Tras aplicar alguna de las técnicas de análisis multivariante a nuestro conjunto de datos veremos reducida su complejidad, sin haber perdido información relevante y además será posible su representación gráfica, lo cual nos facilitará la interpretación del fenómeno estudiado.

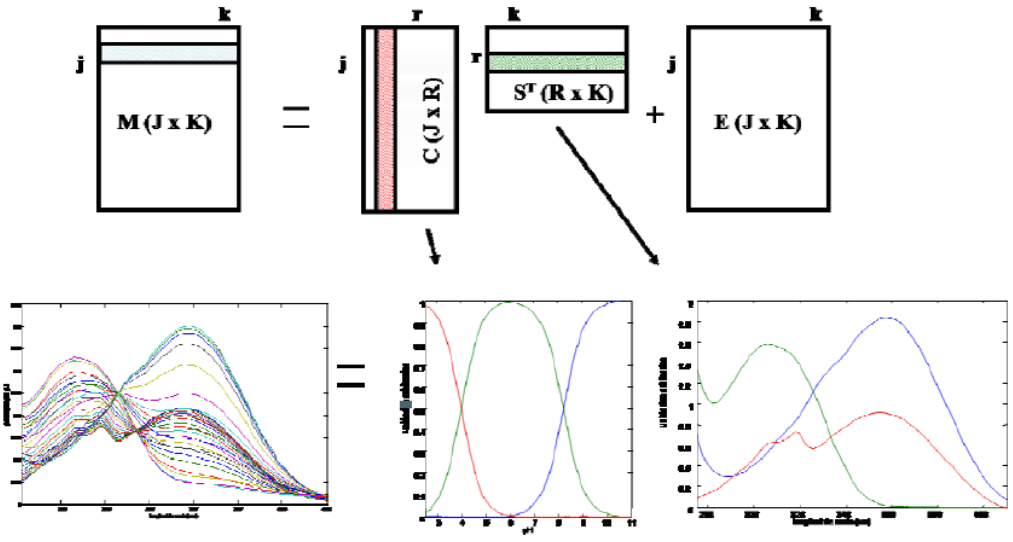


Figura 3.1 – Ejemplo de descomposición de un fenómeno complejo

La principal ventaja de este tipo de análisis es que tiene en cuenta el carácter multivariante de las variables de estudio, permitiendo conseguir información a la cual son incapaces de llegar los métodos univariantes y bivariantes. Esto convierte al análisis multivariante en una herramienta muy potente y de gran utilidad en campo de la investigación científica, especialmente en la quimiometría, donde se busca predecir las propiedades de muestras que no se pueden obtener de forma sencilla mediante medidas químicas directas pero sí estableciendo relaciones entre ellas a priori desconocidas.

Podemos clasificar los análisis multivariantes en dos tipos:

- No supervisados: Son técnicas en las que el resultado no se influencia y por tanto se obtiene un resultado más fiel a los datos originales.

- Supervisados: Son técnicas en las que se establece un criterio de discriminación de las muestras para que el resultado final se aproxime lo máximo posible a nuestro objetivo.

Debido a la gran cantidad de datos a analizar y su variedad se hace necesario dividir el proceso de los análisis en dos etapas:

- Pre-tratamiento: En esta etapa se buscan los datos que mayor información nos aportan, permitiéndonos reducir al máximo el número de variables. Además, se preparan y dan forma a los datos para poder proceder con el análisis.

- Tratamiento: En esta etapa se analizan los datos obtenidos en el pre-tratamiento mediante el uso de las técnicas multivariantes.

4. Métodos

En este capítulo trataremos de exponer los métodos matemáticos utilizados para realizar nuestro análisis multivariante, consistente este en un análisis en componentes principales (PCA) y dos análisis de tipo N-Way (Parafac y Tucker).

4.1. Pre-procesamiento

Como se ha comentado, el análisis multivariante se aplica en situaciones en las que disponemos de gran cantidad de datos y cuya influencia en el resultado es compleja de hallar. Debido a la dimensión y complejidad de los valores a analizar mediante estos métodos, resulta necesario eliminar aquello no aporta información al resultado final pero que lo puede influenciar. Además, la diversidad de los valores nos lleva a una estandarización de los mismo con el fin de ser comparables entre ellos. Son estas las necesidades que dan lugar a las técnicas de pre-procesamiento.

Denominaremos pre-procesamiento al conjunto de procedimientos de reducción y técnicas matemáticas, que tienen como finalidad simplificar y dar el formato necesario a los datos originales obtenidos en los ensayos, para la correcta aplicación de los métodos de análisis multivariantes sin perder información útil en el proceso.

En el presente capítulo veremos las técnicas de pre-procesamiento implementadas en nuestra herramienta informática.

4.1.1 Kernel

Los kernels son funciones matemáticas de pretratamiento de datos que permiten comprimir toda la información contenida en las curvas de estudio a un número reducido de parámetros, sin perder las características dinámicas de la curva ni su respuesta global.

Mediante el uso de estas funciones podemos reducir las necesidades computacionales del método y ganar en rapidez de cálculo.

Definiremos N_k como el número de kernels utilizados para reducir una curva. N_k deberá ser un valor lo más pequeño posible, pero suficientemente grande como para que la curva quede totalmente representada. El número de parámetros de todas las curvas de estudio debe ser el mismo, por lo que N_k será el mismo parar todas ellas.

Una vez aplicada la compresión de datos mediante kernels tendremos cada curva R_j reducida a un vector de parámetros cuya dimensión será igual a N_k.

$$V_j = [v_{j1}, v_{j2}, \dots, v_{jN_k}]$$

Ecuación 4.1

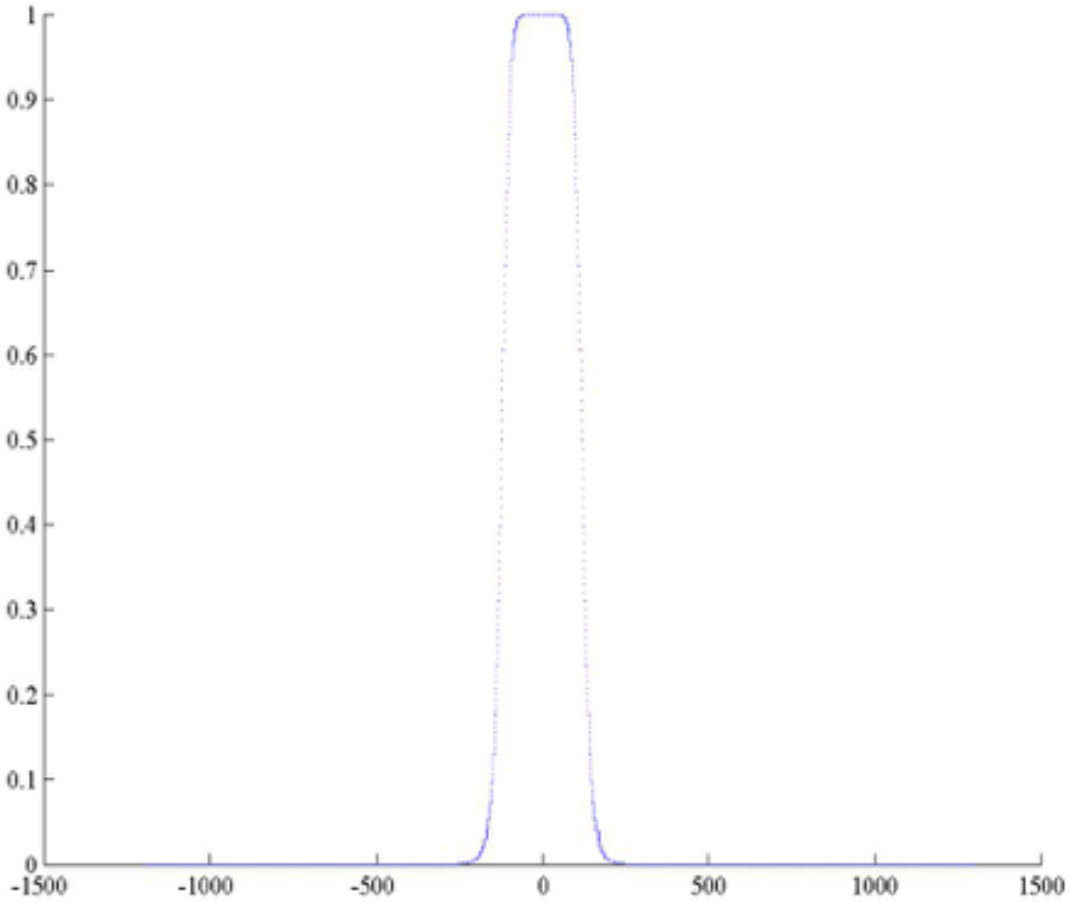


Figura 4.1 – Representación de una función Kernel

Una función Kernel K_i se define como:

$$K_i(R_j) = \frac{1}{1 - \left(\frac{R_j - c_i}{a_i}\right) * 2b_i}$$

Ecuación 4.2

Donde a_i define la anchura , b_i forma y c_i el centro de la función.

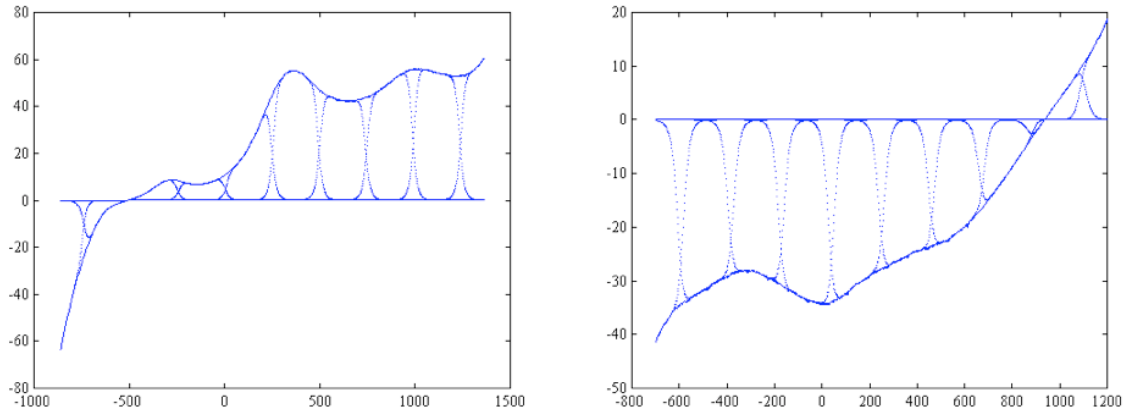


Figura 4.2 – Ejemplos de multiplicación de los Kernel y la curva estudiada

Cada elemento de un vector de parámetros v_{ji} es el resultado de multiplicar la función Kernel K_i por la curva de datos R_j correspondiente.

$$v_{ji} = K_i * R_j$$

Ecuación 4.3

Los vectores generados por estas funciones nos servirán de entrada para los métodos de análisis.

4.1.2 Centrado

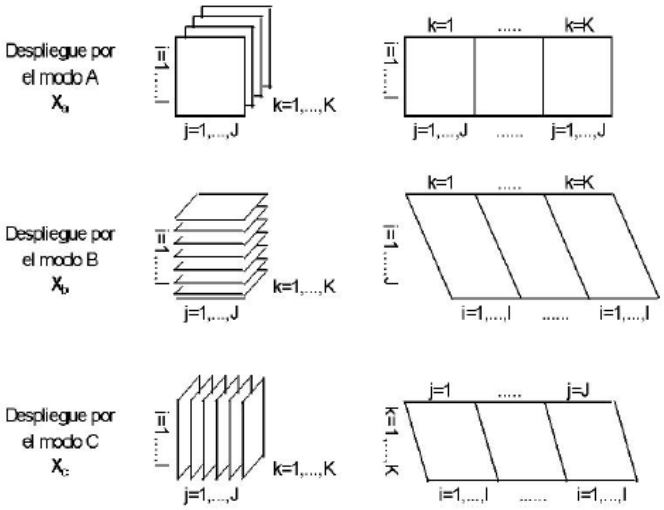


Figura 4.3 – Modos de despliegue de la matriz

El centrado se realiza a través de las columnas de la matriz. En primer lugar despliega la matriz por alguno de los 3 modos (Figura 4.3) obteniendo la nueva matriz X_m y posteriormente a cada elemento se le resta el promedio de la columna correspondiente.

$$x_{ijk}^{centrado} = x_{ijk} - \frac{\sum_{i=1}^{I} x_{ijk}}{I}$$

Ecuación 4.4

El centrado permite comparar datos cuyos ordenes de magnitud son distintos.

4.1.3 Escalado

Para realizar el escalado de la matriz se toma la matriz X_b y dividir cada elemento por la correspondiente desviación S_j.

$$x_{ijk}^{escalado} = \frac{x_{ijk}}{s_j}$$

Ecuación 4.5

donde:

$$S_j = \sqrt{\sum_{i=1}^{I}\sum_{k=1}^{K} x_{ijk}^2}$$

Ecuación 4.6

Cuando utilizamos distintos aparatos de medida, aun siendo iguales, los resultados devueltos en un mismo caso cambian de valor pero mantienen sus proporciones. El escalado permite homogeneizar los resultados de varias mediciones haciéndoles comparables entre sí de manera directa.

4.2. Análisis en Componentes Principales

El objetivo del análisis en componentes principales o PCA es transformar el espacio de representación de los datos P en un nuevo espacio P' en el que los datos estén incorrelados. Para ello buscaremos un nuevo conjunto de ejes ortogonales en el que la varianza de los datos sea máxima, reduciendo así la dimensionalidad del problema a sólo dos.

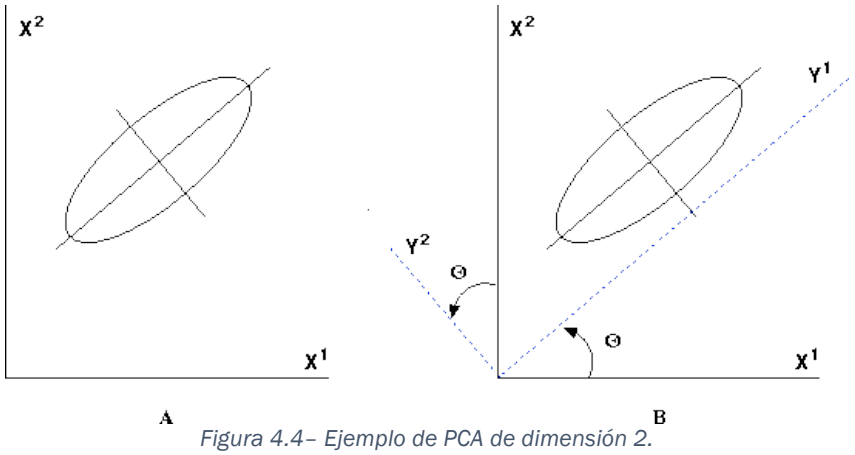


Figura 4.4– Ejemplo de PCA de dimensión 2.

La solución al método es una transformación lineal W tal que aplica a los datos originales P, X de lugar a P', Y. Con esto podemos definir la ecuación característica del análisis en componentes principales como:

$$Y = W^T X$$

Ecuación 4.7

Que escrito de manera explícita nos queda:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_d \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1d} \\ W_{21} & W_{22} & \dots & W_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ W_{d1} & W_{d2} & \cdots & W_{dd} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_d \end{bmatrix}$$

Ecuación 4.8

Para calcular W debemos establecer una serie de restricciones.

La primera de ellas es que los ejes del nuevo espacio P' deben ser ortogonales. Para esto se cumpla la matriz de transformación W debe ser ortogonal, cumpliéndose que:

$$W^{-1} = W^T$$

Ecuación 4.9

Por lo que se tiene que cumplir que:

$$W^T W = W W^T = I$$

Ecuación 4.10

La siguiente restricción que establecemos es que los datos de P' estén incorrelados. Esto supone que la matriz de covarianza del nuevo espacio P' debe ser diagonal.

Con lo anterior podemos formular el problema como la maximización de la varianza de P' con restricción de ortogonalidad de W.

$$\sum_Y = W^T \sum_X W$$

Ecuación 4.11

Para resolver la ecuación anterior haremos uso de los multiplicadores de Lagrange.

$$F = f\left(v_1, v_2, \dots, v_p\right) - \lambda g(v_1, v_2, \dots, v_p)$$

Ecuación 4.12

Donde $f(v_1, v_2,..., v_p)$ es la función a maximizar con la condición $g(v_1, v_2,..., v_p)=0$. Con esto nuestro problema consistirá en maximizar la nueva función *F* sin restricciones. Con esto, la ecuación a maximizar sería:

$$F = W^T \Sigma_X W - \lambda(W^T W - I)$$

Ecuación 4.13

Para obtener el máximo de *F* derivamos respecto *W* e igualamos a 0:

$$(\Sigma_X - \lambda I)W = 0$$

Ecuación 4.14

Para que la ecuación X.X.X sea cierta se tiene cumplir alguna de las siguientes condiciones:

- W = 0
- $|\Sigma_X - \lambda I| = 0$

Si W = 0 estaríamos ante una solución trivial, lo cual no nos interesa.

Si se cumple la ecuación $|\Sigma_X - \lambda I| = 0$ significa que $\Sigma_X - \lambda I$ es singular, esto es, que no es invertible. Esta ecuación es la ecuación característica de la matriz Σ_Xy depende de λ. Las soluciones (valores de λ) se denominan autovalores de Σ_X, o lo que es lo mismo, cada autovalor $\lambda_i$es la solución para una ecuación del sistema (ecuación X). Cada una de estas soluciones asociadas a W son un autovector Φ_i, con lo que podemos expresar la matriz de transformación W como un vector de *d* autovectores:

$$W = [\phi_1, \phi_2, \dots, \phi_d]$$

Ecuación 4.15

Como Σ_X es una matriz simétrica de orden *d* x *d* tendrá *d* autovalores reales asociados, que además, al definir positiva Σ_X estarán ordenados.

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$$

Ecuación 4.16

Con lo anterior podemos determinar que Σ_Y será una matriz diagonal formada por los autovalores de Σ_X.

$$\begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_d \end{bmatrix}$$

Ecuación 4.17

La matriz de transformación W será la matriz de autovectores de Σ_X, asumiendo que es ortogonal.

Como ya hemos visto, cada autovalor Σ_i tiene asociado un autovector Φ_X que define la dirección del eje en el espacio transformado P'. Ya que los autovalores están ordenados por el valor de la varianza de cada eje de P', podemos establecer el orden de la variables transformadas:

- Y_1: Primer eje de P' (primera componente principal). Indica la dirección en P de máxima varianza.
- Y_2: Segundo eje de P' (segunda componente principal). Indica la dirección en P de máxima varianza de entre todos los ejes ortogonales a Y_1.

Con todo lo anterior, podemos enunciar el *teorema fundamental del análisis en componentes principales*:

> *Dado un conjunto de variables* X^i *(* i = 1, 2,..., d*) con matriz de covarianza* $\mathbf{\Sigma_X}$*, no singular, siempre se puede derivar a partir de ellos un conjunto de variables incorreladas* Y^i *(* i = 1, 2,..., d*) mediante un conjunto de transformaciones lineales que corresponden a una rotación rígida cuya matriz de transformación* W *está formada, por columnas, por los* d *autovectores de* $\mathbf{\Sigma_X}$*. La matriz de covarianza del nuevo conjunto de variables,* $\mathbf{\Sigma_Y}$*, es diagonal, y contiene los autovalores de* $\mathbf{\Sigma_X}$.

Con todo lo anterior podemos plantear el algoritmo de cálculo del método de la siguiente manera:

- Cálculo de matriz de covarianza global Σ_X
- Cálculo de los autovalores de Σ_X
- Cálculo de los autovectores $\phi_1, \phi_2, \dots, \phi_d$ asociados a $\lambda_1, \lambda_2, \dots, \lambda_d$
- Formación de la matriz $W = [\phi_1, \phi_2, \dots, \phi_d]$
- Cálculo del resultado del cambio de espacio Y

El cálculo de Y se plantea a partir de la matriz de transformación W como:

$$Y = W^T X$$

Ecuación 4.18

Que de manera explícita sería:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_d \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1d} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{d1} & \phi_{d2} & \cdots & \phi_{dd} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_d \end{bmatrix}$$

Ecuación 4.19

4.3. Parafac

El modelo de Parafac fue propuesto por Harshman y Carroll de manera independiente en 1970. Destaca por por su simplicidad y posibilidades, siendo muy utilizado en en quimiometría, donde permite procesar grandes cantidades de datos obtenidos mediante instrumentos cada vez más complejos.

Parafac es un modelo de análisis que descompone la matriz tridimensional de estudio X en componentes trilineales (triadas), obteniendo como resultado tres sub-matrices bidimensionales A (ixj), B (jxk), C (kxi) (scores y dos loading).

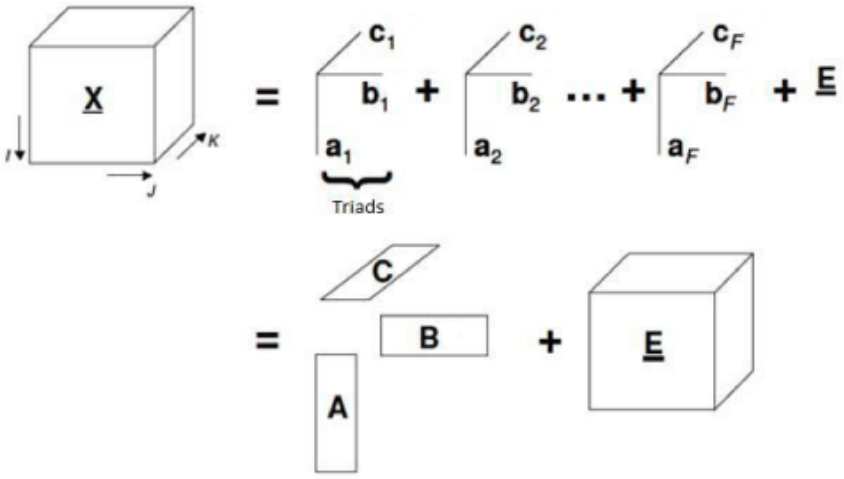


Figura 4.5

Podemos definir la ecuación característica del modelo como:

$$x_{ijk} = \sum_{f=1}^{F} a_{if}\, b_{jf} c_{kf} + e_{ijk}$$

Ecuación 4.20

Matemáticamente el objetivo será encontrar el modelo trilineal (A, B, C) que minimice la suma de los cuadrados de los residuales e_{ijk}.

La principal ventaja del método reside en que presenta solución única. Esto se consigue mediante la optimización por mínimos cuadrados alternados. Además, al ser un método supervisado, podemos imponer condiciones para que la solución sea más fácil de interpretar o se ajuste mejor a nuestro modelo.

Un punto crítico del modelo es la correcta determinación del número de componentes de nuestro modelo. El método más utilizado para decidir este parámetro es el análisis de la "*consistencia del núcleo*" o "*core consistency*".

El análisis de la "consistencia del núcleo" fue propuesto por Bro en 1998 y consiste en comparar los resultados obtenidos con la matriz original. Este parámetro resulta muy útil para discernir si la solución alcanzada es válida o si el modelo ha resultado poco fiable e inestable.

La solución al modelo de Parafac puede encontrarse mediante un proceso iterativo conocido como ALS (*Alternanting Least Squares*). En primer lugar se deben inicializar las matrices B y C. Realizar una buena estimación de estas componentes ayuda a que el algoritmo de cálculo encuentre el mínimo absoluto de convergencia. Las métodos más comunes de obtener los valores

iniciales son la Descomposición Trilinear o Métodos Generalizados de Anulación de Rango. Una vez realizada estimación, se procede con el algoritmo iterativo:

- Estimación de A a partir de B y C.
- Estimación de B y C de la misma manera.
- Aplicación del criterio de convergencia.

Este proceso debe repetirse hasta que se cumpla el criterio de convergencia. Es bastante común que el criterio establecido sea parar cuando el porcentaje de falta de ajuste (ecuación 4.21) en dos iteraciones no supere un cierto valor.

$$\%\ de\ falta\ de\ ajuste = 100 * \sqrt{\frac{\sum_{jk} e_{jk}^2}{\sum_{jk} m_{jk}^2}}$$

Ecuación 4.21

La estimación de A, B Y C se realiza mediante el método de mínimos cuadrados:

$$A = XZ_A{}^T(Z_A Z_A{}^T)^{-1}$$

Ecuación 4.22

$$B = XZ_B{}^T\left(Z_B Z_B{}^T\right)^{-1}$$

Ecuación 4.23

$$C = XZ_C{}^T(Z_C Z_C{}^T)^{-1}$$

Ecuación 4.24

donde:

$$Z_A = B \otimes C$$

Ecuación 4.25

$$Z_B = C \otimes A$$

Ecuación 4.26

$$Z_C = B \otimes A$$

Ecuación 4.27

En este punto debemos introducir un operador matemático conocido como "producto de Kronecker". Este se designa mediante el símbolo ⊗ y consiste en la multiplicación término a término de los elementos de la primera matriz por la segunda.

Dadas las matrices A (m x n) y B (p x q), el resultado del producto de Kronecker $A \otimes B$ será una matriz bloque (mp x nq). Esto es:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

Ecuación 4.28

Si desarrollamos esta ecuación tendríamos:

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \ldots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \ldots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p2} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p2} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \ldots & \ldots & a_{mn}b_{12} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p2} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p2} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix}$$

Ecuación 4.29

Cabe destacar que para este operador no se cumple la propiedad conmutativa.

4.4. Tucker

El modelo Tucker, además de la manera de calcular los parámetros del modelo, fue propuesto por Ledyard R. Tucker en 1966. Desde entonces el algoritmo de cálculo a recibido múltiples mejoras.

Tucker es un método general para el análisis de matrices de datos de orden 3 o superior, pudiéndose considerar Parafac como un caso particular del mismo. Su generalidad le convierte en un método muy potente para la manipulación e interpretación de datos en diversas aplicaciones.

El modelo de Tucker nos permite la extracción de un número diferente factores para cada modo. En nuestro caso nos centraremos en el modo 3, también conocido como Tucker3. En esta variante el primer índice indica el número de especies de estudio, el segundo las variables medidas y el tercero las repeticiones de la medición.

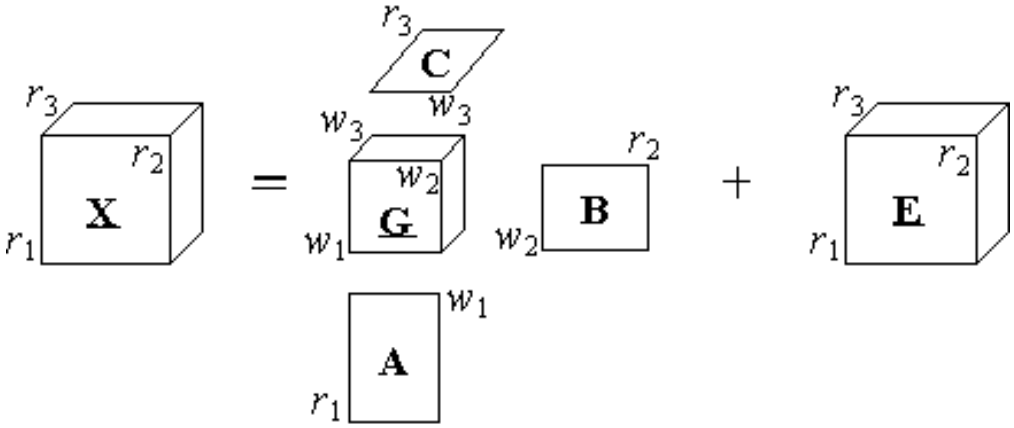


Figura 4.6

Observando la figura 4.6 se ve como la principal diferencia con el modelo de Parafac es la la existencia de matriz núcleo G (w1, w2, w3).

Lo normal es que Tucker se limite para ser ortogonal, es decir, se trata de un modelo supervisado. Con ello se consigue un núcleo más fácil de interpretar y una mayor velocidad de cálculo.

$$x_{ijk} = \sum_{l=1}^{w_1} \sum_{m=1}^{w_2} \sum_{n=1}^{w_3} a_{il} b_{jm} c_{kn} g_{lmn} + e_{ijk}$$

Ecuación 4.30

En la Ecuación 4.30 se puede ver como el modelo Tucker3 se define de forma muy similar al de Parafac. Esta ecuación se puede escribir de forma matricial para cualquiera de los tres modos:

$$X_a = AG_a(C^T \otimes B^T) + E_a$$

Ecuación 4.31

$$X_b = BG_b(C^T \otimes A^T) + E_b$$

Ecuación 4.32

$$X_c = CG_c(B^T \otimes A^T) + E_c$$

Ecuación 4.33

Antes de iniciar el análisis de Tucker es común el uso de procedimientos de pre-procesamiento como el centrado y escaldo que, aplicados de forma conjunta, permiten la estandarización de los datos.

Al igual que en el método de Parafac, podemos aplicar el método iterativo ALS para resolver el modelo. En primer se deben inicializar la matrices B y C, siendo válidos los mismos métodos de estimación que Parafac. A continuación se comienza con el proceso iterativo:

- $A = svd(X_a(C \otimes B), P)$
- $B = svd(X_b(C \otimes A), Q)$
- $C = svd(X_c(B \otimes A), R)$
- Aplicar el criterio de convergencia.

Este proceso se debe repetir hasta cumplir el criterio de convergencia establecido. Es habitual que el criterio utilizado consista en que la diferencia de la suma de los cuadrados de los elementos del núcleo no sea superior en a 1E-10 en dos iteraciones consecutivas.

Los valores P, Q y R se corresponden con el número de componentes establecido para el modo 1, 2 y 3 respectivamente. Para conseguir resultados válidos, resulta crítico escoger correctamente el número de componentes para cada modo. El método más utilizado para determinar la dimensionalidad óptima es seleccionar como mejor combinación aquella que presenta menor variación entre su ajuste y el obtenido al considerar una componente adicional.

El porcentaje de ajuste se calcula como el cociente entre la suma de los cuadrados de los elementos del núcleo G y la de los elementos de la matriz original X.

$$\%\ ajuste = \left(\frac{\sum_{pqr} g_{pqr}^2}{\sum_{pqr} x_{pqr}^2}\right) * 100\%$$

Ecuación 4.34

En el proceso iterativo hemos hecho uso de dos herramientas matemáticas que conviene comentar.

En primer lugar el "producto de Kronecker", designado por el símbolo ⊗. Este ya ha sido comentado en el capítulo 4.3, donde puede encontrase más información.

En segundo lugar el operador *svd*:

$$[U, S, V] = svd(Z, P)$$

Ecuación 4.35

Aplicar un SVD consiste en realizar una descomposición en valores singulares de la matriz Z reteniendo las primeras P componentes. En la Ecuación 4.35 la matriz U son los primeros P vectores singulares, V los P últimos y S la matriz diagonal compuesta por la raíz cuadrada de los primeros P valores singulares. Con esto, el resultado se obtiene como:

$$Z = USV^T$$

Ecuación 4.36

En el método de Tucker, el núcleo G contiene el peso que tiene cada triada posible en el resultado final. Con el fin de que los valores de G sean comparables entre distintos ensayos, se aplica un proceso de escalado de los factores de tal manera que la suma de de los cuadrados de todos los elementos sea igual a 1. Los elementos al cuadrado de mayor valor indican los factores más influyentes.

Como ya sabemos, el modelo Tucker3 no proporciona soluciones únicas. Es posible encontrar infinidad de soluciones diferentes A, B, C y G con el mismo nivel de ajuste para una matriz de datos X dada y utilizando el mismo número

de componentes para cada modo. Sin embargo, esto no repercute en la interpretación de los resultados y por tanto el comportamiento capturado por una de las soluciones es el mismo que para todas ellas. Se puede demostrar que cualquier solución del modelo Tucker3 se puede rotar utilizando matrices de rotación ortonormales arbitrarias sin afectar a la solución del análisis de X.

Principalmente existen tres métodos para la rotación del núcleo:

- Diagonalizar los planos.
- Diagonalizar la totalidad del núcleo.
- Rotación contralada para que la suma de los cuadrados de los elementos en la diagonal sea máxima.

Hay que indicar que, debido a que trabajamos con las diagonales de la matriz, para poder rotar el núcleo este debe ser una matriz cuadrada, es decir, debemos tener el mismo número de componentes para los 3 modos ya que si no la diagonal no está definida.

El propósito de la rotación del núcleo es la simplificación del mismo, haciendo así más fácil su interpretación.

5. Software desarrollado y puesta a punto

En este capítulo se expondrá todo lo relacionado con el software desarrollado y su puesta a punto, el cual es el objeto principal del presente trabajo, y que pretende ser una herramienta informática rápida y sencilla de análisis multivariante.

5.1. MatLab

MatLab® es una herramienta de software matemático multiplataforma desarrollada por la compañía MathWorks, muy extendida en universidades y centros de investigación y desarrollo. Permite el desarrollo de aplicaciones con un lenguaje propio (lenguaje M) que es interpretado en un entorno interactivo o mediante scripts (archivos *.m).

Aunque la interfaz que nos ofrece MatLab no es la más estética y nos limita a trabajar con la ventana de comandos, se escoge esta plataforma debido a sus capacidades matemáticas.

Todo el software de este proyecto se ha desarrollado mediante scripts para MatLab y es compatible con la versión R2014a y posteriores.

5.2. Funciones N-Way Toolbox

Para el desarrollo del software se utilizan las funciones nativas de MatLab, funciones propias y las funciones de "N-Way Toolbox". La biblioteca "N-Wat Toolbox" está disponible de forma gratuita en http://www.models.kvl.dk/source/ y se compone por una colección de funciones relacionadas con los métodos multivariantes que complementas a las funciones propias de MatLab, lo cual nos resulta muy útil en el desarrollo de nuestro software. Esta biblioteca está desarrollada por los profesores Claus A. Andersson y Rasmus Bro dentro del departamento de "Ciencias de los Alimentos" de la "Real Universidad de Agricultura y Ganadería" de Frederiksberg (Dinamarca).

Para compatibilizar las funciones de esta biblioteca con nuestro programa, y así poder integrarlas en nuestros algoritmos de cálculo, se han tenido que revisar y modificar algunas partes de su código fuente. Además se desarrollan funciones que modifican los datos de nuestros ensayos para darles el formato necesario que permite procesarlos con las funciones de "N-Way Toolbox".

5.3. Instalación

El presente documento se acompaña de un paquete de archivos digitales contenidos en una carpeta de nombre TFG.

Para poder ejecutar el programa deberemos copiar la carpeta TFG en el directorio específico de MatLab del ordenador donde se vaya a ejecutar el software.

Podemos ver cuál es dicho directorio en la parte superior de la ventana de MatLab.

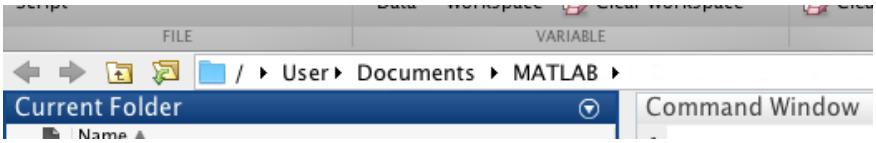


Figura 5.1

5.4. Inicialización

Una vez iniciado MatLab, en la parte izquierda de la pantalla, observaremos el contenido de nuestra carpeta de trabajo. Hacemos un click con el botón derecho en la carpeta TFG que añadida anteriormente (véase el capítulo 5.2.). En el menú desplegado seleccionamos la opción '*Add to Path, Selected Folder and Subfolders*' (figura 5.2).

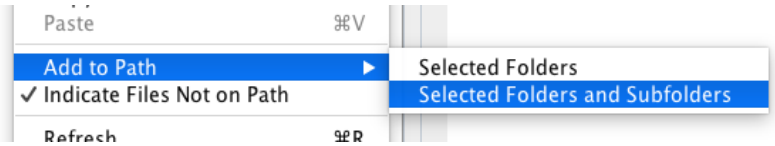


Figura 5.2

A continuación introducimos el comando '*init*' en la ventana de comandos (figura 5.3).

Figura 5.3

5.5. Selección del proyecto

Tras completar el capítulo 5.4 nos aparecerá un menú que nos permite seleccionar el proyecto con el que vamos a trabajar. Podemos escoger entre crear un nuevo proyecto (capítulo 5.5.1) o abrir uno que hayamos creado anteriormente (capítulo 5.5.2).

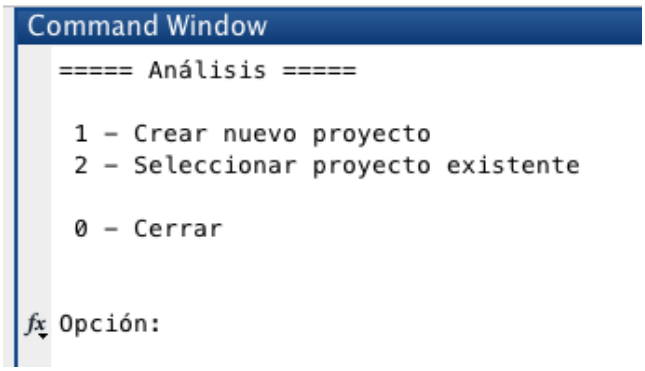


Figura 5.4

Escogiendo la opción '0' cerraremos el programa.

5.5.1. Creación de un nuevo proyecto

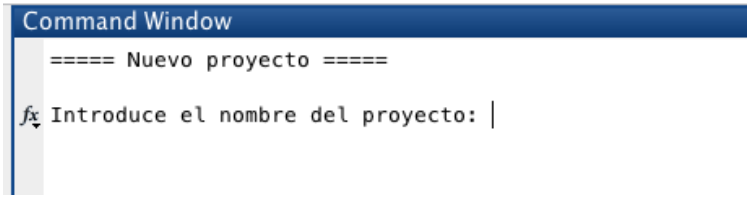


Figura 5.5

Cuando se selecciona la opción de crear un nuevo proyecto, lo primero que se debe hacer es introducir el nombre que le queremos dar. Se recomienda no utilizar espacios ni caracteres especiales para evitar problemas con archivos generados en el proceso de cálculo.

A continuación se nos aparecerá una ventana en la que deberemos seleccionar el archivo que contiene los datos que van a ser analizados. En el capítulo 5.6. se encuentra toda la información referente al formato que dicho archivo debe tener.

5.5.2. Carga de un proyecto existente

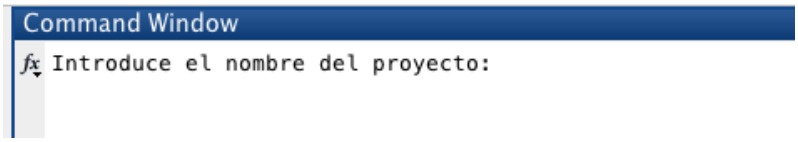


Figura 5.6

Para cargar un proyecto existente únicamente tendremos que introducir el nombre que le dimos al crearlo (figura 5.6). Con esto cargaremos en el *Workspace* actual de MatLab todos los datos calculados anteriormente para ese proyecto (figura 57).

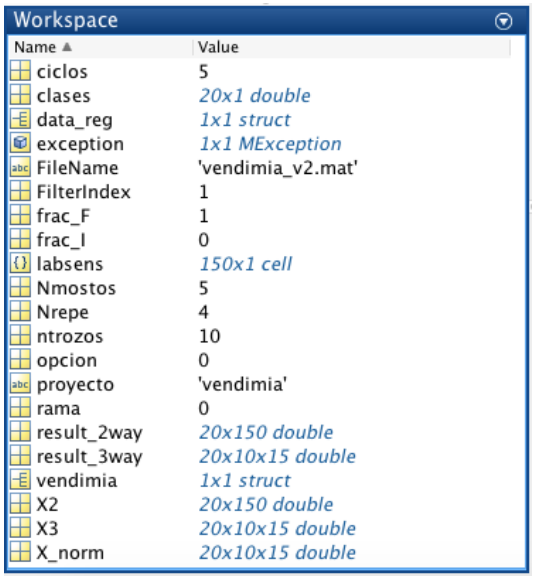


Figura 5.7

5.6. Datos

Los datos que queramos analizar deben guardarse en un documento '.mat', dentro del cual se encontrará una estructura que contenga todas las variables necesarias para el manejo de los datos. Distinguimos entre una variable índice y el resto, que serán las que contengan las curvas de estudio.

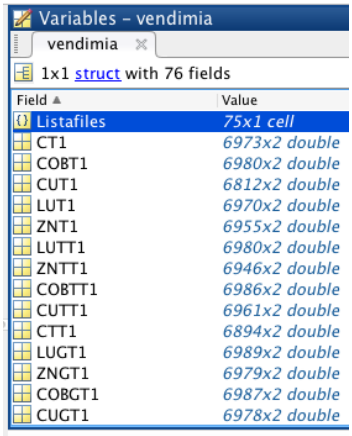


Figura 5.8

La variable índice, que debe llamarse '*Listafiles*', es un vector cuyos elementos serán los nombres del resto de variables que componen la estructura principal.

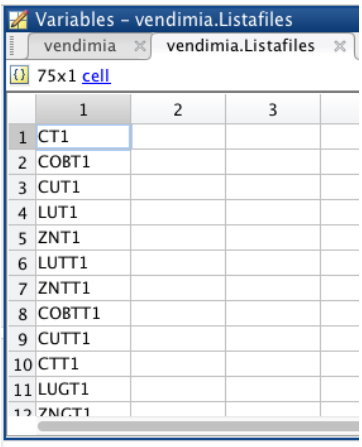
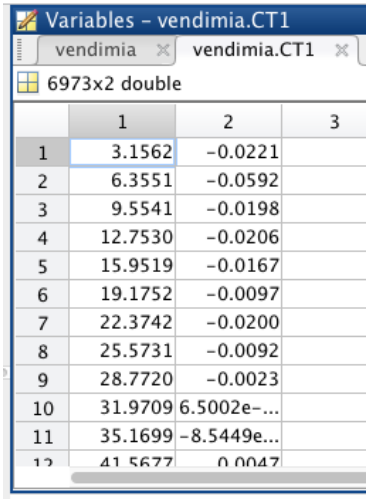


Figura 5.9

Cada una de las demás variables serán matrices de dimensión Nx2, donde N es el número puntos de la curva voltamétrica almacenada.



	1	2	3
1	3.1562	-0.0221	
2	6.3551	-0.0592	
3	9.5541	-0.0198	
4	12.7530	-0.0206	
5	15.9519	-0.0167	
6	19.1752	-0.0097	
7	22.3742	-0.0200	
8	25.5731	-0.0092	
9	28.7720	-0.0023	
10	31.9709	6.5002e-...	
11	35.1699	-8.5449e...	
12	41.5677	0.0047	

Figura 5.10

5.7. Proyecto

Después de crear un proyecto o cargar uno existente, se nos muestra un menú (figura 5.11) con todas opciones de configuración y cálculo disponibles.

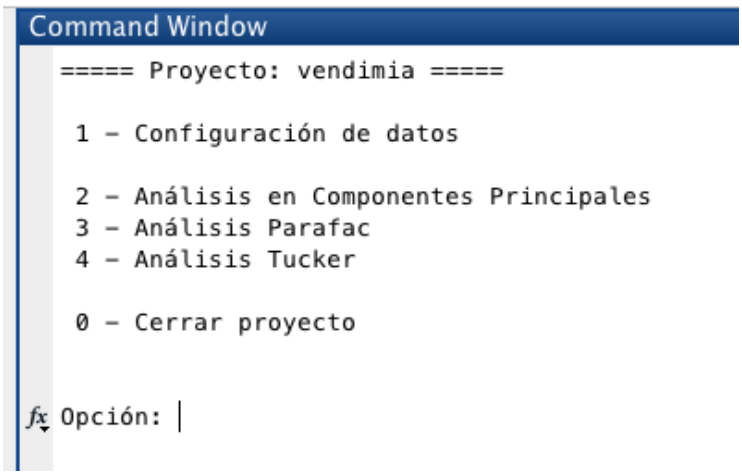


Figura 5.11

La opción de configuración de datos (capítulo 5.8) debe ejecutarse siempre después de crear un nuevo proyecto y opcionalmente si se desea cambiar alguno de los parámetros.

5.8. Configuración

Cuando accedamos a la configuración de datos (Figura 5.12) el programa nos solicitará los parámetros necesarios para posteriormente aplicar los métodos de análisis.

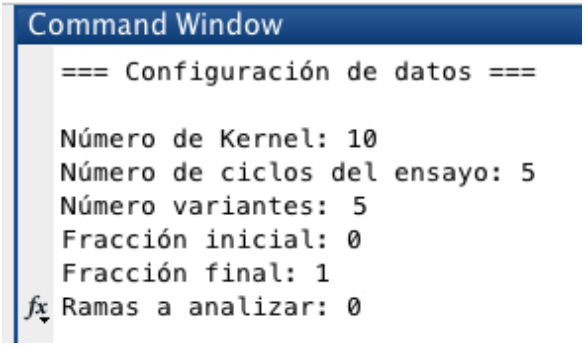


Figura 5.12

A continuación se describen los parámetros solicitados por el programa:

- Número de Kernel

Como ya se ha visto anteriormente (capítulo 4.1.1), el número de Kernel determina la dimensión de los vectores de parámetros obtenidos tras aplicar el método de reducción de las curvas de estudio. Debe ser un número entero igual o mayor que la unidad.

- Número de ciclos del ensayo

Determina el número de repeticiones que sean realizado para unas condiciones de ensayo dadas. Debe ser un número entero igual o mayor que la unidad.

- Número de variantes

Indica el número de especies diferentes sobre las que se han llevado a cabo los ensayos. Debe ser un número entero igual o mayor que la unidad.

- Fracción inicial

Con el fin de analizar únicamente aquellos datos que más información nos aporta, este valor nos permite eliminar los datos extremos del principio de las curvas de estudio. Este parámetro puede tomar cualquier valor decimal entre 0 y 1, significando 0 que no se elimina ningún dato y 1 que eliminamos el 100% de los datos.

- Fracción final

Similar al parámetro anterior, siendo la única diferencia que en este caso los valores se eliminan empezando por el final de las curvas.

- Ramas a analizar

Dependiendo del tipo de sustancia ensayadas nos puede interesar analizar todas ramas o sólo aquella que sea más representativa. Con este parámetro podemos seleccionar el número de ramas a analizar, siendo los valores admitidos: 0 (oxidación), 1 (ambas), 2 (reducción).

Una vez introducidos todos los parámetros del proyecto, el programa calcula las variables que serán utilizadas como datos de entrada posteriormente en los métodos de análisis. Esto se realiza mediante la función *genera_mat_v3()*.

$$[result\_2way, result\_3way, clases, labsens] = genera\_mat\_v3(vinos, ntrozos, Nmostos, Nrepe, fr\_I, fr\_F, rama)$$

Función 5.1

Las variables que se pasan a esta función son:

- vinos: El archivo donde se almacena la estructura de datos que contiene los resultados de los ensayos (véase el capítulo 5.6).
- ntrozos: Número de kernels que se van a obtener.
- Nmostos: Número de especies diferentes que se han ensayado.
- Nrepe: Número de ciclos completos que se han realizado, es decir, el número de ciclos totales menos uno.
- fr_I: Indica la fracción inicial de la curva que se va a eliminar. Se expresa en tanto por uno.
- fr_F: Indica la fracción final de la curva que se va a eliminar. Se expresa en tanto por uno.
- rama: Indica los tramos de la curva a analizar: anódica (0), catódica (2) o ambas (1).

Para utilizar esta función el programa utiliza los parámetros introducidos por el usuario al crear el proyecto y al realizar la configuración.

Esta función es capaz de detectar el número de sensores que se han utilizado para realizar los ensayos a partir de los datos contenidos en el archivo .mat que contiene las mediciones. Debido a esto no es necesario introducir este valor en la configuración del proyecto.

Antes de analizar los datos se elimina el primer ciclo. Para entender esto debemos observar el método experimental. Al iniciar un ensayo partimos de 0V y vamos incrementando el voltaje hasta alcanzar el valor máximo, por ejemplo 1V. Después se va disminuyendo el voltaje hasta llegar al mínimo, por ejemplo -1V. Hasta aquí el primer ciclo. Los ciclos sucesivos empiezan en el valor mínimo, suben hasta el máximo y regresan al mínimo. Como se puede ver, el primer ciclo está incompleto y de ahí surge el motivo de ser eliminado.

Seguidamente, la función aplica una de las técnicas de pre-procesamiento vistas en el capítulo 4.1.1: los kernels. Para ello se calcula cada componente del vector resultante como la suma integrada de la multiplicación punto a punto de cada curva Kernel por la curva respuesta.

Además de todo lo anterior, la función *genera_matV3()* añade otras variables al '*Workspace*' actual que pueden resultar útiles. Este es el caso de *clases* (vector con la etiqueta de las distintas especies) o *labsens* (vector con las etiquetas de las especies para las gráficas de los resultados).

5.9. Cálculo

Una vez realiza la configuración del proyecto (capítulo 5.8) podemos realizar los análisis disponibles en el programa. Esto se hace con la opción 2, 3 o 4 del menú del proyecto, según el tipo de análisis deseado.

En este capítulo se describe el proceso de cálculo de los diferentes tipos de análisis, dejando para el capítulo 6 la puesta en práctica dentro del proyecto de la lengua electrónica.

5.9.1. Análisis en Componentes Principales

En la biblioteca de funciones de MatLab nos encontramos con la siguiente función:

$$[\text{coeff}, \text{score}, \text{latent}, \text{tsquared}, \text{explained}, \text{mu}] = \text{pca}(\mathbf{X})$$

Función 5.2

Está función aplica el método de análisis en componentes principales a la matriz de parámetros X. Los resultados devueltos que nos interesan son los siguientes:

- coeff: matriz que contiene los coeficientes de las componentes principales (loadings).
- score: matriz que contiene los valores de las componentes principales (score). Las filas se refieren al valor de estudio y las columnas a las componentes.
- latent: vector que contiene la varianza de las columnas de la matriz score.
- explained: vector que contiene el porcentaje de la varianza total para cada componente.

Como datos de entrada de este método se utiliza la matriz *result_2way* generada anteriormente. Para que *result_2way* sea compatible con la función *pca()* debemos aplicarla un proceso de estandarización. Esto se realiza con la función *zscore()* de MatLab.

$$Z = zscore(X)$$

Función 5.3

Esta función nos devuelve la matriz original de tal forma que sus datos están centrados por columnas para tener una media igual a 0 y además escalada para que su desviación estándar sea igual a 1.

Tras realizar el análisis se nos muestra en pantalla los valores del vector *explained* que contiene el porcentaje de la varianza total para cada componente, lo que nos permite conocer la aportación de cada componente al resultado final.

Mediante al uso de la función *plotscat_uva()* podemos graficar los resultados.

$$plotscat\_uva(x, clab, testb, dim)$$

Función 5.4

Los valores que se deben pasar a esta función son los siguientes:

- x: Matriz con los datos a graficar. En nuestro caso será *score* o alguno de las matrices *loading*.
- clab: Vector que contiene las etiquetas de los ensayos. Se genera automáticamente al configurar el proyecto.
- testb: Indica si se desean obtener los resultados definitivos (0) o si se esta realizando una prueba (1).
- dim: Número de componentes a graficar. Deberá tomar valor 2 (2D) o 3 (3D).

En nuestro caso utilizamos la función de la siguiente manera:

$$plotscat\_uva(score, clases, 0{,}3)$$

Función 5.5

Con esto obtenemos la representación gráfica de las componentes de una manera similar a lo mostrado en la Figura 5.13. En estas gráficas en el eje 1 se representa la primera componente, en el 2 la segunda y en el 3 la tercera.

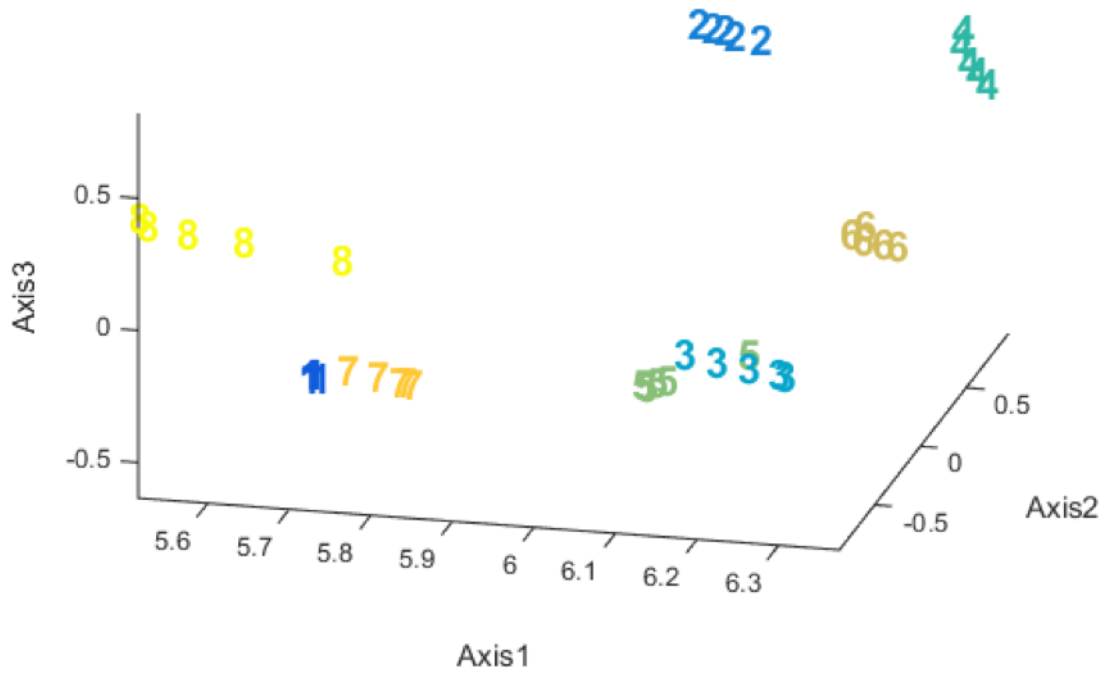


Figura 5.13 - Ejemplo de gráfica obtenida con plotscat_uva()

Por último se crea un biplot. Para ello hacemos uso de la función *biplot()*, utilizando como variable de entrada de la Función 5.6 la variable coeff generada por la función *pca()* (Función 5.2).

$$biplot(coefs)$$

Función 5.6

Un biplot es una gráfica que permite visualizar la magnitud y signo de la contribución de cada variable a las dos o tres primeras componentes principales. Se utiliza un criterio de signos que fuerza a que el elemento de mayor valor absoluto sea positivo. En algunos casos esto supone la inversión de los elementos de una columna, lo cual no afecta a la interpretación de los resultados y a menudo facilita la interpretación de los mismos.

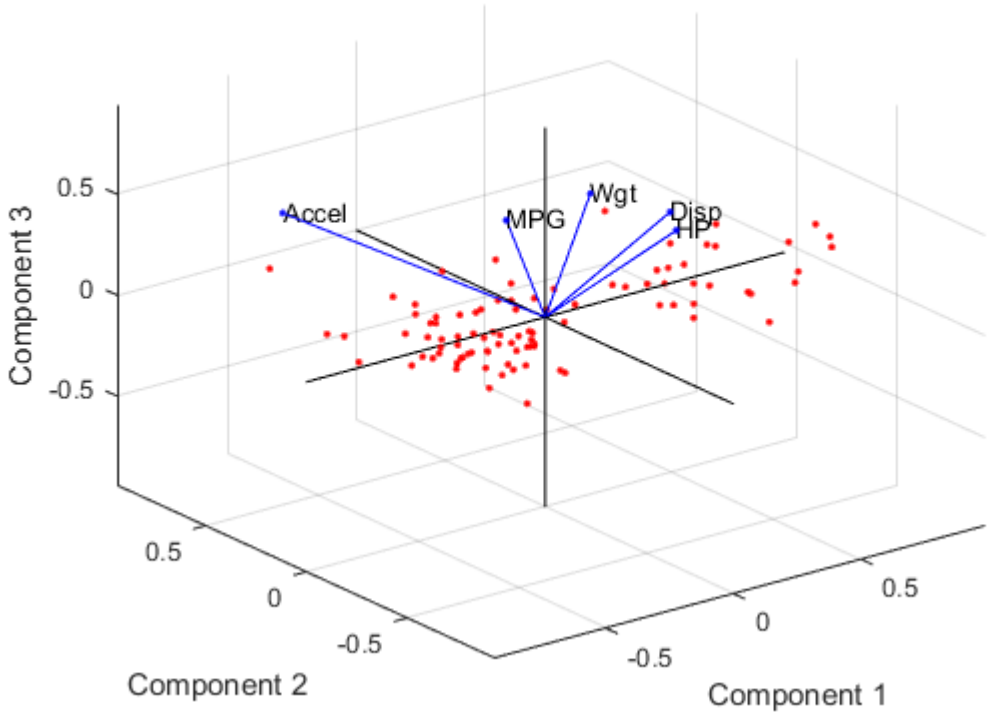


Figura 5.14 – Ejemplo de gráfica 3D generada por biplot()

Para que los resultados obtenidos en caso sean comparables con otros, *biplot()* realiza un ajuste. Este consiste en dividir cada componente representada por el valor máximo absoluto de todos los valores y multiplicarla por la longitud del vector (columna) de mayor número de elementos.

biplot() nos genera gráficos 2D si la matriz *coeffs* tiene dos columnas y 3D si tiene tres. En nuestro caso, el resultado debería ser tridimensional, sin embargo, se observa que la tercera coordenada empeora el resultado al reducir la claridad del gráfico. Por ello se elimina la tercera columna de la matriz *coeffs*.

Además, se pueden introducir algunos parámetros para adaptar la representación a nuestro gusto o necesidades. En nuestro caso eliminamos algunos elementos, dejando únicamente los ejes, los puntos representados y la etiqueta que permite su identificación.

Teniendo en cuenta todo lo anterior, el comando utilizado en MatLab quedaría:

$$biplot(coeff(:,1:2),'varlabels',labsense,'LineStyle','none')$$

Función 5.7

Con el comando anterior el resultado mostrado por pantalla sería similar a lo siguiente:

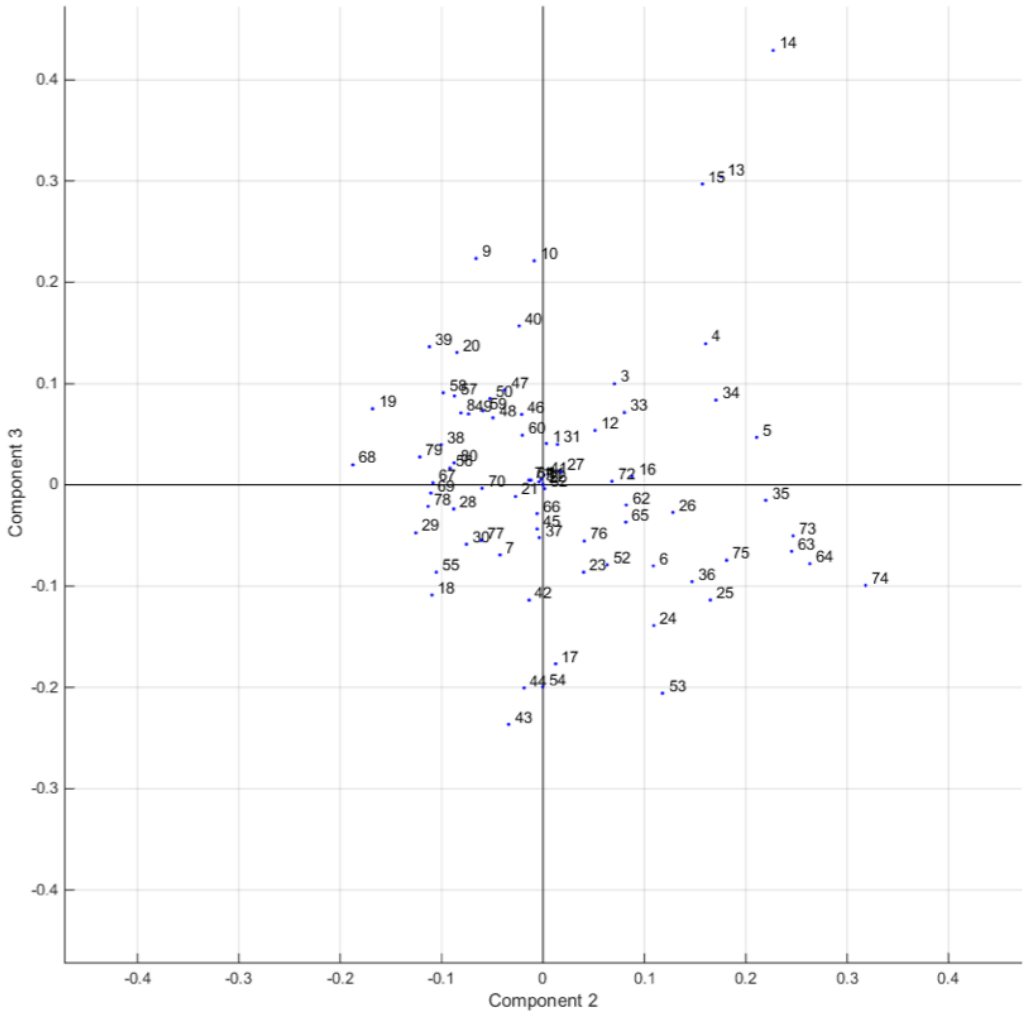


Figura 5.15 – Gráfico 2D generado por la función biplot() modificada

5.9.2. Análisis Parafac

Para realizar el cálculo del análisis de Parafac se sigue el algoritmo definido en el capítulo 4.3:

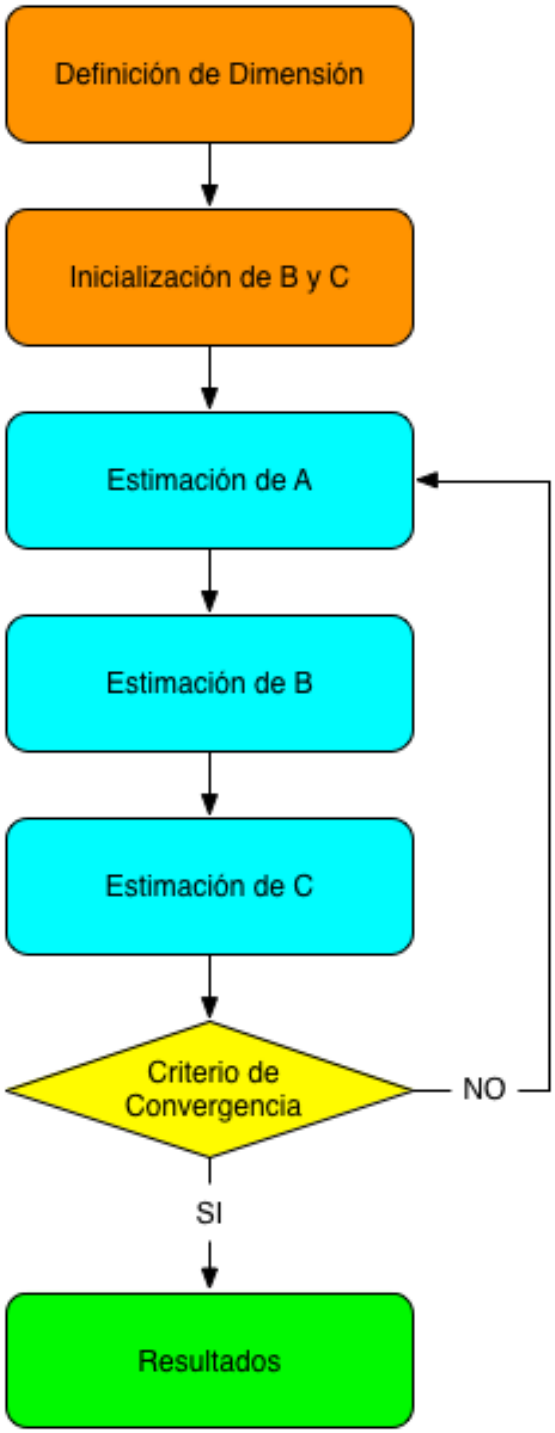


Figura 5.16

Para realizar el análisis Parafac utilizamos como datos de entrada la matriz *result_3way*. Para que sea compatible con los procedimientos posteriores es necesario realizar un proceso de estandarizado. Para ello hacemos uso de la función *nprocess()*.

$$[Xnew] = nprocess(X, Cent, Scal)$$

Función 5.8

Esta función estandariza la matriz mediante la aplicación de los métodos de pre-procesado: centrado y escalado. Los datos de entrada de esta función son:

- X: La matriz de datos que se desea estandarizar (*result_3way*).
- Cent: Vector que indica en qué modo se desea aplicar el centrado. Para realizarlo respecto el primero sería: [1,0,0].
- Scal: Vector que indica el valor de la desviación estándar que se desea fijar como objetivo en el proceso de escalado. En nuestro caso deseamos un valor 0, por tanto debe ser: [0,0,0].

Tras este proceso obtenemos la matriz estandarizada *X3*, que será nuestra matriz de datos a partir de este punto.

Antes de realizar el análisis se debe buscar el número de componentes más adecuado. Para ello hacemos uso de la función *pftest()*.

$$[ssX, Corco, It] = pftest(NumReps, X, Fac)$$

Función 5.9

Los valores de entrada de esta función son:

- NumReps: Número de veces que se repite el test para cada número de componentes.
- X: Matriz que contiene los datos que se van a analizar.
- Fac: Indica el número máximo de componentes que se quiere ensayar, obteniendo los resultados del test des una componente hasta el valos indicado por este parámetro.

Los valores devueltos por la función son:

- ssX: El error en la suma de los cuadrados. Se debe prestar atención cuando entre componentes sucesivas se produzca un aumento brusco

en su valor, ya que esto significa que los valores que menos información aportan están influyendo en el resultado final.

- Corco: Consistencia del núcleo expresada en forma de porcentaje. Valores por encima de 80 indican que el modelo es válido, por debajo de 40 el modelo no es válido y para valores intermedios puede ser bueno pero es difícil de estimar.
- It: Número de iteraciones hasta la solución. Un aumento brusco de este valor indica que se ha producido algún problema en el cálculo, normalmente debido a un número excesivo de componentes.

Si aplicamos la función *pftest()* a la matriz normalizada (*X3*) obtenemos algo similar al siguiente resultado:

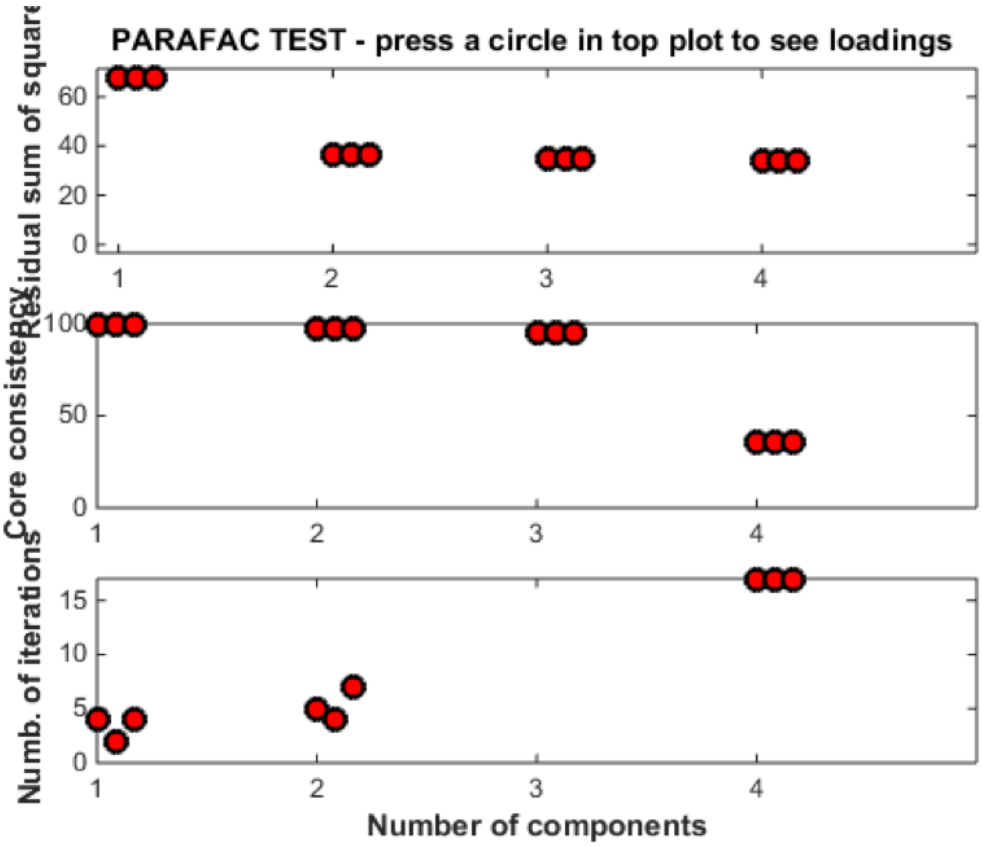


Figura 5.17

En la abscisa de las gráficas se muestra el número de componentes. Para decidir cuál es número más adecuado debemos prestar atención a los resultados mostrados en la segunda gráfica '*Core consistency*'. Para que el cálculo sea bueno, este valor debe ser lo más próximo posible a 100.

Una vez introduzcamos el número de componentes podemos proceder al análisis. Este se realiza con la función *parafac()*

$$[Factors, it, err, corcondia] = parafac(X, Fac)$$

Función 5.10

La variable de entrada *X* (en nuestro caso *X3*) es la matriz que contiene los datos a analizar y la variable *Fac* el número de componentes. Los valores devueltos son:

- Factors: Vector que contiene los resultados del análisis. Cada elemento es una de las matrices solución, esto es, score y loadings.
- it: Indica el número de iteraciones hasta la solución. El número máximos de iteraciones es 2500, por lo que si se alcanza este valor significa que no se ha alcanzado la convergencia.
- err: Indica el ajuste del modelo calculado como la suma de los cuadrados de los errores.
- corcondia: Mide la consistencia del núcleo en porcentaje. Si este valor esta muy lejos del 100% significa que el modelo no es válido.

Para aplicar el algoritmo de cálculo del método debemos hacer uso del *producto de Kronecker*. Este se realiza mediante la función *kron()*:

$$K = kron(A, B)$$

Función 5.11

Donde *K* es la matriz solución y *A* y *B* las matriz a las que se aplica el operador. Recuérdese que para el *producto de Kronecker* no se cumple la propiedad conmutativa, por lo que A y B no se pueden cambiar de orden.

Como hemos visto, después de aplicar la función *parafac()* disponemos una variable *Factors* que contiene la solución del análisis. *Factors* se puede graficar mediante el uso de *plotfac()*.

$$plotfac(Factors)$$

Función 5.12

La función *plotfac()* nos muestra por pantalla una gráfica para cada uno de los elementos que de los que se compone *Factors*, permitiéndonos realizar un análisis de cada sus modo según sus componentes.

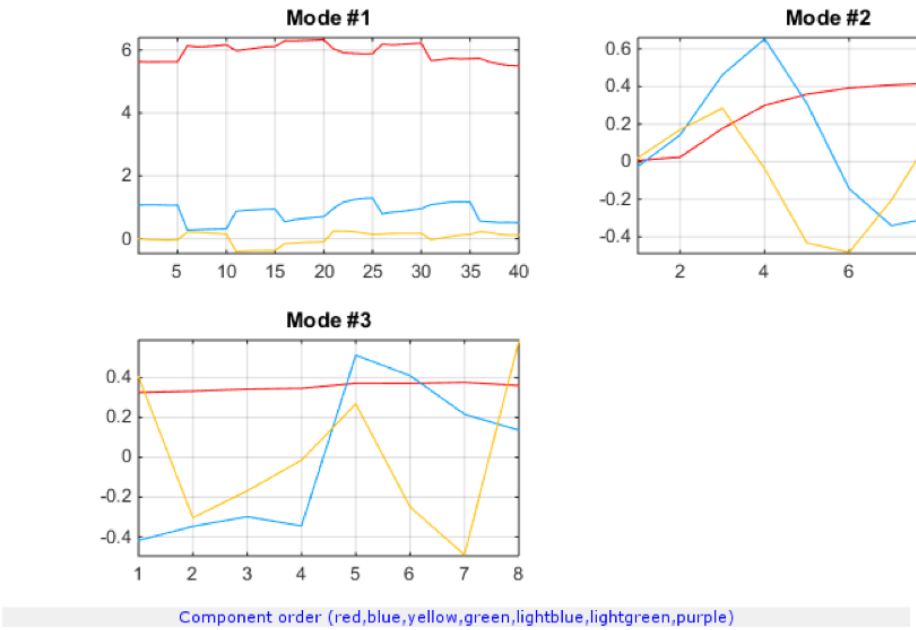


Figura 5.18 – Ejemplo de gráficas generadas mediante plotfac

Aunque de las gráficas generadas por *plotfac()* (Figura 5.18) podemos extraer varias conclusiones, resulta más útil graficar las componentes mediante la función *plotscat_uva()* (Función 5.5). Antes de proceder a ello debemos extraer las matrices de *Factors*. Estas son: una *score* (A) y N-1 *loadings* (B, C, etc.), donde N es el número de componentes seleccionado anteriormente. Para ello utilizamos la función *fact2let()*.

$$[A, B, C, D, \dots] = fac2let(Factors)$$

Función 5.13

Si pasamos la variable Factors a la función *fac2let()* nos devuelve tantas matrices (score y loadings) como número de componentes hemos utilizado. Las matrices A, B, C… obtenidas son compatibles con *plotscat_uva()*, debiendo hacerse la llamada a la función de la siguiente forma para el score:

$$plotscat\_uva(M, clases, 0, size(M, comps))$$

Función 5.14

Y para los loadings:

$$plotscat\_uva(M, [1, size(M, 1)], 0, size(M, comps))$$

Función 5.15

En la Función 5.14 y Función 5.15 'M' es la matriz escogida y '*comps*' el número de componentes que se desea graficar. Los resultados obtenidos son similares a los mostradas a continuación:

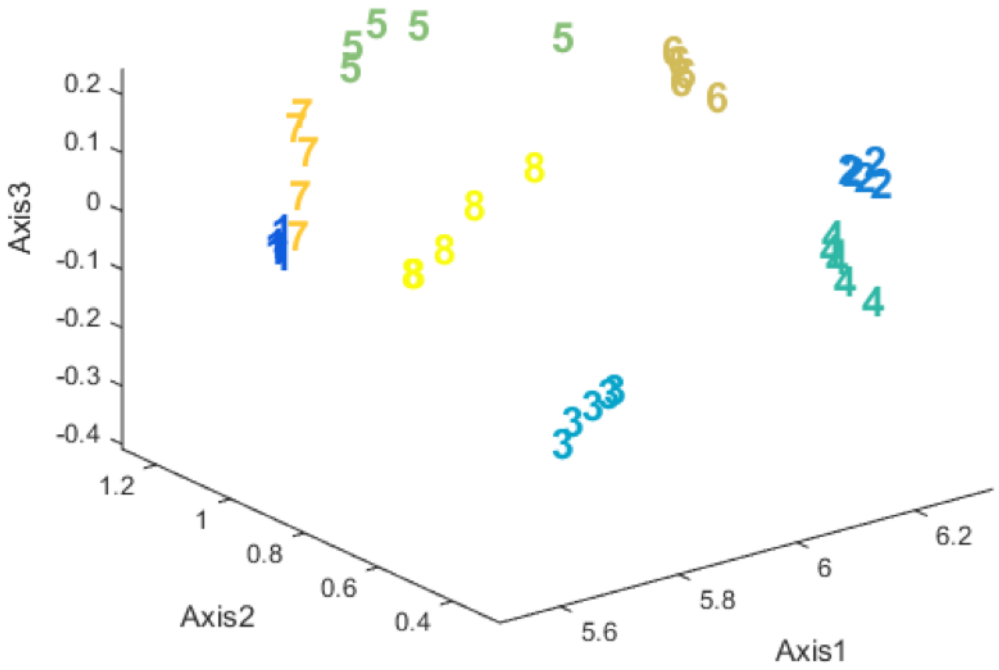


Figura 5.19 – Ejemplo de plotscat_uva() aplicado a una matriz A

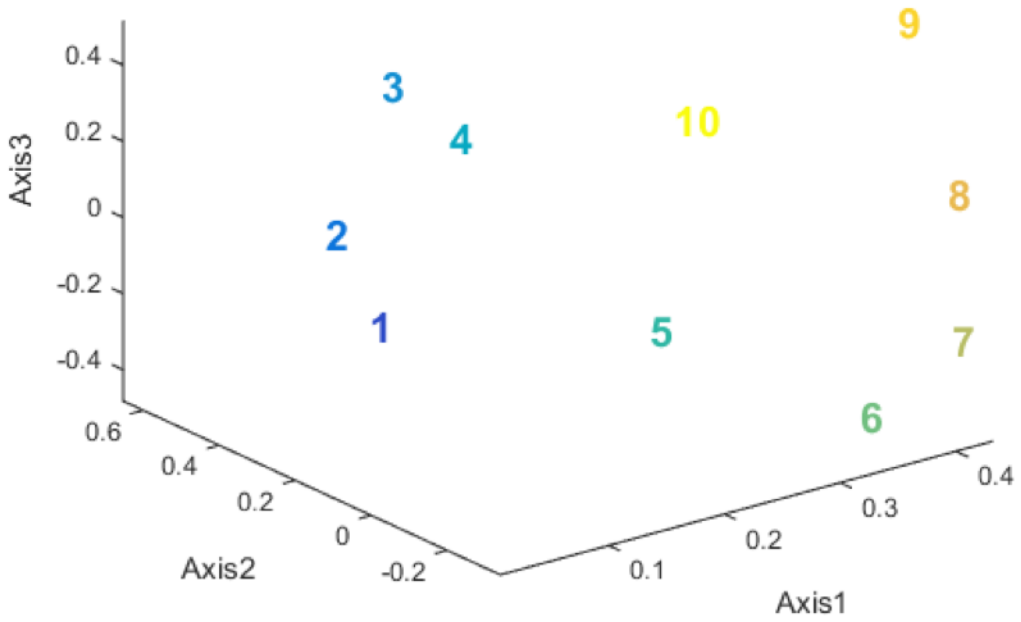


Figura 5.20 – Ejemplo de plotscat_uva() aplicado a una matriz B

Las gráficas mostradas tendrán la misma dimensión que el parámetro 'comps', el cual pude ser igual o inferior al número de componentes seleccionado para realizar el análisis. Por ejemplo, en la Figura 5.19 se representan tres componentes de una matriz A de muestra, dando como resultado una gráfica con tres dimensiones. En algunos casos, si la tercera componente no nos aporta un extra de información o si dificulta la visualización de los resultados, es posible representar sólo dos componentes (Figura 5.21).

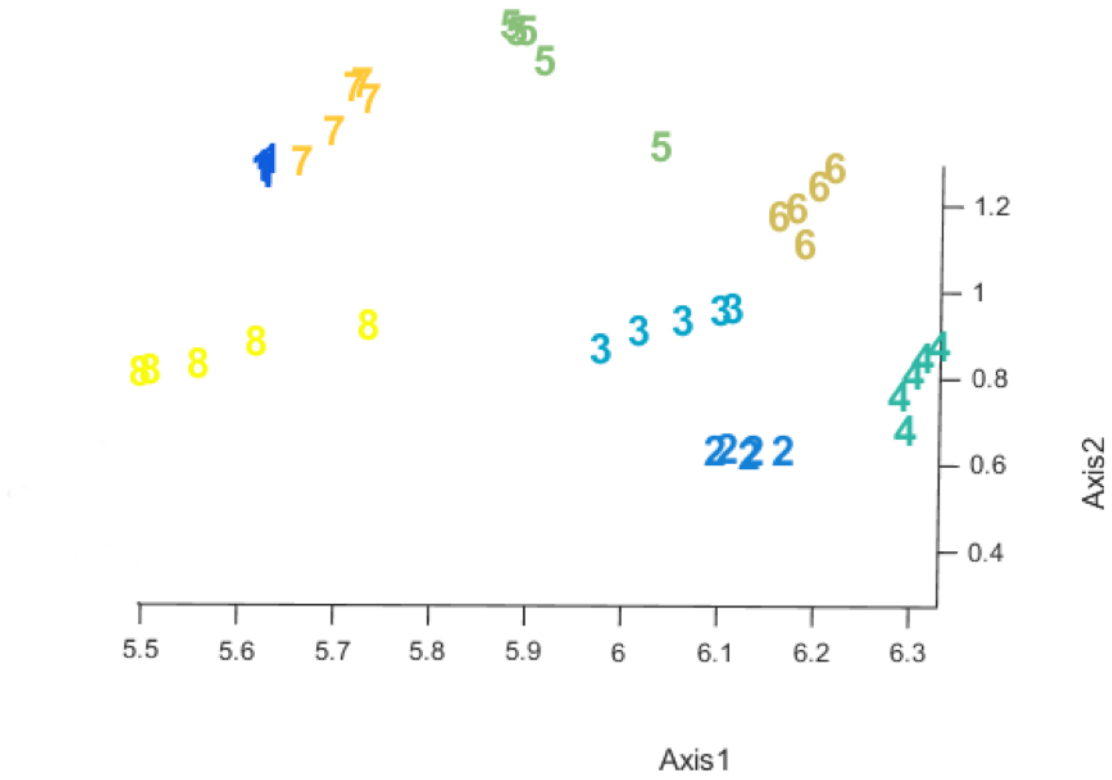


Figura 5.21 – Ejemplo de plotscat_uva() para dos componentes de una matriz A

5.9.3. Análisis Tucker

Para realizar el cálculo del análisis de Tucker se sigue el algoritmo definido en el capítulo 4.4:

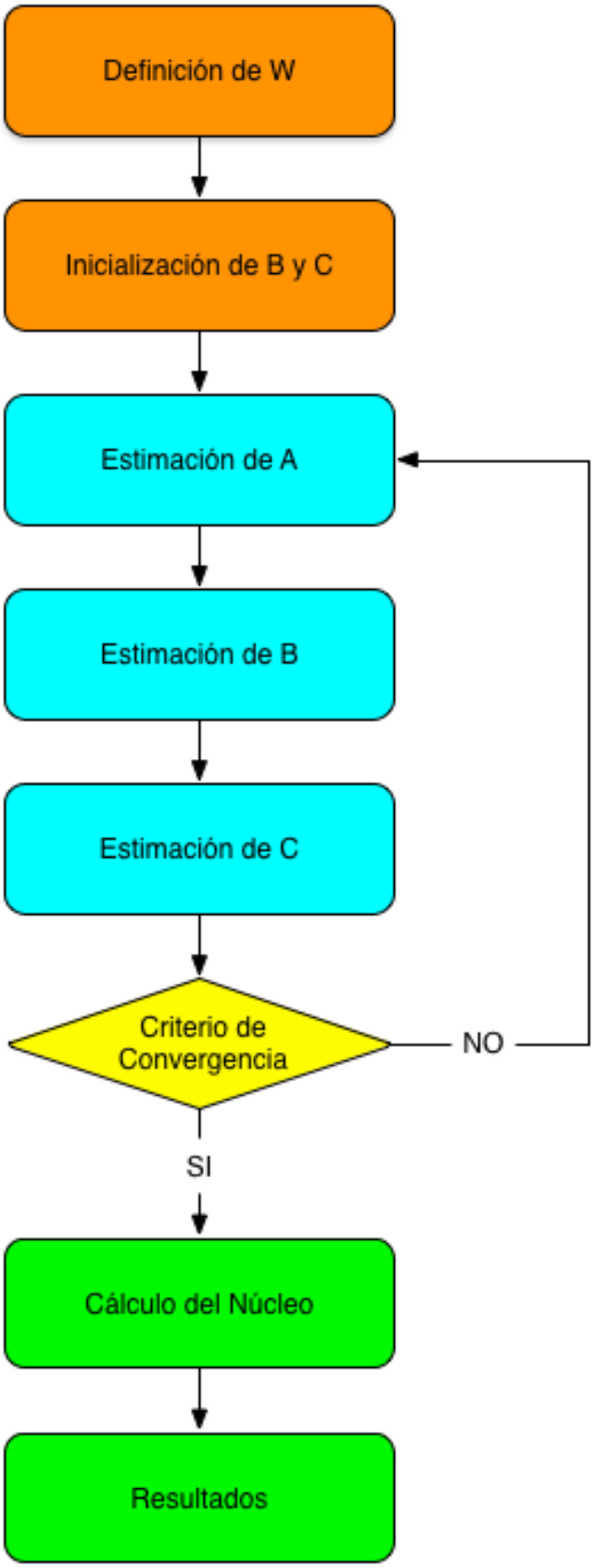

Definición de W
Inicialización de B y C
Estimación de A
Estimación de B
Estimación de C
Criterio de Convergencia
NO
SI
Cálculo del Núcleo
Resultados

Figura 5.22

En primer lugar debemos definir el vector W. Este vector de dimensión 3 contiene el número de componentes que se va a utilizar para cada uno de los tres modos. Debe cumplir que el producto de los dos índices más bajos sea mayor o igual que el índice más alto. Generamos todas las combinaciones posibles para índices con valores entre 1 y 4, y graficamos las características de cada uno.

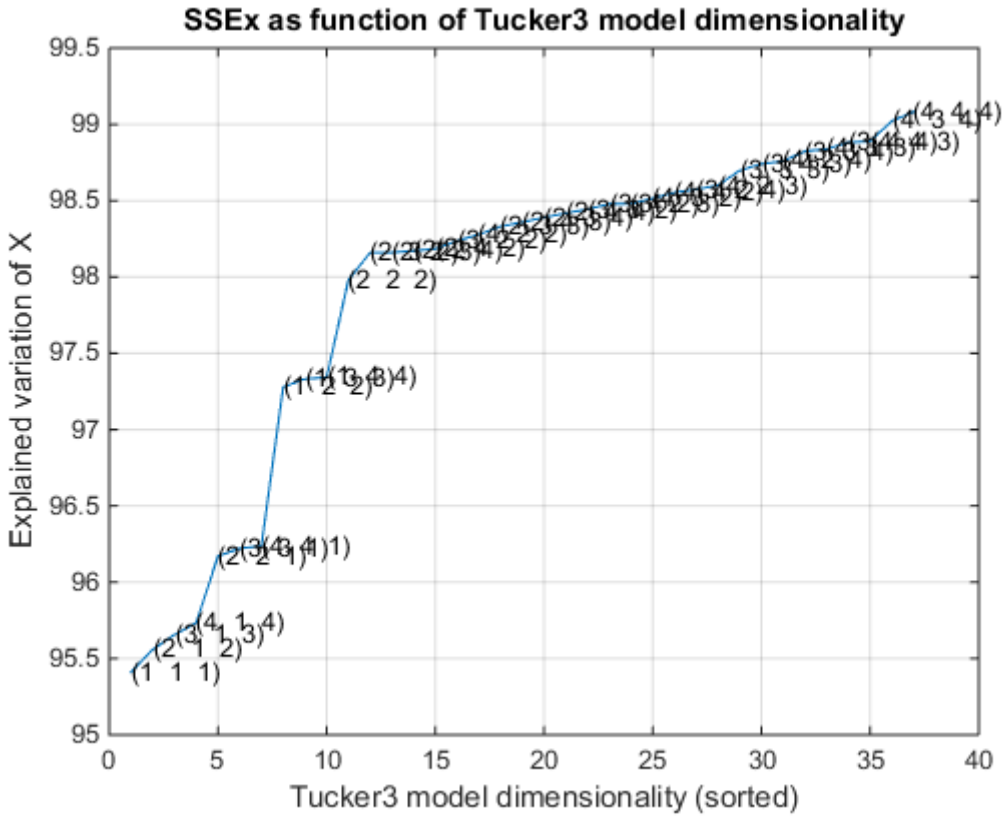


Figura 5.23

A continuación, el programa nos solicita introducir las componentes del vector W (Figura 5.25). Para seleccionar los valores más adecuados debemos atender a los resultados de la gráfica de dimensionalidad del modelo (Figura 5.23).

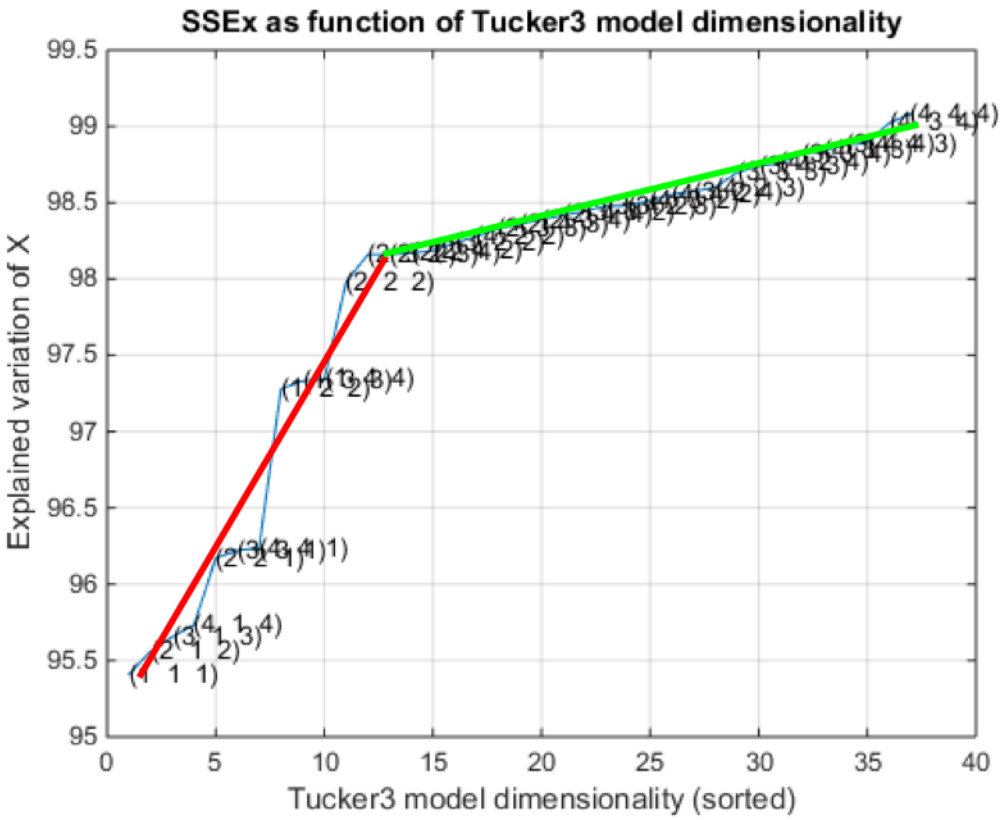


Figura 5.24

Normalmente en la curva obtenida pueden observarse dos tramos bien diferenciados, como vemos en la Figura 5.24. En un primer tramo (línea roja) se observa una pendiente muy marcada. En el segundo (línea verde), la pendiente se suaviza. Esto es debido a la estabilización del modelo.

Se recomienda escoger alguno de los conjuntos de valores que pertenezca al segundo tramo de la curva para que el modelo sea estable, y que esté próximo al punto de intersección, ya que al reducir el número de componentes reduciremos también los requisitos y coste del cálculo.

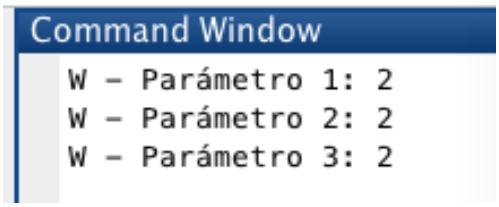


Figura 5.25

Con esto podemos proceder con el análisis. Este se realiza con el comando *tucker()*.

$$[Factors, G] = tucker(X, Fac)$$

Función 5.16

Las variables de entrada a esta función son:

- X: Matriz que contiene los datos.
- Fac: Vector con el número de componentes para cada modo.

Esta función nos devuelve las variables Factors y G. Factors es un vector cuyas componentes son la matriz score y las loadings. G es la matriz núcleo.

Como ya se ha visto, dentro del algoritmo de cálculo del método de Tucker se incluyen el *producto de Kronecker* () y . Este último se realiza mediante el comando *svd()*.

$$S = svd(A, econ)$$

Función 5.17

Donde S es la matriz devuelta, *A* la matriz de entrada y *econ* el número de columnas de la matriz S.

Una vez realizado el análisis el programa nos muestra en la ventana de comandos los valores característicos del núcleo generados con la función *explcore()*.

$$explcore(G, n)$$

Función 5.18

Las variables de entrada de esta función son:

- G: Núcleo del método generado por la función *Tucker()*.
- n: Muestra únicamente las 'n' combinaciones de factores con mayor variación.

A continuación podemos ver un ejemplo de los valores que muestra en pantalla:

```
Col1: Number in list
Col2: Index to elements
Col3: Explained variation (sum of squares) of the core.
Col4: Core entry.
Col5: Sq. core entry.
 1    ( 1, 1, 1)        41.77015%        -0.00001          0.00000
 2    ( 1, 2, 2)        18.61738%         0.00000          0.00000
 3    ( 2, 2, 2)        11.09745%        -0.00000          0.00000
 4    ( 2, 1, 1)         8.98136%        -0.00000          0.00000
```

Figura 5.26 – Ejemplo de valores devueltos por explcore()

Si las 3 componentes del vector W son iguales, tiene sentido el concepto de diagonal de la matriz G (núcleo). Cuando se cumple esta condición se procede a analizar de nuevo el núcleo pero antes aplicándole dos procesos de rotación.

En primer lugar se rota el núcleo mediante el método de diagonalización del mismo. Para ello utilizamos la función *maxdia3()*.

$$[Gd, Od1, Od2, Od3] = maxdia3(G)$$

Función 5.19

Cuando pasamos a esta función la matriz G nos devuelve el núcleo rotado (Gd) y las matrices de rotación ortogonales para cada uno de los modos (Od1, Od2, Od3).

Tras este proceso pasamos la matriz *Gd* a la función *explcore()*, que nos muestra en pantalla los resultados obtenidos:

```
Rotación del núcleo: diagonalización

Col1: Number in list
Col2: Index to elements
Col3: Explained variation (sum of squares) of the core.
Col4: Core entry.
Col5: Sq. core entry.
 1    ( 1, 1, 1)        55.25908%        -0.00001          0.00000
 2    ( 2, 2, 2)        36.95602%        -0.00001          0.00000
 3    ( 1, 2, 2)         3.86327%        -0.00000          0.00000
 4    ( 2, 1, 1)         2.58294%        -0.00000          0.00000
 5    ( 2, 2, 1)         0.69637%         0.00000          0.00000
 6    ( 1, 1, 2)         0.46572%         0.00000          0.00000
 7    ( 2, 1, 2)         0.10581%        -0.00000          0.00000
```

Figura 5.27 – Ejemplo de núcleo rotado por el método de las digonales

También se realiza el caso de rotación por optimización de los cuadrados de los elementos de la diagonal. Este método se aplica mediante la función *maxvar3()*.

$$[Gv, Ov1, Ov2, Ov3] = maxvar3(G)$$

Función 5.20

La variable de entrada que utilizamos para esta función es el núcleo original G (no el rotado Gd). Las variables devueltas, al igual que en el otro método de rotación, son: el nuevo núcleo rotado (Gv) y las matrices de rotación ortogonales para cada uno de los modos (Ov1, Ov2, Ov3).

Tras completar el proceso de rotación analizamos el núcleo rotado Gv mediante la función *explcore()*, la cual nos muestra los resultados por pantalla:

```
Rotación del núcleo: optimización de los cuadrados

Col1: Number in list
Col2: Index to elements
Col3: Explained variation (sum of squares) of the core.
Col4: Core entry.
Col5: Sq. core entry.
 1    ( 1, 1, 1)        57.11107%        -0.00001          0.00000
 2    ( 2, 2, 2)        34.42957%        -0.00001          0.00000
 3    ( 1, 2, 2)         6.12164%         0.00000          0.00000
 4    ( 2, 2, 1)         1.02129%        -0.00000          0.00000
 5    ( 2, 1, 1)         0.93625%         0.00000          0.00000
 6    ( 1, 1, 2)         0.20920%        -0.00000          0.00000
 7    ( 2, 1, 2)         0.14426%        -0.00000          0.00000
```

Figura 5.28 – Ejemplo de núcleo rotado por el método de la optimización de cuadrados

Para poder extraer las componentes de *Factors* debemos hacer uso de *fact2let()* (Función 5.13), obteniendo así las matrices A, B y C. Con esto podemos proceder con el análisis gráfico.

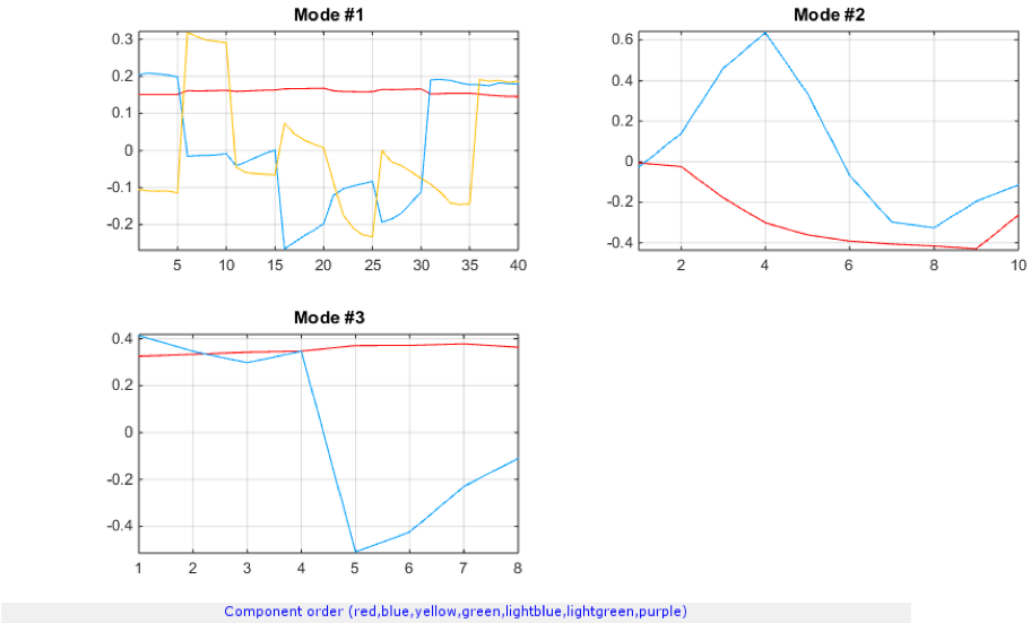


Figura 5.29 – Ejemplo de representación de los tres modos de Tucker3

Al igual que en el análisis de Parafac, las gráficas de las componentes (Figura 5.29) permiten interpretar la solución, sin embargo nos resulta más práctico realizar la representación mediante *plotscat_uva()* (Función 5.14).

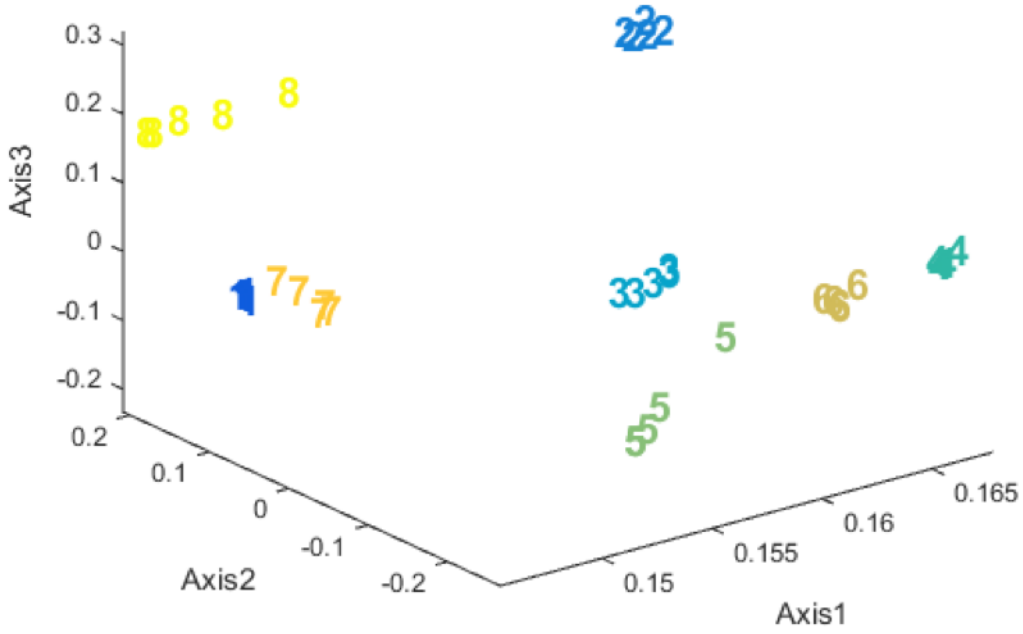


Figura 5.30 –Ejemplo de representación del modo 1 (matriz A) de Tucker3

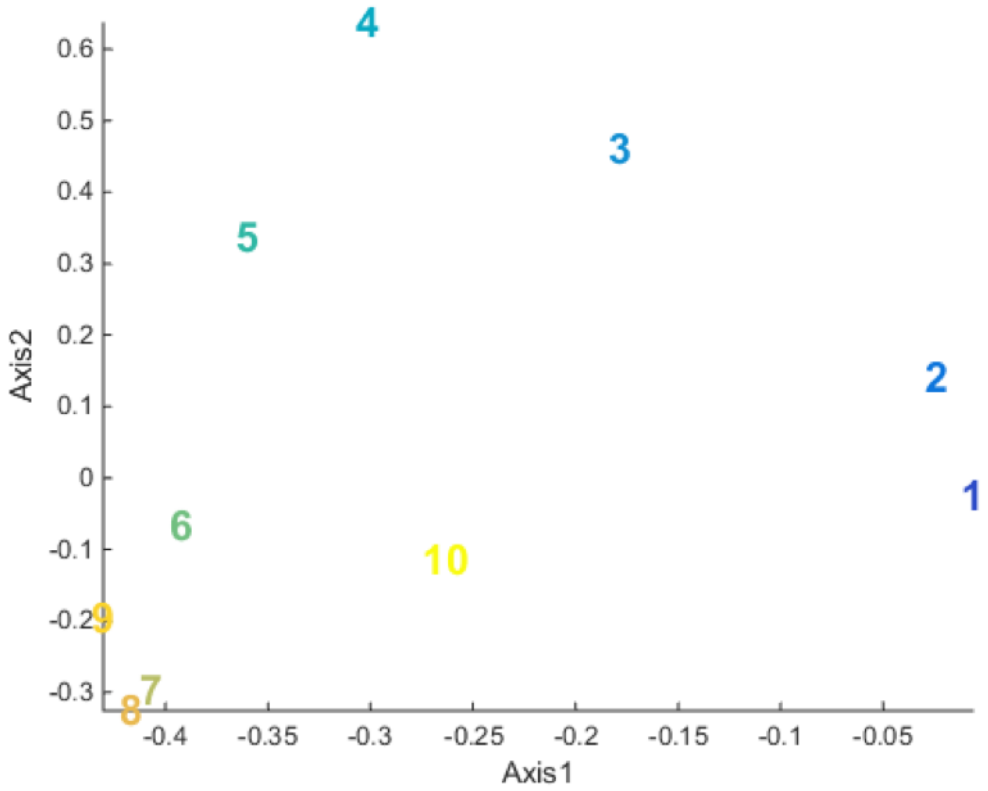


Figura 5.31 – Ejemplo de representación del modo 2 (matriz B) de Tucker3

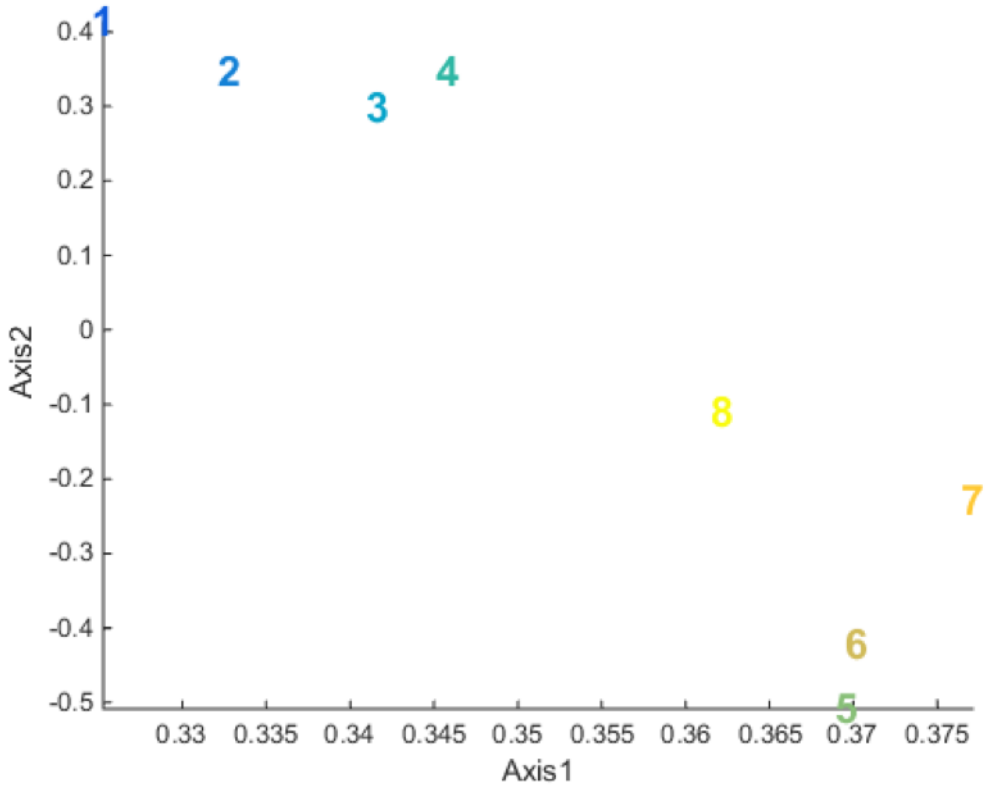


Figura 5.32 – Ejemplo de representación del modo 3 (matriz C) de Tucker3

La dimensión de las gráficas devueltas por *plotscat_uva()* coinciden con el número de componentes establecido en el vector W para el modo representado, excepto cuando este valor sea mayor a 3, que se reducirá a una gráfica tridimensional por ser el mayor número de dimensiones que podemos representar gráficamente.

5.10. Resultados

Todos los resultados mostrados en forma de gráficas durante los procesos de cálculo de los diferentes análisis son guardados en forma de imagen en la carpeta 'data'. Esta carpeta se ubica dentro de la carpeta 'TFG' de archivos de programa.

6. Caso práctico

Como se ha comentado en el capítulo 1, desde hace años en el laboratorio del departamento se está desarrollando un proyecto de lengua electrónica cuya finalidad es la de ser capaz de distinguir distintos tipo de vinos al ser expuesta a ellos.

El software expuesto en el presente documento se sitúa dentro del tratamiento de los datos obtenidos en los ensayos realizados con la lengua electrónica. Es por ello que se considera necesario la puesta en práctica de programa dentro del proyecto citado con el finalidad de comprobar su correcta integración con el resto del conjunto.

6.1. Descripción

El objetivo del proyecto es la creación de una lengua electrónica que mediante su exposición a un determinado medio nos permita caracterizarlo. El motivo para investigar las lenguas electrónicas una legislación de la industria alimentaria cada vez más estricta y a un aumento de las exigencias en el estándar de calidad de los productos. Actualmente las características organolépticas de los productos son analizadas por un grupo de expertos catadores. El principal problema es la pérdida de capacidad sensitivas tras un periodo prolongado debido a la saturación de sus órganos receptores. Además, la objetividad del catador está afectada por las condiciones del entorno, lo cual afecta a la reproducibilidad del ensayo. Es por esto el interés creciente en sistemas automáticos que permitan la medición y clasificación de propiedades como el sabor, calidad, impurezas, etc.

La idea del proyecto es diseñar una lengua electrónica compuesta por una red de sensores de tal manera que cada uno mida una propiedad de la muestra aportando una respuesta al conjunto y obtener así información acerca de la solución analizada.

En el laboratorio se ha desarrollado un sistema de sensores electroquímicos compuesto por una red de sensores voltamétricos a partir de electrodos de materiales electroactivos. Nuestra lengua electrónica está compuesta por cinco biosensores enzimáticos basados en glucosa oxidasa o tirosina y diez sensores basados en ftalocianinas.

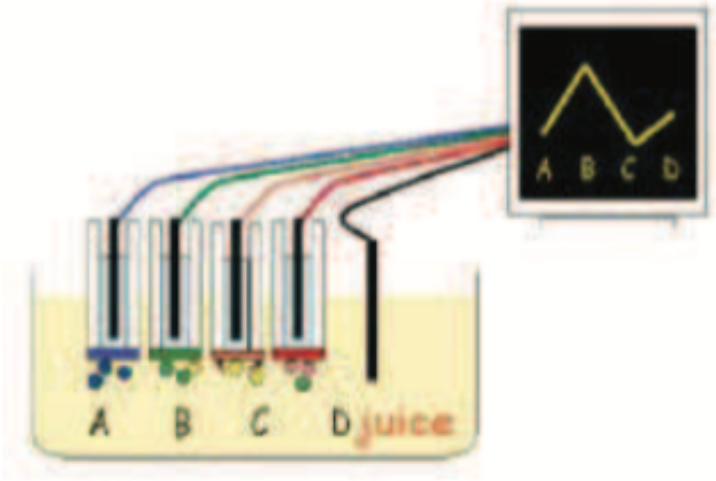


Figura 6.1 – Esquema de medición mediante una lengua electrónica

Las mediciones de la lengua electrónica se basan en la voltametría o voltamperometría, la cual se compone de un conjunto de técnicas de análisis electroquímico que permiten detectar especies electroquímicas a partir de la medida de los cambios de intensidad en función del potencial de la corriente aplicada en el electrodo introducido dentro de la solución a analizar, debido a las reacciones de oxidación-reducción en el electrodo.

El resultado obtenido en los ensayos son curvas voltamétricas bi-evaluadas, es decir, se miden de forma indirecta e inversa. Esto se debe a que en el ensayo se realiza un barrido anódico en el que el potencial va aumentado (curva superior) y otro catódico en el que va disminuyendo (curva inferior).

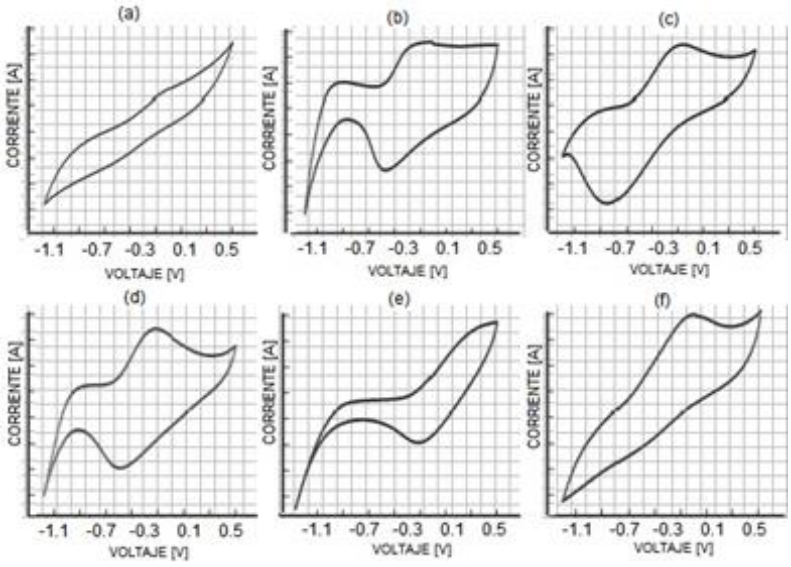


Figura 6.2- Ejemplo de voltametrías cíclicas

6.2. Datos

Tenemos de los datos de un ensayo realizado sobre una serie de mostos de la vendimia de 2012. Disponemos de 15 sensores que registra distintos parámetros de los 5 tipos de vinos diferentes y se repiten 5 veces cada una de las mediciones. Con esto podemos agrupar nuestros datos en una matriz tridimensional de dimensiones 5x15x5 (tipos x sensores x repeticiones), cuya complejidad nos lleva a la realización de análisis multivariante para su interpretación y obtención de conclusiones.

Para que el programa pueda leer los datos de los ensayos se les debe dar el formato según lo especificado en el capítulo 5.6. En nuestro, para crear el archivo .m de MatLab, establecemos un protocolo que nos permita identificar fácilmente los datos contenidos en él. Este consiste en denominar a cada matriz que contiene los registros de curvas voltamétricas de la siguiente manera:

NOMBRE SENSOR + CÓDIGO VINO

Los códigos de nuestros cinco tipos de vino son:

- T1
- P1
- M1
- C1
- G1

A los sensores se les denomina dependiendo de la sustancia química que analizan. Estos son:

- C
- COB
- CU
- LU
- ZN
- LUT
- ZNT
- COBT
- CUT
- CT
- LUG
- ZNG
- COBG
- CG

Con esto, nuestro archivo *vendimia.mat* tendrá el siguiente aspecto:

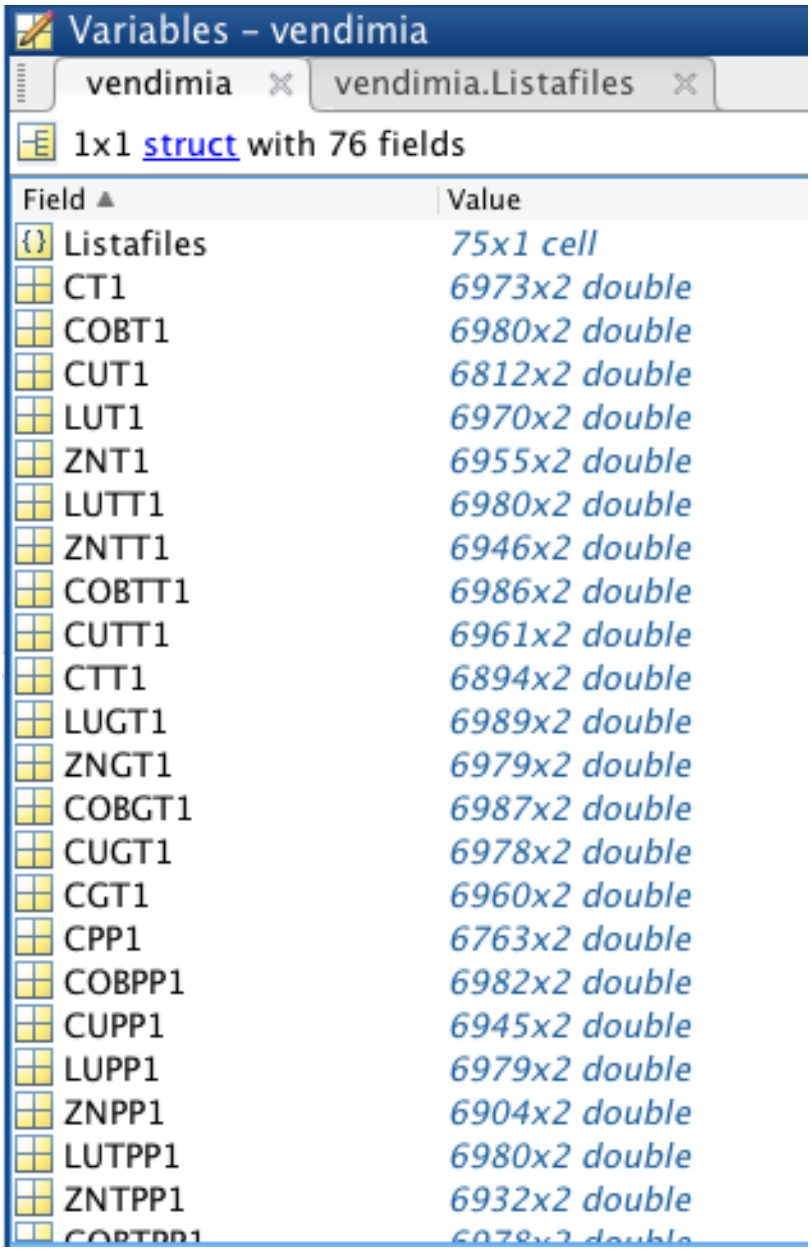


Figura 6.3

6.3. Configuración

Antes de introducir los valores en la configuración se deben tener en cuenta una serie de consideraciones relacionadas con las curvas voltamétricas cíclicas y aplicarlo a nuestro caso particular.

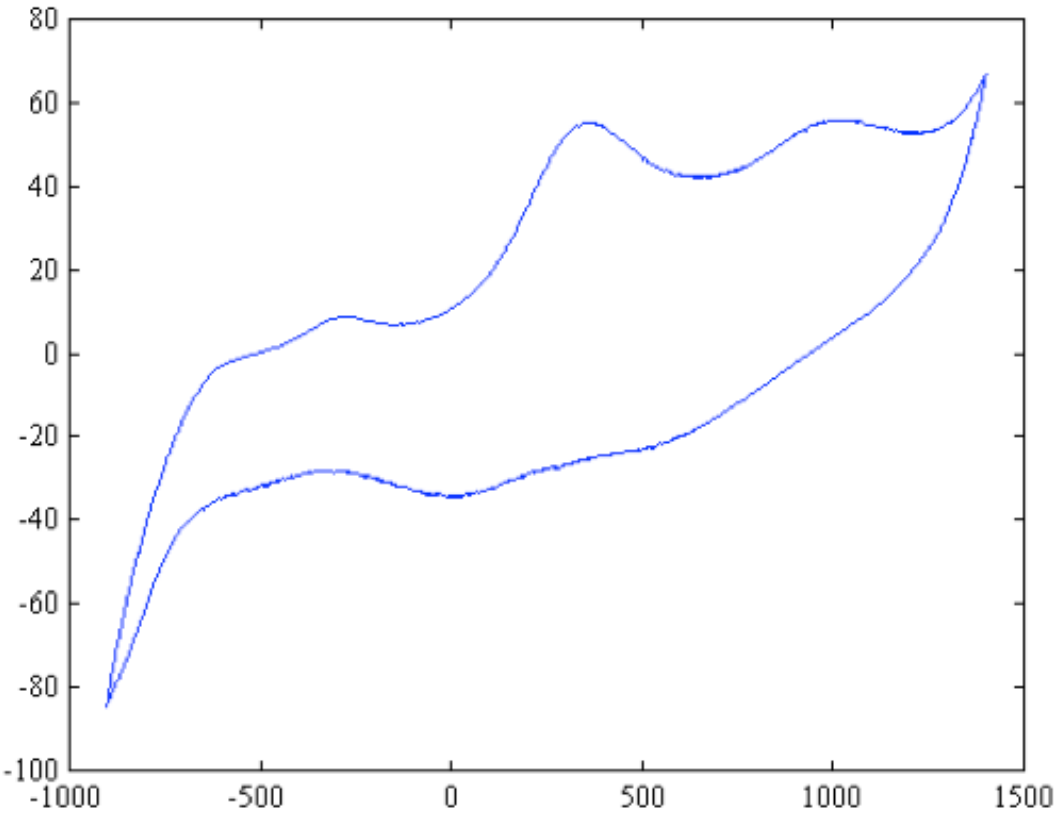


Figura 6.4 – Curva voltamétrica cíclica

En primer lugar el software nos solicitará el número de Kernel. No se ha encontrado ningún criterio exacto para escoger el número más adecuado de Kernel para cada caso, por lo que debemos observar las características de las curvas voltamétricas. Nos apoyaremos principalmente en dos factores: el número de picos y nuestra experiencia. El número de Kernel debe ser tal que no se pierda información de los picos en el proceso de simplificación. Además se observa la importancia de la experiencia en la aplicación del método de los Kernel a la hora de decidir el número de estos. En nuestro caso consideramos 10 el número de Kernel adecuado.

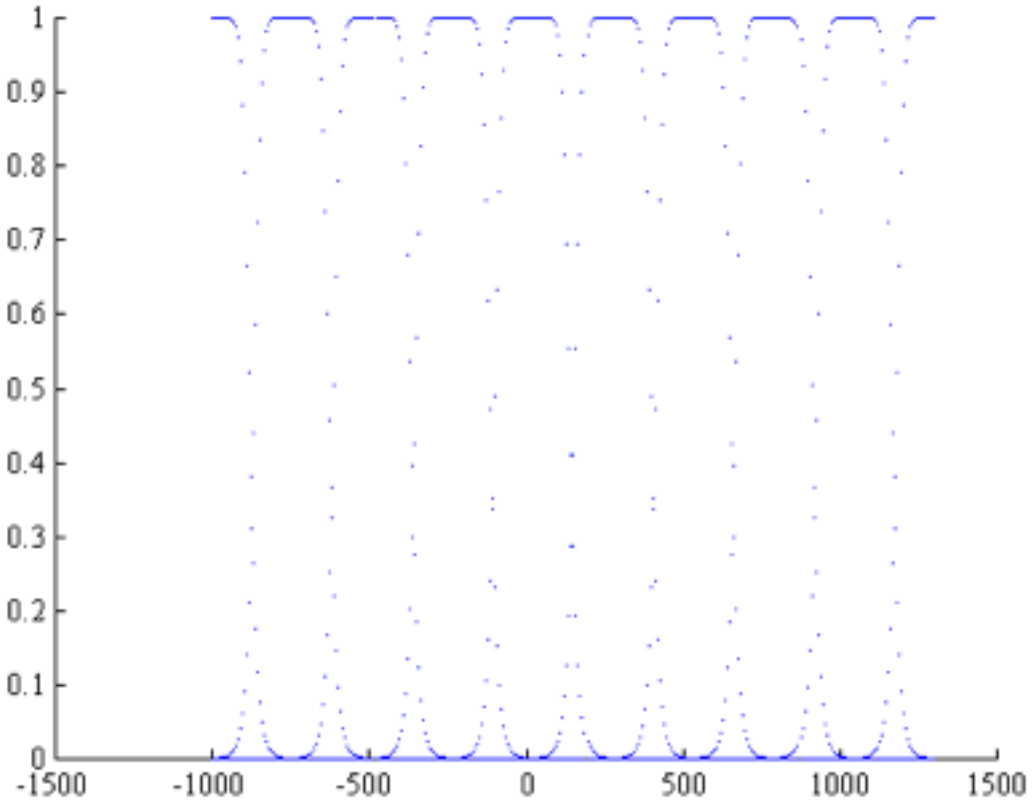


Figura 6.5 – Representación de 10 funciones Kernel

Como ya se ha comentado en el capítulo 6.2 disponemos de 5 tipos de vinos diferentes y cada medición es repetida 5 veces, por lo que estos valores son los que se deben establecer como número de ciclos del ensayo y número de variantes.

Un problema típico del tipo de ensayos que estamos tratando es la brusca variación de la intensidad en los extremos de las curvas voltamétricas cíclicas, alcanzo el mínimo en la parte izquierda y el máximo en la derecha. Estos valores carecen de sentido desde un punto de vista y no aportan una información útil a la hora de realizar el análisis, por lo que se cortan los extremos de las curvas eliminando los valores correspondientes. En nuestro decidimos eliminar un 10% de la curva por cada lado, quedándonos así con las zonas que más información nos ofrecen y evitando que los valores extremos nos alteren lo mínimo posible el resultado final.

Es frecuente que una de los dos partes del ciclo (curva anódica y curva catódica) nos aporte más información que la otra, normalmente debido a un mayor número de picos o una mayor repetitividad. Es por ello que en la configuración del análisis podemos seleccionar entre analizar sólo una de las dos partes o ambas, ajustando así el coste de cálculo a nuestras necesidades. En nuestro caso, la zona inferior de la curva (catódica) no presenta valores

característicos que resulten útiles para el análisis, por lo que nos quedaremos con la curva superior (anódica).

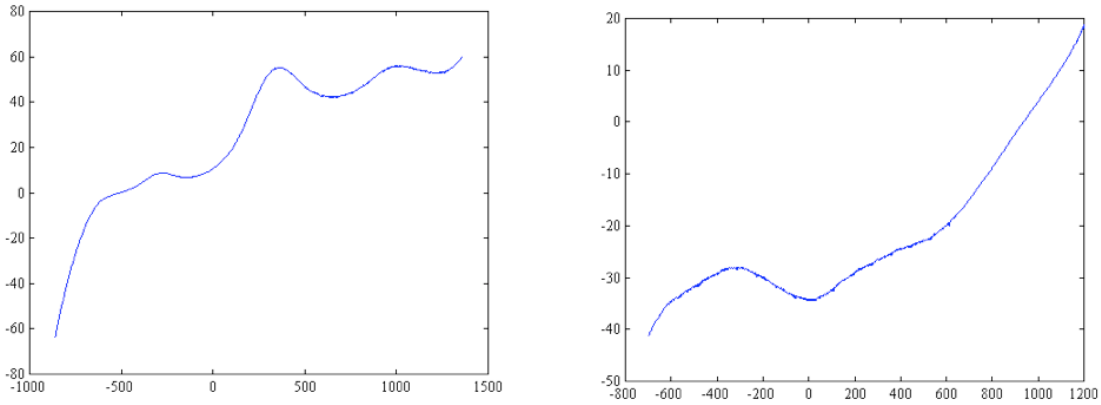


Figura 6.6 – Separación de la zona anódica (izquierda) y catódica (derecha) de la curva

Teniendo en cuenta todo lo anterior, la configuración escogida sería la siguiente:

- Número de Kernel: 10
- Número de ciclos del ensayo: 5
- Número de variantes: 5
- Fracción inicial: 0.1
- Fracción final: 0.9
- Ramas a analizar: 0

6.4. Cálculo

Una vez completado el proceso de configuración procedemos con la realización de los diferentes análisis.

6.4.1. Análisis en Componentes Principales

Procedemos a realizar el análisis en componentes principales seleccionando la opción 2 en el menú del proyecto.

En primer lugar se nos muestra en la ventana de comandos los valores del vector '*explained*'. Este contiene los resultados de la estimación de la contribución de cada componente al resultado principal. Este es el resultado:

```
explained:
4.301336e+01
2.634573e+01
1.914777e+01
1.008403e+01
1.019319e+00
2.148830e-01
7.792365e-02
3.446941e-02
2.140582e-02
8.641588e-03
6.705897e-03
5.920928e-03
4.679294e-03
4.569273e-03
3.410936e-03
2.574918e-03
1.977522e-03
1.423598e-03
1.204790e-03
```

Figura 6.7

Como se puede observar, la primera componente tiene un peso del 43.01% en el resultado final, la segunda un 26.35%, la tercera 19.15% y así sucesivamente.

A continuación el programa nos muestra la representación gráfica de los resultados:

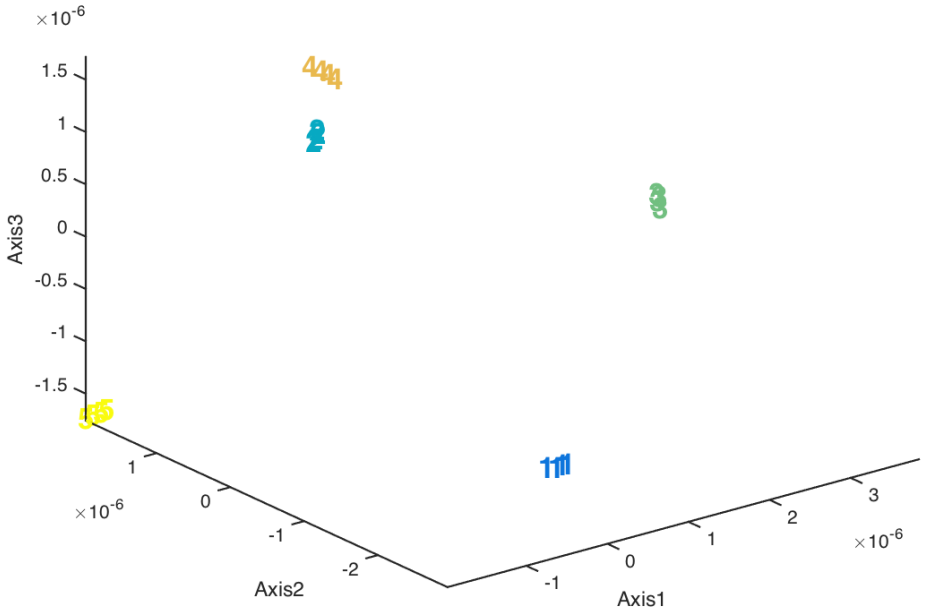


Figura 6.8

Atendiendo a lo representado en la gráfica devuelta, podemos distinguir cada uno de los 5 tipos de vinos, tal y como se muestra a continuación:

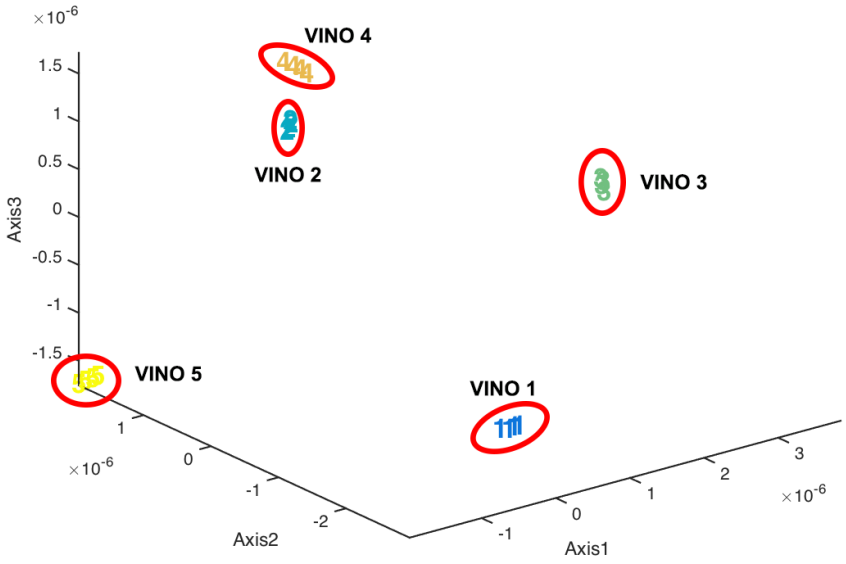


Figura 6.9

Para finalizar, se nos muestra el biplot:

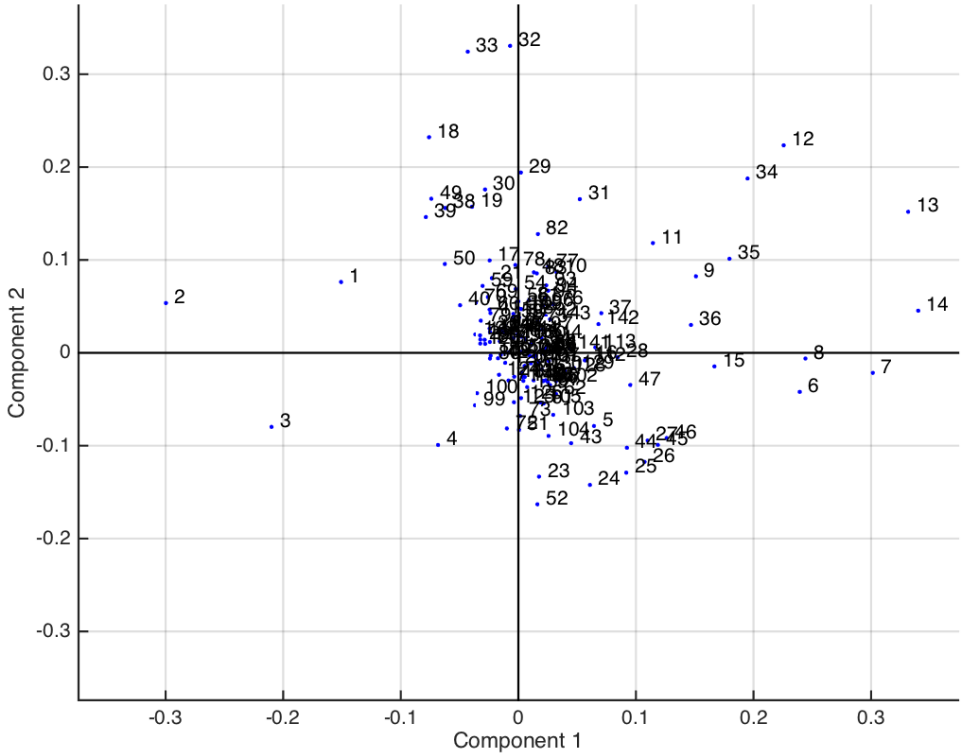


Figura 6.10

En él podemos visualizar el peso de cada medición en cada componente.

6.4.2. Análisis Parafac

Procedemos a realizar el análisis Parafac seleccionando la opción 3 en el menú del proyecto.

En primer lugar, el programa nos muestra los indicadores principales para cada número de componentes.

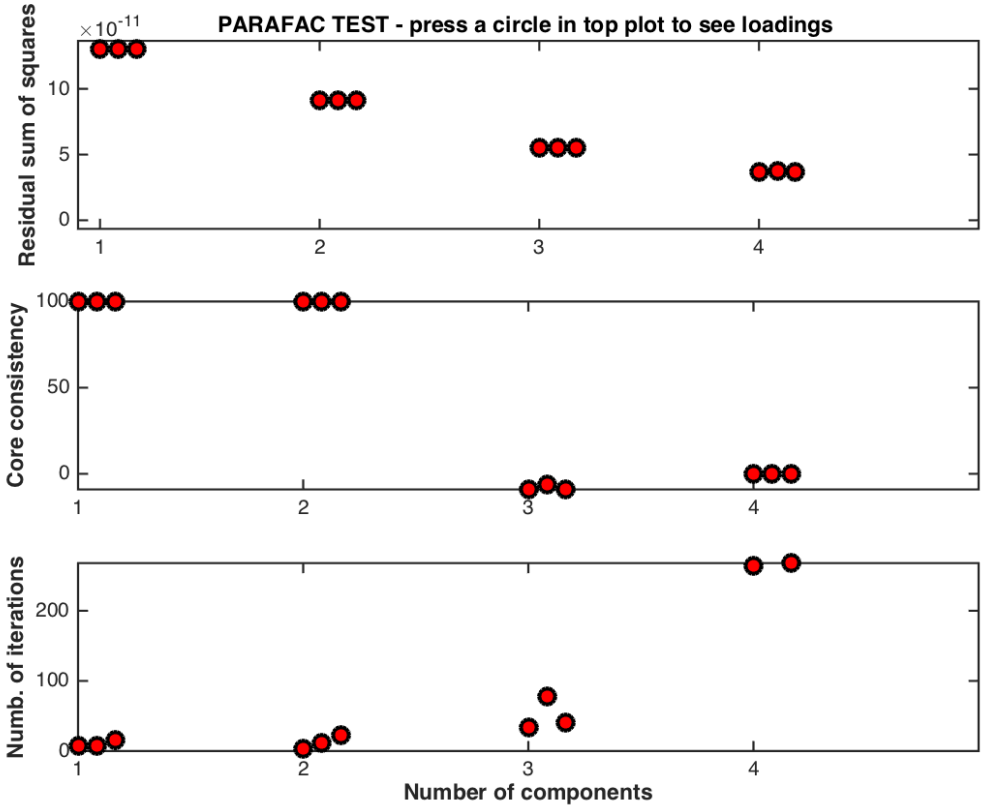


Figura 6.11

Como ya se comento en el capítulo 5.9.2, se debe escoger un número de componentes cuyo valor de 'core consistency' sea lo más próximo posible a 100. En este caso tenemos dos posibles opciones que cumplen el requisito: 1 y 2. Utilizar una única componente, dependiendo de los datos a analizar, puede que no sea suficiente para alcanzar resultados concluyentes por lo que decidimos trabajar con 2 componentes.

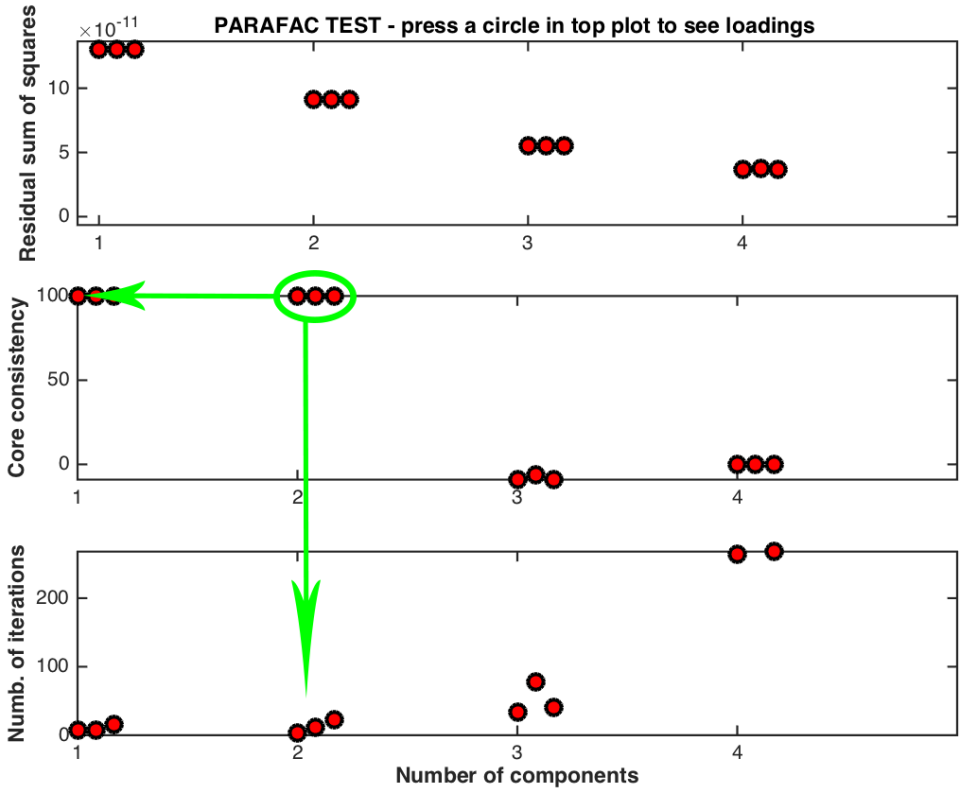


Figura 6.12

Introducimos el número de componentes:

```
Número de componentes: 2
```

Figura 6.13

Seguidamente se procede con el análisis, obteniendo los siguientes resultados:

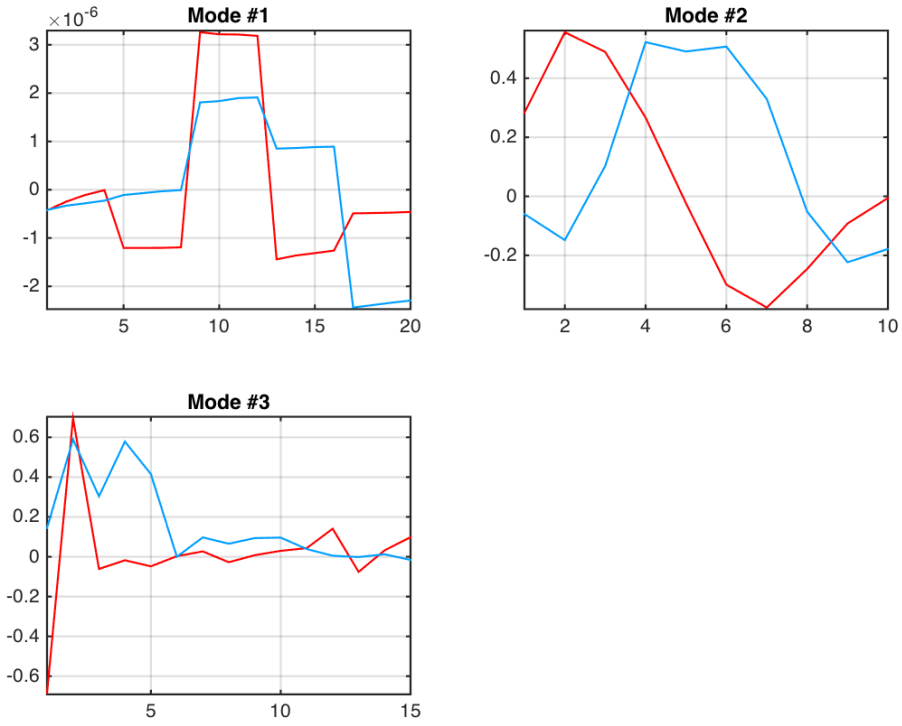


Figura 6.14

De estas gráficas podemos sacar varias conclusiones.

En primer lugar nos centramos en la gráfica 1 (Mode #1). El valor 20 del eje de las abscisas se corresponde con el número total de ensayos realizados. Como vemos en la Figura 5.23, para la primera y segunda componente podemos agrupar cinco grupos de resultados, con cuatro valores por grupo.

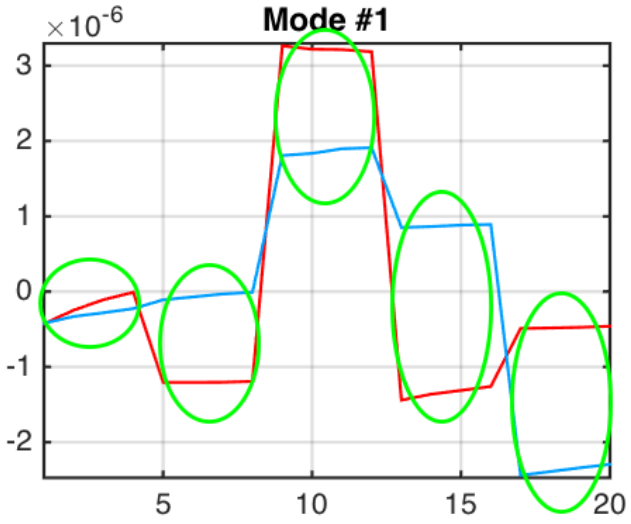


Figura 6.15

Cada uno de estos grupos se corresponde con un tipo de vino. El motivo de que cada grupo tenga cuatro valores se corresponde con el número de ciclos de ensayo para cada muestra. Como vemos, en este caso, podemos distinguir cada tipo de vino únicamente observando la primera componente (línea roja), aunque combinación con la segunda componente (línea azul) nos aporta mayor claridad.

En el gráfico 2 (Mode #2) el eje de las abscisas se corresponde con el número de kernels utilizado para simplificar la curva. Atendiendo a la primera componente (línea roja) podemos distinguir 2 zonas. La primera, de valores positivos, se debe al tramos de las curvas de ensayo en los que el potencial es negativo. La segunda, de valores negativos, a los tramos en los que el potencial es positivo. Atendiendo al valor absoluto de la componente observamos como es el primer tramo de potenciales negativos el que más información nos aporta al resultado final.

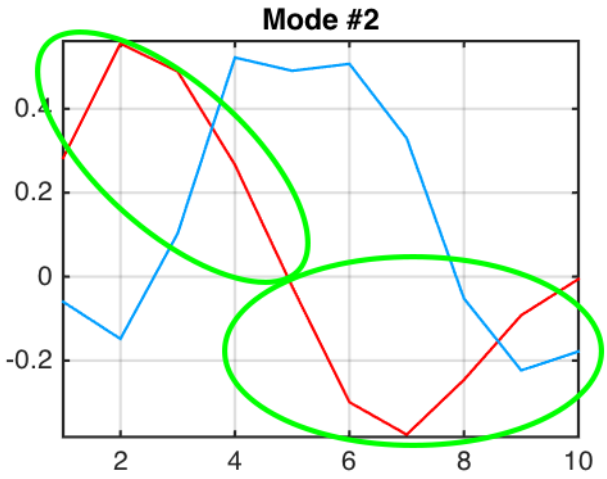


Figura 6.16

En el gráfico 3 (Mode #3) el eje de las abscisas se corresponde con el número de sensores utilizados en los ensayos. Los diferentes valores para cada componente nos permite distinguir el tipo de sensor utilizado.

Figura 6.17

Tras el análisis de los tres modos se nos muestra la gráfica de los resultados:

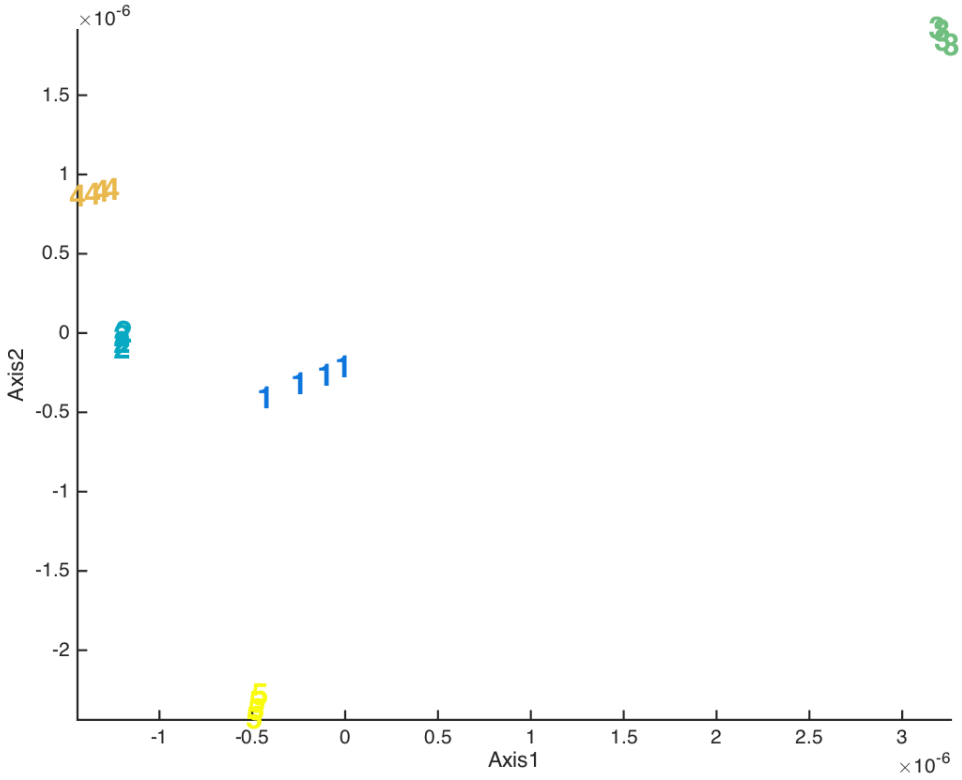


Figura 6.18 – Gráfica de la matriz A de Parafac

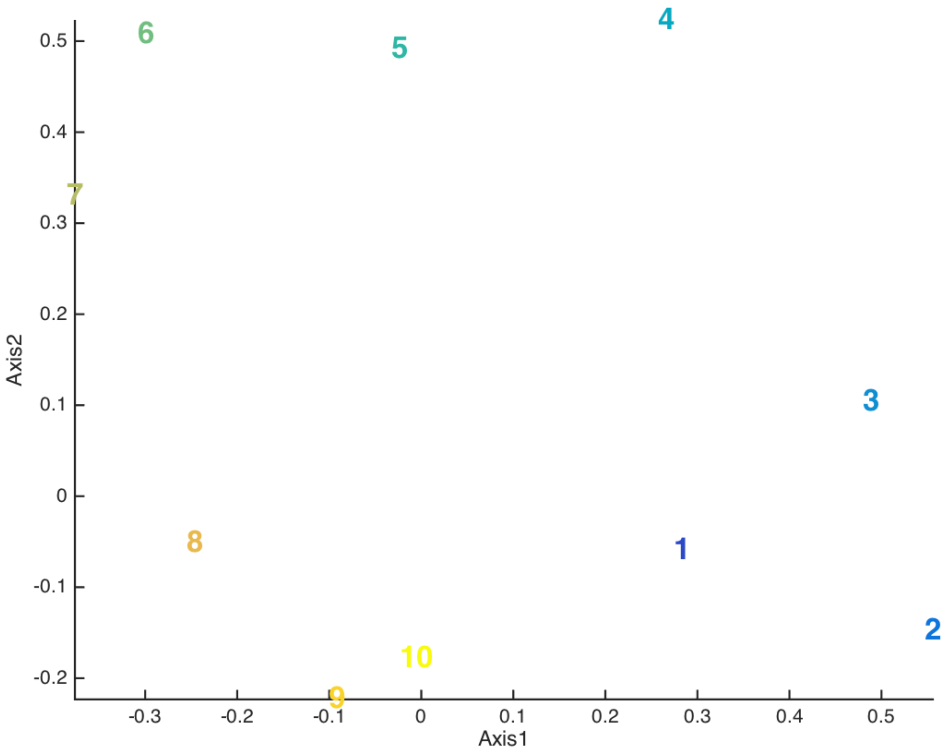


Figura 6.19 – Gráfica de la matriz B de Parafac

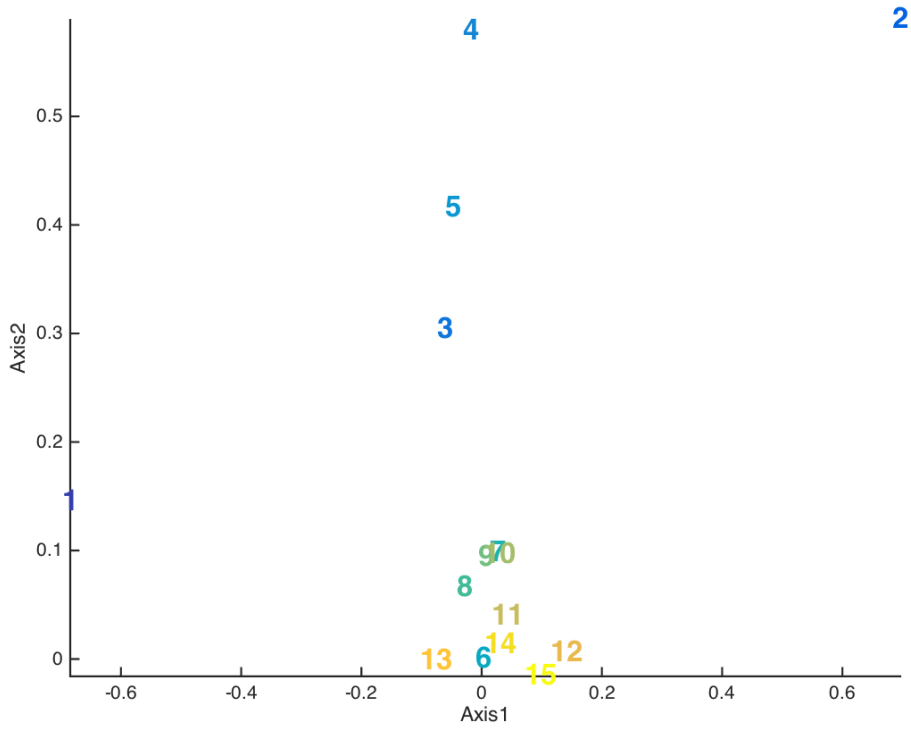


Figura 6.20 – Gráfica de la matriz C de Parafac

En la primera gráfica (Figura 6.18), correspondiente al modo 1 (matriz A), podemos distinguir todos los tipos de vinos:

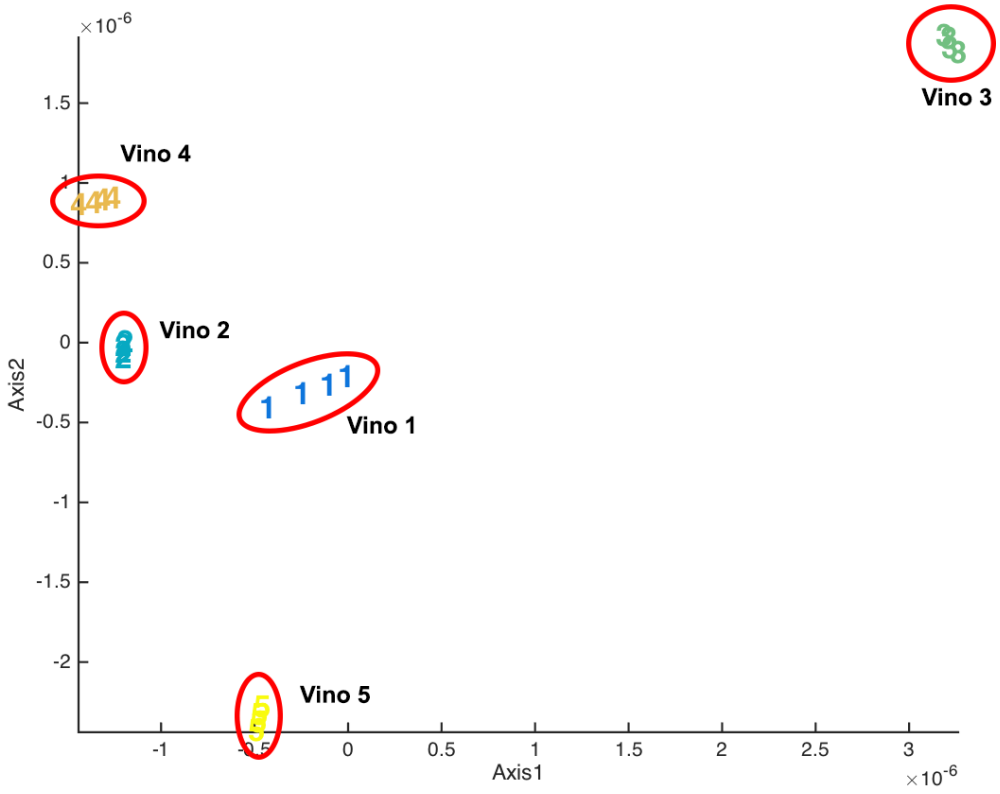


Figura 6.21

En la segunda gráfica (Figura 6.19), correspondiente al modo 2 (matriz B), observamos el siguiente fenómeno:

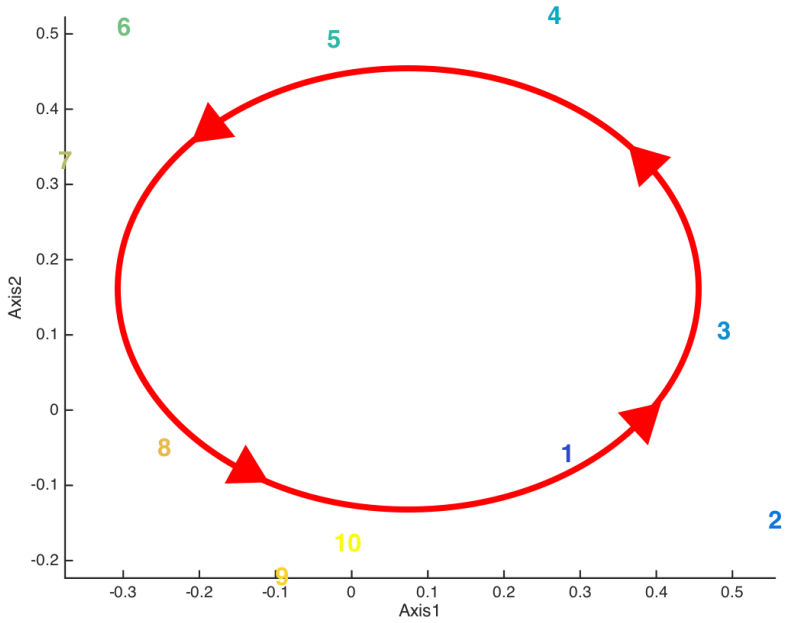


Figura 6.22

Cuando graficamos los resultados del modo 2, se observa que los puntos se posicionen en la periferia de la gráfica y además de forma ordenada, tal y como puede verse en la Figura 6.22. No se ha encontrado la explicación a este fenómeno.

Por último, en la gráfica del modo 3 podemos distinguir los distintos tipos de sensores. Vemos como los sensores enzimáticos se sitúan de manera dispersa mientras que los basados en ftalocianinas se agrupan en un entorno.

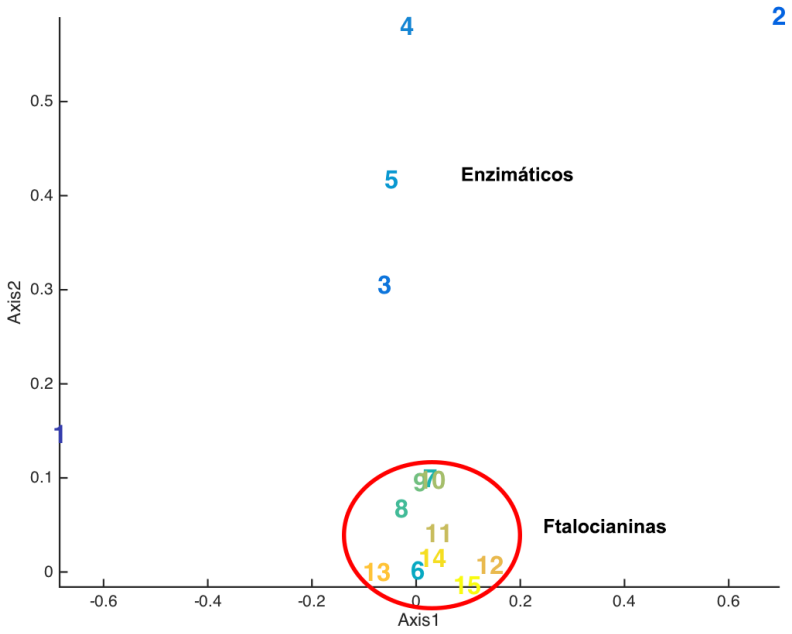
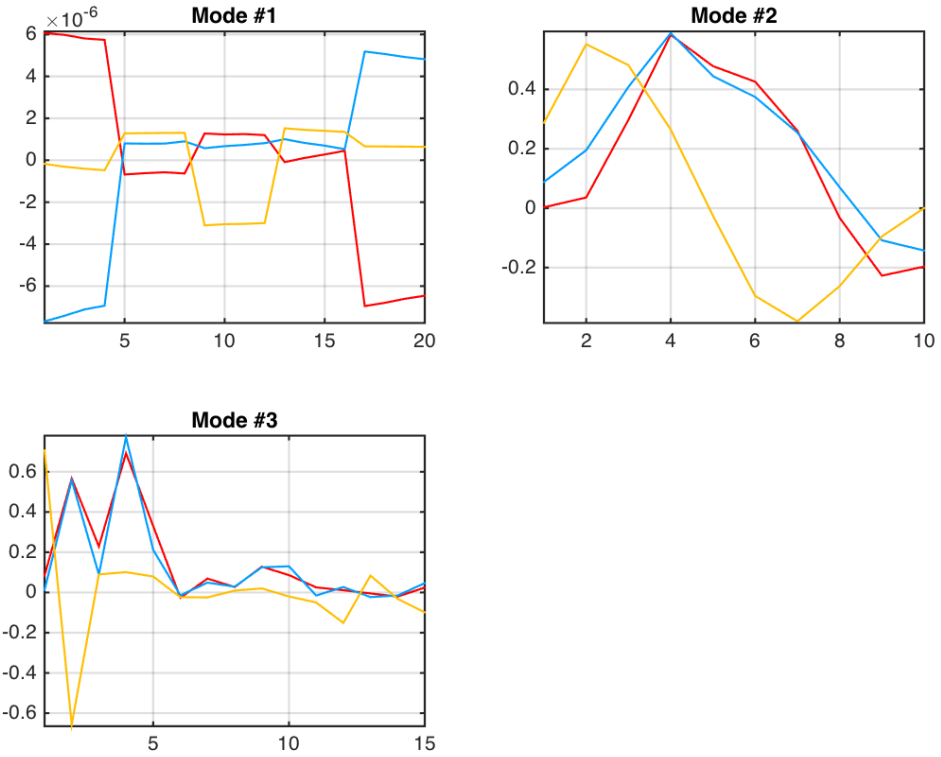


Figura 6.23

Completado el análisis se considera necesario destacar un punto crítico en la correcta utilización del programa. Para ello repetimos el análisis, pero en esta ocasión utilizamos 3 componentes. Los resultados obtenidos son los siguientes:




```
Component order (red,blue,yellow,green,lightblue,lightgreen,purple)
```

Figura 6.24

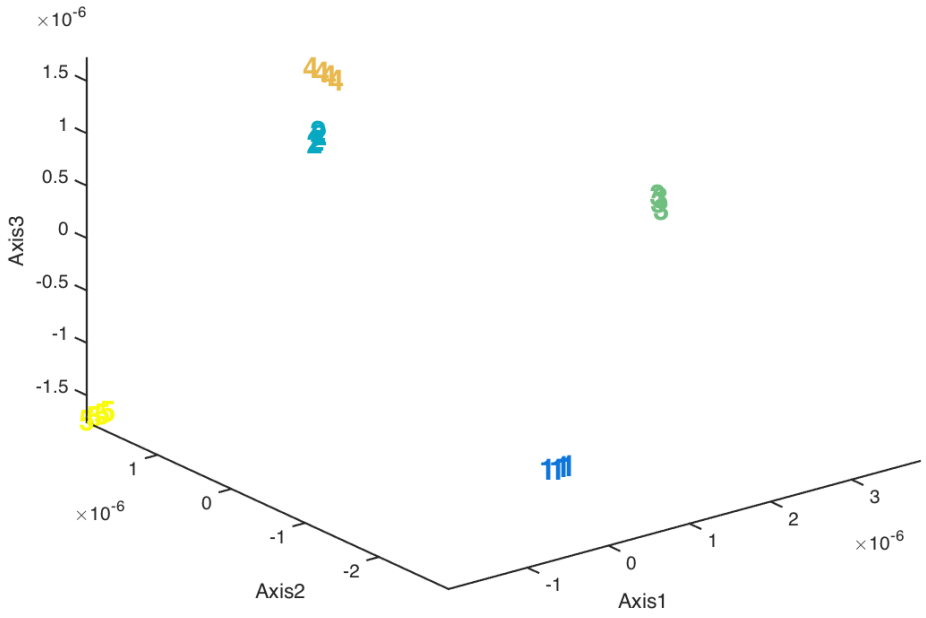


Figura 6.25

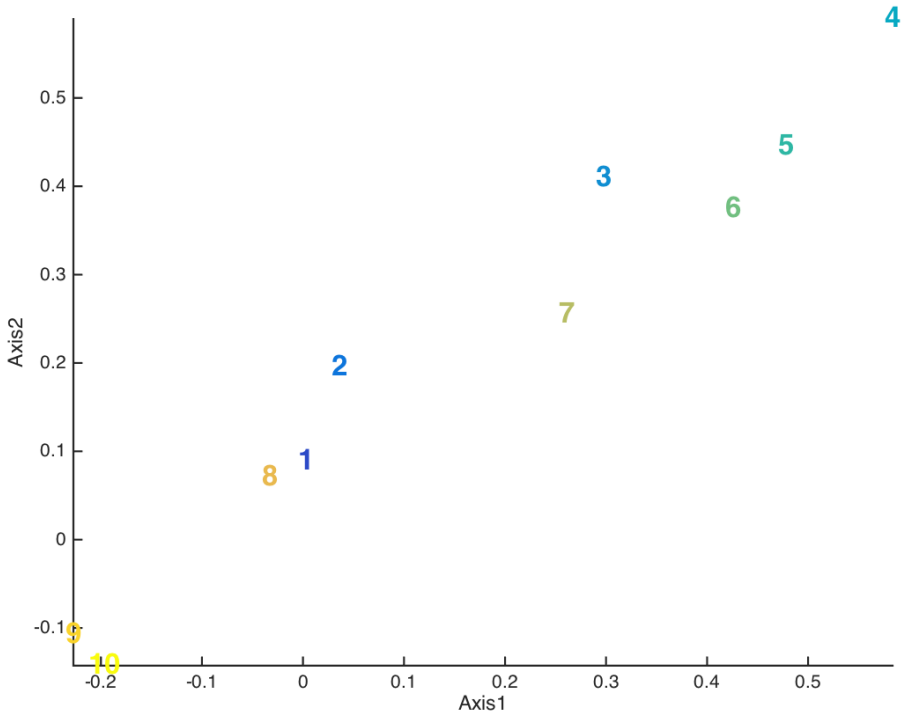


Figura 6.26

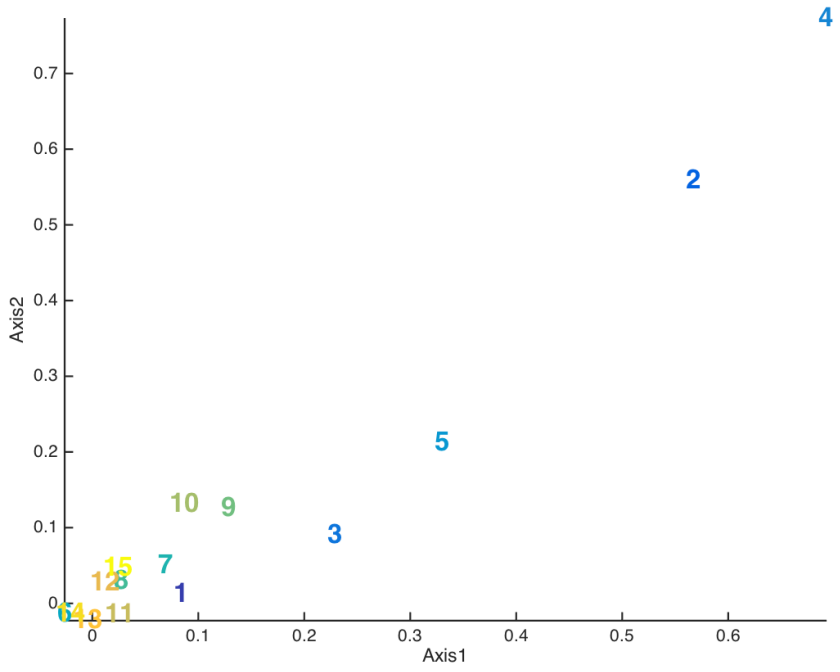


Figura 6.27

Como podemos ver, al igual que en caso anterior en el que usábamos 2 componentes, obtenemos resultados que nos permiten realizar las mismas interpretaciones. La gráfica más interesante es la mostrada en la Figura 6.25 que se corresponde con la representación gráfica del modo 1 (matriz A).

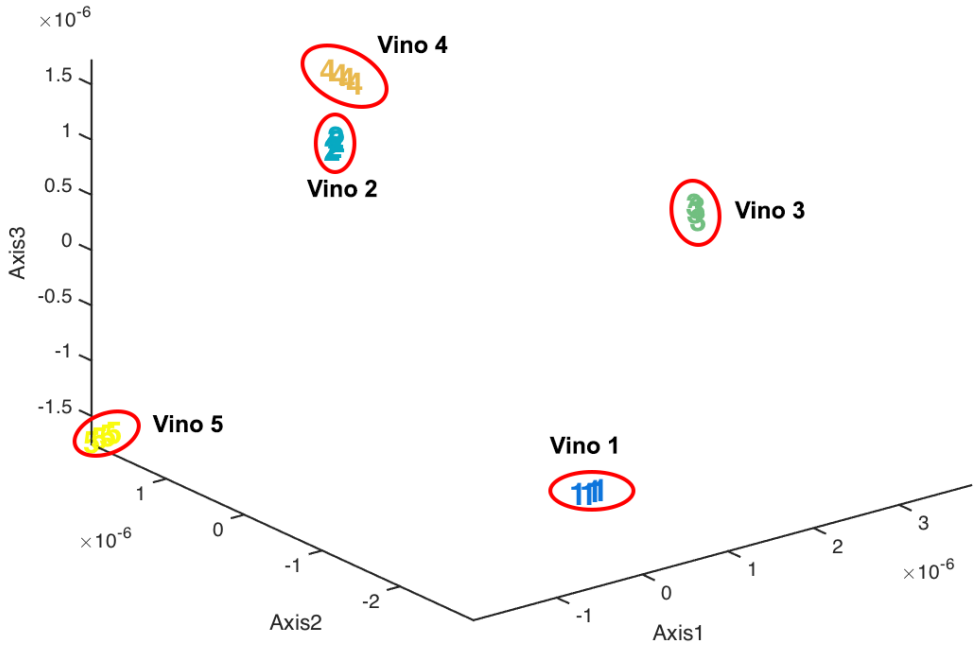


Figura 6.28

Esta gráfica (Figura 6.28) nos permite distinguir los distintos tipo de vino. Sin embargo, este valor se considera casuístico. El programa aquí desarrollado, al igual que la mayoría de software de cálculo y análisis, permite obtener representaciones gráficas de los resultados. Aunque a la vista parezcan correctos, siempre debemos comprobar la validez de los mismos.

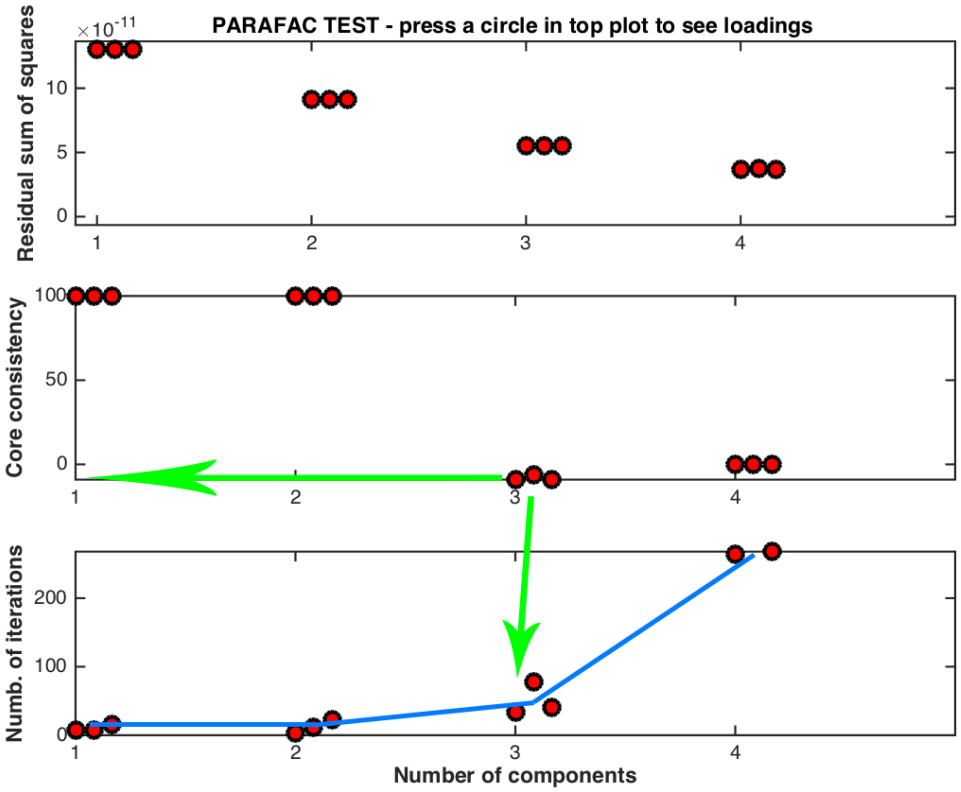


Figura 6.29

Como ya se dijo en el capítulo 5.9.2, para comprobar la validez del método debe atenderse a los resultados mostrados en la gráfica 'PARAFAC TEST' que el programa nos muestra antes de solicitarnos el número de componentes para realizar el análisis.

Para este caso hemos escogido trabajar con 3 componentes (Figura 6.29). Como puede observarse, el valor del 'Core consistency' está muy por debajo del 40% (prácticamente 0), lo cual supone que el modelo no es válido. Además, si observamos la evolución del número de iteraciones hasta la convergencia del algoritmo, vemos que a partir de 3 componentes empieza a aumentar bruscamente, lo cual es uno de los indicativos de que el método no está siendo eficaz en esas condiciones.

Como conclusión se vuelve a insistir en la importancia de los valores mostrados en 'PARAFAC TEST' y el especial cuidado que hay que tener a la hora de seleccionar el número de componentes para poder asegurar la validez del modelo y tratar de alcanzar resultados útiles y concluyentes.

6.4.3. Análisis Tucker

Procedemos a realizar el análisis Tucker seleccionando la opción 4 en el menú del proyecto.

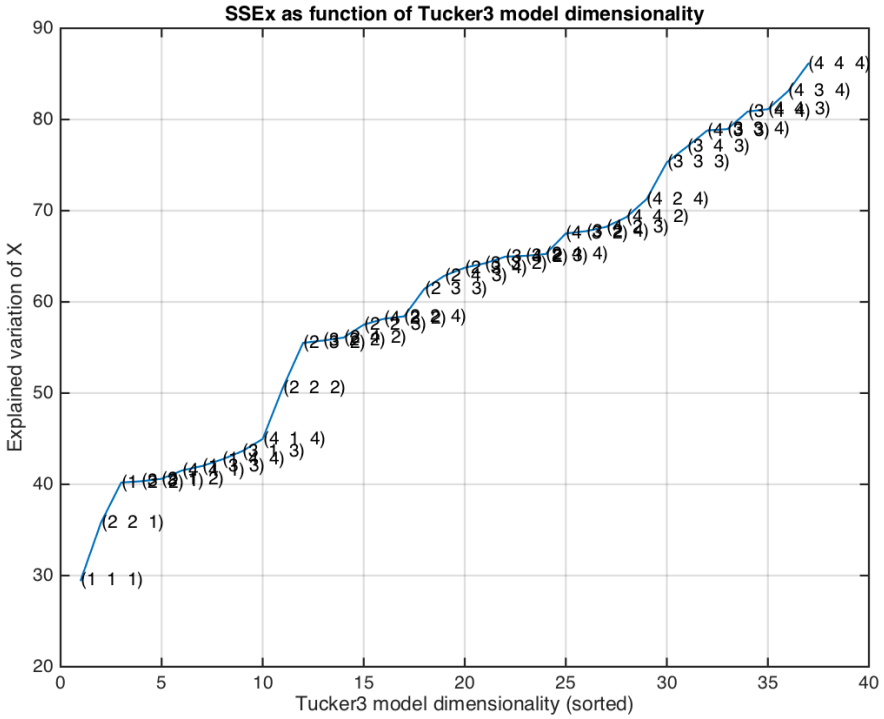


Figura 6.30

A partir de los resultados mostrados en la Figura 6.30 se decide definir el vector W de la siguiente manera:

```
W - Parámetro 1: 3
W - Parámetro 2: 3
W - Parámetro 3: 3
```

Figura 6.31 – Definición del vector W

Para tomar esta decisión nos basamos en primer lugar en el parámetro *'Explained variation of X'*. Este valor deberá ser próximo o superior al 80%. Dentro de los valores que cumplen esta condición se decide tomar [3 3 3] (Figura 6.32) por ser todos sus elementos iguales, lo cual nos permite realizar la simplificación del núcleo mediante los procesos de rotación. Además su elemento de mayor valor es inferior al del resto de combinaciones que cumplían la primera condición, lo cual reduce el tiempo y la potencia de cálculo.

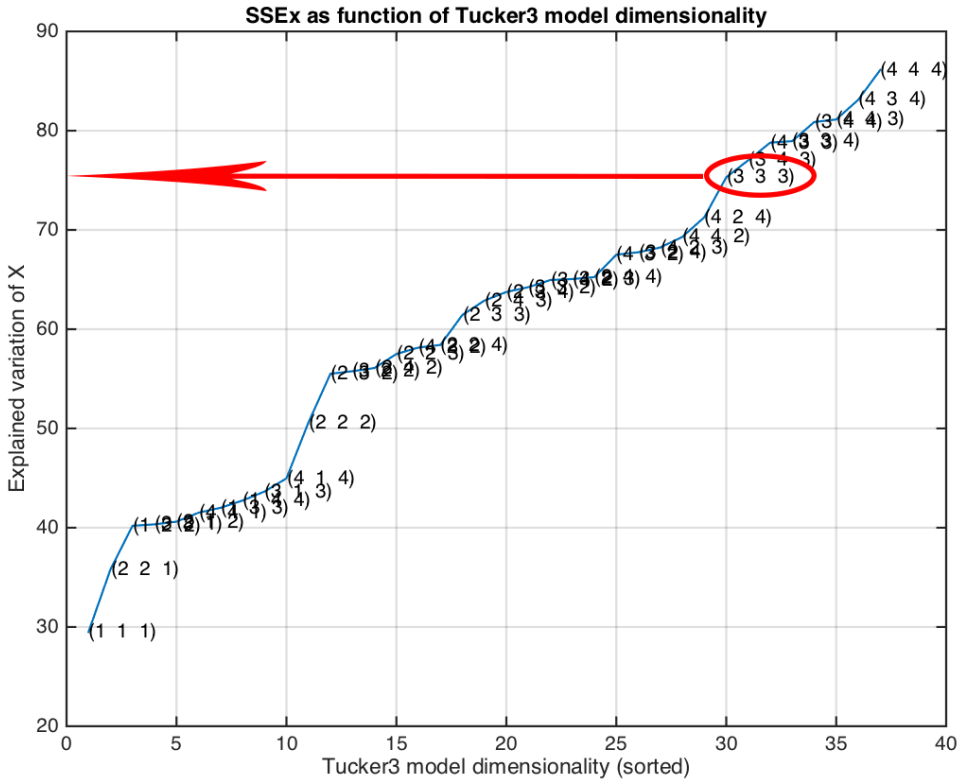


Figura 6.32

En el capítulo 5.9.3, cuando se explico el comportamiento de 'Explained variation' para Tucker3, se comento que en estas gráficas es común distinguir 2 tramos diferenciados por su pendiente. En nuestro caso no es así, pero esto no significa que el resultado no sea válido ya que, simplemente, puede deberse a que el cambio de pendiente se produzca para una combinación de componentes superior a las consideradas.

Tras introducir las componentes del vector W se procede con el análisis. En primer lugar el programa nos muestra los valores característicos del núcleo:

Col1: Number in list
Col2: Index to elements
Col3: Explained variation (sum of squares) of the core.
Col4: Core entry.
Col5: Sq. core entry.

1	(1, 1, 1)	41.76845%	-0.00001	0.00000
2	(1, 2, 2)	18.61617%	0.00000	0.00000
3	(2, 2, 2)	11.09770%	0.00000	0.00000
4	(2, 1, 1)	8.98220%	0.00000	0.00000

Como en este caso los tres elementos de W son iguales se procede a la rotación del núcleo:

Rotación del núcleo: diagonalización

Col1: Number in list
Col2: Index to elements
Col3: Explained variation (sum of squares) of the core.
Col4: Core entry.
Col5: Sq. Core entry.

1	(3, 3, 3)	35.64100%	0.00001	0.00000
2	(1, 1, 1)	15.86298%	-0.00000	0.00000
3	(2, 2, 2)	15.67883%	-0.00000	0.00000
4	(3, 2, 1)	8.08314%	-0.00000	0.00000
5	(2, 2, 1)	4.38566%	0.00000	0.00000
6	(1, 1, 2)	4.33107%	0.00000	0.00000
7	(2, 3, 2)	4.31505%	-0.00000	0.00000

Rotación del núcleo: optimización de los cuadrados

Col1: Number in list
Col2: Index to elements
Col3: Explained variation (sum of squares) of the core.
Col4: Core entry.
Col5: Sq. Core entry.

1	(3, 1, 2)	37.77418%	-0.00001	0.00000
2	(2, 3, 1)	23.24828%	0.00001	0.00000
3	(1, 2, 1)	12.65144%	0.00000	0.00000
4	(1, 3, 3)	6.74048%	-0.00000	0.00000
5	(1, 1, 1)	3.97523%	0.00000	0.00000
6	(2, 1, 3)	3.51071%	-0.00000	0.00000
7	(3, 3, 1)	3.37254%	0.00000	0.00000

A continuación se muestra por pantalla los resultados de las tres componentes para cada uno de los modos:

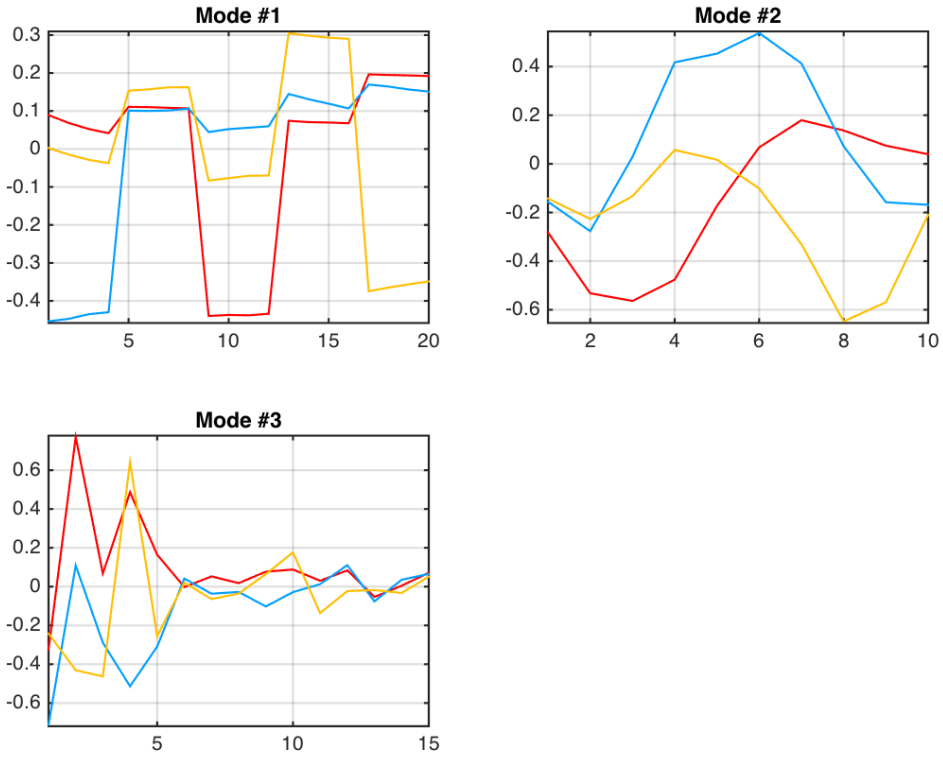


Figura 6.33

En la primera gráfica (Mode #1) el eje de abscisas se corresponde con los 20 ensayos realizados, esto es, 4 ensayos para cada una de las 5 clases de vino analizadas. Los valores de cada una de la componentes nos permite distinguir cada uno vinos:

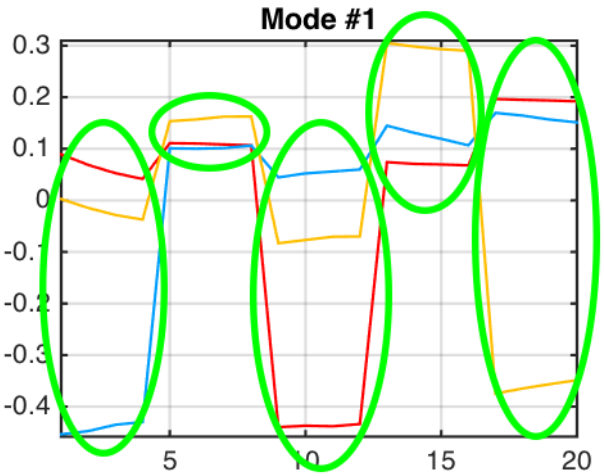


Figura 6.34

Para la segunda gráfica (Mode #2) en el eje de las abscisas tenemos 10 subdivisiones, correspondientes al número de kernels. Si observamos la primera componente (línea roja) podemos distinguir el tramo de la curva en que el potencial eléctrico es negativo y el tramo en el que es positivo. En este caso, al contrario de lo que sucedía en Parafac, los valores negativos de la componente se corresponden con las zonas de potencial negativo y los positivos con potencial positivo. Atendiendo al valor absoluto de las componentes observamos como las zonas de potencial negativo (primer tramo) aportan más información al resultado final.

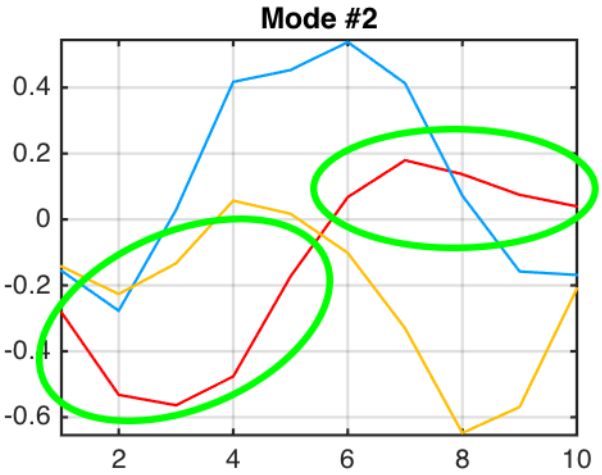


Figura 6.35

Por último, en el eje de abscisas de la tercera gráfica (Mode #3) se tienen 15 divisiones correspondientes a los 15 sensores utilizados. Observando al valor y comportamiento de las componentes podemos distinguir los distintos tipos de sensores.

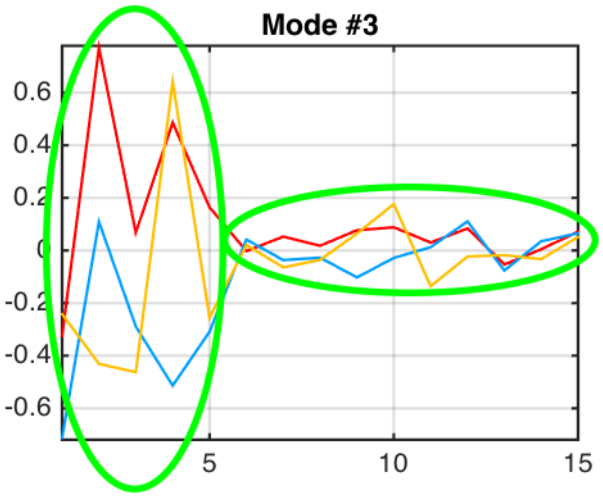


Figura 6.36

Para finalizar se obtiene la representación gráfica de cada uno de los tres modos:

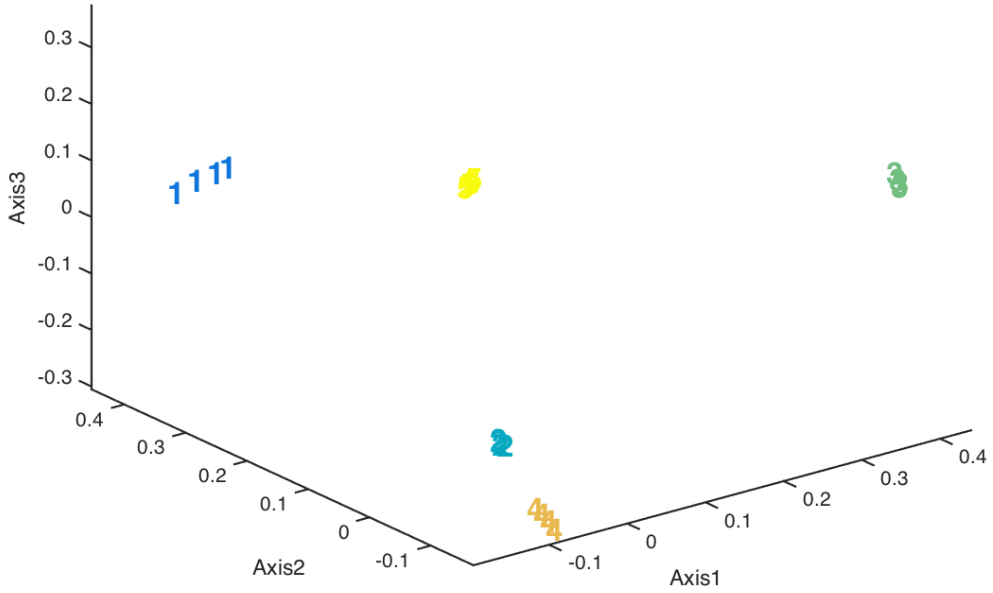


Figura 6.37 – Representación gráfica del modo 1 (matriz A)

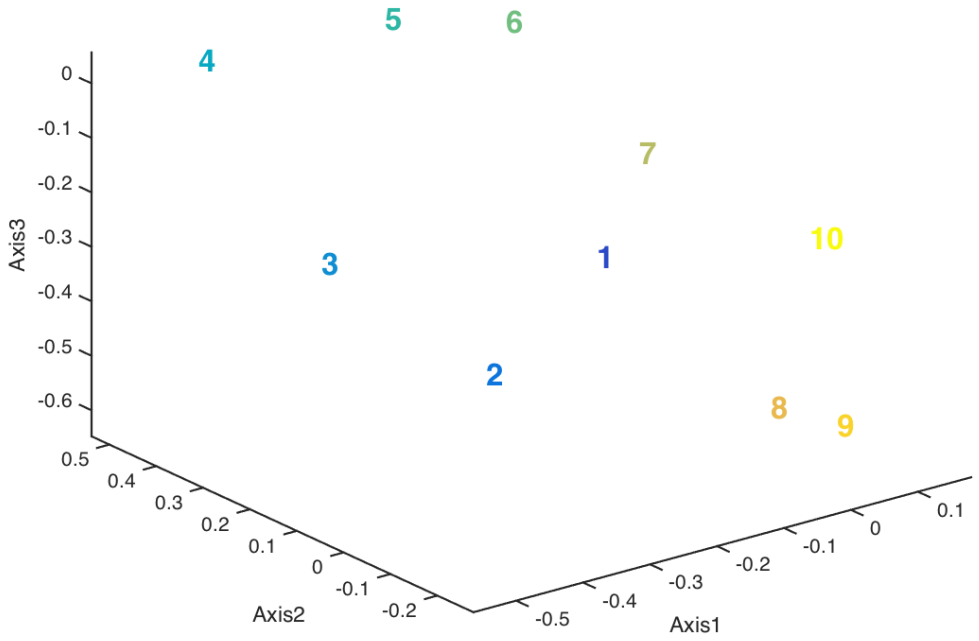


Figura 6.38 – Representación gráfica del modo 2 (matriz B)

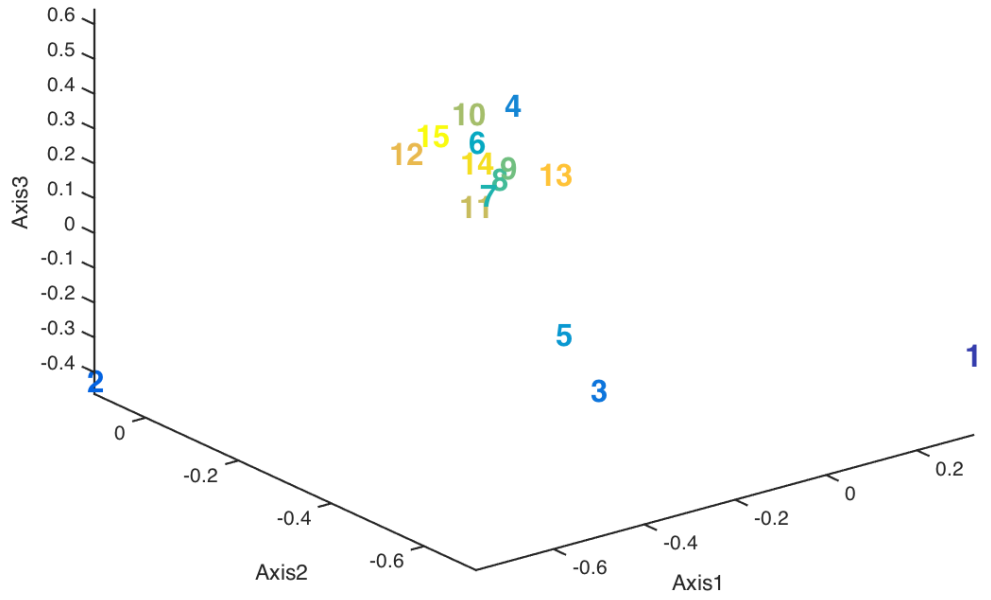


Figura 6.39 – Representación gráfica del modo 3 (matriz C)

En la primera gráfica podemos distinguir la distribución de todos los tipos de vinos:

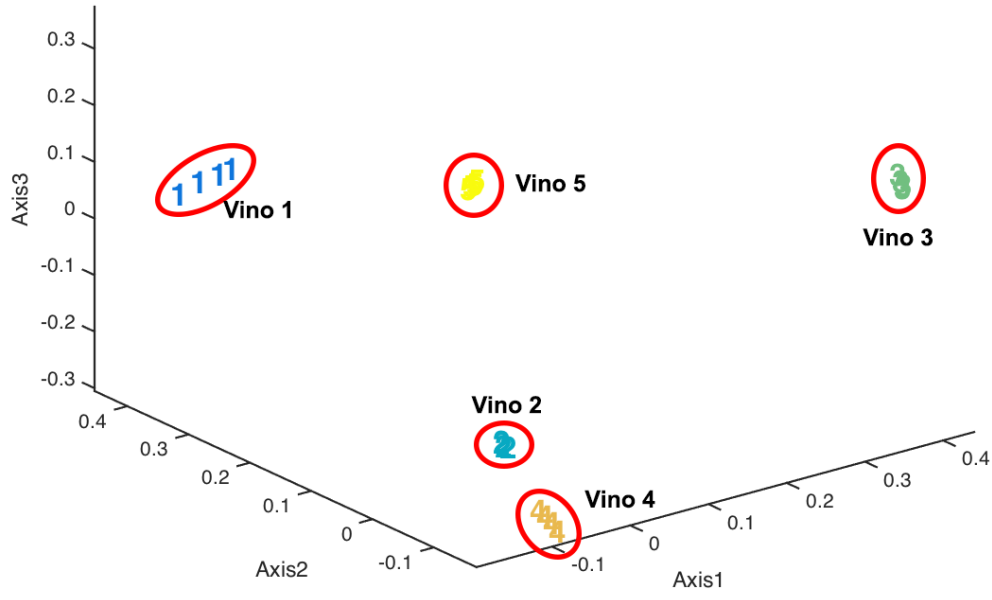


Figura 6.40

En el caso de la segunda gráfica (Modo 2) se observa el fenómeno de disposición de los datos que también se vio en Parafac. Para facilitar su visualización utilizaremos una versión bidimensional de la Figura 6.38 en la que se representan las dos primeras componentes:

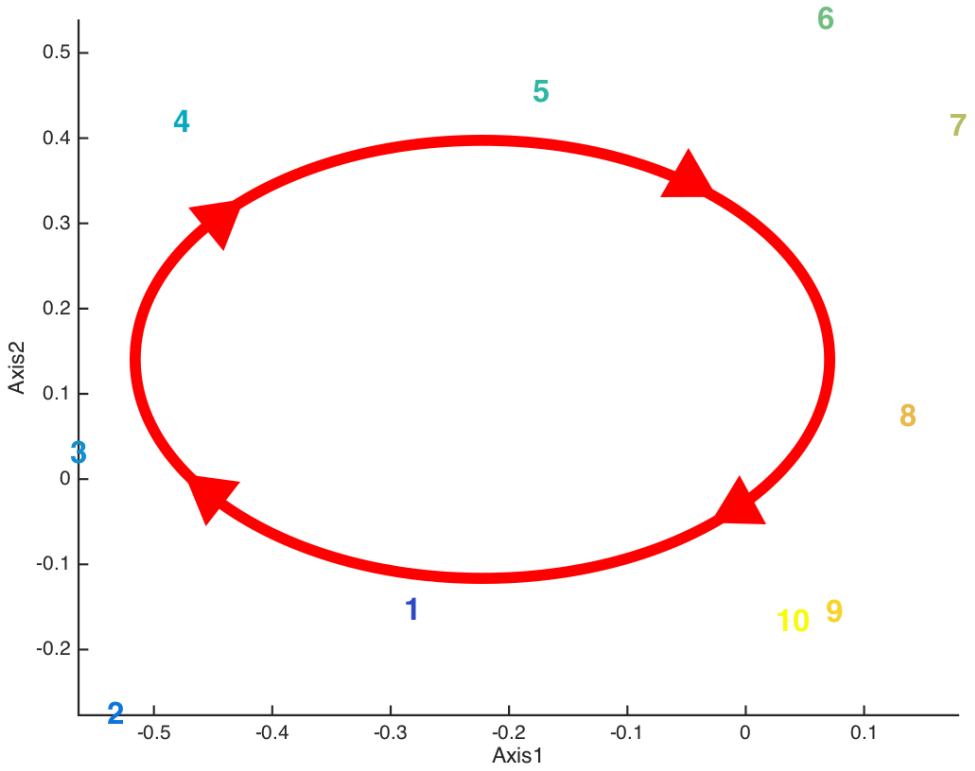


Figura 6.41

Los valores representados se disponen en la periferia de la gráfica y de manera ordenada. Se observa que el sentido del 'ciclo' de los valores es contrario al caso de Parafac.

Por último analizamos la gráfica correspondiente al modo 3. En ella se pueden distinguir los distintos tipos de sensores. Vemos como los sensores enzimáticos presentan valores dispersos mientras que los basados en ftalocianinas se concentran en un entorno. Para ver mejor esto, se muestra la versión bidimensional de la Figura 6.39, correspondiente al modo 3.

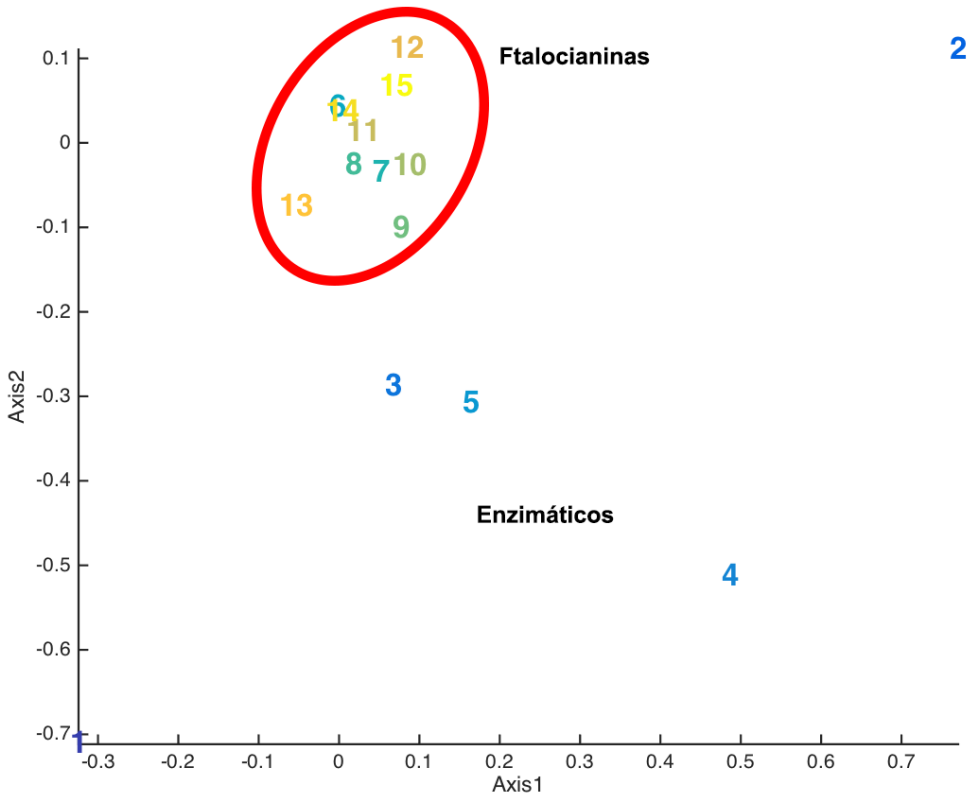


Figura 6.42

En este apartado se vuelve a destacar la importancia de la configuración del análisis para obtener resultados válidos y concluyentes. En el método que estamos utilizando, Tucker3, el valor crítico al que se debe prestar especial atención es el valor de '*Explained variation*' para la combinación de componentes escogida. Este valor lo muestra el programa al comenzar el proceso de análisis.

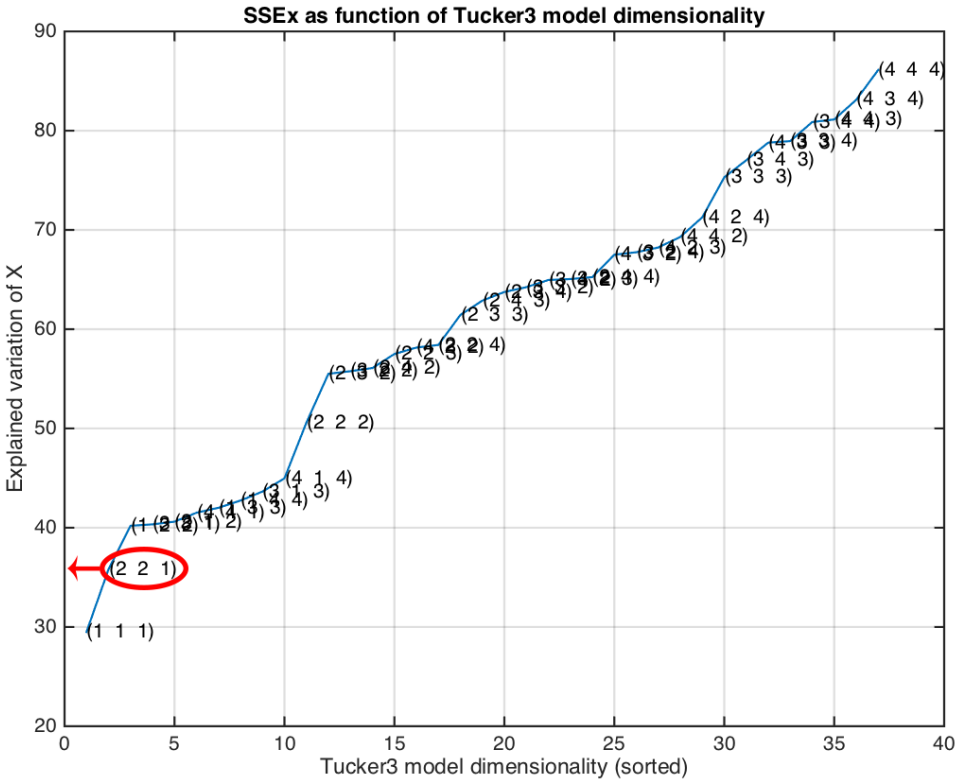


Figura 6.43

Para esta demostración definimos el vector W como [2 2 1]. Según los resultados mostrados en la Figura 6.43, esta combinación presenta un '*Explained variaton*' de un 36% aproximadamente. Como ya se vio en el capítulo 5.9.3, valores por debajo del 40% indican que el método no es válido. A pesar de esto el programa es capaz de alcanzar una serie de resultados tanto numéricos como gráficos.

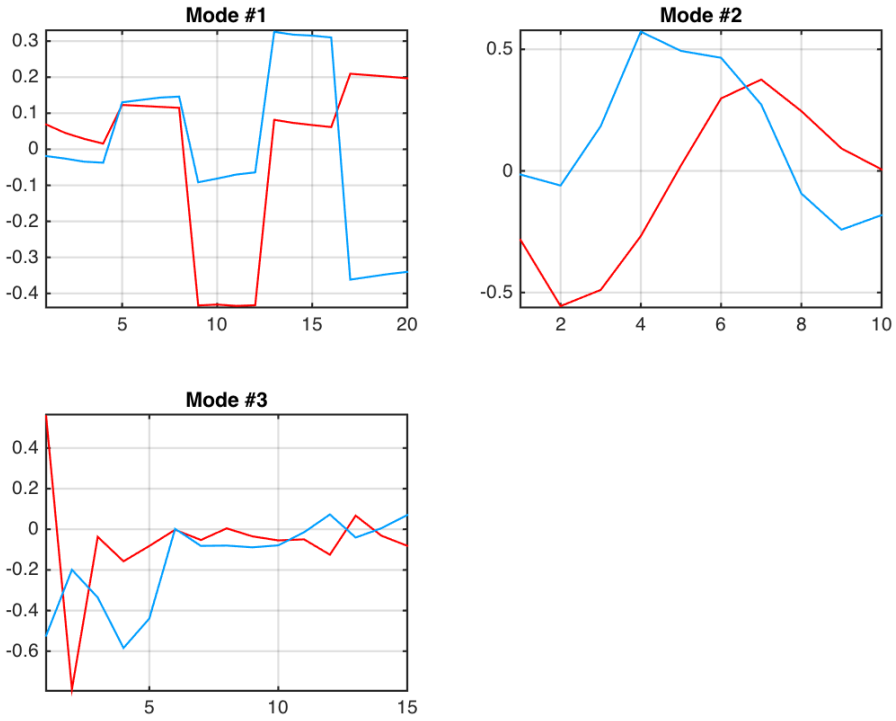


Figura 6.44

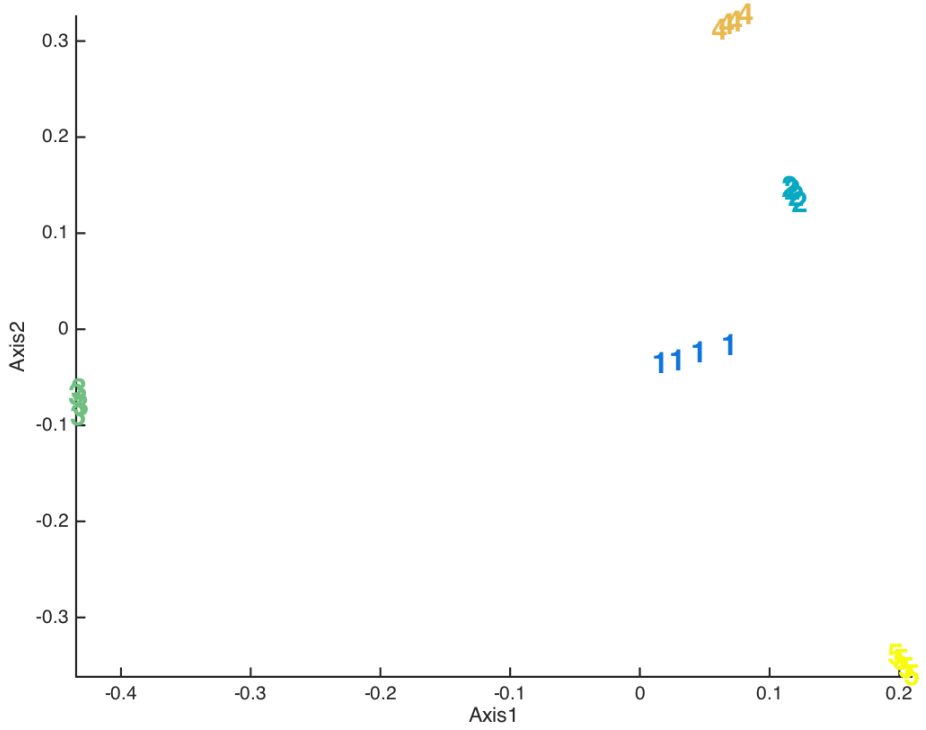


Figura 6.45

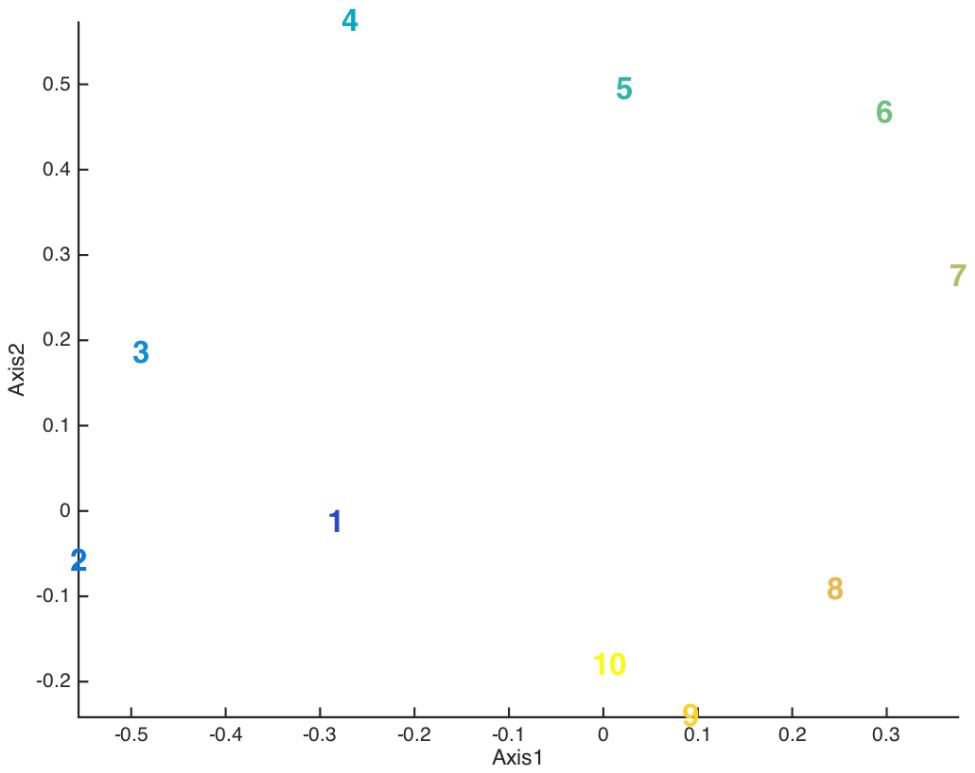


Figura 6.46

Observando la gráfica correspondiente al modo 1 podemos distinguir los distintos vino (Figura 6.47), sin embargo, ni en este ni otro caso se pueden considerar válidos los resultados y nos puede llevar a conclusiones erróneas.

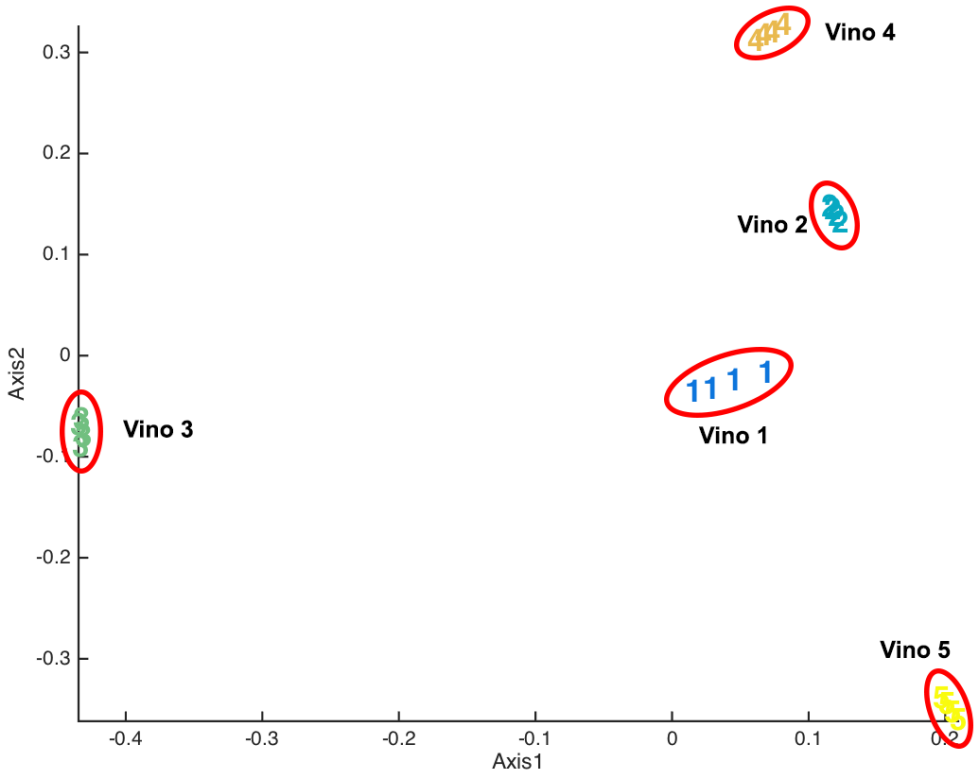
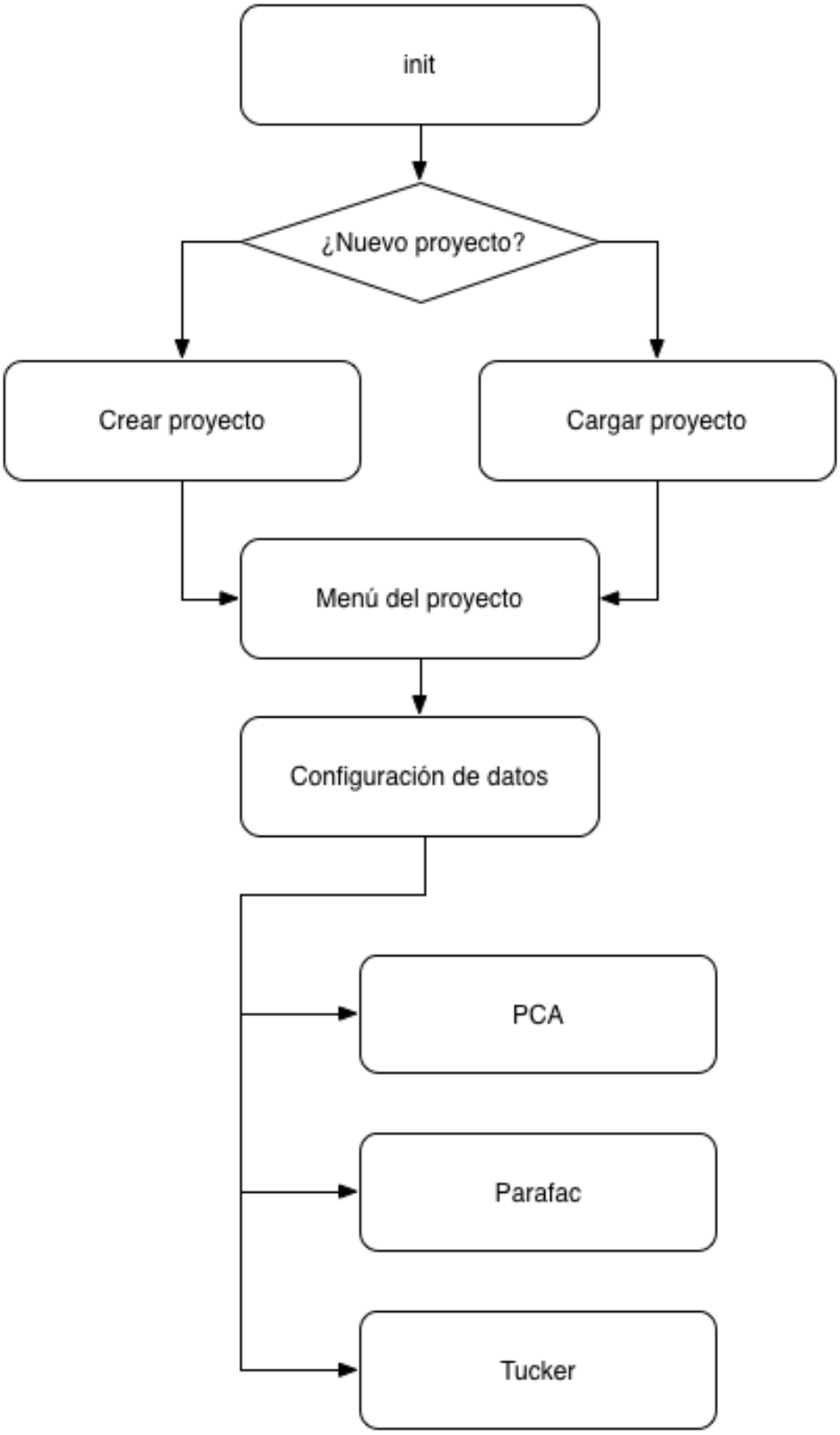


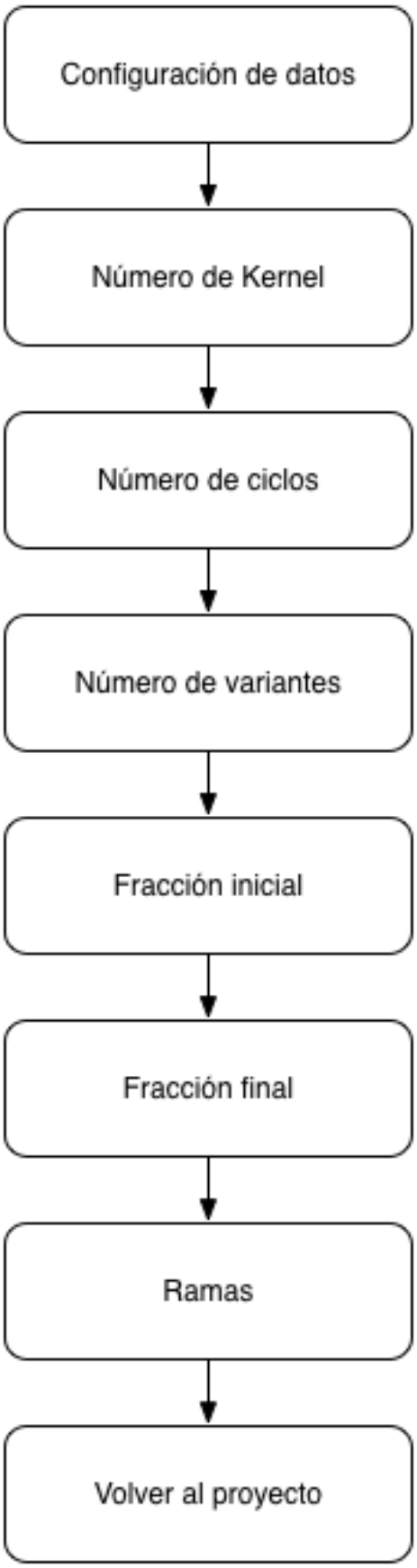
Figura 6.47

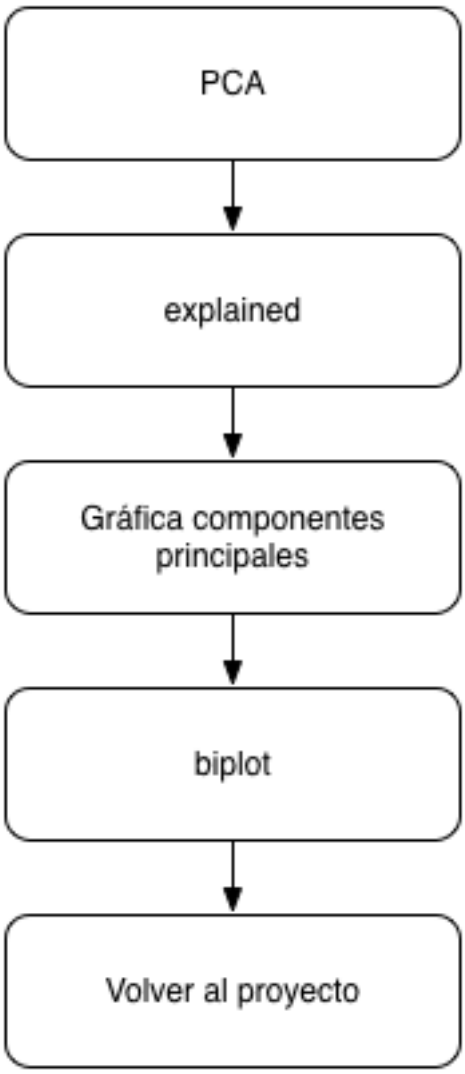
Terminamos este capítulo destacando de nuevo la importancia de configurar correctamente el análisis Tucker3, prestando especial atención a un parámetro crítico como es el '*Explained variation*' con el objetivo de alcanzar resultados válidos y concluyentes.

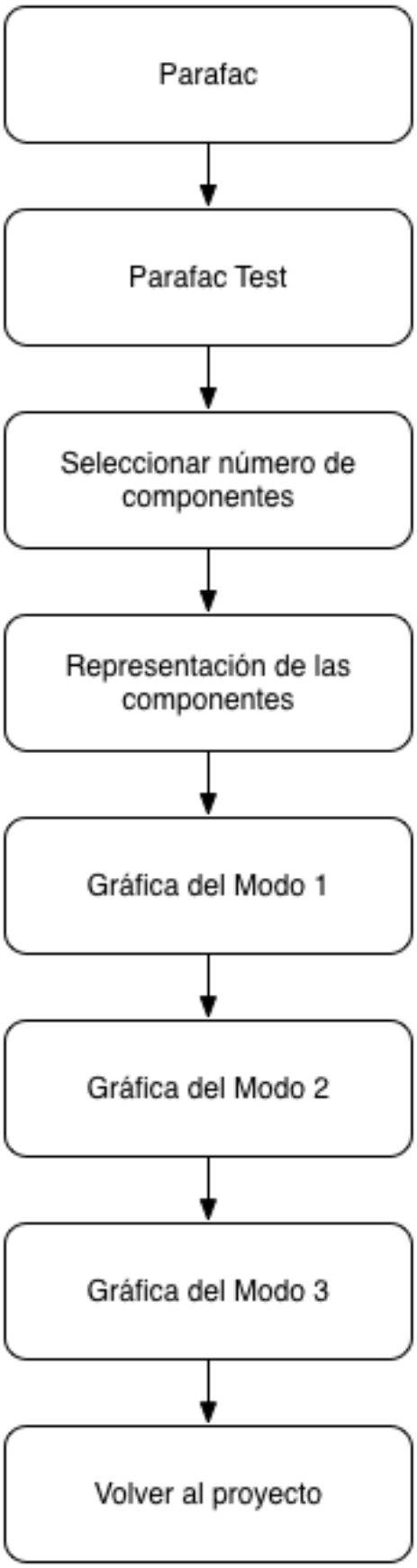
ANEXOS

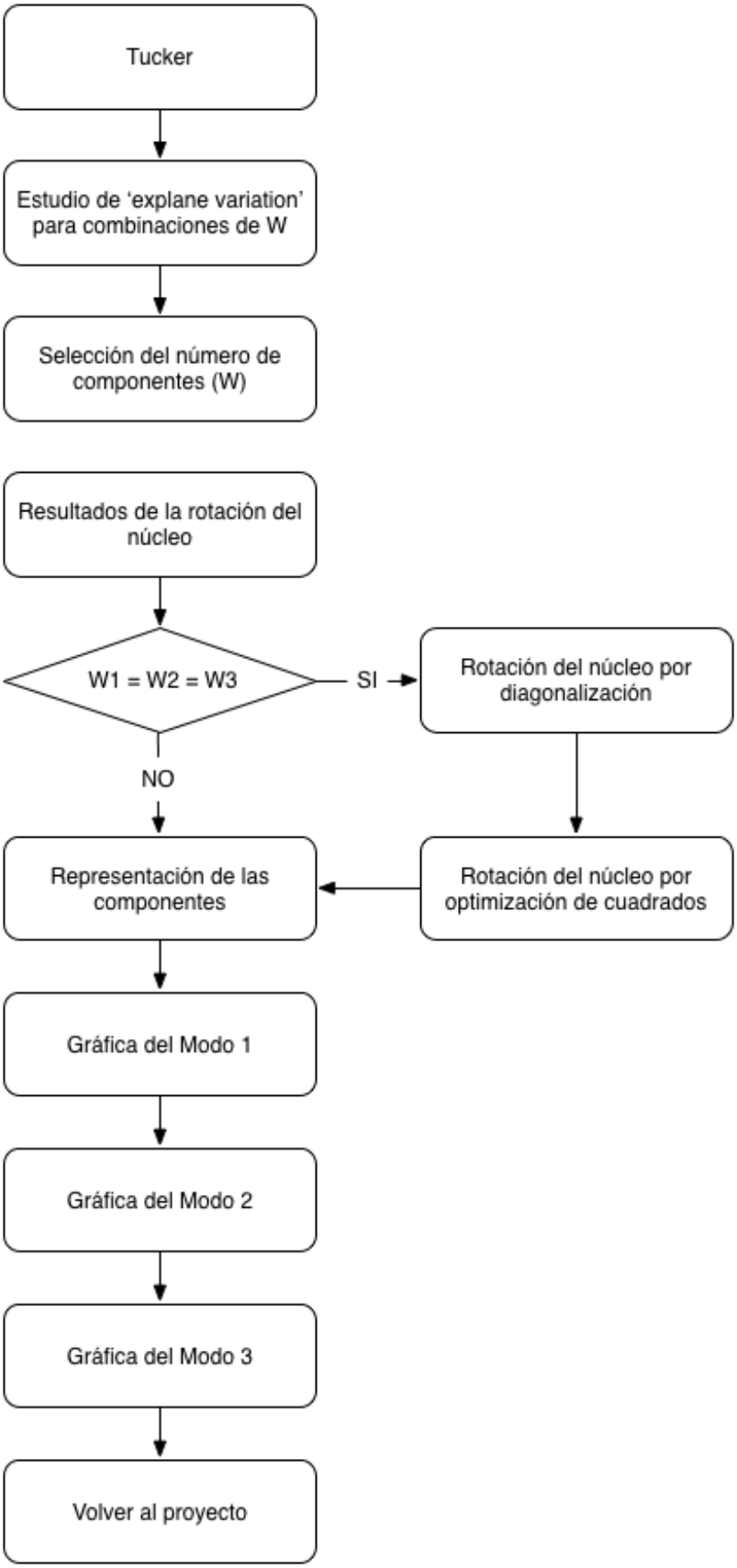
7. Anexo – Diagramas de flujo del programa




Configuración de datos
Número de Kernel
Número de ciclos
Número de variantes
Fracción inicial
Fracción final
Ramas
Volver al proyecto


PCA
explained
Gráfica componentes principales
biplot
Volver al proyecto


Parafac
Parafac Test
Seleccionar número de componentes
Representación de las componentes
Gráfica del Modo 1
Gráfica del Modo 2
Gráfica del Modo 3
Volver al proyecto


Tucker
Estudio de 'explane variation' para combinaciones de W
Selección del número de componentes (W)
Resultados de la rotación del núcleo
W1 = W2 = W3
SI
Rotación del núcleo por diagonalización
NO
Representación de las componentes
Rotación del núcleo por optimización de cuadrados
Gráfica del Modo 1
Gráfica del Modo 2
Gráfica del Modo 3
Volver al proyecto

8. Anexo – Funciones Principales de MatLab

calc_pca()

```
reset_uva;
fprintf('===== An·lisis PCA =====\n\n');
%process;
% PAC
X = zscore(result_2way);
% Principal Component Analysis
[coeff,score,latent,tsquared,explained,mu] = pca(X2);
%Mostrar variables
fprintf('explained:\n');
fprintf('%i\n',explained);
% Graficar
plotscat_uva(score,clases,0,3);
%Guardar la gr·fica
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_graf_pca.png'));
fprintf('\nPulsa una tecla para ver la contribuciÛn de cada
variable.\n');
pause;
%Lista de etiquetas
labels = cell(1,size(coeff,1));
for i=1:size(coeff,1)
    labels{i} = int2str(i);
end
biplot(coeff(:,2:3),'varlabels',labels,'LineStyle','none')
%Guardar la gr·fica
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_biplot_pca.png'));
post;
```

calc_parafac()

```
reset_uva;
fprintf('===== An·lisis Parafac =====\n\n');
% Centrar los datos
Cent = [1 0 0];
Scal = [0 0 0];
%Estandarizado de los datos
X_norm = nprocess(result_3way,Cent,Scal)
%d = size(X)
%pause;
% Buscar el n˙mero adecuado de componentes
[ssX,Corcondia,It] = pftest(3,X3,4);
clc;
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_test_parafac.png'));
fprintf('A continuaciÛn debes introducir el n˙mero de
componentes.\n');
fprintf('Se recomienda usar el n˙mero de componentes que cuyo
"core consistency" se acerque m·s a 100.\n\n');
%num_componentes es el n˙mero de componentes
%En vista a los resultados anteriores el ideal es 2
while 1
    num_componentes = input('N˙mero de componentes: ');
    if num_componentes>=1 && num_componentes<=4
        break
    end
end
close all;
% Parafac
fprintf('\nAn·lisis PARAFAC\n');
%InicializaciÛn al azar
Options = [];
Options(2) = 2;
%Aplicacion del metodo de Parafac
[Factors,it,err,corcondia] = parafac(X3,num_componentes);
% Separar matrices tipo cell
[A,B,C] = fac2let(Factors);
% Graficar
close all;
clc;
fprintf('\nMostrando los resultados del conjunto.\n')
%Guardar la gr·fica
plotfac(Factors)
%Guardar la gr·fica
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_order_parafac.png'))
;
fprintf('\nPulsa una tecla para mostrar los resultados de A.\n')
pause;
close all;
%Graficar los resultados del modo 1
plotscat_uva(A,clases,0,num_componentes)
%Guardar la gr·fica
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_graf_parafac.png'));
fprintf('Mostrando los resultados A.\n');
fprintf('\nPulsa una tecla para mostrar los resultados de B.\n')
pause;
close all;
%Graficar los resultados del modo 2
plotscat_uva(B,[1:size(B,1)],0,2);
%Guardar la gr·fica
```

```
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_grafB_parafac.png'))
;
fprintf('Mostrando los resultados B.\n');
fprintf('\nPulsa una tecla para mostrar los resultados de C.\n')
pause;
close all;
%Graficar los resultados del modo 3
plotscat_uva(C,[1:size(C,1)],0,2);
%Guardar la gr·fica
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_grafC_parafac.png'))
;
fprintf('Mostrando los resultados C.\n');
fprintf('\nPulsa una tecla para continuar.\n')
pause;
%Comparar los datos con los experimentales
%M = nmodel(Factors);
post;
```

calc_tucker()

```
reset_uva;
fprintf('===== An·lisis Tucker =====\n\n');
%---------------
%Se generan todas las posibles combinaciones
%v·lidas para el an·lisis de Tucker3
p=0;
for i=1:4,
    for j=1:4,
        for k=1:4,
            W=[i j k];
            if prod(W)/max(W)>=max(W),
                W;
                p=p+1;
                Z(p,1:3)=W;
                [Factors,G,SSE(p)]=tucker(X3,W);
            end
        end
    end
end
%Gr·ficamos los resultados obtenidos
[i j]=sort(SSE);plot(SSE(j));grid on;
for i=1:length(SSE(:));text(i,SSE(j(i)),['(' num2str(Z(j(i),:))
')']);end;
title('SSEx as function of Tucker3 model dimensionality');
xlabel('Tucker3 model dimensionality (sorted)');
ylabel('Explained variation of X');
clc;
%Guardar la gr·fica
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_dimen_tucker.png'));
%Solictud del vector W al usuario
fprintf('Seleccionar el n˙mero de componentes para cada modo.\n');
fprintf('Se recomienda escoger una combinaciÛn que se acerque o
supere el valor 80 de "explained variation:"\n\n');
W_parametro_1 = input('W - Par·metro 1: ');
W_parametro_2 = input('W - Par·metro 2: ');
W_parametro_3 = input('W - Par·metro 3: ');
%CreaciÛn de W
W=[W_parametro_1 W_parametro_1 W_parametro_1];
%AplicaciÛn del mÈtodo de Tucker3
[Factors,G,SSE]=tucker(X3,W);
clc;
fprintf('RotaciÛn del n˙cleo\n\n');
%Se simplifica el n˙cleo mediante su rotaciÛn
explcore(G,4);
fprintf('\n\nPulsa una tecla para continuar.\n');
pause;
%Si los elementos del vector W son iguales se puede
%rotar el n˙cleo por el mÈtodo de diagonalizaciÛn
%y por el de optimizaciÛn de cuadrados
if W_parametro_1 == W_parametro_2 && W_parametro_2 ==
W_parametro_3
    %Core rotation:
    [Factors,Go]=tucker(X3,W); %Make the Tucker model
    [A B C]=fac2let(Factors); %Convert to component matrices
    [Gd,Od1,Od2,Od3]=maxdia3(Go); %Rotate to optimum diagonality
    [Gv,Ov1,Ov2,Ov3]=maxvar3(Go); %Rotate to optimum variance-of-
squares
    %explcore(Go,7); %Inspect the unrotated solution
```

```
    clc;
    fprintf('RotaciÛn del n˙cleo: diagonalizaciÛn\n\n');
    %RotaciÛn del n˙cleo por diagonalizaciÛn
    explcore(Gd,7);
    fprintf('\nPulsa una tecla para continuar.\n');
    pause;
    clc;
    fprintf('RotaciÛn del n˙cleo: optimizaciÛn de los
cuadrados\n\n');
    %RotaciÛn del n˙cleo por optimizaciÛn de cuadrados
    explcore(Gv,7);
    fprintf('\nPulsa una tecla para continuar.\n');
    pause;
    %Reshape X and G to unfolded matrices
    Xunf = reshape(X3,size(X3,1),size(X3,2)*size(X3,3));
    Gounf = reshape(Go,size(Go,1),size(Go,2)*size(Go,3));
    Gdunf = reshape(Gd,size(Go,1),size(Go,2)*size(Go,3));
    Gvunf = reshape(Gv,size(Go,1),size(Go,2)*size(Go,3));
    sum(sum( (Xunf - A*Gounf*kron(C',B')).^2 )); %Error
    sum(sum( (Xunf - (A*Od1)*Gdunf*kron((C*Od3)',(B*Od2)')).^2 ));
%Error
    sum(sum( (Xunf - (A*Ov1)*Gvunf*kron((C*Ov3)',(B*Ov2)')).^2 ));
%Error
end
close all;
%Desplegar los 3 modos
[A B C]=fac2let(Factors);
%Gr·fica de componentes
plotfac(Factors);
clc;
%Guardar gr·fica
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_comp_tucker.png'));
fprintf('Pulsa una tecla para mostrar los resultados de A.\n');
pause;
close all;
%Graficar el modo 1 seg˙n la dimensiÛn de A
if W_parametro_1 >= 3
    plotscat_uva(A,clases,0,3);
else
    plotscat_uva(A,clases,0,2);
end
%Guardar gr·fica
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_graf_tucker.png'));
fprintf('Mostrando los resultados A.\n');
fprintf('\nPulsa una tecla para mostrar los resultados de B.\n');
pause;
close all;
%Graficar el modo 2 seg˙n la dimensiÛn de B
if W_parametro_2 >= 3
    plotscat_uva(B,[1:size(B,1)],0,3);
else
    plotscat_uva(B,[1:size(B,1)],0,2);
end
%Guardar gr·fica
saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_grafB_tucker.png'));
fprintf('Mostrando los resultados B.\n');
if W_parametro_3 ~= 1
    fprintf('\nPulsa una tecla para mostrar los resultados de
B.\n');
    pause;
```

```
    close all;
    %Graficar el modo 3 seg˙n la dimensiÛn de C
    if W_parametro_3 >= 3
        plotscat_uva(C,[1:size(C,1)],0,3);
    else
        plotscat_uva(C,[1:size(C,1)],0,2);
    end
    %Guardar gr·fica

saveas(gcf,strcat(pwd,'/TFG/data/',proyecto,'_grafC_tucker.png'));
    fprintf('Mostrando los resultados C.\n');
end
pause;
post;
```

data()

```
reset_uva;
fprintf('=== ConfiguraciÛn de datos ===\n\n');

%Se solicitan los datos para el procesamiento de los datos
ntrozos = input('N˙mero de Kernel: ');
ciclos = input('N˙mero de ciclos del ensayo: ');
Nmostos = input('N˙mero variantes: ');
frac_I = input('FracciÛn inicial: ');
frac_F = input('FracciÛn final: ');
rama = input('Ramas a analizar: ');

%El n˙mero m·ximo de ramas de oxidaciÛn debe ser igual al n˙mero
de ciclos - 1
Nrepe = ciclos - 1;

%Se procesan los datos.
% Cargar los datos
load(strcat(PathName,FileName));

% Generar la matriz de datos
[result_2way, result_3way, clases,labsens] =
genera_mat_v3(vendimia,ntrozos,Nmostos,Nrepe,frac_I,frac_F,rama);

% Normalizar result_3way con zscore
X_norm = zscore(result_3way);

% EstandarizaciÛn mediante nprocess
X2 = nprocess(result_2way,[1 0],[0 0]);
X3 = nprocess(result_3way,[1 0 0],[0 0 0]);

%Guardar datos
save_proy;

%Se vuelve al men˙
menu_pro;
```

genera_mat_v3()

```
Function
[result_2way,result_3way,clases,labsens]=genera_mat_v3(vinos,ntroz
os,Nmostos,Nrepe,fr_I,fr_F,rama)
% Tomo los Nrepe ultimos ciclos. Es decir Nrepe son las
repeticiones que
% voy a obtener. Se cumple que Nrepe=Nciclos-1
% ntrozos el n˙mero de kernels a obtener
% fr_I y fr_F es la fracciÛn de la rama que tomo para los kernel,
fr_I
% indica el punto (en tanto por uno) donde tomo el punto inicial y
fr_F el
% punto donde tomo el punto final. Si quiero usar toda la rama
pondrÈ
% fr_I=0 y fr_F=1
%
% rama decide si se usa oxidaciÛn, reducciÛn o ambas
% si    = 0 , oxidaciÛn
% si    = 1 , todo
% si    = 2 , reducciÛn

[f,~]=size(vinos.Listafiles);
PyM_red.Listafiles=vinos.Listafiles;
Nsensores=f/Nmostos;

% Nrepe es el n˙mero de repeticiones de cada mosto
% Reduce los datos CV al Nrepe y quita 3/4 del primer ciclo
% Homogeiniza unidades --> todo en mV y microA
% TambiÈn quito el ramal de reducciÛn y quito los 100mV primeros
de los
% extremos
% averigua n˙mero de ciclos por CV
for i=1:f
    var=vinos.(vinos.Listafiles{i});
    var(:,2)=zscore(var(:,2));  % normalizo
%     var(:,2)=normc(var(:,2));
    [fv,~]=size(var);

    if var(10,1) > 10 % se ha hecho con potenciostato viejo (en mV
y microA/cm2)
        var(:,2)=var(:,2).*0.01*1e-6;
    else
        var(:,1)=var(:,1).*1000;
    end
    % me aseguro que en la var(:,1), es decir, en la variable
independiente
    % no se repite un mismo valor. Si ocurre por alg˙n error en la
toma de
    % datos, el segundo repetido se modifica ligeramente
    viejo=var(1,1);
    for jj=2:fv-1
        if var(jj,1)==viejo
            if (var(jj+1,1)-var(jj,1))>0
                var(jj,1)=viejo+0.001;
            else
                var(jj,1)=viejo-0.001;
            end
        end
```

```
        viejo=var(jj,1);
    end

    % esto que sigue es nuevo respecto a la funciÛn
'extrae_resultado...'
    % genero una matriz 'puntero(k)' donde k va creciendo cada vez
que
    % se termina una rama y empieza la siguiente
    k=2;
    viejo=var(1,1);
    puntero(1)=1;
    suav=1; % =1 indica que V crece (rama ascendente); =-1 rama
descendente
    for jj=2:fv-3
        switch suav
            case 1
                if (viejo >= var(jj,1)) && (var(jj,1)>
var(jj+3,1))
                    suav=-1;
                    puntero(k)=jj-1;
                    k=k+1;
                end
            case -1
                if viejo<var(jj,1)&& (var(jj,1) < var(jj+3,1))
                    suav=1;
                    puntero(k)=jj-1;
                    k=k+1;
                end
        end
        viejo=var(jj,1);
    end
    puntero(k)=fv;  % hay algunos ciclos que no terminan donde se
espera
    ktotal=k-1;
    n_ciclos=(ktotal-1)/2;

% figure;
% plot(var(:,end));

    temp=[];
    ns=0;
    ciclo=n_ciclos-Nrepe+1;

    for j=1:Nrepe   % se quita 3/4 del primer ciclo
        indice(i,j,1)=1+ns; % apunta al inicio de la rama usada
        % manipulo datos var para tomar una o las dos ramas. Uso
la
        % variable rama (= 0, 1 Û 2)
        if rama==0 || rama==1  % rama oxidaciÛn
            PiniO=puntero(2*ciclo-1);
            PfinO=puntero(2*ciclo);
            ValInf=var(PiniO,1);
            ValFin=var(PfinO,1);
            % elimina los extremos de la rama. Quito 100 mV
            cont=PiniO;
            while var(cont,1)< ValInf+100
                cont=cont+1;
            end
            PiniOD=cont;
```

```
            cont=PfinO;
            while var(cont,1)> ValFin-100
                cont=cont-1;
            end
            PfinOD=cont;

            temp=[temp; var(PiniOD:PfinOD,:)];
            PiniD=PiniOD;
            PfinD=PfinOD;
            ns=ns+PfinD-PiniD+1;
        end

        if rama==1 || rama==2  %  la y de la rama de reducciÛn no
            % la cambio de signo. Lo que hago es una imagen
simÈtrica
            % respecto de un eje vertical, cambio la x para
            % que vaya a continuaciÛn de la x de la rama de
            % oxidaciÛn y siga creciendo

            PiniR=puntero(2*ciclo)+1;
            PfinR=puntero(2*ciclo+1)-1;
            % cambio la x como digo en el comentario de arriba
            temp1=var(PiniR,1);
            for k=PiniR+1:PfinR
                temp2=var(k,1);
                var(k,1)=var(k-1,1)+abs(temp2-temp1);
                temp1=temp2;
            end

            ValInf=var(PiniR,1);
            ValFin=var(PfinR,1);
            % elimina los extremos de la rama. Quito 100 mV
            cont=PiniR;
            while var(cont,1)< ValInf+100
                cont=cont+1;
            end
            PiniRD=cont;
            cont=PfinR;
            while var(cont,1)> ValFin-100
                cont=cont-1;
            end
            PfinRD=cont;
            if rama==1 % aÒadir rama reducciÛn a rama oxidaciÛn
                % Desplazar la y para que no haya discontinuidad
al
                % pasar de oxidaciÛn a reducciÛn. TambiÈn hay que
desplazar
                % la x por lo mismo
                deltaX=var(PiniRD,1)-var(PfinOD,1);
                deltaY=var(PiniRD,2)-var(PfinOD,2);
                for k=PiniRD:PfinRD
                    var(k,1)=var(k,1)-deltaX;
                    var(k,2)=var(k,2)-deltaY;
                end
                PiniRD=PiniRD+1;  % el primer punto no lo cuento
porque
                                  % es el mismo que el de
oxidaciÛn
                PiniD=PiniRD;
```

```
                PfinD=PfinRD; % el PiniD es el obtenido de la rama
de oxidaciÛn
            else
                PiniD=PiniRD; % el PiniD y PfinD son de la rama de
reducciÛn
                PfinD=PfinRD;
            end
            temp=[temp; var(PiniD:PfinD,:)];
            ns=ns+PfinD-PiniD+1;
        end
        indice(i,j,2)=ns; % apunta al final de la rama usada
        ciclo=ciclo+1;
    end
    PyM_red.(PyM_red.Listafiles{i})=temp; % aqui se almacena ya la
rama
end

% Extraigo los kernel de la rama de oxidaciÛn que he extraido
antes
% y lo pongo en R temporalmente

for i=1:f
    temp=[];
    for im=1:Nrepe
        fin=indice(i,im,2);
        ini=indice(i,im,1);
        delta=fin-ini+1;
        inicio=ini+floor(fr_I*delta);
        final=ini+floor(fr_F*delta);
        if final > fin
            final=fin;
        end

        vark=PyM_red.(PyM_red.Listafiles{i})(inicio:final,:);
        R=size(ntrozos,1);
%         R=[];
        [fv,~]=size(vark);

        x=vark(:,1);  % esto es la X
%         pmin=min(vark(:,2)); % mÌnimo de la Y
%         pmax=max(vark(:,2)); % m·ximo de la Y
        y=vark(:,2); % no normalizo
%         y=zscore(var(:,2));
%         y=(vark(:,2)-pmin)./(pmax-pmin);

        inter_total=x(fv,1)-x(1,1);
        ancho=inter_total/ntrozos; % en lo de vinos era ancho=120
        caida=10;

        paso=inter_total/(ntrozos-1);
        k=[];
        aux=[];
        ij=1;
        % lazo para la X dividida en ntrozos
        for j=x(1,1):paso:x(fv,1)   % calcula la suma punto por
punto del producto k * Y * incremento X
            for ii=2:fv % calcula la suma punto por punto del
producto k * Y
                k(ii)=1/(1+((x(ii,1)-j)/ancho)^caida);
```

```
                aux(ii)=k(ii)*y(ii,1)*abs(x(ii-1,1)-x(ii,1));  %
multiplica el kernel por la Y y por delta X
            end
            R(ij,1)=sum(aux);
            ij=ij+1;
            k=0;
            aux=0;
        end
        temp=[temp;R];
    end
    recortado.(PyM_red.Listafiles{i})=temp;
end
% end FINAL funciÛn KERNEL

% Generar matriz resultado. Se llamar· result_2way y tendr·
% columnas = Nsensores * ntrozos
% filas =Nmostos * (min_ciclos-1)

result_2way=zeros(Nmostos*Nrepe,Nsensores*ntrozos);
cont=1;
for i=1:Nmostos   % los Mostos
    for ij=1:Nrepe % las repeticiones de cada Mosto
        nt=0;
        for j=1:Nsensores % los sensores que hay
            punt=((j-1)*Nmostos)+i;
            varr=recortado.(PyM_red.Listafiles{punt})((ij-
1)*ntrozos+1:ij*ntrozos,1);
            result_2way(cont,nt+1:nt+ntrozos)=varr(:,1)';
            nt=nt+ntrozos;
        end
        cont=cont+1;
    end
end

result_3way=reshape(result_2way,Nmostos*Nrepe,ntrozos,Nsensores);

% Creamos la matriz de clases

clases=zeros(Nmostos*Nrepe,1);
cont=1;
j=1;
for i=1:Nmostos
    for ij=1:Nrepe
        clases(j)=cont;
        j=j+1;
    end
    cont=cont+1;
end

% Creamos la matriz de etiquetas para los loadings
% Hay tantas como ntrozos*Nsensores
% data = strsplit(num2str(1:N));
labsens={};
cont=1;
j=1;
for i=1:Nsensores
    for ij=1:ntrozos
        labsens(j)=strsplit(num2str(cont));
        j=j+1;
```

```
    end
    cont=cont+1;
end
labsens=labsens';

end
```

plotscat_uva()

```
function plotscat_uva(x, clab, testb, ndim)

nc = max(clab);

colormap('default');
cmap = colormap;
for e = 1:size(x,1)
  ccolor = cmap(round(clab(e)*size(cmap,1)/nc),:);
  str = num2str(clab(e));
  fw = 'Bold';
  fa = 'Normal';
  if testb
    str = [num2str(clab(e)) '*'];
    fw = 'Normal';
    fa = 'Italic';
  end
  if ndim==2
    text(x(e,1), x(e,2), str, ...
      'FontSize', [16], 'FontName', 'Helvetica', ...
      'FontWeight', fw, 'FontAngle', fa, ...
      'color', ccolor, 'HorizontalAlignment', 'center');
    axis([min(x(:,1)) max(x(:,1)) min(x(:,2)) max(x(:,2))]);
    xlabel('Axis1')
    ylabel('Axis2')
  elseif ndim==3
    text(x(e,1), x(e,2), x(e,3), str, ...
      'FontSize', [16], 'FontName', 'Helvetica', ...
      'FontWeight', fw, 'FontAngle', fa, ...
      'color', ccolor, 'HorizontalAlignment', 'center');
    axis([min(x(:,1)) max(x(:,1)) min(x(:,2)) max(x(:,2))
min(x(:,3)) max(x(:,3))]);
    xlabel('Axis1')
    ylabel('Axis2')
    zlabel('Axis3')
    view(3)
  end
end
```

9. Bibliografía

- REVILLA ALLENDE, Gema. “Diseño de una lengua bioelectrónica para el análisis de mostos”. Directora: Rodríguez Méndez, María Luz. Universidad de Valladolid, Escuela de Ingenierías Industriales, 2013.
- MURPHY, Kate. “The use of PARAFAC in the analysis of CDOM fluorescence.”Smithsonian Environmental Research Center, Edgewater (USA).
- GREGORIO CABRERO, Raúl. “Aplicación del análisis multivariante tridimensional al estudio de muestras medioambientales”. Tutor: Sánchez Bascones, Isabel. Universidad de Valladolid, Escuela de Ingenierías Industriales, 2012.
- SANCHO, Juna J. “Análisis Multivariante. Introducción” [Material gráfico proyectable]. Societat Catalana de Cirugia: 2012, 22 diapositivas.
- PORCEL, Marta. “Aplicación de técnicas quimiométricas para el desarrollo de nuevos métodos cinético-espectrofométricos de análisis”. Universitat Autònoma de Barcelona, Bellaterra, 2001.
- CUADRAS, Carles M. “Nuevos métodos de análisis multivariante”. Barcelona: 2014. CMC Editions. 305 p.
- VEGA ÁLVAREZ, Julia, et al. “Introducción al análisis multivariante: Técnicas básicas de ordenación y clasificación en R” [Material gráfico proyectable]. Universidad de Salamanca: 2015, 35 diapositivas.
- AMIGO RUBIO, José Manuel. “Desarrollo y aplicación de métodos quimiométricos ultidimensionales”. Director : Maspoch Andrés, Santiago; Coello Bonilla, Jordi. Universitat Autònoma de Barcelona, 2007.
- Pacheco D., Pedro N., Amaya L., Jeannette L. “Análisis factorial dinámico mediante el método Tucker3” [en línea]. Revista Colombiana de Estadística. [Consulta: 8 de mayo de 2016]. Disponible en Web: http://www.redalyc.org/articulo.oa?id=89925105. ISSN 0120-1751.
- NATAL, Marcela, et al. “El modelo Tucker3 en un problema de química ambiental”. Departamento de Matemáticas, Universidad Nacional de Mar de Plata, Argentina, 2012.