Aplicación de modelos de filtración y ensuciamiento en biorreactores de membrana usados para depuración de aguas residuales

Matheus Sampaio Galvao

Tutores:
Juan M. Rodríguez Sevilla y Luisa Mª Vera Peña

Grado Ingeniería Química Industrial

Curso: 2015/16

1 Tabla de contenido

Agradecimientos

Han sido unos cuantos años ya desde que por primera vez pisé tierras tinerfeñas para empezar esta aventura. No obstante, si miro hacia atrás, parece que ha pasado solo el tiempo de un pestañeo.

Entre las personas a las que quiero nombrar, la primera es la impulsora de todo esto: Rosangela da silva Sampaio, o como la llamo yo, "Mãe" (mamá).

Gracias también a todas esas personas con las que comencé, que me han ayudado y han pasado su tiempo conmigo: Roberto, Nohemy, Victoria, etc. A esas otras que conocí en el camino: Mustafa, Samuel, Matteo, Jesús y por último a todas aquellas que no están nombradas y que me "matarán" si leen esto y no ven su nombre escrito.

También, gracias por la paciencia y la amabilidad de los profesores de la Universidad de La Laguna, y en especial aquellos de la parte de Ingeniería Química. Gracias a mis tutores de TFG, Juan y Luisa, por la paciencia, la predisposición y la voluntad con la que me han ayudado a finalizar mi trabajo. Por último, agradecer a la profesora María Soledad Pérez Rodríguez, por ayudarme a darle continuidad a este trabajo.

Nomenclatura

A	Área de la superficie de la membrana	m^2
BRF_v	Caudal de burbujeo de biogás en el tanque	$Nm^3\ s^{-1}\ m^{-3}$
dt	Diferencial de tiempo de filtración	s
dV	Diferencial de volumen de permeado	m^3
FR	Velocidad de ensuciamiento	$Pa\ s^{-1}$
I_{MS}	Índice de limpieza de la membrana	-
J	Flujo de permeado	$m\ s^{-1}$
J_{20}	Flujo de permeado a 20 ºC aprox.	$m\ s^{-1}$
J_{20BF}	Flujo de retrolavado a 20 ºC aprox.	$m\ s^{-1}$
Q_{20P}	Caudal de permeado a 20 ºC aprox.	$m^3\ s^{-1}$
Q_{20BF}	Caudal de retrolavado a 20 ºC aprox.	$m^3\ s^{-1}$
K_F	Parámetro de ajuste para cuando J_{20} tiende a cero	$Pa\ s^{-1}$
k_{RI}	Constante de proporcionalidad	s
k_{SF}	Constante de ensuciamiento en condiciones subcríticas.	$m\ kg^{-1}\ s^{-1}$
$K_{S,XmC}$	Coeficiente de saturación media	kg
k_t	Constante de tiempo	s^{-1}
M_{XmC}	Función de conmutación de saturación media	-
TMP	Presión transmembrana	Pa
TMP_0	TMP inicial	Pa
TMP_a	TMP necesaria para doblar la resistencia específica	Pa
TS	Sólidos totales	kg
$q_{BF,max}$	Velocidad máxima de retrolavado	m^{-3}
$q_{IF,max}$	Constante cinética para máximo ensuciamiento irreversible	s^{-1}
$q_{MS,max}$	Velocidad máxima de limpieza de la membrana	-
R_C	Resistencia de la torta formada en la membrana	m^{-1}
R_I	Resistencia por ensuciamiento irreversible	m^{-1}

$\mathbf{R_{It}}$	Resistencia irreversible a tiempo determinado.	m^{-1}
$\mathbf{R_{I0}}$	Resistencia irreversible a tiempo inicial	m^{-1}
$\mathbf{R_M}$	Resistencia hidráulica de la membrana	m^{-1}
$\mathbf{R_T}$	Resistencia de filtración total	m^{-1}
t	tiempo	s
$\mathbf{X_{mC}}$	Masa seca formada en la superficie de la membrana	kg
$\mathbf{X_{mI}}$	Masa seca formada por ensuciamiento irreversible	kg
$\mathbf{X_{TS}}$	Concentración de MLTS (sólidos totales del licor mezcla)	$kg\ m^{-3}$

Letras Griegas

$\boldsymbol{\alpha}$	Resistencia específica	$m\ kg^{-1}$
$\boldsymbol{\alpha_C}$	Resistencia específica media de la torta	$m\ kg^{-1}$
$\boldsymbol{\alpha_{C,\ TMP}}$	Resistencia específica de torta a la TMP de operación	$m\ kg^{-1}$
$\boldsymbol{\alpha_I}$	Resistencia específica media del ensuciamiento irreversible	$m\ kg^{-1}$
$\boldsymbol{\alpha_{C,\ 0}}$	Resistencia específica de torta a una TMP cero	$m\ kg^{-1}$
$\boldsymbol{\beta_1}$	Parámetro 1 del modelo de FR	$s^2\ m^{-1}$
$\boldsymbol{\beta_2}$	Parámetro 2 del modelo de FR	$s\ m^2\ kg^{-1}$
$\boldsymbol{\gamma}$	Parámetro 3 del modelo de FR	$s\ m^{-1}$
$\boldsymbol{\gamma_0}$	Valor inicial de γ	$s\ m^{-1}$
$\boldsymbol{\mu}$	Viscosidad dinámica del solvente	$kg\ m^{-1}\ s^{-1}$
$\boldsymbol{\omega}$	Masa de partículas depositadas por unidad de área de membrana	$kg\ m^{-2}$
$\boldsymbol{\omega_C}$	Masa de la torta depositada por unidad de área de membrana	$kg\ m^{-2}$
$\boldsymbol{\omega_I}$	Masa del ensuciamiento irreversible por área de membrana	$kg\ m^{-2}$

2 Resumen

El objetivo de este trabajo consiste en el desarrollo de un código de programación que permita reproducir el proceso de filtración de un biorreactor anaerobio de membrana sumergida (SAnMBR).

El proceso de filtración sigue un modelo del tipo resistencias en serie, caracterizado por la suma de tres tipos de ensuciamiento que afectan a la membrana y que hace que aumente la presión transmembrana a medida que transcurre el tiempo. Los tres tipos de resistencias de filtración son: la resistencia hidráulica intrínseca de la membrana, el ensuciamiento producido por la formación y compresión de la torta, y el ensuciamiento irreversible que se consolida en la membrana.

El trabajo de ha dividido en tres partes, cada una está contenida en un guión de programación o *script* diferente, estos son los tres bloques de trabajo:

- Calibración "off –line" a corto plazo: nos permite obtener los parámetros relacionados con la velocidad de ensuciamiento, a partir de experimentos en escalón, de corta duración, a diferentes flujos de permeado y de biogás de limpieza.
- Calibración dinámica a corto plazo: nos permite obtener los parámetros relacionados con los procesos de formación y compresión de la torta, y el arrastre de torta provocado por el burbujeo de biogás.
- Simulación del proceso de filtración: una vez establecidos los parámetros (mediante los ficheros anteriores), se obtiene en un gráfico la simulación de la filtración, que contempla todos los procesos expuestos en el modelo.

Para la modelización computacional se ha empleado Python como lenguaje de programación debido a su legibilidad, fácil uso y que es de código abierto.

3 Abstract

The aim of this study is to develop a code for simulating a filtration process of submerged anaerobic membrane bioreactors (SAnMBRs).

This process is a "resistance-in-series" membrane filtration model that considers overall membrane resistance in terms of three components: intrinsic hydraulic membrane resistance, cake layer resistance due to the accumulation of solids in the membrane surface, and irreversible fouling resistance.

The work was divided in three blocks; each one is developed through an individual script:

- Off-line calibration in the short term: from different flux-step trials data is obtained in order to calibrate the parameters for membrane scouring by biogas sparging.
- Dynamic calibration in the short term: with experimental data, parameter estimation related to cake build-up and compression and membrane scouring by biogas sparging was completed.
- Simulation of the filtration process: once the parameters needed for the model were estimated, a simulation of the filtration process was obtained and represented in a graph.

In order to complete the work, computational modeling was completed through programming in Python language. Python was selected as a programming language due to legibility, easy to use and it is open-source.

4 Introducción

El término biorreactor de membrana (en inglés *membrane bioreactor* o MBR) se aplica a todos los procesos de tratamiento de aguas y aguas residuales que integran una membrana selectiva junto a un tratamiento biológico. Todos los procesos comerciales actuales de MBR emplean la membrana como filtro, rechazando los materiales sólidos producidos por procesos biológicos para proveer así un producto clarificado y desinfectado [1].

Esta tecnología se está viendo favorecida en los sectores en los que se requiere una calidad alta del agua tratada, sobre todo para su reutilización, y donde el espacio resulta limitado. Las zonas de costa con un atractivo paisajístico (como Agulo en La Gomera, Agaete en Gran Canaria o Valldemossa en Mallorca) utilizan la tecnología de MBR como la mejor opción [1].

En todas las regiones geográficas parece que se produce una expansión del mercado del MBR. Los MBRs están implementados en más de 200 países y su valor de mercado esperado en 2019 se pronostica en 3 billones de dólares; actualmente tiene un valor de 1,2 billones de dólares. También, parece ser una tendencia, el aumento en la confianza de esta tecnología, con el aumento en número y tamaño de estas instalaciones, con unas 40 plantas municipales de más de 100 megalitros/día (MLD) de capacidad (expresado como flujo pico diario, o PFD) actualmente. Incluso, existen en la actualidad instalaciones de tratamiento de efluyentes industriales de más de 50 MLD de capacidad [2].

Resulta de interés, tanto para propósitos académicos como para la práctica recoger el estatus de esta tecnología haciendo referencia a:

1. Las tecnologías comerciales de MBR disponibles
2. Las claves de diseño, operación y mantenimiento de las aplicaciones más comerciales
3. Mecanismos de ensuciamiento

4.1 Materiales de las membranas y módulos

Las membranas forman parte de módulos, que son las unidades base de una instalación de este tipo. Los módulos de membrana para los MBRs están configurados como planos o cilíndricos. El último comprende a los de tipo fibra hueca (HF, en inglés *hollow fibre*) y multitubo (MT), la diferencia entre estos dos radica en el diámetro y la dirección del flujo. Las membranas tipo HF (las más importantes en cuanto a superficie instalada) son filamentos estrechos (0.4 – 2.6 mm) donde el flujo de permeado circula de afuera hacia adentro. Las membranas tipo MT son más anchas (5 – 12 mm) con el sentido de flujo de dentro hacia afuera.

Los procesos de MBR están configurados con las membranas situadas en el interior o exterior del tanque (Figura 1). Si están inmersas (iMBR), el permeado se extrae o por presión de succión en el lado del permeado y/o a través de una presión estática de cabeza en el lado del retentado. Si la configuración es externa y opera por corriente lateral (sMBR) el agua permea la membrana bajo presión, la presión transmembrana requerida (TMP), generalmente, es mucho más alta que la empleada en membrana inmersa. Como las condiciones de la configuración iMBR son más favorables, estas se emplean en aguas residuales municipales, que representan un desafío menor que los efluentes industriales.

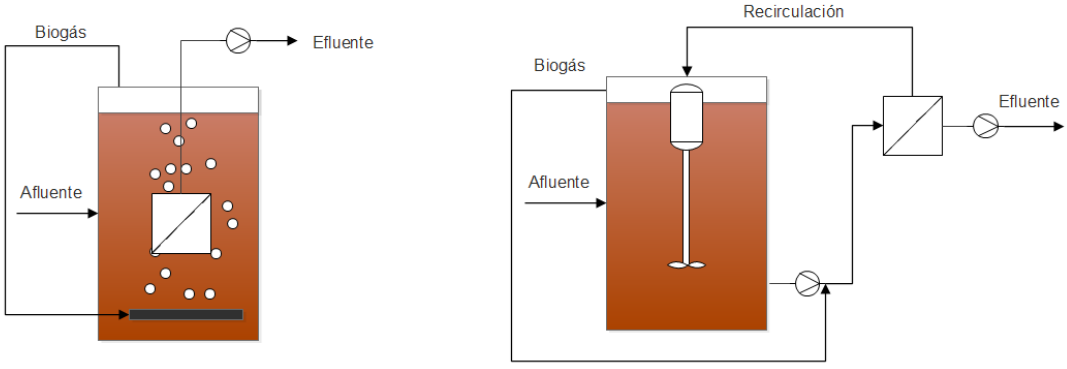

Figura 1. Configuración interna (izquierda) y configuración externa (derecha) de un MBR

El rango de materiales empleados comercialmente está bastante limitado para tecnologías de MBR, y los tamaños de poro se encuentran en el rango 0.03 – 0.4 μm. Casi la mitad de las membranas poliméricas de MBR están hechas de polifluoruro de vinildeno (o PVDF), muchas de las membranas restantes son de poliolefina (polipropileno, PP, o polietileno, PE) o de polietersulfona (PES). También existe un número creciente de productos cerámicos.

4.2 Operación, Mecanismos de ensuciamiento y Limpieza de membranas

Se pueden dar diferentes tipos de condiciones para operar con una membrana. Según este tipo de condiciones se puede dar una velocidad de ensuciamiento más alta o baja. El parámetro que más influye para esta variable es el flujo con el que se está operando. Si este flujo es alto, la velocidad de ensuciamiento es elevada, por el contrario, si el flujo es bajo el comportamiento de esta velocidad es menor. Es una variable que repercute de manera significante en el sistema ya que una productividad alta equivale a mayores costos de limpieza y menores costes de limpieza a una baja productividad.

Sin embargo, se ha observado que hay un valor para el flujo donde se da la filtración sin que se vayan depositando sustancias en la superficie de la membrana evitando así su ensuciamiento; este valor se denomina flujo crítico.

4.2.1 Mecanismos de ensuciamiento

Normalmente, los MBR operan bajo condiciones de flujo de permeado constantes, por lo que los agentes ensuciadores se mantienen en un ratio constante determinado por dicho flujo. Como la velocidad de ensuciamiento aumenta de forma exponencial con el flujo (ver Figura 7), una forma sostenible de operación sería trabajar bajo flujos modestos y, preferiblemente, por debajo del flujo crítico. No obstante, incluso trabajando por debajo del flujo crítico puede llevar al ensuciamiento de la membrana: se produce un pequeño incremento de la TMP después de un tiempo inicial, seguido de rápido incremento después de un tiempo crítico.

El efecto del ensuciamiento se traduce a una reducción de la permeabilidad de la membrana, es decir, se puede observar un aumento de la presión a un flujo constante. Esto conlleva a una reducción de la productividad, acorta la vida de la membrana y puede modificar la selectividad de la misma.

Este ensuciamiento en la membrana puede suceder en dos etapas: por ensuciamiento externo y/o ensuciamiento interno, que no sería de fácil limpieza mediante retrolavado.

El ensuciamiento interno es causado por la adsorción o el depósito de pequeñas partículas y macromoléculas dentro de la estructura interna de los poros, para este tipo de ensuciamiento se puede ver un comportamiento en la filtración donde se dan dos casos extremos. El primero corresponde al modelo de ensuciamiento estándar y el segundo, al modelo de bloqueo de poros, que se puede dividir en el modelo de bloqueo completo de poros y en el modelo de bloqueo intermedio de poros. Además, el ensuciamiento externo viene definido por el modelo de formación de torta.

a) Bloqueo completo de poros. Para este modelo se tiene en cuenta que cada partícula tiene un diámetro superior al del poro de tal manera que le permita encajar perfectamente en él obstruyéndolo por completo.

b) Bloqueo estándar de poros. Este modelo considera que la membrana está compuesta por poros cilíndricos rectos cuyo radio disminuye en relación a la acumulación de materia sólida que se acumula en sus paredes, aumentando así la resistencia hidráulica y permitiendo que disminuya el flujo o aumente la presión transmembrana..

c) Bloqueo intermedio de poro. En este caso, se tiene en cuenta que cada partícula se puede acercar a cualquier punto de la membrana depositándose en algún lugar de su superficie de forma aleatoria, es decir, que cada partícula no tiene por qué obstruir un poro sino que puede colocarse en la superficie limpia o encima de alguna otra partícula.

d) Formación de torta. En este modelo se considera una continua deposición de partículas a lo largo de toda la membrana generando así una capa de ensuciamiento o torta, el espesor de esta torta va aumentando en función del tiempo de manera lineal.

A continuación, se pueden observar en la siguiente figura los mecanismos de ensuciamiento:

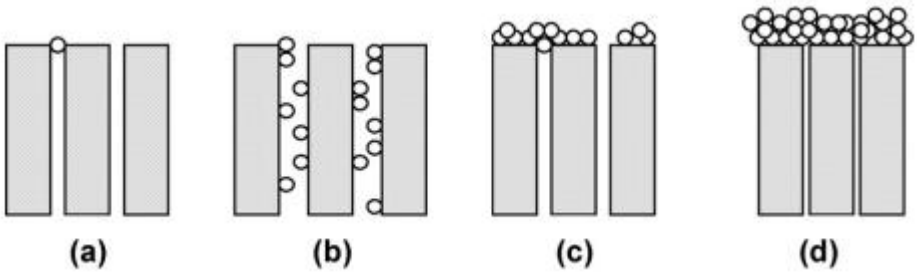

Figura 2. Mecanismos de ensuciamiento: (a) bloqueo completo, (b) bloqueo estándar, (c) bloqueo intermedio, (d) formación de torta

4.2.2 Limpieza de membranas

En el apartado anterior se han podido ver diversos mecanismos de ensuciamiento de una membrana, el ensuciamiento de una membrana condiciona la operación y el mantenimiento de los sistemas de filtración, ya que limita la vida útil de la membrana y repercute en el costo de energía pudiendo hacer inviable la operación. A medida que se ensucia la superficie de la membrana cuesta más que se produzca la filtración, aumenta la resistencia de la membrana siendo necesario más potencia para mantener constante el caudal de permeado.

Las técnicas para minimizar el ensuciamiento tratan de optimizar las propiedades de la membrana, las condiciones de la operación y las características de la biomasa. Pero por otro lado, estas técnicas no acaban con la necesidad de limpiezas periódicas. La limpieza puede ser física (se basa en métodos mecánicos) o química (se utiliza un agente oxidante).

4.2.2.1 Limpieza física

La limpieza física es el método de limpieza más sencillo y económico, además, al no introducir agentes químicos no daña las características de la membrana. No obstante, este método de limpieza solo afecta al ensuciamiento irreversible. La limpieza se puede hacer mediante dos métodos:

- El retrolavado: Consiste en invertir el flujo de permeado; esta acción (permite eliminar la mayor parte del ensuciamiento debido al bloqueo de los poros y una parte del ensuciamiento causada por la torta de filtración). Las variables relacionadas en este proceso de limpieza son: la frecuencia y el tiempo en el que se produce el retrolavado y el caudal empleado.
- Relajación: Tiene la misma finalidad que el retrolavado y consiste en detener la filtración. Incrementa la acumulación lenta de la suciedad, pero no se pierde permeado y conserva la biopelícula de la membrana. Esta biopelícula es más selectiva que la membrana, por lo que puede ser beneficiosa siempre que la resistencia no sea excesiva. Además, consume menos energía y por tanto tiene menos costes económicos que en el proceso anterior, ya que no requiere de la impulsión de un caudal al otro lado de la membrana,

Cabe destacar que en ambos procesos mecánicos de limpieza se suele ejercer un continuo burbujeo, este burbujeo está en continuo contacto con la torta formada en la membrana provocando una pequeña fuerza de cizalladura, de esta manera ayuda a optimizar el proceso aumentando la frecuencia de retrolavado o relajación.

5 Dispositivo experimental

Los resultados experimentales para realizar la comparativa con los valores que se obtienen a través de la simulación han sido recogidos de una instalación a escala laboratorio, facilitados por el grupo de investigación de Tratamiento y Reutilización de Aguas Residuales de la Universidad de La Laguna.

Los resultados corresponden a la evolución del ensuciamiento de una membrana a lo largo del tiempo. La instalación emplea una membrana de la compañía GE Water & Process Technologies, modelo Zeeweed® (ZW-1). La membrana opera sumergida en el biorreactor de tal manera que funciona filtrando a vacío. El biorreactor se alimenta con una corriente de agua en continuo, de este modo produce un permeado de forma continua.

5.1 Condiciones del agua a tratar

En la instalación se trata agua residual proveniente de la estación de depuración de aguas residuales (EDAR) de Santa Cruz de Tenerife, recogido del aula de reparto. Este tipo de agua ha sido sometida a las operaciones de desbaste y sedimentación en el clarificador primario.

Con la finalidad de mantener la homogeneidad de la suspensión biológica que compone el licor mezcla en el biorreactor, este se agita de forma continua mediante un imán que gira a 300 rpm, que se encuentra en el fondo del reactor.

Se facilitó la evolución de dos experimentos en los que el agua de alimentación presentó diferentes características. Lo cual es importante a la hora de establecer el grado de desviación del modelo en relación al tipo de agua. En especial en lo que se refiere a la concentración de sólidos, MLTS (sólidos totales del licor mezcla), que es uno de los parámetros más significativos a la hora de analizar los resultados de la simulación.

5.2 Equipos en la instalación

La instalación se compone de diversas partes para que se pueda acondicionar el agua. En este apartado se han dividido estas partes de la siguiente manera: propiedades y características de la membrana, depósitos empleados, sistema de conducción de gases y software.

5.2.1 Propiedades y características de la membrana

Un módulo de la compañía GE Water & Process Technologies, modelo Zeeweed®-1 (ZW-1) de PVDF con 80 fibras de una longitud de 20 cm. Cada fibra tiene un diámetro de poro nominal de 0,03 micras. Las fibras tienen una distribución radial en torno a una conducción más ancha que se encuentra en el centro del módulo; esta conducción se emplea para la circulación del biogás. A continuación se muestra el rango de operación y el esquema del módulo:

- Valor máximo TMP: 62 kPa.
- Valor máximo TMP en retrolavado: 55 kPa.
- Rango de operación TMP: 10-50 kPa.
- Temperatura de operación: 40 ºC.
- Rango de operación de pH: 5-9.
- Rango de limpieza de pH: 2-10,5.

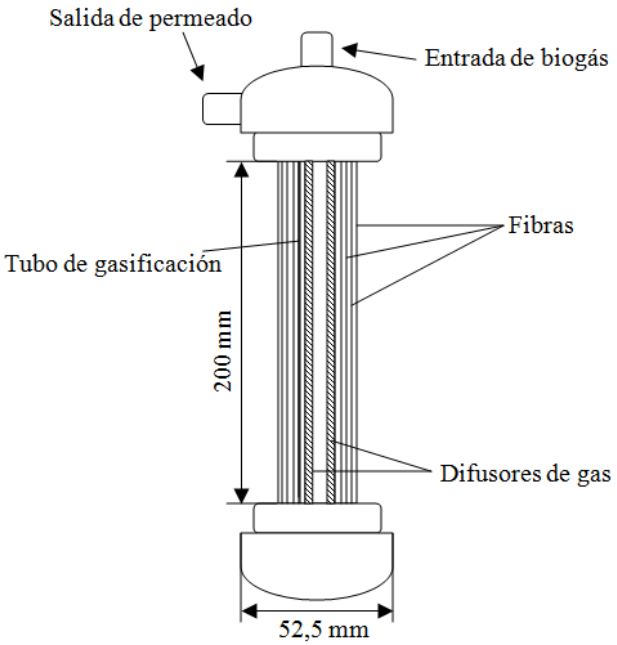

Figura 3. Esquema de un módulo Zeeweed-1.

5.2.2 Depósitos empleados

Para el tratamiento de agua se han empleado tres depósitos cada uno con una finalidad diferente: depósito de permeado, de operación y de alimentación.

El primero de los nombrados se emplea para recoger el permeado y está conectada al depósito de operación, concretamente al cabezal del módulo de membranas. Está conexión cuenta con una bomba reversible. La bomba se emplea para poder ejercer vacío y producir una diferencia de presiones respecto a la presión en el depósito y así producir la filtración. Además, al ser una bomba reversible se puede invertir el flujo con la finalidad de limpiar la membrana en el proceso de retrolavado.

El depósito de operación cuenta con una capacidad de 3 litros de volumen útil, el módulo se encuentra sumergido en dicho depósito de manera vertical y sujeto a la tapa del depósito, cabe destacar que se encuentra sellado para conseguir ausencia de aire y por tanto de oxígeno. El reactor se rellena de manera continua y de forma automática por medio de una bomba peristáltica (P-2) (Masterflex L/S). Esta bomba funciona según un controlador de nivel, de esta manera permite encenderse y apagarse con el fin de mantener un rango de nivel en el interior del depósito que permita la operación de filtración.

La alimentación del reactor proviene de un tanque con una capacidad de aproximadamente, 100 litros. De este tanque se sustrae agua con la bomba P-2 ya mencionada. Al igual que el reactor, cuenta con una varilla que se agita para mantener la homogeneidad de la alimentación.

5.2.3 Sistema de conducción de gases

El agua que alimenta el reactor se somete a un proceso de degradación anaerobia, este proceso genera biogás que es extraído del depósito de operación. Una parte del biogás que se genera es recirculado con la finalidad de provocar turbulencias que permitan disminuir el ensuciamiento que se va produciendo en la membrana.

La fracción de biogás que se purga es contabilizada por medio de un contador (GC) (*Ritter miligas counter*) antes de ser expulsado, mientras que la fracción del biogás que se recircula es controlada-regulada por un control de flujo másico. Esta fracción se extrae mediante un compresor (*Secoh air pump*). El sistema de recirculación cuenta además con una trampa de agua para eliminar del gas, el agua que haya podido extraerse con el compresor.

5.2.4 Software

La instalación cuenta con un software que permite la automatización del proceso. Este software se encarga de los sistemas de control que hay en la instalación, el control de nivel en el depósito de operación y el control de flujo másico para la recirculación del biogás. Además, recoge varios datos a partir de los sensores que hay en los distintos puntos, entre ellos, recoge los valores del ensuciamiento de la membrana permitiendo que se trabaje a flujo constante y pudiendo graficar esta variable con el tiempo para poder analizarlo posteriormente, y poder optimizar el caudal de biogás y su frecuencia o los caudales de permeado y retrolavado.

5.3 Configuración de la instalación

La instalación tiene una configuración de membrana sumergida interna, es decir, la membrana permanece sumergida en el reactor, de este modo tiene contacto directo con la suspensión microbiana del biorreactor. Al permanecer sumergida, esta membrana requiere de un funcionamiento a vacío para poder obtener la diferencia de presión que permita la filtración.

Además, el equipo emplea microorganismos anaerobios con el fin de aprovechar su descomposición, es decir, dichos microorganismos se degradan produciendo metano y dióxido de carbono. El gas generado es el empleado para el burbujeo, optimizando la energía del proceso. En la planta se recircula una parte, produciendo una pequeña turbulencia en la superficie de la membrana y dificultando la deposición de partículas en la superficie.

A continuación se muestra un esquema del funcionamiento de la planta, se muestran los equipos empleados y se puede apreciar el tipo de configuración y el sistema de conducción de los gases generados por digestión anaeróbica.

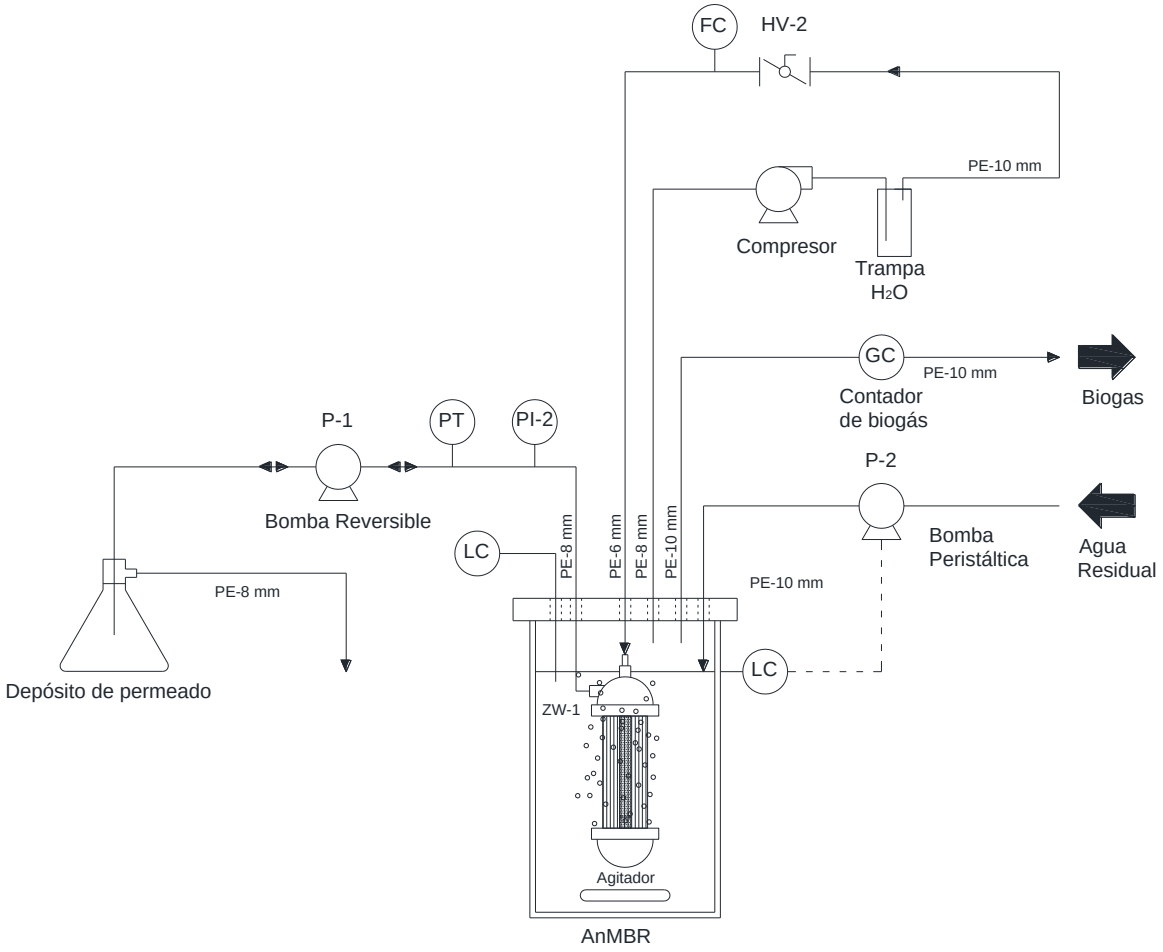

Figura 4. Diagrama de flujo de la unidad experimental.

5.4 Valores experimentales

Los valores experimentales facilitados son resultado de dos experimentos con diferentes variables y se han empleado para analizar cómo se adapta el modelo propuesto en diferentes condiciones. El modelo estudia las diferencias que puede haber para flujos y características diferentes del agua a tratar.

El flujo es una variable bastante relevante como se puede ver en la ecuación de Darcy más adelante. Además, uno de las variables que más nos interesa para modelizar este tipo de operaciones es la concentración de sólidos del licor mezcla (MLTS) que está en el interior del tanque.

A continuación se muestran las condiciones en cada experimento donde se ha obtenido la evolución de la TMP con respecto al tiempo.

Experimento 1:

- Flujo, 12 (L h^{-1} m^{-2}).
- Resistencia de la membrana, $5{,}27 \cdot 10^{11}$ (m^{-1}).
- MLTS, 5520 (mg L^{-1}).

Experimento 2:

- Flujo, 16 (L h^{-1} m^{-2}).
- Resistencia de la membrana, $6{,}32 \cdot 10^{11}$ (m^{-1}).
- MLTS, 6750 (mg L^{-1}).

Cabe destacar que la membrana, en cada experimento, es la misma, a pesar de observarse una leve variación de su resistencia hidráulica intrínseca. Esta variación se debe al ensuciamiento irreversible que se produce en cada operación que no se puede eliminar por el lavado químico que hay entre una y otra.

6 Modelización computacional

6.1 Introducción

Las simulaciones computacionales son usadas de forma rutinaria en la investigación, ayudando a entender experimentos, y para reemplazar, por ejemplo, la fabricación de muestras o experimentos caros, siempre que sea posible. En el contexto industrial, el diseño de productos y procesos puede ser llevado a cabo a un coste más eficiente en forma de simulaciones que si se realizasen prototipos y estos fueran testeados. Estas simulaciones son necesarias, sobre todo, en áreas donde construir modelos resulta muy caro, como en la nanociencia o en la industria aeroespacial. También, hay procesos que solo pueden llevarse a cabo de forma virtual, como el estudio de la astrofísica o algunos procesos químicos o nucleares [3].

La modelización computacional, incluido el uso de herramientas de post-procesado, análisis y visualización de datos, ha sido usada en ingeniería, física y química durante varias décadas, aunque se está volviendo cada vez más importante debido al bajo coste de los recursos computacionales. También, los modelos computacionales juegan un papel cada vez más importante en el estudio de sistemas biológicos, la economía, arqueología y la medicina, entre otros dominios [3].

6.2 Los modelos computacionales

En el estudio de un proceso mediante una simulación por ordenador, se distinguen dos pasos: el primero es desarrollar el modelo del sistema real. El modelo puede representar de forma aproximada el comportamiento real de un proceso/sistema, y esto se realiza a través de ecuaciones matemáticas, generalmente, en forma de matrices, ecuaciones diferenciales ordinarias (ODE) o ecuaciones en derivadas parciales (EDP) [3].

En ciencias naturales, como la física, química o ingeniería, normalmente no resulta muy difícil encontrar un modelo apropiado, aunque las ecuaciones resultantes suelen ser difíciles de resolver, y en muchos casos no se pueden resolver de forma analítica [3].

Por otro lado, en temas en los que el comportamiento de los objetos de estudio son imposibles de predecir de forma determinística (como los seres humanos), resulta mucho más difícil encontrar un buen modelo que se ajuste a la realidad. Se pueden citar los siguientes ejemplos: simular la economía, el uso de los recursos mundiales, el comportamiento de un grupo de personas que entra en pánico, etc. [3].

Hasta ahora, no se ha mencionado el uso de ningún trabajo computacional o numérico, simplemente se ha tratado el desarrollo de modelos que describen la realidad. De hecho, siempre que se pueda resolver un modelo de forma analítica, entonces se debería resolver por ese camino y emplear dicha solución [3].

No obstante, en la práctica, existen pocos modelos que puedan resolverse de forma analítica, y aquí es donde tienen cabida los ordenadores: usando métodos numéricos, al menos se puede estudiar el modelo dentro de unos límites y condiciones [3].

La solución numérica puede ser calculada usando un ordenador, este mostrará el cambio de una o varias variables con respecto al tiempo (normalmente). El modelo proporcionará una gran lista de pares de valores de tiempo y la/s variable/s a estudiar. Dichos pares de valores se pueden representar en una gráfica o ajustar los datos a una curva. Después, se podrá entender la tendencia de dichos datos (si se tuviese la solución de forma analítica se podría ver directamente) [3].

El nombre *modelización computacional* deriva de los dos pasos: 1. *Modelización*, encontrar un modelo que describa al sistema real, y 2. Resolver las ecuaciones de modelo resultante usando métodos *computacionales*, porque de hecho, a veces es el único modo en el que se pueden resolver dichas ecuaciones [3].

6.3 La programación como soporte de la modelización computacional

Existen un gran número de herramientas de modelización computacional. Si se pueden satisfacer las necesidades para el diseño, procesado y visualización de los datos, sin el uso de habilidades de programación, entonces no es necesario tener ningún conocimiento de programación [3].

Sin embargo, si se llega a un punto en el que las necesidades en cuanto a investigación o desarrollo de un sistema o proceso, mediante el uso de herramientas convencionales quedan obsoletas, es aquí donde las habilidades de programación se hacen necesarias [3].

Entonces, para construir una simulación por ordenador, se necesita: 1. Encontrar el modelo (mediante las ecuaciones adecuadas), 2. Saber cómo resolver dichas ecuaciones numéricamente, y 3. Implementar dichos métodos numéricos para obtener las soluciones (lo que implica programar) [3].

6.4 ¿Por qué usar Python para computación científica e ingeniería?

Python es un lenguaje de programación fácil de aprender, fácil de usar y de código abierto. El hecho de que su sintaxis sea sencilla lo hace atractivo para no programadores, además su sintaxis es parecida a las usadas en MATLAB, Java, C++ o Visual Basic.

Uno de sus puntos fuertes es que no solo se trata de un lenguaje de programación de uso genérico, sino que también es fácil de usar para propósitos de computación científica. Python se ha estado empleando, durante más de una década, en campos tan diversos como el sector del gas y petróleo, el mundo financiero, en el procesamiento de señales y en física.

El diseño del lenguaje Python se centra en su productividad y la legibilidad del código. A continuación se muestran algunas de sus características:

- Consola Interactiva de Python.
- Sintaxis fácil de leer y clara; se usa el sangrado por espacios en blanco.
- Gran capacidad de introspección; rápido reconocimiento de las estructuras.
- Separado en módulos (o librerías).
- Manejo de errores basado en excepciones.
- Tipos de datos dinámicos y manejo de la memoria automática.

Aunque, a veces, se le haya criticado de lenguaje "lento" de programación, no hay que olvidar la visión global: Python es suficientemente rápido para la mayoría de tareas computacionales, y es uno de los lenguajes de programación interpretada más amigables para el usuario en general (comparado con los lenguajes compilados).

6.5 Instalación, versión y uso de Python

Python se puede instalar fácilmente con la distribución gratuita *Anaconda* de *Continuum Analytics* (https://www.continuum.io/downloads#_windows) o *Canopy* de *Enthought* (https://store.enthought.com/downloads/#default).

Actualmente existen dos versiones del lenguaje de Python: Python 2.x y Python 3.x. Existen pequeñas diferencias entre las dos y aunque los cambios se produjeron para una mejora en el lenguaje, pueden existir ciertas incompatibilidades. En este caso se ha programado en la versión de Python 3.4.3.

6.5.1 Instalación de Python (con Anaconda)

Al hacer *click* en el enlace proporcionado anteriormente para Anaconda, se encuentran disponibles dos opciones (Python 2.7/Python 3.5) en dos modalidades (64-bit/32-bit). Se recomienda descargar el instalador gráfico Python 3.5 y 64-bit para un mayor aprovechamiento de los recursos del ordenador.

A continuación, hacer doble *click* en el archivo *.exe* para instalar Anaconda y seguir las instrucciones que se dan por pantalla.

6.5.2 Instalación de la libreria LMFIT (Anaconda)

Algunas librerías no vienen por defecto en las distribuciones anteriores. Este es el caso de LMFIT. Esta librería se debe instalar desde la consola de Anaconda (*Anaconda Command Prompt).* Para ello se deben seguir los siguientes pasos:

Ir a *Inicio > Todos los programas > Anaconda3 (64-bit) > Anaconda Prompt*

Desde la consola de Anaconda escribir y seguidamente presionar la tecla *Enter*:

```
pip install lmfit
```

A partir de ahora, se podrá importar y usar la librería LMFIT.

6.5.3 Ejecución del código

Para la correcta ejecución del código se necesitarán dos cosas: 1. El fichero con el código, en la extensión *.py;* y, 2. El fichero con los datos experimentales en formato Microsoft Excel.

Para ejecutar el código se pueden seguir varias alternativas. Se recomienda utilizar el editor de texto que se instala por defecto con la distribución de Anaconda, este editor se llama *Spyder*. Para acceder a él ir a *Inicio > Todos los programas > Anaconda3 (64-bit) > Spyder*.

Una vez abierto el editor de textos *Spyder*, en la esquina superior derecha se podrá elegir el directorio de trabajo. En dicho directorio, debe estar guardado el fichero del código en Python

y el archivo Excel. A continuación, ir a *File > Open* (o presionar *Ctrl + O*) y seleccionar el archivo *.py*.

En el panel de la izquierda del editor se abrirá el código contenido en el fichero. Una vez abierto se puede ejecutar el código presionando la tecla *F3* o mediante *Run > Run*. En el panel derecho, se muestra la consola interactiva de Python (*IPython*), en la que se mostrarán los resultados obtenidos de ejecutar el código.

6.5.4 Material de aprendizaje

Aunque el código empleado en la simulación se explicará en las siguientes secciones, es necesario dominar aspectos básicos de programación en Python. Como dentro del objetivo de este trabajo no entra explicar cómo usar Python, se proporcionan una serie de fuentes útiles para aprender los conceptos básicos y/o consultar información:

- Tutoriales para no programadores con orientación científica (http://software-carpentry.org/lessons/)
- John Kitchin, Python Computations in Science and Engineering (http://kitchingroup.cheme.cmu.edu/pycse)
- CaCheme, Video tutorial de Python Científico en castellano (https://www.youtube.com/user/CAChemEorg).

También puede ser útil la siguiente documentación:

- La documentación de Python (https://docs.python.org/3.4/)
- Numpy y Scipy (http://docs.scipy.org/doc/)
- Matplotlib (http://matplotlib.org/1.5.1/index.html)

7 Modelización del ensuciamiento de una membrana

Un modelo es una representación de una realidad compleja, es decir, trata de desarrollar una descripción lo más exacta posible de un sistema. En este caso se basa en la extracción de una serie de funciones matemáticas capaces de prever el comportamiento de las variables de operación del equipo en el proceso de filtración por membranas para el tratamiento de aguas residuales.

Hay diferentes tipos de modelo para un mismo sistema según los factores del propio sistema o exactitud de los resultados que se desean. Los factores que pueden influir en la elección de un modelo incluye el tipo de problema de calidad del agua que se desea resolver, las características del agua, la disponibilidad de datos observados que permita simular estrategias para la optimización del propio proceso, etc.

La elección del modelo aplicado viene dado porque hace referencia a los fenómenos físicos más destacados que tienen lugar, tratando una serie de parámetros de los que se puede tener conocimiento y, por tanto, permitiendo la simulación del comportamiento de la membrana. Además, este modelo permite la simulación a diferentes condiciones con la finalidad de conseguir una optimización energética y por tanto un ahorro económico.

7.1 Descripción del modelo

El modelo aplicado está basado en un modelo de resistencias en serie, Robles et al. [4]. Usando los resultados experimentales obtenidos en un proceso de ultrafiltración de aguas residuales, mediante un módulo de membranas de fibras huecas a escala laboratorio, Vera et al. [5], [6].

El modelo contempla dos parámetros que usualmente se miden en el proceso: la concentración de sólidos en el agua a tratar, MLTS, y el biogás que se recircula con la

finalidad de limpiar la superficie de la membrana y aumentar el tiempo del ciclo de filtración. Aunque MLTS es un parámetro elemental en comparación con la complejidad del modelo, la concentración de sólidos en el agua se definió como una variable de entrada en el modelo porque puede estar directamente relacionada con los modelos biológicos existentes. Además, es fácil de medir. Estos dos parámetros son considerados clave en el ensuciamiento que se produce en la membrana.

Este estudio teórico del ensuciamiento de membrana reproduce los principales procesos que ocurren durante la filtración en este tipo de equipos, a saber:

- Formación y compresión de torta durante la filtración.
- Limpieza mediante el uso de biogás.
- Una limpieza más efectiva a través de la inversión del flujo de permeado, es decir, retrolavado.
- Ensuciamiento irreversible que se va produciendo en la membrana.

7.2 Modelo de resistencias en serie

El modelo de resistencias en serie describe el flujo (J, $m^3\ s^{-1}\ m^{-2}$) a través de la membrana empleando la Ley de Darcy. En ella se considera que el volumen de permeado (V, m^3) es conducido a través de la membrana por la diferencia en la presión transmembrana hidráulica (TMP, Pa) frente a la viscosidad dinámica del disolvente (μ, $kg\ m^{-1}\ s^{-1}$) y a la resistencia total de filtración (R_T, m^{-1}), que se asume igual a la suma de las diferentes resistencias parciales:

$$J = \frac{dV}{A \cdot dt} = \frac{TMP}{\mu \cdot R_T} \qquad \textbf{ec. 1}$$

El modelo contempla las siguientes tres resistencias parciales: la resistencia ofrecida por la torta (R_C, m^{-1}), la resistencia hidráulica intrínseca de la membrana (R_M, m^{-1}) y la resistencia de ensuciamiento irreversible (R_I, m^{-1}):

$$R_T = R_C + R_M + R_I \qquad \textbf{ec. 2}$$

La resistencia que ofrece la torta está definida por el cociente m_C / A como ω_C (la masa de torta depositada en la membrana por unidad de superficie de membrana, en kg m^{-2}) y la resistencia específica media que ofrece la torta (α_C, m kg^{-1}):

$$R_C = \omega_C \cdot \alpha_C \qquad \textbf{ec. 3}$$

En lo que respecta a R_I, se utiliza la misma aproximación que las empleadas en ω_C y α_C para definir la resistencia específica media de ensuciamiento irreversible (α_I, m kg^{-1}) y la masa seca de ensuciamiento irreversible depositada por unidad de superficie de membrana (ω_I, m kg^{-1}):

$$R_I = \omega_I \cdot \alpha_I \qquad \textbf{ec. 4}$$

En relación a la resistencia hidráulica de la membrana, su valor depende de la naturaleza y características del equipo, es decir, del material, el diámetro y longitud de poro, si este poro es regular o irregular, etc. Este valor es ligeramente diferente para cada caso, debido a que a pesar de la limpieza química entre un experimento y otro no se puede mantener el valor inicial original.

El valor de la resistencia hidráulica se halla de forma experimental, mediante la ley de Darcy, en experimentos con agua destilada. Se aplica este fluido porque así se garantiza que no se adhieren partículas en la membrana, es decir, no habrá ensuciamiento de ningún tipo en la superficie de la membrana siendo la resistencia total equivalente a la resistencia hidráulica.

Por otra parte, debido a que el flujo en este estudio va a permanecer constante, la variable a seguir será la evolución de la presión transmembrana. Por tanto, se puede asumir, que la siguiente ecuación representa la evolución dinámica de la TMP:

$$TMP(t) = J \cdot \mu \cdot [\, \omega_C\,(t) \cdot \alpha_C(t) + \omega_I(t) \cdot \alpha_I + R_M\,] \qquad \textbf{ec. 5}$$

7.3 Evolución de la resistencia específica

Se ha obtenido la evolución de la resistencia específica siguiendo la metodología de Robles et al. [4], propuesta por Bugge et al. [7], y Jørgensen et al. [8] Este método asume una relación lineal entre la resistencia específica y la TMP:

$$\alpha_{C,TMP} = \alpha_{C,0} \cdot \left(1 + \frac{TMP}{TMP_a}\right) \qquad \textbf{ec. 6}$$

Siendo $\alpha_{C,\,TMP}$ (m kg^{-1}) la resistencia especifica de la torta a la TMP de operación; $\alpha_{C,0}$ (m kg^{-1}), la resistencia específica de la torta para una presión de operación cero; y TMP_a (Pa), la presión transmembrana necesaria para doblar la resistencia específica.

La compresión de la torta causada por una caída de la presión es dependiente del tiempo debido a: la deformación de los flóculos de los fangos, y el reajuste estructural de las

partículas. El aumento de la resistencia específica de la torta como resultado de una caída de presión en función del tiempo, se describe como:

$$\frac{d\alpha_C}{dt} = k_t \cdot \left(\alpha_{C,TMP} - \alpha_C\right) \quad \textbf{ec. 7}$$

Donde $d\alpha_C/dt$ es la variación de la resistencia específica. El parámetro k_t es una constante de tiempo, (s^{-1}).

Además, se ha observado que la resistencia total de filtración aumenta, incluso cuando se opera en condiciones subcríticas, durante periodos prolongados de filtración. Esto se atribuye a que en este tipo de modelos no se tiene en cuenta mecanismos específicos de ensuciamiento como la absorción de la materia coloidal. En este aspecto, Hughes y Field [9] observaron que las absorción de materia coloidal aumenta la resistencia específica de los depósitos tipo torta. En consecuencia, el modelo incorpora otra dependencia de α_C:

$$\frac{d\alpha_C}{dt} = k_{SF} \quad \textbf{ec. 8}$$

, donde k_{SF} (m kg^{-1} s^{-1}) representa el parámetro asociado al ensuciamiento en condiciones subcríticas.

El modelo combina la ec. 7 y la ec. 8 para obtener la variación final de la resistencia específica, de forma que:

$$\frac{d\alpha_C}{dt} = máx.(k_{SF},\ k_t \cdot (\alpha_{C,TMP} - \alpha_C)) \qquad \textbf{ec. 9}$$

Por tanto, cuando hay un máximo en el incremento de la resistencia específica, éste se puede relacionar con la compresión de las partículas a una cierta TMP o también es posible atribuirlo a la absorción de coloides.

7.4 Enfoque

El modelo utiliza un enfoque tipo "caja negra" (*black-box approach*) para describir las interacciones más importantes que ocurren en el proceso de filtración de un SAnMBR, como son: el efecto de la concentración de sólidos en la superficie de la membrana, la eliminación de dichos sólidos por diversos métodos y el ensuciamiento irreversible que se va produciendo a lo largo del proceso. El modelo contempla tres componentes en suspensión:

- X_{TS} (kg m^{-3}), concentración de sólidos totales en el licor mezcla (MLTS).
- X_{mC} (kg), masa seca de torta sobre la superficie de la membrana.
- X_{mI} (kg), masa seca de ensuciamiento irreversible sobre la superficie de la membrana.

Además, el modelo aplicado contempla un total de cuatro procesos físicos y cada proceso se definirá con una ecuación cinética:

1. Formación de la torta durante la filtración.
2. Eliminación de la torta por burbujeo de la membrana con biogás.
3. Eliminación de torta mediante un proceso de retrolavado.
4. Consolidación del ensuciamiento irreversible.

En la siguiente tabla se muestra la estequiometria de estos cuatro procesos

j Proceso	Componente *i*		
	X_{TS} (kg TS m^{-3})	X_{m_C} (kg TS)	X_{mI} (kg TS)
1. Formación de la torta	-1	1	
2. Limpieza por burbujeo de biogás	1	-1	
3. Limpieza por retrolavado	1	-1	
4. Consolidación del ensuciamiento irreversible		-1	1

Tabla 1. Estequiometría de las cinéticas contempladas en el modelo

La estequiometría de esta tabla indica con un número negativo si desaparece o, por el contrario, si aparece cada uno de estos componentes en cada cinética; el espacio en blanco quiere decir que dicho componente al que hace referencia no interviene en la cinética indicada.

La Tabla 2 muestra las expresiones cinéticas de los procesos incluidos en el modelo. El proceso 1 (formación de la torta) representa el transporte convectivo de los agentes ensuciantes (X_{TS} en el modelo) hacia la membrana, que es función del caudal de permeado (Q_{20P}, $m^3\ s^{-1}$) y de la concentración (X_{TS}). El proceso 2 (burbujeo de la membrana con biogás) representa el impacto de las condiciones hidrodinámicas en el tanque de la membrana, causado por el burbujeo de un caudal de biogás (medido como BRF_V, caudal de biogás reciclado por volumen en el tanque de la membrana). Se define también para el proceso 2, una velocidad máxima de limpieza de la membrana ($q_{MS,\ max}$).

j Proceso	Expresión cinética
1. Formación de la torta	$Q_{20P}\ X_{TS}$
2. Limpieza por burbujeo de biogás	$q_{MS,max}\ M_{X_{mC}}\ I_{MS}\ BRF_V\ X_{m_C}$
3. Limpieza por retrolavado	$q_{BF,max}\ Q_{20BF}\ M_{X_{mC}}\ X_{m_C}$
4. Consolidación del ensuciamiento irreversible	$q_{IF,max}\ X_{m_C}$

Tabla 2. Expresiones cinéticas de los procesos incluidos en el modelo.

Una función de conmutación de saturación media, M_{XmC}, influye de manera proporcional en los dos procesos de limpieza: por burbujeo de biogás (proceso 2) y por retrolavado (proceso 3). Dicha función viene definida de la siguiente manera:

$$M_{X_{mC}} = \frac{X_{mC}}{K_{S,XmC} + X_{mC}} \quad \textbf{ec. 10}$$

Donde $K_{S,XmC}$ (kg) es el coeficiente de saturación media para la masa de torta durante los procesos de limpieza a los que se somete la membrana.

Por otro lado, para obtener el índice de limpieza de la membrana (I_{MS}) se ha de calcular la velocidad de ensuciamiento (*FR*), ya que ambos parámetros van ligados estrechamente. Para el cálculo de esta velocidad se han comparado varios experimentos en los cuales se observaba la *FR* a diferentes caudales de burbujeo y flujos (J_{20}). Los resultados conducen a una dependencia de la velocidad de ensuciamiento con el caudal de biogás empleado y con la concentración *MLTS*, definida por la siguiente función:

$$FR = K_F \cdot \exp[J_{20} \cdot (\beta_1 \cdot BRF_V + \beta_2 \cdot MLTS + \gamma)] \quad \textbf{ec. 11}$$

El valor K_F (Pa s^{-1}) es un parámetro de ajuste del que depende la velocidad de ensuciamiento. Este parámetro está normalizado para un flujo a una temperatura de 20ºC. Los valores de β_1 (s^2 m^{-1}), β_2 (s m^2 kg^{-1}), γ (s m^{-1}) son parámetros del modelo que se hallan de forma experimental.

En experimentos de larga duración, el valor del parámetro γ se define como una función de la resistencia irreversible de la membrana, para tener en cuenta la reducción de la capacidad de

filtración de la membrana con el tiempo. La ecuación que define dicha reducción es la siguiente:

$$\gamma_t = \gamma_0 - (R_{It} + R_{I0}) \cdot k_{RI} \quad \textbf{ec. 12}$$

, siendo

γ_t (s m^{-1}) , el valor de γ para un tiempo determinado.

γ_0 (s m^{-1}), el valor de γ en el momento inicial.

R_{It} (m^{-1}), la resistencia irreversible en un tiempo determinado.

R_{I0} (m^{-1}), la resistencia irreversible en el momento inicial de la filtración.

k_{RI} (s), una constante de proporcionalidad.

Sin embargo, en el estudio realizado por Robles et. al [4] se pudo observar que el efecto es inapreciable, por lo que en el modelo aplicado, para facilitar los cálculos, se ha definido este parámetro como constante e igual al que presenta en un tiempo inicial, es decir, $\gamma_t = \gamma_0$.

La siguiente ecuación expresa el índice de limpieza que tiene lugar en la membrana, en tanto por uno. Este valor indica la efectividad de la limpieza que está teniendo lugar en la membrana. La expresión está en función de la velocidad de ensuciamiento, tal y como se dijo anteriormente. Si la velocidad de ensuciamiento es alta implicaría un índice de limpieza que tiende a cero, mientras que una velocidad de ensuciamiento baja mostraría una tendencia al valor uno.

$$I_{MS} = \frac{1}{1 + FR} \quad \textbf{ec. 13}$$

Aparte de los parámetros mencionados, también influye la masa seca que se forma en la superficie de la membrana (X_{mC}). Este valor no es constante y va variando con el tiempo.

En lo que respecta al proceso de limpieza por retrolavado (proceso 3), se puede observar (ver Tabla 2) que la limpieza de la torta es función del caudal de retrolavado (Q_{20BF}, $m^3\ s^{-1}$) y de una constante cinética ($q_{BF,max}$, m^{-3}). Además, también se indica una relación con la función de conmutación de saturación media y la masa seca que haya adherida en la membrana.

Finalmente, en el proceso 4 se observa la relación del ensuciamiento irreversible con la masa seca que se van adhiriendo a la superficie de la membrana, de forma que según aumenta se va incrementando el ensuciamiento irreversible. Como en los procesos físicos anteriores, este tipo de ensuciamiento se vincula a una constante cinética, máximo ensuciamiento irreversible ($q_{IF,max}$, s^{-1}).

7.5 Operación de filtración

Una vez que se han explicado los fenómenos físicos y se han definido las ecuaciones, se detallará el comportamiento de la simulación.

Durante la simulación, el proceso de filtración en el tratamiento de aguas residuales transcurre en dos etapas:

- Etapa de operación: aquí tiene lugar el proceso de filtración.
- Etapa de lavado: en esta parte del proceso se limpia la superficie de la membrana para que se pueda filtrar con la mayor productividad posible.

En la etapa de operación, tiene lugar la formación de la torta, el burbujeo de biogás y el ensuciamiento irreversible (procesos 1, 2 y 4, respectivamente). En la segunda fase, etapa de lavado, tiene lugar el fenómeno de limpieza por retrolavado (proceso 3).

En la etapa de operación se calculará: la evolución de masa seca que se va depositando en la superficie de la membrana (X_{mC}), la masa seca que va adhiriéndose de forma irreversible a esta superficie (X_{mI}), además de la resistencia específica de la torta (α_C). Estas ecuaciones se obtienen a partir de la Tabla 1:

$$\frac{dX_{mC}}{dt} = (Q_{20P} \cdot X_{TS}) - \left(q_{MS,max} \cdot \frac{X_{mC}}{K_{S,XmC} + X_{mC}} \cdot I_{MS} \cdot BRF_V \cdot X_{mC}\right) - \left(q_{IF,max} \cdot X_{mC}\right) \quad \textbf{ec. 14}$$

$$\frac{dX_{mI}}{dt} = q_{IF,max} \cdot X_{mC} \quad \textbf{ec. 15}$$

$$\frac{d\alpha_C}{dt} = \max\left(k_{SF}\,, \quad k_t \cdot \left(\alpha_{C,0} \cdot \left(1 + \frac{TMP_{sim}}{TMP_a}\right) - \alpha_C\right)\right. \quad \textbf{ec. 16}$$

Para la etapa de lavado, la masa seca que forma la torta se va eliminando de la superficie de la membrana (proceso 3) y, por el contrario, las partículas más pequeñas que se incrustan y forman el ensuciamiento irreversible (X_{mI}) permanecen constantes. La expresión que permite hallar la eliminación de la masa de torta es la siguiente:

$$\frac{dX_{mC}}{dt} = -q_{BF,max} \cdot Q_{20BF} \cdot \frac{X_{mC}}{K_{S,XmC} + X_{mC}} \cdot X_{mC} \quad \textbf{ec. 17}$$

Los valores de X_{mC} y X_{mI} corresponden a los valores de m_C y m_I incluídas en la ec. 3 y ec. 4 para el cálculo de las resistencias de ensuciamiento reversible e irreversible.

8 Programación del modelo

La programación se ha llevado a cabo mediante tres ficheros de código o *scripts*. Los ficheros son:

1. "flux_step.py": es el encargado de obtener los parámetros β_1 , β_2 , γ y K_F a partir de los experimentos de flujo-escalón ;
2. "dyn_calib_4.8.py": realiza la estimación de los parámetros k_{SF}, $\alpha_{C,0}$, TMP_a, y $q_{MS,max}$ y;
3. "exp_vs_sim_4.8.py": fichero que realiza la simulación de la filtración.

Los ficheros deben ejecutarse en el orden anterior, ya que los resultados de un fichero se deben introducir en el otro de la siguiente forma:

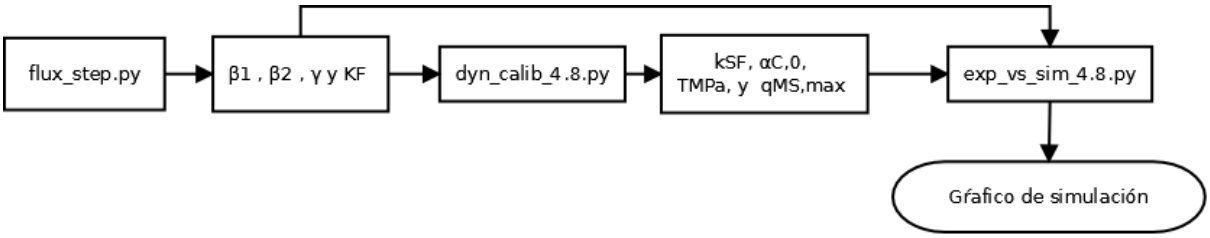

Figura 5. Orden del uso de los *scripts*

Por otro lado, para los *script* "dyn_calib_4.8.py" y "exp_vs_sim_4.8.py" se pedirá que se introduzca por la consola de Python si se quieren analizar los datos 1 (Experimento 1) o los datos 2 (Experimento 2), ya que se tienen dos sets de datos para experimentos con distintos valores de flujo, MLTS y resistencia de membrana.

A continuación, se pasa a la explicación del código de programación de cada *script*. Se ha utilizado el formato que se vería en Python con propósitos de mejorar la asimilación del contenido.

8.1 *Script* "flux_step.py"

Este *script* permite obtener los cuatro parámetros β_1 (variable *beta1*), β_2 (variable *beta2*), γ (variable *gamma*) y K_F (variable *kf*). Para ello se emplean los datos del experimento de flujo escalón.

Se comienza importando aquellas librerías necesarias; se emplearán abreviaciones (*np, plt* y *pd*) para escribir el código más rápido.

```
# -*- coding: utf-8 -*-
"""
Programa que estima los 4 parametros:
kf, beta1, beta2, gamma

Se realiza con la funcion minimize (una funcion para ajuste de curvas en
sistemas lineales y no lineales).
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from lmfit import minimize, Parameters, report_fit
```

Se pasa a la importación de los datos experimentales desde una hoja de Excel. Los datos se guardarán en una variable y se realizará una copia de ella (así no se modificarán los datos originales).

```
# Guarda los datos de calibracion en objeto tipo DataFrame (df) de pandas:
cal_df = pd.read_excel("TFGPy.xlsm", sheetname="CalibraciÃ³nPy", skiprows=1)

#Se realiza una copia del DataFrame
df1 = cal_df.copy()
```

Seguidamente se separan los datos por columnas y se pasa a tratar los datos. Se debe tener en cuenta que el programa recoge un rango de filas constante.

En este caso no es un problema, pero como se verá en el *Script* "dyn_calib_4.8.py", si existieran varias series (columnas) de datos de distinto número de filas, el programa recoge aquel número de filas de mayor rango y lo aplica a cada serie de datos. Por lo tanto, se debería especificar que la variable guarde aquellos datos que en cuyas filas solo existan valores no nulos de datos.

También, se debe cambiar el tipo de variable. Se pasa de un *DataFrame* (*pandas*) a un *array* (*NumPy*). Se realiza así porque los *array* (matrices) son los tipos de datos más convenientes (y eficientes) para realizar cálculos.

```
# Convierte  DataFrame en arrays de numpy:
flujo = df1["l/hm2"].as_matrix()
fr1   = df1["Pa/s"].as_matrix()
fr2   = df1["Pa/s.1"].as_matrix()
fr3   = df1["Pa/s.2"].as_matrix()
fr4   = df1["Pa/s.3"].as_matrix()

# Convierte el  flujo de l h-1m-2 a ms-1:
flujo = flujo * (1 / (1000*3600))

# Se guardan los array en una lista para ahorrar lineas de codigo
brfv = [0.00694, 0.00972, 0.0139, 0.0208]
frData   = [fr1, fr2, fr3, fr4]
```

Como se calcularán las velocidades de ensuciamiento (*FR* o *Fouling Rate* en inglés) teóricas, definiremos una función que nos permita hacer esto:

```
def fRate(parameters, flux, brfv, mlts=8.22):
    ''''' Nos proporciona FR teoricos a partir de datos de Flujo, cambiando
    el valor del brfv'''
    kf, beta1, beta2, gamma = parameters
    FR = kf * np.exp(flux * (beta1 * brfv + beta2 * mlts + gamma))
    return FR
```

En esta parte se realizará la calibración *off-line*. Esto es, obtener con los datos experimentales algunos de los parámetros necesarios para emplearlos, más tarde, en la simulación.

Se define una función para obtener valores simulados de *FR*. Esta función calcula valores simulados de *FR* en función de los parámetros introducidos. Después, se calcula su valor residual (diferencia del teórico con el simulado). Este valor residual es el que se minimiza. Cuando la función encuentra un mínimo residual, devuelve aquellos parámetros estimados "óptimos".

En este caso al tener distintos caudales de burbujeo, se construyen cuatro modelos de ensuciamiento para cada caudal de burbujeo (variable *brfv)*. Y, se calcula el residual para cada modelo, para finalmente realizar la minimización de la suma de los cuatro residuales.

```
def fit_function(params, flujo, frData, brfv):

    kf = params['kf'].value
    beta1 = params['beta1'].value
    beta2 = params['beta2'].value
    gamma = params['gamma'].value

    model1 = kf * np.exp(flujo *(beta1 * brfv[0] + beta2 * 8.22 + gamma))

    model2 = kf * np.exp(flujo *(beta1 * brfv[1] + beta2 * 8.22 + gamma))

    model3 = kf * np.exp(flujo *(beta1 * brfv[2] + beta2 * 8.22 + gamma))

    model4 = kf * np.exp(flujo *(beta1 * brfv[3] + beta2 * 8.22 + gamma))

    resid1 = frData[0] - model1
    resid2 = frData[1] - model2
    resid3 = frData[2] - model3
    resid4 = frData[3] - model4

    return np.concatenate((resid1, resid2, resid3, resid4))
```

Este es el diccionario de parámetros que se introducen en la función anterior. Como se puede observar, se le asigna un nombre y un valor inicial (como mínimo).

En el caso del *script* "dyn_calib_4.8.py", se puede también especificar si se quiere variar o no el parámetro, y si se quiere acotar (definiendo un valor mínimo y máximo):

```
params = Parameters()
params.add(   'kf', value=   5.6e-04)
params.add('beta1', value= -2.48e+08)
params.add('beta2', value=   5.1e+04)
params.add('gamma', value=  2.81e+06)
```

Dicha función de minimización de residuales se guarda en una variable, que al imprimirla muestra los parámetros óptimos estimados.

```
out = minimize(fit_function, params, args=(flujo, frData, brfv))
print(report_fit(out))
```

A continuación se procede a representar gráficamente los datos:

```
# Grafico:
# Listas de marcador y de parametros obtenidos con lmfit
lineStyle = ['bo', 'r^', 'gs', 'yp']
plmfit = [3.3e-02, -4.3e+06, -5.5e+04, 1.5e+06]

plt.figure(figsize=(8, 6))

# Representacion de los datos experimentales
for i in range(4):
    plt.plot(flujo, frData[i], lineStyle[i], label='%s'%(brfv[i]))

# Valores de Flujo para representar
x  = np.linspace(min(flujo), max(flujo) * 1.01, 50)
# Valores de FR calculados con los parametros estimados
y1, y2, y3, y4 = fRate(plmfit, x, brfv[0]), fRate(plmfit, x, brfv[1]), \
                 fRate(plmfit, x, brfv[2]), fRate(plmfit, x, brfv[3])
# Representación de los datos calculados
plt.plot(x, y1, 'b-', x, y2, 'r-', x, y3, 'g-', x, y4, 'y-')

plt.legend(loc='best', fontsize='medium',
           title=r'$BRF_v  (Nm^{3} s^{-1} m^{-3})$')

plt.xlabel(r'$J_{20} (m s^{-1})$', fontsize='x-large')
plt.ylabel(r'$FR (Pa s^{-1})$', fontsize='x-large')
plt.title("Ajuste del experimento flujo-escalón")
plt.ticklabel_format(style='sci', axis='x', scilimits=(0, 0))
plt.grid()
plt.margins(0.05)
```

```
plt.savefig("flujo-escalon.png", dpi=96)
plt.show()
plt.close()
```

8.2 *Script* "dyn_calib_4.8.py"

Este *script* hace la función de obtener los 4 parámetros k_{SF}, $\alpha_{C,0}$, TMP_a, y $q_{MS,max}$. Se describirán aquellas partes que difieran en relación al *Script* "flux_step.py".

En primer lugar, se definen las variables necesarias para la construcción del modelo. Estás variables son: variables por experimento (experimento 1 y 2), otras variables (propias del diseño del experimento), parámetros por defecto y los parámetros estimados en el fichero "flux_step.py".

Hay que destacar que al poseer dos series de datos, necesitamos ordenar al programa que datos quiere que importe (o variables por experimento). Para ello, al ejecutar el código, se le pedirá por consola que introduzca: "datos1" (referido al experimento 1) o "datos2" (referido al experimento 2).

```
# -*- coding: utf-8 -*-
"""
Programa que estima los 4 parametros:
qMS_max, TMPa, alphaC_0 y kSF

Se realiza con las funciones odeint(integra sistemas de ec. diferenciales)
y minimize(una funcion para ajuste de curvas)
"""
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import pandas as pd
from lmfit import minimize, Parameters, fit_report
import sys


# Variables por experimento
RM = np.array([5.27e+11, 6.32e+11]) # (m-1)
MLTS = np.array([5.52, 6,75])   # (kg m-3)
J20_l = np.array([12, 16])      # flujo permeado (l h m-2)
```

```
J20 = J20_l/ (1000 * 3600)      # flujo permeado (m3 s-1 m-2)

# Se elije que datos se quiere analizar
dataToAnalyze = input("Que datos quieres analizar?\
Introduce datos1 o datos2: ")   # datos 1 y 2 para J=12 y 16 lhm-2 respect.
if dataToAnalyze == "datos1":
    RM, MLTS, J20 = RM[0], MLTS[0], J20[0]
elif dataToAnalyze == "datos2":
    RM, MLTS, J20 = RM[1], MLTS[1], J20[1]
else:
    print("No se ha introducido nombre de datos correcto")
    sys.exit(1)

# Otras variables:
XTS = MLTS                      # (= MLTS(kg/mÂ³))
TMPconsigna = 28000             # (Pa)
J20_BF = 60 / (1000*3600)       # (m3 s-1 m-2)
tLavado = 30                    # (s)
BRFv = 1.65 / (0.003*60*1000)   # (Nm3 s-1 m-3)
mu = 0.001                      # viscosidad (kg m-1 s-1)
superfMemb = 0.047              # (m2)
Q20P = J20 * superfMemb         # (m3 s-1)
Q20_BF = J20_BF * superfMemb

# Parametros por defecto o que se pueden poner como tal
qBF_max = 1                     # por defecto en Robles
alphaI = 1e+14                  # por defecto en Robles
kt = 1                          # por defecto en Robles
KS_XmC = 0.2                    # estimado en Robles(largo plazo)
qIF_max = 3.0e-
07              # estimado en Robles(inversa de SRT, 38.5 d)

# Parametros estimados en el script flux_step.py
KF    =  3.3e-02
beta1 = -4.3e+06
beta2 = -5.5e+04
gamma =  1.5e+06

FR = KF * np.exp(J20 * (beta1 * BRFv + beta2 * MLTS + gamma))
IMS = 1 / (1 + FR)
```

En este caso, al tener columnas (series) de datos de distinto número de filas, debemos ignorar aquellas filas que contengan celdas vacías. También aquello valores negativos de *TMP* se le reasignarán el valor cero.

```
# PARTE I: Importacion y tratamiento de datos-----------------------------
```

```
fil_df = pd.read_excel("TFGPy.xlsm", sheetname="Experimentales", skiprows=7,
                       header=1)
df1 = fil_df.copy()

# Asignar columnas de datos a variables
if dataToAnalyze == "datos1":
    t,tmp = df1.t1, df1.TMP1
elif dataToAnalyze == "datos2":
    t,tmp = df1.t2, df1.TMP2

# Se tratan los datos
t   = t[~t.isnull()]      # selecciona valores no "nulos"(NaNs)
tmp = tmp[~tmp.isnull()]
tmp[tmp < 0] = 0          # aquellos valores negativos se fijan como 0

# Se convierten los objetos Pandas a arrays de NumPy
t   = t.as_matrix().astype(int)
tmp = tmp.as_matrix()

# Definir el rango de presiones hasta la primera etapa de filtracion
for i in range(1, len(tmp)-1):
    if tmp[i] == 0:
        TMPexp = tmp[:i]
        t1     = t[:i]
        break
```

En un principio se definen las variables que se calcularán en la simulación. Todas estas variables son del tipo *arrays* de *numpy* de tipo *float* y con rango de igual tamaño que la variable tiempo. Al inicializar un vector se le puede asignar valores aleatorios o valores cero. En este caso se han puesto valores cero.

```
# PARTE II: CALIBRACION DINAMICA-------------------------------------------

# Definicion de los vectores:
XmI    = np.zeros(t1.shape, np.float)
omegaI = np.zeros(t1.shape, np.float)
XmC    = np.zeros(t1.shape, np.float)
omegaC = np.zeros(t1.shape, np.float)
alphaC = np.zeros(t1.shape, np.float)
alphaC[0] = alphaC_0   # el parametro estimado representa su valor a t=0
TMPsim = np.zeros(t1.shape, np.float)

# Define una funcion cuyos parametros queremos obtener, para luego poder
# simular la TMP y poder comparar con los TMP experimentales. Se
# obtendran aquellos parametros con el menor resiudal.
```

```
def fit_function(params, TMPexp, t):

    qMS_max  = params['qMS_max'].value
    TMPa     = params['TMPa'].value
    alphaC_0 = params['alphaC_0'].value
    kSF      = params['kSF'].value


    def dXmCdt(y, t):
        XmC = y[0]
        dXmCdt = Q20P * XTS - (qMS_max * IMS * BRFv *
                               (XmC / (KS_XmC + XmC)) * XmC) - qIF_max * XmC
        dXmIdt = qIF_max * XmC

        return [dXmCdt, dXmIdt]


    def dalphaCdt(alphaC, t, TMPsim):
        alphaC_TMP = alphaC_0 * (1 +(TMPsim / TMPa))
        dalphaCdt = max(kSF, kt * (alphaC_TMP - alphaC))
        return dalphaCdt


    for i in range(1, len(TMPexp)):

        tspan = [t1[i-1], t1[i]]
        y0    = [XmC[i-1], XmI[i-1]]

        sol_XmC = odeint(dXmCdt,y0, tspan, rtol=1e-3,atol=1e-3)
        sol_alphaC = odeint(dalphaCdt, alphaC[i-1], tspan,
                            args=(TMPsim[i-1],), rtol=1e-3,atol=1e-3)

        XmC[i]    = (sol_XmC[-1,0])
        XmI[i]    = (sol_XmC[-1,1])
        alphaC[i] = (sol_alphaC[-1, 0])

        TMPsim[i] = ( J20 * mu * (RM + (XmC[i] / superfMemb) * (alphaC[i]) +

                                    (XmI[i] / superfMemb) * alphaI) )

    resid1 = (TMPsim - TMPexp)

    return resid1

# Diccionario de parametros a estimar en el modelo
p = Parameters()
#           (      Name,    Value,  Vary,      Min,      Max, Expr)
p.add_many( ( 'qMS_max',     6.31,  True,        1,       10, None),
            (    'TMPa',    18900,  True,    5e+03,    5e+04, None),
            ('alphaC_0', 1.02e+13,  True,    1e+10,    1e+18, None),
```

```
            (     'kSF', 4.09e+10,  True,    4e+07,    4e+13, None) )

out = minimize(fit_function, p, args=(TMPexp, t1))
print(fit_report(out))

# Grafico del modelo y experimental

plt.plot(t1, TMPsim,'r-', label='TMPsim')
plt.plot(t1, TMPexp, 'b-', label='TMPexp')
plt.legend(loc='best')
plt.savefig("Cal_dyn %s.png"%(dataToAnalyze))
plt.show()
plt.close()
```

8.3 *Script* "exp_vs_sim_4.8.py"

A este último *script* se le debe introducir los parámetros estimados para "flux_step.py" y "dyn_calib_4.8.py". También, al ejecutar el código se pedirá pro pantalla escribir si deseamos trabajar con los "datos1" o los "datos2".

La función de este fichero es realizar la simulación para el rango completo de tiempo (tiempo total igual al experimental) obteniendo: gráfico de la filtración simulada y la filtración experimental, valores de ensuciamiento reversible, irreversible y resistencia específica media, tiempos de cada etapa de filtración, tanto simulada como experimental.

```
"""
Project: Representacion de todo el rango de datos exp. Con los parametros
estimados, representar tambien todo el rango de datos simulado.
Comparar graficos.
"""
import pandas as pd
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import sys

# Variables por experimento
RM = np.array([5.27e+11, 6.32e+11]) # (m-1)
MLTS = np.array([5.52, 6,75])   # (kg m-3)
J20_l = np.array([12, 16])      # flujo permeado (l h-1 m-2)
J20 = J20_l/ (1000 * 3600)      # flujo permeado (m3 s-1 m-2)
```

```
# Se elije que datos se quiere analizar
dataToAnalyze = input("Que datos quieres analizar?\
Introduce datos1 o datos2: ")   #datos 1 y 2 para J=12 y 16 lh-1m-2 respec.
if dataToAnalyze == "datos1":
    RM, MLTS, J20 = RM[0], MLTS[0], J20[0]
elif dataToAnalyze == "datos2":
    RM, MLTS, J20 = RM[1], MLTS[1], J20[1]
else:
    print("No se ha introducido nombre de datos correcto")
    sys.exit(1)

# Otras variables
XTS = MLTS                          # (= MLTS(kg/mÂ³))
TMPconsigna = 28000                 # (Pa)
J20_BF = 60 / (1000*3600)           # (m3Â·s-1Â·m-2)
tLavado = 30                        # (s)
BRFv = 1.65 / (0.003*60*1000)       # (Nm3Â·s-1Â·m-3)
mu = 0.001                          # viscosidad
superfMemb = 0.047                  #
Q20P = J20 * superfMemb
Q20_BF = J20_BF * superfMemb

# Parametros por defecto o que se pueden poner como tal:
qBF_max = 1                         # por defecto en Robles
alphaI = 1e+14                      # por defecto en Robles
kt = 1                              # por defecto en Robles
KS_XmC = 0.2                        # estimado en Robles(largo plazo)
qIF_max = 3.0e-07                   # inversa del SRT de operacion

# Parametros estimados en el script flux_step.py
KF    =  3.3e-02
beta1 = -4.3e+06
beta2 = -5.5e+04
gamma =  1.5e+06

FR = KF * np.exp(J20 * (beta1 * BRFv + beta2 * MLTS + gamma))
IMS = 1 / (1 + FR)

# Parametros estimados en el script dyn_calib_4.8.py
qMS_max  = np.array([1.0, 1])
TMPa     = np.array([18400, 20000])
alphaC_0 = np.array([4.196e+13, 1e+13])
kSF      = np.array([9.2e+09, 2.45e+10])

if dataToAnalyze == "datos1":
    qMS_max, TMPa, alphaC_0, kSF = qMS_max[0], TMPa[0], alphaC_0[0], kSF[0]
elif dataToAnalyze == "datos2":
    qMS_max, TMPa, alphaC_0, kSF = qMS_max[1], TMPa[1], alphaC_0[1], kSF[1]


# PARTE I: Importacion y tratamiento de datos-----------------------------
```

```
fil_df = pd.read_excel("TFGPy.xlsm", sheetname="Experimentales", skiprows=7,
                       header=1)
df1 = fil_df.copy()

# Asignar columnas de datos a variables
if dataToAnalyze == "datos1":
    t,tmp = df1.t1, df1.TMP1
elif dataToAnalyze == "datos2":
    t,tmp = df1.t2, df1.TMP2

# Se tratan los datos
t   = t[~t.isnull()]      # selecciona valores no "nulos"(NaNs)
tmp = tmp[~tmp.isnull()]
tmp[tmp < 0] = 0          # aquellos valores negativos se fijan como 0

# Se convierten los objetos panda a arrays de NumPy
t   = t.as_matrix().astype(int)
tmp = tmp.as_matrix()


# PARTE II: Simulacion-----------------------------------------------------

# 1. Condiciones iniciales para TMPsim, XmC, XmI,  aplha_C

XmI    = np.zeros(t.shape, np.float)
omegaI = np.zeros(t.shape, np.float)
XmC    = np.zeros(t.shape, np.float)
omegaC = np.zeros(t.shape, np.float)
alphaC = np.zeros(t.shape, np.float)
alphaC[0] = alphaC_0   # el parametro estimado representa su valor a t=0
TMPsim = np.zeros(t.shape, np.float)

# Etapa de filtracion/lavado


def dXmCdt(y, t):
    '''''ec. formacion de la torta durante filtrado'''
    XmC = y[0]
    dXmCdt = Q20P * XTS - (qMS_max * IMS * BRFv *
                                (XmC / (KS_XmC + XmC)) * XmC) - qIF_max * XmC
    dXmIdt = qIF_max * XmC

    return [dXmCdt, dXmIdt]


def dXmCdt1(y, t):
    ''''' Process 3: cake layer detachment during back-flushing '''
    XmC = y[0]
    dXmCdt = - qBF_max * Q20_BF * (XmC / (KS_XmC + XmC)) * XmC
    dXmIdt = 0
```

```
    return [dXmCdt, dXmIdt]


def dalphaCdt(alphaC, t, TMPsim):
    alphaC_TMP = alphaC_0 * (1 + (TMPsim / TMPa))
    dalphaCdt  = max(kSF, kt * (alphaC_TMP - alphaC))
    return dalphaCdt
```

Se procede a simular la filtración. Para ello utilizamos un contador (variable *i*) y bucles *while/else* e *if/else*.

La simulación contendrá el mismo rango de datos que el rango de tiempo experimental. Es decir, si los datos van del tiempo 0 hasta 1400, y se recogieron datos cada 10 segundos, tendremos un rango de 140 elementos.

Mientras que la *TMP* simulada sea inferior a la de consigna, se ejecutará el código que simula los procesos que ocurren durante la filtración.

```
i = 0
while i < (len(t)-1):

    while TMPsim[i] < TMPconsigna:   # Filtracion
```

Como los procesos están descritos por ecuaciones diferenciales, se necesitará resolver dichas ecuaciones para cada elemento de tiempo. Las soluciones de las ecuaciones diferenciales para cada intervalo de tiempo se guardarán en cada elemento de las variables que se inicializaron anteriormente.

```
        tspan = np.linspace(t[i], t[i+1], 20)
        y0    = [XmC[i], XmI[i]]

        sol_XmC    = odeint(dXmCdt, y0, tspan, rtol=1e-3, atol=1e-3)
        sol_alphaC = odeint(dalphaCdt, alphaC[i], tspan,
                            args=(TMPsim[i],), rtol=1e-3, atol=1e-3)

        XmC[i+1]    = (sol_XmC[-1, 0])
        XmI[i+1]    = (sol_XmC[-1, 1])
        alphaC[i+1] = (sol_alphaC[-1, 0])
```

```
        TMPsim[i+1] = (J20 * mu * (RM + (XmC[i+1] / superfMemb) *
                          (alphaC[i+1]) + (XmI[i+1] / superfMemb) * alphaI))
```

Una vez calculada la *TMP* simulada se aumenta el contador en una unidad. A continuación se comprueba si el siguiente elemento (*i*) con el que volveremos a calcular la *TMP* simulada, es igual al elemento final del rango de tiempo. Si es así, la simulación termina.

Si se alcanza la *TMP* de consigna se procede a ejecutar el código que simula los procesos de lavado. En este caso, mientras que transcurre el lavado, la *TMP* es cero. El lavado se producirá mientras que el valor del tiempo transcurrido sea inferior al tiempo de lavado.

```
        i += 1
        if i == (len(t)-1):
            break

    else:

        TMPsim[i] = 0
        i = i - 1
        tInicLavado = t[i]
        while np.around(t[i] - tInicLavado) < 30: #tlavado inferior a 30 s

            tspan = np.linspace(t[i], t[i+1], 20)
            y0    = [0, XmI[i]]

            sol_XmC1   = odeint(dXmCdt1, y0, tspan, rtol=1e-3, atol=1e-3)
            sol_alphaC = odeint(dalphaCdt, alphaC[i], tspan,
                                args=(TMPsim[i],), rtol=1e-3, atol=1e-3)

            XmC[i+1]    = sol_XmC1[-1, 0]
            XmI[i+1]    = sol_XmC1[-1, 1]
            alphaC[i+1] = sol_alphaC[-1, 0]

            TMPsim[i+1] = 0

            i += 1
            if i == (len(t)-1):
                break

# Graficos Filtracion: dos subplots, se comparten ejes X e Y.
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, sharey=True,
                               figsize=(8,6))
ax1.plot(t, TMPsim, 'b-')               # se crea subplot 1
ax1.set_title("Filtración-Simulación")
```

```
ax2.plot(t, tmp, 'r-')                  # se crea subplot 2
ax2.set_title("Filtración-Experimental")

#Se crean titulos unicos para cada Eje
fig.text(0.5, 0.04, 'Tiempo (s)', ha='center', va='center')
fig.text(0.04, 0.5, 'TMP (Pa)', ha='center', va='center',
         rotation='vertical')

plt.savefig("filtracion %s.png"%(dataToAnalyze), dpi=96,
            bbox_inches='tight')

plt.show()
plt.close()

# Graficos Ensuciamiento: dos subplots, se comparte eje X.
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(8,6))

ax1.plot(t, (XmC/superfMemb), 'y--')
ax1.set_title("Ensuciamiento Reversible")
ax1.yaxis.get_major_formatter().set_powerlimits((0, 1)) # formato cient. eje

ax2.plot(t, (XmI/superfMemb), 'g--')
ax2.set_title("Ensuciamiento Irreversible")
ax2.yaxis.get_major_formatter().set_powerlimits((0, 1))

fig.text(0.5, 0.04,'Tiempo (s)', ha='center', va='center')
fig.text(0.04, 0.5, r'$\omega (kg m^{-2})$', ha='center', va='center',
         rotation='vertical', fontsize='x-large')

plt.savefig("ensuciamiento %s.png"%(dataToAnalyze), dpi=96,
            bbox_inches='tight')
plt.show()
plt.close()
```

Aparte, se ha creado una pequeña función que compara los tiempos de filtración de cada etapa. Así podemos comprobar si los tiempos experimentales de filtración son iguales a los simulados(o cercanos).

```
# Tiempos experimentales y simulados de filtracion
tFiltExp = tFiltr(t, tmp, tLavado)
tFiltSim = tFiltr(t, TMPsim, tLavado)

# Graficos
fig, ax = plt.subplots()
barWidth = 0.35
opacity = 0.6
plt.bar(np.arange(1, len(tFiltExp) + 1), tFiltExp, barWidth, alpha=opacity,
```

```
        color='red', label='Experimental')
plt.bar(np.arange(1, len(tFiltSim) + 1) + barWidth, tFiltSim, barWidth,
        alpha=opacity, color='blue', label='Simulado')
plt.legend()
plt.title("Tiempos de filtraciÃ³n por nÃºmero de fÃltrado")
plt.xlabel("NÃºmero de filtrado")
plt.xticks(np.arange(1, len(tFiltSim) + 2) + barWidth,
           (np.arange(1, len(tFiltSim) + 2)))
plt.ylabel("Tiempo (s)")
plt.tight_layout()
plt.show()
plt.close()
```

9 Resultados

9.1 Calibración del modelo

En el modelo hay diversos parámetros que se deben ajustar para realizar la simulación de la filtración. La estimación de parámetros se basa en su obtención a través de datos disponibles: medidas de MLTS, TMP y caudal de biogás recirculado.

Para la estimación de los diversos parámetros propuestos en el modelo se han realizado diferentes procedimientos: calibración "off-line" por experimentos de flujo en escalón y calibración dinámica. Aparte se han establecido unos valores por defecto.

9.1.1 Calibración "off-line" a corto plazo

Los parámetros necesarios para la obtención de la velocidad de ensuciamiento (FR), en el modelo propuesto, se han determinado con ayuda de una serie de datos de experimentos realizados por el método de flujo escalonado. Estos datos experimentales se tienen para diferentes caudales de biogás que permanecen constantes para cada experimento mientras se va incrementando progresivamente el flujo de permeado, J_{20}. En cada uno de ellos se puede observar cómo va aumentando la velocidad de ensuciamiento a medida que aumenta el flujo. Cabe mencionar que todos los experimentos se han determinado con una concentración de MLTS aproximado de 8,22 mg/L.

Como se ha mencionado con anterioridad, en el modelo propuesto se ha definido una expresión que permite describir el comportamiento de la velocidad de ensuciamiento (ver ec. 11). Los parámetros que se quieren conocer mediante este procedimiento son: β_1 ($s^2\ m^{-1}$), β_2 ($s\ m^2\ kg^{-1}$), γ ($s\ m^{-1}$) y K_F ($Pa\ s^{-1}$). Con estos valores ya se podría predecir la velocidad de ensuciamiento de una membrana en condiciones similares a las empleadas en estos

experimentos: caudal de biogás empleado, concentración de sólidos e intervalo de flujo de permeado en el que se han obtenido la evolución del *FR*.

Para la estimación de estos parámetros se ha procedido a realizar una minimización de una función objetivo (residual = modelo – experimental), que devuelve los valores residuales conjuntos de cada experimento. Por tanto, los parámetros obtenidos son aquellos que mejor se ajustan al rango de caudal de burbujeo realizado entre 0.00694 y 0.0208 Nm3 s-1 m-3.

Con Python se han obtenido los siguientes resultados:

```
[[Fit Statistics]]
    # function evals   = 148
    # data points      = 28
    # variables        = 4
    chi-square         = 128.429
    reduced chi-square = 5.351
[[Variables]]
    kf:      0.03265925 +/- 0.008538 (26.14%) (init= 0.00056)
    beta1:  -4.2923e+06 +/- 4.00e+05 (9.31%) (init=-2.48e+08)
    beta2:  -55238.9035 +/- nan      (nan%) (init= 51000)
    gamma:   1.5040e+06 +/- nan      (nan%) (init= 2810000)
[[Correlations]] (unreported correlations are <  0.100)
None
```

Figura 6. Informe de los parámetros estimados para FR

Seguidamente, dichos parámetros se han introducido en la ec. 11, y se han obtenido las curvas ajustadas (representadas con una línea coloreada). Estos resultados se representan en la Figura 7. El modelo representa adecuadamente los resultados experimentales y reproduce la condición de flujo crítico ($J_{crítico}$), entendido éste como aquel flujo de permeado a partir del cual la velocidad de ensuciamiento (*FR*) crece exponencialmente. De acuerdo con la Figura 7 el valor del $J_{crítico}$ puede establecerse aproximadamente en $3{,}5 \cdot 10^{-6}$ m·s^{-1} (12,6 L h^{-1} m^{-2}). Teniendo en cuenta este valor, los experimentos con los que se calibrará dinámicamente el modelo y se ensayará la simulación corresponden a valores ligeramente inferiores al flujo crítico (Experimento 1: J = 12 L h^{-1} m^{-2}) y superiores al flujo crítico (Experimento 2: J = 16 L h^{-1} m^{-2})

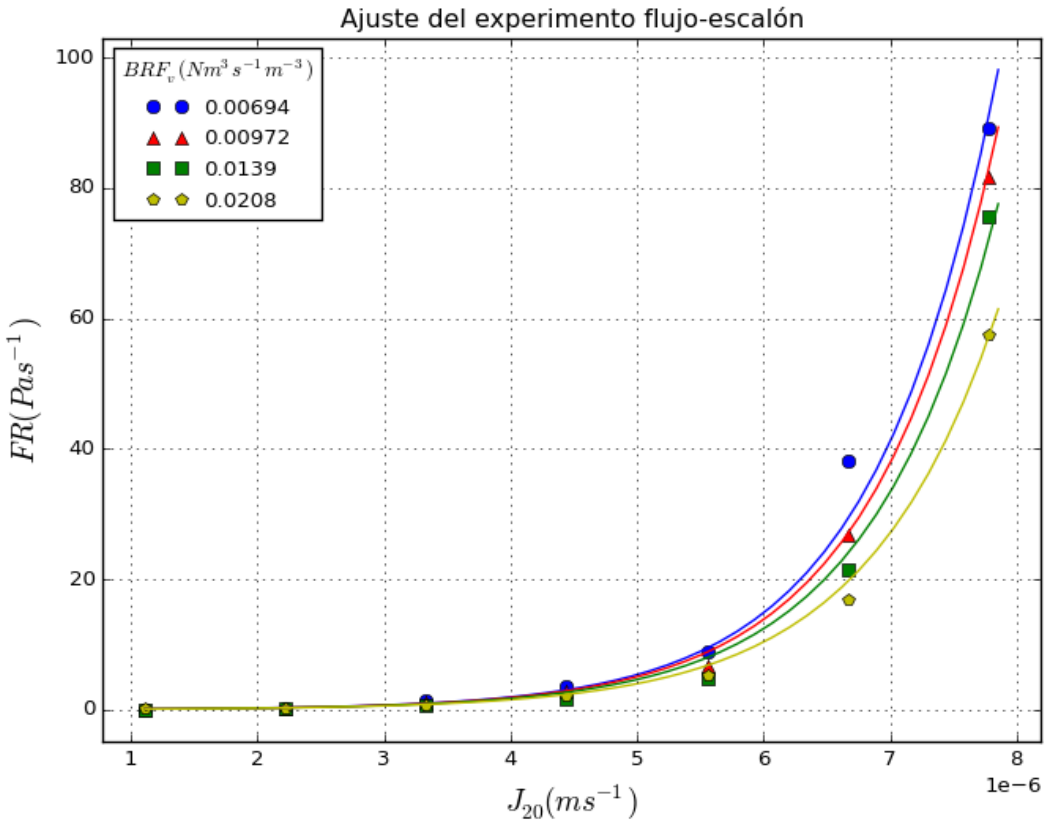

Figura 7. Ajuste de las curvas de velocidad de ensuciamiento (datos experimentales recogidos en el Anexo 1).

9.1.2 Calibración dinámica a corto plazo

En el modelo propuesto hay una serie de parámetros que deben establecerse mediante una calibración dinámica. La calibración dinámica consiste en ajustar los valores de TMP simulados (TMP_{sim}) a los valores experimentales de TMP (TMP_{exp}).

Los parámetros que se van a ajustar son los siguientes: k_{SF}, $\alpha_{C,0}$, TMP_a, y $q_{MS,max}$. Estos parámetros están relacionados con la formación y compresión de la torta que se crea en la superficie de la membrana (proceso 1) y con la limpieza por la acción del biogás empleado (proceso 2).

La metodología seguida es parecida a la realizada en la calibración “off-line”, pero a diferencia de esta, tanto las variables de entrada en el modelo, como las variables de salida varían con el tiempo.

En este apartado se debe clarificar que los parámetros obtenidos se realizaron solo con la primera etapa de filtración. Se ha realizado de este modo debido a la complejidad de la programación. Es recomendable, si se sigue con este proyecto, que se realice una estimación para todo el rango de datos. Los resultados obtenidos se indican en la Figura 8. La Figura 9 muestra el ajuste del modelo a los datos experimentales de la primera etapa de filtración, usando los parámetros obtenidos en la calibración.

```
[[Fit Statistics]]
    # function evals   = 403
    # data points      = 442
    # variables        = 4
    chi-square         = 88127557.414
    reduced chi-square = 201204.469
[[Variables]]
    qMS_max:    1.00006400 +/- 0.114657 (11.47%) (init= 6.31)
    TMPa:       18389.0002 +/- 283.9026 (1.54%) (init= 18900)
    alphaC_0:   4.1957e+13 +/- 7.17e+10 (0.17%) (init= 1.02e+13)
    kSF:        9.2468e+09 +/- 2.01e+08 (2.17%) (init= 4.09e+10)
[[Correlations]] (unreported correlations are <  0.100)
    C(qMS_max, TMPa)             = -0.975
    C(TMPa, kSF)                 = -0.280
    C(TMPa, alphaC_0)            =  0.275
    C(qMS_max, kSF)              =  0.257
    C(alphaC_0, kSF)             = -0.198
```

```
[[Fit Statistics]]
    # function evals   = 106
    # data points      = 271
    # variables        = 4
    chi-square         = 48892088.921
    reduced chi-square = 183116.438
[[Variables]]
    qMS_max:    1.00000003 +/- 1.644940 (164.49%) (init= 6.31)
    TMPa:       19543.0017 +/- 2.64e+06 (13530.75%) (init= 18900)
    alphaC_0:   1.0411e+13 +/- 1.76e+14 (1692.84%) (init= 1.02e+13)
    kSF:        2.4504e+10 +/- 9.36e+08 (3.82%) (init= 4.09e+10)
[[Correlations]] (unreported correlations are <  0.100)
    C(TMPa, alphaC_0)            =  1.000
    C(qMS_max, kSF)              = -0.958
```

Figura 8. Informe de los parámetros estimados para los Procesos 1 y 2. Exp. 1 (arriba) y Exp. 2 (abajo)

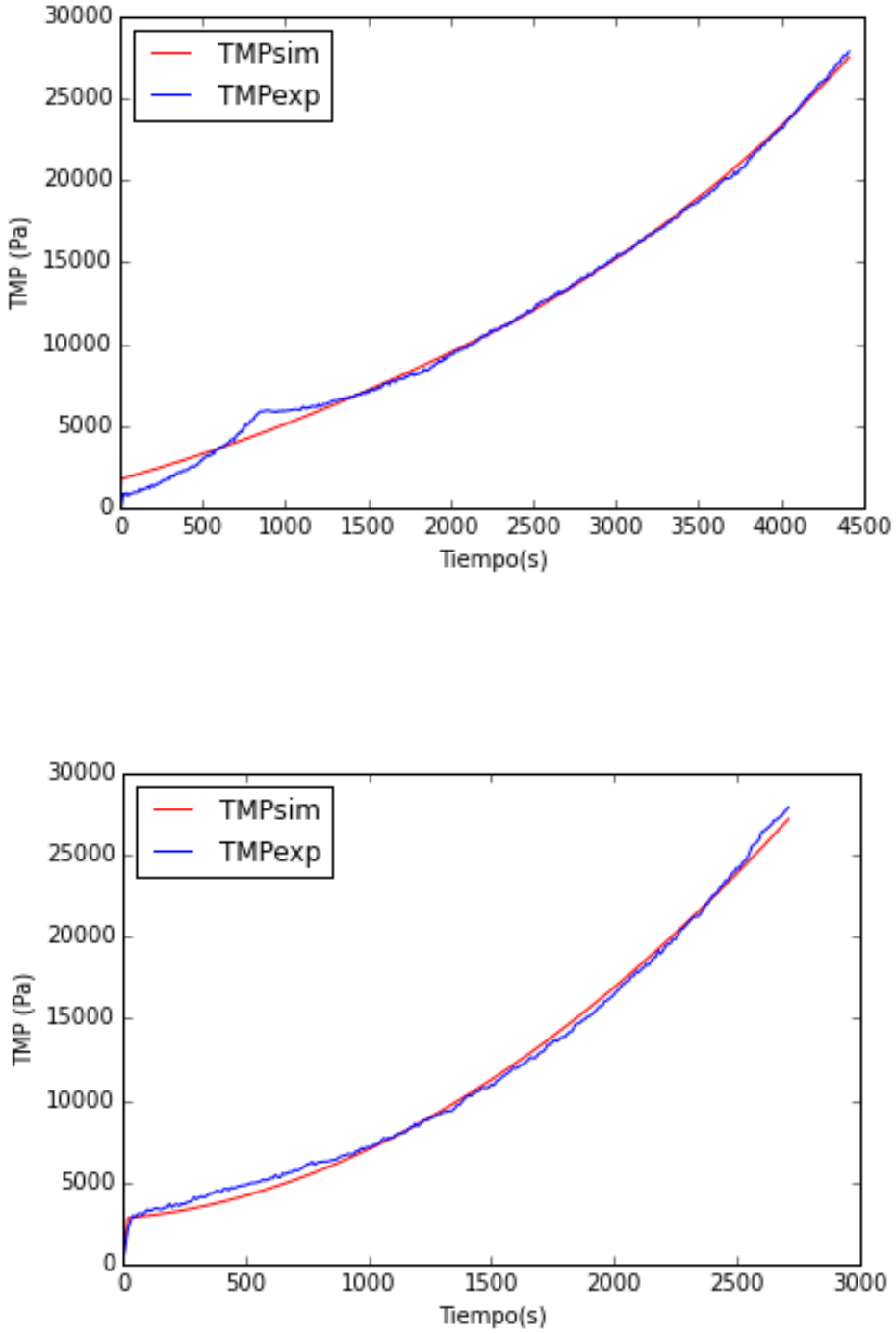

Figura 9. Ajuste de datos experimentales de TMP. Exp. 1 (arriba) y Exp. 2 (abajo)

9.1.3 Valores por defecto

Se puede observar que hay una serie de parámetros de los que no se tiene conocimiento y necesitan más investigación. Debido a la falta de una metodología que permita conocer estos valores, se establecen valores por defecto (tomados de Robles et al. [4]). Los valores establecidos por defecto son: $q_{BF,\,max}$, α_I, k_t, $K_{S,XmC}$, y $q_{IF,\,max}$. No obstante cabe señalar que los mismos autores, mediante un análisis de sensibilidad basado en el método de Morris, determinaron que estos parámetros son los que tienen menor influencia en el modelo y, por tanto, pueden ser ajustados a valores por defecto con objeto de simplificar la calibración [10].

A continuación se muestra la Tabla 3 donde se indica los parámetros empleados, y los valores obtenidos por las distintas vías que se han detallado en los apartados anteriores.

Parámetro	Unidad	Valor (Exp.1)	Valor (Exp.2)	Método de cálculo
K_F	Pa s^{-1}	$3{,}3 \cdot 10^{-2}$	$3{,}3 \cdot 10^{-2}$	Calibrado “off-line”
β_1	$s^2\ m^{-1}$	$-4{,}3 \cdot 10^{6}$	$-4{,}3 \cdot 10^{6}$	Calibrado “off-line”
β_2	$s\ m^2\ kg^{-1}$	$-5{,}5 \cdot 10^{4}$	$-5{,}5 \cdot 10^{4}$	Calibrado “off-line”
Υ	$s\ m^{-1}$	$1{,}5 \cdot 10^{6}$	$1{,}5 \cdot 10^{6}$	Calibrado “off-line”
$q_{MS,max}$	-	1	1	Calibrado dinámico
TMP_a	Pa	18 400	20 000	Calibrado dinámico
$\alpha_{C,0}$	$m\ kg^{-1}$	$4{,}196 \cdot 10^{13}$	$1 \cdot 10^{13}$	Calibrado dinámico
k_{SF}	$m\ kg^{-1}\ s^{-1}$	$9{,}2 \cdot 10^{9}$	$2{,}45 \cdot 10^{10}$	Calibrado dinámico
$q_{BF,max}$	m^{-3}	1	1	Valor por defecto
$q_{IF,max}$	s^{-1}	$3 \cdot 10^{-7}$	$3 \cdot 10^{-7}$	Valor por defecto
$K_{S,\,Xmc}$	kg	0,2	0,2	Valor por defecto
kt	s^{-1}	1	1	Valor por defecto
α_I	$m\ kg^{-1}$	$1 \cdot 10^{14}$	$1 \cdot 10^{14}$	Valor por defecto

Tabla 3. Valores empleados para los diferentes parámetros que se incluyen en la simulación

9.2 Simulación

Para la simulación, se han utilizado las ecuaciones del Apartado 7.5 (Operación de filtración) y los parámetros contenidos en la Tabla 3.

Normalmente, la operación de filtración transcurre mientras no se alcance una presión transmembrana de consigna ($TMP_{consigna}$). Una vez la $TMP_{consigna}$ es alcanzada, se pasa a un periodo de 30 segundos de lavado de la membrana, para que, a continuación se siga con el proceso de filtración. La filtración finaliza cuando el tiempo total de operación sea igual al recogido experimentalmente.

Los resultados se han obtenido para el experimento 1 (Flujo de 12 L h^{-1} m^{-2}) y el experimento 2 (Flujo, 16 L h^{-1} m^{-2}):

Las Figura 10 y 11 muestran la evolución de la TMP experimental y simulada. Se observa un mejor ajuste de los resultados a las condiciones del experimento 1, reproduciendo el modelo aproximadamente el mismo número de ciclos de filtración a lo largo de todo el tiempo de duración del ensayo. Para el experimento 2 el modelo prevé un mayor número de ciclos de filtración, lo que refleja un ensuciamiento reversible mayor que el experimental.

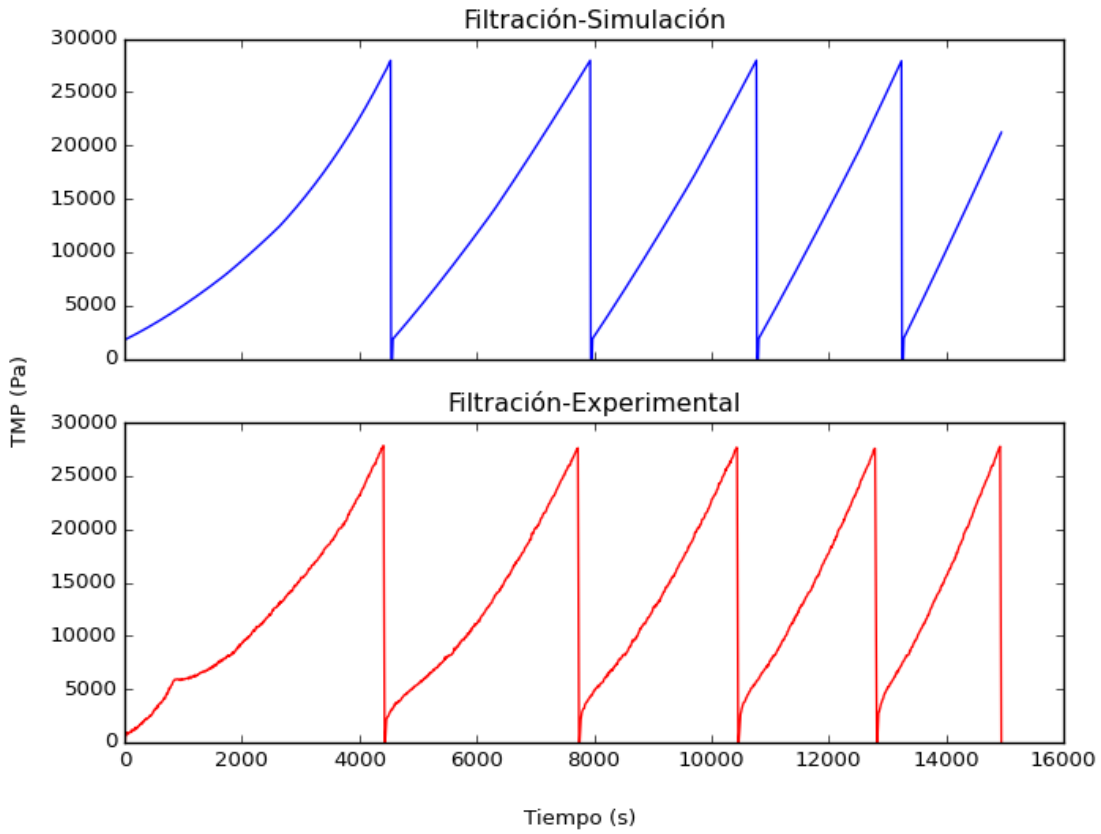

Figura 10. Filtración simulada (arriba) y filtración experimental (debajo) del Experimento 1 (J = 12 L h^{-1} m^{-2}).

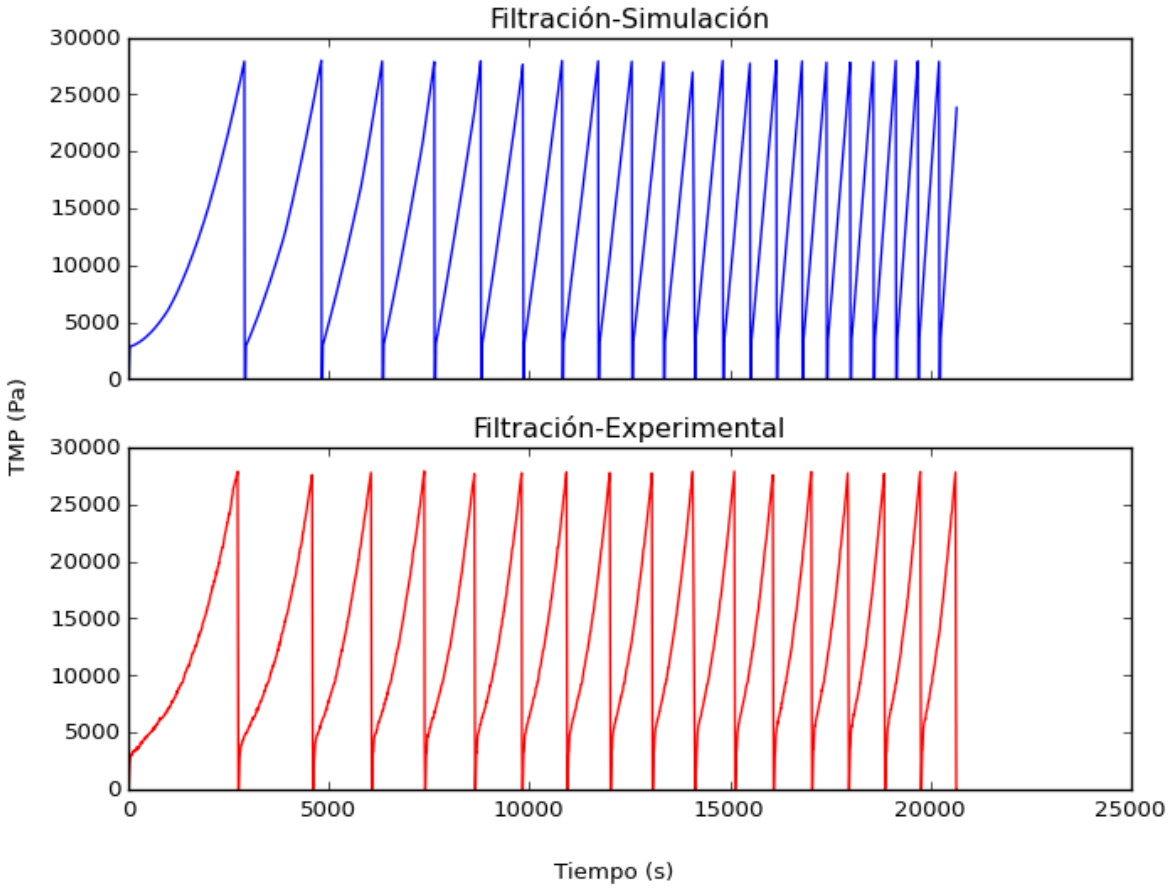

Figura 11. Filtración simulada (arriba) y filtración experimental (debajo) del Experimento 2 (J = 16 L h^{-1} m^{-2}).

El modelo permite obtener también una predicción de la evolución del ensuciamiento reversible e irreversible (Figura 12 y 13). Se observa como, a medida que transcurren los ciclos de filtración, la masa de torta asociada al ensuciamiento reversible, ω_C, va siendo cada vez menor. Esto puede deberse a que una menor duración del ciclo de filtración conduce a una menor deposición de sólidos sobre la superficie. Sin embargo, esa menor cantidad de sólidos produce la misma TMP al final del ciclo ($TMP_{consigna}$). Esto no puede atribuirse a la masa asociada al ensuciamiento irreversible ya que es 2-3 órdenes de magnitud menor. Por tanto, el hecho sólo podría explicarse si la compactación de la torta, esto es, su resistencia específica, aumentara en cada ciclo.

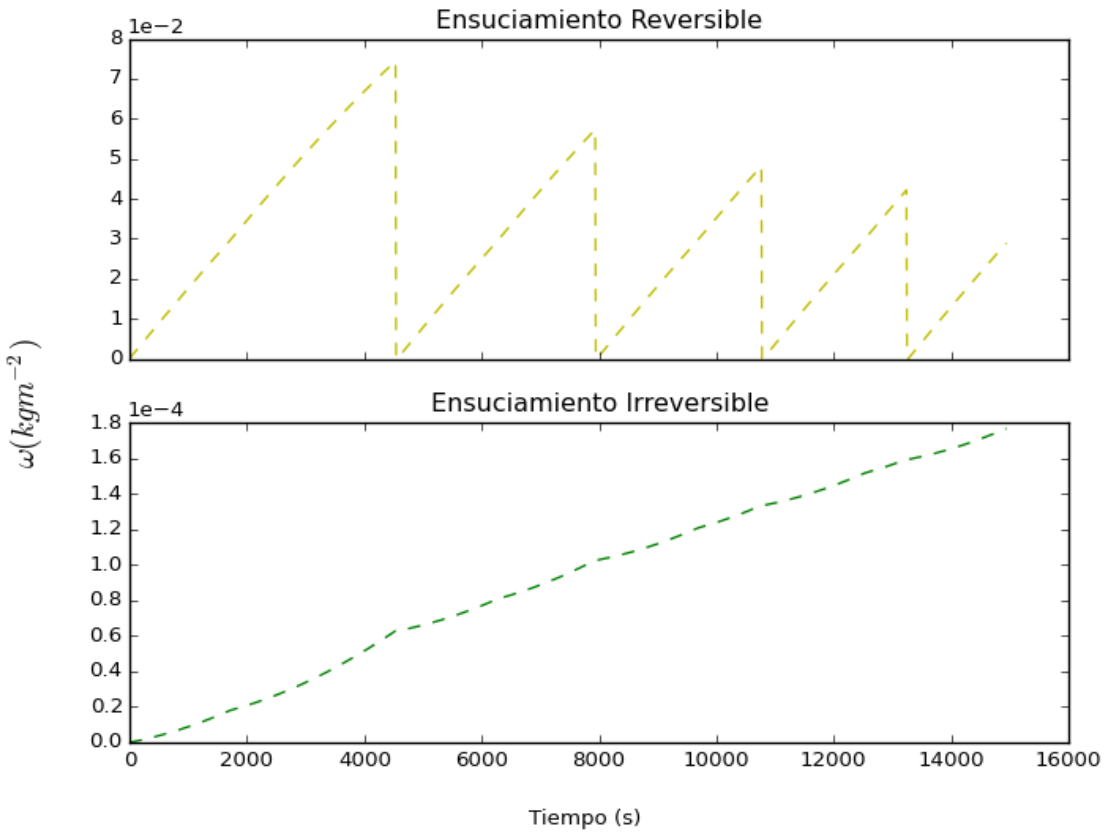

Figura 12. Ensuciamiento reversible (ω_C) y ensuciamiento irreversible (ω_I) en la simulación del Exp. 1 (J = 12 L h^{-1} m^{-2})

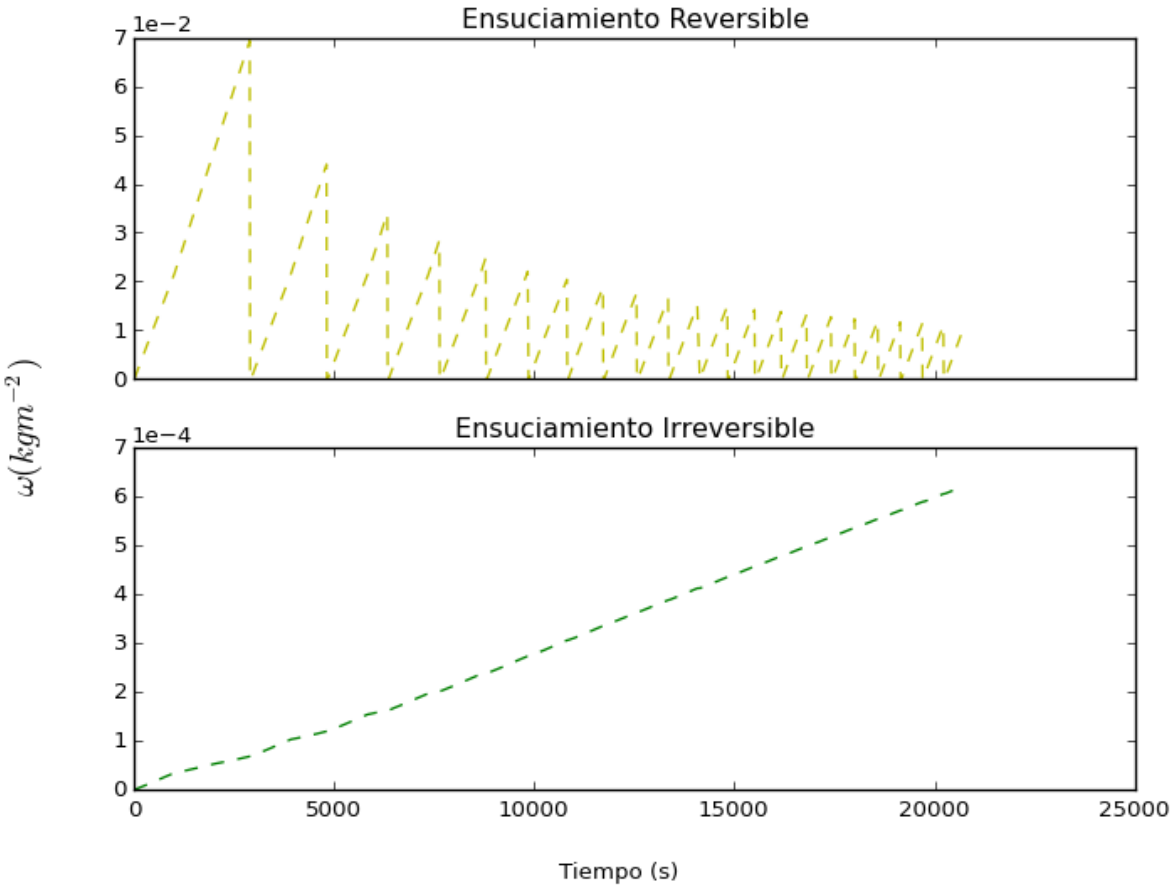

Figura 13. Ensuciamiento reversible (ω_C) y ensuciamiento irreversible (ω_I) en la simulación del Experimento 2 (J = 16 L h^{-1} m^{-2})

Por último, se debe indicar que el *script* es capaz de proporcionar los tiempos de filtrado experimental y simulados. Estos resultados se muestran en la Figura 14. Como puede observarse, a medida que aumenta el número de ciclos de filtrado, la duración del ciclo disminuye. Se aprecia que, para el experimento de menor flujo de permeado (J = 12 L h^{-1} m^{-2}), los tiempos de filtrado experimental y simulado son similares, lo que indica un ajuste adecuado del modelo en estas condiciones. Para el experimento de mayor flujo de permeado (J = 16 L h^{-1} m^{-2}) se observa un buen ajuste del modelo en los primeros ciclos de filtración, para a continuación desviarse progresivamente, prediciendo ciclos de filtrado más cortos que los experimentales.

Cabe señalar que el flujo en el Experimento 1 es ligeramente inferior al crítico, mientras que en el Experimento 2, el flujo es superior a este valor. Se podría concluir entonces que el modelo parece representar adecuadamente el comportamiento experimental a corto plazo en condiciones subcríticas, pero que requeriría modificaciones para su aplicación a flujos de permeado superiores al flujo crítico.

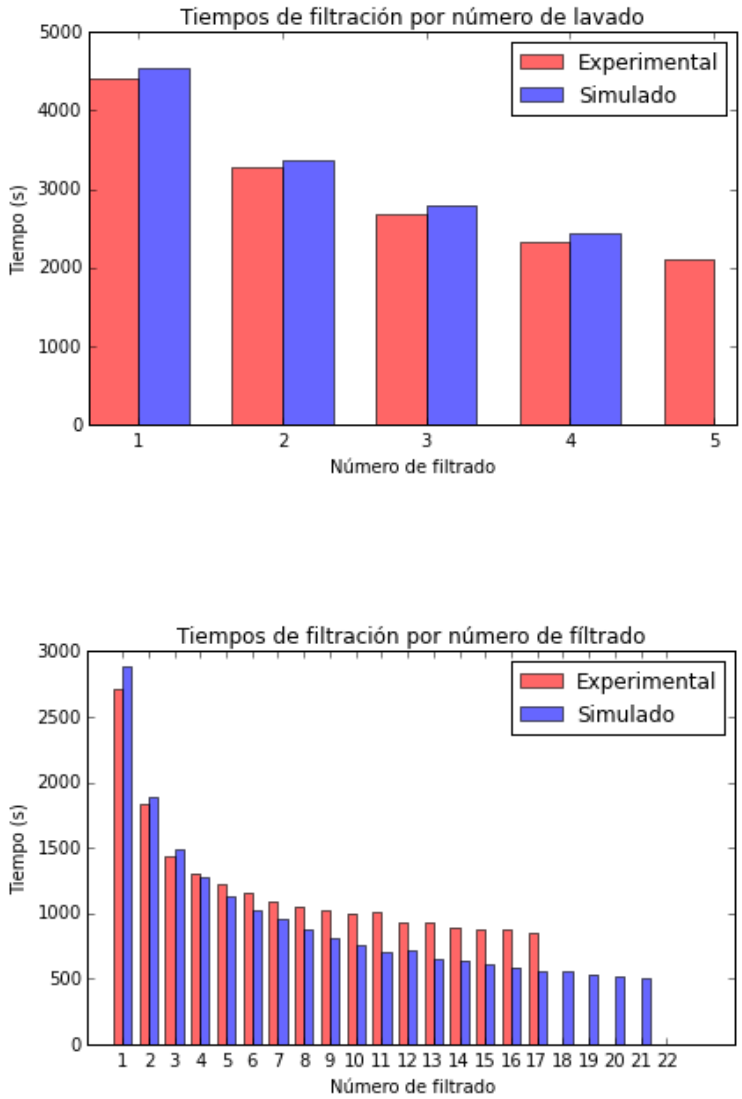

Figura 14. Tiempos de filtrado para un flujo de J=12 L h^{-1} m^{-2} (arriba) y J = 16 L h^{-1} m^{-2} (debajo)

10 Conclusiones

- La modelización computacional junto a los métodos numéricos, nos permite simular procesos complejos en ingeniería.
- La simulación nos permite ahorrar tiempo y dinero, además de ganar una mejor comprensión del sistema estudiado.
- El modelo ensayado se ha calibrado frente a datos empíricos obtenidos mediante experimentos de flujo escalonado y mediante ensayos de filtración a corto plazo (< 6 h)
- Se ha obtenido un conjunto de valores para parámetros del modelo que representa adecuadamente tanto los datos de flujo en escalón como los primeros ciclos de los ensayos de filtración.
- De forma preliminar, el modelo parece representar adecuadamente los resultados experimentales cuando el flujo de permeado es inferior al flujo crítico, pero se desvía progresivamente con el tiempo de duración del ensayo cuando el flujo es superior al valor crítico.
- El modelo ensayado posee valores por defecto que necesitan más investigación.
- Se recomienda, asimismo, ensayar otras estrategias de calibración dinámica, como podría ser la optimización de la función objetivo:
    - En un rango más amplio de tiempo y de números de ciclos de filtración,
    - Usando experimentos con diferentes flujos de permeado.
- La simulación no incluye los modelos biológicos (que nos permiten obtener X_{TS}, en lugar de ponerlo como una constante). Estos modelos nos permitirán ganar mayor comprensión y mejorar la simulación.
- Existe un rango de mejora en el uso de los métodos numéricos empleados para mejorar la rapidez del código, su legibilidad y consistencia.
- El código empleado para la simulación puede reutilizarse, modificarse y mejorarse para futuras investigaciones.

11 Conclusions

- Computational modeling in conjunction with numerical methods allows us to simulate complex models in engineering.
- Simulations let us save time and money, and to gain a insight in our process.
- The model simulated has been calibrated with empirical data through flux-step trials and filtration experiments in the short-term (< 6 h).
- We have obtained a group of parameters that represents correctly the flux-step trials and the first's filtration steps.
- It seems that the model works better when the operative flux is below the critical flux. Even though, above the critical flux, the model represents the filtration behavior the output gives us an increasing filtration time.
- The model has some default parameter values that need more research.
- It is recommended to follow different approaches to the calibration of the model, like optimize the objective function through:
    - Applying it to a higher range of time and number of filtration steps.
    - Using experiments with different fluxes.
- This simulation does not include biological models (it will allow us to get X_{TS} instead of declaring it as a constant). This model can help us to get a better understanding of the process and get better results in the simulation.
- There is an improvement range in the use of the numerical methods. The code can be faster, more robust and more legible.
- The code used in this project can be re-used, modified, and developed for future research.

12 Bibliografía

[1] S. Judd, The MBR Book (Second Edition), Elsevier Ltd., 2011.

[2] S. Judd, «The status of industrial and municipal effluent treatment with membrane bioreactor technology (pendiente de publicación),» *Chemical Engineering Journal*, 2015.

[3] H. Fangohr, Introduction to Python for Computational Science and Engineering (a begginer's guide), Faculty of Engineering and the Environment. University of Southampton, 2015.

[4] A. Robles, M. Ruano, J. Ribes, A. Seco y J. Ferrer, «A filtration model applied to submerged anaerobic MBRs (SAnMBRs),» *Journal of Membrane Science*, nº 444, pp. 139-147, 2013.

[5] L. Vera, E. González, I. Ruigómez, J. Gómez y S. Delgado, «Influence of Gas Sparging Intermittence on Ultrafiltration Performance of Anaerobic Suspensions,» *nd. Eng. Chem. Res.*, nº 55 (16), p. 4668–4675, 2016.

[6] L. Vera, E. González, I. Ruigómez, J. Gómez y S. Delgado, «Analysis of backwashing efficiency in dead-end hollow-fibre ultrafiltration of anaerobic suspensions,» *Environ Sci*

Pollut Res Int., nº 22, pp. 16600-16609, 2015.

[7] T. V. Bugge, M. K. Jørgensen, M. L. Christensen y K. Keiding, «Modeling cake buildup under TMP-step filtration in a membrane bioreactor: Cake compressibility is significant,» *Water Research,* nº 46, pp. 4330-4338, 2012.

[8] M. K. Jørgensen, T. V. Bugge, M. L. Christensen y K. Keiding, «Modeling approach to determine cake buildup and compression in a high-shear membrane bioreactor,» *Journal of Membrane Science,* nº 409–410, pp. 335-345, 2012.

[9] L. Hughes y R. Field, «Crossflow filtration o fwashed and unwashed yeast suspensions at constant shear under nominally sub-critical conditions,» *Journal of Membrane Science,* nº 280, pp. 89-98, 2006.

[10] A. Robles, M. Ruano, J. Ribes, A. Seco y J. Ferrer, «Global sensitivity analysis of a filtration model for submerged anaerobic membrane bioreactors (AnMBR),» *Bioresource Technology,* nº 158, pp. 365-373, 2014.

13 Anexos

13.1 Anexo 1: Datos experimentales

13.1.1 Datos de los experimentos de flujo-escalón

		BRF_V = 1,25 L/min	BRF_V = 1,75 L/min	BRF_V = 2,50 L/min	BRF_V = 3,75 L/min
Flujo		FR			
l/hm2	m/s	Pa/s	Pa/s	Pa/s	Pa/s
4	1,11E-06	0,06	0,05	0,02	0,11
8	2,22E-06	0,28	0,28	0,23	0,18
12	3,33E-06	1,29	1,28	0,60	0,59
16	4,44E-06	3,53	2,87	1,58	2,20
20	5,56E-06	8,81	6,78	4,77	5,22
24	6,67E-06	38,10	26,68	21,47	16,91
28	7,78E-06	89,23	81,59	75,68	57,44

Tabla 4. Datos experimentales de flujo-escalón

13.2 Anexo 2: Código de programación

13.2.1 “flux_step.py”

```
# -*- coding: utf-8 -*-
"""
Programa que estima los 4 parametros:
kf, beta1, beta2, gamma

Se realiza con la funcion minimize (una funcion para ajuste de curvas en
sistemas lineales y no lineales).
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from lmfit import minimize, Parameters, report_fit

# Guarda los datos de calibracion en objeto tipo DataFrame (df) de pandas:
cal_df = pd.read_excel("TFGPy.xlsm", sheetname="CalibraciÃ³nPy", skiprows=1)

#Se realiza una copia del DataFrame
df1 = cal_df.copy()

# Convierte  DataFrame en arrays de numpy:
flujo = df1["l/hm2"].as_matrix()
fr1   = df1["Pa/s"].as_matrix()
fr2   = df1["Pa/s.1"].as_matrix()
fr3   = df1["Pa/s.2"].as_matrix()
fr4   = df1["Pa/s.3"].as_matrix()

# Convierte el  flujo de l h-1m-2 a ms-1:
flujo = flujo * (1 / (1000*3600))

# Se guardan los array en una lista para ahorrar lineas de codigo
brfv = [0.00694, 0.00972, 0.0139, 0.0208]
frData   = [fr1, fr2, fr3, fr4]


def fRate(parameters, flux, brfv, mlts=8.22):
    ''''' Nos proporciona FR teoricos a partir de datos de Flujo, cambiando
    el valor del brfv'''
    kf, beta1, beta2, gamma = parameters
    FR = kf * np.exp(flux * (beta1 * brfv + beta2 * mlts + gamma))
    return FR


def fit_function(params, flujo, frData, brfv):
```

```
    kf = params['kf'].value
    beta1 = params['beta1'].value
    beta2 = params['beta2'].value
    gamma = params['gamma'].value

    model1 = kf * np.exp(flujo *(beta1 * brfv[0] + beta2 * 8.22 + gamma))

    model2 = kf * np.exp(flujo *(beta1 * brfv[1] + beta2 * 8.22 + gamma))

    model3 = kf * np.exp(flujo *(beta1 * brfv[2] + beta2 * 8.22 + gamma))

    model4 = kf * np.exp(flujo *(beta1 * brfv[3] + beta2 * 8.22 + gamma))

    resid1 = frData[0] - model1
    resid2 = frData[1] - model2
    resid3 = frData[2] - model3
    resid4 = frData[3] - model4

    return np.concatenate((resid1, resid2, resid3, resid4))

params = Parameters()
params.add(   'kf', value=   5.6e-04)
params.add('beta1', value= -2.48e+08)
params.add('beta2', value=   5.1e+04)
params.add('gamma', value=  2.81e+06)

out = minimize(fit_function, params, args=(flujo, frData, brfv))
print(report_fit(out))

# Grafico:
# Listas de marcador y de parametros obtenidos con lmfit
lineStyle = ['bo', 'r^', 'gs', 'yp']
plmfit = [3.3e-02, -4.3e+06, -5.5e+04, 1.5e+06]

plt.figure(figsize=(8, 6))

# Representacion de los datos experimentales
for i in range(4):
    plt.plot(flujo, frData[i], lineStyle[i], label='%s'%(brfv[i]))

# Valores de Flujo para representar
x  = np.linspace(min(flujo), max(flujo) * 1.01, 50)
# Valores de FR calculados con los parametros estimados
y1, y2, y3, y4 = fRate(plmfit, x, brfv[0]), fRate(plmfit, x, brfv[1]), \
                 fRate(plmfit, x, brfv[2]), fRate(plmfit, x, brfv[3])
# RepresentaciÃ³n de los datos calculados
plt.plot(x, y1, 'b-', x, y2, 'r-', x, y3, 'g-', x, y4, 'y-')

plt.legend(loc='best', fontsize='medium',
           title=r'$BRF_v  (Nm^{3} s^{-1} m^{-3})$')
```

```
plt.xlabel(r'$J_{20} (m s^{-1})$', fontsize='x-large')
plt.ylabel(r'$FR (Pa s^{-1})$', fontsize='x-large')
plt.title("Ajuste del experimento flujo-escalÃ³n")
plt.ticklabel_format(style='sci', axis='x', scilimits=(0, 0))
plt.grid()
plt.margins(0.05)
plt.savefig("flujo-escalon.png", dpi=96)
plt.show()
plt.close()
```

13.2.2 “dyn_calib_4.8.py”

```
# -*- coding: utf-8 -*-
"""
Programa que estima los 4 parametros:
qMS_max, TMPa, alphaC_0 y kSF

Se realiza con las funciones odeint(integra sistemas de ec. diferenciales)
y minimize(una funcion para ajuste de curvas)
"""
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import pandas as pd
from lmfit import minimize, Parameters, fit_report
import sys


# Variables por experimento
RM = np.array([5.27e+11, 6.32e+11]) # (m-1)
MLTS = np.array([5.52, 6,75])   # (kg m-3)
J20_l = np.array([12, 16])      # flujo permeado (l h m-2)
J20 = J20_l/ (1000 * 3600)      # flujo permeado (m3 s-1 m-2)

# Se elije que datos se quiere analizar
dataToAnalyze = input("Que datos quieres analizar?\
Introduce datos1 o datos2: ")   # datos 1 y 2 para J=12 y 16 lhm-2 respect.
if dataToAnalyze == "datos1":
    RM, MLTS, J20 = RM[0], MLTS[0], J20[0]
elif dataToAnalyze == "datos2":
    RM, MLTS, J20 = RM[1], MLTS[1], J20[1]
else:
    print("No se ha introducido nombre de datos correcto")
    sys.exit(1)

# Otras variables:
XTS = MLTS                      # (= MLTS(kg/mÂ³))
TMPconsigna = 28000             # (Pa)
J20_BF = 60 / (1000*3600)       # (m3 s-1 m-2)
tLavado = 30                    # (s)
BRFv = 1.65 / (0.003*60*1000)   # (Nm3 s-1 m-3)
mu = 0.001                      # viscosidad (kg m-1 s-1)
superfMemb = 0.047              # (m2)
Q20P = J20 * superfMemb         # (m3 s-1)
Q20_BF = J20_BF * superfMemb

# Parametros por defecto o que se pueden poner como tal
qBF_max = 1                     # por defecto en Robles
alphaI = 1e+14                  # por defecto en Robles
kt = 1                          # por defecto en Robles
```

```
KS_XmC = 0.2                    # estimado en Robles(largo plazo)
qIF_max = 3.0e-
07              # estimado en Robles(inversa de SRT, 38.5 d)

# Parametros estimados en el script flux_step.py
KF    =  3.3e-02
beta1 = -4.3e+06
beta2 = -5.5e+04
gamma =  1.5e+06

FR = KF * np.exp(J20 * (beta1 * BRFv + beta2 * MLTS + gamma))
IMS = 1 / (1 + FR)


# PARTE I: Importacion y tratamiento de datos-----------------------------

fil_df = pd.read_excel("TFGPy.xlsm", sheetname="Experimentales", skiprows=7,
                       header=1)
df1 = fil_df.copy()

# Asignar columnas de datos a variables
if dataToAnalyze == "datos1":
    t,tmp = df1.t1, df1.TMP1
elif dataToAnalyze == "datos2":
    t,tmp = df1.t2, df1.TMP2

# Se tratan los datos
t   = t[~t.isnull()]      # selecciona valores no "nulos"(NaNs)
tmp = tmp[~tmp.isnull()]
tmp[tmp < 0] = 0          # aquellos valores negativos se fijan como 0

# Se convierten los objetos Pandas a arrays de NumPy
t   = t.as_matrix().astype(int)
tmp = tmp.as_matrix()

# Definir el rango de presiones hasta la primera etapa de filtracion
for i in range(1, len(tmp)-1):
    if tmp[i] == 0:
        TMPexp = tmp[:i]
        t1     = t[:i]
        break

# PARTE II: CALIBRACION DINAMICA-------------------------------------------

# Definicion de los vectores:
XmI    = np.zeros(t1.shape, np.float)
omegaI = np.zeros(t1.shape, np.float)
XmC    = np.zeros(t1.shape, np.float)
omegaC = np.zeros(t1.shape, np.float)
alphaC = np.zeros(t1.shape, np.float)
TMPsim = np.zeros(t1.shape, np.float)
```

```
# Define una funcion cuyos parametros queremos obtener, para luego poder
# simular la TMP y poder comparar con los TMP experimentales. Se
# obtendran aquellos parametros con el menor resiudal.


def fit_function(params, TMPexp, t):

    qMS_max  = params['qMS_max'].value
    TMPa     = params['TMPa'].value
    alphaC_0 = params['alphaC_0'].value
    kSF      = params['kSF'].value


    def dXmCdt(y, t):
        XmC = y[0]
        dXmCdt = Q20P * XTS - (qMS_max * IMS * BRFv *
                               (XmC / (KS_XmC + XmC)) * XmC) - qIF_max * XmC
        dXmIdt = qIF_max * XmC

        return [dXmCdt, dXmIdt]


    def dalphaCdt(alphaC, t, TMPsim):
        alphaC_TMP = alphaC_0 * (1 +(TMPsim / TMPa))
        dalphaCdt = max(kSF, kt * (alphaC_TMP - alphaC))
        return dalphaCdt


    for i in range(1, len(TMPexp)):

        tspan = [t1[i-1], t1[i]]
        y0    = [XmC[i-1], XmI[i-1]]

        sol_XmC = odeint(dXmCdt,y0, tspan, rtol=1e-3,atol=1e-3)
        sol_alphaC = odeint(dalphaCdt, alphaC[i-1], tspan,
                            args=(TMPsim[i-1],), rtol=1e-3,atol=1e-3)

        XmC[i]    = (sol_XmC[-1,0])
        XmI[i]    = (sol_XmC[-1,1])
        alphaC[i] = (sol_alphaC[-1, 0])

        TMPsim[i] = ( J20 * mu * (RM + (XmC[i] / superfMemb) * (alphaC[i]) +

                                    (XmI[i] / superfMemb) * alphaI) )

    resid1 = (TMPsim - TMPexp)

    return resid1

# Diccionario de parametros a estimar en el modelo
```

```
p = Parameters()
#           (      Name,    Value,  Vary,      Min,      Max, Expr)
p.add_many( ( 'qMS_max',     6.31,  True,        1,       10, None),
            (    'TMPa',    18900,  True,    5e+03,    5e+04, None),
            ('alphaC_0', 1.02e+13,  True,    1e+10,    1e+18, None),
            (     'kSF', 4.09e+10,  True,    4e+07,    4e+13, None) )

out = minimize(fit_function, p, args=(TMPexp, t1))
print(fit_report(out))

# Grafico del modelo y experimental

plt.plot(t1, TMPsim,'r-', label='TMPsim')
plt.plot(t1, TMPexp, 'b-', label='TMPexp')
plt.xlabel("Tiempo(s)")
plt.ylabel("TMP (Pa)")
plt.legend(loc='best')
plt.savefig("Cal_dyn %s.png"%(dataToAnalyze))
plt.show()
plt.close()
```

13.2.3 “exp_vs_sim_4.8py”

```
"""
Project: Representacion de todo el rango de datos exp. Con los parametros
estimados, representar tambien todo el rango de datos simulado.
Comparar graficos.
"""
import pandas as pd
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import sys

# Variables por experimento
RM = np.array([5.27e+11, 6.32e+11]) # (m-1)
MLTS = np.array([5.52, 6,75])   # (kg m-3)
J20_l = np.array([12, 16])      # flujo permeado (l h-1 m-2)
J20 = J20_l/ (1000 * 3600)      # flujo permeado (m3 s-1 m-2)

# Se elije que datos se quiere analizar
dataToAnalyze = input("Que datos quieres analizar?\
Introduce datos1 o datos2: ")   #datos 1 y 2 para J=12 y 16 lh-1m-2 respec.
if dataToAnalyze == "datos1":
    RM, MLTS, J20 = RM[0], MLTS[0], J20[0]
elif dataToAnalyze == "datos2":
    RM, MLTS, J20 = RM[1], MLTS[1], J20[1]
else:
    print("No se ha introducido nombre de datos correcto")
    sys.exit(1)

# Otras variables
XTS = MLTS                      # (= MLTS(kg/mÂ³))
TMPconsigna = 28000             # (Pa)
J20_BF = 60 / (1000*3600)       # (m3Â·s-1Â·m-2)
tLavado = 30                    # (s)
BRFv = 1.65 / (0.003*60*1000)   # (Nm3Â·s-1Â·m-3)
mu = 0.001                      # viscosidad
superfMemb = 0.047              #
Q20P = J20 * superfMemb
Q20_BF = J20_BF * superfMemb

# Parametros por defecto o que se pueden poner como tal:
qBF_max = 1                     # por defecto en Robles
alphaI = 1e+14                  # por defecto en Robles
kt = 1                          # por defecto en Robles
KS_XmC = 0.2                    # estimado en Robles(largo plazo)
qIF_max = 3.0e-07               # inversa del SRT de operacion

# Parametros estimados en el script flux_step.py
KF    =  3.3e-02
```

```
beta1 = -4.3e+06
beta2 = -5.5e+04
gamma =  1.5e+06

FR = KF * np.exp(J20 * (beta1 * BRFv + beta2 * MLTS + gamma))
IMS = 1 / (1 + FR)

# Parametros estimados en el script dyn_calib
qMS_max  = np.array([1.0, 1])
TMPa     = np.array([18400, 20000])
alphaC_0 = np.array([4.196e+13, 1e+13])
kSF      = np.array([9.2e+09, 2.45e+10])

if dataToAnalyze == "datos1":
    qMS_max, TMPa, alphaC_0, kSF = qMS_max[0], TMPa[0], alphaC_0[0], kSF[0]
elif dataToAnalyze == "datos2":
    qMS_max, TMPa, alphaC_0, kSF = qMS_max[1], TMPa[1], alphaC_0[1], kSF[1]


# PARTE I: Importacion y tratamiento de datos-----------------------------

fil_df = pd.read_excel("TFGPy.xlsm", sheetname="Experimentales", skiprows=7,
                       header=1)
df1 = fil_df.copy()

# Asignar columnas de datos a variables
if dataToAnalyze == "datos1":
    t,tmp = df1.t1, df1.TMP1
elif dataToAnalyze == "datos2":
    t,tmp = df1.t2, df1.TMP2

# Se tratan los datos
t   = t[~t.isnull()]      # selecciona valores no "nulos"(NaNs)
tmp = tmp[~tmp.isnull()]
tmp[tmp < 0] = 0          # aquellos valores negativos se fijan como 0

# Se convierten los objetos panda a arrays de NumPy
t   = t.as_matrix().astype(int)
tmp = tmp.as_matrix()


# PARTE II: Simulacion----------------------------------------------------

# 1. Condiciones iniciales para TMPsim, XmC, XmI,  aplha_C

XmI    = np.zeros(t.shape, np.float)
omegaI = np.zeros(t.shape, np.float)
XmC    = np.zeros(t.shape, np.float)
omegaC = np.zeros(t.shape, np.float)
alphaC = np.zeros(t.shape, np.float)
alphaC[0] = alphaC_0   # el parametro estimado representa su valor a t=0
```

```
TMPsim = np.zeros(t.shape, np.float)

# Etapa de filtracion/lavado


def dXmCdt(y, t):
    '''''ec. formacion de la torta durante filtrado'''
    XmC = y[0]
    dXmCdt = Q20P * XTS - (qMS_max * IMS * BRFv *
                                (XmC / (KS_XmC + XmC)) * XmC) - qIF_max * XmC
    dXmIdt = qIF_max * XmC

    return [dXmCdt, dXmIdt]


def dXmCdt1(y, t):
    ''''' Process 3: cake layer detachment during back-flushing '''
    XmC = y[0]
    dXmCdt = - qBF_max * Q20_BF * (XmC / (KS_XmC + XmC)) * XmC
    dXmIdt = 0
    return [dXmCdt, dXmIdt]


def dalphaCdt(alphaC, t, TMPsim):
    alphaC_TMP = alphaC_0 * (1 + (TMPsim / TMPa))
    dalphaCdt  = max(kSF, kt * (alphaC_TMP - alphaC))
    return dalphaCdt

i = 0
while i < (len(t)-1):

    while TMPsim[i] < TMPconsigna:   # Filtracion

        tspan = np.linspace(t[i], t[i+1], 20)
        y0    = [XmC[i], XmI[i]]

        sol_XmC    = odeint(dXmCdt, y0, tspan, rtol=1e-3, atol=1e-3)
        sol_alphaC = odeint(dalphaCdt, alphaC[i], tspan,
                            args=(TMPsim[i],), rtol=1e-3, atol=1e-3)

        XmC[i+1]    = (sol_XmC[-1, 0])
        XmI[i+1]    = (sol_XmC[-1, 1])
        alphaC[i+1] = (sol_alphaC[-1, 0])
        TMPsim[i+1] = (J20 * mu * (RM + (XmC[i+1] / superfMemb) *
                         (alphaC[i+1]) + (XmI[i+1] / superfMemb) * alphaI))
        i += 1
        if i == (len(t)-1):
            break

    else:
```

```
        TMPsim[i] = 0
        i = i - 1
        tInicLavado = t[i]
        while np.around(t[i] - tInicLavado) < 30: #tlavado inferior a 30 s

            tspan = np.linspace(t[i], t[i+1], 20)
            y0    = [0, XmI[i]]

            sol_XmC1   = odeint(dXmCdt1, y0, tspan, rtol=1e-3, atol=1e-3)
            sol_alphaC = odeint(dalphaCdt, alphaC[i], tspan,
                                args=(TMPsim[i],), rtol=1e-3, atol=1e-3)

            XmC[i+1]    = sol_XmC1[-1, 0]
            XmI[i+1]    = sol_XmC1[-1, 1]
            alphaC[i+1] = sol_alphaC[-1, 0]

            TMPsim[i+1] = 0

            i += 1
            if i == (len(t)-1):
                break

# Graficos Filtracion: dos subplots, se comparten ejes X e Y.
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, sharey=True,
                               figsize=(8,6))
ax1.plot(t, TMPsim, 'b-')               # se crea subplot 1
ax1.set_title("FiltraciÃ³n-SimulaciÃ³n")
ax2.plot(t, tmp, 'r-')                  # se crea subplot 2
ax2.set_title("FiltraciÃ³n-Experimental")

#Se crean titulos Ãºnicos para cada Eje
fig.text(0.5, 0.04, 'Tiempo (s)', ha='center', va='center')
fig.text(0.04, 0.5, 'TMP (Pa)', ha='center', va='center',
         rotation='vertical')

plt.savefig("filtracion %s.png"%(dataToAnalyze), dpi=96,
            bbox_inches='tight')

plt.show()
plt.close()

# Graficos Ensuciamiento: dos subplots, se comparte eje X.
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(8,6))

ax1.plot(t, (XmC/superfMemb), 'y--')
ax1.set_title("Ensuciamiento Reversible")
ax1.yaxis.get_major_formatter().set_powerlimits((0, 1)) # formato cient. eje

ax2.plot(t, (XmI/superfMemb), 'g--')
ax2.set_title("Ensuciamiento Irreversible")
ax2.yaxis.get_major_formatter().set_powerlimits((0, 1))
```

```
fig.text(0.5, 0.04,'Tiempo (s)', ha='center', va='center')
fig.text(0.04, 0.5, r'$\omega (kg m^{-2})$', ha='center', va='center',
         rotation='vertical', fontsize='x-large')

plt.savefig("ensuciamiento %s.png"%(dataToAnalyze), dpi=96,
            bbox_inches='tight')
plt.show()
plt.close()

# Funcion que calcula los tiempos de filtracion
def tFiltr(t, tmp, tLavado):
    '''''calcula los tiempos de filtracion'''
    tmpEqualZero   = t[np.abs(tmp[1:] - tmp[:-1]) > 5000]
    tiempoFilt     = np.zeros(len(tmpEqualZero))
    tiempoFilt[0]  = tmpEqualZero[0]
    tiempoFilt[1:] = tmpEqualZero[1:] - tmpEqualZero[:-1] - tLavado
    return tiempoFilt

# Tiempos experimentales y simulados de filtracion
tFiltExp = tFiltr(t, tmp, tLavado)
tFiltSim = tFiltr(t, TMPsim, tLavado)

# Graficos
fig, ax = plt.subplots()
barWidth = 0.35
opacity = 0.6
plt.bar(np.arange(1, len(tFiltExp) + 1), tFiltExp, barWidth, alpha=opacity,
        color='red', label='Experimental')
plt.bar(np.arange(1, len(tFiltSim) + 1) + barWidth, tFiltSim, barWidth,
        alpha=opacity, color='blue', label='Simulado')
plt.legend()
plt.title("Tiempos de filtraciÃ³n por nÃºmero de fÃltrado")
plt.xlabel("NÃºmero de filtrado")
plt.xticks(np.arange(1, len(tFiltSim) + 2) + barWidth,
           (np.arange(1, len(tFiltSim) + 2)))
plt.ylabel("Tiempo (s)")
plt.tight_layout()
plt.show()
plt.close()
```