Implementación del esquema de discretización temporal *BDF4* en *OpenFOAM*.

1. Esquemas de discretización temporal en OpenFOAM.

En OpenFOAM, los esquemas de discretización temporal juegan un rol fundamental en la solución de ecuaciones diferenciales en derivadas parciales que rigen el comportamiento de flujos transitorios. El marco estándar de OpenFOAM ofrece principalmente esquemas implícitos de primer y segundo orden, tales como Euler y backward. Estos métodos se caracterizan por su robustez y facilidad de implementación, pero presentan limitaciones en cuanto a precisión temporal cuando se requiere resolver fenómenos con alta sensibilidad al paso de tiempo.

En este trabajo se aborda la incorporación de un esquema de cuarto orden basado en la fórmula de diferenciación hacia atrás (BDF4, por sus siglas en inglés *Backward Differentiation Formula*). Este esquema forma parte de una familia de métodos implícitos utilizados para la integración numérica de ecuaciones diferenciales ordinarias. Su principio consiste en aproximar la derivada de una función en un instante dado utilizando valores previamente calculados en pasos de tiempo anteriores, lo que permite mejorar la precisión de la solución. En particular, el BDF4, al ser de orden superior respecto a los esquemas actualmente implementados en OpenFOAM, posibilita alcanzar una mayor exactitud temporal sin requerir una reducción del tamaño del paso de tiempo. Esto resulta especialmente beneficioso en simulaciones donde se busca un compromiso entre precisión y eficiencia computacional.

La implementación del esquema BDF4 se realizó extendiendo la funcionalidad de la clase `backwardDdtScheme`. A continuación, se describen en detalle tanto los aspectos técnicos de la implementación como los casos de validación empleados.

2. Derivación de los coeficientes de BDF4 para pasos de tiempo no uniformes.

La derivada temporal primera, de acuerdo al esquema BDF4, se representa de la forma:

$$\frac{d\phi}{dt}_{BDF4} \equiv c\phi + c^{0}\phi^{0} + c^{00}\phi^{00} + c^{000}\phi^{000} + c^{0000}\phi^{0000} \tag{1}$$

La notación con superíndices '0', '00', '000', etc., representa valores en pasos de tiempo anteriores, siendo la cantidad de ceros indicativa del número de pasos previos al instante actual. El tamaño de los pasos de tiempo se definen como:

$$\Delta t = t - t^{0}$$
$$\Delta t^{0} = t^{0} - t^{00}$$
$$\Delta t^{00} = t^{00} - t^{000}$$
$$\Delta t^{000} = t^{000} - t^{0000}$$

También se definen los siguientes coeficientes para simplificar la notación:

$$\alpha = \Delta t$$
$$\beta = \Delta t + \Delta t^{0}$$
$$\gamma = \Delta t + \Delta t^{0} + \Delta t^{00}$$
$$\delta = \Delta t + \Delta t^{0} + \Delta t^{00} + \Delta t^{000}$$

Para la obtención de los coeficientes "c" se desarrolló la ecuación (1) en series de Taylor alrededor del paso de tiempo que se quiere calcular:

$$c\phi + c^{0}\left[\phi - \alpha\frac{d\phi}{dt} + \frac{\alpha^{2}}{2}\frac{d^{2}\phi}{dt^{2}} - \frac{\alpha^{3}}{6}\frac{d^{3}\phi}{dt^{3}} + \frac{\alpha^{4}}{24}\frac{d^{4}\phi}{dt^{4}} + \ \cdots\right] +$$
$$+ c^{00}\left[\phi - \beta\frac{d\phi}{dt} + \frac{\beta^{2}}{2}\frac{d^{2}\phi}{dt^{2}} - \frac{\beta^{3}}{6}\frac{d^{3}\phi}{dt^{3}} + \frac{\beta^{4}}{24}\frac{d^{4}\phi}{dt^{4}} + \ \cdots\right] +$$
$$+ c^{000}\left[\phi - \gamma\frac{d\phi}{dt} + \frac{\gamma^{2}}{2}\frac{d^{2}\phi}{dt^{2}} - \frac{\gamma^{3}}{6}\frac{d^{3}\phi}{dt^{3}} + \frac{\gamma^{4}}{24}\frac{d^{4}\phi}{dt^{4}} + \ \cdots\right] +$$
$$+ c^{0000}\left[\phi - \delta\frac{d\phi}{dt} + \frac{\delta^{2}}{2}\frac{d^{2}\phi}{dt^{2}} - \frac{\delta^{3}}{6}\frac{d^{3}\phi}{dt^{3}} + \frac{\delta^{4}}{24}\frac{d^{4}\phi}{dt^{4}} + \ \cdots\right]$$

Para que el esquema sea de orden 4, los coeficientes "c" deben ser tales que

$$\begin{bmatrix}1 & 1 & 1 & 1 & 1\\ 0 & \alpha & \beta & \gamma & \delta\\ 0 & \alpha^2 & \beta^2 & \gamma^2 & \delta^2\\ 0 & \alpha^3 & \beta^3 & \gamma^3 & \delta^3\\ 0 & \alpha^4 & \beta^4 & \gamma^4 & \delta^4\end{bmatrix}\begin{bmatrix}c\\ c^0\\ c^{00}\\ c^{000}\\ c^{0000}\end{bmatrix}=\begin{bmatrix}0\\ -1\\ 0\\ 0\\ 0\end{bmatrix}$$

De ésta manera se obtiene que $\frac{d\phi}{dt}_{BDF4} = \frac{d\phi}{dt} + O\left(\Delta t^4\right)$

Para resolver el sistema de ecuaciones se utilizó el método de eliminación de Gauss-Jordan, y se obtuvo que:

$$c^{0000} = \frac{\alpha\beta\gamma}{\delta(\delta-\alpha)(\delta-\beta)(\delta-\gamma)}$$

$$c^{000} = \frac{\alpha\beta}{(\alpha-\gamma)(\gamma-\beta)}\left(\frac{1}{(\delta-\gamma)} + \frac{1}{\gamma}\right)$$

$$c^{00} = \frac{\alpha}{(\beta-\alpha)}\left(\frac{1}{\beta} + \frac{1}{(\gamma-\beta)}\right) + \frac{\alpha\gamma}{(\beta-\alpha)(\delta-\gamma)}\left(\frac{1}{(\gamma-\beta)} - \frac{1}{(\delta-\beta)}\right)$$

$$c^{0} = \frac{\beta\gamma}{(\delta-\gamma)}\left(\frac{1}{(\gamma-\beta)(\gamma-\alpha)} + \frac{1}{(\beta-\alpha)(\delta-\beta)} - \frac{1}{(\delta-\alpha)(\delta-\beta)} - \frac{1}{(\beta-\alpha)(\gamma-\beta)}\right) + \frac{\beta}{(\gamma-\beta)}\left(\frac{1}{(\gamma-\alpha)} - \frac{1}{(\beta-\alpha)}\right) - \frac{1}{\alpha} - \frac{1}{(\beta-\alpha)}$$

$$c = -\left(c^{0} + c^{00} + c^{000} + c^{0000}\right)$$

3. Implementación de la clase `Foam::fv::bdf4DdtScheme`

La clase `bdf4DdtScheme<Type>` implementa el esquema *Fourth-order Backward Euler* (BDF4) para la derivada temporal primera en OpenFOAM, usando el valor actual y los cuatro valores previos del campo. El código fuente de la implementación de dicha clase se realizó en la siguiente ubicación:

```
$WM_PROJECT_USER_DIR/src/finiteVolume/finiteVolume/ddtSchemes/

bdf4DdtScheme/
├── bdf4DdtScheme.H
├── bdf4DdtScheme.C
```

Siguiendo el formato de la documentación disponible en la página oficial de OpenFOAM (https://www.openfoam.com/), se realiza a continuación, una descripción de la clase listando sus distintos componentes.

3.1. Herencia

Como se muestra en la Figura 1, la clase `bdf4DdtScheme<Type>` hereda de `ddtScheme<Type>`. Esta última es una clase abstracta que sirve como base para todos los esquemas de discretización temporal implementados en OpenFOAM. Al tratarse de una clase abstracta, declara de forma virtual todos los métodos que deben ser implementados por cualquier esquema temporal derivado.

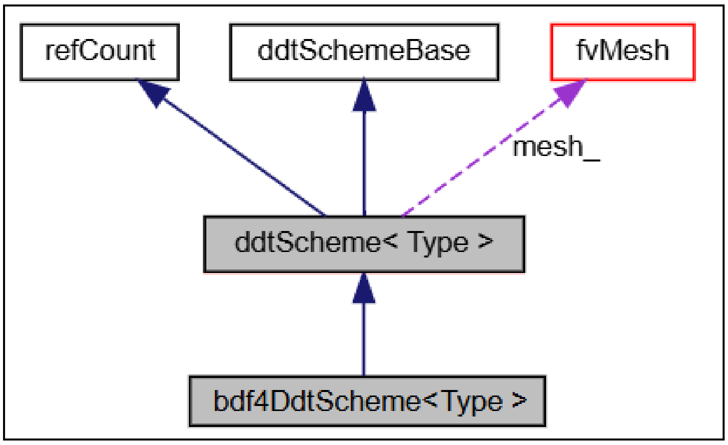

Figura 1 - Diagrama de herencia de la clase `bdf4DdtScheme<Type>`.

3.2. Tipos públicos

`fluxFieldType` es un alias de tipo que representa un campo de flujo superficial que se utiliza para funciones miembro públicas de corrección de flujo.

`typedef ddtScheme< Type >::fluxFieldType`	**`fluxFieldType`**

3.3. Funciones miembro privadas

En la columna izquierda se indica si el método fue implementado o no.

`scalar`	**`deltaT_`** `() const` Retorna el paso de tiempo actual
`scalar`	**`deltaT0_`** `() const` Retorna el paso de tiempo previo
`scalar`	**`deltaT0_`** `(const GeoField&) const` Retorna el paso de tiempo previo o GREAT si el paso de tiempo previo no está disponible, en cuyo caso se utiliza el esquema temporal Euler.

`scalar` Implementado parcialmente. Actualmente devuelve deltaT0_() o GREAT.	**`deltaT00_`** `(const GeoField&) const` Retorna el paso de tiempo previo-previo o GREAT si el paso de tiempo previo-previo no está disponible, en cuyo caso se utiliza el esquema temporal backward.
`scalar` Implementado parcialmente. Actualmente devuelve deltaT0_() o GREAT.	**`deltaT000_`** `(const GeoField&) const` Retorna el paso de tiempo previo-previo-previo o GREAT si el paso de tiempo previo-previo-previo no está disponible, en cuyo caso se utiliza el esquema temporal BDF3.
`scalar`	**`alpha_`** `() const` Retorna el valor del coeficiente α
`scalar`	**`beta_`** `(const GeoField&) const` Retorna el valor del coeficiente β
`scalar`	**`gamma_`** `(const GeoField&) const` Retorna el valor del coeficiente γ
`scalar`	**`delta_`** `(const GeoField&) const` Retorna el valor del coeficiente δ

`scalar`	**`c0000_`** `(const scalar alphaC,`
	`const scalar beta,`
	`const scalar gamma,`
	`const scalar delta) const`
	Retorna el valor del coeficiente $c0000$

`scalar`	**`c000_`** `(const scalar alphaC,`
	`const scalar beta,`
	`const scalar gamma,`
	`const scalar delta) const`
	Retorna el valor del coeficiente $c000$

`scalar`	**`c00_`** `(const scalar alphaC,`
	`const scalar beta,`
	`const scalar gamma,`
	`const scalar delta) const`
	Retorna el valor del coeficiente $c00$

`scalar`	**`c0_`** `(const scalar alphaC,`
	`const scalar beta,`
	`const scalar gamma,`

	`const scalar delta) const` Retorna el valor del coeficiente $c0$
	`bdf4DdtScheme` `(const bdf4DdtScheme&) = delete` Elimina el constructor por copia
`void`	**`operator=`** `(const bdf4DdtScheme&) = delete` Elimina el operador de asignación por copia

Las funciones miembro privadas `deltaT0_ (const GeoField&)`, `deltaT00_ (const GeoField&)` y `deltaT000_ (const GeoField&)` devuelven los tamaños de los pasos de tiempo previos correspondientes. Si no hay suficiente historial temporal disponible (es decir, no se cuenta con la cantidad necesaria de pasos de tiempo anteriores), la función correspondiente retorna el valor **`GREAT`**. En OpenFOAM, este valor es una constante escalar que representa un número muy grande.

Los coeficientes "c" del esquema temporal están definidos de modo que, al hacer tender a infinito los pasos de tiempo que no existen (simulado mediante `GREAT`), se obtienen automáticamente los coeficientes del método de diferencias hacia atrás (BDF) del orden más alto posible con la información disponible. Por ejemplo, si solo se tienen tres pasos de tiempo previos, la función `deltaT000_` retornará `GREAT`, lo que hará que el coeficiente c^{0000} sea igual a cero, y los coeficientes restantes corresponderán al esquema BDF3.

3.4. Funciones miembro públicas

En la columna izquierda se indica si el método fue implementado o no.

	`TypeName ("bfd4")` Información del Runtime type.
	`bfd4DdtScheme` `(const fvMesh &mesh)` Constructor desde la malla.
	`bfd4DdtScheme` `(const fvMesh &mesh, Istream &is)` Constructor desde la malla y Istream.
`const fvMesh &`	`mesh () const` Retorna una referencia a la malla.
`tmp< GeometricField< Type, fvPatchField, volMesh > >` No se ha implementado el soporte para mallas móviles.	**`fvcDdt`** `(const dimensioned< Type > &)`

tmp< **GeometricField**< Type, **fvPatchField**, **volMesh** > > No se ha implementado el soporte para mallas móviles.	**fvcDdt** (const **GeometricField**< Type, **fvPatchField**, **volMesh** > &)
tmp< **GeometricField**< Type, **fvPatchField**, **volMesh** > > No se ha implementado el soporte para mallas móviles.	**fvcDdt** (const **dimensionedScalar** &, const **GeometricField**< Type, **fvPatchField**, **volMesh** > &)
tmp< **GeometricField**< Type, **fvPatchField**, **volMesh** > > No se ha implementado el soporte para mallas móviles.	**fvcDdt** (const **volScalarField** &, const **GeometricField**< Type, **fvPatchField**, **volMesh** > &)
tmp< **GeometricField**< Type, **fvPatchField**, **volMesh** > > No se ha implementado el soporte para mallas móviles.	**fvcDdt** (const **volScalarField** &**alpha**, const **volScalarField** &**rho**, const **GeometricField**< Type, **fvPatchField**, **volMesh** > &**psi**)
tmp< **fvMatrix**< Type > > No se ha implementado el soporte para mallas móviles.	**fvmDdt** (const **GeometricField**< Type, **fvPatchField**, **volMesh** > &)
tmp< **fvMatrix**< Type > > No se ha implementado el soporte para mallas móviles.	**fvmDdt** (const **dimensionedScalar** &, const **GeometricField**< Type, **fvPatchField**, **volMesh** > &)
tmp< **fvMatrix**< Type > > No se ha implementado el soporte para mallas móviles.	**fvmDdt** (const **volScalarField** &, const **GeometricField**< Type, **fvPatchField**, **volMesh** > &)
tmp< **fvMatrix**< Type > > No se ha implementado el soporte para mallas móviles.	**fvmDdt** (const **volScalarField** &**alpha**, const **volScalarField** &**rho**, const **GeometricField**< Type, **fvPatchField**, **volMesh** > &**psi**)
tmp< **fluxFieldType** > No implementado.	**fvcDdtUfCorr** (const **GeometricField**< Type, **fvPatchField**, **volMesh** > &**U**, const **GeometricField**< Type, **fvsPatchField**, **surfaceMesh** > &**Uf**)

tmp< fluxFieldType > No implementado.	**fvcDdtPhiCorr** (const GeometricField< Type, fvPatchField, volMesh > &U, const fluxFieldType &phi)
tmp< fluxFieldType > No implementado.	**fvcDdtUfCorr** (const volScalarField &rho, const GeometricField< Type, fvPatchField, volMesh > &U, const GeometricField< Type, fvsPatchField, surfaceMesh > &Uf)
tmp< fluxFieldType > No implementado.	**fvcDdtPhiCorr** (const volScalarField &rho, const GeometricField< Type, fvPatchField, volMesh > &U, const fluxFieldType &phi)
tmp< surfaceScalarField > No implementado.	**meshPhi** (const GeometricField< Type, fvPatchField, volMesh > &)
tmp< surfaceScalarField > No implementado.	**fvcDdtUfCorr** (const GeometricField< scalar, fvPatchField, volMesh > &U, const GeometricField< scalar, fvsPatchField, surfaceMesh > &Uf)
tmp< surfaceScalarField > No implementado.	**fvcDdtPhiCorr** (const volScalarField &U, const surfaceScalarField &phi)
tmp< surfaceScalarField > No implementado.	**fvcDdtUfCorr** (const volScalarField &rho, const volScalarField &U, const surfaceScalarField &Uf)
tmp< surfaceScalarField > No implementado.	**fvcDdtPhiCorr** (const volScalarField &rho, const volScalarField &U, const surfaceScalarField &phi)

La función miembro pública fvcDdt se encarga de calcular la derivada temporal explícita utilizando BDF4. Sus múltiples sobrecargas permiten que reciba como entrada diferentes combinaciones de campos, retornando un GeometricField< Type, fvPatchField, volMesh >.

La función miembro pública fvmDdt se encarga de ensamblar la matriz de coeficientes que representa la contribución la derivada temporal primera. Sus diferentes sobrecargas permiten aceptar como entrada diferentes combinaciones de campos. Retorna un fvMatrix <Type>.

Dado que todos los métodos mencionados anteriormente requieren los valores de los campos en tiempos previos, se verificó que estos puedan obtenerse encadenando, tantas veces como sea necesario, el método oldTime(). Los campos correspondientes a tiempos anteriores (cuando están disponibles) se almacenan en los directorios de tiempo con los sufijos _0, _0_0 y _0_0_0. Esto permite que puedan ser reutilizados en caso de reanudar una simulación desde el último paso de tiempo calculado.

Los métodos `fvcDdtPhiCorr` & `fvcDdtUfCorr` calculan de manera explícita correcciones para el flujo superficial. Retornan un `fluxFieldType`, el cuál es un alias de tipo que representa un campo de flujo superficial. Estos métodos no fueron implementados en este trabajo.

Para manejar los casos no implementados (como mallas móviles y métodos de corrección de flujo) se incorporó, en los bloques correspondientes, un control de error en tiempo de ejecución utilizando la macro `FatalErrorInFunction`, junto con un mensaje que indica la causa específica.

3.5. Registro de la clase bdf4DdtScheme<Type> en el Runtime type selection (RTS).

En el archivo `bdf4DdtScheme.C` se agregó la siguiente línea de código:

```
makeFvDdtScheme(bdf4DdtScheme);
```

Dicha macro se encarga de registrar el nuevo esquema de tipo `ddtSchemes` en el RTS. Esto permite que pueda ser seleccionado desde el archivo de configuración `fvSchemes.` En la línea 151 del archivo `bdf4DdtScheme.H` se indicó que dicha selección se debe realizar a través del nombre 'bdf4':

```
TypeName("bdf4");
```

A través del archivo Make/files, se indica que la librería se compila en la ruta `$(FOAM_USER_LIBBIN)/libmyBdf4DdtScheme`

Para que el *solver* encuentre dicha librería, se debe agregar la siguiente línea en el *controlDict* del caso a correr:

```
libs (libmyBdf4DdtScheme);
```

4. Test de orden de convergencia en pasos de tiempo del esquema implementado

Una vez que el esquema BDF4 fue implementado en la clase `bfd4DdtSchemes<Type>,` se procedió a verificar que efectivamente se comporte como un esquema de cuarto orden. Para ello, se programó un *solver* llamado `cosODEFoam` que resuelve la siguiente ecuación diferencial ordinaria lineal de primer orden:

$$\frac{d\phi}{dt} = -\lambda sin(\lambda t)$$

La cuál tiene como solución:

$\phi(t) = cos(\lambda t) + \phi(0)$

Dicho *solver* se encuentra ubicado en:

```
$WM_PROJECT_USER_DIR/applications/solvers/

cosODEFoam/
├── cosODEFoam.C
├── createFields.H
```

En el archivo `createFields.H` se creó una variable `dimensionedScalar` llamada `lambda`, con unidades de inverso de tiempo $[T^{-1}]$, cuyo valor se obtiene del diccionario `transportProperties`. Asimismo, se lee el campo escalar volumétrico `volScalarField phi`, el cuál es de lectura obligatoria y se guarda automáticamente en los resultados del caso.

4.1. Configuración de los casos

Los casos a simular se encuentran en el directorio `$WM_PROJECT_USER_DIR/run/cosODE/`. Se configuraron un total de 8 casos, cada uno con un paso de tiempo uniforme distinto. El caso con mayor paso de tiempo corresponde a $\Delta T = 0.001\ [s]$, y los restantes se generaron disminuyendo consecutivamente el paso de tiempo a la mitad. Cada caso se ubicó en un subdirectorio cuyo nombre contiene el valor de su paso de tiempo, permitiendo una rápida identificación.

Para la condición inicial, se estableció un valor de $\phi(0) = 1$ en el archivo `0/phi`. A partir de la solución exacta, correspondiente a la condición inicial $\phi(0) = 1$, se calcularon los valores del campo ϕ en los tres tiempos anteriores $-\Delta t, -2\Delta t$ & $-3\Delta t$. Estos valores se almacenaron en los archivos `0/phi_0`; `0/phi_0_0` & `0/phi_0_0_0`, respectivamente.

Además, se creó el archivo `time` ubicado en `0/uniform/`, en el cual se especifican los valores de `deltaT`, `deltaT0` y el índice de tiempo asociado al tiempo '0' (`index`). Estos tres valores son requeridos por los métodos `deltaT_`, `deltaT0_`, `deltaT00_` y `deltaT000_`.

Los valores de `deltaT` y `deltaT0` se obtienen mediante los métodos `deltaTValue()` y `deltaT0Value()` de la clase `Time`, respectivamente. Por su parte, el índice de tiempo (index) se recupera utilizando el método `timeIndex()` de la misma clase. Este índice permite determinar si existe una cantidad suficiente de pasos de tiempo previos almacenados.

En todos los casos, se configuró manualmente un valor de index = 4 en el archivo correspondiente al tiempo 0, indicando la disponibilidad de cuatro valores anteriores del campo a calcular. Gracias a ello, la simulación puede comenzar directamente utilizando el esquema temporal BDF4, sin necesidad de recurrir a métodos de orden inferior en los primeros pasos.

La configuración de los 8 casos fue la siguiente:

- Geometría de 1 celda, con condiciones de borde `empty` en todas sus caras.
- Tiempo simulado 0.01 [s]
- $\lambda = 1000$. Este valor permite tener al menos un ciclo del coseno en el tiempo total simulado, logrando un mayor rango de gradientes a calcular.
- `writePrecision = 16;` en el *controlDict* para realizar comparaciones con precisión de máquina.
- Condición inicial $\phi(0) = 1$

4.2. Cálculo del error y orden del esquema

Para calcular el error, simplemente se obtuvo el valor de la solución de cada caso en el último paso de tiempo, y se tomó la diferencia absoluta respecto a la solución exacta en el mismo instante, es decir, $e = |Ts - Te|$. El orden (P) del esquema se calculó entre casos consecutivos como sigue:

$$P = log_2\left(\frac{e_{\Delta t}}{e_{\Delta t/2}}\right)$$

Dichos cálculos están implementados en el *script* de *Python* `postProcess.py,` ubicado en el directorio de los casos `cosODE/.` En la tabla 1, se muestran los resultados obtenidos.

Nombre del caso	Δt [s]	Te	Ts	Error	Orden
case_bdf4_dt _0000078125	7,8125E-06	-0,83907153	-0,83907153	1,3755E-09	4,00510581
case_bdf4_dt _000015625	0,000015625	-0,83907153	-0,83907155	2,2086E-08	4,0097916
case_bdf4_dt _00003125	0,00003125	-0,83907153	-0,83907188	3,5579E-07	4,01698006
case_bdf4_dt _0000625	0,0000625	-0,83907153	-0,83907729	5,76E-06	4,02375222
case_bdf4_dt _000125	0,000125	-0,83907153	-0,83916522	9,3689E-05	4,00785746

case_bdf4_dt_00025	0,00025	-0,83907153	-0,84057875	0,00150722	3,85563446
case_bdf4_dt_0005	0,0005	-0,83907153	-0,86089065	0,02181912	2,49173134
case_bdf4_dt_001	0,001	-0,83907153	-0,96179371	0,12272218	-

Tabla 1 - Resumen de los resultados obtenidos en los test de convergencia del esquema BDF4 implementado.

Como se puede observar, se obtuvo que efectivamente el orden del esquema implementado es de cuarto orden.

5. Trabajo a futuro

A continuación, se listan las tareas necesarias para completar la implementación del esquema BDF4:

1. Implementar los métodos de corrección de flujo `fvcDdtPhiCorr` & `fvcDdtUfCorr`.
2. Implementar soporte para mallas móviles.
3. Implementar la posibilidad de obtener los tamaños de paso de tiempo `deltaT00` y `deltaT000`, necesarios para que el esquema funcione en paso de tiempo variable. Dicha funcionalidad se deberá incorporar a las clases `Time` y `TimeState`.