schemes/divergenceExample

Allrun スクリプト	概説
#!/bin/bash	bash スクリプトであることの宣言。
cd "${0%/*}" || exit　　# Run from this directory . ${WM_PROJECT_DIR:?}/bin/tools/RunFunctions　　# Tutorial run functions	OpenFOAM の実行等を手助けする関数群を読み込む
#--	
# Save the line plot unset savePlots	savePlots という変数を削除する
if notTest "$@" then 　　savePlots=true fi	`notTest`は，RunFunctions 内で定義されている関数である。コマンドライン引数中の`-test`オプション有無を確認する。
restore0Dir	0.orig ディレクトリをコピーして 0 ディレクトリを作成する。
runApplication blockMesh	runApplication は RunFunctions 内で定義されている関数である。blockMesh を実行し，ログを log.blockMesh ファイルに保存する。
while read -r scheme do	`system/schemesToTest` ファイルを読み込む。読み込んだ 1 行が `scheme` という変数に格納する。do から done の間の作業を読み込むファイルの行がなくなるまで繰り返す。`-r`オプションは，バックスラッシュ（/）記号をそのまま使用する（エスケープ文字としない）ため。
echo "Updating fvSchemes to use $scheme" 　　sed "s/DIVSCHEME/$scheme/g" system/fvSchemes.template > system/fvSchemes	system/fvSchemes.template ファイルの”DIVSCHEME”を読み込んだ `scheme` の内容に書き換える。それを system/fvSchemes として書き出す。

`    # Create a sanitised name for the scheme - remove 'special' characters` `    schemeTag=$(sed -e 's# #_#g#' -e 's#[.()]##g' <<< "$scheme")`	ファイル名に使えない文字を取り除く（サニタイズ）。Stream editer の sed を使って，変数`scheme`内の文字列の空白を"_"に変更し，ピリオドとカッコを削除する。
`    runApplication -s "${schemeTag}" scalarTransportFoam`	`runApplication` 関数をオプションとともに使って `scalarTransportFoam` ソルバを実行する。`-s` オプションに続けて文字列を指定すると，保存される log ファイルの最後にその文字列が追加される。（実行後には，サニタイズされたスキーム名のついたログファイルが残される。）

`    if [ "$savePlots" = true ]`
`    then`
`        # Save the line plot`
`        mv -f postProcessing/sample1/100/line1_T.xy line1_T_"${schemeTag}".xy`
`    fi`
	このスクリプトを `-test` オプションをつけずに実行した場合に実行される。
	
	ポスト処理用にサンプリングしたファイル `postProcessing/sample1/100/line1_T.xy`を，Allrun と同じディレクトリに名前を変えて移動（mv: move）させる。変更後の名前には，サニタイズしたスキーム名を追加する。（実行後には，スキーム名が付加された `.xy` ファイルが保存されている。）

`done < system/schemesToTest`	繰り返し作業（do　done）に使用するファイルを指定する。

schemes/nonOrthogonalChannel

Allrun スクリプト	概説

`#!/bin/sh`
`cd "${0%/*}" || exit　　# Run from this directory`
`. ${WM_PROJECT_DIR:?}/bin/tools/RunFunctions　　# Tutorial run functions`
`#------------------------------------------------------------------------`
	bash スクリプトであることの宣言。
	
	OpenFOAM の実行等を手助けする関数群を読み込む

-
 	
# settings 	
 	
    # operand setups
    setups="
    0
    10
    20
    30
    40
    50
    60
    70
    80
    85
    "
 	
    # flag to enable computations
    run=true
 	
    # flag to enable computations in parallel mode
    parallel=false
 	
    # flag to enable to use a common mesh
    common_mesh=false
 	
    # flag to enable to use a common dynamic code
    common_dynamic_code=false
	setups という変数に，処理したい設定の名前のリストを入力する。
	
	実行制御用の変数に，設定値を与える。
	計算実行可 run
	並列計算可 false
	共通メッシュ使用 common_mesh
	共通実行時コード common_dynamic_code

```
#------------------------------------------------------------------------
-

#########################################
# Create the given setup
# Arguments:
#    $1 = Path to create the setup
# Outputs:
#    Writes info to stdout
#########################################
dry_run_setup() {

    [ $# -eq 0 ] && { echo "Usage error: $0"; exit 1; }

    setup="$1"
    dirSetup="setups/$setup"
    dirSetupOrig="setups.orig/$setup"
    dirOrig="$dirSetupOrig/0.orig"
    dirConstant="$dirSetupOrig/constant"
    dirSystem="$dirSetupOrig/system"

    printf "\n# Create the setup: %s\n" "$setup"

    if [ ! -d "$dirSetup" ]
    then
        mkdir -p "$dirSetup"
```

実行テスト（ドライ ラン）用セットアップ関数 dry_run_setup

この関数は，実行用セットアップ関数からも呼び出される。

ディレクトリ名の設定
関数呼び出し時の引数を setup に格納する
dirSetup = setups / $setup

$dirSetup ディレクトリが存在しないときには作成する.（親ディレクトリから（-p））

```
        cp -aRfL "setups.orig/common/." "$dirSetup"
        cp -afL "$dirSetupOrig"/All* "$dirSetup" 2>/dev/null || :
        [ -d "$dirOrig" ] && cp -aRfL "$dirOrig/." "$dirSetup/0.orig"
        [ -d "$dirConstant" ] && cp -aRfL "$dirConstant/." "$dirSetup/constant"
        [ -d "$dirSystem" ] && cp -aRfL "$dirSystem/." "$dirSetup/system"
    else
        printf "\n      # Directory %s already exists\n" "$dirSetup"
        printf "      # Skipping the creation of a new setup\n"
    fi
}



##########################################
# Run the given setup
# Arguments:
#    $1 = Path to the setup to run
# Outputs:
#    Writes info to stdout
##########################################
run_setup() {

    [ $# -eq 0 ] && { echo "Usage error: $0"; exit 1; }

    setup="$1"
    dirSetup="setups/$setup"
    dirResult="results/$setup"

    dry_run_setup "$setup"
    [ -d results ] || mkdir -p results
```

計算実行用セットアップ関数 run_setup

使用時には，実行したい設定を引数として与える。そのため，$1 には 10 のような名前が入っている。

ディレクトリ名の設定
関数呼び出し時の引数を setup に格納する
設定格納
dirSetup = setups / $setup
結果保存用ディレクトリ $dirResult = results / $setup

$setup を引数として与え，dry_run_setup 関数を呼び出す。設定名 $setup 用に，各種ディレクトリを準備する。

`    printf "\n# Run the setup: %s\n\n" "$setup"`	

`    if [ ! -d "$dirResult" ]`
`    then`
`        cp -Rf "$dirSetup" "$dirResult"`
	結果保存ディレクトリの準備

`        if [ "$common_mesh" = true ]`
`        then`
`            if [ -d results/mesh ]`
`            then`
`                printf "## Copy the common mesh to the setup: %s\n\n" "$setup"`
`                cp -Rf results/mesh/polyMesh "$dirResult"/constant/.`
`            fi`
`        fi`
	共通メッシュを使用する場合には，メッシュ情報をコピーする。

`        if [ "$common_dynamic_code" = true ]`
`        then`
`            if [ -d results/dynamicCode ]`
`            then`
`                printf "## Copy the common dynamic code to the setup: %s\n\n" "$setup"`
`                cp -Rf results/dynamicCode "$dirResult"/.`
`            fi`
`        fi`
	共通実行時コードを使用する場合には，情報をコピーする。
	
	runApplication は RunFunctions 内で定義されている関数である。
	blockMesh を実行し，ログを log.blockMesh ファイルに保存する。

`        if [ "$parallel" = true ]` `        then`	

(cd "$dirResult" && ./Allrun-parallel) else (cd "$dirResult" && ./Allrun) fi if ["$common_mesh" = true] then if [! -d results/mesh] then printf "\n## Store the mesh of %s as the common mesh\n\n" "$setup" mkdir -p results/mesh cp -Rf "$dirResult"/constant/polyMesh results/mesh/. fi fi if ["$common_dynamic_code" = true] then if [! -d results/dynamicCode] then printf "\n## Store the dynamic code of %s as the common dynamic code\n\n" "$setup" cp -Rf "$dirResult"/dynamicCode results/. fi fi else	並列計算実行時には Allrun-parallel スクリプトを, 非並列計算実行時には Allrun スクリプトを 実行する。

```
        printf "      # Directory %s already exists\n" "$dirResult"
        printf "      # Skipping the computation of the given setup\n"
    fi
}


#------------------------------------------------------------------------

for setup in $setups
do
    dirSetupOrig="setups.orig/$setup"

    if [ ! -d "$dirSetupOrig" ]
    then
        echo "Setup directory: $dirSetupOrig" \
             "could not be found - skipping execution" 1>&2
        continue
    fi

    if [ "$run" = true ]
    then
        run_setup "$setup"
    else
        dry_run_setup "$setup"
    fi
done


if notTest "$@" && [ "$run" = true ]
```

スクリプトのメイン部分

$setups 変数の中の項目 1 つずつを取出して，その値を $setup に入れる。do から done の間を繰り返す。

設定格納場所 setups.orig ディレクトリの下に，設定名 $setup のディレクトリが存在するかを確認する。なければ，メッセージを表示して，次の設定に進む。

計算実行のフラグに応じて，
計算を実行する，または，
計算実行用ディレクトリを準備する。

then ./plot fi #----------	計算実行時には，グラフ作成スクリプト plot を実行する。