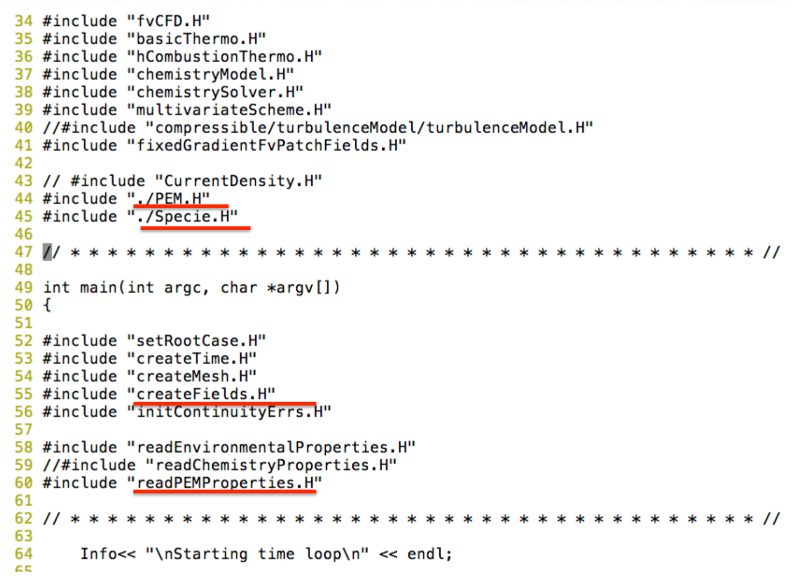
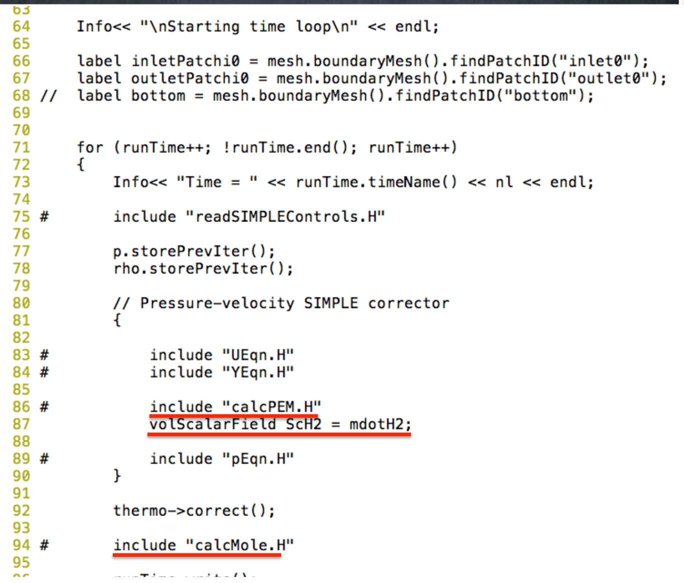
ipemSimpleFoam.C (1)



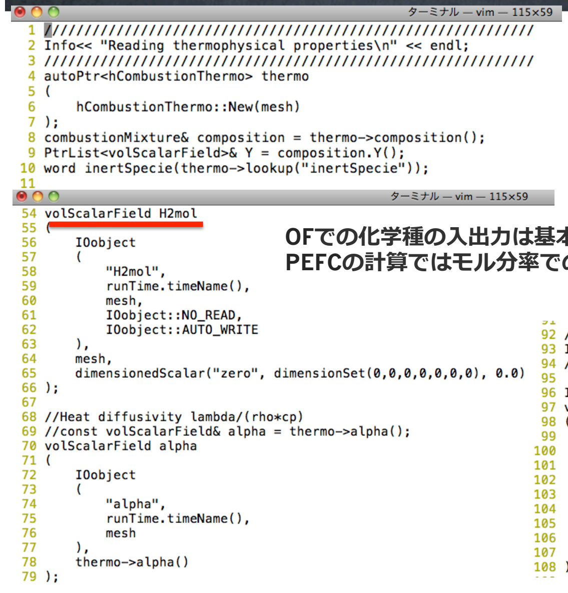
```
#include "fvCFD.H"
#include "basicThermo.H"
#include "hCombustionThermo.H"
#include "chemistryModel.H"
#include "chemistrySolver.H"
#include "multivariateScheme.H"
//#include "compressible/turbulenceModel/turbulenceModel.H"
#include "fixedGradientFvPatchFields.H"

// #include "CurrentDensity.H"
#include "./PEM.H"
#include "./Specie.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{

#include "setRootCase.H"
#include "createTime.H"
#include "createMesh.H"
#include "createFields.H"
#include "initContinuityErrs.H"

#include "readEnvironmentalProperties.H"
//#include "readChemistryProperties.H"
#include "readPEMProperties.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;
```

ipemSimpleFoam.C (2)



```
    Info<< "\nStarting time loop\n" << endl;

    label inletPatchi0 = mesh.boundaryMesh().findPatchID("inlet0");
    label outletPatchi0 = mesh.boundaryMesh().findPatchID("outlet0");
//  label bottom = mesh.boundaryMesh().findPatchID("bottom");


    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

#       include "readSIMPLEControls.H"

        p.storePrevIter();
        rho.storePrevIter();

        // Pressure-velocity SIMPLE corrector
        {

#           include "UEqn.H"
#           include "YEqn.H"

#           include "calcPEM.H"
            volScalarField ScH2 = mdotH2;

#           include "pEqn.H"
        }

        thermo->correct();

#       include "calcMole.H"

```

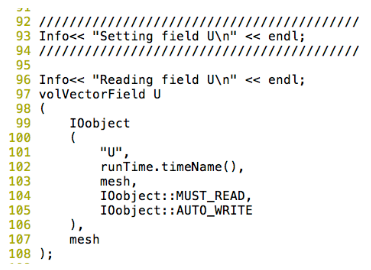
creatFields.H



```
////////////////////////////////////////////////////////////////////
Info<< "Reading thermophysical properties\n" << endl;
////////////////////////////////////////////////////////////////////
autoPtr<hCombustionThermo> thermo
(
    hCombustionThermo::New(mesh)
);
combustionMixture& composition = thermo->composition();
PtrList<volScalarField>& Y = composition.Y();
word inertSpecie(thermo->lookup("inertSpecie"));

```

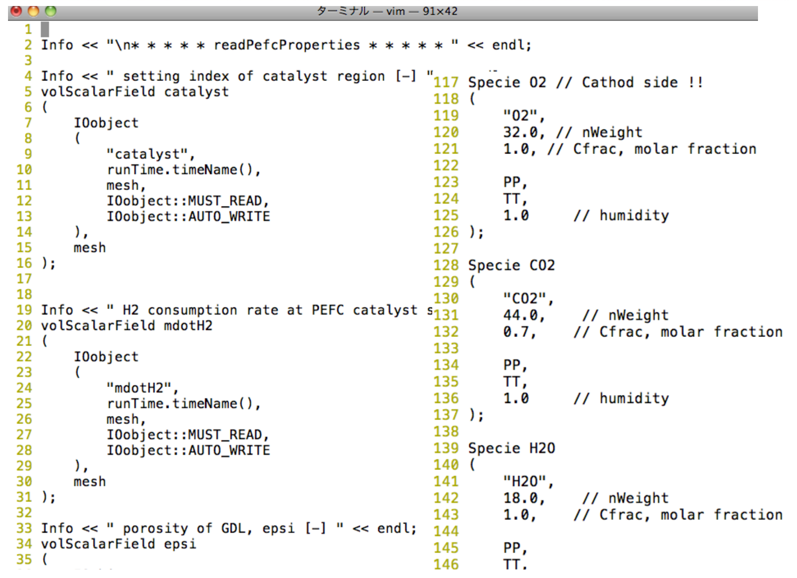
```
volScalarField H2mol
(
    IOobject
    (
        "H2mol",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionedScalar("zero", dimensionSet(0,0,0,0,0,0,0), 0.0)
);

//Heat diffusivity lambda/(rho*cp)
//const volScalarField& alpha = thermo->alpha();
volScalarField alpha
(
    IOobject
    (
        "alpha",
        runTime.timeName(),
        mesh
    ),
    thermo->alpha()
);
```

OFでの化学種の入出力は基本，質量分率。
PEFCの計算ではモル分率での値が欲しいので，追加。



```
///////////////////////////////////////////////
Info<< "Setting field U\n" << endl;
///////////////////////////////////////////////

Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

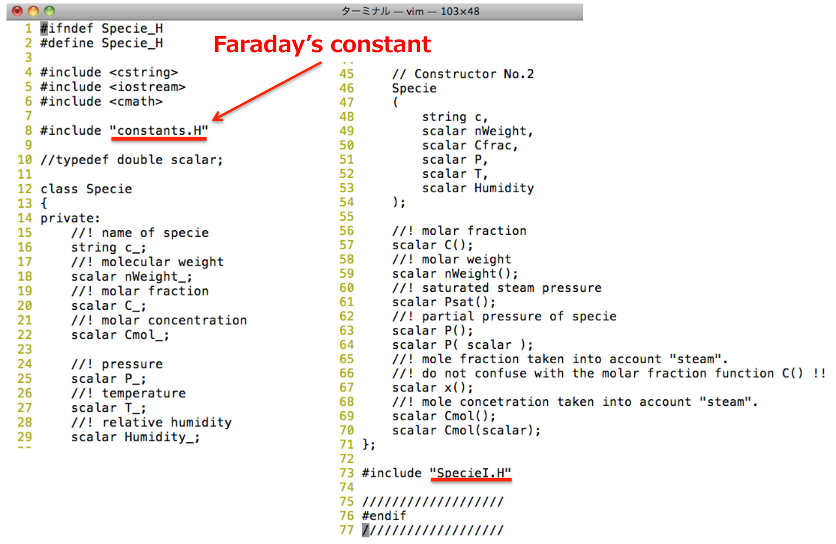
readPEFCProperties.H



```

Info << "\n* * * * * readPefcProperties * * * * * " << endl;

Info << " setting index of catalyst region [-] "
volScalarField catalyst
(
    IOobject
    (
        "catalyst",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info << " H2 consumption rate at PEFC catalyst s
volScalarField mdotH2
(
    IOobject
    (
        "mdotH2",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info << " porosity of GDL, epsi [-] " << endl;
volScalarField epsi
(
```

```
Specie O2 // Cathod side !!
(
    "O2",
    32.0, // nWeight
    1.0, // Cfrac, molar fraction

    PP,
    TT,
    1.0     // humidity
);

Specie CO2
(
    "CO2",
    44.0,    // nWeight
    0.7,    // Cfrac, molar fraction

    PP,
    TT,
    1.0     // humidity
);

Specie H2O
(
    "H2O",
    18.0,    // nWeight
    1.0,    // Cfrac, molar fraction

    PP,
    TT.
```

Specie.H



```
#ifndef Specie_H
#define Specie_H

#include <cstring>
#include <iostream>
#include <cmath>

#include "constants.H"

//typedef double scalar;

class Specie
{
private:
    //! name of specie
    string c_;
    //! molecular weight
    scalar nWeight_;
    //! molar fraction
    scalar C_;
    //! molar concentration
    scalar Cmol_;

    //! pressure
    scalar P_;
    //! temperature
    scalar T_;
    //! relative humidity
    scalar Humidity_;
```

```
    // Constructor No.2
    Specie
    (
        string c,
        scalar nWeight,
        scalar Cfrac,
        scalar P,
        scalar T,
        scalar Humidity
    );

    //! molar fraction
    scalar C();
    //! molar weight
    scalar nWeight();
    //! saturated steam pressure
    scalar Psat();
    //! partial pressure of specie
    scalar P();
    scalar P( scalar );
    //! mole fraction taken into account "steam".
    //! do not confuse with the molar fraction function C() !!
    scalar x();
    //! mole concetration taken into account "steam".
    scalar Cmol();
    scalar Cmol(scalar);
};

#include "SpecieI.H"

///////////////////
#endif
///////////////////
```

PEFC性能解析モデルの電気-化学反応モデル

電流密度の関数

$$V = E - \eta_{act} - \eta_{con} - \eta_{ohm}$$

- E :Electromotive force
  - Nernst Eq.
- η_{act} : Activation overvoltage
  - Bulter-Volmer Eq.
- η_{con} :Concentration overvoltage
  - Limiting current density
- η_{ohm} :Resistance overvoltage
  - Springer’s Eq.

未知数

$$i = 2Ff^{a} D_{H_2} \frac{C_{H_2}{}^{g} - C_{H_2}{}^{e}}{l}$$ *

- $C_{H_2}{}^{g}$: H_2 conc. on the surface between GDL and separator channel
- $C_{H_2}{}^{e}$: H_2 conc. on the surface between GDL and catalyst layer

* Inoue G. et. al., J. Power Sources 2006:139(5)

PEFC MODEL: Electromotive force

□ Electromotive force is shown by the following Nernst eq. …

$$E = E^0 + \frac{RT}{nF} \ln \left[\frac{P_{\mathrm{H_2}}{}^{a} \cdot \left(P_{\mathrm{O_2}}{}^{c}\right)^{0.5}}{P_{H2O}{}^{C}} \right]$$

- E_0 : standard electromotive force
- F : the Faraday’s constant
- R : the gas constant
- $P_{\mathrm{H_2}}{}^{a}$: anode hydrogen partial pressure
- $P_{\mathrm{O_2}}{}^{c}$: cathode oxygen partial pressure

PEFC MODEL: Anode activation overvoltage

□ Anode activation overvoltage is calculated by the following Tafel eq. …

$$\eta_{act} = \frac{RT}{\alpha_2{}^{c} F} \ln \frac{i}{A_e i_0{}^{+}}$$

- i : current density
- F : the Faraday’s constant
- A_e : effective surface area per unit projection area
- $i_0{}^{+}$: oxygen exchange current density
- $\alpha_2{}^{c}$: transfer coefficient *

* Parthansarathy A., J. Electrochem. Soc.,1992

PEFC MODEL: Concentration overvoltage

□ Anode and cathode concentration overvoltage is calculated by the limiting current density …

$$\eta_{con} = -\frac{RT}{\alpha^a 2F}\ln\left(1-\frac{i}{i_{L(\mathrm{H}_2)}}\right) - \frac{RT}{\alpha_1^{\ c} 2F}\ln\left(1-\frac{i}{i_{L(\mathrm{O}_2)}}\right)$$

- i : current density
- F : the Faraday's constant
- α^a : transfer coefficient of anode concentration overvoltage (correction parameter)
- $\alpha_1^{\ c}$: transfer coefficient of cathode concentration overvoltage (correction parameter)
- $i_{L(H_2)}$: anode limiting current density
- $i_{L(O_2)}$: cathode limiting current density

PEFC MODEL: Resistance overvoltage

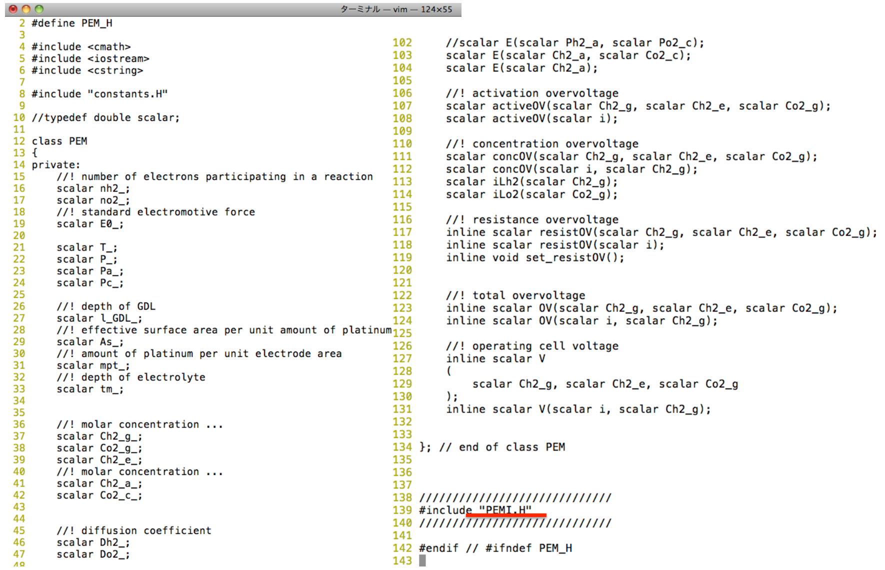
□ Resistance overvoltage is calculated using Springer's eq. of ion conductivity …

$$\eta_{ohm} = \frac{t^m}{\sigma_e^{\ m}} i$$

- i : current density
- t^m : thickness of membrane
- $\sigma_e^{\ m}$: ionic conductivity of electrolyte membrane

*:Springer,J.Electrochem.Soc.,138,1991

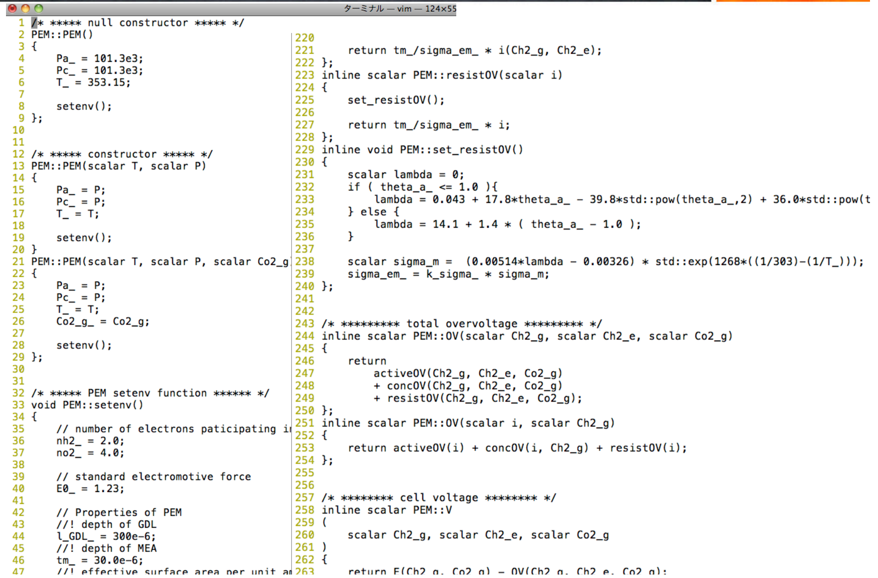
PEM.H


ターミナル — vim — 124×55

```
#define PEM_H

#include <cmath>
#include <iostream>
#include <cstring>

#include "constants.H"

//typedef double scalar;

class PEM
{
private:
    //! number of electrons participating in a reaction
    scalar nh2_;
    scalar no2_;
    //! standard electromotive force
    scalar E0_;

    scalar T_;
    scalar P_;
    scalar Pa_;
    scalar Pc_;

    //! depth of GDL
    scalar l_GDL_;
    //! effective surface area per unit amount of platinum
    scalar As_;
    //! amount of platinum per unit electrode area
    scalar mpt_;
    //! depth of electrolyte
    scalar tm_;


    //! molar concentration ...
    scalar Ch2_g_;
    scalar Co2_g_;
    scalar Ch2_e_;
    //! molar concentration ...
    scalar Ch2_a_;
    scalar Co2_c_;


    //! diffusion coefficient
    scalar Dh2_;
    scalar Do2_;
```

```
    //scalar E(scalar Ph2_a, scalar Po2_c);
    scalar E(scalar Ch2_a, scalar Co2_c);
    scalar E(scalar Ch2_a);

    //! activation overvoltage
    scalar activeOV(scalar Ch2_g, scalar Ch2_e, scalar Co2_g);
    scalar activeOV(scalar i);

    //! concentration overvoltage
    scalar concOV(scalar Ch2_g, scalar Ch2_e, scalar Co2_g);
    scalar concOV(scalar i, scalar Ch2_g);
    scalar iLh2(scalar Ch2_g);
    scalar iLo2(scalar Co2_g);

    //! resistance overvoltage
    inline scalar resistOV(scalar Ch2_g, scalar Ch2_e, scalar Co2_g);
    inline scalar resistOV(scalar i);
    inline void set_resistOV();


    //! total overvoltage
    inline scalar OV(scalar Ch2_g, scalar Ch2_e, scalar Co2_g);
    inline scalar OV(scalar i, scalar Ch2_g);

    //! operating cell voltage
    inline scalar V
    (
        scalar Ch2_g, scalar Ch2_e, scalar Co2_g
    );
    inline scalar V(scalar i, scalar Ch2_g);


}; // end of class PEM



//////////////////////////////
#include "PEMI.H"
//////////////////////////////

#endif // #ifndef PEM_H
```

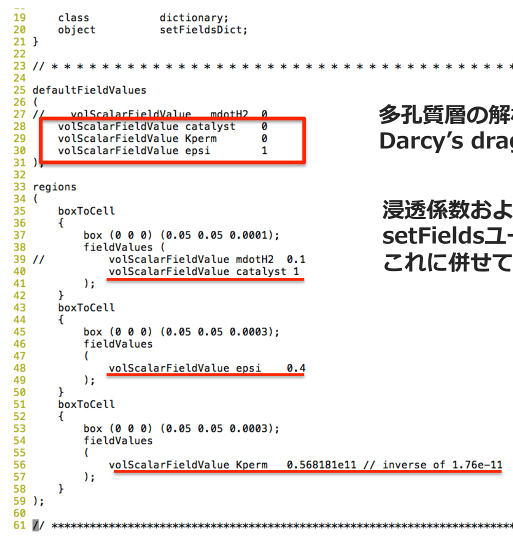
PEMI.H


ターミナル — vim — 124×55

```
/* ***** null constructor ***** */
PEM::PEM()
{
    Pa_ = 101.3e3;
    Pc_ = 101.3e3;
    T_ = 353.15;

    setenv();
};


/* ***** constructor ***** */
PEM::PEM(scalar T, scalar P)
{
    Pa_ = P;
    Pc_ = P;
    T_ = T;

    setenv();
}
PEM::PEM(scalar T, scalar P, scalar Co2_g
{
    Pa_ = P;
    Pc_ = P;
    T_ = T;
    Co2_g_ = Co2_g;

    setenv();
};


/* ***** PEM setenv function ****** */
void PEM::setenv()
{
    // number of electrons paticipating i
    nh2_ = 2.0;
    no2_ = 4.0;

    // standard electromotive force
    E0_ = 1.23;

    // Properties of PEM
    //! depth of GDL
    l_GDL_ = 300e-6;
    //! depth of MEA
    tm_ = 30.0e-6;
    //! effective surface area per unit a
```

```

    return tm_/sigma_em_ * i(Ch2_g, Ch2_e);
};
inline scalar PEM::resistOV(scalar i)
{
    set_resistOV();

    return tm_/sigma_em_ * i;
};
inline void PEM::set_resistOV()
{
    scalar lambda = 0;
    if ( theta_a_ <= 1.0 ){
        lambda = 0.043 + 17.8*theta_a_ - 39.8*std::pow(theta_a_,2) + 36.0*std::pow(t
    } else {
        lambda = 14.1 + 1.4 * ( theta_a_ - 1.0 );
    }

    scalar sigma_m =  (0.00514*lambda - 0.00326) * std::exp(1268*((1/303)-(1/T_)));
    sigma_em_ = k_sigma_ * sigma_m;
};


/* ********* total overvoltage ********* */
inline scalar PEM::OV(scalar Ch2_g, scalar Ch2_e, scalar Co2_g)
{
    return
        activeOV(Ch2_g, Ch2_e, Co2_g)
        + concOV(Ch2_g, Ch2_e, Co2_g)
        + resistOV(Ch2_g, Ch2_e, Co2_g);
};
inline scalar PEM::OV(scalar i, scalar Ch2_g)
{
    return activeOV(i) + concOV(i, Ch2_g) + resistOV(i);
};


/* ******** cell voltage ******** */
inline scalar PEM::V
(
    scalar Ch2_g, scalar Ch2_e, scalar Co2_g
)
{
    return E(Ch2_g, Co2_g) - OV(Ch2_g, Ch2_e, Co2_g);
```

setFieldsDict

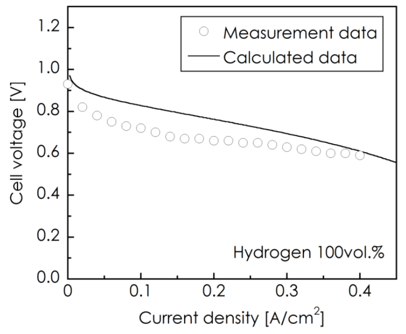


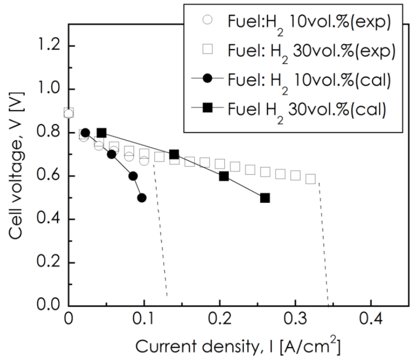
```
    class           dictionary;
    object          setFieldsDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
//     volScalarFieldValue   mdotH2  0
    volScalarFieldValue catalyst    0
    volScalarFieldValue Kperm       0
    volScalarFieldValue epsi        1
);

regions
(
    boxToCell
    {
        box (0 0 0) (0.05 0.05 0.0001);
        fieldValues (
//          volScalarFieldValue mdotH2  0.1
            volScalarFieldValue catalyst 1
        );
    }
    boxToCell
    {
        box (0 0 0) (0.05 0.05 0.0003);
        fieldValues
        (
            volScalarFieldValue epsi    0.4
        );
    }
    boxToCell
    {
        box (0 0 0) (0.05 0.05 0.0003);
        fieldValues
        (
            volScalarFieldValue Kperm   0.568181e11 // inverse of 1.76e-11
        );
    }
);

// ************************************************************************* //
```

多孔質層の解析に際しては，
Darcy's drag forceにて流動抵抗を算出。

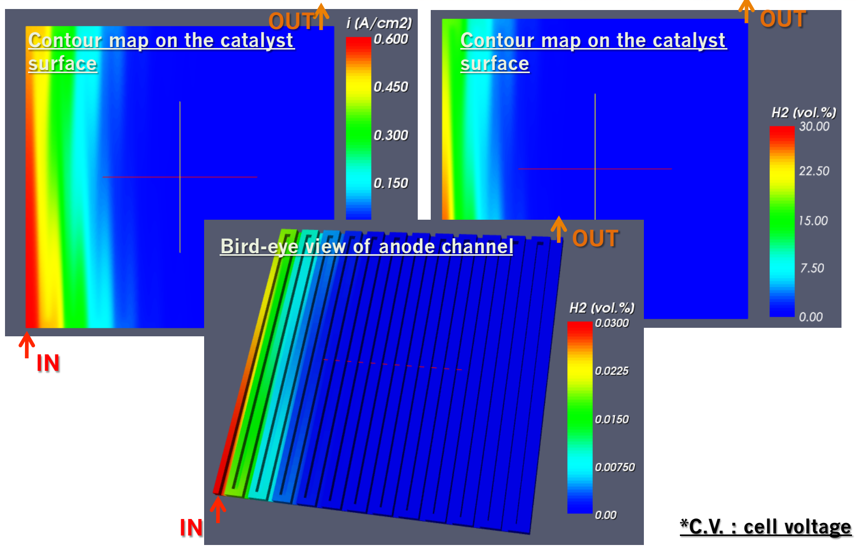
浸透係数および空隙率を
setFieldsユーティリティで与える。
これに併せて触媒層のインデックスを付す。

実測データとの比較（ターフェルプロット）



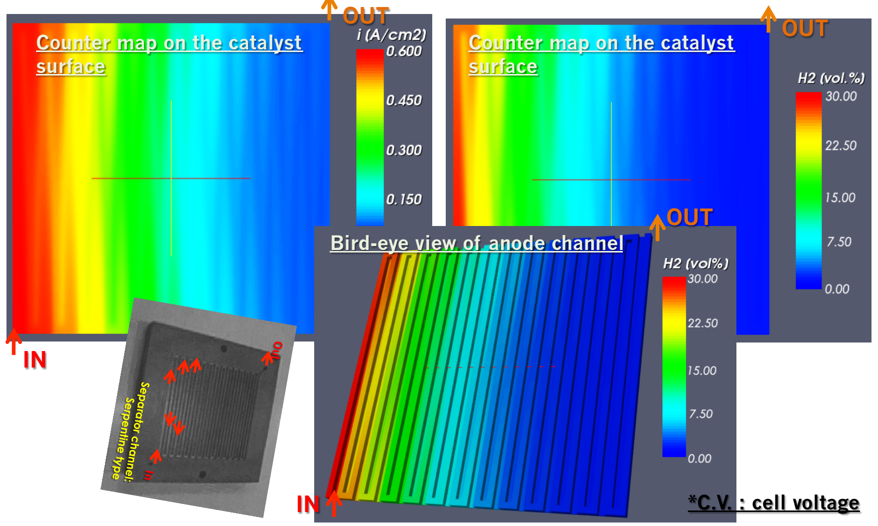


SIMULATION RESULTS: H_2 30vol.%, 100sccm, C.V. 0.5V



*C.V. : cell voltage

PEFC性能解析結果: H_2 30vol.%, 200sccm, C.V. 0.5V



*C.V. : cell voltage

※ 燃料電池の最適な運転条件を探索。また，流路形状の最適化に適用。

まとめ

- OpenFOAMに自作の関数（クラス）を導入することは比較的簡単。少々，強引な実装をしても動く！
- Ver. Upの度にライブラリに手が入ったり，新しいモデルやツールが追加され，ついて行くのが大変。ユーザーミーティングなどで情報収集することが重要。
  （知らない間にOFが広まっていて驚いた。）

- 生体モデルへのsnappyHexMeshの適用で，どの程度の品質のメッシュができるかを確認する。
- 以前のバージョンに比べて収束性が上がっている？
- 計算格子の原点の設定はOF側ではできないのか？