

ipemSimpleFoam.C (1)

```
34 #include "fvCFD.H"
35 #include "basicThermo.H"
36 #include "hCombustionThermo.H"
37 #include "chemistryModel.H"
38 #include "chemistrySolver.H"
39 #include "multivariateScheme.H"
40 //#include "compressible/turbulenceModel/turbulenceModel.H"
41 #include "fixedGradientFvPatchFields.H"
42
43 // #include "CurrentDensity.H"
44 #include "PEM.H"
45 #include "Specie.H"
46
47 // *****
48
49 int main(int argc, char *argv[])
50 {
51
52 #include "setRootCase.H"
53 #include "createTime.H"
54 #include "createMesh.H"
55 #include "createFields.H"
56 #include "initContinuityErrs.H"
57
58 #include "readEnvironmentalProperties.H"
59 //#include "readChemistryProperties.H"
60 #include "readPEMProperties.H"
61
62 // *****
63
64     Info<< "\nStarting time loop\n" << endl;
65
```

ipemSimpleFoam.C (2)

```
63     Info<< "\nStarting time loop\n" << endl;
64
65     label inletPatchi0 = mesh.boundaryMesh().findPatchID("inlet0");
66     label outletPatchi0 = mesh.boundaryMesh().findPatchID("outlet0");
67     label bottom = mesh.boundaryMesh().findPatchID("bottom");
68
69     for (runTime++; !runTime.end(); runTime++)
70     {
71         Info<< "Time = " << runTime.timeName() << nl << endl;
72
73         #include "readSIMPLEControls.H"
74
75         p.storePrevIter();
76         rho.storePrevIter();
77
78         // Pressure-velocity SIMPLE corrector
79         {
80             #include "UEqn.H"
81             #include "YEqn.H"
82
83             #include "calcPEM.H"
84             volScalarField Sch2 = mdotH2;
85
86             #include "pEqn.H"
87         }
88
89         thermo->correct();
90
91         #include "calcMole.H"
92     }
93
94     ~~~~~
95
```

creatFields.H

```
1 Info<< "Reading thermophysical properties\n" << endl;
2
3 autoPtr<hCombustionThermo> thermo
4 (
5     hCombustionThermo::New(mesh)
6 );
7
8 combustionMixture& composition = thermo->composition();
9 PtrList<volScalarField> & Y = composition.Y();
10 word inertSpecie(thermo->lookup("inertSpecie"));
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
```

ターミナル - vim - 115x59

ターミナル - vim - 115x59

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。

```
54 volScalarField H2mol
55 (
56     IObject
57     (
58         "H2mol",
59         runTime.timeName(),
60         mesh,
61         IObject::NO_READ,
62         IObject::AUTO_WRITE
63     ),
64     mesh,
65     dimensionedScalar("zero", dimensionSet(0,0,0,0,0,0), 0.0)
66 );
67
68 //Heat diffusivity lambda/(rho*cp)
69 //const volScalarField& alpha = thermo->alpha();
70 volScalarField alpha
71 (
72     IObject
73     (
74         "alpha",
75         runTime.timeName(),
76         mesh
77     ),
78     thermo->alpha()
79 );
```

readPEFCProperties.H

```
1 Info << "\n***** readPefcProperties *****" << endl;
2
3
4 Info << " setting index of catalyst region [-]" 117 Specie O2 // Cathod side !!
5 volScalarField catalyst
6 (
7     IObject
8     (
9         "catalyst",
10        runTime.timeName(),
11        mesh,
12        IObject::MUST_READ,
13        IObject::AUTO_WRITE
14    ),
15    mesh
16 );
17
18 Specie C02
19 (
20     "C02",
21     44.0, // nWeight
22     0.7, // Cfrac, molar fraction
23 );
24
25 Info << " H2 consumption rate at PEFC catalyst 131
26 volScalarField mdotH2
27 (
28     IObject
29     (
30         "mdotH2",
31         runTime.timeName(),
32         mesh,
33         IObject::MUST_READ,
34         IObject::AUTO_WRITE
35     ),
36     mesh
37 );
38
39 Specie H2O
40 (
41     "H2O",
42     18.0, // nWeight
43     1.0, // Cfrac, molar fraction
44 );
45
46 Specie H2O
47 (
48     "H2O",
49     18.0, // nWeight
50     1.0, // Cfrac, molar fraction
51 );
52
53 Specie H2O
54 (
55     "H2O",
56     18.0, // nWeight
57     1.0, // Cfrac, molar fraction
58 );
59
60 Specie H2O
61 (
62     "H2O",
63     18.0, // nWeight
64     1.0, // Cfrac, molar fraction
65 );
66
67 Specie H2O
68 (
69     "H2O",
70     18.0, // nWeight
71     1.0, // Cfrac, molar fraction
72 );
73
74 Specie H2O
75 (
76     "H2O",
77     18.0, // nWeight
78     1.0, // Cfrac, molar fraction
79 );
```

ターミナル - vim - 91x42

Specie.H

```

1 #ifndef Specie_H
2 #define Specie_H
3
4 #include <string>
5 #include <iostream>
6 #include <cmath>
7
8 #include "constants.H"
9
10 //typedef double scalar;
11
12 class Specie
13 {
14 private:
15     //! name of specie
16     string c_;
17     //! molecular weight
18     scalar nWeight_;
19     //! molar fraction
20     scalar C_;
21     //! molar concentration
22     scalar CmOl_;
23
24     //! pressure
25     scalar P_;
26     //! temperature
27     scalar T_;
28     //! relative humidity
29     scalar Humidity_;
30
31     // Constructor No.2
32     Specie
33     (
34         string c,
35         scalar nWeight,
36         scalar Cfrac,
37         scalar P,
38         scalar T,
39         scalar Humidity
40     );
41
42     //! molar fraction
43     scalar C();
44     //! molar weight
45     scalar nWeight();
46     //! saturated steam pressure
47     scalar Psat();
48     //! partial pressure of specie
49     scalar P();
50     scalar P( scalar );
51     //! mole fraction taken into account "steam".
52     //! do not confuse with the molar fraction function C() !!
53     scalar x();
54     //! mole concentration taken into account "steam".
55     scalar CmOl();
56     scalar CmOl( scalar );
57
58     //! include "SpecieI.H"
59     #include "SpecieI.H"
60
61     ///////////////////////////////////////////////////////////////////
62     #endif
63     ///////////////////////////////////////////////////////////////////
64
65 };
66
67 #endif
68
69 ///////////////////////////////////////////////////////////////////
70
71 ///////////////////////////////////////////////////////////////////
72
73
74
75
76
77

```

Faraday's constant

PEFC性能解析モデルの電気-化学反応モデル

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

電流密度の関数

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

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

未知数

- $C_{H_2}^g$: 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. ...

$$E = E^0 + \frac{RT}{nF} \ln \left[\frac{P_{H_2}^a \cdot (P_{O_2}^c)^{0.5}}{P_{H_2O}^c} \right]$$

- E_0 : standard electromotive force
- F : the Faraday's constant
- R : the gas constant
- $P_{H_2}^a$: anode hydrogen partial pressure
- $P_{O_2}^c$: 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
- α_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 ...

$$\eta_{con} = -\frac{RT}{\alpha^a 2F} \ln\left(1 - \frac{i}{i_{L(H_2)}}\right) - \frac{RT}{\alpha_1^c 2F} \ln\left(1 - \frac{i}{i_{L(O_2)}}\right)$$

- i : current density
- F : the Faraday's constant
- α^a : transfer coefficient of anode concentration overvoltage (correction parameter)
- α_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 ...

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

- i : current density
- t^m : thickness of membrane
- σ_e^m : ionic conductivity of electrolyte membrane

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

PEM.H

```

1 #define PEM_H
2
3
4 #include <math> 102 //scalar E(scalar Ph2_a, scalar Po2_c);
5 #include <iostream> 103 scalar E(scalar Ch2_g, scalar Co2_c);
6 #include <string> 104 scalar E(scalar Ch2_g);
7
8 #include "constants.H" 105
9
10 //typedef double scalar; 106
11
12 class PEM 107
13 { 108
14 private: 109
15 //! number of electrons participating in a reaction 110
16 scalar nh2; 111 scalar concOV(scalar Ch2_g, scalar Ch2_e, scalar Co2_g);
17 scalar no2; 112 scalar concOV(scalar i, scalar Ch2_g);
18 //! standard electromotive force 113 scalar iH2(scalar Ch2_g);
19 scalar E0; 114 scalar iLO2(scalar Co2_g);
20
21 scalar T; 115 //! resistance overvoltage
22 scalar Pa; 116 inline scalar resistOV(scalar Ch2_g, scalar Ch2_e, scalar Co2_g);
23 scalar Pc; 117 inline scalar resistOV(scalar i);
24 scalar PC; 118 inline void set_resistOV();
25
26 //! depth of GDL 119
27 scalar LGDL; 120 //! total overvoltage
28 //! effective surface area per unit amount of platinum 121 inline scalar OV(scalar Ch2_g, scalar Ch2_e, scalar Co2_g);
29 scalar As; 122 inline scalar OV(scalar i, scalar Ch2_g);
30 //! amount of platinum per unit electrode area 123 //! operating cell voltage
31 scalar mpt; 124 inline scalar V
32 //! depth of electrolyte ( scalar Ch2_g, scalar Ch2_e, scalar Co2_g
33 scalar tm; 125 );
34
35 //! molar concentration ... 126
36 scalar Ch2_g; 127 // end of class PEM
37 scalar Co2_g; 128
38 scalar Ch2_e; 129
39 scalar Ch2_e; 130
40 //! molar concentration ... 131
41 scalar Ch2_g; 132
42 scalar Co2_c; 133
43
44 //! diffusion coefficient 134
45 scalar Dh2; 135
46 scalar Do2; 136
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145

```

PEM.H

```

1 /* ***** null constructor ***** */
2 PEM::PEM()
3 {
4     Pa_ = 101.3e3;
5     Pc_ = 101.3e3;
6     T_ = 353.15;
7
8     setenv();
9 };
10
11
12 /* ***** constructor ***** */
13 PEM::PEM(scalar T, scalar P)
14 {
15     Pa_ = P;
16     Pc_ = P;
17     T_ = T;
18
19     setenv();
20
21 PEM::PEM(scalar T, scalar P, scalar Co2_g)
22 {
23     Pa_ = P;
24     Pc_ = P;
25     T_ = T;
26     Co2_g_ = Co2_g;
27
28     setenv();
29 };
30
31 /* ***** PEM setenv function ***** */
32 void PEM::setenv()
33 {
34     // number of electrons participating in a reaction
35     nh2_ = 2.0;
36     no2_ = 4.0;
37
38     // standard electromotive force
39     E0_ = 1.23;
40
41     // Properties of PEM
42     //! depth of GDL
43     LGDL_ = 300e-6;
44     //! depth of MEA
45     tm_ = 30.0e-6;
46     //! effective surface area per unit amount of platinum
47
48     return tm_/sigma_em_ * i(Ch2_g, Ch2_e);
49
50 inline scalar PEM::resistOV(scalar i)
51 {
52     set_resistOV();
53     return tm_/sigma_em_ * i;
54 };
55
56 inline void PEM::set_resistOV()
57 {
58     scalar lambda = 0;
59     if ( theta_a_ <= 1.0 ){
60         lambda = 0.043 + 17.8*theta_a_ - 39.8*std::pow(theta_a_,2) + 36.0*std::pow(theta_a_,3);
61     } else {
62         lambda = 14.1 + 1.4 * ( theta_a_ - 1.0 );
63     }
64
65     scalar sigma_m = (0.00514*lambda - 0.00326) * std::exp(1268*((1/303)-(1/T_)));
66     sigma_em_ = k_sigma_ * sigma_m;
67 };
68
69 /* ***** total overvoltage ***** */
70 inline scalar PEM::OV(scalar Ch2_g, scalar Ch2_e, scalar Co2_g)
71 {
72     return
73         activeOV(Ch2_g, Ch2_e, Co2_g)
74         + concOV(Ch2_g, Ch2_e, Co2_g)
75         + resistOV(Ch2_g, Ch2_e, Co2_g);
76 };
77
78 inline scalar PEM::OV(scalar i, scalar Ch2_g)
79 {
80     return activeOV(i) + concOV(i, Ch2_g) + resistOV(i);
81 };
82
83 // ***** cell voltage ***** */
84 inline scalar PEM::V
85 (
86     scalar Ch2_g, scalar Ch2_e, scalar Co2_g
87 )
88 {
89     return F(Ch2_g.n_ - Co2_g.n_ - OV(Ch2_g.n_ - Ch2_e.n_ - Co2_g.n_));
90 };
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145

```

setFieldsDict

```

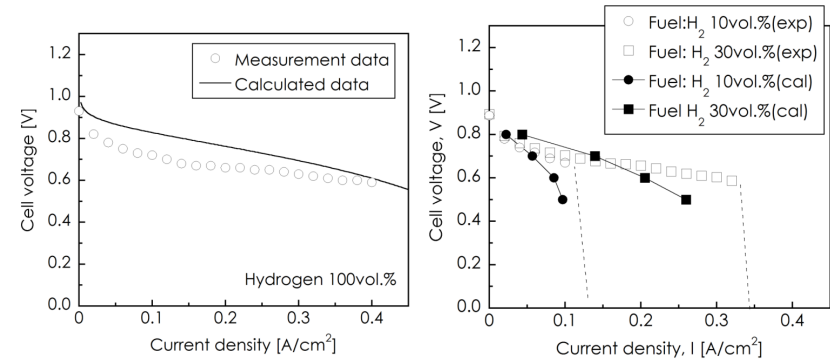
19 class dictionary;
20 object setFieldsDict;
21 };
22 // ***** //
23 defaultFieldValues
24 {
25     volScalarFieldValue mdotH2 0
26     volScalarFieldValue catalyst 0
27     volScalarFieldValue Kperm 0
28     volScalarFieldValue epsi 1
29 };
30 regions
31 {
32     boxToCell
33     {
34         box (0 0 0) (0.05 0.05 0.0001);
35         fieldValues
36         (
37             volScalarFieldValue mdotH2 0.1
38             volScalarFieldValue catalyst 1
39         );
40     }
41     boxToCell
42     {
43         box (0 0 0) (0.05 0.05 0.0003);
44         fieldValues
45         (
46             volScalarFieldValue epsi 0.4
47         );
48     }
49     boxToCell
50     {
51         box (0 0 0) (0.05 0.05 0.0003);
52         fieldValues
53         (
54             volScalarFieldValue Kperm 0.568181e11 // inverse of 1.76e-11
55         );
56     }
57 };
58 // ***** //
59 // ***** //
60 // ***** //
61 // ***** //

```

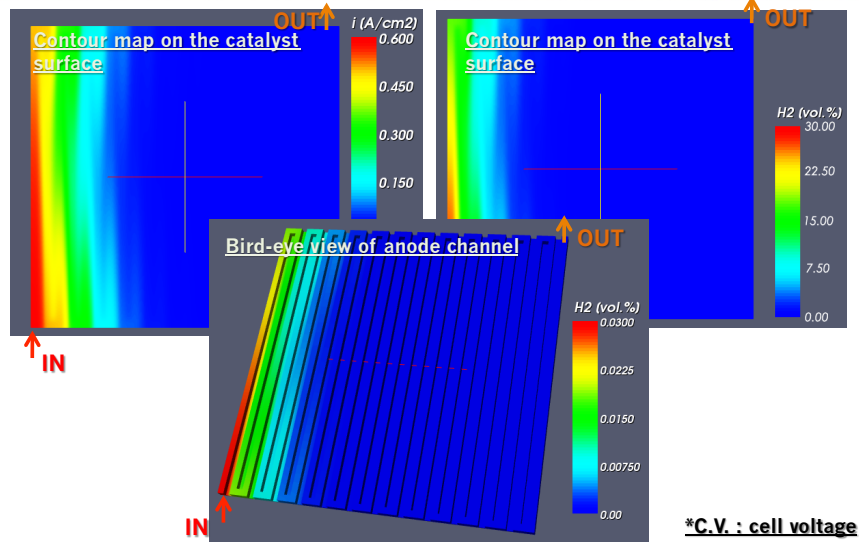
多孔質層の解析に際しては、
Darcy's drag forceにて流動抵抗を算出。

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。

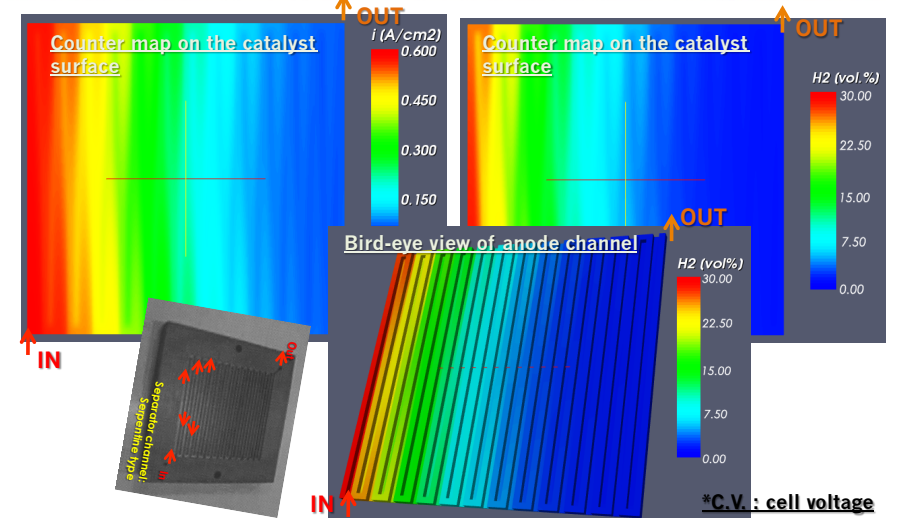
実測データとの比較 (ターフェルプロット)



SIMULATION RESULTS: H₂ 30vol.%, 100sccm, C.V. 0.5V



PEFC性能解析結果: H₂ 30vol.%, 200sccm, C.V. 0.5V



※ 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？