OpenFOAM® v.4.0
`RectingTwoPhaseEulerFoam`: `class ReadTable`

Simone Colucci, INGV-Pisa

June 2017.

Abstract

A new class, called ReadTable, has been created in InterfacialCompositionModels/interfaceCompositionModels for reading from tables the saturation surface and return multilinearly interpolated values. The class can be used for calculating the liquid-gas phase equilibria in two-phase multicomponent systems.

Note: this documentation is not approved not endorsed by the OpenFOAM Foundation or by ESI Ltd, the owner of OpenFOAM®.

1 Lookup table

Input tables in binary format have to be created using the following loop order:

```
count = 0
for(i=0;i<Np+1;i++) // pressure loop
{
  for(j=0;j<NT+1;j++)  // temperature loop
  {
    for(k=0;k<Nspecie1+1;k++)  // specie 1 loop
    {
      for(l=0;l<Nspecie2+1;l++) // specie 2 loop
      {
        [...]
        //  call your programm for calculating the saturation surface
        table_specie1[count] = myProgramm(p[i],T[j],specie1[k],specie2[l],...
        table_specie2[count] = myProgramm(p[i],T[j],specie1[k],specie2[l],...
        [...]
        count += 1;
        [...]
```

It is worth noting that for one-component systems pressure and temperature only are needed, since the phase equilibria are independent of components. For multicomponent systems phase equilibria depend also on the number of components, hence the number of loops equals the number of species + pressure and temperature. In the class constructor the tables are read and saved in the hash map `saturation`

```
forAllConstIter(hashedWordList, this->speciesNames_, iter)
{
  const label index = this->speciesNames_[*iter]; // specie number
```

```
    std::ifstream inFile(fileName_[index], std::ios::in | std::ios::binary);

    std::vector<double> dataTable(N_[0]);

    inFile.read(reinterpret_cast<char*>(&dataTable[0]),
                dataTable.size()*sizeof(dataTable[0]));

    int key = index;
    std::pair<int,std::vector<double>> key_values (key,dataTable);
    saturation_.insert (key_values);

    inFile.close();
}
```

`forAllConstIter` loops, for each phase, over all the components (i.e., species) identified by a number following the order used in *phaseProperties*. The hash map `saturation` is populated using the specie number as key and the values read from specie table.

2 Interpolation

Interpolation is performed in the member function `Yf`

```
template<class Thermo, class OtherThermo>
Foam::tmp<Foam::volScalarField>
Foam::interfaceCompositionModels::ReadTable<Thermo, OtherThermo>::Yf
(
    const word& speciesName,
    const volScalarField& Tf
) const
{
[...]
}
```

In the following, the generalized equations for multilinear interpolation will be derived. Bilinear interpolation is given by

$$\alpha_i \left[\alpha_j \mathbf{S}_{i+1,j+1} + (1-\alpha_j)\,\mathbf{S}_{i+1,j}\right] + (1-\alpha_i)\left[\alpha_j \mathbf{S}_{i,j+1} + (1-\alpha_j)\,\mathbf{S}_{i,j}\right]. \tag{1}$$

$\mathbf{S}$ is a $[N_x\ N_y]$ matrix storing the values of the function to be interpolated at given points (x, y) . For a given pair of values $(\overline{x}, \overline{y})$, with $\overline{x} \in [x_{min}, x_{max}]$ and $\overline{y} \in [y_{min}, y_{max}]$, the interpolation indexes, i and j, are given by

$$i = \frac{(\overline{x} - x_{min})}{\Delta x} = \frac{(\overline{x} - x_{min})}{x_{max} - x_{min}} N_x, \tag{2}$$

$$j = \frac{(\overline{y} - y_{min})}{\Delta y} = \frac{(\overline{y} - y_{min})}{y_{max} - y_{min}} N_y \tag{3}$$

The interpolation coefficients α_i and $\alpha_j \in [0, 1]$ are given by

$$\alpha_i = \frac{(\overline{x} - x_i)}{x_{i+1} - x_i} \tag{4}$$

$$\alpha_j = \frac{(\overline{y} - y_j)}{y_{j+1} - x_j} \tag{5}$$

Since

$$x_i = \frac{(x_{max} - x_{min})}{N_x} i + x_{min} \tag{6}$$

$$y_j = \frac{(y_{max} - y_{min})}{N_y} j + y_{min} \tag{7}$$

the interpolation coefficients can be written as

$$\alpha_i = \frac{N_x \overline{x} - N_x x_{min} - i\,(x_{max} - x_{min})}{x_{max} - x_{min}} \tag{8}$$

$$\alpha_i = \frac{N_y \overline{y} - N_y y_{min} - j\,(j_{max} - j_{min})}{y_{max} - y_{min}} \tag{9}$$

Equations above can be generalized to a multilinear interpolation. To linearly interpolate a N variable function, we define an [N X 2] index matrix I and [N X 2] coefficient matrix A

$$\mathbf{I} = \begin{bmatrix} (1-\alpha_0) & \alpha_0 \\ . & . \\ . & . \\ . & . \\ (1-\alpha_{N-1}) & \alpha_{N-1} \end{bmatrix}, \tag{10}$$

$$\mathbf{A} = \begin{bmatrix} i & i+1 \\ . & . \\ . & . \\ . & . \\ n & n+1 \end{bmatrix}, \tag{11}$$

$$\tag{12}$$

```
// Lists of indexes (index matrix I)
PtrList<PtrList<volScalarField> > Index(yindex_0.size());
// Lists of coefficients (coefficient matrix A)
PtrList<PtrList<volScalarField> > Alpha(yindex_0.size());

// set pressure index [0]
Index.set(0, new PtrList<volScalarField>(2));

Index[0].set // set i    [0][0]
(
    0,
    new volScalarField
    (
        IOobject
        (
            "Index.pressure" ,
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        Np_[0]*(p - one_p*pmin_[0])/( one_p*pmax_[0] - one_p*pmin_[0] )
    )
);
```

```
// correct internalField
forAll(Index[0][0],celli)
{
    Index[0][0][celli] = floor(Index[0][0][celli]);
}

Index[0].set // set (i+1)    [0][1]
(
    1,
    new volScalarField
    (
        IOobject
        (
            "Index.pressure",
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        Index[0][0] + one_
    )
);

// set pressure coefficients [0]
Alpha.set(0, new PtrList<volScalarField>(2));

Alpha[0].set // set (1-alpha)    [0][0]
(
    0,
    new volScalarField
    (
        IOobject
        (
            "Alpha.pressure" ,
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        one_ - (Np_[0]*p - Np_[0]*one_p*pmin_[0] -
        one_p*(pmax_[0]-pmin_[0])*Index[0][0])/( one_p*(pmax_[0]-pmin_[0]))
    )
);

Alpha[0].set // set alpha    [0][1]
(
    1,
    new volScalarField
    (
        IOobject
        (
            "Alpha.pressure",
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        (Np_[0]*p - Np_[0]*one_p*pmin_[0] -
        one_p*(pmax_[0]-pmin_[0])*Index[0][0])/( one_p*(pmax_[0]-pmin_[0]))
    )
);

// set temperature index [1]
Index.set(1, new PtrList<volScalarField>(2));

Index[1].set // set i    [1][0]
(
    0,
    new volScalarField
    (
        IOobject
```

```
        (
            "Index.temperature" ,
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        NT_[0]*(T - one_T*Tmin_[0])/( one_T*Tmax_[0] - one_T*Tmin_[0] )
    )
);

// correct internalField
forAll(Index[1][0],celli)
{
    Index[1][0][celli] = floor(Index[1][0][celli]);
}

Index[1].set // set (i+1)    [1][1]
(
    1,
    new volScalarField
    (
        IOobject
        (
            "Index.temperature",
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        Index[1][0] + one_
    )
);

// set temperature coefficients [1]
Alpha.set(1, new PtrList<volScalarField>(2));

Alpha[1].set // set (1-alpha)    [1][0]
(
    0,
    new volScalarField
    (
        IOobject
        (
            "Alpha.temperature" ,
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        one_ - (NT_[0]*T - NT_[0]*one_T*Tmin_[0] -
        one_T*(Tmax_[0]-Tmin_[0])*Index[1][0])/( one_T*(Tmax_[0]-Tmin_[0]))
    )
);

Alpha[1].set // set alpha    [1][1]
(
    1,
    new volScalarField
    (
        IOobject
        (
            "Alpha.temperature",
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        (NT_[0]*T - NT_[0]*one_T*Tmin_[0] -
        one_T*(Tmax_[0]-Tmin_[0])*Index[1][0])/( one_T*(Tmax_[0]-Tmin_[0]))
    )
```

```
);

if( this->speciesNames_.size() > 1 )
{
    // set coefficient for species
    forAllConstIter(hashedWordList, this->speciesNames_, iter)
    {
        const label indexList = this->speciesNames_[*iter];

        // set species index [2:Nspecies-1]
        Index.set(indexList+2, new PtrList<volScalarField>(2));

        Index[indexList+2].set // set i    [indexList][0]
        (
            0,
            new volScalarField
            (
                IOobject
                (
                    "Index." + *iter,
                          this->pair_.phase1().time().timeName(),
                          this->pair_.phase1().mesh(),
                          IOobject::NO_READ,
                          IOobject::NO_WRITE
                ),
                Nspecie_[indexList]*( Yspecie[indexList] -
                one_*speciemin_[indexList])/( one_*speciemax_[indexList] -
                one_*speciemin_[indexList] )
            )
        );

        // correct internalField
        forAll(Index[indexList+2][0],celli)
        {
            Index[indexList+2][0][celli] = floor(Index[indexList+2][0][celli]);
        }

        Index[indexList+2].set // set (i+1)    [indexList][1]
        (
            1,
            new volScalarField
            (
                IOobject
                (
                    "Index." + *iter,
                          this->pair_.phase1().time().timeName(),
                          this->pair_.phase1().mesh(),
                          IOobject::NO_READ,
                          IOobject::NO_WRITE
                ),
                Index[indexList+2][0] + one_
            )
        );

        // set species coefficients [2:Nspecies-1]
        Alpha.set(indexList+2, new PtrList<volScalarField>(2));

        Alpha[indexList+2].set // set (1-alpha)    [indexList][0]
        (
            0,
            new volScalarField
            (
                IOobject
                (
                    "Alpha." + *iter,
                          this->pair_.phase1().time().timeName(),
                          this->pair_.phase1().mesh(),
                          IOobject::NO_READ,
                          IOobject::NO_WRITE
                ),
                one_ - (Nspecie_[indexList]*Yspecie[indexList] -
```

```
                Nspecie_[indexList]*one_*speciemin_[indexList] -
                (speciemax_[indexList]-speciemin_[indexList])*
                Index[indexList+2][0])/
                (speciemax_[indexList]-speciemin_[indexList])
            )
        );

        Alpha[indexList+2].set // set alpha    [indexList][1]
        (
            1,
            new volScalarField
            (
                IOobject
                (
                    "Alpha." + *iter,
                          this->pair_.phase1().time().timeName(),
                          this->pair_.phase1().mesh(),
                          IOobject::NO_READ,
                          IOobject::NO_WRITE
                ),
                (Nspecie_[indexList]*Yspecie[indexList] -
                 Nspecie_[indexList]*one_*speciemin_[indexList] -
                 (speciemax_[indexList]-speciemin_[indexList])*
                 Index[indexList+2][0])/
                 (speciemax_[indexList]-speciemin_[indexList])
            )
        );

    }
}
```

and a [2^{N} X N] matrix of the interpolation indexes Y.

$$\mathbf{Y} = \begin{bmatrix} 1 & 1 & 1 & . & . & . \\ 1 & 1 & 0 & . & . & . \\ 1 & 0 & 1 & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \end{bmatrix}. \tag{13}$$

(14)

The entries of Y are calculated by a loop

```
//- Define indexes for multilinear interpolation
if (  Nspecie_.size() == 1 )
{
    // bilinear
    int count = 1;
    int tot_key = pow(2,Nspecie_.size()+2);
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
        {
            // key [0,1,...,2^Nspecie )
            // values [0,1,...,Nspecie)
            std::pair<int,std::vector<int>> key_values (tot_key-count,{i,j});
            yindex_.insert(key_values);
            count += 1;
        }
    }
}
else if (  Nspecie_.size() == 2 )
{
    // quadrilinear
    int count = 1;
```

```
    int tot_key = pow(2,Nspecie_.size()+2);
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
        {
            for(int k=0;k<2;k++)
            {
                for(int l=0;l<2;l++)
                {
                    std::pair<int,std::vector<int>>
                    key_values (tot_key-count,{i,j,k,l});

                    yindex_.insert(key_values);
                    count += 1;
                }
            }
        }
    }
}
```

In our case the variables of the interpolation function are pressure, temperature and the N components (i.e., species) defined in *phaseProperties*.

Multilinear interpolation is given by

$$\mathbf{A}_{0,\mathbf{Y}_{0,0}} \cdot \mathbf{A}_{1,\mathbf{Y}_{0,1}} \cdot \ldots \cdot \mathbf{A}_{N-1,\mathbf{Y}_{0,N-1}} \cdot \mathbf{S}_{\mathbf{I}_{0,\mathbf{Y}_{0,0}},\ldots,I_{N-1,\mathbf{Y}_{0,N-1}}} + \ldots \tag{15}$$

$$\mathbf{A}_{0,\mathbf{Y}_{2^N,0}} \cdot \mathbf{A}_{1,\mathbf{Y}_{2^N,1}} \cdot \ldots \cdot \mathbf{A}_{N-1,\mathbf{Y}_{2^N,N-1}} \cdot \mathbf{S}_{\mathbf{I}_{0,\mathbf{Y}_{2^N,0}},\ldots,\mathbf{I}_{N-1,\mathbf{Y}_{2^N,N-1}}} \tag{16}$$

Since in the code the values in the lookup table *saturation* are identified by a single index, representing a combination of N indexes, the tensor $\mathbf{S}$ has to redefined as a function of a single index, k

$$\begin{cases} k_{0,l} = \mathbf{I}_{0,l}, \\ k_{N,l} = k_{N-1} + \mathbf{I}_{N,l} \prod_{j=0}^{N-1} N_j \end{cases} \tag{17}$$

$$\mathbf{A}_{0,\mathbf{Y}_{0,0}} \cdot \mathbf{A}_{1,\mathbf{Y}_{0,1}} \cdot \ldots \cdot \mathbf{A}_{N-1,\mathbf{Y}_{0,N-1}} \cdot \mathbf{S}_{k_{N-1,\mathbf{Y}_{0,N-1}}} + \ldots \tag{18}$$

$$\mathbf{A}_{0,\mathbf{Y}_{2^N,0}} \cdot \mathbf{A}_{1,\mathbf{Y}_{2^N,1}} \cdot \ldots \cdot \mathbf{A}_{N-1,\mathbf{Y}_{2^N,N-1}} \cdot \mathbf{S}_{k_{N-1,\mathbf{Y}_{2^N,N-1}}} \tag{19}$$

```
// interpolation

volScalarField F_lookup
(
    IOobject
    (
        "F_lookup",
            this->pair_.phase1().time().timeName(),
            this->pair_.phase1().mesh(),
            IOobject::NO_READ,
            IOobject::NO_WRITE
    ),
    this->pair_.phase1().mesh(),
    dimensionedScalar("F_lookup", dimensionSet(0,0,0,0,0,0,0), 0)
);

std::vector<int> Nvector(yindex_0.size()); // number of values for each variable
Nvector[0] = Np_[0]+1;
Nvector[1] = NT_[0]+1;
for (int j=2; j< yindex_0.size(); j++)
{
    Nvector[j] = Nspecie_[j-2]+1;
}
```

```
int key = index; // index for specie
std::vector<double> saturation(saturation_.at(key)); // lookup table for specie

// loop over interpolation terms (from 0 to 2^Nspecie)
for (int i=0;i<std::pow(2, yindex_0.size() );i++)
{
     volScalarField Alpha_Pi
    (
        IOobject
        (
            "Alpha_Pi",
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        this->pair_.phase1().mesh(),
        dimensionedScalar("Alpha_Pi", dimensionSet(0,0,0,0,0,0,0), 1)
    );

    std::vector<int> yindex(yindex_.at(i));

    // loop over variables (p, T, species)  to calculate alpha Pi
    for (int j=0; j< yindex_0.size(); j++)
    {
        Alpha_Pi = Alpha[j][yindex[j]]*Alpha_Pi;
    }

    // calculate single index for lookup table
    volScalarField K
    (
        IOobject
        (
            "K",
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        // initialize with value for last specie
        Index[yindex_0.size()-1][yindex[yindex_0.size()-1]]
    );

    for (int j=2; j<yindex_0.size()+1; j++)
    {
        int N_Pi = 1;
        for (int k=1; k< j; k++)
        {
            // calculate N Pi
            N_Pi = Nvector[yindex_0.size()-k]*N_Pi;
        }
        K += Index[yindex_0.size()-j][yindex[yindex_0.size()-j]]*N_Pi;
    }

    // evaluate the ith interpolation term
    volScalarField F_lookup_last
    (
        IOobject
        (
            "F_lookup_last",
                  this->pair_.phase1().time().timeName(),
                  this->pair_.phase1().mesh(),
                  IOobject::NO_READ,
                  IOobject::NO_WRITE
        ),
        this->pair_.phase1().mesh(),
        dimensionedScalar("F_lookup_last", dimensionSet(0,0,0,0,0,0,0), 1)
    );

    forAll(K,celli)
    {
```

```
        int kindx = K[celli];

        F_lookup_last[celli] = Alpha_Pi[celli]*saturation[kindx];
    }

    // calculate iteratively the interpolation function for the specie
    F_lookup += F_lookup_last;
```