Como primer paso, se deben llamar a todas las librerías y modelos necesarios para el correcto funcionamiento de la Libreria Sur.

In [1]:
```
%config InlineBackend.close_figures=False
%matplotlib inline
from matplotlib import interactive
interactive(False)
from sur.models import *
from sur.plots import multiplot
```

Se debe llamar a la clase "Mixture" y crear un objeto dentro de ella, que para nosotros será equivalente a crear el sistema con el cual se trabajará de aquí en adelante. El sistema será llamado a elección por el usuario. Se recomienda un nombre sencillo y que rápidamente lo identifique/ diferencie de otros posibles sistemas que se carguen en la misma sesión. En este caso llamaremos GN al sistema ya que se trata de un fluido "Gas Natural"

In [2]:
```
GN = Mixture()
```

Ahora se debe "llenar" el sistema creado. Esto significa incorporarle constituyentes y sus correspondientes composiciones al objeto creado GN. Esto se puede realizar de varias formas y dependerá de la complejidad del sistema. Una de las maneras es crear dos objetos que pueden llamarse: "constituyentes" y "fracciones" en los cuales se detallarán los compuestos químicos que componen el sistema y sus correspondientes composiciones.

In [3]:
```
constituyentes= """methane ethane propane isobutane n-butane benzene"""
```

In [4]:
```
fracciones= """0.84080 0.09973 0.04037 0.00603 0.01012 0,002959"""
```

Note que los compuestos fueron escritos por su nombre en inglés pero también podrían haberse introducido sus fórmulas moleculares. Sin embargo, se recomienda esta forma ya que no existe posibilidad de confusión con sus isómeros. En cuanto a las composiciones, todas fracciones molares, los números decimales son separados por PUNTOS y la suma de las fracciones debe sumar siempre la suma de 1. Caso contrario el sistema arrojará error.

Ahora debemos cargar los objetos anteriores en el sistema GN.

In [5]:
```
GN.add_many(constituyentes, fracciones)
```

Para poder visualizar el sistema sólo se debe escribir su nombre y luego presionar Shift+enter.

In [6]:
```
GN
```

Out[6]:
```
[(<Compound: METHANE>, Decimal('0.8408')), (<Compound: ETHANE>, Decimal('0.09973')), (<C
ompound: PROPANE>, Decimal('0.04037')), (<Compound: ISOBUTANE>, Decimal('0.00603')), (<C
ompound: n-BUTANE>, Decimal('0.01012')), (<Compound: BENZENE>, Decimal('0.00295'))]
```

Antes de avanzar con lo siguiente es recomendable que se organice el sistema por orden creciente de peso molecular, esto es importante cuando se carguen los parametros de interacción, como veremos más adelante.

In [7]:
```
GN.sort(True)
GN
```

Out[7]:
```
[(<Compound: METHANE>, Decimal('0.8408')), (<Compound: ETHANE>, Decimal('0.09973')), (<C
ompound: PROPANE>, Decimal('0.04037')), (<Compound: ISOBUTANE>, Decimal('0.00603')), (<C
ompound: n-BUTANE>, Decimal('0.01012')), (<Compound: BENZENE>, Decimal('0.00295'))]
```

Ahora se procede al cálculo termodinámico. Se debe especificar una ecuacion de estado (PR/SRK/RKPR), un criterio para definir al parámetro de interacción Kij y luego otro para el parámetro Lij. En primer lugar elegiremos por ejemplo la EoS PR, valores de Kij constantes (que deberemos cargar a continuación) y valores para Lij iguales a cero.

In [8]:
```
setup = EosSetup.objects.create(eos='PR', kij_mode=EosSetup.CONSTANTS,  lij_mode=EosSetu
p.ZERO)
```

Cuando se deben utilizar otros valores de parámetros de interacción que no son los valores "default" propuestos, se deben ingresar manualmente y reemplazarán en esta sesión a los valores predeterminados. Si la cantidad de valores a cargar es importante, entonces es recomendable cargarlos en forma de matriz. La matriz está ordenada según el peso molecular creciente, es por esto que era útil ordenar el sistema anteriormente utilizando la función "Sort".

In [9]:
```
matrizGN = """0 0.0114 0.0167 0.0218 0.0218 0
0 0 0.0011 0.0096 0.0096 0
0 0 0 0.0033 0.0033 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0"""
```

Este objeto denominado "matrizGN" en este caso, sólo lee los valores cargados desde la diagonal de la matriz hacia la derecha, por lo que no es necesario cargar dos veces los parámetros, pero si es necesario cargar los valores de la diagonal hacia la derecha. Recordar nuevamente que los valores decimales se escriben con PUNTO.

In [10]:
```
setup.set_interaction_matrix('kij', GN, matrizGN)
```

En el paso anterior, lo que se hizo fue cargar los valores asignados en "matrizGN" ingresados por el usuario a las funciones que se encargan de reemplazar los valores predeterminados por esos valores y se les asignan su categoria correspondiente, es decir: qué tipo de parametros son y a qué sistema corresponden.

Si se quiere visualizar la matriz de parametros Kij para el sistema GN en la EoS PR debemos escribir la siguiente línea:

In [11]:
```
setup.kij(GN)
```

Out[11]:
```
array([[ 0.    ,  0.0114,  0.0167,  0.0218,  0.0218,  0.    ],
       [ 0.0114,  0.    ,  0.0011,  0.0096,  0.0096,  0.    ],
       [ 0.0167,  0.0011,  0.    ,  0.0033,  0.0033,  0.    ],
       [ 0.0218,  0.0096,  0.0033,  0.    ,  0.    ,  0.    ],
       [ 0.0218,  0.0096,  0.0033,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ]])
```

Ahora ya se poseen todos los elementos necesarios para realizar el cálculo de la envolvente de fase del sistema GN. Recordar el nombre que se le de a este objeto, ya que en caso de querer comparar dos EoS distintas o en el caso de querer comparar con información experimental, deberemos llamar nuevamente a este elemento.

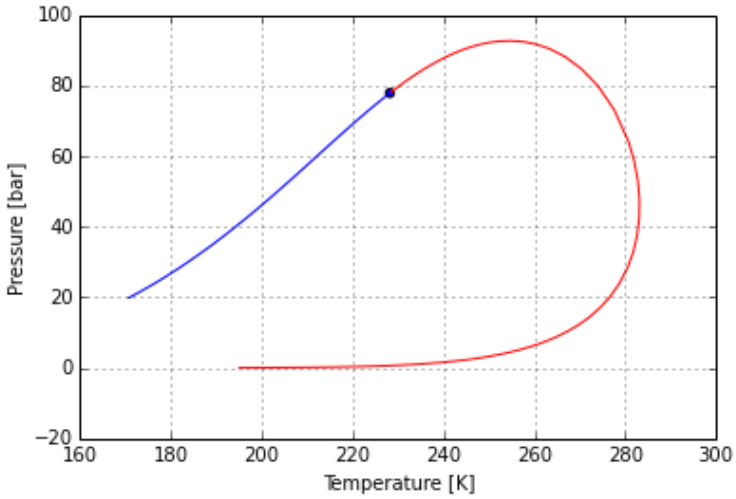
In [12]:
```
envolvente_GN_PR= GN.get_envelope(setup)
envolvente_GN_PR
```

Out[12]:
```
<EosEnvelope: PR - kij constants - lij zero>
```

Con el paso anterior hemos calculado la envolvente de GN con la EoS PR y parametros Kij constantes // Lij cero. Ahora es necesario visualizar esta envolvente:

In [13]:
```
Fig_envolvente_GN_PR= envolvente_GN_PR.plot()
Fig_envolvente_GN_PR
```

Out[13]:



Si se dispone de información experimental del sistema (puntos de burbuja o de rocío), se pueden incorporar para posteriormente comparar dichos datos con los modelos de cálculo.

Similarmente a lo realizado con la creación de los objetos "constituyentes" y "fracciones" para cargar el sistema, debemos crear nuevamente dos objetos para las temperaturas y presiones experimentales que pueden llamarse por ejemplo "temperatura_exp_GN" y "presion_exp_GN". Allí cargaremos los pares de T y P correspondientes a puntos de saturacion del sistema GN. Las unidades aceptadas son grados Kelvin para las temperaturas y Bares para las presiones.

In [14]:
```
temperatura_exp_GN= """259,4
261,1
262,9
264,2
265,5
266,8
268,6
270,5
271,4
273
273,6
275,1
276
276,4
276,7
277,1
277,3
277,3
276,9
276,2
275,2
273,7
271,4
269,4
268,1
264,9
260
255,2"""
```

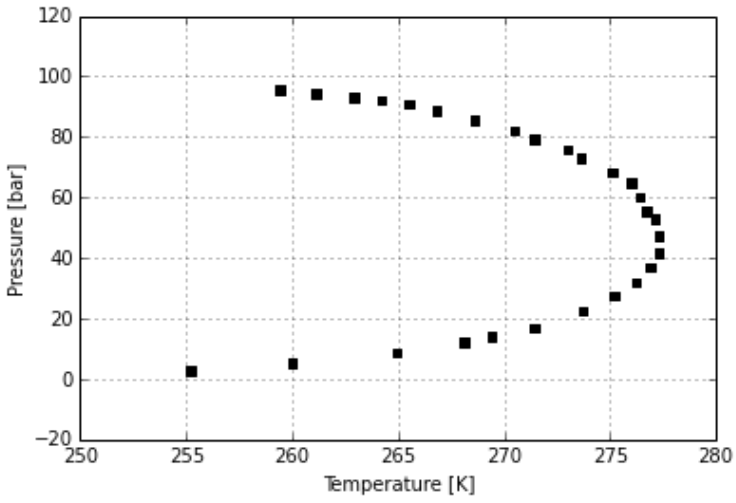
Note que en el caso de los datos experimentales, es indistinto cargar los números decimales con PUNTO o COMA, el sistema los lee indistintamente.

In [15]:
```
presion_exp_GN= """95,5
94,4
93,2
92
90,8
88,7
85,7
82
79,4
76
73,1
68,2
64,8
60,1
55,5
53
47,2
41,6
37
32
27,6
22,4
16,9
14
12,1
8,8
5,4
2,9"""
```

Ahora nuevamente, como se hizo previamente en la carga inicial del sistema, se entregan los objetos de datos experimentales a una funcion que grafica la "envolvente experimental", similar a lo que ocurre con la envolvente calculada.

In [16]:
```
exp_envGN = GN.experimental_envelope(temperatura_exp_GN, presion_exp_GN)
exp_figGN = exp_envGN.plot()
exp_figGN
```

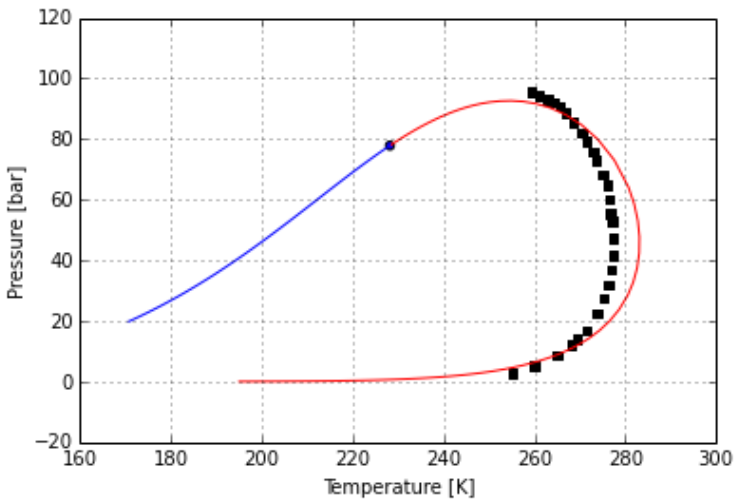
Out[16]:



Ahora podemos comparar la envolvente calculada con PR y los puntos experimentales del siguiente modo:

In [17]:
```
envolvente_GN_PR.plot(exp_figGN)
```

Out[17]:



In [18]:
```
methane = GN.compounds [0]
methane.delta1 = 0.50
methane.save()
```

In [19]:
```
c2h6= GN.compounds [1]
c2h6.delta1= 0.8
c2h6.save()
```

In [20]:
```
c3h8= GN.compounds [2]
c3h8.delta1= 1.60
c3h8.save()
```

In [21]:
```
c4h10=GN.compounds [4]
c4h10.delta1= 1.82758999
c4h10.save()
```

In [22]:
```
for compound in GN.compounds:
    print compound.delta1
    print compound._get_eos_params('RKPR')
```

```
0.5
[ 2.30376808  0.0304338   0.5         1.54083759]
0.8
[ 5.61628283  0.04561112  0.8         1.90385075]
1.6
[ 9.77170891  0.06017136  1.6         1.96439211]
None
[ 13.88620375   0.07725098   1.63021558   2.08497025]
1.82758999
[ 14.61252217   0.07602789   1.82758999   2.11479414]
None
[ 20.20708826   0.07544854   2.22553038   2.05873355]
```

Podemos volver a repetir el cálculo de la envolvente de fase con otra ecuación de estado tal como RKPR. Volvemos a llamar a las funciones correspondientes:

In [23]:
```
setup = EosSetup.objects.create(eos='RKPR', kij_mode=EosSetup.T_DEP,  lij_mode=EosSetup.
CONSTANTS)
```

En este caso se ha elegido EoS RKPR y en cuanto a los criterios de cálculo de los parámetros de interacción hemos seleccionado: valores de Kij dependientes de la temperatura (para lo cual se deberán ingresar los valores correspondientes de K0ij) y Lij valores constantes.

Cuando seleccionamos Kij_mode=EosSetup.T_DEP, además de tener que ingresar los valores de K0ij, debemos dejar asentado el criterio de la T* (T star). Para nosotros, este valor de T, toma el valor de la Temperatura Critica del componente más volátil en el par de compuestos que se esté analizando. Por ejemplo: si deseamos calcular Kij de C1 con C2, el T star en este caso será la temperatura critica de C1 que es más volatil que C2.

In [24]:
```
for c1, c2 in combinations(GN.compounds, 2):
    t = c1.tc if c1.weight < c2.weight else c2.tc
    setup.set_interaction('tstar', c1, c2, t)
```

Ahora debemos cargar los valores de los parámetros de interacción. Aquí utilizaremos una manera distinta a la utilizada anteriormente para el cálculo con PR. Se cargarán los valores de a uno por vez. Se recomienda este método cuando el sistema contenga pocos componentes y en consecuencia, la cantidad de parametros de interacción a ingresar sea reducida.

In [25]:
```
setup.set_interaction('k0', 'methane', 'n-butane', 0.02178)
```

Out[25]:
```
<K0InteractionParameter: K0InteractionParameter object>
```

Ya se encuentra cargado el set de interacciones "K0ij" entre metano y n-butano. Ahora procederemos a cargar su valor Lij.

In [26]:
```
setup.set_interaction('lij', 'ethane', 'n-butane', -0.04378)
```

Out[26]:
```
<LijInteractionParameter: LijInteractionParameter object>
```

Seguimos de esta manera, cargando cada uno de los parametros (ya sean K0ij o Lij) para todos los compuestos del sistema GN.

In [27]:
```
setup.set_interaction('k0', 'ethane', 'n-butane', 0)
setup.set_interaction('lij', 'ethane', 'n-butane', -0.04378)
setup.set_interaction('k0', 'propane', 'n-butane', 0)
setup.set_interaction('lij', 'propane', 'n-butane', -0.01158)
```

Out[27]:
```
<LijInteractionParameter: LijInteractionParameter object>
```

Una vez cargados todos los parámetros correspondientes, estamos en condiciones de calcular la envolvente del sistema con la ecuación RKPR. Se realiza de manera análoga a lo ya descrito para la EoS PR.

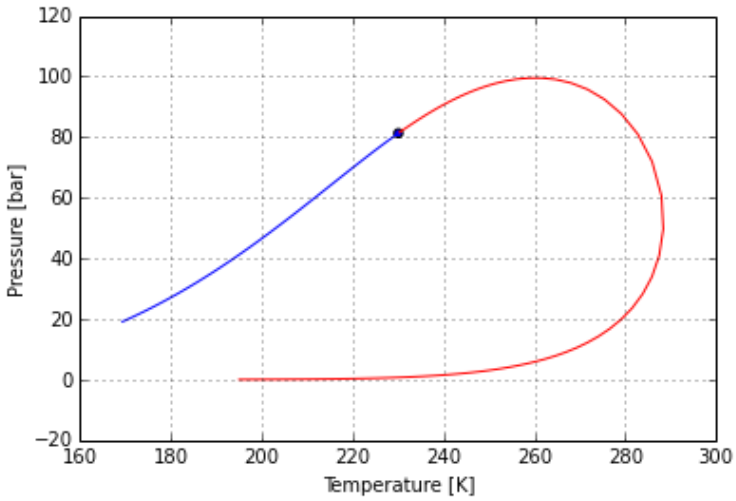
In [28]:
```
envolvente_GN_RKPR= GN.get_envelope(setup)
envolvente_GN_RKPR
```

Out[28]:
```
<EosEnvelope: RKPR - kij t_dep - lij constants>
```

Para visualizar la envolvente obtenida debemos escribir la siguiente linea:

In [29]:
```
figGN_RKPR= envolvente_GN_RKPR.plot()
figGN_RKPR
```

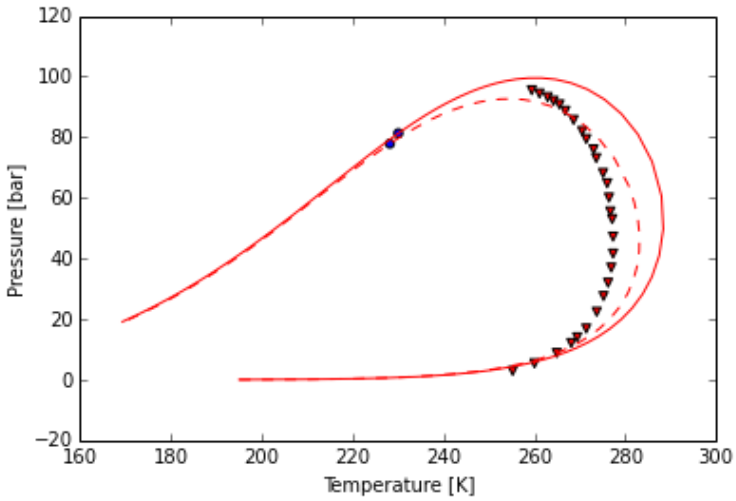
Out[29]:



Una vez obtenido todos los gráficos necesarios (envolvente calculada con PR, envolvente calculada con RKPR, envolvente de datos experimentales), podemos graficarlos todos juntos con objeto de compararlos a través de la función "multiplot"

In [30]:
```
figGN= multiplot([envolvente_GN_RKPR, envolvente_GN_PR],[exp_envGN], formats=['r','--r']
, experimental_colors=['red'], experimental_markers=['v'])
figGN
```

Out[30]:



Se eligió la linea punteada para representar la ecuación PR, la continua para RKPR y los puntos experimentales son representados por los triangulos rojos. Se pueden modificar todos aquellos parametros concernientes a los graficos.

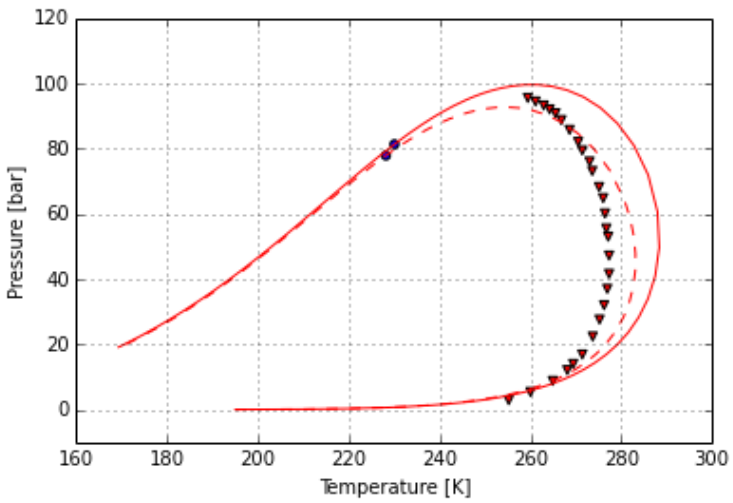
In [31]:
```
ax = figGN.gca()
```

In [32]:
```
ax.set_ylim([-10,120]),ax.grid(True)
```

Out[32]:
```
((-10, 120), None)
```

In [33]:
```
figGN
```

Out[33]:



Para guardar los graficos se debe realizar lo siguiente:

In [34]:
```
figGN.savefig('figuradeGN.png', dpi=900)
```