Objetivo del laboratorio

El objetivo de la presenta práctica es conocer el estándar de simulación de circuitos SPICE y realizar pequeñas simulaciones en corriente continua con el mismo. SPICE es una forma elegante y sencilla de codificar circuitos eléctricos de manera que puedan ser procesados por un ordenador. Mediante un sencillo lenguaje podemos definir resistencias, fuentes de alimentación, etc., las conexiones entre ellos y los resultados que deseamos obtener.

El estándar SPICE

SPICE es una abreviabiación de *Simulation Program with Integrated Circtuit Emphasis*. Se trata básicamente de un método estándar para describir circuitos usando texto plano en lugar de una representación gráfica (o *esquemática*). A esta descripción en texto se la llama también **netlist** y básicamente se corresponde con la *lista* de los componentes del circuito y cómo estos están conectados entre sí, es decir, de los nodos de unión. Los ficheros netlist pueden tener extensiones `.cir`, `.net`, `.ckt`, ó `.sp` y es muy común encontrárselos con cualquiera de estas.

Existen en el mercado muchas variantes (intérpretes) de Spice, aunque el original fue descrito en la Universidad de Berkeley. En la lista de intérpretes de Spice tenemos desde esfuerzos y proyectos comerciales hasta *open source* y regidos por distintas comunidades de usuarios y programadores.

> **Pregunta:** Enumera todos los intérprete de Spice que puedas encontrar. Crea una tabla en Markdown con varias columnas (para el nombre, fabricante, versión actual, licencia y alguna característica sobresaliente). Aquí tienes un ejemplo del que puedes partir y seguir completando:

Intérprete	Licencia	Fabricante	Características
Ahkab	GPL	Giuseppe Venturini	Basado en Python

> **Pregunta:** ¿Qué comparación puedes efectuar entre C y Spice como estándares (lenguajes) y sus respectivas implementaciones en software? ¿Qué implementaciones reales (compiladores) del lenguaje C conoces?

Elementos de un netlist

Como acabamos de comentar, un netlist se corresponde con la codificación de los elementos electrónicos de un circuito y las uniones entre los mismos. Veamos con más concreción qué partes y secciones lo componen.

Comentarios

La primera línea de un netlist se corresponderá siempre con un comentario. A partir de esta línea se pueden introducir más comentarios pero tienen que ir siempre precedidos de un `*`. Ejemplo:

```
Mi primer circuito
* Otro comentario
* más comentarios
*
```

Dispositivos básicos de un circuito

Los elementos de un netlist son los mismos que encontramos en cualquier circuito eléctrico sencillo, tales como resistencias, **condensadores**, **bobinas**, **interruptores**, **hilos** y **fuentes** de alimentación. Para distinguir uno de otro, se reserva una letra característica: `V` para fuentes de alimentación, `R` para resistencias, `C` para condensadores y `L` para bobinas. También es posible usar estas letras en su versión en minúscula (`r`, `v`, `c`, `l`, etc.). Después de esta letra característica se puede sufijar cualquier texto para diferenciar un elemento de otro (números, letras, palabras, etc.). Ejemplo:

```
* Una resistencia
R1
*  Otra resistencia
R2
* Fuente de alimentación
V
* Un condensador
Cprincipal
```

Conexiones

A continuación de indicar el elemento eléctrico, tenemos que informar a Spice cuáles son los puntos de unión tanto a un lado como al otro del elemento. Así es como Spice sabe qué está conectado a qué: porque comparten un **punto** (o **nodo**, aunque este término se reserva sobretodo a uniones de más de dos elementos) que hemos señalizado correctamente. Para nombrar nodos, lo mejor es emplear una numeración secuencial: 0...n. **La enumeración de los puntos de unión es completamente a nuestro criterio**.

```
* Una resistencia
* entre cables 0 y 1
R1 0 1
```

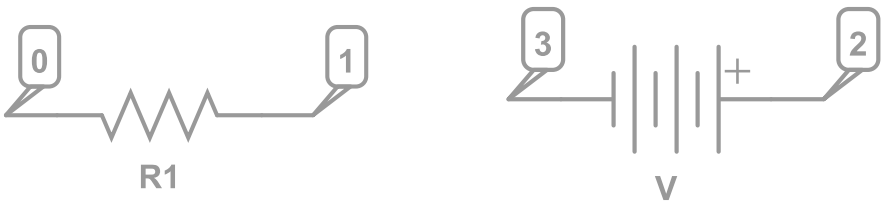
Sólo es necesario seguir un criterio: en el caso de una fuente de alimentación, el nodo que pondremos primero será aquel que está más cerca del *borne* positivo. Ejemplo:

```
* Para una fuente indicamos primeramente conexión a nodo positivo.
v 2 3 type=vdc vdc=1
```

En el *caso de LTspice* no es necesario indicar los parámetros `type=vdc` y `vdc=X`, sino que si no se especifica nada, se supone que el último valor es el del voltaje a corriente continua:

```
* Especificación de una fuente de alimentación de 10 V en corrient co
v 0 1 10
```

Aquí tienes un ejemplo gráfico de los componentes comentados justo arriba (resistencia y voltaje):



Unidades en SPICE

Las unidades de las magnitudes características del circuito son siempre unidades del Sistema Internacional y no es necesario indicarlo explícitamente en el netlist.

La forma de especificar múltiplos de estas cantidades es añadiendo una letra. Básicamente las que nos interesan y las que suelen aparecer mayoritariamente son `k` para "kilo-," `m` para "mili?" y `u` para "micro?".

> **Pregunta:** Crea una tabla en Markdown con todos los prefijos de múltiplos que puedas, su abreviatura y su equivalencia numérica.

En el caso de las fuentes de alimentación hemos de especificar si se trata de corriente contínua (`vdc`) o alterna (`ac`).

```
* Una resistencia de 5 Ohmios
R2 1 0 5
* Una pila de 10 Voltios (continua)
V1 1 0 type=vdc vdc=10
* Una resistencia de 5 kΩ
RX 2 4 5k
```

> **Pregunta**: ¿qué unidades del Sistema Internacional relacionadas con la asignatura –y los circuitos en general– conoces? Responde aquí mismo en una celda de Markdown con una tabla.

Valores iniciales

Aparecen justo al final de la definición del componente (`ic`). Suelen aplicarse principalmente con condensadores.

```
* Una condensador inicialmente no cargado
c 1 0 1u ic=0
```

Fin del circuito

El fin de la descripción de un netlist se especifica mediante el comando `.end`.

```
* Mi primer circuito
V 1 0 vdc=10 type=vdc
R 1 0 5
* Fin del circuito
.end
```

Comandos SPICE para circuitos en corriente continua

Además de la descripción del circuito, hemos de indicar al intérprete de Spice qué tipo de análisis queremos realizar en sobre el mismo y cómo queremos presentar la salida de la simulación. Los comandos en Spice empiezan por un `.` y suelen escribirse justo al final del circuito, pero antes del comando `.end`.

```
 Mi primer circuito
* Aquí van los componentes
R 1 0 6k
...
* Comandos
.op
...
* Fin del circuito
.end
```

> **Pregunta**: Hasta lo que has visto del lenguaje Spice, ¿dentro de qué tipo o conjunto de lenguajes encajaría? ¿Funcionales? ¿Específicos de dominio? ¿Procedurales? ¿Estructurados? ¿Orientado a Objetos ¿Funcionales? Justifica tu respuesta.

Veamos los principales comandos de simulación:

- `.op` es el comando más sencillo que podemos emplear en. Devuelve el voltaje e intensidad en cada ramal y componente del circuito. Este comando no necesita parámetros.
- `.dc` es uy parecido al comando `.op` pero nos permite cambiar el valor del voltaje de una fuente de alimentación en pasos consecutivos entre el valor A y el valor B. En el caso de que la fuente tuviera asignada ya un valor para su voltaje, este sería ignorado. Ejemplo:

```
* Variamos el valor del voltaje
* de la fuente "v" de 1 a 1000
* en pasos de 5 voltios
v 1 0 type=vdc vdc=10
.dc v 1 start=1 stop=1000 step=20
v2a 2 4 type=vdc vdc=9
* Igual para v2a. Se ignora su voltaje de 9V
.dc v2a start=0 stop=10 step=2
```

- El comando `.tran` realiza un análisis en el tiempo de los parámetros del circuito. Si no se emplea la directiva `uic` (*use initial conditions*) o esta es igual a cero, este análisis

se realiza desde el punto estable de funcionamiento del circuito hasta un tiempo `tfinal`. y en intervalos `tstep`. Si empleamos un varlor distinto para parámetro `uic`, entonces se hará uso de las condiciones iniciales definidas para cada componente (típicamente `ic=X` en el caso de los condensadores, que da cuenta de la carga incial que estos pudieran tener).

```
* Hacemos avanzar el tiempo entre
* tinicial y tfinal en pasos tstep
.tran tstart=X tstop=Y tstep=Z uic=0/1/2/3
```

`X`, `Y` y `Z` tienen, evidentemente unidades de tiempo en el S.I. (segundos).

> **Pregunta**: El parámetro `uic` puede tener varios valores y cada uno significa una cosa. Detállalo usando un celda Markdown y consultando la documentación de Ahkab.

Intérprete SPICE que vamos a usar: Ahkab

Tras un estándar siempre hay una o varias implementaciones. Ahkab no deja de ser una implmentación más en Python del estándar Spice.

> **Pregunta:** Comenta las distintas implementaciones de lenguajes y estándares que conozcas. Hazlo usando una tabla en Markdown. Aquí tienes un poco de ayuda (aunque antes ya se ha puesto el ejemplo de una tabla).**Pregunta:** Describe brevemente este software (creador, objetivos, versiones, licencia, características principales, dependencias, etc.).

Trabajo práctico

Muy bien, ahora toca definir circuitos y ejecutar simulaciones sobre los mismos gracias a Ahkab.

Instalación de bibliotecas necesarias

Si estás utilizando Anaconda, asegúrate de tener su entorno activado:

```
C:\> conda activate base (en el caso de Windows)
```

ó

```
$ source /usr/local/Caskroom/miniconda/base/bin/activate (en el caso
```

En el caso de Windows tienes que tener en el PATH el directorio donde se encuentre el comando `conda` (visita la sección de Environment Variables del Panel de Control). Si has instalado Anaconda con esta opción marcada, ya no tienes que preocuparte por ello.

Ahora ya puedes instalar Ahkab:

```
(base) $ pip install ahkab
```

Como siempre, una vez instalado cualquier framework para Python, ya lo podemos utilizar, tanto desde el REPL como desde un entorno Jupyter (Jupyter, Jupyterlab, VS Code o nteract). Recuerda que para usar el kernel Python (que viene con Anaconda) desde nteract debes seguir las instrucciones que se indican en su documentación oficial.

[1]
```
import pylab as plt
import ahkab
```

```
W: Locale appears not set! please export LANG="en_US.UTF-8" or
equivalent,
W: or ahkab's unicode support is broken.
```

También vamos a importar Sympy para hacer algún cálculo más *manual* más adelante:

[2]
```
import sympy.physics.units as u
from sympy.physics.units import Dimension
from sympy import *
from sympy.physics.units import convert_to
```

> **Pregunta:** ¿Qué es y para qué sirve PyLab?

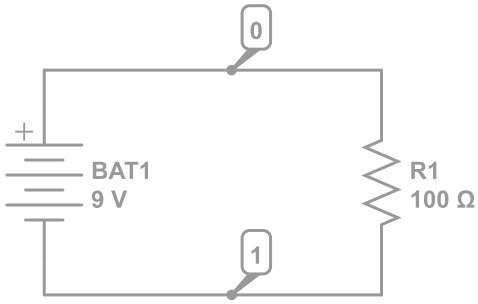
Circuitos sencillos para trabjar con la ley de Ohm:

La *mal llamada* ley de Ohm reza que el voltaje (la *energía por unidad de carga*) que se disipa en un tramo de un circuito eléctrico es equivalente a la intensidad (I) de la corriente (es decir, cuántos electrones circulan por unidad de tiempo) por la resistencia del material (R) en el que está desplazándose dicha corriente. Matemáticamente:

$$V = I \cdot R$$

> **Pregunta:** comprueba que la ecuación anterior está ajustada a nivel dimensional, es decir, que la naturaleza de lo que está a ambos lados del signo igual es la misma. Realiza este ejercicio con LaTeX en una celda Markdown.

Comencemos con el circuito más sencillo posible de todos:



Vamos a escribir su contenido (componentes o *netlist*) en disco con el nombre `circuito sencillo.sp`. Esto lo podemos lograr directamente y en tiempo real desde una celda de Jupyter gracias a los *comandos mágicos* de este entorno de programación literaria. En concreto vamos a utilizar `%%writefile` que guarda los contenidos de una celda como un fichero.

[3]
```
%%writefile "circuito sencillo.sp"
* Este es un circuito sencillo
r1 1 0 100
v1 0 1 type=vdc vdc=9
.op
.dc v1 start=0 stop=9 step=1
.end
```

```
Overwriting circuito sencillo.sp
```

Ahora vamos a leer su descripción con Ahkab, interpretar y ejecutar las simulaciones que en él estén descritas.

[4]
```
circuito_y_análisis =
ahkab.netlist_parser.parse_circuit('circuito sencillo.sp')
```

Separamos la información del netlist (componentes) de los análisis (uno de tipo `op` y otro de tipo `dc`):

[5]
```
circuito = circuito_y_análisis[0]
análisis_en_netlist = circuito_y_análisis[1]
lista_de_análisis = ahkab.netlist_parser.parse_analysis(circuito,
análisis_en_netlist)
print(lista_de_análisis)
```

```
[{'type': 'op', 'guess': True, 'x0': None}, {'type': 'dc', 'source':
'v1', 'start': 0.0, 'stop': 9.0, 'step': 1.0, 'sweep_type': 'LIN'}]
```

> **Pregunta:** ¿qué tipo de estructura de Python es `lista_de_análisis`?

Las simulaciones que implican listas de datos (`.dc`, `.tran`, etc.) necesitan de un fichero temporal (`outfile`) donde almacenar los resultados. Para ello tenemos que definir la propiedad `outfile`.

[6]
```
lista_de_análisis[1]['outfile'] = "simulación dc.tsv"
```

> **Pregunta:** escribe el código Python necesario para identificar qué análisis de `lista_de_análisis` son de tipo `dc` ó `tran` y sólo añadir la propiedad `outfile` en estos casos. Aquí tenéis un post de Stackoverflow con algo de ayuda. Un poco más de ayuda: el siguiente código (sí, una única línea) devuelve el índice de la simulación que es de tipo `dc`. Para simplificar un poco el ejercicio, suponed que, como máximo, habrá un análisis de tipo `tran` y/o `dc`.

[7]
```
[i for i, d in enumerate(lista_de_análisis) if "dc" in
d.values()][0]
```

```
1
```

Una vez que ya hemos separado netlists de simulaciones, ahora ejecutamos las segundas (¡todas a la vez!) gracias al método `.run` de Ahkab:

[8]
```
resultados = ahkab.run(circuito, lista_de_análisis)
```

```
Starting op analysis:
Calculating guess: skipped. (linear circuit)
Solving...  -\ done.
Solving...  -\ done.
Difference check within margins.
(Voltage: er=0.001, ea=1e-06, Current: er=0.001, ea=1e-09)
Starting DC analysis:
Solving...  -\|/-\|/-\|done
```

Resultados de la simulación `.dc`

Imprimimos información sobre la simulación de tipo `.dc`:

[9]
```
print(resultados['dc'])
```

```
<DC simulation results for '* este es un circuito sencillo' (netlist
circuito sencillo.sp). LIN sweep of V1 from 0 to 9 V. Run on 2019-12-05
19:08:19, data file simulación dc.tsv>
```

Veamos qué variables podemos dibujar para el caso del análisis `dc`.

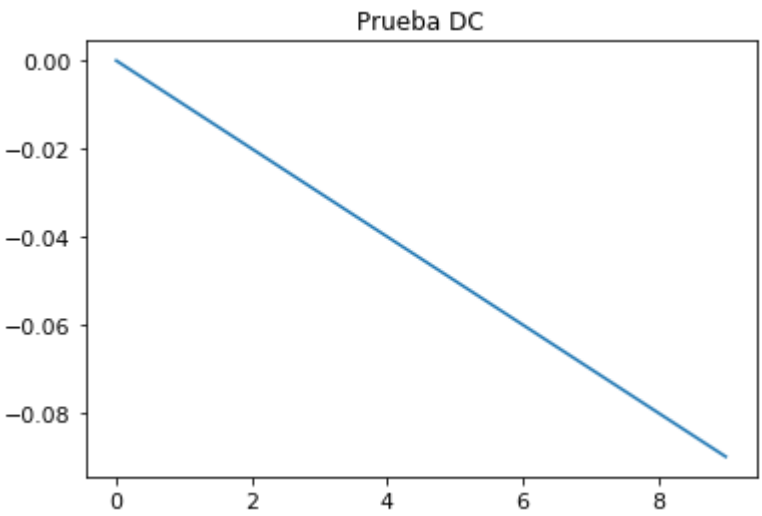
[10]
```
print(resultados['dc'].keys())
```

```
['V1', 'V1', 'I(V1)']
```

Y ahora graficamos el resultado del análisis anterior. Concretamente vamos a representar el voltaje en el borne 1 (`V1`) con respecto a la intensidad del circuito (`I(V1)`).

[11]
```
figura = plt.figure()
plt.title("Prueba DC")
plt.plot(resultados['dc']['V1'], resultados['dc']['I(V1)'],
label="Voltaje (V1)")
```

```
[<matplotlib.lines.Line2D at 0x129460a20>]
```



> **Pregunta:** comenta la gráfica anterior… ¿qué estamos viendo exactamente? Etiqueta los ejes de la misma convenientemente. Así como ningún número puede *viajar* solo sin hacer referencia a su naturaleza, ninguna gráfica puede estar sin sus ejes convenientemente etiquetados. Algo de ayuda. ¿Qué biblioteca estamos usando para graficar? Una pista.

Resultados de la simulación `.op`

El método `.results` nos devuelve un diccionario con los resultados de la simulación.

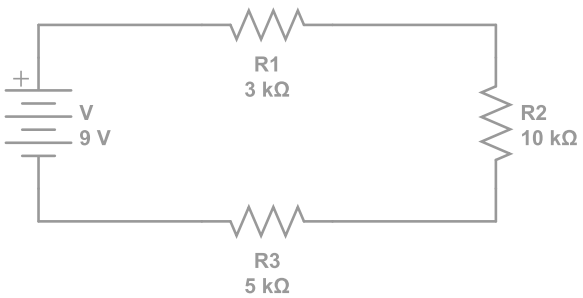
[12]
```
print(resultados['op'].results)
```

```
{V1: -9.0, I(V1): -0.09}
```

> **Pregunta:** justifica el sencillo resultado anterior (análisis op). Repite el cálculo con Sympy, atendiendo con mimo a las unidades y al formateo de los resultados (tal y como hemos visto en muchos otros notebooks en clase).

Análisis de circuito con resistencias en serie

Vamos a resolver (en punto de operación) el siguiente circuito:



Al igual que antes, grabamos el netlist en disco desde Jupyter gracias a la *palabra mágica* `%writefile`.

[13]
```
%%writefile "resistencias en serie.net"
```

```
* circuito con tres resistencias en serie
v1 1 0 type=vdc vdc=9
R1 0 2 3k
R2 2 3 10k
R3 3 1 5k
* análisis del circuito
.op
.end
```

```
Overwriting resistencias en serie.net
```

[14]
```
circuito_y_análisis =
ahkab.netlist_parser.parse_circuit('resistencias en serie.net')
circuito = circuito_y_análisis[0]
análisis_en_netlist = circuito_y_análisis[1]
lista_de_análisis = ahkab.netlist_parser.parse_analysis(circuito,
análisis_en_netlist)
resultados = ahkab.run(circuito, lista_de_análisis)
```

```
Starting op analysis:
Calculating guess: skipped. (linear circuit)
Solving...  -\ done.
Solving...  -\ done.
Difference check within margins.
(Voltage: er=0.001, ea=1e-06, Current: er=0.001, ea=1e-09)
```

Imprimos los resultados del análisis `.op`:

[15]
```
print(resultados['op'])
```

```
OP simulation results for '* circuito con tres resistencias en
serie'(netlist resistencias en serie.net).
Run on 2019-12-05 19:08:19, data file None.
Variable    Units      Value     Error    %
----------  -------  -------  --------  ---
V1          V          9       -9e-12      0
V2          V          1.5     -1.5e-12    0
V3          V          6.5     -6.5e-12    0
I(V1)       A         -0.0005   0          0
```

Los cantidades `V1`, `V2` y `V3` hacen referencia a los distintos valores del potencial que se ha perdido en cada uno de los bornes que has elegido para describir el netlist (`1`, `2`, etc.). Por ejemplo, podemos calcular el *potencial consumido* por la resistencia `R1` y verás que coincide con el del punto `V2` devuelto por Ahkab.

[16]
```
r1 = 3E3*u.ohms
intensidad_ahkab = resultados['op']['I(V1)'][0][0]*u.ampere
v2 = convert_to(intensidad_ahkab*r1, [u.volt])
pprint(v2)
```

```
-1.5·volt
```

> **Pregunta**: reproduce el resto de los valores anteriores de manera *manual* mediante Sympy (es decir, aplicando la ley de Ohm, pero con tun *toque computacional*). Te pongo aquí un ejemplo del que puedes partir… En él sólo calculo la corriente que circula por el circuito (sí, justo la que antes Ahkab ha devuelto de manera automática). Para ello necesito previamente computar la resistencia total (`r_total`). Faltarían el resto de resultados y convertirlos a unidades más *vistosas* (mediante la orden `convert_to` y `.n()`).

[17]
```
v1 = 9*u.volts
r1 = 3*u.kilo*u.ohms
r2 = 10*u.kilo*u.ohms
r3 = 5*u.kilo*u.ohms
r_total = r1 + r2 + r3
intensidad = u.Quantity('i')
intensidad.set_dimension(u.current)
ley_ohm = Eq(v1, intensidad*r_total)
solucion_para_intensidad = solve(ley_ohm, intensidad)
pprint(convert_to(solucion_para_intensidad[0], [u.ampere]).n(2))
```

```
0.0005·ampere
```

> **Pregunta**: Demuestra que se cumple la Ley de Kirchhoff de la energía en un circuito, es decir, que la suma de la energía suministrada por las fuentes (pilas) es igual a la consumida por las resistencias. Realiza la operación con Sympy.

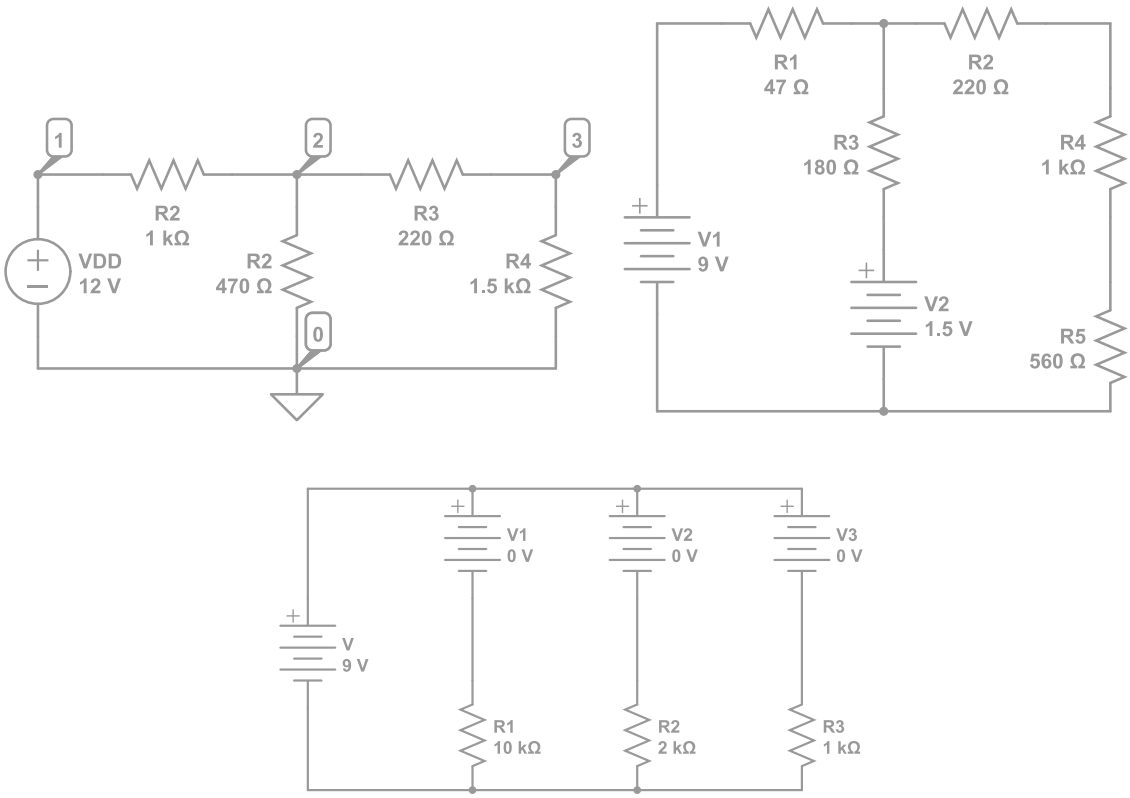
$$\sum_{i}^{N} V_{\text{fuentes}} = \sum_{j}^{M} V_{\text{consumido en resistencias}}$$

Ten en cuenta que en este caso sólo hay una fuente.

Análisis `.op` de circuitos con resistencias en paralelo

Vamos a complicar un poco el trabajo añadiendo elementos en paralelo.

> **Pregunta**: realiza los análisis `.op` de los siguientes circuitos. Para ello crea un netlist separado para cada uno donde queden correctamente descritos junto con la simulación (`.op`). Comenta los resultados que devuelve Ahkab (no imprimas los resultados de las simulaciones *sin más*).



Aquí tienes el análisis del primer circuito, para que sirva de ejemplo:

[18]
```
%%writefile "resistencias en paralelo 1.cir"
* resistencias en paralelo
vdd 0 1 vdc=12 type=vdc
r2 1 2 1k
r3 2 3 220
r4 3 0 1.5k
r5 2 0 470
.op
.end
```

```
Overwriting resistencias en paralelo 1.cir
```

[19]
```
circuito_y_análisis =
ahkab.netlist_parser.parse_circuit('resistencias en paralelo
```

```
1.cir')
circuito = circuito_y_análisis[0]
análisis_en_netlist = circuito_y_análisis[1]
lista_de_análisis = ahkab.netlist_parser.parse_analysis(circuito,
análisis_en_netlist)
resultados = ahkab.run(circuito, lista_de_análisis)
```

```
Starting op analysis:
Calculating guess: skipped. (linear circuit)
Solving...  -\ done.
Solving...  -\ done.
Difference check within margins.
(Voltage: er=0.001, ea=1e-06, Current: er=0.001, ea=1e-09)
```

Imprimimos los resultados del análisis `.op`. Como puedes comprobar, Ahkab sólo reporta la intensidad de corriente en las ramas en las que hay una pila (en este caso, la rama donde está la pila `VDD`).

[20]
```
print(resultados['op'])
```

```
OP simulation results for '* resistencias en paralelo'(netlist
resistencias en paralelo 1.cir).
Run on 2019-12-05 19:08:19, data file None.
Variable    Units           Value        Error    %
----------  -------  ------------  -----------  ---
V1          V        -12           1.2e-11        0
V2          V         -3.23533     3.23533e-12    0
V3          V         -2.8215      2.82151e-12    0
I(VDD)      A         -0.00876467  0              0
```

> **Pregunta:** inserta dos *pilas virtuales* de 0 voltios en el resto de ramas del circuito (`Vdummy1` en la rama donde está `R5` y `Vdummy2` en la rama donde está `R3` y `R4`) para que Ahkab nos imprima también la corriente en las mismas. Es muy parecido al tercer circuito que tienes que resolver, donde `V1`, `V2` y `V3` tienen cero voltios. Estas *pilas nulas* son, a todos los efectos, *simples cables*. Una vez que ya tienes las corrientes en todas las ramas, comprueba que se cumple la Ley de Kirchhoff para las corrientes:

$$I_{\text{entrante}} = \sum_{i}^{N} I_{\text{salientes}}$$

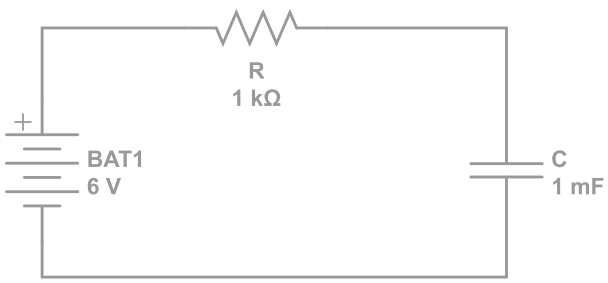
Repite lo mismo para los otros dos circuitos. Realiza además los cálculos con Sympy (recalcula los mismos voltajes que devuelve Ahkab a partir de la corriente que sí te devuelve la simulación) y cuidando de no olvidar las unidades. Recuerda que el objeto `resultados`

alberga toda la información que necesitas de manera indexada. Ya han aparecido un ejemplo más arriba. Es decir: no *copies* los números *a mano*, trabaja de manera informáticamente elegante (usando la variable `resultados`).

Circuitos en DC que evolucionan con el tiempo

Carga de un condensador

Vamos a ver qué le pasa a un circuito de corriente continua cuando tiene un condensador en serie.



Al igual que antes, primero guardamos el circuito en un netlist externo:

[21]
```
%%writefile "condensador en continua.ckt"
* Carga condensador
v1 0 1 type=vdc vdc=6
r1 1 2 1k
c1 2 0 1m ic=0
.op
.tran tstep=0.1 tstop=8 uic=0
.end
```

```
Overwriting condensador en continua.ckt
```

> **Pregunta:** ¿qué significa el parámetro `ic=0`? ¿qué perseguimos con un análisis de tipo `.tran`?

Leamos el circuito:

[22]
```
circuito_y_análisis =
ahkab.netlist_parser.parse_circuit("condensador en continua.ckt")
```

Separamos el netlist de los análisis y asignamos un fichero de almacenamiento de datos (`outfile`):

[23]
```
circuito = circuito_y_análisis[0]
análisis_en_netlist = circuito_y_análisis[1]
lista_de_análisis = ahkab.netlist_parser.parse_analysis(circuito,
análisis_en_netlist)
lista_de_análisis[1]['outfile'] = "simulación tran.tsv"
```

Ejecutamos la simulación:

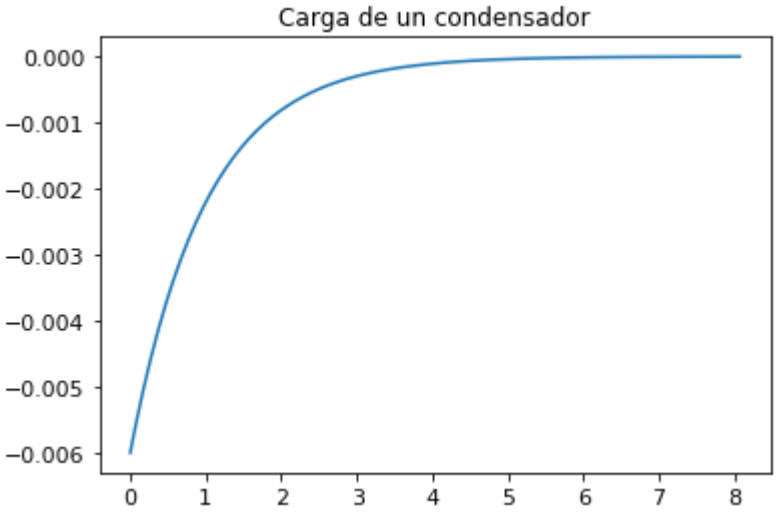
[24]
```
resultados = ahkab.run(circuito, lista_de_análisis)
print(resultados['op'])
#print(resultados['tran'].keys())
```

```
Starting op analysis:
Calculating guess: skipped. (linear circuit)
Solving...  -\ done.
Solving...  -\ done.
Difference check within margins.
(Voltage: er=0.001, ea=1e-06, Current: er=0.001, ea=1e-09)
Starting transient analysis:
Selected method: TRAP
Solving...
-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/
-\|/-\|/-\|/-\|/-\|/-done.
Average time step: 0.0869565
OP simulation results for '* carga condensador'(netlist condensador en
continua.ckt).
Run on 2019-12-05 19:08:20, data file None.
Variable    Units      Value    Error    %
----------  -------  -------  -------  ---
V1          V             -6    6e-12    0
V2          V             -6    6e-12    0
I(V1)       A              0    0        0
```

Dibujamos la gráfica de carga del condensador con el tiempo, centrándonos en la intensidad que circula por la pila.

```
[25]  figura = plt.figure()
      plt.title("Carga de un condensador")
      plt.plot(resultados['tran']['T'], resultados['tran']['I(V1)'],
      label="Una etiqueta")
```

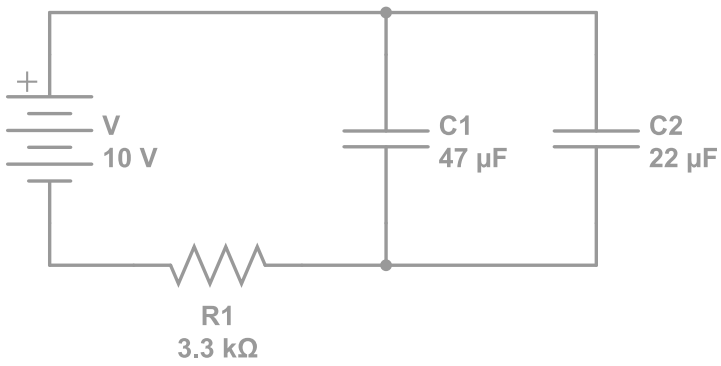
```
[<matplotlib.lines.Line2D at 0x1294f3ac8>]
```



> **Pregunta:** Etiqueta los ejes convenientemente y comenta la gráfica. Dibuja otra gráfica con el voltaje en el borne `V1`. ¿Por qué son *opuestas*? ¿Qué le ocurre al voltaje a medida que evoluciona el circuito en el tiempo? Dibuja las gráficas en un formato estándar de representación vectorial (SVG, por ejemplo). Algo de ayuda aquí. ¿Qué valores devuelve el análisis de tipo `.op`? Justifícalo.

Carrera de condensadores

Ahora tenemos un circuito con dos condensadores en paralelo:



> **Pregunta:** Crea el netlist de este circuito e identifica qué condensador se satura primero. Dibuja la evolución de la intensidad en ambas ramas de manera simultánea. Aquí tienes un ejemplo de cómo se hace esto en Matplotlib. Recuerda que para que Ahkab nos devuelva la corriente en una rama, debe de estar presente una pila. Si es necesario, inserta pilas virtuales de valor nulo (cero voltios), tal y como hemos comentado antes. Grafica también los voltajes (en otra gráfica, pero que aparezcan juntos).