sepy Documentation

Release 0.1

pysg

Aug 14, 2018

Contents

Sepy (Supercritical Extraction with Python) es una biblioteca open source orientada a cálculos de extracción con fluidos supercriticos.

CHAPTER 1

Preprocesamiento de los datos

```
[['Boldo_0W', 'Boldo_160W', 'Boldo_200W', 'Boldo_280W', 'Boldo_400W'], ['Brasil_0W',
↪'Brasil_200W', 'Brasil_280W', 'Brasil_400W'], ['Uruguay_0W', 'Uruguay_200W',
↪'Uruguay_280W', 'Uruguay_400W', 'Uruguay_600W'], ['Romero_0W', 'Romero_200W',
↪'Romero_280W', 'Romero_400W'], ['Congorosa_0W', 'Congorosa_200W', 'Congorosa_280W',
↪'Congorosa_400W']]
[[   minutos  rendimiento
0    0.000       0.0000
1    6.683       0.9460
2    6.953       1.1193
3   17.053       2.3520
4   25.069       2.8202
5   34.586       3.2030
6   41.586       3.3810
7   52.353       3.5706
8   69.019       3.7933
9   84.019       3.6280,    minutos  rendimiento
0    0.000        0.000
1    7.250        1.127
2   20.117        2.557
3   25.200        2.835
4   35.917        3.113
5   43.033        3.272
6   53.333        3.406
7   72.950        3.591
8   88.100        3.686,    minutos  rendimiento
0    0.000        0.000
1    6.850        1.095
2   17.250        2.350
3   25.333        2.872
4   34.750        3.228
5   41.400        3.383
6   58.317        3.555,    minutos  rendimiento
0    0.000        0.000
1    7.383        1.093
```

(continues on next page)

(continued from previous page)

```
2   18.333         2.339
3   25.333         2.861
4   37.750         3.157
5   43.883         3.301
6   55.750         3.700
7   68.150         3.700
8   82.750         3.821,    minutos  rendimiento
0    0.000         0.000
1    6.833         1.013
2   18.250         2.206
3   25.483         2.656
4   34.900         2.991
5   41.533         3.433
6   53.183         3.650
7   68.150         3.829
8   82.750         3.934], [   minutos  rendimiento
0     0.00        0.0000
1     3.00        0.0508
2    17.58        0.4874
3    25.00        0.7331
4    35.23        1.0231
5    40.65        1.1210
6    50.37        1.1710
7    70.28        1.2448
8    96.00        1.3626,    minutos  rendimiento
0     0.00         0.000
1     5.00         0.100
2    16.45         0.761
3    25.27         1.073
4    35.53         1.317
5    42.28         1.438
6    55.08         1.532
7    72.85         1.643
8    87.20         1.766,     minutos  rendimiento
0     0.000         0.000
1     6.733         0.270
2    16.450         0.761
3    17.400         0.846
4    25.270         1.073
5    26.000         1.126
6    35.530         1.317
7    37.067         1.365
8    42.280         1.438
9    45.300         1.522
10   55.080         1.532
11   58.750         1.697
12   77.000         1.760
13   87.200         1.766,    minutos  rendimiento
0    0.000         0.000
1    6.900         0.174
2   17.367         0.622
3   26.050         0.797
4   38.500         1.006
5   45.900         1.062
6   58.683         1.184
7   77.517         1.354
8   90.183         1.493], [    minutos  rendimiento
```

(continues on next page)

(continued from previous page)

```
0     0.000        0.000
1     6.667        0.200
2     7.550        0.283
3    17.117        0.750
4    25.583        0.977
5    35.000        1.146
6    41.667        1.231
7    52.450        1.335
8    59.000        1.365
9    79.333        1.450
10   96.733        1.344,    minutos  rendimiento
0     0.000       0.000
1     6.967       0.095
2    17.750       0.234
3    26.150       0.298
4    36.667       0.345
5    44.667       0.384
6    53.083       0.416
7    74.867       0.451
8    84.917       0.506,    minutos  rendimiento
0     0.000       0.000
1     7.417       0.078
2    17.750       0.226
3    25.717       0.294
4    37.033       0.365
5    44.567       0.395
6    55.667       0.426
7    73.250       0.480
8    86.667       0.540,    minutos  rendimiento
0     0.000       0.000
1     7.583       0.074
2    19.417       0.221
3    28.167       0.357
4    37.917       0.432
5    44.200       0.462
6    54.883       0.498
7    70.233       0.556
8    81.850       0.579,    minutos  rendimiento
0     0.000       0.000
1     7.000       0.092
2    17.500       0.221
3    25.100       0.381
4    38.450       0.469
5    46.450       0.520
6    57.167       0.564
7    70.100       0.618], [   minutos  rendimiento
0     0.000       0.000
1     7.283       1.031
2    17.333       2.284
3    24.750       2.790
4    34.117       3.215
5    40.817       3.379
6    56.705       3.621
7    73.972       3.767
8    84.972       3.833,    minutos  rendimiento
0     0.000       0.000
1     6.683       1.456
```

(continues on next page)

(continued from previous page)

```
2   15.767        2.491
3   23.083        2.916
4   32.150        3.300
5   38.667        3.350
6   48.250        3.438
7   64.533        3.553
8   79.917        3.719,    minutos  rendimiento
0    0.000        0.000
1    7.333        1.399
2   21.083        2.378
3   28.483        2.976
4   38.317        3.619
5   45.117        3.750
6   55.650        3.908
7   70.867        4.124
8   84.867        4.292,    minutos  rendimiento
0    0.000        0.000
1    7.400        1.461
2   17.467        2.412
3   25.800        3.128
4   35.200        3.239
5   45.800        3.254], [   minutos  rendimiento
0     0.00       0.0000
1     5.00       0.0080
2     6.83       0.0250
3    16.27       0.2689
4    24.33       0.3520
5    33.80       0.4290
6    39.45       0.4292
7    50.47       0.4336
8    64.15       0.5580
9    79.05       0.6090,     minutos  rendimiento
0     0.000        0.000
1     5.000        0.119
2     6.917        0.154
3     7.480        0.230
4    16.983        0.634
5    18.120        0.638
6    25.833        0.879
7    27.030        0.788
8    33.983        0.948
9    37.330        0.908
10   44.580        0.957
11   49.783        1.012
12   51.067        1.092
13   55.700        1.023
14   69.667        1.155
15   74.830        1.182
16   87.100        1.317
17   91.233        1.281,      minutos  rendimiento
0   0.000000        0.000
1   6.470000        0.140
2  15.720000        0.498
3  22.580000        0.638
4  31.870000        0.712
5  38.830000        0.721
6  49.220000        0.721
```

(continues on next page)

(continued from previous page)

```
7  65.000000        0.756
8  79.816667        0.884,    minutos  rendimiento
0      0.00        0.000
1      7.47        0.089
2     18.10        0.319
3     26.27        0.559
4     35.83        0.659
5     42.25        0.800
6     52.00        0.803
7     68.73        0.864
8     81.43        1.042]]
```

```
Index(['minutos', 'rendimiento', 'minutos', 'rendimiento', 'minutos',
       'rendimiento', 'minutos', 'rendimiento', 'minutos', 'rendimiento'],
      dtype='object')
```

CHAPTER 2

Datos operación Yerba Romero

```
[   0.  200.  280.  400.]
tr =  [ 7.89494813  7.34917765  7.84414133  8.27932474]
```

```
Yerba Romero - 400.0 W
X0 =  [ 0.03833  0.03719  0.04292  0.03254] <class 'numpy.ndarray'>
xo = 0.03253999999999999, gamma = 2.1692508973607825, yr = 0.0074, TAO = 5.
↪531852105882878
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-02   7.40000000e+00   1.74670000e+01   2.58000000e+01
   3.52000000e+01   4.58000000e+01] <class 'numpy.ndarray'>
[ 0.01        0.8937927   2.10971312  3.11619616  4.25155446  5.53185211] <class
↪'numpy.ndarray'>
[ 0.0001   0.01461  0.02412  0.03128  0.03239  0.03254]
```

CHAPTER 3

Datos operación Yerba uruguay

```
[   0.  200.  280.  400.  600.]
tr =  [ 6.90315722  7.06313825  7.19084665  6.90573444  6.96705883]
```

```
Yerba Uruguay - 600.0 W
X0 =  [ 0.0145   0.00506  0.0054   0.00579  0.00618] <class 'numpy.ndarray'>
xo = 0.00618, gamma = 1.8798398443346804, yr = 0.0013, TAO = 10.061634577110048
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-02   1.00472813e+00   2.51182033e+00   3.60266802e+00
   5.51882810e+00   6.66708882e+00   8.20532759e+00   1.00616346e+01] <class 'numpy.
↪ndarray'>
[ 0.0001   0.00092  0.00221  0.00381  0.00469  0.0052   0.00564  0.00618]
```

CHAPTER 4

fin yu

CHAPTER 5

Datos de operación yerba bolbo

```
tr =  [ 6.0855294   6.4236725   7.25440914  6.17722886  6.18946974]
[ 0.28606652  0.27100794  0.23997354  0.28181994  0.28126258]
```

```
['bolbo' 'bolbo' 'bolbo' 'bolbo' 'bolbo' 'brasil' 'brasil' 'brasil'
 'brasil' 'congorosa' 'congorosa' 'congorosa' 'congorosa']
```

```
{'Densidad gr/cm3': {'bolbo': 1.25, 'brasil': 1.34, 'congorosa': 1.31},
 'Porosidad': {'bolbo': 0.7208, 'brasil': 0.7179, 'congorosa': 0.7137}}
```

```
280.0
Yerba Bolbo - 400.0 W
X0 =  [ 0.037933  0.03686   0.03555   0.03821   0.03934 ] <class 'numpy.ndarray'>
xo = 0.03934, gamma = 1.7408662464183382, yr = 0.0059, TAO = 13.369481303875684
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  3.00000000e-03   1.10397179e+00   2.94855630e+00   4.11715398e+00
   5.63860903e+00   6.71026788e+00   8.59249697e+00   1.10106363e+01
   1.33694813e+01] <class 'numpy.ndarray'>
[ 0.001    0.01013  0.02206  0.02656  0.02991  0.03433  0.0365   0.03829
  0.03934]
```

CHAPTER 6

Datos de operación yerba brasil

```
[   0.  200.  280.  400.]
tr =  [ 6.27556792  5.90557503  6.16106767  6.39774269]
```

```
Yerba Brasil - 400.0 W
X0 =  [ 0.013626  0.01766   0.01766   0.01493 ] <class 'numpy.ndarray'>
xo = 0.01493, gamma = 1.6007817435333078, yr = 0.0059, TAO = 14.096065489128808
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-04   1.07850539e+00   2.71455118e+00   4.07174862e+00
   6.01774748e+00   7.17440544e+00   9.17245391e+00   1.21163047e+01
   1.40960655e+01] <class 'numpy.ndarray'>
[ 0.0001   0.00174  0.00622  0.00797  0.01006  0.01062  0.01184  0.01354
  0.01493]
```

CHAPTER 7

Datos de operación yerba congorosa

```
tr =  [ 5.42987582  6.11480591  5.44828666  5.78986569]
```

```
Yerba Congorosa - 400.0 W
X0 =  [ 0.00609  0.01317  0.00884  0.01042] <class 'numpy.ndarray'>
xo = 0.01042, gamma = 1.6039805840774504, yr = 0.0059, TAO = 14.064229522040602
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-02   1.29018537e+00   3.12615196e+00   4.53723824e+00
   6.18839916e+00   7.29723317e+00   8.98121006e+00   1.18707417e+01
   1.40642295e+01] <class 'numpy.ndarray'>
[ 0.0001   0.00089  0.00319  0.00559  0.00659  0.008    0.00803  0.00864
  0.01042]
```

CHAPTER 8

Modelo Lack

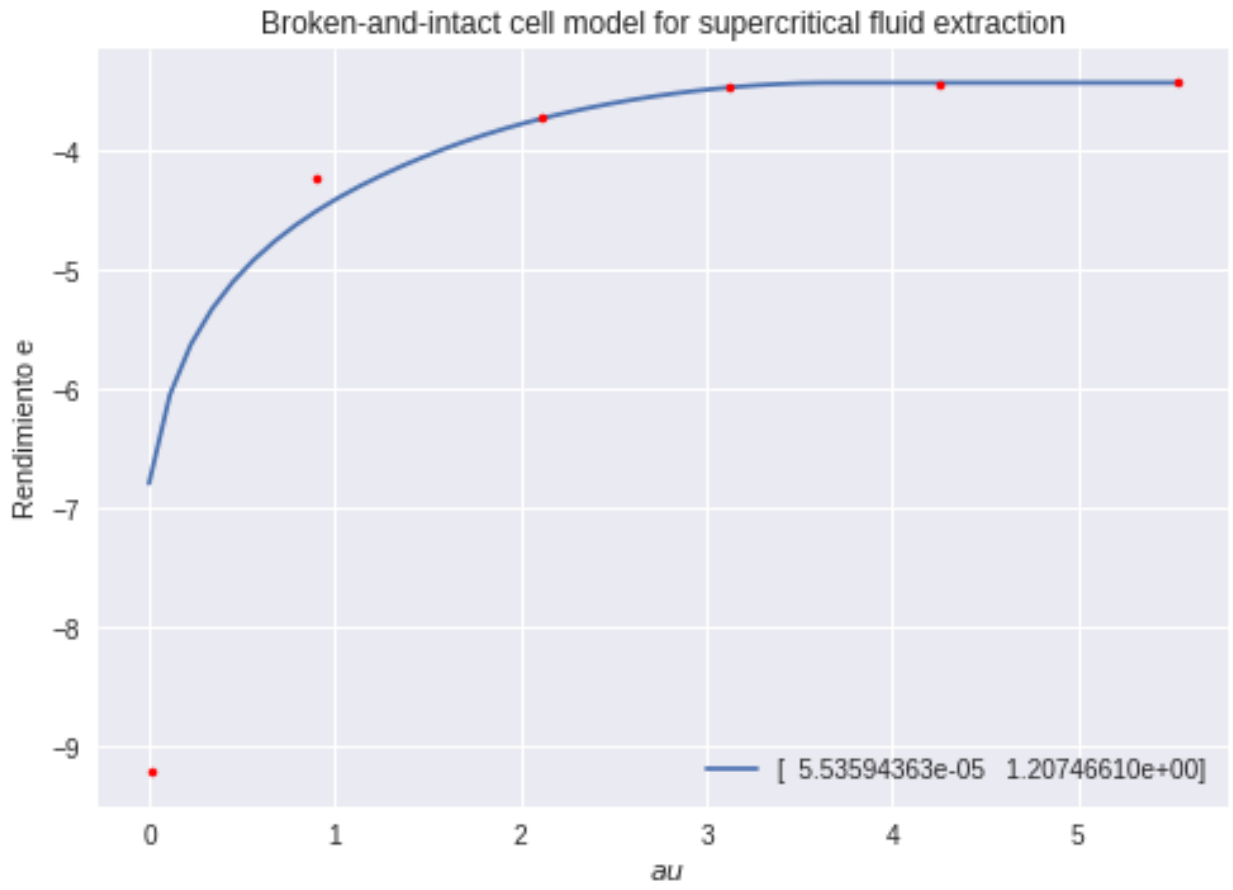
CHAPTER 9

Ajuste de los parámetros del modelo 1: Modelo Lack

```
active_mask: array([0, 0])
       cost: 0.018300978793627454
        fun: array([-0.11809526,  0.37330728,  0.02808377,  0.00718511, -0.
↪00462037,  0.        ])
       grad: array([  8.33403474e-04,  -1.54227422e-08])
        jac: array([[  0.00000000e+00,  -6.35444581e-09],
      [  0.00000000e+00,  -6.35444434e-09],
      [  2.06735361e-02,  -3.11057628e-01],
      [  3.17151307e-02,  -1.76298562e-01],
      [  0.00000000e+00,   0.00000000e+00],
      [  0.00000000e+00,   0.00000000e+00]])
    message: 'Both ftol and xtol termination conditions are satisfied.'
       nfev: 22
       njev: 12
 optimality: 6.7744917811128669e-08
     status: 4
    success: True
          x: array([  5.53594363e-05,   1.20746610e+00])
```

```
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:5:␣
↪RuntimeWarning: invalid value encountered in log
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:4:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:5:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:14:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:17:␣
↪RuntimeWarning: invalid value encountered in log
```

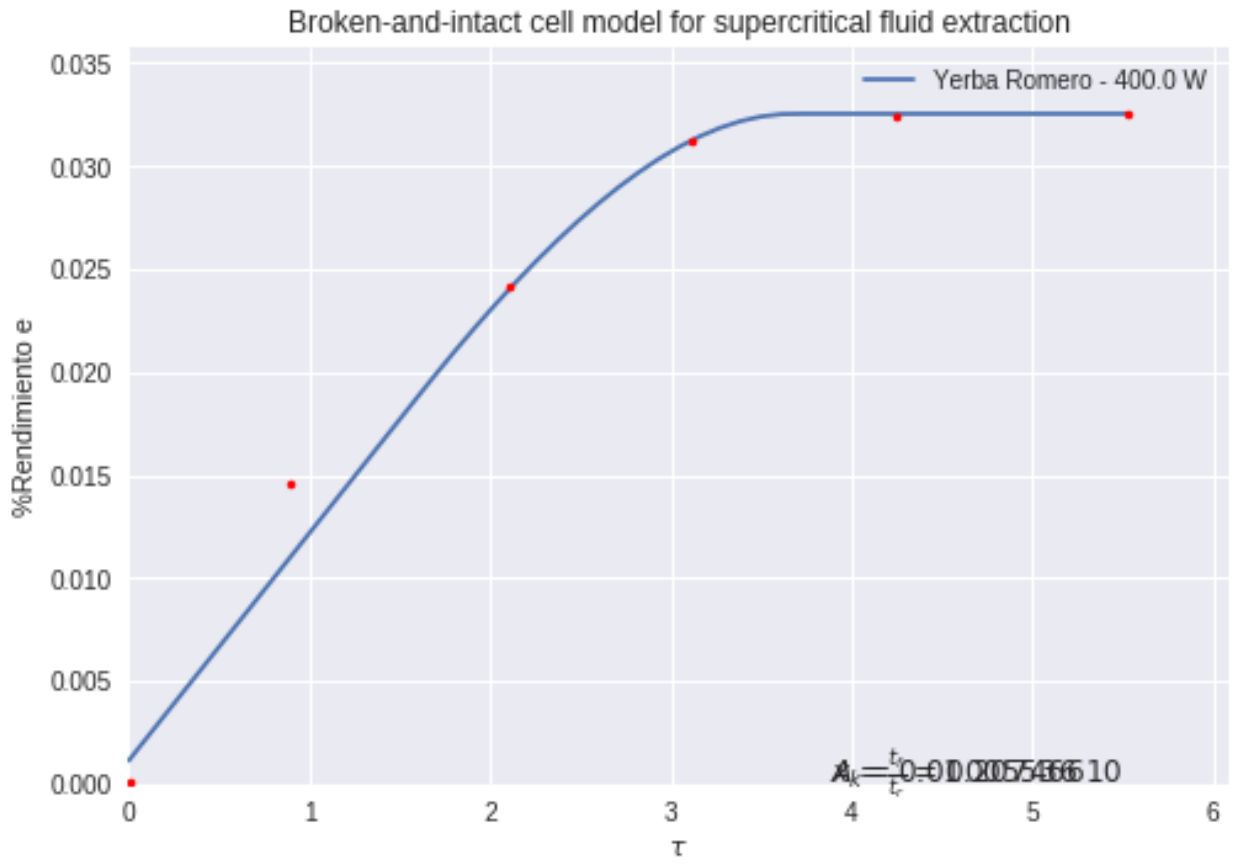
```
[<matplotlib.lines.Line2D at 0x7f203f10ff98>]
```



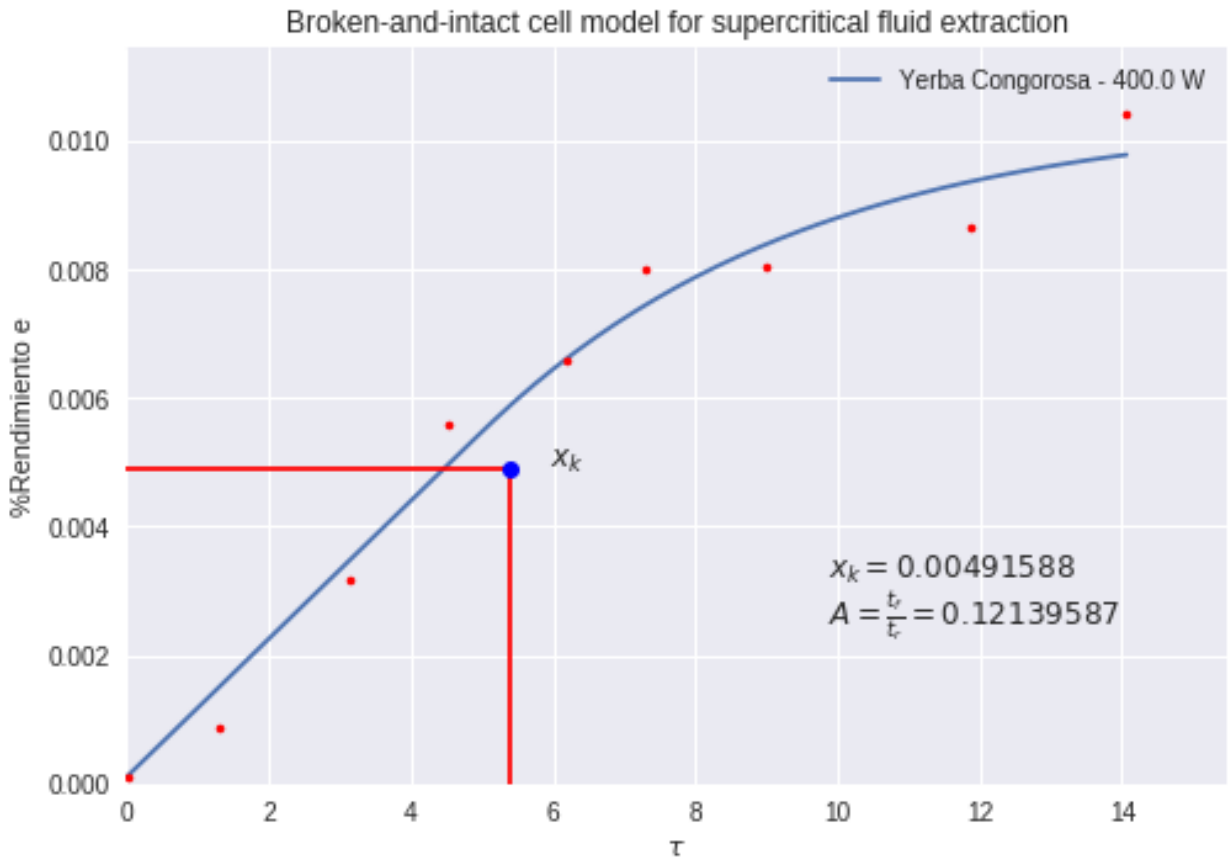

```
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:4:␣
↪RuntimeWarning: overflow encountered in exp
```

```
(1.6759521390448247, 3.7030513340813194, inf)
```

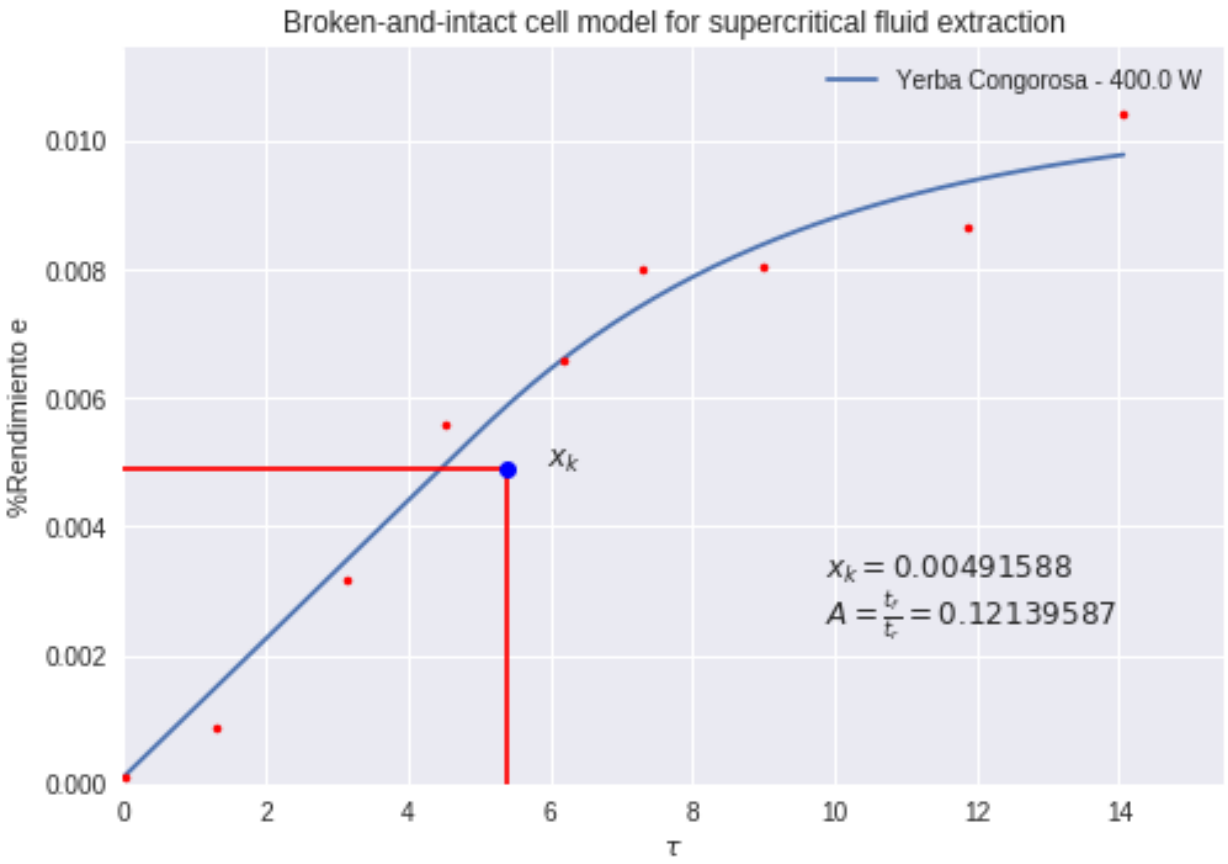
```
[<matplotlib.lines.Line2D at 0x7f203f0cd240>]
```




```
NOMBRE: Congorosa
```




```
NOMBRE: Congorosa
```


Broken-and-intact cell model for supercritical fluid extraction
Yerba Congorosa - 400.0 W
%Rendimiento e
0.010
0.008
0.006
0.004
0.002
0.000
0
2
4
6
8
10
12
14
τ
x_k
$x_k = 0.00491588$
$A = \frac{t_r}{t_r} = 0.12139587$

CHAPTER 10

Modelo Sovova

```
nan nan
```

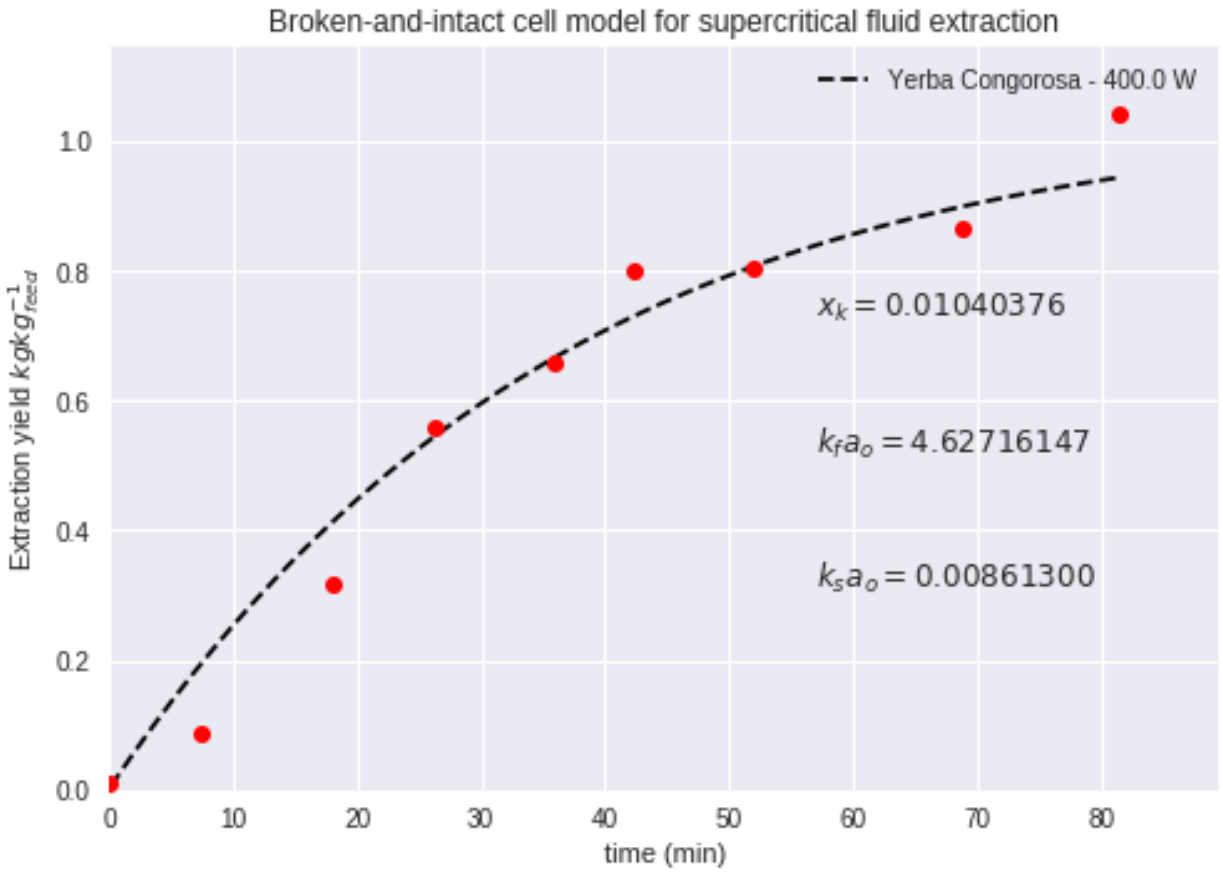
```
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:9:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:10:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:28:␣
↪RuntimeWarning: invalid value encountered in log
```

```
0.032539999999999993
```

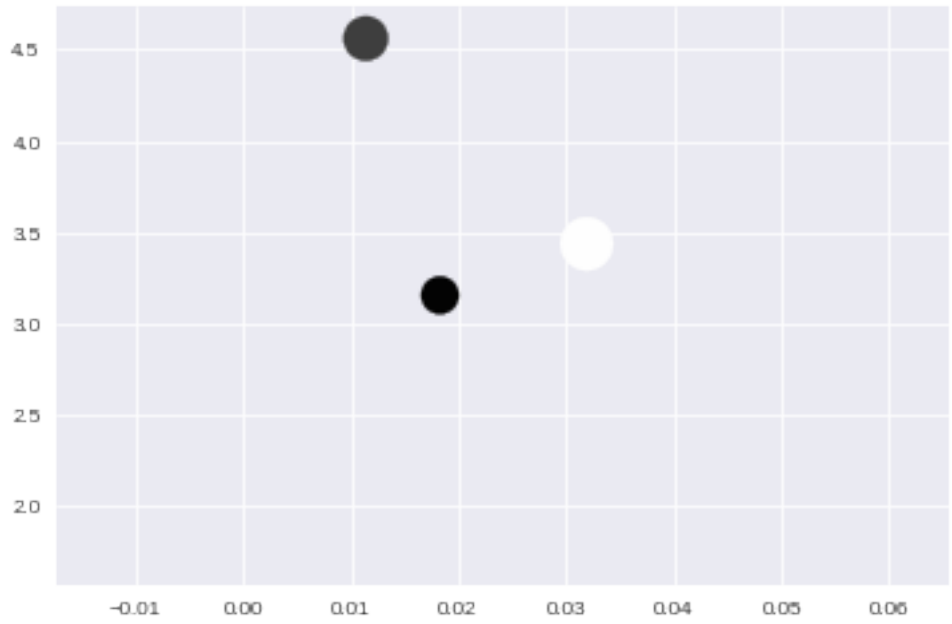
```
active_mask: array([0, 0, 0])
       cost: 0.067573820707988189
        fun: array([ 1.81182816, -0.78683877, -0.26158694,  0.02518217, -0.
↪00921332,
       0.09234707, -0.00521931, -0.0395414 ,  0.09887852])
       grad: array([  4.91259316e-02,  -2.61716140e-06,  -1.03208360e-05])
        jac: array([[  1.64962614e-05,  -7.38580870e-12,  -2.72356004e-09],
      [  5.71402153e-06,  -1.02692647e-13,  -1.42693543e-06],
      [  2.03829982e-06,  -3.70461323e-14,  -1.24830113e-06],
      [  7.58134626e+01,  -1.35986181e-06,  -6.79077966e+01],
      [  5.16927813e+01,  -9.15747753e-07,  -6.36407023e+01],
      [  8.24606189e+00,  -1.49376459e-07,  -1.20250951e+01],
      [  2.71129286e+01,  -5.00329665e-07,  -4.89541775e+01],
      [  1.19393497e+01,  -2.25101586e-07,  -2.87292828e+01],
      [  7.43796280e-01,  -1.26455894e-08,  -2.13102359e+00]])
    message: '`xtol` termination condition is satisfied.'
       nfev: 31
       njev: 16
 optimality: 0.000515952907784016
     status: 3
    success: True
```

```
        x: array([ 0.01040376,  4.62716147,  0.008613  ])
```

```
[<matplotlib.lines.Line2D at 0x7effbc368080>]
```




```
  File "<ipython-input-337-5e85ffa89130>", line 1
    x: array([ 0.03829315,  1.70819818,  0.01795442]) #0
     ^
SyntaxError: invalid syntax
```




```
array([ 0.00017276,  0.00280808,  0.00535308,  0.0078023 ,  0.01015068,
        0.01239369,  0.01452747,  0.01654892,  0.01845584,  0.02024697,
        0.02192205,  0.02348181,  0.02492799,  0.02626321,  0.02749096,
        0.02861545,  0.02964145,  0.03057426,  0.03141945,  0.03218284,
        0.0328703 ,  0.0334877 ,  0.03404079,  0.03453511,  0.03497599,
        0.03536845,  0.03571721,  0.03602665,  0.03630083,  0.03654345,
        0.03675791,  0.03694729,  0.03711438,  0.03726167,  0.03739143,
        0.03750567,  0.0376062 ,  0.03769461,  0.03777233,  0.03784064,
        0.03790064,  0.03795334,  0.03799961,  0.03804023,  0.03807588,
        0.03810716,  0.0381346 ,  0.03815868,  0.0381798 ,  0.03819832])
```

```
(0.22974968576541693,
 24.199480533020207,
 5.9097726601403524,
 23.083006310073888,
 0.023157530596497361,
 3.292814746661628)
```

```
Help on function plot in module matplotlib.pyplot:

plot(args, **kwargs)
    Plot y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], data=None, **kwargs)
        plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, y.

    The optional parameter fmt is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string
    notation described in the Notes section below.
```

```
>>> plot(x, y)        # plot x and y using default line style and color
>>> plot(x, y, 'bo')  # plot x and y using blue circle markers
>>> plot(y)           # plot y using x as index array 0..N-1
>>> plot(y, 'r+')     # ditto, but with red plusses
```

You can use *.Line2D* properties as keyword arguments for more control on the appearance. Line properties and *fmt* can be mixed. The following two calls yield identical results:

```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
        linewidth=2, markersize=12)
```

When conflicting with *fmt*, keyword arguments take precedence.

Plotting labelled data

There's a convenient way for plotting objects with labelled data (i.e. data that can be accessed by index obj['y']). Instead of giving the data in *x* and *y*, you can provide the object in the *data* parameter and just give the labels for *x* and *y*::

```
>>> plot('xlabel', 'ylabel', data=obj)
```

All indexable objects are supported. This could e.g. be a *dict*, a *pandas.DataFame* or a structured numpy array.

Plotting multiple sets of data

There are various ways to plot multiple sets of data.

- The most straight forward way is just to call *plot* multiple times. Example:

  ```
  >>> plot(x1, y1, 'bo')
  >>> plot(x2, y2, 'go')
  ```

- Alternatively, if your data is already a 2d array, you can pass it directly to *x*, *y*. A separate data set will be drawn for every column.

  Example: an array a where the first column represents the *x* values and the other columns are the *y* columns::

  ```
  >>> plot(a[0], a[1:])
  ```

- The third way is to specify multiple sets of *[x]*, *y*, *[fmt]* groups::

  ```
  >>> plot(x1, y1, 'g^', x2, y2, 'g-')
  ```

  In this case, any additional keyword argument applies to all datasets. Also this syntax cannot be combined with the *data*

```
  parameter.

By default, each line is assigned a different style specified by a
'style cycle'. The fmt and line property parameters are only
necessary if you want explicit deviations from these defaults.
Alternatively, you can also change the style cycle using the
'axes.prop_cycle' rcParam.

Parameters
----------
x, y : array-like or scalar
    The horizontal / vertical coordinates of the data points.
    x values are optional. If not given, they default to
    [0, ..., N-1].

    Commonly, these parameters are arrays of length N. However,
    scalars are supported as well (equivalent to an array with
    constant value).

    The parameters can also be 2-dimensional. Then, the columns
    represent separate data sets.

fmt : str, optional
    A format string, e.g. 'ro' for red circles. See the Notes
    section for a full description of the format strings.

    Format strings are just an abbreviation for quickly setting
    basic line properties. All of these and more can also be
    controlled by keyword arguments.

data : indexable object, optional
    An object with labelled data. If given, provide the label names to
    plot in x and y.

    .. note::
        Technically there's a slight ambiguity in calls where the
        second label is a valid fmt. plot('n', 'o', data=obj)
        could be plt(x, y) or plt(y, fmt). In such cases,
        the former interpretation is chosen, but a warning is issued.
        You may suppress the warning by adding an empty format string
        plot('n', 'o', '', data=obj).


Other Parameters
----------------
scalex, scaley : bool, optional, default: True
    These parameters determined if the view limits are adapted to
    the data limits. The values are passed on to autoscale_view.

kwargs : `.Line2D` properties, optional
    *kwargs* are used to specify properties like a line label (for
    auto legends), linewidth, antialiasing, marker face color.
    Example::
```

```
        >>> plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2)
        >>> plot([1,2,3], [1,4,9], 'rs',  label='line 2')

        If you make multiple lines with one plot command, the kwargs
        apply to all those lines.

        Here is a list of available `.Line2D` properties:

          agg_filter: a filter function, which takes a (m, n, 3) float array␣
↪and a dpi value, and returns a (m, n, 3) array
      alpha: float (0.0 transparent through 1.0 opaque)
      animated: bool
      antialiased or aa: bool
      clip_box: a `.Bbox` instance
      clip_on: bool
      clip_path: [(`~matplotlib.path.Path`, `.Transform`) | `.Patch` | None]
      color or c: any matplotlib color
      contains: a callable function
      dash_capstyle: ['butt' | 'round' | 'projecting']
      dash_joinstyle: ['miter' | 'round' | 'bevel']
      dashes: sequence of on/off ink in points
      drawstyle: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-
↪post']
      figure: a `.Figure` instance
      fillstyle: ['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']
      gid: an id string
      label: object
      linestyle or ls: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-
↪off-dash-seq) | ``'-'`` | ``'--'`` | ``'-.'`` | ``':'`` | ``'None'`` | ``'␣
↪'`` | ``''``]
      linewidth or lw: float value in points
      marker: :mod:`A valid marker style <matplotlib.markers>`
      markeredgecolor or mec: any matplotlib color
      markeredgewidth or mew: float value in points
      markerfacecolor or mfc: any matplotlib color
      markerfacecoloralt or mfcalt: any matplotlib color
      markersize or ms: float
      markevery: [None | int | length-2 tuple of int | slice | list/array of␣
↪int | float | length-2 tuple of float]
      path_effects: `.AbstractPathEffect`
      picker: float distance in points or callable pick function ``fn(artist,␣
↪event)``
      pickradius: float distance in points
      rasterized: bool or None
      sketch_params: (scale: float, length: float, randomness: float)
      snap: bool or None
      solid_capstyle: ['butt' | 'round' |  'projecting']
      solid_joinstyle: ['miter' | 'round' | 'bevel']
      transform: a :class:`matplotlib.transforms.Transform` instance
      url: a url string
      visible: bool
      xdata: 1D array
      ydata: 1D array
      zorder: float
```

Returns
----
lines
A list of `.Line2D` objects representing the plotted data.

See Also
----
scatter : XY scatter plot with markers of varying size and/or color (
sometimes also called bubble chart).

Notes

****Format Strings**

A format string consists of a part for color, marker and line::

fmt = '[color][marker][line]'

Each of them is optional. If not provided, the value from the style
cycle is used. Exception: If line is given, but no marker,
the data will be a line without markers.

Colors

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

If the color is the only part of the format string, you can
additionally use any *matplotlib.colors* spec, e.g. full names
('green') or hex strings ('#008000').

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker

```
    'v'          triangle_down marker
    '^'          triangle_up marker
    '<'          triangle_left marker
    '>'          triangle_right marker
    '1'          tri_down marker
    '2'          tri_up marker
    '3'          tri_left marker
    '4'          tri_right marker
    's'          square marker
    'p'          pentagon marker
    '*'          star marker
    'h'          hexagon1 marker
    'H'          hexagon2 marker
    '+'          plus marker
    'x'          x marker
    'D'          diamond marker
    'd'          thin_diamond marker
    '|'          vline marker
    '_'          hline marker
    =============    ==============================

    Line Styles

    =============    ==============================
    character        description
    =============    ==============================
    '-'          solid line style
    '--'         dashed line style
    '-.'         dash-dot line style
    ':'          dotted line style
    =============    ==============================

    Example format strings::

        'b'    # blue markers with default shape
        'ro'   # red circles
        'g-'   # green solid line
        '--'   # dashed line with default color
        'k^:'  # black triangle_up markers connected by a dotted line

    .. note::
        In addition to the above described arguments, this function can take a
        data keyword argument. If such a data argument is given, the
        following arguments are replaced by data[<arg>]:

        * All arguments with the following names: 'x', 'y'.
```

```
  File "<ipython-input-104-c3b0085c4838>", line 1
    cielo, la persona de mkt es la que tiene que ofrecer un listado de servicios y␣
↪productos:
                   ^
SyntaxError: invalid syntax
```

CHAPTER 11

Preprocesamiento de los datos

```
[['Boldo_0W', 'Boldo_160W', 'Boldo_200W', 'Boldo_280W', 'Boldo_400W'], ['Brasil_0W',
↪'Brasil_200W', 'Brasil_280W', 'Brasil_400W'], ['Uruguay_0W', 'Uruguay_200W',
↪'Uruguay_280W', 'Uruguay_400W', 'Uruguay_600W'], ['Romero_0W', 'Romero_200W',
↪'Romero_280W', 'Romero_400W'], ['Congorosa_0W', 'Congorosa_200W', 'Congorosa_280W',
↪'Congorosa_400W']]
[[   minutos  rendimiento
0    0.000       0.0000
1    6.683       0.9460
2    6.953       1.1193
3   17.053       2.3520
4   25.069       2.8202
5   34.586       3.2030
6   41.586       3.3810
7   52.353       3.5706
8   69.019       3.7933
9   84.019       3.6280,    minutos  rendimiento
0    0.000        0.000
1    7.250        1.127
2   20.117        2.557
3   25.200        2.835
4   35.917        3.113
5   43.033        3.272
6   53.333        3.406
7   72.950        3.591
8   88.100        3.686,    minutos  rendimiento
0    0.000        0.000
1    6.850        1.095
2   17.250        2.350
3   25.333        2.872
4   34.750        3.228
5   41.400        3.383
6   58.317        3.555,    minutos  rendimiento
0    0.000        0.000
1    7.383        1.093
```

(continues on next page)

(continued from previous page)

```
2   18.333        2.339
3   25.333        2.861
4   37.750        3.157
5   43.883        3.301
6   55.750        3.700
7   68.150        3.700
8   82.750        3.821,    minutos  rendimiento
0    0.000        0.000
1    6.833        1.013
2   18.250        2.206
3   25.483        2.656
4   34.900        2.991
5   41.533        3.433
6   53.183        3.650
7   68.150        3.829
8   82.750        3.934], [   minutos  rendimiento
0     0.00       0.0000
1     3.00       0.0508
2    17.58       0.4874
3    25.00       0.7331
4    35.23       1.0231
5    40.65       1.1210
6    50.37       1.1710
7    70.28       1.2448
8    96.00       1.3626,    minutos  rendimiento
0     0.00        0.000
1     5.00        0.100
2    16.45        0.761
3    25.27        1.073
4    35.53        1.317
5    42.28        1.438
6    55.08        1.532
7    72.85        1.643
8    87.20        1.766,     minutos  rendimiento
0     0.000        0.000
1     6.733        0.270
2    16.450        0.761
3    17.400        0.846
4    25.270        1.073
5    26.000        1.126
6    35.530        1.317
7    37.067        1.365
8    42.280        1.438
9    45.300        1.522
10   55.080        1.532
11   58.750        1.697
12   77.000        1.760
13   87.200        1.766,    minutos  rendimiento
0    0.000        0.000
1    6.900        0.174
2   17.367        0.622
3   26.050        0.797
4   38.500        1.006
5   45.900        1.062
6   58.683        1.184
7   77.517        1.354
8   90.183        1.493], [    minutos  rendimiento
```

(continues on next page)

(continued from previous page)

```
0     0.000        0.000
1     6.667        0.200
2     7.550        0.283
3    17.117        0.750
4    25.583        0.977
5    35.000        1.146
6    41.667        1.231
7    52.450        1.335
8    59.000        1.365
9    79.333        1.450
10   96.733        1.344,    minutos  rendimiento
0     0.000       0.000
1     6.967       0.095
2    17.750       0.234
3    26.150       0.298
4    36.667       0.345
5    44.667       0.384
6    53.083       0.416
7    74.867       0.451
8    84.917       0.506,    minutos  rendimiento
0     0.000       0.000
1     7.417       0.078
2    17.750       0.226
3    25.717       0.294
4    37.033       0.365
5    44.567       0.395
6    55.667       0.426
7    73.250       0.480
8    86.667       0.540,    minutos  rendimiento
0     0.000       0.000
1     7.583       0.074
2    19.417       0.221
3    28.167       0.357
4    37.917       0.432
5    44.200       0.462
6    54.883       0.498
7    70.233       0.556
8    81.850       0.579,    minutos  rendimiento
0     0.000       0.000
1     7.000       0.092
2    17.500       0.221
3    25.100       0.381
4    38.450       0.469
5    46.450       0.520
6    57.167       0.564
7    70.100       0.618], [   minutos  rendimiento
0     0.000       0.000
1     7.283       1.031
2    17.333       2.284
3    24.750       2.790
4    34.117       3.215
5    40.817       3.379
6    56.705       3.621
7    73.972       3.767
8    84.972       3.833,    minutos  rendimiento
0     0.000       0.000
1     6.683       1.456
```

(continues on next page)

(continued from previous page)

```
2   15.767        2.491
3   23.083        2.916
4   32.150        3.300
5   38.667        3.350
6   48.250        3.438
7   64.533        3.553
8   79.917        3.719,    minutos  rendimiento
0    0.000        0.000
1    7.333        1.399
2   21.083        2.378
3   28.483        2.976
4   38.317        3.619
5   45.117        3.750
6   55.650        3.908
7   70.867        4.124
8   84.867        4.292,    minutos  rendimiento
0    0.000        0.000
1    7.400        1.461
2   17.467        2.412
3   25.800        3.128
4   35.200        3.239
5   45.800        3.254], [   minutos  rendimiento
0     0.00       0.0000
1     5.00       0.0080
2     6.83       0.0250
3    16.27       0.2689
4    24.33       0.3520
5    33.80       0.4290
6    39.45       0.4292
7    50.47       0.4336
8    64.15       0.5580
9    79.05       0.6090,     minutos  rendimiento
0     0.000        0.000
1     5.000        0.119
2     6.917        0.154
3     7.480        0.230
4    16.983        0.634
5    18.120        0.638
6    25.833        0.879
7    27.030        0.788
8    33.983        0.948
9    37.330        0.908
10   44.580        0.957
11   49.783        1.012
12   51.067        1.092
13   55.700        1.023
14   69.667        1.155
15   74.830        1.182
16   87.100        1.317
17   91.233        1.281,      minutos  rendimiento
0   0.000000        0.000
1   6.470000        0.140
2  15.720000        0.498
3  22.580000        0.638
4  31.870000        0.712
5  38.830000        0.721
6  49.220000        0.721
```

(continues on next page)

(continued from previous page)

```
7  65.000000        0.756
8  79.816667        0.884,    minutos  rendimiento
0      0.00        0.000
1      7.47        0.089
2     18.10        0.319
3     26.27        0.559
4     35.83        0.659
5     42.25        0.800
6     52.00        0.803
7     68.73        0.864
8     81.43        1.042]]
```

```
Index(['minutos', 'rendimiento', 'minutos', 'rendimiento', 'minutos',
       'rendimiento', 'minutos', 'rendimiento', 'minutos', 'rendimiento'],
      dtype='object')
```

CHAPTER 12

Datos operación Yerba Romero

```
[   0.  200.  280.  400.]
tr =  [ 7.89494813  7.34917765  7.84414133  8.27932474]
```

```
Yerba Romero - 400.0 W
X0 =  [ 0.03833  0.03719  0.04292  0.03254] <class 'numpy.ndarray'>
xo = 0.03253999999999999, gamma = 2.1692508973607825, yr = 0.0074, TAO = 5.
↪531852105882878
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-02   7.40000000e+00   1.74670000e+01   2.58000000e+01
   3.52000000e+01   4.58000000e+01] <class 'numpy.ndarray'>
[ 0.01        0.8937927   2.10971312  3.11619616  4.25155446  5.53185211] <class
↪'numpy.ndarray'>
[ 0.0001   0.01461  0.02412  0.03128  0.03239  0.03254]
```

CHAPTER 13

Datos operación Yerba uruguay

```
[   0.  200.  280.  400.  600.]
tr =  [ 6.90315722  7.06313825  7.19084665  6.90573444  6.96705883]
```

```
Yerba Uruguay - 600.0 W
X0 =  [ 0.0145   0.00506  0.0054   0.00579  0.00618] <class 'numpy.ndarray'>
xo = 0.00618, gamma = 1.8798398443346804, yr = 0.0013, TAO = 10.061634577110048
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-02   1.00472813e+00   2.51182033e+00   3.60266802e+00
   5.51882810e+00   6.66708882e+00   8.20532759e+00   1.00616346e+01] <class 'numpy.
↪ndarray'>
[ 0.0001   0.00092  0.00221  0.00381  0.00469  0.0052   0.00564  0.00618]
```

CHAPTER 14

fin yu

CHAPTER 15

Datos de operación yerba bolbo

```
tr =  [ 6.0855294   6.4236725   7.25440914  6.17722886  6.18946974]
[ 0.28606652  0.27100794  0.23997354  0.28181994  0.28126258]
```

```
['bolbo' 'bolbo' 'bolbo' 'bolbo' 'bolbo' 'brasil' 'brasil' 'brasil'
 'brasil' 'congorosa' 'congorosa' 'congorosa' 'congorosa']
```

```
{'Densidad gr/cm3': {'bolbo': 1.25, 'brasil': 1.34, 'congorosa': 1.31},
 'Porosidad': {'bolbo': 0.7208, 'brasil': 0.7179, 'congorosa': 0.7137}}
```

```
280.0
Yerba Bolbo - 400.0 W
X0 =  [ 0.037933  0.03686   0.03555   0.03821   0.03934 ] <class 'numpy.ndarray'>
xo = 0.03934, gamma = 1.7408662464183382, yr = 0.0059, TAO = 13.369481303875684
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  3.00000000e-03   1.10397179e+00   2.94855630e+00   4.11715398e+00
   5.63860903e+00   6.71026788e+00   8.59249697e+00   1.10106363e+01
   1.33694813e+01] <class 'numpy.ndarray'>
[ 0.001    0.01013  0.02206  0.02656  0.02991  0.03433  0.0365   0.03829
  0.03934]
```

CHAPTER 16

Datos de operación yerba brasil

```
[   0.  200.  280.  400.]
tr =  [ 6.27556792  5.90557503  6.16106767  6.39774269]
```

```
Yerba Brasil - 400.0 W
X0 =  [ 0.013626  0.01766   0.01766   0.01493 ] <class 'numpy.ndarray'>
xo = 0.01493, gamma = 1.6007817435333078, yr = 0.0059, TAO = 14.096065489128808
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-04   1.07850539e+00   2.71455118e+00   4.07174862e+00
   6.01774748e+00   7.17440544e+00   9.17245391e+00   1.21163047e+01
   1.40960655e+01] <class 'numpy.ndarray'>
[ 0.0001   0.00174  0.00622  0.00797  0.01006  0.01062  0.01184  0.01354
  0.01493]
```

CHAPTER 17

Datos de operación yerba congorosa

```
tr =  [ 5.42987582  6.11480591  5.44828666  5.78986569]
```

```
Yerba Congorosa - 400.0 W
X0 =  [ 0.00609  0.01317  0.00884  0.01042] <class 'numpy.ndarray'>
xo = 0.01042, gamma = 1.6039805840774504, yr = 0.0059, TAO = 14.064229522040602
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-02   1.29018537e+00   3.12615196e+00   4.53723824e+00
   6.18839916e+00   7.29723317e+00   8.98121006e+00   1.18707417e+01
   1.40642295e+01] <class 'numpy.ndarray'>
[ 0.0001   0.00089  0.00319  0.00559  0.00659  0.008    0.00803  0.00864
  0.01042]
```

CHAPTER 18

Modelo Lack

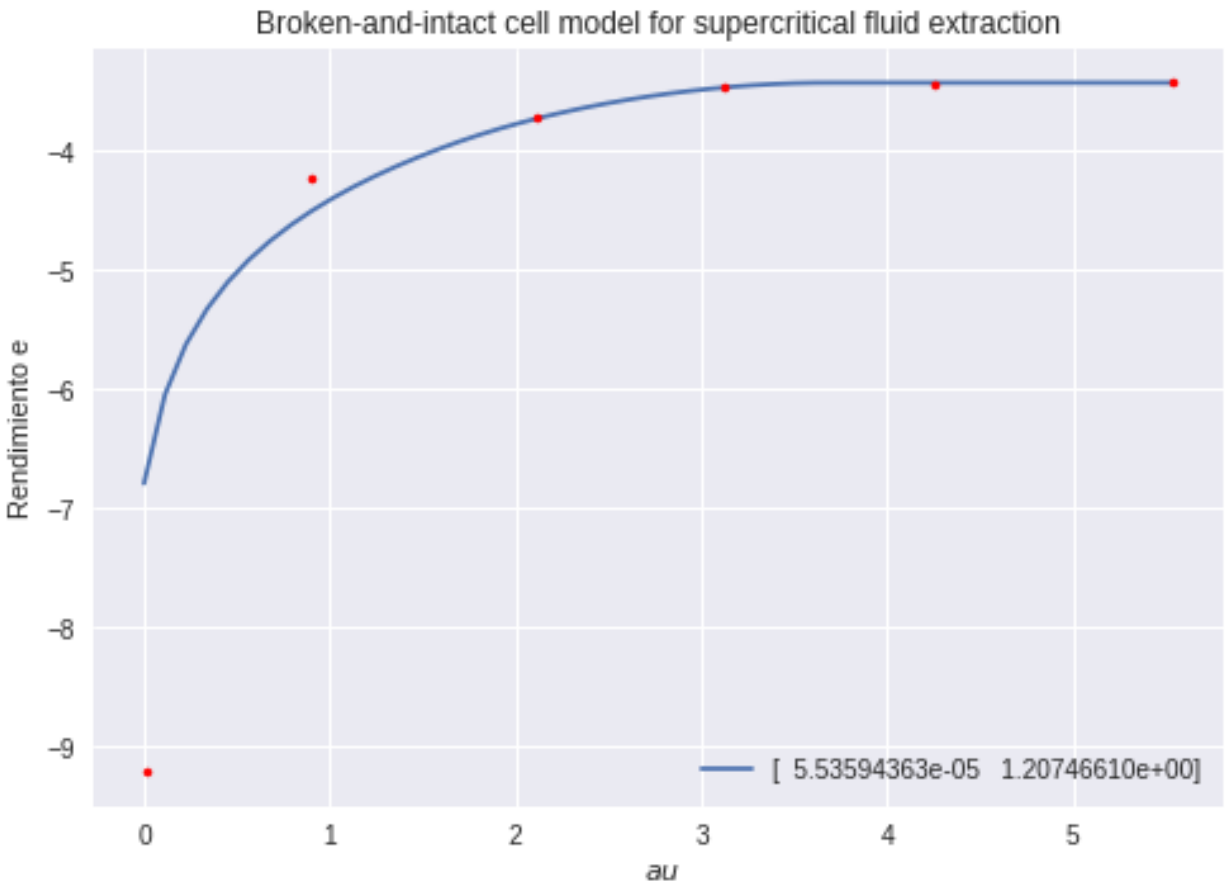
CHAPTER 19

Ajuste de los parámetros del modelo 1: Modelo Lack

```
active_mask: array([0, 0])
       cost: 0.018300978793627454
        fun: array([-0.11809526,  0.37330728,  0.02808377,  0.00718511, -0.
↪00462037,  0.        ])
       grad: array([  8.33403474e-04,  -1.54227422e-08])
        jac: array([[  0.00000000e+00,  -6.35444581e-09],
      [  0.00000000e+00,  -6.35444434e-09],
      [  2.06735361e-02,  -3.11057628e-01],
      [  3.17151307e-02,  -1.76298562e-01],
      [  0.00000000e+00,   0.00000000e+00],
      [  0.00000000e+00,   0.00000000e+00]])
    message: 'Both ftol and xtol termination conditions are satisfied.'
       nfev: 22
       njev: 12
 optimality: 6.7744917811128669e-08
     status: 4
    success: True
          x: array([  5.53594363e-05,   1.20746610e+00])
```

```
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:5:␣
↪RuntimeWarning: invalid value encountered in log
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:4:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:5:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:14:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:17:␣
↪RuntimeWarning: invalid value encountered in log
```

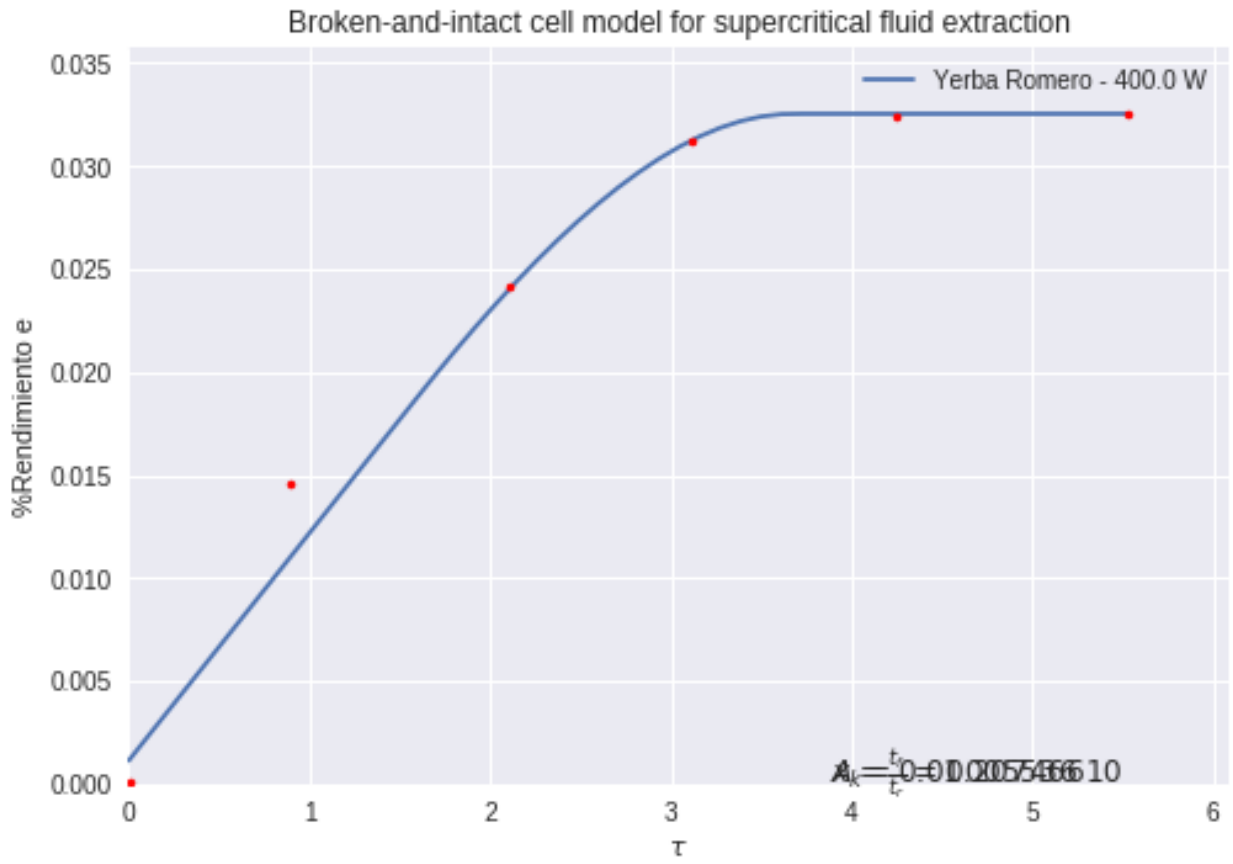
```
[<matplotlib.lines.Line2D at 0x7f203f10ff98>]
```



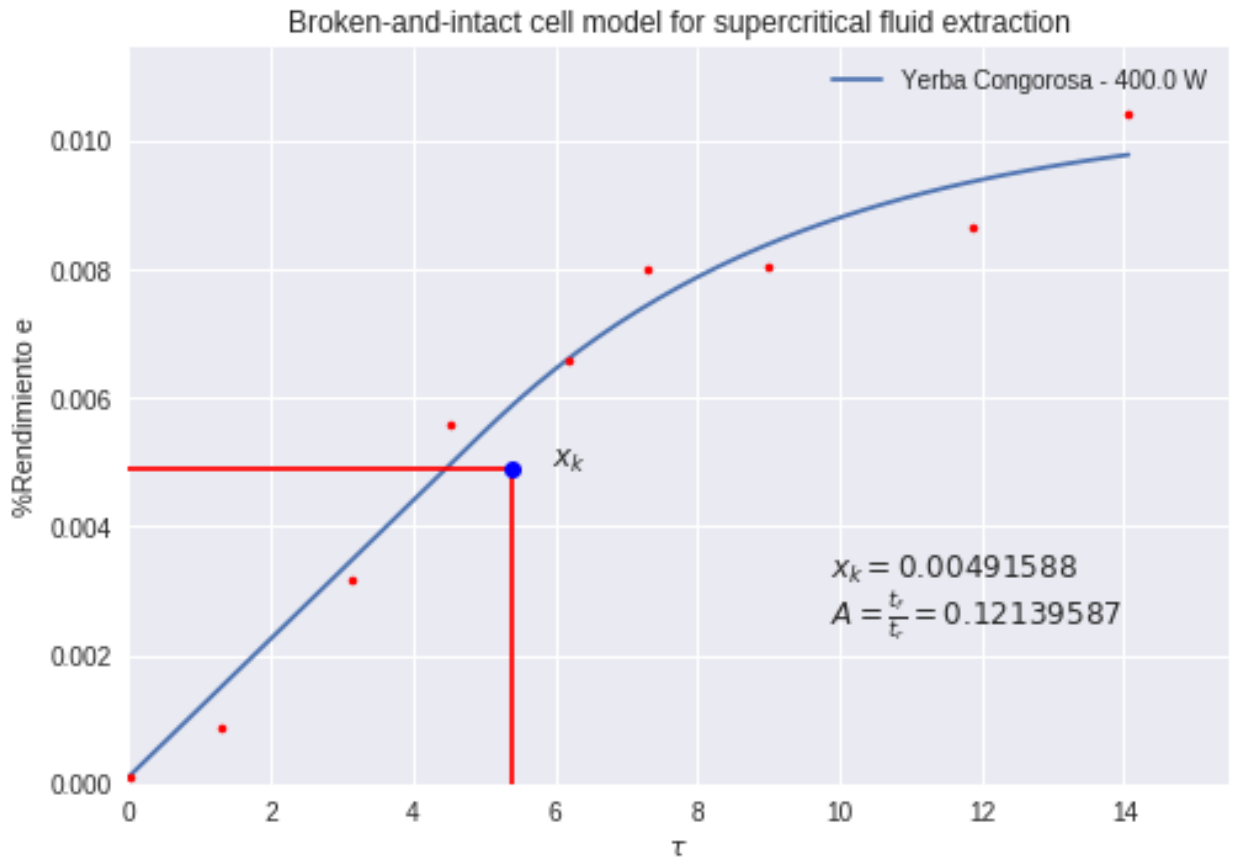

```
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:4:␣
↪RuntimeWarning: overflow encountered in exp
```

```
(1.6759521390448247, 3.7030513340813194, inf)
```

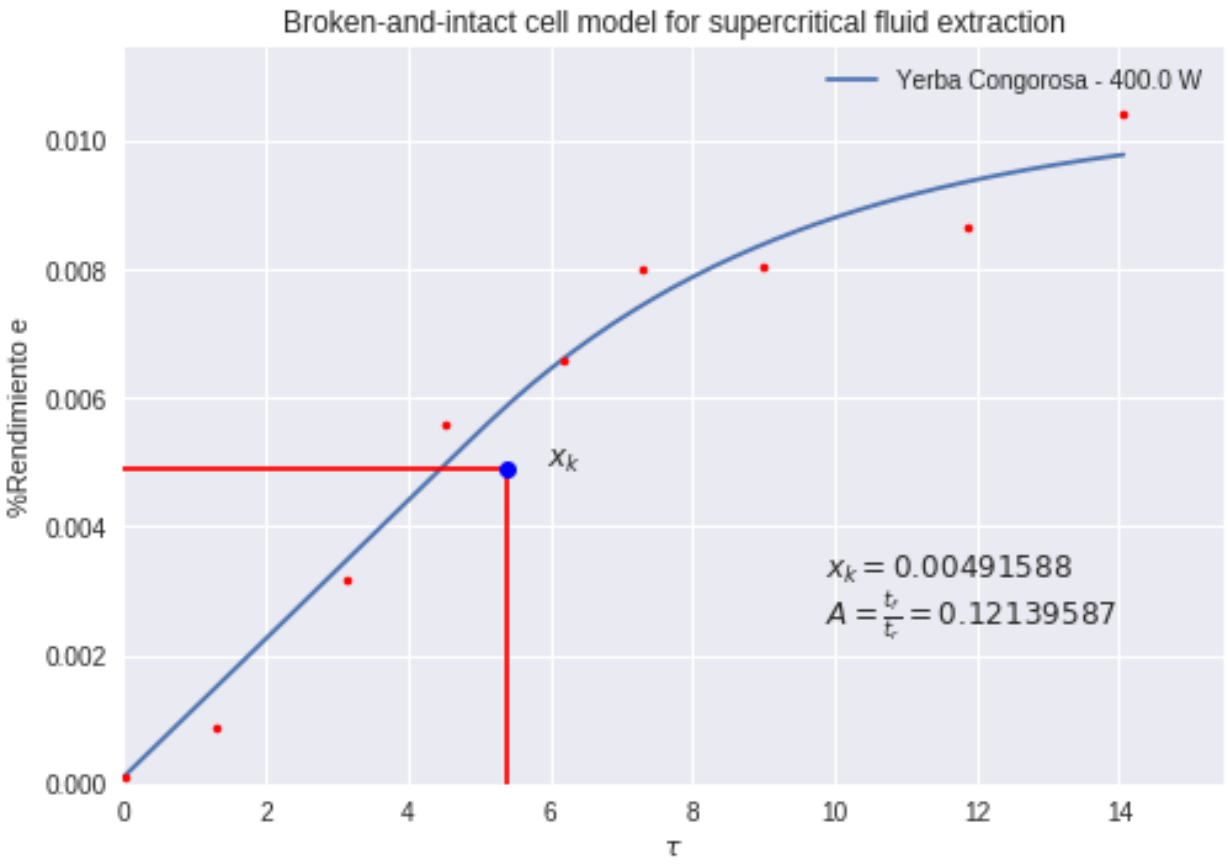
```
[<matplotlib.lines.Line2D at 0x7f203f0cd240>]
```




```
NOMBRE: Congorosa
```




```
NOMBRE: Congorosa
```


Broken-and-intact cell model for supercritical fluid extraction
Yerba Congorosa - 400.0 W
%Rendimiento e
τ
x_k
$x_k = 0.00491588$
$A = \frac{t_r}{t_r} = 0.12139587$
0.000
0.002
0.004
0.006
0.008
0.010
0
2
4
6
8
10
12
14

CHAPTER 20

Modelo Sovova

```
nan nan
```

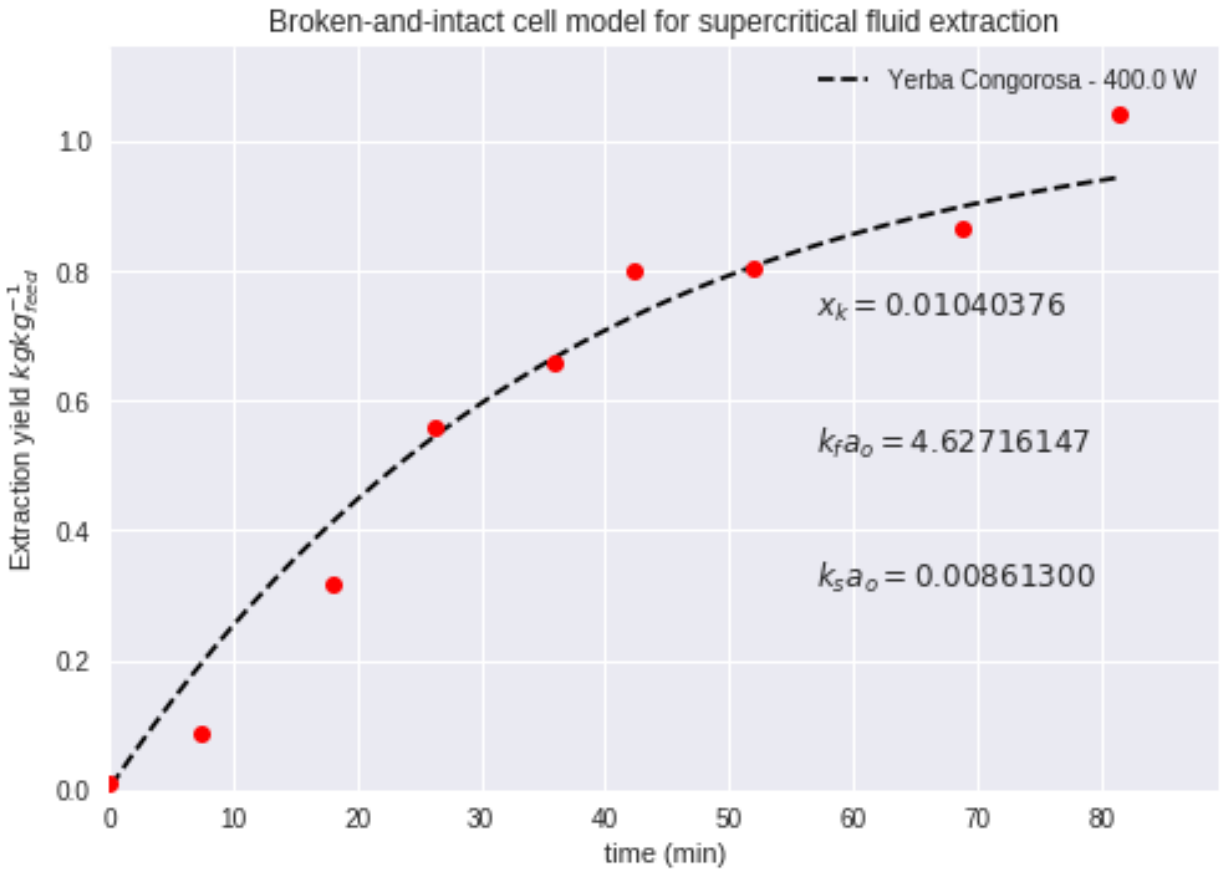
```
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:9:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:10:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:28:␣
↪RuntimeWarning: invalid value encountered in log
```

```
0.032539999999999993
```

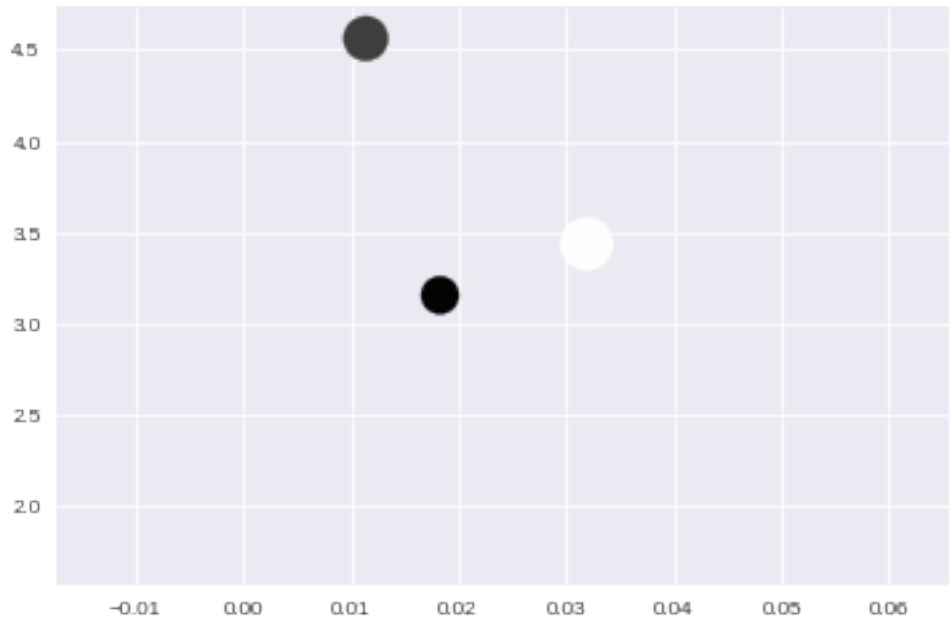
```
active_mask: array([0, 0, 0])
       cost: 0.067573820707988189
        fun: array([ 1.81182816, -0.78683877, -0.26158694,  0.02518217, -0.
↪00921332,
       0.09234707, -0.00521931, -0.0395414 ,  0.09887852])
       grad: array([  4.91259316e-02,  -2.61716140e-06,  -1.03208360e-05])
        jac: array([[  1.64962614e-05,  -7.38580870e-12,  -2.72356004e-09],
      [  5.71402153e-06,  -1.02692647e-13,  -1.42693543e-06],
      [  2.03829982e-06,  -3.70461323e-14,  -1.24830113e-06],
      [  7.58134626e+01,  -1.35986181e-06,  -6.79077966e+01],
      [  5.16927813e+01,  -9.15747753e-07,  -6.36407023e+01],
      [  8.24606189e+00,  -1.49376459e-07,  -1.20250951e+01],
      [  2.71129286e+01,  -5.00329665e-07,  -4.89541775e+01],
      [  1.19393497e+01,  -2.25101586e-07,  -2.87292828e+01],
      [  7.43796280e-01,  -1.26455894e-08,  -2.13102359e+00]])
    message: '`xtol` termination condition is satisfied.'
       nfev: 31
       njev: 16
 optimality: 0.000515952907784016
     status: 3
    success: True
```

```
        x: array([ 0.01040376,  4.62716147,  0.008613  ])
```

```
[<matplotlib.lines.Line2D at 0x7effbc368080>]
```




```
  File "<ipython-input-337-5e85ffa89130>", line 1
    x: array([ 0.03829315,  1.70819818,  0.01795442]) #0
     ^
SyntaxError: invalid syntax
```




```
array([ 0.00017276,  0.00280808,  0.00535308,  0.0078023 ,  0.01015068,
        0.01239369,  0.01452747,  0.01654892,  0.01845584,  0.02024697,
        0.02192205,  0.02348181,  0.02492799,  0.02626321,  0.02749096,
        0.02861545,  0.02964145,  0.03057426,  0.03141945,  0.03218284,
        0.0328703 ,  0.0334877 ,  0.03404079,  0.03453511,  0.03497599,
        0.03536845,  0.03571721,  0.03602665,  0.03630083,  0.03654345,
        0.03675791,  0.03694729,  0.03711438,  0.03726167,  0.03739143,
        0.03750567,  0.0376062 ,  0.03769461,  0.03777233,  0.03784064,
        0.03790064,  0.03795334,  0.03799961,  0.03804023,  0.03807588,
        0.03810716,  0.0381346 ,  0.03815868,  0.0381798 ,  0.03819832])
```

```
(0.22974968576541693,
 24.199480533020207,
 5.9097726601403524,
 23.083006310073888,
 0.023157530596497361,
 3.292814746661628)
```

```
Help on function plot in module matplotlib.pyplot:

plot(args, **kwargs)
    Plot y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], data=None, **kwargs)
        plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, y.

    The optional parameter fmt is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string
    notation described in the Notes section below.
```

```
>>> plot(x, y)        # plot x and y using default line style and color
>>> plot(x, y, 'bo')  # plot x and y using blue circle markers
>>> plot(y)           # plot y using x as index array 0..N-1
>>> plot(y, 'r+')     # ditto, but with red plusses
```

You can use *.Line2D* properties as keyword arguments for more control on the appearance. Line properties and *fmt* can be mixed. The following two calls yield identical results:

```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
        linewidth=2, markersize=12)
```

When conflicting with *fmt*, keyword arguments take precedence.

Plotting labelled data

There's a convenient way for plotting objects with labelled data (i.e. data that can be accessed by index obj['y']). Instead of giving the data in *x* and *y*, you can provide the object in the *data* parameter and just give the labels for *x* and *y*::

```
>>> plot('xlabel', 'ylabel', data=obj)
```

All indexable objects are supported. This could e.g. be a *dict*, a *pandas.DataFame* or a structured numpy array.

Plotting multiple sets of data

There are various ways to plot multiple sets of data.

- The most straight forward way is just to call *plot* multiple times. Example:

  ```
  >>> plot(x1, y1, 'bo')
  >>> plot(x2, y2, 'go')
  ```

- Alternatively, if your data is already a 2d array, you can pass it directly to *x*, *y*. A separate data set will be drawn for every column.

  Example: an array a where the first column represents the *x* values and the other columns are the *y* columns::

  ```
  >>> plot(a[0], a[1:])
  ```

- The third way is to specify multiple sets of *[x]*, *y*, *[fmt]* groups::

  ```
  >>> plot(x1, y1, 'g^', x2, y2, 'g-')
  ```

  In this case, any additional keyword argument applies to all datasets. Also this syntax cannot be combined with the *data*

```
  parameter.

By default, each line is assigned a different style specified by a
'style cycle'. The fmt and line property parameters are only
necessary if you want explicit deviations from these defaults.
Alternatively, you can also change the style cycle using the
'axes.prop_cycle' rcParam.

Parameters
----------
x, y : array-like or scalar
    The horizontal / vertical coordinates of the data points.
    x values are optional. If not given, they default to
    [0, ..., N-1].

    Commonly, these parameters are arrays of length N. However,
    scalars are supported as well (equivalent to an array with
    constant value).

    The parameters can also be 2-dimensional. Then, the columns
    represent separate data sets.

fmt : str, optional
    A format string, e.g. 'ro' for red circles. See the Notes
    section for a full description of the format strings.

    Format strings are just an abbreviation for quickly setting
    basic line properties. All of these and more can also be
    controlled by keyword arguments.

data : indexable object, optional
    An object with labelled data. If given, provide the label names to
    plot in x and y.

    .. note::
        Technically there's a slight ambiguity in calls where the
        second label is a valid fmt. plot('n', 'o', data=obj)
        could be plt(x, y) or plt(y, fmt). In such cases,
        the former interpretation is chosen, but a warning is issued.
        You may suppress the warning by adding an empty format string
        plot('n', 'o', '', data=obj).


Other Parameters
----------------
scalex, scaley : bool, optional, default: True
    These parameters determined if the view limits are adapted to
    the data limits. The values are passed on to autoscale_view.

kwargs : `.Line2D` properties, optional
    *kwargs* are used to specify properties like a line label (for
    auto legends), linewidth, antialiasing, marker face color.
    Example::
```

```
        >>> plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2)
        >>> plot([1,2,3], [1,4,9], 'rs',  label='line 2')

        If you make multiple lines with one plot command, the kwargs
        apply to all those lines.

        Here is a list of available `.Line2D` properties:

          agg_filter: a filter function, which takes a (m, n, 3) float array␣
↪and a dpi value, and returns a (m, n, 3) array
      alpha: float (0.0 transparent through 1.0 opaque)
      animated: bool
      antialiased or aa: bool
      clip_box: a `.Bbox` instance
      clip_on: bool
      clip_path: [(`~matplotlib.path.Path`, `.Transform`) | `.Patch` | None]
      color or c: any matplotlib color
      contains: a callable function
      dash_capstyle: ['butt' | 'round' | 'projecting']
      dash_joinstyle: ['miter' | 'round' | 'bevel']
      dashes: sequence of on/off ink in points
      drawstyle: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-
↪post']
      figure: a `.Figure` instance
      fillstyle: ['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']
      gid: an id string
      label: object
      linestyle or ls: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-
↪off-dash-seq) | ``'-'`` | ``'--'`` | ``'-.'`` | ``':'`` | ``'None'`` | ``'␣
↪'`` | ``''``]
      linewidth or lw: float value in points
      marker: :mod:`A valid marker style <matplotlib.markers>`
      markeredgecolor or mec: any matplotlib color
      markeredgewidth or mew: float value in points
      markerfacecolor or mfc: any matplotlib color
      markerfacecoloralt or mfcalt: any matplotlib color
      markersize or ms: float
      markevery: [None | int | length-2 tuple of int | slice | list/array of␣
↪int | float | length-2 tuple of float]
      path_effects: `.AbstractPathEffect`
      picker: float distance in points or callable pick function ``fn(artist,␣
↪event)``
      pickradius: float distance in points
      rasterized: bool or None
      sketch_params: (scale: float, length: float, randomness: float)
      snap: bool or None
      solid_capstyle: ['butt' | 'round' |  'projecting']
      solid_joinstyle: ['miter' | 'round' | 'bevel']
      transform: a :class:`matplotlib.transforms.Transform` instance
      url: a url string
      visible: bool
      xdata: 1D array
      ydata: 1D array
      zorder: float
```

Returns
----
lines
A list of `.Line2D` objects representing the plotted data.

See Also
----
scatter : XY scatter plot with markers of varying size and/or color (
sometimes also called bubble chart).

Notes

****Format Strings**

A format string consists of a part for color, marker and line::

fmt = '[color][marker][line]'

Each of them is optional. If not provided, the value from the style
cycle is used. Exception: If line is given, but no marker,
the data will be a line without markers.

Colors

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

If the color is the only part of the format string, you can
additionally use any *matplotlib.colors* spec, e.g. full names
('green') or hex strings ('#008000').

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker

```
    'v'          triangle_down marker
    '^'          triangle_up marker
    '<'          triangle_left marker
    '>'          triangle_right marker
    '1'          tri_down marker
    '2'          tri_up marker
    '3'          tri_left marker
    '4'          tri_right marker
    's'          square marker
    'p'          pentagon marker
    '*'          star marker
    'h'          hexagon1 marker
    'H'          hexagon2 marker
    '+'          plus marker
    'x'          x marker
    'D'          diamond marker
    'd'          thin_diamond marker
    '|'          vline marker
    '_'          hline marker
    =============    ==============================

    Line Styles

    =============    ==============================
    character        description
    =============    ==============================
    '-'          solid line style
    '--'         dashed line style
    '-.'         dash-dot line style
    ':'          dotted line style
    =============    ==============================

    Example format strings::

        'b'    # blue markers with default shape
        'ro'   # red circles
        'g-'   # green solid line
        '--'   # dashed line with default color
        'k^:'  # black triangle_up markers connected by a dotted line

    .. note::
        In addition to the above described arguments, this function can take a
        data keyword argument. If such a data argument is given, the
        following arguments are replaced by data[<arg>]:

        * All arguments with the following names: 'x', 'y'.
```

```
  File "<ipython-input-104-c3b0085c4838>", line 1
    cielo, la persona de mkt es la que tiene que ofrecer un listado de servicios y␣
↪productos:
                  ^
SyntaxError: invalid syntax
```

CHAPTER 21

Preprocesamiento de los datos

```
[['Boldo_0W', 'Boldo_160W', 'Boldo_200W', 'Boldo_280W', 'Boldo_400W'], ['Brasil_0W',
↪'Brasil_200W', 'Brasil_280W', 'Brasil_400W'], ['Uruguay_0W', 'Uruguay_200W',
↪'Uruguay_280W', 'Uruguay_400W', 'Uruguay_600W'], ['Romero_0W', 'Romero_200W',
↪'Romero_280W', 'Romero_400W'], ['Congorosa_0W', 'Congorosa_200W', 'Congorosa_280W',
↪'Congorosa_400W']]
[[   minutos  rendimiento
0    0.000       0.0000
1    6.683       0.9460
2    6.953       1.1193
3   17.053       2.3520
4   25.069       2.8202
5   34.586       3.2030
6   41.586       3.3810
7   52.353       3.5706
8   69.019       3.7933
9   84.019       3.6280,    minutos  rendimiento
0    0.000        0.000
1    7.250        1.127
2   20.117        2.557
3   25.200        2.835
4   35.917        3.113
5   43.033        3.272
6   53.333        3.406
7   72.950        3.591
8   88.100        3.686,    minutos  rendimiento
0    0.000        0.000
1    6.850        1.095
2   17.250        2.350
3   25.333        2.872
4   34.750        3.228
5   41.400        3.383
6   58.317        3.555,    minutos  rendimiento
0    0.000        0.000
1    7.383        1.093
```

(continues on next page)

(continued from previous page)

```
2   18.333         2.339
3   25.333         2.861
4   37.750         3.157
5   43.883         3.301
6   55.750         3.700
7   68.150         3.700
8   82.750         3.821,    minutos  rendimiento
0    0.000         0.000
1    6.833         1.013
2   18.250         2.206
3   25.483         2.656
4   34.900         2.991
5   41.533         3.433
6   53.183         3.650
7   68.150         3.829
8   82.750         3.934], [   minutos  rendimiento
0     0.00        0.0000
1     3.00        0.0508
2    17.58        0.4874
3    25.00        0.7331
4    35.23        1.0231
5    40.65        1.1210
6    50.37        1.1710
7    70.28        1.2448
8    96.00        1.3626,    minutos  rendimiento
0     0.00         0.000
1     5.00         0.100
2    16.45         0.761
3    25.27         1.073
4    35.53         1.317
5    42.28         1.438
6    55.08         1.532
7    72.85         1.643
8    87.20         1.766,     minutos  rendimiento
0     0.000         0.000
1     6.733         0.270
2    16.450         0.761
3    17.400         0.846
4    25.270         1.073
5    26.000         1.126
6    35.530         1.317
7    37.067         1.365
8    42.280         1.438
9    45.300         1.522
10   55.080         1.532
11   58.750         1.697
12   77.000         1.760
13   87.200         1.766,    minutos  rendimiento
0    0.000         0.000
1    6.900         0.174
2   17.367         0.622
3   26.050         0.797
4   38.500         1.006
5   45.900         1.062
6   58.683         1.184
7   77.517         1.354
8   90.183         1.493], [    minutos  rendimiento
```

(continues on next page)

(continued from previous page)

```
0     0.000        0.000
1     6.667        0.200
2     7.550        0.283
3    17.117        0.750
4    25.583        0.977
5    35.000        1.146
6    41.667        1.231
7    52.450        1.335
8    59.000        1.365
9    79.333        1.450
10   96.733        1.344,    minutos  rendimiento
0     0.000       0.000
1     6.967       0.095
2    17.750       0.234
3    26.150       0.298
4    36.667       0.345
5    44.667       0.384
6    53.083       0.416
7    74.867       0.451
8    84.917       0.506,    minutos  rendimiento
0     0.000       0.000
1     7.417       0.078
2    17.750       0.226
3    25.717       0.294
4    37.033       0.365
5    44.567       0.395
6    55.667       0.426
7    73.250       0.480
8    86.667       0.540,    minutos  rendimiento
0     0.000       0.000
1     7.583       0.074
2    19.417       0.221
3    28.167       0.357
4    37.917       0.432
5    44.200       0.462
6    54.883       0.498
7    70.233       0.556
8    81.850       0.579,    minutos  rendimiento
0     0.000       0.000
1     7.000       0.092
2    17.500       0.221
3    25.100       0.381
4    38.450       0.469
5    46.450       0.520
6    57.167       0.564
7    70.100       0.618], [   minutos  rendimiento
0     0.000       0.000
1     7.283       1.031
2    17.333       2.284
3    24.750       2.790
4    34.117       3.215
5    40.817       3.379
6    56.705       3.621
7    73.972       3.767
8    84.972       3.833,    minutos  rendimiento
0     0.000       0.000
1     6.683       1.456
```

(continues on next page)

(continued from previous page)

```
2   15.767        2.491
3   23.083        2.916
4   32.150        3.300
5   38.667        3.350
6   48.250        3.438
7   64.533        3.553
8   79.917        3.719,    minutos  rendimiento
0    0.000        0.000
1    7.333        1.399
2   21.083        2.378
3   28.483        2.976
4   38.317        3.619
5   45.117        3.750
6   55.650        3.908
7   70.867        4.124
8   84.867        4.292,    minutos  rendimiento
0    0.000        0.000
1    7.400        1.461
2   17.467        2.412
3   25.800        3.128
4   35.200        3.239
5   45.800        3.254], [   minutos  rendimiento
0     0.00       0.0000
1     5.00       0.0080
2     6.83       0.0250
3    16.27       0.2689
4    24.33       0.3520
5    33.80       0.4290
6    39.45       0.4292
7    50.47       0.4336
8    64.15       0.5580
9    79.05       0.6090,     minutos  rendimiento
0     0.000        0.000
1     5.000        0.119
2     6.917        0.154
3     7.480        0.230
4    16.983        0.634
5    18.120        0.638
6    25.833        0.879
7    27.030        0.788
8    33.983        0.948
9    37.330        0.908
10   44.580        0.957
11   49.783        1.012
12   51.067        1.092
13   55.700        1.023
14   69.667        1.155
15   74.830        1.182
16   87.100        1.317
17   91.233        1.281,      minutos  rendimiento
0   0.000000        0.000
1   6.470000        0.140
2  15.720000        0.498
3  22.580000        0.638
4  31.870000        0.712
5  38.830000        0.721
6  49.220000        0.721
```

(continues on next page)

(continued from previous page)

```
7  65.000000        0.756
8  79.816667        0.884,    minutos  rendimiento
0      0.00        0.000
1      7.47        0.089
2     18.10        0.319
3     26.27        0.559
4     35.83        0.659
5     42.25        0.800
6     52.00        0.803
7     68.73        0.864
8     81.43        1.042]]
```

```
Index(['minutos', 'rendimiento', 'minutos', 'rendimiento', 'minutos',
       'rendimiento', 'minutos', 'rendimiento', 'minutos', 'rendimiento'],
      dtype='object')
```

CHAPTER 22

Datos operación Yerba Romero

```
[   0.  200.  280.  400.]
tr =  [ 7.89494813  7.34917765  7.84414133  8.27932474]
```

```
Yerba Romero - 400.0 W
X0 =  [ 0.03833  0.03719  0.04292  0.03254] <class 'numpy.ndarray'>
xo = 0.03253999999999999, gamma = 2.1692508973607825, yr = 0.0074, TAO = 5.
↪531852105882878
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-02   7.40000000e+00   1.74670000e+01   2.58000000e+01
   3.52000000e+01   4.58000000e+01] <class 'numpy.ndarray'>
[ 0.01        0.8937927   2.10971312  3.11619616  4.25155446  5.53185211] <class
↪'numpy.ndarray'>
[ 0.0001   0.01461  0.02412  0.03128  0.03239  0.03254]
```

CHAPTER 23

Datos operación Yerba uruguay

```
[   0.  200.  280.  400.  600.]
tr =  [ 6.90315722  7.06313825  7.19084665  6.90573444  6.96705883]
```

```
Yerba Uruguay - 600.0 W
X0 =  [ 0.0145   0.00506  0.0054   0.00579  0.00618] <class 'numpy.ndarray'>
xo = 0.00618, gamma = 1.8798398443346804, yr = 0.0013, TAO = 10.061634577110048
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-02   1.00472813e+00   2.51182033e+00   3.60266802e+00
   5.51882810e+00   6.66708882e+00   8.20532759e+00   1.00616346e+01] <class 'numpy.
↪ndarray'>
[ 0.0001   0.00092  0.00221  0.00381  0.00469  0.0052   0.00564  0.00618]
```

CHAPTER 24

fin yu

CHAPTER 25

Datos de operación yerba bolbo

```
tr =  [ 6.0855294   6.4236725   7.25440914  6.17722886  6.18946974]
[ 0.28606652  0.27100794  0.23997354  0.28181994  0.28126258]
```

```
['bolbo' 'bolbo' 'bolbo' 'bolbo' 'bolbo' 'brasil' 'brasil' 'brasil'
 'brasil' 'congorosa' 'congorosa' 'congorosa' 'congorosa']
```

```
{'Densidad gr/cm3': {'bolbo': 1.25, 'brasil': 1.34, 'congorosa': 1.31},
 'Porosidad': {'bolbo': 0.7208, 'brasil': 0.7179, 'congorosa': 0.7137}}
```

```
280.0
Yerba Bolbo - 400.0 W
X0 =  [ 0.037933  0.03686   0.03555   0.03821   0.03934 ] <class 'numpy.ndarray'>
xo = 0.03934, gamma = 1.7408662464183382, yr = 0.0059, TAO = 13.369481303875684
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  3.00000000e-03   1.10397179e+00   2.94855630e+00   4.11715398e+00
   5.63860903e+00   6.71026788e+00   8.59249697e+00   1.10106363e+01
   1.33694813e+01] <class 'numpy.ndarray'>
[ 0.001    0.01013  0.02206  0.02656  0.02991  0.03433  0.0365   0.03829
  0.03934]
```

CHAPTER 26

Datos de operación yerba brasil

```
[   0.  200.  280.  400.]
tr =  [ 6.27556792  5.90557503  6.16106767  6.39774269]
```

```
Yerba Brasil - 400.0 W
X0 =  [ 0.013626  0.01766   0.01766   0.01493 ] <class 'numpy.ndarray'>
xo = 0.01493, gamma = 1.6007817435333078, yr = 0.0059, TAO = 14.096065489128808
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-04   1.07850539e+00   2.71455118e+00   4.07174862e+00
   6.01774748e+00   7.17440544e+00   9.17245391e+00   1.21163047e+01
   1.40960655e+01] <class 'numpy.ndarray'>
[ 0.0001   0.00174  0.00622  0.00797  0.01006  0.01062  0.01184  0.01354
  0.01493]
```

CHAPTER 27

Datos de operación yerba congorosa

```
tr =  [ 5.42987582  6.11480591  5.44828666  5.78986569]
```

```
Yerba Congorosa - 400.0 W
X0 =  [ 0.00609  0.01317  0.00884  0.01042] <class 'numpy.ndarray'>
xo = 0.01042, gamma = 1.6039805840774504, yr = 0.0059, TAO = 14.064229522040602
```

```
array([[ 0.01    ,  1.      ],
       [ 0.01    ,  0.5     ],
       [ 0.030321,  1.5     ]])
```

```
[  1.00000000e-02   1.29018537e+00   3.12615196e+00   4.53723824e+00
   6.18839916e+00   7.29723317e+00   8.98121006e+00   1.18707417e+01
   1.40642295e+01] <class 'numpy.ndarray'>
[ 0.0001   0.00089  0.00319  0.00559  0.00659  0.008    0.00803  0.00864
  0.01042]
```

CHAPTER 28

Modelo Lack

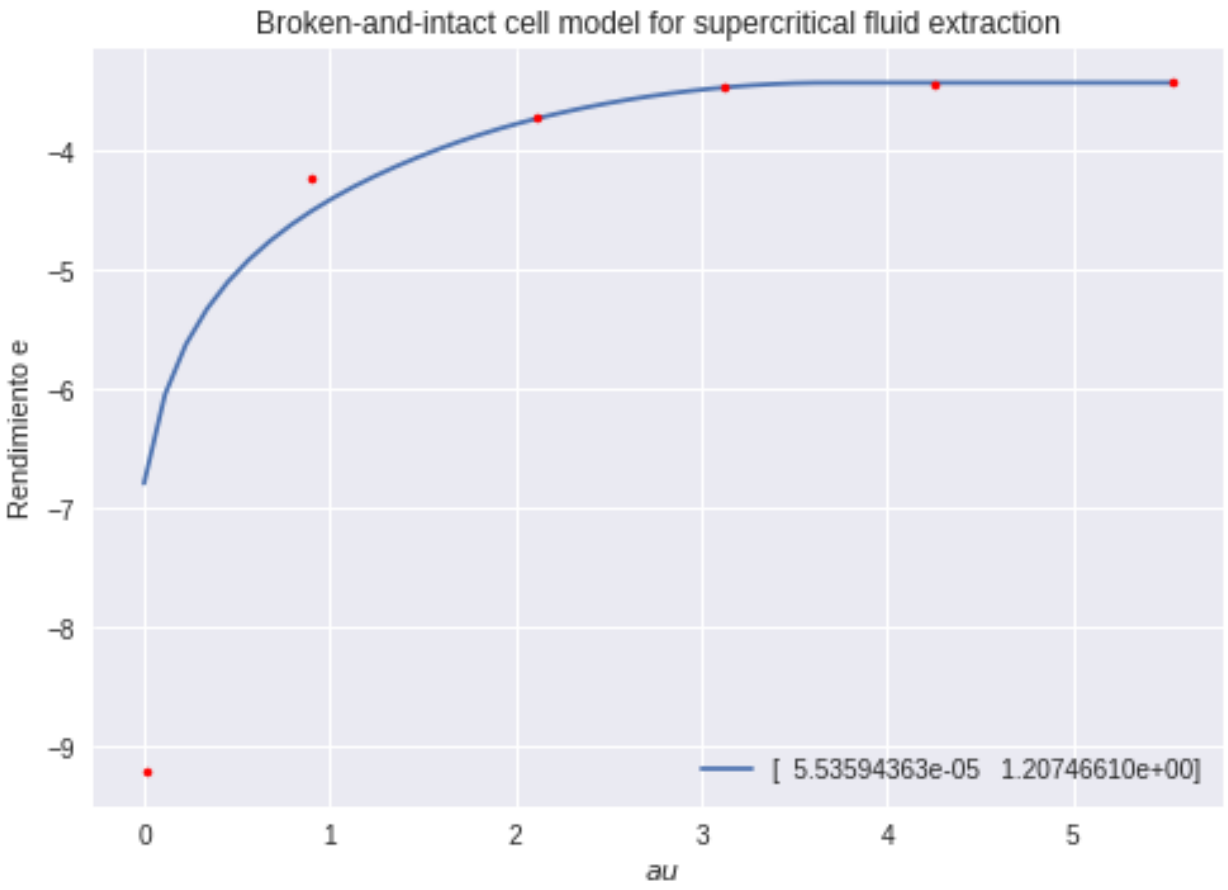
CHAPTER 29

Ajuste de los parámetros del modelo 1: Modelo Lack

```
active_mask: array([0, 0])
       cost: 0.018300978793627454
        fun: array([-0.11809526,  0.37330728,  0.02808377,  0.00718511, -0.
↪00462037,  0.        ])
       grad: array([  8.33403474e-04,  -1.54227422e-08])
        jac: array([[  0.00000000e+00,  -6.35444581e-09],
      [  0.00000000e+00,  -6.35444434e-09],
      [  2.06735361e-02,  -3.11057628e-01],
      [  3.17151307e-02,  -1.76298562e-01],
      [  0.00000000e+00,   0.00000000e+00],
      [  0.00000000e+00,   0.00000000e+00]])
    message: 'Both ftol and xtol termination conditions are satisfied.'
       nfev: 22
       njev: 12
 optimality: 6.7744917811128669e-08
     status: 4
    success: True
          x: array([  5.53594363e-05,   1.20746610e+00])
```

```
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:5:␣
↪RuntimeWarning: invalid value encountered in log
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:4:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:5:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:14:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:17:␣
↪RuntimeWarning: invalid value encountered in log
```

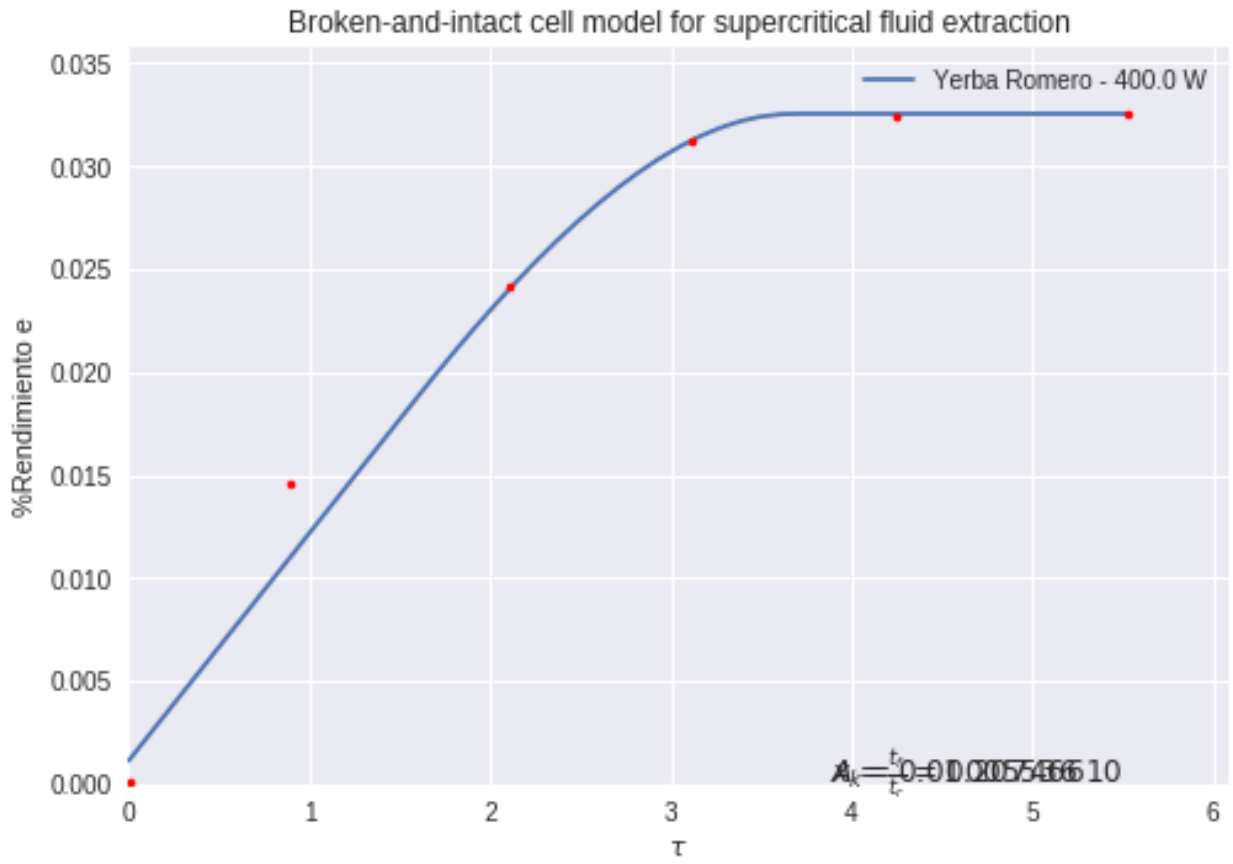
```
[<matplotlib.lines.Line2D at 0x7f203f10ff98>]
```



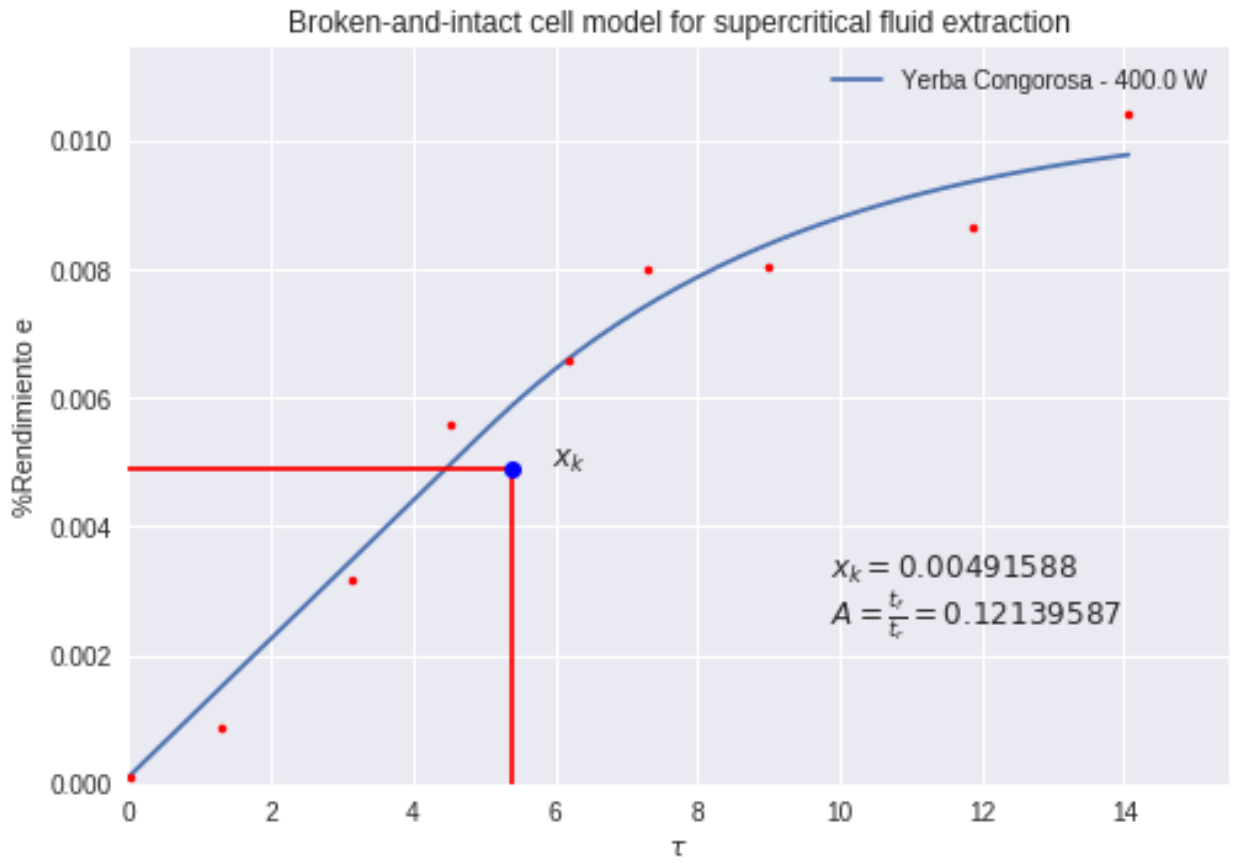

```
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:4:␣
↪RuntimeWarning: overflow encountered in exp
```

```
(1.6759521390448247, 3.7030513340813194, inf)
```

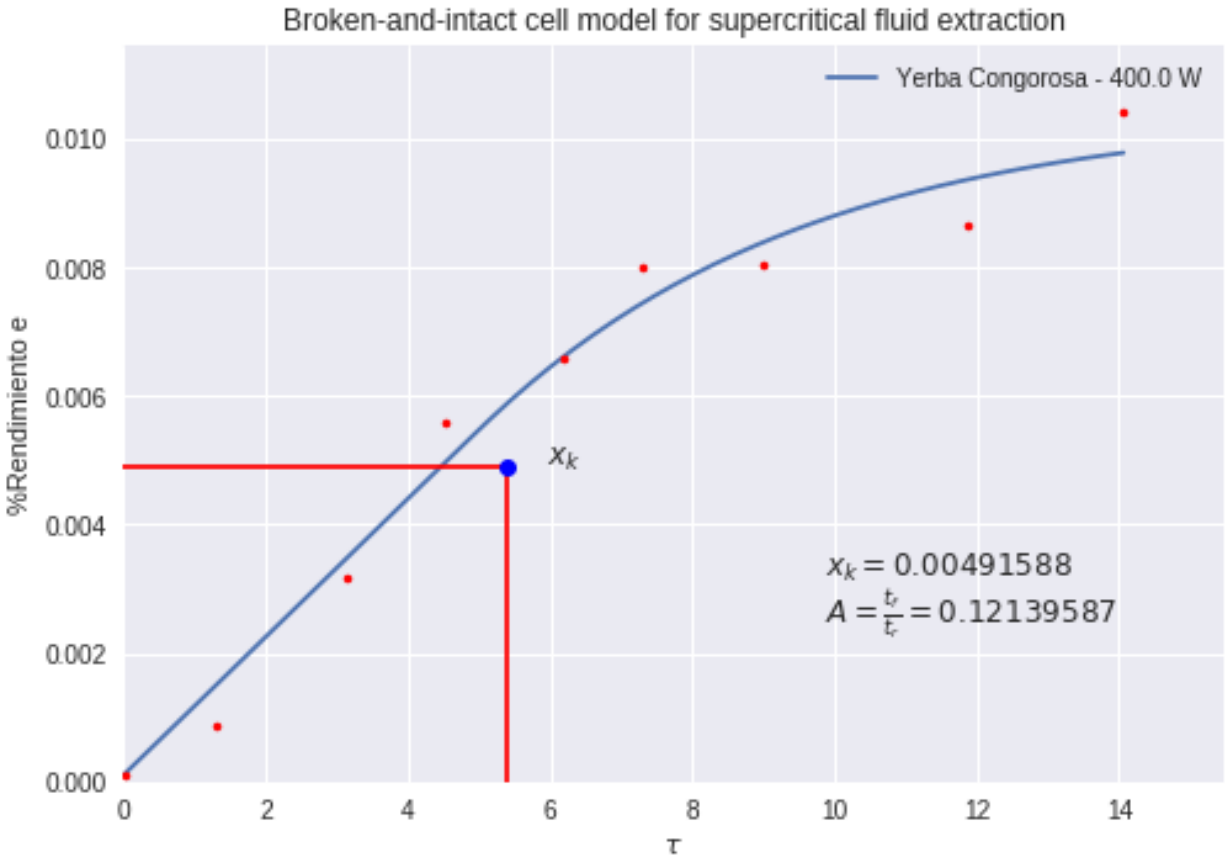
```
[<matplotlib.lines.Line2D at 0x7f203f0cd240>]
```




```
NOMBRE: Congorosa
```




```
NOMBRE: Congorosa
```


Broken-and-intact cell model for supercritical fluid extraction
Yerba Congorosa - 400.0 W
%Rendimiento e
0.010
0.008
0.006
0.004
0.002
0.000
0
2
4
6
8
10
12
14
τ
x_k
$x_k = 0.00491588$
$A = \frac{t_r}{t_r} = 0.12139587$

CHAPTER 30

Modelo Sovova

```
nan nan
```

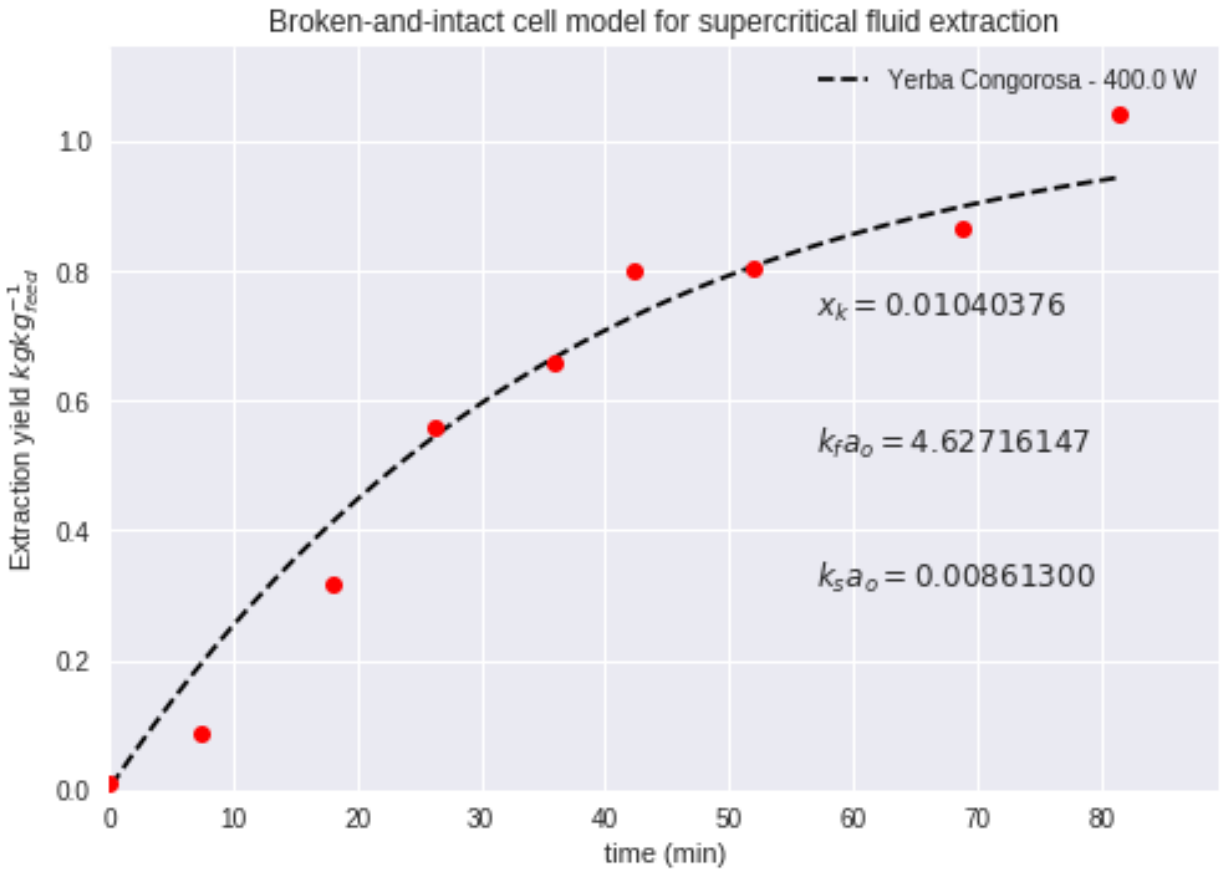
```
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:9:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:10:␣
↪RuntimeWarning: overflow encountered in exp
/home/andres-python/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:28:␣
↪RuntimeWarning: invalid value encountered in log
```

```
0.032539999999999993
```

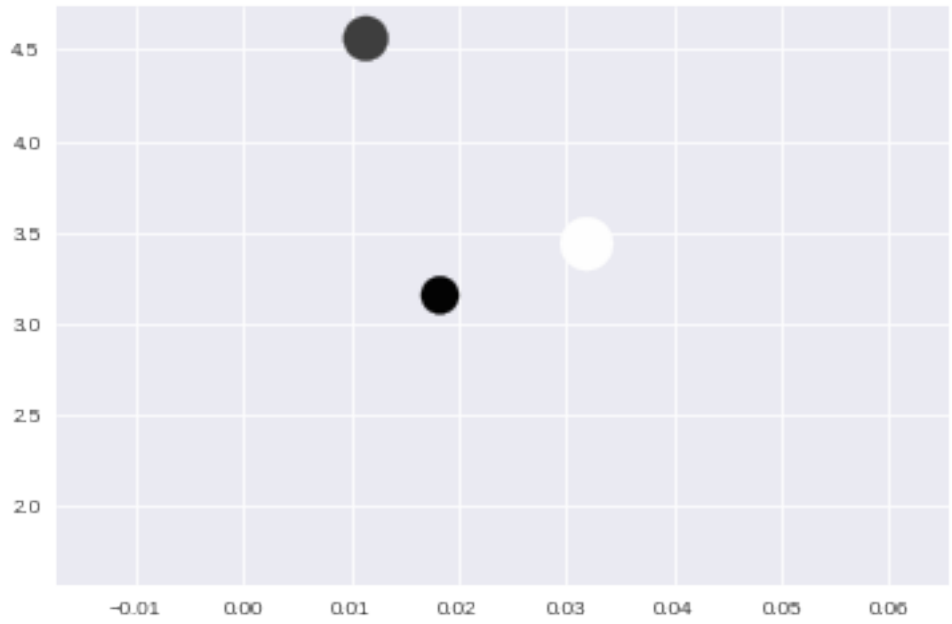
```
active_mask: array([0, 0, 0])
       cost: 0.067573820707988189
        fun: array([ 1.81182816, -0.78683877, -0.26158694,  0.02518217, -0.
↪00921332,
       0.09234707, -0.00521931, -0.0395414 ,  0.09887852])
       grad: array([  4.91259316e-02,  -2.61716140e-06,  -1.03208360e-05])
        jac: array([[  1.64962614e-05,  -7.38580870e-12,  -2.72356004e-09],
      [  5.71402153e-06,  -1.02692647e-13,  -1.42693543e-06],
      [  2.03829982e-06,  -3.70461323e-14,  -1.24830113e-06],
      [  7.58134626e+01,  -1.35986181e-06,  -6.79077966e+01],
      [  5.16927813e+01,  -9.15747753e-07,  -6.36407023e+01],
      [  8.24606189e+00,  -1.49376459e-07,  -1.20250951e+01],
      [  2.71129286e+01,  -5.00329665e-07,  -4.89541775e+01],
      [  1.19393497e+01,  -2.25101586e-07,  -2.87292828e+01],
      [  7.43796280e-01,  -1.26455894e-08,  -2.13102359e+00]])
    message: '`xtol` termination condition is satisfied.'
       nfev: 31
       njev: 16
 optimality: 0.000515952907784016
     status: 3
    success: True
```

```
        x: array([ 0.01040376,  4.62716147,  0.008613  ])
```

```
[<matplotlib.lines.Line2D at 0x7effbc368080>]
```




```
  File "<ipython-input-337-5e85ffa89130>", line 1
    x: array([ 0.03829315,  1.70819818,  0.01795442]) #0
     ^
SyntaxError: invalid syntax
```




```
array([ 0.00017276,  0.00280808,  0.00535308,  0.0078023 ,  0.01015068,
        0.01239369,  0.01452747,  0.01654892,  0.01845584,  0.02024697,
        0.02192205,  0.02348181,  0.02492799,  0.02626321,  0.02749096,
        0.02861545,  0.02964145,  0.03057426,  0.03141945,  0.03218284,
        0.0328703 ,  0.0334877 ,  0.03404079,  0.03453511,  0.03497599,
        0.03536845,  0.03571721,  0.03602665,  0.03630083,  0.03654345,
        0.03675791,  0.03694729,  0.03711438,  0.03726167,  0.03739143,
        0.03750567,  0.0376062 ,  0.03769461,  0.03777233,  0.03784064,
        0.03790064,  0.03795334,  0.03799961,  0.03804023,  0.03807588,
        0.03810716,  0.0381346 ,  0.03815868,  0.0381798 ,  0.03819832])
```

```
(0.22974968576541693,
 24.199480533020207,
 5.9097726601403524,
 23.083006310073888,
 0.023157530596497361,
 3.292814746661628)
```

```
Help on function plot in module matplotlib.pyplot:

plot(args, **kwargs)
    Plot y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], data=None, **kwargs)
        plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, y.

    The optional parameter fmt is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string
    notation described in the Notes section below.
```

```
>>> plot(x, y)        # plot x and y using default line style and color
>>> plot(x, y, 'bo')  # plot x and y using blue circle markers
>>> plot(y)           # plot y using x as index array 0..N-1
>>> plot(y, 'r+')     # ditto, but with red plusses
```

You can use *.Line2D* properties as keyword arguments for more control on the appearance. Line properties and *fmt* can be mixed. The following two calls yield identical results:

```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
        linewidth=2, markersize=12)
```

When conflicting with *fmt*, keyword arguments take precedence.

Plotting labelled data

There's a convenient way for plotting objects with labelled data (i.e. data that can be accessed by index obj['y']). Instead of giving the data in *x* and *y*, you can provide the object in the *data* parameter and just give the labels for *x* and *y*::

```
>>> plot('xlabel', 'ylabel', data=obj)
```

All indexable objects are supported. This could e.g. be a *dict*, a *pandas.DataFame* or a structured numpy array.

Plotting multiple sets of data

There are various ways to plot multiple sets of data.

- The most straight forward way is just to call *plot* multiple times. Example:

  ```
  >>> plot(x1, y1, 'bo')
  >>> plot(x2, y2, 'go')
  ```

- Alternatively, if your data is already a 2d array, you can pass it directly to *x*, *y*. A separate data set will be drawn for every column.

  Example: an array a where the first column represents the *x* values and the other columns are the *y* columns::

  ```
  >>> plot(a[0], a[1:])
  ```

- The third way is to specify multiple sets of *[x]*, *y*, *[fmt]* groups::

  ```
  >>> plot(x1, y1, 'g^', x2, y2, 'g-')
  ```

  In this case, any additional keyword argument applies to all datasets. Also this syntax cannot be combined with the *data*

```
  parameter.

By default, each line is assigned a different style specified by a
'style cycle'. The fmt and line property parameters are only
necessary if you want explicit deviations from these defaults.
Alternatively, you can also change the style cycle using the
'axes.prop_cycle' rcParam.

Parameters
----------
x, y : array-like or scalar
    The horizontal / vertical coordinates of the data points.
    x values are optional. If not given, they default to
    [0, ..., N-1].

    Commonly, these parameters are arrays of length N. However,
    scalars are supported as well (equivalent to an array with
    constant value).

    The parameters can also be 2-dimensional. Then, the columns
    represent separate data sets.

fmt : str, optional
    A format string, e.g. 'ro' for red circles. See the Notes
    section for a full description of the format strings.

    Format strings are just an abbreviation for quickly setting
    basic line properties. All of these and more can also be
    controlled by keyword arguments.

data : indexable object, optional
    An object with labelled data. If given, provide the label names to
    plot in x and y.

    .. note::
        Technically there's a slight ambiguity in calls where the
        second label is a valid fmt. plot('n', 'o', data=obj)
        could be plt(x, y) or plt(y, fmt). In such cases,
        the former interpretation is chosen, but a warning is issued.
        You may suppress the warning by adding an empty format string
        plot('n', 'o', '', data=obj).


Other Parameters
----------------
scalex, scaley : bool, optional, default: True
    These parameters determined if the view limits are adapted to
    the data limits. The values are passed on to autoscale_view.

kwargs : `.Line2D` properties, optional
    *kwargs* are used to specify properties like a line label (for
    auto legends), linewidth, antialiasing, marker face color.
    Example::
```

```
        >>> plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2)
        >>> plot([1,2,3], [1,4,9], 'rs',  label='line 2')

        If you make multiple lines with one plot command, the kwargs
        apply to all those lines.

        Here is a list of available `.Line2D` properties:

          agg_filter: a filter function, which takes a (m, n, 3) float array␣
↪and a dpi value, and returns a (m, n, 3) array
      alpha: float (0.0 transparent through 1.0 opaque)
      animated: bool
      antialiased or aa: bool
      clip_box: a `.Bbox` instance
      clip_on: bool
      clip_path: [(`~matplotlib.path.Path`, `.Transform`) | `.Patch` | None]
      color or c: any matplotlib color
      contains: a callable function
      dash_capstyle: ['butt' | 'round' | 'projecting']
      dash_joinstyle: ['miter' | 'round' | 'bevel']
      dashes: sequence of on/off ink in points
      drawstyle: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-
↪post']
      figure: a `.Figure` instance
      fillstyle: ['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']
      gid: an id string
      label: object
      linestyle or ls: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-
↪off-dash-seq) | ``'-'`` | ``'--'`` | ``'-.'`` | ``':'`` | ``'None'`` | ``'␣
↪'`` | ``''``]
      linewidth or lw: float value in points
      marker: :mod:`A valid marker style <matplotlib.markers>`
      markeredgecolor or mec: any matplotlib color
      markeredgewidth or mew: float value in points
      markerfacecolor or mfc: any matplotlib color
      markerfacecoloralt or mfcalt: any matplotlib color
      markersize or ms: float
      markevery: [None | int | length-2 tuple of int | slice | list/array of␣
↪int | float | length-2 tuple of float]
      path_effects: `.AbstractPathEffect`
      picker: float distance in points or callable pick function ``fn(artist,␣
↪event)``
      pickradius: float distance in points
      rasterized: bool or None
      sketch_params: (scale: float, length: float, randomness: float)
      snap: bool or None
      solid_capstyle: ['butt' | 'round' |  'projecting']
      solid_joinstyle: ['miter' | 'round' | 'bevel']
      transform: a :class:`matplotlib.transforms.Transform` instance
      url: a url string
      visible: bool
      xdata: 1D array
      ydata: 1D array
      zorder: float
```

Returns
----
lines
A list of `.Line2D` objects representing the plotted data.

See Also
----
scatter : XY scatter plot with markers of varying size and/or color (
sometimes also called bubble chart).

Notes

****Format Strings**

A format string consists of a part for color, marker and line::

```
fmt = '[color][marker][line]'
```

Each of them is optional. If not provided, the value from the style
cycle is used. Exception: If line is given, but no marker,
the data will be a line without markers.

Colors

The following color abbreviations are supported:

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

If the color is the only part of the format string, you can
additionally use any *matplotlib.colors* spec, e.g. full names
('green') or hex strings ('#008000').

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker

```
    'v'          triangle_down marker
    '^'          triangle_up marker
    '<'          triangle_left marker
    '>'          triangle_right marker
    '1'          tri_down marker
    '2'          tri_up marker
    '3'          tri_left marker
    '4'          tri_right marker
    's'          square marker
    'p'          pentagon marker
    '*'          star marker
    'h'          hexagon1 marker
    'H'          hexagon2 marker
    '+'          plus marker
    'x'          x marker
    'D'          diamond marker
    'd'          thin_diamond marker
    '|'          vline marker
    '_'          hline marker
    =============    ==============================

    Line Styles

    =============    ==============================
    character        description
    =============    ==============================
    '-'          solid line style
    '--'         dashed line style
    '-.'         dash-dot line style
    ':'          dotted line style
    =============    ==============================

    Example format strings::

        'b'    # blue markers with default shape
        'ro'   # red circles
        'g-'   # green solid line
        '--'   # dashed line with default color
        'k^:'  # black triangle_up markers connected by a dotted line

    .. note::
        In addition to the above described arguments, this function can take a
        data keyword argument. If such a data argument is given, the
        following arguments are replaced by data[<arg>]:

        * All arguments with the following names: 'x', 'y'.
```

```
  File "<ipython-input-104-c3b0085c4838>", line 1
    cielo, la persona de mkt es la que tiene que ofrecer un listado de servicios y␣
↪productos:
                   ^
SyntaxError: invalid syntax
```

CHAPTER 31

Indices and tables

- genindex
- modindex
- search