SCRIPTs de la primera fase de análisis de CDRS: análisis comparativo

Desarrollado por: Javier Gómez Villamayor  
durante sus prácticas en Telefónica I+D

Dirigido por: Arturo Canales y Alberto Martín

## Intro

Los procesos que a continuación se detallan han sido desarrollados para **Python 2.7.5**. Responden a la motivación de analizar series de **archivos CDRS** (del inglés: call detail record o call data record; registros de datos de llamadas) relativos a un determinado país. En esta fase la intención era reflejar las diferencias entre el comportamiento registrado un día particular y el comportamiento esperado[[1]](#footnote-1) para dicho día.

Analizaremos los datos atendiendo al tiempo (momento en que se produce un registro CDRS) y a los operadores implicados (clientes telefónicos de salida y destino). De forma similar podría procederse también contemplando datos geográficos (en función de las antenas de envío y recepción); no hemos entrado aquí.

Los **registros CDRS** manejados son como los que siguen:

Ejemplo registro CALLS

380510028|34794006|380510028|55117523|1|LDN|20141231|00:00:42|10|8-TELEFONICA GUATEMALA|11-SERCOM (PCS DIGITAL)|||012940007517080|NOROAMI

Ejemplo registro SMS

52816504|54040001|1|20141231|00:00:52|8-TELEFONICA GUATEMALA|8-TELEFONICA GUATEMALA||380420283|NOROAMI

Ejemplo registro MMS

34630867|34630867|2|20141231|12:27:51|846-TELEFONICA GUATEMALA|846-TELEFONICA GUATEMALA|||

(se han resaltado los apartados para nosotros de interés). Cada registro constituye una línea de archivo CDRS. Para archivos CDRS con otra estructura no serán directamente útiles los scripts desarrollados, aunque puede presuponerse que su adaptación no sea complicada en exceso.

|  |  |
| --- | --- |
| Mapa del documento | (pág.) |
| [1) Resumen](#_1)_Resumen) | 2 |
| [2) Código](#_2)_Código) | 3 |
| [3) Ejemplo de uso](#_3)_Ejemplo_de) | 23 |

## [1) Resumen](#MapaDocumento)

Como primera aproximación a la biblioteca *AnalisisCDRS1*, a continuación se resume la función de cada uno de los procedimientos:

|  |
| --- |
| [**auxiliares**](#Auxiliares)  Contiene funciones auxiliares: norma1, normmax, piccolore.  [**analisisphpc(r, [tm, sw])**](#analisisphpc)  Desglosa por intervalo de tiempo y por operadores de salida y destino los registros leídos de un archivo cdrs.  Devuelve lo que llamaremos un tipo dgl (desglose): [SxE::keys, Criba::listasdedatos].  [**perfilmedio(dgls)**](#perfilmedio)  Construye el perfil medio como la media aritmética componente a componente de los perfiles dgl dados.  [**panoramica(dgls)**](#panoramica)  Construye un perfil conteniendo los subperfiles totales de los perfiles dgl dados.  [**acomparativo(dgl\_p, dgl\_m, [pan])**](#acomparativo)  Construye un perfil diferencia entre dos dados (dgl\_p – dgl\_m).  [**enproporcion(dgl\_a, dgl\_b)**](#enproporcion)  Construye un perfil cociente (total) entre dos dados (dgl\_a / dgl\_b).  [**pintar(dgl, opt)**](#pintar)  Ploteo de (un mapa de) gráficos para el análisis del perfil desglose dado. Engloba las funciones  pintar0, pintar1, …, pintar4, …, pintar7.  [**pintar1g(dgl, [n, s])**](#pintar1g)  Generalización de pintar1: admite n curvas destacadas y pintar perfil suelo.  [**pintar8(dgl, [re, s, leg, labelpp, labelpm])**](#pintar8)  Como pintar1 (para perfiles diferencia), enfocado a contrastar dos subperfiles. |

Las variables básicas que manejaremos durante el análisis son lo que hemos llamado estructuras dgl, **tipo desglose**, o **perfil**es. Fundamentalmente son el resumen de un archivo CDRS. Como estructura de datos, están constituidas como una lista (::list) de dos elementos

dgl = [dgl[0], dgl[1]] = [SxE, Criba]

siendo el primer elemento de nuevo una lista y el segundo elemento un diccionario (::dict). dgl[0] almacena las claves de entrada de dgl[1]; asociándose así para cada key (::tuple) en dgl[0] el **subperfil** dgl[1][key] (::list), que contendrá el número de operaciones en cada intervalo de tiempo (usualmente dgl[1][key] tendrá 24 entradas, una por hora). El primer elemento de dgl[0], dgl[0][0], corresponderá al subperfil total; el resto de elementos key serán pares de clientes de entrada y salida.

Nota: será común la confusión en el uso de los términos perfil y subperfil. Esto no es grave, resolviéndose a cuál nos referimos del contexto, aunque intentaremos diferenciar su uso cuando sea necesario. Adviértase también que un perfil puede contener a un único subperfil; diremos que está formado por un único subperfil.

## [2) Código](#MapaDocumento)

Pasamos ahora a detallar el código de cada script.

**[Auxiliares](#auxP)**

Contiene norma1() y normmax(), para el cálculo del peso (asociado a la norma correspondiente) de un subperfil; y piccolore(), una paleta que proporcione un color determinado a cada subperfil. Las tres son auxiliares de las funciones pintar\_().

|  |
| --- |
| def norma1(v): # suma elemento a elemento del vector (float)  s = 0 # si norma1=0 devuelve 1  for i in v:  s += abs(i)  if s != 0:  return float(s)  else:  return 1 |

|  |
| --- |
| def normmax(v): # máximo de los elementos del vector (float)  s = 0 # si normmax=0 devuelve 1  for i in v:  if abs(i) > s:  s = abs(i)  if s != 0:  return float(s)  else:  return 1 |

|  |
| --- |
| def piccolore(t, blue=False): # paleta de colores  if t <= 6 and not blue:  colores = ['red', 'green', 'yellow', 'magenta', 'cyan', 'black']  elif t <= 7 and blue:  colores = ['blue', 'red', 'green', 'yellow', 'magenta', 'cyan', 'black']  else:  colores = [(rd.random(), rd.random(), rd.random()) for k in range(t)]  while (1):  for co in colores:  yield co  # QUEDA AÚN POR DESARROLLAR DICCIONARIODECOLOR |

**[analisisphpc(f, [tm, sw])](#anaP)**

Genera el perfil desglose para un archivo CDRS.

Parámetros opcionales:

tm \_ para explicitar el intervalo de tiempo (en minutos); por defecto, 60 min.

sw \_ activa o desactiva la impresión del desglose por pantalla.

|  |
| --- |
| def analisisphpc(f, \*\*opt):  # Para la lectura de un archivo cdrs y el desglose de sus registros por horas y por clientes de salida y entrada  # Preparado para recibir como imput el nombre del archivo; típicamente '/ruta/tipooperacionaaaammdd.dat'  # siendo tipooperacion in {calls, sms, mms}  # Devuelve [SxE, criba], donde:  # SxE \_ una lista con todos los pares (clienteSalida, clienteEntrada) (tuplas), y cuya primera entrada corresponde al total del día  # criba \_ un diccionario relacionando cada elemento en SxE con su vector operaciones/hora (24 dimensional) asociado  # NOTA: a este tipo de estructura de salida la llamaremos en adelante tipo desglose  #  # Invocación típica:  # dglcalls141231 = analisisphpc('/disk0/home/bjgv/cdrs/calls20141231.dat')  # Admite generalización a desglose por minuto, por intervalo de 5min, 10min, 15min...  # dglcalls141231 = analisisphpc('/disk0/home/bjgv/cdrs/calls20141231.dat', tm = 5)  #  # Opciones: tm::integer \_ especifica subintervalo de tiempo  # sw::boolean \_ escritura de los resultados por pantalla  #CÓDIGO  if 'tm' not in opt:  frac = 60  elif opt['tm'] in {1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60}:  frac = opt['tm']  else:  return False  loc = f.split('/')[-1].split('.')[0] # nos quedamos con el nombre del archivo, sin su extensión  metodo = loc[:-8] # '/ruta/tipooperacionaaaammdd.dat' --> 'tipooperacionaaaammdd' --> 'tipooperacion'  fecha = loc[-8:] # --> 'aaaammdd'  SxE = [('TotalOP'.replace('OP', metodo), fecha)] # Primera entrada en SxE = (total, día)  horas = [0]\*(24\*60/frac) # Vector 24-dimensional (si frac=1) tipo; una posición por intervalo temporal  criba = {('TotalOP'.replace('OP', metodo), fecha): horas}  if metodo in {'sms', 'mms'}: #SMS&MMS  for g in open(f):  l = g.split('|') #registro típico: \_\_\_[...]\_\_\_|hh:mm:ss|clienteSalida|clienteEntrada|\_\_\_[...]\_\_\_  if (l[5], l[6]) not in SxE: # [4] [5] [6]  SxE.append((l[5], l[6]))  criba[(l[5], l[6])] = [0]\*(24\*60/frac)  h = l[4].split(':')  h = int(h[0])\*60/frac + int(h[1])/frac  horas[h] += 1  criba[(l[5], l[6])][h] += 1  elif metodo == 'calls': #CALLS  for g in open(f):  l = g.split('|') #registro típico: \_\_\_[...]\_\_\_|hh:mm:ss|\_\_\_\_|clienteSalida|clienteEntrada|\_\_\_[...]\_\_\_  if (l[9], l[10]) not in SxE: # [7] [9] [10]  SxE.append((l[9], l[10]))  criba[(l[9], l[10])] = [0]\*(24\*60/frac)  h = l[7].split(':')  h = int(h[0])\*60/frac + int(h[1])/frac  horas[h] += 1  criba[(l[9], l[10])][h] += 1  else:  return False    if 'sw' in opt:  if opt['sw']:  for d in SxE:  print "{} : {}".format(d, criba[d])  return [SxE, criba] |

**[perfilmedio(dgl1, dgl2, …)](#pmeP)**

Construye el perfil medio de los perfiles suministrados.

|  |
| --- |
| def perfilmedio(\*perfiles):  # Construye el perfil medio (de la forma natural, como la media aritmética componente a componente) de los perfiles proporcionados;  # restringido a que todos los perfiles correspondan al mismo tipo de operación (en caso contrario devuelve False). Si el  # conjunto de perfiles proporcionado es vacío retorna un perfil por horas (24-dimensional) nulo  # Imput: perfiles \_ un número indefinido de perfiles (con estructura tipo desglose: como genera 'analisisphpc(.)')  #  # Invocación típica:  # dglcallsMieMed = perfilmedio(dglcalls141217, dglcalls141210, dglcalls141126, dglcalls150114)  # CÓDIGO  if perfiles:  metodo = perfiles[0][0][0][0]  size = len(perfiles[0][1][perfiles[0][0][0]])  SxE = [(metodo, 'med')]  Criba = {(metodo, 'med'): [0]\*size}  for perfil in perfiles:  if perfil[0][0][0] == metodo and len(perfil[1][perfil[0][0]]) == size:  Criba[(metodo, 'med')] = [Criba[(metodo, 'med')][i]+perfil[1][perfil[0][0]][i] for i in range(0, size)]  for n in perfil[0][1:]:  if n not in SxE:  SxE.append(n)  Criba[n] = perfil[1][n]  else:  Criba[n] = [Criba[n][i]+perfil[1][n][i] for i in range(0, size)]  else:  return False  m = len(perfiles)  for k in Criba.keys():  Criba[k] = [Criba[k][i]/m for i in range(0, size)]  return [SxE, Criba]  else:  return [['cero'], {'cero': [0]\*24}] |

**[panoramica(dgl1, dgl2, …)](#panP)**

Construye un perfil formado por los subperfiles totales de los perfiles suministrados.

|  |
| --- |
| def panoramica(\*perfiles):  # Construye (como unión disjunta) un perfil constituido por todos los subperfiles totales de los distintos perfiles proporcionados.  # Si el conjunto de perfiles proporcionado es vacío retorna un perfil por horas (24-dimensional) nulo  # Imput: perfiles \_ un número indefinido de perfiles (con estructura tipo desglose: como genera 'analisisphpc(.)')  # Nótese que, puesto que sólo intervienen los subperfiles totales, es indiferente proporcionar  # un desglose completo o un desglose que sólo contenga al subperfil total para cada perfil  #  # Invocación típica:  # totalescallsMMNbNv = panoramica(dglcallsMieMed, dglcalls141231, dglcalls141224)  # CÓDIGO  if perfiles:  Totales = []  Criba = {}  i = 0  for perfil in perfiles:  Totales.append("{}. {}".format(i, perfil[0][0])) # para prevenir la posibilidad de que coincidan las claves, anteponemos  # un índice para conseguir la unión disjunta (de otra forma lo sobrescribiríamos)  Criba["{}. {}".format(i, perfil[0][0])] = perfil[1][perfil[0][0]]  i += 1  return [Totales, Criba]  else:  return [['cero'], {'cero': [0]\*24}] |

**[acomparativo(dgl\_p, dgl\_m, [pan])](#acomP)**

Construye el perfil diferencia entre los dos perfiles suministrados (el primero menos el segundo).

Parámetros opcionales:

pan \_ activa o desactiva la construcción del perfil diferencia completo: para pan=True únicamente se construye el subperfil total del perfil diferencia (por defecto pan=False).

|  |
| --- |
| def acomparativo(pp, pm, pan=False):  # Construye el perfil diferencia entre dos dados (típicamente, un perfil particular y un perfil medio previamente generado por el usuario)  # Si se explicita "pan=True", se limita a construir ciegamente la diferencia de los subperfiles totales; en otro caso  # construye el perfil diferencia completo (desglosado), bajo la restricción de tratarse de registros del mismo  # tipo de operación (en caso contrario devuelve False)  # Imput: pp, pm \_ perfilparticular, perfilmedio (ambos con estructura tipo desglose: como genera 'analisisphpc(.)')  # pan::boolean \_ (por defecto False) decide la forma de proceder del programa:  # False = genera perfil diferencia completo (desglosado)  # True = genera únicamente el perfil Total de diferencia  #  # Invocación típica:  # signcallsNvieja = acomparativo(dglcalls141231, dglcallsMieMed)  # signsmspcallsNvieja = acomparativo(dglsmspcallsNvieja, dglsmspcallsMieMed, pan=True)  #  # NOTA: Obsérvese el carácter antisimétrico de ambos parámetros (pp, pm) en el resultado (pp-pm)  # CÓDIGO  if pan:  size = len(pp[1][pp[0][0]])  c = [pp[1][pp[0][0]][i] - pm[1][pm[0][0]][i] for i in range(0, size)]  return [[(pp[0][0][0], '{}dif'.format(pp[0][0][1]))], {(pp[0][0][0], '{}dif'.format(pp[0][0][1])): c}]  metodo = pm[0][0][0] # comprobar mismo tipo sms/calls/mms  size = len(pp[1][pp[0][0]])  if pp[0][0][0] == metodo:  Criba = {(metodo, '{}dif'.format(pp[0][0][1])): [pp[1][pp[0][0]][i]-pm[1][pm[0][0]][i] for i in range(0, size)]}  for k in pp[0][1:]: # subperfiles del desglose en pp  Criba[k] = pp[1][k]  for k in set(pm[0][1:])-set(pp[0][1:]): # subperfiles no tratados en pp que estén presentes en pm  Criba[k] = [0]\*size  for k in pm[0][1:]: # diferencia  Criba[k] = [Criba[k][i]-pm[1][k][i] for i in range(0, size)]  SxE = Criba.keys() # cuidado no ordenado!!!! Para ser consistentes con el formato de desgloses, colocamos el índice del subperfil total en primer lugar  SxE.insert(0, SxE.pop(SxE.index((metodo, '{}dif'.format(pp[0][0][1])))))  return [SxE, Criba]  else:  return False |

**[enproporcion(dgl\_a, dgl\_b)](#nproP)**

Construye el perfil total cociente entre los dos perfiles suministrados (el (subperfil total) primero entre el (subperfil total) segundo).

Nota: sin desarrollar aún por completo. Línea de evolución: (A) precaución denominador nulo, en caso de anularse dgl\_b (en algún intervalo de tiempo i) definir el perfil cociente (en i) como normmax(dgl\_a); (B) ampliar enproporcion() también a la construcción del perfil cociente completo.

|  |
| --- |
| def enproporcion(a, b):  # Construye el perfil TOTAL cociente a/b (como es natural, componente a componente) de los perfiles proporcionados  # Imput: a, b \_ perfiles a comparar (con estructura tipo desglose, como la generada por 'analisisphpc(.)')  # Puesto que solo operamos con los subperfiles totales es indiferente proporcionar un desglose completo  # o un desglose que solo contenga al subperfil total  #  # Invocación típica:  # dglsmspcallsMieMed = enproporcion(dglsmsMieMed, dglcallsMieMed)  #  # NOTA: obsérvese el carácter inverso de ambos parámetros en el resultado (a/b)  size = len(a[1][a[0][0]])  if len(b[1][b[0][0]]) != size:  return False  else:  c = [float(a[1][a[0][0]][i])/b[1][b[0][0]][i] for i in range(0, size)]  return [[("{}/{}".format(a[0][0], b[0][0]), "rat")], {("{}/{}".format(a[0][0], b[0][0]), "rat"):c}] |

**[pintar(dgl, [opt])](#pinP)**

Engloba pintardesglose(), pintar1(), pintar2(), pintar3(), pintar4(), pintar4e(), pintar5(), pintar6(), pintar7() y pintar7e().

pintardesglose() dibuja un mapa de gráficos de análisis para el perfil suministrado. Cada uno de estos subgráficos puede generarse separadamente con el resto de funciones.

pintar1() dibuja llanamente el perfil suministrado.

pintar2(), pintar3() y pintar4[e]() lo grafican escalando cada subperfil en proporción a su norma1. Además mientras pintar2() muestra todos los subperfiles, pintar3() los degrada por nivel de trasparencia en función a su porcentaje de peso respecto del subperfil total, y pintar4() los criba mostrando únicamente aquellos que superen un determinado umbral de peso (respecto de la norma1 (total de operaciones)).

pintar5(), pintar6() y pintar7[e]() se comportan de forma análoga, actuando respecto al peso asociado a normmax (máximo por intervalo de tiempo).

En pintar() se unifican todas las funciones anteriores, pudiendo invocarlas (especificando su índice: 0,1,..7) en una sola orden. Por ejemplo, algunos ejemplos de llamada serían:

pintar(dgl) == pintar(dgl, 0) == pintardesglose(dgl)

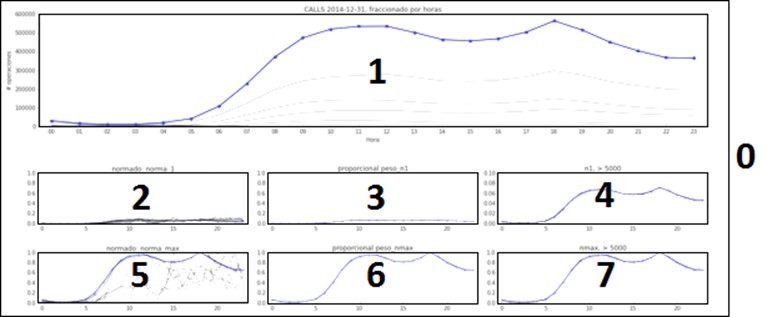
pintar(dgl,1,4,6) == pintar1(dgl); pintar4(dgl); pintar6(dgl)

pintar(dgl, (4), (7, ‘25p’)) == pintar4(dgl); pintar7(dgl, 0.25)

pintar(dgl, 1, (4, 1000, 2000)) == pintar(dgl, 1, (4, 1000), (4, 2000))

Note que las funciones pintar4(dgl, [cota]) y pintar 7(dgl, [cota]) admiten sus cotas como segundo argumento (pudiendo especificarse de manera absoluta (como un entero ≥ 1)) o de manera relativa (como porcentaje).

Esquema: Tipos de gráficos pintar\_



1. - Vista global
2. - Perfiles
3. - Perfiles (escalado respecto a su total de llamadas (en adelante norma1 o N1))
4. - Perfiles (escalado N1, nivel de trasparencia proporcional a N1Total)
5. - Perfiles con más de \_ llamadas totales [usualmente N1>5000] (escalado N1)
6. - Perfiles (escalado respecto a su máximo en una hora (en adelante normmáx o Nmáx))
7. - Perfiles (escalado Nmáx, nivel de trasparencia proporcional a NmáxTotal)
8. - Perfiles con más de \_ llamadas en alguna hora [usualmente Nmáx>5000] (escalado Nmáx)

NOTA: Se resalta en cada una el perfil total del día [al valor de cuyas normas hacen referencia N1Total y NmáxTotal] en azul grueso, siendo el resto de curvas correspondientes al perfil de un par (servidorSalida, servidorEntrada). Obsérvese que en 1 siempre debe quedar el perfil total superando al resto, mientras que en el resto de representaciones no hay error en superposiciones con este: se refleja del perfil de comportamiento (respecto a su total de llamadas o a su máximo de llamadas por hora) comparativamente.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| import matplotlib.pyplot as plt # plot EN PINTAR\_\_\_  import random as rd # EN PICCOLORE  import re # regular expression EN PINTAR4, ... para buscar p o %   |  | | --- | | def pintardesglose(c):  # Pintar\_ general. Dibuja un mapa de los gráficos que sirven al análisis; fundamentalmente tres:  # (a) Representación fiel  # (b) Escalado por el número total de llamadas (norm1)  # (c) Escalado en función al número máximo de llamadas por hora (normmáx)  # Imput: c \_ perfil con estructura tipo desglose (como la generada por 'analisisphpc(.)')  # Output: salida gráfica tipo 0 \_ mapa de gráficos asociados al perfil proporcionado  # -------------------------  # -- 1 -- Nivel (a): tipo1  # -------------------------  # - 2 - 3 - 4 - Nivel (b): tipo2 (todos los perfiles), tipo3 (todos, transparencia proporcional al peso (n1)),  # ------------------------- tipo4 (perfiles con más de 5000 llamadas totales)  # - 5 - 6 - 7 - Nivel (c): tipo5 (todos los perfiles), tipo4 (todos, transparencia proporcional al peso (nmáx)),  # ------------------------- tipo4 (perfiles con más de 1000 llamadas en alguna hora)  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15', '16',  '17', '18', '19', '20', '21', '22', '23', '24']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]    fig = plt.figure(figsize=(18, 8), dpi=80)  plt.subplot(2, 1, 1)  plt.plot(xx, c[1][c[0][0]], 'o-', linewidth=2.5, color='blue', label=c[0][0])  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)  for k in (set(c[0])-{c[0][0]}):  plt.plot(xx, c[1][k], '-', linewidth=1.0, color='black', label=k)  plt.xlabel('Hora')  plt.ylabel('# operaciones')  plt.title(u'OP, fraccionado por horas'.replace('OP', ('%s %s-%s-%s' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8]))))    plt.subplot(4, 3, 7)  peson1 = {i: 0 for i in c[0]}  peson1[c[0][0]] = norma1(c[1][c[0][0]])  yy = [i/peson1[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, '+-', linewidth=1.5, color='blue')  plt.xlim(rr[0]-1, rr[24])  plt.xticks([rr[6\*i]-0.5 for i in range(5)], [reloj[6\*i] for i in range(5)])  for k in set(c[0])-{c[0][0]}:  peson1[k] = norma1(c[1][k])  yy = [i/peson1[k] for i in c[1][k]]  plt.plot(xx, yy, '-', linewidth=0.5, color='black')  plt.title(u'normado: norma\_1')    plt.subplot(4, 3, 10)  pesonm = {i: 0 for i in c[0]}  pesonm[c[0][0]] = normmax(c[1][c[0][0]])  yy = [i/pesonm[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, '+-', linewidth=1.5, color='blue')  plt.xlim(rr[0]-1, rr[24])  plt.xticks([rr[6\*i]-0.5 for i in range(5)], [reloj[6\*i] for i in range(5)])  for k in set(c[0])-{c[0][0]}:  pesonm[k] = normmax(c[1][k])  yy = [i/pesonm[k] for i in c[1][k]]  plt.plot(xx, yy, '-', linewidth=0.5, color='black')  plt.title(u'normado: norma\_max')    plt.subplot(4, 3, 8)  yy = [i/peson1[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, '+-', linewidth=1.5, color='blue')  plt.xlim(rr[0]-1, rr[24])  plt.xticks([rr[6\*i]-0.5 for i in range(5)], [reloj[6\*i] for i in range(5)])  s1 = norma1([peson1[k] for k in set(c[0])-{c[0][0]}])  luz1 = {k: peson1[k]/s1 for k in set(c[0])-{c[0][0]}}  for k in set(c[0])-{c[0][0]}:  yy = [i/peson1[k] for i in c[1][k]]  plt.plot(xx, yy, '-', linewidth=0.5, color='black', alpha=luz1[k])  plt.title(u'proporcional peso\_n1')    plt.subplot(4, 3, 11)  yy = [i/pesonm[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, '+-', linewidth=1.5, color='blue')  plt.xlim(rr[0]-1, rr[24])  plt.xticks([rr[6\*i]-0.5 for i in range(5)], [reloj[6\*i] for i in range(5)])  sm = normmax([pesonm[k] for k in set(c[0])-{c[0][0]}])  luzm = {k: pesonm[k]/sm for k in set(c[0])-{c[0][0]}}  for k in set(c[0])-{c[0][0]}:  yy = [i/pesonm[k] for i in c[1][k]]  plt.plot(xx, yy, '-', linewidth=0.5, color='black', alpha=luzm[k])  plt.title(u'proporcional peso\_nmax')    plt.subplot(4, 3, 9)  cota = 5000  yy = [i/peson1[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, '+-', linewidth=1.5, color='blue')  plt.xlim(rr[0]-1, rr[24])  plt.xticks([rr[6\*i]-0.5 for i in range(5)], [reloj[6\*i] for i in range(5)])  for k in [k for k in set(c[0])-{c[0][0]} if peson1[k] >= cota]:  yy = [i/peson1[k] for i in c[1][k]]  plt.plot(xx, yy, '-', linewidth=0.5, color='black')  plt.title(u'n1, > %s' % cota)    plt.subplot(4, 3, 12)  cota = 1000  yy = [i/pesonm[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, '+-', linewidth=1.5, color='blue')  plt.xlim(rr[0]-1, rr[24])  plt.xticks([rr[6\*i]-0.5 for i in range(5)], [reloj[6\*i] for i in range(5)])  for k in [k for k in set(c[0])-{c[0][0]} if pesonm[k] >= cota]:  yy = [i/pesonm[k] for i in c[1][k]]  plt.plot(xx, yy, '-', linewidth=0.5, color='black')  plt.title(u'nmax, > %s' % cota)    plt.tight\_layout() | | def pintar1(c):  # Pintar\_ tipo1. Representación fiel (sin escalar) del perfil recibido  #  # Imput: c \_ perfil con estructura tipo desglose (como la generada por 'analisisphpc(.)')  # Output: salida gráfica tipo (a)1  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '07:00', '08:00',  '09:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',  '18:00', '19:00', '20:00', '21:00', '22:00', '23:00', '24:00']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]  paleta = piccolore(len(c[0]))  fig = plt.figure(figsize=(18, 8), dpi=80)  plt.plot(xx, c[1][c[0][0]], 'o-', linewidth=2.5, color='blue', label=c[0][0])  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)  for k in (set(c[0])-{c[0][0]}):  plt.plot(xx, c[1][k], '+-', linewidth=1.0, color=paleta.next(), label=k)  plt.legend(loc=2, fontsize=8)  plt.xlabel('Hora')  plt.ylabel('# operaciones')  plt.title(u'OP, fraccionado por horas'.replace('OP', ('%s %s-%s-%s' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8])))) | | def pintar2(c):  # Pintar\_ tipo2. Representación del perfil recibido normando cada curva de acuerdo a su número de llamadas totales (norma1).  #  # Imput: c \_ perfil con estructura tipo desglose (como la generada por 'analisisphpc(.)')  # Output: salida gráfica tipo (b)2  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '07:00', '08:00',  '09:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',  '18:00', '19:00', '20:00', '21:00', '22:00', '23:00', '24:00']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]  paleta = piccolore(len(c[0]))    fig = plt.figure(figsize=(18, 8), dpi=80)    peson1 = {i: 0 for i in c[0]}  for k in c[0]:  peson1[k] = norma1(c[1][k])  yy = [i/peson1[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, 'o-', linewidth=2.5, color='blue', label=c[0][0])  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)  for k in set(c[0])-{c[0][0]}:  yy = [i/peson1[k] for i in c[1][k]]  plt.plot(xx, yy, '+-', linewidth=1.0, color=paleta.next(), label=k)  plt.legend(loc=2, fontsize=9)  plt.xlabel('Hora')  plt.ylabel(u'Proporción de operaciones sobre el total del operador')  plt.title(u'OP, fraccionado por horas; escalado según norma \  uno'.replace('OP', ('%s %s-%s-%s' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8])))) | | def pintar3(c):  # Pintar\_ tipo3. Representación del perfil recibido normando cada curva de acuerdo a su norma1 (número total de llamadas).  # Cada curva se perfila con un nivel de trasnparencia proporcionado a su peso en dicha norma (i.e. a  # la cantidad de llamadas en el perfil respecto al perfil total)  #  # Imput: c \_ perfil con estructura tipo desglose (como la generada por 'analisisphpc(.)')  # Output: salida gráfica tipo (b)3  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '07:00', '08:00',  '09:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',  '18:00', '19:00', '20:00', '21:00', '22:00', '23:00', '24:00']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]  paleta = piccolore(len(c[0]))    fig = plt.figure(figsize=(18, 8), dpi=80)    peson1 = {i: 0 for i in c[0]}  for k in c[0]:  peson1[k] = norma1(c[1][k])  yy = [i/peson1[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, 'o-', linewidth=2.5, color='blue', label=c[0][0])  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)    s1 = norma1([peson1[k] for k in set(c[0])-{c[0][0]}])  luz1 = {k: peson1[k]/s1 for k in set(c[0])-{c[0][0]}}  for k in [k for k in set(c[0])-{c[0][0]} if luz1[k] < 0.07]:  luz1[k] = 0.07    for k in set(c[0])-{c[0][0]}:  yy = [i/peson1[k] for i in c[1][k]]  plt.plot(xx, yy, '+-', linewidth=1.0, color=paleta.next(), alpha=luz1[k], label=k)  plt.legend(loc=2, fontsize=9)  plt.xlabel('Hora')  plt.ylabel(u'Proporción de operaciones sobre el total del operador')  plt.title(u'OP, fraccionado por horas; e\_N1 (intensidad proporcional al peso \  del operador para la norma uno)'.replace('OP', ('%s %s-%s-%s' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8])))) | | def pintar5(c):  # Pintar\_ tipo5. Representación del perfil recibido normando cada curva en función a su número máximo de llamadas en  # alguna hora (normamáx).  #  # Imput: c \_ perfil con estructura tipo desglose (como la generada por 'analisisphpc(.)')  # Output: salida gráfica tipo (c)5  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '07:00', '08:00',  '09:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',  '18:00', '19:00', '20:00', '21:00', '22:00', '23:00', '24:00']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]  paleta = piccolore(len(c[0]))    fig = plt.figure(figsize=(18, 8), dpi=80)    pesonm = {i: 0 for i in c[0]}  for k in c[0]:  pesonm[k] = normmax(c[1][k])  yy = [i/pesonm[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, 'o-', linewidth=2.5, color='blue', label=c[0][0])  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)  for k in set(c[0])-{c[0][0]}:  yy = [i/pesonm[k] for i in c[1][k]]  plt.plot(xx, yy, '+-', linewidth=1.0, color=paleta.next(), label=k)  plt.legend(loc=2, fontsize=9)  plt.xlabel('Hora')  plt.ylabel(u'Proporción de operaciones sobre la máxima por hora del operador')  plt.title(u'SMS, fraccionado por horas; ')  plt.title(u'OP, fraccionado por horas; escalado según norma del \  máximo'.replace('OP', ('%s %s-%s-%s' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8])))) | | def pintar6(c):  # Pintar\_ tipo6. Representación del perfil recibido normando cada curva por normamáx (número máximo de llamadas en  # una hora). Cada curva se perfila con un nivel de trasnparencia proporcionado a su peso para  # dicha norma (i.e. máxporhora del perfil respecto al máxporhora del perfil total)  #  # Imput: c \_ perfil con estructura tipo desglose (como la generada por 'analisisphpc(.)')  # Output: salida gráfica tipo (c)6  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '07:00', '08:00',  '09:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',  '18:00', '19:00', '20:00', '21:00', '22:00', '23:00', '24:00']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]  paleta = piccolore(len(c[0]))    fig = plt.figure(figsize=(18, 8), dpi=80)    pesonm = {i: 0 for i in c[0]}  for k in c[0]:  pesonm[k] = normmax(c[1][k])  yy = [i/pesonm[c[0][0]] for i in c[1][c[0][0]]]  plt.plot(xx, yy, 'o-', linewidth=2.5, color='blue', label=c[0][0])  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)    sm = normmax([pesonm[k] for k in set(c[0])-{c[0][0]}])  luzm = {k: pesonm[k]/sm for k in set(c[0])-{c[0][0]}}  for k in [k for k in set(c[0])-{c[0][0]} if luzm[k] < 0.07]:  luzm[k] = 0.07  for k in set(c[0])-{c[0][0]}:  yy = [i/pesonm[k] for i in c[1][k]]  plt.plot(xx, yy, '+-', linewidth=1.0, color=paleta.next(), alpha=luzm[k], label=k)  plt.legend(loc=2, fontsize=9)  plt.xlabel('Hora')  plt.ylabel(u'Proporción de operaciones sobre la máxima por hora del operador')  plt.title(u'SMS, fraccionado por horas; ')  plt.title(u'OP, fraccionado por horas; e\_NM (intensidad proporcional al peso del operador para \  la norma del máximo)'.replace('OP', ('%s %s-%s-%s' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8])))) | | def pintar4(c, cota=5000):  # Pintar\_ tipo3. Representación (de un cribado) del perfil recibido normando cada curva de acuerdo a su norma1  # (número total de llamadas). Sólo se grafican aquellos perfiles con norma1 >= cota  #  # Imput: c::dgl \_ perfil con estructura tipo desglose (como la generada por 'analisisphpc(.)')  # cota::"numeric" \_ establece el número total de llamadas mínimo para que un perfil se represente  # (5000 salvo que se especifique otra cosa). Admite valores "absolutos" (enteros >=1)  # o "relativos" (que se entenderán como proporción del total); estos últimos pueden ser  # especificados como coeficiente (0 <= nn < 1) o como string ('nn%' = 'nnp').  # Output: salida gráfica tipo (b)3  #  # Invocación típica:  # pintar4(dglcalls141231) (equiv. pintar4(dglcalls141231, cota=5000))  # pintar4(dglcalls141231, 0.012) (equiv. pintar4(dglcalls141231, cota='1.2%'))    if type(c) == tuple and len(c) == 2:  cota = c[1]  c = c[0]  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '07:00', '08:00',  '09:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',  '18:00', '19:00', '20:00', '21:00', '22:00', '23:00', '24:00']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]  paleta = piccolore(len(c[0])) # POSIBLEMEJORA: crear paleta en función del número real de curvas pintadas  # POSIBLEMEJORA2: crear paleta como diccionario para mantener colores fijos en cada par SxE  # ¿paleta::dic global, y se van añadiendo colores que no estén?    fig = plt.figure(figsize=(18, 8), dpi=80)    peson1 = {i: 0 for i in c[0]}  for k in c[0]:  peson1[k] = norma1(c[1][k])  yy = [i/peson1[c[0][0]] for i in c[1][c[0][0]]]  if peson1[c[0][0]] > 1000:  leyend = u"{} : {:,.0f}k (total)".format(c[0][0], peson1[c[0][0]]/1000)  else:  leyend = u"{} : {:,.0f} (total)".format(c[0][0], peson1[c[0][0]])  plt.plot(xx, yy, 'o-', linewidth=2.5, color='blue', label=leyend)  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)    if re.search('p|%', str(cota)): # permite entrar cota como string(numporcent[p%])  cota = float(str(cota)[:-1])  cota = int(peson1[c[0][0]]\*cota/100)  elif cota < 1: # en otro caso (presuponemos numérico), se entiende 0.xx como porcentaje (xx/100)  cota = int(peson1[c[0][0]]\*cota)    for k in [k for k in set(c[0])-{c[0][0]} if peson1[k] >= cota]:  yy = [i/peson1[k] for i in c[1][k]]  if peson1[k] > 1000:  leyend = u"{} : {:,.0f}k".format(k, peson1[k]/1000)  else:  leyend = u"{} : {:,.0f}".format(k, peson1[k])  plt.plot(xx, yy, '+-', linewidth=1.0, color=paleta.next(), label=leyend)  plt.legend(loc=2, fontsize=9)  plt.xlabel('Hora')  plt.ylabel(u'Proporción de operaciones sobre el total del operador')  plt.title(u'SMS, fraccionado por horas; ')  titulo = u'OP, fraccionado por horas; e\_N1 (operadores con más de %s operaciones)' % cota  titulo = titulo.replace('OP', ('%s %s-%s-%s' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8])))  plt.title(titulo) | | def pintar4e(c, \*mini):  # pintar4 extendido. Generaliza y relaja la invocación de pintar4  # Invocación típica:  # pintar4e(dglcalls141231, 5000, 0.12, 0.012)    if type(c) == tuple and len(c) >= 2: # pintar4'((dgl, m1, m2)) = pintar4'(dgl, m1, m2) (para estúpidos)  mini = c[1:]  c = c[0]    if type(mini) == tuple and len(mini) == 1 and type(mini[0]) == tuple:  mini = [k for k in mini[0]] # pintar4'(dgl, ((m1, m2))) = pintar4'(dgl, m1, m2)  if len(mini) != 0: # pintar4'(dgl, m1, m2, ...) = pintar4(dgl, m1); pintar4(dgl,m2); ...  for k in [k for k in mini]:  pintar4(c, k)  else:  pintar4(c) | | def pintar7(c, cota=1000):  # Pintar\_ tipo7. Representación (de un cribado) del perfil recibido normando cada curva por normamáx (número máximo  # de llamadas en una hora). Sólo se grafican aquellos perfiles con normamáx >= cota  #  # Imput: c::dgl \_ perfil con estructura tipo desglose (como la generada por 'analisisphpc(.)')  # cota::"numeric" \_ establece el valor mínimo del máximo de operaciones por hora del perfil para que  # éste se represente (1000 salvo que se especifique otra cosa). Admite valores "absolutos"  # (enteros >=1) o "relativos" (que se entenderán como proporción del máximo por horas del  # perfil total), especificados como coeficiente (0 <= nn < 1) o como string ('nn%' = 'nnp').  # Output: salida gráfica tipo (c)7  #  # Invocación típica:  # pintar7(dglcalls141231) (equiv. pintar7(dglcalls141231, 5000))  # pintar7(dglcalls141231, cota=0.012) (equiv. pintar7(dglcalls141231, '1.2%'))  if type(c) == tuple and len(c) == 2:  cota = c[1]  c = c[0]  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '07:00', '08:00',  '09:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',  '18:00', '19:00', '20:00', '21:00', '22:00', '23:00', '24:00']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]  paleta = piccolore(len(c[0])) # POSIBLEMEJORA: crear paleta en función del número real de curvas pintadas  # POSIBLEMEJORA2: crear paleta como diccionario para mantener colores fijos en cada par SxE  # ¿paleta::dic global, y se van añadiendo colores que no estén?  fig = plt.figure(figsize=(18, 8), dpi=80)  pesonm = {i: 0 for i in c[0]}  for k in c[0]:  pesonm[k] = normmax(c[1][k])  yy = [i/pesonm[c[0][0]] for i in c[1][c[0][0]]]  if pesonm[c[0][0]] > 1000:  leyend = u"{} : {:,.0f}k (máx)".format(c[0][0], pesonm[c[0][0]]/1000)  else:  leyend = u"{} : {:,.0f} (máx)".format(c[0][0], pesonm[c[0][0]])  plt.plot(xx, yy, 'o-', linewidth=2.5, color='blue', label=leyend)  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)  if re.search('p|%', str(cota)): # permite entrar cota como string(numporcent[p%])  cota = float(str(cota)[:-1])  cota = int(pesonm[c[0][0]]\*cota/100)  elif cota < 1: # en otro caso (presuponemos numérico), se entiende 0.xx como porcentaje (xx/100)  cota = int(pesonm[c[0][0]]\*cota)  for k in [k for k in set(c[0])-{c[0][0]} if pesonm[k] >= cota]:  yy = [i/pesonm[k] for i in c[1][k]]  if pesonm[k] > 1000:  leyend = u"{} : {:,.0f}k".format(k, pesonm[k]/1000)  else:  leyend = u"{} : {:,.0f}".format(k, pesonm[k])  plt.plot(xx, yy, '+-', linewidth=1.0, color=paleta.next(), label=leyend)  plt.legend(loc=2, fontsize=9)  plt.xlabel('Hora')  plt.ylabel(u'Proporción de operaciones sobre la máxima por hora del operador')  plt.title(u'SMS, fraccionado por horas; ')  titulo = u'OP, fraccionado por horas; e\_NM (operadores con más de %s operaciones en alguna hora)' % cota  titulo = titulo.replace('OP', ('%s %s-%s-%s' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8])))  plt.title(titulo) | | def pintar7e(c, \*mini):  # pintar7 extendido. Generaliza y relaja la invocación de pintar7  # Invocación típica:  # pintar7e(dglcalls141231, 5000, 0.12, 0.012)    if type(c) == tuple and len(c) >= 2: # pintar7'((dgl, m1, m2)) = pintar7'(dgl, m1, m2) (para estúpidos)  mini = c[1:]  c = c[0]    if type(mini) == tuple and len(mini) == 1: # pintar7'(dgl, ((m1, m2))) = pintar7'(dgl, m1, m2)  mini = [k for k in mini[0]]  if len(mini) != 0: # pintar7'(dgl, m1, m2, ...) = pintar7(dgl, m1); pintar7(dgl,m2); ...  for k in [k for k in mini]:  pintar7(c, k)  else:  pintar7(c) | | def pintar(c, \*m):  # Unifica el grupo de scripts pintar\_ asociados con pintardesglose  # Cada tipo de gráfico (0, 1, 2, .., 7) se especifica por su índice  # Los gráficos tipo4 y tipo7 admiten también sus cotas como segundo argumento  # OK versión3: adaptados a cualquier tamaño de perfil  #  # Invocaciones típicas:  # pintar(dgl) == pintar(dgl, 0) == pintardesglose(dgl)  # pintar(dgl,1,4,6) == pintar1(dgl); pintar4(dgl); pintar6(dgl)  # pintar(dgl, (4), (7, 0.25)) == pintar4(dgl); pintar7(dgl, 0.25)  # pintar(dgl, 1, (4, 1000, 2000)) == pintar(dgl, 1, (4, 1000), (4, 2000))  posibles = {0: pintardesglose, 1: pintar1, 2: pintar2, 3: pintar3, 4: pintar4, 5: pintar5, 6: pintar6, 7: pintar7}  posiblese = {4: pintar4e, 7: pintar7e}  if len(m) != 0:  for n in m:  try:  posibles[n](c)  except:  if type(n) == tuple and len(n) != 0:  if len(n) == 1:  posibles[n[0]](c)  else:  posiblese[n[0]](c, n[1:])  else:  print ""  else:  pintardesglose(c) | |

**[pintar1g(dgl, [n, s])](#pin1P)**

Grafica, como pintar1(), llanamente el perfil suministrado.

Como opciones adicionales permite:

n \_ explicitar el número de curvas destacadas (por defecto n=1)

s \_ (por defecto False) para s=True añade al gráfico la curva suelo (correspondiente al subperfil idénticamente nulo)

|  |
| --- |
| def pintar1g(c, n=1, s=False):  # Opciones: n::integer \_ número de curvas destacadas (por defecto 1)  # s::boolean \_ curva suelo/cero (por defecto False)  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '07:00', '08:00',  '09:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',  '18:00', '19:00', '20:00', '21:00', '22:00', '23:00', '24:00']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]  paleta = piccolore(len(c[0]), blue=True)  fig = plt.figure(figsize=(18, 8), dpi=80)  if s:  plt.plot(xx, [0]\*size, 'o-', linewidth=2.5, color=paleta.next())  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)  if len(c[0]) < n:  n = len(c[0])  for i in range(n):  plt.plot(xx, c[1][c[0][i]], 'o-', linewidth=2.5, color=paleta.next(), label=c[0][i])  for k in (set(c[0])-{c[0][i] for i in range(n)}):  plt.plot(xx, c[1][k], '+-', linewidth=1.0, color=paleta.next(), label=k)  plt.legend(loc=2, fontsize=8)  plt.xlabel('Hora')  plt.ylabel('# operaciones')  plt.title(u'%s %s-%s-%s, fraccionado por horas' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8])) |

**[pintar8(dgl\_Tot, [re, s, leg, labelpp, labelpm])](#pin8P)**

Esta función está dirigida al graficado de perfiles de subperfil único. Grafica, como pintar1(), llanamente el subperfil suministrado, que se concibe como un perfil diferencia; por defecto se rellenará el espacio entre el subperfil y la curva suelo.

Parámetros opcionales:

re \_ activa o desactiva el rellenado de la curva.

s \_ (por defecto True) activa o desactiva el dibujo de la curva suelo.

leg \_ (por defecto False) activa o desactiva la leyenda del subperfil.

labelpp, labelpm \_ las etiquetas de los imaginarios perfiles diferencia.

|  |
| --- |
| def pintar8(c, re=True, s=True, labelpp='', labelpm='', leg=False):  # Opciones: labelpp::string \_ etiqueta perfil particular (el positivo)  # labelpm::string \_ etiqueta perfil medio (el minus, respecto al que se compara)  # s::boolean \_ curva suelo/cero (por defecto True)  # re::boolean \_ rellenado (por defecto True)  if len(c[0]) != 1:  return False  else:  size = len(c[1][c[0][0]])  xx = range(size)  rr = [i\*size/24 for i in range(25)]  reloj = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '07:00', '08:00',  '09:00', '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '17:00',  '18:00', '19:00', '20:00', '21:00', '22:00', '23:00', '24:00']  metodo = c[0][0][0][5:]  fecha = c[0][0][1]  fig = plt.figure(figsize=(18, 8), dpi=80)  if s:  plt.plot(xx, [0]\*size, 'o-', linewidth=2.5, color='black')  plt.xlim(rr[0]-1, rr[24])  plt.xticks([x-0.5 for x in rr], reloj)  plt.plot(xx, c[1][c[0][0]], 'o-', linewidth=2.5, color='red', label=c[0][0])  if leg:  plt.legend(loc=2, fontsize=8)  if re:  plt.fill\_between(xx, c[1][c[0][0]], [0]\*size, color='blue')  plt.xlabel('Hora')  plt.ylabel('# operaciones')  plt.title(u'%s %s-%s-%s, fraccionado por horas' % (metodo.upper(), fecha[0:4], fecha[4:6], fecha[6:8]))  fig.text(0.8, 0.8, labelpp) # Modificar parámetros si se requiere cambiar localización  fig.text(0.8, 0.04, labelpm) |

## [3) Ejemplo de uso](#MapaDocumento)

Por último, adjuntamos un ejemplo de uso para lo anterior:



[*“EjemplodeUso.html”*](https://www.dropbox.com/s/fa3mfzudt84ky4x/EjemplodeUso.html?dl=0)

## [4) A modo de *Fe de Erratas*](#_Intro)

En el script pintar1g se debió haber incluido, como en pintar8, la opción leg

Los títulos de gráfica… inapropiado formateo

Titulo incorrecto – corregir al menos “por horas”

1. Entiéndase esperado no en el sentido comportamiento más probable, si no como un marco de comportamiento fijado (e.g. media\_díaslaborables, media\_miércoles2014, media\_noviembre) [↑](#footnote-ref-1)