Sumari

A. Codi informàtic per al càlcul de la distribució de consums per usos finals en el sector dels edificis corporatius

A continuació s'observa el codi informàtic desenvolupat amb l'objectiu de poder extreure la informació dels edificis corporatius de la base de dades de DEXMA, amb dos objectius principals:

1. Explorar els subconsums dels edificis monitoritzats per determinar el percentatge mitjà d'ús energètic per cada tipus de servei (climatització, il·luminació, equipaments, etc,.)

2. Caracterització del patró anual i patró setmanal dels edificis corporatius.

El codi que s'exposa ha estat desenvolupat en llenguatge Python, llenguatge de programació d'alt nivell aconsellat per aquests tipus d'anàlisis estadístics exploratoris.

El resultat del codi informàtic és un fitxer EXCEL amb les dades necessàries per a complir els objectius descrits anteriorment.

Per evitar problemes de confidencialitat de dades, els noms d'usuari i mots de pas s'han modificat.

```
__author__ = 'dutges'

import psycopg2
from datetime import datetime
from dateutil.relativedelta import relativedelta
import xlwt

from dexma.dexcell import DexcellRestApi as Drapi

class staticalBasic():

    def __init__(self):
        self.conn = psycopg2.connect("dbname=***** user=***** password=*******
host=********")

    def get_active_inactives_locations(self):
        query=""" SELECT d.status, COUNT(l.id)
                  FROM dexh_deployment d, dexh_location l
                  WHERE l.deployment_fk = d.id
```

```
                GROUP BY d.status
            """
        cur = self.conn.cursor()
        cur.execute(query)
        result = cur.fetchall()
        cur.close()
        return result

    def get_count_type_locations(self, type_location):
        query = """
                SELECT COUNT(l.id)
                FROM dexh_deployment d, dexh_location l, dexh_location_info li
                WHERE d.status = 'ACTIVE'
                      and li.activity_fk = %s
                      and l.deployment_fk = d.id
                      and li.id = l.info_fk
                """
        data = (type_location, )
        cur = self.conn.cursor()
        cur.execute(query, data)
        result = cur.fetchall()
        cur.close()
        return result

    def get_submetering_type_location(self, type_location):
        query = """
                SELECT l.id,
                       l.name,
                       string_agg(rm.type, ', ') AS types,
                       array_to_string(ARRAY_AGG(rm.mote_fk),', ') AS motes,
                       li.surface
                FROM dexh_location l
                INNER JOIN dexh_deployment d ON l.deployment_fk = d.id
                INNER JOIN dexh_location_info li ON li.id = l.info_fk
                INNER JOIN dexh_reference_mote rm ON l.id = rm.location_fk
                INNER JOIN dexh_mote m ON m.id = rm.mote_fk
                WHERE d.status = 'ACTIVE'
                      and li.activity_fk = %s
                      and m.status = 'ACCEPTED'
                GROUP BY l.id, li.surface
                ORDER BY l.id
                """
        data = (type_location, )
        cur = self.conn.cursor()
        cur.execute(query, data)
        result = cur.fetchall()
        cur.close()
        return result

    def get_percentatge_submetering(self, type_location):
        locations = self.get_submetering_type_location(type_location)
        dict_types = {}
        for location in locations:
```

```
            types = location[2].split(", ")
            for type in types:
                if type not in dict_types:
                    dict_types.update({type: 0})
                dict_types[type] += 1
        return {key: round(100.0 * dict_types[key] / len(locations), 2) for key in
dict_types}

    def _get_total_readings(self, mote, s_nid, dx, init_date, last_date):
        try:
            readings = dx.get_readings(int(mote), s_nid, init_date, last_date)
        except Exception as e:
            readings = None
            #print e

        total = 0
        if type(readings) == list:
            for read in readings:
                total += read["value"]
        return total

    def _get_if_readings(self, motes, s_nid, types, result, dx, init_date, last_date):
        for pos, mote in enumerate(motes):
            readings = self._get_total_readings(mote, s_nid, dx, init_date, last_date)
            if readings > 0:
                if types[pos] not in result:
                    result.update({types[pos]: 0})
                result[types[pos]] += readings
        return result

    def _append_result(self, list_result, result):
        count = 0
        for key, value in result.items():
            if value != 0:
                count += 1
        if count > 1:
            list_result.append(result)
        return list_result

    def get_pie_subconsumptions(self, type_location, s_nid, dx):
        locations = self.get_submetering_type_location(type_location)
        last_date = datetime.now()
        init_date = last_date - relativedelta(years=1)
        list_result = []
        for location in locations:
            types = location[2].split(", ")
            motes = location[3].split(", ")
            surface = location[4]
            result = {"location": location[1], "surface": surface}
            result = self._get_if_readings(motes, s_nid, types, result, dx, init_date,
last_date)
            list_result = self._append_result(list_result, result)
        return list_result
```

```
##################################################

    def _get_monthly_readings(self, mote, s_nid, dx, init_date, last_date, spaces):
        try:
            readings = dx.get_readings(int(mote), s_nid, init_date, last_date)
        except Exception as e:
            readings = None
        months_result = {}
        if type(readings) == list:
            for read in readings:
                if spaces == 12:
                    pos = read["ts"].month - 1
                elif spaces == 168:
                    pos = read["ts"].weekday() * 24 + read["ts"].hour
                if pos not in months_result:
                    months_result[pos] = 0
                months_result[pos] += read["value"]
        return months_result

    def _get_monthly_if_readings(self, motes, s_nid, types, dx, init_date, last_date,
spaces):
        readings = None
        for pos, mote in enumerate(motes):
            if types[pos] == "MAINSUPPLY":
                readings = self._get_monthly_readings(mote, s_nid, dx, init_date, last_date,
spaces)
                break
        return readings

    def get_values_for_statistics(self, type_location, s_nid, rest_val, spaces, dx,
last_date=datetime.now()):
        locations = self.get_submetering_type_location(type_location)
        init_date = last_date - rest_val
        list_result = []
        count = 0
        for location in locations:
            if count == 15: break
            types = location[2].split(", ")
            motes = location[3].split(", ")
            result = self._get_monthly_if_readings(motes, s_nid, types, dx, init_date,
last_date, spaces)
            if result is not None:
                result["location"] = location[1]
                result["surface"] = location[4]
                list_result.append(result)
            count += 1
        return list_result

    def get_end_last_week(self):
        date = datetime.now()
        while date.weekday() != 0:
            date -= relativedelta(days=1)
```

```
        return date

    def get_medians_week(self, type_location, s_nid, spaces=168):
        locations = self.get_submetering_type_location(type_location)
        last_date = self.get_end_last_week()
        init_date = last_date - relativedelta(days=364)
        list_totals = []
        for location in locations:
            print "executing for location %s" % location[1]
            types = location[2].split(", ")
            motes = location[3].split(", ")
            total_result = []
            init_date = last_date - relativedelta(days=364)
            while init_date + relativedelta(days=7) < last_date:
                result = self._get_monthly_if_readings(motes, s_nid, types, dx, init_date,
last_date, spaces)
                if type(result) == dict and len(result) > 0:
                    total_result.append(result)
                init_date += relativedelta(days=7)
            total_reads = {}
            total_values = {}
            for result in total_result:
                for key, val in result.items():
                    if key not in total_reads:
                        total_reads[key] = 0
                        total_values[key] = 0
                    total_reads[key] += 1
                    total_values[key] += val
            totals = {}
            for key, val in total_values.items():
                totals[key] = val / total_reads[key]
            if len(totals) > 0:
                totals["location"] = location[1]
                totals["surface"] = location[4]
                list_totals.append(totals)
        return list_totals


    def get_end_some_week(self, month):
        date = datetime.strptime("20/%s/2014" % month, "%d/%m/%Y")
        while date.weekday() != 0:
            date -= relativedelta(days=1)
        return date

    def get_values_each_month(self, type_location):
        dict_values = {}
        for i in range(1, 13):
            print "month %s" % i
            last_date = self.get_end_some_week(i)
            dict_values[i] = self.get_values_for_statistics(type_location, 40211,
relativedelta(days=7), 168, dx, last_date=last_date)
        return dict_values
```

```
    def close_connection(self):
        self.conn.close()

    def create_header(self, ws, keys):
        ws.write(0, 0, unicode("buildings"))
        ws.write(0, 1, "surface")
        for pos, key in enumerate(keys):
            ws.write(0, pos + 2, unicode(key))

    def _inflate_body(self, ws, values, keys):
        for pos, value in enumerate(values):
            ws.write(pos + 1, 0, value["location"])
            ws.write(pos + 1, 1, value["surface"])
            for pos_y, key in enumerate(keys):
                if key in value:
                    ws.write(pos + 1, pos_y + 2, value[key])

    def convert_to_excel(self, results, name):
        wb = xlwt.Workbook(encoding='utf-8')
        ws = wb.add_sheet('stats Grader')
        keys = set()
        for result in results:
            new_keys = result.keys()
            keys = keys | set(new_keys)
        keys = list(keys)
        keys.remove("location")
        keys.remove("surface")
        self.create_header(ws, keys)
        self._inflate_body(ws, results, keys)
        wb.save('/home/dcortes/PycharmProjects/statical_calculation/%s.xls' % name)

endpoint="http://www.dexcell.com/api/v2/"
token="***********"
dx = Drapi(endpoint, token)
x = staticalBasic()
print x.get_active_inactives_locations() ## DONE
print x.get_count_type_locations("OFFICE-BUILDING") ## DONE
print x.get_percentatge_submetering("OFFICE-BUILDING") ## DONE
try:
    result = x.get_pie_subconsumptions("OFFICE-BUILDING", 40241, dx)
    x.convert_to_excel(result, u"pie_sub_consumptions")
except:
    print "problem with pie sub consumption"

try:
    result = x.get_values_for_statistics("OFFICE-BUILDING", 40241, relativedelta(years=1),
12, dx)
    x.convert_to_excel(result, u"montlhy_values")
except:
    print "problem with montlhy_values"

try:
    result = x.get_medians_week("OFFICE-BUILDING", 40211)
```

```
    x.convert_to_excel(result, u"hardcore_weekly_median")
except:
    print "problem with hardcore median"
#print x.get_values_each_month("OFFICE-BUILDING")
x.close_connection()
```

B. Comparativa del consum energètic de diversos sistemes de climatització

L'objectiu d'aquest annex és extendre les dades aconseguides mitjançant el programa de simulació de sistemes de climatització per tal d'evaluar diferents tecnologies de climatització i així conèixer quines es poden aplicar al client per a que redueixi el seu consum energètic.

En cap moment pretén ser una guia del programari utilitzatm ja que no és l'objectiu principal del projecte.

Les tres tecnologies de climatització que s'analitzen són les següents:

- A: Sistema de climatització per fan-coils (FC) i plantes refredadores condensades per aire (PRA)
- B: Sistema de climatització per VAV amb free-cooling (VAV + fc) i plantes refredadores condensades per aire (PRA)
- C: Sistema de climatització per VAV amb free-cooling (VAV + fc) i plantes refredadores condensades per aigua amb torres de refrigeració (PRWT)

Per a cada cas es realitza el càlcul de càrregues i la simulació tèrmica del edifici.

L'eina de càlcul utilitzada ha sigut el programa CARRIER hourly Analysis Program v 4.12b en la seva versió Free Trial (prova gratuïta), ja que és la única a la que s'ha pogut tenir accés.

Com a edifici de referència, s'ha escollit un edifici d'oficines situat a Barcelona, de superfície aproximada de 10.000m^2 i amb una superfície climatitzada de 8.200m^2. La façana del edifici és de vidre en un 40%. La Figura B.1 representa una planta tipus del edifici analitzat.

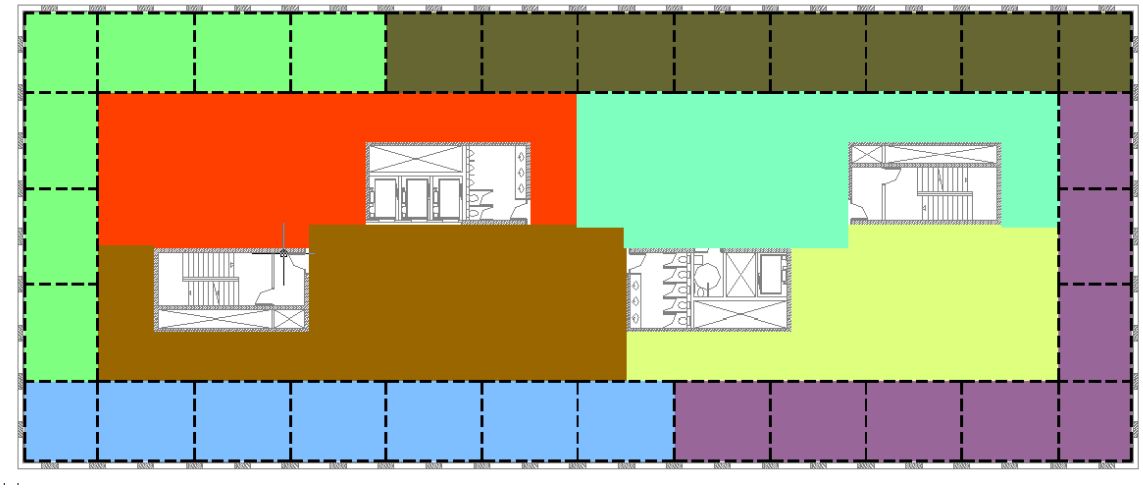

Figura B.1 – Planta del edifici estudiat, amb les àrees diàfanes i els compartiments d'oficines

A la Taula B.1 es mostren els consums energètics mensuals per cadascuna de les opcions un cop realitzades les simulacions.

Mes	Opció A (kWh)	Opció B (kWh)	Opció C (kWh)
Gener	37.244	21.876	7.860
Febrer	38.584	25.859	9.460
Març	44.691	32.036	11.060
Abril	49.500	39.086	15.550
Maig	58.209	56.653	23.717
Juny	66.483	69.401	30.045
Juliol	83.863	88.284	37.425
Agost	81.300	83.066	35.085
Setembre	70.938	74.342	30.380
Octubre	58.839	57.291	24.099
Novembre	39.805	31.038	11.514
Desembre	38.546	24.067	8.648
Total	**668.206**	**603.661**	**244.843**

Taula B.1 – Detall dels consums mensuals segons la opció de climatització

A la Figura B.2 s'observa el gràfic de consums mensuals simulats segons la tecnologia escollida. Resulta evident, doncs, que la opció C és la que té un consum menor. La Opció A i B són molt properes en consum, si bé el consum de la A és superior al B, sobretot en les èpoques fredes.

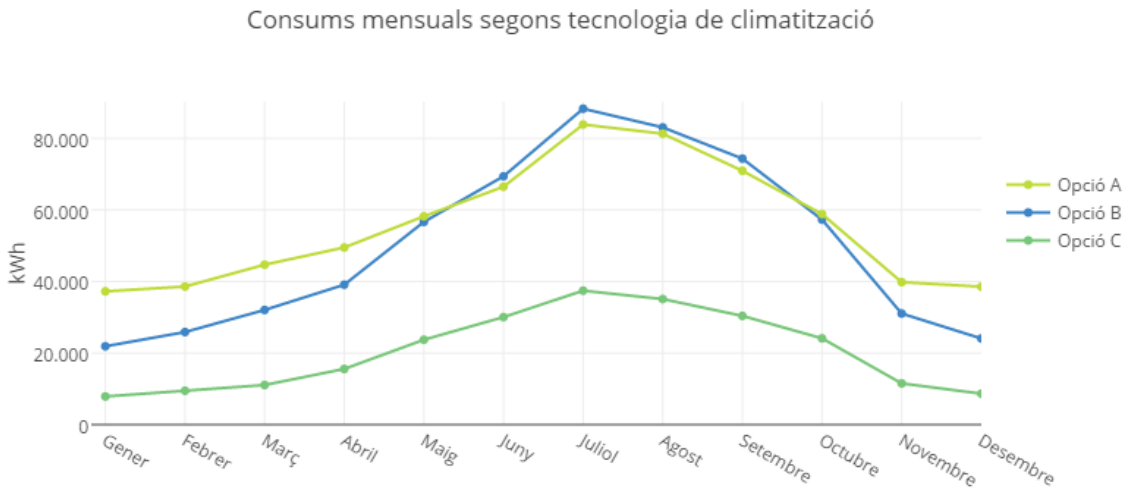

Figura B.2 – Simulació del consum mensual segons la opció de climatització

Amb el programari de Carrier s'ha calculat la demanda tèrmica necessària i el dimensionament del sistema segons cadascuna de les opcions per a que els usuaris del edifici realitzin la seva activitat en un rang de confort, donant com a resultats els valors mostrats en la Taula B.2.

Opció	Potència tèrmica clima (kWt)	Rati tèrmic (kWt/m^2)	Consum elèctric anual climatització (kWh)	% estalvi climatització sobre opció A
A	1.019,9	124,35	668.206	-
B	1.056,1	128,00	603.661	9,6%
C	1.056,1	128,00	244.843	63,4%

Taula B.2 – Dimensionat del sistema de climatització segons la opció triada.

Per aconseguir els costos associats s'ha contactat amb el departament comercial de Carrier Espanya mitjançant el telèfon 902 44 54 64 per a aconseguir un ordre de magnitud aproximat de preus segons la opció triada. Els preus que s'han pogut aconseguir són preus de venta al públic, tenint en compte el cost del material en un edifici de dimensions similars al descrit anteriorment. No es tenen presents els costos d'instal·lació ja que no s'ha pogut disposar de cap estimació.

El comercial de Carrier comenta en conversació telefónica mantinguda el 22 de setembre del 2016 que el preu de la Opció A, corresponent al sistema de clima per fan coils i plantes refredadores condensades per aire és de 25€/m^2.

El cost de la Opció B, corresponent al sistema de climatització per VAV amb free-cooling i plantes refredadores condensades per aire és de 30€/m^2.

Per últim, el cost de la opció C, la més eficient energèticament està al voltant dels 50€/m^2, però s'ha de tenir present el consum d'aigua com a conseqüència de disposar de torres de refrigeració condensades per aigua, a part d'un major cost de manteniment.

Per tant, s'ha de realitzar el càlcul del cost de l'aigua utilitzada en la refrigeració del edifici.

Tenint en compte que

$$Cost\ aigua\ (€) = Consum\ aigua\ (l) * preu\ aigua\ (€/l)$$

Equació B.1

On:

- Preu de l'aigua és igual a 0,0013€/l

Calculem el consum d'aigua amb l'Equació B.2:

$$Consum\ aigua\ (l) = (Cabal\ evaporació\ +\ Cabal\ arrossegament\ +\ Cabal\ purga) * t$$

Equació B.2

On:

- t és el temps de funcionament de les refredadores durant l'any. El programa de Carrier marca un valor de 1.440 h/any.

El cabal d'evaporació es calcula segons l'Equació B.3:

Equació B.3

$$Cabal\ Evaporació\ (l/h)\ =\ Potència\ (kcal/h)\ /\ Calor\ evaporació\ (kcal/kg\ H2O\ evap.)$$

On:

- Calor d'evaporació és de 500 kcal/kg H_2O evaporada.
- Potència és de 908.604 kcal/h (transformant 1.056 kW)

Per la seva banda, el cabal d'arrossegament es calcula segons l'Equació B.4:

$$Cabal\ arrossegament\ (l/h)\ =\ 0.2\%\ x\ Cabal\ aigua\ circulada\ (l/h)$$

Equació B.4

Finalment, el cabal de purga es calcula segons l'Equació B.5:

$$Cabal\ Purga\ (l/h)\ =\ (1(l)/1000\ (kcal))\ x\ Potència\ (kcal/h)$$

Equació B.5

On:

- Potència és de 908.604 kcal/h (transformant 1056 kW)

Un cop definides les fórmules es realitzen els càlculs per a la opció C, donant com a resultat un cost d'aigua de:

$$Cabal\ Evaporació\ =\ 908.604,2\ kcal/h\ /\ 500\ kcal/kg\ *\frac{1kg}{1l} = 1.817,2\ l/h$$

$$Cabal\ arrossegament\ (l/h)\ =\ 0.2\%\ x\ 1.817,2\ l/h\ = 363,44\ l/h$$

$$Cabal\ Purga\ (l/h)\ =\ (1(l)/1000\ (kcal))\ x\ 908.604,2\ kcal/h\ = 908,6\ l/h$$

$$Consum\ aigua\ (l) = (1.817,2\ l/h + \ 363,4\ l/h + \ 908,6\ l/h) * 1.440\ h = 4.448.448\ l/any$$

$$Cost\ aigua\ anual = 4.448.448\ l/any * 0,0013\ €/l\ = 5.782\ €/any$$

Per tant, la instal·lació del sistema de climatització descrit en la opció C requerirà de gairebé 6000€ addicionals de consum d'aigua, que equivalen a 0,6 €/m2 en l'edifici que s'està analitzant.

C. Implementació algoritme optimització de potència contractada

A continuació s'observa el codi que s'ha desenvolupat per a la millora d'optimització de potència contractada, la qual forma part de la llibreria de millores centrades en la reducció del preu de compra de l'energia.

Com s'explicita en la memòria, es realitza la segmentació segons la estratègia de penalització que marca el govern espanyol.

Les parts de codi que es mostren a continuació no es poden executar per elles mateixes, donat que son classes informàtiques del codi global de l'aplicació d'auditories virtuals.

C.1. Algoritme de 3 períodes – Regla 85-105%

Aquest algoritme es basa en la regla del 85-105% descrita en la memòria del projecte. A continuació la part del codi que s'ha desenvolupat amb l'objectiu d'oferir al client el mínim cost possible en el terme de potència contractada.

```
__author__ = 'dutges'

# coding=utf-8

from dateutil.relativedelta import relativedelta
#from data_utils.simulatedBill.wrapper_bill import wrapper_bill
from datetime import datetime, timedelta
from dexma.dexcell import DexcellRestApi as DRapi

from ..utils.formatter import Formatter
from data_utils.simulatedBill.wrapper_bill import wrapper_bill

formatNum = Formatter.formatNum


class algorithm_three_periods(object):

    def __init__(self, loc_id, data_init, data_fi, optLP, optIVA, optIEE, logger):
        self.loc_id = loc_id
        self.data_init = data_init
        self.data_fi = data_fi
        self.optLP = optLP
        self.optIVA = optIVA
        self.optIEE = optIEE
```

```
        self.dx = DRapi("http://www.dexcell.com/api/v2", "1c9004c6632e1bbbf2bd")
        self.dx25 = DRapi("http://www.dexcell.com/api/v25", "1c9004c6632e1bbbf2bd")
        self.totalDaysOcuped = (data_fi - data_init).days
        self.vector_days = []
        self.logger = logger
        self.periodos = ['P' + str(i + 1) for i in range(3)]

    def annualpowerCost(self, power, price):
        return power * price * 365

    def annualCostPenalty(self, power, maximeterArray, price, invoice_days):
        penaltyArray = []
        i = 0
        for dd in maximeterArray:
            if dd <= 0.85 * power:
                penalty = -0.15 * power * price * invoice_days[i]
            elif dd <= 1.05 * power:
                penalty = (dd - power) * price * invoice_days[i]
            elif dd > 1.05 * power:
                penalty = ((dd - power) + 2 * (dd - 1.05 * power)) * price * invoice_days[i]
            i = i + 1
            penaltyArray.append(penalty)
        return sum(penaltyArray)

    def facturedPower(self, maximeter, power):
        if maximeter <= 0.85 * power:
            return (0.85) * power
        elif maximeter <= 1.05 * power:
            return power
        else:
            return maximeter + 2 * (maximeter - 1.05 * power)

    def compara(self, x, y):
        if x['savings'] < y['savings']:
            rst = -1
        elif x['savings'] > y['savings']:
            rst = 1
        else:
            rst = 0
        return rst

    def generate_dict(self):
        return {'contracted_power': 3 * [0],
                 'price': 3 * [0],
                 'invoice_days': [],
                 'maximetro_p1': [],
                 'maximetro_p2': [],
                 'maximetro_p3': []
                }

    def obtencio_concret_value(self, value_type, period_value, powers_info, type_value_list):
        value = 0
        if value_type in powers_info:
```

```
            for key, period in powers_info[value_type].items():
                if key == period_value and type_value_list in period and
period[type_value_list] != None:
                    value = period[type_value_list]
                    if type_value_list == 'list':
                        if len(value) != 0:
                            value = sum(value) / len(value)
                        else:
                            value = 0
        return value

    def modifie_vector_ocupation(self, data_ini, data_fi):
        self.totalDaysOcuped -= ((data_fi - data_ini).days + 1)
        hours_ini = data_ini.hour
        hours_fi = data_fi.hour
        data_ini -= timedelta(hours=hours_ini)
        data_fi -= timedelta(hours=hours_fi)
        self.vector_days.append({'init': data_ini, 'fi': data_fi})

    def get_location_invoice(self):
        try:
            dev_id = self.get_main_location()
            return self.dx25.get_simulated_bill(int(dev_id), self.data_init, self.data_fi,
parameters="ASSNNA", time='HOUR')
        except:
            self.logger.exception('grr')
            return []

    def obtencion_datos_por_facturas(self):
        dict_info_invoice = self.generate_dict()
        maximetros = ['maximetro_p1', 'maximetro_p2', 'maximetro_p3']
        wrapper = wrapper_bill(self.logger)
        try:
            suplies = self.dx.get_location_supplies(int(self.loc_id))
            self.logger.info(suplies)
            for suply in suplies:
                invoices = self.dx25.get_supply_bills(int(suply['id']), self.data_init,
self.data_fi, parameters="ASSSNA", time='HOUR')
                for invo in invoices:
                    try:
                        invoze_last = wrapper.wrapper_bill_general(invo, 'SSRN',
'ELECTRICAL')
                        data_ini = datetime.fromtimestamp(invo['info']['from'] / 1e3)
                        data_fi = datetime.fromtimestamp(invo['info']['to'] / 1e3)
                        self.modifie_vector_ocupation(data_ini, data_fi)
                        dict_info_invoice['invoice_days'].append((data_fi - data_ini).days +
1)
                        if 'contracted_load' in invoze_last:
                            for i in range(3):
                                dict_info_invoice['contracted_power'][i] +=
self.obtencio_concret_value('blocks', self.periodos[i], invoze_last['contracted_load'],
'cload') / len(invoices)
                            for i in range(3):
```

```
                                dict_info_invoice['price'][i] =
self.obtencio_concret_value('blocks', self.periodos[i], invoze_last['contracted_load'],
'price')
                        if 'excess_load' in invoze_last:
                            for i in range(3):

dict_info_invoice[maximetros[i]].append(self.obtencio_concret_value('generic',
self.periodos[i], invoze_last['excess_load']['raw'], 'max'))
                    except:
                        self.logger.exception('problem with supplie')
        except:
            self.logger.exception('problem with supplies')
        try:
            invoze = self.get_location_invoice()
            invoze_last = wrapper.wrapper_bill_general(invoze[-1], 'SSNN', 'ELECTRICAL')
            #data_fi = datetime.fromtimestamp(invoze['toDate'] / 1e3)
            #data_ini = datetime.fromtimestamp(invoze['fromDate'] / 1e3)
            for _, invo in invoze_last['contracted_load']['blocks'].items():
                    i = int(invo['key'][1]) - 1
                    if dict_info_invoice['price'][i] == 0:
                        dict_info_invoice['price'][i] = invo["price"]
                        dict_info_invoice['contracted_power'][i] = invo['cload']
            type_fact = invoze[-1]['consumption']["tariff_structure"][:3]
        except:
            type_fact = '3.0'
            self.logger.exception('invoze')
        print dict_info_invoice
        return dict_info_invoice, type_fact

    def get_main_location(self):
        try:
            locat = self.dx.get_location(int(self.loc_id))
            for dev in locat['reference_devices_list']:
                if dev['type'] == 'MAINSUPPLY':
                    return dev['id']
        except:
            self.logger.exception("problem with main")
            return None

    def get_cost(self, main):
        try:
            return self.dx.get_cost(main, self.data_init, self.data_fi)
        except:
            self.logger.exception("problem with cost")
            return [], []

    def get_readings(self, main):
        try:
            self.logger.debug(main)
            self.logger.debug(self.data_init)
            self.logger.debug(self.data_fi)
            return self.dx.get_readings(main, 40205, self.data_init, self.data_fi)
        except:
```

```
            self.logger.exception("problem with readings")
            return []

    def dictionary_periods_id(self, periods):
        try:
            dict_periods = {}
            for period in periods:
                dict_periods.update({period['id']: period['name'][:2]})
            return dict_periods
        except:
            return {}

    def add_timestamp_with_period(self, read, periods, values, readsT):
        if 'period' and 'ts' in read:
                moment = read['ts']
                finish = moment + timedelta(hours=1)
                hoursT = read['ts'].hour
                while moment < finish:
                    if read['ts'] - timedelta(hours=hoursT) not in values:
                        if moment in readsT:
                            values.update({read['ts'] - timedelta(hours=hoursT): [{'period':
periods[read['period']], 'value': readsT[moment]}]})
                    else:
                        if moment in readsT:
                            values[read['ts'] - timedelta(hours=hoursT)].append({'period':
periods[read['period']], 'value': readsT[moment]})
                    moment += timedelta(minutes=15)
        return values

    def found_periods_timestamp(self, periods, reads, readsT):
        periods = self.dictionary_periods_id(periods)
        values_period = {}
        for read in reads:
            values_period = self.add_timestamp_with_period(read, periods, values_period,
readsT)
        return values_period

    def obtencion_days_without_data(self):
        vector_dias_libres = []
        day_watched = self.data_init
        while day_watched < self.data_fi:
            founded = False
            i = 0
            for reads in self.vector_days:
                if day_watched >= reads['init'] and day_watched <= reads['fi']:
                    days_before_init = (day_watched - reads['init']).days
                    if days_before_init > 1:
                        vector_dias_libres.append({'init': day_watched, 'fi': day_watched +
timedelta(days=days_before_init)})
                    days_after_end = (day_watched + relativedelta(months=1) -
reads['fi']).days
                    if days_after_end > 1:
                        if i + 1 < len(self.vector_days) and self.vector_days[i + 1]['init']
```

```
>= day_watched + relativedelta(months=1):
                            vector_dias_libres.append({'init': day_watched +
relativedelta(months=1) - timedelta(days=days_after_end), 'fi': day_watched +
relativedelta(months=1)})
                    founded = True
                    i += 1
            if not founded:
                vector_dias_libres.append({'init': day_watched, 'fi': day_watched +
relativedelta(months=1)})
            day_watched += relativedelta(months=1)
        return vector_dias_libres

    def obtencio_maximetres_per_periodes(self, dict_invoice, period_timestamps,
days_without_data):
        for days in days_without_data:
            ini = days['init']
            dict_invoice['maximetro_p1'].append(0)
            dict_invoice['maximetro_p2'].append(0)
            dict_invoice['maximetro_p3'].append(0)
            dict_invoice['invoice_days'].append((days['fi'] - ini).days)
            while ini <= days['fi']:
                if ini in period_timestamps:
                    for array in period_timestamps[ini]:
                        period = array['period'][1]
                        if dict_invoice['maximetro_p' + period][-1] < array['value']:
                            dict_invoice['maximetro_p' + period][-1] = array['value']
                ini += timedelta(days=1)
        return dict_invoice

    def change_readsT(self, readsT):
        reads = {}
        for read in readsT:
            reads.update({read['tsutc']: read['value']})
        return reads

    def obtencion_por_maximetros(self, dict_invoice):
        days_without_data = self.obtencion_days_without_data()
        if len(days_without_data) > 0:
            main_device = self.get_main_location()
            reads, periods = self.get_cost(main_device)
            readsT = self.get_readings(main_device)
            readsT = self.change_readsT(readsT)
            period_timestamps = self.found_periods_timestamp(periods, reads, readsT)
            return self.obtencio_maximetres_per_periodes(dict_invoice, period_timestamps,
days_without_data)
        return dict_invoice

    def cuerpo_triple_bucle(self, dict_info_invoice, iterations, iteracio, r, s, t, IVA, IEE,
total_Cost):
        try:
            steps = [r, s, t]
            b_costPowerTotal = self.calculatePowerCost(steps, dict_info_invoice['price'])
            b_costPenaltyTotal = self.calculatePenaltyCost(dict_info_invoice, steps)
```

```
            b_totalCost = (1 + IVA + IEE) * (b_costPowerTotal + b_costPenaltyTotal)
            savings = total_Cost - b_totalCost
            if savings > iterations['savings']:
                iterations = {'iteration': iteracio, 'savings': savings, "P1": r, "P2": s,
"P3": t}
        except:
            pass
        return iterations, iteracio + 1

    def bucle_primer_periodo(self, Pavg, stepkW, dict_info_invoice, iterations, iteracio,
IVA, IEE, total_Cost, type_fact, r=0, s=0):
        t = 0.5 * Pavg
        if type_fact[2] == '1' and Pavg * 2 > 450:
            pavgthree = 450
        else:
            pavgthree = Pavg * 2
        while t <= pavgthree:
            if t >= s:
                iterations, iteracio = self.cuerpo_triple_bucle(dict_info_invoice,
iterations, iteracio, r, s, t, IVA, IEE, total_Cost)
            t = t + stepkW
        return iterations, iteracio

    def bucle_dos_periodos(self, Pavg, stepkW, dict_info_invoice, iterations, iteracio, IVA,
IEE, total_Cost, type_fact, r=0):
        s = 0.5 * Pavg
        while s <= Pavg * 2:
            if s >= r:
                if not self.optLP:
                    iterations, iteracio = self.bucle_primer_periodo(Pavg, stepkW,
dict_info_invoice, iterations, iteracio, IVA, IEE, total_Cost, type_fact, r, s)
                elif dict_info_invoice['contracted_power'][2] >= s:
                    iterations, iteracio = self.cuerpo_triple_bucle(dict_info_invoice,
iterations, iteracio, r, s, dict_info_invoice['contracted_power'][2], IVA, IEE, total_Cost)
            s = s + stepkW
        return iterations, iteracio

    def bucle_tres_periodos(self, Pavg, dict_info_invoice, stepkW, iteracio, IVA, IEE,
total_Cost, type_fact):
        if type_fact[2] == '0':
            iterations = {'iteration': 1, 'savings': 0, "P1": 15, "P2": 15, "P3": 15}
        else:
            iterations = {'iteration': 1, 'savings': 0, "P1": 0, "P2": 0, "P3": 0}
        if Pavg != 0:
            r = 0.5 * Pavg
            while r <= Pavg * 2:
                iterations, iteracio = self.bucle_dos_periodos(Pavg, stepkW,
dict_info_invoice, iterations, iteracio, IVA, IEE, total_Cost, type_fact, r)
                r = r + stepkW
        return iterations, iteracio

    def calculation_average_and_steps(self, dict_info_invoice):
        maximetros = ['maximetro_p1', 'maximetro_p2', 'maximetro_p3']
```

```
        PnAVG = 3 * [0]
        maxs = {'maximetro_p1': [], 'maximetro_p2': [], 'maximetro_p3': []}
        for i in range(3):
            for j in range(len(dict_info_invoice[maximetros[i]])):
                if dict_info_invoice[maximetros[i]][j] > 0:
                    maxs[maximetros[i]].append(dict_info_invoice[maximetros[i]][j])

        for i in range(3):
            try:
                PnAVG[i] = sum(maxs[maximetros[i]]) / len(maxs[maximetros[i]])
            except:
                PnAVG[i] = 0

        pAvg = sum(PnAVG) / len(PnAVG)
        if pAvg == 0:
            pAvg = 1
        stepkW = 0.015 * max(PnAVG)
        if stepkW < 1:
            stepkW = 1
        return pAvg, stepkW

    def calculation_three_periods_optimization(self, dict_info_invoice, total_Cost, IVA, IEE,
type_fact):
        try:
            pAvg, stepkW = self.calculation_average_and_steps(dict_info_invoice)
            if pAvg != 0:
                iterations, _ = self.bucle_tres_periodos(pAvg, dict_info_invoice,
int(stepkW), 0, IVA, IEE, total_Cost, type_fact)
                return [iterations['P1'], iterations['P2'], iterations['P3'],
iterations['savings']]
            else:
                return [int(dict_info_invoice['price'][i]) for i in range(3)] + [0]
        except:
            return [int(dict_info_invoice['price'][i]) for i in range(3)] + [0]

    def calculatePowerCost(self, power, price):
        annualCostPower = 0
        for i in range(3):
            annualCostPower += self.annualpowerCost(power[i], price[i])
        return annualCostPower

    def calculatePenaltyCost(self, dict_info_invoice, power):
        annualCostPenalty = 0
        maximetros = ['maximetro_p1', 'maximetro_p2', 'maximetro_p3']
        for i in range(3):
            annualCostPenalty += self.annualCostPenalty(power[i],
dict_info_invoice[maximetros[i]], dict_info_invoice['price'][i],
dict_info_invoice['invoice_days'])
        return annualCostPenalty

    def format_arrays_to_string(self, array, symbol):
        for i in range(len(array)):
            try:
```

```
                array[i] = formatNum(array[i], places=0, dp='') + symbol
            except:
                pass
        return array

    def div_percentatge(self, savings, totalCost):
        try:
            return formatNum(savings / totalCost * 100) + u'%'
        except:
            return u'0 %'

    def obtain_iva_or_iee(self):
        IEE = 0
        IVA = 0
        try:
            if self.optIVA or self.optIEE:
                invoice = self.get_location_invoice()
                totals = invoice[0]['total_and_taxes']['global_cost']['taxes']
                if self.optIEE:
                    for tax in totals:
                        if tax['type'] == 'TAX':
                            IEE = tax['val'] / 100
                if self.optIVA:
                    for tax in totals:
                        if tax['type'] == 'VAT':
                            IVA = tax['val'] / 100
        except:
            self.logger.exception('problem with taxes')
        print IEE, IVA
        return IEE, IVA

    def totalCostCalculation(self, dict_info_invoice, IVA, IEE):
        try:
            costPowerTotal = self.calculatePowerCost(dict_info_invoice['contracted_power'],
dict_info_invoice['price'])
            costPenaltyTotal = self.calculatePenaltyCost(dict_info_invoice,
dict_info_invoice['contracted_power'])
            return (1 + IVA + IEE) * (costPowerTotal + costPenaltyTotal)
        except:
            return 0

    def results_obtantion(self, dict_info_invoice, best_iteration, totalCost, IVA, IEE,
savings, text):
        b_totalCost = totalCost - savings
        try:
            days = (self.data_fi - self.data_init).days
            totalCost = 365 * totalCost / days
            b_totalCost = 365 * b_totalCost / days
            self.logger.info("obtained information correctly")
            self.logger.info(best_iteration)
            return {'power_opt': self.format_arrays_to_string(best_iteration, ' kW'),
                    'contracted_power':
self.format_arrays_to_string(dict_info_invoice['contracted_power'], ' kW'),
```

```
                    'total_cost': formatNum(totalCost),
                    'optimized_cost': formatNum(b_totalCost),
                    'saving': formatNum(savings),
                    'percentatge': self.div_percentatge(savings, totalCost),
                    'periods': self.periodos,
                    'price': dict_info_invoice['price']}
                    #'loggeo': unicode(text)}
        except:
            self.logger.exception('problem getting some info')
            return {'power_opt':
self.format_arrays_to_string(dict_info_invoice['contracted_power'], ' kW'),
                    'contracted_power':
self.format_arrays_to_string(dict_info_invoice['contracted_power'], ' kW'),
                    'total_cost': formatNum(totalCost),
                    'optimized_cost': '-',
                    'saving': '-',
                    'percentatge': '-',
                    'periods': self.periodos,
                    'price': dict_info_invoice['price']}
                    #'loggeo': unicode(text)}

    def period_optimization_3_main(self):
        dict_info_invoice, type_fact = self.obtencion_datos_por_facturas()
        dict_info_invoice = self.obtencion_por_maximetros(dict_info_invoice)
        IEE, IVA = self.obtain_iva_or_iee()
        self.logger.info(IEE)
        days_bad = 0
        text = ''
        for i in range(len(dict_info_invoice['maximetro_p1'])):
            if dict_info_invoice['maximetro_p1'][i] == 0 and
dict_info_invoice['maximetro_p2'][i] == 0 and dict_info_invoice['maximetro_p3'][i] == 0:
                days_bad += dict_info_invoice['invoice_days'][i]
        if days_bad != 0:
            text = 'Atencion hay ' + str(days_bad) + ' dias en el periodo seleccionado sin
datos'
        totalCost = self.totalCostCalculation(dict_info_invoice, IVA, IEE)
        if dict_info_invoice != None:
            best_iteration = self.calculation_three_periods_optimization(dict_info_invoice,
totalCost, IVA, IEE, type_fact)
            return self.results_obtantion(dict_info_invoice, best_iteration[:3], totalCost,
IVA, IEE, best_iteration[3], text)

    def period_optimization_3_main_with_concret_values(self, best_iteration):
        self.logger.info("calculating cost for personalized iteration with three periods")
        dict_info_invoice, _ = self.obtencion_datos_por_facturas()
        dict_info_invoice = self.obtencion_por_maximetros(dict_info_invoice)
        IEE, IVA = self.obtain_iva_or_iee()
        self.logger.info("obtained dictionary info invoice")
        days_bad = 0
        text = ''
        for i in range(len(dict_info_invoice['maximetro_p1'])):
            if dict_info_invoice['maximetro_p1'][i] == 0 and
dict_info_invoice['maximetro_p2'][i] == 0 and dict_info_invoice['maximetro_p3'][i] == 0:
```

```
                days_bad += dict_info_invoice['invoice_days'][i]
        self.logger.info("total of bad days: " + str(days_bad))
        if days_bad != 0:
            text = 'Atención hay ' + str(days_bad) + 'en el periodo seleccionado sin datos'
        totalCost = self.totalCostCalculation(dict_info_invoice, IVA, IEE)
        b_costPowerTotal = self.calculatePowerCost(best_iteration,
dict_info_invoice['price'])
        b_costPenaltyTotal = self.calculatePenaltyCost(dict_info_invoice, best_iteration)
        b_totalCost = (1 + IVA + IEE) * (b_costPowerTotal + b_costPenaltyTotal)
        savings = totalCost - b_totalCost
        self.logger.info("savings: " + str(savings))
        return self.results_obtantion(dict_info_invoice, best_iteration, totalCost, IVA, IEE,
savings, text)

if __name__ == "__main__":
    logger = configLoggerError()
    logger.info("Starting bridge in bottle mode")
    data_ini = date_object = datetime.strptime('12 18 2012', '%m %d %Y')
    data_fi = date_object = datetime.strptime('12 18 2013', '%m %d %Y')
    optim = algorithm_three_periods(14317, data_ini, data_fi, False, True, True, logger)
    #optim = period_optimization_3(260, data_ini, data_fi, False, True, True, logger)
    best_iteration = [126, 126, 126]
    #result = optim.period_optimization_3_main_with_concret_values(best_iteration)
    result = optim.period_optimization_3_main()
    print result
```

C.2. Algoritme de 6 períodes – Penalització quarthorària segons potència contractada

```
__author__ = 'dutges'

# coding=utf-8

from flask import current_app
from math import sqrt

from dateutil.relativedelta import relativedelta

from datetime import datetime, timedelta
from dexma.dexcell import DexcellRestApi as DRapi

from ..utils.formatter import Formatter
from data_utils.simulatedBill.wrapper_bill import wrapper_bill

formatNum = Formatter.formatNum
```

```
class algorithm_six_periods(object):

    def __init__(self, loc_id, data_init, data_fi, optLP, optIVA, optIEE):
        self.loc_id = loc_id
        self.data_init = data_init
        self.data_fi = data_fi
        self.optLP = optLP
        self.optIVA = optIVA
        self.optIEE = optIEE
        self.logger = current_app.logger
        self.dx = DRapi("http://www.dexcell.com/api/v2", "1c9004c6632e1bbbf2bd")
        self.dx25 = DRapi("http://www.dexcell.com/api/v25", "1c9004c6632e1bbbf2bd")
        self.totalDaysOcuped = (data_fi - data_init).days
        self.FactorbyPeriod = {'P1': 1,
                               'P2': 0.5,
                               'P3': 0.37,
                               'P4': 0.37,
                               'P5': 0.37,
                               'P6': 0.17,
                               }
        self.festivos = [[1, 1], [1, 5], [15, 8], [12, 10], [1, 11], [6, 12], [25, 12]]

#####################################
############  data-drapi #############
#####################################
    def get_main_location(self):
        try:
            locat = self.dx.get_location(int(self.loc_id))
            for dev in locat['reference_devices_list']:
                if dev['type'] == 'MAINSUPPLY':
                    return dev['id']
        except:
            self.logger.exception('problem obtaining device id!!!')
            return None

    def get_cost(self, main):
        try:
            return self.dx.get_cost(main, self.data_init, self.data_fi)
        except:
            return [], []

    def get_readings(self, main):
        try:
            return self.dx.get_readings(main, 40205, self.data_init, self.data_fi)
        except:
            return []

    def get_location_invoice(self, data_ini):
        try:
            dev_id = self.get_main_location()
            return self.dx25.get_simulated_bill(dev_id, data_ini, data_ini +
```

```
timedelta(days=7), parameters="NNSNNN", pod=None, time='HOUR')
        except:
            self.logger.exception('problem obtaining invoice!!!')
            return []

    def dictionary_periods_id(self, periods):
        try:
            dict_periods = {}
            for period in periods:
                dict_periods.update({period['id']: period['name'][:2]})
            return dict_periods
        except:
            return {}

    def compare_with_party_days(self, dayT, periods, read):
        for party in self.festivos:
            if dayT.day == party[0] and dayT.month == party[1]:
                return 'P6'
        return periods[read['period']]

    def add_timestamp_with_period(self, read, periods, values):
        if 'period' and 'ts' in read:
                values.update({read['ts']: self.compare_with_party_days(read['ts'], periods,
read)})
        return values

    def found_periods_timestamp(self, periods, reads):
        periods = self.dictionary_periods_id(periods)
        values_period = {}
        for read in reads:
            values_period = self.add_timestamp_with_period(read, periods, values_period)
        return values_period

    def concatenate_readings_and_cost(self, period_timestamps, main_device):
        dict_periods = {'P1': [], 'P2': [], 'P3': [], 'P4': [], 'P5': [], 'P6': []}
        readings = self.get_readings(main_device)
        for read in readings:
            hour = read['ts'] - timedelta(minutes=read['ts'].minute)
            if hour in period_timestamps:
                dict_periods[period_timestamps[hour]].append({'value': read['value'], 'ts':
read['ts']})
        return dict_periods

    def compara2(self, x, y):
        if x['ts'] < y['ts']:
            rst = -1
        elif x['ts'] > y['ts']:
            rst = 1
        else:
            rst = 0
        return rst

    def sort_dict_by_timestamps(self, dict_with_values):
```

```
        for period in dict_with_values:
            dict_with_values[period].sort(self.compara2)
        return dict_with_values

    def obtencion_por_maximetros(self):
        main_device = self.get_main_location()
        reads, periods = self.get_cost(main_device)
        period_timestamps = self.found_periods_timestamp(periods, reads)
        dict_with_values = self.concatenate_readings_and_cost(period_timestamps, main_device)
        return self.sort_dict_by_timestamps(dict_with_values)

    def obtain_contracteds_powers(self, trobat):
        contracted_powers = {}
        price_powers = {}
        this_date = self.data_fi - timedelta(days=7)
        while not trobat and this_date > self.data_init:
            invoice = self.get_location_invoice(this_date)
            wrapper = wrapper_bill(self.logger)
            try:
                invoice = wrapper.wrapper_bill_general(invoice[0], 'NSNN', 'ELECTRICAL')
                self.logger.info(invoice)
                for _, invo in invoice['contracted_load']['blocks'].items():
                    contracted_powers.update({invo["key"]: invo["cload"]})
                    price_powers.update({invo["key"]: invo["price"]})
                if len(contracted_powers) != 0 and len(price_powers) != 0:
                    trobat = True
            except Exception as e:
                self.logger.warning('problem with bill error: %s' % e)
            this_date -= timedelta(days=7)
        return contracted_powers, price_powers

    def obtencion_last_excesos(self, dict_info_invoice, contracted_power):
        excesos_periods = {'P1': 0, 'P2': 0, 'P3': 0, 'P4': 0, 'P5': 0, 'P6': 0}
        months = (self.data_fi.year - self.data_init.year) * 12 + self.data_fi.month -
self.data_init.month
        for period in dict_info_invoice:
            periods = (months + 1) * [0]
            k = 0
            i = 0
            first_month = self.data_init - timedelta(self.data_init.day - 1)
            while first_month < self.data_fi:
                while i < len(dict_info_invoice[period]):
                    value = dict_info_invoice[period][i]
                    try:
                        if value['ts'] < first_month + relativedelta(months=1):
                            if value['value'] > contracted_power[period]:
                                periods[k] += (value['value'] - contracted_power[period]) **
2
                        else:
                            break
                    except:
                        pass
                    i += 1
```

```
                k += 1
                first_month += relativedelta(months=1)
            for per in periods:
                excesos_periods[period] += sqrt(per) * self.FactorbyPeriod[period] * 1.4064
        total = 0
        for period in excesos_periods:
            total += excesos_periods[period]
        return excesos_periods, total

    def obtencion_excesos(self, dict_info_invoice, contracted_power):
        excesos_periods = {'P1': 0, 'P2': 0, 'P3': 0, 'P4': 0, 'P5': 0, 'P6': 0}
        for period in dict_info_invoice:
            for value in dict_info_invoice[period]:
                try:
                    if value['value'] > contracted_power[period]:
                        excesos_periods[period] += (value['value'] -
contracted_power[period]) ** 2
                    else:
                        break
                except:
                    pass

            excesos_periods[period] = sqrt(excesos_periods[period]) *
self.FactorbyPeriod[period] * 1.4064
        total = 0
        for period in excesos_periods:
            total += excesos_periods[period]
        return excesos_periods, total

    def costePotenciaContratada(self, contracted_powers, contracted_prices):
        COSTAPTotalValue = 0.0
        for dd in contracted_powers:
            try:
                COSTAPTotalValue += contracted_powers[dd] * contracted_prices[dd] *
self.totalDaysOcuped
            except:
                pass
        return COSTAPTotalValue

    def suma_excesos_cost(self, exces, cost):
        for exc in exces:
            cost += exces[exc]
        return cost

    def neighbour(self, state, jump):
        periods = ['P1', 'P2', 'P3', 'P4', 'P5']
        for period in periods:
            state[period] += jump
        return state

    def calculate_actual_last_cost(self, dict_info_invoice, contracted_powers,
contracted_prices):
        excess_info, total_exces = self.obtencion_last_excesos(dict_info_invoice,
```

```
contracted_powers)
        cost_without_excess = self.costePotenciaContratada(contracted_powers,
contracted_prices)
        return self.suma_excesos_cost(excess_info, cost_without_excess), total_exces,
excess_info

    def calculate_actual_cost(self, dict_info_invoice, contracted_powers, contracted_prices):
        excess_info, total_exces = self.obtencion_excesos(dict_info_invoice,
contracted_powers)
        cost_without_excess = self.costePotenciaContratada(contracted_powers,
contracted_prices)
        return self.suma_excesos_cost(excess_info, cost_without_excess), total_exces,
excess_info

    def calculation_optimization_simulated_annealing(self, dict_info_invoice,
contracted_powers, contracted_prices):
        self.logger.debug(contracted_powers)
        state = {'P1': 1,
                 'P2': 1,
                 'P3': 1,
                 'P4': 1,
                 'P5': 1,
                 'P6': contracted_powers['P6']}
        jump = int(0.005 * contracted_powers['P1'])
        if jump < 1:
            jump = 1
        best_state = state.copy()
        Eact, _, _ = self.calculate_actual_last_cost(dict_info_invoice, state,
contracted_prices)
        Ebest = Eact
        while state['P1'] < (1.5 * contracted_powers['P1']):
            state = self.neighbour(state, jump)
            Eact, total_exces, _ = self.calculate_actual_last_cost(dict_info_invoice, state,
contracted_prices)
            if Eact < Ebest:
                Ebest = Eact
                best_state = state.copy()
            if total_exces == 0:
                break
        return best_state, Ebest

    def neighbour2(self, state, max_state, step):
        periods = ['P3', 'P2', 'P1']
        mins = [state['P2'] + step, state['P1'] + step, 0]
        for i in range(len(periods)):
            if state[periods[i]] >= max_state:
                state[periods[i]] = mins[i]
                if periods[i] == 'P3':
                    state['P4'] = mins[i]
                    state['P5'] = mins[i]
            elif state[periods[i]] < max_state:
                state[periods[i]] += step
                if periods[i] == 'P3':
```

```
                    state['P4'] += step
                    state['P5'] += step
                return state
        return state

    def calculation_second_optimization_first_period_fixed(self, dict_info_invoice,
best_state, contracted_prices, Ebest):
        max_state = int(best_state['P1'] * 1.05)
        step = int(best_state['P1'] * 0.01)
        if step < 1:
            step = 1
        state = {'P1': int(best_state['P1'] * 0.95),
                 'P2': int(best_state['P2'] * 0.95),
                 'P3': int(best_state['P3'] * 0.95),
                 'P4': int(best_state['P4'] * 0.95),
                 'P5': int(best_state['P5'] * 0.95),
                 'P6': best_state['P6']}
        while state['P1'] < max_state:
            state = self.neighbour2(state, max_state, step)
            Eact, total_exces, _ = self.calculate_actual_cost(dict_info_invoice, state,
contracted_prices)
            if Eact < Ebest:
                Ebest = Eact
                best_state = state.copy()
            if total_exces == 0:
                break
        return best_state, Ebest

    def calculation_optimization_six_period(self, dict_info_invoice, best_state,
contracted_prices, Ebest, contracted_powers):
        state = best_state.copy()
        step = int(best_state['P1'] * 0.01)
        if step < 1:
            step = 1
        if state['P5'] > 450:
            state['P6'] = state['P5']
            best_state['P6'] = state['P5']
        else:
            state['P6'] = 450
        while state['P6'] < contracted_powers['P6'] * 1.5:
            state['P6'] += step
            Eact, total_exces, exces_list = self.calculate_actual_cost(dict_info_invoice,
state, contracted_prices)
            if Eact < Ebest:
                Ebest = Eact
                best_state = state.copy()
            if total_exces == 0 or exces_list['P6'] == 0:
                break
        return best_state, Ebest

#####################################
############  data-main ##############
#####################################
```

```
    def format_arrays_to_string(self, array, symbol):
        new_array = 6 * ['0 kw']
        try:
            for i in array:
                try:
                    numP = i[1:]
                    new_array[int(numP) - 1] = formatNum(array[i], places=0, dp='') + symbol
                except:
                    pass
        except:
            pass
        return new_array

    def format_basic_num(self, IVA, IEE, num):
        try:
            if num < 1000:
                return formatNum((1 + IVA + IEE) * num)
            else:
                return formatNum((1 + IVA + IEE) * num, places=0, dp='')
        except:
            return '-'

    def div_percentatge(self, savings, totalCost):
        try:
            return formatNum(savings / totalCost * 100) + u'%'
        except:
            return u'0 %'

    def obtain_iva_or_iee(self):
        IEE = 0
        IVA = 0
        try:
            self.logger.info('obtaining IVA and IEE')
            if self.optIVA or self.optIEE:
                invoice = self.get_location_invoice()
                totals = invoice[0]['total_and_taxes']['global_cost']['taxes']
                if self.optIEE:
                    for tax in totals:
                        if tax['type'] == 'TAX':
                            IEE = tax['val'] / 100
                if self.optIVA:
                    for tax in totals:
                        if tax['type'] == 'VAT':
                            IVA = tax['val'] / 100
        except:
            self.logger.exception('problem with taxes')
        return IEE, IVA

    def period_optimization_6_main(self):
        self.logger.info('begining six period optimization')
        dict_info_invoice = self.obtencion_por_maximetros()
        contracted_powers, contracted_prices = self.obtain_contracteds_powers(False)
```

```
        if len(contracted_powers) > 0:
            IEE, IVA = self.obtain_iva_or_iee()
            self.real_cost, _, _ = self.calculate_actual_last_cost(dict_info_invoice,
contracted_powers, contracted_prices)
            best_state, Ebest =
self.calculation_optimization_simulated_annealing(dict_info_invoice, contracted_powers,
contracted_prices)
            best_state, Ebest =
self.calculation_second_optimization_first_period_fixed(dict_info_invoice, best_state,
contracted_prices, Ebest)
            if not self.optLP:
                best_state, Ebest =
self.calculation_optimization_six_period(dict_info_invoice, best_state, contracted_prices,
Ebest, contracted_powers)
                Ebest, _, _ = self.calculate_actual_last_cost(dict_info_invoice, best_state,
contracted_prices)
            days = (self.data_fi - self.data_init).days
            self.real_cost = 365 * self.real_cost / days
            Ebest = 365 * Ebest / days
            return {'power_opt': self.format_arrays_to_string(best_state, ' kW'),
                    'contracted_power': self.format_arrays_to_string(contracted_powers, '
kW'),
                    'total_cost': self.format_basic_num(IVA, IEE, self.real_cost),
                    'optimized_cost': self.format_basic_num(IVA, IEE, Ebest),
                    'saving': formatNum((1 + IVA + IEE) * (self.real_cost - Ebest)),
                    'percentatge': self.div_percentatge((1 + IVA + IEE) * (self.real_cost -
Ebest), (1 + IVA + IEE) * self.real_cost),
                    'periods': ['P1', 'P2', 'P3', 'P4', 'P5', 'P6'],
                    'price': contracted_prices}
        else:
            return {'power_opt': self.format_arrays_to_string([0, 0, 0, 0, 0, 0], ' kW'),
                    'contracted_power': self.format_arrays_to_string(contracted_powers, '
kW'),
                    'total_cost': '--',
                    'optimized_cost': '--',
                    'saving': '--',
                    'percentatge': '--',
                    'periods': ['P1', 'P2', 'P3', 'P4', 'P5', 'P6'],
                    'price': contracted_prices}

    def period_optimization_with_concret_values(self, concret_powers):
        dict_info_invoice = self.obtencion_por_maximetros()
        contracted_powers, contracted_prices = self.obtain_contracteds_powers(False)
        if len(contracted_powers) > 0:
            IEE, IVA = self.obtain_iva_or_iee()
            self.real_cost, _, _ = self.calculate_actual_last_cost(dict_info_invoice,
contracted_powers, contracted_prices)
            Eact, _, _ = self.calculate_actual_last_cost(dict_info_invoice, concret_powers,
contracted_prices)
            return {'power_opt': self.format_arrays_to_string(concret_powers, ' kW'),
                    'contracted_power': self.format_arrays_to_string(contracted_powers, '
kW'),
                    'total_cost': self.format_basic_num(IVA, IEE, self.real_cost),
```

```
                    'optimized_cost': self.format_basic_num(IVA, IEE, Eact),
                    'saving': formatNum((1 + IVA + IEE) * (self.real_cost - Eact)),
                    'percentatge': self.div_percentatge((1 + IVA + IEE) * (self.real_cost -
Eact), (1 + IVA + IEE) * self.real_cost),
                    'periods': ['P1', 'P2', 'P3', 'P4', 'P5', 'P6'],
                    'price': contracted_prices}
        else:
            return {'power_opt': self.format_arrays_to_string([0, 0, 0, 0, 0, 0], ' kW'),
                    'contracted_power': {},
                    'total_cost': '--',
                    'optimized_cost': '--',
                    'saving': '--',
                    'percentatge': '--',
                    'periods': ['P1', 'P2', 'P3', 'P4', 'P5', 'P6'],
                    'price': contracted_prices}
```