Trabalho de Prog. Não Linear

Denilson Figueiredo de Sá
Gabriel Ferreira Barros

Chute inicial: bolas alinhadas

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# vi:ts=4 sw=4 et

#############################################################################
#   Trabalho de Programação Não Linear - 2009/2
# Prof.: Luziane
#
# Denilson Figueiredo de Sá
# DRE: 103108905
#
# Gabriel Ferreira Barros
# DRE: 107362179
#
#
# Problema sendo resolvido:
#   Empacotamento de bolas
#   Dadas 180 bolas de tamanhos (raios) diferentes, encontrar uma
#   disposição delas de modo que minimize a área da caixa usada para
#   armazenar todas as bolas.
#
# Método implementado:
#   Busca coordenada (ou busca padrão usando as direções coordenadas)
#
# Detalhes de modelagem:
#   Favor consultar a documentação da função objetivo f(x), mais
#   abaixo.
#
# Resultados encontrados:
#   Sem a restrição para o valor de C2, o método não convergiu e
#   reduziu as coordenadas de C2 de maneira irrestrita, chegando a
#   coordenadas negativas e, por erro na fórmula inicial, à área
#   negativa.
#
#   A proposta é tentar novamente usando a nova fórmula apresentada
#   abaixo, a qual inclui "abs()" no cálculo da área e também inclui a
#   restrição penalidade_caixa
#
#   Uma segunda proposta é não considerar as coordenadas de C2 como
#   variáveis do problema, mas sim calculá-las dentro de f(x) através
#   da fórmula:
#     C2.x = max(bi_x + r_i)
#     C2.y = max(bi_y + r_i)
#     C2.z = max(bi_z + r_i)
#


import sys

import numpy
from numpy import array

numbolas = 180
raio = array(
#    [ raio ] * quant  # <quant> bolas de <raio> cm
    [ 20.0 ] * 100 +
    [ 50.0 ] * 50  +
    [ 70.0 ] * 30
)

direcoes_busca_coordenada = []
iteracoes = []


def f(x):
    """Função objetivo a ser minimizada, já incluindo as restrições.

    Esta função modela <numbolas> de raios diferentes a serem colocadas numa
    caixa. A caixa é definida pelos pontos C1 e C2, dois vértices opostos.
    Por modelagem, C1 está fixo em (0,0,0).

    O parâmetro x deve ser um array da seguinte forma:
    * O array deve conter números de ponto flutuante.
    * As dimensões devem ser (<numbolas> + 1, 3). Ou seja, x[i][0] é a
      coordenada X do i-ésimo elemento.
    * O primeiro elemento são as coordenadas X,Y,Z do vértice C2 da caixa.
    * Os <numbolas> elementos seguintes são as coordenadas X,Y,Z de cada
      uma das bolas.
    """

    # Nota: os resultados citados acima foram obtidos usando esta
    # fórmula e sem o cálculo da penalidade_caixa:
    # area = 2 * (
    #     x[0][0] * x[0][1] +
    #     x[0][0] * x[0][2] +
    #     x[0][1] * x[0][2]
    # )

    area = 2 * (
        abs(x[0][0] * x[0][1]) +
        abs(x[0][0] * x[0][2]) +
        abs(x[0][1] * x[0][2])
    )
```

```
    # Restrição:
    #   C2 tem que ser positivo (ou seja, cada componente de C2 tem
    #   que ser maior que cada componente de C1)
    penalidade_caixa = (
        max(0, -x[0][0]) +
        max(0, -x[0][1]) +
        max(0, -x[0][2])
    )

    # Restrição:
    #   | b_i - b_j |  >=  r_i + r_j  (para todo i,j)
    # ou seja:
    #   r_i + r_j - | b_i - b_j | <= 0
    num_colisoes = 0
    colisoes = 0.0
    for i in range(1, numbolas+1):
        for j in range(i+1, numbolas+1):
            delta = x[i]-x[j]
            dist = numpy.dot(delta, delta)  # quadrado da norma do vetor
            penalidade = raio[i-1] + raio[j-1] - dist
            if penalidade > 0.0:
                colisoes += penalidade
                num_colisoes += 1

    # Restrições:
    #   b_i - r_i  >= 0   (para todo i)
    #   b_i + r_i  <= C2  (para todo i)
    # ou seja:
    #  -b_i + r_i     <= 0
    #   b_i + r_i -C2 <= 0
    num_bordas = 0
    bordas = 0.0
    for i in range(1, numbolas+1):
        penalidade = 0.0
        for j in range(3):  # x,y,z
            penalidade += max(0.0, -x[i][j] + raio[i-1])
            penalidade += max(0.0,  x[i][j] + raio[i-1] - x[0][j])
        if penalidade > 0.0:
            bordas += penalidade
            num_bordas += 1

    total = area + penalidade_caixa + colisoes + bordas

    #return (total, num_colisoes, num_bordas)
    return total


def criar_ponto(zeros=True):
    if zeros:
        return numpy.zeros( shape=(numbolas+1, 3),  dtype=numpy.float64)
    else:
        return numpy.empty( shape=(numbolas+1, 3),  dtype=numpy.float64)


def criar_um_chute_inicial():
    """Retorna um chute inicial "x" com todas as bolas alinhadas.
    """
    x = criar_ponto(zeros=False)
    prev = 0.0
    for i, v in enumerate(x[1:]):
        v[0] = prev + raio[i]
        prev += 2*raio[i]
        v[1] = raio[i]
        v[2] = raio[i]
    x[0][0] = prev
    x[0][1] = 2*max(raio)
    x[0][2] = 2*max(raio)
    return x


def busca_padrao(x_inicial, direcoes, callback):
    iteracao = 0
    delta = 2.0
    epsilon = 10**(-6)

    x  = x_inicial
    fx = f(x)
    callback(iteracao, x, fx)

    while delta > epsilon:
        for d in direcoes:
            xnovo = x + delta*d
            fxnovo = f(xnovo)
            if fxnovo < fx:
                iteracao += 1
                x = xnovo
                fx = fxnovo
                callback(iteracao, x, fx)
                break
        else:  # Este else é em relação ao for
            delta /= 2
```

```
def criar_direcoes_busca_coordenada():
    origem = criar_ponto(zeros=True)
    dirs = []
    for i in range(origem.size):
        for d in (-1, 1):
            x = origem.copy()
            x.flat[i] = d
            dirs.append( x )

    return dirs


def print_point(x, nome="", file=sys.stdout):
    opts = numpy.get_printoptions()
    numpy.set_printoptions(suppress=True, threshold=1000000)
    if nome:
        file.write("%s = %s\n" % (nome, repr(x)))
    else:
        file.write(repr(x) + "\n")
    numpy.set_printoptions(**opts)


def main():
    global direcoes_busca_coordenada, iteracoes

    arquivo = open("points.txt", "w")
    arquivo.write("numbolas = " + str(numbolas) + "\n")
    arquivo.write("raio = " + repr(list(raio)) + "\n")

    direcoes_busca_coordenada = criar_direcoes_busca_coordenada()
    iteracoes = []

    x = criar_um_chute_inicial()

    def registrar_iteracao(itnum, x, fx):
        global iteracoes
        iteracoes.append( (fx, x) )
        print "%d - %f" % (itnum, fx)
        if arquivo:
            arquivo.write("# f(x) = %f\n" % (fx,))
            print_point(x, file=arquivo)


    busca_padrao(x, direcoes_busca_coordenada, registrar_iteracao)

    #for p in iteracoes:
    #    arquivo.write("# f(x) = %f\n" % (p[0],))
    #    print_point(p[1], file=arquivo)

    arquivo.close()


if __name__ == "__main__":
    main()
```